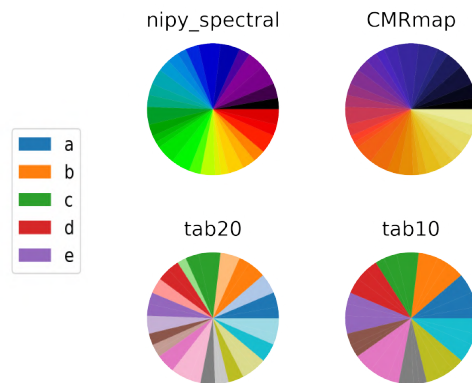
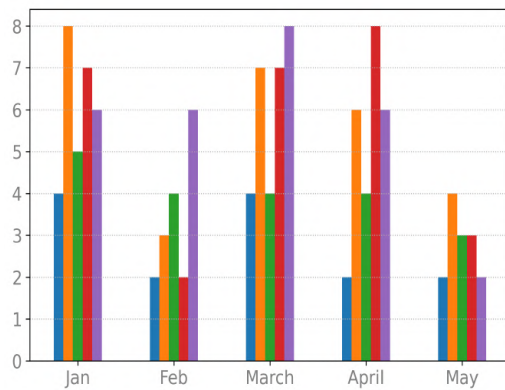
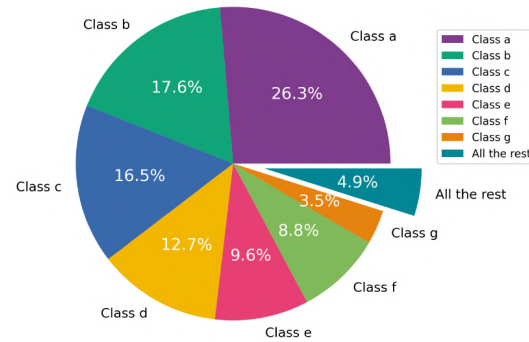
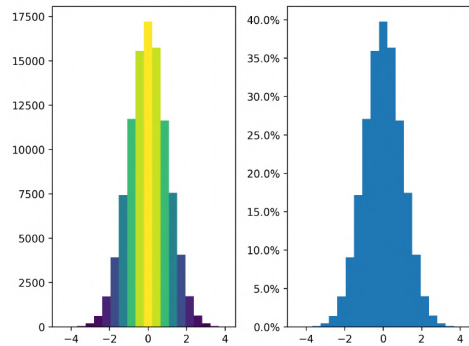
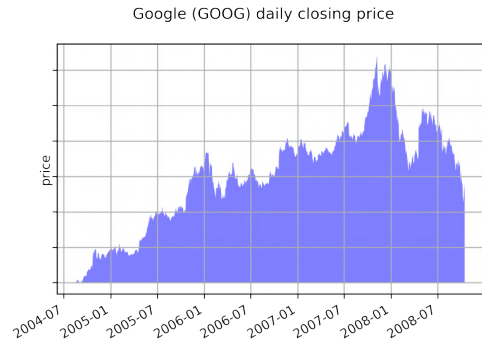




matplotlib

A COMPLETE REVIEW

- MATPLOTLIB ARCHITECTURE
- ANATOMY OF A MATPLOTLIB FIGURE
- PYPLOT INTERFACE
- OBJECT ORIENTED INTERFACE

PREPARED BY :
BHASKAR JYOTI ROY
bhaskarjroy1605@gmail.com

REFERENCES :

- Matplotlib Release 3.4.2, May 08, 2021
- Mastering Matplotlib by Duncan M. McGregor
- StackOverflow and others

INDEX – MATPLOTLIB API

- [Matplotlib Introduction](#)
- [Topics we will cover](#)
- [Matplotlib Architecture – Three layers](#)
- [Relationship of layers to Data and Figure](#)
- [Deep Dive into Backend Layer](#)
- [Deep Dive into Artist Layer](#)
- [Anatomy of a Figure](#)
- [Hierarchy of Artists](#)
- [Scripting Layer](#)
- [Pyplot Interface](#)
- [Frequently Used Pyplot Functions](#)
 - [For the Figure, Axes, Axis](#)
 - [For Charting](#)
 - [For Multiple plots, Layout Customisation](#)
 - [Code Snippet for use of subplot function](#)
 - [For fine tuning plot appearance](#)
 - [rc Params , rcParams for global changes](#)
 - [rcParams dictionary variables to revert back settings](#)
 - [StyleSheets](#)
 - [StyleSheets – Showcasing All styles](#)

INDEX – OO INTERFACE

- [OO interface](#)
- [Create a Figure instance](#)
- [Use Figures to create one or more axes or subplot](#)
 - [Demo of fig.subplots\(\) and enumerate, flatten](#)
 - [Demo of fig.add_subplots\(\) without and with GridSpec](#)
 - [Demo of fig.add_axes\(\)](#)
- [Use Axes helper methods to add artists to respective containers within the Axes object](#)
 - [Axes Helper Methods to create Primitives](#)
 - [Axes Methods to create Different Plots and text](#)
 - [Example Illustrations of various plots – Piecharts, Histograms, bar charts, Grouped bars charts](#)
 - [Example Illustrations of various plots – Line Chart, Bidirectional Chart, Marginal Histograms, Diverging Lollypop chart](#)
- [Use Axes/Axis Helper methods to access and set x-axis and y-axis tick, tick labels and axis labels](#)
 - [Axes Helper Methods for Appearance, Axis Limits](#)
 - [Axes Helper Methods for Ticks and Tick Labels, GridLines](#)
 - [Axes Helper Methods for Axis Labels, Title and legend](#)
 - [Axis Helper Methods for Formatters and Locators, Axis Label](#)
 - [Axis Helper Methods for Ticks, Tick Labels](#)
 - [Axis Helper Methods – X Axis, Y Axis Specific](#)
 - [Tick Params Method – Bulk property Setter, A Note on Practical Implementation](#)
 - [Special Note on Tickers – Locator Class, Demo 1, Demo 2](#)

INDEX – OO INTERFACE

- [Special Note on Tickers – Formatter Class](#)
- [Tickers – Using Simple Formatting, Formatter Class](#)
- [Tickers for Time Series Plotting, Date Locator and Date Formatter](#)
- [Concise date Formatter Example](#)
- [Use Axes Methods \(Accessors\) to access artists through attributes](#)
 - [Accessing Artists by Assigning to a Variable when creating](#)
 - [Accessing Artists through Attributes - Approach](#)
 - [Useful Methods and Python Built in Methods](#)
 - [Accessing Artists through Attributes \(Artist Hierarchy\)](#)
- [Set/modify the Artist properties after accessing the artists](#)
 - [Mapping the Journey – From Axes Instance to Artist Instance Properties](#)
 - [Properties Common to All Artists](#)
 - [Setting Artist Properties, Inspecting Artist Properties](#)
 - [Additional Properties specific to Line 2D , text](#)
 - [Special look into Patch Rectangle](#)
 - [Additional Properties Rectangle Specific](#)
 - [Usecases – Modifying the Rectangle patch properties](#)
 - [Methods of Rectangle Patch](#)
 - [Creating User Defined function to Stylize Axes](#)
 - [Creating User Defined function to Add data labels in bar plots](#)
 - [Demo – Using UDF to stylize Axes and add data labels](#)

INDEX – OO INTERFACE

- Transformations
 - Transformation Framework
 - Transformation Framework – The Coordinate systems
 - Transformation Framework – Transformation Objects
 - Blended Transformations Example
- Colors
 - Specifying Colors – Acceptable Formats
 - Exploring the Color Charts – Use Case Scenario
 - Base Colors : 'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'
 - Tableau Palette
 - CSS colors
 - XKCD Colors – xkcd:black – xkcd:dirty orange
 - XKCD Colors – xkcd:medium brown – xkcd:sunny yellow
 - XKCD Colors – xkcd:parchment – xkcd:leaf
 - XKCD Colors – xkcd:kermit green– xkcd:light seafoam
 - XKCD Colors – xkcd:dark mint– xkcd:ocean blue
 - XKCD Colors – xkcd:dirty blue– xkcd:ultra marine
 - XKCD Colors – xkcd:purpleish blue – xkcd:light plum
 - XKCD Colors – xkcd:pale magenta – xkcd:salmon pink

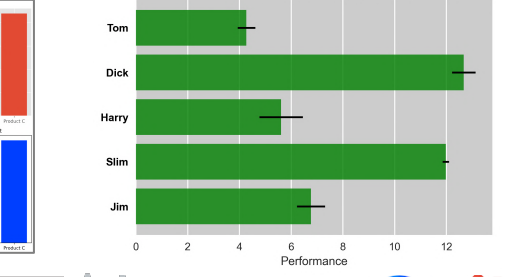
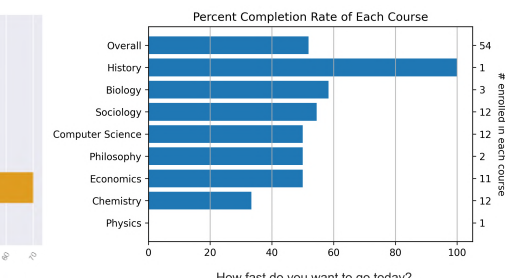
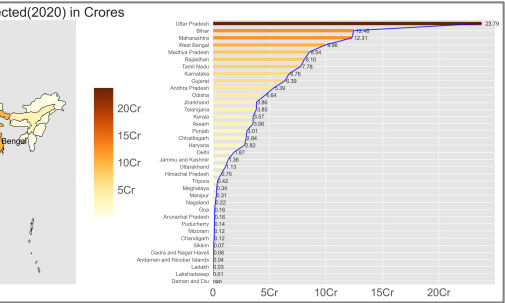
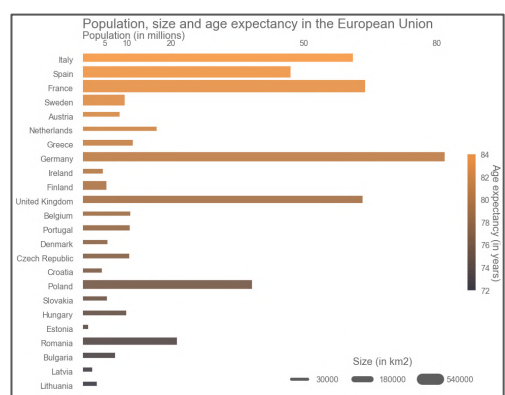
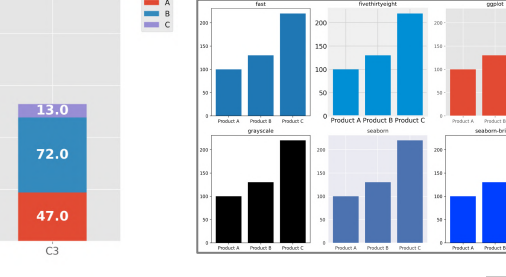
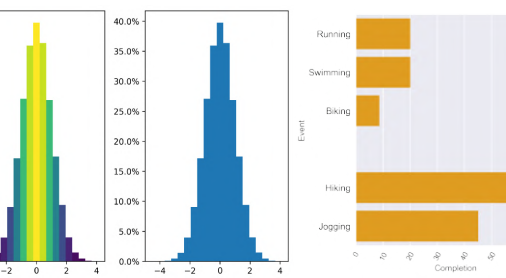
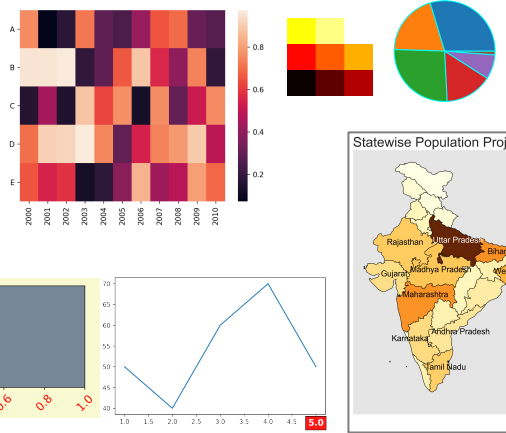
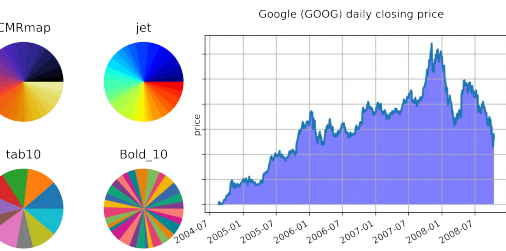
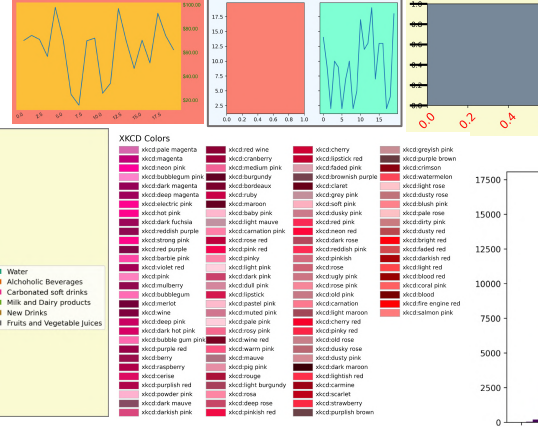
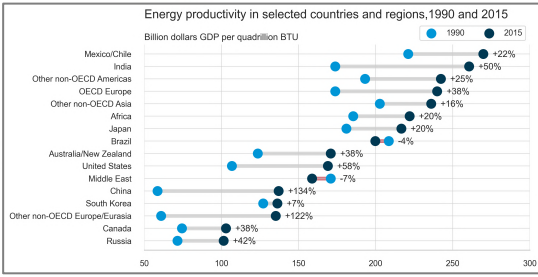
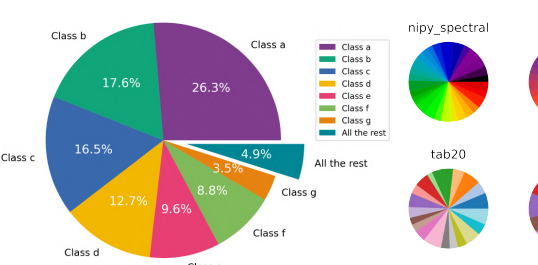
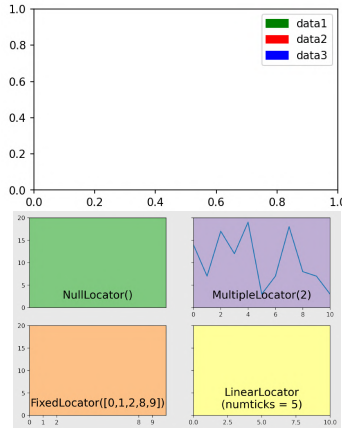
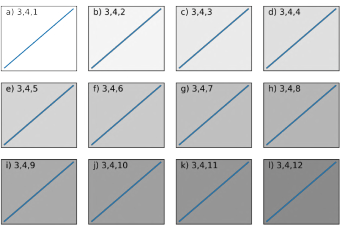
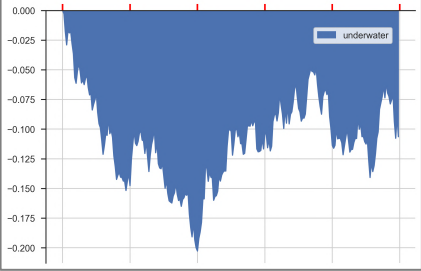
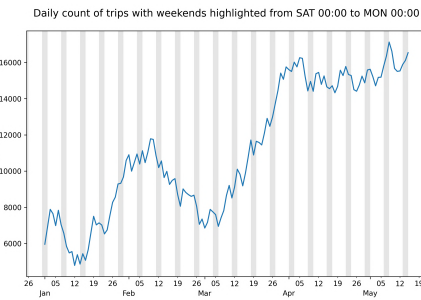
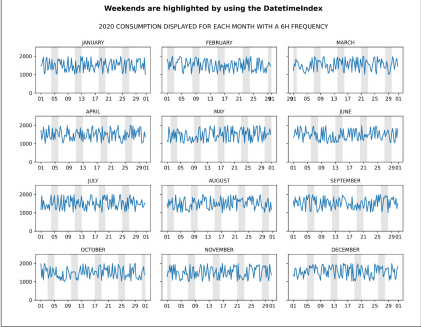
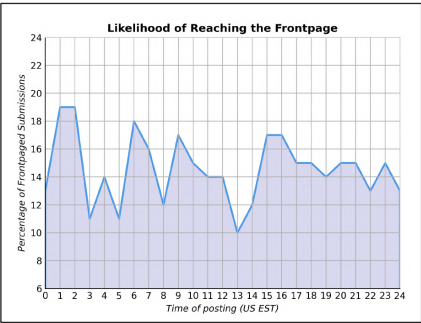
INDEX – OO INTERFACE

- ColorMaps - Overview
 - Colormaps Classification
 - Range of Matplotlib Colormaps
 - Sequential colormaps
 - Diverging colormaps
 - Cyclic colormaps
 - Qualitative colormaps
 - Miscellaneous colormaps
 - Lightness of Matplotlib colormaps
 - Accessing Colors from a Matplotlib Colormap
 - Creating Listed Colormap from Existing Matplotlib Colormaps
 - Code snippet to create listed colormaps and accessing colors
- Add legend and set title to the axes
 - Aspects of Legend
 - Adding Legend to Axes Call Signatures, Important terminologies
 - Automatic detection of elements to be shown in the legend
 - Explicitly defining the elements in the legend
 - With labels explicitly defined in call signature
 - With labels omitted in call signature
 - Creating Proxy Artists and Adding in the legend (Code demo)
 - Controlling Legend Location, Placement – Concept, Demo

INDEX – END

- [Epilogue](#)
- [Data Structures Overview](#)
- [Pseudo Random Number Generation \(in Python, Numpy\)](#)
- [Handling Date and Time](#)
- [Statistics with Python](#)
- [Appendix](#)
- [References](#)

VISUAL INDEX - 01



VISUAL INDEX – 02



MATPLOTLIB INTRODUCTION

Widely used plotting library for the [Python programming language](#)

A core component of the scientific Python stack – a group of scientific computing tools, alongwith Numpy, Scipy and IPython

Initial Release in 2003
Created by neurobiologist [John D. Hunter](#) to plot data of electrical activity in the brains of epilepsy patients, but today is used in number of fields

Original Design Philosophy
Matplotlib is designed with the philosophy that you should be able to create simple plots with just a few commands, or just one! If you want to see a histogram of your data, you shouldn't need to instantiate objects, call methods, set properties, and so on; it should just work.

Two ways to use Matplotlib :

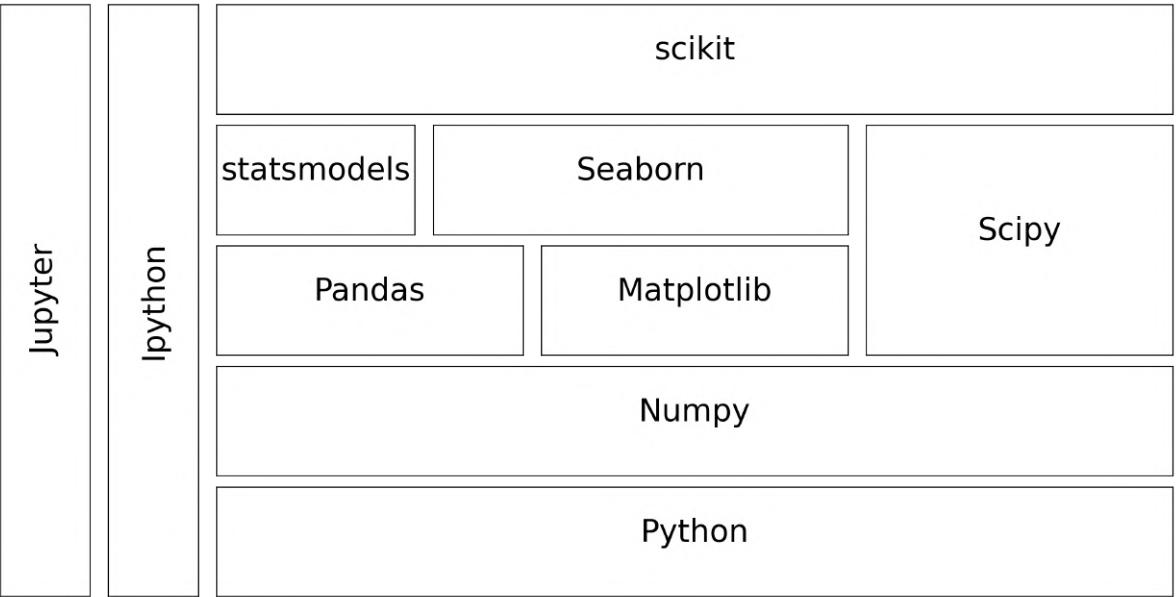
- Pyplot Interface inspired from MATLAB is fast and convenient
- Pythonic OO way is flexible and being explicit gives finely grained control

libraries like [pandas](#) and [Seaborn](#) are “wrappers” over matplotlib - providing high level API's that allow access to a number of matplotlib's methods with less code. For instance, pandas' .plot() combines multiple matplotlib methods into a single method, thus plotting in a few lines.

A comprehensive library for creating static, animated, and interactive visualizations in Python.

STRUCTURE OF THE IMPORTANT PYTHON PACKAGES FOR DATA SCIENCE

Code



- `ipython` : : : For interactive work.
- `numpy` : : : For working with vectors and arrays.
- `scipy` : : : All the essential scientific algorithms, including those for basic statistics.
- `matplotlib` : : : The de-facto standard module for plotting and visualization.
- `pandas` : : : Adds DataFrames (imagine powerful spreadsheets) to Python.
- `statsmodels` : : : For statistical modeling and advanced analysis.
- `seaborn` : : : For visualization of statistical data.
- Scikit extensions to scipy for specific fields like machine learning, , deep learning, x-ray, image processing and many more
- Additional Libraries are Plotly, Bokeh, TensorFlow, Keras, XGBoost

TOPICS WE WILL COVER

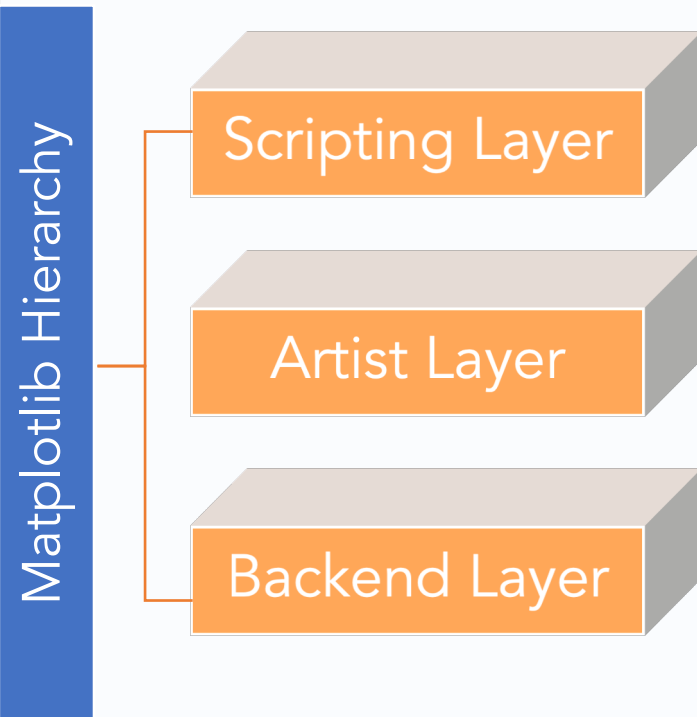


Three Layers of matplotlib architecture	• Backend layer • Artist layer • Scripting layer
Anatomy of a figure	
Artist Hierarchy	
Pyplot API	
Object oriented API - for control and Customisation	

- Overview of Data Structures
- Pseudo Random Number Generation
- Handling Date and Time
- Statistics with Python
- Appendix – Matplotlib plot examples
- References

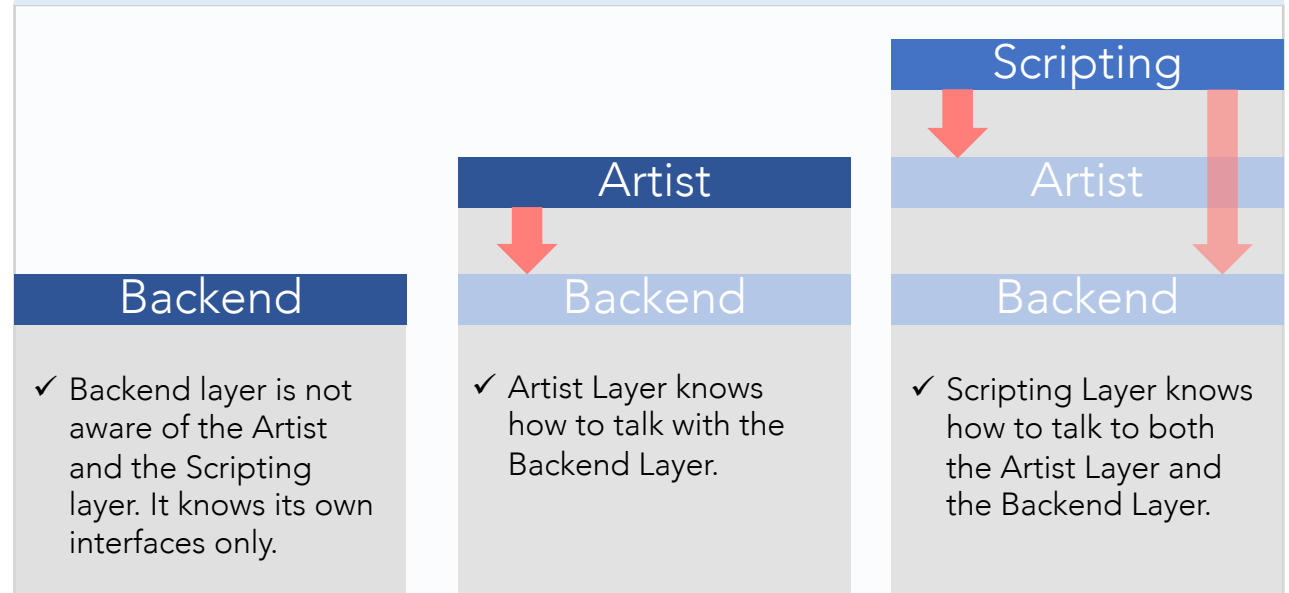
MATPLOTLIB ARCHITECTURE

Logical Separation into Three Layers



- ✓ Enable the users to create, render, and update the figure objects.
- ✓ The Three layers can be visualised as stack from bottom to top
- ✓ The Three layers perform distinct role at different levels of awareness.

ISOLATION OF THE COMPLEXITIES IN EACH LAYER

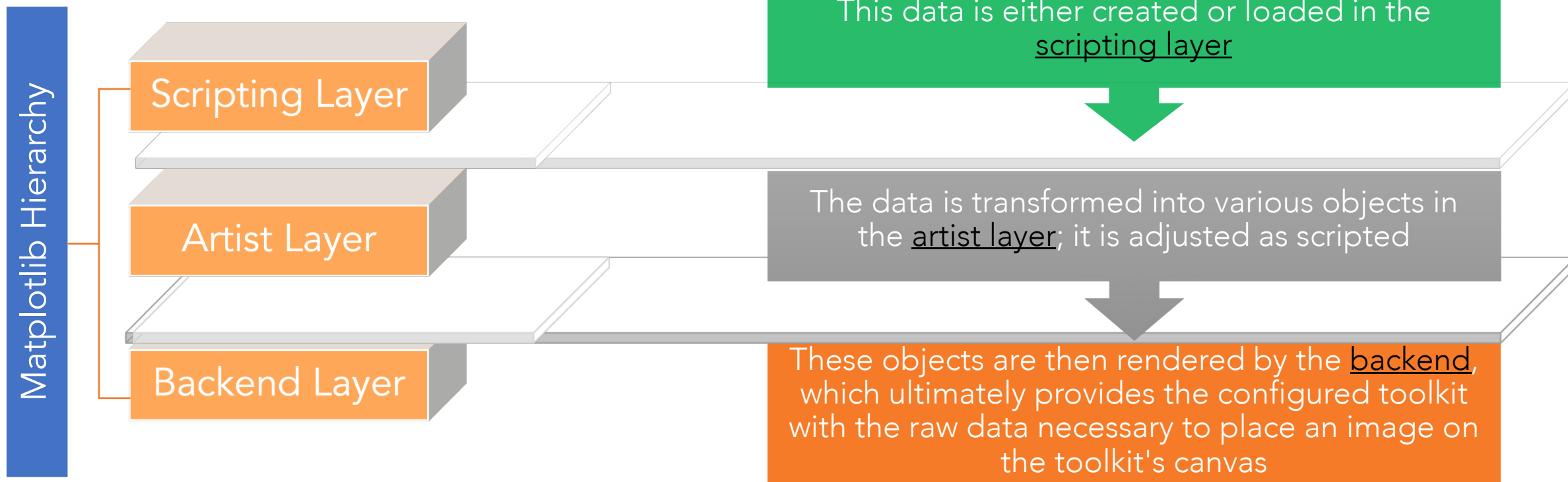


*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGreggor, Pg no.15

RELATIONSHIP OF THE **LAYERS** TO **DATA** AND THE **FIGURE OBJECT** FOR A GIVEN PLOT



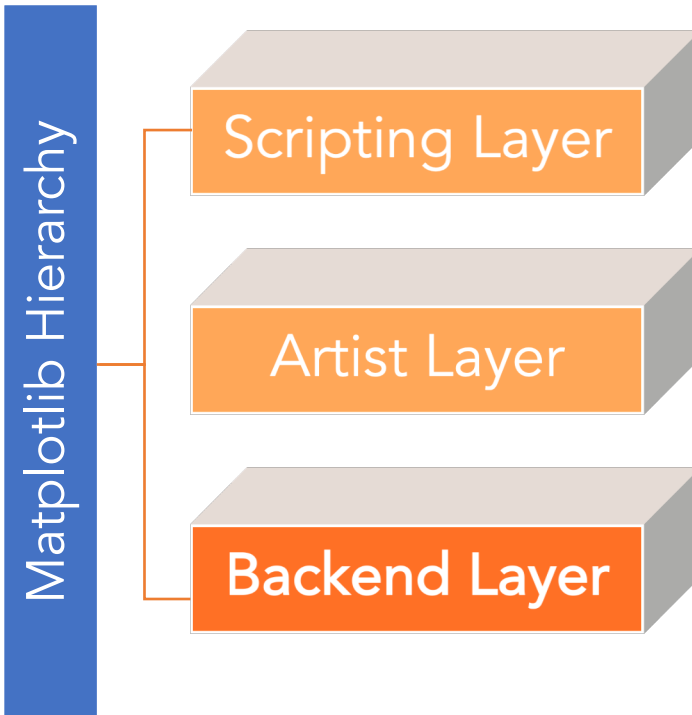
*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGreggor, Pg no.15

DEEP DIVE INTO BACKEND LAYER

Backend Layer refers to matplotlib capabilities to render plots for *different use cases and output formats*.



DIFFERENT USE CASES



Use Matplotlib interactively from the **python shell** and have plotting windows pop up when they type commands.



Some people run **Jupyter notebooks** and draw inline plots for quick data analysis.



Others embed Matplotlib into graphical user interfaces like **PyQt or PyGObject** to build rich applications.



Some people use Matplotlib in **batch scripts** to generate postscript images from numerical simulations



Others run **web application servers** to dynamically serve up graphs.

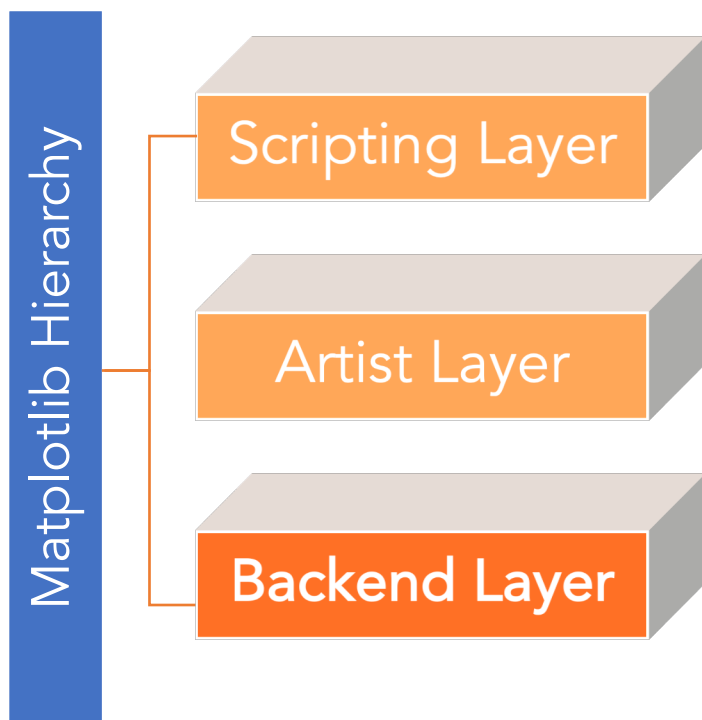
*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

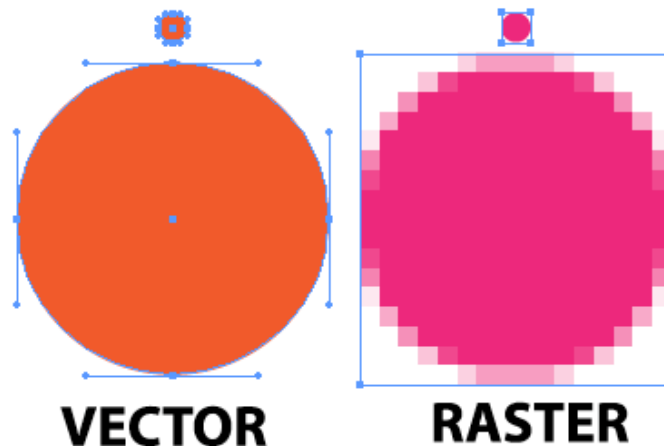
*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

DEEP DIVE INTO BACKEND LAYER

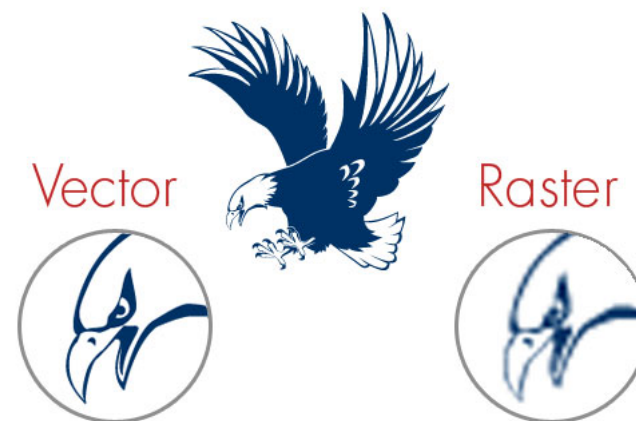
Backend Layer refers to matplotlib capabilities to render plots for *different use cases and output formats*.



DIFFERENT OUTPUT FORMATS VECTOR OR RASTER BASED



Vector Graphics use **clean lines and shapes** that can be scaled to any size



Raster Graphics use **pixels** arranged in grid formation to represent an image

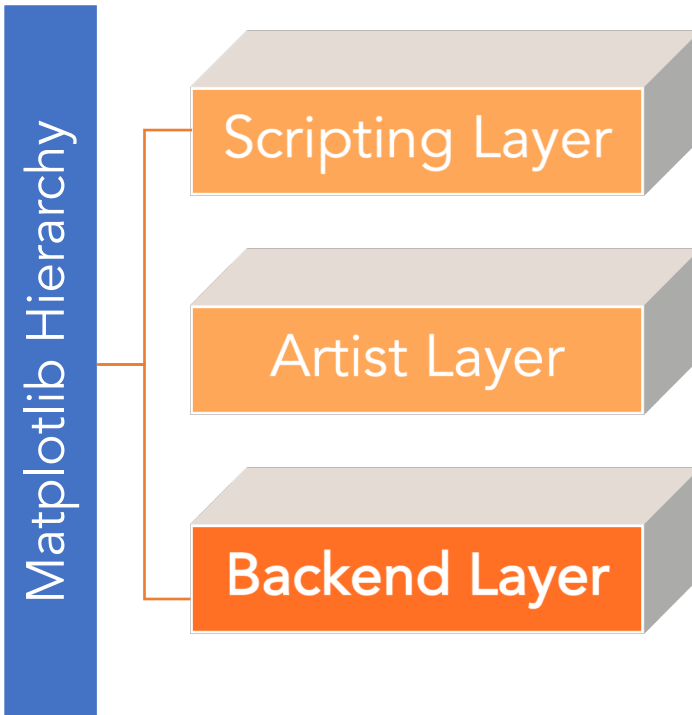
*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. Mcgreggor, Pg no.15

DEEP DIVE INTO BACKEND LAYER

Backend Layer refers to matplotlib capabilities to render plots for *different use cases and output formats*.



DIFFERENT OUTPUT FORMATS VECTOR OR RASTER BASED

Renderer	Output File Types	Description
AGG	PNG	raster graphics -- high quality images using the Anti-Grain Geometry engine
PDF	PDF	vector graphics -- Portable Document Format
PS	PS, EPS	vector graphics -- Postscript output
SVG	SVG	vector graphics -- Scalable Vector Graphics
PGF	PGF, PDF	vector graphics -- using the pgf package
CAIRO	PNG, PS, PDF, SVG	raster or vector graphics -- using the Cairo library

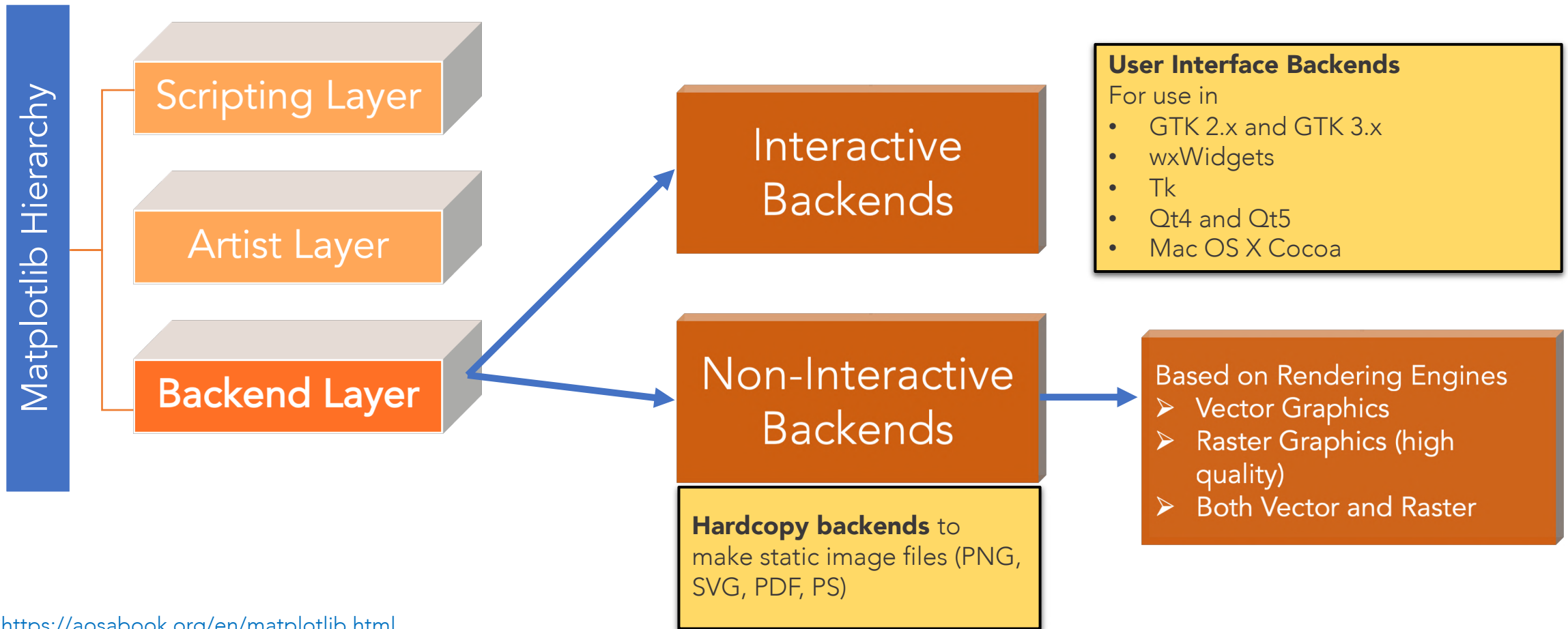
*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGreggor, Pg no.15

DEEP DIVE INTO BACKEND LAYER

Backend Layer refers to matplotlib capabilities to render plots for *different use cases and output formats*.

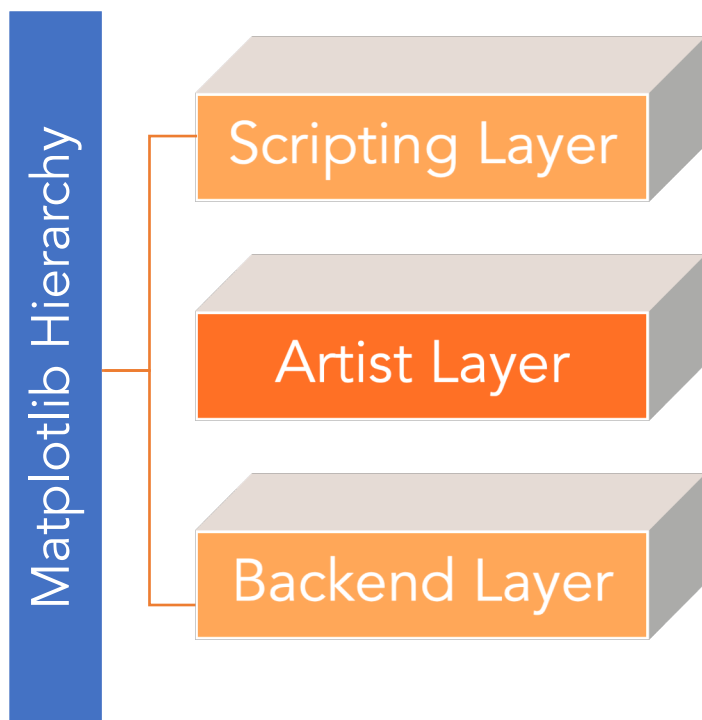


*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGreggor, Pg no.15

DEEP DIVE INTO ARTIST LAYER



The Artist layer constitutes the bulk of what matplotlib actually does — the generation of the plots.

Most work in the artist layer is performed by a number of classes, most of which are derived from the **Artist base class**.

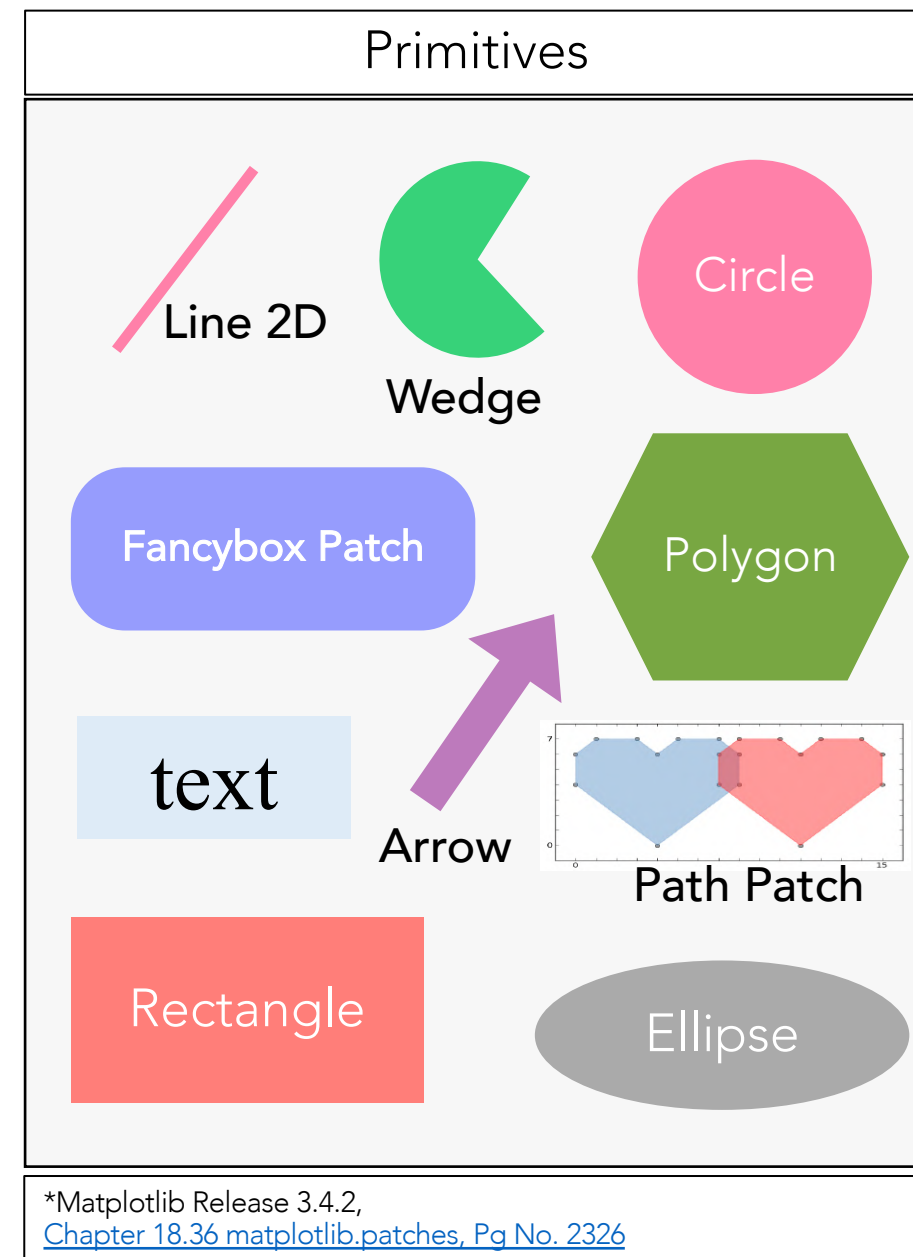
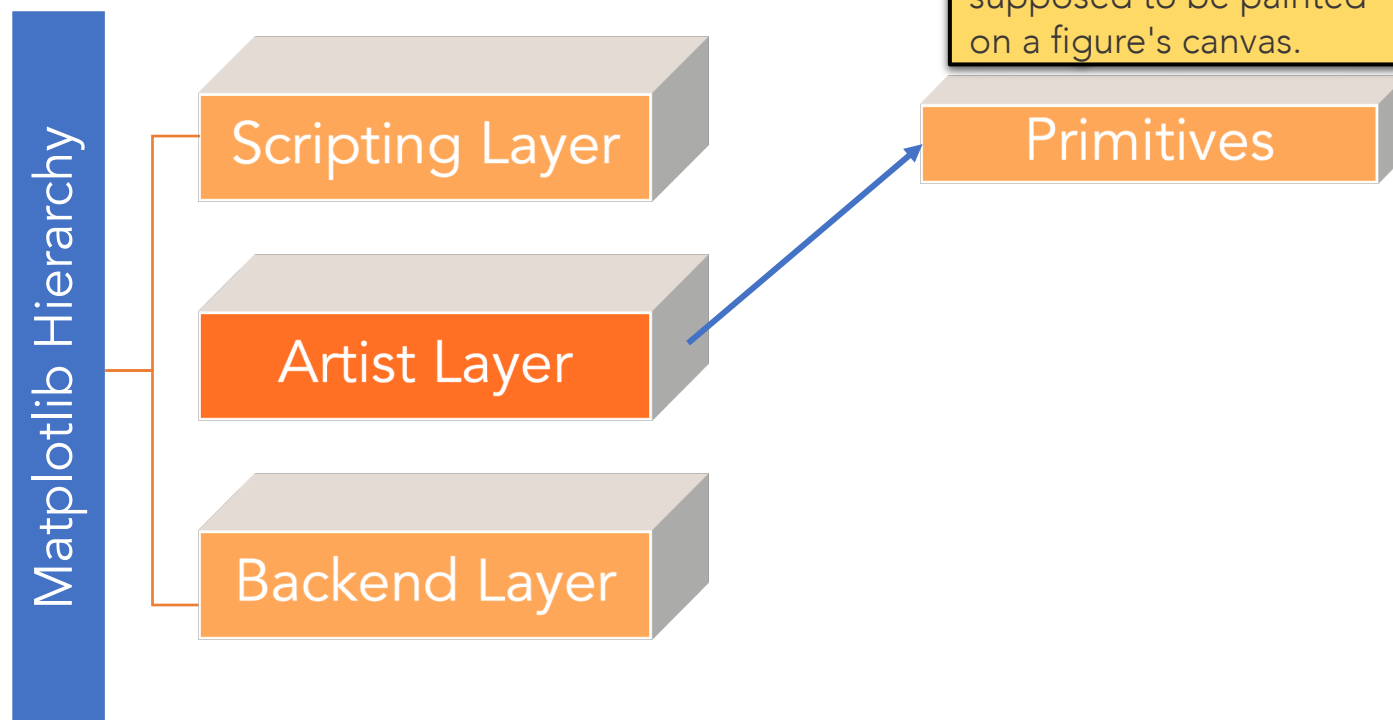
Let's run through the **sub classes** in the Artist Layer...

*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

DEEP DIVE INTO ARTIST LAYER

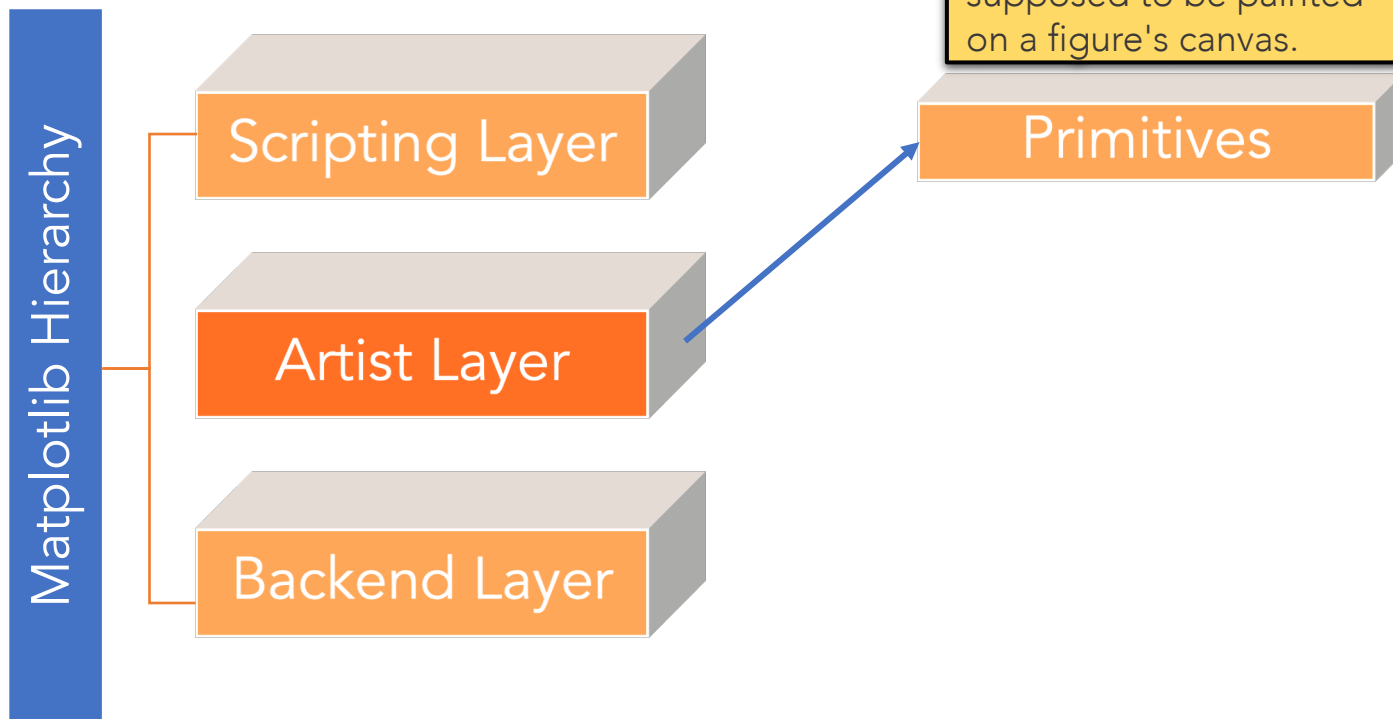


*<https://aosabook.org/en/matplotlib.html>

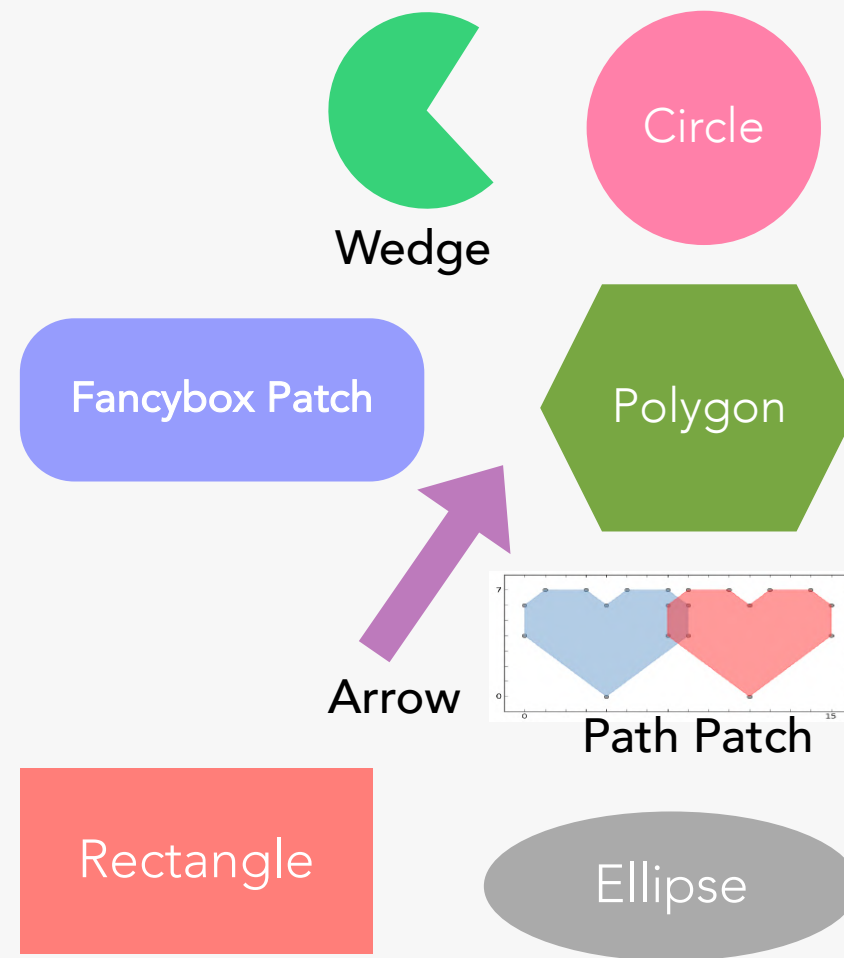
*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. Mcgreggor, Pg no.15

DEEP DIVE INTO ARTIST LAYER



Primitives – Patches Classes



*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

*Matplotlib Release 3.4.2,
[Chapter 18.36 matplotlib.patches, Pg No. 2326](#)

DEEP DIVE INTO ARTIST LAYER

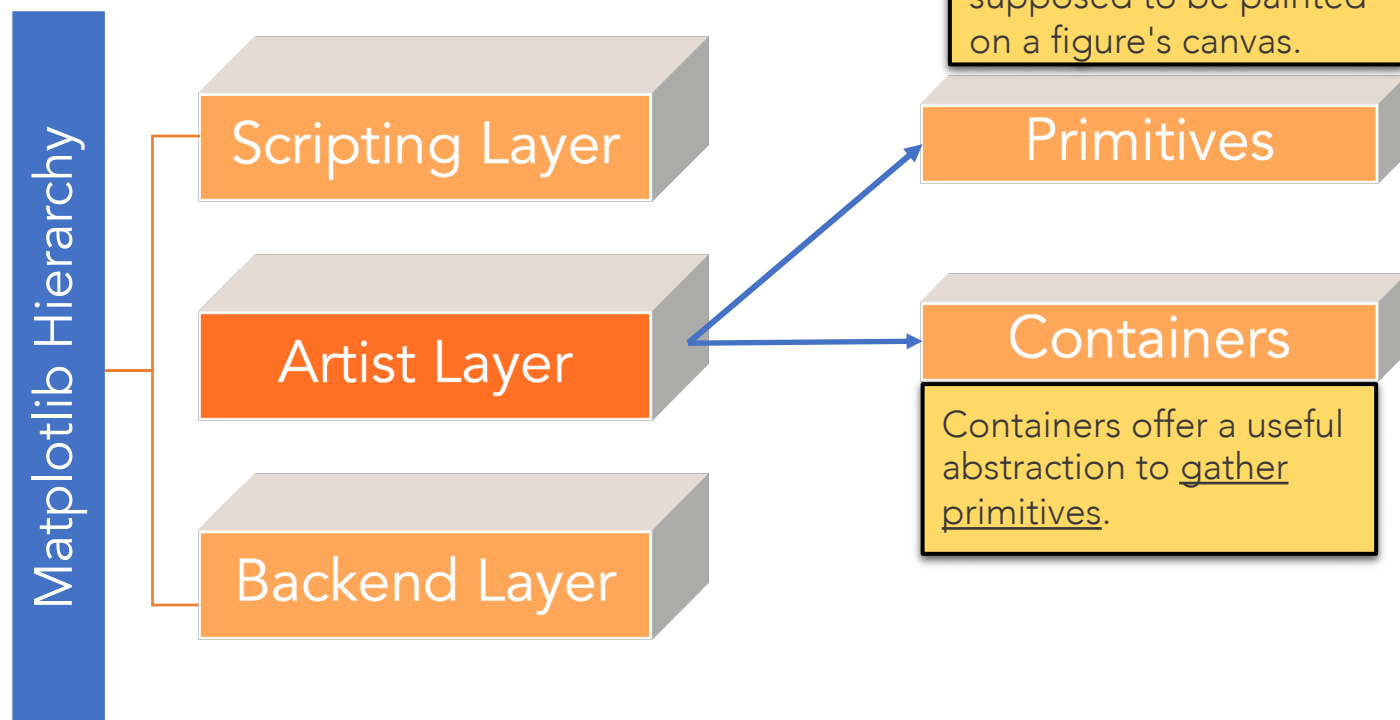
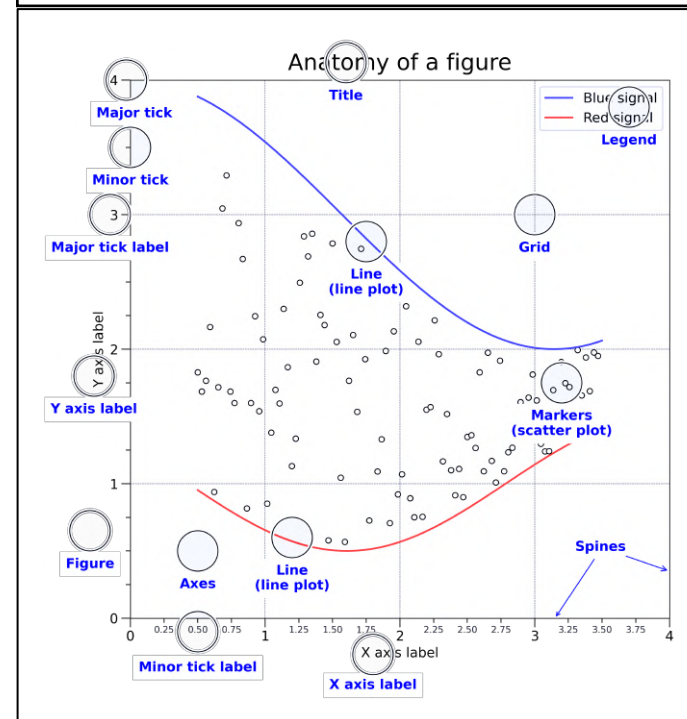


Figure - Top level Artist, which holds all plot elements

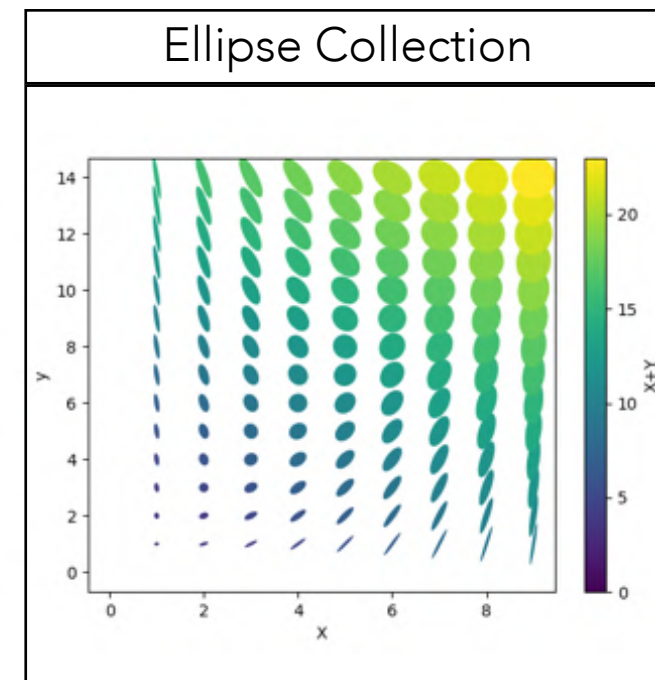
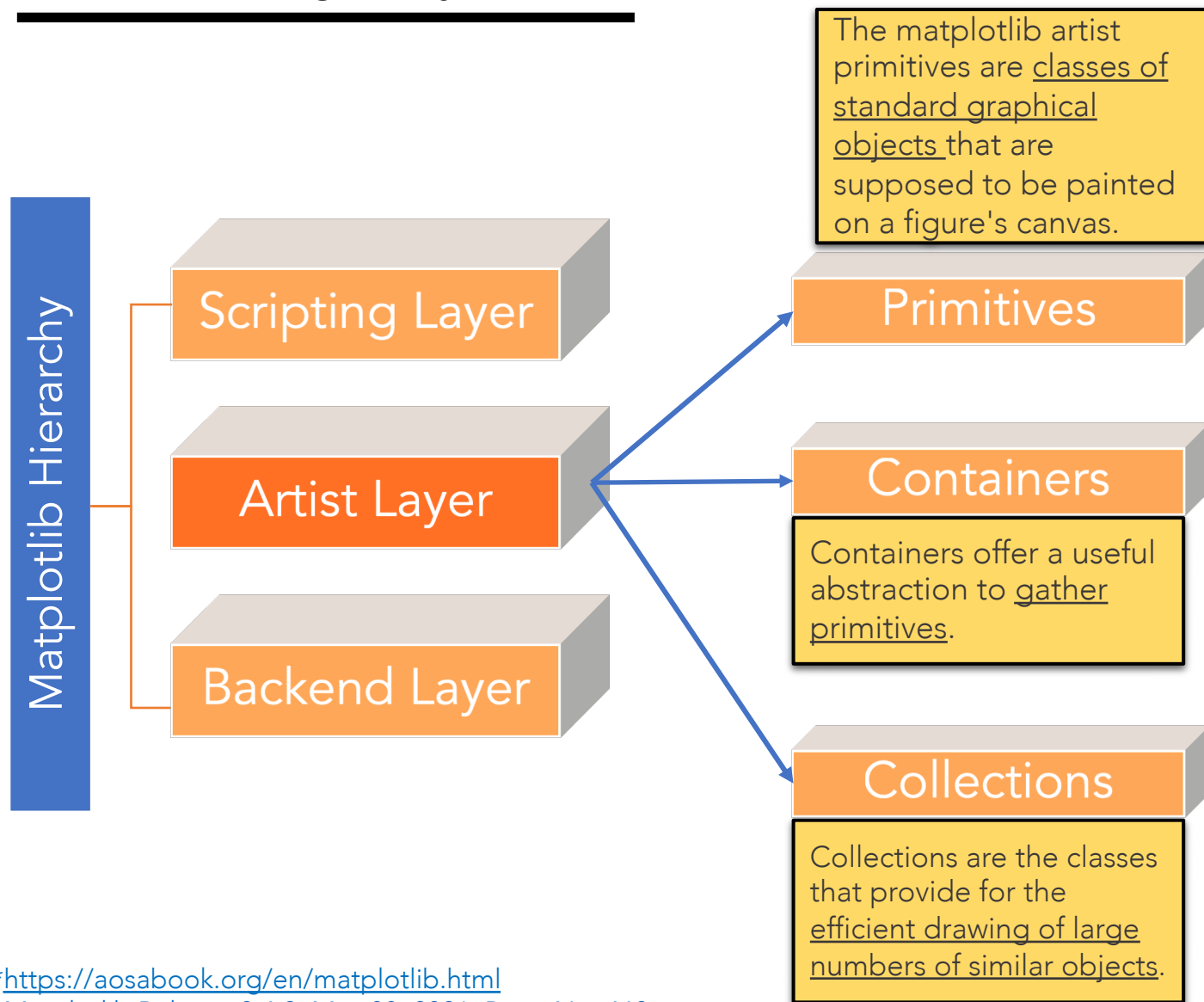


*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

DEEP DIVE INTO ARTIST LAYER

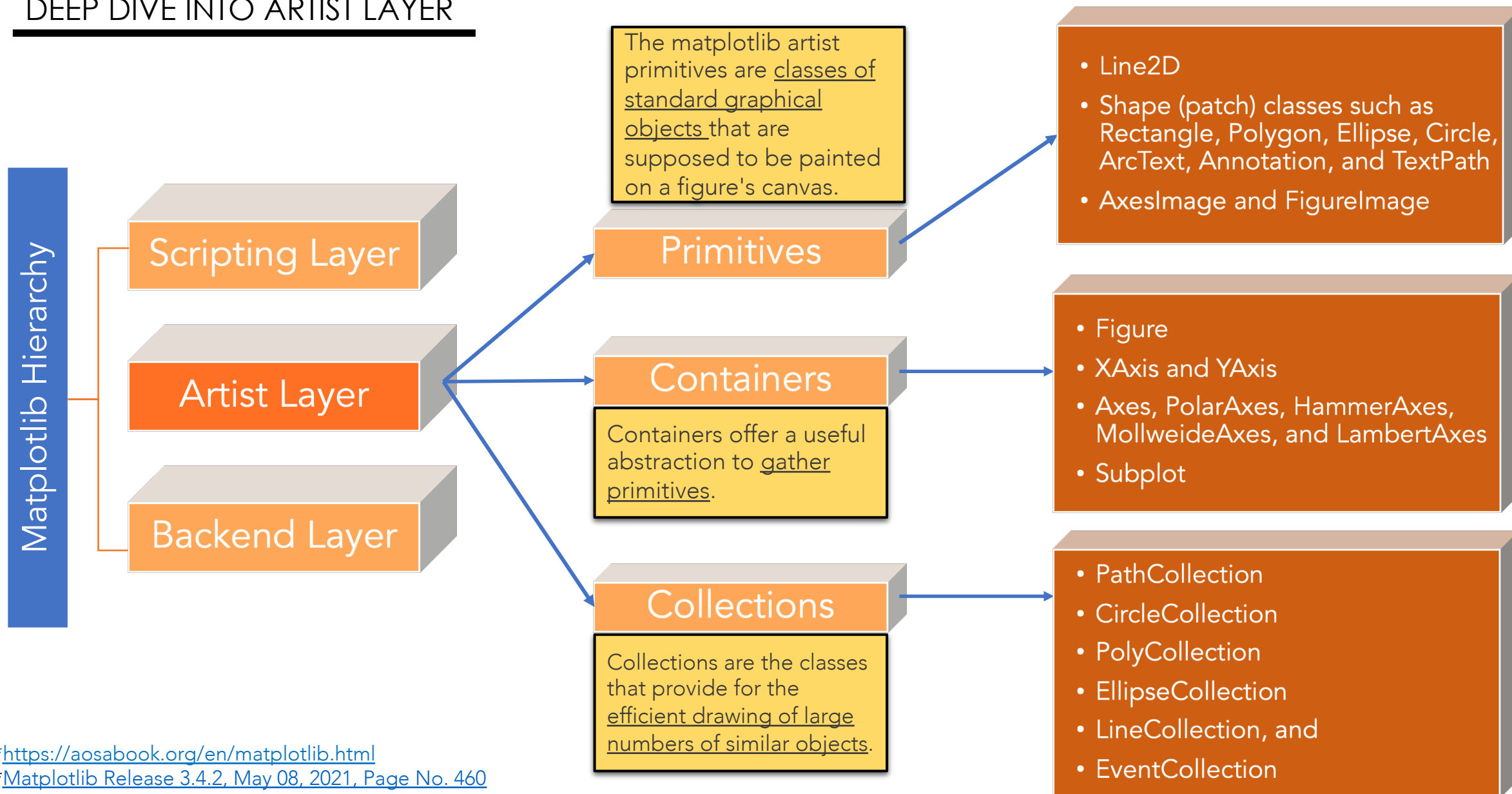


*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

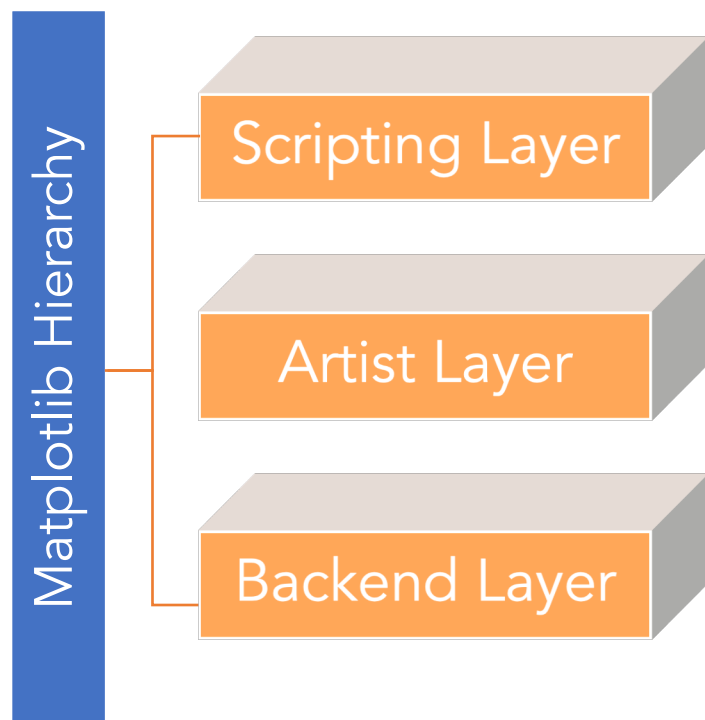
DEEP DIVE INTO ARTIST LAYER



*<https://aosabook.org/en/matplotlib.html>

*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15



The Standard Approach to Plotting using Artists



Create a Figure instance.

Use the Figure to create one or more Axes or Subplot instances

Use the Axes instance helper methods to create the primitives

*<https://aosabook.org/en/matplotlib.html>

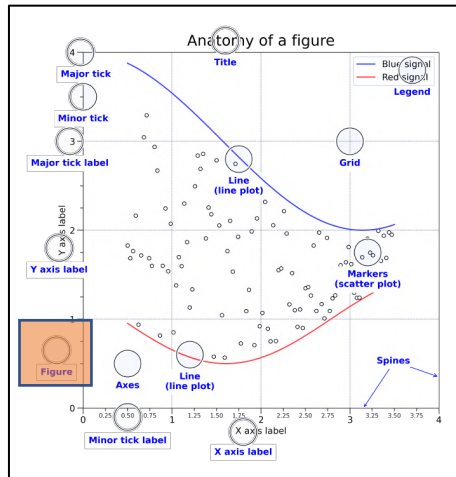
*[Matplotlib Release 3.4.2, May 08, 2021, Page No. 460](#)

*Mastering Matplotlib by Duncan M. McGregor, Pg no.15

We will explore
the approach to plotting in more detail...

But prior to that, it is important to know
the **elements (artists)** of a matplotlib figure.

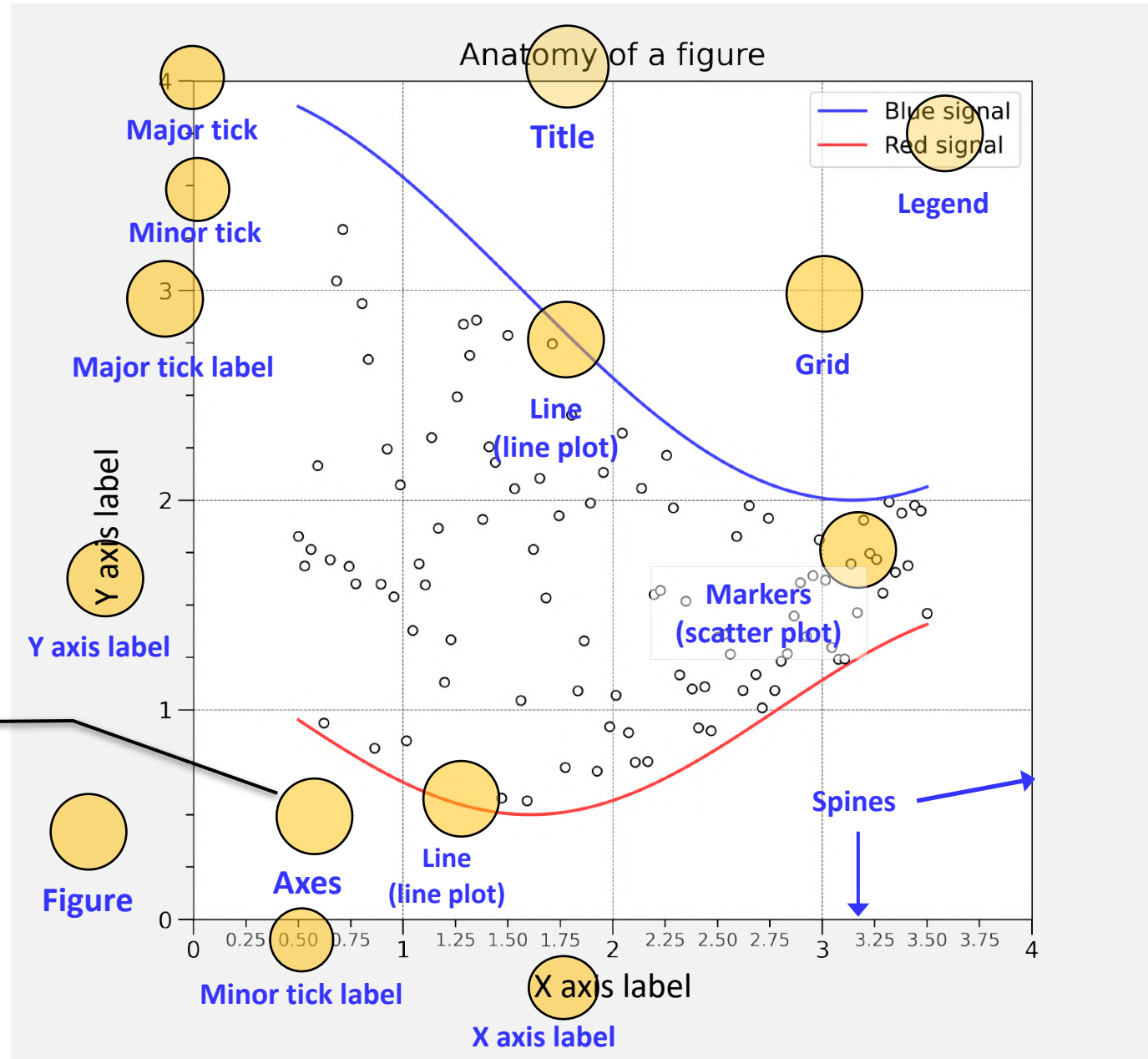
ANATOMY OF A FIGURE



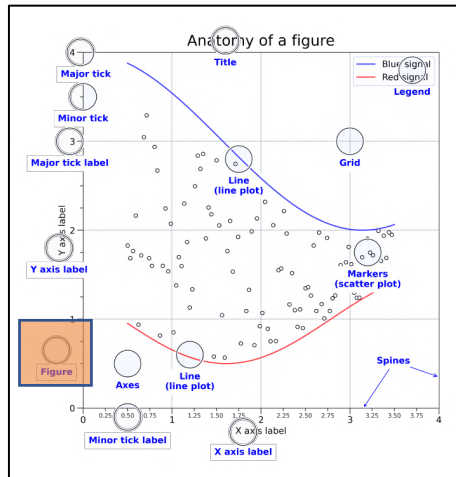
The Axes contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system.

Figure can contain axes or subplots.

Figure is top level container for all plot elements. It means the whole window in user interface.



ANATOMY OF A FIGURE



The Axes contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system.

Figure can contain axes or subplots.

Figure is top level container for all plot elements. It means the whole window in user interface.

Major tick

Minor tick

Major tick label

Title

Legend

Grid

Line
(line plot)

Y axis label

Markers
(scatter plot)

Spines

Figure

Axes

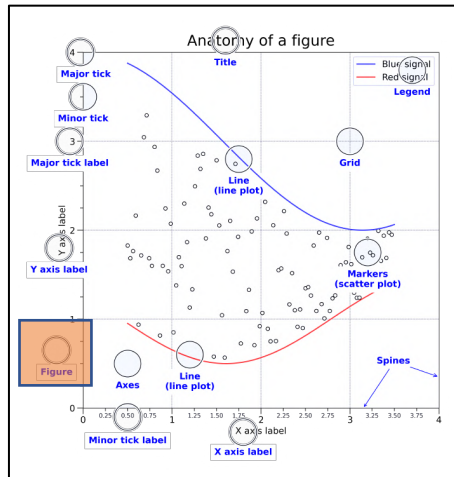
Line
(line plot)

Minor tick label

X axis label

- [Matplotlib Release 3.4.2, May 08, 2021, Pg No. 7-11](#)
- <https://dev.to/skotaro/artist-in-matplotlib---something-i-wanted-to-know-before-spending-hours-on-googling-how-to-3100>

ANATOMY OF A FIGURE



The Axes contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system.

Figure can contain axes or subplots.

Figure is top level container for all plot elements. It means the whole window in user interface.

Figure

Axes

X axis label

Y axis label

Major tick

Major tick label

Minor tick

Minor tick label

Grid

Spines

Line
(line plot)

Line
(line plot)

Markers
(scatter plot)

Legend

Title

- [Matplotlib Release 3.4.2, May 08, 2021, Pg No. 7-11](#)
- <https://dev.to/skotaro/artist-in-matplotlib---something-i-wanted-to-know-before-spending-tremendous-hours-on-googling-how-to-3100>

ANATOMY OF A FIGURE

Basically, everything you can see on the figure is an artist (even the **Figure**, **Axes**, and **Axis** objects).

This includes **Text** objects, **Line2D** objects, **collections** objects, **Patch** objects..

When the figure is rendered, all of the artists are drawn to the canvas.

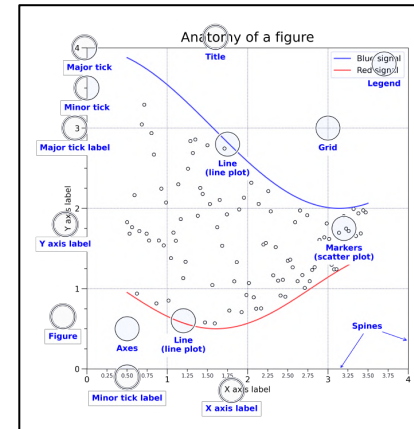
Most **Artists** are tied to an **Axes**; such an Artist cannot be shared by multiple Axes, or moved from one to another.

Figure

Axes

X axis label

Y axis label



Major tick

Major tick label

Minor tick

Minor tick label

Grid

Spines

Line
(line plot)

Line
(line plot)

Markers
(scatter plot)

Legend

Title

- [Matplotlib Release 3.4.2, May 08, 2021, Pg No. 7-11](#)
- <https://dev.to/skotaro/artist-in-matplotlib---something-i-wanted-to-know-before-spending-tremendous-hours-on-googling-how-to-s-31oo>

Let's run down through the hierarchy of Artists...

Representing Containers by yellow colored boxes.

Containers

Representing Primitives by Salmon colored boxes.

Primitives

Figure

Figure is the Top Level container

Figure can contain other Containers and Primitives.

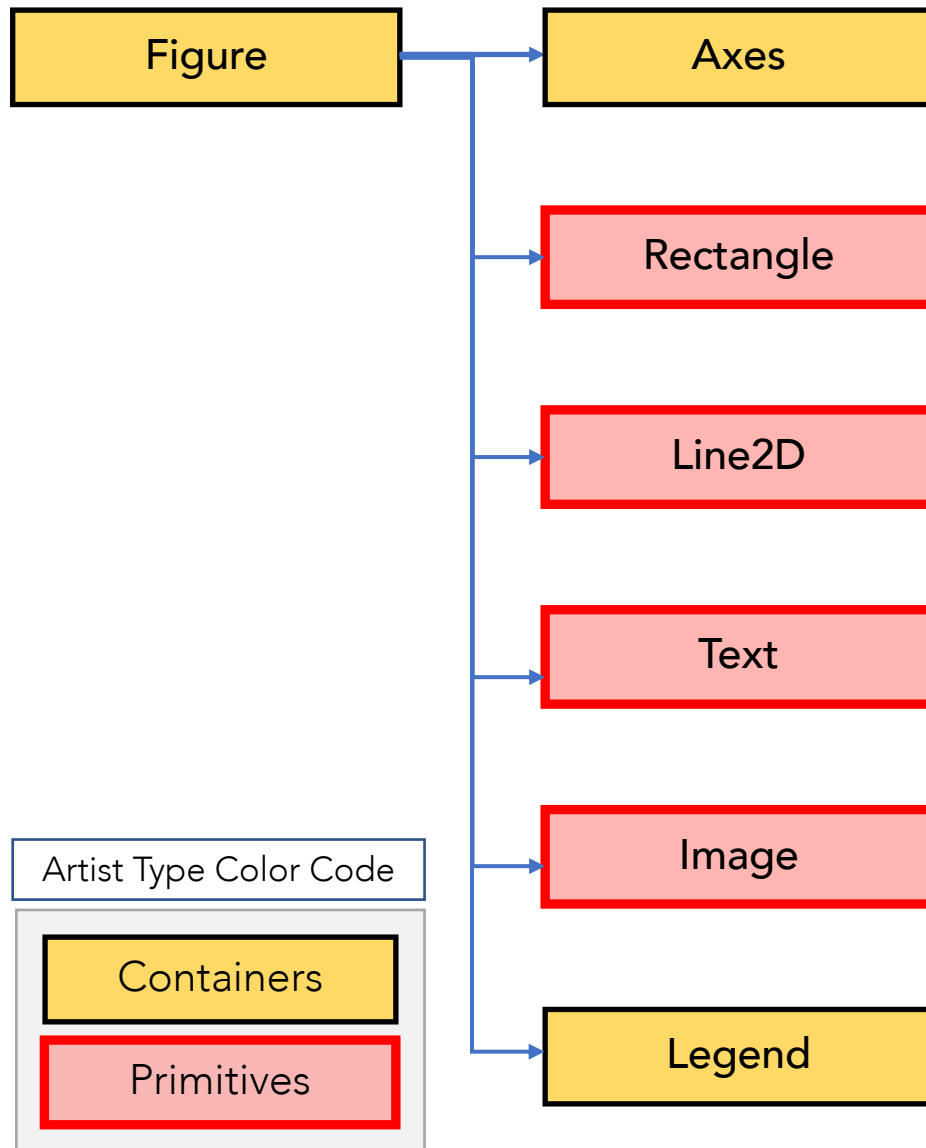
Figure can contain Axes and subplots.

The figure keeps track of all the **child Axes**, a smattering of 'special' artists (**titles**, **figure legends**, etc), and the **canvas**

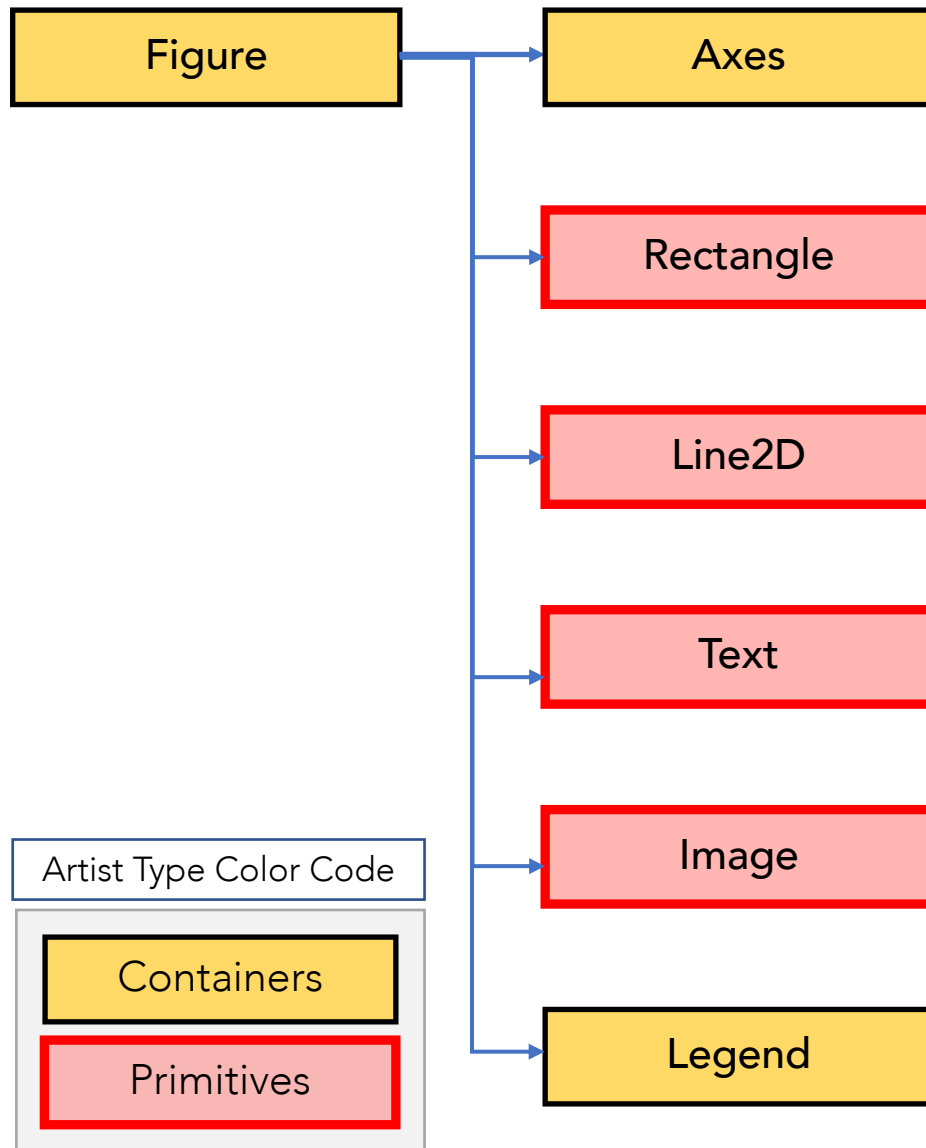
Artist Type Color Code

Containers

Primitives



The figure keeps track of all the **child Axes**, a smattering of 'special' artists (**titles**, **figure legends**, etc), and the **canvas**

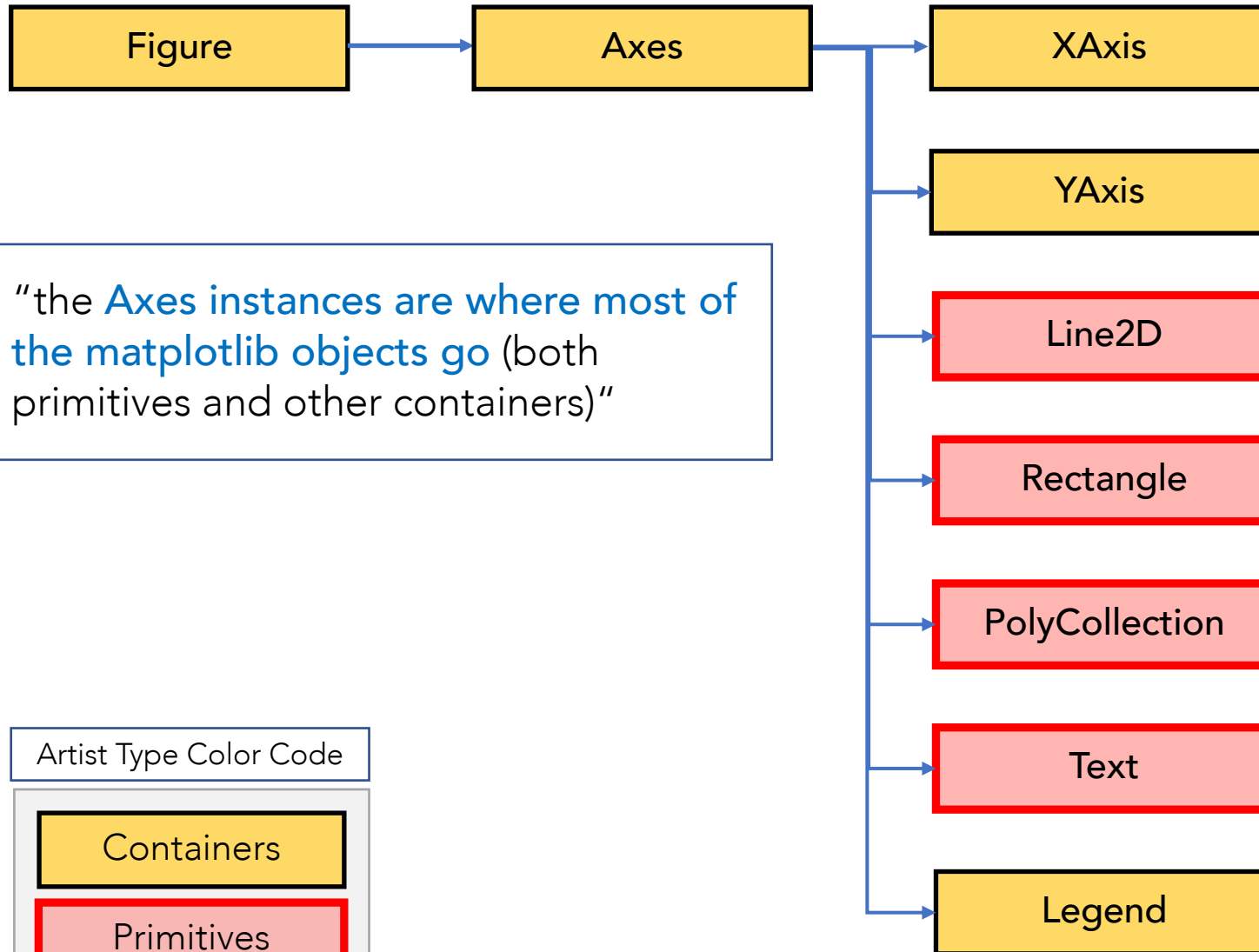


Let us focus on Axes.

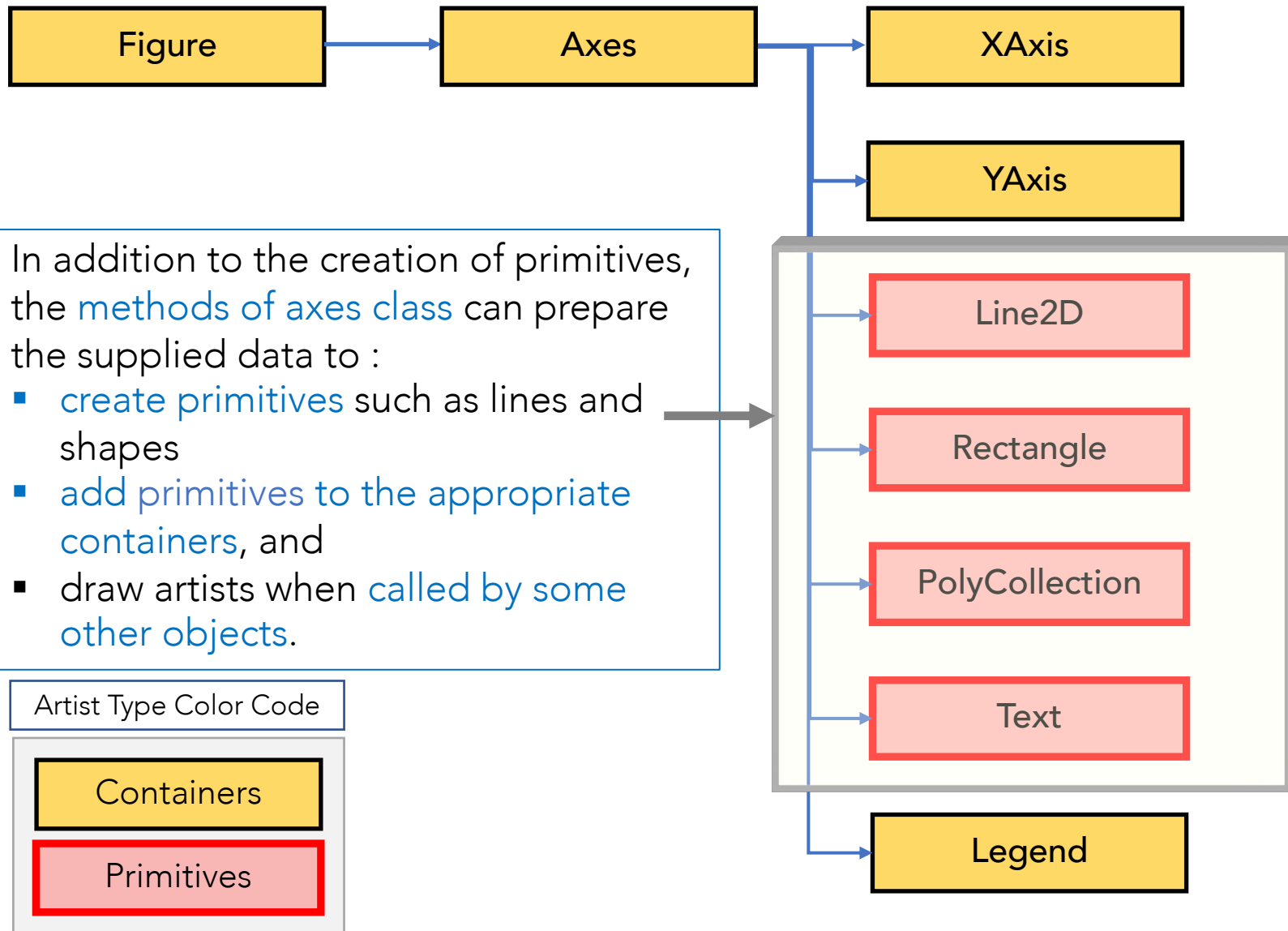
Note the term 'Axes' is not related to term 'axis'.

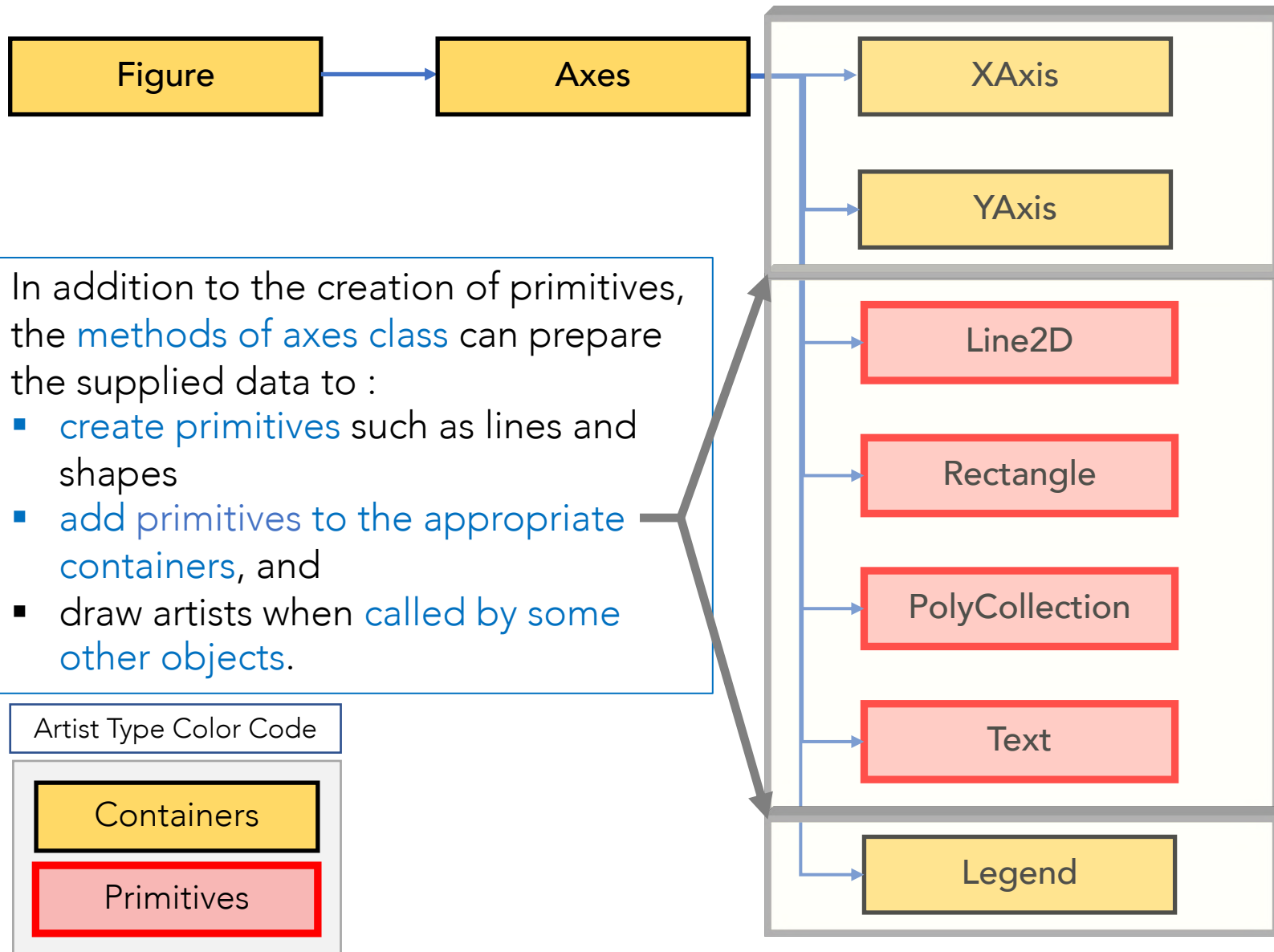
The Axes class is one of the most important. It is the **primary mover and shaker** in the artist layer.

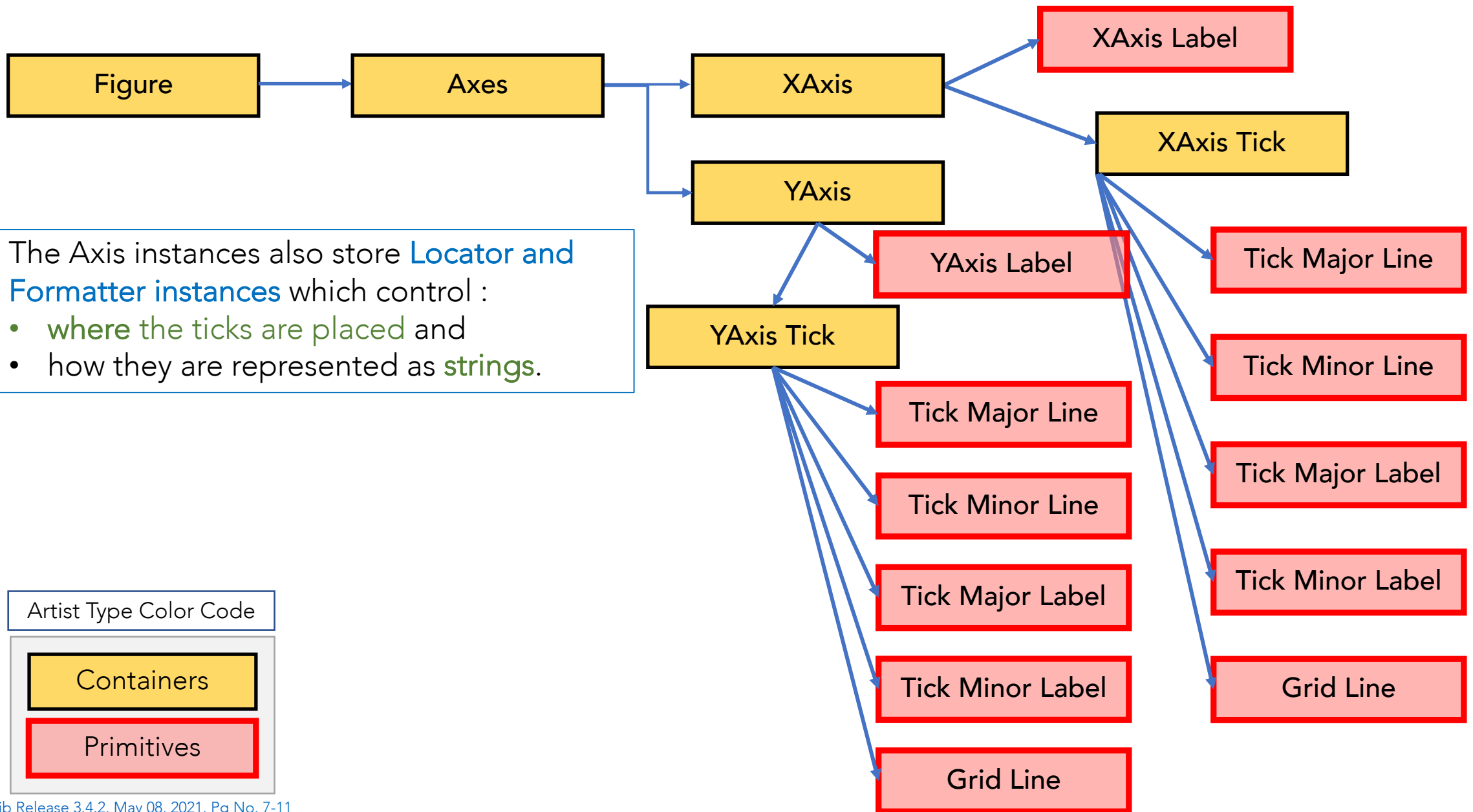
The reason for this is simple—the **Axes instances are where most of the matplotlib objects go** (both primitives and other containers)



• [Matplotlib Release 3.4.2, May 08, 2021, Pg No. 7-11](#)
• <https://dev.to/skotaro/artist-in-matplotlib---something-i-wanted-to-know-before-spending-hours-on-googling-how-to-s-31oo>





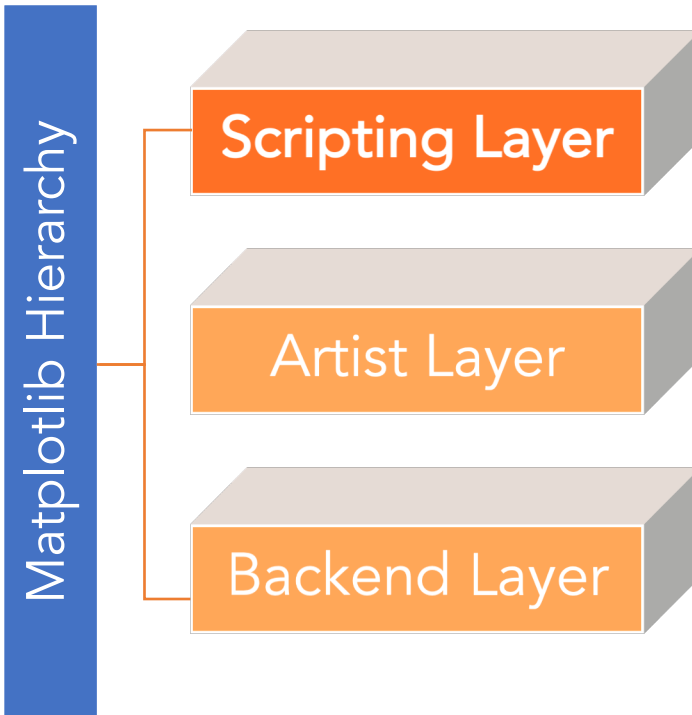


Having reviewed the **Artists Hierarchy**...

We shall move on to the “Scripting Layer”

SCRIPTING LAYER – DUAL INTERFACE TO MATPLOTLIB API

Scripting Layer is the User Facing Interface from where User can make calls to the Artist layer and backend layer.



Two ways to use Matplotlib:

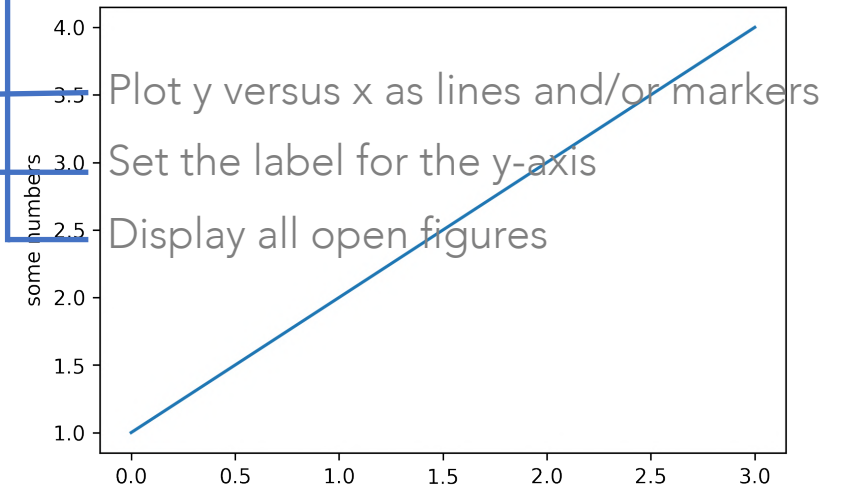
- **Pyplot API** to **automatically create and manage** the figures and axes, and use pyplot functions for plotting
- **Explicitly create figures and axes**, and call methods on them (the "**object-oriented (OO) style**")

Pyplot Interface is a **Stateful Interface**
styled on MATLAB

`matplotlib.pyplot` is a collection of functions that make matplotlib work like MATLAB.

- Each pyplot function makes some change to a figure.
- Various states are preserved across function calls, so that **pyplot keeps track of things** like the current figure and plotting area.
- All plotting functions apply to the **current axes**.

```
import matplotlib.pyplot as plt  
plt.plot([1, 2, 3, 4])  
plt.ylabel('some numbers')  
plt.show()
```



Functions	Description
<code>plot(*args[, scalex, scaley, data])</code>	Plot y versus x as lines and/or markers.
<code>axes([arg])</code>	Add an axes to the current figure and make it the current axes.
<code>axis(*args[, emit])</code>	Convenience method to get or set some axis properties.
<code>xlabel(xlabel[, fontdict, labelpad, loc])</code>	Set the label for the x-axis.
<code>ylabel(ylabel[, fontdict, labelpad, loc])</code>	Set the label for the y-axis.
<code>xticks([ticks, labels])</code>	Get or set the current tick locations and labels of the x-axis.
<code>yticks([ticks, labels])</code>	Get or set the current tick locations and labels of the y-axis.
<code>grid</code>	Configure the grid lines
<code>xlim(*args, **kwargs)</code>	Get or set the x limits of the current axes.
<code>ylim(*args, **kwargs)</code>	Get or set the y-limits of the current axes.
<code>legend(*args, **kwargs)</code>	Place a legend on the Axes.
<code>figlegend(*args, **kwargs)</code>	Place a legend on the figure.
<code>savefig(*args, **kwargs)</code>	Save the current figure.
<code>show(*[, block])</code>	Display all open figures.

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

The **plt.axis()** method allows you to set the **x and y limits** with a single call, by passing a list that specifies **[xmin, xmax, ymin, ymax]**.

Text

Functions	Description
<code>suptitle(t, **kwargs)</code>	Add a centered suptitle to the figure.
<code>annotate(text, xy, **kwargs)</code>	Annotate the point xy with text text.
<code>text(x, y, s[, fontdict])</code>	Add text to the Axes.
<code>title(label[, fontdict, loc, pad, y])</code>	Set a title for the Axes.
<code>figtext(x, y, s[, fontdict])</code>	Add text to figure.

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

Line, Span

Functions	Description
<code>axhline([y, xmin, xmax])</code>	Add a horizontal line across the axis.
<code>axvline([x, ymin, ymax])</code>	Add a vertical line across the Axes.
<code>axhspan(ymin, ymax[, xmin, xmax])</code>	Add a horizontal span (rectangle) across the Axes.
<code>axvspan(xmin, xmax[, ymin, ymax])</code>	Add a vertical span (rectangle) across the Axes.
https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview	

Functions	Description
<code>bar(x, height[, width, bottom, align, data])</code>	Make a bar plot.
<code>barh(y, width[, height, left, align])</code>	Make a horizontal bar plot.
<code>bar_label(container[, labels, fmt, ...])</code>	Label a bar plot.
<code>scatter(x, y[, s, c, marker, cmap, norm, ...])</code>	A scatter plot of y vs. x
<code>colorbar([mappable, cax, ax])</code>	Add a colorbar to a plot
<code>hist(x[, bins, range, density, weights, ...])</code>	Plot a histogram.
<code>hist2d(x, y[, bins, range, density, ...])</code>	Make a 2D histogram plot.
<code>boxplot(x[, notch, sym, vert, whis, ...])</code>	Make a box and whisker plot.
<code>violinplot(dataset[, positions, vert, ...])</code>	Make a violin plot.
<code>stackplot(x, *args[, labels, colors, ...])</code>	Draw a stacked area plot.
<code>pie(x[, explode, labels, colors, autopct, ...])</code>	Plot a pie chart.
<code>stem(*args[, linefmt, markerfmt, basefmt, ...])</code>	Create a stem plot.
<code>step(x, y, *args[, where, data])</code>	Make a step plot.
<code>hexbin(x, y[, C, gridsize, bins, xscale, ...])</code>	Make a 2D hexagonal binning plot of points x, y.
<code>table([cellText, cellColours, cellLoc, ...])</code>	Add a table to an Axes .
https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview	

Line, Area

Functions	Description
<code>hlines(y, xmin, xmax[, colors, linestyles, ...])</code>	Plot horizontal lines at each y from xmin to xmax.
<code>vlines(x, ymin, ymax[, colors, linestyles, ...])</code>	Plot vertical lines at each x from ymin to ymax.
<code>fill(*args[, data])</code>	Plot filled polygons.
<code>fill_between(x, y1[, y2, where, ...])</code>	Fill the area between two horizontal curves.
<code>fill_betweenx(y, x1[, x2, where, step, ...])</code>	Fill the area between two vertical curves.

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

Current Axes, Figure

Functions	Description
<code>gcf()</code>	Get the current figure.
<code>gca(***kwargs)</code>	Get the current Axes, creating one if necessary.
<code>gci()</code>	Get the current colorable artist.
<code>sca()</code>	Set the current Axes to ax and the current Figure to the parent of ax.
<code>cla()</code>	Clear the current axes.
<code>clf()</code>	Clear the current figure.
<code>delaxes([ax])</code>	Remove an Axes (defaulting to the current axes) from its figure.
<code>findobj</code>	Find artist objects.
<code>getp(obj, *\args, ***kwargs)</code>	Return the value of an Artist's property, or print all of them.
<code>box([on])</code>	Turn the axes box on or off on the current axes.
<code>get(obj, *\args, ***kwargs)</code>	Return the value of an Artist's property, or print all of them.
<code>setp(obj, *\args, ***kwargs)</code>	Set one or more properties on an Artist , or list allowed values.
<code>tick_params([axis])</code>	Change the appearance of ticks, tick labels, and gridlines.
<code>locator_params([axis, tight])</code>	Control behavior of major tick locators.
<code>minorticks_off()</code>	Remove minor ticks from the axes.
<code>minorticks_on()</code>	Display minor ticks on the axes

Inspecting, Changing Properties

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

FREQUENTLY USED MATPLOTLIB.PYPLOT FUNCTIONS

For Colormaps, Pseudo Colorplots

Functions	Description
pcolor (*args[, shading, alpha, norm, cmap, ...])	Create a pseudocolor plot with a non-regular rect- angular grid.
pcolormesh (*args[, alpha, norm, cmap, ...])	Create a pseudocolor plot with a non-regular rect- angular grid.
imread (fname[, format])	Read an image from a file into an array.
imsave (fname, arr, **kwargs)	Save an array as an image file.
imshow (X[, cmap, norm, aspect, ...])	Display data as an image, i.e., on a 2D regular raster.

Functions	Description
autumn()	Set the colormap to 'autumn'.
bone()	Set the colormap to 'bone'.
cool()	Set the colormap to 'cool'.
copper()	Set the colormap to 'copper'.
flag()	Set the colormap to 'flag'.
gray()	Set the colormap to 'gray'.
hot()	Set the colormap to 'hot'.
hsv()	Set the colormap to 'hsv'.
inferno()	Set the colormap to 'inferno'.
jet()	Set the colormap to 'jet'.
magma()	Set the colormap to 'magma'.
nipy_spectral()	Set the colormap to 'nipy_spectral'.
pink()	Set the colormap to 'pink'.

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

Functions	Description
<code>subplot(*args, **kwargs)</code>	Add an Axes to the current figure or retrieve an existing Axes.
<code>subplots([nrows, ncols, sharex, sharey, ...])</code>	Create a figure and a set of subplots.
<code>subplot2grid(shape, loc[, rowspan, colspan, fig])</code>	Create a subplot at a specific location inside a regular grid.
<code>subplots_adjust([left, bottom, right, top, ...])</code>	Adjust the subplot layout parameters.
<code>tight_layout(*[, pad, h_pad, w_pad, rect])</code>	Adjust the padding between and around subplots.

https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot#pyplot-function-overview

Code Snippet

```
names = ['group_a', 'group_b', 'group_c']  
values = [1, 10, 100]
```

```
plt.figure(figsize=(9, 3))
```

```
plt.subplot(131)
```

```
plt.bar(names, values)
```

```
plt.subplot(132)
```

```
plt.scatter(names, values)
```

```
plt.subplot(133)
```

```
plt.plot(names, values)
```

```
plt.suptitle('Categorical Plotting')
```

```
plt.savefig('Categorical Plotting.png', \n            bbox_inches = "tight", \n            pad_inches = 0.15, dpi=300)  
plt.show()
```

Subplot function to add/change current axes

Pyplot function on current axes

plt.subplot adds axes to current figure.

Everytime it is called, it resets the current axes.

Subsequent pyplot functions will be applicable to current axes.

131 means in a grid of one row by three columns, the subplot at index position 1 is set as the current axes.

A Bar chart is plotted on the current axes.

The current axes is reset.

132 means in a grid of one row by three columns, the subplot at index position 2 is set as the current axes.

A **Scatter chart** is plotted on the current axes.

Current Axes is reset and a Line plot is drawn.

Centered supitle is set in the Figure.

Figure is saved as png file.

Figure is displayed in user screen

Code Snippet

```
names = ['group_a', 'group_b', 'group_c']  
values = [1, 10, 100]
```

```
plt.figure(figsize=(9, 3))
```

```
plt.subplot(131)
```

```
plt.bar(names, values)
```

```
plt.subplot(132)
```

```
plt.scatter(names, values)
```

```
plt.subplot(133)
```

```
plt.plot(names, values)
```

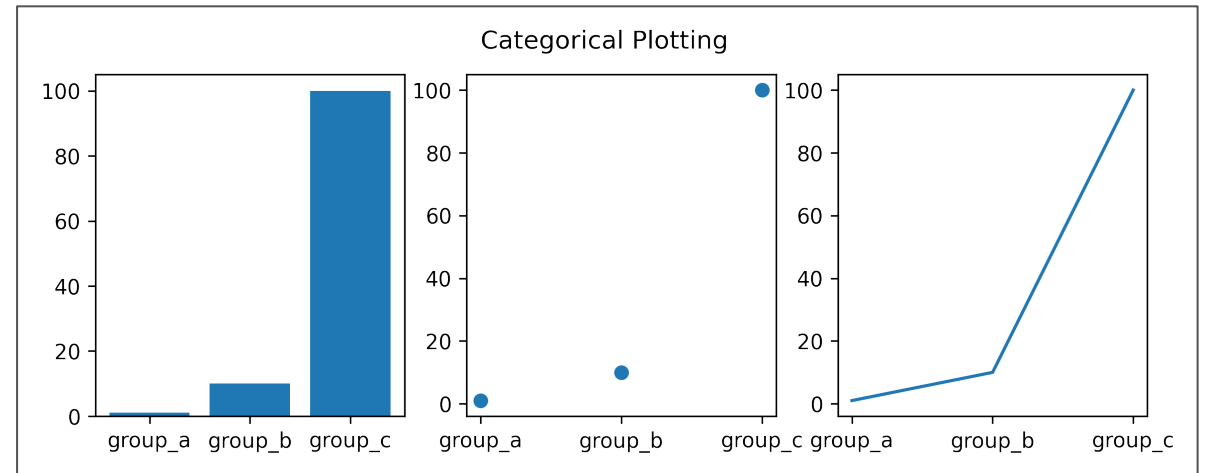
```
plt.suptitle('Categorical Plotting')
```

```
plt.savefig('Categorical Plotting.png', \n            bbox_inches = "tight", \n            pad_inches = 0.15, dpi=300)
```

```
plt.show()
```

Subplot function to add/change current axes

Pyplot function on current axes



Change rcParams settings

CONFIGURATIONS	
Functions	Description
<code>rc(group, **kwargs)</code>	Set the current <code>rcParams</code> . group is the grouping for the rc, e.g., for <code>lines.linewidth</code> the group is <code>lines</code> , for <code>axes.facecolor</code> , the group is <code>axes</code> , and so on. Group may also be a list or tuple of group names, e.g., <code>(xtick, ytick)</code> . <code>kwargs</code> is a dictionary attribute name/value pairs,
<code>rc_context([rc, fname])</code>	Return a context manager for temporarily changing <code>rcParams</code> .
<code>rcdefaults()</code>	Restore the <code>rcParams</code> from Matplotlib's internal default style.

*Matplotlib Release 3.4.2, May 08, 2021, Customizing Matplotlib with style sheets and rcParams, Pg No.84, 25

*rc refers to Matplotlib runtime configurations

Inspect

```
import matplotlib as mpl
mpl.rcParams['lines.linewidth']
```

Change

```
import matplotlib as mpl
mpl.rcParams['lines.linewidth'] = 2
```

“rcParams” – dict like Global variable
Stores settings

Settings Stored

Dictionary Like Variable that stores rc settings and Global to Matplotlib package

Functions

Description

`mpl.rcParamsDefaults`

Stores the default `rcParams`.

`mpl.rcParamsOrig`

Stores the `rcParams` in original `matplotlibrc` file

`mpl.rcParams`

Stores the current `rcParams`

[*Matplotlib Release 3.4.2, May 08, 2021, Customizing Matplotlib with style sheets and rcParams, Pg No.86](#)

rcParams function for global changes

Code Snippet for customizing configurations

```
import matplotlib.pyplot as plt
from matplotlib importycler
colors = cycler('color', ['#EE6666', '#3388BB',
                          '#9988DD', '#EECC55', '#88BB44',
                          '#FFBBBB'])
```

```
plt.rc('axes', facecolor='#E6E6E6',
       edgecolor='none', axisbelow=True,
       grid=True,
       prop_cycle=colors)
```

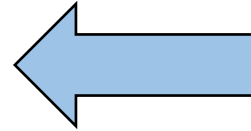
```
plt.rc('grid', color='w', linestyle='solid')
```

```
plt.rc('xtick', direction='out', color='gray')
```

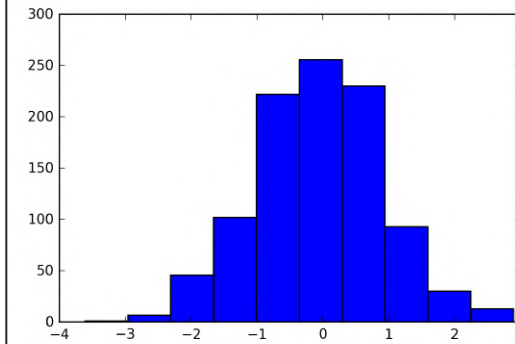
```
plt.rc('ytick', direction='out', color='gray')
```

```
plt.rc('patch', edgecolor='#E6E6E6')
```

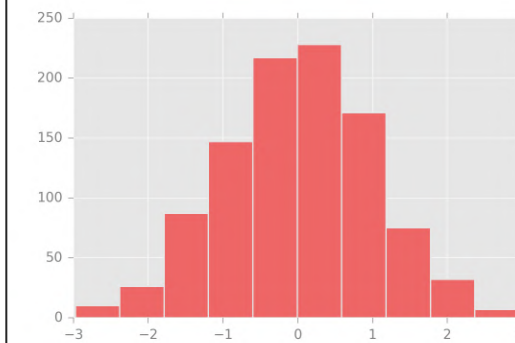
```
plt.rc('lines', linewidth=2)
```



Before Customizing



After Customizing



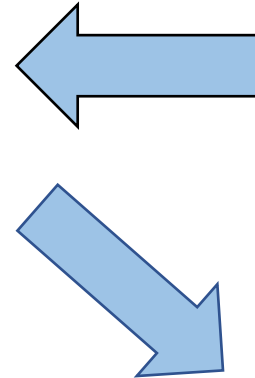
rcParams dictionary variables to revert back settings

Code Snippet

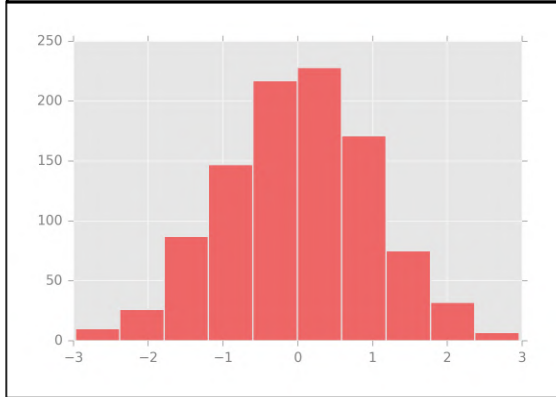
```
mpl.rcParams.update(mpl.rcParamsDefaults)
```

OR

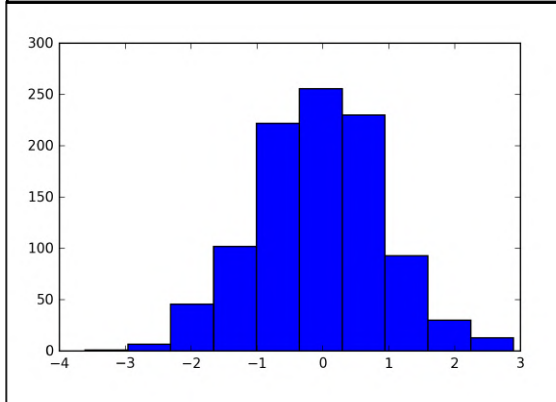
```
mpl.rcParams.update(mpl.rcParamsOrig)
```



Before Reverting Back



After restoring defaults



*<https://stackoverflow.com/questions/26899310/python-seaborn-to-reset-back-to-the-matplotlib>

rcParams dictionary variables to revert back settings

Code Snippet

```
mpl.rcParams.update(mpl.rcParamsDefaults)
```

OR

```
mpl.rcParams.update(mpl.rcParamsOrig)
```

Best Practices

```
import matplotlib as mpl
```

```
my_params = mpl.rcParams
```

```
# apply some change to the rcparams here
```

```
mpl.rcParams.update(my_params)
```

Store current rcParams as a variable prior to any change.

Do some changes in rcParams, plot using new settings

Restore the rcParams with the earlier stored variable.

*<https://stackoverflow.com/questions/26899310/python-seaborn-to-reset-back-to-the-matplotlib>

STYLE SHEETS

`plt.style.use('stylename')` stylename from any of the available style names :
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']

Use `plt.style.available` to get the available style names list

Use `plt.style.context` to apply style temporarily.

[*Matplotlib Release 3.4.2, May 08, 2021, Customizing Matplotlib with style sheets and rcParams, Pg No.84](#)

https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

All Available Stylesheets

Code Snippet for showcasing all styles

```
fig = plt.figure(dpi=100, figsize=(24, 20),
                tight_layout=True)
```

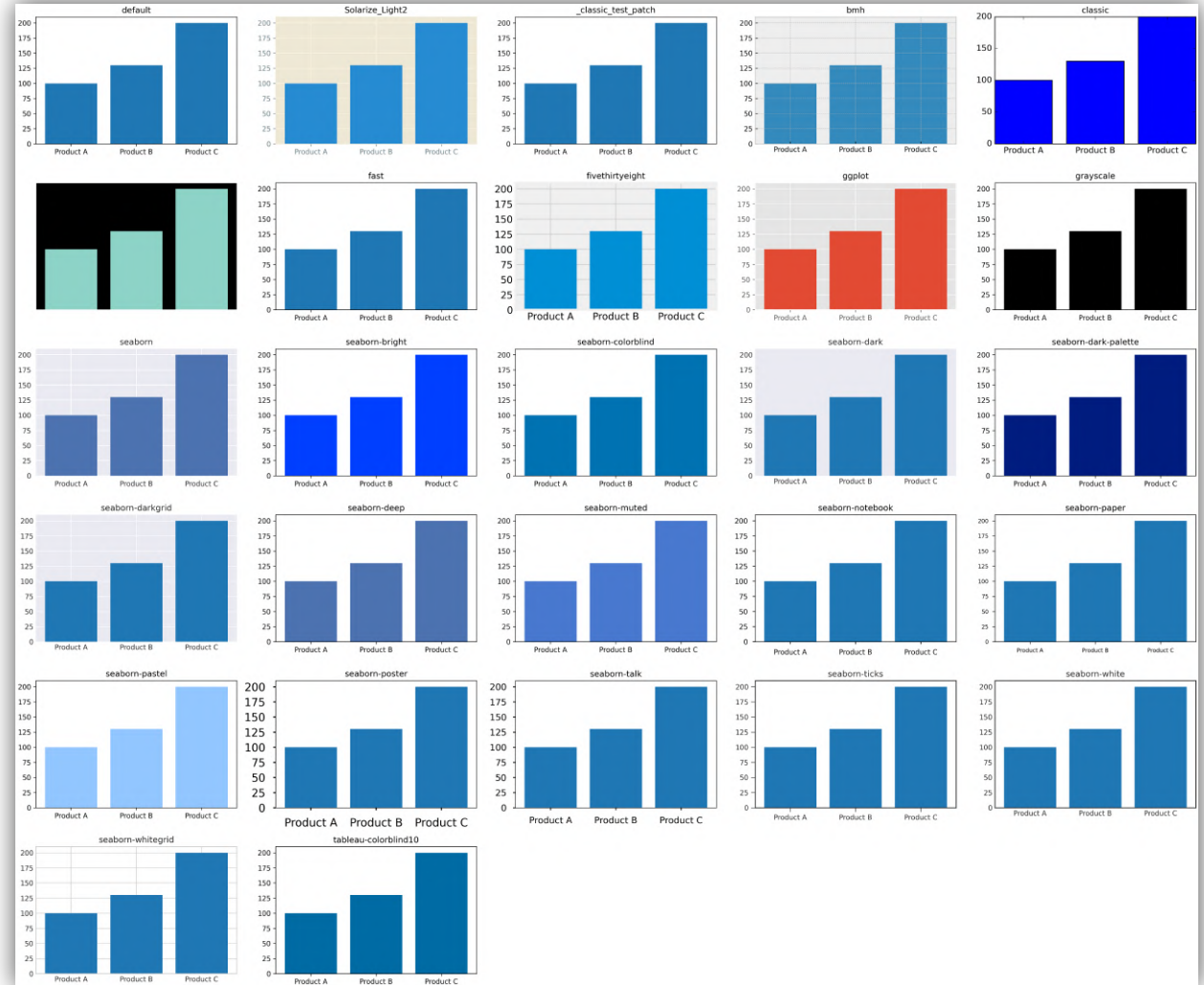
```
available = ['default'] + plt.style.available
x = ['Product A', 'Product B', 'Product C']
y = [100, 130, 200]
```

```
for i, style in enumerate(available):
    with plt.style.context(style):
        ax = fig.add_subplot(6, 5, i + 1)
        ax.bar(x, y)
        ax.set_title(style)
```

```
plt.savefig('All styles.png',
            bbox_inches = "tight",
            pad_inches = 0.05, dpi=300)
```

```
plt.show()
```

Code

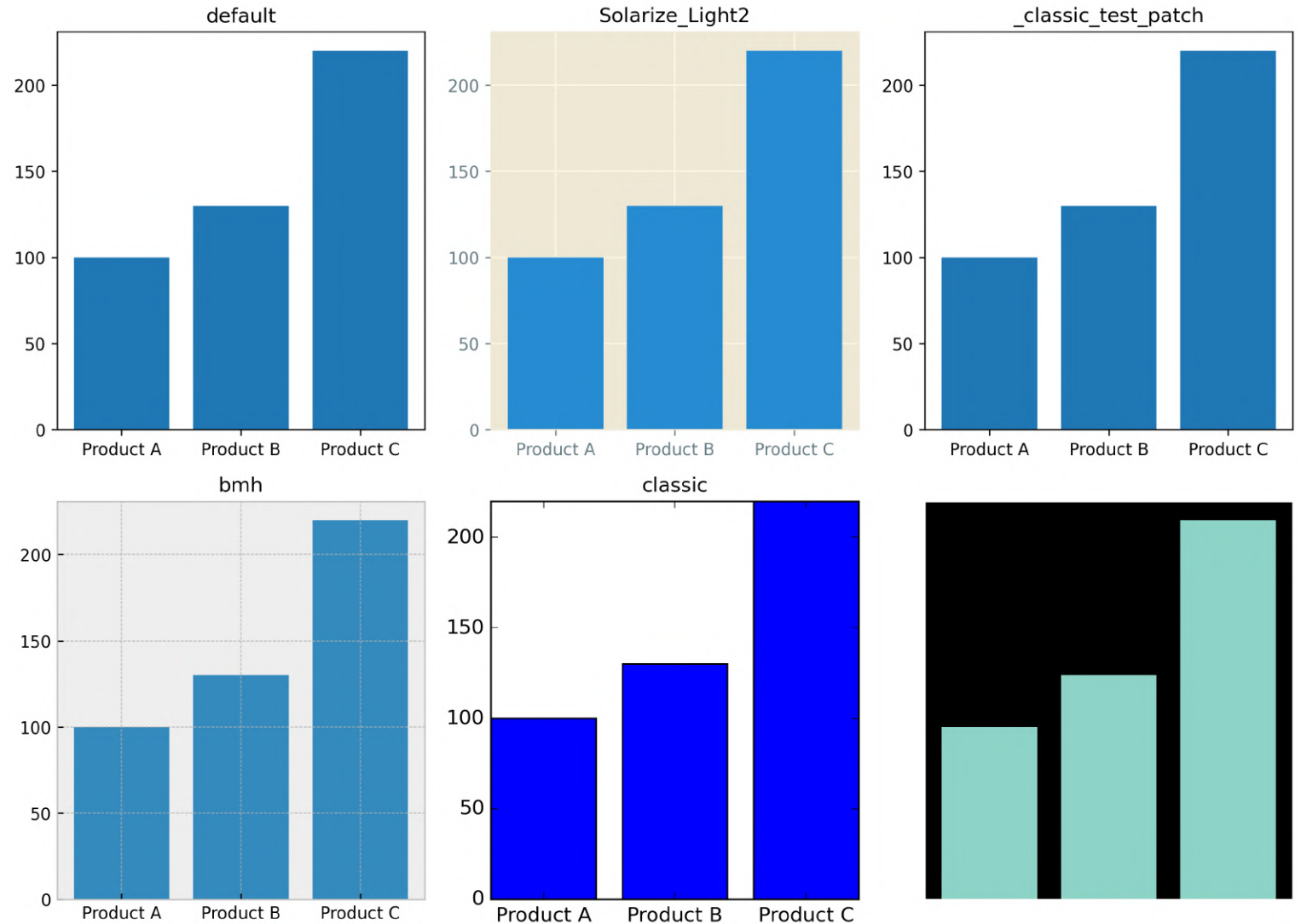


Styles on Right

Styles in this slide

'default'
 'Solarize_Light2'
 '_classic_test_patch'
 'bmh'
 'classic'
 'dark_background '
 'fast'
 'fivethirtyeight'
 'ggplot'
 'grayscale'
 'seaborn'
 'seaborn- bright'
 'seaborn-colorblind'
 'seaborn-dark'
 'seaborn-dark-palette'
 'seaborn-darkgrid'
 'seaborn-deep'
 'seaborn-muted'
 'seaborn-notebook'
 'seaborn-paper'
 'seaborn-pastel'
 'seaborn-poster'
 'seaborn-talk'
 'seaborn-ticks'
 'seaborn-white'
 'seaborn-whitegrid'
 'tableau-colorblind10 '

Zooming in - First set of 6# Stylesheets



Zooming in - Second set of 6# Stylesheets

Styles on Right

Styles in this slide

'default'

'Solarize_Light2'

'_classic_test_patch'

'bmh'

'classic'

'dark_background '

'fast'

'fivethirtyeight'

'ggplot'

'grayscale'

'seaborn'

'seaborn- bright'

'seaborn-colorblind'

'seaborn-dark'

'seaborn-dark-palette'

'seaborn-darkgrid'

'seaborn-deep'

'seaborn-muted'

'seaborn-notebook'

'seaborn-paper'

'seaborn-pastel'

'seaborn-poster'

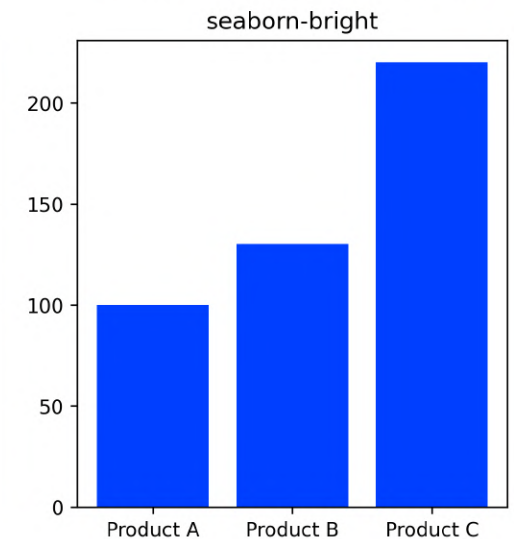
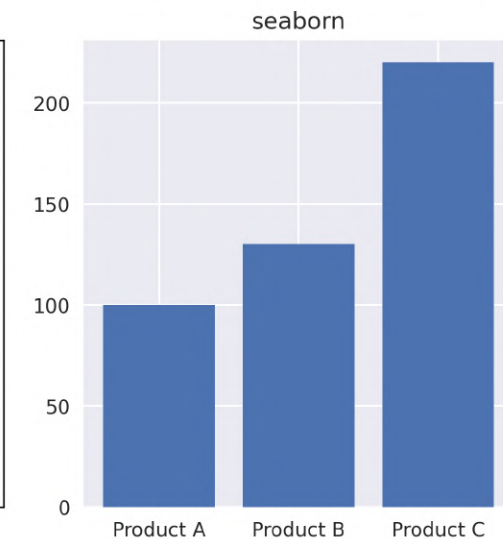
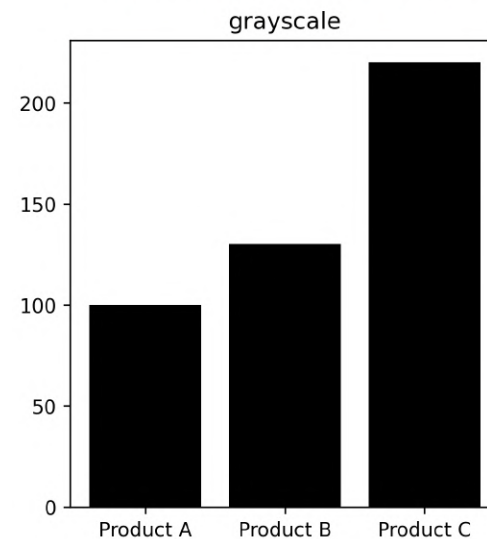
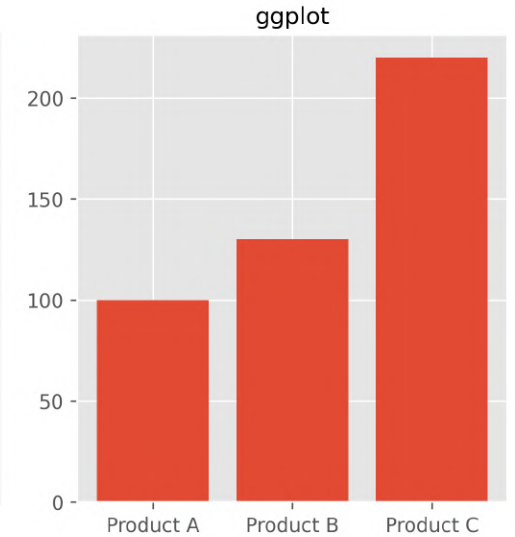
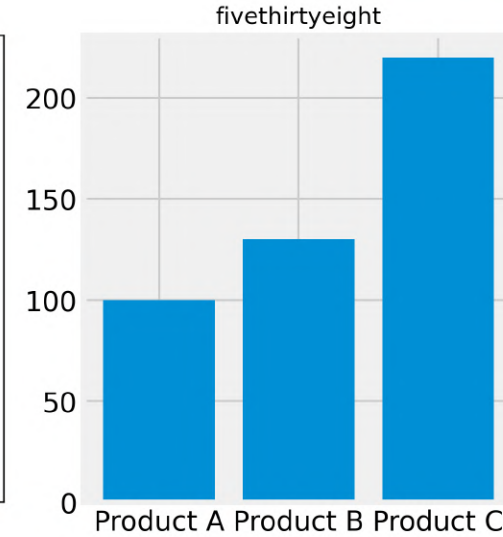
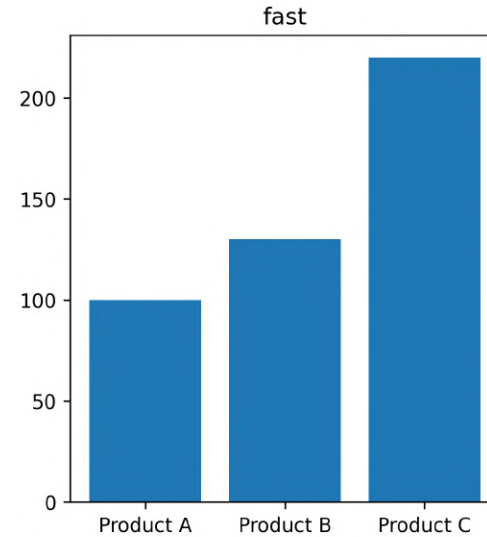
'seaborn-talk'

'seaborn-ticks'

'seaborn-white'

'seaborn-whitegrid'

'tableau-colorblind10 '

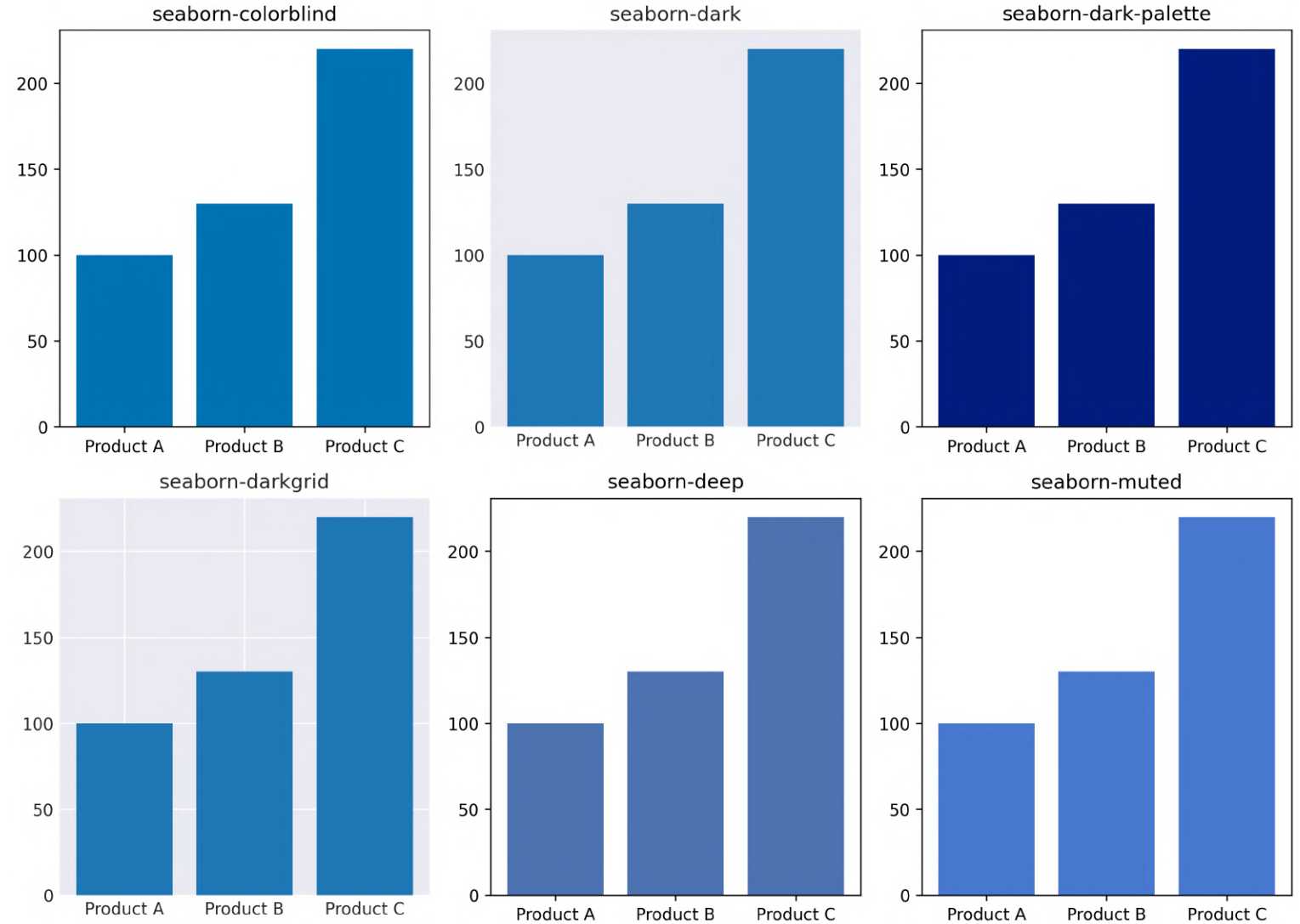


Styles in this slide

'default'
 'Solarize_Light2'
 '_classic_test_patch'
 'bmh'
 'classic'
 'dark_background '
 'fast'
 'fivethirtyeight'
 'ggplot'
 'grayscale'
 'seaborn'
 'seaborn- bright'
 'seaborn-colorblind'
 'seaborn-dark'
 'seaborn-dark-palette'
 'seaborn-darkgrid'
 'seaborn-deep'
 'seaborn-muted'
 'seaborn-notebook'
 'seaborn-paper'
 'seaborn-pastel'
 'seaborn-poster'
 'seaborn-talk'
 'seaborn-ticks'
 'seaborn-white'
 'seaborn-whitegrid'
 'tableau-colorblind10 '

Styles on Right

Zooming in - Third set of 6# Stylesheets

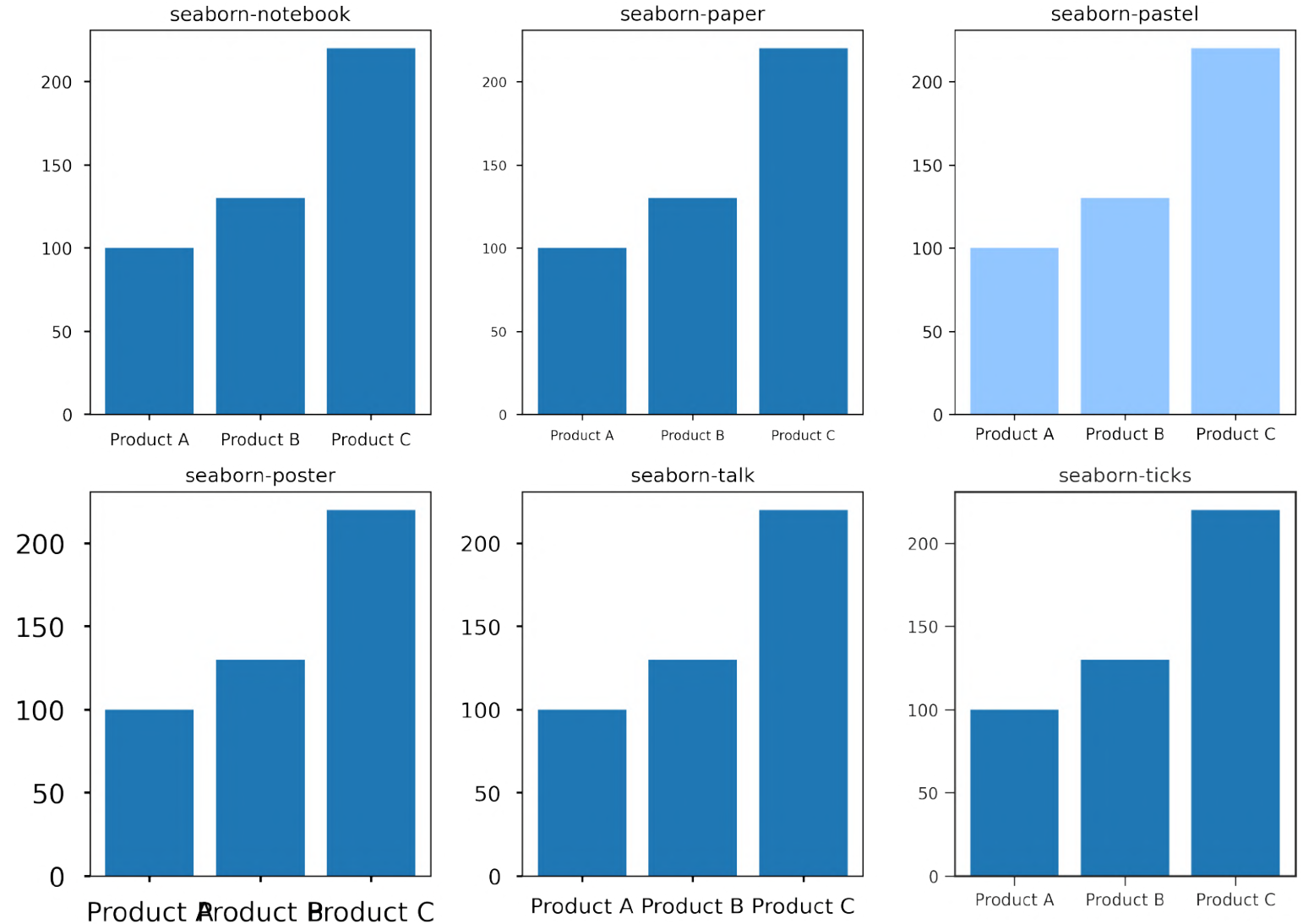


Styles in this slide

'default'
 'Solarize_Light2'
 '_classic_test_patch'
 'bmh'
 'classic'
 'dark_background '
 'fast'
 'fivethirtyeight'
 'ggplot'
 'grayscale'
 'seaborn'
 'seaborn- bright'
 'seaborn-colorblind'
 'seaborn-dark'
 'seaborn-dark-palette'
 'seaborn-darkgrid'
 'seaborn-deep'
 'seaborn-muted'
 'seaborn-notebook'
 'seaborn-paper'
 'seaborn-pastel'
 'seaborn-poster'
 'seaborn-talk'
 'seaborn-ticks'
 'seaborn-white'
 'seaborn-whitegrid'
 'tableau-colorblind10 '

Styles on
Right

Zooming in - Fourth set of 6# Stylesheets



Styles in this slide

'default'

'Solarize_Light2'

'_classic_test_patch'

'bmh'

'classic'

'dark_background '

'fast'

'fivethirtyeight'

'ggplot'

'grayscale'

'seaborn'

'seaborn- bright'

'seaborn-colorblind'

'seaborn-dark'

'seaborn-dark-palette'

'seaborn-darkgrid'

'seaborn-deep'

'seaborn-muted'

'seaborn-notebook'

'seaborn-paper'

'seaborn-pastel'

'seaborn-poster'

'seaborn-talk'

'seaborn-ticks'

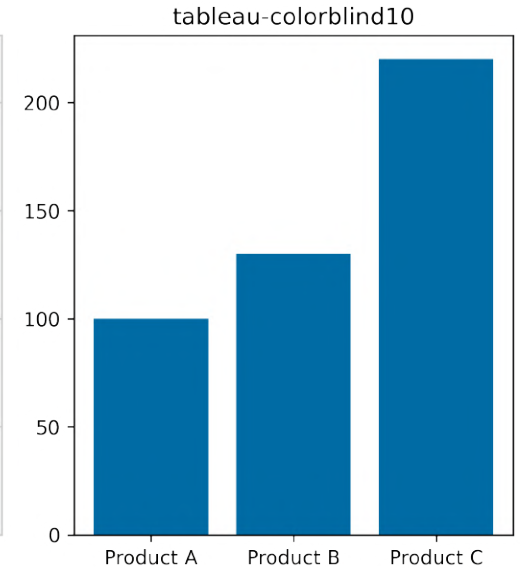
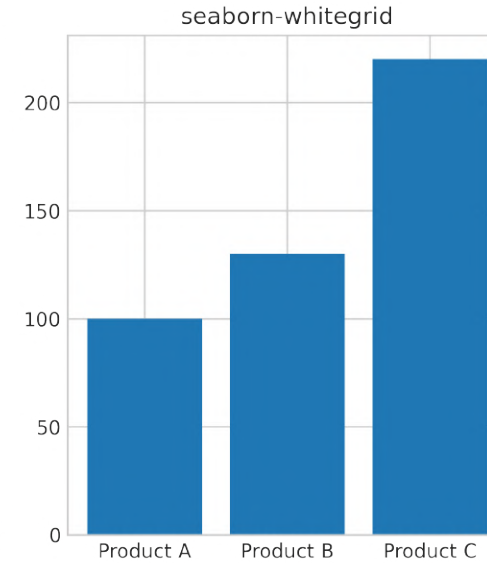
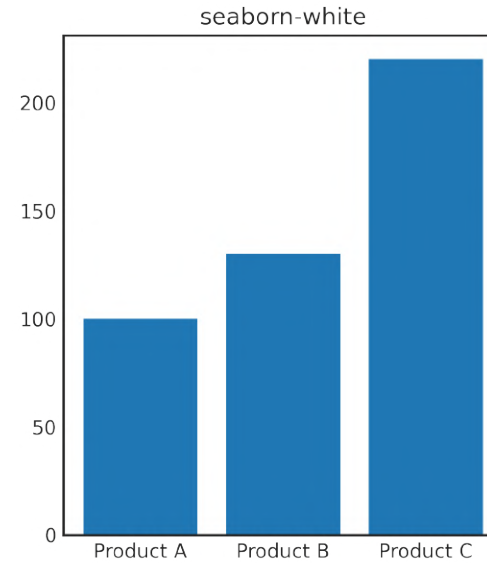
'seaborn-white'

'seaborn-whitegrid'

'tableau-colorblind10 '

**Styles on
Right**

Zooming in - Fifth set of 3# Stylesheets



OO INTERFACE TO MATPLOTLIB

- ✓ A Stateless Interface
- ✓ Does not rely on the concept of current axes and current figure
- ✓ Explicitly object instances are named and methods invoked
- ✓ Anatomy of a figure and the Hierarchy of artists is respected

ENTRY POINT

- ✓ Consider Pyplot Interface as a brief interlude
- ✓ It's a good starting point
- ✓ Hands-on and convenient

INFLECTION POINT

- ✓ But to gain finely grained control over every aspect of plot
- ✓ OO is the way

ACTIONABLES

- ✓ We will take it step by step
- ✓ Review matplotlib landscape
- ✓ Build on our learnings
- ✓ Partly incremental and partly undertake few leaps

BROADER GOAL

- ✓ This is an attempt to bring a structure, build intuition to approach the plotting into breakable tasks
- ✓ The goal is also to be the go-to reference guide for the major part of business analytics related EDA

SPECIFIC OUTCOMES

- ✓ Understand Matplotlib API
- ✓ Understand OO interface and settle on mix of OO and pyplot
- ✓ Eventually use OO interface seamlessly with Seaborn, Pandas, GeoPandas

OO APPROACH – A COMPLETE RUNDOWN

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes Helper Methods to access x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Helper Methods (Accessors) to access artists through attributes.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend and Set Title to the Axes

PREPARING THE DATA

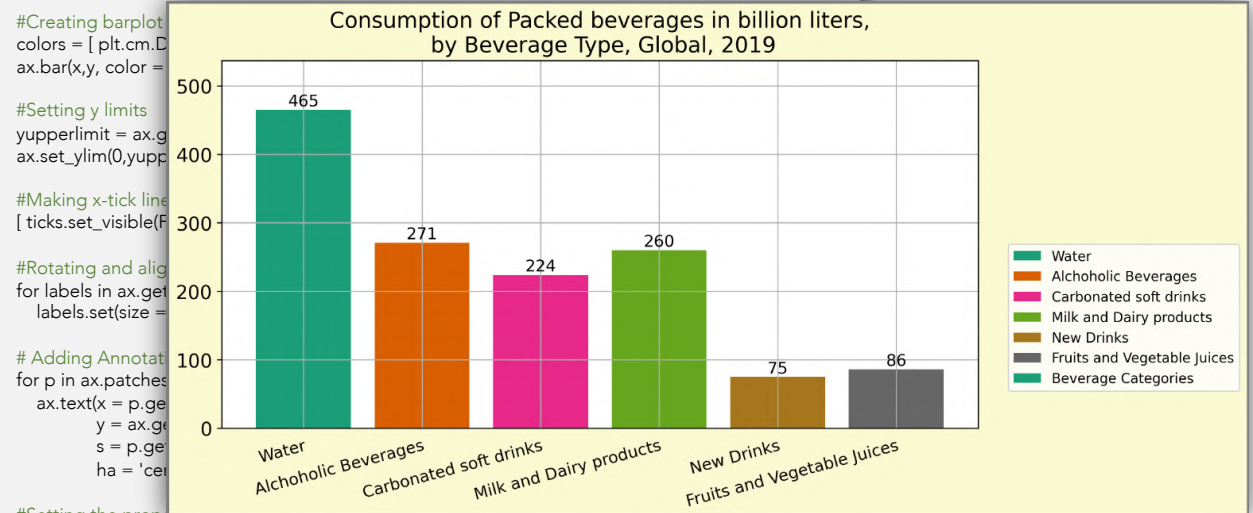
```
data = {'Water':465,  
        'Alcoholic Beverages':271,  
        'Carbonated soft drinks': 224,  
        'Milk and Dairy products': 260,  
        'New Drinks': 75,  
        'Fruits and Vegetable Juices': 86}
```

```
x = list(data.keys())  
y = list(data.values())
```

CODE SCRIPTING

```
import matplotlib.pyplot as plt  
%matplotlib inline  
  
#Creating figure and axes instances  
fig, ax = plt.subplots(figsize = (10,5), frameon = True)  
  
#Creating barplot  
colors = [ plt.cm.DS3_COLORS[i] for i in range(6)]  
ax.bar(x,y, color = colors)  
  
#Setting y limits  
yupperlimit = ax.get_ylim()[1]  
ax.set_ylim(0,yupperlimit)  
  
#Making x-tick labels  
ax.set_xticks(x)  
ax.set_xticklabels(x)  
  
#Rotating and aligning labels  
for p in ax.patches:  
    ax.text(x = p.get_x() + p.get_width()/2,  
            y = ax.get_ylim()[1] - 10,  
            s = p.get_label(),  
            ha = 'center',  
            va = 'bottom',  
            align = 'center',  
            fontweight = 'bold',  
            size = 12)  
  
#Setting the properties of the figure and axes  
ax.patch.set(facecolor = 'white')  
fig.patch.set(facecolor = 'white', edgecolor = 'grey', lw = 4)  
ax.grid(True)  
  
#Setting axes title  
StrTitle = 'Consumption of Packed beverages in billion liters,\n by Beverage Type, Global, 2019'  
ax.set_title(StrTitle, size = 16)  
  
#Adding Legend labels and handles  
[p.set_label(x[i]) for i,p in zip(range(len(x)), ax.patches)]  
ax.legend(bbox_to_anchor = (1.04,5), loc="upper left",borderaxespad=0)  
  
#Saving the figure locally  
fig.savefig('Consumption_2.png', dpi=300, format='png', bbox_inches='tight')
```

PLOTTING OUTPUT



OO APPROACH – PREPARING THE DATA

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes Helper Methods to access x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Helper Methods (Accessors) to access artists through attributes.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend and Set Title to the Axes

PREPARING THE DATA

```
data = {'Water':465,  
        'Alcoholic Beverages':271,  
        'Carbonated soft drinks': 224,  
        'Milk and Dairy products': 260,  
        'New Drinks': 75,  
        'Fruits and Vegetable Juices': 86}  
x = list(data.keys())  
y = list(data.values())
```


OO APPROACH – CODE SCRIPTING

1 Create a Figure instance

2 Use Figure to create one or more Axes or Subplot

3 Use Axes Helper Methods to add artists to respective containers within the Axes object.

4 Use Axes Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5 Use Axes Helper Methods (Accessors) to access artists through attributes.

6 Set/Modify the Artist properties after accessing the artists.

7 Add Legend and Set Title to the Axes

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#Creating figure and axes instances
fig, ax = plt.subplots(figsize = (10,5), frameon = True)
```

```
#Creating barplot using axes.bar() method
colors = [ plt.cm.Dark2(x) for x in np.linspace(0, 1, len(x))] #Accessing colors from colormap
ax.bar(x,y, color = colors)
```

```
#Setting y limits
yupperlimit = ax.get_ylim()[1]*1.1
ax.set_ylim(0,yupperlimit)
```

```
#Making x-tick lines invisible
[ ticks.set_visible(False) for ticks in ax.get_xticklines() ]
```

```
#Rotating and aligning the xtick labels
for labels in ax.get_xticklabels():
    labels.set(size = 12, rotation = 15, rotation_mode = 'anchor', ha = 'right')
```

```
# Adding Annotations to place data labels
for p in ax.patches:
    ax.text(x = p.get_x()+p.get_width()/2,
           y = ax.get_ylim()[1]*0.01+ p.get_height(),
           s = p.get_height(),
           ha = 'center', size = 12)
```

```
#Setting the properties of rectangular patch in axes and figure
ax.patch.set(facecolor = 'white') #OO style to set properties
plt.setp(fig.patch, facecolor = 'white', edgecolor = 'grey', lw = 4). #Pyplot style to set properties
ax.grid(True)
```

```
# Setting axes title
StrTitle = 'Consumption of Packed beverages in billion liters,\n by Beverage Type, Global, 2019'
ax.set_title(StrTitle, size = 16)
```

```
# Adding Legend labels and handles
[p.set_label(x[i]) for i,p in zip(range(len(x)), ax.patches)]
ax.legend(bbox_to_anchor = (1.04,.5), loc="upper left",borderaxespad=0)
```

```
fig.savefig('Consumption_2.png', dpi=300, format='png', bbox_inches='tight')
```

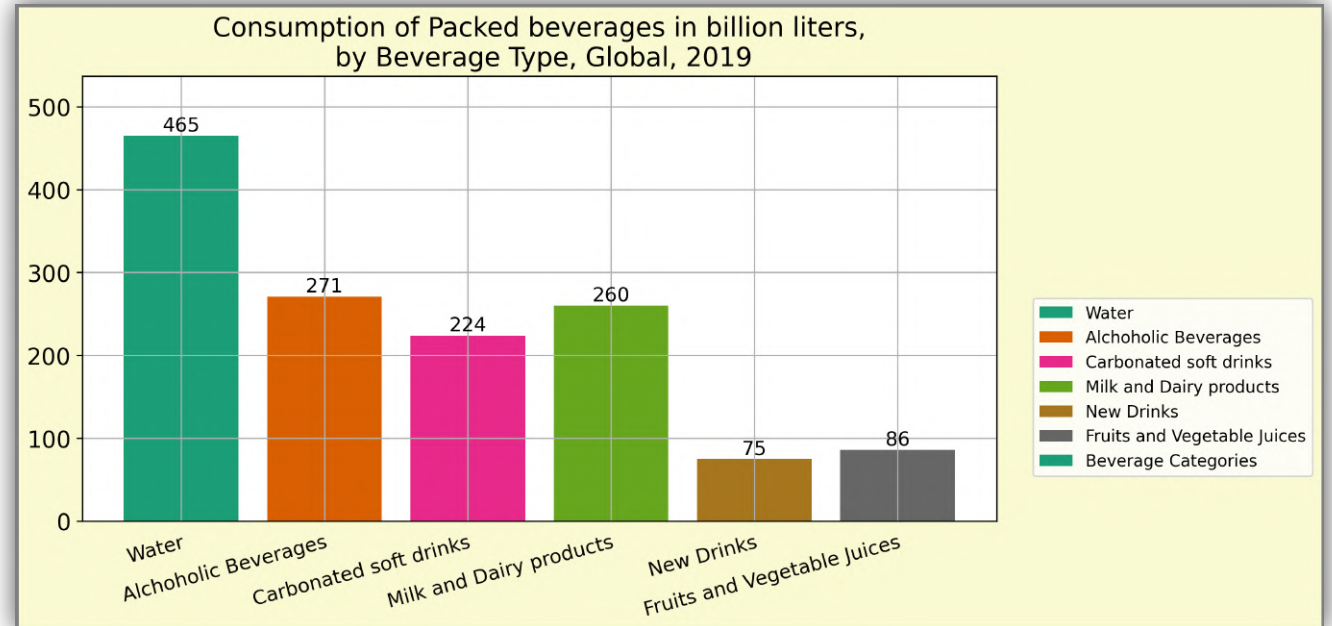
Check slide



OO APPROACH – PLOTTING OUTPUT

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Helper Methods (Accessors) to access artists through attributes.
- 6 Set/Modify the Artist properties after accessing the artists.
- 7 Add Legend and Set Title to the Axes

PLOTTING OUTPUT

[Check slide](#)

OO APPROACH – A DEEP DIVE

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

CREATING FIGURE AND AXES

Method	Description	Page No.
<code>fig, ax = plt.subplots()</code>	Instantiate a figure and axes	2650
<code>fig = plt.figure()</code>	Instantiate a figure	2528

METHODS TO CREATE NEW SUBPLOTS

<code>fig.subplots(nrows, ncols)</code>	Primary function to create and place all axes on the figure at once by dividing figure into grids	2117
<code>fig.add_subplot((nrows, ncols, index)</code>	Figure method to add axes by specifying the index in a grid layout pattern	2077, 2117
<code>fig.add_gridspec(nrows, ncols)</code>	Figure method to return gridspec that specifies the geometry of the grid that a subplot will be placed.	2076, 2174
<code>fig.add_subplot(SubplotSpec)</code>	Figure method to add axes by specifying the location of subplot in a GridSpec	2076 2174
<code>fig.add_axes(rect)</code>	Figure method to add axes at any location within the Figure	2074
<code>fig.delaxes(axes object)</code>	Figure method to delete an axes	
<code>ax.remove()</code>	Axes method to remove an axes	

CUSTOMIZE SUBPLOT

<code>fig.subplots_adjust(left, bottom, right, top, wspace, hspace)</code>	Adjust the subplot layout parameters.	2119
<code>tight_layout(pad, h_pad, w_pad, rect)</code>	Adjust the padding between and around subplots.	2124
<code>ax.set_anchor(anchor)</code>	Define the anchor location. Anchor values can be 'C', N, NW,W, SW,S, SE,E,NE	
<code>fig.align_labels()</code> <code>fig.align_xlabels()</code> <code>fig.align_ylabels()</code>	Align the labels of subplots in the same subplot column if label alignment is being done automatically	2080-2082

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

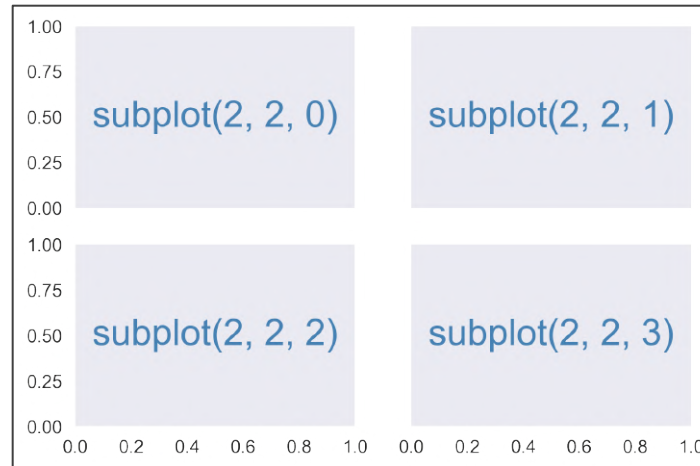
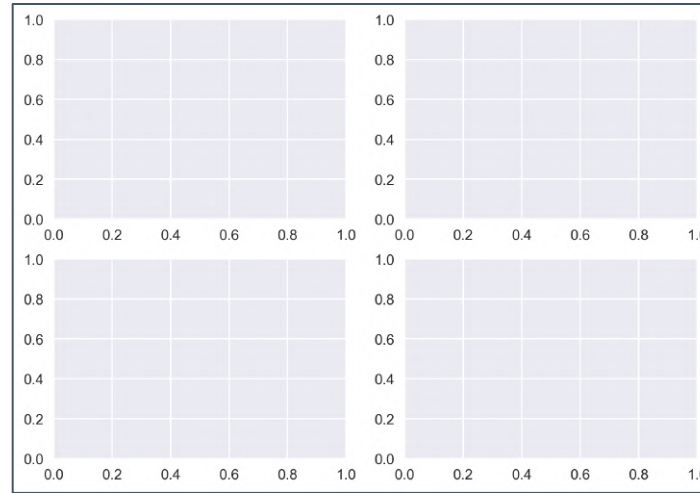
Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes



DEMO OF FIG.SUBPLOTS()

#Using fig.subplots to place multiple axes in the figure

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
%matplotlib inline
```

```
fig = plt.figure()
ax = fig.subplots(2,2)
```

DEMO OF FIG.SUBPLOTS() AND ENUMERATE

#Using fig.subplots

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
%matplotlib inline
```

```
rows, cols = 2,2
fig = plt.figure()
ax = fig.subplots(rows,cols,
                  sharex = True,sharey = True)
```

#Adding text by looping through all the subplots/axes

```
ax_iter = ax.flatten()
for i,axes in enumerate(ax_iter):
    axes.text(0.5, 0.5,
              f'subplot{rows,cols,i}',
              ha='center', va='center',
              size=20, color="steelblue")
axes.grid(False)
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

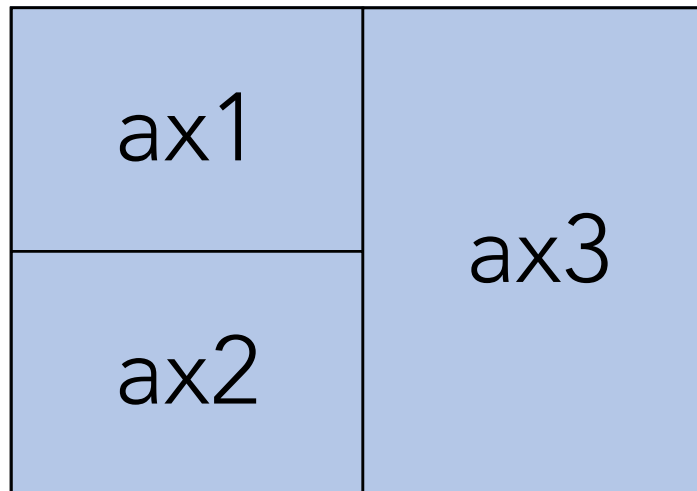
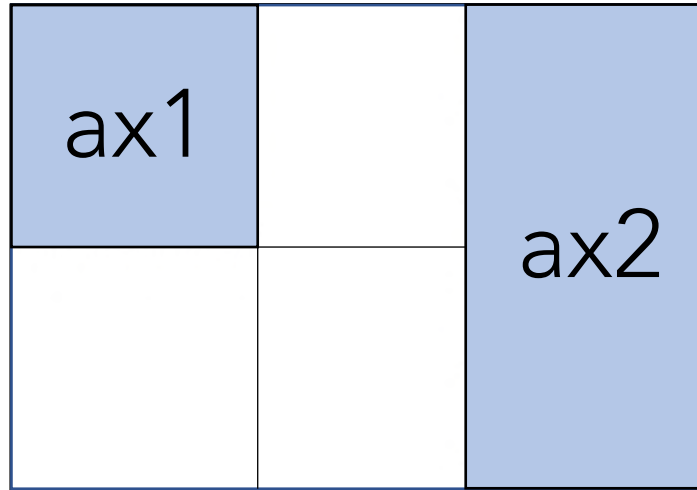
Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes



```
#Using add_subplots to add axes
import matplotlib.pyplot as plt
plt.style.use('seaborn')
%matplotlib inline
```

```
fig = plt.figure()
ax1 = fig.add_subplot(2,3,1)
ax2 = fig.add_subplot(2, 3, (3,6))
```

Note : Indexing is 1 based.

```
#Using Gridspec with add_subplots
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
```

```
fig = plt.figure(figsize=(12, 8))
spec = gridspec.GridSpec(nrows = 2,
                        ncols = 2,
                        wspace = 0.2,
                        hspace=0.1)
```

```
ax1 = fig.add_subplot(spec[0, 0])
ax2 = fig.add_subplot(spec[1, 0])
ax3 = fig.add_subplot(spec[:, 1])
```

Note : Indexing is 0 based.

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

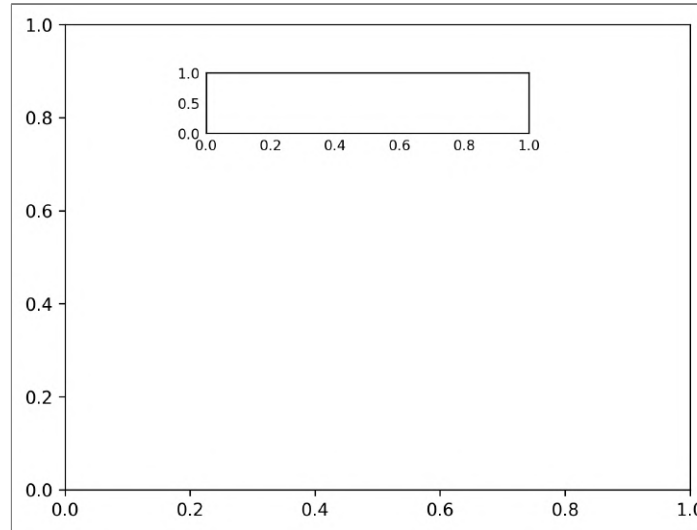
Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes



```
#Using fig.add_axes() to add subplots
import matplotlib.pyplot as plt
plt.rcParams()
fig = plt.figure()
```

```
ax1 = fig.subplots()
ax2 = fig.add_axes([0.3,0.7,0.4,0.1])
ax2.tick_params(labelsize=8, length=0)
```

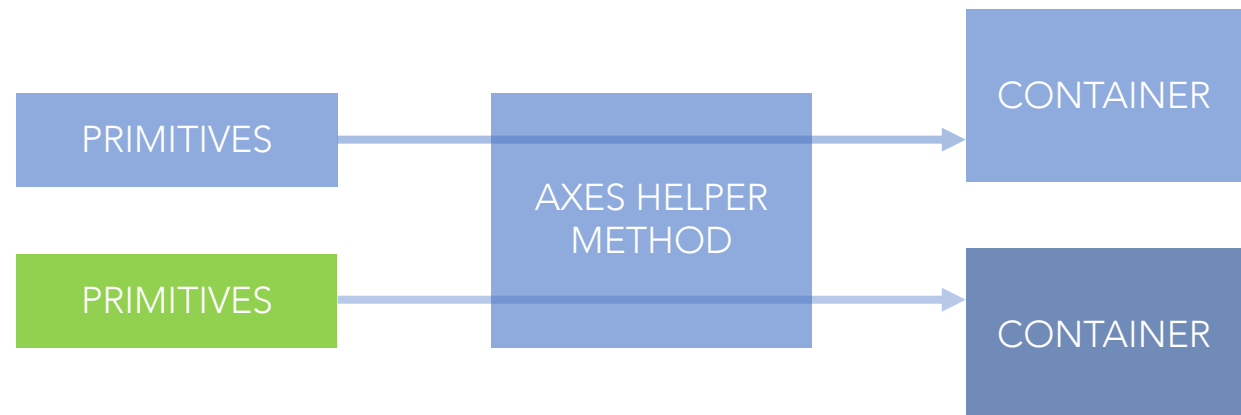
*Note : This method used to Add an axes at position `rect[left, bottom,width, height]` where all quantities are in fractions of figure width and height.

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES HELPER METHODs TO CREATE PRIMITIVES

Axes helper method	Artist	Container
annotate - text annotations	Annotation	ax.texts
bar - bar charts	Rectangle	ax.patches and Barcontainer
errorbar - error bar plots	Line2D and Rectangle	ax.lines and ax.patches
fill - shared area	Polygon	ax.patches
hist - histograms	Rectangle	ax.patches
imshow - image data	AxesImage	ax.images
legend - Axes legends	Legend	ax.legend()
plot - xy plots	Line2D	ax.lines
scatter - scatter charts	PolyCollection	ax.collections
text - text	Text	ax.texts

- ✓ To get a list of all containers in the axes, use `ax.containers`
- ✓ To get list of all artists in the axes, use `ax.get_children()`



*Check [Artist Tutorial](#)

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES METHODS TO CREATE DIFFERENT PLOTS AND TEXT (1/3)

Category	Method	Description	Page No.
Basic	<code>Axes.plot</code>	Plot y versus x as lines and/or markers.	1241
	<code>Axes.errorbar</code>	Plot y versus x as lines and/or markers with attached errorbars.	1241
	<code>Axes.scatter</code>	A scatter plot of y vs.	1241
	<code>Axes.plot_date</code>	Plot coercing the axis to treat floats as dates.	1241
	<code>Axes.step</code>	Make a step plot.	1241
	<code>Axes.loglog</code>	Make a plot with log scaling on both the x and y axis.	1241
	<code>Axes.semilogx</code>	Make a plot with log scaling on the x axis.	1241
	<code>Axes.semilogy</code>	Make a plot with log scaling on the y axis.	1241
	<code>Axes.fill_between</code>	Fill the area between two horizontal curves.	1241
	<code>Axes.fill_betweenx</code>	Fill the area between two vertical curves.	1241
	<code>Axes.bar</code>	Make a bar plot.	1241
	<code>Axes.barh</code>	Make a horizontal bar plot.	1241
	<code>Axes.bar_label</code>	Label a bar plot.	1241
	<code>Axes.stem</code>	Create a stem plot.	1242
	<code>Axes.eventplot</code>	Plot identical parallel lines at the given positions.	1242
	<code>Axes.pie</code>	Plot a pie chart.	1242
	<code>Axes.stackplot</code>	Draw a stacked area plot.	1242
	<code>Axes.broken_barh</code>	Plot a horizontal sequence of rectangles.	1242
	<code>Axes.vlines</code>	Plot vertical lines at each x from ymin to ymax.	1242
	<code>Axes.hlines</code>	Plot horizontal lines at each y from xmin to xmax.	1242
	<code>Axes.fill</code>	Plot filled polygons.	1242

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES METHODS TO CREATE DIFFERENT PLOTS AND TEXT (2/3)

Category	Method	Description	Page No.
Spans	Axes.axhline	Add a horizontal line across the axis.	1296
	Axes.axhspan	Add a horizontal span (rectangle) across the Axes.	1296
	Axes.axvline	Add a vertical line across the Axes.	1296
	Axes.axvspan	Add a vertical span (rectangle) across the Axes.	1296
	Axes.axline	Add an infinitely long straight line.	1296
Spectral	Axes.xcorr	Plot the cross correlation between x and y.	1307
Statistics	Axes.boxplot	Make a box and whisker plot.	1334
	Axes.violinplot	Make a violin plot.	1334
	Axes.violin	Drawing function for violin plots.	1334
	Axes.bxp	Drawing function for box and whisker plots.	1334
Binned	Axes.hexbin	Make a 2D hexagonal binning plot of points x, y.	1344
	Axes.hist	Plot a histogram.	1344
	Axes.hist2d	Make a 2D histogram plot.	1344
	Axes.stairs	A stepwise constant function as a line with bounding edges or a filled plot.	1344
Text and annotations	Axes.annotate	Axes.annotate(self, text, xy, *args, **kwargs) Annotate the point xy with text text.	1399
	Axes.text	Annotate the point xy with text text.	1399
	Axes.table	Add text to the Axes.	1399
	Axes.arrow	Add a table to an Axes.	1399
	Axes.inset_axes	Add an arrow to the Axes.	1399
	Axes.indicate_inset	Add a child inset Axes to this existing Axes.	1399
	Axes.indicate_inset_zoom	Add an inset indicator to the Axes.	1399
	Axes.secondary_xaxis	Add an inset indicator rectangle to the Axes based on the axis limits for an inset_ax and draw connectors between inset_ax and the rectangle.	1399
	Axes.secondary_yaxis	Add a second x-axis to this Axes.	1399

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

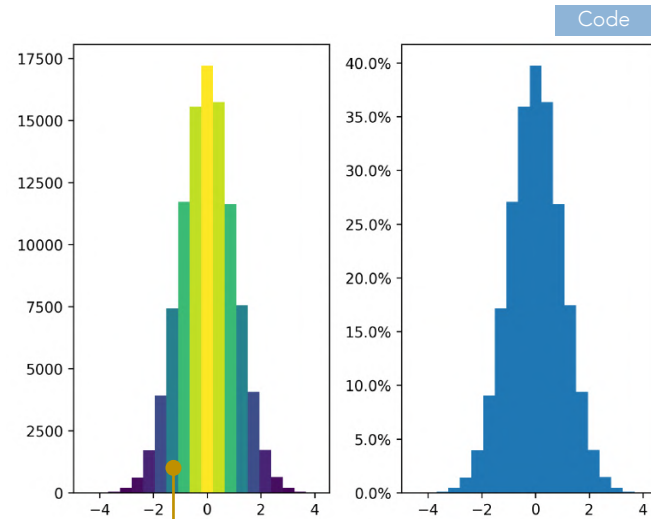
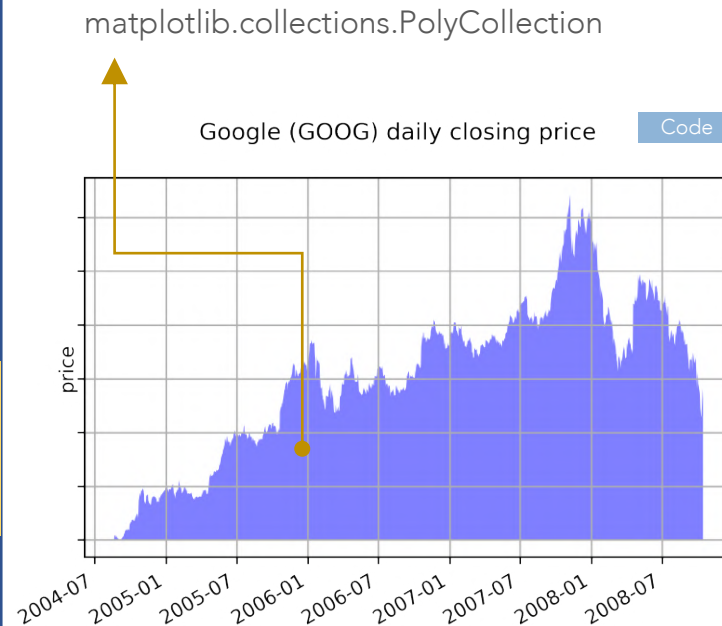
Add Legend, Annotations if any and Set Title to the Axes

AXES METHODS TO CREATE DIFFERENT PLOTS AND TEXT (3/3)

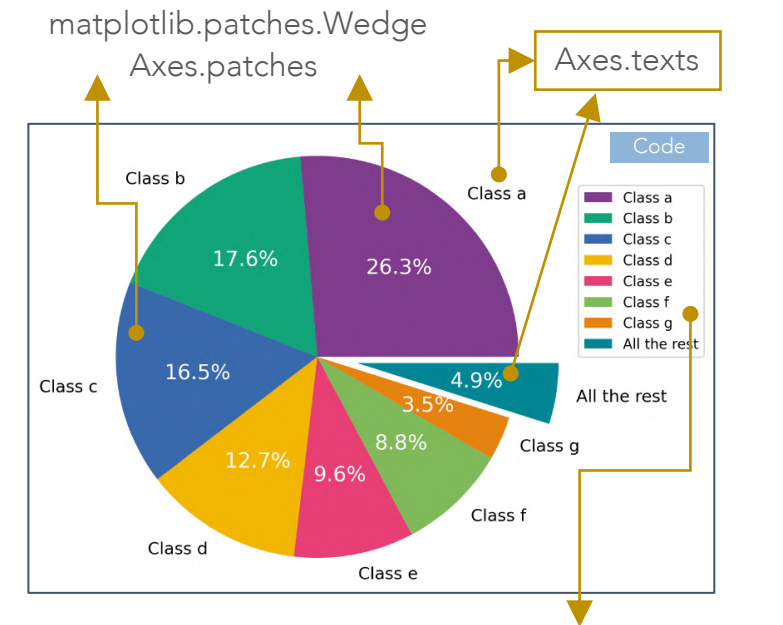
Category	Method	Description	Page No.
2D Arrays	Axes.imshow	Display data as an image, i.e., on a 2D regular raster.	1369
	Axes.matshow	Plot the values of a 2D matrix or array as color-coded image.	1369
	Axes.pcolor	Create a pseudocolor plot with a non-regular rectangular grid.	1369
	Axes.pcolorfast	Create a pseudocolor plot with a non-regular rectangular grid.	1369
	Axes.pcolormesh	Create a pseudocolor plot with a non-regular rectangular grid.	1369
	Axes.spy	Plot the sparsity pattern of a 2D array.	1369

https://matplotlib.org/stable/api/axes_api.html?highlight=axes#module-matplotlib.axes

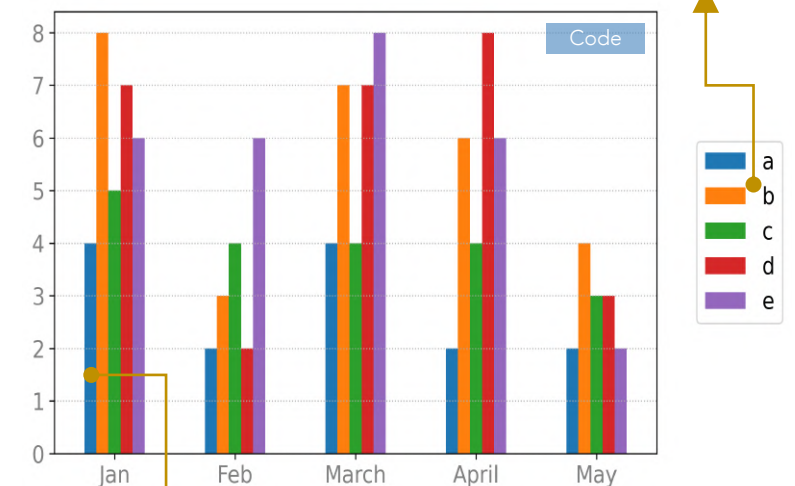
- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes



matplotlib.patches.Rectangle
Axes.patches



matplotlib.legend.Legend



matplotlib.patches.Rectangle
Axes.patches
Barcontainer

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

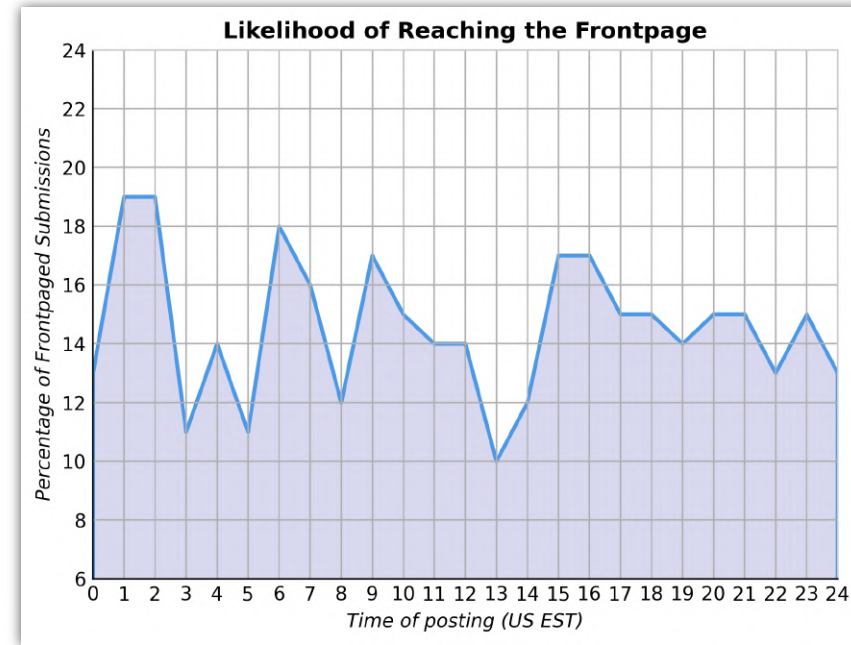
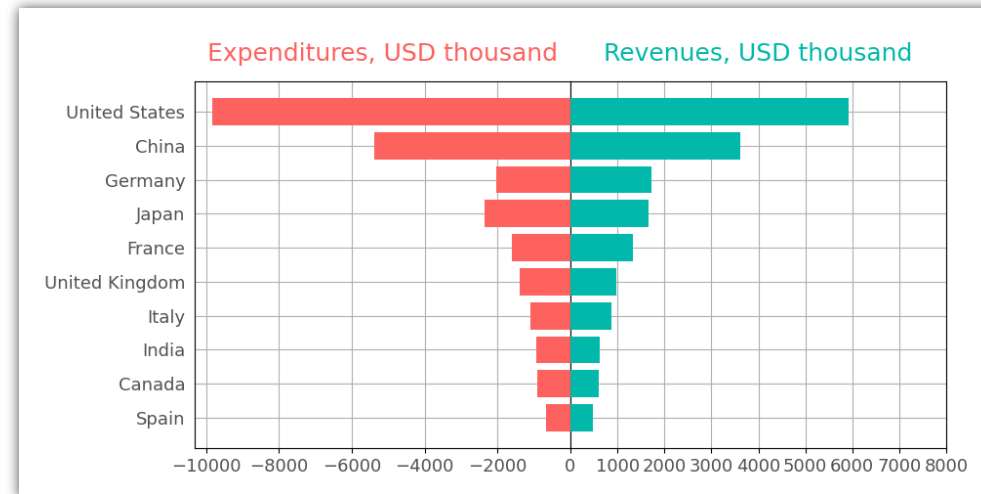
Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes


[Code](#)

[Code](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

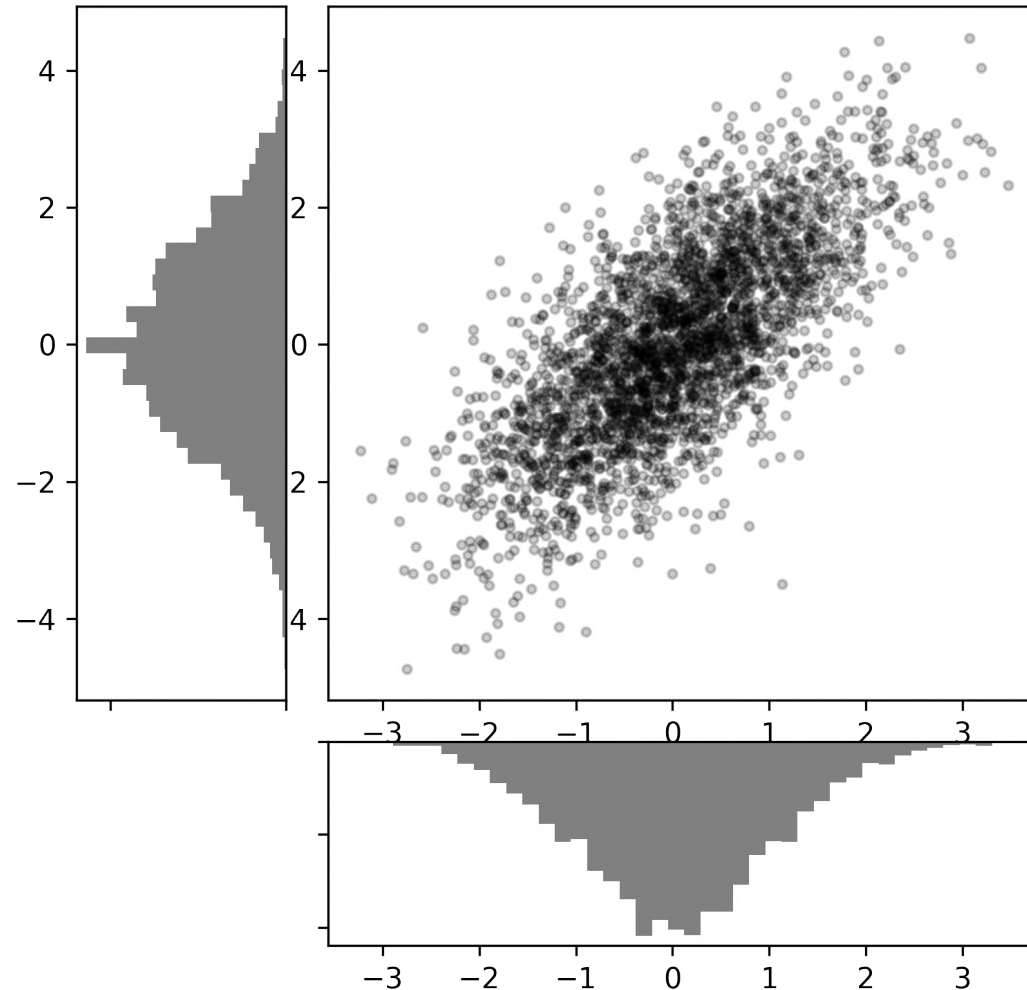
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

VISUALIZING MULTI DIMENSIONAL DISTRIBUTIONS

[Code](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

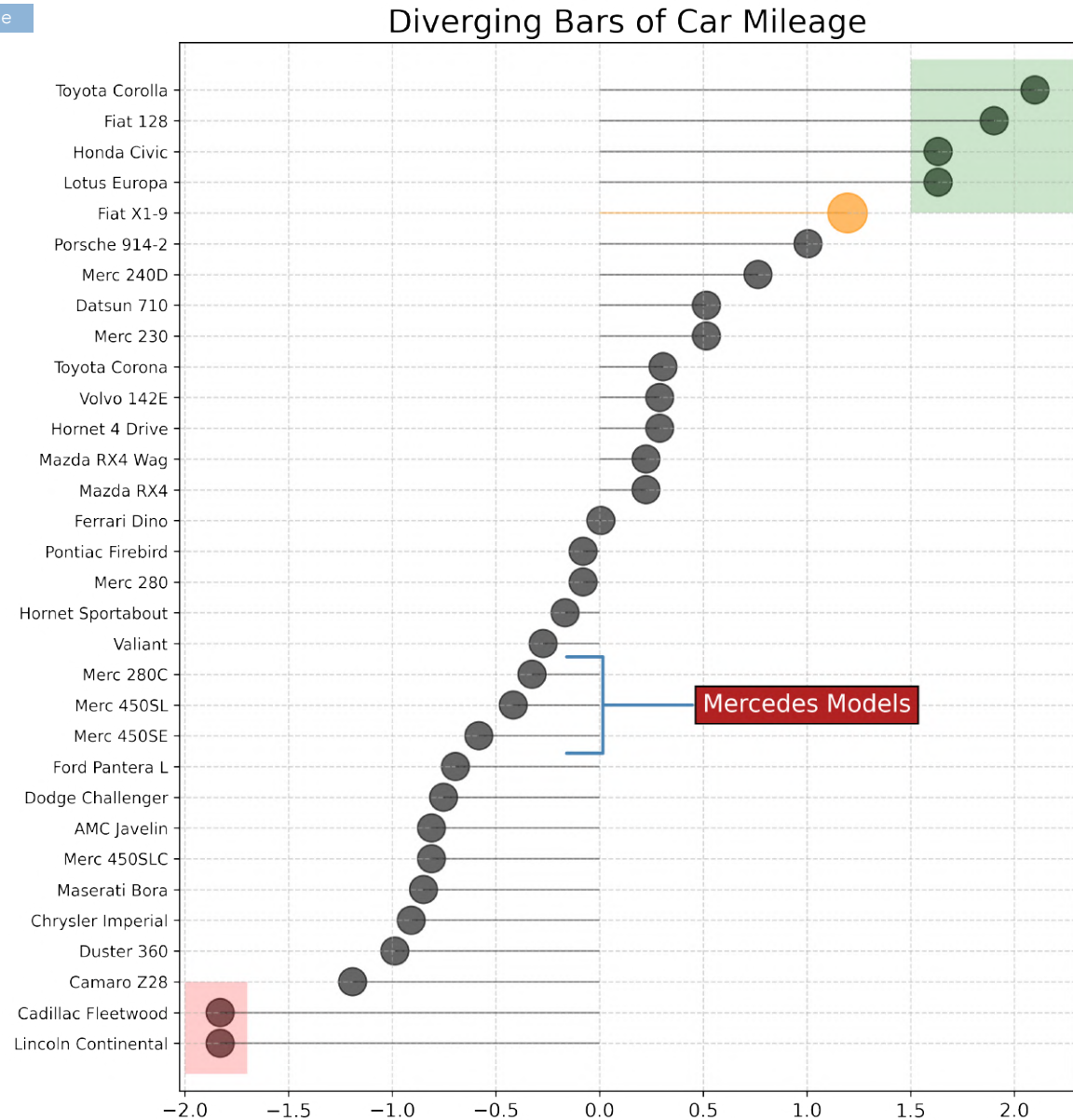
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

Code



Source : <https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/#13.-Diverging-Lollipop-Chart-with-Markers>

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TICK TICK LABELS GRIDLINES

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES HELPER METHODS FOR APPEARANCE, AXIS LIMITS			
Category	Method	Description	Page No.
Appearance	✓ Axes.axis	Convenience method to get or set some axis properties.	1432
	✓ Axes.set_axis_off	Turn the x- and y-axis off.	1432
	✓ Axes.set_axis_on	Turn the x- and y-axis on.	1432
	✓ Axes.set_frame_on	Set whether the axes rectangle patch is drawn.	1432
	Axes.get_frame_on	Get whether the axes rectangle patch is drawn.	1432
	✓ Axes.set_axisbelow	Set whether axis ticks and gridlines are above or below most artists.	1432
	Axes.get_axisbelow	Get whether axis ticks and gridlines are above or below most artists.	1432
	✓ Axes.grid	Configure the grid lines.	1432
	Axes.get_facecolor	Get the facecolor of the Axes.	1432
	✓ Axes.set_facecolor	Set the facecolor of the Axes.	1432
Property Cycle	✓✓ Axes.set_prop_cycle	Set the property cycle of the Axes.	1440
Axis / limits	✓ Axes.get_xaxis	Return the XAxis instance.	1442
	✓ Axes.get_yaxis	Return the YAxis instance.	1442
Axis limits and direction	Axes.invert_xaxis	Invert the x-axis.	1442
	Axes.xaxis_inverted	Return whether the xaxis is oriented in the "inverse" direction.	1442
	✓ Axes.invert_yaxis	Invert the y-axis.	1442
	Axes.yaxis_inverted	Return whether the yaxis is oriented in the "inverse" direction.	1442
	✓✓ Axes.set_xlim	Set the x-axis view limits.	1442
	✓ Axes.get_xlim	Return the x-axis view limits.	1442
	✓✓ Axes.set_ylim	Set the y-axis view limits.	1442
	✓ Axes.get_ylim	Return the y-axis view limits.	1442

✓ indicates frequency and importance of usage of the method

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES HELPER METHODS FOR TICKS AND TICK LABELS, GRID LINES			
Category	Method	Description	Page No.
Ticks and Tick Labels	Axes.set_xticks	Set the xaxis' tick locations.	1491
	Axes.get_xticks	Return the xaxis' tick locations in data coordinates.	1491
	✓ Axes.set_xticklabels	Set the xaxis' labels with list of string labels.	1491
	✓✓ Axes.get_xticklabels	Get the xaxis' tick labels.	1491
	✓ Axes.get_xmajorticklabels	Return the xaxis' major tick labels, as a list of Text.	1491
	✓ Axes.get_xminorticklabels	Return the xaxis' minor tick labels, as a list of Text.	1491
	✓ Axes.get_xgridlines	Return the xaxis' grid lines as a list of Line2Ds.	1491
	✓ Axes.get_xticklines	Return the xaxis' tick lines as a list of Line2Ds.	1491
	Axes.xaxis_date	Set up axis ticks and labels to treat data along the xaxis as dates.	1491
	Axes.set_yticks	Set the yaxis' tick locations.	1491
	Axes.get_yticks	Return the yaxis' tick locations in data coordinates.	1491
	✓ Axes.set_yticklabels	Set the yaxis' labels with list of string labels.	1491
	✓✓ Axes.get_yticklabels	Get the yaxis' tick labels.	1491
	✓ Axes.get_ymajorticklabels	Return the yaxis' major tick labels, as a list of Text.	1491
	✓ Axes.get_yminorticklabels	Return the yaxis' minor tick labels, as a list of Text.	1491
	✓ Axes.get_ygridlines	Return the yaxis' grid lines as a list of Line2Ds.	1491
	✓ Axes.get_yticklines	Return the yaxis' tick lines as a list of Line2Ds.	1491
	Axes.yaxis_date	Set up axis ticks and labels to treat data along the yaxis as dates.	1491
	✓ Axes.minorticks_off	Remove minor ticks from the axes.	1491
	✓ Axes.minorticks_on	Display minor ticks on the axes.	1491
	Axes.ticklabel_format	Configure the ScalarFormatter used by default for linear axes.	1491
	✓✓ Axes.tick_params	Change the appearance of ticks, tick labels, and gridlines.	1491
	✓ Axes.locator_params	Control behavior of major tick locators.	1491

✓ indicates frequency and importance of usage of the method

Also, can use `plt.yticks([ticks, labels])`, `plt.xticks([ticks, labels])` to set ticks and labels in one call.



- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXES HELPER METHODS FOR AXIS LABELS, TITLE AND LEGEND			
Category	Method	Description	Page No.
Axis labels, title, and legend	✓ Axes.set_xlabel	Set the label for the x-axis.	1445
	✓ Axes.get_xlabel	Get the xlabel text string.	1445
	✓ Axes.set_ylabel	Set the label for the y-axis.	1445
	✓ Axes.get_ylabel	Get the ylabel text string.	1445
	✓ Axes.set_title	Set a title for the Axes.	1445
	Axes.get_title	Get an Axes title.	1445
	✓ Axes.legend	Place a legend on the Axes.	1445
	Axes.get_legend	Return the Legend instance, or None if no legend is defined.	1445
	Axes.get_legend_handles_labels	Return handles and labels for legend	1445
	✓ Axes.set_xlabel	Set the label for the x-axis.	1445
	Axes.get_xlabel	Get the xlabel text string.	1445
	✓ Axes.set_ylabel	Set the label for the y-axis.	1445
	Axes.get_ylabel	Get the ylabel text string.	1445
	✓ Axes.set_title	Set a title for the Axes.	1445
Twinning and sharing	Axes.twinx	Create a twin Axes sharing the xaxis.	1513
	Axes.twiny	Create a twin Axes sharing the yaxis. Check link .	1513
	Axes.sharex	Share the x-axis with other.	1513
	Axes.sharey	Share the y-axis with other.	1513
	Axes.get_shared_x_axes	Return a reference to the shared axes Grouper object for x axes.	1513
	Axes.get_shared_y_axes	Return a reference to the shared axes Grouper object for y axes.	1513

✓ indicates frequency and importance of usage of the method

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXIS HELPER METHODS (1/3)			
Category	Method	Description	Page No.
Formatters and Locators	Axis.get_major_formatter	Get the formatter of the major ticker.	1540
	Axis.get_major_locator	Get the locator of the major ticker.	1540
	Axis.get_minor_formatter	Get the formatter of the minor ticker.	1540
	Axis.get_minor_locator	Get the locator of the minor ticker.	1540
	✓ Axis.set_major_formatter	Set the formatter of the major ticker.	1540
	✓ Axis.set_major_locator	Set the locator of the major ticker.	1540
	✓ Axis.set_minor_formatter	Set the formatter of the minor ticker.	1540
	✓ Axis.set_minor_locator	Set the locator of the minor ticker.	1540
	Axis.remove_overlapping_locs	If minor ticker locations that overlap with major ticker locations should be trimmed.	1540
Axis Label	Axis.set_label_coords	Set the coordinates of the label.	1545
	✓ Axis.set_label_position	Set the label position (top or bottom)	1545
	Axis.set_label_text	Set the text value of the axis label.	1545

✓ indicates frequency and importance of usage of the method

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXIS HELPER METHODS (2/3)			
Category	Method	Description	Page No.
Ticks, tick labels, Offset text, GridLines	✓ Axis.get_major_ticks	Return the list of major Ticks.	1547
	✓ Axis.get_majorticklabels	Return this Axis' major tick labels, as a list of Text.	1547
	✓ Axis.get_majorticklines	Return this Axis' major tick lines as a list of Line2Ds.	1547
	Axis.get_majorticklocs	Return this Axis' major tick locations in data coordinates.	1547
	Axis.get_minor_ticks	Return the list of minor Ticks.	1547
	Axis.get_minorticklabels	Return this Axis' minor tick labels, as a list of Text.	1547
	Axis.get_minorticklines	Return this Axis' minor tick lines as a list of Line2Ds.	1547
	Axis.get_minorticklocs	Return this Axis' minor tick locations in data coordinates.	1547
	Axis.get_offset_text	Return the axis offsetText as a Text instance.	1547
	Axis.get_tick_padding		1547
	✓ Axis.get_ticklabels	Get this Axis' tick labels.	1547
	✓ Axis.get_ticklines	Return this Axis' tick lines as a list of Line2Ds.	1547
	✓ Axis.get_ticklocs	Return this Axis' tick locations in data coordinates.	1547
	✓ Axis.get_gridlines	Return this Axis' grid lines as a list of Line2Ds.	1547
	✓ Axis.grid	Configure the grid lines.	1547
	✓✓ Axis.set_tick_params	Set appearance parameters for ticks, ticklabels, and gridlines.	1547
	Axis.axis_date	Set up axis ticks and labels to treat data along this Axis as dates.	1547

✓ indicates frequency and importance of usage of the method

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

AXIS HELPER METHODS (3/3)			
Category	Method	Description	Page No.
XAxis Specific	XAxis.axis_name	Read-only name identifying the axis.	1559
	XAxis.get_text_heights	Return how much space should be reserved for text above and below the axes, as a pair of floats.	1559
	XAxis.get_ticks_position	Return the ticks position ("top", "bottom", "default", or "unknown").	1559
	✓ XAxis.set_ticks_position	Set the ticks position.	1559
	✓ XAxis.tick_bottom	Move ticks and ticklabels (if present) to the bottom of the axes.	1559
	✓ XAxis.tick_top	Move ticks and ticklabels (if present) to the top of the axes.	1559
YAxis Specific	YAxis.axis_name	Read-only name identifying the axis.	1563
	YAxis.get_text_widths		1563
	YAxis.get_ticks_position	Return the ticks position ("left", "right", "default", or "unknown").	1563
	YAxis.set_offset_position		1563
	✓ YAxis.set_ticks_position	Set the ticks position.	1563
	✓ YAxis.tick_left	Move ticks and ticklabels (if present) to the left of the axes.	1563
Other	✓ YAxis.tick_right	Move ticks and ticklabels (if present) to the right of the axes.	1563
	✓ Axis.axes	The Axes instance the artist resides in, or None.	1563
Use with Caution*	Axis.set_ticks	Set this Axis' tick locations.	1564
	Axis.set_ticklabels	Set the text values of the tick labels.	1565
*Warning :This method should only be used after fixing the tick positions using Axis.set_ticks . Otherwise, the labels may end up in unexpected positions.			
✓ indicates frequency and importance of usage of the method			

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

TICK PARAMS METHOD – BULK PROPERTY SETTER

Tick Params to change the appearance of ticks, tick labels, and gridlines.
Call signature : [Axes.tick_params\(\)](#), [axis.set_tick_params\(\)](#)

Parameters	Description
axis : {'x', 'y', 'both'}, default: 'both'	The axis to which the parameters are applied.
which : {'major', 'minor', 'both'}, default: 'major'	The group of ticks to which the parameters are applied.
reset : bool, default: False	Whether to reset the ticks to defaults before updating them.
direction : {'in', 'out', 'inout'}	Puts ticks inside the axes, outside the axes, or both.
length : float	Tick length in points.
width : float	Tick width in points.
color : color	Tick color.
pad : float	Distance in points between tick and label.
labelsize : float or str	Tick label font size in points or as a string (e.g., 'large').
labelcolor : color	Tick label color.
colors : color	Tick color and label color.
zorder: float	Tick and label zorder.
bottom, top, left, right : bool	Whether to draw the respective ticks.
labelbottom, labeltop, labelleft, labelright :bool	Whether to draw the respective tick labels.
labelrotation : float	Tick label rotation
grid_color : color	Gridline color.
grid_alpha : float	Transparency of gridlines: 0 (transparent) to 1 (opaque).
grid_linewidth : float	Width of gridlines in points.
grid_linestyle : str	Any valid Line2D line style spec.

[Matplotlib Release 3.4.2 document, Pg No. 1491, 1504-1506](#)

Enter length = 0 if tick lines are not desired



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

A Note on Practical implementation

1. Multiple ways to set the different parameters of ticks, tick labels and grid lines
2. Tick_params is a bulk property setter allowing all the ticks to be modified. Frequent use cases are :
 - [Direction of the tick either inwards](#) or outwards
 - Making tick lengths equal to zero
 - Placing Tick labels inside the axes
 - Adjusting the padding between the tick and labels
 - Setting the label color and label size
3. However, note that Tick_params does not have horizontal alignment or vertical alignment argument.
4. Use Axes.Axis.get_ticklabels() to access tick labels as text instances within a list.

[Check Examples](#)

```
[In]: type(ax1.xaxis.get_ticklabels()[0])
[Out]: matplotlib.text.Text
```

5. We can [Iterate/Loop](#) over all the tick labels with or without conditions and then use [plt.setp or artist.set\(\)](#) method to modify text instance properties.

```
for labels in ax.yaxis.get_ticklabels():
    plt.setp(labels, size = 14, rotation_mode = 'anchor',
             ha = 'right')
```

```
ax.yaxis.set_tick_params(rotation = 0, labelsiz = 15)
```

```
[ticks.set(alpha = 0.5) for ticks in ax.get_xticklabels()]
[ticks.set(alpha = 0.5) for ticks in ax.get_yticklabels()]
```

[Check Examples](#)

6. Grid lines can be set by using [Axes.grid](#)(b = None, which = 'major', axis = 'both', **kwargs) or [Axes.Axis.grid](#)(b = None, which = 'major', **kwargs).
7. Use [Axes.set_axisbelow\(True\)](#) to place gridlines/tick lines behind or below all the artists.
8. Use plt.xticks([ticks, labels]), plt.yticks([ticks, labels]) to place ticks and labels in one call. This is helpful for simple use-cases.
9. Beyond simple usecases, it is preferable to use [Tick Locator/Formatter pairs](#) to customize the tick positions and the string formatting of the ticks. For general tick locator/formatter, check [matplotlib.ticker module](#). For datetime tick locator/formatter, check [matplotlib.dates module](#).



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TICK LOCATORS FORMATTERS

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – LOCATOR CLASS

Tick locators define the position of the ticks.

Locator class	Description
NullLocator	No ticks
FixedLocator	Tick locations are fixed
IndexLocator	Locator for index plots (e.g., where $x = \text{range}(\text{len}(y))$)
LinearLocator	Evenly spaced ticks from min to max
LogLocator	Logarithmically ticks from min to max
MultipleLocator	Ticks and range are a multiple of base
MaxNLocator	Finds up to a max number of ticks at nice locations
AutoLocator	(Default.) MaxNLocator with simple defaults.
AutoMinorLocator	Locator for minor ticks

*[Matplotlib Release, 3.4.2, Page No. 2812](#)

*<https://jakevdp.github.io/PythonDataScienceHandbook/04.10-customizing-ticks.html>

Use tick **locator/formatter** pairs for controlling tick position and string representation

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

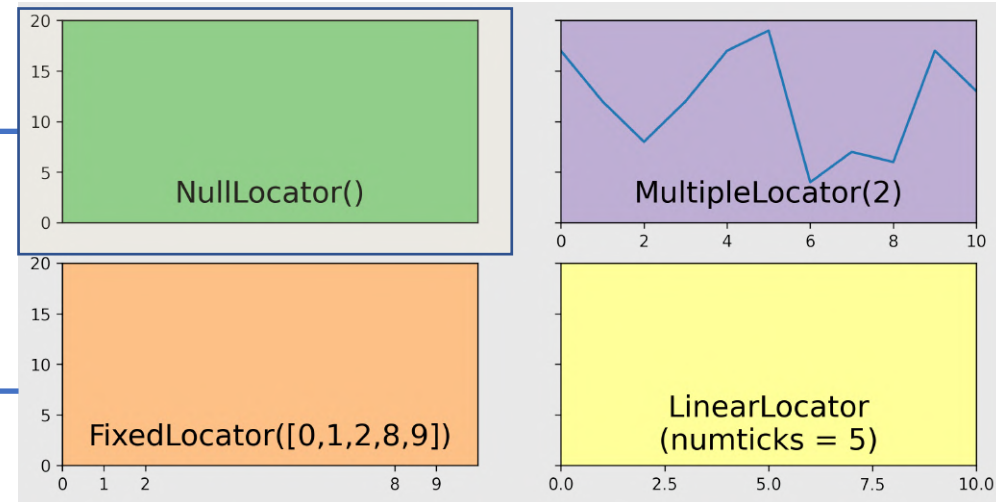
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – LOCATORs - DEMO 1



```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

```
fig, ax = plt.subplots(2,2,figsize= (10,5), sharey = True)
ax1,ax2,ax3,ax4 = ax.flatten() #Unpacking numpy array of axes
[ax.set_ylim(0,20) for ax in fig.axes] #Setting ylim as (0,20)
```

```
ax1.xaxis.set_major_locator(ticker.NullLocator())
ax2.xaxis.set_major_locator(ticker.MultipleLocator(2))
ax3.xaxis.set_major_locator(ticker.FixedLocator([0,1,2,8,9]))
ax4.xaxis.set_major_locator(ticker.LinearLocator(5))
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

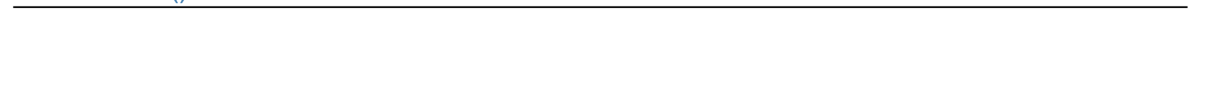
Set/Modify the Artist properties after accessing the artists

7

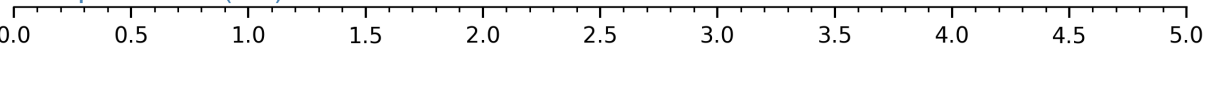
Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – LOCATORs - DEMO 2

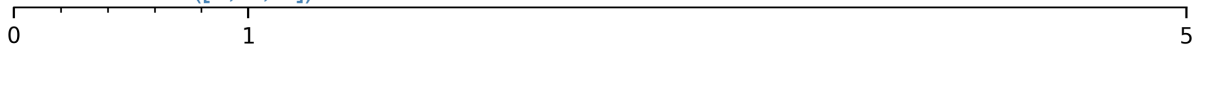
`NullLocator()`



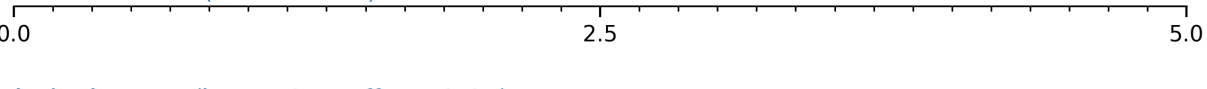
`MultipleLocator(0.5)`



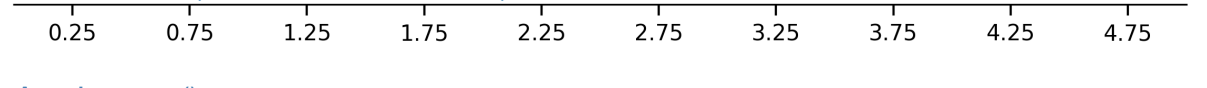
`FixedLocator([0, 1, 5])`



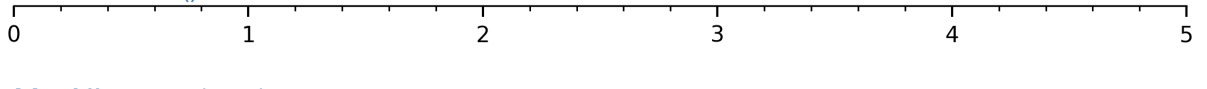
`LinearLocator(numticks=3)`



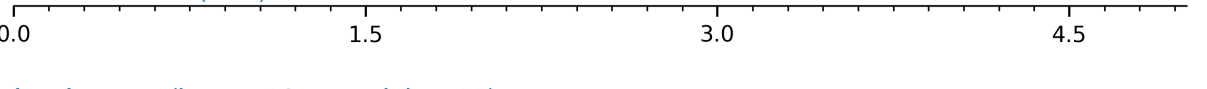
`IndexLocator(base=0.5, offset=0.25)`



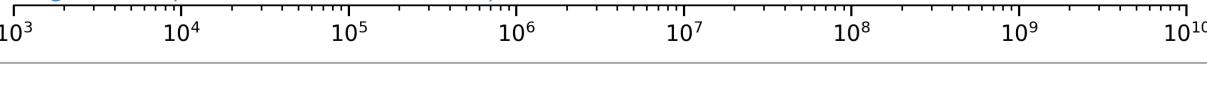
`AutoLocator()`



`MaxNLocator(n=4)`



`LogLocator(base=10, numticks=15)`



- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – FORMATTER CLASS

Tick formatters define how the numeric value associated with a tick on an axis is formatted as a string.

Formatter Class	Description
NullFormatter	No labels on the ticks
IndexFormatter	Set the strings from a list of labels
FixedFormatter	Set the strings manually for the labels
FuncFormatter	User-defined function sets the labels
FormatStrFormatter	Use a format string for each value
ScalarFormatter	(Default.) Formatter for scalar values
LogFormatter	Default formatter for log axes

*[Matplotlib Release, 3.4.2, Page No. 2814](#)

*<https://jakevdp.github.io/PythonDataScienceHandbook/04.10-customizing-ticks.html>

Use tick **locator/formatter** pairs for controlling tick position and string representation

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

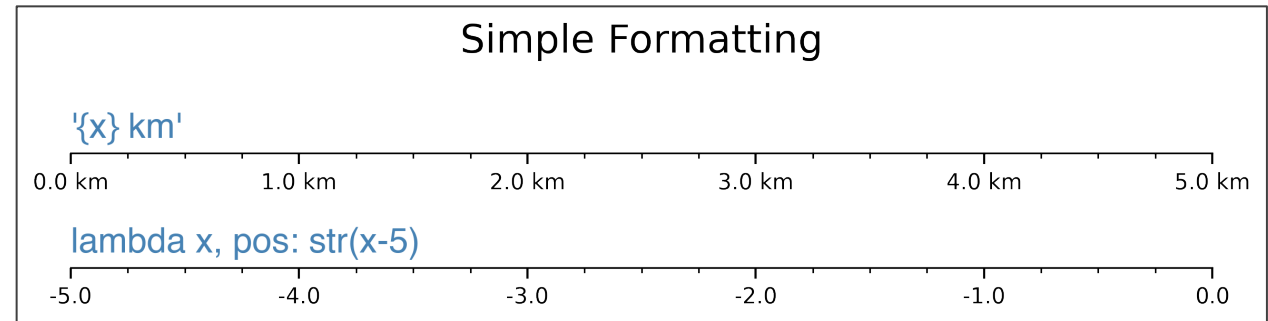
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – USING SIMPLE FORMATTING

[Code](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

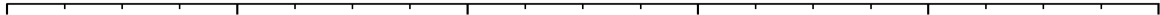
Add Legend, Annotations if any and Set Title to the Axes

SPECIAL NOTE ON TICKERS – USING FORMATTER CLASS

[Code](#)

Formatter Object Formatting

`NullFormatter()`



`StrMethodFormatter('{x:.3f}')`



`FuncFormatter("[{:.2f}"].format)`



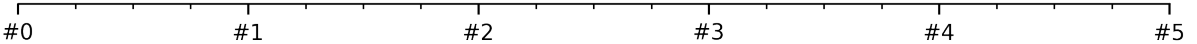
`FixedFormatter(['A', 'B', 'C', ...])`



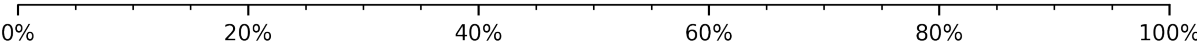
`ScalarFormatter()`



`FormatStrFormatter('#%d')`



`PercentFormatter(xmax=5)`



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TICKERS FOR TIME SERIES PLOTTING (1/3)

- Matplotlib handles dates in its own format.
- Helper Methods are available to convert [datetime objects](#) and [timedelta objects](#) to matplotlib dates and vice-versa.

datetime

Python's built-in module to handle date/ time

The [datetime](#) module is python's built-in module that supplies classes for manipulating dates and times.

Four Main Classes				TZ Awareness
time only time in hours, minutes, seconds and microseconds	date only year, month and day	datetime All components of time and date	timedelta An amount of time with maximum unit of days	tzinfo
				timezone

dateutil

powerful extensions to datetime

The [dateutil](#) module provides additional code to handle date ticking, making it easy to place ticks on any kinds of dates.

Check [Link](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TICKERS FOR TIME SERIES PLOTTING (2/3)

Helper methods in `matplotlib.dates` for conversion between datetime objects and Matplotlib dates

METHOD

DESCRIPTION

[`datestr2num`](#)

Convert a date string to a datenum using [`dateutil.parser.parse`](#)

[`date2num`](#)

Convert datetime objects to Matplotlib dates.

[`num2date`](#)

Convert Matplotlib dates to [`datetime`](#) objects.

[`num2timedelta`](#)

Convert number of days to a [`timedelta`](#) object.

[`drange`](#)

Return a sequence of equally spaced Matplotlib dates.

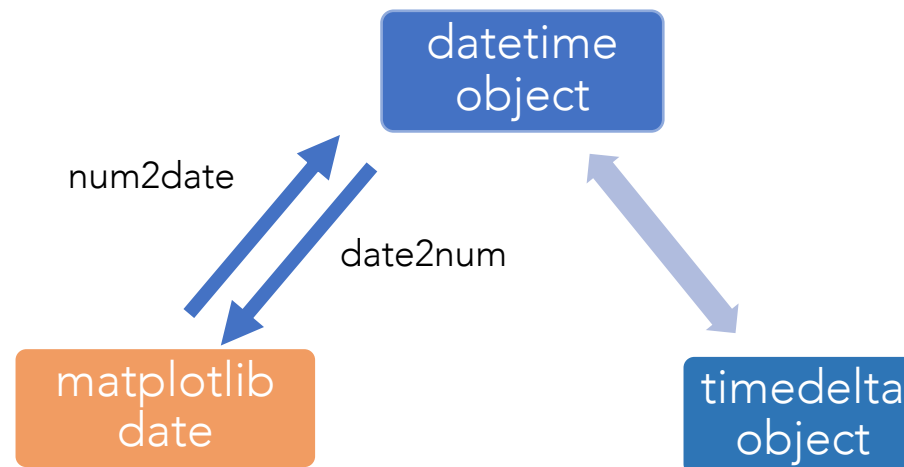
[`set\_epoch`](#)

Set the epoch (origin for dates) for datetime calculations.

[`get\_epoch`](#)

Get the epoch used by [`dates`](#).

Go to
Section



- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

TICKERS FOR TIME SERIES PLOTTING (3/3)

Tick locators and formatters from [matplotlib.dates module](#)

DATE LOCATORS

MicrosecondLocator	Locate microseconds.
SecondLocator	Locate seconds.
MinuteLocator	Locate minutes.
HourLocator	Locate hours.
DayLocator	Locate specified days of the month.
WeekdayLocator	Locate days of the week, e.g., MO, TU.
MonthLocator	Locate months, e.g., 7 for July.
YearLocator	Locate years that are multiples of base.
RRuleLocator	Locate using a matplotlib.dates.rrulewrapper. rrulewrapper is a simple wrapper around dateutil's dateutil.rrule which allow almost arbitrary date tick specifications. See rrule example .
AutoDateLocator	On autoscale, this class picks the best DateLocator (e.g., RRuleLocator) to set the view limits and the tick locations.

DATE FORMATTERS

AutoDateFormatter	attempts to figure out the best format to use. This is most useful when used with the AutoDateLocator .
ConciseDateFormatter	also attempts to figure out the best format to use, and to make the format as compact as possible while still having complete date information. This is most useful when used with the AutoDateLocator .
DateFormatter	use strftime format strings.
IndexDateFormatter	date plots with implicit x indexing.


```
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates # For Date locators and formatters
import numpy as np
```

```
base = datetime.datetime(2005, 2, 1)
dates = [base + datetime.timedelta(hours=(2 * i)) for i in range(732)]
N = len(dates)
```

```
#Fixing random state for reproducibility
np.random.seed(19680801)
y = np.cumsum(np.random.randn(N))
```

```
#Limits for the three subplots/axes
lims = [(np.datetime64('2005-02'), np.datetime64('2005-04')),
        (np.datetime64('2005-02-03'), np.datetime64('2005-02-15')),
        (np.datetime64('2005-02-03 11:00'), np.datetime64('2005-02-04 13:20'))]
```

```
fig, axs = plt.subplots(3, 1, constrained_layout=True, figsize=(6, 6))
for nn, ax in enumerate(axs):
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)

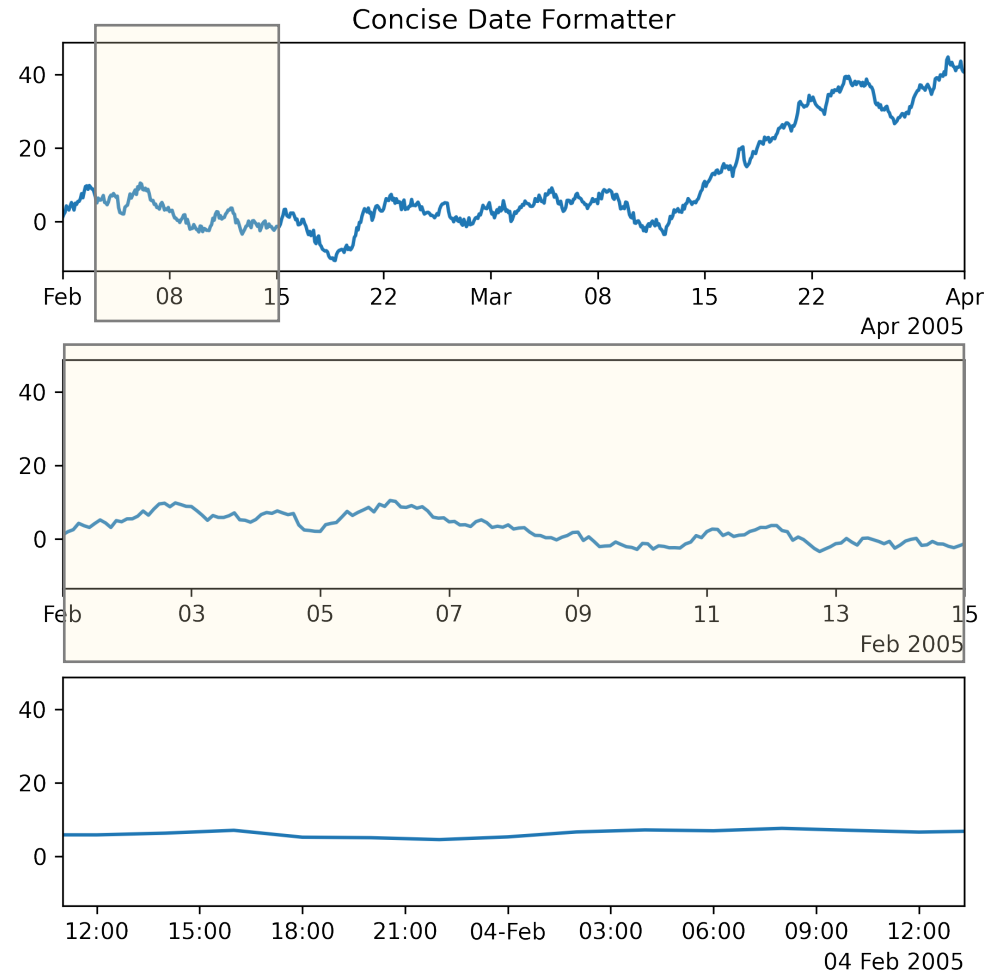
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)

    ax.plot(dates, y)
    ax.set_xlim(lims[nn])

axs[0].set_title('Concise Date Formatter')

plt.show()
```

CONCISE DATE FORMATTER EXAMPLE



https://matplotlib.org/stable/gallery/ticks_and_spines/date_concise_formatter.html?highlight=date

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

OO STYLE ESSENCE

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

- In OO style, we want to access the artists first.
- This would allow us to access the attributes and properties of the artist.
- We can then modify the properties using setter methods.
- We can also inspect into the attributes, add primitives into them and modify using appropriate methods.

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ACCESSING ARTISTS

➤ BY ASSIGNING TO A VARIABLE WHEN CREATING

One way to gain access to the primitives is to assign it to a variable when it is created using Axes helper method.

Instead of :

```
ax.bar(x,y) #returns rectangular patch objects
```

we should assign it to a variable as below :

```
bars = ax.bar(x,y)
```

Now, bars variable contain the *rectangular patch objects* that were returned by the `ax.bar()` method.

We can use below methods to inspect the children of the bars variable:

```
str(bars)
```

After finding the index location of [Barcontainer](#) object, we will access the rectangular patches :

```
bars[0].get_children()
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ACCESSING ARTISTS

➤ THROUGH ATTRIBUTES (1/2)

The Second way is to access the attributes.

In the current example, we will access the patches attribute of ax object.

```
ax.bar(x,y)  
ax.patches
```

We can then iterate through the patches in ax.patches.

In a similar manner, we have access to other axes attributes

```
ax.patch  
ax.patches  
ax.lines  
ax.texts  
ax.spines  
ax.legend()  
ax.collections  
ax.containers
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ACCESSING ARTISTS

➤ THROUGH ATTRIBUTES (2/2)

ITERATING OVER THE ARTIST ATTRIBUTES CONTAINERS

```
bars = [rect for rect in ax.get_children() if  
        isinstance(rect, mpl.patches.Rectangle)]
```

```
[p.set_label(x[i]) for i,p in zip(range(len(x)), ax.patches)]
```

```
[p.set_label(x[i]) for i,p in enumerate(ax.patches)]
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

USEFUL METHODS AND PYTHON BUILT IN METHODS

[Check](#)

ARTIST OBJECT METHODS

- `matplotlib.artist.getp(artistobject)`
- `plt.setp(artistobject)`
- `artist.get_children()`
- `artist.findobj()`

FOR INSPECTION

- `print(dir(object))`
- `vars(ax)` or `ax.__dict__`
- `str(ax)`
- `type(object)`
- `len(object)`
- `isinstance(obj, class)`
- `get_text()`
- `hasattr(), getattr(), setattr()`

FOR LOOPING/ ITERATION

- `enumerate()`
- `zip()`
- `flatten()`
- `ravel()`
- list comprehension
- `dict.key()`
- `dict.values()`
- `list()`
- Lambda function

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

ACCESSING ARTISTS THROUGH ATTRIBUTES

FIGURE	Figure attribute Description	
	axes	A list of Axes instances (includes Subplot)
	patch	The Rectangle background
	images	A list of FigureImage patches - useful for raw pixel display
	legends	A list of Figure Legend instances (different from Axes.legends)
	lines	A list of Figure Line2D instances (rarely used, see Axes.lines)
	patches	A list of Figure Patches (rarely used, see Axes.patches)
	texts	A list Figure Text instances
AXES	Axes attribute Description	
	artists	A list of Artist instances
	patch	Rectangle instance for Axes background
	collections	A list of Collection instances
	images	A list of AxesImage
	legends	A list of Legend instances
	lines	A list of Line2D instances
	patches	A list of Patch instances
	texts	A list of Text instances
	xaxis	A matplotlib.axis.XAxis instance
	yaxis	A matplotlib.axis.YAxis instance
AXIS	Use Axis Accessor/Helper Methods to access axis labels, tick lines, tick labels, tick locators	
TICK	Tick attribute Description	
	tick1line	A Line2D instance
	tick2line	A Line2D instance
	gridline	A Line2D instance
	label1	A Text instance
	label2	A Text instance
	*Each of these is accessible directly as an attribute of the Tick.	

*Matplotlib Release, 3.4.2, Page No. 7-11, 113-120



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

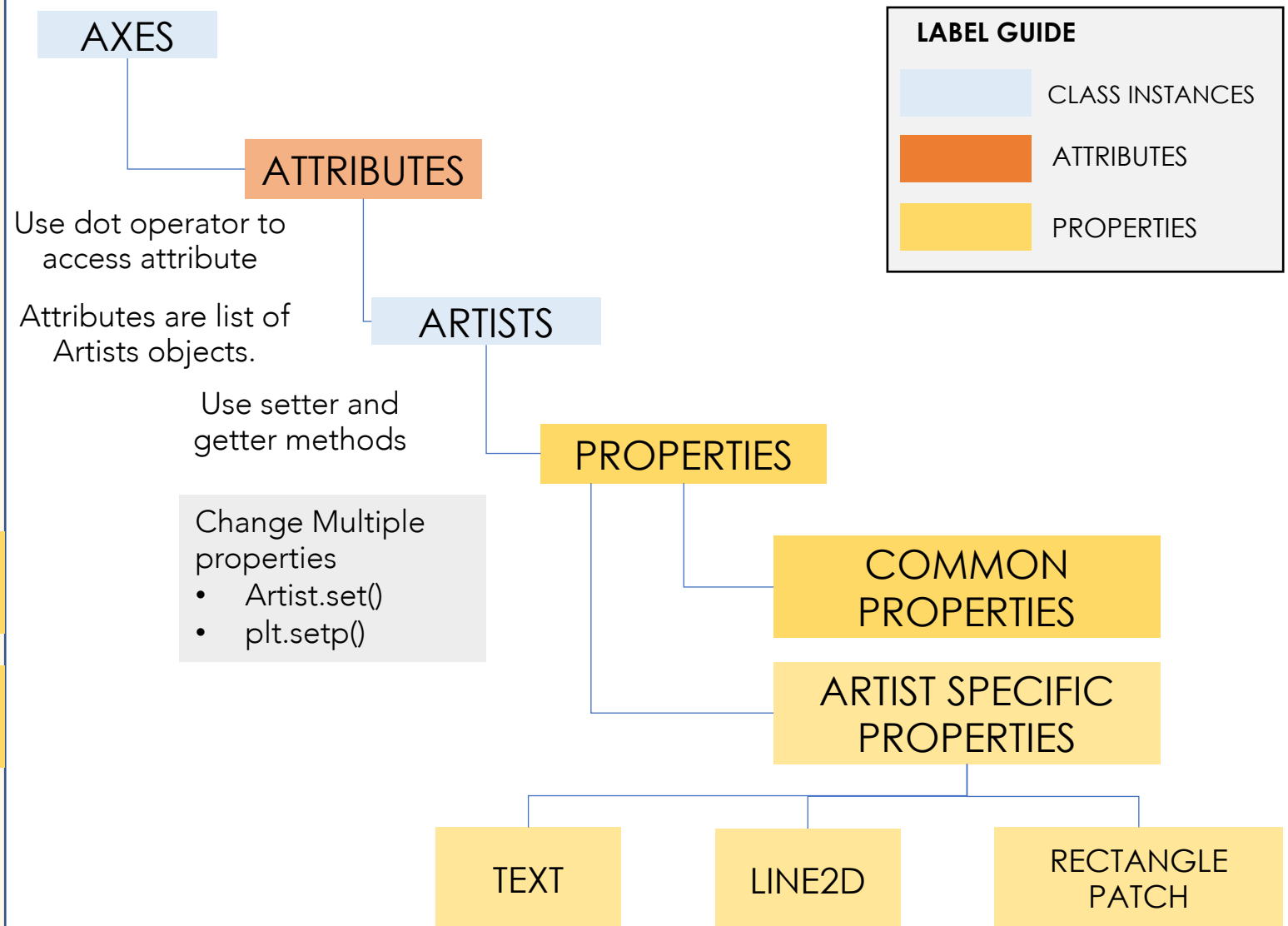
Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

MAPPING THE JOURNEY

FROM AXES INSTANCE TO ARTIST INSTANCE PROPERTIES



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists through attributes.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, and Set Title to the Axes
Set Title to the Axes

PROPERTIES COMMON TO ALL ARTISTS (1/2)

PROPERTY	DESCRIPTION
alpha	The transparency - a scalar from 0-1
animated	A boolean that is used to facilitate animated drawing
axes	The Axes that the Artist lives in, possibly None
clip_box	The bounding box that clips the Artist
clip_on	Whether clipping is enabled
clip_path	The path the artist is clipped to
contains	A picking function to test whether the artist contains the pick point
figure	The figure instance the artist lives in, possibly None
label	A text label (e.g., for auto-labeling)
picker	A python object that controls object picking
transform	The transformation
visible	A boolean whether the artist should be drawn
zorder	A number which determines the drawing order
rasterized	Boolean; Turns vectors into raster graphics (for compression & EPS transparency)

[*Matplotlib Release, 3.4.2, Page No. 113-120](#)

- **"Each of the properties is accessed with setter or getter"**
- (The exception being axes property which can be accessed by dot operator)

```
ax.patch.get_alpha()
ax.patch.set_alpha( ax.patch.get_alpha()*1.2 )
ax.patch.set_alpha( 0.5 )
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists through attributes.

6

Set/Modify the Artist properties after accessing the artists

7

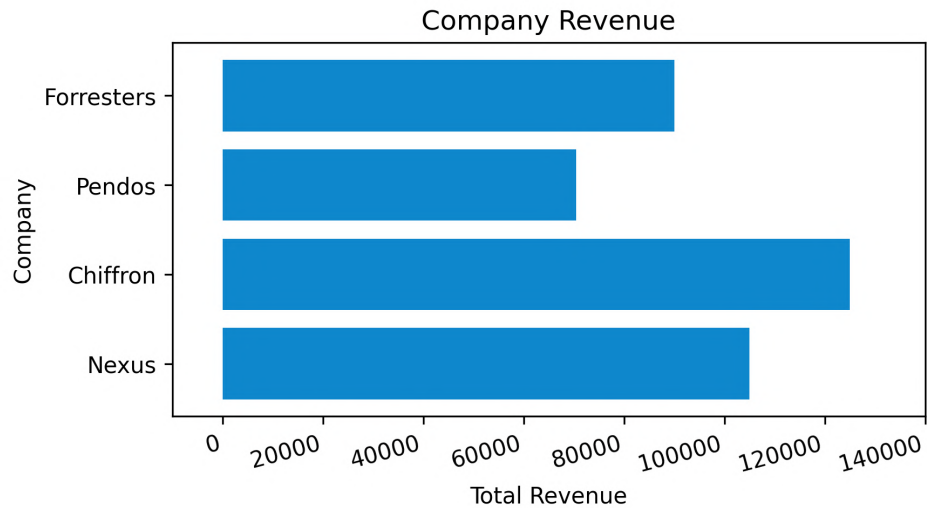
Add Legend, and Set Title to the Axes
Set Title to the Axes

PROPERTIES COMMON TO ALL ARTISTS (2/2)

USE CASE SCENARIO	ARTIST PROPERTY
For User-defined function	figure, axes in conjunction with plt.gcf(), plt.gca(), plt.scf(), plt.sca()
For Adding artist to Legend	label
Setting ticks, axis, spines or even axes(for multiple subplots) invisible	visible
If multiple artists overlap such as axhspan, axvspan, fillbetween	zorder
If text, patch, arrows, annotation etc. are to be added either in data coordinates or in figure coordinates	transform : ax.transData ax.transAxes fig.transfigure fig.dpi_scale_trans where ax is an axes instance, fig is a figure instance. Refer to official docs for full list.

SETTING ARTIST PROPERTIES

- Use `setp` function of `Pyplot` Interface
- Use `Artist.set()` method to set properties of this `Axes` object.



Preparing the Data

```
group_names = ['Nexus', 'Chiffon', 'Pendos', 'Forresters']  
group_data = [105000, 125000, 70500, 90000]
```

```
plt.rcParams()
```

```
fig, ax = plt.subplots(figsize=(8, 4)) #Instantiating fig and axes  
ax.barh(group_names, group_data, height = 0.8)
```

Accessing the artist using axes accessor method

```
labels = ax.get_xticklabels()
```

Setting the artist property using `plt.setp` method

```
plt.setp(labels, rotation=15, ha = 'right', fontsize = 10)
```

Setting the artist property using `artist.set()` method

```
ax.set(xlim=[-10000, 140000],  
      xlabel='Total Revenue', ylabel='Company',  
      title='Company Revenue')
```

Setting the artist property using `artist.set_p()` method

```
[bar.set_facecolor('xkcd:water blue') for bar in ax.patches]
```

```
fig.savefig("Setting property.png", dpi=300, format='png', bbox_inches='tight')
```

```
import matplotlib.pyplot as plt
lines = plt.plot([1, 2, 3])
plt.setp(lines) # list of settable line properties
```

agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha: float or None
animated: bool
antialiased or aa: bool
clip_box: `.Bbox``
clip_on: bool
clip_path: Patch or (Path, Transform) or None
color or c: color
contains: unknown
dash_capstyle: {'butt', 'round', 'projecting'}
dash_joinstyle: {'miter', 'round', 'bevel'}
dashes: sequence of floats (on/off ink in points) or (None, None)
data: (2, N) array or two 1D arrays
drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default'
figure: `.Figure``
fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
gid: str
in_layout: bool
label: object linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
linewidth or lw: float

INSPECTING ARTIST PROPERTIES

- `matplotlib.artist.getp(artistobject)`
- `plt.setp(artistobject)`

marker: marker style string, `~.path.Path`` or `~.markers.MarkerStyle``
markeredgewidth or mew: float
markerfacecolor or mfc: color
markerfacecoloralt or mfcalt: color
markersize or ms: float
markevery: None or int or (int, int) or slice or List[int] or float or (float, float) or List[bool]
path_effects: `.AbstractPathEffect``
picker: unknown pickradius: float
rasterized: bool or None
sketch_params: (scale: float, length: float, randomness: float)
snap: bool or None
solid_capstyle: {'butt', 'round', 'projecting'}
solid_joinstyle: {'miter', 'round', 'bevel'}
transform: `matplotlib.transforms.Transform``
url: str
visible: bool
xdata: 1D array
ydata: 1D array
zorder: float

`#matplotlib.artist.getp` also returns a listing of properties for a given object.

```
import matplotlib.pyplot as plt
import matplotlib.artist as mpa
lines = plt.plot([1, 2, 3])
mpa.getp(lines)
```



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ADDITIONAL PROPERTIES SPECIFIC TO LINE2D

PROPERTY	DESCRIPTION
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
antialiased or aa	[True False]
color or c	any matplotlib color
dash_capstyle	['butt' 'round' 'projecting']
dash_joinstyle	['miter' 'round' 'bevel']
dashes	sequence of on/off ink in points
drawstyle	['default' 'steps' 'steps-pre' 'steps-mid' 'steps-post']
fillstyle	['full' 'left' 'right' 'bottom' 'top' 'none']
gid	an id string
linestyle or ls	['solid' 'dashed' 'dashdot' 'dotted' (offset, on-off-dash-seq) '-' '--' '-.' ':' 'None' '' '']
linewidth or lw	float value in points
marker	A valid marker style
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markerfacecoloralt or mfcalt	any matplotlib color
markersize or ms	float
markevery	[None int length-2 tuple of int slice list/array of int float length-2 tuple of float]
path_effects	AbstractPathEffect
pickradius	float distance in points
solid_capstyle	['butt' 'round' 'projecting']
solid_joinstyle	['miter' 'round' 'bevel']
url	a url string
xdata	1D array
ydata	1D array

*Matplotlib Release, 3.4.2, Page No. 2260-61

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

ADDITIONAL PROPERTIES SPECIFIC TO TEXT (1/2)

PROPERTY	DESCRIPTION
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
backgroundcolor	color
bbox	dict with properties for patches.FancyBboxPatch
color or c	color
fontfamily or family	{FONTNAME, 'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
fontproperties or font or font_manager.FontProperties or str or pathlib.Path	font_properties
fontsize or size	float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}
fontstretch or stretch	horizontal condensation or expansion {a numeric value in range 0-1000, 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded', 'ultra-expanded'}
fontstyle or style	{'normal', 'italic', 'oblique'}
fontvariant or variant	{'normal', 'small-caps'}
fontweight or weight	{a numeric value in range 0-1000, 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
gid	str

[*Matplotlib Release, 3.4.2, Page No. 2795-2796, 326-329](#)

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

ADDITIONAL PROPERTIES SPECIFIC TO TEXT (2/2)

PROPERTY	DESCRIPTION
horizontalalignment or ha	{'center', 'right', 'left'}
in_layout	bool
linespacing	float (multiple of font size)
math_fontfamily	str
multialignment or ma	{'left', 'right', 'center'}
path_effects	AbstractPathEffect
position	(float, float)
rotation	float or {'vertical', 'horizontal'}
rotation_mode	{None, 'default', 'anchor'}
text	object
transform_rotates_text	bool
url	str
usetex	bool or None
verticalalignment or va	{'center', 'top', 'bottom', 'baseline', 'center_baseline'}
visible	bool
wrap	bool
x	float
y	float
zorder	float
url	str
usetex	bool or None

[*Matplotlib Release, 3.4.2, Page No. 2795-2796](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

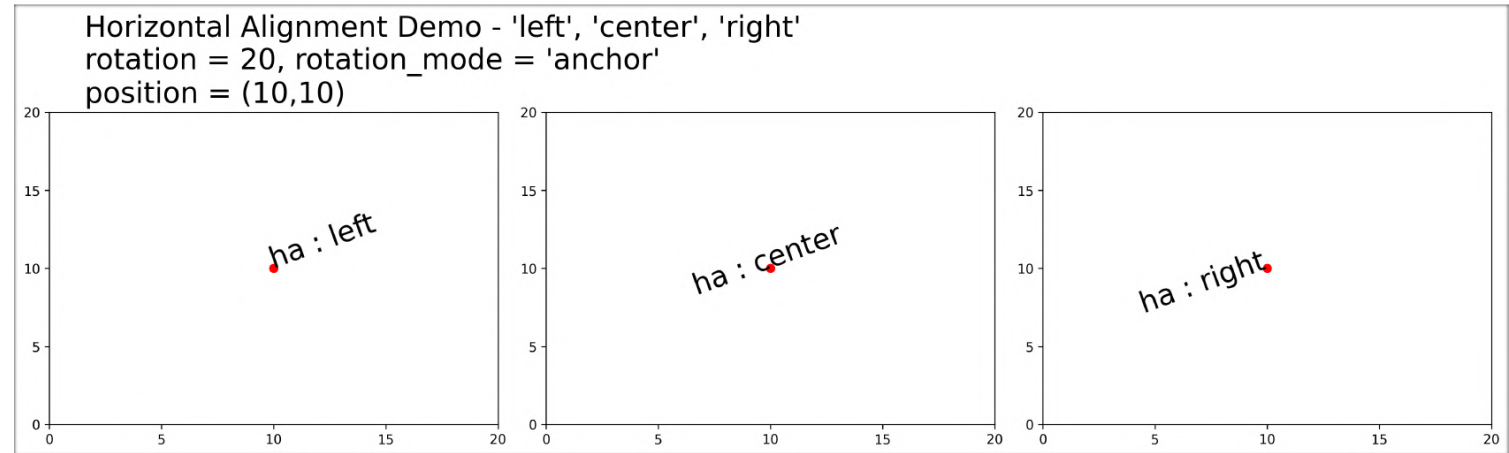
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TEXT INSTANCE – HORIZONTAL ALIGNMENT DEMO

[Code](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

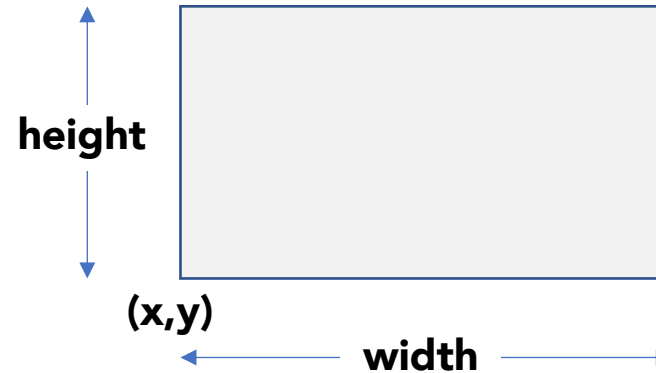
Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

SPECIAL LOOK INTO PATCH RECTANGLE

RECTANGLE PARAMETERS



(x,y)	[(float, float)] The anchor point
width	[float] Rectangle width
height	[float] Rectangle height
angle	[float, default: 0] Rotation in degrees anti-clockwise about xy

[*Matplotlib Release, 3.4.2, Page No. 2409](#)

Note: the float values are in the coordinate system for the data, controlled by xlim and ylim.

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ADDITIONAL PROPERTIES PATCH RECTANGLE SPECIFIC

PROPERTY	DESCRIPTION
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
antialiased or aa	unknown
capstyle	CapStyle or {'butt', 'projecting', 'round'}
color	color
edgecolor or ec	color or None or 'auto'
facecolor or fc	color or None
fill	bool
gid	str
hatch	{'/', '\\', ' ', '-', '+', 'x', 'o', 'O', '.', '*'}
in_layout	bool
joinstyle	JoinStyle or {'miter', 'round', 'bevel'}
linestyle or ls	{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
linewidth or lw	float or None
path_effects	AbstractPathEffect
sketch_params	(scale: float, length: float, randomness: float)
snap	bool or None
url	str

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

USE CASES – MODIFYING THE RECTANGLE PATCH PROPERTIES

Patch instance	Use Case Scenario
fig.patch	Changing color of figure background
ax.patch	Changing color of axes background
ax.patches	Adding data labels on top of the bars in a bar plot
ax.patches	Assigning label values to each of the patches of bar plot for adding to legend

```
import matplotlib.pyplot as plt
import numpy as np

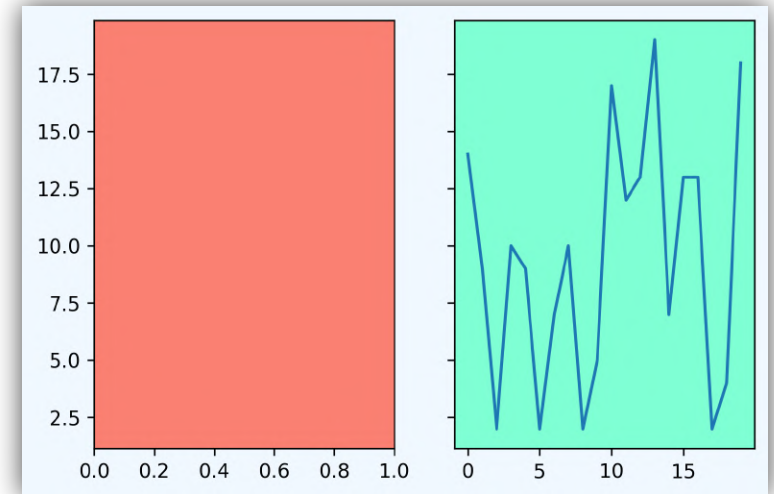
fig, (ax1, ax2) = plt.subplots(1,2, sharey = True)

x = np.random.randint(1,20,20)

#Set the face color of figure background
rect = fig.patch
rect.set(facecolor = 'aliceblue')

#Set the face color of the axes backgrounds
ax1.patch.set(facecolor = 'salmon')
ax2.patch.set(facecolor = 'aquamarine')
ax2.plot(x)

plt.show()
```



- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

METHODS OF RECTANGLE PATCH

Rectangle Patch Method	Description
<code>__str__(self)</code>	Return <code>str(self)</code> .
<code>get_bbox(self)</code>	Return the Bbox.
<code>get_height(self)</code>	Return the height of the rectangle.
<code>get_patch_transform(self)</code>	Return the Transform instance mapping patch coordinates to data coordinates.
<code>get_path(self)</code>	Return the vertices of the rectangle.
<code>get_width(self)</code>	Return the width of the rectangle.
<code>get_x(self)</code>	Return the left coordinate of the rectangle.
<code>get_xy(self)</code>	Return the left and bottom coords of the rectangle as a tuple.
<code>get_y(self)</code>	Return the bottom coordinate of the rectangle.
<code>set_bounds(self, *args)</code>	Set the bounds of the rectangle as left, bottom, width, height. The values may be passed as separate parameters or as a tuple:
<code>set_height(self, h)</code>	Set the height of the rectangle.
<code>set_width(self, w)</code>	Set the width of the rectangle.
<code>set_x(self, x)</code>	Set the left coordinate of the rectangle.
*Matplotlib Release, 3.4.2, Page No. 2409	

USE CASE SCENARIO	LABEL PROPERTY
For Adding data labels on top of the bars in bar plot	<code>get_x()</code> , <code>get_y()</code> , <code>get_height()</code> , <code>get_width()</code>
For User defined function	axes property, figure property in user defined function to retrieve the axes and figure respectively.

DEMO EXAMPLE – APPLYING LEARNINGS TILL NOW

PREPARING SAMPLE DATA

```
data = {'Water':465,  
        'Alcoholic Beverages':271,  
        'Carbonated soft drinks': 224,  
        'Milk and Dairy products': 260,  
        'New Drinks': 75,  
        'Fruits and Vegetable Juices': 86}  
x = list(data.keys())  
y = list(data.values())
```

CODE SCRIPT

```
import matplotlib.pyplot as plt  
import numpy as np  
import matplotlib.spines as spines  
colors = [ plt.cm.Dark2(x) for x in np.linspace(0, 1,  
len(x))]  
  
plt.rcParams()  
%matplotlib inline  
fig, ax = plt.subplots(figsize = (10,6), frameon = True)  
bars = ax.bar(x,y, color = colors)
```

```
StrTitle = 'Consumption of Packed beverages in  
billion liters,\nby Beverage Type, Global, 2019'  
ax.set_title(StrTitle, size = 16)
```

```
Add_data_labels(ax.patches)  
stylize_axes(ax)
```

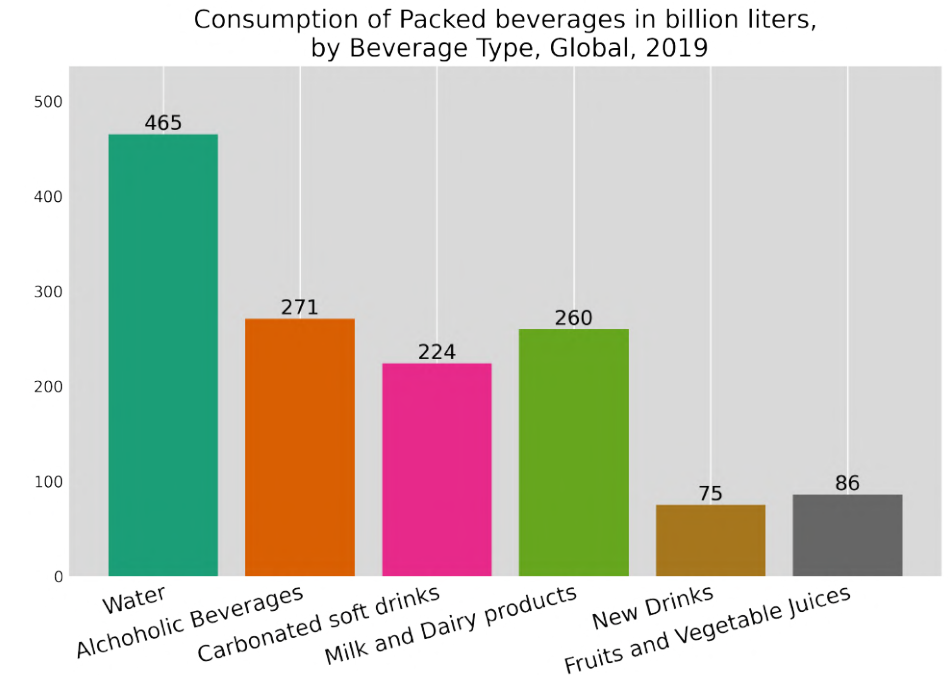
```
fig.savefig('Patches.png', dpi=300, format='png',  
bbox_inches='tight')
```

Check slide

- User defined function to add data labels
- Uses Axes helper methods to access the individual bars and use its properties to add texts

- User defined function to stylize the axes.
- Uses Axes helper methods to access and modify artist properties

PLOT OUTPUT

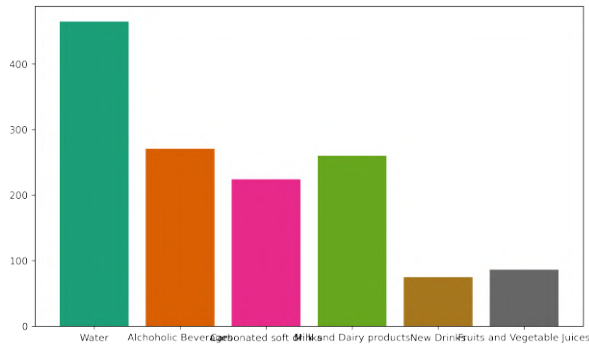


CREATING USER DEFINED FUNCTION TO STYLIZE AXES

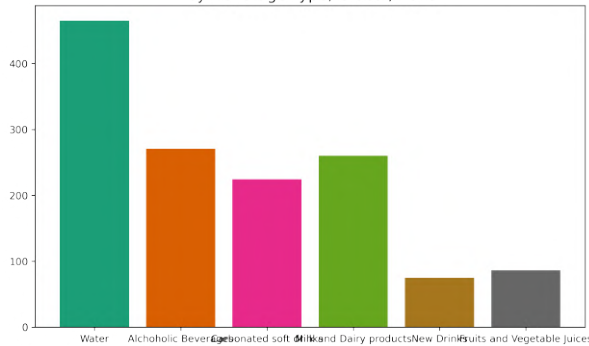
```
def stylize_axes(ax):  
  
    #Making the axes background light gray  
    ax.set_facecolor('.85')  
  
    # Setting the y limits  
    ax.set_ylim(0,ax.get_ylim()[1]*1.1)  
  
    #Making the tick lines disappear  
    ax.tick_params(length=0)  
  
    #Making the xtick labels rotated  
    for labels in ax.get_xticklabels():  
        labels.set(size = 13,rotation = 15,  
                    rotation_mode = 'anchor',  
                    ha = 'right')  
  
    #Setting grid lines to white color  
    ax.grid(True, axis='x', color='white')  
  
    #Setting grid lines below all the artists  
    ax.set_axisbelow(True)  
  
    #Making all the spines invisible  
    [spine.set_visible(False) for spine in ax.spines.values()]
```

CREATING USER DEFINED FUNCTION TO ADD DATA LABELS IN BAR PLOTS

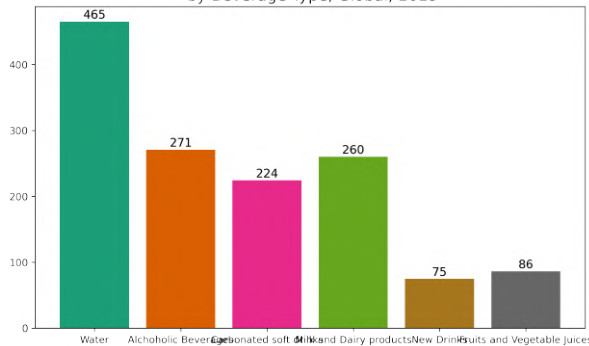
```
# Adding Annotations to place data labels  
def Add_data_labels(rect):    #rect is ax.patches object  
    for p in rect:  
        #Retrieving the Axes container of patch object  
        ax = rect[0].axes  
  
        #Adding text iteratively on top of each bar  
        #Using get and set on x,y parameters of rectangle patch  
        ax.text(x = p.get_x()+p.get_width()/2,  
                y = ax.get_ylim()[1]*0.01+ p.get_height(),  
                s = p.get_height(),  
                ha = 'center', size = 12)
```



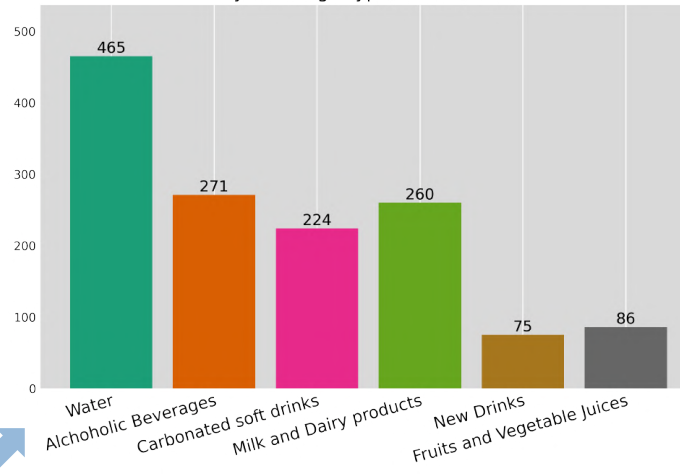
Consumption of Packed beverages in billion liters,
by Beverage Type, Global, 2019



Consumption of Packed beverages in billion liters,
by Beverage Type, Global, 2019



Consumption of Packed beverages in billion liters,
by Beverage Type, Global, 2019



DEMO - USERDEFINED FUNCTIONS TO STYLIZE PLOTS AND ADD LABELS

```
# Doing the necessary imports
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.spines as spines
```

```
plt.rcParams()
%matplotlib inline
```

```
#create a list of rgba colors from a matplotlib.cm colormap
colors = [ plt.cm.Dark2(x) for x in np.linspace(0, 1, len(x))]
```

```
#Instantiating figure, axes
fig, ax = plt.subplots(figsize = (10,6), frameon = True)
```

```
#Creating a bar plot by ax.bar method
# Note the colors passed as the color argument
bars = ax.bar(x,y, color = colors)
```

```
#Setting the title of plot
StrTitle = 'Consumption of Packed beverages in billion liters,\n by
Beverage Type, Global, 2019'
ax.set_title(StrTitle, size = 16)
```

```
#Using the user defined functions – to add data labels and stylize the axes
Add_data_labels(ax.patches)
stylize_axes(ax)
```

```
fig.savefig('Patches.png', dpi=300, format='png', bbox_inches='tight')
```


1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TRANSFORMATIONS

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TRANSFORMATION FRAMEWORK

Use Transformation Objects to easily move between coordinate systems

Coordinates	Transformation object	Description
"data"	ax.transData	The coordinate system for the data, controlled by xlim and ylim.
"axes"	ax.transAxes	The coordinate system of the Axes ; (0, 0) is bottom left of the axes, and (1, 1) is top right of the axes.
"subfigure"	subfigure.transSubfigure	The coordinate system of the SubFigure ; (0, 0) is bottom left of the subfigure, and (1, 1) is top right of the subfigure. If a figure has no subfigures, this is the same as transFigure.
"figure"	fig.transFigure	The coordinate system of the Figure ; (0, 0) is bottom left of the figure, and (1, 1) is top right of the figure.
"figure-inches"	fig.dpi_scale_trans	The coordinate system of the Figure in inches; (0, 0) is bottom left of the figure, and (width, height) is the top right of the figure in inches.
"display"	None, or IdentityTransform()	The pixel coordinate system of the display window; (0, 0) is bottom left of the window, and (width, height) is top right of the display window in pixels.
"xaxis", "yaxis"	ax.get_xaxis_transform(), ax.get_yaxis_transform()	Blended coordinate systems; use data coordinates on one of the axis and axes coordinates on the other.

[*Matplotlib Release, 3.4.2, Page No. 227](#)

[*https://aosabook.org/en/matplotlib.html](https://aosabook.org/en/matplotlib.html)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

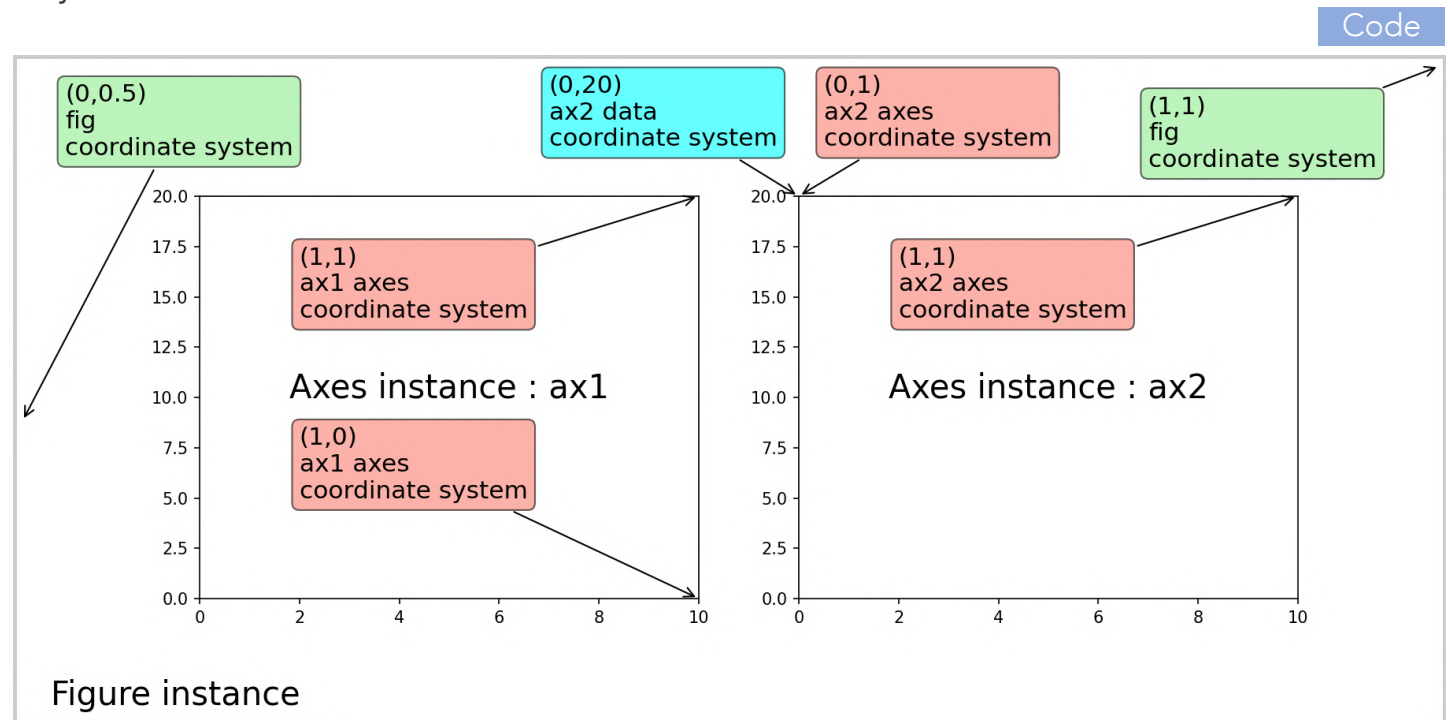
Add Legend, Annotations if any and Set Title to the Axes

TRANSFORMATION FRAMEWORK – THE COORDINATE SYSTEMS

The commonly used coordinate systems are :

- Data
- Axes
- Figure

All the above will be converted to Display coordinates by appropriate transformation objects.



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

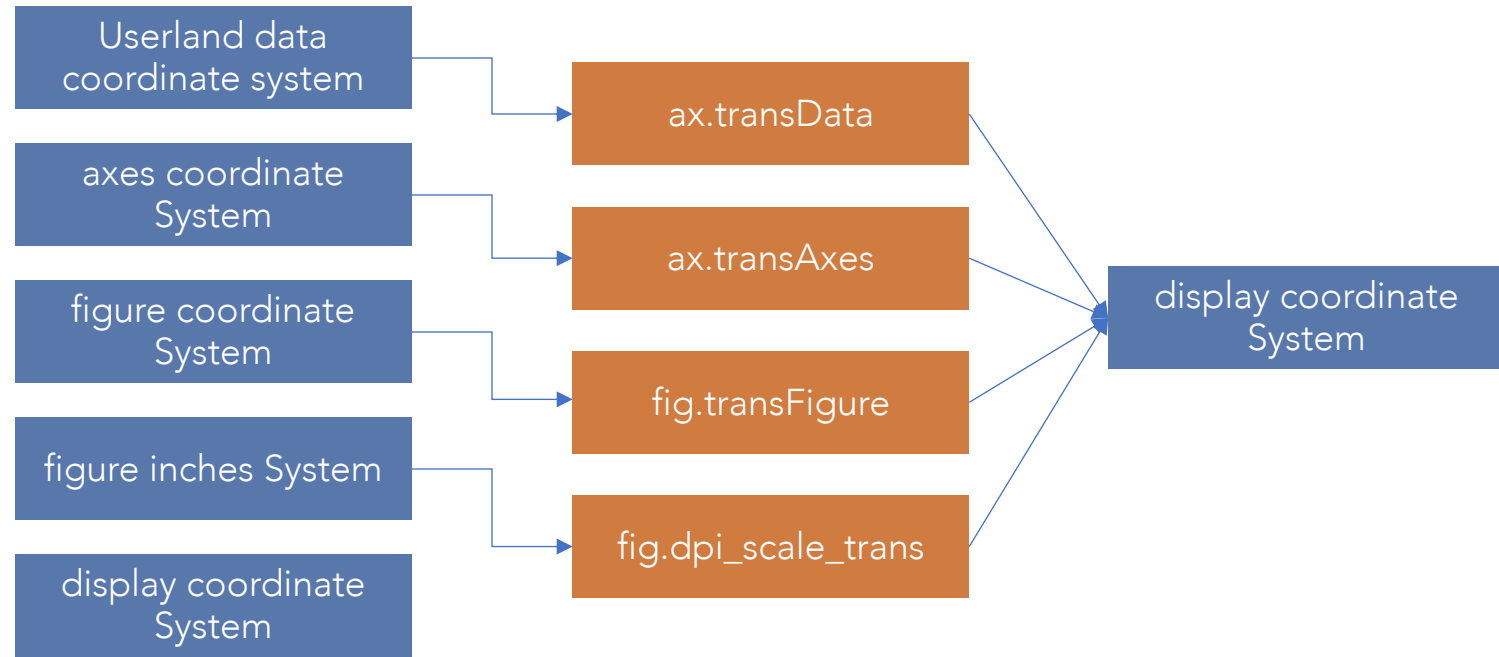
Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

TRANSFORMATION FRAMEWORK – TRANSFORMATION OBJECTS

Transformation Objects to easily move between coordinate systems



- All of the transformation objects in the table above take inputs in their coordinate system and transform the input to the display coordinate system.
- transform is the property common to all artists.
- While it works under the hood not requiring to be explicitly mentioned, but for customization and more control, it is immensely powerful.
- Use case :
 - Adding annotations/text
 - Adding horizontal/vertical spans
 - Legend positioning

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

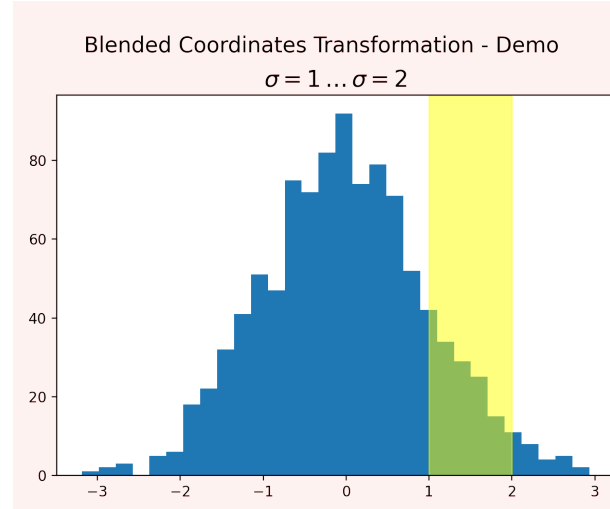
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

BLENDED TRANSFORMATIONS - EXAMPLE



x to be in data coordinates

y to span from 0 -1 in axes coords.

```
import numpy as np
import matplotlib.transforms as transforms
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

```
fig, ax = plt.subplots(figsize = (6,5))
x = np.random.randn(1000)
```

```
ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \vee \dots \vee \sigma=2$', fontsize=16)
```

```
# the x coords of this transformation are data, and the y
# coord are axes
trans = transforms.blended_transform_factory( ax.transData,
                                              ax.transAxes)
```

```
# highlight the 1..2 std dev region with a span.
# x to be in data coordinates
# y to span from 0..1 in axes coords.
rect = mpatches.Rectangle((1, 0), width=1, height=1,
                          transform=trans,
                          color='yellow', alpha=0.5)
```

```
ax.add_patch(rect)
ax.text(0.5,0.9, s = "Blended Coordinates Transformation -
                    Demo", transform = fig.transFigure,
       ha = 'center', fontsize = 15,
       fontweight = 'medium')
```

```
fig.patch.set(fc = 'salmon', alpha = 0.1)
plt.tight_layout(rect = [0,0,1,0.9])
```

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

COLORS

- 1
- Create a Figure instance
- 2
- Use Figure to create one or more Axes or Subplot
- 3
- Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4
- Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5
- Use Axes Methods (Accessors) to access artists.
- 6
- Set/Modify the Artist properties after accessing the artists
- 7
- Add Legend, Annotations if any and Set Title to the Axes

Specifying Colors – Acceptable Formats

	Format	Example
1.	RGB or RGBA (red, green, blue, alpha) tuple of float values in a closed interval [0, 1].	• (0.1, 0.2, 0.5) • (0.1, 0.2, 0.5, 0.3)
2.	Case-insensitive hex RGB or RGBA string .	• '#0f0f0f' • '#0f0f0f80'
3.	Case-insensitive RGB or RGBA string equivalent hex shorthand of duplicated characters.	• '#abc' as '#aabbcc' • '#fb1' as '#ffbb11'
4.	String representation of float value in closed interval [0, 1] for black and white, respectively.	• '0.8' as light gray • '0' as black • '1' as white
5.	Single character shorthand notation for shades of colors. Note : The colors green, cyan, magenta, and yellow do not coincide with X11/CSS4 colors.	• 'b' as blue • 'g' as green • 'r' as red • 'c' as cyan • 'm' as magenta • 'y' as yellow • 'k' as black • 'w' as white
6.	Case-insensitive X11/CSS4 color name with no spaces.	• 'aquamarine' • 'mediumseagreen'
7.	Case-insensitive color name from xkcd color survey with 'xkcd:' prefix.	• 'xkcd:sky blue' • 'xkcd:eggshell'

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

Format	Example
8. Case-insensitive Tableau Colors from 'T10' categorical palette. Note : This is the default color cycle.	'tab:blue' 'tab:orange' 'tab:green' 'tab:red' 'tab:purple' 'tab:brown' 'tab:pink' 'tab:gray' 'tab:olive' 'tab:cyan'
9. "CN" color spec where 'C' precedes a number acting as an index into the default property cycle. Note : Matplotlib indexes color at draw time and defaults to black if cycle does not include color. The default property cycle is rcParams["axes.prop_cycle"] (default: cycler('color', ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']))	'C0' 'C1'

- The alpha for an Artist controls opacity. It indicates how the RGB color of the new Artist combines with RGB colors already on the Axes. zorder of the artist also becomes important.

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

Exploring the Color Charts

- Basic (8 colors)
- Tableau Palette (10 colors)
- CSS colors (140+ colors)
- xkcd colors (954 colors)

<https://colorhunt.co/palette/>

Handy go to reference to pick colors

USE CASE SCENARIO	ARTIST PROPERTY
Assign face color, edge color of figure, axes	edgecolor (ec) , facecolor (fc)
Assign color to texts, axis labels, tick labels	color or c
Assign color to marker edge and face in line plot	Markeredgecolor (mec), markerfacecolor(mfc)
Initialize the color in bar plots, pie charts. Use Tableau palette. If more number of categories present, then colormaps and external libraries like [palettable] and [colorcet].	edgecolor (ec) , facecolor (fc)
Setting the property cycle of the axes (Note : The property cycle controls the style properties such as color, marker and linestyle of future plot commands. The style properties of data already added to the Axes are not modified.)	ax.set_prop_cycle(color=['red', 'green', 'blue'])

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

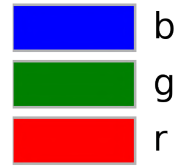
6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

Base Colors



b
g
r



c
m
y



k
w

Tableau Palette



tab:blue
tab:orange
tab:green
tab:red
tab:purple



tab:brown
tab:pink
tab:gray
tab:olive
tab:cyan

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

CSS Colors

	black
	dimgray
	dimgrey
	gray
	grey
	darkgray
	darkgrey
	silver
	lightgray
	lightgrey
	gainsboro
	whitesmoke
	white
	snow
	rosybrown
	lightcoral
	indianred
	brown
	firebrick
	maroon
	darkred
	red
	mistyrose
	salmon
	tomato
	darksalmon
	coral
	orangered
	lightsalmon
	sienna
	seashell
	chocolate
	saddlebrown
	sandybrown
	peachpuff
	peru
	linen

	bisque
	darkorange
	burlywood
	antiquewhite
	tan
	navajowhite
	blanchedalmond
	papayawhip
	moccasin
	orange
	wheat
	oldlace
	floralwhite
	darkgoldenrod
	goldenrod
	cornsilk
	gold
	lemonchiffon
	khaki
	palegoldenrod
	darkkhaki
	ivory
	beige
	lightyellow
	lightgoldenrodyellow
	olive
	yellow
	olivedrab
	yellowgreen
	darkolivegreen
	greenyellow
	chartreuse
	lawngreen
	honeydew
	darkseagreen
	palegreen
	lightgreen

	forestgreen
	limegreen
	darkgreen
	green
	lime
	seagreen
	mediumseagreen
	springgreen
	mintcream
	mediumspringgreen
	mediumaquamarine
	aquamarine
	turquoise
	lightseagreen
	mediumturquoise
	azure
	lightcyan
	paleturquoise
	darkslategray
	darkslategrey
	teal
	darkcyan
	aqua
	cyan
	darkturquoise
	cadetblue
	powderblue
	lightblue
	deepskyblue
	skyblue
	lightskyblue
	steelblue
	aliceblue
	dodgerblue
	lightslategray
	lightslategrey
	slategray

	slategrey
	lightsteelblue
	cornflowerblue
	royalblue
	ghostwhite
	lavender
	midnightblue
	navy
	darkblue
	mediumblue
	blue
	slateblue
	darkslateblue
	mediumslateblue
	mediumpurple
	rebeccapurple
	blueviolet
	indigo
	darkorchid
	darkviolet
	mediumorchid
	thistle
	plum
	violet
	purple
	darkmagenta
	fuchsia
	magenta
	orchid
	mediumvioletred
	deeppink
	hotpink
	lavenderblush
	palevioletred
	crimson
	pink
	lightpink

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:black		xkcd:orange red		xkcd:warm grey		xkcd:orange
	xkcd:white		xkcd:blush		xkcd:brownish		xkcd:apricot
	xkcd:dull red		xkcd:vermillion		xkcd:rust		xkcd:sepia
	xkcd:dried blood		xkcd:orange pink		xkcd:russet		xkcd:dull orange
	xkcd:dark red		xkcd:tomato red		xkcd:chestnut		xkcd:pale orange
	xkcd:red		xkcd:burnt red		xkcd:rust brown		xkcd:pumpkin orange
	xkcd:deep red		xkcd:reddish orange		xkcd:deep orange		xkcd:mocha
	xkcd:mahogany		xkcd:orangish red		xkcd:brick orange		xkcd:milk chocolate
	xkcd:pastel red		xkcd:red brown		xkcd:bright orange		xkcd:light peach
	xkcd:reddish		xkcd:light salmon		xkcd:burnt umber		xkcd:brownish orange
	xkcd:grapefruit		xkcd:melon		xkcd:orangeish		xkcd:warm brown
	xkcd:deep brown		xkcd:rusty red		xkcd:chocolate brown		xkcd:dark brown
	xkcd:dark coral		xkcd:rust red		xkcd:earth		xkcd:pale brown
	xkcd:pale red		xkcd:pinkish orange		xkcd:burnt sienna		xkcd:brownny orange
	xkcd:coral		xkcd:pinkish brown		xkcd:peach		xkcd:orangish brown
	xkcd:dark salmon		xkcd:orangered		xkcd:dusty orange		xkcd:orange brown
	xkcd:brownish pink		xkcd:red orange		xkcd:sienna		xkcd:tan brown
	xkcd:very dark brown		xkcd:pale salmon		xkcd:dark orange		xkcd:pumpkin
	xkcd:indian red		xkcd:clay		xkcd:burnt orange		xkcd:light brown
	xkcd:salmon		xkcd:dark peach		xkcd:pastel orange		xkcd:puce
	xkcd:pinkish grey		xkcd:brown red		xkcd:rusty orange		xkcd:dark taupe
	xkcd:reddy brown		xkcd:terracotta		xkcd:rust orange		xkcd:leather
	xkcd:reddish grey		xkcd:terracota		xkcd:cocoa		xkcd:orangeey brown
	xkcd:brick red		xkcd:reddish brown		xkcd:copper		xkcd:raw umber
	xkcd:tomato		xkcd:blood orange		xkcd:faded orange		xkcd:light orange
	xkcd:peachy pink		xkcd:pinkish tan		xkcd:burnt sienna		xkcd:brown
	xkcd:orangeey red		xkcd:terra cotta		xkcd:cinnamon		xkcd:umber
	xkcd:brick		xkcd:auburn		xkcd:mushroom		xkcd:brown orange
	xkcd:very light pink		xkcd:adobe		xkcd:chocolate		xkcd:tangerine
	xkcd:brownish red		xkcd:orangish		xkcd:clay brown		xkcd:dirty orange

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:medium brown		xkcd:tan		xkcd:ocre		xkcd:sunflower yellow
	xkcd:mango		xkcd:saffron		xkcd:yellowy brown		xkcd:sun yellow
	xkcd:butterscotch		xkcd:putty		xkcd:yellowish brown		xkcd:mustard
	xkcd:dull brown		xkcd:amber		xkcd:pale gold		xkcd:pale
	xkcd:coffee		xkcd:poo brown		xkcd:stone		xkcd:brownish yellow
	xkcd:taupe		xkcd:sandy brown		xkcd:greyish		xkcd:dandelion
	xkcd:dirt		xkcd:yellow orange		xkcd:burnt yellow		xkcd:dull yellow
	xkcd:dirt brown		xkcd:shit brown		xkcd:light gold		xkcd:dark cream
	xkcd:dark tan		xkcd:orangey yellow		xkcd:puke brown		xkcd:sandy yellow
	xkcd:caramel		xkcd:desert		xkcd:hazel		xkcd:mustard yellow
	xkcd:brownish grey		xkcd:bronze		xkcd:ocher		xkcd:muddy yellow
	xkcd:fawn		xkcd:mustard brown		xkcd:dark gold		xkcd:cement
	xkcd:greyish brown		xkcd:poop brown		xkcd:poo		xkcd:ugly brown
	xkcd:dust		xkcd:poop		xkcd:bland		xkcd:greenish brown
	xkcd:toupe		xkcd:golden rod		xkcd:sandy		xkcd:greeny brown
	xkcd:raw sienna		xkcd:ochre		xkcd:yellow tan		xkcd:buff
	xkcd:very light brown		xkcd:shit		xkcd:yellow brown		xkcd:yellowish
	xkcd:camel		xkcd:muddy brown		xkcd:dark mustard		xkcd:green brown
	xkcd:sand brown		xkcd:sunflower		xkcd:gold		xkcd:ugly yellow
	xkcd:yellowish orange		xkcd:marigold		xkcd:beige		xkcd:olive yellow
	xkcd:grey brown		xkcd:brown grey		xkcd:baby shit brown		xkcd:khaki
	xkcd:dark beige		xkcd:golden yellow		xkcd:sand yellow		xkcd:egg shell
	xkcd:orange yellow		xkcd:wheat		xkcd:diarrhea		xkcd:straw
	xkcd:squash		xkcd:mud		xkcd:dark khaki		xkcd:brown green
	xkcd:mud brown		xkcd:yellow ochre		xkcd:olive brown		xkcd:manilla
	xkcd:sandstone		xkcd:goldenrod		xkcd:light tan		xkcd:dirty yellow
	xkcd:macaroni and cheese		xkcd:light mustard		xkcd:baby poo		xkcd:piss yellow
	xkcd:pale peach		xkcd:maize		xkcd:baby poop		xkcd:vomit yellow
	xkcd:dark sand		xkcd:golden		xkcd:brown yellow		xkcd:browny green
	xkcd:golden brown		xkcd:sand		xkcd:dark yellow		xkcd:sunny yellow

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

 xkcd:parchment	 xkcd:brownish green	 xkcd:snot green	 xkcd:dark lime
 xkcd:puke yellow	 xkcd:pea soup	 xkcd:pea green	 xkcd:camouflage green
 xkcd:custard	 xkcd:mud green	 xkcd:neon yellow	 xkcd:yellow green
 xkcd:butter yellow	 xkcd:baby poop green	 xkcd:greenish yellow	 xkcd:dirty green
 xkcd:light beige	 xkcd:olive	 xkcd:ugly green	 xkcd:pear
 xkcd:sunshine yellow	 xkcd:mustard green	 xkcd:sick green	 xkcd:lemon lime
 xkcd:bright yellow	 xkcd:baby puke green	 xkcd:sickly green	 xkcd:camo green
 xkcd:light yellow	 xkcd:bile	 xkcd:lime yellow	 xkcd:lemon green
 xkcd:pastel yellow	 xkcd:shit green	 xkcd:dark yellow green	 xkcd:dark lime green
 xkcd:canary yellow	 xkcd:snot	 xkcd:greeny yellow	 xkcd:electric lime
 xkcd:off white	 xkcd:greenish beige	 xkcd:booger	 xkcd:swamp
 xkcd:eggshell	 xkcd:olive drab	 xkcd:light olive	 xkcd:military green
 xkcd:ivory	 xkcd:poop green	 xkcd:icky green	 xkcd:pale olive green
 xkcd:cream	 xkcd:sickly yellow	 xkcd:yellowish green	 xkcd:bright yellow green
 xkcd:creme	 xkcd:dark olive	 xkcd:muddy green	 xkcd:light yellow green
 xkcd:pale yellow	 xkcd:baby shit green	 xkcd:dark olive green	 xkcd:sap green
 xkcd:yellowish tan	 xkcd:puke green	 xkcd:chartreuse	 xkcd:mossy green
 xkcd:butter	 xkcd:pea soup green	 xkcd:camo	 xkcd:light moss green
 xkcd:banana	 xkcd:green/yellow	 xkcd:yellowy green	 xkcd:navy green
 xkcd:yellow	 xkcd:swamp green	 xkcd:green yellow	 xkcd:lime
 xkcd:puke	 xkcd:murky green	 xkcd:avocado green	 xkcd:acid green
 xkcd:faded yellow	 xkcd:barf green	 xkcd:pale olive	 xkcd:pale lime
 xkcd:lemon yellow	 xkcd:light khaki	 xkcd:army green	 xkcd:light lime green
 xkcd:off yellow	 xkcd:vomit green	 xkcd:slime green	 xkcd:moss green
 xkcd:lemon	 xkcd:olive green	 xkcd:khaki green	 xkcd:leaf green
 xkcd:canary	 xkcd:bright olive	 xkcd:avocado	 xkcd:light pea green
 xkcd:vomit	 xkcd:booger green	 xkcd:yellowgreen	 xkcd:lime green
 xkcd:drab	 xkcd:pea	 xkcd:light olive green	 xkcd:bright lime
 xkcd:ecru	 xkcd:gross green	 xkcd:tan green	 xkcd:kiwi
 xkcd:banana yellow	 xkcd:greenish tan	 xkcd:yellow/green	 xkcd:leaf

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:kermit green		xkcd:light grey		xkcd:greyish green		xkcd:light neon green
	xkcd:drab green		xkcd:tea green		xkcd:lighter green		xkcd:lightgreen
	xkcd:pale lime green		xkcd:toxic green		xkcd:faded green		xkcd:light bright green
	xkcd:light yellowish green		xkcd:light light green		xkcd:easter green		xkcd:light forest green
	xkcd:apple green		xkcd:very light green		xkcd:greenish grey		xkcd:light mint
	xkcd:pistachio		xkcd:off green		xkcd:celadon		xkcd:soft green
	xkcd:kiwi green		xkcd:very pale green		xkcd:mid green		xkcd:dark forest green
	xkcd:moss		xkcd:washed out green		xkcd:highlighter green		xkcd:forest green
	xkcd:light lime		xkcd:greenish grey		xkcd:electric green		xkcd:british racing green
	xkcd:frog green		xkcd:sage green		xkcd:very dark green		xkcd:medium green
	xkcd:key lime		xkcd:dull green		xkcd:dark sage		xkcd:light mint green
	xkcd:lawn green		xkcd:grey/green		xkcd:radioactive green		xkcd:mint green
	xkcd:nasty green		xkcd:light sage		xkcd:dark green		xkcd:deep green
	xkcd:celery		xkcd:pale green		xkcd:dusty green		xkcd:baby green
	xkcd:dark grass green		xkcd:grey		xkcd:hunter green		xkcd:light seafoam green
	xkcd:spring green		xkcd:pale light green		xkcd:fluro green		xkcd:darkish green
	xkcd:grassy green		xkcd:forrest green		xkcd:true green		xkcd:mint
	xkcd:asparagus		xkcd:green grey		xkcd:forest		xkcd:pine
	xkcd:bright lime green		xkcd:fern green		xkcd:racing green		xkcd:bright light green
	xkcd:grass		xkcd:light green		xkcd:vibrant green		xkcd:emerald green
	xkcd:light grass green		xkcd:fern		xkcd:lightish green		xkcd:slate green
	xkcd:turtle green		xkcd:pastel green		xkcd:neon green		xkcd:hospital green
	xkcd:grass green		xkcd:fresh green		xkcd:fluorescent green		xkcd:algae
	xkcd:flat green		xkcd:poison green		xkcd:dark pastel green		xkcd:foam green
	xkcd:apple		xkcd:leafy green		xkcd:bottle green		xkcd:light sea green
	xkcd:light grey green		xkcd:tree green		xkcd:hot green		xkcd:kelly green
	xkcd:lichen		xkcd:muted green		xkcd:bright green		xkcd:irish green
	xkcd:sage		xkcd:light pastel green		xkcd:boring green		xkcd:pine green
	xkcd:green apple		xkcd:vivid green		xkcd:darkgreen		xkcd:tea
	xkcd:medium grey		xkcd:grey green		xkcd:green		xkcd:light seafoam

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:dark mint		xkcd:evergreen		xkcd:green blue		xkcd:very light blue
	xkcd:cool green		xkcd:greenish teal		xkcd:bright teal		xkcd:really light blue
	xkcd:light bluish green		xkcd:bluey green		xkcd:tiffany blue		xkcd:pale blue
	xkcd:seafoam		xkcd:light teal		xkcd:aquamarine		xkcd:almost black
	xkcd:kelly green		xkcd:light aquamarine		xkcd:dark green blue		xkcd:cyan
	xkcd:seafoam green		xkcd:ocean green		xkcd:dusty teal		xkcd:bright cyan
	xkcd:shamrock green		xkcd:teal green		xkcd:blue green		xkcd:deep turquoise
	xkcd:weird green		xkcd:dark seafoam		xkcd:aqua		xkcd:dark teal
	xkcd:spearmint		xkcd:aqua green		xkcd:eggshell blue		xkcd:dark cyan
	xkcd:greenish		xkcd:jade		xkcd:tealish		xkcd:dark aqua
	xkcd:shamrock		xkcd:green teal		xkcd:duck egg blue		xkcd:bright light blue
	xkcd:light blue green		xkcd:viridian		xkcd:ice		xkcd:light sky blue
	xkcd:seaweed green		xkcd:bright sea green		xkcd:turquoise		xkcd:deep teal
	xkcd:emerald		xkcd:dark sea green		xkcd:dark blue green		xkcd:deep aqua
	xkcd:light green blue		xkcd:greenblue		xkcd:blue/green		xkcd:aqua blue
	xkcd:sea green		xkcd:pale teal		xkcd:teal		xkcd:turquoise blue
	xkcd:wintergreen		xkcd:light turquoise		xkcd:light light blue		xkcd:robin's egg blue
	xkcd:jade green		xkcd:greenish cyan		xkcd:sea		xkcd:petrol
	xkcd:algae green		xkcd:bluish green		xkcd:topaz		xkcd:robin's egg
	xkcd:minty green		xkcd:light aqua		xkcd:pale cyan		xkcd:robin egg blue
	xkcd:tealish green		xkcd:greyish teal		xkcd:bright aqua		xkcd:pale sky blue
	xkcd:dark seafoam green		xkcd:seafoam blue		xkcd:light cyan		xkcd:teal blue
	xkcd:silver		xkcd:greenish turquoise		xkcd:greenish blue		xkcd:neon blue
	xkcd:jungle green		xkcd:pale aqua		xkcd:ice blue		xkcd:ocean
	xkcd:dark mint green		xkcd:grey teal		xkcd:very pale blue		xkcd:charcoal grey
	xkcd:seaweed		xkcd:green/blue		xkcd:dark turquoise		xkcd:cool grey
	xkcd:spruce		xkcd:greenish blue		xkcd:bright turquoise		xkcd:bright sky blue
	xkcd:light greenish blue		xkcd:charcoal		xkcd:dark aquamarine		xkcd:sea blue
	xkcd:turquoise green		xkcd:aqua marine		xkcd:bluegreen		xkcd:gunmetal
	xkcd:pale turquoise		xkcd:dull teal		xkcd:dark grey		xkcd:ocean blue

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:dirty blue		xkcd:steel blue		xkcd:powder blue		xkcd:royal
	xkcd:steel grey		xkcd:muted blue		xkcd:light blue grey		xkcd:dark navy
	xkcd:peacock blue		xkcd:slate blue		xkcd:dark		xkcd:true blue
	xkcd:greyblue		xkcd:cool blue		xkcd:bright blue		xkcd:twilight
	xkcd:bluegrey		xkcd:light navy		xkcd:carolina blue		xkcd:dark navy blue
	xkcd:nice blue		xkcd:dusty blue		xkcd:clear blue		xkcd:cobalt blue
	xkcd:bluish grey		xkcd:dull blue		xkcd:french blue		xkcd:darkblue
	xkcd:battleship grey		xkcd:steel		xkcd:cobalt		xkcd:dark royal blue
	xkcd:deep sea blue		xkcd:cadet blue		xkcd:denim blue		xkcd:dark blue
	xkcd:water blue		xkcd:bluish		xkcd:dodger blue		xkcd:very dark blue
	xkcd:cerulean		xkcd:marine blue		xkcd:navy		xkcd:pure blue
	xkcd:lightblue		xkcd:off blue		xkcd:lightish blue		xkcd:pale grey
	xkcd:azure		xkcd:blue/grey		xkcd:pastel blue		xkcd:royal blue
	xkcd:blue grey		xkcd:sky blue		xkcd:dusky blue		xkcd:night blue
	xkcd:dark blue grey		xkcd:flat blue		xkcd:azul		xkcd:primary blue
	xkcd:slate		xkcd:twilight blue		xkcd:electric blue		xkcd:deep blue
	xkcd:slate grey		xkcd:darkish blue		xkcd:blue		xkcd:strong blue
	xkcd:ugly blue		xkcd:denim		xkcd:soft blue		xkcd:lavender blue
	xkcd:grey/blue		xkcd:baby blue		xkcd:navy blue		xkcd:perrywinkle
	xkcd:dark slate blue		xkcd:light navy blue		xkcd:cornflower blue		xkcd:midnight blue
	xkcd:sky		xkcd:cloudy blue		xkcd:vibrant blue		xkcd:midnight
	xkcd:bluey grey		xkcd:mid blue		xkcd:blue blue		xkcd:light royal blue
	xkcd:dark grey blue		xkcd:windows blue		xkcd:sapphire		xkcd:blueberry
	xkcd:prussian blue		xkcd:medium blue		xkcd:dusk		xkcd:dark periwinkle
	xkcd:light blue		xkcd:dark sky blue		xkcd:vivid blue		xkcd:ultramarine blue
	xkcd:metallic blue		xkcd:marine		xkcd:cornflower		xkcd:iris
	xkcd:stormy blue		xkcd:faded blue		xkcd:rich blue		xkcd:periwinkle
	xkcd:light grey blue		xkcd:cerulean blue		xkcd:periwinkle blue		xkcd:light indigo
	xkcd:grey blue		xkcd:deep sky blue		xkcd:light periwinkle		xkcd:blue with a hint of purple
	xkcd:greyish blue		xkcd:dusk blue		xkcd:warm blue		xkcd:ultramarine

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

XKCD Colors

	xkcd:purpleish blue		xkcd:deep lavender		xkcd:faded purple		xkcd:light eggplant
	xkcd:blurple		xkcd:easter purple		xkcd:dark violet		xkcd:eggplant
	xkcd:indigo blue		xkcd:lightish purple		xkcd:hot purple		xkcd:lavender pink
	xkcd:purpley blue		xkcd:light purple		xkcd:heliotrope		xkcd:purply pink
	xkcd:bluey purple		xkcd:pale purple		xkcd:dark lilac		xkcd:pinky purple
	xkcd:blue purple		xkcd:vivid purple		xkcd:dark purple		xkcd:fuchsia
	xkcd:purple blue		xkcd:dark lavender		xkcd:muted purple		xkcd:bright magenta
	xkcd:dark indigo		xkcd:electric purple		xkcd:deep purple		xkcd:dusty lavender
	xkcd:purplish blue		xkcd:amethyst		xkcd:very light purple		xkcd:purpley pink
	xkcd:burple		xkcd:wisteria		xkcd:dusty purple		xkcd:purplish
	xkcd:bluish purple		xkcd:violet		xkcd:barney		xkcd:dull purple
	xkcd:purple/blue		xkcd:bright lavender		xkcd:medium purple		xkcd:candy pink
	xkcd:purpley		xkcd:pale lavender		xkcd:purple/pink		xkcd:purpleish pink
	xkcd:purply blue		xkcd:bright violet		xkcd:plum purple		xkcd:grape purple
	xkcd:blue/purple		xkcd:light lilac		xkcd:pinkish purple		xkcd:purpleish
	xkcd:violet blue		xkcd:grey purple		xkcd:violet pink		xkcd:aubergine
	xkcd:blue violet		xkcd:bright lilac		xkcd:purpley grey		xkcd:hot magenta
	xkcd:pale violet		xkcd:royal purple		xkcd:pink purple		xkcd:grape
	xkcd:indigo		xkcd:greyish purple		xkcd:deep violet		xkcd:bruise
	xkcd:pastel purple		xkcd:neon purple		xkcd:light magenta		xkcd:rich purple
	xkcd:light lavender		xkcd:midnight purple		xkcd:darkish purple		xkcd:purplish pink
	xkcd:light violet		xkcd:bright purple		xkcd:eggplant purple		xkcd:dusky purple
	xkcd:pale lilac		xkcd:light lavender		xkcd:pink/purple		xkcd:bright pink
	xkcd:lilac		xkcd:purple		xkcd:ugly purple		xkcd:dark plum
	xkcd:liliac		xkcd:purply		xkcd:purple grey		xkcd:plum
	xkcd:lighter purple		xkcd:vibrant purple		xkcd:pale mauve		xkcd:velvet
	xkcd:lavender		xkcd:purplish grey		xkcd:orchid		xkcd:dirty purple
	xkcd:light urple		xkcd:soft purple		xkcd:purple pink		xkcd:red violet
	xkcd:deep lilac		xkcd:heather		xkcd:barney purple		xkcd:shocking pink
	xkcd:baby purple		xkcd:very dark purple		xkcd:warm purple		xkcd:light plum

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

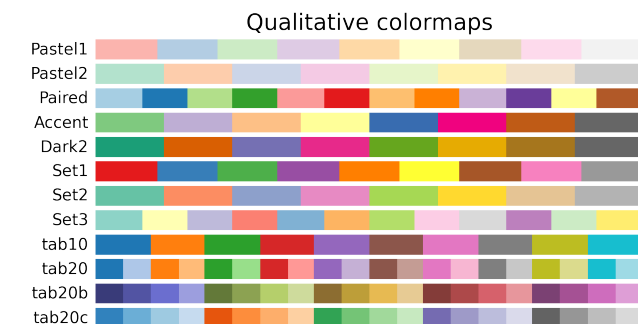
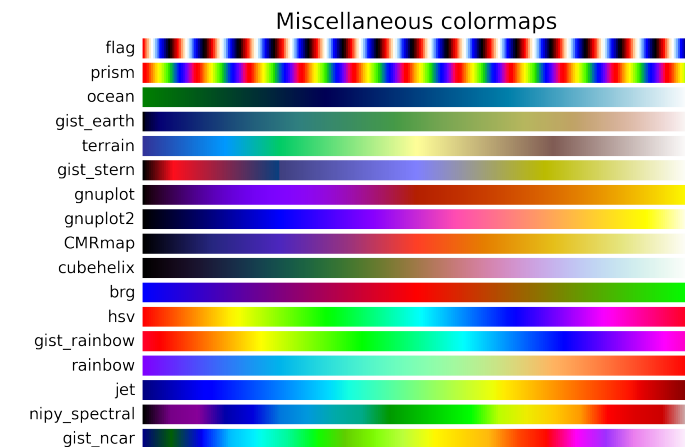
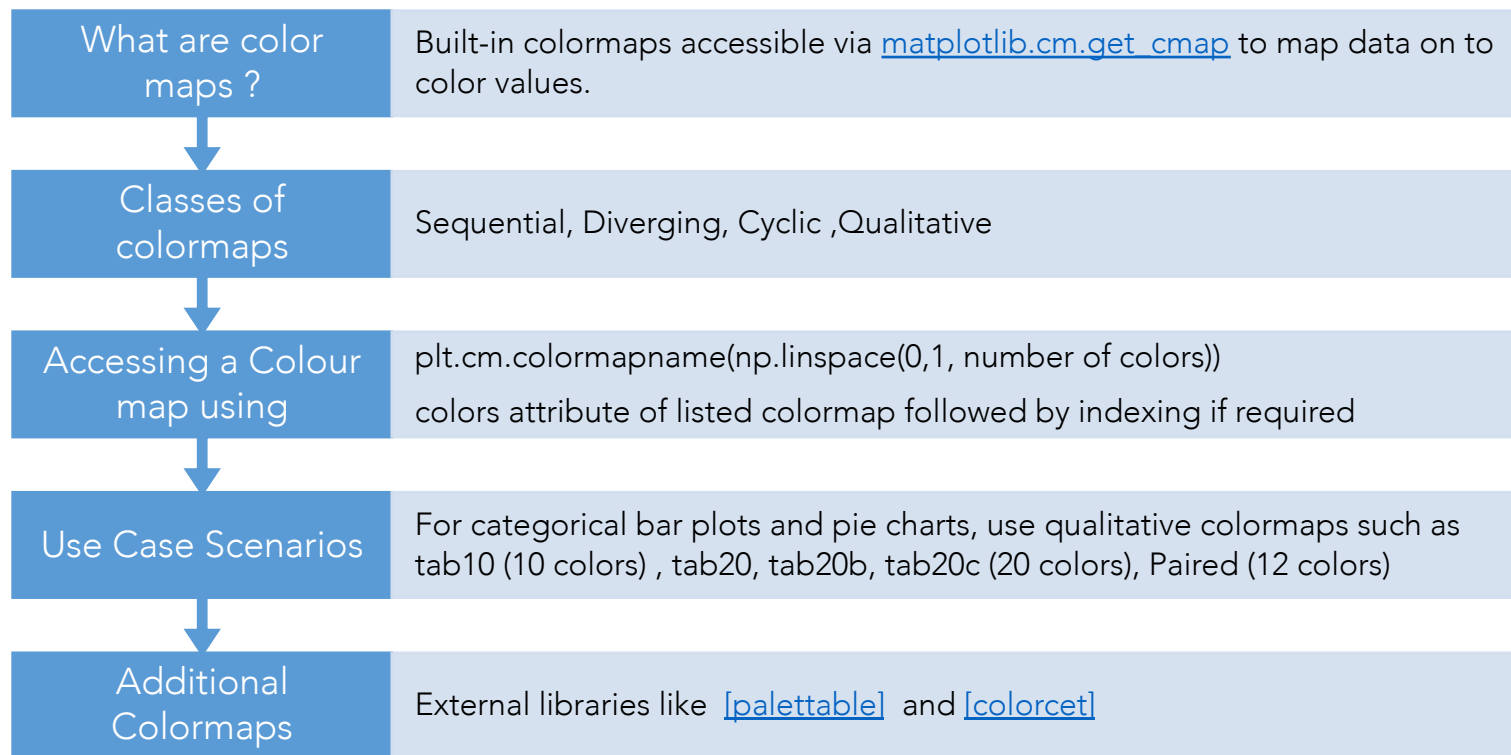
7

Add Legend, Annotations if any and Set Title to the Axes

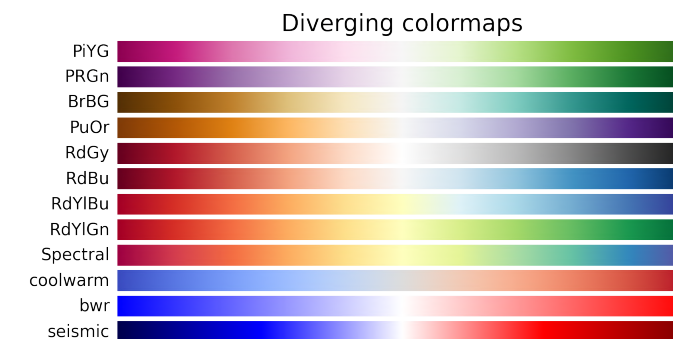
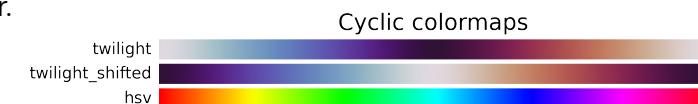
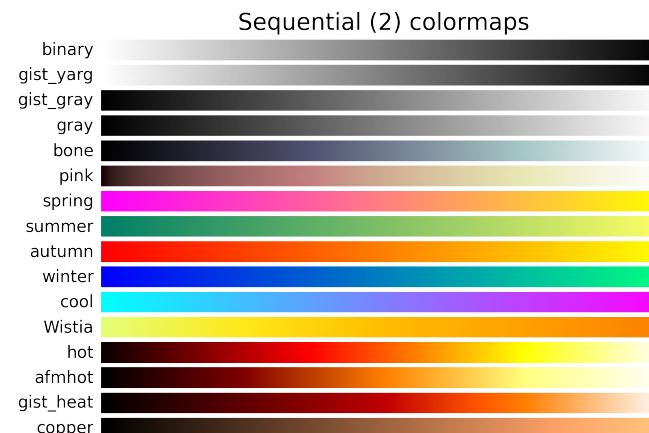
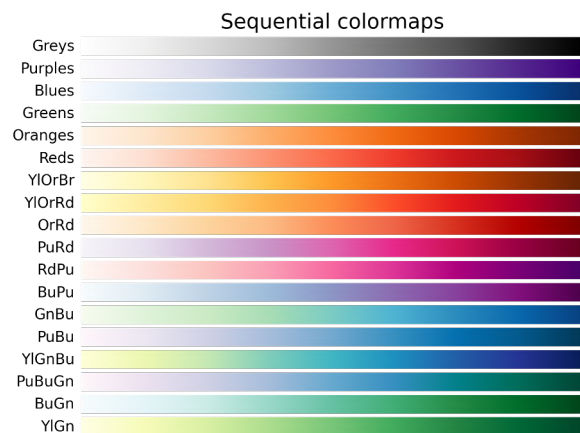
XKCD Colors

Code

	xkcd:pale magenta		xkcd:red wine		xkcd:cherry		xkcd:greyish pink
	xkcd:magenta		xkcd:cranberry		xkcd:lipstick red		xkcd:purple brown
	xkcd:neon pink		xkcd:medium pink		xkcd:faded pink		xkcd:crimson
	xkcd:bubblegum pink		xkcd:burgundy		xkcd:brownish purple		xkcd:watermelon
	xkcd:dark magenta		xkcd:bordeaux		xkcd:claret		xkcd:light rose
	xkcd:deep magenta		xkcd:ruby		xkcd:grey pink		xkcd:dusty rose
	xkcd:electric pink		xkcd:maroon		xkcd:soft pink		xkcd:blush pink
	xkcd:hot pink		xkcd:baby pink		xkcd:dusky pink		xkcd:pale rose
	xkcd:dark fuchsia		xkcd:light mauve		xkcd:red pink		xkcd:dirty pink
	xkcd:reddish purple		xkcd:carnation pink		xkcd:neon red		xkcd:dusty red
	xkcd:strong pink		xkcd:rose red		xkcd:dark rose		xkcd:bright red
	xkcd:red purple		xkcd:pink red		xkcd:reddish pink		xkcd:faded red
	xkcd:barbie pink		xkcd:pinky		xkcd:pinkish		xkcd:darkish red
	xkcd:violet red		xkcd:light pink		xkcd:rose		xkcd:light red
	xkcd:pink		xkcd:dark pink		xkcd:ugly pink		xkcd:blood red
	xkcd:mulberry		xkcd:dull pink		xkcd:rose pink		xkcd:coral pink
	xkcd:bubblegum		xkcd:lipstick		xkcd:old pink		xkcd:blood
	xkcd:merlot		xkcd:pastel pink		xkcd:carnation		xkcd:fire engine red
	xkcd:wine		xkcd:muted pink		xkcd:light maroon		xkcd:salmon pink
	xkcd:deep pink		xkcd:pale pink		xkcd:cherry red		
	xkcd:dark hot pink		xkcd:rosy pink		xkcd:pinky red		
	xkcd:bubble gum pink		xkcd:wine red		xkcd:old rose		
	xkcd:purple red		xkcd:warm pink		xkcd:dusky rose		
	xkcd:berry		xkcd:mauve		xkcd:dusty pink		
	xkcd:raspberry		xkcd:pig pink		xkcd:dark maroon		
	xkcd:cerise		xkcd:rouge		xkcd:lightish red		
	xkcd:purplish red		xkcd:light burgundy		xkcd:carmine		
	xkcd:powder pink		xkcd:rosa		xkcd:scarlet		
	xkcd:dark mauve		xkcd:deep rose		xkcd:strawberry		
	xkcd:darkish pink		xkcd:pinkish red		xkcd:purplish brown		



Note : Colormaps is a very rich immersive topic. Check out the official matplotlib website to dwelve deeper.

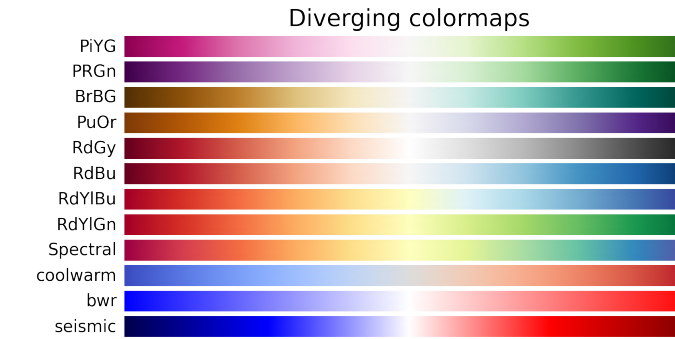
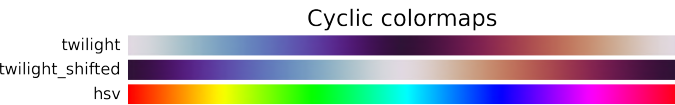
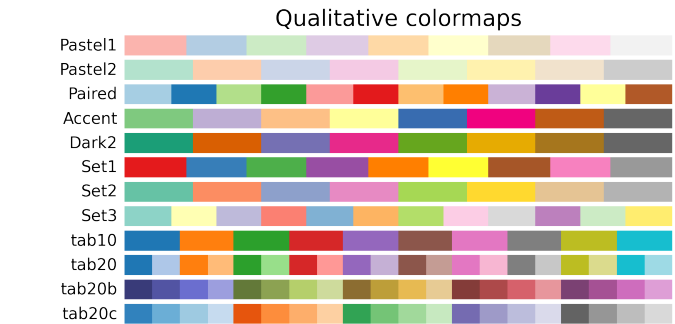
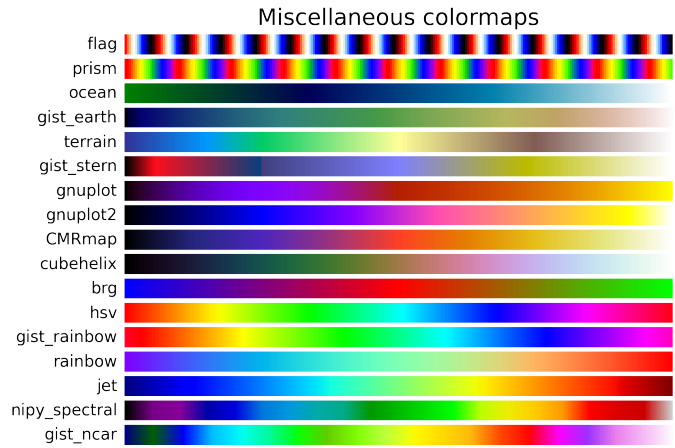
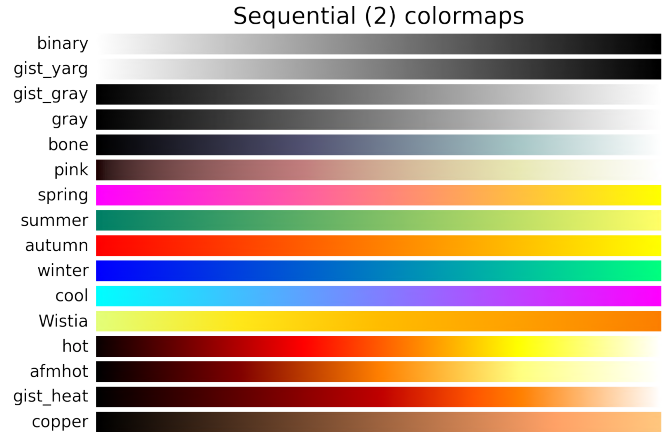
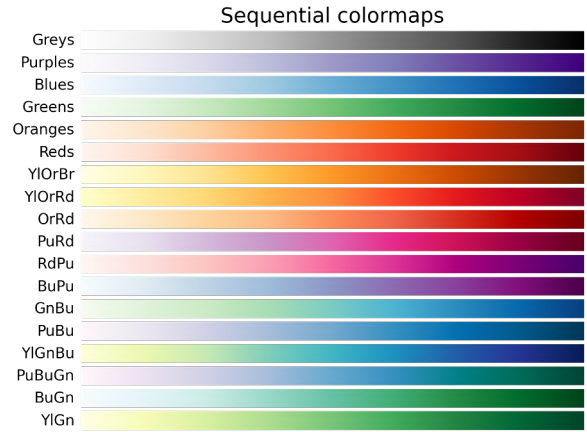


Classes of colormaps Based on Function

Colormap Category	Use Case
Sequential: change in lightness and often saturation of color incrementally, often using a single hue;	for representing information that has ordering.
Diverging: change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color	when the information being plotted has a critical middle value, such as topography or when the data deviates around zero.
Cyclic: change in lightness of two different colors that meet in the middle and beginning/end at an unsaturated color;	for values that wrap around at the endpoints, such as phase angle, wind direction, or time of day.
Qualitative: often are miscellaneous colors	to represent information which does not have ordering or relationships.

Note : Colormaps is a very rich immersive topic. Check out the official matplotlib website to dwelve deeper.

<https://matplotlib.org/stable/tutorials/colors/colormaps.html>
https://matplotlib.org/stable/api/pyplot_summary.html?highlight=pyplot

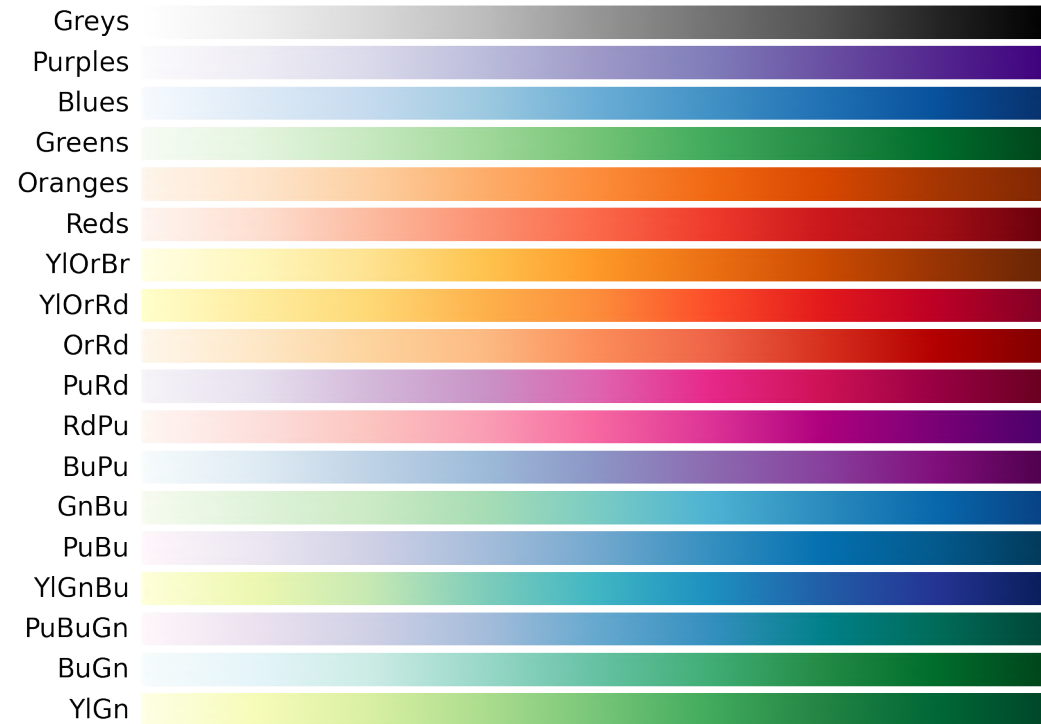


Range of Matplotlib Colormaps

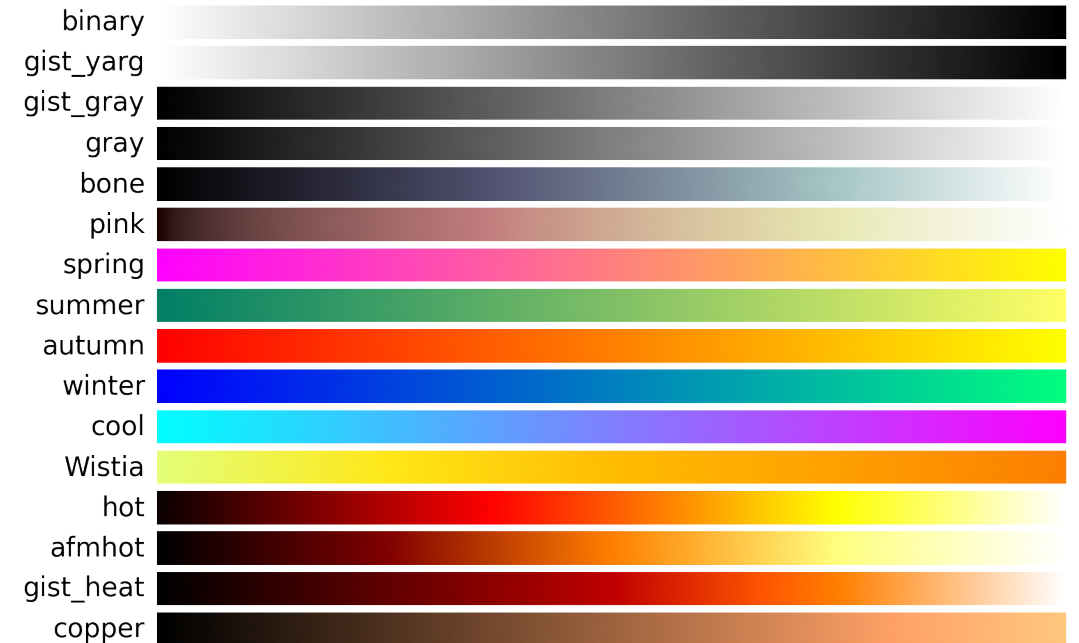
Perceptually Uniform Sequential colormaps



Sequential colormaps

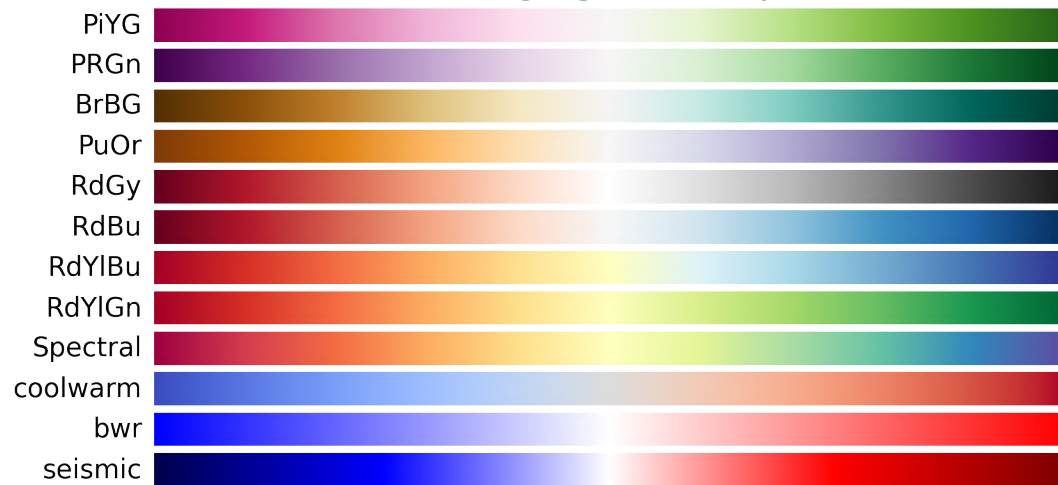


Sequential (2) colormaps

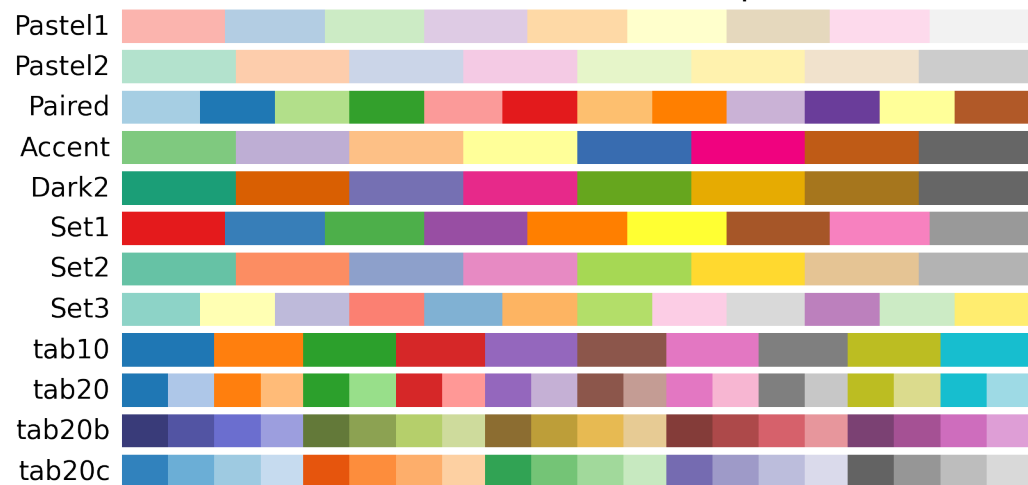


Note : All colormaps can be reversed by appending _r.

Diverging colormaps



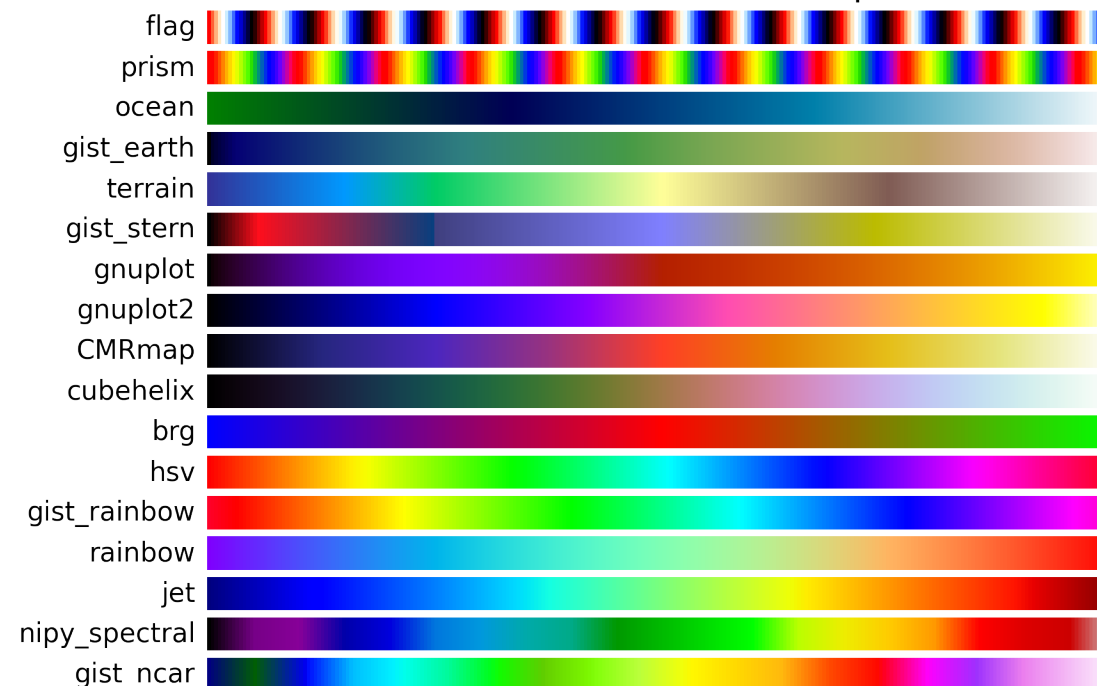
Qualitative colormaps



Cyclic colormaps

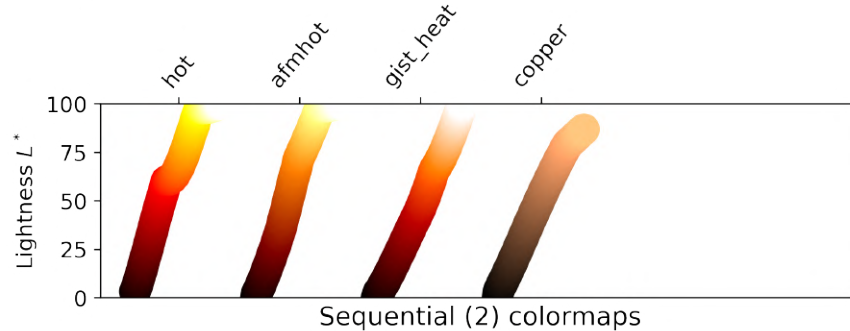
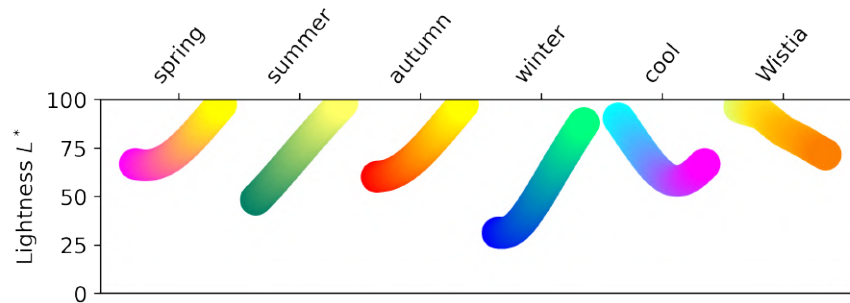
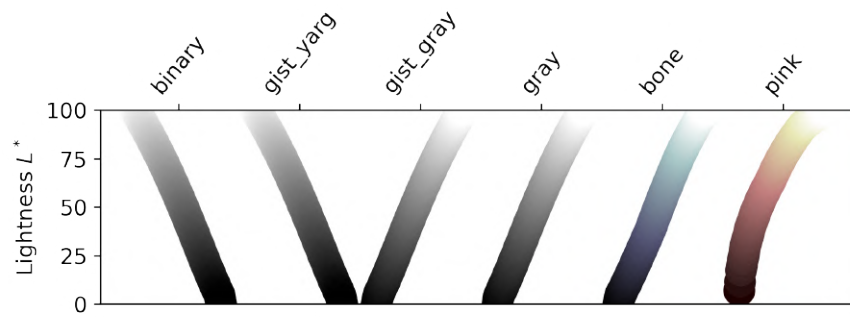
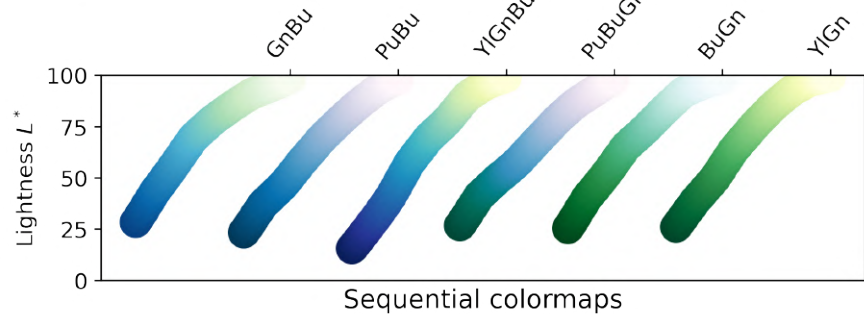
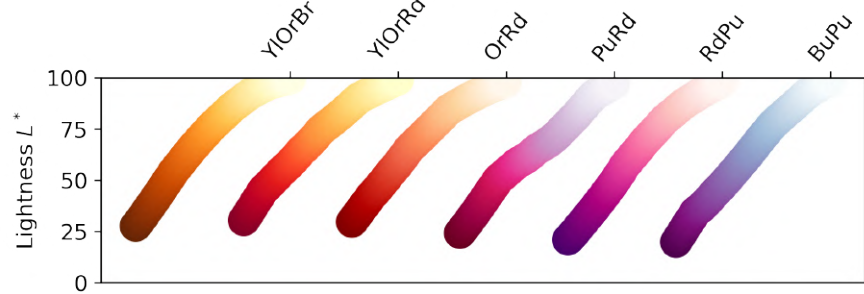
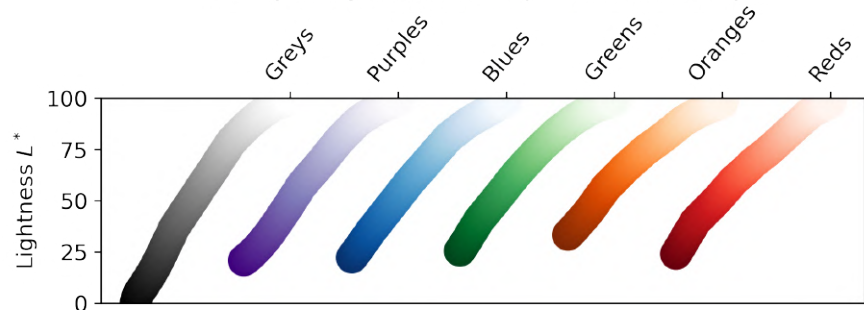
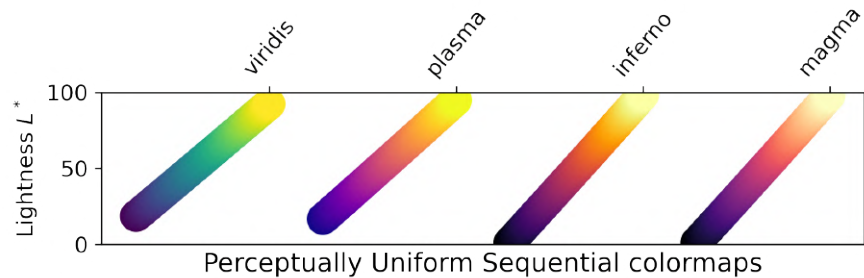


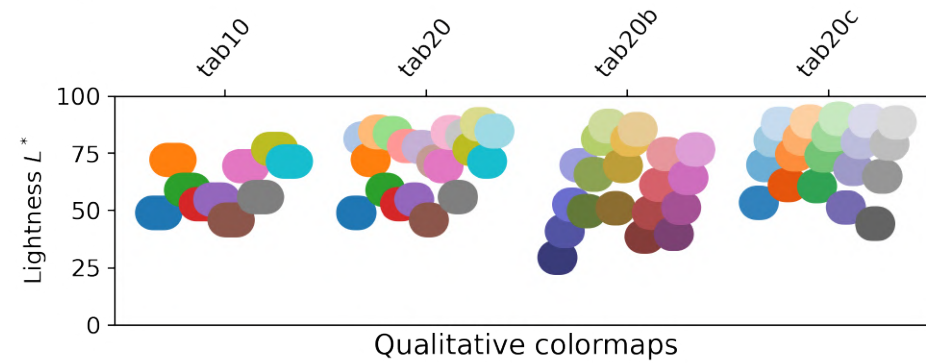
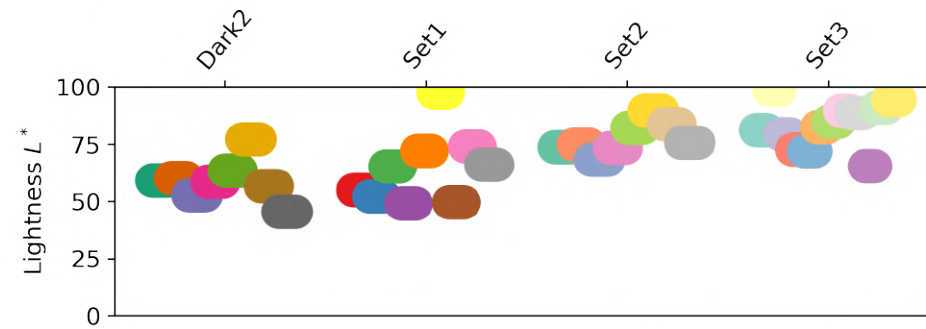
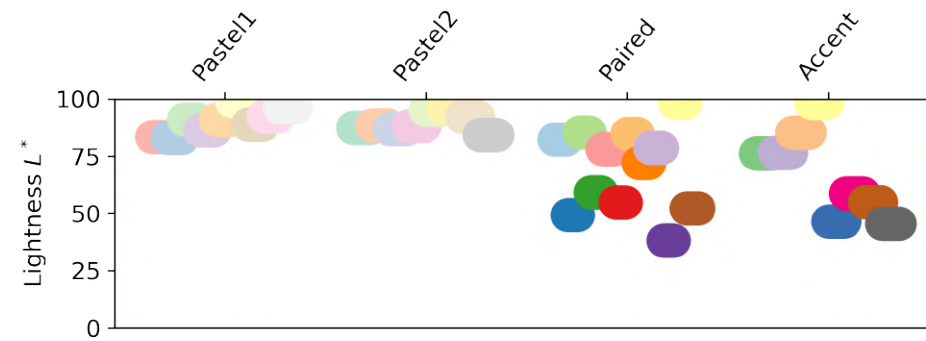
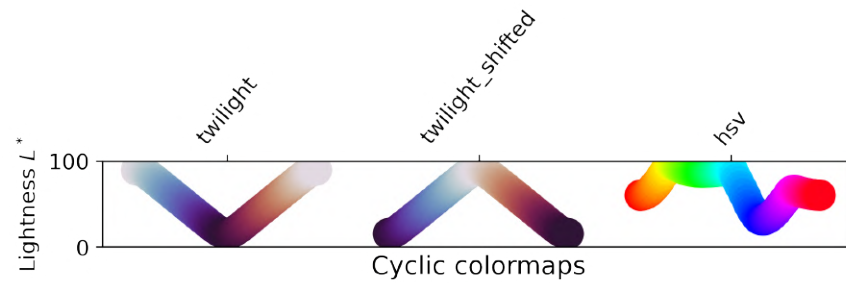
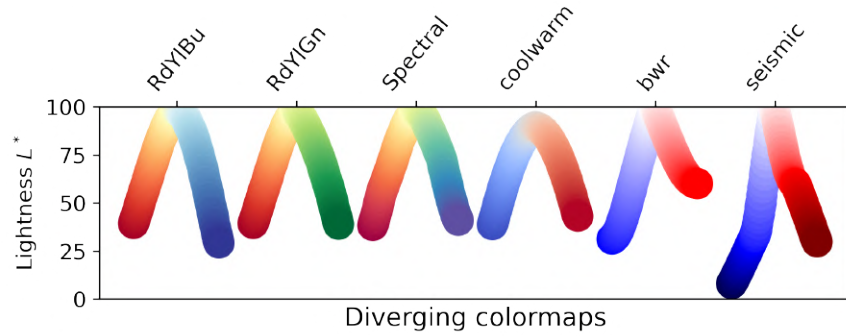
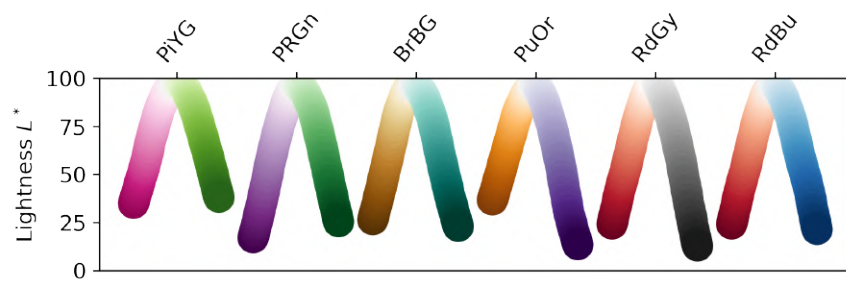
Miscellaneous colormaps

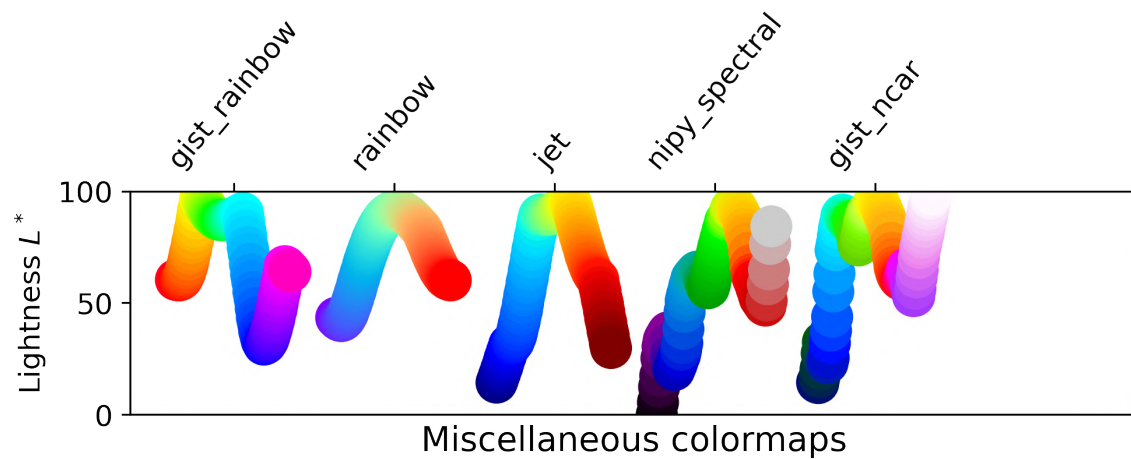
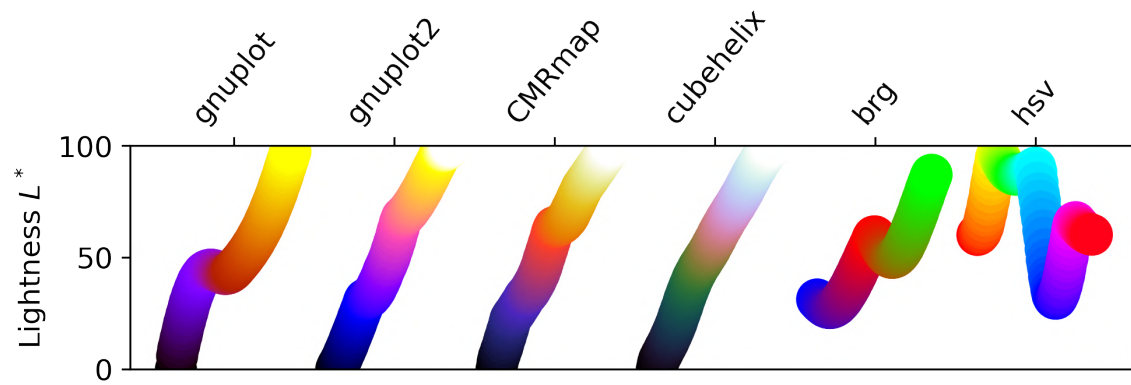
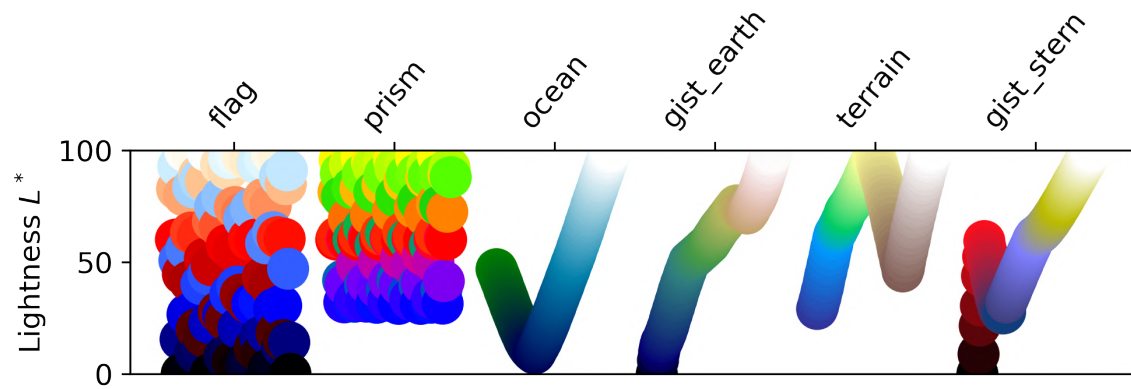


Note : All colormaps can be reversed by appending _r.

Lightness of Matplotlib Colormaps

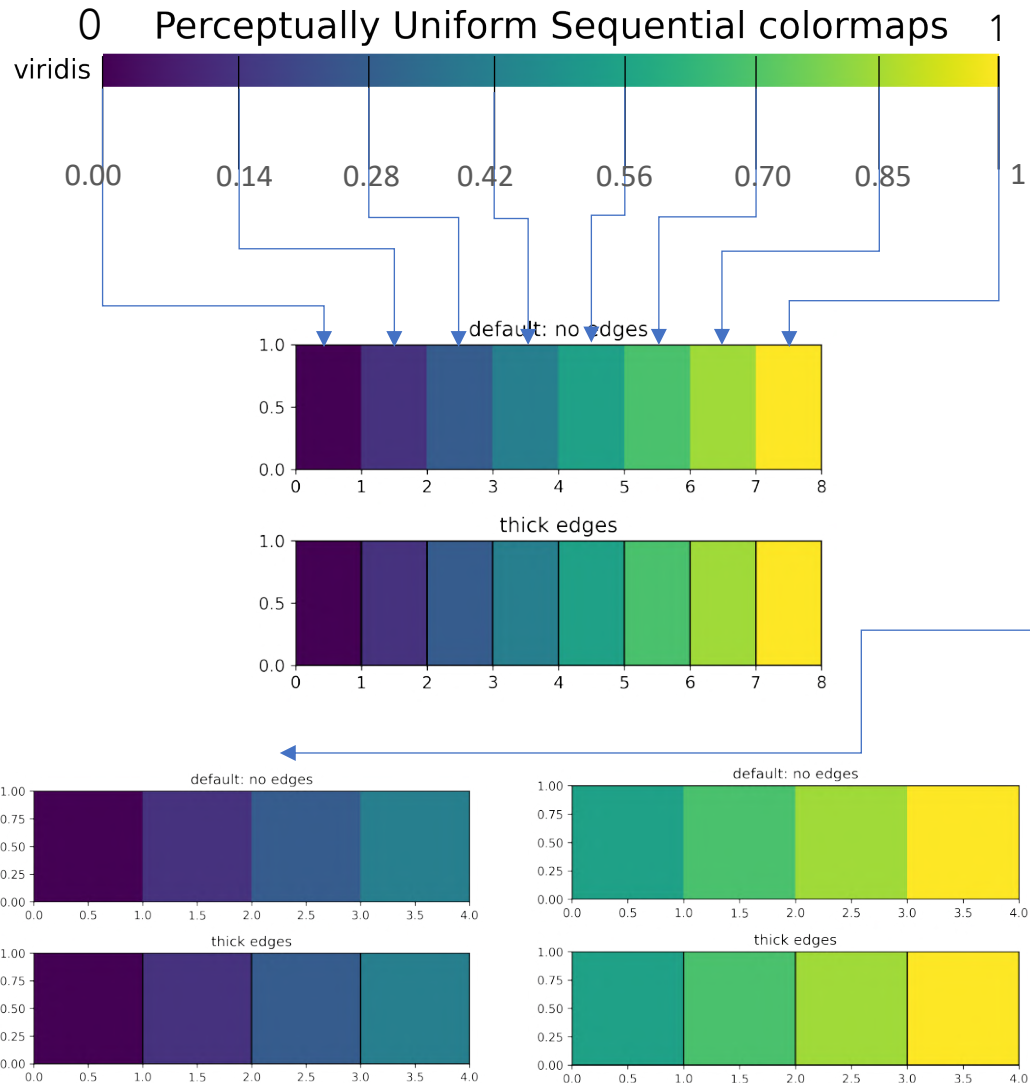






ACCESSING COLORS FROM A MATPLOTLIB COLORMAP

```
plt.cm.viridis(np.linspace(start = 0, stop = 1, num = 8))
```



COLORMAP is a 1D Array of colours.

`matplotlib.colors` module provides functions and classes for color specification conversions, and **mapping data/numbers onto colours** on a colormap.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm #matplotlib.colors module
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
```

#Get 8 colours from viridis colormap

```
viridis = cm.get_cmap('viridis', 8)
```

OR

```
plt.cm.viridis(np.linspace(0, 1, 8))
```

Get first 4# colors (0th – 3rd index) from the colormaps

Using range function to get colour from particular indices

```
viridis(range(4))
```

OR

```
viridis(np.linspace(0, 0.5, 4))
```

Get 4# colors (4th-7th index) from the colormaps

```
viridis(range(4,8))
```

OR

```
viridis(np.linspace(0.5, 1, 4))
```

*Colormap instances are used to convert data values (floats) from the interval [0, 1] to the RGBA color that the respective Colormap represents. For scaling of data into the [0, 1] interval see [matplotlib.colors.Normalize](#).

CREATING LISTED COLORMAP FROM EXISTING MATPLOTLIB COLORMAPS

Linear Segmented ColorMaps	Listed ColorMaps
- Sequential - Diverging - Cyclic	- Perceptually uniform sequential - Qualitative



Colormap object generated from a list of colors.

- most useful when indexing directly into a colormap
- used to generate special colormaps for ordinary mapping.

USE CASE SCENARIO	PARAMETER/PROPERTY
Setting cmap in scatter charts	cmap kwd argument
Initialize the color in bar plots, pie charts with colors of listed colormap . Check out tab10, tab20. Also, External libraries like [palettable] and [colorcet].	facecolor property can take a single color or list of colors #colors = plt.cm.tab10.colors[:8] #colors = plt.cm.Paired.colors
Setting the property cycle of the axes	ax.set_prop_cycle(color=plt.cm.tab10.colors)

Necessary imports – Especially ListedColormap Class

```
[1] : import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
from matplotlib.colors import LinearSegmentedColormap
```

Note : cm.get_cmap() returns a colormap

```
[2] : type(cm.get_cmap('YlGn', 256))
[2] : matplotlib.colors.LinearSegmentedColormap
```

```
[3] : type(cm.get_cmap('viridis', 20))
[3] : matplotlib.colors.ListedColormap
```

```
[4] : type(plt.cm.viridis(np.linspace(0, 1, 8))) #returns numpy array
[4] : numpy.ndarray
```

Creating a Listed Colormap

```
[5] : YlGn_New = cm.get_cmap('YlGn', 256)
```

```
[6] : ListedColormap(YlGn_New(np.linspace(0,1,10)))
[6] : <matplotlib.colors.ListedColormap at 0x7ffdd4625700>
```

```
[7] : viridis_short = cm.get_cmap('viridis',10)
ListedColormap(viridis_short(np.linspace(0,1,10)))
[8] : <matplotlib.colors.ListedColormap at 0x7ffdd4626700>
```


CODE CHUNK TO CREATE LISTED COLORMAPS AND ACCESSING COLORS

```
from matplotlib.colors import ListedColormap
colarray = plt.cm.plasma(range(256))
newcmap = ListedColormap(colarray)
```

#Colormap can be called with an integer array, or with a float array between 0 and 1.

`newcmap.colors` #colors attribute of a listed colormap
Above returns colors in RGBA tuples of float [0,1]. Shape (NX4 Array)

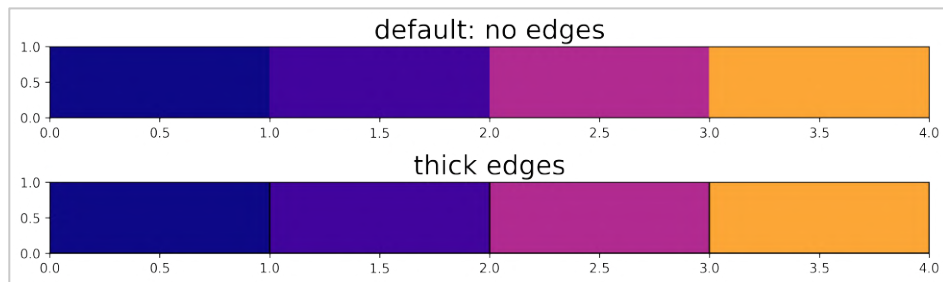
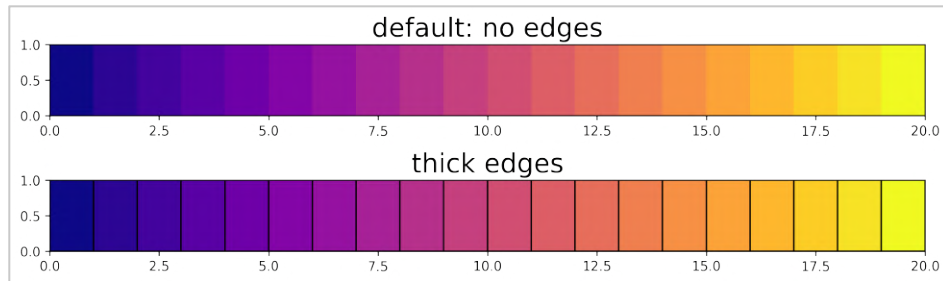
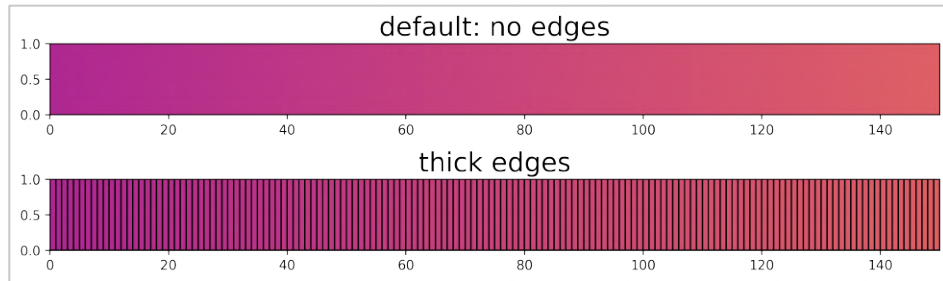
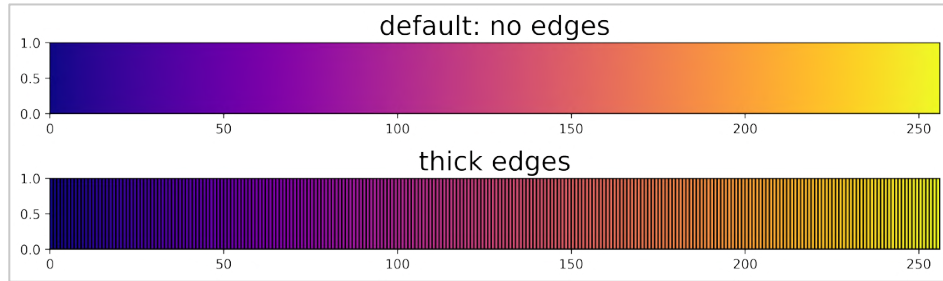
`newcmap(range(100,150))` #Returns colors from index 100 to 150

`newcmap(np.linspace(0,1,20))` #Returns 20 colors corresponding to equally spaced positions in between the specified start and end

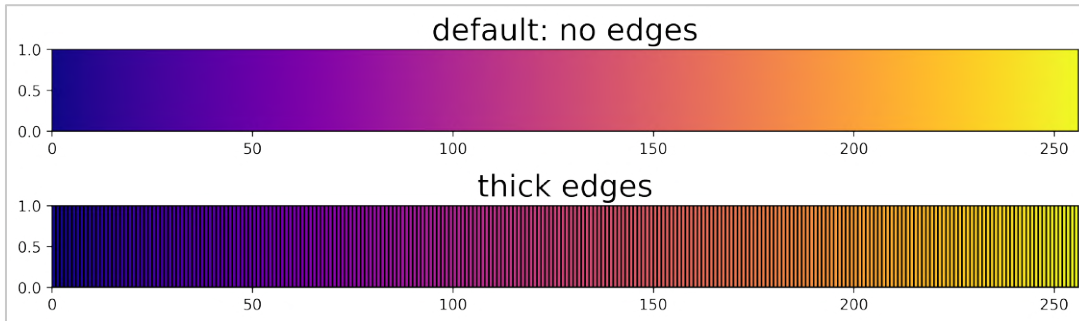
`newcmap([0.0, 0.1, 0.4, 0.8])` #Returns colors from specified locations considering the first color in newcmap is considered 0 and last colour is 1.

`#colors = [ plt.cm.tab20c(x) for x in np.linspace(0, 1, len(dic.values())) ]`

`cmap = ListedColormap(["darkorange", "gold", "lawngreen", "lightseagreen"])`



CODE SCRIPT TO CREATE PSEUDO COLOR PLOT



READY TO USE CODE TO ACCESS COLORS FROM CMAPS

```
#colors = plt.cm.Dark2(range(15))
#colors = plt.cm.tab20(range(15))
#colors = plt.cm.nipy_spectral(np.linspace(0,0.9,len(dic.values())))
#colors = plt.cm.CMRmap(np.linspace(0.2,1,len(dic.values())))

#colors = [ plt.cm.viridis(x) for x in np.linspace(0, 1, len(dic.values()))]
#colors = [ plt.cm.Set3(x) for x in np.linspace(0, 1, len(dic.values()))]
```

.colors attribute for listed colormaps

```
#colors = plt.cm.tab10.colors
#colors = plt.cm.Paired.colors
```

COLORS FROM EXTERNAL LIBRARY PALETTE

```
from palettable.cartocolors.qualitative import Bold_10
#colors = Bold_10.mpl_colors[:5]
colors = Bold_10.mpl_colors
```

```
# Doing the necessary imports
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
plt.rcParamsdefaults()
%matplotlib inline
#
colarray = plt.cm.plasma(range(256))
newcmp = ListedColormap(colarray)
```

#Creating the 1X256 Numpy Array (Matrix in Matlab) of float values from 0..1
`Z = np.linspace(0,1,256)[np.newaxis,:]`

```
fig, (ax0, ax1) = plt.subplots(2, 1, figsize = (10,3))
```

#Creating a pseudocolor plot without distinguishable edge between values
`c = ax0.pcolor(Z,cmap = newcmp)`
`ax0.set_title('default: no edges', fontsize = 20)`

#Creating a pseudocolor plot with distinguishable edge between the values
`c = ax1.pcolor(Z, cmap = newcmp, edgecolors='k', linewidths=1)`
`ax1.set_title('thick edges',fontsize = 20)`

```
fig.tight_layout()
fig.savefig('Plasma_256_colours.png',dpi=300, format='png',
           bbox_inches='tight')
```

- Refer to below Code from Matplotlib Gallery. Note slight tweaking was done to change the shape of the rectangular grid by using `np.newaxis`
- https://matplotlib.org/stable/gallery/images_contours_and_fields/pcolor_demo.html#sphx-glr-gallery-images-contours-and-fields-pcolor-demo-py

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

LEGEND

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

ASPECTS OF LEGEND

- The legend module defines the Legend class responsible for drawing legends associated with axes and/or figures.
- The Legend class is a container of **legend handles** and **legend texts**.

- legend entry
- legend handles
- legend labels
- legend keys

`h,l = ax.get_legend_handles_labels()`

1

CONTROLLING THE LEGEND ENTRIES

- Not all handles can be turned into legend entries automatically, so it is often necessary to create an artist which can.
- Legend handles don't have to exist on the Figure or Axes in order to be used.

- `m.patches`
- `m.lines`

2

CREATING PROXY ARTISTS SPECIFICALLY FOR THE LEGEND

- Next up is specifying the location of the legend
- Two parameters to control this :
 - `loc`
 - `bbox_to_anchor`

- `loc`
- `bbox_to_anchor`
- `bbox_transform`

3

CONTROLLING THE LEGEND LOCATION, PLACEMENT

- The legend handler map specifies how to create legend handles from artists (lines, patches, etc.) in the axes or figures.
- Custom legend handlers can be defined to support arbitrary objects not covered by default legend handlers.

4

DEFINING CUSTOM LEGEND HANDLERS

Matplotlib, Release 3.4.2, Page No. 2567, 2245, 1469

https://matplotlib.org/stable/api/as_gen/matplotlib.axes.Axes.legend.html?highlight=legend#matplotlib.axes.Axes.legend

https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html, [Legend Demo](#)



- 1
- Create a Figure instance
- 2
- Use Figure to create one or more Axes or Subplot
- 3
- Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4
- Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5
- Use Axes Methods (Accessors) to access artists.
- 6
- Set/Modify the Artist properties after accessing the artists
- 7
- Add Legend, Annotations if any and Set Title to the Axes

Adding Legend to Axes

Call signatures:

- `legend()`
- `legend(labels)`
- `legend(handles, labels)`

`handles` [list of [Artist](#)]

A list of Artists (lines, patches) to be added to the legend.

`labels` [list of str]

A list of labels to show next to the artists. The length of handles and labels should be the same. If they are not, they are truncated to the smaller of both lengths.

[Read more on legend entry, key, label and handle](#)

<code>ax.legend()</code>	Automatic detection of elements to be shown in the legend	• Suffices for most use-cases • Requires explicitly labeling the artist either at the time of creation or before calling legend • Artist whose label either starts with underscore or does not have label property set will be ignored from the Legend
<code>ax.legend(labels)</code>	Labeling existing plot elements	This method is discouraged owing to an implicit ordering of labels and artists.
<code>ax.legend(handles, labels)</code>	Explicitly defining the elements in the legend	• Gives Total Control over the artists displayed in the legend • This method is explicit and will accept the handles and labels at face value. • This implies the labels will override the label names assigned at the time of creation. • Also, one can pass additional artists or subset of artist to the handles list.

Matplotlib, Release 3.4.2, Page No. 2567, 2245, 1469

https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html?highlight=legend#matplotlib.axes.Axes.legend

https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html, [Legend Demo](#)



1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

Add Legend, Annotations if any and Set Title to the Axes

Important Terminologies

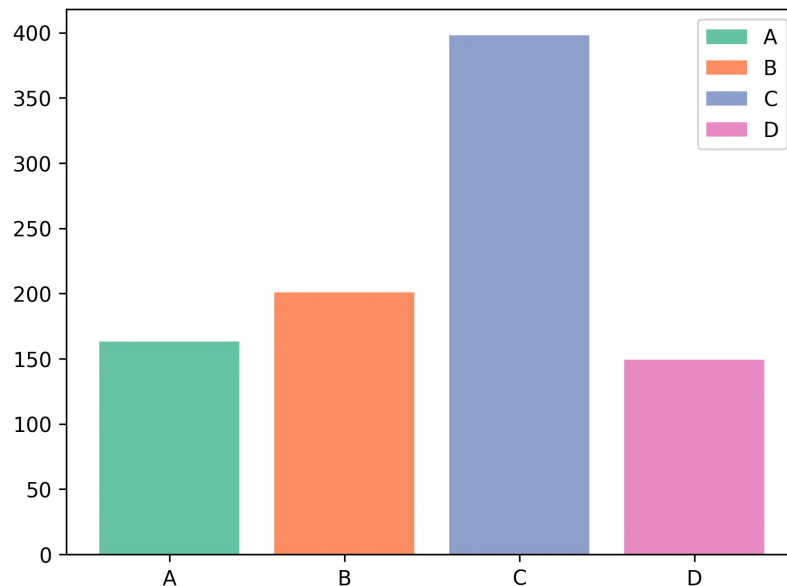
legend entry	A legend is made up of one or more legend entries. An entry is made up of exactly one key and one label.
legend key	The colored/patterned marker to the left of each legend label.
legend label	The text which describes the handle represented by the key.
legend handle	The original object which is used to generate an appropriate entry in the legend.

- 1 Create a Figure instance
- 2 Use Figure to create one or more Axes or Subplot
- 3 Use Axes Helper Methods to add artists to respective containers within the Axes object.
- 4 Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.
- 5 Use Axes Methods (Accessors) to access artists.
- 6 Set/Modify the Artist properties after accessing the artists
- 7 Add Legend, Annotations if any and Set Title to the Axes

Automatic detection of elements to be shown in the legend

Artists with valid labels are automatically detected.

To check the list of handles detected, use `ax.get_legend_handles_labels()`



```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.rcParams.defaults()
np.random.seed(450)
x = ['A', 'B', 'C', 'D']
y = np.random.randint(100, 400, 4)
```

```
fig, ax = plt.subplots()
p = ax.bar(x, y, color = plt.cm.Set2.colors)
```

```
#Setting labels to each of the bars
[bar.set(label = text) for text, bar in zip(x, p)]
```

```
print(ax.get_legend_handles_labels())
```

```
#Calling the legend function for automatic
detection of elements with labels to be shown
ax.legend()
```

Matplotlib, Release 3.4.2, Page No. 2567, 2245, 1469

https://matplotlib.org/stable/api/as_gen/matplotlib.axes.Axes.legend.html?highlight=legend#matplotlib.axes.Axes.legend

https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html, [Legend Demo](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

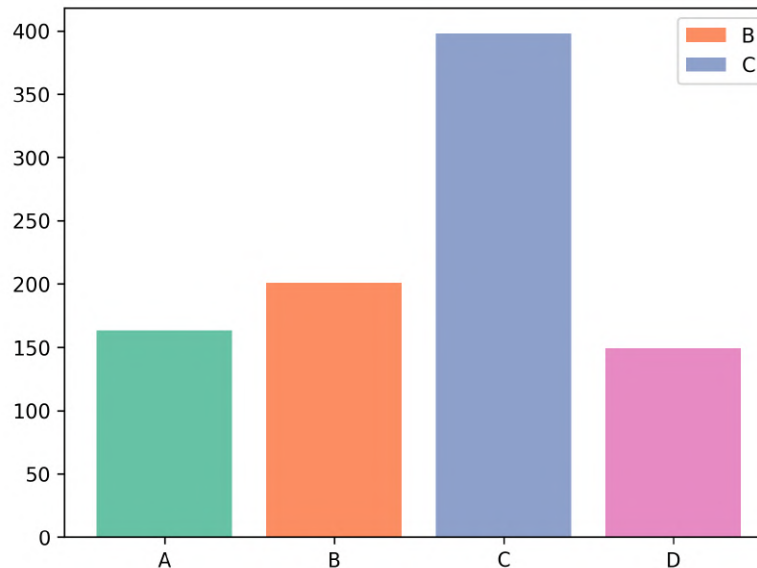
7

Add Legend, Annotations if any and Set Title to the Axes

Explicitly defining the elements in the legend

Labels Omitted

For full control of which artists have a legend entry, it is possible to pass an iterable of legend artists followed by an iterable of legend labels respectively:



Context :
EXCLUDE FROM LEGEND
ARTISTS WITH LABELS 'A' or 'D'

Note : Its okay to omit labels from the call signature if all the handles have valid labels.

Infact, either give the correct desired labels in this call corresponding to all the handles or omit labels altogether.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.rcParams()
np.random.seed(450)
x = ['A', 'B', 'C', 'D']
y = np.random.randint(100,400,4)
```

```
fig, ax = plt.subplots()
p = ax.bar(x,y, color = plt.cm.Set2.colors)
```

```
#Setting labels to each of the bars
[bar.set(label = text) for text,bar in zip(x,p)]
```

```
handles = []
for bar in p:
    if bar.get_label() not in ['A', 'D']:
        handles.append(bar)
```

```
print(handles) #Check the handles list
ax.legend(handles = handles)
```

```
fig.savefig('Explicitly_Passing_Iterable of \
handles.png',dpi=300, format='png',
bbox_inches='tight')
```


1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

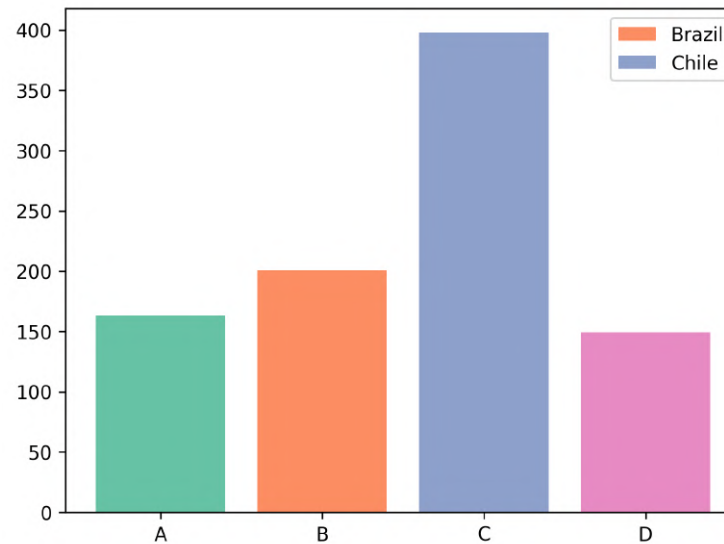
7

Add Legend, Annotations if any and Set Title to the Axes

Explicitly defining the elements in the legend

Labels Included

For full control of which artists have a legend entry, it is possible to pass an iterable of legend artists followed by an iterable of legend labels respectively:



Context :

- INCLUDE IN LEGEND, ARTISTS WITH LABELS 'B' or 'C'
- TWEAK 'B' AND 'C' LABELS IN LEGEND

Note : If labels is included in call signature, then labels in the labels list will override all pre-existing labels of the handles.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.rcParamsdefaults()
np.random.seed(450)
x = ['A', 'B', 'C', 'D']
y = np.random.randint(100,400,4)
```

```
fig, ax = plt.subplots()
p = ax.bar(x,y, color = plt.cm.Set2.colors)
```

```
#Setting labels to each of the bars
[bar.set(label = text) for text,bar in zip(x,p)]
```

```
handles = []
for bar in p:
    if bar.get_label() in ['B','C']: #filter 'B', 'C'
        handles.append(bar)
```

```
labels = ['Brazil', 'Chile'] #Creating labels
ax.legend(handles = handles, labels = labels)
```

```
fig.savefig('Explicitly_Passing_Iterable_ \
handles_labels.png',dpi=300, format='png',
bbox_inches='tight')
```

```
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
```

```
plt.rcParams()
fig, ax = plt.subplots(figsize = (7,4))
```

```
# where some data has already been plotted to ax
```

```
handles, labels = ax.get_legend_handles_labels()
```

```
# manually define a new patch
patch = mpatches.Patch(color='grey', label='Manual Label patch style')
```

```
# manually define a new line without any marker
line = mlines.Line2D([],[],color='blue', label='Manual Label line style')
# manually define a new line with marker '*'
marker1 = mlines.Line2D([],[],marker = '*', color='green',
                        label='Manual Label Marker style')
```

```
# manually define a new line with linestyle None
marker2 = mlines.Line2D([],[],ls = '',marker = '.', color='red',
                        label='Manual Label Marker without linestyle')
```

```
# handles is a list, so append manual patch
[handles.append(obj) for obj in [patch, line, marker1, marker2]]
```

```
# plot the legend
ax.legend(handles=handles,
          loc='upper right', bbox_to_anchor = (1,1), title = 'Legend')
leg = ax.get_legend() #fetches/gets the legend object
```

```
#Use print(dir(leg)) to get list of all attributes and method of legend class
leg._legend_title_box._text.set_color('Brown')
fig.savefig('Legend_Manual_positioning.png',dpi=150, format='png',
          bbox_inches='tight')
```

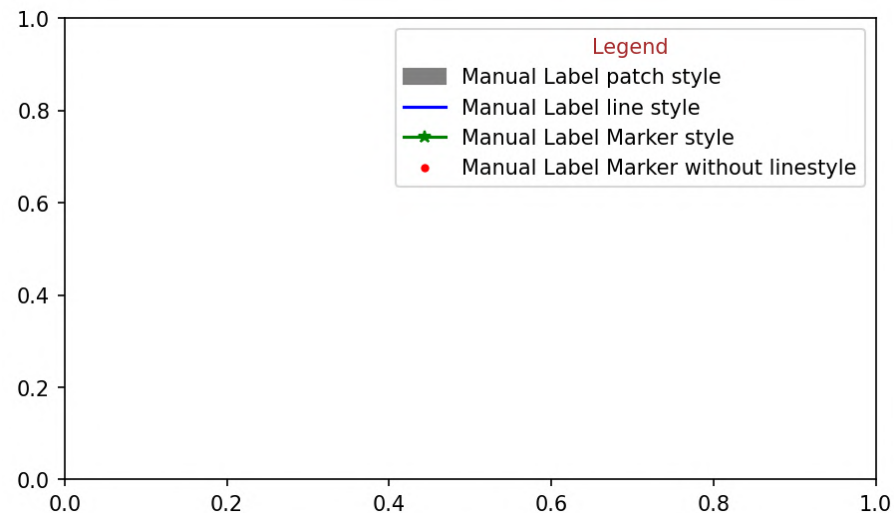
Creating Proxy Artists and Adding in the legend

Demo

1. Use `ax.get_legend_handles_labels()` to get existing handles and labels
2. Create instances of `patch` and `Line2D` without drawing them on the axes. Also, add label in initialization itself
3. Add the `patch` and `Line2D` to the list of handles from step 1
4. Call the `axes.legend()` method with below signature :

Steps

```
ax.legend(handles=handles, **kwargs)
ax.legend(handles=handles, labels = labels , **kwargs)
```



Matplotlib, Release 3.4.2, Page No. 2567, 2245, 1469

https://matplotlib.org/stable/api/as_gen/matplotlib.axes.Axes.legend.html?highlight=legend#matplotlib.axes.Axes.legend
https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html, [Legend Demo](#)

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

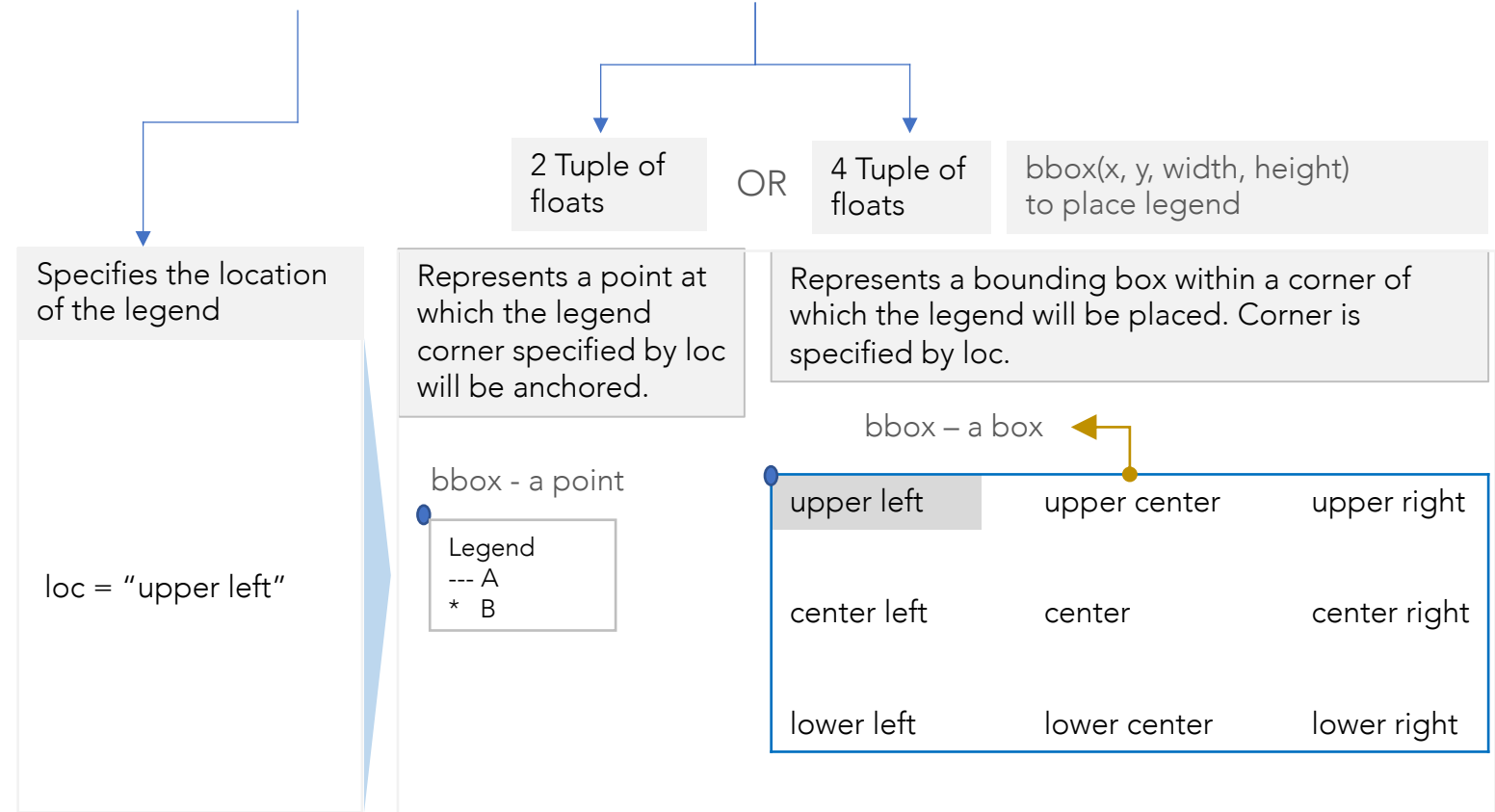
7

Add Legend, Annotations if any and Set Title to the Axes

Controlling Legend Location, Placement

Conceptual

- Use `loc` in conjunction with `bbox_to_anchor`



Note : Use transform object in `bbox_transform` argument to be explicit about the [coordinate system](#) for the `bbox_to_anchor` arguments

1

Create a Figure instance

2

Use Figure to create one or more Axes or Subplot

3

Use Axes Helper Methods to add artists to respective containers within the Axes object.

4

Use Axes/Axis Helper Methods to access and set x-axis and y-axis tick, tick labels and axis labels.

5

Use Axes Methods (Accessors) to access artists.

6

Set/Modify the Artist properties after accessing the artists

7

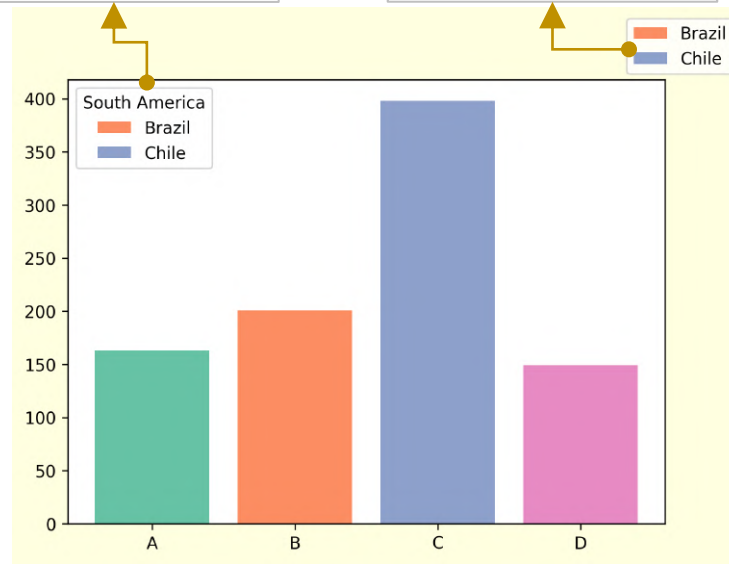
Add Legend, Annotations if any and Set Title to the Axes

Controlling Legend Location, Placement

Demo

Axes Legend
loc = 'upper left'
bbox_to_anchor = (0,1)

Figure Legend
loc = 'upper right'
bbox_to_anchor = (1,1)



```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.rcParams()
np.random.seed(450)
x = ['A', 'B', 'C', 'D']
y = np.random.randint(100,400,4)
```

```
fig, ax = plt.subplots()
p = ax.bar(x,y, color = plt.cm.Set2.colors)
```

```
#Setting labels to each of the bars
[bar.set(label = text) for text,bar in zip(x,p)]
```

```
#Creating handles list
```

```
handles = []
for bar in p:
    if bar.get_label() in ['B','C']:
        handles.append(bar)
```

```
#Creating labels list
```

```
labels = ['Brazil', 'Chile']
```

```
#Creating Axes Legend
```

```
ax.legend(handles = handles, labels = labels,
          loc = 'upper left',
          bbox_to_anchor = (0,1),
          bbox_transform = ax.transAxes)
```

```
leg = ax.get_legend()
leg.set_title('South America')
```

```
#Creating Figure Legend
```

```
fig.legend(handles = handles, labels = labels,
           loc = 'upper right',
           bbox_to_anchor = (1,1))
fig.patch.set(fc = 'lightyellow')
```

```
fig.savefig('Legend_location.png',dpi=300,
           format='png', bbox_inches='tight')
```

EPILOGUE

We are at the end of this review of matplotlib.

We have covered lots of ground. Indeed, the learning curve is steep, As there is a very large code base teeming with both stateless (pyplot) and stateful (OO style) interfaces.

However, certain fundamentals remain the same. That being, Every element in the plot is an artist that knows how to render itself on the figure canvas. Every Artist instance has properties, parameters and possibly attributes. Axes and Figure are the most important Artist/Containers in the matplotlib universe. It is through the the Axes instance, one can access almost all the artists in the figure. Once having accessed the specific artist instance, the properties can be modified using [artist.set method](#). At any moment one can chose to inspect the artist objects.

Deeper Understanding of Matplotlib

A deeper understanding of matplotlib can emerge by understanding “[Anatomy of a Figure](#)” and the [hierarchy of artists](#). This should be followed by sections - “[the Lifecycle of a Plot](#)” and “the Artist Tutorial” in Matplotlib Release, 3.4.2. Without understanding of these two foundational concepts, we would be limiting our capability and fail to leverage the full range of matplotlib functions and different class modules. Further, most efforts in debugging and coding would materialize into suboptimal coding practices.

In terms of the resources, the latest official documentation “[Matplotlib Release, 3.4.2](#)” pdf is a veritable treasure trove along with the examples gallery. Though, it may take some time to establish a pattern to search the required information. It also requires a threshold knowledge on the part of the user. One section to look out for is – “WHAT'S NEW IN MATPLOTLIB 3.4.0” for the latest developments and feature rollouts. Further, I also would suggest to check [Matplotlib Release, 2.0.2 version](#) especially for the matplotlib examples section. Apart from the matplotlib official site, [StackOverflow](#) is the go to site for all queries with an active community.



As we have limited our focus here on becoming comfortable on OO style, we have not covered numerous topics such as color bars, Tight Layout guide, Constraint layout guide, Different Patches and Advanced tutorials such as image tutorials, path tutorials. The official documentation has a comprehensive coverage on those.

Seaborn, Pandas and Matplotlib

If you are majorly a [Seaborn](#) User, this review will let you finetune the seaborn plots. As Seaborn is a high-level wrapper for Matplotlib, in essence, the OO interface will work there as well. While Seaborn plots take very less code and is intuitive from the moment go, this comes at the cost of lesser flexibility unless complemented by matplotlib. For instance, in a facetgrid object, modifying the row titles, column titles or the placement of figure legends, subplot legends becomes fairly easy with understanding of OO style. Same goes with Pandas plotting. Pandas has well developed charting capabilities. Pandas has powerful functionalities to deal with time series data. Using the three in conjunction particularly in multiple subplots will let you extract the best from each of the modules. Yes, this is the way.

Depending on specific use case, we shall go for the best combination. For sophisticated plots like heat maps or even faceted grids, Seaborn churns out beautiful charts with minimal code. For time series, Pandas has a definitive edge. Use matplotlib axes instance methods to customize them further. In that sense, the whole is indeed greater than the sum of parts.

From R, ggplot2 to Matplotlib and Python

For R Users, matplotlib can be analogous to R base graphics. And very likely, one used to the elegant graphics of ggplot2 can find the default matplotlib graphics code verbose and plot outputs very basic or “bare bones” similar to plots in R base graphics. But, it’s only a matter of time before one gets comfortable to the extent of customizing the plots even to the minutest level of setting orientation, thickness, number, padding of ticks. There is immense flexibility and clarity that is on offer within a clear OO framework of matplotlib. While I would rather not choose sides between ggplot2 and Python, instead am in favor of viewing them as tools for analysis to be used depending on particular use contexts. And while ggplot2 is easy to get started with but there is learning curve to gain control there in as well.

Having advocated the matplotlib skills for Python work environment, a suitable learning journey for R user can be becoming comfortable with Python basics, the basic data structures, built in functions, Numpy, Pandas followed by the pyplot interface and Seaborn. Pyplot interface, Seaborn and Pandas would be good entry points to be culminated with matplotlib.

A Note on Excel

And for Excel users, a reason for learning matplotlib and even ggplot2 can be the vast library of visualizations that is simply on offer. Excel spreadsheet shall always remain relevant and has its own place in analysis. But it is no match for advanced data analysis workflow with tidyverse packages in R; and numpy, Pandas in Python. For quick Exploratory data analysis, you have ggplot2 in R and matplotlib/Seaborn/pandas in Python. Also, there are some basic limitations in the charts you can plot in Excel. Trellis chart is not possible unless some tricks in terms of arranging the data in separate columns with blanks is employed. But this is only a trick/manoeuvre around an inherent limitation. To give another limitation, there is no fill and span feature. If daily sales data is plotted across months and I want to highlight particular days and weekends, it's a hard ask. This and lot of other limitations are easily handled in R and Python.

One definitive edge of R and Python over excel is the reproducibility of the plots. While excel is mainly drag and drop, point and click type GUI, so every time you are repeating an analysis, there are number of tasks that have to be redone. Data cleaning and aggregation followed by pivoting and then copying/pasting the relevant outputs. And then doing some EDA and then some graphs if found useful. This entire flow can of course be automated by VBA but that would boil down to individual expertise and additional time of code development and debugging. And if the data is huge, excel is no longer conducive even for pivoting. In comparison, R and Python being code based have a much simpler workflow, less error prone, shorter throughput and exact reproducibility. For a Excel user to get started on matplotlib, the learning journey would be same as suggested previously for R users. To get a headstart, check out below link for comparison of Pandas data structures and spreadsheet :

https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_spreadsheets.html



Back to this Review

Coming back to matplotlib, I would also emphasize being thorough with the basics of the color module, [transformation framework](#) and [legend guide](#). Also, few basics like modifying tick lines, tick labels, spines, gridlines, fig patch, axes patch can elevate the visual quality of plots. You also have [rcparams](#) and [style sheets](#) for global changes. Adding subplots using with and without gridspec also requires due consideration. For modifying font properties, using font dictionary is useful.

There are also args and kwargs that you will frequently encounter with every artist instance. It is convenient to create a dictionary containing keyword argument and value pairings in the code script. And later use dictionary style unpacking (**dict) to pass those arguments for setting the artist property. This helps in keeping code organized and avoid repetition of keyword arguments.

Certain properties are common to all artists. Check this useful [link](#). And there are certain artist specific properties. In this review, I have included the artist methods and instances with high likelihood of use. However, I would strongly recommend being familiar with the official documentation and website for the complete list and final verdict.

It is also inevitable that we will be dealing with date and time. There are multiple classes to handle the various concepts of date and time – either a single moment in time or a duration of time. To capture a moment in time, we have the datetime object in Native Python, datetime64 in Numpy, timestamp in Pandas, matplotlib dates. For parsing strings and ISO 8601 date time formats, we have dateutil parser; datetime strft, strfp; Pandas pd.to_pydatetime to name a few. TimeDelta objects are very handy and will be frequently used in conjunction with the datetime equivalent objects. Eventually, we shall develop liking to a method/module/class and stick with it.

Further, this review would be incomplete without coverage of [basic data structures](#) in Python, Numpy Array Attributes and Methods, Pandas Objects Attributes and Methods, [Pseudo Random number generation](#) in Python and Numpy, [Scipy for Statistics with python](#). I have included short sections on them to make this review self contained. However, I would recommend to go to official documentation of Pandas and Numpy for details.

The Curtain Falls ! But The End is only the beginning !!

On a final note, I have included an appendix containing multiple examples of plots and code implementations with different use cases. The intent here is to cover a gamut of plotting scenarios that can be always be referred to and specific code snippets be adapted for specific contexts. The references/source links for code implementations wherever applicable has been mentioned. There is also a references list at the very end presenting useful links related to matplotlib, python and data viz in general. Feel free to go through it.

To reaffirm, Consider this review as a compilation of distilled personal learnings and a handy reference.

With this, I wrap up this first edition and wish you the very best !

Bhaskar

Aug'2021

THANK YOU !

bhaskarjroy1605@gmail.com

www.linkedin.com/in/bhaskar-j-roy-1605

[Matplotlib Complete Review Link](#)

Other Interests : [Excel VBA](#) | [Documentary Film1](#) | [Documentary Film2](#)

DATA STRUCTURES OVERVIEW

Overview of Data Structures

- Python
- Numpy
- Pandas

Numpy

- Python-based library for mathematical computations and processing arrays.
- Homogenous Arrays of Fixed size.
- Vectorization and Broadcasting makes numpy fast.
- Check [Official Link](#), [Scipy Lectures](#)
- Check "Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib" by Robert Johansson

Python

For analyzing data and building models, arrays is a vital data structure.

The inbuilt data types and containers in Python cannot be restructured into more than one dimension, and also do not lend themselves to complex computations.

List

Tuples

Dictionaries

Sets

These Containers are also called iterables. They are unidimensional.
Check

- Learning Python by Mark Lutz
- Fluent Python by Luciano Ramalho

Pandas

- Provides efficient implementation of a DataFrame.
- DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data
- Powerful data wrangling operations similar to ones in database and spreadsheet framework

ndarray
(N dimensional Array)

Series

DataFrame

Index

Popular tool for data wrangling and analysis

- Grouping (Split-Apply-Combine)
- Handling missing data
- Input/output support
- Data Visualisation
- Integration with Numpy, Matplotlib, Scipy, and Scikit-learn
- Check
 - [Pandas online documentation](#)
 - [Stackoverflow](#)
 - [Python Data Science Handbook](#)

Pandas

- Pandas is implemented on top of NumPy arrays.
- A Pandas dataframe is used to store a structured dataset. It is more like a spreadsheet than an array..

NumPy provides:

- extension package to Python for multi-dimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)
- Also known as array oriented computing

Python objects:

- high-level number objects: integers, floating point
- containers: lists (costless insertion and append), dictionaries (fast lookup)

NumericIndex

CategoricalIndex

IntervalIndex

MultiIndex

DatetimeIndex

TimedeltaIndex

PeriodIndex

Indexing

Indexing is fundamental to Pandas and is what makes retrieval and access to data much faster compared to other tools. It is crucial to set an appropriate index to optimize performance
(Check A Python Data Analyst's Toolkit by Gayathri Rajagopalan)

Operators in Python

Arithmetic Operators

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Assignment Operators

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
//=	$x //= 3$	$x = x // 3$
**=	$x ** = 3$	$x = x ** 3$
&=	$x \& = 3$	$x = x \& 3$
=	$x = 3$	$x = x 3$
^=	$x ^ = 3$	$x = x ^ 3$
>>=	$x >> = 3$	$x = x >> 3$
<<=	$x << = 3$	$x = x << 3$

Comparison Operators

Operator	Name	Example
==	Equal	$x == y$
!=	Not equal	$x != y$
>	Greater than	$x > y$
<	Less than	$x < y$
>=	Greater than or equal to	$x >= y$
<=	Less than or equal to	$x <= y$

Logical Operators : used to combine conditional statements

Operator	Description	Example
and	Returns True if both statements are true	$x < 15$ and $x > 5$
or	Returns True if one of the statements is true	$x < 5$ or $y < 10$
not	Reverse the result, returns False if the result is true	not($x < 5$ or $y < 10$)

Membership Operators : used to test if a sequence is presented in an object

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Identity Operators : used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Difference between float division and floor division

[Link](#)

/	Division	x / y	$101/25 = 4.04$ (Python 3.X version)	Not consistent across different versions of Python (referred to as floating point division in number of versions)
%	Modulus	$x \% y$	$101/25 = 1$	Return the remainder
//	Floor division	$x // y$	$101//25 = 4$	Always Return floored quotient of x and y . (integer floor of division)

Also, check slide no.339 for ceiling division

Built-in Functions

Back

<u>abs()</u>	<u>delattr()</u>	<u>hash()</u>	<u>memoryview()</u>	<u>set()</u>
<u>all()</u>	<u>dict()</u>	<u>help()</u>	<u>min()</u>	<u>setattr()</u>
<u>any()</u>	<u>dir()</u>	<u>hex()</u>	<u>next()</u>	<u>slice()</u>
<u>ascii()</u>	<u>divmod()</u>	<u>id()</u>	<u>object()</u>	<u>sorted()</u>
<u>bin()</u>	<u>enumerate()</u>	<u>input()</u>	<u>oct()</u>	<u>staticmethod()</u>
<u>bool()</u>	<u>eval()</u>	<u>int()</u>	<u>open()</u>	<u>str()</u>
<u>breakpoint()</u>	<u>exec()</u>	<u>isinstance()</u>	<u>ord()</u>	<u>sum()</u>
<u>bytearray()</u>	<u>filter()</u>	<u>issubclass()</u>	<u>pow()</u>	<u>super()</u>
<u>bytes()</u>	<u>float()</u>	<u>iter()</u>	<u>print()</u>	<u>tuple()</u>
<u>callable()</u>	<u>format()</u>	<u>len()</u>	<u>property()</u>	<u>type()</u>
<u>chr()</u>	<u>frozenset()</u>	<u>list()</u>	<u>range()</u>	<u>vars()</u>
<u>classmethod()</u>	<u>getattr()</u>	<u>locals()</u>	<u>repr()</u>	<u>zip()</u>
<u>compile()</u>	<u>globals()</u>	<u>map()</u>	<u>reversed()</u>	<u>__import__()</u>
<u>complex()</u>	<u>hasattr()</u>	<u>max()</u>	<u>round()</u>	

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas General Functions

Data manipulations

melt(frame[, id_vars, value_vars, var_name, ...])	Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.
pivot(data[, index, columns, values])	Return reshaped DataFrame organized by given index / column values.
pivot_table(data[, values, index, columns, ...])	Create a spreadsheet-style pivot table as a DataFrame.
crosstab(index, columns[, values, rownames, ...])	Compute a simple cross tabulation of two (or more) factors.
cut(x, bins[, right, labels, retbins, ...])	Bin values into discrete intervals.
qcut(x, q[, labels, retbins, precision, ...])	Quantile-based discretization function.
merge(left, right[, how, on, left_on, ...])	Merge DataFrame or named Series objects with a database-style join.
merge_ordered(left, right[, on, left_on, ...])	Perform merge with optional filling/interpolation.
merge_asof(left, right[, on, left_on, ...])	Perform an asof merge.
concat(objs[, axis, join, ignore_index, ...])	Concatenate pandas objects along a particular axis with optional set logic along the other axes.
get_dummies(data[, prefix, prefix_sep, ...])	Convert categorical variable into dummy/indicator variables.

Top-level missing data

isna(obj)	Detect missing values for an array-like object.
isnull(obj)	Detect missing values for an array-like object.
notna(obj)	Detect non-missing values for an array-like object.
notnull(obj)	Detect non-missing values for an array-like object.

Top-level conversions

to_numeric(arg[, errors, downcast])	Convert argument to a numeric type.
---	-------------------------------------

Top-level dealing with datetimelike

to_datetime(arg[, errors, dayfirst, ...])	Convert argument to datetime.
to_timedelta(arg[, unit, errors])	Convert argument to timedelta.
date_range([start, end, periods, freq, tz, ...])	Return a fixed frequency DatetimeIndex.
bdate_range([start, end, periods, freq, tz, ...])	Return a fixed frequency DatetimeIndex, with business day as the default frequency.
period_range([start, end, periods, freq, name])	Return a fixed frequency PeriodIndex.
timedelta_range([start, end, periods, freq, ...])	Return a fixed frequency TimedeltaIndex, with day as the default frequency.
infer_freq(index[, warn])	Infer the most likely frequency given the input index.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion

Pandas Series

Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting

Index Objects

Numpy Arrays

Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas General Functions

Top-level dealing with intervals	
interval_range([start, end, periods, freq, ...])	Return a fixed frequency IntervalIndex.
Top-level evaluation	
eval(expr[, parser, engine, truediv, ...])	Evaluate a Python expression as a string using various backends.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/compare/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/compare/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas DataFrame

Attributes	
df.index	The index (row labels) of the DataFrame.
df.columns	The column labels of the DataFrame.
df.dtypes	Return the dtypes in the DataFrame.
df.info([verbose, buf, max_cols, ...])	Print a concise summary of a DataFrame.
df.select_dtypes([include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.
df.values	Return a Numpy representation of the DataFrame.
df.axes	Return a list representing the axes of the DataFrame.
df.ndim	Return an int representing the number of axes / array dimensions.
df.size	Return an int representing the number of elements in this object.
df.shape	Return a tuple representing the dimensionality of the DataFrame.
df.memory_usage([index, deep])	Return the memory usage of each column in bytes.
Indexing, iteration	
df.head([n])	Return the first n rows.
df.at	Access a single value for a row/column label pair.
df.iat	Access a single value for a row/column pair by integer position.
df.loc	Access a group of rows and columns by label(s) or a boolean array.
df.iloc	Purely integer-location based indexing for selection by position.
df.insert(loc, column, value[, ...])	Insert column into DataFrame at specified location.
df._iter_()	Iterate over info axis.
df.items()	Iterate over (column name, Series) pairs.
df.iteritems()	Iterate over (column name, Series) pairs.
df.keys()	Get the 'info axis' (see Indexing for more).
df.iterrows()	Iterate over DataFrame rows as (index, Series) pairs.
df.itertuples([index, name])	Iterate over DataFrame rows as namedtuples.
df.lookup(row_labels, col_labels)	(DEPRECATED) Label-based "fancy indexing" function for DataFrame.
df.pop(item)	Return item and drop from frame.
df.tail([n])	Return the last n rows.
df.xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.
df.get(key[, default])	Get item from object for given key (ex: DataFrame column).
df.isin(values)	Whether each element in the DataFrame is contained in values.
df.where(cond[, other, inplace, ...])	Replace values where the condition is False.
df.mask(cond[, other, inplace, axis, ...])	Replace values where the condition is True.
df.query(expr[, inplace])	Query the columns of a DataFrame with a boolean expression.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/Comparing/Joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/Comparing/Joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas DataFrame

Function application, GroupBy & window

df.apply(func[, axis, raw, ...])	Apply a function along an axis of the DataFrame.
df.applymap(func[, na_action])	Apply a function to a Dataframe elementwise.
df.pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).
df.agg([func, axis])	Aggregate using one or more operations over the specified axis.
df.aggregate([func, axis])	Aggregate using one or more operations over the specified axis.
df.transform(func[, axis])	Call func on self producing a DataFrame with transformed values.
df.groupby([by, axis, level, ...])	Group DataFrame using a mapper or by a Series of columns.
df.rolling(window[, min_periods, ...])	Provide rolling window calculations.
df.expanding([min_periods, center, ...])	Provide expanding transformations.
df.ewm([com, span, halflife, alpha, ...])	Provide exponential weighted (EW) functions.

Binary operator functions

df.add(other[, axis, level, fill_value])	Get Addition of dataframe and other, element-wise (binary operator add).
df.sub(other[, axis, level, fill_value])	Get Subtraction of dataframe and other, element-wise (binary operator sub).
df.mul(other[, axis, level, fill_value])	Get Multiplication of dataframe and other, element-wise (binary operator mul).
df.div(other[, axis, level, fill_value])	Get Floating division of dataframe and other, element-wise (binary operator truediv).
df.truediv(other[, axis, level, ...])	Get Floating division of dataframe and other, element-wise (binary operator truediv).
df.floordiv(other[, axis, level, ...])	Get Integer division of dataframe and other, element-wise (binary operator floordiv).
df.mod(other[, axis, level, fill_value])	Get Modulo of dataframe and other, element-wise (binary operator mod).
df.pow(other[, axis, level, fill_value])	Get Exponential power of dataframe and other, element-wise (binary operator pow).
df.dot(other)	Compute the matrix multiplication between the DataFrame and other.
df.radd(other[, axis, level, fill_value])	Get Addition of dataframe and other, element-wise (binary operator radd).
df.rsub(other[, axis, level, fill_value])	Get Subtraction of dataframe and other, element-wise (binary operator rsub).
df.rmul(other[, axis, level, fill_value])	Get Multiplication of dataframe and other, element-wise (binary operator rmul).
df.rdiv(other[, axis, level, fill_value])	Get Floating division of dataframe and other, element-wise (binary operator rtruediv).
df.rtruediv(other[, axis, level, ...])	Get Floating division of dataframe and other, element-wise (binary operator rtruediv).
df.rfloordiv(other[, axis, level, ...])	Get Integer division of dataframe and other, element-wise (binary operator rfloordiv).
df.rmod(other[, axis, level, fill_value])	Get Modulo of dataframe and other, element-wise (binary operator rmod).
df.rpow(other[, axis, level, fill_value])	Get Exponential power of dataframe and other, element-wise (binary operator rpow).
df.lt(other[, axis, level])	Get Less than of dataframe and other, element-wise (binary operator lt).
df.gt(other[, axis, level])	Get Greater than of dataframe and other, element-wise (binary operator gt).
df.le(other[, axis, level])	Get Less than or equal to of dataframe and other, element-wise (binary operator le).
df.ge(other[, axis, level])	Get Greater than or equal to of dataframe and other, element-wise (binary operator ge).
df.ne(other[, axis, level])	Get Not equal to of dataframe and other, element-wise (binary operator ne).
df.eq(other[, axis, level])	Get Equal to of dataframe and other, element-wise (binary operator eq).

Pandas DataFrame

[Attributes](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting, transposing](#)

[Combining/creating/joining/m](#)

[erging](#)

[Time Series related](#)

[Plotting](#)

[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)

[Conversion](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting](#)

[Combining/creating/joining/m](#)

[erging](#)

[Time Series related](#)

[Accessors](#)

[Datetime properties](#)

[Datetime methods](#)

[Period properties](#)

[Timedelta properties](#)

[Timedelta methods](#)

[String handling](#)

[Categorical Accessor](#)

[Plotting](#)

[Index Objects](#)

Numpy Arrays

[Attributes](#)

[Array Creation Routines](#)

[Shape Manipulation - shape,](#)

[resize, transpose](#)

[Calculation](#)

[Item Selection and Manipulation](#)

[Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas DataFrame

Computations / descriptive stats

df.abs()	Return a Series/DataFrame with absolute numeric value of each element.
df.all([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.
df.any([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
df.clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
df.corr([method, min_periods])	Compute pairwise correlation of columns, excluding NA/null values.
df.corrwith(other[, axis, drop, method])	Compute pairwise correlation.
df.count([axis, level, numeric_only])	Count non-NA cells for each column or row.
df.cov([min_periods, ddof])	Compute pairwise covariance of columns, excluding NA/null values.
df.cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
df.cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
df.cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
df.cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.
df.describe([percentiles, include, ...])	Generate descriptive statistics.
df.diff([periods, axis])	First discrete difference of element.
df.eval(expr[, inplace])	Evaluate a string describing operations on DataFrame columns.
df.kurt([axis, skipna, level, ...])	Return unbiased kurtosis over requested axis.
df.kurtosis([axis, skipna, level, ...])	Return unbiased kurtosis over requested axis.
df.mad([axis, skipna, level])	Return the mean absolute deviation of the values over the requested axis.
df.max([axis, skipna, level, ...])	Return the maximum of the values over the requested axis.
df.mean([axis, skipna, level, ...])	Return the mean of the values over the requested axis.
df.median([axis, skipna, level, ...])	Return the median of the values over the requested axis.
df.min([axis, skipna, level, ...])	Return the minimum of the values over the requested axis.
df.mode([axis, numeric_only, dropna])	Get the mode(s) of each element along the selected axis.
df.pct_change([periods, fill_method, ...])	Percentage change between the current and a prior element.
df.prod([axis, skipna, level, ...])	Return the product of the values over the requested axis.
df.product([axis, skipna, level, ...])	Return the product of the values over the requested axis.
df.quantile([q, axis, numeric_only, ...])	Return values at the given quantile over requested axis.
df.rank([axis, method, numeric_only, ...])	Compute numerical data ranks (1 through n) along axis.
df.round([decimals])	Round a DataFrame to a variable number of decimal places.
df.sem([axis, skipna, level, ddof, ...])	Return unbiased standard error of the mean over requested axis.
df.skew([axis, skipna, level, ...])	Return unbiased skew over requested axis.
df.sum([axis, skipna, level, ...])	Return the sum of the values over the requested axis.
df.std([axis, skipna, level, ddof, ...])	Return sample standard deviation over requested axis.
df.var([axis, skipna, level, ddof, ...])	Return unbiased variance over requested axis.
df.nunique([axis, dropna])	Count number of distinct elements in specified axis.
df.value_counts([subset, normalize, ...])	Return a Series containing counts of unique rows in the DataFrame.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/creating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/creating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas DataFrame

Reindexing / selection / label manipulation

df.add_prefix(prefix)	Prefix labels with string prefix.
df.add_suffix(suffix)	Suffix labels with string suffix.
df.align(other[, join, axis, level, ...])	Align two objects on their axes with the specified join method.
df.at time(time[, asof, axis])	Select values at particular time of day (e.g., 9:30AM).
df.between time(start time, end time)	Select values between particular times of the day (e.g., 9:00-9:30 AM).
df.drop([labels, axis, index, ...])	Drop specified labels from rows or columns.
df.drop_duplicates([subset, keep, ...])	Return DataFrame with duplicate rows removed.
df.duplicated([subset, keep])	Return boolean Series denoting duplicate rows.
df.equals(other)	Test whether two objects contain the same elements.
df.filter([items, like, regex, axis])	Subset the dataframe rows or columns according to the specified index labels.
df.first(offset)	Select initial periods of time series data based on a date offset.
df.head([n])	Return the first n rows.
df.idxmax([axis, skipna])	Return index of first occurrence of maximum over requested axis.
df.idxmin([axis, skipna])	Return index of first occurrence of minimum over requested axis.
df.last(offset)	Select final periods of time series data based on a date offset.
df.reindex([labels, index, columns, ...])	Conform Series/DataFrame to new index with optional filling logic.
df.reindex_like(other[, method, ...])	Return an object with matching indices as other object.
df.rename([mapper, index, columns, ...])	Alter axes labels.
df.rename_axis([mapper, index, ...])	Set the name of the axis for the index or columns.
df.reset_index([level, drop, ...])	Reset the index, or a level of it.
df.sample([n, frac, replace, ...])	Return a random sample of items from an axis of object.
df.set_axis(labels[, axis, inplace])	Assign desired index to given axis.
df.set_index(keys[, drop, append, ...])	Set the DataFrame index using existing columns.
df.tail([n])	Return the last n rows.
df.take(indices[, axis, is_copy])	Return the elements in the given positional indices along an axis.
df.truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas DataFrame

Missing data handling

df.backfill([axis, inplace, limit, ...])	Synonym for DataFrame.fillna() with method='bfill'.
df.bfill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='bfill'.
df.dropna([axis, how, thresh, ...])	Remove missing values.
df.ffill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='ffill'.
df.fillna([value, method, axis, ...])	Fill NA/NaN values using the specified method.
df.interpolate([method, axis, limit, ...])	Fill NaN values using an interpolation method.
df.isna()	Detect missing values.
df.isnull()	Detect missing values.
df.notna()	Detect existing (non-missing) values.
df.notnull()	Detect existing (non-missing) values.
df.pad([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='ffill'.
df.replace(to_replace, value, ...)	Replace values given in to_replace with value.

Reshaping, sorting, transposing

df.droplevel(level[, axis])	Return Series/DataFrame with requested index / column level(s) removed.
df.pivot([index, columns, values])	Return reshaped DataFrame organized by given index / column values.
df.pivot_table(values, index, ...)	Create a spreadsheet-style pivot table as a DataFrame.
df.reorder_levels(order[, axis])	Rearrange index levels using input order.
df.sort_values(by[, axis, ascending, ...])	Sort by the values along either axis.
df.sort_index([axis, level, ...])	Sort object by labels (along an axis).
df.nlargest(n, columns[, keep])	Return the first n rows ordered by columns in descending order.
df.nsmallest(n, columns[, keep])	Return the first n rows ordered by columns in ascending order.
df.swaplevel([i, j, axis])	Swap levels i and j in a MultiIndex.
df.stack([level, dropna])	Stack the prescribed level(s) from columns to index.
df.unstack([level, fill_value])	Pivot a level of the (necessarily hierarchical) index labels.
df.swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.
	Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.
df.melt([id_vars, value_vars, ...])	
df.explode(column[, ignore_index])	Transform each element of a list-like to a row, replicating index values.
df.squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
df.to_xarray()	Return an xarray object from the pandas object.
df.T	Transpose index and columns.
df.transpose(*args[, copy])	Transpose index and columns.

Pandas DataFrame

- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/creating/joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)

Pandas Series

- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/creating/joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)

Index Objects

Numpy Arrays

- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)



Data representation

- [Series](#): 1D
- [DataFrame](#): 2D
- Loading data** with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- [index](#): Series and DataFrame index
- [columns](#): DataFrame column
- [shape](#): Series and DataFrame dimensions
- [info\(\)](#): DataFrame informations
- [values](#): Index and Series values
- [unique\(\)](#): Series unique values
- [nunique\(\)](#): Series number of unique values

Selecting data

- [head\(\)](#) and [tail\(\)](#)
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- [sum\(\)](#)
- [str.len\(\)](#), [str.startswith\(\)](#) and [str.contains\(\)](#)
- [*apply\(\)*](#) and [lambda](#)
- [isin\(\)](#)
- [div\(\)](#)

Organizing data

- [transpose\(\)](#) or [T](#)
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas DataFrame

[Combining / comparing / joining / merging](#)

df.append(other[, ignore_index, ...])	Append rows of other to the end of caller, returning a new object.
df.assign(**kwargs)	Assign new columns to a DataFrame.
df.compare(other[, align_axis, ...])	Compare to another DataFrame and show the differences.
df.join(other[, on, how, suffix, ...])	Join columns of another DataFrame.
df.merge(right[, how, on, left_on, ...])	Merge DataFrame or named Series objects with a database-style join.
df.update(other[, join, overwrite, ...])	Modify in place using non-NA values from another DataFrame.

[Time Series-related](#)

df.asfreq(freq[, method, how, ...])	Convert time series to specified frequency.
df.asof(when[, subset])	Return the last row(s) without any NaNs before when.
df.shift([periods, freq, axis, ...])	Shift index by desired number of periods with an optional time freq.
df.slice_shift([periods, axis])	(DEPRECATED) Equivalent to shift without copying data.
df.tshift([periods, freq, axis])	(DEPRECATED) Shift the time index, using the index's frequency if available.
df.first_valid_index()	Return index for first non-NA value or None, if no NA value is found.
df.last_valid_index()	Return index for last non-NA value or None, if no NA value is found.
df.resample(rule[, axis, closed, ...])	Resample time-series data.
df.to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex.
df.to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at beginning of period.
df.tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.
df.tz_localize(tz[, axis, level, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.

[Plotting](#)

df.plot([x, y, kind, ax, ...])	DataFrame plotting accessor and method
df.plot.area([x, y])	Draw a stacked area plot.
df.plot.bar([x, y])	Vertical bar plot.
df.plot.barh([x, y])	Make a horizontal bar plot.
df.plot.box([by])	Make a box plot of the DataFrame columns.
df.plot.density([bw, method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
df.plot.hexbin(x, y[, C, ...])	Generate a hexagonal binning plot.
df.plot.hist([by, bins])	Draw one histogram of the DataFrame's columns.
df.plot.kde([bw, method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
df.plot.line([x, y])	Plot Series or DataFrame as lines.
df.plot.pie(**kwargs)	Generate a pie plot.
df.plot.scatter(x, y[, s, c])	Create a scatter plot with varying marker point size and color.
df.boxplot([column, by, ax, ...])	Make a box plot from DataFrame columns.
df.hist([column, by, grid, ...])	Make a histogram of the DataFrame's columns.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/Comparing/Joining/Merging
Time Series related
Plotting
Serialization/IO/Conversion

Pandas Series

Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/Comparing/Joining/Merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting

Index Objects

Numpy Arrays

Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)*
- [append\(\)](#) and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#) and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas DataFrame

Serialization / IO / conversion

DataFrame.from_dict(data[, orient, dtype, ...])	Construct DataFrame from dict of array-like or dicts.
DataFrame.from_records(data[, index, ...])	Convert structured or record ndarray to DataFrame.
df.to_parquet([path, engine, ...])	Write a DataFrame to the binary parquet format.
df.to_pickle(path[, compression, ...])	Pickle (serialize) object to file.
df.to_csv([path or buf, sep, na_rep, ...])	Write object to a comma-separated values (csv) file.
df.to_hdf(path or buf, key[, mode, ...])	Write the contained data to an HDF5 file using HDFStore.
df.to_sql(name, con[, schema, ...])	Write records stored in a DataFrame to a SQL database.
df.to_dict([orient, into])	Convert the DataFrame to a dictionary.
df.to_numpy([dtype, copy, na_value])	Convert the DataFrame to a NumPy array.
df.to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex.
df.to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at beginning of period.
df.to_xarray()	Return an xarray object from the pandas object.
df.to_xml([path or buffer, index, root_name, ...])	Render a DataFrame to an XML document.
df.to_excel(excel writer[, ...])	Write object to an Excel sheet.
df.to_json([path or buf, orient, ...])	Convert the object to a JSON string.
df.to_html([buf, columns, col_space, ...])	Render a DataFrame as an HTML table.
df.to_feather(path, **kwargs)	Write a DataFrame to the binary Feather format.
df.to_latex([buf, columns, ...])	Render object to a LaTeX tabular, longtable, or nested table/tabular.
df.to_stata(path[, convert dates, ...])	Export DataFrame object to Stata dta format.
df.to_gbq(destination table[, ...])	Write a DataFrame to a Google BigQuery table.
df.to_records([index, column dtypes, ...])	Convert DataFrame to a NumPy record array.
df.to_string([buf, columns, ...])	Render a DataFrame to a console-friendly tabular output.
df.to_clipboard([excel, sep])	Copy object to the system clipboard.
df.to_markdown([buf, mode, index, ...])	Print DataFrame in Markdown-friendly format.
df.style	Returns a Styler object.

Pandas DataFrame

[Attributes](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting, transposing](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Plotting](#)

[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)

[Conversion](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Accessors](#)

[Datetime properties](#)

[Datetime methods](#)

[Period properties](#)

[Timedelta properties](#)

[Timedelta methods](#)

[String handling](#)

[Categorical Accessor](#)

[Plotting](#)

[Index Objects](#)

Numpy Arrays

[Attributes](#)

[Array Creation Routines](#)

[Shape Manipulation - shape, resize, transpose](#)

[Calculation](#)

[Item Selection and Manipulation](#)

[Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Attributes

Series.index	The index (axis labels) of the Series.
Series.array	The ExtensionArray of the data backing this Series or Index.
Series.values	Return Series as ndarray or ndarray-like depending on the dtype.
Series.dtype	Return the dtype object of the underlying data.
Series.shape	Return a tuple of the shape of the underlying data.
Series.nbytes	Return the number of bytes in the underlying data.
Series.ndim	Number of dimensions of the underlying data, by definition 1.
Series.size	Return the number of elements in the underlying data.
Series.T	Return the transpose, which is by definition self.
Series.memory_usage(index, deep)	Return the memory usage of the Series.
Series.hasnans	Return if I have any nans; enables various perf speedups.
Series.empty	Indicator whether DataFrame is empty.
Series.dtypes	Return the dtype object of the underlying data.
Series.name	Return the name of the Series.
Series.flags	Get the properties associated with this pandas object.
Series.set_flags(*[, copy, ...])	Return a new object with updated flags.

Conversion

Series.astype(dtype[, copy, errors])	Cast a pandas object to a specified dtype dtype.
Series.convert_dtypes([infer_objects, ...])	Convert columns to best possible dtypes using dtypes supporting pd.NA.
Series.infer_objects()	Attempt to infer better dtypes for object columns.
Series.copy([deep])	Make a copy of this object's indices and data.
Series.bool()	Return the bool of a single element Series or DataFrame.
Series.to_numpy([dtype, copy, na_value])	A NumPy ndarray representing the values in this Series or Index.
Series.to_period([freq, copy])	Convert Series from DatetimeIndex to PeriodIndex.
Series.to_timestamp([freq, how, copy])	Cast to DatetimeIndex of Timestamps, at beginning of period.
Series.to_list()	Return a list of the values.
Series._array_ ([dtype])	Return the values as a NumPy array.

Pandas DataFrame

[Attributes](#)
[Indexing, Iteration](#)
[Function application, GroupBy & window](#)
[Binary Operator functions](#)
[Computation/descriptive stats](#)
[Reindexing/selection/label manipulation](#)
[Missing Data handling](#)
[Reshaping, sorting, transposing](#)
[Combining/compare/joining/merging](#)
[Time Series related](#)
[Plotting](#)
[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)
[Conversion](#)
[Indexing, Iteration](#)
[Function application, GroupBy & window](#)
[Binary Operator functions](#)
[Computation/descriptive stats](#)
[Reindexing/selection/label manipulation](#)
[Missing Data handling](#)
[Reshaping, sorting](#)
[Combining/compare/joining/merging](#)
[Time Series related](#)
[Accessors](#)
[Datetime properties](#)
[Datetime methods](#)
[Period properties](#)
[Timedelta properties](#)
[Timedelta methods](#)
[String handling](#)
[Categorical Accessor](#)
[Plotting](#)

Index Objects

Numpy Arrays

[Attributes](#)
[Array Creation Routines](#)
[Shape Manipulation - shape, resize, transpose](#)
[Calculation](#)
[Item Selection and Manipulation](#)
[Array Conversions](#)

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas Series

Indexing, iteration	
Series.get(key[, default])	Get item from object for given key (ex: DataFrame column).
Series.at	Access a single value for a row/column label pair.
Series.iat	Access a single value for a row/column pair by integer position.
Series.loc	Access a group of rows and columns by label(s) or a boolean array.
Series.iloc	Purely integer-location based indexing for selection by position.
Series.iter_()	Return an iterator of the values.
Series.items()	Lazily iterate over (index, value) tuples.
Series.iteritems()	Lazily iterate over (index, value) tuples.
Series.keys()	Return alias for index.
Series.pop(item)	Return item and drops from series.
Series.item()	Return the first element of the underlying data as a Python scalar.
Series.xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.
Function application, GroupBy & window	
Series.apply(func[, convert_dtype, args])	Invoke function on values of Series.
Series.agg([func, axis])	Aggregate using one or more operations over the specified axis.
Series.aggregate([func, axis])	Aggregate using one or more operations over the specified axis.
Series.transform(func[, axis])	Call func on self producing a Series with transformed values.
Series.map(arg[, na_action])	Map values of Series according to input correspondence.
Series.groupby([by, axis, level, as_index, ...])	Group Series using a mapper or by a Series of columns.
Series.rolling(window[, min_periods, ...])	Provide rolling window calculations.
Series.expanding([min_periods, center, ...])	Provide expanding transformations.
Series.ewm([com, span, halflife, alpha, ...])	Provide exponential weighted (EW) functions.
Series.pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Binary operator functions	
Series.add	Return Addition of series and other, element-wise (binary operator add).
Series.sub	Return Subtraction of series and other, element-wise (binary operator sub).
Series.mul	Return Multiplication of series and other, element-wise (binary operator mul).
Series.div	Return Floating division of series and other, element-wise (binary operator truediv).
Series.truediv	Return Floating division of series and other, element-wise (binary operator truediv).
Series.floordiv	Return Integer division of series and other, element-wise (binary operator floordiv).
Series.mod	Return Modulo of series and other, element-wise (binary operator mod).
Series.pow	Return Exponential power of series and other, element-wise (binary operator pow).

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas Series

Computations / descriptive stats	
Series.abs()	Return a Series/DataFrame with absolute numeric value of each element.
Series.all	Return whether all elements are True, potentially over an axis.
Series.any	Return whether any element is True, potentially over an axis.
Series.autocorr([lag])	Compute the lag-N autocorrelation.
Series.between(left, right[, inclusive])	Return boolean Series equivalent to left <= series <= right.
Series.clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
Series.corr(other[, method, min_periods])	Compute correlation with other Series, excluding missing values.
Series.count([level])	Return number of non-NA/null observations in the Series.
Series.cov(other[, min_periods, ddof])	Compute covariance with Series, excluding missing values.
Series.cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
Series.cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
Series.cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
Series.cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.
Series.describe([percentiles, include, ...])	Generate descriptive statistics.
Series.diff([periods])	First discrete difference of element.
Series.factorize([sort, na_sentinel])	Encode the object as an enumerated type or categorical variable.
Series.kurt	Return unbiased kurtosis over requested axis.
Series.mad([axis, skipna, level])	Return the mean absolute deviation of the values over the requested axis.
Series.max	Return the maximum of the values over the requested axis.
Series.mean	Return the mean of the values over the requested axis.
Series.median([axis, skipna, level, ...])	Return the median of the values over the requested axis.
Series.min	Return the minimum of the values over the requested axis.
Series.mode([dropna])	Return the mode(s) of the Series.
Series.nlargest([n, keep])	Return the largest n elements.
Series.nsmallest([n, keep])	Return the smallest n elements.
Series.pct_change	Percentage change between the current and a prior element.
Series.prod([axis, skipna, level, ...])	Return the product of the values over the requested axis.
Series.quantile([q, interpolation])	Return value at the given quantile.
Series.rank	Compute numerical data ranks (1 through n) along axis.
Series.sem([axis, skipna, level, ddof, ...])	Return unbiased standard error of the mean over requested axis.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas Series

Reindexing / selection / label manipulation

Series.align	Align two objects on their axes with the specified join method.
Series.drop	Return Series with specified index labels removed.
Series.droplevel(level[, axis])	Return Series/DataFrame with requested index / column level(s) removed.
Series.drop_duplicates	Return Series with duplicate values removed.
Series.duplicated([keep])	Indicate duplicate Series values.
Series.equals(other)	Test whether two objects contain the same elements.
Series.first(offset)	Select initial periods of time series data based on a date offset.
Series.head([n])	Return the first n rows.
Series.idxmax([axis, skipna])	Return the row label of the maximum value.
Series.idxmin([axis, skipna])	Return the row label of the minimum value.
Series.isin(values)	Whether elements in Series are contained in values.
Series.last(offset)	Select final periods of time series data based on a date offset.
Series.reindex([index])	Conform Series to new index with optional filling logic.
Series.reindex_like	Return an object with matching indices as other object.
Series.rename	Alter Series index labels or name.
Series.rename_axis	Set the name of the axis for the index or columns.
Series.reset_index	Generate a new DataFrame or Series with the index reset.
Series.sample	Return a random sample of items from an axis of object.
Series.set_axis(labels[, axis, inplace])	Assign desired index to given axis.
Series.take(indices[, axis, is_copy])	Return the elements in the given positional indices along an axis.
Series.tail([n])	Return the last n rows.
Series.truncate	Truncate a Series or DataFrame before and after some index value.
Series.where	Replace values where the condition is False.
Series.mask	Replace values where the condition is True.
Series.add_prefix(prefix)	Prefix labels with string prefix.
Series.add_suffix(suffix)	Suffix labels with string suffix.
Series.filter([items, like, regex, axis])	Subset the dataframe rows or columns according to the specified index labels.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Missing data handling	
Series.backfill	Synonym for DataFrame.fillna() with method='bfill'.
Series.bfill	Synonym for DataFrame.fillna() with method='bfill'.
Series.dropna([axis, inplace, how])	Return a new Series with missing values removed.
Series.ffill	Synonym for DataFrame.fillna() with method='ffill'.
Series.fillna([value, method, axis, ...])	Fill NA/NaN values using the specified method.
Series.interpolate	Fill NaN values using an interpolation method.
Series.isna()	Detect missing values.
Series.isnull()	Detect missing values.
Series.notna()	Detect existing (non-missing) values.
Series.notnull()	Detect existing (non-missing) values.
Series.pad	Synonym for DataFrame.fillna() with method='ffill'.
Series.replace	Replace values given in to_replace with value.
Reshaping, sorting	
Series.argsort([axis, kind, order])	Return the integer indices that would sort the Series values.
Series.argmin([axis, skipna])	Return int position of the smallest value in the Series.
Series.argmax([axis, skipna])	Return int position of the largest value in the Series.
Series.reorder_levels(order)	Rearrange index levels using input order.
Series.sort_values	Sort by the values.
Series.sort_index	Sort Series by index labels.
Series.swaplevel([i, j, copy])	Swap levels i and j in a MultiIndex.
Series.unstack([level, fill_value])	Unstack, also known as pivot, Series with MultiIndex to produce DataFrame.
Series.explode([ignore_index])	Transform each element of a list-like to a row.
Series.searchsorted	Find indices where elements should be inserted to maintain order.
Series.ravel([order])	Return the flattened underlying data as an ndarray.
Series.repeat(repeats[, axis])	Repeat elements of a Series.
Series.squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
Series.view([dtype])	Create a new view of the Series.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Combining / comparing / joining / merging	
Series.append(to_append[, ignore_index, ...])	Concatenate two or more Series.
Series.compare(other[, align_axis, ...])	Compare to another Series and show the differences.
Series.update(other)	Modify Series in place using values from passed Series.
Time Series-related	
Series.asfreq(freq[, method, how, ...])	Convert time series to specified frequency.
Series.asof(when[, subset])	Return the last row(s) without any NaNs before when.
Series.shift([periods, freq, axis, fill_value])	Shift index by desired number of periods with an optional time freq.
Series.first_valid_index()	Return index for first non-NA value or None, if no NA value is found.
Series.last_valid_index()	Return index for last non-NA value or None, if no NA value is found.
Series.resample(rule[, axis, closed, label, ...])	Resample time-series data.
Series.tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.
Series.tz_localize(tz[, axis, level, copy, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.
Series.at time(time[, asof, axis])	Select values at particular time of day (e.g., 9:30AM).
Series.between_time(start_time, end_time[,...])	Select values between particular times of the day (e.g., 9:00-9:30 AM).
Series.tshift([periods, freq, axis])	(DEPRECATED) Shift the time index, using the index's frequency if available.
Series.slice_shift([periods, axis])	(DEPRECATED) Equivalent to shift without copying data.

Accessors	
pandas provides dtype-specific methods under various accessors. These are separate namespaces within Series that only apply to specific data types.	
Data Type	Accessor
Datetime, Timedelta, Period	dt
String	str
Categorical	cat
Sparse	sparse

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/compaing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/compaing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas Series

Datetime properties

Series.dt.date	Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).
Series.dt.time	Returns numpy array of datetime.time.
Series.dt.timetz	Returns numpy array of datetime.time also containing timezone information.
Series.dt.year	The year of the datetime.
Series.dt.month	The month as January=1, December=12.
Series.dt.day	The day of the datetime.
Series.dt.hour	The hours of the datetime.
Series.dt.minute	The minutes of the datetime.
Series.dt.second	The seconds of the datetime.
Series.dt.microsecond	The microseconds of the datetime.
Series.dt.nanosecond	The nanoseconds of the datetime.
Series.dt.week	(DEPRECATED) The week ordinal of the year.
Series.dt.weekofyear	(DEPRECATED) The week ordinal of the year.
Series.dt.dayofweek	The day of the week with Monday=0, Sunday=6.
Series.dt.day_of_week	The day of the week with Monday=0, Sunday=6.
Series.dt.weekday	The day of the week with Monday=0, Sunday=6.
Series.dt.dayofyear	The ordinal day of the year.
Series.dt.day_of_year	The ordinal day of the year.
Series.dt.quarter	The quarter of the date.
Series.dt.is_month_start	Indicates whether the date is the first day of the month.
Series.dt.is_month_end	Indicates whether the date is the last day of the month.
Series.dt.is_quarter_start	Indicator for whether the date is the first day of a quarter.
Series.dt.is_quarter_end	Indicator for whether the date is the last day of a quarter.
Series.dt.is_year_start	Indicate whether the date is the first day of a year.
Series.dt.is_year_end	Indicate whether the date is the last day of the year.
Series.dt.is_leap_year	Boolean indicator if the date belongs to a leap year.
Series.dt.daysinmonth	The number of days in the month.
Series.dt.days_in_month	The number of days in the month.
Series.dt.tz	Return timezone, if any.
Series.dt.freq	Return the frequency object for this PeriodArray.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Datetime methods

Series.dt.to_period(*args, **kwargs)	Cast to PeriodArray/Index at a particular frequency.
Series.dt.to_pydatetime()	Return the data as an array of native Python datetime objects.
Series.dt.tz_localize(*args, **kwargs)	Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.
Series.dt.tz_convert(*args, **kwargs)	Convert tz-aware Datetime Array/Index from one time zone to another.
Series.dt.normalize(*args, **kwargs)	Convert times to midnight.
Series.dt.strftime(*args, **kwargs)	Convert to Index using specified date_format.
Series.dt.round(*args, **kwargs)	Perform round operation on the data to the specified freq.
Series.dt.floor(*args, **kwargs)	Perform floor operation on the data to the specified freq.
Series.dt.ceil(*args, **kwargs)	Perform ceil operation on the data to the specified freq.
Series.dt.month_name(*args, **kwargs)	Return the month names of the DateTimeIndex with specified locale.
Series.dt.day_name(*args, **kwargs)	Return the day names of the DateTimeIndex with specified locale.

Period properties

Series.dt.qyear
Series.dt.start_time
Series.dt.end_time

Timedelta properties

Series.dt.days	Number of days for each element.
Series.dt.seconds	Number of seconds (≥ 0 and less than 1 day) for each element.
Series.dt.microseconds	Number of microseconds (≥ 0 and less than 1 second) for each element.
Series.dt.nanoseconds	Number of nanoseconds (≥ 0 and less than 1 microsecond) for each element.
Series.dt.components	Return a Dataframe of the components of the Timedeltas.

Timedelta methods

Series.dt.to_timedelta()	Return an array of native datetime.timedelta objects.
Series.dt.total_seconds(*args, **kwargs)	Return total duration of each element expressed in seconds.

[Check strftime format codes](#) to pass arguments to `Series.dt.strftime(*args, **kwargs)`

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/compairing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion

Pandas Series

Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/compairing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting

Index Objects

Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)*
- [append\(\)](#) and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#) and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

String Handling

Series.str.capitalize()	Convert strings in the Series/Index to be capitalized.
Series.str.casefold()	Convert strings in the Series/Index to be casefolded.
Series.str.cat([others, sep, na_rep, join])	Concatenate strings in the Series/Index with given separator.
Series.str.center(width[, fillchar])	Pad left and right side of strings in the Series/Index.
Series.str.contains(pat[, case, flags, na, ...])	Test if pattern or regex is contained within a string of a Series or Index.
Series.str.count(pat[, flags])	Count occurrences of pattern in each string of the Series/Index.
Series.str.decode(encoding[, errors])	Decode character string in the Series/Index using indicated encoding.
Series.str.encode(encoding[, errors])	Encode character string in the Series/Index using indicated encoding.
Series.str.endswith(pat[, na])	Test if the end of each string element matches a pattern.
Series.str.extract(pat[, flags, expand])	Extract capture groups in the regex pat as columns in a DataFrame.
Series.str.extractall(pat[, flags])	Extract capture groups in the regex pat as columns in DataFrame.
Series.str.find(sub[, start, end])	Return lowest indexes in each strings in the Series/Index.
Series.str.findall(pat[, flags])	Find all occurrences of pattern or regular expression in the Series/Index.
Series.str.fullmatch(pat[, case, flags, na])	Determine if each string entirely matches a regular expression.
Series.str.get(i)	Extract element from each component at specified position.
Series.str.index(sub[, start, end])	Return lowest indexes in each string in Series/Index.
Series.str.join(sep)	Join lists contained as elements in the Series/Index with passed delimiter.
Series.str.len()	Compute the length of each element in the Series/Index.
Series.str.ljust(width[, fillchar])	Pad right side of strings in the Series/Index.
Series.str.lower()	Convert strings in the Series/Index to lowercase.
Series.str.lstrip([to_strip])	Remove leading characters.
Series.str.match(pat[, case, flags, na])	Determine if each string starts with a match of a regular expression.
Series.str.normalize(form)	Return the Unicode normal form for the strings in the Series/Index.
Series.str.pad(width[, side, fillchar])	Pad strings in the Series/Index up to width.
Series.str.partition([sep, expand])	Split the string at the first occurrence of sep.
Series.str.repeat(repeats)	Duplicate each string in the Series or Index.
Series.str.replace(pat, repl[, n, case, ...])	Replace each occurrence of pattern/regex in the Series/Index.
Series.str.rfind(sub[, start, end])	Return highest indexes in each strings in the Series/Index.
Series.str.rindex(sub[, start, end])	Return highest indexes in each string in Series/Index.
Series.str.rjust(width[, fillchar])	Pad left side of strings in the Series/Index.
Series.str.rpartition([sep, expand])	Split the string at the last occurrence of sep.
Series.str.rstrip([to_strip])	Remove trailing characters.

Pandas DataFrame

Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/creating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/creating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Pandas Series

String Handling

Series.str.slice([start, stop, step])	Slice substrings from each element in the Series or Index.
Series.str.slice_replace([start, stop, repl])	Replace a positional slice of a string with another value.
Series.str.split([pat, n, expand])	Split strings around given separator/delimiter.
Series.str.rsplit([pat, n, expand])	Split strings around given separator/delimiter.
Series.str.startswith(pat[, na])	Test if the start of each string element matches a pattern.
Series.str.strip([to_strip])	Remove leading and trailing characters.
Series.str.swapcase()	Convert strings in the Series/Index to be swapcased.
Series.str.title()	Convert strings in the Series/Index to titlecase.
Series.str.translate(table)	Map all characters in the string through the given mapping table.
Series.str.upper()	Convert strings in the Series/Index to uppercase.
Series.str.wrap(width, **kwargs)	Wrap strings in Series/Index at specified line width.
Series.str.zfill(width)	Pad strings in the Series/Index by prepending '0' characters.
Series.str.isalnum()	Check whether all characters in each string are alphanumeric.
Series.str.isalpha()	Check whether all characters in each string are alphabetic.
Series.str.isdigit()	Check whether all characters in each string are digits.
Series.str.isspace()	Check whether all characters in each string are whitespace.
Series.str.islower()	Check whether all characters in each string are lowercase.
Series.str.isupper()	Check whether all characters in each string are uppercase.
Series.str.istitle()	Check whether all characters in each string are titlecase.
Series.str.isnumeric()	Check whether all characters in each string are numeric.
Series.str.isdecimal()	Check whether all characters in each string are decimal.
Series.str.get_dummies([sep])	Return DataFrame of dummy/indicator variables for Series.
Series.str.slice([start, stop, step])	Slice substrings from each element in the Series or Index.
Series.str.slice_replace([start, stop, repl])	Replace a positional slice of a string with another value.
Series.str.split([pat, n, expand])	Split strings around given separator/delimiter.
Series.str.rsplit([pat, n, expand])	Split strings around given separator/delimiter.
Series.str.startswith(pat[, na])	Test if the start of each string element matches a pattern.
Series.str.strip([to_strip])	Remove leading and trailing characters.
Series.str.swapcase()	Convert strings in the Series/Index to be swapcased.
Series.str.title()	Convert strings in the Series/Index to titlecase.
Series.str.translate(table)	Map all characters in the string through the given mapping table.
Series.str.upper()	Convert strings in the Series/Index to uppercase.

Pandas DataFrame

- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/comparing/joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)

Pandas Series

- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/comparing/joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)

Index Objects

Numpy Arrays

- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Categorical accessor

Series.cat.categories	The categories of this categorical.
Series.cat.ordered	Whether the categories have an ordered relationship.
Series.cat.codes	Return Series of codes as well as the index.
Series.cat.rename_categories(*args, **kwargs)	Rename categories.
Series.cat.reorder_categories(*args, **kwargs)	Reorder categories as specified in new_categories.
Series.cat.add_categories(*args, **kwargs)	Add new categories.
Series.cat.remove_categories(*args, **kwargs)	Remove the specified categories.
Series.cat.remove_unused_categories(*args, ...)	Remove categories which are not used.
Series.cat.set_categories(*args, **kwargs)	Set the categories to the specified new_categories.
Series.cat.as_ordered(*args, **kwargs)	Set the Categorical to be ordered.
Series.cat.as_unordered(*args, **kwargs)	Set the Categorical to be unordered.

Plotting

Series.str.fullmatch(pat[, case, flags, na])	Determine if each string entirely matches a regular expression.
Series.plot([kind, ax, figsize, ...])	Series plotting accessor and method
Series.plot.area([x, y])	Draw a stacked area plot.
Series.plot.bar([x, y])	Vertical bar plot.
Series.plot.barh([x, y])	Make a horizontal bar plot.
Series.plot.box([by])	Make a box plot of the DataFrame columns.
Series.plot.density([bw method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
Series.plot.hist([by, bins])	Draw one histogram of the DataFrame's columns.
Series.plot.kde([bw method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
Series.plot.line([x, y])	Plot Series or DataFrame as lines.
Series.plot.pie(**kwargs)	Generate a pie plot.
Series.hist([by, ax, grid, xlabelsize, ...])	Draw histogram of the input series using matplotlib.

Pandas DataFrame

[Attributes](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting, transposing](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Plotting](#)

[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)

[Conversion](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Accessors](#)

[Datetime properties](#)

[Datetime methods](#)

[Period properties](#)

[Timedelta properties](#)

[Timedelta methods](#)

[String handling](#)

[Categorical Accessor](#)

[Plotting](#)

[Index Objects](#)

Numpy Arrays

[Attributes](#)

[Array Creation Routines](#)

[Shape Manipulation - shape, resize, transpose](#)

[Calculation](#)

[Item Selection and Manipulation](#)

[Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Pandas Series

Categorical accessor

Series.cat.categories	The categories of this categorical.
Series.cat.ordered	Whether the categories have an ordered relationship.
Series.cat.codes	Return Series of codes as well as the index.
Series.cat.rename_categories(*args, **kwargs)	Rename categories.
Series.cat.reorder_categories(*args, **kwargs)	Reorder categories as specified in new_categories.
Series.cat.add_categories(*args, **kwargs)	Add new categories.
Series.cat.remove_categories(*args, **kwargs)	Remove the specified categories.
Series.cat.remove_unused_categories(*args, ...)	Remove categories which are not used.
Series.cat.set_categories(*args, **kwargs)	Set the categories to the specified new_categories.
Series.cat.as_ordered(*args, **kwargs)	Set the Categorical to be ordered.
Series.cat.as_unordered(*args, **kwargs)	Set the Categorical to be unordered.

Plotting

Series.str.fullmatch(pat[, case, flags, na])	Determine if each string entirely matches a regular expression.
Series.plot([kind, ax, figsize, ...])	Series plotting accessor and method
Series.plot.area([x, y])	Draw a stacked area plot.
Series.plot.bar([x, y])	Vertical bar plot.
Series.plot.barh([x, y])	Make a horizontal bar plot.
Series.plot.box([by])	Make a box plot of the DataFrame columns.
Series.plot.density([bw method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
Series.plot.hist([by, bins])	Draw one histogram of the DataFrame's columns.
Series.plot.kde([bw method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
Series.plot.line([x, y])	Plot Series or DataFrame as lines.
Series.plot.pie(**kwargs)	Generate a pie plot.
Series.hist([by, ax, grid, xlabelsize, ...])	Draw histogram of the input series using matplotlib.

Pandas DataFrame

[Attributes](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting, transposing](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Plotting](#)

[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)

[Conversion](#)

[Indexing, Iteration](#)

[Function application, GroupBy & window](#)

[Binary Operator functions](#)

[Computation/descriptive stats](#)

[Reindexing/selection/label](#)

[manipulation](#)

[Missing Data handling](#)

[Reshaping, sorting](#)

[Combining/comparing/joining/merging](#)

[Time Series related](#)

[Accessors](#)

[Datetime properties](#)

[Datetime methods](#)

[Period properties](#)

[Timedelta properties](#)

[Timedelta methods](#)

[String handling](#)

[Categorical Accessor](#)

[Plotting](#)

[Index Objects](#)

Numpy Arrays

[Attributes](#)

[Array Creation Routines](#)

[Shape Manipulation - shape,](#)

[resize, transpose](#)

[Calculation](#)

[Item Selection and Manipulation](#)

[Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#) and [reset_index\(\)](#)
- [append\(\)](#) and [concat\(\)](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)
- [groupby\(\)](#)
- [pivot_table\(\)](#) and [crosstab\(\)](#)

Displaying data

- [plot\(\)](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Index Object

Properties	
Index.values	Return an array representing the data in the Index.
Index.is_monotonic	Alias for is_monotonic_increasing.
Index.is_monotonic_increasing	Return if the index is monotonic increasing (only equal or increasing) values.
Index.is_monotonic_decreasing	Return if the index is monotonic decreasing (only equal or decreasing) values.
Index.is_unique	Return if the index has unique values.
Index.has_duplicates	Check if the Index has duplicate values.
Index.hasnans	Return if I have any nans; enables various perf speedups.
Index.dtype	Return the dtype object of the underlying data.
Index.inferred_type	Return a string of the type inferred from the values.
Index.is_all_dates	Whether or not the index values only consist of dates.
Index.shape	Return a tuple of the shape of the underlying data.
Index.name	Return Index or MultiIndex name.
Index.names	
Index.nbytes	Return the number of bytes in the underlying data.
Index.ndim	Number of dimensions of the underlying data, by definition 1.
Index.size	Return the number of elements in the underlying data.
Index.empty	
Index.T	Return the transpose, which is by definition self.
Index.memory_usage(deep)	Memory usage of the values.
Index.values	Return an array representing the data in the Index.
Index.is_monotonic	Alias for is_monotonic_increasing.
Index.is_monotonic_increasing	Return if the index is monotonic increasing (only equal or increasing) values.
Index.is_monotonic_decreasing	Return if the index is monotonic decreasing (only equal or decreasing) values.
Index.is_unique	Return if the index has unique values.
Index.has_duplicates	Check if the Index has duplicate values.
Index.hasnans	Return if I have any nans; enables various perf speedups.
Index.dtype	Return the dtype object of the underlying data.

- [Index Objects](#)
- [Properties](#)
- [Modifying and computations](#)
- [Compatibility with MultiIndex](#)
- [Missing Values](#)
- [Conversion](#)
- [Sorting](#)
- [Timespecific Operations](#)
- [Combining/Joining /Set Operations](#)
- [Selecting](#)

- [Pandas DataFrame](#)
- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)
- [Pandas Series](#)
- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)
- [Numpy Arrays](#)
- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Index Object

Modifying and computations

Index.all() (*args, **kwargs)	Return whether all elements are Truthy.
Index.any() (*args, **kwargs)	Return whether any element is Truthy.
Index.argmin ([axis, skipna])	Return int position of the smallest value in the Series.
Index.argmax ([axis, skipna])	Return int position of the largest value in the Series.
Index.copy ([name, deep, dtype, names])	Make a copy of this object.
Index.delete (loc)	Make new Index with passed location(-s) deleted.
Index.drop (labels[, errors])	Make new Index with passed list of labels deleted.
Index.drop_duplicates ([keep])	Return Index with duplicate values removed.
Index.duplicated ([keep])	Indicate duplicate index values.
Index.equals (other)	Determine if two Index object are equal.
Index.factorize ([sort, na_sentinel])	Encode the object as an enumerated type or categorical variable.
Index.identical (other)	Similar to equals, but checks that object attributes and types are also equal.
Index.insert (loc, item)	Make new Index inserting new item at location.
Index.is (other)	More flexible, faster check like is but that works through views.
Index.is_boolean ()	Check if the Index only consists of booleans.
Index.is_categorical ()	Check if the Index holds categorical data.
Index.is_floating ()	Check if the Index is a floating type.
Index.is_integer ()	Check if the Index only consists of integers.
Index.is_interval ()	Check if the Index holds Interval objects.
Index.is_mixed ()	Check if the Index holds data with mixed data types.
Index.is_numeric ()	Check if the Index only consists of numeric data.
Index.is_object ()	Check if the Index is of the object dtype.
Index.min ([axis, skipna])	Return the minimum value of the Index.
Index.max ([axis, skipna])	Return the maximum value of the Index.
Index.reindex (target[, method, level, ...])	Create index with target’s values.
Index.rename (name[, inplace])	Alter Index or MultiIndex name.
Index.repeat (repeats[, axis])	Repeat elements of a Index.
Index.where (cond[, other])	Replace values where the condition is False.
Index.take (indices[, axis, allow_fill, ...])	Return a new Index of the values selected by the indices.
Index.putmask (mask, value)	Return a new Index of the values set with the mask.
Index.unique ([level])	Return unique values in the index.
Index.nunique ([dropna])	Return number of unique elements in the object.
Index.value_counts ([normalize, sort, ...])	Return a Series containing counts of unique values.
Index.where (cond[, other])	Replace values where the condition is False.
Index.take (indices[, axis, allow_fill, ...])	Return a new Index of the values selected by the indices.

Index Objects
Properties
Modifying and computations
Compatibility with MultiIndex
Missing Values
Conversion
Sorting
Timespecific Operations
Combining/Joining /Set Operations
Selecting

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/creating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/creating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Index Object

Compatibility with MultiIndex	
Index.set_names(names[, level, inplace])	Set Index or MultiIndex name.
Index.droplevel([level])	Return index with requested level(s) removed.
Missing values	
Index.fillna([value, downcast])	Fill NA/NaN values with the specified value.
Index.dropna([how])	Return Index without NA/NaN values.
Index.isna()	Detect missing values.
Index.notna()	Detect existing (non-missing) values.
Conversion	
Index.astype(dtype[, copy])	Create an Index with values cast to dtypes.
Index.item()	Return the first element of the underlying data as a Python scalar.
Index.map(mapper[, na_action])	Map values using input correspondence (a dict, Series, or function).
Index.ravel([order])	Return an ndarray of the flattened values of the underlying data.
Index.to_list()	Return a list of the values.
Index.to_native_types([slicer])	(DEPRECATED) Format specified values of self and return them.
Index.to_series([index, name])	Create a Series with both index and values equal to the index keys.
Index.to_frame([index, name])	Create a DataFrame with a column containing the Index.
Index.view([cls])	
Sorting	
Index.argsort(*args, **kwargs)	Return the integer indices that would sort the index.
Index.searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
Index.sort_values([return_indexer, ...])	Return a sorted copy of the index.
Time-specific operations	
Index.shift([periods, freq])	Shift index by desired number of time frequency increments.

- [Index Objects](#)
- [Properties](#)
- [Modifying and computations](#)
- [Compatibility with MultiIndex](#)
- [Missing Values](#)
- [Conversion](#)
- [Sorting](#)
- [Timespecific Operations](#)
- [Combining/Joining /Set Operations](#)
- [Selecting](#)

- [Pandas DataFrame](#)
- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)
- [Pandas Series](#)
- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)
- [Numpy Arrays](#)
- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

- [Time Series/Date Functionality](#)
- [Group by: split-apply-combine](#)
- [Windowing](#)

Index Object

Combining / joining / set operations

Index.append(other)	Append a collection of Index options together.
Index.join(other[, how, level, ...])	Compute join_index and indexers to conform data structures to the new index.
Index.intersection(other[, sort])	Form the intersection of two Index objects.
Index.union(other[, sort])	Form the union of two Index objects.
Index.difference(other[, sort])	Return a new Index with elements of index not in other.
Index.symmetric_difference(other[, ...])	Compute the symmetric difference of two Index objects.

Selecting

Index.asof(label)	Return the label from the index, or, if not present, the previous one.
Index.asof_locs(where, mask)	Return the locations (indices) of labels in the index.
Index.get_indexer	Compute indexer and mask for new index given the current index.
Index.get_indexer_for(target, **kwargs)	Guaranteed return of an indexer even when non-unique.
Index.get_indexer_non_unique(target)	Compute indexer and mask for new index given the current index.
Index.get_level_values(level)	Return an Index of values for requested level.
Index.get_loc(key[, method, tolerance])	Get integer location, slice or boolean mask for requested label.
Index.get_slice_bound(label, side[, kind])	Calculate slice bound that corresponds to given label.
Index.get_value(series, key)	Fast lookup of value from 1-dimensional ndarray.
Index.isin(values[, level])	Return a boolean array where the index values are in values.
Index.slice_indexer	Compute the slice indexer for input labels and step.
Index.slice_locs([start, end, step, kind])	Compute slice locations for input labels.

- [Index Objects](#)
- [Properties](#)
- [Modifying and computations](#)
- [Compatibility with MultiIndex](#)
- [Missing Values](#)
- [Conversion](#)
- [Sorting](#)
- [Timespecific Operations](#)
- [Combining/Joining /Set Operations](#)
- [Selecting](#)

- [Pandas DataFrame](#)
- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)
- [Pandas Series](#)
- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/Comparing/Joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)
- [Numpy Arrays](#)
- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)

Data representation

- [Series](#): 1D
- [DataFrame](#): 2D
- Loading data** with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- [index](#): Series and DataFrame index
- [columns](#): DataFrame column
- [shape](#): Series and DataFrame dimensions
- [info\(\)](#): DataFrame informations
- [values](#): Index and Series values
- [unique\(\)](#): Series unique values
- [nunique\(\)](#): Series number of unique values

Selecting data

- [head\(\)](#) and [tail\(\)](#)
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- [sum\(\)](#)
- [str.len\(\)](#), [str.startswith\(\)](#) and [str.contains\(\)](#)
- [*apply\(\)*](#) and [lambda](#)
- [isin\(\)](#)
- [div\(\)](#)

Organizing data

- [transpose\(\)](#) or [T](#)
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Numpy : Attributes of N-dimensional Array

Memory layout :

The following attributes contain information about the memory layout of the array:

ndarray.flags	Information about the memory layout of the array.
ndarray.shape	Tuple of array dimensions.
ndarray.strides	Tuple of bytes to step in each dimension when traversing an array.
ndarray.ndim	Number of array dimensions.
ndarray.data	Python buffer object pointing to the start of the array's data.
ndarray.size	Number of elements in the array.
ndarray.itemsize	Length of one array element in bytes.
ndarray.nbytes	Total bytes consumed by the elements of the array.
ndarray.base	Base object if memory is from some other object.

Other Attributes

ndarray.dtype	Data-type of the array's elements.
ndarray.T	The transposed array.
ndarray.real	The real part of the array.
ndarray.imag	The imaginary part of the array.
ndarray.flat	A 1-D iterator over the array.
ndarray.ctypes	An object to simplify the interaction of the array with the ctypes module.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [read_csv\(\)](#)*

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [set_index\(\)](#)* and [reset_index\(\)](#)*
- [append\(\)](#)* and [concat\(\)](#)*

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [apply\(\)](#)* and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [sort_values\(\)](#)*
- [groupby\(\)](#)*
- [pivot_table\(\)](#)* and [crosstab\(\)](#)*

Displaying data

- [plot\(\)](#)*

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Numpy Array Creation Routines

Numerical ranges

np.arange	Return evenly spaced values within a given interval.
np.linspace	Return evenly spaced numbers over a specified interval.
np.logspace	Return numbers spaced evenly on a log scale.
np.geomspace	Return numbers spaced evenly on a log scale (a geometric progression).
np.meshgrid	Return coordinate matrices from coordinate vectors.
np.mgrid	nd_grid instance which returns a dense multi-dimensional "meshgrid".
np.ogrid	nd_grid instance which returns an open multi-dimensional "meshgrid".

From shape or value

np.empty	Return a new array of given shape and type, without initializing entries.
np.empty_like	Return a new array with the same shape and type as a given array.
np.eye	Return a 2-D array with ones on the diagonal and zeros elsewhere.
np.identity	Return the identity array.
np.ones	Return a new array of given shape and type, filled with ones.
np.ones_like	Return an array of ones with the same shape and type as a given array.
np.zeros	Return a new array of given shape and type, filled with zeros.
np.zeros_like	Return an array of zeros with the same shape and type as a given array.
np.full	Return a new array of given shape and type, filled with fill_value.
np.full_like	Return a full array with the same shape and type as a given array.

Building matrices

diag	Extract a diagonal or construct a diagonal array.
diagflat	Create a two-dimensional array with the flattened input as a diagonal.
tri	An array with ones at and below the given diagonal and zeros elsewhere.
tril	Lower triangle of an array.
triu	Upper triangle of an array.
vander	Generate a Vandermonde matrix.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Numpy Array Creation Routines

From existing data

np.array	Create an array.
np.asarray	Convert the input to an array.
np.asanyarray	Convert the input to an ndarray, but pass ndarray subclasses through.
np.ascontiguousarray	Return a contiguous array (ndim >= 1) in memory (C order).
np.asmatrix	Interpret the input as a matrix.
np.copy	Return an array copy of the given object.
np.frombuffer	Interpret a buffer as a 1-dimensional array.
np.fromfile	Construct an array from data in a text or binary file.
np.fromfunction	Construct an array by executing a function over each coordinate.
np.fromiter	Create a new 1-dimensional array from an iterable object.
np.fromstring	A new 1-D array initialized from text data in a string.
np.loadtxt	Load data from a text file.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Numpy Array Methods

Shape manipulation – for shape, resize, transpose

ndarray.reshape	Returns an array containing the same data with a new shape.
ndarray.resize	Change shape and size of array in-place.
ndarray.transpose	Returns a view of the array with axes transposed.
ndarray.swapaxes	Return a view of the array with axis1 and axis2interchanged.
ndarray.flatten	Return a copy of the array collapsed into one dimension.
ndarray.ravel	Return a flattened array.
ndarray.squeeze	Remove axes of length one from a.
ndarray.reshape	Returns an array containing the same data with a new shape.
ndarray.resize	Change shape and size of array in-place.
ndarray.transpose	Returns a view of the array with axes transposed.

Calculation

ndarray.max	Return the maximum along a given axis.
ndarray.argmax	Return indices of the maximum values along the given axis.
ndarray.min	Return the minimum along a given axis.
ndarray.argmin	Return indices of the minimum values along the given axis.
ndarray.ptp	Peak to peak (maximum - minimum) value along a given axis.
ndarray.clip	Return an array whose values are limited to [min, max].
ndarray.conj	Complex-conjugate all elements.
ndarray.round	Return a with each element rounded to the given number of decimals.
ndarray.trace	Return the sum along diagonals of the array.
ndarray.sum	Return the sum of the array elements over the given axis.
ndarray.cumsum	Return the cumulative sum of the elements along the given axis.
ndarray.mean	Returns the average of the array elements along given axis.
ndarray.var	Returns the variance of the array elements, along given axis.
ndarray.std	Returns the standard deviation of the array elements along given axis.
ndarray.prod	Return the product of the array elements over the given axis
ndarray.cumprod	Return the cumulative product of the elements along the given axis.
ndarray.all	Returns True if all elements evaluate to True.
ndarray.any	Returns True if any of the elements of a evaluate to True.

Pandas DataFrame

- [Attributes](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting, transposing](#)
- [Combining/comparing/joining/merging](#)
- [Time Series related](#)
- [Plotting](#)
- [Serialization/IO/Conversion](#)

Pandas Series

- [Attributes](#)
- [Conversion](#)
- [Indexing, Iteration](#)
- [Function application, GroupBy & window](#)
- [Binary Operator functions](#)
- [Computation/descriptive stats](#)
- [Reindexing/selection/label manipulation](#)
- [Missing Data handling](#)
- [Reshaping, sorting](#)
- [Combining/comparing/joining/merging](#)
- [Time Series related](#)
- [Accessors](#)
- [Datetime properties](#)
- [Datetime methods](#)
- [Period properties](#)
- [Timedelta properties](#)
- [Timedelta methods](#)
- [String handling](#)
- [Categorical Accessor](#)
- [Plotting](#)

Index Objects

Numpy Arrays

- [Attributes](#)
- [Array Creation Routines](#)
- [Shape Manipulation - shape, resize, transpose](#)
- [Calculation](#)
- [Item Selection and Manipulation](#)
- [Array Conversions](#)



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and [lambda](#)

- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Numpy Array Methods

Item Selection and Manipulation

ndarray.take	Return an array formed from the elements of a at the given indices.
ndarray.put	Set <code>a.flat[n] = values[n]</code> for all n in indices.
ndarray.repeat	Repeat elements of an array.
ndarray.choose	Use an index array to construct a new array from a set of choices.
ndarray.sort	Sort an array in-place.
ndarray.argsort	Returns the indices that would sort this array.
ndarray.partition	Rearranges the elements in the array in such a way that the value of the element in kth position is in the position it would be in a sorted array.
ndarray.argpartition	Returns the indices that would partition this array.
ndarray.searchsorted	Find indices where elements of v should be inserted in a to maintain order.
ndarray.nonzero	Return the indices of the elements that are non-zero.
ndarray.compress	Return selected slices of this array along given axis.
ndarray.diagonal	Return specified diagonals.
ndarray.take	Return an array formed from the elements of a at the given indices.
ndarray.put	Set <code>a.flat[n] = values[n]</code> for all n in indices.
ndarray.repeat	Repeat elements of an array.
ndarray.choose	Use an index array to construct a new array from a set of choices.
ndarray.sort	Sort an array in-place.
ndarray.argsort	Returns the indices that would sort this array.
ndarray.partition	Rearranges the elements in the array in such a way that the value of the element in kth position is in the position it would be in a sorted array.
ndarray.argpartition	Returns the indices that would partition this array.
ndarray.searchsorted	Find indices where elements of v should be inserted in a to maintain order.
ndarray.nonzero	Return the indices of the elements that are non-zero.
ndarray.compress	Return selected slices of this array along given axis.
ndarray.diagonal	Return specified diagonals.

Array Conversions

ndarray.item	Copy an element of an array to a standard Python scalar and return it.
ndarray.tolist	Return the array as an a.ndim-levels deep nested list of Python scalars.
ndarray.itemset	Insert scalar into an array (scalar is cast to array's dtype, if possible)
ndarray.tostring	A compatibility alias for tobytes, with exactly the same behavior.
ndarray.tobytes	Construct Python bytes containing the raw data bytes in the array.
ndarray.tofile	Write array to a file as text or binary (default).
ndarray.astype	Copy of the array, cast to a specified type.
ndarray.fill	Fill the array with a scalar value.

Pandas DataFrame

[Attributes](#)
[Indexing, Iteration](#)
[Function application, GroupBy & window](#)
[Binary Operator functions](#)
[Computation/descriptive stats](#)
[Reindexing/selection/label manipulation](#)
[Missing Data handling](#)
[Reshaping, sorting, transposing](#)
[Combining/creating/joining/merging](#)
[Time Series related](#)
[Plotting](#)
[Serialization/IO/Conversion](#)

Pandas Series

[Attributes](#)
[Conversion](#)
[Indexing, Iteration](#)
[Function application, GroupBy & window](#)
[Binary Operator functions](#)
[Computation/descriptive stats](#)
[Reindexing/selection/label manipulation](#)
[Missing Data handling](#)
[Reshaping, sorting](#)
[Combining/creating/joining/merging](#)
[Time Series related](#)
[Accessors](#)
[Datetime properties](#)
[Datetime methods](#)
[Period properties](#)
[Timedelta properties](#)
[Timedelta methods](#)
[String handling](#)
[Categorical Accessor](#)
[Plotting](#)

Index Objects

Numpy Arrays

[Attributes](#)
[Array Creation Routines](#)
[Shape Manipulation - shape, resize, transpose](#)
[Calculation](#)
[Item Selection and Manipulation](#)
[Array Conversions](#)



Data representation

- [Series](#): 1D
- [DataFrame](#): 2D
- Loading data** with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- [index](#): Series and DataFrame index
- [columns](#): DataFrame column
- [shape](#): Series and DataFrame dimensions
- [info\(\)](#): DataFrame informations
- [values](#): Index and Series values
- [unique\(\)](#): Series unique values
- [nunique\(\)](#): Series number of unique values

Selecting data

- [head\(\)](#) and [tail\(\)](#)
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- [sum\(\)](#)
- [str.len\(\)](#), [str.startswith\(\)](#) and [str.contains\(\)](#)
- [*apply\(\)*](#) and [lambda](#)
- [isin\(\)](#)
- [div\(\)](#)

Organizing data

- [transpose\(\)](#) or [T](#)
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Useful Tips for Data wrangling with Pandas

General order of precedence for performance of various operations:

1. Vectorization
2. [Cython](#) routines
3. List Comprehensions (vanilla for loop)
4. [DataFrame.apply\(\)](#):
 - Reductions that can be performed in Cython,
 - Iteration in Python space
5. [DataFrame.itertuples\(\)](#) and [iteritems\(\)](#)
6. [DataFrame.iterrows\(\)](#)
iterrows and itertuples should be used in very rare circumstances, such as generating row objects/nametuples for sequential processing, which is really the only thing these functions are useful for.

Check

- [Essential basic functionality](#)
- <https://stackoverflow.com/questions/16476924/>

On Groupby

The groupby object has four methods that accept a function (or functions) to perform a calculation on each group. These four methods are [.agg](#), [.filter](#), [.transform](#), and [.apply](#). Each of the first three of these methods has a very specific output that the function must return. [.agg](#) must return a scalar value, [.filter](#) must return a Boolean, and [.transform](#) must return a Series or DataFrame with the same length as the passed group. The [.apply](#) method, however, may return a scalar value, a Series, or even a DataFrame of any shape, therefore making it very flexible. It is also called only once per group (on a DataFrame), while the [.transform](#) and [.agg](#) methods get called once for each aggregating column (on a Series). [The .apply method's ability to return a single object when operating on multiple columns at the same time is of tremendous utility.](#)

- Read Pandas 1.x Cookbook, Second Edition, Matt Harrison Theodore Petrou [Grouping for Aggregation, Filtration, and Transformation]

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- [Series](#): 1D
- [DataFrame](#): 2D
- Loading data** with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- [index](#): Series and DataFrame index
- [columns](#): DataFrame column
- [shape](#): Series and DataFrame dimensions
- [info\(\)](#): DataFrame informations
- [values](#): Index and Series values
- [unique\(\)](#): Series unique values
- [nunique\(\)](#): Series number of unique values

Selecting data

- [head\(\)](#) and [tail\(\)](#)
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- [sum\(\)](#)
- [str.len\(\)](#), [str.startswith\(\)](#) and [str.contains\(\)](#)
- [*apply\(\)*](#) and [lambda](#)
- [isin\(\)](#)
- [div\(\)](#)

Organizing data

- [transpose\(\)](#) or [T](#)
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Group by: split-apply-combine

By “group by” we are referring to a process involving one or more of the following steps:

- Splitting the data into groups based on some criteria.
- Applying a function to each group independently.
- Combining the results into a data structure.

Out of these, the split step is the most straightforward. In fact, in many situations we may wish to split the data set into groups and do something with those groups. In the apply step, we might wish to do one of the following:

1. [Aggregation](#): compute a summary statistic (or statistics) for each group. Some examples:
 - Compute group sums or means.
 - Compute group sizes / counts.
2. [Transformation](#): perform some group-specific computations and return a like-indexed object. Some examples:
 - Standardize data (zscore) within a group.
 - Filling NAs within groups with a value derived from each group.
3. [Filtration](#): discard some groups, according to a group-wise computation that evaluates True or False. Some examples:
 - Discard data that belongs to groups with only a few members.
 - Filter out data based on the group sum or mean.

Some combination of the above: GroupBy will examine the results of the apply step and try to return a sensibly combined result if it doesn't fit into either of the above two categories.

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/comparing/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/comparing/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Community tutorials (listed in Pandas official documentation)

This is a guide to many pandas tutorials by the community, geared mainly for new users.

pandas cookbook by Julia Evans

The goal of this 2015 cookbook (by [Julia Evans](#)) is to give you some concrete examples for getting started with pandas. These are examples with real-world data, and all the bugs and weirdness that entails. For the table of contents, see the [pandas-cookbook GitHub repository](#).

Learn pandas by Hernan Rojas

A set of lesson for new pandas users: <https://bitbucket.org/hrojas/learn-pandas>

Practical data analysis with Python

This [guide](#) is an introduction to the data analysis process using the Python data ecosystem and an interesting open dataset. There are four sections covering selected topics as [munging data](#), [aggregating data](#), [visualizing data](#) and [time series](#).

Exercises for new users

Practice your skills with real data sets and exercises. For more resources, please visit the main [repository](#).

Modern pandas

Tutorial series written in 2016 by [Tom Augspurger](#). The source may be found in the GitHub repository [TomAugspurger/effective-pandas](#).

- [Modern Pandas](#)
- [Method Chaining](#)
- [Indexes](#)
- [Performance](#)
- [Tidy Data](#)
- [Visualization](#)
- [Timeseries](#)

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions

Data representation

- Series: 1D
- DataFrame: 2D
- Loading data with [*read_csv\(\)*](#)

- header
- names
- sep
- encoding
- converters

Getting information

- index: Series and DataFrame index
- columns: DataFrame column
- shape: Series and DataFrame dimensions
- info(): DataFrame informations
- values: Index and Series values
- unique(): Series unique values
- nunique(): Series number of unique values

Selecting data

- head() and tail()
- Column accessing
- Row accessing
- Fancy indexing
- Logical masking

Indexing and merging data

- [*set_index\(\)*](#) and [*reset_index\(\)*](#)
- [*append\(\)*](#) and [*concat\(\)*](#)

Computing over data

- sum()
- str.len(), str.startswith() and str.contains()
- [*apply\(\)*](#) and lambda
- isin()
- div()

Organizing data

- transpose() or T
- [*sort_values\(\)*](#)
- [*groupby\(\)*](#)
- [*pivot_table\(\)*](#) and [*crosstab\(\)*](#)

Displaying data

- [*plot\(\)*](#)

Check Link – PyParis 2017

[Time Series/Date Functionality](#)
[Group by: split-apply-combine](#)
[Windowing](#)

Community tutorials (listed in Pandas official documentation)

Excel charts with pandas, vincent and xlswriter

- [Using Pandas and XlsxWriter to create Excel charts](#)

Video tutorials

- [Pandas From The Ground Up](#) (2015) (2:24) [GitHub repo](#)
- [Introduction Into Pandas](#) (2016) (1:28) [GitHub repo](#)
- [Pandas: .head\(\) to .tail\(\)](#) (2016) (1:26) [GitHub repo](#)
- [Data analysis in Python with pandas](#) (2016-2018) [GitHub repo](#) and [Jupyter Notebook](#)
- [Best practices with pandas](#) (2018) [GitHub repo](#) and [Jupyter Notebook](#)

Various tutorials

- [Wes McKinney’s \(pandas BDFL\) blog](#)
- [Statistical analysis made easy in Python with SciPy and pandas DataFrames, by Randal Olson](#)
- [Statistical Data Analysis in Python, tutorial videos, by Christopher Fonnesbeck from SciPy 2013](#)
- [Financial analysis in Python, by Thomas Wiecki](#)
- [Intro to pandas data structures, by Greg Reda](#)
- [Pandas and Python: Top 10, by Manish Amde](#)
- [Pandas DataFrames Tutorial, by Karlijn Willems](#)
- [A concise tutorial with real life examples](#)

Pandas DataFrame
Attributes
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting, transposing
Combining/concatenating/joining/merging
Time Series related
Plotting
Serialization/IO/Conversion
Pandas Series
Attributes
Conversion
Indexing, Iteration
Function application, GroupBy & window
Binary Operator functions
Computation/descriptive stats
Reindexing/selection/label manipulation
Missing Data handling
Reshaping, sorting
Combining/concatenating/joining/merging
Time Series related
Accessors
Datetime properties
Datetime methods
Period properties
Timedelta properties
Timedelta methods
String handling
Categorical Accessor
Plotting
Index Objects
Numpy Arrays
Attributes
Array Creation Routines
Shape Manipulation - shape, resize, transpose
Calculation
Item Selection and Manipulation
Array Conversions



PSEUDO RANDOM NUMBER GENERATION

(IN PYTHON, NUMPY)

Pseudo Random Number Generation in Python

- Python Stdlib Random Module
- Numpy.random module
 - Legacy RandomState and
 - Generators

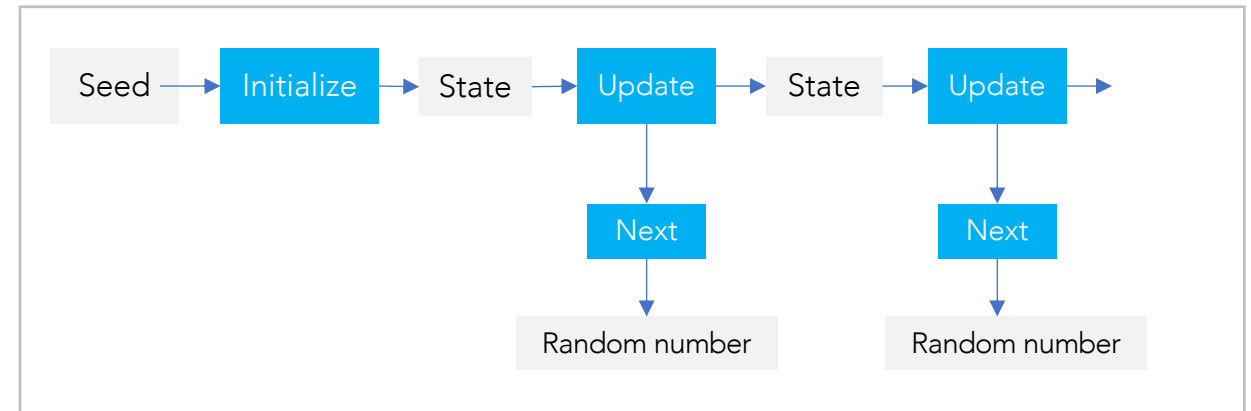
Library	Module	Class	Generator Type
Python Standard Library	random	random.Random(seed)	Mersenne Twister Generator MT19937
Numpy	random	numpy.random.RandomState(seed)	Mersenne Twister Generator MT19937
Numpy	random	numpy.random.Generator(bit_generator)	Permuted Congruential Generator PCG64 Link

Note : PRGN is a vast topic with wide applications in simulations (MonteCarlo), game development, cryptography etc. We shall be covering a small part of it from the point of view of generating random numbers as and when required for simple use cases.

Seeding for Initializing the Pseudo Random Number Generator

What is normally called a random number sequence in reality is a "pseudo-random" number sequence because the values are computed using a deterministic algorithm and probability plays no real role.

The "seed" is a starting point for the sequence and the guarantee is that if you start from the same seed you will get the same sequence of numbers. This is very useful for example for debugging (when you are looking for an error in a program you need to be able to reproduce the problem and study it, a non-deterministic program would be much harder to debug because every run would be different). [Link](#)



Pseudo-Random Number Generator (PRNG) generates a sequence of random numbers based on a seed. Using the same seed makes the PRNG produce the same sequence of random numbers.

*Source : Real-World Cryptography Version 12 by David Wong

Evaluation Metrics PRNG

Statistical Quality
Prediction Difficulty
Reproducible Results
Multiple Streams
Period
Useful Features
Time Performance
Space Usage
Code Size & Complexity
k-Dimensional Equidistribution

```
#Setting the seed of the random number generator
>>>np.random.seed(123456)
#Sequence of numbers generated starting from the seed position
>>>np.random.randint(10,50,5)
array([11, 37, 42, 33, 46])

#Note that internally the state of the generator has changed and a new seed is set for the next function call
>>>np.random.randint(10,50,5)
#Output shall be based on deterministically created new seed internally
array([18, 20, 20, 22, 21])

#Resetting the seed will produce the same sequence of random numbers
>>>np.random.seed(123456)
>>>np.random.randint(10,50,5)
array([11, 37, 42, 33, 46])
```

Seeding for Initializing the Pseudo Random Number Generator

Python Random Module	import random random.seed(10)
numpy.random module	import numpy as np np.random.seed(10) #Seed is changed at global level
Legacy RandomState	rs = np.random.RandomState(12345) #12345 is the seed
Generators	rng = np.random.default_rng(12345) #12345 is the seed

What does 'seeding' mean?

It means: pick a place to start.
Think of a pseudo random number generator as just a really long list of numbers.
This list is circular, it eventually repeats.
To use it, you need to pick a starting place. This is called a "seed".

Computers are deterministic

The computer random number generator is really a "pseudo-random" number generator, because computers are deterministic. If you start from the same point, you get the same sequence! Most random number generators have a method for choosing the starting point, e.g., the clock time (perhaps in milliseconds), so you get a different sequence each time you call the random number generator.

Although this sounds good, it is actually recommended that you set the seed of the random number generator yourself, as you may actually want repeatability: as you modify your code and assess improvements, you don't want to have to worry about whether the improvements (or lack of them) comes from a different set of inputs or from the different code. Of course, you also probably want to check your code with a variety of "random" sequences.

Lowest level random number generators give uniform deviates, i.e., equal probability of results in some range (usually 0 to 1 for floats). E.g., python random.random, random.seed, [numpy.random.random](#) , [numpy.random.seed](#), general random number class [numpy.random.RandomState](#) [Link](#)

[random.seed\(a, version\)](#) in python is used to initialize the pseudo-random number generator (PRNG).

PRNG is algorithm that generates sequence of numbers approximating the properties of random numbers. These random numbers can be reproduced using the seed value. So, if you provide seed value, PRNG starts from an arbitrary starting state using a seed.

[Argument a](#) is the seed value. If the a value is None, then by default, current system time is used. and [version](#) is An integer specifying how to convert the a parameter into a integer. Default value is 2.
if you want the same random number to be reproduced, provide the seed. Specifically, the same seed. [Link](#)

When you call random.seed(), it sets the random seed. [The sequence of numbers you generate from that point forwards will always be the same.](#)

The thing is, you're only set the seed once, and then you're calling np.random.uniform() three times. That means you're getting the next three numbers from your random.seed(). Of course they're different – you haven't reset the seed in between. But every time you run the program, you'll get the same sequence of three numbers, because you set the seed to the same thing before generating them all.

Setting the seed only affects the next random number to be generated, because of how pseudo-random number generation (which np.random uses) works: it uses the seed to generate a new random number deterministically, and then uses the generated number to set a new seed for the next number. It effectively boils down to getting a really really long sequence of random numbers that will, eventually, repeat itself. When you set the seed, you're jumping to a specified point in that sequence – you're not keeping the code there, though.

[Link](#)



Random Module standard Library in Python

Functions supplied by the Random Module

Bookkeeping functions

seed	Initialize the random number generator
getstate	Return an object capturing the current internal state of the generator.
setstate	setstate() restores the internal state of the generator to what it was at the time getstate() was called.

Functions for integers

randrange	Return a randomly selected element from range(start, stop, step).
randint	Return a random integer N such that $a \leq N \leq b$

Functions for sequences

choice	Return a random element from the non-empty sequence seq.
choices	Return a k sized list of elements chosen from the population with replacement.
shuffle	Shuffle the sequence x in place.
sample	Return a k length list of unique elements chosen from the population sequence or set. Used for random sampling without replacement

Real-valued distributions

random	Return the next random floating point number in the range [0.0, 1.0).
uniform	Return a random floating point number N such that $a \leq N \leq b$ for $a \leq b$ and $b \leq N \leq a$ for $b < a$.
betavariate	Beta distribution.
expovariate	Exponential distribution.
gammavariate	Gamma distribution.
gauss	Gaussian distribution.
lognormvariate	Log normal distribution.
normalvariate	Normal distribution. mu is the mean, and sigma is the standard deviation.
paretovariate	Pareto distribution. alpha is the shape parameter.
triangular	Return a random floating point number N such that $low \leq N \leq high$ and with the specified mode between those bounds
vonmisesvariate	von Mises distribution
weibullvariate	Weibull distribution. alpha is the scale parameter and beta is the shape parameter.

random.seed(a=None, version=2)

Initialize the random number generator. If a is omitted or None, the current system time is used. [..]

Alternative Generator

class random.Random([seed]) Class that implements the default pseudo-random number generator used by the [random](#) module.

Note : The functions supplied by Random module are actually bound methods of a hidden instance of the [random.Random](#) class. You can instantiate your own instances of [Random](#) to get generators that don't share state.

Python uses the Mersenne Twister as the core generator.

PRGN by Mersenne Twister Generator

```
>>> import random
>>> #random.seed(123456)
>>> rints = random.randint(low=0, high=10, size=(4,4))
>>> rchoice = random.choice('abcdefgh')
```

Also, check discussion thread on generating random integers.

PRNG in Numpy.Random Module

using combinations of a [BitGenerator](#) to create sequences and a [Generator](#) to use those sequences to sample from different statistical distributions

The [BitGenerator](#) has a limited set of responsibilities. It manages state and provides functions to produce random doubles and random unsigned 32- and 64-bit values.

The [random generator](#) takes the bit generator-provided stream and transforms them into more useful distributions, e.g., simulated normal random values. This structure allows alternative bit generators to be used with little code duplication.

The [Generator](#) is the user-facing object that is nearly identical to the legacy [RandomState](#). It accepts a bit generator instance as an argument. The default is currently [PCG64](#) but this may change in future versions. As a convenience NumPy provides the [default_rng](#) function to hide these details:

```
# Construct a new Generator
import numpy as np
>>> rng = np.random.default_rng(123456)
>>> rints = rng.integers(low=0, high=10, size=(4,4))
>>> rints
array([[0, 6, 8, 3], [3, 0, 0, 9], [2, 9, 3, 4], [5, 2, 4, 6]])
```

```
>>> rng = np.random.default_rng(123456)
>>> rng.random(4)
array([0.63651375, 0.38481166, 0.04744542,
       0.95525274])
```

```
>>> rng = np.random.default_rng(123456)
>>> rng.standard_normal(10)
array([ 0.1928212 , -0.06550702, 0.43550665,
        0.88235875, 0.37132785, 1.15998882, 0.37835254, -
        0.11718594, 2.20800921, 1.95324484])
```

[PRNG by Mersenne Twister Generator](#)

[PRNG by PCG](#)

Random sampling ([numpy.random](#))

Using RandomState Class ([Legacy Random Generation](#))

Seeding and State	
get_state()	Return a tuple representing the internal state of the generator.
set_state(state)	Set the internal state of the generator from a tuple.
seed(self[, seed])	Reseed a legacy MT19937 BitGenerator
Simple random data	
rand(d0, d1, ..., dn)	Random values in a given shape.
randn(d0, d1, ..., dn)	Return a sample (or samples) from the "standard normal" distribution.
randint(low[, high, size, dtype])	Return random integers from low (inclusive) to high (exclusive).
random_integers(low[, high, size])	Random integers of type np.int_ between low and high, inclusive.
random_sample([size])	Return random floats in the half-open interval [0.0, 1.0).
choice(a[, size, replace, p])	Generates a random sample from a given 1-D array
bytes(length)	Return random bytes.
Permutations	
shuffle(x)	Modify a sequence in-place by shuffling its contents.
permutation(x)	Randomly permute a sequence, or return a permuted range.

- RandomState is a class in NumPy. We use an instance of this class to manage random number generation.
- Random numbers are generated by methods in the class (e.g. the rand or randn methods).
- Each instance of RandomState comes with its own specific random number stream.
- The random number stream is initialized ("seeded") when you create a RandomState instance.
- You can reset the "seed" using the seed() method.
- We can use the RandomState objects to have different random streams or to reset a stream.

Distributions	
beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution.
exponential([scale, size])	Draw samples from an exponential distribution.
f(dfnum, dfden[, size])	Draw samples from an F distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
geometric(p[, size])	Draw samples from the geometric distribution.
gumbel([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.
laplace([loc, scale, size])	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
logistic([loc, scale, size])	Draw samples from a logistic distribution.
lognormal([mean, sigma, size])	Draw samples from a log-normal distribution.
logseries(p[, size])	Draw samples from a logarithmic series distribution.
multinomial(n, pvals[, size])	Draw samples from a multinomial distribution.
multivariate_normal(mean, cov[, size, ...])	Draw random samples from a multivariate normal distribution.
negative_binomial(n, p[, size])	Draw samples from a negative binomial distribution.
noncentral_chisquare(df, nonc[, size])	Draw samples from a noncentral chi-square distribution.
noncentral_f(dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
normal([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.
pareto(a[, size])	Draw samples from a Pareto II or Lomax distribution with specified shape.
poisson([lam, size])	Draw samples from a Poisson distribution.
power(a[, size])	Draws samples in [0, 1] from a power distribution with positive exponent a - 1.
rayleigh([scale, size])	Draw samples from a Rayleigh distribution.
standard_cauchy([size])	Draw samples from a standard Cauchy distribution with mode = 0.
standard_exponential([size])	Draw samples from the standard exponential distribution.
standard_gamma(shape[, size])	Draw samples from a standard Gamma distribution.
standard_normal([size])	Draw samples from a standard Normal distribution (mean=0, stdev=1).
standard_t(df[, size])	Draw samples from a standard Student's t distribution with df degrees of freedom.
triangular(left, mode, right[, size])	Draw samples from the triangular distribution over the interval [left, right].
uniform([low, high, size])	Draw samples from a uniform distribution.
vonmises(mu, kappa[, size])	Draw samples from a von Mises distribution.
wald(mean, scale[, size])	Draw samples from a Wald, or inverse Gaussian, distribution.
weibull(a[, size])	Draw samples from a Weibull distribution.
zipf(a[, size])	Draw samples from a Zipf distribution.

Use [numpy.random.default_rng](#) for PRNG instead of the legacy RandomState

`numpy.random.default_rng()` : Construct a new Generator with the default BitGenerator (PCG64).

```
import numpy as np
#Using numpy.random.default\_rng to instantiate a Generator with numpy's
default BitGenerator
# We can specify a seed for reproducible results
>>> rng = np.random.default_rng(12345)
>>> rints = rng.integers(low=0, high=10, size=3)
>>> rints
array([6, 2, 7])

>>> type(rints[0])
<class 'numpy.int64'>
```

Instead of reseeding, instantiate a new Generator object using `np.random.default_rng`

Why So ?

Avoid the use of `numpy.random.seed(seed)` for fixing reproducibility as this operates at the global state level. For multiple threads and projects with imports and scripts, the results will get obscured in case of multiple declarations of the `np.random.seed(seed)` with different seed arguments.

[Accessing the Bit Generator](#) - Attribute

bit_generator	Gets the bit generator instance used by the generator
-------------------------------	---

[Simple random data](#) - Methods

integers	Return random integers from low (inclusive) to high(exclusive), or if endpoint=True, low (inclusive) to high(inclusive).
random	Return random floats in the half-open interval [0.0, 1.0).
choice	Generates a random sample from a given array
bytes	Return random bytes.

[Permutations](#) - Methods

shuffle	Modify an array or sequence in-place by shuffling its contents.
permutation	Randomly permute a sequence, or return a permuted range.
permuted	Randomly permute x along axis axis.

[Distributions](#) - Methods

beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution.
exponential([scale, size])	Draw samples from an exponential distribution.
f(dfnum, dfden[, size])	Draw samples from an F distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
geometric(p[, size])	Draw samples from the geometric distribution.
gumbel([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.
laplace([loc, scale, size])	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
logistic([loc, scale, size])	Draw samples from a logistic distribution.
lognormal([mean, sigma, size])	Draw samples from a log-normal distribution.
logseries(p[, size])	Draw samples from a logarithmic series distribution.
multinomial(n, pvals[, size])	Draw samples from a multinomial distribution.
multivariate_hypergeometric(colors, nsample)	Generate variates from a multivariate hypergeometric distribution.
multivariate_normal(mean, cov[, size, ...])	Draw random samples from a multivariate normal distribution.
negative_binomial(n, p[, size])	Draw samples from a negative binomial distribution.
noncentral_chisquare(df, nonc[, size])	Draw samples from a noncentral chi-square distribution.
noncentral_f(dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
normal([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.
pareto(a[, size])	Draw samples from a Pareto II or Lomax distribution with specified shape.
poisson([lam, size])	Draw samples from a Poisson distribution.
power(a[, size])	Draws samples in [0, 1] from a power distribution with positive exponent a - 1.
rayleigh([scale, size])	Draw samples from a Rayleigh distribution.
standard_cauchy([size])	Draw samples from a standard Cauchy distribution with mode = 0.
standard_exponential([size, dtype, method, out])	Draw samples from the standard exponential distribution.
standard_gamma(shape[, size, dtype, out])	Draw samples from a standard Gamma distribution.
standard_normal([size, dtype, out])	Draw samples from a standard Normal distribution (mean=0, stdev=1).
standard_t(df[, size])	Draw samples from a standard Student's t distribution with df degrees of freedom.
triangular(left, mode, right[, size])	Draw samples from the triangular distribution over the interval [left, right].
uniform([low, high, size])	Draw samples from a uniform distribution.
vonmises(mu, kappa[, size])	Draw samples from a von Mises distribution.
wald(mean, scale[, size])	Draw samples from a Wald, or inverse Gaussian, distribution.
weibull(a[, size])	Draw samples from a Weibull distribution.



Reseeding a BitGenerator and Recreating a Bit Generator

Prior to every function/method call

```
import numpy as np
from numpy.random import MT19937
from numpy.random import RandomState, SeedSequence
import matplotlib.pyplot as plt
```

```
seed=123456789
```

```
# Reseed a BitGenerator
```

```
np.random.seed(seed)
```

```
r1 = np.random.randint(1, 6, 1000)
```

```
# Reseed a BitGenerator
```

```
np.random.seed(seed)
```

```
r2 = np.random.randint(1, 6, 1000)
```

```
# Recreate a BitGenerator
```

```
rs = RandomState(MT19937(SeedSequence(seed)))
```

```
c1 = rs.randint(1, 6, 1000)
```

```
# Recreate a BitGenerator
```

```
rs = RandomState(MT19937(SeedSequence(seed)))
```

```
c2 = rs.randint(1, 6, 1000)
```

```
# Visualise results
```

```
fig, axes = plt.subplots(1, 2)
```

```
axes[0].hist(r1, 11, density=True)
```

```
axes[0].hist(r2, 11, density=True)
```

```
axes[0].set_title('Reseed a BitGenerator')
```

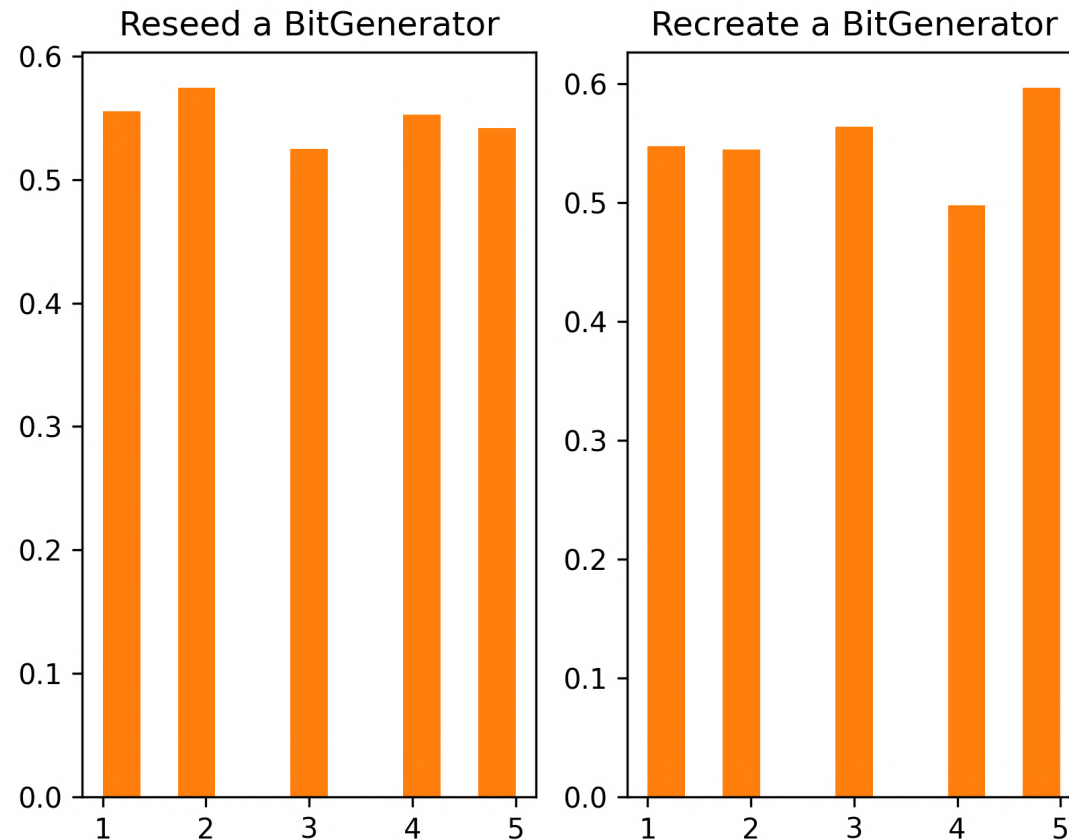
```
axes[1].hist(c1, 11, density=True)
```

```
axes[1].hist(c2, 11, density=True)
```

```
axes[1].set_title('Recreate a BitGenerator')
```

```
plt.show()
```

[Link](#)



Note : We are **reseeding/recreating the BitGenerator everytime** we are using the randint method from np.random or RandomState Class. Hence, we are getting the same sequence of numbers. Thus, resulting in each of the subplot above having two set of overlapping bar plots of same height.

Reseeding a BitGenerator and Recreating a Bit Generator

Only once in the script

```
import numpy as np
from numpy.random import MT19937
from numpy.random import RandomState, SeedSequence
import matplotlib.pyplot as plt
```

```
seed=123456789
```

```
# Reseed a BitGenerator
```

```
np.random.seed(seed)
r1 = np.random.randint(1, 6, 1000)
r2 = np.random.randint(1, 6, 1000)
```

```
# Recreate a BitGenerator
```

```
rs = RandomState(MT19937(SeedSequence(seed)))
c1 = rs.randint(1, 6, 1000)
c2 = rs.randint(1, 6, 1000)
```

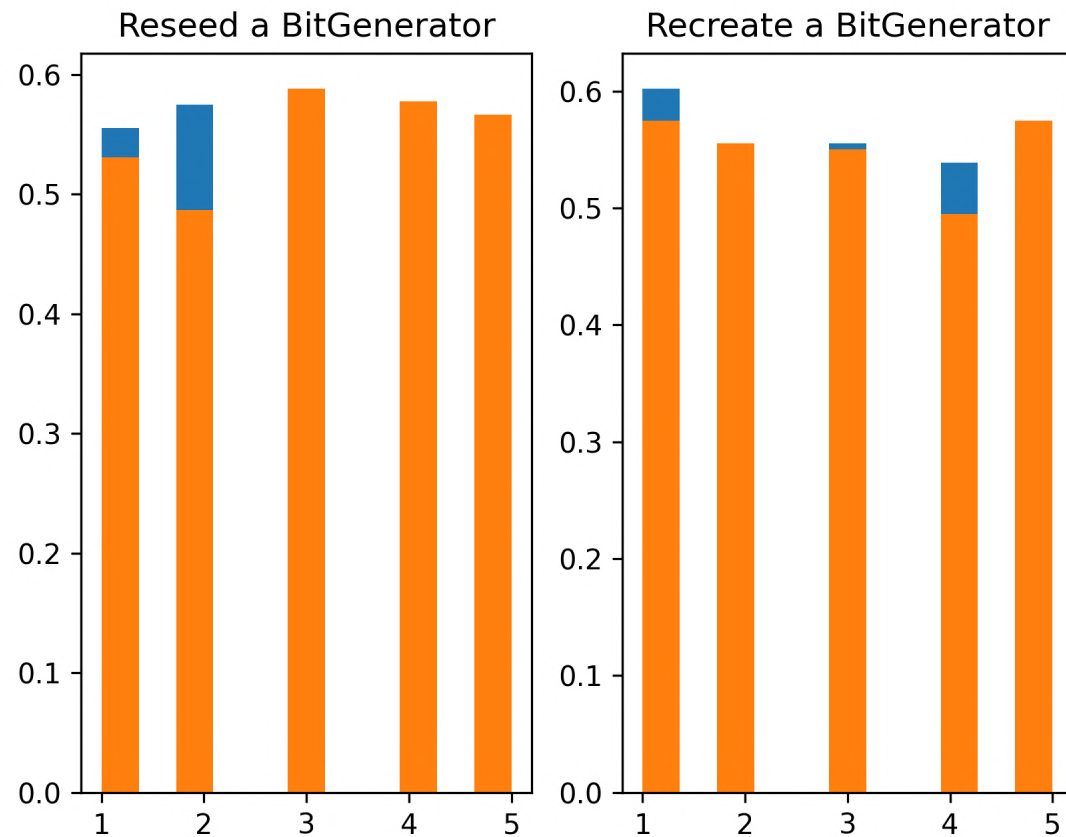
```
# Visualise results
```

```
fig, axes = plt.subplots(1, 2)
axes[0].hist(r1, 11, density=True)
axes[0].hist(r2, 11, density=True)
axes[0].set_title('Reseed a BitGenerator')
```

```
axes[1].hist(c1, 11, density=True)
axes[1].hist(c2, 11, density=True)
axes[1].set_title('Recreate a BitGenerator')
```

```
plt.show()
```

[Link](#)



Note : We are **reseeding/recreating the BitGenerator only once**. After that we are making two successive randint calls from np.random or RandomState Class. The two calls in succession will result in two different sequence of integers. Thus, resulting in each of the subplot above having two set of overalpping bar plots with different heights. The blue color indicates the height difference.

```
import numpy as np
import matplotlib.pyplot

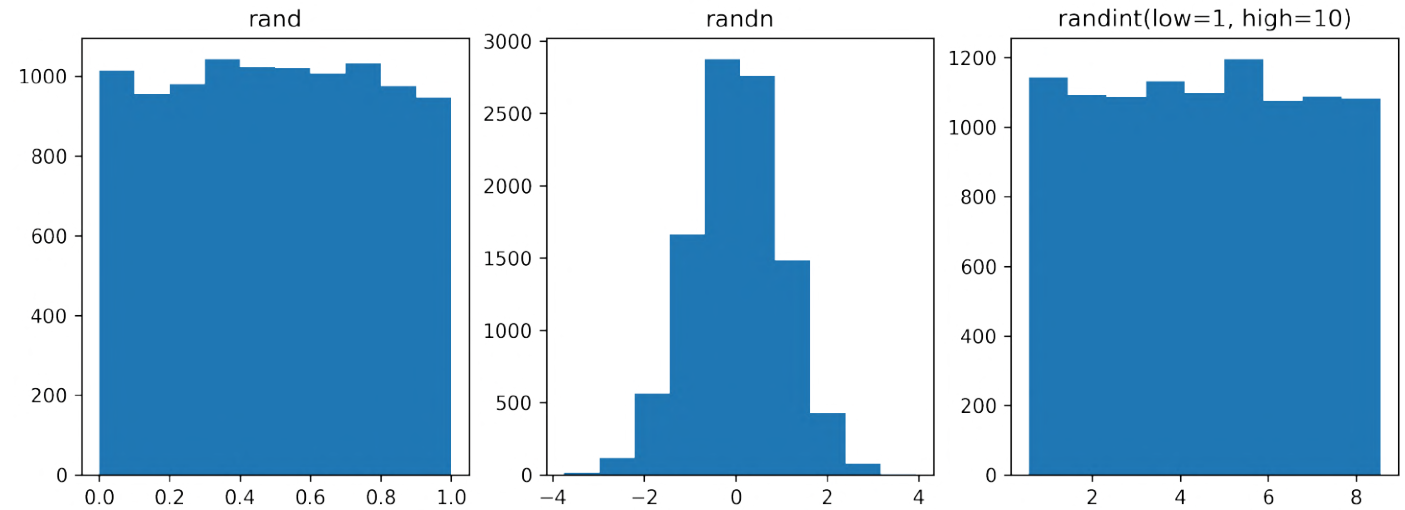
plt.rcParamsdefaults()
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].hist(np.random.rand(10000))
axes[0].set_title("rand")

axes[1].hist(np.random.randn(10000))
axes[1].set_title("randn")

axes[2].hist(np.random.randint(low=1, high=10, size=10000), bins=9,
            align='left')
axes[2].set_title("randint(low=1, high=10)")

fig.savefig('LegacyNpRandomFunctions.png', dpi = 300,
            transparent = True)
plt.show()
```



```
import numpy as np
import matplotlib.pyplot

plt.rcParams.defaults()
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

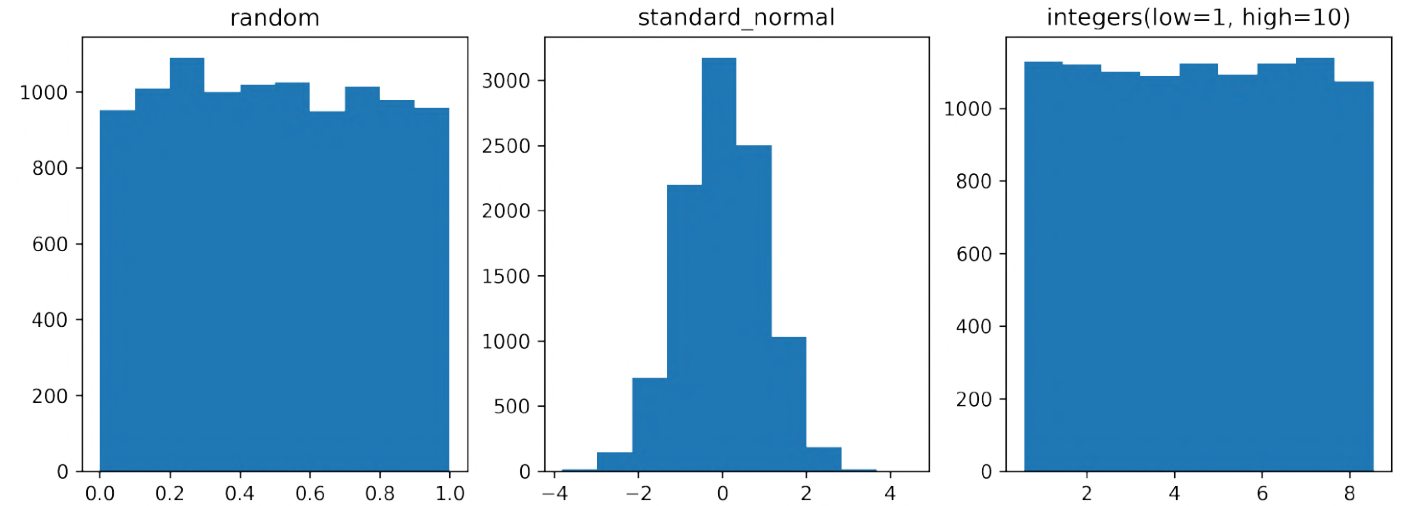
#Instantiating a generator
rng = np.random.default_rng(12345)

axes[0].hist(rng.random(10000))
axes[0].set_title("random")

axes[1].hist(rng.standard_normal(10000))
axes[1].set_title("standard_normal")

axes[2].hist(rng.integers(low=1, high=10, size=10000), bins=9,
             align='left')
axes[2].set_title("integers(low=1, high=10)")

fig.savefig('GeneratorObject.png', dpi = 300, transparent = True)
plt.show()
```



HANDLING DATE AND TIME

Classes/Objects to handle date and time

Python, Numpy, Pandas, Matplotlib

Matplotlib	date						
Pandas	timestamp	timedelta	period	dateoffset	datetimeindex	PeriodIndex	timedeltaindex
Numpy	datetime64	timedelta64					NumPy's datetime64 object allows you to set its precision from hours all the way to attoseconds (10^{-18}).
Python	datetime	timedelta	date	time	tzinfo	timezone	The datetime module's datetime object has microsecond precision (one-millionth of a second).
A Specific Date and Time		Absolute time duration	Note : <u>timedelta</u> only holds days, seconds, microseconds internally				
With TimeZone support							

Conversion Matrix between datetime, Timestamp and datetime64

Across Pandas, numpy, Python, matplotlib, strings

Go to
Matplotlib.dates

Conversion Matrix			Python	Numpy	Pandas	matplotlib
	From To	string	datetime	datetime64	timestamp	matplotlib.date
Python	string strobj: string object		>>>dt.strptime(strobj)	np.datetime64('2005-02-25') #strings in ISO 8601 date or datetime format.	pd.Timestamp('2017-01-01T12') #datetime like string	datestr2num(strobj)
Python	datetime import datetime as dt dtobj : datetime object	dtobj.strptime('%b %d, %Y') dtobj.strptime('The date is %b %d, %Y')		np.datetime64(dtobj)	pd.Timestamp(dtobj) # or pd.to_datetime(dtobj)	date2num(dtobj)
Numpy	datetime64 import numpy as np dt64 : datetime64 object	np.datetime_as_string(d, unit='h') #Check datetime support functions	>>> dt64 = np.datetime64('2017-10-24 05:34:20.123456') >>> unix_epoch = np.datetime64(0, 's') >>> one_second = np.timedelta64(1, 's') >>> seconds_since_epoch = (dt64 - unix_epoch) / one_second >>> seconds_since_epoch 1508823260.123456 >>> datetime.datetime.utcfromtimestamp(seconds_since_epoch) datetime.datetime(2017, 10, 24, 5, 34, 20, 123456)		pd.Timestamp(dt64)	
Pandas	timestamp import pandas as pd ts : timestamp object	ts.dt.strptime('%b %d %Y') #dt is datetime accessor. Link	ts.to_pydatetime()	ts.to_datetime64() # or ts.to_numpy() # or ts.asm8		date2num(ts.to_pydatetime())
matplotlib	matplotlib.date*		num2date(mdate)			

*Currently Matplotlib time is only converted back to datetime objects, which have microsecond resolution, and years that only span 0000 to 9999. Check [Link](#).

<https://stackoverflow.com/questions/466345/converting-string-into-datetime/470303#470303>

<https://www.seehuhn.de/pages/pdate.html#unix>



Conversion Matrix between Unix time vs datetime, Timestamp and datetime64

Across Pandas, numpy, Python, matplotlib, strings

Go to
Matplotlib.dates

Conversion Matrix		Platform Independent	Conversion Matrix	Platform Independent
	From \ To	Unix epoch or Unix time or POSIX time or Unix timestamp		Unix epoch or Unix time or POSIX time or Unix timestamp
Python	string strobj: string object	<pre>>>> datetime.strptime(string).timestamp() >>> datetime.datetime(2021,8,1,0,0).timestamp() 1627756200.0</pre>	string	<pre>>>> import datetime >>> timestamp = datetime.datetime.fromtimestamp(1600000000) >>> print(timestamp.strftime('%Y-%m-%d %H:%M:%S')) 2020-09-13 17:56:40</pre>
Python	datetime import datetime as dt dtobj : datetime object	datetime.timestamp() Check link	datetime	<pre>>>> dt.datetime.fromtimestamp(1172969203.1) datetime.datetime(2007, 3, 4, 6, 16, 43, 100000)</pre>
Numpy	datetime64 import numpy as np dt64 : datetime64 object	<pre>>>> dt64 = np.datetime64('2017-10-24 05:34:20.123456') >>> unix_epoch = np.datetime64(0, 's') >>> one_second = np.timedelta64(1, 's') >>> seconds_since_epoch = (dt64 - unix_epoch) / one_second >>> seconds_since_epoch 1508823260.123456 >>> datetime.datetime.utcfromtimestamp(seconds_since_epoch) datetime.datetime(2017, 10, 24, 5, 34, 20, 123456)</pre>	datetime64	<pre>>>> dt64 = np.datetime64('2017-10-24 05:34:20.123456') >>> unix_epoch = np.datetime64(0, 's') >>> one_second = np.timedelta64(1, 's') >>> seconds_since_epoch = (dt64 - unix_epoch) / one_second >>> seconds_since_epoch 1508823260.123456 >>> datetime.datetime.utcfromtimestamp(seconds_since_epoch) datetime.datetime(2017, 10, 24, 5, 34, 20, 123456)</pre>
Pandas	timestamp import pandas as pd ts : timestamp object	<pre>In [64]: stamps = pd.date_range("2012-10-08 18:15:05", periods=4, freq="D") In [66]: (stamps - pd.Timestamp("1970-01-01")) // pd.Timedelta("1s") Int64Index([1349720105, 1349806505, 1349892905, 1349979305], dtype='int64')</pre>	timestamp	<pre>>>> pd.to_datetime([1627756200.0, 1600000000], unit = 's') DatetimeIndex(['2021-07-31 18:30:00', '2020-09-13 12:26:40'], dtype='datetime64[ns]', freq=None) Link</pre>
matplotlib	matplotlib.date	matplotlib.dates. num2epoch (d) [source]	matplotlib.date	matplotlib.dates. epoch2num (unixtime) [source]

<https://stackoverflow.com/questions/466345/converting-string-into-datetime/470303#470303>

<https://www.seehuhn.de/pages/pdate.html#unix>



Internal Storage of Datetime objects in Python

```
/* Fields are packed into successive bytes, each viewed as unsigned and
 * big-endian, unless otherwise noted:
 *
 * byte offset
 * 0      year   2 bytes, 1-9999
 * 2      month  1 byte, 1-12
 * 3      day    1 byte, 1-31
 * 4      hour   1 byte, 0-23
 * 5      minute 1 byte, 0-59
 * 6      second 1 byte, 0-59
 * 7      usecond 3 bytes, 0-999999
 * 10
 */
...

/* # of bytes for year, month, day, hour, minute, second, and usecond. */
#define _PyDateTime_DATETIME_DATASIZE 10
...

/* The datetime and time types have hashcodes, and an optional tzinfo member,
 * present if and only if hastzinfo is true.
 */
#define _PyTZINFO_HEAD      \
    PyObject_HEAD           \
    Py_hash_t hashcode;      \
    char hastzinfo;          /* boolean flag */
...
```

<https://stackoverflow.com/questions/52357460/how-does-python-store-datetime-internally>

```
/* All datetime objects are of PyDateTime_DateTimeType, but that can be
 * allocated in two ways too, just like for time objects above. In addition,
 * the plain date type is a base class for datetime, so it must also have
 * a hastzinfo member (although it's unused there).
 */
...
#define _PyDateTime_DATETIMEHEAD      \
    _PyTZINFO_HEAD                     \
    unsigned char data[_PyDateTime_DATETIME_DATASIZE];

typedef struct
{
    _PyDateTime_DATETIMEHEAD
} _PyDateTime_BaseDateTime; /* hastzinfo false */

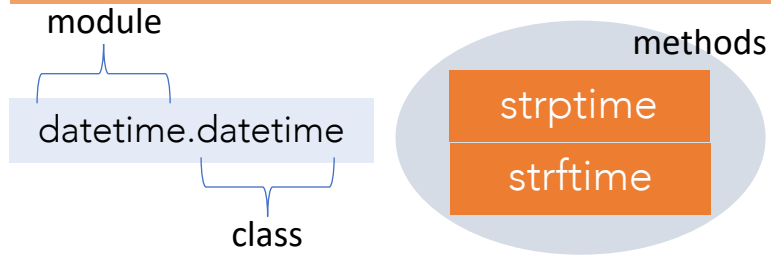
typedef struct
{
    _PyDateTime_DATETIMEHEAD
    unsigned char fold;
    PyObject *tzinfo;
} PyDateTime_DateTime; /* hastzinfo true */
The 48-byte count
```

The 48-byte count breaks down as follows:

- 8-byte refcount
- 8-byte type pointer
- 8-byte cached hash
- 1-byte "hastzinfo" flag
- 7-byte padding
- 10-byte manually packed char[10] containing datetime data
- 6-byte padding

Handling Datetime objects and strings

Strings to datetime



strptime means string parser, this will convert a string format to datetime.

strftime means string formatter, this will format a datetime object to string format.

String to datetime object = strptime
datetime object to other formats = strftime

Jun 1 2021 1:33PM
is equals to
%b %d %Y %I:%M%p

- %b :Month as locale's abbreviated name(Jun)
- %d :Day of the month as a zero-padded decimal number(1)
- %Y :Year with century as a decimal number(2015)
- %I :Hour (12-hour clock) as a zero-padded decimal number(01)
- %M :Minute as a zero-padded decimal number(33)
- %p :Locale's equivalent of either AM or PM(PM)

<https://stackoverflow.com/questions/466345/converting-string-into-datetime/470303#470303>

<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior>

strftime() and strptime() Format Codes

Code	Meaning
%a	Weekday as Sun, Mon
%A	Weekday as full name as Sunday, Monday
%w	Weekday as decimal no as 0,1,2...
%d	Day of month as 01,02
%b	Months as Jan, Feb
%B	Months as January, February
%m	Months as 01,02
%y	Year without century as 11,12,13
%Y	Year with century 2011,2012
%H	24 Hours clock from 00 to 23
%I	12 Hours clock from 01 to 12
%p	AM, PM
%M	Minutes from 00 to 59
%S	Seconds from 00 to 59
%f	Microseconds 6 decimal numbers

Check
`Series.dt.strftime(*args, **kwargs)`

Date Format standards

[SO Thread](#)

Using Pandas for String conversion to datetime object : [link](#)
Check answer by [alexander](#), Also check [dt accessor methods](#)

```
>>>import pandas as pd
>>>dates = ['2015-12-25', '2015-12-26']
# 1) Use a list comprehension.
>>> [d.date() for d in pd.to_datetime(dates)]
[datetime.date(2015, 12, 25), datetime.date(2015, 12, 26)]
```

```
# 2) Convert the dates to a DatetimeIndex and extract the python dates.
>>> pd.DatetimeIndex(dates).date.tolist()
[datetime.date(2015, 12, 25), datetime.date(2015, 12, 26)]
```

dateutil capable of parsing most human intelligible date representations.

dateutil.parser.parse to format a date/time from known formats.

▪ Functions :

`parser.parse(parserinfo=None, **kwargs)`

`classmethod parser.isoparse(dt_str) # full ISO-8601 format parser`

▪ Classes :

`class dateutil.parser.parserinfo(dayfirst=False, yearfirst=False)`

ISO 8601 Date Format : YYYY-MM-DD

- YYYY is the year [all the digits, i.e. 2012]
- MM is the month [01 (January) to 12(December)]
- DD is the day [01 to 31]

<https://www.iso.org/iso-8601-date-and-time-format.html>
<https://www.cl.cam.ac.uk/~mgk25/iso-time.html>

RFC3339 Format

<https://datatracker.ietf.org/doc/html/rfc3339>



Handling Datetime objects and strings

datetime to strings

date and datetime objects (and time as well) support a [mini-language to specify output](#), and there are three ways to access it:

- [direct method call](#): `dt.strftime('format here')`
- [format method](#) (python 2.6+): `'{:format here}'.format(dt)`
- [f-strings](#) (python 3.6+): `f'{dt:format here}'`

So your example could look like:

- `dt.strftime('The date is %b %d, %Y')`
- `'The date is {:%b %d, %Y}'.format(dt)`
- `f'The date is {dt:%b %d, %Y}'`

In all three cases the output is:

The date is Feb 23, 2012

For completeness' sake: you can also directly access the attributes of the object, but then you only get the numbers:

```
'The date is %s/%s/%s' % (dt.month, dt.day, dt.year)
```

```
# The date is 02/23/2012
```

The time taken to learn the mini-language is worth it.

#[Link](#) :Check answer by [Ethan Furman](#)

<https://stackoverflow.com/questions/466345/converting-string-into-datetime/470303#470303>

Pandas timestamp/datetimeindex to strings

Importance of pandas.Series.dt

dt is the datetime accessor. Whenever your column values are Timestamps or Timedeltas, Pandas makes the dt accessor available. From that accessor, you can use many other datetime specific methods. In this case, I used strftime.

[Go to Slide](#)

[Pandas Documentation link](#)

Numpy captures 2 general time related concepts:

datetime64 : datetime object represents a single moment in time

Timedelta64 : An absolute time duration.

What is epoch time?

The Unix epoch (or Unix time or POSIX time or Unix timestamp) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally speaking the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for Unix time.

Assuming you are starting with some ISO 8601 strings in your `np.datetime64[ns]`, you can use `dt.tz_localize` to assign a time zone to them, then `dt.tz_convert` to convert them into another time zone

- NumPy has no separate date and time objects, just a single `datetime64` object to represent a single moment in time.
- The `datetime` module's `datetime` object has microsecond precision (one-millionth of a second). NumPy's `datetime64` object allows you to set its precision from hours all the way to attoseconds (10^{-18}).
- It's constructor is more flexible and can take a variety of inputs.

Datetime Units

The `Datetime` and `Timedelta` data types support a large number of time units, as well as generic units which can be coerced into any of the other units based on input data.

Datetimes are always stored based on POSIX time (though having a TAI mode which allows for accounting of leap-seconds is proposed), with an epoch of 1970-01-01T00:00Z. This means the supported dates are always a symmetric interval around the epoch, called "time span".(check [official link](#))

The date units are : Y, M, W, D

The time units are : h, m, s, ms, us, ns, ps, fs, as

[Numpy Official Link](#)

[Comprehensive Tutorial](#)

[SO Thread](#)



Datetime Support Functions in Numpy

Datetime Support Functions	
datetime_as_string(arr[, unit, timezone, ...])	Convert an array of datetimes into an array of strings.
datetime_data(dtype, /)	Get information about the step size of a date or time type.
Business Day Functions	
busdaycalendar([weekmask, holidays])	A business day calendar object that efficiently stores information defining valid days for the busday family of functions.
is_busday(dates[, weekmask, holidays, ...])	Calculates which of the given dates are valid days, and which are not.
busday_offset(dates, offsets[, roll, ...])	First adjusts the date to fall on a valid day according to the roll rule, then applies offsets to the given dates counted in valid days.
busday_count(begindates, enddates[, ...])	Counts the number of valid days between begindates and enddates, not including the day of enddates.

```
# Using np.arange to generate ranges of dates. Check link  
>>> np.arange('2005-02', '2005-03', dtype='datetime64[D]')
```

```
array(['2005-02-01', '2005-02-02', '2005-02-03', '2005-02-04', '2005-02-05', '2005-02-06',  
      '2005-02-07', '2005-02-08', '2005-02-09', '2005-02-10', '2005-02-11', '2005-02-12', '2005-  
      02-13', '2005-02-14', '2005-02-15', '2005-02-16', '2005-02-17', '2005-02-18', '2005-02-19',  
      '2005-02-20', '2005-02-21', '2005-02-22', '2005-02-23', '2005-02-24', '2005-02-25', '2005-  
      02-26', '2005-02-27', '2005-02-28'], dtype='datetime64[D]')
```

pandas captures 4 general time related concepts:

Date times : A specific date and time with timezone support.
Similar to [datetime.datetime](#) from the standard library.

Time deltas: An absolute time duration.
Similar to [datetime.timedelta](#) from the standard library.

Time spans: A span of time defined by a point in time and its associated frequency.

Date offsets: A relative time duration that respects calendar arithmetic.
Similar to [dateutil.relativedelta.relativedelta](#) from [dateutil](#) package.

What is epoch time?

The Unix epoch (or Unix time or POSIX time or Unix timestamp) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally speaking the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for Unix time.

Concept	Scalar Class	Array Class	pandas Data Type	Primary Creation Method
Date times	Timestamp	DatetimeIndex	datetime64[ns] or datetime64[ns, tz]	to_datetime or date_range
Time deltas	Timedelta	TimedeltaIndex	timedelta64[ns]	to_timedelta or timedelta_range
Time spans	Period	PeriodIndex	period[freq]	Period or period_range
Date offsets	DateOffset	None	None	DateOffset

Under the hood, pandas represents timestamps using instances of Timestamp and sequences of timestamps using instances of DatetimeIndex. For regular time spans, pandas uses Period objects for scalar values and PeriodIndex for sequences of spans

```
import datetime
In [2]: dti = pd.to_datetime( ["1/1/2018", np.datetime64("2018-01-01"), datetime.datetime(2018, 1, 1)]
In [3]: dti
Out[3]: DatetimeIndex(['2018-01-01', '2018-01-01', '2018-01-01'], dtype='datetime64[ns]', freq=None)

In [21]: pd.Series(pd.period_range("1/1/2011", freq="M", periods=3))
Out[21]:
0 2011-01
1 2011-02
2 2011-03 dtype: period[M]
```

[Pandas Official Link](#)

[Comprehensive Tutorial](#)

[SO Thread](#)



Pandas Date and time handling classes : Creation Methods

Concept	Scalar Class	Array Class	pandas Data Type	Primary Creation Method
Date times	Timestamp	DatetimeIndex	datetime64[ns] or datetime64[ns, tz]	to_datetime or date_range
Time deltas	Timedelta	TimedeltaIndex	timedelta64[ns]	to_timedelta or timedelta_range
Time spans	Period	PeriodIndex	period[freq]	Period or period_range
Date offsets	DateOffset	None	None	DateOffset

```
>>> df = pd.DataFrame({'year': [2015, 2016], 'month': [2, 3], 'day': [4, 5]})
>>> pd.to_datetime(df)
0 2015-02-04
1 2016-03-05
dtype: datetime64[ns]
```

```
>>> pd.to_datetime('13000101', format='%Y%m%d', errors='ignore')
datetime.datetime(1300, 1, 1, 0, 0)
>>> pd.to_datetime('13000101', format='%Y%m%d', errors='coerce')
NaT
```

```
>>> pd.to_timedelta(np.arange(5), unit='s')
TimedeltaIndex(['0 days 00:00:00', '0 days 00:00:01', '0 days 00:00:02',
                '0 days 00:00:03', '0 days 00:00:04'], dtype='timedelta64[ns]', freq=None)
>>> pd.to_timedelta(np.arange(5), unit='d')
TimedeltaIndex(['0 days', '1 days', '2 days', '3 days', '4 days'],
                dtype='timedelta64[ns]', freq=None)
```

```
# This particular day contains a day light savings time transition
>>> ts = pd.Timestamp("2016-10-30 00:00:00", tz="Europe/Helsinki")
```

```
# Respects absolute time
>>> ts + pd.Timedelta(days=1)
Timestamp('2016-10-30 23:00:00+0200', tz='Europe/Helsinki')
```

```
# Respects calendar time
>>> ts + pd.DateOffset(days=1)
Timestamp('2016-10-31 00:00:00+0200', tz='Europe/Helsinki')
```

Also, Check [Slide](#) for division of timedeltas to find number of years between two dates.

```
>>> pd.date_range(start='1/1/2018', end='1/08/2018')
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04', '2018-01-05',
              '2018-01-06', '2018-01-07', '2018-01-08'], dtype='datetime64[ns]',
              freq='D')
```

```
>>> pd.date_range(start='1/1/2018', periods=5, freq='M')
DatetimeIndex(['2018-01-31', '2018-02-28', '2018-03-31', '2018-04-30', '2018-05-31'],
              dtype='datetime64[ns]', freq='M')
```

```
>>> pd.date_range(start='1/1/2018', periods=5, freq=pd.offsets.MonthEnd(3))
DatetimeIndex(['2018-01-31', '2018-04-30', '2018-07-31', '2018-10-31', '2019-01-31'],
              dtype='datetime64[ns]', freq='3M')
```

```
>>> pd.period_range(start='2017-01-01', end='2018-01-01', freq='M')
PeriodIndex(['2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06',
            '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12',
            '2018-01'], dtype='period[M]')
```

```
>>> pd.timedelta_range(start='1 day', periods=4)
TimedeltaIndex(['1 days', '2 days', '3 days', '4 days'], dtype='timedelta64[ns]', freq='D')
```

```
>>> pd.timedelta_range(start='1 day', end='2 days', freq='6H')
TimedeltaIndex(['1 days 00:00:00', '1 days 06:00:00', '1 days 12:00:00',
                '1 days 18:00:00', '2 days 00:00:00'], dtype='timedelta64[ns]', freq='6H')
```

```
>>> pd.timedelta_range(start='1 day', end='5 days', periods=4)
TimedeltaIndex(['1 days 00:00:00', '2 days 08:00:00', '3 days 16:00:00', '5 days 00:00:00'],
              dtype='timedelta64[ns]', freq=None)
```

Frequency codes/ Offset Aliases

Alias	Description
B	business day frequency
C	custom business day frequency
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency
BQ	business quarter end frequency
QS	quarter start frequency
BQS	business quarter start frequency
A, Y	year end frequency
BA, BY	business year end frequency
AS, YS	year start frequency
BAS, BYS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds



Datetime Index

The DatetimeIndex class contains many time series related optimizations:

- A large range of dates for various offsets are pre-computed and cached under the hood in order to make generating subsequent date ranges very fast (just have to grab a slice).
- Fast shifting using the shift method on pandas objects.
- Unioning of overlapping DatetimeIndex objects with the same frequency is very fast (important for fast data alignment).
- Quick access to date fields via properties such as year, month, etc.
- Regularization functions like snap and very fast as of logic.

REMEMBER

- Valid date strings can be converted to datetime objects using to_datetimefunction or as part of read functions.
- Datetime objects in pandas support calculations, logical operations and convenient date-related properties using the dt accessor.
- A DatetimeIndex contains these date-related properties and supports convenient slicing.
- Resample is a powerful method to change the frequency of a time series.

Attributes	
year	The year of the datetime.
month	The month as January=1, December=12.
day	The day of the datetime.
hour	The hours of the datetime.
minute	The minutes of the datetime.
second	The seconds of the datetime.
microsecond	The microseconds of the datetime.
nanosecond	The nanoseconds of the datetime.
date	Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).
time	Returns numpy array of datetime.time.
timetz	Returns numpy array of datetime.time also containing timezone information.
dayofyear	The ordinal day of the year.
day_of_year	The ordinal day of the year.
weekofyear	(DEPRECATED) The week ordinal of the year.
week	(DEPRECATED) The week ordinal of the year.
dayofweek	The day of the week with Monday=0, Sunday=6.
day_of_week	The day of the week with Monday=0, Sunday=6.
weekday	The day of the week with Monday=0, Sunday=6.
quarter	The quarter of the date.
tz	Return timezone, if any.
freq	Return the frequency object if it is set, otherwise None.
freqstr	Return the frequency object as a string if its set, otherwise None.
is_month_start	Indicates whether the date is the first day of the month.
is_month_end	Indicates whether the date is the last day of the month.
is_quarter_start	Indicator for whether the date is the first day of a quarter.
is_quarter_end	Indicator for whether the date is the last day of a quarter.
is_year_start	Indicate whether the date is the first day of a year.
is_year_end	Indicate whether the date is the last day of the year.
is_leap_year	Boolean indicator if the date belongs to a leap year.
inferred_freq	Tries to return a string representing a frequency guess, generated by infer_freq.

Datetime Index

Check if two datetime indexes are equal

```
>>> import pandas as pd
>>> idx1 = pd.date_range('2020-01-01','2020-12-31',freq='D')
>>> idx2 = pd.date_range('2020-01-01','2020-11-01',freq='D')
>>> idx3 = pd.date_range('2020-01-01','2020-12-31',freq='D')

>>> help(idx1.equals)
Help on method equals in module
pandas.core.indexes.datetimelike: equals(other: object) -> bool
method of pandas.core.indexes.datetimes.DatetimeIndex instance
Determines if two Index objects contain the same elements.

>>> print(idx1.equals(idx2))
False

>>> print(idx1.equals(idx3))
True
```

Methods

normalize(*args, **kwargs)	Convert times to midnight.
strftime(*args, **kwargs)	Convert to Index using specified date_format.
snap([freq])	Snap time stamps to nearest occurring frequency.
tz_convert(tz)	Convert tz-aware Datetime Array/Index from one time zone to another.
tz_localize	Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.
round(*args, **kwargs)	Perform round operation on the data to the specified freq.
floor(*args, **kwargs)	Perform floor operation on the data to the specified freq.
ceil(*args, **kwargs)	Perform ceil operation on the data to the specified freq.
to_period(*args, **kwargs)	Cast to PeriodArray/Index at a particular frequency.
to_perioddelta(freq)	Calculate TimedeltaArray of difference between index values and index converted to PeriodArray at specified freq.
to_pydatetime	Return Datetime Array/Index as object ndarray of datetime.datetime objects.
to_series	Create a Series with both index and values equal to the index keys useful with map for returning an indexer based on an index.
to_frame([index, name])	Create a DataFrame with a column containing the Index.
month_name	Return the month names of the DateTimeIndex with specified locale.
day_name(*args, **kwargs)	Return the day names of the DateTimeIndex with specified locale.
mean(*args, **kwargs)	Return the mean value of the Array.
to_perioddelta(freq)	Calculate TimedeltaArray of difference between index values and index converted to PeriodArray at specified freq.
to_pydatetime	Return Datetime Array/Index as object ndarray of datetime.datetime objects.
to_series	Create a Series with both index and values equal to the index keys useful with map for returning an indexer based on an index.

DateOffset Objects

Date Offset	Frequency String	Description
DateOffset	None	Generic offset class, defaults to absolute 24 hours
BDay or BusinessDay	'B'	business day (weekday)
CDay or CustomBusinessDay	'C'	custom business day
Week	'W'	one week, optionally anchored on a day of the week
WeekOfMonth	'WOM'	the x-th day of the y-th week of each month
LastWeekOfMonth	'LWOM'	the x-th day of the last week of each month
MonthEnd	'M'	calendar month end
MonthBegin	'MS'	calendar month begin
BMonthEnd or BusinessMonthEnd	'BM'	business month end
BMonthBegin or BusinessMonthBegin	'BMS'	business month begin
CBMonthEnd or CustomBusinessMonthEnd	'CBM'	custom business month end
CBMonthBegin or CustomBusinessMonthBegin	'CBMS'	custom business month begin
SemiMonthEnd	'SM'	15th (or other day_of_month) and calendar month end
SemiMonthBegin	'SMS'	15th (or other day_of_month) and calendar month begin
QuarterEnd	'Q'	calendar quarter end
QuarterBegin	'QS'	calendar quarter begin
BQuarterEnd	'BQ'	business quarter end
BQuarterBegin	'BQS'	business quarter begin
FY5253Quarter	'REQ'	retail (aka 52-53 week) quarter
YearEnd	'A'	calendar year end
YearBegin	'AS' or 'BYS'	calendar year begin
BYearEnd	'BA'	business year end
BYearBegin	'BAS'	business year begin
FY5253	'RE'	retail (aka 52-53 week) year
Easter	None	Easter holiday
BusinessHour	'BH'	business hour
CustomBusinessHour	'CBH'	custom business hour
Day	'D'	one absolute day
Hour	'H'	one hour
Minute	'T' or 'min'	one minute
Second	'S'	one second
Milli	'L' or 'ms'	one millisecond
Micro	'U' or 'us'	one microsecond
Nano	'N'	one nanosecond

[DateOffset](#)
Standard kind of date increment used for a date range.

Properties

- [DateOffset.freqstr](#)
- [DateOffset.kwds](#)
- [DateOffset.name](#)
- [DateOffset.nanos](#)
- [DateOffset.normalize](#)
- [DateOffset.rule_code](#)
- [DateOffset.n](#)
- [DateOffset.is_month_start](#)
- [DateOffset.is_month_end](#)

Methods

- [DateOffset.apply\(other\)](#)
- [DateOffset.apply_index\(other\)](#)
- [DateOffset.copy](#)
- [DateOffset.isAnchored](#)
- [DateOffset.onOffset](#)
- [DateOffset.is_anchored](#)
- [DateOffset.is_on_offset](#)
- [DateOffset.call\(*args,**kwargs\)](#)
- [DateOffset.is_month_start](#)
- [DateOffset.is_month_end](#)
- [DateOffset.is_quarter_start](#)
- [DateOffset.is_quarter_end](#)
- [DateOffset.is_year_start](#)
- [DateOffset.is_year_end](#)

AnchoredOffset Objects

[Anchored offsets](#)

For some frequencies you can specify an anchoring suffix:

Alias	Description
W-SUN	weekly frequency (Sundays). Same as 'W'
W-MON	weekly frequency (Mondays)
W-TUE	weekly frequency (Tuesdays)
W-WED	weekly frequency (Wednesdays)
W-THU	weekly frequency (Thursdays)
W-FRI	weekly frequency (Fridays)
W-SAT	weekly frequency (Saturdays)
(B)Q(S)-DEC	quarterly frequency, year ends in December. Same as 'Q'
(B)Q(S)-JAN	quarterly frequency, year ends in January
(B)Q(S)-FEB	quarterly frequency, year ends in February
(B)Q(S)-MAR	quarterly frequency, year ends in March
(B)Q(S)-APR	quarterly frequency, year ends in April
(B)Q(S)-MAY	quarterly frequency, year ends in May
(B)Q(S)-JUN	quarterly frequency, year ends in June
(B)Q(S)-JUL	quarterly frequency, year ends in July
(B)Q(S)-AUG	quarterly frequency, year ends in August
(B)Q(S)-SEP	quarterly frequency, year ends in September
(B)Q(S)-OCT	quarterly frequency, year ends in October
(B)Q(S)-NOV	quarterly frequency, year ends in November
(B)A(S)-DEC	annual frequency, anchored end of December. Same as 'A'
(B)A(S)-JAN	annual frequency, anchored end of January
(B)A(S)-FEB	annual frequency, anchored end of February
(B)A(S)-MAR	annual frequency, anchored end of March
(B)A(S)-APR	annual frequency, anchored end of April
(B)A(S)-MAY	annual frequency, anchored end of May
(B)A(S)-JUN	annual frequency, anchored end of June
(B)A(S)-JUL	annual frequency, anchored end of July
(B)A(S)-AUG	annual frequency, anchored end of August
(B)A(S)-SEP	annual frequency, anchored end of September
(B)A(S)-OCT	annual frequency, anchored end of October
(B)A(S)-NOV	annual frequency, anchored end of November

These can be used as arguments to `date_range`, `bdate_range`, constructors for `DatetimeIndex`, as well as various other timeseries-related functions in pandas.

[DateOffset](#)

Standard kind of date increment used for a date range.

[Properties](#)

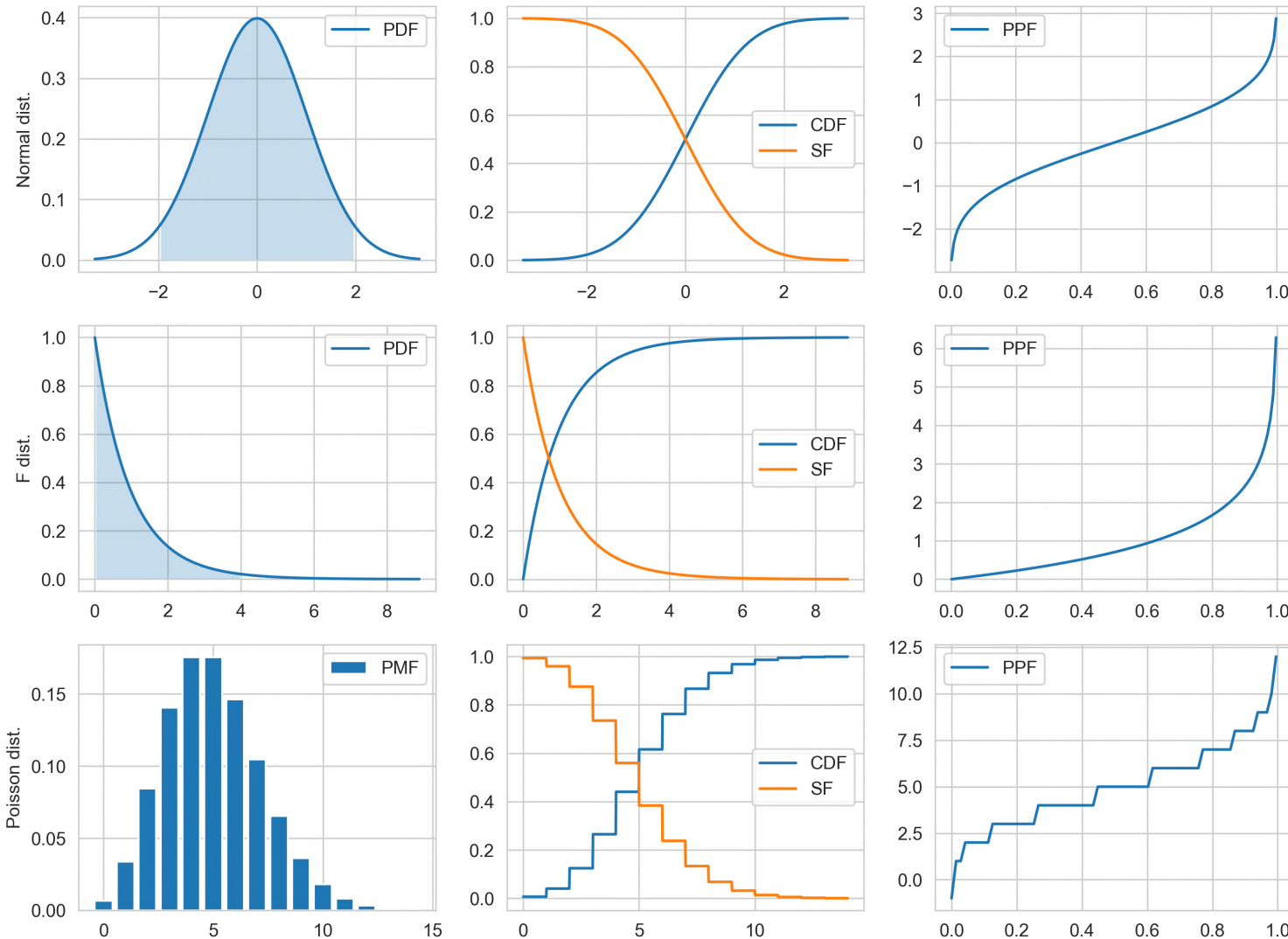
[DateOffset.freqstr](#)
[DateOffset.kwds](#)
[DateOffset.name](#)
[DateOffset.nanos](#)
[DateOffset.normalize](#)
[DateOffset.rule_code](#)
[DateOffset.n](#)
[DateOffset.is_month_start](#)
[DateOffset.is_month_end](#)

[Methods](#)

[DateOffset.apply\(other\)](#)
[DateOffset.apply_index\(other\)](#)
[DateOffset.copy](#)
[DateOffset.isAnchored](#)
[DateOffset.onOffset](#)
[DateOffset.is_anchored](#)
[DateOffset.is_on_offset](#)
[DateOffset.call\(*args,**kwargs\)](#)
[DateOffset.is_month_start](#)
[DateOffset.is_month_end](#)
[DateOffset.is_quarter_start](#)
[DateOffset.is_quarter_end](#)
[DateOffset.is_year_start](#)
[DateOffset.is_year_end](#)



STATISTICS WITH PYTHON



Examples of probability distribution functions (PDF) or probability mass functions (PMFs), cumulative distribution functions (CDF), survival functions (SF), and percent-point functions (PPF) for a normal distribution (top), an F distribution (middle), and a Poisson distribution (bottom)

```
from scipy import stats #Scipy.stats Official documentation
from scipy import optimize
```

```
import seaborn as sns
sns.set_style('whitegrid')
```

```
def plot_rv_distribution(X, axes=None):
    """Plot the PDF or PMF, CDF, SF and PPF of a given random variable"""
```

```
    if axes is None:
        fig, axes = plt.subplots(1, 3, figsize=(12, 3))
        x_min_999, x_max_999 = X.interval(0.999)
        x999 = np.linspace(x_min_999, x_max_999, 1000)
        x_min_95, x_max_95 = X.interval(0.95)
        x95 = np.linspace(x_min_95, x_max_95, 1000)
        if hasattr(X, "pdf"):
            axes[0].plot(x999, X.pdf(x999), label="PDF")
            axes[0].fill_between(x95, X.pdf(x95), alpha=0.25)
        else:
            # discrete random variables do not have a pdf method, instead we use pmf:
            x999_int = np.unique(x999.astype(int))
            axes[0].bar(x999_int, X.pmf(x999_int), label="PMF")
        axes[1].plot(x999, X.cdf(x999), label="CDF")
        axes[1].plot(x999, X.sf(x999), label="SF")
        axes[2].plot(x999, X.ppf(x999), label="PPF")
        for ax in axes:
            ax.legend()
```

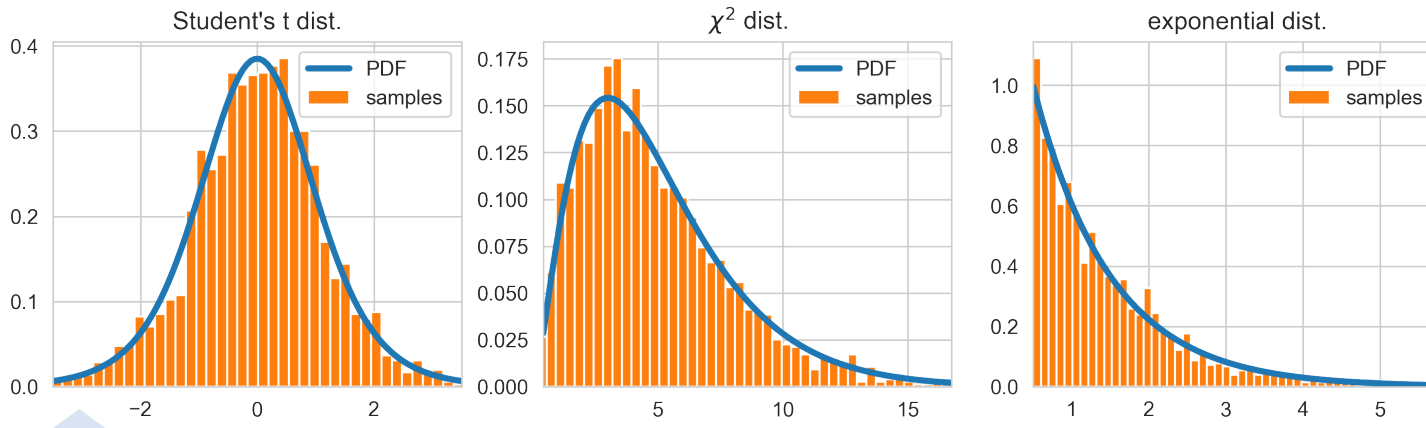
```
fig, axes = plt.subplots(3, 3, figsize=(12, 9))
X = stats.norm() # norm object, an instance of the rv_continuous class
plot_rv_distribution(X, axes=axes[0, :])
axes[0, 0].set_ylabel("Normal dist.")
```

```
X = stats.f(2, 50) # f object, an instance of the rv_continuous class
plot_rv_distribution(X, axes=axes[1, :])
axes[1, 0].set_ylabel("F dist.")
```

```
X = stats.poisson(5) # poisson object, an instance of the rv_discrete class
plot_rv_distribution(X, axes=axes[2, :])
axes[2, 0].set_ylabel("Poisson dist.")
```

```
fig.savefig('Statistical_Distributions.png', dpi = 300)
```





Probability distribution function (PDF) together with histograms of 2000 random samples from the Student's t distribution (left), the χ^2 distribution (middle), and the exponential distribution (right)

The SciPy stats module provides classes for representing random variables with a large number of probability distributions. There are two base classes for discrete and continuous random variables: [rv_discrete](#) and [rv_continuous](#). These classes are not used directly, but rather used as base classes for random variables with specific distributions, and define a common interface for all random variable classes in SciPy stats.

Discrete Distribution

- Binomial distribution
- Poisson distribution
- Hypergeometric distribution

Continuous Distribution

- Normal distribution
- Student's t distribution
- Chi squared distribution
- F distribution
- Uniform distribution
- Exponential Distribution
- Log normal Distribution

Resources :

- Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib by Robert Johansson
- A First Course in Statistical Inference by Jonathan Gillard
- Business statistics Communicating with numbers by Jaggia/Kelly
- Python for Probability, Statistics, and Machine Learning by Jose Unpingco
- An Introduction to Statistics with Python With Applications in the Life Sciences by Thomas Haslwanter
- Practical Statistics for Data Scientists by Peter Bruce, Andrew Bruce & Peter Gedeck

Source of Plot/code : Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib by Robert Johansson

```
def plot_dist_samples(X, X_samples, title=None, ax=None):
    """ Plot the PDF and histogram of samples of a continuous random
    variable """
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=(8, 4))
        # Setting xlim to Confidence interval with equal areas around the median
        x_lim = X.interval(.99)
        x = np.linspace(*x_lim, num=100)
        ax.plot(x, X.pdf(x), label="PDF", lw=3)
        ax.hist(X_samples, label="samples", density = True, bins=75)
        ax.set_xlim(*x_lim)
        ax.legend()
        if title: ax.set_title(title)

    return ax

fig, axes = plt.subplots(1, 3, figsize=(12, 3))
N = 2000

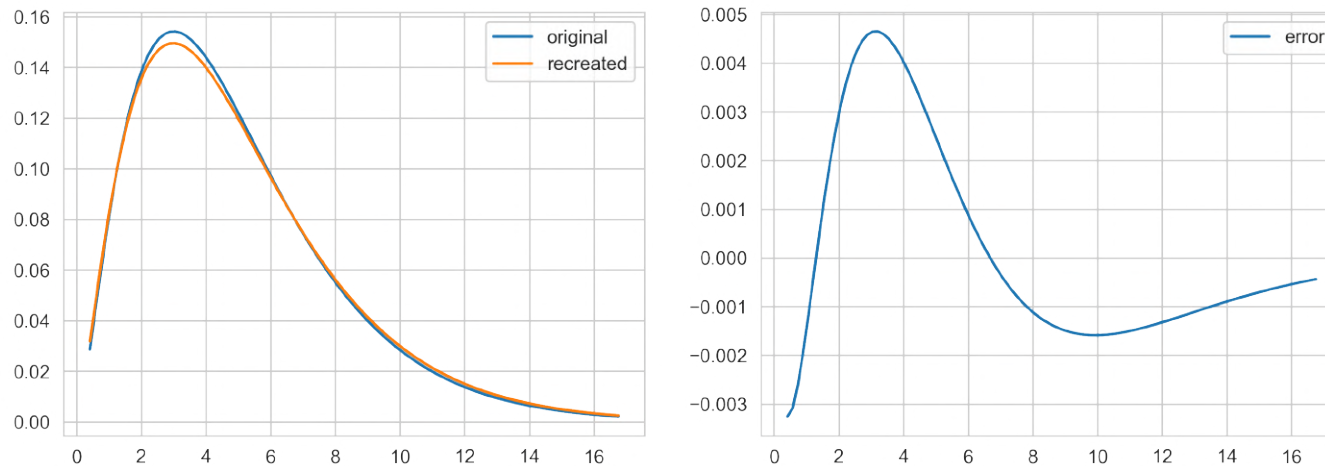
# Student's t distribution
X = stats.t(7.0) # t object, an instance of the rv_continuous class
plot_dist_samples(X, X.rvs(N), "Student's t dist.", ax=axes[0])

# The chisquared distribution
X = stats.chi2(5.0) # chi object, an instance of the rv_continuous class
plot_dist_samples(X, X.rvs(N), r"$\chi^2$ dist.", ax=axes[1])

# The exponential distribution
X = stats.expon(0.5) # expon object, an instance of the rv_continuous class
plot_dist_samples(X, X.rvs(N), "exponential dist.", ax=axes[2])

fig.savefig('Statistical Distributions.png', dpi = 300, transparent = True)
```





Original and recreated probability distribution function (left) and the error (right), from a maximum likelihood fit of 500 random samples of the original distribution

Maximum Likelihood Estimation – A method of estimation of one or more parameters of a population by maximizing the likelihood or log-likelihood function of the sample with respect to the parameter(s). The maximum likelihood estimators are functions of the sample observations that make the likelihood function greatest.

Discrete Distribution

- Binomial distribution
- Poisson distribution
- Hypergeometric distribution

Continuous Distribution

- Normal distribution
- Student's t distribution
- Chi squared distribution
- F distribution
- Uniform distribution
- Exponential Distribution
- Log normal Distribution

Resources :

- Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib by Robert Johansson
- A First Course in Statistical Inference by Jonathan Gillard
- Business statistics Communicating with numbers by Jaggia/Kelly
- Python for Probability, Statistics, and Machine Learning by Jose Unpingco
- An Introduction to Statistics with Python With Applications in the Life Sciences by Thomas Haslwanter

Source of Plot/code : Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib by Robert Johansson

```
import matplotlib.pyplot as plt
from scipy import stats
```

```
# Instantiating a chi-squared distribution with degrees of freedom = 5. Link
```

```
X = stats.chi2(df=5)
```

```
# Generating rvs of chi distribution
```

```
X_samples = X.rvs(500)
```

```
#Return estimates of shape, location, and scale parameters from data using  
Maximum Likelihood Estimation Method. Link
```

```
df, loc, scale = stats.chi2.fit(X_samples)
```

```
Y = stats.chi2(df=df, loc=loc, scale=scale)
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
```

```
# Setting xlim to Confidence interval with equal areas around the median
```

```
x_lim = X.interval(.99) #
```

```
x = np.linspace(*x_lim, num=100)
```

```
axes[0].plot(x, X.pdf(x), label="original")
```

```
axes[0].plot(x, Y.pdf(x), label="recreated")
```

```
axes[0].legend()
```

```
axes[1].plot(x, X.pdf(x) - Y.pdf(x), label="error")
```

```
axes[1].legend()
```

```
fig.savefig('MaximumLikelihoodFitting.png', dpi = 300, transparent = True)
```



Selected Methods for Discrete and Continuous Random Variables in the SciPy stats Module		
Methods	Description	
pdf/pmf	probability distribution function (continuous) or probability mass function (discrete).	
cdf	Cumulative distribution function.	
sf	Survival function (1 – cdf).	
ppf	percent-point function (inverse of cdf).	
moment	Noncentral moments of nth order.	
stats	Statistics of the distribution (typically the mean and variance, sometimes additional statistics).	
fit	Fit distribution to data using a numerical maximum likelihood optimization (for continuous distributions).	
expect	expectation value of a function with respect to the distribution.	
interval	the endpoints of the interval that contains a given percentage of the distribution (confidence interval)	
rvs	random variable samples. takes as argument the size of the resulting array of samples.	
mean, median, std, var	Descriptive statistics: mean, median, standard deviation, and the variance of the distribution.	
Summary of Common Hypothesis Test Cases with the Corresponding Distributions and SciPy Functions		
Null Hypothesis	Distributions	SciPy Functions for Test
test if the mean of a population is a given value.	Normal distribution (stats. norm), or Student’s t distribution (stats.t)	stats. ttest_1samp
test if the means of two random variables are equal (independent or paired samples).	Student’s t distribution (stats.t)	stats.ttest_ind, stats.ttest_re
test goodness of fit of a continuous distribution to data.	Kolmogorov-Smirnov distribution	stats.kstest
test if categorical data occur with given frequency (sum of squared normally distributed variables).	χ^2 distribution (stats.chi2)	stats.chisquare
test for the independence of categorical variables in a contingency table.	χ^2 distribution (stats.chi2)	stats.chi2_ contingency
test for equal variance in samples of two or more variables.	F distribution (stats.f)	stats.barlett, stats.levene
test for noncorrelation between two variables.	Beta distribution (stats.beta,	stats.pearsonr, stats.spearmanr
test if two or more variables have the same population mean (aNOVa – analysis of variance) .	stats.mstats.betainc) F distribution	stats.f_oneway, stats.krskal

Statistical functions ([scipy.stats](#))

[Probability distributions](#)

- [Continuous distributions](#)
- [Multivariate distributions](#)
- [Discrete distributions](#)

[Summary statistics](#)

[Frequency statistics](#)

[Correlation functions](#)

[Statistical tests](#)

[Quasi-Monte Carlo](#)

[Masked statistics functions](#)

[Other statistical functionality](#)

- [Transformations](#)
- [Statistical distances](#)

[Random variate generation / CDF Inversion](#)

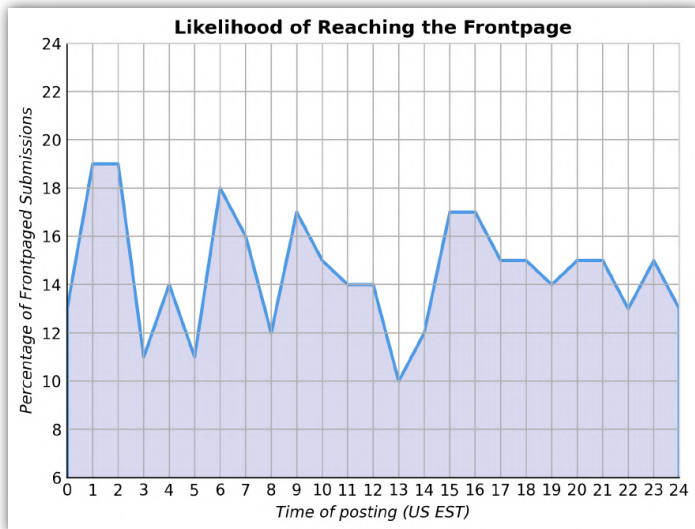
- [Circular statistical functions](#)
- [Contingency table functions](#)
- [Plot-tests](#)
- [Univariate and multivariate kernel density estimation](#)
- [Warnings used in scipy.stats](#)

APPENDIX

All Code examples have been checked with Python Version 3.8.3 and IDE JupyterLab
The package details are as below :

Package Name	pandas	numpy	scipy
Version	1.2.1	1.19.5	1.6.0
Summary	Powerful data structures for data analysis, time series, and statistics	NumPy is the fundamental package for array computing with Python.	SciPy: Scientific Library for Python

* Use `pip show <package_name>` to get details of a package



#<https://stackoverflow.com/questions/24547047/how-to-make-matplotlib-graphs-look-professionally-done-like-this>

Check out the answer by [DrV](#)

Necessary imports

```
import numpy as np
import matplotlib.pyplot as plt
```

```
plt.rcParamsdefaults()
```

create some fictive access data by hour

```
xdata = np.arange(25)
ydata = np.random.randint(10, 20, 25)
ydata[24] = ydata[0]
```

let us make a simple graph

```
fig = plt.figure(figsize=[7,5])
ax = plt.subplot(111)
l = ax.fill_between(xdata, ydata)
```

set the basic properties

```
ax.set_xlabel('Time of posting (US EST)')
ax.set_ylabel('Percentage of Frontpaged Submissions')
ax.set_title('Likelihood of Reaching the Frontpage')
```

set the limits

```
ax.set_xlim(0, 24)
ax.set_ylim(6, 24)
```

set the grid on

```
ax.grid('on')
```

change the fill into a blueish color with opacity .3

```
l.set_facecolors([[.5,.5,.8,.3]])
```

change the edge color (bluish and transparentish) and thickness

```
l.set_edgecolors([[0.3, 0.6, .9, 1]])
l.set_linewidths([2])
```

add more ticks

```
ax.set_xticks(np.arange(25))
```

remove tick marks

```
ax.xaxis.set_tick_params(size=0)
ax.yaxis.set_tick_params(size=0)
```

change the color of the top and right spines to opaque gray

```
ax.spines['right'].set_color((.8,.8,.8))
ax.spines['top'].set_color((.8,.8,.8))
```

tweak the axis labels

```
xlab = ax.xaxis.get_label()
ylab = ax.yaxis.get_label()
```

```
xlab.set_style('italic')
```

```
xlab.set_size(10)
```

```
ylab.set_style('italic')
```

```
ylab.set_size(10)
```

tweak the title

```
t1 = ax.title
```

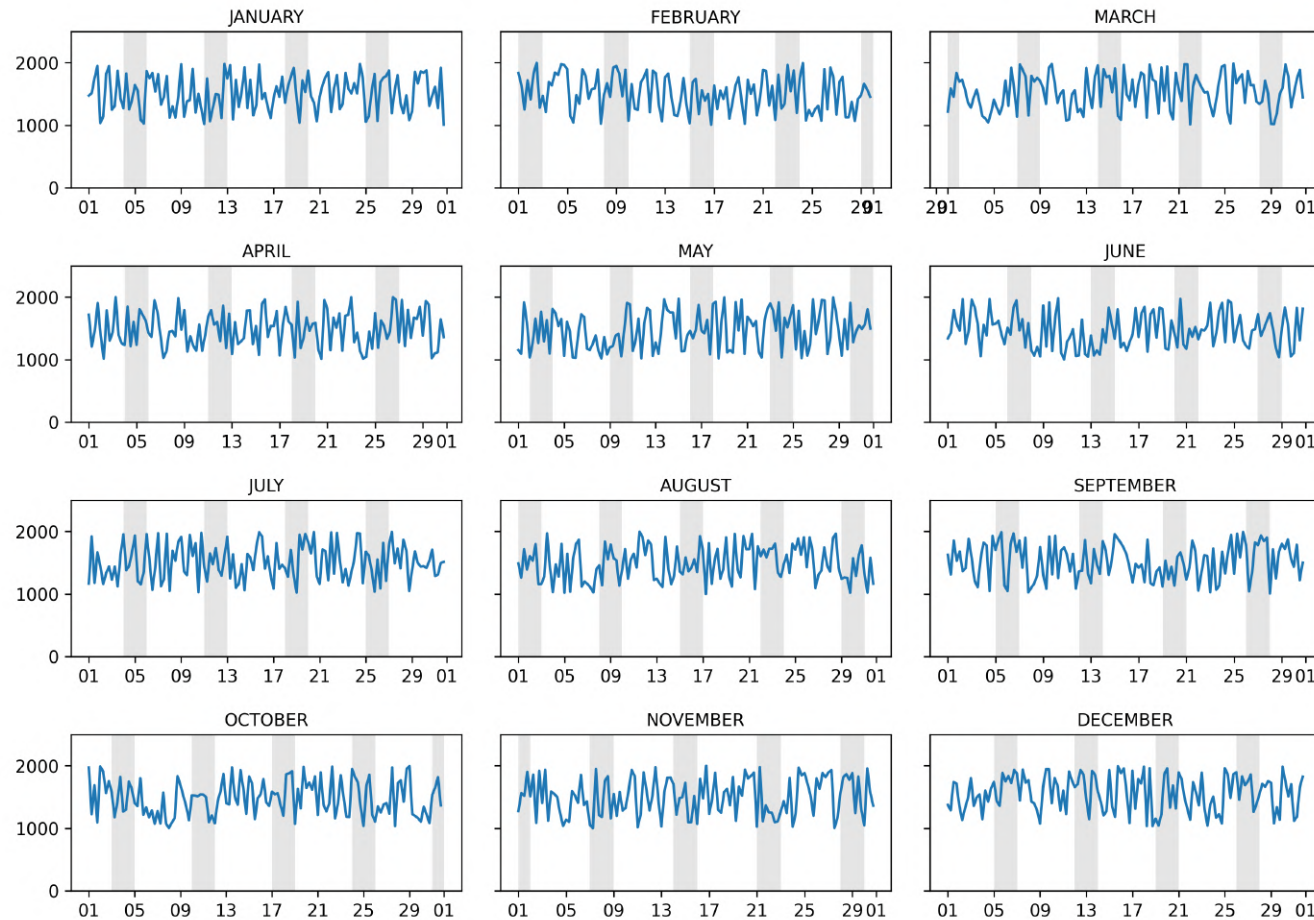
```
t1.set_weight('bold')
```

```
fig.savefig('Filline plot.png',dpi=300, format='png',
bbox_inches='tight')
```

Weekends are highlighted by using the DatetimeIndex

[Code](#)

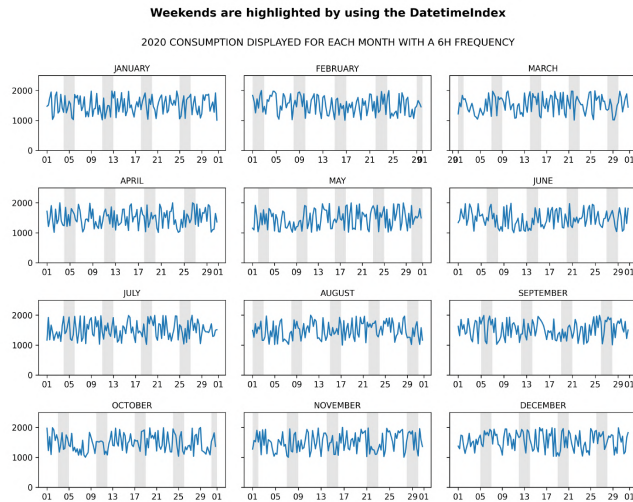
2020 CONSUMPTION DISPLAYED FOR EACH MONTH WITH A 6H FREQUENCY



#<https://stackoverflow.com/questions/66011487/highlighting-weekends-in-small-multiples>
Check the answer by [Patrick Fitzgerald](#)

CONTEXT

- Raw data is time Series with Daily frequency
- Figure with multiple subplots with each one for a particular month (iteration used)
- Weekends need to be highlighted



#<https://stackoverflow.com/questions/66011487/highlighting-weekends-in-small-multiples>
 # Check the answer by [Patrick Fitzgerald](#)

CONTEXT

- Raw data is time Series with Daily frequency
- Figure with multiple subplots with each one for a particular month (iteration used)
- Weekends need to be highlighted

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates # used only for method 2
```

```
plt.rcParamsdefaults()
```

```
# Create sample dataset
# random number generator
rng = np.random.default_rng(seed=1)
```

```
# Create a date time index
dti = pd.date_range('2020-01-01 00:00', '2020-12-31 \
23:59', freq='6H')
consumption = rng.integers(1000, 2000, size=dti.size)
```

```
# Create the data frame
df = pd.DataFrame(dict(consumption=consumption),
index=dti)
```

```
# Draw and format subplots by looping through months
and flattened array of axes
fig, axs = plt.subplots(4, 3, figsize=(13, 9), sharey=True)
```

```
for month, ax in zip(df.index.month.unique(), axs.flat):
    # Select monthly data and plot it
    df_month = df[df.index.month == month]
    ax.plot(df_month.index, df_month['consumption'])

    # set limit similar to plot shown in question
    ax.set_ylim(0, 2500)
```

```
# Draw vertical spans for weekends: computing the time
delta and adding it to the date solves the problem of
exceeding the df_month.index
```

```
timedelta = pd.to_timedelta(df_month.index.freq)
weekends = df_month.index[df_month.index.weekday
>=5].to_series()

for date in weekends:
    ax.axvspan(date, date+timedelta, facecolor='k',
edgecolor=None, alpha=.1)

# Format tick labels
ax.set_xticks(ax.get_xticks())
tk_labels = [pd.to_datetime(tk, unit='D').strftime('%d')
for tk in ax.get_xticks()]
ax.set_xticklabels(tk_labels, rotation=0, ha='center')
```

```
# Add x labels for months
ax.set_xlabel(df_month.index[0].month_name().upper(),
labelpad=5)
ax.xaxis.set_label_position('top')
```

```
# Add title and edit spaces between subplots
year = df.index[0].year
freq = df_month.index.freqstr
```

```
title = f'{year} consumption displayed for each month with
a {freq} frequency'
```

```
fig.suptitle(title.upper(), y=0.95, fontsize=12)
fig.subplots_adjust(wspace=0.1, hspace=0.5)
```

```
fig.text(0.5, 0.99, 'Weekends are highlighted by using \
the DatetimeIndex',
ha='center', fontsize=14, weight='semibold')
```

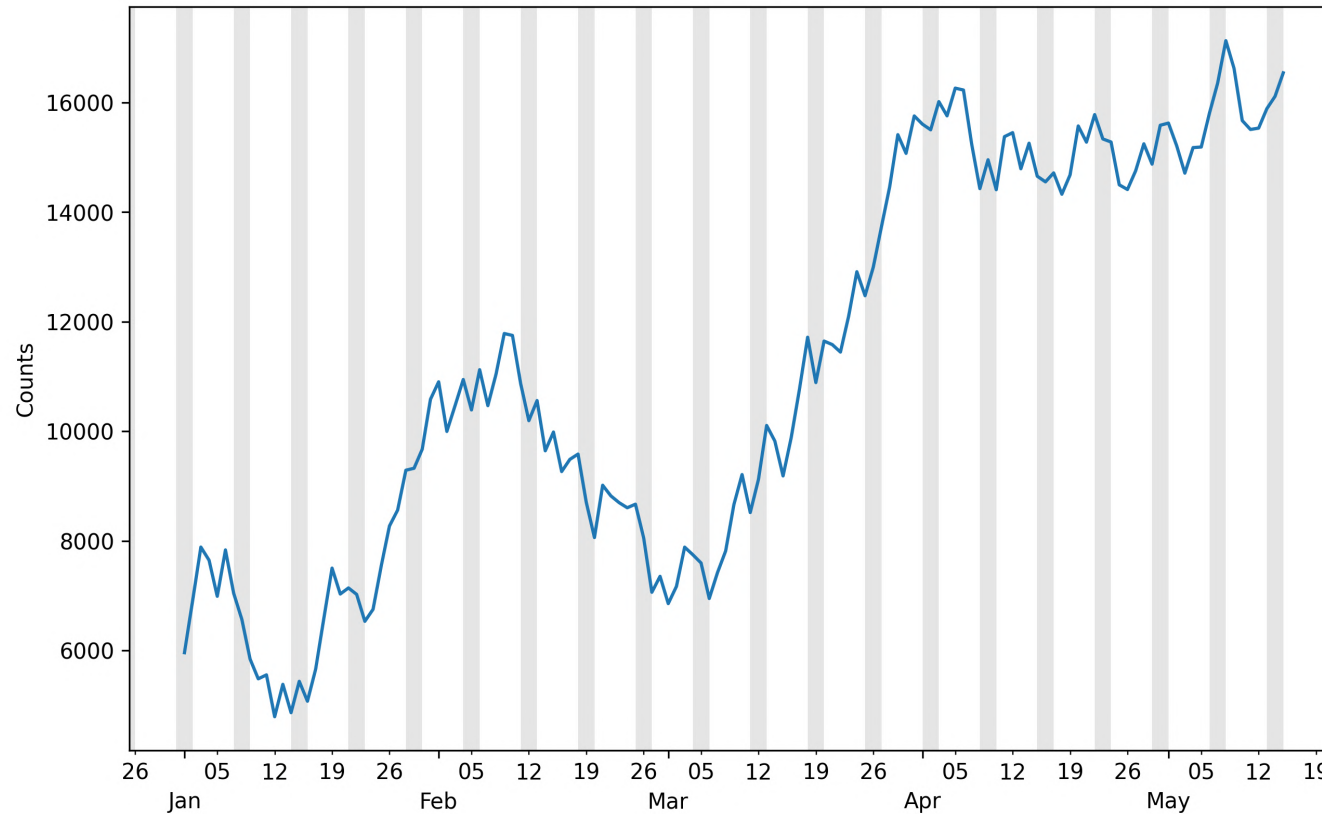
```
plt.savefig('Daily consumption with weekends \
highlighted.png', bbox_inches = "tight", pad_inches =
0.5, dpi=300)
plt.show()
```

Also, Check below excellent links on using spans and fill areas

- <https://stackoverflow.com/questions/8270981/in-a-matplotlib-plot-can-i-highlight-specific-x-value-ranges?noredirect=1&lq=1>
- <https://stackoverflow.com/questions/48973471/how-to-highlight-weekends-for-time-series-line-plot-in-python/66052245#66052245>
- <https://stackoverflow.com/questions/64356412/highlight-time-interval-in-multivariate-time-series-plot-using-matplotlib-and-se?rq=1>

Code

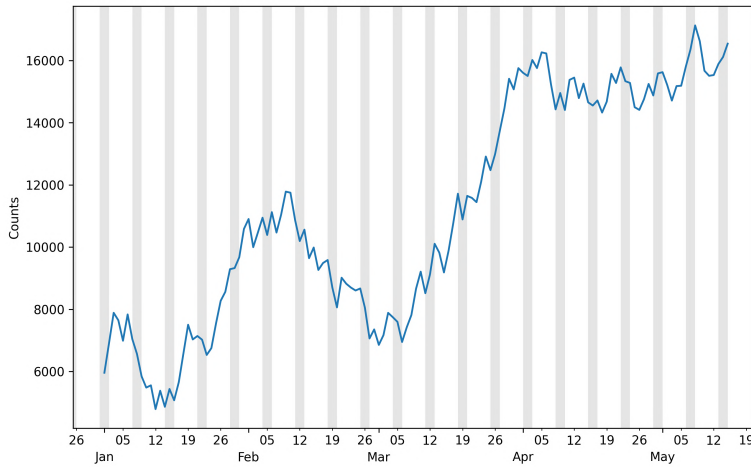
Daily count of trips with weekends highlighted from SAT 00:00 to MON 00:00



[#https://stackoverflow.com/questions/48973471/how-to-highlight-weekends-for-time-series-line-plot-in-python/66052245#66052245](https://stackoverflow.com/questions/48973471/how-to-highlight-weekends-for-time-series-line-plot-in-python/66052245#66052245)
Check the answer by [Patrick Fitzgerald](#)

Highlighting Weekends in a Time Series

Daily count of trips with weekends highlighted from SAT 00:00 to MON 00:00



<https://stackoverflow.com/questions/48973471/how-to-highlight-weekends-for-time-series-line-plot-in-python/66052245#66052245>

Check the answer by [Patrick Fitzgerald](#)

#Highlighting Weekends on a time series plot

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

Create sample dataset

random number generator

```
rng = np.random.default_rng(seed=1234)
```

```
dti = pd.date_range('2017-01-01', '2017-05-15', freq='D')
```

```
counts = 5000 + np.cumsum(rng.integers(-1000, 1000,
                                         size=dti.size))
```

```
df = pd.DataFrame(dict(Counts=counts), index=dti)
```

```
# Draw pandas plot: x_compat=True converts the pandas x-axis units to matplotlib
# date units (not strictly necessary when using a daily frequency like here)
```

```
ax = df.plot(x_compat=True, figsize=(10, 7), legend = None, ylabel='Counts')
```

```
# reset y limits to display highlights without gaps
```

```
ax.set_ylim(*ax.get_ylim())
```

```
# Highlight weekends based on the x-axis units
```

```
xmin, xmax = ax.get_xlim()
```

```
days = np.arange(np.floor(xmin), np.ceil(xmax)+2)
```

```
weekends = [(dt.weekday()>=5)|(dt.weekday()==0) for dt in mdates.num2date(days)]
```

```
ax.fill_between(days, *ax.get_ylim(), where=weekends, facecolor='k', alpha=.1)
```

```
ax.set_xlim(xmin, xmax) # set limits back to default values
```

```
# Create appropriate ticks using matplotlib date tick locators and formatters
```

```
ax.xaxis.set_major_locator(mdates.MonthLocator())
```

```
ax.xaxis.set_minor_locator(mdates.MonthLocator(bymonthday=np.arange(5, 31, step=7)))
```

```
ax.xaxis.set_major_formatter(mdates.DateFormatter("\n%b"))
```

```
ax.xaxis.set_minor_formatter(mdates.DateFormatter('%d'))
```

```
# Additional formatting
```

```
ax.figure.autofmt_xdate(rotation=0, ha='center')
```

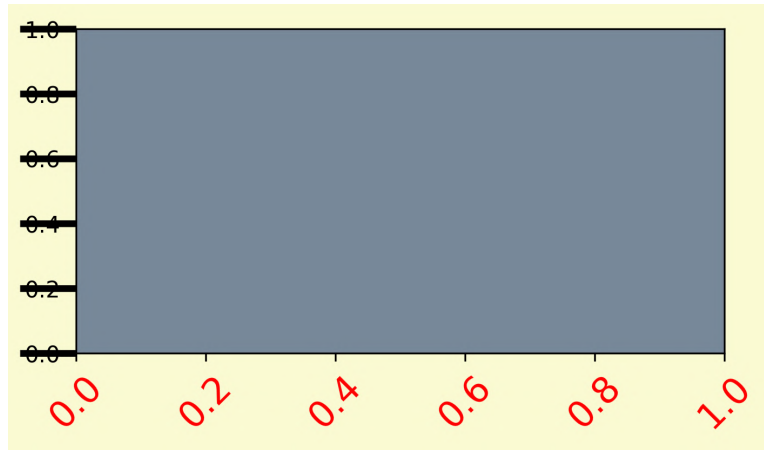
```
title = 'Daily count of trips with weekends highlighted from SAT 00:00 to MON 00:00'
```

```
ax.set_title(title, pad=20, fontsize=14)
```

```
plt.savefig('Daily count of trips with weekends highlighted.png',bbox_inches = "tight",pad_inches = 0.5, dpi=300)
```

```
plt.show()
```

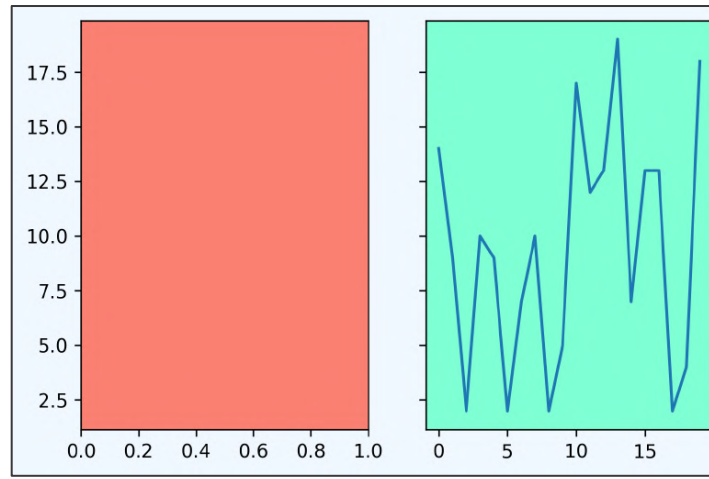
Highlighting Weekends in a Time Series



```
#Matplotlib Release, Page no.120-121
#plt.figure creates a matplotlib.figure.Figure instance
fig = plt.figure(figsize = (10,5))
rect = fig.patch # a rectangle instance
rect.set_facecolor('lightgoldenrodyellow')

ax1 = fig.add_axes([0.1, 0.3, 0.4, 0.4])
rect = ax1.patch
rect.set_facecolor('lightslategray')

for label in ax1.xaxis.get_ticklabels():
    # label is a Text instance
    label.set_color('red')
    label.set_rotation(45)
    label.set_fontsize(16)
for line in ax1.yaxis.get_ticklines():
    # line is a Line2D instance
    line.set_color('green')
    line.set_markersize(25)
    line.set_markeredgewidth(3)
fig.savefig('tick_labels and lines.png',dpi=300,
           format='png', bbox_inches='tight')
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np

fig, (ax1, ax2) = plt.subplots(1,2, sharey = True)

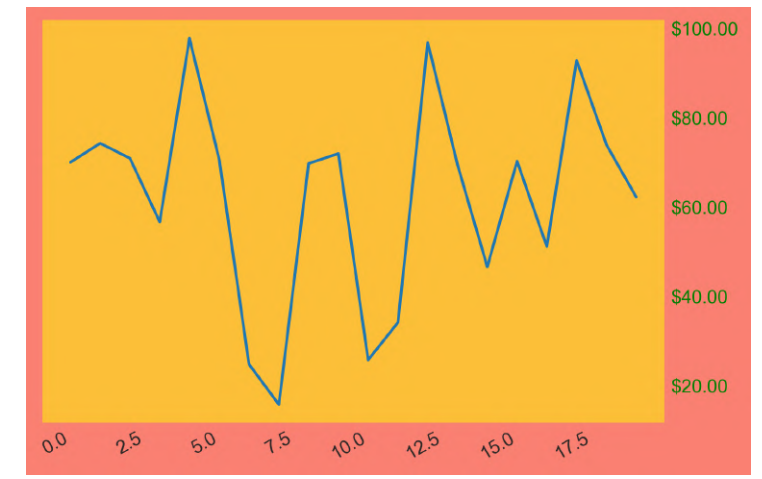
x = np.random.randint(1,20,20)

#Set the face color of figure background
rect = fig.patch
rect.set(facecolor = 'aliceblue')

#Set the face color of the axes backgrounds
ax1.patch.set(facecolor = 'salmon')
ax2.patch.set(facecolor = 'aquamarine')
ax2.plot(x)

plt.show()
```

Appearance of Plot
Fig, Axes background
Tick labels, Tick lines

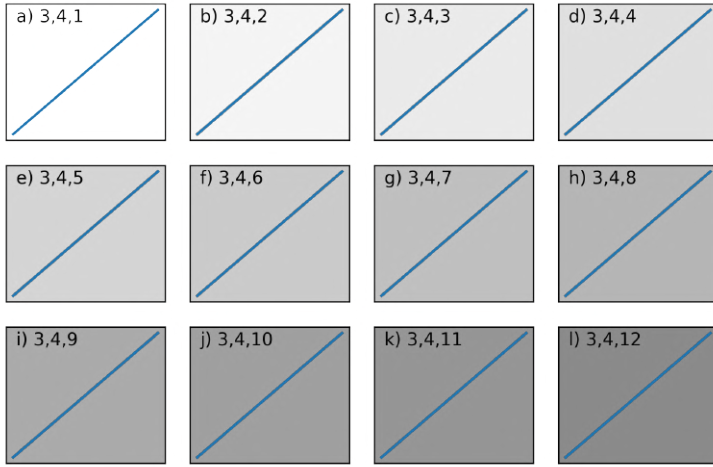


```
#Matplotlib Release, Page no.120-121
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
fig, ax = plt.subplots()
ax.plot(100*np.random.rand(20))

ax.patch.set(facecolor="yellow",alpha =0.5)
ax.patch.set_facecolor("yellow")
ax.patch.set_alpha(0.5)
fig.set(facecolor = "salmon")

# Use automatic StrMethodFormatter
ax.yaxis.set_major_formatter('${x:1.2f}')
ax.yaxis.set_tick_params(which='major', labelcolor='green',
                        labelleft=False, labelright=True)

#ax.set_xticks(ax.get_xticks())
#ax.set_xticklabels(ax.get_xticklabels(), rotation = 30)
plt.setp(ax.get_xticklabels(), rotation=30, ha='right')
plt.show()
```



#<https://stackoverflow.com/questions/3584805/in-matplotlib-what-does-the-argument-mean-in-fig-add-subplot111?rq=1>

Check the answer by [compuphys](#)

The link lucidly discusses the multiple call signatures of fig.add_subplot method.

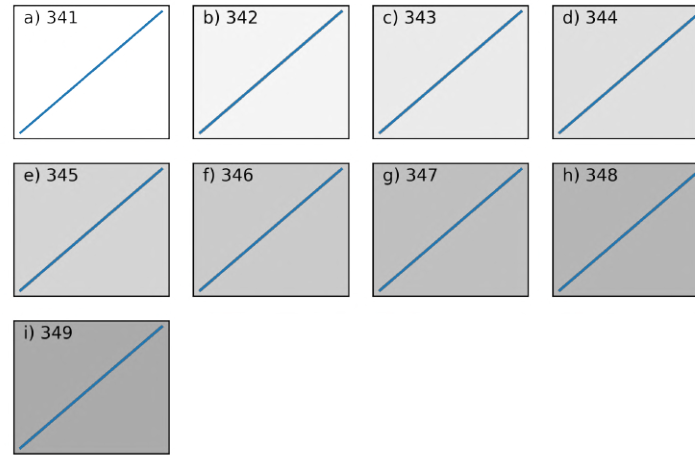
The add_subplot() method has several call signatures:

- add_subplot(nrows, ncols, index, **kwargs)
- add_subplot(pos, **kwargs)
- add_subplot(ax)
- add_subplot()

The following code demonstrates the difference between the first two call signatures.

- add_subplot(nrows, ncols, index, **kwargs)
- add_subplot(pos, **kwargs)

The first call signature can accommodate any number of rows, columns. Whereas the second call signature is limited to total 9 subplots



pos is a three digit integer, where the first digit is the number of rows, the second the number of columns, and the third the index of the subplot. i.e. fig.add_subplot(235) is the same as fig.add_subplot(2, 3, 5).

Also, Check the documentation :

https://matplotlib.org/stable/api/as_gen/matplotlib.pyplot.subplot.html

Making multiple subplots fig.add_subplot() Method

```
import matplotlib.pyplot as plt
```

```
def plot_and_text(axis, text):
```

```
    """Simple function to add a straight line  
    and text to an axis object"""
```

```
    axis.plot([0,1],[0,1])  
    axis.text(0.02, 0.9, text)
```

```
f = plt.figure()  
f2 = plt.figure()
```

```
_max = 12  
for i in range(_max):  
    axis = f.add_subplot(3,4,i+1, fc=(0,0,0,i/(_max*2)), xticks=[],  
yticks=[])  
    plot_and_text(axis,chr(i+97) + ' ' + '3,4,' +str(i+1))
```

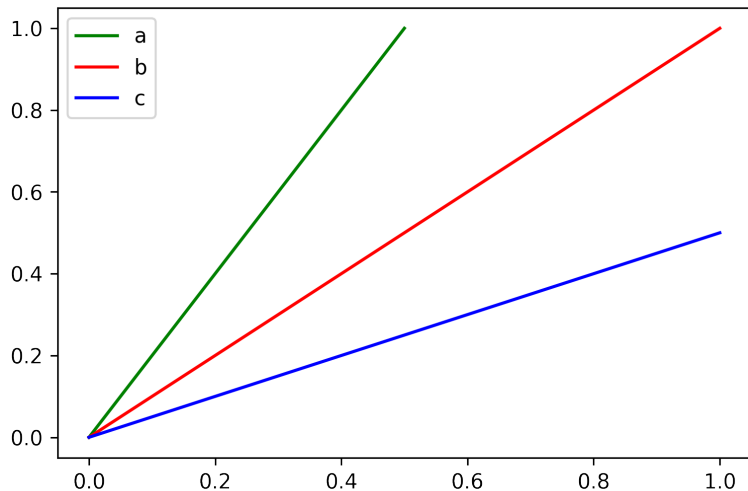
```
# If this check isn't in place, a  
# ValueError: num must be 1 <= num <= 15, not 0 is raised  
if i < 9:  
    axis = f2.add_subplot(341+i, fc=(0,0,0,i/(_max*2)), \  
xticks=[], yticks=[])  
    plot_and_text(axis,chr(i+97) + ' ' + str(341+i))
```

```
f.tight_layout()  
f2.tight_layout()
```

```
f._label = 'fig1'  
f2._label = 'fig2'
```

```
for fig in [f, f2]:  
    print(f'{fig._label}')  
    fig.savefig(f'Adding Subplots_{fig._label}.png',  
dpi = 300,format='png', bbox_inches='tight')  
plt.show()
```

Automatic Detection Legend



#Legend Call Signature : ax.legend()
#All the lines have been assigned valid text labels

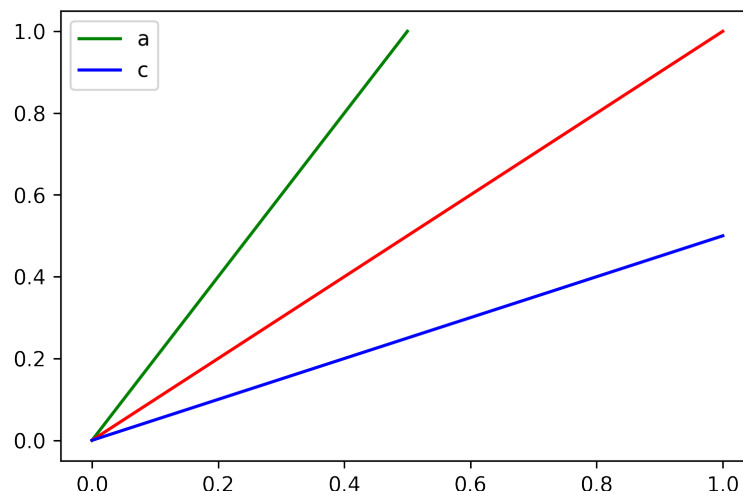
```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
lgh1, = ax.plot([0, 0.5], [0, 1], '-g', label="a")
lgh2, = ax.plot([0, 1], [0, 1], '-r', label="b")
lgh3, = ax.plot([0, 1], [0, 0.5], '-b', label="c")
```

```
#plt.legend() #This is equivalent to ax.legend()
ax.legend()
plt.show()
```

#<https://stackoverflow.com/questions/54390421/matplotlib-b-legend-not-working-correctly-with-handles>

Automatic Detection Legend



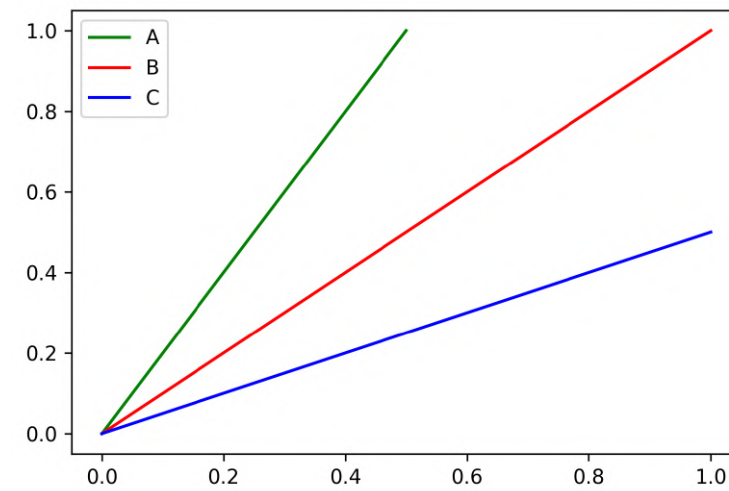
#Legend Call Signature : ax.legend()
#Except the red line, the other lines have been assigned text labels

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
lgh1, = ax.plot([0, 0.5], [0, 1], '-g', label="a")
lgh2, = ax.plot([0, 1], [0, 1], '-r')
lgh3, = ax.plot([0, 1], [0, 0.5], '-b', label="c")
```

```
#plt.legend() #This is equivalent to ax.legend()
ax.legend()
plt.show()
```

Explicitly Define Legend Elements



#Legend Call Signature : ax.legend(handles, labels)
#Explicitly Passing the handles and corresponding labels

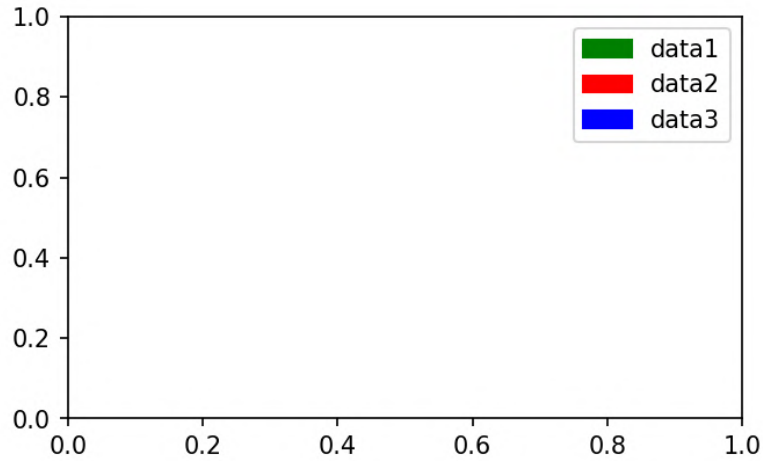
```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
lgh1, = ax.plot([0, 0.5], [0, 1], '-g', label="a")
lgh2, = ax.plot([0, 1], [0, 1], '-r')
lgh3, = ax.plot([0, 1], [0, 0.5], '-b', label="c")
```

```
handles = [lgh1, lgh2, lgh3]
labels = ['A', 'B', 'C']
```

```
#plt.legend(handles = handles, labels = labels)
ax.legend(handles = handles, labels = labels)
plt.show()
```

Proxy Artists : Manual Patch Addition



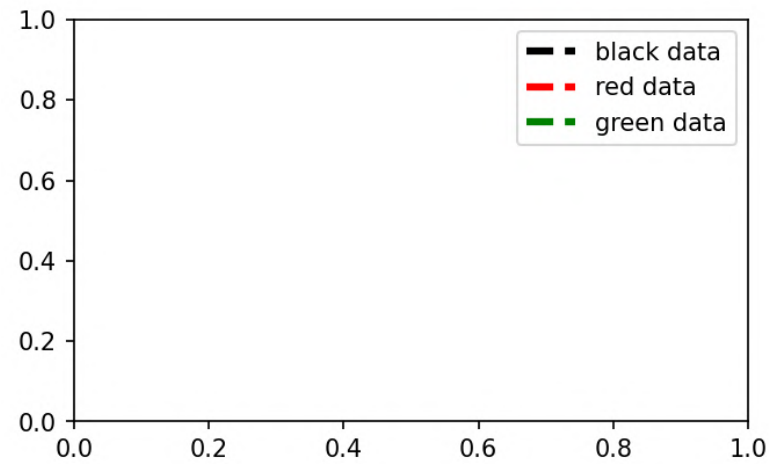
#Legend Call Signature : `ax.legend(handles)`
 #The patches are not in the axes but specifically added for legend
 #Secondly, we have used a dictionary of labels and colors
[#https://stackoverflow.com/questions/39500265/manually-add-legend-items-python-matplotlib](https://stackoverflow.com/questions/39500265/manually-add-legend-items-python-matplotlib)

```
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
legend_dict = { 'data1' : 'green', 'data2' : 'red',
                'data3' : 'blue' }
patchList = []
for key in legend_dict:
    data_key = mpatches.Patch(color=legend_dict[key],
                              label=key)
    patchList.append(data_key)

ax.legend(handles=patchList)
fig.savefig('legend_from dict.png',bbox_inches='tight')
```

Proxy Artists : Manual Lines Addition



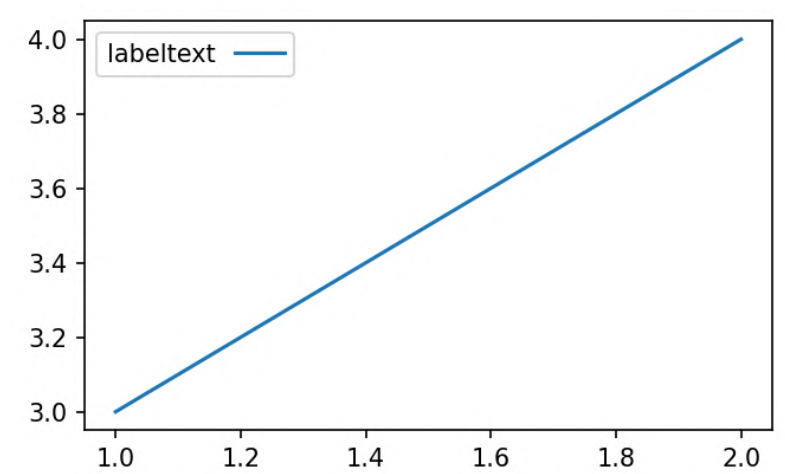
#Legend Call Signature : `ax.legend()`
 #Except the red line, the other lines have been assigned text labels
[#https://stackoverflow.com/questions/39500265/manually-add-legend-items-python-matplotlib](https://stackoverflow.com/questions/39500265/manually-add-legend-items-python-matplotlib)
`import matplotlib.pyplot as plt`
`from matplotlib.lines import Line2D`

`fig, ax = plt.subplots(figsize=(5,3))`
`colors = ['black', 'red', 'green']`

 #Instantiating proxy line artists
`lines = [Line2D([0], [0], color=c, linewidth=3, linestyle='--')`
 `for c in colors]`
`labels = ['black data', 'red data', 'green data']`
`ax.legend(lines, labels)`

`fig.savefig('proxy artists_line 2D.png',dpi = 150,`
 `bbox_inches='tight')`

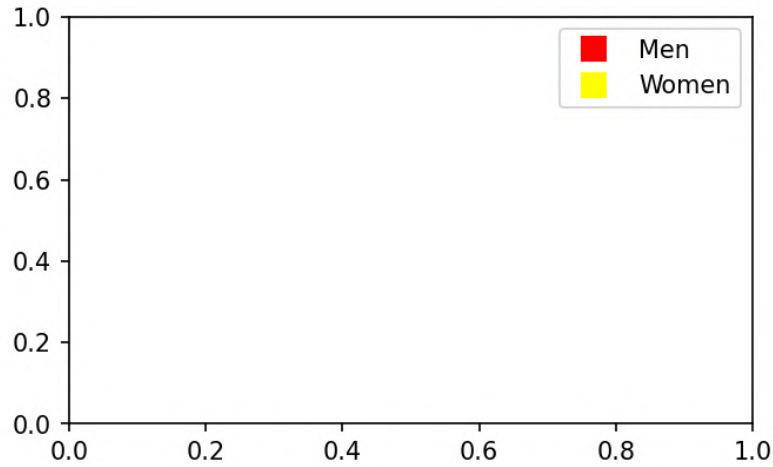
Labels Appearing before Legend key



[#https://stackoverflow.com/questions/24867363/matplotlib-b-put-legend-symbols-on-the-right-of-the-labels](https://stackoverflow.com/questions/24867363/matplotlib-b-put-legend-symbols-on-the-right-of-the-labels)

`fig, ax = plt.subplots(figsize = (5,3))`
`ax.plot([1,2],[3,4], label='labeltext')`
`ax.legend(markerfirst=False)`
`plt.gcf().savefig('markerfirst_legend.png',dpi = 150,`
 `bbox_inches='tight')`
`plt.show()`

Making Legend Keys square

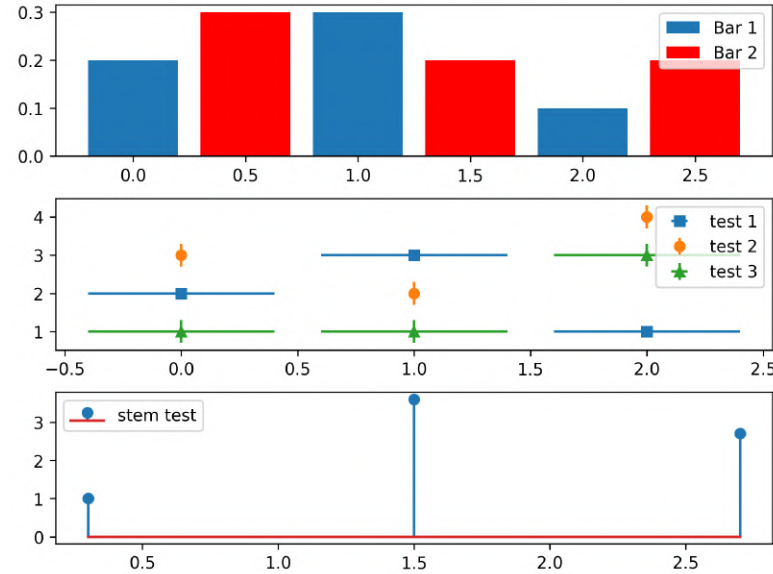


#Legend Call Signature : `ax.legend(handles)`
 #The patches are not in the axes but specifically added for legend
<https://stackoverflow.com/questions/27826064/matplotlib-make-legend-keys-square>
`import matplotlib.lines as mlines`

```
plt.rcParams()
fig, ax = plt.subplots(figsize = (5,3))
rect1 = mlines.Line2D([], [], marker="s", markersize=10,
                      linewidth=0, color="red")
rect2 = mlines.Line2D([], [], marker="s", markersize=10,
                      linewidth=0, color="yellow")

ax.legend((rect1, rect2), ('Men', 'Women'))
fig.savefig('Making square markers.png',dpi = 150,
           bbox_inches='tight')
plt.show()
```

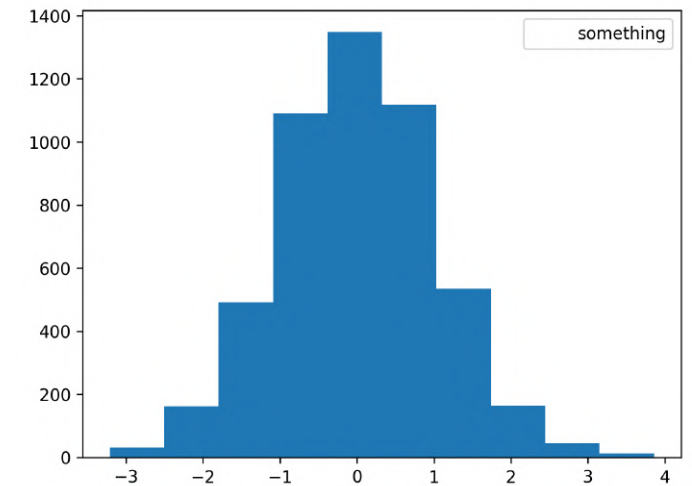
Legend Addition in Complex Plots



https://matplotlib.org/stable/gallery/text_labels_and_annotations/legend_demo.html#sphx-glr-gallery-text-labels-and-annotations-legend-demo-py

Check the link above. There are various other use cases of legend covered.

Remove Legend key



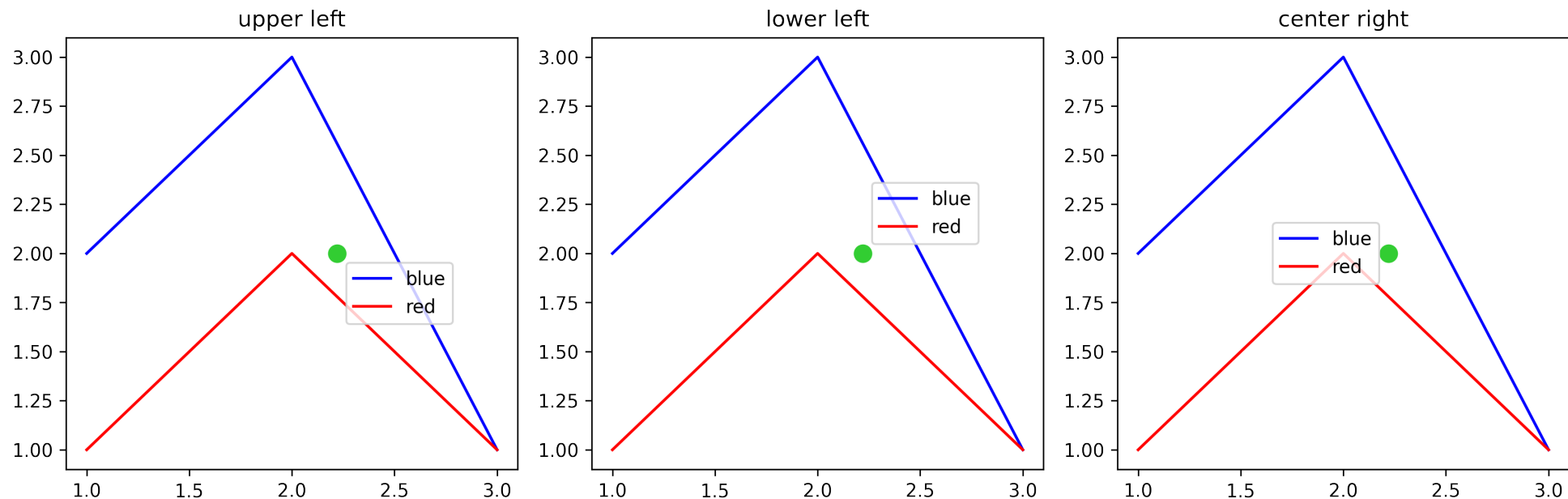
<https://stackoverflow.com/questions/44603480/remove-legend-key-in-matplotlib>

```
import matplotlib.pyplot as plt
import numpy as np
```

```
np.random.seed(100)
x = np.random.normal(size=5000)
fig, ax = plt.subplots()
ax.hist(x, label = 'something')
```

```
ax.legend()
leg = plt.gca().get_legend()
leg.legendHandles[0].set_visible(False)
```

```
plt.show()
```

<https://stackoverflow.com/questions/44413020/how-to-specify-legend-position-in-matplotlib-in-graph-coordinates?>

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = 12, 4
fig, axes = plt.subplots(ncols=3)
locs = ["upper left", "lower left", "center right"]
for l, ax in zip(locs, axes.flatten()): #Iterating over the subplots to create line plots
    ax.set_title(l)
    ax.plot([1,2,3],[2,3,1], "b-", label="blue")
    ax.plot([1,2,3],[1,2,1], "r-", label="red")
    ax.legend(loc=l, bbox_to_anchor=(0.6,0.5)) #Note the green dot in the axes is the bbox_to_anchor
    ax.scatter((0.6),(0.5), s=81, c="limegreen", transform=ax.transAxes)

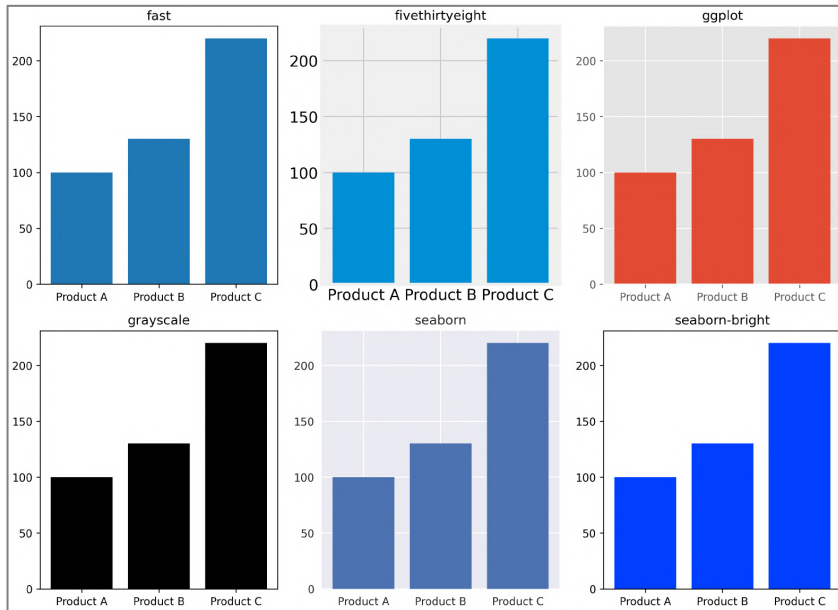
plt.tight_layout()
fig.savefig('Legend loc.png', dpi = 300)
plt.show()
```

upper left	upper center	upper right
center left	center	center right
lower left	lower center	lower right

Legend
'loc' argument

Style Sheets in Matplotlib

[Back](#)

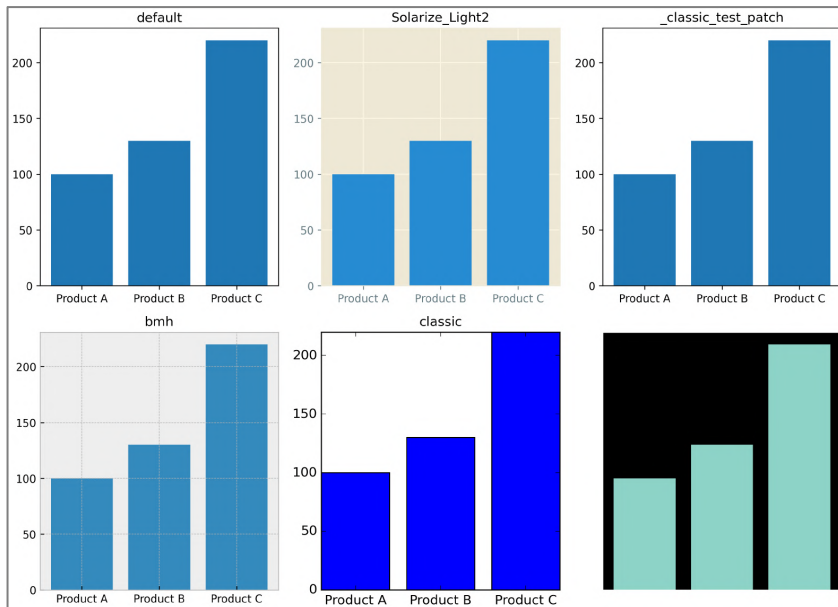


```
import matplotlib.pyplot as plt
fig = plt.figure(dpi=100, figsize=(11, 8), tight_layout=True)
available = ['default'] + plt.style.available
```

#Creating sample data

```
x = ['Product A', 'Product B', 'Product C']
y = [100, 130, 220]
```

```
n, p = 6, 1
for i, style in enumerate(available[n*p:n*(p+1)]):
    with plt.style.context(style):
        ax = fig.add_subplot(2, 3, i + 1)
        ax.bar(x, y)
        ax.set_title(style)
plt.savefig('Styles_second_2.png', bbox_inches = "tight", pad_inches = 0.05, dpi=300)
plt.show()
```

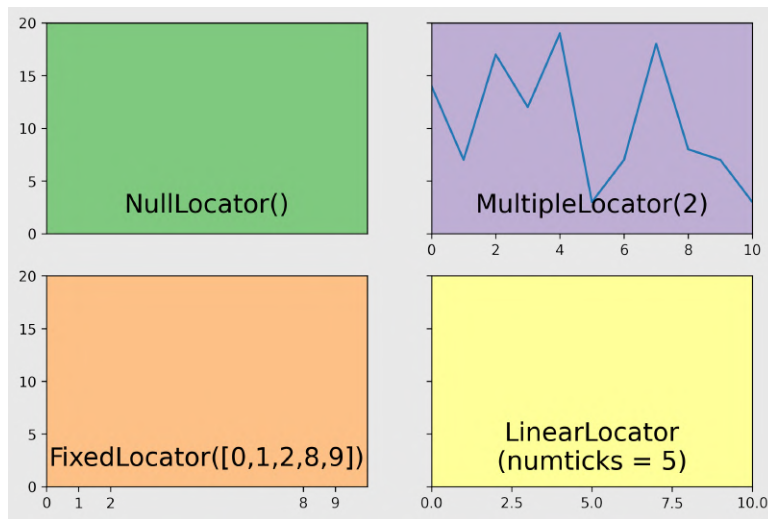


```
import matplotlib.pyplot as plt
fig = plt.figure(dpi=100, figsize=(11, 8), tight_layout=True)
available = ['default'] + plt.style.available
```

#Creating sample data

```
x = ['Product A', 'Product B', 'Product C']
y = [100, 130, 220]
```

```
n, p = 6, 0
for i, style in enumerate(available[n*p:n*(p+1)]):
    with plt.style.context(style):
        ax = fig.add_subplot(2, 3, i + 1)
        ax.bar(x, y)
        ax.set_title(style)
plt.savefig('Styles_second_2.png', bbox_inches = "tight", pad_inches = 0.05, dpi=300)
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker
```

```
#Instantiating figure and axes
fig, ax = plt.subplots(2,2,figsize= (10,5), sharey = True)
```

```
x = np.random.randint(3,20,20)
```

```
#Setting background color of figure
rect = fig.patch
rect.set(facecolor = 'lightgrey', alpha = 0.5)
```

```
ax1,ax2,ax3,ax4 = ax.flatten()
```

```
#ax1.patch.set(facecolor = 'salmon')
#ax2.patch.set(facecolor = 'aquamarine')
```

```
# Setting different colors for the axes using colormap
#colors = [ plt.cm.Pastel1(x) for x in np.linspace(0, 1,4)]
#colors = [ plt.cm.tab20c(x) for x in np.linspace(0, 1,4)]
```

Tick Locators

Back

```
#Accessing colors from the colormap
colors = [ plt.cm.Accent(x) for x in np.linspace(0,0.4,4)]
[ax.patch.set(facecolor = colors[i]) for i,ax in enumerate(fig.axes)]
```

```
#Line Plot in axes ax2
ax2.plot(x)
```

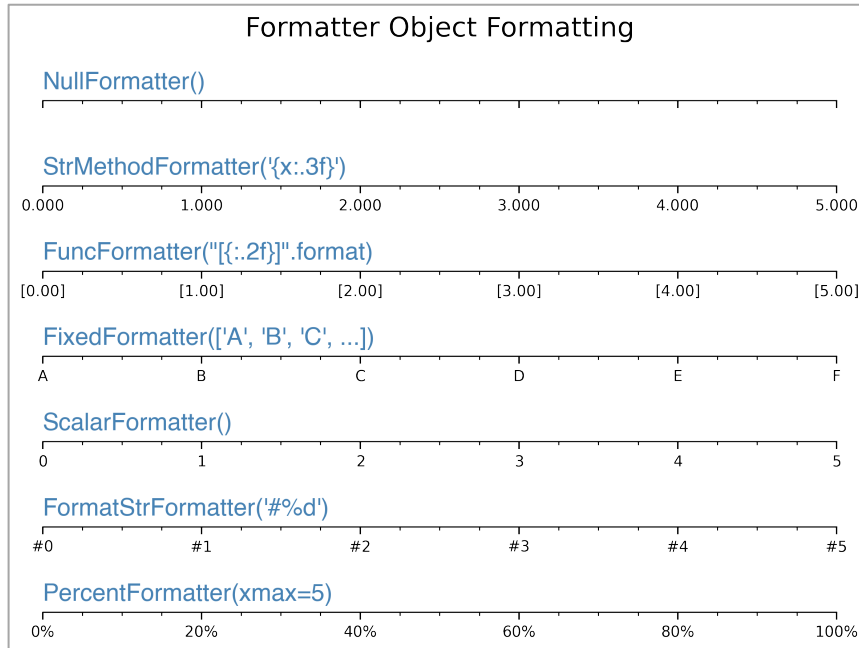
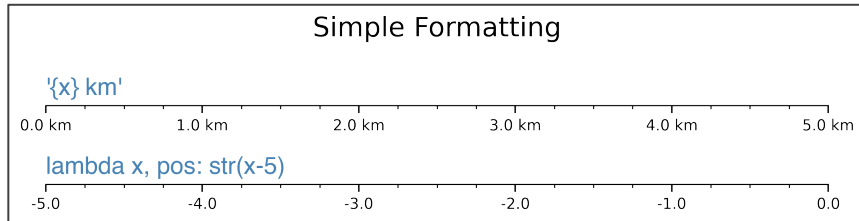
```
# Setting x and y limits in all the axes
[ax.set_xlim(0,10) for ax in fig.axes]
[ax.set_ylim(0,20) for ax in fig.axes]
```

```
#Setting the tick locators using locator class
ax1.xaxis.set_major_locator(ticker.NullLocator())
ax2.xaxis.set_major_locator(ticker.MultipleLocator(2))
ax3.xaxis.set_major_locator(ticker.FixedLocator([0,1,2,8,9]))
ax4.xaxis.set_major_locator(ticker.LinearLocator(5))
```

```
#Making list of Locator labels
Locator_labels = ['NullLocator()', 'MultipleLocator(2)', 'FixedLocator([0,1,2,8,9])', 'LinearLocator\n(numticks = 5)']
```

```
#Adding text iteratively to the figure axes
for ax, label in zip(fig.axes,Locator_labels):
    ax.text(x = 5,y = 2, s = label, rotation_mode = 'anchor', ha = 'center', size = 18)
```

```
fig.savefig('Tick Locators_2.png', dpi=300, format='png', bbox_inches='tight')
plt.show()
```



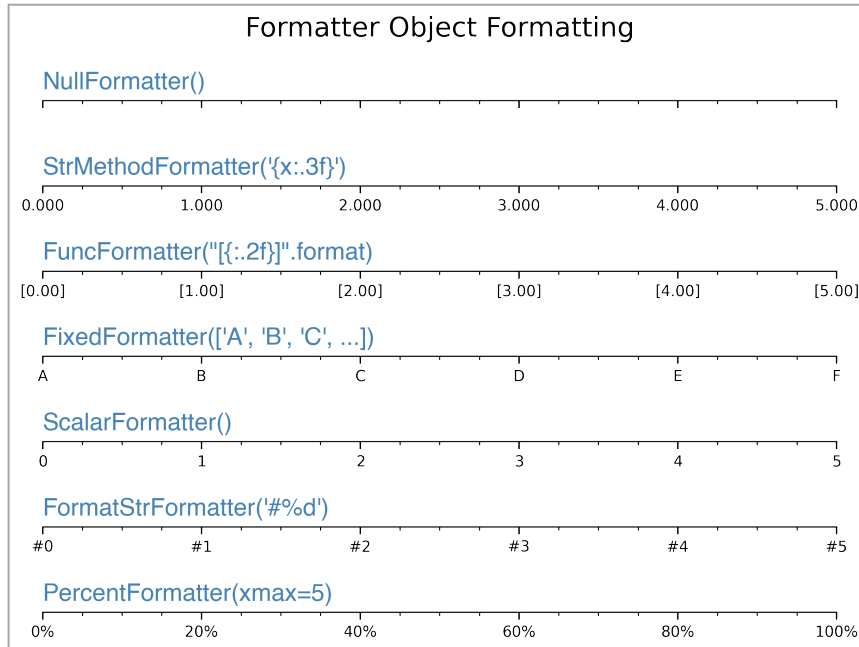
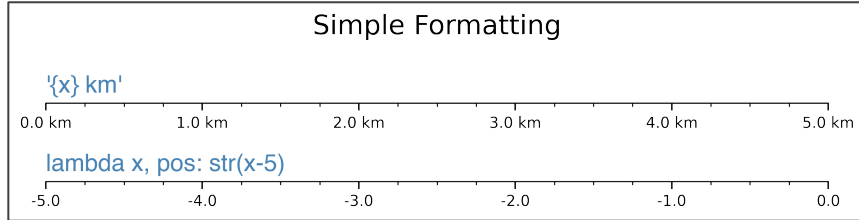
https://matplotlib.org/stable/gallery/ticks_and_spines/tick-formatters.html?highlight=formatter

```
import matplotlib.pyplot as plt
from matplotlib import ticker
def setup(ax, title):
    """Set up common parameters for the Axes in the
    example."""
```

```
    # only show the bottom spine
    ax.yaxis.set_major_locator(ticker.NullLocator())
    ax.spines.right.set_color('none')
    ax.spines.left.set_color('none')
    ax.spines.top.set_color('none')

    # define tick positions
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1.00))
    ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.25))
    ax.xaxis.set_ticks_position('bottom')
    ax.tick_params(which='major', width=1.00, length=5)
    ax.tick_params(which='minor', width=0.75, length=2.5,
                    labelsize=10)
    ax.set_xlim(0, 5)
    ax.set_ylim(0, 1)
    ax.text(0.0, 0.2, title, transform=ax.transAxes,
            fontsize=14, fontname='Monospace',
            color='tab:blue')
```

Code continues to Next slide



https://matplotlib.org/stable/gallery/ticks_and_spines/tick-formatters.html?highlight=formatter

Tick formatters can be set in one of two ways, either by passing a ``str`` # or function to
 ``~.Axis.set\_major\_formatter`` or ``~.Axis.set\_minor\_formatter``,
 # or by creating an instance of one of the various ``~.ticker.Formatter`` classes # and providing that to
 ``~.Axis.set\_major\_formatter`` or ``~.Axis.set\_minor\_formatter``.
 # The first two examples directly pass a ``str`` or function.

```
fig0, axs0 = plt.subplots(2, 1, figsize=(8, 2)) fig0.suptitle('Simple Formatting')
```

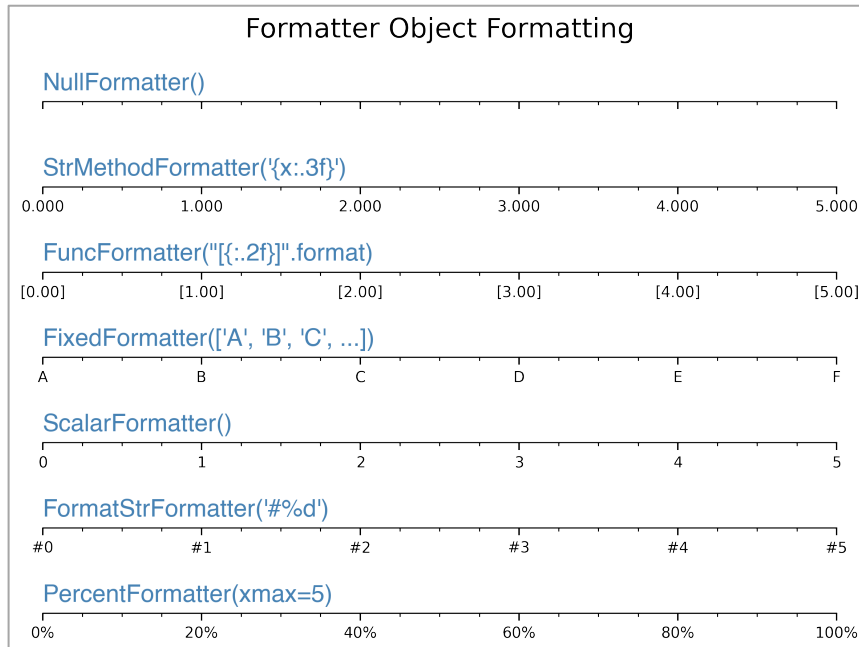
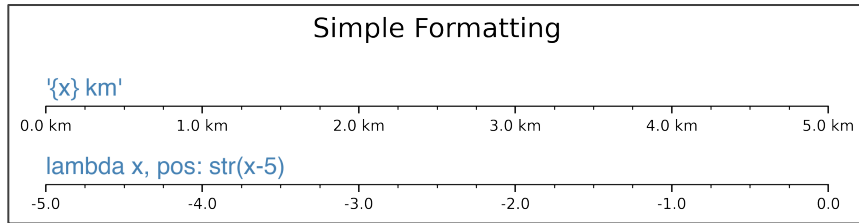
A ``str``, using format string function syntax, can be used directly as a # formatter. The variable ``x`` is the tick
 value and the variable ``pos`` is # tick position. This creates a StrMethodFormatter automatically.

```
setup(axs0[0], title="{x} km")  
axs0[0].xaxis.set_major_formatter('{x} km')
```

A function can also be used directly as a formatter. The function must take # two arguments: ``x`` for the tick
 value and ``pos`` for the tick position, # and must return a ``str`` This creates a FuncFormatter automatically.

```
setup(axs0[1], title="lambda x, pos: str(x-5)") axs0[1].xaxis.set_major_formatter(lambda x, pos: str(x-5))  
fig0.tight_layout()
```

Code continues to Next slide



Code continues from Prev slide

https://matplotlib.org/stable/gallery/ticks_and_spines/tick-formatters.html?highlight=formatter

The remaining examples use Formatter objects.

```
fig1, axs1 = plt.subplots(7, 1, figsize=(8, 6))
fig1.suptitle('Formatter Object Formatting')
```

Null formatter

```
setup(axs1[0], title="NullFormatter()")
axs1[0].axis.set_major_formatter(ticker.NullFormatter())
```

StrMethod formatter

```
setup(axs1[1], title="StrMethodFormatter('{x:.3f}')"
axs1[1].axis.set_major_formatter(ticker.StrMethodFormatter("{x:.3f}"))
```

FuncFormatter can be used as a decorator

```
@ticker.FuncFormatter
def major_formatter(x, pos):
    return f'{x:.2f}'
```

```
setup(axs1[2], title='FuncFormatter("{:.2f}".format)')
axs1[2].axis.set_major_formatter(major_formatter)
```

Fixed formatter

```
setup(axs1[3], title="FixedFormatter(['A', 'B', 'C', ...])")
```

FixedFormatter should only be used together with FixedLocator.

Otherwise, one cannot be sure where the labels will end up.

```
positions = [0, 1, 2, 3, 4, 5]
labels = ['A', 'B', 'C', 'D', 'E', 'F']
axs1[3].axis.set_major_locator(ticker.FixedLocator(positions))
axs1[3].axis.set_major_formatter(ticker.FixedFormatter(labels))
```

Scalar formatter

```
setup(axs1[4], title="ScalarFormatter()")
axs1[4].axis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))
```

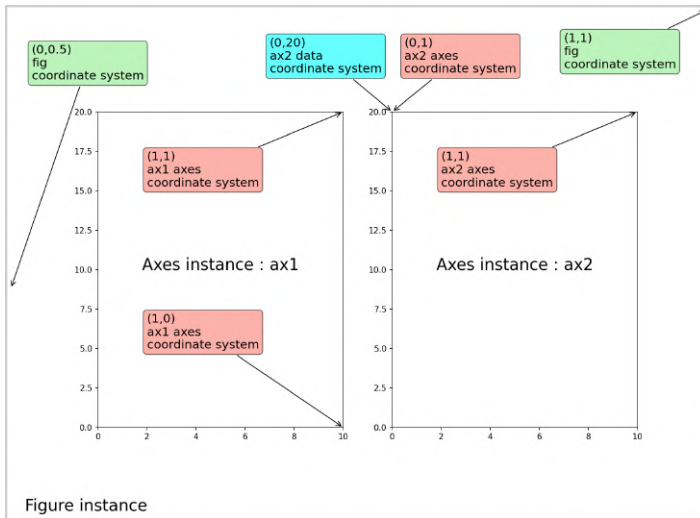
FormatStr formatter

```
setup(axs1[5], title="FormatStrFormatter('%d')")
axs1[5].axis.set_major_formatter(ticker.FormatStrFormatter("%d"))
```

Percent formatter

```
setup(axs1[6], title="PercentFormatter(xmax=5)")
axs1[6].axis.set_major_formatter(ticker.PercentFormatter(xmax=5))
```

```
fig1.tight_layout()
plt.show()
```



Code start

```
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import numpy as np
from matplotlib.text import OffsetFrom
```

```
plt.rcParamsdefaults()
```

```
#Instantiating Figure and Axes
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12,5))
```

```
fig.patch.set(edgecolor = '0.8', lw = 4)
```

```
#Configuring bbox and arrow arguments for axes, data
and fig
```

```
bbox_args_axes = dict(boxstyle = "round", fc = "salmon",
                      alpha = 0.6)
```

```
bbox_args_data = dict(boxstyle = "round", fc = "cyan",
                      alpha = 0.6)
```

```
bbox_args_fig = dict(boxstyle = "round",
                     fc = "lightgreen", alpha = 0.6)
```

```
arrow_args = dict(arrowstyle = "->")
```

```
ax1.annotate(text = '(1,0)\nax1 axes\nncordinate system', xy =(1,0), xycoords = ax1.transAxes,
             xytext=(2,5 ), textcoords=ax1.transData,fontsize = 15,
             bbox = bbox_args_axes, arrowprops = arrow_args, ha = 'left')
```

```
ax1.annotate(text = '(1,1)\nax1 axes\nncordinate system', xy =(1,1),xycoords = ax1.transAxes,
             xytext=(2,17.5), textcoords=ax1.transData,fontsize = 15,
             bbox = bbox_args_axes, arrowprops = arrow_args, va = 'top')
```

```
ax2.annotate(text = '(0,1)\nax2 axes\nncordinate system', xy =(0.0,1),xycoords = ax2.transAxes,
             xytext=(0.5, 22.5), textcoords=ax2.transData,fontsize = 15,
             bbox = bbox_args_axes, arrowprops = arrow_args)
```

```
ax2.annotate(text = '(0,20)\nax2 data\nncordinate system', xy =(0,20),xycoords = ax2.transData,
             xytext=(-5, 22.5), textcoords=ax2.transData,fontsize = 15,
             bbox = bbox_args_data, arrowprops = arrow_args)
```

```
ax1.annotate(text = '(0,0.5)\nfig\nncordinate system', xy =(0,0.5),xycoords = fig.transFigure,
             xytext=(0.03, 0.95), textcoords=fig.transFigure, fontsize = 15,
             bbox = bbox_args_fig, arrowprops = arrow_args)
```

```
ax2.annotate(text = '(1,1)\nfig\nncordinate system', xy =(1,1.1),xycoords = fig.transFigure,
             xytext=(7, 25), textcoords=ax2.transData,fontsize = 15, va = 'top',
             bbox = bbox_args_fig, arrowprops = arrow_args )
```

```
ax2.annotate(text = '(1,1)\nax2 axes\nncordinate system', xy =(1,1),xycoords = ax2.transAxes,
             xytext=(2,17.5), textcoords=ax2.transData,fontsize = 15, va = 'top',
             bbox = bbox_args_axes, arrowprops = arrow_args)
```

```
[ax.set_xlim(0,10) for ax in fig.axes] #Setting x limits in all the axes
```

```
[ax.set_ylim(0,20) for ax in fig.axes] #Setting y limits in all the axes
```

```
ax2.text(x = 0.5, y = 0.5, s = "Axes instance : ax2", ha = 'center',transform = ax2.transAxes, fontsize = 20)
```

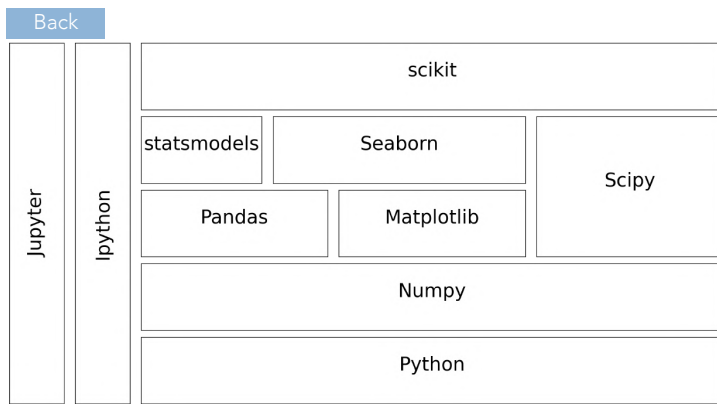
```
ax1.text(x = 0.5, y = 0.5, s = "Axes instance : ax1", ha = 'center',transform = ax1.transAxes, fontsize = 20)
```

```
fig.text(x = 0.02, y = 0.02, s = "Figure instance", fontsize = 20)
```

```
plt.subplots_adjust(bottom =0.2) #Adjusting padding
```

```
#plt.tight_layout(rect = [0,0,0.8,0.8])
```

```
fig.savefig('coordinate systems.png', dpi = 150, format='png', bbox_inches='tight')
```



```
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(13.5, 7))
#Creating Gridspec object
spec = gridspec.GridSpec(nrows = 5,ncols = 11,
                        wspace = 0.2, hspace=0.1)
```

```
#Create Axes for Jupyter
ax1 = fig.add_subplot(spec[:, 0])
#Create Axes Ipython
ax2 = fig.add_subplot(spec[:, 1])
```

```
# Create Axes for scikit
ax3 = fig.add_subplot(spec[0, 2::])
```

```
# Create Axes for Statsmodels
ax4 = fig.add_subplot(spec[1, 2:4])
```

```
# Create Axes for Seaborn
ax5 = fig.add_subplot(spec[1, 4:8])
# Create Axes for Scipy
ax6 = fig.add_subplot(spec[1:3, 8:11])
```

```
# Create Axes for Pandas
ax7 = fig.add_subplot(spec[2, 2:5])
```

```
# Create Axes for Matplotlib
ax8 = fig.add_subplot(spec[2, 5:8])
```

```
# Create Axes for Numpy
ax9 = fig.add_subplot(spec[3, 2:11])
```

```
# Create Axes for Python
ax10 = fig.add_subplot(spec[4, 2:11])
```

```
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks([]) #Removing x ticks
    plt.yticks([]) #Removing y ticks
```

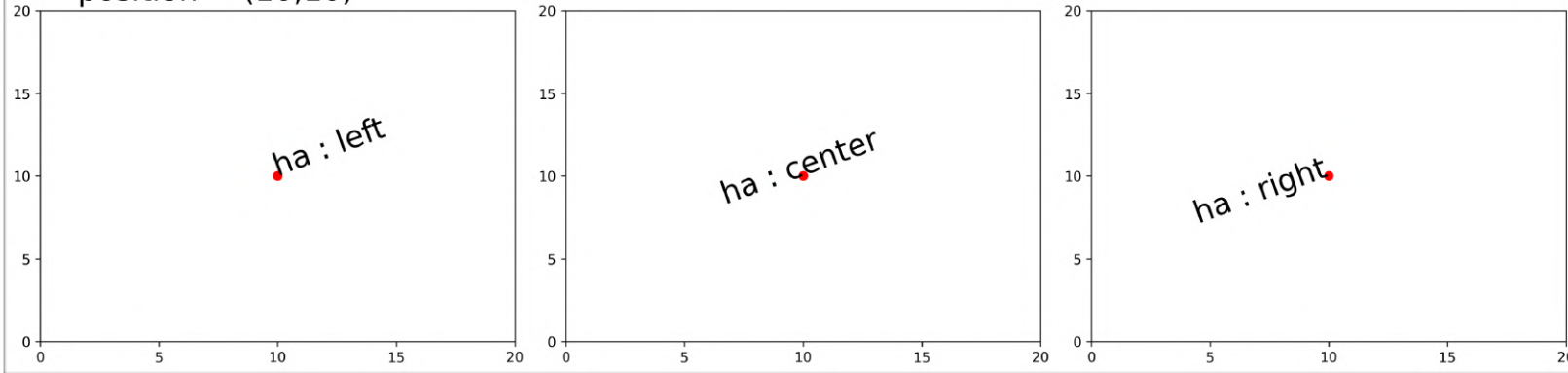
```
#Create labels list containing axes names
labels = ['Jupyter', 'Ipython', 'scikit', 'statsmodels',
          'Seaborn', 'Scipy', 'Pandas', 'Matplotlib',
          'Numpy', 'Python']
ax_list = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10]
```

```
#Adding text to axes
for i,(ax,label) in enumerate(zip(ax_list,labels)):
    if i <=1 :
        ax.text(0.5, 0.5, label, transform = ax.transAxes,
               fontsize = 20, ha = 'center',
               rotation = 90, va = 'center')
    else :
        ax.text(0.5, 0.5, label, transform = ax.transAxes,
               fontsize = 20, ha = 'center')
```

```
fig.savefig('Libraries_Python.png', dpi = 300,
           format='png', bbox_inches='tight')
```

GridSpec Use
Display structure of important packages

Horizontal Alignment Demo - 'left', 'center', 'right'
rotation = 20, rotation_mode = 'anchor'
position = (10,10)



Code Start

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

```
fig, ax = plt.subplots(1,3, figsize=(18,6))
```

```
# Setting both xlim and ylim as (0,20)
[ax_obj.set(ylim = (0,20), xlim = (0,20)) for ax_obj in
fig.axes]
```

```
# Plotting a scatter point at (10,10)
[ax_obj.scatter([10],[10], color = 'r') for ax_obj in fig.axes]
```

```
# Setting number of ticks in both x axis and y axis to 5 in
all the axes
[ax_obj.xaxis.set_major_locator(ticker.LinearLocator(5)) for
ax_obj in fig.axes]
[ax_obj.yaxis.set_major_locator(ticker.LinearLocator(5)) for
ax_obj in fig.axes]
```

```
# Creating a list of horizontal alignment values
ha_values = ['left', 'center', 'right']
# Setting the horizontal alignment by iterating over axes
for i, ax_obj in enumerate(fig.axes):
    if i < len(ha_values):
        ax_obj.text(10,10, f'ha : {ha_values[i]}', rotation = 20,
                    rotation_mode = 'anchor',
                    ha = ha_values[i], fontsize = 22)
```

```
#Setting the figure background color to white
fig.patch.set(color = 'white')
```

```
# Adding a title
fig.suptitle("Horizontal Alignment Demo - 'left', 'center', 'right'\nrotation = 20,
rotation_mode = 'anchor' \nposition = (10,10)", x = 0.05, y = 0.95, fontsize = 22, ha = 'left' )
plt.tight_layout(rect = [0,0,1,0.95])
```

```
fig.savefig("Horizontal Alignment_vertical plots.png",
            dpi=300, format='png', bbox_inches='tight')
```

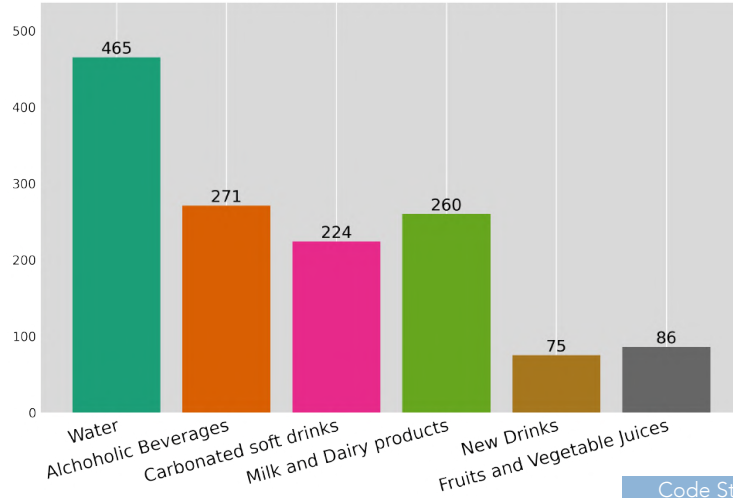
Code End

Horizontal Alignment of Text Instances

Back

[Back](#)

Consumption of Packed beverages in billion liters,
by Beverage Type, Global, 2019

[Code Start](#)

```
data = {'Water':465,
        'Alcoholic Beverages':271,
        'Carbonated soft drinks': 224,
        'Milk and Dairy products': 260,
        'New Drinks': 75,
        'Fruits and Vegetable Juices': 86}
x = list(data.keys())
y = list(data.values())
```

```
# UDF for Adding Annotations to place data labels
def Add_data_labels(rect): #rect is ax.patches object
    for p in rect:
        #Retrieving the Axes container of patch object
        ax = rect[0].axes
        #Adding text iteratively on top of each bar
        #Using get and set on x,y parameters of rectangle
        patch
        ax.text(x = p.get_x()+p.get_width()/2,
                y = ax.get_ylim()[1]*0.01+ p.get_height(),
                s = p.get_height(),
                ha = 'center', size = 12)
```

#User Defined Function to stylize Axes

```
def stylize_axes(ax):
    #Making the axes background light gray
    ax.set_facecolor('.85')

    # Setting the y limits
    ax.set_ylim(0,ax.get_ylim()[1]*1.1)

    #Making the tick lines disappear
    ax.tick_params(length=0)

    #Making the xtick labels rotated
    [labels.set(size = 13,rotation = 15,
                rotation_mode = 'anchor',ha = 'right')
     for labels in ax.get_xticklabels()]

    #Setting grid lines to white color
    ax.grid(True, axis='x', color='white')

    #Setting grid lines below all the artists
    ax.set_axisbelow(True)

    #Making all the spines invisible
    [spine.set_visible(False) for spine in ax.spines.values()]
```

Also check : <https://www.dunderdata.com/blog/create-a-bar-chart-race-animation-in-python-with-matplotlib>

Note : The data figures in the graph are fictitious.

UDF stands for User Defined Function.

Bar Charts

#Code snippet that uses User defined functions – stylize_axes and Add_data_labels

```
# Doing the necessary imports
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.spines as spines
```

```
plt.rcParams()
%matplotlib inline
```

```
#create a list of rgba colors from a matplotlib.cm
colormap
colors = [ plt.cm.Dark2(x) for x in np.linspace(0, 1, len(x))]
```

```
#Instantiating figure, axes
fig, ax = plt.subplots(figsize = (10,6), frameon = True)
```

```
#Creating a bar plot by ax.bar method
# Note the colors passed as the color argument
bars = ax.bar(x,y, color = colors)
```

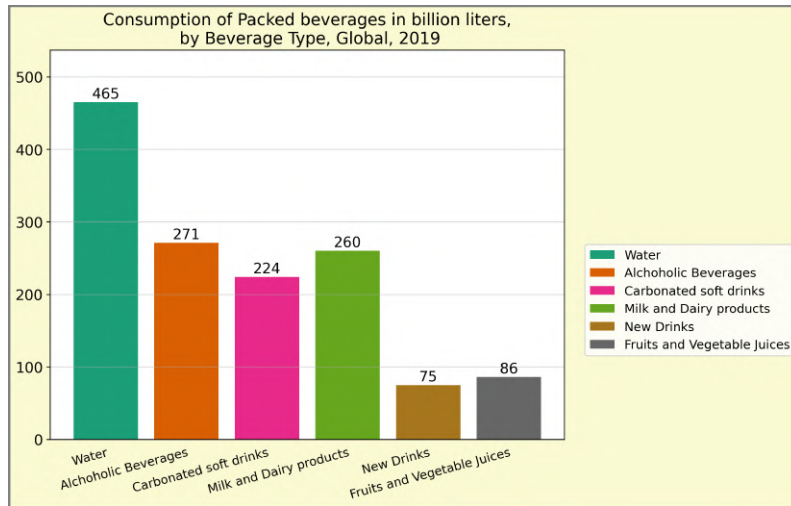
```
#Setting the title of plot
StrTitle = 'Consumption of Packed beverages in billion
liters,\n by Beverage Type, Global, 2019'
ax.set_title(StrTitle, size = 16)
```

```
#Using the user defined functions – to add data labels
and stylize the axes
```

```
Add_data_labels(ax.patches)
stylize_axes(ax)
```

```
fig.savefig('Patches.png', dpi=300, format='png',
bbox_inches='tight')
```

[Code End](#)



```
data = {'Water':465,
        'Alcoholic Beverages':271,
        'Carbonated soft drinks': 224,
        'Milk and Dairy products': 260,
        'New Drinks': 75,
        'Fruits and Vegetable Juices': 86}
x = list(data.keys())
y = list(data.values())
```

Bar Chart With Legend

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.spines as spines
from matplotlib import cm
```

```
colors = [ cm.Dark2(x) for x in np.linspace(0, 1, len(x))]
```

```
%matplotlib inline
```

```
#Instantiating figure and axes
```

```
fig, ax = plt.subplots(figsize = (9,7), frameon = True)
```

```
t = ax.bar(x,y, color = colors)
```

Back

```
#Setting xticklines visibility to false
[ticks.set(visible= False)for ticks in ax.get_xticklines()]
```

```
#Setting the y limits to leave ample room above bar
ax.set_ylim(0,ax.get_ylim()[1]*1.1)
```

```
#Rotating and aligning the xtick labels
for labels in ax.get_xticklabels():
    labels.set(rotation = 15, size = 12,
               rotation_mode = 'anchor',
               ha = 'right'
               )
```

```
#Rotating and aligning the ytick labels
for labels in ax.yaxis.get_ticklabels():
    plt.setp(labels, size = 14, rotation_mode = 'anchor',
             ha = 'right')
```

```
#Setting the figure and axes background colors
ax.patch.set(facecolor = 'white')
plt.setp(fig.patch, facecolor = 'lightgoldenrodyellow',
         edgcolor = 'grey', lw = 4)
```

```
#Making the y gridlines lighter by setting opacity to 0.5
for lines in ax.get_ygridlines():
    lines.set(alpha = 0.5)
```

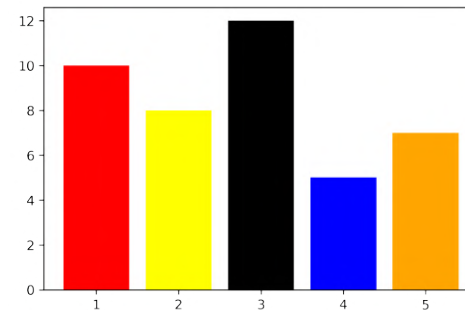
```
#Setting off the xgridlines to False
ax.grid(axis = 'x', b = False)
```

```
# Adding Annotations to place data labels
for p in ax.patches:
    ax.text(p.get_x()+p.get_width()/2,
           ax.get_ylim()[1]*0.01+ p.get_height(),
           p.get_height(),
           ha = 'center', size = 14
           )
```

```
# Adding Legend labels and handles
[p.set_label(x[i]) for i,p in zip(range(len(x)), ax.patches)]
ax.legend(bbox_to_anchor = (1.04,.5),
         loc = "upper left", borderaxespad=0,
         fontsize = 'large')
```

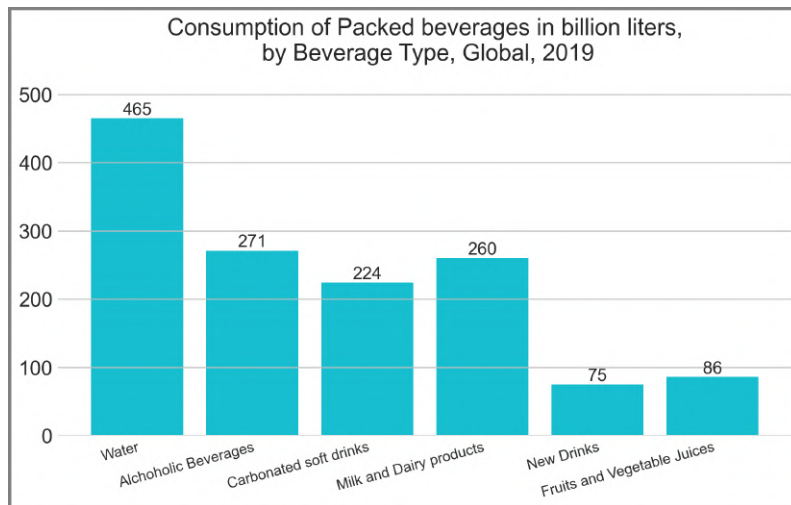
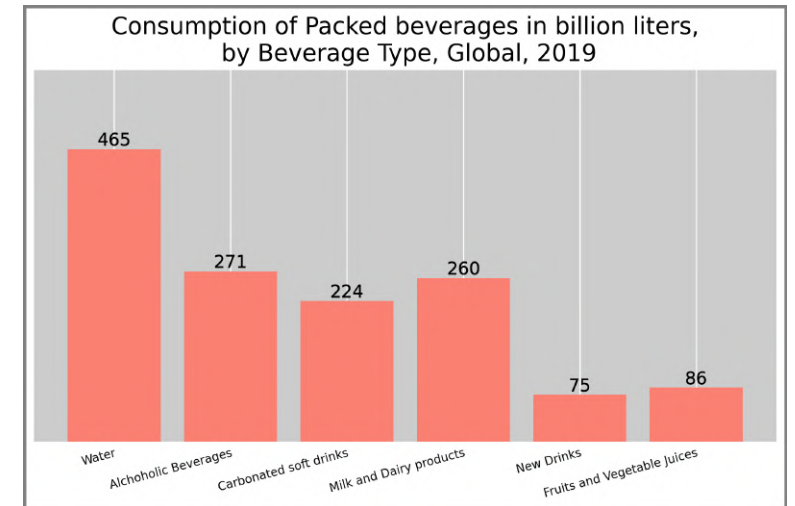
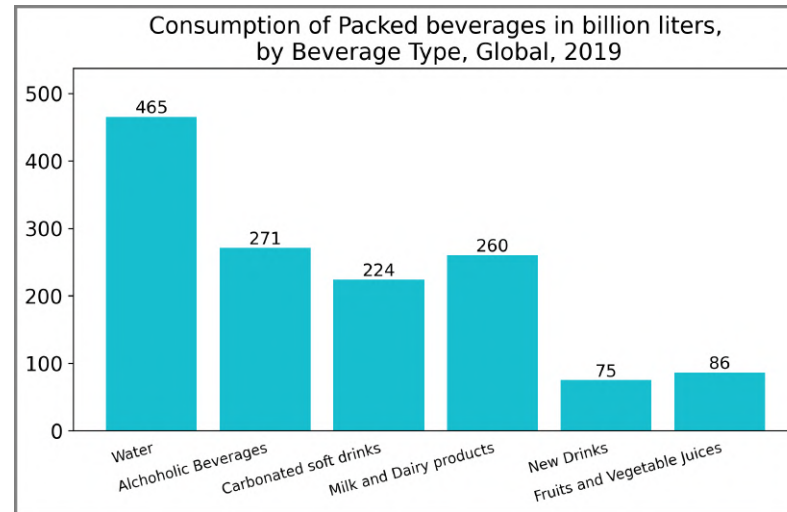
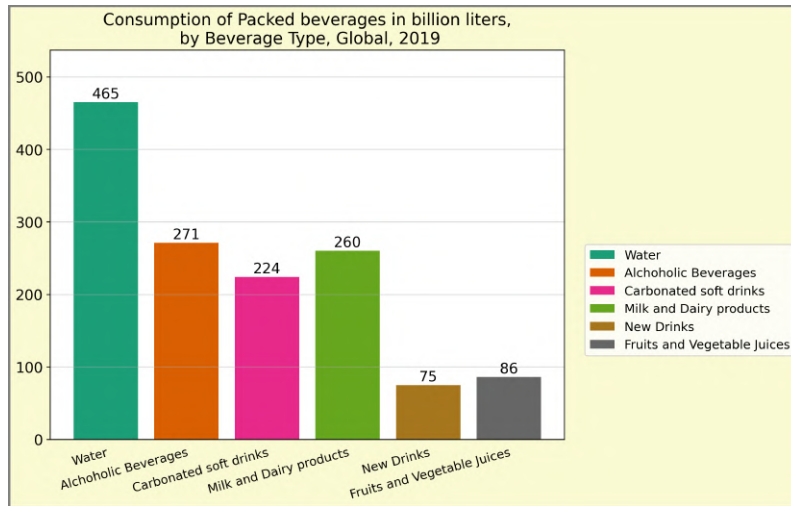
```
#Setting title of the axes
StrTitle = 'Consumption of Packed beverages in billion
liters,\n by Beverage Type, Global, 2019'
ax.set_title(StrTitle, size = 16)
```

```
fig.savefig('Consumption_4.png', dpi=300, format='png',
          bbox_inches='tight')
```



```
#Basic Bar Chart
import matplotlib.pyplot as plt
#data
x = [1, 2, 3, 4, 5]
h = [10, 8, 12, 5, 7]
c = ['red', 'yellow', 'black', 'blue', 'orange']
```

```
#bar plot
plt.bar(x, height = h, color = c)
fig = plt.gcf()
plt.show()
```



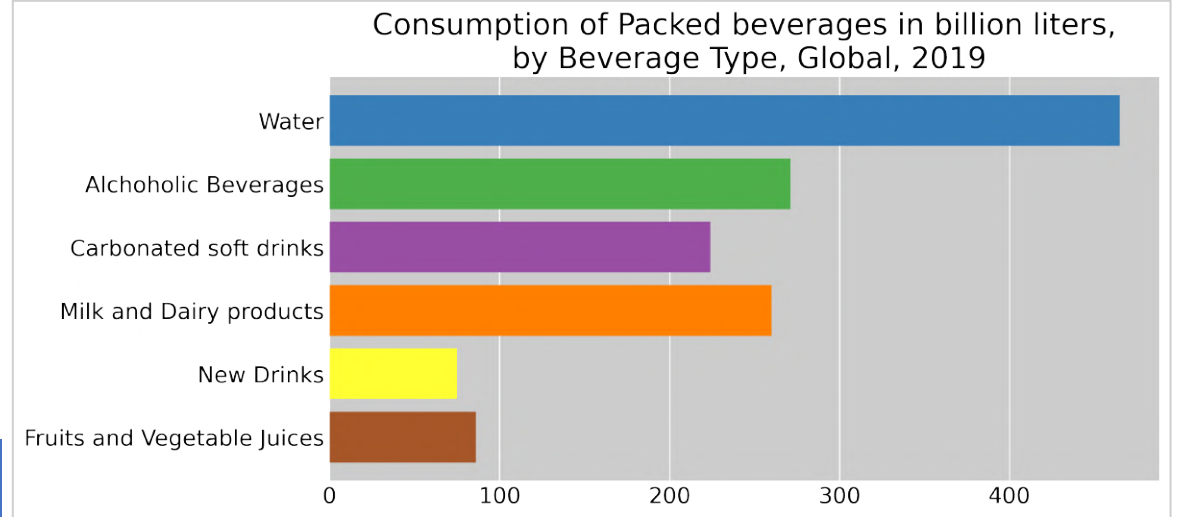
#Using CN color specification
color = 'C9'

#Using Named color in top right for the facecolor or fc of bars
color = 'salmon'
ax.bar(x,y, color = 'salmon')

Bar Chart Styling

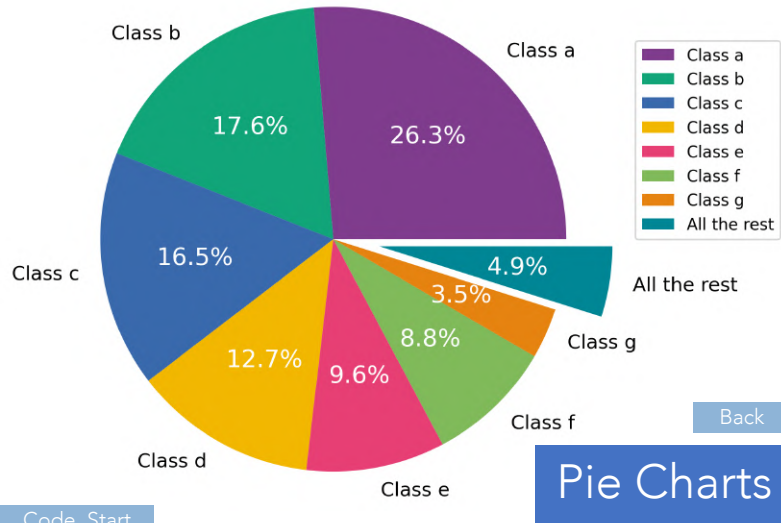
color = 'C9' #Using CN color specification
ax.set_axisbelow(False) #To place grid lines (integral part of axes) above bars
ax.grid(axis = 'y', which = 'major', color = '0.75', b = True, ls = '-')

<https://stackoverflow.com/questions/1726391/matplotlib-draw-grid-lines-behind-other-graph-elements>



```
from matplotlib import cm
colors = cm.Set1.colors[1:]
ax.barh(x,y, color = colors)
```

#Alternatively Accessing colors from [Paletttable external library](https://github.com/jiffyclub/paletttable)
<https://jiffyclub.github.io/paletttable/#matplotlib-discrete-colormap>
colors = paletttable.colorbrewer.qualitative.Dark2_7.mpl_colors



Code Start

#<https://datascience.stackexchange.com/questions/57853/group-small-values-in-a-pie-chart>

#Changes vis-à-vis the code in the link :

- Using colors from external library
- Dictionary key value pairings
- Placement of the legend and the corresponding transform object used

```
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams()
```

```
dic = {'Class a': 26.9,
      'Class b': 18,
      'Class c': 16.8,
      'Class d': 13,
      'Class e': 9.83,
      'Class f': 9,
      'Class g': 3.59,
      'Class h': 0.08,
      'Class i': 1.42,
      'Class j': 1.09,
```

```
'Class k': 0.903,
'Class l': 0.873,
'Class m': 0.28,
'Class n': 0.24,
'Class o': 0.112]
```

```
# group together all elements in the dictionary whose
value is less than 2
# name this group 'All the rest'
import itertools
newdic={}
for key, group in itertools.groupby(dic, lambda k: 'All the
rest' if (dic[k]<2) else k):
    newdic[key] = sum([dic[k] for k in list(group)])
```

#Extracting labels and sizes from the dictionary

```
labels = newdic.keys()
sizes = newdic.values()
```

```
#Accessing colors from external library Palettable
from palettable.cartocolors.qualitative import Bold_10
colors = Bold_10.mpl_colors
```

```
#Instantiating figure and axes
fig, ax = plt.subplots()
```

```
#Creating pie chart and assigning to variable
pie_list = ax.pie(sizes, labels=labels, autopct='%1.1f%%',
                  explode=(0,0,0,0,0,0,0,2), startangle=0,
                  colors = colors)
```

#Note pie_list is a tuple containing list of wedges and text

```
#Set equal scaling of the axes (make circles circular)
ax.axis('equal')
```

```
#Setting the background color of figure to white
fig.patch.set(facecolor = "white")
```

```
#Setting the font size and colors for the value and text
labels
[txt.set(color = 'white', fontsize = 15) for txt in pie_list[2]]
[txt.set(color = 'black', fontsize = 12) for txt in pie_list[1]]
```

```
#Adding a Legend. Note the transform object used
plt.legend(pie_list[0],labels, bbox_to_anchor=(1,0.9),
           loc="upper left", fontsize=10,
           bbox_transform=plt.gcf().transAxes)
```

#Alternate Legend

```
#plt.legend(pie_list[0],labels, bbox_to_anchor=(1,1),
           loc="upper right", fontsize=10,
           bbox_transform=plt.gcf().transFigure)
```

```
plt.tight_layout()
```

```
fig.savefig('PieChart.png',dpi=300, format='png',
           bbox_inches='tight')
```

```
plt.show() #Also check link
```

Code End

#The color code can be replaced by any of the other colormaps as below

#Accessed colors count same as number of values

```
num = len(newdic.values()) #num for number of values
colors = [ plt.cm.tab20c(x) for x in np.linspace(0, 1,num)]
colors = plt.cm.Dark2(range(15))
colors = plt.cm.tab20(range(15))
colors = [ plt.cm.viridis(x) for x in np.linspace(0, 1, num)]
colors = [ plt.cm.Set3(x) for x in np.linspace(0, 1, num)]
colors = plt.cm.tab10.colors
colors = plt.cm.Paired.colors
```

```
colors = plt.cm.nipy_spectral(np.linspace(0.1,1,num))
colors = plt.cm.CMRmap(np.linspace(0.2,1,num))
```

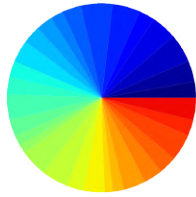

nipy_spectral



CMRmap



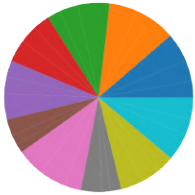
jet



tab20



tab10



Bold_10



```
import palettable
import matplotlib.pyplot as plt
import numpy as np
```

```
#Creating random data for the pie chart
data = np.random.randint(50,200, size=30)
```

```
#Instantiating figure
fig = plt.figure(figsize = (10,7))
```

```
#Axes for nipy_spectral colormap
color = plt.cm.nipy_spectral(np.linspace(0,.9,len(data)))
cc = plt.cycler("color", color)
with plt.style.context({"axes.prop_cycle" : cc}):
    ax1 = fig.add_subplot(231, aspect="equal")
    ax1.pie(data)
    ax1.set_title('nipy_spectral', fontsize = 20,
                  fontweight = 'medium', y = 1, x = 0.5,
                  transform = ax.transAxes)
```

Pie Chart colors

```
#Axes for CMRmap
color = plt.cm.CMRmap(np.linspace(0,0.9,len(data)))
cc = plt.cycler("color", color)
with plt.style.context({"axes.prop_cycle" : cc}):
    ax2 = fig.add_subplot(232, aspect="equal")
    ax2.pie(data)
    ax2.set_title('CMRmap', fontsize = 20,
                  fontweight = 'medium', y = 1, x = 0.5,
                  transform = ax2.transAxes)

#Axes for jet colormap
color = plt.cm.jet(np.linspace(0,0.9,len(data)))
cc = plt.cycler("color", color)
with plt.style.context({"axes.prop_cycle" : cc}):
    ax3 = fig.add_subplot(233, aspect="equal")
    ax3.pie(data)
    ax3.set_title('jet', fontsize = 20, fontweight = 'medium',
                  y = 1, x = 0.5, transform = ax3.transAxes)
```

```
#Axes for tab20 colormap
color = plt.cm.tab20(np.linspace(0,1,len(data)))
cc = plt.cycler("color",
with plt.style.context({"axes.prop_cycle" : cc}):
    ax4 = fig.add_subplot(234, aspect="equal")
    ax4.pie(data)
    ax4.set_title('tab20', fontsize = 20,
                  fontweight = 'medium', y = 1, x = 0.5,
                  transform = ax4.transAxes)
```

```
#Axes for tab10 colormap
color = plt.cm.tab10(np.linspace(0,1,len(data)))
cc = plt.cycler("color", color)
with plt.style.context({"axes.prop_cycle" : cc}):
    ax5 = fig.add_subplot(235, aspect="equal")
    ax5.pie(data)
    ax5.set_title('tab10', fontsize = 20,
                  fontweight 'medium', y = 1, x = 0.5,
                  transform = ax5.transAxes)
```

```
#Axes for Bold_10 colormap from palettable
from palettable.cartocolors.qualitative import Bold_10

color = Bold_10.mpl_colors
cc = plt.cycler("color",Bold_10.mpl_colors)
with plt.style.context({"axes.prop_cycle" : cc}):
    ax6 = fig.add_subplot(236, aspect="equal")
    ax6.pie(data)
    ax6.set_title('Bold_10', fontsize = 20,
                  fontweight = 'medium', y = 1, x = 0.5,
                  transform = ax6.transAxes)

fig.savefig("Piecharts 3.png",dpi=300, format='png',
            bbox_inches='tight')

plt.show()
```

Check the link :

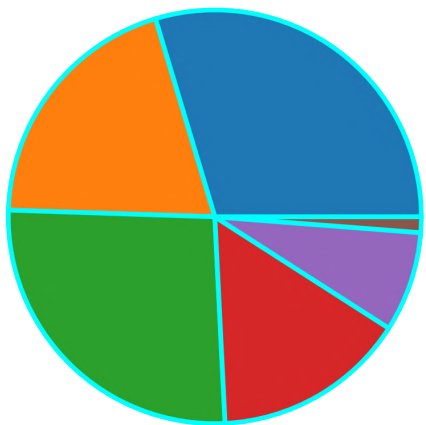
<https://stackoverflow.com/questions/52134364/how-could-i-get-a-different-pie-chart-color>

Check answer by [ImportanceOfBeingErnest](#)

The code has been tweaked for six pie charts in this slide. There is scope to optimize the code with iteration through a list of colors from six colormaps to draw the corresponding pie charts.

Further, check the discussion in the link. [Currently maximum 20 colors available from the listed colormaps under the qualitative colormaps list. Namely the colormaps tab20, tab20b, tab20c.](#)

However, more colors can be obtained from Linear Segmented Colormaps by using the methods in section [ACCESSING COLORS FROM A MATPLOTLIB COLORMAP](#). Note depending on the colormap accessed, the colors extracted may look similar but having different luminosity levels.



Pie Chart Wedge Properties

#<https://stackoverflow.com/questions/20551477/changing-line-properties-in-matplotlib-pie-chart>

Three use case scenarios :

- ✓ Setting the width of the lines separating the pie wedges
- ✓ Setting the facecolor or edgecolor of the wedges
- ✓ Retrieving the wedges from ax.patches if one has an axes object but no direct access to the pie's wedges

Also check :

- Axes.pie() method [Matplotlib, Release 3.4.2, Pg No. 1286](#)
- wedge properties in [Matplotlib, Release 3.4.2, Pg No. 2421-23](#)

#Setting the linewidth, edge color of the wedges

#Necessary imports

```
import matplotlib.pyplot as plt
import numpy as np
```

Fixing random state for reproducibility

```
np.random.seed(19680765)
```

```
ax = plt.subplot(111)
```

```
wedges, texts = ax.pie(np.abs(np.random.randn(6)))
```

```
for w in wedges:
```

```
    w.set_linewidth(2)
```

```
    w.set_edgecolor('cyan')
```

```
fig = ax.figure
```

```
fig.savefig('PieChart_linewidths.png',dpi=300,
            format='png')
```

Retrieving wedges from axes object when no direct access

```
wedges = [patch for patch in ax.patches if \
            isinstance(patch, matplotlib.patches.Wedge)]
```


XKCD Colors

xkcd:pale magenta	xkcd:red wine	xkcd:cherry	xkcd:greyish pink
xkcd:magenta	xkcd:cranberry	xkcd:lipstick red	xkcd:purple brown
xkcd:neon pink	xkcd:medium pink	xkcd:faded pink	xkcd:crimson
xkcd:bubblegum pink	xkcd:burgundy	xkcd:brownish purple	xkcd:watermelon
xkcd:dark magenta	xkcd:bordeaux	xkcd:claret	xkcd:light rose
xkcd:deep magenta	xkcd:ruby	xkcd:grey pink	xkcd:dusty rose
xkcd:electric pink	xkcd:maroon	xkcd:soft pink	xkcd:blush pink
xkcd:hot pink	xkcd:baby pink	xkcd:dusky pink	xkcd:pale rose
xkcd:dark fuchsia	xkcd:light mauve	xkcd:red pink	xkcd:dirty pink
xkcd:reddish purple	xkcd:carnation pink	xkcd:neon red	xkcd:dusty red
xkcd:strong pink	xkcd:rose red	xkcd:dark rose	xkcd:bright red
xkcd:red purple	xkcd:pink red	xkcd:reddish pink	xkcd:faded red
xkcd:barbie pink	xkcd:pinky	xkcd:pinkish	xkcd:darkish red
xkcd:violet red	xkcd:light pink	xkcd:rose	xkcd:light red
xkcd:pink	xkcd:dark pink	xkcd:ugly pink	xkcd:blood red
xkcd:mulberry	xkcd:dull pink	xkcd:rose pink	xkcd:coral pink
xkcd:bubblegum	xkcd:lipstick	xkcd:old pink	xkcd:blood
xkcd:merlot	xkcd:pastel pink	xkcd:carnation	xkcd:fire engine red
xkcd:wine	xkcd:muted pink	xkcd:light maroon	xkcd:salmon pink
xkcd:deep pink	xkcd:pale pink	xkcd:cherry red	
xkcd:dark hot pink	xkcd:rosy pink	xkcd:pinky red	
xkcd:bubble gum pink	xkcd:wine red	xkcd:old rose	
xkcd:purple red	xkcd:warm pink	xkcd:dusky rose	
xkcd:berry	xkcd:mauve	xkcd:dusty pink	
xkcd:raspberry	xkcd:pig pink	xkcd:dark maroon	
xkcd:cerise	xkcd:rouge	xkcd:lightish red	
xkcd:purplish red	xkcd:light burgundy	xkcd:carmine	
xkcd:powder pink	xkcd:rosa	xkcd:scarlet	
xkcd:dark mauve	xkcd:deep rose	xkcd:strawberry	
xkcd:darkish pink	xkcd:pinkish red	xkcd:purplish brown	

Back

#Creating user defined function to return a tuple – first element being a figure containing color patches and second element being the figure name

```
from matplotlib.patches import Rectangle
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

```
def plot_colortable(colors, title, sort_colors=True,
emptyrows=0, x=0, colcount = 4 , rowcount = 30 ):
    """ colors : any of the color tables from mcolors module
    title : title to be displayed in the figure
    sort_colors : Whether the colors are to be sorted
    x : the iteration count
    colcount : number of columns
    rowcount : number of rows
    """
```

```
cell_width = 212
cell_height = 22
swatch_width = 48
margin = 12
topmargin = 40
```

```
p = colcount*rowcount
```

```
# Sort colors by hue, saturation, value and name
if sort_colors is True:
    by_hsv =
sorted((tuple(mcolors.rgb_to_hsv(mcolors.to_rgb(color))),
            name)
        for name, color in colors.items())
    names = [name for hsv, name in by_hsv]
else:
    names = list(colors)
```

```
n = len(names[x*p:(x+1)*p])
nrows = rowcount - emptyrows
ncols = n // nrows + int(n % nrows > 0)
```

```
width = cell_width * ncols + 2 * margin
height = cell_height * nrows + margin + topmargin
dpi = 72
```

```
fig, ax = plt.subplots(figsize=(width / dpi, height / dpi),
                        dpi=dpi)
fig.subplots_adjust(margin/width, margin/height,
                    (width-margin)/width,
                    (height-topmargin)/height)
```

XKCD Colors

```
ax.set_xlim(0, cell_width * ncols)
ax.set_ylim(cell_height * (nrows-0.5), -cell_height/2.)
ax.yaxis.set_visible(False)
ax.xaxis.set_visible(False)
ax.set_axis_off()
ax.set_title(title, fontsize=20, loc="left", pad=10)
```

```
for i, name in enumerate(names[x*p:(x+1)*p]):
    row = i % nrows
    col = i // nrows
    y = row * cell_height
```

```
swatch_start_x = cell_width * col
text_pos_x = cell_width * col + swatch_width + 7
```

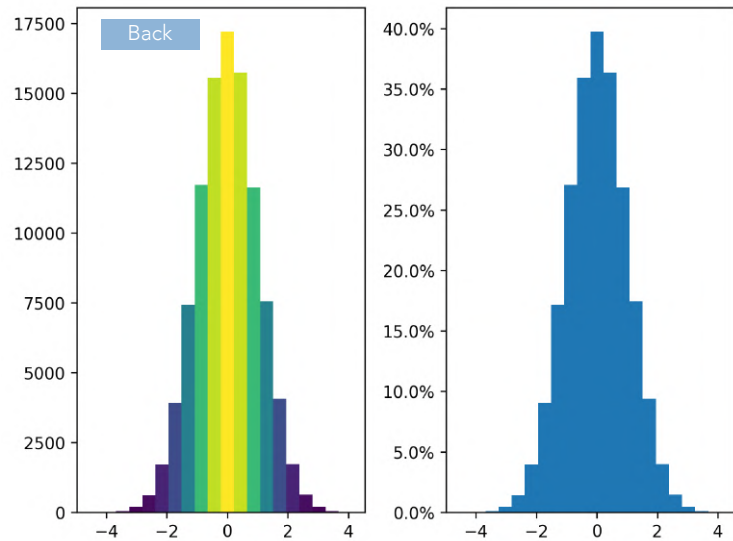
```
ax.text(text_pos_x, y, name, fontsize=14,
        horizontalalignment='left',
        verticalalignment='center')
```

```
ax.add_patch(Rectangle(xy=(swatch_start_x, y-9),
                        width=swatch_width,
                        height=18, facecolor=colors[name],
                        edgecolor='0.7'))
```

```
filename = f'XKCD_Colors_{x+2}.png'
return fig, filename
```

```
#Creating the figure using the user defined function
xkcd_fig, filename =
plot_colortable(mcolors.XKCD_COLORS,
                "XKCD Colors",
                x = 7, rowcount = 30, colcount = 4)
```

```
xkcd_fig.patch.set(facecolor = 'white')
```



<https://matplotlib.org/stable/gallery/statistics/hist.html#sphx-glr-gallery-statistics-hist-py>

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
```

```
plt.rcParamsdefaults()
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)
```

```
N_points = 100000
n_bins = 20
```

```
# Generate a normal distribution, center at x=0 and y=5
x = np.random.randn(N_points)
y = .4 * x + np.random.randn(100000) + 5
```

```
#Instantiating figure and axes
fig, axs = plt.subplots(1, 2, tight_layout=True)
```

```
# N is the count in each bin, bins is the lower-limit of the
bin
N, bins, patches = axs[0].hist(x, bins=n_bins)
```

```
# We'll color code by height, but you could use any scalar
fracs = N / N.max()
```

```
# we need to normalize the data to 0..1 for the full range
of the colormap
norm = colors.Normalize(fracs.min(), fracs.max())
```

```
# Now, we'll loop through our objects and set the color of
each accordingly
for thisfrac, thispatch in zip(fracs, patches):
    color = plt.cm.viridis(norm(thisfrac))
    thispatch.set_facecolor(color)
```

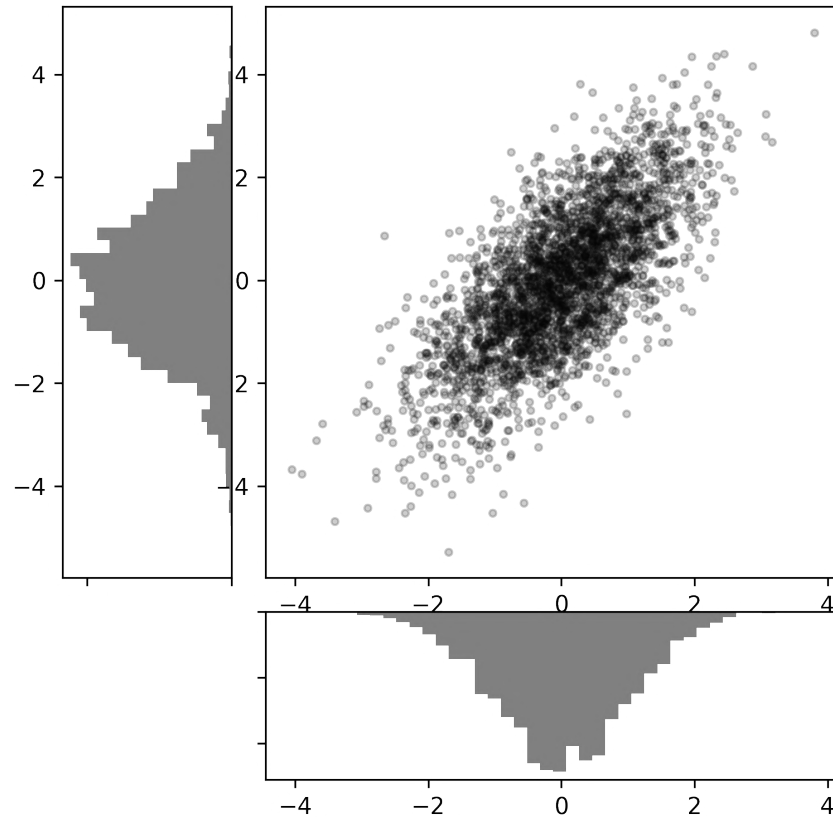
```
# We can also normalize our inputs by the total number of
counts
axs[1].hist(x, bins=n_bins, density=True)
```

```
# Now we format the y-axis to display percentage
axs[1].yaxis.set_major_formatter(PercentFormatter(xmax=
1))
```

```
fig.savefig(f'Histograms_colored.png',dpi=300,
            format='png', bbox_inches='tight')
```

```
plt.show()
```

Histogram Color Coded



<https://jakevdp.github.io/PythonDataScienceHandbook/04.08-multiple-subplots.html>

```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.rcParams()
```

```
# Create some normally distributed data
```

```
mean = [0, 0]
```

```
cov = [[1, 1], [1, 2]]
```

```
prng = np.random.default_rng(123456)
```

```
x, y = prng.multivariate_normal(mean, cov, 3000).T
```

```
# Set up the axes with gridspec
```

```
fig = plt.figure(figsize=(6, 6))
```

```
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
```

```
main_ax = fig.add_subplot(grid[:-1, 1:])
```

```
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
```

```
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)
```

```
# scatter points on the main axes
```

```
main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)
```

```
# histogram on the attached axes
```

```
x_hist.hist(x, 40, histtype='stepfilled', orientation='vertical', color='gray')
```

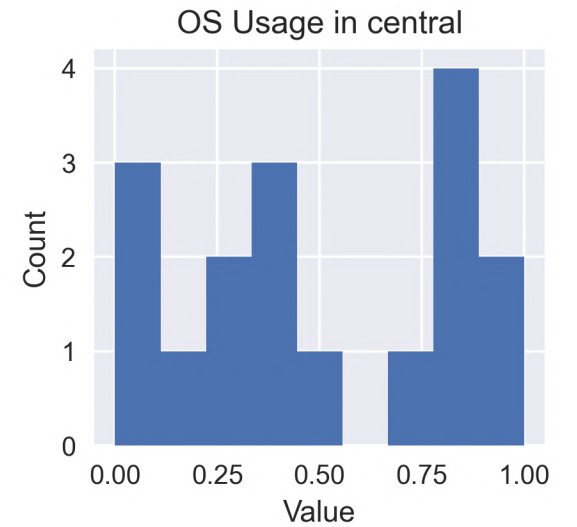
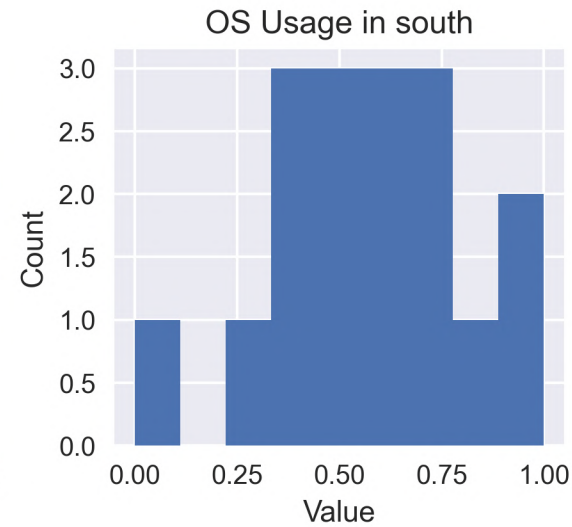
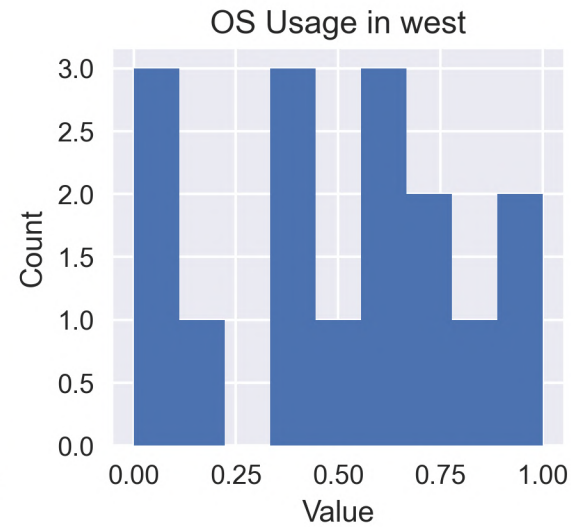
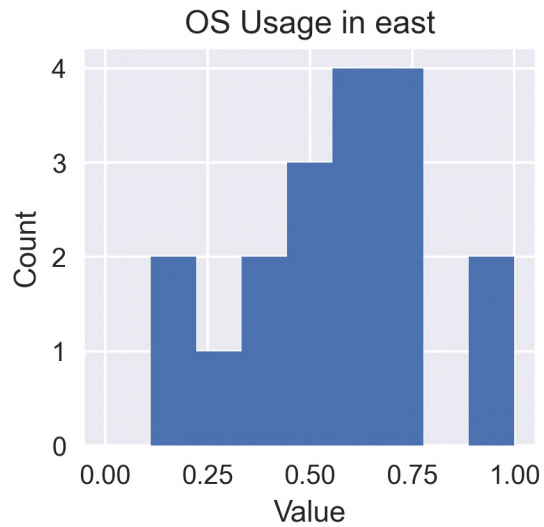
```
x_hist.invert_yaxis()
```

```
y_hist.hist(y, 40, histtype='stepfilled', orientation='horizontal', color='gray')
```

```
y_hist.invert_xaxis()
```

```
fig.savefig('Marginal histograms.png', dpi = 300, transparent = True)
```

Marginal Histogram



<https://stackoverflow.com/questions/43962735/creating-barplots-using-for-loop-using-pandas-matplotlib>

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Create some dummy data
data = pd.DataFrame()
zones = ["east", "west", "south", "central"]
```

```
data['zone'] = np.hstack([[zone]*np.random.randint(10, 20) for zone in zones])
data['OS Usage'] = np.random.random(data.shape[0])
```

```
# Now create a figure
fig, axes = plt.subplots(1, 4, figsize=(12, 3))
```

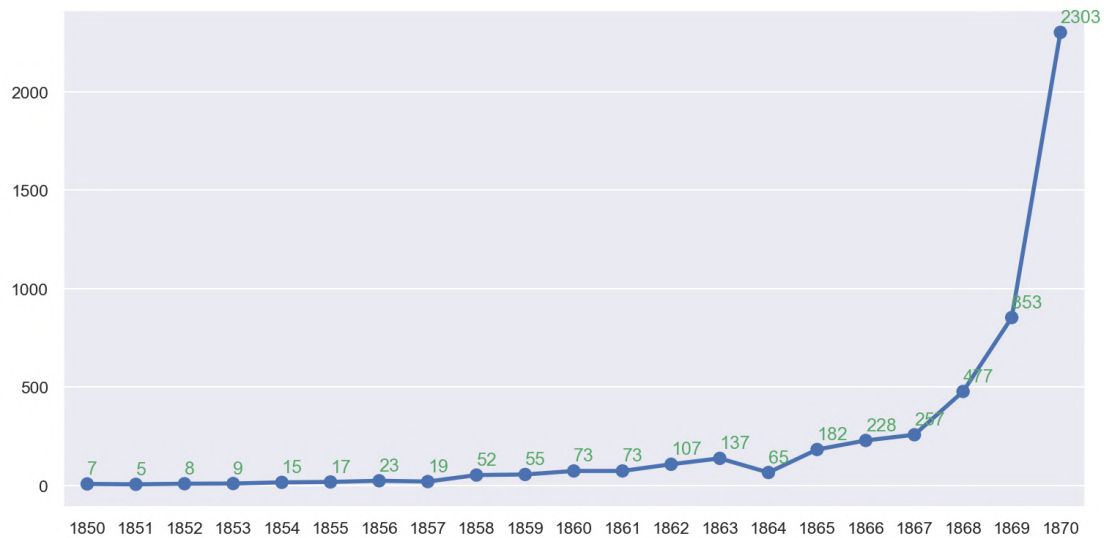
```
# Now plot each zone on a particular axis
```

```
for i, zone in enumerate(zones):
    data.loc[data.zone==zone].hist(column='OS Usage',
                                   bins=np.linspace(0, 1, 10),
                                   ax=axes[i],
                                   sharey=True)
    axes[i].set_title('OS Usage in {0}'.format(zone))
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Count')
```

```
fig.tight_layout()
fig.savefig('Barplots_using_for_loops_Pandas.png', dpi=300, format='png',
          bbox_inches='tight')
```

Histograms

- Using for loop to create histograms in multiple subplots



#<https://stackoverflow.com/questions/37106828/how-to-get-data-labels-on-a-seaborn-pointplot>

```
Soldier_years = [1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870]
num_records_yob = [7, 5, 8, 9, 15, 17, 23, 19, 52, 55, 73, 73, 107, 137, 65, 182, 228, 257, 477, 853, 2303]
```

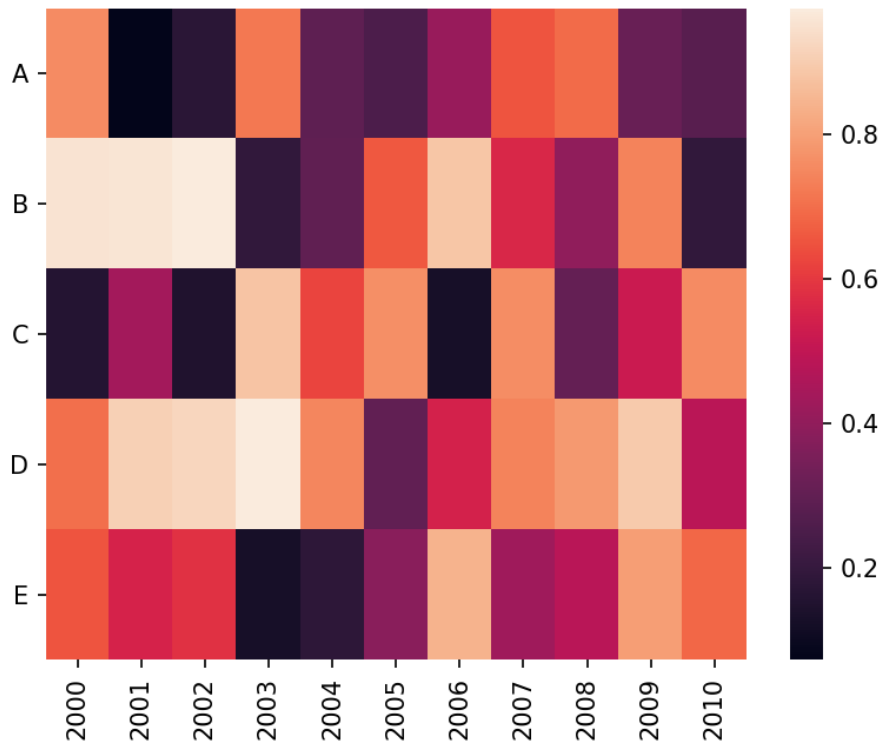
```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="darkgrid")

f, (ax) = plt.subplots(figsize=(12, 6), sharex=True)
```

```
sns.set_style("darkgrid")
ax = sns.pointplot(x=Soldier_years, y=num_records_yob)
[ax.text(p[0], p[1]+50, p[1], color='g') for p in zip(ax.get_xticks(), num_records_yob)]

ax.figure.savefig(f'Datalabels_Seaborn_Pointplot.png', dpi=150, format='png', bbox_inches='tight')
```

Seaborn point plot
Adding data labels



<https://stackoverflow.com/questions/35788626/how-to-plot-a-heatmap-from-pandas-dataframe>

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# create some random data; replace that by your actual dataset
data = pd.DataFrame(np.random.rand(11, 5), columns=['A', 'B', 'C', 'D', 'E'], index = range(2000, 2011, 1))

# plot heatmap
ax = sns.heatmap(data.T)

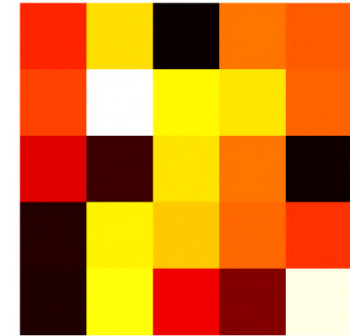
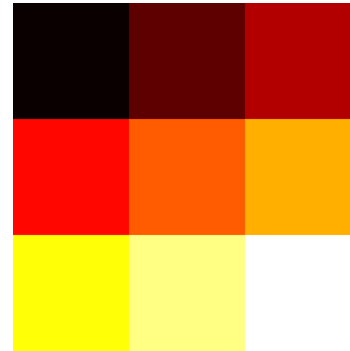
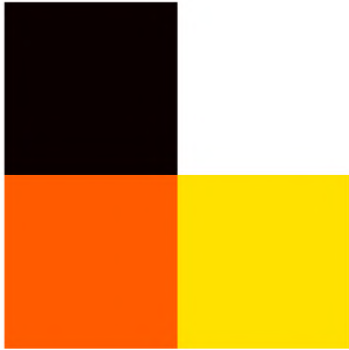
# turn the axis label
for item in ax.get_yticklabels():
    item.set_rotation(0)

for item in ax.get_xticklabels():
    item.set_rotation(90)

# save figure
ax.figure.savefig(f'Heatmap_Pandas_Dataframe.png',dpi=150, format='png', bbox_inches='tight')
plt.show()
```

Plotting Heat Map from Pandas

Pandas + Seaborn + Matplotlib



#<https://stackoverflow.com/questions/9295026/matplotlib-plots-removing-axis-legends-and-white-spaces?rq=1>

```
fig = plt.figure(figsize=(5, 5))
fig.patch.set_visible(False) # turn off the patch

plt.imshow([[0, 1], [0.5, 0.7]], interpolation="nearest",
           cmap='hot')
plt.axis('off')

plt.savefig("test.png", dpi = 300)
```

matplotlib.axes.Axes.imshow

#<https://stackoverflow.com/users/190597/unutbu>

```
import matplotlib.pyplot as plt
import numpy as np

def make_image(data, outputname, size=(1, 1), dpi=80):
    fig = plt.figure()
    fig.set_size_inches(size)
    ax = plt.Axes(fig, [0., 0., 1., 1.])
    ax.set_axis_off()
    fig.add_axes(ax)
    plt.set_cmap('hot')
    ax.imshow(data, aspect='equal')
    plt.savefig(outputname, dpi=dpi)
```

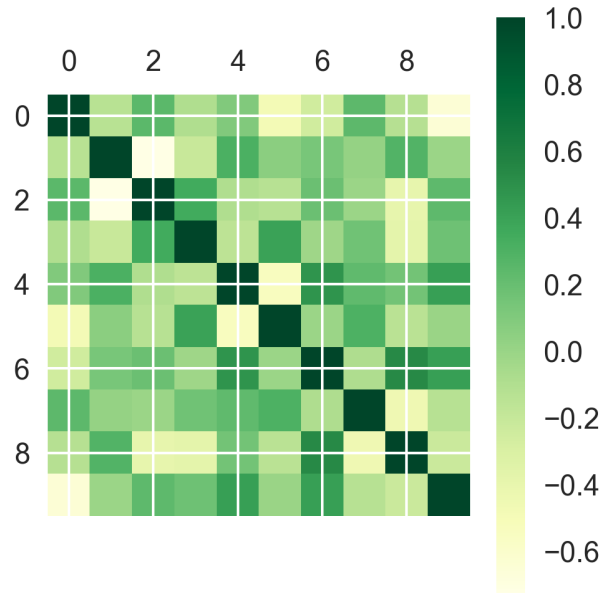
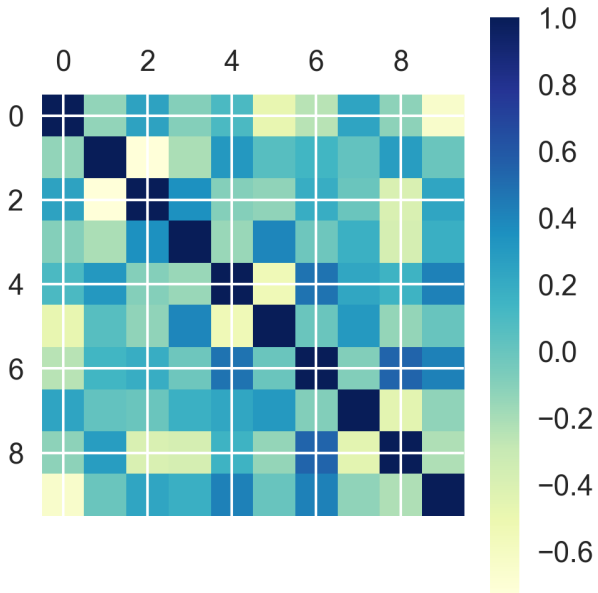
```
# data = mpimg.imread(inputname)[:,:,0]
data = np.arange(1,10).reshape((3, 3))
```

```
make_image(data, 'test2.png', dpi = 300)
```

#<https://stackoverflow.com/users/249341/hooked>

```
from numpy import random
import matplotlib.pyplot as plt

data = random.random((5,5))
img = plt.imshow(data, interpolation='nearest')
img.set_cmap('hot')
plt.axis('off')
plt.savefig("test3.png", bbox_inches='tight', dpi = 300)
```

Code Start

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.DataFrame(np.random.rand(10, 10))
```

```
fig, axes = plt.subplots(ncols=2, figsize=(8, 4))
```

```
ax1, ax2 = axes
```

```
im1 = ax1.matshow(df.corr(), cmap = 'YlGnBu')
im2 = ax2.matshow(df.corr(), cmap = 'YlGn')
```

```
fig.colorbar(im1, ax=ax1)
fig.colorbar(im2, ax=ax2)
fig.savefig('plotting_two_adjacent_heatmaps.png',
            dpi = 300, bbox_inches='tight')
```

Plotting heat maps From Pandas Data frame and Correlation Matrix

Creating User Defined Function

[#https://stackoverflow.com/questions/55663030/
#plotting-two-heat-maps-side-by-side-in-matplotlib/](https://stackoverflow.com/questions/55663030/#plotting-two-heat-maps-side-by-side-in-matplotlib/)

```
def corr_heatmaps(data1, data2, method='pearson'):
```

```
    # Basic Configuration
```

```
    fig, axes = plt.subplots(ncols=2, figsize=(12, 12))
```

```
    ax1, ax2 = axes
```

```
    corr_matrix1 = data1.corr(method=method)
```

```
    corr_matrix2 = data2.corr(method=method)
```

```
    columns1 = corr_matrix1.columns
```

```
    columns2 = corr_matrix2.columns
```

```
    # Heat maps.
```

```
    im1 = ax1.matshow(corr_matrix1, cmap='coolwarm')
```

```
    im2 = ax2.matshow(corr_matrix2, cmap='coolwarm')
```

```
    # Formatting for heat map 1.
```

```
    ax1.set_xticks(range(len(columns1)))
```

```
    ax1.set_yticks(range(len(columns1)))
```

```
    ax1.set_xticklabels(columns1)
```

```
    ax1.set_yticklabels(columns1)
```

```
    ax1.set_title(data1.name, y=-0.1)
```

```
    plt.setp(ax1.get_xticklabels(), rotation=45, ha='left', rotation_mode='anchor')
```

```
    plt.colorbar(im1, fraction=0.045, pad=0.05, ax=ax1)
```

```
    # Formatting for heat map 2.
```

```
    ax2.set_xticks(range(len(columns2)))
```

```
    ax2.set_yticks(range(len(columns2)))
```

```
    ax2.set_xticklabels(columns2)
```

```
    ax2.set_yticklabels(columns2)
```

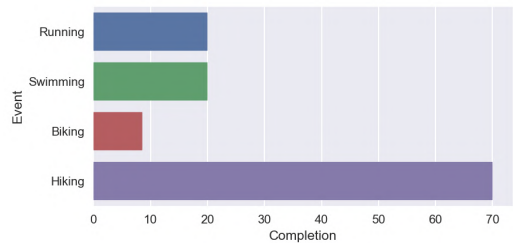
```
    ax2.set_title(data2.name, y=-0.1)
```

```
    plt.setp(ax2.get_xticklabels(), rotation=45, ha='left', rotation_mode='anchor')
```

```
    plt.colorbar(im2, fraction=0.045, pad=0.05, ax=ax2)
```

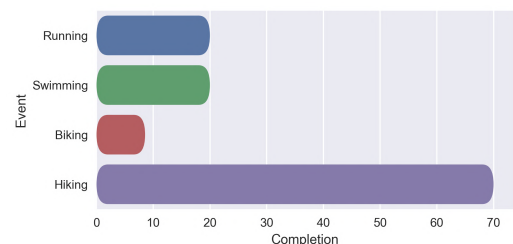
```
    fig.tight_layout()
```

Code End



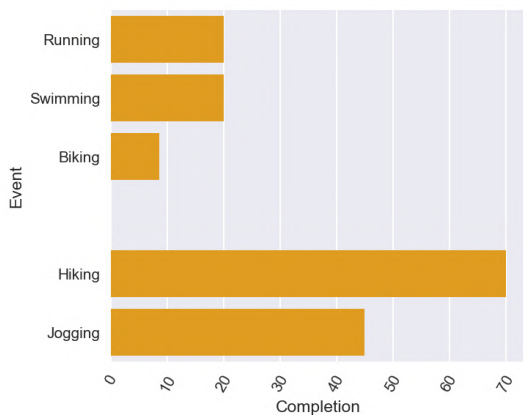
Seaborn horizontal bar plot

boxstyle: round
roundingsize:.015



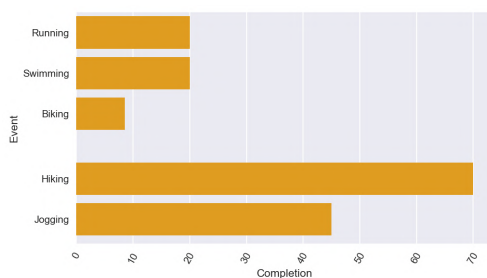
Seaborn horizontal bar plot

boxstyle: round
roundingsize : 2



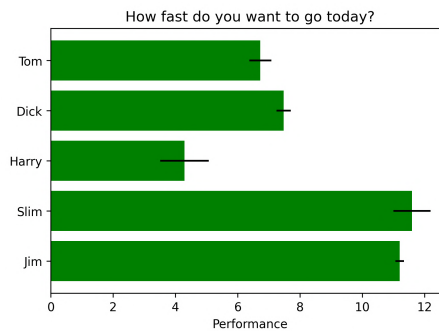
Seaborn horizontal bar plot

- Create spacing between bars
- Adding dummy row in data



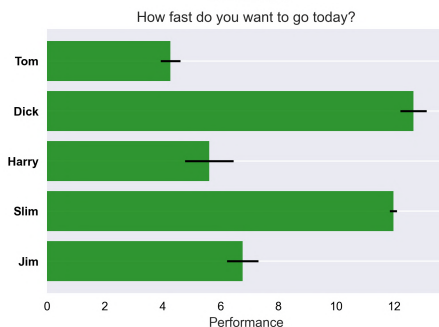
Seaborn horizontal bar plot

- Create spacing between bars
- Resetting position of tick/ ticklabel/



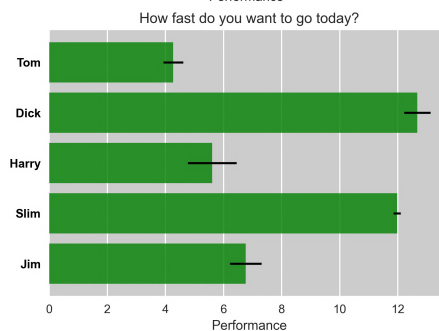
Basic horizontal bar plot

Basic



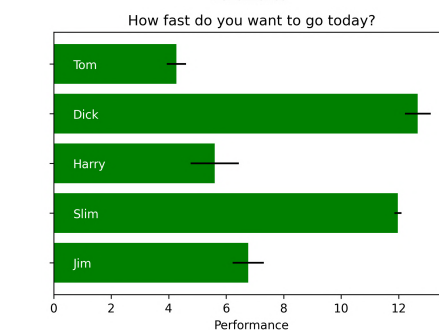
horizontal bar plot

seaborn style
Only y gridlines



horizontal bar plot

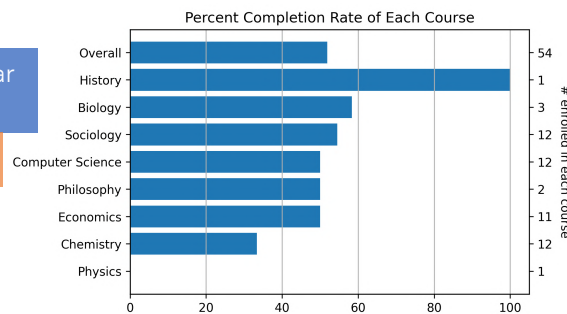
seaborn style
Only xgridlines
Axes.set_belowaxis(True)



horizontal bar plot

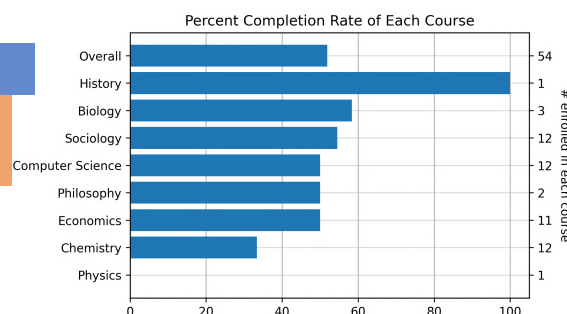
default style
Tick labels inside direction
Padding for tick labels

Note on Practical Implementation of tick related methods



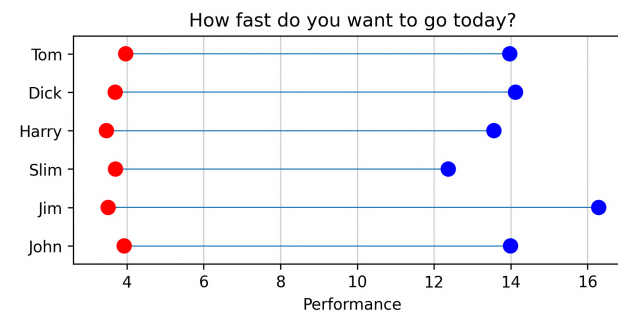
horizontal bar plot

Secondary Axis twinx
Gridlines above bars



horizontal bar plot

Secondary Axis twinx
Gridlines below bars

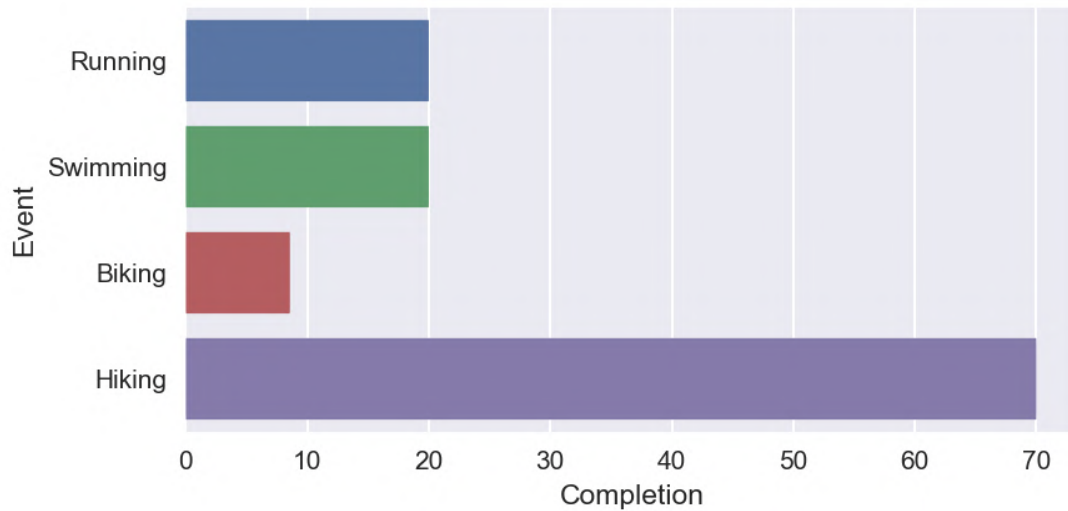


Connected dot plots

default style

Using Circle patches





#<https://stackoverflow.com/questions/61568935/seaborn-barplot-with-rounded-corners>

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import FancyBboxPatch
```

```
plt.style.use("seaborn")
mydict = {
    'Event': ['Running', 'Swimming', 'Biking', 'Hiking'],
    'Completed': [2, 4, 3, 7],
    'Participants': [10, 20, 35, 10]}
```

```
df = pd.DataFrame(mydict).set_index('Event')
df = df.assign(Completion=(df.Completed / df.Participants) * 100)
print(df)
```

```
plt.subplots(figsize=(6, 3))
sns.set_color_codes("pastel")
```

```
ax = sns.barplot(x=df.Completion, y=df.index, orient='h', joinstyle='bevel')
```

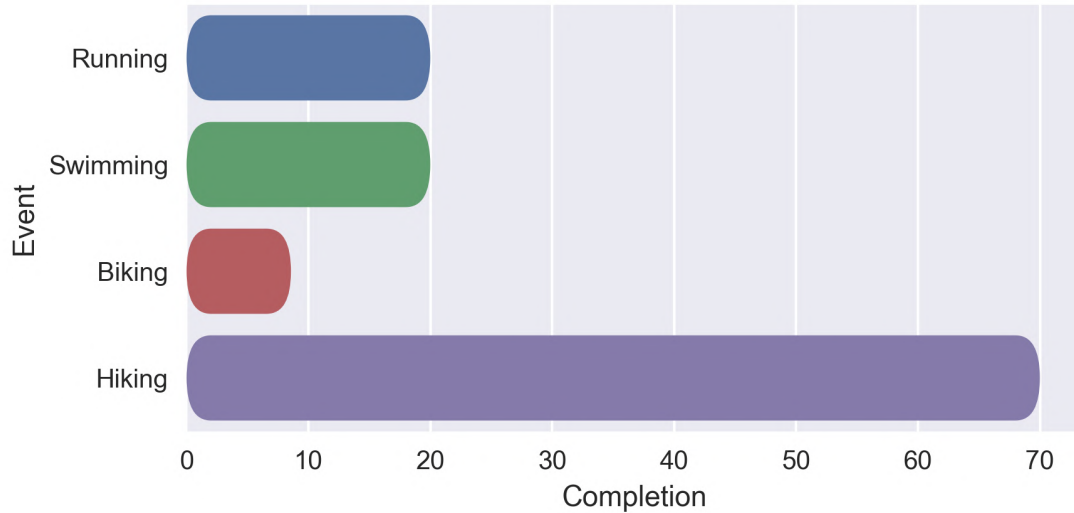
```
new_patches = []
for patch in reversed(ax.patches):
    bb = patch.get_bbox()
    color = patch.get_facecolor()
    p_bbox = FancyBboxPatch((bb.xmin, bb.ymin),
                             abs(bb.width), abs(bb.height),
                             boxstyle="round,pad=-0.0040,rounding_size=0.015",
                             ec="none", fc=color,
                             mutation_aspect=4
                             )
    patch.remove()
    new_patches.append(p_bbox)
```

```
for patch in new_patches:
    ax.add_patch(patch)
```

```
sns.despine(left=True, bottom=True)
ax.tick_params(axis=u'both', which=u'both', length=0) #Make ticklines zero length
plt.tight_layout()
```

```
ax.figure.savefig(f'Seaborn bar plot.png',dpi=150, format='png', bbox_inches='tight')
plt.show()
```

Back to
Bar Charts List



#<https://stackoverflow.com/questions/61568935/seaborn-barplot-with-rounded-corners>

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import FancyBboxPatch
```

```
plt.style.use("seaborn")
mydict = {
    'Event': ['Running', 'Swimming', 'Biking', 'Hiking'],
    'Completed': [2, 4, 3, 7],
    'Participants': [10, 20, 35, 10]}
```

```
df = pd.DataFrame(mydict).set_index('Event')
df = df.assign(Completion=(df.Completed / df.Participants) * 100)
print(df)
```

```
plt.subplots(figsize=(6, 3))
sns.set_color_codes("pastel")
ax = sns.barplot(x=df.Completion, y=df.index, joinstyle='bevel')
```

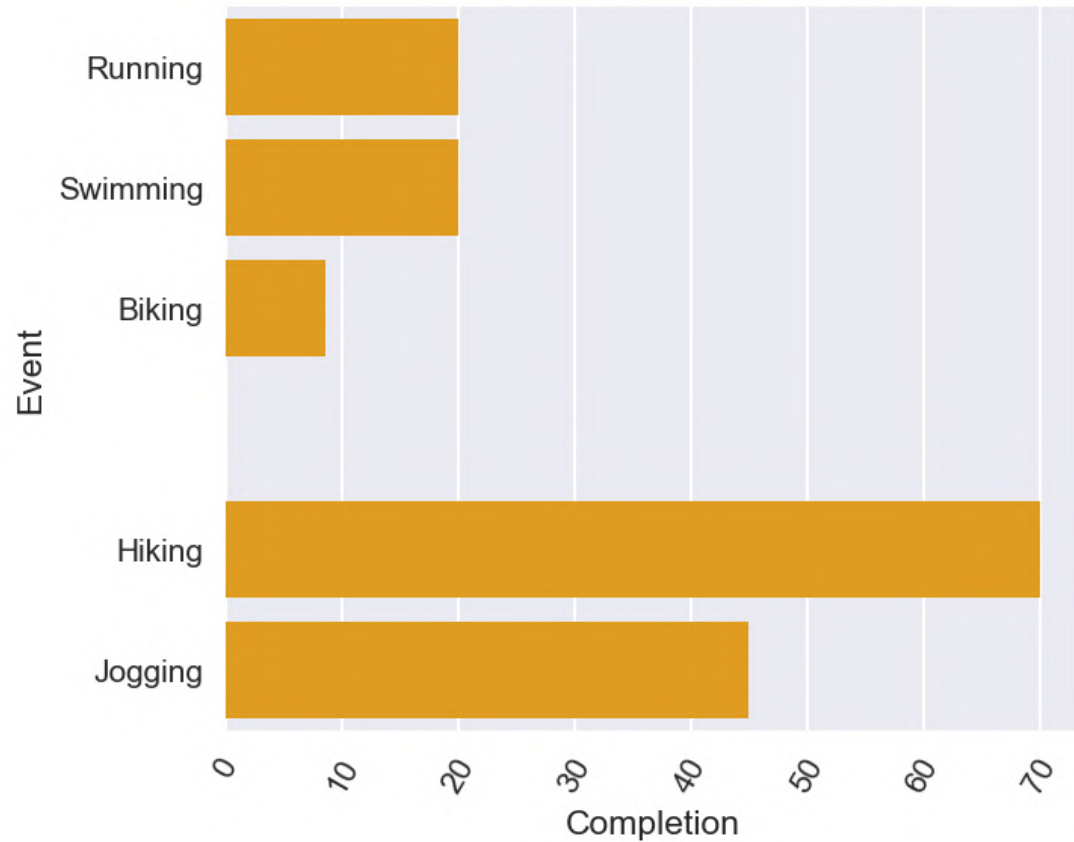
```
new_patches = []
for patch in reversed(ax.patches):
    # print(bb.xmin, bb.ymin, abs(bb.width), abs(bb.height))
    bb = patch.get_bbox()
    color = patch.get_facecolor()
    p_bbox = FancyBboxPatch((bb.xmin, bb.ymin),
                            abs(bb.width), abs(bb.height),
                            boxstyle="round, pad=-0.0040, rounding_size=2",
                            ec="none", fc=color,
                            mutation_aspect=0.2
                            )
    patch.remove()
    new_patches.append(p_bbox)
```

```
for patch in new_patches:
    ax.add_patch(patch)
```

```
sns.despine(left=True, bottom=True) #Removing all spines from plot
ax.grid(b = True, axis = 'x')
ax.tick_params(axis=u'both', which=u'both', length=0) #Make ticklines zero length
plt.tight_layout()
```

```
ax.figure.savefig(f'Seaborn_bar_plot_rounded_corners.png', dpi=300, format='png',
                 bbox_inches='tight')
plt.show()
```

Back to
Bar Charts List



[#https://stackoverflow.com/questions/61705783/matplotlib-control-spacing-between-bars](https://stackoverflow.com/questions/61705783/matplotlib-control-spacing-between-bars)
#Matplotlib Control Spacing Between Bars

```
# Hack of adding dummy row to create spacing between bars
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

mydict = {
    'Event': ['Running', 'Swimming', 'Biking', '', 'Hiking', 'Jogging'],
    'Completed': [2, 4, 3, 0, 7, 9],
    'Participants': [10, 20, 35, 0, 10, 20]}

df = pd.DataFrame(mydict).set_index('Event')
df = df.assign(Completion=(df.Completed / df.Participants) * 100)

plt.subplots(figsize=(5, 4))

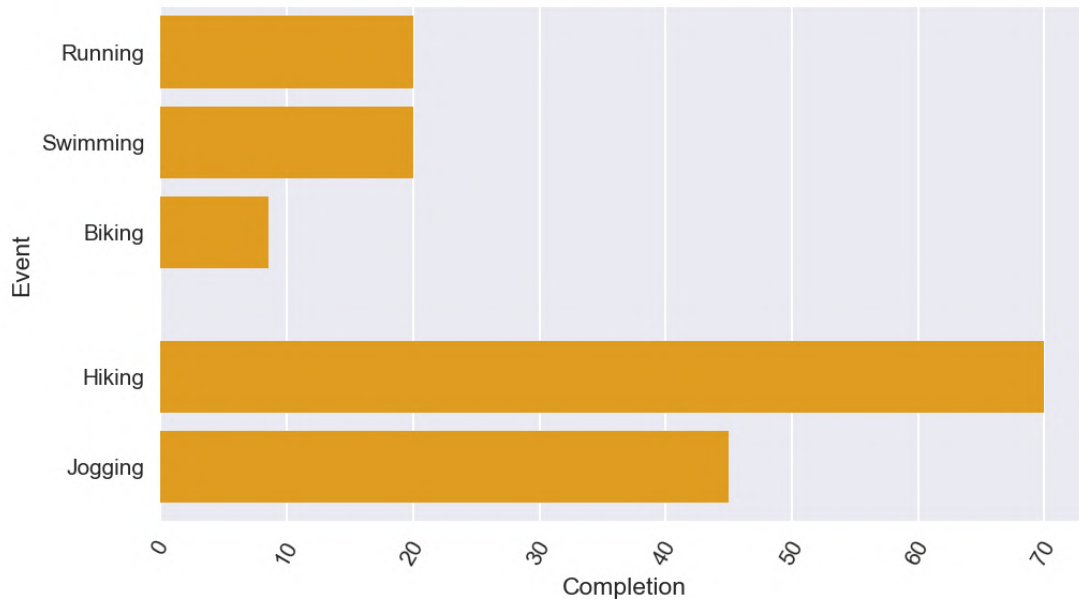
print(df.index)
ax = sns.barplot(x=df.Completion, y=df.index, color="orange", orient='h')

plt.xticks(rotation=60). #Rotating x ticks by 60

plt.tight_layout()
ax.figure.savefig('Matplotlib Spacing between bars_dummy data.png',dpi=150,
                  bbox_inches='tight')

plt.show()
```

[Back to Bar Charts List](#)



#<https://stackoverflow.com/questions/61705783/matplotlib-control-spacing-between-bars>

#Matplotlib Control Spacing Between Bars

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker
```

```
mydict = {
    'Event': ['Running', 'Swimming', 'Biking', 'Hiking', 'Jogging'],
    'Completed': [2, 4, 3, 7, 9],
    'Participants': [10, 20, 35, 10, 20]}
df = pd.DataFrame(mydict).set_index('Event')
df = df.assign(Completion=(df.Completed / df.Participants) * 100)
```

#note: no more blank row

```
plt.style.use('seaborn')
```

#define a function to add space starting a specific label

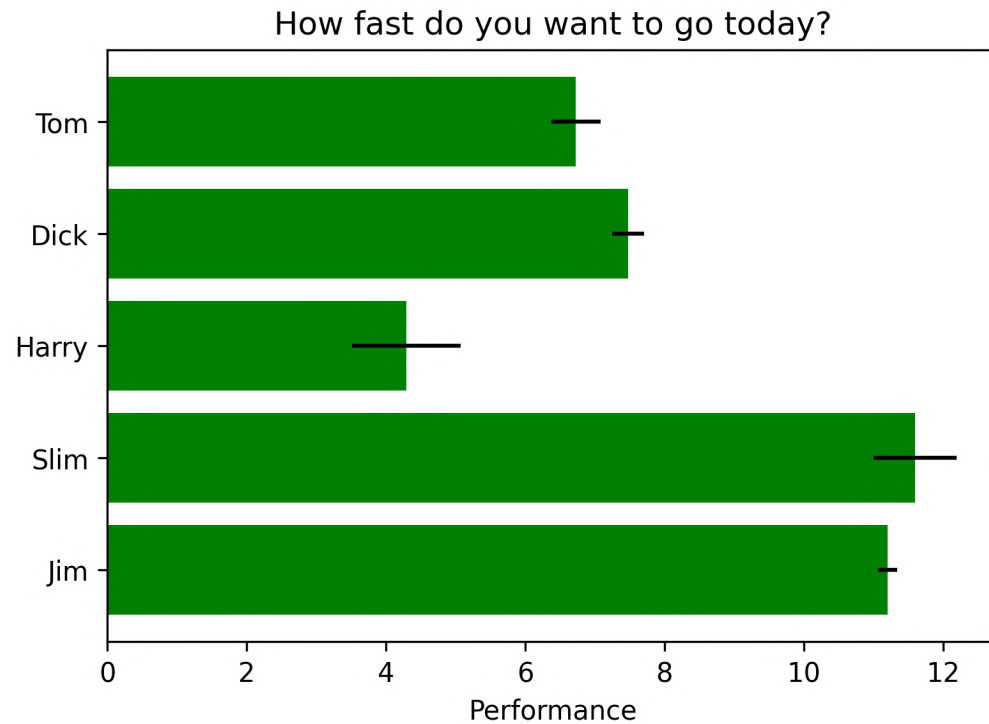
```
def add_space_after(ax, label_shift='', extra_space=0):
    bool_space = False
    label_list = [t.get_text() for t in ax.get_yticklabels()]
    # get position of current ticks
    ticks_position = np.array(ax.get_yticks()).astype(float)
    # iterate over the boxes/ticks/tick label and change the y position
    for i, (patch, label) in enumerate(zip(ax.patches, ax.get_yticklabels())):
        # if the label to start the shift found
        if label.get_text() == label_shift: bool_space = True
        # reposition the boxes and the labels afterward
        if bool_space:
            patch.set_y(patch.get_y() + extra_space)
            ticks_position[i] += extra_space
    # in the case where the spacing is needed
    if bool_space:
        print(f'ytick positions after: {ax.get_yticks()} \n tick labels after: {ax.get_yticklabels()}')
        #ax.yaxis.set_major_locator(ticker.FixedLocator(ticks_position))
        ax.set_yticks(ticks_position)

        ax.set_ylim([ax.get_ylim()[0]+extra_space, ax.get_ylim()[1]])
        print(f'ytick positions after: {ax.get_yticks()} \n tick labels after: {ax.get_yticklabels()}')
        ax.yaxis.set_major_formatter(ticker.FixedFormatter(label_list))
```

```
fig, ax = plt.subplots(figsize = (7,4))
sns.barplot(x=df.Completion, y=df.index, color="orange", ax = ax, orient='h')
plt.xticks(rotation=60) #Rotating x ticks by 60
plt.tight_layout()
```

#use the function

```
add_space_after(ax, 'Hiking', 0.6)
ax.figure.savefig(f'Matplotlib Spacing between bars.png', dpi=150, bbox_inches='tight')
```



#<https://stackoverflow.com/questions/49335512/ticklabels-inside-a-plot-in-matplotlib>
#<https://stackoverflow.com/questions/14800973/matplotlib-tick-labels-position-relative-to-axes>
#https://matplotlib.org/stable/gallery/lines_bars_and_markers/barh.html

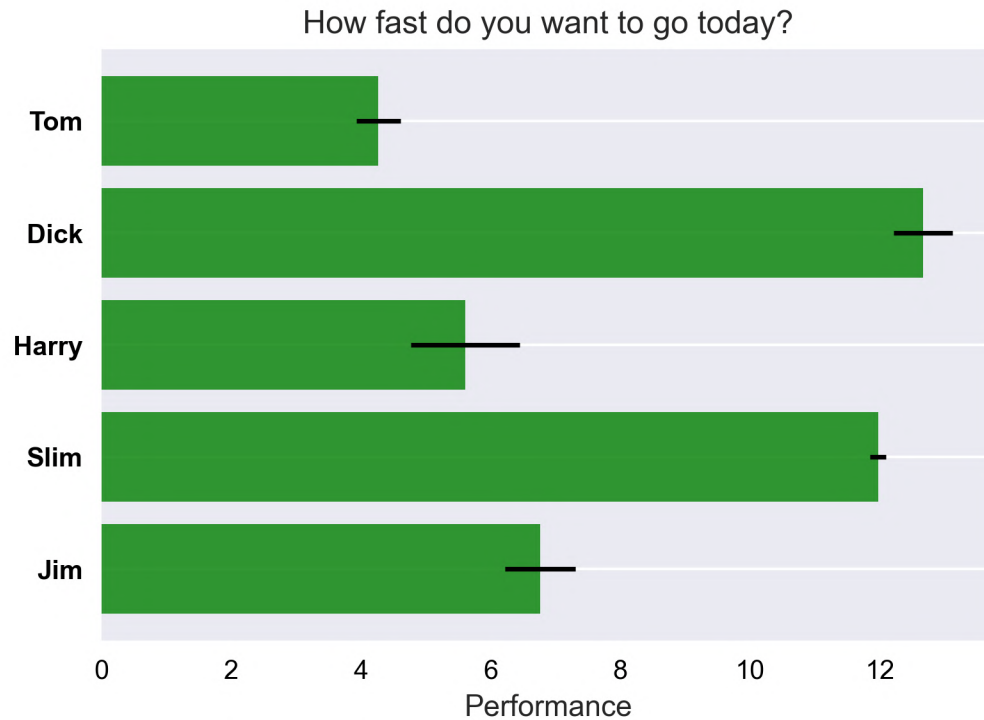
```
plt.rcParams()
fig, ax = plt.subplots()

# Example data
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))

ax.barh(y_pos, performance, xerr=error, align='center', color='green', ecolor='black')
ax.set_yticks(y_pos)
ax.set_yticklabels(people)
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Performance')
ax.set_title('How fast do you want to go today?')

fig.savefig('BasicHorizontalBarplot.png', dpi = 300, bbox_inches='tight')
plt.show()
```

[Back to
Bar Charts List](#)



Note on Practical Implementation of tick related methods

Back to
Bar Charts List

#<https://stackoverflow.com/questions/49335512/ticklabels-inside-a-plot-in-matplotlib>
#<https://stackoverflow.com/questions/14800973/matplotlib-tick-labels-position-relative-to-axes>
#https://matplotlib.org/stable/gallery/lines_bars_and_markers/barh.html

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
np.random.seed(123456) # Fixing random state for reproducibility
plt.style.use('seaborn')
```

```
fig, ax = plt.subplots(figsize = (6,4))
```

```
# Example data
```

```
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
```

```
y_pos = np.arange(len(people))
```

```
performance = 3 + 10 * np.random.rand(len(people))
```

```
error = np.random.rand(len(people))
```

```
ax.barh(y_pos, performance, xerr=error, align='center', color='green',alpha = 0.8, ecolor='black')
```

```
ax.set_yticks(y_pos)
```

```
ax.set_yticklabels(people, ha = 'right')
```

```
ax.invert_yaxis() # labels read top-to-bottom
```

```
ax.set_xlabel('Performance')
```

```
ax.set_title('How fast do you want to go today?')
```

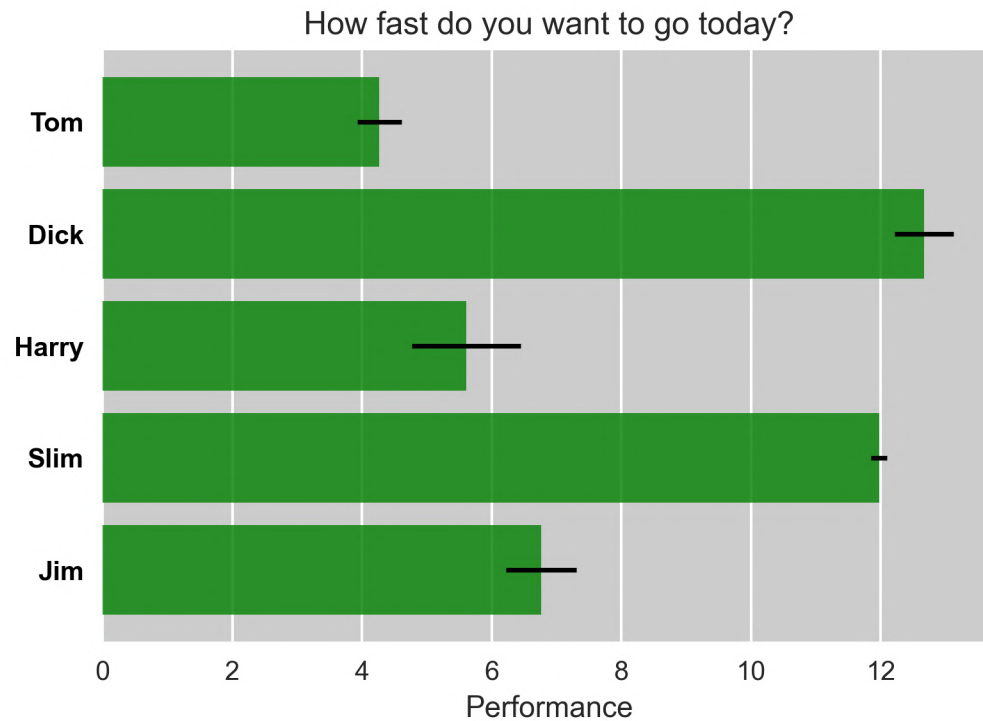
```
[tkl.set(fontweight = 'bold') for tkl in ax.get_yticklabels()] #Making tick labels bold
```

```
ax.tick_params(axis = 'both', labelcolor = 'black', labelsize = 10) #Setting ticklabel size and color
```

```
ax.grid(axis = 'y', b = True). #Setting on gridlines from y axis
```

```
ax.grid(axis = 'x', b = False) #Setting off gridlines from x axis
```

```
fig.savefig('SeabornHorizontalBarplot_textbold.png', dpi = 300, bbox_inches='tight')
plt.show()
```



#Creating a User Defined function

```
def stylize_axes(ax):
    ax.set_facecolor('.8')
    ax.tick_params(labelsize=10, length=0)
    ax.grid(True, axis='x', color='white')
    ax.set_axisbelow(True) #Ticks and gridlines below other artists
    [spine.set_visible(False) for spine in ax.spines.values()]
```

#<https://stackoverflow.com/questions/49335512/ticklabels-inside-a-plot-in-matplotlib>
 #<https://stackoverflow.com/questions/14800973/matplotlib-tick-labels-position-relative-to-axes>
 #https://matplotlib.org/stable/gallery/lines_bars_and_markers/barh.html

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
np.random.seed(123456) # Fixing random state for reproducibility
plt.style.use('seaborn')
```

```
fig, ax = plt.subplots(figsize = (6,4))
```

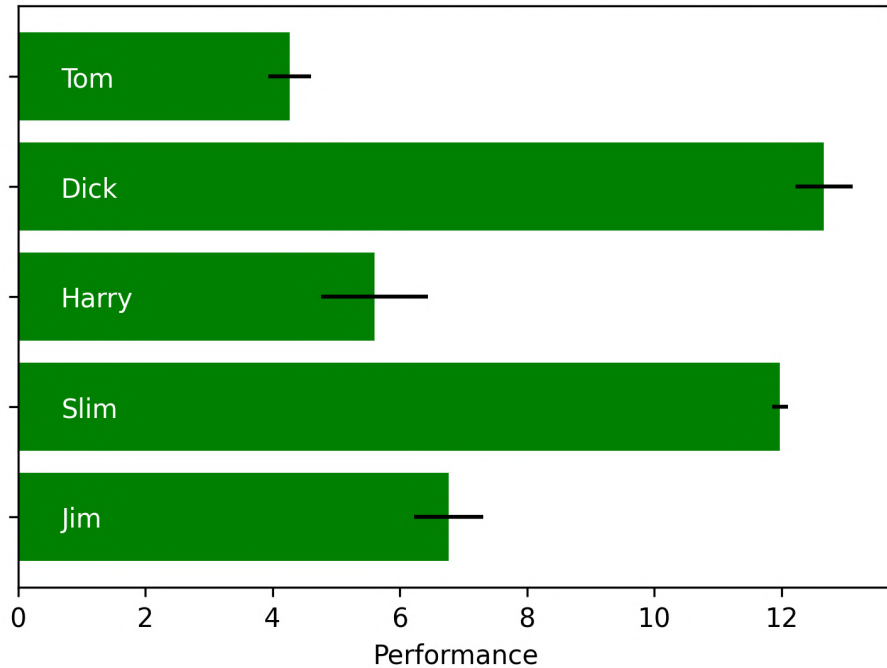
```
# Example data
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
```

```
ax.barh(y_pos, performance, xerr=error, align='center', color='green',alpha = 0.8, ecolor='black')
ax.set_yticks(y_pos)
ax.set_yticklabels(people, ha = 'right')
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Performance')
ax.set_title('How fast do you want to go today?')
[tkl.set(fontweight = 'semibold') for tkl in ax.get_yticklabels()] #Making ticklabels semi bold
ax.tick_params(axis = 'y', labelcolor = 'black') #Setting ticklabel size and color
```

```
ax.grid(axis = 'y', b = False)
stylize_axes(ax). # Using the user defined function stylize_axes
```

```
fig.savefig('SeabornHorizontalBarplot_textbold.png', dpi = 300, bbox_inches='tight')
plt.show()
```

How fast do you want to go today?



- https://matplotlib.org/stable/gallery/misc/zorder_demo.html?highlight=zorder
- https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_axisbelow.html?highlight=axisbelow#matplotlib.axes.Axes.set_axisbelow
- <https://stackoverflow.com/questions/48327099/is-it-possible-to-set-higher-z-index-of-the-axis-labels>

Axes.set_axisbelow(self, b) : Set whether axis ticks and gridlines are above or below most artists. This controls the zorder of the ticks and gridlines.

#<https://stackoverflow.com/questions/49335512/ticklabels-inside-a-plot-in-matplotlib>
 #<https://stackoverflow.com/questions/14800973/matplotlib-tick-labels-position-relative-to-axes>
 #https://matplotlib.org/stable/gallery/lines_bars_and_markers/barh.html

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
np.random.seed(123456)
plt.rcParams()
```

```
fig, ax = plt.subplots(figsize = (6,4))
```

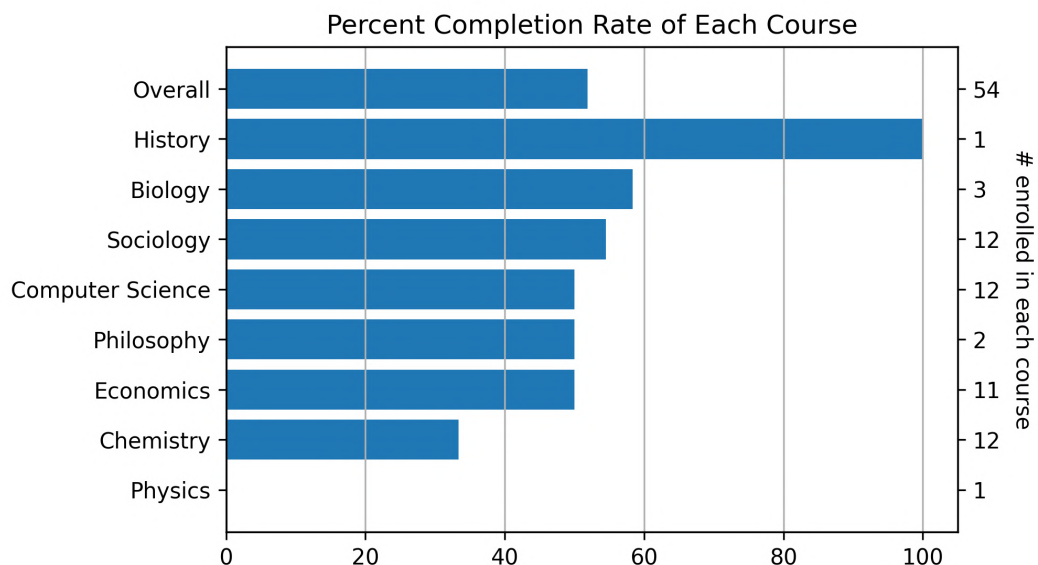
```
# Example data
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
```

```
plt.rcParams["axes.axisbelow"] = 'line' #tick and gridlines above other artists except lines
#Can use the below line as well instead of above
#ax.set_axisbelow(False)
```

```
ax.barh(y_pos, performance, xerr=error, align='center', color='green', ecolor='black')
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Performance')
ax.set_title('How fast do you want to go today?')
```

```
ax.set_yticks(y_pos)
ax.set_yticklabels(people, horizontalalignment = 'left', fontweight = 'medium') #ha is important
ax.tick_params(axis = 'y', pad = -20, labelcolor = 'white') #pad as -20 for ticklabels inside
```

```
plt.show()
```



Creating alternative y axis labels for a horizontal bar plot with matplotlib

- Axes.twinx
- (grid lines above bars)

#<https://stackoverflow.com/questions/52171321/>

[creating-alternative-y-axis-labels-for-a-horizontal-bar-plot-with-matplotlib/52171704](https://stackoverflow.com/questions/52171321/creating-alternative-y-axis-labels-for-a-horizontal-bar-plot-with-matplotlib/52171704)

#Check answer by [Sheldore](#)

```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 12})

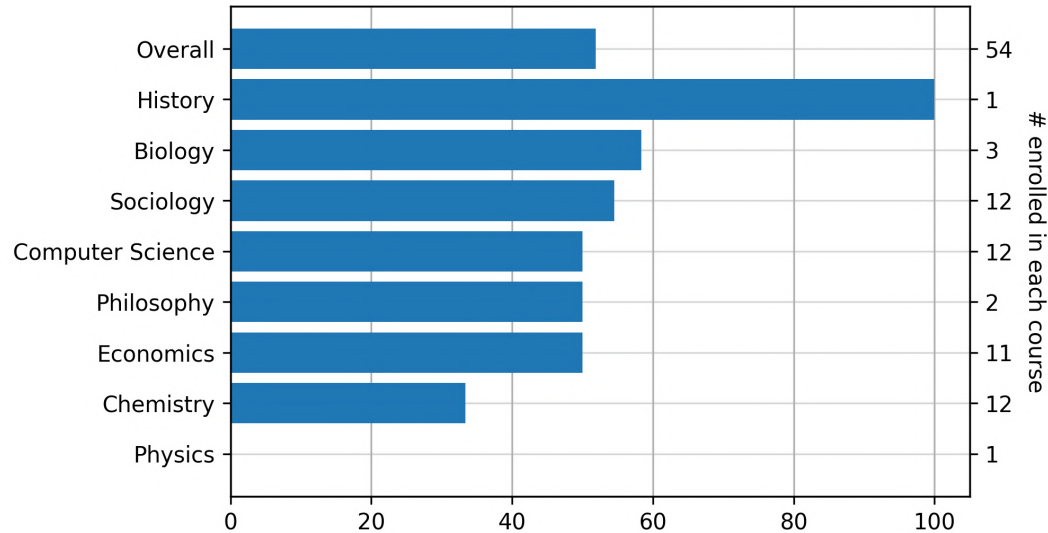
fig, ax = plt.subplots(figsize = (6,4))
y = ['Physics','Chemistry','Economics','Philosophy','Computer Science','Sociology','Biology','History','Overall']
x = [0.0,33.333333333333336,50.0,50.0,50.0,54.54545454545455,58.333333333333336,100.0,51.851851851851855]
alternativeYlabels = ['54', '1', '3', '12', '12', '2', '11', '12', '1']
plt.barh(y,x)
plt.title('Percent Completion Rate of Each Course')
ax = plt.gca()
ax.xaxis.grid(True) # adding vertical grid lines
ax.set_axisbelow('line') #Setting Ticks and gridlines above all Artists except lines
```

Creating the right hand side y axis

```
ax2 = ax.twinx()
ax_lim = ax.get_ylim()
ax2_lim = (ax_lim[0], ax_lim[1]) # Aligning the limits of both y axes
ax2.set_ylim(ax2_lim)
ax2.set_yticks(range(0, len(y)))
ax2.set_yticklabels(alternativeYlabels[::-1]) # Reverses the list
ax2.set_ylabel("# enrolled in each course",rotation = 270, labelpad=15)
```

```
fig.savefig('SecondaryAxisBarplot_2.png', dpi = 300, bbox_inches='tight')
plt.show()
```

Percent Completion Rate of Each Course



Creating alternative y axis labels for a horizontal bar plot with matplotlib

- Axes.twinx
- (grid lines below bars)

#<https://stackoverflow.com/questions/52171321/>

#[creating-alternative-y-axis-labels-for-a-horizontal-bar-plot-with-matplotlib/52171704](https://stackoverflow.com/questions/52171321/creating-alternative-y-axis-labels-for-a-horizontal-bar-plot-with-matplotlib/52171704)

#Check answer by [Sheldore](#)

```
import matplotlib.pyplot as plt
plt.rcParams()
```

```
fig, ax = plt.subplots(figsize = (6,4))
```

```
y = ['Physics','Chemistry','Economics','Philosophy','Computer Science','Sociology','Biology','History','Overall']
```

```
x = [0.0,33.333333333333336,50.0,50.0,50.0,54.54545454545455,58.333333333333336,100.0,51.851851851851855]
```

```
alternativeYlabels = ['54', '1', '3', '12', '12', '2', '11', '12', '1']
```

```
plt.barh(y,x)
```

```
plt.title('Percent Completion Rate of Each Course')
```

```
ax = plt.gca()
```

```
ax.xaxis.grid(True) # adding vertical grid lines
```

```
ax.yaxis.grid(True, alpha = 0.5) # adding horizontal grid lines
```

```
ax.set_axisbelow(True) #Setting Ticks and gridlines below all Artists.
```

```
# Creating the right hand side y axis
```

```
ax2 = ax.twinx()
```

```
ax_lim = ax.get_ylim()
```

```
ax2_lim = (ax_lim[0], ax_lim[1]) # Aligning the limits of both y axes
```

```
ax2.set_ylim(ax2_lim)
```

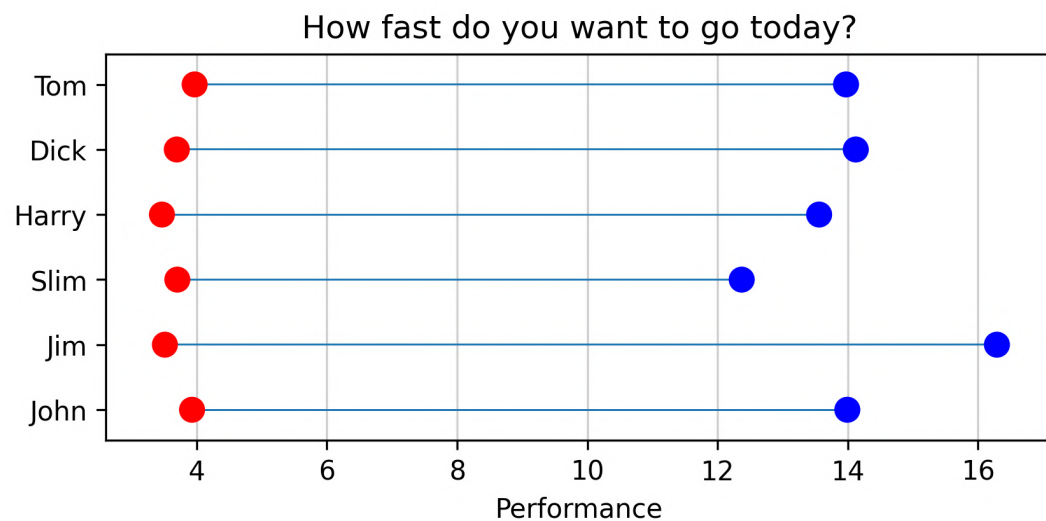
```
ax2.set_yticks(range(0, len(y)))
```

```
ax2.set_yticklabels(alternativeYlabels[::-1]) # Reverses the list
```

```
ax2.set_ylabel("# enrolled in each course",rotation = 270, labelpad=15)
```

```
fig.savefig('SecondaryAxisBarplot_2.png', dpi = 300, bbox_inches='tight')
```

```
plt.show()
```



```
plt.rcdefaults()
```

Cleveland Dot plots Using Circle Patches



Using seaborn plot style, code line :
`plt.style.use('seaborn')`

https://www.reddit.com/r/matplotlib/comments/i3t5uv/any_example_code_or_tips_for_how_to_make_this/

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)
```

```
plt.rcdefaults()
fig, ax = plt.subplots()
```

```
# Example data
```

```
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim', 'John')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
start = 3 + np.random.rand(len(people))
```

```
ax.barh(y_pos, performance, left=start, height=0.03, align='center')
```

```
ax.set_yticks(y_pos)
```

```
ax.set_yticklabels(people)
```

```
ax.invert_yaxis() # labels read top-to-bottom
```

```
ax.set_xlabel('Performance')
```

```
ax.set_title('How fast do you want to go today?')
```

```
ax.xaxis.grid(b=True, alpha = 0.6)
```

```
ax.set_axisbelow(True). #Tick and Gridlines below other artists
```

```
for x,y,x1 in zip(start, y_pos, performance):
```

```
    circ = mpatches.Circle((x,y),0.2,fc='red')
```

#Adding red circles for start position

```
    ax.add_patch(circ)
```

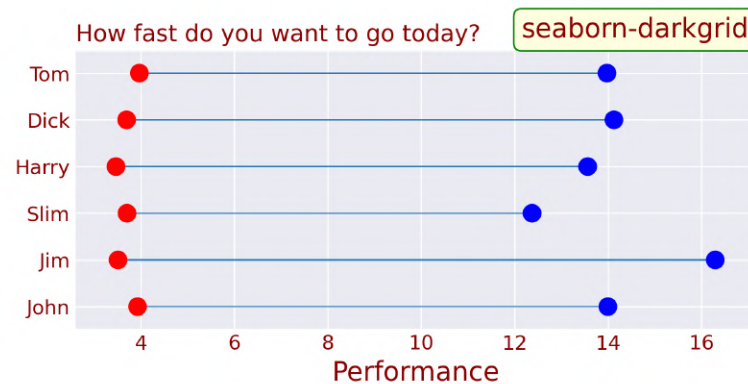
```
    circ = mpatches.Circle((x+x1,y),0.2,fc='blue')
```

#Adding blue circles for start position

```
    ax.add_patch(circ)
```

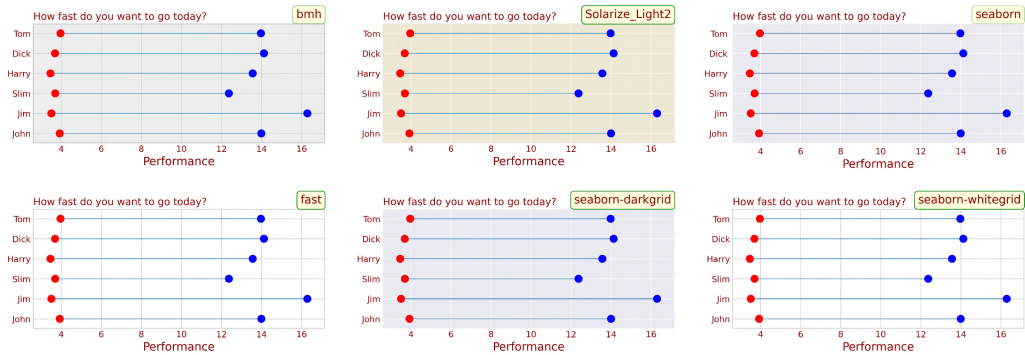
```
plt.axis('scaled')
```

```
fig.savefig('Different Styles_Dot_Plot.png', dpi = 300, format = 'png',bbox_inches='tight')
plt.show()
```

Cleveland Dot plots

Using Circle Patches
Multiple Plot Styles



#https://www.reddit.com/r/matplotlib/comments/i3t5uv/any_example_code_or_tips_for_how_to_make_this/

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
import numpy as np
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)
```

```
#Listing the styles for plotting with for loop
styles = ['bmh', 'Solarize_Light2','seaborn', 'fast',
          'seaborn-darkgrid','seaborn-whitegrid']
```

```
#Creating font dicts for different use cases
font_text = {'family': 'EB Garamond',
             'color': 'darkred', 'weight': 'normal', 'size': 16} #for label plot style
```

```
font_title = {'family': 'Dejavu Sans',
              'color': 'darkred', 'weight': 'normal', 'size': 14} #for axes title
```

```
font_tick = {'family': 'Dejavu Sans',
             'color': 'darkred', 'weight': 'normal', 'size': 12} #for tick labels
```

Example data

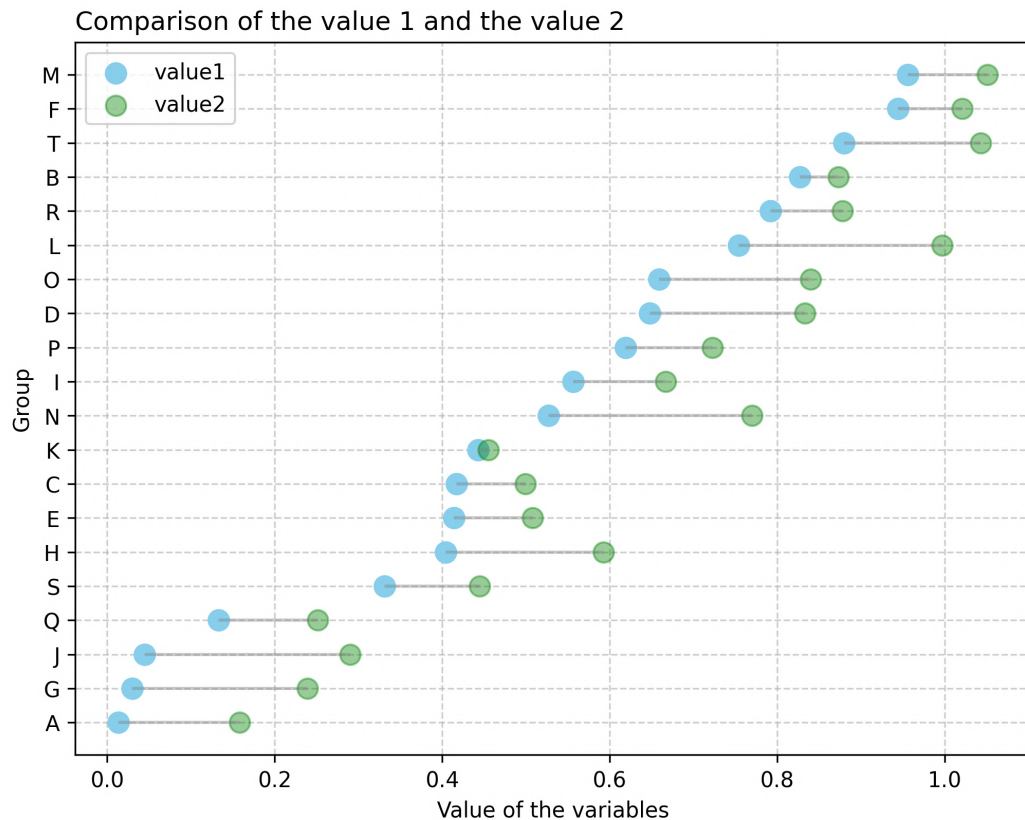
```
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim','John')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
start = 3+np.random.rand(len(people))
```

```
fig = plt.figure(figsize = (25,9))
```

```
for i,ax in enumerate(styles):
    with plt.style.context(styles[i]):
        print(styles[i])
        ax = fig.add_subplot(2,3,i+1)
        ax.barh(y_pos, performance, left=start, height=0.03, align='center')
        ax.set_yticks(y_pos)
        ax.set_yticklabels(people,**font_tick) #Setting yticklabel props using ** unpacking
        [tkl.set(**font_tick) for tkl in ax.get_xticklabels()] #Setting xticklabel props using ** unpacking
        ax.invert_yaxis() # labels read top-to-bottom
        ax.set_xlabel('Performance', **font_text)
        ax.set_title('How fast do you want to go today?', loc = 'left', fontdict = font_title)
        #ax.grid(b=True)
        for x,y,x1 in zip(start, y_pos, performance): #Adding Circle Patches
            circ = Circle((x,y),0.2,fc='red', transform = ax.transData)
            ax.add_patch(circ)
            circ = Circle((x+x1,y),0.2,fc='blue',transform = ax.transData)
            ax.add_patch(circ)
            ax.axis('scaled')

        #Adding name of the plot style for the subplot within fancy bbox
        ax.text(0.99,1.05,s = f'{styles[i]}',ha = 'right',transform = ax.transAxes, **font_text,
              bbox=dict(facecolor='lightyellow', edgecolor='green', boxstyle='round',pad =.3))
```

```
plt.subplots_adjust(wspace = .2, hspace = .1) #Adjust horizontal & vertical space btw subplots
fig.savefig('Different Styles_Dot_Plot_fancybox_text.png', dpi = 300, bbox_inches='tight')
```



#<https://www.python-graph-gallery.com/184-lollipop-plot-with-2-groups>

Changes done in the code in link :

- Changed pyplot interface style to OO style
- Configured the grid lines to dashed line style, set_axisbelow to put ticks/gridlines below other artists
- Changed the scatter marker size to 90

Cleveland
Dot plots

- Using Axes Methods
- Axes.Scatter and Axes.hlines

```
# libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams()
# Create a dataframe
value1=np.random.uniform(size=20)
value2=value1+np.random.uniform(size=20)/4
df = pd.DataFrame({'group':list(map(chr, range(65, 85))), 'value1':value1 , 'value2':value2 })

# Reorder it following the values of the first value:
ordered_df = df.sort_values(by='value1')
my_range=range(1,len(df.index)+1)

fig, ax = plt.subplots(figsize = (8,6))
# The horizontal plot is made using the hline function
ax.hlines(y=my_range, xmin=ordered_df['value1'], xmax=ordered_df['value2'], color='grey',
         alpha=0.4)
ax.scatter(ordered_df['value1'], my_range, color='skyblue', alpha=1, label='value1', s = 90)
ax.scatter(ordered_df['value2'], my_range, color='green', alpha=0.4 , label='value2', s = 90)
ax.legend()

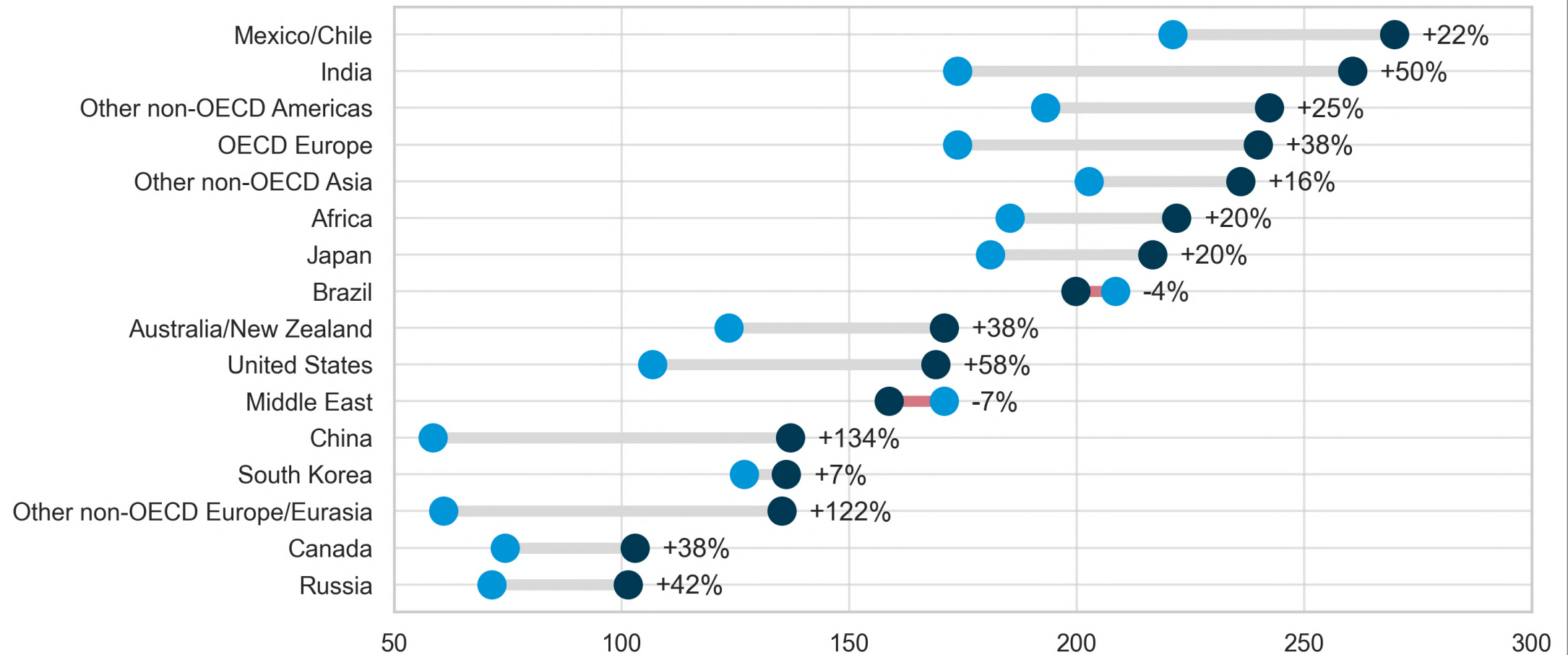
# Add title and axis names
ax.set_yticks(my_range)
ax.set_yticklabels(ordered_df['group'])
ax.set_title("Comparison of the value 1 and the value 2", loc='left')
ax.set_xlabel('Value of the variables')
ax.set_ylabel('Group')

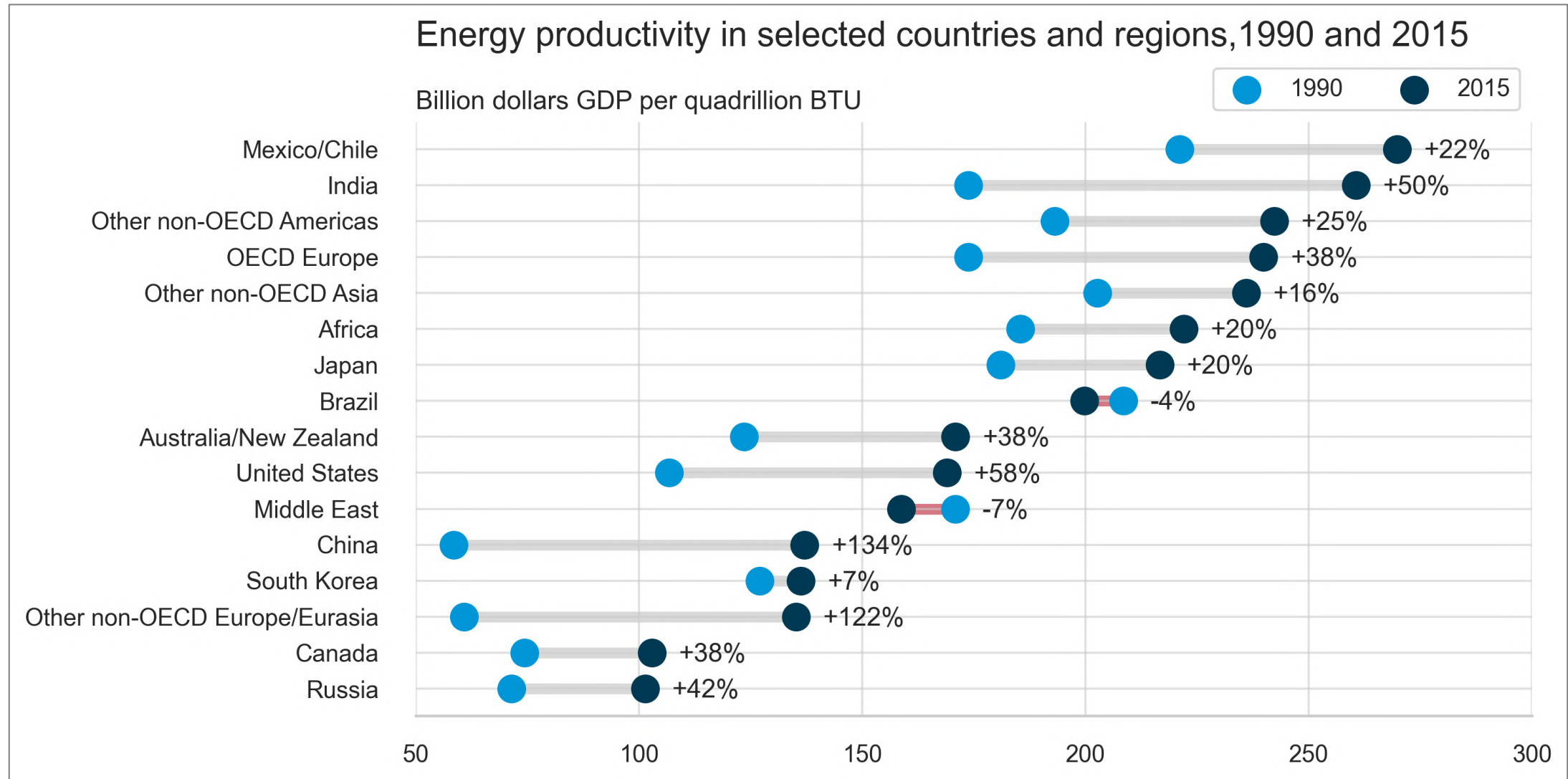
# Configuring the grid lines
ax.grid(True, alpha = 0.6, ls = 'dashed')
ax.set_axisbelow(True)
fig.savefig('lollipop_charts.png', dpi = 300, format = 'png',bbox_inches='tight')
plt.show() #Show the graph
```

Energy productivity in selected countries and regions, 1990 and 2015

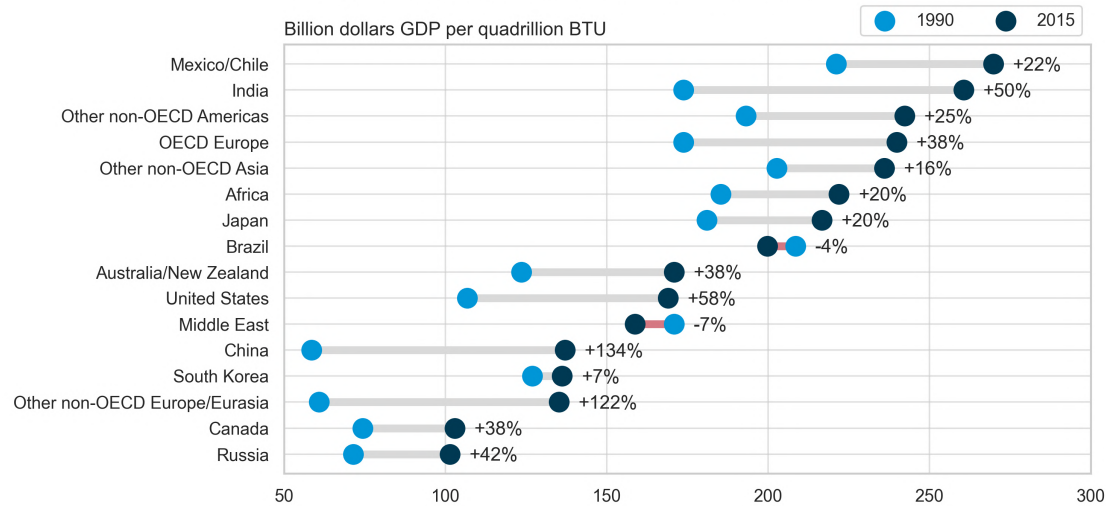
Billion dollars GDP per quadrillion BTU

● 1990 ● 2015





Energy productivity in selected countries and regions, 1990 and 2015



#<https://stats.stackexchange.com/questions/423735/what-is-the-name-of-this-plot-that-has-rows-with-two-connected-dots/423861>

#<http://thewhyaxis.info/gap-remake/> #Dot plots in Excel

#<https://www.eia.gov/outlooks/ieo/pdf/0484%282016%29.pdf>

Cleveland Dot plots

- Using Axes Methods
- Axes.Scatter and Axes.hlines

Code snippet 1 of 2

libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
import seaborn as sns
import io

sns.set(style="whitegrid") # set style
plt.style.use('seaborn-whitegrid')
```

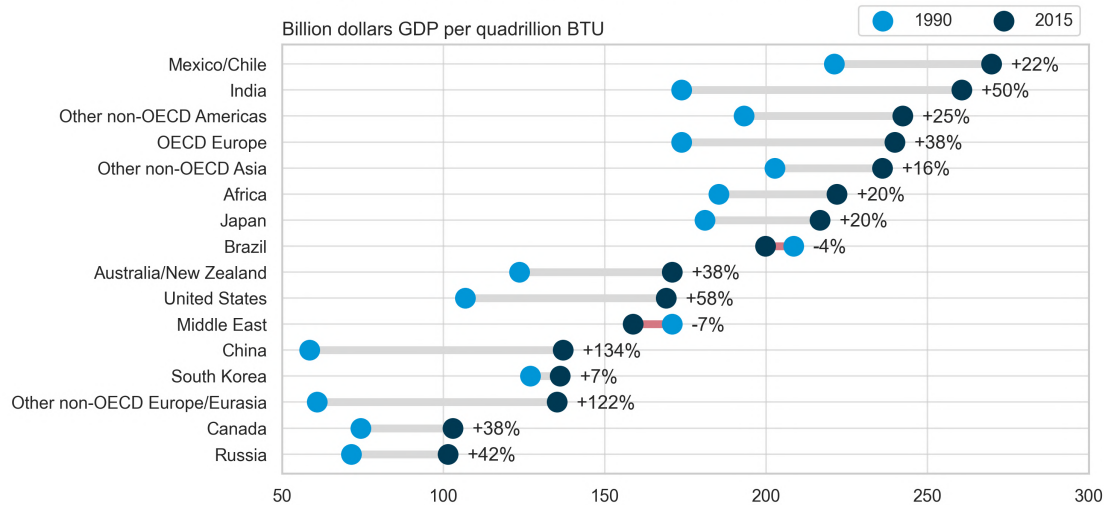
```
data = io.StringIO("""Country" 1990 2015
"Russia" 71.5 101.4
"Canada" 74.4 102.9
"Other non-OECD Europe/Eurasia" 60.9 135.2
"South Korea" 127.0 136.2
"China" 58.5 137.1
"Middle East" 170.9 158.8
"United States" 106.8 169.0
"Australia/New Zealand" 123.6 170.9
"Brazil" 208.5 199.8
"Japan" 181.0 216.7
"Africa" 185.4 222.0
"Other non-OECD Asia" 202.7 236.0
"OECD Europe" 173.8 239.9
"Other non-OECD Americas" 193.1 242.3
"India" 173.8 260.6
"Mexico/Chile" 221.1 269.8""")
```

```
df = pd.read_csv(data, sep="\s+", quotechar='"')
df = df.set_index("Country").sort_values("2015")
df["change"] = df["2015"] / df["1990"] - 1
```

```
font_suptitle = {'fontsize': 16}
```

Code continues to Next slide

Energy productivity in selected countries and regions, 1990 and 2015



#<https://stats.stackexchange.com/questions/423735/what-is-the-name-of-this-plot-that-has-rows-with-two-connected-dots/423861>

#<http://thewhyaxis.info/gap-remake/> #Dot plots in Excel

#<https://www.eia.gov/outlooks/ieo/pdf/0484%282016%29.pdf>

Cleveland Dot plots

- Using Axes Methods
- Axes.Scatter and Axes.hlines

Code snippet 2 of 2

```
fig = plt.figure(figsize=(13,5))
ax = fig.add_subplot()
y_range = np.arange(1, len(df.index) + 1)
colors = np.where(df['2015'] > df['1990'], '#d9d9d9', '#d57883')
ax.hlines(y=y_range, xmin=df['1990'], xmax=df['2015'],
         color=colors, lw=5)
ax.scatter(df['1990'], y_range, color='#0096d7', s=150, label='1990', zorder=3)
ax.scatter(df['2015'], y_range, color='#003953', s=150, label='2015', zorder=3)
for (_, row), y in zip(df.iterrows(), y_range):
    ax.annotate(f"{row['change']:+.0%}", (max(row["1990"], row["2015"]) + 6, y - 0.25))
```

```
ax.grid(True,alpha = 0.6) #Make grid lines lighter
ax.set_axisbelow(False)
```

```
# Make all spines visible
ax.spines['right'].set(visible = True)
ax.spines['top'].set(visible = True)
ax.spines['left'].set(visible = True)
ax.spines['right'].set(visible = True)
```

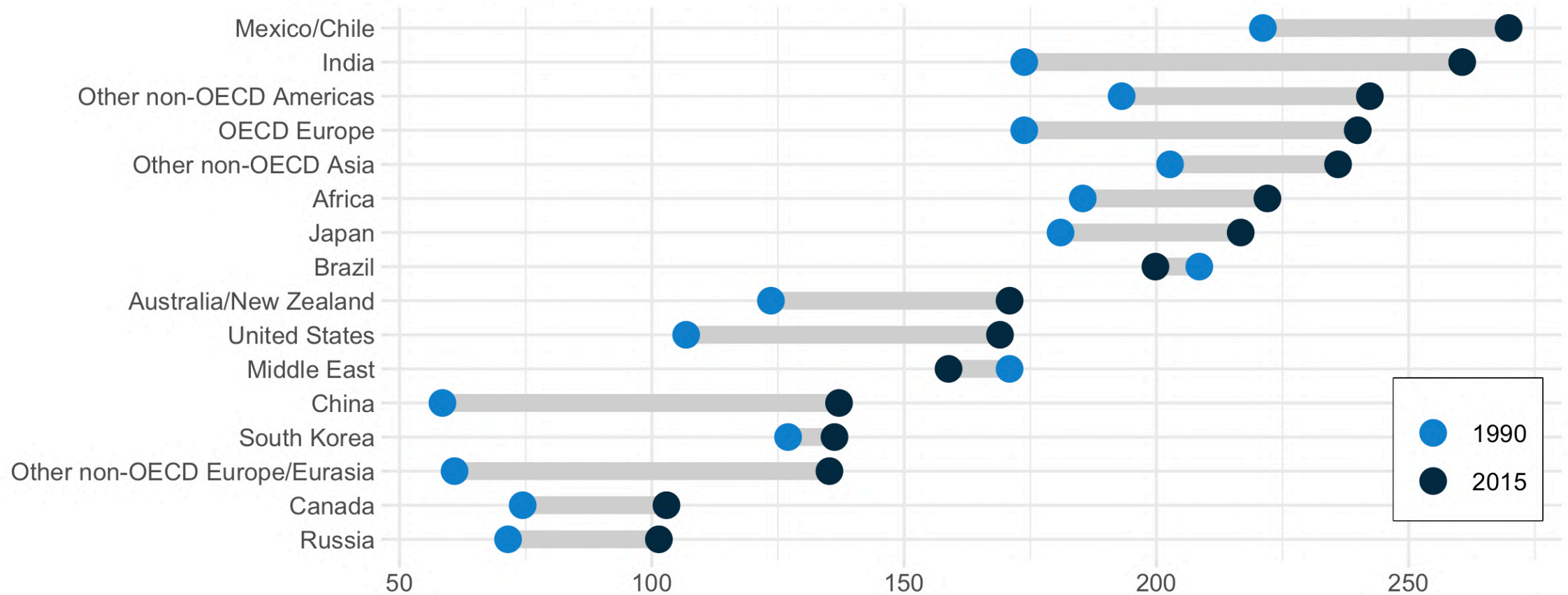
```
ax.set_yticks(y_range)
ax.set_yticklabels(df.index)
```

```
ax.legend(ncol=2, bbox_to_anchor=(1, 1), loc="lower right", frameon=True)
# the x coords of this transformation are data, and the y coord are figure
trans = transforms.blended_transform_factory(ax.transData, fig.transFigure)
fig.suptitle("Energy productivity in selected countries and regions, 1990 and 2015", **font_suptitle,
            x = 50, y = 1, transform = trans,
            ha = 'left', va = 'top')
ax.set_title("Billion dollars GDP per quadrillion BTU", loc = 'left')
ax.set_xlim(50, 300)
```

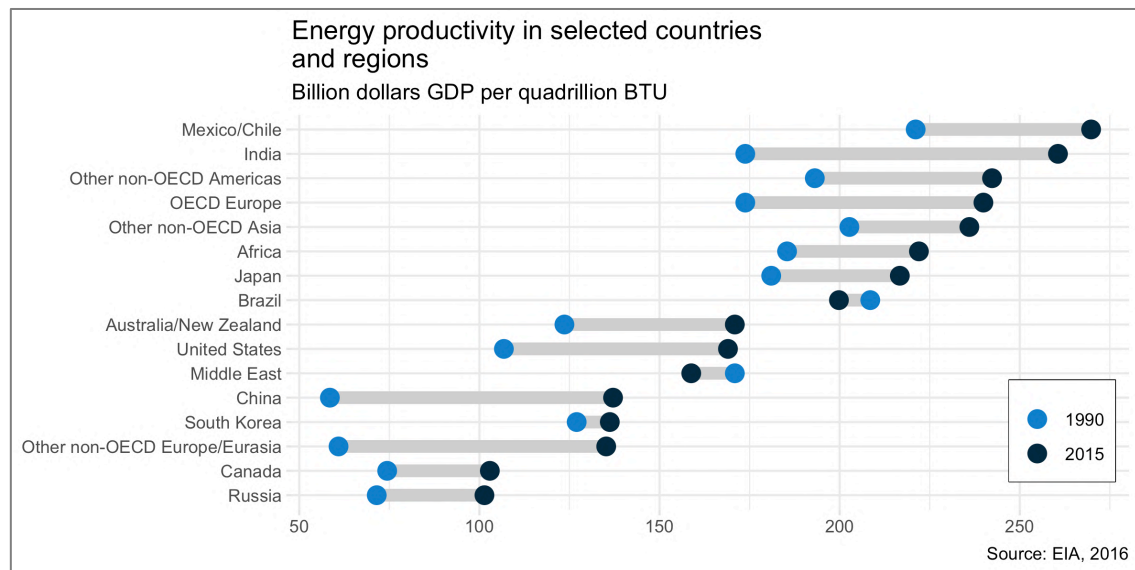
```
fig.subplots_adjust(left=0.35, top = 0.87)
fig.savefig('connected dot plots_sample2.png', dpi = 300, format = 'png', bbox_inches='tight')
plt.show()
```

Energy productivity in selected countries and regions

Billion dollars GDP per quadrillion BTU



Source: EIA, 2016



#<https://stats.stackexchange.com/questions/423735/what-is-the-name-of-this-plot-that-has-rows-with-two-connected-dots/423861>

```
library(dplyr) # for data manipulation
library(tidyr) # for reshaping the data frame
library(stringr) # string manipulation
library(ggplot2) # graphing
```

Cleveland Dot plots

- Using ggplot2 in R

```
# create the data frame # (in wide format, as needed for the line segments):
dat_wide = tibble::tribble( ~Country, ~Y1990, ~Y2015, 'Russia', 71.5, 101.4,
'Canada', 74.4, 102.9, 'Other non-OECD Europe/Eurasia', 60.9, 135.2, 'South Korea',
127, 136.2, 'China', 58.5, 137.1, 'Middle East', 170.9, 158.8, 'United States', 106.8,
169, 'Australia/New Zealand', 123.6, 170.9, 'Brazil', 208.5, 199.8, 'Japan', 181, 216.7,
'Africa', 185.4, 222, 'Other non-OECD Asia', 202.7, 236, 'OECD Europe', 173.8,
239.9, 'Other non-OECD Americas', 193.1, 242.3, 'India', 173.8, 260.6,
'Mexico/Chile', 221.1, 269.8 )
```

```
# a version reshaped to long format (for the points):
```

```
dat_long = dat_wide %>%
```

```
gather(key = 'Year', value = 'Energy_productivity', Y1990:Y2015) %>%
```

```
mutate(Year = str_replace(Year, 'Y', ''))
```

```
# create the graph:
```

```
ggplot() +
```

```
geom_segment(data = dat_wide, aes(x = Y1990, xend = Y2015, y = reorder(Country,
Y2015), yend = reorder(Country, Y2015)), size = 3, colour = '#D0D0D0') +
```

```
geom_point(data = dat_long, aes(x = Energy_productivity, y = Country, colour =
Year), size = 4) +
```

```
labs(title = 'Energy productivity in selected countries \nand regions', subtitle =
'Billion dollars GDP per quadrillion BTU', caption = 'Source: EIA, 2016', x = NULL, y
= NULL) +
```

```
scale_colour_manual(values = c('#1082CD', '#042B41')) +
```

```
theme_bw() +
```

```
theme(legend.position = c(0.92, 0.20), legend.title = element_blank(),
```

```
legend.box.background = element_rect(colour = 'black'), panel.border =
element_blank(), axis.ticks = element_line(colour = '#E6E6E6'))
```

```
ggsave('energy.png', width = 20, height = 10, units = 'cm')
```



Seaborn Facet Grids

- `pd.date_range()` to generate `datetimeindex`
- `DateFormatter` for formatting `xtick` labels

Note on Practical Implementation of tick related methods

#<https://stackoverflow.com/questions/61527817/#formatting-date-labels-using-seaborn-facetgrid>

#Check answer by [Diziet Asahi](#)

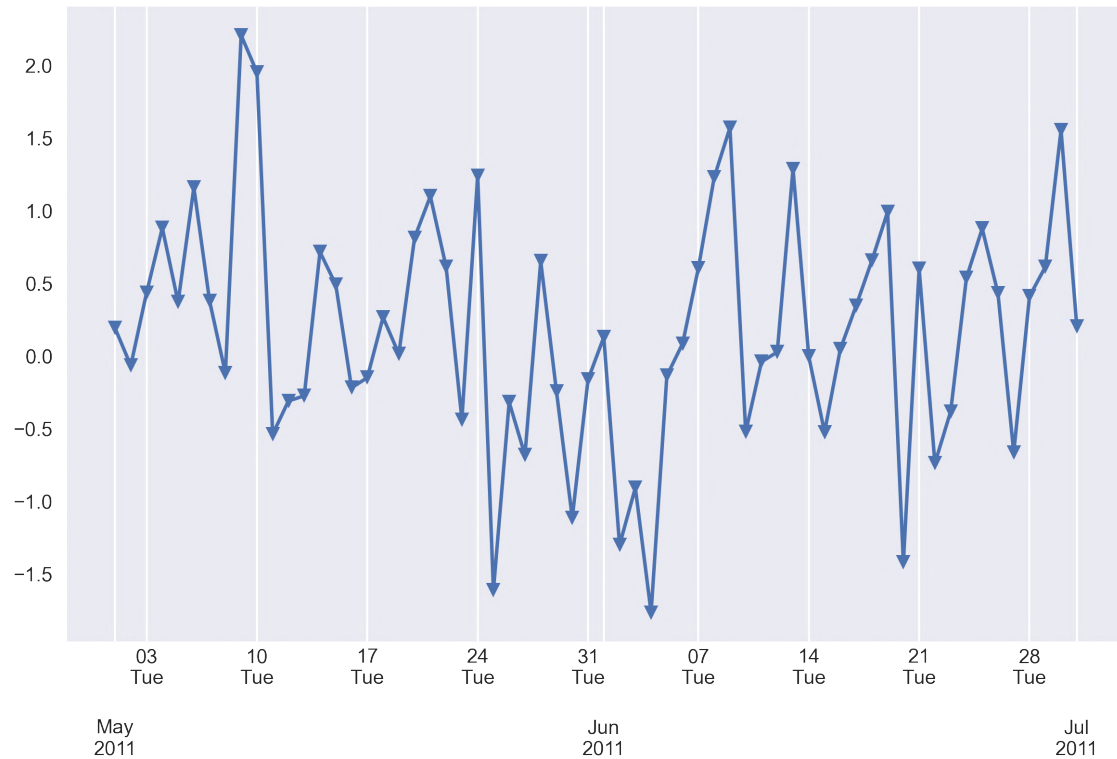
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib.dates as mdates
import random
```

```
datelist = pd.date_range(start="march 1 2020", end="may 20 2020", freq="w").tolist()
varlist = ["x", "y", "z", "x", "y", "z", "x", "y", "z", "x", "y", "z"]
deptlist = ["a", "a", "b", "a", "a", "b", "a", "a", "b", "a", "a", "b"]
vallist = random.sample(range(10, 30), 12)
df = pd.DataFrame({'date': datelist, 'value': vallist, 'variable': varlist,
                  'department': deptlist})
```

```
g = sns.FacetGrid(df, row="department", col="variable", sharey='row')
g = g.map(plt.plot, "date", "value", marker='o', markersize=0.7)
```

```
xformatter = mdates.DateFormatter("%m/%d")
g.axes[0,0].xaxis.set_major_formatter(xformatter)
g.axes[0,0].figure.savefig(f'Date labels Seaborn plot.png', dpi=300, format='png',
                          bbox_inches='tight')

plt.show()
```



dates module

- `pd.date_range()` to generate datetime index
- `MonthLocator()` for major ticks
- `WeekdayLocator()` for minor ticks
- `Date Formatter`

Note on Practical Implementation of tick related methods

#<https://stackoverflow.com/questions/12945971/>

#pandas-timeseries-plot-setting-x-axis-major-and-minor-ticks-and-labels?

#Check answer by [bmu](#)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

```
plt.style.use('seaborn')
idx = pd.date_range('2011-05-01', '2011-07-01')
```

```
prng = np.random.default_rng(123456)
s = pd.Series(prng.standard_normal(len(idx)), index=idx)
```

```
fig, ax = plt.subplots()
```

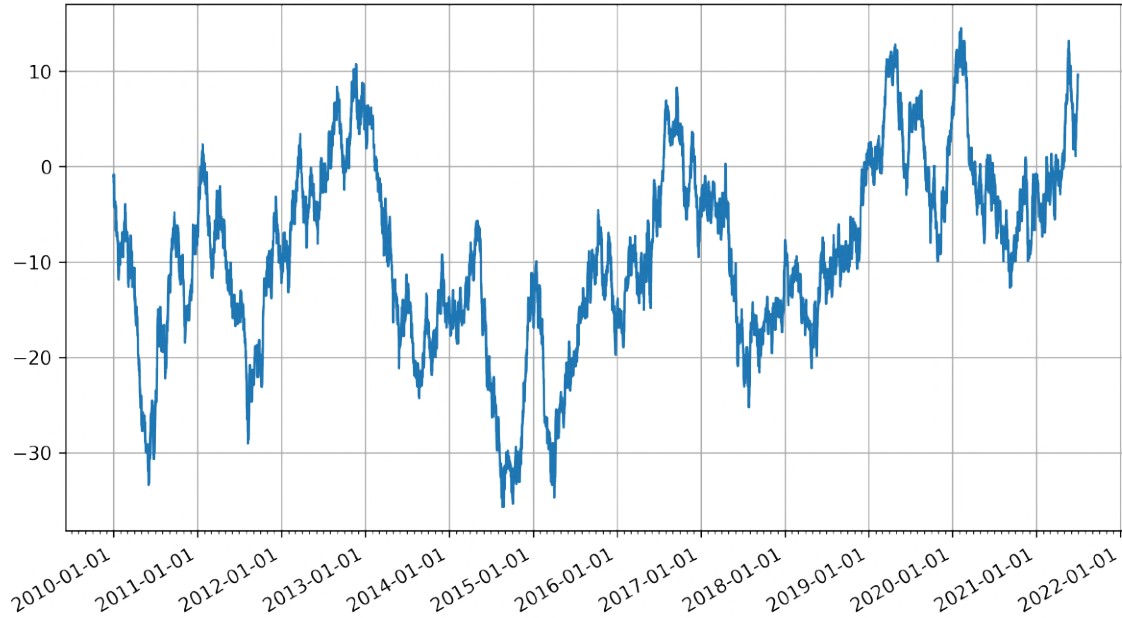
```
ax.plot_date(idx.to_pydatetime(), s, 'v-') #x values converted to python datetime objects
```

```
ax.xaxis.set_minor_locator(mdates.WeekdayLocator(byweekday=(1),interval=1))
ax.xaxis.set_minor_formatter(mdates.DateFormatter('%d\n%a'))
```

```
ax.xaxis.grid(True, which="minor")
ax.yaxis.grid()
```

```
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('\n\n\n%b\n%Y'))
```

```
plt.tight_layout()
fig.savefig('Major and Minor date ticks in multiline.png',dpi=300, format='png',
bbox_inches='tight')
```



Numpy datetime64

- Using `np.arange` to generate array of `datetime64[D]` values
- Using generator `np.random.default_rng`
- `YearLocator` and `MonthLocator` Classes

Note on Practical Implementation of tick related methods

```
import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator, YearLocator
import numpy as np
```

```
plt.rcParams()
fig, ax = plt.subplots(1, 1, figsize = (10,5))
```

```
#Using np.arange to create a range of datetime64 values
x = np.arange('2010-01', '2021-07', dtype='datetime64[D]')
```

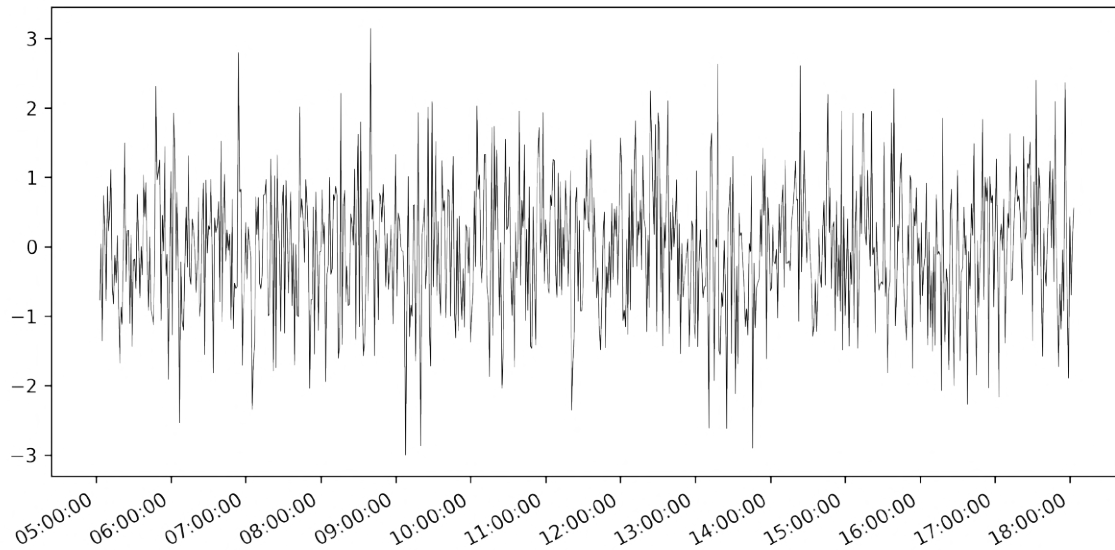
```
# np.random.seed(12345)
#y = np.random.randn(x.size).cumsum()
```

```
# Using the generator np.random.default_rng to generate values from standard normal
rng = np.random.default_rng(12345)
y = rng.standard_normal(x.size).cumsum()
ax.plot(x, y)
```

```
yloc = YearLocator()
mloc = MonthLocator()
ax.xaxis.set_major_locator(yloc)
ax.xaxis.set_minor_locator(mloc)
```

```
ax.xaxis.set_tick_params(rotation = 30)
[tkl.set_ha('right') for tkl in ax.get_xticklabels()]
ax.grid(True)
```

```
savekargs = dict(dpi = 300, transparent = True, pad_inches = 0.5, bbox_inches = 'tight')
fig.savefig('Numpy_datetime64_plotting.png', **savekargs)
```



Pandas Tseries Plotting

- Suppressing Tick Resolution Adjustment by Pandas plotting
- `x_compat = True` to apply matplotlib.dates locator/formatter
- Ticks at 1 hour interval

Note on Practical Implementation of tick related methods

[#https://stackoverflow.com/questions/48790378/how-to-get-ticks-every-hour](https://stackoverflow.com/questions/48790378/how-to-get-ticks-every-hour)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

```
style = ['fivethirtyeight', 'seaborn', 'ggplot', 'default']
plt.style.use(style[3])
```

```
idx = pd.date_range('2017-01-01 05:03', '2017-01-01 18:03', freq = 'min')
df = pd.Series(np.random.randn(len(idx)), index = idx)
```

```
fig, ax = plt.subplots(figsize = (10,5))
hours = mdates.HourLocator(interval = 1)
h_fmt = mdates.DateFormatter('%H:%M:%S')
```

```
ax.plot(df.index, df.values, color = 'black', linewidth = 0.4)
```

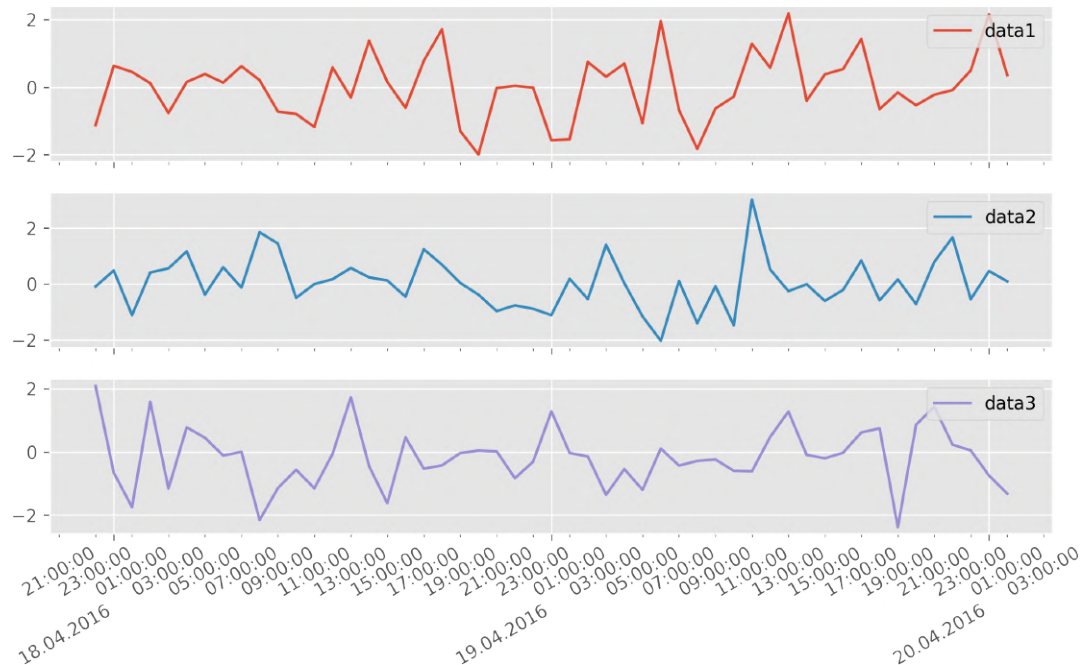
#or use

#df.plot(ax = ax, color = 'black', linewidth = 0.4, x_compat=True, figsize = (10,5))

#Then tick and format with matplotlib:

```
ax.xaxis.set_major_locator(hours)
ax.xaxis.set_major_formatter(h_fmt)
```

```
fig.autofmt_xdate()
savekargs = dict(dpi = 300, transparent = True, pad_inches = 0.5, bbox_inches = 'tight')
fig.savefig('datetimeindex_plotting.png', **savekargs)
plt.show()
```



Pandas Tseries Plotting

- Suppressing Tick Resolution Adjustment by Pandas plotting
- `x_compat = True` to apply matplotlib.dates locator/formatter
- Ticks at 2 hour interval

Note on Practical Implementation of tick related methods

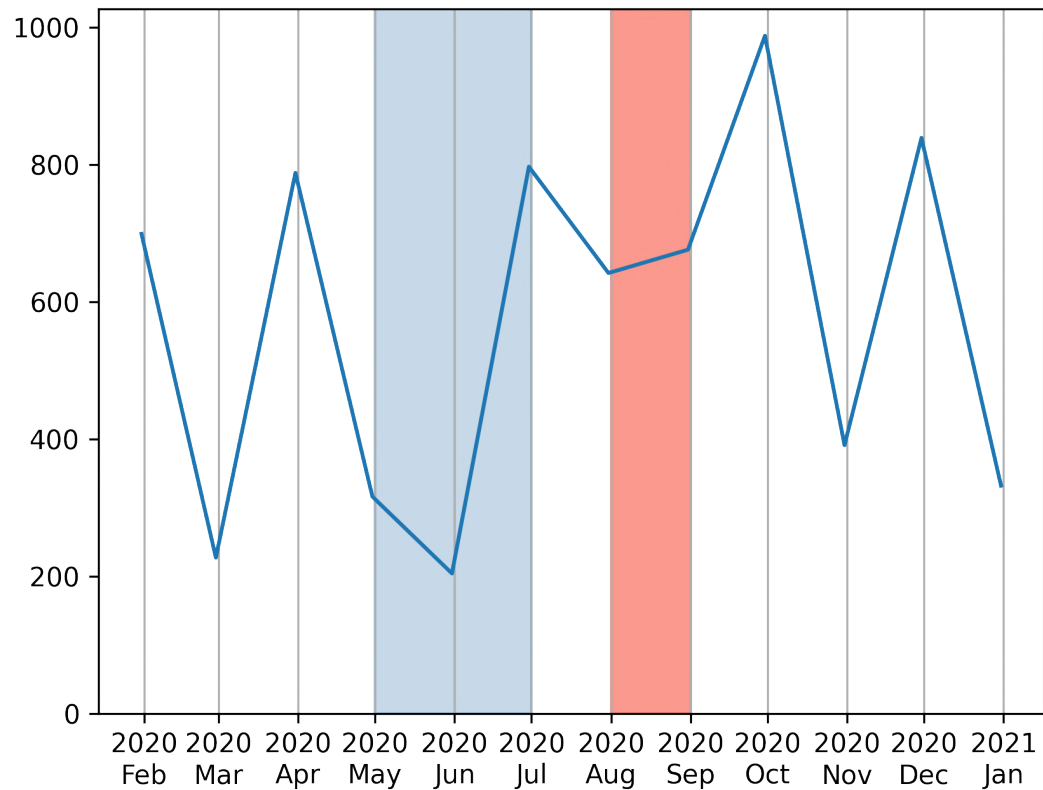
```
#https://stackoverflow.com/questions/38154489/
#force-pandas-xaxis-datetime-index-using-a-specific-format
import pandas as pd
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

timeInd = pd.date_range(start = datetime(2016,4,17,23,0,0),
                        end = datetime(2016,4,20,1,0,0), freq = 'H')
d = {'data1': np.random.randn(len(timeInd)), 'data2': np.random.randn(len(timeInd)),
     'data3': np.random.randn(len(timeInd))}
df = pd.DataFrame(data = d, index = timeInd)

plt.style.use('ggplot')
ax0 = df.plot(subplots=True, grid=True, x_compat=True, figsize = (10,6))

ax = plt.gca()
# set major ticks location every day
ax.xaxis.set_major_locator(mdates.DayLocator())
# set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('\n\n\n%d.%m.%Y'))
# set minor ticks location every two hours
ax.xaxis.set_minor_locator(mdates.HourLocator(interval=2))
# set minor ticks format
ax.xaxis.set_minor_formatter(mdates.DateFormatter('%H:%M:%S'))

for ax in ax0:
    ax.legend(loc = 'upper right')
    ax.figure.patch.set(visible = False)
    # or just set together date and time for major ticks like
    # ax.xaxis.set_major_formatter(mdates.DateFormatter('%d.%m.%Y %H:%M:%S'))
ax.figure.savefig('Pandasplotting_datetime_xcompat.png', dpi = 300, pad_inches = 0.02)
plt.show()
```

TimeSeries Plotting

- `pd.date_range()`
- `dt.datetime`
- `np.random.default_rng(12345)`
- `fill_betweenx()`
- `Axvspan()`
- `MonthLocator()`

Note on Practical Implementation of tick related methods

```
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import datetime as dt
```

```
rng = np.random.default_rng(12345)
dtindex=pd.date_range(start='2020-01-01',end='2020-12-31',freq='M')
df = pd.DataFrame({'values':rng.integers(0,1000, len(dtindex))}, index= dtindex)
```

```
fig,ax1 = plt.subplots()
ax1.plot(df.index,df.values)
```

```
ax1.xaxis.set_major_locator(mdates.MonthLocator())
```

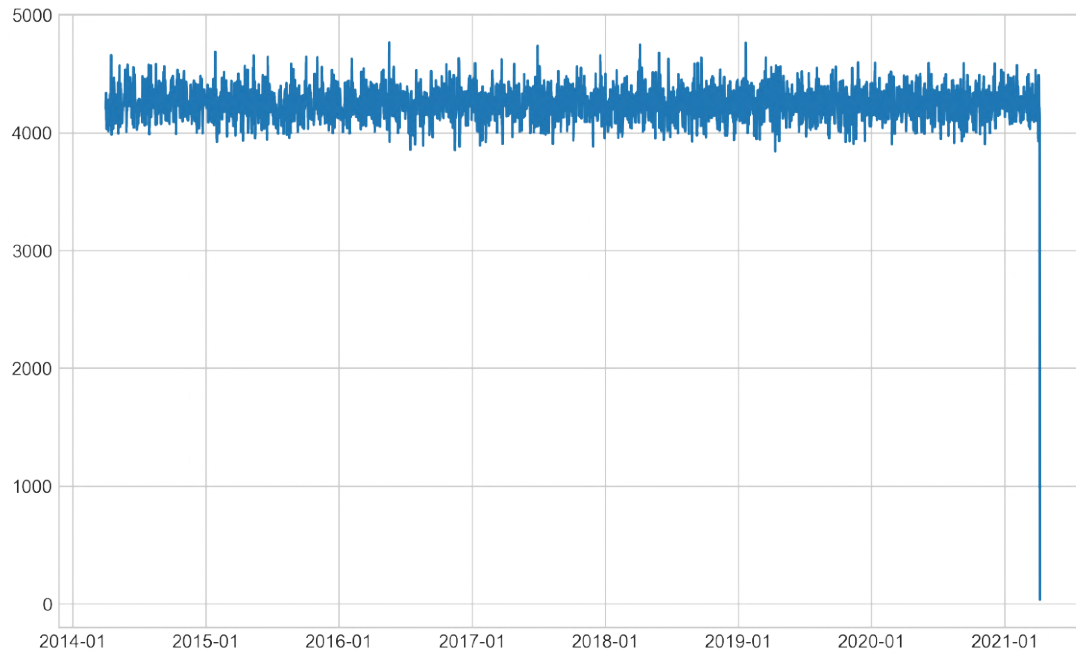
```
monthyearFmt = mdates.DateFormatter('%Y\n%b')
ax1.xaxis.set_major_formatter(monthyearFmt)
```

```
#Underscore and assignment so that tick locs and labels are not shown in output
_ = plt.xticks(ha = 'center')
plt.grid(axis = 'x')
```

```
x1 = dt.datetime(2020,0o5,0o1)
x2 = dt.datetime(2020,0o7,0o1)
ax1.set_ylim(0, ax1.get_ylim()[1])
ax1.fill_betweenx(y = (ax1.get_ylim()[0],ax1.get_ylim()[1]), x1 = x1, x2=x2, alpha = 0.3,
                  color = 'steelblue' )
```

```
x3 = dt.datetime(2020,9,0o1)
x4 = dt.datetime(2020,0o10,0o1)
ax1.axvspan(xmin = x3, xmax = x4, color = 'salmon', alpha = 0.8)
```

```
fig.savefig('Simple TimeSeries plot.png', dpi = 300, transparent = True)
```

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np

# create reproducible dataframe
dates = pd.date_range(start='2014-04-01', end='2021-04-07', freq='10min').tolist()

rng = np.random.default_rng(123456)
utilization = [rng.integers(10, 50) for _ in range(len(dates))]
df = pd.DataFrame({'dates': dates, 'util': utilization})
df.set_index('dates', inplace=True)

# resample dataframe to daily
df_daily = pd.DataFrame(df['util'].resample('D').sum())

# plot the dataframe
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.plot(df_daily.index, df_daily['util'])
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))

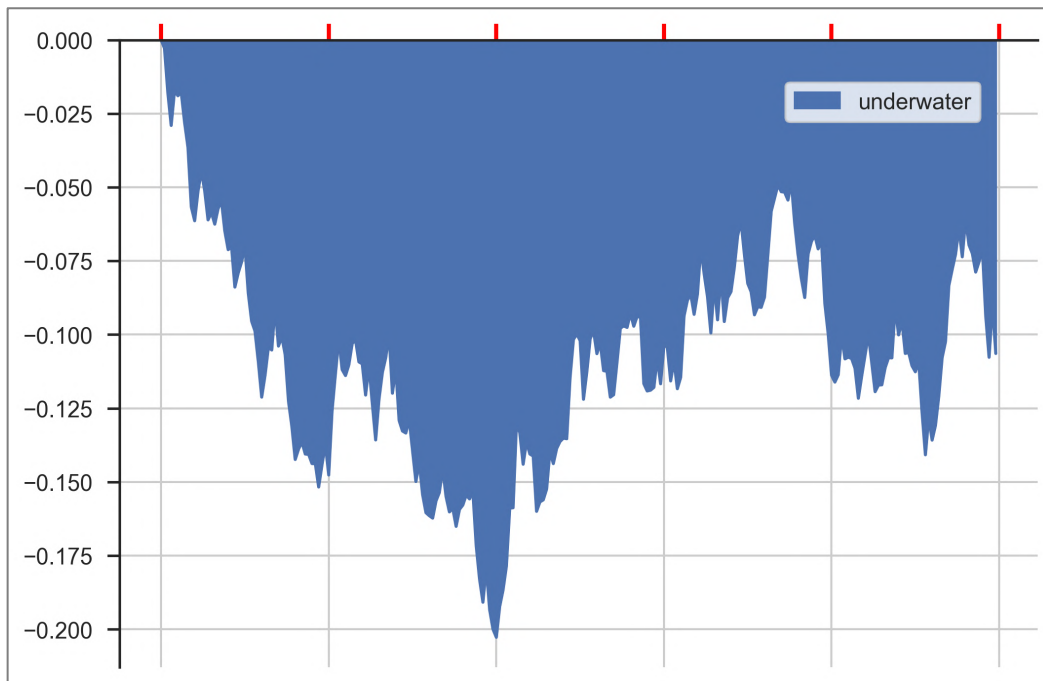
fig.savefig('Resampling.png', dpi = 300, transparent = True)
plt.show()
```

Time series Plotting

- Resampling the dataframe to Day frequency

Note on Practical Implementation of tick related methods

https://pandas.pydata.org/docs/user_guide/timeseries.html#resampling



#<https://stackoverflow.com/questions/57097467/>

#Check answer by [Sheldore](#)

#Matplotlib x-axis at the top, no x axis label, no major tick labels, but outer major and minor tick wanted

Pandas
Seaborn
Matplotlib

- sns.despine
- Adding red ticks in out direction

Note on Practical Implementation of tick related methods

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
```

```
np.random.seed(123456)
perf = np.cumprod(1+np.random.normal(size=250)/100)
df = pd.DataFrame({'underwater': perf/np.maximum.accumulate(perf)-1})
```

```
sns.set_context('notebook')
sns.set_style('ticks', rc = {'axes.grid': True, 'axes.spines.left': True,
                             'axes.spines.right': False, 'axes.spines.bottom': True,
                             'axes.spines.top': False})
```

```
#Assigning current axes to ax
ax = plt.gca()
df.plot.area(ax=ax, label = 'underwater')
ax.xaxis.set_ticklabels([])
ax.set_ylim(ax.get_ylim()[0],0)
```

```
#Removing spines at bottom and right
```

```
sns.despine(left=False, bottom=True, right=True, top = False, ax=ax)
```

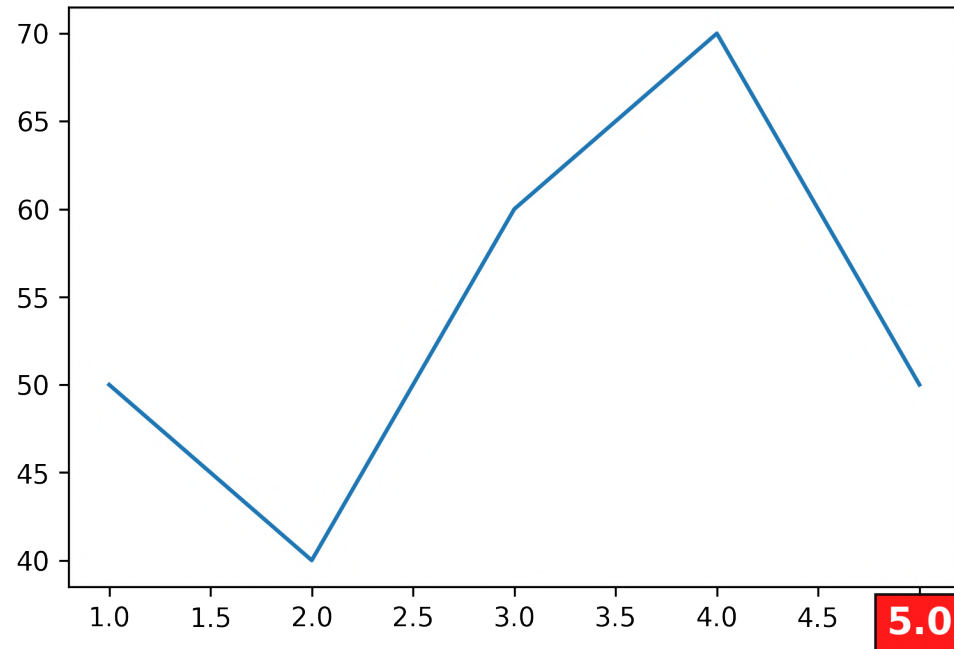
```
#Adding legend with frameon
```

```
ax.legend(loc = 'upper right', bbox_to_anchor = (0.95,0.95),
          bbox_transform = ax.transAxes, frameon = True )
```

```
# Adding red ticks in x axis pointing outwards
```

```
ax.tick_params('x', length=8, width=2, color='red', which='major', direction='out')
```

```
ax.figure.savefig('Major_ticks_Outer_No_AxisLabel.png',dpi=300, format='png',
bbox_inches='tight')
```



Format select few tick labels

- Indexing/slicing axis tick labels
- Setting the bbox for text artist

[matplotlib.patches.FancyBboxPatch](#)

[FancyBboxPatch Demo](#)

Note on Practical Implementation of tick related methods

#<https://stackoverflow.com/questions/41924963/formatting-only-selected-tick-labels>

```
import matplotlib.pyplot as plt
```

```
plt.rcParamsdefaults()
```

```
fig, ax = plt.subplots()
```

```
ax.plot([1, 2, 3, 4, 5], [50, 40, 60, 70, 50])
```

```
ax.get_xticklabels()[-2].set_color("white")
```

```
ax.get_xticklabels()[-2].set_fontsize(14)
```

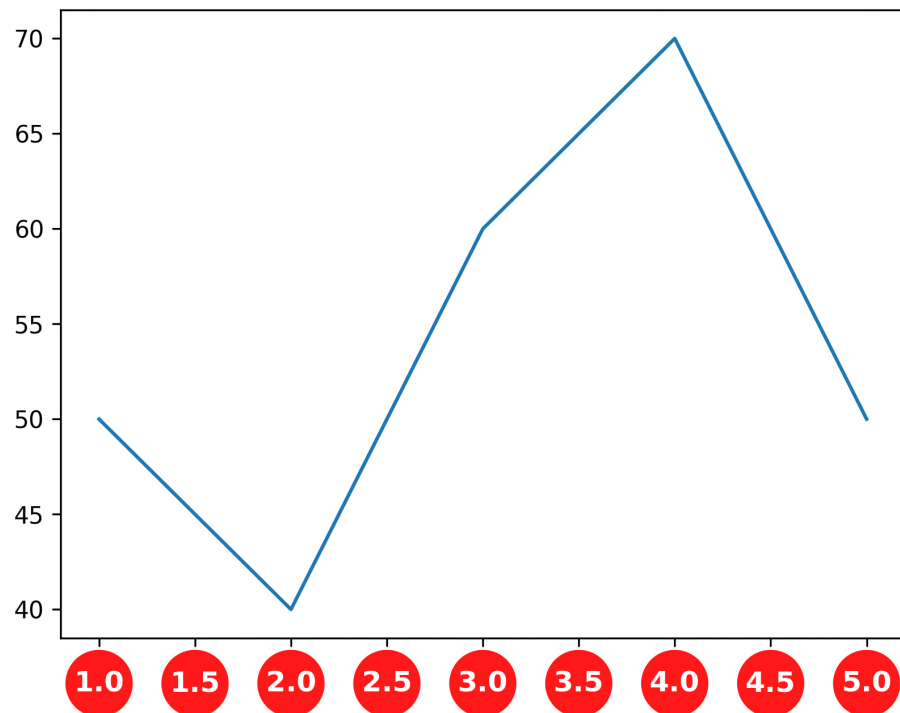
```
ax.get_xticklabels()[-2].set_weight("bold")
```

```
ax.get_xticklabels()[-2].set_bbox(dict(facecolor="red", alpha=0.9))
```

```
fig.savefig('Highlight selected ticks.png', dpi = 300, format = 'png',bbox_inches='tight')
plt.show()
```

[boxstyle types](#)

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3, rounding_size=None
Round4	round4	pad=0.3, rounding_size=None
Roundtooth	roundtooth	pad=0.3, tooth_size=None
Sawtooth	sawtooth	pad=0.3, tooth_size=None
Square	square	pad=0.3



Format all tick labels

- Iterate over axis tick labels
- Setting the bbox for text artist
- Set font properties

[matplotlib.patches.FancyBboxPatch](#)

Note on Practical Implementation of tick related methods

```
#https://stackoverflow.com/questions/41924963/formatting-only-selected-tick-labels
#The above link contains only one tick formatted. The example is extended here for different possible use-cases
plt.rcParams()
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4, 5], [50, 40, 60, 70, 50])

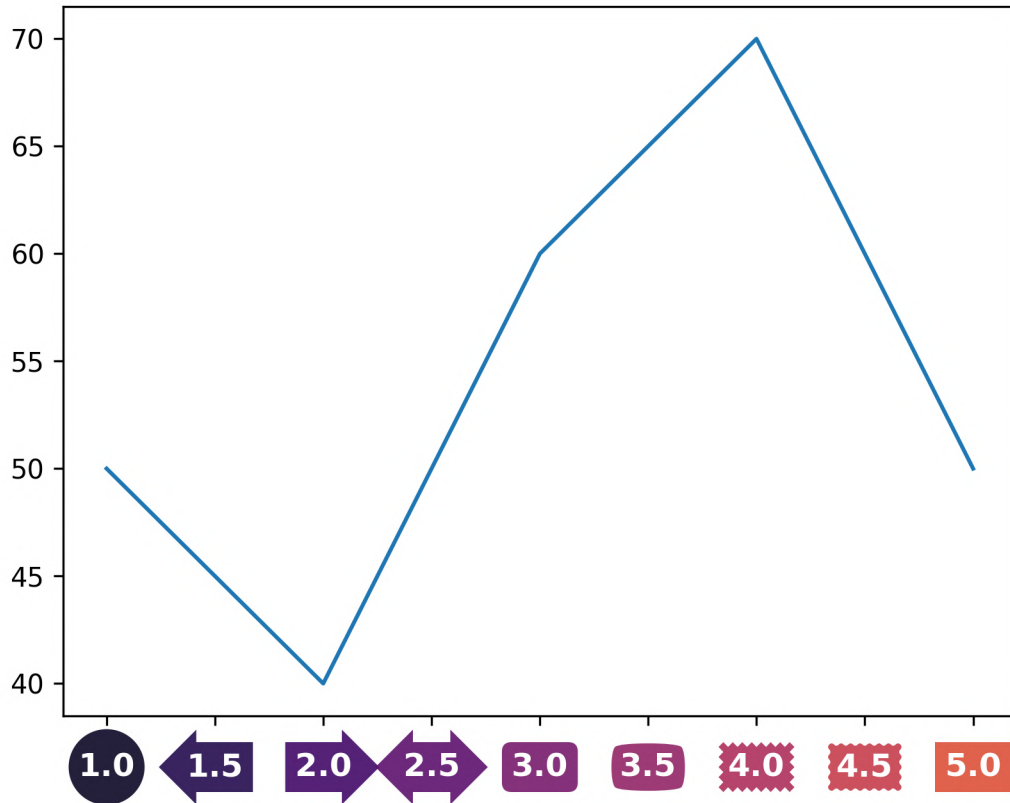
bbox_dict = dict(facecolor='red', alpha=0.9, ec = None, lw = 0, boxstyle = 'circle')
fontargs = dict(fontsize = 12, color = 'white', fontweight = 'bold', va = 'top')

for tkl in ax.get_xticklabels(): #Iterating over the xaxis tick labels
    #Setting the bbox with dict and dict unpacking
    tkl.set(bbox = bbox_dict, **fontargs)

ax.xaxis.set_tick_params(pad = 10) #Increasing the padding between tick and labels
fig.savefig('Highlight selected ticks_2.png', dpi = 300, bbox_inches='tight', pad_inches = 0.3)
plt.show()
```

boxstyle types

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3, rounding_size=None
Round4	round4	pad=0.3, rounding_size=None
Roundtooth	roundtooth	pad=0.3, tooth_size=None
Sawtooth	sawtooth	pad=0.3, tooth_size=None
Square	square	pad=0.3



- [matplotlib.patches.FancyBboxPatch.html](https://matplotlib.org/patches/FancyBboxPatch.html)
- [Formatting only selected tick labels](#)
- [specify-where-to-start-in-an-itertools-cycle-function](#)
- [fancybox_demo](#)
- [add-new-keys-to-a-dictionary](#)

Format all tick labels

- Use cycle iterator object with islice
- Set the bbox style with cycle
- Create ListedColormap from colormap and use that within cycle object to assign color to ticklabels

Note on Practical Implementation of tick related methods

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
from matplotlib.colors import LinearSegmentedColormap
from itertools import cycle, islice #Using iterator to cycle colors and boxstyle
import matplotlib.patches as mpatch

plt.rcParamsdefaults()
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4, 5], [50, 40, 60, 70, 50])

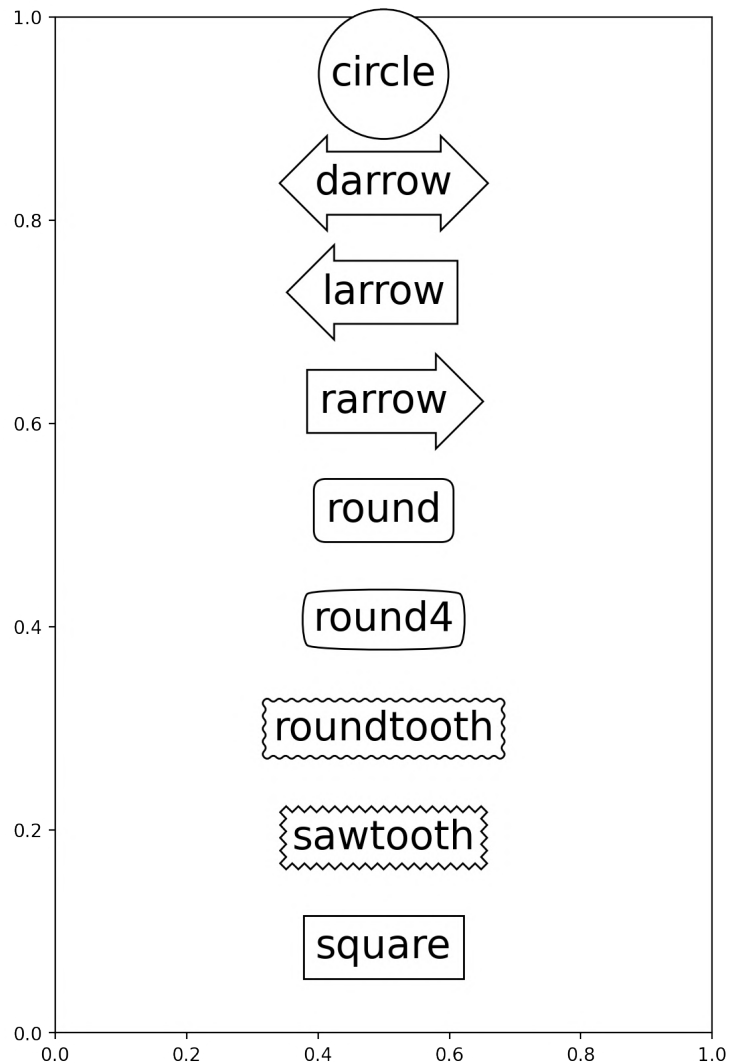
#If more number of ticks present, we want box style to infinitely loop over style list
boxstyle_list = islice(cycle(mpatch.BoxStyle.get_styles()), 0, None) #cycle with islice
#boxstyle_list = islice(cycle(['circle', 'darrow', 'rarrow', 'larrow', 'round', 'round4',
                             'roundtooth', 'square', 'sawtooth']))

cmap = 'inferno' #color map
listedcolormap = ListedColormap(cm.get_cmap(cmap, 256)(np.linspace(0, 0.6, 10)))
#Using islice to always start from first position
color_list = islice(cycle(listedcolormap.colors), 0, None)

#create fontargs and bboxargs to pass as arguments in ticklabel properties set
fontargs = {'color': 'white', 'fontsize': 12, 'fontweight': 'bold', 'ha': 'center', 'va': 'top'}
bboxargs = dict(alpha=0.9, ec=None, lw=0)

ax.xaxis.set_tick_params(pad=10) #Ensuring tick labels have sufficient gap from axis
for tkl in ax.get_xticklabels():
    tkl.set(bbox = (**bboxargs, 'boxstyle': next(boxstyle_list), 'facecolor': next(color_list)),
            **fontargs)

fig.savefig('Highlight selected ticks_3.png', dpi=300, format='png', bbox_inches='tight',
            pad_inches=0.3)
plt.show()
```



```
import matplotlib.pyplot as plt
import matplotlib.transforms as mtransforms
import matplotlib.patches as mpatch
from matplotlib.patches import FancyBboxPatch
```

```
styles = mpatch.BoxStyle.get_styles()
spacing = 1.2
figheight = (spacing * len(styles) + .5)
fig = plt.figure(figsize=(4 / 1.5, figheight / 1.5))
fontsize = 0.3 * 72
for i, stylename in enumerate(sorted(styles)):
    fig.text(0.5, (spacing * (len(styles) - i) - 0.5) / figheight,
            stylename, ha="center", size=fontsize,
            bbox=dict(boxstyle=stylename, fc="w", ec="k"))
```

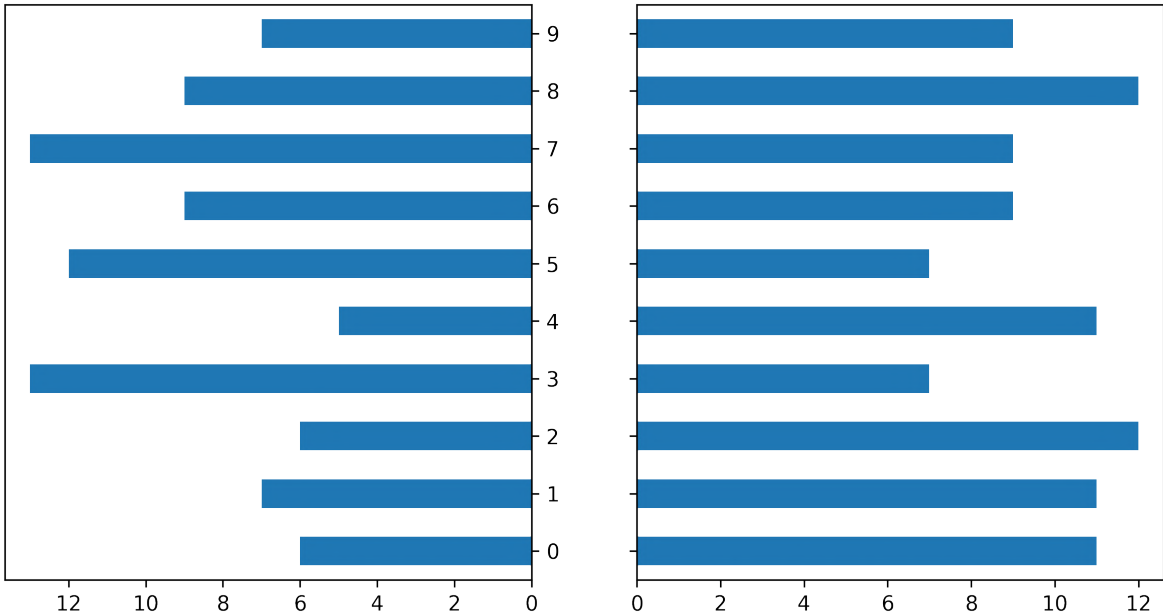
boxstyle types

Class	Name	Attrs
Circle	circle	pad=0.3
DArrow	darrow	pad=0.3
LArrow	larrow	pad=0.3
RArrow	rarrow	pad=0.3
Round	round	pad=0.3, rounding_size=None
Round4	round4	pad=0.3, rounding_size=None
Roundtooth	roundtooth	pad=0.3, tooth_size=None
Sawtooth	sawtooth	pad=0.3, tooth_size=None
Square	square	pad=0.3

Patches

• FancyBboxPatch

- [matplotlib.patches.FancyBboxPatch.html](#)
- [fancybox_demo](#)



Adjacent Barh charts

- Pandas DataFrame + Matplotlib

- <https://stackoverflow.com/questions/44049132/python-pandas-plotting-two-barh-side-by-side>

Context of the Plot

- Two adjacent barh plots side-by-side sharing the y-axis
- Remove the label in y-axis for the lefthand side chart
- Xaxis of the two plots should start from the middle of the two plots

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
np.random.seed(123456)
a = np.random.randint(5,15, size=10)
b = np.random.randint(5,15, size=10)
```

```
df = pd.DataFrame({"a":a})
df2 = pd.DataFrame({"b":b})
```

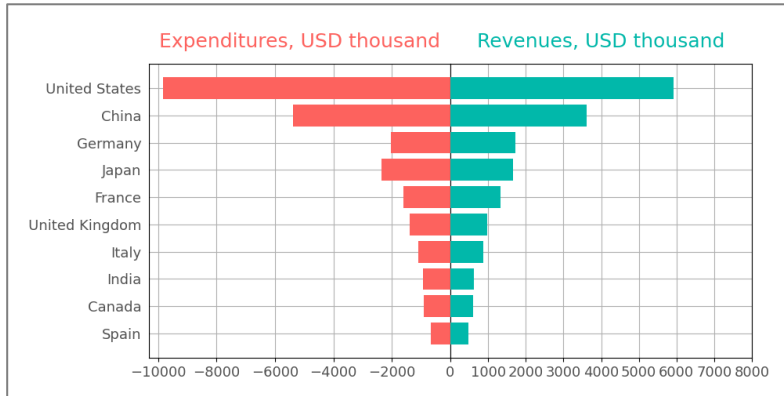
```
fig, (ax, ax2) = plt.subplots(ncols=2, sharey=True, figsize = (10,5))
```

```
ax.invert_xaxis()
ax.yaxis.tick_right()
```

```
df["a"].plot(kind='barh', x='LABEL', legend=False, ax=ax)
df2["b"].plot(kind='barh', x='LABEL', ax=ax2)
```

```
fig.patch.set_visible(False)
ax.patch.set_visible(False)
ax2.patch.set_visible(False)
```

```
fig.savefig('Two barh adjacent.png', dpi = 300)
plt.show()
```

Bi Directional Charts

<https://sharkcoder.com/data-visualization/mpl-bidirectional>

```
import requests
import pandas as pd
```

```
url =
'https://en.wikipedia.org/wiki/List_of_countries_by_government_budget'
```

```
response = requests.get(url)
tables = pd.read_html(response.text)
tables[0].to_csv('data.csv')
```

```
df = pd.read_csv('data.csv')
```

```
data = df[['Country', 'Revenues', 'Expenditures']].head(10)
data.set_index('Country', inplace=True)
data['Revenues'] = data['Revenues'].astype(float)
data['Revenues'] = data['Revenues'] / 1000
data['Expenditures'] = data['Expenditures'].astype(float)
data['Expenditures'] = data['Expenditures'] / 1000 * (-1)
```

```
font_color = '#525252'
hfont = {'fontname': 'Calibri'} # dict for unpacking later
facecolor = '#eaeaf2'
color_red = '#fd625e'
color_blue = '#01b8aa'
index = data.index
column0 = data['Expenditures']
column1 = data['Revenues']
title0 = 'Expenditures, USD thousand'
title1 = 'Revenues, USD thousand'
```

```
import matplotlib.pyplot as plt
plt.rcParams()
```

```
fig, axes = plt.subplots(figsize=(10,5), facecolor=facecolor,
                           ncols=2, sharey=True)
```

```
fig.tight_layout()
```

```
axes[0].barh(index, column0, align='center',
              color=color_red, zorder=10)
axes[0].set_title(title0, fontsize=18, pad=15, color=\
                  color_red, fontweight = 'medium', **hfont)
axes[1].barh(index, column1, align='center',
              color=color_blue, zorder=10)
axes[1].set_title(title1, fontsize=18, pad=15, color=\
                  color_blue, fontweight = 'medium', **hfont)
```

```
# If you have positive numbers and want to invert the x-axis
of the left plot
#axes[0].invert_xaxis()
```

```
# To show data from highest to lowest
plt.gca().invert_yaxis()
```

```
axes[0].set(yticks=data.index, yticklabels=data.index)
axes[0].yaxis.tick_left()
axes[0].tick_params(axis='y', colors='white') # tick color
```

```
axes[1].set_xticks([1000, 2000, 3000, 4000, 5000, 6000,
                    7000, 8000])
axes[1].yaxis.set_tick_params(size=0)

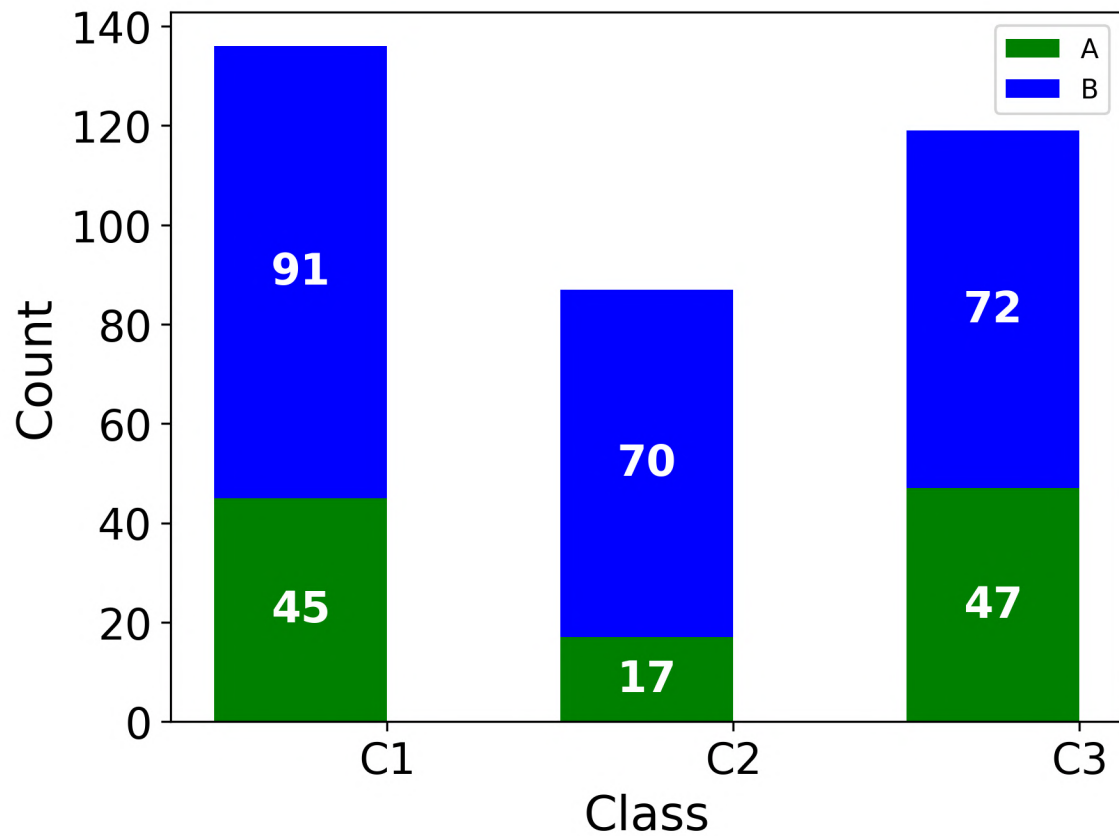
axes[1].set_xticklabels([1000, 2000, 3000, 4000, 5000, 6000,
                        7000,8000])
```

```
for label in (axes[0].get_xticklabels() + axes[0].get_yticklabels()):
    label.set(fontsize=13, color=font_color, **hfont)
for label in (axes[1].get_xticklabels() + axes[1].get_yticklabels()):
    label.set(fontsize=13, color=font_color, **hfont)
```

```
fig.patch.set(facecolor = 'white')
#wspace=0 for no gap between the two axes
plt.subplots_adjust(wspace=0, top=0.85, bottom=0.1,
                    left=0.18, right=0.95)
[ax.grid(True) for ax in fig.axes]
filename = 'mpl-bidirectional2'
plt.savefig(filename+'.png', dpi=300, format='png',
            bbox_inches='tight')
```

The final data frame used for the plot after data aggregation and manipulation is as below (can be used directly) :

Country	Revenues	Expenditures
United States	5923.829	-9818.53
China	3622.313	-5388.81
Germany	1729.224	-2038.25
Japan	1666.454	-2362.68
France	1334.944	-1609.71
United Kingdom	966.407	-1400.78
Italy	863.785	-1103.72
India	620.739	-940.771
Canada	598.434	-917.271
Spain	481.945	-657.75



Stacked Bar Charts

- Adding Labels to the bars

#<https://stackoverflow.com/questions/41296313/stacked-bar-chart-with-centered-labels?>

```
import numpy as np
import matplotlib.pyplot as plt
```

```
A = [45, 17, 47]
```

```
B = [91, 70, 72]
```

```
#Creating dictionary that contains keyword argument property values for text instance
```

```
text_kwargs = {'ha':"center", 'va':"center", 'color':"white", 'fontsize':16, 'fontweight':"bold"}
```

```
fig = plt.figure(facecolor="white")
```

```
ax = fig.add_subplot(1, 1, 1)
```

```
bar_width = 0.5
```

```
bar_l = np.arange(1, 4)
```

```
tick_pos = [i + (bar_width / 2) for i in bar_l]
```

```
ax1 = ax.bar(bar_l, A, width=bar_width, label="A", color="green")
```

```
ax2 = ax.bar(bar_l, B, bottom=A, width=bar_width, label="B", color="blue")
```

```
ax.set_ylabel("Count", fontsize=18)
```

```
ax.set_xlabel("Class", fontsize=18)
```

```
ax.legend(loc="best") #Adding Legend to "best" location
```

```
plt.xticks(tick_pos, ["C1", "C2", "C3"], fontsize=16)
```

```
plt.yticks(fontsize=16)
```

```
for r1, r2 in zip(ax1, ax2): #Adding data labels at the center of each of the bar patches
```

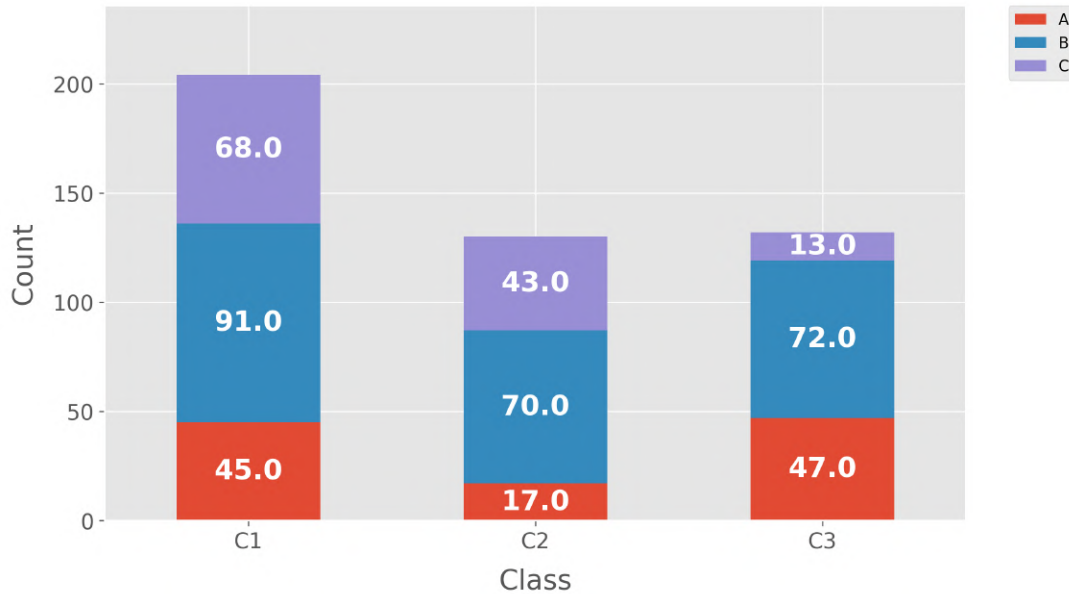
```
    h1 = r1.get_height()
```

```
    h2 = r2.get_height()
```

```
    ax.text(r1.get_x() + r1.get_width() / 2., h1 / 2., "%d" % h1, **text_kwargs) #unpacking dict
```

```
    ax.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2., "%d" % h2, **text_kwargs)
```

```
fig.savefig('labels_stacked_bar_charts.png', dpi = 300, format = 'png', bbox_inches='tight')
plt.show()
```



#<https://stackoverflow.com/questions/41296313/>

#Check answer by [Trenton McKinney](#)

Below changes vis-a-vis above link

- Dictionary style unpacking of text keyword arguments
- Tick labels, Axis labels and labelpad
- Resetting yaxis limits

```
import pandas as pd
import matplotlib.pyplot as plt
```

#Data

A = [45, 17, 47]

B = [91, 70, 72]

C = [68, 43, 13]

pandas dataframe

```
df = pd.DataFrame(data={'A': A, 'B': B, 'C': C})
df.index = ['C1', 'C2', 'C3']
```

Stacked
Bar Charts
ggplot style

```
plt.style.use('ggplot')
text_kwargs = {'ha': "center", 'va': "center", 'color': "white",
               'fontsize': 18, 'fontweight': "semibold"}
```

```
ax = df.plot(stacked=True, kind='bar', figsize=(10, 6), rot='horizontal')
fig = ax.figure
```

.patches gives access to all the patches (bars)

for rect in ax.patches:

Find where everything is located

height = rect.get_height()

width = rect.get_width()

x = rect.get_x()

y = rect.get_y()

The height of the bar is the data value and can be used as the label

label_text = f'{height}' # f'{height:.2f}' to format decimal values

ax.text(x, y, text)

label_x = x + width / 2

label_y = y + height / 2

plot only when height is greater than specified value

if height > 0:

ax.text(label_x, label_y, label_text, **text_kwargs) #Dictionary style unpacking

ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0) #Adding Legend

ax.set_ylabel("Count", fontsize=18, labelpad = 10) #labelpad and fontsize changed

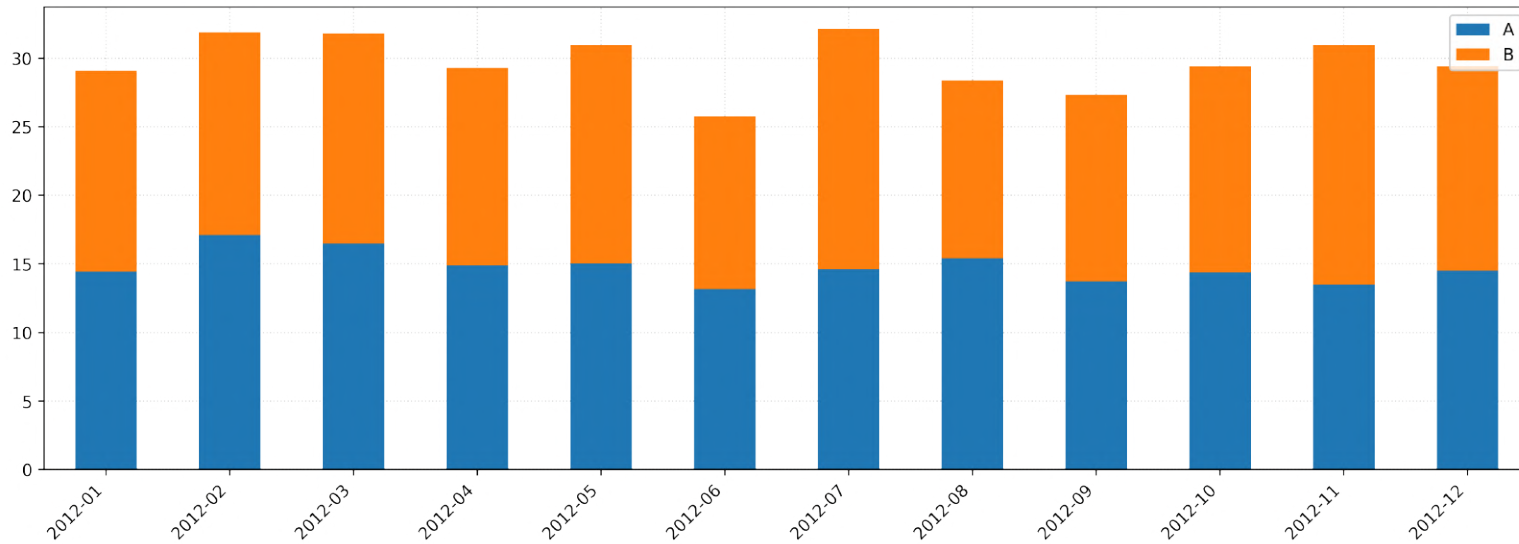
ax.set_xlabel("Class", fontsize=18, labelpad = 10) #labelpad and fontsize changed

ax.tick_params(labelsize = 16)

ax.set_ylim(0, ax.get_ylim()[1]*1.1) #Increasing the upper ylim to create spacing

fig.savefig('labels_stacked_bar_charts_ggplot.png', dpi = 300,
 format = 'png',bbox_inches='tight')

plt.show()



```
# Pandas Bar Plotting with datetime in x axis
#https://stackoverflow.com/questions/30133280/pandas-bar-plot-changes-date-format
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('default')
# generate sample data
start = pd.to_datetime("1-1-2012")
index = pd.date_range(start, periods= 365)

prng = np.random.default_rng(123456)
df = pd.DataFrame({'A' : prng.random(365), 'B' : prng.random(365)}, index=index)

# resample to any timeframe you need, e.g. months
df_months = df.resample("M").sum()
```

```
# plot
fig, ax = plt.subplots()
df_months.plot(kind= "bar", figsize=(16,5), stacked=True, ax=ax)

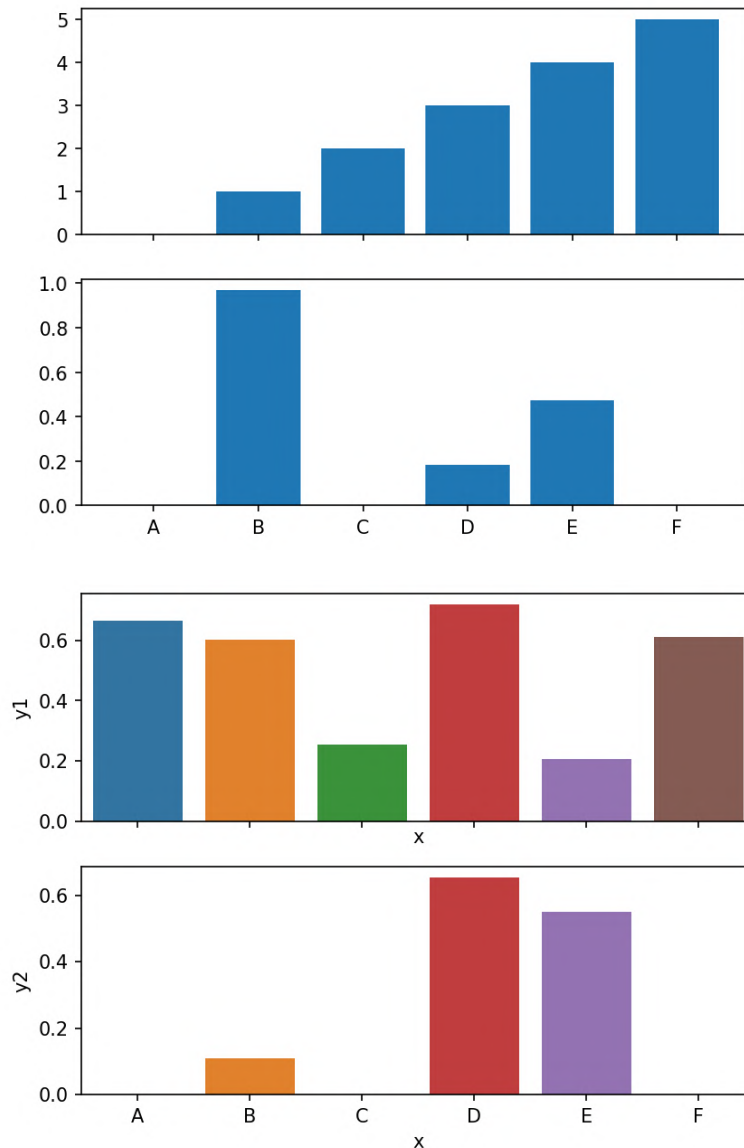
# Decorations/Embellishments
ax.grid(True, alpha = 0.5, ls = 'dotted')
ax.set_axisbelow(True)

# format xtick-labels with list comprehension
ax.set_xticklabels([x.strftime("%Y-%m") for x in df_months.index], ha = 'right', rotation=45)
fig.savefig('ticklabels_using timestamp.strftime.png', dpi = 300, bbox_inches = 'tight',
transparent = True)
plt.show()
```

Pandas Stacked Bar plots

- Datetime Index
- Using `timestamp.strftime` for tick labels
- Using Resample to 'Months'

Share-x on Data with different x values



#<https://stackoverflow.com/questions/55642270/matplotlib-sharex-on-data-with-different-x-values>

#Check answer by [gmds](#)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
fig, axes = plt.subplots(2, sharex='all')
ax1, ax2 = axes
```

```
df = pd.DataFrame({'x': ['A', 'B', 'C', 'D', 'E', 'F'], 'y1': np.arange(6)})
df2 = pd.DataFrame({'x': ['B', 'D', 'E'], 'y2': np.random.rand(3)})
```

```
combined = df.merge(df2, how='left', on='x')
```

```
ax1.bar(combined['x'], combined['y1'])
ax2.bar(combined['x'], combined['y2'])
```

```
fig.savefig(f'Share_x_across_subplots.png', dpi=150, format='png', bbox_inches='tight')
plt.show()
```

#Alternately, one can also use Seaborn

```
df = pd.DataFrame({'x': ['A', 'B', 'C', 'D', 'E', 'F'], 'y1': np.random.rand(6)})
df2 = pd.DataFrame({'x': ['B', 'D', 'E'], 'y2': np.random.rand(3)})
```

```
order = np.unique(list(df.x)+list(df2.x))
```

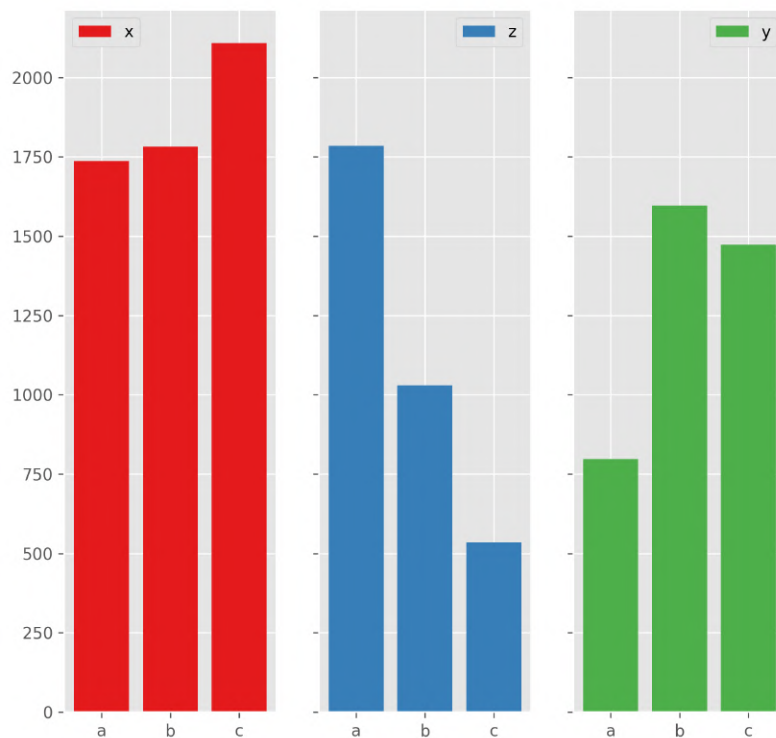
```
fig, axes = plt.subplots(2, sharex='all')
```

```
sns.barplot(x='x', y='y1', data=df, ax=axes[0], order=order)
sns.barplot(x='x', y='y2', data=df2, ax=axes[1], order=order)
plt.show()
```

Matplotlib

Seaborn

Employment By Industry By City



Faceted Bar Charts

Pandas + Matplotlib

Similar to facet_wrap in ggplot2,
Facet_grid in Seaborn

Also check :

- <https://stackoverflow.com/questions/19613486/pandas-matplotlib-faceting-bar-plots>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

Preparing Sample data

N = 100

industry = ['a','b','c']

city = ['x','y','z']

ind = np.random.choice(industry, N)

cty = np.random.choice(city, N)

jobs = np.random.randint(low=1,high=250,size=N)

df_city = pd.DataFrame({'industry':ind,'city':cty,'jobs':jobs})

Extracting the number of panels(cities) from df_city

cols = df_city.city.value_counts().shape[0]

fig, axes = plt.subplots(1, cols, figsize=(8, 8), sharey = True)

for x, city in enumerate(df_city.city.value_counts().index.values): [#for loop to create bar plots](#)

data = df_city[(df_city['city'] == city)]

data = data.groupby(['industry']).jobs.sum()

print (data)

print (type(data.index))

left= [k[0] for k in enumerate(data)]

right= [k[1] for k in enumerate(data)]

color = plt.cm.get_cmap('Set1').colors[x]

#Access colors from Listed colormaps

axes[x].bar(left,right,label="%s" % (city), color = color)

axes[x].set_xticks(left, minor=False)

axes[x].set_xticklabels(data.index.values)

axes[x].legend(loc='best')

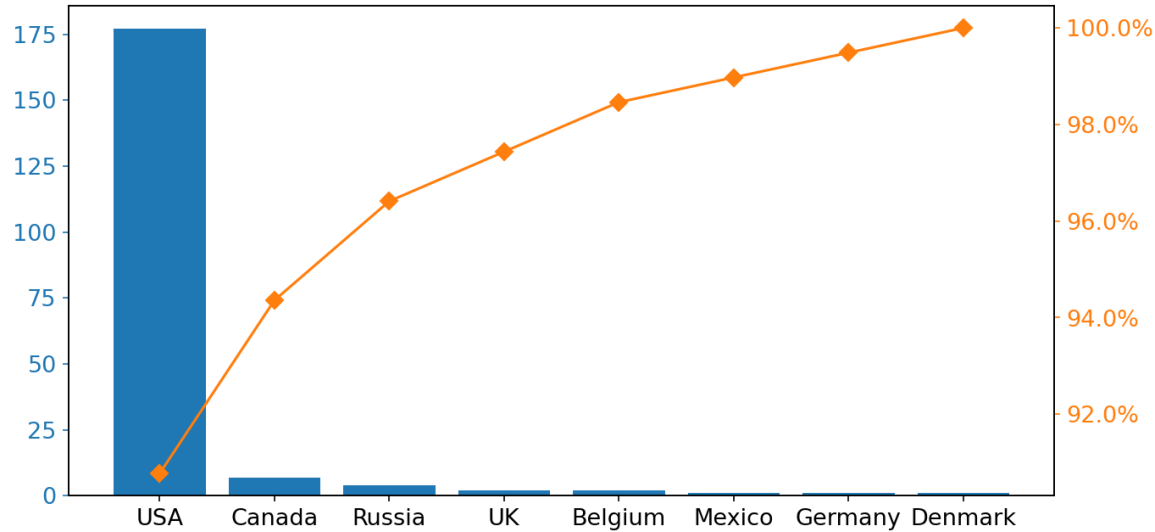
axes[x].grid(True)

fig.suptitle('Employment By Industry By City', fontsize=20)

plt.subplots_adjust(wspace = 0.2)

fig.savefig('Multiple Subplots_shared_y.png',dpi=300, format='png', bbox_inches='tight')





#<https://stackoverflow.com/questions/53577630/how-to-make-pareto-chart-in-python>

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto chart contains both [bars](#) and a [line graph](#), where individual values are represented in descending order by bars, and the cumulative total is represented by the line. The chart is named for the [Pareto principle](#), which, in turn, derives its name from [Vilfredo Pareto](#), a noted Italian economist.

Pareto Chart

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter

df = pd.DataFrame({'country': [177.0, 7.0, 4.0, 2.0, 2.0, 1.0, 1.0, 1.0]})
df.index = ['USA', 'Canada', 'Russia', 'UK', 'Belgium', 'Mexico', 'Germany', 'Denmark']

df = df.sort_values(by='country', ascending=False)
df["cumpercentage"] = df["country"].cumsum()/df["country"].sum()*100

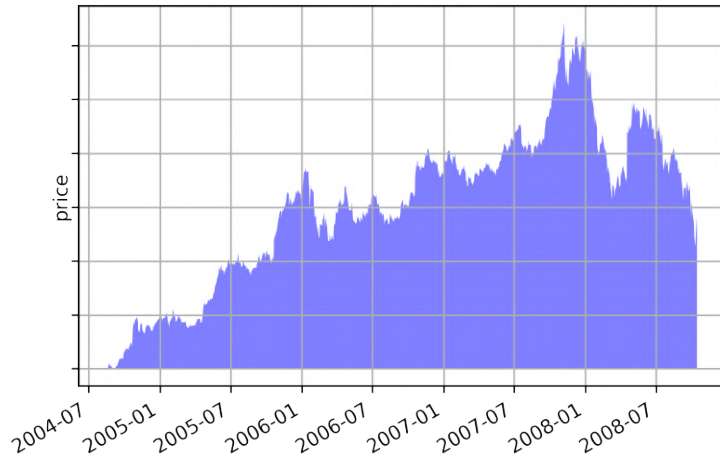
fig, ax = plt.subplots(figsize = (10,5))

ax.bar(df.index, df["country"], color="C0")
ax2 = ax.twinx() #Creating twin axes with shared x-axis
ax2.plot(df.index, df["cumpercentage"], color="C1", marker="D",
        ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())

ax.tick_params(axis="y", colors="C0", labelsize = 13)
ax.tick_params(axis="x", colors="0", rotation = 15)
ax2.tick_params(axis="y", colors="C1", labelsize = 13)

for labels in ax.get_xticklabels():
    labels.set(size = 13, rotation = 0, rotation_mode = 'anchor',
        ha = 'center')

ax.figure.savefig(f'Pareto Chart.png', dpi=150, format='png',
        bbox_inches='tight')
```

https://matplotlib.org/stable/gallery/lines_bars_and_markers/fill_between_alpha.html#sphx-glr-gallery-lines-bars-and-markers-fill-between-alpha-py

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cbook as cbook
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)
```

```
# load up some sample financial data
r = (cbook.get_sample_data('goog.npz',
    np_load=True)['price_data'].view(np.recarray))
```

```
# create two subplots with the shared x and y axes
fig, ax1 = plt.subplots()
```

```
pricemin = r.close.min() #For entering as argument later
```

```
#Optional line plotting with thick lines
```

```
#ax1.plot(r.date, r.close, lw=2)
```

```
p = ax1.fill_between(r.date, pricemin, r.close,
    facecolor='blue', alpha=0.5)
```

```
ax1.grid(True) #Make Gridlines visible
```

```
ax1.set_ylabel('price')
```

```
#Make y ticklabels invisible
```

```
for label in ax1.get_yticklabels():
    label.set_visible(False)
```

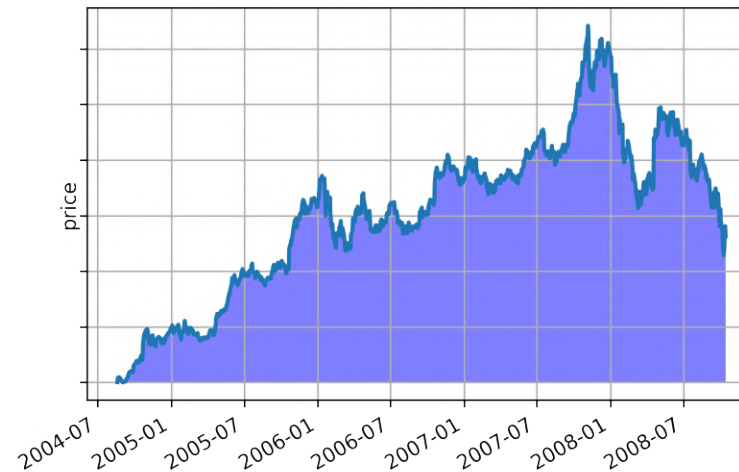
```
fig.suptitle('Google (GOOG) daily closing price')
```

```
fig.autofmt_xdate()
```

```
fig.savefig("Fill Area.png",dpi=300, format='png',
    bbox_inches='tight')
```

```
plt.show()
```

Google (GOOG) daily closing price

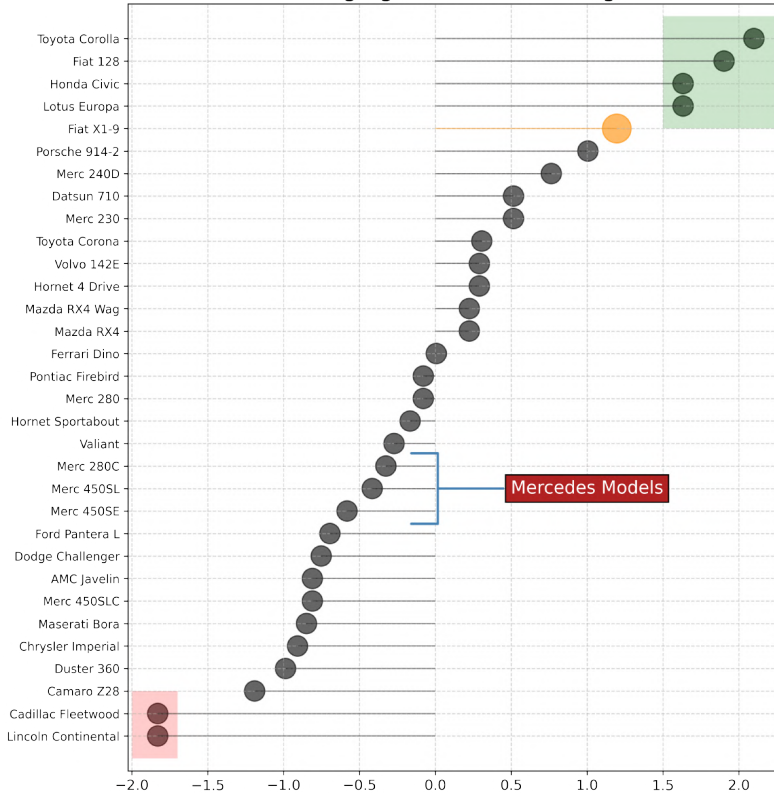


```
#Above plot by uncommenting the optional line plotting
ax1.plot(r.date, r.close, lw=2)
```

Fill Between and Alpha

Diverging Bars of Car Mileage

Back



#<https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/> - 14.- Area-Chart

```
import pandas as pd
```

```
# Prepare Data
```

```
df =
```

```
pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
```

```
x = df.loc[:, ['mpg']]
```

```
df['mpg_z'] = (x - x.mean())/x.std()
```

```
df['colors'] = 'black'
```

```
# color fiat differently
```

```
df.loc[df.cars == 'Fiat X1-9', 'colors'] = 'darkorange'
```

```
df.sort_values('mpg_z', inplace=True)
```

```
df.reset_index(inplace=True)
```

```
# Draw plot
```

```
import matplotlib.patches as patches
```

```
fig, ax = plt.subplots(figsize=(10,12), dpi= 300)
```

```
plt.hlines(y=df.index, xmin=0, xmax=df.mpg_z,
          color=df.colors, alpha=0.4, linewidth=1)
```

```
plt.scatter(df.mpg_z, df.index, color=df.colors,
            s=[600 if x == 'Fiat X1-9' else 300 for x in
               df.cars],
            alpha=0.6)
```

```
plt.yticks(df.index, df.cars)
```

```
plt.xticks(fontsize=12)
```

```
# Annotate
```

```
plt.annotate('Mercedes Models', xy=(0.0, 11.0),
             xytext=(1.0, 11), xycoords='data',
             fontsize=15, ha='center', va='center',
             bbox=dict(boxstyle='square', fc='firebrick',
                       arrowprops=dict(arrowstyle='-[', widthB=2.0,
                                       lengthB=1.5', lw=2.0, color='steelblue'),
                       color='white')
```

```
# Add Patches
```

```
p1 = patches.Rectangle((-2.0, -1), width=.3, height=3,
                       alpha=.2, facecolor='red')
```

```
p2 = patches.Rectangle((1.5, 27), width=.8, height=5,
                       alpha=.2, facecolor='green')
```

```
plt.gca().add_patch(p1)
```

```
plt.gca().add_patch(p2)
```

```
# Decorate
```

```
plt.title('Diverging Bars of Car Mileage',
          fontdict={'size':20})
```

```
plt.grid(linestyle='--', alpha=0.5)
```

```
fig.savefig("aesthetic plot.png", dpi=300, format='png',
            bbox_inches='tight')
```

```
plt.show()
```

Interesting Use Case

- ✓ Horizontal lines
- ✓ Scatter points
- ✓ Annotations
- ✓ Patches

ARTIST

Horizontal lines

- Starts from the base z score of 0 and move in either direction
- Visually appealing and helps mapping the car model label with the scatter point

Scatter points

The scatter points with size 300 for all cars except Fiat X1-9. x position indicates the z score in terms of mileage.

Annotations

To markout the Mercedes models

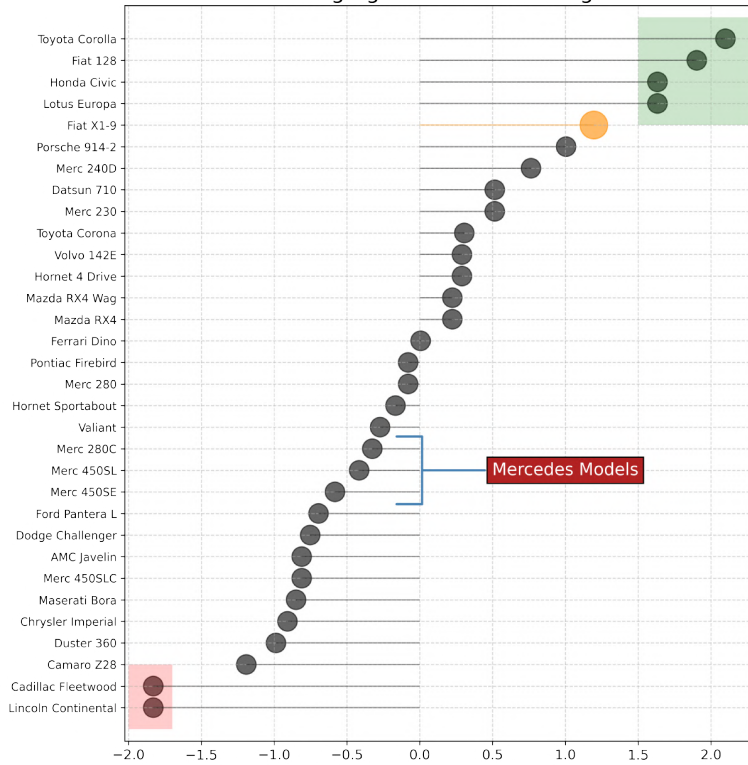
Patches

To highlight the cars at each of the extremes

Raw Data – Next Slide

Diverging Bars of Car Mileage

Back



#<https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/> - 14.-Area-Chart

Raw Data Source Link:

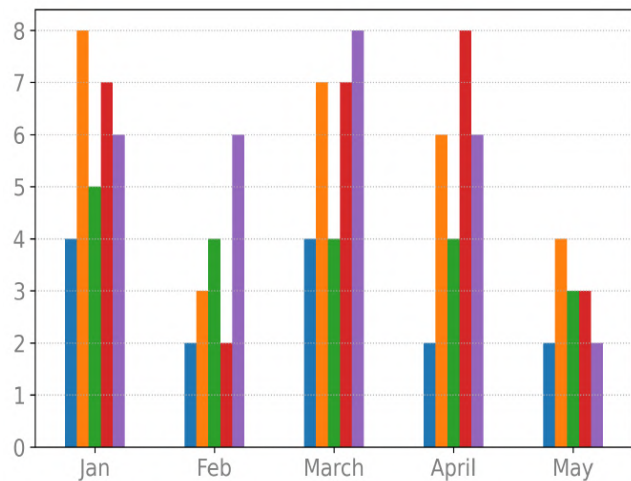
<https://github.com/selva86/datasets/raw/master/mtcars.csv>

Raw Data : Table on the right

- Copy onto excel sheet and save as csv. Use `pd.read_csv` in jupyter notebook.
- Alternately, copy onto excel sheet and Use Text to columns in Data Tab. Proceed to R Studio for EDA in R.

```
"mpg","cyl","dis","hp","drat","wt","qsec","vs","am","gear","carb","fast","cars","carname"
4.58257569495584,6,160,110,3.9,2.62,16.46,0,1,4,4,1,"Mazda RX4","Mazda RX4"
4.58257569495584,6,160,110,3.9,2.875,17.02,0,1,4,4,1,"Mazda RX4 Wag","Mazda RX4 Wag"
4.77493455452533,4,108,93,3.85,2.32,18.61,1,1,4,1,1,"Datsun 710","Datsun 710"
4.62601340248815,6,258,110,3.08,3.215,19.44,1,0,3,1,1,"Hornet 4 Drive","Hornet 4 Drive"
4.32434966208793,8,360,175,3.15,3.44,17.02,0,0,3,2,1,"Hornet Sportabout","Hornet Sportabout"
4.25440947723653,6,225,105,2.76,3.46,20.22,1,0,3,1,1,"Valiant","Valiant"
3.78153408023781,8,360,245,3.21,3.57,15.84,0,0,3,4,0,"Duster 360","Duster 360"
4.93963561409139,4,146.7,62,3.69,3.19,20,1,0,4,2,1,"Merc 240D","Merc 240D"
4.77493455452533,4,140.8,95,3.92,3.15,22.9,1,0,4,2,1,"Merc 230","Merc 230"
4.38178046004133,6,167.6,123,3.92,3.44,18.3,1,0,4,4,1,"Merc 280","Merc 280"
4.2190046219458,6,167.6,123,3.92,3.44,18.9,1,0,4,4,1,"Merc 280C","Merc 280C"
4.04969134626332,8,275.8,180,3.07,4.07,17.4,0,0,3,3,1,"Merc 450SE","Merc 450SE"
4.15932686861708,8,275.8,180,3.07,3.73,17.6,0,0,3,3,1,"Merc 450SL","Merc 450SL"
3.89871773792359,8,275.8,180,3.07,3.78,18.0,0,0,3,3,0,"Merc 450SLC","Merc 450SLC"
3.22490309931942,8,472,205,2.93,5.25,17.98,0,0,3,4,0,"Cadillac Fleetwood","Cadillac Fleetwood"
3.22490309931942,8,460,215,3.5424,17.82,0,0,3,4,0,"Lincoln Continental","Lincoln Continental"
3.83405790253616,8,440,230,3.23,5.345,17.42,0,0,3,4,0,"Chrysler Imperial","Chrysler Imperial"
5.69209978830308,4,78.7,66,4.08,2.2,19.47,1,1,4,1,1,"Fiat 128","Fiat 128"
5.51361950083609,4,75.7,52,4.93,1.615,18.52,1,1,4,2,1,"Honda Civic","Honda Civic"
5.82237065120385,4,71.1,65,4.22,1.835,19.9,1,1,4,1,1,"Toyota Corolla","Toyota Corolla"
4.63680924774785,4,120.1,97,3.7,2.465,20.01,1,0,3,1,1,"Toyota Corona","Toyota Corona"
3.93700393700591,8,318,150,2.76,3.52,16.87,0,0,3,2,0,"Dodge Challenger","Dodge Challenger"
3.89871773792359,8,304,150,3.15,3.435,17.3,0,0,3,2,0,"AMC Javelin","AMC Javelin"
3.64691650576209,8,350,245,3.73,3.84,15.41,0,0,3,4,0,"Camaro Z28","Camaro Z28"
4.38178046004133,8,400,175,3.08,3.845,17.05,0,0,3,2,1,"Pontiac Firebird","Pontiac Firebird"
5.22494019104525,4,79,66,4.08,1.935,18.9,1,1,4,1,1,"Fiat X1-9","Fiat X1-9"
5.09901951359278,4,120.3,91,4.43,2.14,16.7,0,1,5,2,1,"Porsche 914-2","Porsche 914-2"
5.51361950083609,4,95.1,113,3.77,1.513,16.9,1,1,5,2,1,"Lotus Europa","Lotus Europa"
3.97492138287036,8,351,264,4.22,3.17,14.5,0,1,5,4,0,"Ford Pantera L","Ford Pantera L"
4.43846820423443,6,145,175,3.62,2.77,15.5,0,1,5,6,1,"Ferrari Dino","Ferrari Dino"
3.87298334620742,8,301,335,3.54,3.57,14.6,0,1,5,8,0,"Maserati Bora","Maserati Bora"
4.62601340248815,4,121,109,4.11,2.78,18.6,1,1,4,2,1,"Volvo 142E","Volvo 142E"
```

Raw Data



Back

```
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np
```

```
# setting font size to 30
plt.style.use('ggplot')
```

```
plt.rcParams.update({'font.size': 20})
```

```
#Preparing the sample data
```

```
a=np.array([[4,8,5,7,6],
            [2,3,4,2,6],
            [4,7,4,7,8],
            [2,6,4,8,6],
            [2,4,3,3,2]])
```

```
df=DataFrame(a, columns=['a','b','c','d','e'],
             index=['Jan','Feb','March','April','May'])
```

```
r = df.plot(kind='bar') #Using Pandas plotting function
```

```
#Turn off the minor ticks
plt.minorticks_off()
```

```
# Turn on the grid
plt.grid(which='major', linestyle='-', linewidth='0.5',
        color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5',
        color='black')
```

```
#Display the y axis grid lines
plt.grid(True, axis = 'y', alpha = 0.95, ls = ':')
```

```
#Instantiating axes object to use methods
ax = plt.gca()
```

```
#Adding Legend
ax.legend(bbox_to_anchor=(1.05, 0.5), loc='center left',
        fontsize = 15)
```

```
#Setting xtick label properties
[ticklabel.set(rotation =0, ha = 'center', fontsize = 15) for
ticklabel in ax.get_xticklabels()]
```

```
#Setting ytick parameters
ax.yaxis.set_tick_params(rotation =0, labelsz = 15 )
```

```
[ticks.set(alpha = 0.5) for ticks in ax.get_xticklabels()]
[ticks.set(alpha = 0.5) for ticks in ax.get_yticklabels()]
```

```
#Setting figure properties
fig = plt.gcf()
fig.set(figsize = 5, figwidth =8)
```

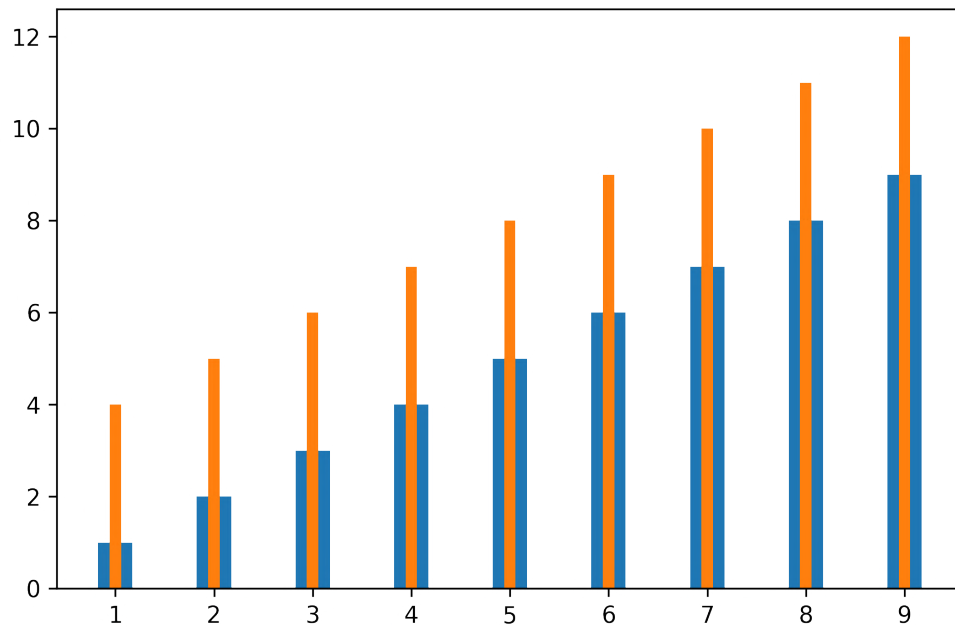
```
fig.savefig("Barplot2.png",dpi=300, format='png',
bbox_inches='tight')
```

```
plt.show()
```

Also check :

- https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py
- <https://www.pythoncharts.com/matplotlib/grouped-bar-charts-matplotlib/>
- To get a list of all containers in the axes, use `ax.containers`
<https://stackoverflow.com/questions/62189677/how-to-get-a-barcontainer-object-from-an-axessubplot-object-in-matplotlib>
- To get list of all artists in the axes, use `ax.get_children()`
- To get list of the attributes and methods of axes object, use `print(dir(ax))`

Grouped Bar Chart Pandas plotting



[#https://stackoverflow.com/questions/23293011/how-to-plot-a-superimposed-bar-chart-using-matplotlib-in-python](https://stackoverflow.com/questions/23293011/how-to-plot-a-superimposed-bar-chart-using-matplotlib-in-python)

```
a = range(1,10)
b = range(4,13)
ind = np.arange(len(a))
```

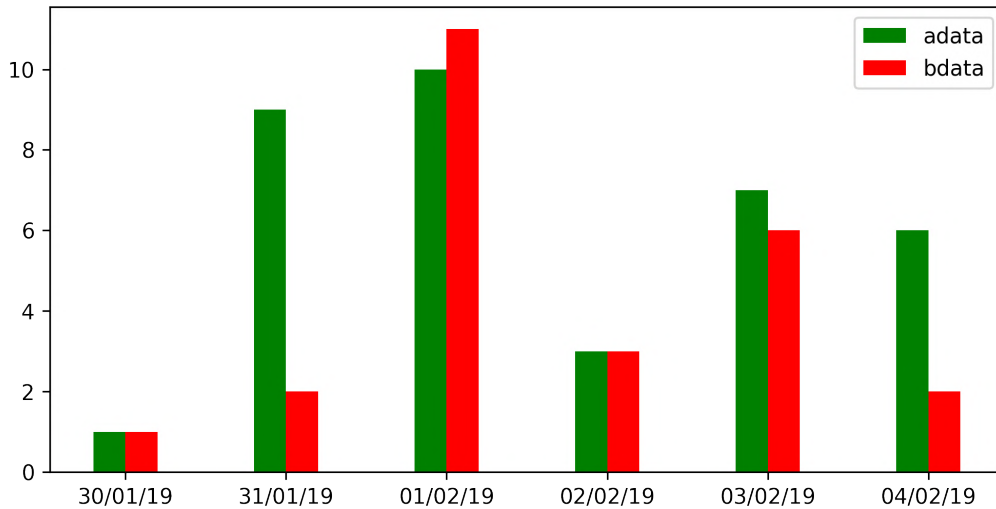
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.bar(x=ind, height=a, width=0.35, align='center')
ax.bar(x=ind, height=b, width=0.35/3, align='center')
```

```
plt.xticks(ind, a)
```

```
plt.tight_layout()
fig.savefig('Bullet_chart.png', dpi = 300, transparent = True)
plt.show()
```

Matplotlib Bar Plots

- Bars of different widths



#<https://stackoverflow.com/questions/59738557/date-format-issues-in-plot-ticks-with-matplotlib-dates-and-datestr2num>

Context : `datestr2num` calls (wrapper) `dateutil` but currently does not forward all keyword arguments. For instance, `dayfirst` is not forwarded by `datestr2num`. Instead, month first is always considered. This creates problems incase of ambiguous date formats. The `date2strnum` is not robust to deal consistently with dates in string form. We therefore first parse date string with Python `datetime` module functions, convert back to date string with month first string format ('%m/%d/%Y') and then pass it to `datestr2num` to convert to matplotlib dates.

Matplotlib Parsing Date strings

- To process strings, use `datetime` functions
 - Convert string to datetime object using `strptime`
 - Convert datetime object back to string using `strftime`
- Convert the string from previous step to matplotlib.date using `datestr2num`

Code Start

```
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.dates import (
    DateFormatter, AutoDateLocator, AutoDateFormatter, datestr2num
)
```

```
days = [
    '30/01/2019', '31/01/2019', '01/02/2019',
    '02/02/2019', '03/02/2019', '04/02/2019'
]
adata = [1, 9, 10, 3, 7, 6]
bdata = [1, 2, 11, 3, 6, 2]
```

#converting/parsing string to python datetime object using `strptime`
#and then converting back to string using `strftime`

```
x = datestr2num([
    datetime.strptime(day, '%d/%m/%Y').strftime('%m/%d/%Y')
    for day in days
])
```

#Alternately instead of `datestr2num`, we can directly use `date2num` on the datetime object
#x = date2num([datetime.strptime(day, '%d/%m/%Y') for day in days])

```
w = 0.10
```

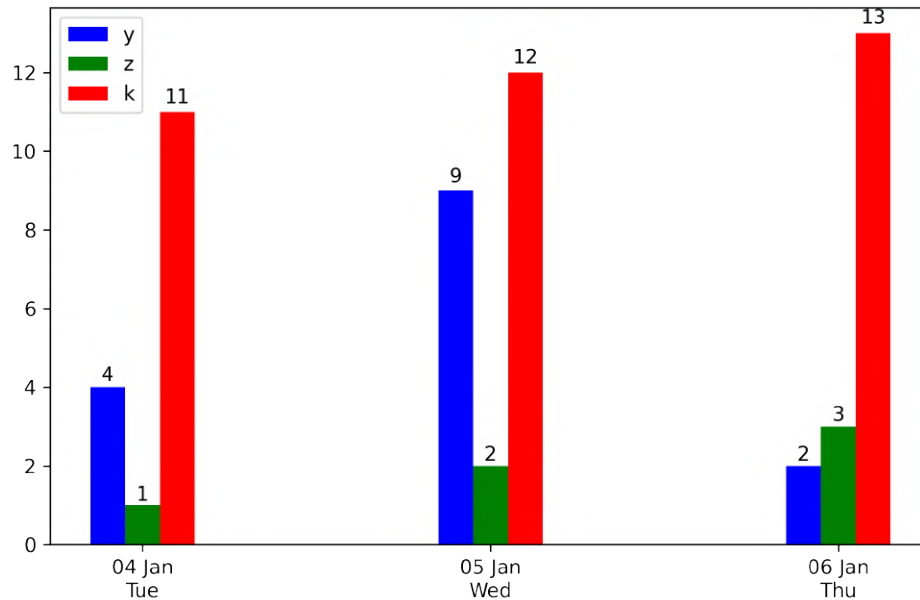
```
fig = plt.figure(figsize=(8, 4))
ax = fig.add_subplot(111)
```

#align:'center' is important and width as 2*w in conjunction with x value ensures adjacent bars
ax.bar(x - w, adata, width=2 * w, color='g', align='center', tick_label=days, label = 'adata')
ax.bar(x + w, bdata, width=2 * w, color='r', align='center', tick_label=days, label = 'bdata')

```
ax.xaxis.set_major_locator(AutoDateLocator(minticks=3, interval_multiples=False))
ax.xaxis.set_major_formatter(DateFormatter("%d/%m/%y"))
ax.legend() #Adding legend - automatic detection of handles, labels
```

```
fig.savefig('Processing_dates_in_string_as_xaxis.png', dpi = 300)
plt.show()
```

Code End



#Check the above [discussion link](#).

I have proposed a solution over here and extended the solution to handle any number of groups. Follow the next few slides.

Context :

- Three Bar plots plotted for the same datetime index
- Each Bar plot corresponds to values for a single group against date values
- Ensure no overlap between the bars
- Bars should be symmetrically positioned in either side of date tick value

Matplotlib Multiple Group Bar plots

- Date Formatting in x axis
- No overlapping between bars

```
import matplotlib.pyplot as plt
from matplotlib.dates import date2num
import datetime
import matplotlib.dates as mdates

x = [
    datetime.datetime(2011, 1, 4, 0, 0),
    datetime.datetime(2011, 1, 5, 0, 0),
    datetime.datetime(2011, 1, 6, 0, 0)
]
x = date2num(x)    #Converting datetime objects to matplotlib dates

y = [4, 9, 2]
z = [1, 2, 3]
k = [11, 12, 13]

w = 0.05
fig, ax = plt.subplots(figsize = (8,5))
bars1 = ax.bar(x-2*w, y, width=2*w, color='b', align='center', label = 'y')
bars2 = ax.bar(x, z, width=2*w, color='g', align='center', label = 'z')
bars3 = ax.bar(x+2*w, k, width=2*w, color='r', align='center', label = 'k')

ax.xaxis.set_major_locator(mdates.DayLocator())
formatter = mdates.DateFormatter('%d %b\n%a')
ax.xaxis.set_major_formatter(formatter)
ax.legend()

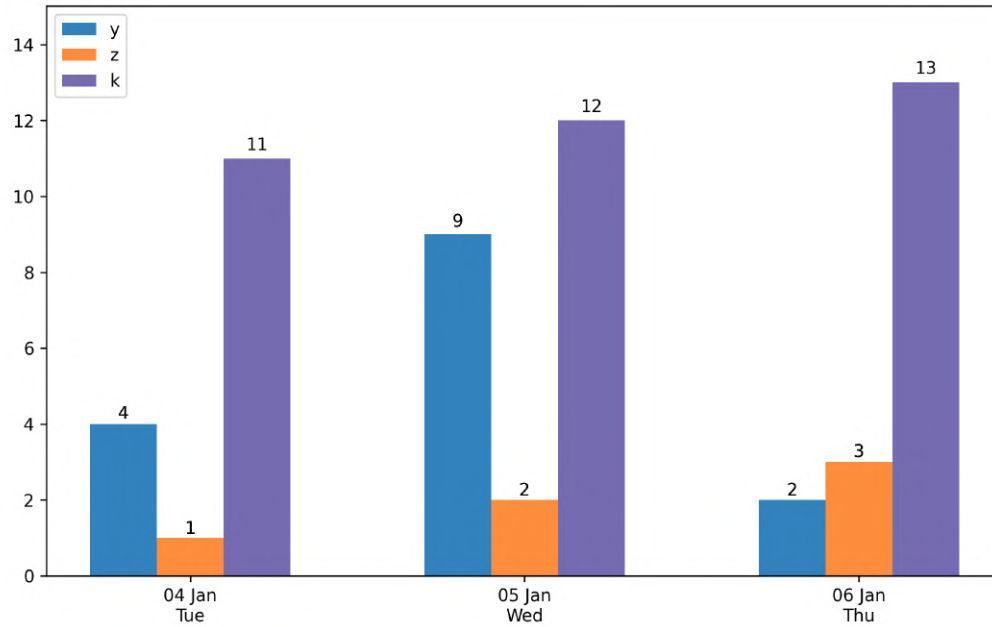
def autolabel(rects):
    for rect in rects:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.01*h, '%d'%int(h),
                ha='center', va='bottom')
autolabel(bars1)
autolabel(bars2)
autolabel(bars3)

plt.show()
```


Matplotlib Multiple Group Bar plots

User Defined function for multiple scenarios

- Date Formatting in x axis
- No overlapping between bars



#Check the [discussion link](#)

Context :

- Three Bar plots plotted for the same datetime index
- Each Bar plot corresponds to values for a single group against date values
- Ensure no overlap between the bars
- Bars should be symmetrically positioned in either side of date tick value
- Date time formatting in x axis

Below is the structure of the code script scalable to any number of groups.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from matplotlib.colors import ListedColormap
from matplotlib.colors import LinearSegmentedColormap
from itertools import cycle, islice #Using iterator to cycle colors and boxstyle
```

Code start

```
def get_colors(y_list, cmap = 'tab20c'):
```

User defined function

```
    """
```

```
    Args :
```

```
        y_list : list containing multiple y series each representing a categorical variable to be plotted
```

```
        cmap = Use next(color_list) to get the next color in each iteration
```

```
    Returns :
```

```
        color_list : a cyler object using islice to infinitely loop over a list of colors but always start from fixed position
```

```
    """
```

```
    listedcolormap = ListedColormap(cm.get_cmap(cmap,256)(np.linspace(0,0.6,len(y_list))))
```

```
    color_list = islice(cycle(listedcolormap.colors),0,None)
```

```
    return color_list
```

Code continue Next slide

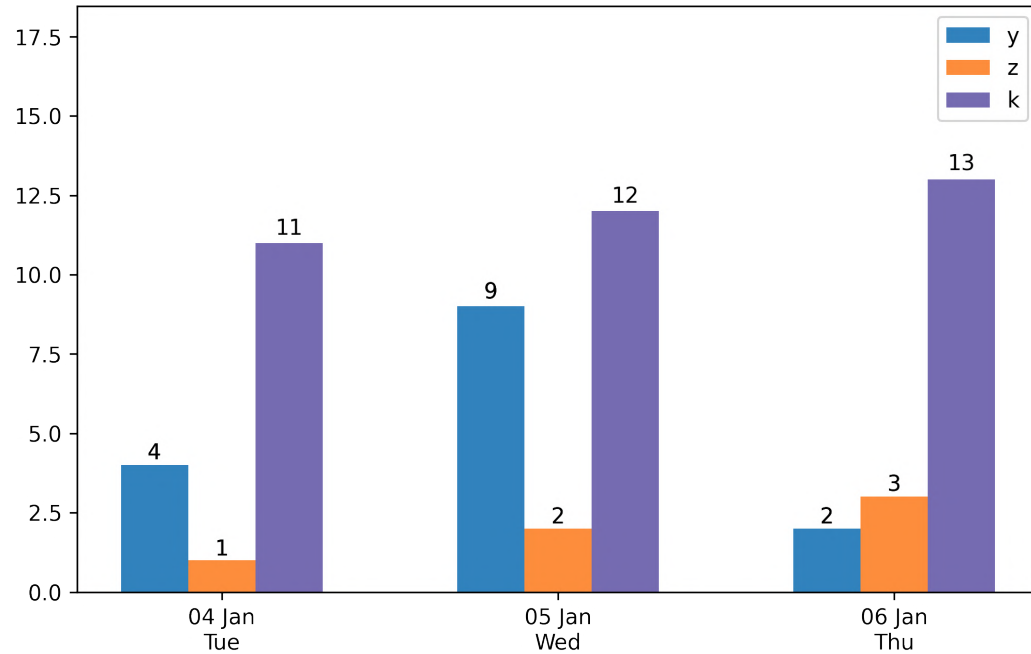
User
Defined
Functions

1. Use cyler object to access colors in loop

2. Make bar plots for proper placement of bars and date time formatting of x axis

Call the
function

3.make_barplots(*args)



Creating User defined Function for returning the desired plot

```
def make_barplots(x, y_list, y_labels = [], w = 0.05, cmap = 'tab20c', style = 'seaborn-
whitegrid'):
```

Code continues from Prev slide

"""

Args :

x : list of x values of the data type matplotlib dates
y_list : list of groupwise values corresponding to the x values
y_labels : list of group labels in the format string
style : plot style to use
cmap : colormap name from the list of matplotlib colormaps
w : half of the width of bar (The relationship is Bar width = 2*w)

If the input data is a Dataframe df with index of type datetimeindex and we want the columns to be plotted, then

below conversion methods helpful :

```
x = date2num(df.index)
y_list = df.transpose().to_numpy().tolist()
```

Returns :

fig : Figure containing the axes and barplot
ax : Axes containing the barplots
bars_list : List containing barcontainer objects

Raises :

Warning : Legend not shown if the y_labels argument is empty.
"""

Code continues
to Next slide

User Defined
Functions

cycler object to access colors in
in a loop
To Make bar plots

Call the
function

make_barplots
(*args)

Logic for consistent placement of the bars

LOGIC

Defining the problem :

- A bar belonging to a group should appear at fixed position versus its xtick
- Bars of different group should not overlap

What we want ?

Symmetrical distribution of bar groups across a tick location.

The group of bar at each of the x axis value should be symmetrically distributed around the tick. There should be equal no. of bar groups on either side of the x-axis tick location.

How will we achieve ? -- It's a problem of placement of bars

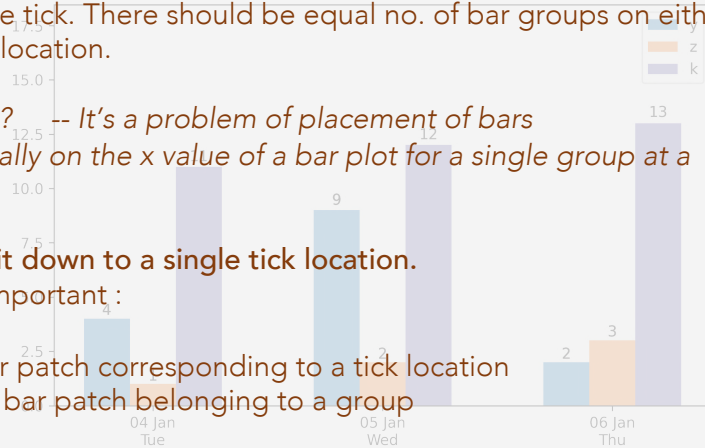
We will focus specifically on the x value of a bar plot for a single group at a time.

Let's further narrow it down to a single tick location.

Three locations are important :

- Tick Location
- Start of the first bar patch corresponding to a tick location
- Actual x value of a bar patch belonging to a group

1. Tick Location is matplotlib date(a floating point) and is fixed in our context.
2. Start Location will be offset from Tick location depending on number of bars (number of groups) for a single tick
3. For even number of groups lets consider 4, the start location will be offset by the width of 2 bars.
4. For odd number of groups lets consider 5, the start location will be offset by the width of 2.5 bars.
5. Actual x value will be offset from the Start location depending on the group sequence



```
plt.style.use(style)
fig, ax = plt.subplots(figsize = (10,6))
bars_list = []
```

Code continues from Prev slide

```
# Creating color_list as a cycler object for cycling through the colors
```

```
color_list = get_colors(y_list, cmap = cmap)
```

```
# Creating bar plots iteratively from the list y_list containing the list of values
```

```
for i,y in enumerate(y_list):
```

```
    n = len(y_list)                # number of groups
```

```
    p = n//2                      # p to be used for finding the start location
```

```
    x_offset = 2*w*n//2+w*n%2      # calculating offset from tick location to locate start location
```

```
# Plotting the bar plot for a single group
```

```
# The start location will be further offset
```

```
# by bar widths multiplied by the index position (i) of the group within the y_list
```

```
bars = ax.bar(x-x_offset+i*2*w, y, width=2*w, color=next(color_list), align='edge')
```

```
bars_list.append(bars)
```

```
# Creating a function to add data labels above bars
```

```
def autolabel(rects):
```

```
    for rect in rects:
```

```
        h = rect.get_height()
```

```
        ax.text(rect.get_x()+rect.get_width()/2., 1.01*h, '%d'%int(h),
```

```
                ha='center', va='bottom')
```

```
# Adding data labels to bars by enumerating over bar containers and
```

```
for i, bars in enumerate(bars_list):
```

```
    autolabel(bars)
```

```
# Setting label property of bars
```

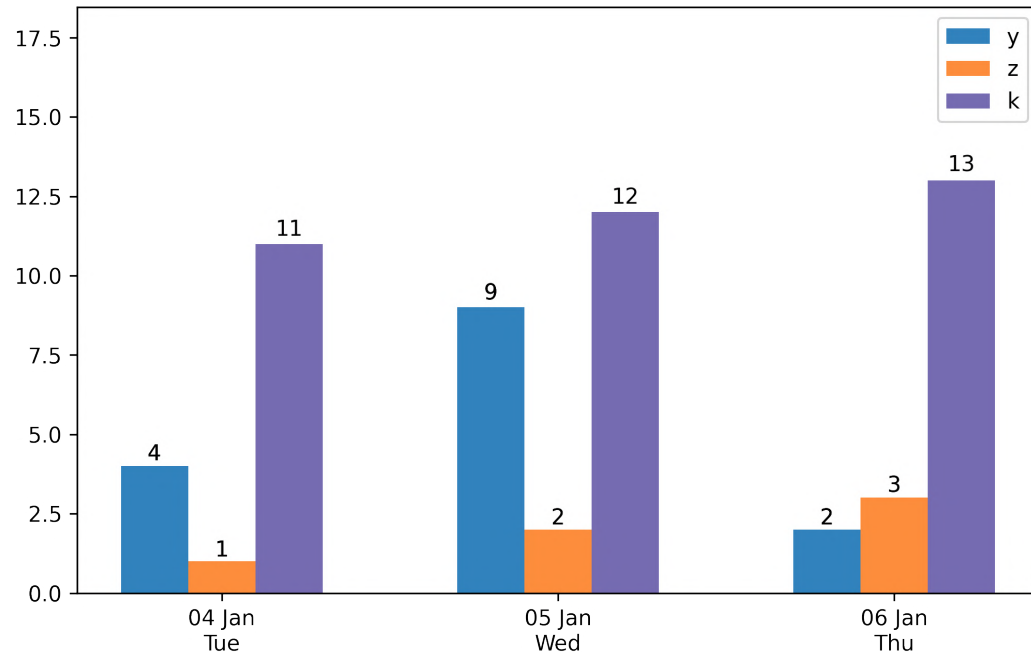
```
# for automatic detection of handles with labels for legend
```

```
if len(y_labels) == len(y_list): #Only if labels are provided while calling function
```

```
    bars.set_label(y_labels[i])
```

Code continues to Next slide

User Defined Functions	cycler object to access colors in a loop Make bar plots	Call the function	make_barplots(*args)
------------------------	--	-------------------	----------------------



```
locator = mdates.DayLocator() #Creating locator object
formatter = mdates.DateFormatter('%d %b\n%a') #Creating locator object
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)

ax.xaxis.grid(False) #Making the xgrid lines disappear

ax.set_ylim(ax.get_ylim()[0]*1.2,ax.get_ylim()[1]*1.1) #Resetting y limits for ample margins

if len(y_labels) == len(y_list):
    #handles, labels = ax.get_legend_handles_labels()
    #print(handles)
    #print(labels)
    ax.legend(loc = 'best')
else :
    print('labels for the bars not entered. Hence, Legend missing.')

fig.savefig('Multiple_barsplot_with_xaxis_datetickers.png', dpi = 300)
return fig, ax, bars_list #returns figure, axes, list containing bar container objects
```

Code continues form prev slide

```
# Preparing the data and making bar plot by using the user defined function make_barplots()
import matplotlib.pyplot as plt
from matplotlib.dates import date2num
import datetime
import matplotlib.dates as mdates

x = [ datetime.datetime(2011, 1, 4, 0, 0), datetime.datetime(2011, 1, 5, 0, 0),
      datetime.datetime(2011, 1, 6, 0, 0)]

x = date2num(x)
y = [4, 9, 2]
z = [1, 2, 3]
k = [11, 12, 13]

fig, ax, bars_list = make_barplots(x, y_list, y_labels, w = 0.1, style = 'default')
```

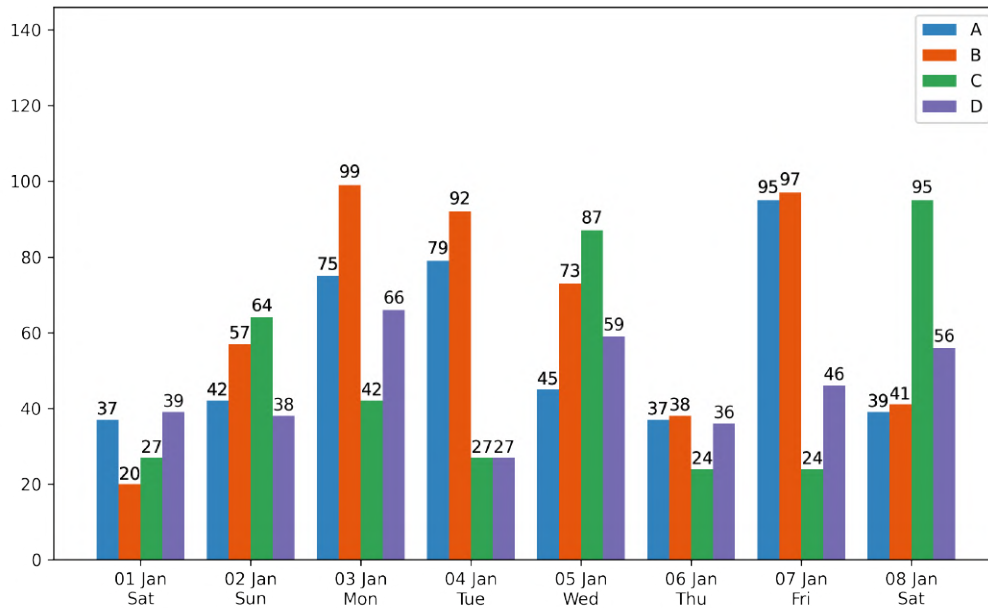
Code End

User
Defined
Functions

cycler object to access colors in
in a loop
Make bar plots

Call the
function

make_barplots
(*args)



```
import pandas as pd
import numpy as np
```

```
#Fixing state for reproducibility
np.random.seed(123456)
```

```
#Creating sample data for plotting using numpy and pandas
```

```
index = pd.date_range("1/1/2000", periods=8)
```

```
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
```

```
df = pd.DataFrame(np.random.randint(20,100, (8,4)), index=index, columns=["A", "B", "C", "D"])
```

```
#Make bar plots using the userdefined function make\_barplots
```

```
make_barplots(x = date2num(df.index), y_list = df.transpose().to_numpy().tolist(), \
y_labels = df.columns, w = 0.1, style = 'default')
```

Creating DataFrames https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe

<https://github.com/pandas-dev/pandas/blob/v1.3.1/pandas/core/frame.py#L456-L10748> line no. 1595

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html#pandas.DataFrame.to_numpy

Alternate way from DataFrame to List

Pandas DataFrame columns are Pandas Series when you pull them out, which you can then call `x.tolist()` on to turn them into a Python list.

```
( df
    .filter(['column_name'])
    .values
    .reshape(1, -1)
    .ravel()
    .tolist()
)
```

DataFrame

	A	B	C	D
2000-01-01	88	82	69	87
2000-01-02	35	47	88	24
2000-01-03	47	93	66	31
2000-01-04	51	56	65	71
2000-01-05	31	41	94	25
2000-01-06	29	84	60	91
2000-01-07	48	66	81	99
2000-01-08	84	96	75	52

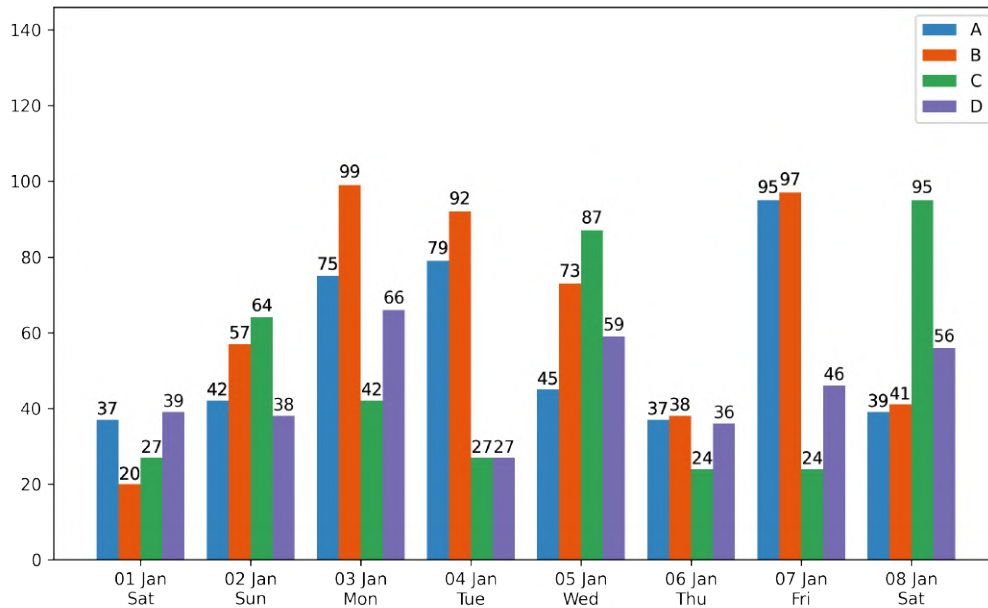
DataFrame - Transpose

	2000-01-01	2000-01-02	2000-01-03	2000-01-04	2000-01-05	2000-01-06	2000-01-07	2000-01-08
A	88	35	47	51	31	29	48	84
B	82	47	93	56	41	84	66	96
C	69	88	66	65	94	60	81	75
D	87	24	31	71	25	91	99	52

Matplotlib Multiple Group Bar plots

Demo using the User defined function [make_barplots](#)

- Date Formatting in x axis
- No overlapping between bars
- Number of bar groups = 4



```
import pandas as pd
import numpy as np
```

```
#Fixing state for reproducibility
np.random.seed(123456)
```

```
#Creating sample data for plotting using numpy and pandas
```

```
index = pd.date_range("1/1/2000", periods=8)
```

```
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
```

```
df = pd.DataFrame(np.random.randint(20,100, (8,4)), index=index, columns=["A", "B", "C", "D"])
```

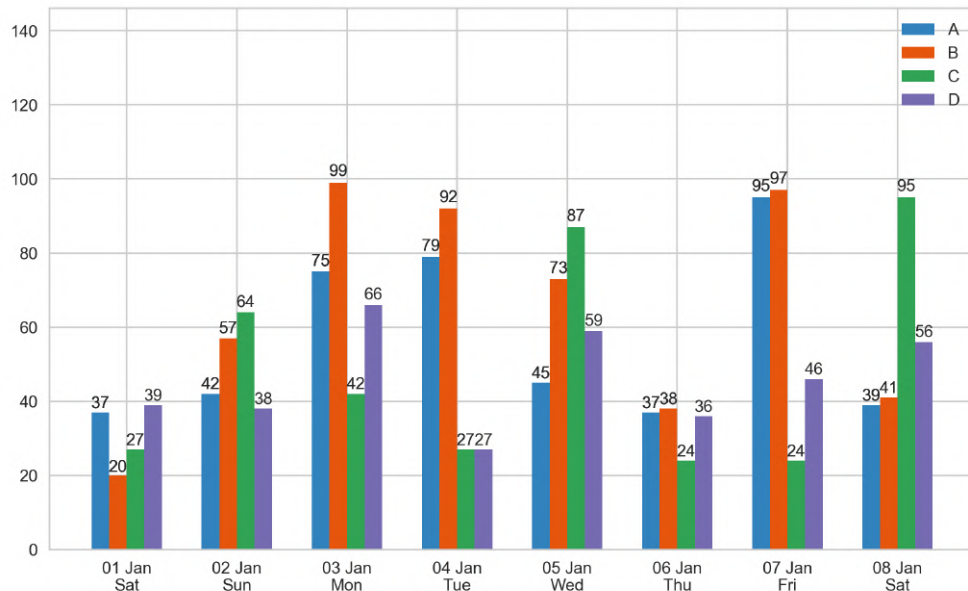
Create sample data

```
#Make bar plots using the userdefined function make\_barplots
```

```
make_barplots(x = date2num(df.index), y_list = df.transpose().to_numpy().tolist(), \
y_labels = df.columns, w = 0.1, style = 'default')
```

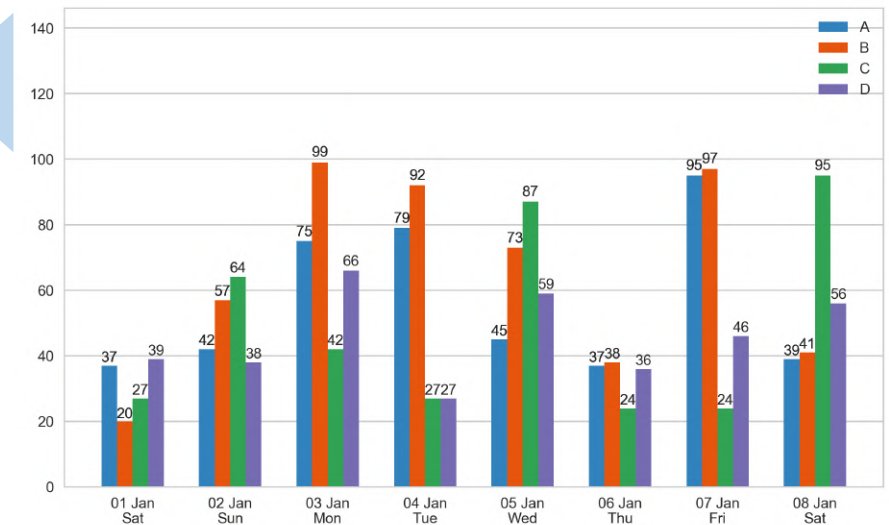
Code End

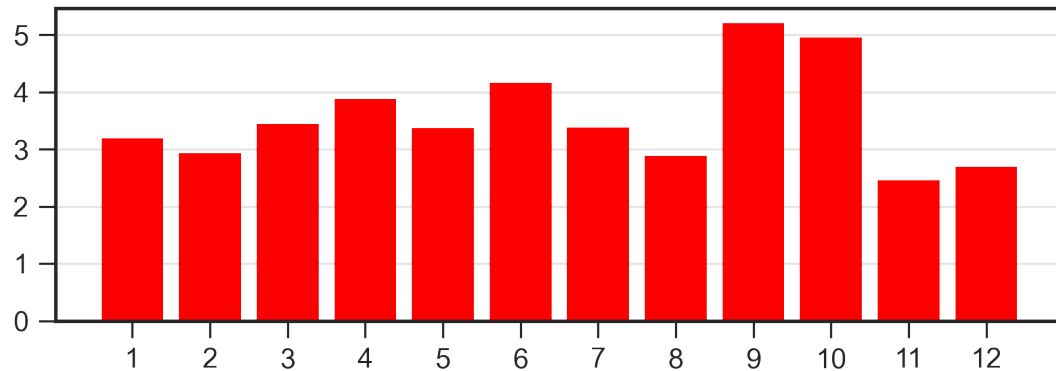
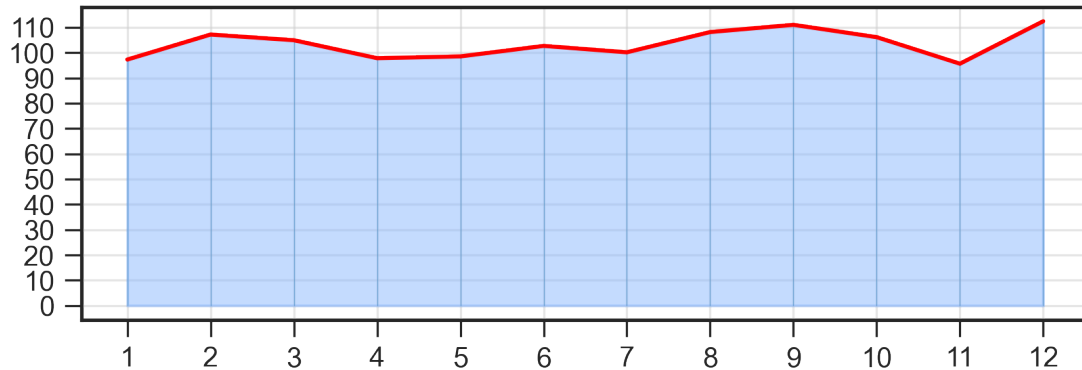
Creating DataFrames https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe



```
style : 'seaborn-whitegrid'
ax.xaxis.grid(False)
```

```
style : 'seaborn-whitegrid'
```





Multiple Subplots Line plot + Bar plot Shared Axis

- Categorical Bar and Line plot across multiple subplots
- Ticker.MultipleLocator(10)
- Axes.fill_between()
- Axes.vlines()

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker
```

```
plt.rcParams()
fig = plt.figure()
```

```
# generate sample data for this example
```

```
count = 12
xs = list(range(1,count+1))
prng = np.random.default_rng(123456)
ys_bars = prng.normal(loc=3.0,size=len(xs))
ys_lines = prng.normal(loc=5.0,size=len(xs),scale=0.5)
```

```
# this is the axes on the top
```

```
ax1= plt.subplot(211)
```

```
# plot the same numbers but multiplied by 20
```

```
ax1.plot(xs,ys_lines*20,color='red')
```

```
# order is important when setting ticks.
```

```
# Ticks must be set after the plot has been drawn
```

```
#ax1.set_yticks(np.arange(0,101,10))
```

```
ax1.yaxis.set_major_locator(ticker.MultipleLocator(10))
```

```
# set ticks for the common x axis (bottom)
```

```
ax1.xaxis.set_ticks(xs)
```

```
ax1.grid(True, alpha = 0.5)
```

```
ax1.fill_between(xs,y1 = ys_lines*20, alpha = 0.5, color = 'xkcd:carolina blue')
```

```
ax1.vlines(x=xs, ymin = 0, ymax = ys_lines*20, alpha = 0.5, lw = 0.3)
```

```
ax2= plt.subplot(212)
```

```
ax2.bar(xs,ys_bars,color='red')
```

```
ax2.yaxis.grid(True, alpha = 0.5)
```

```
ax2.set_axisbelow(True)
```

```
ax2.set_xticks(xs)
```

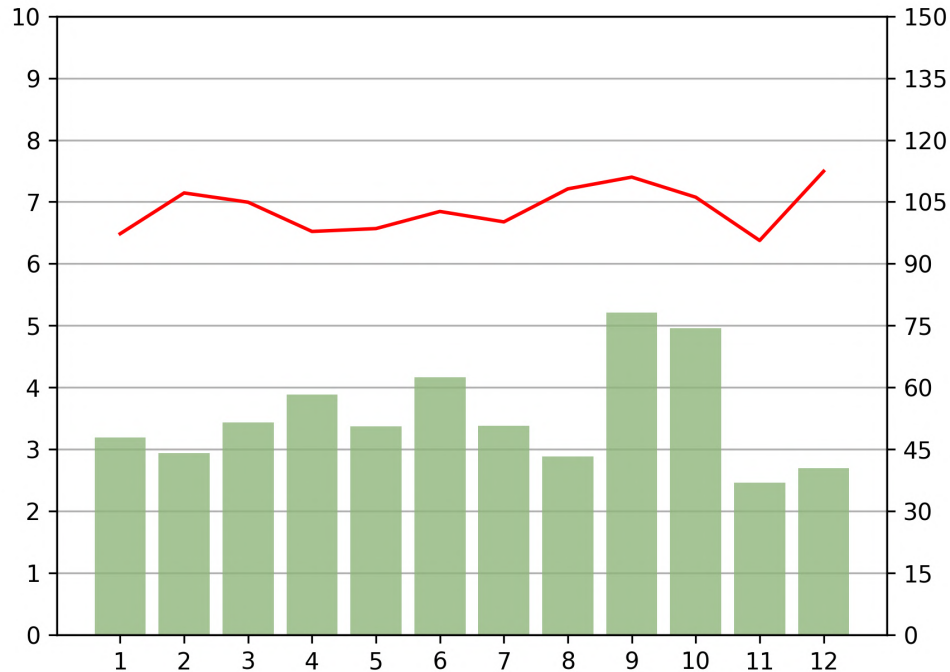
```
ax2.yaxis.set_major_locator(ticker.MultipleLocator(1))
```

```
plt.subplots_adjust(hspace = 0.4)
```

```
fig.savefig('Simple LinePlots_BarPlots_sharedxaxis.png', dpi = 300, transparent = True)
```


Overlaying Charts

- Categorical Bar and Line plot on same Axes
- X axis is numeric but non datetime
- Twin Axes with shared x axis
- Aligning yaxis of the twin axes



Code Start

```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig, ax1 = plt.subplots()
```

```
# generate sample data for this example
```

```
xs = list(range(1,13))
```

```
prng = np.random.default_rng(123456)
```

```
ys_bars = prng.normal(loc=3.0,size=12)
```

```
ys_lines = prng.normal(loc=5.0,size=12,scale=0.5)
```

Check <https://queirozf.com/entries/matplotlib-pyplot-by-example>

Note : Changes have been done visavis the code in above link.

```
ax1.bar(xs,ys_bars,color = 'xkcd:lichen', alpha = 0.8)
```

```
# order is important when setting ticks.
```

```
# Ticks must be set after the plot has been drawn
```

```
ax1.set_yticks(np.arange(0,11,1))
```

```
# define the number of ticks
```

```
NUM_TICKS=11
```

```
# change the tick locator for this axis and set the desired number of ticks
```

```
ax1.yaxis.set_major_locator(plt.LinearLocator(numticks=NUM_TICKS))
```

```
# create the 'twin' axes with the y axis on right
```

```
ax2=ax1.twinx()
```

```
# plot the same numbers but multiplied by 20
```

```
ax2.plot(xs,ys_lines*20,color='red')
```

```
# set the limits for the twin axis
```

```
ymin = (ax2.get_ylim()[1]//100+0.5)*100
```

```
ax2.set_ylim(0, ymin)
```

```
# change the tick locator for this axis and set the desired number of ticks
```

```
ax2.yaxis.set_major_locator(plt.LinearLocator(numticks=NUM_TICKS))
```

```
# set ticks for the common x axis (bottom)
```

```
ax2.xaxis.set_ticks(xs)
```

```
#Show yaxis gridlines
```

```
ax1.yaxis.grid(True)
```

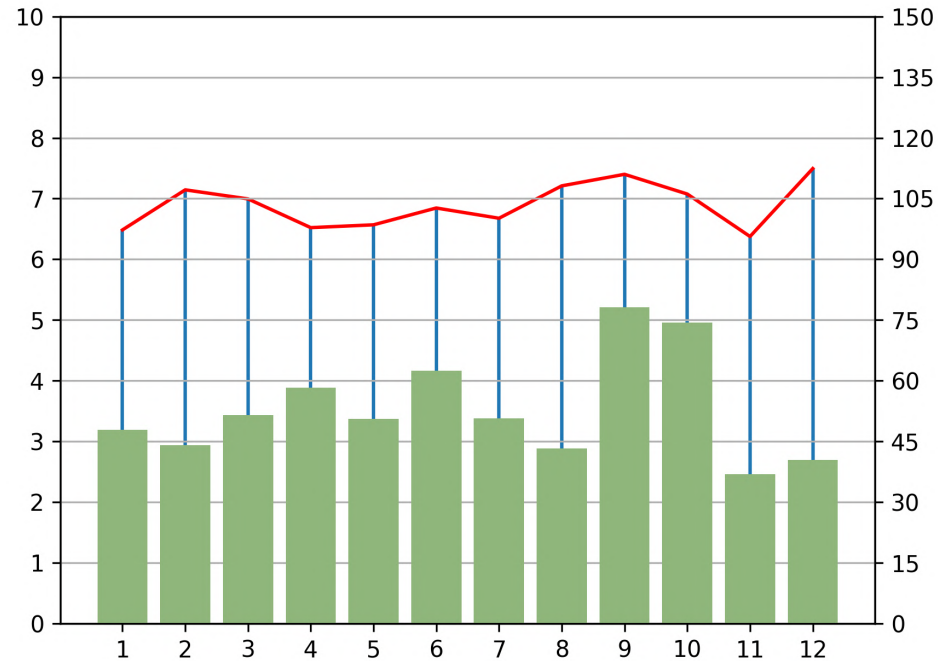
```
ax1.set_axisbelow(True)
```

```
fig.savefig('Barplot_Lineplot.png', dpi = 300, transparent = False)
```

Code End

Overlaying Charts

- Categorical Bar and Line plot on same Axes
- X axis is numeric but non datetime
- Twin Axes with a shared x axis
- Ax.vlines from the line plot



Code Start

```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig, ax1 = plt.subplots()
```

```
# generate sample data for this example
```

```
xs = list(range(1,13))
```

```
prng = np.random.default_rng(123456)
```

```
ys_bars = prng.normal(loc=3.0,size=12)
```

```
ys_lines = prng.normal(loc=5.0,size=12,scale=0.5)
```

Check <https://queirozf.com/entries/matplotlib-pyplot-by-example>

Note : Changes have been done visavis the code in above link.

```
ax1.bar(xs,ys_bars,color = 'xkcd:lichen')
```

```
# order is important when setting ticks.
```

```
# Ticks must be set after the plot has been drawn
```

```
ax1.set_yticks(np.arange(0,11,1))
```

```
# define the number of ticks
```

```
NUM_TICKS=11
```

```
# change the tick locator for this axis and set the desired number of ticks
```

```
ax1.yaxis.set_major_locator(plt.LinearLocator(numticks=NUM_TICKS))
```

```
# create the 'twin' axes with the y axis on the right
```

```
ax2=ax1.twinx()
```

```
# Setting zorder of the twin axes below original axes. Check link1, link2 on zorder
```

```
ax2.set_zorder(ax2.get_zorder()-1)
```

```
ax1.patch.set(visible = False)
```

```
# plot the same numbers but multiplied by 20
```

```
ax2.plot(xs,ys_lines*20,color='red')
```

```
# set the ticks for the twin axis
```

```
#ax2.set_yticks(np.arange(0,101,10))
```

```
ymax = (ax2.get_ylim()[1]//100+0.5)*100
```

```
ax2.set_ylim(0, ymax)
```

```
# change the tick locator for this axis and set the desired number of ticks
```

```
ax2.yaxis.set_major_locator(plt.LinearLocator(numticks=NUM_TICKS))
```

```
# set ticks for the common x axis (bottom)
```

```
ax2.xaxis.set_ticks(xs)
```

```
ax1.yaxis.grid(True)
```

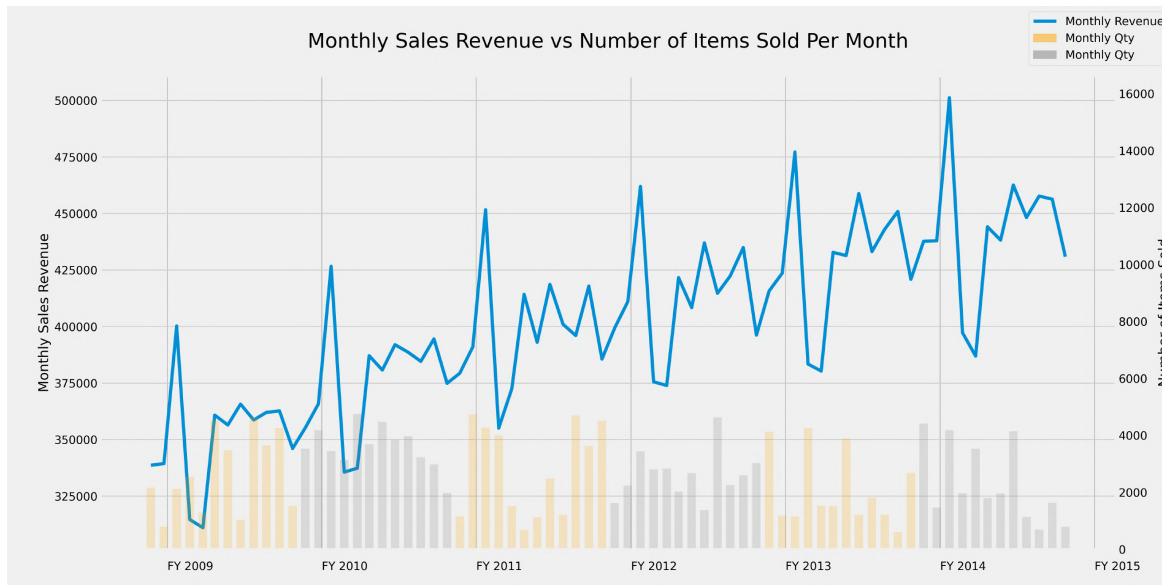
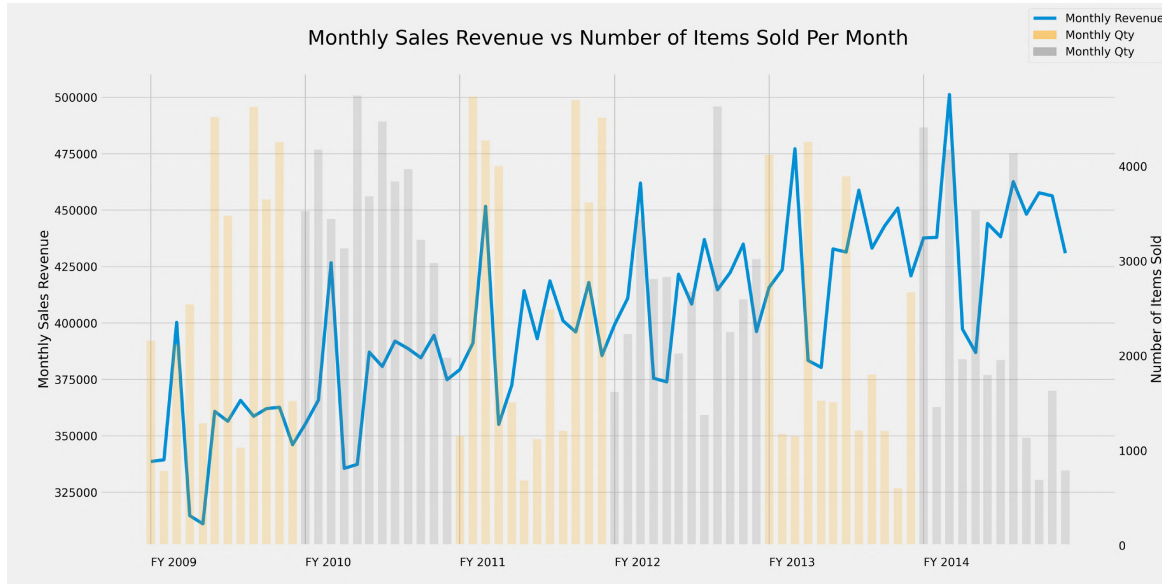
```
ax1.set_axisbelow(True)
```

```
ax2.vlines(x = xs,ymin = 0, ymax = ys_lines*20)
```

```
#Axes.vlines from line plot
```

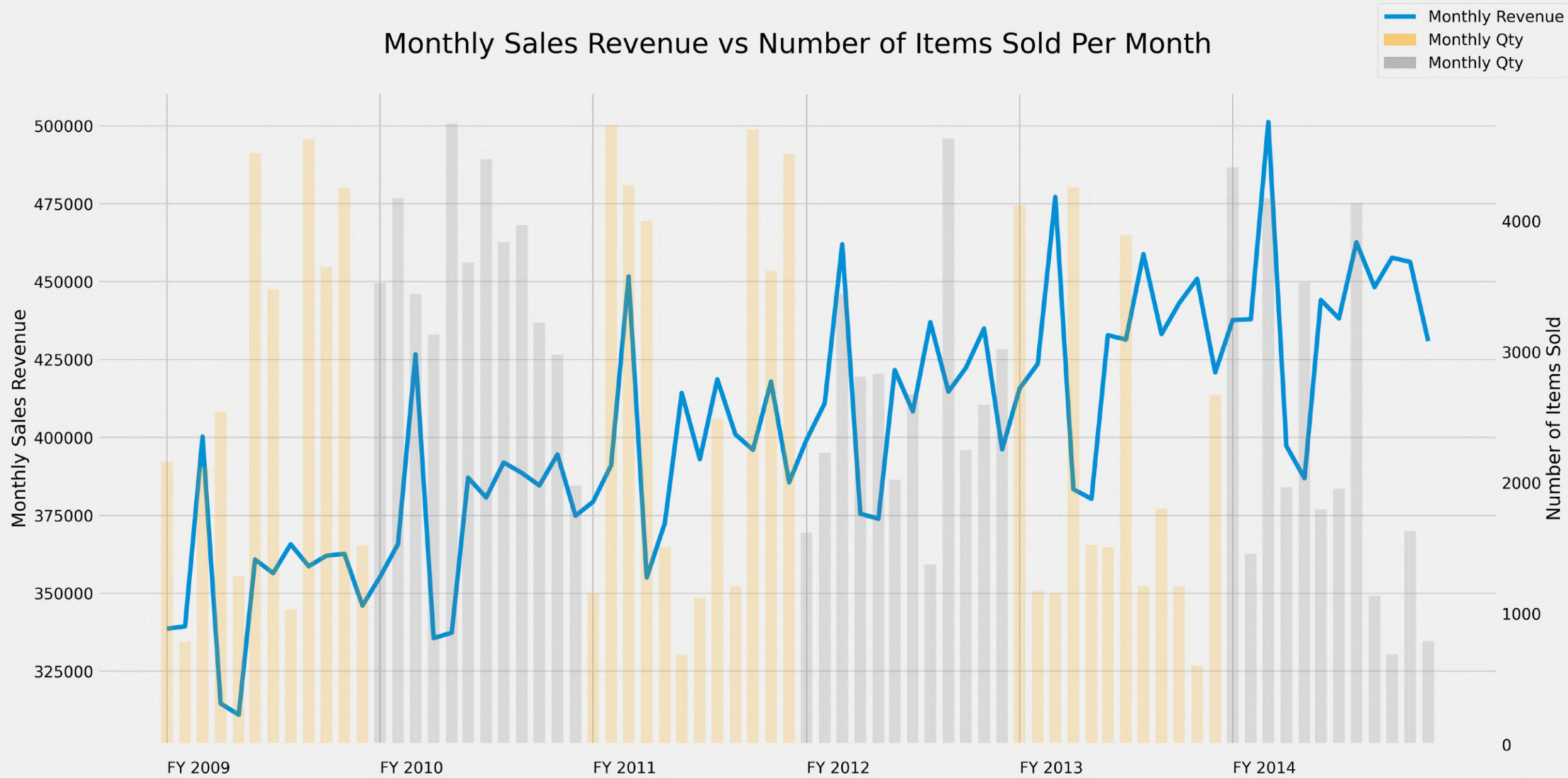
```
fig.savefig('Barplot_Lineplot_2.png', dpi = 300, transparent = False)
```

Code End

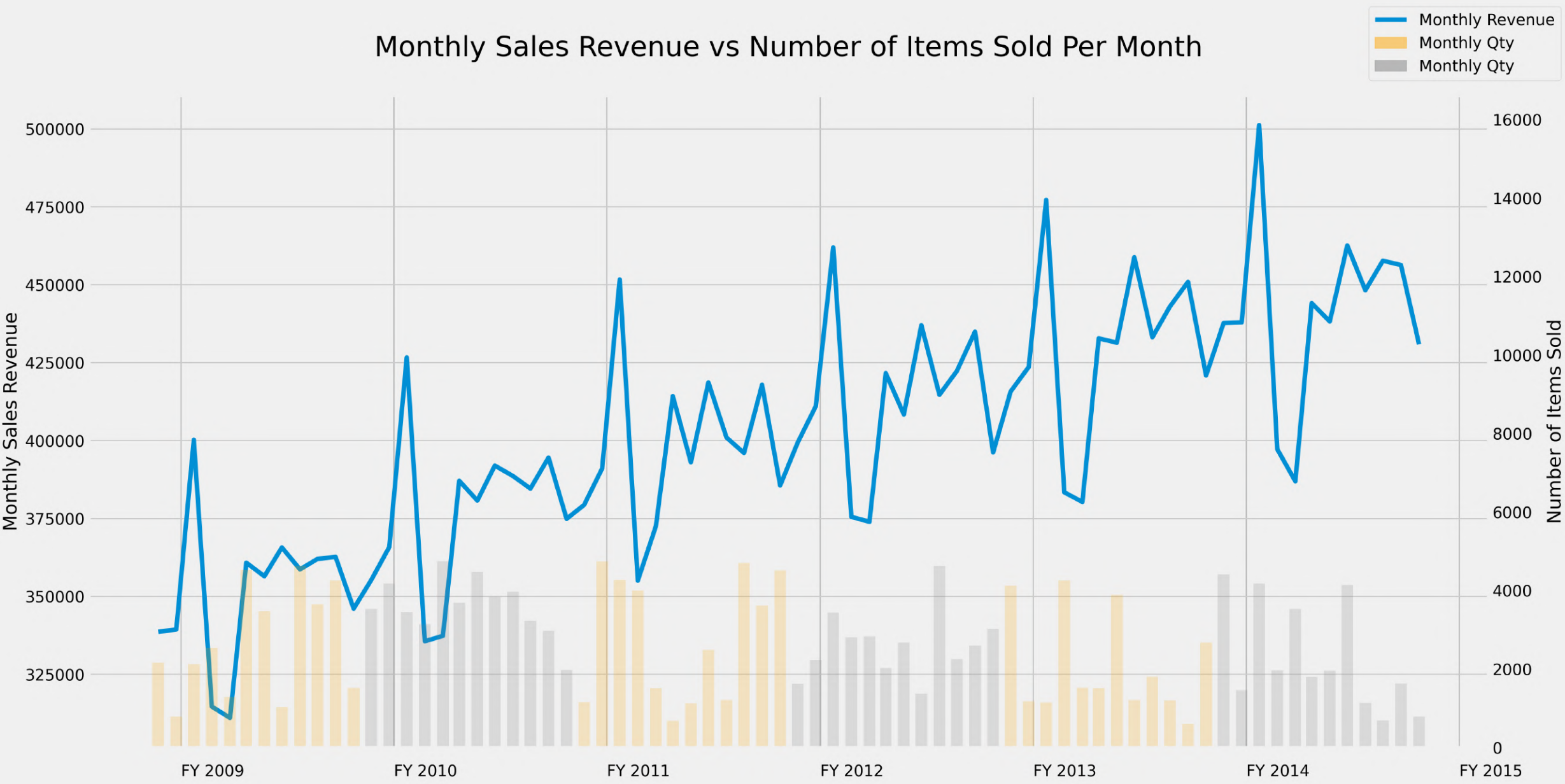


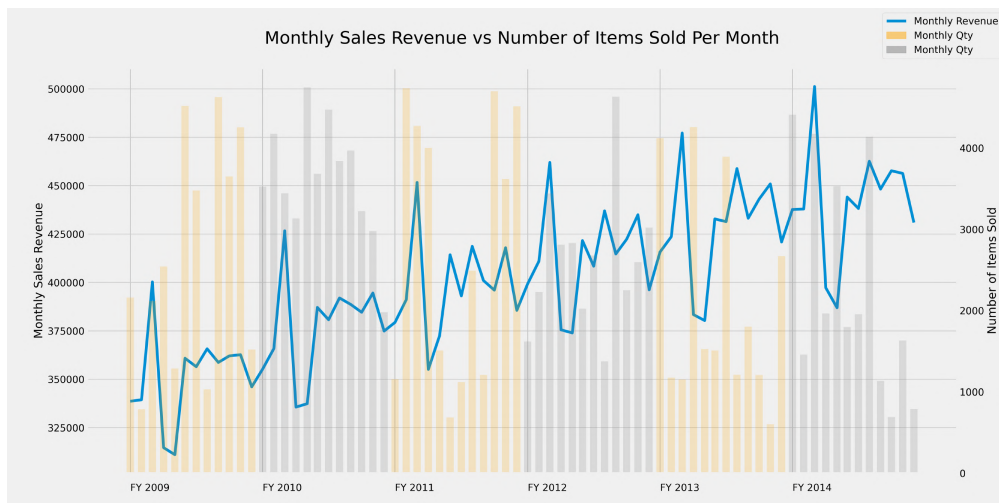
COMBO CHARTS (OVERLAY CHARTS)

Monthly Sales Revenue vs Number of Items Sold Per Month



Monthly Sales Revenue vs Number of Items Sold Per Month





The plot and data is inspired from the below link :

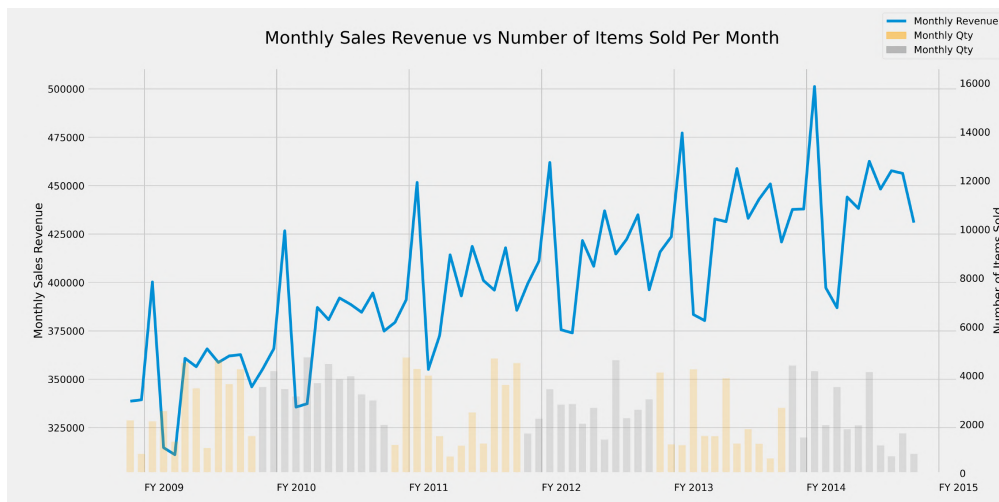
<https://pythondata.com/visualizing-data-overlying-charts/>

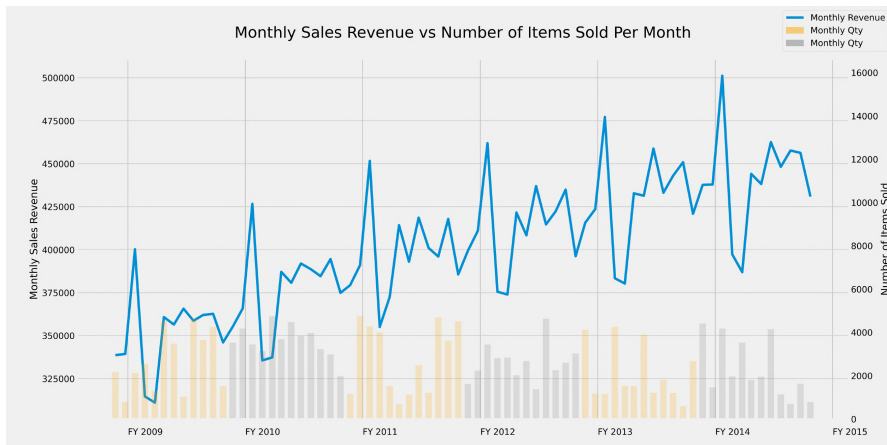
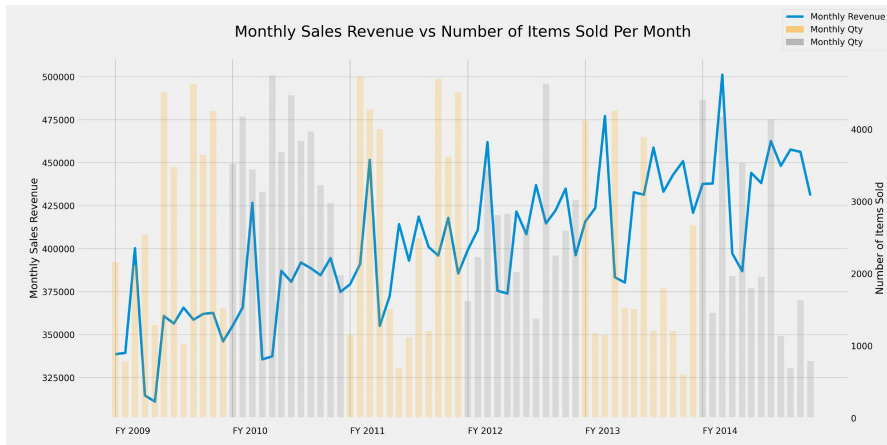
Most of the code and comments of original link is as it is considering it is very informative. However, below changes have been done :

- Locator/formatter pairs created for placement of x-axis tick labels
- For loop to create the bar plots iteratively
- Cyler object with islice from itertools to assign colors to bars
- Addition of legend by mix of automatic detection and proxy artists

For the revised code, go to the link :

https://github.com/Bhaskar-JR/Matplotlib_Overlaying_Charts/blob/main/Matplotlib_Overlaying_Charts.ipynb





The plot and data is inspired from the [link](#).

Below changes have been done in the code :

- Locator/formatter pairs created for placement of x-axis tick labels
- For loop to create the bar plots iteratively
- Cycler object with islice from itertools to assign colors to bars
- Addition of legend by mix of automatic detection and proxy artists

For the revised code, go to the [link](#)

```
plt.rcParams['figure.figsize']=(20,10) # set the figure size
plt.style.use('fivethirtyeight') # using the fivethirtyeight matplotlib theme
```

```
path1 = "https://raw.githubusercontent.com/Bhaskar-JR/Matplotlib_Overlaying_Charts/main/Sales_Data.csv"
#path = '/Users/bhaskarroy/Files/Data Science/PYTHON/Visualisation/Matplotlib/Overlaying charts/sales.csv'
```

```
sales = pd.read_csv(path1) # Read the data in
sales.Date = pd.to_datetime(sales.Date) #set the date column to datetime
sales.set_index('Date', inplace=True) #set the index to the date column
```

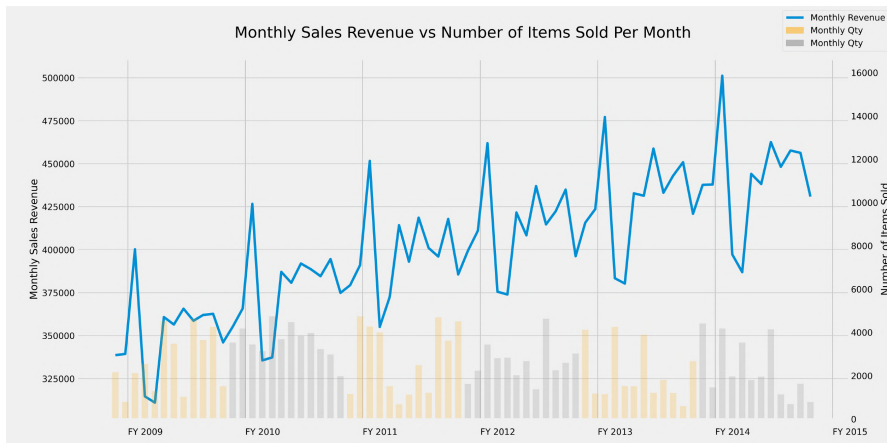
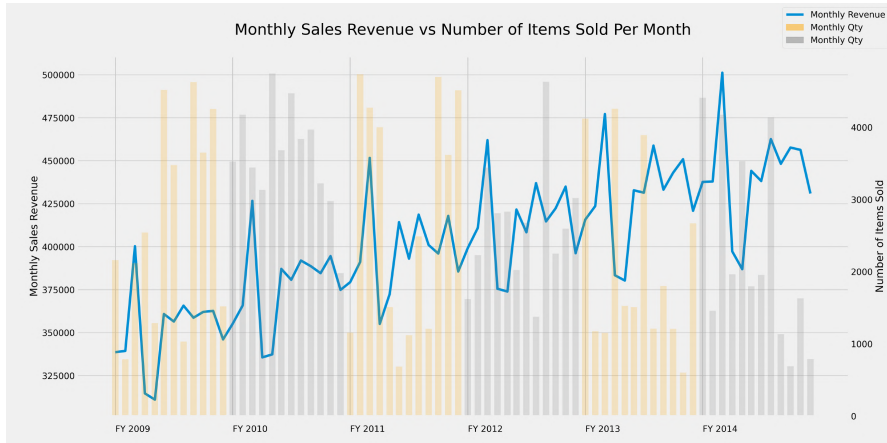
now the hack for the multi-colored bar chart:

create fiscal year dataframes covering the timeframes you are looking for. In this case,
the fiscal year covered October - September.

Note: This should be set up as a function, but for this small amount of data,
I just manually built each fiscal year. This is not very pythonic and would
suck to do if you have many years of data, but it isn't bad for a few years of data.

```
fy10_all = sales[(sales.index >= '2009-10-01') & (sales.index < '2010-10-01')]
fy11_all = sales[(sales.index >= '2010-10-01') & (sales.index < '2011-10-01')]
fy12_all = sales[(sales.index >= '2011-10-01') & (sales.index < '2012-10-01')]
fy13_all = sales[(sales.index >= '2012-10-01') & (sales.index < '2013-10-01')]
fy14_all = sales[(sales.index >= '2013-10-01') & (sales.index < '2014-10-01')]
fy15_all = sales[(sales.index >= '2014-10-01') & (sales.index < '2015-10-01')]
```

Code continues to Next slide



The plot and data is inspired from the [link](#).

Below changes have been done in the code :

- Locator/formatter pairs created for placement of x-axis tick labels
- For loop to create the bar plots iteratively
- Cycler object with islice from itertools to assign colors to bars
- Addition of legend by mix of automatic detection and proxy artists

For the revised code, go to the [link](#)

Let's build our plot

```
fig, ax1 = plt.subplots()
ax2 = ax1.twinx() # set up the 2nd axis
```

```
df = sales.copy()
df.index = mdates.date2num(df.index)
ax1.plot(df.Sales_Dollars, label = 'Monthly Revenue') #plot the Revenue on axis #1
```

```
# Creating cycler object with islice to set bar plot colors to grey, orange alternately
from itertools import cycle, islice
from datetime import datetime
import matplotlib.patches as mpatches
color_list = islice(cycle(['orange', 'grey']), 0, None)
```

```
# Using for loop to plot the fiscal year data sequentially as bar plots
```

```
# Assign color from cycler object
```

```
kwargs = dict(width=20, alpha=0.2) #kwargs dictionary for bar plot arguments
```

```
for fy in [fy10_all, fy11_all, fy12_all, fy13_all, fy14_all, fy15_all]:
```

```
    fyr = fy.copy()
```

```
    fyr.index = mdates.date2num(fyr.index)
```

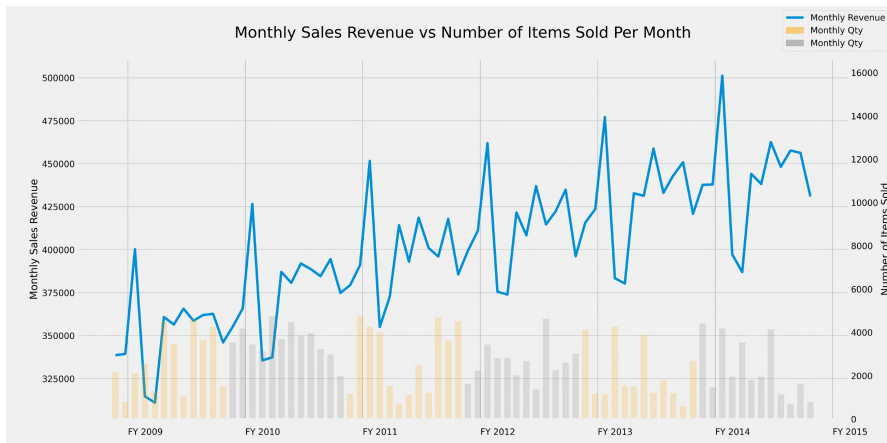
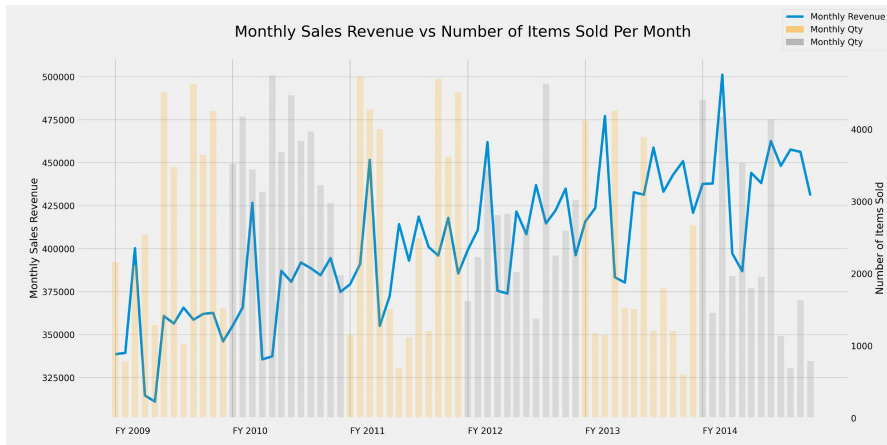
```
    ax2.bar(fyr.index, fyr.Quantity, **kwargs, color = next(color_list), label = 'Monthly Quantity')
```

```
ax2.grid(b=False) # turn off grid #2
```

```
ax1.set_title('Monthly Sales Revenue vs Number of Items Sold Per Month', fontsize = 25, y = 1.05)
```

```
ax1.set_ylabel('Monthly Sales Revenue')
```

```
ax2.set_ylabel('Number of Items Sold')
```



The plot and data is inspired from the [link](#).

Below changes have been done in the code :

- Locator/formatter pairs created for placement of x-axis tick labels
- For loop to create the bar plots iteratively
- Cyler object with islice from itertools to assign colors to bars
- Addition of legend by mix of automatic detection and proxy artists

For the revised code, go to the [link](#)

#Creating locator/formatter

```
from dateutil.rule import rule, MONTHLY, YEARLY
```

```
from datetime import datetime, timedelta #timedelta holds days, seconds, microseconds internally
```

```
import matplotlib.dates as mdates
```

```
import matplotlib.ticker as ticker
```

```
start_date = datetime(2009,10,1)
```

#Let's do ceiling division to get the relevant number of financial years

```
count = -(-(sales.index[-1]-start_date)//timedelta(days = 365.25)) #Division of timedelta by timedelta of 365.25days
```

```
date_list = list(rrule(freq=YEARLY, count=count, dtstart=start_date))
```

```
date_list = mdates.date2num(date_list)
```

#Creating locator and formatter

```
locator = ticker.FixedLocator(date_list)
```

```
date_labels = [datetime.strftime(x, "FY %Y") for x in mdates.num2date(date_list)]
```

```
formatter = ticker.FixedFormatter(date_labels)
```

Set the x-axis labels to be more meaningful than just some random dates.

```
ax1.xaxis.set_major_locator(locator)
```

```
ax1.xaxis.set_major_formatter(formatter)
```

```
ax1.xaxis.grid(b = True, color = 'grey', lw = 1,alpha = 0.5)
```

```
ax1.xaxis.set_tick_params(pad = 15)
```

```
[tkl.set(ha = 'left',rotation = 0, rotation_mode = "anchor") for tkl in ax1.xaxis.get_ticklabels()]
```

Creating Legend with a mix of automatic detection and adding proxy artists

```
handle, label = ax1.get_legend_handles_labels()
```

```
patch1 = mpatches.Patch(color='orange',alpha = 0.5, label='Monthly Qty',zorder = 100)
```

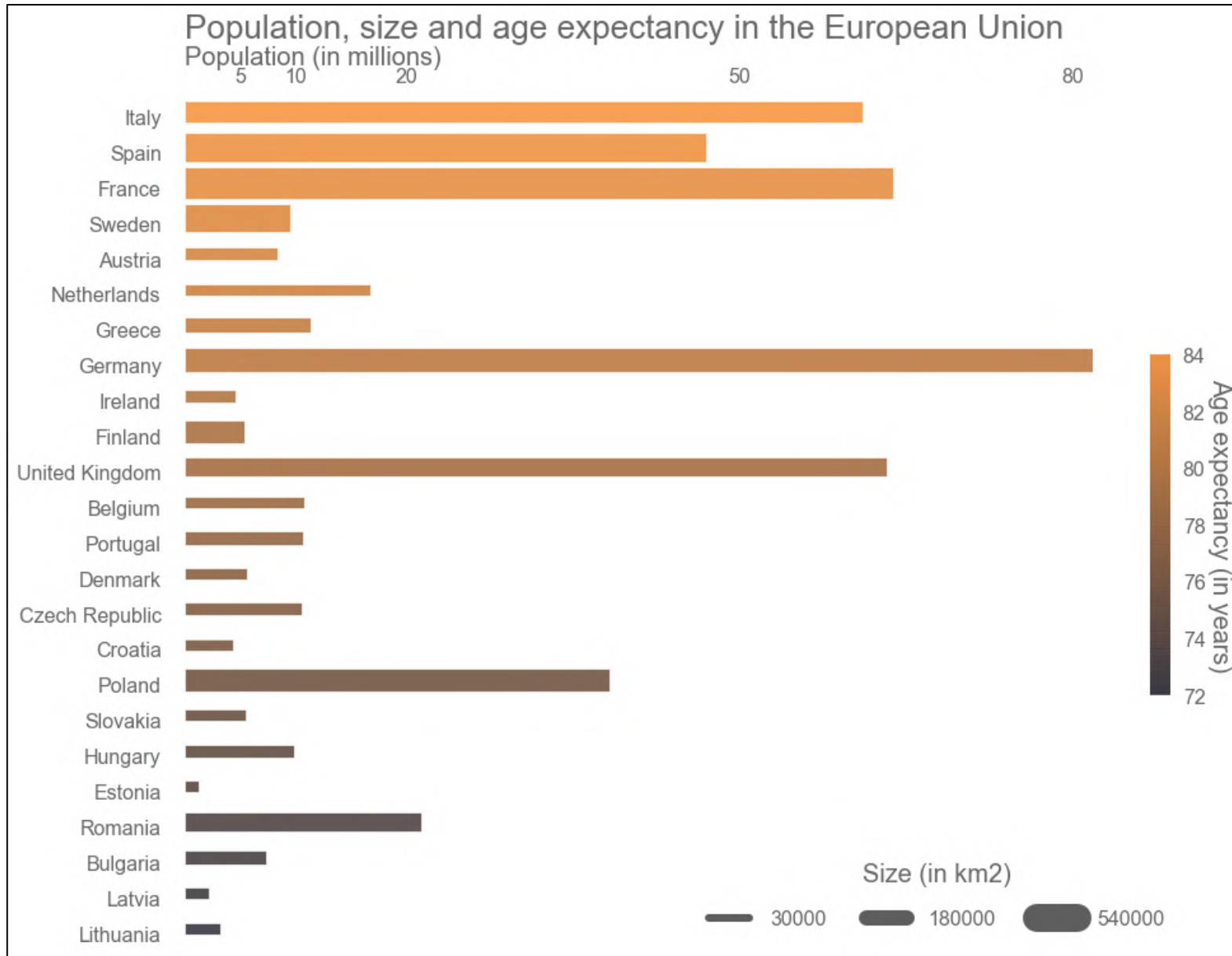
```
patch2 = mpatches.Patch(color='grey',alpha = 0.5, label='Monthly Qty', zorder = 100)
```

```
handle.extend([patch1, patch2])
```

```
ax1.legend(handles = handle,bbox_to_anchor = (1,1), loc = 'upper right', bbox_transform = fig.transFigure)
```

```
#ax2.set_ylim(0,ax2.get_ylim()[1]*10//3) # To restrict the bar plots to only 30% of the height of line plots
```

```
plt.show()
```



<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for detailed run-through of the context and the code for creating the plot.

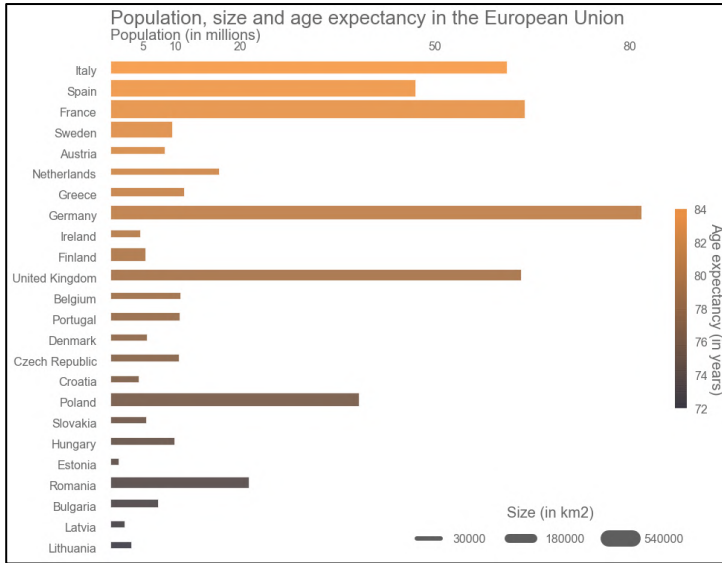
Data Source : demographic data from countries in the European Union obtained from [Wolfram|Alpha](https://www.wolframalpha.com/).

Data Set :

Contains information on population, extension and life expectancy in 24 European countries.

Visualization :

- Our Main objective is exploration on usage of hues, transparencies and simple layouts to create informative, rich and appealing visualisation.
- Specifically
 - bar length (along horizontal direction) is population indicator
 - bar length (along vertical direction) is country size indicator
 - Color of the bar is mapped to Age expectancy



<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for comprehensive run-through of the context and the code for creating the plot.

The code has been reproduced from the link with all the comments intact. There is only a minor change owing to API changes as the link is dated. The change is : [mpl.colors.Normalize](#) has been used for instantiation of norm as required for Scalar Mappable.

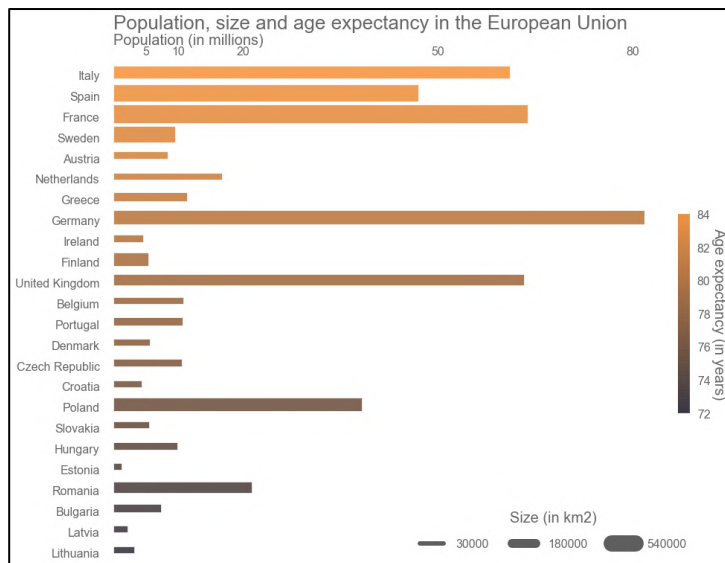
See also [Colormap reference](#) for a list of builtin colormaps. [Creating Colormaps in Matplotlib](#) for examples of how to make colormaps. [Choosing Colormaps in Matplotlib](#) an in-depth discussion of choosing colormaps. [Colormap Normalization](#) for more details about data normalization. [Customized Colorbars tutorials](#)

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib as mpl
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.lines import Line2D
from matplotlib import cm

# Preparing the Data
countries = ['France', 'Spain', 'Sweden', 'Germany', 'Finland', 'Poland', 'Italy',
            'United Kingdom', 'Romania', 'Greece', 'Bulgaria', 'Hungary',
            'Portugal', 'Austria', 'Czech Republic', 'Ireland', 'Lithuania', 'Latvia',
            'Croatia', 'Slovakia', 'Estonia', 'Denmark', 'Netherlands', 'Belgium']
extensions = [547030, 504782, 450295, 357022, 338145, 312685, 301340, 243610, 238391,
            131940, 110879, 93028, 92090, 83871, 78867, 70273, 65300, 64589, 56594,
            49035, 45228, 43094, 41543, 30528]
populations = [63.8, 47.9, 55.8, 1.8, 5.42, 38.3, 61.1, 63.2, 21.3, 11.4, 7.35,
            9.93, 10.7, 8.44, 10.6, 4.63, 3.28, 2.23, 4.38, 5.49, 1.34, 5.61,
            16.8, 10.8]
life_expectancies = [81.8, 82.1, 81.8, 80.7, 80.5, 76.4, 82.4, 80.5, 73.8, 80.8, 73.5,
            74.6, 79.9, 81.1, 77.7, 80.7, 72.1, 72.2, 77.7, 75.4, 74.4, 79.4, 81, 80.5]
data = {'extension' : pd.Series(extensions, index=countries),
        'population' : pd.Series(populations, index=countries),
        'life expectancy' : pd.Series(life_expectancies, index=countries)}

df = pd.DataFrame(data)
df = df.sort_values(by = 'life expectancy')
```

Code continues to Next slide



<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for comprehensive run-through of the context and the code for creating the plot.

The code has been reproduced from the link with all the comments intact. There is only a minor change owing to API changes as the link is dated. The change is : [mpl.colors.Normalize](#) has been used for instantiation of norm as required for Scalar Mappable.

See also [Colormap reference](#) for a list of builtin colormaps. [Creating Colormaps in Matplotlib](#) for examples of how to make colormaps. [Choosing Colormaps in Matplotlib](#) an in-depth discussion of choosing colormaps. [Colormap Normalization](#) for more details about data normalization. [Customized Colorbars tutorials](#)

```
# Create a figure of given size
fig = plt.figure(figsize=(16,12))
# Add a subplot
ax = fig.add_subplot(111)
# Set title
ttl = 'Population, size and age expectancy in the European Union'
```

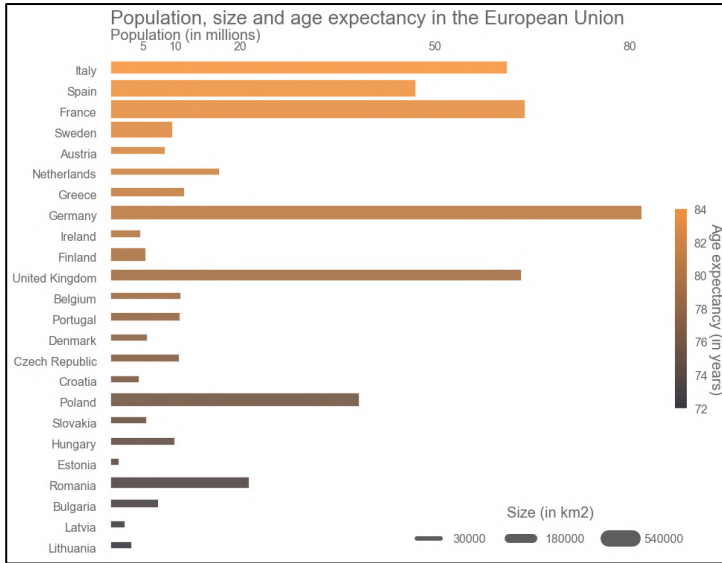
```
# Set color transparency (0: transparent; 1: solid)
a = 0.7
# Create a colormap
customcmap = [(x/24.0, x/48.0, 0.05) for x in range(len(df))]
```

```
# Plot the 'population' column as horizontal bar plot
df['population'].plot(kind='barh', ax=ax, alpha=a, legend=False, color=customcmap,
                      edgecolor='w', xlim=(0,max(df['population'])), title=ttl)
```

```
# Remove grid lines (dotted lines inside plot)
ax.grid(False)
# Remove plot frame
ax.set_frame_on(False)
# Pandas trick: remove weird dotted line on axis
#ax.lines[0].set_visible(False)
```

```
# Customize title, set position, allow space on top of plot for title
ax.set_title(ax.get_title(), fontsize=26, alpha=a, ha='left')
plt.subplots_adjust(top=0.9)
ax.title.set_position((0,1.08))
```

```
# Set x axis label on top of plot, set label text
ax.xaxis.set_label_position('top')
xlab = 'Population (in millions)'
ax.set_xlabel(xlab, fontsize=20, alpha=a, ha='left')
ax.xaxis.set_label_coords(0, 1.04)
```



<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for comprehensive run-through of the context and the code for creating the plot.

The code has been reproduced from the link with all the comments intact. There is only a minor change owing to API changes as the link is dated. The change is : [mpl.colors.Normalize](#) has been used for instantiation of norm as required for Scalar Mappable.

See also
[Colormap reference](#) for a list of builtin colormaps.
[Creating Colormaps in Matplotlib](#) for examples of how to make colormaps.
[Choosing Colormaps in Matplotlib](#) an in-depth discussion of choosing colormaps.
[Colormap Normalization](#) for more details about data normalization.
[Customized Colorbars tutorials](#)

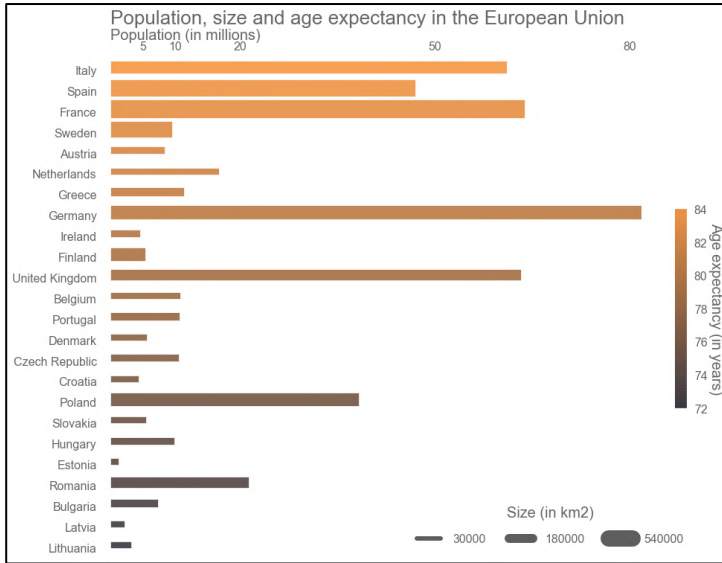
```
# Position x tick labels on top
ax.xaxis.tick_top()
# Remove tick lines in x and y axes
ax.yaxis.set_ticks_position('none')
ax.xaxis.set_ticks_position('none')
```

```
# Customize x tick labels
xticks = [5,10,20,50,80]
ax.xaxis.set_ticks(xticks)
ax.set_xticklabels(xticks, fontsize=16, alpha=a)
```

```
# Customize y tick labels
yticks = [item.get_text() for item in ax.get_yticklabels()]
ax.set_yticklabels(yticks, fontsize=16, alpha=a)
ax.yaxis.set_tick_params(pad=12)
```

```
# Set bar height dependent on country extension
# Set min and max bar thickness (from 0 to 1)
hmin, hmax = 0.3, 0.9
xmin, xmax = min(df['extension']), max(df['extension'])
# Function that interpolates linearly between hmin and hmax
f = lambda x: hmin + (hmax-hmin)*(x-xmin)/(xmax-xmin)
# Make array of heights
hs = [f(x) for x in df['extension']]
```

```
# Iterate over bars
for container in ax.containers:
    # Each bar has a Rectangle element as child
    for i,child in enumerate(container.get_children()):
        # Reset the lower left point of each bar so that bar is centered
        child.set_y(child.get_y()- 0.125 + 0.5-hs[i]/2)
        # Attribute height to each Rectangle according to country's size
        plt.setp(child, height=hs[i])
```

<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for comprehensive run-through of the context and the code for creating the plot.

The code has been reproduced from the link with all the comments intact. There is only a minor change owing to API changes as the link is dated. The change is : [mpl.colors.Normalize](#) has been used for instantiation of norm as required for Scalar Mappable.

See also
[Colormap reference](#) for a list of builtin colormaps.
[Creating Colormaps in Matplotlib](#) for examples of how to make colormaps.
[Choosing Colormaps in Matplotlib](#) an in-depth discussion of choosing colormaps.
[Colormap Normalization](#) for more details about data normalization.
[Customized Colorbars tutorials](#)

Legend

Create fake labels for legend

l1 = Line2D([], [], linewidth=6, color='k', alpha=a)

l2 = Line2D([], [], linewidth=12, color='k', alpha=a)

l3 = Line2D([], [], linewidth=22, color='k', alpha=a)

Set three legend labels to be min, mean and max of countries extensions

(rounded up to 10k km2)

rnd = 10000

rnd = 10000

labels = [str(int(round(l/rnd,1)*rnd)) for l in [min(df['extension']), np.mean(df['extension']), max(df['extension'])]]

Position legend in lower right part

Set ncol=3 for horizontally expanding legend

leg = ax.legend([l1, l2, l3], labels, ncol=3, frameon=False, fontsize=16,
 bbox_to_anchor=[1.1, 0.11], handlelength=2,
 handletextpad=1, columnspacing=2, title='Size (in km2)')

Customize legend title

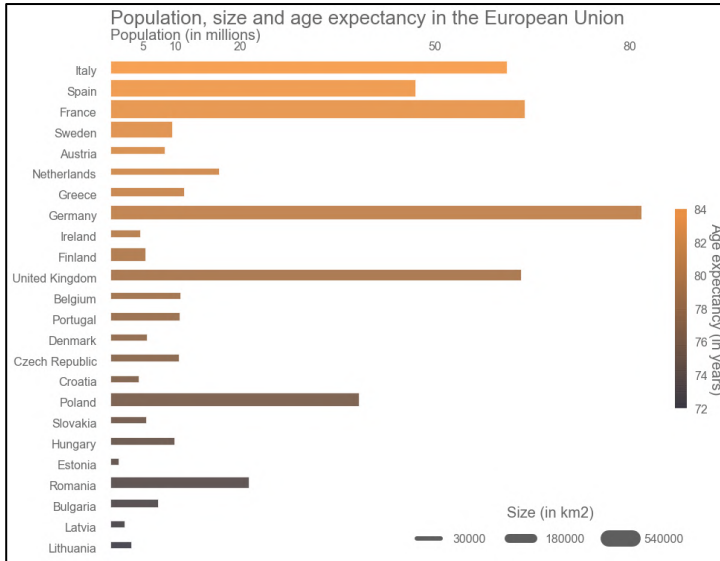
Set position to increase space between legend and labels

plt.setp(leg.get_title(), fontsize=20, alpha=a)

leg.get_title().set_position((0, 10))

Customize transparency for legend labels

[plt.setp(label, alpha=a) for label in leg.get_texts()]



<https://datasciencelab.wordpress.com/2013/12/21/beautiful-plots-with-pandas-and-matplotlib/>

Go to the above link for comprehensive run-through of the context and the code for creating the plot.

The code has been reproduced from the link with all the comments intact. There is only a minor change owing to API changes as the link is dated. The change is : [mpl.colors.Normalize](#) has been used for instantiation of norm as required for Scalar Mappable.

See also
[Colormap reference](#) for a list of builtin colormaps.
[Creating Colormaps in Matplotlib](#) for examples of how to make colormaps.
[Choosing Colormaps in Matplotlib](#) an in-depth discussion of choosing colormaps.
[Colormap Normalization](#) for more details about data normalization.
[Customized Colorbars tutorials](#)

```
# Create a fake colorbar
ctb = LinearSegmentedColormap.from_list('custombar', customcmap, N=2048)
# Trick from http://stackoverflow.com/questions/8342549/
# matplotlib-add-colorbar-to-a-sequence-of-line-plots
# Used mpl.colors.Normalize for Scalar Mappable
sm = plt.cm.ScalarMappable(cmap=ctb, norm=mpl.colors.Normalize(vmin=72, vmax=84))
# Fake up the array of the scalar mappable
sm._A = []

# Set colorbar, aspect ratio
cbar = plt.colorbar(sm, alpha=0.05, aspect=16, shrink=0.4)
cbar.solids.set_edgecolor("face")

# Remove colorbar container frame
cbar.outline.set_visible(False)

# Fontsize for colorbar ticklabels
cbar.ax.tick_params(labelsize=16)

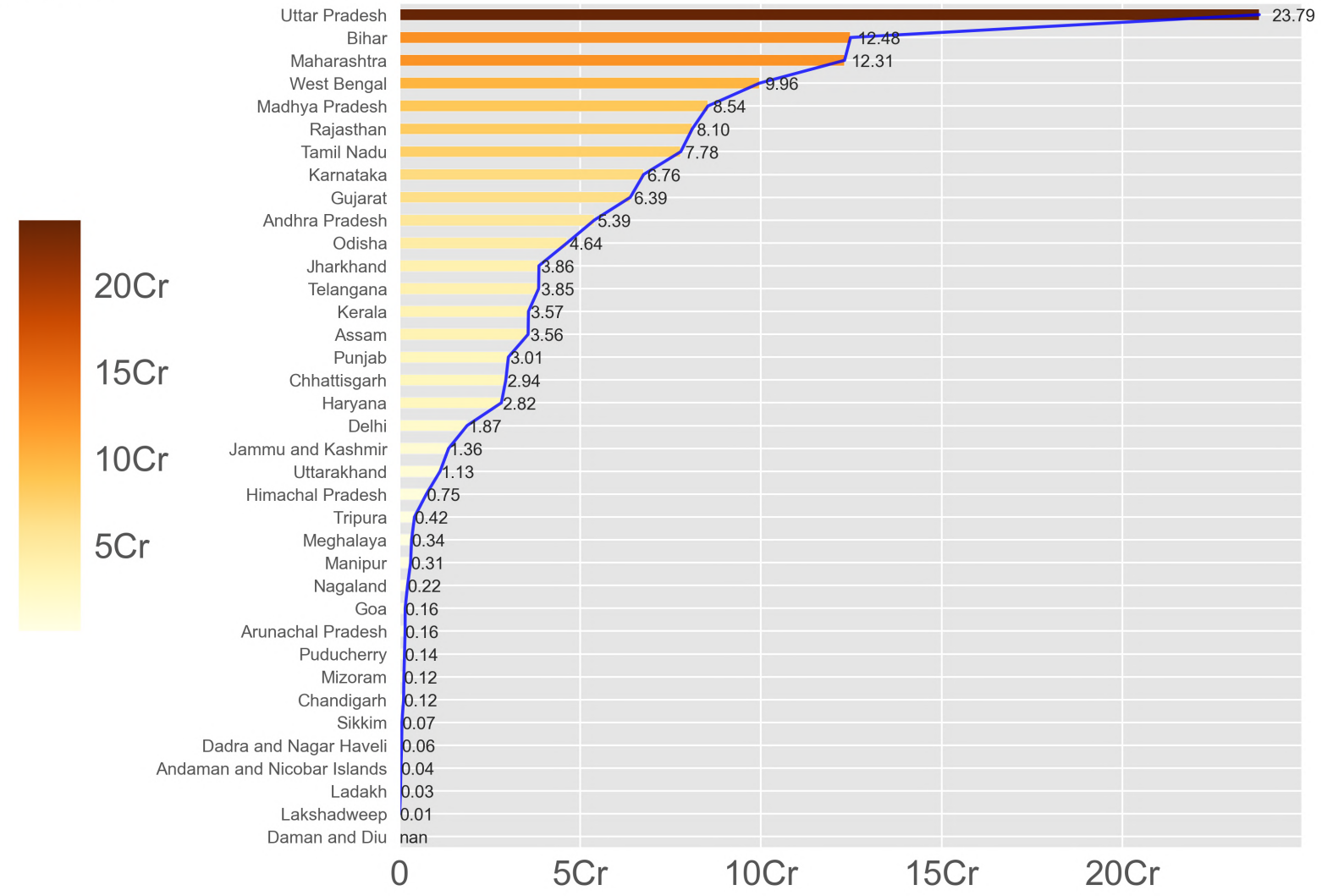
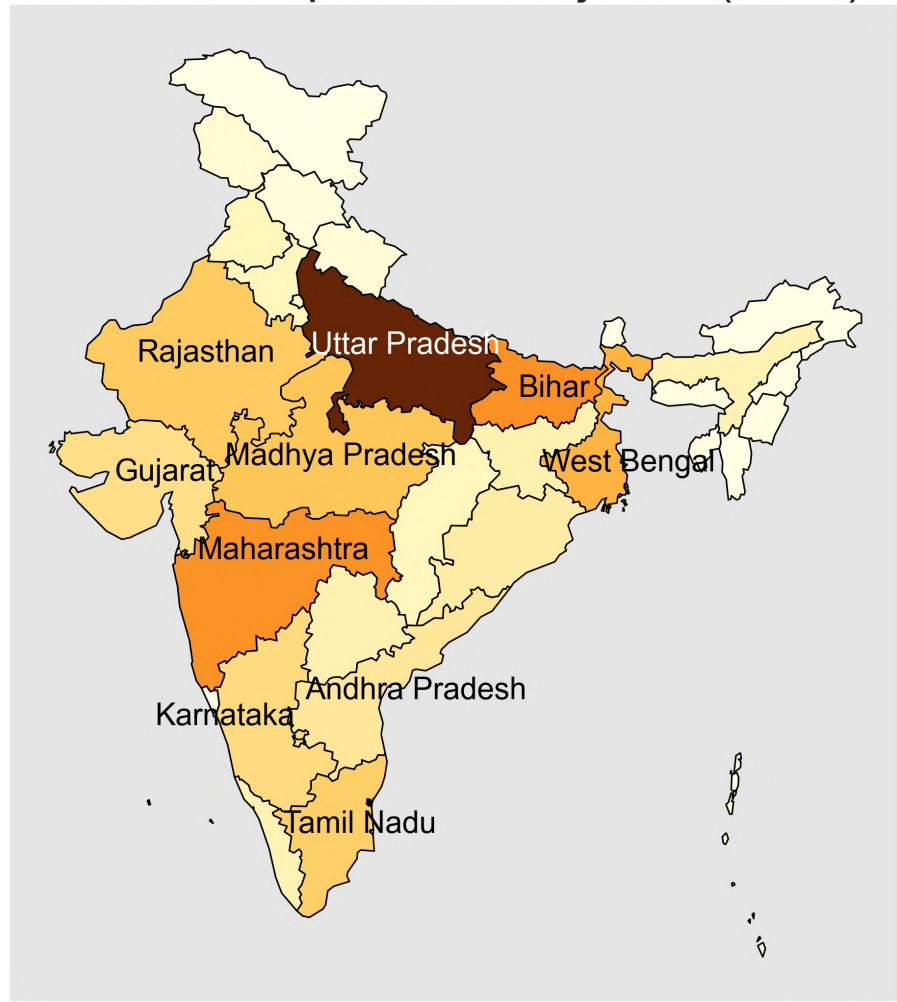
# Customize colorbar tick labels
mytk = range(72, 86, 2)
cbar.set_ticks(mytk)
cbar.ax.set_yticklabels([str(a) for a in mytk], alpha=a)

# Colorbar label, customize fontsize and distance to colorbar
cbar.set_label('Age expectancy (in years)', alpha=a, rotation=270, fontsize=20,
               labelpad=20)

# Remove color bar tick lines, while keeping the tick labels
cbarytk = plt.getp(cbar.ax.axes, 'yticklines')
plt.setp(cbarytk, visible=False)

plt.savefig("beautiful plot.png", bbox_inches='tight')
```

Statewise Population Projected(2020) in Crores

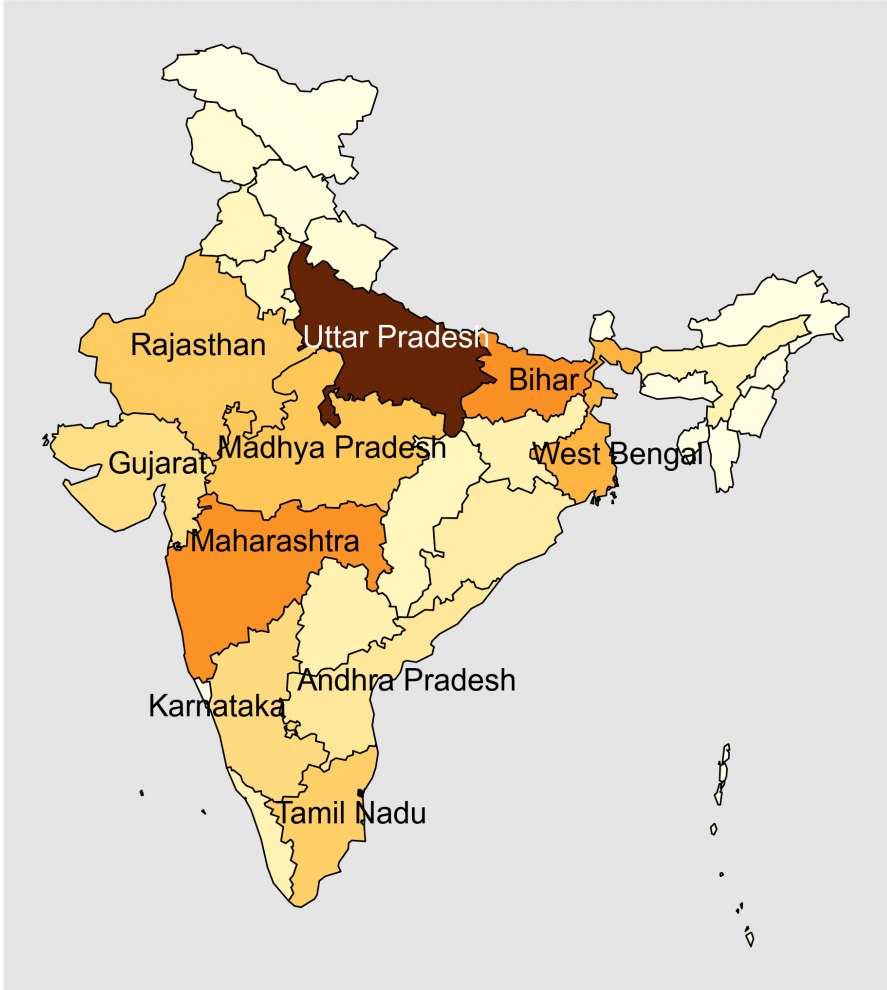


Data Source : <https://uidai.gov.in/images/state-wise-aadhaar-saturation.pdf>

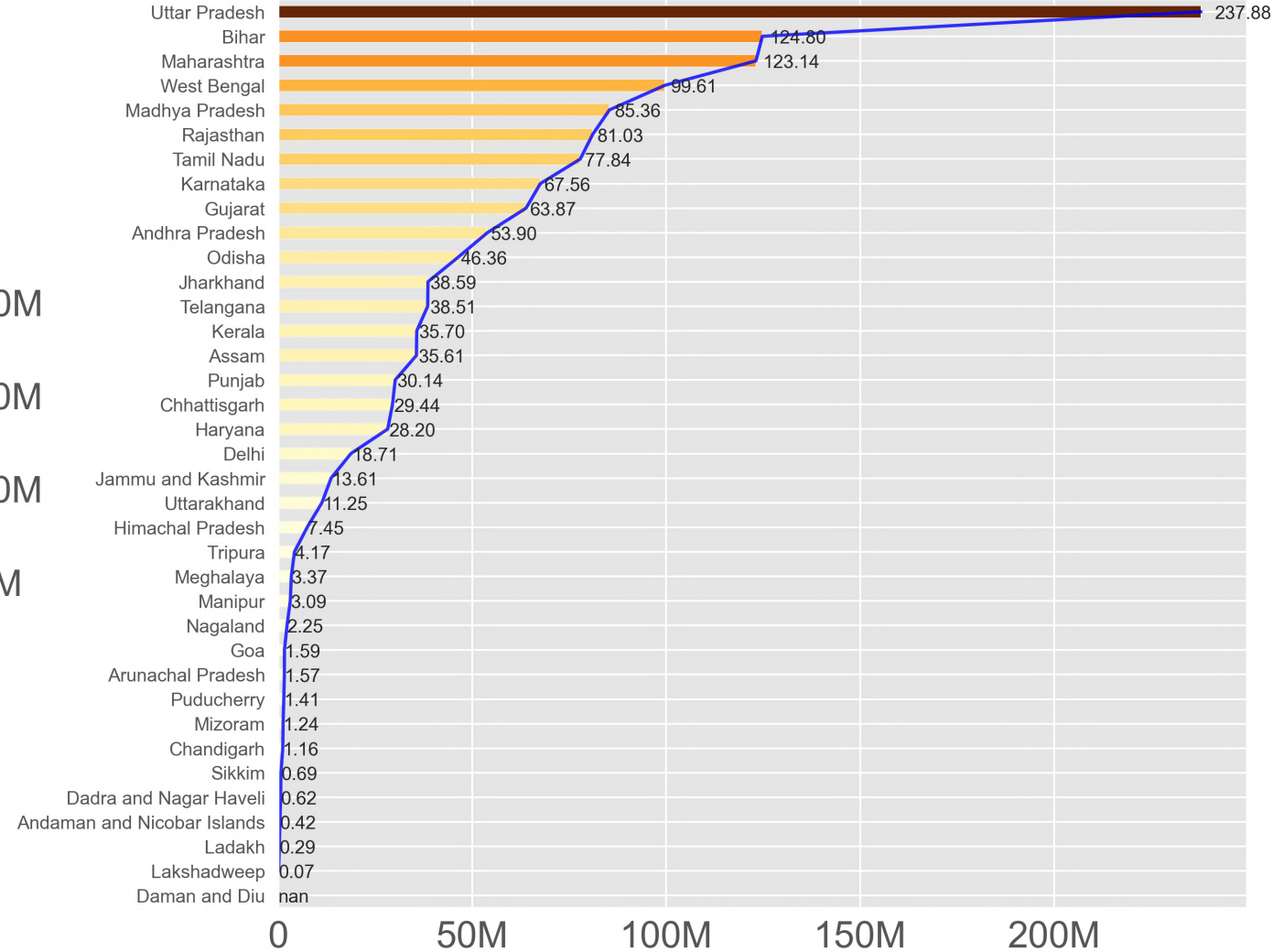
Check out the github repository for the shape files and notebook:
<https://github.com/Bhaskar-JR/India-Statewise-Population-2020>

GeoSpatial Data - Choropleth Map
Using Geopandas, Matplotlib

Statewise Population Projected(2020) in Millions



Data Source : <https://uidai.gov.in/images/state-wise-aadhaar-saturation.pdf>



Check out the github repository for the shape files and notebook:
<https://github.com/Bhaskar-JR/India-Statewise-Population-2020>

GeoSpatial Data - Choropleth Map
Using Geopandas, Matplotlib

REFERENCES

Category	Description	Link
Matplotlib overview, tutorials	Succinct overview of matplotlib	https://www.webpages.uidaho.edu/~stevel/504/Matplotlib%20Notes.pdf
	What does it mean when they say 'stateful'?	https://stackoverflow.com/questions/24764918/what-does-it-mean-when-they-say-stateful/24768434#24768434
	Summary of Matplotlib for python developers	https://radhakrishna.typepad.com/rks_musings/2012/04/matplotlib-for-python-developers-summary.html
	Very Interesting Article "Matplotlib and the Future of Visualization in Python" by Jake VanderPlas	https://jakevdp.github.io/blog/2013/03/23/matplotlib-and-the-future-of-visualization-in-python/
	Excellent resource with examples using mix of Pyplot and OO style - "Python Data Cleaning Cookbook- Modern techniques and Python tools to detect and remove dirty data and extract key insights" by Michael Walker	1
	Advance plotting Tutorial	https://python4astronomers.github.io/plotting/advanced.html
	Nicola P Rougier	https://github.com/rougier/matplotlib-tutorial
	Beautiful tutorial using Pandas and Matplotlib in conjunction :	http://jonathansoma.com/lede/algorithms-2017/classes/fuzziness-matplotlib/how-pandas-uses-matplotlib-plus-figures-axes-and-subplots/
Python related	Blog Post - Python, Machine Learning, and Language Wars	https://sebastianraschka.com/blog/2015/why-python.html
	Understanding df.plot in Pandas :	http://jonathansoma.com/lede/algorithms-2017/classes/fuzziness-matplotlib/understand-df-plot-in-pandas/
	Discussion to wrap all of the matplotlib getter and setter methods with python properties, allowing them to be read and written like class attributes	https://matplotlib.org/stable/devel/MEP/MEP13.html?highlight=artist%20attributes#id1
	On *args and **kwargs	https://stackoverflow.com/questions/36901/what-does-double-star-asterisk-and-star-asterisk-do-for-parameters?rq=1
	On purpose of kwargs in	https://stackoverflow.com/questions/1769403/what-is-the-purpose-and-use-of-kwargs?
	Decorator basics	https://stackoverflow.com/questions/739654/how-to-make-function-decorators-and-chain-them-together?rq=1
	Sorting a dictionary by value	https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value/613218#613218
	Using ax parameter	https://stackoverflow.com/questions/43085427/using-dataframe-plot-to-make-a-chart-with-subplots-how-to-use-ax-parameter?
	Infinite Iterating always from a fixed position :	https://stackoverflow.com/questions/42459582/how-to-specify-where-to-start-in-an-itertools-cycle-function
	Difference between property and attribute :	https://stackoverflow.com/questions/7374748/whats-the-difference-between-a-python-property-and-attribute/7377013#7377013
	Introduction to Probability Using Python :	https://nbviewer.jupyter.org/url/norvig.com/ipython/Probability.ipynb
	Discussion on Array Programming with Numpy :	https://www.repository.cam.ac.uk/bitstream/handle/1810/315595/41586_2020_2649_MOESM1_ESM.pdf?sequence=2
	On Random Number	https://people.astro.umass.edu/~schloerb/ph281/Lectures/RandomNumbers/RandomNumbers.pdf
	Fluent Python by Luciano Ramalho. Chapter - 'Iterables, Iterators, and Generators' is relevant for many operations in matplotlib, pandas, numpy	Also check https://scipy-lectures.org/advanced/advanced_python/index.html#id2
	'The NumPy array: a structure for efficient numerical computation' by Stefan van der Walt provides indepth understanding of Numpy Arrays	https://hal.inria.fr/inria-00564007/document

Category	Description	Link
Figures, subplots, gridspec	Relationship between DPI and figure size	https://stackoverflow.com/questions/47633546/relationship-between-dpi-and-figure-size/47639545#47639545
	Change the figure size	https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib/638443#638443
	Using bbox_inches = 'tight' in plt.savefig() to ensure all artists are in saved output behaviour of jupyter "inline" backend (%matplotlib inline)	https://stackoverflow.com/questions/44642082/text-or-legend-cut-from-matplotlib-figure-on-savefig
	On Multiple Subplots	https://actrue.com/en/object-oriented-way-of-using-matplotlib-4-multiple-subplots/
	Combining different figures and putting in single subplot	https://stackoverflow.com/questions/16748577/matplotlib-combine-different-figures-and-put-them-in-a-single-subplot-sharing-a?noredirect=1&lq=1
	Brilliant Link on adding subplots -	https://stackoverflow.com/questions/3584805/in-matplotlib-what-does-the-argument-mean-in-fig-add-subplot111?rq=1
	Excellent link on using GridSpec	https://www.python-course.eu/matplotlib_gridspec.php
Fig patch, axes patch, zorder	Usecase Scenarios of plt.cla, plt.clf, plt.close :	https://stackoverflow.com/questions/8213522/when-to-use-cls-clf-or-close-for-clearing-a-plot-in-matplotlib
	Superb Example on setting fc of fig patch and axes patch Zorder demo :	https://stackoverflow.com/questions/4581504/how-to-set-opacity-of-background-colour-of-graph-with-matplotlib https://matplotlib.org/stable/gallery/misc/zorder_demo.html?highlight=zorder
Ticks, tick labels	On removing ticks	https://stackoverflow.com/questions/12998430/remove-xticks-in-a-matplotlib-plot
	Rotating Axis Labels in Matplotlib	https://www.pythoncharts.com/2019/05/17/rotating-axis-labels/
	Tick Label Rotation and spacing	https://stackoverflow.com/questions/43152502/how-can-i-rotate-xticklabels-in-matplotlib-so-that-the-spacing-between-each-xtic?noredirect=1&lq=1
	Making tick label font size smaller	https://stackoverflow.com/questions/6390393/matplotlib-make-tick-labels-font-size-smaller/11386056#11386056
	Formatter for large tick values -	https://dfrieds.com/data-visualizations/how-format-large-tick-values.html
	Discussion on tick labels zorder being above axes and patches whereas the grid lines below the axes :	https://stackoverflow.com/questions/48327099/is-it-possible-to-set-higher-z-index-of-the-axis-labels
	Tick locator and formatter for timeseries without using matplotlib.dates	https://stackoverflow.com/questions/33743394/
Time Series	AR, MA and ARIMA Models in Python	https://www.ritchievink.com/blog/2018/09/26/algorithm-breakdown-ar-ma-and-arima-models/
	Time Series Related Books	Practical Time Series Analysis - Prediction with Statistics & Machine Learning by Aileen Nielson Machine Learning for Time Series Forecasting with Python by Francesca Lazzeri Advanced Data Science and Analytics with Python by Jesus Rogel-Salazar Analysis of Financial Time Series, 3d Edition, by Ruey S. Tsay Hands-on Time Series Analysis with Python by B V Vishwas and A. Patel Data Science Revealed by Tshepo Chris Nokeri

Category	Description	Link
Legend	Go to reference for understanding the purport of Legend arguments	https://stackoverflow.com/questions/4700614/how-to-put-the-legend-out-of-the-plot/4701285#4701285
	On adding text artists in Legend	https://stackoverflow.com/questions/27174425/how-to-add-a-string-as-the-artist-in-matplotlib-legend
	Drawing and Animating Shapes with Matplotlib	https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html
	Tuple Arguments in bbox_to_anchor	https://stackoverflow.com/questions/39803385/what-does-a-4-element-tuple-argument-for-bbox-to-anchor-mean-in-matplotlib/39806180#39806180
Annotation resources - from matplotlib v3.4.2	• Adding value labels on a matplotlib bar chart	
	• How to annotate each segment of a stacked bar chart	
	• Stacked Bar Chart with Centered Labels	
	• How to plot and annotate multiple data columns in a seaborn barplot	
	• stack bar plot in matplotlib and add label to each section	
	• How to add multiple annotations to a barplot	
	• How to plot and annotate a grouped bar chart	
Font properties	Using font dictionary :	https://matplotlib.org/stable/gallery/text_labels_and_annotations/text_fontdict.html?highlight=font%20dictionary
	Fonts demo object oriented style :	https://matplotlib.org/stable/gallery/text_labels_and_annotations/fonts_demo.html
	List of font families or name :	https://stackoverflow.com/questions/18821795/
	set([f.name for f in matplotlib.font_manager.FontManager.afmList]) and set([f.name for f in matplotlib.font_manager.FontManager.ttfList])	
Data Visualizations	Ten Simple Rules for Better Figures	https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003833
	Excellent link on beautiful data viz	http://www.randalolson.com/2014/06/28/how-to-make-beautiful-data-visualizations-in-python-with-matplotlib/
	Superb data viz	https://www.dunderdata.com/blog/create-a-bar-chart-race-animation-in-python-with-matplotlib
	A Dramatic tour through Python's Visualisation landscape including ggplot2 and Altair	https://dsaber.com/2016/10/02/a-dramatic-tour-through-pythons-data-visualization-landscape-including-ggplot-and-altair/
	Superb link covering different kind of plots. Can serve as the base code which can be modified as per requirement	https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/#14.-Area-Chart
	Superb blog on wide ranging topics related to data wrangling/viz/programming with Pandas, matplotlib and python -	https://dfrieds.com/#dataviz
	Pie vs dot plots	https://pierreh.eu/pie-vs-dots-powersystem/
	Visualizing Data by William Cleveland	
	Cleveland dot plots in excel	https://mbounthavong.com/blog/2018/7/16/communicating-data-effectively-with-data-visualizations-part-9-cleveland-plots
	Styling the plots	https://dfrieds.com/data-visualizations/style-plots-python-matplotlib.html
	Extensive coverage of code for multiple aspects of plotting	https://queirozf.com/entries/matplotlib-pyplot-by-example

Category	Description	Link
Pandas Plotting, Seaborn	Legend in plotting with Pandas	https://stackoverflow.com/questions/21988196/legend-only-shows-one-label-when-plotting-with-pandas
	Seaborn :	https://michaelwaskom.medium.com/three-common-seaborn-difficulties-10fdd0cc2a8b
	Superb Article on using Matplotlib, pandas and seaborn together :	https://towardsdatascience.com/matplotlib-seaborn-pandas-an-ideal-amalgamation-for-statistical-data-visualisation-f619c8e8baa3
	Time Series in Seaborn :	https://www.geeksforgeeks.org/creating-a-time-series-plot-with-seaborn-and-pandas/
	On limitation of plotting time series data in Seaborn :	https://stackoverflow.com/questions/22795348/plotting-time-series-data-with-seaborn
	Plotting time in python:	https://stackoverflow.com/questions/1574088/plotting-time-in-python-with-matplotlib?rq=1
	Time Series in Matplotlib :	https://www.earthdatascience.org/courses/use-data-open-source-python/use-time-series-data-in-python/date-time-types-in-pandas-python/customize-dates-matplotlib-plots-python/
	Comprehensive coverage of datetime in Pandas Time series :	https://developers.arcgis.com/python/guide/part5-time-series-analysis-with-pandas/
R ggplot2	Converting between datetime, timestamp and datetime64	https://stackoverflow.com/questions/13703720/converting-between-datetime-timestamp-and-datetime64?
	An Introduction to ggplot2 by Joey Stanley	https://joeystanley.com/downloads/171012-ggplot2_handout.pdf
	"ggplot2 Elegant Graphics for Data Analysis" by Hadley Wickham - Highly recommended for ggplot2	https://ggplot2-book.org/introduction.html
	"R Graphics Essentials for Great Data Visualization" by Alboukadel KASSAMBAR	Highly recommended for creating great graphics for the right data using either the ggplot2 package and extensions or the traditional R graphics.
	Books on Data Science with R	Modern Data Science with R by Benjamin S. Baumer Daniel T. Kaplan Nicholas J. Horton Machine Learning Essentials by Alboukadel KASSAMBAR
	Application oriented Code Reference and R handbook for experienced and new users	https://epirhandbook.com/ggplot-basics.html#labels
	Bar charts in R ggplot :	http://bradleyboehmke.github.io/tutorials/barchart
	Manipulation of tabular data in R	http://barryrowlingson.github.io/hadleyverse/
	Comparison of viz tools in Python and R :	https://shankarmsy.github.io/posts/exploringviz1.html
	Covering baseline R vocabulary and knowledge for the primary data wrangling processes.	"Data Wrangling with R" by Bradley C. Boehmke
	"R for Data Science" by Hadley Wickham and Garrett Grolemund – Highly recommended	https://r4ds.had.co.nz/introduction.html
	"Building a plot layer by layer" by Hadley Wickham -	https://rpubs.com/hadley/ggplot2-layers
geopandas	Shapely documentation – Page No. 27 -	https://geopandas.org/getting_started/introduction.html
	Creating another layer to take care of Nan values	https://buildmedia.readthedocs.org/media/pdf/shapely/latest/shapely.pdf
	Labeling polygons in geopandas plotting	https://stackoverflow.com/questions/38473257/how-can-i-set-a-special-colour-for-nans-in-my-plot
	Placement of colorbars using cax	https://stackoverflow.com/questions/38899190/geopandas-label-polygons
	Fixing colorbars	https://matplotlib.org/stable/gallery/subplots_axes_and_figures/colorbar_placement.html
	Setting single common colorbar for two subplots	https://joseph-long.com/writing/colorbars/
	Delhi Map Geopandas discussion	https://stackoverflow.com/questions/13784201/matplotlib-2-subplots-1-colorbar
	Interesting Case study Airbnb Paris analysis	https://stackoverflow.com/questions/63644131/how-to-use-geopandas-to-plot-latitude-and-longitude-on-a-more-detailed-map-with/63646936#63646936

