Module 3

Data Analytics with Python - Applied analytics

Section: Data Visualization in Python using Matplotlib & Seaborn

Data Visualization

Data Visualization is used to communicate information clearly and efficiently to users by the usage of information graphics such as tables and charts. It helps users in analyzing a large amount of data in a simpler way. It makes complex data more accessible, understandable, and usable.

Python Libraries for data visualization

Matplotlib

Matplotlib is a data visualization library and 2-D plotting library of Python It was initially released in 2003 and it is the most popular and widely-used plotting library in the Python community.

It comes with an interactive environment across multiple platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, etc.

Seaborn

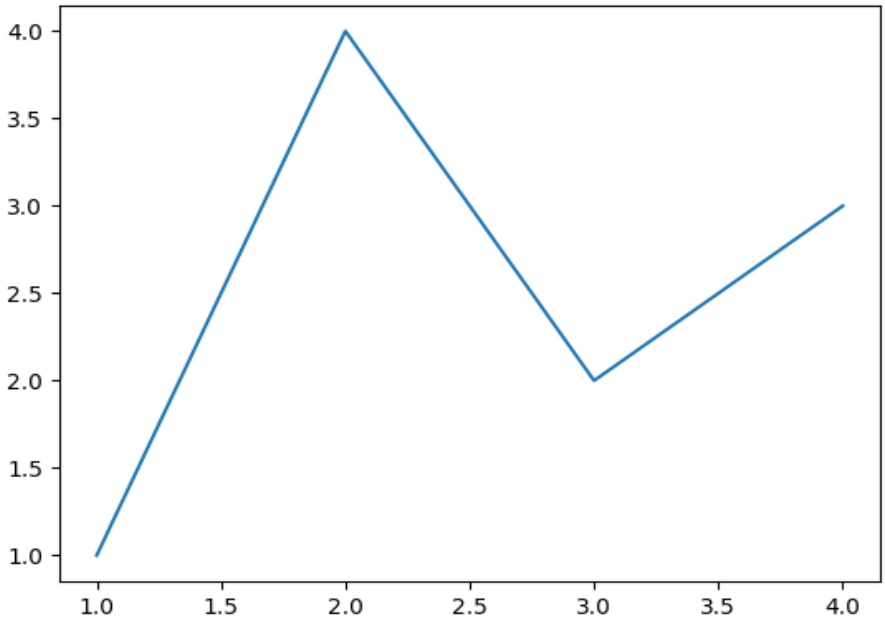
Seaborn harnesses the power of matplotlib to create beautiful charts in a few lines of code. The key difference is Seaborn's default styles and color palettes, which are designed to be more aesthetically pleasing and modern. Since Seaborn is built on top of matplotlib, you'll need to know matplotlib to tweak Seaborn's defaults.

Line Plots

Matplotlib plots your data on **Figure**s (e.g., windows, Jupyter widgets, etc.), each of which can contain one or more **Axes**, an area where points can be specified in terms of x-y coordinates (or theta-r in a polar plot, x-y-z in a 3D plot, etc). The simplest way of creating a Figure with an Axes is using **pyplot.subplots**.

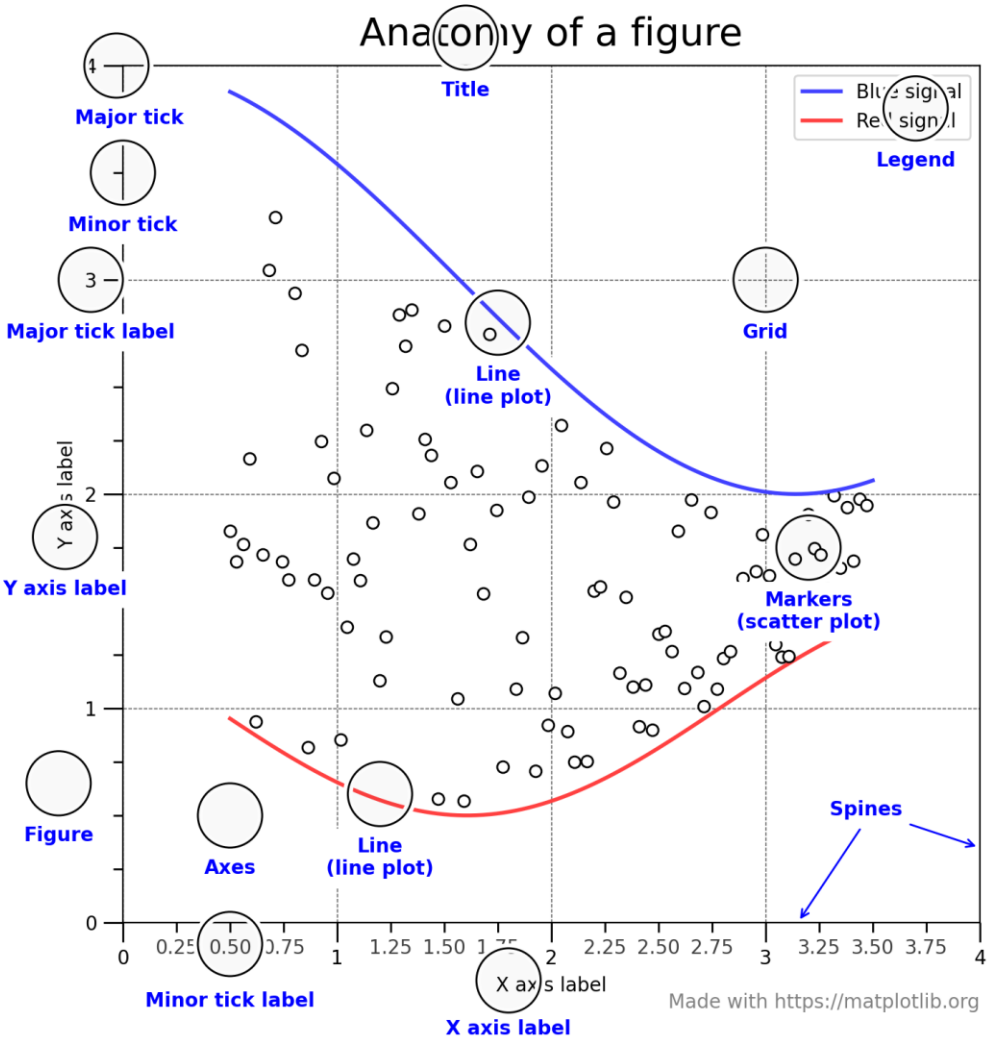
We can then use **Axes.plot** to draw some data on the Axes:

```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3]);  # Plot some data on the axes.
```



Parts of a Figure

Here are the components of a Matplotlib Figure.



Figure

The **whole** figure. The Figure keeps track of all the child **Axes**, a group of 'special' Artists (titles, figure legends, colorbars, etc), and even nested subfigures.

Axes

An Axes is an Artist attached to a Figure that contains a region for plotting data, and usually includes two (or three in the case of 3D) **Axis** objects (be aware of the difference between **Axes** and **Axis**) that provide ticks and tick labels to provide scales for the data in the Axes. Each **Axes** also has a title (set via **set title()**), an x-label (set via **set_xlabel()**), and a y-label set via **set_ylabel()**).

The **Axes** class and its member functions are the primary entry point to working with the OOP interface, and have most of the plotting methods defined on them (e.g. ax.plot(), shown above, uses the **plot** method)

Axis

These objects set the scale and limits and generate ticks (the marks on the Axis) and ticklabels (strings labelling the ticks). The location of the ticks is determined by a **Locator** object and the ticklabel strings are formatted by a **Formatter**. The combination of the correct **Locator** and **Formatter** gives very fine control over the tick locations and labels.

Artist

Basically, everything visible on the Figure is an Artist (even **Figure**, **Axes**, and **Axis** objects). This includes **Text** objects, **Line2D** objects, **collections** objects, **Patch** objects, etc. 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.

Controlling Line Patterns and Colors

Matplotlib has an additional parameter to control the colour and style of the plot.

```
plt.plot(xa, ya 'g')
```

This will make the line green. You can use any colour of red, green, blue, cyan, magenta, yellow, white or black just by using the first character of the colour name in lower case (use "k" for black, as "b" means blue).

You can also alter the linestyle, for example two dashes -- makes a dashed line. This can be used added to the colour selector, like this:

```
plt.plot(xa, ya 'r--')
```

You can use "-" for a solid line (the default), "-." for dash-dot lines, or ":" for a dotted line.

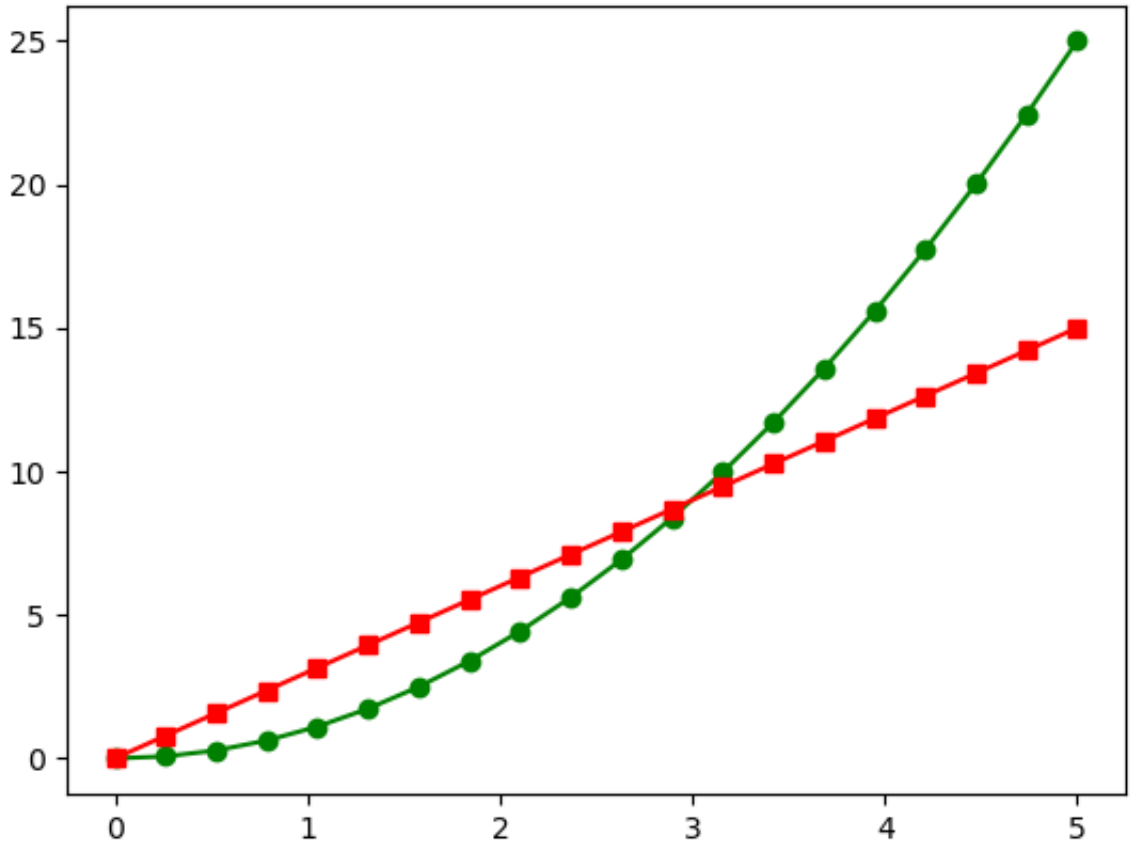
Colour parameter takes a string. The options are

- Single character colors, as above, e.g. "r", "g" etc.
- CSS colour names. There are 140 shades, with names such as seagreen, steelblue, tomato, and so on. They are listed here, but for Matplotlib the names should be written in lower case.
- RGB colours. These take the form "#rrggbb" which indicates the color as a six-digit hex value indicating the amount of red, green and blue each as a 2 digit hex value (00 to FF), again in the same way that HTML colours are specified in hex.

Marker is a symbol such as a symbol such as a small dot, square, diamond etc that indicates a data point on the graph. As with lines, markers can be controlled by a simple text string, or by a set of parameters that give more options.

Controlling markers with a string

In this example, we have added markers to our two curves:



The green curve has circle markers, the red curve has square markers.

The markers appear at the data points we have defined for the plot. Since we have used no line space to define 20 equally spaced points between x=0 and x=5, we see 20 markers at these positions.

Here is the code:

```
1. from matplotlib import pyplot as plt
2. import numpy as np
3.
4. xa = np.linspace(0, 5, 20)
5. ya = xa**2
6. plt.plot(xa, ya, 'g-o')
7. ya = 3*xa
8. plt.plot(xa, ya, 'r-s')
9. plt.show()
```

All we have changed from the original red and green curve at the start of the article are the style strings. "g-o" specifies a green curve, a solid line and a circle marker (the letter o). "r-s" specifies a red curve, a solid line, and a square marker (the letter s).

Notice that you must specify a line style, otherwise no line will be drawn (just the markers). You can used other line styles, for example "g-. o" for dash-dot lines.

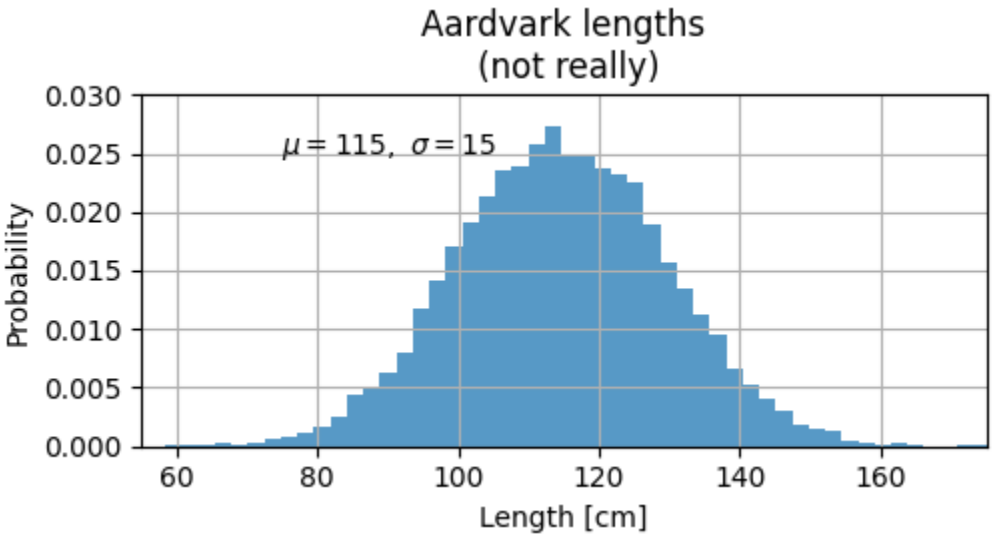
Matplotlib supports quite a few marker shapes, here are some of the common ones:

- o - circle
- s - square
- v, <, >, ^ - triangles pointing down, left, right, up
- d, D - thin or thick diamond
- x, X - cross (x is line only, X is filled shape)
- h – hexagon

Axis, labels & legends

set_xlabel, **set_ylabel**, and **set_title** are used to add text in the indicated locations (see Text in Matplotlib Plots for more discussion). Text can also be directly added to plots using **text**

```
mu, sigma = 115, 15

x = mu + sigma * np.random.randn(10000)

fig, ax = plt.subplots(figsize=(5, 2.7),
layout='constrained')

# the histogram of the data

n, bins, patches = ax.hist(x, 50, density=1, facecolor='C0',
alpha=0.75)

ax.set_xlabel('Length [cm]')

ax.set_ylabel('Probability')

ax.set_title('Aardvark lengths\n (not really)')

ax.text(75, .025, r'$\mu=115,\ \sigma=15$')

                ax.axis([55, 175, 0, 0.03])

ax.grid(True);
```



Following are some more attributes of function legend ():

- **shadow**: [None or bool] Whether to draw a shadow behind the legend. It's Default value is None.
- **font size**: The font size of the legend. If the value is numeric the size will be the absolute font size in points.
- **face color**: [None or "inherit" or color] The legend's background color.
- **edge color**: [None or "inherit" or color] The legend's background patch edge color.

Plot multiple plots in Matplotlib

We can draw multiple graphs in a single plot in two ways. One is by using subplot () function and other by superimposition of second graph on the first i.e, all graphs will appear on the same plot.

pyplot.subplots creates a figure and a grid of subplots with a single call, while providing reasonable control over how the individual plots are created.

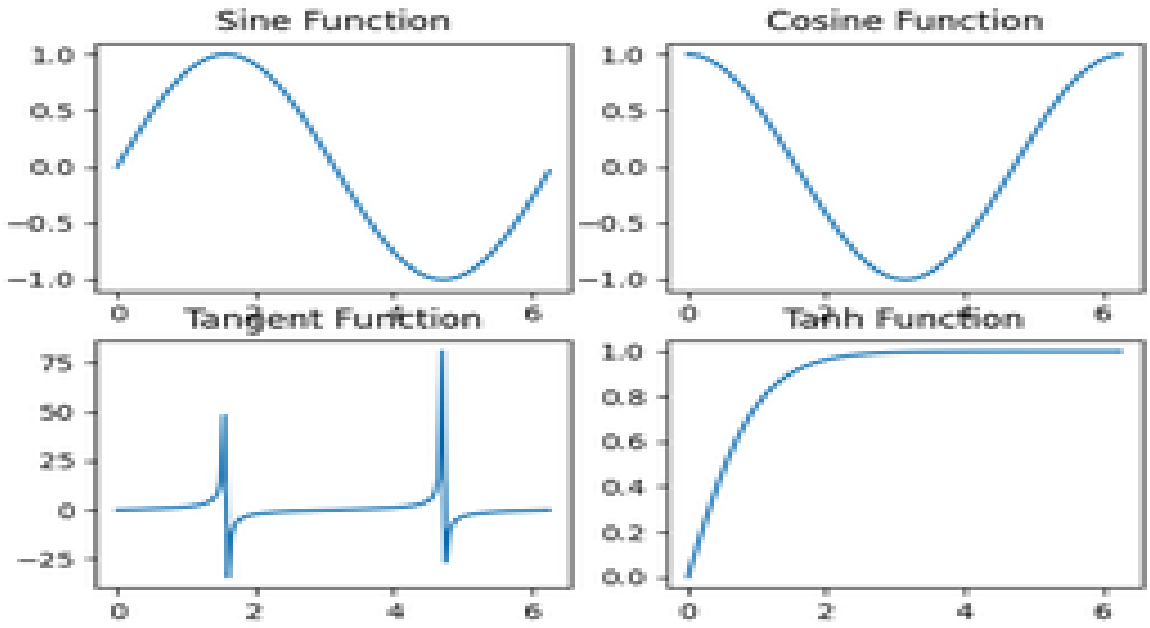
Syntax: matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)

Parameters

1. nrows, ncols: These gives the number of rows and columns respectively. Also, it must be noted that both these parameters are optional and the default value is 1.
2. sharex, sharey: These parameters specify about the properties that are shared among a and y axis.Possible values for them can be, row, col, none or default value which is False.
3. squeeze: This parameter is a boolean value specified, which asks the programmer whether to squeeze out, meaning remove the extra dimension from the array. It has a default value False.
4. subplot_kw: This parameters allow us to add keywords to each subplot and its default value is None.
5. gridspec_kw: This allows us to add grids on each subplot and has a default value of None.
6. **fig_kw: This allows us to pass any other additional keyword argument to the function call and has a default value of None.

```
import matplotlib.pyplot as plt

import numpy as np
import math

# Get the angles from 0 to 2 pie (360 degree) in narray object
X = np.arange(0, math.pi*2, 0.05)

# Using built-in trigonometric function we can directly plot
# the given cosine wave for the given angles
Y1 = np.sin(X)
Y2 = np.cos(X)
Y3 = np.tan(X)
Y4 = np.tanh(X)

# Initialise the subplot function using number of rows and columns
figure, axis = plt.subplots(2, 2)

# For Sine Function
axis[0, 0].plot(X, Y1)
axis[0, 0].set_title("Sine Function")

# For Cosine Function
axis[0, 1].plot(X, Y2)
axis[0, 1].set_title("Cosine Function")

# For Tangent Function
axis[1, 0].plot(X, Y3)
axis[1, 0].set_title("Tangent Function")

# For Tanh Function
axis[1, 1].plot(X, Y4)
axis[1, 1].set_title("Tanh Function")

# Combine all the operations and display
plt.show()
```

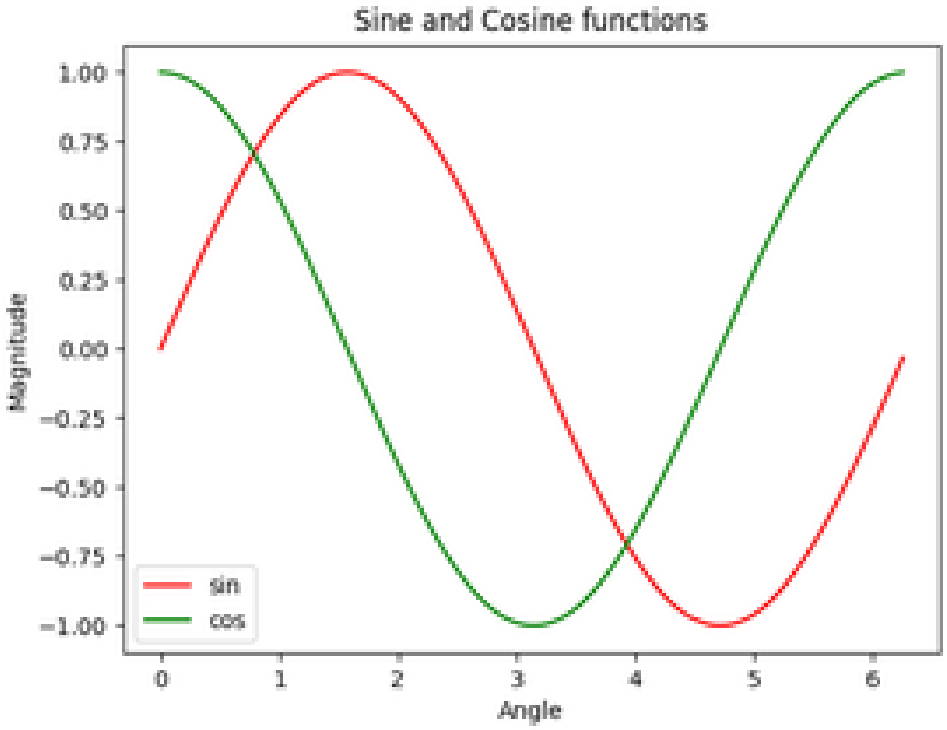
Output-


Sine Function
Cosine Function
Tangent Function
Tanh Function
1.0
0.5
0.0
−0.5
−1.0
75
50
25
0
−25
0.8
0.6
0.4
0.2
0.0
0
2
4
6

we will now have a look at plotting multiple curves by superimposing them. In this method we do not use any special function instead we directly plot the curves one above other and try to set the scale.

```
import matplotlib.pyplot as plt

import numpy as np

import math

# create an array X
X = np.arange(0, math.pi*2, 0.05)
# Assign variables to the y axis part of the curve
y = np.sin(X)
z = np.cos(X)
# Plotting both the curves
plt.plot(X, y, color='r', label='sin')
plt.plot(X, z, color='g', label='cos')

# Naming the axis and the whole graph
plt.xlabel("Angle")
plt.ylabel("Magnitude")
plt.title("Sine and Cosine functions")

# Adding legend
plt.legend()

# To load the display window
plt.show()
```

Output-


Sine and Cosine functions
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Magnitude
sin
cos
0
1
2
3
4
5
6
Angle

Types of Plots and Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Types of Plots in Seaborn Covered here are-

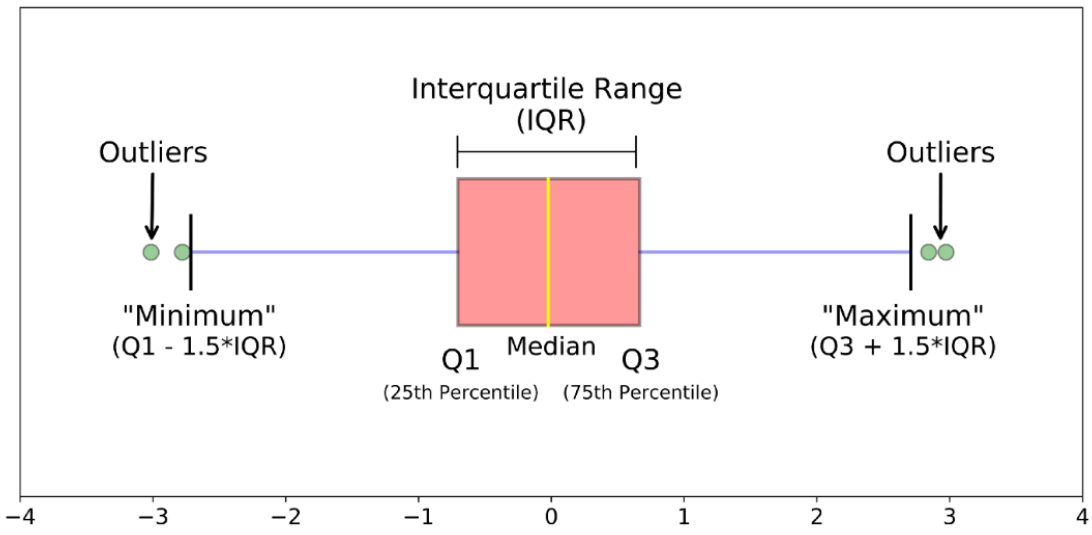
- Scatter Plot
- Line Plot
- Bar Plot
- Box Plots
- Distribution Plots
- Heatmaps
- Swarm plot
- Count plot
- Density plot

Boxplot

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

Input data can be passed in a variety of formats, including:

- Vectors of data represented as lists, NumPy arrays, or pandas Series objects passed directly to the x, y, and/or hue parameters.
- A “long-form” DataFrame, in which case the x, y, and hue variables will determine how the data are plotted.
- A “wide-form” DataFrame, such that each numeric column will be plotted.
- An array or list of vectors.

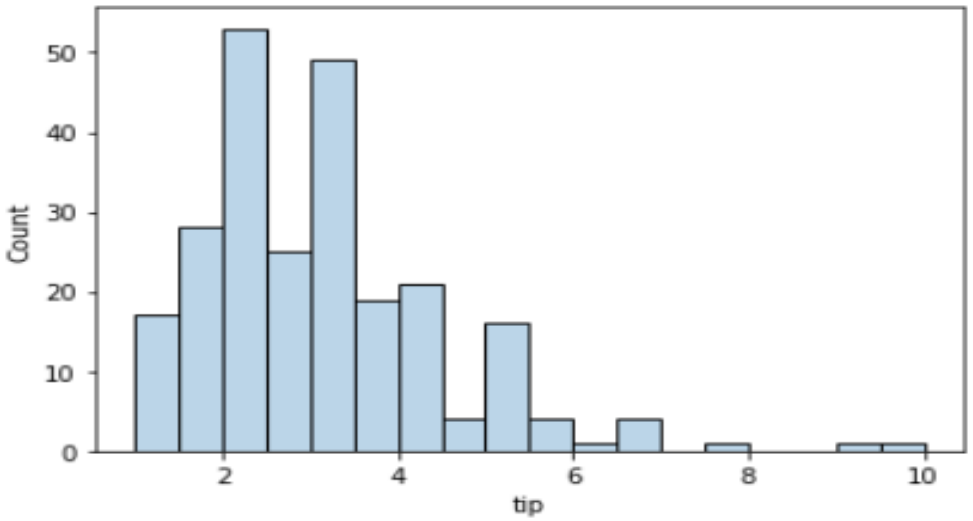


Distribution Plots

We will discuss on 3 types of distribution plots namely:

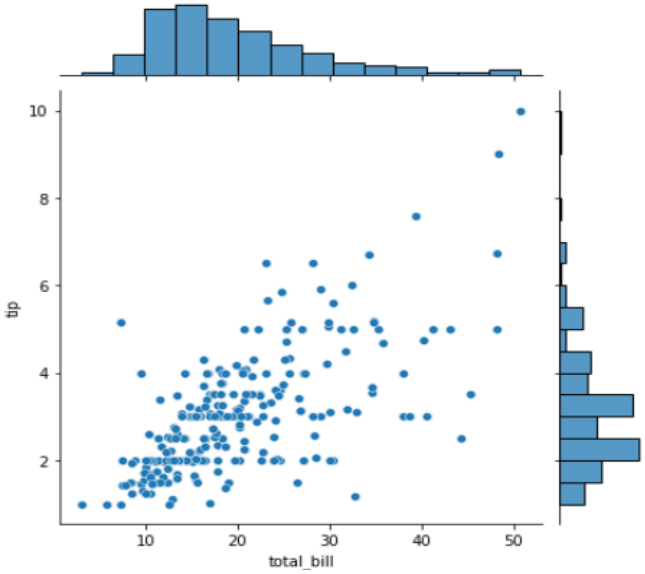
1. **Histplot**

It is used basically for univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.



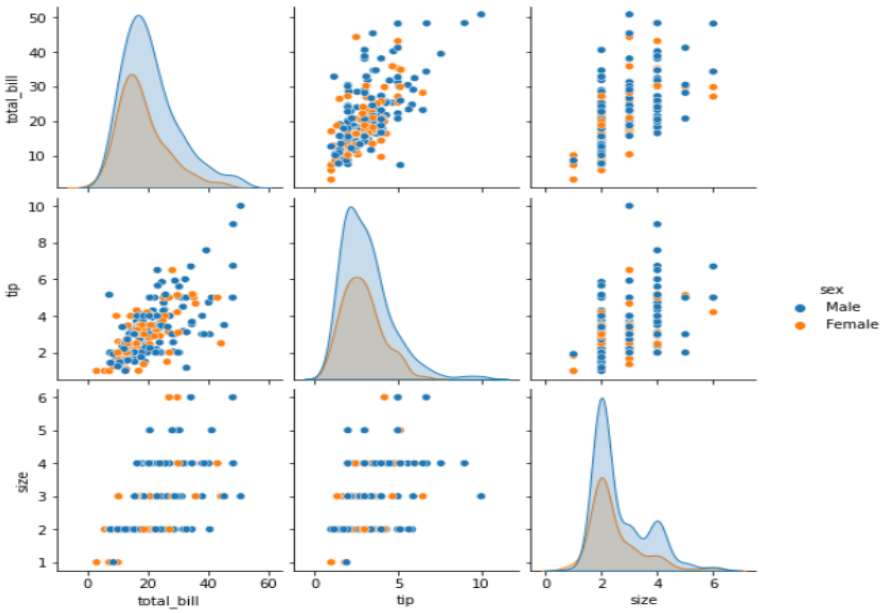
2. **Joint plot**

It is used to draw a plot of two variables with bivariate and univariate graphs. It basically combines two different plots.



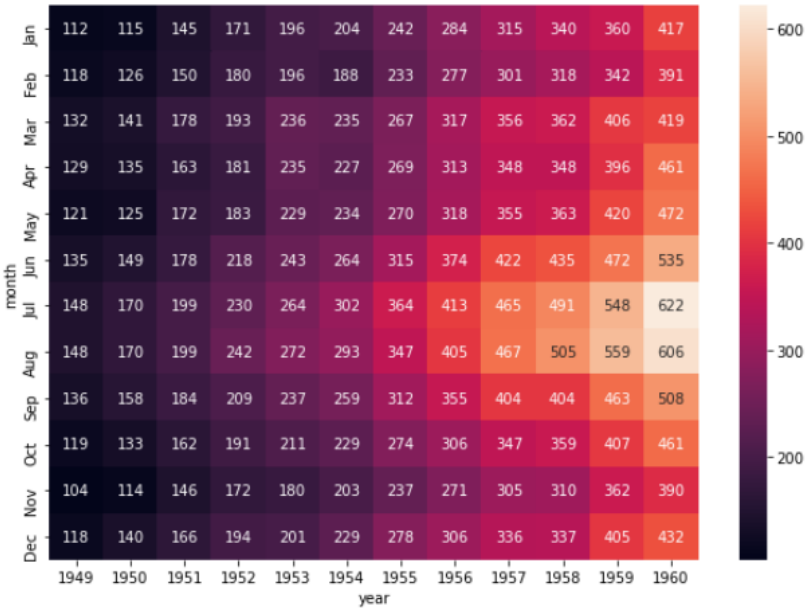
3. **Pair plot**

To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function.



Heatmaps

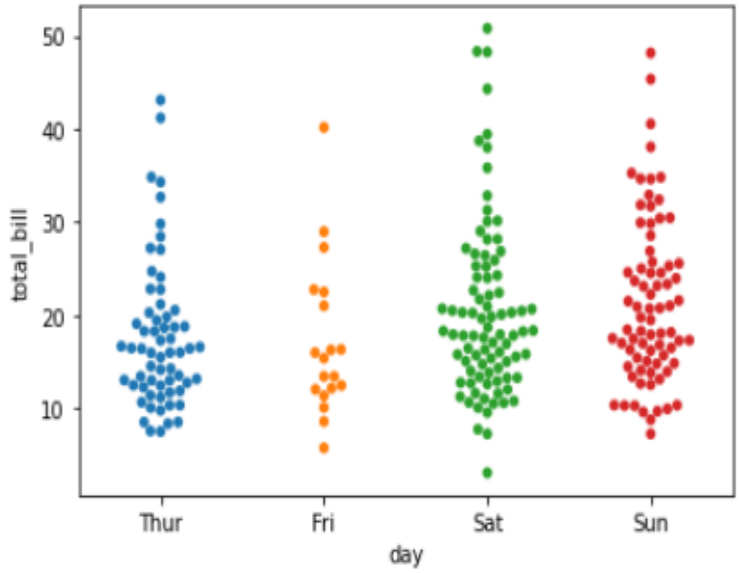
A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.



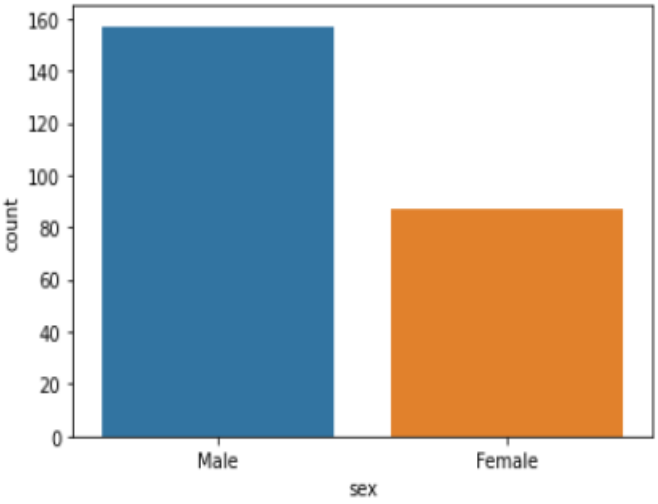
Swarmplots And Countplots

Swarmplots are used to draw a categorical scatterplot with non-overlapping points. A swarm plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

Count plots are used to show the counts of observations in each categorical bin using bars.



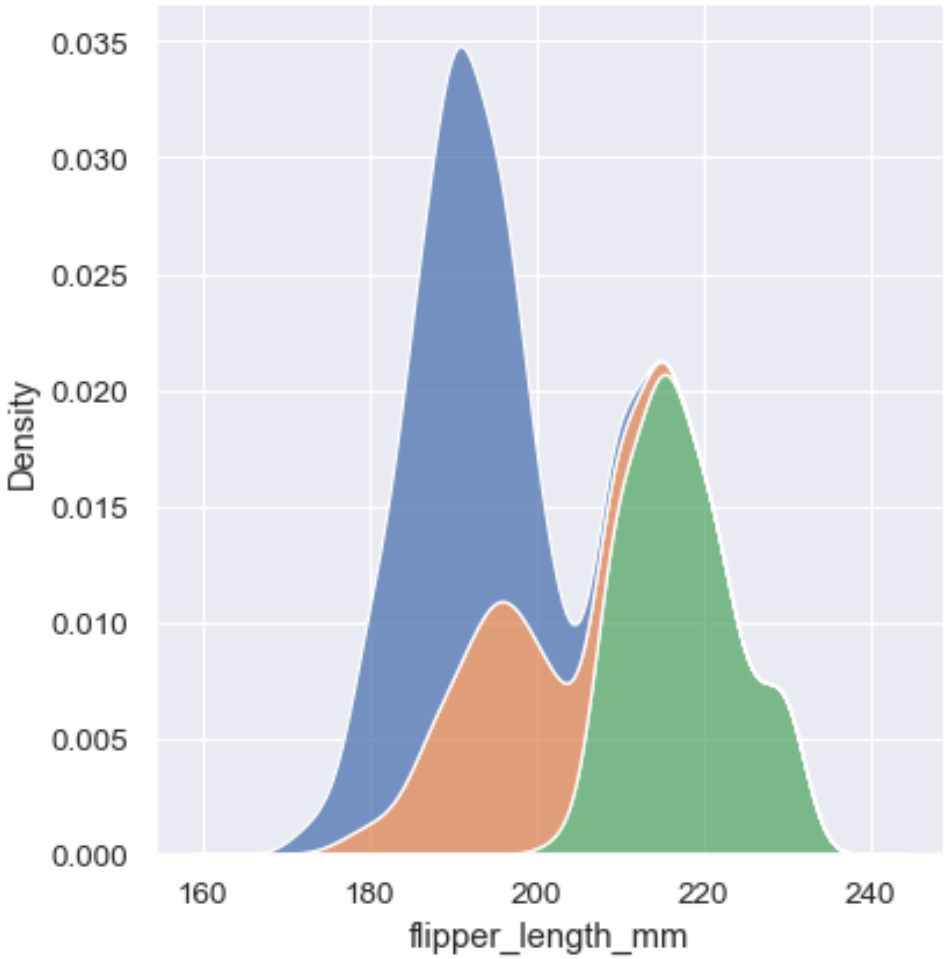
Swarmplot



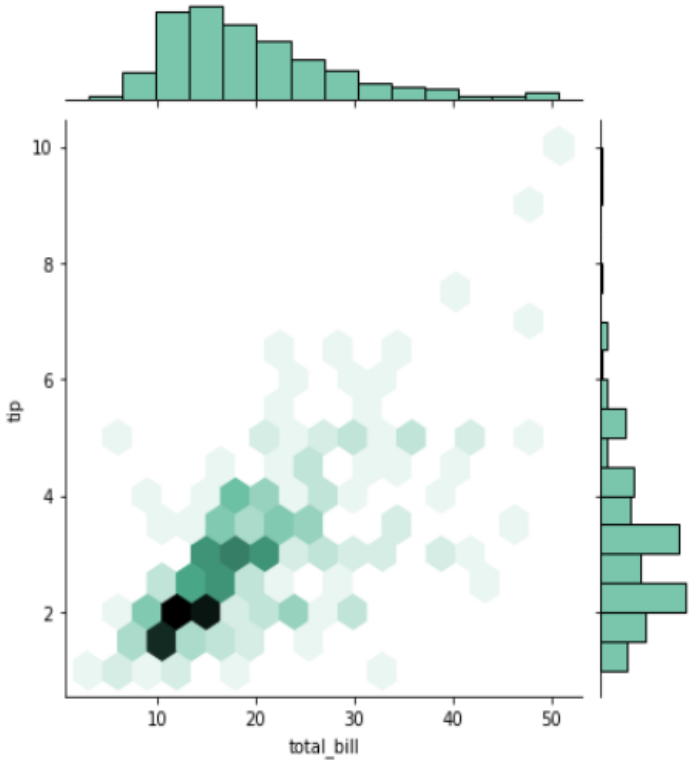
Countplot

Density Plot, Joint Distribution Plot

Density plots are used to observe the distribution of a variable in a dataset. An advantage of Density Plots over Histograms is that they're better at determining the distribution shape because they're not affected by the number of bins.



A joint density plot is useful to represent the relationship of 2 numerical variables when you have a lot of data points. Without overlapping of the points, the plotting window is split into several hexbins. The color of each hexbin denotes the number of points in it.


10
8
6
4
2
tip
10
20
30
40
50
total_bill

Summary

- Data Visualization is used to communicate information clearly and efficiently to users by the usage of information graphics such as tables and charts.
- Matplotlib and Seaborn are libraries for data visualization in python.
- The key difference is Seaborn's default styles and color palettes, which are designed to be more aesthetically pleasing and modern.
- Marker is a symbol such as a symbol such as a small dot, square, diamond etc. that indicates a data point on the graph.
- We can draw multiple graphs in a single plot in two ways. One is by using subplot () function and other by superimposition of second graph on the first
- A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.
- Histplot is used basically for univariant set of observations and visualizes it through a histogram.
- A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as colour in two dimensions.
- Density plots are used to observe the distribution of a variable in a dataset.