

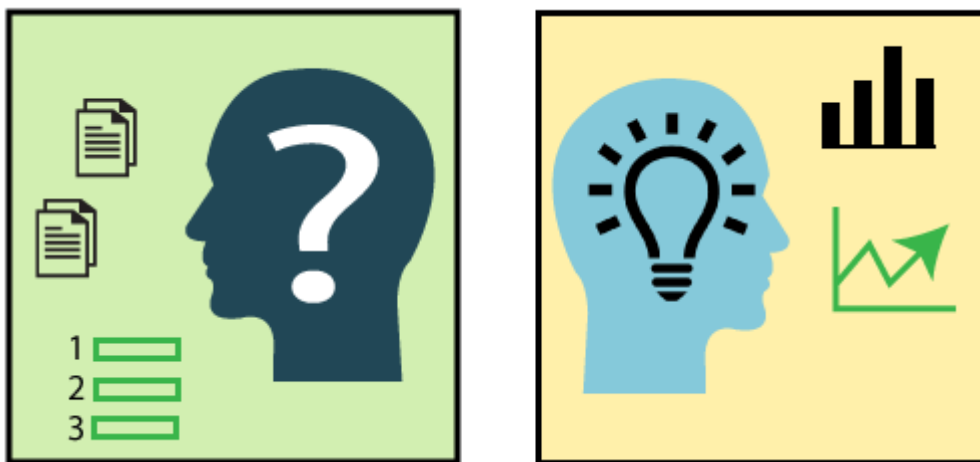
Module-5

Chapter 1:

Plotting and Visualization: A Brief matplotlib API Primer, Figures and Subplots, Plotting Functions in pandas, Plotting Maps

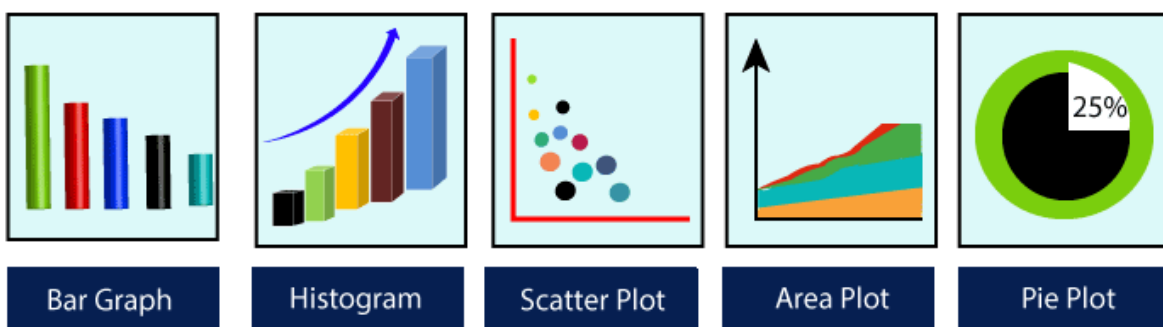
Data Visualization:

When it comes to presenting results, graphics provide an excellent method for exploring the data. The term "data visualization" is new. It conveys the idea that data representation goes beyond graphical representation (in place of textual representation).



This can help classify patterns, corrupt data, outliers, and a lot more when learning about a dataset and discovering it. Data visualizations can be used to express and demonstrate key relationships in plots and charts with a little domain knowledge. In fact, the static focuses on data estimation and quantitative description. It gives you a useful set of tools for understanding qualitatively.

There are five key plots that are used for data visualization.



The following five steps must be completed before the organization can decide:



- **Visualize:** By analyzing the raw data, complex data are made more user-friendly, understandable, and accessible. When a user needs to look up a specific measurement, a tabular data representation is used. A variety of charts are used to show patterns or relationships in the data for one or more variables.
- **Analysis:** Cleaning, inspecting, transforming, and modeling data to extract useful information is what is meant by the term "data analysis." Whether it's a business or personal decision, we always base our decisions on our past experiences. Analyzing our past is all that will be done when making a specific decision. That may change in the future, so any business or organization can make better decisions with the right analysis.
- **Document Analysis:** The process of organizing the useful data or information in a document in a standard format is known as document insight.
- **Change the Data Set:** To make the decision more effectively, standard data is used.

Why need data visualization?

Data visualization can perform below tasks:

- It identifies areas that need improvement and attention.
- It clarifies the factors.
- It helps to understand which product to place where.
- Predict sales volumes.

Benefits of Data Visualization:

Data visualization has a several advantages that help businesses and organizations make better decisions:

1. Developing methods for information absorption

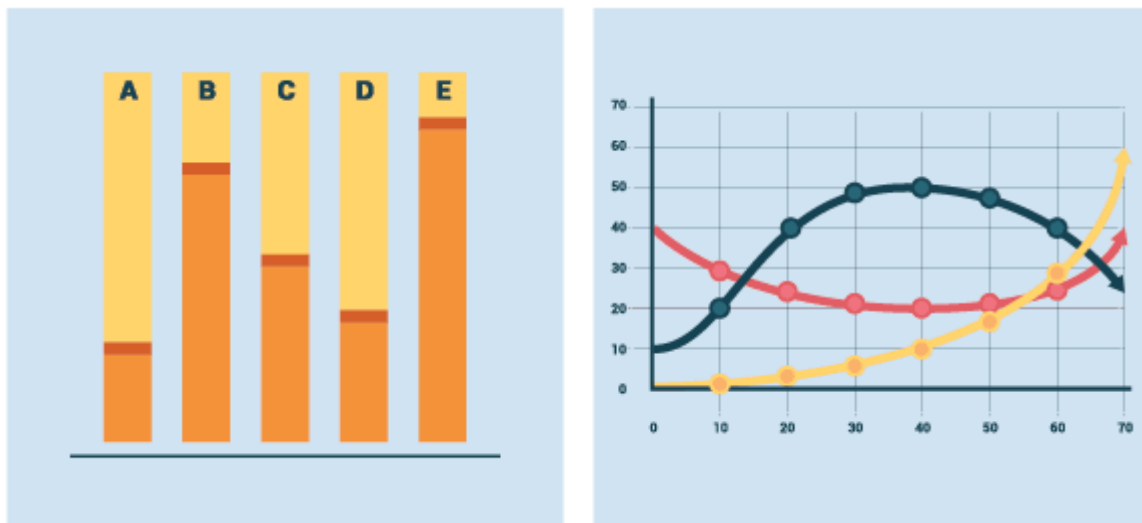
Data visualization gives users access to a vast amount of information about business and operational conditions. Understanding the connections between multidimensional data sets aids decision-makers. Using maps, fever charts, and other rich graphical representations, it provides novel approaches to data analysis.

Visual data discovery is more likely to locate the information that an organization requires and, as a result, to result in a company that is more productive than its rivals.

2. Visualize relationship and patterns in Businesses

In today's highly competitive business environment, finding a correlation between operating conditions and business performance is essential, which is a key advantage of data visualization.

The executives are able to quickly identify the problem's root cause thanks to their ability to make such correlations.



Assume a food organization is looking their month to month client information, and the information is given bar graphs, which shows that the organization's score has dropped by five in the earlier months in that specific district; The data suggest that customer satisfaction is an issue in this area.

3. **Accelerate your response** to emerging trends Data visualization enables decision-makers to more effectively comprehend shifts in customer behavior and market conditions across multiple data sets.

An emerging opportunity for the company to act on new business opportunities before their competitor is revealed by having an idea about the sentiments of the customer and other data.

4. Geological based Visualization

A Brief Matplotlib API Primer:

Compared to textual data, human minds are more adaptable to the visual representation of data. Things are easier to comprehend when they are visualized. The graph is a better way to represent the data because it allows us to analyze it more effectively and make specific decisions based on data analysis. Understanding data visualization and the significance of data visualization is necessary prior to learning matplotlib.



Matplotlib is a comprehensive Python library for creating static, interactive, and animated visualizations. It serves as a foundational tool in data analytics, enabling users to transform raw numerical data into insightful charts and graphs.

In 2002, John D. Hunter came up with the idea for matplotlib. It is distributed under a license similar to BSD and has a thriving community of developers. The most recent version, 3.1.1, was released on July 1, 2019, and the first version was released in 2003.

Up until June 23, 2007, Python versions 2.7 to 3.6 are supported by Matplotlib 2.0.x. Matplotlib 1.2 marked the beginning of Python3 support. The most recent version of Matplotlib that supports Python 2.6 is 1.4.

Matplotlib's functionality can be enhanced with the help of a variety of toolkits. Some of these tools can be moved using the matplotlib source code, while others require external dependencies and must be downloaded separately.

Core API Interfaces:

Matplotlib offers two primary ways to interact with its plotting engine:

- **Pyplot API (plt):** A state-based, implicit interface designed to mimic MATLAB's style. It is ideal for quick, interactive plotting and simple scripts because it automatically manages figures and axes behind the scenes.
- **Object-Oriented (OO) API:** An explicit interface where users directly manipulate Figure and Axes objects. This is the preferred method for complex visualizations, subplots, or when embedding plots into larger applications, as it provides finer control over every element.

Matplotlib Architecture:

There are three different layers in the architecture of the matplotlib which are the following:

- Backend Layer
- Artist layer
- Scripting layer

Backend layer:

The various functions that are required for plotting are implemented in the backend layer, which is the figure's bottom layer. From the backend layer, there are three essential classes: FigureCanvas, which is the surface on which the figure will be drawn; Renderer, which is the class that handles drawing on the surface; and Event, which handles mouse and keyboard events.

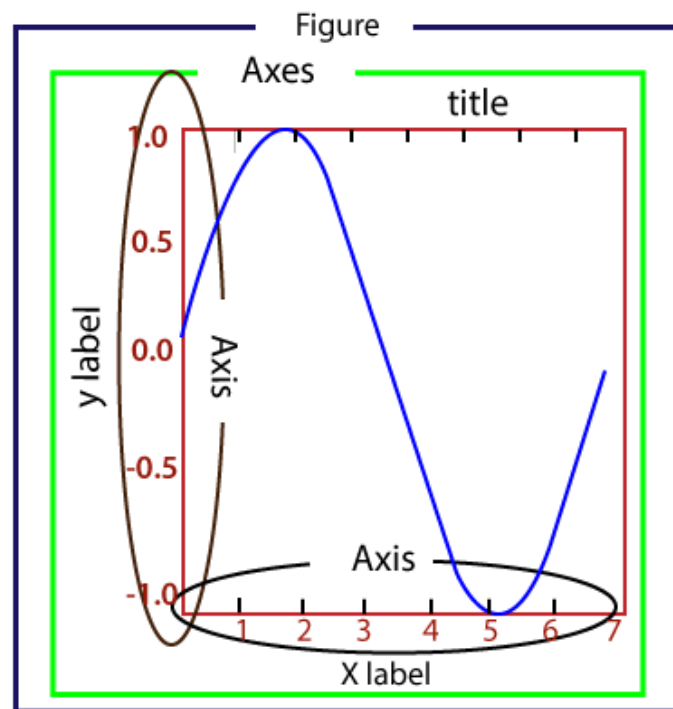
Artist Layer:

The architecture's second layer is the artist layer. It oversees the various plotting functions, such as the axis, which coordinates the renderer's use on the figure canvas.

Scripting layer:

The topmost layer, the scripting layer, is where the majority of our code will run. The methods in the scripting layer take care of the other layers almost automatically; all we need to worry about is the current state (figure and subplot).

The Overarching Idea of Matplotlib a Matplotlib figure can be broken down into the following parts:



Concept of Matplotlib in General Figure:

- It is a complete figure that could be holding multiple axes (plots). A Figure can be thought of as a canvas that holds plots.
- Axes: A figure can have multiple axes. In the case of 3D models, it is made up of two or three Axis objects. There is a title, an x-label, and a y-label in each Axes.
- Axis: Axes are the number of objects that look like lines and are in charge of creating the limits of the graph
- Artist: Text objects, Line2D objects, and collection objects are all manifestations of an artist. The majority of artists use axes.

Getting Started with Matplotlib Pyplot Api

Step 1: Import Matplotlib

In this step, we are importing matplotlib that we will use in further steps.

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

Step 2: Create Data

To visualize your desired data start by preparing it. This can include organizing a list of values, arrays or importing data from a source, like a CSV file.

Step 3: Create a Figure and Axes

Now, **create a figure and axes**; for your plot using the `plt.subplots()` function. This will provide you with the framework to create your representation.

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

Step 4: Plot Your Data

Create a representation of your data by choosing the plot type, such, as a line plot or scatter plot. Utilize the respective Pyplot function to generate your desired plot.

```
ax.plot(x_data, y_data, label='Data')
```

Step 5: Enhance Your Plot

In this step, enhance your plot; labels, titles, legends and additional annotations to create an visually appealing visualization.

```
ax.set_xlabel('X-axis Label')  
ax.set_ylabel('Y-axis Label')  
ax.set_title('Title')  
ax.legend()
```

Step 6: View or Store The Plot

Use `plt.show()` to display it in a window or utilize `plt.savefig('plot.png')` to save it as an image.

```
plt.show()
```

Figures and Subplots:

Matplotlib.figure.Figure.subplots() in Python:

The **figure module** provides the top-level Artist, the Figure, which contains all the plot elements. This module is used to control the default spacing of the subplots and top level container for all plot elements.

matplotlib.figure.Figure.subplots() method

The **subplots() method** figure module of matplotlib library is used to display the figure window.

Syntax: `subplots(self, nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None)`

Below examples illustrate the `matplotlib.figure.Figure.subplots()` function in `matplotlib.figure`:

Example 1:

Implementation of matplotlib function

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 400)

y = np.sin(x**2)

fig = plt.figure()

ax = fig.subplots()

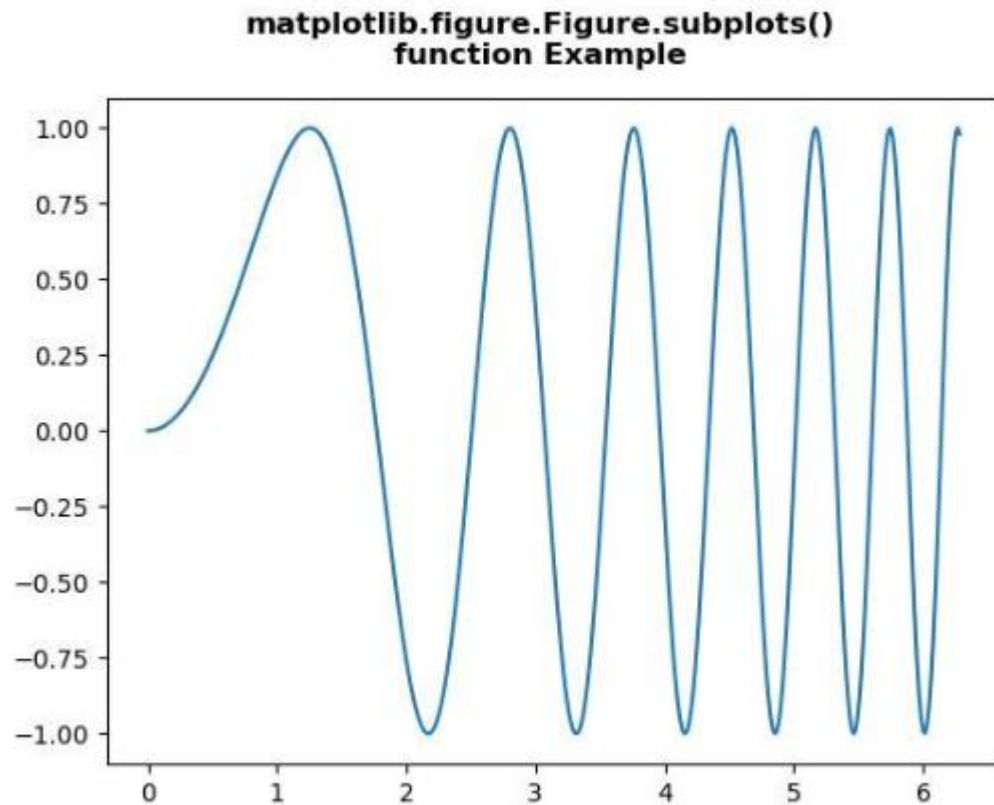
ax.plot(x, y)

fig.suptitle("""matplotlib.figure.Figure.subplots()

function Example\n\n""", fontweight="bold")

fig.show()
```

Output:



Example 2:

Implementation of matplotlib function

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 1.5 * np.pi, 100)

y = np.sin(x**2)+np.cos(x**2)

fig = plt.figure()

axs = fig.subplots(2, 2, subplot_kw = dict(polar = True))

axs[0, 0].plot(x, y)

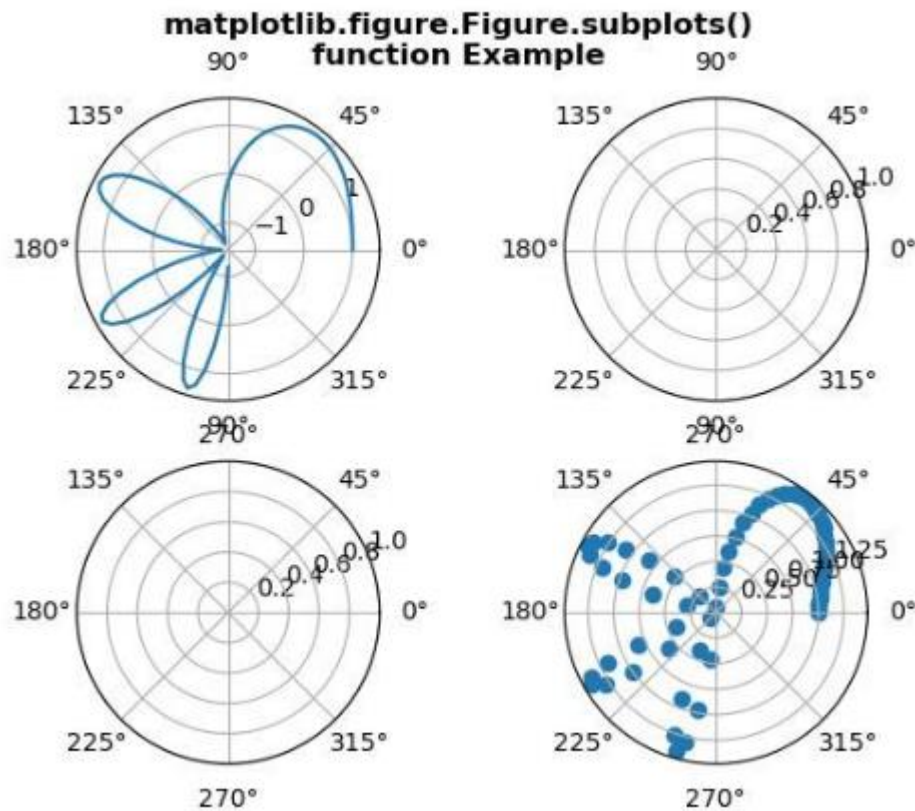
axs[1, 1].scatter(x, y)

fig.suptitle("""matplotlib.figure.Figure.subplots()

function Example\n\n""", fontweight = "bold")

fig.show()
```


Output:



Plotting Functions in pandas :

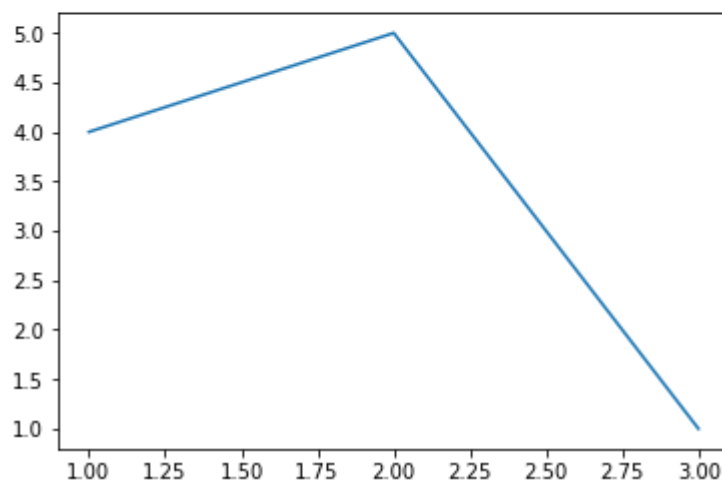
Pandas has a range of charting methods that are based on the matplotlib package. This allows for the convenient creation of charts straight from DataFrame objects. These functions include a diverse array of plot forms, such as line graphs, bar plots, histograms, scatter plots, and further variations. By using these functions, users may effectively depict trends, distributions, correlations, and linkages within their data.

Pandas plotting capabilities facilitate the process of data visualization, making it smooth and effortless. Users may easily invoke the required charting function on a DataFrame or Series object and modify the plot using different parameters. In addition, Pandas seamlessly interfaces with Matplotlib, enabling advanced customization and precise adjustments of visuals.

Basic Example of plotting Graph

```
from matplotlib import pyplot as plt  
#plotting our canvas  
plt.plot([1,2,3],[4,5,1])  
#display the graph  
plt.show()
```

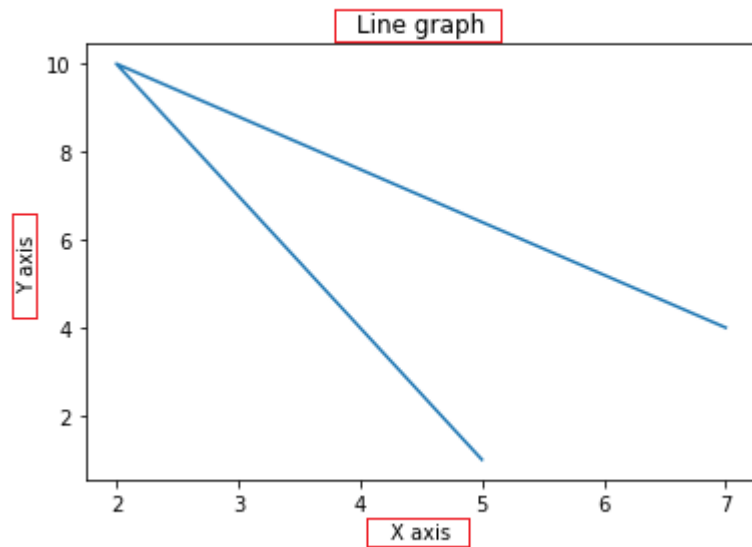
Output:



It takes only three lines to plot a simple graph using the Python matplotlib. We can add titles, labels to our chart which are created by Python matplotlib library to make it more meaningful. The example is the following:

```
from matplotlib import pyplot as plt  
  
x = [5, 2, 7]  
y = [1, 10, 4]  
plt.plot(x, y)  
plt.title('Line graph')  
plt.ylabel('Y axis')  
plt.xlabel('X axis')  
plt.show()
```

Output:

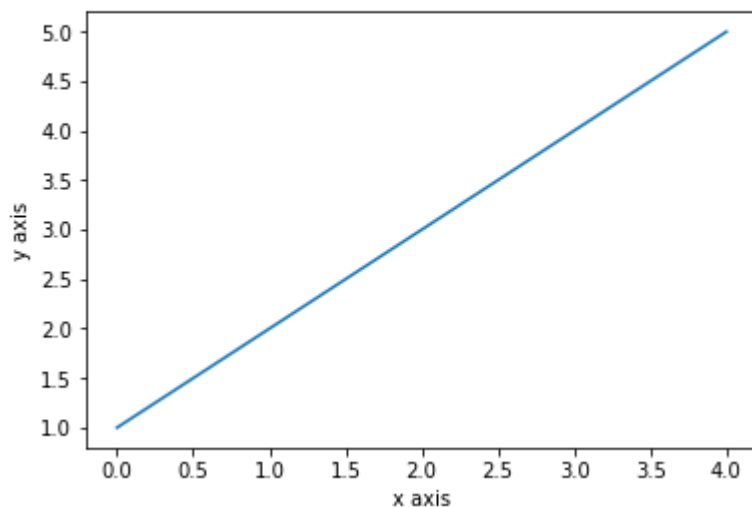


The graph is more understandable from the previous graph.

The `plot()` function, which is frequently used to plot a graph, is provided by the `pyplot` module. Let's look at a straightforward illustration:

```
from matplotlib import pyplot as plt
plt.plot([1,2,3,4,5])
plt.ylabel("y axis")
plt.xlabel("x axis")
plt.show()
```

Output:



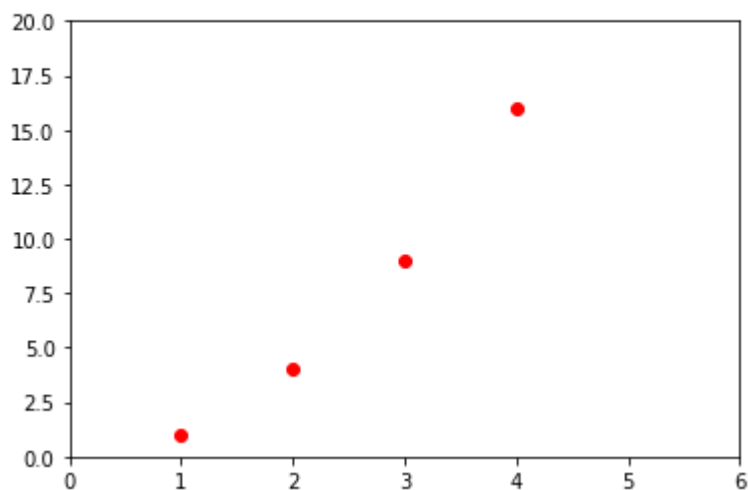
The x-axis and y-axis of the graph are plotted using the program, and they range from 0 to 5. Matplotlib automatically generates the x values if we provide a single list to the plot() function, assuming that it is a sequence of y values. The default x vector has the same length as the y vector but starts at 0, as we know that the Python index starts at 0. As a result, [0, 1, 2, 3, 4,] are the x data.

Formatting the style of the plot

The third argument is optional and is a format string that specifies the plot's color and line type. As can be seen in the graph that has been plotted above, the default format string is "b-," which is the solid blue. Consider the following illustration, in which the graph with the red circle is plotted.

1. from matplotlib **import** pyplot as plt
2. plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25], 'ro')
3. plt.axis([0, 6, 0, 20])
4. plt.show()

Output:



Example format String

'b'	Using for the blue marker with default shape.
'ro'	Red circle
'-g'	Green solid line
'--'	A dashed line with the default color
'^k:'	Black triangle up markers connected by a dotted line

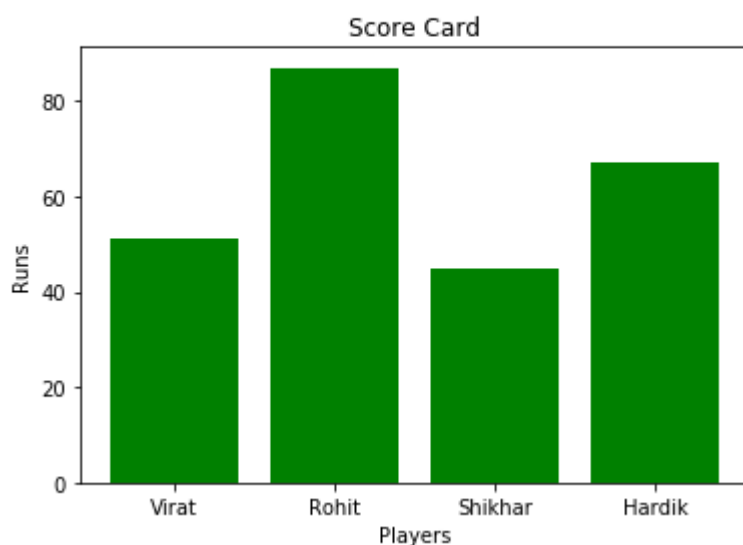
The matplotlib supports the following color abbreviation:

Bar graphs:

Bar graphs are one of the most common types of graphs and are used to show data associated with the categorical variables. Matplotlib provides a **bar()** to make bar graphs which accepts arguments such as: categorical variables, their value and color.

```
from matplotlib import pyplot as plt
players = ['Virat','Rohit','Shikhar','Hardik']
runs = [51,87,45,67]
plt.bar(players,runs,color = 'green')
plt.title('Score Card')
plt.xlabel('Players')
plt.ylabel('Runs')
plt.show()
```

Output:



Pie Chart:

A pie chart is a circular graph that is broken down in the segment or slices of pie. It is generally used to represent the percentage or proportional data where each slice of pie represents a particular category. Let's have a look at the below example:

```

from matplotlib import pyplot as plt

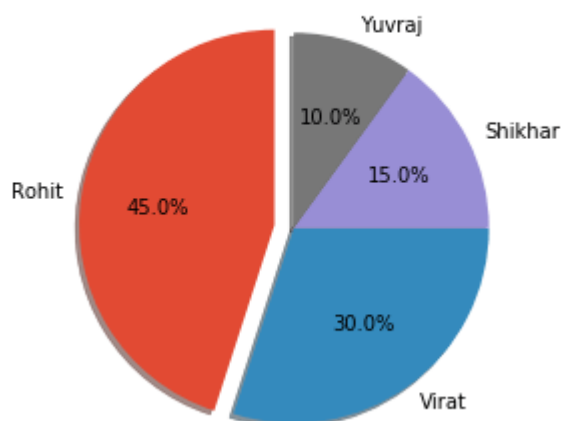
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
Players = 'Rohit', 'Virat', 'Shikhar', 'Yuvraj'
Runs = [45, 30, 15, 10]
explode = (0.1, 0, 0, 0) # it "explodes" the 1st slice

fig1, ax1 = plt.subplots()
ax1.pie(Runs, explode=explode, labels=Players, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

```

Output:



Histogram:

Understanding the distinction between a bar graph and a histogram is the first step. A bar chart is used to compare various entities, while a histogram is used for the distribution. A histogram is a type of bar plot that compares the frequency of a set of values to a range of values.

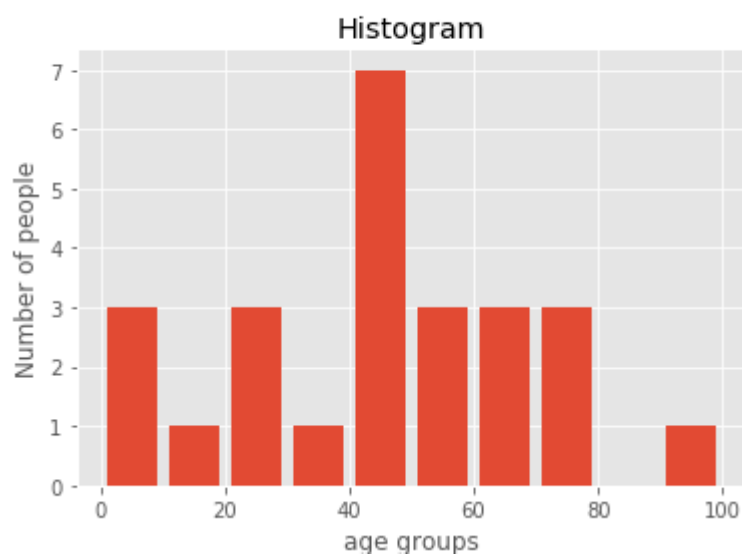
For instance, we plot a histogram with respect to the bin using data from various age groups. Now, the range of values divided into a series of intervals is represented by the bin. Most bins are made of the same size.

```

from matplotlib import pyplot as plt
from matplotlib import pyplot as plt
population_age = [21,53,60,49,25,27,30,42,40,1,2,102,95,8,15,105,70,65,55,70,75,60,52,44,43,42,45]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar', rwidth=0.8)
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()

```

Output:



Let's consider the another example of plotting histogram:

```

from matplotlib import pyplot as plt
# Importing Numpy Library
import numpy as np
plt.style.use('fivethirtyeight')

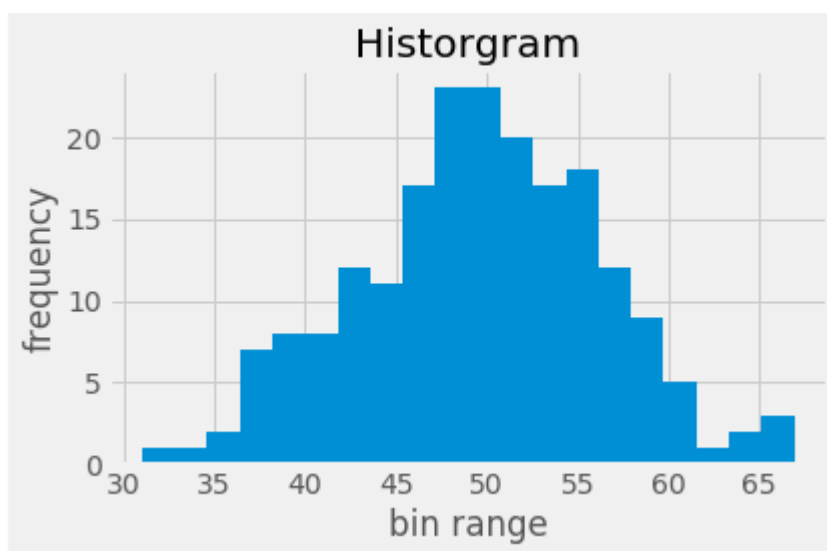
mu = 50
sigma = 7
x = np.random.normal(mu, sigma, size=200)
fig, ax = plt.subplots()

ax.hist(x, 20)
ax.set_title('Histogram')
ax.set_xlabel('bin range')
ax.set_ylabel('frequency')

fig.tight_layout()
plt.show()

```

Output:



Scatter plot:

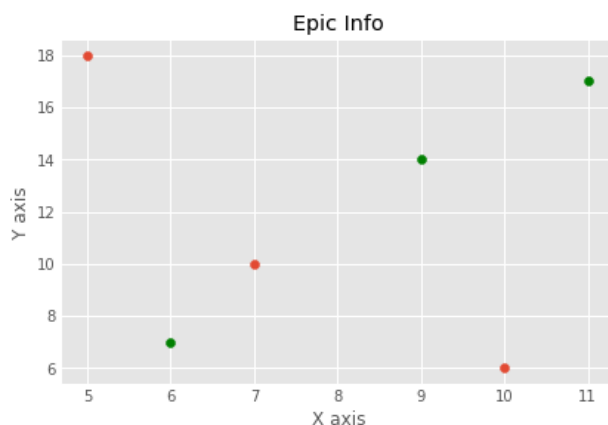
When determining how much one variable affects another, scatter plots are typically used to compare variables. The data are presented as a series of points. The value of one variable, which determines the point's position along the horizontal and vertical axes, is assigned to each point. The value of the other variable, on the other hand, is assigned to each point.

Let's consider the following simple example:

Example-1:

```
1. from matplotlib import pyplot as plt
2. from matplotlib import style
3. style.use('ggplot')
4.
5. x = [5,7,10]
6. y = [18,10,6]
7.
8. x2 = [6,9,11]
9. y2 = [7,14,17]
10.
11. plt.scatter(x, y)
12.
13. plt.scatter(x2, y2, color='g')
14.
15. plt.title('Epic Info')
16. plt.ylabel('Y axis')
17. plt.xlabel('X axis')
18.
19. plt.show()
```

Output:



Example-2

```
import matplotlib.pyplot as plt
x = [2, 2.5, 3, 3.5, 4.5, 4.7, 5.0]
y = [7.5, 8, 8.5, 9, 9.5, 10, 10.5]

x1 = [9, 8.5, 9, 9.5, 10, 10.5, 12]
y1 = [3, 3.5, 4.7, 4, 4.5, 5, 5.2]
plt.scatter(x, y, label='high income low saving', color='g')
plt.scatter(x1, y1, label='low income high savings', color='r')
plt.xlabel('saving*100')
plt.ylabel('income*1000')
plt.title('Scatter Plot')
plt.legend()
plt.show()
```

Output:

