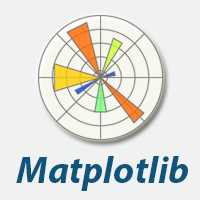
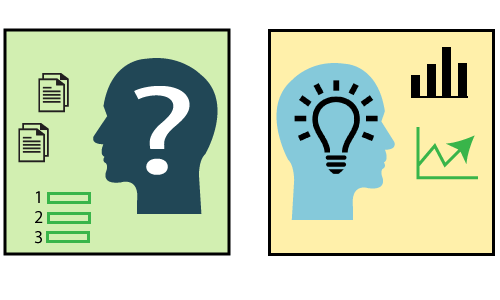
Matplotlib (Python Plotting Library)



Human minds are more adaptive for the visual representation of data rather than textual data. We can easily understand things when they are visualized. It is better to represent the data through the graph where we can analyze the data more efficiently and make the specific decision according to data analysis. Before learning the matplotlib, we need to understand data visualization and why data visualization is important.

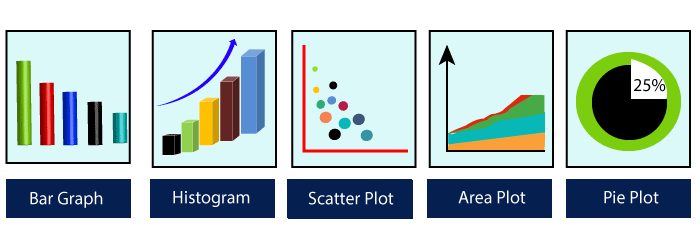
Data Visualization



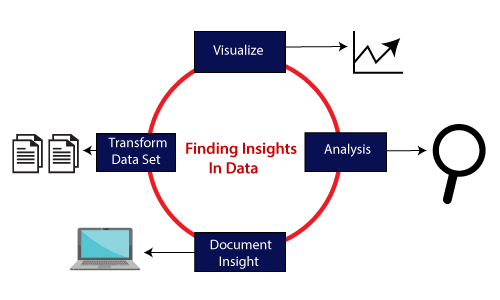
Graphics provides an excellent approach for exploring the data, which is essential for presenting results. Data visualization is a new term. It expresses the idea that involves more than just representing data in the graphical form (instead of using textual form).

This can be very helpful when discovering and getting to know a dataset and can help with classifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts. The static does indeed focus on quantitative description and estimations of data. It provides an important set of tools for gaining a qualitative understanding.

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

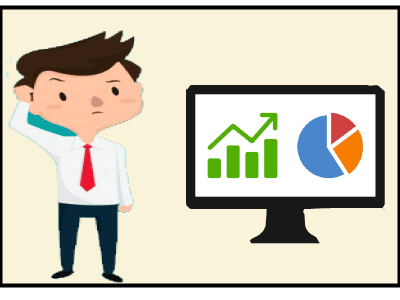


here are five phases which are essential to make the decision for the organization:



* **Visualize:** We analyze the raw data, which means it makes complex data more accessible, understandable, and more usable. Tabular data representation is used where the user will look up a specific measurement, while the chart of several types is used to show patterns or relationships in the data for one or more variables.
* **Analysis:** Data analysis is defined as cleaning, inspecting, transforming, and modeling data to derive useful information. Whenever we make a decision for the business or in daily life, is by past experience. **What will happen to choose a particular decision**, it is nothing but analyzing our past. That may be affected in the future, so the proper analysis is necessary for better decisions for any business or organization.
* **Document Insight:** Document insight is the process where the useful data or information is organized in the document in the standard format.
* **Transform Data Set:** Standard data is used to make the decision more effectively.

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.

Benefit of Data Visualization

Here are some benefits of the data visualization, which helps to make an effective decision for the organizations or business:

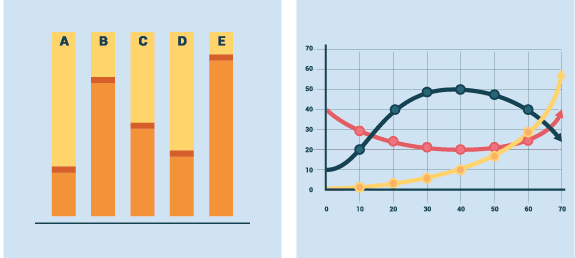
**1. Building ways of absorbing information**

Data visualization allows users to receive vast amounts of information regarding operational and business conditions. It helps decision-makers to see the relationship between multi-dimensional data sets. It offers new ways to analyses data through the use of maps, fever charts, and other rich graphical representations.

Visual data discovery is more likely to find the information that the organization needs and then end up with being more productive than other competitive companies.

**2. Visualize relationship and patterns in Businesses**

The crucial advantage of data visualization is that it is essential to find the correlation between operating conditions and business performance in today's highly competitive business environment.



The ability to make these types of correlations enables the executives to identify the root cause of the problem and act quickly to resolve it.

Suppose a food company is looking their monthly customer data, and the data is presented with bar charts, which shows that the company's score has dropped by five points in the previous months in that particular region; the data suggest that there's a problem with customer satisfaction in this area.

**3. Take action on the emerging trends faster**

Data visualization allows the decision-maker to grasp shifts in customer behavior and market conditions across multiple data sets more efficiently.

Having an idea about the customer's sentiments and other data discloses an emerging opportunity for the company to act on new business opportunities ahead of their competitor.

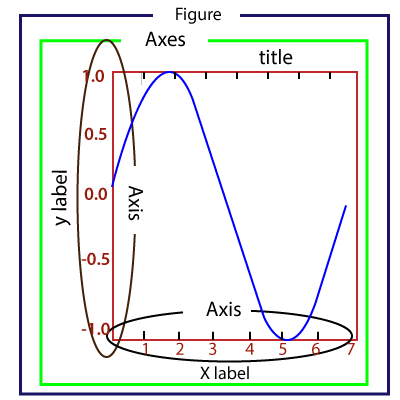
**4. Geological based Visualization**

Geo-spatial visualization is occurred due to many websites providing web-services, attracting visitor's interest. These types of websites are required to take benefit of location-specific information, which is already present in the customer details.

**Matplotlib** is a Python library which is defined as a multi-platform data visualization library built on Numpy array. It can be used in python scripts, shell, web application, and other graphical user interface toolkit.

The General Concept of Matplotlib

A Matplotlib figure can be categorized into various parts as below:



**Figure:** It is a whole figure which may hold one or more axes (plots). We can think of a Figure as a canvas that holds plots.

**Axes:** A Figure can contain several Axes. It consists of two or three (in the case of 3D) Axis objects. Each Axes is comprised of a title, an x-label, and a y-label.

**Axis:** Axises are the number of line like objects and responsible for generating the graph limits.

**Artist:** An artist is the all which we see on the graph like Text objects, Line2D objects, and collection objects. Most Artists are tied to Axes.

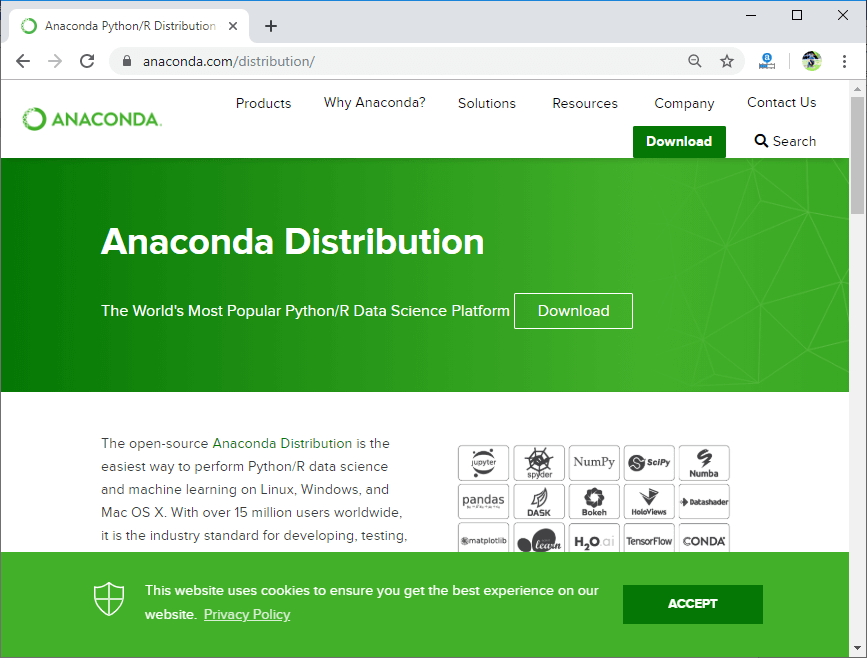
## Installing Matplotlib

Before start working with the Matplotlib or its plotting functions first, it needs to be installed. The installation of matplotlib is dependent on the distribution that is installed on your computer. These installation methods are following:

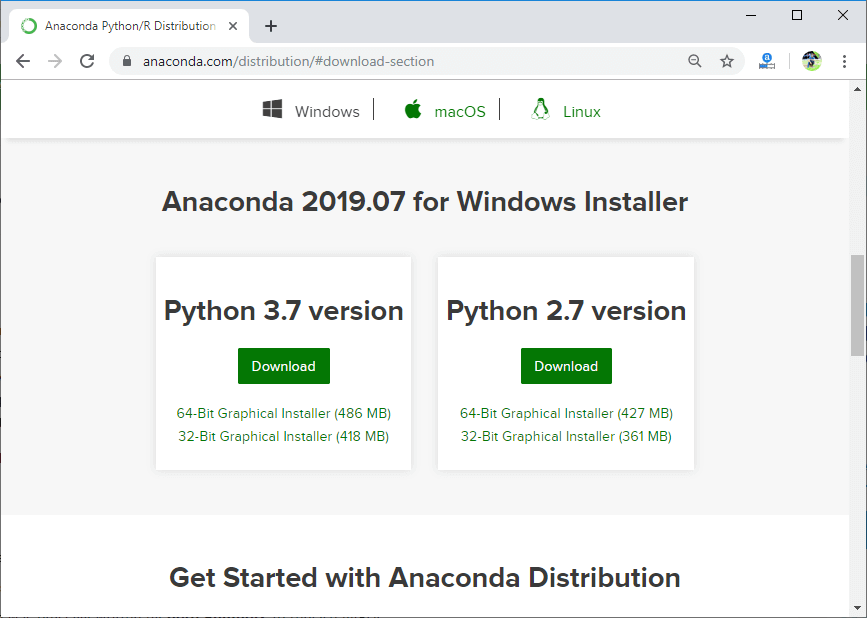
**Use the Anaconda distribution of Python**

The easiest way to install Matplotlib is to download the Anaconda distribution of Python. Matplotlib is pre-installed in the anaconda distribution No further installation steps are necessary.

* Visit the official site of Anaconda and click on the Download Button



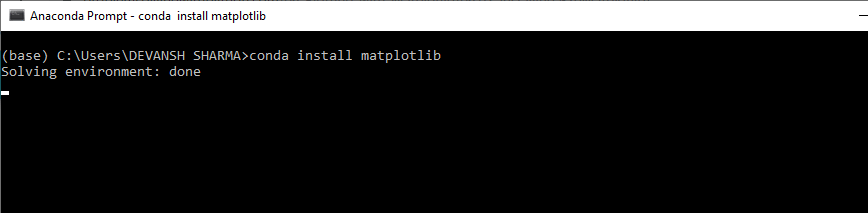
* Choose download according to your Python interpreter configuration.



**Install Matplotlib using with Anaconda Prompt**

Matplotlib can be installed using with the Anaconda Prompt by typing command. To install matplotlib, open Anaconda Prompt and type the following command:

1. conda install matplotlib



**Install Matplotlib with pip**

The python package manager pip is also used to install matplotlib. Open the command prompt window, and type the following command:

1. pip install matplotlib

## Verify the Installation

To verify that matplotlib is installed properly or not, type the following command includes calling .\_\_version \_\_ in the terminal.

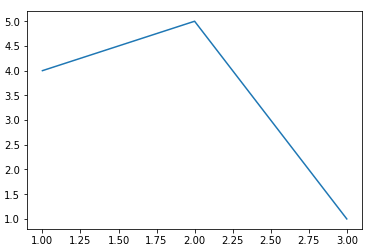
1. **import** matplotlib
2. matplotlib.\_\_version\_\_
3. '3.1.1'

## Basic Example of plotting Graph

Here is the basic example of generating a simple graph; the program is following:

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

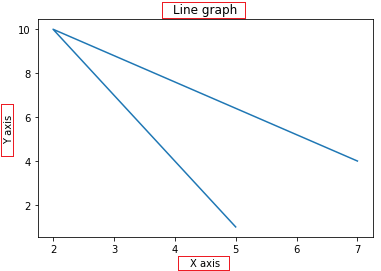
**Output:**



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:

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

**Output:**



The graph is more understandable from the previous graph.

## Working with Pyplot

The **matplotlib.pyplot** is the collection command style functions that make matplotlib feel like working with MATLAB. The pyplot functions are used to make some changes to figure such as create a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot including labels, etc.

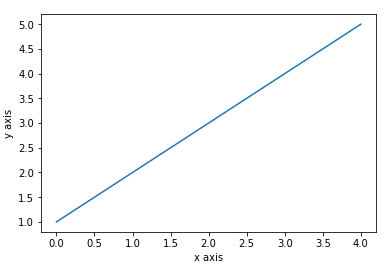
It is good to use when we want to plot something quickly without instantiating any figure or Axes.

While working with **matplotlib.pyplot**, some states are stored across function calls so that it keeps track of the things like current figure and plotting area, and these plotting functions are directed to the current axes.

The pyplot module provide the **plot()** function which is frequently use to plot a graph. Let's have a look on the simple example:

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

**Output:**

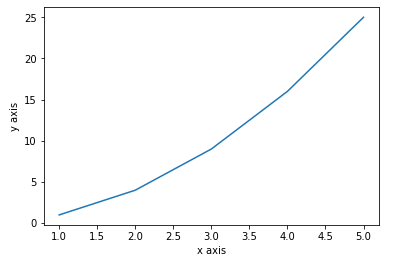


In the above program, it plots the graph x-axis ranges from 0-4 and the y-axis from 1-5. If we provide a single list to the plot(), matplotlib assumes it is a sequence of y values, and automatically generates the x values. Since we know that python index starts at 0, the default x vector has the same length as y but starts at 0. Hence the x data are [0, 1, 2, 3, 4].

We can pass the arbitrary number of arguments to the plot(). For example, to plot x versus y, we can do this following way:

1. from matplotlib **import** pyplot as plt
2. plt.plot([1,2,3,4,5],[1,4,9,16,25])
3. plt.ylabel("y axis")
4. plt.xlabel('x axis')
5. plt.show()

**Output:**



**Formatting the style of the plot**

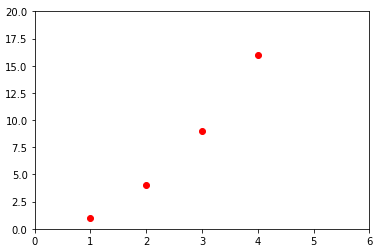
There is an optional third argument, which is a format string that indicates the color and line type of the plot. The default format string is '**b-**'which is the solid blue as you can observe in the above plotted graph. Let's consider the following example where we plot the graph with the red circle.

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])   #You can set the axis limits by passing a list or tuple of

four values: plt.axis([xmin, xmax, ymin, ymax])

1. 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:

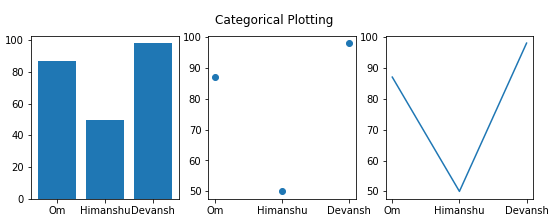
|  |  |
| --- | --- |
| **Character** | **Color** |
| **'b' Blue** |  |
| **'g'** | Green |
| **'r'** | Red |
| **'c'** | Cyan |
| **'m'** | Magenta |
| **'y'** | Yellow |
| **'k'** | Black |
| **'w'** | White |

### **Plotting with categorical variables**

Matplotlib allows us to pass categorical variables directly to many plotting functions: consider the following example

1. from matplotlib **import** pyplot
2. names = ['Abhishek', 'Himanshu', 'Devansh']
3. marks= [87,50,98]
5. plt.figure(figsize=(9,3))
7. plt.subplot(131)
8. plt.bar(names, marks)
9. plt.subplot(132)
10. plt.scatter(names, marks)
11. plt.subplot(133)
12. plt.plot(names, marks)
13. plt.suptitle('Categorical Plotting')
14. plt.show()

**Output:**



In the above program, we have plotted the categorical graph using the **subplot()** function. Let's a have a look on the subplot() function.

## What is subplot()

The Matplotlib **subplot()** function is defined as to plot two or more plots in one figure. We can use this method to separate two graphs which plotted in the same axis Matplotlib supports all kinds of subplots, including 2x1 vertical, 2x1 horizontal, or a 2x2 grid.

It accepts the three arguments: they are **nrows, ncols, and index**. It denote the number of rows, number of columns and the index.

The subplot() function can be called in the following way:

1. subplot(nrows,ncols,index,\*\*kwargs)
2. subplot(pos,\*\*kwargs)
3. subplot(ax)

**Parameters:**

* **\*args:**

Three separate integers or three-digit integer describes the position of the subplot. If the three integers are **nrows, ncols, and index** in order, the subplot will take the index position on a grid with **nrows row** and **ncol column**.

The argument **pos** are a three-digit integer, where the first digit is denoted the number of rows, the second digit denoted the number of columns, and the third represents the index of the subplot. **For example,** subplot (1, 3, 2) is the same as the subplot (132).

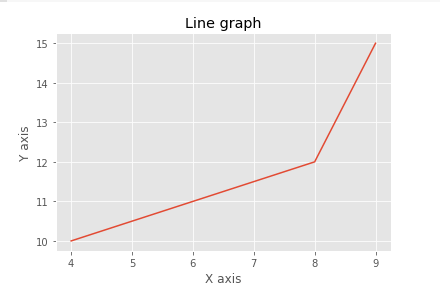
## Creating different types of graph

### **1. Line graph**

The line graph is one of charts which shows information as a series of the line. The graph is plotted by the plot() function. The line graph is simple to plot; let's consider the following example:

1. from matplotlib **import** pyplot as plt
3. x = [4,8,9]
4. y = [10,12,15]
6. plt.plot(x,y)
8. plt.title("Line graph")
9. plt.ylabel('Y axis')
10. plt.xlabel('X axis')
11. plt.show()

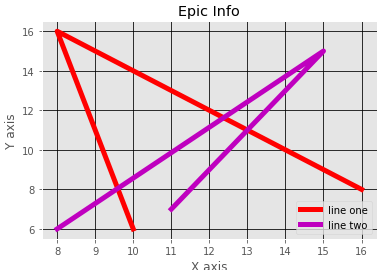
**Output:**



We can customize the graph by importing the style module. The style module will be built into a matplotlib installation. It contains the various functions to make the plot more attractive. In the below program, we are using the style module:

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** style
4. style.use('ggplot')
5. x = [16, 8, 10]
6. y = [8, 16, 6]
7. x2 = [8, 15, 11]
8. y2 = [6, 15, 7]
9. plt.plot(x, y, 'r', label='line one', linewidth=5)
10. plt.plot(x2, y2, 'm', label='line two', linewidth=5)
11. plt.title('Epic Info')
12. fig = plt.figure()
13. plt.ylabel('Y axis')
14. plt.xlabel('X axis')
15. plt.legend()
16. plt.grid(True, color='k')
17. plt.show()

**Output:**



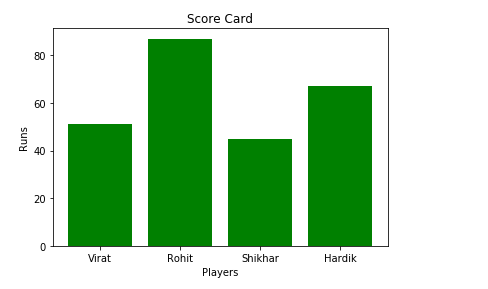
In Matplotlib, the figure (an instance of class plt.Figure) can be supposed of as a single container that consists of all the objects denoting axes, graphics, text, and labels.

### **2. 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.

1. from matplotlib **import** pyplot as plt
2. players = ['Virat','Rohit','Shikhar','Hardik']
3. runs = [51,87,45,67]
4. plt.bar(players,runs,color = 'green')
5. plt.title('Score Card')
6. plt.xlabel('Players')
7. plt.ylabel('Runs')
8. plt.show()

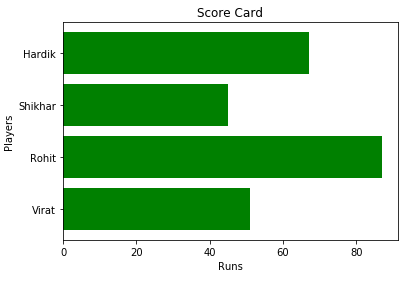
**Output:**



Another function **barh()** is used to make horizontal bar graphs. It accepts **xerr** or **yerr** as arguments (in case of vertical graphs) to depict the variance in our data as follows:

1. from matplotlib **import** pyplot as plt
2. players = ['Virat','Rohit','Shikhar','Hardik']
3. runs = [51,87,45,67]
4. plt.barh(players,runs, color = 'green')
5. plt.title('Score Card')
6. plt.xlabel('Players')
7. plt.ylabel('Runs')
8. plt.show()

**Output:**



Let's have a look on the other example using the **style()** function:

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** style
4. style.use('ggplot')
6. x = [5,8,10]
7. y = [12,16,6]
9. x2 = [6,9,11]
10. y2 = [7,15,7]

13. plt.bar(x, y, color = 'y', align='center')
14. plt.bar(x2, y2, color='c', align='center')
16. plt.title('Information')
18. plt.ylabel('Y axis')
19. plt.xlabel('X axis')

**Output:**

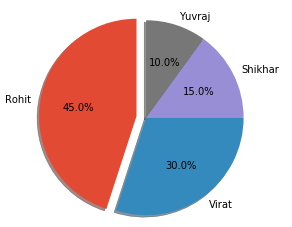


### **3. Pie Chart**

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:

1. from matplotlib **import** pyplot as plt
3. # Pie chart, where the slices will be ordered and plotted counter-clockwise:
4. Players = 'Rohit', 'Virat', 'Shikhar', 'Yuvraj'
5. Runs = [45, 30, 15, 10]
6. explode = (0.1, 0, 0, 0)  # it "explode" the 1st slice
8. fig1, ax1 = plt.subplots()
9. ax1.pie(Runs, explode=explode, labels=Players, autopct='%1.1f%%',
10. shadow=True, startangle=90)
11. ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
13. plt.show()

**Output:**



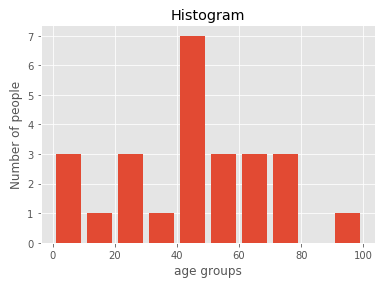
### **4. Histogram**

First, we need to understand the difference between the bar graph and histogram. A histogram is used for the distribution, whereas a bar chart is used to compare different entities. A histogram is a type of bar plot that shows the frequency of a number of values compared to a set of values ranges.

**For example** we take the data of the different age group of the people and plot a histogram with respect to the bin. Now, bin represents the range of values that are divided into series of intervals. Bins are generally created of the same size.

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** pyplot as plt
3. 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]
4. bins = [0,10,20,30,40,50,60,70,80,90,100]
5. plt.hist(population\_age, bins, histtype='bar', rwidth=0.8)
6. plt.xlabel('age groups')
7. plt.ylabel('Number of people')
8. plt.title('Histogram')
9. plt.show()

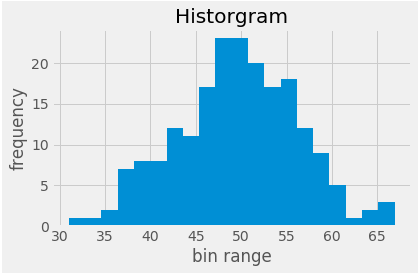
**Output:**



Let's consider the another example of plotting histogram:

1. from matplotlib **import** pyplot as plt
2. # Importing Numpy Library
3. **import** numpy as np
4. plt.style.use('fivethirtyeight')
6. mu = 50
7. sigma = 7
8. x = np.random.normal(mu, sigma, size=200)
9. fig, ax = plt.subplots()
11. ax.hist(x, 20)
12. ax.set\_title('Historgram')
13. ax.set\_xlabel('bin range')
14. ax.set\_ylabel('frequency')
16. fig.tight\_layout()
17. plt.show()

**Output:**



### **5. Scatter plot**

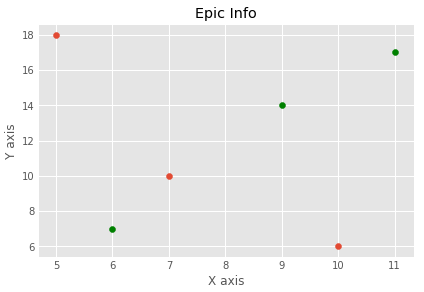
The scatter plots are mostly used for comparing variables when we need to define how much one variable is affected by another variable. The data is displayed as a collection of points. Each point has the value of one variable, which defines the position on the horizontal axes, and the value of other variable represents the position on the vertical axis.

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')
5. x = [5,7,10]
6. y = [18,10,6]
8. x2 = [6,9,11]
9. y2 = [7,14,17]
11. plt.scatter(x, y)
13. plt.scatter(x2, y2, color='g')
15. plt.title('Epic Info')
16. plt.ylabel('Y axis')
17. plt.xlabel('X axis')
19. plt.show()

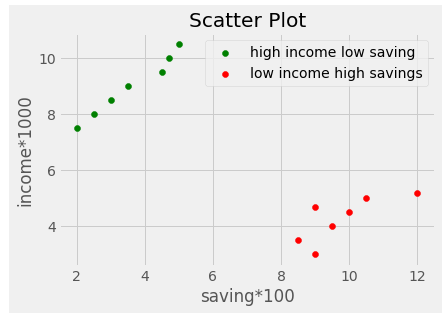
**Output:**



**Example-2**

1. **import** matplotlib.pyplot as plt
2. x = [2, 2.5, 3, 3.5, 4.5, 4.7, 5.0]
3. y = [7.5, 8, 8.5, 9, 9.5, 10, 10.5]
5. x1 = [9, 8.5, 9, 9.5, 10, 10.5, 12]
6. y1 = [3, 3.5, 4.7, 4, 4.5, 5, 5.2]
7. plt.scatter(x, y, label='high income low saving', color='g')
8. plt.scatter(x1, y1, label='low income high savings', color='r')
9. plt.xlabel('saving\*100')
10. plt.ylabel('income\*1000')
11. plt.title('Scatter Plot')
12. plt.legend()
13. plt.show()

**Output:**



**Python Pandas**



The term "Pandas" refers to an open-source library for manipulating high-performance data in Python. This instructional exercise is intended for the two novices and experts.

It was created in 2008 by Wes McKinney and is used for data analysis in Python. Pandas is an open-source library that provides high-performance data manipulation in Python. All of the basic and advanced concepts of Pandas, such as Numpy, data operation, and time series, are covered here.

Pandas Introduction

The name of Pandas is gotten from the word Board Information, and that implies an Econometrics from Multi-faceted information. It was created in 2008 by Wes McKinney and is used for data analysis in Python.

Processing, such as restructuring, cleaning, merging, etc., is necessary for data analysis. Numpy, Scipy, Cython, and Panda are just a few of the fast data processing tools available. Yet, we incline toward Pandas since working with Pandas is quick, basic and more expressive than different apparatuses.

## Key Features of Pandas

* It has a DataFrame object that is quick and effective, with both standard and custom indexing.
* Utilized for reshaping and turning of the informational indexes.
* For aggregations and transformations, group by data.
* It is used to align the data and integrate the data that is missing.
* Provide Time Series functionality.
* Process a variety of data sets in various formats, such as matrix data, heterogeneous tabular data, and time series.
* Manage the data sets' multiple operations, including subsetting, slicing, filtering, groupBy, reordering, and reshaping.
* It incorporates with different libraries like SciPy, and scikit-learn.
* Performs quickly, and the Cython can be used to accelerate it even further.

## Benefits of Pandas

The following are the advantages of pandas overusing other languages:

**Representation of Data:** Through its DataFrame and Series, it presents the data in a manner that is appropriate for data analysis.

**Clear code:** Pandas' clear API lets you concentrate on the most important part of the code. In this way, it gives clear and brief code to the client.

DataFrame and Series are the two data structures that Pandas provides for processing data. These data structures are discussed below:

### **1) Series**

A one-dimensional array capable of storing a variety of data types is how it is defined. The term "index" refers to the row labels of a series. We can without much of a stretch believer the rundown, tuple, and word reference into series utilizing "series' technique. Multiple columns cannot be included in a Series. Only one parameter exists:

**Data:** It can be any list, dictionary, or scalar value.

**Creating Series from Array:**

Before creating a Series, Firstly, we have to import the numpy module and then use array() function in the program.

1. **import** pandas as pd
2. **import** numpy as np
3. info = np.array(['P','a','n','d','a','s'])
4. a = pd.Series(info)
5. **print**(a)

**Output**

0 P

1 a

2 n

3 d

4 a

5 s

dtype: object

**Explanation:** In this code, firstly, we have imported the **pandas** and **numpy** library with the **pd** and **np** alias. Then, we have taken a variable named "info" that consist of an array of some values. We have called the **info** variable through a **Series** method and defined it in an "**a**" variable. The Series has printed by calling the **print(a)** method.

## Python Pandas DataFrame

It is a generally utilized information design of pandas and works with a two-layered exhibit with named tomahawks (lines and segments). As a standard method for storing data, DataFrame has two distinct indexes-row index and column index. It has the following characteristics:

The sections can be heterogeneous sorts like int, bool, etc.

It can be thought of as a series structure dictionary with indexed rows and columns. It is referred to as "columns" for rows and "index" for columns.

**Create a DataFrame using List:**

ADVERTISEMENT

We can easily create a DataFrame in Pandas using list.

1. **import** pandas as pd
2. # a list of strings
3. x = ['Python', 'Pandas']
5. # Calling DataFrame constructor on list
6. df = pd.DataFrame(x)
7. **print**(df)

**Output**

0

0 Python

1 Pandas

**Explanation:** In this code, we have characterized a variable named "x" that comprise of string values. On a list, the values are being printed by calling the DataFrame constructor.

**Create a Series from dict**

We can also create a Series from dict. **If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index**.

If index is passed, then values correspond to a particular label in the index will be extracted from the **dictionary**.

1. #**import** the pandas library
2. **import** pandas as pd
3. **import** numpy as np
4. info = {'x' : 0., 'y' : 1., 'z' : 2.}
5. a = pd.Series(info)
6. print (a)

**Output**

x 0.0

y 1.0

z 2.0

dtype: float64

**Create a Series using Scalar:**

If we take the scalar values, then the index must be provided. The scalar value will be repeated for matching the length of the index.

1. #**import** pandas library
2. **import** pandas as pd
3. **import** numpy as np
4. x = pd.Series(4, index=[0, 1, 2, 3])
5. print (x)

**Output**

ADVERTISEMENT

0 4

1 4

2 4

3 4

dtype: int64

## Accessing data from series with Position:

Once you create the Series type object, you can access its indexes, data, and even individual elements.

The data in the Series can be accessed similar to that in the ndarray.

1. **import** pandas as pd
2. x = pd.Series([1,2,3],index = ['a','b','c'])
3. #retrieve the first element
4. print (x[0])

ADVERTISEMENT

**Output**

1

### **Series object attributes**

The Series attribute is defined as any information related to the Series object such as size, datatype. etc. Below are some of the attributes that you can use to get the information about the Series object:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| **Series.index** | Defines the index of the Series. |
| **Series.shape** | It returns a tuple of shape of the data. |
| **Series.dtype** | It returns the data type of the data. |
| **Series.size** | It returns the size of the data. |
| **Series.empty** | It returns True if Series object is empty, otherwise returns false. |
| **Series.hasnans** | It returns True if there are any NaN values, otherwise returns false. |
| **Series.nbytes** | It returns the number of bytes in the data. |
| **Series.ndim** | It returns the number of dimensions in the data. |
| **Series.itemsize** | It returns the size of the datatype of item. |

### **Retrieving Index array and data array of a series object**

We can retrieve the index array and data array of an existing Series object by using the attributes index and values.

1. **import** numpy as np
2. **import** pandas as pd
3. x=pd.Series(data=[2,4,6,8])
4. y=pd.Series(data=[11.2,18.6,22.5], index=['a','b','c'])
5. print(x.index)
6. print(x.values)
7. print(y.index)
8. print(y.values)

**Output**

RangeIndex(start=0, stop=4, step=1)

[2 4 6 8]

Index(['a', 'b', 'c'], dtype='object')

[11.2 18.6 22.5]

### **Retrieving Types (dtype) and Size of Type (itemsize)**

You can use attribute dtype with Series object as <objectname> dtype for retrieving the data type of an individual element of a series object, you can use the **itemsize** attribute to show the number of bytes allocated to each data item.

1. **import** numpy as np
2. **import** pandas as pd
3. a=pd.Series(data=[1,2,3,4])
4. b=pd.Series(data=[4.9,8.2,5.6],
5. index=['x','y','z'])
6. print(a.dtype)
7. print(a.itemsize)
8. print(b.dtype)
9. print(b.itemsize)

**Output**

int64

8

float64

8

**Tkinter**

Python provides various options for developing graphical user interfaces (GUIs). The most important features are listed below.

* **Tkinter** − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look at this option in this chapter.
* **wxPython** − This is an open-source Python interface for wxWidgets GUI toolkit. You can find a complete tutorial on WxPython [here](https://www.tutorialspoint.com/wxpython/index.htm).
* **PyQt** − This is also a Python interface for a popular cross-platform Qt GUI library. TutorialsPoint has a very good tutorial on PyQt5 [here](https://www.tutorialspoint.com/pyqt/index.htm).
* **PyGTK** − PyGTK is a set of wrappers written in Python and C for GTK + GUI library. The complete PyGTK tutorial is available [here](https://www.tutorialspoint.com/pygtk/index.htm).
* **PySimpleGUI** − PySimpleGui is an open source, cross-platform GUI library for Python. It aims to provide a uniform API for creating desktop GUIs based on Python's Tkinter, PySide and WxPython toolkits. For a detaile PySimpleGUI tutorial, click [here](https://www.tutorialspoint.com/pysimplegui/index.htm).
* **Pygame** − Pygame is a popular Python library used for developing video games. It is free, open source and cross-platform wrapper around Simple DirectMedia Library (SDL). For a comprehensive tutorial on Pygame, [visit](https://www.tutorialspoint.com/pygame/index.htm) this link.
* **Jython** − Jython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machinehttp: [//www.jython.org](http://www.jython.org/).

There are many other interfaces available, which you can find them on the net.

## Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

The tkinter package includes following modules −

* **Tkinter** − Main Tkinter module.
* **tkinter.colorchooser** − Dialog to let the user choose a color.
* **tkinter.commondialog** − Base class for the dialogs defined in the other modules listed here.
* **tkinter.filedialog** − Common dialogs to allow the user to specify a file to open or save.
* **tkinter.font** − Utilities to help work with fonts.
* **tkinter.messagebox** − Access to standard Tk dialog boxes.
* **tkinter.scrolledtext** − Text widget with a vertical scroll bar built in.
* **tkinter.simpledialog** − Basic dialogs and convenience functions.
* **tkinter.ttk** − Themed widget set introduced in Tk 8.5, providing modern alternatives for many of the classic widgets in the main tkinter module.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps.

* Import the Tkinter module.
* Create the GUI application main window.
* Add one or more of the above-mentioned widgets to the GUI application.
* Enter the main event loop to take action against each event triggered by the user.

### **Example**

# note that module name has changed from Tkinter in Python 2

# to tkinter in Python 3

import tkinter

top = tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()

This would create a following window −



When the program becomes more complex, using an object-oriented programming approach makes the code more organized.

import tkinter as tk

class App(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

app = App()

app.mainloop()

ADVERTISEMENT

## Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | [**Button**](https://www.tutorialspoint.com/python/tk_button.htm)  The Button widget is used to display the buttons in your application. |
| 2 | [**Canvas**](https://www.tutorialspoint.com/python/tk_canvas.htm)  The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application. |
| 3 | [**Checkbutton**](https://www.tutorialspoint.com/python/tk_checkbutton.htm)  The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time. |
| 4 | [**Entry**](https://www.tutorialspoint.com/python/tk_entry.htm)  The Entry widget is used to display a single-line text field for accepting values from a user. |
| 5 | [**Frame**](https://www.tutorialspoint.com/python/tk_frame.htm)  The Frame widget is used as a container widget to organize other widgets. |
| 6 | [**Label**](https://www.tutorialspoint.com/python/tk_label.htm)  The Label widget is used to provide a single-line caption for other widgets. It can also contain images. |
| 7 | [**Listbox**](https://www.tutorialspoint.com/python/tk_listbox.htm)  The Listbox widget is used to provide a list of options to a user. |
| 8 | [**Menubutton**](https://www.tutorialspoint.com/python/tk_menubutton.htm)  The Menubutton widget is used to display menus in your application. |
| 9 | [**Menu**](https://www.tutorialspoint.com/python/tk_menu.htm)  The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton. |
| 10 | [**Message**](https://www.tutorialspoint.com/python/tk_message.htm)  The Message widget is used to display multiline text fields for accepting values from a user. |
| 11 | [**Radiobutton**](https://www.tutorialspoint.com/python/tk_radiobutton.htm)  The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time. |
| 12 | [**Scale**](https://www.tutorialspoint.com/python/tk_scale.htm)  The Scale widget is used to provide a slider widget. |
| 13 | [**Scrollbar**](https://www.tutorialspoint.com/python/tk_scrollbar.htm)  The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes. |
| 14 | [**Text**](https://www.tutorialspoint.com/python/tk_text.htm)  The Text widget is used to display text in multiple lines. |
| 15 | [**Toplevel**](https://www.tutorialspoint.com/python/tk_toplevel.htm)  The Toplevel widget is used to provide a separate window container. |
| 16 | [**Spinbox**](https://www.tutorialspoint.com/python/tk_spinbox.htm)  The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values. |
| 17 | [**PanedWindow**](https://www.tutorialspoint.com/python/tk_panedwindow.htm)  A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically. |
| 18 | [**LabelFrame**](https://www.tutorialspoint.com/python/tk_labelframe.htm)  A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. |
| 19 | [**tkMessageBox**](https://www.tutorialspoint.com/python/tk_messagebox.htm)  This module is used to display message boxes in your applications. |

Let us study these widgets in detail.

## Standard Attributes

Let us look at how some of the common attributes, such as sizes, colors and fonts are specified.

* [Dimensions](https://www.tutorialspoint.com/python/tk_dimensions.htm)
* [Colors](https://www.tutorialspoint.com/python/tk_colors.htm)
* [Fonts](https://www.tutorialspoint.com/python/tk_fonts.htm)
* [Anchors](https://www.tutorialspoint.com/python/tk_anchors.htm)
* [Relief styles](https://www.tutorialspoint.com/python/tk_relief.htm)
* [Bitmaps](https://www.tutorialspoint.com/python/tk_bitmaps.htm)
* [Cursors](https://www.tutorialspoint.com/python/tk_cursors.htm)

Let us study them briefly −

ADVERTISEMENT

## Geometry Management

All Tkinter widgets have access to the specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

* [**The pack() Method**](https://www.tutorialspoint.com/python/tk_pack.htm) − This geometry manager organizes widgets in blocks before placing them in the parent widget.
* [**The grid() Method**](https://www.tutorialspoint.com/python/tk_grid.htm) − This geometry manager organizes widgets in a table-like structure in the parent widget.
* [**The place() Method**](https://www.tutorialspoint.com/python/tk_place.htm) − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Let us study the geometry management methods briefly −

## SimpleDialog

The simpledialog module in tkinter package includes a dialog class and convenience functions for accepting user input through a modal dialog. It consists of a label, an entry widget and two buttons Ok and Cancel. These functions are −

* **askfloat(title, prompt, \*\*kw)** − Accepts a floating point number.
* **askinteger(title, prompt, \*\*kw)** − Accepts an integer input.
* **askstring(title, prompt, \*\*kw)** − Accepts a text input from the user.

The above three functions provide dialogs that prompt the user to enter a value of the desired type. If Ok is pressed, the input is returned, if Cancel is pressed, None is returned.

### **askinteger**

from tkinter.simpledialog import askinteger

from tkinter import \*

from tkinter import messagebox

top = Tk()

top.geometry("100x100")

def show():

num = askinteger("Input", "Input an Integer")

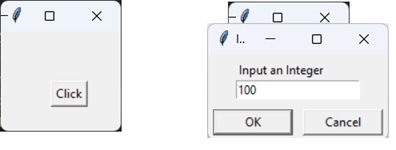
print(num)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following **output** −



### **askfloat**

from tkinter.simpledialog import askfloat

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

num = askfloat("Input", "Input a floating point number")

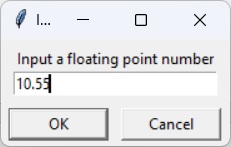
print(num)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following **output** −



### **askstring**

from tkinter.simpledialog import askstring

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

name = askstring("Input", "Enter you name")

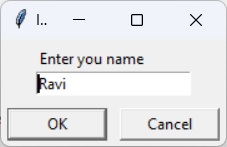
print(name)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following **output** −



ADVERTISEMENT

## The FileDialog Module

The filedialog module in Tkinter package includes a FileDialog class. It also defines convenience functions that enable the user to perform open file, save file, and open directory activities.

* filedialog.asksaveasfilename()
* filedialog.asksaveasfile()
* filedialog.askopenfilename()
* filedialog.askopenfile()
* filedialog.askdirectory()
* filedialog.askopenfilenames()
* filedialog.askopenfiles()

### **askopenfile**

This function lets the user choose a desired file from the filesystem. The file dialog window has Open and Cancel buttons. The file name along with its path is returned when Ok is pressed, None if Cancel is pressed.

from tkinter.filedialog import askopenfile

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

filename = askopenfile()

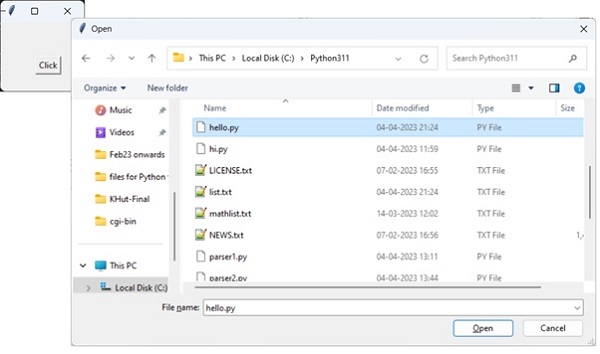
print(filename)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following **output** −



### **ColorChooser**

The colorchooser module included in tkinter package has the feature of letting the user choose a desired color object through the color dialog. The askcolor() function presents with the color dialog with predefined color swatches and facility to choose custome color by setting RGB values. The dialog returns a tuple of RGB values of chosen color as well as its hex value.

from tkinter.colorchooser import askcolor

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

color = askcolor()

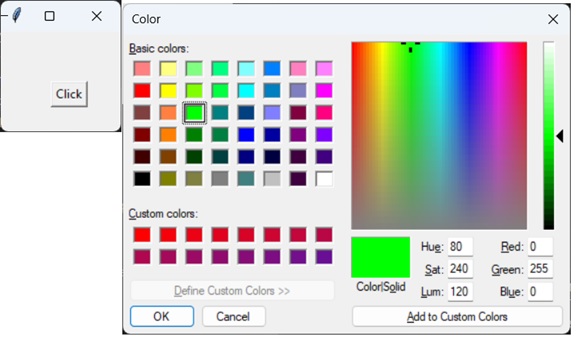
print(color)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following **output** −



((0, 255, 0), '#00ff00')

## ttk module

The term ttk stands from Tk Themed widgets. The ttk module was introduced with Tk 8.5 onwards. It provides additional benefits including anti-aliased font rendering under X11 and window transparency. It provides theming and styling support for Tkinter.

The ttk module comes bundled with 18 widgets, out of which 12 are already present in Tkinter. Importing ttk over-writes these widgets with new ones which are designed to have a better and more modern look across all platforms.

The 6 new widgets in ttk are, the Combobox, Separator, Sizegrip, Treeview, Notebook and ProgressBar.

To override the basic Tk widgets, the import should follow the Tk import −

from tkinter import \*

from tkinter.ttk import \*

The original Tk widgets are automatically replaced by tkinter.ttk widgets. They are Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale and Scrollbar.

New widgets which gives a better look and feel across platforms; however, the replacement widgets are not completely compatible. The main difference is that widget options such as "fg", "bg" and others related to widget styling are no longer present in Ttk widgets. Instead, use the ttk.Style class for improved styling effects.

The new widgets in ttk module are −

* **Notebook** − This widget manages a collection of "tabs" between which you can swap, changing the currently displayed window.
* **ProgressBar** − This widget is used to show progress or the loading process through the use of animations.
* **Separator** − Used to separate different widgets using a separator line.
* **Treeview** − This widget is used to group together items in a tree-like hierarchy. Each item has a textual label, an optional image, and an optional list of data values.
* **ComboBox** − Used to create a dropdown list of options from which the user can select one.
* **Sizegrip** − Creates a little handle near the bottom-right of the screen, which can be used to resize the window.

ADVERTISEMENT

## Combobox Widget

The Python ttk Combobox presents a drop down list of options and displays them one at a time. It is a sub class of the widget Entry. Hence it inherits many options and methods from the Entry class.

### **Syntax**

from tkinter import ttk

Combo = ttk.Combobox(master, values.......)

The get() function to retrieve the current value of the Combobox.

### **Example**

from tkinter import \*

from tkinter import ttk

top = Tk()

top.geometry("200x150")

frame = Frame(top)

frame.pack()

langs = ["C", "C++", "Java",

"Python", "PHP"]

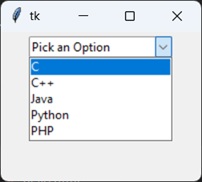
Combo = ttk.Combobox(frame, values = langs)

Combo.set("Pick an Option")

Combo.pack(padx = 5, pady = 5)

top.mainloop()

It will produce the following **output** −



## Progressbar

The ttk ProgressBar widget, and how it can be used to create loading screens or show the progress of a current task.

### **Syntax**

ttk.Progressbar(parent, orient, length, mode)

### **Parameters**

* **Parent** − The container in which the ProgressBar is to be placed, such as root or a Tkinter frame.
* **Orient** − Defines the orientation of the ProgressBar, which can be either vertical of horizontal.
* **Length** − Defines the width of the ProgressBar by taking in an integer value.
* **Mode** − There are two options for this parameter, determinate and indeterminate.