[Home](https://www.mygreatlearning.com/blog/" \o ")  [Machine Learning](https://www.mygreatlearning.com/blog/artificial-intelligence/machine-learning/)  [Learn Python](https://www.mygreatlearning.com/blog/artificial-intelligence/machine-learning/learnpython/)  Matplotlib in Python: Data Visualization Plots & how to use it

* [Machine Learning](https://www.mygreatlearning.com/blog/artificial-intelligence/machine-learning/)
* [Learn Python](https://www.mygreatlearning.com/blog/artificial-intelligence/machine-learning/learnpython/)

Matplotlib in Python: Data Visualization Plots & how to use it

By

[**Great Learning Team**](https://www.mygreatlearning.com/blog/author/greatlearning/)

 -

Aug 5, 2021

5205

[0](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#respond)

**Share**

1. [Introduction to Matplotlib](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#introduction)
2. [Installing Matplotlib](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#install)
3. [How to use Matplotlib](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#howtouse)
4. [The Relation between – Matplotlib, Pyplot and Python](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#relationship)
5. [Create a simple plot](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#simpleplot)
6. [Adding elements to a plot](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#addelements)
7. [Making Multiple Plots in One Figure](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#multipleplots)
8. [Create Subplots](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#subplots)
9. [Figure Object](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#figureobject)
10. [Axes Object](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#axesobject)
11. [Different Types of Plots](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#plottypes)
12. [Saving Plot](https://www.mygreatlearning.com/blog/matplotlib-tutorial-for-data-visualisation/#saveplot)

***Contributed by: Mr. Sridhar Anchoori  
LinkedIn profile:***[***https://www.linkedin.com/in/sridhar-anchoori-42156722/***](https://www.linkedin.com/in/sridhar-anchoori-42156722/)

*‘A Picture is Worth more than a thousand words’, similarly in the context of data ‘A visualisation is worth more than a complex data table or report’.*

Data Visualisation is one of the critical skills expected from data scientists. Most of the business problems could be understood and addressed using visualisation techniques. Visualisation basically involves Exploratory Data Analysis (EDA) and Graphical Plots. Effective visualisation helps the users to understand the patterns from the data and solve the business problem more effectively. Another advantage of visualisation is to simplify the complex data into an understandable format.

People find it very easy to read an image much easier than text. Visualisation is the best communication platform to analyse and interpret the data. It helps the users to understand vast amounts of information easily. Data visualisation helps in understanding the trends, correlation, patterns, distributions etc.,

There are multiple tools and technologies available in the industry for data visualisation, python being the most used. [Python](https://www.mygreatlearning.com/blog/python-tutorial-for-beginners-a-complete-guide/)offers multiple libraries for data visualisation, few of the popular graphic libraries are:

* **Matplotlib**
* **Seaborn**
* **Pandas visualisation**
* **Plotly**

This document helps in understanding the matplotlib library which is widely used in the industry. Matplotlib has a variety of graphical features and is very easy to understand. This article focuses on different graphical features including syntax.

**Installing Matplotlib**

There are multiple ways to install the matplotlib library. The easiest way to install matplotlib is to download the Anaconda package. Matplotlib is default installed with Anaconda package and does not require any additional steps.

* Download anaconda package from the official site of Anaconda
* To install matplotlib, go to anaconda prompt and run the following command

pip install matplotlib

or

conda install matplotlib

* Verify whether the matplotlib is properly installed using the following command in Jupyter notebook

import matplotlib

matplotlib.\_\_version\_\_

**How to use Matplotlib**

Before using matplotlib, we need to import the package. This can be done using the ‘import’ method in Jupyter notebook. PyPlot is the graphical module in matplotlib which is mostly used for data visualisation, importing PyPlot is sufficient to work around data visualisation.

# import matplotlib library as mpl

import matplotlib as mpl

#import the pyplot module from matplotlib as plt (short name used for referring the object)

import matplotlib.pyplot as plt

**The relation between – Matplotlib, Pyplot and Python**

* Python Is a very popular programming language, used for web development, mathematics and statistical analysis. Python works on most of the platforms and is also simple to use.
* Python has multiple libraries used for specific purposes, below libraries are mostly used for visualisation and data analysis.
  + NumPy
  + Pandas
  + Matplotlib
  + Seaborn
  + Plotly
  + SciKit-Learn
* As you observe one of the packages is matplotlib which is developed using python. This library is very widely used for data visualisations.
* PyPlot is a module in matplotlib which provides MATLAB like interface. MATLAB is heavily used for statistical analysis in the manufacturing industry. MATLAB is a licensed software and requires a significant amount of money to buy and use, whereas PyPlot is an open-source module and gives similar functionality as MATLAB using python. Just to conclude PyPlot has been seen as a replacement of MATLAB in the context of open source.

**Create a Simple Plot**

Here we will be depicting a basic plot using some random numbers generated using NumPy. The simplest way to create a graph is using the ‘plot()’ method. To generate a basic plot, we need two axes (X) and (Y), and we will generate two random numbers using the ‘linspace()’ method from Numpy.

# import the NumPy package

import numpy as np

# generate random number using NumPy, generate two sets of random numbers and store in x, y

x = np.linspace(0,50,100)

y = x \* np.linspace(100,150,100)

# Create a basic plot

plt.plot(x,y)

# Basic plot is generated as shown below:

**Adding Elements to Plot**

The plot generated above does not have all the elements to understand it better. Let’s try to add different elements for the plot for better interpretation. The elements that could be added for the plot includes title, x-Label, y-label, x-limits, y-limits.

# set different elements to the plot generated above

# Add title using ‘plt.title’

# Add x-label using ‘plt.xlabel’

# Add y-label using ‘plt.ylabel’

# set x-axis limits using ‘plt.xlim’

# set y-axis limits using ‘plt.ylim’

# Add legend using ‘plt.legend’

# Refer chart below, that has the elements added i.e., title, x-label, y-label, x-limits and y-limits

Let’s, add few more elements to the plot like colour, markers, line customisation.

# add color, style, width to line element

plt.plot(x, y, c = 'r', linestyle = '--', linewidth=2)

# add markers to the plot, marker has different elements i.e., style, color, size etc.,

plt.plot (x, y, marker='\*', markersize=3, c=’g’)

# add grid using grid() method

Plt.grid(True)

# add legend and label

plt.legend()

Plots could be customised at three levels:

* Colours
  + b – blue
  + c – cyan
  + g – green
  + k – black
  + m – magenta
  + r – red
  + w – white
  + y – yellow
  + Can use Hexadecimal, RGB formats
* Line Styles
  + ‘-‘ : solid line
  + ‘- -‘: dotted line
  + ‘- .’: dash-dot line
  + ‘:’ – dotted line
* Marker Styles
  + .  – point marker
  + ,  – Pixel marker
  + v – Triangle down marker
  + ^ – Triangle up marker
  + < – Triangle left marker
  + > – Triangle right marker
  + 1 – Tripod down marker
  + 2 – Tripod up marker
  + 3 – Tripod left marker
  + 4 – Tripod right marker
  + s – Square marker
  + p – Pentagon marker
  + \* – Star marker
* Other configurations
  + color or c
  + linestyle
  + linewidth
  + marker
  + markeredgewidth
  + markeredgecolor
  + markerfacecolor
  + markersize

**Making Multiple Plots in One Figure**

There could be some situations where the user may have to show multiple plots in a single figure for comparison purpose. For example, a retailer wants to know the sales trend of two stores for the last 12 months and he would like to see the trend of the two stores in the same figure.

Let’s plot two lines sin(x) and cos(x) in a single figure and add legend to understand which line is what.

# lets plot two lines Sin(x) and Cos(x)

# loc is used to set the location of the legend on the plot

# label is used to represent the label for the line in the legend

# generate the random number

x= np.arange(0,1500,100)

plt.plot(np.sin(x),label='sin function x')

plt.plot(np.cos(x),label='cos functon x')

plt.legend(loc='upper right')

# To show the multiple plots in separate figure instead of a single figure, use plt.show() statement before the next plot statement as shown below

x= np.linspace(0,100,50)

plt.plot(x,'r',label='simple x')

plt.show()

plt.plot(x\*x,'g',label='two times x')

plt.show()

plt.legend(loc='upper right')

**Create Subplots**

There could be some situations where we should show multiple plots in a single figure to show the complete storyline while presenting to stakeholders. This can be achieved with the use of subplot in matplotlib library. For example, a retail store has 6 stores and the manager would like to see the daily sales of all the 6 stores in a single window to compare. This can be visualised using subplots by representing the charts in rows and columns.

# subplots are used to create multiple plots in a single figure

# let’s create a single subplot first following by adding more subplots

x = np.random.rand(50)

y = np.sin(x\*2)

#need to create an empty figure with an axis as below, figure and axis are two separate objects in matplotlib

fig, ax = plt.subplots()

#add the charts to the plot

ax.plot(y)

# Let’s add multiple plots using subplots() function

# Give the required number of plots as an argument in subplots(), below function creates 2 subplots

fig, axs = plt.subplots(2)

#create data

x=np.linspace(0,100,10)

# assign the data to the plot using axs

axs[0].plot(x, np.sin(x\*\*2))

axs[1].plot(x, np.cos(x\*\*2))

# add a title to the subplot figure

fig.suptitle('Vertically stacked subplots')

# Create horizontal subplots

# Give two arguments rows and columns in the subplot() function

# subplot() gives two dimensional array with 2\*2 matrix

# need to provide ax also similar 2\*2 matrix as below

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

# add the data to the plots

ax1.plot(x, x\*\*2)

ax2.plot(x, x\*\*3)

ax3.plot(x, np.sin(x\*\*2))

ax4.plot(x, np.cos(x\*\*2))

# add title

fig.suptitle('Horizontal plots')

# another simple way of creating multiple subplots as below, using axs

fig, axs = plt.subplots(2, 2)

# add the data referring to row and column

axs[0,0].plot(x, x\*\*2,'g')

axs[0,1].plot(x, x\*\*3,'r')

axs[1,0].plot(x, np.sin(x\*\*2),'b')

axs[1,1].plot(x, np.cos(x\*\*2),'k')

# add title

fig.suptitle('matrix sub plots')

**Figure Object**

Matplotlib is an object-oriented library and has objects, calluses and methods. Figure is also one of the classes from the object ‘figure’. The object figure is a container for showing the plots and is instantiated by calling figure() function.

‘plt.figure()’ is used to create the empty figure object in matplotlib. Figure has the following additional parameters.

* Figsize – (width, height) in inches
* Dpi – used for dots per inch (this can be adjusted for print quality)
* facecolor
* edgecolor
* linewidth

# let’s create a figure object

# change the size of the figure is ‘figsize = (a,b)’ a is width and ‘b’ is height in inches

# create a figure object and name it as fig

fig = plt.figure(figsize=(4,3))

# create a sample data

X = np.array([1,2,3,4,5,6,8,9,10])

Y = X\*\*2

# plot the figure

plt.plot(X,Y)

# let’s change the figure size and also add additional parameters like facecolor, edgecolor, linewidth

fig = plt.figure(figsize=(10,3),facecolor='y',edgecolor='r',linewidth=5)

**Axes Object**

Axes is the region of the chart with data, we can add the axes to the figure using the ‘add\_axes()’ method.  This method requires the following four parameters i.e., left, bottom, width, and height

* Left – position of axes from left of figure
* bottom – position of axes from the bottom of figure
* width – width of the chart
* height – height of the chart

Other parameters that can be used for the axes object are:

* Set title using ‘ax.set\_title()’
* Set x-label using ‘ax.set\_xlabel()’
* Set y-label using ‘ax.set\_ylabel()’

# lets add axes using add\_axes() method

# create a sample data

y = [1, 5, 10, 15, 20,30]

x1 = [1, 10, 20, 30, 45, 55]

x2 = [1, 32, 45, 80, 90, 122]

# create the figure

fig = plt.figure()

# add the axes

ax = fig.add\_axes([0,0,2,1])

l1 = ax.plot(x1,y,'ys-')

l2 = ax.plot(x2,y,'go--')

# add additional parameters

ax.legend(labels = ('line 1', 'line 2'), loc = 'lower right')

ax.set\_title("usage of add axes function")

ax.set\_xlabel('x-axix')

ax.set\_ylabel('y-axis')

plt.show()

**Different Types of Matplotlib Plots**

Matplotlib has a wide variety of plot formats, few of them include bar chart, line chart, pie chart, scatter chart, bubble chart, waterfall chart, circular area chart, stacked bar chart etc., We will be going through most of these charts in this document with some examples. There are some elements that are common for each plot that can be customised like axis, color etc., and there could be some elements that are specific to the respective chart.

Bar Graph

**Overview:**

Bar graph represents the data using bars either in Horizontal or Vertical directions. Bar graphs are used to show two or more values and typically the x-axis should be categorical data. The length of the bar is proportional to the counts of the categorical variable on x-axis.

**Function:**

* The function used to show bar graph is ‘plt.bar()’
* The bar() function expects two lists of values one on x-coordinate and another on y-coordinate

**Customisations:**

plt.bar() function has the following specific arguments that can be used for configuring the plot.

* Width, Color, edge colour, line width, tick\_label, align, bottom,
* Error Bars – xerr, yerr

**Example:**

# lets create a simple bar chart

# x-axis is shows the subject and y -axis shows the markers in each subject

subject = ['maths','english','science','social','computer']

marks =[70,80,50,30,78]

plt.bar(subject,marks)

plt.show()

#let’s do some customizations

#width – shows the bar width and default value is 0.8

#color – shows the bar color

#bottom – value from where the y – axis starts in the chart i.e., the lowest value on y-axis shown

#align – to move the position of x-label, has two options ‘edge’ or ‘center’

#edgecolor – used to color the borders of the bar

#linewidth – used to adjust the width of the line around the bar

#tick\_label – to set the customized labels for the x-axis

plt.bar(subject,marks,color ='g',width = 0.5,bottom=10,align ='center',edgecolor='r',linewidth=2,tick\_label=subject)

# errors bars could be added to represent the error values referring to an array value

# here in this example we used standard deviation to show as error bars

plt.bar(subject,marks,color ='g',yerr=np.std(marks))

# to plot horizontal bar plot use plt.barh() function

plt.barh(subject,marks,color ='g',xerr=np.std(marks))

Pie Chart:

**Overview:**

Pie charts display the proportion of each value against the total sum of values. This chart requires a single series to display. The values on the pie chart shows the percentage contribution in terms of a pie called ***Wedge/Widget***. The angle of the wedge/widget is calculated based on the proportion of values. This visualisation is best when we are trying to compare different segments within the total values. For example, a sales manager wants to know the contribution of type of payments in a month i.e., paid through cash, credit card, debit card, PayPal, any other online apps.

**Function:**

* The function used for pie chart is ‘plt.pie()’
* To draw a pie chart, we need only one list of values, each wedge is calculated as proportion converted into angle.

**Customisations:**

plt.pie() function has the following specific arguments that can be used for configuring the plot.

* labels – used to show the widget categories
* explode – used to pull out the widget/wedge slice
* autopct – used to show the % of contributions for the widgets
* Set\_aspect – used to
* shadow – to show the shadow for a slice
* colours – to set the custom colours for the wedges
* startangle – to set the angles of the wedges

**Example:**

# Let’s create a simple pie plot

# Assume that we have a data on number of tickets resolved in a month

# the manager would like to know the individual contribution in terms of tickets closed in the week

# data

Tickets\_Closed = [10, 20, 8, 35, 30, 25]

Agents = ['Raj', 'Ramesh', 'Krishna', 'Arun', 'Virag', 'Mahesh']

# create pie chart

plt.pie(Tickets\_Closed, labels = Agents)

#Let’s add additional parameters to pie plot

#explode – to move one of the wedges of the plot

#autopct – to add the contribution %

explode = [0.2,0.1,0,0.1,0,0]

plt.pie(Tickets\_Closed, labels = Agents, explode=explode, autopct='%1.1f%%' )

Scatter Plot

**Overview:**

Scatterplot is used to visualise the relationship between two columns/series of data. The graph displays the collection of data points without connecting. The chart needs two variables, one variable shows X-position and the second variable shows Y-position. Scatterplot is used to represent the association between variables and mostly advised to use before regression. Scatterplot helps in understanding the following information across the two columns

* Any relationship exists between the two columns
* + ve Relationship
* Or -Ve relationship

**Function:**

* The function used for the scatter plot is ‘plt.scatter()’

**Customizations:**

plt.scatter() function has the following specific arguments that can be used for configuring the plot.

* size – to manage the size of the points
* color – to set the color of the points
* marker – type of marker
* alpha – transparency of point
* norm – to normalize the data (scaling between 0 to 1)

**Example:**

# let's create a simple scatter plot

# generate the data with random numbers

x = np.random.randn(1000)

y = np.random.randn(1000)

plt.scatter(x,y)

# as you observe there is no correlation exists between x and y

# let’s try to add additional parameters

# size – to manage the size of the points

#color – to set the color of the points

#marker – type of marker

#alpha – transparency of point

size = 150\*np.random.randn(1000)

colors = 100\*np.random.randn(1000)

plt.scatter(x, y, s=size, c = colors, marker ='\*', alpha=0.7)

Histogram

**Overview:**

Histogram is used to understand the distribution of the data. It is an estimate of the probability distribution of continuous data. It is similar to bar graph as discussed above but this is used to represent the distribution of a continuous variable whereas bar graph is used for discrete variable. Every distribution is characterised by four different elements including

* Center of the distribution
* Spread of the distribution
* Shape of the distribution
* Peak of the distribution

Histogram requires two elements x-axis shown using bins and y-axis shown with the frequency of the values in each of the bins form the data set. Every bin has a range with minimum and maximum values.

**Function:**

* The function used for scatter plot is ‘plt.hist()’

**Customisations:**

plt.hist() function has the following specific arguments that can be used for configuring the plot.

* bins – number of bins
* color
* edgecolor
* alpha – transparency of the color
* normed
* xlim – to set the x-limits
* ylim – to set the y-limits
* xticks, yticks
* facecolor, edgecolor, density

**Example:**

# let’s generate random numbers and use the random numbers to generate histogram

data = np.random.randn(1000)

plt.hist(data)

# let’s add additional parameters

# facecolor

# alpha

# edgecolor

# bins

data = np.random.randn(1000)

plt.hist(data, facecolor ='y',linewidth=2,edgecolor='k', bins=30, alpha=0.6)

# lets create multiple histograms in a single plot

# Create random data

hist1 = np.random.normal(25,10,1000)

hist2 = np.random.normal(200,5,1000)

#plot the histogram

plt.hist(hist1,facecolor = 'yellow',alpha = 0.5, edgecolor ='b',bins=50)

plt.hist(hist2,facecolor = 'orange',alpha = 0.8, edgecolor ='b',bins=30)

**Saving Plot**

Saving plot as an image using ‘savefig()’ function in matplotlib. The plot can be saved in multiple formats like .png, .jpeg, .pdf and many other supporting formats.

# let's create a figure and save it as image

items = [5,10,20,25,30,40]

x = np.arange(6)

fig = plt.figure()

ax = plt.subplot(111)

ax.plot(x, y, label='items')

plt.title('Saving as Image')

ax.legend()

fig.savefig('saveimage.png')

Image is saved with a filename as ‘saveimage.png’.

#To display the image again, use the following package and commands

import matplotlib.image as mpimg

image = mpimg.imread("saveimage.png")

plt.imshow(image)

plt.show()

**Matplotlib Tutorial FAQs**

**Q: What is Matplotlib used for?**

**A:** Matplotlib is a plotting library meant especially for Python programming language as well as for NumPy, which is its numerical mathematics extension. An object-oriented API is provided by it to embed plots into applications by using general-purpose GUI toolkits such as wxPython, Tkinter, Qt, or GTK.

**Q: How do I learn Matplotlib?**

**A:** There are several resources available on the internet to learn Matplotlib. However, if you want to choose any one, you can consider Great Learning. You can get a tutorial on Matplotlib and learn about it.

**Q: How do you plot a line in Python?**

**A:** You can use Matplotlib’s pyplot library to plot a line in Python. To plot a line, you first have to import Matplotib. After the import is done, including the line %matplotlib inline by using a Jupyter notebook.

**Q: When should I use Matplotlib?**

**A:**Matplotlib is used by some people interactively from the python shell, and the plotting windows pop up as they type commands. While Jupyter notebooks are run by some people to draw inline plots for a fast and quick data analysis. Some people embed Matplotlib into graphical user interfaces such as PyQt or PyGObject to create rich applications.

**Q: Is Matplotlib difficult to learn?**

**A:** To learn Matplotlib, you have to thoroughly know Python. It is not at all difficult to learn Matplotlib but may take around 5-10 weeks, including basic Python syntax, object-oriented programming, variables, data types, loops, and functions.

**Q: How do you install Matplotlib?**

**A:** Installation of Matplotlib depends on the distribution of Python that is installed in your computer. You can install Matplotlib with Anaconda Prompt and pip. To install with Anaconda Prompt, you have to open it, and type > conda install matplotlib. To install Matplotlib with pip, you need to open a terminal window and type $ pip install matplotlib.

**Q: What is the difference between Matplotlib and Seaborn?**

**A:** Matplotlib is a Python library that is used for plotting graphs through other libraries such as Numpy and Pandas.  Seaborn is also a Python library that is used for plotting graphs through Pandas, Matplotlib, and Numpy.

**Q: Is MATLAB a Matplotlib?**

**A:** Matplotlib is a Python visualization library; however, Matlab is a programming language as well as a numerical computing environment. Matplotlib can also be called a data plotting library in Python. MATLAB is a totally different programming language than Python.

This brings us to the end of this Matplotlib tutorial. If you wish to learn more about Python, upskill with [Great Learning’s PG program in Artificial Intelligence and Machine Learning.](https://www.greatlearning.in/pg-program-artificial-intelligence-course?Blog_ID=12715)