Unit-5: Data Visualization using dataframe:

5.1 importing matplotlib.pyplot and plotting: (only two dimensional Plots)

Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

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

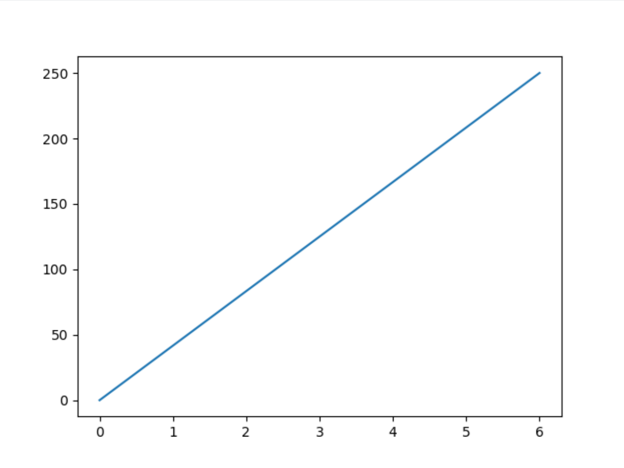
Now the Pyplot package can be referred to as plt.

Example

Draw a line in a diagram from position (0,0) to position (6,250):

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

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

Result:



Python range() Function

Example

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

```
x = range(6)
for n in x:
  print(n)
```

Definition and Usage

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax

```
range(start, stop, step)
```

Parameter Values

Parameter	**Description**
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

More Examples

Example

Create a sequence of numbers from 3 to 5, and print each item in the sequence:

```
x = range(3, 6)
for n in x:
  print(n)
```

Example

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
x = range(3, 20, 2)
for n in x:
  print(n)
```

Subplots() Function

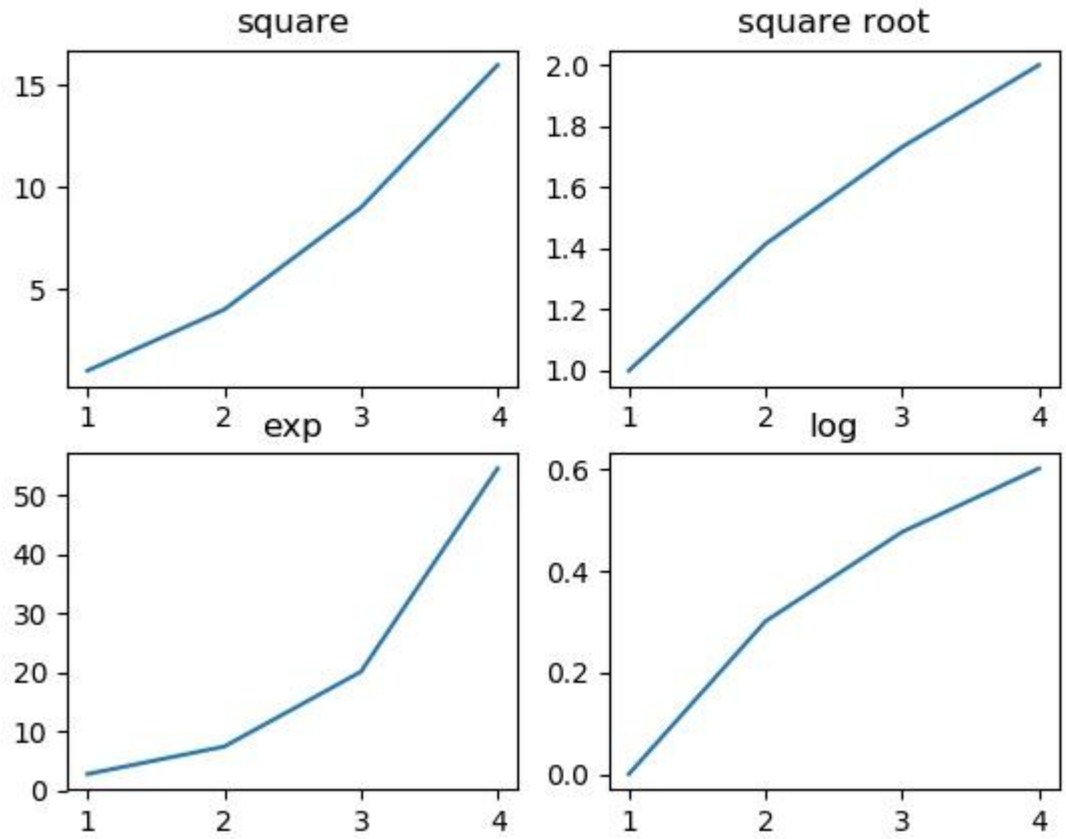
Matplotlib’spyplot API has a convenience function called subplots() which acts as a utility wrapper and helps in creating common layouts of subplots, including the enclosing figure object, in a single call.

```
Plt.subplots(nrows, ncols)
```

The two integer arguments to this function specify the number of rows and columns of the subplot grid. The function returns a figure object and a tuple containing axes objects equal to nrows*ncols. Each axes object is accessible by its index. Here we create a subplot of 2 rows by 2 columns and display 4 different plots in each subplot.

```
import matplotlib.pyplot as plt
fig,a =  plt.subplots(2,2)
import numpy as np
x = np.arange(1,5)
a[0][0].plot(x,x*x)
a[0][0].set_title('square')
a[0][1].plot(x,np.sqrt(x))
a[0][1].set_title('square root')
a[1][0].plot(x,np.exp(x))
a[1][0].set_title('exp')
a[1][1].plot(x,np.log10(x))
a[1][1].set_title('log')
plt.show()
```

The above line of code generates the following output −


square
15
10
5
1
2
3
4
square root
2.0
1.8
1.6
1.4
1.2
1.0
1
2
3
4
exp
50
40
30
20
10
0
1
2
3
4
log
0.6
0.4
0.2
0.0
1
2
3
4

Matplotlib.pyplot.legend() in Python

- Difficulty Level : Basic
- Last Updated : 12 Apr, 2020

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. **Pyplot** is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Matplotlib.pyplot.legend()

A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called **legend()** which is used to Place a legend on the axes.
The attribute **Loc** in `legend()` is used to specify the location of the legend.Default value of loc is loc="best" (upper left). The strings 'upper left', 'upper right', 'lower left', 'lower right' place the legend at the corresponding corner of the axes/figure.
The attribute **bbox_to_anchor=(x, y)** of legend() function is used to specify the coordinates of the legend, and the attribute **ncol** represents the number of columns that the legend has.It's default value is 1.
Syntax:
matplotlib.pyplot.legend(["blue", "green"], bbox_to_anchor=(0.75, 1.15), ncol=2)

The Following are some more attributes of function `legend()` :

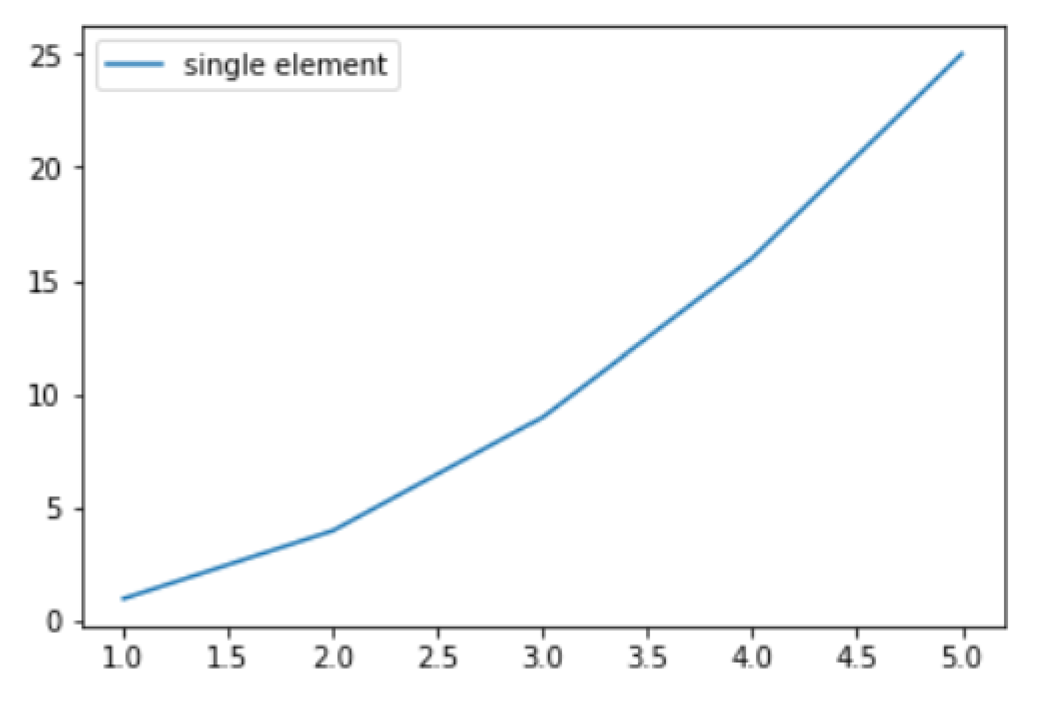
- **shadow**: [None or bool] Whether to draw a shadow behind the legend.It's Default value is None.
- **markerscale**: [None or int or float] The relative size of legend markers compared with the originally drawn ones.The Default is None.
- **numpoints**: [None or int] The number of marker points in the legend when creating a legend entry for a Line2D (line).The Default is None.
- **fontsize**: The font size of the legend.If the value is numeric the size will be the absolute font size in points.
- **facecolor**: [None or "inherit" or color] The legend's background color.
- **edgecolor**: [None or "inherit" or color] The legend's background patch edge color.

Ways to use legend() function in Python –
Example 1:

```
import numpy as np

import matplotlib.pyplot as plt



# X-axis values

x = [1, 2, 3, 4, 5]
```

```
# Y-axis values

y = [1, 4, 9, 16, 25]


# Function to plot

plt.plot(x, y)


# Function add a legend

plt.legend(['single element'])


# function to show the plot

plt.show()
```

Output :

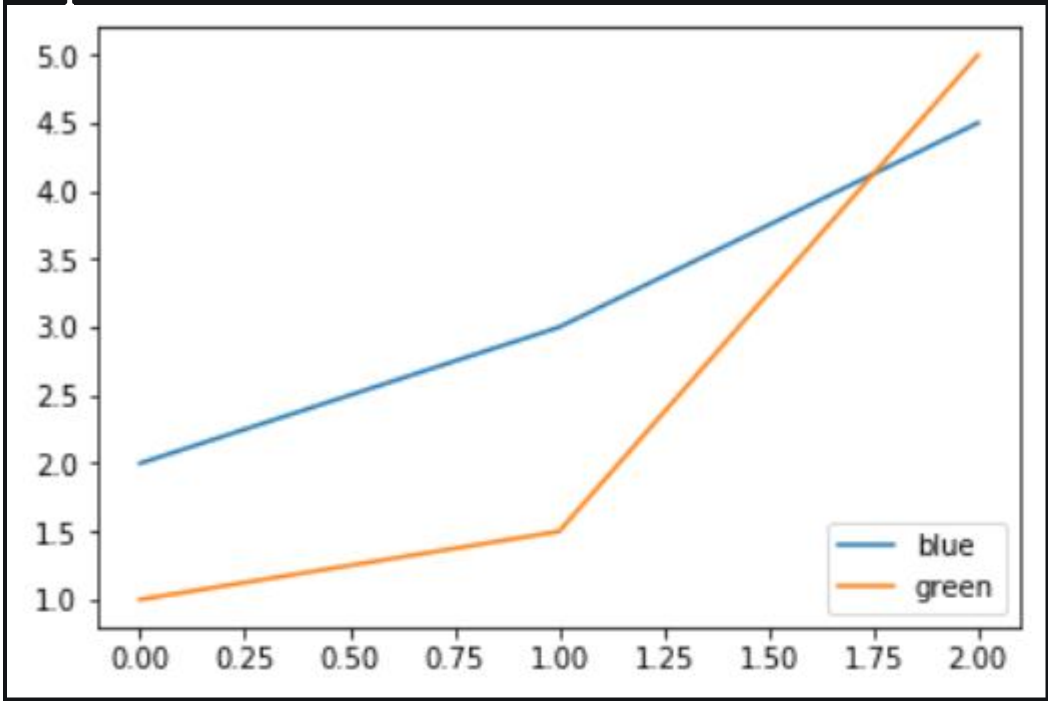


Example 2:

```
# importing modules

import numpy as np

import matplotlib.pyplot as plt

```

```
# Y-axis values

y1 = [2, 3, 4.5]


# Y-axis values

y2 = [1, 1.5, 5]


# Function to plot

plt.plot(y1)

plt.plot(y2)


# Function add a legend

plt.legend(["blue", "green"], loc ="lower right")


# function to show the plot

plt.show()
```

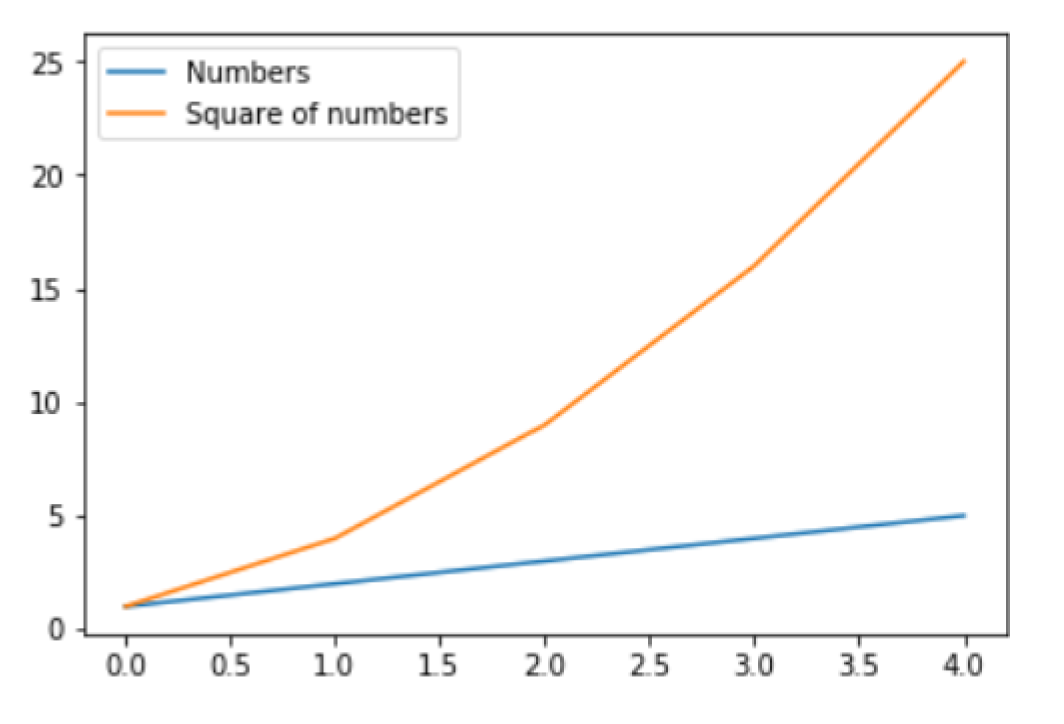
Output :



Example 3:

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

# X-axis values
x = np.arange(5)

# Y-axis values
y1 = [1, 2, 3, 4, 5]

# Y-axis values
y2 = [1, 4, 9, 16, 25]

# Function to plot
plt.plot(x, y1, label ='Numbers')
plt.plot(x, y2, label ='Square of numbers')

# Function add a legend
plt.legend()

# function to show the plot
plt.show()
```

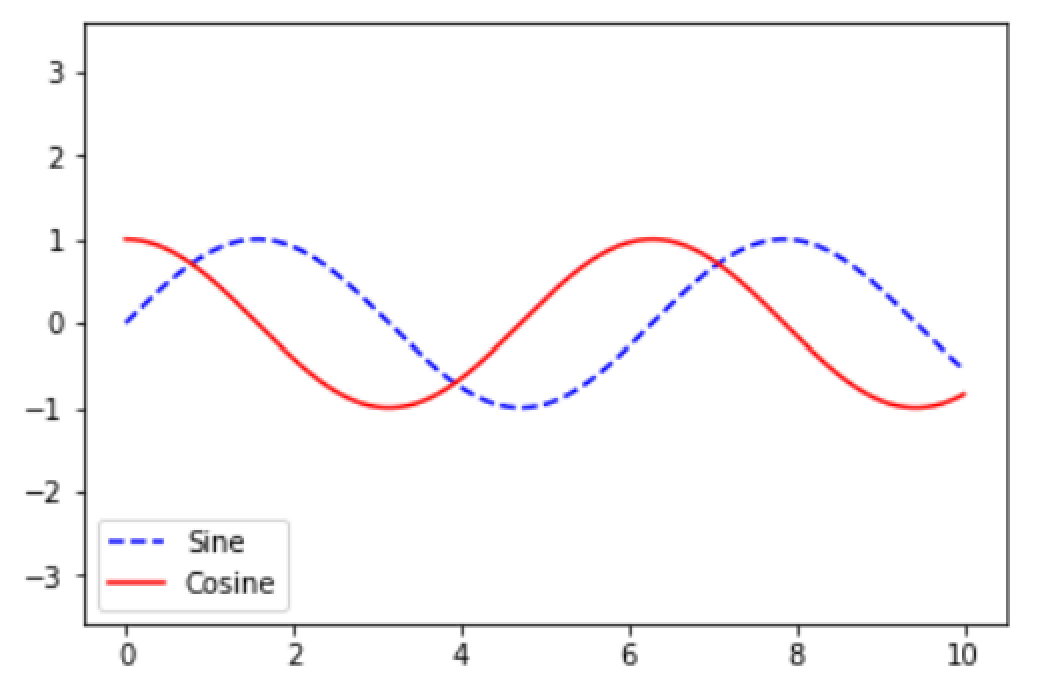
Output :



Example 4:

```
import numpy as np

import matplotlib.pyplot as plt


x = np.linspace(0, 10, 1000)

fig, ax = plt.subplots()


ax.plot(x, np.sin(x), '--b', label ='Sine')

ax.plot(x, np.cos(x), c ='r', label ='Cosine')

ax.axis('equal')


leg = ax.legend(loc ="lower left");
```

Output:

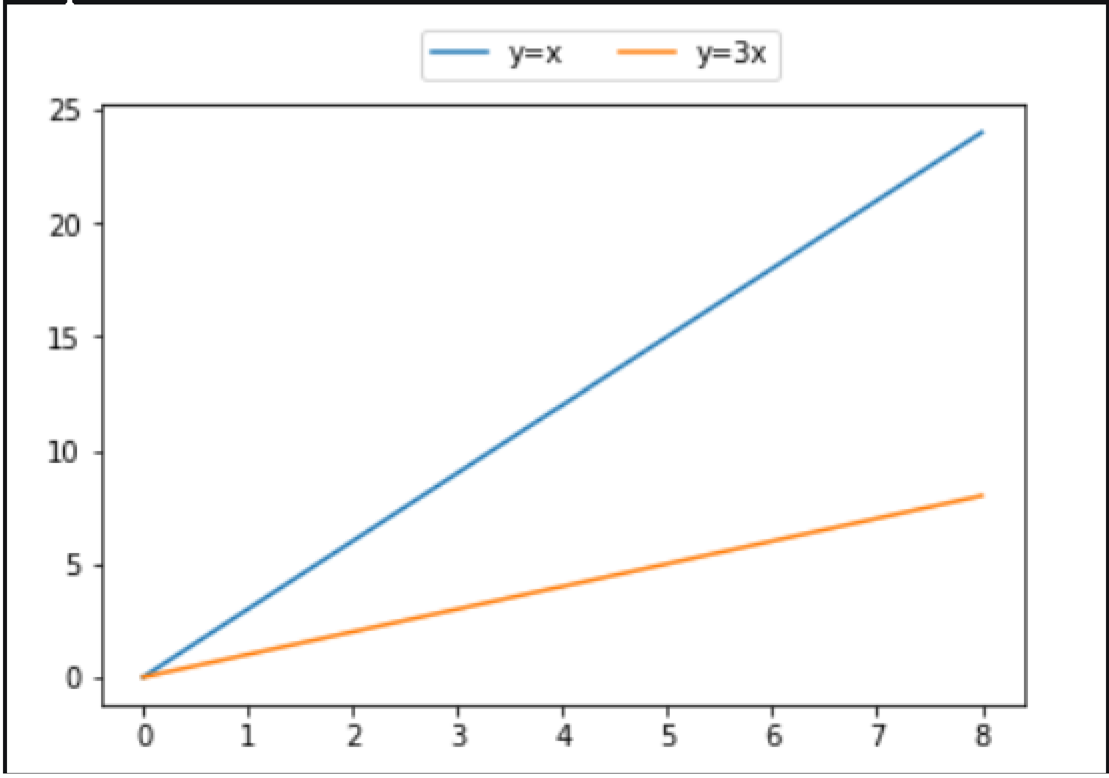


Example 5:

```
# importing modules

import numpy as np

import matplotlib.pyplot as plt


# X-axis values

x = [0, 1, 2, 3, 4, 5, 6, 7, 8]


# Y-axis values

y1 = [0, 3, 6, 9, 12, 15, 18, 21, 24]

# Y-axis values

y2 = [0, 1, 2, 3, 4, 5, 6, 7, 8]


# Function to plot

plt.plot(y1, label ="y = x")

plt.plot(y2, label ="y = 3x")


# Function add a legend
```

```
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2)


# function to show the plot

plt.show()
```

Output:



Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects. This is the primary data structure of the Pandas.

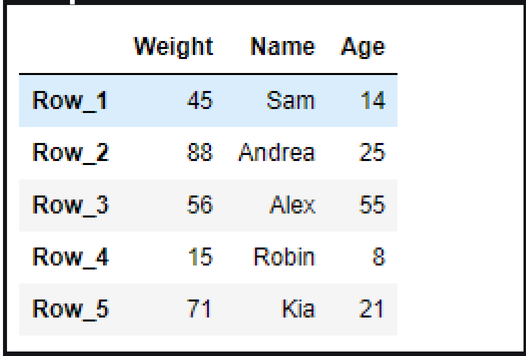
Pandas **DataFrame.columns** attribute return the column labels of the given Dataframe.

Syntax: *DataFrame.columns*
Parameter : *None*
Returns : *column names*

Example #1: Use `DataFrame.columns` attribute to return the column labels of the given Dataframe.

```
# importing pandas as pd
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({'Weight':[45, 88, 56, 15, 71],
                   'Name':['Sam', 'Andrea', 'Alex', 'Robin', 'Kia'],
                   'Age':[14, 25, 55, 8, 21]})

# Create the index
index_ = ['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5']

# Set the index
df.index = index_

# Print the DataFrame
print(df)
```

Output :

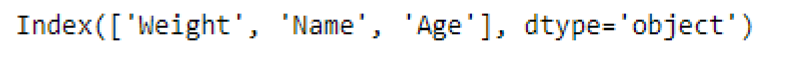


	Weight	Name	Age
Row_1	45	Sam	14
Row_2	88	Andrea	25
Row_3	56	Alex	55
Row_4	15	Robin	8
Row_5	71	Kia	21

Now we will use `DataFrame.columns` attribute to return the column labels of the given dataframe.

```
# return the column labels
result = df.columns

# Print the result
print(result)
```

Output :



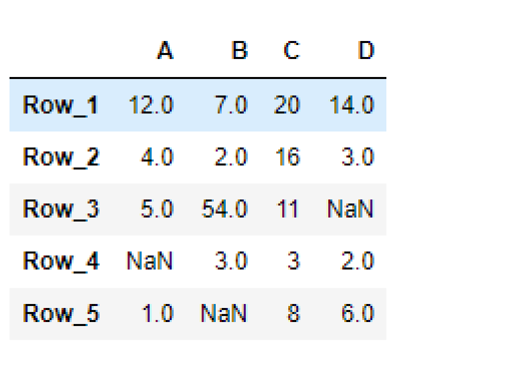
```
Index(['Weight', 'Name', 'Age'], dtype='object')
```

As we can see in the output, the `DataFrame.columns` attribute has successfully returned all of the column labels of the given dataframe.

Example #2: Use `DataFrame.columns` attribute to return the column labels of the given Dataframe.

```
# importing pandas as pd
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[7, 2, 54, 3, None],
                   "C":[20, 16, 11, 3, 8],
                   "D":[14, 3, None, 2, 6]})

# Create the index
index_ = ['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5']

# Set the index
df.index = index_

# Print the DataFrame
print(df)
```

Output :

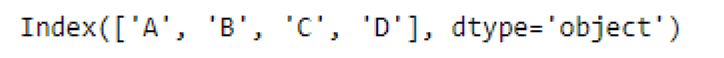


```
          A     B   C     D
Row_1  12.0   7.0  20  14.0
Row_2   4.0   2.0  16   3.0
Row_3   5.0  54.0  11   NaN
Row_4   NaN   3.0   3   2.0
Row_5   1.0   NaN   8   6.0
```

Now we will use `DataFrame.columns` attribute to return the column labels of the given dataframe.

```
# return the column labels
result = df.columns

# Print the result
print(result)
```

Output :



```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

As we can see in the output, the `DataFrame.columns` attribute has successfully returned all of the column labels of the given dataframe.

Python len() Function

Example

Return the number of items in a list:

```
mylist = ["apple", "banana", "cherry"]
x = len(mylist)
```

Definition and Usage

The `len()` function returns the number of items in an object.

When the object is a string, the `len()` function returns the number of characters in the string.

Syntax

```
len(object)
```

Parameter Values

Parameter	Description
object	Required. An object. Must be a sequence or a collection

More Examples

Example

Return the number of characters in a string:

```
mylist = "Hello"
x = len(mylist)
```

matplotlib.pyplot.scatter() in Python

- Last Updated : 15 Feb, 2022

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used for plotting various plots in Python like scatter plot, bar charts, pie charts, line plots, histograms, 3-D plots and many more. We will learn about the scatter plot from the matplotlib library.

Note: For more information, refer to Python Matplotlib – An Overview

matplotlib.pyplot.scatter()

Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

Syntax

The syntax for scatter() method is given below:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

The scatter() method takes in the following parameters:

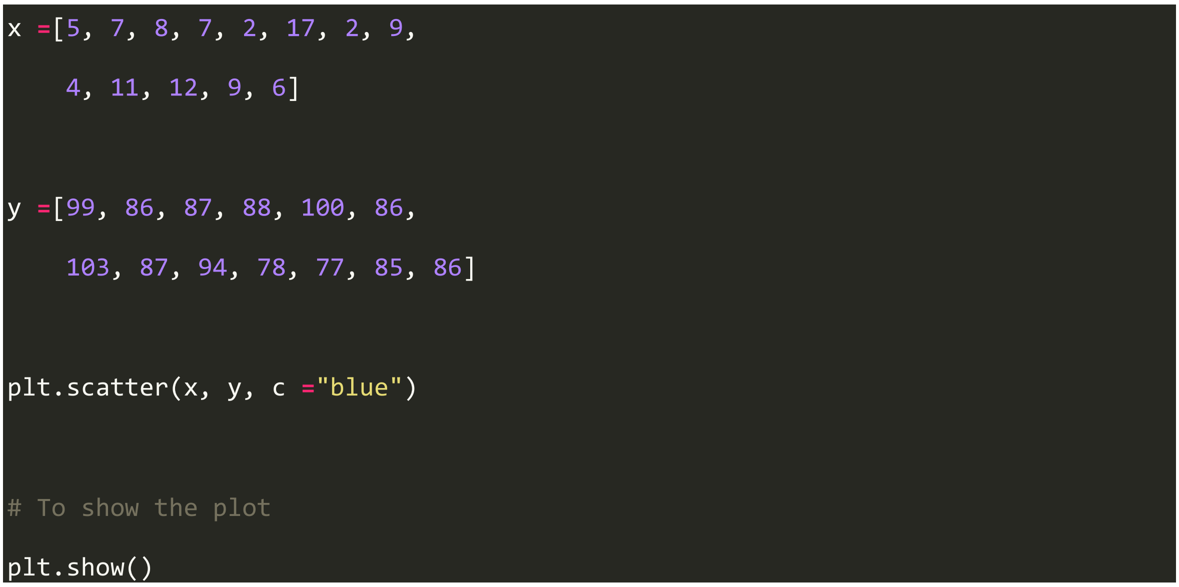
- **x_axis_data-** An array containing x-axis data
- **y_axis_data-** An array containing y-axis data
- **s-** marker size (can be scalar or array of size equal to size of x or y)
- **c-** color of sequence of colors for markers
- marker- marker style
- **cmap-** cmap name
- **linewidths-** width of marker border
- **edgecolor-** marker border color
- **alpha-** blending value, between 0 (transparent) and 1 (opaque)

Except x_axis_data and y_axis_data all other parameters are optional and their default value is None. Below are the scatter plot examples with various parameters.

Example 1: This is the most basic example of a scatter plot.

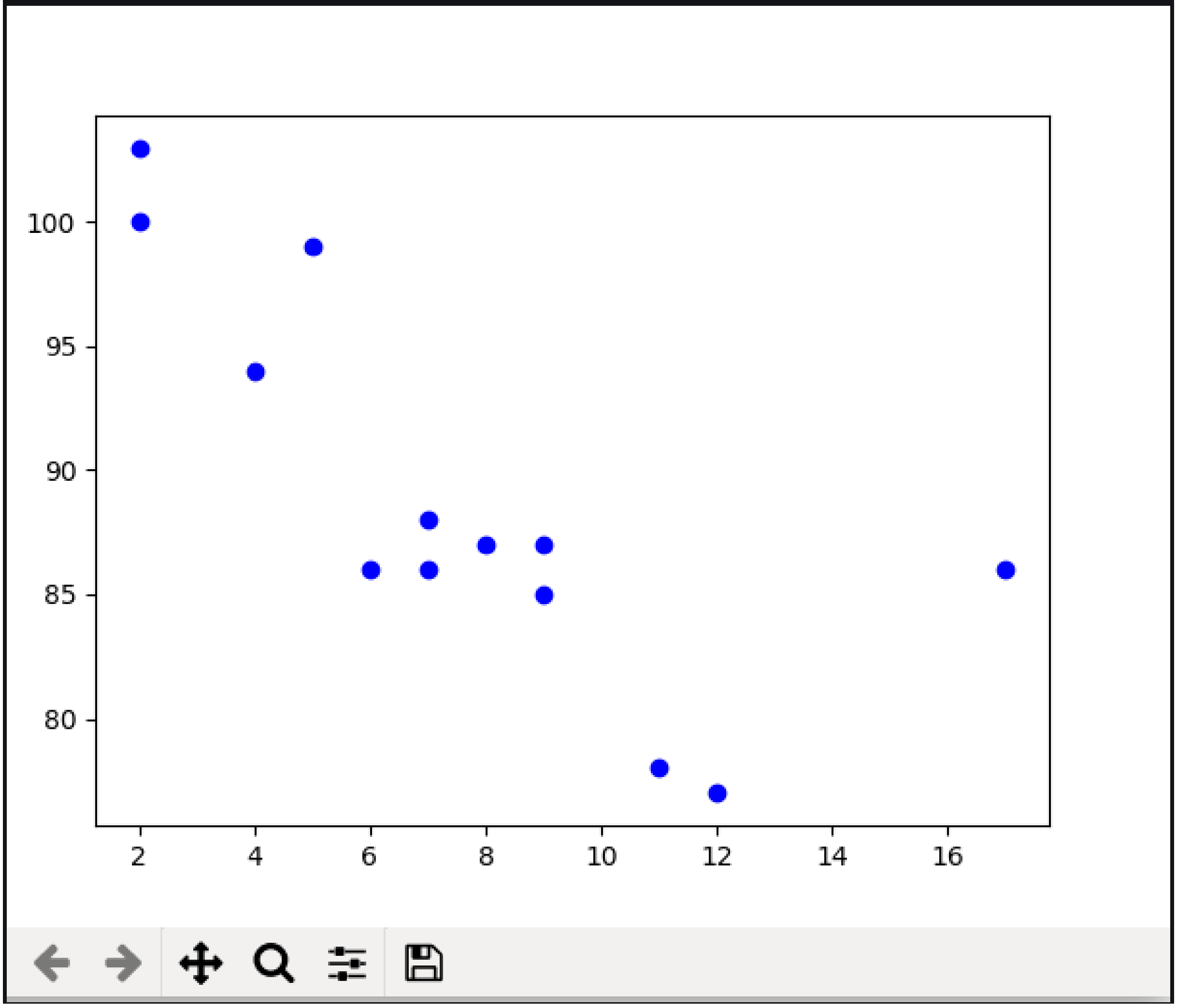
- Python3

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



```
x =[5, 7, 8, 7, 2, 17, 2, 9,

    4, 11, 12, 9, 6]


y =[99, 86, 87, 88, 100, 86,

    103, 87, 94, 78, 77, 85, 86]


plt.scatter(x, y, c ="blue")


# To show the plot

plt.show()
```

Output



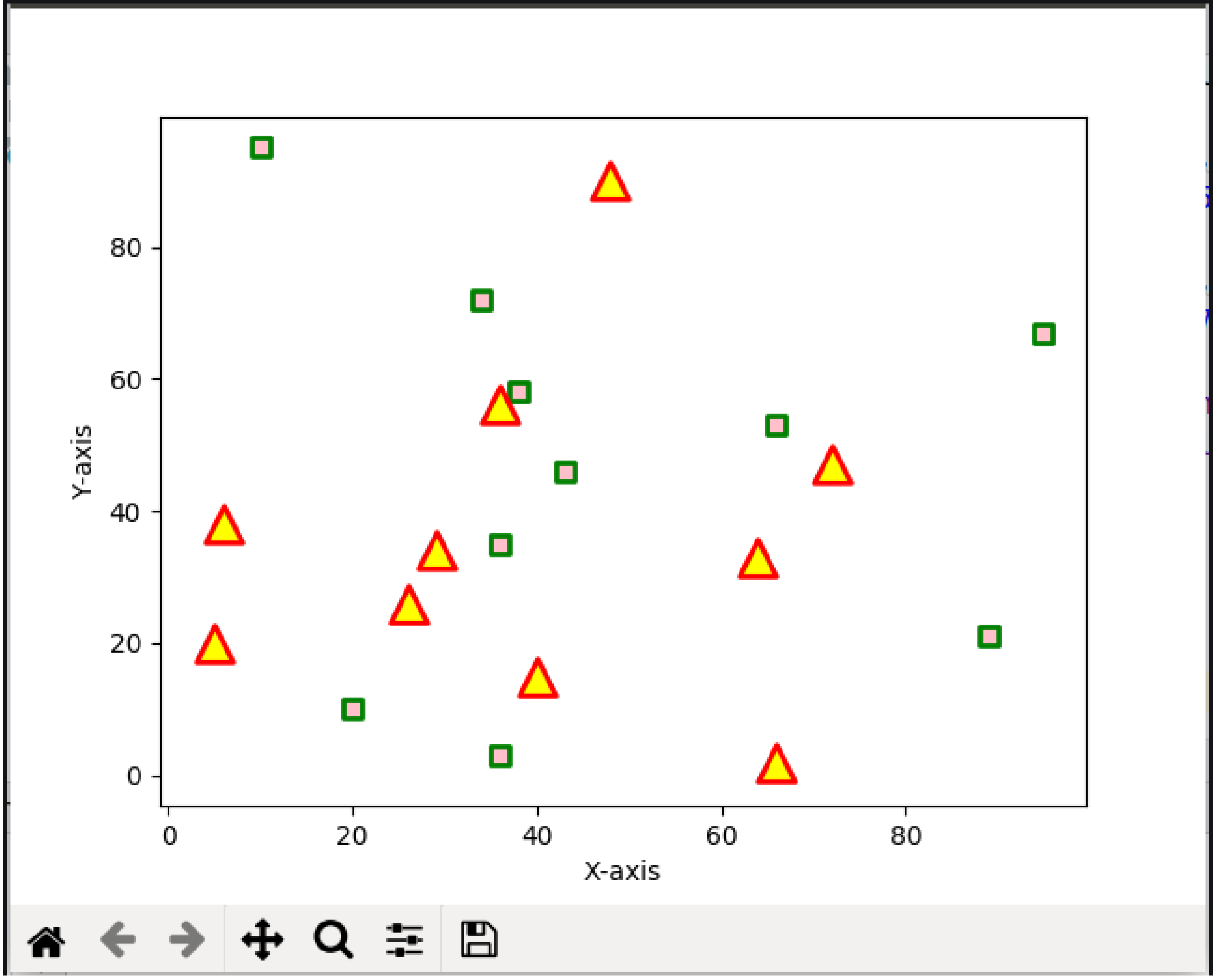
Example 2: Scatter plot with different shape and colour for two datasets.

- Python3

```
import matplotlib.pyplot as plt


# dataset-1

x1 = [89, 43, 36, 36, 95, 10,

      66, 34, 38, 20]


y1 = [21, 46, 3, 35, 67, 95,

      53, 72, 58, 10]


# dataset2

x2 = [26, 29, 48, 64, 6, 5,

      36, 66, 72, 40]


y2 = [26, 34, 90, 33, 38,

      20, 56, 2, 47, 15]


plt.scatter(x1, y1, c ="pink",

            linewidths = 2,

            marker ="s",

            edgecolor ="green",

            s = 50)


plt.scatter(x2, y2, c ="yellow",

            linewidths = 2,

            marker ="^",

            edgecolor ="red",

            s = 200)
```

```
plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()
```

Output



Line chart in Matplotlib – Python

- Difficulty Level : Medium
- Last Updated : 20 Oct, 2020

Matplotlib is a data visualization library in Python. The pyplot, a sublibrary of matplotlib, is a collection of functions that helps in creating a variety of charts. ***Line charts*** are used to represent the relation between two data X and Y on a different axis. Here we will see some of the examples of a line chart in Python :

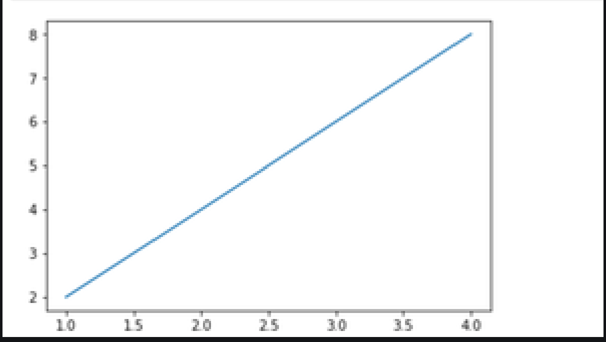
Simple line plots

First import Matplotlib.pyplot library for plotting functions. Also, import the Numpy library as per requirement. Then define data values x and y.

- Python3

```
# importing the required libraries

import matplotlib.pyplot as plt

import numpy as np


# define data values

x = np.array([1, 2, 3, 4])  # X-axis points

y = x*2  # Y-axis points


plt.plot(x, y)  # Plot the chart

plt.show()  # display
```

Output:



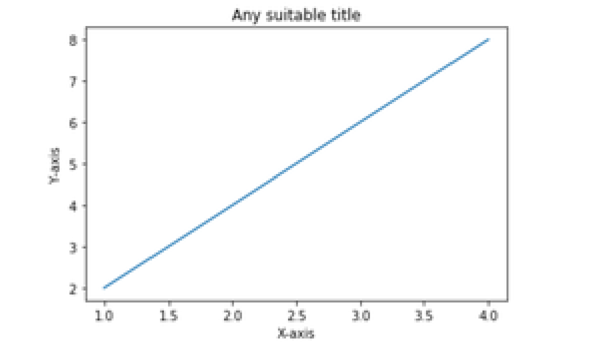
Simple line plot between X and Y data

we can see in the above output image that there is no label on the x-axis and y-axis. Since labeling is necessary for understanding the chart dimensions. In the following example, we will see how to add labels, Ident in the charts

- Python3

```
import matplotlib.pyplot as plt

import numpy as np



# Define X and Y variable data

x = np.array([1, 2, 3, 4])

y = x*2


plt.plot(x, y)

plt.xlabel("X-axis")  # add X-axis label

plt.ylabel("Y-axis")  # add Y-axis label

plt.title("Any suitable title")  # add title

plt.show()
```

Output:



Simple line plot with labels and title

Multiple charts

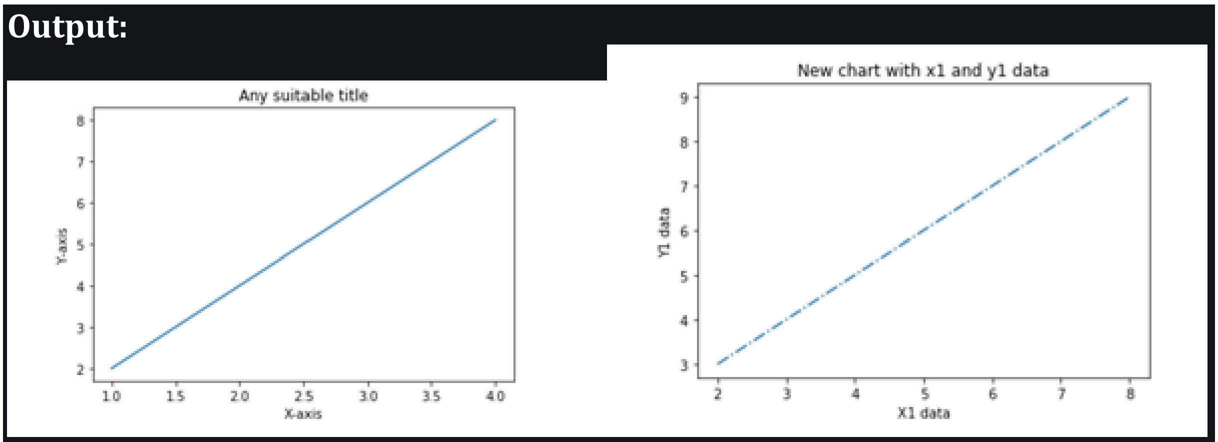
We can display more than one chart in the same container by using pyplot.figure() function. This will help us in comparing the different charts and also control the look and feel of charts .

- Python3

```
import matplotlib.pyplot as plt

import numpy as np
```

```
x = np.array([1, 2, 3, 4])
y = x*2

plt.plot(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Any suitable title")
plt.show()  # show first chart

# The figure() function helps in creating a
# new figure that can hold a new chart in it.
plt.figure()
x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]
plt.plot(x1, y1, '-.')

# Show another chart with '-' dotted line
plt.show()
```

Output:



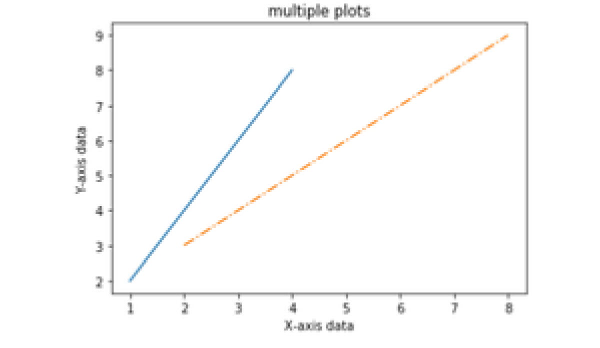
Multiple plots on the same axis

Here we will see how to add 2 plots within the same axis.

- Python3

```
import matplotlib.pyplot as plt

import numpy as np


x = np.array([1, 2, 3, 4])

y = x*2


# first plot with X and Y data

plt.plot(x, y)


x1 = [2, 4, 6, 8]

y1 = [3, 5, 7, 9]


# second plot with x1 and y1 data

plt.plot(x1, y1, '-.')


plt.xlabel("X-axis data")

plt.ylabel("Y-axis data")

plt.title('multiple plots')

plt.show()
```

Output:



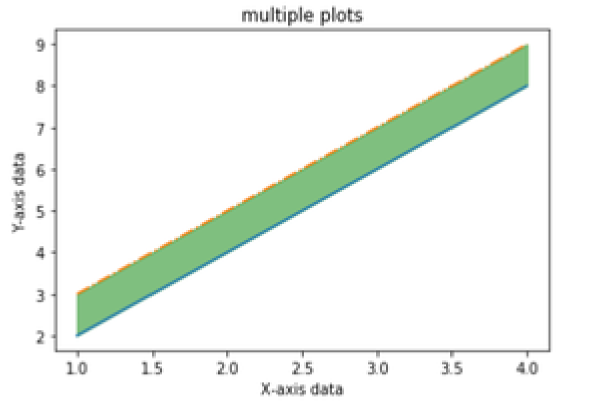
Fill the area between two plots

Using the pyplot.fill_between() function we can fill in the region between two line plots in the same graph. This will help us in understanding the margin of data between two line plots based on certain conditions.

- Python3

```
import matplotlib.pyplot as plt

import numpy as np


x = np.array([1, 2, 3, 4])

y = x*2


plt.plot(x, y)


x1 = [2, 4, 6, 8]

y1 = [3, 5, 7, 9]


plt.plot(x, y1, '-.')

plt.xlabel("X-axis data")

plt.ylabel("Y-axis data")

plt.title('multiple plots')


plt.fill_between(x, y, y1, color='green', alpha=0.5)

plt.show()
```

Output:



Plotting Histogram in Python using Matplotlib

- Difficulty Level : Easy
- Last Updated : 29 Jul, 2021

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution.It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

Creating a Histogram

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals.Bins are clearly identified as consecutive, non-overlapping intervals of variables.The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

The following table shows the parameters accepted by matplotlib.pyplot.hist() function :

Attribute	parameter
x	array or sequence of array
bins	optional parameter contains integer or sequence or strings
density	optional parameter contains boolean values
range	optional parameter represents upper and lower range of bins
histtype	optional parameter used to create type of histogram [bar, barstacked, step, stepfilled], default is “bar”
align	optional parameter controls the plotting of histogram [left, right, mid]
weights	optional parameter contains array of weights having same dimensions as x
bottom	location of the basline of each bin

Attribute	parameter
rwidth	optional parameter which is relative width of the bars with respect to bin width
color	optional parameter used to set color or sequence of color specs
label	optional parameter string or sequence of string to match with multiple datasets
log	optional parameter used to set histogram axis on log scale

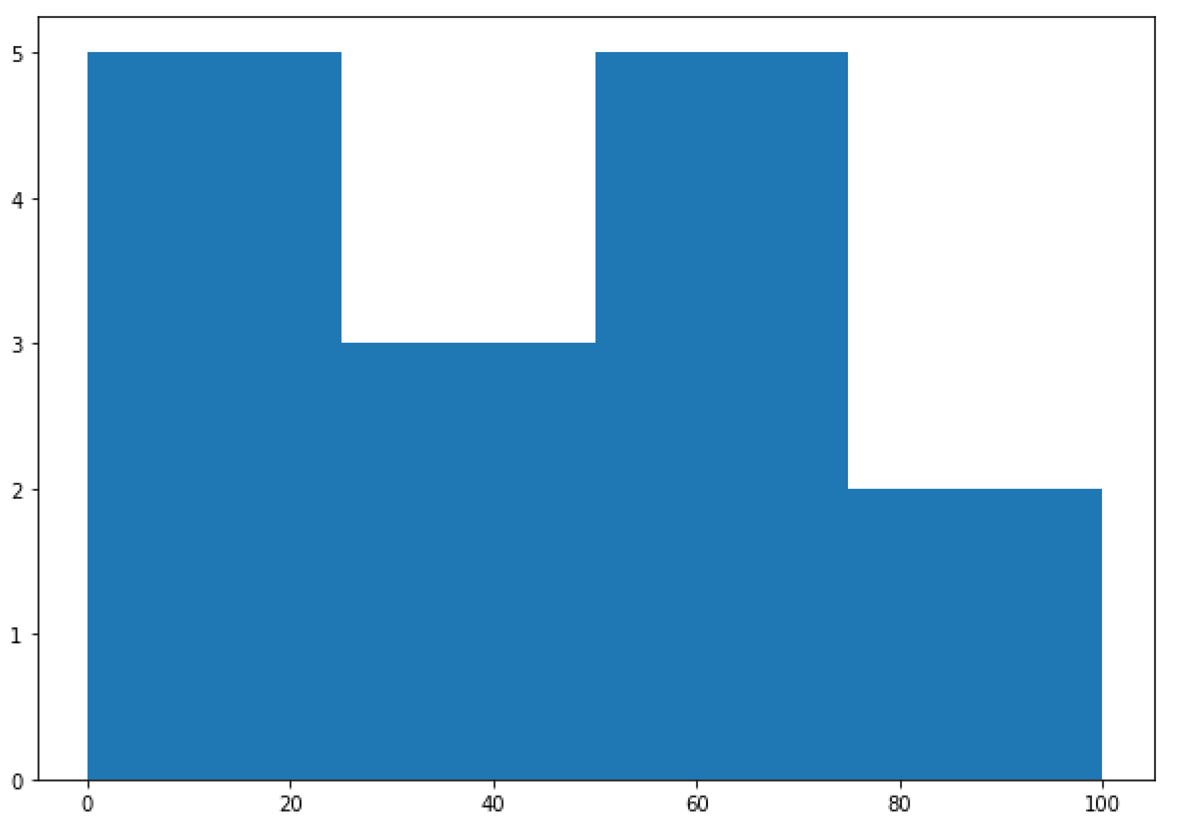
Let’s create a basic histogram of some random values. Below code creates a simple histogram of some random values:

- Python3

```
from matplotlib import pyplot as plt

import numpy as np



# Creating dataset

a = np.array([22, 87, 5, 43, 56,

              73, 55, 54, 11,

              20, 51, 5, 79, 31,

              27])


# Creating histogram

fig, ax = plt.subplots(figsize =(10, 7))

ax.hist(a, bins = [0, 25, 50, 75, 100])


# Show plot
```

```
plt.show()
```

Output :



Customization of Histogram

Matplotlib provides a range of different methods to customize histogram. matplotlib.pyplot.hist() function itself provides many attributes with the help of which we can modify a histogram.The hist() function provide a patches object which gives access to the properties of the created objects, using this we can modify the plot according to our will.

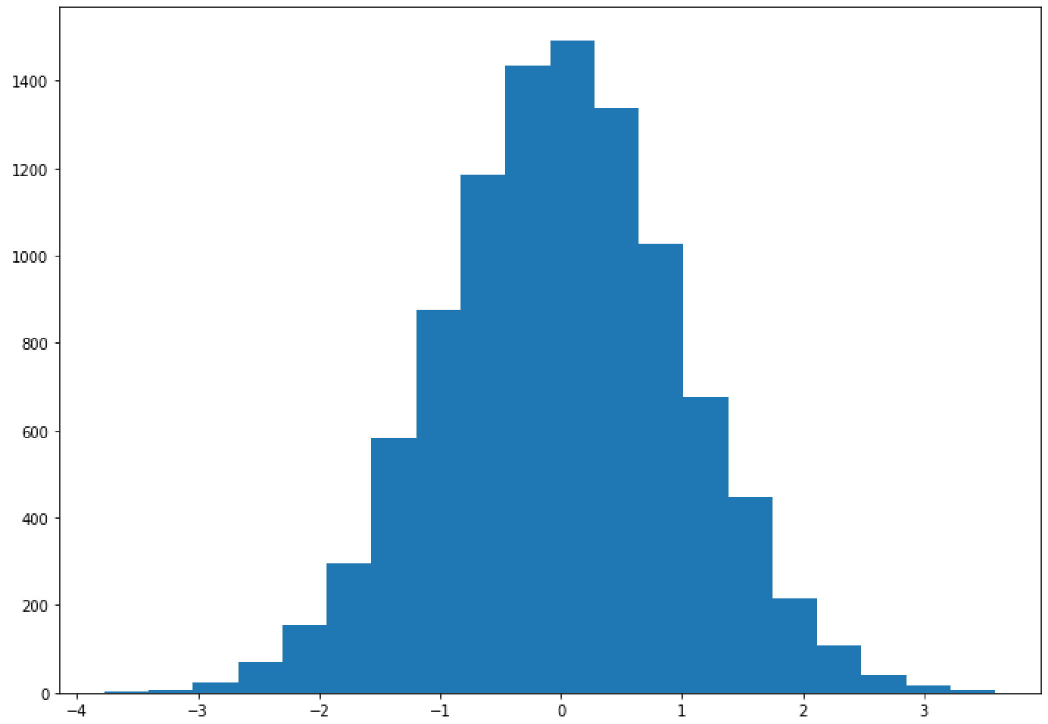
Example 1:

- Python3

```
import matplotlib.pyplot as plt

import numpy as np

from matplotlib import colors

from matplotlib.ticker import PercentFormatter



# Creating dataset
```

```
np.random.seed(23685752)
N_points = 10000
n_bins = 20

# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25

# Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)

axs.hist(x, bins = n_bins)

# Show plot
plt.show()
```

Output :



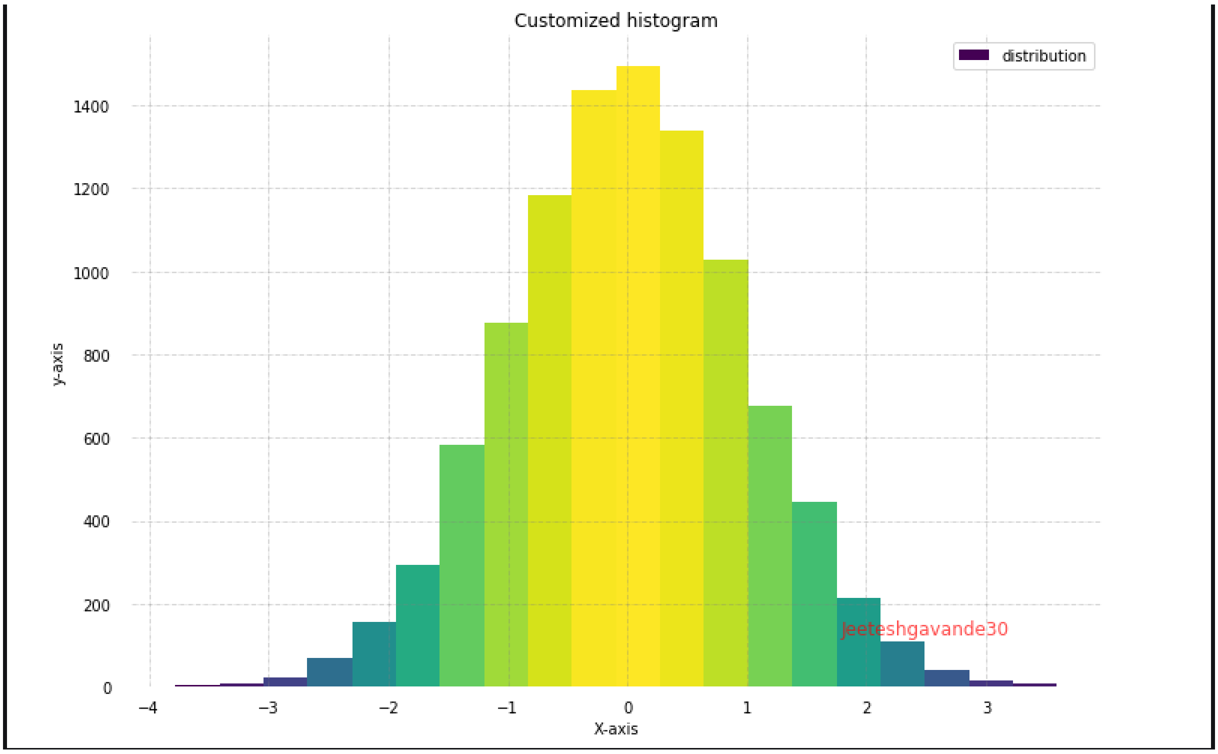
Example 2: The code below modifies the above histogram for a better view and accurate readings.

- Python3

```
import matplotlib.pyplot as plt

import numpy as np

from matplotlib import colors

from matplotlib.ticker import PercentFormatter




# Creating dataset

np.random.seed(23685752)

N_points = 10000

n_bins = 20


# Creating distribution

x = np.random.randn(N_points)

y = .8 ** x + np.random.randn(10000) + 25
```

```
legend = ['distribution']

# Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)


# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    axs.spines[s].set_visible(False)

# Remove x, y ticks
axs.xaxis.set_ticks_position('none')
axs.yaxis.set_ticks_position('none')

# Add padding between axes and labels
axs.xaxis.set_tick_params(pad = 5)
axs.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
axs.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.6)

# Add Text watermark
fig.text(0.9, 0.15, 'Jeeteshgavande30',
         fontsize = 12,
         color ='red',
         ha ='right',
         va ='bottom',

         alpha = 0.7)


# Creating histogram

N, bins, patches = axs.hist(x, bins = n_bins)


# Setting color

fracs = ((N**(1 / 5)) / N.max())

norm = colors.Normalize(fracs.min(), fracs.max())


for thisfrac, thispatch in zip(fracs, patches):

    color = plt.cm.viridis(norm(thisfrac))

    thispatch.set_facecolor(color)


# Adding extra features

plt.xlabel("X-axis")

plt.ylabel("y-axis")

plt.legend(legend)

plt.title('Customized histogram')


# Show plot

plt.show()
```

Output :


Customized histogram
distribution
1400
1200
1000
800
600
400
200
0
y-axis
−4
−3
−2
−1
0
1
2
3
X-axis
Jeeteshgavande30

Matplotlib Bars

❮ PreviousNext ❯

Creating Bars

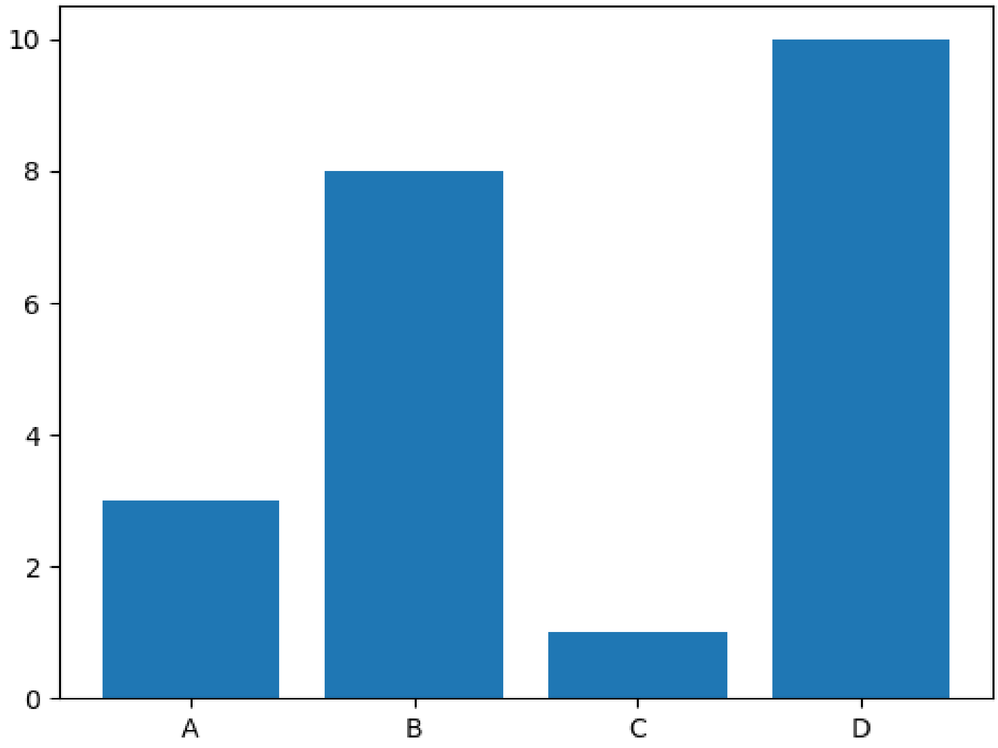
With Pyplot, you can use the `bar()` function to draw bar graphs:

Example

Draw 4 bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

Result:



Try it Yourself »

The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first* and *second* argument as arrays.

Example

```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```

Try it Yourself »

Horizontal Bars

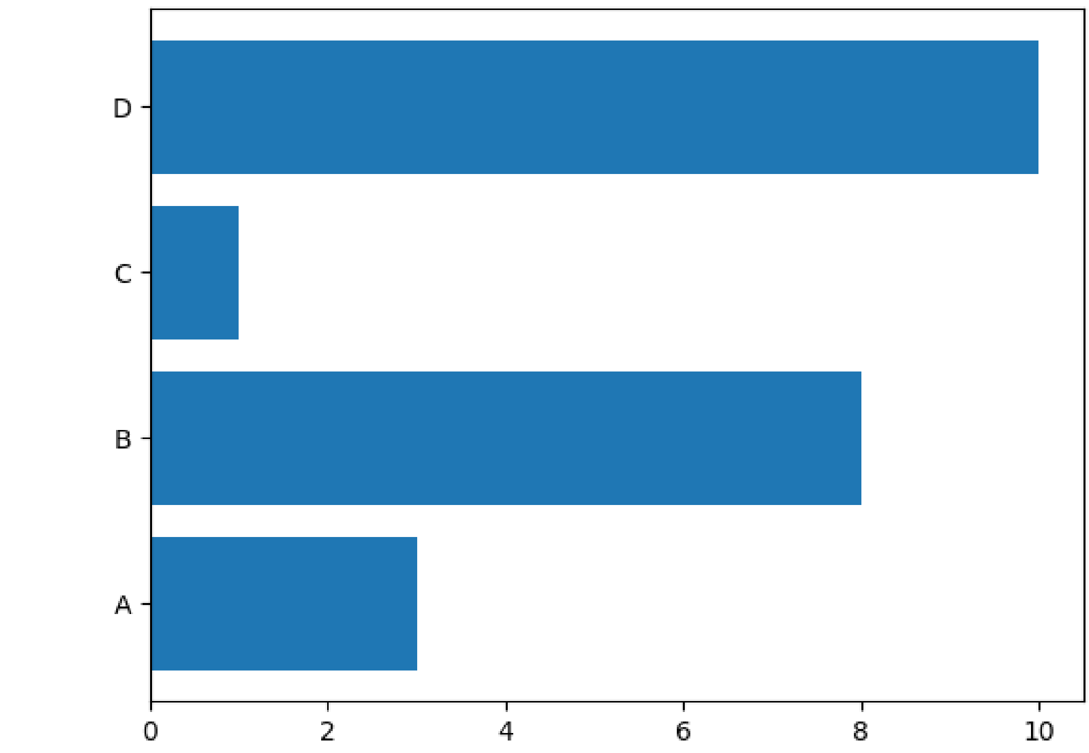
If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function:

Example

Draw 4 horizontal bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```

Result:



Try it Yourself »

Bar Color

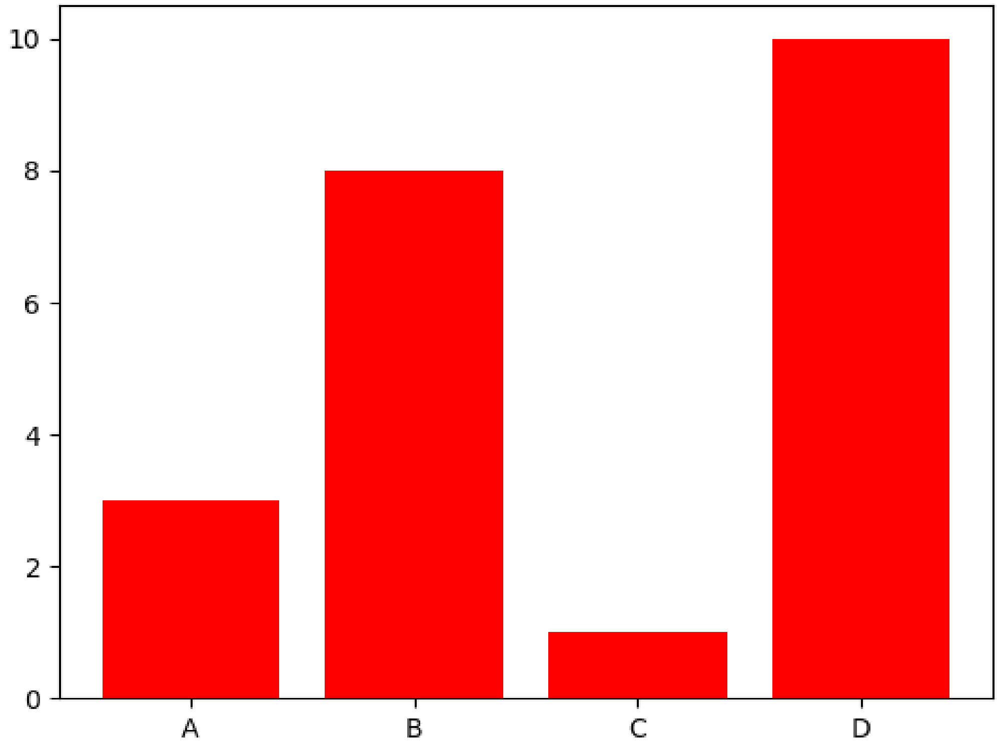
The `bar()` and `barh()` takes the keyword argument `color` to set the color of the bars:

Example

Draw 4 red bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

Result:



Try it Yourself »

Color Names

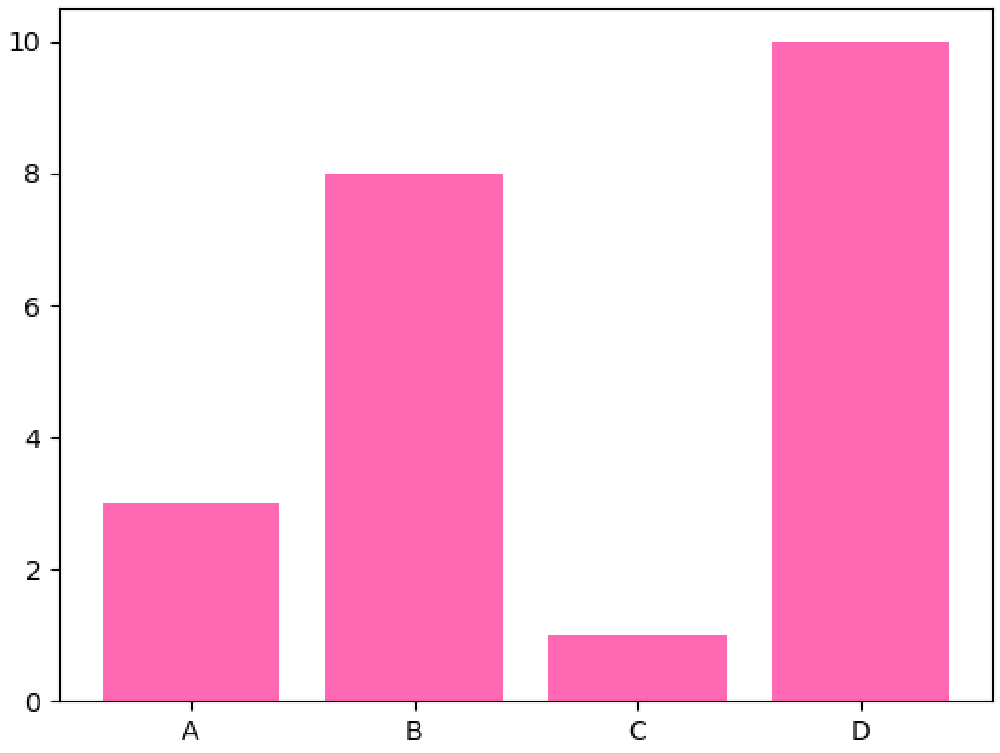
You can use any of the 140 supported color names.

Example

Draw 4 "hot pink" bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "hotpink")
plt.show()
```

Result:



Try it Yourself »

Color Hex

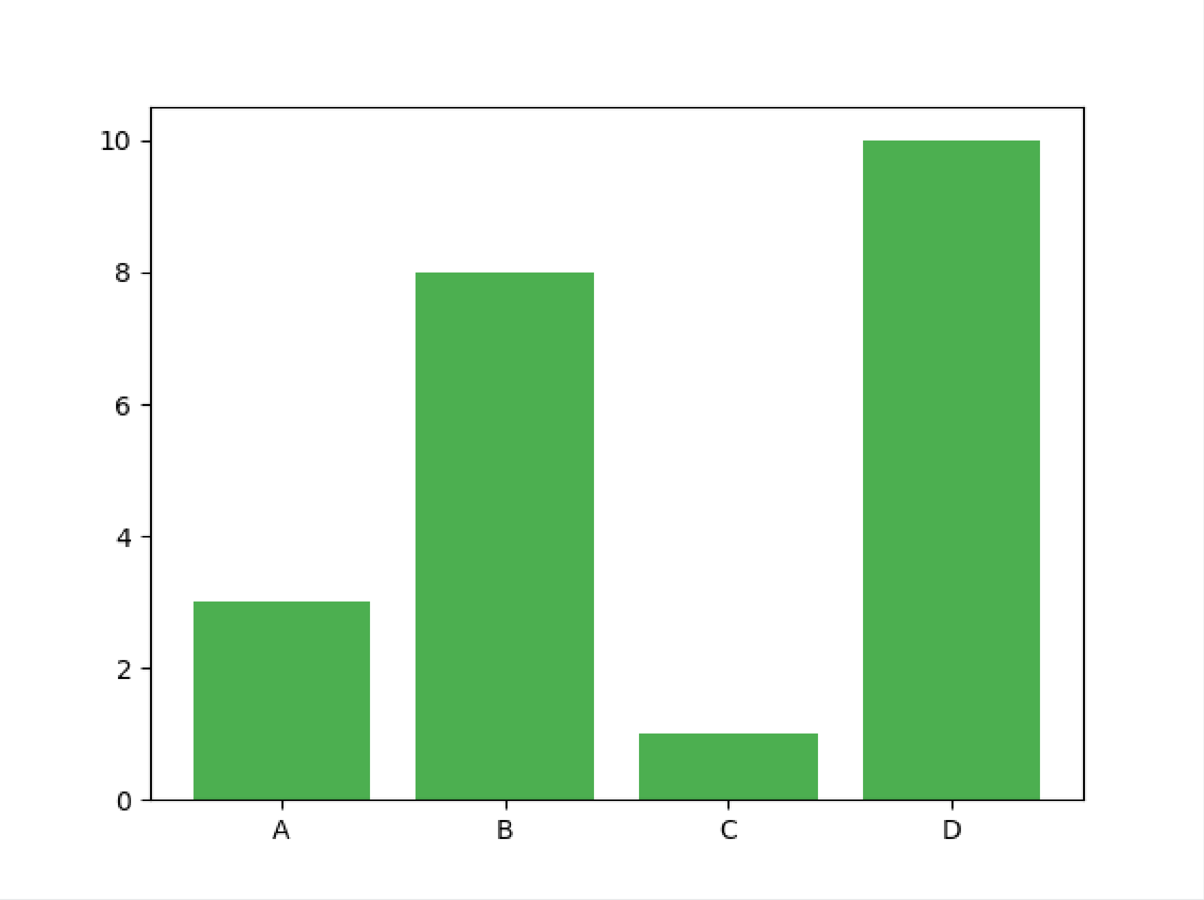
Or you can use Hexadecimal color values:

Example

Draw 4 bars with a beautiful green color:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```

Result:



Try it Yourself »

Bar Width

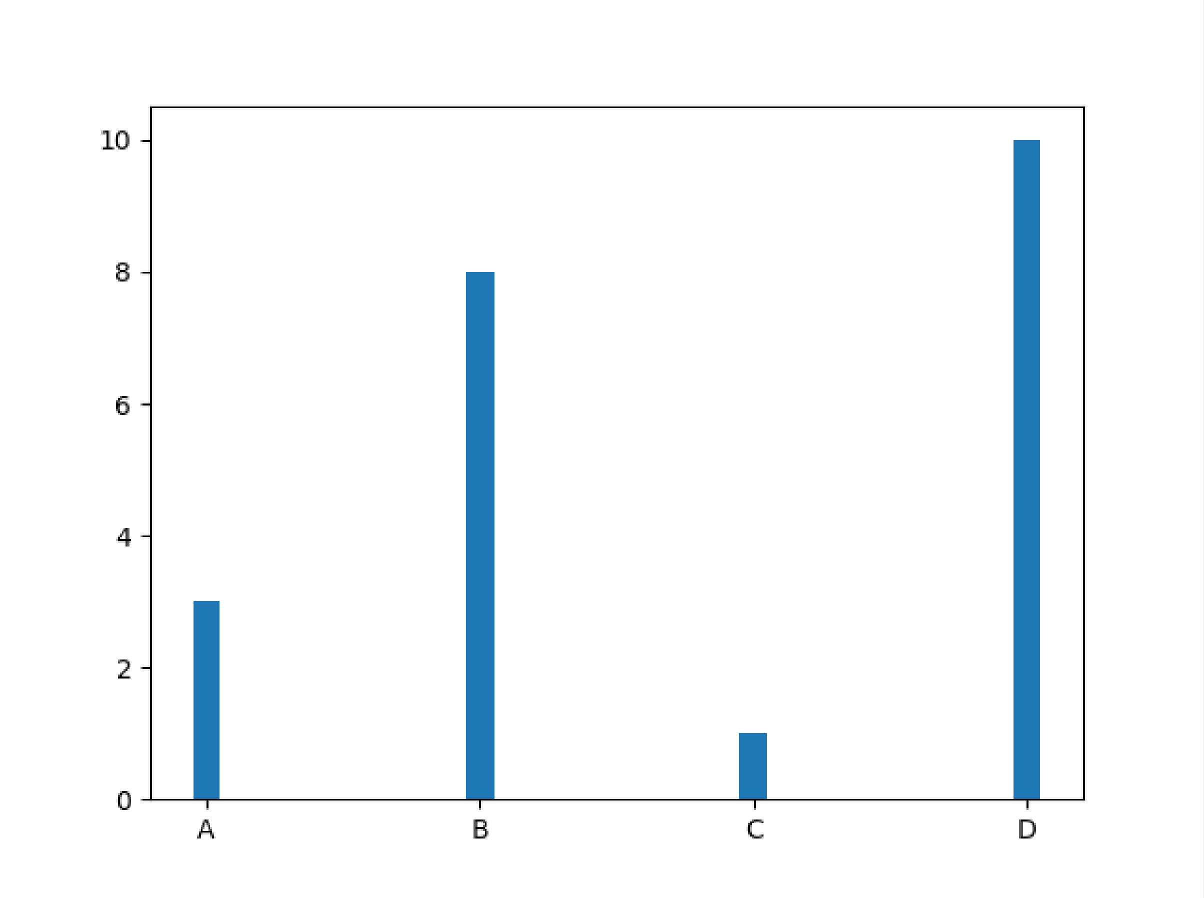
The `bar()` takes the keyword argument `width` to set the width of the bars:

Example

Draw 4 very thin bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```

Result:



Try it Yourself »

The default width value is 0.8

Note: For horizontal bars, use `height` instead of `width`.

Bar Height

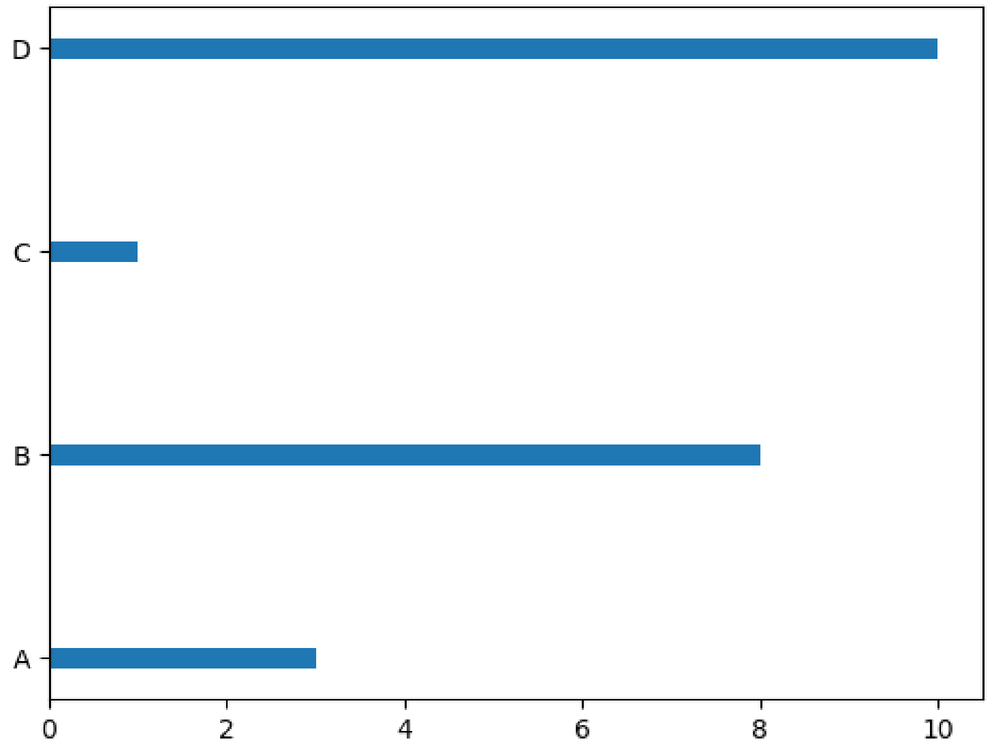
The `barh()` takes the keyword argument `height` to set the height of the bars:

Example

Draw 4 very thin bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

Result:



Try it Yourself »