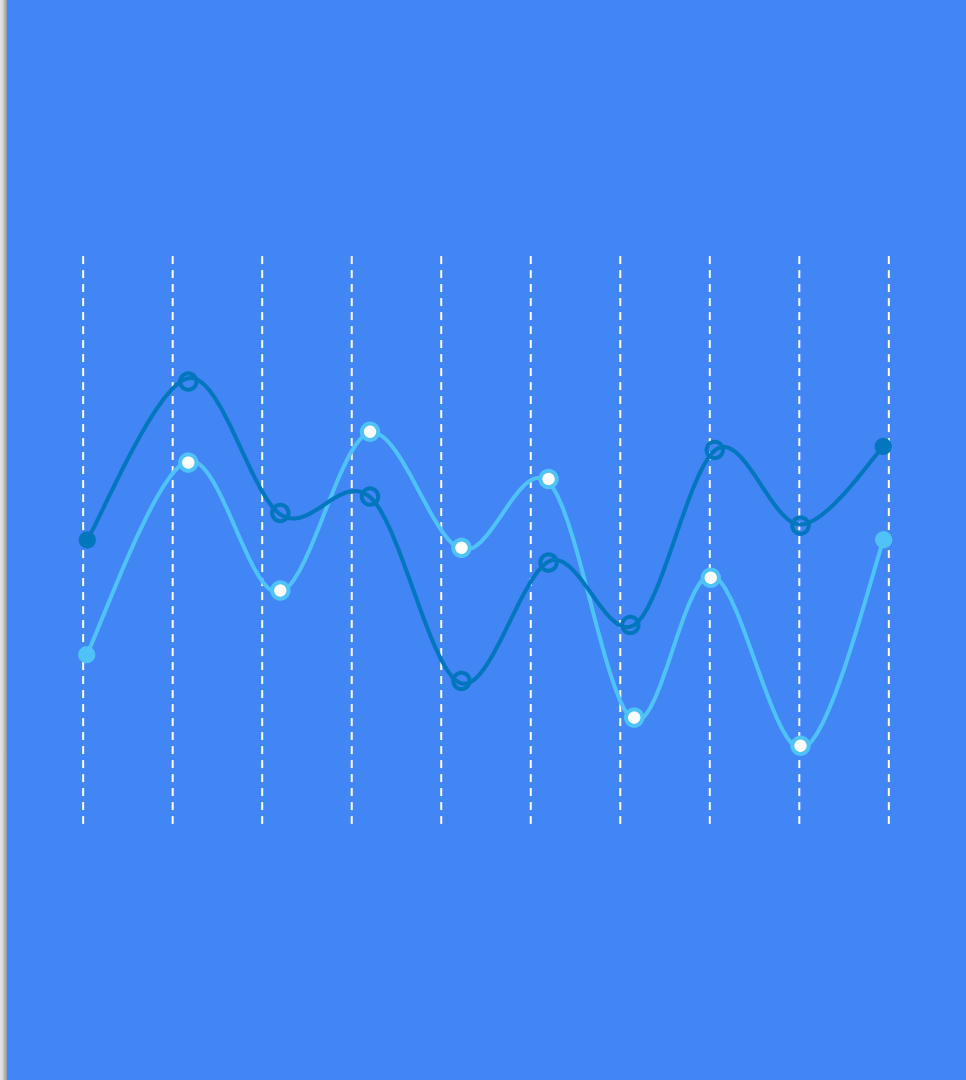
Certification Training Courses for Graduates and Professionals

www.signitivetech.com

Matplotlib: Python Plotting

Agenda

- Introduction
- Line Plots
- Bar Plots
- Scatter Plots
- Histograms
- Pie Charts
- Customization
- Styles
- Pandas Plot() Method
- Annotations and Text
- Subplots
- Custom Stuffs



Introduction

Introduction

- One of the most popular uses for Python is data analysis. Naturally, data scientists want a way to visualize their data. Either they want to see it for themselves to get a better grasp of the data, or they want to display the data to convey their results to someone.
- With Matplotlib, arguably the most popular graphing and data visualization module for Python, this is very simplistic to do.
- Matplotlib is a tool for data visualization and this tool built upon the Numpy and Scipy framework. Matplotlib is a library for making 2D plots of arrays in Python.
- Before to use matplotlib, we have to import this library as:
  - `import matplotlib.pyplot as plt`

Introduction

Graph: What is a graph?

- Two-dimensional drawing showing a relationship (usually between two set of numbers) by means of a line, curve, a series of bars, or other symbols.
- Typically, an independent variable is represented on the horizontal line (X-axis) and an dependent variable on the vertical line (Y-axis).
- The perpendicular axis intersect at a point called origin, and are calibrated in the units of the quantities represented.
- Though a graph usually has four quadrants representing the positive and negative values of the variables, usually only the north-east quadrant is shown when the negative values do not exist or are of no interest.

Line Plots

Line Plots

Line Graph:

- Graphical device that displays quantitative information or illustrates relationships between two changing quantities (variables) with a line or curve that connects a series of successive data points.
- A grouped line graph compares a trend with one or more other trends, and shows if its rate of change is increasing, decreasing, fluctuating, or remaining constant.
- Line graphs are the most versatile and most extensively used family of graphs and they also called as line chart.

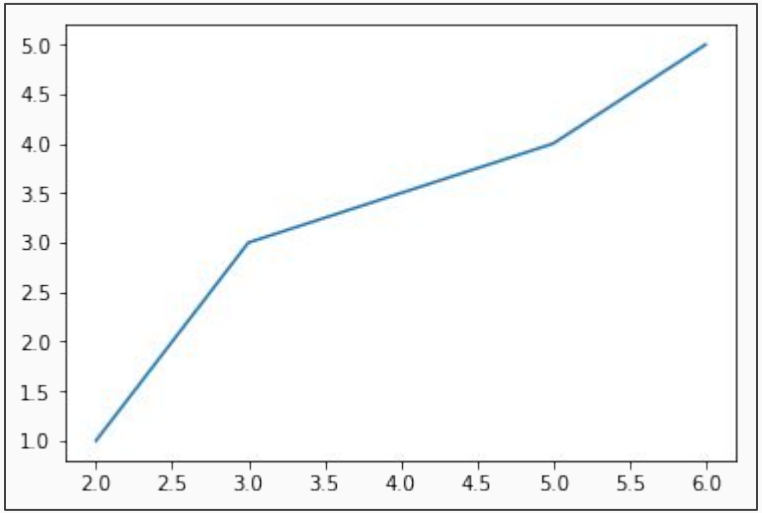
- Line Graph is a graph with points connected by lines to show how something changes in value:
  - as time goes by
  - or as something else changes.

Line Plots

```
x = [2, 3, 5, 6]
y = [1, 3, 4, 5]
plt.plot(x, y)      → plot(): Method used for Plotting the Line
plt.show()          → show(): Method used for Displaying the Figure
```

Line Plots

Output:

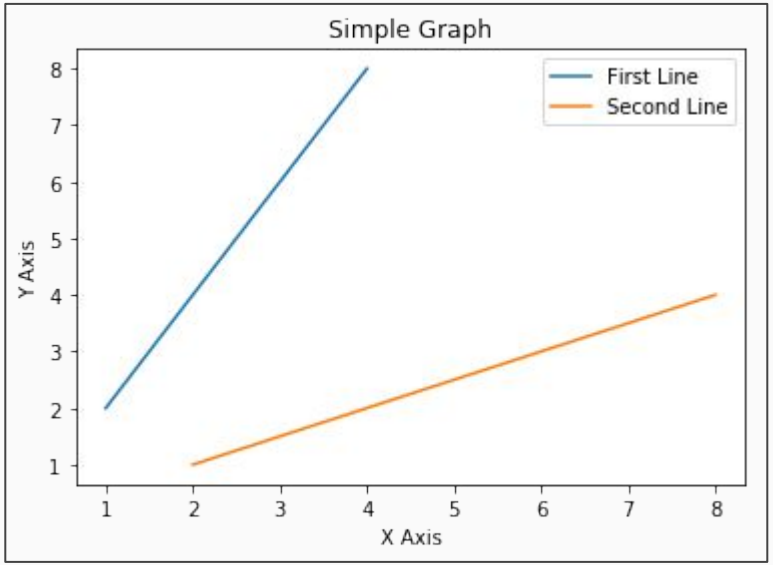


Line Plots

```
x1 = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]

x2 = [2, 4, 6, 8]
y2 = [1, 2, 3, 4]

plt.plot(x1, y1, label = 'First Line') → label: To assign a name to the line
plt.plot(x2, y2, label = 'Second Line')
plt.xlabel('X Axis')      → xlabel(): To assign a label to x - axis
plt.ylabel('Y Axis')      → ylabel(): To assign a label to y - axis
plt.title('Simple Graph')    → title(): To provide the plot's title
plt.legend()              → legend(): To place the legends in the axes
plt.show()
```

Line Plots

Output:



Line Plots

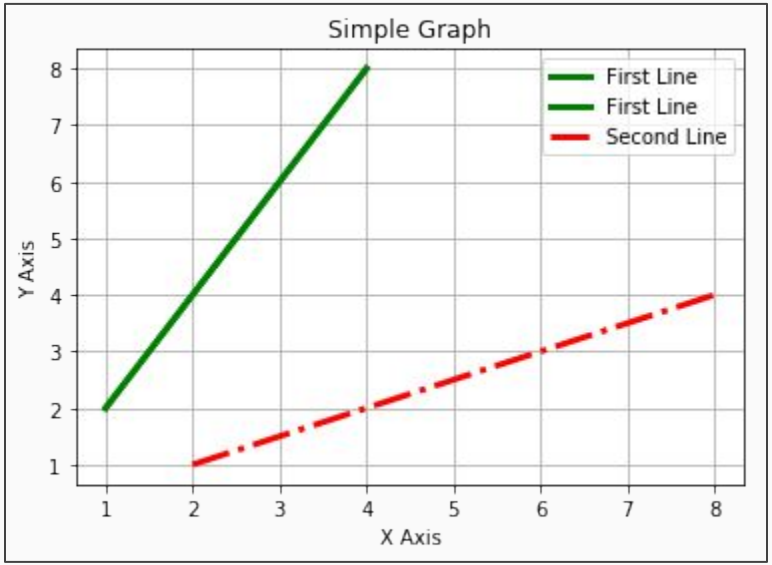
```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

plt.plot(x1, y1, color = 'g', linewidth = 3, linestyle = '-', label = 'First Line')
plt.plot(x2, y2, color = 'r', linewidth = 3, linestyle = '-.', label = 'Second
Line')

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid() → grid(): To turn the axes grids on or off
plt.legend()
plt.show()
```

Line Plots

```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

plt.plot(x1, y1, 'g-' , label = 'First Line')       → Shortcut Declaration
plt.plot(x2, y2, 'r-.' , label = 'Second Line')  → Shortcut Declaration

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid() → grid(): To turn the axes grids on or off
plt.legend()
plt.show()
```

Line Plots

Output:



Line Plots

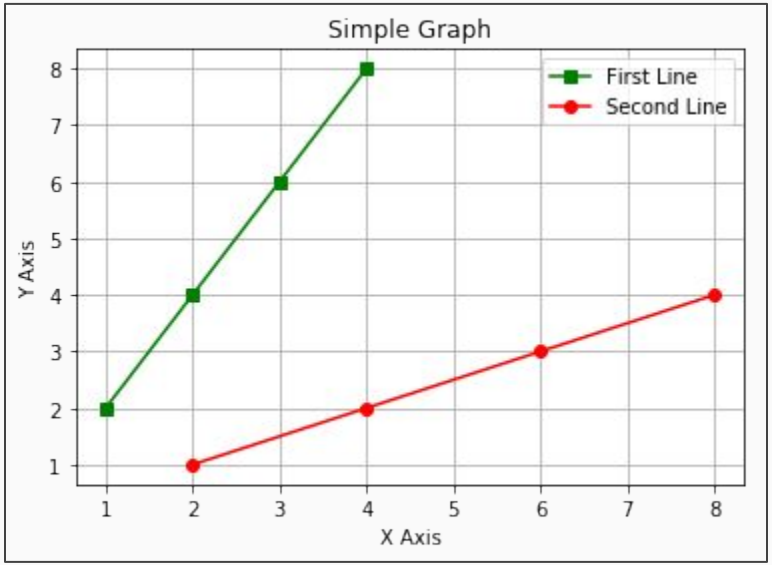
```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

plt.plot(x1, y1, 'g-', marker = 's', markersize = 6, label = 'First Line')
plt.plot(x2, y2, 'r-', marker = 'o', markersize = 6, label = 'Second Line')

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid()
plt.legend()
plt.show()
```

Line Plots

```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

# alternative to previous code:
plt.plot(x1, y1, 'g-s', markersize = 6, label = 'First Line')
plt.plot(x2, y2, 'r-o', markersize = 6, label = 'Second Line')

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid()
plt.legend()
plt.show()
```

Line Plots

Output:



Line Plots

Line Styles	
'-'	Solid Line
'--'	Dashed Line
'-.'	Dash-Dot Line
':'	Dotted Line

Colors	
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Line Plots

Marker Style	
'.'	Point
','	Pixel
'o'	Circle
'v'	Triangle_Down
'^'	Triangle_Up
'<'	Triangle_Left
'>'	Triangle_Right

Marker Style	
'1'	Tri_Down
'2'	Tri_Up
'3'	Tri_Left
'4'	Tri_Right
's'	Square
'p'	Pentagon
'*'	Star

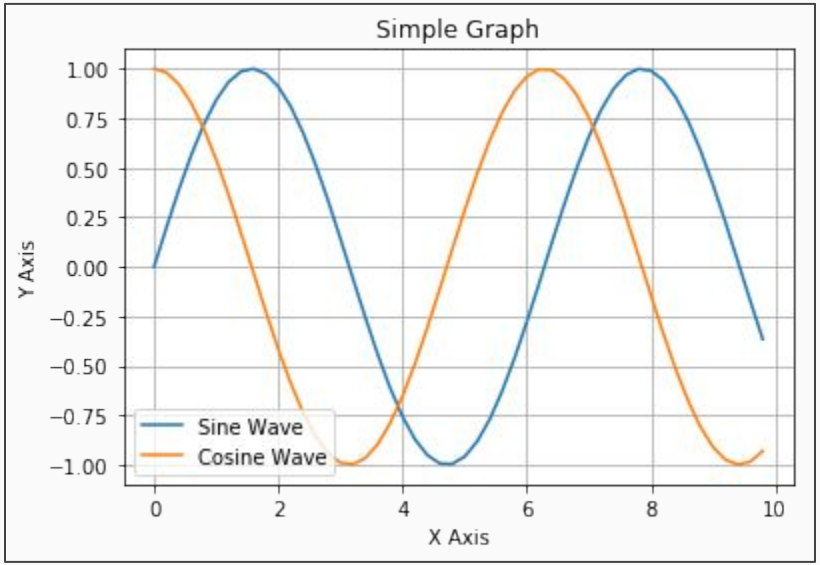
Marker Style	
'h'	Hexagon1
'H'	Hexagon2
'+'	Plus
'x'	X
'D'	Diamond
'd'	Thin_Diamond
'|'	VLine
'_'	HLine

Line Plots

```
# Another Example:
x = np.arange(0, 10, 0.2)
s = np.sin(x)
c = np.cos(x)

plt.plot(x, s, label='Sine Wave')
plt.plot(x, c, label='Cosine Wave')

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid()
plt.legend()
plt.show()
```

Line Plots

Output:



Bar Plots

Bar Plots

Bar Graph / Bar Chart:

- Horizontal rectangles (bars) chart in which the length of a bar is proportional to the value (as measured along the horizontal axis) of the item (entity or quantity) it represents.
- This is also called bar graph, it is used commonly to compare the values of several items in a group at a given point in time.

- Bar Graph is a graph drawn using rectangular bars to show how large each value is.
- The bars can be horizontal or vertical.

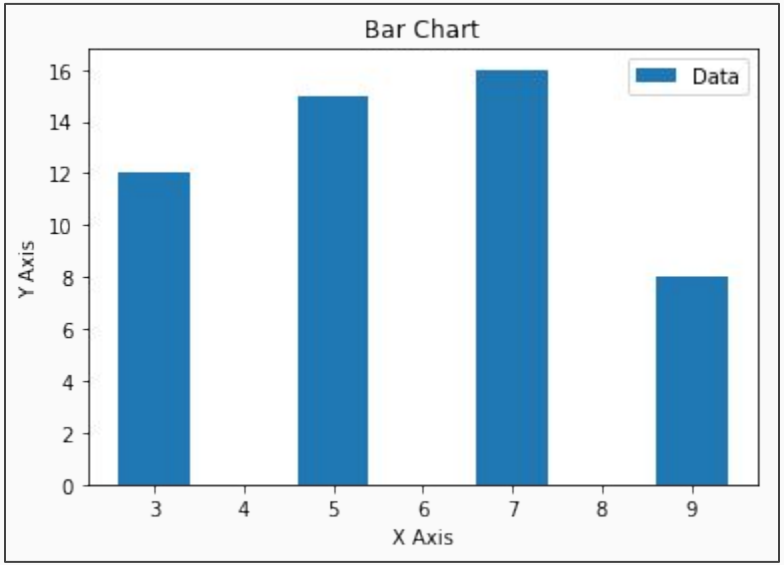
Bar Plots

```
x = [3, 5, 7, 9]
y = [12, 15, 16, 8]

plt.bar(x, y, label = 'Data')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Bar Chart')
plt.legend()
plt.show()
```

→ bar(): To make a Bar Plot

Bar Plots

Output:

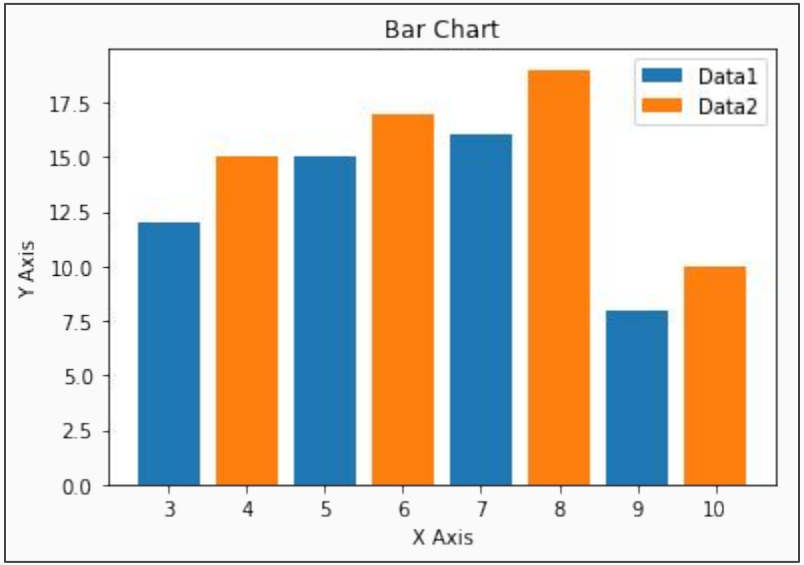


Bar Plots

```
x1 = [3, 5, 7, 9]
y1 = [12, 15, 16, 8]

x2 = [4, 6, 8, 10]
y2 = [15, 17, 19, 10]

plt.bar(x1, y1, label = 'Data1')
plt.bar(x2, y2, label = 'Data2')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Bar Chart')
plt.legend()
plt.show()
```

Bar Plots

Output:

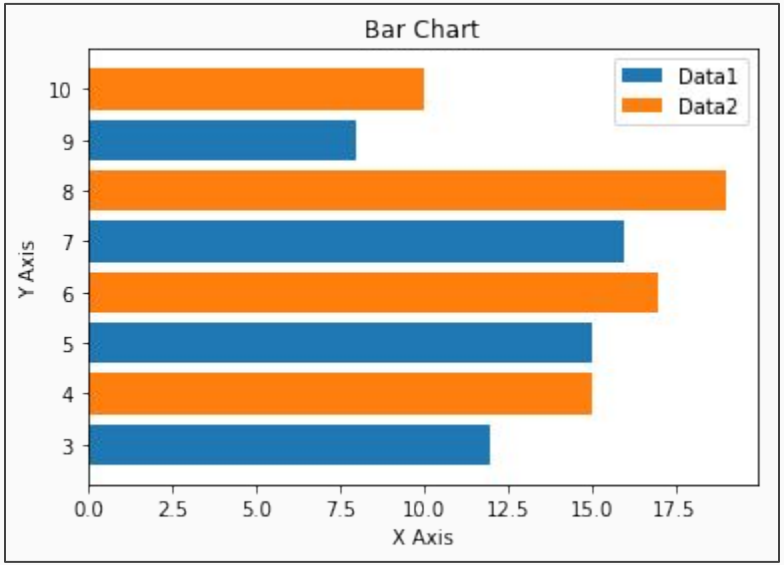


Bar Plots

```
x1 = [3, 5, 7, 9]
y1 = [12, 15, 16, 8]

x2 = [4, 6, 8, 10]
y2 = [15, 17, 19, 10]

plt.barh(x1, y1, label = 'Data1') → barh(): To make a Horizontal Bar Plot
plt.barh(x2, y2, label = 'Data2') → barh(): To make a Horizontal Bar Plot
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Bar Chart')
plt.legend()
plt.show()
```

Bar Plots

Output:



Bar Plots

Consider the following Dataset:

```
df = pd.DataFrame(np.random.randint(1, 40, size = [5, 4]),
columns=['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
0	20	6	31	26
1	31	27	21	28
2	28	13	4	30
3	22	25	28	2
4	19	9	20	23

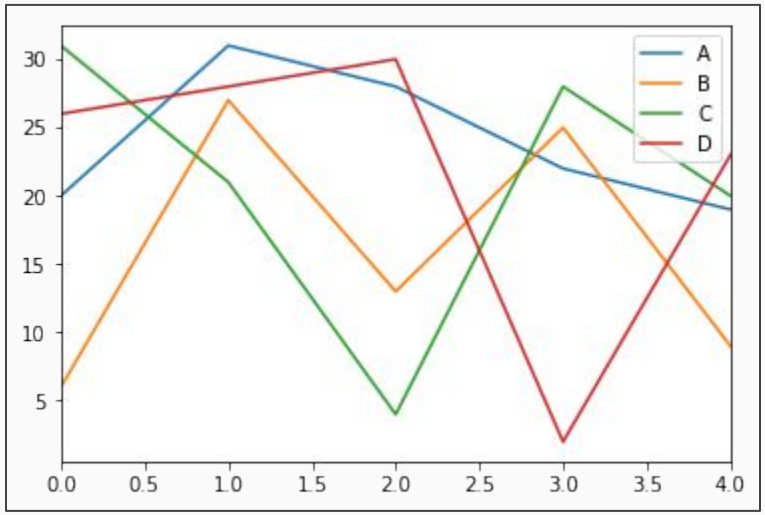
Bar Plots

We use Pandas built-in method plot() to make a graph:

df.plot.line()

plt.show()

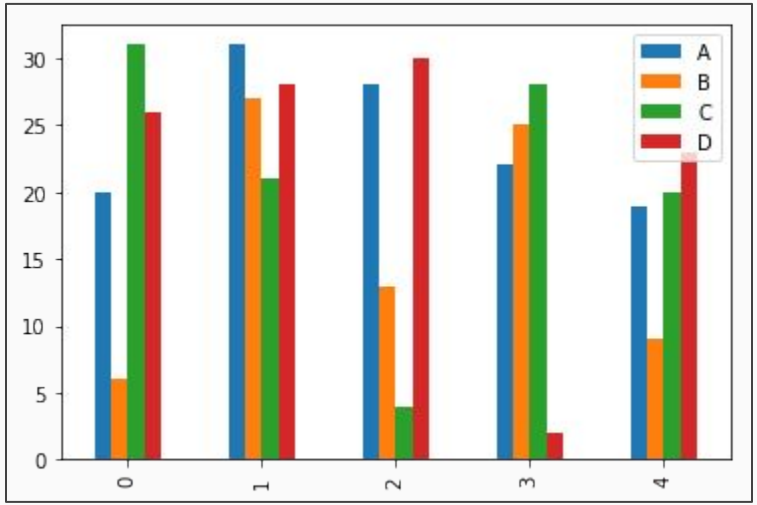
Output:



Bar Plots

```
df.plot.bar()
plt.show()
```

Output:



Bar Plots

Consider another Dataset:

```
company = ['Infosys', 'TCS', 'HCL',
'Wipro', 'Google', 'FB', 'Amazon', 'Twitter']

df = pd.DataFrame({'Revenue (in
$million)': [200, 240, 300, 140, 600, 770,
250, 450]}, index=company)

df
```

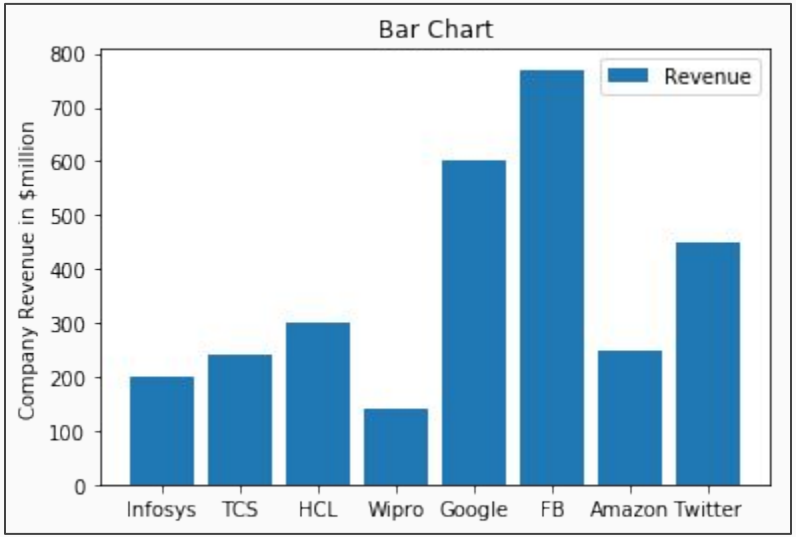
	Revenue (in $million)
Infosys	200
TCS	240
HCL	300
Wipro	140
Google	600
FB	770
Amazon	250
Twitter	450

Bar Plots

```
y = df.values

y = y.flatten()

xpos = np.arange(len(company))
plt.bar(xpos, y, label = 'Revenue')
plt.xticks(xpos, company)        → xticks(): To set tick labels on the x-axis
plt.ylabel('Company Revenue in $million')
plt.title('Bar Chart')
plt.legend()
plt.show()
```

Bar Plots

Output:



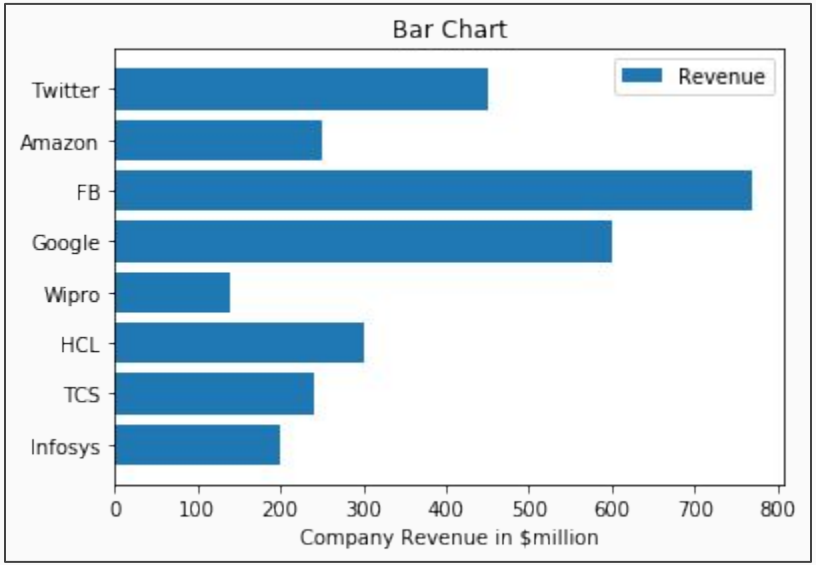
Bar Plots

```
y = df.values

y = y.flatten()

xpos = np.arange(len(company))
plt.barh(xpos, y, label = 'Revenue')
plt.yticks(xpos, company)
plt.xlabel('Company Revenue in $million')
plt.title('Bar Chart')
plt.legend()
plt.show()
```

→ yticks(): To Set tick labels on the y-axis

Bar Plots

Output:



Scatter Plots

Scatter Plots

Scatter Plot:

- Two or three dimensional chart in which the density and direction of the plotted points indicates the type of relationship (or a lack thereof) between dependent and independent variables.
- This is also called as scatter diagram, scatter graph, or scatter plot, it is one of the seven tools of quality.

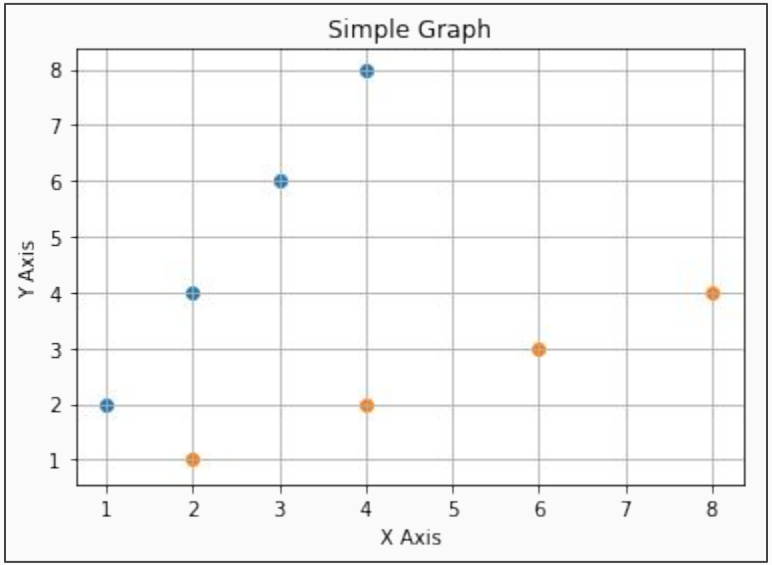
- Scatter Plot is a graph of plotted points that show the relationship between two sets of data.

Scatter Plots

```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

plt.scatter(x1, y1) → scatter(): To make a Scatter Plot of x vs y
plt.scatter(x2, y2)

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid()
plt.show()
```

Scatter Plots

Output:

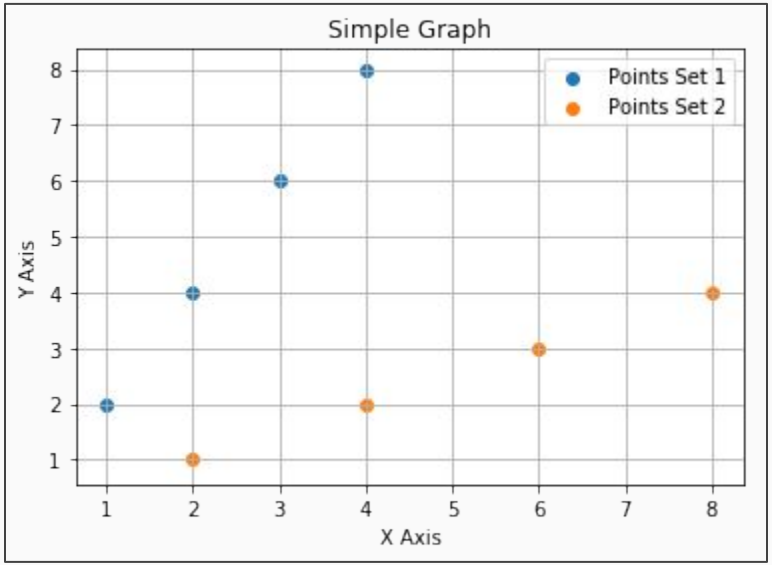


Scatter Plots

```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]

plt.scatter(x1, y1, label='Points Set 1')
plt.scatter(x2, y2, label='Points Set 2')

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Graph')
plt.grid()
plt.legend()
plt.show()
```

Scatter Plots

Output:

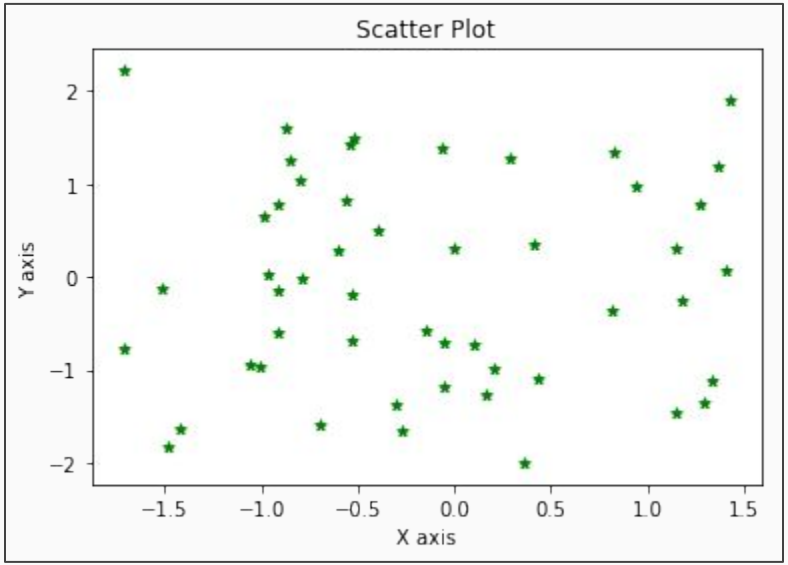


Scatter Plots

```
x = np.random.randn(1, 50)
y = np.random.randn(1, 50)

plt.scatter(x, y, color='green', marker='*', s=30)

plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('Scatter Plot')
plt.show()
```

Scatter Plots

Output:



Histogram

Histogram

Histogram:

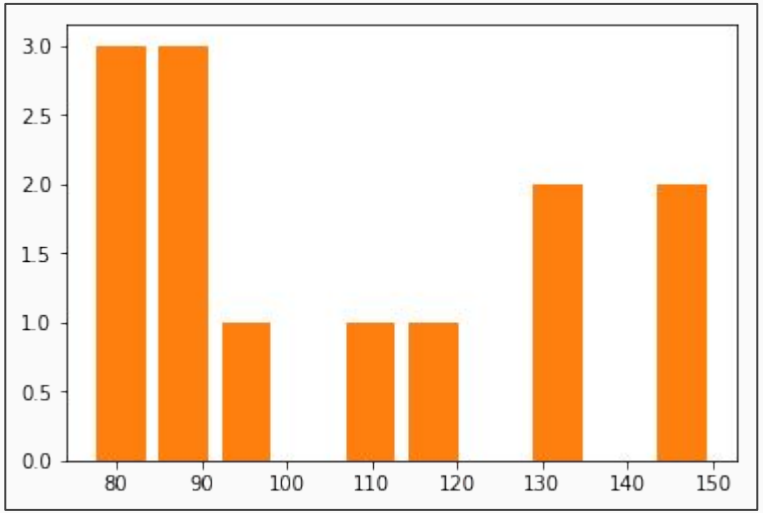
- Step-column chart that displays a summary of the variations in (frequency distribution of) quantities (called Classes) that fall within certain lower and upper limits in a set of data.
- Classes are measured on the horizontal ('X') axis, and the number of times they occur (or the percentages of their occurrences) are measured on the vertical ('Y') axis.
- To construct a histogram, rectangles or blocks are drawn on the x-axis (without any spaces between them) whose areas are proportional to the classes they represent.
- Histograms are used commonly where the subject item is discrete (such as the number of students in a school) instead of being continuous (such as the variations in their heights).

Histogram

```
blood_sugar = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]

plt.hist(blood_sugar, rwidth = 0.8) → hist(): To plot a Histogram
                                    → rwidth: The relative width of bars

plt.show()
```

Histogram

Output:

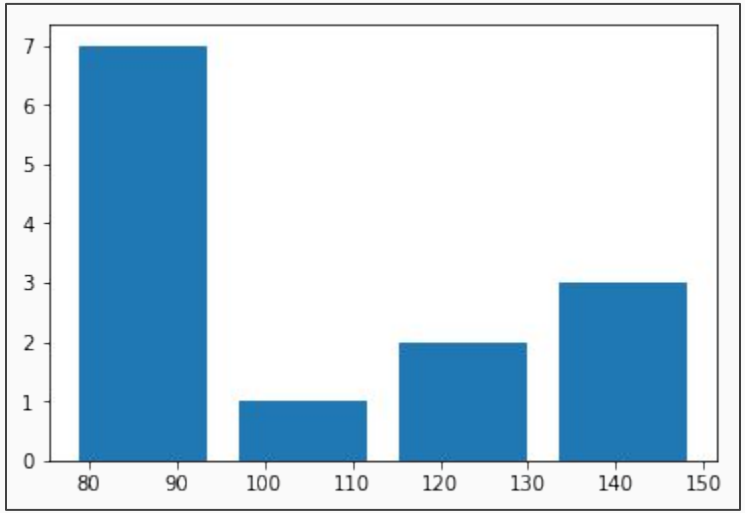


Histogram

```
blood_sugar = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]

plt.hist(blood_sugar, rwidth = 0.8, bins = 4)
                                    → bins: Number of Bars, default = 10

plt.show()
```

Histogram

Output:



Histogram

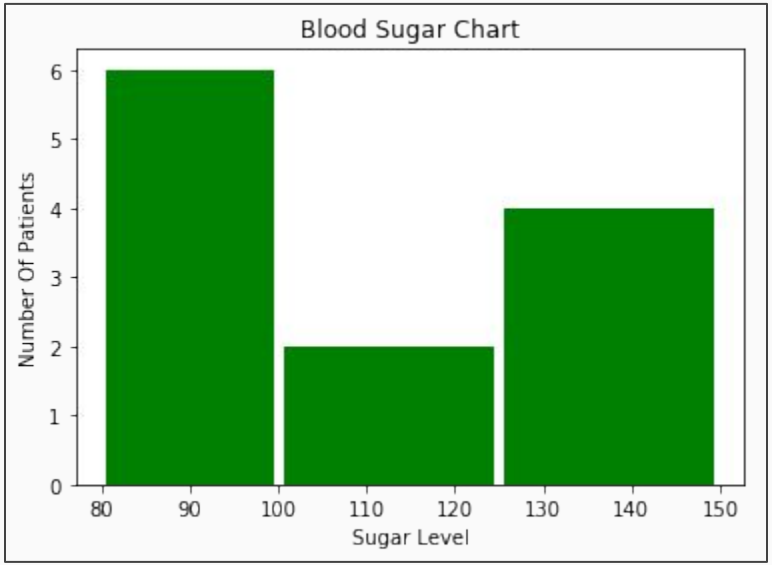
```
blood_sugar = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]

plt.hist(blood_sugar, bins=[80,100,125,150], rwidth=0.95, color='g')

plt.xlabel("Sugar Level")
plt.ylabel("Number Of Patients")
plt.title("Blood Sugar Chart")
plt.show()
```

Bin Range:

- 80-100: Normal
- 100-125: Pre-Diabetic
- 125-150: Diabetic

Histogram

Output:

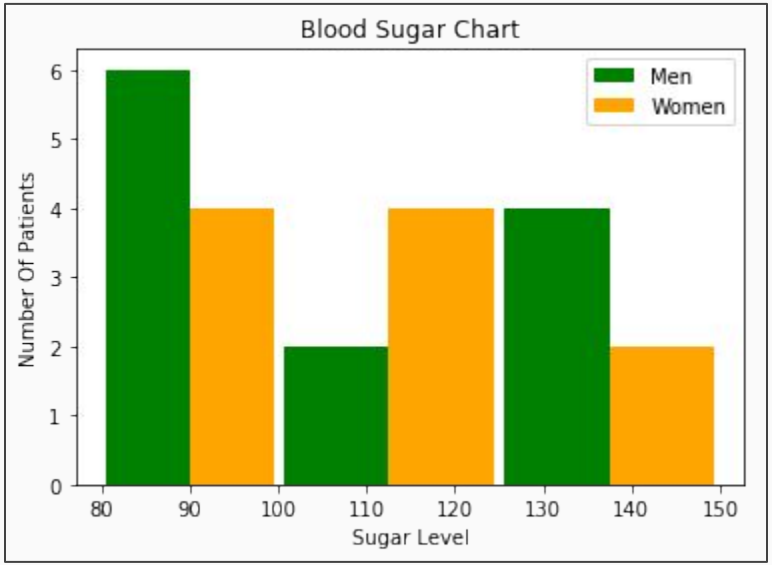


Histogram

```
blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([blood_sugar_men,blood_sugar_women],
bins=[80,100,125,150], rwidth=0.95, color=['green','orange'],
label=['Men','Women'])

plt.xlabel("Sugar Level")
plt.ylabel("Number Of Patients")
plt.title("Blood Sugar Chart")
plt.legend()
plt.show()
```

Histogram

Output:



Pie Charts

Pie Charts

Pie Charts:

- Proportional area chart used almost exclusively in showing relative sizes of the components of a data-set, in comparison to one another and to the whole set. It consists of a circle (disc) divided into several (usually not exceeding six) segments.
- Area of each segment (called slice or wedge) is of the same percentage of the circle as the component it represents is of the whole data set.
- Most important segment is commonly shown in the '12 o'clock' position with each successive less important segment following in a clockwise direction. This is also called as circle diagram, circle graph, pizza chart, or sector graph.
- A Pie Chart (or Pie Graph) is a circular chart divided into sectors, each sector shows the relative size of each value.

Pie Charts

```
expense_values = [1400,600,300,410,250]
expense_labels = ["HomeRent", "Food", "Phone/Internet Bill", "Car
Maintenance", "Other Utilities"]

plt.pie(expense_values, labels=expense_labels)
```

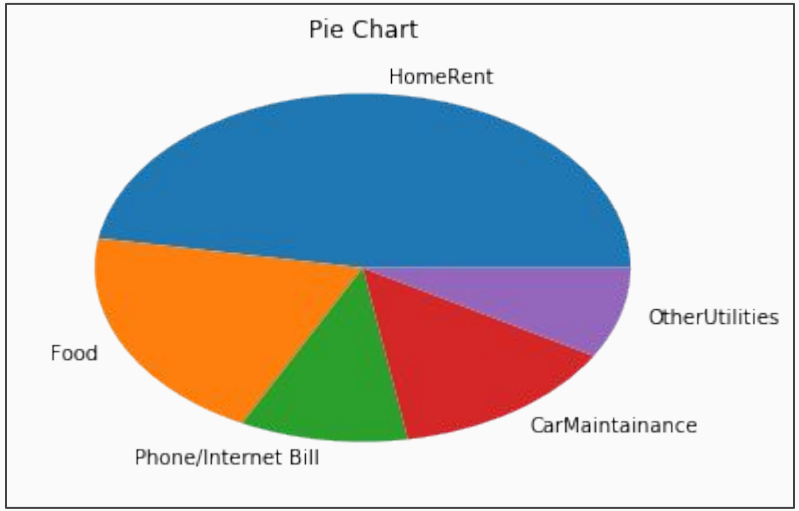
→ pie(): To Plot a Pie Chart

→ labels: TO give a sequence of strings providing the labels for each wage

```
plt.title('Pie Chart')
plt.show()
```

Pie Charts

Output:



Pie Charts

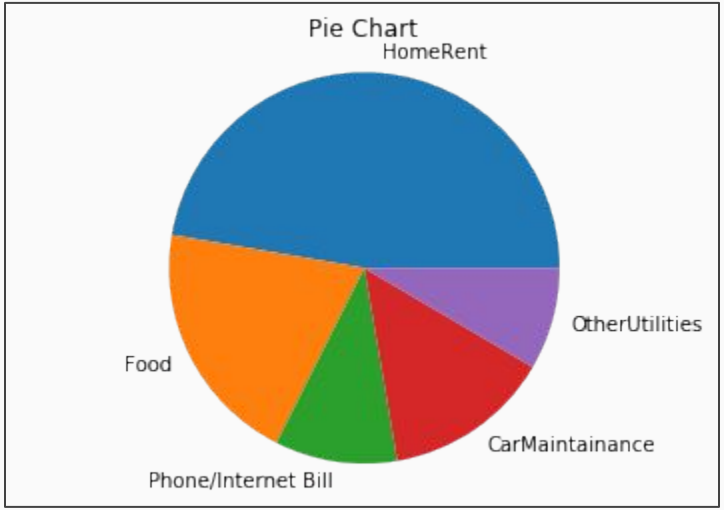
```
expense_values = [1400,600,300,410,250]
expense_labels = ["HomeRent", "Food", "Phone/Internet Bill", "Car Maintenance", "Other Utilities"]

plt.pie(expense_values, labels=expense_labels)
plt.title('Pie Chart')
```

plt.axis('equal') → axis(): Convenience method to set or get axis properties, and ‘equal’ means x-axis and y-axis have same length i.e. a Circle is Circular

```
plt.show()
```

Pie Charts

Output:

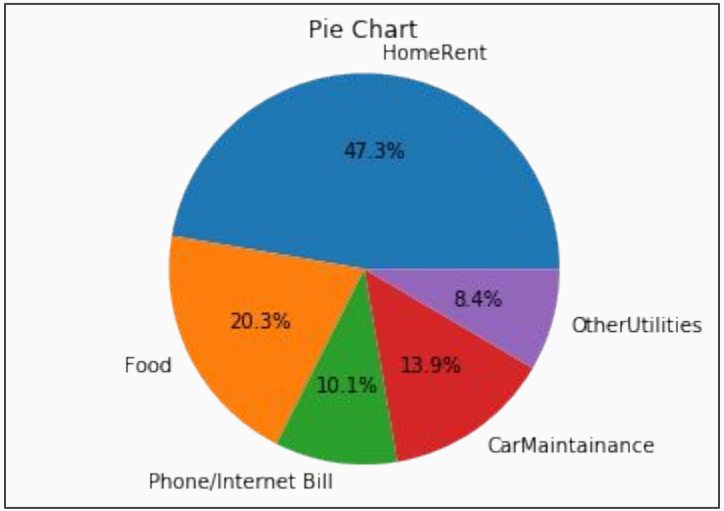


Pie Charts

```
expense_values = [1400,600,300,410,250]
expense_labels = ["HomeRent", "Food", "Phone/Internet Bill", "Car
Maintenance", "Other Utilities"]

plt.pie(expense_values, labels=expense_labels, autopct='%1.1f%%',
radius=1.5) → autopct: To label the wages with their Numeric Values
                → radius: The radius of the Pie

plt.title('Pie Chart')
plt.axis('equal')
plt.show()
```

Pie Charts

Output:

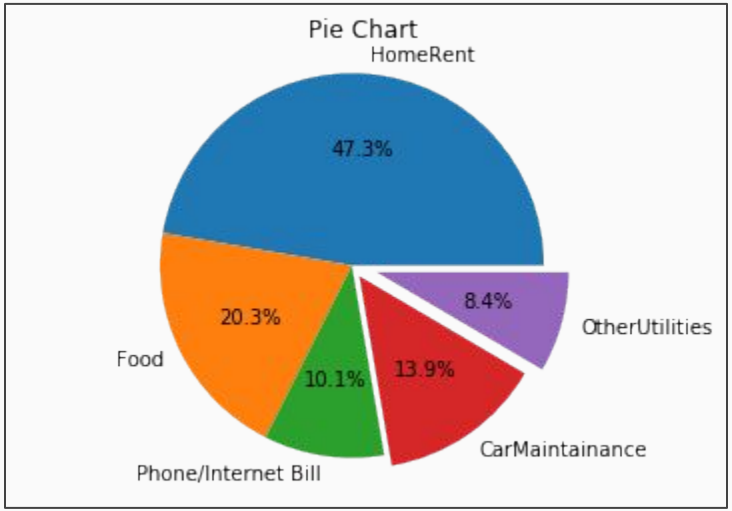


Pie Charts

```
expense_values = [1400,600,300,410,250]
expense_labels = ["HomeRent", "Food", "Phone/Internet Bill", "Car
Maintenance", "Other Utilities"]

plt.pie(expense_values, labels=expense_labels, autopct='%1.1f%%',
radius=1.5, explode=[0,0,0,0.1,0.2])
→ explode: To offset each Wage by specified fraction

plt.title('Pie Chart')
plt.axis('equal')
plt.show()
```

Pie Charts

Output:



Pie Charts

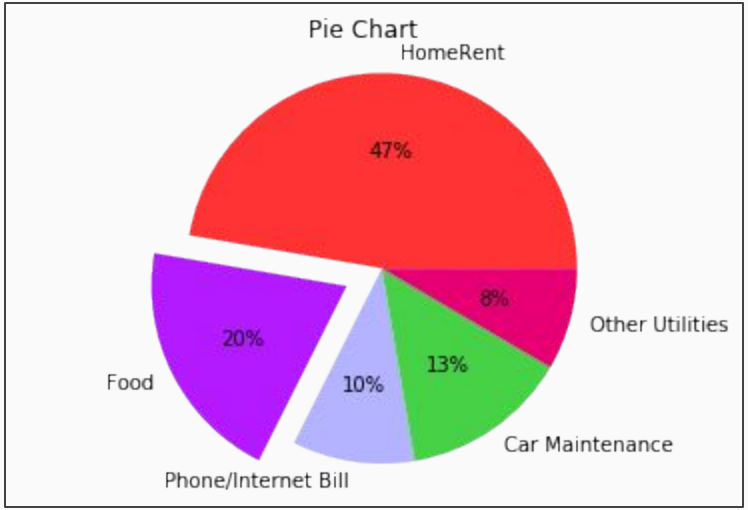
```
expense_values = [1400,600,300,410,250]
expense_labels = ["HomeRent", "Food", "Phone/Internet Bill", "Car
Maintenance", "Other Utilities"]

plt.pie(expense_values, labels=expense_labels, autopct='%d%%',
        colors=['#ff3333', '#b31aff', '#b3b3ff', '#47d147', '#e60073'],
        explode = [0,0.2,0,0,0])
```

→ colors: To give different colors to each Pie (Hexadecimal Code Required for Color Codes)

```
plt.title('Pie Chart')
plt.axis('equal')
plt.show()
```

Pie Charts

Output:



Customization

Customization

Consider the Dataset:

```
data = pd.read_csv("../DataSets/Countries.csv")
data.head()
```

	country	continent	year	lifeExpectancy	population	gdpPerCapita
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

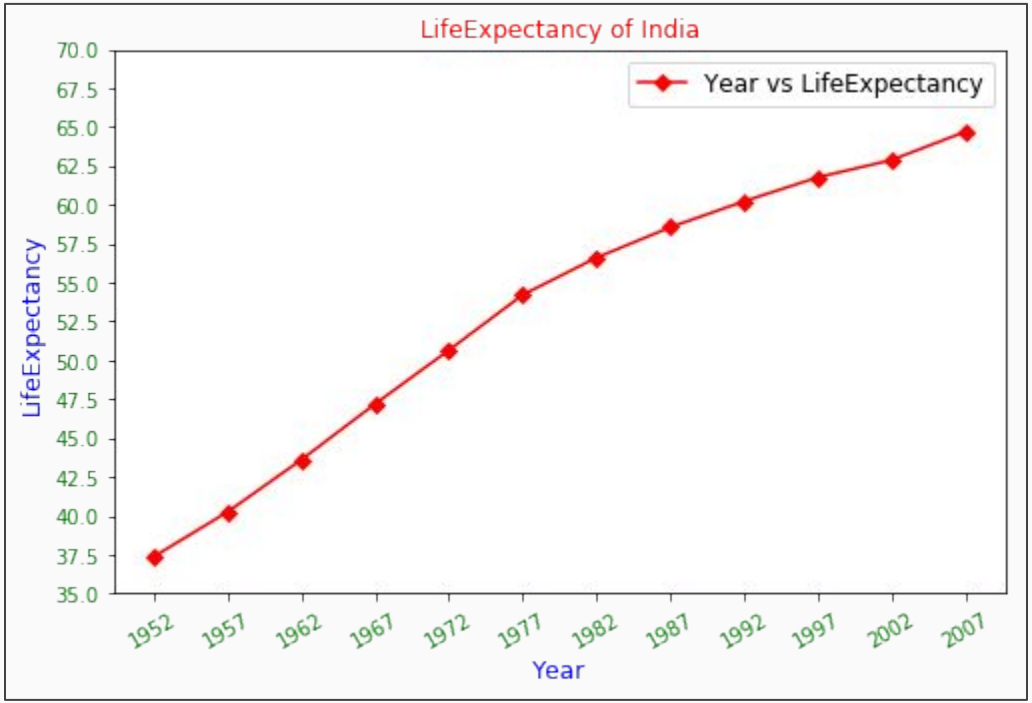
Customization

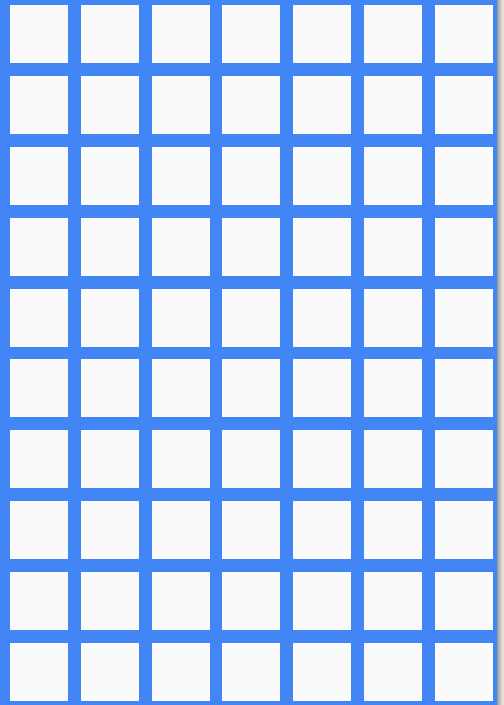
```
plt.figure(figsize=(8, 5))
plt.plot(india.year, india.lifeExpectancy, 'r-D',
     label="Year vs LifeExpectancy")
plt.xlabel('Year', fontsize = 12, color = 'blue')
plt.ylabel('LifeExpectancy', fontsize = 12, color = 'blue')
plt.title('LifeExpectancy of India', fontsize = 12, color = 'red')
plt.legend(fontsize = 12)
plt.xticks(list(india.year), rotation = 30, fontsize = 10, color = 'green')
plt.yticks(np.linspace(35, 70, 15), fontsize = 10, color = 'green')
plt.show()
```

→ fontsize: To set fontsize for xlabel, ylabel, title, legend, xticks, yticks
→ color: To set color for xlabel, ylabel, title, xticks, yticks

Customization

Output:





Styles

Styles

from matplotlib import style → First import style from matplotlib

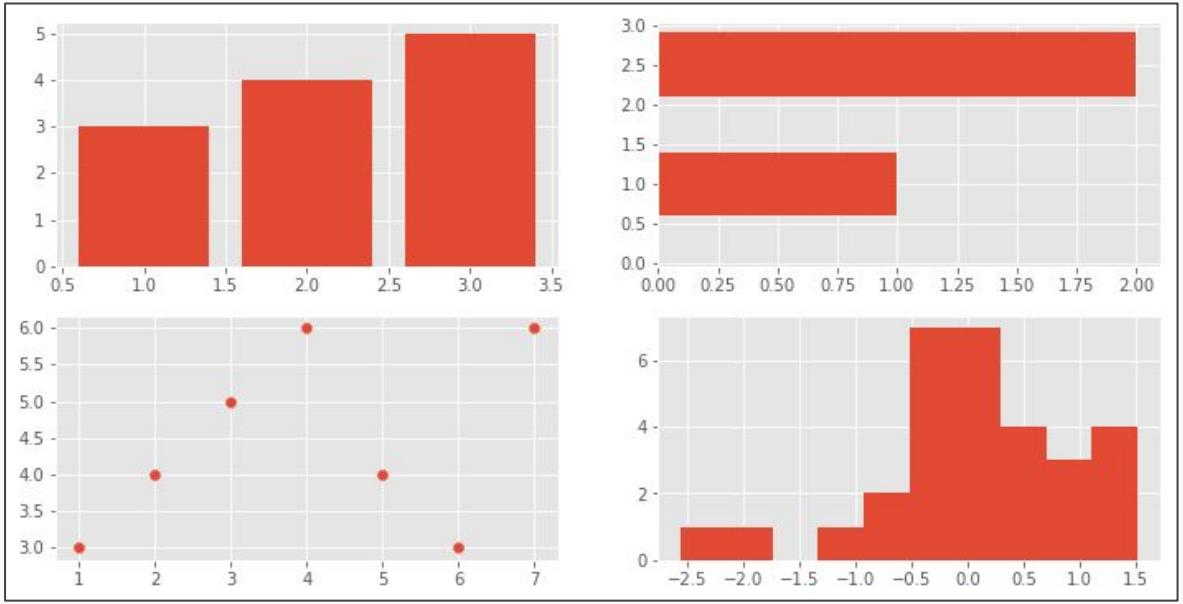
plt.style.available → This will return the list of all available styles

['bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn', 'Solarize_Light2', '_classic_test']

style.use('ggplot') → To use one of the style

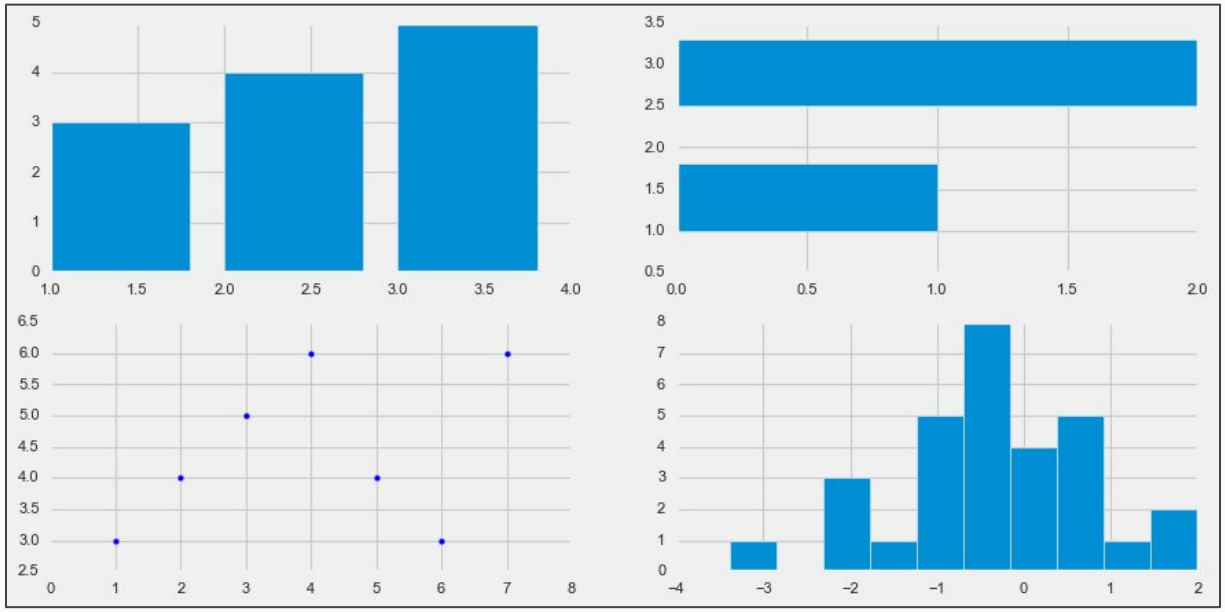
Styles

Output: #Matplotlib Style: ggplot



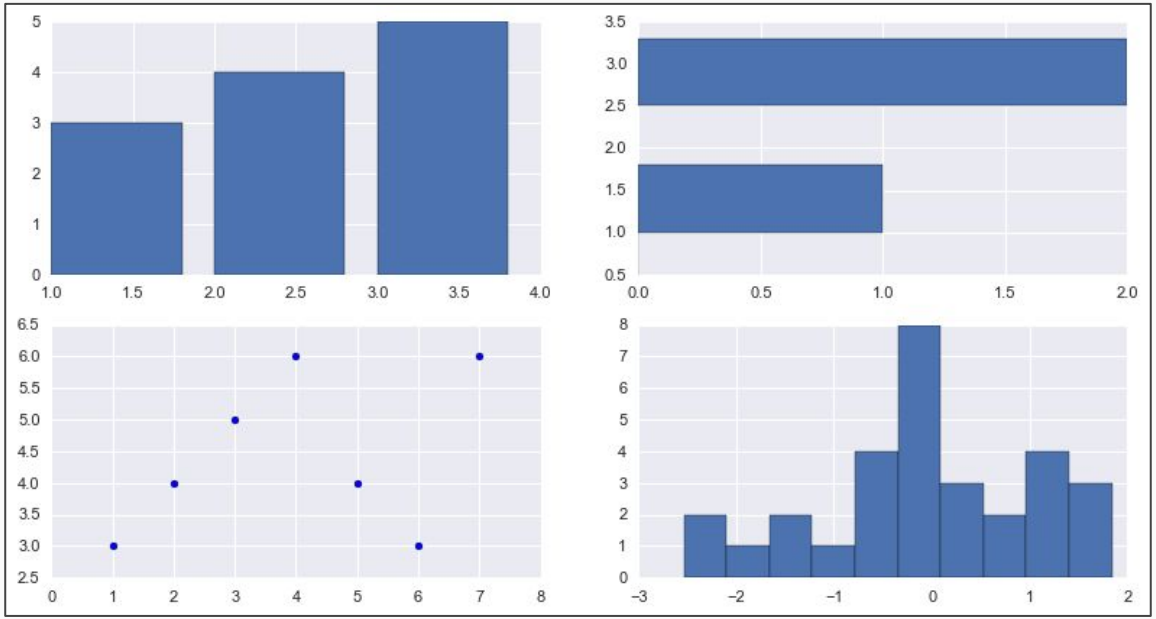
Styles

Output: #Matplotlib Style: fivethirtyeight



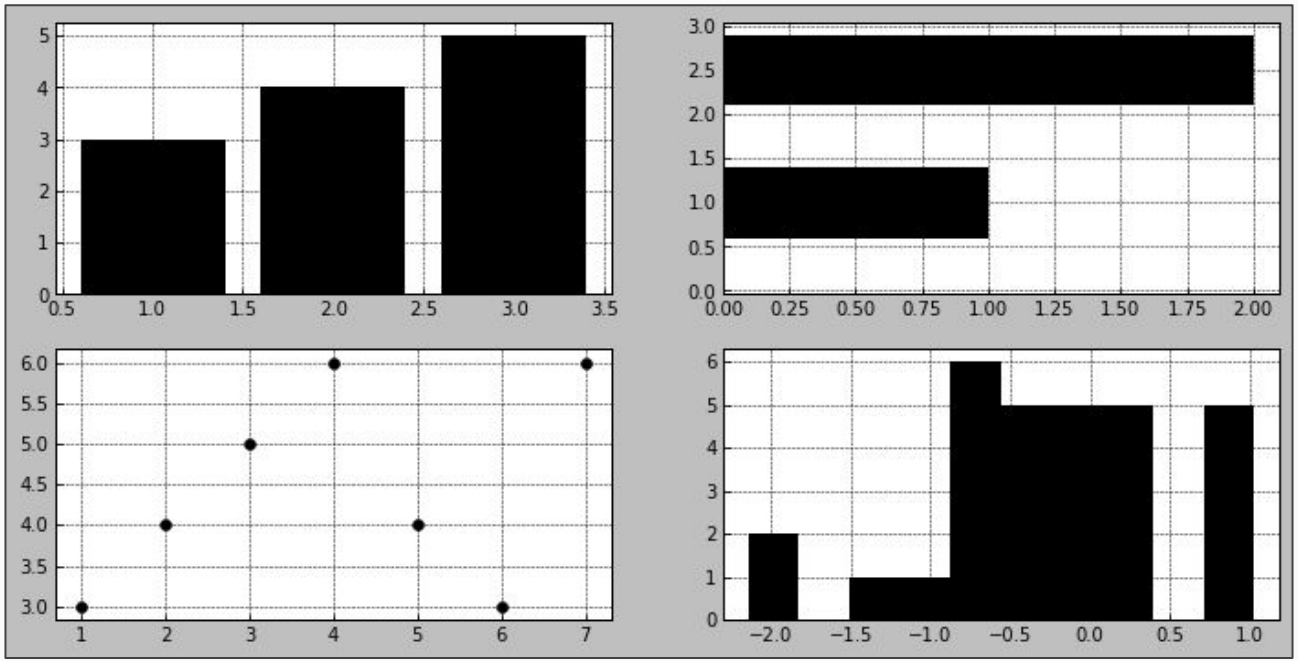
Styles

Output: #Matplotlib Style: seaborn



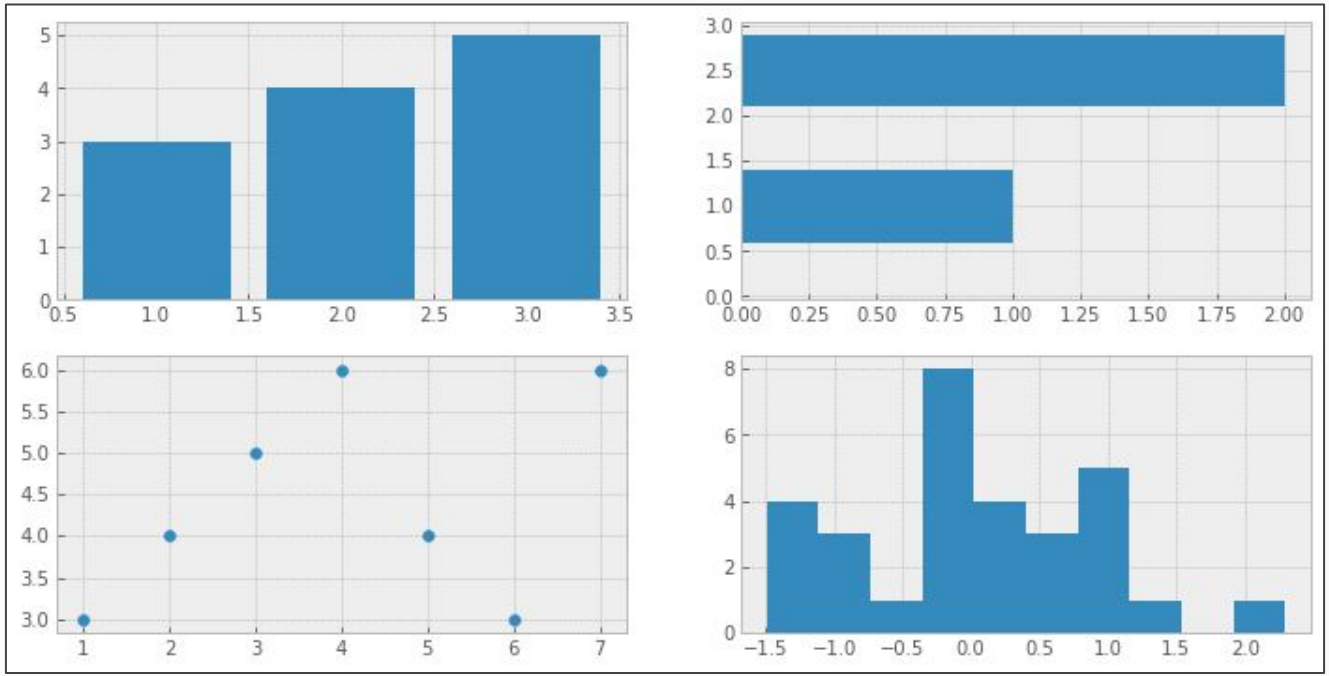
Styles

Output: #Matplotlib Style: grayscale



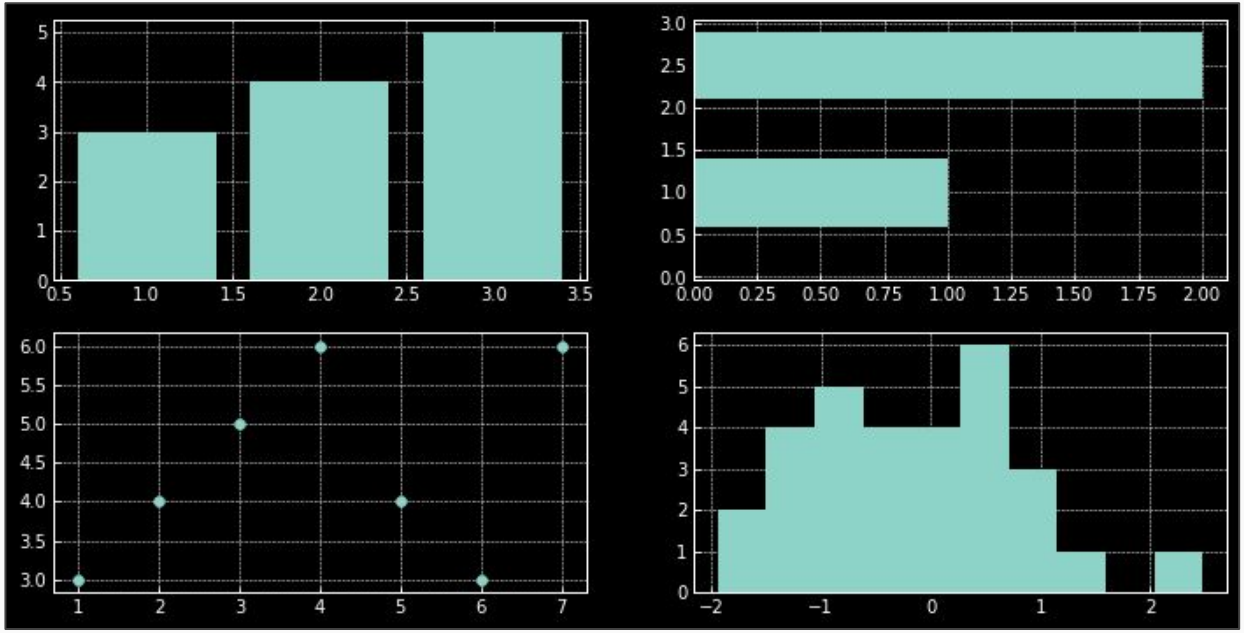
Styles

Output: #Matplotlib Style: bmh



Styles

Output: #Matplotlib Style: dark_background



Pandas Plot() Method

Pandas Plot() Method

Consider the Dataset:

```
data = pd.read_csv("../DataSets/Countries.csv")
data.head()
```

	country	continent	year	lifeExpectancy	population	gdpPerCapita
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

Pandas Plot() Method

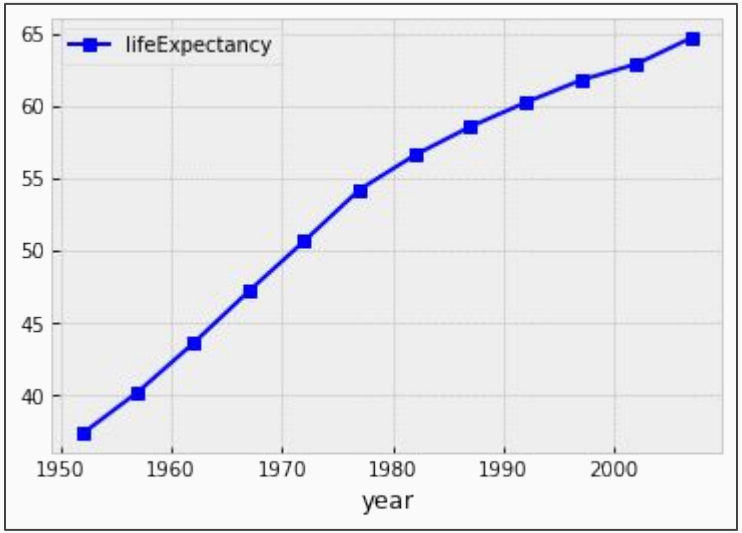
Line Plot:

```
result = data[data.country == 'India']
result.plot.line(x='year', y='lifeExpectancy', color = 'b', marker = 's')
plt.show()
```

 # pandas plot() method is used to plot various graphs
 # plot.line() method is used for Line Plot

Pandas Plot() Method

Output:



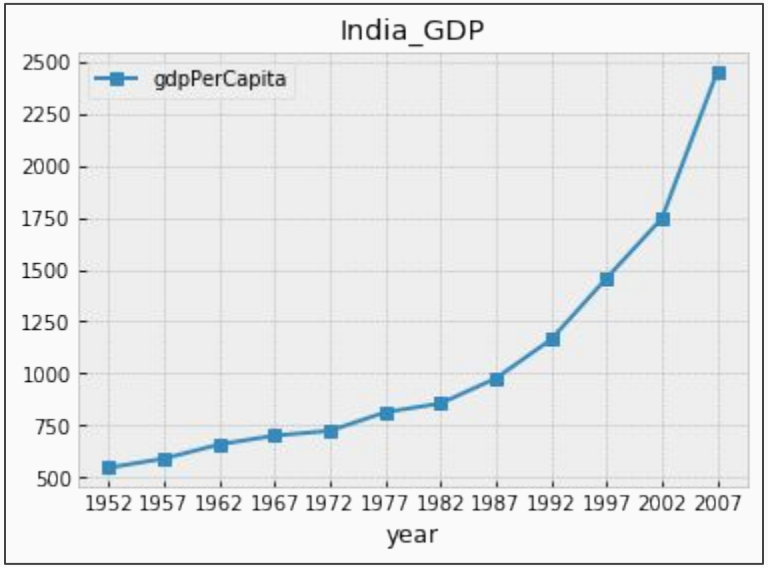
Pandas Plot() Method

Line Plot:

```
result.plot(kind='line', x='year', y='gdpPerCapita', xticks=list(result.year),
marker='s', title='India_GDP')
plt.show()


# pandas plot() method is used to plot various graphs
# Here, kind parameter defines which graph we want to plot
```

Pandas Plot() Method

Output:



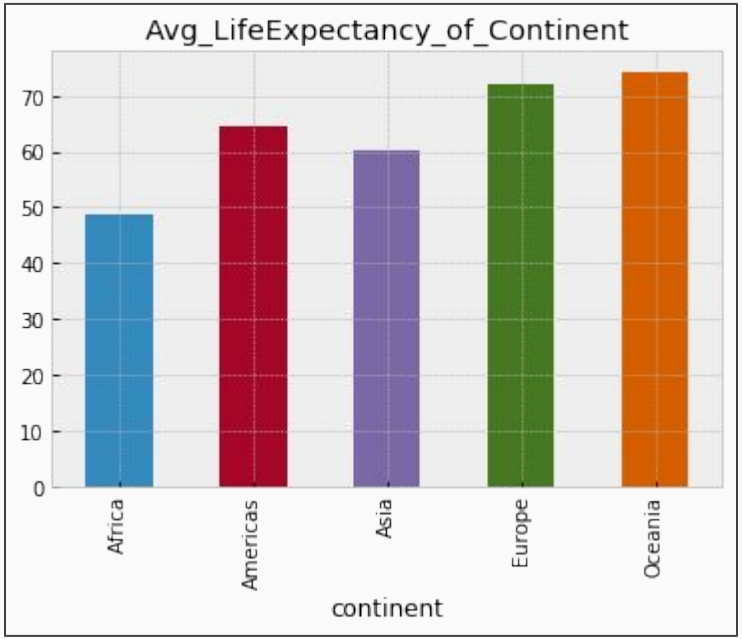
Pandas Plot() Method

Bar Plot:

```
data.groupby("continent")['lifeExpectancy'].mean().plot.bar()
plt.show()

# Alternative:
data.groupby("continent")['lifeExpectancy'].mean().plot(kind='bar',
title='Avg_LifeExpectancy_of_Continent')
plt.show()

# plot.bar() method is used for Bar Plot
```

Pandas Plot() Method

Output:



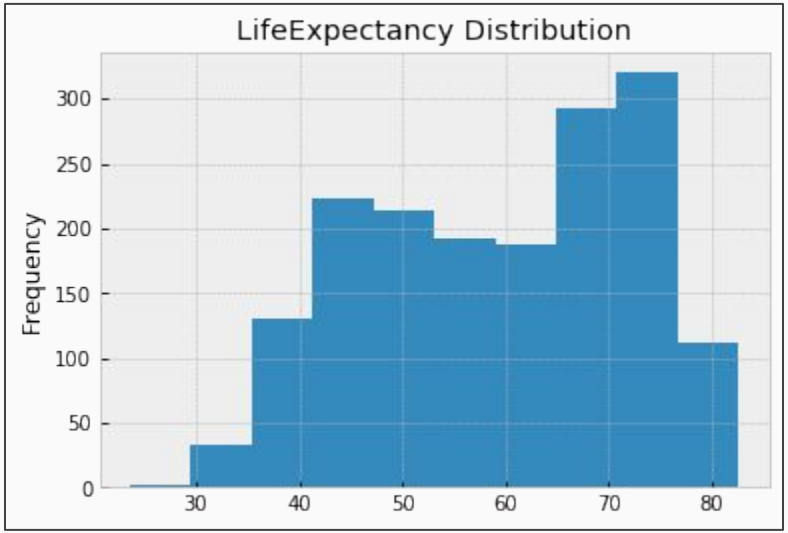
Pandas Plot() Method

Histogram:

```
data.lifeExpectancy.plot.hist()
plt.show()

# Alternative:
data.lifeExpectancy.plot(kind='hist', title='LifeExpectancy Distribution')
plt.show()


# plot.hist() method is used for Histogram
```

Pandas Plot() Method

Output:



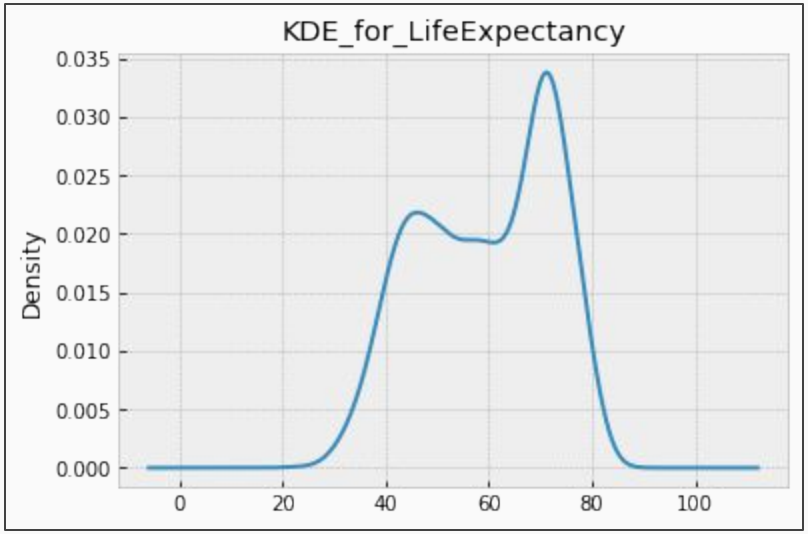
Pandas Plot() Method

KDE (Kernel Density Estimation) Plot:

```
data.lifeExpectancy.plot.kde()
plt.show()

# Alternative:
data.lifeExpectancy.plot(kind='kde', title='KDE_for_LifeExpectancy')
plt.show()


# plot.kde() method is used for KDE Plot
```

Pandas Plot() Method

Output:



Pandas Plot() Method

Consider the Dataset:

```
data = pd.read_csv("../DataSets/IRIS.csv")
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

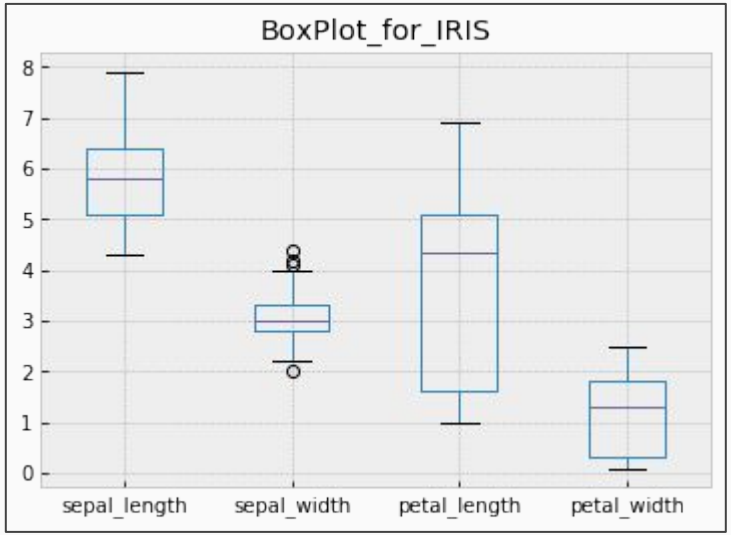
Pandas Plot() Method

Box Plot:

```
data.plot.box()
plt.show()

# Alternative:
data.plot(kind='box', title='BoxPlot_for_IRIS')
plt.show()


# plot.box() method is used for Box Plot
```

Pandas Plot() Method

Output:



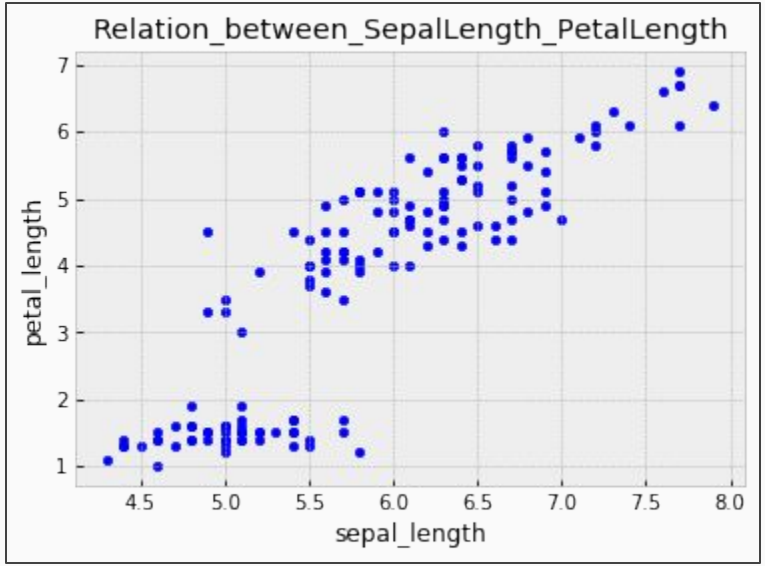
Pandas Plot() Method

Scatter Plot:

```
data.plot.scatter(x='sepal_length', y='petal_length')
plt.show()

# Alternative:
data.plot(kind='scatter', x='sepal_length', y='petal_length',
title='Relation_between_SepalLength_PetalLength')
plt.show()


# plot.scatter() method is used for Scatter Plot
```

Pandas Plot() Method

Output:



Annotations and Text

Annotations and Text

Consider the Dataset:

```
company = ['Infosys', 'TCS', 'HCL',
'Wipro', 'Google', 'FB', 'Amazon', 'Twitter']

df = pd.DataFrame({'Revenue (in
$million)': [200, 240, 300, 140, 600, 770,
250, 450]}, index=company)

df
```

	Revenue (in $million)
Infosys	200
TCS	240
HCL	300
Wipro	140
Google	600
FB	770
Amazon	250
Twitter	450

Annotations and Text

```
y = df.values
y = y.flatten()
xpos = np.arange(len(company))
plt.bar(xpos, y, label = 'Revenue')
plt.xticks(xpos, company)
plt.ylabel('Company Revenue in $million')
plt.title('Bar Chart')
plt.legend()

plt.annotate('FB is KING', (5, 770), xytext=(0.8, 0.7), textcoords='axes
fraction', arrowprops = dict(arrowstyle='fancy', color='black'))

plt.show()
```

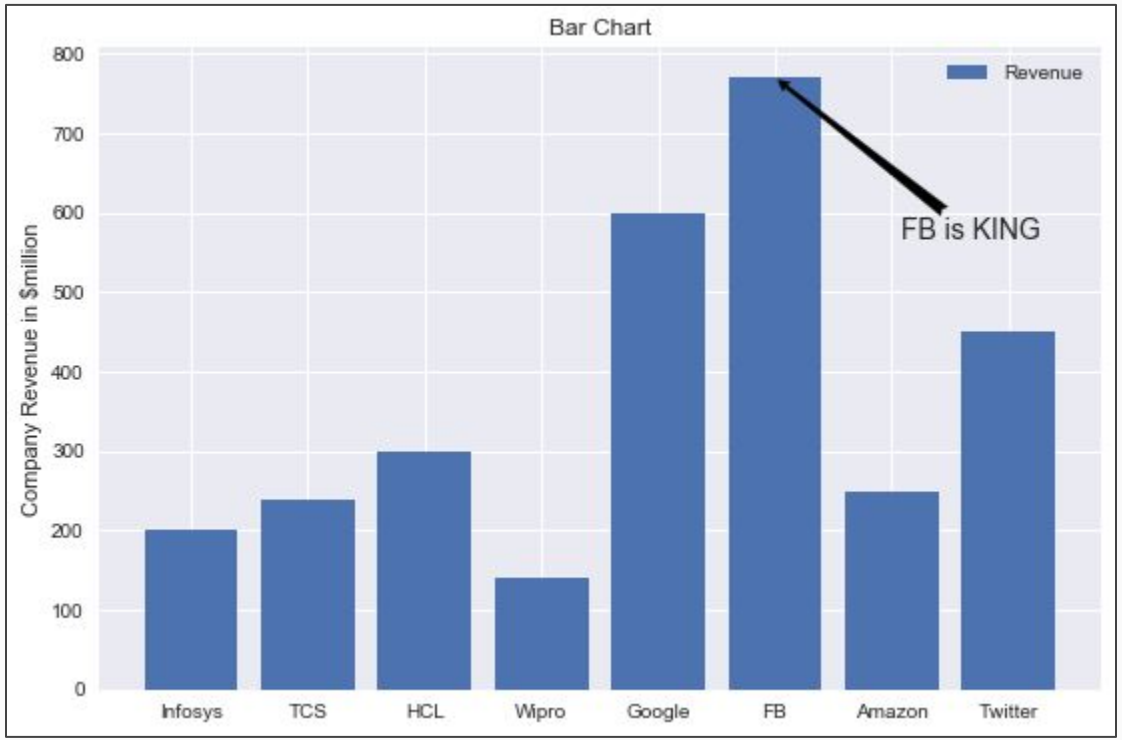
Annotations and Text

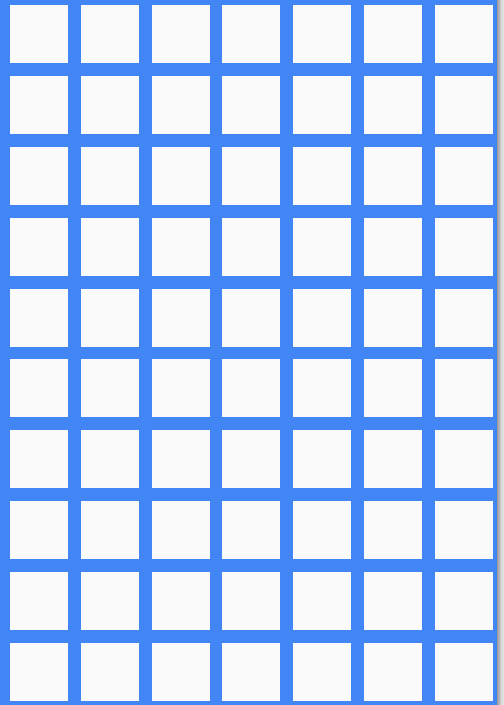
```
plt.annotate('FB is KING', (5, 770), xytext=(0.8, 0.7), textcoords='axes fraction', arrowprops = dict(arrowstyle='fancy', color='black'))
```

→ annotate(): To annotate the point (x, y) with text ‘s’
→ s: str - The text of the annotation
→ (x, y): Point to annotate
→ xytext: Annotation text placement
→ arrowprops: Dictionary containing arrow properties

Annotations and Text

Output:





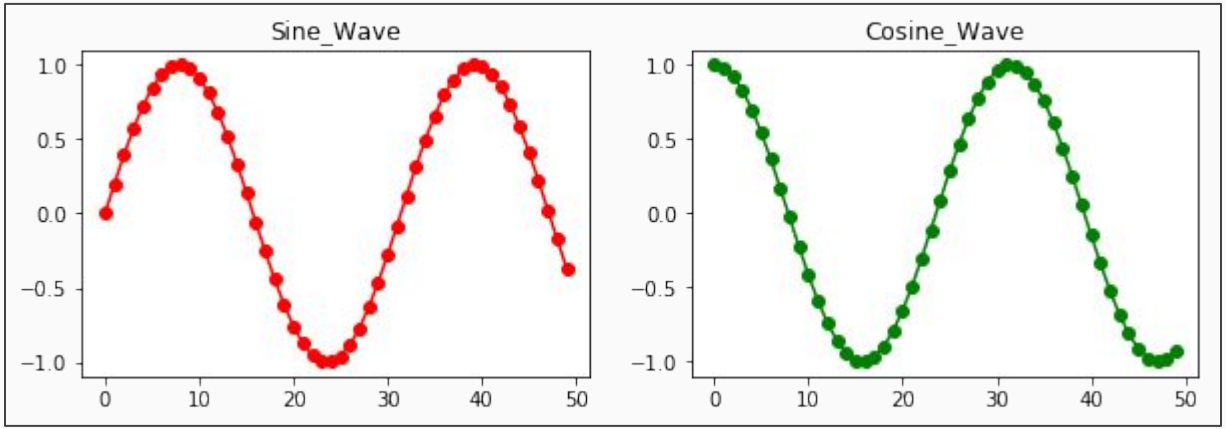
Subplots

Subplots

```
# Method 1:
fig, axes = plt.subplots(1, 2, figsize=(10,3))
x = np.arange(0, 10, 0.2)

# 1st Plot
axes[0].plot(np.sin(x), 'r-o')
axes[0].set_title('Sine_Wave')

# 2nd Plot
axes[1].plot(np.cos(x), 'g-o')
axes[1].set_title('Cosine_Wave')

plt.show()
```

Subplots

Output:

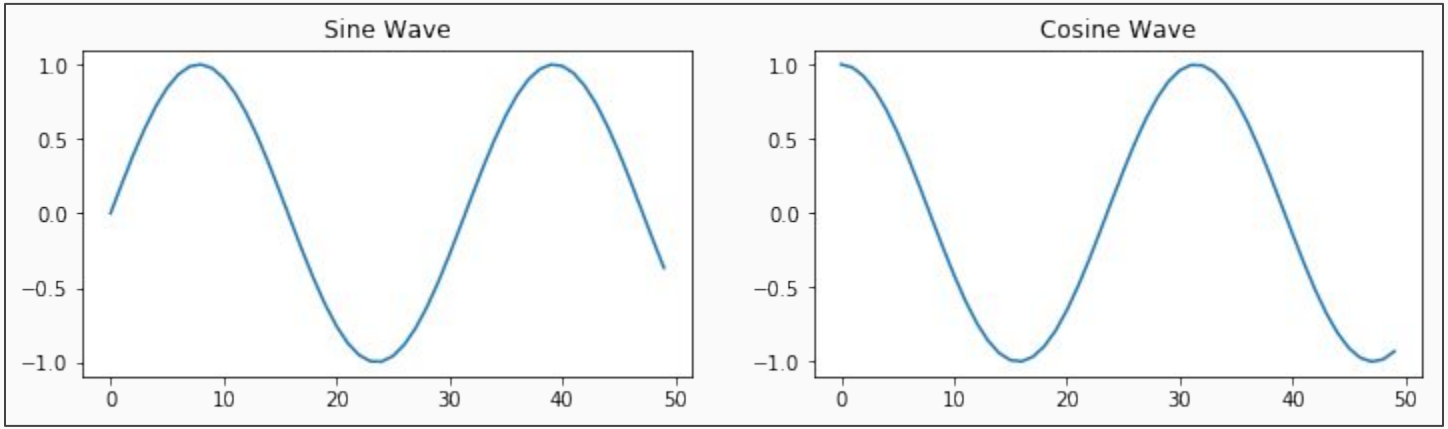


Subplots

```
# Method 2:
plt.figure(figsize=(12, 3)) → figure: Creates a new figure
x = np.arange(0, 10, 0.2)

ax1 = plt.subplot(121) → plt.subplot(): Creates a subplot
ax1.plot(np.sin(x))
ax1.set_title('Sine Wave')

ax2 = plt.subplot(122)
ax2.plot(np.cos(x))
ax2.set_title('Cosine Wave')

plt.show()
```

Subplots

Output:



Subplots

 # Method 3:

fig = plt.figure() → figure: Creates a new figure

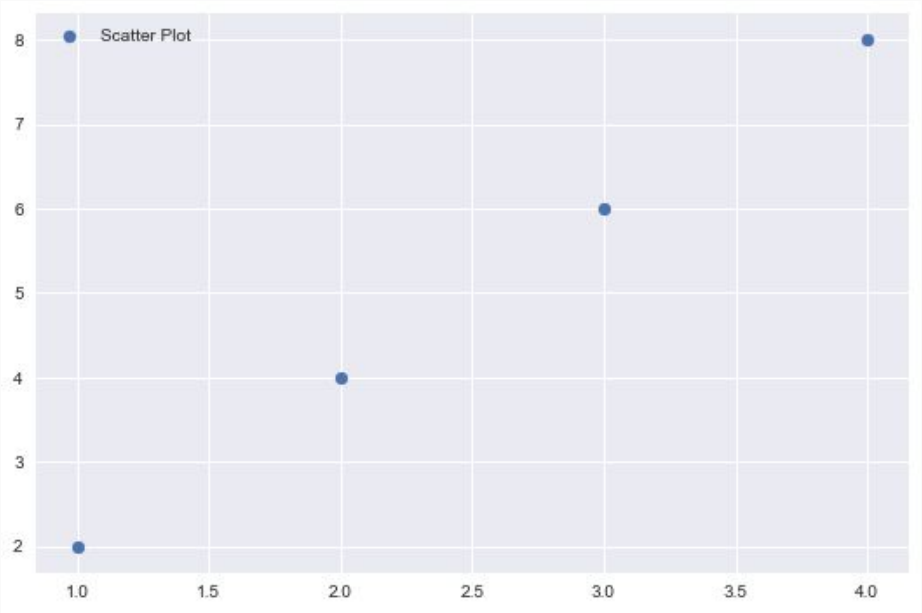
ax = fig.add_subplot(1, 1, 1) → add_subplot(): To add a Subplot

→ 1, 1, 1: (I, J, K) → Kth plot on a grid with I Rows and J Columns i.e. 1st plot on grid with 1 row and 1 column

```
x1 = [1, 2, 3, 4]; y1 = [2, 4, 6, 8]
x2 = [2, 4, 6, 8]; y2 = [1, 2, 3, 4]
ax.scatter(x1, x2, label='Scatter Plot')
ax.legend()
plt.show()
```

Subplots

Output:

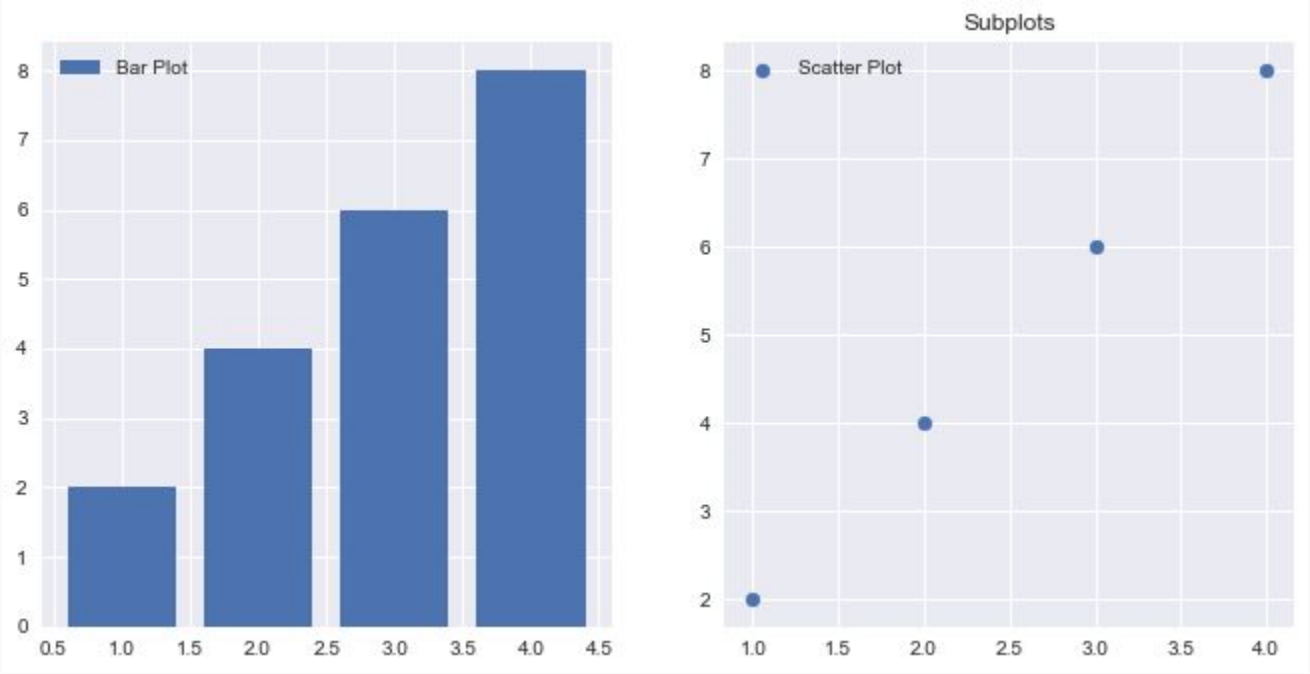


Subplots

```
fig = plt.figure(figsize=(10, 5)) → figsize: To set the figsize
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

x1 = [1, 2, 3, 4]
x2 = [2, 4, 6, 8]

ax1.bar(x1, x2, label='Bar Plot')
ax2.scatter(x1, x2, label='Scatter Plot')
ax1.legend()
ax2.legend()
plt.title('Subplots')
plt.show()
```

Subplots

Output:

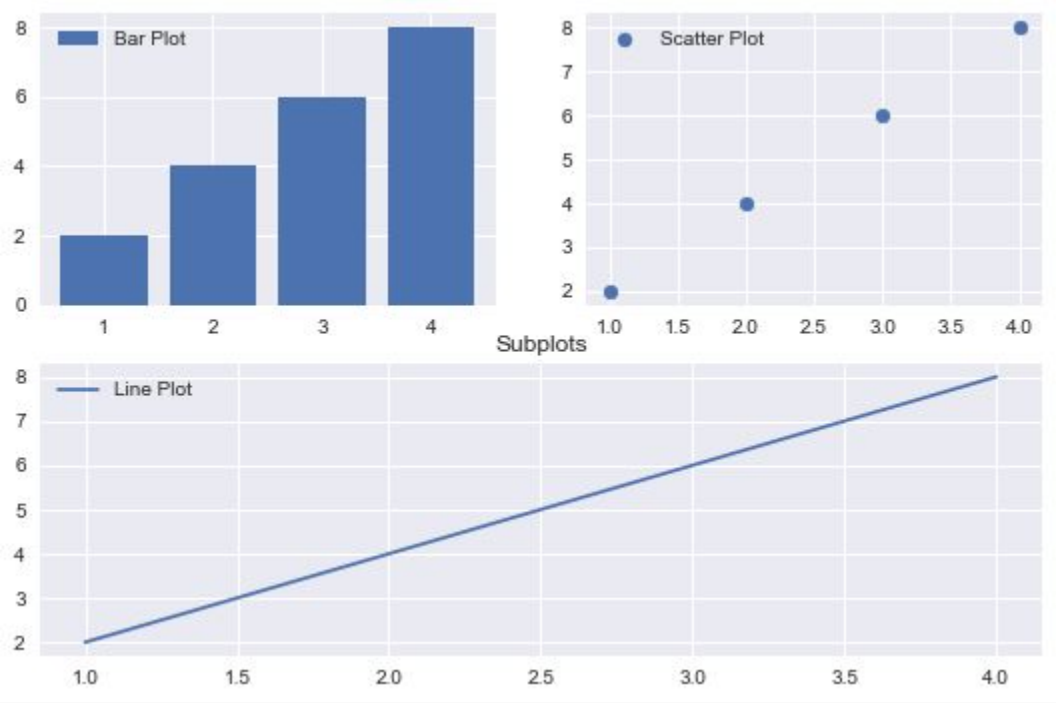


Subplots

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 1, 2)
x1 = [1, 2, 3, 4]; x2 = [2, 4, 6, 8]
ax1.bar(x1, x2, label=Bar Plot')
ax2.scatter(x1, x2, label='Scatter Plot')
ax3.plot(x1, x2, label='Line Plot')
ax1.legend()
ax2.legend()
ax3.legend()
plt.title('Subplots')
plt.show()
```

Subplots

Output:



Subplots

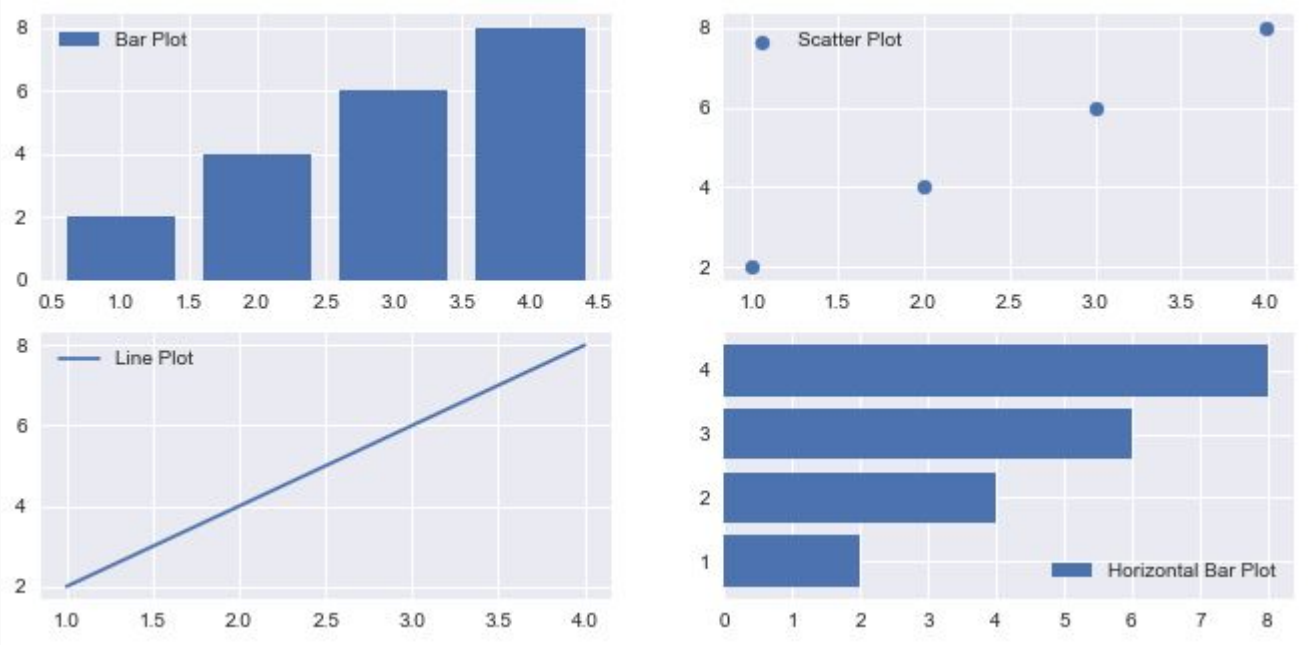
```
fig = plt.figure()
fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)
x1 = [1, 2, 3, 4]; x2 = [2, 4, 6, 8]
ax1.bar(x1, x2, label='Bar Plot')
ax2.scatter(x1, x2, label='Scatter Plot')
ax3.plot(x1, x2, label='Line Plot')
ax4.barh(x1, x2, label='Horizontal Bar Plot')
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
plt.show()
```

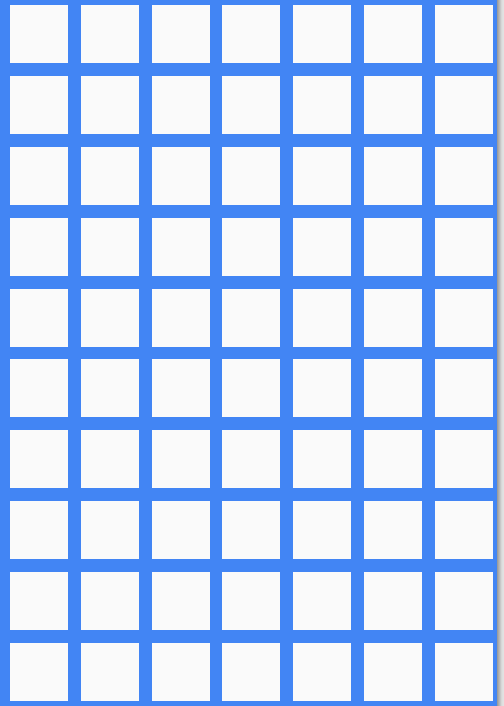
Subplots

```
fig = plt.figure()
fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(221) → This is an another equivalent way
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)
x1 = [1, 2, 3, 4]; x2 = [2, 4, 6, 8]
ax1.bar(x1, x2, label='Bar Plot')
ax2.scatter(x1, x2, label='Scatter Plot')
ax3.plot(x1, x2, label='Line Plot')
ax4.barh(x1, x2, label='Horizontal Bar Plot')
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
plt.show()
```

Subplots

Output:





Custom Stuffs

Custom Stuffs

We can do Lot more Custom Stuffs in Matplotlib including:

- Custom Grids
- Step Chart and Fill Between Chart
- Custom Text to Each Graph
- Twin Axes

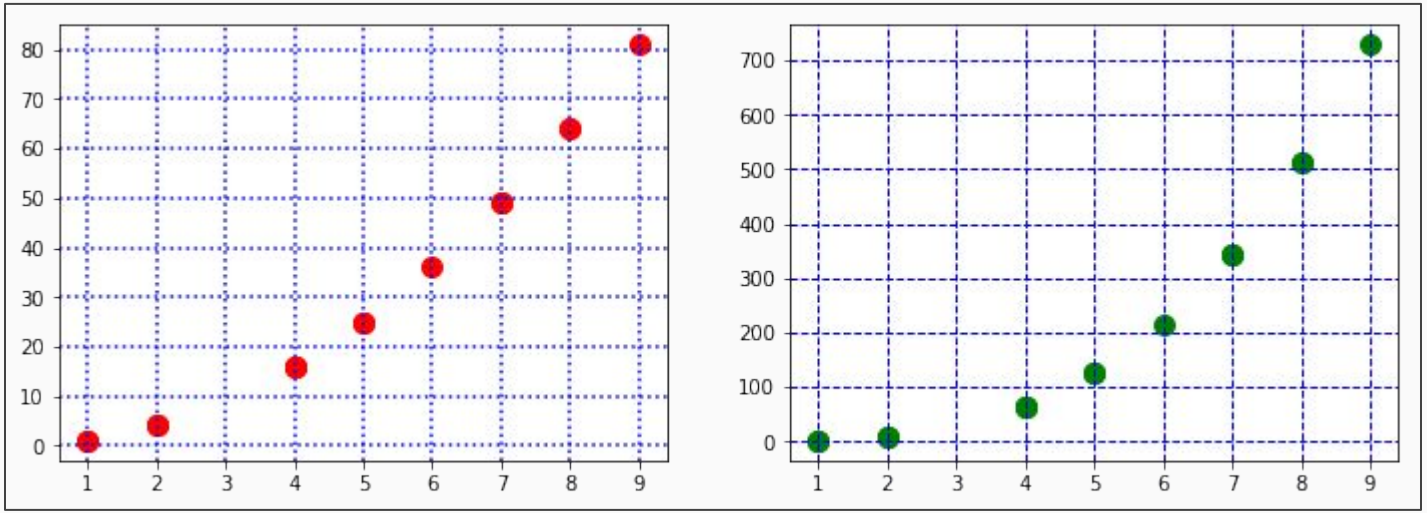
Custom Stuffs: Custom Grids

```
fig = plt.figure(figsize=(12,4))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
x = np.random.randint(1, 10, size=[20])

ax1.scatter(x, x**2, c='r', s=100)
ax1.grid(color='b', linewidth=1.5, linestyle=':')
ax2.scatter(x, x**3, c='g', s=100)
ax2.grid(color='b', linewidth=1, linestyle='--')

plt.show()
```

→ grid(): To set custom color, linewidth and linestyle for Grids

Custom Stuffs: Custom Grids

Output:



Custom Stuffs: Step Chart and Fill Between Chart

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)

n = np.arange(1,10)
ax1.step(n, n**2)
```

→ step(): To create a step chart

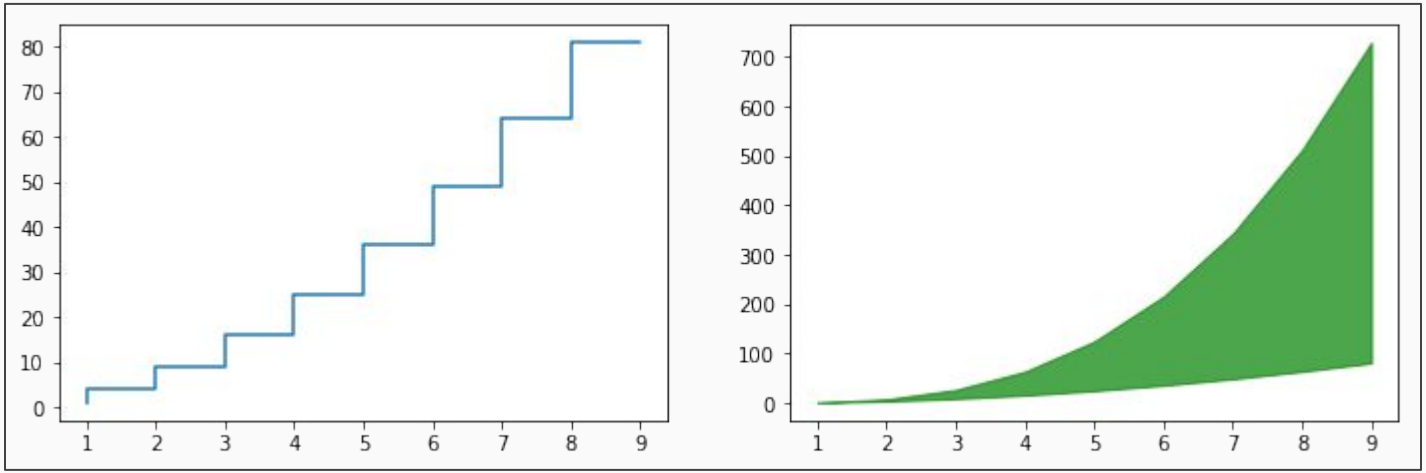
```
ax2.fill_between(n, n**2, n**3, color='g', alpha=0.7)
```

→ fill_between(): To create a fill between chart, here color gives the color to fill between chart and alphas stands for transparancy level

```
plt.show()
```

Custom Stuffs: Step Chart and Fill Between Chart

Output:



Custom Stuffs: Custom Text

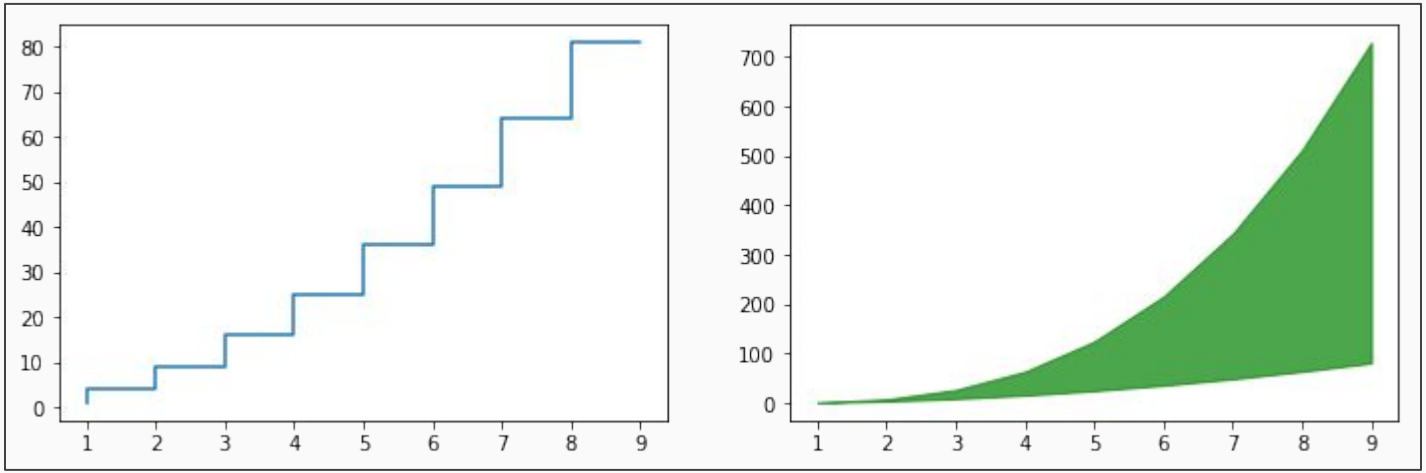
```
plt.figure(figsize=(6,4))
n = np.arange(1,10)

plt.plot(n, n**2, n, n**3)

plt.text(7, 80, r"$y=x^2$", fontsize=20, color="blue")
plt.text(5.5, 400, r"$y=x^3$", fontsize=20, color="orange")
```

→ text(): To give the Text for the Graph

```
plt.show()
```

Custom Stuffs: Custom Text

Output:

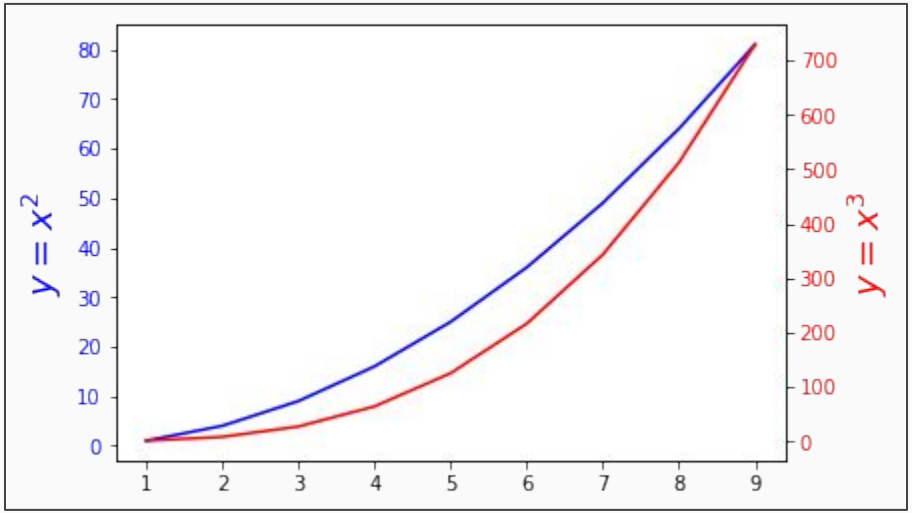


Custom Stuffs: Twin Axes

```
fig, ax1 = plt.subplots()
ax1.plot(n, n**2, color='b')
ax1.set_ylabel(r"$y=x^2$", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx() → twinx(): To Create a Twin X axes (twiny() for Y axes)
ax2.plot(n, n**3, color='r')
ax2.set_ylabel(r"$y=x^3$", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")

plt.show()
```

Custom Stuffs: Twin Axes

Output:



End

Thanks!

Signitive Technologies

Sneh Nagar, Behind ICICI Bank,
Chhatrapati Square, Nagpur 15

Landmark:
Bharat Petrol Pump
Chhatrapati Square

Contact: 9011033776
www.signitivetech.com

“Keep Learning, Happy Learning”

Best Luck!

Have a Happy Future