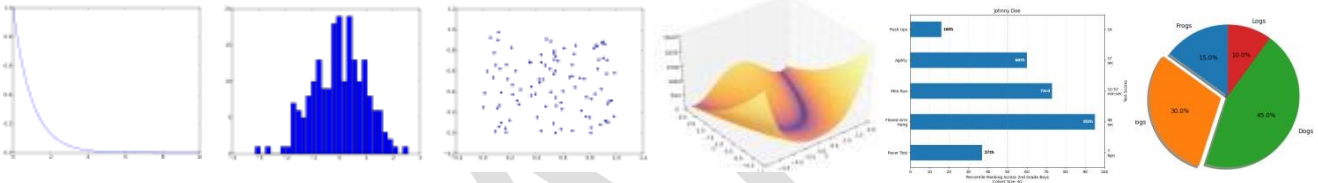
UNIT 6 DATA VISUALIZATION

Introduction to Matplotlib.

- Most people visualize information better when they see it in graphic versus textual format.
- Graphics help people see relationships and make comparisons with greater ease.
- Fortunately, Python makes the task of converting textual data into graphics relatively easy using libraries, one of the most commonly used library for this is Matplotlib.
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, bar charts, scatterplots, etc., with just a few lines of code.
- A Graph or chart is simply a visual representation of numeric data, Matplotlib makes a large number of graph and chart types.
- Some of graphs/charts are shown below,



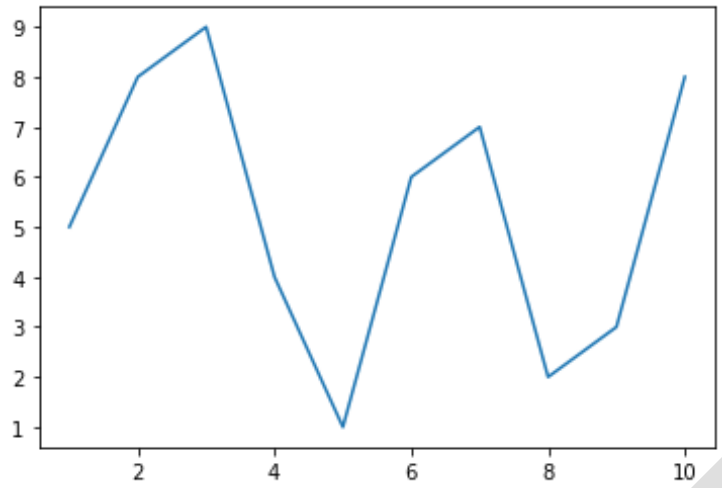
Line Chart | Histogram | Scatter Plot | 3D Plot | Bar Chart | Pie Chart

Line chart

- A line chart (also known as a line plot or line graph) is a graph that uses lines to connect individual data points that display quantitative values over specified x-axis values.
- Line graphs use data point "**markers**" that are connected by straight lines to aid in visualization.
- In finance, line graphs are the most frequently used visual representation of values over time.
- To create a line chart in Matplotlib we need some values and the **matplotlib.pyplot** module.
- Example:

```
import matplotlib.pyplot as plt
%matplotlib inline
values = [5,8,9,4,1,6,7,2,3,8]
plt.plot(range(1,11),values)
plt.show()
```

Output:

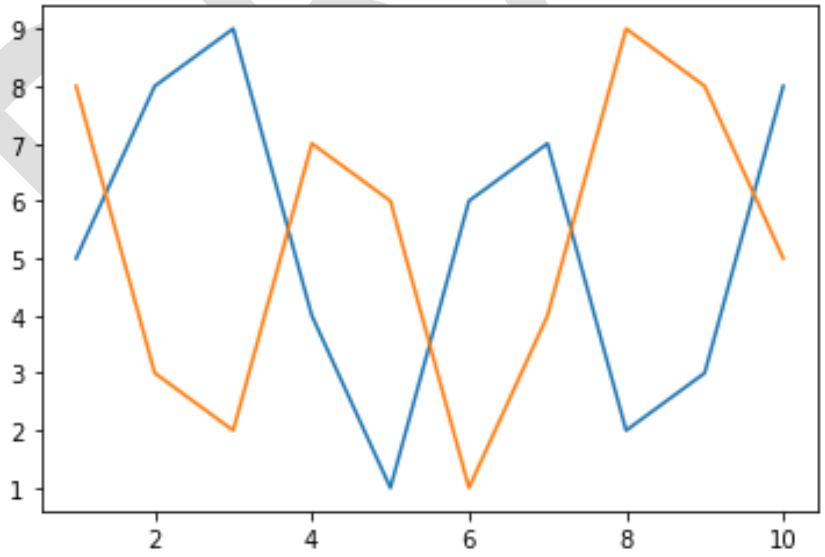


Drawing multiple lines

- We can draw multiple lines in a plot by making multiple plt.plot() calls.
- Example:

```
import matplotlib.pyplot as plt
%matplotlib inline
values1 = [5,8,9,4,1,6,7,2,3,8]
values2 = [8,3,2,7,6,1,4,9,8,5]
plt.plot(range(1,11),values1)
plt.plot(range(1,11),values2)
plt.show()
```

Output:



Saving/Exporting Graphs

- We can export/save our plots on a drive using savefig() method.

- Example:

```
import matplotlib.pyplot as plt
%matplotlib inline
values1 = [5,8,9,4,1,6,7,2,3,8]
values2 = [8,3,2,7,6,1,4,9,8,5]
plt.plot(range(1,11),values1)
plt.plot(range(1,11),values2)
#plt.show()
plt.savefig('SaveToPath.png',format='png')
```

Output: stored image file named *SaveToPath.png*

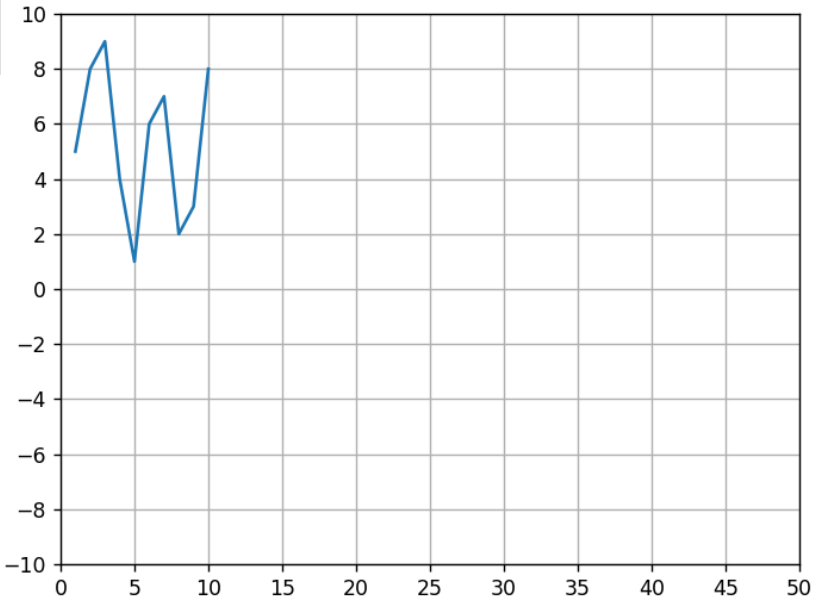
Note: some of the possible values for the *format* is png, svg, pdf etc…

Axis, Ticks and Grid

- We can access and format the axis, ticks, and grid on the plot using the axis() method of the matplotlib.pyplot.plt
- Example:

```
import matplotlib.pyplot as plt
%matplotlib notebook
values = [5,8,9,4,1,6,7,2,3,8]
ax = plt.axes()
ax.set_xlim([0,50])
ax.set_ylim([-10,10])
ax.set_xticks([0,5,10,15,20,25,30,35,40,45,50])
ax.set_yticks([-10,-8,-6,-4,-2,0,2,4,6,8,10])
ax.grid()
plt.plot(range(1,11),values)
```

Output:

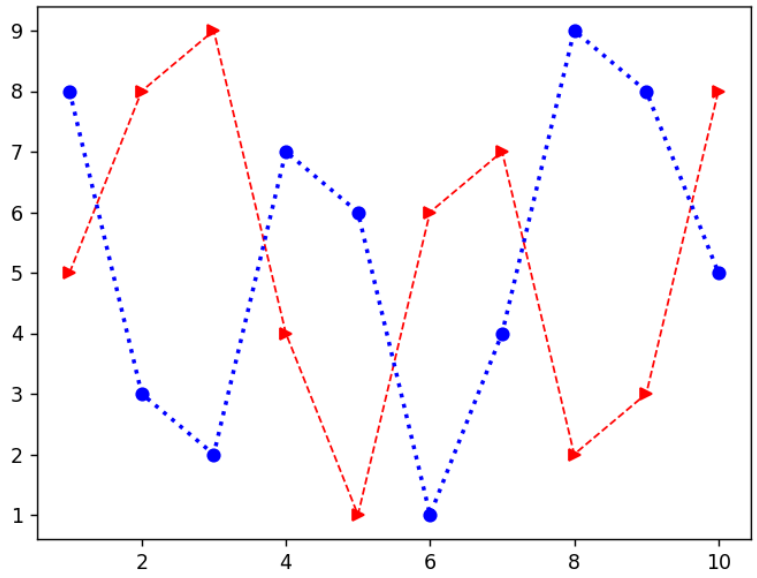


Line Appearance

- We need different line styles to differentiate when having multiple lines in the same plot, we can achieve this using many parameters, some of them are listed below.
  - Line style (*linestyle* or *ls*)
  - Line width (*linewidth* or *lw*)
  - Line color (*color* or *c*)
  - Markers (*marker*)
- Example:

```
import matplotlib.pyplot as plt
%matplotlib inline
values1 = [5,8,9,4,1,6,7,2,3,8]
values2 = [8,3,2,7,6,1,4,9,8,5]
plt.plot(range(1,11),values1,c='r',lw=1,ls='--',marker='>')
plt.plot(range(1,11),values2,c='b',lw=2,ls=':',marker='o')
plt.show()
```

Output:



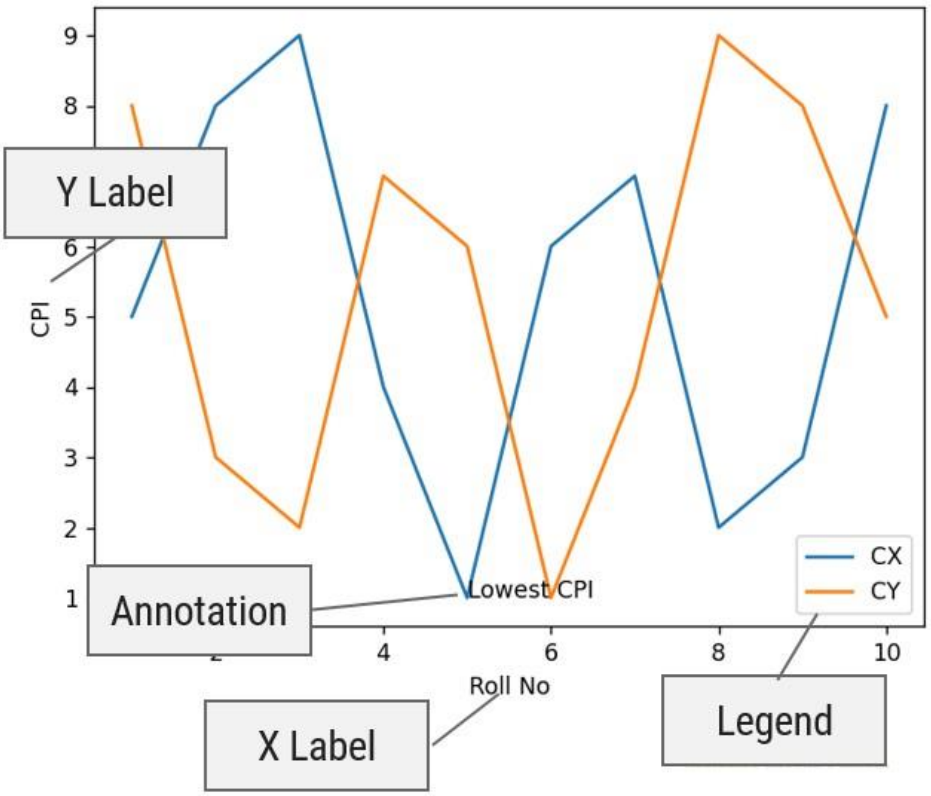
- Possible Values for each parameter are,

Values	Line Style		Values	Colors		Values	Markers
'-'	Solid line		'b'	Blue		'.'	Point
'--'	Dashed line		'g'	Green		','	Pixel
'-.'	Dash-dot line		'r'	Red		'o'	Circle
':'	Dotted line		'c'	Cyan		'v'	Triangle down
			'm'	Magenta		'^'	Triangle up
			'y'	Yellow		'>'	Triangle right
			'k'	Black		'<'	Triangle left
			'w'	White		'*'	Star

			'+'	Plus
			'x'	X
			Etc....	

Labels, Annotation and Legends

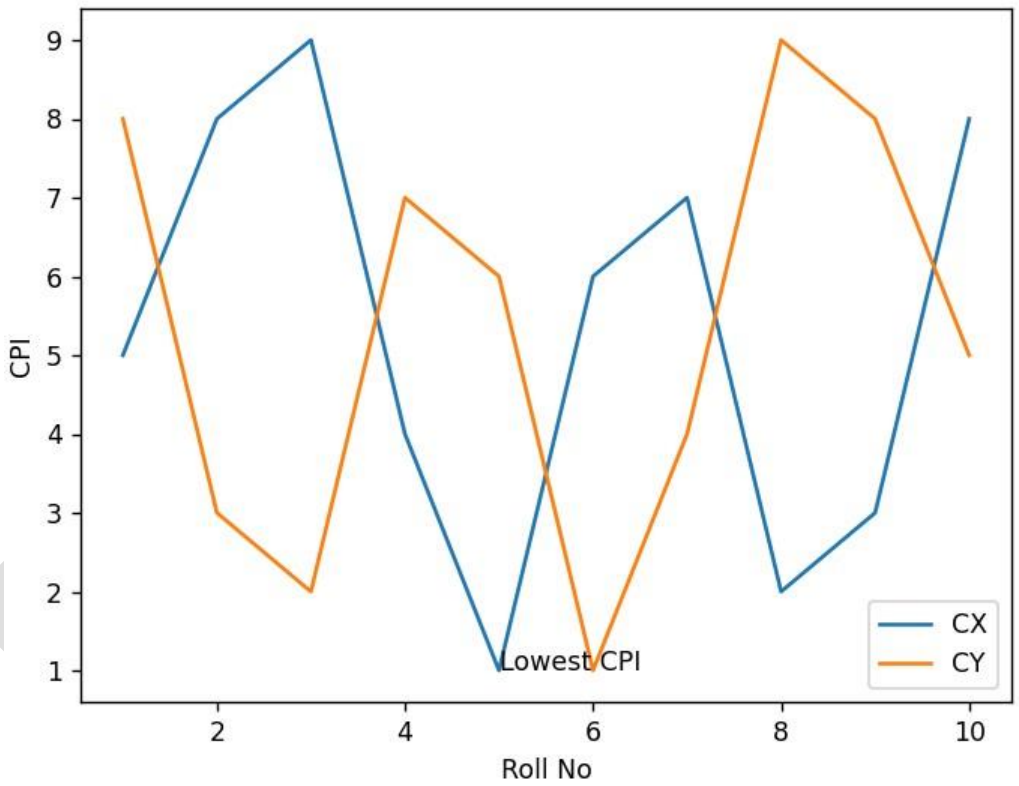
- To fully document our graph, we have to resort the labels, annotation, and legends.
- Each of this element have a different purpose as follows,
  - **Label:** provides identification of a particular data element or grouping, it will make it easy for the viewer to know the name or kind of data illustrated.
  - **Annotation:** augments the information the viewer can immediately see about the data with notes, sources, or other useful information.
  - **Legend:** presents a listing of the data groups within the graph and often provides cues (such as line type or color) to identify the line with the data.



- Example:

```
import matplotlib.pyplot as plt
%matplotlib inline
values1 = [5,8,9,4,1,6,7,2,3,8]
values2 = [8,3,2,7,6,1,4,9,8,5]
plt.plot(range(1,11),values1)
plt.plot(range(1,11),values2)
plt.xlabel('Roll No')
plt.ylabel('CPI')
plt.annotate(xy=[5,1],s='Lowest CPI')
plt.legend(['CX','CY'],loc=4)
plt.show()
```

Output:



Choosing the Right Graph

The kind of graph we choose determines how people view the associated data, so choosing the right graph from the outset is important.

we should choose,

- **Pie Chart**, if we want to show how various data elements **contribute towards a whole.**
- **Bar Chart**, if we want to **compare data elements.**
- **Histograms**, if we want to show the **distribution of elements.**

- **Box Plot**, if we want to **depict groups in elements**.
- **Scatter Plot**, if we want to **find patterns in data**.
- **Line Chart,** if we want to display **trends over time.**
- **Basemap,** if we want to display **geographical data.**
- **NetworkX,** if we want to **display network.**

Pie Chart

A pie chart (or a circle chart) is a circular statistical graphic, which is divided into slices to illustrate numerical proportion.

In a pie chart, the arc length of each slice is proportional to the quantity it represents.

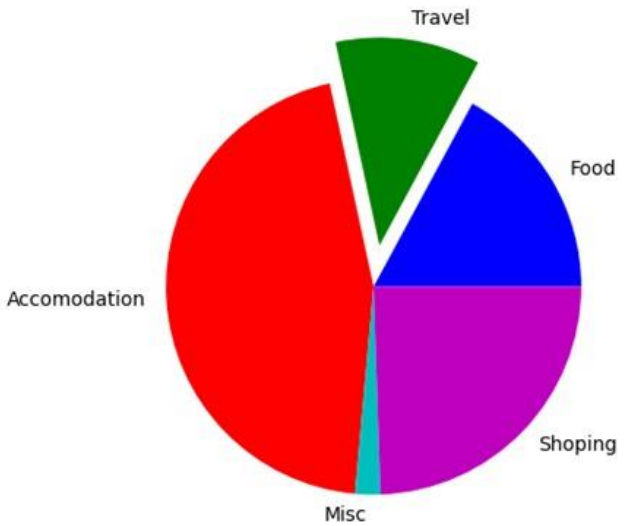
The pie chart focus on showing parts of a whole, the entire pie would be 100 percentages, the question is how much of that percentage each value occupies.

To draw a pie chart, we need to import Matplotlib library, and use its method named pie with the values and optional parameters like colors, labels, and explode.

Example:

```
import matplotlib.pyplot as plt
%matplotlib notebook
values = [305,201,805,35,436]
l = ['Food','Travel','Accomodation','Misc','Shoping']
c = ['b','g','r','c','m']
e = [0,0.2,0,0,0]
plt.pie(values,colors=c,labels=l,explode=e)
plt.show()
```

Output:



plotted vertically or horizontally. A vertical bar chart is sometimes called a column chart.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

Bar charts make comparing values easy, wide bars and segregated measurements emphasize the difference between values, rather than the flow of one value to another as a line graph.

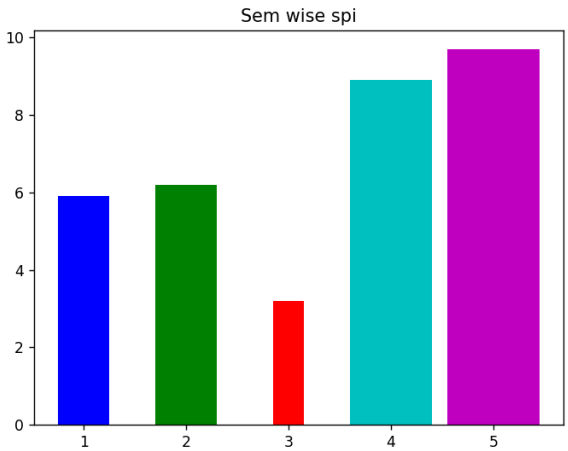
To draw a bar chart, we need to import Matplotlib library and use its bar/barh method with required parameters x and y, we can also supply other optional parameters like color, label, and width.

If we want to draw a vertical bar chart we can use **bar** method, and if we want to draw a horizontal bar chart we can use **barh** method.

Example:

```
import matplotlib.pyplot as plt
%matplotlib notebook
x = [1,2,3,4,5]
y = [5.9,6.2,3.2,8.9,9.7]
l = ['1st','2nd','3rd','4th','5th']
c = ['b','g','r','c','m']
w = [0.5,0.6,0.3,0.8,0.9]
plt.title('Sem wise spi')
plt.bar(x,y,color=c,label=l,width=w)
plt.show()
```

Output:



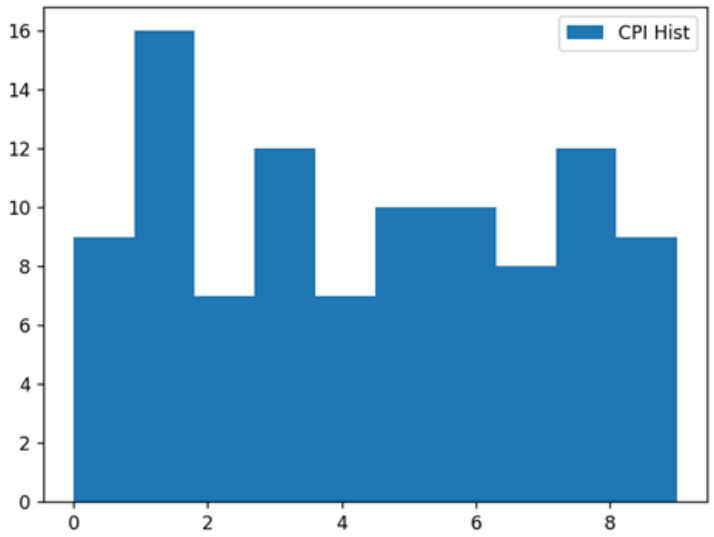
or “bucket”, and then count how many values fall into each interval.

To draw a histogram in python, we can use hist method of Matplotlib library with the required x parameter and many optional parameters like bins, histtype, align, label etc..

Example:

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib notebook
cpis = np.random.randint(0,10,100)
plt.hist(cpis,bins=10, histtype='stepfilled',align='mid',label='CPI
Hist')
plt.legend()
plt.show()
```

Output:



Boxplot

Boxplots provide a means of depicting groups of numbers through their quartiles, Quartiles means three points (25^{th} percentile, median, 75^{th} percentile) dividing a group into four equal parts.

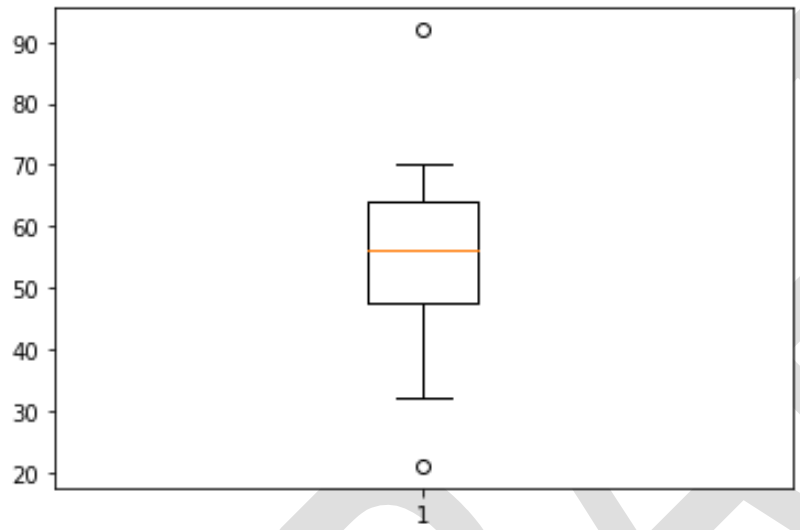
Boxplot basically used to detect **outliers** in the data, let’s see an example where we need boxplot.

We have a dataset where we have time taken to check the paper, and we want to find the faculty which either takes more time or very little time to check the paper.

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
timetaken = pd.Series([50,45,52,63,70,21,56,68,54,57,35,62,65,92,32])
plt.boxplot(timetaken)
```

Output:



Scatter plot

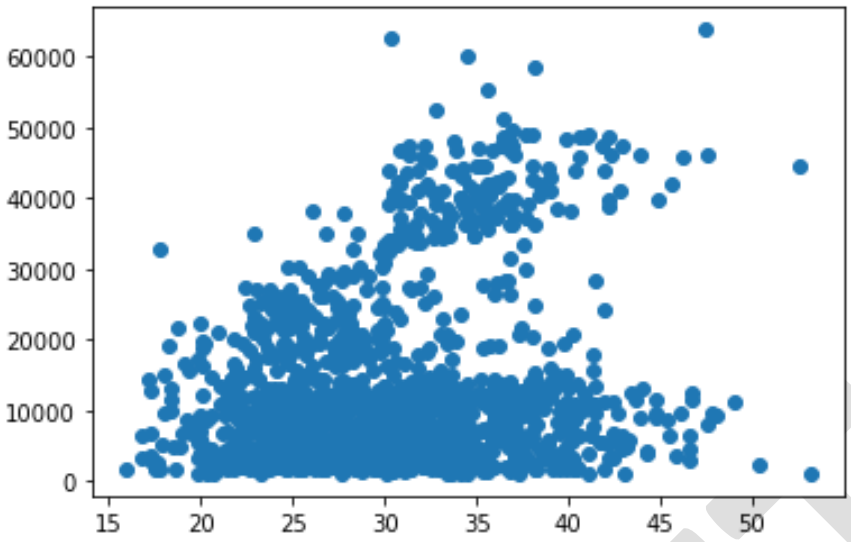
Scatter plot is a type of plot that shows the data as a collection of points, the position of a point depends on its two-dimensional value, where each value is a position on either the horizontal or vertical dimension, it is really useful in the study of the relationship or pattern between variables.

To draw scatter plot we can use Matplotlib library's scatter method with the x and y parameters.

Example:

```
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
df = pd.read_csv('insurance.csv')
plt.scatter(df['bmi'], df['charges'])
plt.show()
```

Output:



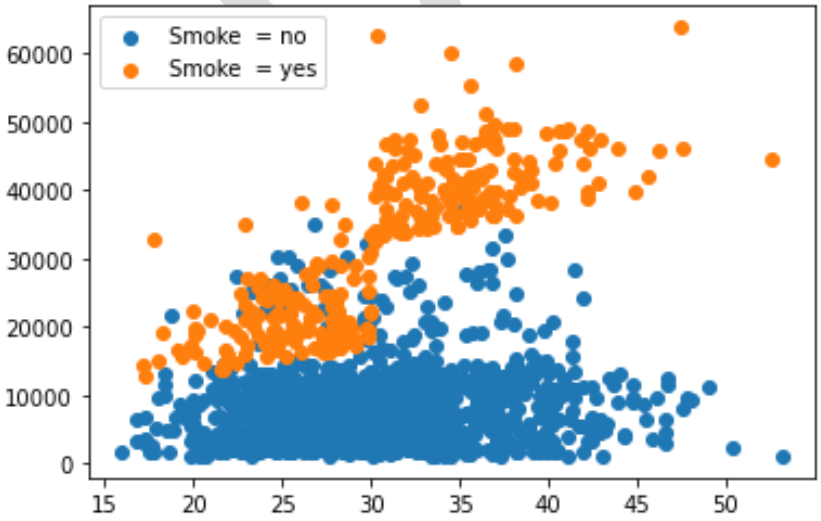
To find a specific pattern from the data, we can further divide the data and plot scatter plot.

We can do this with the help of groupby method of DataFrame and then using tuple unpacking while looping the group.

Example:

```
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
df = pd.read_csv('insurance.csv')
grouped = df.groupby(['smoker'])
for key, group in grouped:
   plt.scatter(group['bmi'], group['charges'], label='Smoke = '+key)
plt.legend()
plt.show()
```

Output:



we can specify marker, color, and size of the marker with the help of marker, color and s parameters respectively.

Time series

Observations over time can be considered as a Time Series, visualization plays an important role in time series analysis and forecasting, time Series plots can provide valuable diagnostics to identify temporal structures like trends, cycles, and seasonality.

To create a Time Series, we first need to get the date range, which can be created with the help of datetime and pandas library.

Example:

```
import pandas as pd
import datetime as dt
start_date = dt.datetime(2020,8,28)
end_date = dt.datetime(2020,9,05)
daterange = pd.date_range(start_date,end_date)
print(daterange)
```

Output:

```
DatetimeIndex(['2020-08-28', '2020-08-29', '2020-08-30', '2020-08-31', '2020-09-01',
'2020-09-02', '2020-09-03', '2020-09-04', '2020-09-05'],
dtype='datetime64[ns]', freq='D')
```

We can use some more parameters for date_range() function like

- freq, to specify the frequency at which we want the date range (default is 'D' for days)
- periods, number of periods to generate in between start/end or from start with freq.

Some of important possible values for the freq are

- D, for calendar day
- W, for week
- M, for month
- Y, for year
- H, for hour
- T/min, for minute
- S, for seconds
- L, for milliseconds
- B, for business day
- SM, for semi month-end
- Q, for quarter-end
- BQ, for business quarter-end

Basemap

One common type of visualization in data science is that of geographic data. Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits that lives under the mpl_toolkits namespace.

Admittedly, Basemap feels a bit clunky to use, and often even simple visualizations take much longer to render than we might hope. More modern solutions such as leaflet or the Google Maps API may be a better choice for more intensive map visualizations. Still, Basemap is a useful tool for Python users for some tasks.

Installation of Basemap is straightforward; if you're using conda you can execute "**conda install basemap**" command in the command prompt/terminal and the package will be downloaded.

NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Advantages of NetworkX,

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Open-source
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform

We can use networkx library in order to deal with any kind of networks, which includes a social network, railway network, road connectivity, etc….

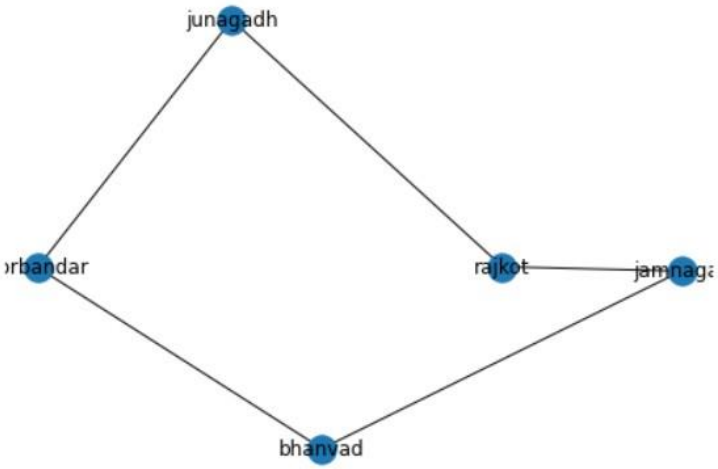
To install NetworkX we can use,

- pip install networkx **OR**
- conda install networkx

Example:

```
import networkx as nx
g = nx.Graph() # undirected graph
g.add_edge('rajkot','junagadh')
g.add_edge('junagadh','porbandar')
g.add_edge('rajkot','jamnagar')
g.add_edge('jamnagar','bhanvad')
g.add_edge('bhanvad','porbandar')
nx.draw(g,with_labels=True)
```

Output:

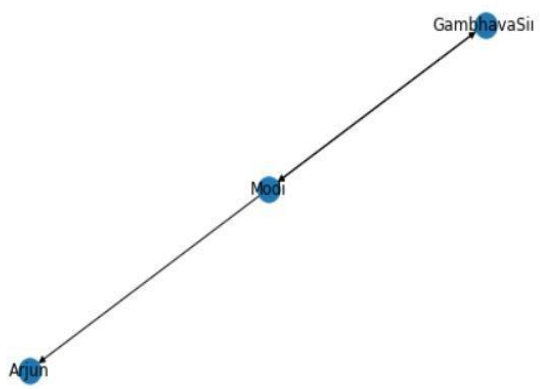


We can also use NetworkX's DiGraph to generate the directed graph,

Example:

```
import networkx as nx
gD = nx.DiGraph() # directed graph
gD.add_edge('Modi','Arjun')
gD.add_edge('Modi','GambhavaSir')
gD.add_edge('GambhavaSir','Modi')
nx.draw(gD, with_labels=True)
```

Output:



We can use many analysis functions available in NetworkX library, some of functions are as below,

- nx.shortest_path(g,'rajkot','porbandar')
  - Will return ['rajkot', 'junagadh', 'porbandar']
- nx.clustering(g)
  - Will return clustering value for each node
- nx.degree_centrality(g)
  - Will return the degree of centrality for each node, we can find the most popular/influential node using this method.
- nx.density(g)
  - Will return the density of the graph.
  - The density is 0 for a graph without edges and 1 for a complete graph.
- nx.info(g)
  - Return a summary of information for the graph G.
  - The summary includes the number of nodes and edges and their average degree.