Data Visualization using Matplotlib

Data Visualization

Data visualization is a technique to present the data in a pictorial or graphical format.

Well, you might wonder why data visualization is important?

Data Visualization

The main benefits of data visualization are as follows:

Data Visualization Considerations

Three major considerations for data visualization:

Clarity

Accuracy

Efficiency

Ensure the dataset is complete and relevant. This enables the Data Scientist to use the new patterns obtained from the data in the relevant places.

Data Visualization Considerations

Three major considerations for data visualization:

Clarity

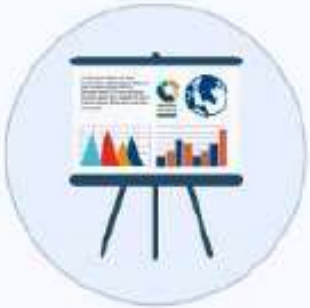
Accuracy

Efficiency

Ensure you use appropriate graphical representation to convey the intended message.

Data Visualization Considerations

Three major considerations for data visualization:



Clarity

Accuracy

Efficiency

Use efficient visualization techniques that highlight all the data points.

Data Visualization Tool- Python

How is data visualization performed for large and complex data?

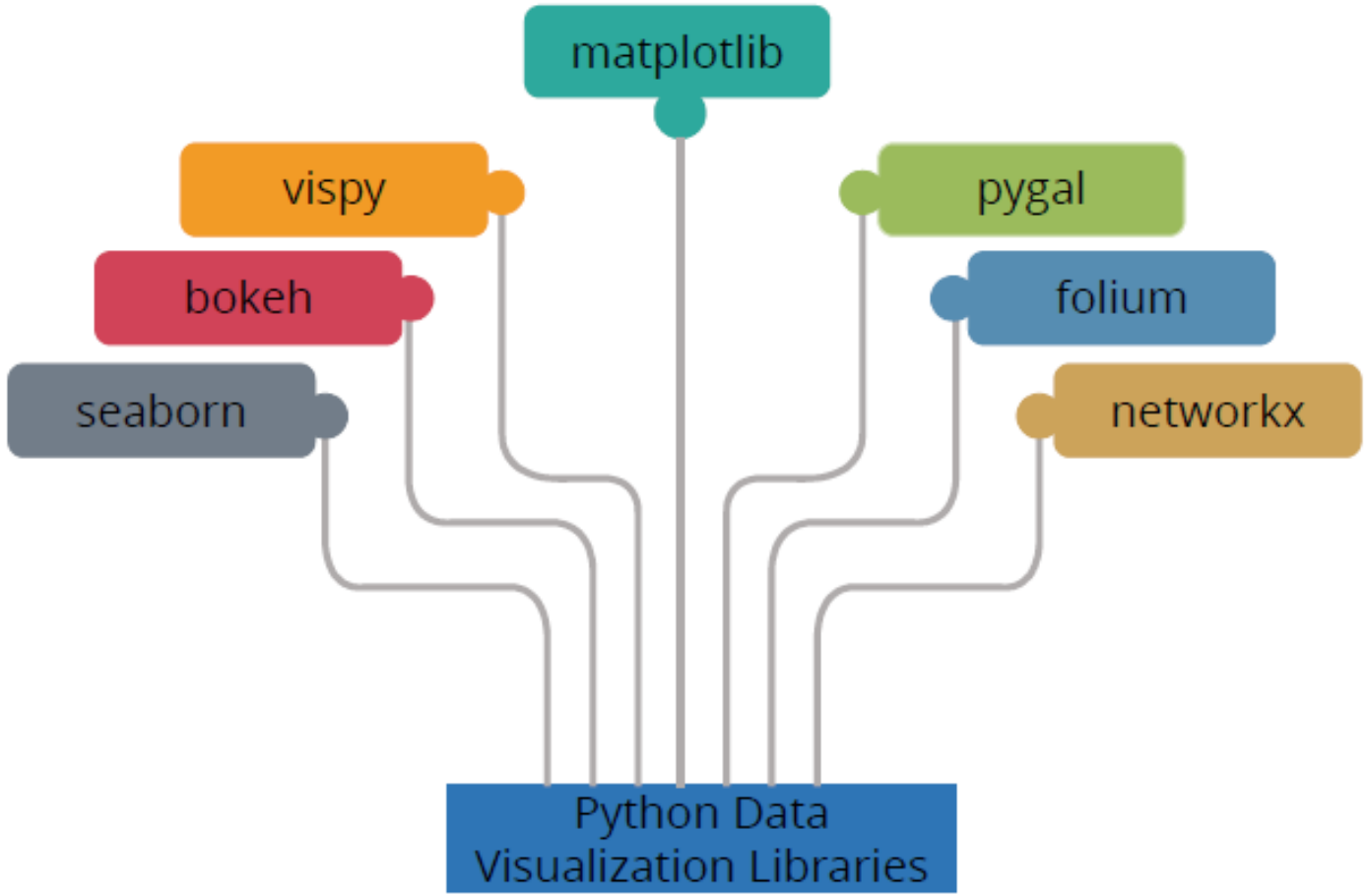
?

What data visualization is?

How data visualization helps interpret results with large data

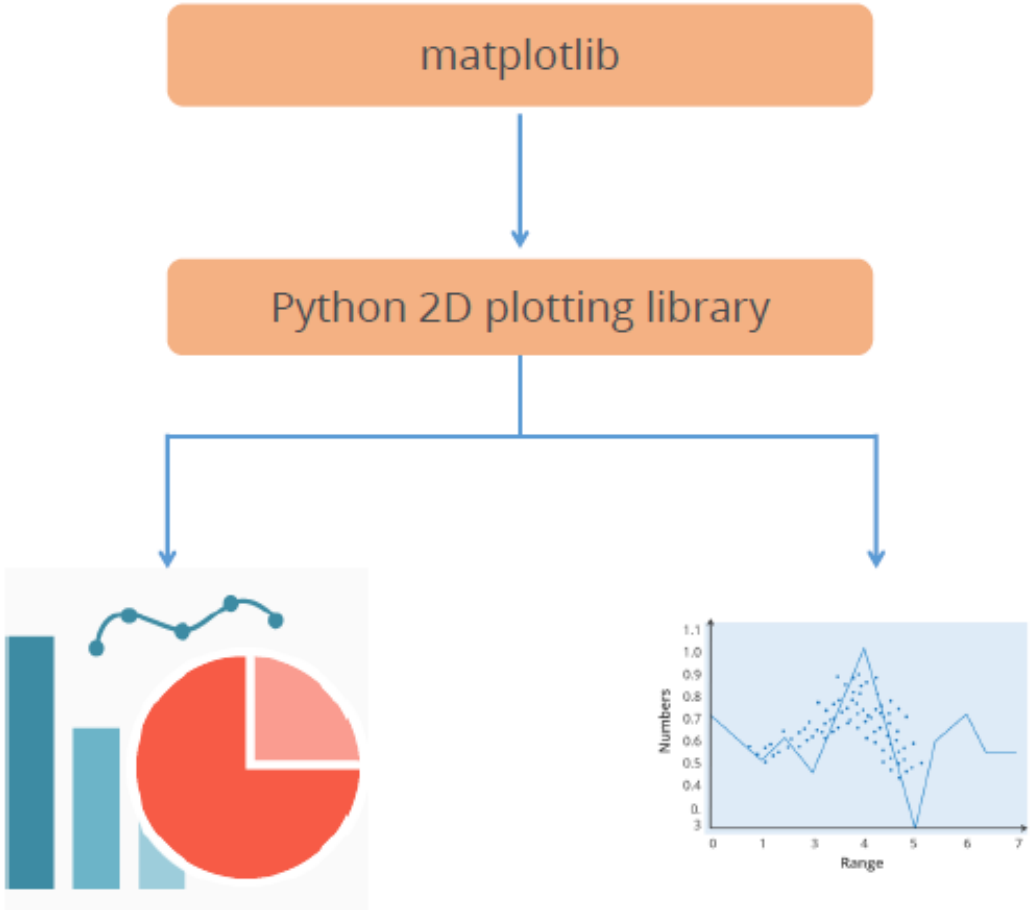
Python Libraries

Many new Python data visualization libraries are introduced recently such as:



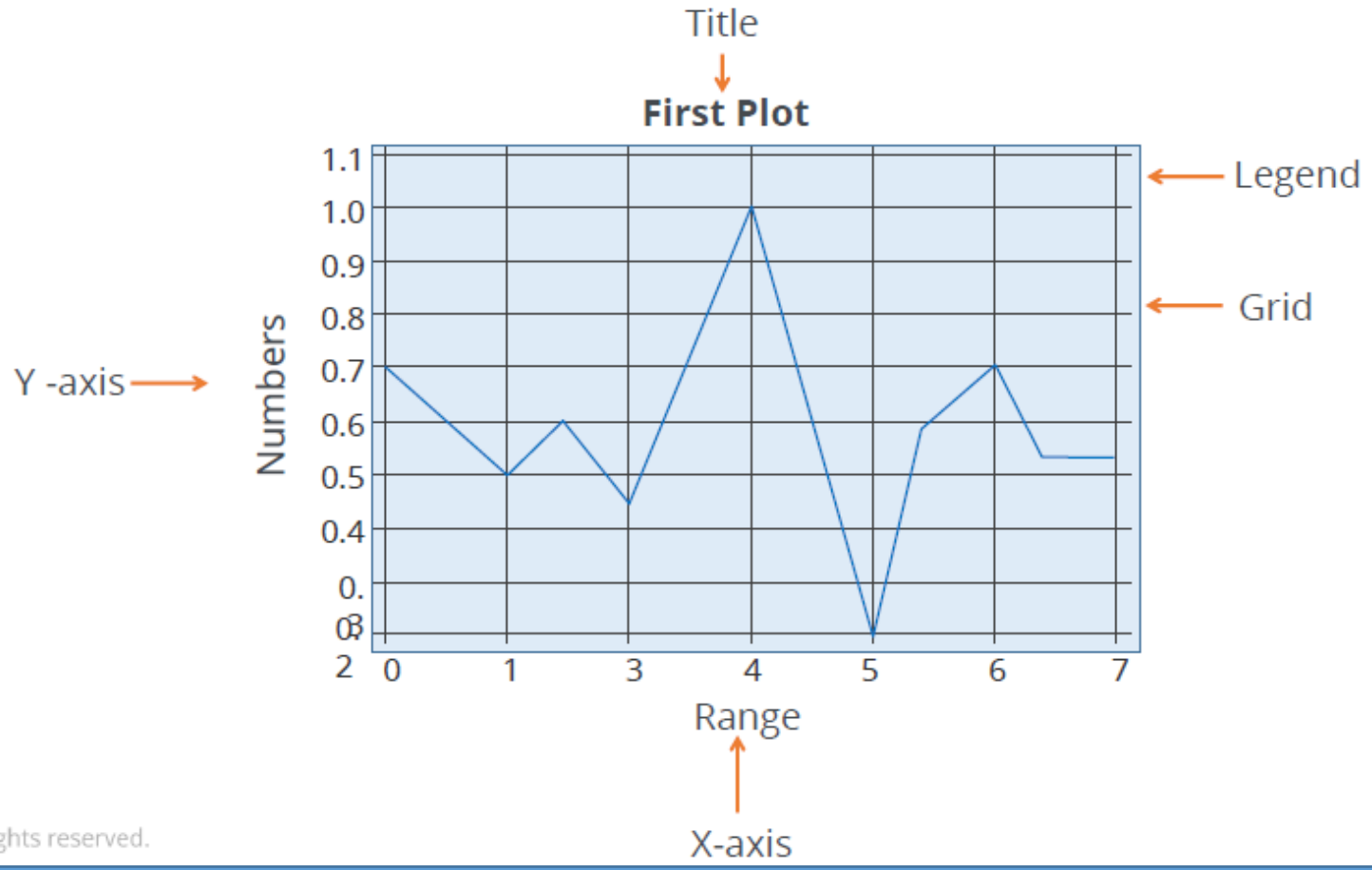
Python Libraries - matplotlib

Using Python's matplotlib, the data visualization of large and complex data becomes easy.



The Plot

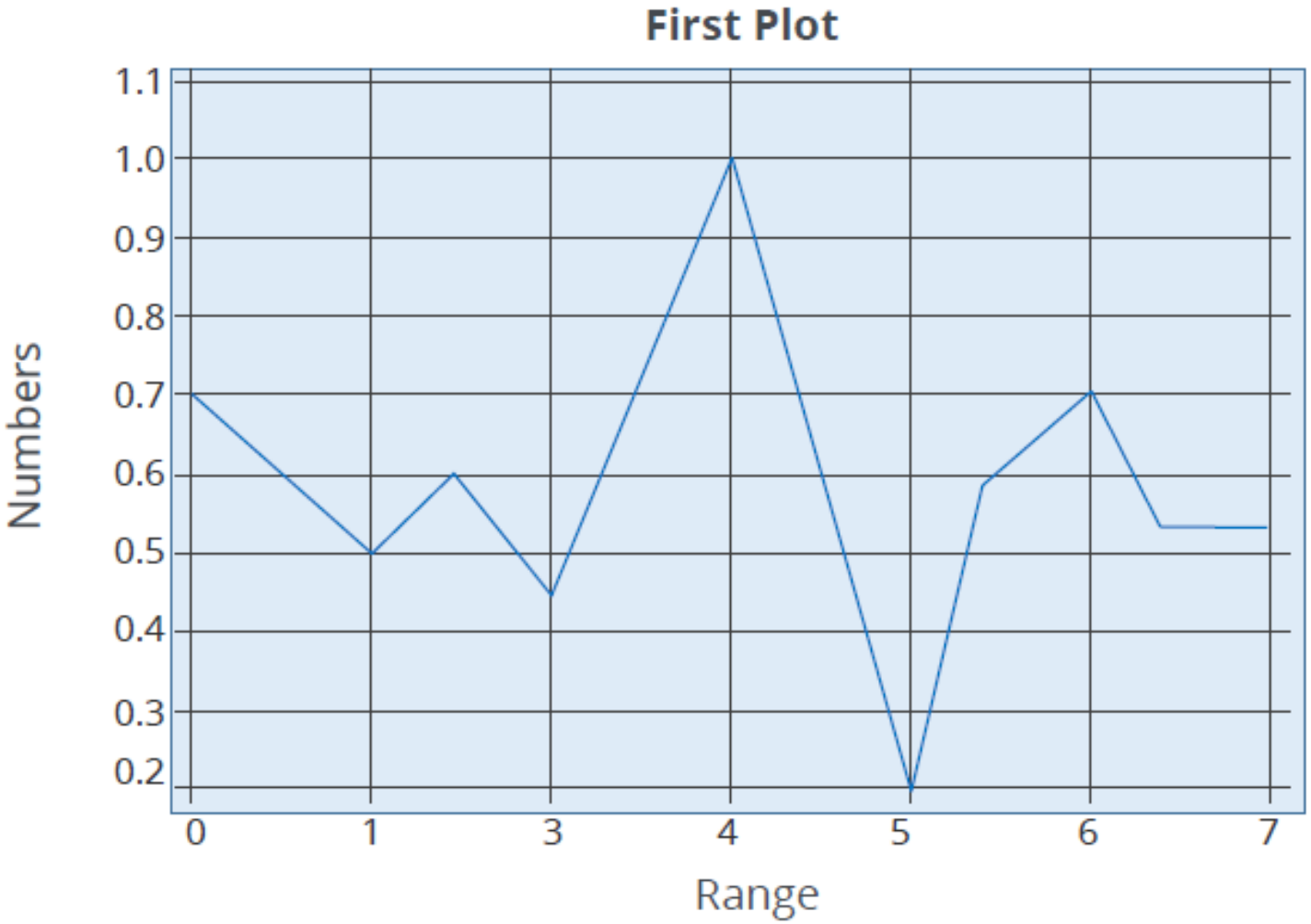
A plot is a graphical representation of data, which shows the relationship between two variables or the distribution of data.



Steps to Create a Plot

You can create a plot using four simple steps.

Steps to Create Plot- Example



Steps to Create Plot- Example

```
In [1]: #import numpy for generating random numbers
        import numpy as np
        #import matplotib library
        import matplotlib.pyplot as plt
        from matplotlib import style
        %matplotlib inline
```

Generate random numbers — numpy
Plot the numbers — pyplot
set the grid style — style

```
In [21]: #generate random numbers (total 10)
         randomNumber = np.random.rand(10)
```

used numpy random method — Defined the dataset

```
In [22]: #view them
         print randomNumber
```

view the created random numbers — Print method

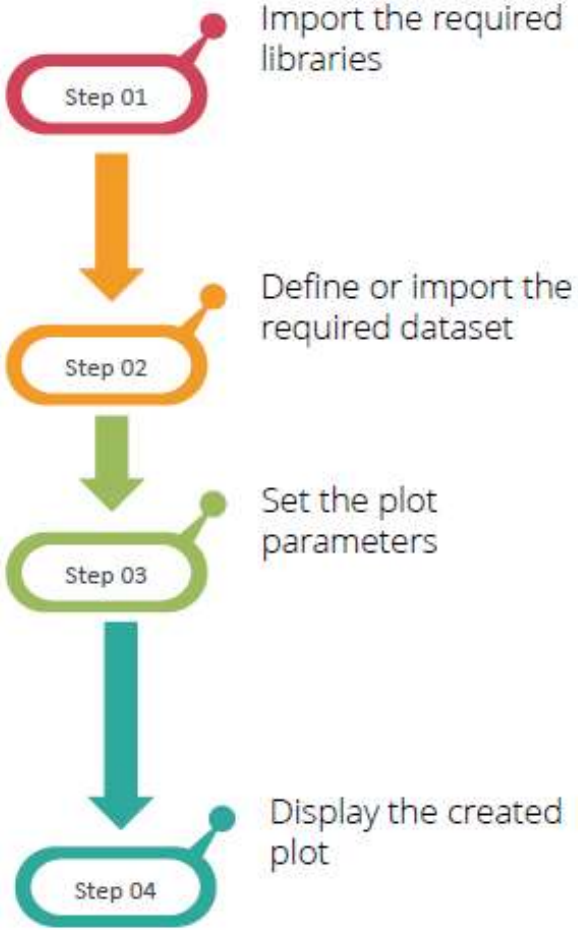
```
[ 0.71892609  0.49065612  0.61092193  0.43397501  0.94771363  0.31505178
  0.58568599  0.6929941   0.4288734   0.43774794]
```

```
In [23]: #select the style of the plot
         style.use('ggplot')
         #plot the random number
         plt.plot(randomNumber,'g',label='line one',linewidth=2)
         #x axis is number of random numbers (index)
         plt.xlabel('Range')
         #y axis is actual random number
         plt.ylabel('Numbers')
         #Title of the plot
         plt.title('First Plot')
```

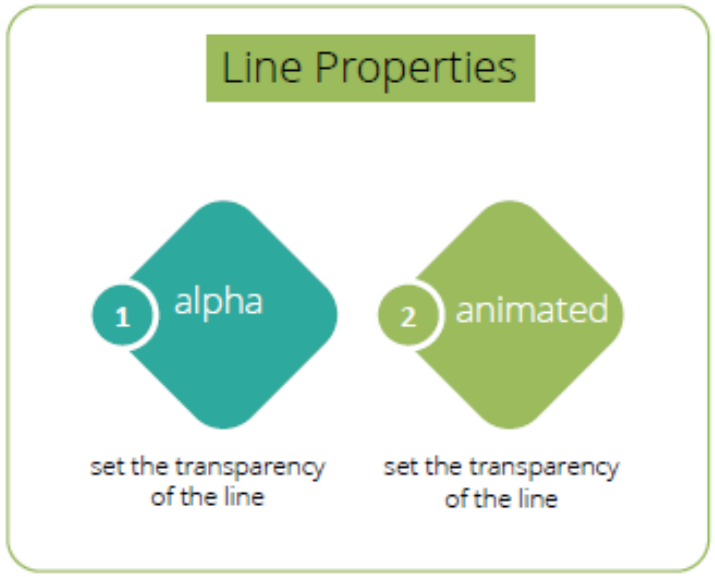
ggplot — Set the style
Set the legend
Set line width
Set coordinates labels
Set the title

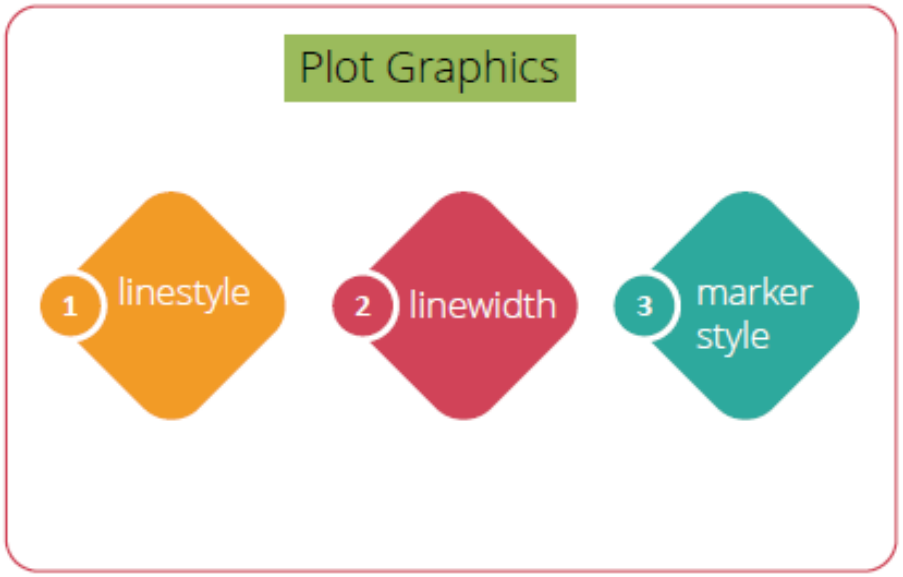
```
plt.legend()
plt.show()
```

Plot the graph
Display the created plot



Line Properties





matplotlib also offers various line colors.

View Line Properties

Click **View Line Properties** *to know more.*

Line Properties

Property	Value Type
alpha	float
animated	[True | False]
antialiased or aa	[True | False]
clip_box	a matplotlib.transform.Bbox instance
clip_on	[True | False]
clip_path	a Path instance and a Transform instance, a Patch
color or c	any matplotlib color
contains	the hit testing function
dash_capstyle	['butt' | 'round' | 'projecting']
linestyle or ls	['-' | '--' | '-.' | ':' | 'steps' | ...]
linewidth or lw	float value in points
marker	['+' | ',' | '.' | '1' | '2' | '3' | '4']

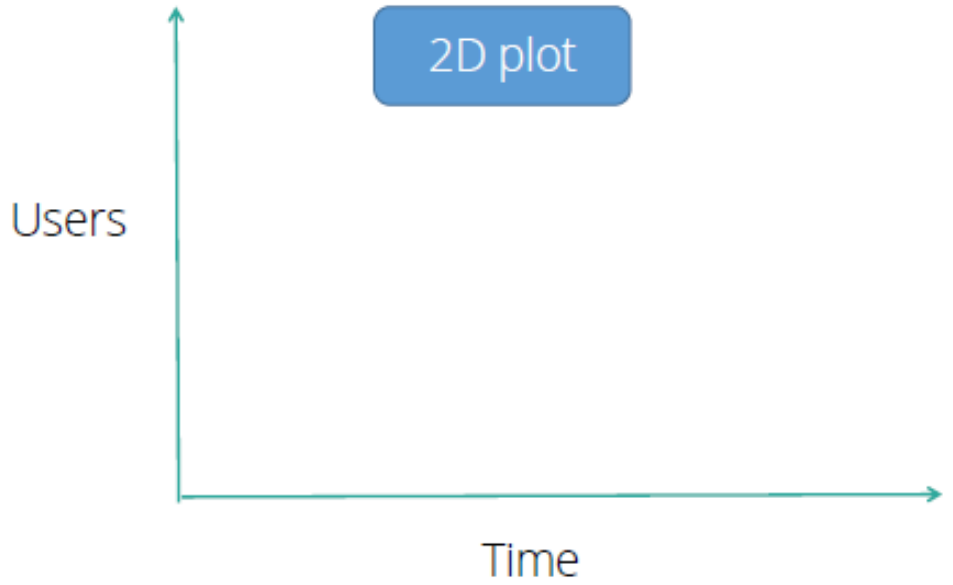
Alias	Color
b	Blue
r	Red
c	Cyan
m	Magenta
g	Green
y	Yellow
k	Black
w	White

View Line Properties

Click **View Line Properties** *to know more.*

Plot with (x,y)

A leading global organization wants to know how many people visit its website in a particular time. This analysis helps it control and monitor the website traffic.



Data Science

Plot with (x,y)

```
In [1]: #import matplotib library
        import matplotlib.pyplot as plt
        from matplotlib import style
        %matplotlib inline
```

```
In [2]: #website traffic data
        #number of users/ visitors on the web site
        web_customers = [123,645,950,1290,1630,1450,1034,1295,465,205,80 ]
        #Time distribution (hourly)
        time_hrs = [7,8,9,10,11,12,13,14,15,16,17]
```

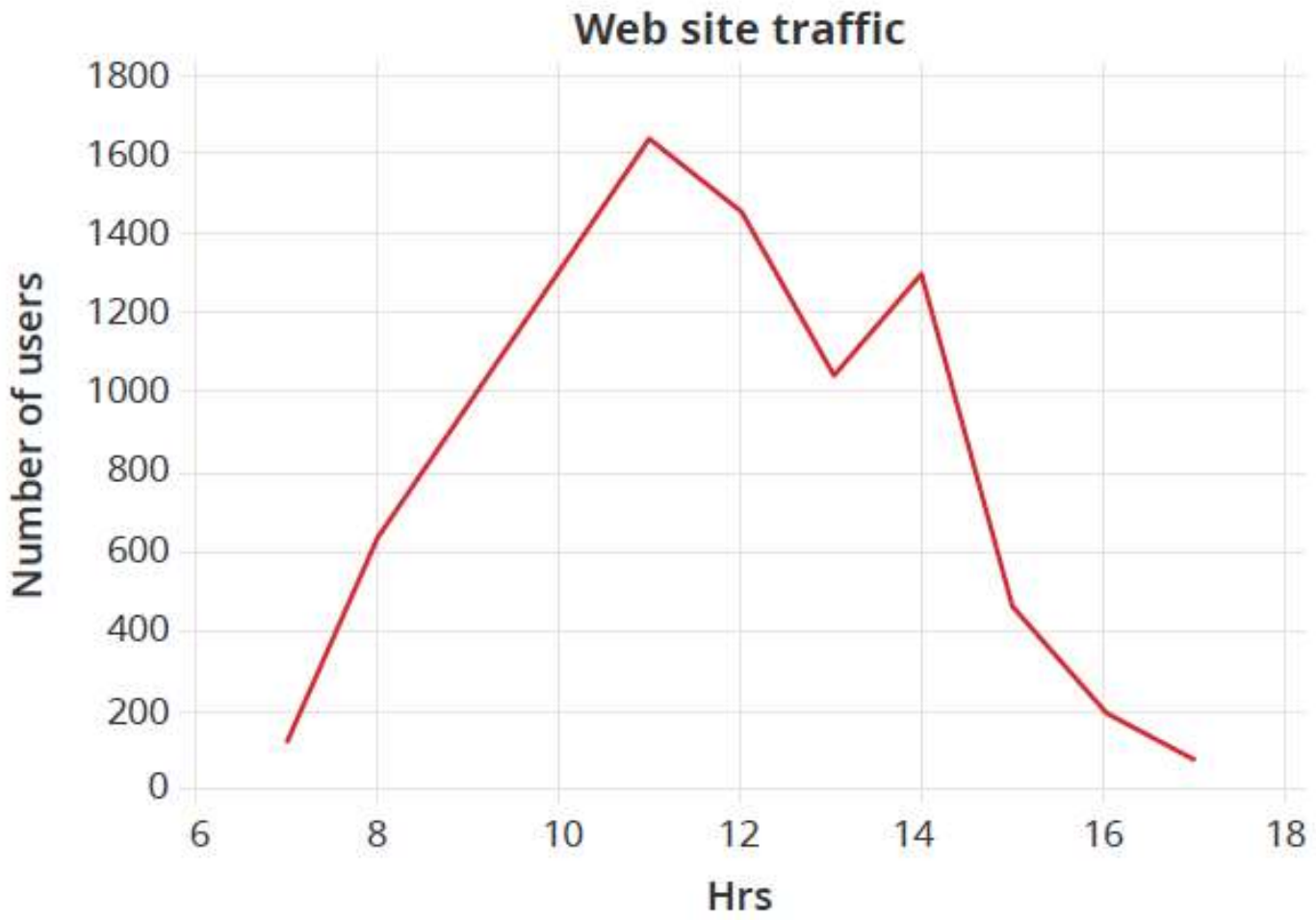
List of users

Time

```
In [3]: #select the style of the plot
        style.use('ggplot')
        #plot the web site traffif data (X-axis hrs and Y axis as number of users)
        plt.plot(time_hrs,web_customers)
        #set the title of the plot
        plt.title('Web site traffic')
        #set label for x axis
        plt.xlabel('Hrs')
        #set label for y axis
        plt.ylabel('Number of users')
        plt.show()
```

Use %matplotlib inline to display or view the plot on Jupyter notebook.

Plot with (x,y)

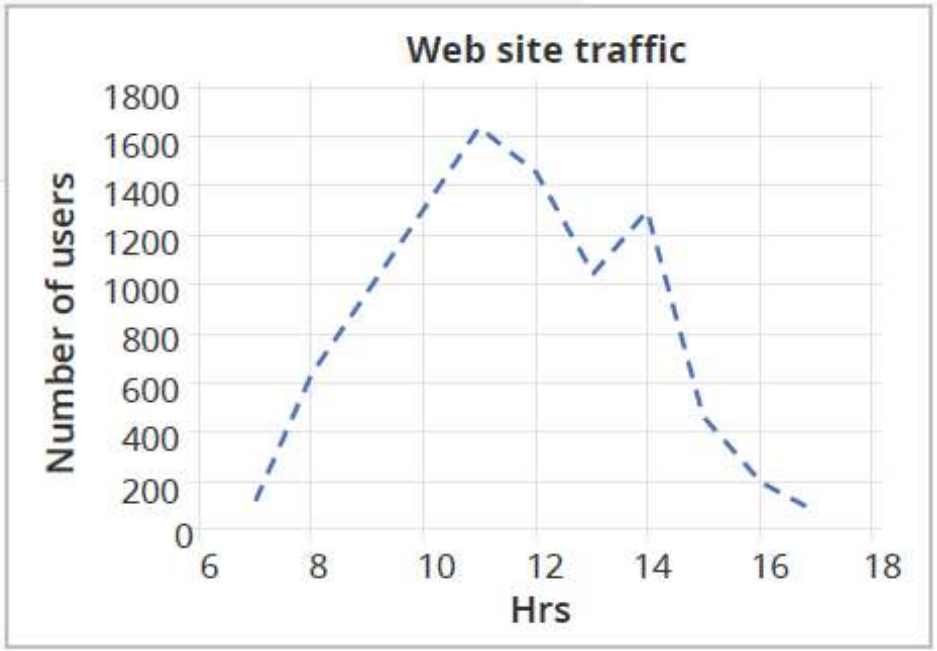

Web site traffic
Number of users
1800
1600
1400
1200
1000
800
600
400
200
0
6
8
10
12
14
16
18
Hrs

Controlling Line Patterns and Colors

```
#select the style of the plot
style.use('ggplot')
#plot the web stite traffic data (x axis hrs and y asis as number of users)
plt.plot(time_hrs,web_customers,color = 'b',linestyle = '--',linewidth=2.5)
#set the title of the plot
plt.title('Web site traffic')
#set the label for x axis
plt.xlabel('hrs')
#set the label for y axis
plt.ylabel('number of users')
plt.show()
```

Line Color (blue)

Dashed (--)



Data Science

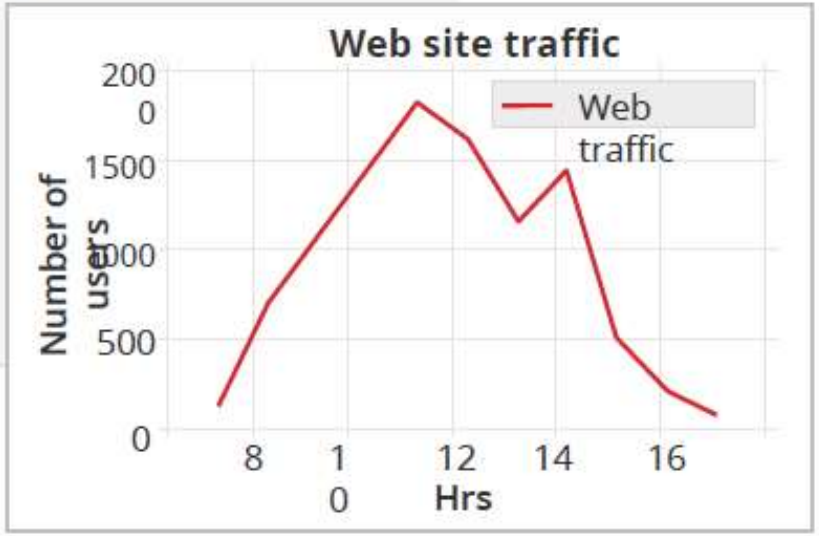
Set Axis, Labels, and Legend Property

Using matplotlib, it is also possible to set the desired axis to interpret the result.

Axis is used to define the range on the x axis and y axis.

```
:  #select the style of the plot
   style.use('ggplot')
   #plot the web site traffif data (X-axis hrs and Y axis as number of users)
   plt.plot(time_hrs,web_customers,'r',label='web traffic',linewidth=1.5)
   plt.axis([6.5,17.5,50,2000])
   #set the title of the plot
   plt.title('Web site traffic')
   #set label for x axis
   plt.xlabel('Hrs')
   #set label for y axis
   plt.ylabel('Number of users')
   plt.legend()
   plt.show()
```

Set the axis



Alpha and Annotation

Alpha is an attribute that controls the transparency of the line.
The lower the alpha value, the more transparent the line is.

```
#select the style of the plot
style.use('ggplot')
#plot the web stite traffic data (x axis hrs and y asis as number of users)
#also setting the alpha value for transparency
plt.plot(time_hrs,web_customers,alpha=.4)
#set the title of the plot
plt.title('Website Traffic')
#Annotate
plt.annotate('Max',ha='center',va='bottom',xytext=(8,1500),xy=(11,1630),arrowprops =
             { 'facecolor' : 'green'})
#set the label for x axis
plt.xlabel('hrs')
#set the label for y axis
plt.ylabel('number of users')

plt.show()
```

Alpha and Annotation

Annotate() method is used to annotate the graph. It has several attributes which help annotate the plot.

```
#select the style of the plot
style.use('ggplot')
#plot the web stite traffic data (x axis hrs and y asis as number of users)
#also setting the alpha value for transparency
plt.plot(time_hrs,web_customers,alpha=.4)
#set the title of the plot
plt.title('Website Traffic')
#Annotate
plt.annotate('Max',ha='center',va='bottom',xytext=(8,1500),xy=(11,1630),arrowprops =
             { 'facecolor' : 'green'})
#set the label for x axis
plt.xlabel('hrs')
#set the label for y axis
plt.ylabel('number of users')

plt.show()
```

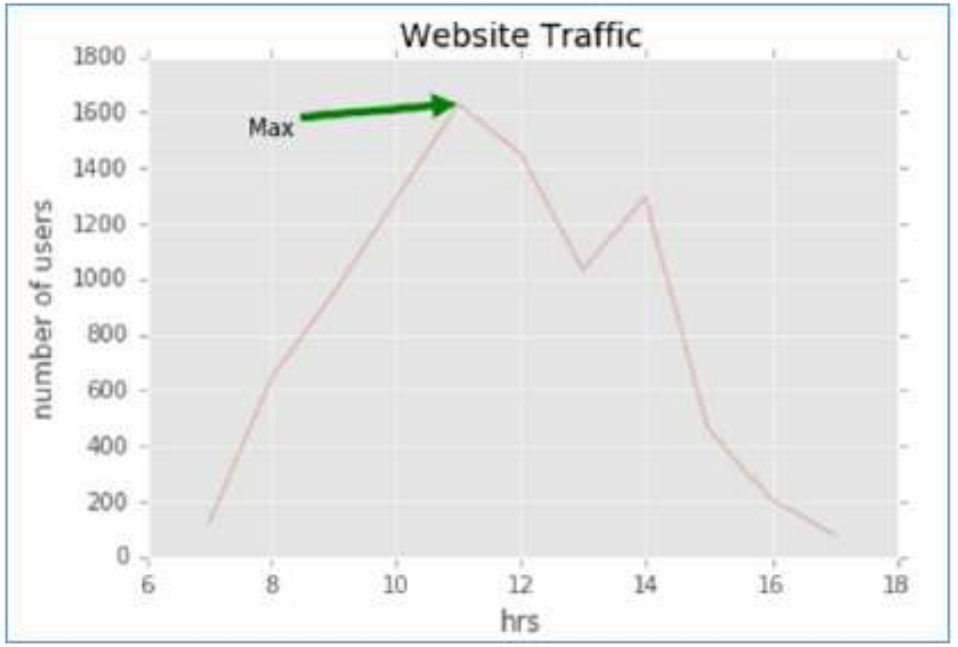
"Max" denotes the annotation text,
"ha" indicates the horizontal alignment,
"va" indicates the vertical alignment,
"xytext" indicates the text position,
"xy" indicates the arrow position, and
"arrowprops" indicates the properties of the arrow.

Alpha and Annotation

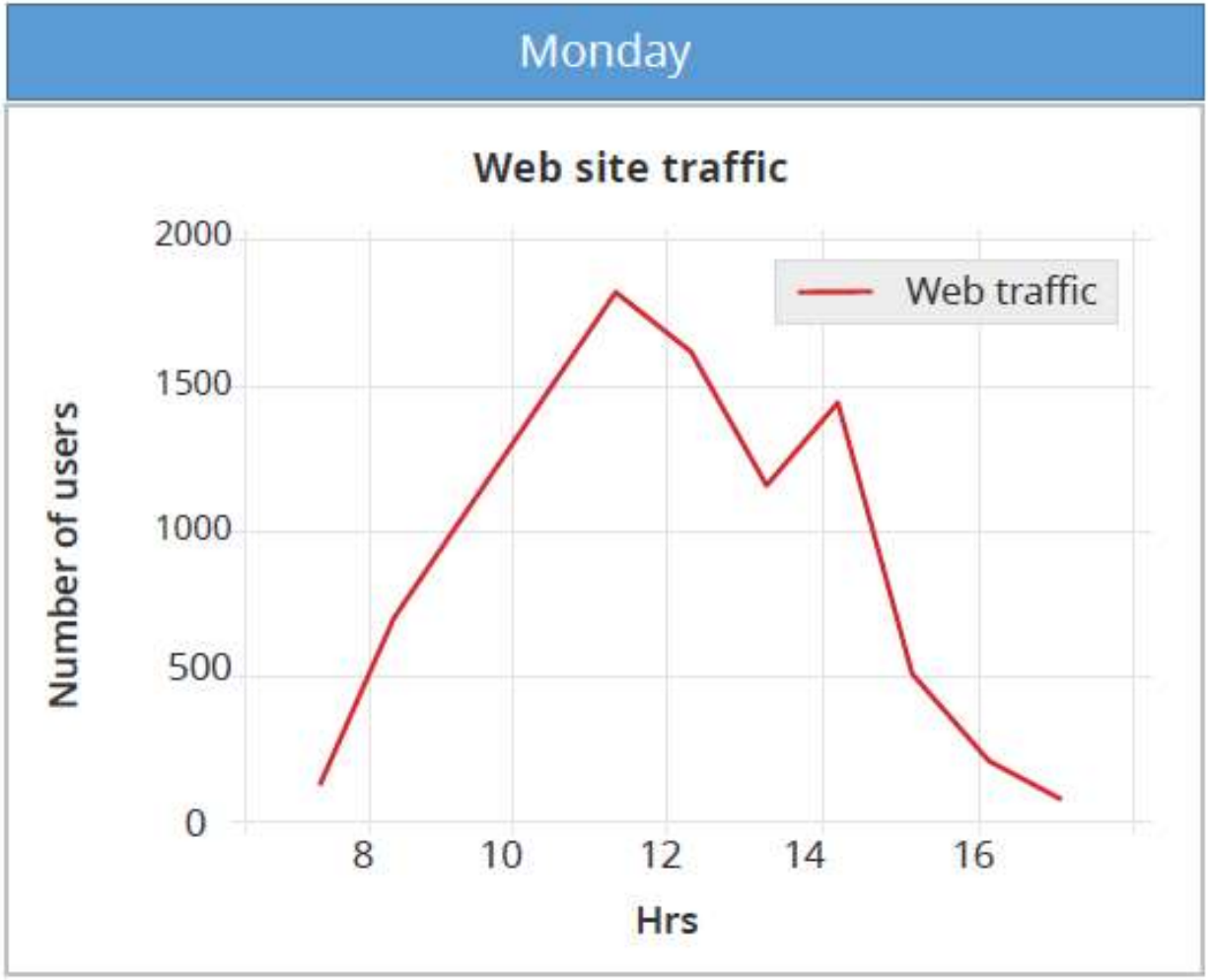
Annotate() method is used to annotate the graph. It has several attributes which help annotate the plot.

```
#select the style of the plot
style.use('ggplot')
#plot the web stite traffic data (x axis hrs and y asis as number of users)
#also setting the alpha value for transparency
plt.plot(time_hrs,web_customers,alpha=.4)
#set the title of the plot
plt.title('Website Traffic')
#Annotate
plt.annotate('Max',ha='center',va='bottom',xytext=(8,1500),xy=(11,1630),arrowprops =
             { 'facecolor' : 'green'})
#set the label for x axis
plt.xlabel('hrs')
#set the label for y axis
plt.ylabel('number of users')

plt.show()
```



Multiple Plots



Multiple Plots

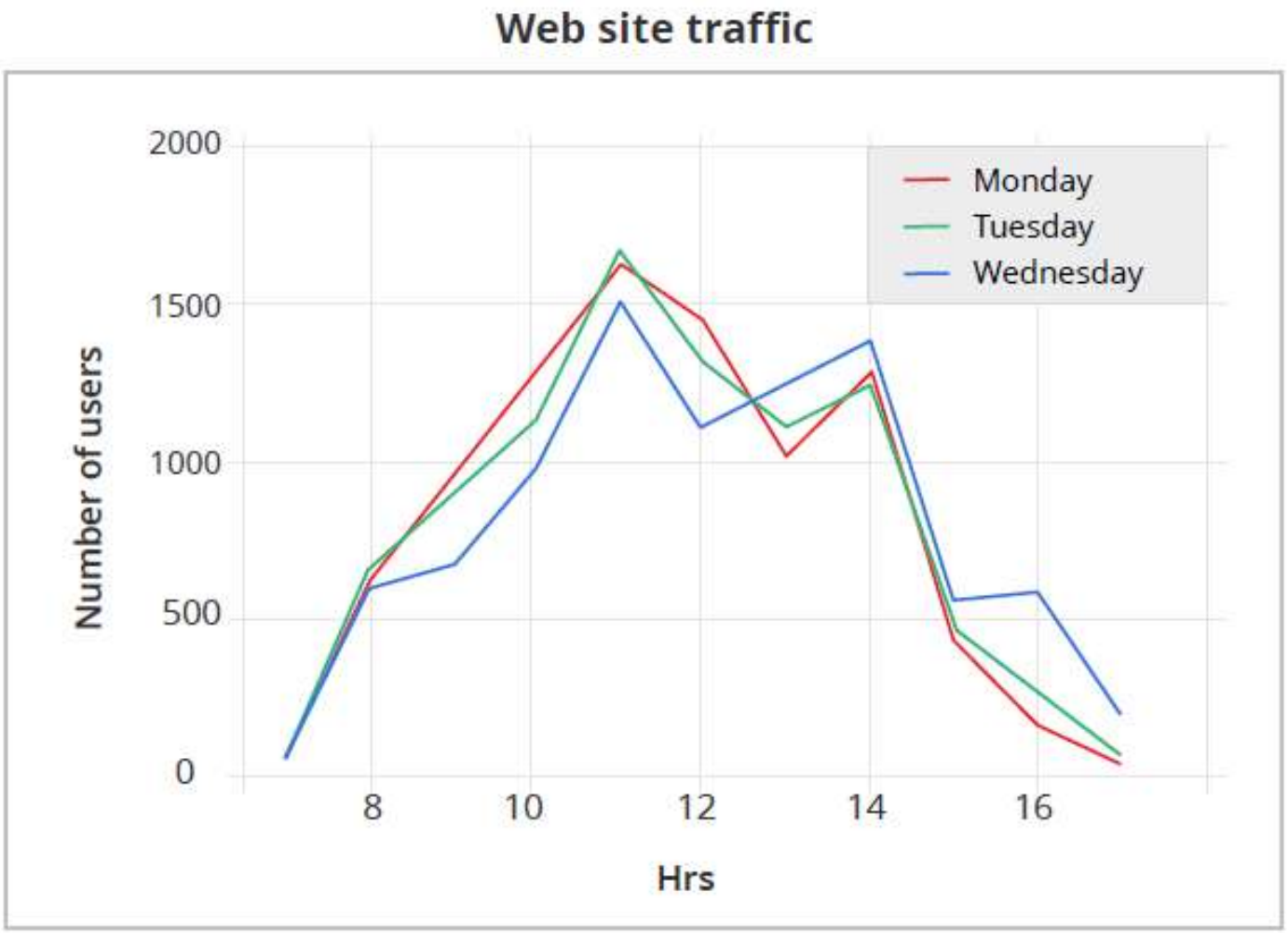
In [4]:
```
#website traffic data
#number of users/ visitors on the web site
#monday web traffic
web_monday = [123,645,950,1290,1630,1450,1034,1295,465,205,80]
#tuesday web traffic
web_tuesday= [95,680,889,1145,1670,1323,1119,1265,510,310,110]
#wednesday web traffic
web_wednesday= [105,630,700,1006,1520,1124,1239,1380,580,610,230]
#Time distribution (hourly)
time_hrs = [7,8,9,10,11,12,13,14,15,16,17]
```

Web traffic data

In [5]:
```
#select the style of the plot
style.use('ggplot')
#plot the web site traffic data (X-axis hrs and Y axis as number of users)
#plot the monday web traffic with red color
plt.plot(time_hrs,web_monday,'r',label='monday',linewidth=1)
#plot the monday web traffic with green color
plt.plot(time_hrs,web_tuesday,'g',label='tuesday',linewidth=1.5)
#plot the monday web traffic with blue color
plt.plot(time_hrs,web_wednesday,'b',label='wednesday',linewidth=2)
plt.axis([6.5,17.5,50,2000])
#set the title of the plot
plt.title('Web site traffic')
#set label for x axis
plt.xlabel('Hrs')
#set label for y axis
plt.ylabel('Number of users')
plt.legend()
plt.show()
```

Set different colors and line widths for different days

Multiple Plots



Subplots

Subplots are used to display multiple plots in the same window.

With subplot, you can arrange plots in a regular grid.

The syntax for subplot is

subplot(m,n,p). → It divides the current window into an m-by-n grid and creates an axis for a subplot in the position specified by p.

For example,

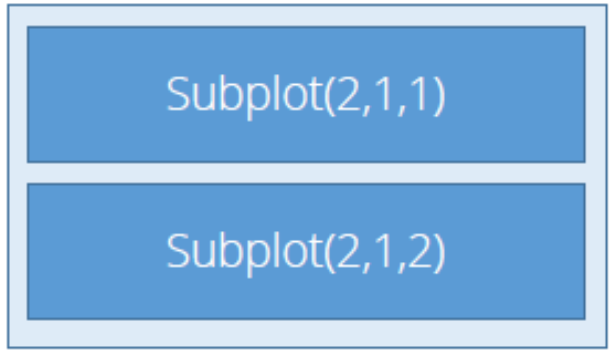
subplot(2,1,2) creates two subplots which are stacked vertically on a grid.
subplot(2,1,4) creates four subplots in one window.

Subplots

Subplots are used to display multiple plots in the same window.

With subplots, you can arrange plots in a regular grid.

Grid divided into two vertically stacked plots



Grid divided into four plots

Layout

Layout and Spacing adjustments are two important factors to be considered while creating subplots.

Use the plt.subplots_adjust() method with the parameters hspace and wspace to adjust the distances between the subplots and move them around on the grid.

Types of Plots

You can create different types of plots using matplotlib:

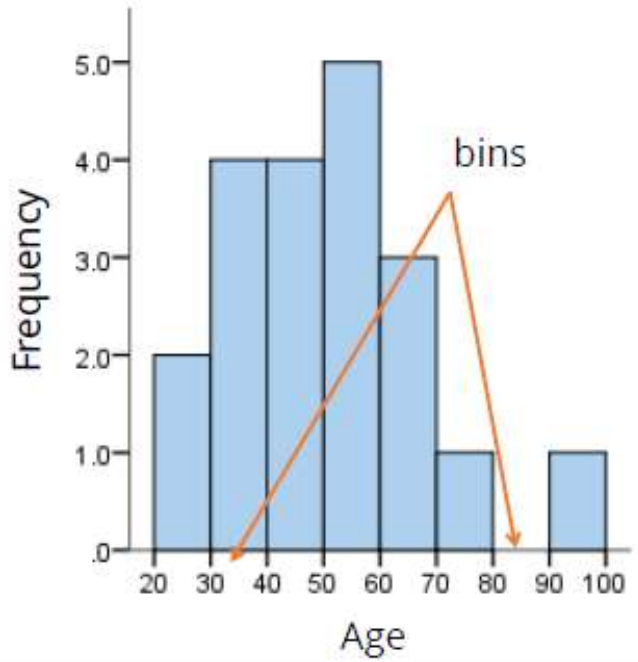
Click each plot to know more.

- Histogram
- Scatter Plot
- Heat Map
- Pie Chart

Histograms are graphical representations of a probability distribution. A histogram is a kind of a bar chart.
Using matplotlib and its bar chart function, you can create histogram charts.

Advantages of Histogram charts:

- They display the number of values within a specified interval.
- They are suitable for large datasets as they can be grouped within the intervals.



Types of Plots

You can create different types of plots using matplotlib:

Click each plot to know more.

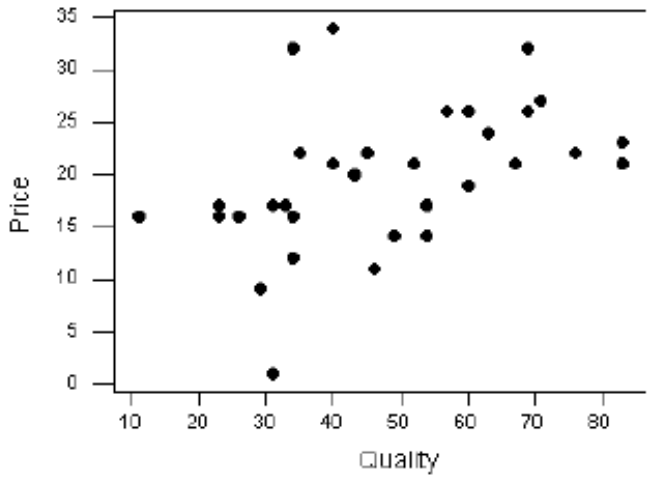
- Histogram
- Scatter Plot
- Heat Map
- Pie Chart

A scatter plot is used to graphically display the relationships between variables.

However, to control a plot, it is recommended to use scatter() method.

It has several advantages:

- Shows the correlation between variables
- Is suitable for large datasets
- Is easy to find clusters
- Is possible to represent each piece of data as a point on the plot



Types of Plots

You can create different types of plots using matplotlib:

Click each plot to know more.

- Histogram
- Scatter Plot
- Heat Map
- Pie Chart

A heat map is a way to visualize two-dimensional data. Using heat maps, you can gain deeper and faster insights about data than other types of plots.

It has several advantages:

- Draws attention to the risk-prone area
- Uses the entire dataset to draw meaningful insights
- Is used for cluster analysis and can deal with large datasets

Types of Plots

You can create different types of plots using matplotlib:

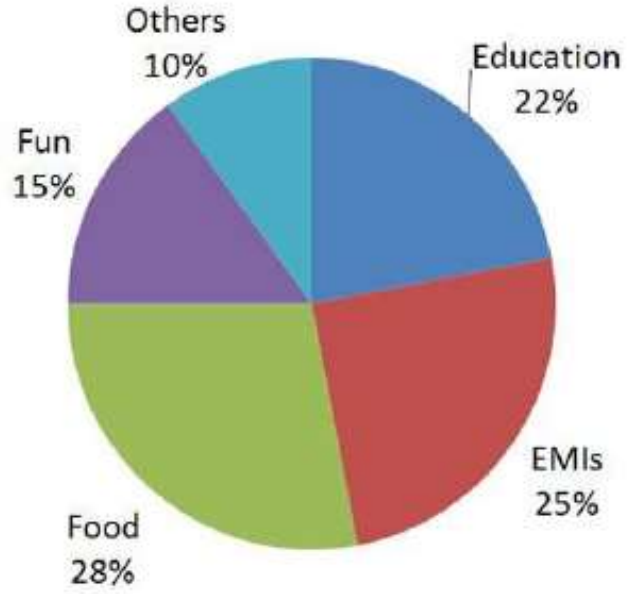
Click each plot to know more.

- Histogram
- Scatter Plot
- Heat Map
- Pie Chart

Pie charts are used to show percentage or proportional data.
matplotlib provides the pie() method to create pie charts.

It has several advantages:

- Summarizes a large dataset in visual form
- Displays the relative proportions of multiple classes of data
- Size of the circle is made proportional to the total quantity

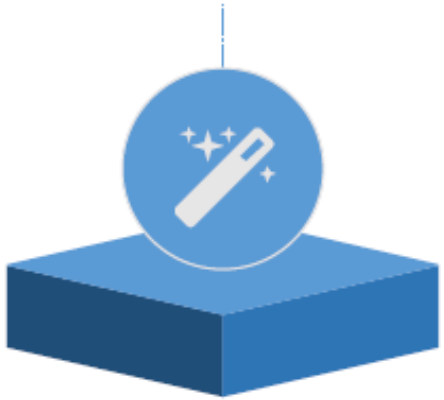


Seaborn

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface to draw attractive statistical graphics.

There are several advantages:

Possesses built-in themes for better visualizations



Has built-in statistical functions which reveal hidden patterns in the dataset

Has functions to visualize matrices of data