PYTHON PROGRAMMING & DATA SCIENCE

Python Data Files

- The file handling plays an important role when the data needs to be stored permanently into the file.
- A file is a named location on disk to store related information.
- We can access the information from the file whenever we need.
- In Python, files are treated in two modes as text or binary.
- A file operation can be done in the following order.

Open a file

Read or write - Performing operation

Close the file

TYPES OF DATA FILES

- 1. TEXT FILE
- 2. BINARY FILE
- **Text Files -** This type of file consists of the normal characters in ASCII format. Each row is terminated by the special character. This special character is called EOL (End of Line). In Python, the new line ('\n') is used by default.
- **Binary Files -** In this file format, the data is stored in the binary format (1 or 0). The binary file doesn't have any terminator for a newline.

FILE HANDLING FUNCTIONS:

1. open(): The open function is used to open the existing data file or to create new file.

    syntax: **file object = open(<file-name>, <access-mode>)**

file-name represents data file name, which is a physical file name in the disk.

File object is a file pointer which is used to access the data from the file. All File operations are performed through file object only.

File access-modes are given in the following table.

FILE ACCESS METHODS

- 1. Sequential Access: In sequential access method, the file contents will be accessed from beginning of the file to bottom of the file.

- 2. Random Access : In Random Access method, the file contents will be accessed as randomly based on requirement.

SN	Access mode	Description
1	r	It opens the file to read-only mode. The file pointer exists at the beginning.
2	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously.
7	w+	It opens the file to write and read both. It creates a new file if no file exists.
8	wb+	It opens the file to write and read both in binary format.
9	a	It opens the file in the append mode.
10	ab	It opens the file in the append mode in binary format
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

1.Write a program to create data file

Write() method is used to write a the given data into the file.

Syntax: fileobj.write()

Example:1

```
fp = open(“sample.txt",“w")
fp.write(''Python is the modern day language. It makes things so simple.  It is the fastest-
growing programing language'')
print("{} File created succesfully".format("sample.txt"))
fp.close()
```

<u>2n method with statement:</u>

```
with open("file.txt", "w") as f:
      f.write("Hello World!!!")
      f.close()
```

2. Reading data from the file

- read() method is used to read the content from the file.
- Syntax: fileobject.read()

Write a program to read the data from the file.

```
f=open("sample.txt","r")
content=f.read()

print(content)
f.close()
```

3. Write a program to append given data

```
f=open("sample.txt","a")
f.write("python is used in many application developments")
f.close()
with open("sample.txt","r") as f:
   content=f.read()
   print("File content after append")
   print(content)
   f.close()
```

- The **read()** method reads a string from the file. It can read the data in the text as well as a binary format.
- The syntax of the **read()** method is given below.

fileobj.read(<count>)

write a program to read first 10 characters from the file

```
fp = open(“sample.txt","r")
content = fp.read(10)
 print(type(content))
 print(content)
 fp.close()
```

write a program to read first 10 characters from the file using exception handling.

```
try:
        fp = open(“sample.txt","r")
except IOError: print(“sample.txt is not exist”)
else:
       content = fp.read(10)
       print(type(content))
       print(content)
       fp.close()
```

write a program to read entire file from the file using exception handling.

```
try:
      fp = open(“sample.txt","r")
except IOError: print(“sample.txt is not exist”)
else:
     content = fp.read()
     print(content)
     fp.close()
```

Read Lines of the file

- Python facilitates to read the file line by line by using a function **readline()** method. The **readline()** method reads the lines of the file from the beginning
- Example.

```
try:
        fp=open("sample.txt","r")
except IOError: print("sample.txt is not exist")
else :
    line1=fp.readline()
    line2=fp.readline()
    print(line1)
    print(line2)
    fp.close()
```

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can't be read or write anymore.
2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.
7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.

File.readlines:

Python provides also the **readlines()** method which is used for the reading lines. It returns the list of the lines till the end of **file(EOF)** is reached.

Syntax : File.readlines([sizehint])

Example : Reading Lines Using readlines() function

```
#open the file.txt in read mode. causes error if no such file exists.

fileptr = open("file2.txt","r");
content = fileptr.readlines()
 print(content)
fileptr.close()
```

File.writelines()

- Python file method **writelines()** writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings. There is no return value.
- Syntax : fileObject.writelines(sequence)

```
f = open("demofile.txt", “w")
f.writelines([“python is very powerful language.", “used in DL/ML apps."])
f.close()

#open and read the file after the appending:
f = open("demofile.txt", "r")
print(f.read())
```

Random file Access:

- F.seek()
- Python seek() method is used for changing the current location of the file handle. The file handle is like a cursor, which is used for defining the location of the data which has to be read or written in the file.
- Syntax: fi.seek(offset, from_where), where fi is the file pointer
- Parameters:
- **Offset:** This is used for defining the number of positions to move forward.
- **from_where:** This is used for defining the point of reference.

- **0:** The 0 value is used for setting the whence argument at the beginning of the file.
- **1:** The 1 value is used for setting the whence argument at the current position of the file.
- **2:** The 2 value is used for setting the whence argument at the end of the file.

- **Example1:** (The user has to read the text file, which contains the following text:
- "This is the sentence I am Writing to show the example of the seek() method working in Python."

```
fi = open("text.txt", "r")
# the second parameter of the seek method is by default 0
fi.seek(30)
# now, we will print the current position
print(fi.tell())
print(fi.readline())
fi.close()
```

- **Output:**
- 30 ing to show the example of the seek() method working in Python.

Linecache:

```
import linecache
print linecache.getline(your_file.txt, randomLineNumber) # Note: first line is 1, not 0
```

Example:

```
import linecache as lc
a = lc.getline('File.txt', 2)
print(a)
```

Data Science

Data Science

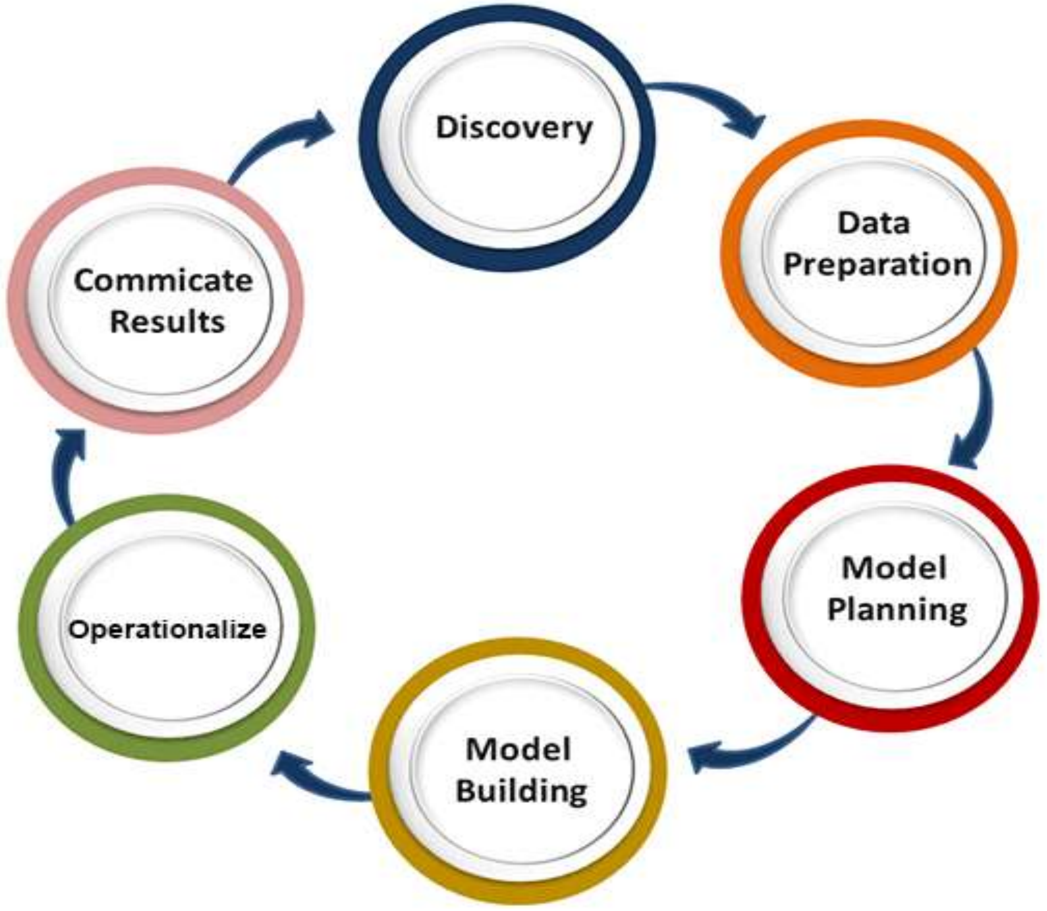
- ➢ Data science is a deep study of the massive amount of data, which involves extracting meaningful insights from raw, structured, and unstructured data that is processed using the scientific method, different technologies, and algorithms.
- ➢ It is a multidisciplinary field that uses tools and techniques to manipulate the data so that we can find something new and meaningful.
- ➢In short, we can say that data science is all about:

1. Asking the correct questions and analyzing the raw data.
2. Modeling the data using various complex and efficient algorithms.
3. Visualizing the data to get a better perspective.
4. Understanding the data to make better decisions and finding the final result.

Data Science

Data Science Lifecycle

- ➢The life-cycle of data science consists of 6 stages.



Data Science

<u>Data Science Lifecycle</u>

1. Discovery:

- ➢The first phase is discovery, which involves asking the right questions.
- ➢When we start any data science project, we need to determine what are the basic requirements, priorities, and project budget.
- ➢In this phase, we need to determine all the requirements of the project such as the number of people, technology, time, data, an end goal, and then we can frame the business problem on first hypothesis level.

Data Science

Data Science Lifecycle

2. Data preparation:

- Data preparation is also known as Data Munging.
- In this phase, we need to perform the following tasks:

1. Data cleaning
2. Data Reduction
3. Data integration
4. Data transformation

- After performing all the above tasks, we can easily use this data for our further processes.

Data Science

Data Science Lifecycle

3. Model Planning:

➢ In this phase, we need to determine the various methods and techniques to establish the relation between input variables.

➢We will apply Exploratory data analytics(EDA) by using various statistical formula and visualization tools to understand the relations between variable and to see what data can inform us.

➢ Common tools used for model planning are:

1. SQL Analysis Services
2. R
3. SAS
4. Python

Data Science

Data Science Lifecycle

4. Model-building:

- ➢In this phase, the process of model building starts.
- ➢We will create datasets for training and testing purpose.
- ➢We will apply different techniques such as association, classification, and clustering, to build the model.
- ➢Following are some common Model building tools:

1. SAS Enterprise Miner
2. WEKA
3. SPCS Modeler
4. MATLAB

Data Science

<u>Data Science Lifecycle</u>

<u>5. Operationalize:</u>

➢In this phase, we will deliver the final reports of the project, along with briefings, code, and technical documents.

➢This phase provides us a clear overview of complete project performance and other components on a small scale before the full deployment.

<u>6. Communicate results:</u>

➢In this phase, we will check if we reach the goal, which we have set on the initial phase. We will communicate the findings and final result with the business team.

Data Visualization

<u>Data Visualization</u>

- Data Visualization is the presentation of data in graphical format.
- It helps in understands the significance of data by summarizing and presenting huge amount of data in a simple and easy-to-understand format and helps communicate information clearly and effectively.
- It enables stakeholders and decision makers to analyze data visually.
- The data in a graphical format allows them to identify new trends and patterns easily.

Data Visualization

- The main benefits of data visualization are as follows:

1. It simplifies the complex quantitative information
2. It helps analyze and explore big data easily
3. It identifies the areas that need attention or improvement
4. It identifies the relationship between data points and variables
5. It explores new patterns and reveals hidden patterns in the data

- Three major considerations for Data Visualization:
- Clarity
- Accuracy
- Efficiency

Data Visualization

Clarity -

Clarity ensures that the data set is complete and relevant.

Accuracy –

Accuracy ensures using appropriate graphical representation to convey the right message.

Efficiency -

Efficiency uses efficient visualization technique which highlights all the data points

Data Visualization

➢some basic factors to be aware of before visualizing the data.

1. Visual effect
2. Coordination System
3. Data Types and Scale
4. Informative Interpretation

Data Visualization

Visual effect -

Visual Effect includes the usage of appropriate shapes, colors, and size to represent the analyzed data.

Coordination System -

The Coordinate System helps to organize the data points within the provided coordinates.

Data Types and Scale -

The Data Types and Scale choose the type of data such as numeric or categorical.

Informative Interpretation –

The Informative Interpretation helps create visuals in an effective and easily interpreted ill manner using labels, title legends, and pointers.

Data Visualization

Python Libraries

➢Python offers multiple great graphing libraries .

➢ Some popular plotting libraries:

1. Matplotlib
2. Pandas Visualization
3. Seaborn
4. ggplot
5. Plotly

Matplotlib

Matplotlib:

➢ matplotlib is a python two-dimensional plotting library for data visualization and creating interactive graphics or plots.

➢ Using pythons matplotlib, the data visualization of large and complex data becomes easy.

matplotlib Advantages

➢ There are several advantages of using matplotlib to visualize data.

1. A multi-platform data visualization tool built on the numpy and sidepy framework. Therefore, it's fast and efficient.
2. It possesses the ability to work well with many operating systems and graphic backends.

Matplotlib

matplotlib Advantages:

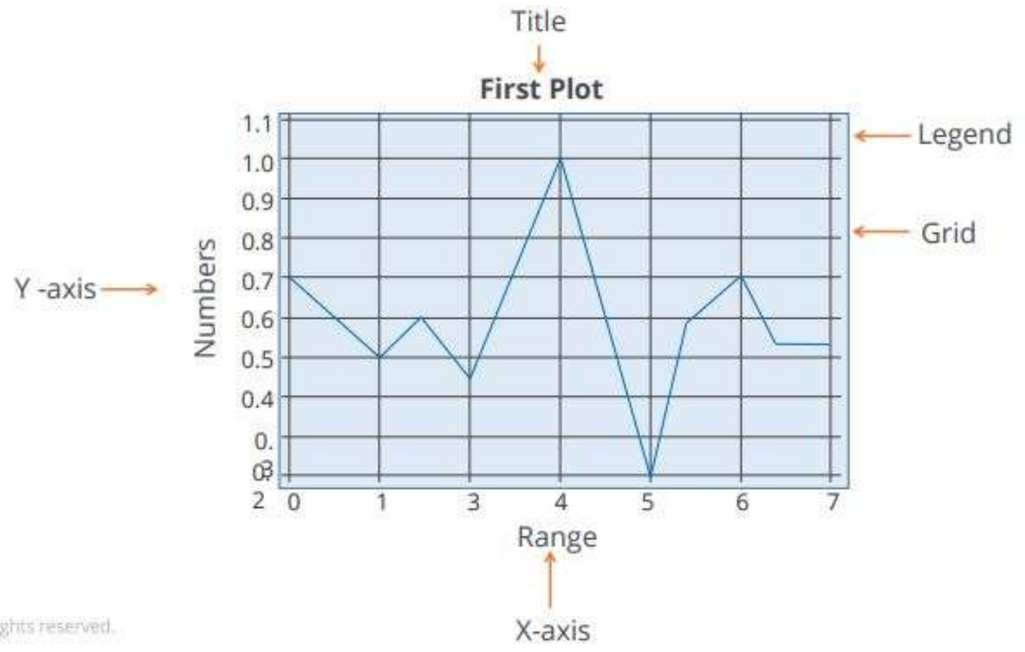
3. It possesses high-quality graphics and plots to print and view for a range of graphs such as histograms, bar charts, pie charts, scatter plots and heat maps.
4. With Jupyter notebook integration, the developers have been free to spend their time implementing features rather than struggling with compatibility.
5. It has large community support and cross-platform support as it is an open source tool.
6. It has full control over graph or plot styles such as line properties, thoughts, and access properties.

Matplotlib

Plot:

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

Example:



➢ This is a line plot of the random numbers on the y-axis and the range on the x-axis. The background of the plot is called a grid. The text first plot denotes the title of the plot and text line one denotes the legend.

Matplotlib

Plot:

- plot() function draws a line from point to point.

  The function takes parameters for specifying points in the diagram.

  Parameter 1 is an array containing the points on the **x-axis**.

  Parameter 2 is an array containing the points on the **y-axis**.

Example:

- To plot a line from (1, 3) to (8, 10). pass two arrays as [1, 8] and [3, 10] to the plot function

Matplotlib

Creating a plot:

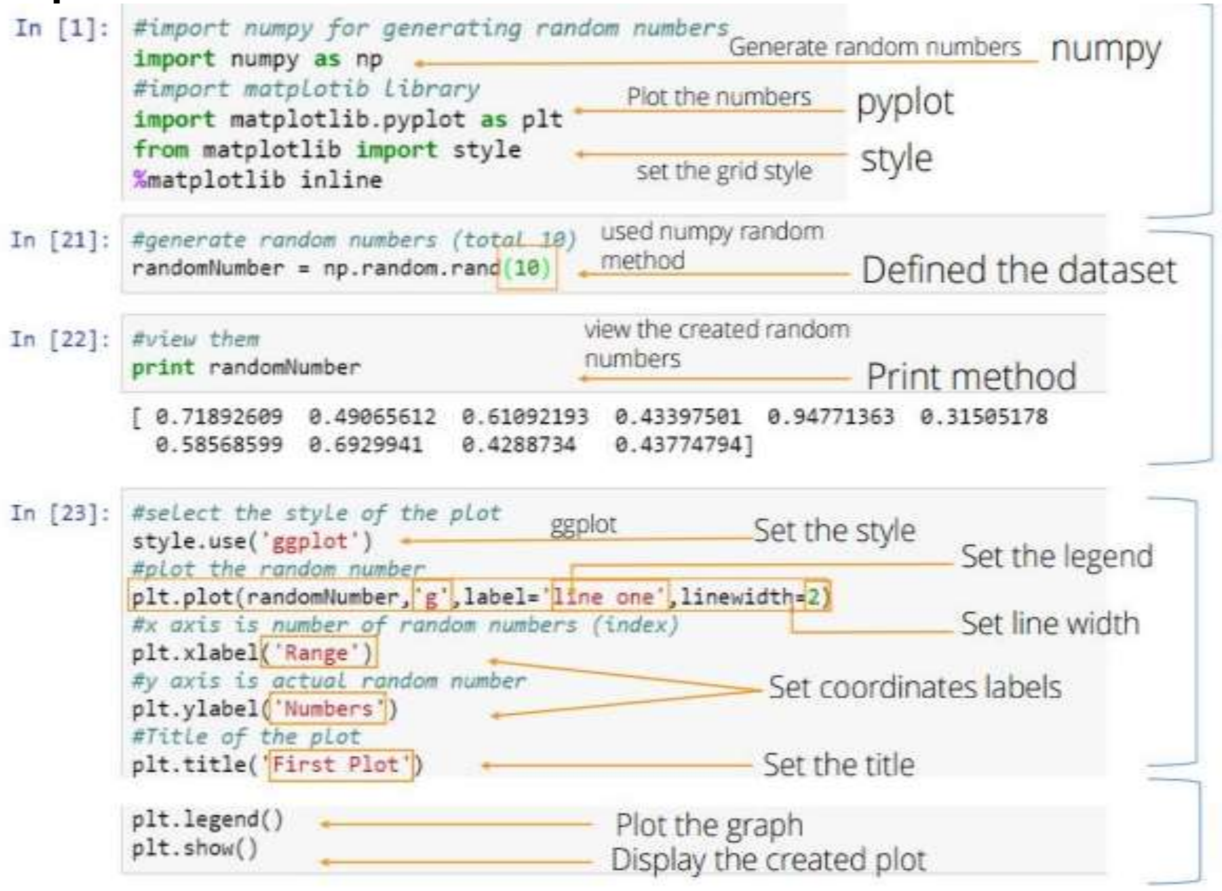
- We can create a plot using four simple steps.

1. Import the required libraries
2. Define or import the required data set
3. Set the plot parameters
4. Display the created plot

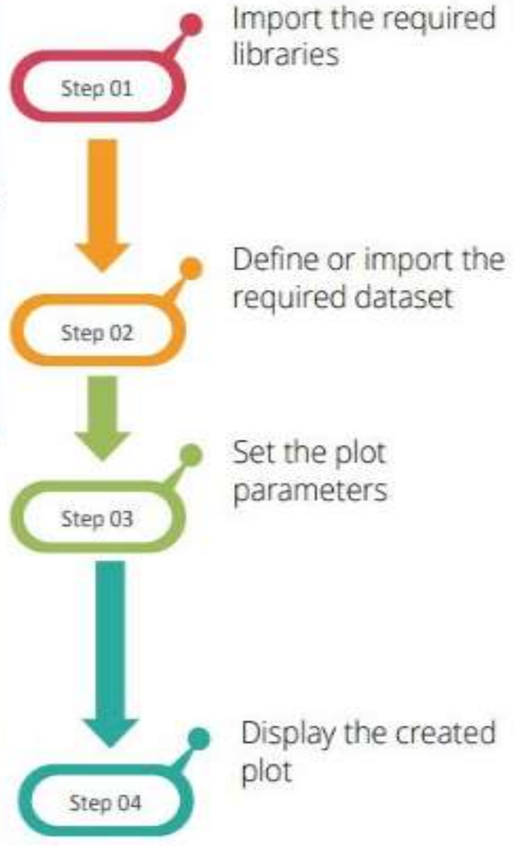
Matplotlib

Creating a plot:

Example:



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

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

In [22]: #view them
         print randomNumber

         [ 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')

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



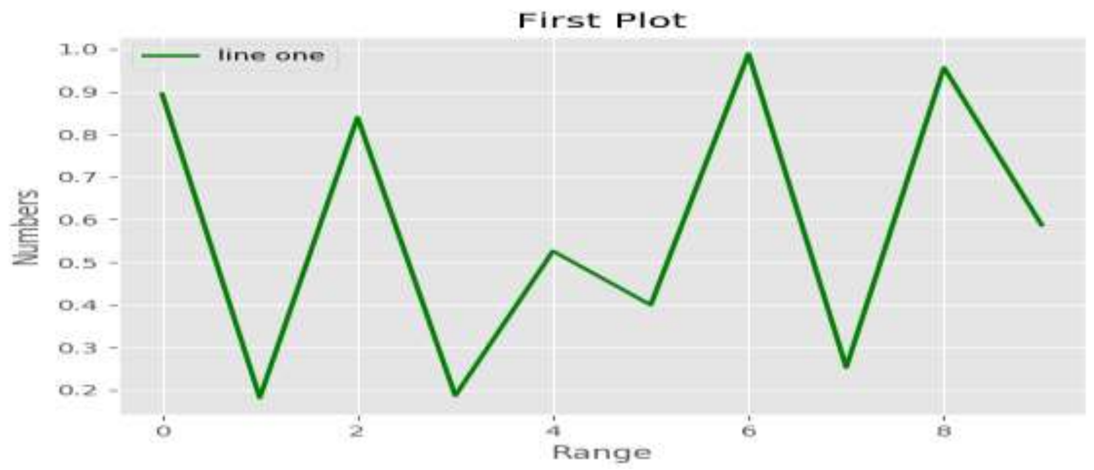
Matplotlib

Creating a plot:

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
r=np.random.rand(10)
style.use('ggplot')
plt.plot(r,'g',label='line one',linewidth=2)
plt.xlabel('Range')
plt.ylabel('Numbers')
plt.title('First Plot')
plt.legend()
plt.show()
```

Output:



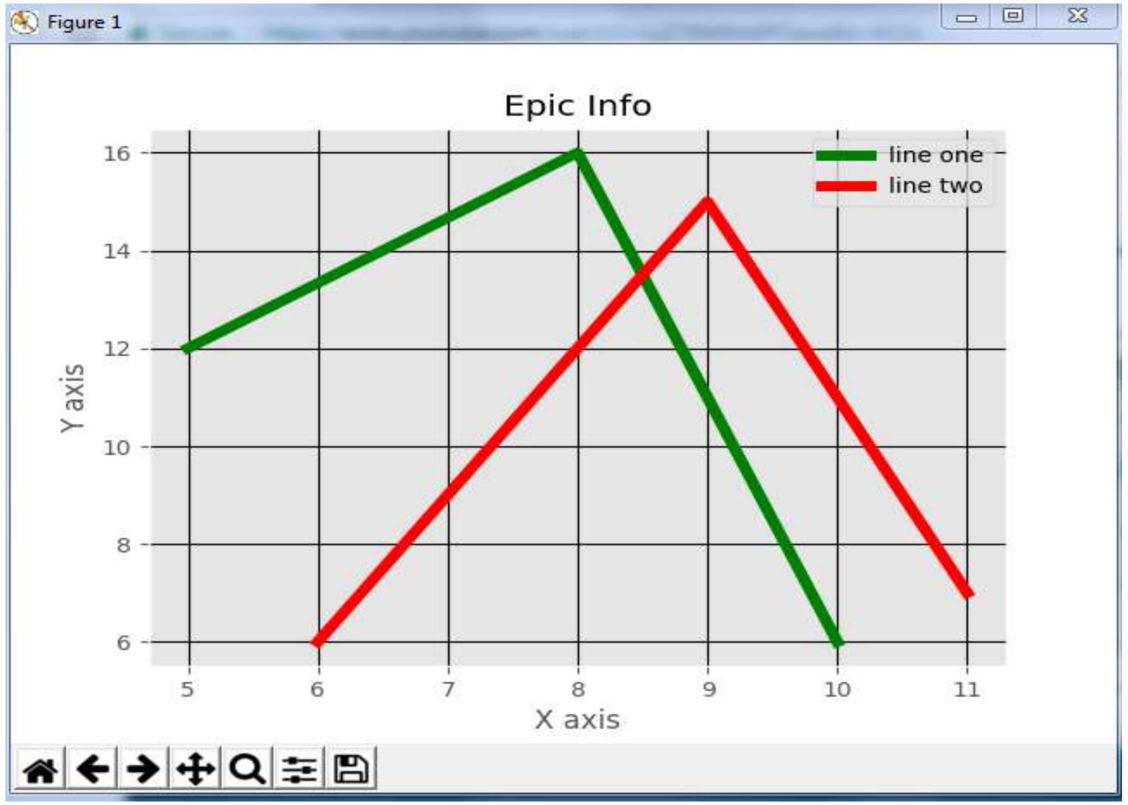
Matplotlib

Creating a plot:

Multiple Plots

```
from matplotlib import pyplot as plt
from matplotlib import style
 style.use('ggplot')
x = [5,8,10] ,y = [12,16,6], x2 = [6,9,11]
y2 = [6,15,7]
plt.plot(x,y,'g',label='line one', linewidth=5)
plt.plot(x2,y2,'c',label='line two',linewidth=5)
plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.legend()
plt.grid(True,color='k')
plt.show()
```

Output:



Matplotlib

Types of Plots:

- The following type of plots can be created using matplotlib.

1. Histogram
2. Scatter Plot
3. Heat Map
4. Pie Chart
5. Error Bar , ….etc

Matplotlib

Scatter Plot:

- ➢ scatter plots are used to compare variables.
- ➢ The data is displayed as a collection of points, each having the value of one variable which determines the position on the horizontal axis and the value of other variable determines the position on the vertical axis.
- ➢ scatter() function is used to draw a scatter plot.
- ➢ It has several advantages:

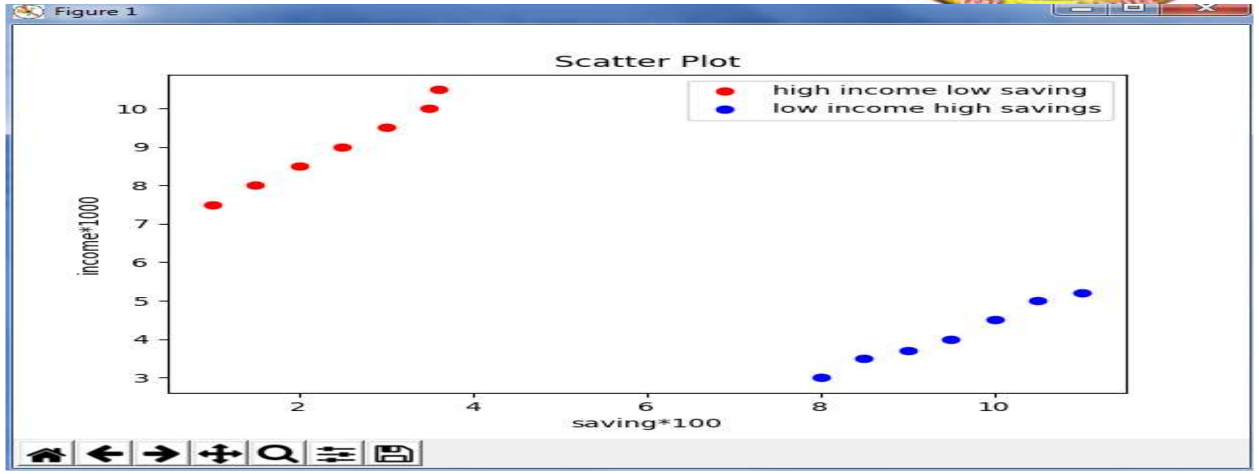
1. It shows the correlation between variables
2. It is suitable for large data sets
3. It is easy to find clusters
4. It is possible to represent each piece of data as a point on the plot.

Matplotlib

Scatter Plot:

Example:

```
import matplotlib.pyplot as plt
x = [1,1.5,2,2.5,3,3.5,3.6]
y = [7.5,8,8.5,9,9.5,10,10.5]
 x1=[8,8.5,9,9.5,10,10.5,11]
y1=[3,3.5,3.7,4,4.5,5,5.2]
 plt.scatter(x,y, label='high income low saving',color='r')
plt.scatter(x1,y1,label='low income high savings',color='b')
plt.xlabel('saving*100')
plt.ylabel('income*1000')
plt.title('Scatter Plot')
plt.legend()
plt.show()
```



Matplotlib

Bar Graph:

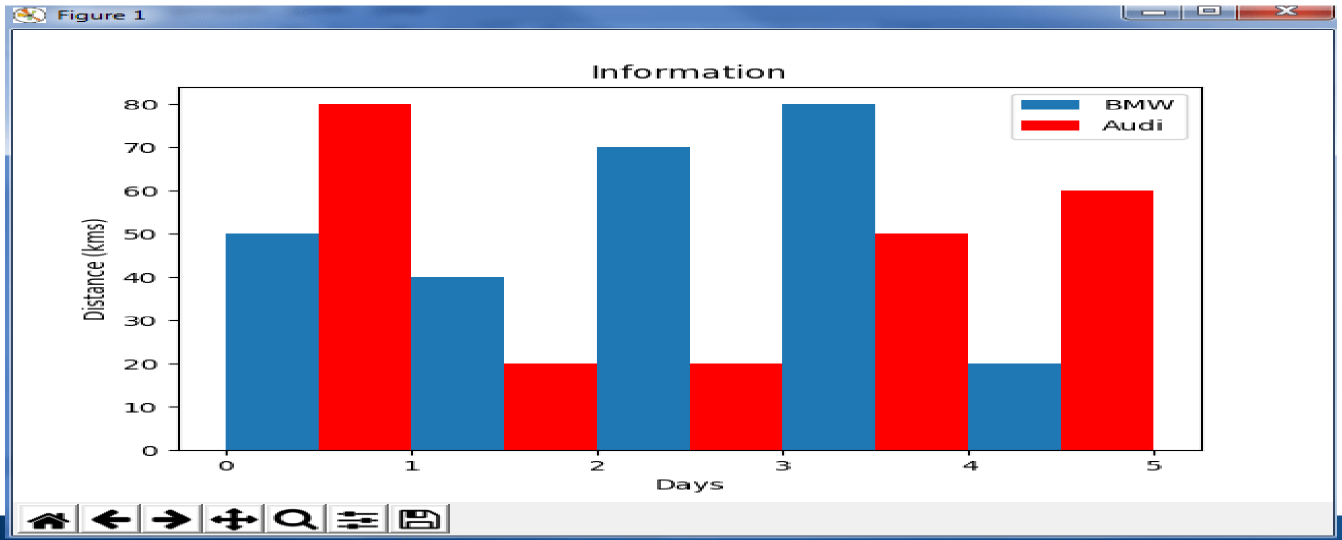
- A bar graph uses bars to compare data among different categories.
- It is well suited when we want to measure the changes over a period of time.
- It can be represented horizontally or vertically.
- the bar() function is used to draw bar graphs.

Matplotlib

Bar Graph:

Example:

```
from matplotlib import pyplot as plt
 plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],
label="BMW",width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],
label="Audi", color='r',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```



Matplotlib

Histograms:

- ➢ Histograms are graphical representations of a probability distribution (normal distribution).
- ➢ histogram is a kind of bar chart.
- ➢ In Matplotlib Histogram is created using the hist() method.
- ➢ The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- ➢ A histogram chart has several advantages.

1. It displays the number of values within a specified interval.
2. It is suitable for large data sets as they can be grouped within the intervals.

Matplotlib

Histograms:

Example:

NumPy is used to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
```

Result:

```
[167.62255766 175.32495609 152.84661337 165.50264047 163.17457988
162.29867872 172.83638413 168.67303667 164.57361342 180.81120541
170.57782187 167.53075749 176.15356275 176.95378312 158.4125473
187.8842668  159.03730075 166.69284332 160.73882029 152.22378865
164.01255164 163.95288674 176.58146832 173.19849526 169.40206527
```

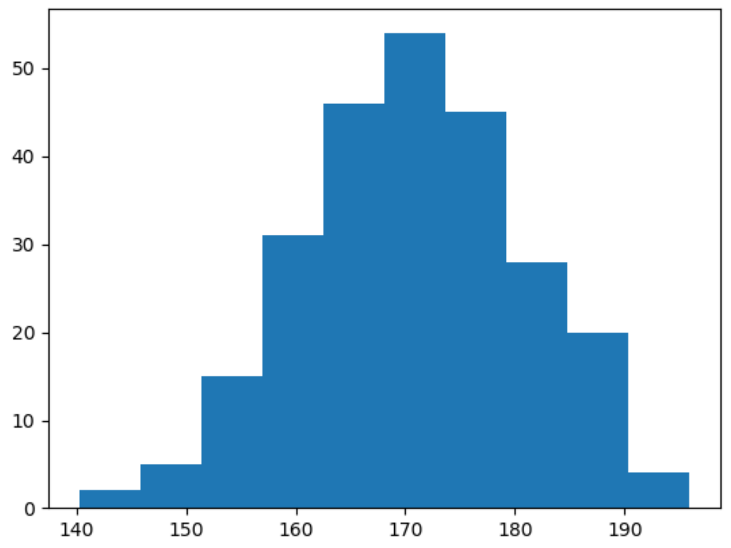
Matplotlib

Histograms:

Example:

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

Output:



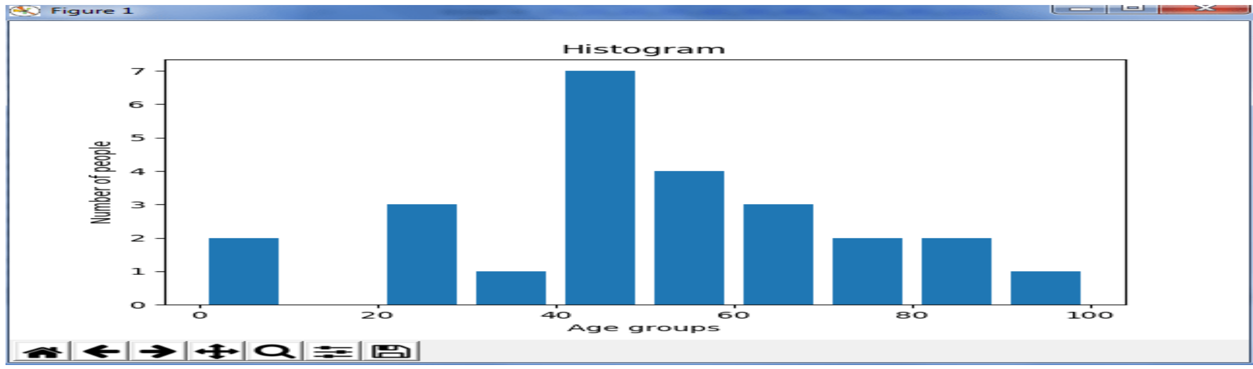
Matplotlib

Histograms:

Example 2:

```
import matplotlib.pyplot as plt
population_age = [22,55,62,45,21,22,34,42,42,4,2,102,95,
85,55,110,120,70,65,55,111,115,80,75,65,54,44,43,42,48]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar',
 rwidth=0.8)
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```

Output:



Matplotlib

Heat Maps

- ➢ A heat map is a better way to visualize two-dimensional data.
- ➢ Using heat maps, we can gain deeper and quicker insight into data than those afforded by other types of plots.
- ➢It has several advantages:

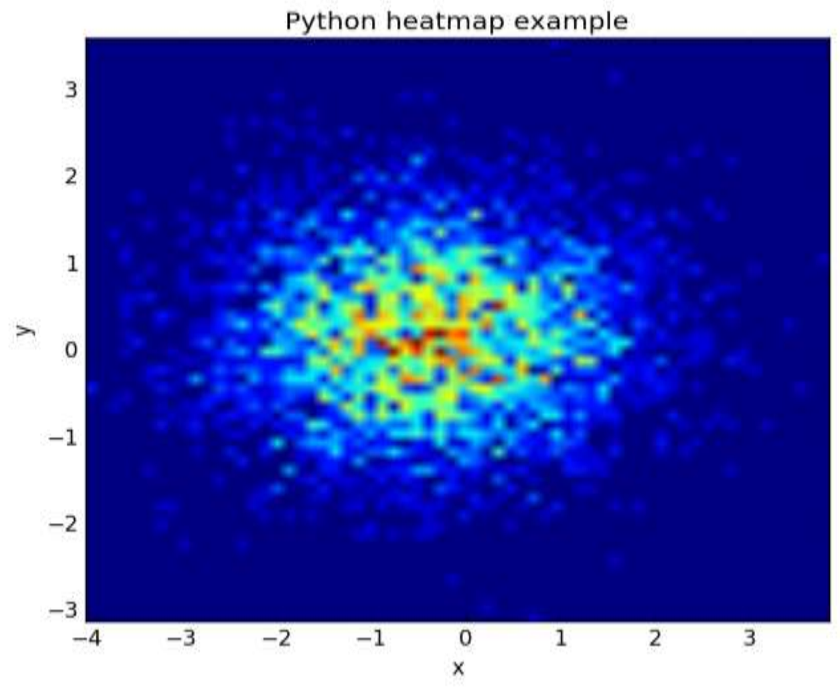
1. It draws attention to the risky-prone area.
2. It uses the entire data set to draw bigger and more meaningful insights.
3. It's used for cluster analysis and deals with large data sets.

Matplotlib

Heat Maps

```
import numpy as np
import numpy.random
import matplotlib.pyplot as plt
# Create data
x = np.random.randn(4096)
y = np.random.randn(4096)
# Create heatmap
heatmap, xedges, yedges = np.histogram2d(x, y, bins=(64,64))
extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
# Plot heatmap
plt.clf()
plt.title('Pythonspot.com heatmap example')
plt.ylabel('y')
plt.xlabel('x')
plt.imshow(heatmap, extent=extent)
plt.show()
```

Output:



Matplotlib

Box Plot:

- A **Box Plot** is also known as **Whisker plot** is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum.
- In the box plot, a box is created from the first quartile to the third quartile, a verticle line is also there which goes through the box at the median.
- Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.
- boxplot() function is used to create box plots.

matplotlib.pyplot.boxplot(data, notch=None, vert=None, patch_artist=None, widths=None)

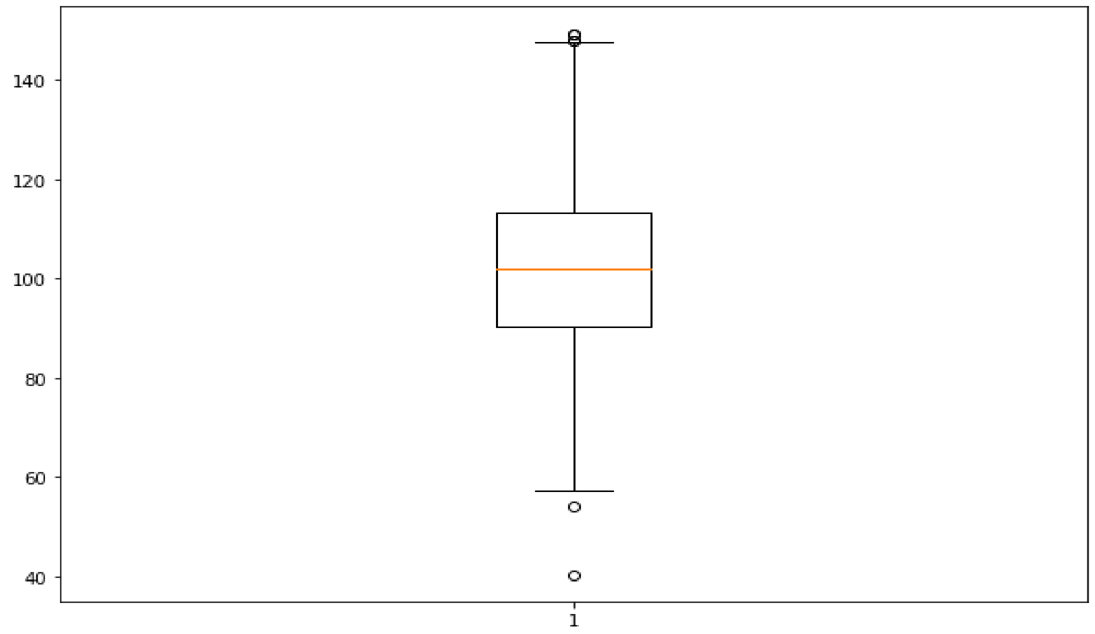
Matplotlib

Box Plot

Example:

```
import matplotlib.pyplot as plt
import numpy as np
 # Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
# Creating plot
plt.boxplot(data)
# show plot
plt.show()
```

Output:



Matplotlib

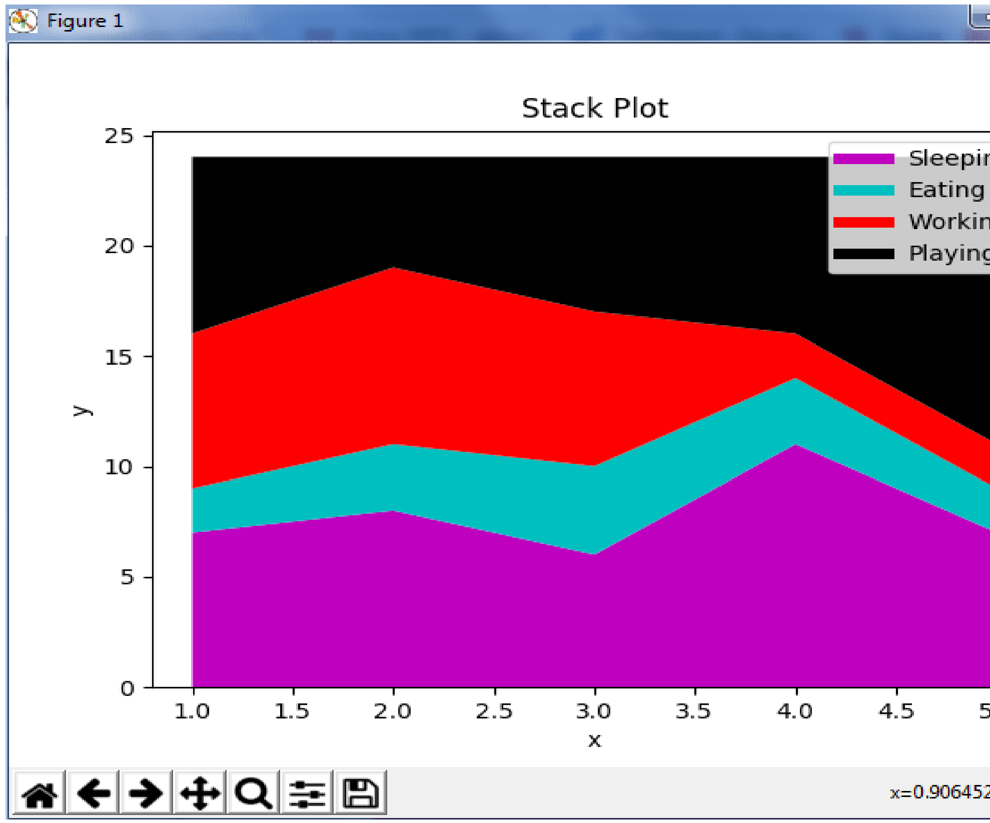
<u>Area Plot:</u>

- ➢ Area plots are pretty much similar to the line plot. They are also known as stack plots.
- ➢These plots can be used to track changes over time for two or more related groups that make up one whole category.
- ➢ For example, let’s compile the work done during a day into categories, say sleeping, eating, working and playing.

Matplotlib

Area Plot

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5],    sleeping =[7,8,6,11,7]
 eating = [2,3,4,3,2] ,  working =[7,8,7,2,2]
 playing = [8,5,7,8,13]
  plt.plot([],[],color='m', label='Sleeping', linewidth=5)
 plt.plot([],[],color='c', label='Eating', linewidth=5)
 plt.plot([],[],color='r', label='Working', linewidth=5)
 plt.plot([],[],color='k', label='Playing', linewidth=5)
  plt.stackplot(days, sleeping,eating,working,playing,
colors=['m','c','r','k'])
  plt.xlabel('x')
 plt.ylabel('y')
 plt.title('Stack Plot')
 plt.legend()
 plt.show()
```



Matplotlib

Pie Chart:

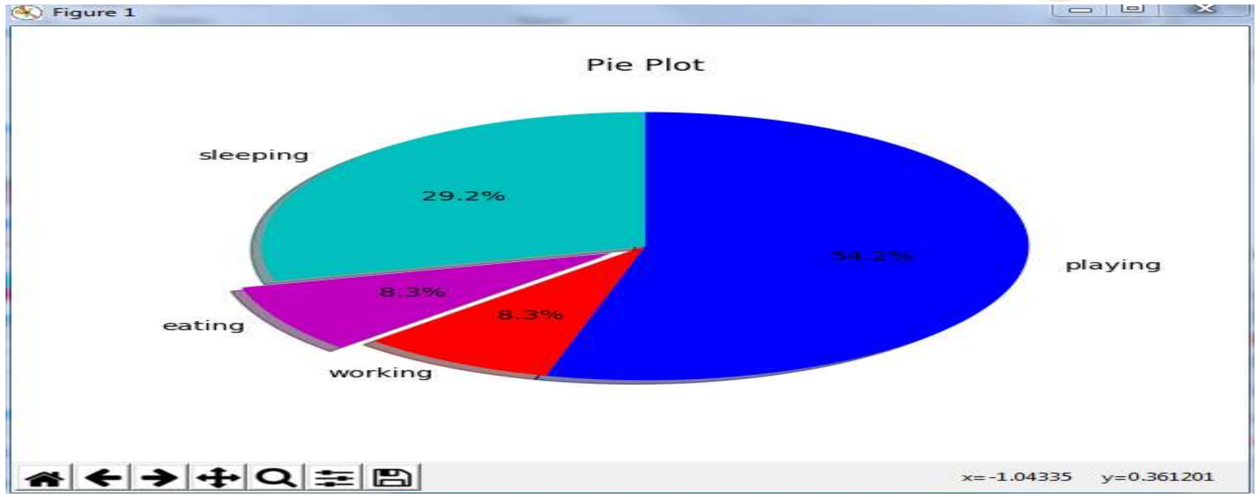
➢ A pie chart refers to a circular graph which is broken down into segments i.e. slices of pie.

➢ It is basically used to show the percentage or proportional data where each slice of pie represents a category.

Matplotlib

Pie Chart

```
import matplotlib.pyplot as plt
 days = [1,2,3,4,5],
sleeping =[7,8,6,11,7]
eating = [2,3,4,3,2]
working =[7,8,7,2,2]
playing = [8,5,7,8,13]
slices = [7,2,2,13]
activities = ['sleeping','eating','working','playing']
cols = ['c','m','r','b']
 plt.pie(slices,   labels=activities,  colors=cols,
  startangle=90,  shadow= True,  explode=(0,0.1,0,0),
  autopct='%1.1f%%')
 plt.title('Pie Plot')
plt.show()
```



Matplotlib

<u>Subplots</u>

- A subplot is used to display multiple plots in the same window.
- Subplot arranges plots in a regular grid.
- To draw the subplots specify the number of rows, columns, and plot.
- The syntax for subplot is shown below.

```
subplot(m,n,p).
```

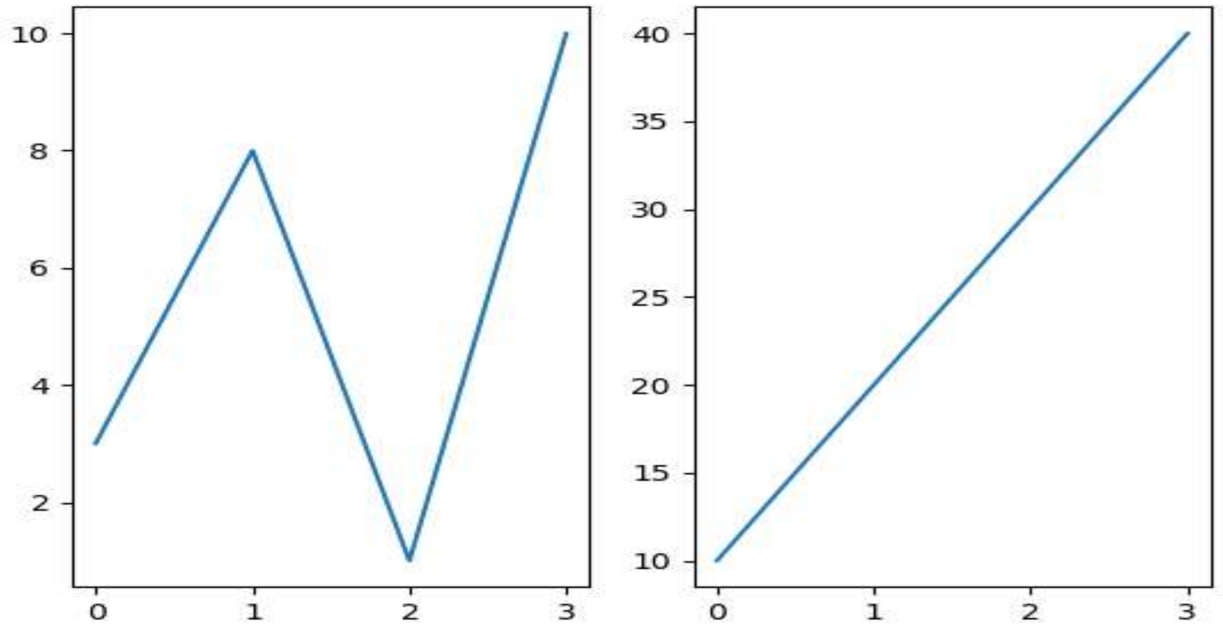
- It divides the current window into an m by n grid and creates an axis for the subplot in the position specified by p.
- For example, Subplot(2,1,2) creates two subplots, which are stacked vertically on a grid.

  If we want to plot four graphs in one window, then the syntax used should be Subplot(2,1,4).

Matplotlib

Subplots

```
import matplotlib.pyplot as plt
 #plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
 plt.subplot(1, 2, 1)
plt.plot(x,y)
 #plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
 plt.subplot(1, 2, 2)
plt.plot(x,y)
 plt.show()
```



Turtle graphics

- Turtle is a special feathers of Python.
- Using Turtle, we can easily draw in a drawing board.

Turtle graphics

turtle methods:

Commonly used turtle methods are :

METHOD	PARAMETER	DESCRIPTION
Turtle()	None	It creates and returns a new turtle object
forward()	amount	It moves the turtle forward by the specified amount
backward()	amount	It moves the turtle backward by the specified amount
right()	angle	It turns the turtle clockwise
left()	angle	It turns the turtle counter clockwise
penup()	None	It picks up the turtle's Pen
pendown()	None	Puts down the turtle's Pen

Turtle graphics

turtle methods:

METHOD	PARAMETER	DESCRIPTION
up()	None	Picks up the turtle’s Pen
down()	None	Puts down the turtle’s Pen
color()	Color name	Changes the color of the turtle’s pen
fillcolor()	Color name	Changes the color of the turtle will use to fill a polygon
heading()	None	It returns the current heading
position()	None	It returns the current position
goto()	x, y	It moves the turtle to position x,y
begin_fill()	None	Remember the starting point for a filled polygon

Turtle graphics

turtle methods:

METHOD	PARAMETER	DESCRIPTION
end_fill()	None	It closes the polygon and fills with the current fill color
dot()	None	Leaves the dot at the current position
stamp()	None	Leaves an impression of a turtle shape at the current location
shape()	shapename	Should be ‘arrow’, ‘classic’, ‘turtle’ or ‘circle’

Turtle graphics

Plotting using Turtle(or)Executing a Turtle Program

➢The following 4 steps are used for executing a turtle program :

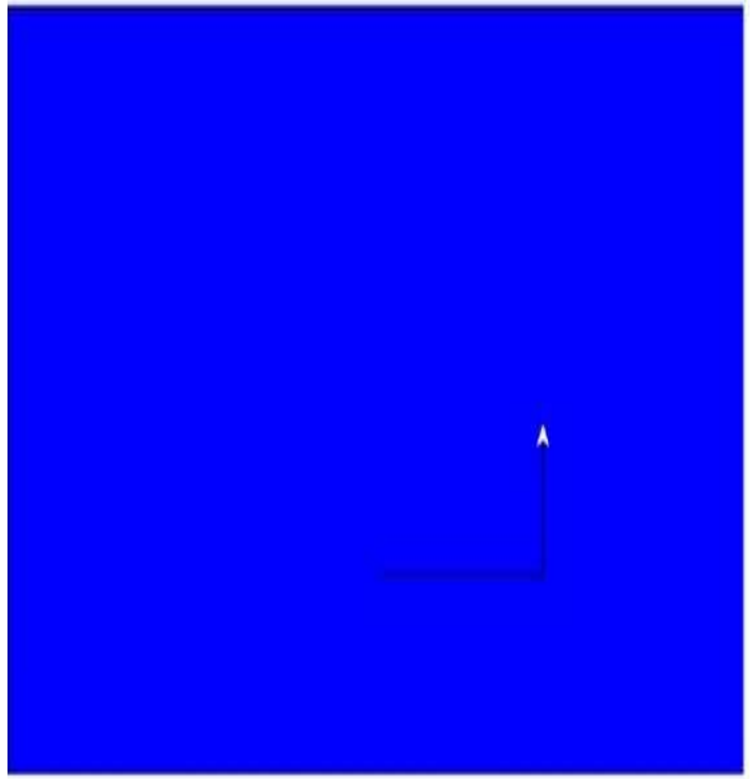
1. Import the turtle module
2. Create a turtle to control.
3. Draw around using the turtle methods.
4. Run turtle.done().

Turtle graphics

Examples

creates a graphics window

```
 import turtle
 my_window = turtle.Screen()
my_window.bgcolor("blue")
my_pen = turtle.Turtle()
my_pen.forward(150)
my_pen.left(90)
my_pen.forward(75)
my_pen.color("white")
my_pen.pensize(12)
```



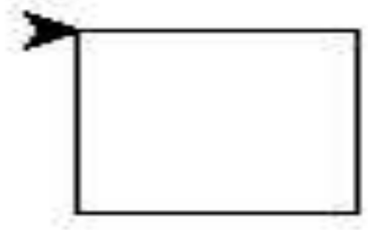
Turtle graphics

Examples

Example 2

Draw a Square

```
import turtle
my_pen = turtle.Turtle()
for i in range(4):
   my_pen.forward(50)
   my_pen.right(90)
turtle.done()
```



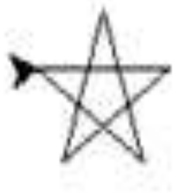
Turtle graphics

Examples

Example3

Draw a star

```
import turtle
my_pen = turtle.Turtle()
for i in range(50):
   my_pen.forward(50)
   my_pen.right(144)
turtle.done()
```



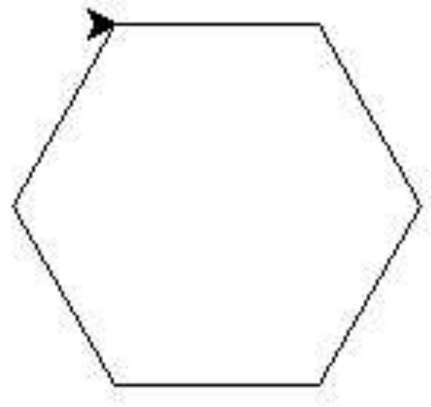
Turtle graphics

Examples

Example 4

Draw a Hexagon

```
import turtle
polygon = turtle.Turtle()
my_num_sides = 6
my_side_length = 70
my_angle = 360.0 / my_num_sides
for i in range(my_num_sides):
   polygon.forward(my_side_length)
   polygon.right(my_angle)
turtle.done()
```

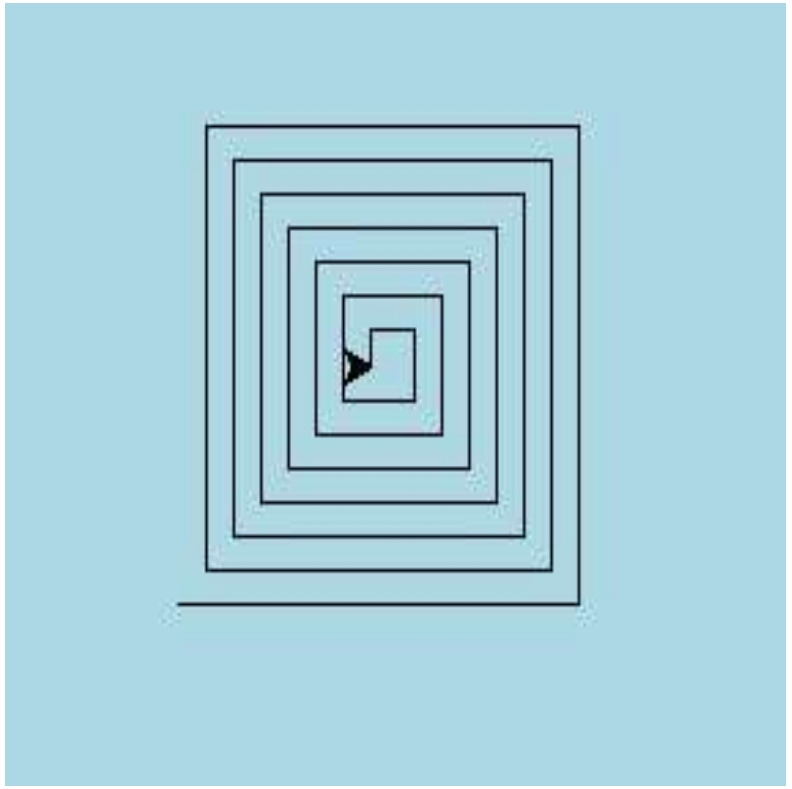


Turtle graphics

Example 5

Draw a square inside another square box.

```
import turtle
my_wn = turtle.Screen()
my_wn.bgcolor("light blue")
my_pen = turtle.Turtle()
my_pen.color("black")
def my_sqrfunc(size):
  for i in range(4):
    my_pen.fd(size)
    my_pen.left(90)
    size = size - 5
my_sqrfunc(146)
my_sqrfunc(126)
my_sqrfunc(106)
my_sqrfunc(86)
my_sqrfunc(66)
my_sqrfunc(46)
my_sqrfunc(26)()
```

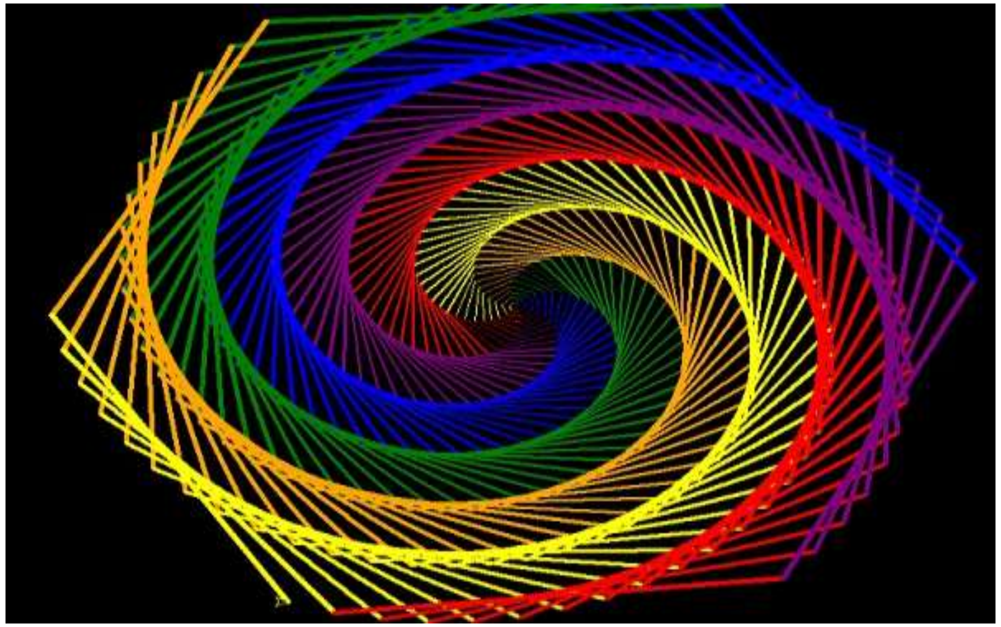


Turtle graphics

Example 6

Drawing a pattern

```
import turtle
colors = [ "red","purple","blue","green","orange","yellow"]
my_pen = turtle.Pen()
turtle.bgcolor("black")
for x in range(360):
   my_pen.pencolor(colors[x % 6])
   my_pen.width(x/100 + 1)
   my_pen.forward(x)
   my_pen.left(59)
```



UNIT 3

Introduction to NumPy, Pandas, Matplotlib.

Exploratory Data Analysis (EDA), Data Science life cycle, Descriptive Statistics, Basic tools (plots, graphs and summary statistics) of EDA, Philosophy of EDA. Data Visualization: Scatter plot, bar chart, histogram, boxplot, heat maps, etc.

Data Analysis

Data Analysis

Data Analysis is a process of inspecting, cleaning, transforming, and modeling data to discover useful information for business decision-making.

Steps for Data Analysis, Data Manipulation and Data Visualization:

1. Transform Raw Data in a Desired Format
2. Clean the Transformed Data (Step 1 and 2 also called as a Pre-processing of Data)
3. Prepare a Model
4. Analyze Trends and Make Decisions

NumPy

Why do we need NumPy ?

- Why NumPy is required if python lists are already there.
- operations can’t applied on all the elements of two list directly.

Example:

```
list1 = [1, 2, 3, 4 ,5, 6]
list2 = [10, 9, 8, 7, 6, 5]
print(list1*list2)
```

Output:

```
        TypeError: can't multiply sequence by non-int of type 'list'
```

- To overcome this problem NumPy is used.

NumPy

What is NumPy?

- NumPy stands for 'Numerical Python' or 'Numeric Python'.
- It is an open source module of Python which provides fast mathematical computation on arrays and matrices.
- NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation.

NumPy

Use Of NumPy

- Some of uses of NumPY are:

1. To represent Multi dimensional array
2. Methods for processing arrays
3. Element by element operations
4. Mathematical operations like logical, Fourier transform, shape manipulation, linear algebra and random number generation

NumPy

NumPy

- An array class in Numpy is called as **ndarray**.
- Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.
- In numpy dimensions are called as axes.

NumPy

To use numpy import numpy module in to program as follows:

```
import numpy as np
```

NumPy

Creating a Numpy Array

- Arrays in Numpy can be created by multiple ways.

1. Creating array from list

Example:

```
a = np.array([[1, 2, 4], [5, 8, 7]])
print ("Array created using passed list:\n", a)
```

output:

```
Array created using passed list: [1, 2, 4] [5, 8, 7]
```

NumPy

2. Creating array from tuple

Example:

```
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)
```

output:

```
Array created using passed tuple: [1, 3, 2]
```

3. Creating a 3X4 array with all zeros

Example:

```
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)
```

output:

```
An array initialized with all zeros:
[ [ 0.,  0.,  0.,  0.],      [ 0.,  0.,  0.,  0.],      [ 0.,  0.,  0.,  0.]]
```

NumPy

4. Create a constant value array of complex type

Example:

```
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s.“)
Print("Array type is complex:\n", d)
```

Output:

```
An array initialized with all 6s.Array type is complex:
[[ 6.+0.j  6.+0.j  6.+0.j]

 [ 6.+0.j  6.+0.j  6.+0.j]

 [ 6.+0.j  6.+0.j  6.+0.j]]
```

NumPy

5. Create an array with random values

Example:

```
e = np.random.random((2, 2))
print ("\nA random array:\n", e)
```

output:

```
A random array:

 [[ 0.2721846   0.61584512]

 [ 0.01898119  0.2220523 ]]
```

NumPy

6. Create a sequence of integers from 0 to 30 with steps of 5

Example:

```
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)
```

output:

```
A sequential array with steps of 5:

 [ 0,  5 ,10,15, 20, 25]
```

NumPy

7. Create a sequence of 10 values in range 0 to 5

Example:

```
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between 0 and 5:\n", g)
```

Output:

```
A sequential array with 10 values between0 and 5:

 [ 0.          0.55555556  1.11111111  1.66666667  2.22222222  2.77777778

  3.33333333  3.88888889  4.44444444  5.        ]
```

NumPy

8. Reshaping 3X4 array to 2X2X3 array

Example:

```
arr = np.array([[1, 2, 3, 4],[5, 2, 4, 2],
                              [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)
```

Output:

```
Original array:
  [[1 2 3 4]
 [5 2 4 2]

 [1 2 0 1]]
 Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]
```

NumPy

9. Flatten array

Example:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

Output:

```
Original array:
[[1 2 3]

 [4 5 6]]

Fattened array:

 [1 2 3 4 5 6]
```

NumPy

Access Array Elements

- Array indexing is the same as accessing an array element.
- array indexing is important for analysing and manipulating the array object.
- NumPy provides 3 different ways for array indexing.

1. Slicing
2. Integer array indexing
3. Boolean array indexing

NumPy

1. Slicing:

- NumPy arrays can be sliced.
- As arrays can be multidimensional, specify slicing for each dimension of the array.

Example:

```
arr = np.array([[-1, 2, 0, 4], [4, -0.5, 6, 0], [2.6, 0, 7, 8], [3, -7, 4, 2.0]])
# Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate columns(0 and 2):\n", temp)
```

Output:

```
Array with first 2 rows and alternate columns(0 and 2):
[[-1.  0.] [ 4.  6.]]
```

Multidimensional Slicing in NumPy Array

```
import numpy as np
array2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("-" * 10)
print(array2d[:, 0:2])
print("-" * 10)
print(array2d[1:3, 0:3])
print("-" * 10)
print(array2d[-1::-1, -1::-1])
```

- **Sample output of above program.**
-
```
----------
[[1 2]
[4 5]
[7 8]]
----------
[[4 5 6]
[7 8 9]]
----------
[[9 8 7]
[6 5 4]
[3 2 1]]
```

NumPy

2. Integer array indexing:

- lists are passed for indexing for each dimension.
- One to one mapping of corresponding elements is done to construct a new arbitrary array.

Example:

```
arr = np.array([[-1, 2, 0, 4], [4, -0.5, 6, 0], [2.6, 0, 7, 8], [3, -7, 4, 2.0]])
# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),(3, 0):\n", temp)
```

Output:

```
Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [ 4.  6.  0.  3.]
```

NumPy

3. Boolean array indexing

- This method is used to pick elements from array which satisfy some condition.

Example:

```
arr = np.array([[-1, 2, 0, 4], [4, -0.5, 6, 0], [2.6, 0, 7, 8], [3, -7, 4, 2.0]])
# boolean array indexing example
cond = arr > 0 # cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 0:\n", temp)
```

Output:

```
Elements greater than 0:
[ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

NumPy

Basic operations

- Some of the basic operations are:

1. Operations on single array
2. Operations using Unary operators
3. Operations using Binary operators
4. Sorting array:

NumPy

Basic operations

Operations on single array:

- arithmetic operators can be used to do element wise operation on array to create a new array.
- In case of +=, -=, *= operators, the existing array is modified.

Example:

```
# Python program to demonstrate  basic operations on single array
import numpy as np
a = np.array([1, 2, 5, 3])
# add 1 to every element
print "Adding 1 to every element:", a+1
```

Output:

```
Adding 1 to every element:
[2 3 6 4]
```

NumPy

Basic operations

Operations on single array:

```
a = np.array([1, 2, 5, 3])
# subtract 3 from each element
print "Subtracting 3 from each element:", a-3
Output:  Subtracting 3 from each element: [-2 -1  2  0]
# multiply each element by 10
print "Multiplying each element by 10:", a*10
Output: Multiplying each element by 10: [10 20 50 30]
# square each element
print "Squaring each element:", a**2
Output:   Squaring each element: [ 1  4 25  9]
```

NumPy

Basic operations

Operations on single array:

```
a = np.array([1, 2, 5, 3])
# modify existing array
a *= 2
print "Doubled each element of original array:", a
```

Output:

```
Doubled each element of original array: [ 2  4 10  6]
```

NumPy

Basic operations

Operations on single array:

```
# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])
print "\nOriginal array:\n", a
print "Transpose of array:\n", a.T
```

Output:

```
Original array:
[[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
[[1 3 9]
 [2 4 6]
 [3 5 0]]
```

NumPy

Basic operations

2. Unary operators:

➢ Many unary operations are provided as a method of **ndarray** class. Some of them are sum, min, max, etc.

➢ These functions can also be applied row-wise or column-wise by setting an axis parameter.

Example:

```
 import numpy as np
 arr = np.array([[1, 5, 6],[4, 7, 2],[3, 1, 9]])
 # maximum element of array
print "Largest element is:", arr.max()
```

Output:

```
 Largest element is: 9
```

NumPy

Basic operations

Unary operators:

```
arr = np.array([[1, 5, 6],[4, 7, 2],[3, 1, 9]])
print "Row-wise maximum elements:“,arr.max(axis = 1)
 Output:
Row-wise maximum elements: [6 7 9]
# minimum element of array
print "Column-wise minimum elements:",arr.min(axis = 0)
 Output:
Column-wise minimum elements: [1 1 2]
```

NumPy

Basic operations

Unary operators:

```
arr = np.array([[1, 5, 6],[4, 7, 2],[3, 1, 9]])
 # sum of array elements
print "Sum of all array elements:",arr.sum()
 Output:
Sum of all array elements: 38
# cumulative sum along each row
print "Cumulative sum along each row:\n",arr.cumsum(axis = 1)
Output:
Cumulative sum along each row:
[[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

NumPy

Basic operations

3. Binary operators:

- all basic arithmetic operators like +, -, /, , *etc can be used.*
- These operations apply on array elementwise and a new array is created.

Example:

```
 a = np.array([[1, 2], [3, 4]])
b = np.array([[4, 3],[2, 1]])
 # add arrays
print "Array sum:\n", a + b
```

Output:

```
Array sum:
[[5 5]
 [5 5]]
```

NumPy

Basic operations

3. Binary operators:

Example:

```
 a = np.array([[1, 2], [3, 4]])
b = np.array([[4, 3],[2, 1]])
 # multiply arrays (elementwise multiplication)
print "Array multiplication:\n", a*b
```

Output:

```
Array multiplication:
[[4 6]
 [6 4]]
```

NumPy

Basic operations

3. Binary operators:

Example:

```
 a = np.array([[1, 2], [3, 4]])
b = np.array([[4, 3],[2, 1]])
 # matrix multiplication
print "Matrix multiplication:\n", a.dot(b)
```

Output:

```
Matrix multiplication:
[[ 8  5]
 [20 13]]
```

NumPy

Basic operations

4. Sorting array:

- **np.sort** method is used for sorting NumPy arrays

Example :

```
import numpy as np
 a = np.array([[1, 4, 2],[3, 4, 6],[0, -1, 5]])
 # sorted array
print ("Array elements in sorted order:\n",np.sort(a, axis = None))
```

output:

```
 Array elements in sorted order:
[-1  0  1  2  3  4  4  5  6]
```

NumPy

Basic operations

4. Sorting array:

Example :

```
import numpy as np
 a = np.array([[1, 4, 2],[3, 4, 6],[0, -1, 5]])
 # sort array row-wise
print "Row-wise sorted array:\n",np.sort(a, axis = 1)
```

output:

```
 Row-wise sorted array:
[[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
```

NumPy

Basic operations

4. Sorting array:

Example :

```
import numpy as np
 a = np.array([[1, 4, 2],[3, 4, 6],[0, -1, 5]])
# specify sort algorithm
print "Column wise sort by applying merge-sort:\n",np.sort(a, axis = 0,
kind = 'mergesort')
```

***#kind :** [‘quicksort’{default}, ‘mergesort’, ‘heapsort’]Sorting algorithm*

output:

```
 Column wise sort by applying merge-sort:
[[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]
```

- # Python program to demonstrate to
- # sorting numbers in descending Order
- # Creating List of Numbers
- numbers = [1, 3, 4, 2]
- # Sorting list of Integers in descending
- numbers.sort(reverse = True)
- print(numbers)

NumPy

NumPy – Mathematical Functions

- NumPy contains a large number of various mathematical operations.
- NumPy provides standard trigonometric functions, functions for arithmetic operations, handling complex numbers, etc

Arithmetic Functions

Function	Description
add(arr1, arr2,..)	Add arrays element wise
reciprocal(arr)	Returns reciprocal of elements of the argument array
negative(arr)	Returns numerical negative of elements of an array
multiply(arr1,arr2,…)	Multiply arrays element wise
divide(arr1,arr2)	Divide arrays element wise
power(arr1,arr2)	Return the first array with its each of its elements raised to the power of elements in the second array (element wise)
subtract(arr1,arr2,…)	Subtract arrays element wise
true_divide(arr1,arr2)	Returns true_divide of an array element wise
floor_divide(arr1,arr2)	Returns floor after dividing an array element wise
float_power(arr1,arr2)	Return the first array with its each of its elements raised to the power of elements in the second array (elementwise)
fmod(arr1,arr2)	Returns floor of the remainder after division elementwise
mod(arr1,arr2)	Returns remainder after division elementwise
remainder(arr1,arr2)	Returns remainder after division elementwise
divmod(arr1,arr2)	Returns remainder and quotient after division elementwise

Example:

```
import numpy as np
a = np.array([10,20,30])
b= np.array([1,2,3])
print("addition of a and b :",np.add(a,b))
print("multiplication of a and b :",np.multiply(a,b))
print("subtraction of a and b :",np.subtract(a,b))
print("a raised to b is:",np.power(a,b))
```

Output:

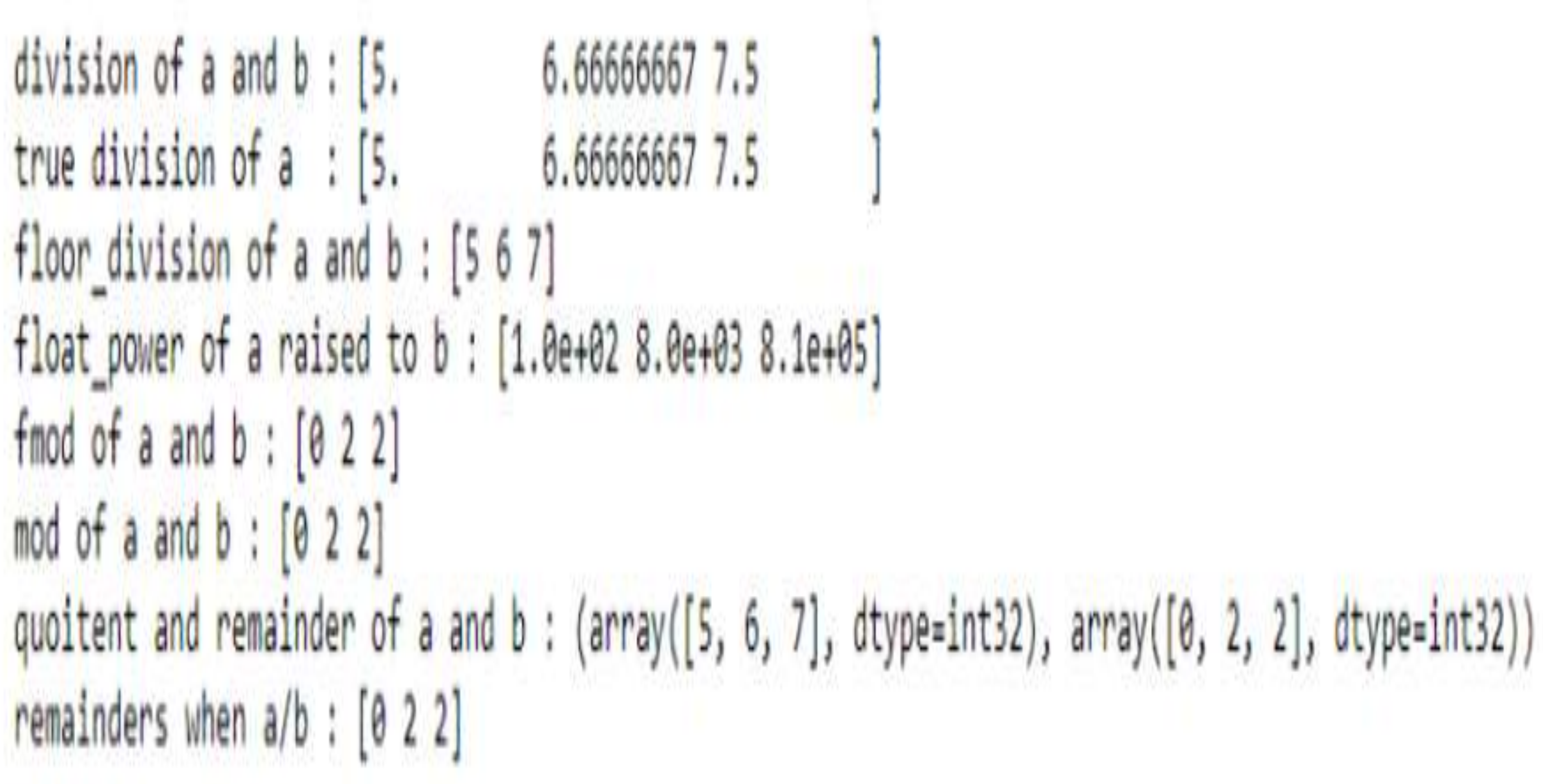
```
addition of a and b : [11 22 33]
multiplication of a and b : [10 40 90]
subtraction of a and b : [ 9 18 27]
a raised to b is: [    10   400 27000]
```

NumPy

Example:

```
import numpy as np
a = np.array([10,20,30])
b= np.array([2,3,4])
print("division of a and b :",np.divide(a,b))
print("true division of a  :",np.true_divide(a,b))
print("floor_division of a and b :",np.floor_divide(a,b))
print("float_power of a raised to b :",np.float_power(a,b))
print("fmod of a and b :",np.fmod(a,b))
print("mod of a and b :",np.mod(a,b))
print("quotient and remainder of a and b :",np.divmod(a,b))
print("remainders when a/b :",np.remainder(a,b))
```

Output



```
division of a and b : [5.         6.66666667 7.5       ]
true division of a  : [5.         6.66666667 7.5       ]
floor_division of a and b : [5 6 7]
float_power of a raised to b : [1.0e+02 8.0e+03 8.1e+05]
fmod of a and b : [0 2 2]
mod of a and b : [0 2 2]
quoitent and remainder of a and b : (array([5, 6, 7], dtype=int32), array([0, 2, 2], dtype=int32))
remainders when a/b : [0 2 2]
```

2. Trigonometric Functions

Function	Description
sin(arr)	Returns trigonometric sine element wise
cos(arr)	Returns trigonometric cos element wise
tan(arr)	Returns trigonometric tan element wise
arcsin(arr)	Returns trigonometric inverse sine element wise
arccos(arr)	Returns trigonometric inverse cosine element wise
arctan(arr)	Returns trigonometric inverse tan element wise
hypot(a,b)	Returns hypotenuse of a right triangle with perpendicular and base as arguments

degrees(arr)
rad2deg(arr)
	Covert input angles from radians to degrees

radians(arr)
deg2rad(arr)
	Covert input angles from degrees to radians

Example:

```
import numpy as np
angles = np.array([0,np.pi/2, np.pi])     sin_angles = np.sin(angles)
cosine_angles = np.cos(angles)
tan_angles = np.tan(angles)
rad2degree = np.degrees(angles)
print("sin of angles:",sin_angles)
print("cosine of angles:",cosine_angles)
print("tan of angles:",tan_angles)
print("angles in radians",rad2degree)
```

Output:

```
sin of angles: [  0.00000000e+00   1.00000000e+00   1.22464680e-16]
cosine of angles: [  1.00000000e+00   6.12323400e-17  -1.00000000e+00]
tan of angles: [  0.00000000e+00   1.63312394e+16  -1.22464680e-16]
angles in radians [   0.   90.  180.]
```

3. Logarithmic and Exponential Functions

Function	Description
exp(arr)	Returns exponential of an input array element wise
expm1(arr)	Returns exponential exp(x)-1 of an input array element wise
exp2(arr)	Returns exponential 2**x of all elements in an array
log(arr)	Returns natural log of an input array element wise
log10(arr)	Returns log base 10 of an input array element wise
log2(arr)	Returns log base 2 of an input array element wise
logaddexp(arr)	Returns logarithm of the sum of exponentiations of all inputs
logaddexp2(arr)	Returns logarithm of the sum of exponentiations of the inputs in base 2

Code:

```
import numpy as np
a = np.array([1,2,3,4,5])
a_log = np.log(a)
a_exp = np.exp(a)
print("log of input array a is:",a_log)
print("exponent of input array a is:",a_exp)
```

Output:

```
log of input array a is: [ 0.         0.69314718  1.09861229  1.38629436
1.60943791]
exponent of input array a is: [   2.71828183    7.3890561    20.08553692
54.59815003  148.4131591 ]
```

4. Rounding Functions

Function	Description
around(arr,decimal)	Rounds the elements of an input array upto given decimal places
round_(arr,decimal)	Rounds the elements of an input array upto given decimal places
rint(arr)	Round the elements of an input array to the nearest integer towards zero
fix(arr)	Round the elements of an input array to the nearest integer towards zero
floor(arr)	Returns floor of input array element wise
ceil(arr)	Returns ceiling of input array element wise
trunc(arr)	Return the truncated value of an input array element wise

Code:

```
import numpy as np
a = np.array([1.23,4.165,3.8245])
rounded_a = np.round_(a,2)
print(rounded_a)
```

Output:

```
[ 1.23  4.16  3.82]
```

5. Miscellaneous Functions

Function	Description
sqrt(arr)	Returns the square root of an input array element wise
cbrt(arr)	Returns cube root of an input array element wise
absolute(arr)	Returns absolute value each element in an input array
maximum(arr1,arr2,…)	Returns element wise maximum of the input arrays
minimum(arr1,arr2,…)	Returns element wise minimum of the input arrays
interp(arr, xp, fp)	Calculates one-dimensional linear interpolation
convolve(arr, v)	Returns linear convolution of two one-dimensional sequences
clip(arr, arr_min, arr_max)	Limits the values in an input array

NumPy

Finding the Maxima:

Code:

```
import numpy as np
a = [1,2,3] b = [3,1,2] maximum_elementwise = np.maximum(a,b)
print("maxima are:",maximum_elementwise)
```

Output:

```
maxima are: [3 2 3]
```

NumPy Aggregate and Statistical Functions

Functions	**Description**
np.mean()	Compute the arithmetic mean along the specified axis.
np.std()	Compute the standard deviation along the specified axis.
np.var()	Compute the variance along the specified axis.
np.sum()	Sum of array elements over a given axis.
np.prod()	Return the product of array elements over a given axis.
np.cumsum()	Return the cumulative sum of the elements along a given axis.
np.cumprod()	Return the cumulative product of elements along a given axis.
np.min(), np.max()	Return the minimum / maximum of an array or minimum along an axis.
np.argmin(), np.argmax()	Returns the indices of the minimum / maximum values along an axis
np.all()	Test whether all array elements along a given axis evaluate to True.
np.any()	Test whether any array element along a given axis evaluates to True.

```
import numpy as np

arr = np.array([[10, 20, 30], [40, 50, 60]])
print("Mean: ", np.mean(arr))
print("Std: ", np.std(arr))
print("Var: ", np.var(arr))
print("Sum: ", np.sum(arr))
print("Prod: ", np.prod(arr))
```

Sample output of above program.

```
Mean: 35.0
Std: 17.07825127659933
Var: 291.6666666666667
Sum: 210
Prod: 720000000
```

NumPy Example of Where function

- The where() function is used to chooses values from arrays depending on the value of a specific condition.

```
import numpy as np

before = np.array([[1, 2, 3], [4, 5, 6]])

# If element is less than 4, mul by 2 else by 3

after = np.where(before < 4, before * 2, before * 3)

print(after)
```

- **Sample output of above program.**

```
[[ 2 4 6]
[12 15 18]]
```

NumPy Example of Select function

- The select() function return an array drawn from elements in choice list, depending on conditions.

```
import numpy as np
before = np.array([[1, 2, 3], [4, 5, 6]])
# If element is less than 4, mul by 2 else by 3
after = np.select([before < 4, before], [before * 2, before * 3])
print(after)
```

Sample output of above program.

```
[[ 2  4  6]
 [12 15 18]]
```

Exploratory Data Analysis(EDA)

- ➢ Exploratory Data Analysis (EDA) is the first step in data analysis process .
- ➢ Exploratory Data Analysis (EDA) is developed by "**John Tukey**" in the 1970s.

What is Exploratory Data Analysis ?

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually.

- ➢ This step is very important especially when we arrive at modeling the data in order to apply Machine learning.
- ➢ Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more.
- ➢It often takes much time to explore the data.

Exploratory Data Analysis(EDA)

Exploratory Data Analysis helps us to –

1. *To give insight into a data set.*
2. *Understand the underlying structure.*
3. *Extract important parameters and relationships that hold between them.*
4. *Test underlying assumptions*

How to perform Exploratory Data Analysis ?

- There is no one method or common methods in order to perform EDA.
- EDA is performed depends on the dataset that we are working.

Exploratory Data Analysis(EDA)

Example:

Consider a data set related to car.

- This dataset contains more of 10, 000 rows and more than 10 columns which contains features of the car such as Engine Fuel Type, Engine HP, Transmission Type, highway MPG, city MPG and many more.

explore the data and make it ready for modeling.

Exploratory Data Analysis(EDA)

data exploring:

1. Importing the required libraries for EDA

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
```

Exploratory Data Analysis(EDA)

data exploring:

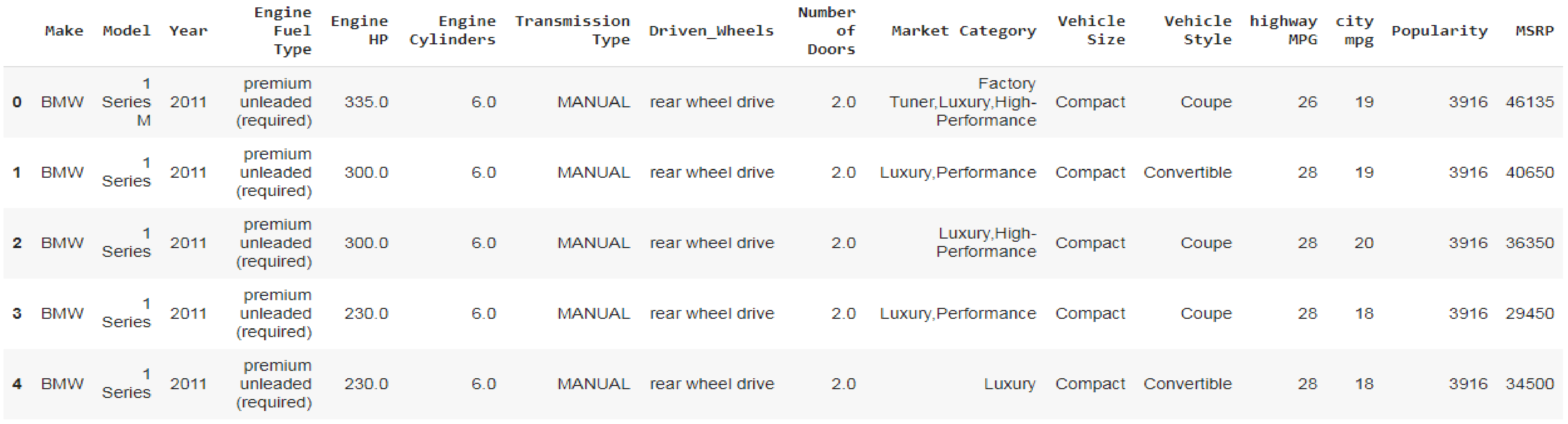
2. Loading the data into the data frame.

- Loading the data into the pandas data frame is certainly one of the most important steps in EDA.
- The value from the data set is comma-separated. So read the CSV into a data frame
- pandas data frame is used to read the data.

Exploratory Data Analysis(EDA)

<u>data exploring:</u>

- df = pd.read_csv("data.csv")
- df.head(5) # To display the top 5 rows

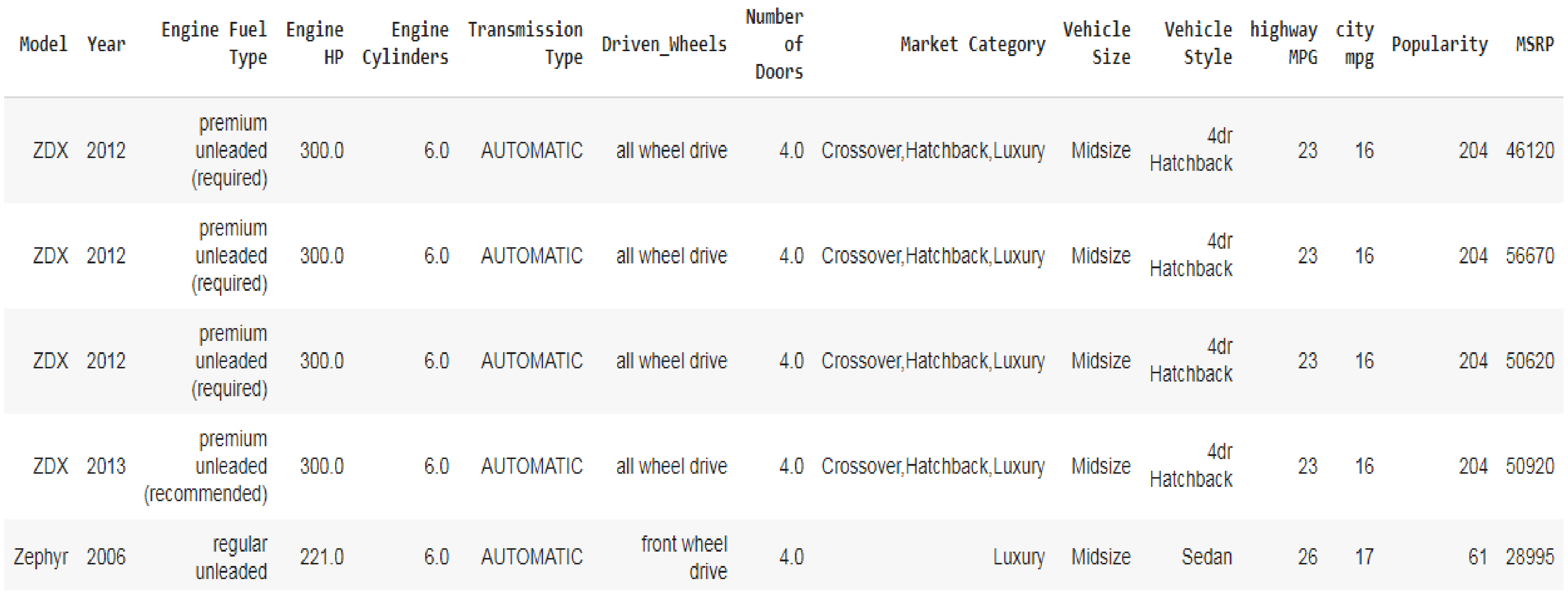


	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916	34500

Exploratory Data Analysis(EDA)

data exploring:

- df.tail(5) # To display the botton 5 rows



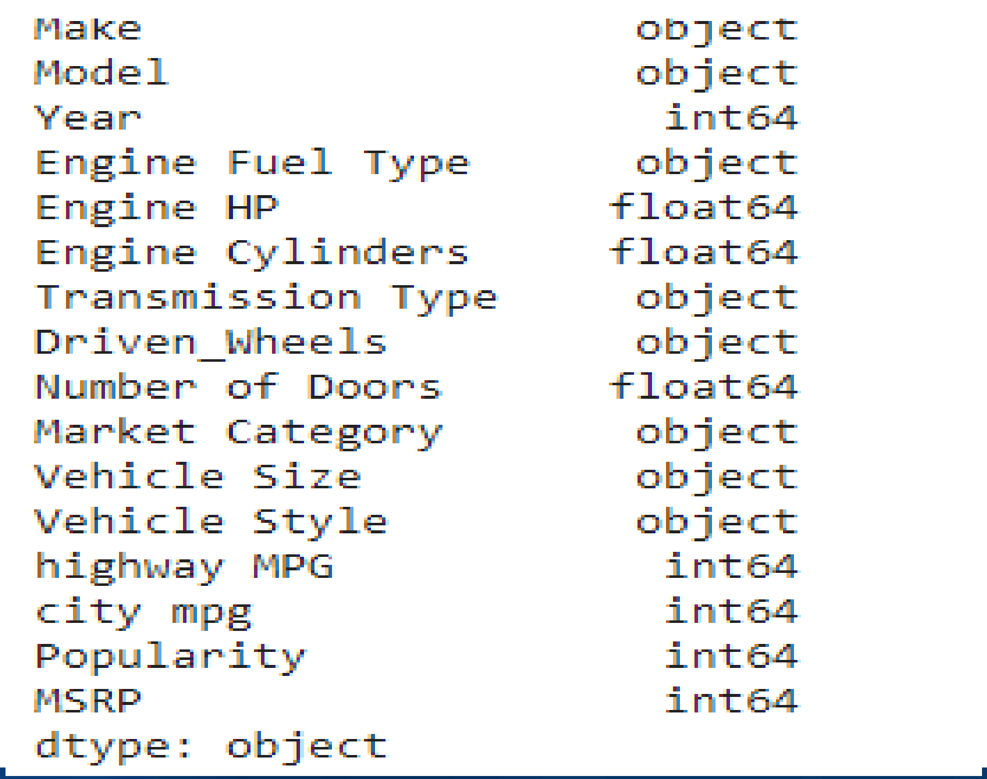
Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	46120
ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	56670
ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50620
ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50920
Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	front wheel drive	4.0	Luxury	Midsize	Sedan	26	17	61	28995

Exploratory Data Analysis(EDA)

data exploring:

3. Checking the types of data

- df.dtypes



```
Make                  object
Model                 object
Year                   int64
Engine Fuel Type      object
Engine HP            float64
Engine Cylinders     float64
Transmission Type     object
Driven_Wheels         object
Number of Doors      float64
Market Category       object
Vehicle Size          object
Vehicle Style         object
highway MPG            int64
city mpg               int64
Popularity             int64
MSRP                   int64
dtype: object
```

- Here we check for the datatypes because sometimes the MSRP or the price of the car would be stored as a string, if in that case, we have to convert that string to the integer data only then we can plot the data via a graph.
- Here, in this case, the data is already in integer format so nothing to worry.

Exploratory Data Analysis(EDA)

data exploring:

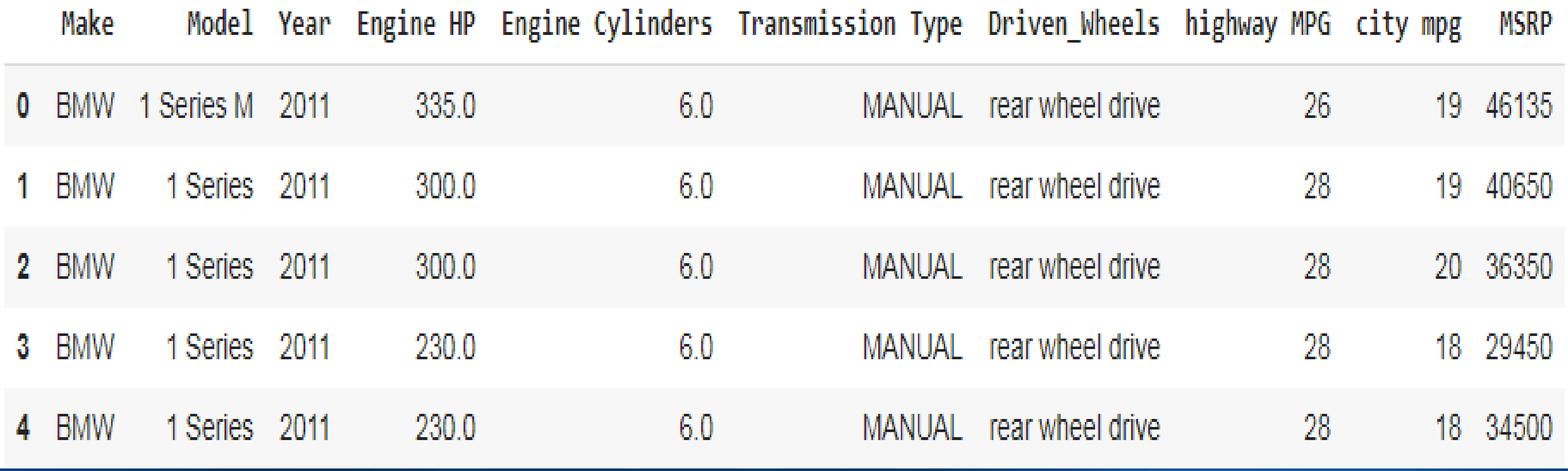
4. Dropping irrelevant columns.

- This step is certainly needed in every EDA because sometimes there would be many columns that we never use. In such cases drop the irrelevant columns.
- In this case, the columns such as Engine Fuel Type, Market Category, Vehicle style, Popularity, Number of doors, Vehicle Size doesn't make any sense so just drop

Exploratory Data Analysis(EDA)

data exploring:

- df = df.drop(['Engine Fuel Type', 'Market Category', 'Vehicle Style', 'Popularity', 'Number of Doors', 'Vehicle Size'], axis=1)
- df.head(5)



	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city mpg	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

Exploratory Data Analysis(EDA)

<u>data exploring:</u>

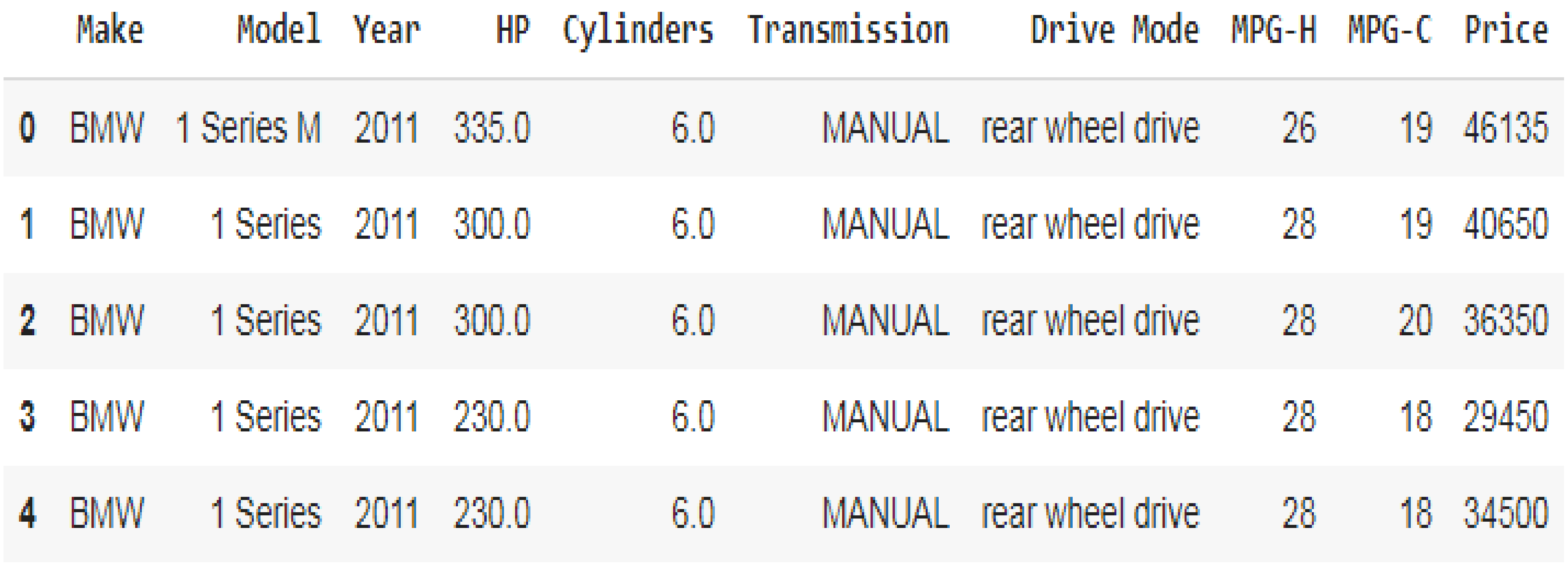
5. Renaming the columns

- In this instance, most of the column names are very confusing to read, so rename their column names.
- This is a good approach it improves the readability of the data set.
- df = df.rename(columns={"Engine HP": "HP", "Engine Cylinders": "Cylinders", "Transmission Type": "Transmission", "Driven_Wheels": "Drive Mode","highway MPG": "MPG-H", "city mpg": "MPG-C", "MSRP": "Price" })

Exploratory Data Analysis(EDA)

data exploring:

df.head(5)



	Make	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

Exploratory Data Analysis(EDA)

data exploring:

6. Dropping the duplicate rows

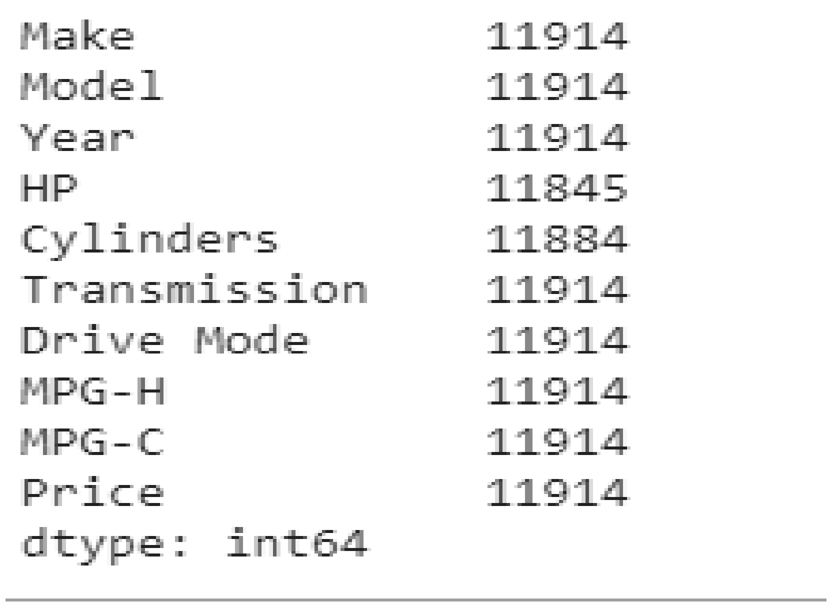
Procedure:

- **First finding the no of rows & columns.**

  df.shape

(11914, 10)

- df.count()   # Used to count the number of rows



```
Make            11914
Model           11914
Year            11914
HP              11845
Cylinders       11884
Transmission    11914
Drive Mode      11914
MPG-H           11914
MPG-C           11914
Price           11914
dtype: int64
```

- there are 11914 rows

Exploratory Data Analysis(EDA)

<u>data exploring:</u>

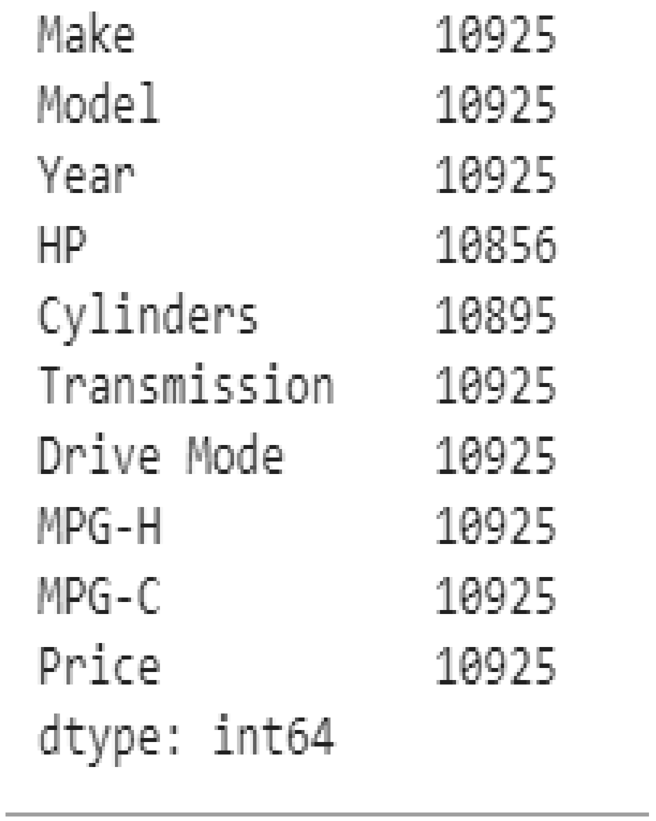
➢ **Finding no of duplicate data.**

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ",
                duplicate_rows_df.shape)
```

number of duplicate rows: (989, 10)

➢ **removing 989 rows of duplicate data**

```
df = df.drop_duplicates()
df.count() # Used to count the number of rows
```



```
Make            10925
Model           10925
Year            10925
HP              10856
Cylinders       10895
Transmission    10925
Drive Mode      10925
MPG-H           10925
MPG-C           10925
Price           10925
dtype: int64
```

Exploratory Data Analysis(EDA)

data exploring:

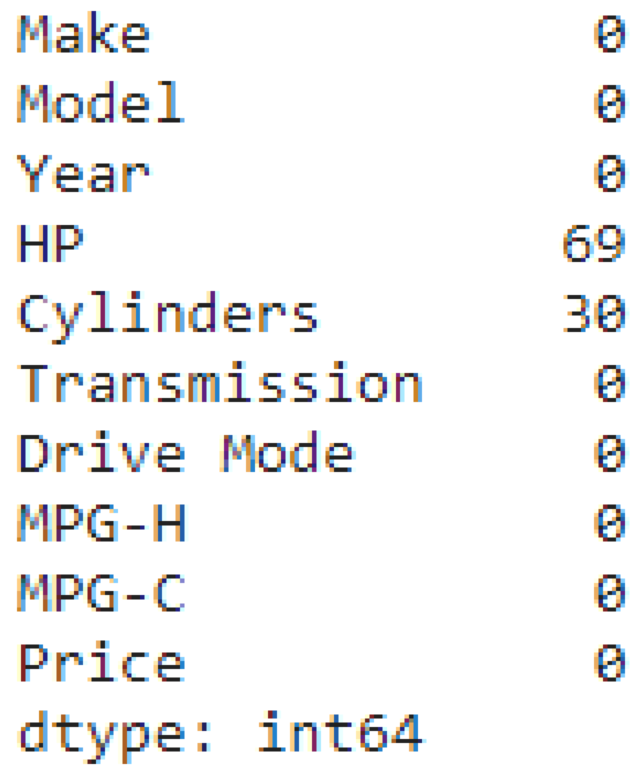
7. Dropping the missing or null values.

➢replace the missing values with the mean or the average of that column.

print(df.isnull().sum())

➢ but in this case drop the missing values. This is because there is nearly 100 missing value compared to 10, 000 values this is a small number and this is negligible. so just dropped those values.

➢This is the reason in the above step while counting both Cylinders and Horsepower (HP) had 10856 and 10895 over 10925 rows



```
Make             0
Model            0
Year             0
HP              69
Cylinders       30
Transmission     0
Drive Mode       0
MPG-H            0
MPG-C            0
Price            0
dtype: int64
```

Exploratory Data Analysis(EDA)

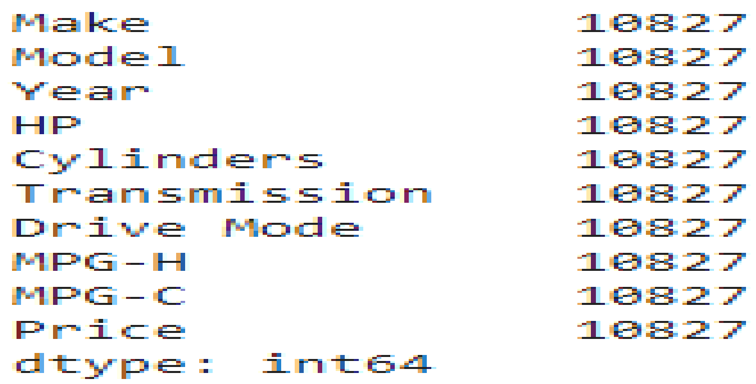
<u>data exploring:</u>

df = df.dropna()   # Dropping the missing values.

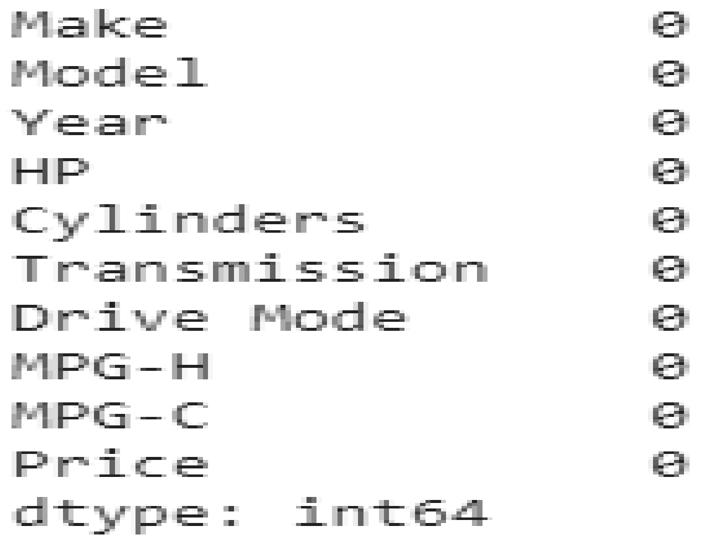
df.count()

➢Now we have removed all the rows which contain the Null or N/A values (Cylinders and Horsepower (HP)).

➢print(df.isnull().sum()) # After dropping the values



```
Make            10827
Model           10827
Year            10827
HP              10827
Cylinders       10827
Transmission    10827
Drive Mode      10827
MPG-H           10827
MPG-C           10827
Price           10827
dtype: int64
```



```
Make            0
Model           0
Year            0
HP              0
Cylinders       0
Transmission    0
Drive Mode      0
MPG-H           0
MPG-C           0
Price           0
dtype: int64
```

Exploratory Data Analysis(EDA)

<u>data exploring:</u>

8. Detecting Outliers

- An outlier is a point or set of points that are different from other points.
- Sometimes they can be very high or very low.
- It's often a good idea to detect and remove the outliers. Because outliers are one of the primary reasons for resulting in a less accurate model. Hence it's a good idea to remove them.
- IQR (Inter-Quartile Range) score technique is used to detect and remove outlier.

Exploratory Data Analysis(EDA)

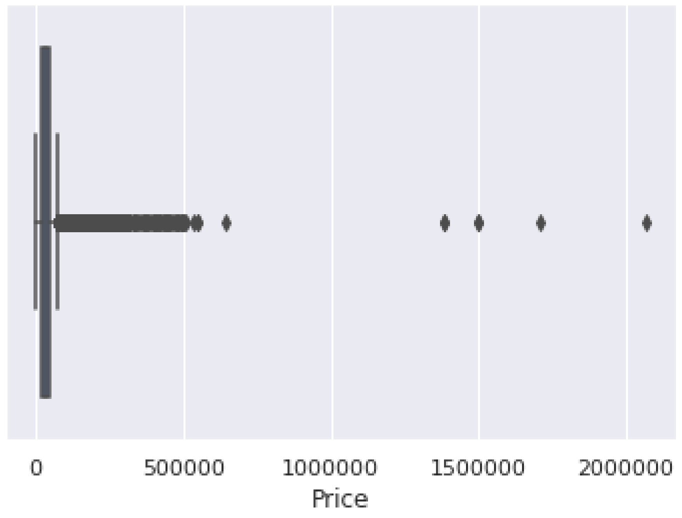
data exploring:

outliers can be seen with visualizations using a box plot.

```
sns.boxplot(x=df['Price'])
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0d36a38be0>
```

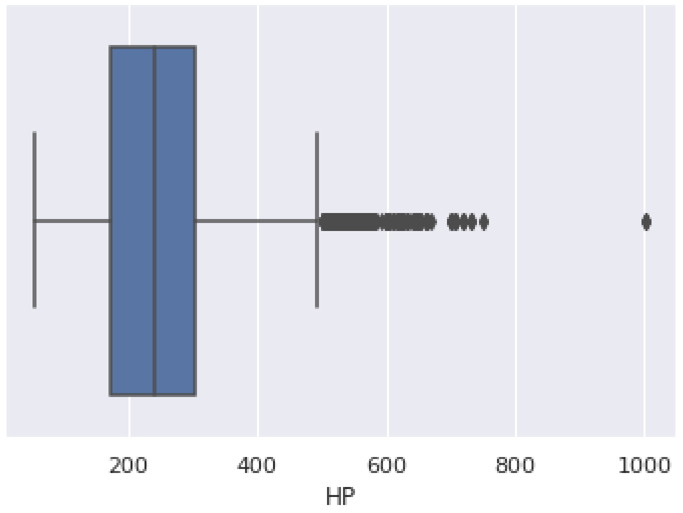


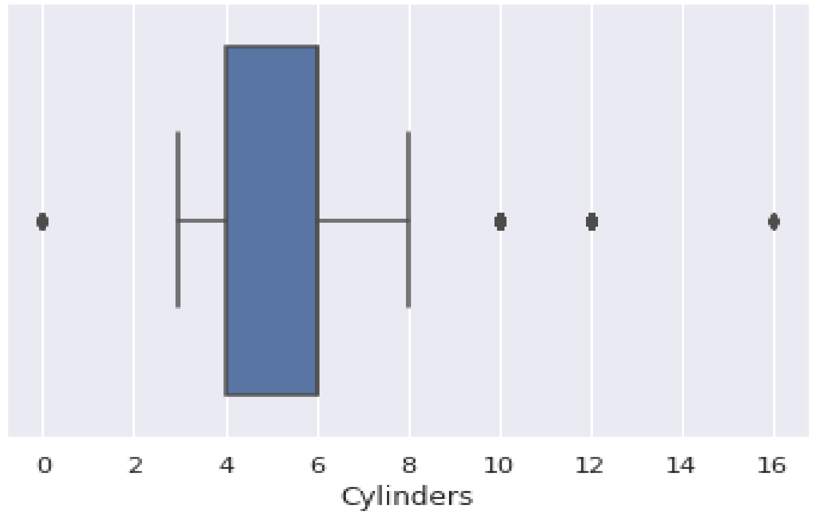
Exploratory Data Analysis(EDA)

data exploring:

```
sns.boxplot(x=df['HP'])
sns.boxplot(x=df['Cylinders'])
```

- These are the box plot of MSRP, Cylinders, Horsepower and EngineSize.
- Herein all the plots, we can find some points are outside the box they are none other than outliers.

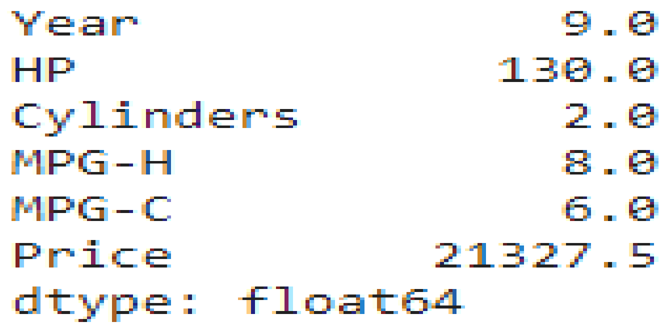




Exploratory Data Analysis(EDA)

data exploring:

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```



```
Year              9.0
HP              130.0
Cylinders         2.0
MPG-H             8.0
MPG-C             6.0
Price         21327.5
dtype: float64
```

```
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
(9191, 10)
```

As seen above there were around 1600 rows were outliers. But you cannot completely remove the outliers because even after we use the above technique there maybe 1–2 outlier unremoved but that ok because there were more than 100 outliers.

Exploratory Data Analysis(EDA)

data exploring:

9. Plot different features against one another (scatter), against frequency (histogram)

Histogram

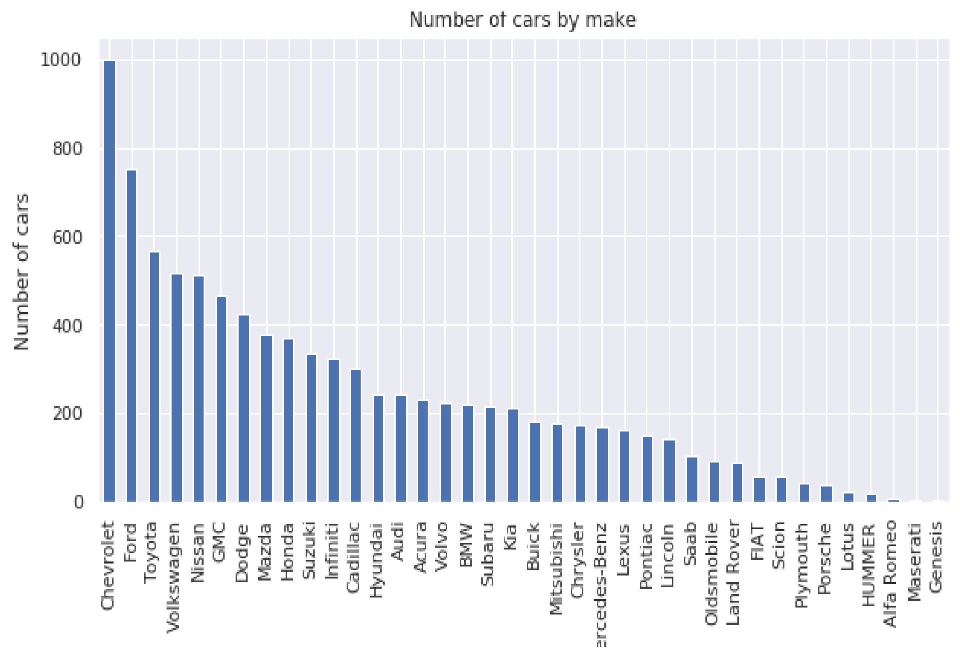
- Histogram refers to the frequency of occurrence of variables in an interval.
- In this case, there are mainly 10 different types of car manufacturing companies, but it is often important to know who has the most number of cars.

Exploratory Data Analysis(EDA)

data exploring:

- histogram is used to know the total number of car manufactured by a different company.

```
df.Make.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make');
```



Exploratory Data Analysis(EDA)

data exploring:

Heat Maps

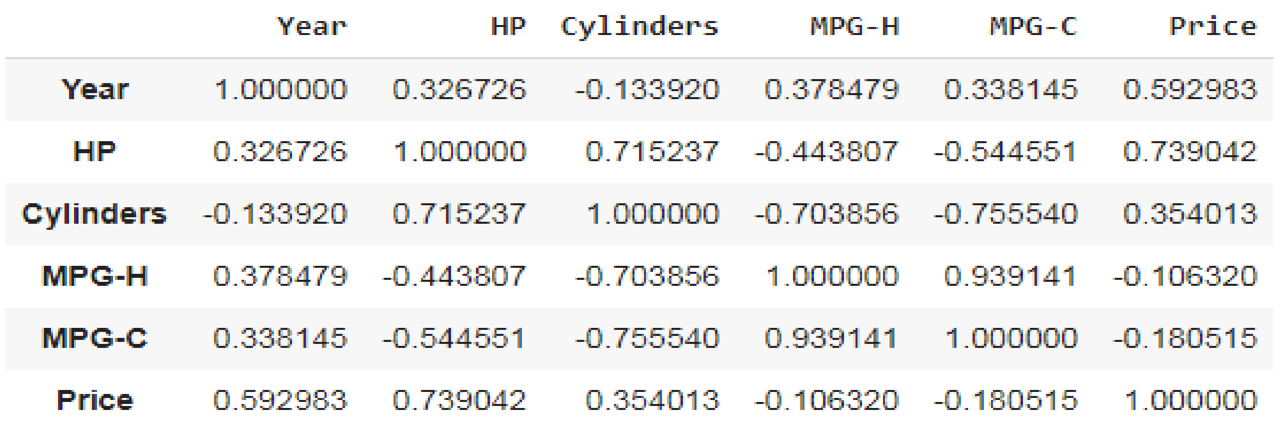
- Heat Maps is a type of plot which is necessary when we need to find the dependent variables.
- One of the best way to find the relationship between the features can be done using heat maps.

Exploratory Data Analysis(EDA)

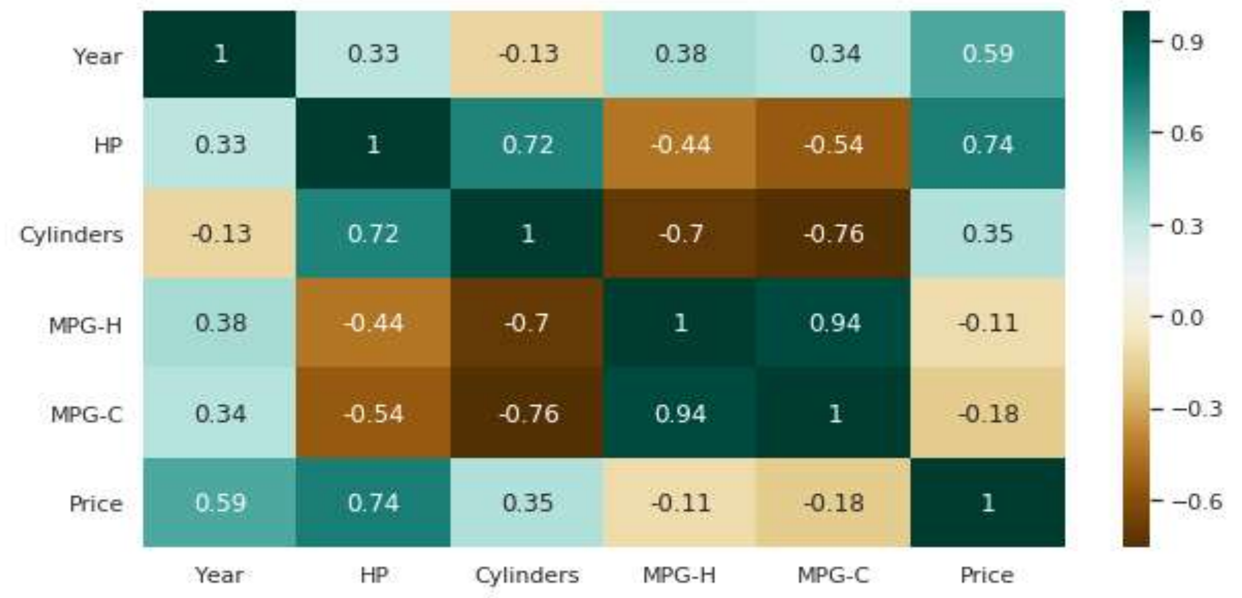
data exploring:

Heat Maps

```
   plt.figure(figsize=(10,5))
c= df.corr()
sns.heatmap(c,cmap="BrBG",
  annot=True)
c
```



	Year	HP	Cylinders	MPG-H	MPG-C	Price
Year	1.000000	0.326726	-0.133920	0.378479	0.338145	0.592983
HP	0.326726	1.000000	0.715237	-0.443807	-0.544551	0.739042
Cylinders	-0.133920	0.715237	1.000000	-0.703856	-0.755540	0.354013
MPG-H	0.378479	-0.443807	-0.703856	1.000000	0.939141	-0.106320
MPG-C	0.338145	-0.544551	-0.755540	0.939141	1.000000	-0.180515
Price	0.592983	0.739042	0.354013	-0.106320	-0.180515	1.000000



- heat map describes the price feature depends mainly on the Engine Size, Horsepower, and Cylinders.

Exploratory Data Analysis(EDA)

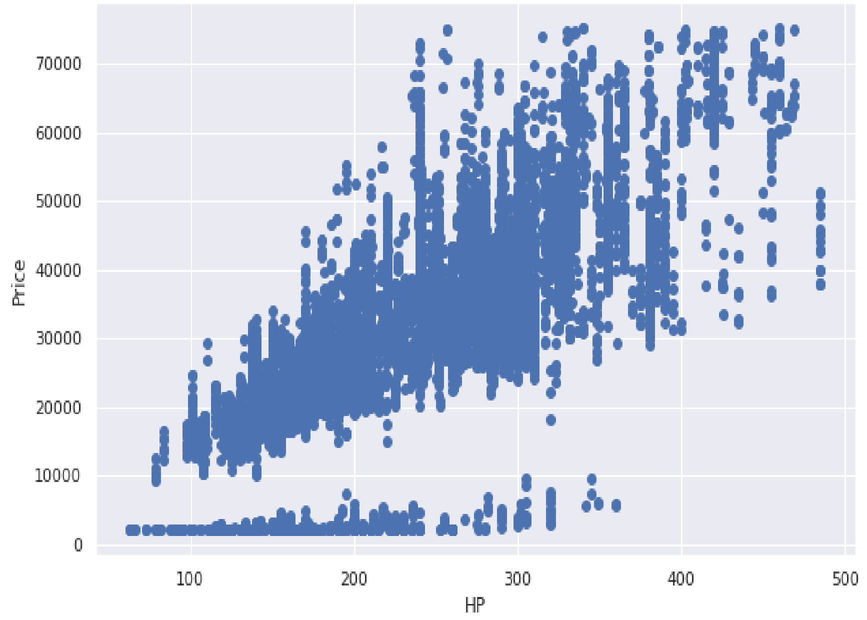
data exploring:

Scatterplot

➢ scatter plots are used to find the correlation between two variables.

➢Here the scatter plots are plotted between Horsepower and Price.

```
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['HP'], df['Price'])
ax.set_xlabel('HP')
ax.set_ylabel('Price')
plt.show()
```



With the plot given we can easily draw a trend line.

Exploratory Data Analysis(EDA)

data exploring:

- **Hence the above are some of the steps involved in Exploratory data analysis, these are some general steps that must follow in order to perform EDA.**

Descriptive Statistics

➢ Descriptive Statistics is the default process in Data analysis.

➢ Exploratory Data Analysis (EDA) is not complete without a Descriptive Statistic analysis.

➢Descriptive Statistics is divided into two parts:

1. **Measure of Central Data points and**
2. **Measure of Dispersion.**

1. Measure of Central Data points:

The following operations are performed under Measure of Central Data points

1. **Count**
2. **Mean**
3. **Mode**
4. **Median**

Descriptive Statistics

2. Measure of Dispersion

The following operations are performed under Measure of Dispersion

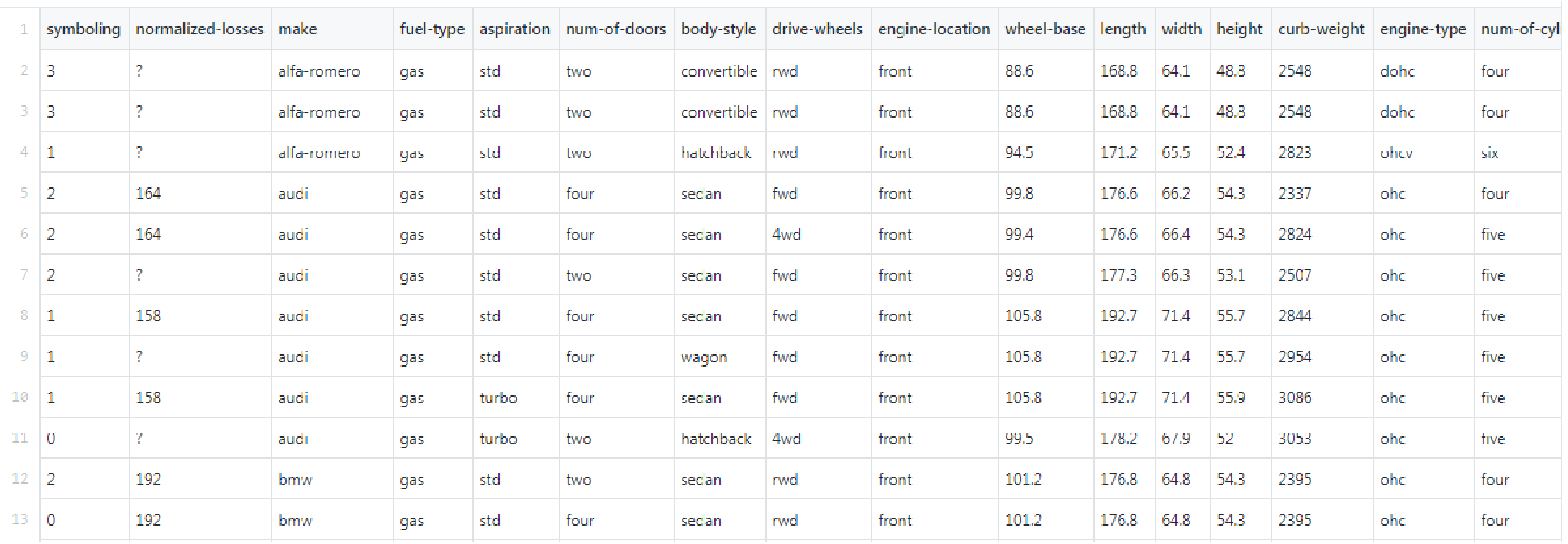
1. **Range**
2. **Percentiles (or) Quartiles**
3. **Standard deviation**
4. **Variance**
5. **Skewness**

Descriptive Statistics

Example:

➢Consider a file:

Data.csv



symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	engine-type	num-of-cyl
3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four
3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four
1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six
2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four
2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five
2	?	audi	gas	std	two	sedan	fwd	front	99.8	177.3	66.3	53.1	2507	ohc	five
1	158	audi	gas	std	four	sedan	fwd	front	105.8	192.7	71.4	55.7	2844	ohc	five
1	?	audi	gas	std	four	wagon	fwd	front	105.8	192.7	71.4	55.7	2954	ohc	five
1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	192.7	71.4	55.9	3086	ohc	five
0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	178.2	67.9	52	3053	ohc	five
2	192	bmw	gas	std	two	sedan	rwd	front	101.2	176.8	64.8	54.3	2395	ohc	four
0	192	bmw	gas	std	four	sedan	rwd	front	101.2	176.8	64.8	54.3	2395	ohc	four

➢It contains 206 lines and 26 columns

➢Before starting descriptive statistics analysis complete the **data collection and cleaning process**.

Descriptive Statistics

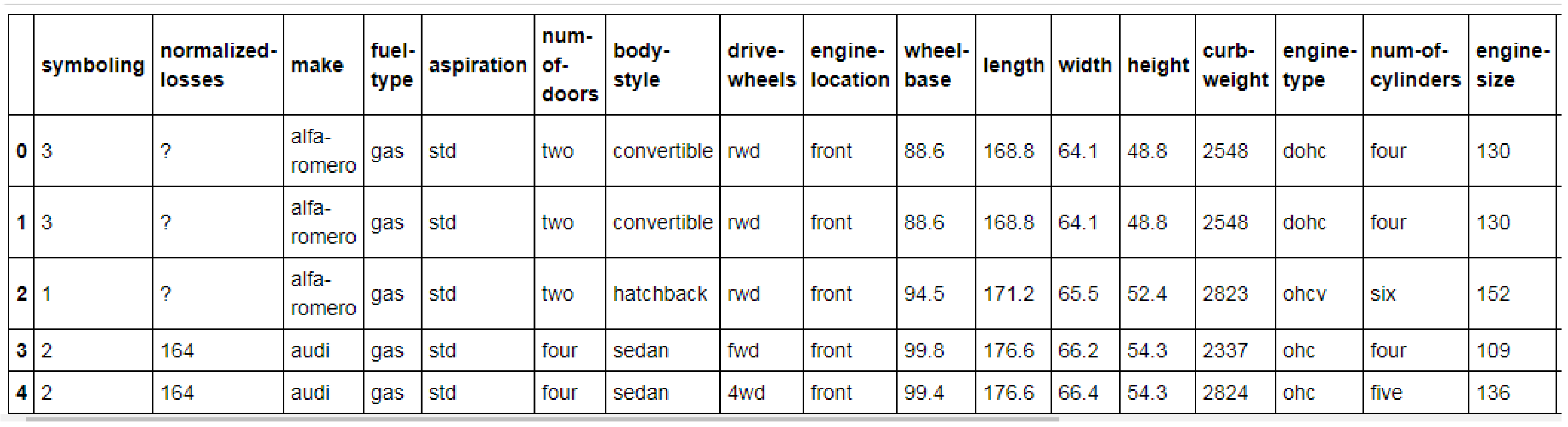
Data Collection:

```
# loading data set as Pandas dataframe
import pandas as pd
df = pd.DataFrame()
df = pd.read_csv("https://raw.githubusercontent.com/PacktPublishing/hands-on-exploratory-data-analysis-with-python/master/Chapter%205/data.csv")
```

- df.head()

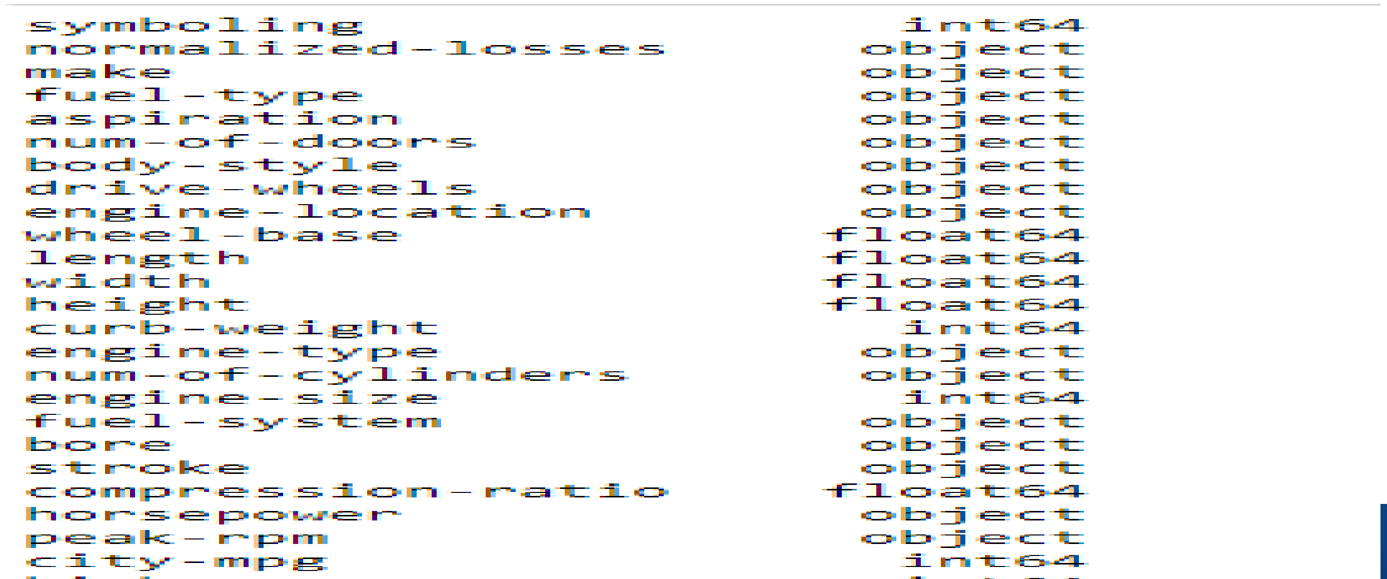
Descriptive Statistics

Data Collection:



	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	engine-type	num-of-cylinders	engine-size
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136

- df.dtypes



```
symboling                int64
normalized-losses       object
make                    object
fuel-type               object
aspiration              object
num-of-doors            object
body-style              object
drive-wheels            object
engine-location         object
wheel-base             float64
length                 float64
width                  float64
height                 float64
curb-weight              int64
engine-type             object
num-of-cylinders        object
engine-size              int64
fuel-system             object
bore                    object
stroke                  object
compression-ratio      float64
horsepower              object
peak-rpm                object
city-mpg                 int64
```

Descriptive Statistics

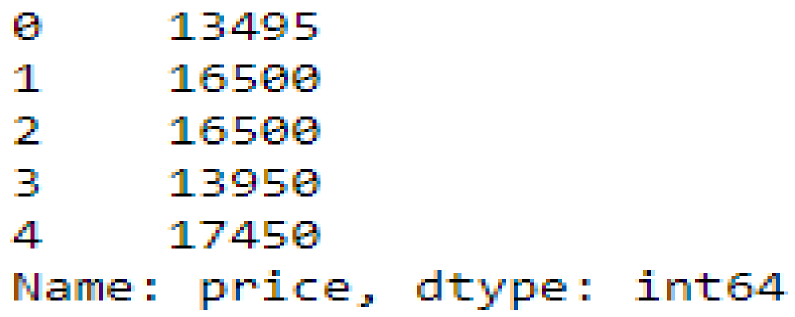
Data Cleaning

```
# Find out the number of values which are not numeric
df['price'].str.isnumeric().value_counts()
# List out the values which are not numeric
df['price'].loc[df['price'].str.isnumeric() == False]
#Setting the missing value to mean of price and convert the datatype to integer
price = df['price'].loc[df['price'] != '?']
pmean = price.astype(str).astype(int).mean()
df['price'] = df['price'].replace('?',pmean).astype(int)
```

Descriptive Statistics

Data Cleaning

➢df['price'].head()



```
0      13495
1      16500
2      16500
3      13950
4      17450
Name: price, dtype: int64
```

# Cleaning the horsepower losses field

df['horsepower'].str.isnumeric().value_counts()

horsepower = df['horsepower'].loc[df['horsepower'] != '?']

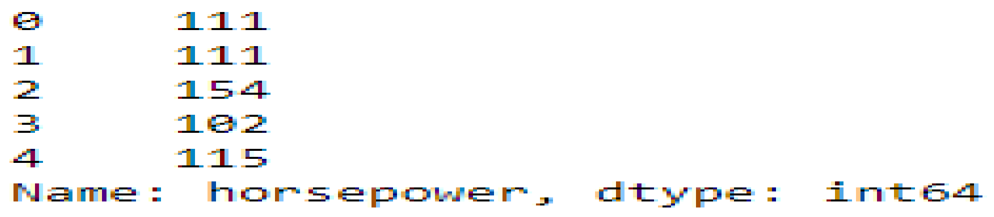
hpmean = horsepower.astype(str).astype(int).mean()

df['horsepower'] = df['horsepower'].replace('?',hpmean).astype(int)

Descriptive Statistics

Data Cleaning

- df['horsepower'].head()



```
0       111
1       111
2       154
3       102
4       115
Name: horsepower, dtype: int64
```

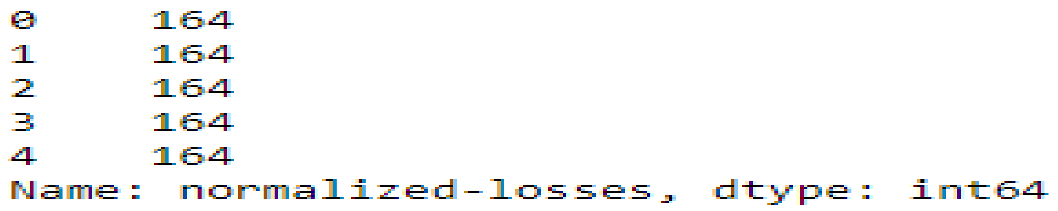
# Cleaning the Normalized losses field

```
df[df['normalized-losses']=='?'].count()
nl=df['normalized-losses'].loc[df['normalized-losses'] !='?'].count()
nmean=nl.astype(str).astype(int).mean()
df['normalized-losses'] = df['normalized-losses'].replace('?',nmean).astype(int)
```

Descriptive Statistics

Data Cleaning

➢df['normalized-losses'].head()



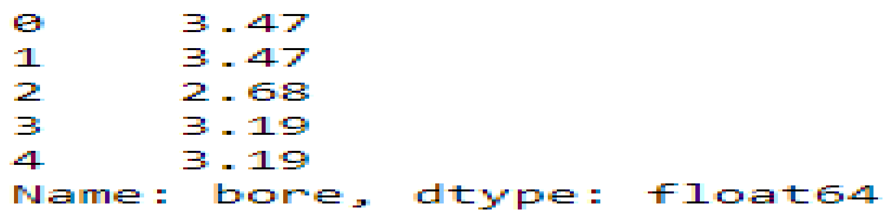
```
0      164
1      164
2      164
3      164
4      164
Name: normalized-losses, dtype: int64
```

# cleaning the bore
# Find out the number of invalid value
df['bore'].loc[df['bore'] == '?']
# Replace the non-numeric value to null and convert the datatype
df['bore'] = pd.to_numeric(df['bore'],errors='coerce')

Descriptive Statistics

Data Cleaning

➢df.bore.head()

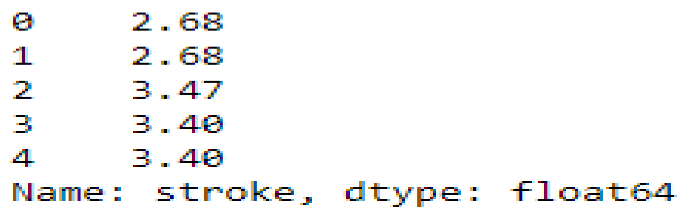


```
0      3.47
1      3.47
2      2.68
3      3.19
4      3.19
Name: bore, dtype: float64
```

# Cleaning the column stoke

df['stroke'] = pd.to_numeric(df['stroke'],errors='coerce')

df['stroke'].head()

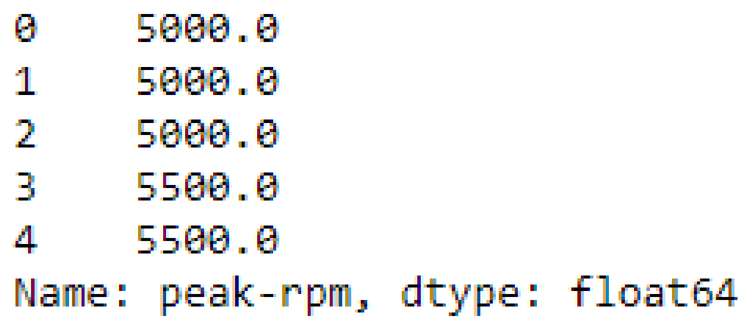


```
0     2.68
1     2.68
2     3.47
3     3.40
4     3.40
Name: stroke, dtype: float64
```

Descriptive Statistics

Data Cleaning

```
# Cleaning the column peak-rpm
df['peak-rpm'] = pd.to_numeric(df['peak-rpm'],errors='coerce')
df['peak-rpm'].head()
```

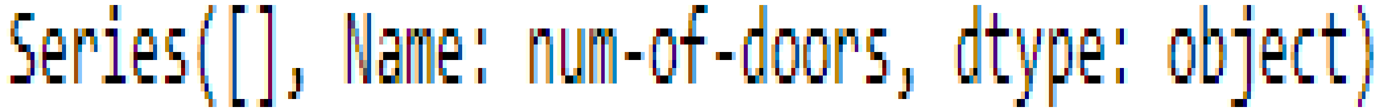


```
0    5000.0
1    5000.0
2    5000.0
3    5500.0
4    5500.0
Name: peak-rpm, dtype: float64
```

Descriptive Statistics

Data Cleaning

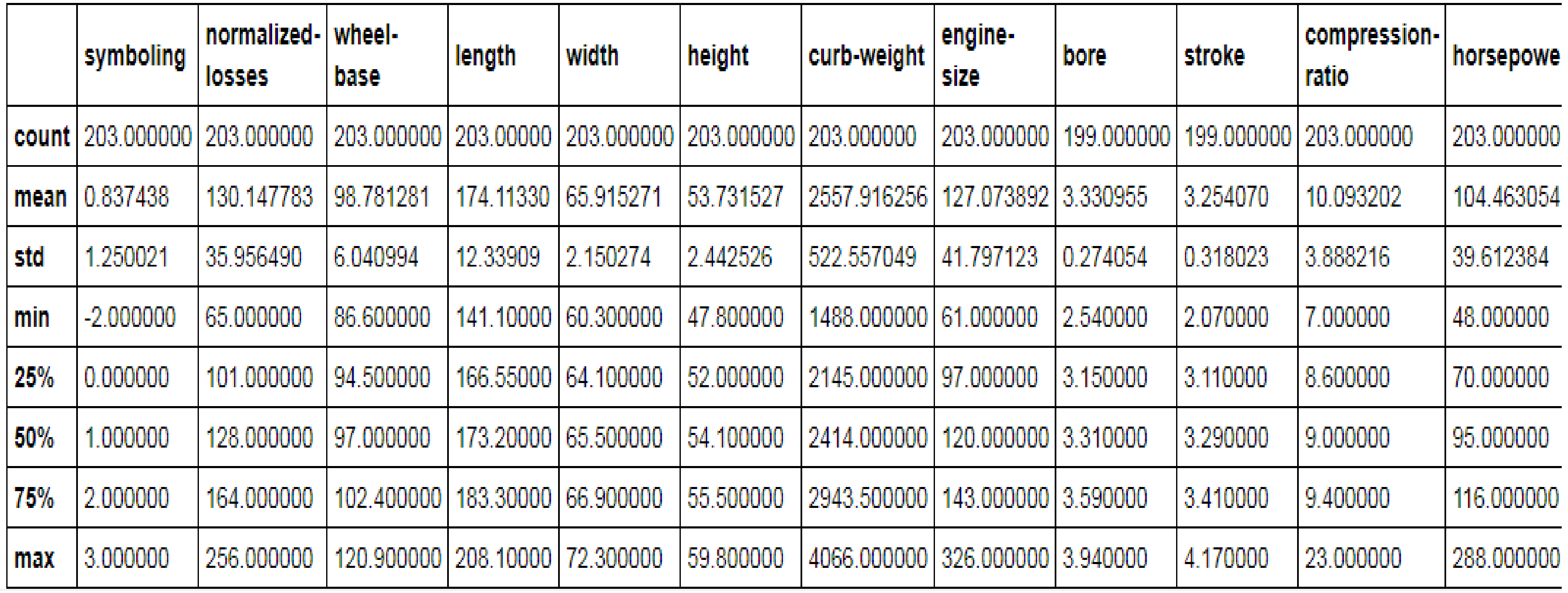
```
# Cleaning the Column num-of-doors data
# remove the records which are having the value '?'
df['num-of-doors'].loc[df['num-of-doors'] == '?']
df= df[df['num-of-doors'] != '?']
df['num-of-doors'].loc[df['num-of-doors'] == '?']
```



```
Series([], Name: num-of-doors, dtype: object)
```

Descriptive Statistics

- df.describe()



	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepowe
count	203.000000	203.000000	203.000000	203.00000	203.000000	203.000000	203.000000	203.000000	199.000000	199.000000	203.000000	203.000000
mean	0.837438	130.147783	98.781281	174.11330	65.915271	53.731527	2557.916256	127.073892	3.330955	3.254070	10.093202	104.463054
std	1.250021	35.956490	6.040994	12.33909	2.150274	2.442526	522.557049	41.797123	0.274054	0.318023	3.888216	39.612384
min	-2.000000	65.000000	86.600000	141.10000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000
25%	0.000000	101.000000	94.500000	166.55000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000
50%	1.000000	128.000000	97.000000	173.20000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000
75%	2.000000	164.000000	102.400000	183.30000	66.900000	55.500000	2943.500000	143.000000	3.590000	3.410000	9.400000	116.000000
max	3.000000	256.000000	120.900000	208.10000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000

Descriptive Statistics

computing Measure of central tendency

Count

It calculates the total count of numerical column data (or) each category of the categorical variables.

```
# get column height from df
height =df["height"]
print(height)
```

Output:

```
0      48.8
1      48.8
2      52.4
3      54.3
4      54.3
       ...
200    55.5
201    55.5
202    55.5
203    55.5
204    55.5
Name: height, Length: 203,
dtype: float64
```

Descriptive Statistics

computing Measure of central tendency

Mean

The Sum of values present in the column divided by total rows of that column is known as mean.

$$\bar{x} = \frac{\sum x}{N}$$

Here,
$\sum$ represents the summation
X represents observations
N represents the number of observations .

- It is also known as average.

Descriptive Statistics

computing Measure of central tendency

Median

- ➢The center value of an attribute is known as a median.
- ➢***The median value divides the data points into two parts. That means 50% of data points are present above the median and 50% below.***

If the total number of observations (n) is an odd number, then the formula is given below:

$$Median = \left(\frac{n+1}{2}\right)^{th} observation$$

If the total number of the observations (n) is an even number, then the formula is given below:

$$Median = \frac{\left(\frac{n}{2}\right)^{th} observation + \left(\frac{n}{2}+1\right)^{th} observation}{2}$$

Descriptive Statistics

computing Measure of central tendency

Mode

- The mode is that data point whose count is maximum in a column.
- There is only one mean and median value for each column. But, attributes can have more than one mode value

```
#Calculating mean, median, mode of dataset height
mean = height.mean()
median =height.median()
mode = height.mode()
print(mean , median, mode)
```

Output:

```
53.73152709359609      54.1 0         50.8  dtype: float64
```

Descriptive Statistics

computing Measure of central tendency

- For categorical variables which has discrite values.
- categorical data can be summarized by using the function value_counts().

```
#summarize categories of drive-wheels
drive_wheels_count =df["drive-wheels"].value_counts()
print(drive_wheels_count)
```

Output:

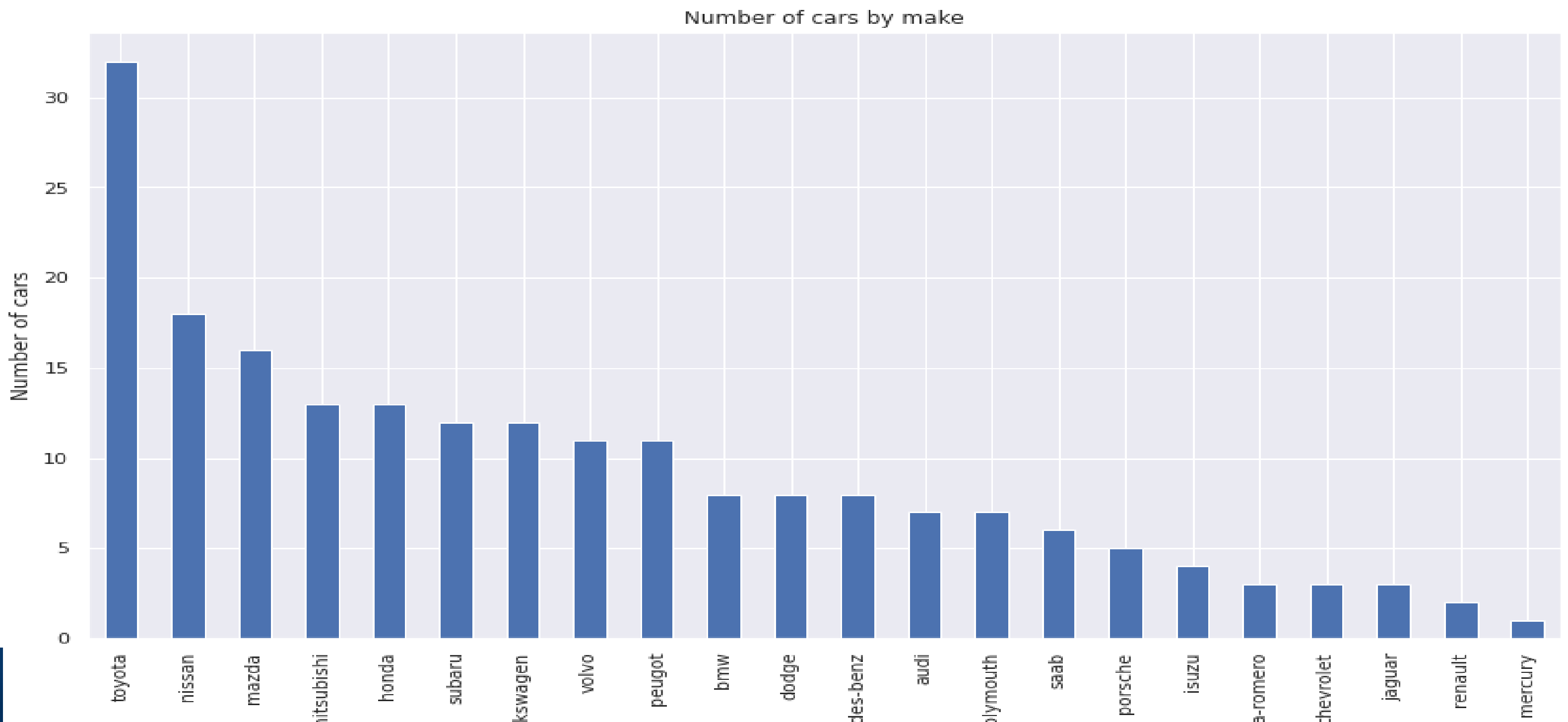
```
fwd   118     rwd     764     wd     9     Name: drive-wheels,  dtype: int64
```

Descriptive Statistics

computing Measure of central tendency

Example 2

```
import matplotlib.pyplot as plt
df.make.value_counts().nlargest(30).plot(kind='bar', figsize=(14,8))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make of the cars');
```



Descriptive Statistics

<u>Measures of Dispersion</u>

<u>Range</u>

- The difference between the max value to min value in a column is known as the range.

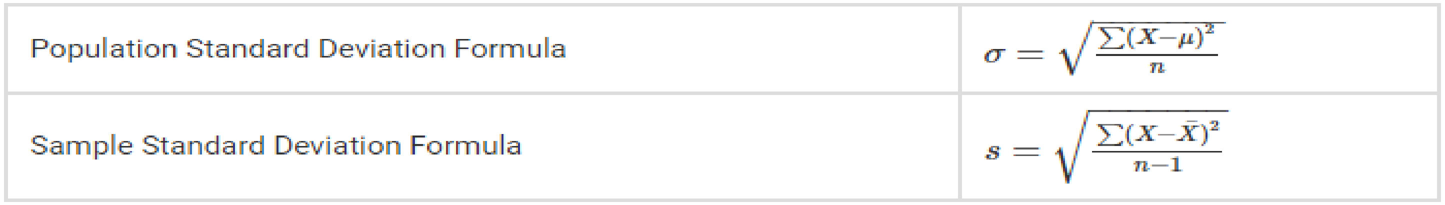
<u>Standard Deviation</u>

- The standard deviation value tells us how much all data points deviate from the mean value.
- The standard deviation is affected by the outliers because it uses the mean for its calculation.

Descriptive Statistics

Measures of Dispersion

- Formulas for Standard Deviation



Population Standard Deviation Formula	$\sigma = \sqrt{\frac{\sum(X-\mu)^2}{n}}$
Sample Standard Deviation Formula	$s = \sqrt{\frac{\sum(X-\bar{X})^2}{n-1}}$

- **Notations for Standard Deviation**

σ = Standard Deviation

x_i = Terms Given in the Data

$\bar{x}$ = Mean

n = Total number of Terms

Descriptive Statistics

Measures of Dispersion

Standard Deviation

➢Example for standard deviation

```
#standard deviation of data set using std()
function
std_dev =df.std()
print(std_dev)
# standard deviation of the specific column
sv_height=df.loc[:,"height"].std()
print(sv_height)
```

```
2.442525704031867
```

Output:

```
symboling              1.250021
normalized-losses     35.956490
wheel-base             6.040994
length                12.339090
width                  2.150274
height                 2.442526
curb-weight          522.557049
engine-size           41.797123
bore                   0.274054
stroke                 0.318023
compression-ratio      3.888216
horsepower            39.612384
peak-rpm             479.820136
city-mpg               6.529812
highway-mpg            6.874645
price               7898.957924
dtype: float64
```

Descriptive Statistics

Measures of Dispersion

Variance

- Variance is the square of standard deviation.

Variance = (Standard deviation)^2= σ^2

- In the case of outliers, the variance value becomes large and noticeable.

Descriptive Statistics

Measures of Dispersion

Variance

➢Example for standard variance

```
# variance of data set using var()
functionvariance=df.var()
print(variance)
#variance of the specific column
var_height=df.loc[:,"height"].var()
print(var_height)
5.965931814856368
```

Output:

```
symboling           1.562552e+00
normalized-losses    1.292869e+03
wheel-base           3.649361e+01l
ength               1.522531e+02
width               4.623677e+00
height              5.965932e+00
curb-weight          2.730659e+05
engine-size         1.746999e+03
bore                7.510565e-02
stroke              1.011384e-01
compression-ratio    1.511822e+01
horsepower           1.569141e+03
peak-rpm            2.302274e+05
city-mpg            4.263844e+01
highway-mpg          4.726074e+01
price               6.239354e+07
dtype: float64
```

Descriptive Statistics

Measures of Dispersion

Skewness

- Ideally, the distribution of data should be in the shape of Gaussian (bell curve).
- But practically, data shapes are skewed or have asymmetry. This is known as skewness in data.
- Formula for skewness is:

Skew = 3 * (Mean – Median) / Standard Deviation

- Skewness value can be negative (left) skew or positive (right) skew. Its value should be close to zero.

Descriptive Statistics

Measures of Dispersion

Skewness

- Example for skewness
- df.skew()

```
# skewness of the specific column
df.loc[:,"height"].skew()
0.06413448813322854
```

Output:

```
symboling 0.204275
normalized-losses 0.209007
wheel-base 1.041170
 length 0.154086
width 0.900685
height 0.064134
curb-weight 0.668942
engine-size 1.934993
bore 0.013419
 stroke -0.669515
compression-ratio 2.682640
horsepower 1.391224
peak-rpm 0.073094
city-mpg 0.673533
highway-mpg 0.549104
price 1.812335
 dtype: float64
```

Descriptive Statistics

Measures of Dispersion

Percentiles or Quartiles

- Column values can be spread by calculating the summary of several percentiles.
- Median is also known as the 50th percentile of data.
- Here is a different percentile.
  1. The minimum value equals to 0th percentile.
  2. The maximum value equals to 100th percentile.
  3. The first quartile equals to 25th percentile.
  4. The third quartile equals to 75th percentile.

Descriptive Statistics

Measures of Dispersion

Quartiles

- It divides the data set into four equal points.
- First quartile = 25th percentile
- Second quartile = 50th percentile (Median)
- Third quartile = 75th percentile
- Based on the quartile, there is a another measure called inter-quartile range that also measures the variability in the dataset. It is defined as:

IQR = Q3 - Q1

- IQR is not affected by the presence of outliers.

Descriptive Statistics

<u>Measures of Dispersion</u>

<u>Quartiles</u>

Example 1:

```
price = df.price.sort_values()
Q1 = np.percentile(price, 25)
Q2 = np.percentile(price, 50)
Q3 = np.percentile(price, 75)

IQR = Q3 - Q1
IQR
8718.5
```

Descriptive Statistics

Measures of Dispersion

Quartiles

Example 2:

```
#calculating 30th percentile of heights in dataset
height = df["height"]
percentile = np.percentile(height, 50,)
print(percentile)

54.1
```

summary statistics of EDA

- Exploratory data analysis or "EDA" is a critical first step in analyzing the data from an experiment.
- The uses of EDA are:

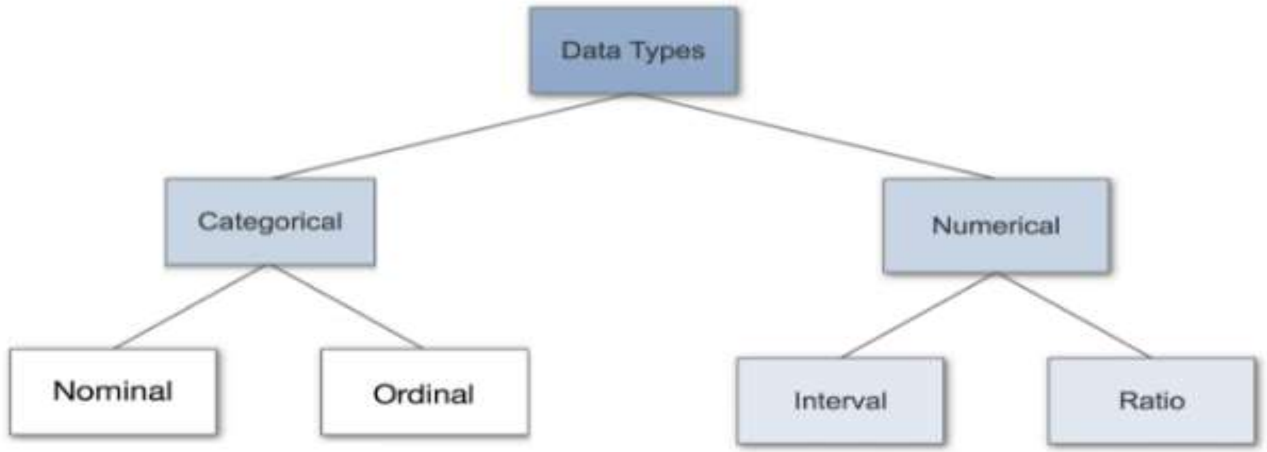
1. detection of mistakes
2. checking of assumptions
3. preliminary selection of appropriate models
4. determining relationships among the explanatory variables, and
5. assessing the direction and rough size of relationships between explanatory and outcome variables.

summary statistics of EDA

Typical data format (Data Types) and the types of EDA

Data Types:

Data types are mainly classified in to 2 types. Those are



summary statistics of EDA

Categorical Data

- Categorical data represents characteristics.
- It can represent things like a person’s gender, language etc.
- Categorical data can also take on numerical values.

Example: 1 for female and 0 for male.

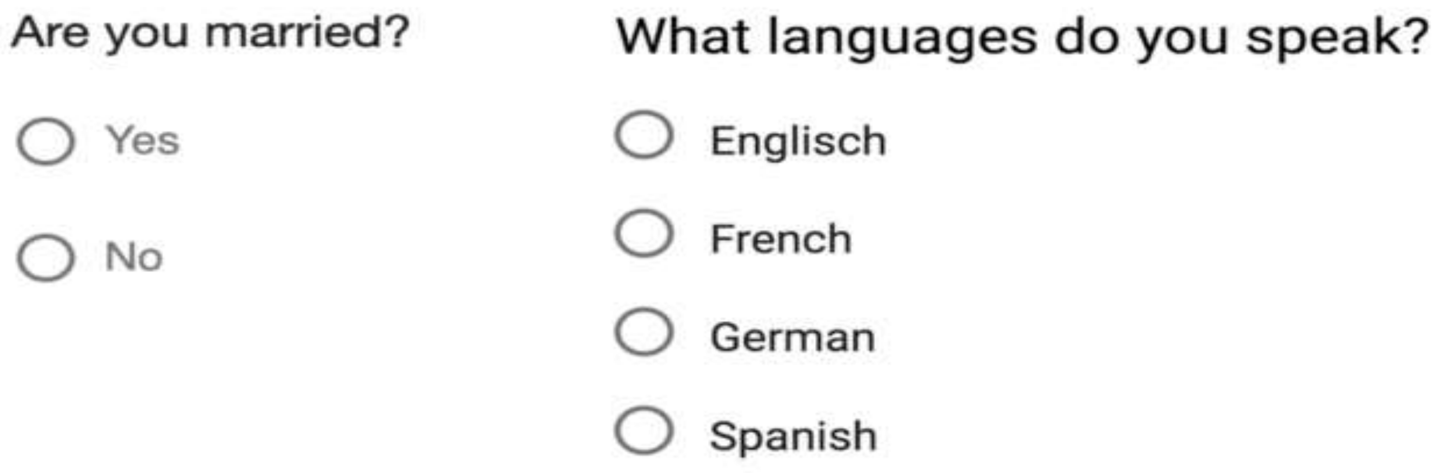
- These numbers don’t have mathematical meaning.

summary statistics of EDA

Nominal Data

- ➢Nominal values represent discrete units and are used to label variables, that have no quantitative value.
- ➢ nominal data has no order.
- ➢If the order of the values changed their meaning would not change.

Examples:



Are you married?

○ Yes

○ No

What languages do you speak?

○ Englisch

○ French

○ German

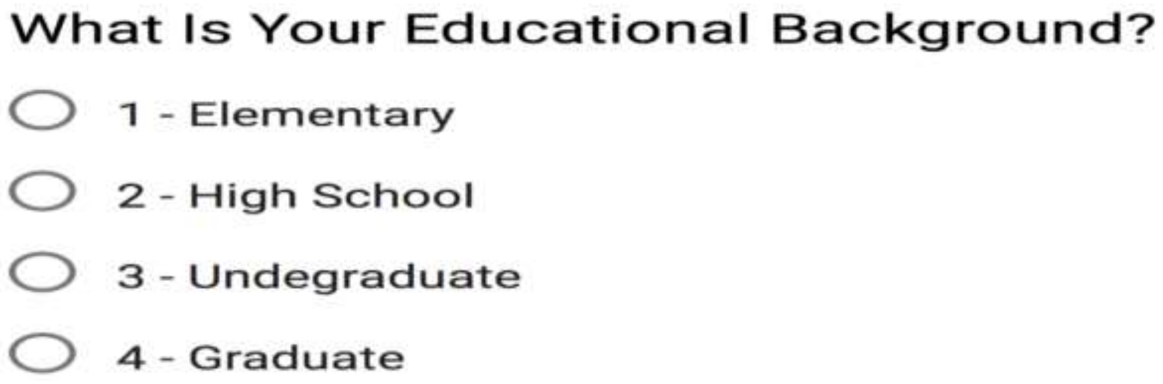
○ Spanish

summary statistics of EDA

Ordinal Data

- Ordinal values represent discrete and ordered units.
- It is same as nominal data, except that it’s ordering matters.

Example :



What Is Your Educational Background?

○ 1 - Elementary

○ 2 - High School

○ 3 - Undegraduate

○ 4 - Graduate

summary statistics of EDA

Numerical Data

1. Discrete Data

- discrete data contains values as distinct and separate.
- This type of data **can't be measured but it can be counted**.
- It basically represents information that can be categorized into a classification.
- An example is the number of heads in 100 coin flips.

summary statistics of EDA

Numerical Data

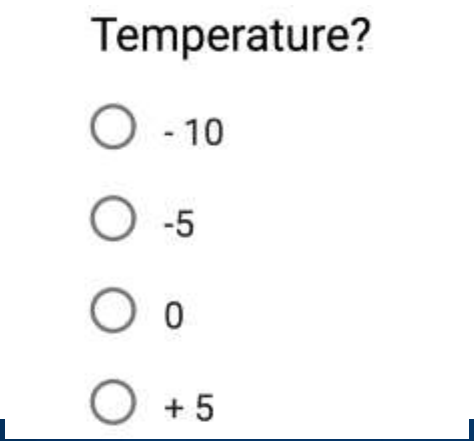
2. Continuous Data

- Continuous Data represents measurements and therefore their values **can't be counted but they can be measured**.
- An example would be the height of a person, which can describe by using intervals on the real number line.

Interval Data

- Interval values represent **ordered units that have the same difference**.

Example:



Temperature?

○ - 10

○ -5

○ 0

○ + 5

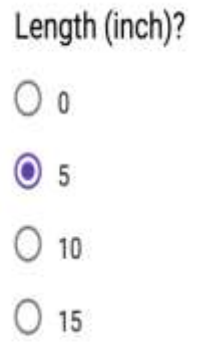
summary statistics of EDA

Numerical Data

Ratio Data

- Ratio values are also ordered units that have the same difference.
- Ratio values are **the same as interval values, with the difference that they do have an absolute zero**.
- Good examples are height, weight, length etc.

Example:


Length (inch)?

○ 0

[✓ 5]

○ 10

○ 15

summary statistics of EDA

Types of EDA

- ➢The EDA types of techniques are either graphical or quantitative (non-graphical).
- ➢ While the graphical methods involve summarising the data in a diagrammatic or visual way.
- ➢ the quantitative method, on the other hand, involves the calculation of summary statistics.
- ➢ These two types of methods are further divided into univariate and multivariate methods.

summary statistics of EDA

Types of EDA

- Univariate methods consider one variable (data column) at a time.
- multivariate methods consider two or more variables at a time to explore relationships.
- Totally there are four types of EDA .

1. univariate graphical,
2. multivariate graphical,
3. univariate non-graphical, and
4. multivariate non-graphical.

summary statistics of EDA

Types of EDA

Univariate non-graphical:

- This is the simplest form of data analysis among the four options.
- In this type of analysis, the data that is being analysed consists of just a single variable.
- The main purpose of this analysis is to describe the data and to find patterns.

summary statistics of EDA

Types of EDA

Univariate graphical:

- the graphical method provides the full picture of the data.
- The three main methods of analysis under this type are histogram, stem and leaf plot, and box plots.
- The histogram represents the total count of cases for a range of values.
- Along with the data values, the stem and leaf plot shows the shape of the distribution.
- The box plots graphically depict a summary of minimum, first quartile median, third quartile, and maximum.

summary statistics of EDA

Types of EDA

Multivariate non-graphical:

➢The multivariate non-graphical type of EDA generally depicts the relationship between multiple variables of data through cross-tabulation or statistics.

Multivariate graphical:

➢This type of EDA displays the relationship between two or more set of data.

➢A bar chart, where each group represents a level of one of the variables and each bar within the group represents levels of other variables.

summary statistics of EDA

Summary Statistics of EDA

- one purpose of EDA is to spot problems in data (as part of data wrangling) and understand variable properties like:

1. central trends (mean)
2. spread (variance)
3. skew
4. suggest possible modeling strategies (e.g., probability distributions)

- EDA is used to understand relationship between pairs of variables, e.g. their correlation or covariance.

summary statistics of EDA

Summary Statistics of EDA

COVARIANCE

- Covariance is a measure of relationship between 2 variables that is scale dependent, i.e. how much will a variable change when another variable changes.
- This can be represented with the following equation:

$$\text{Covariance}\,(x, y) = \sum \frac{(x_i - \bar{x})(y_i - \bar{y})}{N - 1}$$

xi is the ith observation in variable x,

x¯ is the mean for variable x,

yi is the ith observation in variable y,

y¯ is the mean for variable y, and

N is the number of observations

summary statistics of EDA

Summary Statistics of EDA

COVARIANCE

➢ this can be calculated easily within Python - particulatly when using Pandas as

```
import pandas as pd
df = pd.DataFrame()
df = pd.read_csv(“data.csv")
df.cov()
```

CORRELATION

➢It is a statistical metric to measure what extent different variables are interdependent.

➢In short, if one variable changes, how does it affect other variable.

summary statistics of EDA

Summary Statistics of EDA

CORRELATION

$$r = \sum \frac{(x_i - \bar{x})(y_i - \bar{y})}{(N-1)(s_x)(s_y)}$$

Where,

xi is the ith observation in variable x,

x¯ is the mean for variable x,

yi is the ith observation in variable y,

y¯ is the mean for variable y, and

N is the number of observations

sx is the standard deviation for variable x

sy is the standard deviation for variable y

summary statistics of EDA

Summary Statistics of EDA

CORRELATION

- this can be calculated easily within Python - particulatly when using Pandas as

```
import pandas as pd
df = pd.DataFrame()
df = pd.read_csv(“data.csv")

df.corr()
```

Philosophy of EDA

<u>Philosophy of Exploratory Data Analysis</u>

➢ The important reasons to implement EDA when working with data are:

1. to gain intuition about the data;
2. to make comparisons between distributions;
3. for sanity checking (making sure the data is on the scale we expect, in the format we thought it should be);
4. to find out where data is missing or if there are outliers; and to summarize the data.

➢In the context of data generated from logs, EDA also helps with debugging the logging process.

Philosophy of EDA

Philosophy of Exploratory Data Analysis

- ➢In the end, EDA helps us to make sure the product is performing as intended.
- ➢ There's lots of visualization involved in EDA.
- ➢The distinguish between EDA and data visualization is that EDA is done toward the beginning of analysis, and data visualization is done toward the end to Communicate one's findings.
- ➢With EDA, the graphics are solely done for us to understand what's going on.
- ➢EDA are used to improve the development of algorithms.

Pandas

What is Pandas?

➢ Pandas is one of the most widely used python libraries in data science.

➢ Pandas provide efficient, easy-to-use data structure and data analysis tools.

➢ The name Pandas is derived from "Panel Data" - an Econometrics from Multidimensional Data.

➢Using Pandas five typical steps in the processing and analysis of data can be accomplished. (removing duplicates, finding missed values, …etc)

➢ Pandas is well suited for many different kinds of data:

1. Tabular data with heterogeneously-typed columns
2. Ordered and unordered time series data
3. Arbitrary matrix data with row & column labels
4. Unlabelled data
5. Any other form of observational or statistical data sets

Pandas

Core components of pandas (or) Data Structures

➢ Pandas provides three data structures. Those are: -

1. Series (1D)
2. Data Frames (2D)
3. Panels (3D)

➢ All data structures are build on top of the NumPy array.

➢All Pandas data structures are value mutable (can be changed) and except Series all are size mutable.

➢Series is size immutable.

Pandas

Core components of pandas (or) Data Structures

1. **Series**

- A Series is a single-dimensional array structures that stores homogenous data i.e., data of a single type.
- All the elements of a Series are value-mutable and size-immutable.
- Data can be of multiple data types such as ndarray, lists, constants, series, dict etc.
- A pandas Series can be created using the following constructor

pandas.Series(data, index, dtype, copy)

Pandas

Series

Sno	Parameter & Description
1	**Data** - data takes various forms like ndarray, list, constants
2	**Index** - Index values must be unique and hashable, same length as data.
3	**Dtype** - dtype is for data type. If None, data type will be inferred
4	**Copy -** Copy data. Default False

- A series can be created using various inputs like –

1. Scalar value or constant
2. Dict
3. Array

Pandas

Core components of pandas (or) Data Structures

Creating Series:

1. Creating an empty series.

```
import pandas as pd
s = pd.Series()
print s
```

Output:

```
Series([], dtype: float64)
```

Pandas

<u>Creating Series:</u>

2. Creating series with scalar values.

Example 1:

```
import pandas as pd
Data =[10, 30, 40, 50, 60, 20, 90]
# Creating series with default index values
s = pd.Series(Data)
print s
```

Output:

```
  0     10
  1     30
  2     40,
 3      50
 4       60
 5      20
 6       90
```

Pandas

Creating Series:

2. Creating series with scalar values

Example 2:

```
import pandas as pd
Data =[10, 30, 40, 50, 60, 20, 90]
# predefined index values
Index =['a', 'b', 'c', 'd', 'e', 'f', 'g']
# Creating series with predefined index values
si = pd.Series(Data, Index)
print si
```

Output:

```
  a      10
   b     30
  c      40,
 d       50
 e        60
 f       20
 g        90
```

Pandas

Creating Series:

2. Creating series with scalar values

Example 3:

```
import pandas as pd
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

output :

```
0       5
1       5
2        5
3        5
dtype: int64
```

Pandas

Creating Series:

3. Create a Series from ndarray

Example 1 :

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Output:

```
0   a
1   b
2   c
3   d
dtype: object
```

Pandas

Creating Series:

Example 2:

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print s
```

Note:

- If data is an ndarray, then index passed must be of the same length.
- If no index is passed, then by default index will be **range(n)** where **n** is array length.

output:

```
100   a
101   b
102   c
103   d
dtype: object
```

Pandas

Creating Series:

4. Create a Series from dict

➢ A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index.

➢ If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

Pandas

Creating Series:

Example 1:

```
Import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

Output:

```
a 0.0
b 1.0
c 2.0
dtype: float64
```

Pandas

Creating Series:

Example 2:

```
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print s
```

Output:

```
b 1.0
c 2.0
d NaN
a 0.0
dtype: float64
```

Pandas

Accessing Data from Series with Position

- Data in the series can be accessed similar to that in an **ndarray.**

Examples:

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print   s[0]
print   s[:3]
print   s[-3:]
```

Output:

```
1
a  1   b  2   c  3
dtype: int64

c  3   d  4  e  5
dtype: int64
```

Pandas

Retrieve Data Using Label (Index)

- A Series is like a fixed-size **dict** .
- label can be used to get and set values.

Example:

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print s['a']
print s[['a','c','d']]
print s['f']
```

Output:

```
1
a  1  c  3   d  4
dtype: int64
KeyError: 'f'
```

Pandas

2. DataFrame:

➢ A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

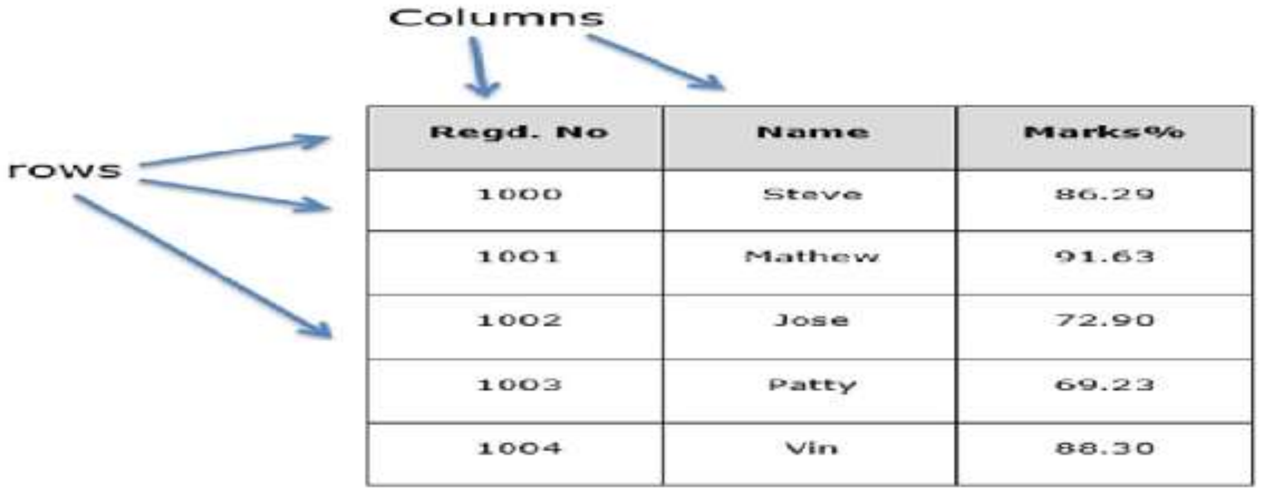
Features of DataFrame

1. Potentially columns are of different types
2. Size – Mutable
3. Labeled axes (rows and columns)
4. Can Perform Arithmetic operations on rows and columns

Pandas

2. DataFrame:

Consider the structure:



Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Consider it as an SQL table or a spreadsheet data representation.

Pandas

➢A pandas DataFrame can be created using the following constructor −

pandas.DataFrame(data, index, columns, dtype, copy)

The parameters of the constructor are as follows −

S.No	Parameter & Description
1	**Data -** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index-** For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
3	**columns-** For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
4	**dtype-** Data type of each column.
5	**copy-** This command (or whatever it is) is used for copying of data, if the default is False.

Pandas

2. DataFrame:

- A pandas DataFrame can be created using various inputs like –

1. Lists
2. dict
3. Series
4. Numpy ndarrays
5. Another DataFrame

Pandas

Creating DataFrame

➢An Empty Dataframe is created as follows:

```
import pandas as pd
df = pd.DataFrame()
print df
```

➢The DataFrame can be created using a single list or a list of lists.

Example 1:

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Output:

```
Empty
DataFrameColumn
s: []Index: []
```

output:

```
      0
0   1
1    2
2    3
3    4
4     5
```

Pandas

Creating DataFrame

Example 2:

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Output:

```
        Name     Age
  0     Alex     10
  1      Bob     12
  2     Clarke   13
```

- Create a DataFrame from Dict of ndarrays / Lists

Example 1

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve',
'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

output:

```
        Age     Name
0       28       Tom
1       34      Jack
2       29     Steve
 3      42     Ricky
```

Pandas

Example 2:

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve',
'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data,
index=['rank1','rank2','rank3','rank4'])
print df
```

Output:

```
         Age   Name
rank1    28     Tom
rank2    34    Jack
rank3    29   Steve
rank4    42   Ricky
```

Note:

- All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.
- If no index is passed, then by default, index will be range(n), where **n** is the array length.

Pandas

Creating DataFrames

➢Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame.

The dictionary keys are by default taken as column names.

Example 1

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Output:

```
     a   b     c
0    1   2   NaN
1    5  10  20.0
```

NaN (Not a Number) is appended in missing areas.

Pandas

Create a DataFrame from List of Dicts

Example 2:

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
 #With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b'])
print df1
#With two column indices with one index with other
name
df2 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b1'])
print df2
```

Output:

```
#df1 output
          a    b
first     1    2
second  5    10
#df2 output
          a    b1
first     1   NaN
second   5  NaN
```

Pandas

Creating DataFrames

- **Create a DataFrame from Dict of Series**

Dictionary of Series can be passed to form a DataFrame.
The resultant index is the union of all the series indexes passed.
Example

```
import pandas as pd d = {'one' : pd.Series([1, 2, 3],
index=['a', 'b', 'c']),   'two' : pd.Series([1, 2, 3, 4],
 index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df
```

Output:

```
         one   two
a         1.0   1
b         2.0   2
c         3.0   3
d         NaN   4
```

there is no label **'d'** passed, but in the result, for the **d** label, NaN is appended with NaN.

Pandas

Column Selection

Example:

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df ['one']
```

Output:

```
 a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype:
float64
```

Pandas

Column Addition

Example:

```
import pandas as pd
 d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
        'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print df
print ("Adding a new column using the existing
columns in DataFrame:")
df['four']=df['one']+df['three']
print df
```

Output:

```
Adding a new column by passing
as Series:
     one   two   three
a    1.0    1    10.0
b    2.0    2    20.0
c    3.0    3    30.0
d    NaN   4    NaN
Adding a new column using the
existing columns in DataFrame:
     one   two   three    four
a    1.0    1    10.0     11.0
b    2.0    2    20.0     22.0
c    3.0    3    30.0     33.0
d    NaN    4    NaN      NaN
```

Pandas

Column Deletion

➢ Del or pop methods are used.

Example:

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
         'three' : pd.Series([10,20,30], index=['a','b','c'])}
 df = pd.DataFrame(d)
print ("Our dataframe is:")
print df
print ("Deleting the first column using DEL function:")
del df['one']
print df
```

Output:

```
Our dataframe is:
        one   three  two
a       1.0   10.0   1
b       2.0   20.0   2
c       3.0   30.0   3
d       NaN    NaN   4

Deleting the first
column using DEL
function:
        three   two
a        10.0    1
b       20.0    2
c       30.0    3
```

Pandas

Column Deletion

```
print ("Deleting another column using POP function:")
df.pop('two')
print df
```

output:

```
Deleting another column using POP function:
       three
a      10.0
b      20.0
c      30.0
D      NaN
```

Pandas

Row Selection, Addition, and Deletion

Selection:

- Rows can be selected by passing row label to a **loc** function.

Example:

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df.loc['b‘]
```

Output:

```
one    2.0
Two    2.0
Name: b, dtype: float64
```

Pandas

Row Selection, Addition, and Deletion

Selection:

- Rows can be selected by passing integer location to an **iloc** function.

Example

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',     'd'])}
df = pd.DataFrame(d)
print df.iloc[2]
```

Output:

```
   one   3.0
    two   3.0
Name: c, dtype: float64
```

Pandas

Row Selection, Addition, and Deletion

Addition of Rows

- Add new rows to a DataFrame using the **append** function.
- This function will append the rows at the end.

```
import pandas as pd
 df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
print df
```

Output:

```
        a   b
0       1   2
1       3   4
0       5   6
1       7   8
```

Pandas

Row Selection, Addition, and Deletion

Deletion of Rows

- ➢Use index label to delete or drop rows from a DataFrame.
- ➢ If label is duplicated, then multiple rows will be dropped.

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
 df = df.append(df2)
 # Drop rows with label 0
df = df.drop(0)
print df
```

Output:

```
       a   b
1      3   4
1      7   8
```

Pandas

- ➢Dataframes can also be easily exported and imported from CSV, Excel, JSON, HTML and SQL database.
- ➢Some other essential methods that are present in data frames are:

- **head():** returns the top 5 rows in the dataframe object
- **tail():** returns the bottom 5 rows in the dataframe
- **info():** prints the summary of the dataframe
- **describe():** gives a nice overview of the main aggregated values over each column

Pandas

3.Panel

- A **panel** is a 3D container of data.
- A Panel can be created using the following constructor −

pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)

The parameters of the constructor are as follows −

Parameter	Description
data	Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame
items	each item in this axis corresponds to one data frame, and this is called axis 0.
major_axis	This axis actually contains the rows or indexes of each of the data frames, and this is called axis 1
minor_axis	This axis actually contains all the columns of each of the data frames, and this is called axis 2.
dtype	Data type of each column
copy	Copy data. Default, **false**

Pandas

Creating a Panel

➢A Panel can be created using multiple ways like –

1. From ndarrays
2. From dict of DataFrames

1.From 3D ndarray

```
import pandas as pd
import numpy as np
 pan = pd.Panel(np.random.randn(5,10,5),items = ['A'
   ,'B' ,'C', 'D','E'],major_axis =
   pd.date_range('1/2/2019', periods=10),
   minor_axis=['1','2','3','4','5'])
print(pan)
```

Output:

```
<class 'pandas.core.panel.Panel'>
Dimensions: 5 (items) x 10
(major_axis) x 5 (minor_axis)
Items axis: A to E
Major_axis axis: 2019-01-02 00:00:
to 2019-01-11 00:00:00
Minor_axis axis: 1 to 5
```

Pandas

Creating a Panel

2. From dict of DataFrames

Example:

```
import pandas as pd
import numpy as np


data = {'Item1' :
  pd.DataFrame(np.random.randn(4, 3)),
          'Item2' :
  pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p
```

Output:

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_ax
x 3 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 2
```

Pandas

Operations on Panel Items

➢ Some of the basic operations performed on panel data structure are:

1. Selection of Panel items
2. Addition of Panel Items
3. Deletion of Panel Items
4. Panel transpose
5. Data selection from a Panel

Pandas

<u>Pandas - Basic Functionality</u>

➢Some Basic Functionalities are:

S.No.	**Attribute or Method & Description**
1	**axes -** Returns a list of the row axis labels
2	**Dtype -** Returns the dtype of the object.
3	**Empty -** Returns True if series is empty.
4	**Ndim -** Returns the number of dimensions of the underlying data, by definition 1.
5	**Size -** Returns the number of elements in the underlying data.
6	**Values -** Returns the Series as ndarray.

Pandas

Pandas - Basic Functionality

➢Some Basic Functionalities are:

S.No.	**Attribute or Method & Description**
7	**head() -** Returns the first n rows.
8	**tail() -** Returns the last n rows.

Pandas

Pandas - Descriptive Statistics

➢A large number of methods collectively compute descriptive statistics and other related operations on DataFrame .

➢ Most of the are aggregations like **sum(), mean(),…etc.**

Steps to Get the Descriptive Statistics for Pandas DataFrame

Step 1: Collect the Data

collect the data for DataFrame.

For example, consider the following data related to cars:

Pandas

Pandas - Descriptive Statistics

Brand	Price	Year
• Honda Civic	22000	2014
• Ford Focus	27000	2015
• Toyota Corolla	25000	2016
• Toyota Corolla	29000	2017
• Audi A4	35000	2018

Pandas

Pandas - Descriptive Statistics

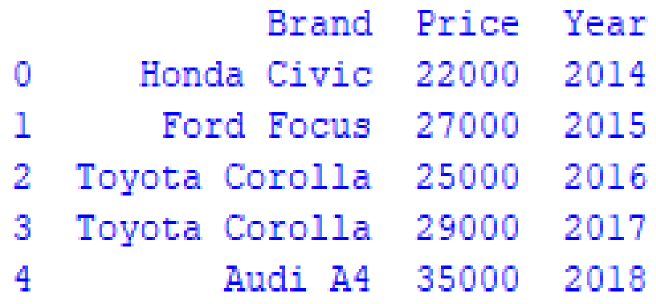
Step 2: Create the DataFrame

create the DataFrame based on the data collected.

```
from pandas import DataFrame
Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota
  Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
        'Year': [2014,2015,2016,2017,2018]     }
df = DataFrame(Cars, columns= ['Brand',
  'Price','Year'])
print (df)
```

output

DataFrame is as follows:



```
            Brand  Price  Year
0     Honda Civic  22000  2014
1      Ford Focus  27000  2015
2  Toyota Corolla  25000  2016
3  Toyota Corolla  29000  2017
4         Audi A4  35000  2018
```

Pandas

Pandas - Descriptive Statistics

Step 3: Get the Descriptive Statistics for Pandas DataFrame

Once Data Frame is ready we can get the descriptive statistics.

Syntax:

```
df['DataFrame Column'].describe()
```

➢ To get the descriptive statistics for the ‘Price’ field, which contains *numerical data.*

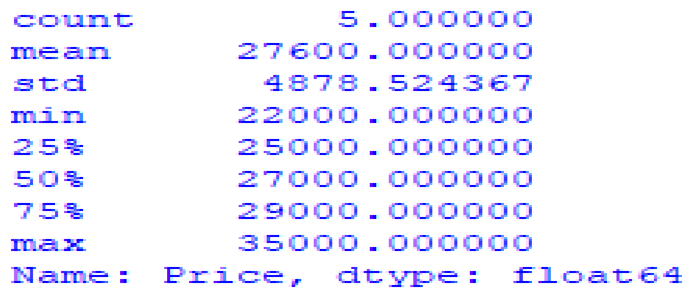
```
df['Price'].describe()
```

Pandas

Pandas - Descriptive Statistics

```
from pandas import DataFrame
Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota
  Corolla','Toyota Corolla','Audi A4'],
     'Price': [22000,27000,25000,29000,35000],
      'Year': [2014,2015,2016,2017,2018]      }
df = DataFrame(Cars, columns= ['Brand',
  'Price','Year'])
stats_numeric = df['Price'].describe()
print (stats_numeric)
```

output
the descriptive
statistics for
the ‘Price’ field
is:



```
count        5.000000
mean     27600.000000
std       4878.524367
min      22000.000000
25%      25000.000000
50%      27000.000000
75%      29000.000000
max      35000.000000
Name: Price, dtype: float64
```

Pandas

Pandas - Descriptive Statistics

➢Use **astype (int)** method to get integer values.

from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],

'Price': [22000,27000,25000,29000,35000],

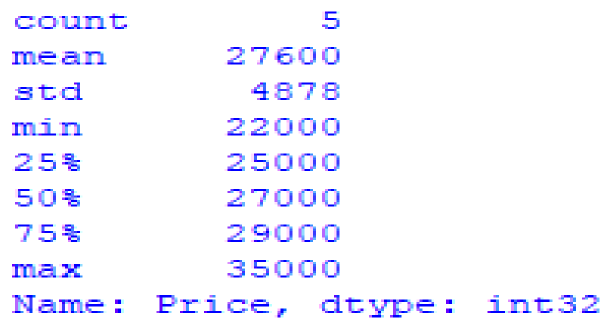
'Year': [2014,2015,2016,2017,2018] }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

- stats_numeric = df['Price'].describe().astype (int)
- print (stats_numeric)

output

Output contains only integers:



```
count        5
mean     27600
std       4878
min      22000
25%      25000
50%      27000
75%      29000
max      35000
Name: Price, dtype: int32
```

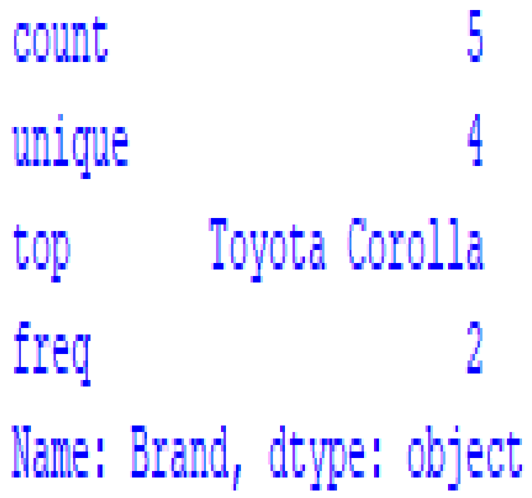
Pandas

Descriptive Statistics for Categorical Data

➢To get descriptive statistics for the ‘Brand’ field is as follows:

```
from pandas import DataFrame
Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]       }
df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])
stats_categorical = df['Brand'].describe()
print (stats_categorical)
```

Output:



```
count                  5
unique                 4
top       Toyota Corolla
freq                   2
Name: Brand, dtype: object
```

Pandas

Breaking Down the Descriptive Statistics

1. **Count**:

```
df['DataFrame Column'].count()
```

2. **Mean**:

```
df['DataFrame Column'].mean()
```

3. **Standard deviation**:

```
df['DataFrame Column'].std()
```

4. **Minimum**:

```
df['DataFrame Column'].min()
```

5. **0.25 Quantile**:

```
df['DataFrame Column'].quantile(q=0.25)
```

Pandas

Breaking Down the Descriptive Statistics

6. **0.50 Quantile (Median)**:

```
df['DataFrame Column'].quantile(q=0.50)
```

7. **0.75 Quantile**:

```
df['DataFrame Column'].quantile(q=0.75)
```

8. **Maximum**:

```
df['DataFrame Column'].max()
```

Pandas

Example:

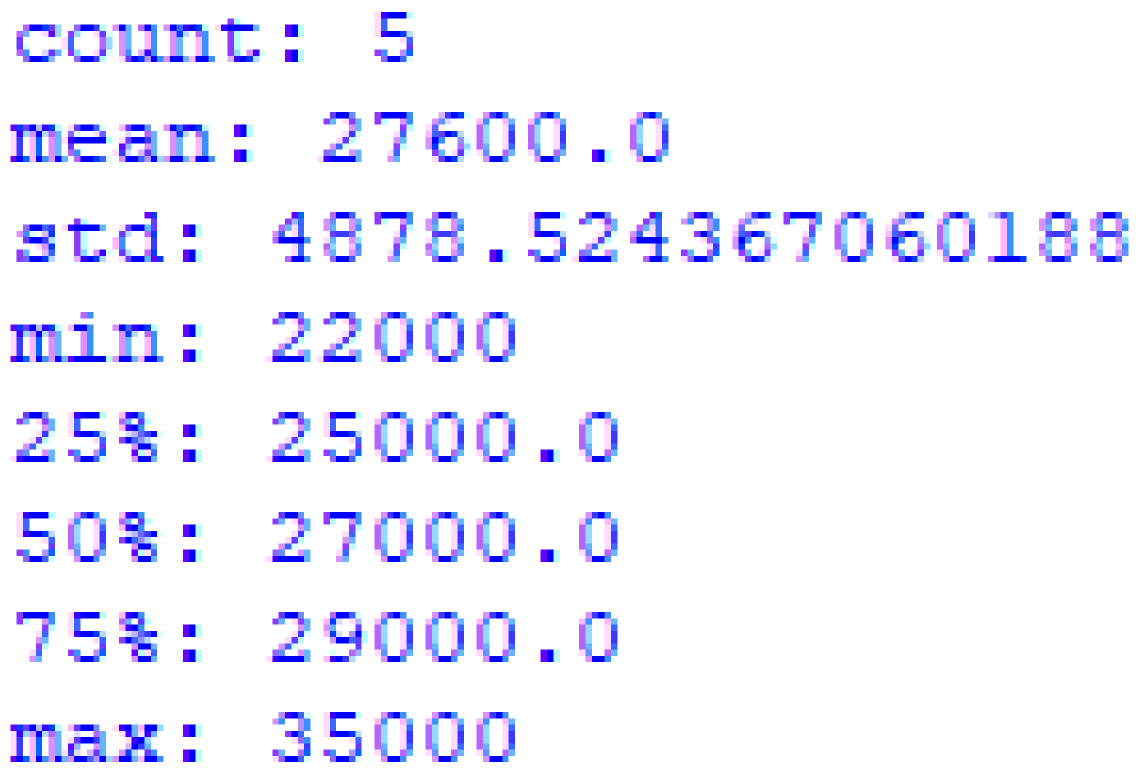
```
from pandas import DataFrame
Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi
   A4'], 'Price': [22000,27000,25000,29000,35000],
  'Year': [2014,2015,2016,2017,2018]        }
df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])
count1 = df['Price'].count()
print('count: ' + str(count1))
mean1 = df['Price'].mean()
print('mean: ' + str(mean1))
std1 = df['Price'].std()
print('std: ' + str(std1))
```

Pandas

```
min1 = df['Price'].min()
print('min: ' + str(min1))
quantile1 = df['Price'].quantile(q=0.25)
print('25%: ' + str(quantile1))
quantile2 = df['Price'].quantile(q=0.50)
print('50%: ' + str(quantile2))
quantile3 = df['Price'].quantile(q=0.75)
print('75%: ' + str(quantile3))
max1 = df['Price'].max()
print('max: ' + str(max1))
```

Pandas

Output:



```
count: 5
mean: 27600.0
std: 4878.524367060188
min: 22000
25%: 25000.0
50%: 27000.0
75%: 29000.0
max: 35000
```

Thank You