Exploratory Data Analysis (EDA):

It refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Understanding EDA with a Dataset

Domain: Automobile Industry

The Automotive (Automobile) industry is **the industry of automobiles**. It is the industry that designs, develops, manufactures, markets, and sells motor vehicles, and is one of the earth's most important and largest economic sectors by revenue.

Toyota Dataset:

- Toyota Data set contains used cars information about Price, Horse Power,Number of KM travalled,Fuel type,Age,Aotomatic,number of doors,MetalColor,CC & Weight.

Summary of DataFrame

import pandas as pd
import numpy as np
stoyota=pd.read_csv('../input/toyatacars/Toyota.csv')
stoyota.head()
stoyota.tail()

(i) Checking format of each column

info( ) – returns concise summary of dataframe

```
[23] stoyota.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price        1436 non-null int64
Age          1336 non-null float64
KM           1436 non-null object
FuelType     1336 non-null object
HP           1436 non-null object
MetColor     1286 non-null float64
Automatic    1436 non-null int64
CC           1436 non-null int64
Doors        1436 non-null object
Weight       1436 non-null int64
dtypes: float64(2), int64(4), object(4)
memory usage: 123.4+ KB
```

By using info( ), we can see that 'KM' has been read as object instead of integer 'HP' has been read as object instead of integer. 'MetColor' and 'Automatic' have been read as float64 and int64 respectively since it has values 0/1.Ideally, 'Doors' should've been read as int64 since it has values 2, 3, 4, 5. But it has been read as Object.

(ii) Finding unique elements of columns

- ***unique( )*** – returns unique elements of a column

```
[19] stoyota['HP'].unique()

     array(['90', '????', '192', '110', '97', '71', '116', '98', '69', '86',
            '72', '107', '73'], dtype=object)
```

```
[28] import numpy as np
     np.unique(stoyota['KM'])

     array(['1', '10000', '100123', ..., '99865', '99971', '??'], dtype=object)
```

```
[29] stoyota['Doors'].unique()

     array(['three', '3', '5', '4', 'four', 'five', '2'], dtype=object)
```

```
[20] stoyota['Automatic'].unique()

     array([0, 1])
```

```
[18] stoyota['MetColor'].unique()

     array([ 1., nan,  0.])
```

Values 0. , 1. and 0 , 1 made these attributes treated as 'float' and 'int', but these are categories.

Importing data with other forms of missing values .We need to know how missing values are represented in the dataset in order to makereasonable decisions.

Missing values exists in the form of 'nan', '??' and '????' Python, by default replace blank values with 'nan'. Now, import the data considering other forms of missing values in a dataframe.

Observation :

- 'KM' has been read as object instead of float64
- 'HP' has been read as object instead of float64
- 'MetColor' and 'Automatic' have been read as float64 and int64 respectively since it has values 0/1
- Ideally, 'Doors' should've been read as int64 since it has values 2, 3, 4, 5. But it has been read as Object.
- Missing values present in few variables

(iii) Converting Variable's data types

Converting Variable's data types

- Converting 'MetColor' , 'Automatic' to object data type:
- ***astype( )*** - explicitly converts from one to another data type
- Syntax: dataframe.***astype(****dtype****)***

```
[ ]  stoyota['MetColor']=stoyota['MetColor'].astype('object')
     stoyota.MetColor.dtype

     dtype('O')
```

```
[ ]  stoyota['Automatic']=stoyota['Automatic'].astype('object')
     stoyota.Automatic.dtype

     dtype('O')
```

Recheck the data types

Summary – before converting variable's data type	Summary – after converting variable's data type

Before:

```
[14] stoyota.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price        1436 non-null int64
Age          1336 non-null float64
KM           1421 non-null float64
FuelType     1336 non-null object
HP           1430 non-null float64
MetColor     1286 non-null float64
Automatic    1436 non-null int64
CC           1436 non-null int64
Doors        1436 non-null object
Weight       1436 non-null int64
dtypes: float64(4), int64(4), object(2)
memory usage: 123.4+ KB
```

After:

```
[37] stoyota.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price        1436 non-null int64
Age          1336 non-null float64
KM           1421 non-null float64
FuelType     1336 non-null object
HP           1430 non-null float64
MetColor     1286 non-null object
Automatic    1436 non-null object
CC           1436 non-null int64
Doors        1436 non-null object
Weight       1436 non-null int64
dtypes: float64(3), int64(3), object(4)
memory usage: 123.4+ KB
```

(iv) Cleaning 'Doors' Column

- Checking unique values of variable 'Doors':

```
print(np.unique(stoyota['Doors']))

['2' '3' '4' '5' 'five' 'four' 'three']
```

```
[41] stoyota['Doors'].replace('three',3,inplace=True)
     stoyota['Doors'].replace('four',4,inplace=True)
     stoyota['Doors'].replace('five',5,inplace=True)
```

(v) Converting 'Doors' datatype

- Converting 'Doors' into int64

```
[45] stoyota['Doors'] = stoyota['Doors'].astype('int64')
```

- replace() is used to replace a value with the desired value
- Syntax: DataFrame.replace([to_replace, value, …])

```
6] stoyota.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price          1436 non-null int64
Age            1336 non-null float64
KM             1421 non-null float64
FuelType       1336 non-null object
HP             1430 non-null float64
MetColor       1286 non-null object
Automatic      1436 non-null object
CC             1436 non-null int64
Doors          1436 non-null int64
Weight         1436 non-null int64
dtypes: float64(3), int64(4), object(3)
memory usage: 123.4+ KB
```

Identifying missing values

- Check the count of missing values present in each column
- *Dataframe.**isnull( ).sum( )***

```
[165] stoyota.isnull().sum()

      Price          0
      Age          100
      KM            15
      FuelType     100
      HP             6
      MetColor     150
      Automatic      0
      CC             0
      Doors          0
      Weight         0
      dtype: int64
```

```
[169] stoyota.isna().sum()

      Price          0
      Age          100
      KM            15
      FuelType     100
      HP             6
      MetColor     150
      Automatic      0
      CC             0
      Doors          0
      Weight         0
      dtype: int64
```

Handling missing values

Two ways:

- Deleting missing values
- Imputing/filling the missing values

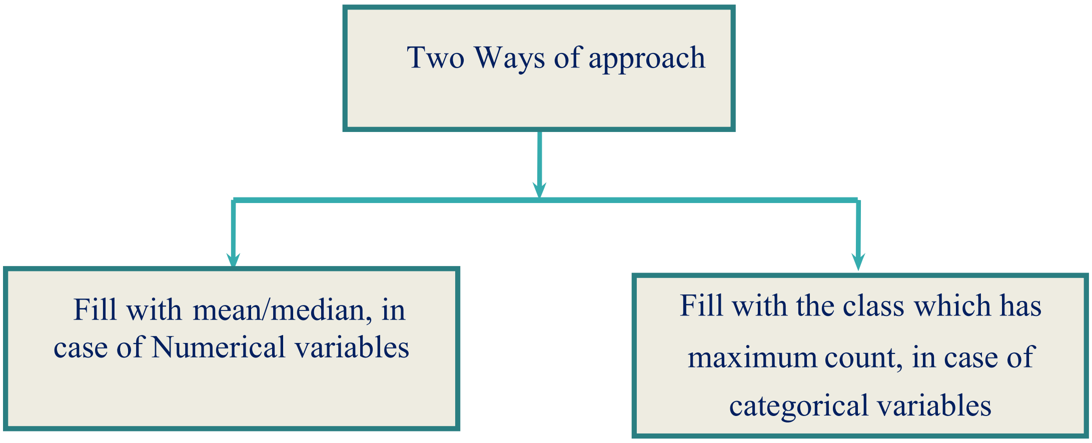
(i) Handling missing values - by deleting

- ***dropna():*** Method used to drop rows/cols containing missing values

```
[85]  stoyota.dropna(inplace=True)
      stoyota.isnull().sum()

      Price        0
      Age          0
      KM           0
      FuelType     0
      HP           0
      MetColor     0
      Automatic    0
      CC           0
      Doors        0
      Weight       0
      dtype: int64
```

```
[83]  stoyota.shape

      (1096, 10)
```

(ii) Handling missing values- by imputing




```
[48] stoyota.isnull().sum()

Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

- *DataFrame.**describe( ):*** Returns the statistical information about the data frame

```
[ ] stoyota.describe()
```

	Price	Age	KM	HP	CC	Doors	Weight
count	1436.000000	1336.000000	1421.000000	1430.000000	1436.000000	1436.000000	1436.00000
mean	10730.824513	55.672156	68647.239972	101.478322	1566.827994	4.033426	1072.45961
std	3626.964585	18.589804	37333.023589	14.768255	187.182436	0.952677	52.64112
min	4350.000000	1.000000	1.000000	69.000000	1300.000000	2.000000	1000.00000
25%	8450.000000	43.000000	43210.000000	90.000000	1400.000000	3.000000	1040.00000
50%	9900.000000	60.000000	63634.000000	110.000000	1600.000000	4.000000	1070.00000
75%	11950.000000	70.000000	87000.000000	110.000000	1600.000000	5.000000	1085.00000
max	32500.000000	80.000000	243000.000000	192.000000	2000.000000	5.000000	1615.00000

Imputing missing values for ‘Age’ variable

```
[58] stoyota['Age'].mean()

55.67215568862275
```

- fillna() is used to fill NA/NaN values using the specified value
- Syntax: *DataFrame.**fillna()***

```
[ ] stoyota['Age'].fillna(stoyota['Age'].mean(),inplace=True)
```

Before imputing Null values

```
[ ] print(missing)

      Price   Age       KM FuelType  ...
2     13950  24.0  41711.0   Diesel  ...
6     16900  27.0      NaN   Diesel  ...
7     18600  30.0  75889.0      NaN  ...
9     12950  23.0  71138.0   Diesel  ...
15    22000  28.0  18739.0   Petrol  ...
...     ...   ...      ...      ...  ...
1428   8450  72.0      NaN   Petrol  ...
1431   7500   NaN  20544.0   Petrol  ...
1432  10845  72.0      NaN   Petrol  ...
1433   8500   NaN  17016.0   Petrol  ...
1434   7250  70.0      NaN      NaN  ...

[340 rows x 10 columns]
```

After imputing Null values

```
[ ] stoyota['Age'].tail(10)

1426    78.000000
1427    55.672156
1428    72.000000
1429    78.000000
1430    80.000000
1431    55.672156
1432    72.000000
1433    55.672156
1434    70.000000
1435    76.000000
Name: Age, dtype: float64
```

Imputing missing values for ‘KM’

```
[61] stoyota['KM'].median()

63634.0
```

```
[ ] stoyota['KM'].fillna(stoyota['KM'].median(),inplace=True)
```

```
[ ]  stoyota['KM'].head(10)

0     46986.0
1     72937.0
2     41711.0
3     48000.0
4     38500.0
5     61000.0
6     63634.0
7     75889.0
8     19700.0
9     71138.0
Name: KM, dtype: float64
```

Imputing missing values for 'HP'

```
[107] stoyota['HP'].mean()

101.47832167832168
```

```
[108] stoyota['HP'].fillna(stoyota['HP'].mean(),inplace=True)
```

```
[99] missing.loc[1:50]
```

	Price	Age	KM	FuelType	HP
2	13950	24.0	41711.0	Diesel	90.0
6	16900	27.0	NaN	Diesel	NaN
7	18600	30.0	75889.0	NaN	90.0
9	12950	23.0	71138.0	Diesel	NaN
15	22000	28.0	18739.0	Petrol	NaN
21	16950	29.0	43905.0	NaN	110.0
26	17495	27.0	34545.0	NaN	110.0

```
[109] stoyota['HP'].loc[5:20]

    5      90.000000
    6     101.478322
    7      90.000000
    8     192.000000
    9     101.478322
    10    192.000000
    11    192.000000
    12    192.000000
    13    192.000000
    14    192.000000
    15    101.478322
    16    192.000000
    17    110.000000
    18    110.000000
    19    110.000000
    20    110.000000
    Name: HP, dtype: float64
```

Imputing missing values for 'FuelType'

- Most frequently occurred category is to be found
- Syntax: Series.***value_counts()***
- Returns a Series containing counts of unique values in descending order
- First element will be the most frequently-occurring element
- Excludes 'nan' values by default

```
[ ] stoyota['FuelType'].value_counts()

    Petrol    1277
    Diesel     144
    CNG         15
    Name: FuelType, dtype: int64
```

```
[ ] stoyota['FuelType'].fillna(stoyota['FuelType'].value_counts().index[0],inplace=True)
```

Imputing missing values for 'MetColor'

```
[110] stoyota['MetColor'].mode()

    0    1
    dtype: object
```

```
[111] stoyota['MetColor'].fillna(stoyota['MetColor'].mode().index[0],inplace=True)
```

Rechecking missing values

Before imputing Null values

```
[48] stoyota.isnull().sum()

Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

After imputing Null values

```
[ ] stoyota.isnull().sum()

Price         0
Age           0
KM            0
FuelType      0
HP            0
MetColor      0
Automatic     0
CC            0
Doors         0
Weight        0
dtype: int64
```

Correlation

```
DataFrame.corr(self, method='pearson')
```

It is used to compute pair wise correlation of columns excluding NA/null values
Excluding the categorical variables to find the Pearson's correlation

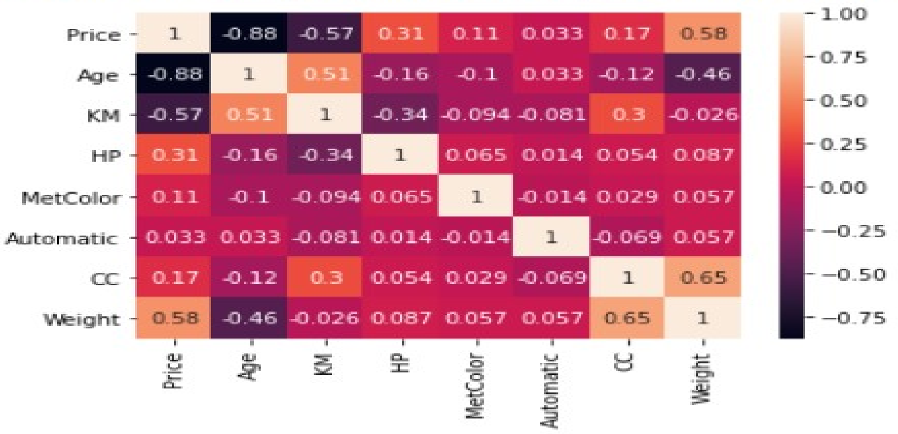
A strong negative correlationbetween Age and Price.

A Fair positive correlation between Weight and Price& KM and Age.

A Fair negative correlation between KM and Price.

Correlation – using heatmap
Import seaborn as sns
corr_matrix = stoyota.corr()
sns.heatmap(corr_matrix, annot=True)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb85b065e48>
```



EDA for Categorical Variables

What is the problem with Categorical data?

Categorical data are variables that contain label values rather than numeric values.

- The number of possible values is often limited to a fixed set.
- Categorical variables are often called '*nominal*'.

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric. This means that categorical data must be converted to a numerical form.

Converting categorical variables into numeric

- Label Encoding
- One-Hot Encoding

Label Encoding

Label encoding is simply converting each value in a column to a number

One method is converting a column to a category, and then uses those categoryvalues for your label encoding.

```
[ ]  df1['FuelType'] = df1['FuelType'].astype('category')
```

Then assign the encoded variable to a new column using the '*cat.codes'* accessor

One – Hot Encoding

A new binary variable is added for each of the categorical value. Pandas supports this feature using '*get_dummies()*'

Syntax:

```
pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None) ¶
```

```
df1["FuelType_cat"] = df1["FuelType"].cat.codes
print(df1['FuelType_cat'].value_counts())
print(df1['FuelType'].value_counts())

2     968
1     116
0      12
Name: FuelType_cat, dtype: int64
Petrol     968
Diesel     116
CNG         12
Name: FuelType, dtype: int64
```

```
new_carsdf = pd.get_dummies(stoyota)
new_carsdf.shape
```

```
(1436, 13)
```

```
new_carsdf.columns
```

```
Index(['Price', 'Age', 'KM', 'HP', 'MetColor', 'CC', 'Doors', 'Weight',
       'FuelType_CNG', 'FuelType_Diesel', 'FuelType_Petrol', 'Automatic_0',
       'Automatic_1'],
      dtype='object')
```

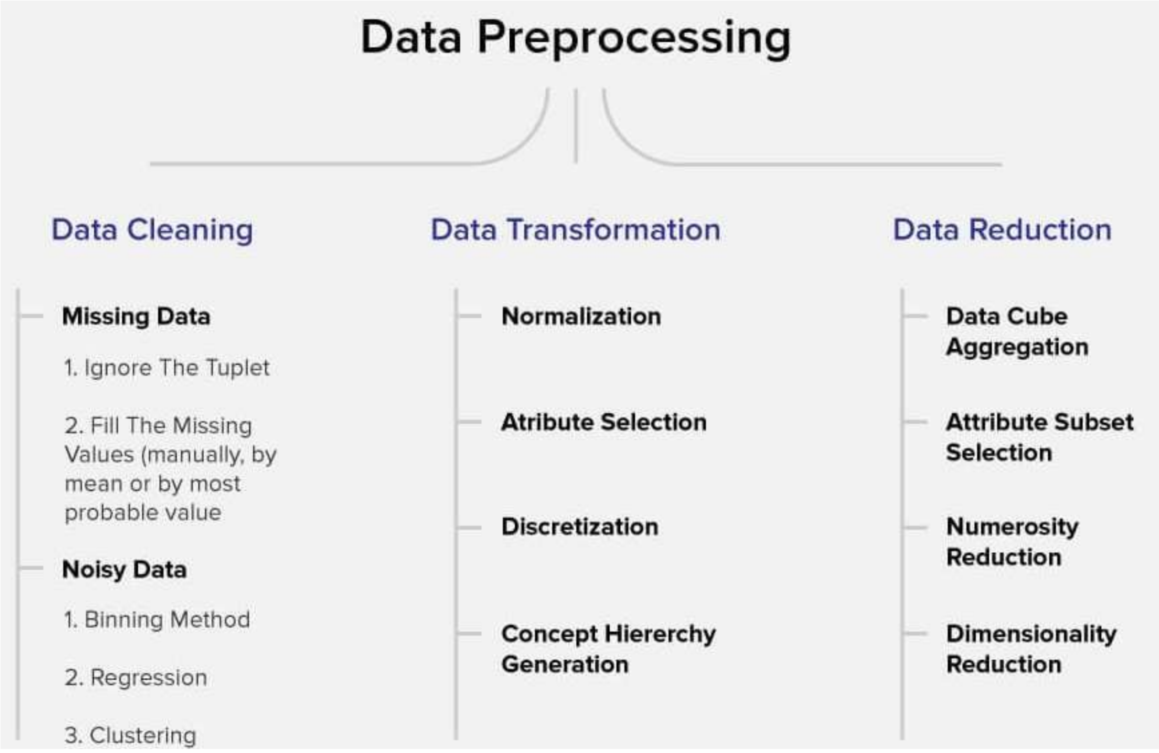
```
new_carsdf.iloc[1:6,7:]
```

	Weight	FuelType_CNG	FuelType_Diesel	FuelType_Petrol	Automatic_0	Automatic_1
1	1165	0	1	0	1	0
2	1165	0	1	0	1	0
3	1165	0	1	0	1	0
4	1170	0	1	0	1	0
5	1170	0	1	0	1	0

```
new_carsdf.dtypes
```

```
Price                int64
Age                float64
KM                 float64
HP                 float64
MetColor           float64
CC                   int64
Doors                int64
Weight               int64
FuelType_CNG         uint8
FuelType_Diesel      uint8
FuelType_Petrol      uint8
Automatic_0          uint8
Automatic_1          uint8
dtype: object
```

Data Preprocessing



Data preprocessing is a technique which is used to transform the raw data in a useful and efficient format.

Steps Involved in Data Pre-processing:

1. Data Cleaning:

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

(a) Missing Data:
This situation arises when some data is missing in the data. It can be handled in various ways.Some of them are:

- **Ignore the tuples:**

  This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.

- **Fill the Missing values:**

  There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

(b) Noisy Data:

Noisy data is a meaningless data that can't be interpreted by machines. It can be generated due to faulty data collection, data entry errors etc. It canbe handled in following ways:

- **Binning Method:**

  This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values.

- **Regression:**

  Here data can be made smooth by fitting it to a regression function. The regression used may be linear (having one independent variable) or multiple (having multiple independent variables).

- **Clustering:**

  This approach groups the similar data in a cluster. The outliers may be undetected or it will fall outside the clusters.

2. Data Transformation:

This step is taken in order to transform the data in appropriate forms. This involves following ways:

(a) Normalization:

It is done in order to scale the data values in a specified range (-1.0 to1.0 or 0.0 to 1.0)

(b) Attribute Selection:

In this strategy, new attributes are constructed from the given set of attributes to help the mining process.

(c) Discretization:

This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.

(d) Concept Hierarchy Generation:

Here attributes are converted from lower level to higher level in hierarchy. Example-The attribute "city" can be converted to country".

3. Data Reduction

The size of the dataset in a data warehouse can be too large to be handled by data analysis and data mining algorithms.

One possible solution is to obtain a reduced representation of thedataset that is much smaller in volume but produces the same quality of analytical results. Here is a walkthrough of various Data Reduction strategies.

(a) Data cube aggregation

It is a way of data reduction, in which the gathered data is expressed in a summary form.

(b) Numerosity reduction

The data can be represented as a model or equation like a regressionmodel. This would save the burden of storing huge datasets insteadof a model.

(c) Attribute subset selection

It is very important to be specific in the selection of attributes.

Otherwise, it might lead to high dimensional data, which are difficult to train due to underfitting / overfitting problems. Only attributes that add more value towards model training should be considered, and the rest all can be discarded.

(d) Dimensionality reduction

Dimensionality reduction techniques are used to perform feature extraction. The dimensionality of a dataset refers to the attributes orindividual features of the data. This technique aims to reduce the number of redundant features we consider in machine learning algorithms. Dimensionality reduction can be done using techniqueslike Principal Component Analysis etc

Data visualization

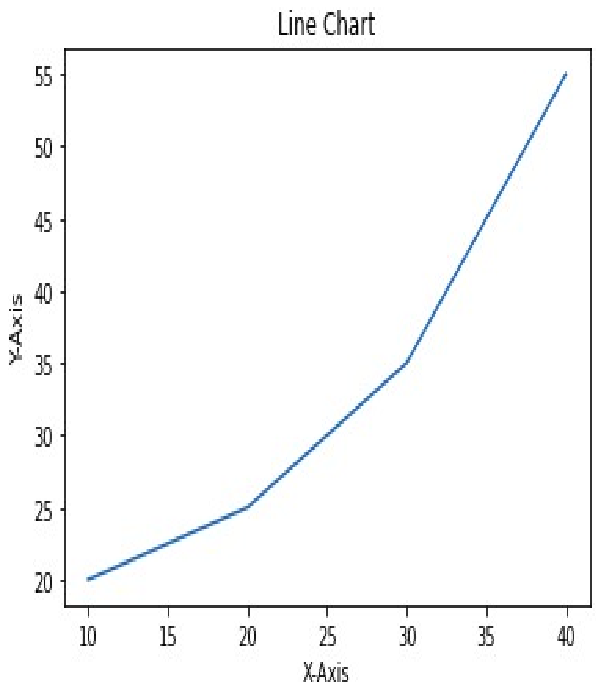
- Data visualization is the graphical representation of information and data. By using visual elements likecharts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

- Matplotlib has a module called pyplot which aids in plotting figure.
- Importing required libraries and dataset to plot using Pandas pd.read_csv()
- plt.plot()for plotting line chart similarly in place of plot other functions are used for plotting.
- plot.xlabel , plt.ylabel for labeling x and y-axis respectively.
- plt.title() for setting the title of the plot.
- plot.show() for displaying the plot.

(a) Line Graph/Chart

Line chart is one of the basic plots and can be created using the plot() function. It is used to represent a relationship between two data X and Y on a different axis.

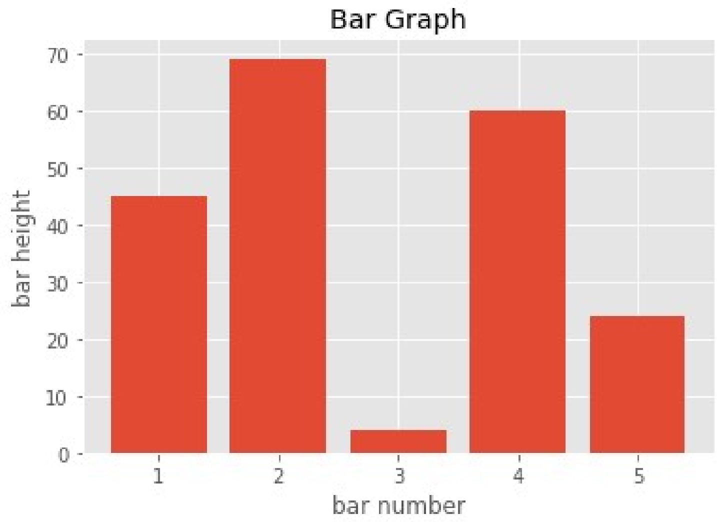
```
import matplotlib.pyplot as plt


# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]

# plotting the data
plt.plot(x, y)

# Adding title to the plot
plt.title("Line Chart")

# Adding label on the y-axis
plt.ylabel('Y-Axis')

# Adding label on the x-axis
plt.xlabel('X-Axis')

plt.show()
```



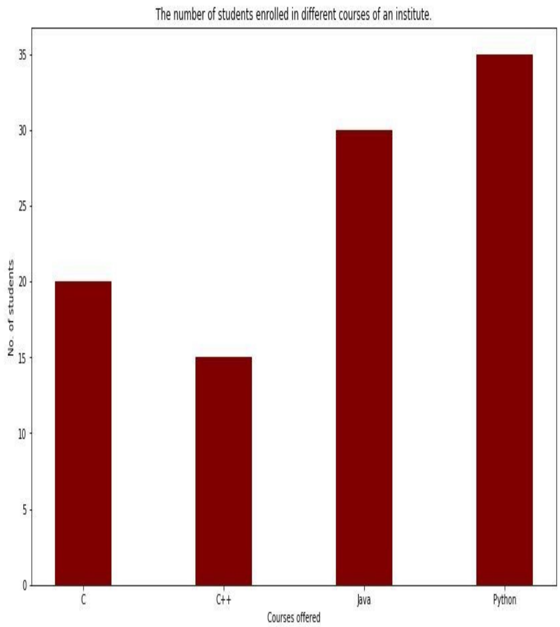
(b) Bar Graph

- Bar graphs used to show the changes over time and to compare attributes.
- Bar graph will have an x-axis (horizontal) and a y-axis (vertical).
- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.

```
#import matplotlib
from matplotlib import pyplot as plt
#Bar Graph
plt.bar([1,2,3,4,5],[45,69,4,60,24])
plt.xlabel('bar number')
plt.ylabel('bar height')
plt.title("Bar Graph")
plt.show()
```



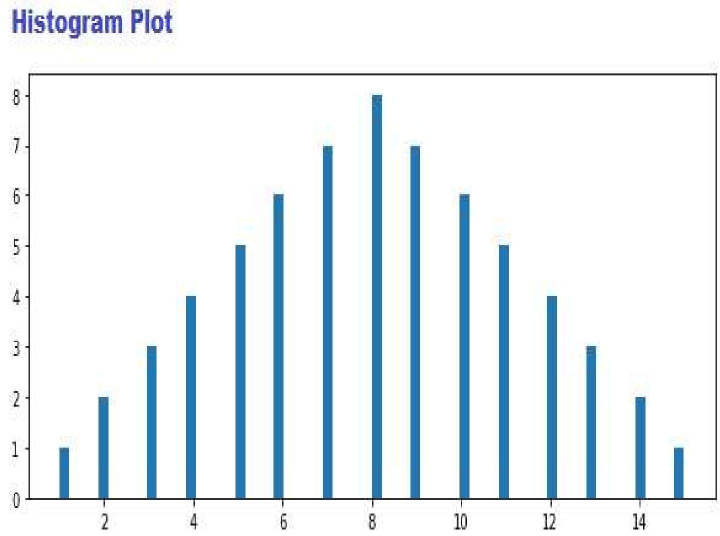
```python
import numpy as np
import matplotlib.pyplot as plt


# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```



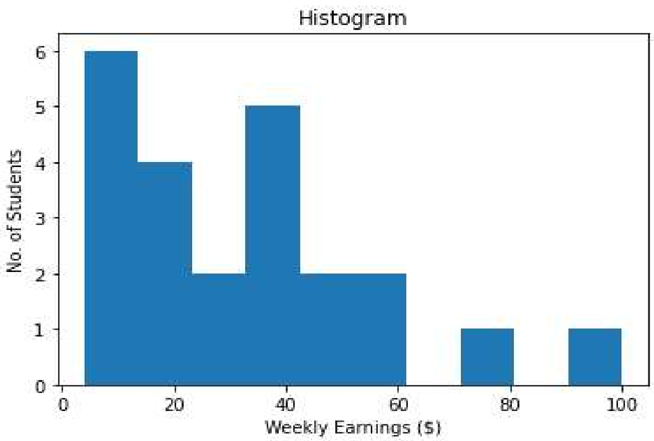
(c) Histogram

- A histogram takes in a series of data and divides the data into a number of bins. It then plots the frequency data points in each bin (i.e. the interval of points). It is useful in understanding the count of data ranges.
- Histogram graph looks like Bar Graph but this is continuous type of chart. In a histogram, each bar groups numbers into ranges.
- Taller bars show that more data falls in that range.
- Bins are used to create data with n number of bins

```
import numpy as np
import matplotlib.pyplot as plt
Dataset1 =
    [1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,10,10,10,10,10,10,11,11,11,11,11,12,12,12,12,13,13,13,14,14,15]
y=Dataset1
plt.figure(figsize=(10, 4))
plt1 = plt.hist(Dataset1,bins=64)
plt.show()
```




```
from matplotlib import pyplot as plt
x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,60,55,35,36,37,18,49,50,100]
num_bins = 10
plt.hist(x, num_bins)
plt.xlabel("Weekly Earnings ($)")
plt.ylabel("No. of Students")
plt.title("Histogram")
plt.show()
```

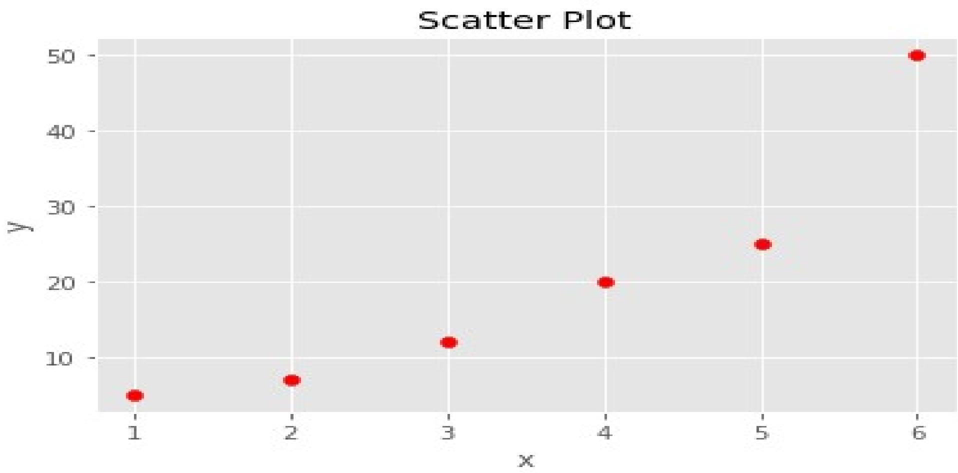


(d) Scatter Plot

- A scatter plot uses dots to represent values for two different numeric variables. The position of each dot onthe horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

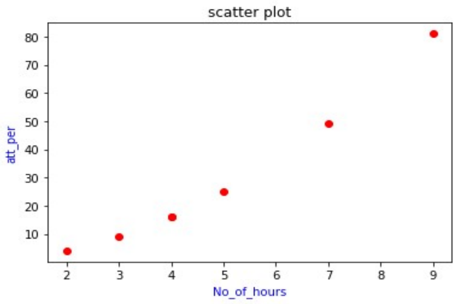
Code:

from matplotlib import pyplot as plt
x= [1,2,3,4,5,6]
y= [5,7,12,20,25,50]
plt.scatter(x,y, color = 'r') plt.xlabel('x')plt.ylabel('y') plt.title('Scatter Plot')plt.show()




```
import numpy as np
import matplotlib.pyplot as plt
x=np.array([2,3,4,5,7,9,4])
print(x)
y=x**2
print(y)
plt.scatter(x,y,c='r')
plt.xlabel('No_of_hours',c='blue')
plt.ylabel('att_per',c='blue')
plt.title('scatter plot')
plt.figure(figsize=(5,5))
plt.show()
```

```
[2 3 4 5 7 9 4]
[ 4  9 16 25 49 81 16]
```

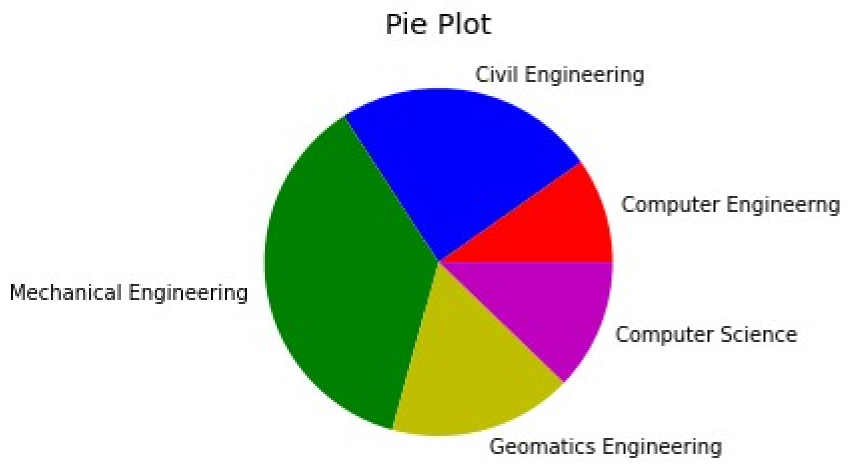



```
<Figure size 360x360 with 0 Axes>
```

(e) Pie Chart

- A pie chart is a circular statistical graph, which is divided into slices to illustrate numerical proportion. In a pie chart, the arc length of each slice is proportional to the quantity it represents.

```
from matplotlib import pyplot as plt
students = [400, 1000, 1500, 700, 500]
interests = ['Computer Engineering','Civil Engineering','Mechanical Engineering','Geomatics Engineering','Computer Science']
col= ['r','b','g','y','m']
plt.pie(students,labels =interests, colors= col)
plt.title('Pie Plot')
plt.show()
```


Pie Plot
Civil Engineering
Computer Engineerng
Mechanical Engineering
Computer Science
Geomatics Engineering

```
import numpy as np
import matplotlib.pyplot as plt
x=np.array([2,3,4,5,7,9,4])
print(x)
y=x**2
print(y)
plt.subplot(231)
plt.scatter(x,y,c='r')
plt.subplot(232)
plt.bar(x,y)
plt.subplot(233)
plt.pie(x)
plt.subplot(234)
plt.boxplot(y)
plt.subplot(235)
plt.plot(x,x/y)
plt.subplot(236)
plt.hist(y)
plt.show()
```

```
[2 3 4 5 7 9 4]
[ 4  9 16 25 49 81 16]
```