1. Write a Python program to Find union and intersection of two list

In [3]:
```
l1 = list()
l2 = list()
l3 = list()
l4= list()

n1 = int(input("Enter the number of elements in first list:"))
for i in range(n1):
    ele = int(input("Enter the element of first list:"))
    l1.append(ele)

print("First list is: ",l1)

n2 = int(input("Enter the number of elements in second list:"))
for i in range(n2):
    ele = int(input("Enter the element of second list:"))
    l2.append(ele)

print("second list is: ",l2)

l3.extend(l1)
for ele in l2:
    if ele in l3:
        continue
    l3.append(ele)

print("Union of:", l1, "and ", l2, "is: ",l3)

for ele in l1:
    if ele in l2:
        l4.append(ele)

print ("Intersection of ", l1, "and ", l2, "is", l4)
```

```
Enter the number of elements in first list:3
Enter the element of first list:2
Enter the element of first list:3
Enter the element of first list:4
First list is:  [2, 3, 4]
Enter the number of elements in second list:3
Enter the element of second list:3
Enter the element of second list:4
Enter the element of second list:5
second list is:  [3, 4, 5]
Union of: [2, 3, 4] and  [3, 4, 5] is:  [2, 3, 4, 5]
Intersection of  [2, 3, 4] and  [3, 4, 5] is [3, 4]
```

In [ ]:

2.Write a Python program to map two lists into a dictionary and create manual dataset.

Most pythonic and generic method to perform this very task is by using zip(). This function pairs the list element with other list element at corresponding index in form of key-value pairs.

In [6]:
```
# Python3 code to demonstrate
# conversion of lists to dictionary
# using zip()

# initializing lists
test_keys = ["Rash", "Kil", "Varsha"]
test_values = [1, 4, 5]

# Printing original keys-value lists
print ("Original key list is : " + str(test_keys))
print ("Original value list is : " + str(test_values))

# using zip()
# to convert lists to dictionary
res = dict(zip(test_keys, test_values))

# Printing resultant dictionary
print ("Resultant dictionary is : " + str(res))
```

```
Original key list is : ['Rash', 'Kil', 'Varsha']
Original value list is : [1, 4, 5]
Resultant dictionary is : {'Rash': 1, 'Kil': 4, 'Varsha': 5}
```

In [ ]:

3. Write a Program to handle missing value using fillna(), replace() and interpolate()

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- fillna()
- replace()
- interpolate()

1. fillna()

In [28]:
```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
# filling a missing value with previous ones
df1=df.fillna(method ='pad')
# filling a missing value with forward fill method
df2=df.fillna(method ='ffill')
# filling a missing value with backward fill method
df3=df.fillna(method ='bfill')
print(df1)
print("\n")
print(df2)
print("\n")
print(df3)
```

```
   First Score  Second Score  Third Score
0        100.0          30.0          NaN
1         90.0          45.0         40.0
2         90.0          56.0         80.0
3         95.0          56.0         98.0


   First Score  Second Score  Third Score
0        100.0          30.0          NaN
1         90.0          45.0         40.0
2         90.0          56.0         80.0
3         95.0          56.0         98.0


   First Score  Second Score  Third Score
0        100.0          30.0         40.0
1         90.0          45.0         40.0
2         95.0          56.0         80.0
3         95.0           NaN         98.0
```

2. replace()

In [29]:
```
df
# will replace  Nan value in dataframe with value -99
df.replace(to_replace = np.nan, value = -99)
```

Out[29]:

	First Score	Second Score	Third Score
0	100.0	30.0	-99.0
1	90.0	45.0	40.0
2	-99.0	56.0	80.0
3	95.0	-99.0	98.0

3. Using interpolate() function to fill the missing values using linear method.

Interpolate the missing values using Linear method. Note that Linear method ignore the index and treat the values as equally spaced

In [30]:
```
# to interpolate the missing values
df.interpolate(method ='linear', limit_direction ='forward')
```

Out[30]:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	92.5	56.0	80.0
3	95.0	56.0	98.0

4. Dropping missing values using dropna()

In [31]:
```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
print(df)
#drop NaN value
clean_data= df.dropna()
print("\nAfter DropNa():\n",clean_data)
```

```
   First Score  Second Score  Third Score  Fourth Score
0        100.0          30.0           52           NaN
1         90.0           NaN           40           NaN
2          NaN          45.0           80           NaN
3         95.0          56.0           98          65.0

After DropNa():
    First Score  Second Score  Third Score  Fourth Score
3         95.0          56.0           98          65.0
```

4. Write a Program to import the dataset from local disk , sklearn library and Kaggle API.

A. Import a CSV and excel File into Python From local disk using Pandas:

Step 1: Capture the File Path
Step 2: Apply the Python code
Step 3: Run the Code

In [6]:
```
#import CSV file
import pandas as pd
dataset= pd.read_csv("car.csv")
dataset.head(5)
```

Out[6]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing XO eRLX Euro III	Hyundai	2007	80,000	45,000 kms	Petrol
1	Mahindra Jeep CL550 MDI	Mahindra	2006	4,25,000	40 kms	Diesel
2	Maruti Suzuki Alto 800 Vxi	Maruti	2018	Ask For Price	22,000 kms	Petrol
3	Hyundai Grand i10 Magna 1.2 Kappa VTVT	Hyundai	2014	3,25,000	28,000 kms	Petrol
4	Ford EcoSport Titanium 1.5L TDCi	Ford	2014	5,75,000	36,000 kms	Diesel

In [7]:
```
#import excel file
import pandas as pd
dataset= pd.read_excel("T-shirt dataset.xls")
dataset.head(5)
```

Out[7]:

	HEIGHT(IN CMS)	WEIGHT(IN KGS)	T SHIRT SIZE
0	158	58	M
1	158	59	M
2	158	63	M
3	160	59	M
4	160	60	M

B. Dataset from Kaggle API

API - Application Programming Interface

which is a software intermediary that allows two applications to talk to each other. In other words, an API is the messenger that delivers your request to the provider that you're requesting it from and then delivers the response back to you.

Steps to Import a data from Kaggle API:

Steps:

1. Create Kaggle account
2. Get Your API (kaggle.json file)

3. Installing the kaggle library
4. Upload your kaggle.json file
5. Configure some setups

In [8]:
```
#installing the kaggle library
!pip install kaggle
```

```
Requirement already satisfied: kaggle in c:\users\dell\anaconda3\lib\site-packages (1.
5.12)
Requirement already satisfied: python-slugify in c:\users\dell\anaconda3\lib\site-packa
ges (from kaggle) (5.0.2)
Requirement already satisfied: python-dateutil in c:\users\dell\anaconda3\lib\site-pack
ages (from kaggle) (2.8.2)
Requirement already satisfied: six>=1.10 in c:\users\dell\anaconda3\lib\site-packages
(from kaggle) (1.16.0)
Requirement already satisfied: certifi in c:\users\dell\anaconda3\lib\site-packages (fr
om kaggle) (2021.5.30)
Requirement already satisfied: tqdm in c:\users\dell\anaconda3\lib\site-packages (from
kaggle) (4.62.3)
Requirement already satisfied: urllib3 in c:\users\dell\anaconda3\lib\site-packages (fr
om kaggle) (1.26.6)
Requirement already satisfied: requests in c:\users\dell\anaconda3\lib\site-packages (f
rom kaggle) (2.26.0)
Requirement already satisfied: text-unidecode>=1.3 in c:\users\dell\anaconda3\lib\site-
packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\dell\anaconda3\lib
\site-packages (from requests->kaggle) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dell\anaconda3\lib\site-package
s (from requests->kaggle) (2.10)
Requirement already satisfied: colorama in c:\users\dell\anaconda3\lib\site-packages (f
rom tqdm->kaggle) (0.4.4)
```

In [10]:
```
#API to fetch the dataset from kaggle
!kaggle datasets download -d schirmerchad/bostonhoustingmlnd
```

```
Downloading bostonhoustingmlnd.zip to C:\Users\dell\ML LAB


  0%|          | 0.00/4.35k [00:00<?, ?B/s]
100%|##########| 4.35k/4.35k [00:00<00:00, 2.23MB/s]
```

In [12]:
```
#Extract data the compressed data
from zipfile import ZipFile
dataset_path = 'bostonhoustingmlnd.zip'
with ZipFile(dataset_path, 'r') as zip:
  zip.extractall()
```

In [15]:
```
import pandas as pd
dataset=pd.read_csv('housing.csv')
dataset.head(5)
```

Out[15]:

	RM	LSTAT	PTRATIO	MEDV
0	6.575	4.98	15.3	504000.0
1	6.421	9.14	17.8	453600.0
2	7.185	4.03	17.8	728700.0
3	6.998	2.94	18.7	701400.0
4	7.147	5.33	18.7	760200.0

C.Dataset import from sklearn library

In [26]:
```
from sklearn import datasets
from sklearn.datasets import load_digits
digits = load_digits()
```

In [27]:
```
print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits (http
s://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits)

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [ ]:

5. Write a Program to handle categorical data (One Hot Enconding)

Code: Python code implementation of Manual One-Hot Encoding Technique Loading the data

In [7]:
```
# Program for demonstration of one hot encoding

# import libraries
import numpy as np
import pandas as pd

# import the data required
data = pd.read_excel("T-shirt dataset.xls")
print(data.head())
```

```
   HEIGHT(IN CMS)  WEIGHT(IN KGS) T SHIRT SIZE
0             158              58            M
1             158              59            M
2             158              63            M
3             160              59            M
4             160              60            M
```

In [8]:
```
#Checking for the labels in the categorical parameters
print(data['T SHIRT SIZE'].unique())
```

```
['M' 'L']
```

In [9]:
```
#Checking for the label counts in the categorical parameters
data['T SHIRT SIZE'].value_counts()
```

Out[9]:
```
M    15
L    11
Name: T SHIRT SIZE, dtype: int64
```

In [10]:
```
#One-Hot encoding the categorical parameters using get_dummies()
one_hot_encoded_data = pd.get_dummies(data, columns = ['T SHIRT SIZE'])
print(one_hot_encoded_data)
```

	HEIGHT(IN CMS)	WEIGHT(IN KGS)	T SHIRT SIZE_L	T SHIRT SIZE_M
0	158	58	0	1
1	158	59	0	1
2	158	63	0	1
3	160	59	0	1
4	160	60	0	1
5	163	60	0	1
6	163	61	0	1
7	160	64	1	0
8	163	64	1	0
9	165	61	1	0
10	165	62	1	0
11	165	65	1	0
12	168	62	1	0
13	168	63	1	0
14	168	66	1	0
15	170	63	1	0
16	170	64	1	0
17	170	68	1	0
18	156	55	0	1
19	157	57	0	1
20	158	61	0	1
21	159	62	0	1
22	160	59	0	1
23	161	55	0	1
24	162	58	0	1
25	163	60	0	1

One Hot Encoding using Sci-kit learn Library:

In [11]:
```
#importing libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder

#Retrieving data
data = pd.read_excel("T-shirt dataset.xls")

# Converting type of columns to category
data['T SHIRT SIZE']=data['T SHIRT SIZE'].astype('category')



#Assigning numerical values and storing it in another columns
data['T SHIRT SIZE_new']=data['T SHIRT SIZE'].cat.codes


#Create an instance of One-hot-encoder
enc=OneHotEncoder()

#Passing encoded columns
enc_data=pd.DataFrame(enc.fit_transform(data[['T SHIRT SIZE_new']]).toarray())

#Merge with main
New_df=data.join(enc_data)

print(New_df)
```

	HEIGHT(IN CMS)	WEIGHT(IN KGS)	T SHIRT SIZE	T SHIRT SIZE_new	0	1
0	158	58	M	1	0.0	1.0
1	158	59	M	1	0.0	1.0
2	158	63	M	1	0.0	1.0
3	160	59	M	1	0.0	1.0
4	160	60	M	1	0.0	1.0
5	163	60	M	1	0.0	1.0
6	163	61	M	1	0.0	1.0
7	160	64	L	0	1.0	0.0
8	163	64	L	0	1.0	0.0
9	165	61	L	0	1.0	0.0
10	165	62	L	0	1.0	0.0
11	165	65	L	0	1.0	0.0
12	168	62	L	0	1.0	0.0
13	168	63	L	0	1.0	0.0
14	168	66	L	0	1.0	0.0
15	170	63	L	0	1.0	0.0
16	170	64	L	0	1.0	0.0
17	170	68	L	0	1.0	0.0
18	156	55	M	1	0.0	1.0
19	157	57	M	1	0.0	1.0
20	158	61	M	1	0.0	1.0
21	159	62	M	1	0.0	1.0
22	160	59	M	1	0.0	1.0
23	161	55	M	1	0.0	1.0
24	162	58	M	1	0.0	1.0
25	163	60	M	1	0.0	1.0

6. Write a Program to perform Data Normalization with Pandas.

Steps Needed

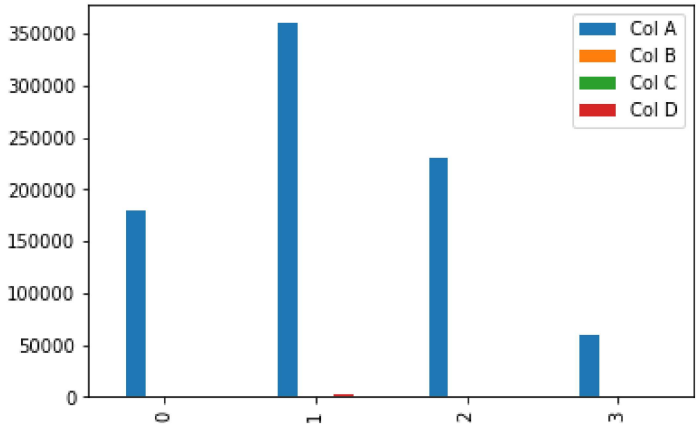
1.Import Library (Pandas)
2. Import / Load / Create data.
3. Use the technique to normalize the data.

In [2]:
```
# importing packages
import pandas as pd

# create data
df = pd.DataFrame([
            [180000, 110, 18.9, 1400],
            [360000, 905, 23.4, 1800],
            [230000, 230, 14.0, 1300],
            [60000, 450, 13.5, 1500]],
columns=['Col A', 'Col B','Col C', 'Col D'])
# view data
display(df)
```

	Col A	Col B	Col C	Col D
0	180000	110	18.9	1400
1	360000	905	23.4	1800
2	230000	230	14.0	1300
3	60000	450	13.5	1500

In [4]:
```
#See the plot of this dataframe:
import matplotlib.pyplot as plt
df.plot(kind = 'bar')
```

Out[4]: <AxesSubplot:>



Using The maximum absolute scaling

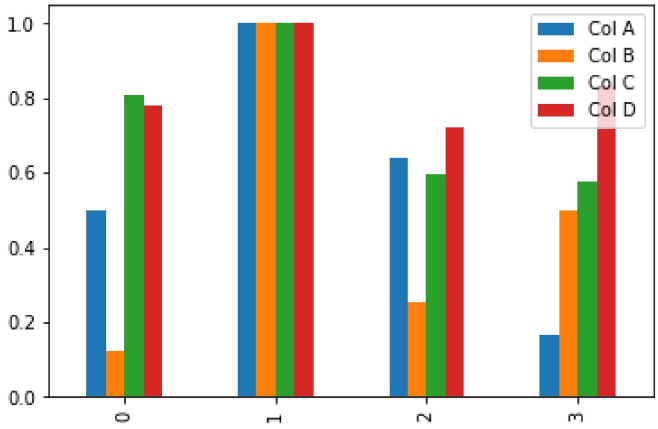
The maximum absolute scaling rescales each feature between -1 and 1 by dividing every observation by its maximum absolute value. We can apply the maximum absolute scaling in Pandas using the .max() and .abs() methods, as shown below.

In [6]:
```
# copy the data
df_max_scaled = df.copy()

# apply normalization techniques
for column in df_max_scaled.columns:
    df_max_scaled[column] = df_max_scaled[column] / df_max_scaled[column].abs().max()
# view normalized data
display(df_max_scaled)
```

	Col A	Col B	Col C	Col D
0	0.500000	0.121547	0.807692	0.777778
1	1.000000	1.000000	1.000000	1.000000
2	0.638889	0.254144	0.598291	0.722222
3	0.166667	0.497238	0.576923	0.833333

In [7]:
```
import matplotlib.pyplot as plt
df_max_scaled.plot(kind = 'bar')
```

Out[7]: <AxesSubplot:>



Using The min-max feature scaling

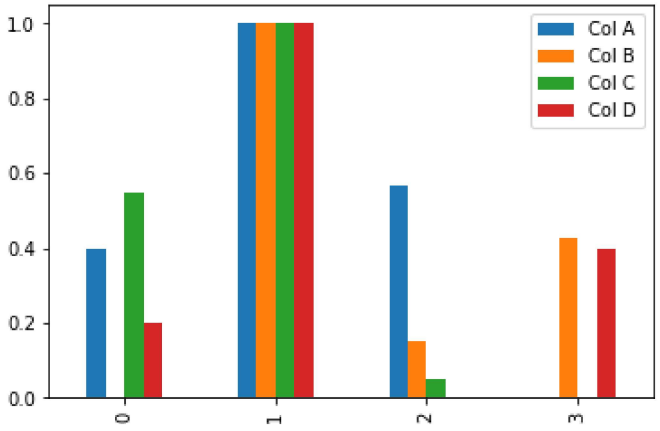
The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.

In [8]:
```
# copy the data
df_min_max_scaled = df.copy()

# apply normalization techniques
for column in df_min_max_scaled.columns:
    df_min_max_scaled[column] = (df_min_max_scaled[column] - df_min_max_scaled[column].m

# view normalized data
print(df_min_max_scaled)
```

```
      Col A     Col B     Col C  Col D
0  0.400000  0.000000  0.545455    0.2
1  1.000000  1.000000  1.000000    1.0
2  0.566667  0.150943  0.050505    0.0
3  0.000000  0.427673  0.000000    0.4
```

In [9]:
```
import matplotlib.pyplot as plt
df_min_max_scaled.plot(kind = 'bar')
```

Out[9]: <AxesSubplot:>



7. Write a Program to perform Data Normalization with sklearn.

In [22]:
```
import pandas as pd
from sklearn.datasets import load_boston
boston = load_boston()
feature=boston.feature_names
dataset=pd.DataFrame(boston.data, columns=feature)
dataset.head()
```

Out[22]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Standard Scaler

Standard Scaler helps to get standardized distribution, with a zero mean and standard deviation of one (unit variance). It standardizes features by subtracting the mean value from the feature and then dividing the result by feature standard deviation.

The standard scaling is calculated as:

z = (x - u) / s

In [36]:
```
# import module
from sklearn.preprocessing import StandardScaler
# compute required values
scaler = StandardScaler()
model = scaler.fit(dataset)
scaled_data = model.transform(dataset)
out=pd.DataFrame(scaled_data)
# print scaled data
print(out)
```

```
           0         1         2         3         4         5         6   \
0   -0.419782  0.284830 -1.287909 -0.272599 -0.144217  0.413672 -0.120013
1   -0.417339 -0.487722 -0.593381 -0.272599 -0.740262  0.194274  0.367166
2   -0.417342 -0.487722 -0.593381 -0.272599 -0.740262  1.282714 -0.265812
3   -0.416750 -0.487722 -1.306878 -0.272599 -0.835284  1.016303 -0.809889
4   -0.412482 -0.487722 -1.306878 -0.272599 -0.835284  1.228577 -0.511180
..        ...       ...       ...       ...       ...       ...       ...
501 -0.413229 -0.487722  0.115738 -0.272599  0.158124  0.439316  0.018673
502 -0.415249 -0.487722  0.115738 -0.272599  0.158124 -0.234548  0.288933
503 -0.413447 -0.487722  0.115738 -0.272599  0.158124  0.984960  0.797449
504 -0.407764 -0.487722  0.115738 -0.272599  0.158124  0.725672  0.736996
505 -0.415000 -0.487722  0.115738 -0.272599  0.158124 -0.362767  0.434732

           7         8         9         10        11        12
0    0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562
1    0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439
2    0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727
3    1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517
4    1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501
..        ...       ...       ...       ...       ...       ...
501 -0.625796 -0.982843 -0.803212  1.176466  0.387217 -0.418147
502 -0.716639 -0.982843 -0.803212  1.176466  0.441052 -0.500850
503 -0.773684 -0.982843 -0.803212  1.176466  0.441052 -0.983048
504 -0.668437 -0.982843 -0.803212  1.176466  0.403225 -0.865302
505 -0.613246 -0.982843 -0.803212  1.176466  0.441052 -0.669058

[506 rows x 13 columns]
```

MinMax Scaler

There is another way of data scaling, where the minimum of feature is made equal to zero and the maximum of feature equal to one. MinMax Scaler shrinks the data within the given range, usually of 0 to 1. It transforms data by scaling features to a given range. It scales the values to a specific value range without changing the shape of the original distribution.

The MinMax scaling is done using:

x_std = (x – x.min(axis=0)) / (x.max(axis=0) – x.min(axis=0))

x_scaled = x_std * (max – min) + min

In [40]:
```
# import module
from sklearn.preprocessing import MinMaxScaler
# compute required values
scaler = MinMaxScaler()
model = scaler.fit(dataset)
scaled_data = model.transform(dataset)
out=pd.DataFrame(scaled_data)
# print scaled data
print(out)
```

```
           0     1         2    3         4         5         6         7   \
0    0.000000  0.18  0.067815  0.0  0.314815  0.577505  0.641607  0.269203
1    0.000236  0.00  0.242302  0.0  0.172840  0.547998  0.782698  0.348962
2    0.000236  0.00  0.242302  0.0  0.172840  0.694386  0.599382  0.348962
3    0.000293  0.00  0.063050  0.0  0.150206  0.658555  0.441813  0.448545
4    0.000705  0.00  0.063050  0.0  0.150206  0.687105  0.528321  0.448545
..        ...   ...       ...  ...       ...       ...       ...       ...
501  0.000633  0.00  0.420455  0.0  0.386831  0.580954  0.681771  0.122671
502  0.000438  0.00  0.420455  0.0  0.386831  0.490324  0.760041  0.105293
503  0.000612  0.00  0.420455  0.0  0.386831  0.654340  0.907312  0.094381
504  0.001161  0.00  0.420455  0.0  0.386831  0.619467  0.889804  0.114514
505  0.000462  0.00  0.420455  0.0  0.386831  0.473079  0.802266  0.125072

           8         9         10        11        12
0    0.000000  0.208015  0.287234  1.000000  0.089680
1    0.043478  0.104962  0.553191  1.000000  0.204470
2    0.043478  0.104962  0.553191  0.989737  0.063466
3    0.086957  0.066794  0.648936  0.994276  0.033389
4    0.086957  0.066794  0.648936  1.000000  0.099338
..        ...       ...       ...       ...       ...
501  0.000000  0.164122  0.893617  0.987619  0.219095
502  0.000000  0.164122  0.893617  1.000000  0.202815
503  0.000000  0.164122  0.893617  1.000000  0.107892
504  0.000000  0.164122  0.893617  0.991301  0.131071
505  0.000000  0.164122  0.893617  1.000000  0.169702

[506 rows x 13 columns]
```

In [ ]:

8. Write a Program to Perform Data preprocessing on car datasets and save cleaned data in new file (CSV format)

In [38]:
```
import pandas as pd
import numpy as np
```

In [39]:
```
car=pd.read_csv('car.csv')
```

In [40]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        892 non-null    object
 1   company     892 non-null    object
 2   year        892 non-null    object
 3   Price       892 non-null    object
 4   kms_driven  840 non-null    object
 5   fuel_type   837 non-null    object
dtypes: object(6)
memory usage: 41.9+ KB
```

In [42]:
```
car.head(5)
```

Out[42]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing XO eRLX Euro III	Hyundai	2007	80,000	45,000 kms	Petrol
1	Mahindra Jeep CL550 MDI	Mahindra	2006	4,25,000	40 kms	Diesel
2	Maruti Suzuki Alto 800 Vxi	Maruti	2018	Ask For Price	22,000 kms	Petrol
3	Hyundai Grand i10 Magna 1.2 Kappa VTVT	Hyundai	2014	3,25,000	28,000 kms	Petrol
4	Ford EcoSport Titanium 1.5L TDCi	Ford	2014	5,75,000	36,000 kms	Diesel

In [43]:
```
car.shape
```

Out[43]:
```
(892, 6)
```

In [44]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        892 non-null    object
 1   company     892 non-null    object
 2   year        892 non-null    object
 3   Price       892 non-null    object
 4   kms_driven  840 non-null    object
 5   fuel_type   837 non-null    object
dtypes: object(6)
memory usage: 41.9+ KB
```

In [45]:
```
car['year'].unique()
```

Out[45]:
```
array(['2007', '2006', '2018', '2014', '2015', '2012', '2013', '2016',
       '2010', '2017', '2008', '2011', '2019', '2009', '2005', '2000',
       '...', '150k', 'TOUR', '2003', 'r 15', '2004', 'Zest', '/-Rs',
       'sale', '1995', 'ara)', '2002', 'SELL', '2001', 'tion', 'odel',
       '2 bs', 'arry', 'Eon', 'o...', 'ture', 'emi', 'car', 'able', 'no.',
       'd...', 'SALE', 'digo', 'sell', 'd Ex', 'n...', 'e...', 'D...',
       ', Ac', 'go .', 'k...', 'o c4', 'zire', 'cent', 'Sumo', 'cab',
       't xe', 'EV2', 'r...', 'zest'], dtype=object)
```

In [46]:
```
car['Price'].unique()
```

Out[46]:
```
array(['80,000', '4,25,000', 'Ask For Price', '3,25,000', '5,75,000',
       '1,75,000', '1,90,000', '8,30,000', '2,50,000', '1,82,000',
       '3,15,000', '4,15,000', '3,20,000', '10,00,000', '5,00,000',
       '3,50,000', '1,60,000', '3,10,000', '75,000', '1,00,000',
       '2,90,000', '95,000', '1,80,000', '3,85,000', '1,05,000',
       '6,50,000', '6,89,999', '4,48,000', '5,49,000', '5,01,000',
       '4,89,999', '2,80,000', '3,49,999', '2,84,999', '3,45,000',
       '4,99,999', '2,35,000', '2,49,999', '14,75,000', '3,95,000',
       '2,20,000', '1,70,000', '85,000', '2,00,000', '5,70,000',
       '1,10,000', '4,48,999', '18,91,111', '1,59,500', '3,44,999',
       '4,49,999', '8,65,000', '6,99,000', '3,75,000', '2,24,999',
       '12,00,000', '1,95,000', '3,51,000', '2,40,000', '90,000',
       '1,55,000', '6,00,000', '1,89,500', '2,10,000', '3,90,000',
       '1,35,000', '16,00,000', '7,01,000', '2,65,000', '5,25,000',
       '3,72,000', '6,35,000', '5,50,000', '4,85,000', '3,29,500',
       '2,51,111', '5,69,999', '69,999', '2,99,999', '3,99,999',
       '4,50,000', '2,70,000', '1,58,400', '1,79,000', '1,25,000',
       '2,99,000', '1,50,000', '2,75,000', '2,85,000', '3,40,000',
       '70,000', '2,89,999', '8,49,999', '7,49,999', '2,74,999',
       '9,84,999', '5,99,999', '2,44,999', '4,74,999', '2,45,000',
       '1,69,500', '3,70,000', '1,68,000', '1,45,000', '98,500',
       '2,09,000', '1,85,000', '9,00,000', '6,99,999', '1,99,999',
       '5,44,999', '1,99,000', '5,40,000', '49,000', '7,00,000', '55,000',
       '8,95,000', '3,55,000', '5,65,000', '3,65,000', '40,000',
       '4,00,000', '3,30,000', '5,80,000', '3,79,000', '2,19,000',
       '5,19,000', '7,30,000', '20,00,000', '21,00,000', '14,00,000',
       '3,11,000', '8,55,000', '5,35,000', '1,78,000', '3,00,000',
       '2,55,000', '5,49,999', '3,80,000', '57,000', '4,10,000',
       '2,25,000', '1,20,000', '59,000', '5,99,000', '6,75,000', '72,500',
       '6,10,000', '2,30,000', '5,20,000', '5,24,999', '4,24,999',
       '6,44,999', '5,84,999', '7,99,999', '4,44,999', '6,49,999',
       '9,44,999', '5,74,999', '3,74,999', '1,30,000', '4,01,000',
       '13,50,000', '1,74,999', '2,39,999', '99,999', '3,24,999',
       '10,74,999', '11,30,000', '1,49,000', '7,70,000', '30,000',
       '3,35,000', '3,99,000', '65,000', '1,69,999', '1,65,000',
       '5,60,000', '9,50,000', '7,15,000', '45,000', '9,40,000',
       '1,55,555', '15,00,000', '4,95,000', '8,00,000', '12,99,000',
       '5,30,000', '14,99,000', '32,000', '4,05,000', '7,60,000',
       '7,50,000', '4,19,000', '1,40,000', '15,40,000', '1,23,000',
       '4,98,000', '4,80,000', '4,88,000', '15,25,000', '5,48,900',
       '7,25,000', '99,000', '52,000', '28,00,000', '4,99,000',
       '3,81,000', '2,78,000', '6,90,000', '2,60,000', '90,001',
       '1,15,000', '15,99,000', '1,59,000', '51,999', '2,15,000',
       '35,000', '11,50,000', '2,69,000', '60,000', '4,30,000',
       '85,00,003', '4,01,919', '4,90,000', '4,24,000', '2,05,000',
       '5,49,900', '3,71,500', '4,35,000', '1,89,700', '3,89,700',
       '3,60,000', '2,95,000', '1,14,990', '10,65,000', '4,70,000',
       '48,000', '1,88,000', '4,65,000', '1,79,999', '21,90,000',
       '23,90,000', '10,75,000', '4,75,000', '10,25,000', '6,15,000',
       '19,00,000', '14,90,000', '15,10,000', '18,50,000', '7,90,000',
       '17,25,000', '12,25,000', '68,000', '9,70,000', '31,00,000',
       '8,99,000', '88,000', '53,000', '5,68,500', '71,000', '5,90,000',
       '7,95,000', '42,000', '1,89,000', '1,62,000', '35,999',
       '29,00,000', '39,999', '50,500', '5,10,000', '8,60,000',
       '5,00,001'], dtype=object)
```

In [47]:
```
car['kms_driven'].unique()
```

Out[47]:
```
array(['45,000 kms', '40 kms', '22,000 kms', '28,000 kms', '36,000 kms',
       '59,000 kms', '41,000 kms', '25,000 kms', '24,530 kms',
       '60,000 kms', '30,000 kms', '32,000 kms', '48,660 kms',
       '4,000 kms', '16,934 kms', '43,000 kms', '35,550 kms',
       '39,522 kms', '39,000 kms', '55,000 kms', '72,000 kms',
       '15,975 kms', '70,000 kms', '23,452 kms', '35,522 kms',
       '48,508 kms', '15,487 kms', '82,000 kms', '20,000 kms',
       '68,000 kms', '38,000 kms', '27,000 kms', '33,000 kms',
       '46,000 kms', '16,000 kms', '47,000 kms', '35,000 kms',
       '30,874 kms', '15,000 kms', '29,685 kms', '1,30,000 kms',
       '19,000 kms', nan, '54,000 kms', '13,000 kms', '38,200 kms',
       '50,000 kms', '13,500 kms', '3,600 kms', '45,863 kms',
       '60,500 kms', '12,500 kms', '18,000 kms', '13,349 kms',
       '29,000 kms', '44,000 kms', '42,000 kms', '14,000 kms',
       '49,000 kms', '36,200 kms', '51,000 kms', '1,04,000 kms',
       '33,333 kms', '33,600 kms', '5,600 kms', '7,500 kms', '26,000 kms',
       '24,330 kms', '65,480 kms', '28,028 kms', '2,00,000 kms',
       '99,000 kms', '2,800 kms', '21,000 kms', '11,000 kms',
       '66,000 kms', '3,000 kms', '7,000 kms', '38,500 kms', '37,200 kms',
       '43,200 kms', '24,800 kms', '45,872 kms', '40,000 kms',
       '11,400 kms', '97,200 kms', '52,000 kms', '31,000 kms',
       '1,75,430 kms', '37,000 kms', '65,000 kms', '3,350 kms',
       '75,000 kms', '62,000 kms', '73,000 kms', '2,200 kms',
       '54,870 kms', '34,580 kms', '97,000 kms', '60 kms', '80,200 kms',
       '3,200 kms', '0,000 kms', '5,000 kms', '588 kms', '71,200 kms',
       '1,75,400 kms', '9,300 kms', '56,758 kms', '10,000 kms',
       '56,450 kms', '56,000 kms', '32,700 kms', '9,000 kms', '73 kms',
       '1,60,000 kms', '84,000 kms', '58,559 kms', '57,000 kms',
       '1,70,000 kms', '80,000 kms', '6,821 kms', '23,000 kms',
       '34,000 kms', '1,800 kms', '4,00,000 kms', '48,000 kms',
       '90,000 kms', '12,000 kms', '69,900 kms', '1,66,000 kms',
       '122 kms', '0 kms', '24,000 kms', '36,469 kms', '7,800 kms',
       '24,695 kms', '15,141 kms', '59,910 kms', '1,00,000 kms',
       '4,500 kms', '1,29,000 kms', '300 kms', '1,31,000 kms',
       '1,11,111 kms', '59,466 kms', '25,500 kms', '44,005 kms',
       '2,110 kms', '43,222 kms', '1,00,200 kms', '65 kms',
       '1,40,000 kms', '1,03,553 kms', '58,000 kms', '1,20,000 kms',
       '49,800 kms', '100 kms', '81,876 kms', '6,020 kms', '55,700 kms',
       '18,500 kms', '1,80,000 kms', '53,000 kms', '35,500 kms',
       '22,134 kms', '1,000 kms', '8,500 kms', '87,000 kms', '6,000 kms',
       '15,574 kms', '8,000 kms', '55,800 kms', '56,400 kms',
       '72,160 kms', '11,500 kms', '1,33,000 kms', '2,000 kms',
       '88,000 kms', '65,422 kms', '1,17,000 kms', '1,50,000 kms',
       '10,750 kms', '6,800 kms', '5 kms', '9,800 kms', '57,923 kms',
       '30,201 kms', '6,200 kms', '37,518 kms', '24,652 kms', '383 kms',
       '95,000 kms', '3,528 kms', '52,500 kms', '47,900 kms',
       '52,800 kms', '1,95,000 kms', '48,008 kms', '48,247 kms',
       '9,400 kms', '64,000 kms', '2,137 kms', '10,544 kms', '49,500 kms',
       '1,47,000 kms', '90,001 kms', '48,006 kms', '74,000 kms',
       '85,000 kms', '29,500 kms', '39,700 kms', '67,000 kms',
       '19,336 kms', '60,105 kms', '45,933 kms', '1,02,563 kms',
       '28,600 kms', '41,800 kms', '1,16,000 kms', '42,590 kms',
       '7,400 kms', '54,500 kms', '76,000 kms', '00 kms', '11,523 kms',
       '38,600 kms', '95,500 kms', '37,458 kms', '85,960 kms',
       '12,516 kms', '30,600 kms', '2,550 kms', '62,500 kms',
       '69,000 kms', '28,400 kms', '68,485 kms', '3,500 kms',
       '85,455 kms', '63,000 kms', '1,600 kms', '77,000 kms',
       '26,500 kms', '2,875 kms', '13,900 kms', '1,500 kms', '2,450 kms',
       '1,625 kms', '33,400 kms', '60,123 kms', '38,900 kms',
       '1,37,495 kms', '91,200 kms', '1,46,000 kms', '1,00,800 kms',
       '2,100 kms', '2,500 kms', '1,32,000 kms', 'Petrol'], dtype=object)
```

In [48]:
```
car['fuel_type'].unique()
```

Out[48]:
```
array(['Petrol', 'Diesel', nan, 'LPG'], dtype=object)
```

In [49]:
```
car['name'].unique()
```

Out[49]:
```
array(['Hyundai Santro Xing XO eRLX Euro III', 'Mahindra Jeep CL550 MDI',
       'Maruti Suzuki Alto 800 Vxi',
       'Hyundai Grand i10 Magna 1.2 Kappa VTVT',
       'Ford EcoSport Titanium 1.5L TDCi', 'Ford Figo', 'Hyundai Eon',
       'Ford EcoSport Ambiente 1.5L TDCi',
       'Maruti Suzuki Alto K10 VXi AMT', 'Skoda Fabia Classic 1.2 MPI',
       'Maruti Suzuki Stingray VXi', 'Hyundai Elite i20 Magna 1.2',
       'Mahindra Scorpio SLE BS IV', 'Audi A8', 'Audi Q7',
       'Mahindra Scorpio S10', 'Maruti Suzuki Alto 800',
       'Hyundai i20 Sportz 1.2', 'Maruti Suzuki Alto 800 Lx',
       'Maruti Suzuki Vitara Brezza ZDi', 'Maruti Suzuki Alto LX',
       'Mahindra Bolero DI', 'Maruti Suzuki Swift Dzire ZDi',
       'Mahindra Scorpio S10 4WD', 'Maruti Suzuki Swift Vdi BSIII',
       'Maruti Suzuki Wagon R VXi BS III',
       'Maruti Suzuki Wagon R VXi Minor',
       'Toyota Innova 2.0 G 8 STR BS IV', 'Renault Lodgy 85 PS RXL',
       'Skoda Yeti Ambition 2.0 TDI CR 4x2',
       'Maruti Suzuki Baleno Delta 1.2',
       'Renault Duster 110 PS RxZ Diesel Plus',
```

In [50]:
```
car['company'].unique()
```

Out[50]:
```
array(['Hyundai', 'Mahindra', 'Maruti', 'Ford', 'Skoda', 'Audi', 'Toyota',
       'Renault', 'Honda', 'Datsun', 'Mitsubishi', 'Tata', 'Volkswagen',
       'I', 'Chevrolet', 'Mini', 'BMW', 'Nissan', 'Hindustan', 'Fiat',
       'Commercial', 'MARUTI', 'Force', 'Mercedes', 'Land', 'Yamaha',
       'selling', 'URJENT', 'Swift', 'Used', 'Jaguar', 'Jeep', 'tata',
       'Sale', 'very', 'Volvo', 'i', '2012', 'Well', 'all', '7', '9',
       'scratch', 'urgent', 'sell', 'TATA', 'Any', 'Tara'], dtype=object)
```

Creating backup copy

In [51]:
```
backup=car.copy()
```

Quality

- names are pretty inconsistent
- names have company names attached to it
- some names are spam like 'Maruti Ertiga showroom condition with' and 'Well mentained Tata Sumo'
- company: many of the names are not of any company like 'Used', 'URJENT', and so on.
- year has many non-year values
- year is in object. Change to integer
- Price has Ask for Price
- Price has commas in its prices and is in object
- kms_driven has object values with kms at last.
- It has nan values and two rows have 'Petrol' in them
- fuel_type has nan values

Cleaning Data

year has many non-year values

In [52]:
```
car['year'].unique()
```

Out[52]:
```
array(['2007', '2006', '2018', '2014', '2015', '2012', '2013', '2016',
       '2010', '2017', '2008', '2011', '2019', '2009', '2005', '2000',
       '...', '150k', 'TOUR', '2003', 'r 15', '2004', 'Zest', '/-Rs',
       'sale', '1995', 'ara)', '2002', 'SELL', '2001', 'tion', 'odel',
       '2 bs', 'arry', 'Eon', 'o...', 'ture', 'emi', 'car', 'able', 'no.',
       'd...', 'SALE', 'digo', 'sell', 'd Ex', 'n...', 'e...', 'D...',
       ', Ac', 'go .', 'k...', 'o c4', 'zire', 'cent', 'Sumo', 'cab',
       't xe', 'EV2', 'r...', 'zest'], dtype=object)
```

In [53]:
```
car=car[car['year'].str.isnumeric()]
car['year'].unique()
```

Out[53]:
```
array(['2007', '2006', '2018', '2014', '2015', '2012', '2013', '2016',
       '2010', '2017', '2008', '2011', '2019', '2009', '2005', '2000',
       '2003', '2004', '1995', '2002', '2001'], dtype=object)
```

In [ ]:
```

```

year is in object. Change to integer

In [54]:
```
car['year']=car['year'].astype(int)
```

In [55]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 842 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        842 non-null    object
 1   company     842 non-null    object
 2   year        842 non-null    int32 
 3   Price       842 non-null    object
 4   kms_driven  840 non-null    object
 5   fuel_type   837 non-null    object
dtypes: int32(1), object(5)
memory usage: 42.8+ KB
```

Price has Ask for Price

In [56]:
```
car['Price'].unique()
```

Out[56]:
```
array(['80,000', '4,25,000', 'Ask For Price', '3,25,000', '5,75,000',
       '1,75,000', '1,90,000', '8,30,000', '2,50,000', '1,82,000',
       '3,15,000', '4,15,000', '3,20,000', '10,00,000', '5,00,000',
       '3,50,000', '1,60,000', '3,10,000', '75,000', '1,00,000',
       '2,90,000', '95,000', '1,80,000', '3,85,000', '1,05,000',
       '6,50,000', '6,89,999', '4,48,000', '5,49,000', '5,01,000',
       '4,89,999', '2,80,000', '3,49,999', '2,84,999', '3,45,000',
       '4,99,999', '2,35,000', '2,49,999', '14,75,000', '3,95,000',
       '2,20,000', '1,70,000', '85,000', '2,00,000', '5,70,000',
       '1,10,000', '4,48,999', '18,91,111', '1,59,500', '3,44,999',
       '4,49,999', '8,65,000', '6,99,000', '3,75,000', '2,24,999',
       '12,00,000', '1,95,000', '3,51,000', '2,40,000', '90,000',
       '1,55,000', '6,00,000', '1,89,500', '2,10,000', '3,90,000',
       '1,35,000', '16,00,000', '7,01,000', '2,65,000', '5,25,000',
       '3,72,000', '6,35,000', '5,50,000', '4,85,000', '3,29,500',
       '2,51,111', '5,69,999', '69,999', '2,99,999', '3,99,999',
       '4,50,000', '2,70,000', '1,58,400', '1,79,000', '1,25,000',
       '2,99,000', '1,50,000', '2,75,000', '2,85,000', '3,40,000',
       '70,000', '2,89,999', '8,49,999', '7,49,999', '2,74,999',
       '9,84,999', '5,99,999', '2,44,999', '4,74,999', '2,45,000',
       '1,69,500', '3,70,000', '1,68,000', '1,45,000', '98,500',
       '2,09,000', '1,85,000', '9,00,000', '6,99,999', '1,99,999',
       '5,44,999', '1,99,000', '5,40,000', '49,000', '7,00,000', '55,000',
       '8,95,000', '3,55,000', '5,65,000', '3,65,000', '40,000',
       '4,00,000', '3,30,000', '5,80,000', '3,79,000', '2,19,000',
       '5,19,000', '7,30,000', '20,00,000', '21,00,000', '14,00,000',
       '3,11,000', '8,55,000', '5,35,000', '1,78,000', '3,00,000',
       '2,55,000', '5,49,999', '3,80,000', '57,000', '4,10,000',
       '2,25,000', '1,20,000', '59,000', '5,99,000', '6,75,000', '72,500',
       '6,10,000', '2,30,000', '5,20,000', '5,24,999', '4,24,999',
       '6,44,999', '5,84,999', '7,99,999', '4,44,999', '6,49,999',
       '9,44,999', '5,74,999', '3,74,999', '1,30,000', '4,01,000',
       '13,50,000', '1,74,999', '2,39,999', '99,999', '3,24,999',
       '10,74,999', '11,30,000', '1,49,000', '7,70,000', '30,000',
       '3,35,000', '3,99,000', '65,000', '1,69,999', '1,65,000',
       '5,60,000', '9,50,000', '7,15,000', '45,000', '9,40,000',
       '1,55,555', '15,00,000', '4,95,000', '8,00,000', '12,99,000',
       '5,30,000', '14,99,000', '32,000', '4,05,000', '7,60,000',
       '7,50,000', '4,19,000', '1,40,000', '15,40,000', '1,23,000',
       '4,98,000', '4,80,000', '4,88,000', '15,25,000', '5,48,900',
       '7,25,000', '99,000', '52,000', '28,00,000', '4,99,000',
       '3,81,000', '2,78,000', '6,90,000', '2,60,000', '90,001',
       '1,15,000', '15,99,000', '1,59,000', '51,999', '2,15,000',
       '35,000', '11,50,000', '2,69,000', '60,000', '4,30,000',
       '85,00,003', '4,01,919', '4,90,000', '4,24,000', '2,05,000',
       '5,49,900', '4,35,000', '1,89,700', '3,89,700', '3,60,000',
       '2,95,000', '1,14,990', '10,65,000', '4,70,000', '48,000',
       '1,88,000', '4,65,000', '1,79,999', '21,90,000', '23,90,000',
       '10,75,000', '4,75,000', '10,25,000', '6,15,000', '19,00,000',
       '14,90,000', '15,10,000', '18,50,000', '7,90,000', '17,25,000',
       '12,25,000', '68,000', '9,70,000', '31,00,000', '8,99,000',
       '88,000', '53,000', '5,68,500', '71,000', '5,90,000', '7,95,000',
       '42,000', '1,89,000', '1,62,000', '35,999', '29,00,000', '39,999',
       '50,500', '5,10,000', '8,60,000', '5,00,001'], dtype=object)
```

In [57]:
```
car=car[car['Price']!='Ask For Price']
```

Price has commas in its prices and is in object

In [58]:
```
car['Price'].unique()
```

Out[58]:
```
array(['80,000', '4,25,000', '3,25,000', '5,75,000', '1,75,000',
       '1,90,000', '8,30,000', '2,50,000', '1,82,000', '3,15,000',
       '4,15,000', '3,20,000', '10,00,000', '5,00,000', '3,50,000',
       '1,60,000', '3,10,000', '75,000', '1,00,000', '2,90,000', '95,000',
       '1,80,000', '3,85,000', '1,05,000', '6,50,000', '6,89,999',
       '4,48,000', '5,49,000', '5,01,000', '4,89,999', '2,80,000',
       '3,49,999', '2,84,999', '3,45,000', '4,99,999', '2,35,000',
       '2,49,999', '14,75,000', '3,95,000', '2,20,000', '1,70,000',
       '85,000', '2,00,000', '5,70,000', '1,10,000', '4,48,999',
       '18,91,111', '1,59,500', '3,44,999', '4,49,999', '8,65,000',
       '6,99,000', '3,75,000', '2,24,999', '12,00,000', '1,95,000',
       '3,51,000', '2,40,000', '90,000', '1,55,000', '6,00,000',
       '1,89,500', '2,10,000', '3,90,000', '1,35,000', '16,00,000',
       '7,01,000', '2,65,000', '5,25,000', '3,72,000', '6,35,000',
       '5,50,000', '4,85,000', '3,29,500', '2,51,111', '5,69,999',
       '69,999', '2,99,999', '3,99,999', '4,50,000', '2,70,000',
       '1,58,400', '1,79,000', '1,25,000', '2,99,000', '1,50,000',
       '2,75,000', '2,85,000', '3,40,000', '70,000', '2,89,999',
       '8,49,999', '7,49,999', '2,74,999', '9,84,999', '5,99,999',
       '2,44,999', '4,74,999', '2,45,000', '1,69,500', '3,70,000',
       '1,68,000', '1,45,000', '98,500', '2,09,000', '1,85,000',
       '9,00,000', '6,99,999', '1,99,999', '5,44,999', '1,99,000',
       '5,40,000', '49,000', '7,00,000', '55,000', '8,95,000', '3,55,000',
       '5,65,000', '3,65,000', '40,000', '4,00,000', '3,30,000',
       '5,80,000', '3,79,000', '2,19,000', '5,19,000', '7,30,000',
       '20,00,000', '21,00,000', '14,00,000', '3,11,000', '8,55,000',
       '5,35,000', '1,78,000', '3,00,000', '2,55,000', '5,49,999',
       '3,80,000', '57,000', '4,10,000', '2,25,000', '1,20,000', '59,000',
       '5,99,000', '6,75,000', '72,500', '6,10,000', '2,30,000',
       '5,20,000', '5,24,999', '4,24,999', '6,44,999', '5,84,999',
       '7,99,999', '4,44,999', '6,49,999', '9,44,999', '5,74,999',
       '3,74,999', '1,30,000', '4,01,000', '13,50,000', '1,74,999',
       '2,39,999', '99,999', '3,24,999', '10,74,999', '11,30,000',
       '1,49,000', '7,70,000', '30,000', '3,35,000', '3,99,000', '65,000',
       '1,69,999', '1,65,000', '5,60,000', '9,50,000', '7,15,000',
       '45,000', '9,40,000', '1,55,555', '15,00,000', '4,95,000',
       '8,00,000', '12,99,000', '5,30,000', '14,99,000', '32,000',
       '4,05,000', '7,60,000', '7,50,000', '4,19,000', '1,40,000',
       '15,40,000', '1,23,000', '4,98,000', '4,80,000', '4,88,000',
       '15,25,000', '5,48,900', '7,25,000', '99,000', '52,000',
       '28,00,000', '4,99,000', '3,81,000', '2,78,000', '6,90,000',
       '2,60,000', '90,001', '1,15,000', '15,99,000', '1,59,000',
       '51,999', '2,15,000', '35,000', '11,50,000', '2,69,000', '60,000',
       '4,30,000', '85,00,003', '4,01,919', '4,90,000', '4,24,000',
       '2,05,000', '5,49,900', '4,35,000', '1,89,700', '3,89,700',
       '3,60,000', '2,95,000', '1,14,990', '10,65,000', '4,70,000',
       '48,000', '1,88,000', '4,65,000', '1,79,999', '21,90,000',
       '23,90,000', '10,75,000', '4,75,000', '10,25,000', '6,15,000',
       '19,00,000', '14,90,000', '15,10,000', '18,50,000', '7,90,000',
       '17,25,000', '12,25,000', '68,000', '9,70,000', '31,00,000',
       '8,99,000', '88,000', '53,000', '5,68,500', '71,000', '5,90,000',
       '7,95,000', '42,000', '1,89,000', '1,62,000', '35,999',
       '29,00,000', '39,999', '50,500', '5,10,000', '8,60,000',
       '5,00,001'], dtype=object)
```

In [59]:
```
car['Price']=car['Price'].str.replace(',', '')
```

In [60]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 819 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        819 non-null    object
 1   company     819 non-null    object
 2   year        819 non-null    int32 
 3   Price       819 non-null    object
 4   kms_driven  819 non-null    object
 5   fuel_type   816 non-null    object
dtypes: int32(1), object(5)
memory usage: 41.6+ KB
```

In [61]:
```
car['Price']=car['Price'].astype(int)
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 819 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        819 non-null    object
 1   company     819 non-null    object
 2   year        819 non-null    int32 
 3   Price       819 non-null    int32 
 4   kms_driven  819 non-null    object
 5   fuel_type   816 non-null    object
dtypes: int32(2), object(4)
memory usage: 38.4+ KB
```

kms_driven has object values with kms at last.

In [62]:
```
car['kms_driven']=car['kms_driven'].str.replace(',','')
print(car['kms_driven'])
```

```
0       45000 kms
1          40 kms
3       28000 kms
4       36000 kms
6       41000 kms
          ...
886    132000 kms
888     27000 kms
889     40000 kms
890        Petrol
891        Petrol
Name: kms_driven, Length: 819, dtype: object
```

In [63]:
```
car['kms_driven']=car['kms_driven'].str.split().str.get(0)
```

In [64]:
```
print(car['kms_driven'])
```

```
0       45000
1          40
3       28000
4       36000
6       41000
        ...
886    132000
888     27000
889     40000
890    Petrol
891    Petrol
Name: kms_driven, Length: 819, dtype: object
```

It has nan values and two rows have 'Petrol' in them

In [65]:
```
car=car[car['kms_driven'].str.isnumeric()]
```

In [66]:
```
car['kms_driven']=car['kms_driven'].astype(int)
print(car['kms_driven'])
```

```
0       45000
1          40
3       28000
4       36000
6       41000
        ...
883     50000
885     30000
886    132000
888     27000
889     40000
Name: kms_driven, Length: 817, dtype: int32
```

fuel_type has nan values

In [67]:
```
car=car[~car['fuel_type'].isna()]
```

In [68]:
```
car.shape
```

Out[68]: (816, 6)

name and company had spammed data...but with the previous cleaning, those rows got removed.

Company does not need any cleaning now. Changing car names. Keeping only the first three words

In [69]:
```
car['name']=car['name'].str.split().str.slice(start=0,stop=3).str.join(' ')
```

Resetting the index of the final cleaned data

In [70]:
```
car=car.reset_index(drop=True)
```

Cleaned Data

In [71]:
```
car
```

Out[71]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	80000	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	425000	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	325000	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	575000	36000	Diesel
4	Ford Figo	Ford	2012	175000	41000	Diesel
...	...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	270000	50000	Petrol
812	Tata Indica V2	Tata	2009	110000	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	300000	132000	Petrol
814	Tata Zest XM	Tata	2018	260000	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	390000	40000	Diesel

816 rows × 6 columns

In [72]:
```
#exporting the cleaned data to new file
car.to_csv('Cleaned_Car_data.csv')
```

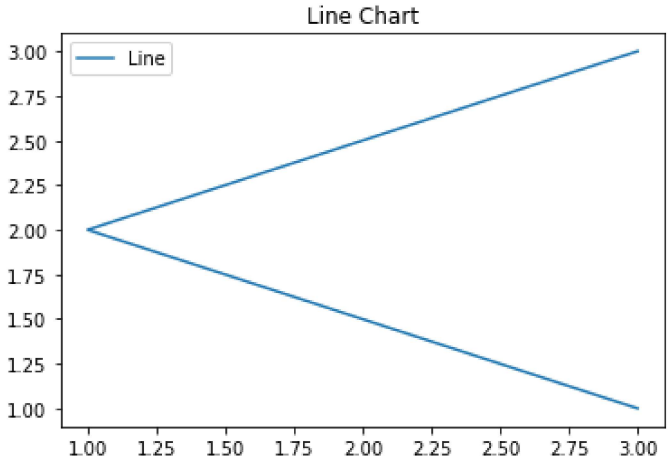
9. Write a program to illustrate matplotlib library. Draw Bar Graph, Line Graph, Scatter Graph, Hisogram.

Creating Different Types of Plots:

1. Line Graph

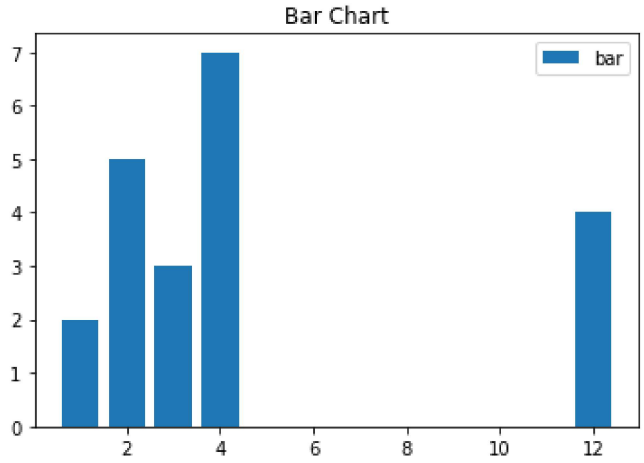
Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the pot() function.

In [1]:
```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3]
y = [3, 2, 1]
# This will plot a simple line chart
# with elements of x as x axis and y
# as y axis
plt.plot(x, y)
plt.title("Line Chart")
# Adding the legends
plt.legend(["Line"])
plt.show()
```




```
<h4>2. Bar chart</h4>
<p> 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. The bar plots can be plotted horizontally or vertically. A bar chart
describes the comparisons between the discrete categories. It can be created using the
bar() method.</p>
```

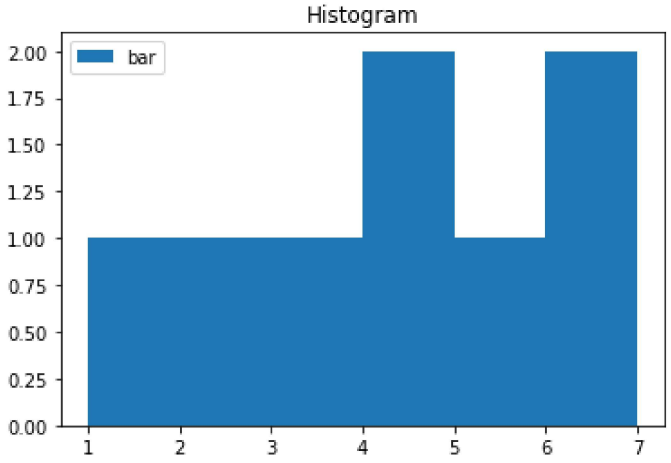
In [2]:
```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
# This will plot a simple bar chart
plt.bar(x, y)
# Title to the plot
plt.title("Bar Chart")
# Adding the legends
plt.legend(["bar"])
plt.show()
```



3. Histograms

A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. To create a histogram the first step is to create a bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The hist() function is used to compute and create histogram of x.

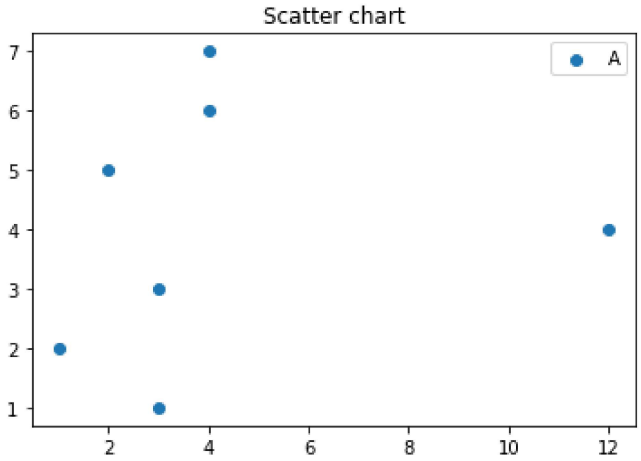
In [3]:
```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4, 5, 6, 7, 4]
# This will plot a simple histogram
plt.hist(x, bins = [1, 2, 3, 4, 5, 6, 7])
# Title to the plot
plt.title("Histogram")
# Adding the legends
plt.legend(["bar"])
plt.show()
```



Scatter Plot

Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The scatter() method in the matplotlib library is used to draw a scatter plot.

In [5]:
```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
# This will plot a simple scatter chart
plt.scatter(x, y)
# Adding legend to the plot
plt.legend("A")
# Title to the plot
plt.title("Scatter chart")
plt.show()
```



10. (A) Write a Program to implement Linear Regression using cleaned_car dataset.

In [34]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
mpl.style.use('ggplot')
```

In [3]:
```
car=pd.read_csv('Cleaned_Car_data.csv')
```

In [4]:
```
car.info()
```

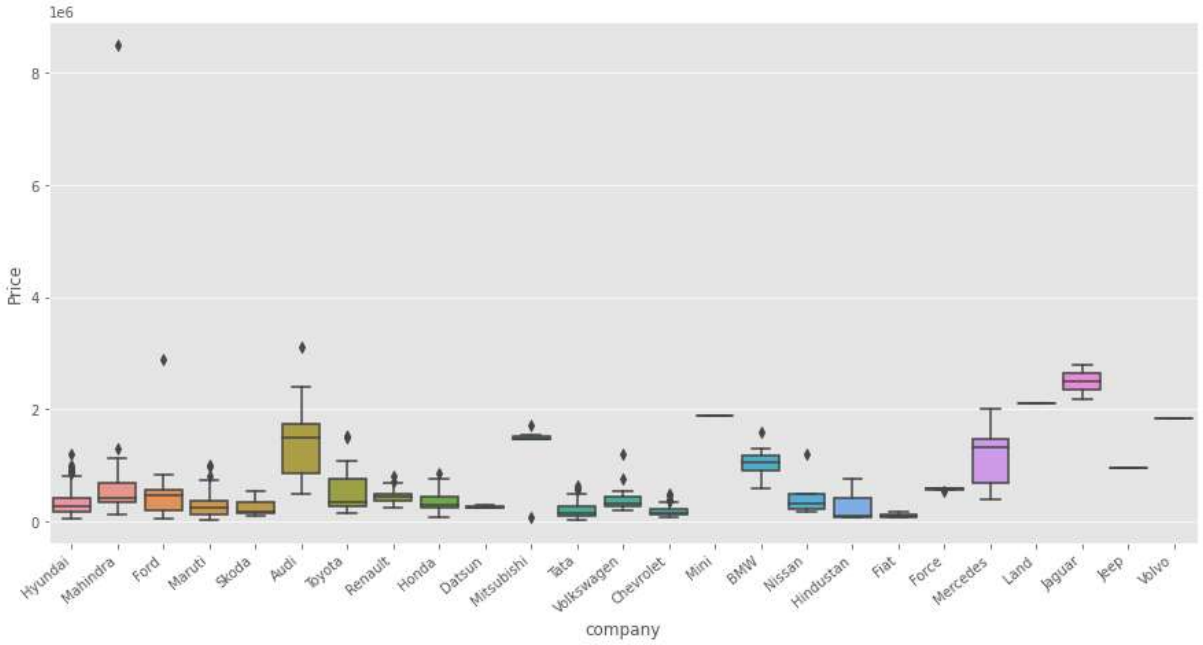
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 816 entries, 0 to 815
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  816 non-null    int64 
 1   name        816 non-null    object
 2   company     816 non-null    object
 3   year        816 non-null    int64 
 4   Price       816 non-null    int64 
 5   kms_driven  816 non-null    int64 
 6   fuel_type   816 non-null    object
dtypes: int64(4), object(3)
memory usage: 44.8+ KB
```

In [5]:
```
car.describe(include='all')
```

Out[5]:

	Unnamed: 0	name	company	year	Price	kms_driven	fuel_type
count	816.000000	816	816	816.000000	8.160000e+02	816.000000	816
unique	NaN	254	25	NaN	NaN	NaN	3
top	NaN	Maruti Suzuki Swift	Maruti	NaN	NaN	NaN	Petrol
freq	NaN	51	221	NaN	NaN	NaN	428
mean	407.500000	NaN	NaN	2012.444853	4.117176e+05	46275.531863	NaN
std	235.703203	NaN	NaN	4.002992	4.751844e+05	34297.428044	NaN
min	0.000000	NaN	NaN	1995.000000	3.000000e+04	0.000000	NaN
25%	203.750000	NaN	NaN	2010.000000	1.750000e+05	27000.000000	NaN
50%	407.500000	NaN	NaN	2013.000000	2.999990e+05	41000.000000	NaN
75%	611.250000	NaN	NaN	2015.000000	4.912500e+05	56818.500000	NaN
max	815.000000	NaN	NaN	2019.000000	8.500003e+06	400000.000000	NaN

In [6]:
```
import seaborn as sns
plt.subplots(figsize=(15,7))
ax=sns.boxplot(x='company',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```



In [7]:
```
#removing outliers
car[car['Price']>4000000]
```

Out[7]:

	Unnamed: 0	name	company	year	Price	kms_driven	fuel_type
534	534	Mahindra XUV500 W6	Mahindra	2014	8500003	45000	Diesel

In [8]:
```
car=car[car['Price']<4000000]
```

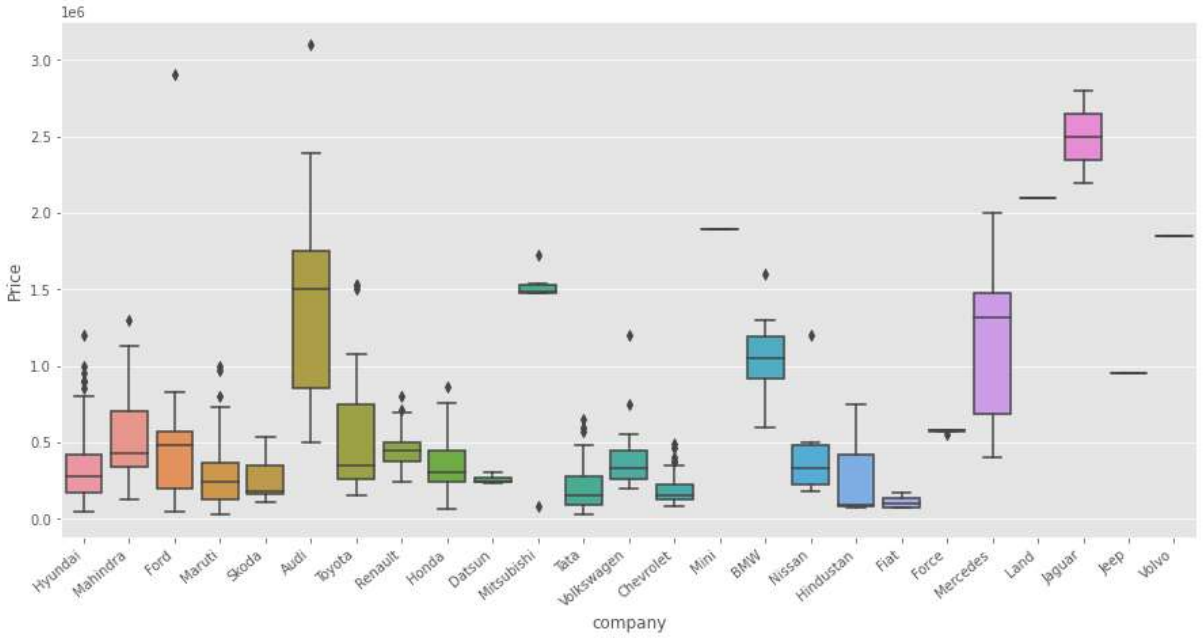
Checking relationship of Company with Price

In [9]:
```
car['company'].unique()
```

Out[9]:
```
array(['Hyundai', 'Mahindra', 'Ford', 'Maruti', 'Skoda', 'Audi', 'Toyota',
       'Renault', 'Honda', 'Datsun', 'Mitsubishi', 'Tata', 'Volkswagen',
       'Chevrolet', 'Mini', 'BMW', 'Nissan', 'Hindustan', 'Fiat', 'Force',
       'Mercedes', 'Land', 'Jaguar', 'Jeep', 'Volvo'], dtype=object)
```

In [10]:
```
import seaborn as sns
```

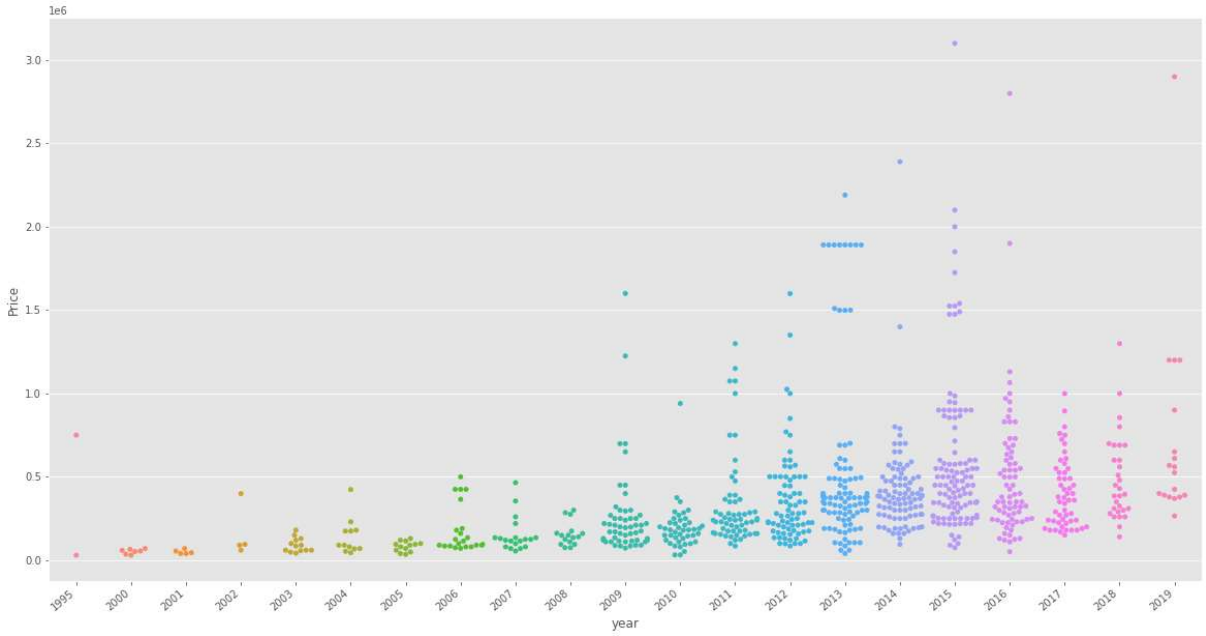
In [11]:
```
plt.subplots(figsize=(15,7))
ax=sns.boxplot(x='company',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```



Checking relationship of Year with Price

In [12]:
```
plt.subplots(figsize=(20,10))
ax=sns.swarmplot(x='year',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```

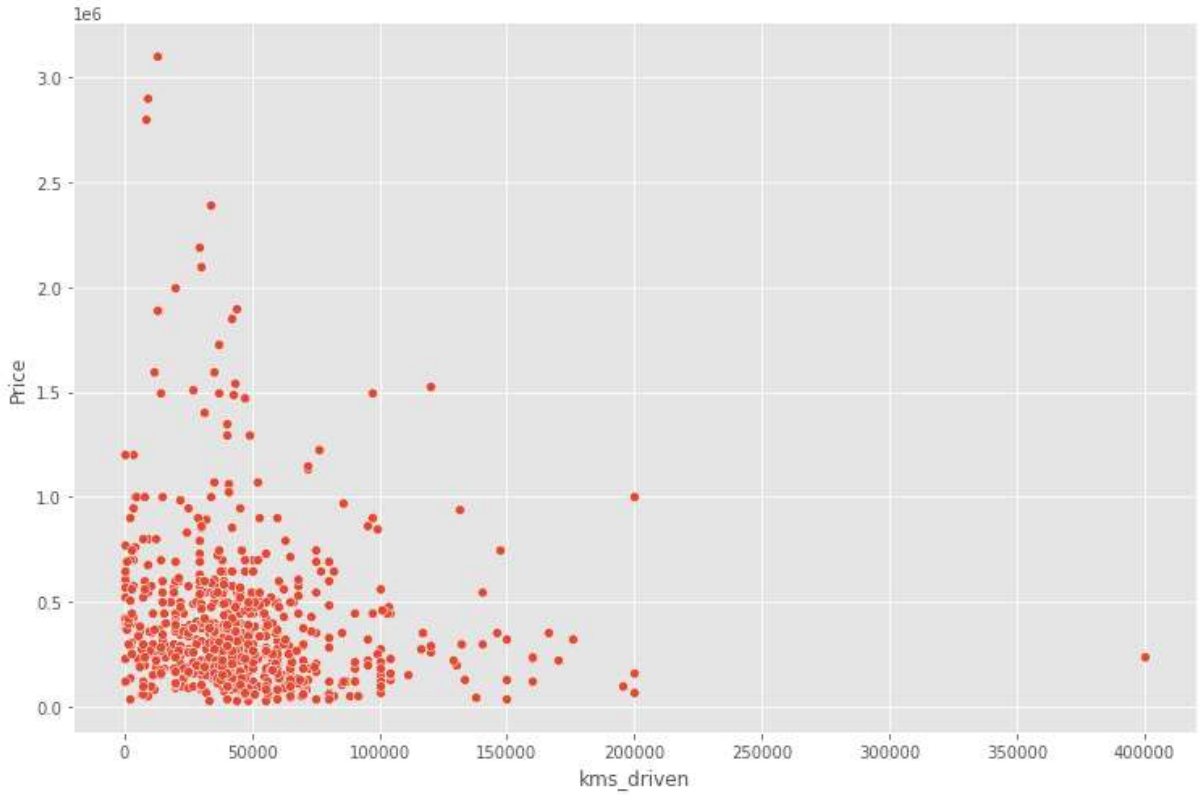
```
C:\Users\dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 9.
3% of the points cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 6.
8% of the points cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 10.
6% of the points cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 5.
5% of the points cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
```



Checking relationship of kms_driven with Price

In [13]:
```
sns.relplot(x='kms_driven',y='Price',data=car,height=7,aspect=1.5)
```

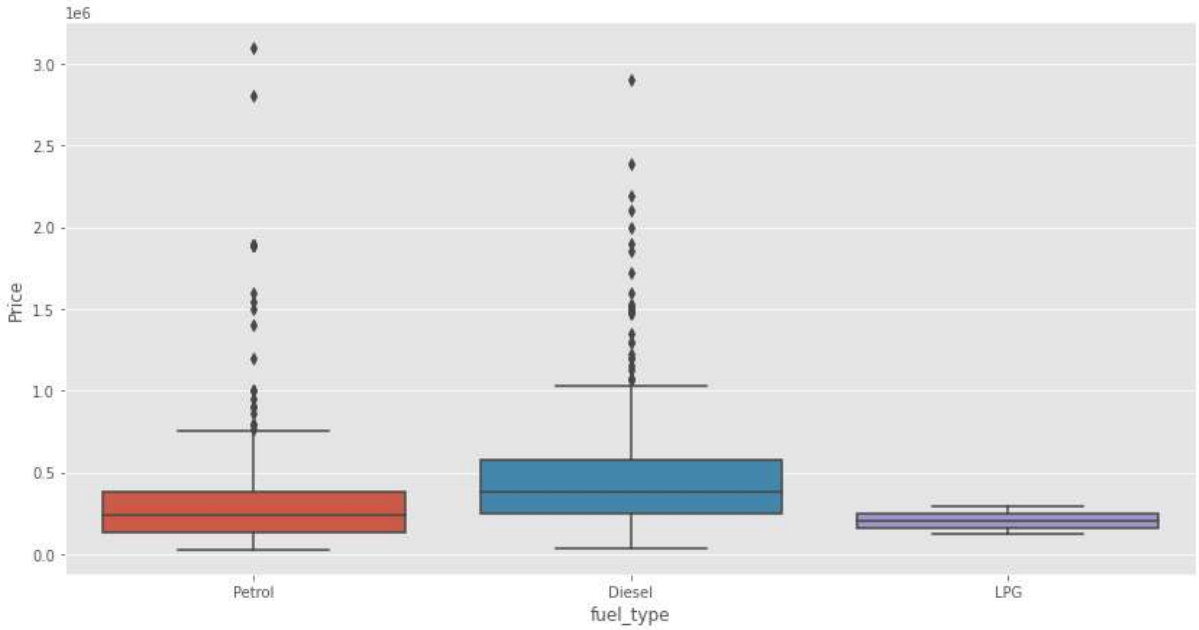
Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x18fc558f430>
```



Checking relationship of Fuel Type with Price

In [14]:
```
plt.subplots(figsize=(14,7))
sns.boxplot(x='fuel_type',y='Price',data=car)
```

Out[14]: <AxesSubplot:xlabel='fuel_type', ylabel='Price'>



Extracting Training Data

In [15]:
```
X=car[['name','company','year','kms_driven','fuel_type']]
y=car['Price']
```

In [16]:
```
X
```

Out[16]:

	name	company	year	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	36000	Diesel
4	Ford Figo	Ford	2012	41000	Diesel
...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	50000	Petrol
812	Tata Indica V2	Tata	2009	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	132000	Petrol
814	Tata Zest XM	Tata	2018	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	40000	Diesel

815 rows × 5 columns

In [17]:
```
y.shape
```

Out[17]: (815,)

Applying Train Test Split

In [18]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

In [19]:
```
from sklearn.linear_model import LinearRegression
```

In [20]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

Creating an OneHotEncoder object to contain all the possible categories

In [21]:
```
ohe=OneHotEncoder()
ohe.fit(X[['name','company','fuel_type']])
```

Out[21]:
```
OneHotEncoder()
```

Creating a column transformer to transform categorical columns

In [22]:
```
column_trans=make_column_transformer((OneHotEncoder(categories=ohe.categories_),['name',
                                     remainder='passthrough')
```

Linear Regression Model

In [23]:
```
lr=LinearRegression()
```

Making a pipeline

In [24]:
```
pipe=make_pipeline(column_trans,lr)
```

Fitting the model

In [25]:
```
pipe.fit(X_train,y_train)
```

Out[25]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories=[array(['Aud
i A3 Cabriolet', 'Audi A4 1.8', 'Audi A4 2.0', 'Audi A6 2.0',
       'Audi A8', 'Audi Q3 2.0', 'Audi Q5 2.0', 'Audi Q7', 'BMW 3 Series',
       'BMW 5 Series', 'BMW 7 Series', 'BMW X1', 'BMW X1 sDrive20d',
       'BMW X1 xDrive20d', 'Chevrolet Beat', 'Chevrolet Beat...
                                                                        array(['Aud
i', 'BMW', 'Chevrolet', 'Datsun', 'Fiat', 'Force', 'Ford',
       'Hindustan', 'Honda', 'Hyundai', 'Jaguar', 'Jeep', 'Land',
       'Mahindra', 'Maruti', 'Mercedes', 'Mini', 'Mitsubishi', 'Nissan',
       'Renault', 'Skoda', 'Tata', 'Toyota', 'Volkswagen', 'Volvo'],
      dtype=object),
                                                                        array(['Die
sel', 'LPG', 'Petrol'], dtype=object)]),
                                                  ['name', 'company',
                                                   'fuel_type'])])),
                ('linearregression', LinearRegression())])
```

In [26]:
```
y_pred=pipe.predict(X_test)
```

Checking R2 Score

In [27]:
```
r2_score(y_test,y_pred)
```

Out[27]:
```
0.5561093778833734
```

Finding the model with a random state of TrainTestSplit where the model was found to give almost 0.92 as r2_score

In [28]:
```
scores=[]
for i in range(10):
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=i)
    lr=LinearRegression()
    pipe=make_pipeline(column_trans,lr)
    pipe.fit(X_train,y_train)
    y_pred=pipe.predict(X_test)
    scores.append(r2_score(y_test,y_pred))
```

In [29]:
```
np.argmax(scores)
```

Out[29]:
```
4
```

In [30]:
```
scores[np.argmax(scores)]
```

Out[30]:
```
0.8781865811842144
```

In [31]:
```
pipe.predict(pd.DataFrame(columns=X_test.columns,data=np.array(['Maruti Suzuki Swift','M
```

Out[31]:
```
array([429517.34989336])
```

The best model is found at a certain random state

In [32]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=np.argmax(
lr=LinearRegression()
pipe=make_pipeline(column_trans,lr)
pipe.fit(X_train,y_train)
y_pred=pipe.predict(X_test)
r2_score(y_test,y_pred)
```

Out[32]:
```
0.8781865811842144
```

In [ ]:
```

```

10. B. Write a Program to implement Linear Regression using boston housing dataset.

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import seaborn as sns

%matplotlib inline
```

In [20]:
```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
```

In [21]:
```
print(boston_dataset.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [22]:
```
print(boston_dataset.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [23]:
```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head(5)
```

Out[23]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [24]:
```
boston['MEDV'] = boston_dataset.target
```

Data Pre-processing

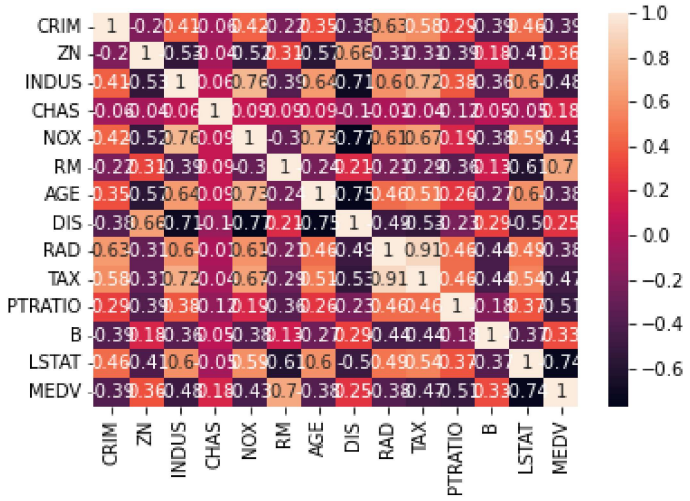
In [25]:
```
boston.isnull().sum()
```

Out[25]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Exploratory Data Analysis

In [26]:
```
correlation_matrix = boston.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[26]: <AxesSubplot:>



Preparing the data for training the model

In [27]:
```
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
Y = boston['MEDV']
```

In [28]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

Training and testing the model

In [29]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

Out[29]: LinearRegression()

Model evaluation

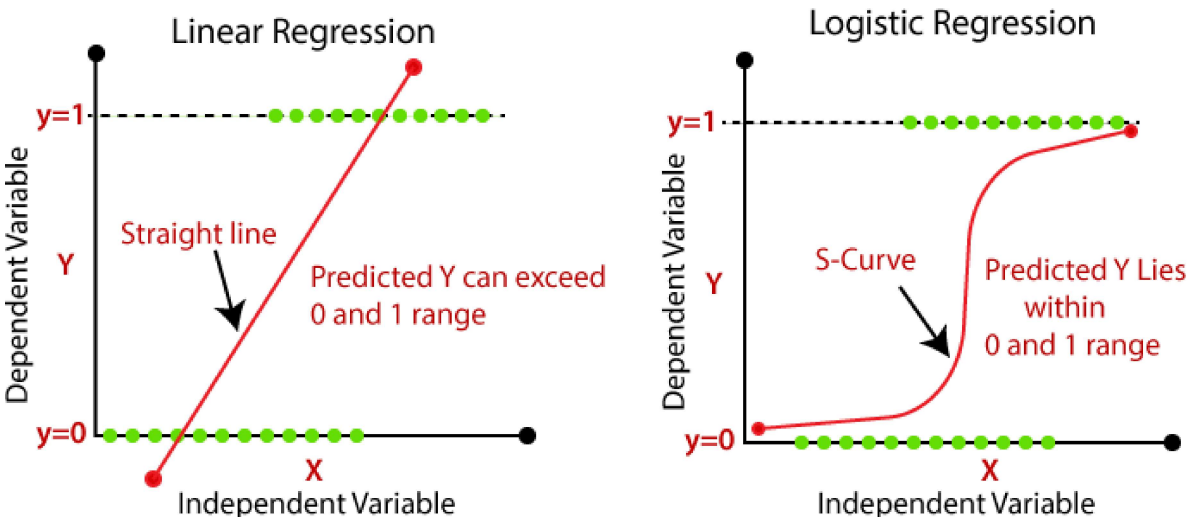
In [30]:
```
# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701


The model performance for testing set
--------------------------------------
RMSE is 5.13740078470291
R2 score is 0.6628996975186954
```

11. Write a Program to implement Logistic Regression using Admission_Predict dataset

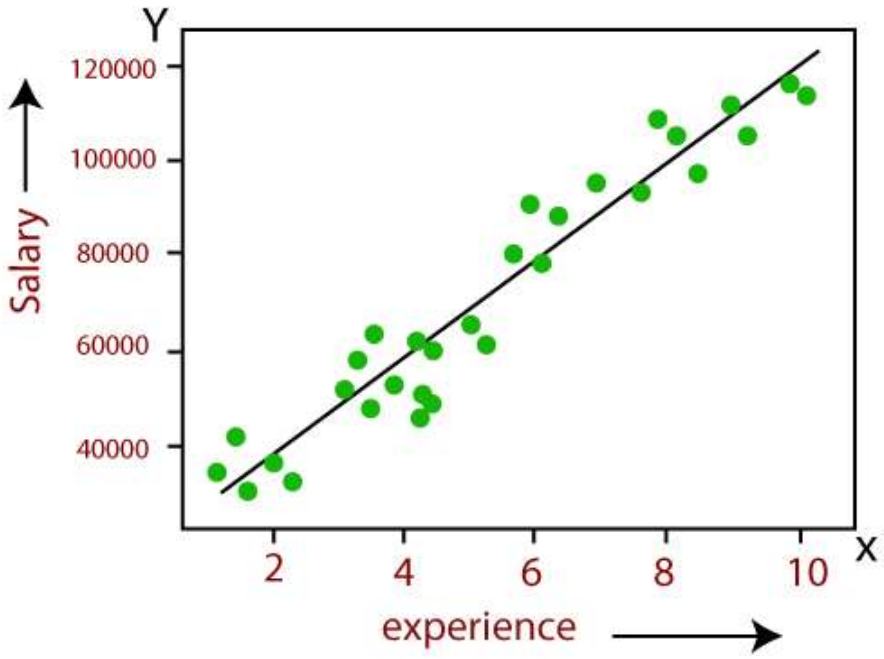
Linear Regression vs Logistic Regression

Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems. The description of both the algorithms is given below along with difference table.



Linear Regression:

- Linear Regression is one of the most simple Machine learning algorithm that comes under Supervised Learning technique and used for solving regression problems.
- It is used for predicting the continuous dependent variable with the help of independent variables.
- The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable.
- If single independent variable is used for prediction then it is called Simple Linear Regression and if there are more than two independent variables then such regression is called as Multiple Linear Regression.
- By finding the best fit line, algorithm establish the relationship between dependent variable and independent variable. And the relationship should be of linear nature.
- The output for Linear regression should only be the continuous values such as price, age, salary, etc. The relationship between the dependent variable and independent variable can be shown in below image:



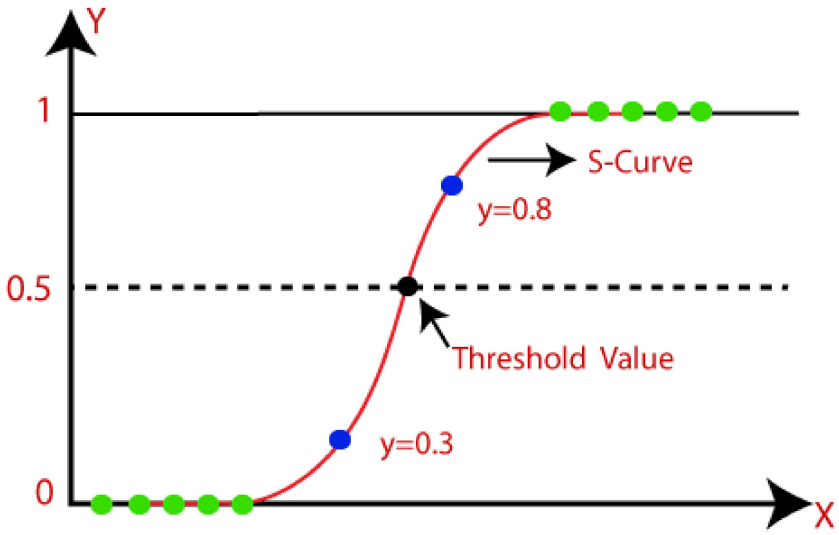
In above image the dependent variable is on Y-axis (salary) and independent variable is on x-axis(experience). The regression line can be written as:

$$y= a_0+a_1x+ \varepsilon$$

Where, a_0 and a_1 are the coefficients and ε is the error term.

Logistic Regression:

- Logistic regression is one of the most popular Machine learning algorithm that comes under Supervised Learning techniques.
- It can be used for Classification as well as for Regression problems, but mainly used for Classification problems.
- Logistic regression is used to predict the categorical dependent variable with the help of independent variables.
- The output of Logistic Regression problem can be only between the 0 and 1.
- Logistic regression can be used where the probabilities between two classes is required. Such as whether it will rain today or not, either 0 or 1, true or false etc.
- Logistic regression is based on the concept of Maximum Likelihood estimation. According to this estimation, the observed data should be most probable.
- In logistic regression, we pass the weighted sum of inputs through an activation function that can map values in between 0 and 1. Such activation function is known as **sigmoid function** and the curve obtained is called as sigmoid curve or S-curve. Consider the below image:



- The equation for logistic regression is:

$$\log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

Difference between Linear Regression and Logistic Regression:

Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.
Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line, by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear.	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent variables.	In logistic regression, there should not be collinearity between the independent variable.

Implementation of Logistic Regression

How to find whether Logistic Regression to be applied?

1. Is the dataset supervised ? (having output column?)
2. Is the input data numeric?
3. Is the input data linear?
4. Is the output column categorical?
5. Is the output column probabilistic?

Yes!... you may use logistic regression

Is the dataset Linear?

1. Use Correlation
2. Plot scatter chart
3. may use pair plots

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sb
from sklearn import linear_model
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
df = pd.read_csv("Admission_Predict.csv")
print(df.shape)
print(df.info())
```

```
(400, 10)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
 9   output             400 non-null    object 
dtypes: float64(4), int64(5), object(1)
memory usage: 31.4+ KB
None
```

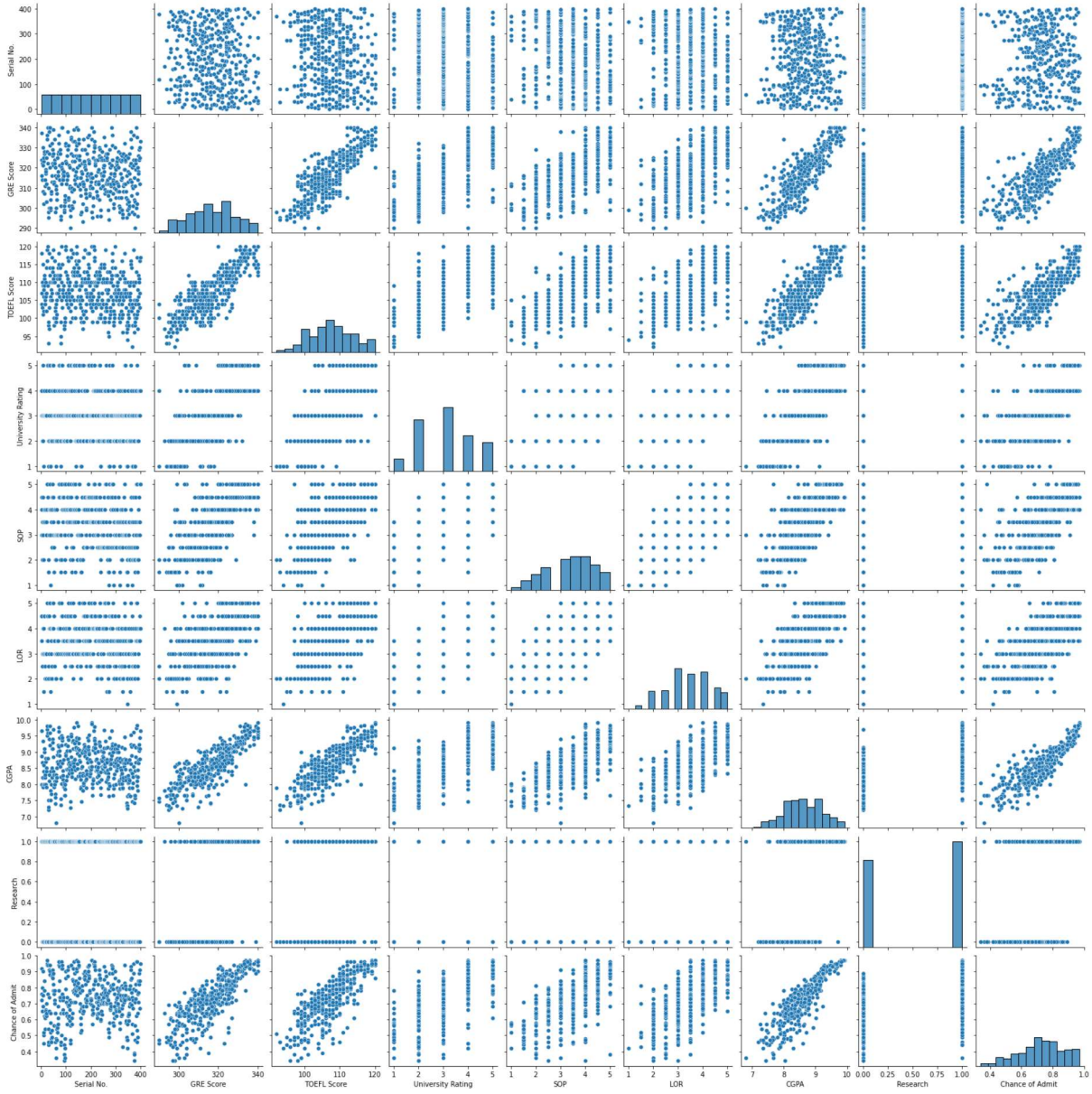
In [4]:
```
print(df[df['output']=='YES'].count())
```

```
Serial No.           247
GRE Score            247
TOEFL Score          247
University Rating    247
SOP                  247
LOR                  247
CGPA                 247
Research             247
Chance of Admit      247
output               247
dtype: int64
```

Draw scatter plot

In [5]:
```
sb.pairplot(df)
```

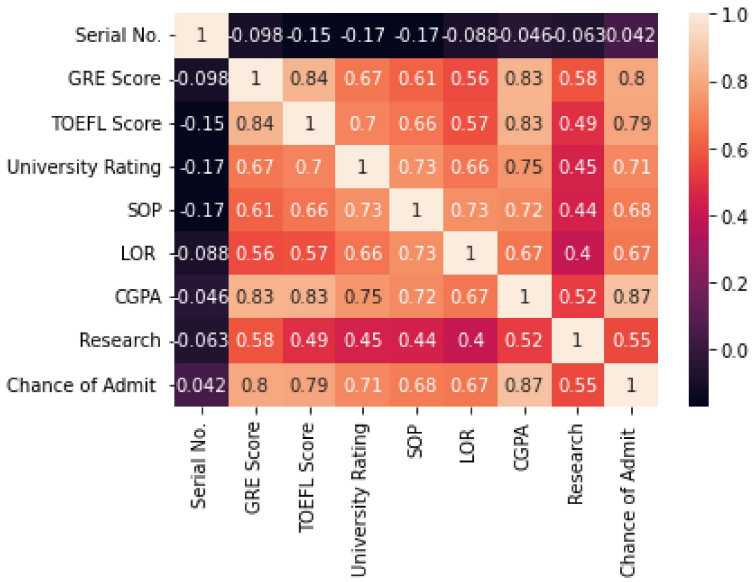
Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x2383ae24a30>
```



In [6]:
```
df1  = df.corr()
print(df1)
sb.heatmap(df1,annot=True)
```

```
                   Serial No.  GRE Score  TOEFL Score  University Rating  \
Serial No.           1.000000  -0.097526    -0.147932          -0.169948
GRE Score           -0.097526   1.000000     0.835977           0.668976
TOEFL Score         -0.147932   0.835977     1.000000           0.695590
University Rating   -0.169948   0.668976     0.695590           1.000000
SOP                 -0.166932   0.612831     0.657981           0.734523
LOR                 -0.088221   0.557555     0.567721           0.660123
CGPA                -0.045608   0.833060     0.828417           0.746479
Research            -0.063138   0.580391     0.489858           0.447783
Chance of Admit      0.042336   0.802610     0.791594           0.711250

                         SOP       LOR       CGPA  Research  Chance of Admit
Serial No.         -0.166932 -0.088221 -0.045608 -0.063138          0.042336
GRE Score           0.612831  0.557555  0.833060  0.580391          0.802610
TOEFL Score         0.657981  0.567721  0.828417  0.489858          0.791594
University Rating   0.734523  0.660123  0.746479  0.447783          0.711250
SOP                 1.000000  0.729593  0.718144  0.444029          0.675732
LOR                 0.729593  1.000000  0.670211  0.396859          0.669889
CGPA                0.718144  0.670211  1.000000  0.521654          0.873289
Research            0.444029  0.396859  0.521654  1.000000          0.553202
Chance of Admit     0.675732  0.669889  0.873289  0.553202          1.000000
```

Out[6]: <AxesSubplot:>



In [7]:
```
df.drop(columns=['Serial No.'],inplace=True)
df.head()
```

Out[7]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	output
0	337	118	4	4.5	4.5	9.65	1	0.92	YES
1	324	107	4	4.0	4.5	8.87	1	0.76	YES
2	316	104	3	3.0	3.5	8.00	1	0.72	YES
3	322	110	3	3.5	2.5	8.67	1	0.80	YES
4	314	103	2	2.0	3.0	8.21	0	0.65	NO

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
output               0
dtype: int64
```

train & test

In [9]:
```
from sklearn import linear_model
regress = linear_model.LogisticRegression()
df2 = df.values
#print(df2)
train_x = (df2[:,0:7])
train_y = df2[:,8]
print(train_x)
print(train_y)
```

```
[[337 118 4 ... 4.5 9.65 1]
 [324 107 4 ... 4.5 8.87 1]
 [316 104 3 ... 3.5 8.0 1]
 ...
 [330 116 4 ... 4.5 9.45 1]
 [312 103 3 ... 4.0 8.78 0]
 [333 117 4 ... 4.0 9.66 1]]
['YES' 'YES' 'YES' 'YES' 'NO' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES'
 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO'
 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO'
 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO'
 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO'
 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'YES' 'NO' 'NO'
 'NO' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES'
 'NO' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'NO' 'NO' 'NO' 'NO' 'NO'
 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO'
 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO'
 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES'
 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES' 'YES' 'NO'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'NO'
 'NO' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES' 'NO' 'YES' 'YES' 'YES'
 'NO' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES'
 'YES' 'YES' 'NO' 'NO' 'YES' 'NO' 'NO' 'YES' 'NO' 'YES' 'YES' 'YES' 'YES'
 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'NO' 'NO'
 'NO' 'NO' 'NO' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES' 'YES' 'NO' 'YES' 'YES'
 'YES' 'YES' 'YES' 'NO' 'YES' 'YES' 'YES' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES'
 'YES' 'NO' 'NO' 'NO' 'NO' 'NO' 'YES' 'YES' 'YES' 'YES' 'NO' 'YES' 'YES'
 'NO' 'NO' 'NO' 'YES' 'NO' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'YES' 'NO'
 'YES']
```

Minmax normalization

In [10]:
```
from sklearn.preprocessing import MinMaxScaler

ms = MinMaxScaler()
train_x = ms.fit_transform(train_x)
print(train_x)
```

```
[[0.94       0.92857143 0.75       ... 0.875      0.91346154 1.        ]
 [0.68       0.53571429 0.75       ... 0.875      0.66346154 1.        ]
 [0.52       0.42857143 0.5        ... 0.625      0.38461538 1.        ]
 ...
 [0.8        0.85714286 0.75       ... 0.875      0.84935897 1.        ]
 [0.44       0.39285714 0.5        ... 0.75       0.63461538 0.        ]
 [0.86       0.89285714 0.75       ... 0.75       0.91666667 1.        ]]
```

Regression line fitting

In [11]:
```
regress.fit (train_x,train_y)
# The coefficients y= m0x0+m1x1+...m11x11+c
print ('Coefficients: ', regress.coef_)
print ('Intercept: ',regress.intercept_)
```

```
Coefficients:  [[3.06261347 1.9615749  1.57578933 0.19147494 1.31976518 2.72350744
  0.7347784 ]]
Intercept:  [-5.45961447]
```

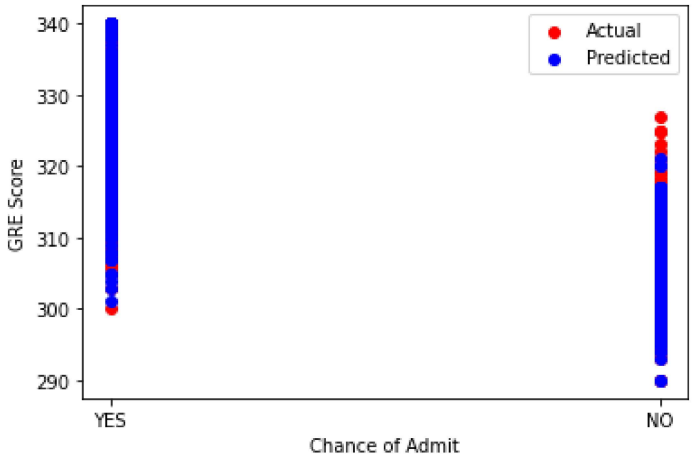
Prediction

In [12]:
```
y_predicted = regress.predict(train_x)
for i in range(0,len(train_x)):
    print(train_y[i],y_predicted[i])
df['Pred']=y_predicted
df
df.to_csv("log_result_1.csv")
```

```
YES YES
YES YES
YES YES
YES YES
NO NO
YES YES
YES YES
NO NO
NO NO
NO YES
NO YES
YES YES
YES YES
NO YES
NO NO
NO NO
NO YES
NO YES
NO YES
```

Plotting

In [13]:
```
plt.scatter(df['output'],df['GRE Score'],color='red',label='Actual')
plt.scatter(df['Pred'],df['GRE Score'],color = 'blue',label = 'Predicted' )
plt.xlabel('Chance of Admit')
plt.ylabel('GRE Score')
plt.legend()
plt.show()
```



Metrics

In [14]:
```
from sklearn import metrics
print('Accucary:', metrics.accuracy_score(train_y, y_predicted))
print('Confusion Matrix\n',metrics.confusion_matrix(train_y,y_predicted))
```

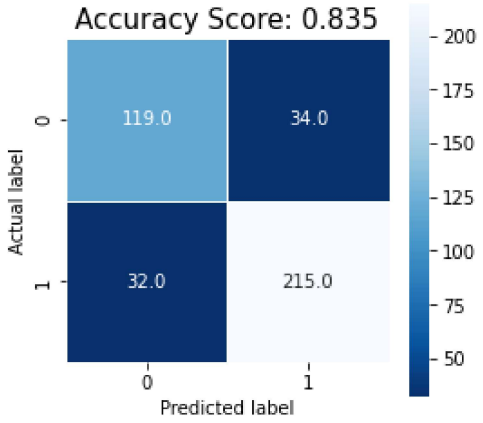
```
Accucary: 0.835
Confusion Matrix
 [[119  34]
 [ 32 215]]
```

		Actual	
		Negative	Positive
Predicted	Negative	TN	FN
	Positive	FP	TP

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

In [15]:
```
plt.figure(figsize=(4,4))
sb.heatmap(metrics.confusion_matrix(train_y,y_predicted), annot=True, fmt=".1f", linewid
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score(train_y, y_predic
plt.title(all_sample_title, size = 15);
```



In [16]:
```
from sklearn.metrics import classification_report
print(classification_report(train_y,y_predicted))
```

```
              precision    recall  f1-score   support

          NO       0.79      0.78      0.78       153
         YES       0.86      0.87      0.87       247

    accuracy                           0.83       400
   macro avg       0.83      0.82      0.82       400
weighted avg       0.83      0.83      0.83       400
```

12. Write a Python program to implement confusion matrix and also find precision, recall, F1-Score and support

Logistic regression is a type of regression we can use when the response variable is binary. One common way to evaluate the quality of a logistic regression model is to create a confusion matrix, which is a 2×2 table that shows the predicted values from the model vs. the actual values from the test dataset.

#To create a confusion matrix for a logistic regression model in Python, we can use the confusion_matrix() function from the sklearn package:

In [1]:
```
#define array of actual values
y_actual = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [2]:
```
#define array of predicted values
y_predicted = [0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [3]:
```
from sklearn import metrics
#create confusion matrix
c_matrix = metrics.confusion_matrix(y_actual, y_predicted)

#print confusion matrix
print(c_matrix)
```

```
[[6 4]
 [2 8]]
```

In [4]:
```
TN = c_matrix[0][0]
FP = c_matrix[0][1]
FN = c_matrix[1][0]
TP = c_matrix[1][1]
```

In [5]:
```
#Precision = TP/(TP+FP)
precision= TP / (TP + FP)
print(precision)
```

```
0.6666666666666666
```

In [6]:
```
#Recall = TP/(TP+FN)
recall= TP / (TP + FN)
print(recall)
```

```
0.8
```

In [7]:
```
#F1-score= 2 * [(Precision*recall)/(Precision+recall)]
F1_score= 2 * (precision*recall)/(precision+recall)
print(F1_score)
```

```
0.7272727272727272
```

In [8]:
```
#Accuracy= (TP+FN)/(TP+TN+FP+FN]
Accuracy= (TP+TN)/(TP+TN+FP+FN)
print(Accuracy)
```

```
0.7
```

In [9]:
```
import sklearn
print("Precision:",sklearn.metrics.precision_score(y_actual, y_predicted))
print("Recall:",sklearn.metrics.recall_score(y_actual, y_predicted))
print("F1-Score:",sklearn.metrics.f1_score(y_actual, y_predicted))
print("Accuracy:",sklearn.metrics.accuracy_score(y_actual, y_predicted))
```

```
Precision: 0.6666666666666666
Recall: 0.8
F1-Score: 0.7272727272727272
Accuracy: 0.7
```

In [10]:
```
import sklearn
print(sklearn.metrics.classification_report(y_actual, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.75      0.60      0.67        10
           1       0.67      0.80      0.73        10

    accuracy                           0.70        20
   macro avg       0.71      0.70      0.70        20
weighted avg       0.71      0.70      0.70        20
```

13. Write a Program to implement Decision Tree Regressor using custom dataset

1. Import the required libraries.

In [2]:
```
# import numpy package for arrays and stuff
import numpy as np

# import matplotlib.pyplot for plotting our result
import matplotlib.pyplot as plt

# import pandas for importing csv files
import pandas as pd
```

2. Initialize and print the Dataset.

In [3]:
```
#dataset = pd.read_csv('Data.csv')
# import dataset
# dataset = pd.read_csv('Data.csv')
# alternatively open up .csv file to read data

dataset = np.array(
[['Asset Flip', 100, 1000],
['Text Based', 500, 3000],
['Visual Novel', 1500, 5000],
['2D Pixel Art', 3500, 8000],
['2D Vector Art', 5000, 6500],
['Strategy', 6000, 7000],
['First Person Shooter', 8000, 15000],
['Simulator', 9500, 20000],
['Racing', 12000, 21000],
['RPG', 14000, 25000],
['Sandbox', 15500, 27000],
['Open-World', 16500, 30000],
['MMOFPS', 25000, 52000],
['MMORPG', 30000, 80000]
])
```

In [4]:
```
dataset
```

Out[4]:
```
array([['Asset Flip', '100', '1000'],
       ['Text Based', '500', '3000'],
       ['Visual Novel', '1500', '5000'],
       ['2D Pixel Art', '3500', '8000'],
       ['2D Vector Art', '5000', '6500'],
       ['Strategy', '6000', '7000'],
       ['First Person Shooter', '8000', '15000'],
       ['Simulator', '9500', '20000'],
       ['Racing', '12000', '21000'],
       ['RPG', '14000', '25000'],
       ['Sandbox', '15500', '27000'],
       ['Open-World', '16500', '30000'],
       ['MMOFPS', '25000', '52000'],
       ['MMORPG', '30000', '80000']], dtype='<U20')
```

3. Select all the rows and column 1 from the dataset to “X”.

In [5]:
```
# select all rows by : and column 1
# by 1:2 representing features
X = dataset[:, 1:2].astype(int)

# print X
print(X)
```

```
[[  100]
 [  500]
 [ 1500]
 [ 3500]
 [ 5000]
 [ 6000]
 [ 8000]
 [ 9500]
 [12000]
 [14000]
 [15500]
 [16500]
 [25000]
 [30000]]
```

4: Select all of the rows and column 2 from the dataset to “y”.

In [6]:
```
# select all rows by : and column 2
# by 2 to Y representing labels
y = dataset[:, 2].astype(int)

# print y
print(y)
```

```
[ 1000  3000  5000  8000  6500  7000 15000 20000 21000 25000 27000 30000
 52000 80000]
```

5. Fit decision tree regressor to the dataset

In [7]:
```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, y)
```

Out[7]:
```
DecisionTreeRegressor(random_state=0)
```

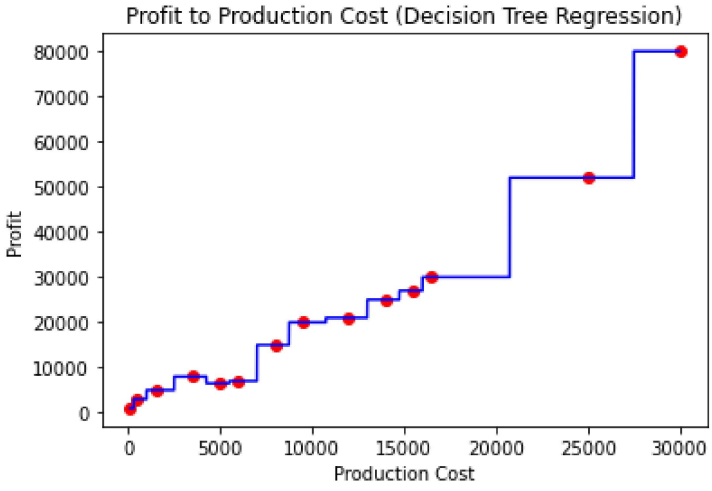
6. Predicting a new value

In [8]:
```
# predicting a new value

# test the output by changing values, like 3750
y_pred = regressor.predict([[1000]])

# print the predicted price
print("Predicted price: % d\n"% y_pred)
```

```
Predicted price:  3000
```

7: Visualising the result

In [9]:
```
# arange for creating a range of values
# from min value of X to max value of X
# with a difference of 0.01 between two
# consecutive values
X_grid = np.arange(min(X), max(X), 0.01)

# reshape for reshaping the data into
# a len(X_grid)*1 array, i.e. to make
# a column out of the X_grid values
X_grid = X_grid.reshape((len(X_grid), 1))

# scatter plot for original data
plt.scatter(X, y, color = 'red')

# plot predicted data
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')

# specify title
plt.title('Profit to Production Cost (Decision Tree Regression)')

# specify X axis label
plt.xlabel('Production Cost')

# specify Y axis label
plt.ylabel('Profit')

# show the plot
plt.show()
```



8: The tree is finally exported and shown in the TREE STRUCTURE below, visualized using http://www.webgraphviz.com/ by copying the data from the 'tree.dot' file.

In [10]:
```
# import export_graphviz
from sklearn.tree import export_graphviz

# export the decision tree to a tree.dot file
# for visualizing the plot easily anywhere
export_graphviz(regressor, out_file ='tree.dot',
                feature_names =['Production Cost'])
```

14. Write a Program to implement KNN using diabetes dataset.

k-NN is one of the most fundamental algorithms for classification and regression in the Machine Learning world.

Python implementation

In [62]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
#Let's start with importing necessary libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import KFold
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [63]:
```
data = pd.read_csv("diabetes.csv") # Reading the Data
data.head()
```

Out[63]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

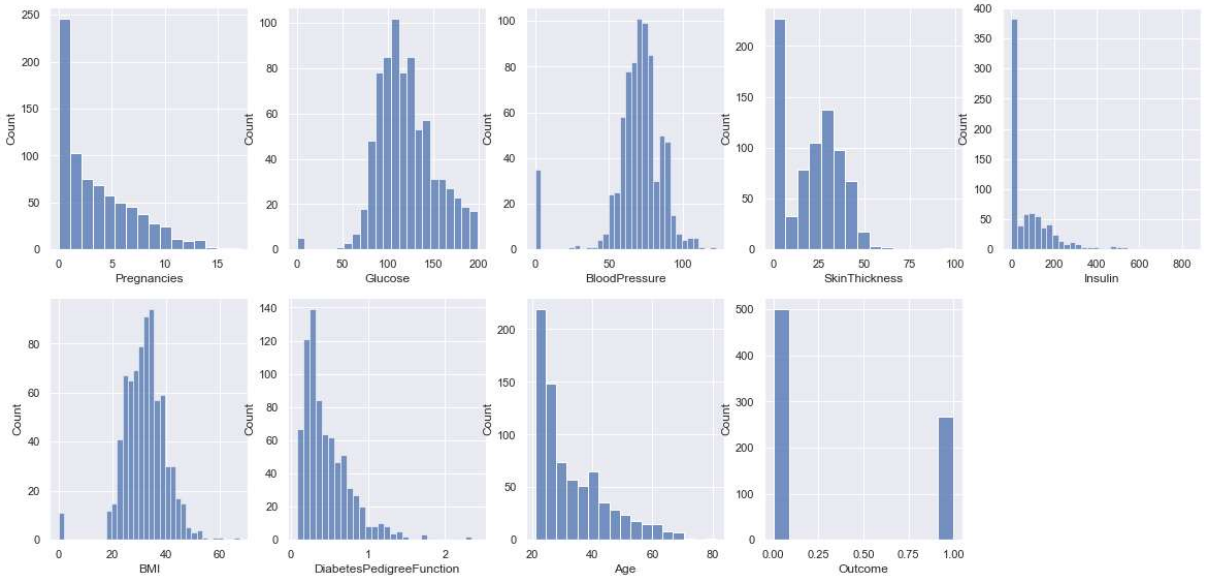
In [64]:
```
data.describe()
```

Out[64]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.4
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.3
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.2
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.3
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.6
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.4

It seems that there are no missing values in our data. Great, let's see the distribution of data:

In [65]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=9 :     # as there are 9 columns in the data
        ax = plt.subplot(5,5,plotnumber)
        sns.histplot(data[column])
        plt.xlabel(column,fontsize=12)
        #plt.ylabel('Salary',fontsize=12)
    plotnumber+=1
plt.show()
```

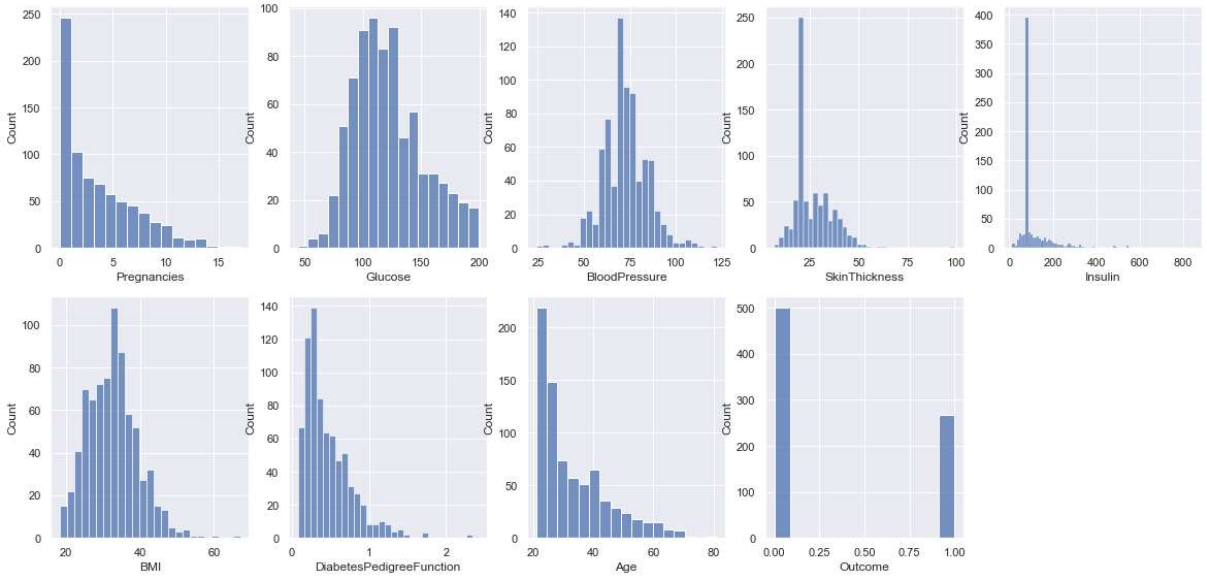


We can see there is some skewness in the data, let's deal with data.

Also, we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible. You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

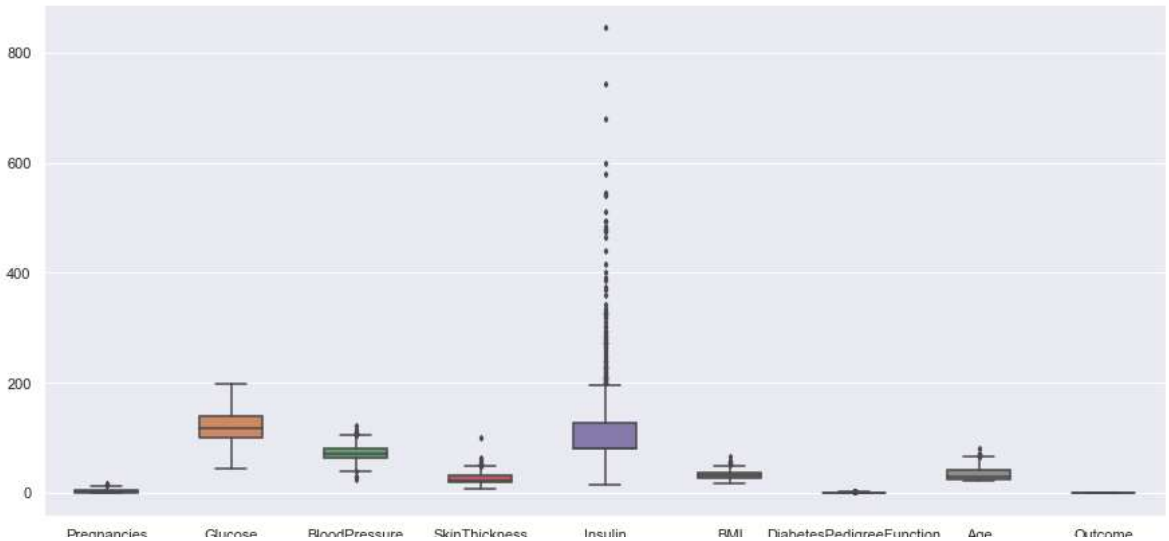
In [66]:
```
# replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [67]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=9 :
        ax = plt.subplot(5,5,plotnumber)
        sns.histplot(data[column])
        plt.xlabel(column,fontsize=12)
        #plt.ylabel('Salary',fontsize=12)
    plotnumber+=1
plt.show()
```



In [68]:
```
fig, ax = plt.subplots(figsize=(15,7))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

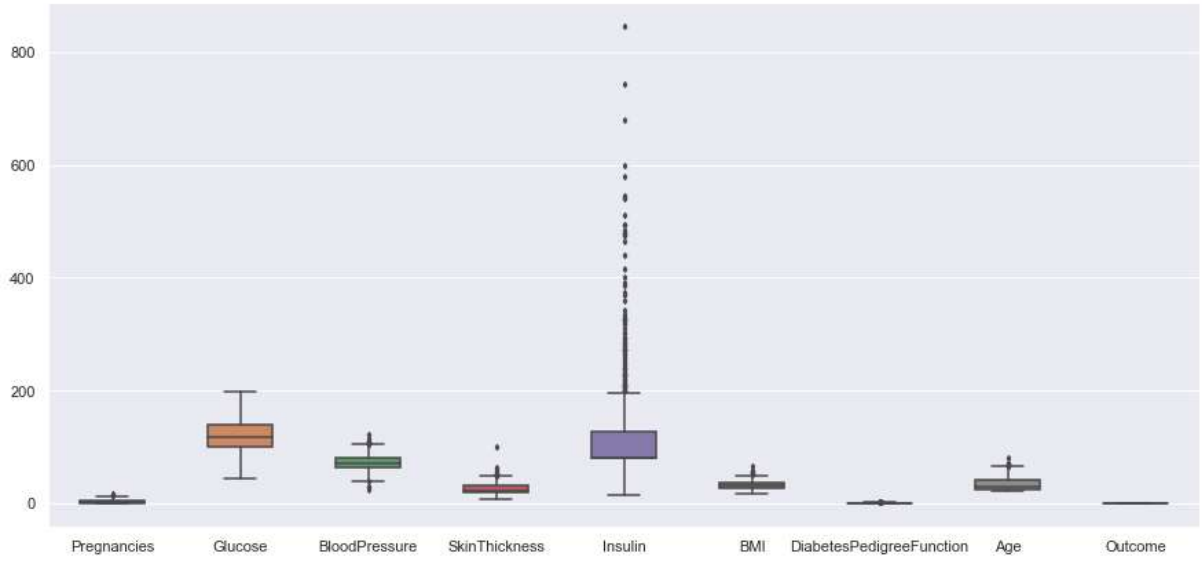
Out[68]: <AxesSubplot:>



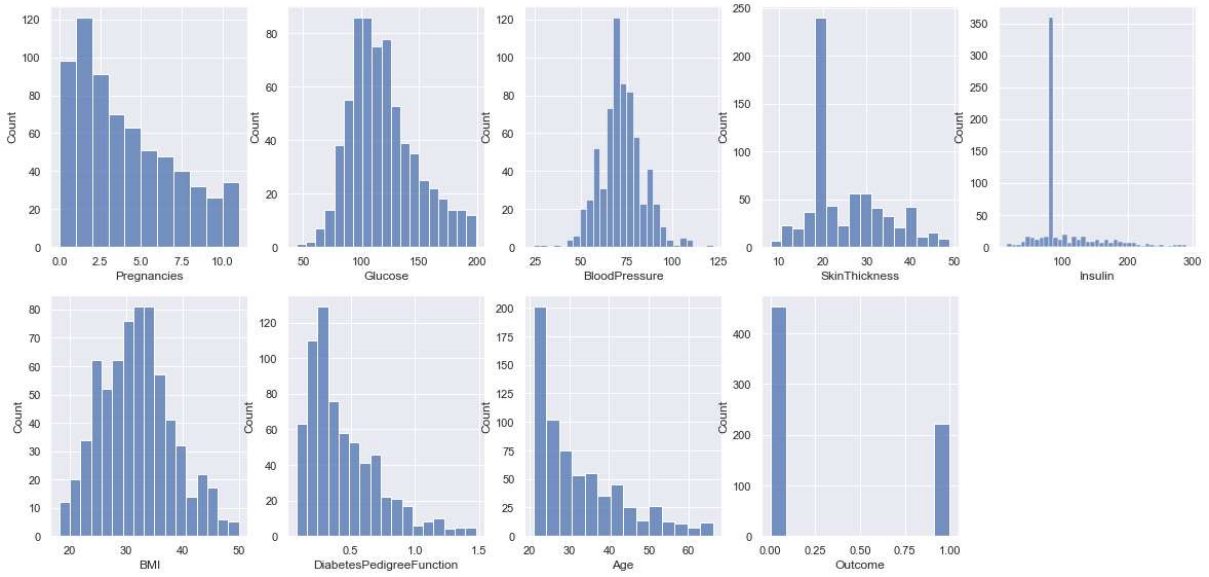
In [69]:
```
q = data['Pregnancies'].quantile(0.98)
# we are removing the top 2% data from the Pregnancies column
data_cleaned = data[data['Pregnancies']<q]
q = data_cleaned['BMI'].quantile(0.99)
# we are removing the top 1% data from the BMI column
data_cleaned  = data_cleaned[data_cleaned['BMI']<q]
q = data_cleaned['SkinThickness'].quantile(0.99)
# we are removing the top 1% data from the SkinThickness column
data_cleaned  = data_cleaned[data_cleaned['SkinThickness']<q]
q = data_cleaned['Insulin'].quantile(0.95)
# we are removing the top 5% data from the Insulin column
data_cleaned  = data_cleaned[data_cleaned['Insulin']<q]
q = data_cleaned['DiabetesPedigreeFunction'].quantile(0.99)
# we are removing the top 1% data from the DiabetesPedigreeFunction column
data_cleaned  = data_cleaned[data_cleaned['DiabetesPedigreeFunction']<q]
q = data_cleaned['Age'].quantile(0.99)
# we are removing the top 1% data from the Age column
data_cleaned  = data_cleaned[data_cleaned['Age']<q]
```

In [70]:
```
fig, ax = plt.subplots(figsize=(15,7))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

Out[70]: <AxesSubplot:>



In [71]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data_cleaned:
    if plotnumber<=9 :
        ax = plt.subplot(5,5,plotnumber)
        sns.histplot(data_cleaned[column])
        plt.xlabel(column,fontsize=12)
        #plt.ylabel('Salary',fontsize=12)
    plotnumber+=1
plt.show()
```

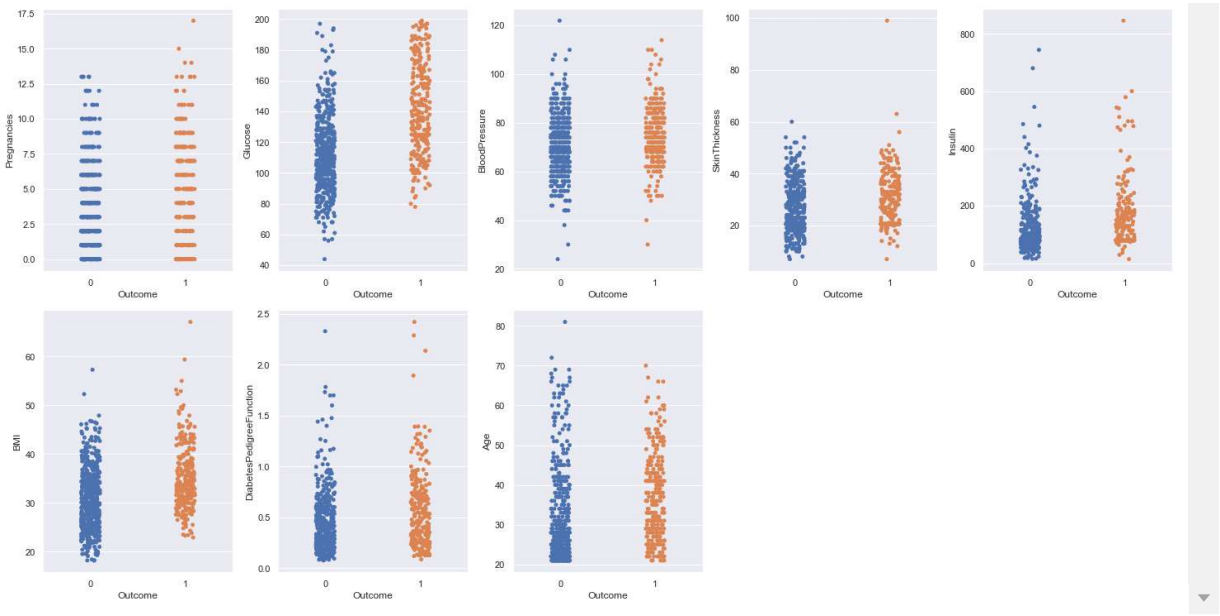


In [72]:
```
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [73]:

```python
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in X:
    if plotnumber<=9 :
        ax = plt.subplot(5,5,plotnumber)
        sns.stripplot(y,X[column])
    plotnumber+=1
plt.tight_layout()
```

```
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variables as keyword args: x, y. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
```



Great!! Let's proceed by checking multicollinearity in the dependent variables. Before that, we should scale our data. Let's use the standard scaler for that.

In [74]:
```
scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

In [75]:
```
vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor(X_scaled,i) for i in range(X_scaled.shape[1])]
vif["Features"] = X.columns

#Let's check the values
vif
```

Out[75]:

	vif	Features
0	1.431075	Pregnancies
1	1.347308	Glucose
2	1.247914	BloodPressure
3	1.450510	SkinThickness
4	1.262111	Insulin
5	1.550227	BMI
6	1.058104	DiabetesPedigreeFunction
7	1.605441	Age

In [76]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scaled,y, test_size= 0.25)
```

In [77]:
```
# let's fit the data into kNN model and see how well it performs:
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
```

Out[77]:
```
KNeighborsClassifier()
```

In [78]:
```
y_pred = knn.predict(x_test)
```

In [79]:
```
knn.score(x_train,y_train)
```

Out[79]:
```
0.8246527777777778
```

In [80]:
```
print("The accuracy score is : ", accuracy_score(y_test,y_pred))
```

```
The accuracy score is :  0.7552083333333334
```

Let's try to increase the accuracy by using hyperparameter tuning.

In [81]:
```
param_grid = { 'algorithm' : ['ball_tree', 'kd_tree', 'brute'],
               'leaf_size' : [18,20,25,27,30,32,34],
               'n_neighbors' : [3,5,7,9,10,11,12,13]
             }
```

In [82]:
```
gridsearch = GridSearchCV(knn, param_grid,verbose=3)
```

In [83]:
```
gridsearch.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 168 candidates, totalling 840 fits
[CV 1/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=3;, score=0.776 total ti
me=   0.0s
[CV 2/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=3;, score=0.696 total ti
me=   0.0s
[CV 3/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=3;, score=0.730 total ti
me=   0.0s
[CV 4/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=3;, score=0.722 total ti
me=   0.0s
[CV 5/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=3;, score=0.661 total ti
me=   0.0s
[CV 1/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=5;, score=0.724 total ti
me=   0.0s
[CV 2/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=5;, score=0.704 total ti
me=   0.0s
[CV 3/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=5;, score=0.739 total ti
me=   0.0s
[CV 4/5] END algorithm=ball_tree, leaf_size=18, n_neighbors=5;, score=0.748 total ti
me=   0.0s
```

In [84]:
```
# let's see the  best parameters according to gridsearch
gridsearch.best_params_
```

Out[84]:
```
{'algorithm': 'ball_tree', 'leaf_size': 18, 'n_neighbors': 13}
```

In [85]:
```
# we will use the best parameters in our k-NN algorithm and check if accuracy is increas
knn = KNeighborsClassifier(algorithm = 'ball_tree', leaf_size =18, n_neighbors =13)
```

In [86]:
```
knn.fit(x_train,y_train)
```

Out[86]:
```
KNeighborsClassifier(algorithm='ball_tree', leaf_size=18, n_neighbors=13)
```

In [87]:
```
knn.score(x_train,y_train)
```

Out[87]:
```
0.78125
```

In [88]:
```
knn.score(x_test,y_test)
```

Out[88]:
```
0.7552083333333334
```

15. Write a Program to implement SVM using iris dataset and perform hyper parameter tuning.

(dataset available in scikit-learn library)

Loading Data

In [2]:
```
#Import scikit-learn dataset library
from sklearn import datasets
```

In [3]:
```
#Load dataset
iris = datasets.load_iris()
```

Exploring Data

In [4]:
```
dir(iris)
```

Out[4]:
```
['DESCR',
 'data',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [5]:
```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [6]:
```
# print the names of the 4 features
print("Features: ", iris.feature_names)
```

```
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

In [8]:
```
# print the label type of iris(class_0:setosa, class_1:versicolor, class_2:virginica)
print("Labels: ", iris.target_names)
```

```
Labels:  ['setosa' 'versicolor' 'virginica']
```

In [9]:
```
# print data(feature)shape
iris.data.shape
```

Out[9]:
```
(150, 4)
```

In [10]:
```
# print the iris labels (0:setosa, 1:versicolor, 2:virginica)
print(iris.target)
```

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

In [11]:
```
import pandas as pd
df=pd.DataFrame(iris.data, columns=iris.feature_names)
df["target"]=iris.target
df
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [12]:
```
df["flowers_name"]=df.target.apply(lambda x:iris.target_names[x])
df
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flowers_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

Visualization

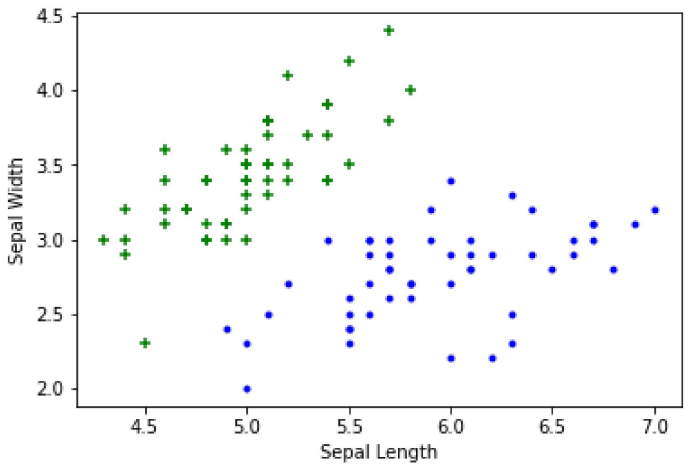
In [13]:
```
from matplotlib import pyplot as plt
%matplotlib inline
```

In [14]:
```
df0=df[df.target==0]
df1=df[df.target==1]
df2=df[df.target==2]
```

Sepal length vs Sepal Width (Setosa vs Versicolor)

In [15]:
```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

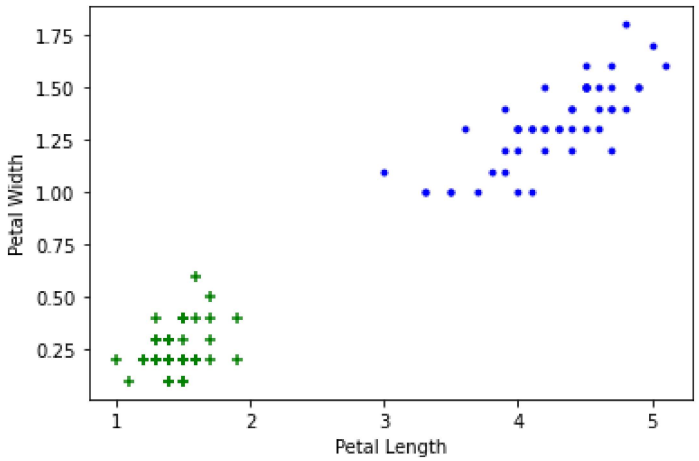
Out[15]:
```
<matplotlib.collections.PathCollection at 0x2079b587c40>
```



Petal length vs Pepal Width (Setosa vs Versicolor)

In [16]:
```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x2079bd3d1f0>
```



Splitting Data

In [17]:
```
# Import train_test_split function
from sklearn.model_selection import train_test_split
X = df.drop(['target','flowers_name'], axis='columns')
y = df.target
# Split dataset into training set and test set
x_train,x_test,y_train,y_test = train_test_split(X, y, test_size= 0.25, random_state = 3
len(x_train)
len(x_test)
```

Out[17]:
```
38
```

Train Using Support Vector Machine (SVM)

In [18]:
```
#Import SVM algorithm and call SVC(Support vector classifier)
from sklearn.svm import SVC
#Create a Gaussian Classifier
model = SVC()
```

In [19]:
```
#Train the model using the training sets
model.fit(x_train,y_train)
```

Out[19]:
```
SVC()
```

In [20]:
```
#Predict the response for test dataset
y_pred = model.predict(x_test)
```

In [21]:
```
model.score(x_test, y_test)
```

Out[21]:
```
0.9210526315789473
```

Evaluating Model

In [22]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn.metrics import accuracy_score

# Model Accuracy, how often is the classifier correct?
print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Accuracy:  0.9210526315789473
```

In [23]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[23]:
```
array([[12,  0,  0],
       [ 0, 11,  0],
       [ 0,  3, 12]], dtype=int64)
```

In [24]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.79      1.00      0.88        11
           2       1.00      0.80      0.89        15

    accuracy                           0.92        38
   macro avg       0.93      0.93      0.92        38
weighted avg       0.94      0.92      0.92        38
```

Tune parameters:

1. Regularization (C)

In [25]:
```
model_C = SVC(C=1)
model_C.fit(x_train, y_train)
model_C.score(x_test, y_test)
```

Out[25]:
```
0.9210526315789473
```

In [26]:
```
model_C = SVC(C=5)
model_C.fit(x_train, y_train)
model_C.score(x_test, y_test)
```

Out[26]:
```
0.9736842105263158
```

2. Gamma

In [27]:
```
model_g = SVC(gamma=2)
model_g.fit(x_train, y_train)
model_g.score(x_test, y_test)
```

Out[27]:
```
0.9473684210526315
```

As we increase gamma values accuracy decreases, hence we use gamma = 2 only

3. Kernel (linear, rbf)

In [28]:
```
model_linear_kernal = SVC(kernel='rbf')
model_linear_kernal.fit(x_train, y_train)
model_linear_kernal.score(x_test, y_test)
```

Out[28]:
```
0.9210526315789473
```

FINAL MODEL

In [29]:
```
final_model = SVC(C=5, gamma=2, kernel='linear')
final_model.fit(x_train, y_train)
print("Accuracy:",final_model.score(x_test, y_test))
```

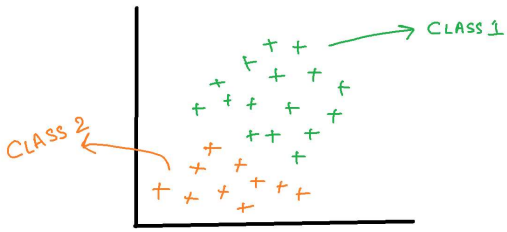
```
Accuracy: 0.9736842105263158
```

In [ ]:

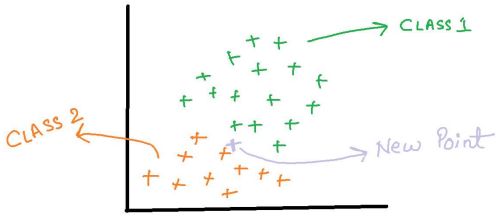
16. Write a Program to implement Naive Bayes with binary class label using diabetes dataset.

Algorithm steps: ¶

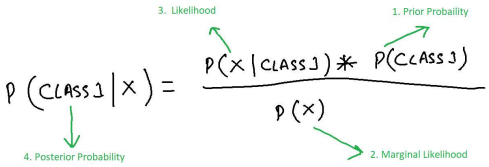
1. Let's consider that we have a binary classification problem i.e., we have two classes in our data as shown below.



2. Now suppose if we are given with a new data point, to which class does that point belong to?

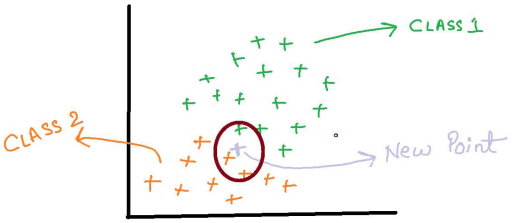


3. The formula for a point 'X' to belong in class1 can be written as:



Where the numbers represent the order in which we are going to calculate different probabilities.

4. A similar formula can be utilised for class 2 as well.
5. Probability of class 1 can be written as: $P(class1) = \frac{Numberofpointsinclass1}{Totalnumberofpoints} = \frac{16}{26} = 0.62$
6. For calculating the probability of X, we draw a circle around the new point and see how many points(excluding the new point) lie inside that circle.



The points inside the circle are considered to be similar points.
$P(X) = \frac{Numberofsimilarobservation}{TotalObservations} = \frac{3}{26} = 0.12$ 7. Now, we need to calculate the probability of a point to be in the circle that we have made given that it's of class 1.
$P(X|Class1) = \frac{Numberofpointsinclass1insidethecircle}{Totalnumberofpointsinclass1} = \frac{1}{16} = 0.06$ 8. We can substitute all the values into the formula in step 3. We get: $P(Class1|X) = \frac{0.06*0.62}{0.12} = 0.31$ 9. And if we calculate the probability that X belongs to Class2, we'll get 0.69. It means that our point belongs to class 2.

The Generalization for Multiclass:

The approach discussed above can be generalised for multiclass problems as well. Suppose, P1, P2, P3… Pn are the probabilities for the classes C1,C2,C3…Cn, then the point X will belong to the class for which the probability is maximum. Or mathematically the point belongs to the result of : $argmax(P1, P2, P3 \ldots . Pn)$

The Difference

You can notice a major difference in the way in which the Naïve Bayes algorithm works form other classification algorithms. It does not first try to learn how to classify the points. It directly uses the label to identify the two separate classes and then it predicts the class to which the new point shall belong.

Why it is called Naïve Bayes?

The entire algorithm is based on Bayes's theorem to calculate probability. So, it also carries forward the assumptions for the Bayes's theorem. But those assumptions(that the features are independent) might not always be true when implemented over a real-world dataset. So, those assumptions are considered *Naïve* and hence the name.

Python Implementation

In [4]:
```
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
data = pd.read_csv("diabetes.csv") # Reading the Data
data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [6]:
```
data.describe()
```

Out[6]:

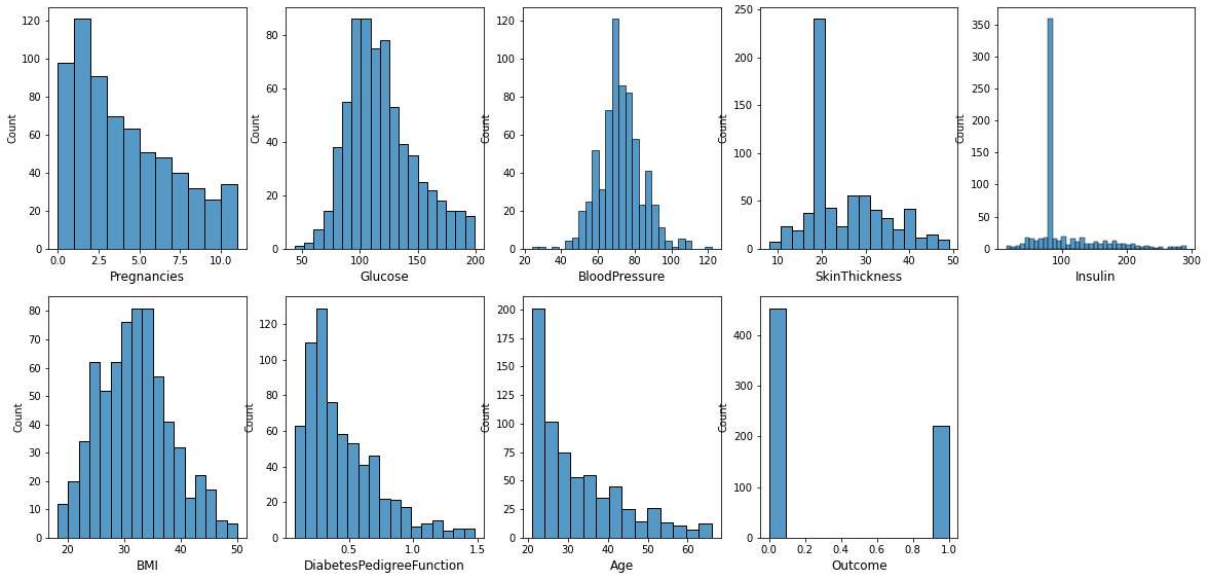
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.4
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.3
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.2
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.3
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.6
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.4

we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible. You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [7]:
```
# replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [8]:
```
# Handling the Outliers

q = data['Pregnancies'].quantile(0.98)
# we are removing the top 2% data from the Pregnancies column
data_cleaned = data[data['Pregnancies']<q]
q = data_cleaned['BMI'].quantile(0.99)
# we are removing the top 1% data from the BMI column
data_cleaned  = data_cleaned[data_cleaned['BMI']<q]
q = data_cleaned['SkinThickness'].quantile(0.99)
# we are removing the top 1% data from the SkinThickness column
data_cleaned  = data_cleaned[data_cleaned['SkinThickness']<q]
q = data_cleaned['Insulin'].quantile(0.95)
# we are removing the top 5% data from the Insulin column
data_cleaned  = data_cleaned[data_cleaned['Insulin']<q]
q = data_cleaned['DiabetesPedigreeFunction'].quantile(0.99)
# we are removing the top 1% data from the DiabetesPedigreeFunction column
data_cleaned  = data_cleaned[data_cleaned['DiabetesPedigreeFunction']<q]
q = data_cleaned['Age'].quantile(0.99)
# we are removing the top 1% data from the Age column
data_cleaned  = data_cleaned[data_cleaned['Age']<q]
```

In [10]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data_cleaned:
    if plotnumber<=9 :
        ax = plt.subplot(5,5,plotnumber)
        sns.histplot(data_cleaned[column])
        plt.xlabel(column,fontsize=12)
        #plt.ylabel('Salary',fontsize=12)
    plotnumber+=1
plt.show()
```



In [11]:
```
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [12]:
```
# we need to scale our data as well

scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

In [13]:
```
# This is how our data looks now after scaling.
X_scaled
```

Out[13]:
```
array([[ 0.63994726,  0.86527574, -0.0210444 , ...,  0.16725546,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.20598931, -0.51658286, ..., -0.85153454,
        -0.36506078, -0.19067191],
       [ 1.23388019,  2.01597855, -0.68176235, ..., -1.33182125,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 , -0.02240928, -0.0210444 , ..., -0.90975111,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.14197684, -1.01212132, ..., -0.34213954,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.94297153, -0.18622389, ..., -0.29847711,
        -0.47378505, -0.87137393]])
```

In [14]:
```
# now we will check for multicollinearity using VIF(Variance Inflation factor)
vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor(X_scaled,i) for i in range(X_scaled.shape[1])]
vif["Features"] = X.columns

#Let's check the values
vif
```

Out[14]:

	vif	Features
0	1.431075	Pregnancies
1	1.347308	Glucose
2	1.247914	BloodPressure
3	1.450510	SkinThickness
4	1.262111	Insulin
5	1.550227	BMI
6	1.058104	DiabetesPedigreeFunction
7	1.605441	Age

All the VIF values are less than 5 and are very low. That means no multicollinearity. Now, we can go ahead with fitting our data to the model. Before that, let's split our data in test and training set.

In [15]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scaled,y, test_size= 0.25, random_sta
```

In [16]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

In [17]:
```
model.fit(x_train,y_train)
```

Out[17]:
```
GaussianNB()
```

In [18]:
```
import pickle
# Writing different model files to file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(model,f)

with open('standardScalar.sav', 'wb') as f:
    pickle.dump(scalar,f)
```

In [19]:
```
y_pred = model.predict(x_test)
```

In [20]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.7864583333333334
```

In [21]:
```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[21]:
```
array([[109,  16],
       [ 25,  42]], dtype=int64)
```

In [22]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [23]:
```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_nega
Accuracy
```

Out[23]:
```
0.7864583333333334
```

In [24]:
```
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[24]:
```
0.872
```

In [25]:
```
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[25]:
```
0.8134328358208955
```

In [26]:
```
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[26]:
```
0.8416988416988417
```

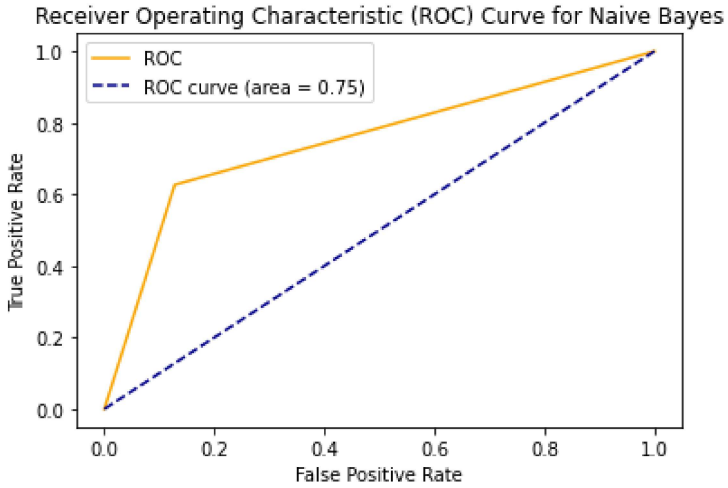
In [27]:
```
# Area Under Curve
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[27]:
```
0.7494328358208956
```

So far we have been doing grid search to maximise the accuracy of our model. Here, we'll follow a different approach. We'll create two models, one with Logistic regression and other with Naïve Bayes and we'll compare the AUC. The algorithm having a better AUC shall be considered for production deployment.

In [28]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [29]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
plt.legend()
plt.show()
```



Advantages:

- Naive Bayes is extremely fast for both training and prediction as they not have to learn to create separate classes.
- Naive Bayes provides a direct probabilistic prediction.
- Naive Bayes is often easy to interpret.
- Naive Bayes has fewer (if any) parameters to tune

Disadvantages:

- The algorithm assumes that the features are independent which is not always the scenario
- Zero Frequency i.e. if the category of any categorical variable is not seen in training data set even once then model assigns a zero probability to that category and then a prediction cannot be made.

17. Write a Program to implement Naive Bayes with multi class label using wine dataset.

(dataset available in scikit-learn library)

Loading Data

In [1]:
```
#Import scikit-learn dataset library
from sklearn import datasets
```

In [2]:
```
#Load dataset
wine = datasets.load_wine()
```

Exploring Data

In [3]:
```
dir(wine)
```

Out[3]:
```
['DESCR', 'data', 'feature_names', 'frame', 'target', 'target_names']
```

In [ ]:
```
print(wine.DESCR)
```

In [5]:
```
# print the names of the 13 features
print("Features: ", wine.feature_names)
```

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_ph
enols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hu
e', 'od280/od315_of_diluted_wines', 'proline']
```

In [6]:
```
# print the label type of wine(class_0, class_1, class_2)
print("Labels: ", wine.target_names)
```

```
Labels:  ['class_0' 'class_1' 'class_2']
```

In [7]:
```
# print data(feature)shape
wine.data.shape
```

Out[7]:
```
(178, 13)
```

In [8]:
```
# print the wine data features (top 5 records)
print(wine.data[0:5])
```

```
[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
  3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
  2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
 [1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
  3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]]
```

In [9]:
```
# print the wine labels (0:Class_0, 1:class_2, 2:class_2)
print(wine.target)
```

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

Splitting Data

In [10]:
```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
x_train,x_test,y_train,y_test = train_test_split(wine.data, wine.target, test_size= 0.25
```

Model Generation

In [11]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
model = GaussianNB()
```

In [12]:
```
#Train the model using the training sets
model.fit(x_train,y_train)
```

Out[12]:
```
GaussianNB()
```

In [13]:
```
#Predict the response for test dataset
y_pred = model.predict(x_test)
```

Evaluating Model

In [14]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn.metrics import accuracy_score

# Model Accuracy, how often is the classifier correct?
print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Accuracy:  0.9555555555555556
```

In [15]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[15]:
```
array([[10,  0,  0],
       [ 2, 16,  0],
       [ 0,  0, 17]], dtype=int64)
```

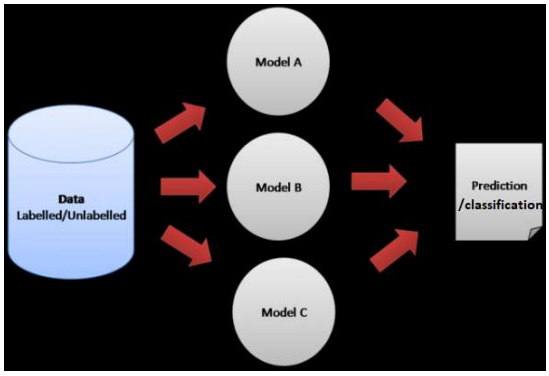
In [16]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91        10
           1       1.00      0.89      0.94        18
           2       1.00      1.00      1.00        17

    accuracy                           0.96        45
   macro avg       0.94      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

18. Write a Program to implement Ensemble learning (Model 1: DT, Model 2: SVM, Model 3: Logistic, Model 4: KNN) using T-shirt size dataset.

Ensemble Learning:

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).



Model 1: Logistic
Model 2 : Decision Tree
Model 3: SVM
Model 4: KNN

In [1]:
```
#importing Reauired library
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```
dataset= pd.read_excel('T-shirt dataset.xls')
```

In [3]:
```
dataset.shape
```

Out[3]:
```
(26, 3)
```

In [4]:
```
dataset.head()
```

Out[4]:

	HEIGHT(IN CMS)	WEIGHT(IN KGS)	T SHIRT SIZE
0	158	58	M
1	158	59	M
2	158	63	M
3	160	59	M
4	160	60	M

In [5]:
```
#independent attribute (height and weight)
x = dataset.iloc[:,[0,1]].values
#dependent attribute T-shirt size
y = dataset.iloc[:,2].values
y
```

Out[5]:
```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'L', 'L', 'L', 'L', 'L', 'L',
       'L', 'L', 'L', 'L', 'L', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M'],
      dtype=object)
```

As we know that our T-shirt column is categorical data, we have to encode this attribute. LabelEncoder is used for encoding the data.

In [6]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder() # le is an object from label encoder
```

In [7]:
```
y_new = y
```

In [8]:
```
y_new=le.fit_transform(y_new)
```

In [9]:
```
print(y) # values in the form of M / L
print(y_new) # values in the form of 1 and 0
```

```
['M' 'M' 'M' 'M' 'M' 'M' 'M' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L'
 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M']
[1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
```

In [10]:
```
y= y_new # updating y_new values in our target variable
```

Splitting data into train and test

In [11]:
```
# Split into training and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state
```

In [12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
model1_logistic = LogisticRegression()
model2_decision = DecisionTreeClassifier(max_depth=5)
model3_SVM = SVC()
model4_KNN = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
```

In [13]:
```
model1_logistic.fit(x_train, y_train)
```

Out[13]:
```
LogisticRegression()
```

In [14]:
```
model2_decision.fit(x_train, y_train)
```

Out[14]:
```
DecisionTreeClassifier(max_depth=5)
```

In [15]:
```
model3_SVM.fit(x_train, y_train)
```

Out[15]:
```
SVC()
```

In [16]:
```
model4_KNN.fit(x_train, y_train)
```

Out[16]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=3)
```

In [17]:
```
y_pred1 = model1_logistic.predict(x_test) #LR
y_pred2 = model2_decision.predict(x_test) #DT
y_pred3 = model3_SVM.predict(x_test) #SVM
y_pred4 = model4_KNN.predict(x_test) #KNN
```

In [18]:
```
print("Actual y_test data : ", y_test)
print("Prediction by LR : ", y_pred1)
print("Prediction by DT :", y_pred2)
print("Prediction by SVM : ", y_pred3)
print("Prediction by KNN :", y_pred4)
```

```
Actual y_test data :  [0 1 1 0 0 0 1]
Prediction by LR :  [0 1 1 0 0 0 1]
Prediction by DT : [0 1 1 0 0 1 1]
Prediction by SVM :  [1 1 1 1 1 1 1]
Prediction by KNN : [0 1 1 0 0 1 1]
```

Evaluation of Model

In [19]:
```
#confusiojn matrix for different models
from sklearn.metrics import confusion_matrix
cm_LR = confusion_matrix(y_test, y_pred1)
cm_DT = confusion_matrix(y_test, y_pred2)
cm_SVM = confusion_matrix(y_test, y_pred3)
cm_KNN = confusion_matrix(y_test, y_pred4)
```

In [20]:
```
print("Confusion_Matrix for LR:\n", cm_LR)
print("\nConfusion_Matrix for DT:\n", cm_DT)
print("\nConfusion_Matrix for SVM:\n",cm_SVM)
print("\nConfusion_Matrix for KNN:\n",cm_KNN)
```

```
Confusion_Matrix for LR:
 [[4 0]
 [0 3]]

Confusion_Matrix for DT:
 [[3 1]
 [0 3]]

Confusion_Matrix for SVM:
 [[0 4]
 [0 3]]

Confusion_Matrix for KNN:
 [[3 1]
 [0 3]]
```

In [21]:
```
#when we need to calculate the accuracy of model under classification
#always use testing dataset to get the result.
from sklearn.metrics import accuracy_score
LR_acc = accuracy_score(y_pred1, y_test)
DT_acc = accuracy_score(y_pred2, y_test)
SVM_acc = accuracy_score(y_pred3, y_test)
KNN_acc = accuracy_score(y_pred4, y_test)
```

In [22]:
```
print("Accuracy of LR:", LR_acc)
print("\nAccuracy of DT:", DT_acc)
print("\nAccuracy of SVM:",SVM_acc)
print("\nAccuracy of KNN:",KNN_acc)
```

```
Accuracy of LR: 1.0

Accuracy of DT: 0.8571428571428571

Accuracy of SVM: 0.42857142857142855

Accuracy of KNN: 0.8571428571428571
```

Ensemble Learning Model

In this work, we have defined each of the four machine learning model 5 times (5*4=20). Then finally max voting classifier method is used where the class which has been predicted mostly by the week learners will be the fianl class prediction of the ensemble model.

In [23]:
```
#estimator is a variable and it contains or append all the model details in list
estimator = []
#Definig 5 logistic Regression models
#model11, model12, model13, model14, model15 are the object of LR
model11=LogisticRegression()
estimator.append(('logistic1', model11))
model12=LogisticRegression()
estimator.append(('logistic2', model12))
model13=LogisticRegression()
estimator.append(('logistic3', model13))
model14=LogisticRegression()
estimator.append(('logistic4', model14))
model15=LogisticRegression()
estimator.append(('logistic5', model15))

#printing the value of estimators
estimator
```

Out[23]:
```
[('logistic1', LogisticRegression()),
 ('logistic2', LogisticRegression()),
 ('logistic3', LogisticRegression()),
 ('logistic4', LogisticRegression()),
 ('logistic5', LogisticRegression())]
```

In [24]:
```
#Definig 5 Decision Tree models
#model21, model22, model23, model24, model25 are the object of DT
model21=DecisionTreeClassifier(max_depth = 2)
estimator.append(('cart1', model21))
model22=DecisionTreeClassifier(max_depth = 5)
estimator.append(('cart2', model22))
model23=DecisionTreeClassifier(max_depth = 3)
estimator.append(('cart3', model23))
model24=DecisionTreeClassifier(max_depth = 6)
estimator.append(('cart4', model24))
model25=DecisionTreeClassifier(max_depth = 1)
estimator.append(('cart5', model25))

#printing the value of estimators
estimator
```

Out[24]:
```
[('logistic1', LogisticRegression()),
 ('logistic2', LogisticRegression()),
 ('logistic3', LogisticRegression()),
 ('logistic4', LogisticRegression()),
 ('logistic5', LogisticRegression()),
 ('cart1', DecisionTreeClassifier(max_depth=2)),
 ('cart2', DecisionTreeClassifier(max_depth=5)),
 ('cart3', DecisionTreeClassifier(max_depth=3)),
 ('cart4', DecisionTreeClassifier(max_depth=6)),
 ('cart5', DecisionTreeClassifier(max_depth=1))]
```

In [25]:
```
#Definig 5 SVM models
#model31, model32, model33, model34, model35 are the object of SVM
model31=SVC(kernel ='linear')
estimator.append(('svm1', model31))
model32=SVC(kernel ='rbf')
estimator.append(('svm2', model32))
model33=SVC(kernel ='linear')
estimator.append(('svm3', model33))
model34=SVC(kernel ='poly')
estimator.append(('svm4', model34))
model35=SVC(kernel ='linear')
estimator.append(('svm5', model35))

#printing the value of estimators
estimator
```

Out[25]:
```
[('logistic1', LogisticRegression()),
 ('logistic2', LogisticRegression()),
 ('logistic3', LogisticRegression()),
 ('logistic4', LogisticRegression()),
 ('logistic5', LogisticRegression()),
 ('cart1', DecisionTreeClassifier(max_depth=2)),
 ('cart2', DecisionTreeClassifier(max_depth=5)),
 ('cart3', DecisionTreeClassifier(max_depth=3)),
 ('cart4', DecisionTreeClassifier(max_depth=6)),
 ('cart5', DecisionTreeClassifier(max_depth=1)),
 ('svm1', SVC(kernel='linear')),
 ('svm2', SVC()),
 ('svm3', SVC(kernel='linear')),
 ('svm4', SVC(kernel='poly')),
 ('svm5', SVC(kernel='linear'))]
```

In [26]:
```
#Definig 5 KNN models
#model41, model42, model43, model44, model45 are the object of KNN
model41=KNeighborsClassifier(n_neighbors = 3, metric ='euclidean')
estimator.append(('knn1', model41))
model42=KNeighborsClassifier(n_neighbors = 5, metric ='euclidean')
estimator.append(('knn2', model42))
model43=KNeighborsClassifier(n_neighbors = 4, metric ='euclidean')
estimator.append(('knn3', model43))
model44=KNeighborsClassifier(n_neighbors = 7, metric ='euclidean')
estimator.append(('knn4', model44))
model45=KNeighborsClassifier(n_neighbors = 2, metric ='euclidean')
estimator.append(('knn5', model45))

#printing the value of estimators
estimator
```

Out[26]:
```
[('logistic1', LogisticRegression()),
 ('logistic2', LogisticRegression()),
 ('logistic3', LogisticRegression()),
 ('logistic4', LogisticRegression()),
 ('logistic5', LogisticRegression()),
 ('cart1', DecisionTreeClassifier(max_depth=2)),
 ('cart2', DecisionTreeClassifier(max_depth=5)),
 ('cart3', DecisionTreeClassifier(max_depth=3)),
 ('cart4', DecisionTreeClassifier(max_depth=6)),
 ('cart5', DecisionTreeClassifier(max_depth=1)),
 ('svm1', SVC(kernel='linear')),
 ('svm2', SVC()),
 ('svm3', SVC(kernel='linear')),
 ('svm4', SVC(kernel='poly')),
 ('svm5', SVC(kernel='linear')),
 ('knn1', KNeighborsClassifier(metric='euclidean', n_neighbors=3)),
 ('knn2', KNeighborsClassifier(metric='euclidean')),
 ('knn3', KNeighborsClassifier(metric='euclidean', n_neighbors=4)),
 ('knn4', KNeighborsClassifier(metric='euclidean', n_neighbors=7)),
 ('knn5', KNeighborsClassifier(metric='euclidean', n_neighbors=2))]
```

In [27]:
```
#At last we need to ensemble our models by using estimators
#variable and fit the model using estimator varaible
#Defining the ensemble model
# here we are using VotingClassifier because it is basic
#ensemble learning model, others are bagging and boosting classifier
from sklearn.ensemble import VotingClassifier
ensemble = VotingClassifier(estimator)
ensemble.fit(x_train, y_train)
```

Out[27]:
```
VotingClassifier(estimators=[('logistic1', LogisticRegression()),
                             ('logistic2', LogisticRegression()),
                             ('logistic3', LogisticRegression()),
                             ('logistic4', LogisticRegression()),
                             ('logistic5', LogisticRegression()),
                             ('cart1', DecisionTreeClassifier(max_depth=2)),
                             ('cart2', DecisionTreeClassifier(max_depth=5)),
                             ('cart3', DecisionTreeClassifier(max_depth=3)),
                             ('cart4'...
                             ('svm4', SVC(kernel='poly')),
                             ('svm5', SVC(kernel='linear')),
                             ('knn1',
                              KNeighborsClassifier(metric='euclidean',
                                                   n_neighbors=3)),
                             ('knn2', KNeighborsClassifier(metric='euclidean')),
                             ('knn3',
                              KNeighborsClassifier(metric='euclidean',
                                                   n_neighbors=4)),
                             ('knn4',
                              KNeighborsClassifier(metric='euclidean',
                                                   n_neighbors=7)),
                             ('knn5',
                              KNeighborsClassifier(metric='euclidean',
                                                   n_neighbors=2))])
```

In [28]:
```
y_pred = ensemble.predict(x_test)
```

In [29]:
```
y_pred
```

Out[29]:
```
array([0, 1, 1, 0, 0, 1, 1])
```

In [30]:
```
y_test
```

Out[30]:
```
array([0, 1, 1, 0, 0, 0, 1])
```

In [31]:
```
#confusion matrix
cm_Ensembler = confusion_matrix(y_test, y_pred)
cm_Ensembler
```

Out[31]:
```
array([[3, 1],
       [0, 3]], dtype=int64)
```

In [32]:
```
acc_Ensembler = accuracy_score(y_pred, y_test)
print("Accuracy: ",acc_Ensembler)
```

```
Accuracy:  0.8571428571428571
```