MACHINE LEARNING JOURNAL

M.Sc.(Statistics)

SHABNAM ANWAR
Roll No. 927
Ramniranjan Jhunjhunwala College

Index

PRACTICAL NO: 1

AIM: Perform Data preprocessing and feature engineering.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
import warnings
```

```
warnings.filterwarnings("ignore")
```

```
train = pd.read_csv("../input/titanic/train.csv")
```

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
train.shape
 (891, 12)
```

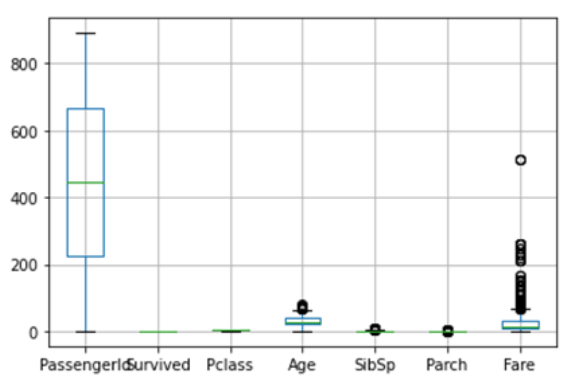
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

train.isnull().sum()/len(train)

```
PassengerId    0.000000
Survived       0.000000
Pclass         0.000000
Name           0.000000
Sex            0.000000
Age            0.198653
SibSp          0.000000
Parch          0.000000
Ticket         0.000000
Fare           0.000000
Cabin          0.771044
Embarked       0.002245
dtype: float64
```

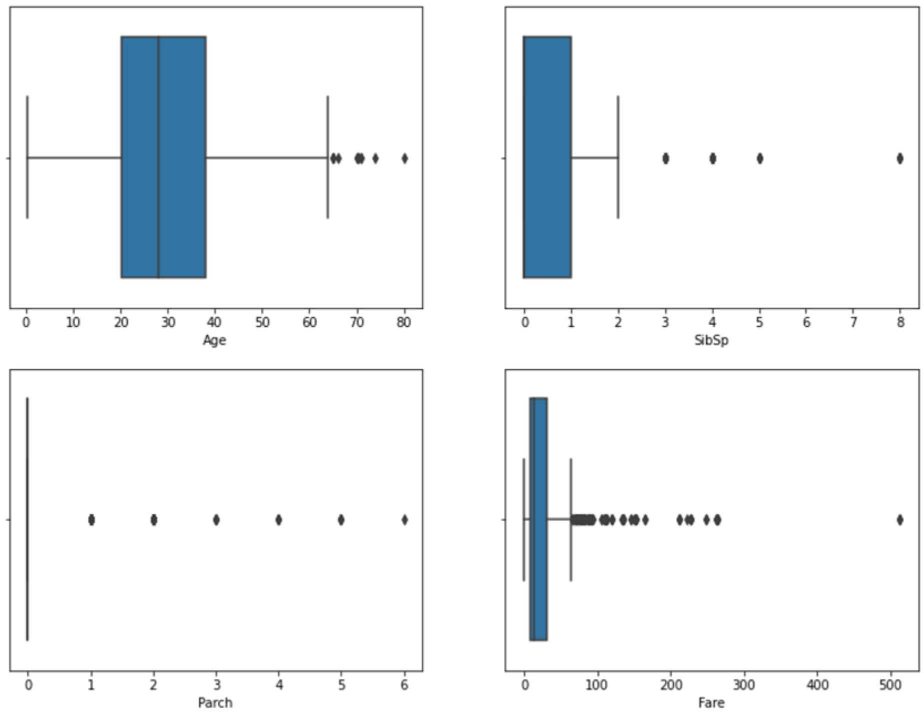
train.boxplot()



plt.figure(figsize=(12.8,9.6))

ax1 = plt.subplot(2,2,1)
ax2 = plt.subplot(2,2,2)
ax3 = plt.subplot(2,2,3)
ax4 = plt.subplot(2,2,4)

sns.boxplot(data=train,x='Age',ax=ax1)
sns.boxplot(data=train,x='SibSp',ax=ax2)
sns.boxplot(data=train,x='Parch',ax=ax3)
sns.boxplot(data=train,x='Fare',ax=ax4);




```
train.Embarked.mode()

train['Age'].fillna(train.Age.mean(), inplace=True)
train['Embarked'].fillna('S',inplace=True)

train.drop(columns=['Cabin'], inplace=True, axis=1)

train['family'] = train['SibSp'] + train['Parch']
train = train.iloc[:,[2,4,5,9,10,11,1]]

train.head(2)
```

	Pclass	Sex	Age	Fare	Embarked	family	Survived
0	3	male	22.0	7.2500	S	1	0
1	1	female	38.0	71.2833	C	1	1

Replacing Categorical variables to numbers:

```
train.Sex.replace('male', 0, inplace = True)
```

```
train.Sex.replace('female', 1, inplace = True)
train.Embarked.replace('S', 0, inplace = True)
train.Embarked.replace('C', 1, inplace = True)
train.Embarked.replace('Q', 2, inplace = True)

train_x = train.iloc[:,:-1]
train_y = train.iloc[:,-1]

from sklearn.feature_selection import SelectKBest, chi2

selector = SelectKBest(chi2, k=4)
selector.fit_transform(train_x,train_y)
cols = selector.get_support(indices=True)
train_new = train_x.iloc[:,cols]
pd.DataFrame(train_new)
```

	Pclass	Sex	Age	Fare
0	3	0	22.000000	7.2500
1	1	1	38.000000	71.2833
2	3	1	26.000000	7.9250
3	1	1	35.000000	53.1000
4	3	0	35.000000	8.0500
...	...	...	...	...
886	2	0	27.000000	13.0000
887	1	1	19.000000	30.0000
888	3	1	29.699118	23.4500
889	1	0	26.000000	30.0000
890	3	0	32.000000	7.7500

891 rows × 4 columns

PRACTICAL NO: 2

AIM: Implementing Simple Linear Regression Algorithm.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import
mean_squared_error,r2_score,mean_absolute_error,mean_absolute_percentage_error
```

```
df = pd.read_csv("placement.csv")
df
```

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57
...	...	...
195	6.93	2.46
196	5.89	2.57
197	7.21	3.24
198	7.63	3.96
199	6.22	2.33

200 rows × 2 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   cgpa     200 non-null    float64
 1   package  200 non-null    float64
dtypes: float64(2)
memory usage: 3.2 KB
```

df.isnull().sum()

```
cgpa       0
package    0
dtype: int64
```

df.shape

```
(200, 2)
```

df.size

```
400
```

df.describe()

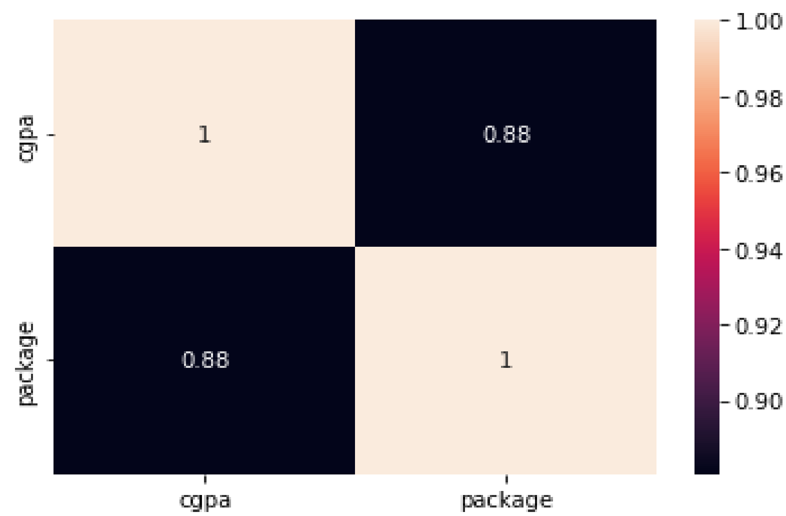
	cgpa	package
count	200.000000	200.000000
mean	6.990500	2.996050
std	1.069409	0.691644
min	4.260000	1.370000
25%	6.190000	2.487500
50%	6.965000	2.995000
75%	7.737500	3.492500
max	9.580000	4.620000

df.corr()

	cgpa	package
cgpa	1.000000	0.880692
package	0.880692	1.000000

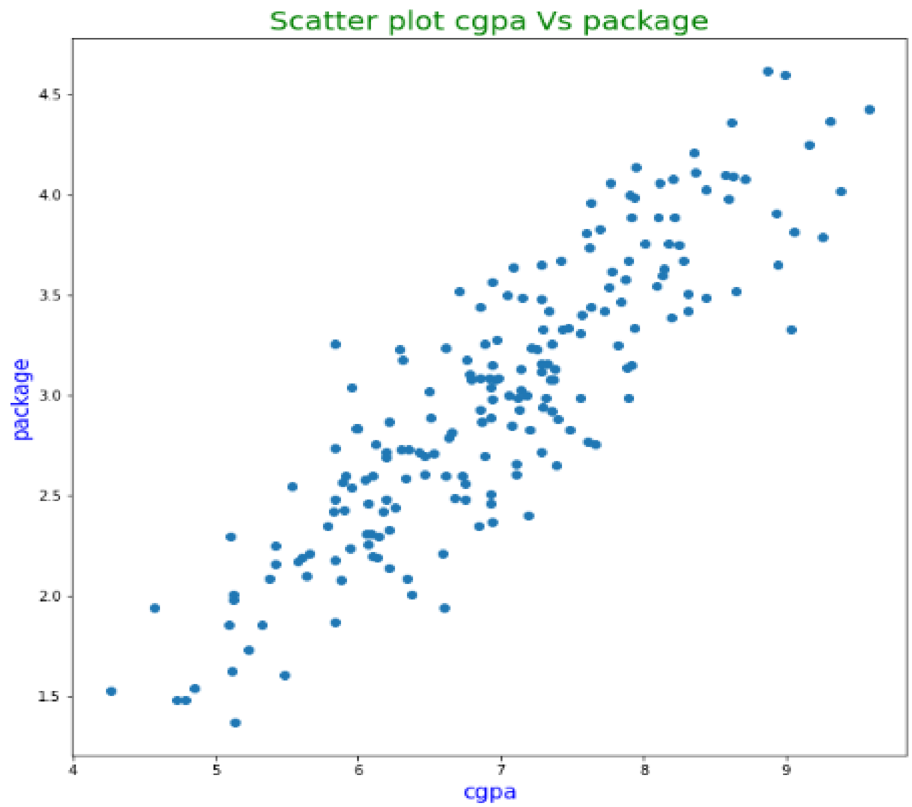
sns.heatmap(df.corr(),annot = True)

```
<AxesSubplot:>
```



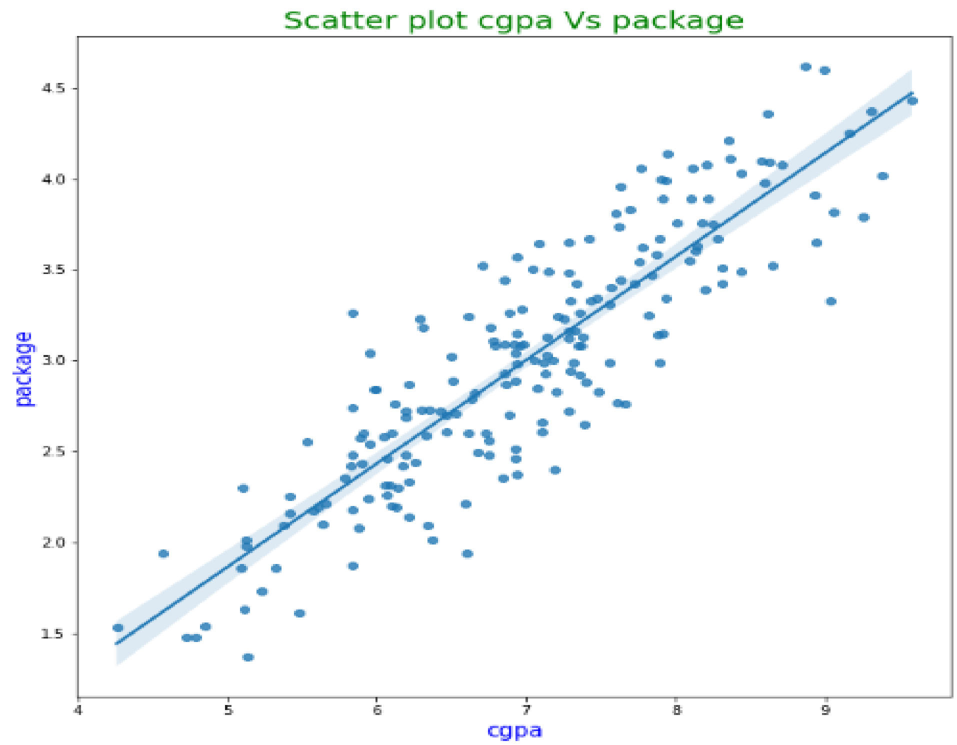

```
plt.figure(figsize=(10,10))
plt.scatter(df["cgpa"], df["package"])
plt.title("Scatter plot cgpa Vs package", size = 20, color = "Green")
plt.xlabel("cgpa", size = 15, color = "Blue")
plt.ylabel("package", size = 15, color = "Blue")
```

```
Text(0, 0.5, 'package')
```




```
plt.figure(figsize=(10,10))
sns.regplot(x = "cgpa", y = "package", data = df )
plt.title("Scatter plot cgpa Vs package", size = 20, color = "Green")
plt.xlabel("cgpa", size = 15, color = "Blue")
plt.ylabel("package", size = 15, color = "Blue")
```

```
Text(0, 0.5, 'package')
```



x = df.iloc[:, :-1].values

y = df.iloc[:, 1].values

x

```
array([[6.89],
       [5.12],
       [7.82],
       [7.42],
       [6.94],
       [7.89],
       [6.73],
       [6.75],
       [6.09],
       [8.31],
       [5.32],
       [6.61],
       [8.94],
       [6.93],
```

y

```
array([3.26, 1.98, 3.25, 3.67, 3.57, 2.99, 2.6 , 2.48, 2.31, 3.51, 1.86,
       2.6 , 3.65, 2.89, 3.42, 3.23, 2.35, 2.09, 2.98, 2.83, 3.16, 2.93,
       2.3 , 2.48, 2.71, 3.65, 3.42, 2.16, 2.24, 3.49, 3.26, 3.89, 3.08,
       2.73, 3.42, 2.87, 2.84, 2.43, 4.36, 3.33, 4.02, 2.7 , 2.54, 2.76,
       1.86, 3.58, 2.26, 3.26, 4.09, 4.62, 4.43, 3.79, 4.11, 2.61, 3.09,
       3.39, 2.74, 1.94, 3.09, 3.31, 2.19, 1.61, 2.09, 4.25, 2.92, 3.81,
       1.63, 2.89, 2.99, 2.94, 2.35, 3.34, 3.62, 4.03, 3.44, 3.28, 3.15,
       4.6 , 2.21, 3.  , 3.44, 2.2 , 2.17, 3.49, 1.53, 1.48, 2.77, 3.55,
       1.48, 2.72, 2.66, 2.14, 4.  , 3.08, 2.42, 2.79, 2.61, 2.84, 3.83,
       3.24, 4.14, 3.52, 1.37, 3.  , 3.74, 2.82, 2.19, 2.59, 3.54, 4.06,
       3.76, 2.25, 4.1 , 2.37, 1.87, 4.21, 3.33, 2.99, 2.88, 2.65, 1.73,
       3.02, 2.01, 2.3 , 2.31, 3.16, 2.6 , 3.11, 3.34, 3.12, 2.49, 2.01,
       2.48, 2.58, 2.83, 2.6 , 2.1 , 3.13, 3.89, 2.4 , 3.15, 3.18, 3.04,
       1.54, 2.42, 2.18, 2.46, 2.21, 3.4 , 3.67, 2.73, 2.76, 3.08, 3.99,
       2.85, 3.09, 3.13, 2.7 , 3.04, 4.08, 2.93, 3.33, 2.55, 3.91, 3.82,
       4.08, 3.98, 3.6 , 3.52, 4.37, 2.87, 3.76, 2.51, 2.56, 2.99, 3.5 ,
       3.23, 3.64, 3.63, 3.03, 2.72, 3.89, 2.08, 2.72, 3.14, 3.18, 3.47,
       2.44, 3.08, 4.06, 2.69, 3.48, 3.75, 1.94, 3.67, 2.46, 2.57, 3.24,
       3.96, 2.33])
```

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2,random_state = 0)

regressor = LinearRegression()
regressor.fit(x_train,y_train)
y_predic = regressor.predict(x_test)
y_predic

```
array([2.97012606, 2.55516816, 2.61856451, 3.40237388, 3.05657563,
       2.35921582, 3.51763996, 2.4687186 , 4.1227869 , 3.21794814,
       2.12868365, 3.19489493, 2.53787825, 3.05081232, 3.36779405,
       2.86062328, 2.63009112, 2.82028015, 2.37074243, 2.91825632,
       3.1660784 , 2.3361626 , 3.50611335, 3.9902309 , 2.20360661,
       1.90967809, 3.08539215, 2.6070379 , 1.9212047 , 1.90391479,
       2.91249302, 3.69630239, 2.60127459, 2.09410383, 2.50906173,
       3.64443265, 3.17184171, 2.97012606, 3.83462169, 2.50329842])
```

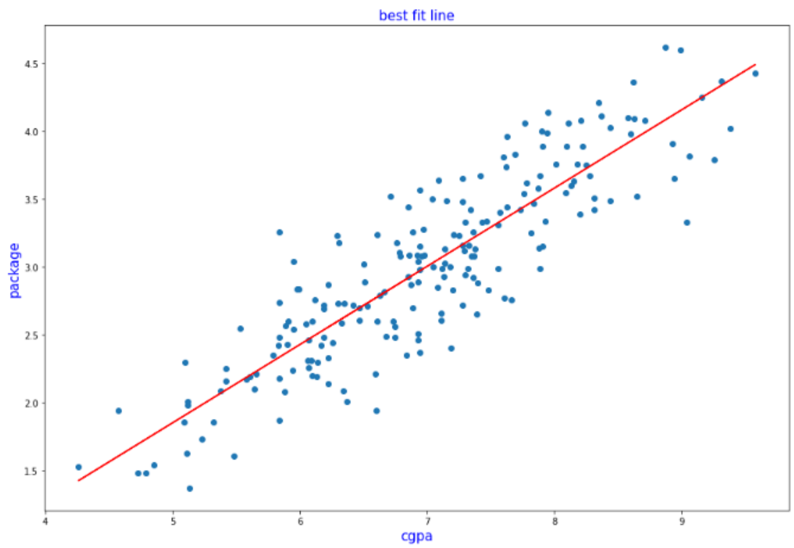
<u>To find the coefficient of determination:</u>
r2 = r2_score(y_test, y_predic)
print("R-Square:",r2)

```
R-Square: 0.7297167943957027
```

```
meanAbErr = mean_absolute_error(y_test, y_predic)
meanSqErr = mean_squared_error(y_test, y_predic)
rootMeanSqErr = np.sqrt(mean_squared_error(y_test, y_predic))
MAPE = mean_absolute_percentage_error(y_test, y_predic)

print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
print('Mean Absolute Percentage Error:', MAPE)
```

```
Mean Absolute Error: 0.2552433448620408
Mean Square Error: 0.10221131059334207
Root Mean Square Error: 0.31970503685951224
Mean Absolute Percentage Error: 0.09153058426151217
```

```
best_fitline = regressor.coef_*x+regressor.intercept_
plt.figure(figsize=(15,10))
plt.scatter(x, y)
plt.plot(x, best_fitline, color = 'Red')
plt.title('best fit line',size = 15,color = "Blue")
plt.xlabel("cgpa", size = 15, color = "Blue")
plt.ylabel("package", size = 15, color = "Blue")
plt.show()
```




```
df1 = pd.DataFrame({'Actual Revenue': y_test, 'Predicted Revenue': y_predic})
df1
```

	Actual Revenue	Predicted Revenue
0	2.98	2.970126
1	2.87	2.555168
2	2.59	2.618565
3	3.83	3.402374
4	3.64	3.056576
5	2.08	2.359216
6	2.99	3.517640
7	2.46	2.468719
8	3.65	4.122787
9	3.08	3.217948
10	1.61	2.128684
11	3.16	3.194895
12	2.72	2.537878
13	2.85	3.050812
14	3.44	3.367794
15	2.48	2.860623
16	2.73	2.630091
17	2.49	2.820280
18	2.43	2.370742
19	3.44	2.918256
20	2.72	3.166078
21	2.18	2.336163
22	3.58	3.506113

23	4.08	3.990231
24	2.19	2.203607
25	2.30	1.909678
26	3.03	3.085392
27	3.18	2.607038
28	2.01	1.921205
29	1.86	1.903915
30	2.35	2.912493
31	3.39	3.696302
32	2.73	2.601275
33	2.25	2.094104
34	2.30	2.509062
35	4.06	3.644433
36	3.12	3.171842
37	3.57	2.970126
38	3.49	3.834622
39	2.19	2.503298

PRACTICAL NO: 3
AIM: Implementing Multiple Linear Regression Algorithm.

```
from sklearn.datasets import make_regression
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
# import statsmodels.formula.api as sm
```

```
X,y = make_regression(n_samples=100, n_features=2, n_informative=2, n_targets=1,
noise=50)
```

```
(array([[-0.07416887, -1.50106347],
        [ 0.08981862,  0.86804921],
        [ 0.13763914,  0.6246196 ],
        [ 1.32497048,  0.09987875],
        [ 0.16985415, -0.75412521],
        [ 1.63966224, -0.66655202],
        [-0.16577141, -0.92542433],
        [-0.32379002, -0.05208789],
        [ 0.67152029,  0.32017016],
        [ 1.40472564,  0.23215087],
        [-0.13499266, -0.99507425],
        [ 0.02815869,  0.85490403],
        [-1.26675129, -0.44335334],
        [ 1.39521304,  0.8417314 ],
        [ 1.55804029,  0.65161665],
        [ 0.69981428, -0.79800113],
        [ 1.60789951, -1.33565222],
        [-0.80546426, -0.68018126],
        [ 0.98820902,  0.28969568],
```

```
df = pd.DataFrame({'feature1':X[:,0],'feature2':X[:,1],'target':y})
```

	feature1	feature2	target
0	-0.074169	-1.501063	-80.823951
1	0.089819	0.868049	-32.977755
2	0.137639	0.624620	15.357429
3	1.324970	0.099879	134.225794
4	0.169854	-0.754125	-3.998324
...	...	...	...
95	-0.824139	1.549801	-93.114267
96	-1.126762	-0.331811	28.577427
97	0.153932	1.175406	-48.435920
98	-0.686613	-0.370197	-38.114909
99	-0.024599	-1.320400	-18.787095

df.shape

(100,3)

df.corr()

```
: df.corr()
```

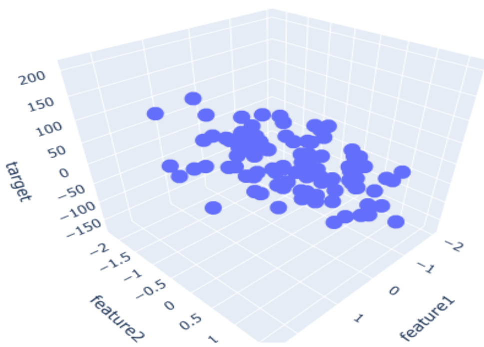
:

	feature1	feature2	target
feature1	1.000000	-0.071088	0.687024
feature2	-0.071088	1.000000	0.071115
target	0.687024	0.071115	1.000000

df.head()

	feature1	feature2	target
0	-0.074169	-1.501063	-80.823951
1	0.089819	0.868049	-32.977755
2	0.137639	0.624620	15.357429
3	1.324970	0.099879	134.225794
4	0.169854	-0.754125	-3.998324

```
# model = sm.ols('target ~ feature1 + feature2 ', df).fit()
# print(model.params)
fig = px.scatter_3d(df, x='feature1', y='feature2', z='target')

fig.show()
```




```
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=3)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
LinearRegression()
y_pred = lr.predict(X_test)
```

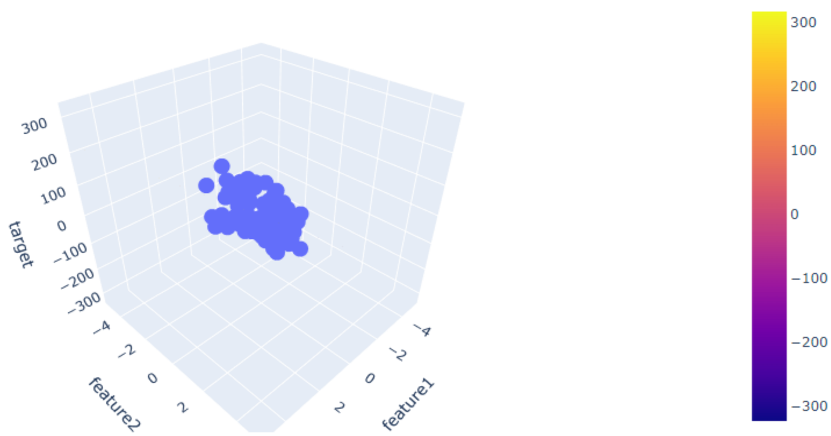
```
y_pred
```

```
array([-100.42590644,  -12.84926445,  -21.06724151,  -49.31343077,
        -66.55233686,   32.61403393,  -43.29716578,   22.33635923,
        -50.30394727,   26.77077492,  -46.16852522,   75.33903526,
        -43.86308315,   17.90724849, -100.68660491,  -16.18964946,
         62.73854042,   32.71289983,  -31.36184527,   17.23441678])
```

```
print("MAE",mean_absolute_error(y_test,y_pred))
print("MSE",mean_squared_error(y_test,y_pred))
print("R2 score",r2_score(y_test,y_pred))
```

```
MAE 37.34277411417523
MSE 1861.697408513403
R2 score 0.7247754218756458
```

```
x = np.linspace(-5, 5, 10)
y = np.linspace(-5, 5, 10)
xGrid, yGrid = np.meshgrid(y, x)
final = np.vstack((xGrid.ravel().reshape(1,100),yGrid.ravel().reshape(1,100))).T
z_final = lr.predict(final).reshape(10,10)
z = z_final

fig = px.scatter_3d(df, x='feature1', y='feature2', z='target')
fig.add_trace(go.Surface(x = x, y = y, z =z ))
fig.show()
```




```
lr.coef_
```

```
array([37.29151454, 71.34814438])
```

```
lr.intercept_
```

```
-2.5585253430261123
```

PRACTICAL NO: 4

AIM: Implementing Ridge & Lasso Regression.

PART 1(a) - Lasso-regression

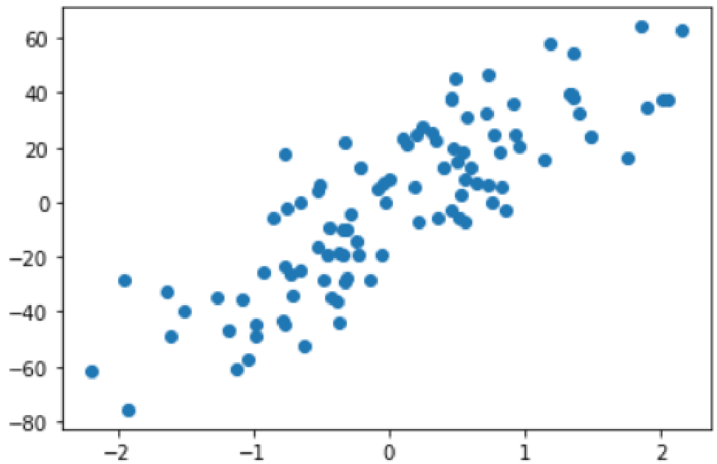
Libraries used -

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

from sklearn.linear_model import Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
```

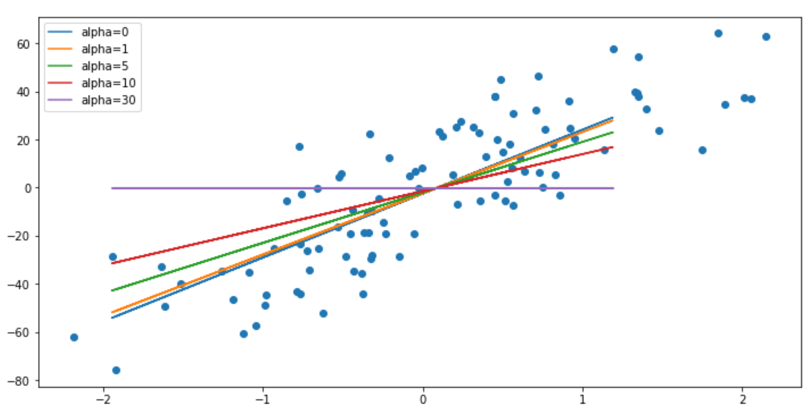
Generate a dataset using make_regression() function and fit the linear regression model-

```
X,y = make_regression(n_samples=100, n_features=1, n_informative=1,
n_targets=1,noise=20,random_state=13)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)

plt.scatter(X,y)

from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg.fit(X_train,y_train)
print(reg.coef_)
print(reg.intercept_)
```

```
[26.58287928]
-2.4682276831081924
```

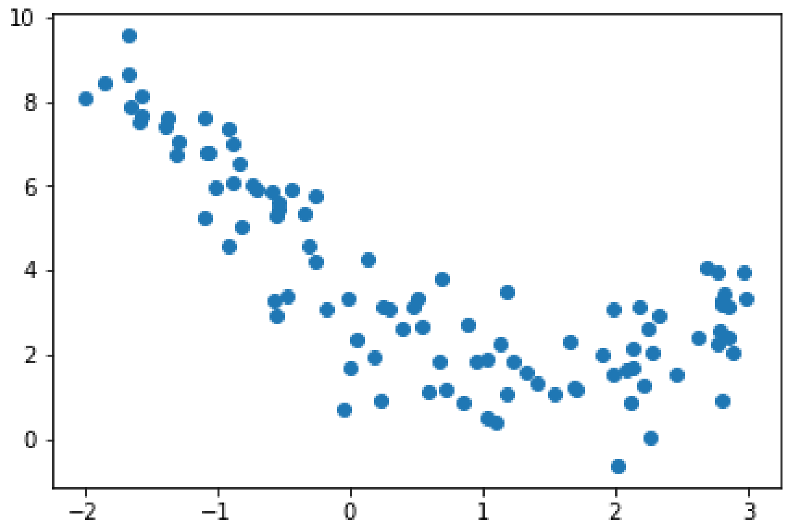


```
alphas = [0,1,5,10,30]
plt.figure(figsize=(12,6))
plt.scatter(X,y)
for i in alphas:
    L = Lasso(alpha=i)
    L.fit(X_train,y_train)
    plt.plot(X_test,L.predict(X_test),label='alpha={}'.format(i))
plt.legend()
plt.show()
```



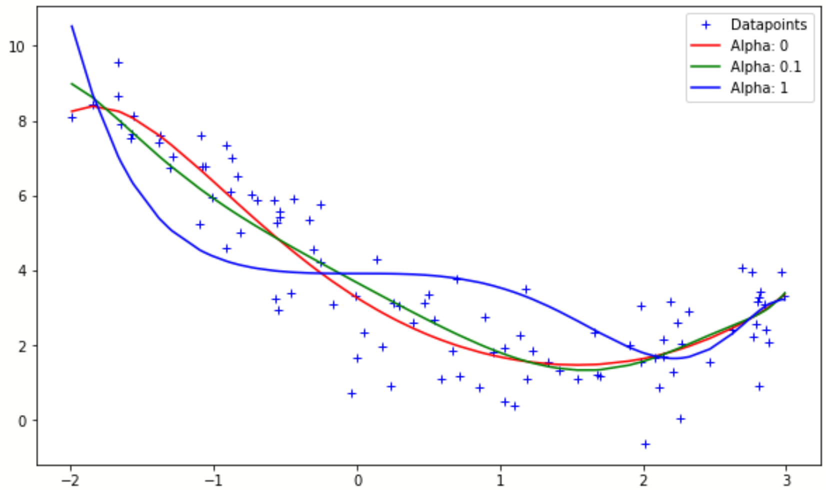

```
m = 100
x1 = 5 * np.random.rand(m, 1) - 2
x2 = 0.7 * x1 ** 2 - 2 * x1 + 3 + np.random.randn(m, 1)
```

```
plt.scatter(x1, x2)
plt.show()
```




```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge

def get_preds_lasso(x1, x2, alpha):
  model = Pipeline([
    ('poly_feats', PolynomialFeatures(degree=16)),
   ('lasso', Lasso(alpha=alpha))
  ])
  model.fit(x1, x2)
  return model.predict(x1)

alphas = [0, 0.1, 1]
cs = ['r', 'g', 'b']

plt.figure(figsize=(10, 6))
plt.plot(x1, x2, 'b+', label='Datapoints')

for alpha, c in zip(alphas, cs):
  preds = get_preds_lasso(x1, x2, alpha)
  # Plot
  plt.plot(sorted(x1[:, 0]), preds[np.argsort(x1[:, 0])], c, label='Alpha: {}'.format(alpha))

plt.legend()
```

```
plt.show()
```



PART 1 (b) - Lasso-regression-intuition

1. How are coefficients affected?

```
from sklearn.datasets import load_diabetes
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")

data = load_diabetes()

df = pd.DataFrame(data.data,columns=data.feature_names)
df['TARGET'] = data.target

df.head()
```

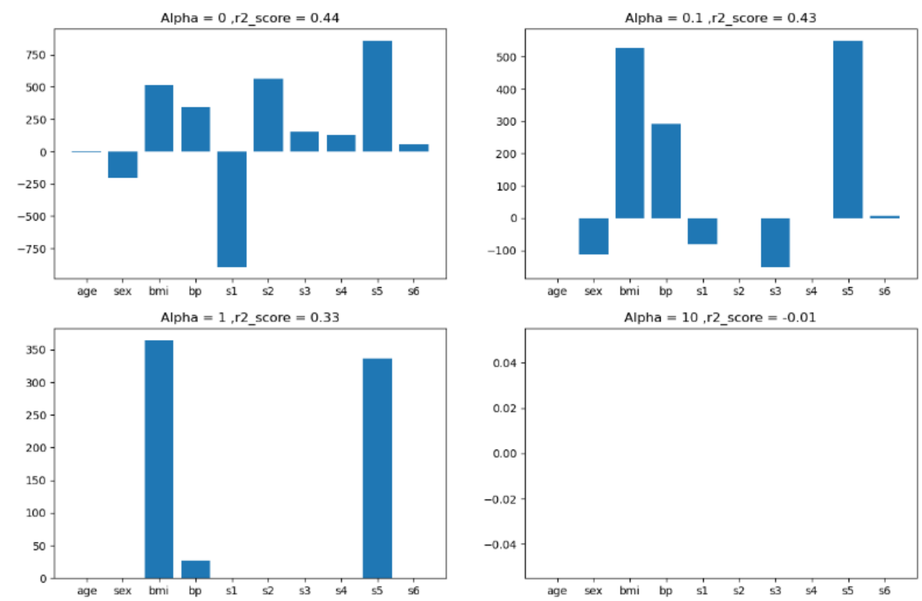
	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	TARGET
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

df.describe()

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	TARG
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	442.0000
mean	-1.444295e-18	2.543215e-18	-2.255925e-16	-4.854086e-17	-1.428596e-17	3.898811e-17	-6.028360e-18	-1.788100e-17	9.243486e-17	1.351770e-17	152.1334
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	77.0930
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123988e-01	-1.267807e-01	-1.156131e-01	-1.023071e-01	-7.639450e-02	-1.260971e-01	-1.377672e-01	25.0000
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665608e-02	-3.424784e-02	-3.035840e-02	-3.511716e-02	-3.949338e-02	-3.324559e-02	-3.317903e-02	87.0000
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670422e-03	-4.320866e-03	-3.819065e-03	-6.584468e-03	-2.592262e-03	-1.947171e-03	-1.077698e-03	140.5000
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564379e-02	2.835801e-02	2.984439e-02	2.931150e-02	3.430886e-02	3.243232e-02	2.791705e-02	211.5000
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320436e-01	1.539137e-01	1.987880e-01	1.811791e-01	1.852344e-01	1.335973e-01	1.356118e-01	346.0000

```
X_train,X_test,y_train,y_test = train_test_split(data.data,data.target,
                                    test_size=0.2,random_state=2)

coefs = []
r2_scores = []

for i in [0,0.1,1,10]:
   reg = Lasso(alpha=i)
   reg.fit(X_train,y_train)

   coefs.append(reg.coef_.tolist())
   y_pred = reg.predict(X_test)
   r2_scores.append(r2_score(y_test,y_pred))

plt.figure(figsize=(14,9))
plt.subplot(221)
plt.bar(data.feature_names,coefs[0])
plt.title('Alpha = 0 ,r2_score = {}'.format(round(r2_scores[0],2)))

plt.subplot(222)
plt.bar(data.feature_names,coefs[1])
plt.title('Alpha = 0.1 ,r2_score = {}'.format(round(r2_scores[1],2)))
```

```
plt.subplot(223)
plt.bar(data.feature_names,coefs[2])
plt.title('Alpha = 1 ,r2_score = {}'.format(round(r2_scores[2],2)))

plt.subplot(224)
plt.bar(data.feature_names,coefs[3])
plt.title('Alpha = 10 ,r2_score = {}'.format(round(r2_scores[3],2)))

plt.show()
```



2. Higher Coefficients are affected more

```
alphas = [0,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

coefs = []

for i in alphas:
    reg = Lasso(alpha=i)
    reg.fit(X_train,y_train)

    coefs.append(reg.coef_.tolist())

input_array = np.array(coefs)

coef_df = pd.DataFrame(input_array,columns=data.feature_names)
```

```
coef_df['alpha'] = alphas
coef_df.set_index('alpha')
```

alpha	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0.0000	-9.158653	-205.454322	516.693745	340.619999	-895.551989	561.220669	153.893104	126.731395	861.126997	52.421122
0.0001	-9.069064	-205.329406	516.789418	340.532379	-888.660904	555.958584	150.593655	125.450143	858.645541	52.380294
0.0010	-8.262770	-204.205364	517.650073	339.743901	-826.663603	508.617395	120.908607	113.921773	836.320753	52.012849
0.0100	-1.359721	-192.937180	526.356514	332.641101	-430.226975	191.295480	-44.034913	68.988987	688.396028	47.940616
0.1000	0.000000	-113.969928	526.744396	292.628472	-82.693681	-0.000000	-152.685338	0.000000	551.080291	7.170992
1.0000	0.000000	0.000000	363.885742	27.273163	0.000000	0.000000	-0.000000	0.000000	336.137262	0.000000
10.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000
100.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000
1000.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000
10000.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000

PART 2 - Ridge_Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split, cross_val_score
from statistics import mean
from sklearn.metrics import accuracy_score
```

```
data = pd.read_csv('kc_house_data.csv')
data.head()
```

	price	sqft_living	sqft_lot	floors	grade	sqft_above	sqft_basement	yr_built	zipcode	sqft_living15	sqft_lot15
0	221900.0	1180	5650	1.0	7	1180	0	1955	98178	1340	5650
1	538000.0	2570	7242	2.0	7	2170	400	1951	98125	1690	7639
2	180000.0	770	10000	1.0	6	770	0	1933	98028	2720	8062
3	604000.0	1960	5000	1.0	7	1050	910	1965	98136	1360	5000
4	510000.0	1680	8080	1.0	8	1680	0	1987	98074	1800	7503

```
dropColumns = [ 'zipcode']
data = data.drop(dropColumns, axis = 1)

y = data['price']
X = data.drop('price', axis = 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

# Building and fitting the Linear Regression model
linearModel = LinearRegression()
linearModel.fit(X_train, y_train)

# Evaluating the Linear Regression model
print(linearModel.score(X_test, y_test))
```

```
0.6158772627644511
```

```
# List to maintain the different cross-validation scores
cross_val_scores_ridge = []

# List to maintain the different values of alpha
alpha = []

# Loop to compute the different values of cross-validation scores
for i in range(1, 9):
   ridgeModel = Ridge(alpha = i * 0.25)
   ridgeModel.fit(X_train, y_train)
   scores = cross_val_score(ridgeModel, X, y, cv = 10)
   avg_cross_val_score = mean(scores)*100
   cross_val_scores_ridge.append(avg_cross_val_score)
   alpha.append(i * 0.25)

# Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_ridge[i]))
```

```
0.25 : 60.26529996408061
0.5 : 60.265291469441436
0.75 : 60.26528296424786
1.0 : 60.26527444850186
1.25 : 60.26526592220542
1.5 : 60.26525738536059
1.75 : 60.26524883796931
2.0 : 60.26524028003363
```

```
# Building and fitting the Ridge Regression model
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(X_train, y_train)

# Evaluating the Ridge Regression model
print(ridgeModelChosen.score(X_test, y_test))
```

```
0.6158752081631746
```

PRACTICAL NO: 5

AIM: For a given set of data implement Logistic Regression Algorithm.

```
import pandas as pd
df=pd.read_excel('Insurance.xlsx')
df.head()
```

	Age	Claim
0	22.0	No
1	25.0	No
2	47.0	Yes
3	52.0	No
4	46.0	Yes

data=pd.get_dummies(df,columns = ['Claim'])
print(data)

```
     Age  Claim_No  Claim_Yes
0   22.0         1          0
1   25.0         1          0
2   47.0         0          1
3   52.0         1          0
4   46.0         0          1
5   56.0         0          1
6   55.0         1          0
7   60.0         0          1
8   62.0         0          1
9   61.0         0          1
10  18.0         1          0
```

df['Claim'].value_counts()

```
No     13
Yes    13
Name: Claim, dtype: int64
```

data.isnull().sum()

```
Age          0
Claim_No     0
Claim_Yes    0
dtype: int64
```

```
x=df.iloc[:,0:1]
x
```

	Age
0	22.0
1	25.0
2	47.0
3	52.0
4	46.0
5	56.0
6	55.0
7	60.0
8	62.0
9	61.0
10	18.0

```
y=df.iloc[:,-1]
print(y)
```

```
0     0
1     0
2     1
3     0
4     1
5     1
6     0
7     1
8     1
9     1
10    0
```

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['Claim'] = label_encoder.fit_transform(df['Claim'])
df['Claim'].value_counts()
```

```
0    13
1    13
Name: Claim, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42
x_train.shape
```

```
(18, 1)
```

```
Data.shape
```

```
(26, 3)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr
```

```
LogisticRegression()
```

```
lr.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred=lr.predict(x_test)
y_pred
```

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

```
y_test
```

```
19    0
5     1
14    1
7     1
1     0
2     1
11    0
23    1
Name: Claim, dtype: int32
```

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)

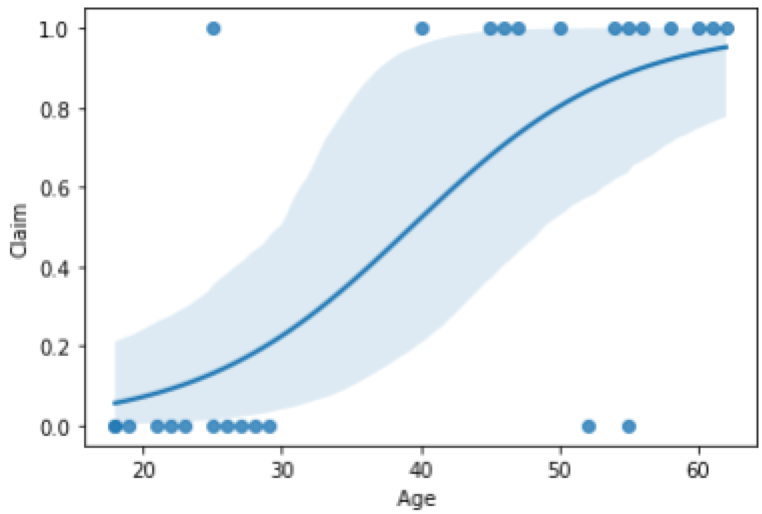
```
array([[3, 0],
       [0, 5]], dtype=int64)
```

from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)

```
1.0
```

import seaborn as sns
sns.regplot(x=x,y=y,data=df,logistic=True)

```
<AxesSubplot:xlabel='Age ', ylabel='Claim'>
```



PRACTICAL NO: 6

AIM: Implementing Decision Tree Algorithm.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.datasets import load_iris

df = load_iris()
```

```
df.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
```

```
x = df.data
y = df.target
```

<u>CHECKING NULL VALUES (NAN)</u>

```
np.isnan(y).sum()
```

```
0
```

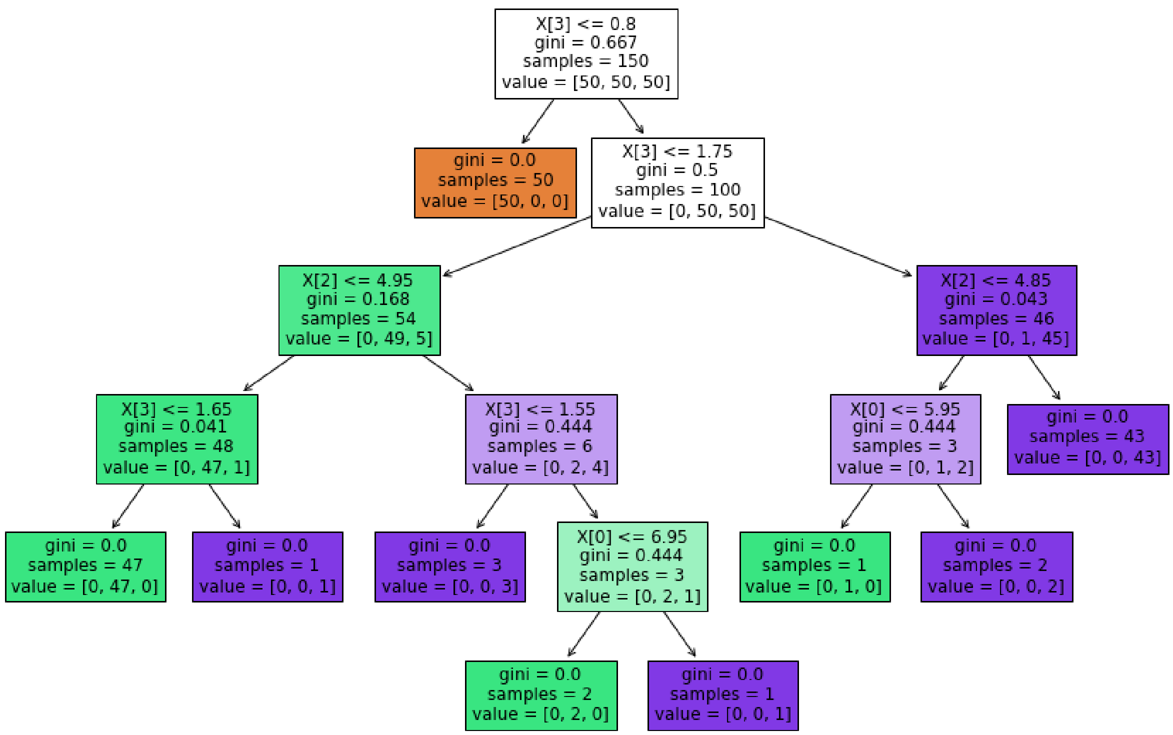
```
np.isnan(x).sum()
```

```
0
```

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x,y)
```

```
DecisionTreeClassifier()
```

```
from sklearn import tree
plt.figure(figsize = (15,10))
tree.plot_tree(dt,filled = True)
```



PRACTICAL NO: 7

AIM: Perform Hyperparameter Tuning on Random Forest Algorithm.

Importing different libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

df = pd.read_csv("heart.csv")


df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
df.shape
```

```
(303, 14)
```

```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
print(X_train.shape)
print(X_test.shape)
```

```
(242, 13)
(61, 13)
```

```
rf = RandomForestClassifier()
gb = GradientBoostingClassifier()
svc = SVC()
lr = LogisticRegression()
```

```
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.8360655737704918
```

```
gb.fit(X_train,y_train)
y_pred = gb.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.7704918032786885
```

```
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.7049180327868853
```

```
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.8852459016393442
```

```
rf = RandomForestClassifier(max_samples=0.75,random_state=42)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.9016393442622951
```

from sklearn.model_selection import cross_val_score
np.mean(cross_val_score(RandomForestClassifier(max_samples=0.75),X,y,cv=10,scoring='accuracy'))

```
0.8412903225806451
```

GridSearchCV   #greedy method

Random serach   # randomly select

 # Number of trees in random forest
n_estimators = [20,60,100,120]

 # Number of features to consider at every split
max_features = [0.2,0.6,1.0]

 # Maximum number of levels in tree
max_depth = [2,8,None]

 # Number of samples
max_samples = [0.5,0.75,1.0]

 # 108 diff random forest train

```
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
              'max_samples':max_samples
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_depth': [2, 8, None], 'max_samples': [0.5, 0.75,
```

rf = RandomForestClassifier()

```
from sklearn.model_selection import GridSearchCV

rf_grid = GridSearchCV(estimator = rf,
                       param_grid = param_grid,
                       cv = 5,
                       verbose=2,
                       n_jobs = -1)
```

```
rf_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [2, 8, None],
                         'max_features': [0.2, 0.6, 1.0],
                         'max_samples': [0.5, 0.75, 1.0],
                         'n_estimators': [20, 60, 100, 120]},
             verbose=2)
```

```
rf_grid.best_params_
```

```
{'max_depth': None,
 'max_features': 0.2,
 'max_samples': 0.75,
 'n_estimators': 20}
```

```
rf_grid.best_score_
```

```
0.8387755102040815
```

```
# Number of trees in random forest
n_estimators = [20,60,100,120]

# Number of features to consider at every split
max_features = [0.2,0.6,1.0]
```

```
# Maximum number of levels in tree
max_depth = [2,8,None]

# Number of samples
max_samples = [0.5,0.75,1.0]

# Bootstrap samples
bootstrap = [True,False]

# Minimum number of samples required to split a node
min_samples_split = [2, 5]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]


param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
             'max_samples':max_samples,
             'bootstrap':bootstrap,
             'min_samples_split':min_samples_split,
             'min_samples_leaf':min_samples_leaf
            }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_depth': [2, 8, None], 'max_samples': [0.5, 0.75
```

```
from sklearn.model_selection import RandomizedSearchCV

rf_grid = RandomizedSearchCV(estimator = rf,
                    param_distributions = param_grid,
                    cv = 5,
                    verbose=2,
                    n_jobs = -1)


rf_grid.fit(X_train,y_train)
```

rf_grid.best_params_

```
{'n_estimators': 120,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_samples': 0.75,
 'max_features': 0.2,
 'max_depth': None,
 'bootstrap': True}
```

Rf_grid.best_score_

```
0.8178571428571428
```

PRACTICAL NO: 8

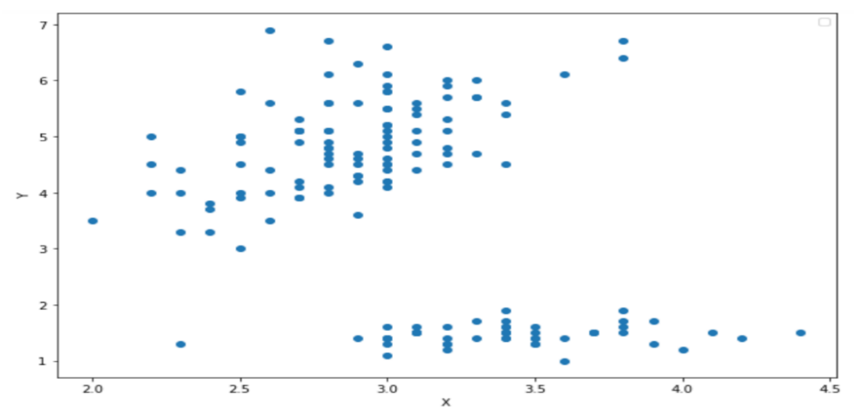
AIM: Implementing K-Means Clustering Algorithm.

A] For cluster.csv data

```
import pandas as pd
import numpy as np

df=pd.read_csv("cluster.csv")
df.head()
```

	X	Y
0	3.5	1.4
1	3.0	1.4
2	3.2	1.3
3	3.1	1.5
4	3.6	1.4

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,7))
plt.scatter(df['X'],df['Y'])
plt.ylabel("Y")
plt.xlabel("X")
plt.show()
```

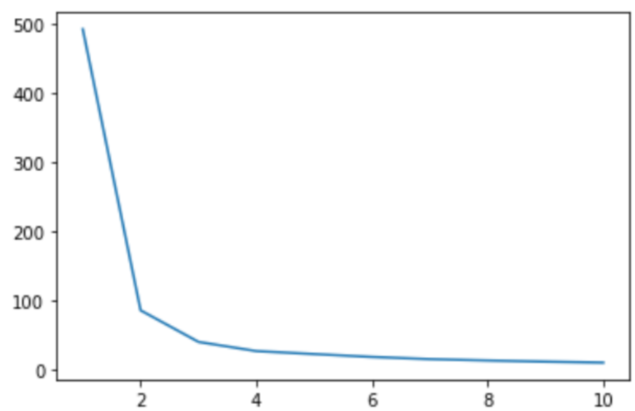



```
from sklearn.cluster import KMeans
#within cluster sum of squares WcSS/Elbow method
WCSS=[ ]
for i in range (1,11):
   km=KMeans(n_clusters=i)
   km.fit_predict(df)
   WCSS.append(km.inertia_)

WCSS
```

```
[491.8763333333335,
 86.35692216280445,
 40.80747409220729,
 27.55509523809523,
 23.252325396825395,
 19.127455003931395,
 15.9417413793l034,
 13.967600636910976,
 12.365316017316017,
 10.859924830071892]
```

plt.plot(range(1,11),WCSS)

```
[<matplotlib.lines.Line2D at 0x2802031b2b0>]
```



x=df.iloc[:,:].values
km = KMeans(n_clusters=2)
y_means=km.fit_predict(x)
y_means

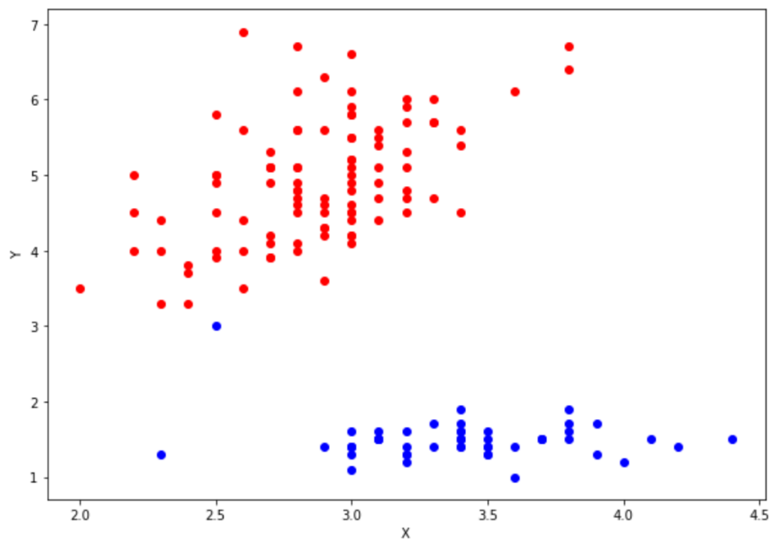
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    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

x[y_means==3,1]

plt.figure(figsize=(10,7))

```
plt.scatter(x[y_means==0,0],x[y_means==0,1],color="r")
plt.scatter(x[y_means==1,0],x[y_means==1,1],color="b")
plt.xlabel("X")
plt.ylabel("Y")
```

```
Text(0, 0.5, 'Y')
```



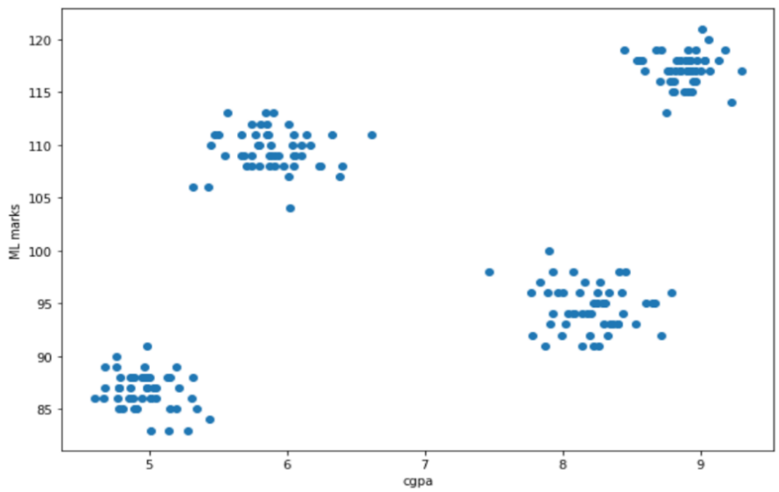
<u>B] For student_clustering.csv data</u>

```
import pandas as pd
import numpy as np

df=pd.read_csv("student_clustering.csv")
df.head()
```

	cgpa	ML
0	5.13	88
1	5.90	113
2	8.36	93
3	8.27	97
4	5.45	110

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,7))
```

```
plt.scatter(df['cgpa'],df['ML'])
plt.ylabel("ML marks")
plt.xlabel("cgpa")
plt.legend()
plt.show()
```




```
from sklearn.cluster import KMeans
```

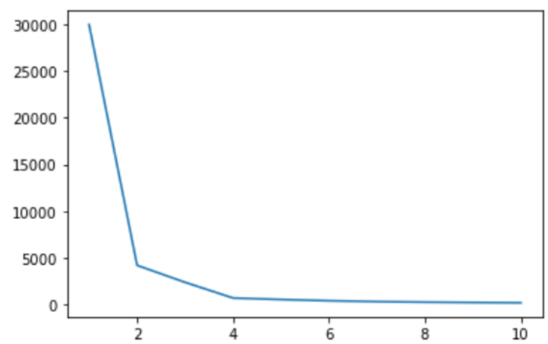
#within cluster sum of squares WcSS/Elbow method

```
WCSS=[]
for i in range (1,11):
   km=KMeans(n_clusters=i)
   km.fit_predict(df)
   WCSS.append(km.inertia_)

WCSS
```

```
[29957.898287999997,
 4184.141269999999,
 2362.7133490000015,
 681.9696600000001,
 523.7131894763968,
 388.8524026875981,
 295.4391895943191,
 240.7551346189791,
 203.38836189856835,
 172.4919501547381]
```

plt.plot(range(1,11),WCSS)

```
[<matplotlib.lines.Line2D at 0x1915d0b8700>]
```



x=df.iloc[:,:].values

km = KMeans(n_clusters=4)
y_means=km.fit_predict(x)
y_means

```
array([0, 2, 3, 3, 2, 2, 3, 1, 2, 3, 0, 2, 3, 0, 2, 3, 2, 3, 2, 2, 3, 0,
       3, 0, 0, 3, 0, 1, 3, 2, 1, 2, 1, 2, 3, 3, 1, 2, 0, 2, 0, 3, 3, 0,
       1, 1, 3, 2, 1, 2, 0, 0, 1, 3, 1, 2, 2, 1, 2, 1, 2, 3, 3, 1, 0, 1,
       3, 0, 2, 3, 2, 1, 3, 0, 2, 1, 2, 1, 0, 3, 3, 1, 2, 0, 1, 0, 1, 2,
       1, 2, 1, 1, 3, 0, 3, 3, 1, 3, 0, 1, 2, 0, 0, 1, 0, 0, 3, 0, 1, 1,
       3, 1, 2, 2, 3, 1, 3, 2, 1, 0, 0, 2, 3, 1, 3, 0, 3, 2, 0, 3, 3, 2,
       0, 0, 2, 1, 2, 0, 3, 3, 3, 0, 2, 0, 0, 1, 0, 1, 2, 0, 1, 0, 1, 1,
       0, 3, 2, 1, 2, 3, 0, 1, 2, 3, 1, 0, 2, 0, 0, 1, 1, 2, 1, 0, 0, 3,
       1, 2, 0, 1, 1, 2, 2, 2, 3, 0, 3, 3, 1, 2, 3, 3, 0, 0, 3, 0, 1, 2,
       2, 1])
```

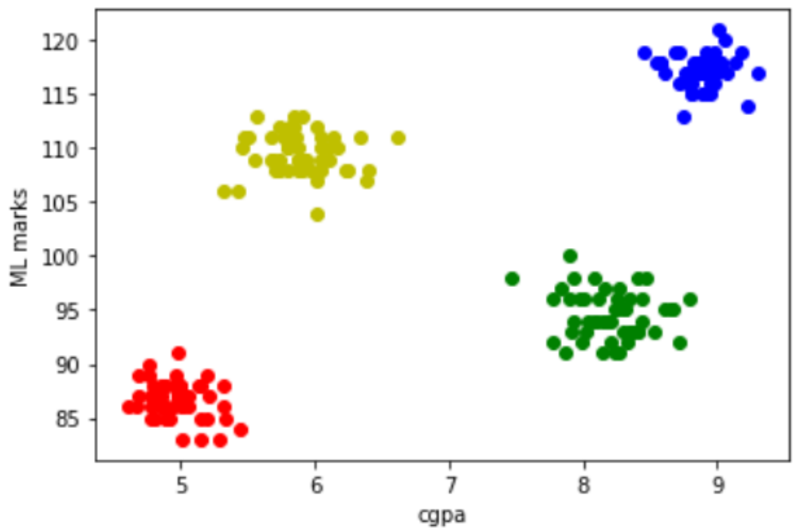
x[y_means==3,1]

```
array([ 93.,  97.,  98.,  94.,  97.,  95.,  91.,  98.,  92.,  98.,  94.,
        96.,  96.,  96.,  93.,  94.,  96.,  96.,  95.,  93.,  95.,  94.,
        92.,  91.,  92.,  95.,  94.,  95.,  92.,  94.,  91.,  95.,  93.,
        97.,  98.,  96.,  93., 100.,  96.,  94.,  95.,  93.,  92.,  98.,
        96.,  93.,  91.,  93.,  94.,  96.])
```

plt.scatter(x[y_means==0,0],x[y_means==0,1],color="r")
plt.scatter(x[y_means==1,0],x[y_means==1,1],color="b")
plt.scatter(x[y_means==2,0],x[y_means==2,1],color="y")
plt.scatter(x[y_means==3,0],x[y_means==3,1],color="g")
plt.xlabel("cgpa")

plt.ylabel("ML marks")

```
Text(0, 0.5, 'ML marks')
```



PRACTICAL NO: 9
AIM: Implementing Hierarchical Clustering Algorithm.

import pandas as pd
import numpy as np
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

df=pd.read_excel('segmented_customers.xlsx')
df.head()

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	1.0	1.0	19.0	15.0	39.0	3.0
1	2.0	1.0	21.0	15.0	81.0	4.0
2	3.0	0.0	20.0	16.0	6.0	3.0
3	4.0	0.0	23.0	16.0	77.0	4.0
4	5.0	0.0	31.0	17.0	40.0	3.0

df1 = df.drop(['cluster'],axis = 1)
df1.head()

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1.0	1.0	19.0	15.0	39.0
1	2.0	1.0	21.0	15.0	81.0
2	3.0	0.0	20.0	16.0	6.0
3	4.0	0.0	23.0	16.0	77.0
4	5.0	0.0	31.0	17.0	40.0

x = df.iloc[:,3:5]
x

	Annual Income (k$)	Spending Score (1-100)
0	15.0	39.0
1	15.0	81.0
2	16.0	6.0
3	16.0	77.0
4	17.0	40.0
...	...	...
195	120.0	79.0
196	126.0	28.0
197	126.0	74.0
198	137.0	18.0
199	137.0	83.0

200 rows × 2 columns

x=x.values
x

```
array([[ 15.,  39.],
       [ 15.,  81.],
       [ 16.,   6.],
       [ 16.,  77.],
       [ 17.,  40.],
       [ 17.,  76.],
       [ 18.,   6.],
       [ 18.,  94.],
       [ 19.,   3.],
       [ 19.,  72.],
       [ 19.,  14.],
       [ 19.,  99.],
       [ 20.,  15.],
       [ 20.,  77.],
       [ 20.,  13.],
       [ 20.,  79.],
       [ 21.,  35.],
       [ 21.,  66.],
```

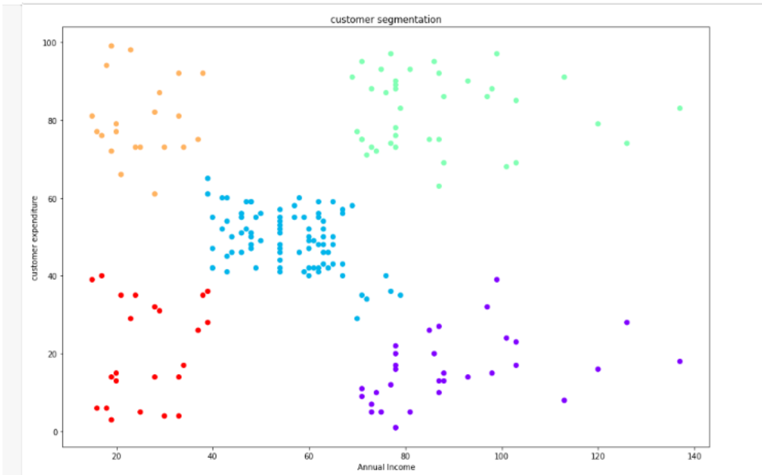
cluster = AgglomerativeClustering(n_clusters = 5,linkage = 'ward',affinity='euclidean')
cluster

```
AgglomerativeClustering(n_clusters=5)
```

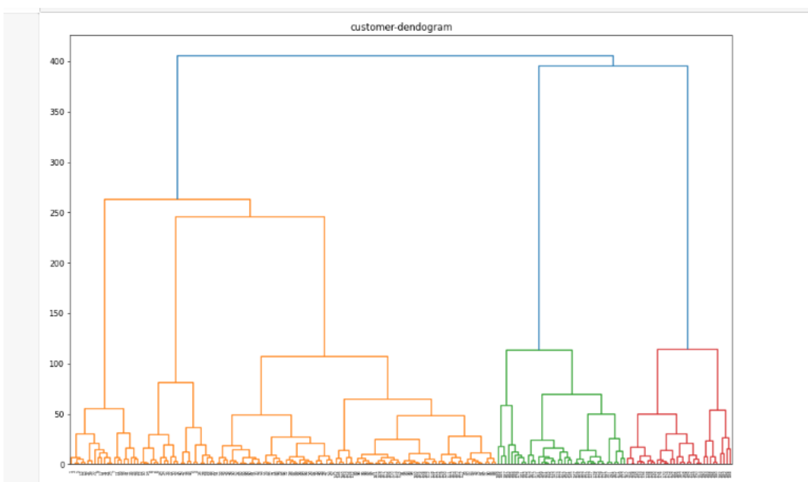
labels = cluster.fit_predict(x)
labels

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 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, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int64)
```

plt.scatter(x[:,0],x[:,1],c=cluster.labels_,cmap='rainbow')
plt.title('customer segmentation')
plt.xlabel('Annual Income')
plt.ylabel('customer expenditure');




```
plt.title('customer-dendogram')
dendo = sch.dendrogram(sch.linkage(x,method='ward'))
```



<u>PRACTICAL NO: 10</u>

<u>AIM: Implementing Density Based Spatial Clustering Of Application with Noise.</u>

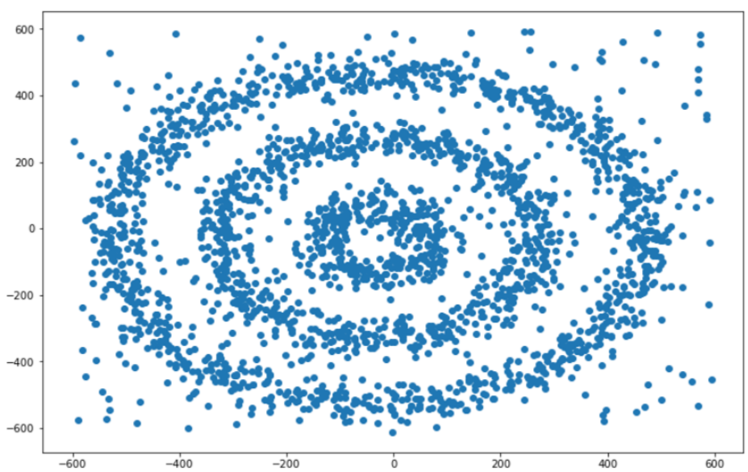
```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('DBSCAN.csv')
```

df.head()

	sr.no	A	B
0	0	484.891555	-31.006357
1	1	489.391178	21.973916
2	2	462.886575	-27.599889
3	3	517.218479	5.588090
4	4	455.669049	1.982181

plt.figure(figsize=(12,8),facecolor='white')

plt.scatter(df['A'], df['B'])



x = df.iloc[:,1:]

x.head()

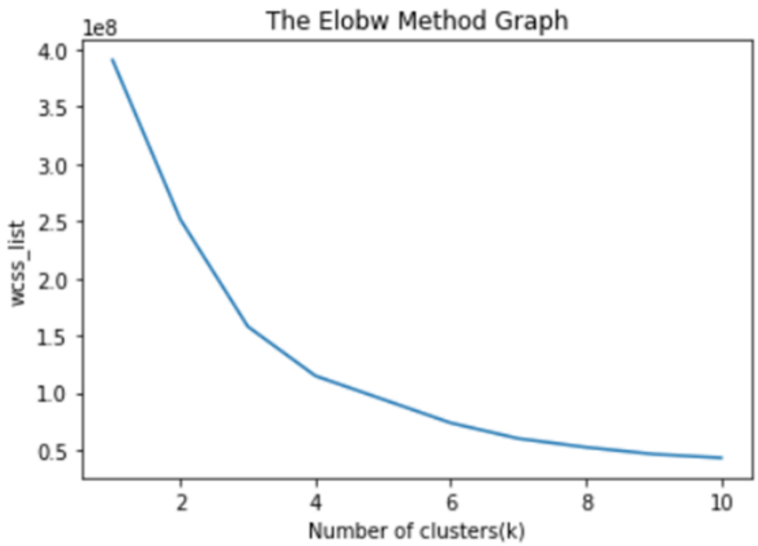
	A	B
0	484.891555	-31.006357
1	489.391178	21.973916
2	462.886575	-27.599889
3	517.218479	5.588090
4	455.669049	1.982181

Finding optimal number of clusters using the elbow method :

```
from sklearn.cluster import KMeans

wcss_list= []  #Initializing the list for the values of WCSS
```

Using for loop for iterations from 1 to 10:

```
for i in range(1, 11):

        kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)

        kmeans.fit(x)

  wcss_list.append(kmeans.inertia_)

plt.figure(figsize=(10,6),facecolor='white')

plt.plot(range(1, 11), wcss_list)

plt.title('The Elobw Method Graph')

plt.xlabel('Number of clusters(k)')

plt.ylabel('wcss_list')

plt.show()                         # by observing in the elbow graph we will make 4 clusters
```



X = df.iloc[:,1:].values

km = KMeans(n_clusters = 4) ***<u># No of clusters = 4</u>***

<u>Using k-means algorithm:</u>

y_means = km.fit_predict(X)

y_means

```
array([2, 2, 2, ..., 2, 0, 1])
```

<u>#Feature1 = A = 0 , #Feature2 = B = 1</u>

X[y_means == 0,1] ***<u># it will give the results for feature1 & cluster1</u>***

```
array([-550.3596874 , -523.206023  , -578.0535956 , -514.219064  ,
       -505.8619101 , -515.6353626 , -546.3602917 , -527.8923182 ,
       -516.2202799 , -540.3857696 , -541.3575639 , -501.5387424 ,
       -453.0961161 , -524.3746371 , -469.7777355 , -493.7391435 ,
       -512.2241915 , -546.5257019 , -530.0617558 , -543.5080177 ,
       -501.1829384 , -485.4273778 , -535.4545201 , -585.5125659 ,
       -533.0209036 , -510.2764245 , -473.3905872 , -511.3433221 ,
```

....

```
-197.          , -461.          , -468.          , -151.          ,
-138.          ,  -98.          , -275.          , -421.          ,
-108.          , -439.          , -137.          , -387.          ,
-468.          , -228.          , -519.          , -420.          ,
-377.          , -580.          , -535.          ])
```

plt.figure(figsize=(12,8),facecolor='white')

plt.scatter(X[y_means == 0,0],X[y_means == 0,1],color = 'blue')

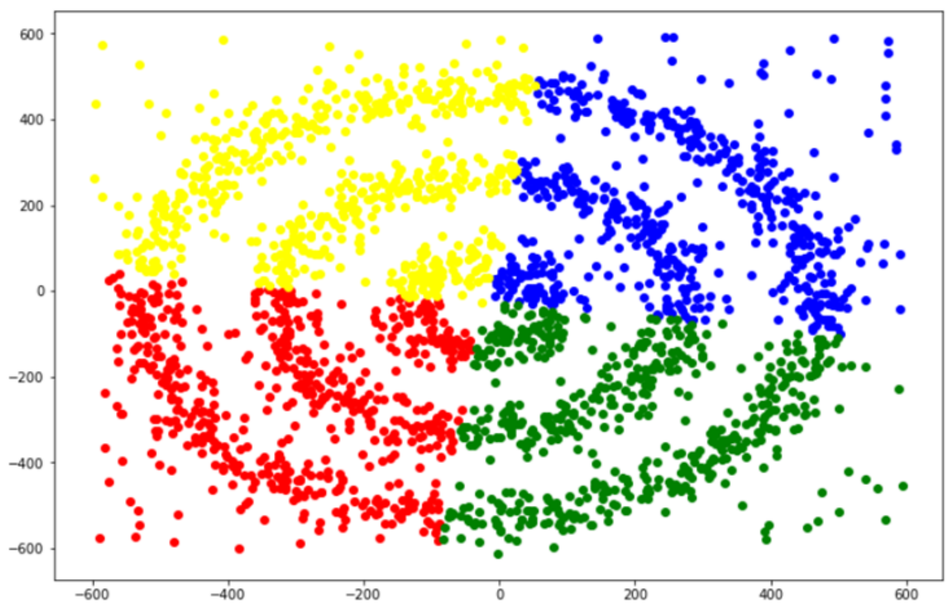
plt.scatter(X[y_means == 1,0],X[y_means == 1,1],color = 'red')

plt.scatter(X[y_means == 2,0],X[y_means == 2,1],color = 'green')

plt.scatter(X[y_means == 3,0],X[y_means == 3,1],color = 'yellow')

plt.figure(figsize=(12,8),facecolor='white')

plt.show()



<u>Using agglomerative clustering:</u>

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5,affinity='euclidean', linkage='ward')

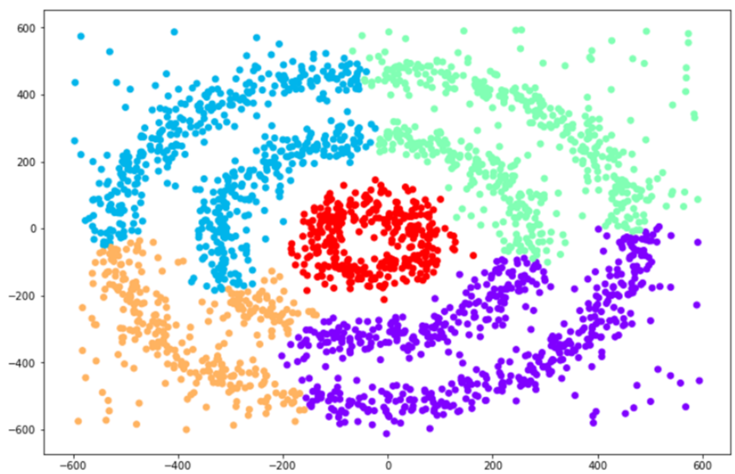
labels_ = cluster.fit_predict(X)

labels_

```
array([0, 2, 0, ..., 2, 0, 2], dtype=int64)
```

plt.figure(figsize=(12,8),facecolor='white')

plt.scatter(X[:,0],X[:,1], c=cluster.labels_, cmap='rainbow')



Using DBSCAN algorithm:

from sklearn.cluster import DBSCAN

DB =DBSCAN(eps=30,min_samples=5)

DB.fit(df[['A','B']])

df['DBSCAN_labels']=DB.labels_

plt.figure(figsize=(12,8),facecolor='white')

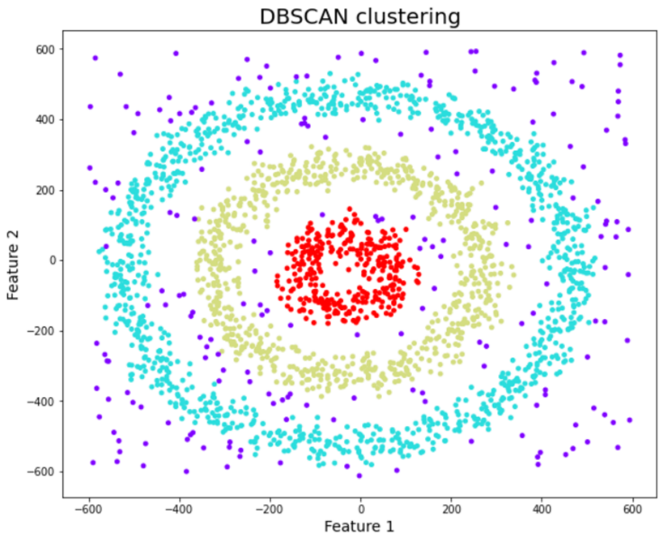
plt.scatter(df['A'],df['B'],c=df['DBSCAN_labels'],cmap='rainbow',s=15)

plt.title('DBSCAN clustering',fontsize=20)

plt.xlabel('Feature 1',fontsize=14)

plt.ylabel('Feature 2',fontsize=14)

```
plt.show()
```


DBSCAN clustering
Feature 2
Feature 1
600
400
200
0
−200
−400
−600
−600
−400
−200
0
200
400
600

PRACTICAL NO: 11
AIM: Apply ensemble learning Boosting Technique.

from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
 # Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

iris = datasets.load_iris()

x = iris.data
x

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3]
```

y = iris.target
y

```
array([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])
```

```
#Split the dataset into a training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and
30% test

#Create AdaBoost classifier object
abc = AdaBoostClassifier(n_estimators=50,
                learning_rate=1)

#Train Adaboost Classifer
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)
y_pred
```

```
array([1, 0, 1, 1, 2, 1, 2, 0, 0, 2, 2, 0, 0, 1, 0, 1, 0, 2, 0, 2, 2,
1,
       1, 1, 1, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 1, 1, 1, 1, 0, 0, 1, 0,
0,
       2])
```

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

```
Accuracy: 0.9333333333333333
```

```
# Load libraries
from sklearn.ensemble import AdaBoostClassifier
```

```
# Import Support Vector Classifier
from sklearn.svm import SVC
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
svc=SVC(probability=True, kernel='linear')

# Create adaboost classifer object
abc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)

# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)
y_pred
```

```
array([1, 0, 2, 1, 2, 1, 2, 0, 0, 2, 2, 0, 0, 2, 0, 1, 0, 2, 0, 2, 2,
1,
       1, 1, 1, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 1, 1, 1, 1, 0, 0, 1, 0,
0,
       2])
```

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

```
Accuracy: 0.9777777777777777
```