COURSE OUTCOME 2

PROGRAM - 5

AIM:

Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

DATASET:

sonar_csv.csv

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

```
df = pd.read_csv('/content/sonar_csv.csv')
```

```
df.head()
```

	attribute_1	attribute_2	attribute_3	attribute_4	attribute_5	attribute_6	attribu
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0

```
from sklearn .preprocessing import StandardScaler
sc = StandardScaler()
```

```
sc.fit(df.drop('Class',axis = 1))
```

```
StandardScaler()
```

```
s = sc.transform(df.drop('Class',axis = 1))
s
```

```
array([[-0.39955135, -0.04064823, -0.02692565, ...,  0.06987027,
         0.17167808, -0.65894689],
```

```
        0.17167808, -0.65894689],
       [ 0.70353822,  0.42163039,  1.05561832, ..., -0.47240644,
        -0.44455424, -0.41985233],
       [-0.12922901,  0.60106749,  1.72340448, ...,  1.30935987,
         0.25276128,  0.25758223],
       ...,
       [ 1.00438083,  0.16007801, -0.67384349, ...,  0.90652575,
        -0.03913824, -0.67887143],
       [ 0.04953255, -0.09539176,  0.13480381, ..., -0.00759783,
        -0.70402047, -0.34015415],
       [-0.13794908, -0.06497869, -0.78861924, ..., -0.6738235 ,
        -0.29860448,  0.99479044]])
```

```
a = pd.DataFrame(s,columns = df.columns[:-1] )
a
```

	attribute_1	attribute_2	attribute_3	attribute_4	attribute_5	attribute_6	attr
0	-0.399551	-0.040648	-0.026926	-0.715105	0.364456	-0.101253	0
1	0.703538	0.421630	1.055618	0.323330	0.777676	2.607217	1
2	-0.129229	0.601067	1.723404	1.172176	0.400545	2.093337	1
3	-0.835555	-0.648910	0.481740	-0.719414	-0.987079	-1.149364	-0
4	2.050790	0.856537	0.111327	-0.312227	-0.292365	-0.672796	-0
...	...	...	...	...	...	...	
203	-0.456232	-0.116681	-0.705146	-0.779738	-0.647842	0.990954	1
204	0.136733	-0.861801	-0.366036	0.054026	0.014392	-0.148740	-0
205	1.004381	0.160078	-0.673843	-0.531979	-0.723629	0.212502	0
206	0.049533	-0.095392	0.134804	0.148821	-1.055648	0.522865	0
207	-0.137949	-0.064979	-0.788619	-0.575067	-0.970839	-1.200244	-0

208 rows × 60 columns

TRAINING AND TESTING:

```
from sklearn.model_selection import train_test_split
```

```
x = a
y = df['Class']
```

```
y
```

```
0      Rock
1      Rock
2      Rock
```

```
2       Rock
3       Rock
4       Rock
        ...
203     Mine
204     Mine
205     Mine
206     Mine
207     Mine
Name: Class, Length: 208, dtype: object
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .3,random_state=42)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
c = KNeighborsClassifier(n_neighbors=1)
```

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

```
KNeighborsClassifier(n_neighbors=1)
```

```
pred=c.predict(x_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

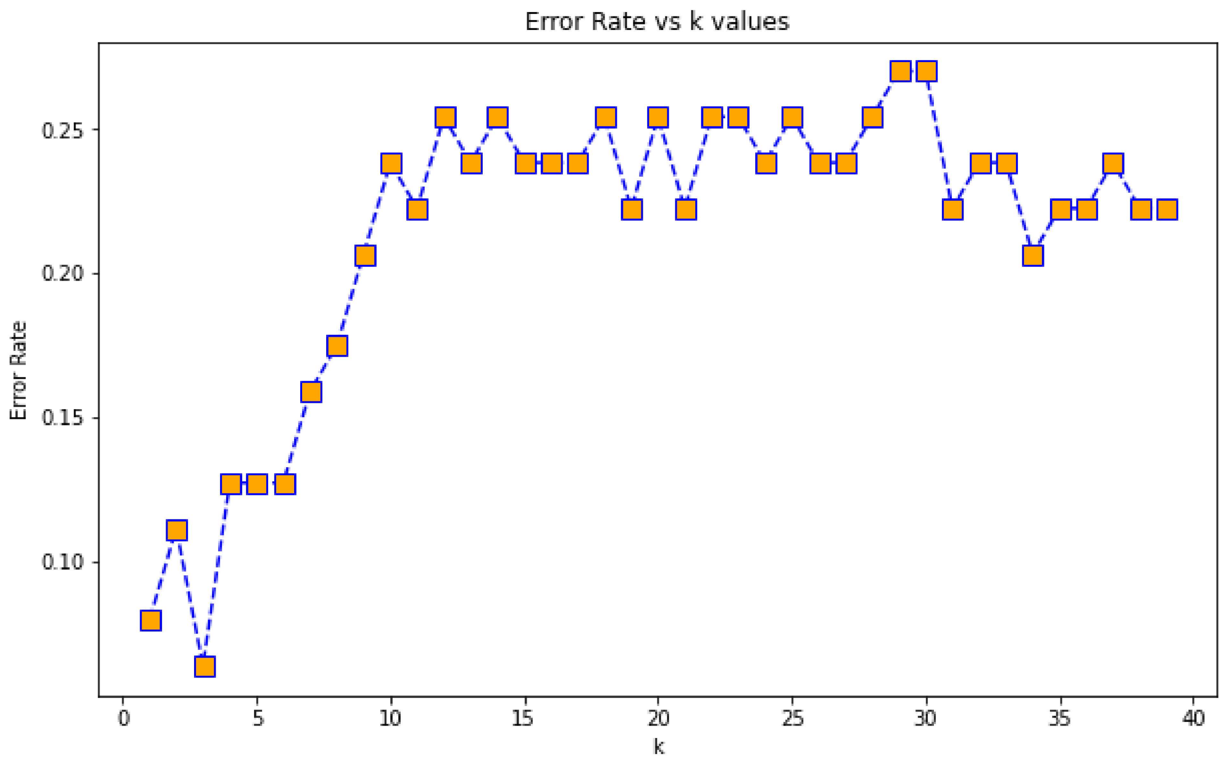
```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[32  3]
 [ 2 26]]
              precision    recall  f1-score   support

        Mine       0.94      0.91      0.93        35
        Rock       0.90      0.93      0.91        28

    accuracy                           0.92        63
   macro avg       0.92      0.92      0.92        63
weighted avg       0.92      0.92      0.92        63
```

```
#analyzing better k value through iterations.
error_rate=[]

for i in range(1,40):
  k = KNeighborsClassifier(n_neighbors=i)
  k.fit(x_train,y_train)
  pred_i= k.predict(x_test)
  error_rate.append(np.mean(pred_i!= y_test))
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='s',markerfacecolor='o
plt.title('Error Rate vs k values')
plt.xlabel('k')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



RESULT:

The program executed successfully and obtained the output.

PROGRAM - 6

AIM:

Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm

DATASET:

Iris.csv

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

```
dataset=pd.read_csv('/content/Iris (1).csv')
```

```
dataset.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
```

```
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
from sklearn.preprocessing import StandardScaler


scalar=StandardScaler()


scalar.fit(dataset.drop('Species',axis=1))
```

```
StandardScaler()
```

```
x=scalar.transform(dataset.drop('Species',axis=1))




x
```

```
array([[-1.72054204e+00, -9.00681170e-01,  1.03205722e+00,
        -1.34127240e+00, -1.31297673e+00],
       [-1.69744751e+00, -1.14301691e+00, -1.24957601e-01,
        -1.34127240e+00, -1.31297673e+00],
       [-1.67435299e+00, -1.38535265e+00,  3.37848329e-01,
        -1.39813811e+00, -1.31297673e+00],
       [-1.65125846e+00, -1.50652052e+00,  1.06445364e-01,
        -1.28440670e+00, -1.31297673e+00],
       [-1.62816394e+00, -1.02184904e+00,  1.26346019e+00,
        -1.34127240e+00, -1.31297673e+00],
       [-1.60506942e+00, -5.37177559e-01,  1.95766909e+00,
        -1.17067529e+00, -1.05003079e+00],
       [-1.58197489e+00, -1.50652052e+00,  8.00654259e-01,
        -1.34127240e+00, -1.18150376e+00],
       [-1.55888037e+00, -1.02184904e+00,  8.00654259e-01,
        -1.28440670e+00, -1.31297673e+00],
       [-1.53578584e+00, -1.74885626e+00, -3.56360566e-01,
        -1.34127240e+00, -1.31297673e+00],
       [-1.51269132e+00, -1.14301691e+00,  1.06445364e-01,
        -1.28440670e+00, -1.44444970e+00],
       [-1.48959680e+00, -5.37177559e-01,  1.49486315e+00,
        -1.28440670e+00, -1.31297673e+00],
       [-1.46650227e+00, -1.26418478e+00,  8.00654259e-01,
        -1.22754100e+00, -1.31297673e+00],
       [-1.44340775e+00, -1.26418478e+00, -1.24957601e-01,
        -1.34127240e+00, -1.44444970e+00],
       [-1.42031323e+00, -1.87002413e+00, -1.24957601e-01,
        -1.51186952e+00, -1.44444970e+00],
       [-1.39721870e+00, -5.25060772e-02,  2.18907205e+00,
        -1.45500381e+00, -1.31297673e+00],
       [-1.37412418e+00, -1.73673948e-01,  3.11468391e+00,
        -1.28440670e+00, -1.05003079e+00],
       [-1.35102965e+00, -5.37177559e-01,  1.95766909e+00,
        -1.39813811e+00, -1.05003079e+00],
       [-1.32793513e+00, -9.00681170e-01,  1.03205722e+00,
```

```
       -1.34127240e+00, -1.18150376e+00],
      [-1.30484061e+00, -1.73673948e-01,  1.72626612e+00,
       -1.17067529e+00, -1.18150376e+00],
      [-1.28174608e+00, -9.00681170e-01,  1.72626612e+00,
       -1.28440670e+00, -1.18150376e+00],
      [-1.25865156e+00, -5.37177559e-01,  8.00654259e-01,
       -1.17067529e+00, -1.31297673e+00],
      [-1.23555703e+00, -9.00681170e-01,  1.49486315e+00,
       -1.28440670e+00, -1.05003079e+00],
      [-1.21246251e+00, -1.50652052e+00,  1.26346019e+00,
       -1.56873522e+00, -1.31297673e+00],
      [-1.18936799e+00, -9.00681170e-01,  5.69251294e-01,
       -1.17067529e+00, -9.18557817e-01],
      [-1.16627346e+00, -1.26418478e+00,  8.00654259e-01,
       -1.05694388e+00, -1.31297673e+00],
      [-1.14317894e+00, -1.02184904e+00, -1.24957601e-01,
       -1.22754100e+00, -1.31297673e+00],
      [-1.12008441e+00, -1.02184904e+00,  8.00654259e-01,
       -1.22754100e+00, -1.05003079e+00],
      [-1.09698989e+00, -7.79513300e-01,  1.03205722e+00,
       -1.28440670e+00, -1.31297673e+00],
      [-1.07389537e+00, -7.79513300e-01,  8.00654259e-01,
       -1.34127240e+00, -1.31297673e+00],
```

```
newdataset=pd.DataFrame(x,columns=dataset.columns[:-1])
```

```
newdataset
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-1.720542	-0.900681	1.032057	-1.341272	-1.312977
1	-1.697448	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.674353	-1.385353	0.337848	-1.398138	-1.312977
3	-1.651258	-1.506521	0.106445	-1.284407	-1.312977
4	-1.628164	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...	...
145	1.628164	1.038005	-0.124958	0.819624	1.447956
146	1.651258	0.553333	-1.281972	0.705893	0.922064
147	1.674353	0.795669	-0.124958	0.819624	1.053537
148	1.697448	0.432165	0.800654	0.933356	1.447956
149	1.720542	0.068662	-0.124958	0.762759	0.790591

150 rows × 5 columns

```
from sklearn.model_selection import train_test_split
```

```
x=newdataset
y=dataset['Species']
```

```
x
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-1.720542	-0.900681	1.032057	-1.341272	-1.312977
1	-1.697448	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.674353	-1.385353	0.337848	-1.398138	-1.312977
3	-1.651258	-1.506521	0.106445	-1.284407	-1.312977
4	-1.628164	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...	...
145	1.628164	1.038005	-0.124958	0.819624	1.447956
146	1.651258	0.553333	-1.281972	0.705893	0.922064
147	1.674353	0.795669	-0.124958	0.819624	1.053537
148	1.697448	0.432165	0.800654	0.933356	1.447956
149	1.720542	0.068662	-0.124958	0.762759	0.790591

150 rows × 5 columns

```
y
```

```
0          Iris-setosa
1          Iris-setosa
2          Iris-setosa
3          Iris-setosa
4          Iris-setosa
              ...
145     Iris-virginica
146     Iris-virginica
147     Iris-virginica
148     Iris-virginica
149     Iris-virginica
Name: Species, Length: 150, dtype: object
```

```
X_train,X_test,Y_train,Y_test=train_test_split(x,y,stratify=y,test_size=0.5,random_state=90)
x.shape
```

```
(150, 5)
```

naive bayes

dataset values are continuous use quotient classifier. dataset values are in type of yes/no,true/false Bernoullie classifier. multinomial

```
from sklearn.naive_bayes import GaussianNB,BernoulliNB,CategoricalNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
```

```
scores=[]
classifier=GaussianNB()
classifier.fit(X_train,Y_train)
y_pred=classifier.predict(X_test)
scores.append(accuracy_score(Y_test,y_pred))
cm=confusion_matrix(Y_test,y_pred)
cm
```

```
array([[25,  0,  0],
       [ 0, 25,  0],
       [ 0,  0, 25]])
```

```
print(classification_report(Y_test,y_pred))
```

```
[[25  0  0]
 [ 0 25  0]
 [ 0  0 25]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        25
Iris-versicolor       1.00      1.00      1.00        25
 Iris-virginica       1.00      1.00      1.00        25

       accuracy                           1.00        75
      macro avg       1.00      1.00      1.00        75
   weighted avg       1.00      1.00      1.00        75
```

RESULT:

The program executed successfully and obtained the output.

PROGRAM - 7

AIM:

Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

DATASET:

Salary_Data

```
#Simple Linear Regression

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

df=pd.read_csv('/content/sample_data/Salary_Data.csv')

df.head()

   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0

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

# split the dataset into training set and testing set
from sklearn.model_selection import train_test_split
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size=0.3,rando
m_state=30)

# fit the training dataset
from sklearn.linear_model import LinearRegression
regsr=LinearRegression()
regsr.fit(x_train,y_train)

LinearRegression()

y_pred=regsr.predict(x_test)

#visualize
plt.scatter(x_train,y_train,color='red')
plt.plot(x_train,regsr.predict(y_train),color='green')
plt.title("salary VS experience")
plt.xlabel("experience")
```

```
plt.ylabel("salary")
plt.show()
```


```
# visualizing the test results
plt.scatter(x_test,y_test,color='red')
plt.plot(x_test,regsr.predict(y_test),color='green')
plt.title("salary VS experience")
plt.xlabel("experience")
plt.ylabel("salary")
plt.show()
```


```
# Multiple Linear Regression

ds=pd.read_csv('/content/sample_data/combined cycle powerplant.csv')

ds.head()

      AT      V       AP     RH      PE
0   8.34  40.77  1010.84  90.01  480.48
1  23.64  58.49  1011.40  74.20  445.75
2  29.74  56.90  1007.15  41.91  438.76
3  19.07  49.69  1007.22  76.79  453.09
4  11.80  40.66  1017.13  97.20  464.43

u=ds.drop('PE',axis=1).values
v=ds['PE'].values

from sklearn.model_selection import train_test_split
u_train,u_test,v_train,v_test =
train_test_split(u,v,test_size=0.3,random_state=45)

#fitting
from sklearn.linear_model import LinearRegression
rgr=LinearRegression()
rgr.fit(u_train,v_train)

LinearRegression()

v_pred=rgr.predict(u_test)
print(v_pred)
```

```
[451.12275809 472.67973273 434.23317529 ... 479.20120415 470.80190333
 437.26990979]
```

```
#take the values of first row in u and compare our predicted v values
with actual v value
rgr.predict([[23.64   ,58.49,    1011.40,   74.20]])
```

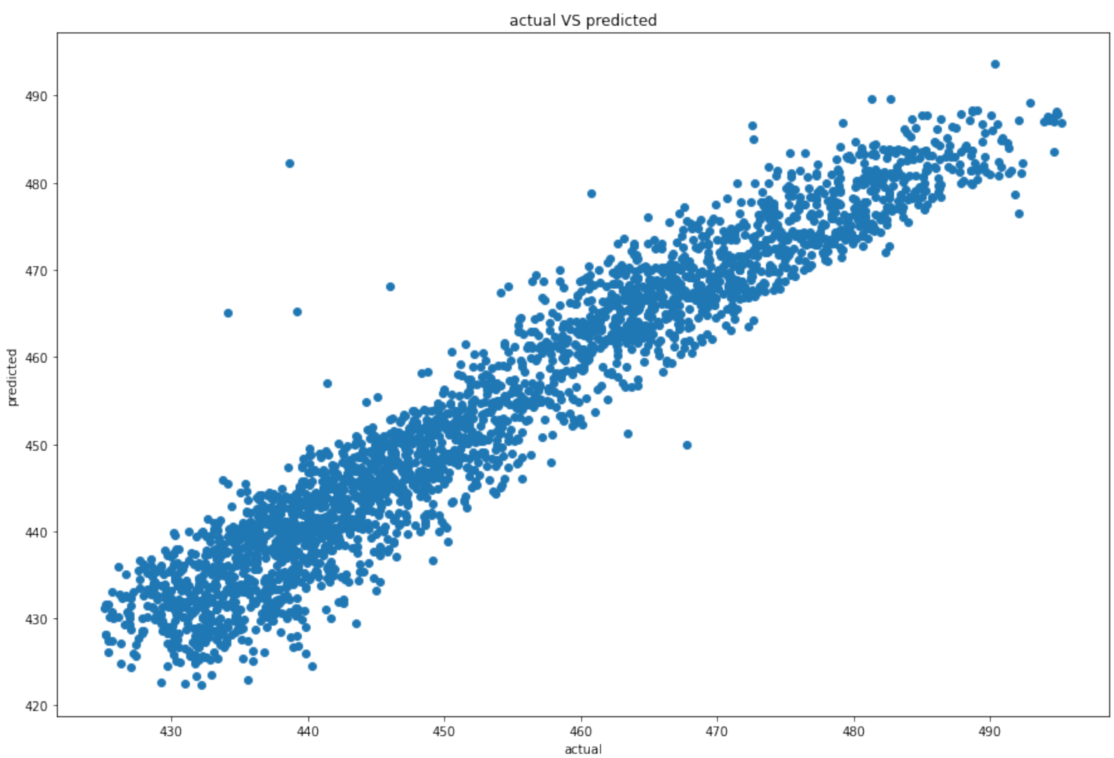
```
array([445.23162503])
```

```
from sklearn.metrics import r2_score
r2_score(v_test,v_pred)
```

```
0.9270483924843018
```

```
# visualize
plt.figure(figsize=(15,10))
plt.scatter(v_test,v_pred)
plt.title("actual VS predicted")
plt.xlabel("actual")
plt.ylabel("predicted")
```

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




```
#print prdedicted values of our model
pred_ds=pd.DataFrame({'Actual value':v_test ,'Predicted
value':v_pred})
pred_ds
```

```
      Actual value  Predicted value
0           449.23       451.122758
1           474.70       472.679733
2           434.18       434.233175
3           436.70       442.479946
4           477.27       481.164400
...            ...              ...
2866        465.26       462.625918
2867        441.71       442.161640
2868        477.51       479.201204
2869        467.62       470.801903
2870        438.52       437.269910

[2871 rows x 2 columns]
```

RESULT:

The program executed successfully and obtained the output.