AIM

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

Programming code:

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']

# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

# Label or target varible

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']



 from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```

OUTPUT:

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

Programming code:

```
temp_encoded=le.fit_transform(temp)
print(temp_encoded)
print(" ")
label=le.fit_transform(play)
print(label)
```

OUTPUT:

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

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

Programming code:

```
features=list(zip(weather_encoded,temp_encoded))
print(features)
```

OUTPUT:

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

Programming code:

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)


from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)
predicted= model.predict([[0,1]]) # 0:Overcast, 1:Hot
print(predicted)
```

OUTPUT:

```
[1]
```

Dataset used: iris.csv

Programming code:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
dataset = pd.read_csv("iris.csv")
print(dataset.describe)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
```

OUTPUT:

```
<bound method NDFrame.describe of      sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]>
```

Programming code:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
print(X_test)
```

OUTPUT:

```
[[ 2.17968894 -0.14585275  1.67223212  1.24723193]
 [ 1.09336496  0.52731379  1.16075926  1.24723193]
 [ 0.61055431 -1.26779698  0.76294703  0.98465678]
 [ 1.09336496 -0.14585275  0.76294703  0.72208164]
 [-0.23436434 -0.59463044  0.70611671  1.11594435]
 [-1.32068831  0.30292494 -1.33977476 -1.24723193]
 [-0.47576967  0.75170264 -1.11245349 -1.24723193]
 [-0.11366168 -0.59463044  0.47879543  0.19693136]
 [-0.83787766 -1.26779698 -0.37365934 -0.06564379]
 [-1.07928299  1.20048033 -1.28294444 -1.3785195 ]
 [ 0.61055431 -0.81901929  0.70611671  0.85336921]
 [-0.83787766  1.64925802 -0.99879285 -0.98465678]
 [ 0.12774365  0.30292494  0.64928639  0.85336921]
 [-0.11366168 -0.3702416   0.30830448  0.19693136]
 [ 0.61055431 -1.26779698  0.70611671  0.4595065 ]
 [ 0.61055431 -0.59463044  0.81977735  0.4595065 ]
 [ 0.00704099  2.09803572 -1.39660508 -1.24723193]
 [-0.83787766  1.64925802 -1.16928381 -1.24723193]
 [-1.07928299 -1.26779698  0.47879543  0.72208164]
 [-0.71717499  0.75170264 -1.28294444 -1.24723193]
 [ 1.45547296  0.30292494  0.59245607  0.32821893]
 [ 0.12774365 -0.14585275  0.81977735  0.85336921]
 [-1.19998565 -0.14585275 -1.28294444 -1.3785195 ]
 [ 0.00704099 -1.04340814  0.19464384  0.06564379]
 [ 1.21406763  0.30292494  1.27441989  1.50980707]
 [-0.71717499 -0.81901929  0.13781352  0.32821893]
 [ 0.36914898 -0.59463044  0.59245607  0.06564379]
 [ 0.36914898 -1.04340814  1.10392894  0.32821893]
 [ 0.00704099 -0.59463044  0.81977735  1.64109464]

[-0.83787766  0.75170264 -1.22611412 -1.24723193]
[-1.44139098  0.0785361  -1.22611412 -1.24723193]
[-1.07928299 -1.49218583 -0.20316839 -0.19693136]
[-1.07928299 -0.14585275 -1.28294444 -1.24723193]
[ 0.73125697  0.30292494  0.93343798  1.50980707]
[ 0.48985164  0.75170264  0.9902683   1.50980707]
[ 1.33477029  0.30292494  1.16075926  1.50980707]
[-0.83787766  1.42486918 -1.22611412 -0.98465678]
[-1.68279631 -0.14585275 -1.33977476 -1.24723193]

[ 0.85195964 -0.59463044  0.53562575  0.4595065 ]
[-1.44139098  1.20048033 -1.51026572 -1.24723193]
[-0.83787766  1.64925802 -1.22611412 -1.11594435]
[-0.71717499  2.32242456 -1.22611412 -1.3785195 ]
[ 0.73125697  0.0785361   1.04709862  0.85336921]
[-0.95858032 -0.14585275 -1.16928381 -1.24723193]
[-0.95858032  0.97609148 -1.33977476 -1.11594435]
```

```
[ 1.33477029  0.0785361   0.9902683   1.24723193]
[-1.44139098  0.75170264 -1.28294444 -1.11594435]
[ 0.61055431  0.52731379  1.33125021  1.77238221]
[-0.23436434 -1.26779698  0.13781352 -0.06564379]
[ 0.48985164 -0.59463044  0.64928639  0.85336921]
[ 0.85195964 -0.14585275  1.04709862  0.85336921]
[-0.355067   -1.49218583  0.02415289 -0.19693136]
[-0.23436434 -0.81901929  0.30830448  0.19693136]
[ 1.21406763 -0.14585275  1.04709862  1.24723193]
[ 0.61055431  0.75170264  1.10392894  1.64109464]
[ 1.69687828  1.20048033  1.38808053  1.77238221]
[ 2.3003916  -0.14585275  1.38808053  1.50980707]
[-1.07928299  0.0785361  -1.22611412 -1.24723193]
[ 0.61055431 -0.3702416   1.10392894  0.85336921]
[ 0.36914898 -0.59463044  0.19464384  0.19693136]
[-0.47576967  1.87364687 -1.11245349 -0.98465678]
[-0.47576967 -0.14585275  0.47879543  0.4595065 ]
[ 1.93828361 -0.59463044  1.38808053  0.98465678]
[ 1.09336496 -0.14585275  0.87660766  1.50980707]
[-1.80349897 -0.14585275 -1.4534354  -1.3785195 ]
[-0.11366168  2.9955911  -1.22611412 -0.98465678]
[-0.95858032 -1.71657468 -0.20316839 -0.19693136]
[-0.95858032 -2.38974122 -0.08950775 -0.19693136]
[-0.23436434 -0.14585275  0.47879543  0.4595065 ]
[ 0.36914898 -0.14585275  0.53562575  0.32821893]
[ 0.24844632 -0.14585275  0.64928639  0.85336921]
[-1.19998565  0.0785361  -1.16928381 -1.24723193]

[ 0.12774365 -0.14585275  0.30830448  0.4595065 ]
[ 0.73125697  0.30292494  0.47879543  0.4595065 ]
[-0.95858032  0.97609148 -1.16928381 -0.72208164]
[ 2.3003916   1.64925802  1.72906244  1.3785195 ]
[ 0.73125697 -0.59463044  1.10392894  1.24723193]
[ 1.33477029  0.0785361   0.81977735  1.50980707]
[-0.83787766  0.52731379 -1.11245349 -0.85336921]
[ 1.57617562 -0.14585275  1.27441989  1.24723193]
[-0.355067    0.97609148 -1.33977476 -1.24723193]
[-0.47576967  0.75170264 -1.22611412 -0.98465678]
[-1.19998565  0.75170264 -0.99879285 -1.24723193]
[-0.355067   -1.26779698  0.19464384  0.19693136]
[ 0.73125697 -0.59463044  1.10392894  1.3785195 ]
[ 0.00704099 -0.81901929  0.81977735  0.98465678]
[ 1.09336496  0.0785361   1.10392894  1.64109464]
[-0.11366168 -0.59463044  0.25147416  0.19693136]
[ 0.61055431  0.52731379  0.59245607  0.59079407]
[ 0.48985164 -1.94096352  0.47879543  0.4595065 ]
[ 0.85195964  0.30292494  0.81977735  1.11594435]
[-0.83787766  0.97609148 -1.28294444 -1.11594435]
[ 1.69687828 -0.14585275  1.21758958  0.59079407]
[ 1.09336496 -1.26779698  1.21758958  0.85336921]]
```

Programming code:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_test)
print(' ')
print(y_pred)
```

OUTPUT:

```
['Virginica' 'Virginica' 'Setosa' 'Virginica' 'Versicolor' 'Virginica'
 'Virginica' 'Setosa' 'Versicolor' 'Versicolor' 'Setosa' 'Virginica'
 'Virginica' 'Virginica' 'Setosa' 'Virginica' 'Setosa' 'Setosa' 'Setosa'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Virginica' 'Setosa' 'Virginica' 'Virginica']

['Virginica' 'Virginica' 'Setosa' 'Virginica' 'Versicolor' 'Versicolor'
 'Virginica' 'Setosa' 'Versicolor' 'Versicolor' 'Setosa' 'Virginica'
 'Virginica' 'Virginica' 'Setosa' 'Virginica' 'Setosa' 'Setosa' 'Setosa'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Virginica' 'Setosa' 'Virginica' 'Virginica']
```

Programming code:

```
from sklearn.metrics import classification_report, confusion_matr
ix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

OUTPUT:

```
[[ 8  0  0]
 [ 0  7  0]
 [ 0  1 14]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         8
  Versicolor       0.88      1.00      0.93         7
   Virginica       1.00      0.93      0.97        15

    accuracy                           0.97        30
   macro avg       0.96      0.98      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Programming code:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fruits=pd.read_table('/content/fruit_data_with_colors.txt')


fruits.head()
```

OUTPUT:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

Programming code:

```
fruits.shape
predct = dict(zip(fruits.fruit_label.unique(), fruits.fruit_name.u
nique()))
predct
```

OUTPUT:

```
(59, 7)

{1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}
```

Programming code:

```
apple_data=fruits[fruits['fruit_name']=='apple']
orange_data=fruits[fruits['fruit_name']=='orange']
lemon_data=fruits[fruits['fruit_name']=='lemon']
mandarin_data=fruits[fruits['fruit_name']=='mandarin']


apple_data.head()
```

OUTPUT:

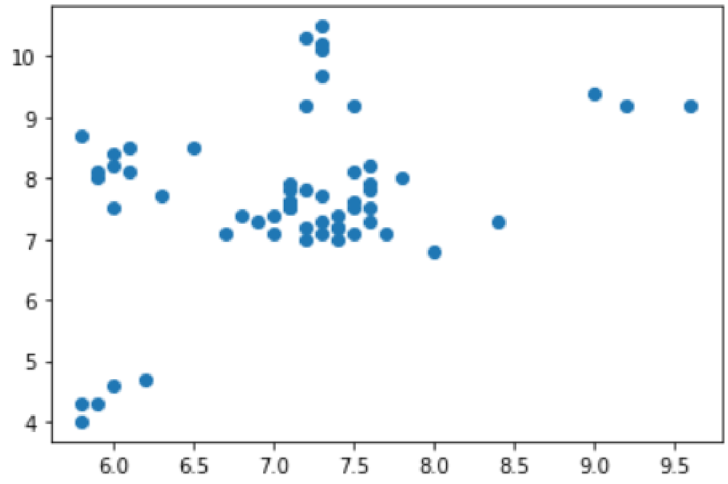
	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89

Programming code:

```
plt.scatter(fruits['width'],fruits['height'])
```

OUTPUT:

```
<matplotlib.collections.PathCollection at 0x7f1a659c7690>
```

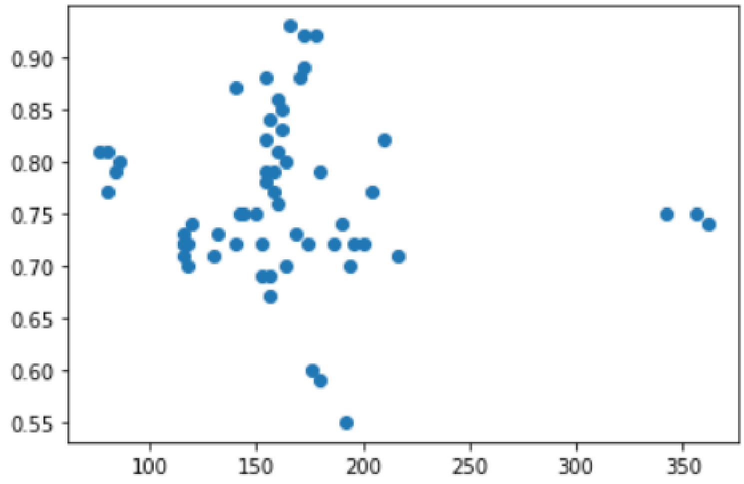


Programming code:

```
plt.scatter(fruits['mass'],fruits['color_score'])
```

OUTPUT:

```
<matplotlib.collections.PathCollection at 0x7f1a65485a50>
```



Programming code:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


X=fruits[['mass','width','height']]
Y=fruits['fruit_label']
X_train,X_test,y_train,y_test=train_test_split(X,Y,random_state=0
)

X_train.describe()
```

OUTPUT:

	mass	width	height
count	44.000000	44.000000	44.000000
mean	159.090909	7.038636	7.643182
std	53.316876	0.835886	1.370350
min	76.000000	5.800000	4.000000
25%	127.500000	6.175000	7.200000
50%	157.000000	7.200000	7.600000
75%	172.500000	7.500000	8.250000
max	356.000000	9.200000	10.500000

Programming code:

```
X_test.describe()
```

OUTPUT:

	mass	width	height
count	15.000000	15.00000	15.000000
mean	174.933333	7.30000	7.840000
std	60.075508	0.75119	1.369463
min	84.000000	6.00000	4.600000
25%	146.000000	7.10000	7.250000
50%	166.000000	7.20000	7.600000
75%	185.000000	7.45000	8.150000
max	362.000000	9.60000	10.300000

Programming code:

```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

OUTPUT:

```
KNeighborsClassifier()
```

Programming code:

```
knn.score(X_test,y_test)
```

OUTPUT:

```
0.5333333333333333
```

Programming code:

```
prediction1=knn.predict([['100','6.3','8']])
predct[prediction1[0]]
```

OUTPUT:

lemon

Programming code:

```
prediction2=knn.predict([['300','7','10']])
predct[prediction2[0]]
```

OUTPUT: