Data Science Lab

Haritha Krishnan

RMCA S3

Roll No : 40

Date:24/11/2021

PROGRAM NO: 1

AIM: Program to perform Matrix Operations

PROGRAM CODE

```
import numpy as np
import random
def PrintMatrix(matrix_in):
    for x in range(0, matrix_in.shape[0]):
        for y in range(0, matrix_in.shape[1]):
            print("%d \t" % (matrix_in[x][y]), end='')
            if (y % 3 > 1):
                print("\n")
def FillMatrix(matrix_in):
    for x in range(0, matrix_in.shape[0]):
        for y in range(0, matrix_in.shape[1]):
            matrix_in[x][y] = random.randrange(2, 10) + 2
matrix1 = np.ndarray((3,3))
matrix2 = np.ndarray((3,3))
FillMatrix(matrix1)
FillMatrix(matrix2)
add_results = np.add(matrix1,matrix2)
sub_results=np.subtract(matrix1,matrix2)
mult_results=np.multiply(matrix1,matrix2)
div_results=np.divide(matrix1,matrix2)
dot_results=np.dot(matrix1,matrix2)
sqrt1_results=np.sqrt(matrix1)
sqrt2_results=np.sqrt(matrix2)
trans_results=add_results.T
print("Matrix1:")
PrintMatrix(matrix1)
print("Matrix2:")
PrintMatrix(matrix2)
```

```
print("Adding")
PrintMatrix(add_results)
print("Subtraction")
PrintMatrix(sub_results)
print("Multiplication")
PrintMatrix(mult_results)
print("Dot Operation")
PrintMatrix(dot_results)
print("squareroot Operation")
print("matrix 1")
PrintMatrix(sqrt1_results)
print("matrix 2")
PrintMatrix(sqrt2_results)
print("Transpose")
PrintMatrix(trans_results)
```

OUTPUT

```
Matrix1:
4    4    11
6    4    6
9    11   5
Matrix2:
8    10   10
11   9    8
8    11   10
Adding
12   14   21
17   13   14
17   22   15
Subtraction
-4   -6   1
-5   -5   -2
1    0    -5
```

Multiplication

32	40	110
66	36	48
72	121	50

Dot Operation

164	197	182
140	162	152
233	244	228

Squareroot Operation

matrix 1

2	2	3
2	2	2
3	3	2

matrix 2

2	3	3
3	3	2
2	3	3

Transpose

12	17	17
14	13	22
21	14	15

Process finished with exit code 0

Date:01/12/2021

PROGRAM NO: 2

AIM: Program to perform SVD (Singular value Decomposition) using Python

PROGRAM CODE

```
from scipy. linalg import svd
from numpy import array
A= ([[1,2,5], [2,0,1], [1,4,4]])
print(A)
X, B, T=svd(A)
print("decomposition")
print(X)
print("inverse")
print(B)
print("transpose")
print(T)
```

OUTPUT

```
 [[1, 2, 5], [2, 0, 1], [1, 4, 4]]
decomposition
[[-0.68168247 -0.26872313 -0.68051223]
 [-0.15885378 -0.85356116  0.49618427]
 [-0.71419499  0.44634205  0.53916999]]
inverse
[7.87492    2.01650097 1.38540929]
transpose
[[-0.21760031 -0.53589686 -0.81576017]
 [-0.75849376  0.61885512 -0.20421939]
 [ 0.61427789  0.5743108  -0.54113749]]
Process finished with exit code 0
```

Date:01/12/2021

PROGRAM NO: 3

AIM: Program to implement KNN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

PROGRAM CODE

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
iris = load_iris()
x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train,y_train)
print(knn.predict(x_test))
V=knn.predict(x_test)
result=accuracy_score (y_test, V)
print ("accuracy:", result)
```

OUTPUT

```
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
accuracy: 0.9666666666666667
Process finished with exit code 0
```

Date:01/12/2021

PROGRAM NO: 4

AIM: Program to implement KNN classification using any random dataset without using inbuilt packages.

PROGRAM CODE

```
from math import sqrt
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1) - 1):
        distance += (row1[i] - row2[i]) ** 2
    return sqrt(distance)
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
# Test distance function
dataset = [[2.781, 2.550,0],
        [1.465, 2.326,3],
        [3.398, 4.429,5],
        [1.388, 1.857,11],
        [3.064, 3.393,3],
```

```
        [7.624, 2.235,4],
        [5.338, 2.775,8]]
prediction = predict_classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

OUTPUT

Expected 2, Got 3.

Process finished with exit code 0

Date:08/12/2021

PROGRAM NO: 5

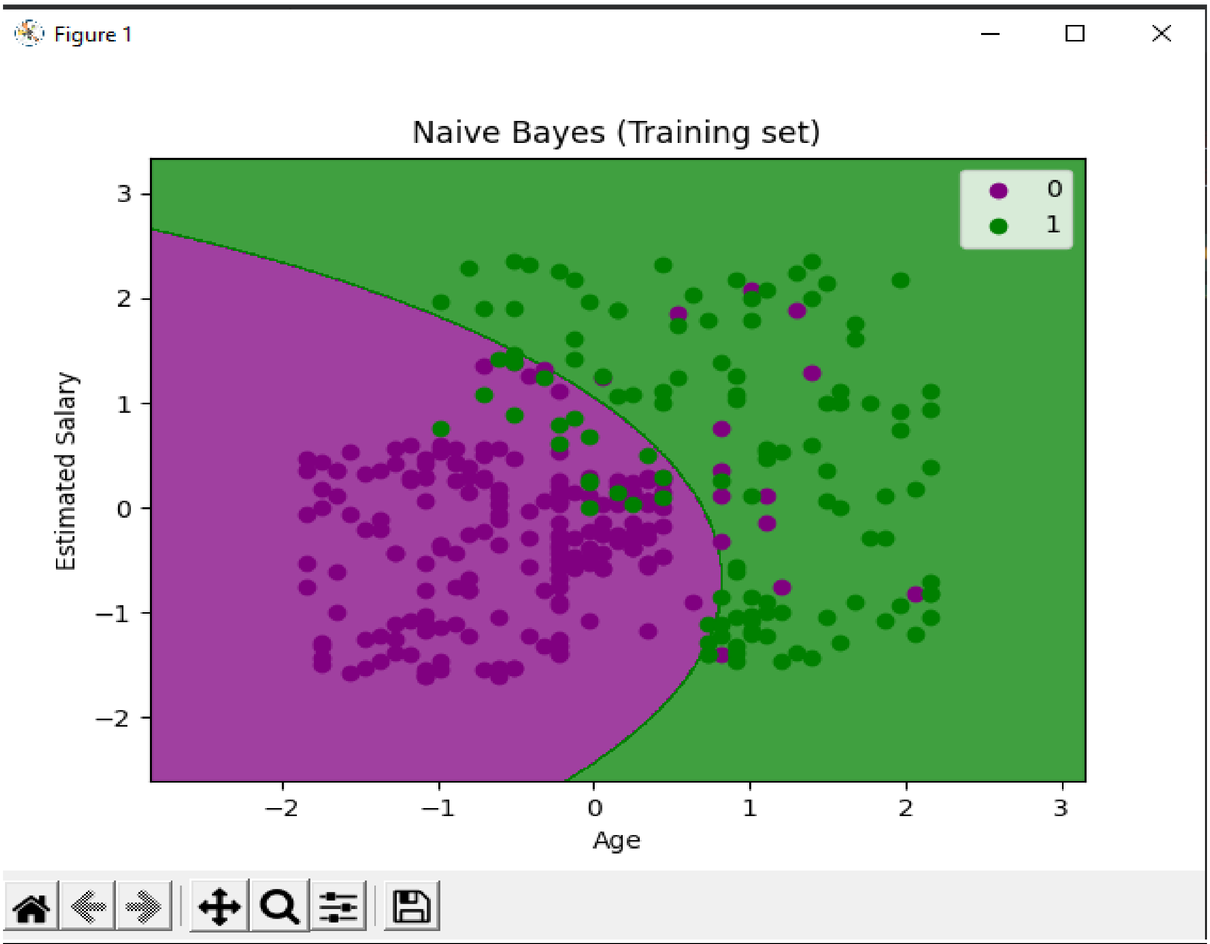
AIM: Program to implement Naive Bayes Algorithm using any standard dataset available in the public domain and find accuracy.

PROGRAM CODE

```
import pandas as pd

dataset = pd.read_csv('Social_Network_Ads.csv')

x = dataset.iloc[:, [2,3]].values

y = dataset.iloc[:,-1].values

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=10)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(x_train, y_train)

y_pred = gnb.predict(x_test)

print(y_pred)

from sklearn import metrics

print("Accuracy", metrics.accuracy_score(y_test, y_pred) * 100)

import numpy as nm

import matplotlib.pyplot as mtp

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),

  nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(X1, X2, gnb.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('purple', 'green')))

mtp.xlim(X1.min(), X1.max())
```

```
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
  c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step = 0.01),
 nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, gnb.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
 alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
 c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT



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

Accuracy 91.25

Date:08/12/2021

PROGRAM NO: 6

AIM: Program to implement Linear Regression with inbuilt functions using any standard dataset in public domain and evaluate performance.

PROGRAM CODE

```
import numpy as np

from sklearn.linear_model import LinearRegression

x  = np.array([2,6,7,8]).reshape((-1,1))

y = np.array([16,7,8,9])

model = LinearRegression()

model.fit(x,y)

r_sq = model.score(x,y)

print("Score: ",r_sq)

print("Intercept: ",model.intercept_)

print("Slope: ",model.coef_)

y_pred = model.predict(x)

print("Y-prediction : ",y_pred)
```

OUTPUT

```
Score:  0.7556626506024098

Intercept:  17.759036144578314

Slope:  [-1.34939759]

Y-prediction :  [15.06024096  9.6626506   8.31325301  6.96385542]

Process finished with exit code 0
```

Date:08/12/2021

PROGRAM NO: 7

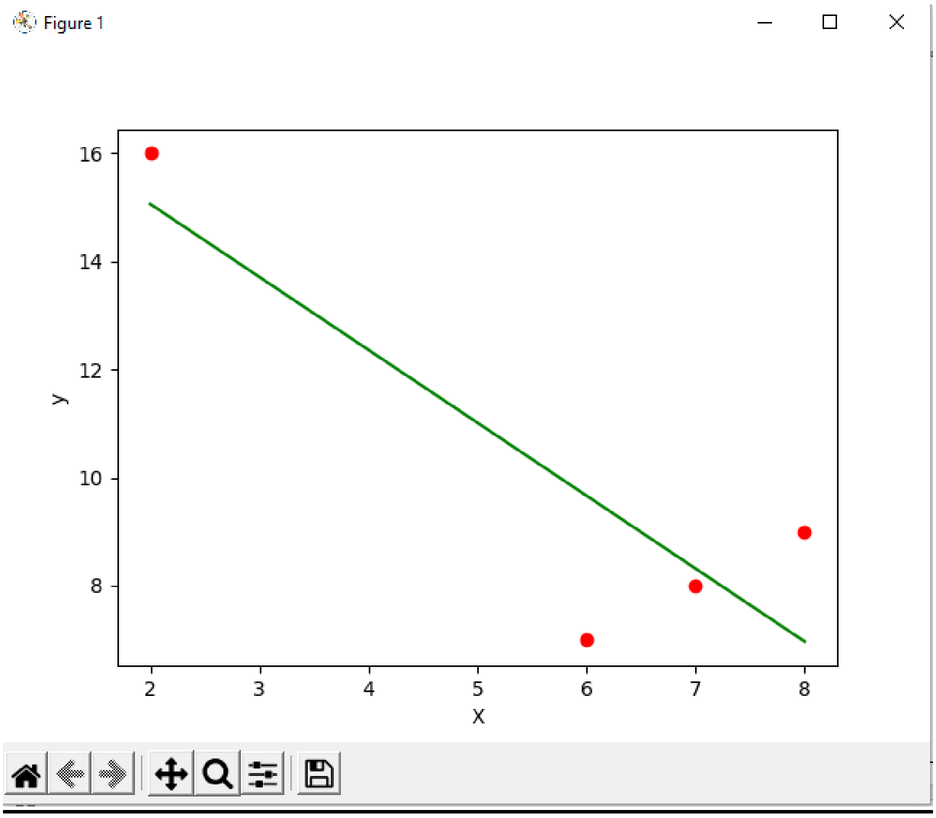
AIM:

Program to implement Linear Regression without inbuilt functions..

PROGRAM CODE

```
import numpy as np

import matplotlib.pyplot as plt

x  = np.array([2,6,7,8])

y = np.array([16,7,8,9])

n = np.size(x)

n_x = np.mean(x)

n_y = np.mean(y)

SS__xy = np.sum(y*x)-n* n_y*n_x

SS__xx = np.sum(x*x)-n* n_x*n_x

b_1 = SS__xy/SS__xx

b_0 = n_y - b_1*n_x

y_pred = b_1 * x + b_0

print(y_pred)

plt.scatter(x, y, color='red')

plt.plot(x, y_pred, color='green')

plt.xlabel('X')

plt.ylabel('y')

plt.show()
```

OUTPUT



[15.06024096 9.6626506   8.31325301 6.96385542]

Date:15/12/2021

PROGRAM NO: 8

AIM:

Program to implement Multiple Linear Regression.

PROGRAM CODE

```
import pandas

from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]

y = df['CO2']

regr = linear_model.LinearRegression()

regr.fit(X, y)

#predict the CO2

predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

OUTPUT

```
 [107.2087328]

Process finished with exit code 0
```

Date:15/12/2021

PROGRAM NO: 9

AIM:

Program to implement Multiple Linear Regression with inbuilt functions using and dataset in public domain and evaluate performances.

PROGRAM CODE

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear_model, metrics

from sklearn.metrics import r2_score

boston = datasets.load_boston(return_X_y=False)

X = boston.data

y = boston.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,random_state=1)

reg = linear_model.LinearRegression()

reg.fit(X_train, y_train)

V=reg.predict(X_test)

result=r2_score(y_test, V)

print("accuracy :", result)

print('Coefficients: ', reg.coef_)

print('Variance score:{}'.format(reg.score(X_test, y_test)))
```

OUTPUT

accuracy : 0.7209056672661767

Coefficients: [-8.95714048e-02 6.73132853e-02 5.04649248e-02 2.18579583e+00

 -1.72053975e+01 3.63606995e+00 2.05579939e-03 -1.36602886e+00

  2.89576718e-01 -1.22700072e-02 -8.34881849e-01 9.40360790e-03

 -5.04008320e-01]

Variance score:0.720905667266176

Process finished with exit code 0

Date:22/12/2021

PROGRAM NO: 10

AIM:

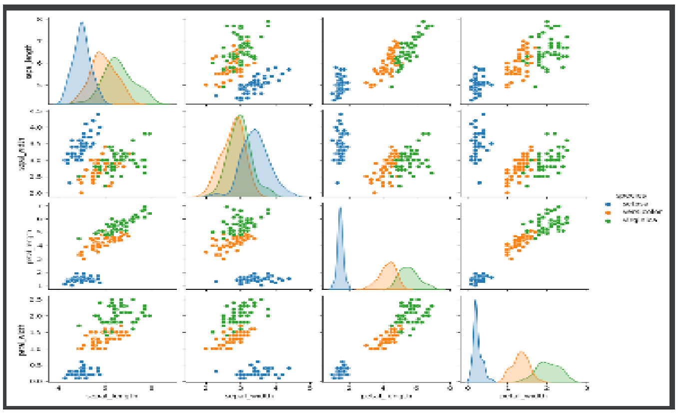
Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

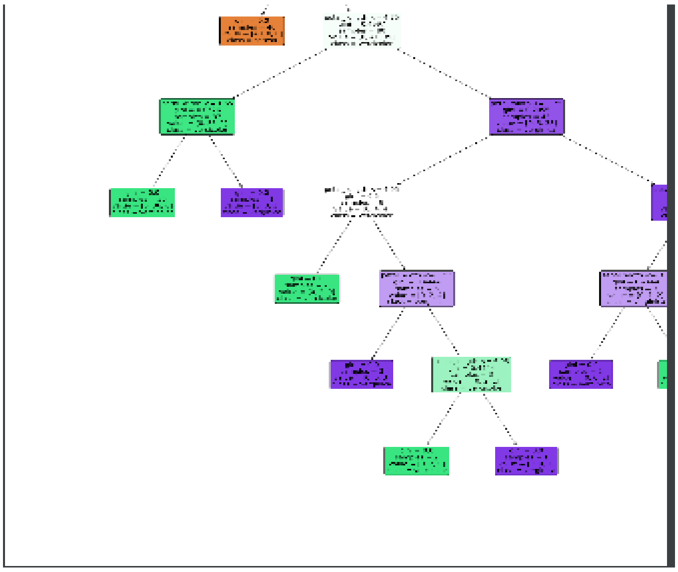
<u>PROGRAM CODE</u>

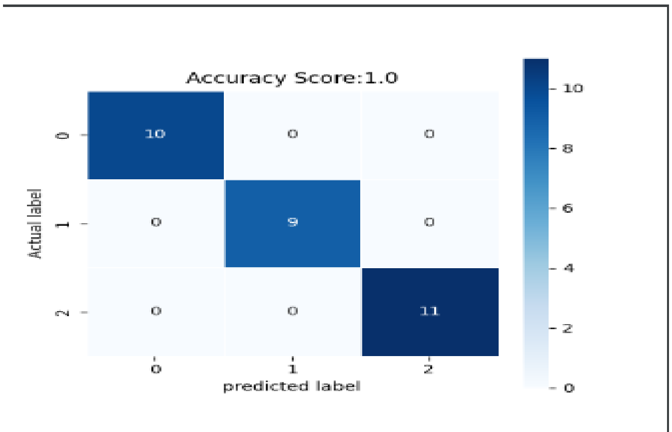
```
Import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from  sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from  sklearn.tree import DecisionTreeClassifier
from  sklearn.metrics import classification_report, confusion_matrix
from  sklearn.tree import  plot_tree
df=sns.load_dataset('iris')
print(df.head())
print(df.info())
df.isnull().any()
print(df.shape)
sns.pairplot(data=df, hue ='species')
plt.savefig("pne.png")
sns.heatmap(df.corr())
plt.savefig("next.png")
target =df['species']
df1 = df.copy()
df1 = df1.drop('species', axis=1)

print(df1.shape)
print(df1.head())
```

```
x=df1
print(target)
le = LabelEncoder()
target = le.fit_transform(target)
print(target)
y= target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state= 42)
print("training split input" , x_train.shape)
print("test split input",x_test.shape)
dtree=DecisionTreeClassifier()
dtree.fit(x_train, y_train)
print("decision tree classifer created")
y_pred = dtree.predict(x_test)
print("classification report-\n",classification_report(y_test,y_pred))
cm = confusion_matrix(y_test,y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5,annot=True,square=True,cmap='Blues')
plt.ylabel('Actual label')
plt.xlabel('predicted label')
all_sample_title = 'Accuracy Score:{0}'.format(dtree.score(x_test,y_test))
plt.title(all_sample_title,size=12)
plt.savefig("two.png")
plt.figure(figsize=(20,20))
dec_tree=plot_tree(decision_tree=dtree,feature_names=df1.columns,class_names=["setosa","
vercicolor","verginica"],filled=True ,precision=4,rounded=True)
plt.savefig("three.png")
```

OUTPUT






Accuracy Score:1.0
Actual label
predicted label
10
0
0
0
9
0
0
0
11

Date:05/01/2022

PROGRAM NO: 11

AIM:

Program to implement k-means Clustering using any standard dataset available in the public domain.

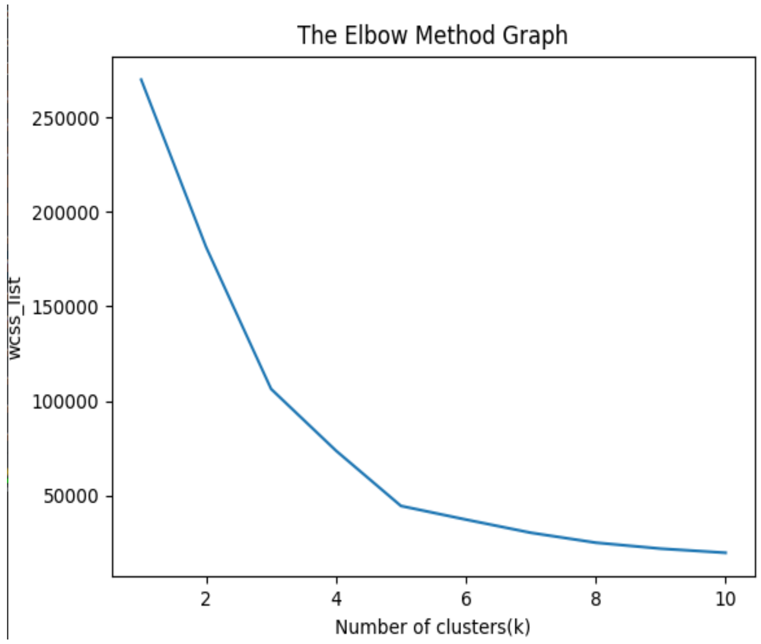
<u>PROGRAM CODE</u>

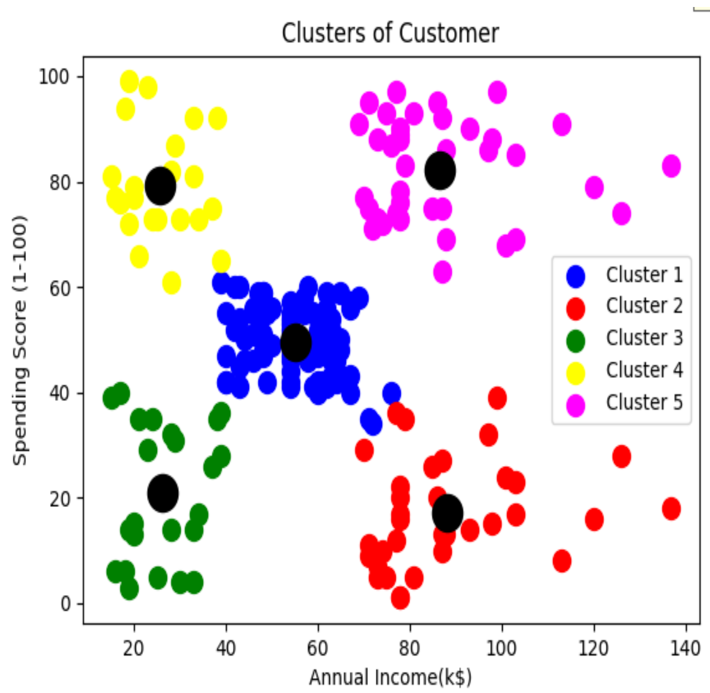
```
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

dataset=pd.read_csv('Mall_Customers.csv')

x=dataset.iloc[:,[3,4]].values

print(x)

from sklearn.cluster import KMeans

wcss_list=[]

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_)

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

mtp.title('The Elbow Method Graph')

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

mtp.ylabel('wcss_list')

mtp.show()

kmeans=KMeans(n_clusters=5,init='k-means++',random_state=42)

y_predict=kmeans.fit_predict(x)
```

```
print('predict=',y_predict)

mtp.scatter(x[y_predict==0,0],x[y_predict==0,1],s=100,c='blue',label='Cluster 1')

mtp.scatter(x[y_predict==1,0],x[y_predict==1,1],s=100,c='red',label='Cluster 2')

mtp.scatter(x[y_predict==2,0],x[y_predict==2,1],s=100,c='green',label='Cluster 3')

mtp.scatter(x[y_predict==3,0],x[y_predict==3,1],s=100,c='yellow',label='Cluster 4')

mtp.scatter(x[y_predict==4,0],x[y_predict==4,1],s=100,c='magenta',label='Cluster 5')

mtp.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300,c='black')

mtp.title('Clusters of Customer')

mtp.xlabel('Annual Income(k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend();

mtp.show()
```

OUTPUT

```
[[ 15  39]….

 [137  18]

 [137  83]]

predict= [2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2

 3 2 3 2 3 2 0 2 3 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 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4

 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1

 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```


The Elbow Method Graph
wcss_list
250000
200000
150000
100000
50000
2
4
6
8
10
Number of clusters(k)


Clusters of Customer
Spending Score (1-100)
100
80
60
40
20
0
20
40
60
80
100
120
140
Annual Income(k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5

\

Date:05/01/2022

PROGRAM NO: 12

AIM:

Program to implement k-means Clustering using any standard dataset available in the public domain.

<u>PROGRAM CODE</u>

```
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

dataset=pd.read_csv('world_country_and_usa_states_latitude_and_longitude_values.csv')

x=dataset.iloc[:,[1,2]].values

print(x)

from sklearn.cluster import KMeans

wcss_list=[]

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_)

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

mtp.title('The Elbow Method Graph')

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

mtp.ylabel('wcss_list')

mtp.show()

kmeans=KMeans(n_clusters=3,init='k-means++',random_state=42)

y_predict=kmeans.fit_predict(x)
```

```
print('predict=',y_predict)

mtp.scatter(x[y_predict==0,0],x[y_predict==0,1],s=100,c='blue',label='Cluster 1')

mtp.scatter(x[y_predict==1,0],x[y_predict==1,1],s=100,c='red',label='Cluster 2')

mtp.scatter(x[y_predict==2,0],x[y_predict==2,1],s=100,c='green',label='Cluster 3')

mtp.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300,c='black')

mtp.title('Clusters of world Country')

mtp.xlabel('latitude')

mtp.ylabel('longitude')

mtp.legend();

mtp.show()
```

<u>OUTPUT</u>