MANGALORE UNIVERSITY

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

23CSP301: ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LAB

SUBMITED BY

III SEMESTER MSC

Department Of Computer Science

Lectures, In-Charge:

1.

2.

Mangalore University

Dept. Of Post-Graduate Studies and Research in Computer Science

Mangalagangothri – 574199

INDEX

1. Write python code implement Principle Component Analysis (PCA)

[In]

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

# Step 1: Create some sample data (replace this with your dataset)
data = np.random.rand(100, 4)  # 100 samples with 3 features
np.set_printoptions(precision=4, suppress=True)
print("Formatted Array:")
print(data)
```

[Op]

```
Formatted Array:
[[0.2666 0.0937 0.3418 0.7863]
 [0.8568 0.5699 0.047  0.7688]
[0.4946 0.3422 0.2269 0.4906]
 [0.8033 0.34   0.2486 0.2008]
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
[0.0993 0.9452 0.2312 0.5525]
[0.8113 0.2682 0.4019 0.0768]
```

```
 [0.5901 0.8746 0.6367 0.8935]
 [0.5838 0.0718 0.7347 0.4778]
 [0.2914 0.1526 0.0812 0.8008]]
```

[In]

```
# Step 2: Standardize the data
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)
standardized_data = (data - mean) / std_dev
print (mean, std_dev)
print(standardized_data)
```

[Op]

```
[0.4628 0.5032 0.465  0.4608] [0.2756 0.2957 0.2768 0.2939]
[[-0.7117 -1.3848 -0.445   1.1077]
 [ 1.4294  0.2258 -1.5098  1.048 ]
 [ 1.2975 -0.0156  1.4554 -1.4551]
[ 0.6488 -0.6156  0.1956  0.289 ]
 [ 0.4247  1.6494  1.6967 -0.6075]
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
 [-0.6528 -1.1078  1.0338  1.6347]
[-1.3186  1.4949 -0.8445  0.312 ]
```

```
[ 0.4621  1.2563  0.6204  1.4723]
 [ 0.439  -1.4589  0.9743  0.0577]
 [-0.6217 -1.1856 -1.3863  1.157 ]]
```

[In]

```
# Step 3: Compute the covariance matrix
covariance_matrix = np.cov(standardized_data, rowvar=False)
size_cc = covariance_matrix.size
shape_cc = covariance_matrix.shape
print (size_cc, shape_cc)
print(covariance_matrix)
```

[Op]

```
16 (4, 4)
[[ 1.0101 -0.0036 -0.1223 -0.2014]
 [-0.0036  1.0101  0.1335 -0.0181]
 [-0.1223  0.1335  1.0101 -0.1144]
 [-0.2014 -0.0181 -0.1144  1.0101]]
```

[In]

```
# Step 4: Compute the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
```

```
print(eigenvalues)

print(eigenvectors)
```

[Op]

```
[0.7038 0.9329 1.1916 1.2121]
[[ 0.5892 -0.2907 -0.3059  0.689 ]
 [-0.1917 -0.7955  0.5691  0.081 ]
 [ 0.5337  0.4057  0.7407  0.0437]
 [ 0.5755 -0.3436 -0.1841 -0.7189]]
```

[In]

```
# Step 5: Sort eigenvalues and corresponding eigenvectors in descending order

sorted_indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[sorted_indices]

eigenvectors = eigenvectors[:, sorted_indices]

print(eigenvalues)

print(eigenvectors)
```

[Op]

```
[1.2121 1.1916 0.9329 0.7038]
[[ 0.689  -0.3059 -0.2907  0.5892]
 [ 0.081   0.5691 -0.7955 -0.1917]
 [ 0.0437  0.7407  0.4057  0.5337]
```

```
 [-0.7189 -0.1841 -0.3436  0.5755]]
```

[In]

```
# Step 6: Choose the number of components (or a threshold for explained variance)
n_components = 3 # Choose the number of principal components

# Step 7: Select the top 'n_components' eigenvectors
selected_eigenvectors = eigenvectors[:, :n_components]
print(selected_eigenvectors)
```

[Op]

```
[[ 0.689  -0.3059 -0.2907]
 [ 0.081   0.5691 -0.7955]
 [ 0.0437  0.7407  0.4057]
 [-0.7189 -0.1841 -0.3436]]
```

[In]

```
# Step 8: Project the data onto the selected eigenvectors to obtain the principal components
final_result = np.dot(standardized_data, selected_eigenvectors)

Step 9: Print the final result
print("Final Result after PCA:")
print(final_result)
```

[Op]

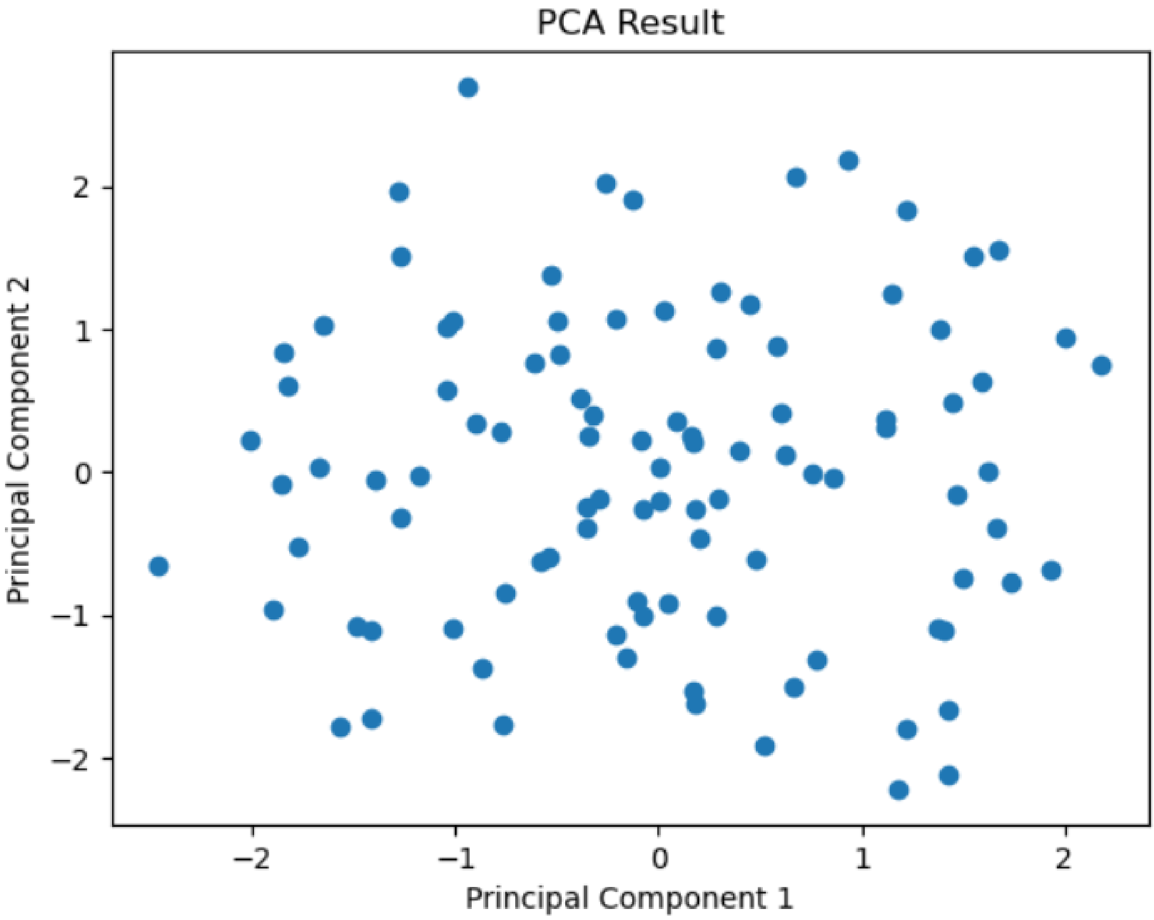
```
Final Result after PCA:
[[-1.4182 -1.1039  0.7475]
 [ 0.1837 -1.6202 -1.5678]
 [ 2.0023  0.9402  0.7256]
[-0.2059 -1.1382  0.0144]
 [-0.7615 -0.8458  1.0348]
 [-1.8561 -0.0797 -1.5945]
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
[-0.266   2.0247 -0.5115]
 [ 1.7362 -0.7673  0.6211]
 [-0.6113  0.762  -1.3879]
 [ 0.1853 -0.2534  1.4084]
 [-1.4166 -1.7244  0.164 ]]
```

[In]

```
# Step 10: Visualize the results (for 2D data)
if n_components == 3:
    plt.scatter(final_result[:, 0], final_result[:, 1])
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('PCA Result')
```

```
plt.show()
```

[Op]


PCA Result
Principal Component 2
Principal Component 1
2
1
0
−1
−2
−2
−1
0
1
2

2. Write a program to perform Linear Regression using Ordinary Least Square

[In]

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


df = pd.read_csv('BostonHousing.csv')

df
```

[Op]

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

[In]

```
# Features (inputs) are all columns except the target column

X = df.drop('medv', axis=1)
```

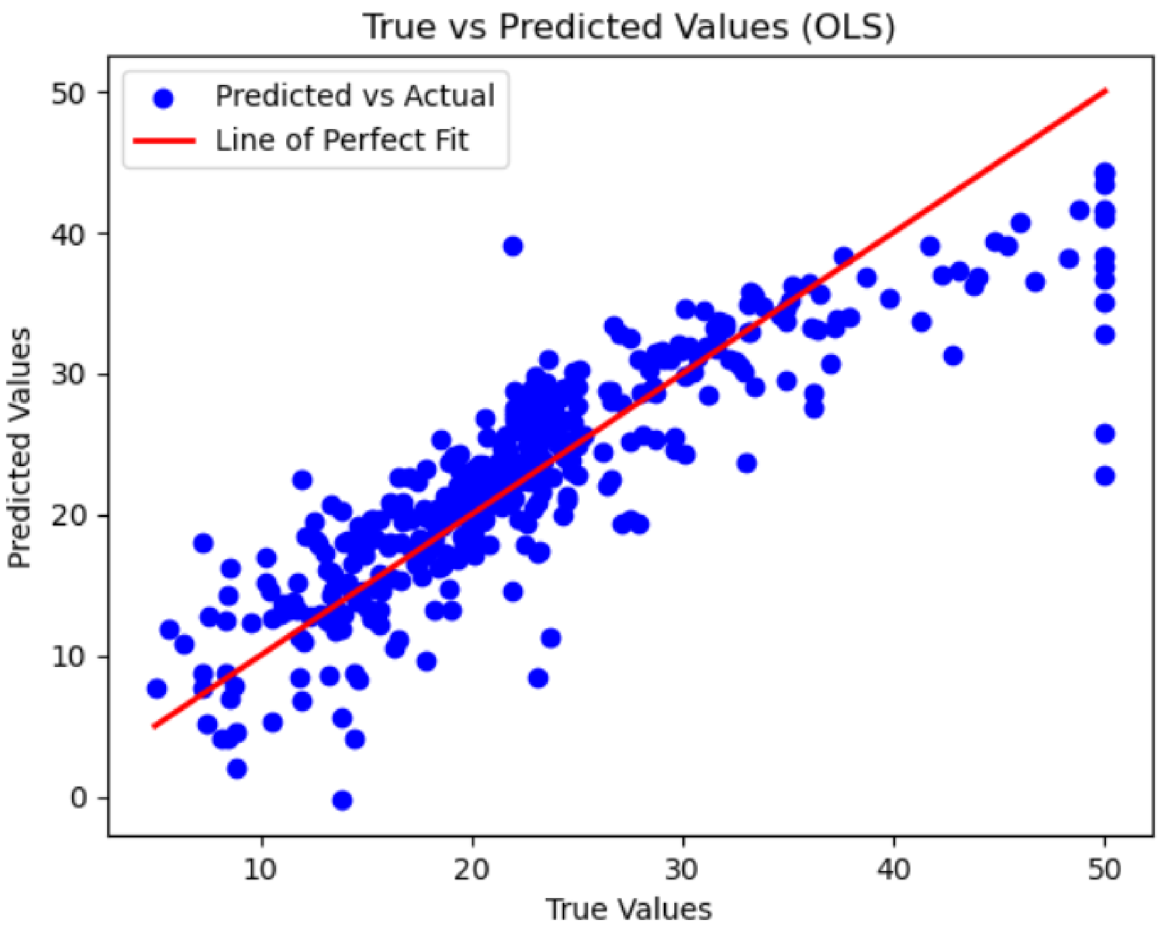
```
# Target variable (output) is the target column
y = df['medv']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

[In]

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Add a constant to the independent variable
X_train = sm.add_constant(X_train)

# Fit the OLS model
model = sm.OLS(y_train, X_train).fit()

# Predict the values
y_pred = model.predict(X_train)

# Plot the actual vs predicted values
plt.scatter(y_train, y_pred, color='blue', label='Predicted vs
Actual')
plt.plot([y_train.min(), y_train.max()], [y_train.min(),
y_train.max()], color='red', lw=2, label='Line of Perfect Fit')
plt.xlabel('True Values')
```

```
plt.ylabel('Predicted Values')
plt.title('True vs Predicted Values (OLS)')
plt.legend()
plt.show()
```

[Op]

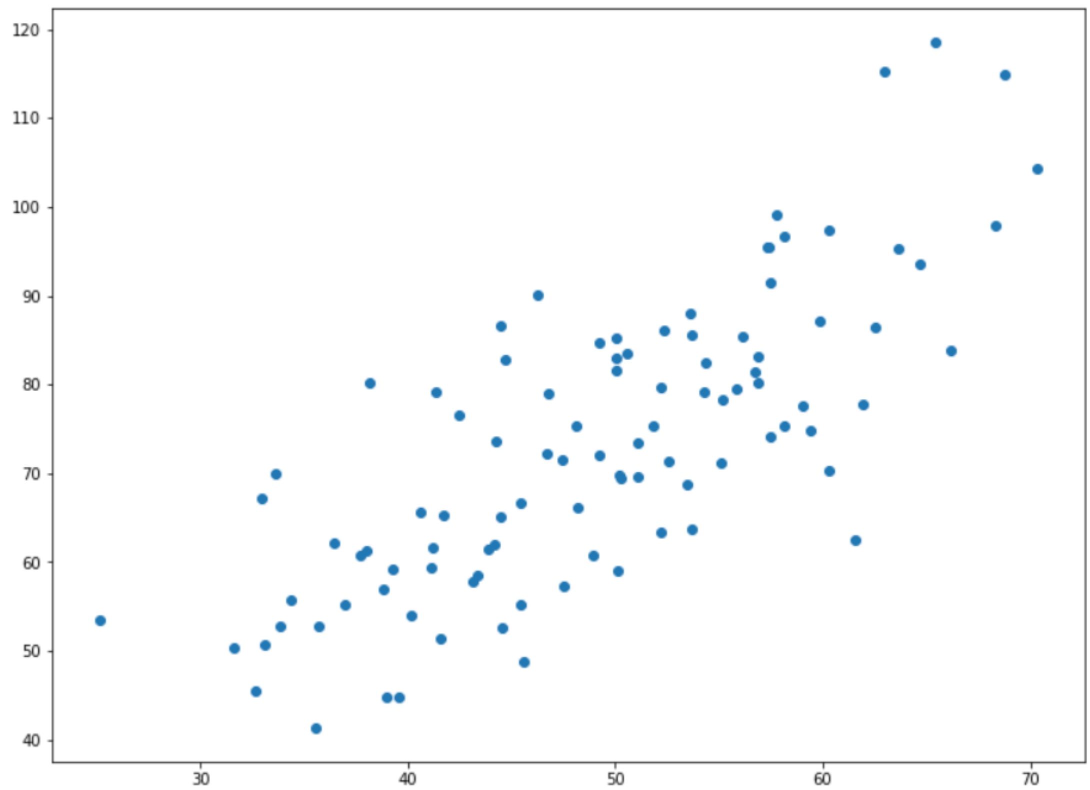


3. Write a program to perform Linear Regression using Gradient Descent Algorithm

[In]

```
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data
data = pd.read_csv('data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```

[Op]




```
[In]

# Building the model
m = 0
c = 0
L = 0.0001  # The learning Rate
epochs = 1000  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
```

```
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

print (m, c)
```

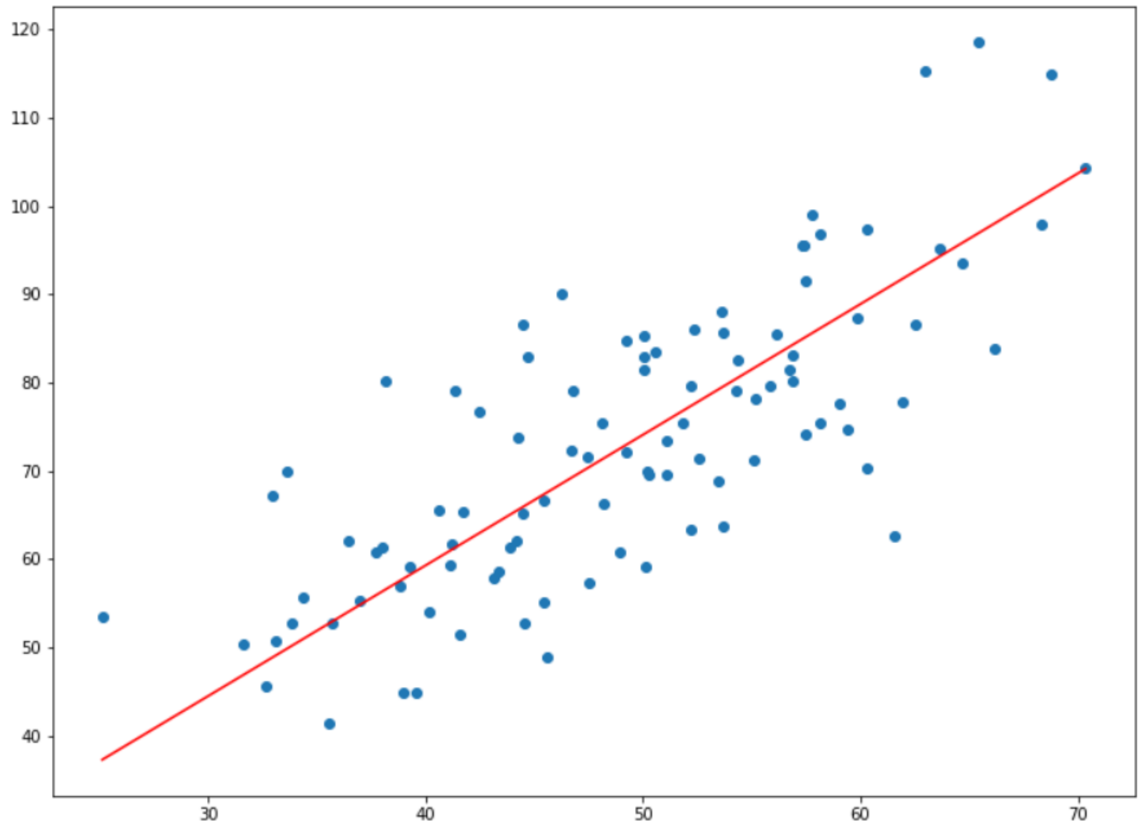
[Op]

```
1.4796491688889395 0.10148121494753726
```

[In]

```
# Making predictions
Y_pred = m*X + c

plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
# predicted
plt.show()
```

[Op]


120
110
100
90
80
70
60
50
40
30
40
50
60
70

4. Write a program to perform k-mean clustering for Customer Segment

[In]

```
import pandas as pd
df=pd.read_csv('Mall_Customers.csv') #Reading csv file
df.head()
```

[Op]

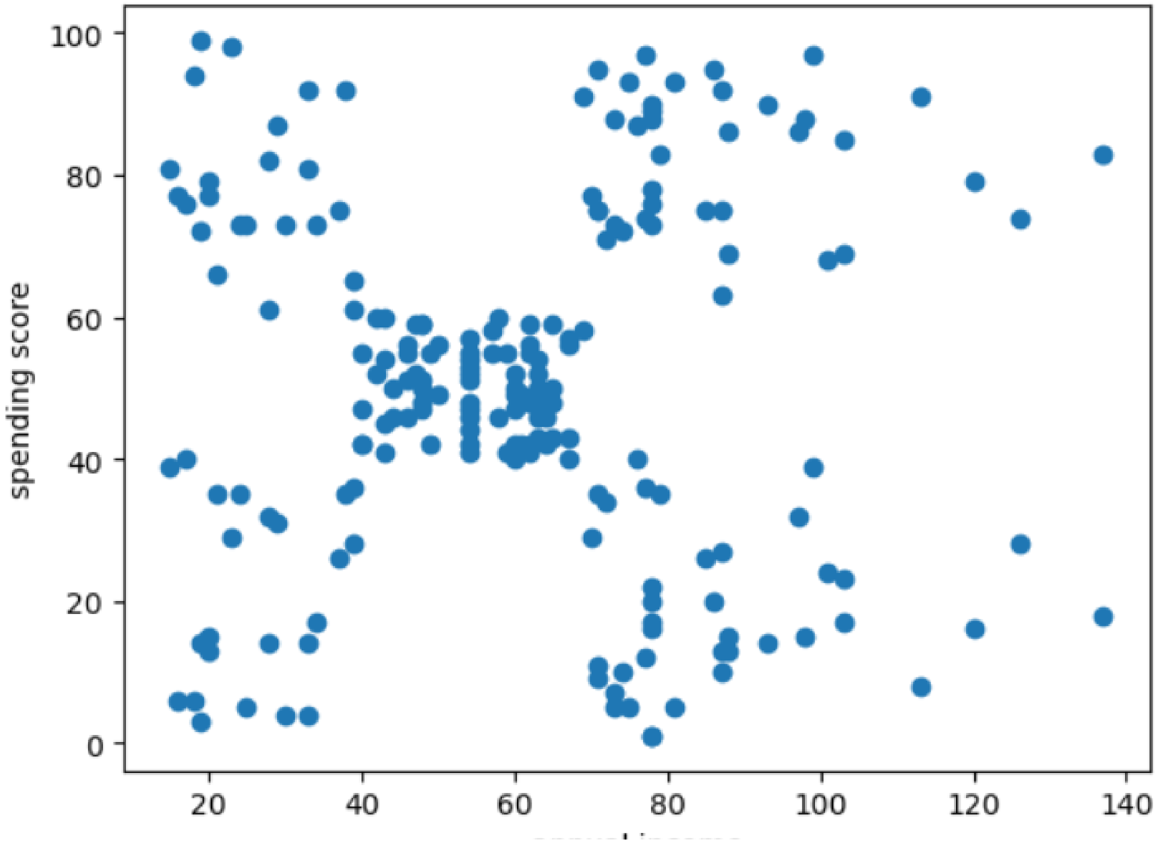
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[In]

```
#plotting graph of Annual income vs spending score
import matplotlib.pyplot as plt
income=df.iloc[:,3].values
score=df.iloc[:,4].values
plt.scatter(income,score)
plt.xlabel('annual income')
plt.ylabel('spending score')
```

```
plt.show()
```

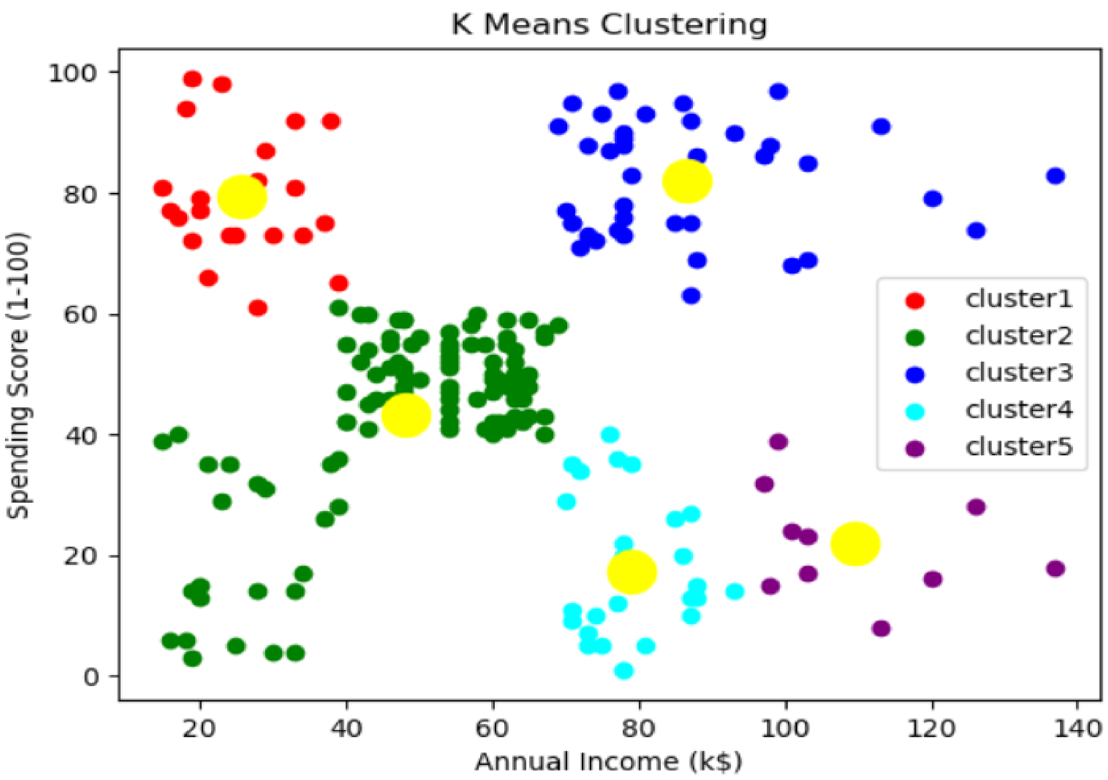
[Op]



[In]

```
x=df.iloc[:,[3,4]].values #extracting feature
#kmean clustering
from sklearn.cluster import KMeans
km=KMeans(n_clusters=5)
km
y=km.fit_predict(x)


#plotting clusters
```

```
plt.scatter(x[y==0,0],x[y==0,1],color='red',label='cluster1')
plt.scatter(x[y==1,0],x[y==1,1],color='green',label='cluster2')
plt.scatter(x[y==2,0],x[y==2,1],color='blue',label='cluster3')
plt.scatter(x[y==3,0],x[y==3,1],color='cyan',label='cluster4')
plt.scatter(x[y==4,0],x[y==4,1],color='purple',label='cluster5')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],
            color='yellow',s=300)
plt.title('K Means Clustering')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



5. Write a program to implement Density based clustering (DBSCAN)

[In]

```
import pandas as pd
df=pd.read_csv("blobs.csv") ## Read data from CSV file
df.head()
```

[Op]

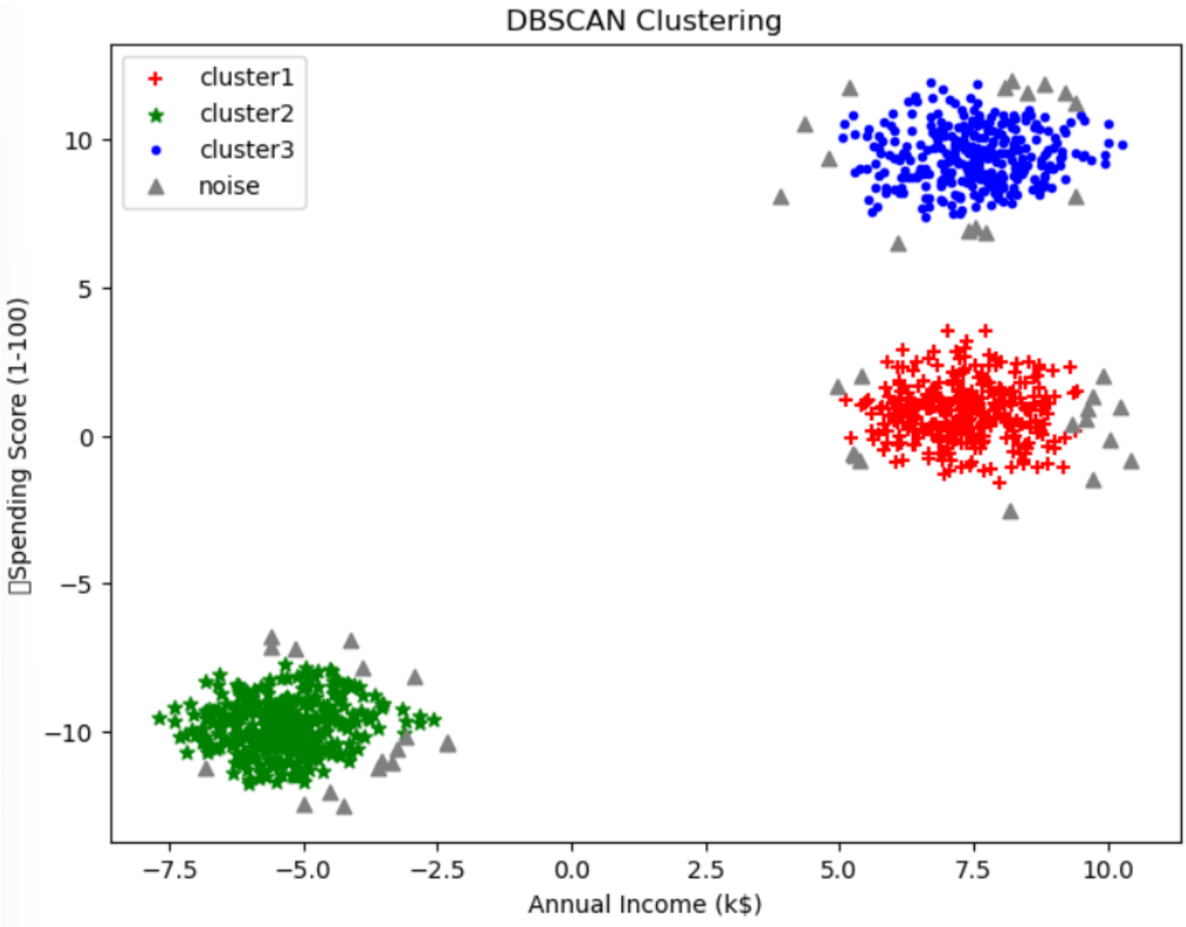
	0	1
0	8.622185	1.935796
1	-4.736710	-7.970958
2	9.621222	0.925423
3	6.162095	-0.273254
4	8.697488	-1.057452

[In]

```
# Extract the features
x=df.iloc[:,[0,1]].values

# DBSCAN clustering
from sklearn.cluster import DBSCAN
db=DBSCAN(eps=0.5,min_samples=5)
```

```
y=db.fit_predict(x)

# Plot the clusters
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(x[y==0][:,0],x[y==0][:,1],color='red',label='cluster1',
    marker='+')
plt.scatter(x[y==1][:,0],x[y==1][:,1],color='green',label='cluster2',
        marker='*')
plt.scatter(x[y==2][:,0],x[y==2][:,1],color='blue',label='cluster3',
        marker='.')
plt.scatter(x[y==-1][:,0],x[y==-1][:,1],color='grey',label='noise',
        marker='^')
plt.title('DBSCAN Clustering')
plt.xlabel('Annual Income (k$)')
plt.ylabel(' Spending Score (1-100)')
plt.legend()
plt.show()
```

[Op]


DBSCAN Clustering
cluster1
cluster2
cluster3
noise
10
5
0
−5
−10
Spending Score (1-100)
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Annual Income (k$)

6. Write a program to perform hierarchical clustering for customer segment

[In]

```
import pandas as pd
df=pd.read_csv('Mall_Customers.csv') #Reading csv file
df.head()
```

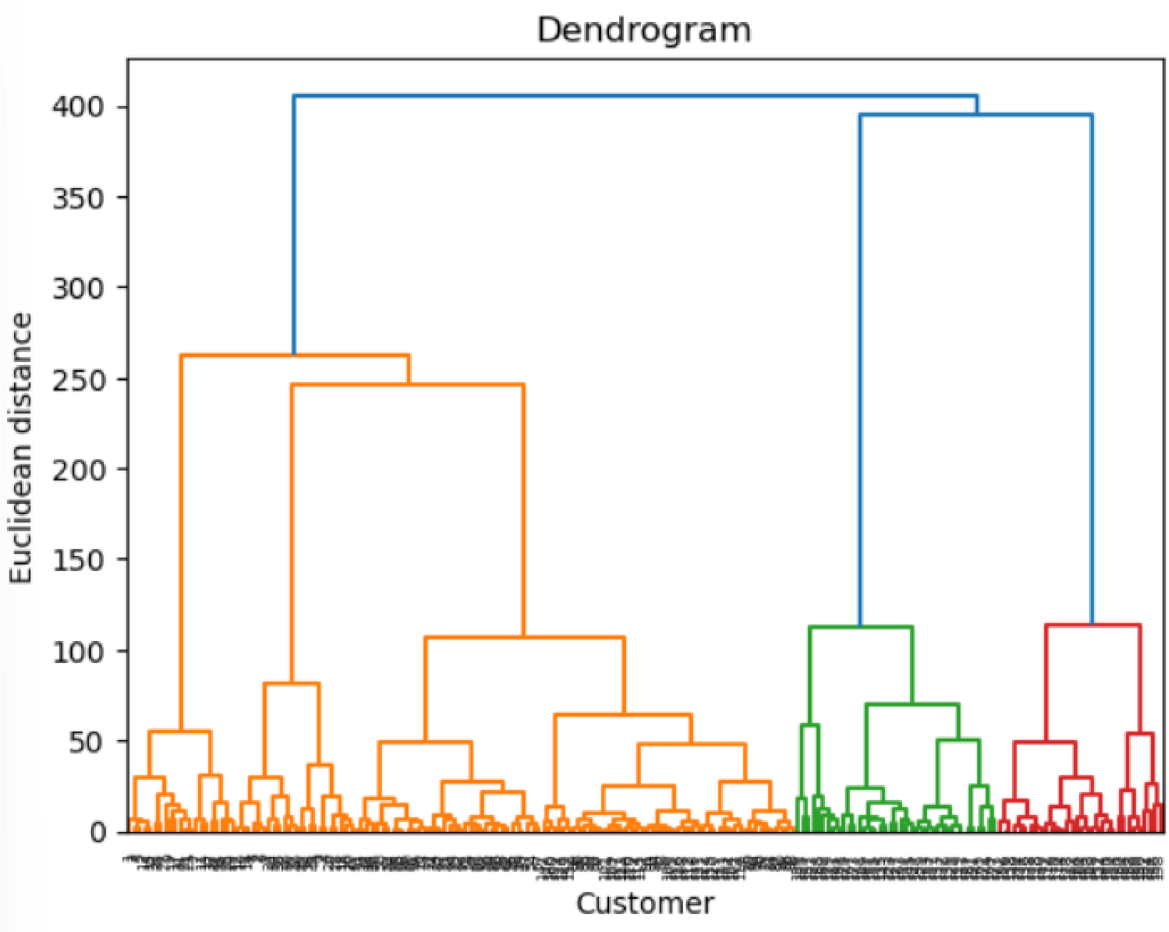
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[In]

```
x=df.iloc[:,[3,4]].values #extracting features

#Dendrogram
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
dendrogram=sch.dendrogram(sch.linkage(x,method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customer')
plt.ylabel('Euclidean distance')
```

plt.show()

[Op]



[In]

#Agglomerative clustering

from sklearn.cluster import AgglomerativeClustering

hc=AgglomerativeClustering(n_clusters=5,linkage='ward')

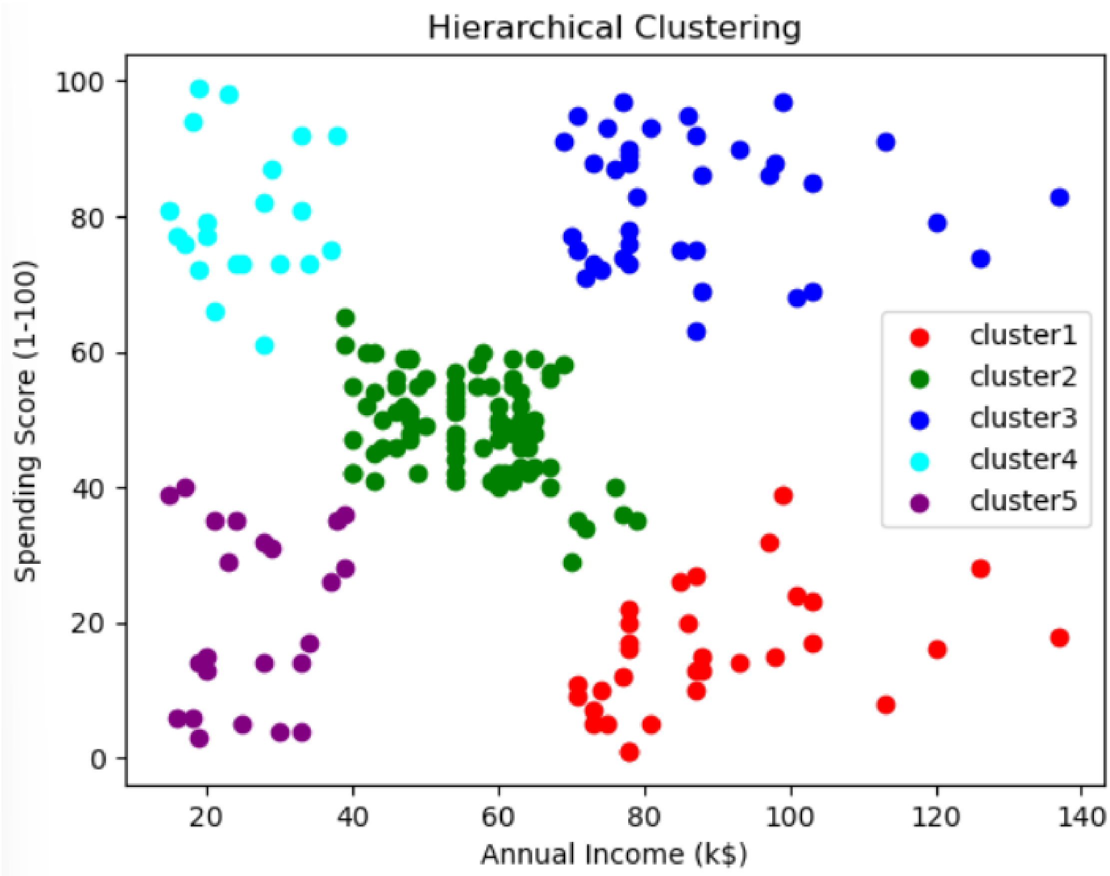
hc

y=hc.fit_predict(x)

#plotting clusters

```
plt.scatter(x[y==0,0],x[y==0,1],color='red',label='cluster1')
plt.scatter(x[y==1,0],x[y==1,1],color='green',label='cluster2')
plt.scatter(x[y==2,0],x[y==2,1],color='blue',label='cluster3')
plt.scatter(x[y==3,0],x[y==3,1],color='cyan',label='cluster4')
plt.scatter(x[y==4,0],x[y==4,1],color='purple',label='cluster5')
plt.title('Hierarchical Clustering')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

[Op]



7. Write a program to implement Decision tree using ID3 algorithm

[In]

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split


df = pd.read_csv('customer_churn_dataset-testing-master.csv')

df.head()
```

[Op]

	CustomerID	Age	Gender	Tenure	Usage Frequency	Support Calls	Payment Delay	Subscription Type	Contract Length	Total Spend	Last Interaction	Churn
0	1	22	Female	25	14	4	27	Basic	Monthly	598	9	1
1	2	41	Female	28	28	7	13	Standard	Monthly	584	20	0
2	3	47	Male	27	10	2	29	Premium	Annual	757	21	0
3	4	35	Male	9	12	5	17	Premium	Quarterly	232	18	0
4	5	53	Female	58	24	9	2	Standard	Annual	533	18	0

[In]

```
df = df.drop(['CustomerID'], axis = 1)
df.head()


def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)

    return dataframe_series


df = df.apply(lambda x: object_to_int(x))
df.head()
```

[Op]

	Age	Gender	Tenure	Usage Frequency	Support Calls	Payment Delay	Subscription Type	Contract Length	Total Spend	Last Interaction	Churn
0	22	0	25	14	4	27	0	1	598	9	1
1	41	0	28	28	7	13	2	1	584	20	0
2	47	1	27	10	2	29	1	0	757	21	0
3	35	1	9	12	5	17	1	2	232	18	0
4	53	0	58	24	9	2	2	0	533	18	0

[In]

```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
0.30, random_state = 40, stratify=y)

from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score

model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

[Op]

```
Accuracy: 0.9987573137265054
```

[In]

```
print(export_text(model, feature_names=list(X.columns)))
```

[Op]

```
|--- Payment Delay <= 15.50
|   |--- Usage Frequency <= 5.50
|   |   |--- Age <= 50.50
|   |   |   |--- Usage Frequency <= 2.
```

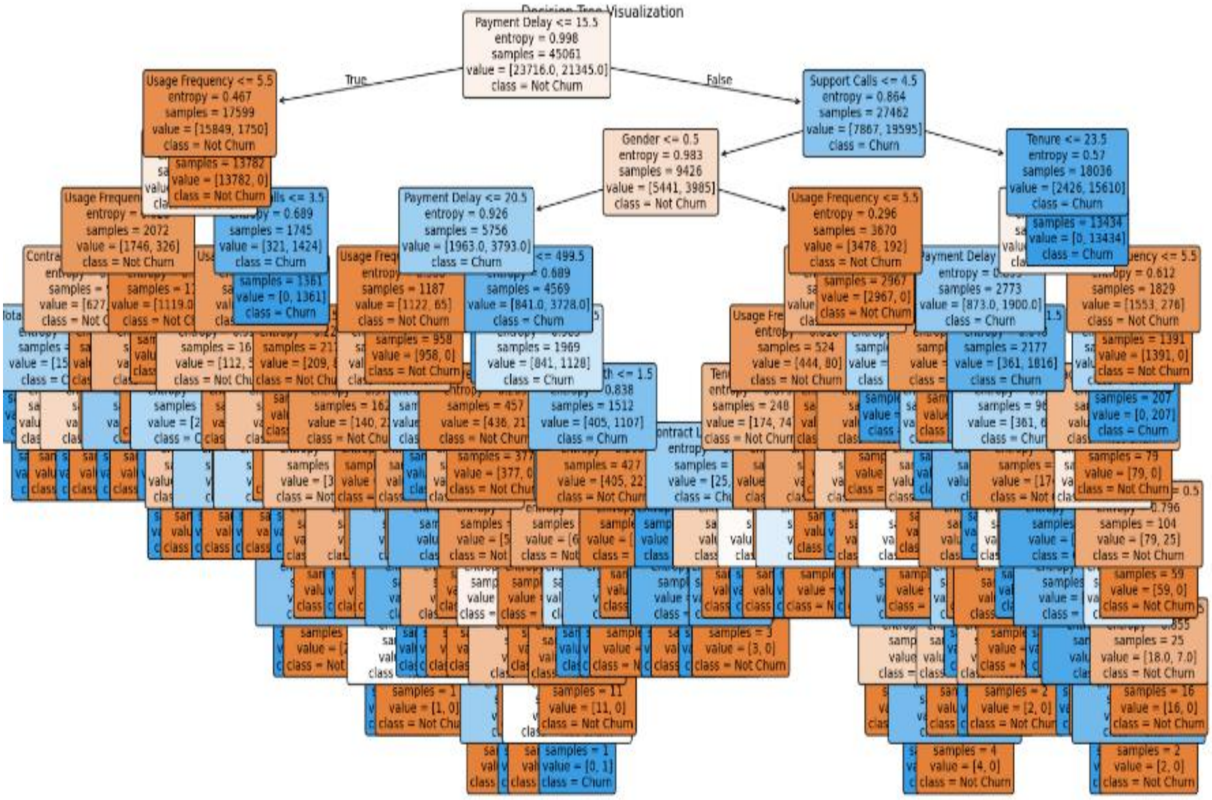
```
.          . .        .        .        .
.          . .        .        .        .
.          . .        .        .        .
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- Age >  50.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- Usage Frequency >  5.50
|   |   |   |   |   |--- class: 0
|   |   |--- Tenure >  23.50
|   |   |   |--- class: 1
```

[In]

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
# Plot the decision tree
plt.figure(figsize=(20, 10))  # Adjust the figure size as needed
plot_tree(
    model,
    feature_names=list(X.columns),  # Names of your features
    class_names=['Not Churn', 'Churn'],  C
    filled=True,  # Color nodes by class
    rounded=True,  # Round corners of nodes
    fontsize=10  # Set font size
)
```

```python
plt.title("Decision Tree Visualization")  # Add a title to the plot
plt.show()
```

[Op]



8. Write a program to implement Support vector machine for digit recognition

[In]

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

train_df = pd.read_csv("train.csv")

test_df = pd.read_csv("test.csv")


X = train_df.drop('label', axis=1)

y = train_df['label']


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 = 100)


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# fit_transform use to do some calculation and then do
transformation

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

```
from sklearn.svm import SVC
rbf_model = SVC(kernel='linear')
rbf_model.fit(X_train, y_train)

y_rbf_pred = rbf_model.predict(X_test)

print('Predictad Values :\n ',y_rbf_pred[10:15])
print ('Actual Values :\n',y_test[10:15])
```

[Op]

```
Predictad Values :
  [3 9 6 7 1]
Actual Values :
 24273    3
32691    9
34526    6
11625    7
6614     1
Name: label, dtype: int64
```

[In]

```
from sklearn import metrics
acc_rbf= metrics.accuracy_score(y_test, y_rbf_pred)
print("accuracy:","{:.2f}".format(acc_rbf*100),"%")
```

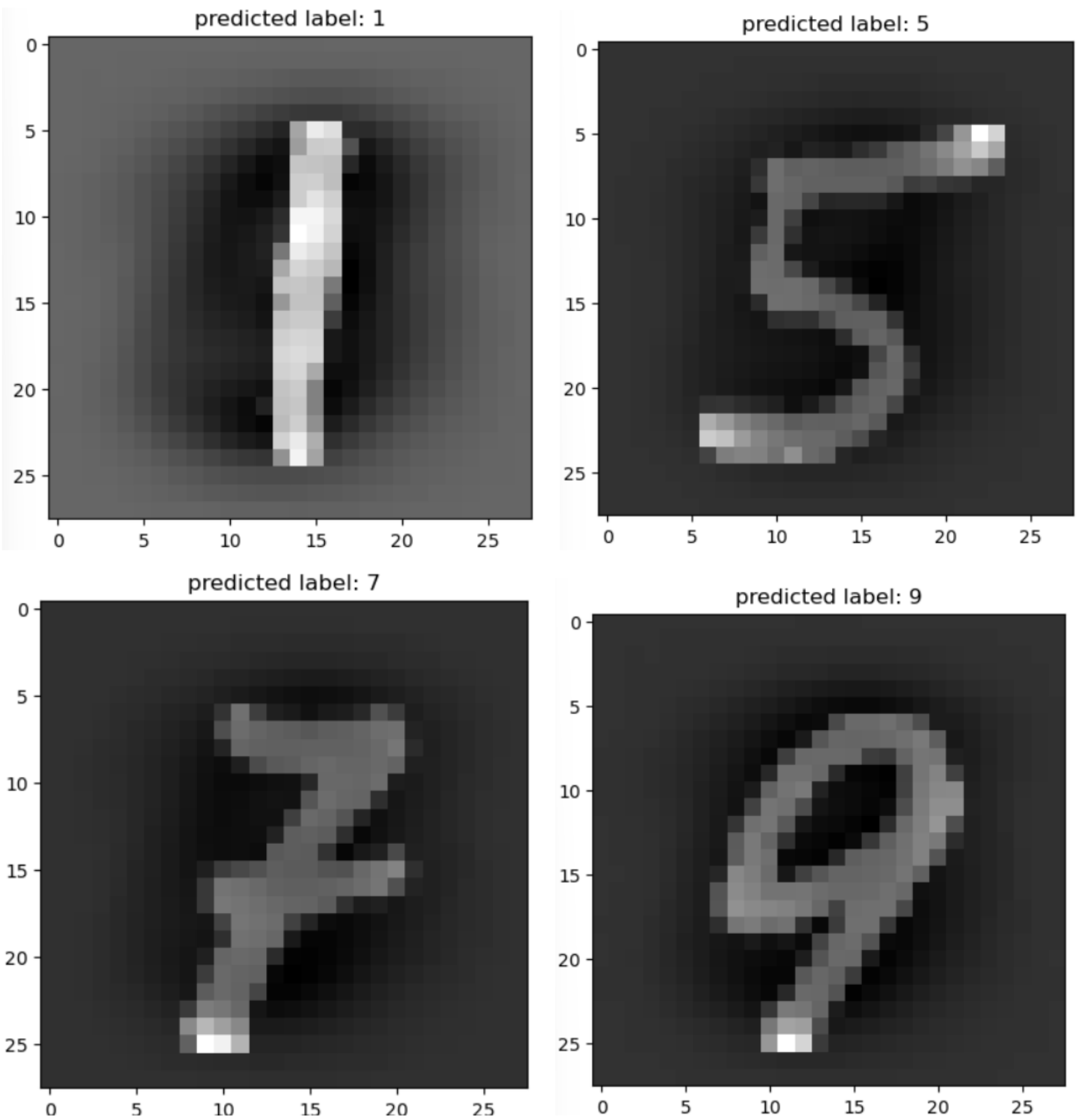
[Op]

accuracy: 91.46 %

[In]

```
for i in (np.random.randint(0,270,4)):
    two_d = (np.reshape(X_test[i], (28, 28)))
    plt.title('predicted label: {0}'. format(y_rbf_pred[i]))
    plt.imshow(two_d, cmap='gray')
    plt.show()
```

[Op]



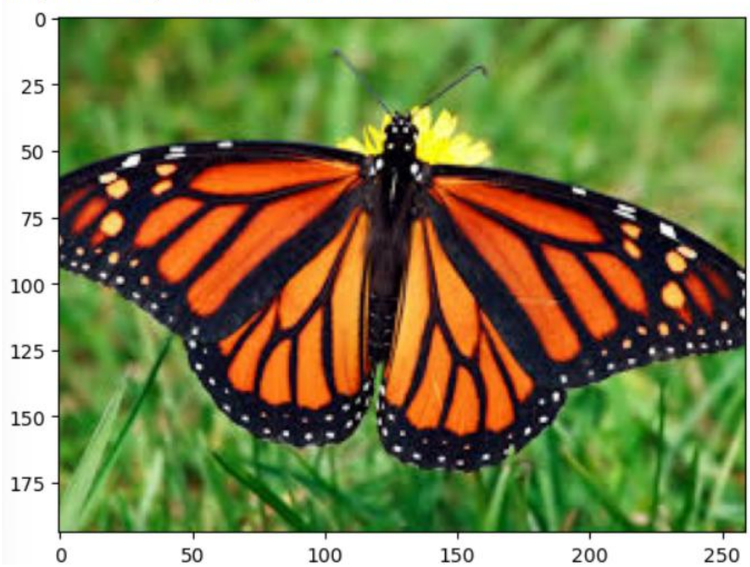
9. Write a program to perform image segmentation using k- mean clustering

[In]

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline

# Read in the image
image = cv2.imread('monarch.jpg')
# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

[Op]

```
<matplotlib.image.AxesImage at 0x1856191eab0>
```

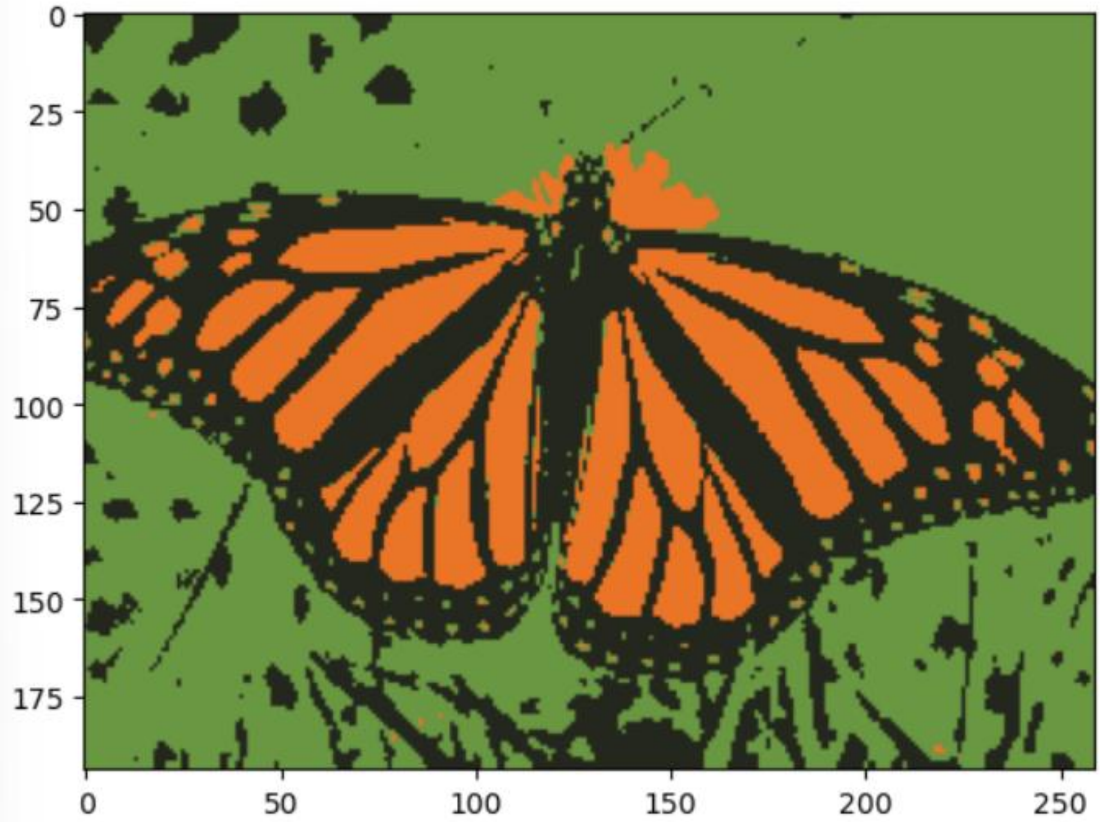


[In]

```
# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3))
# Convert to float type
pixel_vals = np.float32(pixel_vals)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)
k = 3

retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
# convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))
plt.imshow(segmented_image)
```

[Op]

<matplotlib.image.AxesImage at 0x18562da8290>



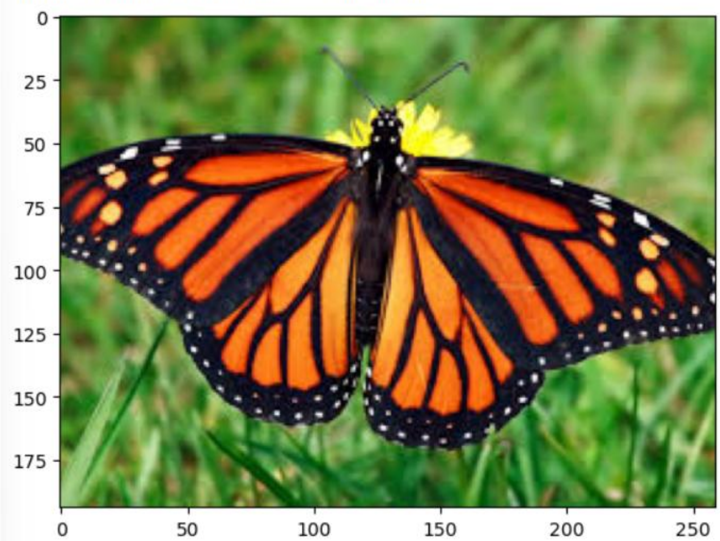
10. Write a program to perform image segmentation using DBSCAN

[In]

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
 %matplotlib inline


# Read in the image
image = cv2.imread('monarch.jpg')
# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

[Op]

<matplotlib.image.AxesImage at 0x28944e8b7a0>



[In]

```
# Reshaping the image into a 2D array of pixels and 3 color values (RGB)

pixel_vals = image.reshape((-1,3))

# Convert to float type

pixel_vals = np.float32(pixel_vals)


from sklearn.cluster import DBSCAN

# Perform DBSCAN clustering

# The maximum distance between two samples to be considered in the same neighborhood

eps = 5

# The minimum number of points needed to form a cluster

min_samples = 50

clustering = DBSCAN(eps=eps, min_samples=min_samples, metric='euclidean').fit(pixel_vals)

labels = clustering.labels_

# Get unique labels and assign random colors to each cluster

unique_labels = np.unique(labels)

colors = np.random.randint(0, 255, size=(len(unique_labels), 3), dtype=np.uint8)

segmented_data = np.array([colors[label] if label != -1 else [0, 0, 0] for label in labels])  # Noise as black


# Reshape back to original image dimensions
```

```
segmented_image = segmented_data.reshape(image.shape)
# Display the segmented image
plt.imshow(segmented_image)
plt.axis('off')
plt.show()
```

[Op]