Artificial Intelligence and Machine Learning

Subject Code: MCAL21

A Project Report Submitted in
Fulfilment of the Degree of

MASTER

In

COMPUTER APPLICATION

Year 2023-2024

By

Mr. Ketan Ravindra Daki

(Application Id:- 78749)

Semester-II

Under the Guidance of

Prof. Prashant Londhe

Institute of Distance and Open Learning
Vidya Nagari, Kalina, Santacruz East – 400098.
University of Mumbai

PCP Center

[Satish Pradhan Dyanasadhana College, Thane]

Institute of Distance and Open
Learning Vidya Nagari, Kalina,
Santacruz East – 400098.

CERTIFICATE

This to certify that, **"KETAN RAVINDRA DAKI"** appearing
**Master's in computer application (Semester II) Application
ID: 78749** has satisfactory completed the prescribed practical of
MCAL21 - Artificial Intelligence And Machine Learning as
laid down bythe University of Mumbai for the academic year
2023-24.

_____
–

_____ _____

Teacher In Charge   External Examiner   Coordinator
M.C.A

Date:
Place:

INDEX

Practical 01

Aim: Introduction to python programming: learn the different libraries.

a) Generate normally distributed random numbers using NumPy.

Code:
```
import numpy as np
# numpy. random.normal() method
r = np. random.normal(size=6)
# printing numbers
print(r)
```

Output:

```
>>
    = RESTART: C:/Users/vsawant/RandomNumber.py
    [ 1.88826877  0.11510576 -0.20527798  1.07856708  1.53998143  0.41321256]
>>
```

b) Create data frame using pandas.

Code:
```
import pandas as pd
# Calling DataFrame constructor
df = pd.DataFrame()
print(df)
# list of strings
lst = ['Madhu','For','Madhusri','is','portal','for','students']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
#Import CSV
df = pd.read_csv('data.csv')
print(df)
```

Output:

```
>>>
= RESTART: C:\Users\vsawant\prac2.py
Empty DataFrame
Columns: []
Index: []
          0
0      Madhu
1        For
2   Madhusri
3         is
4     portal
5        for
6   students
   Identifier First name Last Name
0      901242     Rachel    Booker
1      207074     Lauren      Grey
2      408129      Craig   Johnson
3      934600       Mary   Jenkins
4      507916      Jamie     Smith
```
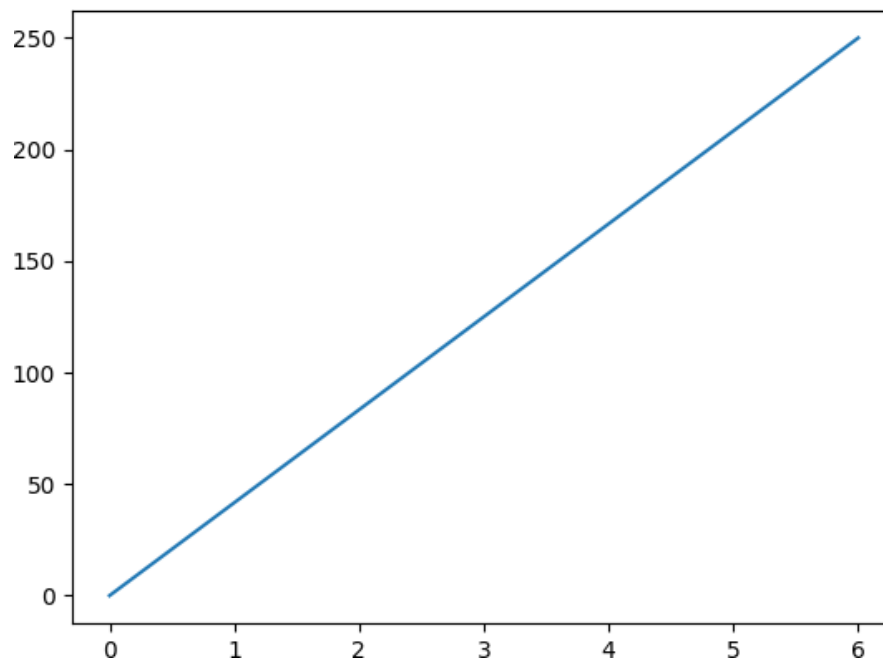
c) Plot data points using matplotlib

Code:
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

Output:

d) import sklearn and print features of iris dataset.

Code:

```python
from sklearn.datasets import load_iris
iris = load_iris()
A= iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of A:\n", A[:10])
```

Output:

```
======================= RESTART: C:/Users/vsawant/Prac4.py =======================
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of A:
 [[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]]
>>
```
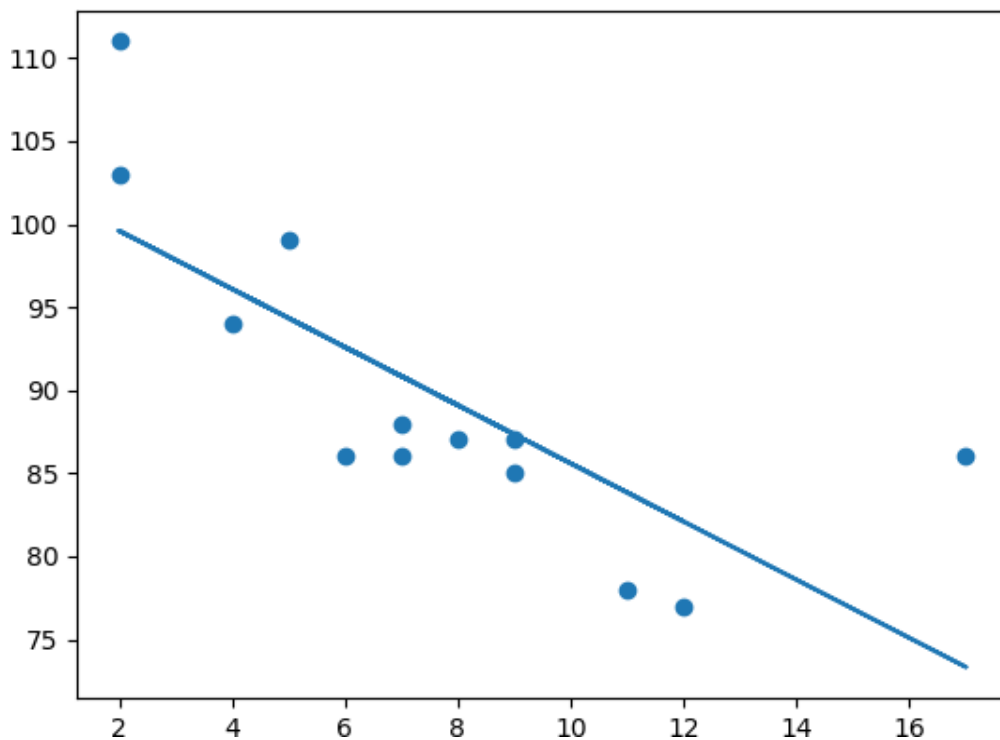
Practical 02

Aim: Supervised learning

a) Implement the Linear regression model

Code:

```
import matplotlib.pyplot as plt
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Output:

b) Implement Logistic regression model.

Code:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# For binary classification, let's consider only two classes (0 and 1)
X = X[y != 2]
y = y[y != 2]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

```
Accuracy: 1.0
Confusion Matrix:
[[12  0]
 [ 0  8]]
```

```python
# Plot decision boundary (works only for 2D datasets)
if X_train.shape[1] == 2:
    h = .02  # Step size in the mesh
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
```
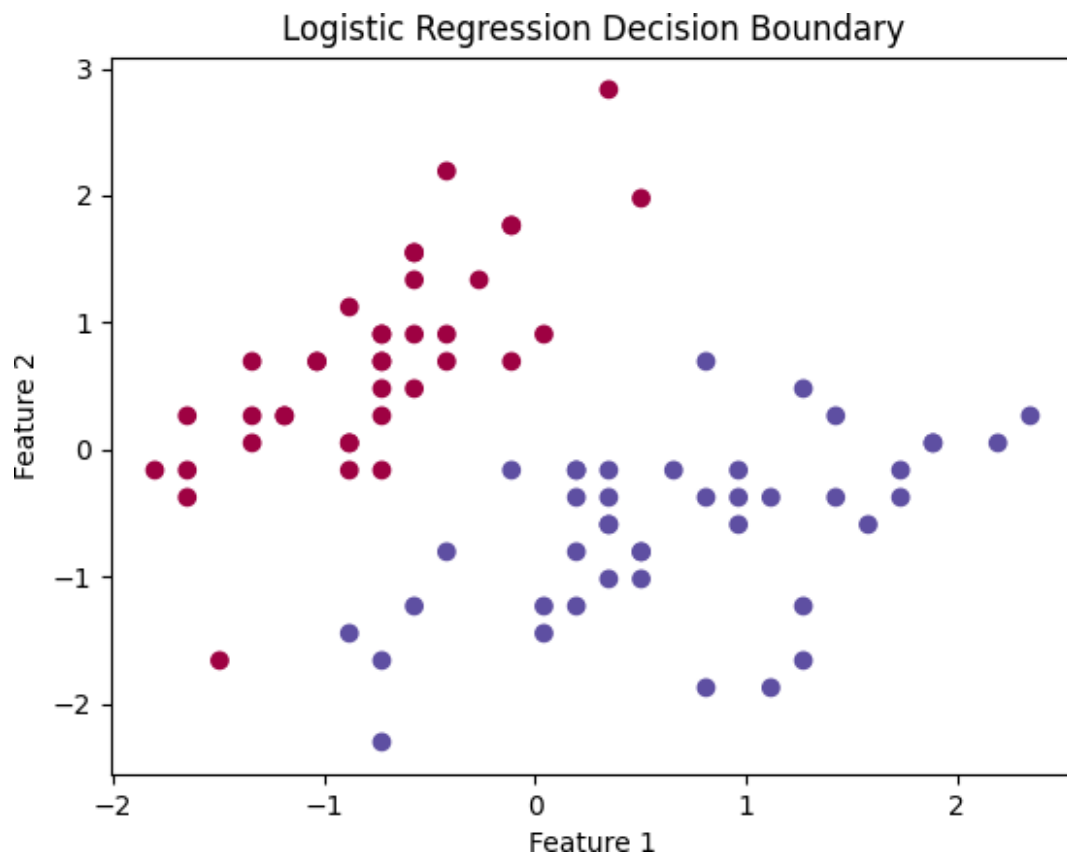
```
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)

# Plot the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.Spectral)
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Output:

Practical 03

Aim: Supervised Learning

a) **K-nearest Neighbours (KNN) Classification Model.**

Code:
```
Implement import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
X, y = make_blobs(n_samples = 500, n_features = 2, centers = 4,cluster_std = 1.5, random_state = 4)
plt.style.use('seaborn')
plt.figure(figsize = (10,10))
plt.scatter(X[:,0], X[:,1], c=y, marker= '.',s=100,edgecolors='black')
plt.show()
```
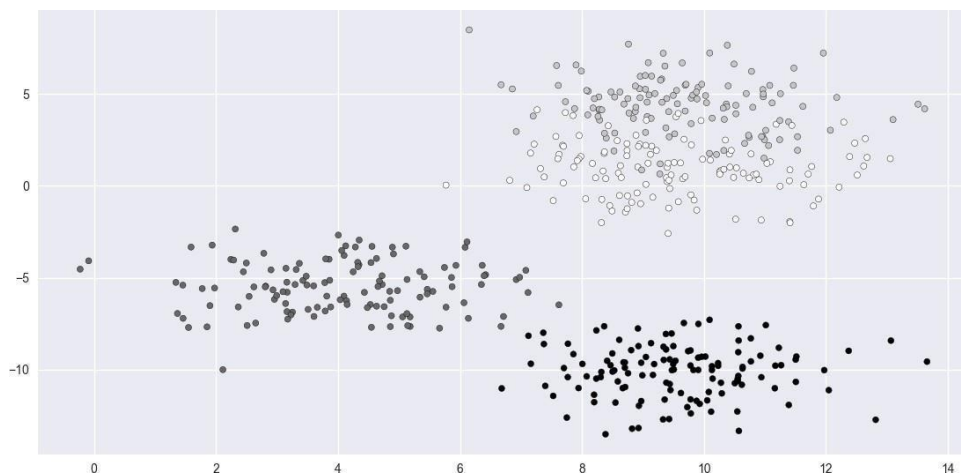


```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)

y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)


plt.figure(figsize = (15,5))
```

```
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=5", fontsize=20)

plt.subplot(1,2,2)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=1", fontsize=20)
plt.show()
```
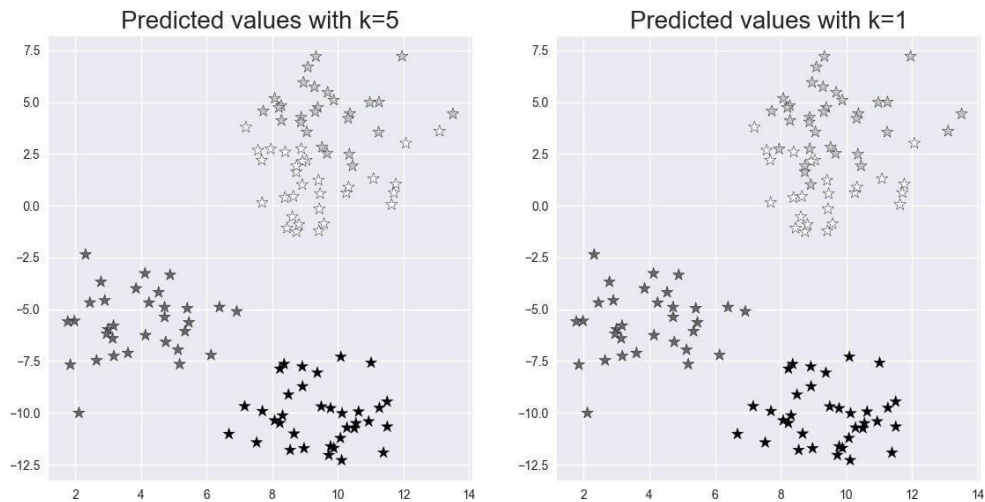

Accuracy with k=5 93.60000000000001
Accuracy with k=1 90.4

Practical 04

Aim: Features and Extraction

a) Identify the features in Iris dataset that are strongly correlated

import numpy as np

import pandas as pd

from sklearn import datasets

import matplotlib.pyplot as plt

iris=datasets.load_iris()

iris.data

iris.feature_names

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

cov_data=np.corrcoef(iris.data.T)

cov_data

```
array([[ 1.        , -0.11756978,  0.87175378,  0.81794113],
       [-0.11756978,  1.        , -0.4284401 , -0.36612593],
       [ 0.87175378, -0.4284401 ,  1.        ,  0.96286543],
       [ 0.81794113, -0.36612593,  0.96286543,  1.        ]])
```
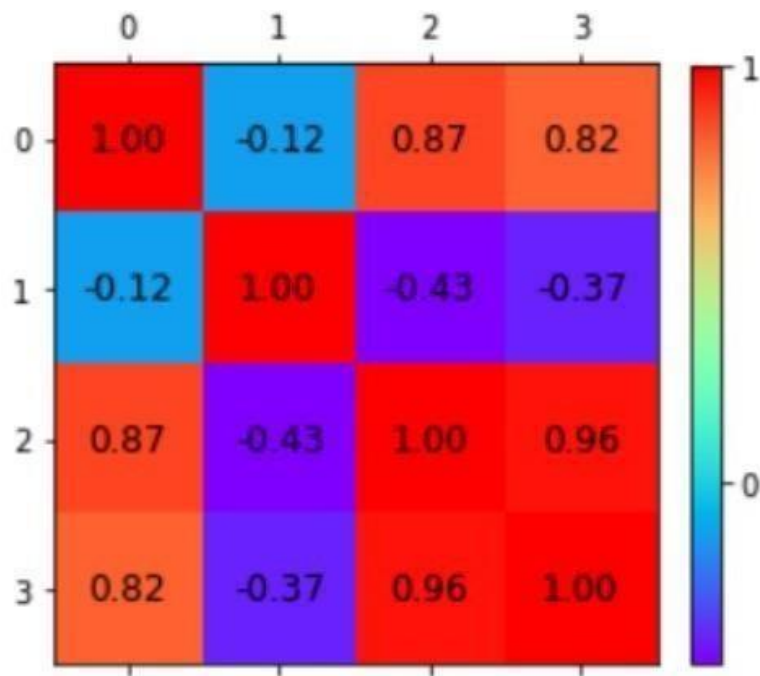
img=plt.matshow(cov_data,cmap=plt.cm.rainbow)

plt.colorbar(img,ticks=[-1,0,1],fraction=0.045)

for x in range(cov_data.shape[0]):

  for y in range(cov_data.shape[1]):

    plt.text(x,y,"%0.2f"%cov_data[x,y],size=12,color='black',ha="center",va="center")

    plt.show()

Output:

b) Implementation of principal component analysis (PCA) on the Iris dataset with Python

Code:

```
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d
from sklearn import datasets
iris = datasets.load_iris()
from sklearn.decomposition import PCA
fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azim=110)

X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=iris.target,
    s=40,
)

ax.set_title("First three PCA dimensions")
ax.set_xlabel("1st Eigenvector")
ax.xaxis.set_ticklabels([])
ax.set_ylabel("2nd Eigenvector")
ax.yaxis.set_ticklabels([])
ax.set_zlabel("3rd Eigenvector")
ax.zaxis.set_ticklabels([])

plt.show()
```

Output:

First three PCA dimensions
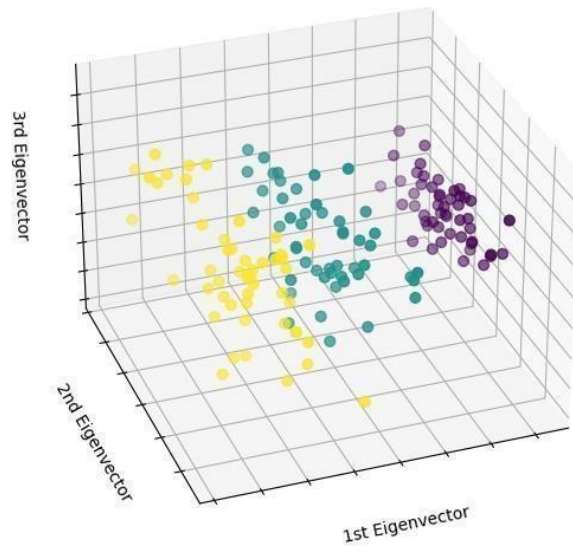
Practical 05

Aim: Unsupervised Learning

a) Implement the K-Means clustering method

Code:

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline


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

```
      Name   Age   Income($)
0      Rob    27      70000
1  Michael    29      90000
2    Mohan    29      61000
3   Ismail    28      60000
4     Kory    42     150000
```

```
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.show()
```
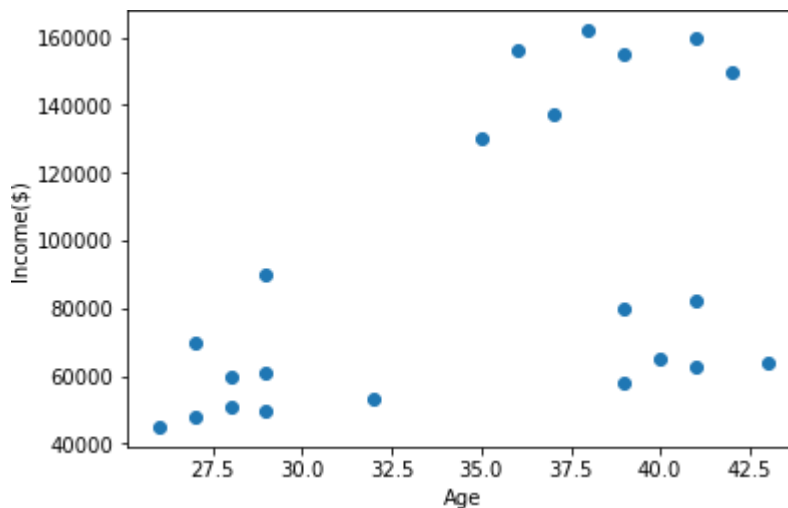


```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
array([2, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0])
```

```
df['cluster']=y_predicted
df.head()
```

```
        Name  Age  Income($)   cluster
0       Rob   27      70000         2
1    Michael  29      90000         2
2     Mohan   29      61000         0
3    Ismail   28      60000         0
4      Kory   42     150000         1
```

km.cluster_centers_

```
array([[3.29090909e+01, 5.61363636e+04],
       [3.82857143e+01, 1.50000000e+05],
       [3.40000000e+01, 8.05000000e+04]])
```

```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
plt.show()
```
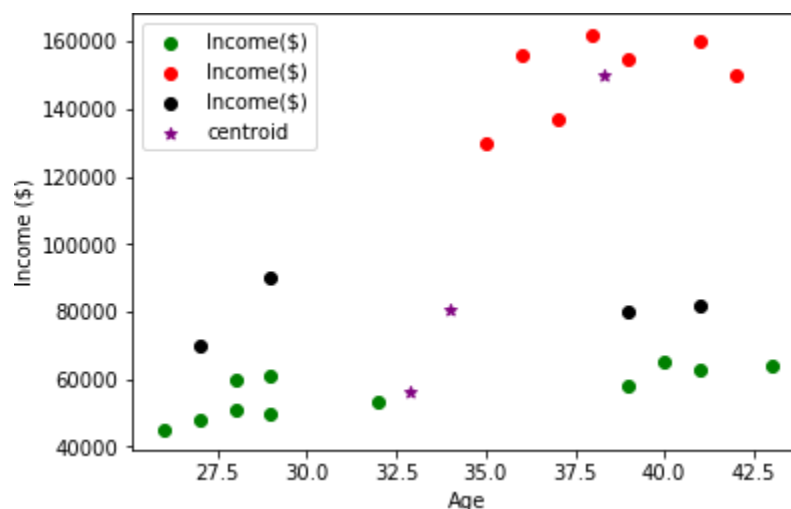


```
scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])
```

```
        Name       Age   Income($)   cluster
0       Rob   0.058824    0.213675         2
1    Michael  0.176471    0.384615         2
2     Mohan   0.176471    0.136752         0
3    Ismail   0.117647    0.128205         0
4      Kory   0.941176    0.897436         1
```

```
plt.scatter(df.Age,df['Income($)'])
```
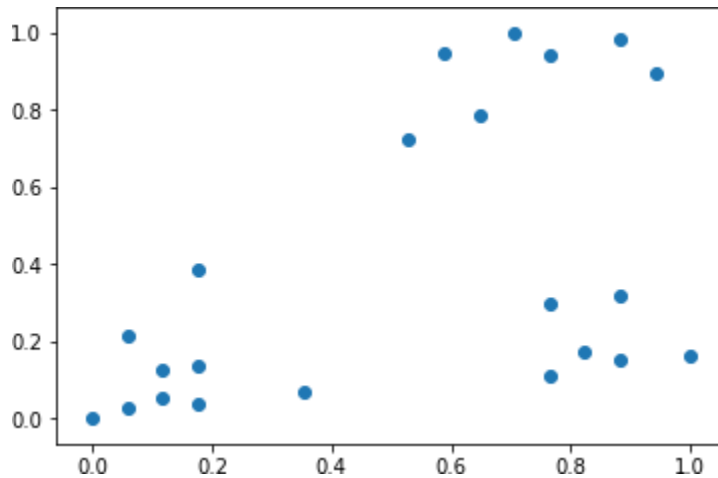


```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
```

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

```
df['cluster']=y_predicted
df.head()
      Name       Age    Income($)   cluster
0      Rob   0.058824    0.213675         1
1  Michael   0.176471    0.384615         1
2    Mohan   0.176471    0.136752         1
3   Ismail   0.117647    0.128205         1
4     Kory   0.941176    0.897436         2
```

```
km.cluster_centers_
```

```
array([[0.85294118, 0.2022792 ],
       [0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ]])
```

```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
plt.show()
```
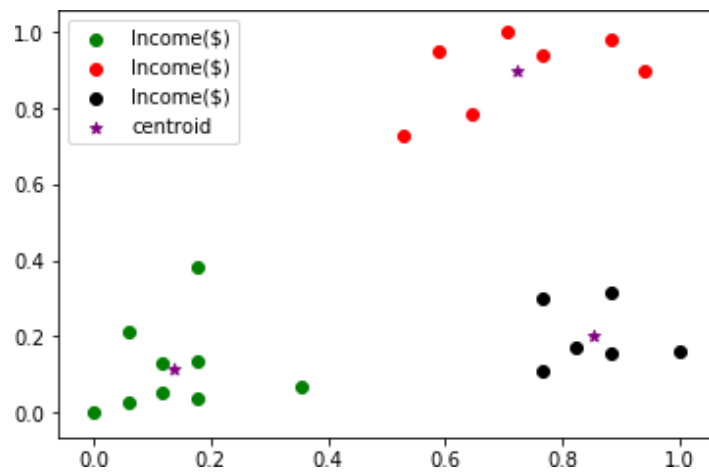
Output:

Practical 06

Aim: Classify the data using Support vector machine

Code:

```
# importing scikit learn with make_blobs

from sklearn.datasets import make_blobs

# creating datasets X containing n_samples

# Y containing two classes

X, Y = make_blobs(n_samples=500, centers=2,random_state=0, cluster_std=0.40)

import matplotlib.pyplot as plt

import numpy as np

# plotting scatters

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');

# creating linspace between -1 to 3.5

xfit = np.linspace(-1, 3.5)

# plotting scatter

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

# plot a line between the different sets of data

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:

        yfit = m * xfit + b

        plt.plot(xfit, yfit, '-k')

        plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',

        color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);

plt.show()
```

Output:
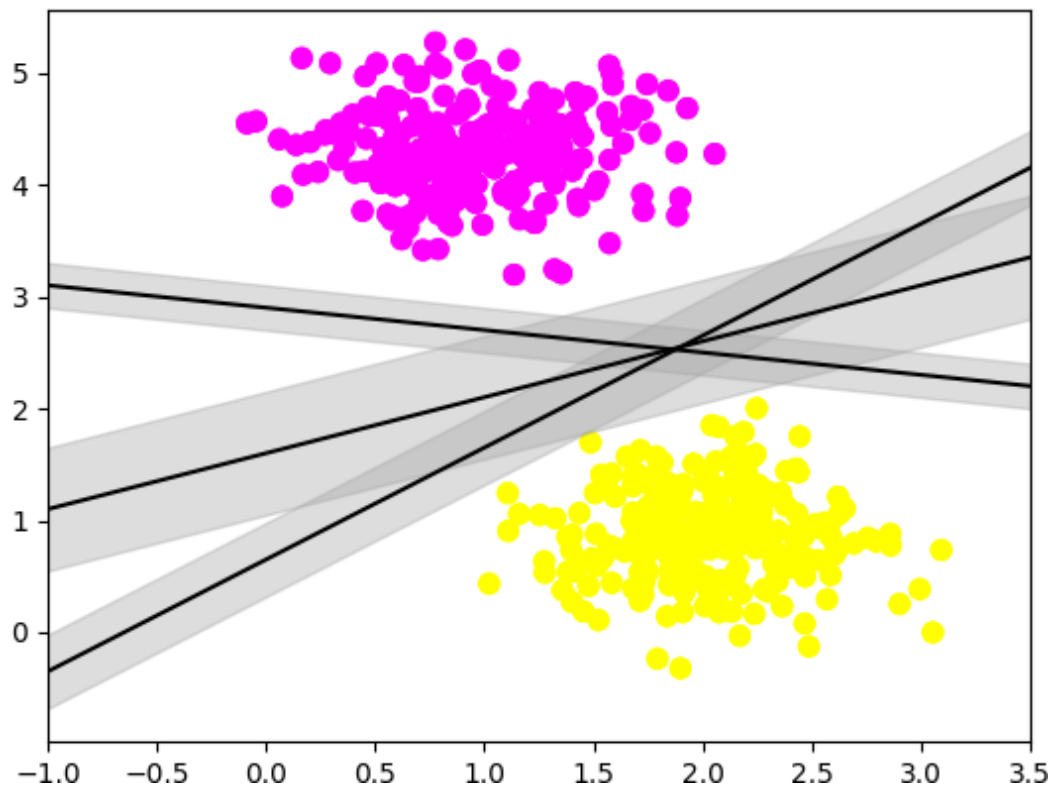
Practical 07

Aim: Implement the decision tree using python

Code:

import pandas as pd

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

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

d = {'UK': 0, 'USA': 1, 'N': 2}

df['Nationality'] = df['Nationality'].map(d)

d = {'YES': 1, 'NO': 0}

df['Go'] = df['Go'].map(d)

print(df)

```
    Age  Experience  Rank  Nationality  Go
0    36          10     9            0   0
1    42          12     4            1   0
2    23           4     6            2   0
3    52           4     4            1   0
4    43          21     8            1   1
5    44          14     5            0   0
6    66           3     7            2   1
7    35          14     9            0   1
8    52          13     7            2   1
9    35           5     9            2   1
10   24           3     5            1   0
11   18           3     7            0   1
12   45           9     9            0   1
```

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]

y = df['Go']

print(X)

```
      Age  Experience  Rank  Nationality
0      36          10     9            0
1      42          12     4            1
2      23           4     6            2
3      52           4     4            1
4      43          21     8            1
5      44          14     5            0
6      66           3     7            2
7      35          14     9            0
8      52          13     7            2
9      35           5     9            2
10     24           3     5            1
11     18           3     7            0
12     45           9     9            0
```

print(y)

```
0      0
1      0
2      0
3      0
4      1
5      0
6      1
7      1
8      1
9      1
10     0
11     1
12     1
```

dtree = DecisionTreeClassifier()

dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)

Output:

```
Rank <= 6.5
gini = 0.497
samples = 13
value = [6, 7]
```

```
gini = 0.0
samples = 5
value = [5, 0]
```

```
Nationality <= 0.5
gini = 0.219
samples = 8
value = [1, 7]
```

```
Age <= 35.5
gini = 0.375
samples = 4
value = [1, 3]
```

```
gini = 0.0
samples = 4
value = [0, 4]
```

```
gini = 0.0
samples = 2
value = [0, 2]
```

```
Experience <= 9.5
gini = 0.5
samples = 2
value = [1, 1]
```

```
gini = 0.0
samples = 1
value = [0, 1]
```

```
gini = 0.0
samples = 1
value = [1, 0]
```