Panipat Institute Of Engineering & Technology

Samalkha, Panipat

Department of Computer Science & Engineering-Emerging Technology

Practical Lab: Applied Machine Learning Lab

PC-CS-AIML-312A

Submitted To:
Dr. Anju Gandhi
Professor
CSE-ET Department

Submitted By:
Shweta Jain
B.Tech. CSE-ET(AIML)
Semester-6th
Roll No. - 2820270
Section-D1

Affiliated to

INDEX

S.No.	Name of the Practical	Page No.	Date	Signature
1.	Write a program for Linear Regression in Python.			
2.	Write a program for Logistic Regression in Python.			
3.	Write a program to implement decision tree for classification.			
4.	Write a program to implement Random Forests For Classification			
5.	Write a program to implement KNN Algorithm			
6.	Write a program to implement Naive Bayes Algorithm.			
7.	Write a program to implement Support Vector Machine (SVM).			
8.	Write a program to Time Series Analysis Stock Price.			
9.	Write a program for Principal Component Analysis (PCA).			
10.	Write a program to implement K-Means Clustering Algorithm.			

PRACTICAL-1

Write a program for Linear Regression in Python.

PROGRAM:

#importing libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

 # Loading the data

```
car_data = pd.read_csv('car_data.csv')
```

#Getting info of first five rows

```
car_data.head()
```

Output:

Out[4]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

 # Getting some information about the dataset

```
car_data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Seller_Type    301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

#encoding columns

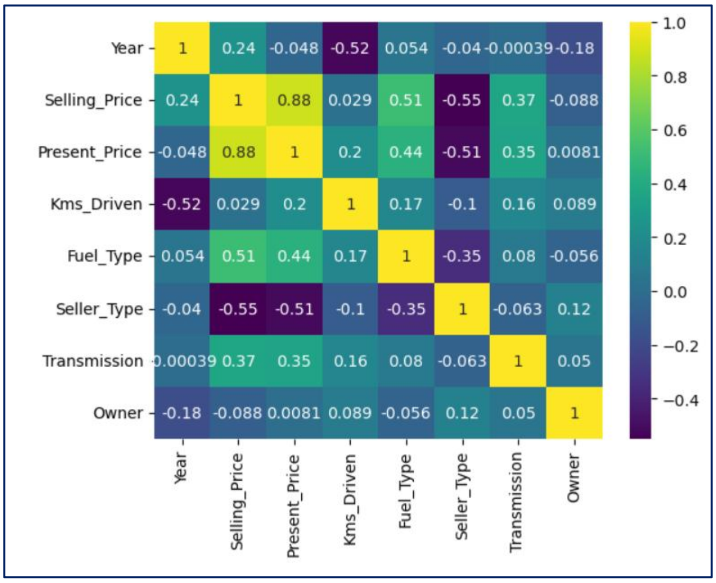
car_data.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True) car_data.replace({'Seller_Type':{'Dealer':0,'Individual':1}},inplace=True) car_data.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)

#To understand the relationship between different attributes in the dataset, we will plot a correlation matrix

corrMatrix=car_data.corr() sns.heatmap(corrMatrix, annot=True, cmap=”viridis”)

plt.show()

Output:



#Splitting the dataset

X = car_data.drop(['Car_Name','Selling_Price'],axis=1)

Y = car_data['Selling_Price']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=42)

 # Loading the linear regression model

lin_reg_model = LinearRegression()

#Now we can fit the model to our dataset

lin_reg_model.fit(X_train,Y_train)

```
Out[12]: LinearRegression()
```

 # Prediction on Training data

training_data_prediction = lin_reg_model.predict(X_train)

 # R squared Error

train_error_score = metrics.r2_score(Y_train, training_data_prediction)

print("R squared Error - Training : ", train_error_score)

```
R squared Error - Training :  0.8839793496750799
```

 # prediction on Training data

Y_pred = lin_reg_model.predict(X_test)

 # R squared Error

test_error_score = metrics.r2_score(Y_test, Y_pred)

print("R squared Error - Test: ", test_error_score)

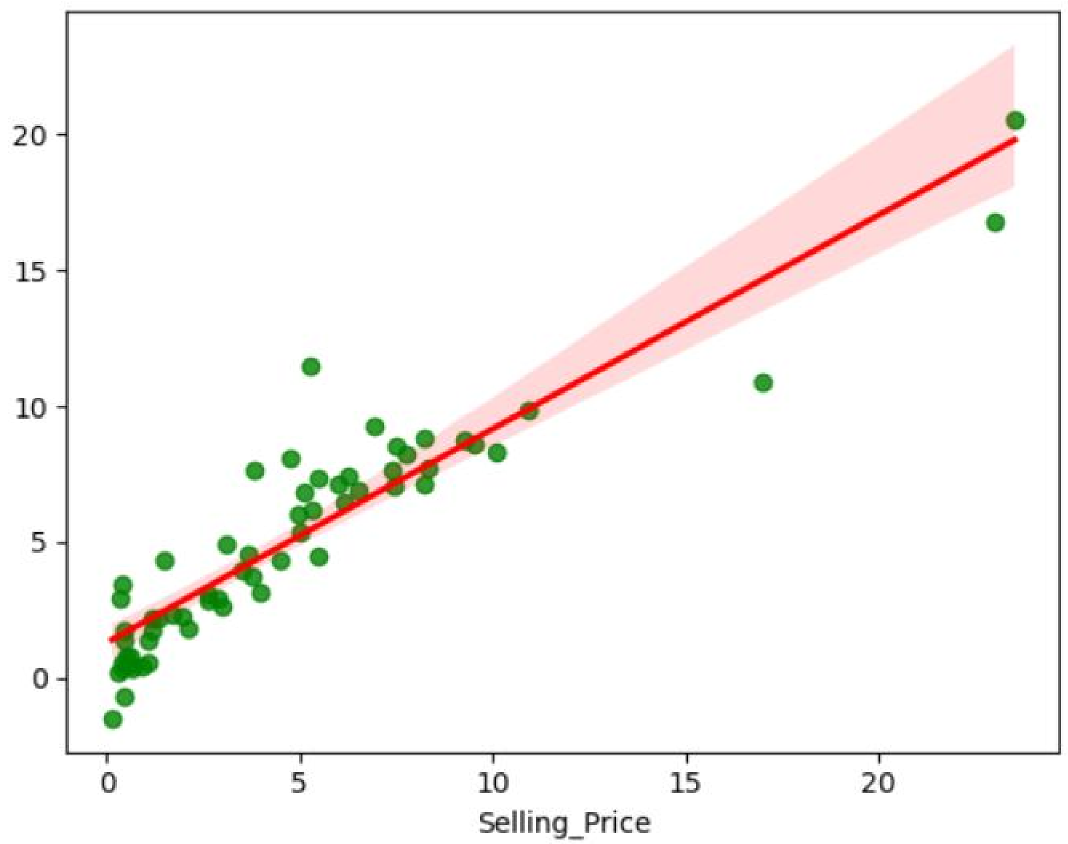
```
R squared Error - Test:  0.8468053957652042
```

 # create scatter plot with regression line

sns.regplot(Y_test, Y_pred, scatter_kws={"color": "green"}, line_kws={"color": "blue"})

<u>Output:</u>

Out[33]: <AxesSubplot:xlabel='Selling_Price'>



PRACTICAL-2

Write a program for Logistic Regression in Python.

PROGRAM:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#loading the dataset
dataset = pd.read_csv("User_Data.csv")
# input
x = dataset.iloc[:, [2, 3]].values
# output
y = dataset.iloc[:, 4].values

#splitting the dataset
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    x, y, test_size=0.25, random_state=0)
#scaling the features
from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
```

print (xtrain[0:10, :])

#Training the model

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(xtrain, ytrain)

```
Out[14]: LogisticRegression(random_state=0)
```

#prediction

y_pred = classifier.predict(xtest)

#evaluation using Confusion Matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(ytest, y_pred)

print ("Confusion Matrix : \n", cm)

```
Confusion Matrix :
 [[65  3]
 [ 8 24]]
```

 # Finding the Accuracy

from sklearn.metrics import accuracy_score

print ("Accuracy : ", accuracy_score(ytest, y_pred))

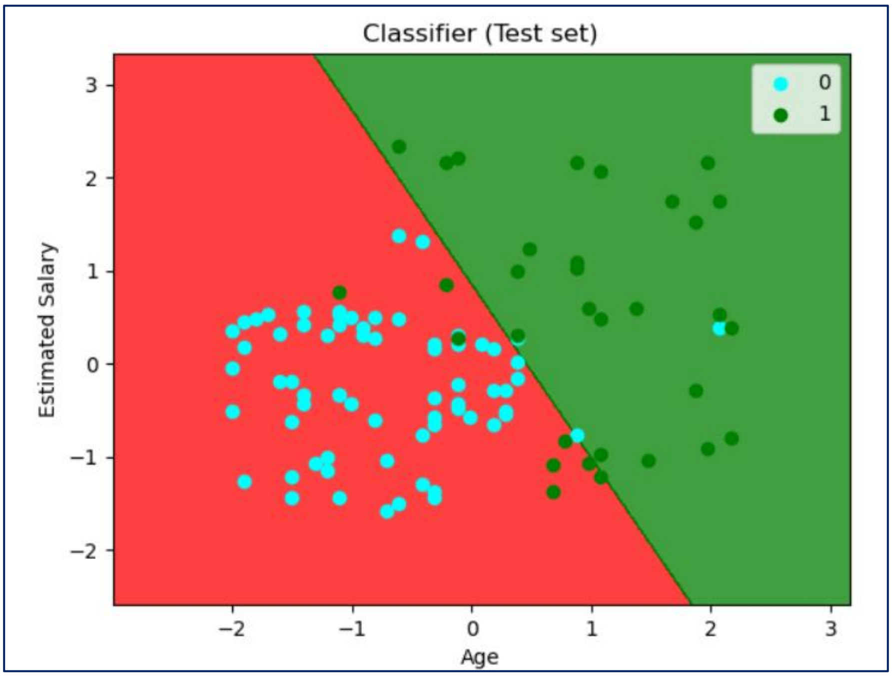
```
Accuracy :  0.89
```

#Visualization

from matplotlib.colors import ListedColormap

```
X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(
          np.array([X1.ravel(), X2.ravel()]).T).reshape(
          X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Output:



PRACTICAL-3

Write a program to implement decision tree for classification.

PROGRAM:

#importing the libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

from sklearn.tree import plot_tree

 # Load the dataset

data = pd.read_csv('winequality-red.csv')

 # Split the dataset into features and target variable

X = data.drop('quality', axis=1)

y = data['quality'].astype(str)

 # 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)

 # Create the decision tree classifier

clf = DecisionTreeClassifier()

 # Train the classifier on the training data

clf.fit(X_train, y_train)

```
Out[50]: DecisionTreeClassifier()
```

 # Make predictions on the testing data

y_pred = clf.predict(X_test)

 # Calculate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

<u>Output:</u>

```
Accuracy: 0.565625
```

#Visualizing Decision Trees

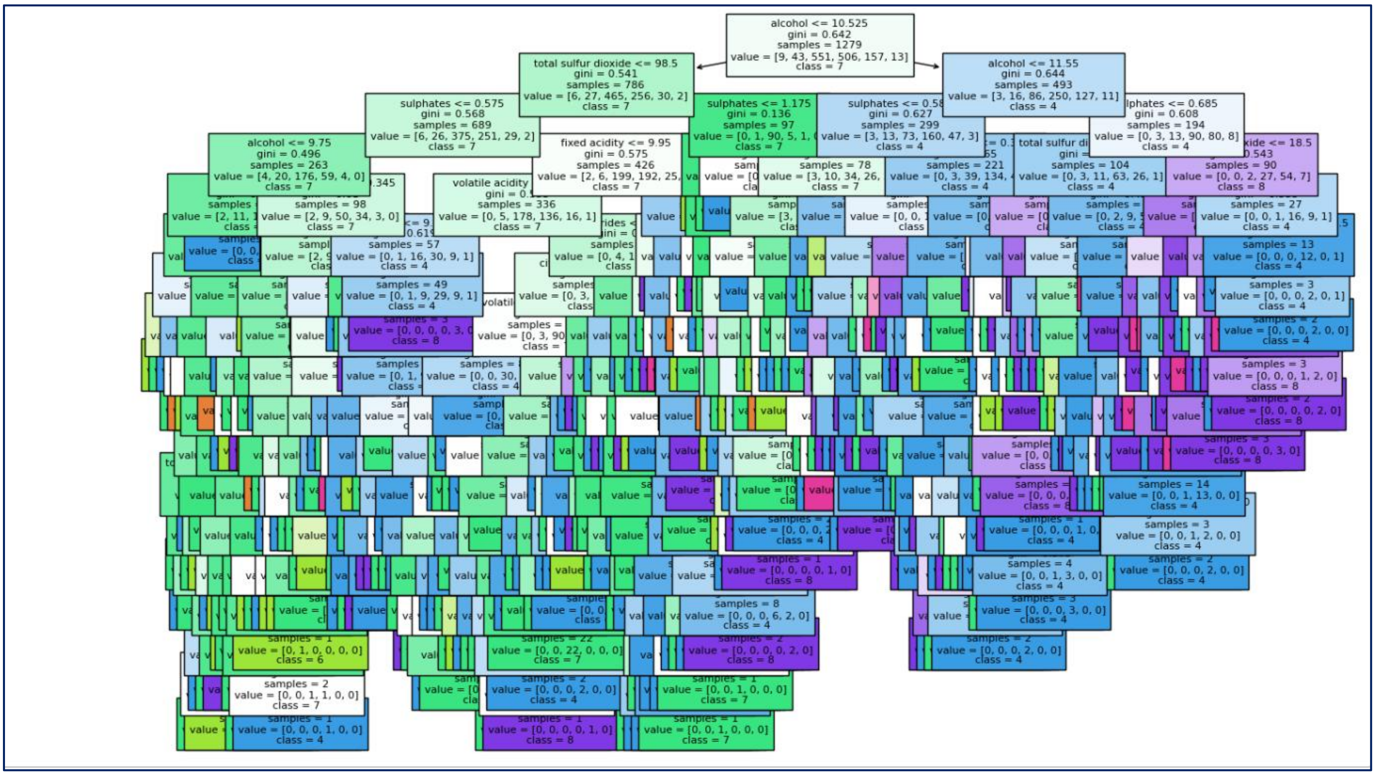
print("Accuracy:", accuracy)

plt.figure(figsize=(15, 10))

plot_tree(clf, filled=True, feature_names=X.columns, class_names=y.unique().tolist(), fontsize=8)

plt.show()

<u>Output:</u>



PRACTICAL-4

Write a program to implement Random Forests For Classification

PROGRAM:

```
#importing libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

# Load the dataset
data = pd.read_csv('winequality-red.csv')

# Split the dataset into features and target variable
X = data.drop('quality', axis=1)
y = data['quality'].astype(str)

# 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)

# Create the Random Forest classifier
rf_classifier = RandomForestClassifier()
```

 # Train the classifier on the training data

rf_classifier.fit(X_train, y_train)

```
Out[81]: RandomForestClassifier()
```

 # Make predictions on the testing data

y_pred = rf_classifier.predict(X_test)

 # Calculate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

```
Accuracy: 0.653125
```

 # Visualize the confusion matrix

confusion_mat = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4, 2))

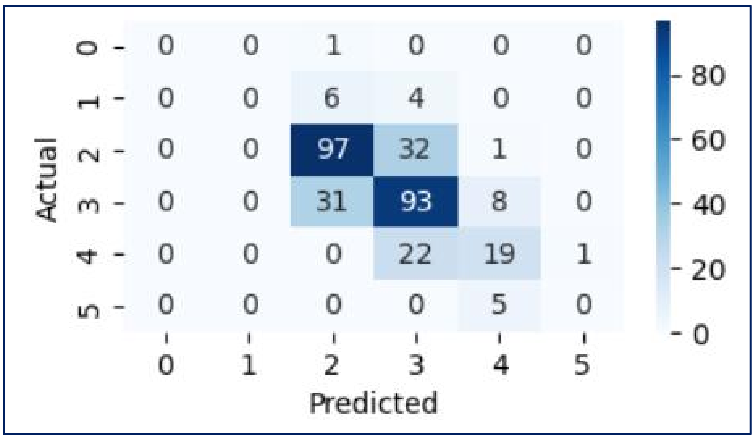
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

Output:



 # Display the classification report

class_report = classification_report(y_test, y_pred)

print(class_report)

<u>Output:</u>

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        10
           5       0.72      0.75      0.73       130
           6       0.62      0.70      0.66       132
           7       0.58      0.45      0.51        42
           8       0.00      0.00      0.00         5

    accuracy                           0.65       320
   macro avg       0.32      0.32      0.32       320
weighted avg       0.62      0.65      0.64       320
```

 # Visualize the predicted class distribution

plt.figure(figsize=(4, 2))

sns.countplot(y_pred, palette='Blues')

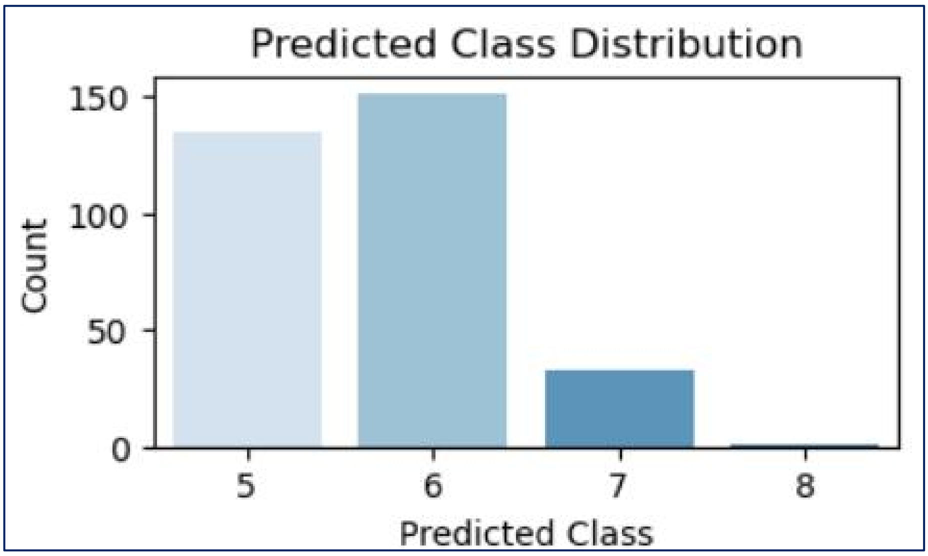
plt.xlabel('Predicted Class')

plt.ylabel('Count')

plt.title('Predicted Class Distribution')

plt.show()

<u>Output:</u>



PRACTICAL-5

Write a program to implement KNN Algorithm

PROGRAM:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

#Importing the dataset
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
```

 # Compute training and test data accuracy

train_accuracy[i] = knn.score(X_train, y_train)

test_accuracy[i] = knn.score(X_test, y_test)

 # Predict on dataset which model has not seen before

print(knn.predict(X_test))

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

 # Calculate the accuracy of the model

print(knn.score(X_test, y_test))

```
1.0
```

 # Generate plot

plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')

plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

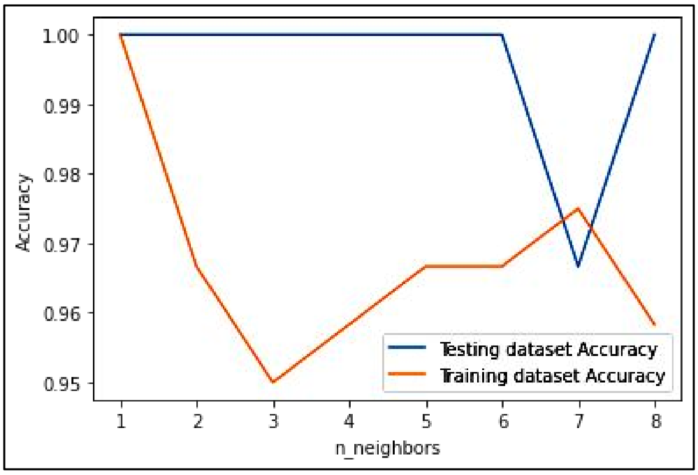
plt.legend()

plt.xlabel('n_neighbors')

plt.ylabel('Accuracy')

plt.show()

Output:



PRACTICAL-6

Write a program to implement Naive Bayes Algorithm.

PROGRAM:

#Loading Initial Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#Importing Dataset

dataset = pd.read_csv("cancer.csv")

#Exploring Dataset

dataset.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

#Clearing the dataset by dropping unnecessary columns

dataset = dataset.drop(["id"], axis = 1)

#Clearing the dataset by dropping unnecessary columns

dataset = dataset.drop(["Unnamed: 32"], axis = 1)

#Visualizing Dataset

M = dataset[dataset.diagnosis == "M"]

B = dataset[dataset.diagnosis == "B"]

plt.title("Malignant vs Benign Tumor")

plt.xlabel("Radius Mean")

plt.ylabel("Texture Mean")

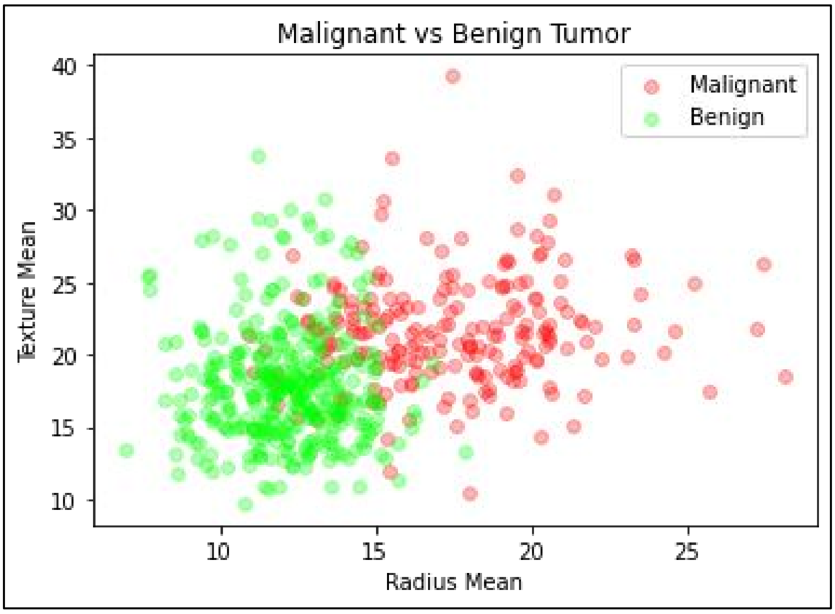
plt.scatter(M.radius_mean, M.texture_mean, color = "red", label = "Malignant", alpha = 0.3)

plt.scatter(B.radius_mean, B.texture_mean, color = "lime", label = "Benign", alpha = 0.3)

plt.legend()

plt.show()

Output:



#Preprocessing

dataset.diagnosis = [1 if i == "M" else 0 for i in dataset.diagnosis]

x = dataset.drop(["diagnosis"], axis = 1)

y = dataset.diagnosis.values

Normalization:

x = (x - np.min(x)) / (np.max(x) - np.min(x))

#Splitting dataset

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 = 42)

#Sklearn Gaussian Naive Bayes Model

from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(x_train, y_train)

```
Out[13]: GaussianNB()
```

#Prediction Score

print("Naive Bayes score: ",nb.score(x_test, y_test))

```
Naive Bayes score:  0.935672514619883
```

PRACTICAL-7

Write a program to implement Support Vector Machine (SVM).

PROGRAM:

#importing libraries

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report , confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

 # Load the Breast Cancer Wisconsin dataset

```
breast_cancer = datasets.load_breast_cancer()
```

 # Create a DataFrame from the dataset

```
df = pd.DataFrame(data=np.c_[breast_cancer['data'],
breast_cancer['target']],columns=list(breast_cancer['feature_names']) + ['target'])

df.head(5)
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374

5 rows × 31 columns

df.shape

```
Out[25]:  (569, 31)
```

df.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    float64
dtypes: float64(31)
memory usage: 137.9 KB
```

 # Split the dataset into features (X) and target (y)

X = df.drop('target', axis=1)

y = df['target']

 # 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)

 # Create an SVM classifier

svm = SVC(kernel='linear')

 # Train the SVM classifier

svm.fit(X_train, y_train)

```
Out[14]:  SVC(kernel='linear')
```

 # Make predictions on the test set

y_pred = svm.predict(X_test)

 # Print the classification report

print("Classification Report:")

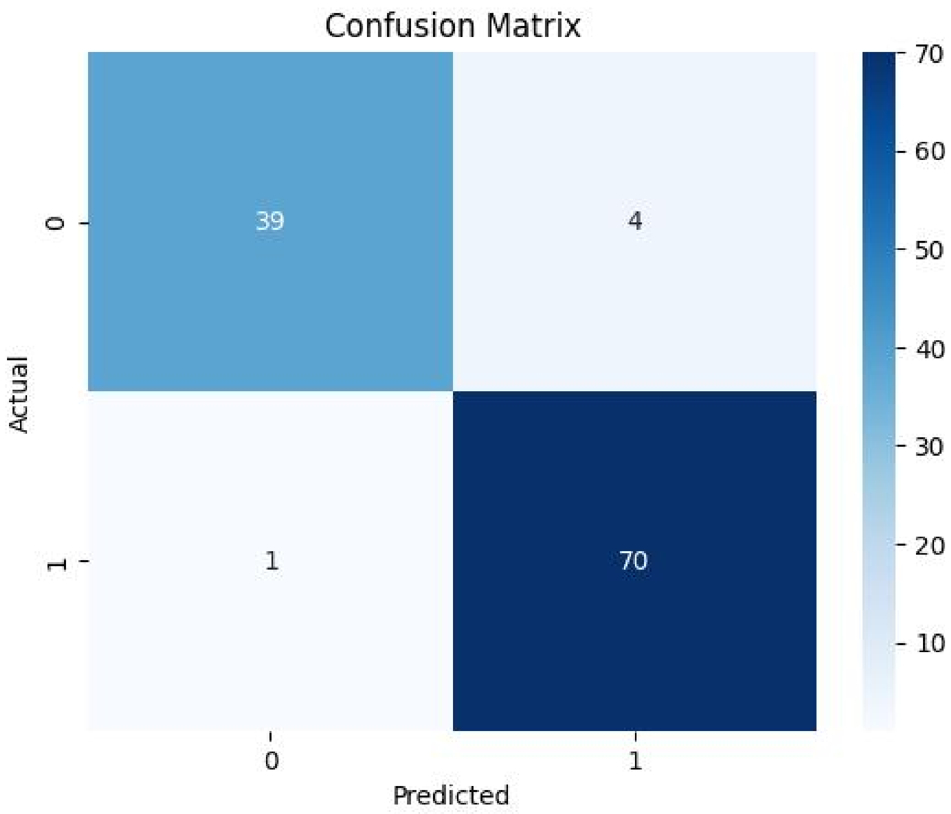
print(classification_report(y_test, y_pred))

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.91      0.94        43
         1.0       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

 # Plot the confusion matrix

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Output:



PRACTICAL-8

Write a program to Time Series Analysis Stock Price.

PROGRAM:

```
#Importing required libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('AAPL.csv')

# Convert the "Date" column to a DateTime index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Sort the data by date
df.sort_index(inplace=True)

# Plot the time series
plt.plot(df['Close'])
plt.xlabel('Date')
plt.ylabel('Stock Price ($)')
plt.title('Apple Inc. Stock Prices')
plt.show()
```

Output:

 # Calculate the daily returns

daily_returns = df['Close'].pct_change()

 # Plot the histogram of daily returns

plt.hist(daily_returns.dropna(), bins=50)

plt.xlabel('Daily Return')

plt.ylabel('Frequency')

plt.title('Distribution of Daily Returns')

plt.show()

 # Calculate the rolling mean and standard deviation

rolling_mean = df['Close'].rolling(window=30).mean()

rolling_std = df['Close'].rolling(window=30).std()

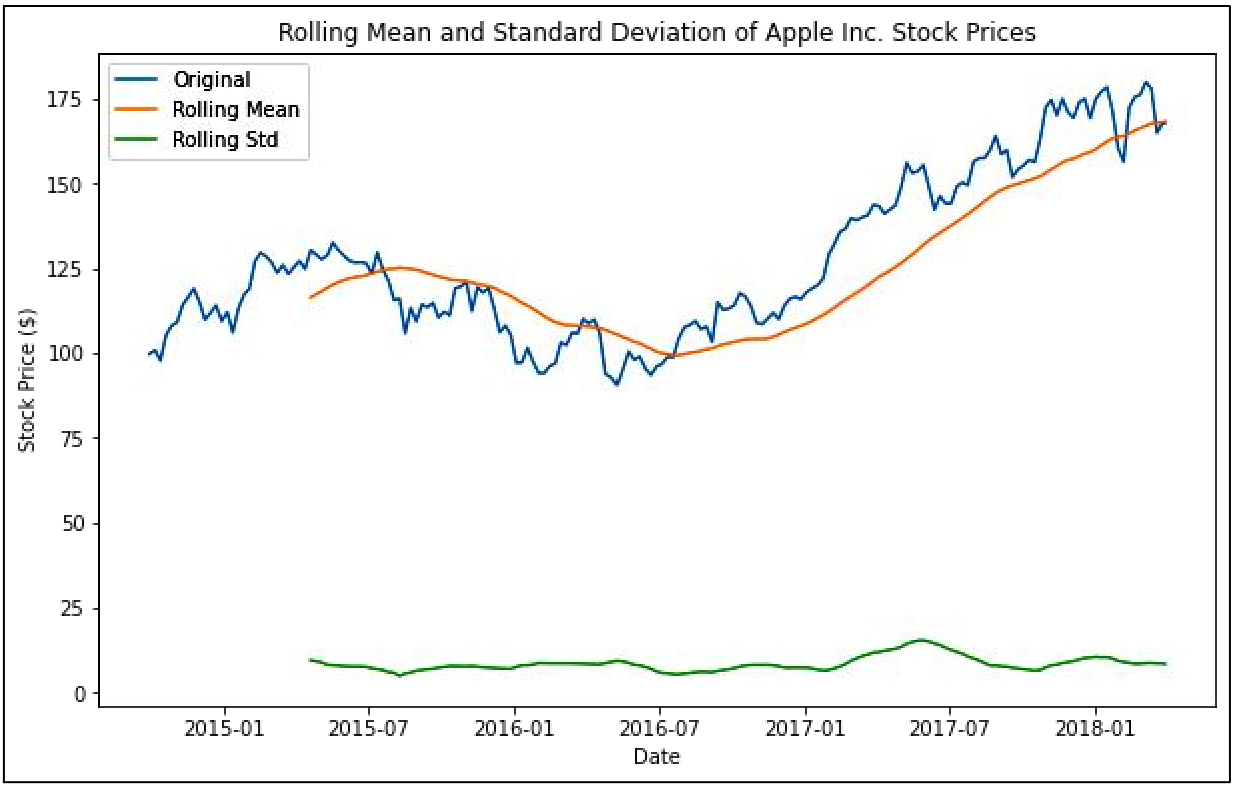
 # Plot the rolling statistics

plt.figure(figsize=(10, 6))

plt.plot(df['Close'], label='Original')

```
plt.plot(rolling_mean, label='Rolling Mean')
plt.plot(rolling_std, label='Rolling Std')
plt.xlabel('Date')
plt.ylabel('Stock Price ($)')
plt.title('Rolling Mean and Standard Deviation of Apple Inc. Stock Prices')
plt.legend()
plt.show()
```

Output:



PRACTICAL-9

Write a program to implement K-Means Clustering Algorithm.

PROGRAM:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

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

# Split the dataset 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)

# Determine the optimal number of clusters using the elbow method
inertias = [ ]
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train)
    inertias.append(kmeans.inertia_)
```

 # Plot the elbow curve

plt.plot(k_values, inertias, 'bx-')

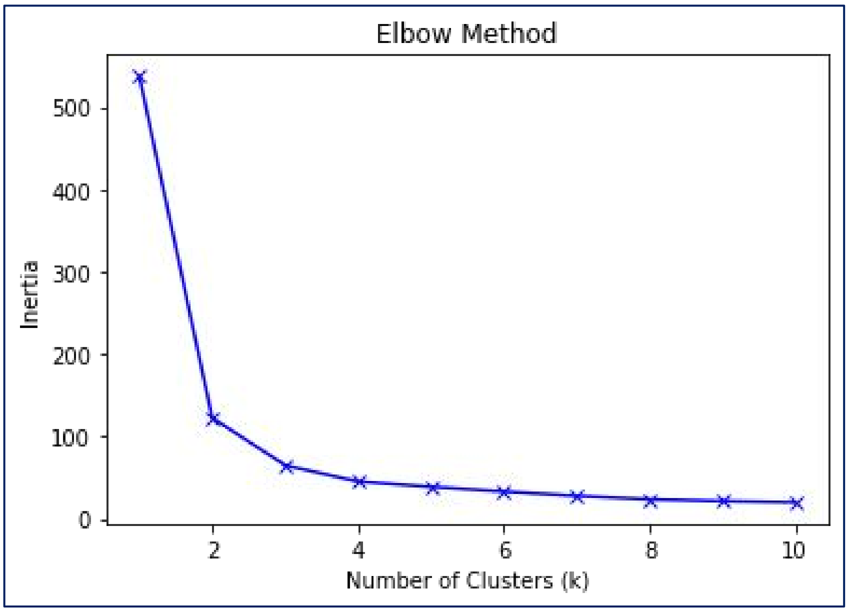
plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.title('Elbow Method')

plt.show()

<u>Output:</u>



 # Determine the optimal k value

optimal_k = 3 # Update with the elbow value determined from the plot

 # Initialize the K-means clustering model with the optimal number of clusters

kmeans = KMeans(n_clusters=optimal_k, random_state=42)

 # Train the model

kmeans.fit(X_train)

```
Out[10]:  KMeans(n_clusters=3, random_state=42)
```

 # Predict the clusters for the test set

y_pred = kmeans.predict(X_test)

 # Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

classification_rep = classification_report(y_test, y_pred)

 # Print the accuracy and classification report

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification_rep)

```
Accuracy: 0.36666666666666664
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.00      0.00      0.00         9
           2       0.10      0.09      0.10        11

    accuracy                           0.37        30
   macro avg       0.37      0.36      0.37        30
weighted avg       0.37      0.37      0.37        30
```

 # Predict the clusters for the dataset

labels = kmeans.labels_

 # Visualize the clusters

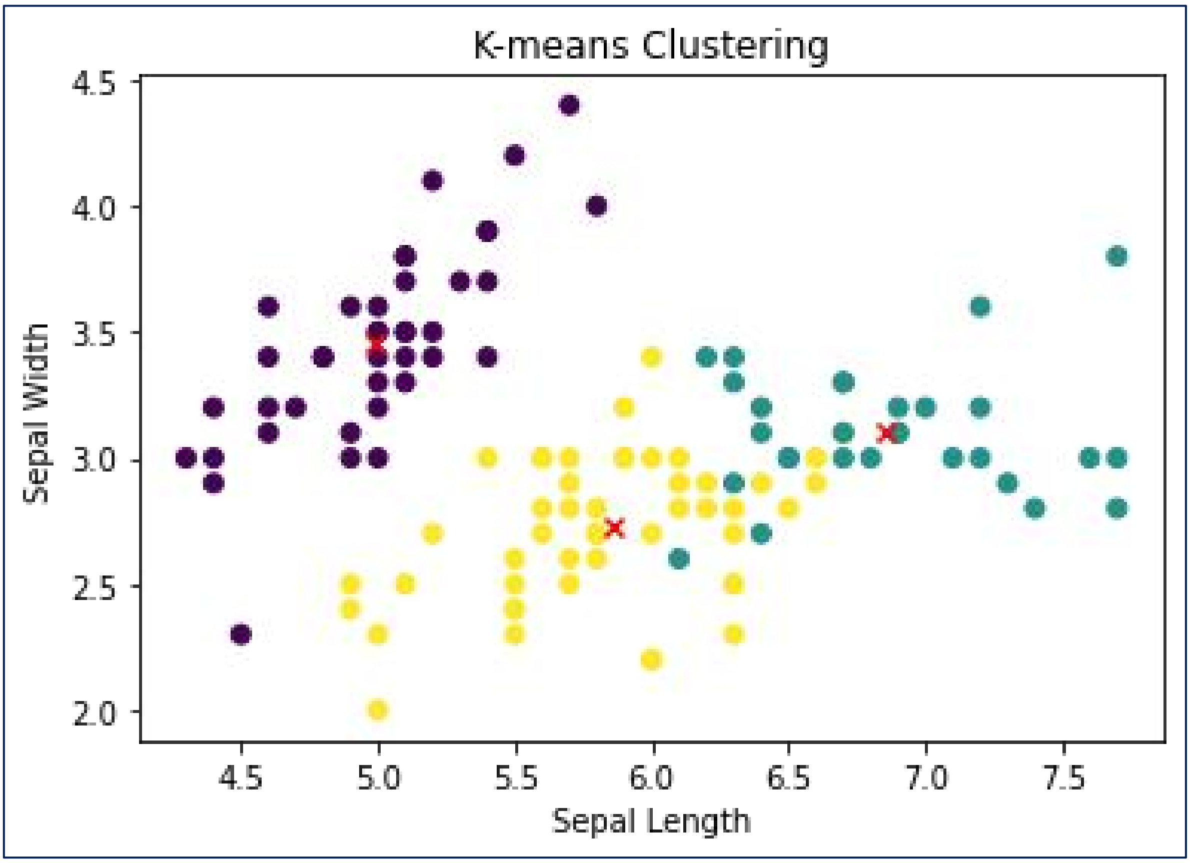
plt.scatter(X_train[:, 0], X_train[:, 1], c=labels, cmap='viridis')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='x', color='r')

plt.title("K-means Clustering")

```
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.show()
```

Output:



PRACTICAL-10

Write a program for Principal Component Analysis (PCA).

PROGRAM:

```
#importing libraries
import pandas as pd
import numpy as np
# Here we are using inbuilt dataset of scikit learn
from sklearn.datasets import load_breast_cancer
# instantiating
cancer = load_breast_cancer(as_frame=True)
# creating dataframe
df = cancer.frame
# checking shape
print('Original Dataframe shape :',df.shape)
# Input features
X = df[cancer['feature_names']]
print('Inputs Dataframe shape :', X.shape)
```

```
Original Dataframe shape : (569, 31)
Inputs Dataframe shape : (569, 30)
```

```
# Mean
X_mean = X.mean()

# Standard deviation
X_std = X.std()

# Standardization
```

Z = (X - X_mean) / X_std

 # covariance

c = Z.cov()

 # Plot the covariance matrix

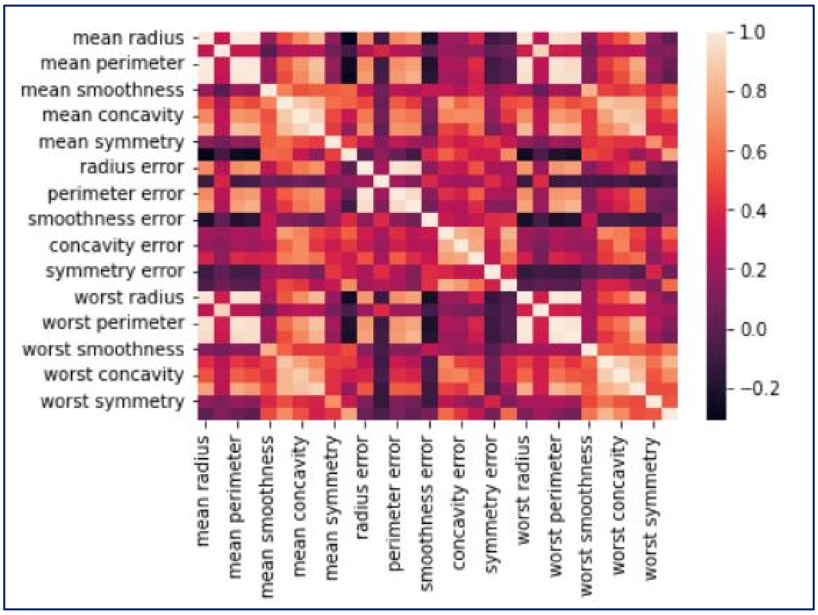
import matplotlib.pyplot as plt

import seaborn as sns

sns.heatmap(c)

plt.show()

Output:



#getting the Eigen values

eigenvalues, eigenvectors = np.linalg.eig(c)

print('Eigen values:\n', eigenvalues)

print('Eigen values Shape:', eigenvalues.shape)

print('Eigen Vector Shape:', eigenvectors.shape)

Output:

```
Eigen values:
 [1.32816077e+01 5.69135461e+00 2.81794898e+00 1.98064047e+00
 1.64873055e+00 1.20735661e+00 6.75220114e-01 4.76617140e-01
 4.16894812e-01 3.50693457e-01 2.93915696e-01 2.61161370e-01
 2.41357496e-01 1.57009724e-01 9.41349650e-02 7.98628010e-02
 5.93990378e-02 5.26187835e-02 4.94775918e-02 1.33044823e-04
 7.48803097e-04 1.58933787e-03 6.90046388e-03 8.17763986e-03
 1.54812714e-02 1.80550070e-02 2.43408378e-02 2.74394025e-02
 3.11594025e-02 2.99728939e-02]
Eigen values Shape: (30,)
Eigen Vector Shape: (30, 30)
```

 # Index the eigenvalues in descending order

idx = eigenvalues.argsort()[::-1]

 # Sort the eigenvalues in descending order

eigenvalues = eigenvalues[idx]

 # sort the corresponding eigenvectors accordingly

eigenvectors = eigenvectors[:,idx]

explained_var = np.cumsum(eigenvalues) / np.sum(eigenvalues)

explained_var

```
Out[6]: array([0.44272026, 0.63243208, 0.72636371, 0.79238506, 0.84734274,
               0.88758796, 0.9100953 , 0.92598254, 0.93987903, 0.95156881,
               0.961366  , 0.97007138, 0.97811663, 0.98335029, 0.98648812,
               0.98915022, 0.99113018, 0.99288414, 0.9945334 , 0.99557204,
               0.99657114, 0.99748579, 0.99829715, 0.99889898, 0.99941502,
               0.99968761, 0.99991763, 0.99997061, 0.99999557, 1.        ])
```

n_components = np.argmax(explained_var >= 0.50) + 1

n_components

```
Out[7]: 2
```

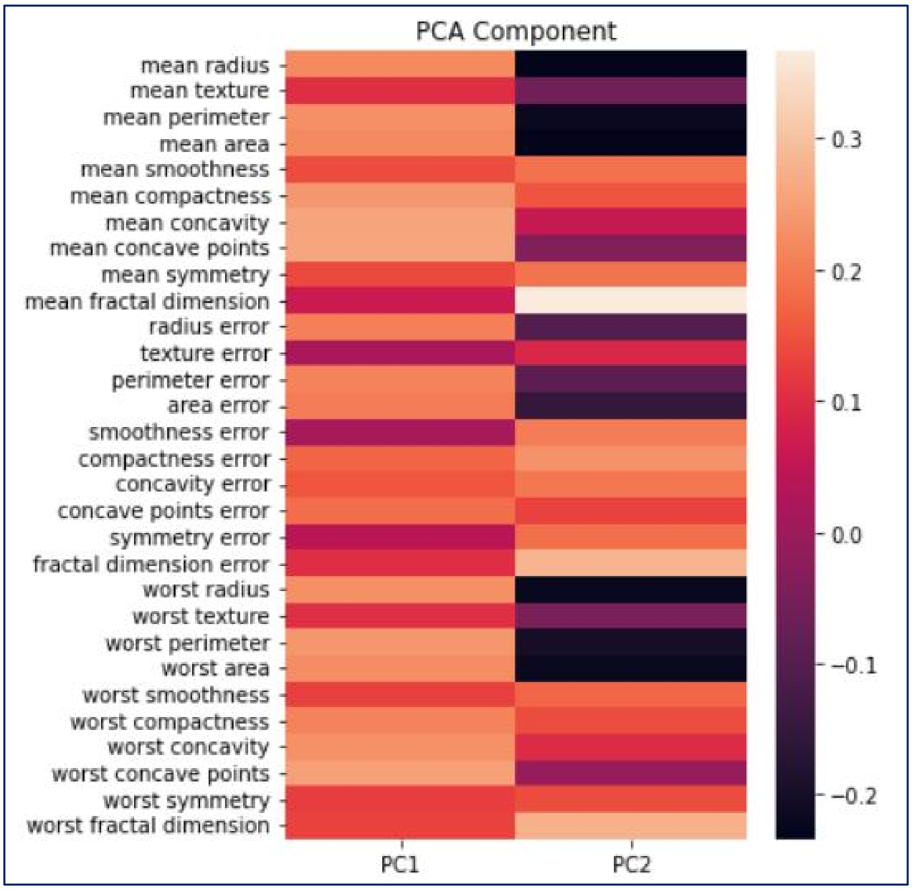
 # PCA component or unit matrix

```
u = eigenvectors[:,:n_components]
pca_component = pd.DataFrame(u,
                             index = cancer['feature_names'],
                             columns = ['PC1','PC2']
                            )
```

 # plotting heatmap

```
plt.figure(figsize =(5, 7))
sns.heatmap(pca_component)
plt.title('PCA Component')
plt.show()
```

Output:



 # Matrix multiplication or dot Product

```
Z_pca = Z @ pca_component


Z_pca = pd.DataFrame(Z_pca.values,
                     columns = ['PCA1','PCA2']
                     )
Z_pca.head()
```

Output:

Out[9]:

	PCA1	PCA2
0	9.184755	1.946870
1	2.385703	-3.764859
2	5.728855	-1.074229
3	7.116691	10.266556
4	3.931842	-1.946359