EX.N0 : 1	PREDICTING HOUSE PRICES
DATE :	

AIM: To build a regression model to predict house prices based on features like location, size, and amenities.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

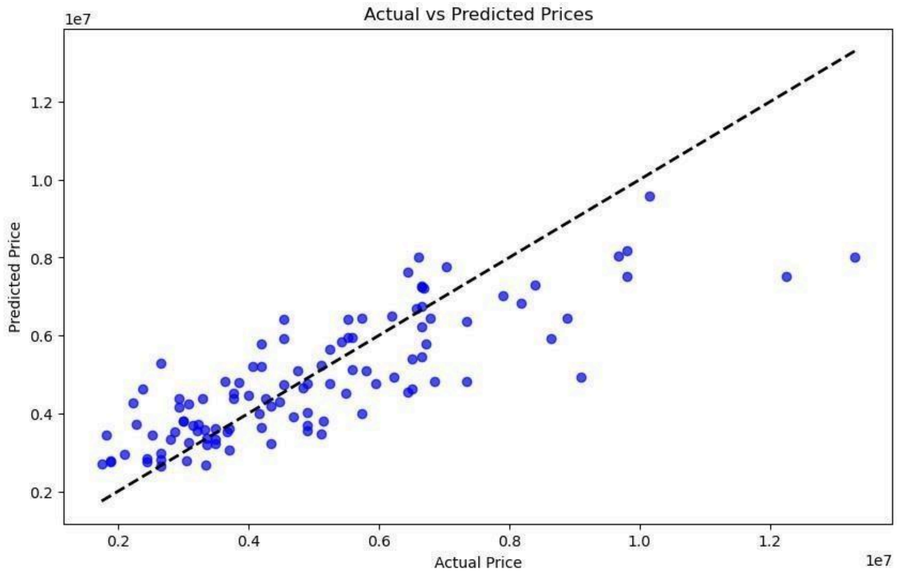
Step 6: Print equal metric & test the cell.

PROGRAM:

```
import pandas as pd from sklearn.preprocessing

import LabelEncoder from sklearn.model_selection

import train_test_split from sklearn.linear_model

import LinearRegression from sklearn.metrics

import r2_score, mean_absolute_error

import matplotlib.pyplot as plt

file_path = 'C:/Users/APPU/Downloads/Housing.csv'

housing_data = pd.read_csv(file_path)
```

```
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
'prefarea', 'furnishingstatus']  le = LabelEncoder() for feature in categorical_features:
housing_data[feature] = le.fit_transform(housing_data[feature])

X = housing_data.drop('price', axis=1)

y = housing_data['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
 model = LinearRegression()
model.fit(X_train, y_train)
 y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='b')
plt.plot([y_test.min(), y_test.max()],  [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual Price') plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices')  plt.show() print(f'R-squared (R²): {r2}')
 print(f'Mean Absolute Error (MAE): {mae}')
```

OUTPUT:




```
import numpy as np
test=np.array([ 7420,4,2,3,1,0,0,0,1,2,1,0]).reshape(-12,12)
model.predict(test)
```

```
array([8004072.41154001])
```

RESULT:

Thus, the program for house price prediction is executed successfully.

EX.N0 : 2	CUSTOMER SEGMENTATION FOR AN E-COMMERCE COMPANY
DATE :	

AIM:

To perform cluster analysis to segment customers based on purchasing behaviour.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

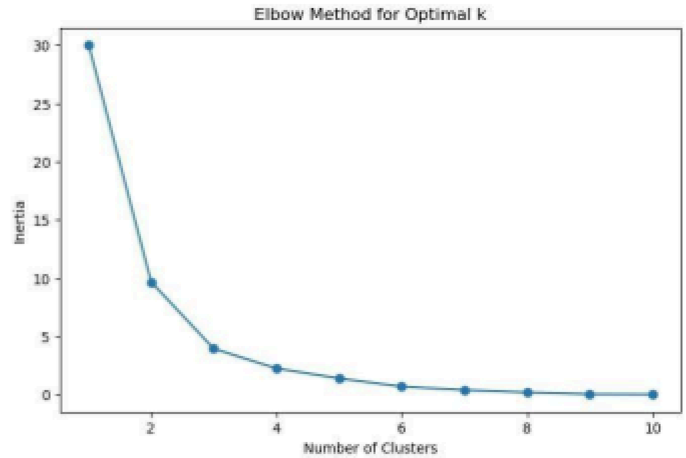
Step 6: Print equal metric & test the cell.

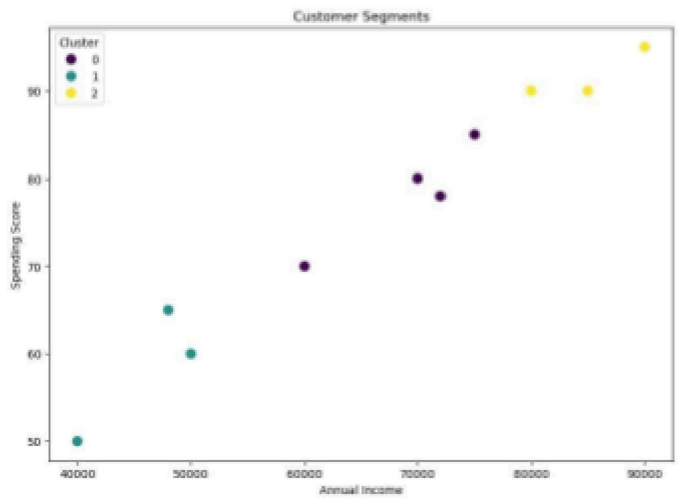
PROGRAM:

```
import pandas as pd import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

import os os.environ['OMP_NUM_THREADS'] = '1'

data = {'CustomerID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Age': [25, 45, 35, 50, 23, 33, 43, 36, 29, 55],
```

```
'AnnualIncome': [50000, 60000, 70000, 80000, 40000, 75000, 85000, 72000, 48000, 90000],

'SpendingScore': [60, 70, 80, 90, 50, 85, 90, 78, 65, 95] } df =

pd.DataFrame(data)

features = df[['Age', 'AnnualIncome', 'SpendingScore']]

scaler = StandardScaler()

scaled_features = scaler.fit_transform(features)

inertia = []

 k_range = range(1, 11) for k in k_range:

kmeans = KMeans(n_clusters=k, n_init=10, random_state=0)

kmeans.fit(scaled_features)  inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))  plt.plot(k_range, inertia, marker='o')

plt.xlabel('Number of Clusters')  plt.ylabel('Inertia')

 plt.title('Elbow Method for Optimal k')  plt.show()

optimal_k = 3

kmeans    =    KMeans(n_clusters=optimal_k,    n_init=10,    random_state=0)

df['Cluster'] = kmeans.fit_predict(scaled_features)

plt.figure(figsize=(10, 7))

sns.scatterplot(data=df, x='AnnualIncome', y='SpendingScore', hue='Cluster', palette='viridis',s=100)

plt.title('Customer Segments')

plt.xlabel('Annual Income')

plt.ylabel('Spending Score')

plt.legend(title='Cluster')

 plt.show()

print(df)
```

OUTPUT:





	CustomerID	Age	AnnualIncome	SpendingScore	Cluster
0	1	25	50000	60	1
1	2	45	60000	70	0
2	3	35	70000	80	0
3	4	50	80000	90	2
4	5	23	40000	50	1
5	6	33	75000	85	0
6	7	43	85000	90	2
7	8	36	72000	78	0
8	9	29	48000	65	1
9	10	55	90000	95	2

RESULT:

Thus, the program for Customer Segmentation for an E-commerce Company is executed successfully.

EX.N0 : 3	SENTIMENT ANALYSIS OF MOVIE REVIEWS
DATE :	

AIM:

To classify movie reviews as positive or negative using text Data.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

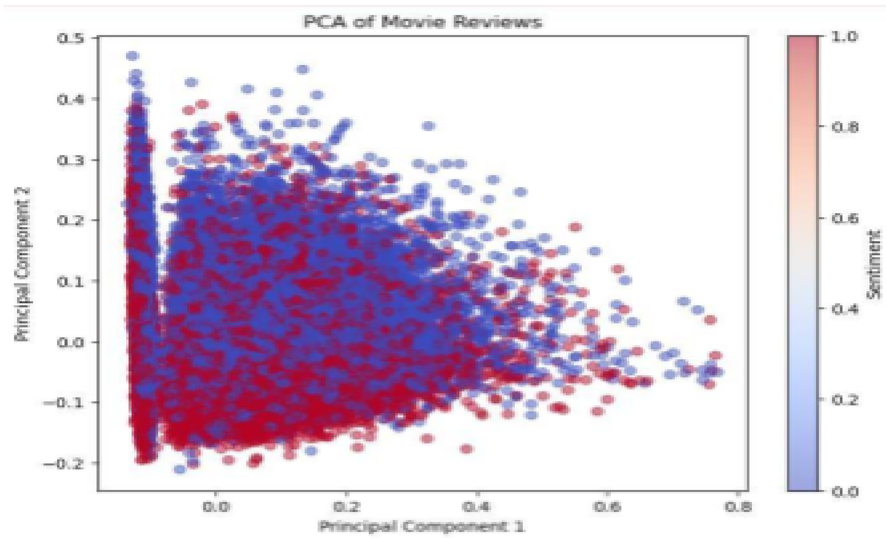
PROGRAM:

```
import pandas as pd

sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

import nltk from nltk.corpus

import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer

import seaborn as sns nltk.download('punkt') nltk.download('stopwords')
```

```
df = pd.read_csv('C:/Users/AI_LAB/Downloads/IMDB Dataset.csv')

stop_words = set(stopwords.words('english')) stemmer = PorterStemmer()

def preprocess_text(text):

tokens = word_tokenize(text.lower()) tokens = [stemmer.stem(word) for word in tokens if

word.isalpha() and word not in stop_words] return ' '.join(tokens) df['cleaned_review'] =

df['review'].apply(preprocess_text) vectorizer = TfidfVectorizer(max_features=5000)

X = vectorizer.fit_transform(df['cleaned_review']).toarray()

encoder = LabelEncoder() y =

encoder.fit_transform(df['sentiment']) pca =

PCA(n_components=2) X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', alpha=0.5)

plt.title('PCA of Movie Reviews') plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2') plt.colorbar(label='Sentiment') plt.show()

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

 model = LogisticRegression(max_iter=1000) model.fit(X_train, y_train)

y_pred = model.predict(X_test) print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred)) print("\nClassification Report:")

print(classification_report(y_test, y_pred))

positive_reviews = ' '.join(df[df['sentiment'] == 1]['cleaned_review'])

negative_reviews = ' '.join(df[df['sentiment'] == 0]['cleaned_review'])

plt.figure(figsize=(12, 6)) if len(positive_reviews.strip()) > 0:

plt.subplot(1, 2, 1)
```

```
plt.imshow(WordCloud(width=800, height=400,background_color='white').generate(positive_reviews),
interpolation='bilinear')
plt.title('Positive Reviews') plt.axis('off')
else:
print("No content available for positive reviews.")
if len(negative_reviews.strip()) > 0:
plt.subplot(1, 2, 2)
plt.imshow(WordCloud(width=800, height=400,
background_color='white').generate(negative_reviews), interpolation='bilinear')
plt.title('Negative Reviews') plt.axis('off') else:
print("No content available for negative reviews.") plt.show()
sns.countplot(x='sentiment', data=df)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count') plt.show()
```

OUTPUT:




```
Confusion Matrix:
[[4306  655]
 [ 511 4528]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88      4961
           1       0.87      0.90      0.89      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

RESULT:

Thus, the program for sentiment analysis of movie reviews is executed successfully.

EX.NO : 4	HANDWRITTEN DIGIT RECOGNITION
DATE :	

AIM:

To implement handwritten digit recognition using python.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

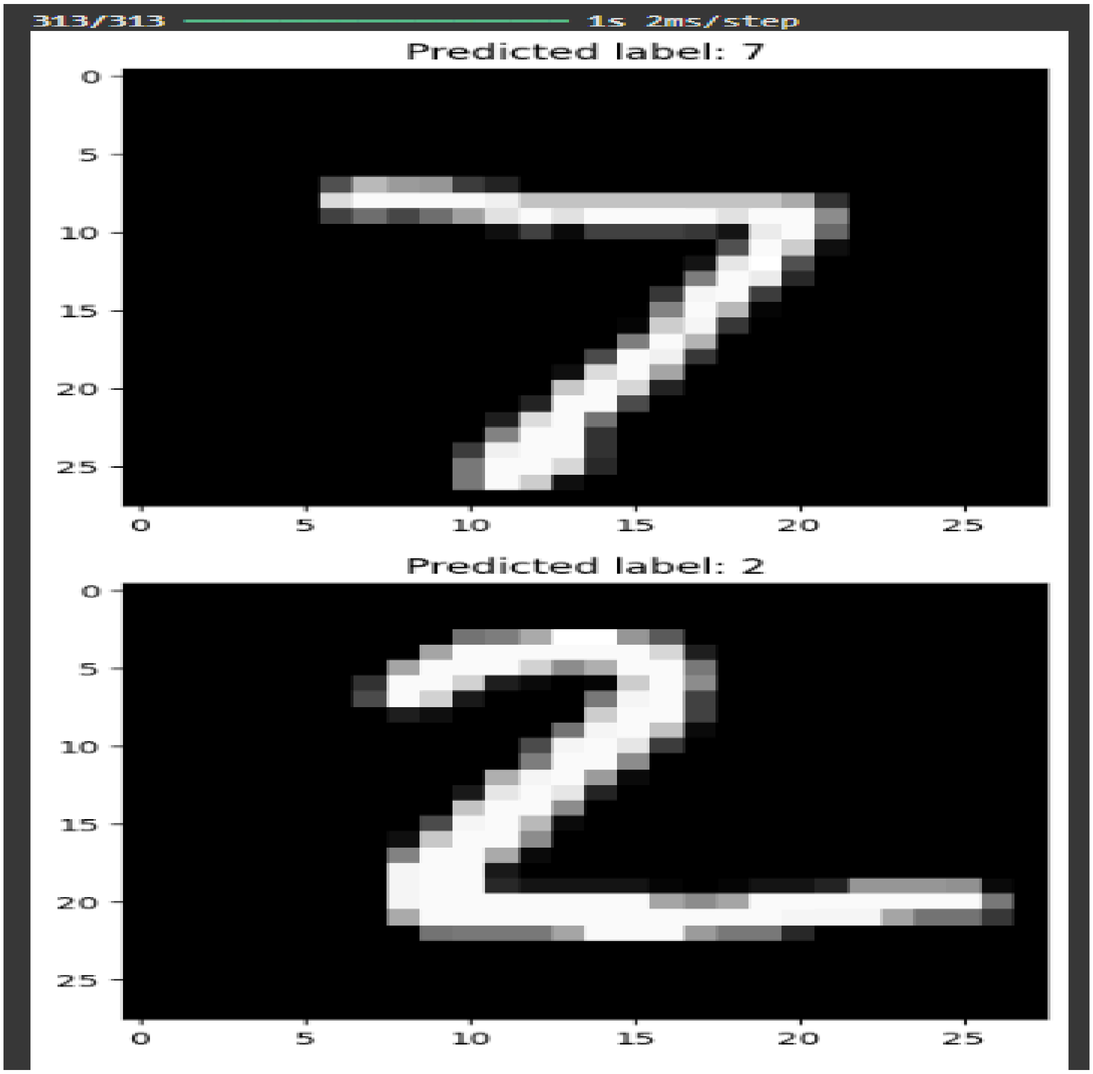
SOURCE CODE:

```
import numpy as np

from keras.datasets import mnist

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Input, Flatten, Dense

import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = mnist.load_data()

plt.imshow(x_train[0], cmap='gray')

plt.show()
```

```
x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, num_classes=10)

y_test = to_categorical(y_test, num_classes=10)

# Define the model

model = Sequential([

    Input(shape=(28, 28)),

    Flatten(),

    Dense(128, activation='relu'),

    Dense(128, activation='relu'),

    Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f'Test accuracy: {test_acc}')

predictions = model.predict(x_test)

for i in range(5):

    plt.imshow(x_test[i], cmap='gray')
```

```
plt.title(f'Predicted label: {np.argmax(predictions[i])}')

plt.show()
```

OUTPUT:



RESULT:

Thus the the program for implementing handwritten digit recognition is successfully executed and output is obtained.

EX.N0 : 5	STOCK MARKET ANALYSIS
DATE :	

AIM:

To analyse stock market data to predict future stock prices.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

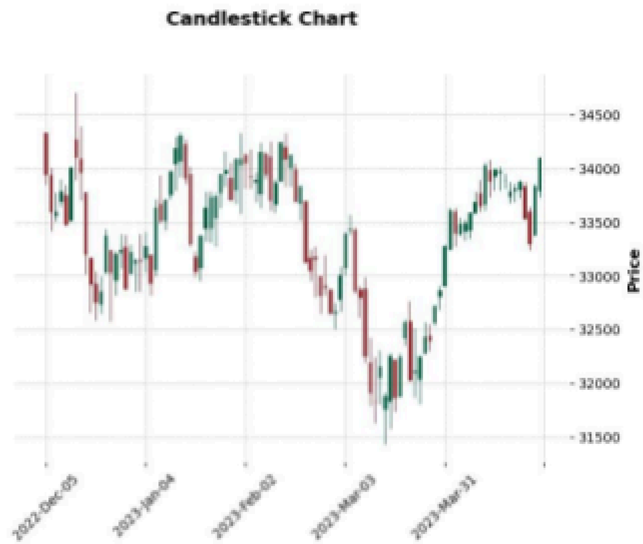
PROGRAM:

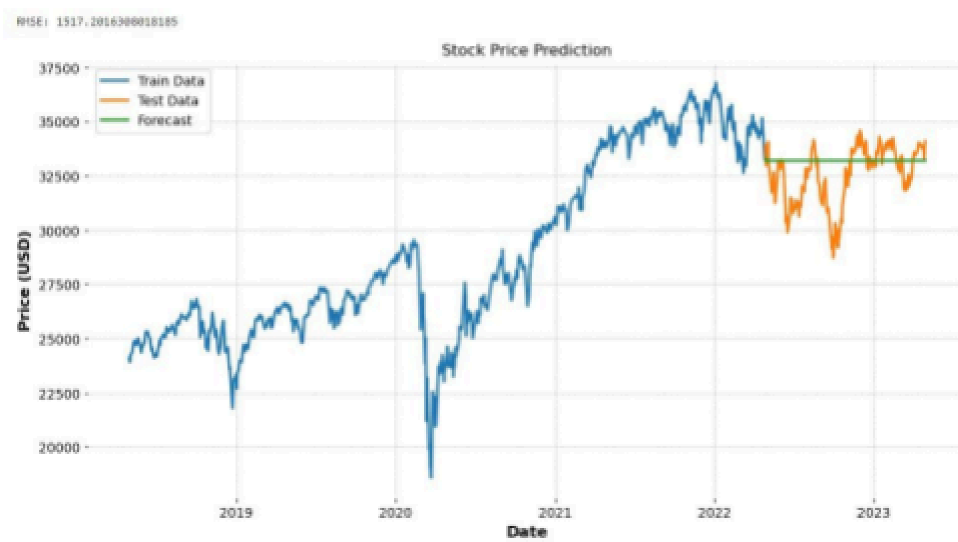
```
import pandas as pd

import matplotlib.pyplot as plt

import mplfinance as mpf

from statsmodels.tsa.arima.model

import ARIMA

from sklearn.metrics import mean_squared_error

import numpy as np

file_path = r'C:\Users\APPU\Downloads\yahoo_data.xlsx'

data = pd.read_excel(file_path, index_col='Date', parse_dates=True)
```

```
data.rename(columns={'Close*': 'Close', 'Adj Close**': 'Adj Close'}, inplace=True)

data.sort_index(inplace=True) data.ffill(inplace=True)

if 'Adj Close' in data.columns:

plt.figure(figsize=(12, 6))

plt.plot(data['Adj Close'], label='Adjusted Close Price')

plt.title('Adjusted Close Price Over Time')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.show() reduced_data = data[-100:]

mpf.plot(reduced_data, type='candle', style='charles', title='Candlestick Chart')

train_data, test_data = data['Adj Close'][:int(len(data)*0.8)], data['Adj Close'][int(len(data)*0.8):]

model = ARIMA(train_data, order=(5, 1, 0))

model_fit = model.fit()

forecast = model_fit.forecast(steps=len(test_data))

mse = mean_squared_error(test_data, forecast)

rmse = np.sqrt(mse) print(f'RMSE: {rmse}')

plt.figure(figsize=(12, 6))

plt.plot(train_data.index, train_data, label='Train Data')

plt.plot(test_data.index, test_data, label='Test Data')

plt.plot(test_data.index, forecast, label='Forecast')

plt.title('Stock Price Prediction')
```

```
plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.show()
```

OUTPUT:





RESULT:

Thus, the program for stock market analysis is executed successfully.

EX.N0 : 6	LOAN DEFAULT PREDICTION
DATE:	

AIM:

Predict loan default probability based on borrower information.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

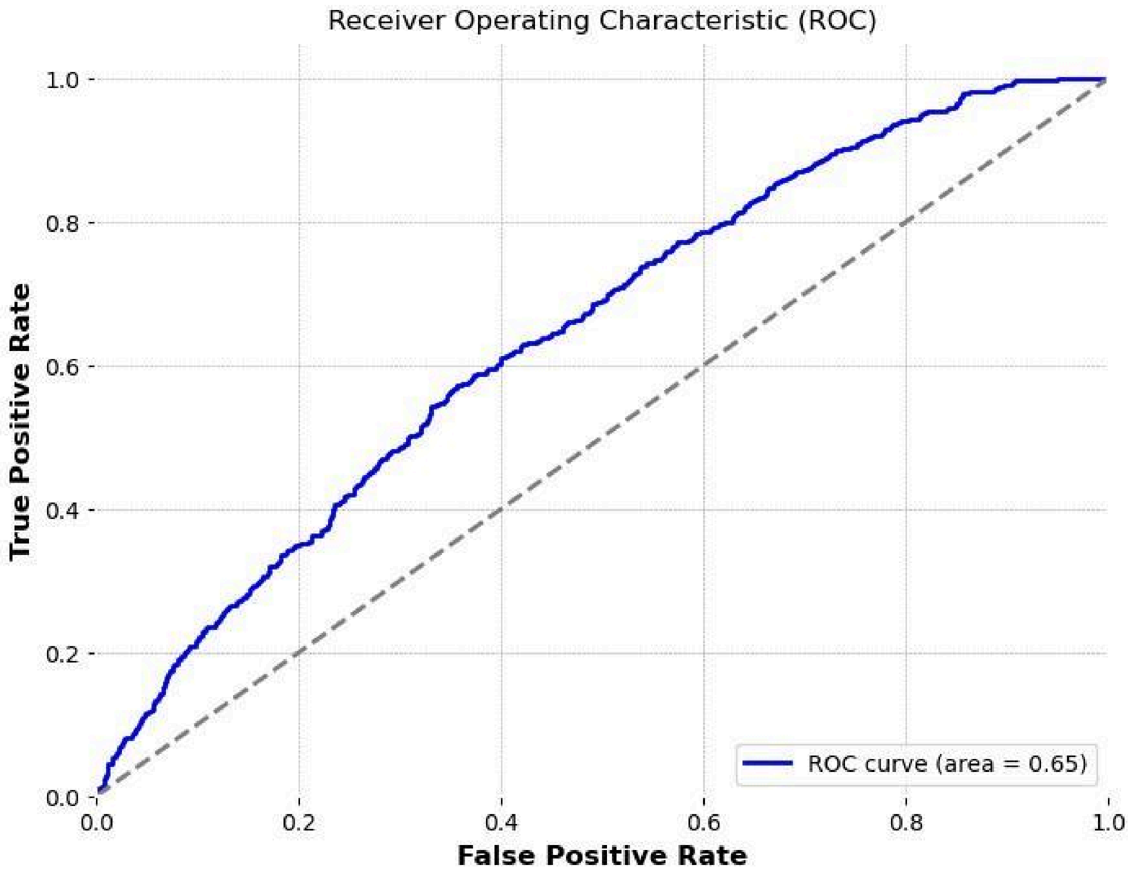
Step 6: Print equal metric & test the cell.

PROGRAM:

```
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 LogisticRegression from sklearn.metrics

import roc_curve, auc from sklearn.preprocessing

import StandardScaler from sklearn.decomposition

import PCA import os

file_path = 'C:/Users/APPU/Downloads/loan_data.csv'  # Update path accordingly if

os.path.exists(file_path): df = pd.read_csv(file_path)
```

```
print("Data loaded successfully.")

else:

print(f"File not found: {file_path}")

dummies = pd.get_dummies(df['purpose'], drop_first=True)

df = pd.concat([df, dummies], axis=1)

df.drop('purpose', inplace=True, axis=1)

X = df.drop(['not.fully.paid'], axis=1)

y = df['not.fully.paid']

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X) pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.33, random_state=42)

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred_prob = model.predict_proba(X_test)[:, 1] fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right') plt.show()
```

OUTPUT:



RESULT:

Thus, the program for loan default prediction is executed successfully.

EX.NO : 7	IMAGE RECOGNITION
DATE :	

AIM:

To implement image recognition to classify images into categories using various features.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

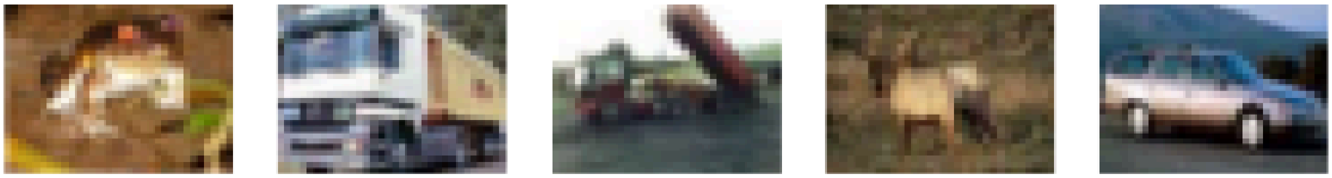
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from keras.datasets import cifar10

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report
```

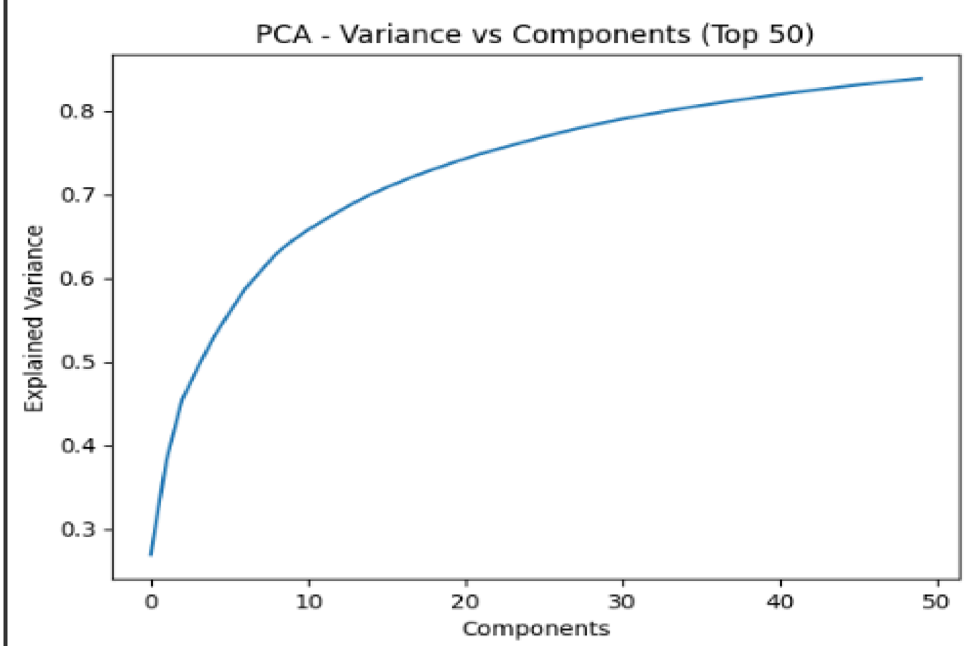
```
class SimpleImageClassifier:
    def __init__(self):
        (self.X_train, self.y_train), (self.X_test, self.y_test) = cifar10.load_data()
        self.X_train = self.X_train.reshape(len(self.X_train), -1) / 255.0
        self.X_test = self.X_test.reshape(len(self.X_test), -1) / 255.0
    def visualize_images(self, n=5):
        plt.figure(figsize=(10, 2))
        for i in range(n):
            plt.subplot(1, n, i + 1)
            plt.imshow(self.X_train[i].reshape(32, 32, 3))
            plt.axis('off')
        plt.show()
    def pca_analysis(self, components=50):
        scaler = StandardScaler()
        self.X_train_scaled = scaler.fit_transform(self.X_train)
        self.X_test_scaled = scaler.transform(self.X_test)
        pca = PCA(n_components=components)
        self.X_train_pca = pca.fit_transform(self.X_train_scaled)
        self.X_test_pca = pca.transform(self.X_test_scaled)
        cumulative_explained_variance = np.cumsum(pca.explained_variance_ratio_)
        plt.plot(cumulative_explained_variance)
```

```
        plt.xlabel('Number of Components')

        plt.ylabel('Cumulative Explained Variance')

        plt.title(f'PCA - Variance vs Components (Top {components})')

        plt.show()

    def train_knn(self, n_neighbors=5):

        knn = KNeighborsClassifier(n_neighbors=n_neighbors)

        knn.fit(self.X_train_pca, self.y_train.ravel())

        y_pred = knn.predict(self.X_test_pca)

        accuracy = accuracy_score(self.y_test, y_pred)

        print(f"KNN Accuracy with {n_neighbors} neighbors: {accuracy:.2f}")

        print(classification_report(self.y_test, y_pred))

    def clustering(self, n_clusters=10):

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

        cluster_labels = kmeans.fit_predict(self.X_train_pca)

        plt.scatter(self.X_train_pca[:, 0], self.X_train_pca[:, 1], c=cluster_labels, cmap='rainbow', s=2)

        plt.title(f'KMeans Clustering on CIFAR-10 Images (PCA Reduced, {n_clusters} Clusters)')

        plt.show()

if __name__ == "__main__":

    classifier = SimpleImageClassifier()

    classifier.visualize_images()
```

```
classifier.pca_analysis(components=50)

classifier.train_knn(n_neighbors=5)

classifier.clustering(n_clusters=10)
```

OUTPUT:






```
KNN Accuracy: 0.39
              precision    recall  f1-score   support

           0       0.39      0.59      0.47      1000
           1       0.57      0.39      0.46      1000
           2       0.27      0.44      0.33      1000
           3       0.32      0.22      0.26      1000
           4       0.28      0.42      0.34      1000
           5       0.47      0.25      0.33      1000
           6       0.35      0.43      0.39      1000
           7       0.59      0.32      0.41      1000
           8       0.49      0.59      0.54      1000
           9       0.66      0.28      0.39      1000

    accuracy                           0.39     10000
   macro avg       0.44      0.39      0.39     10000
weighted avg       0.44      0.39      0.39     10000
```

RESULT:

Thus the program for implementing image recognition to classify images into categories using various features is successfully executed and output is obtained.

EX.NO : 8	PREDICTING DIABETES
DATE :	

AIM:

To predict the onset of diabetes based on medical measurements.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
from sklearn.metrics import confusion_matrix, classification_report

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",

        "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]

data = pd.read_csv(url, header=None, names=columns)

print(data.head())

print(data.describe())

sns.pairplot(data, hue='Outcome')

plt.show()

plt.figure(figsize=(10, 8))

sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap="coolwarm")

plt.title("Correlation Heatmap")

plt.show()

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

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

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

log_reg = LogisticRegression()
```

```
log_reg.fit(X_train_scaled, y_train)

y_pred_log = log_reg.predict(X_test_scaled)

print("Logistic Regression Classification Report:")

print(classification_report(y_test, y_pred_log))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log))

lda = LinearDiscriminantAnalysis()

lda.fit(X_train_scaled, y_train)

y_pred_lda = lda.predict(X_test_scaled)

print("LDA Classification Report:")

print(classification_report(y_test, y_pred_lda))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lda))
```

OUTPUT:

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

           0       0.82      0.81      0.81        99
           1       0.66      0.67      0.67        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154

Confusion Matrix:
 [[80 19]
 [18 37]]
```

RESULT:

Thus the program to predict the onset of diabetes based on medical measurements is successfully predicted and output is obtained.

EX.NO : 9	WINE QUALITY PREDICTION
DATE :	

AIM:

To predict the quality of wine based on various chemical properties.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

```
import pandas as pd

import numpy as np

import seaborn as sns

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

from sklearn.decomposition import FactorAnalysis
```

```
import pickle

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"

wine_data = pd.read_csv(url, sep=";")

print(wine_data.head())

print(wine_data.isnull().sum())

plt.figure(figsize=(8,6))

sns.histplot(wine_data['quality'], bins=10, kde=True)

plt.title('Distribution of Wine Quality')

plt.xlabel('Wine Quality')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(16,10))

sns.boxplot(data=wine_data, orient='h')

plt.title('Boxplot of Wine Chemical Properties')

plt.show()

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

y = wine_data['quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```

```
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-Squared: {r2}")

fa = FactorAnalysis(n_components=6, random_state=42)

X_fa = fa.fit_transform(X)

X_train_fa, X_test_fa, y_train_fa, y_test_fa = train_test_split(X_fa, y, test_size=0.3, random_state=42)

regressor_fa = LinearRegression()

regressor_fa.fit(X_train_fa, y_train_fa)

y_pred_fa = regressor_fa.predict(X_test_fa)

mse_fa = mean_squared_error(y_test_fa, y_pred_fa)

r2_fa = r2_score(y_test_fa, y_pred_fa)

print(f"Factor Analysis - Mean Squared Error: {mse_fa}")

print(f"Factor Analysis - R-Squared: {r2_fa}")

class WineQualityModel:

    def __init__(self, model):

        self.model = model

    def save_model(self, filename='wine_quality_model.pkl'):

        with open(filename, 'wb') as file:

            pickle.dump(self.model, file)
```

```
        print(f"Model saved to {filename}")

    def load_model(self, filename='wine_quality_model.pkl'):

        with open(filename, 'rb') as file:

            self.model = pickle.load(file)

        print(f"Model loaded from {filename}")

    def predict_quality(self, X):

        return self.model.predict(X)

wine_quality_model = WineQualityModel(regressor)

wine_quality_model.save_model()

wine_quality_model.load_model()

predictions = wine_quality_model.predict_quality(X_test)

print(predictions[:5])
```

OUTPUT:

```
Model saved to wine_quality_model.pkl
Model loaded from wine_quality_model.pkl
[5.35676319 5.09071476 5.62553757 5.44886088 5.74478368]
```

RESULT:

Thus the program to predict the quality of wine based on various chemical properties is successfully executed and output is obtained.

EX.NO : 10	HEART DISEASE PREDICTION
DATE :	

AIM:

To predict heart disease based on clinical parameters.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

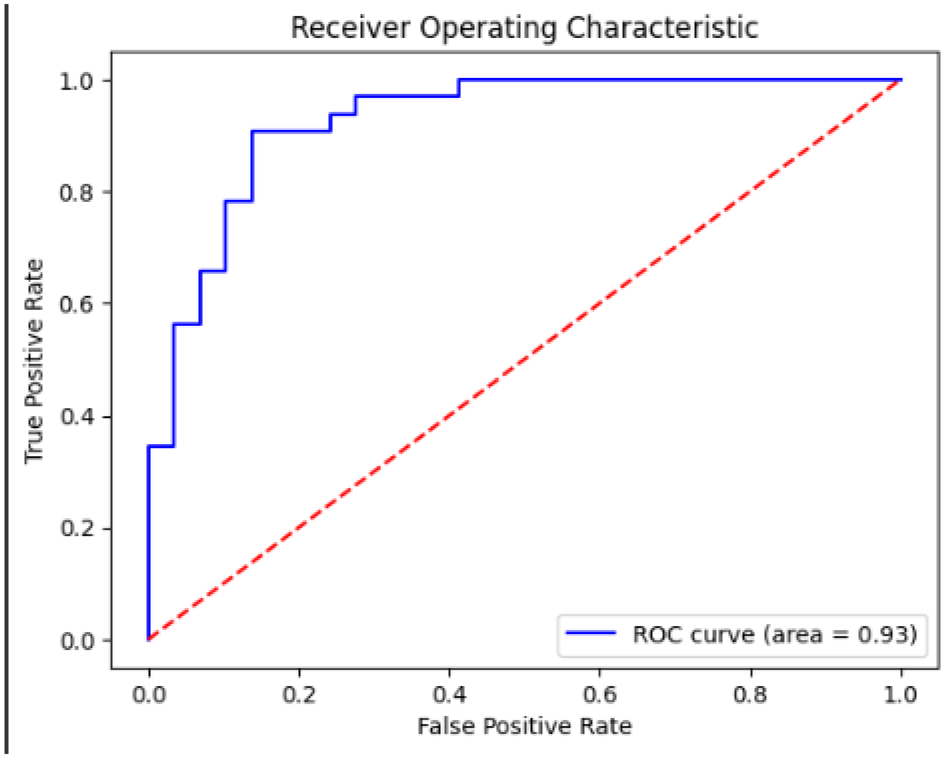
SOURCE CODE:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, roc_curve, auc
```

```python
from sklearn.decomposition import PCA

data = pd.read_csv("/content/dataset.csv")

print(data.info())

sns.pairplot(data, hue='target')

plt.show()

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

y = data['target']

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

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

y_probs = model.predict_proba(X_test)[:, 1]

conf_matrix = confusion_matrix(y_test, y_pred)

fpr, tpr, thresholds = roc_curve(y_test, y_probs)

roc_auc = auc(fpr, tpr)

print("Confusion Matrix:")

print(conf_matrix)

plt.plot(fpr, tpr, color='blue', label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic')

plt.legend(loc='lower right') plt.show()
```

OUTPUT:



RESULT:

Thus the program to predict heart disease based on clinical parameters is successfully executed and output is obtained.

EX.N0 : 11	**BREAST CANCER DIAGNOSIS**
DATE :	

AIM:

To classify tumors as benign or malignant based on features.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

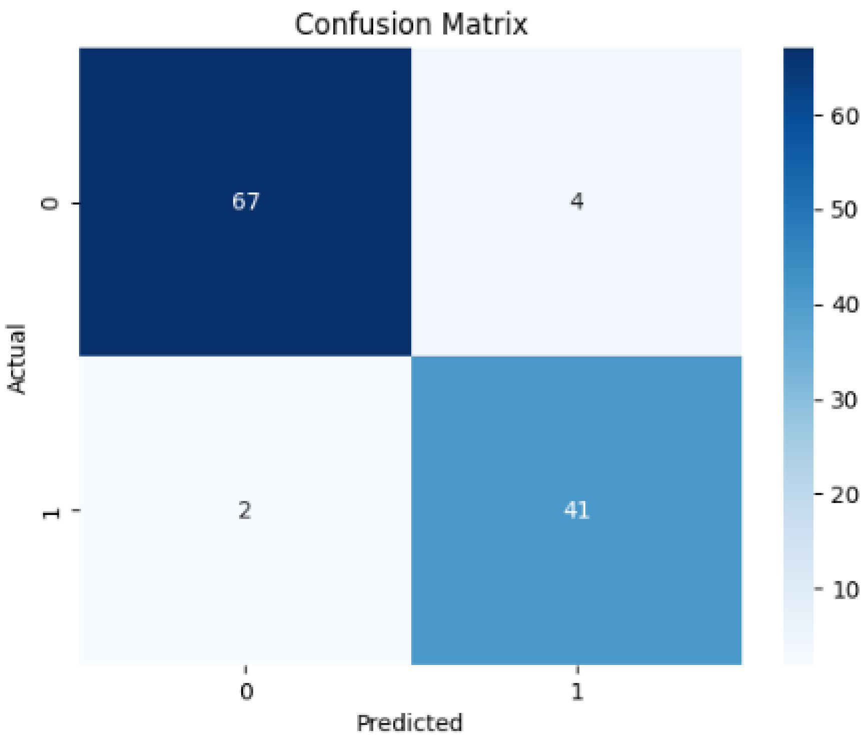
SOURCE CODE:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

data = pd.read_csv('/content/breast cancer.csv')

print(data.info())

print(data.describe())

X = data.drop(['id', 'diagnosis'], axis=1)

y = data['diagnosis'].map({'M': 1, 'B': 0})

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

lda = LinearDiscriminantAnalysis()

X_train_lda = lda.fit_transform(X_train, y_train)

X_test_lda = lda.transform(X_test)

model = LogisticRegression(max_iter=1000)

model.fit(X_train_lda, y_train)

y_pred = model.predict(X_test_lda)

conf_matrix = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)

class_report = classification_report(y_test, y_pred)

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

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()
```

```
print(f'Accuracy: {accuracy:.2f}')

print(class_report)
```

OUTPUT:




```
Accuracy: 0.95
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        71
           1       0.91      0.95      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.94       114
weighted avg       0.95      0.95      0.95       114
```

RESULT:

Thus the classification of tumors as benign or malignant is successfully executed and output is obtained.

EX.N0 : 12	PREDICTING FLIGHT DELAYS
DATE :	

AIM:

To predict flight delays based on historical data.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Encode categorical variable, define feature & testing set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

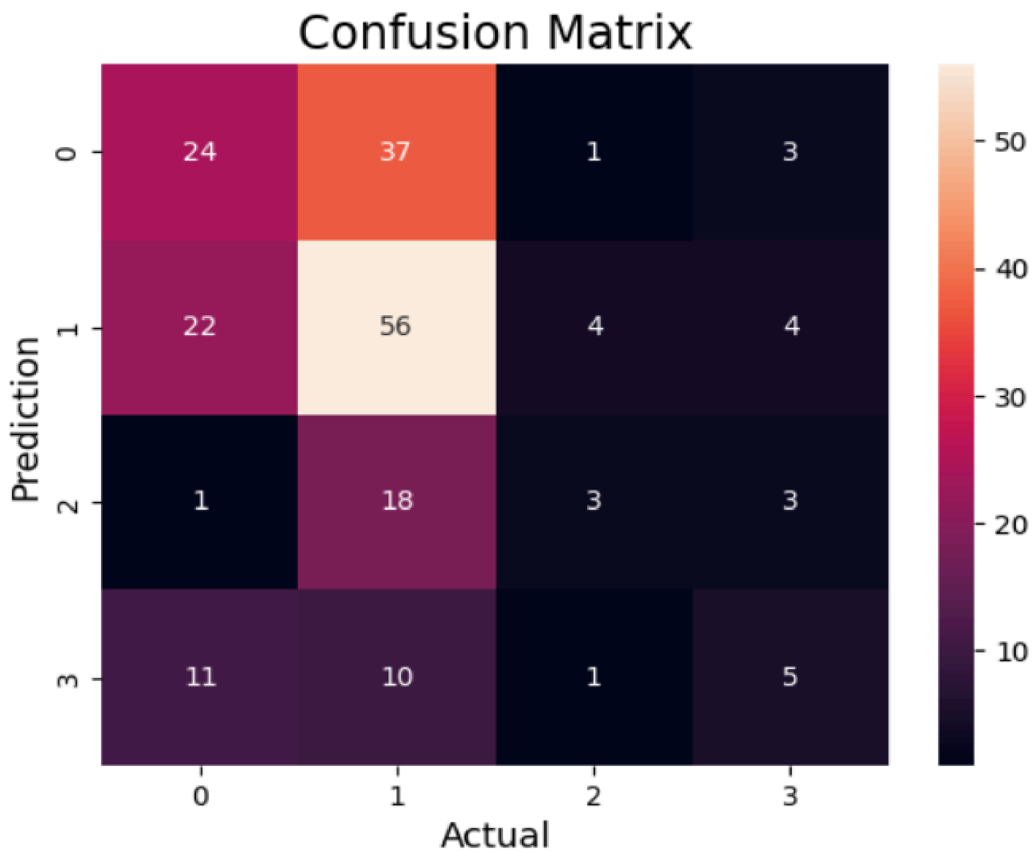
```
# Import necessary libraries

import pandas as pd

import numpy as np

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.cluster import KMeans

from sklearn.metrics import mean_squared_error, r2_score

data = pd.read_csv('flight_delay_dataset.csv')  # Replace with your file path

print("Dataset Head:")

print(data.head())

print("\nDataset Information:")

print(data.info())

print("\nDataset Statistics:")

print(data.describe())

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

plt.plot(data['Date'], data['Delay'], label='Delay over Time')

plt.xlabel('Date')

plt.ylabel('Delay (Minutes)')

plt.title('Flight Delay Trend Over Time')

plt.xticks(rotation=45)

plt.legend()

plt.show()

plt.figure(figsize=(8, 6))

sns.scatterplot(x='Distance', y='Delay', data=data, hue='Airline', palette='viridis')

plt.xlabel('Distance (Miles)')

plt.ylabel('Delay (Minutes)')
```

```
plt.title('Flight Delay vs Distance')

plt.show()

X = data[['Distance', 'Airline_Code', 'Scheduled_Departure_Hour']]

y = data['Delay']

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

regressor = LinearRegression()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"\nMean Squared Error: {mse}")

print(f"R-squared: {r2}")

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred, color='blue', alpha=0.5)

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')

plt.xlabel('Actual Delay')

plt.ylabel('Predicted Delay')

plt.title('Actual vs Predicted Flight Delay')

plt.show()

kmeans = KMeans(n_clusters=3, random_state=42)
```

```
data['Cluster'] = kmeans.fit_predict(data[['Delay', 'Distance']])

plt.figure(figsize=(8, 6))

sns.scatterplot(x='Distance', y='Delay', hue='Cluster', data=data, palette='Set1')

plt.xlabel('Distance')

plt.ylabel('Delay')

plt.title('Clustering of Flights based on Delay and Distance')

plt.show()
```

OUTPUT:




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

           0       0.41      0.37      0.39        65
           1       0.46      0.65      0.54        86
           2       0.33      0.12      0.18        25
           3       0.33      0.19      0.24        27

    accuracy                           0.43       203
   macro avg       0.39      0.33      0.34       203
weighted avg       0.41      0.43      0.41       203
```

RESULT:

Thus the program for predicting flight delays based on historical data is successfully executed and output is obtained.

EX.N0 : 13	TITANIC SURVIVAL PREDICTION
DATE :	

AIM:

To predict whether passengers survived the Titanic disaster.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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, classification_report
```

```
titanic_data = pd.read_csv('tested.csv')

print("Missing values before handling:\n", titanic_data.isnull().sum())

titanic_data = titanic_data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)

titanic_data['Age'].fillna(titanic_data['Age'].median(), inplace=True)

titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0], inplace=True)

titanic_data['Fare'].fillna(titanic_data['Fare'].median(), inplace=True)

print("Missing values after handling:\n", titanic_data.isnull().sum())

titanic_data = pd.get_dummies(titanic_data, columns=['Sex', 'Embarked'], drop_first=True)

print(titanic_data.head())

sns.countplot(x='Survived', data=titanic_data)

plt.title('Survival Count')

plt.show()

plt.hist(titanic_data['Age'], bins=20, color='lightblue')

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

X = titanic_data.drop('Survived', axis=1)

y = titanic_data['Survived']

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

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

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

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

```
Confusion Matrix:
 [[50  0]
 [ 0 34]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        34

    accuracy                           1.00        84
   macro avg       1.00      1.00      1.00        84
weighted avg       1.00      1.00      1.00        84
```

```
PCA-Based Logistic Regression Accuracy: 0.6190476190476191
```

RESULT:

Thus the program to predict whether passengers survived the Titanic disaster is successfully executed and output is obtained.

EX.N0 : 14	**ENERGY CONSUMPTION FORECASTING**
DATE :	

AIM:

To forecast energy consumption based on historical data.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

SOURCE CODE:

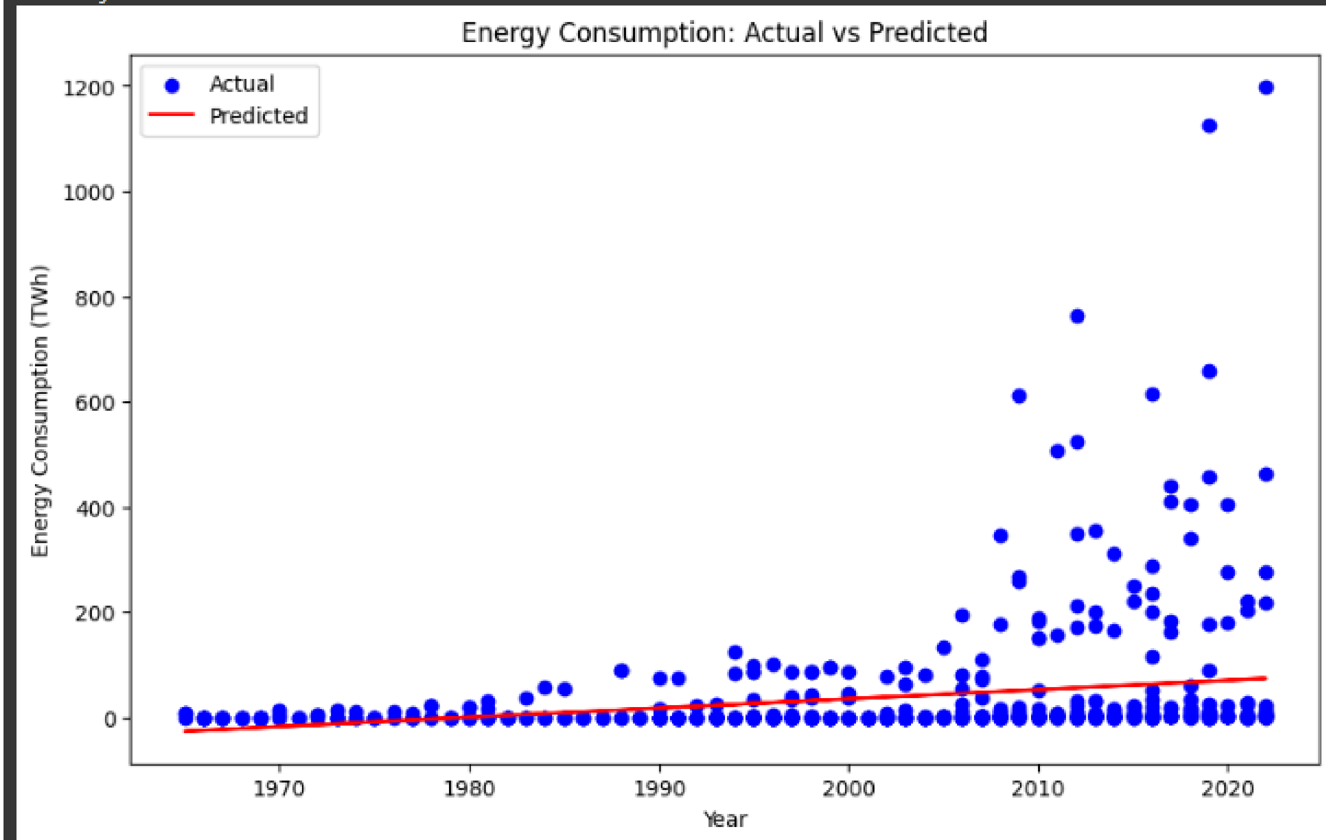
```
import pandas as pd

import numpy as np

import seaborn as sns

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, accuracy_score

energy_data = pd.read_csv("World Energy Consumption.csv")

energy_data_clean = energy_data[['year', 'biofuel_consumption']].dropna()

print(energy_data_clean.head())
```

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

plt.plot(energy_data_clean['year'], energy_data_clean['biofuel_consumption'], marker='o')

plt.title('Energy Consumption Over Time')

plt.xlabel('Year')

plt.ylabel('Energy Consumption (TWh)')

plt.grid(True)

plt.show()

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

sns.heatmap(energy_data_clean.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap of Energy Data')

plt.show()

X = energy_data_clean[['year']]

y = energy_data_clean['biofuel_consumption']

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

threshold = y_train.mean()

y_train_class = (y_train > threshold).astype(int)

y_test_class = (y_test > threshold).astype(int)

model = LinearRegression()

model.fit(X_train, y_train_class)

y_pred_class = (model.predict(X_test) > 0.5).astype(int)

accuracy = accuracy_score(y_test_class, y_pred_class)

print(f"Accuracy Score: {accuracy}")

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

plt.scatter(X_test, y_test, color='blue', label='Actual')
```

```
plt.plot(X_test, model.predict(X_test), color='red', label='Predicted')

plt.title('Energy Consumption: Actual vs Predicted')

plt.xlabel('Year')

plt.ylabel('Energy Consumption (TWh)')

plt.legend()

plt.show()

future_years = pd.DataFrame({'year': np.arange(2025, 2036)})

future_predictions = model.predict(future_years)

for year, consumption in zip(future_years['year'], future_predictions):

    print(f"Year {year}: Predicted Energy Consumption = {consumption:.2f} TWh")
```

OUTPUT:

```
Accuracy Score: 0.8501805054151624
```




```
Year 2025: Predicted Energy Consumption = 78.87 TWh
Year 2026: Predicted Energy Consumption = 80.63 TWh
Year 2027: Predicted Energy Consumption = 82.39 TWh
Year 2028: Predicted Energy Consumption = 84.15 TWh
Year 2029: Predicted Energy Consumption = 85.90 TWh
Year 2030: Predicted Energy Consumption = 87.66 TWh
Year 2031: Predicted Energy Consumption = 89.42 TWh
Year 2032: Predicted Energy Consumption = 91.18 TWh
Year 2033: Predicted Energy Consumption = 92.93 TWh
Year 2034: Predicted Energy Consumption = 94.69 TWh
Year 2035: Predicted Energy Consumption = 96.45 TWh
```

RESULT:

Thus the program to forecast energy consumption based on historical data is successfully executed and output is obtained.

EX.N0 : 15	HUMAN ACTIVITY RECOGNITION
DATE :	

AIM:

To classify different human activities using smartphone sensor data.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

Step 4: Extract and define feature in testing set and training set.

Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

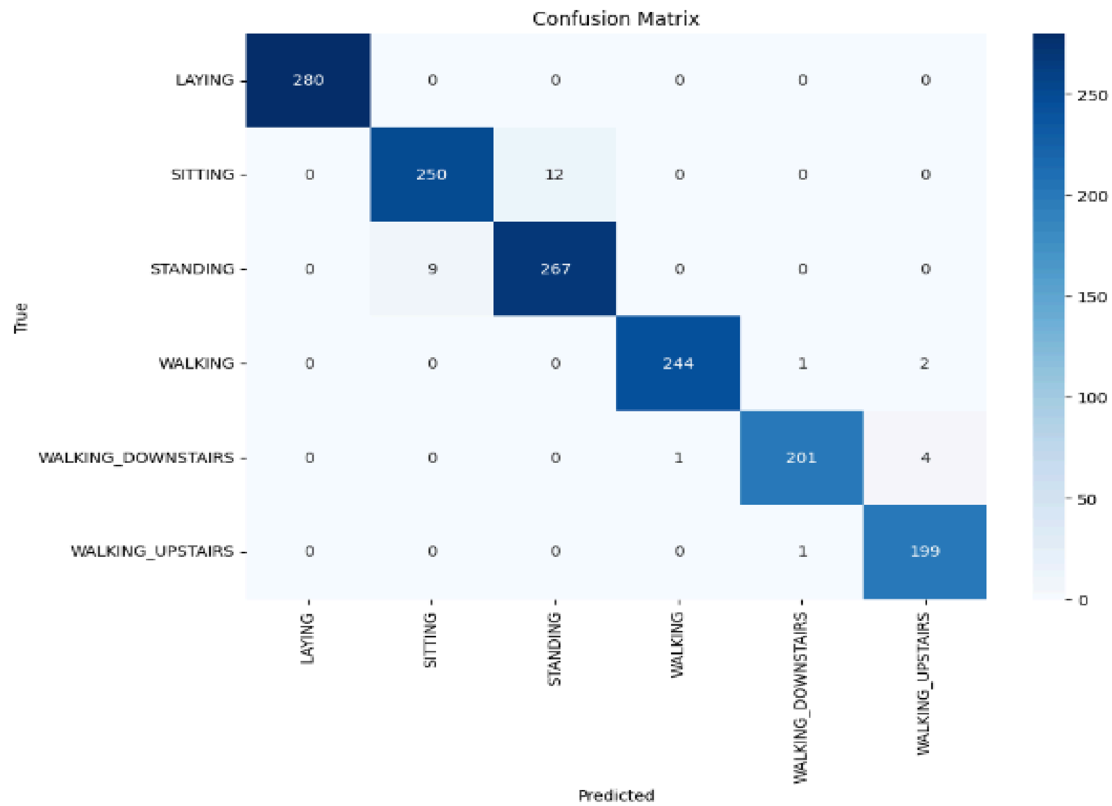
SOURCE CODE:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.decomposition import PCA

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
data = pd.read_csv("train.csv")

label_encoder = LabelEncoder()

data['Activity'] = label_encoder.fit_transform(data['Activity'])

feature_subset = data[['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',

'Activity']]

sns.pairplot(feature_subset, hue="Activity", markers=".")

plt.show()

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

y = data['Activity']

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

pca = PCA(n_components=2)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

plt.figure(figsize=(10, 7))

plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='rainbow', s=50)

plt.title('PCA of Training Data')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()

lda = LinearDiscriminantAnalysis(n_components=2)

X_train_lda = lda.fit_transform(X_train, y_train)
```

```
X_test_lda = lda.transform(X_test)

plt.figure(figsize=(10, 7))

plt.scatter(X_train_lda[:, 0], X_train_lda[:, 1], c=y_train, cmap='rainbow', s=50)

plt.title('LDA of Training Data')

plt.xlabel('Linear Discriminant 1')

plt.ylabel('Linear Discriminant 2')

plt.show()

clf = RandomForestClassifier(n_estimators=100, random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")

print("Classification Report:\n", classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,

yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()
```

OUTPUT:




```
Accuracy: 97.96%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       280
           1       0.97      0.95      0.96       262
           2       0.96      0.97      0.96       276
           3       1.00      0.99      0.99       247
           4       0.99      0.98      0.98       206
           5       0.97      0.99      0.98       200

    accuracy                           0.98      1471
   macro avg       0.98      0.98      0.98      1471
weighted avg       0.98      0.98      0.98      1471
```

RESULT:

Thus the program to classify different human activities using smartphone sensor data.