1. PREDICTING HOUSE PRICES

EX.N0 : 1	**Predicting House Prices**
DATE : 24/07/2024	

PROBLEM STATEMENT: Build a regression model to predict house prices based on features like location, size, and amenities.

PYTHON CONCEPTS: Functions, classes, numeric types, sequences.

VISUALIZATION: Plotting regression line, residual plots.

MULTIVARIATE ANALYSIS: Multiple regression.

DATASET: Kaggle House Prices

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries.

Step 3: Load the dataset.

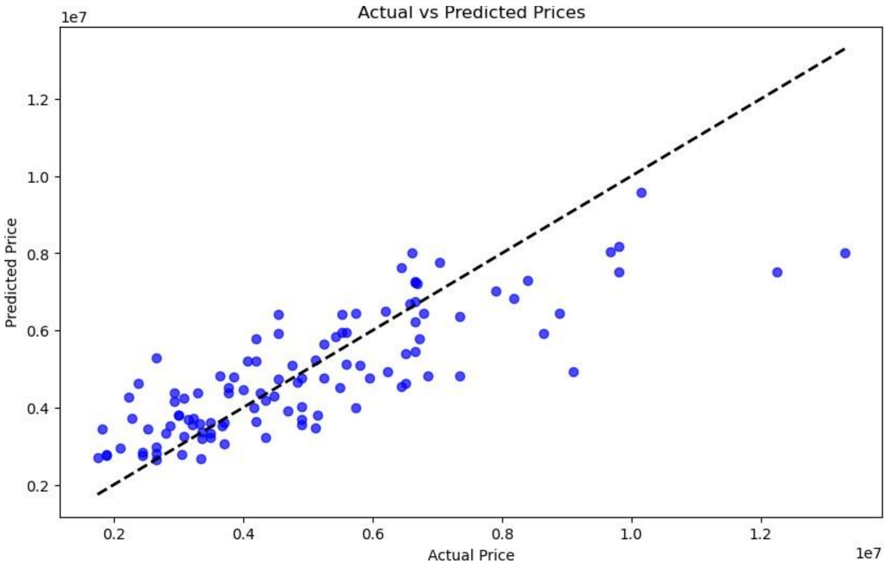
Step 4: Encode categorical variable, define feature & testing set.

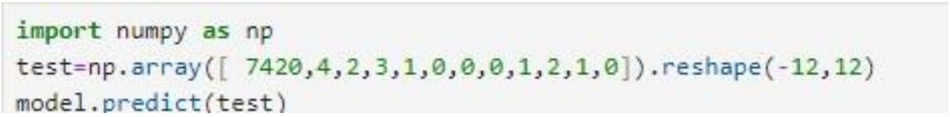
Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

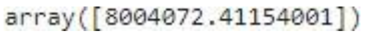
PROGRAM:

```
import pandas as pd 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/HARISH/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}')
```





```
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.

2. CUSTOMER SEGMENTATION FOR AN E-COMMERCE COMPANY

EX.N0 : 2	**Customer Segmentation for an E-commerce Company**
DATE : 05/08/2024	

PROBLEM STATEMENT: Perform cluster analysis to segment customers based on purchasing behaviour.

PYTHON CONCEPTS: Data structures, file reading/writing.

VISUALIZATION: Cluster plots.

MULTIVARIATE ANALYSIS: Cluster analysis with k-means, hierarchical clustering.

DATASET: Online Retail Dataset

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.

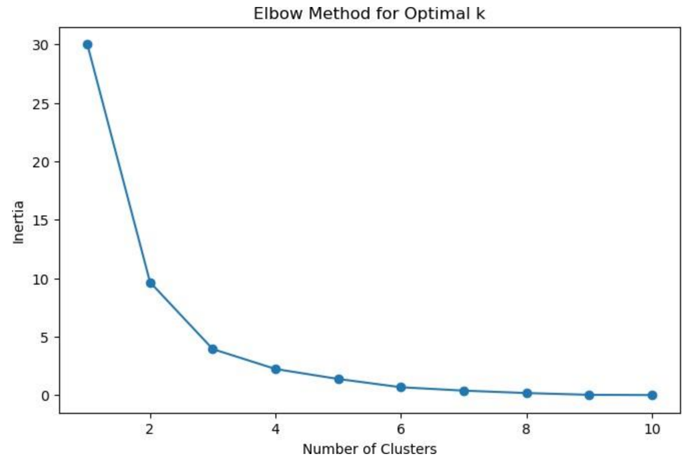
Step 5: Split the dataset into training & testing set, create trained model.

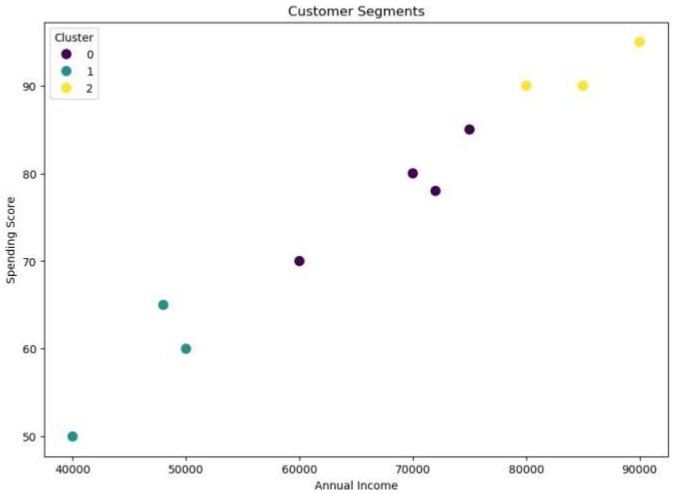
Step 6: Print equal metric & test the cell.

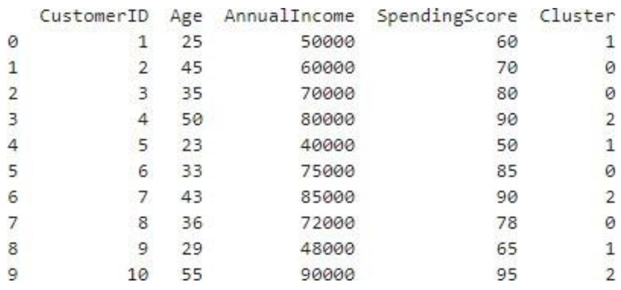
PROGRAM:

```
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.

3. SENTIMENT ANALYSIS OF MOVIE REVIEWS

EX.N0 : 3	**SENTIMENT ANALYSIS OF MOVIE REVIEWS**
DATE : 07/08/2024	

PROBLEM STATEMENT: Classify movie reviews as positive or negative using text Data.

PYTHON CONCEPTS: Text files, sequences, flow controls.

VISUALIZATION: Word cloud, bar plots.

MULTIVARIATE ANALYSIS: PCA for text data, logistic regression.

DATASET: IMDB Movie Reviews.

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.

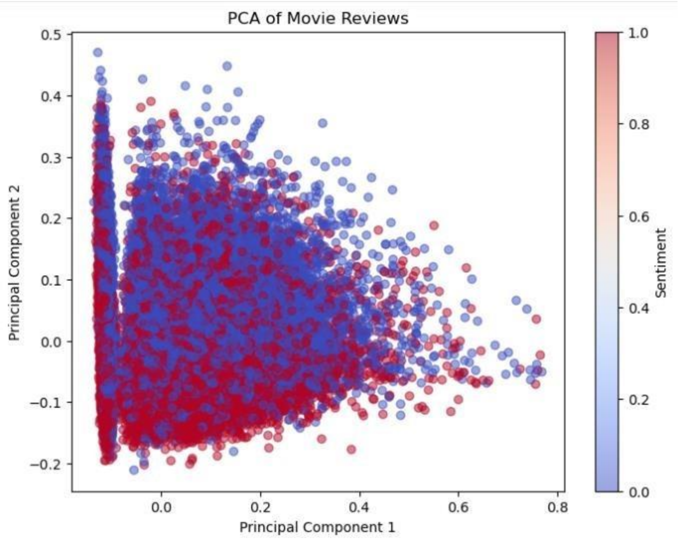
Step 6: Print equal metric & test the cell.

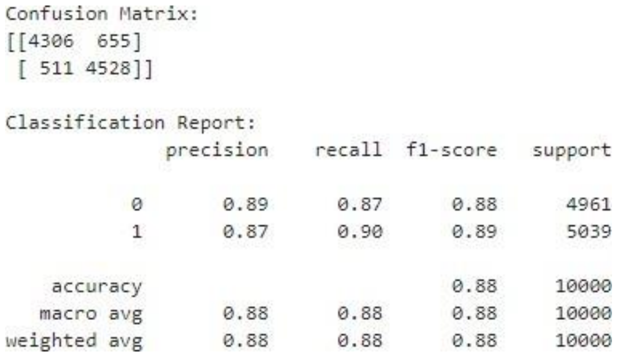
PROGRAM:

```
import pandas as pd import matplotlib.pyplot as plt from
wordcloud import WordCloud from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression from
sklearn.metrics import classification_report, confusion_matrix from
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.

4. STOCK MARKET ANALYSIS

EX.N0 : 4	**STOCK MARKET ANALYSIS**

DATE : 14/08/2024

PROBLEM STATEMENT: Analyse stock market data to predict future stock prices.

PYTHON CONCEPTS: Data structures, file reading/writing, functions.

VISUALIZATION: Line plots, candlestick charts.

MULTIVARIATE ANALYSIS: Time series analysis, regression.

DATASET: Yahoo Finance Stock 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.

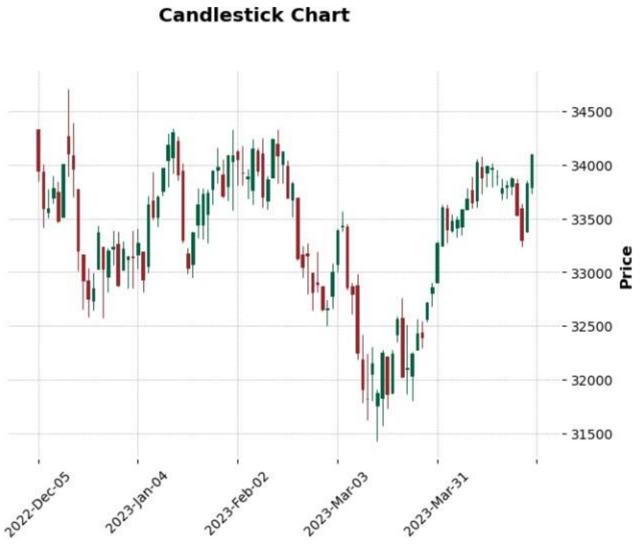
Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

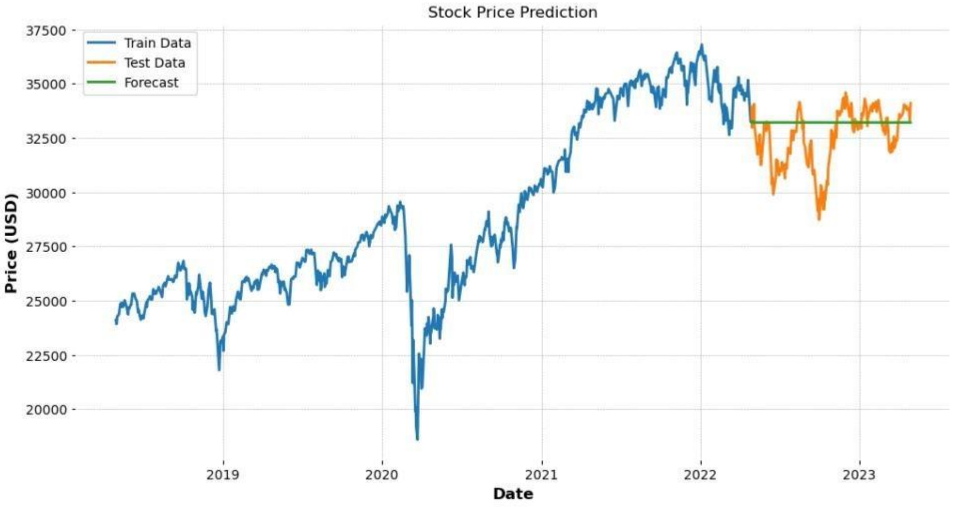
PROGRAM:

```
import pandas as pd 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\HARISH\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:] # Reduce data points for candlestick chart
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:



```
RMSE: 1517.2016308018185
```



RESULT:

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

5. LOAN DEFAULT PREDICTION

EX.N0 : 5	**LOAN DEFAULT PREDICTION**
DATE : 21/08/2024	

PROBLEM STATEMENT: Predict loan default probability based on borrower information.

PYTHON CONCEPTS: Classes, functions, sequences.

VISUALIZATION: ROC curve, bar plots.

MULTIVARIATE ANALYSIS: Logistic regression, factor analysis.

DATASET: Lending Club Loan 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.

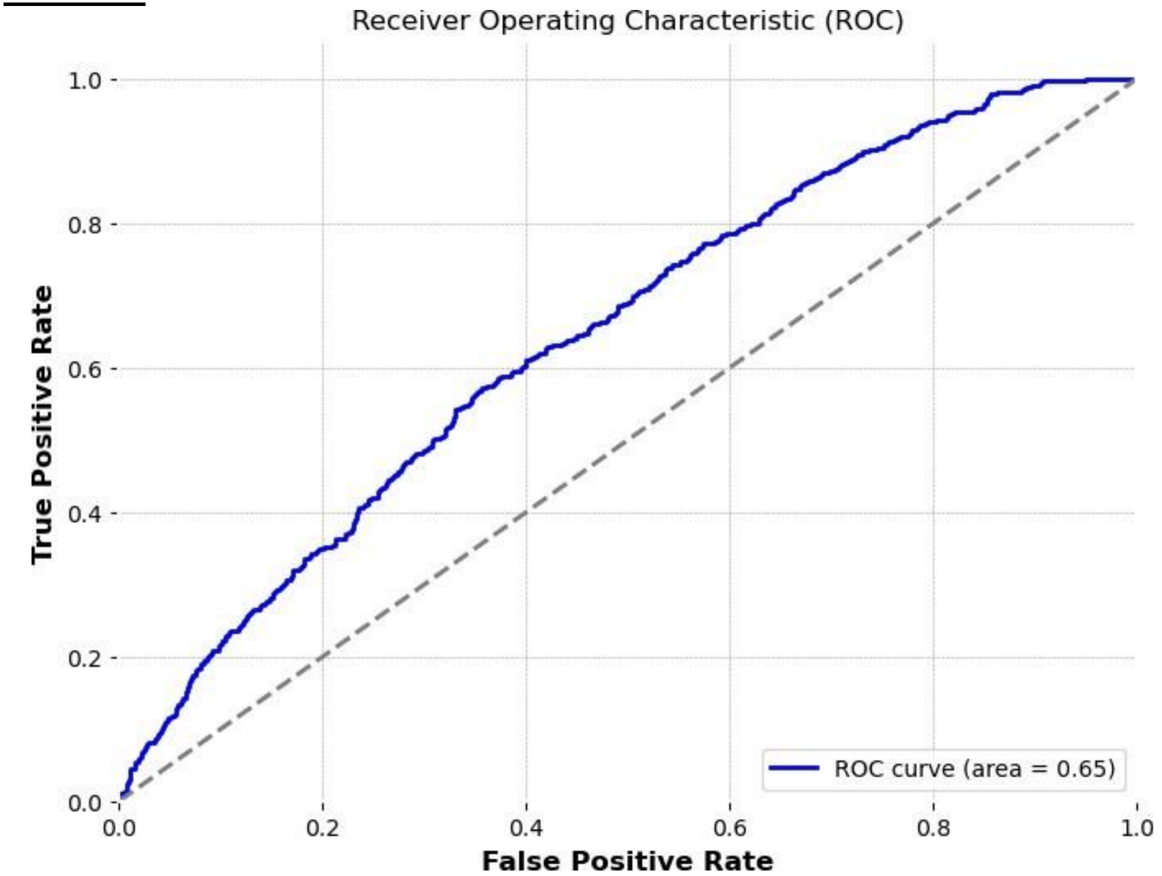
Step 6: Print equal metric & test the cell.

PROGRAM:

```
import pandas as pd 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/HARISH/Downloads/loan_data.csv' # Update path accordinglyif
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.

6. IMAGE CLASSIFICATION

EX.N0 : 6	**IMAGE CLASSIFICATION**
DATE : 04/09/2024	

PROBLEM STATEMENT: Classify images into categories using various features.

PYTHON CONCEPTS: File handling, classes.

VISUALIZATION: Image plots, feature importance plots.

MULTIVARIATE ANALYSIS: PCA, clustering.

DATASET: CIFAR-10 Dataset **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.

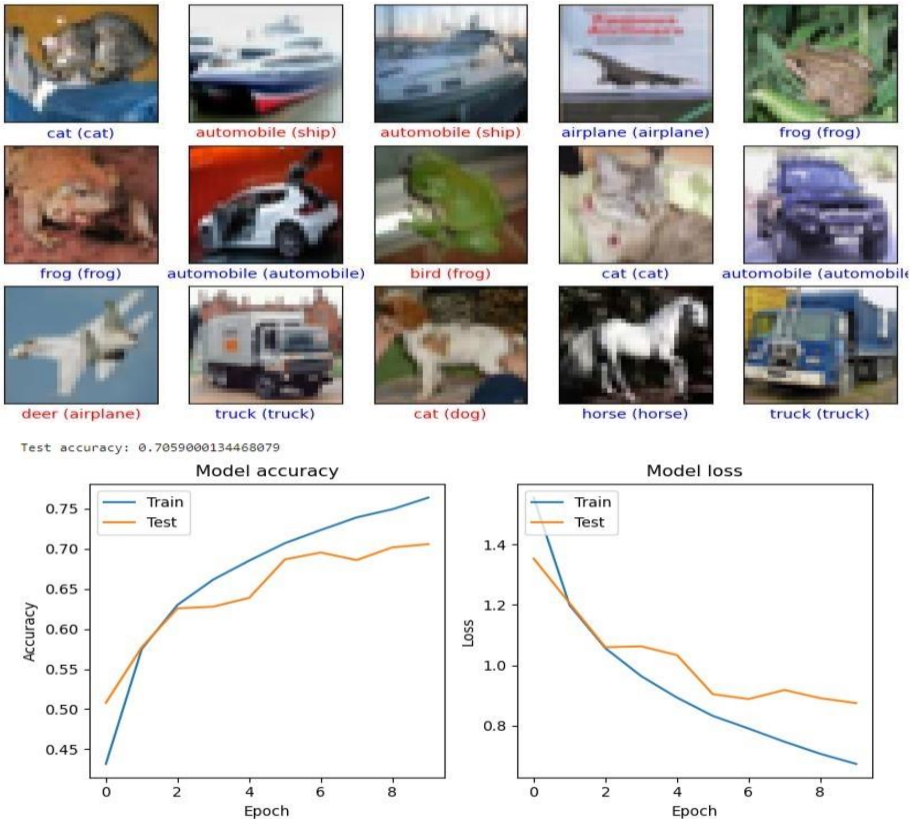
PROGRAM:

```
import tensorflow as tf from tensorflow.keras import layers, models from
tensorflow.keras.preprocessing.image     import     ImageDataGenerator
import matplotlib.pyplot as plt import numpy as np
```

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck'] plt.figure(figsize=(10,10)) for i in range(25):
plt.subplot(5,5,i+1) plt.xticks([]) plt.yticks([]) plt.grid(False)
plt.imshow(X_train[i], cmap=plt.cm.binary)
plt.xlabel(class_names[y_train[i][0]]) plt.show() model =
models.Sequential([ layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)), layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'), layers.MaxPooling2D((2,
2)), layers.Conv2D(64, (3, 3), activation='relu'), layers.Flatten(),
layers.Dense(64, activation='relu'), layers.Dense(10) ])
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10,
validation_data=(X_test, y_test)) test_loss, test_acc =
model.evaluate(X_test, y_test, verbose=2) print(f"\nTest
accuracy: {test_acc}") plt.figure(figsize=(8, 4)) plt.subplot(1,
2, 1) plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy']) plt.title('Model
accuracy') plt.ylabel('Accuracy') plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left') plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss']) plt.title('Model loss')
plt.ylabel('Loss') plt.xlabel('Epoch') plt.legend(['Train', 'Test'],
loc='upper left') plt.tight_layout() plt.show()
```

```
predictions = model.predict(X_test) plt.figure(figsize=(10, 10)) for i in range(25):
plt.subplot(5, 5, i+1) plt.xticks([]) plt.yticks([]) plt.grid(False) plt.imshow(X_test[i],
cmap=plt.cm.binary) predicted_label = np.argmax(predictions[i]) true_label =
y_test[i][0] color = 'blue' if predicted_label == true_label else 'red'
plt.xlabel(f"{class_names[predicted_label]} ({class_names[true_label]})", color=color)
plt.show()
```

OUTPUT:



```
Test accuracy: 0.7059000134468079
```

RESULT:

Thus, the program for Image Classification is executed successfully.

7. PREDICTING DIABETES

EX.N0 : 7	**PREDICTING DIABETES**
DATE : 11/09/2024	

PROBLEM STATEMENT: Predict the onset of diabetes based on medical measurements.

PYTHON CONCEPTS: Data structures, numeric types, functions.

VISUALIZATION: Scatter plots, heatmaps.

MULTIVARIATE ANALYSIS: Logistic regression, LDA.

DATASET: Pima Indians Diabetes Database **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.

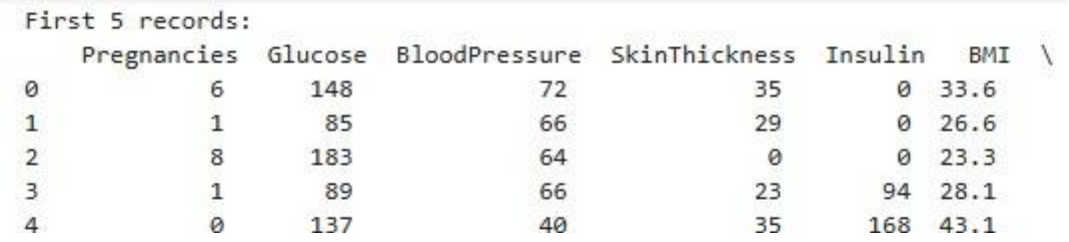
Step 5: Split the dataset into training & testing set, create trained model.

Step 6: Print equal metric & test the cell.

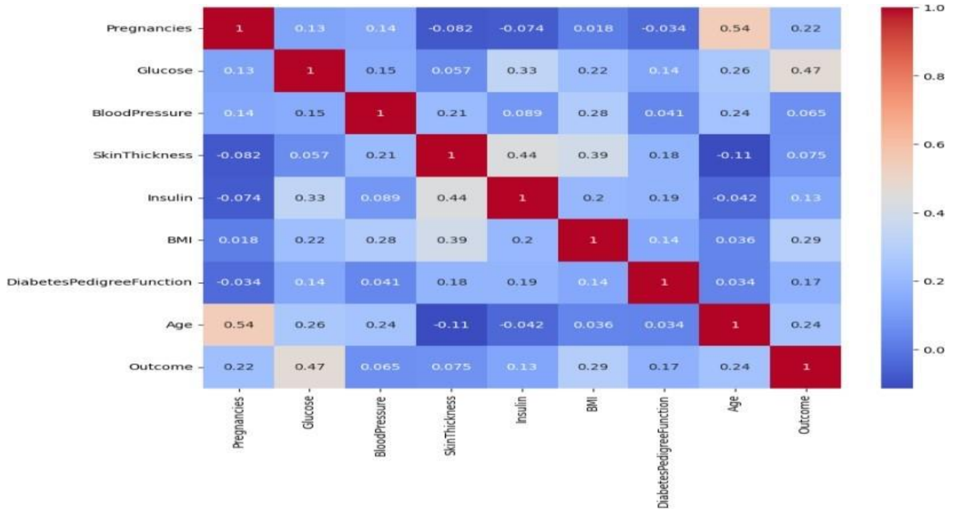
PROGRAM:

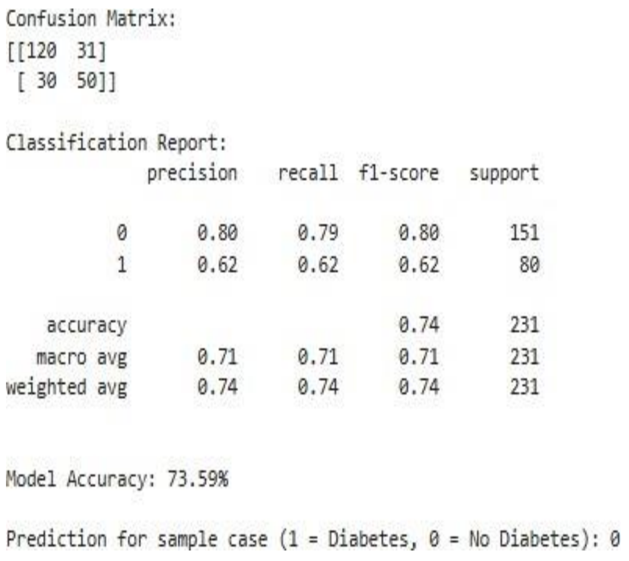
```
import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from
sklearn.model_selection import train_test_split from sklearn.linear_model import
LogisticRegression from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score url = https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age', 'Outcome']
data = pd.read_csv(url, header=None, names=columns)
print("First 5 records:\n", data.head()) print("\nStatistical
Summary:\n", data.describe()) print("\nDataset Info:\n")
print(data.info()) sns.pairplot(data, hue='Outcome') plt.show()
correlation_matrix = data.corr() plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
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=0.3, 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)) accuracy =
accuracy_score(y_test, y_pred) print(f"\nModel Accuracy: {accuracy * 100:.2f}%") sample =
X_test.iloc[0].values.reshape(1, -1) sample_prediction = model.predict(sample)
```

print(f"\nPrediction for sample case (1 = Diabetes, 0 = No Diabetes): {sample_prediction[0]}") **OUTPUT:**



```
First 5 records:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1
```





```
Confusion Matrix:
[[120  31]
 [ 30  50]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.79      0.80       151
           1       0.62      0.62      0.62        80

    accuracy                           0.74       231
   macro avg       0.71      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231


Model Accuracy: 73.59%

Prediction for sample case (1 = Diabetes, 0 = No Diabetes): 0
```

RESULT:

Thus, the program for predicting diabetes is executed successfully.

8. WINE QUALITY PREDICTION

EX.N0 : 8	**WINE QUALITY PREDICTION**
DATE : 18/09/2024	

PROBLEM STATEMENT: Predict the quality of wine based on various chemical properties.

PYTHON CONCEPTS: Classes, sequences, file handling.

VISUALIZATION: Histograms, box plots.

MULTIVARIATE ANALYSIS: Multiple regression, factor analysis.

DATASET: Wine Quality Dataset **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.

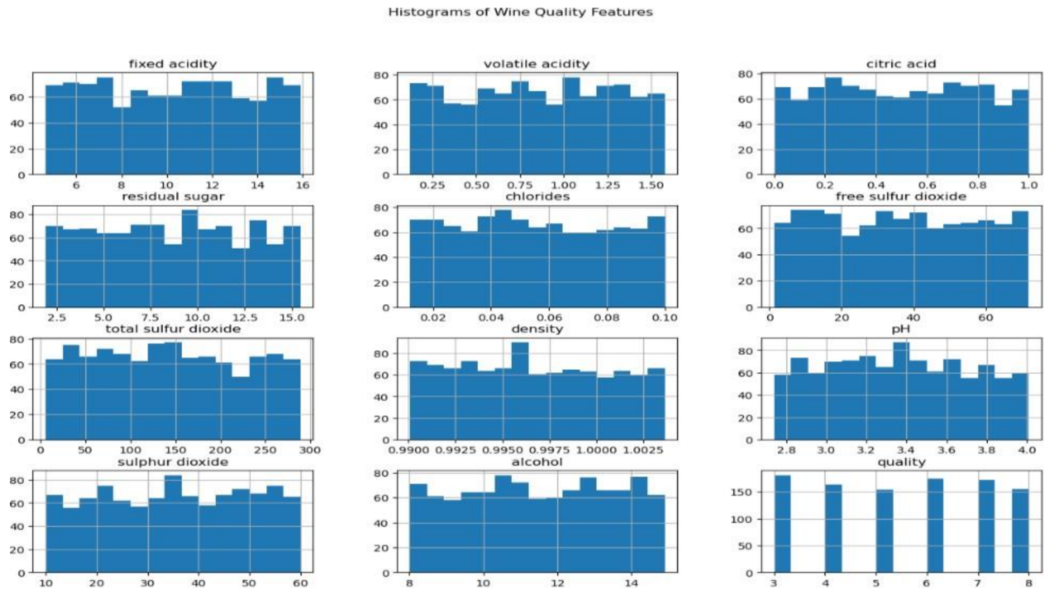
PROGRAM:

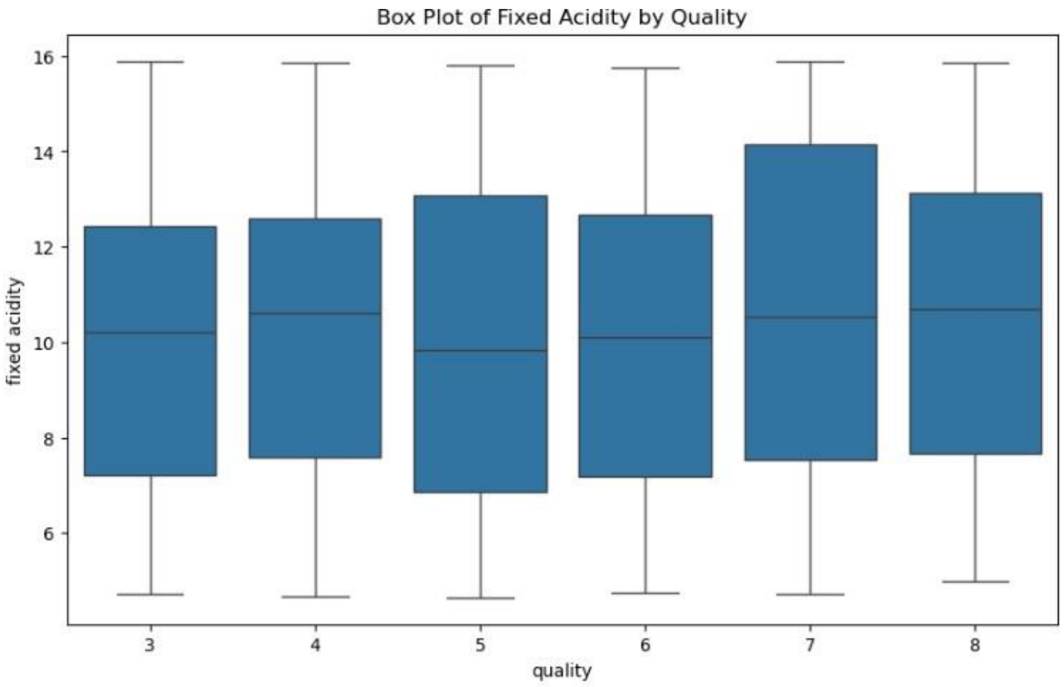
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
r2_score class WineQualityPredictor: def init (self,
num_samples=1000): self.num_samples =
num_samples self.data = None self.model = None
def generate_data(self): np.random.seed(42) quality
= np.random.randint(3, 9, self.num_samples) #
Quality scores between 3 and 8 fixed_acidity =
np.random.uniform(4.6, 15.9, self.num_samples)
```

```
volatile_acidity = np.random.uniform(0.12, 1.58,
self.num_samples) citric_acid =
np.random.uniform(0, 1, self.num_samples)
residual_sugar = np.random.uniform(1.9, 15.5,
self.num_samples) chlorides =
np.random.uniform(0.012, 0.1, self.num_samples)
free_sulfur_dioxide = np.random.uniform(1, 72,
self.num_samples) total_sulfur_dioxide =
np.random.uniform(6, 289, self.num_samples)
density = np.random.uniform(0.99007, 1.00369,
self.num_samples) pH = np.random.uniform(2.74,
4.01, self.num_samples) sulfur_dioxide =
np.random.uniform(10, 60, self.num_samples)
alcohol = np.random.uniform(8.0, 14.9,
self.num_samples) self.data = pd.DataFrame({
'fixed acidity': fixed_acidity, 'volatile acidity': volatile_acidity, 'citric acid': citric_acid,
'residual sugar': residual_sugar, 'chlorides': chlorides, 'free sulfur dioxide': free_sulfur_dioxide,
'total sulfur dioxide': total_sulfur_dioxide, 'density': density, 'pH': pH, 'sulphur
dioxide': sulfur_dioxide, 'alcohol': alcohol, 'quality': quality })
print(f"Synthetic Data Generated: {self.data.shape[0]} rows and {self.data.shape[1]} columns") def
visualize_data(self):
self.data.hist(bins=15, figsize=(15, 10))
plt.suptitle('Histograms of Wine Quality Features')
plt.show() plt.figure(figsize=(10, 6))
sns.boxplot(x='quality', y='fixed acidity',
data=self.data) plt.title('Box Plot of Fixed Acidity by
Quality') plt.show() def preprocess_data(self): X =
self.data.drop('quality', axis=1) y = self.data['quality']
return X, y def train_model(self, X, y):
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
self.model = LinearRegression() self.model.fit(X_train, y_train) y_pred =
self.model.predict(X_test) return y_train, y_test, y_pred def evaluate_model(self, y_test,
y_pred): mse = mean_squared_error(y_test, y_pred) r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}') print(f'R^2 Score: {r2}') def predict_quality(self,
input_features):
input_df = pd.DataFrame([input_features], columns=self.data.columns[:-
1]) prediction = self.model.predict(input_df) return prediction[0] def
```

```
run(self): self.generate_data() self.visualize_data() X, y =
self.preprocess_data() y_train, y_test, y_pred = self.train_model(X, y)
self.evaluate_model(y_test, y_pred) if          name    == "     main     ":
wine_predictor                                                           =
WineQualityPredictor(num_samples=1000)
wine_predictor.run() example_features = {
'fixed acidity': 7.4, 'volatile acidity': 0.7, 'citric acid': 0.0,
'residual sugar': 1.9, 'chlorides': 0.076, 'free sulfur dioxide': 11.0,
'total sulfur dioxide': 34.0, 'density': 0.9978, 'pH': 3.51,
'sulphur    dioxide':    45.0,    'alcohol':    9.4    }    predicted_quality    =
wine_predictor.predict_quality(example_features)      print(f'Predicted
Wine Quality: {predicted_quality:.2f}')
```

OUTPUT:





```
Mean Squared Error: 2.8525212491984275
R^2 Score: -0.0010251435985495494
Predicted Wine Quality: 5.51
```

RESULT:

Thus, the program for wine quality prediction is executed successfully.

9. HEART DISEASE PREDICTION

EX.N0 : 9	**HEART DISEASE PREDICTION**
DATE : 07/10/2024	

PROBLEM STATEMENT: Predict heart disease based on clinical parameters **PYTHON**

CONCEPTS: Functions, data structures.

VISUALIZATION: Pair plots, ROC curve.

MULTIVARIATE ANALYSIS: Logistic regression, PCA.

DATASET: Heart Disease Dataset **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.

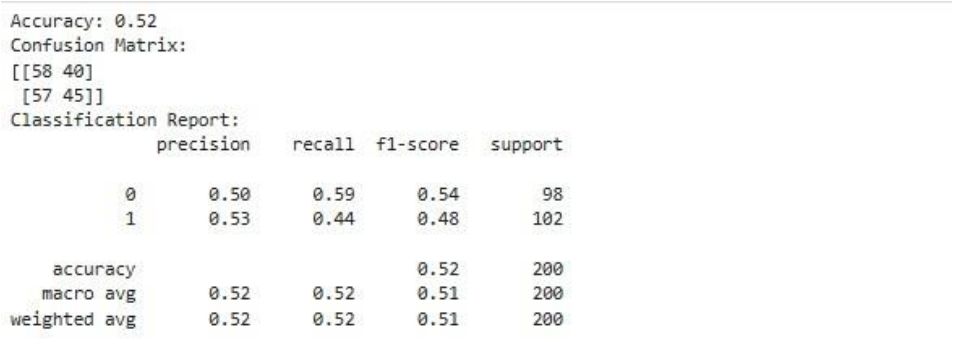
Step 6: Print equal metric & test the cell.

PROGRAM:

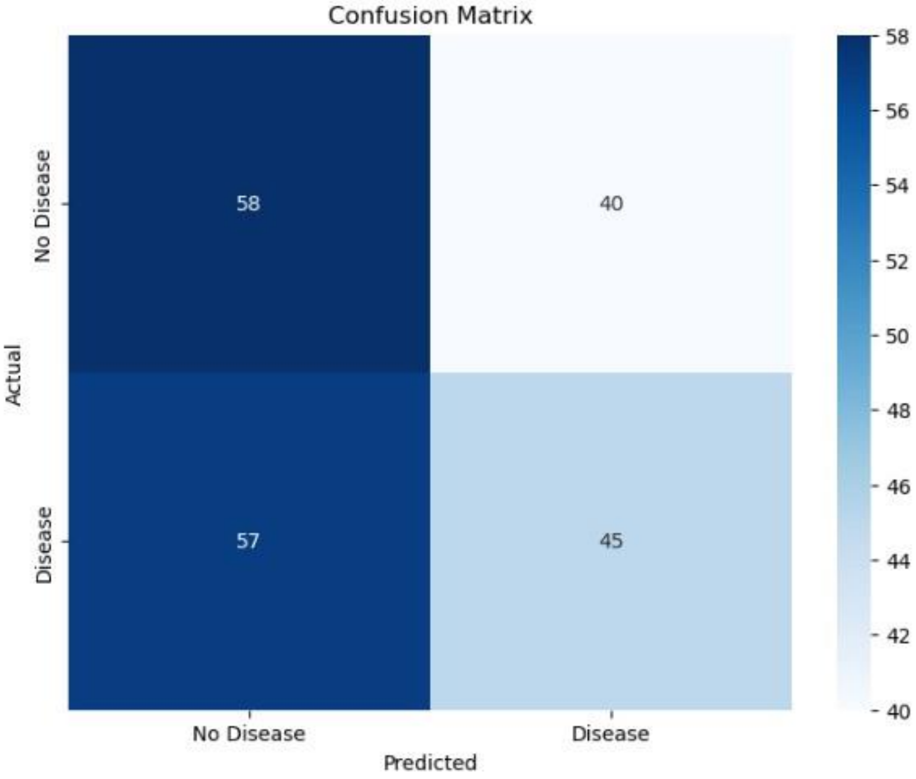
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt import seaborn as sns from
sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import
LogisticRegression from sklearn.metrics import
accuracy_score, confusion_matrix,
classification_report np.random.seed(42) # For
reproducibility num_samples = 1000 age =
np.random.randint(30, 80, num_samples) sex =
np.random.randint(0, 2, num_samples) cp =
np.random.randint(0, 4, num_samples) trestbps =
np.random.randint(90, 200, num_samples) chol =
np.random.randint(150, 300, num_samples) fbs =
np.random.randint(0, 2, num_samples) restecg =
np.random.randint(0, 2, num_samples) thalach =
np.random.randint(60, 200, num_samples) exang =
np.random.randint(0, 2, num_samples) oldpeak =
np.random.uniform(0, 6, num_samples) slope =
np.random.randint(0, 3, num_samples) ca =
np.random.randint(0, 4, num_samples) thal =
np.random.randint(1, 4, num_samples) target =
np.random.randint(0, 2, num_samples) data =
pd.DataFrame({
'age': age, 'sex': sex, 'cp': cp,
'trestbps': trestbps, 'chol': chol,
'fbs': fbs, 'restecg': restecg, 'thalach': thalach, 'exang': exang,
'oldpeak': oldpeak, 'slope': slope, 'ca': ca,
'thal': thal, 'target': target}) 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)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) X_test
= scaler.transform(X_test) model =
LogisticRegression() model.fit(X_train,
y_train) y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test,
y_pred) class_report =
classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:') print(conf_matrix)
print('Classification Report:')
print(class_report) plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=['No
Disease','Disease'],
yticklabels=['No Disease', 'Disease'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show() importance =
model.coef_[0] features =
X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6)) sns.barplot(data=importance_df, x='Importance',
y='Feature', palette='viridis') plt.title('Feature Importance')
plt.xlabel('Coefficient Value') plt.ylabel('Features') plt.axvline(0, color='red',
linestyle='--') # Adding a vertical line at 0 plt.show()
```

OUTPUT:



```
Accuracy: 0.52
Confusion Matrix:
[[58 40]
 [57 45]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.59      0.54        98
           1       0.53      0.44      0.48       102

    accuracy                           0.52       200
   macro avg       0.52      0.52      0.51       200
weighted avg       0.52      0.52      0.51       200
```



RESULT:

Thus, the program for heart disease prediction is executed successfully.

10. BREAST CANCER DIAGNOSIS

EX.N0 : 10	**Breast Cancer Diagnosis**

DATE : 09/10/2024

PROBLEM STATEMENT: Classify tumors as benign or malignant based on features.

PYTHON CONCEPTS: Classes, sequences.

VISUALIZATION: Confusion matrix, bar plots.

MULTIVARIATE ANALYSIS: LDA, logistic regression.

DATASET: Breast Cancer Wisconsin Dataset **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.

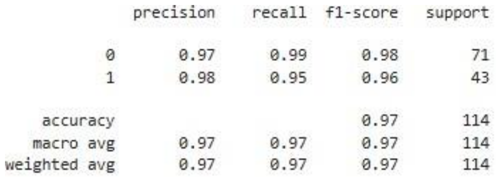
Step 6: Print equal metric & test the cell.

PROGRAM:

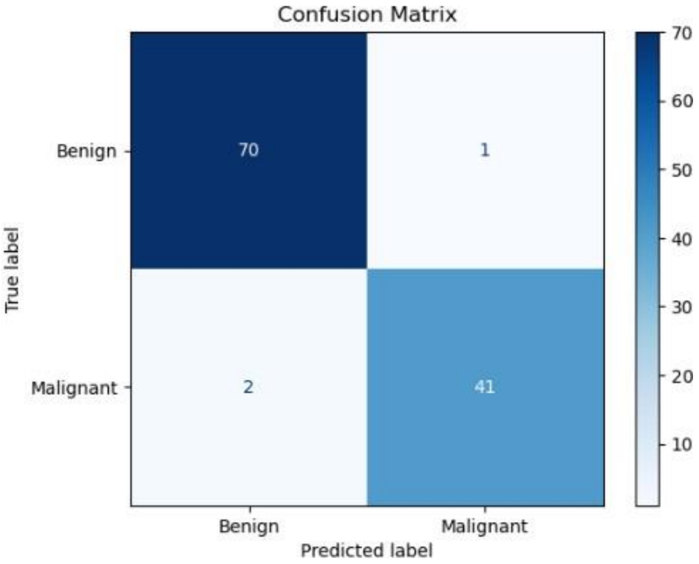
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt import seaborn as sns from
sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import
LogisticRegression from sklearn.metrics import
accuracy_score, confusion_matrix,
classification_report np.random.seed(42) # For
reproducibility num_samples = 1000 age =
np.random.randint(30, 80, num_samples) sex =
np.random.randint(0, 2, num_samples) cp =
np.random.randint(0, 4, num_samples) trestbps =
np.random.randint(90, 200, num_samples) chol =
np.random.randint(150, 300, num_samples) fbs =
np.random.randint(0, 2, num_samples) restecg =
np.random.randint(0, 2, num_samples) thalach =
np.random.randint(60, 200, num_samples) exang =
np.random.randint(0, 2, num_samples) oldpeak =
np.random.uniform(0, 6, num_samples) slope =
np.random.randint(0, 3, num_samples) ca =
np.random.randint(0, 4, num_samples) thal =
np.random.randint(1, 4, num_samples) target =
np.random.randint(0, 2, num_samples) data =
pd.DataFrame({
'age': age, 'sex': sex, 'cp': cp,
'trestbps': trestbps, 'chol': chol,
'fbs': fbs, 'restecg': restecg, 'thalach': thalach, 'exang': exang,
'oldpeak': oldpeak, 'slope': slope, 'ca': ca,
'thal': thal, 'target': target}) 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)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) X_test
= scaler.transform(X_test) model =
LogisticRegression() model.fit(X_train,
y_train) y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test,
y_pred) class_report =
classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:') print(conf_matrix)
print('Classification Report:')
print(class_report) plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=['No
Disease','Disease'],
yticklabels=['No Disease', 'Disease'])
plt.title('Confusion   Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()     importance     =
model.coef_[0]    features    =
X.columns
```

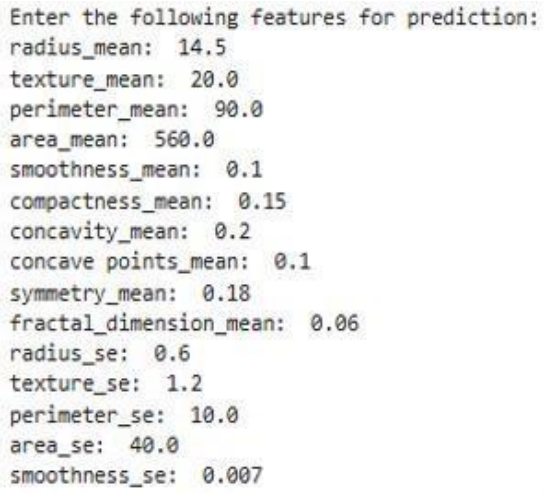
```
importance_df = pd.DataFrame({'Feature': features, 'Importance': importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6)) sns.barplot(data=importance_df, x='Importance',
y='Feature', palette='viridis') plt.title('Feature Importance')
plt.xlabel('Coefficient Value') plt.ylabel('Features') plt.axvline(0, color='red',
linestyle='--') # Adding a vertical line at 0 plt.show()
```

OUTPUT:

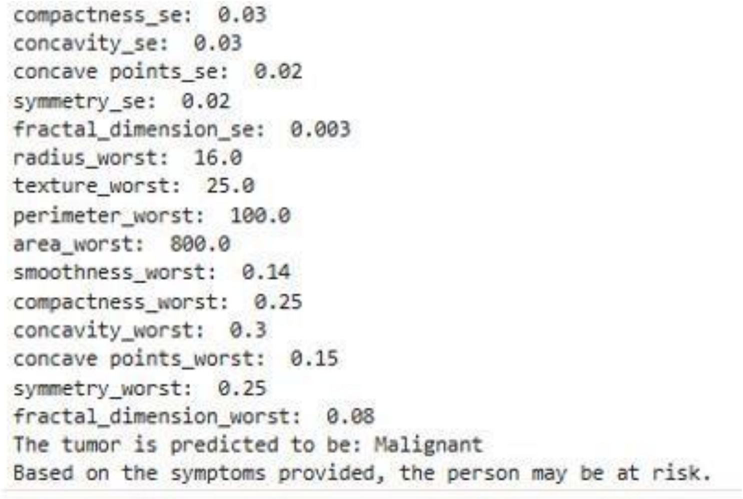


```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```





```
Enter the following features for prediction:
radius_mean:  14.5
texture_mean:  20.0
perimeter_mean:  90.0
area_mean:  560.0
smoothness_mean:  0.1
compactness_mean:  0.15
concavity_mean:  0.2
concave points_mean:  0.1
symmetry_mean:  0.18
fractal_dimension_mean:  0.06
radius_se:  0.6
texture_se:  1.2
perimeter_se:  10.0
area_se:  40.0
smoothness_se:  0.007
```



```
compactness_se:  0.03
concavity_se:  0.03
concave points_se:  0.02
symmetry_se:  0.02
fractal_dimension_se:  0.003
radius_worst:  16.0
texture_worst:  25.0
perimeter_worst:  100.0
area_worst:  800.0
smoothness_worst:  0.14
compactness_worst:  0.25
concavity_worst:  0.3
concave points_worst:  0.15
symmetry_worst:  0.25
fractal_dimension_worst:  0.08
The tumor is predicted to be: Malignant
Based on the symptoms provided, the person may be at risk.
```

RESULT:

Thus, the program for breast cancer diagnosis is executed successfully.

11. PREDICTING FLIGHT DELAYS

EX.N0 : 11	**PREDICTING FLIGHT DELAYS**
DATE : 16/10/2024	

PROBLEM STATEMENT: Predict flight delays based on historical data.

PYTHON CONCEPTS: File reading/writing, functions.

VISUALIZATION: Line plots, scatter plots.

MULTIVARIATE ANALYSIS: Regression, clustering.

DATASET: Flight Delay Dataset **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.

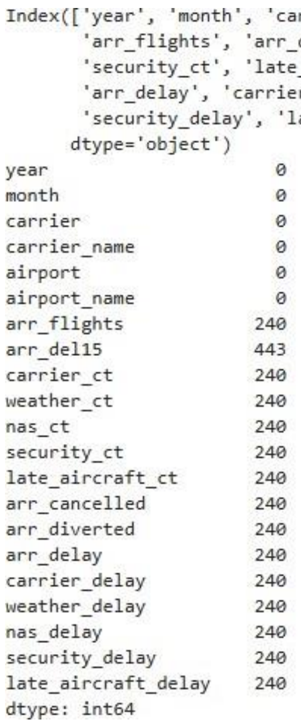
Step 6: Print equal metric & test the cell.

PROGRAM:

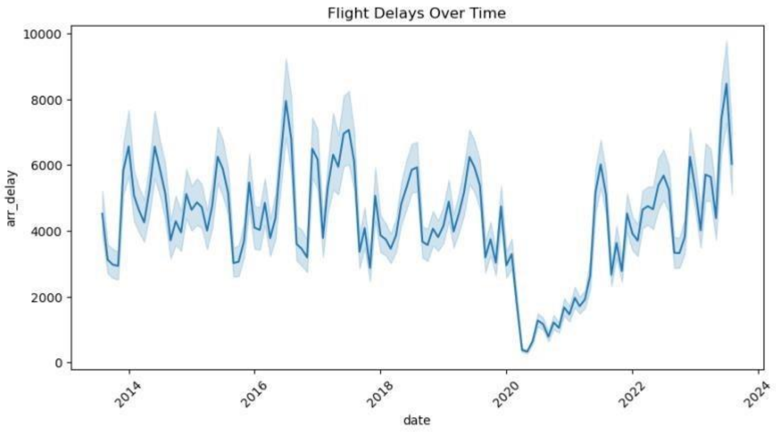
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score df =
pd.read_csv('C:/Users/HARISH/Downloads/Airline
_Delay_Cause.csv') print(df.columns)
print(df.isnull().sum()) df.dropna(inplace=True) # or
```

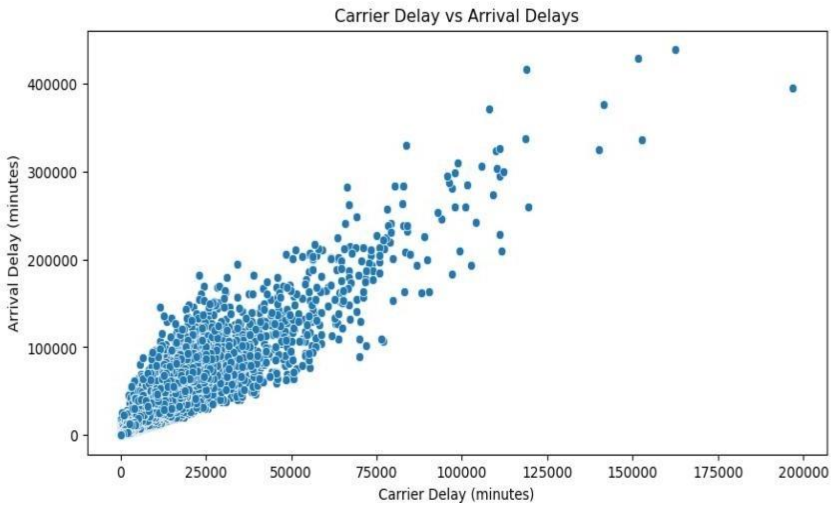
```
df.fillna(method='ffill', inplace=True) if 'year' in
df.columns and 'month' in df.columns:
df['date'] = pd.to_datetime(df[['year', 'month']].assign(day=1))
plt.figure(figsize=(10, 5)) sns.lineplot(data=df, x='date',
y='arr_delay') # Adjust if necessary plt.title('Flight Delays Over
Time') plt.xticks(rotation=45)plt.show()
delay_column = 'arr_delay' # Using 'arr_delay' for now if
'carrier_delay' in df.columns and delay_column in df.columns:
plt.figure(figsize=(10, 5)) sns.scatterplot(data=df, x='carrier_delay',
y=delay_column) # Adjust as needed plt.title('Carrier Delay vs Arrival Delays')
plt.xlabel('Carrier Delay (minutes)') plt.ylabel('Arrival Delay (minutes)')
plt.show()
else: print("Check the delay columns: 'carrier_delay' or 'arr_delay' do not exist in
the DataFrame.") df['day_of_week'] = df['date'].dt.dayofweek # Monday=0,
Sunday=6 features = ['day_of_week', 'arr_flights', 'carrier_ct'] # Modify as needed
X = df[features] y = df[delay_column]
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) predictions = model.predict(X_test)
print('Mean Absolute Error:', mean_absolute_error(y_test, predictions)) print('Mean Squared
Error:', mean_squared_error(y_test, predictions)) print('R-squared:', r2_score(y_test,
predictions)) plt.figure(figsize=(10, 5)) plt.scatter(y_test, predictions) plt.plot([min(y_test),
max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2) # Line of equality
plt.title('Predictions vs Actual Delays') plt.xlabel('Actual Delays') plt.ylabel('Predicted Delays')
plt.show()
```

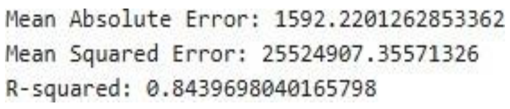
OUTPUT:



```
Index(['year', 'month', 'carrier', 'carrier_name', 'airport', 'airport_name',
       'arr_flights', 'arr_del15', 'carrier_ct', 'weather_ct', 'nas_ct',
       'security_ct', 'late_aircraft_ct', 'arr_cancelled', 'arr_diverted',
       'arr_delay', 'carrier_delay', 'weather_delay', 'nas_delay',
       'security_delay', 'late_aircraft_delay'],
      dtype='object')
year                     0
month                    0
carrier                  0
carrier_name             0
airport                  0
airport_name             0
arr_flights            240
arr_del15              443
carrier_ct             240
weather_ct             240
nas_ct                 240
security_ct            240
late_aircraft_ct       240
arr_cancelled          240
arr_diverted           240
arr_delay              240
carrier_delay          240
weather_delay          240
nas_delay              240
security_delay         240
late_aircraft_delay    240
dtype: int64
```







```
Mean Absolute Error: 1592.2201262853362
Mean Squared Error: 25524907.35571326
R-squared: 0.8439698040165798
```

RESULT:

Thus the program for predicting flight delays is executed successfully.

12. ENERGY CONSUMPTION FORECASTING

EX.N0 : 12	**ENERGY CONSUMPTION FORECASTING**
DATE : 23/10/2024	

PROBLEM STATEMENT: Forecast energy consumption based on historical data.

PYTHON CONCEPTS: Functions, numeric types.

VISUALIZATION: Line plots, heatmaps.

MULTIVARIATE ANALYSIS: Time series analysis, regression.

DATASET: Energy Consumption Dataset **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.

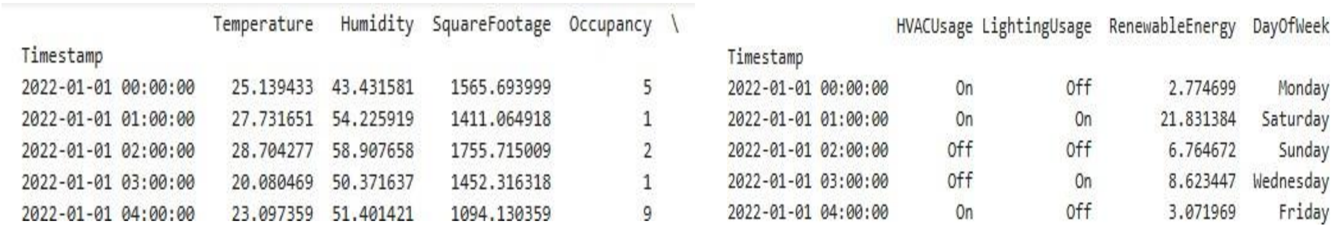
Step 6: Print equal metric & test the cell.

PROGRAM:

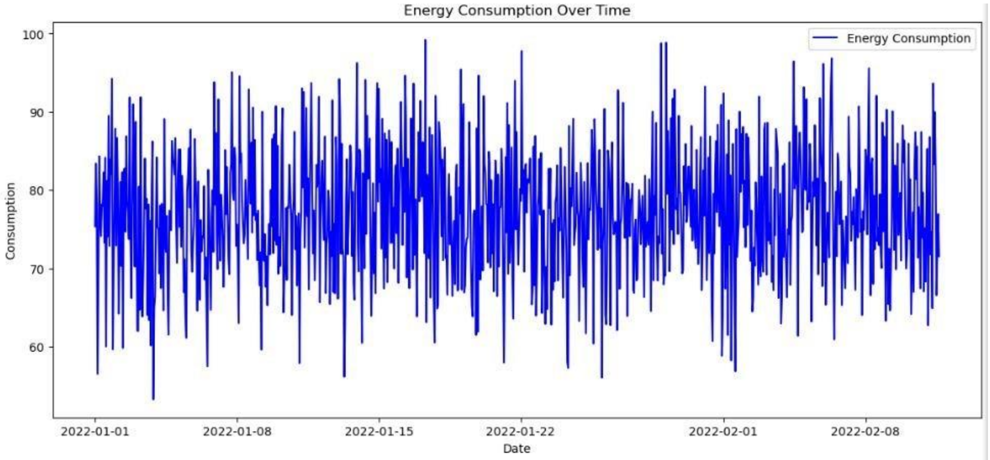
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as
sns from statsmodels.tsa.arima.model import
ARIMA from sklearn.metrics import
mean_squared_error data =
pd.read_csv('C:/Users/HARISH/Downloads/energy_consumption_dataset.csv',
parse_dates=['Timestamp'], index_col='Timestamp') print(data.head())
print(data.info())
data = data.fillna(method='ffill') plt.figure(figsize=(14, 6))
plt.plot(data['EnergyConsumption'], color='blue', label='Energy Consumption')
plt.title('Energy Consumption Over Time') plt.xlabel('Date')
```

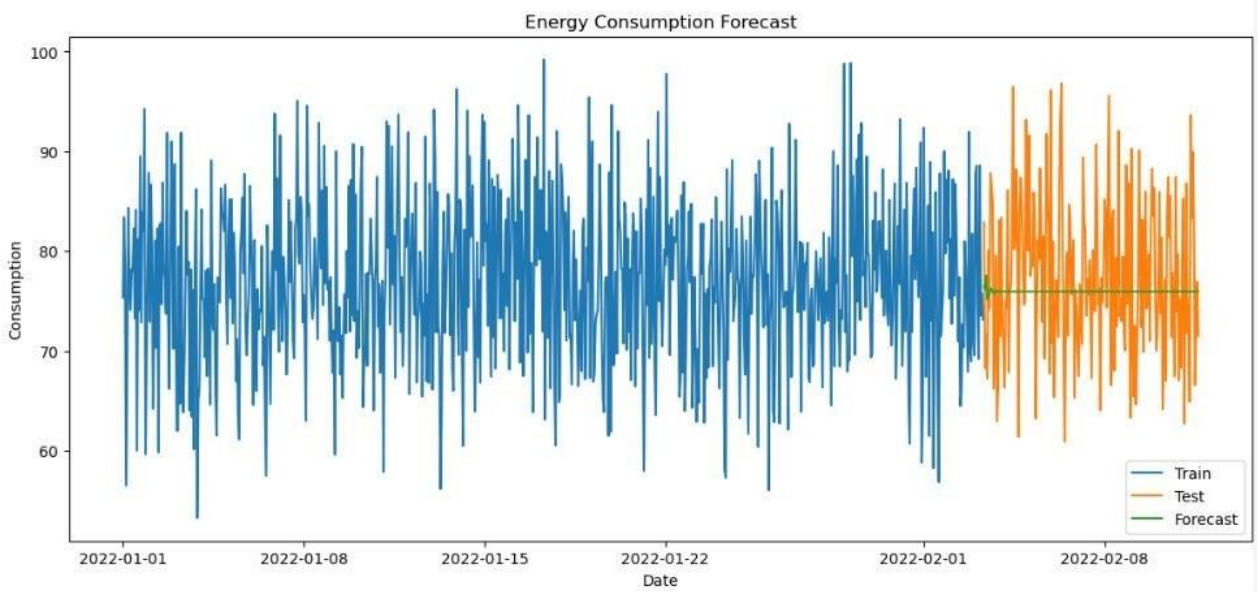
plt.ylabel('Consumption') plt.legend() plt.show() numeric_data = data.select_dtypes(include=[np.number]) plt.figure(figsize=(10, 8)) sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm') plt.title('Correlation Matrix') plt.show() from statsmodels.tsa.seasonal import seasonal_decompose result = seasonal_decompose(data['EnergyConsumption'], model='additive', period=24) # Adjust period based on your data's frequency result.plot() plt.show() train_size = int(len(data) * 0.8) train, test = data['EnergyConsumption'][:train_size], data['EnergyConsumption'][train_size:] model = ARIMA(train, order=(5, 1, 0)) # Adjust (p,d,q) based on your data's behavior fitted_model = model.fit() forecast = fitted_model.forecast(steps=len(test)) forecast_index = test.index mse = mean_squared_error(test, forecast) rmse = np.sqrt(mse) print(f'RMSE: {rmse}') plt.figure(figsize=(14, 6)) plt.plot(train, label='Train') plt.plot(test, label='Test') plt.plot(forecast_index, forecast, label='Forecast') plt.title('Energy Consumption Forecast') plt.xlabel('Date') plt.ylabel('Consumption') plt.legend() plt.show()

OUTPUT:



```
                     Temperature   Humidity  SquareFootage  Occupancy  \
Timestamp
2022-01-01 00:00:00    25.139433  43.431581    1565.693999          5
2022-01-01 01:00:00    27.731651  54.225919    1411.064918          1
2022-01-01 02:00:00    28.704277  58.907658    1755.715009          2
2022-01-01 03:00:00    20.080469  50.371637    1452.316318          1
2022-01-01 04:00:00    23.097359  51.401421    1094.130359          9

                    HVACUsage LightingUsage  RenewableEnergy  DayOfWeek
Timestamp
2022-01-01 00:00:00        On           Off         2.774699     Monday
2022-01-01 01:00:00        On            On        21.831384   Saturday
2022-01-01 02:00:00       Off           Off         6.764672     Sunday
2022-01-01 03:00:00       Off            On         8.623447  Wednesday
2022-01-01 04:00:00        On           Off         3.071969     Friday
```





RESULT:

Thus, the program for energy consumption forecasting is executed successfully.