# 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])

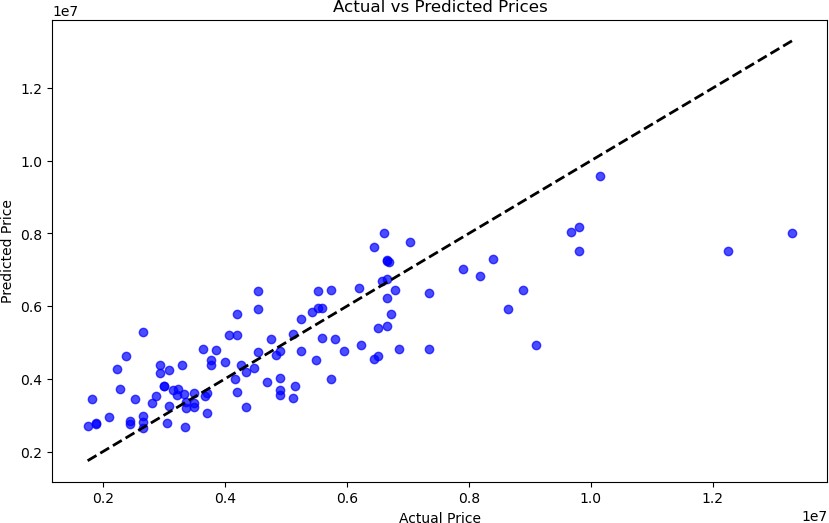
X = housing\_data.drop('price', axis=1)y = housing\_data['price']

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}')







**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],

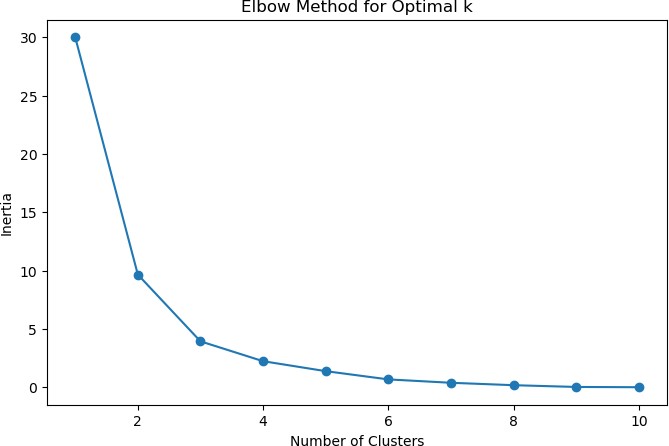
'AnnualIncome': [50000, 60000, 70000, 80000, 40000, 75000, 85000, 72000, 48000, 90000],

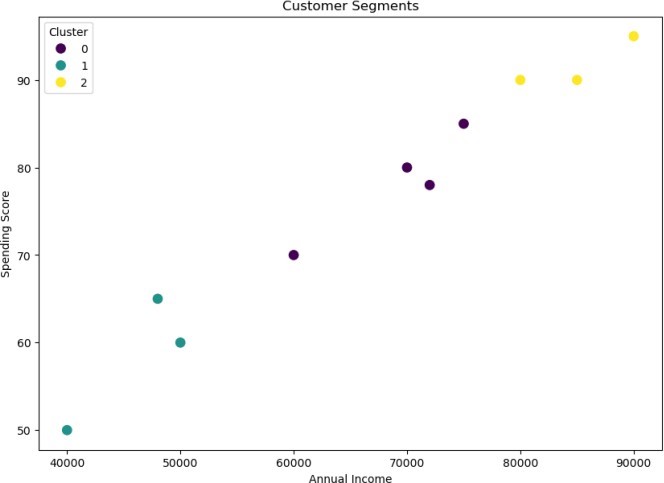
'SpendingScore': [60, 70, 80, 90, 50, 85, 90, 78, 65, 95] } df = pd.DataFrame(data) features = df[['Age', 'AnnualIncome', 'SpendingScore']] scaler = StandardScaler() scaled\_features = scaler.fit\_transform(features) inertia = [] k\_range = range(1, 11) for k in k\_range:

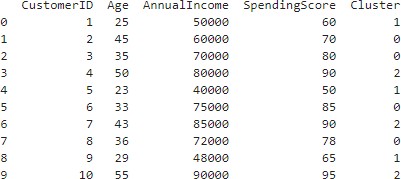
kmeans = KMeans(n\_clusters=k, n\_init=10, random\_state=0) kmeans.fit(scaled\_features) inertia.append(kmeans.inertia\_) plt.figure(figsize=(8, 5)) plt.plot(k\_range, inertia, marker='o') plt.xlabel('Number of Clusters') plt.ylabel('Inertia') plt.title('Elbow Method for Optimal k') plt.show() optimal\_k = 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:**







**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):

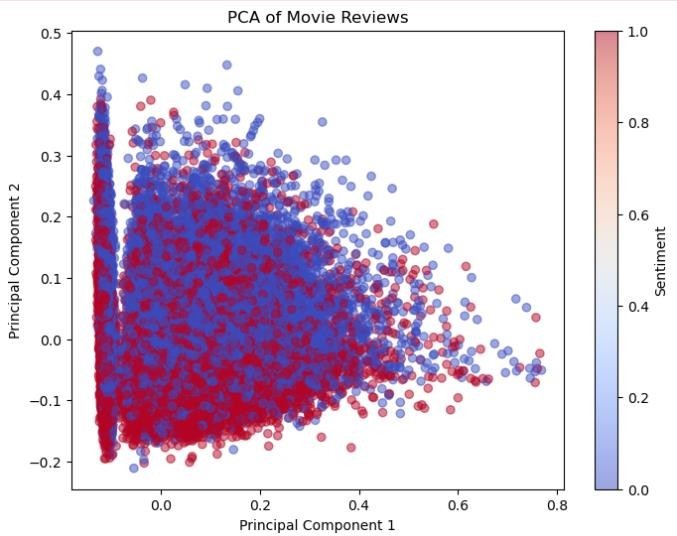
tokens = word\_tokenize(text.lower()) tokens = [stemmer.stem(word) for word in tokens if word.isalpha() and word not in stop\_words] return ' '.join(tokens) df['cleaned\_review'] = df['review'].apply(preprocess\_text) vectorizer = TfidfVectorizer(max\_features=5000)

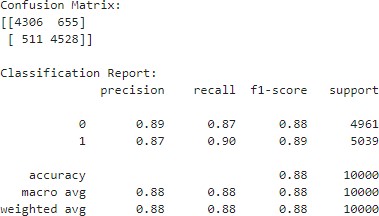
X = vectorizer.fit\_transform(df['cleaned\_review']).toarray() encoder = LabelEncoder() y = encoder.fit\_transform(df['sentiment']) pca = PCA(n\_components=2) X\_pca = pca.fit\_transform(X) plt.figure(figsize=(8, 6)) plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='coolwarm', alpha=0.5) plt.title('PCA 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()  COMPUTATIONAL  STATISTICS | 221501006 |

**OUTPUT:**





**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:**

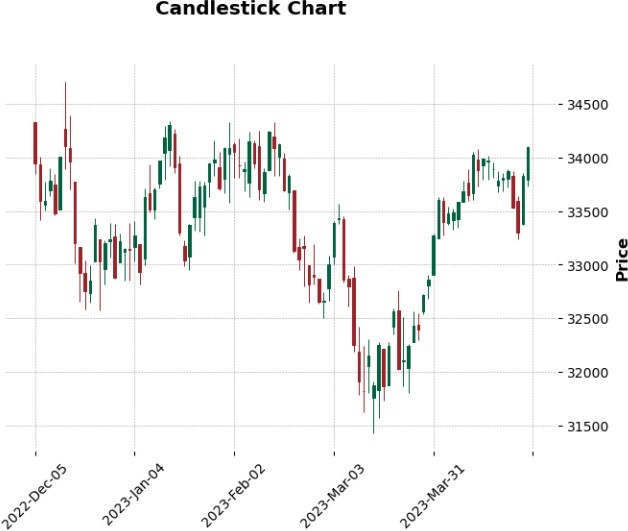
import pandas as pd import matplotlib.pyplot as plt import mplfinance as mpf from statsmodels.tsa.arima.model import ARIMA from sklearn.metrics import mean\_squared\_error import numpy as np

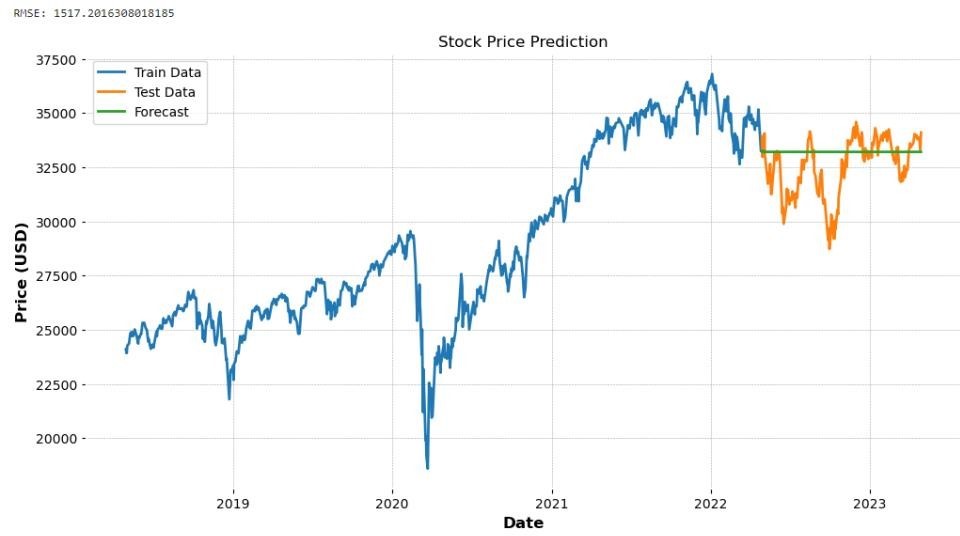
file\_path = r'C:\Users\HARISH\Downloads\yahoo\_data.xlsx' data = pd.read\_excel(file\_path, index\_col='Date', parse\_dates=True) data.rename(columns={'Close\*': 'Close', 'Adj Close\*\*': 'Adj Close'}, inplace=True) data.sort\_index(inplace=True) data.ffill(inplace=True) if 'Adj Close' in data.columns:

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:**





**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

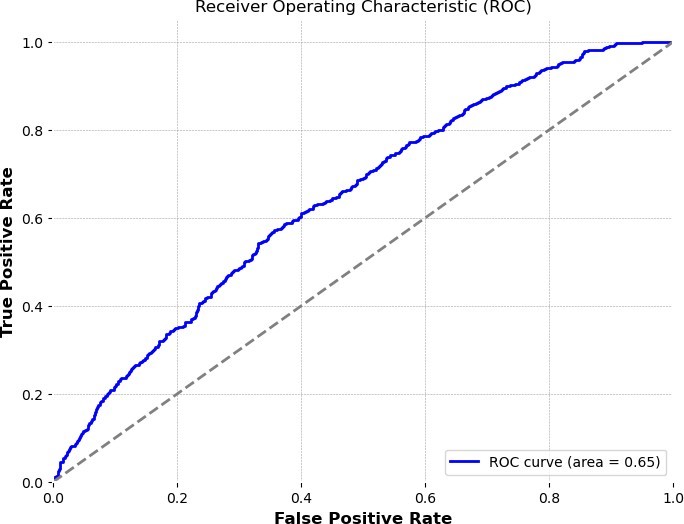
file\_path = 'C:/Users/HARISH/Downloads/loan\_data.csv' # Update path accordinglyif os.path.exists(file\_path):

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.

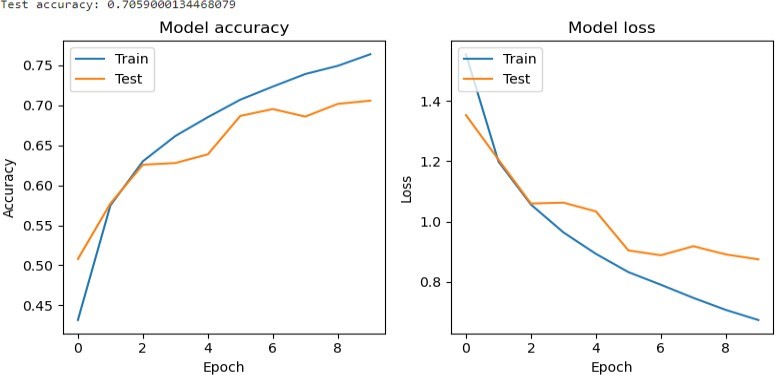
**PROGRAM:**

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()  COMPUTATIONAL  STATISTICS | 221501006 |

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:**



**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:**

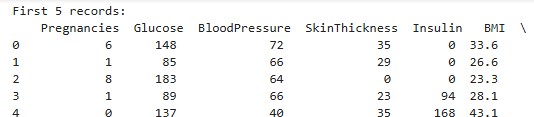
import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score url = https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

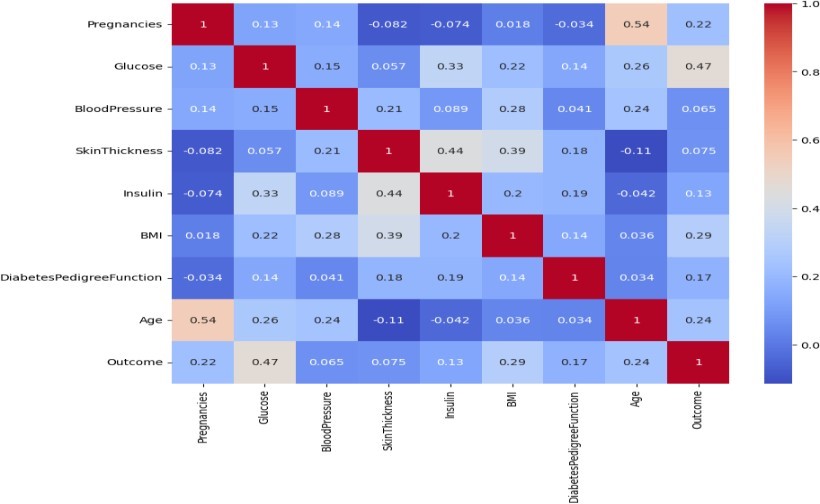
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

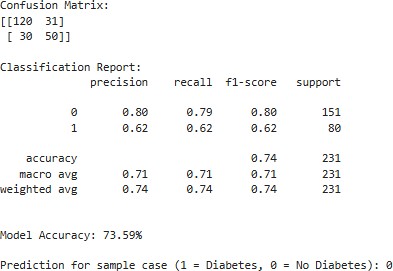
data = pd.read\_csv(url, header=None, names=columns) print("First 5 records:\n", data.head()) print("\nStatistical Summary:\n", data.describe()) print("\nDataset Info:\n") print(data.info()) sns.pairplot(data, hue='Outcome') plt.show() correlation\_matrix = data.corr() plt.figure(figsize=(10, 8)) sns.heatmap(correlation\_matrix, 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:**







**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 = {

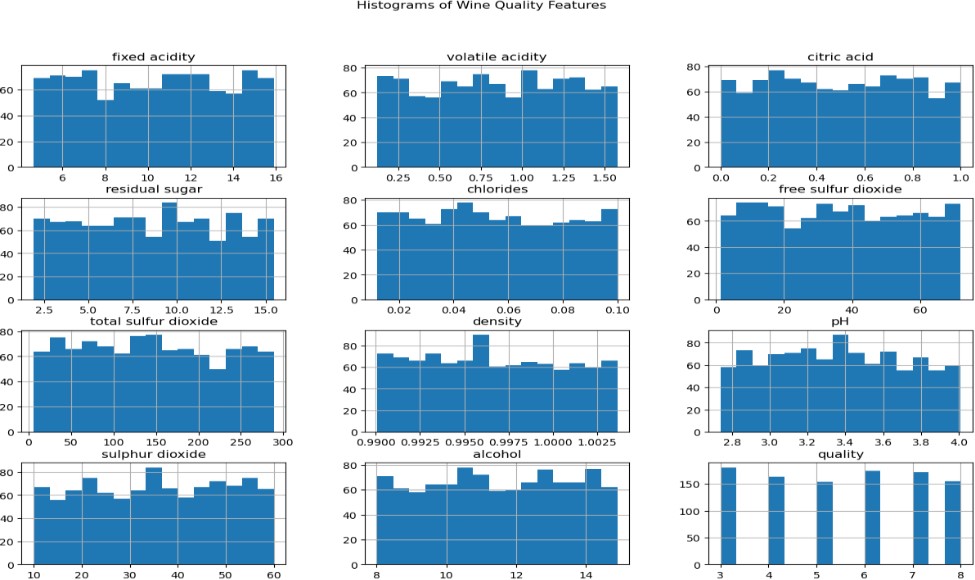
'fixed acidity': 7.4, 'volatile acidity': 0.7, 'citric acid': 0.0,

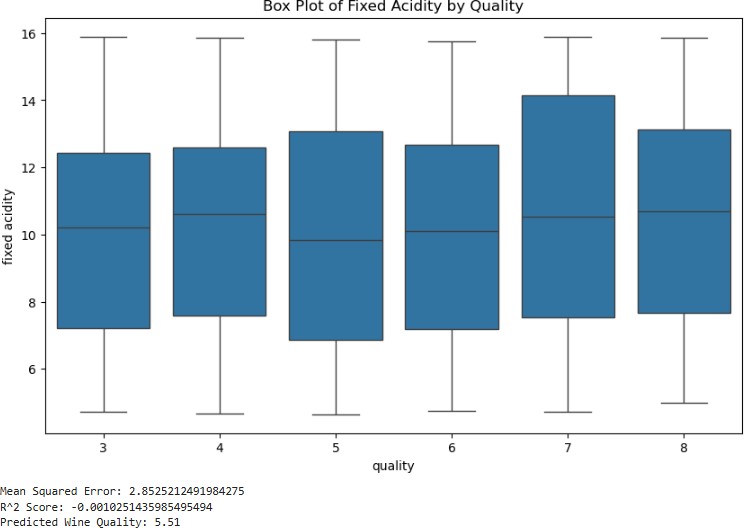
'residual sugar': 1.9, 'chlorides': 0.076, 'free sulfur dioxide': 11.0,

'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:**





**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()

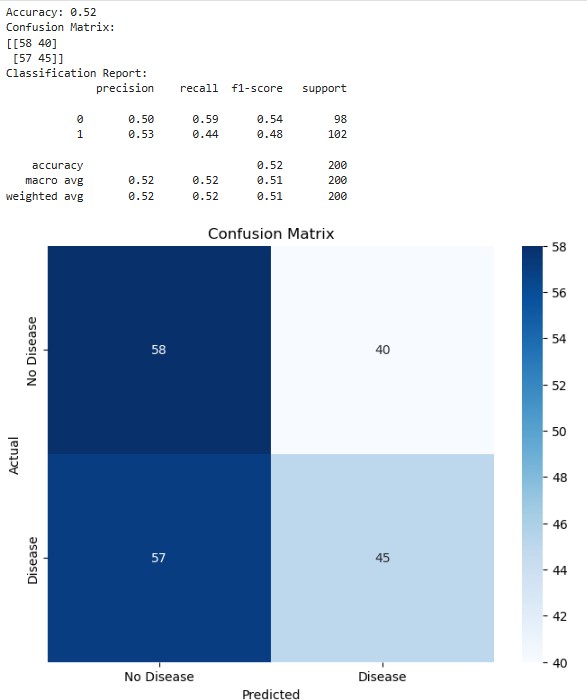
X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test) model = LogisticRegression() model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) conf\_matrix = confusion\_matrix(y\_test, y\_pred) class\_report = classification\_report(y\_test, 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:**



**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()

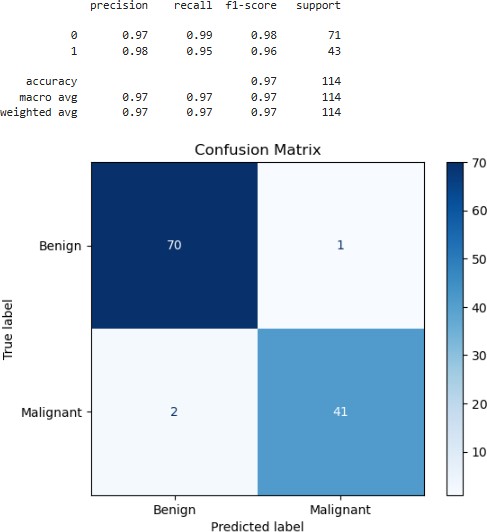
X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test) model = LogisticRegression() model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) conf\_matrix = confusion\_matrix(y\_test, y\_pred) class\_report = classification\_report(y\_test, 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'],

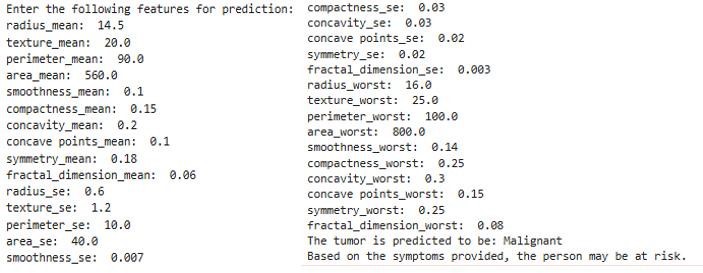
yticklabels=['No Disease', 'Disease'])

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:**





**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()

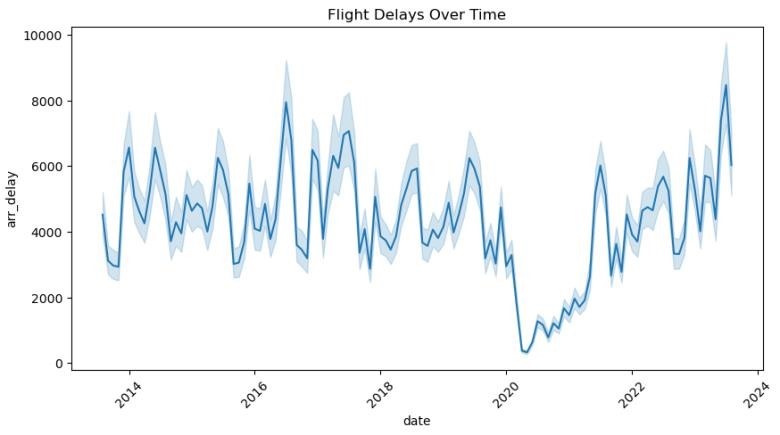
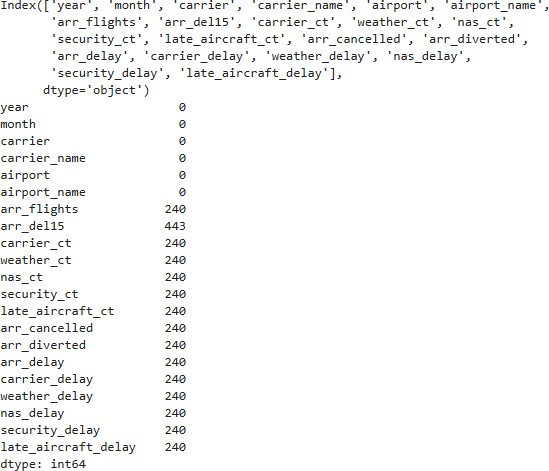
delay\_column = 'arr\_delay' # Using 'arr\_delay' for now if 'carrier\_delay' in df.columns and delay\_column in df.columns:

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()



**O**

**U**

**T**

**P**

**U**

**T**

**:**

**R**

**E**

**S**

**U**

**L**

**T**

**:**

T

h

u

s

,

t

h

e

p

r

o

g

r

a

m

f

o

r

p

r

e

d

i

c

t

i

n

g

f

l

i

g

h

t

d

e

l

a

y

s

i

s

e

x

e

c

u

t

e

d

s

u

c

c

e

s

s

f

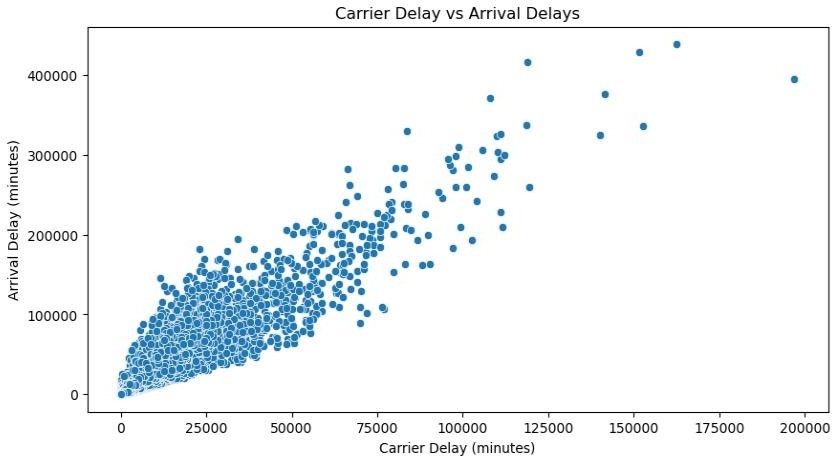
u

l

l

y

.



**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()

**O**

**U**

**T**

**P**

**U**

**T**

**:**

**R**

**E**

**S**

**U**

**L**

**T**

**:**

T

h

u

s

,

t

h

e

p

r

o

g

r

a

m

f

o

r

e

n

e

r

g

y

c

o

n

s

u

m

p

t

i

o

n

f

o

r

e

c

a

s

t

i

n

g

i

s

e

x

e

c

u

t

e

d

s

u

c

c

e

s

s

f

u

l

l

y

.

