# 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:**

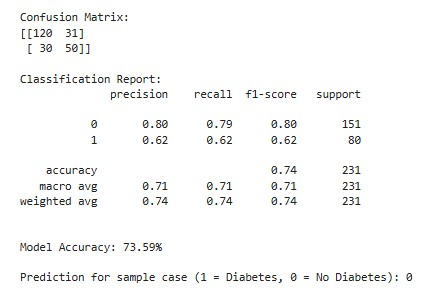
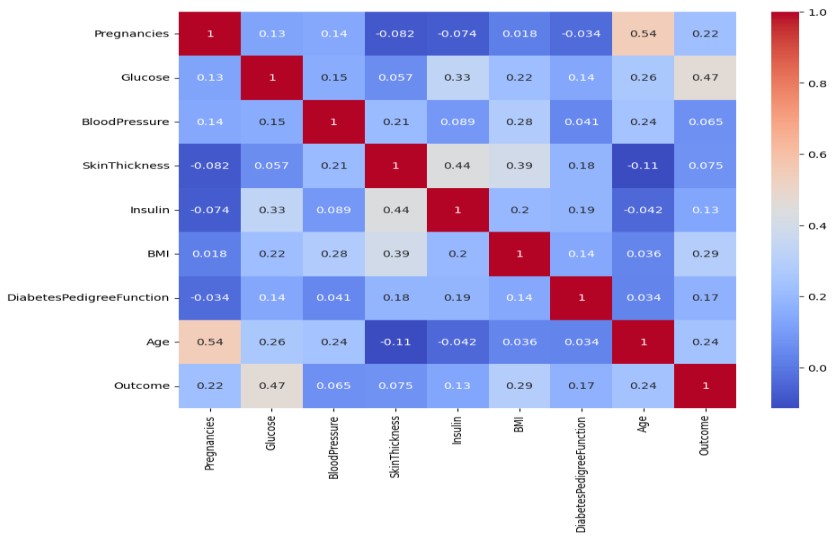
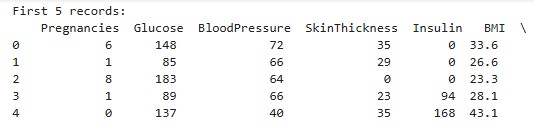
import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score url = https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.c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

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 = {

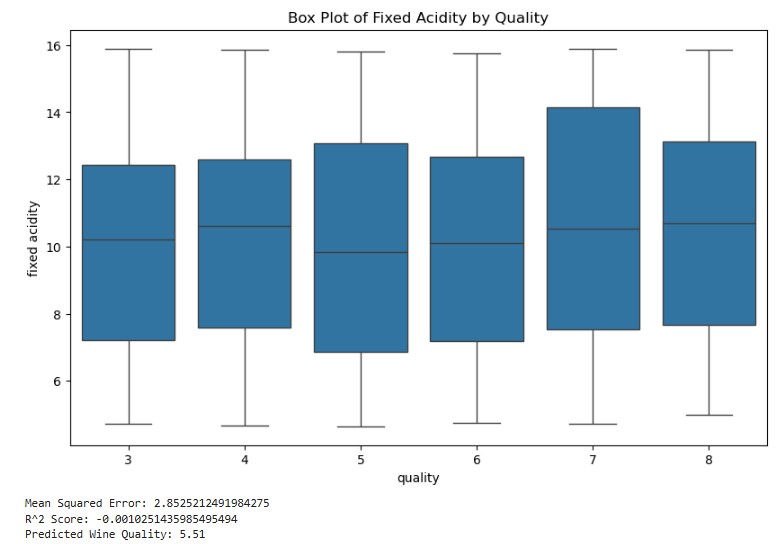
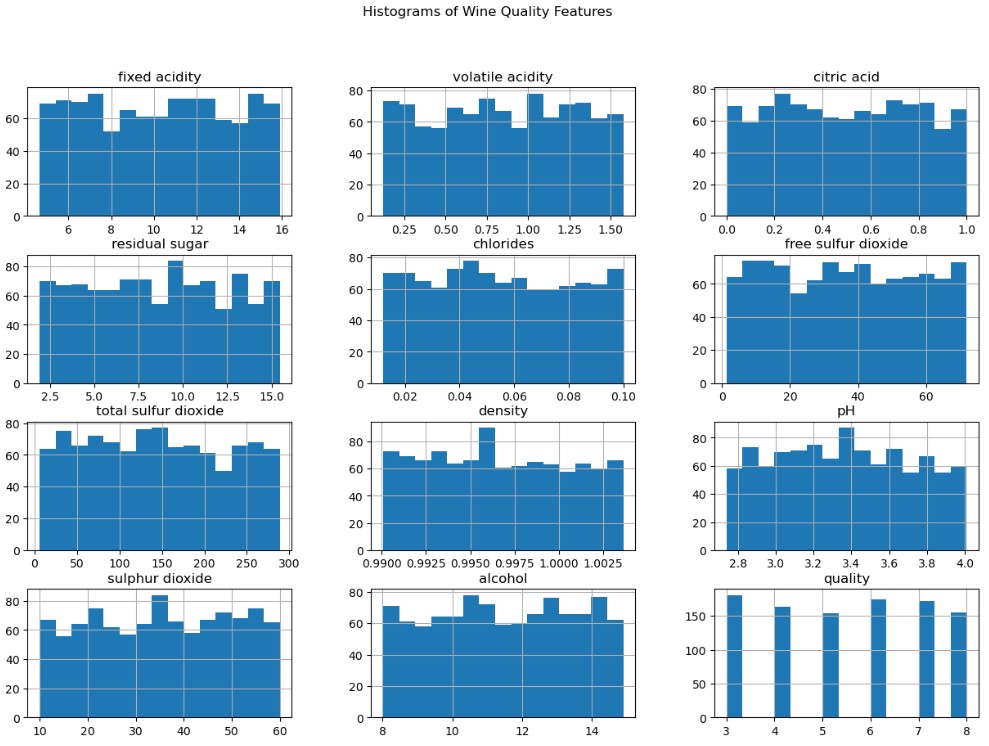
'fixed acidity': 7.4, 'volatile acidity': 0.7, 'citric acid': 0.0,

'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})

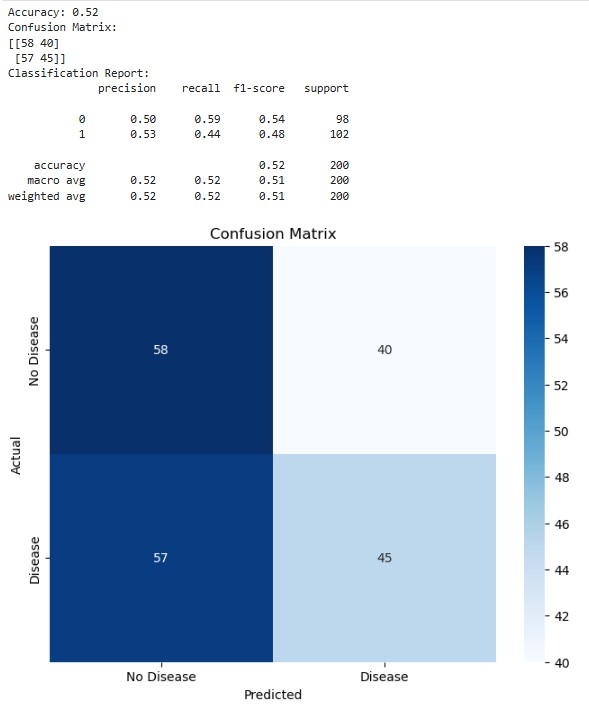
X = data.drop('target', axis=1) y = data['target']

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})

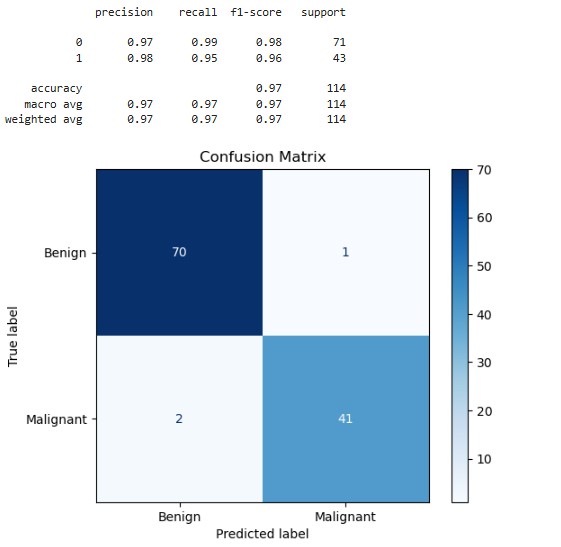
X = data.drop('target', axis=1) y = data['target']

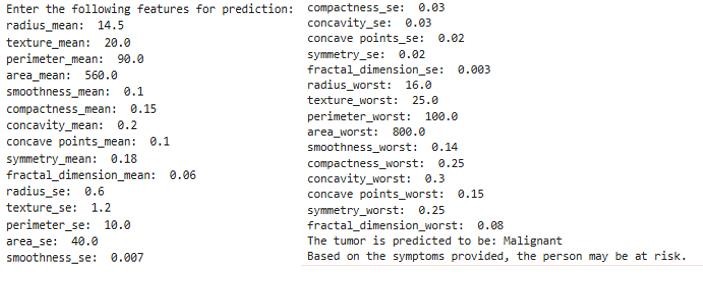
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) scaler = StandardScaler()

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/APPU/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:

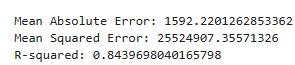
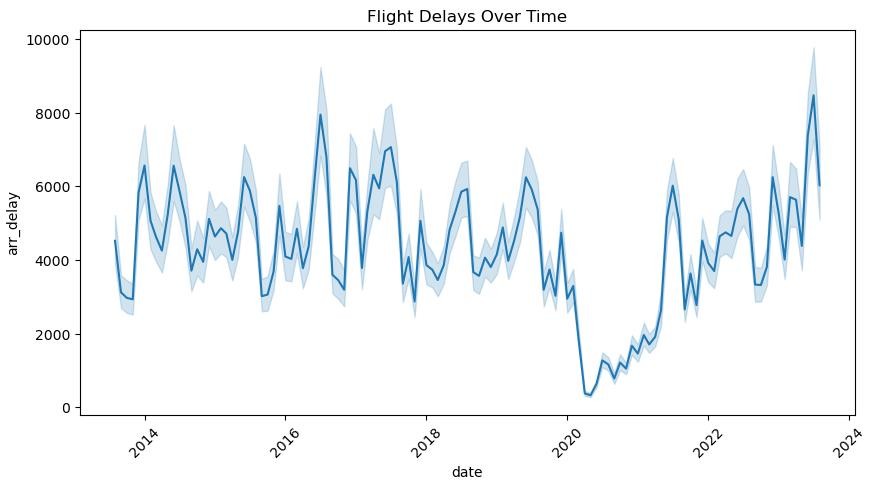
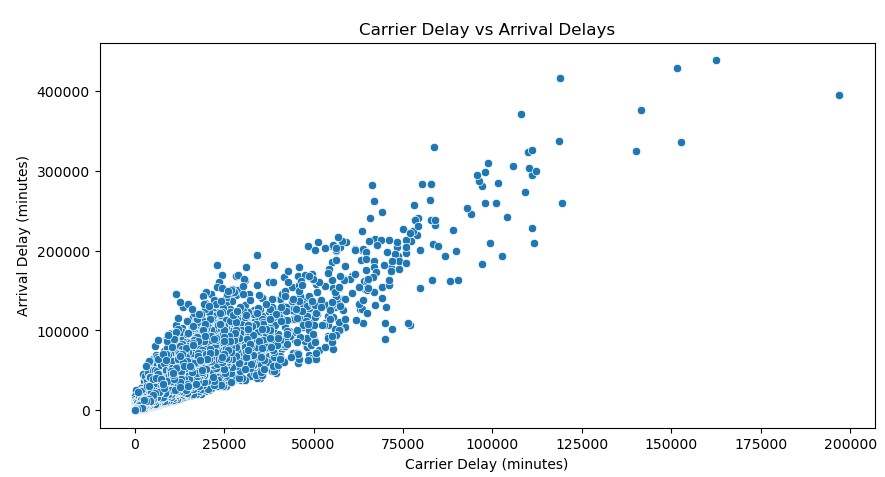
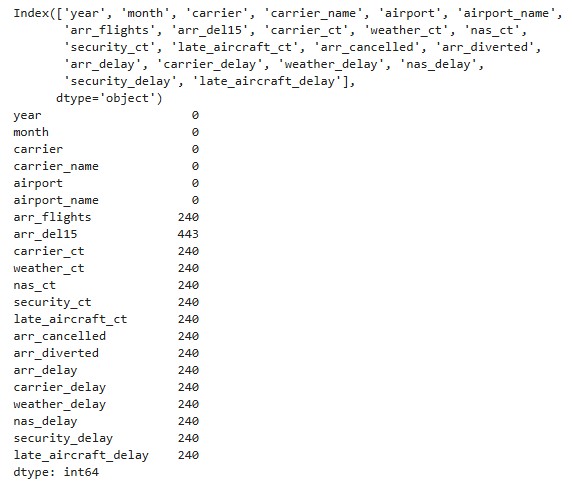
plt.figure(figsize=(10, 5)) sns.scatterplot(data=df, x='carrier\_delay', y=delay\_column) # Adjust as needed plt.title('Carrier Delay vs Arrival Delays') plt.xlabel('Carrier Delay (minutes)') plt.ylabel('Arrival Delay (minutes)') plt.show()

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:**



**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.

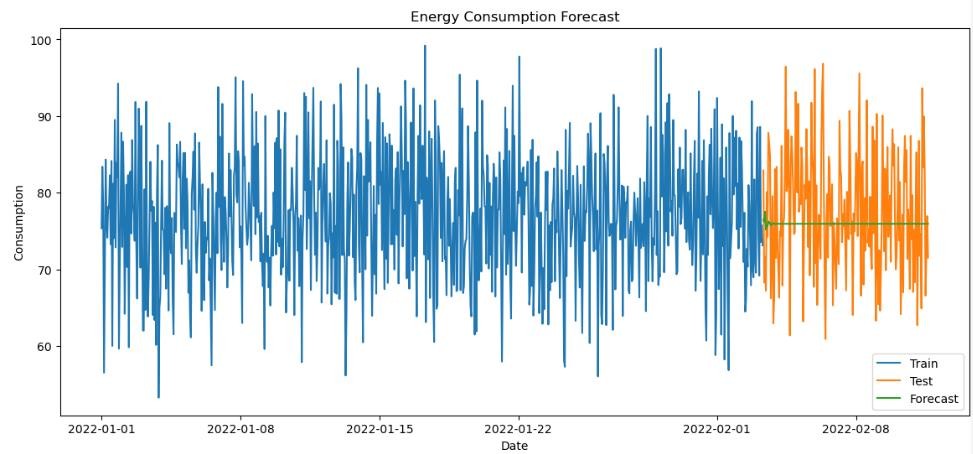
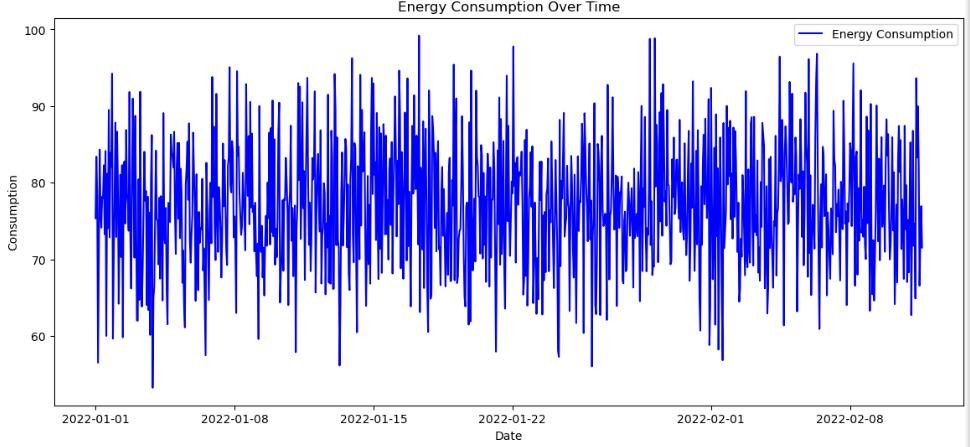
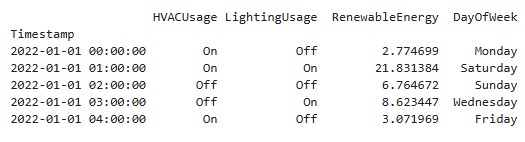
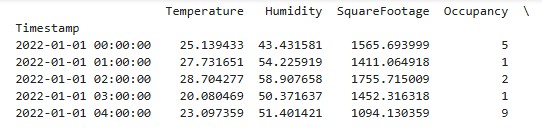
**PROGRAM:**

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/APPU/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:**



**RESULT:**

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