WEEK 1 - A DATE: 02-12-
2024

AIR QUALITY INDEX ANALYSIS

AIM:

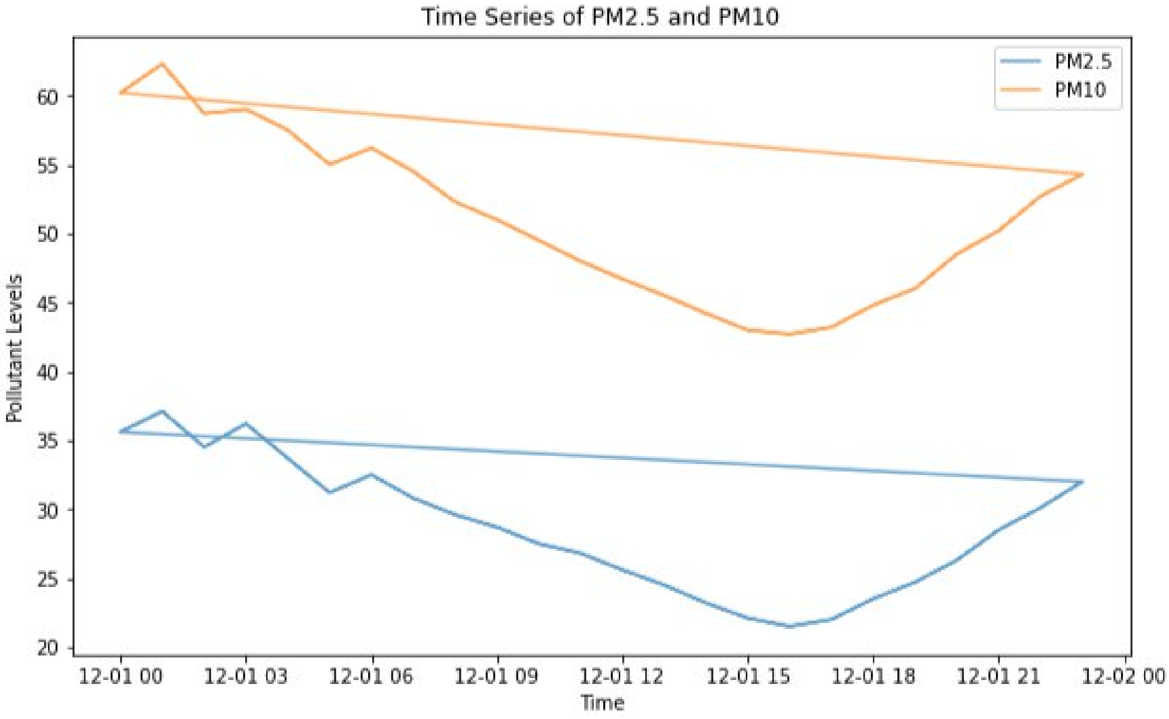
To write a program to Analysis the Air Quality Index.

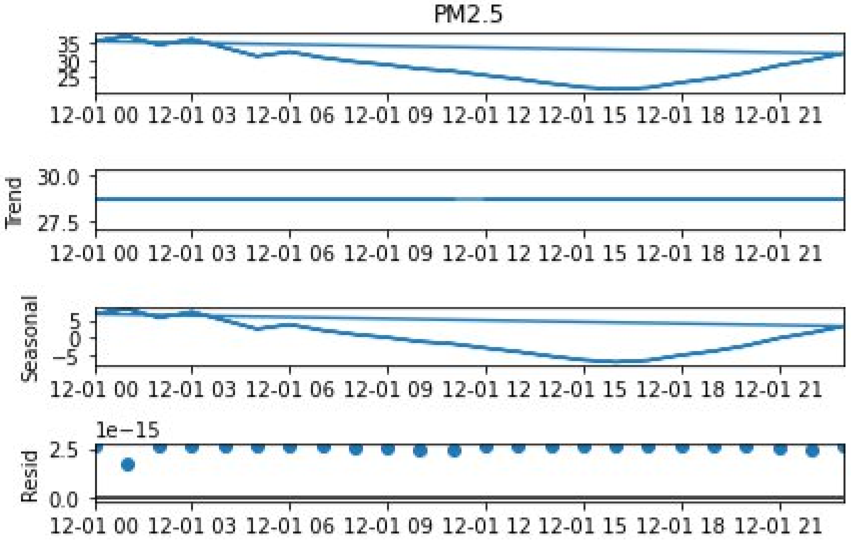
CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
data = {
   "date": ["2024-12-01"] * 24,
   "time": [f"{i:02}:00" for i in range(24)],
   "location": ["CityA"] * 24,
   "PM2.5": [35.6, 37.1, 34.5, 36.2, 33.7, 31.2, 32.5, 30.8, 29.6, 28.7, 27.5, 26.8, 25.6, 24.5, 23.2,
22.1, 21.5, 22.0, 23.5, 24.7, 26.3, 28.5, 30.1, 32.0],
   "PM10": [60.2, 62.3, 58.7, 59.0, 57.5, 55.0, 56.2, 54.5, 52.3, 51.0, 49.5, 48.0, 46.7, 45.5, 44.2,
43.0, 42.7, 43.2, 44.8, 46.0, 48.5, 50.2, 52.7, 54.3],
   "temperature": [22.5, 22.1, 21.9, 21.5,
                 21.2, 21.0, 21.1, 21.0, 22.0, 23.0, 24.2, 25.1, 26.0, 27.0, 28.0, 28.5, 27.5, 26.5, 25.2,
24.0, 23.0, 22.5, 22.1, 22.0],
   "wind_speed": [3.2, 3.0, 3.1, 3.3, 3.4, 3.5, 3.2, 3.1, 3.0, 2.9, 3.1, 3.2, 3.4, 3.5, 3.6, 3.8, 3.7, 3.5, 3.3,
3.2, 3.0, 2.8, 2.7, 2.5],
}
df = pd.DataFrame(data)
# Convert date and time columns into a single datetime column
df['datetime'] = pd.to_datetime(df['date'] + ' ' + df['time'])
df.set_index('datetime', inplace=True)
# Expand the dataset to simulate two full cycles
df_extended = pd.concat([df, df.copy()], ignore_index=False)
```

```
# Handle missing data (if any)
df_extended.fillna(df_extended.select_dtypes(include=np.number).mean(), inplace=True)
# Plot time-series data
plt.figure(figsize=(10, 6))
plt.plot(df_extended.index, df_extended['PM2.5'], label='PM2.5', alpha=0.7)
plt.plot(df_extended.index, df_extended['PM10'], label='PM10', alpha=0.7)
plt.title('Time Series of PM2.5 and PM10')
plt.xlabel('Time')
plt.ylabel('Pollutant Levels')
plt.legend()
plt.show()
# Decompose time series for seasonality analysis
result = seasonal_decompose(df_extended['PM2.5'], model='additive', period=24)  # Period is 24 hours
result.plot()
plt.show()
```

OUTPUT:





RESULT:

Thus, The program is successfully executed.

WEEK 1 - B DATE: 02-12-2024

E-COMMERCE SALES TREND

AIM:

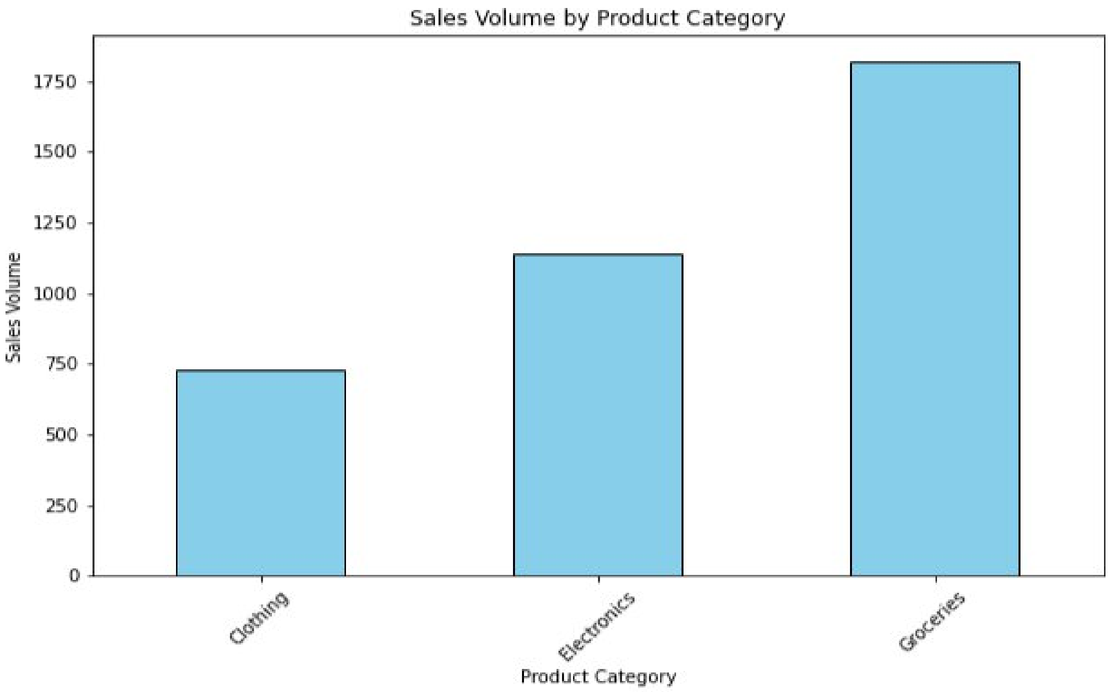
To write a program to E-Commerce Sales Trends Analysis.

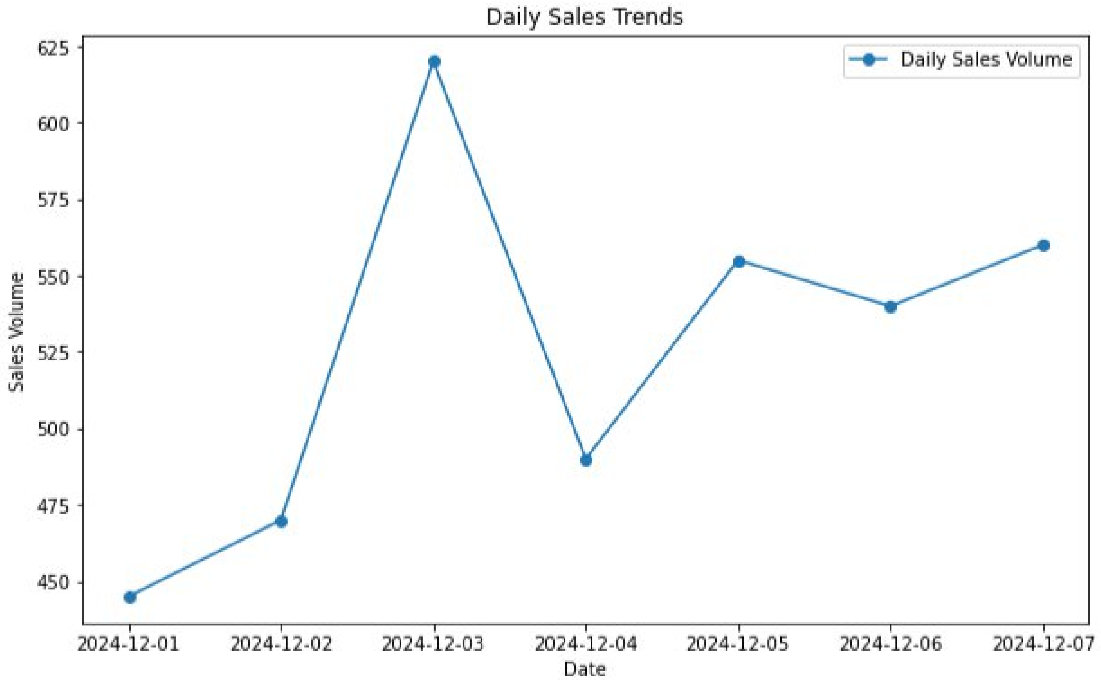
CODE:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

# Load the dataset into a DataFrame

sales_data = {

   "Product Category": ["Electronics", "Clothing", "Groceries", "Electronics", "Clothing", "Groceries",

                    "Electronics", "Clothing", "Groceries", "Electronics", "Clothing", "Groceries",

                    "Electronics", "Clothing", "Groceries", "Electronics", "Clothing", "Groceries",

                    "Electronics", "Clothing", "Groceries"],

   "Sales Volume": [120, 85, 240, 150, 100, 220, 200, 120, 300, 140, 90, 260, 175, 110, 270, 190, 100, 250, 160, 120, 280],

   "Price": [2500, 1500, 500, 2700, 1700, 520, 3000, 1800, 480, 2500, 1600, 450, 2800, 1750, 520, 2900, 1650, 500, 2650, 1800, 530],

   "Customer Demographics": ["Urban", "Rural", "Urban", "Suburban", "Urban", "Rural",

                         "Urban", "Urban", "Rural", "Suburban", "Rural", "Urban",

                         "Urban", "Suburban", "Rural", "Rural", "Urban", "Urban",

                         "Suburban", "Rural", "Urban"],

   "Time": ["2024-12-01", "2024-12-01", "2024-12-01", "2024-12-02", "2024-12-02", "2024-12-02",

          "2024-12-03", "2024-12-03", "2024-12-03", "2024-12-04", "2024-12-04", "2024-12-04",

          "2024-12-05", "2024-12-05", "2024-12-05", "2024-12-06", "2024-12-06", "2024-12-06",

          "2024-12-07", "2024-12-07", "2024-12-07"],

}
```

```
sales_df = pd.DataFrame(sales_data)

# Convert 'Time' column to datetime
sales_df['Time'] = pd.to_datetime(sales_df['Time'])

# Group data by product category and aggregate sales volume and price
category_sales = sales_df.groupby('Product Category')[['Sales Volume', 'Price']].sum()
# Plot sales volume by product category
plt.figure(figsize=(10, 6))
category_sales['Sales Volume'].plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Sales Volume by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Sales Volume')
plt.xticks(rotation=45)
plt.show()
# Group by time and visualize sales trends
time_sales = sales_df.groupby('Time')['Sales Volume'].sum()
plt.figure(figsize=(10, 6))
plt.plot(time_sales.index, time_sales, marker='o', label='Daily Sales Volume')
plt.title('Daily Sales Trends')
plt.xlabel('Date')
plt.ylabel('Sales Volume')
plt.legend()
plt.show()
```

OUTPUT:


Sales Volume by Product Category
Sales Volume
0
250
500
750
1000
1250
1500
1750
Clothing
Electronics
Groceries
Product Category


Daily Sales Trends
Daily Sales Volume
Sales Volume
450
475
500
525
550
575
600
625
2024-12-01
2024-12-02
2024-12-03
2024-12-04
2024-12-05
2024-12-06
2024-12-07
Date

RESULT:

Thus, The program is successfully executed.

WEEK 1 - C DATE: 02-12-2024

COVID - 19 CASE STUDY

AIM:

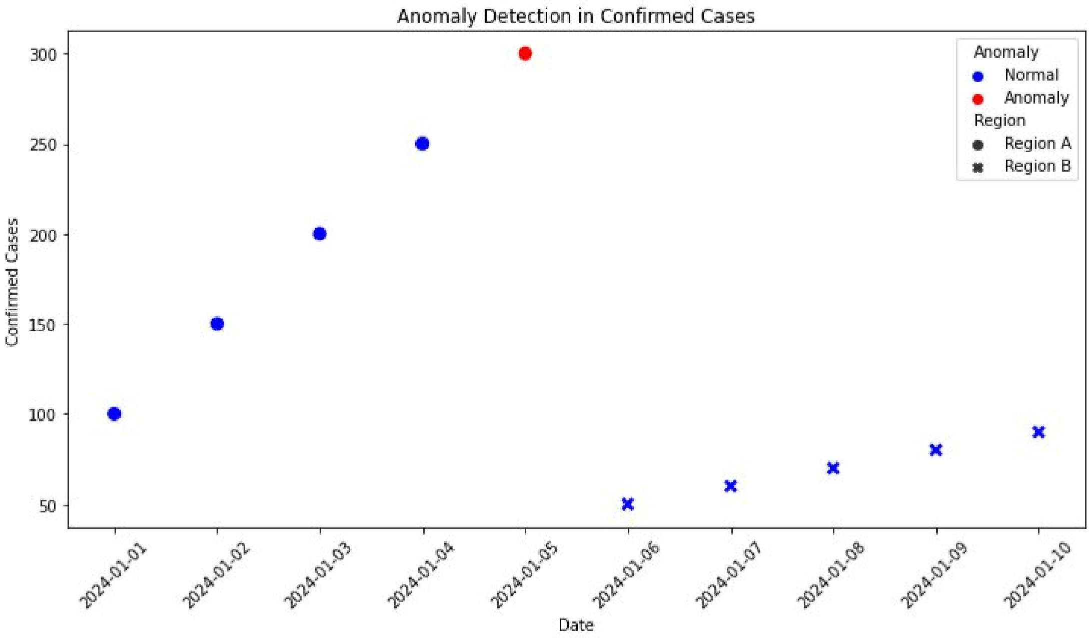
To write a program to Analysis the Covid-19 Case Study.

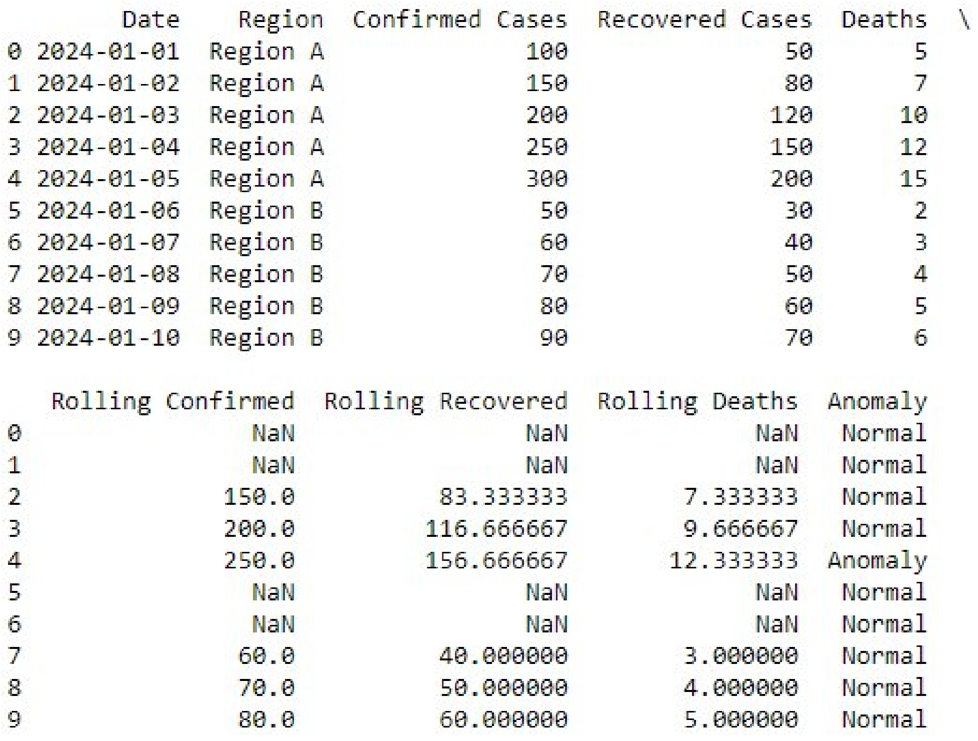
CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest
# Example COVID-19 dataset
data = {
    'Date': pd.date_range(start='2024-01-01', periods=10, freq='D'),
    'Region': ['Region A']*5 + ['Region B']*5,
    'Confirmed Cases': [100, 150, 200, 250, 300, 50, 60, 70, 80, 90],
    'Recovered Cases': [50, 80, 120, 150, 200, 30, 40, 50, 60, 70],
    'Deaths': [5, 7, 10, 12, 15, 2, 3, 4, 5, 6]
}
# Convert dictionary into a pandas DataFrame
df = pd.DataFrame(data)
# Calculate rolling averages (3-day window for example)
df['Rolling Confirmed'] = df.groupby('Region')['Confirmed Cases'].rolling(window=3).mean().reset_index(0, drop=True)
df['Rolling Recovered'] = df.groupby('Region')['Recovered Cases'].rolling(window=3).mean().reset_index(0, drop=True)
df['Rolling Deaths'] = df.groupby('Region')['Deaths'].rolling(window=3).mean().reset_index(0, drop=True)
```

```
# Prepare data for anomaly detection
features = ['Confirmed Cases', 'Recovered Cases', 'Deaths']  # Using raw features for simplicity
df_for_model = df[features]
iso_forest = IsolationForest(contamination=0.1, random_state=42)  # Adjust contamination as needed
df['Anomaly'] = iso_forest.fit_predict(df_for_model)
# Map anomaly values to more intuitive labels
df['Anomaly'] = df['Anomaly'].map({1: 'Normal', -1: 'Anomaly'})
# Visualize the results
plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=df,
    x='Date',
    y='Confirmed Cases',
    hue='Anomaly',
    style='Region',
    palette={'Normal': 'blue', 'Anomaly': 'red'},
    s=100
)
plt.title('Anomaly Detection in Confirmed Cases')
plt.xticks(rotation=45)
plt.show()
# Print the DataFrame to verify anomalies
print(df)
```

OUTPUT:





```
        Date    Region  Confirmed Cases  Recovered Cases  Deaths  \
0 2024-01-01  Region A              100               50       5
1 2024-01-02  Region A              150               80       7
2 2024-01-03  Region A              200              120      10
3 2024-01-04  Region A              250              150      12
4 2024-01-05  Region A              300              200      15
5 2024-01-06  Region B               50               30       2
6 2024-01-07  Region B               60               40       3
7 2024-01-08  Region B               70               50       4
8 2024-01-09  Region B               80               60       5
9 2024-01-10  Region B               90               70       6

   Rolling Confirmed  Rolling Recovered  Rolling Deaths  Anomaly
0                NaN                NaN             NaN   Normal
1                NaN                NaN             NaN   Normal
2              150.0          83.333333        7.333333   Normal
3              200.0         116.666667        9.666667   Normal
4              250.0         156.666667       12.333333  Anomaly
5                NaN                NaN             NaN   Normal
6                NaN                NaN             NaN   Normal
7               60.0          40.000000        3.000000   Normal
8               70.0          50.000000        4.000000   Normal
9               80.0          60.000000        5.000000   Normal
```

RESULT:

Thus, The program is successfully executed.

WEEK 2 - A DATE: 10-12-2024

SPAM (OR) NOT SPAM

AIM:

To write a program to classify the email as spam or not spam using KNN neighbour algorithm.

CODE:

```
#Import the needed libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data=pd.read_csv('dataset.csv')
print(data.head())
data.dropna(inplace =True)
data['text']=data['text'].str.lower()
X=data['text']
Y=data['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
new_email = ["Congratulations! You've won a $1000 gift card. Click here to claim."]
new_email_tfidf = vectorizer.transform(new_email)
prediction = model.predict(new_email_tfidf)
print("\nPrediction for new email:", "Scam" if prediction[0] == 1 else "Not Scam")
```

OUTPUT:

```
Cross-Validation Accuracy: 0.87
Test Accuracy: 0.60

Classification Report:
              precision    recall     f1-score   support

          0       1.00      0.50       0.67           4
          1       0.33      1.00       0.50           1
 accuracy                             0.60           5
 macro avg   0.67      0.75        0.58            5
weighted avg    0.87    0.60         0.63            5
Model saved as 'spam_classifier_model.pkl'
Prediction for test email (0 = Not Spam, 1 = Spam): 1
```

RESULT:

Thus, The program is successfully executed.

WEEK 2 – B DATE: 10-12-
2024

PIZZA LIKING PREDICTION

AIM:

To write a program to Pizza Liking Prediction using KNN neighbour algorithm.

CODE:

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report
data={ "age":[15,25,35,45,55,22,40,28,18,60],
   "likes_chease":[1,1,0,1,0,1,0,1,1,0],
   "Fast_food":[0,0,1,0,1,0,1,0,0,1],
   "Salary":[50000,100000,10000,80000,8000,45000,11000,120000,130000,13000],
   "location":["Urban","Urban","Rural","Urban","Rural","Urban","Rural","Urban","Urban","Rural"],
"Awareness":["know","know","Unknow","know","Unknow","know","Unknow","know","know","Unknow"],
   "working":["IT","IT","Farmer","Engineer","Driver","Pilot","Painter","IT","Artisr","Player"],
    "Health":["Good","Bad","Bad","Good","Bad","Good","Bad","Good","Good","Bad"],
   "likes_pizza":["yes","yes","no","yes","no","yes","no","yes","yes","no"]}
df=pd.DataFrame(data)
x=df[["age","likes_chease","Fast_Food","Salary","location","working","Health","Awareness"]]
y=df["likes_pizza"]
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.3, random_state=42)
model=RandomForestClassifier(random_state=42)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print(df)
print("Accuracy:",accuracy_score(y_test,y_pred))
print("\n Classification Report:\n",classification_report(y_test,y_pred))
example_person={"age":30,"likes_cheese":1,"vegetarian":0,"like_spicy":1}
example_df=pd.DataFrame([example_person])
prediction=model.predict(example_df)
```

```
print(f"\nPrediction: The person {'likes' if prediction[0] == 'yes' else 'does not like'} pizza.")
```

OUTPUT:

```
   age  likes_chease  Fast_food  Salary location Awareness   working
Health  \
0   15             1          0   50000    Urban      know        IT
Good
1   25             1          0  100000    Urban      know        IT
Bad
2   35             0          1   10000    Rural    Unknow    Farmer
Bad
3   45             1          0   80000    Urban      know  Engineer
Good
4   55             0          1    8000    Rural    Unknow    Driver
Bad
5   22             1          0   45000    Urban      know     Pilot
Good
6   40             0          1   11000    Rural    Unknow   Painter
Bad
7   28             1          0  120000    Urban      know        IT
Good
8   18             1          0  130000    Urban      know    Artisr
Good
9   60             0          1   13000    Rural    Unknow    Player
Bad
likes_pizza
0         yes
1         yes
2          no
3         yes
4          no
5         yes
6          no
7         yes
8         yes
9          no
Accuracy: 1.0

 Classification Report:
               precision    recall  f1-score   support

         yes       1.00      1.00      1.00         3

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


Prediction: The person likes pizza.
```

RESULT:

Thus, The program is successfully executed.

WEEK 2 – C DATE: 10-12-2024

MOVIE GENRE PREDICTION

AIM:

To write a program to Analysis the Movie Genre Prediction.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
# Load the dataset
df = pd.read_csv("movie.csv")
# Encode categorical features
label_encoder = LabelEncoder()
df['language'] = label_encoder.fit_transform(df['language'])
df['genre'] = label_encoder.fit_transform(df['genre'])
df['director'] = label_encoder.fit_transform(df['director'])
# Remove rare classes with fewer than 2 samples
class_counts = df['genre'].value_counts()
rare_classes = class_counts[class_counts < 2].index
df = df[~df['genre'].isin(rare_classes)]
# Features and target
X = df[['duration', 'language', 'average_rating', 'number_of_reviews', 'year', 'budget', 'revenue']]
y = df['genre']
# Check class distribution
print("Class distribution in the target variable:")
print(df['genre'].value_counts())
# Scale features
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split the data with stratification
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42,
stratify=y)
# Train a classifier with class weights to handle imbalance
clf = RandomForestClassifier(random_state=42, class_weight="balanced")
clf.fit(X_train, y_train)
# Predictions
y_pred = clf.predict(X_test)
# Evaluate using classification report with zero_division parameter
print("Classification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

OUTPUT:

```
Class distribution in the target variable:
1    3
0    2
Name: genre, dtype: int64
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

RESULT:

Thus, The program is successfully executed.

WEEK 2 – D DATE: 10-12-2024

SPORTS PERFORMANCE ANALYSIS

AIM:

To write a program to Analysis the Sports Performance.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
# Load the dataset
df = pd.read_csv("sports.csv")

# Features and target
X = df[['accuracy', 'speed', 'stamina', 'age']].copy()  # Ensure X is a copy
y = df['performance'].copy()  # Ensure y is a copy
# Include outliers
outlier = pd.DataFrame([[200, 15, 150, 30]], columns=X.columns)
X = pd.concat([X, outlier], ignore_index=True) # Add a corresponding target value for the outlier using pd.concat
y = pd.concat([y, pd.Series(['excellent'])], ignore_index=True)
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train k-NN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
# Predictions
y_pred = knn.predict(X_test)
```

```
# Evaluate with zero_division set to 1 to avoid warnings
print(classification_report(y_test, y_pred, zero_division=1))
```

OUTPUT:

```
              precision    recall  f1-score   support

     Average       0.00      1.00      0.00       0.0
   Excellent       1.00      0.00      0.00       1.0
        Good       1.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.67      0.33      0.00       2.0
weighted avg       1.00      0.00      0.00       2.0
```

RESULT:

Thus, The program is successfully executed.

WEEK 3 – A DATE: 19-12-2024

FUEL AMOUNT PREDICTION

AIM:

To write a program to Analysis the fuel amount prediction using linear regression.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
df=pd.read_csv("Fuel_data.csv")
print("Dataset Preview:")
print(df.head())
X = df[['distance']]  # Feature: distance traveled
y = df['fuel']  # Target: fuel consumed
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)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print("Model Coefficients:")
print(f"Intercept: {model.intercept_}")
print(f"Slope: {model.coef_[0]}")
new_distance = pd.DataFrame({'distance': [150]})  # Create DataFrame for new input
predicted_fuel = model.predict(new_distance)
print(f"Predicted fuel for {new_distance.iloc[0, 0]} km: {predicted_fuel[0]:.2f} liters")
```

OUTPUT:

```
Dataset Preview:
   distance  fuel
0        10   0.8
1        20   1.6
2        30   2.4
3        40   3.2
4        50   4.0
Mean Squared Error: 9.860761315262648e-32
Model Coefficients:
Intercept: 8.881784197001252e-16
Slope: 0.07999999999999999
Predicted fuel for 150 km: 12.00 liters
```

RESULT:

Thus, The program is successfully executed.

WEEK 3 – B DATE: 19-12-
2024

SALARY PREDICTION

AIM:

To write a program to Analysis the fuel amount prediction using linear regression.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
df = pd.read_csv("salary_data.csv")
df = pd.get_dummies(df, columns=['industry', 'location'], drop_first=True)
X = df[['years_experience', 'qualification'] + [col for col in df.columns if 'industry_' in col or 'location_' in col]]
y = df['salary']
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)
print("Dataset Preview:")
print(df.head())
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
new_data = pd.DataFrame({  'years_experience': [5],
   'qualification': [3],
   'industry_IT': [1],
   'industry_Marketing': [0],  # Ensure this matches training columns
   'location_CityB': [1],
   'location_CityC': [0],
}, columns=X_train.columns)  # Ensure exact column alignment
```

predicted_salary = model.predict(new_data)

print(f"Predicted salary for the new profile: {predicted_salary[0]:.2f}")

OUTPUT:

```
Dataset Preview:
Years experience  qualification  salary industry IT industry Marketing
0                 2              2   50000           1
0
1                 5              3   80000           0
0
2                 3              2   55000           1
0
3                 7              4   90000           0
0
4                 1              1   45000           0
1

   location_CityB  location_CityC
0               0               0
1               1               0
2               0               0
3               1               0
4               0               1
Mean Squared Error: 4674945.215485762
Predicted salary for the new profile: 72162.16
```

RESULT:

Thus, The program is successfully executed.

WEEK 4 – A DATE: 24-01-2025

HOUSE PRICE PREDICTION

AIM:

To write a program to Analysis the House Price Prediction using L-R regulation.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
data = pd.read_csv("house_prices.csv")
X = data[['size', 'bedrooms', 'age']]
y = data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = Ridge(alpha=1.0)  # Alpha controls the regularization strength
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test,y_pred))
```

OUTPUT:

```
Mean Squared Error: 3325492.320725987
```

RESULT:

Thus, The program is successfully executed.

WEEK 4 – B DATE: 24-01-2025

CROP YIELD PREDICTION

AIM:

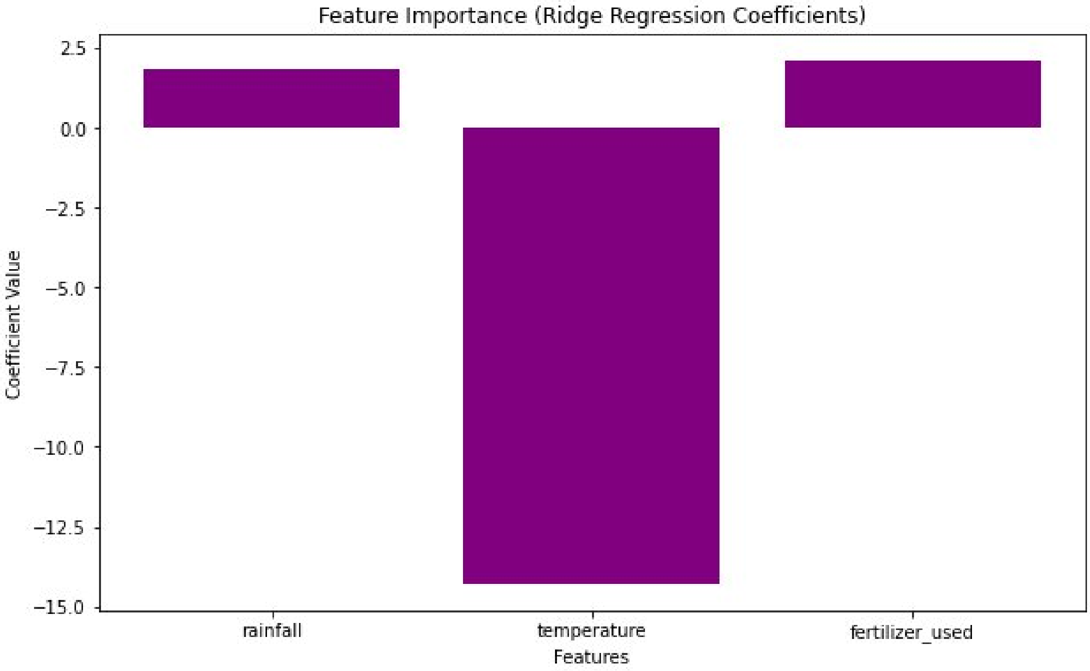
To write a program to Analysis the Crop Yield Prediction using L-R regulation.

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
data = pd.read_csv("crop_yield.csv")
X = data[['rainfall', 'temperature', 'fertilizer_used']]
y = data['yield']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
coefficients = model.coef_
plt.figure(figsize=(10, 6))
plt.bar(X.columns, coefficients, color='purple')
plt.title('Feature Importance (Ridge Regression Coefficients)')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.show()
```

OUTPUT:

```
Mean Squared Error: 2498.780533491533
```



RESULT:

Thus, The program is successfully executed.

WEEK 4 – C DATE: 24-01-2025

ENERGY EFFICIENCY PREDICTION

AIM:

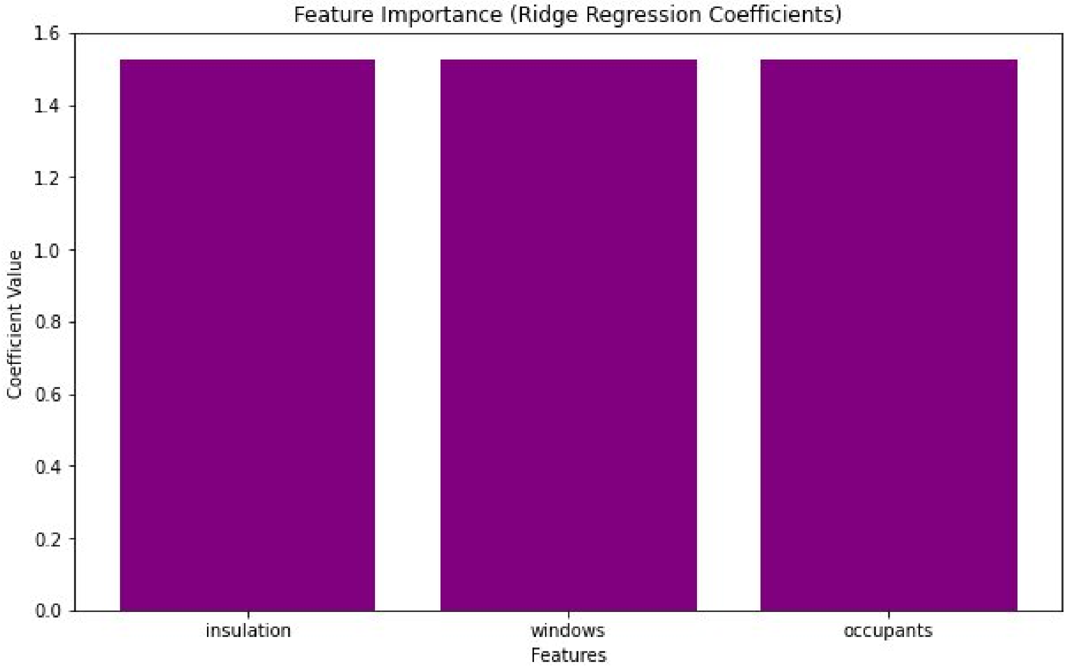
To write a program to Analysis the Energy Efficiency Prediction using L-R regulation.

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
data = pd.read_csv("energy_efficiency.csv")
X = data[['insulation', 'windows', 'occupants']]
y = data['efficiency']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test,y_pred))
coefficients = model.coef_
plt.figure(figsize=(10, 6))
plt.bar(X.columns, coefficients, color='purple')
plt.title('Feature Importance (Ridge Regression Coefficients)')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.show()
```

OUTPUT:

```
Mean Squared Error: 0.03251814028486949
```



RESULT:

Thus, The program is successfully executed.

WEEK 5 – A DATE: 24-01-2025

DIABETES CLASSIFICATION

AIM:

To write a program to Diabetes Classification using L-R regulation.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
data = pd.read_csv("diabetes.csv")
X = data[['age', 'bmi', 'blood_pressure', 'glucose']]
y = data['diabetes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,y_pred))
```

OUTPUT:

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1
```

RESULT:

Thus, The program is successfully executed.

WEEK 5 – B DATE: 24-01-
2025

CREDIT CARD DEFAULT PREDICTION

AIM:

To write a program to Analysis the Credit Card Default Prediction using L-R regulation.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
data = pd.read_csv("credit_card_default.csv")
X = data[['income', 'credit_score', 'age', 'loan_amount']]
y = data['default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,y_pred))
```

OUTPUT:

```
Accuracy: 0.0
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       0.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0
```

RESULT:

Thus, The program is successfully executed.

WEEK 5 – C DATE: 24-01-
2025

HEART DISEASE CLASSIFICATION

AIM:

To write a program to Analysis the Heart Disease Classification using L-R regulation.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
data = pd.read_csv("heart_disease.csv")
X = data[['age', 'cholesterol', 'blood_pressure', 'exercise']]
y = data['heart_disease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,y_pred))
```

OUTPUT:

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1
```

RESULT:

Thus, The program is successfully executed.

Week-6A

Date: 31/01/2025

PREDICTIVE ANALYTICS FOR HOSPITALS

DISEASE OUTBREAK PREDICATION

AIM:

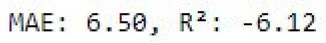
To write a program to Disease outbreak predication

PROGRAM CODE:

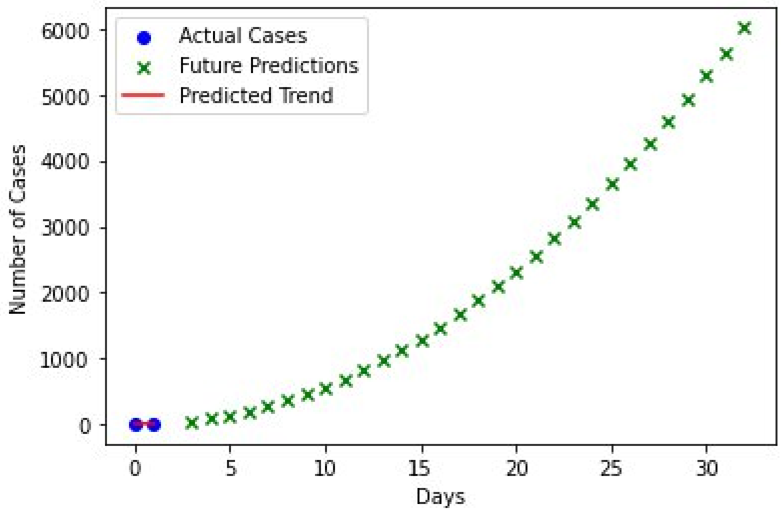
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, r2_score

# Load data
df = pd.read_csv('disease_outbreak_data.csv')
df['date'] = pd.to_datetime(df['date'])
df['day_number'] = (df['date'] - df['date'].min()).dt.days

X = df[['day_number']]
y = df['number_of_cases']

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

# Use Polynomial Regression (degree=2) for better prediction
model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f'MAE: {mae:.2f}, R²: {r2:.2f}')

# Forecast future cases (next 30 days)
future_days = pd.DataFrame(np.arange(X.max().values[0] + 1, X.max().values[0] + 31),
columns=['day_number'])
future_predictions = model.predict(future_days)

# Sort X_test for correct plotting
X_test_sorted, y_pred_sorted = zip(*sorted(zip(X_test.values.flatten(), y_pred)))

# Plot results
plt.scatter(X_test, y_test, color='blue', label='Actual Cases')
plt.scatter(future_days, future_predictions, color='green', marker='x', label='Future Predictions')
plt.plot(X_test_sorted, y_pred_sorted, color='red', label='Predicted Trend')
plt.xlabel("Days")
plt.ylabel("Number of Cases")
plt.legend()
plt.show()
```

OUTPUT:



```
MAE: 6.50, R²: -6.12
```



RESULT:

Thus the program executed successfully.

Week-6B

Date: 31/01/2025

BED OCCUPANCY DATA

AIM:

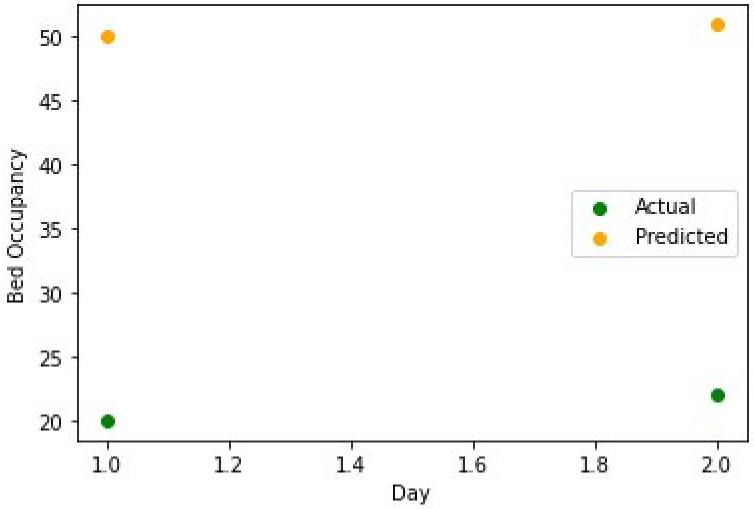
To write a program to predictive analytics bed occupancy data

PROGRAM CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Load data
data = pd.read_csv('bed_occupancy_data.csv')

# Ensure 'day' column is numerical
X = data[['day']].values.ravel()  # Convert to 1D array
y = data['bed_occupancy']

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

# Reshape X for model input
X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

# Model training
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Prediction
predicted_occupancy = rf_model.predict(X_test)

# Plot
plt.scatter(X_test, y_test, color='green', label='Actual')
plt.scatter(X_test, predicted_occupancy, color='orange', label='Predicted')  # Use scatter instead
of plot
```

```
plt.xlabel('Day')
plt.ylabel('Bed Occupancy')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus the program executed successfully.

Week-6C

Date: 31/01/2025

MEDICATION EFFECTIVENESS DATA

AIM:

To write a program to predictive analytics medication effectiveness data

PROGRAM CODE:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
# Load data
data = pd.read_csv('medication_effectiveness_data.csv')
# Encode categorical variables
le = LabelEncoder()
data['medication'] = le.fit_transform(data['medication'])
X = data[['medication', 'duration']]
y = data['effectiveness_score'] > 7 # Binary classification: Effective (True/False)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
# Predictions
y_pred = clf.predict(X_test)
print("Predictions (Effective/Not Effective):", y_pred)
```

OUTPUT:

```
Predictions (Effective/Not Effective): [ True]
```

RESULT:

Thus the program executed successfully.

Week: 7. A **LOAN APPROVAL CLASSIFICATION** Date:

Aim:

To classify loan applications as approved or rejected based on applicant features using Support Vector Machine (SVM).

Program Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
LabelEncoder from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
classification_report # Load the dataset
dataset =
pd.read_csv('loan_approval_data.csv
') # Encode categorical features
manually label_encoder =
LabelEncoder()
dataset['Employment_Status']  =
label_encoder.fit_transform(dataset['Employment_Status'])  #
Preprocess the data
X =
dataset.drop('Loan_Status',
axis=1)  y =
dataset['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)
# Train the SVM model
model =
SVC(kernel='linear')
model.fit(X_train, y_train)
# Predict and evaluate
```

```
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0
Classification Report:
            precision recall  f1-score  support


         0     1.00    1.00    1.00      2
         1     1.00    1.00    1.00      4


  accuracy                     1.0       6
                               0
 macro avg    1.00    1.00    1.0        6
                              0
 weighted      1.00    1.00    1.0        6
     avg                       0
```

Result:

Thus, the output was executed successfully.

Week: 7. B **CREDIT WORTHNESS ASSESSMENT** Date:

Aim:

To assess the creditworthiness of individuals by classifying them as creditworthy or not using SVM.

Program Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
LabelEncoder from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
classification_report # Load the dataset
dataset =
pd.read_csv('credit_worthiness_data.csv
') # Encode categorical features
manually label_encoder =
LabelEncoder()
dataset['Employment_Status'] =
label_encoder.fit_transform(dataset['Employment_Status']) #
Preprocess the data
X = dataset.drop('Credit_Worthy',
axis=1) y =
dataset['Credit_Worthy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)
# Train the SVM model
model =
SVC(kernel='linear')
model.fit(X_train, y_train)
# Predict and evaluate
```

```
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0

Classification Report:

            precision recall f1-score  support


        0     1.00    1.00    1.00      4

        1     1.00    1.00    1.00      3


  accuracy                   1.0        7
                             0
 macro avg    1.00    1.00    1.0        7
                              0
 weighted      1.00    1.00    1.0        7
     avg                       0
```

Result:

Thus, the output was executed successfully.

Week: 7. C **FAKE NEWS DETECTION** Date:

Aim:

To classify news articles as real or fake using SVM based on textual data features.

Program Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
TfidfVectorizer from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
classification_report # Load the dataset
dataset =
pd.read_csv('fake_news_data.csv')
# Preprocess the data
X =
dataset['text']
y =
dataset['label'
]
vectorizer =
TfidfVectorizer() X =
vectorizer.fit_transform(
X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # Train the SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 0.6666666666666666

Classification Report:

            precision recall f1-score  support


        0     0.75    0.75    0.75      4

        1     0.50    0.50    0.50      2


  accuracy                     0.6       6
                               7
 macro avg    0.62    0.62    0.6       6
                                  2
 weighted       0.67    0.67    0.6       6
     avg                           7
```

Result:

Thus, the output executed successfully.

Week: 8. A **ANIMAL CLASSIFICATION** Date:

Aim:

To classify animals into various species or categories based on their characteristics using a Decision Tree.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.tree import
DecisionTreeClassifier from
sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import accuracy_score,
classification_report # Load dataset
df =
pd.read_csv("animal_classification.c
sv") # Drop non-essential columns
df = df.drop(columns=["name"])
# Encode categorical columns
manually le_diet = LabelEncoder()
df["diet"] =
le_diet.fit_transform(df["diet"])
le_class = LabelEncoder()
df["class"] =
le_class.fit_transform(df["class"]) #
Split features and target
X =
df.drop(columns=["class"
]) y = df["class"]
# Ensure balanced split using stratification
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=42) # Train Decision Tree model
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

```
# Predictions
y_pred =
clf.predict(X_test) #
Results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred,
zero_division=1))
```

Output:

```
Accuracy: 0.5

Classification Report:

          precision recall f1-score  support


       0     0.00    1.00    0.00      0

       1     1.00    1.00    1.00      1

       3     1.00    0.00    0.00      1


  accuracy                       0.5       2
                                 0
 macro avg    0.67    0.67    0.3        2
                                  3
 weighted       1.00    0.50    0.5        2
     avg                          0
```

Result:

Thus, the output was executed successfully.

Week: 8. B

PLANT DISEASE CLASSIFICATION

Date:

Aim:

To classify different plant diseases based on plant leaf characteristics using a Decision Tree.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.tree import
DecisionTreeClassifier from
sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import accuracy_score,
classification_report # Load dataset
df = pd.read_csv("plant_disease_classification.csv") # Replace
with actual dataset # Strip spaces from column names (to avoid
hidden errors)
df.columns =
df.columns.str.strip() #
Identify categorical columns
categorical_columns = df.select_dtypes(include=["object"]).columns # Select only non-numeric columns
# Apply Label Encoding to categorical columns
df[categorical_columns] =
df[categorical_columns].apply(LabelEncoder().fit_transform) # Split
features and target
X = df.drop(columns=["disease"]) #
Feature columns y = df["disease"] #
Target column
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Train Decision Tree model
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
# Predictions
```

```
y_pred =
clf.predict(X_test) #
Results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0

Classification Report:

            precision recall f1-score  support


         1     1.00    1.00    1.00      2


  accuracy                     1.0        2
                               0
 macro avg    1.00    1.00    1.0        2
                                0
 weighted       1.00    1.00    1.0        2
     avg                         0
```

Result:

Thus, the output was executed successfully.

Week: 8. C **VEHICLE TYPE CLASSIFICATION** Date:

Aim:

To classify types of vehicles (e.g., car, truck, bike) based on their features using a Decision Tree.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.tree import
DecisionTreeClassifier from
sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import accuracy_score,
classification_report # Load dataset
df = pd.read_csv("vehicle_classification.csv") # Replace with
actual dataset # Strip spaces from column names
df.columns = df.columns.str.strip()
# Apply Label Encoding to all categorical columns
label_encoder = LabelEncoder()
df[df.select_dtypes(include=['object']).columns] =
df.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
# Split features and target
X = df.drop(columns=["vehicle_type"]) #
Feature columns y = df["vehicle_type"] #
Target column
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Train Decision Tree model
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
# Predictions
y_pred =
clf.predict(X_test) #
Results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0

Classification Report:

              precision recall f1-score  support


        1      1.00    1.00    1.00      1

        3      1.00    1.00    1.00      1


   accuracy                    1.0       2
                               0
  macro avg    1.00    1.00    1.0       2
                               0
  weighted     1.00    1.00    1.0       2
     avg                       0
```

Result:

Thus, the output was executed successfully.

Week: 9. A **EMPLOYEE HOPING PREDICTION** Date:

Aim:

To predict whether an employee will hop to another job based on their work history and personal details using Random Forest.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.ensemble
import RandomForestClassifier from
sklearn.preprocessing import
StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score # Load dataset
df = pd.read_csv("employee_hopping.csv")
# Convert categorical features to numeric using
one-hot encoding df = pd.get_dummies(df,
drop_first=True)
# Handle missing values by filling with the mean for
numeric columns
df.fillna(df.select_dtypes(include=['number']).mean(),
inplace=True) # Split data into features and target
X = df.drop(columns=["Hopped"])
y = df["Hopped"] # Target variable indicating if an
employee hopped jobs # Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # Standardize the data
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)
# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
model.fit(X_train_scaled, y_train)
y_pred =
model.predict(X_test_scaled) #
Output results
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))
```

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

Output:

```
Accuracy: 0.5
Confusion Matrix:
 [[0 1]
 [0 1]]
Classification Report:
              precision recall  f1-score  support

        0     0.00    0.00    0.00      1
        1     0.50    1.00    0.67      1

  accuracy                    0.5       2
                              0
 macro avg    0.25    0.50    0.3       2
                              3
 weighted      0.25    0.50    0.3       2
     avg                      3
```

Result:

Thus, the output was executed successfully.

Week: 9. B **PROMOTION ELIGIBILITY PREDICTION** Date:

Aim:

To predict the eligibility of employees for promotion based on their work performance and other factors using Random Forest.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.ensemble
import RandomForestClassifier from
sklearn.preprocessing import
StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score # Load dataset
df = pd.read_csv("promotion_eligibility.csv")
# Convert categorical features to numeric using
one-hot encoding df = pd.get_dummies(df,
drop_first=True)
# Handle missing values by filling with the mean for
numeric columns
df.fillna(df.select_dtypes(include=['number']).mean(),
inplace=True) # Split data into features and target
X = df.drop(columns=["Promotion_Eligible"])
y = df["Promotion_Eligible"] # Target variable indicating
eligibility for promotion # Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # Standardize the data
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)
# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
model.fit(X_train_scaled, y_train)
y_pred =
model.predict(X_test_scaled) #
Output results
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))
```

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

Output:

```
Accuracy: 1.0
Confusion Matrix:
 [[1 0]
 [0 2]]
Classification Report:
              precision recall f1-score  support


        0      1.00    1.00    1.00       1
        1      1.00    1.00    1.00       2


  accuracy                     1.0        3
                               0
 macro avg    1.00    1.00    1.0        3
                                0
 weighted      1.00    1.00    1.0        3
     avg                        0
```

Result:

Thus, the output was executed successfully.

Week: 9. C **FRAUD DETECTION IN BANKING** Date:

Aim:

To detect fraudulent transactions in banking based on transactional features using Random
Forest.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.ensemble
import RandomForestClassifier from
sklearn.preprocessing import
StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score # Load dataset
df = pd.read_csv("bank_fraud.csv")
# Convert categorical features to numeric using
one-hot encoding df = pd.get_dummies(df,
drop_first=True)
# Handle missing values by filling with the mean for
numeric columns
df.fillna(df.select_dtypes(include=['number']).mean(),
inplace=True) # Split data into features and target
X = df.drop(columns=["Fraud"])
y = df["Fraud"] # Target variable indicating if a
transaction is fraudulent # Split into training and testing
sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) # Standardize the data
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)
# Train Random Forest model
```

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)
y_pred =
model.predict(X_test_scaled) #
Output results
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))
```

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

Output:

```
Accuracy: 0.6666666666666666

Confusion Matrix:

 [[1 0]

 [1 1]]

Classification Report:

              precision recall f1-score  support


          0      0.50    1.00    0.67       1

          1      1.00    0.50    0.67       2


   accuracy                     0.6        3
                                7
  macro avg    0.75    0.75    0.6        3
                                7
  weighted      0.83    0.75    0.6        3
      avg                       7
```

Result:

Thus, the output was executed successfully.

Week: 10. A **PATIENT PHYSICAL ACTIVITIES PREDICTION** Date:

Aim:

To predict different types of physical activities performed by patients using Gradient Boosting.

Program Code:

```
import pandas as
pd import numpy as
np import xgboost
as xgb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
LabelEncoder from sklearn.metrics import
accuracy_score
# Load dataset
df = pd.read_csv("mhealth.csv")  # Update with
actual dataset path  # Encode categorical labels
label_encoder = LabelEncoder()
df["Activity"] =
label_encoder.fit_transform(df["Activity"])  #
Define features and labels
X =
df.drop(columns=["Activity"
]) y = df["Activity"]
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Standardization
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)
# Train XGBoost model
model = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1,
random_state=42) model.fit(X_train, y_train)
# Predictions
y_pred =
model.predict(X_test) #
Accuracy
accuracy = accuracy_score(y_test,
y_pred) print(f"Model Accuracy:
{accuracy:.2f}")
```

```
# Decode labels back to original values
y_pred_original =
label_encoder.inverse_transform(y_pred)
print("Predicted Activities:", y_pred_original)
```

Output:

```
Model Accuracy: 0.00
Predicted Activities:
['Walking']
```

Result:

Thus, the output was executed successfully.

Week: 10. B **DISEASE PROGRESSION PREDICTION** Date:

Aim:

To predict the progression of diseases in patients based on clinical data using Gradient Boosting.

Program Code:

```
import pandas as
pd import numpy as
np import xgboost
as xgb
from sklearn.model_selection import
train_test_split from sklearn.preprocessing
import StandardScaler
from sklearn.metrics import
mean_absolute_error, r2_score # Load dataset
df = pd.read_csv("disease_progression.csv")  # Update with
actual dataset path # Handle missing values (if any)
df.fillna(df.mean(), inplace=True)
# Define features and target variable
X =
df.drop(columns=["Diabetes_Progressi
on"]) y = df["Diabetes_Progression"]
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Standardization
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)
# Train XGBoost Regressor
model = xgb.XGBRegressor(n_estimators=200, learning_rate=0.05,
max_depth=5, random_state=42) model.fit(X_train, y_train)
# Predictions
```

```
y_pred =
model.predict(X_test) #
Model Evaluation
mae =
mean_absolute_error(y_test,
y_pred) r2 = r2_score(y_test,
y_pred)
```

```
print(f"Mean Absolute Error:
{mae:.2f}") print(f"R² Score:
{r2:.2f}")
```

Output:

```
Mean Absolute Error:
44.98 R² Score: nan
```

Result:

Thus, the output was executed successfully.

Week: 10. C **HEALTH INSURANCE CLAIM PREDICTION** Date:

Aim:

To predict whether a health insurance claim will be approved or rejected using Gradient Boosting.

Program Code:

```
import pandas as
pd import numpy as
np
from sklearn.model_selection import
train_test_split from sklearn.ensemble
import AdaBoostClassifier from sklearn.tree
import DecisionTreeClassifier from
sklearn.metrics import accuracy_score
# Load dataset
df = pd.read_csv("health_insurance_claims.csv")  # Update with
actual dataset path  # Define features and labels
X = df.drop(columns=["Claim_Status"]) # 'Claim_Status' should be
the target variable y = df["Claim_Status"]
# Convert categorical features
if needed X =
pd.get_dummies(X)
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Train AdaBoost model
model = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=2),
n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train,
y_train) # Predictions
on test set
y_pred =
model.predict(X_test) #
Accuracy
accuracy = accuracy_score(y_test,
y_pred) print(f"Model Accuracy:
```

```
{accuracy:.2f}") # Prediction for
new claim
new_claim =
   pd.DataFrame([{ "Age
   ": 45,
   "BMI": 28.5,
   "Smoker": "Yes",
   "Claim_Amount":
   5000,
   "Hospital_Visits": 2
}])  # Modify based on actual dataset features
new_claim = pd.get_dummies(new_claim) # Convert
categorical features # Align new data with training columns
new_claim = new_claim.reindex(columns=X.columns,
fill_value=0) # Predict claim status
predicted_status =
model.predict(new_claim)[0] print(f"Predicted
Claim Status: {predicted_status}")
```

Output:

```
Model Accuracy: 1.00
Predicted Claim Status: Approved
```

Result:

Thus, the output was executed successfully.

Week: 11. A **SHOPPING MALL CUSTOMER SEGMENTATION** Date:

Aim:

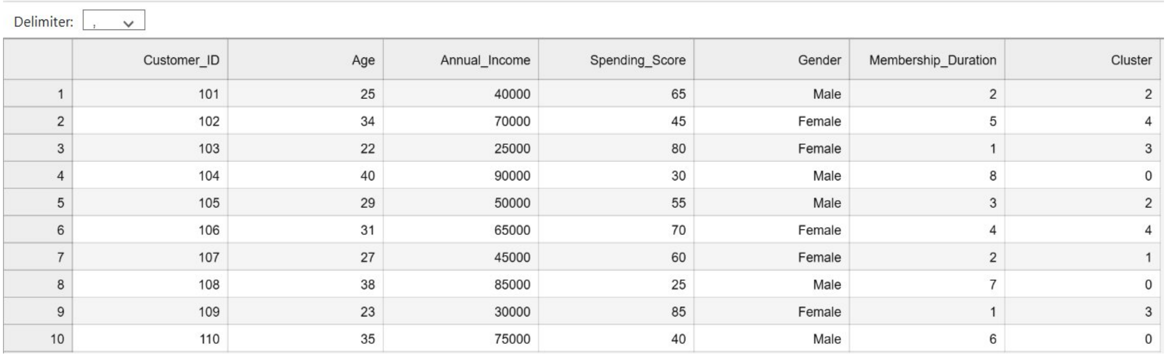
To segment customers of a shopping mall into different groups based on their spending patterns and income using clustering.

Program Code:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import
StandardScaler # Load dataset
df =
pd.read_csv("shoppingmall_customer_segmentatio
n.csv") # Select relevant features for clustering
X = df[['Age', 'Annual_Income', 'Spending_Score', 'Gender',
'Membership_Duration']] # Convert categorical variables to
numerical
X = pd.get_dummies(X, columns=['Gender'],
drop_first=True) # Standardize the data
scaler = StandardScaler()
X_scaled =
scaler.fit_transform(X)
# Determine the optimal number of clusters using the
Elbow Method wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',
    random_state=42) kmeans.fit(X_scaled)
    wcss.append(kmeans.ine
rtia_) # Train K-Means
clustering model
optimal_k = 5 # Choose the number of clusters based on the
elbow method kmeans = KMeans(n_clusters=optimal_k, init='k-
means++', random_state=42) df['Cluster'] =
kmeans.fit_predict(X_scaled)
# Save the clustered data
```

```
df.to_csv("shopping_mall_customers_clustered.csv",
index=False)
```

Output:



Delimiter: ,

	Customer_ID	Age	Annual_Income	Spending_Score	Gender	Membership_Duration	Cluster
1	101	25	40000	65	Male	2	2
2	102	34	70000	45	Female	5	4
3	103	22	25000	80	Female	1	3
4	104	40	90000	30	Male	8	0
5	105	29	50000	55	Male	3	2
6	106	31	65000	70	Female	4	4
7	107	27	45000	60	Female	2	1
8	108	38	85000	25	Male	7	0
9	109	23	30000	85	Female	1	3
10	110	35	75000	40	Male	6	0

Result:

Thus, the output was executed successfully.

Week: 11. B **PRODUCT RECOMMENDATION** Date:

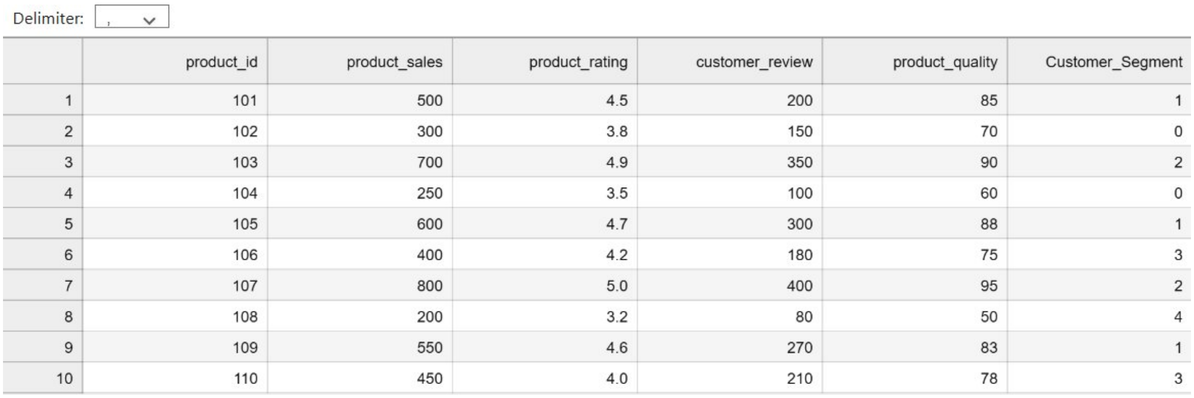
Aim:

To recommend products to customers by clustering similar customers based on their purchasing behavior.

Program Code:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import
StandardScaler # Load dataset
df =
pd.read_csv("product_recommendation.
csv") # Select appropriate features for
clustering
X = df[['product_sales', 'product_rating', 'customer_review',
'product_quality']] # Standardize the data
scaler = StandardScaler()
X_scaled =
scaler.fit_transform(X)
# Determine the optimal number of clusters using the
Elbow Method wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++',
   random_state=42) kmeans.fit(X_scaled)
   wcss.append(kmeans.ine
rtia_) # Train K-Means
clustering model
optimal_k = 5 # Choose the number of clusters based on the
elbow method kmeans = KMeans(n_clusters=optimal_k, init='k-
means++', random_state=42) df['Customer_Segment'] =
kmeans.fit_predict(X_scaled)
# Save the clustered data
df.to_csv("customer_segments.csv",
index=False)
```

Output:



Delimiter: ,

	product_id	product_sales	product_rating	customer_review	product_quality	Customer_Segment
1	101	500	4.5	200	85	1
2	102	300	3.8	150	70	0
3	103	700	4.9	350	90	2
4	104	250	3.5	100	60	0
5	105	600	4.7	300	88	1
6	106	400	4.2	180	75	3
7	107	800	5.0	400	95	2
8	108	200	3.2	80	50	4
9	109	550	4.6	270	83	1
10	110	450	4.0	210	78	3

Result:

Thus, the output was executed successfully.

Week: 11. C **TOURIST BEHAVIOR ANALYSIS** Date:

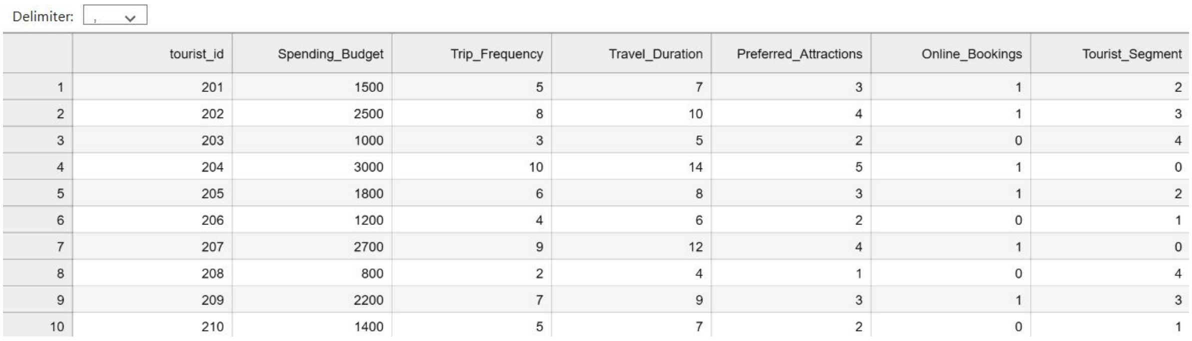
Aim:

To analyze and segment tourists' behaviors based on their spending and travel patterns using clustering.

Program Code:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import
StandardScaler # Load dataset
df =
pd.read_csv("tourist_behaviour.cs
v") # Select relevant features for
clustering
X = df[['Spending_Budget', 'Trip_Frequency', 'Travel_Duration',
'Preferred_Attractions', 'Online_Bookings']]
# Standardize the
data scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X)
# Determine the optimal number of clusters using the
Elbow Method wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++',
   random_state=42) kmeans.fit(X_scaled)
   wcss.append(kmeans.ine
rtia_) # Train K-Means
clustering model
optimal_k = 5 # Choose the number of clusters based on the
elbow method kmeans = KMeans(n_clusters=optimal_k, init='k-
means++', random_state=42) df['Tourist_Segment'] =
kmeans.fit_predict(X_scaled)
# Save the clustered data
df.to_csv("tourist_segments.csv",
index=False)
```

Output:



Delimiter: ,

	tourist_id	Spending_Budget	Trip_Frequency	Travel_Duration	Preferred_Attractions	Online_Bookings	Tourist_Segment
1	201	1500	5	7	3	1	2
2	202	2500	8	10	4	1	3
3	203	1000	3	5	2	0	4
4	204	3000	10	14	5	1	0
5	205	1800	6	8	3	1	2
6	206	1200	4	6	2	0	1
7	207	2700	9	12	4	1	0
8	208	800	2	4	1	0	4
9	209	2200	7	9	3	1	3
10	210	1400	5	7	2	0	1

Result:

Thus, the output was executed successfully.

Week: 12. A **CUSTOMER SEGMENTATION** Date:

Aim:

To segment customers into groups with similar behaviors using DBSCAN clustering.

Program Code:

```
import pandas as
pd import numpy as
np
from sklearn.preprocessing import
StandardScaler from sklearn.cluster
import KMeans
# Load the dataset
df = pd.read_csv("customer_segmentation.csv")  # Replace with
your actual CSV file # Select relevant features for clustering
features = df[['Age', 'Annual_Income',
'Spending_Score']] # Standardize the data
scaler = StandardScaler()
X_scaled =
scaler.fit_transform(features) #
Apply K-Means Clustering
num_clusters = 4  # Choose optimal clusters using Elbow Method
kmeans = KMeans(n_clusters=num_clusters, random_state=42,
n_init=10)  df['Cluster'] = kmeans.fit_predict(X_scaled)
# Save the segmented data
df.to_csv("customer_segmented_dataset.csv",
index=False)  # Print the cluster distribution
print(df['Cluster'].value_counts())
print("Customer segmentation completed and saved as
'customer_segmented_dataset.csv'.")
```

Output:

```
2   4
0   3
1   2
3   1
```

```
Name: Cluster, dtype: int64
Customer segmentation completed and saved as 'customer_segmented_dataset.csv'.
```

Result:

Thus, the output was executed successfully.

Week: 12. B **SOCIAL NETWORK ANALYSIS** Date:

Aim:

To analyze social network structures and group users into clusters based on their connections using DBSCAN.

Program Code:

```
import pandas as
pd import numpy as
np
from sklearn.preprocessing import
StandardScaler from sklearn.cluster
import DBSCAN
# Load the dataset
df = pd.read_csv("social_network_data.csv")  # Replace
with actual file # Select relevant features
features = df[['Num_Friends', 'Mutual_Connections', 'Interaction_Frequency',
'Network_Centrality']] # Standardize the data
scaler = StandardScaler()
X_scaled =
scaler.fit_transform(features) #
Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.9, min_samples=3)  # Adjust parameters based on network
structure df['Cluster'] = dbscan.fit_predict(X_scaled)
# Save results
df.to_csv("social_network_clusters.csv",
index=False) # Print cluster distribution
print("Clusters found:",
df['Cluster'].value_counts()) print("Results
saved as 'social_network_clusters.csv'.")
```

Output:

```
Clusters found: 0        7
1   3
Name: Cluster, dtype: int64
Results saved as 'social_network_clusters.csv'.
```

Result:

Thus, the output was executed successfully.

Week: 13 Date:

CLASSIFY IRIS FLOWERS INTO SPECIES (SETOSA, VERSICOLOR OR VIRGINCIA) BASED ON THEIR SEPAL & PETAL MEASUREMENT

Aim:

To classify Iris flowers into species (setosa, versicolor, or virginica) based on sepal and petal measurements using classification techniques.

Program Code:

```
import pandas as pd
from sklearn.model_selection import
train_test_split from sklearn.ensemble
import RandomForestClassifier from
sklearn.preprocessing import
StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix # Load dataset
df =
pd.read_csv("iris_flower.csv
") # Select features and
target variable
X = df[['sepal_length', 'sepal_width', 'petal_length',
'petal_width']] y = df['species']
# Standardize the
data scaler =
StandardScaler()
X_scaled =
scaler.fit_transform(X) #
Split into training and
testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42) # Train Random Forest model
model = RandomForestClassifier(n_estimators=100,
random_state=42)  model.fit(X_train, y_train)
# Make predictions
```

```
y_pred =
model.predict(X_test) #
Evaluate model
performance
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred)) print("Classification Report:\n",
classification_report(y_test, y_pred)) # Display
predictions with actual labels
predictions_df = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
```

```
print("\nPredicted Species for Test
Data:\n")
print(predictions_df.to_string(index
=False))
```

Output:

```
Accuracy:   1.0
Confusion
Matrix:
 [[2 0 0]
 [0 3 0]
 [0 0 4]]
Classification Report:
            precision recall f1-score  support


     setosa    1.00    1.00    1.00      2
   versicol     1.00    1.00    1.00       3
         or
   virginic    1.00    1.00    1.00       4
          a


   accuracy                    1.0        9
                               0
 macro avg    1.00    1.00    1.0        9
                                0
  weighted       1.00    1.00    1.0        9
      avg                            0
Predicted Species for Test Data:
 Actual Predicted
virginica virginica
versicolor versicolor
virginica virginica
versicolor versicolor
  setosa  setosa
  setosa  setosa
      virginica
      virginica
      virginica
 virginica
versicolor
versicolor
```

Result:

Thus, the output was executed successfully.