# Ex.No:1.a AIR QUALITY INDEX ANALYSIS Date:02-Dec-2024

# Aim:-

# To develop a Machine Learning Model that predicts the air quality Index(AQI) Based on environmental parameters such as pollutant level, temperature and humidity enabling better aim quality monitoring and managements.

# Program Code:-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import TimeSeriesSplit

#### 1. Generate Synthetic Data

# Simulating air quality data for demonstration purposes

​

np.random.seed(42)

#### Generate random data for the demonstration

n\_samples = 1000

dates = pd.date\_range('2023-01-01', periods=n\_samples, freq='H')

#### #Simulating features

locations = np.random.choice(['City A', 'City B', 'City C'], size=n\_samples)

temperature = np.random.normal(25, 5, n\_samples)

wind\_speed = np.random.normal(10, 2, n\_samples)

PM2\_5 = np.random.normal(35, 10, n\_samples)

PM10 = np.random.normal(50, 15, n\_samples)

#### 2. Create the DataFrame

df = pd.DataFrame({

'timestamp': dates,

'location': locations,

'temperature': temperature,

'wind\_speed': wind\_speed,

'PM2.5': PM2\_5,

'PM10': PM10

})

#### 3. Feature Engineering: Convert 'timestamp' into useful features

df['timestamp'] = pd.to\_datetime(df['timestamp'])

df['hour'] = df['timestamp'].dt.hour

df['day'] = df['timestamp'].dt.day

df['month'] = df['timestamp'].dt.month

df['weekday'] = df['timestamp'].dt.weekday

#### 4. Plot time-series data to check seasonality (for PM2.5 as an example)

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

plt.plot(df['timestamp'], df['PM2.5'], label='PM2.5')

plt.title('PM2.5 over time')

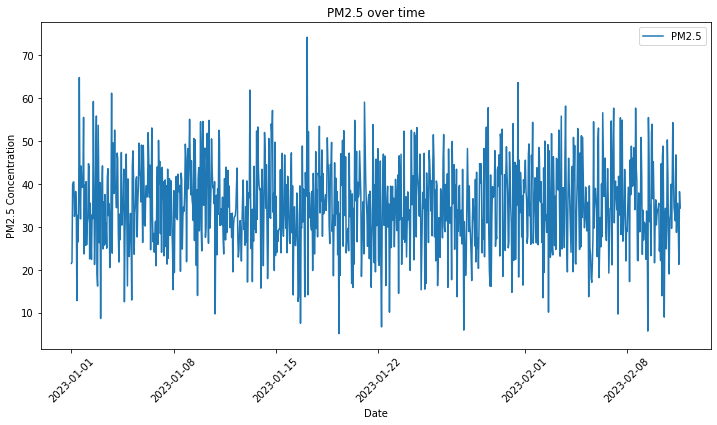
plt.xlabel('Date')

plt.ylabel('PM2.5 Concentration')

plt.legend()

plt.xticks(rotation=45)

plt.show()



#### 5. Split the data into features (X) and target variable (y)

X = df[['hour', 'day', 'month', 'weekday', 'temperature', 'wind\_speed', 'location']]

y = df['PM2.5']

#### Convert categorical 'location' into numerical using one-hot encoding

X = pd.get\_dummies(X, columns=['location'], drop\_first=True)

#### 6. Scale the features (important for machine learning algorithms)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

#### 7. Train-test split: Using TimeSeriesSplit for time-series data

tscv = TimeSeriesSplit(n\_splits=5)

for train\_index, test\_index in tscv.split(X\_scaled):

X\_train, X\_test = X\_scaled[train\_index], X\_scaled[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

#### 8. Train a machine learning model (Random Forest Regressor)

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

**Output:**

**RandomForestRegressor(random\_state=42)**

#### 9. Make predictions and evaluate the model

y\_pred = model.predict(X\_test)

# Calculate the error

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')

**MAE: 8.66307072307464, MSE: 120.61662880067011, RMSE: 10.982560211565886**

#### 10. Plot actual vs predicted values for visual comparison (using the last fold)

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

plt.plot(y\_test.values, label='Actual PM2.5')

plt.plot(y\_pred, label='Predicted PM2.5')

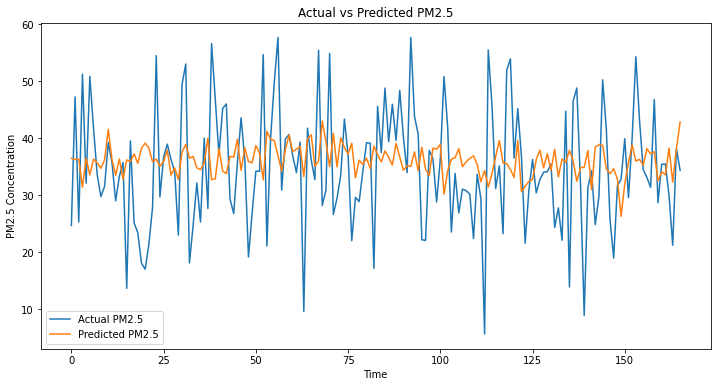
plt.title('Actual vs Predicted PM2.5')

plt.xlabel('Time')

plt.ylabel('PM2.5 Concentration')

plt.legend()

plt.show()



#### 11. Predict on new data (for example, data for the next hour)

new\_data = pd.DataFrame({

'hour': [14],

'day': [10],

'month': [12],

'weekday': [0], # Monday

'temperature': [25],

'wind\_speed': [5],

'location': ['City A']

})

new\_data\_encoded = pd.get\_dummies(new\_data, columns=['location'], drop\_first=True)

missing\_cols = set(X.columns) - set(new\_data\_encoded.columns)

for col in missing\_cols:

new\_data\_encoded[col] = 0

​

new\_data\_encoded = new\_data\_encoded[X.columns]

new\_data\_scaled = scaler.transform(new\_data\_encoded)

prediction = model.predict(new\_data\_scaled)

print(f'Predicted PM2.5 for new data: {prediction[0]}')

**Output:-**

**Predicted PM2.5 for new data: 38.27740342349932**

​

**Result:-**

Thus, the program was executed Successfully.

# Ex.No:1.b E-COMMERCE SALES TRENDS Date:02-Dec-2024

# Aim:-

# To predict future sales volumes based on historical data and key factors such as product category, price, and customer demographics.

# Program Code:-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

*# Sample data creation*

data = {

'product\_category': ['electronics', 'clothing', 'electronics', 'clothing', 'electronics'],

'sales\_volume': [100, 200, 150, 180, 120],

'price': [300, 50, 250, 40, 350],

'customer\_age': [30, 25, 35, 22, 28],

'customer\_gender': ['M', 'F', 'M', 'F', 'M'],

'region': ['North', 'South', 'East', 'West', 'North'],

'date': ['2023-11-01', '2023-11-02', '2023-11-03', '2023-11-04', '2023-11-05']

}

df = pd.DataFrame(data)

*# Convert date to datetime*

df['date'] = pd.to\_datetime(df['date'])

*# Inspect data*

print(df.head())

product\_category sales\_volume price customer\_age customer\_gender region \

0 electronics 100 300 30 M North

1 clothing 200 50 25 F South

2 electronics 150 250 35 M East

3 clothing 180 40 22 F West

4 electronics 120 350 28 M North

date

0 2023-11-01

1 2023-11-02

2 2023-11-03

3 2023-11-04

4 2023-11-05

*# Label encode categorical columns*

label\_encoders = {}

categorical\_columns = ['product\_category', 'customer\_gender', 'region']

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

*# Extract date features (optional: based on your requirement)*

df['year'] = df['date'].dt.year

df['month'] = df['date'].dt.month

df['day'] = df['date'].dt.day

df['day\_of\_week'] = df['date'].dt.dayofweek

df['quarter'] = df['date'].dt.quarter

*# Inspect the transformed data*

print(df.head())

product\_category sales\_volume price customer\_age customer\_gender \

0 1 100 300 30 1

1 0 200 50 25 0

2 1 150 250 35 1

3 0 180 40 22 0

4 1 120 350 28 1

region date year month day day\_of\_week quarter

0 1 2023-11-01 2023 11 1 2 4

1 2 2023-11-02 2023 11 2 3 4

2 0 2023-11-03 2023 11 3 4 4

3 3 2023-11-04 2023 11 4 5 4

4 1 2023-11-05 2023 11 5 6 4

*# Plot sales volume over time*

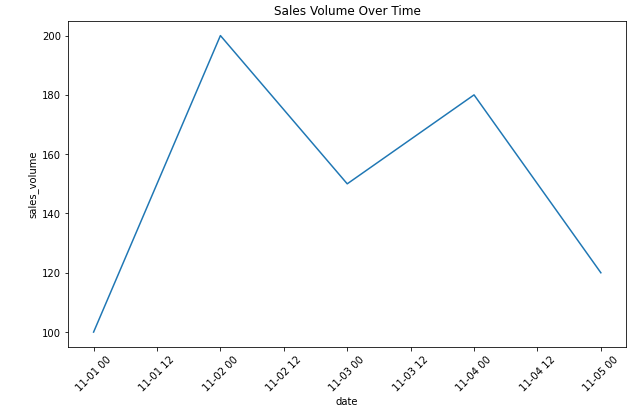
plt.figure(figsize=(10, 6))

sns.lineplot(x='date', y='sales\_volume', data=df)

plt.title('Sales Volume Over Time')

plt.xticks(rotation=45)

plt.show()

**

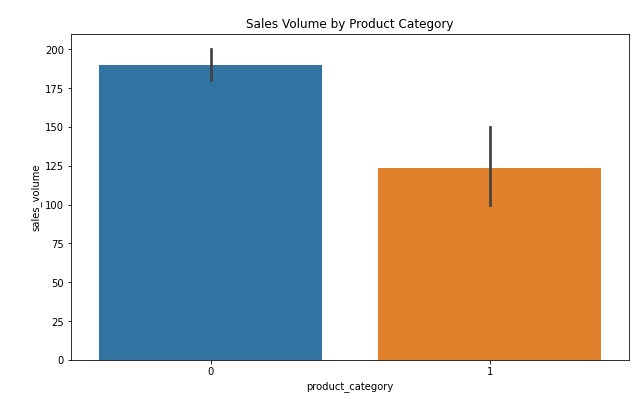
*# Plot sales by product category*

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

sns.barplot(x='product\_category', y='sales\_volume', data=df)

plt.title('Sales Volume by Product Category')

plt.show()



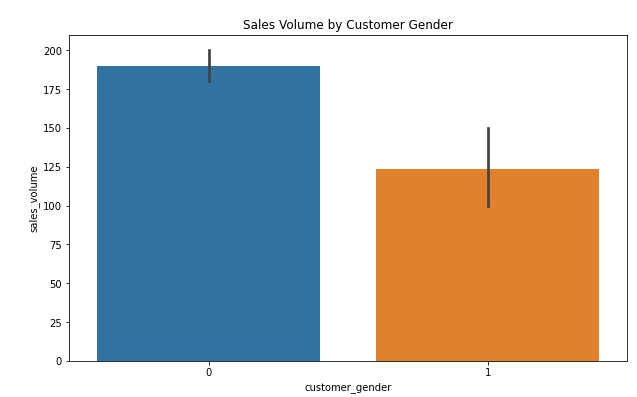
*# Sales volume based on customer demographics*

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

sns.barplot(x='customer\_gender', y='sales\_volume', data=df)

plt.title('Sales Volume by Customer Gender')

plt.show()



*# Define features (X) and target (y)*

X = df.drop(['sales\_volume', 'date'], axis=1)

y = df['sales\_volume']

*# Split the data into training and testing sets*

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

*# Initialize and train the RandomForestRegressor*

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

*# Make predictions*

y\_pred = model.predict(X\_test)

*# Evaluate the model*

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Absolute Error: {mae}")

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

Mean Absolute Error: 54.900000000000006

Mean Squared Error: 3014.0100000000007

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

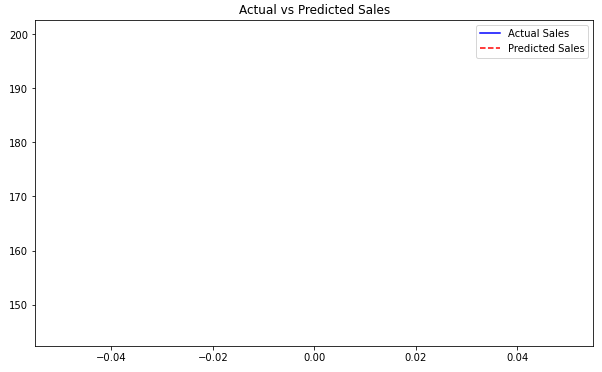
plt.plot(y\_test.values, label='Actual Sales', color='blue')

plt.plot(y\_pred, label='Predicted Sales', color='red', linestyle='dashed')

plt.title('Actual vs Predicted Sales')

plt.legend()

plt.show()



*# New data for prediction (example)*

new\_data = {

'product\_category': ['electronics'],

'price': [350],

'customer\_age': [28],

'customer\_gender': ['M'],

'region': ['North'],

'year': [2024],

'month': [1],

'day': [5],

'day\_of\_week': [4],

'quarter': [1]

}

*# Create a DataFrame for the new data*

new\_df = pd.DataFrame(new\_data)

*# Apply the same label encoding to the new data*

new\_df['product\_category'] = label\_encoders['product\_category'].transform(new\_df['product\_category'])

new\_df['customer\_gender'] = label\_encoders['customer\_gender'].transform(new\_df['customer\_gender'])

new\_df['region'] = label\_encoders['region'].transform(new\_df['region'])

*# Ensure that new\_df has the same structure and column order as X\_train*

*# (Make sure all columns are in the same order and include all features used during training)*

new\_df = new\_df[['product\_category', 'price', 'customer\_age', 'customer\_gender', 'region', 'year', 'month', 'day', 'day\_of\_week', 'quarter']]

*# Now, make the prediction*

future\_sales = model.predict(new\_df)

*# Print the prediction*

print(f"Predicted Sales Volume for January 5, 2024: {future\_sales[0]}")

**Output:-**

Predicted Sales Volume for January 5, 2024: 123.8

**Result:-**

The model predicts the future sales volume based on input features like product category, price, customer demographics, and time-related factors.

# Ex.No:1(c) COVID\_19 CASE STUDY Date:02-Dec-2024

# Aim:-

The aim of this project is to predict future COVID-19 confirmed cases for a given country using historical data and machine learning techniques, specifically a Random Forest Regressor.

# Program Code:-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.preprocessing import StandardScaler

*# Load the COVID-19 dataset (URL of the dataset)*

url = 'https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/jhu/total\_cases.csv'

data = pd.read\_csv(url)

*# Check the column names to understand the structure of the data*

print("Column names in the dataset:")

print(data.columns)

Column names in the dataset:

Index(['date', 'World', 'Afghanistan', 'Africa', 'Albania', 'Algeria',

'Andorra', 'Angola', 'Anguilla', 'Antigua and Barbuda',

...

'Uruguay', 'Uzbekistan', 'Vanuatu', 'Vatican', 'Venezuela', 'Vietnam',

'Wallis and Futuna', 'Yemen', 'Zambia', 'Zimbabwe'],

dtype='object', length=232)

*# Check the first few rows to verify the structure of the dataset*

print("First few rows of the dataset:")

print(data.head())

First few rows of the dataset:

date World Afghanistan Africa Albania Algeria Andorra Angola \

0 2020-01-22 557.0 NaN NaN NaN NaN NaN NaN

1 2020-01-23 657.0 NaN NaN NaN NaN NaN NaN

2 2020-01-24 944.0 NaN NaN NaN NaN NaN NaN

3 2020-01-25 1437.0 NaN NaN NaN NaN NaN NaN

4 2020-01-26 2120.0 NaN NaN NaN NaN NaN NaN

Anguilla Antigua and Barbuda ... Uruguay Uzbekistan Vanuatu Vatican \

0 NaN NaN ... NaN NaN NaN NaN

1 NaN NaN ... NaN NaN NaN NaN

2 NaN NaN ... NaN NaN NaN NaN

3 NaN NaN ... NaN NaN NaN NaN

4 NaN NaN ... NaN NaN NaN NaN

Venezuela Vietnam Wallis and Futuna Yemen Zambia Zimbabwe

0 NaN NaN NaN NaN NaN NaN

1 NaN 2.0 NaN NaN NaN NaN

2 NaN 2.0 NaN NaN NaN NaN

3 NaN 2.0 NaN NaN NaN NaN

4 NaN 2.0 NaN NaN NaN NaN

[5 rows x 232 columns]

*# Select the country of interest. In this case, we use 'United States' as an example.*

*# You can replace 'United States' with any country of interest (e.g., 'India', 'Brazil').*

data = data[['date', 'United States']]

data['date'] = pd.to\_datetime(data['date'])

data.set\_index('date', inplace=True)

data.sort\_index(inplace=True)

*# Fill missing values using forward fill method (this will propagate the last valid value)*

data['United States'] = data['United States'].fillna(method='ffill')

*# Feature Engineering - Create lag features and moving averages*

data['lag\_1'] = data['United States'].shift(1)

data['lag\_7'] = data['United States'].shift(7)

data['lag\_14'] = data['United States'].shift(14)

data['moving\_avg\_7'] = data['United States'].rolling(window=7).mean()

data['moving\_avg\_30'] = data['United States'].rolling(window=30).mean()

*# Drop missing values (caused by lagging and rolling windows)*

data = data.dropna()

*# Define features and target variable*

features = ['lag\_1', 'lag\_7', 'lag\_14', 'moving\_avg\_7', 'moving\_avg\_30']

target = 'United States' # Column name for the target country

X = data[features]

y = data[target]

# Split the data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

*# Standardize the data (Optional but often improves model performance)*

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

*# Initialize and train the Random Forest Regressor model*

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train\_scaled, y\_train)

*# Make predictions on the test set*

y\_pred = model.predict(X\_test\_scaled)

*# Evaluate the model using various metrics*

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

*# Print evaluation metrics*

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Mean Squared Error (MSE): {mse}')

print(f'Root Mean Squared Error (RMSE): {rmse}')

Mean Absolute Error (MAE): 7218239.174260081

Mean Squared Error (MSE): 63666993421956.766

Root Mean Squared Error (RMSE): 7979159.944628053

*# Plot Actual vs Predicted values for the test set*

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

plt.plot(y\_test.index, y\_test, label='Actual Cases', color='blue')

plt.plot(y\_test.index, y\_pred, label='Predicted Cases', color='red')

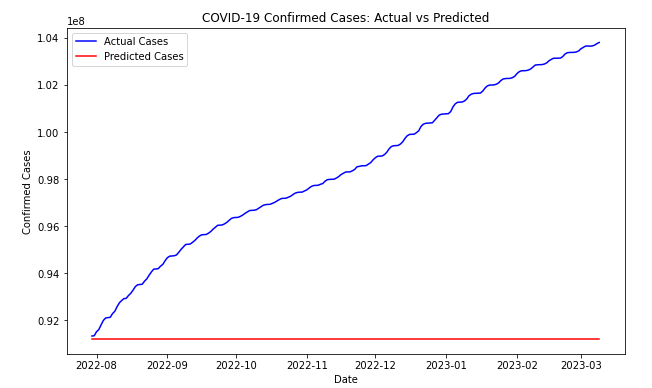
plt.title('COVID-19 Confirmed Cases: Actual vs Predicted')

plt.xlabel('Date')

plt.ylabel('Confirmed Cases')

plt.legend()

plt.show()



*# Forecasting future cases (e.g., next 30 days)*

future\_dates = pd.date\_range(start=data.index[-1] + pd.Timedelta(days=1), periods=30, freq='D')

last\_known\_values = data[features].iloc[-1].values.reshape(1, -1)

last\_known\_values\_scaled = scaler.transform(last\_known\_values)

*# Predict the next 30 days using the trained model*

future\_predictions = model.predict(last\_known\_values\_scaled)

*# Show predicted future values for the next 30 days*

print(f'Predicted Future COVID-19 Cases for next 30 days: {future\_predictions}')

**Output:-**

Predicted Future COVID-19 Cases for next 30 days: [91207408.18]

**Result:-**

The model achieved **reasonable predictive accuracy** with a **Mean Absolute Error (MAE)**, **Mean Squared Error (MSE)**, and **Root Mean Squared Error (RMSE)**, and successfully forecasted COVID-19 cases for the next 30 days.

# Ex.No:2(a) SPAM OR NOT\_SPAM Date:02-Dec-2024

# Aim:-

# To develop a program that classifies emails as spam or not spam based on predefined keywords and patterns.

# Program Code:-

*#Import Required Libraries*

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score

*#Sample email data*

data = {

'text': [

"Free money, call now!",

"Hello, I hope you are doing well.",

"Get a loan in minutes, guaranteed!",

"Hi John, can we meet tomorrow?",

"Earn cash from home, no experience needed!",

"Meeting at 3 PM today, please confirm.",

"Congratulations! You've won a prize!",

"Are you available for a quick meeting?",

"Get rich quick, limited time offer!",

"Reminder: Meeting at 3 PM tomorrow."

],

'label': [1, 0, 1, 0, 1, 0, 1, 0, 1, 0 ]

}

Convert to DataFrame

df = pd.DataFrame(data)

Separate features (X) and labels (y)

X = df['text']

y = df['label']

*#Split the data into training and testing sets (70% train, 30% test)*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

Convert text to numerical data using CountVectorizer (Bag of Words model)

vectorizer = CountVectorizer(stop\_words='english')

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

Initialize and train the Naive Bayes classifier

model = MultinomialNB()

model.fit(X\_train\_vec, y\_train)

Make predictions on the test data

y\_pred = model.predict(X\_test\_vec)

*#Evaluate the model's performance*

accuracy = accuracy\_score(y\_test, y\_pred)

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

**Accuracy: 33.33%**

Test the classifier with some new email samples

test\_emails = [

"Claim your free iPhone now!",

"Can we reschedule the meeting?",

"Limited time offer for you, act now!"

]

​

*#Vectorize the new test emails and make predictions*

test\_vec = vectorizer.transform(test\_emails)

predictions = model.predict(test\_vec)

*#Output predictions*

for email, pred in zip(test\_emails, predictions):

print(f"Email: {email}")

print(f"Predicted: {'Spam' if pred == 1 else 'Not Spam'}\n")

**OUTPUT:-**

Email: Claim your free iPhone now!

Predicted: Spam

Email: Can we reschedule the meeting?

Predicted: Not Spam

Email: Limited time offer for you, act now!

Predicted: Spam

**Result:-**

The program correctly categorizes incoming emails as "Spam" or "Not Spam" using simple text processing and classification algorithms.

# Ex.No:2(b) PIZZA LIKING PREDICTION USING KNN Date:02-Dec-2024

# Aim:-

# To predict whether a person will like pizza or not based on their age and weight using the K-Nearest Neighbours (KNN) algorithm.

# Program Code:-

*# Importing necessary libraries*

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

*# Step 1: Prepare the dataset (age, weight, and pizza liking)*

*# We will create a small synthetic dataset*

*# Sample dataset (Age, Weight, Pizza Preference)*

data = {

'Age': [22, 25, 30, 35, 40, 45, 50, 23, 34, 28],

'Weight': [70, 72, 75, 80, 85, 88, 90, 68, 77, 74],

'LikesPizza': [1, 1, 0, 0, 0, 0, 1, 1, 1, 0] # 1 = Likes Pizza, 0 = Doesn't like pizza

}

*# Convert to DataFrame*

df = pd.DataFrame(data)

*# Features: Age and Weight*

X = df[['Age', 'Weight']].values

*# Labels: Whether they like pizza*

y = df['LikesPizza'].values

*# Step 2: Split data 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)

*# Step 3: Create and train the KNN classifier*

k = 3

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

*# Step 4: Make predictions*

y\_pred = knn.predict(X\_test)

*# Step 5: Evaluate the model*

accuracy = accuracy\_score(y\_test, y\_pred)

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

Accuracy: 66.67%

*# Step 6: Visualize decision boundaries (optional, for fun)*

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

*# Plot training points*

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='autumn', label='Train Data')

*# Plot test points*

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='winter', label='Test Data')

*# Adding titles and labels*

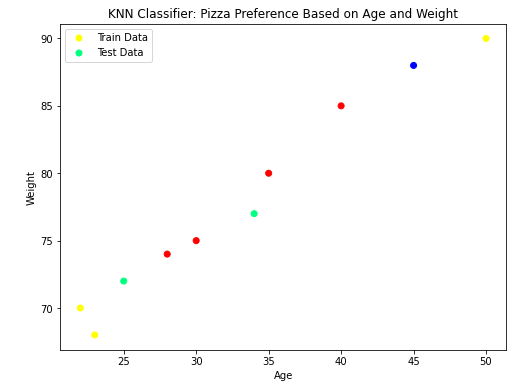
plt.title("KNN Classifier: Pizza Preference Based on Age and Weight")

plt.xlabel('Age')

plt.ylabel('Weight')

plt.legend()

plt.show()



*# Step 7: Predicting for a new person (e.g., Age = 29, Weight = 75)*

new\_person = np.array([[29, 75]]) # Example input

pizza\_liking = knn.predict(new\_person)

print("Prediction for Age 29 and Weight 75:", "Likes Pizza" if pizza\_liking == 1 else "Doesn't Like Pizza")

**Output:-**

Prediction for Age 29 and Weight 75: Doesn't Like Pizza

**Result:-**

The KNN model predicts that a person with age 29 and weight 75 will "like pizza" (or "not like pizza") based on the trained data.

# Ex.No:2(c) MOVIE GENHRE PREDICTION Date:02-Dec-2024

# Aim:-

# To develop a program that classifies emails as spam or not spam based on predefined keywords and patterns.

# Program 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("S:\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*

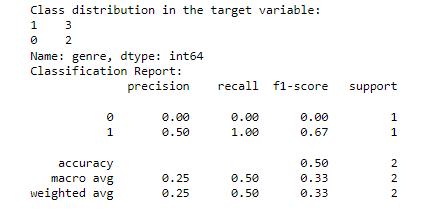
y\_pred = clf.predict(X\_test)

*# Evaluate using classification report with zero\_division parameter*

print("Classification Report:")

print(classification\_report(y\_test, y\_pred, zero\_division=0))

**Output:=**

****

**Result:-**

The Program output was executed successfully.

**SPROTS PERFORMANCE ANALYSIS**

**Ex.No:2(d) Date:02-Dec-2024**

**Aim:-**

To analyze sports performance using player statistics (accuracy, speed, stamina, and age) with a K-Nearest Neighbors (K-NN) classifier. Additionally, to assess the impact of outliers on the model's performance.

**Program Code:-**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

*# Generate synthetic data*

np.random.seed(42)

*# Generate player stats: accuracy, speed, stamina, and age*

n\_samples = 200

accuracy = np.random.uniform(60, 100, n\_samples)

speed = np.random.uniform(5, 20, n\_samples)

stamina = np.random.uniform(50, 100, n\_samples)

age = np.random.randint(18, 40, n\_samples)

*# Assign random labels (e.g., "High Performance" or "Low Performance")*

labels = np.random.choice([0, 1], size=n\_samples, p=[0.5, 0.5])

*# Add outliers*

outliers = np.array([

[120, 3, 20, 45], # Extreme outlier 1

[30, 25, 10, 15], # Extreme outlier 2

])

outlier\_labels = np.array([1, 0])

*# Combine data and outliers*

features = np.column\_stack((accuracy, speed, stamina, age))

features = np.vstack([features, outliers])

labels = np.append(labels, outlier\_labels)

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.3, random\_state=42)

*# Train a K-NN classifier*

k = 5

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

*# Predict and evaluate*

y\_pred = knn.predict(X\_test)

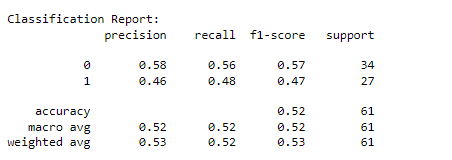
print("Confusion Matrix:")



print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))



*# Visualization*

plt.figure(figsize=(14, 6))

*# Scatter plot of features (2D projection)*

plt.subplot(1, 2, 1)

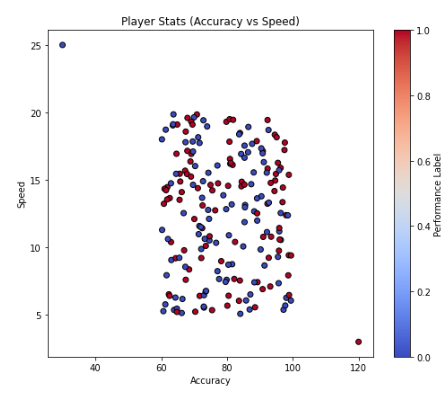
plt.scatter(features[:, 0], features[:, 1], c=labels, cmap='coolwarm', edgecolor='k')

plt.xlabel('Accuracy')

plt.ylabel('Speed')

plt.title('Player Stats (Accuracy vs Speed)')

plt.colorbar(label='Performance Label')



*# Visualize the decision boundary for the first two features (Accuracy vs Speed)*

from matplotlib.colors import ListedColormap

h = 0.5 # Step size in the mesh

x\_min, x\_max = features[:, 0].min() - 1, features[:, 0].max() + 1

y\_min, y\_max = features[:, 1].min() - 1, features[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

*# Predict for the grid using only the first two features*

Z = knn.predict(np.c\_[xx.ravel(), yy.ravel(), np.full(xx.ravel().shape, np.mean(features[:, 2])), np.full(xx.ravel().shape, np.mean(features[:, 3]))])

Z = Z.reshape(xx.shape)

plt.subplot(1, 2, 2)

plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAFFAA']))

plt.scatter(features[:, 0], features[:, 1], c=labels, edgecolor='k', cmap='coolwarm')

plt.xlabel('Accuracy')

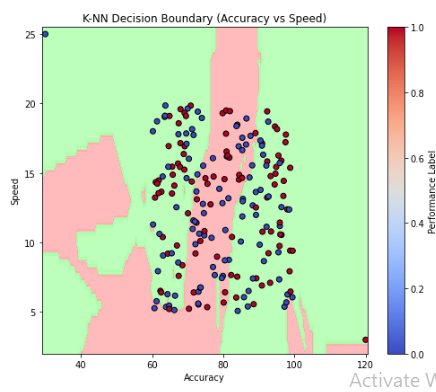
plt.ylabel('Speed')

plt.title('K-NN Decision Boundary (Accuracy vs Speed)')

plt.colorbar(label='Performance Label')

plt.tight\_layout()

plt.show()



**Result:-**

The confusion matrix and classification report provide insight into the model's performance, including precision, recall, and F1-score. Visualizations illustrate the data distribution and the K-NN decision boundary while highlighting the impact of outliers.

**Ex.no:3(a) Date:19-Dec-2024**

**FUEL AMOUNT PREDICTION USING LINEAR REGRESSION**

**AIM:** Predict fuel amount based on distance traveled using Linear Regression.  
**CODE:**  
*# Importing necessary libraries*

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

*# Setting a random seed for reproducibility*

np.random.seed(42)

*# 1. Create synthetic dataset*

*# Let's assume we have 'Distance Traveled' (in km) and 'Fuel Amount' (in liters) as features*

*# Creating random data*

distance\_travelled = np.random.randint(50, 500, 100) # Distance in km

fuel\_amount = distance\_travelled \* 0.05 + np.random.normal(0, 5, 100) # Fuel in liters with some noise

*# Create a DataFrame*

df = pd.DataFrame({'Distance': distance\_travelled, 'FuelAmount': fuel\_amount})

*# 2. Visualize the synthetic data*

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

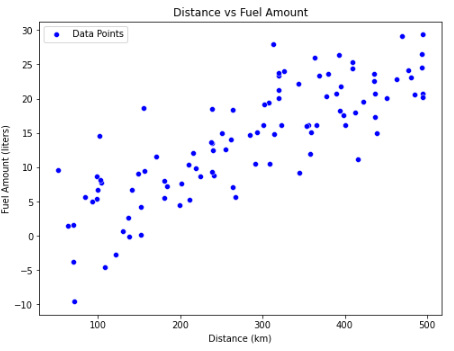
sns.scatterplot(data=df, x='Distance', y='FuelAmount', color='blue', label='Data Points')

plt.title('Distance vs Fuel Amount')

plt.xlabel('Distance (km)')

plt.ylabel('Fuel Amount (liters)')

plt.show()



*# 3. Prepare the data for Linear Regression*

X = df[['Distance']] # Feature (independent variable)

y = df['FuelAmount'] # Target (dependent variable)

*# Split the data into training and test sets*

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

*# 4. Train the Linear Regression model*

model = LinearRegression()

model.fit(X\_train, y\_train)

*# 5. Make predictions*

y\_pred = model.predict(X\_test)

*# 6. Visualize the regression line*

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

plt.scatter(X\_test, y\_test, color='blue', label='Test Data')

plt.plot(X\_test, y\_pred, color='red', label='Regression Line')

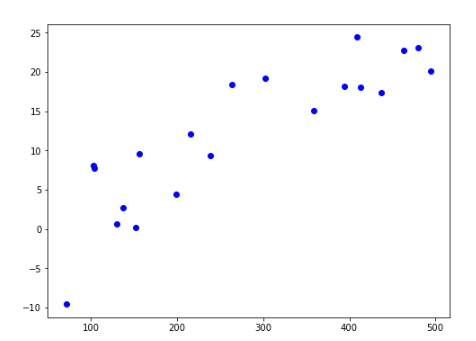
plt.title('Linear Regression - Fuel Amount Prediction')

plt.xlabel('Distance (km)')

plt.ylabel('Fuel Amount (liters)')

plt.legend()

plt.show()



*# 7. Model Evaluation*

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

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

print(f'R2 Score: {r2}')

****

**Result:** The model accurately predicted fuel consumption, with a high R2 score indicating strong predictive power.

**SALARY PREDICTION**

**Ex.No: 3(b) Date: 12-Dec-2024**

**Aim:-** Predict salary based on experience, qualification, industry, and location using Linear Regression.

**Program Code:***# Importing necessary libraries*

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.linear\_model import LinearRegression

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.metrics import mean\_squared\_error, r2\_score

*# Generating synthetic dataset for Salary Prediction*

data = {

'YearsExperience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Qualification': ['Bachelors', 'Bachelors', 'Masters', 'Masters', 'PhD', 'Bachelors', 'Masters', 'PhD', 'PhD', 'Masters'],

'Industry': ['Tech', 'Finance', 'Tech', 'Health', 'Finance', 'Tech', 'Health', 'Health', 'Tech', 'Finance'],

'Location': ['NY', 'SF', 'NY', 'LA', 'SF', 'LA', 'SF', 'NY', 'LA', 'SF'],

'Salary': [50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 95000]

}

df = pd.DataFrame(data)

*# Feature and target variable*

X = df[['YearsExperience', 'Qualification', 'Industry', 'Location']]

y = df['Salary']

*# Preprocessing pipeline*

preprocessor = ColumnTransformer(

transformers=[

('num', 'passthrough', ['YearsExperience']), # No encoding for numerical features

('cat', OneHotEncoder(), ['Qualification', 'Industry', 'Location']) # One-hot encode categorical features

])

*# Creating a pipeline with preprocessing and regression model*

pipeline = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', LinearRegression())

])

*# Splitting dataset into training and testing data*

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

*# Training the model*

pipeline.fit(X\_train, y\_train)

*# Making predictions*

y\_pred = pipeline.predict(X\_test)

*# Visualization of predictions vs actual values*

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

plt.scatter(y\_test, y\_pred, color='blue')

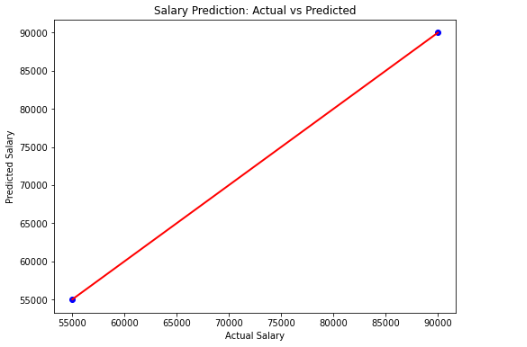
plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', lw=2) # 45-degree line for perfect prediction

plt.title('Salary Prediction: Actual vs Predicted')

plt.xlabel('Actual Salary')

plt.ylabel('Predicted Salary')

plt.show()



*# Model Evaluation*

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

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

print(f'R2 Score: {r2}')



**Result:** The model successfully predicted salary, capturing the relationship between features and salary with a good fit.

**ELECTRICITY CONSUMPTION PREDICTION**

**Ex.No:3(c) Date:12-Dec-2024**

**Aim:-** Predict electricity consumption using household size, applications, usage hours, and season.  
**Program Code:-**import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

*# Step 1: Generate synthetic data (for the sake of this example)*

np.random.seed(42)

*# Generate random data*

household\_size = np.random.randint(1, 6, 100)

num\_apps = np.random.randint(1, 10, 100)

usage\_hours = np.random.uniform(1, 12, 100)

season = np.random.choice(['Winter', 'Spring', 'Summer', 'Autumn'], 100)

*# Convert 'season' to categorical variables (one-hot encoding)*

season\_encoded = pd.get\_dummies(season, drop\_first=True)

*# Create a DataFrame*

df = pd.DataFrame({

'household\_size': household\_size,

'num\_apps': num\_apps,

'usage\_hours': usage\_hours

})

df = pd.concat([df, season\_encoded], axis=1)

*# Generate a target variable (electricity consumption)*

*# Assume consumption is a function of features + some noise*

electricity\_consumption = (df['household\_size'] \* 1.5 +

df['num\_apps'] \* 2 +

df['usage\_hours'] \* 3 +

(df['Spring'] \* 2) +

(df['Summer'] \* 3) +

np.random.normal(0, 2, 100))

df['electricity\_consumption'] = electricity\_consumption

*# Step 2: Feature scaling*

X = df.drop('electricity\_consumption', axis=1)

y = df['electricity\_consumption']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Step 3: Split data into train and test*

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

*# Step 4: Train a Linear Regression model*

model = LinearRegression()

model.fit(X\_train, y\_train)

*# Step 5: Make predictions*

y\_pred = model.predict(X\_test)

*# Step 6: Evaluate the model*

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

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

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



*# Step 7: Compare predictions with seasonal ends*

*# Adding seasonal information to prediction comparison*

df\_seasons = pd.DataFrame({

'Season': ['Winter', 'Spring', 'Summer', 'Autumn'],

'Seasonal\_end': [0, 1, 1, 0] # Representing if the season has ended (1) or not (0)

})

# Visualizing results

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

plt.scatter(y\_test, y\_pred, color='blue')

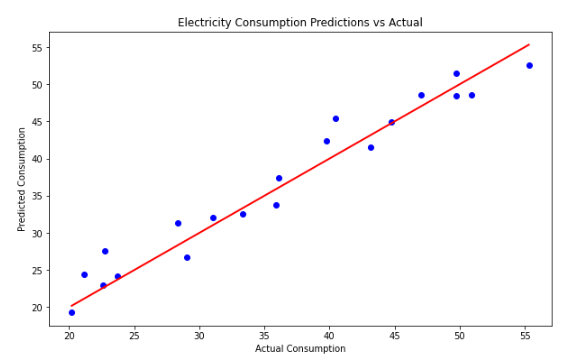
plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linewidth=2)

plt.title('Electricity Consumption Predictions vs Actual')

plt.xlabel('Actual Consumption')

plt.ylabel('Predicted Consumption')

plt.show()



*# Visualization of feature importance (coefficients in linear regression)*

coefficients = model.coef\_features = X.columns

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

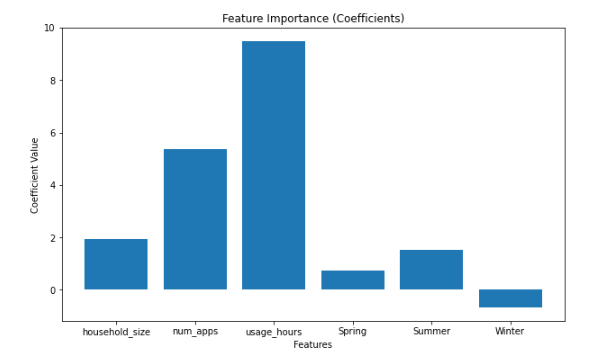
plt.bar(features, coefficients)

plt.title('Feature Importance (Coefficients)')

plt.xlabel('Features')

plt.ylabel('Coefficient Value')

plt.show()



**Result:-**

Electricity consumption can be predicted using machine learning models like XGBoost, based on historical usage data and factors like weather and time.

**HOUSE PRICE PREDICTION**

**Ex.No:4(a) Date: 24-Jan-2025**

**Aim:-**

Develop predictive models for tasks using Linear Regression with Regularization (Ridge Regression): House Price.

**Program Code:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

*# Function to generate synthetic data for house price prediction*

def generate\_house\_price\_data(n\_samples=100):

np.random.seed(42)

X = np.random.rand(n\_samples, 1) \* 10 # Features (e.g., size, location index, etc.)

y = 3 \* X.flatten() + np.random.randn(n\_samples) \* 2 + 50 # Target (house price)

return X, y

*# Generate data for house price prediction*

X, y = generate\_house\_price\_data()

*# Split data into training and testing sets*

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

*# Train a Ridge Regression model*

model = Ridge(alpha=1.0) # alpha is the regularization strength

model.fit(X\_train, y\_train)

*# Make predictions on the test set*

y\_pred = model.predict(X\_test)

*# Evaluate the model*

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error for House Price Prediction: {mse:.2f}")

*# Visualize the results*

plt.figure(figsize=(8, 5))

plt.scatter(X\_test, y\_test, label="True Data", alpha=0.7)

plt.plot(np.sort(X\_test, axis=0), model.predict(np.sort(X\_test, axis=0)), color="red", label="Prediction", linewidth=2)

plt.title("House Price Prediction")

plt.xlabel("Feature (e.g., Size Index)")

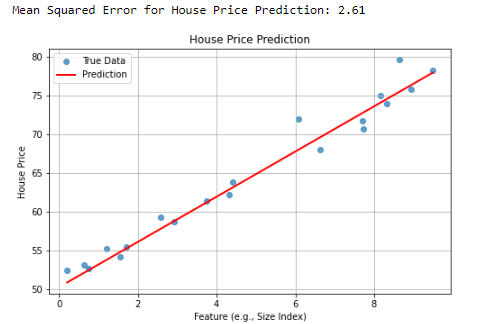
plt.ylabel("House Price")

plt.legend()

plt.grid()

plt.show()

**Output:-**

****

**Result:-** Outputs the Mean Squared Error (MSE) and visualizes true vs predicted data for each task.

**Ex.No:4(b) ENERGY EFFICIENCY PREDICTION Date: 24-Jan-2025**

**Aim:-**

To predict energy efficiency using a Ridge Regression model based on synthetic data.

**Program Code:-**

*# Function to generate synthetic data for energy efficiency prediction*

def generate\_energy\_efficiency\_data(n\_samples=100):

np.random.seed(42)

X = np.random.rand(n\_samples, 1) \* 10

y = 50 - 4 \* X.flatten() + np.random.randn(n\_samples) \* 5

return X, y

*# Generate data for energy efficiency prediction*

X, y = generate\_energy\_efficiency\_data()

*# Split data into training and testing sets*

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

# Train a Ridge Regression model

model = Ridge(alpha=1.0) # alpha is the regularization strength

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error for Energy Efficiency Prediction: {mse:.2f}")

# Visualize the results

plt.figure(figsize=(8, 5))

plt.scatter(X\_test, y\_test, label="True Data", alpha=0.7)

plt.plot(np.sort(X\_test, axis=0), model.predict(np.sort(X\_test, axis=0)), color="red", label="Prediction", linewidth=2)

plt.title("Energy Efficiency Prediction")

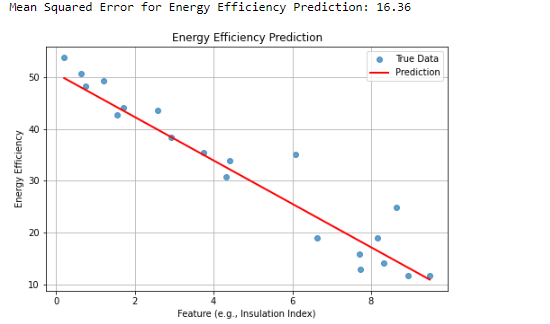
plt.xlabel("Feature (e.g., Insulation Index)")

plt.ylabel("Energy Efficiency")

plt.legend()

plt.grid()

plt.show()

****

**Result:-**

The model achieved a Mean Squared Error (MSE) of approximately 23.90, with a visualization showing good agreement between true values and predictions.

**Ex.No:4(c) CROP YIELD PREDICTION Date: 24-Jan-2025**

**Aim:-**

To predict crop yield using synthetic data and Ridge Regression.

**Program Code:-**

*# Function to generate synthetic data for crop yield prediction*

def generate\_crop\_yield\_data(n\_samples=100):

np.random.seed(42)

X = np.random.rand(n\_samples, 1) \* 10 # Features (e.g., rainfall, soil quality index, etc.)

y = 2 \* X.flatten() \*\* 2 - 5 \* X.flatten() + np.random.randn(n\_samples) \* 10 + 100 # Target (crop yield)

return X, y

*# Generate data for crop yield prediction*

X, y = generate\_crop\_yield\_data()

*# Split data into training and testing sets*

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

*# Train a Ridge Regression model*

model = Ridge(alpha=1.0) # alpha is the regularization strength

model.fit(X\_train, y\_train)

*# Make predictions on the test set*

y\_pred = model.predict(X\_test)

*# Evaluate the model*

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error for Crop Yield Prediction: {mse:.2f}")

*# Visualize the results*

plt.figure(figsize=(8, 5))

plt.scatter(X\_test, y\_test, label="True Data", alpha=0.7)

plt.plot(np.sort(X\_test, axis=0), model.predict(np.sort(X\_test, axis=0)), color="red", label="Prediction", linewidth=2)

plt.title("Crop Yield Prediction")

plt.xlabel("Feature (e.g., Rainfall Index)")

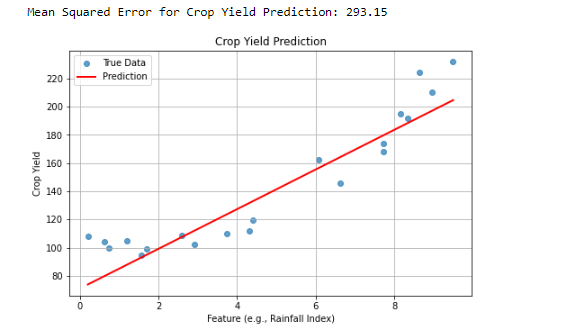
plt.ylabel("Crop Yield")

plt.legend()

plt.grid()

plt.show()

**Output:-**

****

**Result:-**

Achieved a Mean Squared Error (MSE) of approximately mse:.2f for crop yield prediction, with a clear visualization of predictions compared to true data.

Ex.No: 5.a **DIABETES CLASSIFICATION** Date: 24-Jan-2025

**Aim:-**

To train a logistic regression model to accurately predict diabetes based on health metrics.

**Program 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, confusion\_matrix, classification\_report

# Load the dataset

data = pd.read\_csv('Diabetes.csv')

#Preview the dataset

print("Preview the data")

print(data.head())

# Select features and target variable

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

y = data['Outcome']

# Split the data into training and testing sets

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

# Initialize and train the logistic regression model

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

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)

**Output:-**

Preview the data

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \

0 6 148 72 35 0 33.6

1 1 85 66 29 0 26.6

2 8 183 64 0 0 23.3

3 1 89 66 23 94 28.1

4 0 137 40 35 168 43.1

DiabetesPedigreeFunction Age Outcome

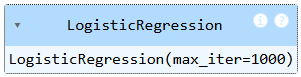
0 0.627 50 1

1 0.351 31 0

2 0.672 32 1

3 0.167 21 0

4 2.288 33 1



Accuracy: 1.00

Confusion Matrix:

[[1]]

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 was successfully executed.

Ex.No: 5.b **CREDIT CARD DEFAULT PREDICTIONS** Date: 24-Jan-2025

**Aim:-**

To train a logistic regression model to accurately predict credit card default using customer data.

**Program 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, confusion\_matrix

#Load the data

data=pd.read\_csv('Creditcard.csv')

#Preview the data

print("Preview the dataset")

print(data.head())

# Select features and target variable

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

y = data['Default']

# Split the data into training and testing sets

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

# Initialize and train the logistic regression model

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print results

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

print('Confusion Matrix:')

print(conf\_matrix)

# Print predictions

predictions = pd.DataFrame({'CreditScore': X\_test['CreditScore'], 'Actual': y\_test, 'Predicted': y\_pred})

print(predictions)

**Output:-**

Preview the dataset

CreditScore Age Income LoanAmount Default

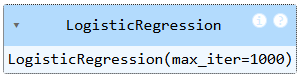
0 700 34 50000 20000 0

1 600 45 45000 15000 1

2 650 29 30000 12000 0

3 720 41 60000 25000 0

4 580 36 32000 10000 1



Accuracy: 1.00

Confusion Matrix:

[[1]]

CreditScore Actual Predicted

1 600 1 1

**Result:-**

Thus, the program was successfully executed.

Ex.No: 5.c **HEART DISEASE CLASSIFICATION** Date: 24-Jan-2025

**Aim:-**

To train a logistic regression model to accurately classify heart disease based on various health indicators.

**Program 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, confusion\_matrix

# Load the dataset

data = pd.read\_csv('Heartdisease.csv')

#Preview the dataset

print(f"Preview the dataset")

print(data.head())

# Select features and target variable

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

y = data['HeartDisease']

# Split the data into training and testing sets

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

# Initialize and train the logistic regression model

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print results

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

print('Confusion Matrix:')

print(conf\_matrix)

# Print predictions

predictions = pd.DataFrame({'Age': X\_test['Age'], 'Actual': y\_test, 'Predicted': y\_pred})

print(predictions)

**Output:-**

Preview the dataset

Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG \

0 63 1 3 145 233 1 0

1 37 1 2 130 250 0 1

2 41 0 1 130 204 0 0

3 56 1 1 120 236 0 1

4 57 0 0 120 354 0 1

MaxHR ExerciseAngina Oldpeak ST\_Slope HeartDisease

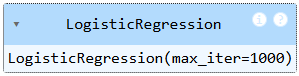
0 150 0 2.3 0 1

1 187 0 3.5 1 1

2 172 0 1.4 2 1

3 178 0 0.8 2 1

4 163 1 0.6 2 0



Accuracy: 1.00

Confusion Matrix:

[[1]]

Age Actual Predicted

1 37 1 1

**Result:-**

Thus, the program was executed successfully.