PRODUCT DEMAND PREDICTION WITH MACHINE LEARNING

FEATURE ENGINEERING

1.Data Collection:

•Gather historical data on product demand. The dataset should contain information such as date, product attributes, sales quantity, price, promotions, and any other relevant data.

2. Data Preprocessing:

•Handle missing values, outliers, and format the data appropriately. Ensure that your data is in a structured format that can be used for analysis.

3. Time-Series Features:

•*Extract relevant date-time features, such as day of the week, month, quarter, and year. Also, create lag features to capture historical demand patterns (e.g., demand from the previous day, week, or month).*

```
# Example: Creating lag features

data['lag_1'] = data['sales_quantity'].shift(1)

data['lag_7'] = data['sales_quantity'].shift(7)
```

4. Rolling Statistics:

- *Calculate rolling statistics like moving averages to capture trends and seasonality.*

```
# Example: 7-day rolling average
data['7-day_avg'] =
data['sales_quantity'].rolling(window=7).mean()
```

5. Price and Promotion Features:

- *Include information on price changes and promotion periods as binary flags or numeric variables.*

```
# Example: Creating a binary flag for promotions
data['promotion_flag'] = (data['promotion'] > 0).astype(int)
```

6. Categorical Features:

•*Encode categorical variables like product categories, store locations, or brands using techniques like one-hot encoding or label encoding.*

7. External Data:

•*Incorporate external data that might influence demand, such as economic indicators, weather data, or social media mentions.*

8. Feature Scaling:

•*Scale numerical features to have similar ranges using methods like Min-Max scaling or Z-score normalization.*

9. Data Splitting:

•*Split the data into training, validation, and test sets while maintaining the time order. The training set should cover earlier time periods, and the test set should cover the most recent periods.*

10. Model Selection: *- Choose an appropriate machine learning model for demand prediction. Time series models (e.g., ARIMA, Prophet) or regression models (e.g., Linear Regression, Random Forest, XGBoost) are common choices.*

11. Model Training: *- Train the selected model using the training dataset. Be sure to provide the features you've engineered.*

12. Model Evaluation: -

- *Evaluate the model's performance on the validation or test set using relevant metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R-squared.*

13. Hyperparameter Tuning: -

- *Fine-tune the model's hyperparameters to optimize its performance.*

14. Predictions and Deployment: -

- *Use the trained model to make demand predictions for future time periods. Deploy the model in your business operations for real-time or batch forecasting.*

15. Monitoring and Updating: -

- *Continuously monitor the model's performance and update it as new data becomes available.*

Coding:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

data = pd.read_csv('productdemand.csv')
```

```python
# Create a sample date column for demonstration
data['date'] = pd.date_range(start='2023-01-01',
periods=len(data), freq='D')
# Extract and assign date features
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day_of_week'] = data['date'].dt.dayofweek
data['day_of_month'] = data['date'].dt.day
# Print the updated data
print(data)
```

OUTPUT:

```
   ID  Store ID  Total Price  Base Price  Units Sold        date  year  month  \
0   1      8091      99.0375    111.8625          20  2023-01-01  2023      1
1   2      8091      99.0375     99.0375          28  2023-01-02  2023      1
2   3      8091     133.9500    133.9500          19  2023-01-03  2023      1
3   4      8091     133.9500    133.9500          44  2023-01-04  2023      1
4   5      8091     141.0750    141.0750          52  2023-01-05  2023      1

   day_of_week  day_of_month  Price Difference
0            6             1            12.825
1            0             2             0.000
2            1             3             0.000
3            2             4             0.000
4            3             5             0.000
```

 # Calculate the price difference

data['Price Difference'] = data['Base Price'] - data['Total Price']

 # Check the updated DataFrame

print(data)

OUTPUT:

```
   ID  Store ID  Total Price  Base Price  Units Sold       date  year  month  \
0   1      8091      99.0375    111.8625          20 2023-01-01  2023      1
1   2      8091      99.0375     99.0375          28 2023-01-02  2023      1
2   3      8091     133.9500    133.9500          19 2023-01-03  2023      1
3   4      8091     133.9500    133.9500          44 2023-01-04  2023      1
4   5      8091     141.0750    141.0750          52 2023-01-05  2023      1

   day_of_week  day_of_month  Price Difference
0            6             1            12.825
1            0             2             0.000
2            1             3             0.000
3            2             4             0.000
4            3             5             0.000
```

```
# Calculate store-level statistics
store_stats = data.groupby('Store ID').agg({'Total Price': ['mean', 'median', 'std'], 'Units Sold': 'sum’})
store_stats.columns = ['store_mean_price', 'store_median_price', 'store_price_std', 'store_total_units_sold']
# Merge store-level statistics back into the original dataset
data = data.merge(store_stats, on='Store ID', how='left')
# Print the dataset with store-level statistics
print(data)
```

OUTPUT:

	ID	Store ID	Total Price	Base Price	Units Sold	store_mean_price_x
0	1	8091	99.0375	111.8625	20	121.41
1	2	8091	99.0375	99.0375	28	121.41
2	3	8091	133.9500	133.9500	19	121.41
3	4	8091	133.9500	133.9500	44	121.41
4	5	8091	141.0750	141.0750	52	121.41

	store_median_price_x	store_price_std_x	store_total_units_sold_x
0	133.95	20.629305	163
1	133.95	20.629305	163
2	133.95	20.629305	163
3	133.95	20.629305	163
4	133.95	20.629305	163

	store_mean_price_y	store_median_price_y	store_price_std_y
0	121.41	133.95	20.629305
1	121.41	133.95	20.629305
2	121.41	133.95	20.629305
3	121.41	133.95	20.629305
4	121.41	133.95	20.629305

	store_total_units_sold_y
0	163
1	163
2	163
3	163
4	163

MODEL TRAINING:

- *Train the selected model on the training dataset.*

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from statsmodels.tsa.arima.model import ARIMA
from fbprophet import Prophet
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# Example with a Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X_train, y_train)
```

```
#model training

# Split the data into features and target

X = df[["Store ID", "Total Price", "Base Price"]]

y = df["Units Sold"]

# 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)
```

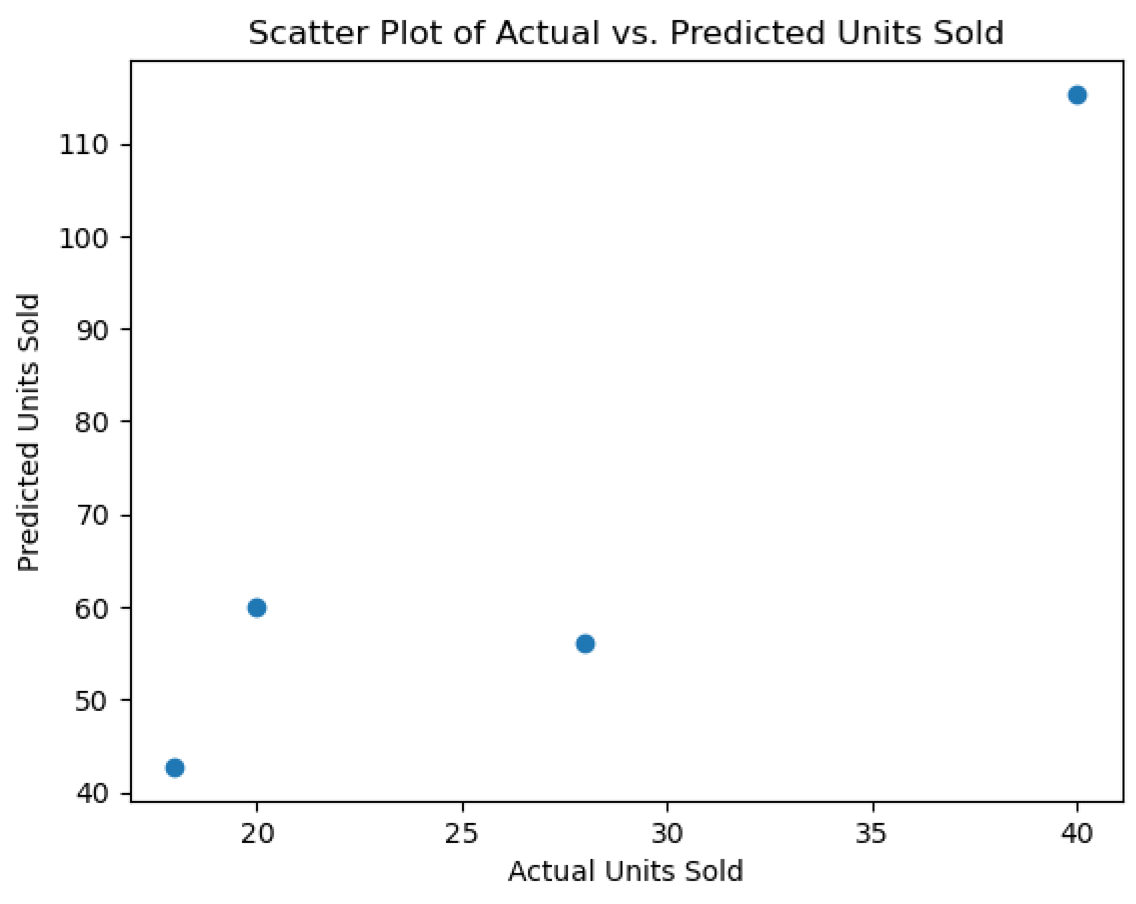
```python
# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Print the model coefficients
print("Model Coefficients:")
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
# Print the testing set
print("\nTesting Set:")
print(X_test)
```

OUTPUT:

```
Model Coefficients:
Intercept: -119723.07317719368
Coefficients: [14.80526018 -0.42193316  0.31835396]

Testing Set:
    Store ID  Total Price  Base Price
0       8091      99.0375    111.8625
5       8091     227.2875    227.2875
11      8095      98.3250     98.3250
1       8091      99.0375     99.0375
```

```python
# Visualize the scatter plot
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Units Sold")
plt.ylabel("Predicted Units Sold")
plt.title("Scatter Plot of Actual vs. Predicted Units Sold")
plt.show()
```

OUTPUT:


Scatter Plot of Actual vs. Predicted Units Sold
Predicted Units Sold
110
100
90
80
70
60
50
40
20
25
30
35
40
Actual Units Sold

MODEL EVALUATION:

```
df = pd.DataFrame(data)

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

X = df[["Store ID", "Total Price", "Base Price"]]

y = df["Units Sold"]

# Split the data into a training set and a testing set

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

# Initialize and train a linear regression model

model = LinearRegression()

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)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

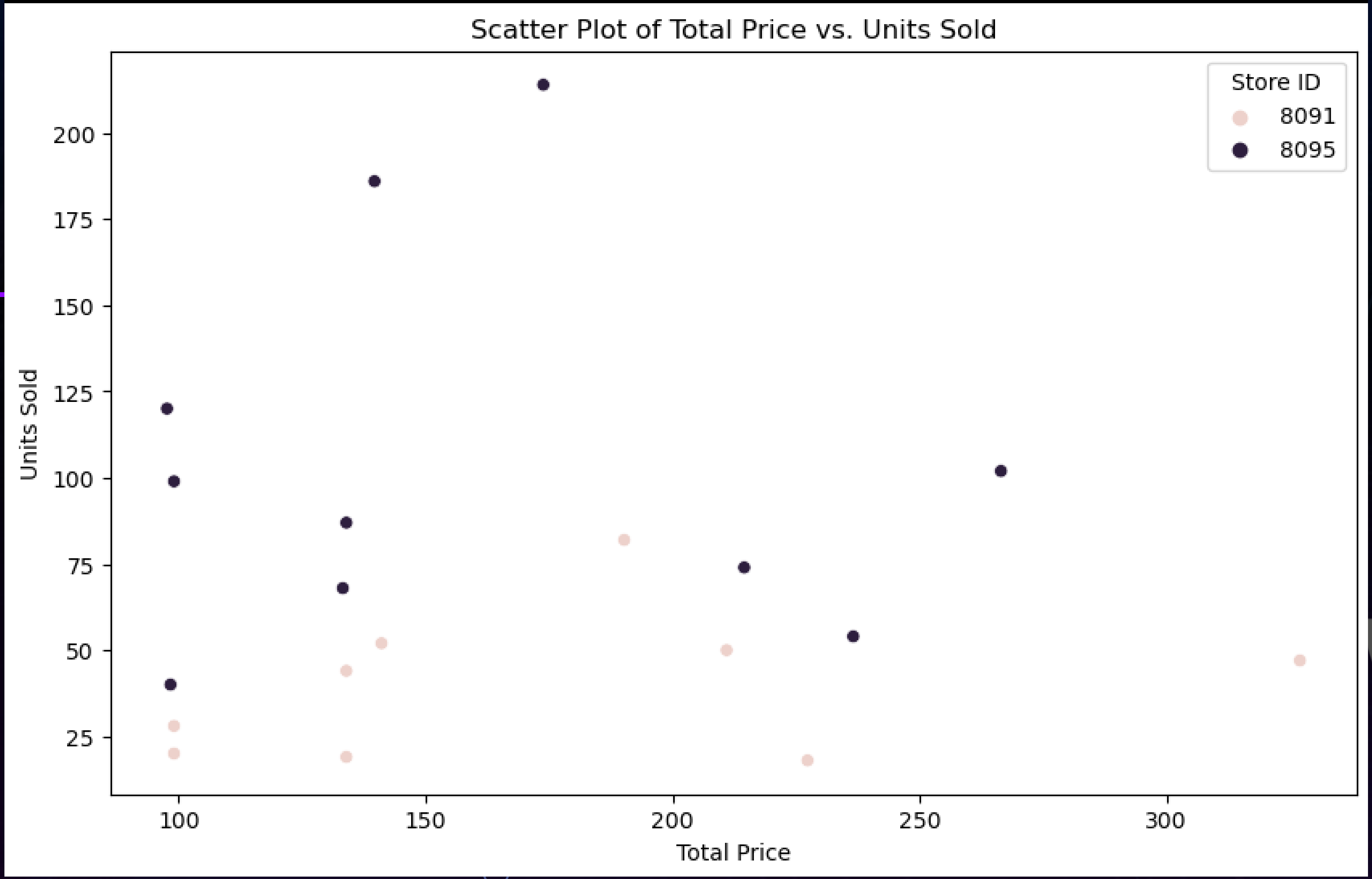
OUTPUT:

```
Mean Squared Error: 2170.1240553918774
R-squared: -28.031759938352874
```

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read_csv('productdemand.csv')

# Print the dataset

print("Dataset:")

print(data)

# Create a scatter plot of Total Price vs. Units Sold

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

sns.scatterplot(data=df, x="Total Price", y="Units Sold",hue="Store ID")

plt.title("Scatter Plot of Total Price vs. Units Sold")

plt.xlabel("Total Price")

plt.ylabel("Units Sold")

plt.show()
```

OUTPUT:

```
...  Dataset:
              ID  Store ID  Total Price  Base Price  Units Sold
     0         1      8091      99.0375    111.8625          20
     1         2      8091      99.0375     99.0375          28
     2         3      8091     133.9500    133.9500          19
     3         4      8091     133.9500    133.9500          44
     4         5      8091     141.0750    141.0750          52
     ...     ...       ...          ...         ...         ...
     150145  212638   9984     235.8375    235.8375          38
     150146  212639   9984     235.8375    235.8375          30
     150147  212642   9984     357.6750    483.7875          31
     150148  212643   9984     141.7875    191.6625          12
     150149  212644   9984     234.4125    234.4125          15

     [150150 rows x 5 columns]
```



Scatter Plot of Total Price vs. Units Sold
Store ID
8091
8095
Units Sold
200
175
150
125
100
75
50
25
100
150
200
250
300
Total Price