PRODUCT DEMAND PREDICTION

PHASE – 4

Development Phase Part – 2

Sanjay Kadavarath Ajayakumar

(Team Leader)

Explanation for Development part – 1

Step 1: Importing the required libraries and loading the dataset

[1]:
```
#importing necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

#importing the netflix dataset
file_path = r"C:\Users\Saranya\Downloads\archive (1)\PoductDemand.csv"
encoding = "ISO-8859-1"
df = pd.read_csv(file_path, encoding=encoding)
df
```

[1]: ,,,

	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

Step 2: Handling Missing Data

[4]:
```
#to display null values
df.isnull()
```

[4]: ,,,

	ID	Store ID	Total Price	Base Price	Units Sold
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
150145	False	False	False	False	False
150146	False	False	False	False	False
150147	False	False	False	False	False
150148	False	False	False	False	False
150149	False	False	False	False	False

,

150150 rows × 5 columns

,

[5]:
```
#handling null values

df.fillna(df.mean(), inplace=True)
df.dropna(inplace=True)
```

Step 3: Label encoder for Total Price Column

[6]:

```
label_encoder = LabelEncoder()
df['Store ID'] = label_encoder.fit_transform(df['Store ID'])
df
```

[6]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	3	99.0375	111.8625	20
1	2	3	99.0375	99.0375	28
2	3	3	133.9500	133.9500	19
3	4	3	133.9500	133.9500	44
4	5	3	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	75	235.8375	235.8375	38
150146	212639	75	235.8375	235.8375	30
150147	212642	75	357.6750	483.7875	31
150148	212643	75	141.7875	191.6625	12
150149	212644	75	234.4125	234.4125	15

150150 rows × 5 columns

Step 4: Feature Scaling using StandardScaler

[8]:

```
#scaling

scaler = StandardScaler()
df['Base Price'] = scaler.fit_transform(df['Base Price'].values.reshape(-1, 1))
df
```

[8]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	3	-1.041443	-0.969377	20
1	2	3	-1.041443	-1.084958	28
2	3	3	-0.703497	-0.770322	19
3	4	3	-0.703497	-0.770322	44
4	5	3	-0.634528	-0.706110	52
...	...	...	...	...	...
150145	212638	75	0.282754	0.147904	38
150146	212639	75	0.282754	0.147904	30
150147	212642	75	1.462118	2.382466	31
150148	212643	75	-0.627632	-0.250208	12
150149	212644	75	0.268961	0.135061	15

[7]:

```
#scaling

scaler = StandardScaler()
df['Total Price'] = scaler.fit_transform(df['Total Price'].values.reshape(-1, 1))
df
```

[7]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	3	-1.041443	111.8625	20
1	2	3	-1.041443	99.0375	28
2	3	3	-0.703497	133.9500	19
3	4	3	-0.703497	133.9500	44
4	5	3	-0.634528	141.0750	52
...	...	...	...	...	...
150145	212638	75	0.282754	235.8375	38
150146	212639	75	0.282754	235.8375	30
150147	212642	75	1.462118	483.7875	31
150148	212643	75	-0.627632	191.6625	12
150149	212644	75	0.268961	234.4125	15

Step 5: Splitting the data into a training set and a test

```
[9]: #train_test split

X = df.drop('Units Sold', axis=1)
y = df['Units Sold']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[10]: print("\n X_test info")
print(X_test.info())
```

```
 X_test info
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30030 entries, 144782 to 110483
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           30030 non-null  int64
 1   Store ID     30030 non-null  int64
 2   Total Price  30030 non-null  float64
 3   Base Price   30030 non-null  float64
dtypes: float64(2), int64(2)
memory usage: 1.1 MB
None
```

Feature Scaling, Model Training, and Evaluation Algorithm for Product Demand Prediction

Objective:

This algorithm aims to guide the development of a predictive model for product demand prediction using the provided dataset. It covers essential steps, including feature engineering, model training, and evaluation, to ensure accurate predictions.

Steps:

1. Load and Preprocess the Dataset:

- Load the dataset, which includes information on products such as ID, StoreID, Total Price, Base Price, Units Sold.
- Ensure that you understand the dataset's structure and contents.

2. Feature Engineering:

- Review the dataset to identify which features will be used for predicting the demand for products.
- Handle any missing data. It appears that the dataset does not have any missing values.
- Encode categorical data, using techniques like label encoding or one-hot encoding to convert them into a numerical format.

3. Feature Scaling

- Analyze the dataset and determine if feature scaling is required. Some machine learning algorithms benefit from scaled features.
- If needed, apply feature scaling to numerical features. For example, you can use standardization to scale the feature.

4. Split the Dataset:

- Split the dataset into training and testing sets to assess the model's performance.
- A common split ratio is 80% for training and 20% for testing. Ensure that the split is random to avoid any potential biases.

5. Select a Machine Learning Model:

- -Choose an appropriate machine learning model for regression tasks.

6. Train the Model:

- Initialize the chosen model.
- Fit the model to the training data, using the selected features as input and IMDb scores as the target variable.
- During training, the model will learn patterns in the data.

7. Make Predictions:

- Utilize the trained model to make IMDb score predictions on the testing data.
- The model predicts IMDb scores based on the test feature data.

8. Evaluate the Model:

- Assess the model's performance using regression evaluation metrics. Common metrics include:
- Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual price.
- Mean Squared Error (MSE): Measures the average of the squared differences between predicted and actual score
- Root Mean Squared Error (RMSE): The square root of MSE, providing error in the original score units.
- R-squared (R2): Measures the proportion of the variance in IMDb scores explained by the model.
- Visualize the results, such as scatter plots comparing actual IMDb scores vs. predicted scores or distribution plots.

This algorithm provides a structured approach to developing a product demand prediction model specifically tailored to the dataset.

Execution of the model:

Importing the necessary libraries:

```
In [28]: # Import necessary libraries for model training and evaluation
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Train test split:

```
In [29]: # Split the dataset 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)
```

Linear Model:

```
In [30]: # Initialize the Linear Regression model
         model = LinearRegression()

         # Train the model on the training data
         model.fit(X_train, y_train)

         # Make predictions on the test data
         y_pred = model.predict(X_test)
```

Random Forest:

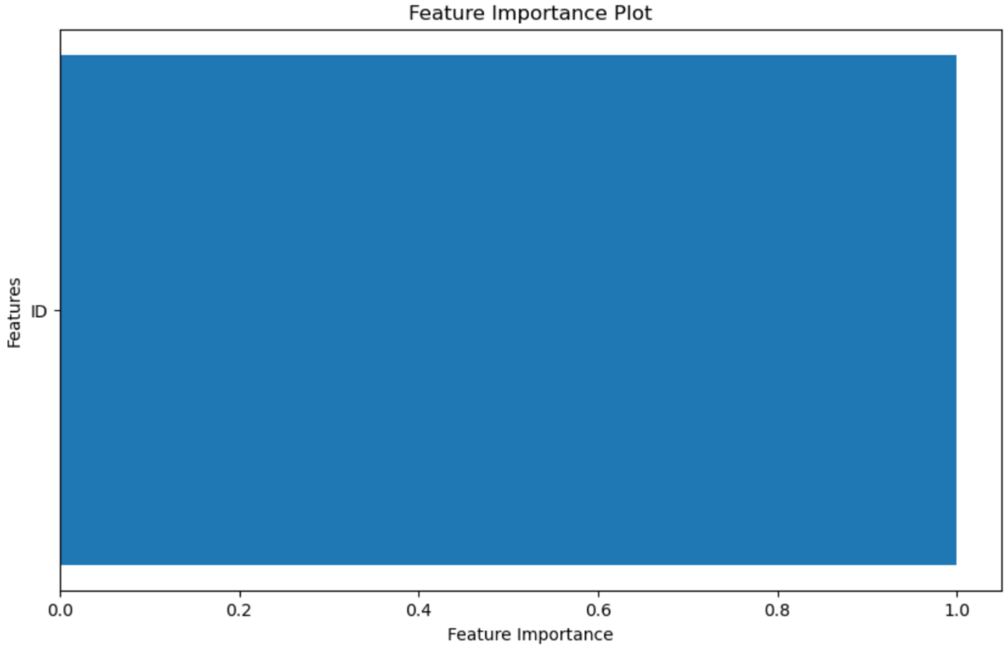
```
In [49]: from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
         import matplotlib.pyplot as plt
         import seaborn as sns
```

```
In [50]: X = df.drop(["Store ID", "Total Price", "Base Price", "Units Sold"], axis=1)
         y = df["Units Sold"]

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

```
In [51]: model = RandomForestRegressor(random_state=42)
         model.fit(X_train, y_train)

Out[51]:
         RandomForestRegressor
         RandomForestRegressor(random_state=42)
```

Feature importance plot for Random Forest

```
In [52]: if isinstance(model, RandomForestRegressor):
             feature_importance = model.feature_importances_
             feature_names = X_train.columns
             plt.figure(figsize=(10, 6))
             plt.barh(feature_names, feature_importance)
             plt.xlabel("Feature Importance")
             plt.ylabel("Features")
             plt.title("Feature Importance Plot")
             plt.show()
```



Evaluating the Model:

Using MAE, MSE, RMSE and R2

```
In [45]: # Evaluate the model
         mae = mean_absolute_error(y_test, y_pred)
         mse = mean_squared_error(y_test, y_pred)
         rmse = mean_squared_error(y_test, y_pred, squared=False)
         r2 = r2_score(y_test, y_pred)

         print(f"Mean Absolute Error (MAE): {mae}")
         print(f"Mean Squared Error (MSE): {mse}")
         print(f"Root Mean Squared Error (RMSE): {rmse}")
         print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 35.00456348618878
Mean Squared Error (MSE): 3281.7516859029047
Root Mean Squared Error (RMSE): 57.28657509314817
R-squared (R2): -0.00019064320674355706
```

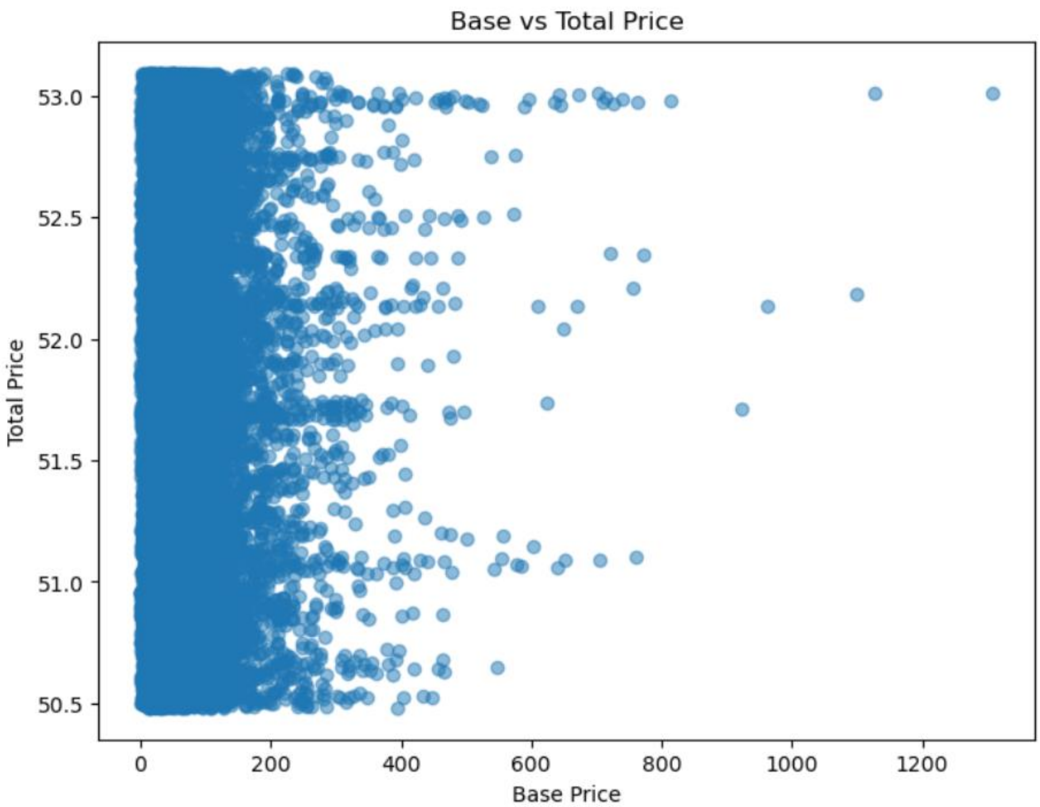
Visualization of the result:

Importing the libraries:

```
In [46]: import matplotlib.pyplot as plt
         import seaborn as sns
```

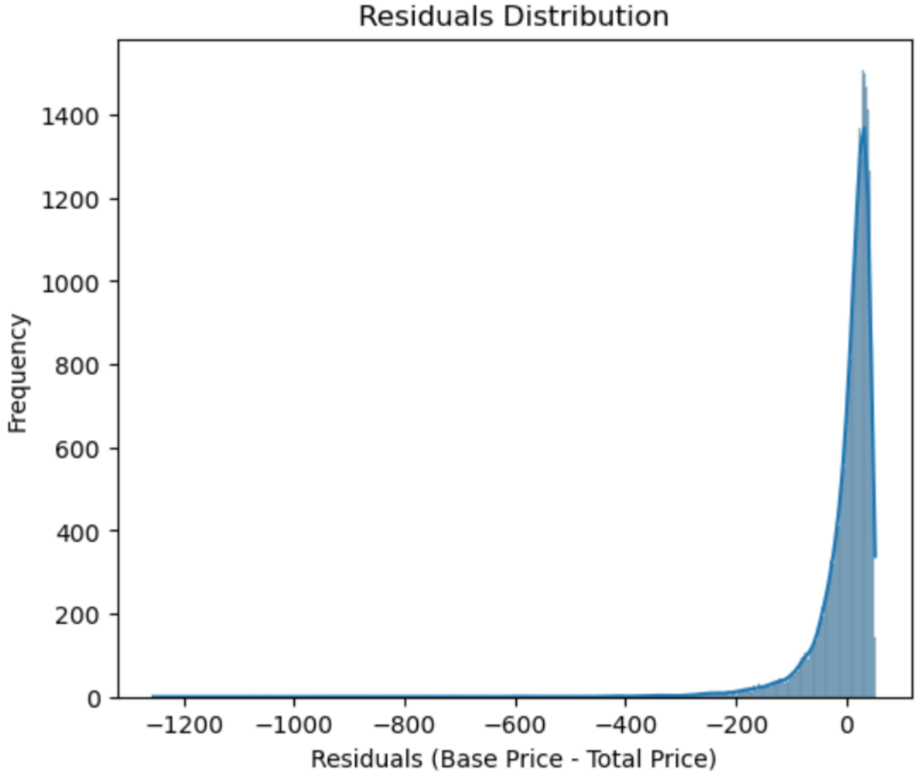
Price vs Total Price:

```
In [47]: # Scatter plot of actual IMDb scores vs. predicted IMDb scores
         plt.figure(figsize=(8, 6))
         plt.scatter(y_test, y_pred, alpha=0.5)
         plt.xlabel("Base Price")
         plt.ylabel("Total Price")
         plt.title("Base vs Total Price")
         plt.show()
```



Residual plot:

```
In [48]: # Distribution plot of the residuals (predicted - actual IMDb scores)
residuals = y_pred - y_test
plt.figure(figsize=(6, 5))
sns.histplot(residuals, kde=True)
plt.xlabel("Residuals (Base Price - Total Price)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
plt.show()
```



In this phase, we embarked on the journey of building a product demand prediction model for the dataset of units sold. We began by loading and preprocessing the dataset, which included handling missing data, encoding categorical features, and scaling numerical attributes.

Our model selection led us to a Random Forest Regressor, which has the advantage of capturing complex relationships within the data. After training the model, we evaluated its performance using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the R-squared (R2) coefficient. These metrics allowed us to assess the accuracy of our predictions.

Visualizations, including feature importance plots, residual plot and scatter plot provided additional insights into the model's performance. This comprehensive process equipped us with a powerful tool for predicting product demand, which can be invaluable for retailers, wholesalers and other businesses in assessing the potential success of the company.