Project 10: Product demand prediction with machine learnings

1.PAVITHRA M - 211521104103 (Team Leader)

2.BALASANDHIYA M - 211521104019

3.BENITA SHARON G - 211521104020

4.JIVITHA M - 211521104064

5.RITHIKA S - 211521104126

Phase 2: Innovation

Problem Definition

The core problem we aim to address is predicting product demand based on historical sales data and related features. Accurate demand prediction is crucial for efficient inventory management and planning. By leveraging machine learning techniques, we intend to build a model that forecasts future product demand, allowing businesses to optimize their operations.

Objective

Our primary objective is to develop a robust predictive model that accurately forecasts product demand. The model will utilize features such as historical sales, pricing information, and store details to generate predictions. By achieving accurate demand forecasts, we aim to assist businesses in making informed decisions regarding inventory, pricing strategies, and resource allocation.

This documentation will elaborate on the steps planned to enhance the product demand prediction model. The goal is to improve accuracy and efficiency in forecasting product demand by utilizing innovative methodologies and leveraging the provided dataset's columns, including ID, Store ID, Base Price, Total Price, and Unit Sold.

Dataset Overview

The dataset used for this project is sourced from Kaggle, a popular platform for sharing and discovering datasets related to various domains. The dataset titled "Product Demand Prediction with Machine Learning" provides essential information necessary for predicting product demand, including data columns such as ID, Store ID, Base Price, Total Price, and Unit Sold.

- **Dataset Title**: Product Demand Prediction with Machine Learning
- **Source**: Kaggle
- **Dataset Link**: https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning

The dataset we are working with contains crucial information related to product demand. The provided columns are as follows:

Details about columns (columns which you gonna use)

1. **Store ID:** This column can help you differentiate and analyze product demand by store location. It's useful if you have multiple stores and want to understand demand variations across different locations.
2. **Total Price:** The total price of products sold. This can be a valuable feature as it can influence product demand. Seasonal variations in prices can be important for forecasting.
3. **Base Price:** The base price of products. Base price fluctuations can affect demand, and including this information can help identify price-demand relationships.
4. **Units Sold:** This is your target variable, representing the number of units of a product sold during a specific time period.

This dataset serves as the foundation for our product demand prediction model, enabling us to analyze historical sales data and forecast future demand accurately.

Details of libraries to be used and way to download

For this project, we will utilize several Python libraries for data analysis, machine learning, and time series forecasting. Below are the libraries used:

- **Pandas:** Used for data manipulation and analysis.
- **NumPy:** Provides support for numerical operations and array manipulation.
- **Scikit-Learn:** Used for machine learning tasks.
- **Matplotlib:** A popular library for data visualization.
- **Seaborn:** Another visualization library for enhancing data presentation.

```
!pip install pandas
!pip install numpy
!pip install scikit-learn
!pip install matplotlib
!pip install seaborn
```

How to train and test

To evaluate model performance, we adopt a common practice in machine learning, where we split the dataset into two subsets: the training set and the testing set. This split is essential to ensure the model's performance is accurately assessed and to prevent overfitting.

- **Training Set (80%):** The training set comprises 80% of the dataset. This larger portion is used to train the machine learning model. During training, the model learns patterns and relationships within the data, allowing it to make predictions.
- **Testing Set (20%):** The remaining 20% of the dataset is reserved for testing the model. This set serves as unseen data, which the model has not encountered during training. Evaluating the model on unseen data helps us understand how well it generalizes to new, unseen instances.

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
# Specify the features and target variable
X = df[['Store ID', 'Total Price', 'Base Price']]
y = df['Units Sold']
# Split the data into an 80-20 training-testing ratio
```

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

Rest of explanation

In this section, the chosen model and its implementation for predicting product demand are discussed. The primary objective is to provide accurate forecasts of the number of units sold, which is a pivotal component in inventory management and decision-making processes.

Model Selection

After a comprehensive evaluation of available options, the Random Forest Regressor was selected for product demand prediction. The rationale for opting for this model is outlined as follows:

1. **Flexibility:** Random Forest is a versatile model capable of handling both regression and classification tasks. In the context of our regression problem, it offers the desired flexibility.
2. **Ensemble Learning**: Random Forest is an ensemble learning technique that amalgamates multiple decision trees to make predictions. This ensemble approach often results in more accurate and robust predictions.
3. **Feature Importance**: Random Forest is equipped to provide insights into the importance of various features, a valuable aspect for comprehending the factors influencing product demand.
4. **Robustness**: Random Forest demonstrates robustness against outliers compared to certain other models, making it an apt choice for real-world data.

Model Implementation

The implementation of the Random Forest Regressor unfolds as follows:

Missing values in the dataset are systematically addressed by substituting missing values in numerical columns with their respective means and in categorical columns with their modes.

The "Store ID" column is subjected to one-hot encoding to render it compatible with the model.

Feature Engineering: Alongside the original features, a novel feature labeled "Price Difference" is introduced. This feature denotes the disparity between "Total Price" and "Base Price" and is expected to capture crucial insights into price dynamics affecting product demand.

The dataset is partitioned into training and testing sets, with 80% of the data allocated for training and the remaining 20% reserved for testing.

The Random Forest Regressor is initialized with specific hyperparameters: n_estimators=100 (indicating the number of trees in the forest) and max_depth=10 (representing the maximum depth of individual trees). These initial values serve as starting points but remain amenable to fine-tuning in subsequent analyses.

The model is then trained using the training data, with the input features and the "Units Sold" variable as the target.

Predictions are generated using the test dataset to evaluate the model's performance.

The goal is to employ this Random Forest model to deliver precise product demand predictions that can prove instrumental in optimizing inventory management and the decision-making process.

Why Not ARIMA or Prophet?

The exclusion of ARIMA or Prophet for time series forecasting may raise questions. These methods are widely recognized for their utility in such tasks, but the decision to not utilize them for this specific project is rooted in the characteristics of our dataset. Notably, our dataset lacks a pivotal element for time series forecasting: time.

ARIMA and Prophet excel when applied to time series data, where observations are systematically recorded at regular intervals. These methods inherently account for the chronological order of data points, enabling them to make informed predictions. Given the absence of temporal data in our dataset, it is more prudent to adopt a regression-based model like Random Forest for predicting product demand based on the available features.

What metrics used for the accuracy check

To evaluate model performance, we will undertake these steps:

- Splitting the dataset into training and validation sets, allocating 80% for training and 20% for validation.
- Utilizing Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) as evaluation metrics.

Model evaluation and comparison will be vital in this phase. By comparing the performance of ARIMA, Prophet, LSTM, and XGBoost models, we aim to identify the most accurate approach for demand prediction. This comparison will guide us in selecting the best-performing model for fine-tuning and optimization.

MODEL

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset

df = pd.read_csv('/content/PoductDemand.csv')

# Handle missing values in numerical columns (replace with mean)
numerical_columns = ['Total Price', 'Base Price']
for col in numerical_columns:
    df[col].fillna(df[col].mean(), inplace=True)

# Handle missing values in categorical columns (replace with mode)
categorical_columns = ['Store ID']
for col in categorical_columns:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Perform one-hot encoding on the "Store ID" column
df = pd.get_dummies(df, columns=['Store ID'], drop_first=True)

# Feature Engineering (Adding a new feature)
# Let's add a new feature that represents the price difference between
Total Price and Base Price.
df['Price Difference'] = df['Total Price'] - df['Base Price']
```

```
# Define features and target
X = df[['Total Price', 'Base Price', 'Price Difference'] + [col for col
in df.columns if col.startswith('Store ID_')]]
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)

# Initialize and fit a Random Forest Regressor model with
hyperparameter tuning
model = RandomForestRegressor(n_estimators=100, max_depth=10,
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)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print("Product Demand Prediction with Random Forest Regressor")
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Plot the actual vs. predicted values
plt.scatter(y_test, y_pred, alpha=0.5)
plt.title("Actual vs. Predicted Values")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
```

OUTPUT

```
Product Demand Prediction with Random Forest Regressor
Mean Absolute Error: 24.96421436161803
Root Mean Squared Error: 41.84299974541368
```