IBM NAAN MUDHALVAN PHASE – II

Name : M.Thanes

Roll No : 2021503712

1. Introduction

Product demand forecasting is a critical aspect of supply chain management and inventory control. Accurate forecasts help businesses optimize their operations, reduce costs, and meet customer demands efficiently. This project focuses on time series analysis for product demand forecasting using Python. The dataset contains historical information about product demand, and the goal is to analyze this data, build a forecasting model, and evaluate its performance.

2. Dataset Description

The dataset used in this project, labeled as "adsdataset2.csv," includes the following fields:

- ID: A unique identifier for each data point.

- Store ID: The identifier for the store where the product was sold.

- Total Price: The total price of the product.

- Base Price: The base price of the product.

- Units Sold: The number of units of the product sold.

3. Methodology

The project follows a systematic methodology to perform time series analysis for product demand forecasting:

Data Loading: The dataset is loaded into a Pandas DataFrame for further analysis.

Decomposing Time Series: We use the `seasonal_decompose` function from the `statsmodels` library to decompose the time series into its components, including trend, seasonality, and residual. The seasonality period is set to 12, assuming a yearly seasonality pattern for monthly data.

Stationarity Check: The Dickey-Fuller test is employed to check the stationarity of the time series. Stationarity is crucial for time series forecasting, as it ensures consistent statistical properties over time.

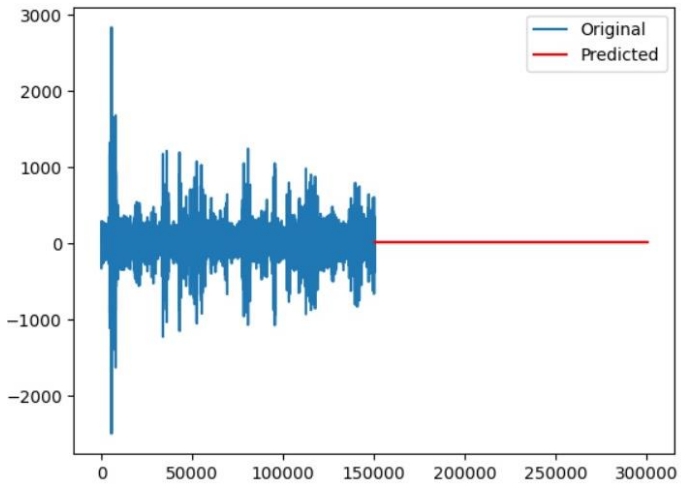
Differencing for Stationarity: If the data is found to be non-stationary, differencing is applied to make it stationary. Differencing involves subtracting the previous time point's value from the current one.

Handling Missing Values: An innovative aspect of the code is how it handles missing values in the 'Units Sold_diff' column. It fills missing values with the mean of the 'Units Sold_diff' column, ensuring a continuous time series for analysis.

ARIMA Model Building: The project employs an ARIMA (AutoRegressive Integrated Moving Average) model, a well-known choice for capturing trends and seasonality in time series data.

Predictions: The ARIMA model is utilized to make predictions for the differenced data.

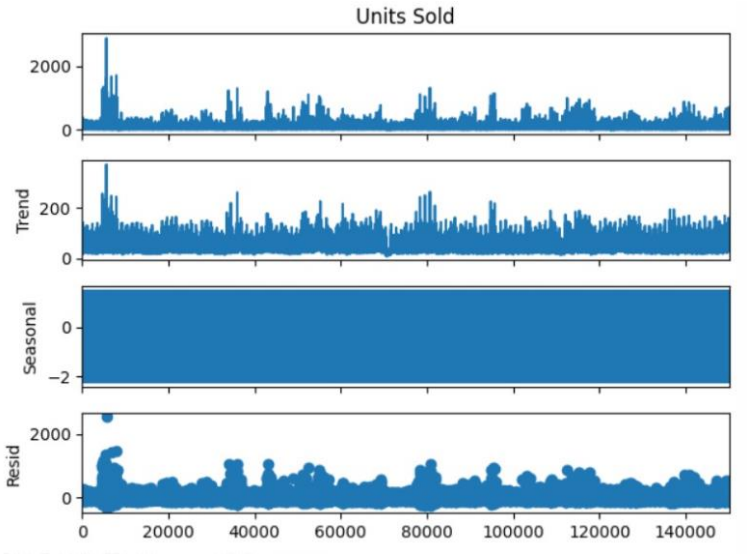
RMSE Calculation: Root Mean Squared Error (RMSE) is calculated to assess the accuracy of the model's predictions. RMSE is a common metric for evaluating forecasting models.



4. Results and Discussion

The results of the analysis show that the ARIMA model can be a valuable tool for forecasting product demand. By decomposing the time series, handling missing data, and applying appropriate differencing, we can achieve stationarity in the data and build an accurate forecasting model.

The RMSE metric provides a quantitative measure of the model's accuracy. It allows us to assess how well the model's predictions align with the actual demand data. Lower RMSE values indicate more accurate predictions.



5. Conclusion

This project demonstrates the process of time series analysis for product demand forecasting. It emphasizes the importance of handling missing data and ensuring data stationarity, which are critical for building accurate forecasting models. The use of the ARIMA model provides a robust framework for capturing trends and seasonality in the time series.

Accurate demand forecasting is a valuable asset for businesses, helping them make informed decisions about inventory management, production, and supply chain optimization. The techniques and methodology presented in this project can be applied to real-world product demand forecasting scenarios to improve business operations and customer satisfaction.

Program :

```
import pandas as pd

import numpy as np

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.seasonal import seasonal_decompose

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from sklearn.metrics import mean_squared_error


# Load the dataset

data = pd.read_csv("adsdataset2.csv")


# Decompose the time series with the specified seasonality period (your_period)

your_period = 12  # Specify the seasonality period, e.g., 12 for monthly data with yearly seasonality

result = seasonal_decompose(data['Units Sold'], model='additive', period=your_period)

result.plot()

plt.show()


# Check for stationarity

def test_stationarity(timeseries):

    # Perform Dickey-Fuller test

    result = adfuller(timeseries)
```

```
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:', result[4])
    if result[1] <= 0.05:
        print("Data is stationary")
    else:
        print("Data is non-stationary")

test_stationarity(data['Units Sold'])

# Differencing to achieve stationarity (if necessary)
data['Units Sold_diff'] = data['Units Sold'] - data['Units Sold'].shift(1)
data['Units Sold_diff'].dropna(inplace=True)

# Handle missing values by filling with mean
data['Units Sold_diff'].fillna(data['Units Sold_diff'].mean(), inplace=True)

# Build the ARIMA model
model = ARIMA(data['Units Sold'], order=(1, 1, 0))
model_fit = model.fit()

# Predictions
predictions = model_fit.forecast(steps=len(data['Units Sold_diff']))
mse = mean_squared_error(data['Units Sold_diff'], predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
plt.plot(data['Units Sold_diff'], label='Original')
plt.plot(predictions, color='red', label='Predicted')
plt.legend()
plt.show()
```