Forecasting Unit Sales (Task 1)

NAME: ANCY SHARMILA D
REG NO: 20MIS0211

STEPS FOLLOWED

1. IMPORTING LIBRARIES

pandas
numpy
matplotlib.pyplot
seaborn
prophet
sklearn

```
[1] pip install prophet

Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.4)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.26.4)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (2.1.4)
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/dist-packages (from prophet) (0.53)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from prophet) (4.66.4)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from prophet) (6.4.0)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (3.1.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->holidays>=0.25->prophet) (1.16.0)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
from prophet import Prophet
from sklearn.metrics import mean_squared_error
import gc

# Function to optimize memory usage
def optimize_memory(df):
    for col in df.select_dtypes(include=['float64']).columns:
        df[col] = pd.to_numeric(df[col], downcast='float')
    for col in df.select_dtypes(include=['int64']).columns:
        df[col] = pd.to_numeric(df[col], downcast='integer')
    return df
```

2. IMPORTING DATASET

From google drive the train.csv and test.csv

```python
[3] from google.colab import drive
    drive.mount('/content/drive')

    # Load a CSV file from Google Drive
    train_data = pd.read_csv('/content/drive/MyDrive/NapQueen/forecasting-unit-sales-vit-task-2/train.csv')
    test_data = pd.read_csv('/content/drive/MyDrive/NapQueen/forecasting-unit-sales-vit-task-2/test.csv')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

3. DATA CLEANING

3.1 Convert 'date' column to datetime

```python
[4] # Optimize memory usage
    train_data = optimize_memory(train_data)
    test_data = optimize_memory(test_data)

[5] # Convert 'date' to datetime
    train_data['date'] = pd.to_datetime(train_data['date'])
    test_data['date'] = pd.to_datetime(test_data['date'])
```

3.2 Check for missing values

```
# Check for missing values
print(train_data.isnull().sum())
print(test_data.isnull().sum())
```

```
ID                0
date              0
Item Id           2
Item Name      1832
ad_spend      24187
anarix_id         0
units         17898
unit_price        0
dtype: int64
ID               0
date             0
Item Id          0
Item Name      344
ad_spend      1451
anarix_id        0
unit_price       0
dtype: int64
```

- Drop rows where 'Item Id' is missing
- Drop columns with excessive missing values (e.g., more than 50% missing values)
- Impute 'ad_spend' with median value
- Impute 'units' with median value

```
[7] # Drop rows where 'Item Id' is missing
    train_data = train_data.dropna(subset=['Item Id'])

    # Drop columns with excessive missing values (e.g., more than 50% missing values)
    train_data = train_data.drop(columns=['Item Name'])
    test_data = test_data.drop(columns=['Item Name'])
```

```
#Impute 'ad_spend' with median value
median_ad_spend = train_data['ad_spend'].median()
train_data['ad_spend'].fillna(median_ad_spend, inplace=True)
test_data['ad_spend'].fillna(median_ad_spend, inplace=True)

# Impute 'units' with median value
median_units = train_data['units'].median()
train_data['units'].fillna(median_units, inplace=True)
```

```
<ipython-input-8-469446b570ca>:4: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise in a future error of pandas. Value '4.23' has dty
  test_data['ad_spend'].fillna(median_ad_spend, inplace=True)
```

Initial exploration of the dataset to understand its structure, identify missing values, and analyze basic statistics

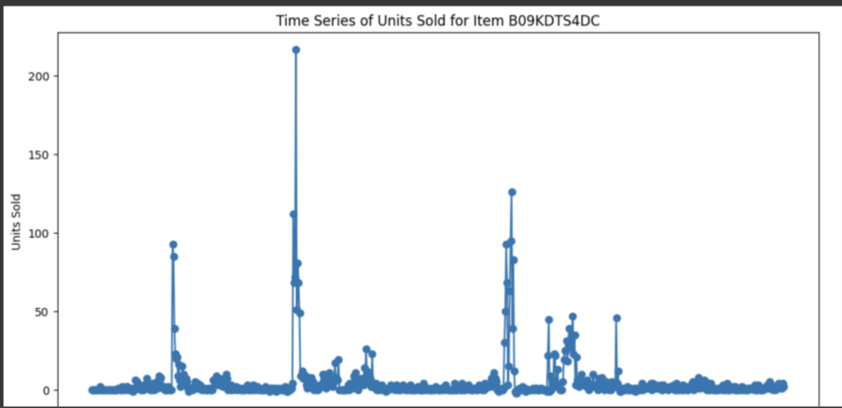
```
[9] print(train_data.isnull().sum())
    print(test_data.isnull().sum())
```

```
ID            0
date          0
Item Id       0
ad_spend      0
anarix_id     0
units         0
unit_price    0
dtype: int64
ID            0
date          0
Item Id       0
ad_spend      0
anarix_id     0
unit_price    0
dtype: int64
```

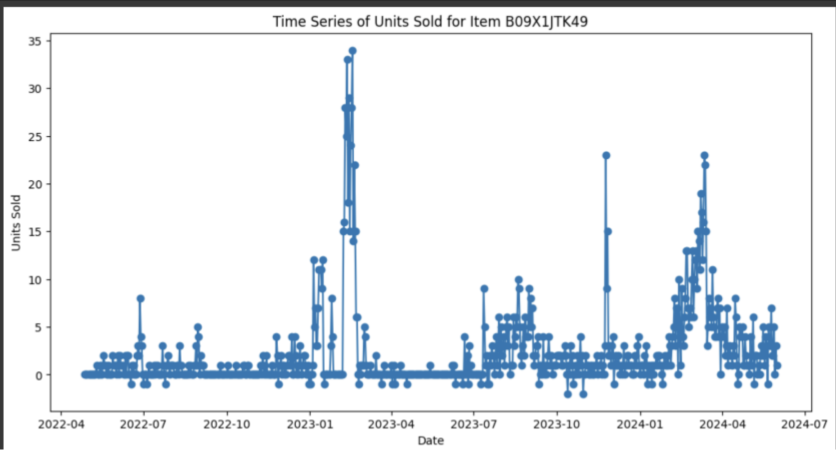
4. DATA VISUALIZATION

- Time Series Plot: Provides insights into trends over time.

```python
#TIME SERIES PLOT
# Plot sales trends over time for a sample item
sample_item = train_data['Item Id'].unique()[0]
sample_data = train_data[train_data['Item Id'] == sample_item]

plt.figure(figsize=(12, 6))
plt.plot(sample_data['date'], sample_data['units'], marker='o')
plt.title(f'Time Series of Units Sold for Item {sample_item}')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.show()
```

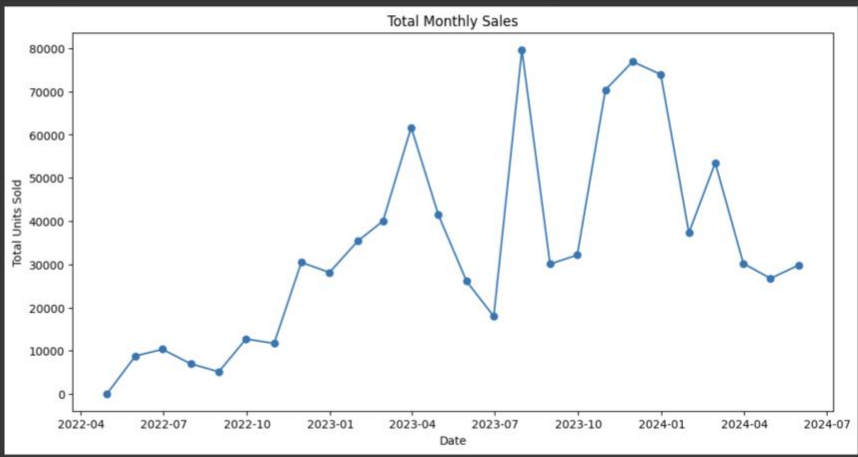



```python
sample_item = train_data['Item Id'].unique()[30]
sample_data = train_data[train_data['Item Id'] == sample_item]

plt.figure(figsize=(12, 6))
plt.plot(sample_data['date'], sample_data['units'], marker='o')
plt.title(f'Time Series of Units Sold for Item {sample_item}')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.show()
```

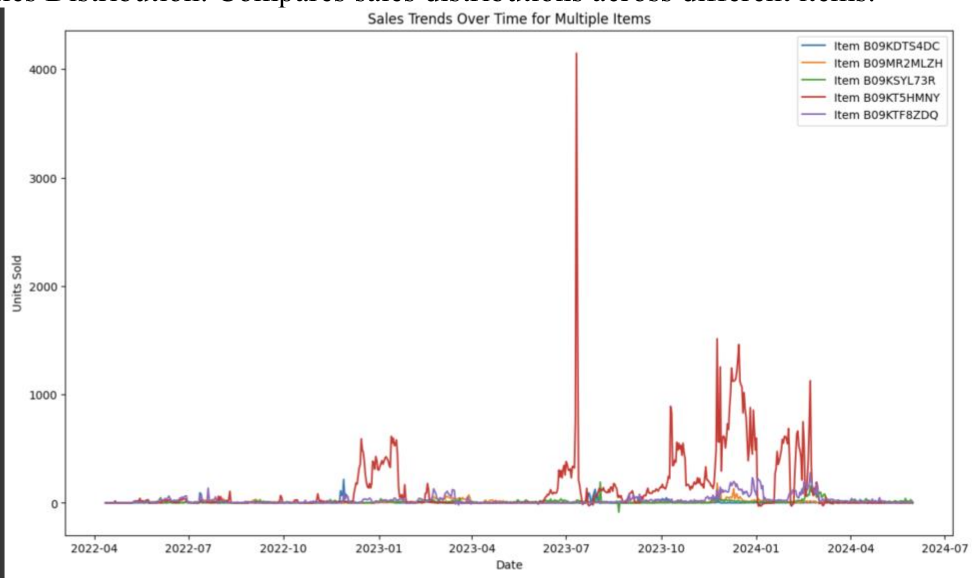


- Monthly Sales Aggregation: Helps understand overall sales trends on a monthly basis.

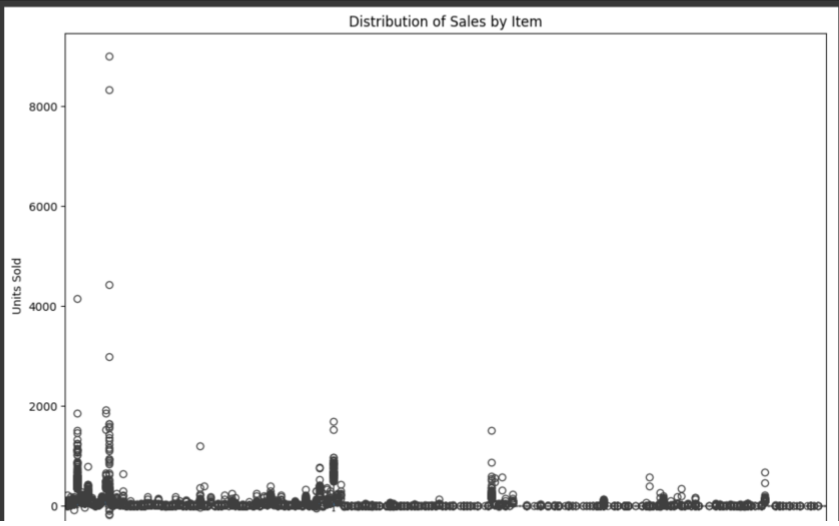
```
#MONTHLY SALES AGGREGATION
# Aggregate sales data by month
monthly_sales = train_data.resample('M', on='date').agg({'units': 'sum'})

plt.figure(figsize=(12, 6))
plt.plot(monthly_sales.index, monthly_sales['units'], marker='o', linestyle='-')
plt.title('Total Monthly Sales')
plt.xlabel('Date')
plt.ylabel('Total Units Sold')
plt.show()
```



- Sales Distribution: Compares sales distributions across different items.

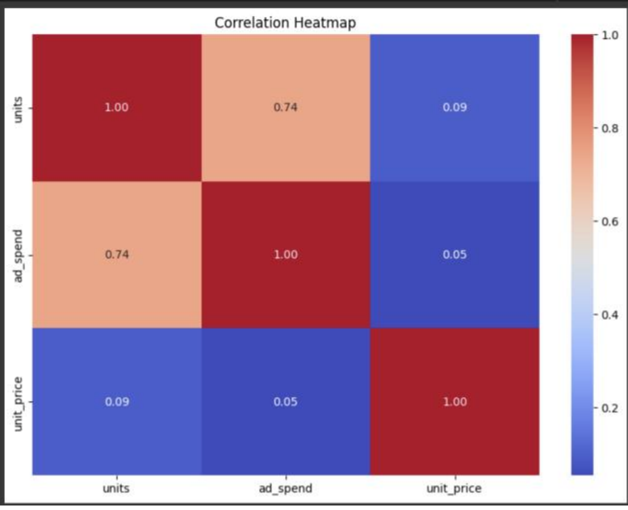


```python
plt.figure(figsize=(12, 8))
sns.boxplot(x='Item Id', y='units', data=train_data)
plt.title('Distribution of Sales by Item')
plt.xlabel('Item ID')
plt.ylabel('Units Sold')
plt.xticks(rotation=90)
plt.show()
```



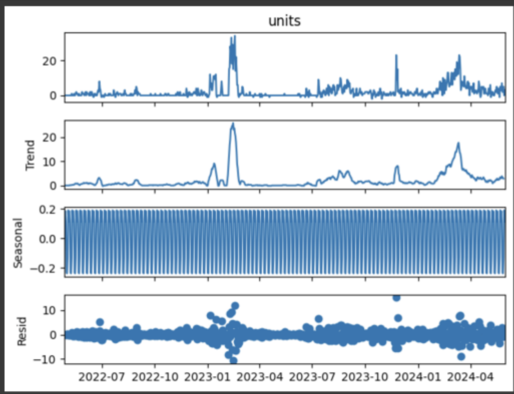
- Correlation Heatmap: Examines relationships between features.

```python
# CORREALTION HEATMAP
correlation_matrix = train_data[['units', 'ad_spend', 'unit_price']].corr()

plt.figure(figsize=(10, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



- Seasonal Decomposition: Breaks down the time series into trend, seasonality, and residuals.

```
# SEASONAL DECOMPOSITION
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
decomposition = seasonal_decompose(sample_data.set_index('date')['units'], model='additive')
decomposition.plot()
plt.show()
```




```
[17] #Create Time-Based Features
# Extract time-based features
train_data['year'] = train_data['date'].dt.year
train_data['month'] = train_data['date'].dt.month
train_data['day'] = train_data['date'].dt.day
train_data['dayofweek'] = train_data['date'].dt.dayofweek
```

Plotting the time series data to identify trends, seasonality, and any anomalies. Visualizations revealed complex seasonal patterns and trends, suggesting the need for a model that can handle such intricacies.

5. FEATURE ENGINEERING

5.1. Create Time-Based Features

- Extract features like year, month, day, day of the week, etc.

```
[17] #Create Time-Based Features
# Extract time-based features
train_data['year'] = train_data['date'].dt.year
train_data['month'] = train_data['date'].dt.month
train_data['day'] = train_data['date'].dt.day
train_data['dayofweek'] = train_data['date'].dt.dayofweek
train_data['is_weekend'] = (train_data['dayofweek'] >= 5).astype(int)

test_data['year'] = test_data['date'].dt.year
test_data['month'] = test_data['date'].dt.month
test_data['day'] = test_data['date'].dt.day
test_data['dayofweek'] = test_data['date'].dt.dayofweek
test_data['is_weekend'] = (test_data['dayofweek'] >= 5).astype(int)
```

```
# Inspect data
print(train_data.info())
print(train_data.head())
print(train_data.describe())
print(test_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 101488 entries, 0 to 101489
Data columns (total 12 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   ID          101488 non-null  object
 1   date        101488 non-null  datetime64[ns]
 2   Item Id     101488 non-null  object
 3   ad_spend    101488 non-null  float64
 4   anarix_id   101488 non-null  object
 5   units       101488 non-null  float32
 6   unit_price  101488 non-null  float64
 7   year        101488 non-null  int32
 8   month       101488 non-null  int32
 9   day         101488 non-null  int32
 10  dayofweek   101488 non-null  int32
 11  is_weekend  101488 non-null  int64
dtypes: datetime64[ns](1), float32(1), float64(2), int32(4), int64(1), object(3)
memory usage: 8.1+ MB
None
```

```
1  2022-04-12_B09MR2MLZH 2022-04-12  B09MR2MLZH      4.23  NAPQUEEN    0.0
2  2022-04-12_B09KSYL73R 2022-04-12  B09KSYL73R      4.23  NAPQUEEN    0.0
3  2022-04-12_B09KT5HMNY 2022-04-12  B09KT5HMNY      4.23  NAPQUEEN    0.0
4  2022-04-12_B09KTF8ZDQ 2022-04-12  B09KTF8ZDQ      4.23  NAPQUEEN    0.0

   unit_price  year  month  day  dayofweek  is_weekend
0         0.0  2022      4   12          1           0
1         0.0  2022      4   12          1           0
2         0.0  2022      4   12          1           0
3         0.0  2022      4   12          1           0
4         0.0  2022      4   12          1           0
                                date       ad_spend          units  \
count                         101488  101488.000000  101488.000000
mean   2023-07-09 19:09:44.297650944      85.382291       8.647209
min              2022-04-12 00:00:00       0.000000    -173.000000
25%              2023-02-26 00:00:00       0.540000       0.000000
50%              2023-07-16 00:00:00       4.230000       1.000000
75%              2023-12-13 00:00:00      21.642500       3.000000
max              2024-05-31 00:00:00   47934.990000    9004.000000
std                              NaN     464.174945      62.672401

          unit_price           year          month            day  \
count  101488.000000  101488.000000  101488.000000  101488.000000
mean      106.753025    2023.060175       6.051405      15.875759
min     -8232.000000    2022.000000       1.000000       1.000000
25%         0.000000    2023.000000       3.000000       8.000000
50%         0.000000    2023.000000       6.000000      16.000000
75%         0.000000    2023.000000       9.000000      23.000000
max     21557.390000    2024.000000      12.000000      31.000000
std       425.708664       0.612774       3.488462       8.805080

           dayofweek     is_weekend
count  101488.000000  101488.000000
mean        2.996876       0.284625
min         0.000000       0.000000
25%         1.000000       0.000000
50%         3.000000       0.000000
75%         5.000000       1.000000
max         6.000000       1.000000
std         1.999103       0.451238
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2833 entries, 0 to 2832
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          2833 non-null   object
```

6. MODEL DECISION & HYPERPARAMETER TUNING

PROPHET (BY FACEBOOK)-

The sales data exhibits multiple seasonal patterns with fluctuations.
Prophet canhandle these complexities better than traditional models like ARIMA or Holt-Winters.

- The dataset contains missing values in various columns. Prophet's robustness to missing data makes it a reliable choice without the need for extensive preprocessing.

- Prophet is designed to be easy to use with minimal parameter tuning. This is beneficial when dealing with a large dataset where hyperparameter tuning can be resource-intensive compared to the traditional ARIMA model.

- Prophet decomposes the time series into trend and seasonal components, making it easier to understand and interpret the results.

Prophet emerged as the best model for this task due to its robustness to missing data, capability to handle multiple seasonality, minimal parameter tuning, and ease of use. This makes it a suitable choice for forecasting unit sales in a dataset with complex patterns and potential data quality issues.

```python
# Convert 'date' to datetime to run the model
train_data['date'] = pd.to_datetime(train_data['date'])
test_data['date'] = pd.to_datetime(test_data['date'])
```

```python
[20] # Prepare the data for Prophet
train_data_prophet = train_data[['date', 'units']].rename(columns={'date': 'ds', 'units': 'y'})
```

```python
# Train/validation split
train_test_split = int(len(train_data_prophet) * 0.8)
train_prophet = train_data_prophet[:train_test_split]
val_prophet = train_data_prophet[train_test_split:]
```

```python
[22] # Fit the Prophet model
model = Prophet()
model.fit(train_prophet)
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp2f9iy9u3/m4cqjjgk.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp2f9iy9u3/m33c4_gl.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=80092', 'data', 'file=/tmp/tmp2f9iy9u3
12:01:55 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:02:16 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x79a30a957fd0>
```

```python
# Make predictions
future = model.make_future_dataframe(periods=len(val_prophet), freq='D')
forecast = model.predict(future)
```

```python
[24] # Evaluate the model on validation set
val_forecast = forecast[-len(val_prophet):]['yhat']
mse = mean_squared_error(val_prophet['y'], val_forecast)
print(f'Mean Squared Error on validation set: {mse}')
```

7. PREDICTIONS & EVALUATION

Mean Squared Error on validation set: 143776.08711872145

```python
[23] # Make predictions
future = model.make_future_dataframe(periods=len(val_prophet), freq='D')
forecast = model.predict(future)
```

```python
[24] # Evaluate the model on validation set
val_forecast = forecast[-len(val_prophet):]['yhat']
mse = mean_squared_error(val_prophet['y'], val_forecast)
print(f'Mean Squared Error on validation set: {mse}')

# Calculate RMSE for better interpretability
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error on validation set: {rmse}')
```

```
Mean Squared Error on validation set: 143776.08711872145
Root Mean Squared Error on validation set: 379.17817331529176
```

```python
[25] # Forecast future values
future_test = model.make_future_dataframe(periods=len(test_data), freq='D', include_history=True)
forecast_test = model.predict(future_test)

# Extract only the forecasted values for the test period
forecast_future = forecast_test[-len(test_data):]
```

8. SAVE SUBMISSION FILE

```python
[26] # Prepare submission
submission = pd.DataFrame({
    'date': test_data['date'],
    'Item Id': test_data['Item Id'],
    'TARGET': forecast_future['yhat'].values
})
```

```python
[27] # Save submission to CSV
submission.to_csv('submission.csv', index=False)
```

```python
[28] # Clear memory
del train_data, test_data
gc.collect()
```