Data Glacier Data Scientist Internship

Batch: LISUM23: 30

Week13: Final Project Report and Code

Project: Retail Forecasting

Team member's details:

Group Name: Retail_forecasting

Name	Richa Mishra	Shalu Kumar	Madoka Fujii	Shushun Ren
Email	mricha828@gmail.com	ss4676@njit.edu	mdkfji@gmail.com	shushunr@umich.edu
Country	United States	United States	United States	United States
College/Company	NJIT	NJIT	Sumitomo Mitsui Trust Bank	Umich
Specialization	Data Science	Data Science	Data Science	Data Science

Project life cycle along with deadline:

Project weeks	**Deadline**	**Lifecycle**
Week7	Aug 19, 2023	Problem statement, Pre-process
Week8	Aug 26, 2023	Data process, understanding
Week9	Sep 02, 2023	Data Cleaning, Merge, Review
Week10	Sep 09, 2023	EDA, Final recommendation
Week11	Sep 16, 2023	EDA presentation for business users
Week12	Oct 7, 2023 (extended)	Model Selection and Model Building/Dashboard
Week13	**Oct 14, 2023 (extended)**	**Final Project Report and Code**

Tabular data details: forecasting_case_study.xlsx:

Total number of observations	1218
Total number of files	1
Total number of features	12
Base format of the file	.xlsx
Size of the data	80KB

Problem Description:

This major Australian beverage corporation operates within the beverage industry. Their product distribution spans across multiple supermarket chains, and they actively conduct robust promotional campaigns year-round. The demand for their products is subject to fluctuations driven by factors such as holidays and seasonal trends. They require a weekly item-level forecast for each of their products, categorized into weekly intervals.

Data Preparation:

1. Check if there are missing values

   There is no NULL value. So we can surely go ahead with the dataset.

```
[ ] from pyspark.sql.functions import col, isnan, when, count

    missing_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns if df.schema[c].dataType != 'date'])
    missing_counts.show()

    +-------+----+-----+------------------+-------------+---------------+---------------+---------------+----------+-----+------+---------+
    |Product|date|Sales|Price Discount (%)|In-Store Promo|Catalogue Promo|Store End Promo|Google_Mobility|Covid_Flag|V_DAY|EASTER|CHRISTMAS|
    +-------+----+-----+------------------+-------------+---------------+---------------+---------------+----------+-----+------+---------+
    |      0|   0|    0|                 0|            0|              0|              0|              0|         0|    0|     0|        0|
    +-------+----+-----+------------------+-------------+---------------+---------------+---------------+----------+-----+------+---------+
```

2. Validate the name of columns

```
[ ] df = (
        df
        .withColumnRenamed("Price Discount (%)", "Price_Discount")
        .withColumnRenamed("In-Store Promo", "In-Store_Promo")
        .withColumnRenamed("Catalogue Promo", "Catalogue_Promo")
        .withColumnRenamed("Store End Promo", "Store_End_Promo")
    )
```

3. Zero Sales

   For 0 Sales of production of SKU1 to SKU5, decided to keep them, not drop. SKU6 of missing data between 2020-11-22 to 2020-12-27, decided to create them to balance with other products.

```
[ ] from pyspark.sql import Row

    data_to_append = [
        Row(Product="SKU6", date="2020-11-22", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1,
        Row(Product="SKU6", date="2020-11-29", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1,
        Row(Product="SKU6", date="2020-12-6", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1, \
        Row(Product="SKU6", date="2020-12-13", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1,
        Row(Product="SKU6", date="2020-12-20", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1,
        Row(Product="SKU6", date="2020-12-27", Sales=0, Price_Discount= 0, In_Store_Promo=0, Catalogue_Promo=0, Store_End_Promo=0, Google_Mobility= 0, Covid_Flag=1,
    ]
```

4. Combine 3 holidays

For the 3 holidays of V_DAY, EASTER, and CHRISTMAS, we decided to combine them into 1 column because the dates of each holiday are different without duplication.

```
[ ] from pyspark.sql.functions import when, lit, col

    df = df.withColumn("Holiday_Flag", when((col("V_DAY") == 1) | (col("EASTER") == 1) | (col("CHRISTMAS") == 1), lit(1)).otherwise(lit(0)))
    df= df.drop("V_DAY", "EASTER", "CHRISTMAS")
    df.show()
```

```
+-------+----------+------+--------------+--------------+---------------+---------------+---------------+----------+------------+
|Product|      date| Sales|Price_Discount|In-Store_Promo|Catalogue_Promo|Store_End_Promo|Google_Mobility|Covid_Flag|Holiday_Flag|
+-------+----------+------+--------------+--------------+---------------+---------------+---------------+----------+------------+
|   SKU1|2017-02-05| 27750|           0.0|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-02-12| 29023|           0.0|             1|              0|              1|            0.0|         0|           1|
|   SKU1|2017-02-19| 45630|          0.17|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-02-26| 26789|           0.0|             1|              0|              1|            0.0|         0|           0|
|   SKU1|2017-03-05| 41999|          0.17|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-03-12| 29731|           0.0|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-03-19| 27365|           0.0|             1|              0|              0|            0.0|         0|           0|
|   SKU1|2017-03-26| 27722|           0.0|             1|              0|              1|            0.0|         0|           0|
|   SKU1|2017-04-02| 44339|          0.17|             1|              0|              0|            0.0|         0|           0|
|   SKU1|2017-04-09| 54655|          0.17|             1|              0|              0|            0.0|         0|           1|
|   SKU1|2017-04-16|108159|          0.44|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-04-23| 30361|           0.0|             1|              0|              1|            0.0|         0|           0|
|   SKU1|2017-04-30| 42154|          0.17|             1|              0|              1|            0.0|         0|           0|
|   SKU1|2017-05-07| 39782|          0.17|             0|              0|              0|            0.0|         0|           0|
|   SKU1|2017-05-14| 29490|           0.0|             0|              0|              0|            0.0|         0|           0|
```

5. Divided data into 6

The dataset has been effectively partitioned into **six** distinct product datasets using **PySpark**. This division allows us to focus on specific product categories individually, streamlining data preparation, exploratory analysis, feature engineering, modeling, and subsequent analysis for each product group. This approach enhances our ability to gain insights, build tailored models, and optimize our analysis for each product category while ensuring efficient and manageable data processing.

```
[ ] df.write.option("header", True) \
            .partitionBy("Product") \
            .mode("overwrite") \
            .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast")
```

```
df1=spark.read.option("header",True) \
           .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU1")
```

```
[ ] df2=spark.read.option("header",True) \
          .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU2")
```

```
[ ] df3=spark.read.option("header",True) \
           .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU3")
```

```
[ ] df4=spark.read.option("header",True) \
           .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU4")
```

```
[ ] df5=spark.read.option("header",True) \
           .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU5")
```

```
[ ] df6=spark.read.option("header",True) \
           .csv("/content/drive/MyDrive/Colab Notebooks/DG/Forecast/Product=SKU6")
```

6. <u>Validate the data type</u>

We validated the data type as below.

-Variables with numeric value to int

-Validates with one-hot encoding to categorical

-date to datetime

```
cols=['Sales','In-Store_Promo','Catalogue_Promo','Store_End_Promo','Google_Mobility','Covid_Flag','Holiday_Flag']
for i in range(len(products)):
  products[i] = ps.DataFrame(products[i])
  products[i]['Price_Discount']=products[i]['Price_Discount'].apply(pd.to_numeric)
  products[i]['Sales']=products[i]['Sales'].apply(pd.to_numeric)
  products[i]['In-Store_Promo']=products[i]['In-Store_Promo'].astype('category')
  products[i]['Catalogue_Promo']=products[i]['Catalogue_Promo'].astype('category')
  products[i]['Store_End_Promo']=products[i]['Store_End_Promo'].astype('category')
  products[i]['Google_Mobility']=products[i]['Google_Mobility'].apply(pd.to_numeric)
  products[i]['Covid_Flag']=products[i]['Covid_Flag'].astype('category')
  products[i]['Holiday_Flag']=products[i]['Holiday_Flag'].astype('category')
  #i['date']=i['date'].apply(pd.to_datetime,axis=1)
  print(products[i].dtypes)
```

```
/usr/local/lib/python3.10/dist-packages/pyspark/pandas/base.py:1697: FutureWarning: Argument `na_sentinel` will be removed ir
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pyspark/pandas/base.py:1697: FutureWarning: Argument `na_sentinel` will be removed ir
  warnings.warn(
date                 object
Sales                 int64
Price_Discount        int64
In-Store_Promo     category
Catalogue_Promo    category
Store_End_Promo    category
Google_Mobility     float64
Covid_Flag         category
Holiday_Flag       category
```

7. <u>Remove outlier with narrow range</u>

To remove outliers using the Interquartile Range (IQR) method, calculate the IQR by finding the difference between the third quartile (Q3) and the first quartile (Q1). Then, define a lower bound (Q1 - 1.0 * IQR) and an upper bound (Q3 + 1.0 * IQR) and filter out data points that fall outside this range. This method helps identify and exclude extreme values from the dataset.

```
[ ]  #q_hi  = df1['Sales'].quantile(0.99)
     Q1 = np.percentile(df1['Sales'], 25, method='midpoint')
     Q3 = np.percentile(df1['Sales'], 75, method='midpoint')
     IQR = Q3 - Q1
     print(IQR)
     upper= 1.0*IQR #make the range of outliers narrower.
     df1 = df1[(df1['Sales'] < upper)]
```

```
24606.0
```

Data Analysis / EDA:

1. Adfuller test

   Check if the data is stationary for time series.

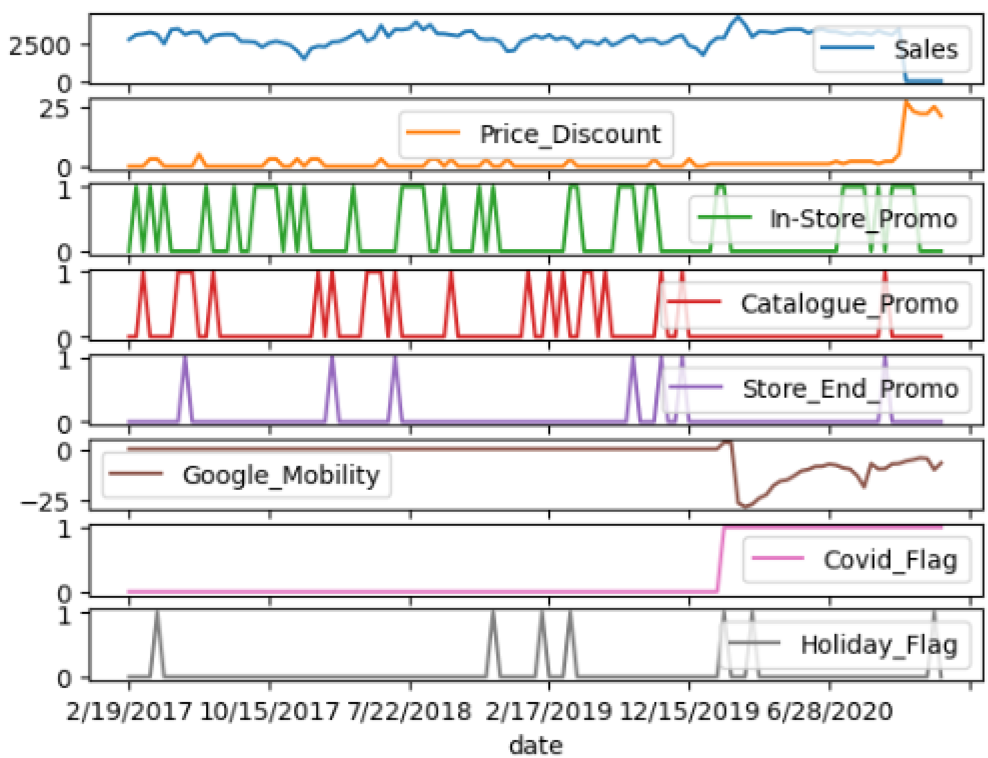
```
from statsmodels.tsa.stattools import adfuller
def adf_test(dataset):
     df2test = adfuller(dataset, autolag = 'AIC')
     #df2test = adfuller(dataset.diff()[1:])
     print("1. ADF : ",df2test[0])
     print("2. P-Value : ", df2test[1])
     print("3. Num Of Lags : ", df2test[2])
     print("4. Num Of Observations Used For ADF Regression:",      df2test[3])
     print("5. Critical Values :")
     for key, val in df2test[4].items():
         print("\t",key, ": ", val)
adf_test(df2_2['Sales'])
```

```
1. ADF :  -2.4102471662160756
2. P-Value :  0.138865084825955517
3. Num Of Lags :  0
4. Num Of Observations Used For ADF Regression: 116
5. Critical Values :
         1% :  -3.4880216384691867
         5% :  -2.8867966864160075
         10% :  -2.5802408234244947
```

2. Visualization of each product

Below is product SKU2. Each product has different sales and characteristics.

```
array([<Axes: xlabel='date'>, <Axes: xlabel='date'>,
       <Axes: xlabel='date'>, <Axes: xlabel='date'>,
       <Axes: xlabel='date'>, <Axes: xlabel='date'>,
       <Axes: xlabel='date'>, <Axes: xlabel='date'>], dtype=object)
```



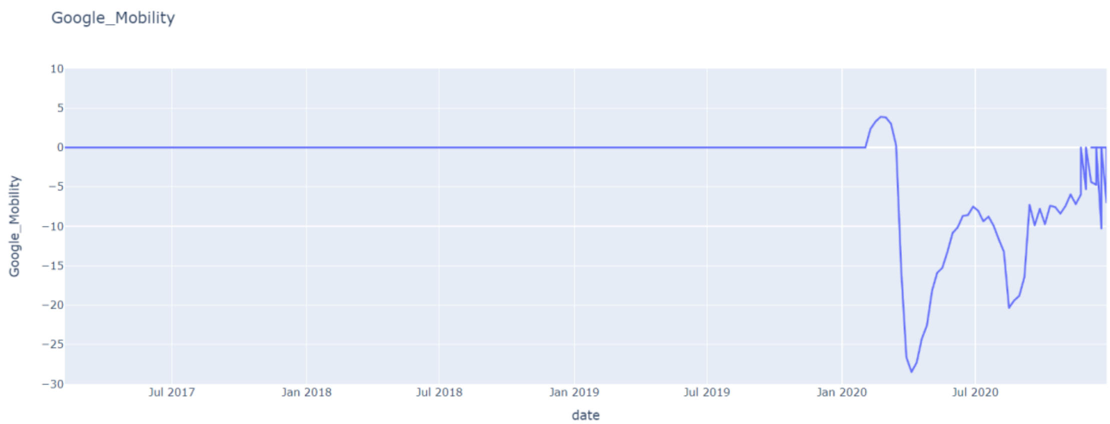
3. Covid_Flag

Covid Flag started from February 09, 2020.

```
[ ] window_spec = Window.orderBy("date")
    df = df.withColumn("prev_flag", F.lag("Covid_Flag").over(window_spec))

    start_dates = df.filter((F.col("Covid_Flag") == 1) & (F.col("prev_flag") == 0))

    start_dates.select("Date").show()
```

```
+----------+
|      Date|
+----------+
|2020-02-09|
+----------+
```

4. Google Mobility



-Google Mobility is related Covid19. This is because the line is flat until February 2, 2020 above the plot. The flat line means there are no activities and no existing record.

-After February 9, 2020, it started fluctuating and keeps changing. According to the variable of Covid Flag, it started being recorded as 1 after February 9, 2020. The timing between Google Mobility and Covid Flag is exactly coinciding.

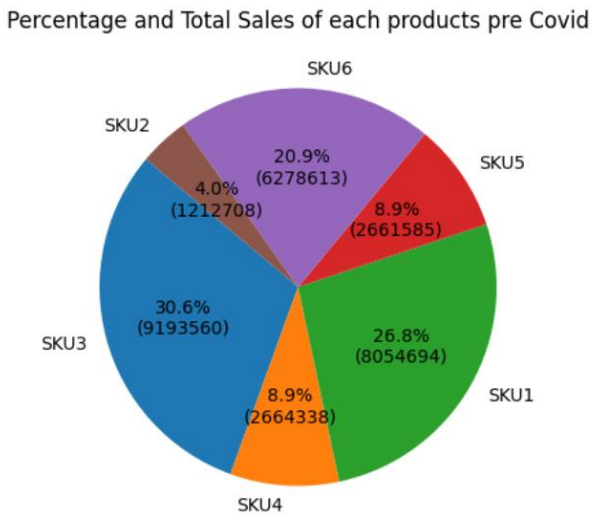
-Google Mobility data tracks travel patterns in detail, such as how often people go to public places and how much time they spend commuting or shopping. This will allow us to assess the risk of spread of infection and predict the spread of infection in a particular region or city.

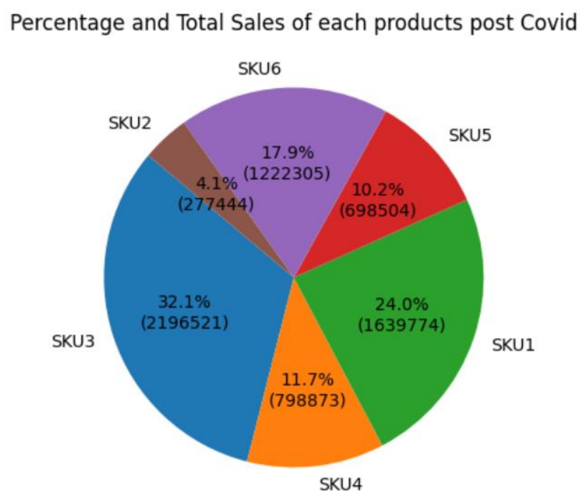
5. To analyze the data pre-Covid and post-Covid, we divide the data to 2

```python
before_date = df.filter(col("date") < "2020-02-09")

after_date = df.filter(col("date") >= "2020-02-09")

before_date.show()
after_date.show()
```

Product	date	Sales	Price_Discount	In-Store_Promo	Catalogue_Promo	Store_End_Promo	Google_Mobility	Covid_Flag	Holiday_Flag	prev_flag
SKU1	2017-02-05	27750	0.0	0	0	0	0.0	0	0	null
SKU2	2017-02-05	7180	0.25	1	0	0	0.0	0	0	0
SKU3	2017-02-05	39767	0.3	0	1	1	0.0	0	0	0
SKU4	2017-02-05	12835	0.3	0	1	1	0.0	0	0	0

Comparison of percentage of sharing each product before Covid and after Covid.





In terms of the percentages of the products to sales, there is no very big difference between pre covid and post covid.

In general both before Covid and after Covid, SKU3 is the most popular product. SKU1 is the secondest popular. And SKU6 is the 3rd.

SKU4 and SKU5 have the same sales amounts. SKU2 is the least popular product.

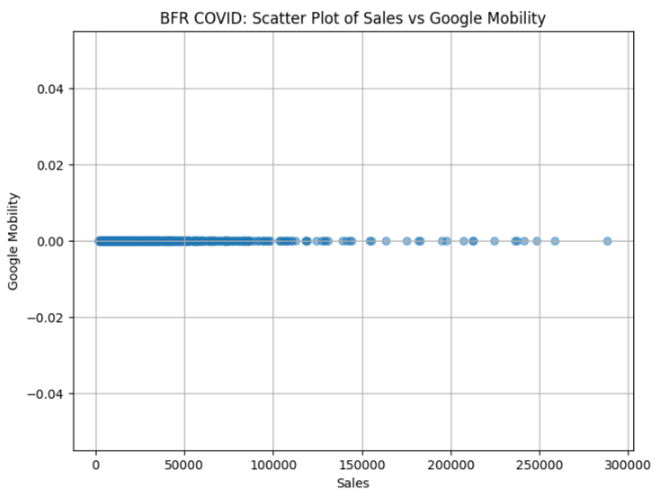
-Comparing the sales amount before Covid and after Covid

Product	Sales_before_covid	Sales_after_covid	Sales_change
SKU1	8054694.0	1639774	-79.6420075051889
SKU2	1212708.0	277444	-77.12194526629659
SKU3	9193560.0	2196521	-76.10804737229104
SKU4	2664338.0	798873	-70.01607904102258
SKU5	2661585.0	698504	-73.75608894699963
SKU6	6278613.0	1222305	-80.5322449400847

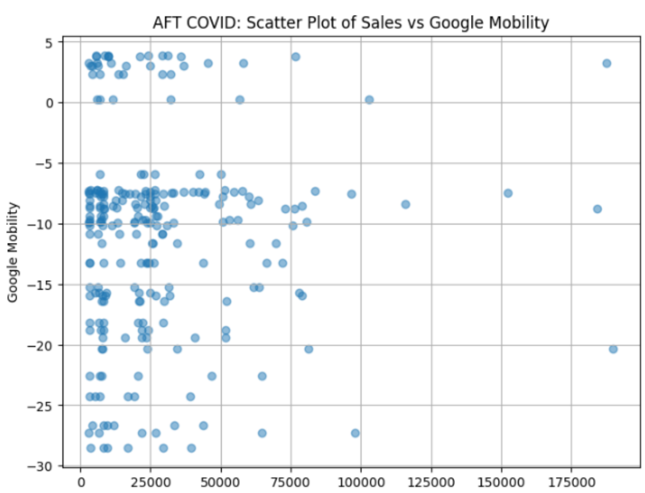
Sales of each of the products significantly reduced between 70% and 80% minus after Covid compared to before Covid.

6. Correlation between Sales and Google Mobility

We don't see any correlation between the two variables. (Google Mobility was no activated in BFR Covid so the plot is just flat.)



In AFT Covid, there is no correlation.



<u>Models:</u>

We use multivariate time series for this dataset.

Tried methods are below and the best model is **Ensemble model with ARIMA, Random Forest, Gradient Boosting, and Prophet.**

-**LSTM**: (The method is for large amount of dataset. This dataset is too small, it was not appropriate to apply this time.)

-**ARIMA**: The result of SKU2, MAPE: 14.82% and Accuracy 85.18%.

-**Prophet**: The result of MAPE and Accuracy is inf. Need to investigate more.

-**XGBoost**: (The method is for large amount of dataset. This dataset is too small, it was not appropriate to apply this time.)

-**VAR**: Applying VAR, we need to have stational data. That means we can't use it because this data is not stational

-**Synthetic data with Ensemble model with ARIMA, Random Forest, Gradient Boosting, Prophet.** The dataset is too small to forecast with accuracy. So, applied synthetic data. The results were not good as Ensemble model.

'SKU1': {'MAPE': 1647681.2717655227, 'Accuracy': -1647581.2717655227},

'SKU2': {'MAPE': 299203.142829213, 'Accuracy': -299103.142829213},

'SKU3': {'MAPE': 1610205.7865024037, 'Accuracy': -1610105.7865024037},

'SKU4': {'MAPE': 507888.4638358564, 'Accuracy': -507788.4638358564},

'SKU5': {'MAPE': 353632.54354865255, 'Accuracy': -353532.54354865255},

'SKU6': {'MAPE': 59.62359792253421, 'Accuracy': 40.37640207746579}

- **Ensemble model with ARIMA, Random Forest, Gradient Boosting, Prophet.**

'SKU1': {'MAPE': 6.725454844298099e+19, 'Accuracy': -6.725454844298099e+19},

'SKU2': {'MAPE': 1.0609801207744612e+19, 'Accuracy': -1.0609801207744612e+19},

'SKU3': {'MAPE': 8.719252794930592e+19, 'Accuracy': -8.719252794930592e+19},

'SKU4': {'MAPE': 2.7614022422728913e+19, 'Accuracy': -2.7614022422728913e+19},

'SKU5': {'MAPE': 2.021129320263402e+19, 'Accuracy': -2.021129320263402e+19},

'SKU6': {'MAPE': 2.9092289663681716e+19, 'Accuracy': -2.9092289663681716e+19}}