3.2 Our Methodology

3.2.1 Data Collection and Preparation

The dataset used for this project is based on a Retail Dataset, Whole Sale Retail Data from Branches and Resellers, was gotten from Kaggle website, that contains information describing individual orders were ordered in each day for about four consecutive years of sales data from 2 Jan. 2019 to 25 Mar. 2023.

The dataset we've got is stored in a csv file and was read as a Data Frame in our model, the dataset includes: invoice id, date, product id and quantity attributes.

	InvoiceID	Date	ProductID	TotalSales	Discount	CustomerID	Quantity
0	328	2019-12-27	1684	796.610169	143.389831	185	4
1	329	2019-12-27	524	355.932203	64.067797	185	2
2	330	2019-12-27	192	901.694915	162.305085	230	4
3	330	2019-12-27	218	182.754237	32.895763	230	1
4	330	2019-12-27	247	780.101695	140.418305	230	4
...	...	...	...	...	...	...	...
29098	11092	2023-01-13	1644	6573.000000	1183.140000	269	1
29099	11093	2023-01-13	352	5179.728814	932.351186	250	4
29100	11094	2023-01-13	683	7741.423729	1393.456271	415	4
29101	11095	2023-01-14	1830	3644.067797	655.932203	59	4
29102	11096	2023-01-14	351	5813.559322	1046.440678	423	4

29103 rows × 7 columns

Figure 3-17: Original dataset

3.2.2 Exploratory Data Analysis (EDA)

Analysis of past transactions data can provide very valuable information on customer behavior and business decisions.

In order to clearly understand the nature of our data, understanding the distributions, the association relationship among the large number of dataset items and describe the patterns of customers' purchase an exploratory analysis is conducted.

Our dataset contains of almost 28945 rows and 4 attributes:

- **Invoice ID:** This column represents the unique identifier for each order.
- **Date:** This column stores the date when each order was placed.
- **Product ID**: This column represents the unique identifier for each Product.
- **Total Sales:** This column refers to the overall monetary value of product sales generated from a particular transaction.
- **Discount:** This column represents the amount of price reduction applied to a product.
- **Customer ID:** This column represents a unique identifier assigned to each customer.
- **Quantity:** This column indicates the quantity of the corresponding item ordered in each transaction.

Initial analysis of data was done graphically, as a time line for weekly orders was plotted, which would ideally give an idea of the kind of preprocessing it would require which comes after EDA.

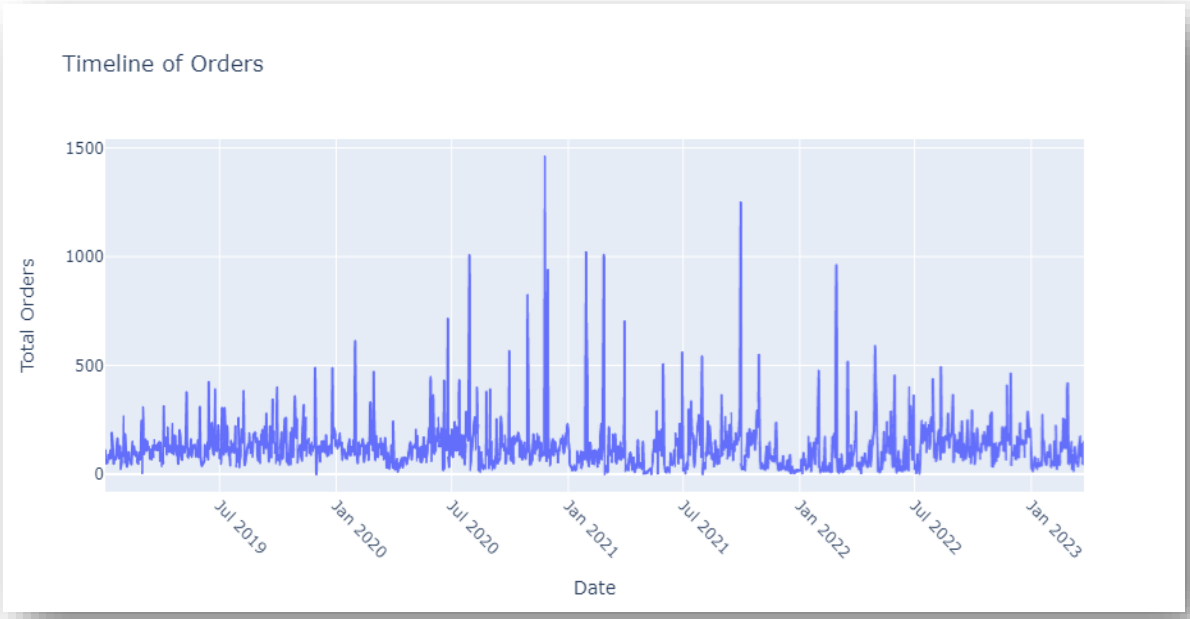

Figure 3-18: Timeline for orders

As can be seen from the figure, for some items, the availability in the dataset is low and this is like a margin of error, a little noisy for us to even bother forecasting, so we will transform this dataset into a weekly dataset starting from the day Monday on each week.

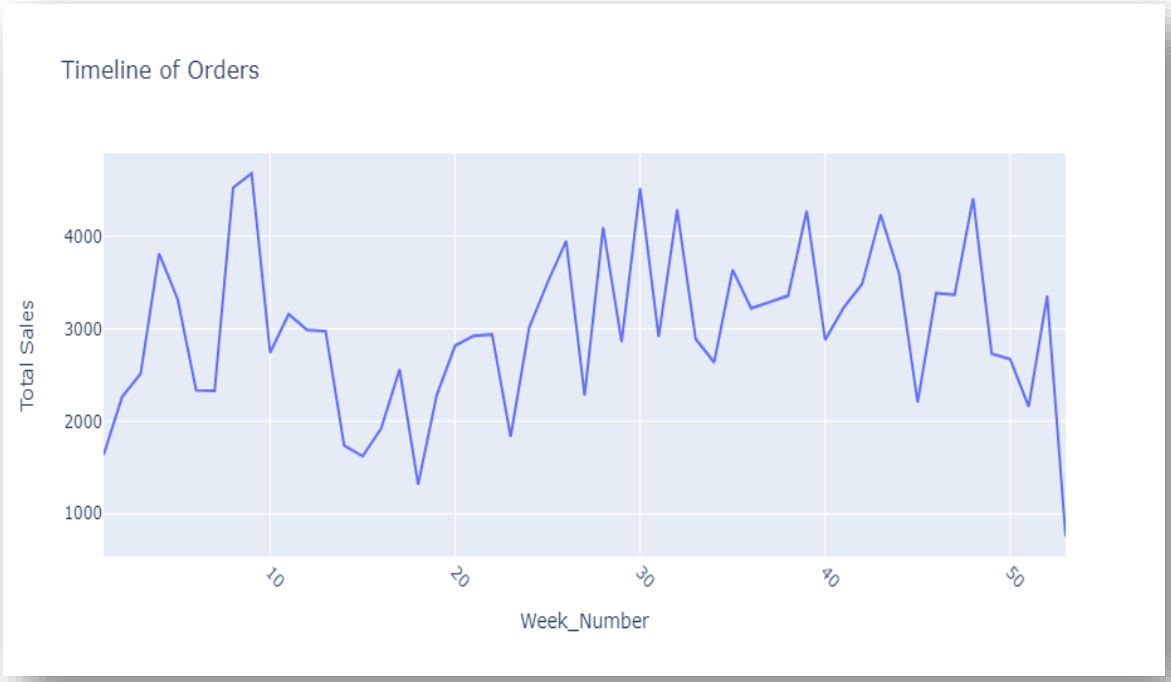

Figure 3-19: weekly sales

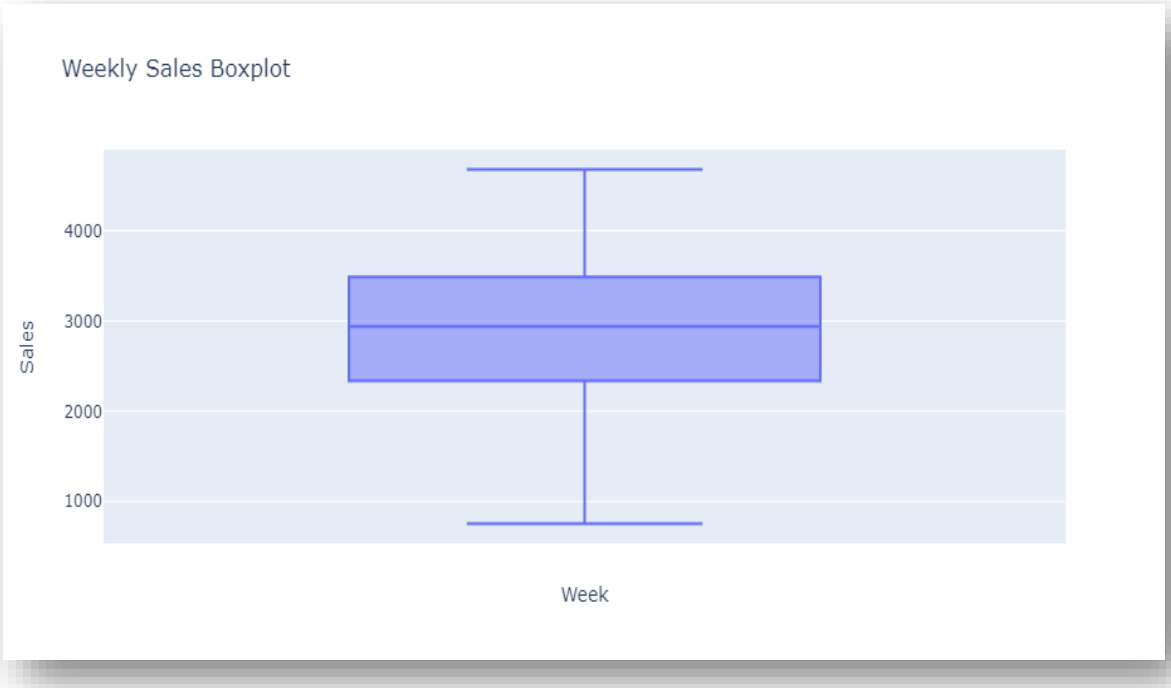

Figure 3-20: Boxplot weekly sales

These figures show the visualization about the number of sales generated in the years from 2019 to 2023 and in the weeks represented as 1, 2, 3, ……, 51, 52 weeks respectively. It demonstrates that the sales in a week is high (which is week 9 total sales = 4677) and a greatest decrease in the sales generated in another week (which is week 53 total sales = 754).

For analyzing order patterns by **day of the week**: we will examine the distribution of orders across different days of the week. This can help us identify any specific days that consistently generate higher or lower sales.

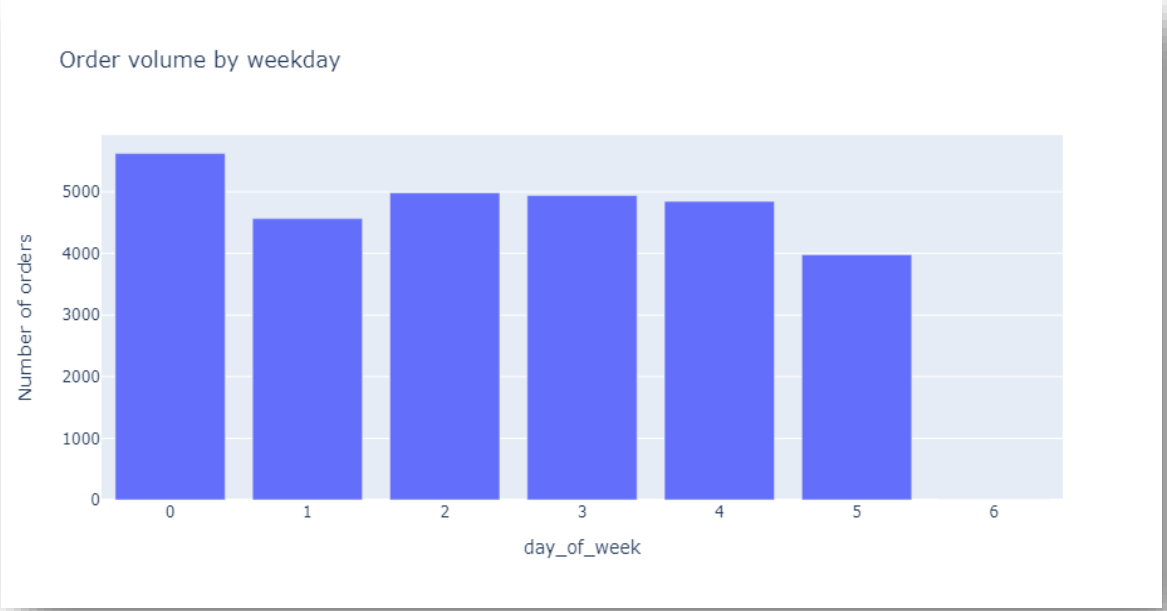

Figure 3-21: Number of orders by weekday

The days of the week are represented in the figure numerically, starting with Monday being represented by the number 0, Tuesday being represented by the number 1, and so on.

As can be seen from the figure, Monday shows an increase in orders, and Sunday is the least.

3.2.3 Data Preprocessing

To prepare the dataset for analysis, several preprocessing steps were undertaken:

- **Data Formatting:** The order date, initially represented as strings, was converted into a standardized Date/Time format to facilitate temporal analysis. The time component was subsequently removed, as it was deemed unnecessary for our forecasting and inventory management tasks.
- **Column Selection:** To streamline the dataset for each analytical task, irrelevant columns were excluded. For item predictions and product placement optimization, essential columns such as Invoice Id, Date, Product Id, and Quantity were retained. Conversely, for order predictions, the Product Id column was omitted as it pertained solely to individual product identification.
- **Handling Missing Values and Zero Quantity Transaction**s: A comprehensive check for missing values was conducted, as specified in the dataset metadata. Fortunately, no missing values were identified. Additionally, transactions with a Quantity of 0 were scrutinized to detect and rectify any anomalies that could impact subsequent analyses.
- **Dataset Transformation:** Dataset was transformed into weekly dataset such that it presents for each item its weekly sales data and daily orders for orders count prediction.

3.2.4 Product Placement Optimization with Apriori

In our methodology, we leveraged the Apriori algorithm for product placement optimization:

Overview of Apriori Algorithm: Apriori is well-suited for mining association rules from transactional datasets, making it a valuable tool for identifying patterns and relationships among products. By analyzing co-occurrences of items in transactions, Apriori generates association rules that indicate potential product affinities and customer purchasing behaviors.

- **How does the Apriori algorithm work?**
  - **Step 1 – Find frequent items:**

It starts by identifying individual items (like products in a store) that appear frequently in the dataset.

  - **Step 2 – Generate candidate itemsets:**

Then, it combines these frequent items to create sets of two or more items. These sets are called “itemsets”.

  - **Step 3 – Count support for candidate itemsets:**

Next, it counts how often each of these itemsets appears in the dataset.

  - **Step 4 – Eliminate infrequent itemsets:**

It removes itemsets that don’t meet a certain threshold of frequency, known as the “support threshold”.

  - **Repeat Steps 2-4:**

The process is repeated, creating larger and larger itemsets, until no more can be made.

- **Find associations:**

Finally, Apriori uses the frequent itemsets to find associations. For example, if “bread” and “milk” are often bought together, it will identify this as an association.

- **Integration with Forecasting Model:**

The association rules derived from Apriori, filtered based on both support and confidence thresholds, were seamlessly integrated into our XGBoost forecasting model. These rules served as actionable insights for strategic product placement decisions, guiding inventory stocking levels and promotional efforts. By incorporating association rule insights into our forecasting model, we enhanced the granularity and accuracy of our sales forecasts, ultimately driving better decision-making and resource allocation.

3.2.5 Features Engineering

Feature engineering involves creating new features, extracting features from the data, understanding the factors the decisions and predictions would be based on or transforming existing ones to improve the performance of machine learning models as well as encoding variables to make them suitable for the chosen algorithm.

- **Lag Features**:

also known as lagged variables or time-shifted features, are a type of feature engineering technique used in time series analysis. They involve creating new features by shifting the values of a variable or multiple variables backward or forward in time.

In a time-series context, a lag feature represents the value of a variable at a previous time point relative to the current time point. By incorporating lag features, you can capture the

historical behavior and temporal dependencies in the data, which can be helpful for predicting future values or understanding the relationship between variables.

The lag features are basically the target variable but shifted with a period of time, it is used to know the behavior of our target value in the past.

in our dataset we created lag features of a day before, a week and a month for predicting the orders count for each day, and for a week and a month before for items predictions.

- **Time-based Features:**

Extracting features based on time components can be helpful in capturing periodic patterns, an extracting of the day of the week, month, day of year, day of month, week of year and year as a numerical representation was done.

- **Time-based Aggregations:**

Aggregating data at different time intervals (daily, weekly, monthly) and calculating summary statistics (mean, max, sum) were done in order to filter our data for both prediction tasks and to transform it in the way which gives the best performance to each task.

And here's a picture shows a sample of our data for orders count prediction after data preprocessing and feature engineering process is done, with each row represents the Orders count column which is the column to be predicted, the lag features, and time-based features.

Orders_...	Day_of_Week	Day_of_Year	Month	Year	Week_N...	Day_of_Mo...	Prev_Day_...	Prev_Week...	Prev_M...	Rolling_Av
25	0	7	1	2019	2	7	0	0	0	0
28	0	14	1	2019	3	14	0	25	0	22.5714
32	0	21	1	2019	4	21	0	28	0	19.4285
27	0	28	1	2019	5	28	0	32	0	19.1428
27	0	35	2	2019	6	4	0	27	12	26.5714
22	0	42	2	2019	7	11	0	27	39	17.4285
22	0	49	2	2019	8	18	0	22	24	21.8571
29	0	56	2	2019	9	25	0	22	4	16.7142
17	0	63	3	2019	10	4	0	29	34	21.8571
26	0	70	3	2019	11	11	0	17	20	22.1428
30	0	77	3	2019	12	18	0	26	26	24
34	0	84	3	2019	13	25	0	30	16	24.4285
14	0	91	4	2019	14	1	0	34	55	24
43	0	98	4	2019	15	8	0	14	32	29.2857
30	0	105	4	2019	16	15	0	43	22	25.4285

Figure 3-22: Sample of dataset for orders count prediction

And here's a picture shows a sample of our data for items prediction after data preprocessing and feature engineering process is done, with each row represents the Quantity column which is the column to be predicted, the lag features, and time-based features.

ProductID	Quant...	Week	Month	Year	Quantity_Lag_1	Quantity_Lag_2	Monthly_Sum	Monthly_Avg	min	max
628	0	23	6	2022	12	0	4	1	0	4
628	0	24	6	2022	0	12	4	1	0	4
628	0	25	6	2022	0	0	4	1	0	4
628	4	26	6	2022	0	0	4	1	0	4
628	0	27	7	2022	4	0	6	1.5	0	4
628	2	28	7	2022	0	4	6	1.5	0	4
628	4	29	7	2022	2	0	6	1.5	0	4
628	0	30	7	2022	4	2	6	1.5	0	4
628	16	31	8	2022	0	4	28	5.6	0	16
628	0	32	8	2022	16	0	28	5.6	0	16
628	4	33	8	2022	0	16	28	5.6	0	16
628	0	34	8	2022	4	0	28	5.6	0	16
628	8	35	8	2022	0	4	28	5.6	0	16
628	0	36	9	2022	8	0	104	26	0	62
628	6	37	9	2022	0	8	104	26	0	62

Figure 3-23: Sample of dataset for item sales prediction

3.2.6 Machine Learning Model & Prediction

Prediction deals with events occurring in the future. The use of Machine learning algorithms improves the intelligence of the system without manual intervention. “Machine Learning (ML) is used to optimize the performance criterion using sample data or the past experience” as defined by Ethem Alpaydin.

The ML algorithms are classified in three categories. They are supervised, unsupervised and semi supervised.

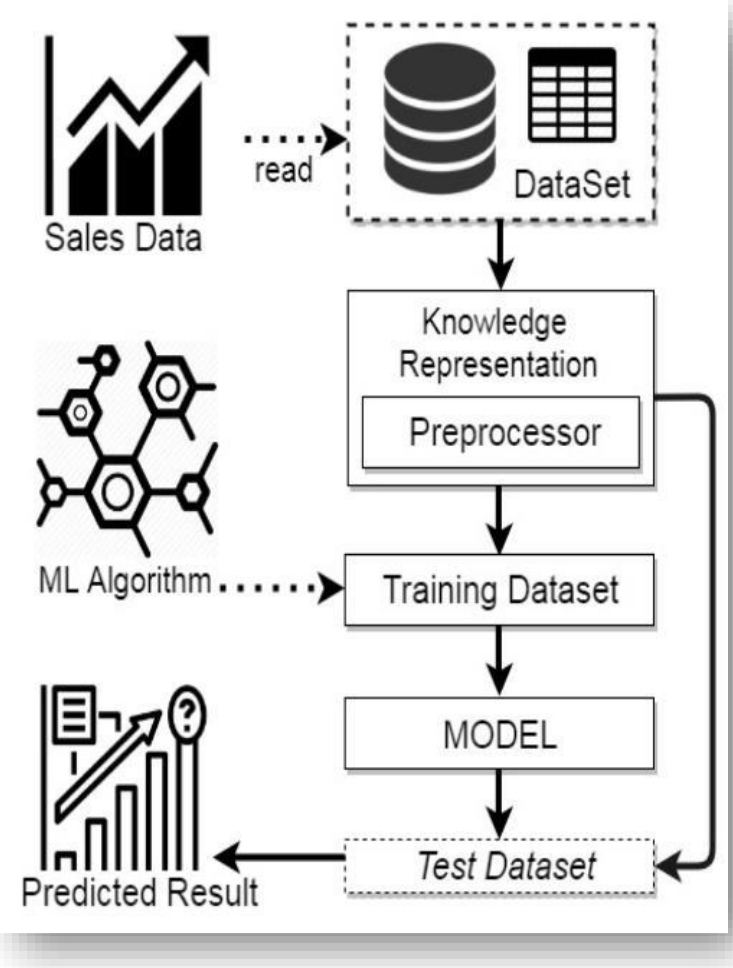

Figure 3-24: System architecture

The dataset for **orders counts prediction model** was split into 90% training data and 10% test data for getting the accuracy, the test data is the data after the date 21-10-2022 and before that date the data used for training the model.

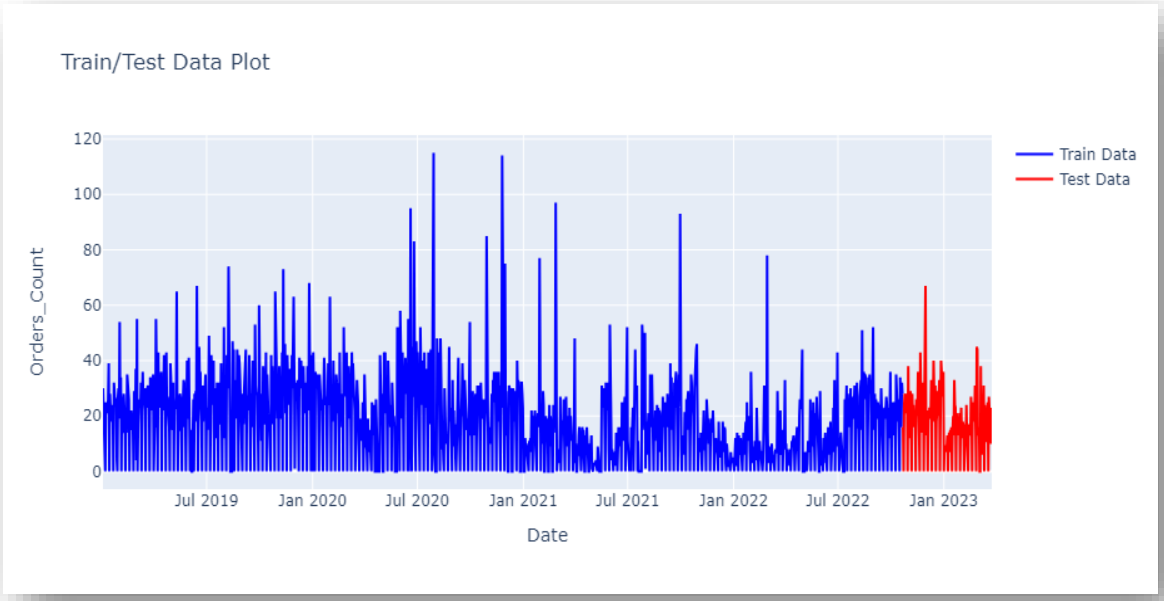

Figure 3-25: train-test data split for orders

The dataset for **items prediction model** was split into 80% training data and 20% test data for each item, the test data is the data after the date 9-5-2022 and before that date the data used for training the model.

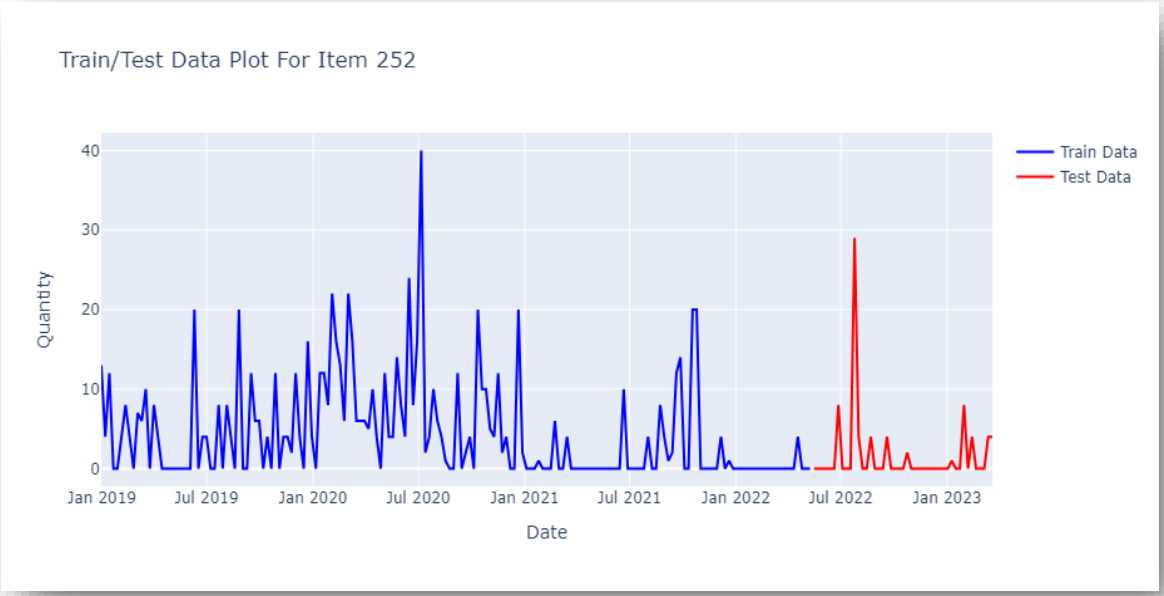

Figure 3-26: train-test data split for item 252

In this project, we implemented eXtreme Gradient Boosting (XGBoost) model on the training dataset and the model is then tested for the performance.

3.2.7 eXtreme Gradient Boosting (XGBoost) Model

XGBoost Regressor is a machine learning algorithm that is used for regression tasks. It is based on the gradient boosting framework, and it builds an ensemble of decision trees to make predictions. XGBoost Regressor is similar to XGBoost Classifier, except that it is designed for continuous target variables, rather than categorical ones.

The XGBoost Regressor works by building decision trees sequentially, where each subsequent tree is trained to reduce the errors of the previous tree. The algorithm learns to predict the target variable by combining the predictions of all the decision trees in the ensemble.

The XGBoost Regressor has the ability to handle complex nonlinear relationships between the features and the target variable. It also has a number of hyperparameters that can be tuned to improve its performance, including the learning rate, number of trees, maximum depth of each tree, and regularization parameters.

How does the model work:

- **Step 1: Initializing the model**

We'll start by initializing the XGBoost model with some default parameters such as the learning rate, maximum depth of trees, and the number of trees in the ensemble.

- **Step 2: Fitting the initial model**

We'll fit the initial model using the training data.

Calculate the mean value (y_mean) of the target variable (y) in the training set.

- **Step 3: Calculating the residuals**

Calculate the residuals (r) by subtracting the predicted values (ŷ) from the actual values (y) in the training set**:** $\mathbf{r} = \mathbf{y} - \hat{y}$**…… Equation (3.2.1)**

The residuals represent the errors made by the initial model.

- **Step 4: Building a decision tree to predict the residuals**

The decision tree will take the features (X) and the residuals (r) as inputs and output the predicted residuals (r_pred).

Calculating the similarity values (S) based on the second-order derivative of the loss function:

$\mathbf{Similarity\ Score} = (\mathbf{Sum\ of\ residuals})\ \mathbf{\hat{}2} / (\mathbf{Number\ of\ residuals} + \mathbf{lambda})$**……. Equation (3.2.2)**

- **Step 5: Calculate the gain to determine how to split the data.**

Calculating the gain for each leaf node in the pruned decision tree using the similarity values:

Gain = Left tree (similarity score) + Right (similarity score) - Root (similarity score)

Determine the optimal splitting points in the tree based on the calculated gain.

- **Step 6: Pruning the tree**

Prune the decision tree by evaluating the gain (Gain) for each potential split and comparing it with gamma.

Calculating the difference between Gain and gamma (user-defined tree-complexity parameter) **Gain - gamma** If the result is a positive number, then do not prune and if the result is negative, then prune (stop splitting and convert the node into a leaf) and again subtract gamma from the next Gain value way up the tree.

- **Step 7: Calculate output value for the remaining leaves**

Calculating the output of the current tree by taking the sum of the updated residuals and dividing it by the count of the updated residuals plus lambda:

$$\mathbf{Tree\_output} = \mathbf{Sum\ of\ residuals} / (\mathbf{Number\ of\ residuals} + \mathbf{lambda}) \quad \text{.....}$$

Equation (3.2.3)

Note: If the value of lambda is greater than 0, it results in more pruning by shrinking the similarity scores and it results in smaller output values for the leaves.

Update the model predictions by adding the output of the decision tree to the previous predictions:

$$\mathbf{Output} = \mathbf{initial\ prediction} + \boldsymbol{\varepsilon} * (\mathbf{tree\_1\_output} + \mathbf{tree\_2\_output} + \ldots + \mathbf{tree\_n\_output})$$ **..... Equation (3.2.4)**

Where $\boldsymbol{\varepsilon}$:$\boldsymbol{learning\ rate}$.

- **Step 8: Update the residuals**

Update the residuals by subtracting the predicted output (y_pred) from the actual output(y).

The updated residuals (r_updated) are calculated as $\mathbf{r\_updated} = \mathbf{y} - \mathbf{y\_pred}$. ..... **Equation (3.2.5)**

- **Step 9:** Repeat steps 4-8 (tree building, pruning, model updating, and residual updating)

Repeat steps 4-7 for a specified number of iterations or until a stopping criterion is met.

At each iteration, build a new decision tree to predict the updated residuals, update the model predictions, and update the residuals.

- **Step 10: Finalize the model**

After the specified number of iterations, finalize the model and obtain the final predictions.

- **Step 11: Make predictions on new data**

Use the finalized model to make predictions on new unseen data by feeding the features (X) into the model, using the equation (3.2.4):

Output = initial prediction + ε * (tree_1_output + tree_2_output + ... + tree_n_output)

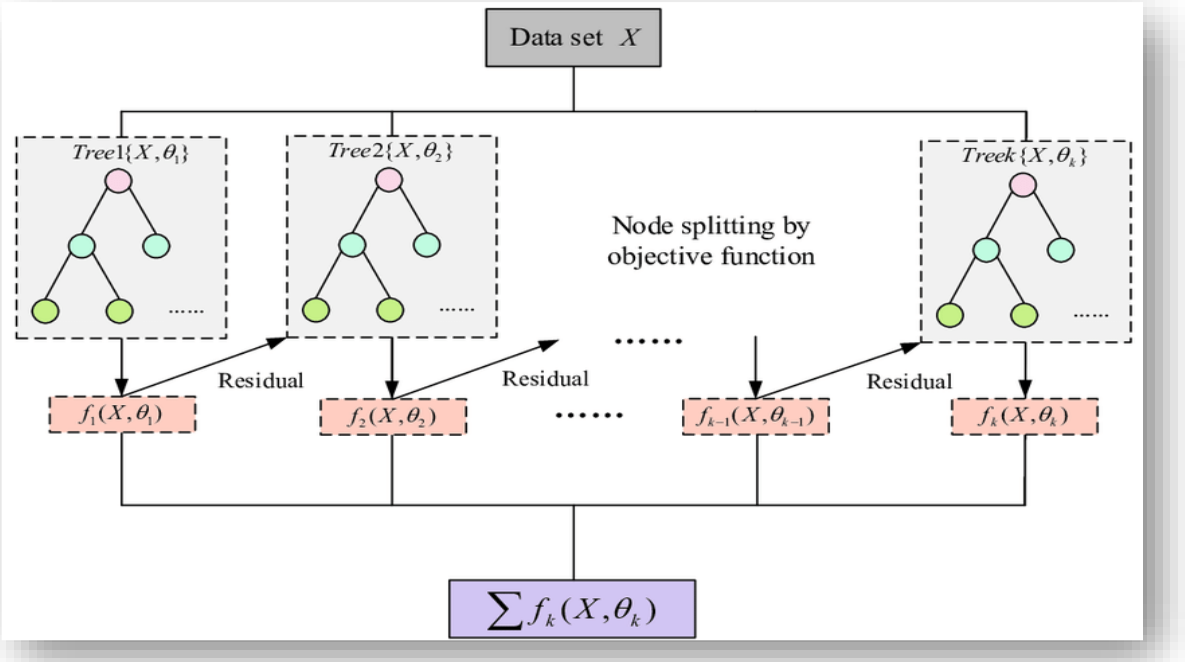

Figure 3-27: The XGBoost Flow-chart

Finding the optimal hyperparameters for the XGBoost model, thereby improving its performance and generalization ability on unseen data, we utilized GridSearchCV. It's a powerful tool for fine-tuning machine learning models and is commonly used in practice to optimize model performance.