Capstone Team 3

Demand Forecasting for Brazilian Logistic Company using Machine Learning Models & their Comparative Analysis

Capstone Consultant	Team3
ShashankPullela	Hussnain Ashraf
TechnicalConsultant	Kundan Kumar
PriyaTony	Kundan Kumar
Vishnu D	Sakshi Gupta
Sandeep Vijay	Shailendra Kumar Ravi

Index

3. Dataset:
4. Exploratory Data Analysis (EDA)
5. Architecture Diagram / Approach
6. Machine learning model

   Approach 1:

   6.1 Catboost

   6.2 Adaboost

   6.3 XGBoost

   6.4 Lasso Regression

   Approach 2: Time series

   6.5 Arima

   6.6 HWES

   6.7 Sarimax

   6.8 Sarimax With regressors

   6.9 Prophet

   6.10 Prophet with regressors
7. Evaluation Metrics
8. Visualizations and comparative analysis/Result
9. Conclusion

1. Problem Statement

For Brazilian Logistic Company, build a Model that forecasts the demand for daily treatment.

2. Problem understanding

Context:

Brazilian Logistics Company: This company likely handles various transportation, storage, or delivery services.

Importance of Demand Forecasting: Accurate predictions help in optimizing resources, managing inventory, and improving operational efficiency.

Dataset Overview:

Span and Source: The dataset covers 60 Business days and originates from the company's database.

Attributes: Contains 12 predictive features along with a target variable representing daily orders.

Objective: Forecasting Daily Orders: The primary goal is to predict the number of orders the company will receive each day.

3. Dataset Given:-

The dataset, titled "Daily Demand Forecasting Orders," was collected over a span of 60 days and originates from a real database of a prominent Brazilian logistics company. It consists of twelve predictive attributes and a target variable representing the total number of daily orders, making it suitable for daily demand forecasting.

Attribute of Dataset

Week_of_the_month {1.0, 2.0, 3.0, 4.0, 5.0}	Fiscal_sector_orders
Day_of_the_week(Monday_to_Friday) {2.0, 3.0, 4.0, 5.0,6.0}	Orders_from_the_traffic_controller_sector
Non_urgent_order integer	Banking_orders_(1)
Urgent_order	Banking_orders_(2)
Order_type_A	Banking_orders_(3)
Order_type_B	Target_(Total_orders)
Order_type_C	

This dataset was utilized in academic research at the Universidade Nove de Julho, emphasizing its relevance and suitability for scholarly investigations in the domain of logistics and demand forecasting. Researchers and data analysts can leverage this dataset to develop predictive models and gain insights into the factors influencing daily order volumes in the logistics sector. In summary, this dataset serves as a valuable resource for studying and modeling the intricate patterns of daily demand in the logistics industry. Dataset is available in excel and csv format.

4. Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is used to analyze and investigate datasets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to

manipulate data sources to get the answers you need, making it easier to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

In EDA we have done following step to explore the dataset:

3.1 Univariate Analysis: In the initial part analysis of data, we found the number of rows and columns in the dataset. The dataset consists of 60 rows and 14 columns. After that we get a concise summary of the dataset structure and content. Data Frame appears to contain information about orders across different sectors, categorized by week, day, and order type. The non-null counts for all columns are 60, indicating a complete dataset without missing values. Columns in dataset are either integer or float type.

In the next step we have done statistical analysis. Statistical summary shows that the average week of the month is approximately 3, with a moderate level of variability, suggesting a relatively consistent distribution of orders throughout the month. Similarly, the average day of the week is around 4, indicating a balanced distribution across weekdays. Analysis of different order types reveals that non-urgent orders tend to be larger on average than urgent orders, and order type C has the highest mean value among the three types. Sector-wise examination highlights the considerable variability in fiscal sector orders, indicating potential financial fluctuations, while orders from the traffic controller sector exhibit a more stable distribution. Banking orders, particularly type 2, show a broader range of values, suggesting diverse transaction sizes within this category. Order values for banking type 3 are comparatively lower on average. The overall analysis provides an understanding of the data.

After that we perform data cleaning on the dataset. First, we checked the null values in the dataset and found that there is no null value in the dataset.

```
df.isnull().sum() #no null values found
```

```
Week of the month (first week, second, third, fourth or fifth week    0
Day of the week (Monday to Friday)                                    0
Non-urgent order                                                      0
Urgent order                                                          0
Order type A                                                          0
Order type B                                                          0
Order type C                                                          0
Fiscal sector orders                                                  0
Orders from the traffic controller sector                             0
Banking orders (1)                                                    0
Banking orders (2)                                                    0
Banking orders (3)                                                    0
Target (Total orders)                                                 0
```

Fig.01

Then we examined the duplicate row in the dataset and found that there is no duplicate row in the dataset.

```
print(f'No. of Duplicate rows: {df.duplicated().sum()}')
d = df.duplicated()
d.value_counts()
```

```
No. of Duplicate rows: 0
False    60
dtype: int64
```

Fig.02

Continuation of data cleaning process we perform data validation on 'Week of the month (first week, second, third, fourth or fifth week' and 'Day of the week (Monday to Friday)'. This validation ensures that days are within valid date ranges in these two columns.

3.2. Bivariate Analysis: After completing basic dataset analysis, we perform bivariate analysis. It involves the analysis of two variables simultaneously to understand the relationships between them. It is a fundamental step in statistical analysis and data exploration, providing insights into how two variables interact or correlate with each other. In the first step of bivariate analysis, we present a bivariate analysis of the selected columns using boxplots. The goal is to visually explore the distribution and potential outliers in the specified columns.

3.2.1 Boxplots: The boxplots are presented in a 3x6 grid, where each subplot corresponds to one of the selected columns. A set of boxplots has been generated for each column to provide insights into their distribution. Each boxplot includes: **-** Box: Represents that the box represents the range between the first quartile (Q1:25%) and the third quartile (Q3:75%), providing insights into the spread of the central 50% of the data. Outliers: Individual data points lying outside the whiskers.

Following are few insights from box plot which we get of every attribute in the dataset.

i. **Non-Urgent order, Urgent Order:** The boxplot suggests a relatively concentrated distribution with minimal outliers, range of Q1 and Q3 is from A wider interquartile range is observed, indicating greater variability in the data.
ii. **Order Type A, B and C:** Each order type exhibits a distinct distribution, with potential outliers.
iii. **Fiscal Sector Orders:** Most of the data is concentrated within a narrow range having most of the values around zero, with maximum outliers.
iv. **Banking orders (1), (2) and (3):** Varied distributions are observed for each banking order type. In the Banking Order (1), narrow range is observed, indicating most of the data having smaller variability. Outliers are more as compared to the other Column Parameter. Banking Orders (3) is the best data as per our observations as it has lowest Outliers comprising whole data.
v. **Orders from Traffic Controller sector:** The distribution appears relatively uniform, with no outliers.

vi. **Target (Total Orders):** A boxplot of the target variable shows the spread and potential outliers.

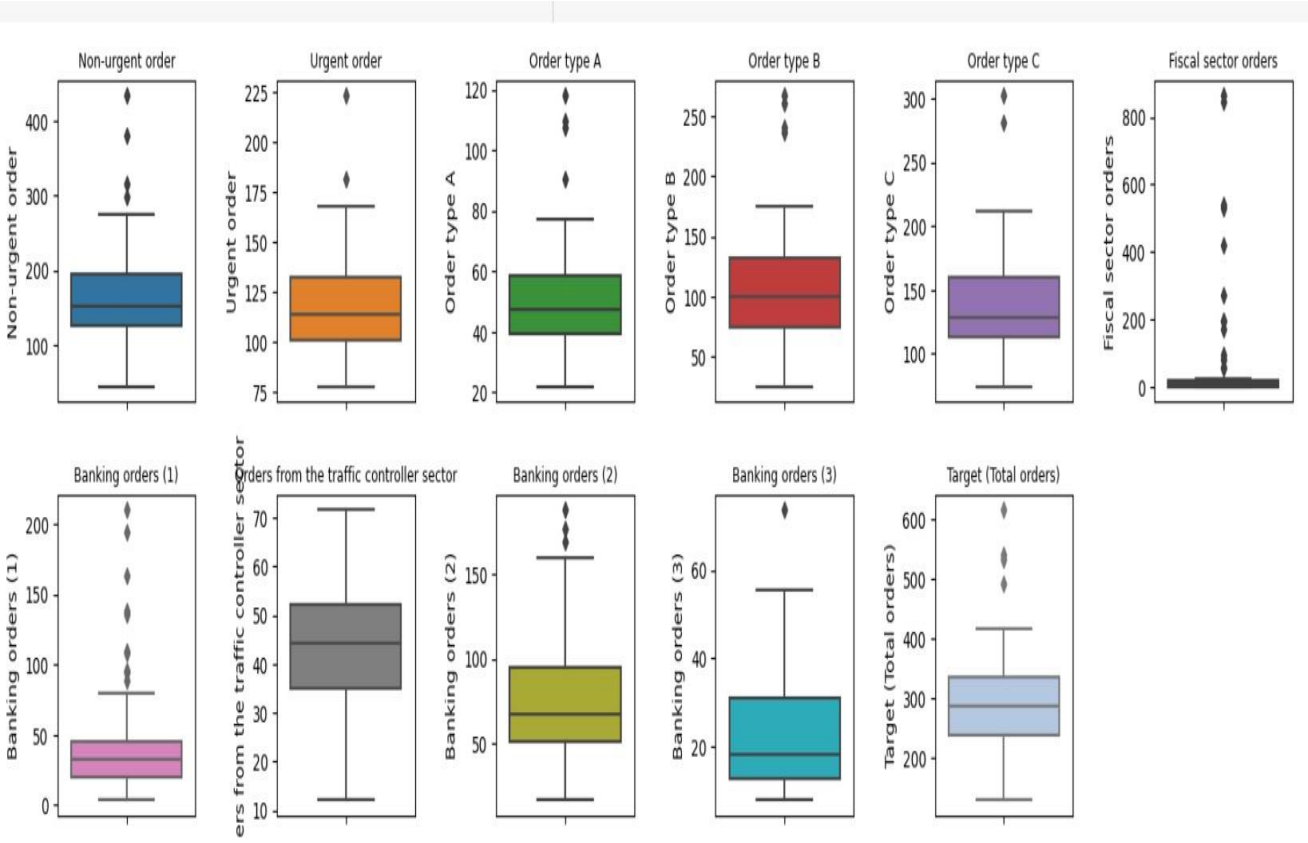


Fig. 03

3.2.2 Pair Plot Analysis Summary: Objective of pair plot analysis is seeing relationship between the of all independent with dependent variable Target (Total Orders). Independent variable are 'Urgent order,' ‘Order Type A, B and C’, ‘Fiscal Sector Orders’, ‘Banking orders (1), (2) and (3)’, ‘Orders from Traffic Controller sector’ and 'Total Orders'

Key Findings:

a) Positive correlation observed between 'non-urgent order', 'Urgent order' and 'Total Orders', though strong positive for 'non-urgent order' and Moderate positive for 'Urgent order’.
b) Order type (A), (B) and (C) showing positive correlation, strong positive in (B), moderate in (C) and week for the (A).
c) There is No correlation in banking orders (3) and Fiscal sector Orders, while negative correlation is observed in ‘Orders from Traffic Controller sector’.
d) Outliers detected in the scatter plots.

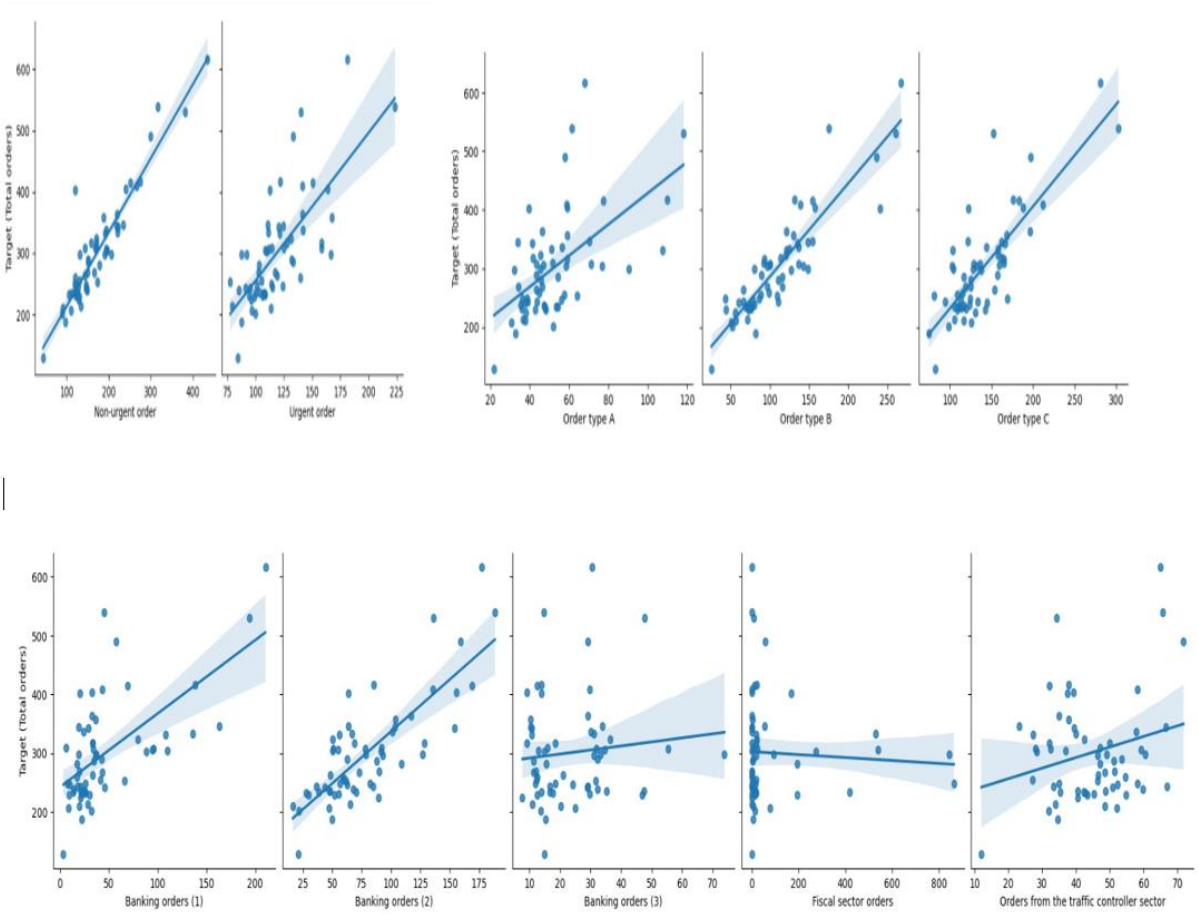


Fig. 04

3.2.3 Bar Plot Analysis Summary:

Objective: The objective of this bar plot analysis is to examine the variation in 'Total Orders' considering both the 'Day of the week (Monday to Friday)' and the 'Week of the month (first week, second, third, fourth or fifth week)'. This analysis specifically focuses on analysis in different categories i.e. 'Non-urgent', 'Urgent orders' and 'Order Type A,' 'Order Type B,' and 'Order Type C.' These categories establish direct connection between with Taret Order

Data Preparation: The dataset was transformed into a long format to facilitate visualization. The melted dataset includes variables such as 'Day of the week,' 'Week of the month,' 'Order Type,' and 'Total Orders.'

Key findings: Day-wise Analysis

i. In day-wise analysis generally, 'Non-urgent orders' and 'Urgent orders' showing peeks significantly on **Monday in Day of the week while in Week of Month in fifth week.**
ii. 'Urgent orders' show relatively consistent patterns across the week (2 to 6) and Week of the Month.
iii. Order type A showing almost consistent patterns of ‘Total Orders’ across the “Day of the week” and “Week of the Month”.
iv. 'Order Type B' and 'Order Type C' show higher 'Total Orders' during weekdays.
v. 'Order Type B' exhibits relatively consistent patterns across different days of the week.

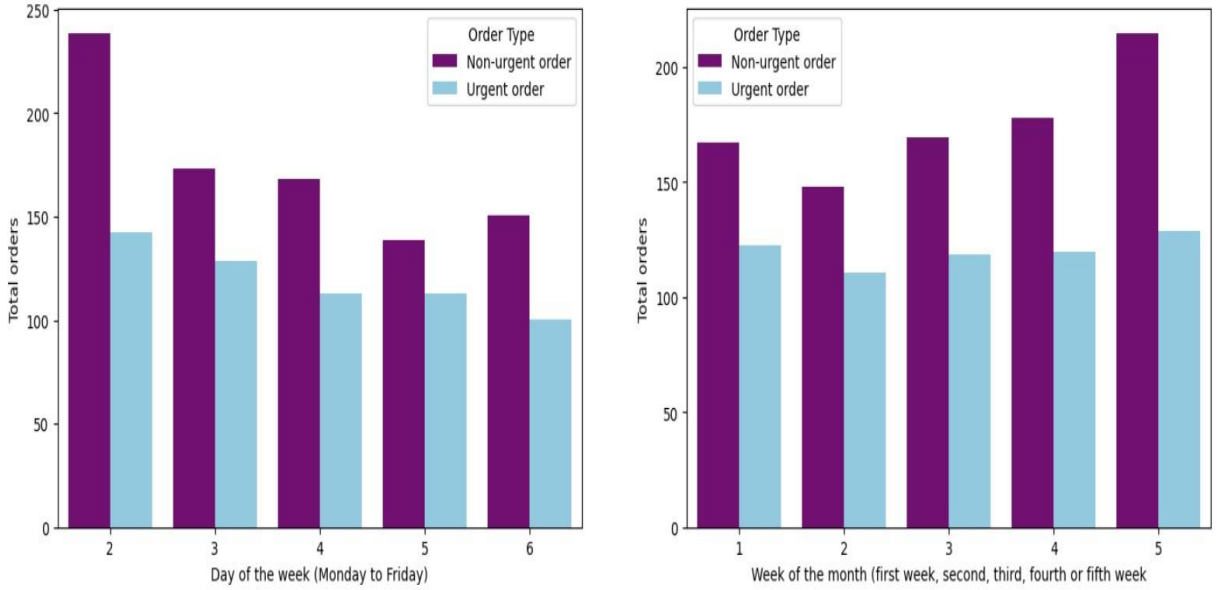


Fig 05

Key finding: Week-wise Analysis:

i. 'Total Orders' exhibit fluctuations across different weeks of the month.
ii. 'Non-urgent orders' demonstrate varying trends during different weeks.
iii. 'Urgent orders' maintain relatively stable patterns throughout the month.
iv. 'Total Orders' demonstrate fluctuations across different weeks of the month for all three order types.
v. 'Order Type B' exhibits more stability in weekly patterns compared to 'Order Type A' and 'Order Type C.'

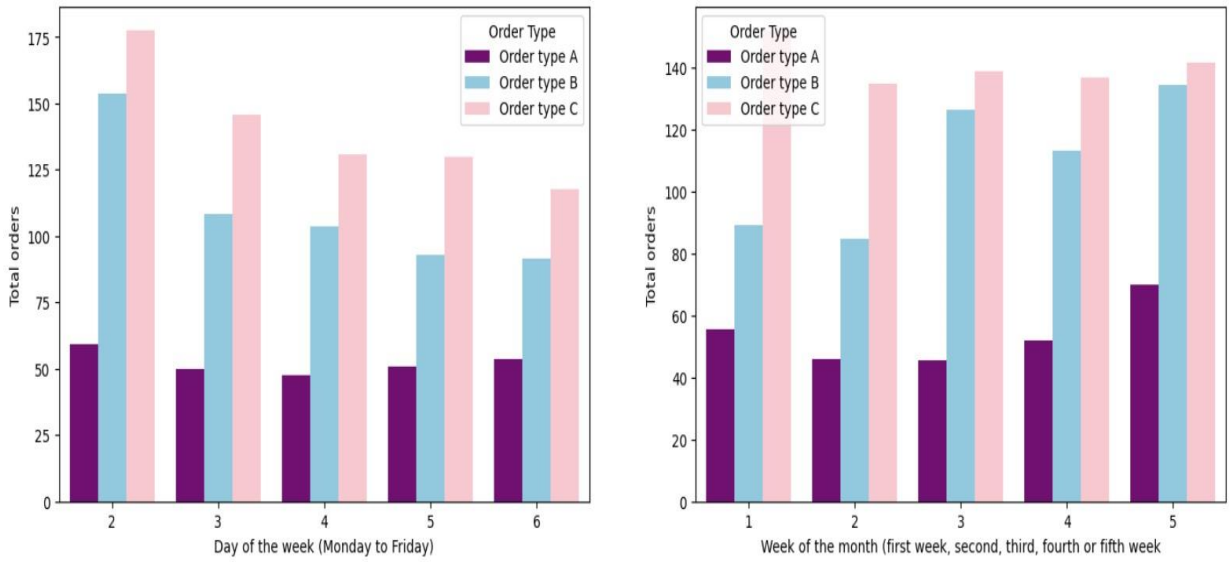


Fig. 06

3.2.4 Histogram Analysis Documentation: The objective of this histogram analysis is to explore the distribution of different variables against the number of such type of order to show the demand of each attribute. The dataset contains various columns representing different types of orders, sector orders, and the Count of those orders. The histograms are used to visualize the frequency distribution and patterns of each variable.

Key Findings:

1. We have shown distribution of Category Order: ‘Urgent Order’ and ‘Non-urgent Order’; Order Types: 'Order type A,' 'Order type B,' and 'Order type C'; Sector Orders: 'Fiscal sector orders,' 'Banking orders (1, 2, and 3),' and 'Orders from the traffic controller sector'; Total Orders:
2. In the Sector Orders: Orders from Traffic controller sector having normal distribution of data.
3. Fiscal sector orders are completely flat lying at x-axis line (y=0), banking orders 2 followed by 1 and 3 normal distributions of the data.

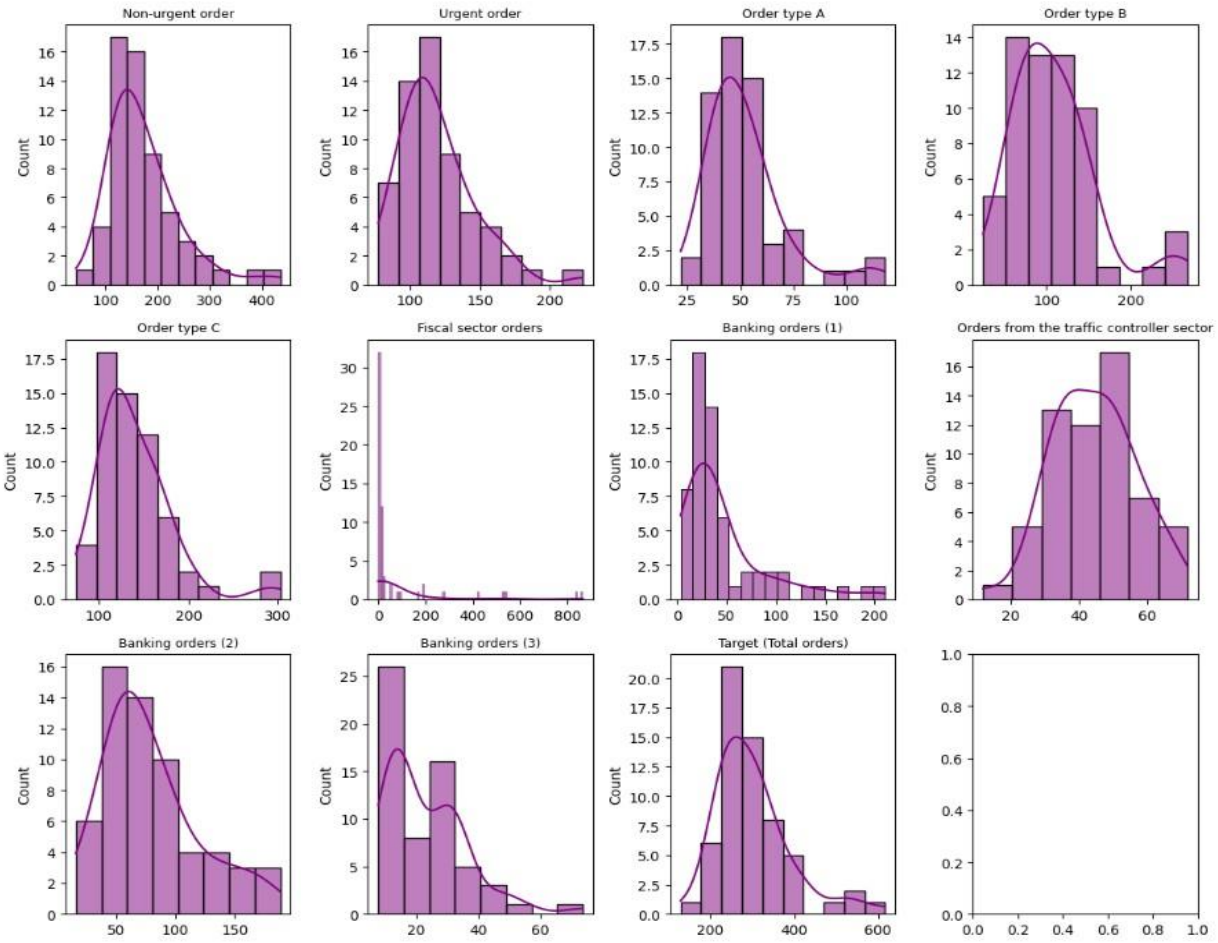


Fig. 07

3.2.5. Correlation Heatmap Analysis: A correlation heatmap is a visual representation of the correlation matrix, which shows the pairwise correlations between different variables in a dataset. It is particularly useful when dealing with multiple variables, allowing you to quickly identify patterns and relationships. These insights can guide decision-making processes, feature selection for modelling, and a deeper understanding of the dataset's interdependencies. Correlation is two type **Positive and Negative correlations:** Positive correlations are indicated by values close to 1, suggesting that as one variable increases, the other tends to increase as well. Negative correlations are indicated by values close to -1, suggesting that as one variable increases, the other tends to decrease.

Objective: The heatmap provides a visual representation of the strength and direction of relationships between variables.

Data and Visualization: The dataset contains various columns representing different types of orders, sector orders, and the target variable 'Total Orders.' The correlation heatmap is employed to visualize the pairwise correlations between these variables.

Key Findings:

i. Positive correlation between 'Order type A' and ‘Total Orders’
ii. Negative correlation between 'Urgent order' and ‘Order type C’
iii. Darker shades in the heatmap represent stronger correlations. Example: Strong positive correlation between 'Fiscal sector orders' and 'Banking orders (2).

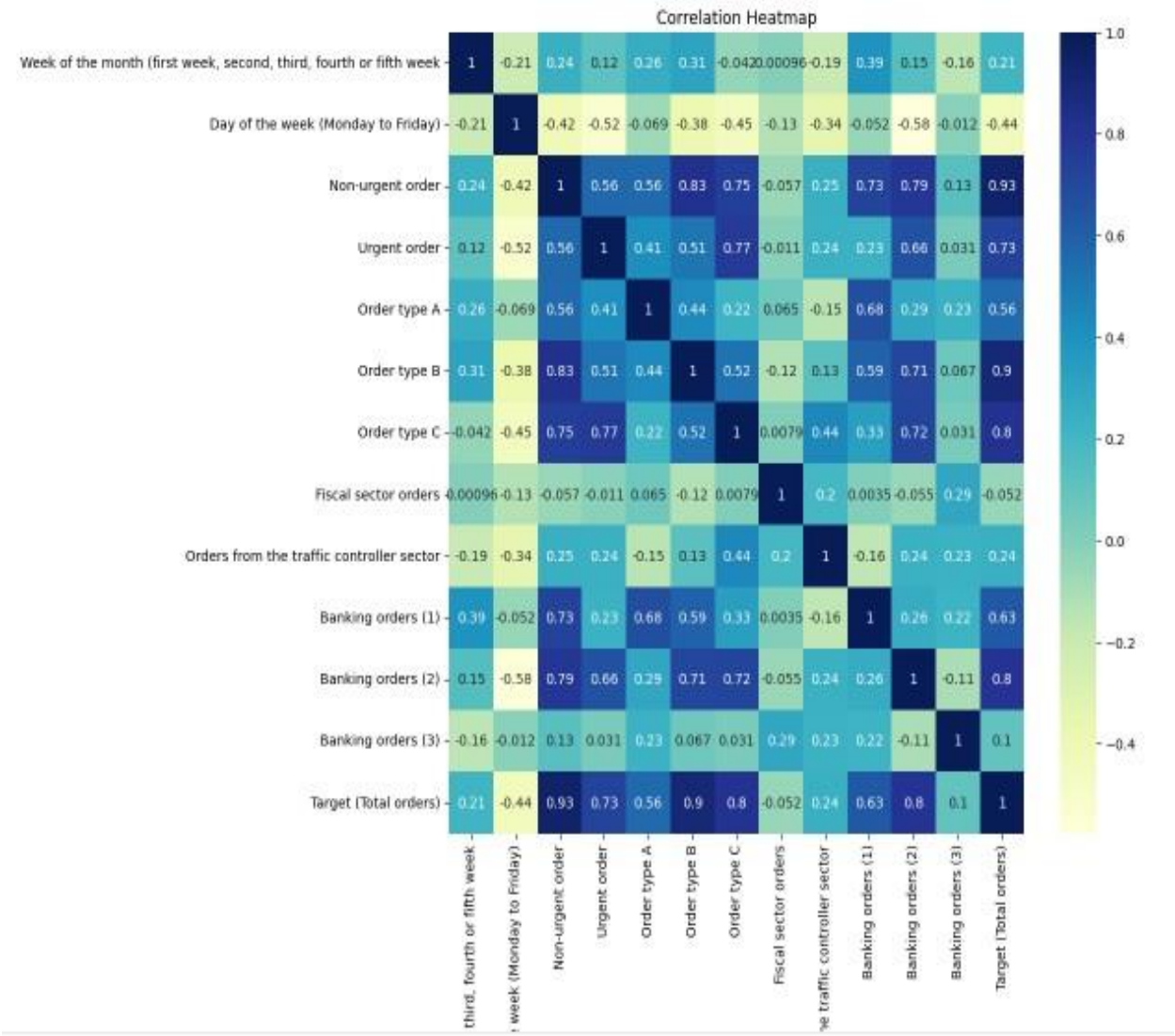


Fig.08

5. Architecture Diagram / Approach

An architecture diagram is a visual representation that illustrates the structure, components, activity, relationships, and interactions within a system or application. It provides a high-level view of the system's design and helps understand the overall architecture.

In this architecture diagram, we are depicting the components of our solution approach. The major components include the EDA, supervised learning, Machine learning Model, deployment, and visualization.

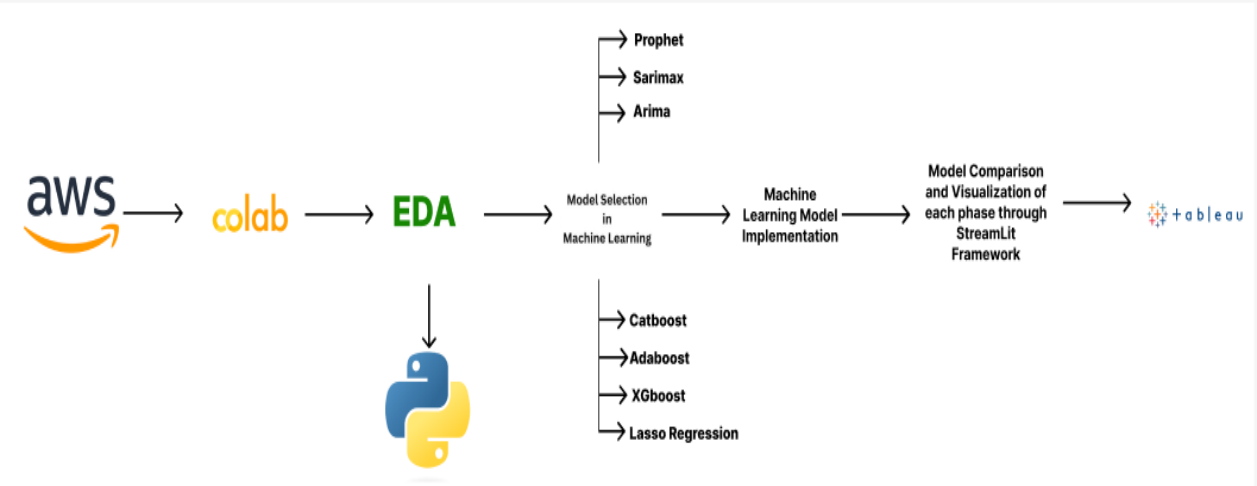


Fig. 09

In Brazilian Data Exploration activity, we done following activity: -

i. We explore the data looking for missing values, analyzing the data visualization, etc. This activity is comprised of Data cleaning and EDA activity. In Data cleaning we look for duplicate, missing, null value and data normalization. In EDA we look for the relation between different features and their behaviors and trends as well as correlation.

ii. In the next step we see whether data is supervised or unsupervised. Our data is supervised learning data then we move to check whether data is continuous or categorical type. After knowing that our data is continuous, we move forward to check whether data has time factor or not. Since our data have Time factor also, we move forward with 2 approach one with time series and Regression approach. Using these approaches, we selected 5 different models for forecasting. During exploration of machine learning model, we come up with following models which can be used to for forecasting: -

   a. Adaboost
   b. XGBoost
   c. Lasso Regression
   d. Catboost and LSTM

   As our problem is forecasting problem so we also explore following time series model: -

a. Arima
b. HWES
c. Sarimax
d. Sarimax With regressors
e. Prophet
f. Prophet with regressors

iii. In the next step we perform model learning and implementation in python to see how these models can work in forecasting.

iv. Then for every model we generated the next 30 days forecast data for every model which worked effectively in csv format.

v. Next step after forecasting comparison of models is using metric parameter or visualization. For metric parameters we have chosen R^2 score and RMSE.

vi. At last, we generate insight and answer business questions we done visualization in tableau.

Below diagram show architecture diagram of our solution approach

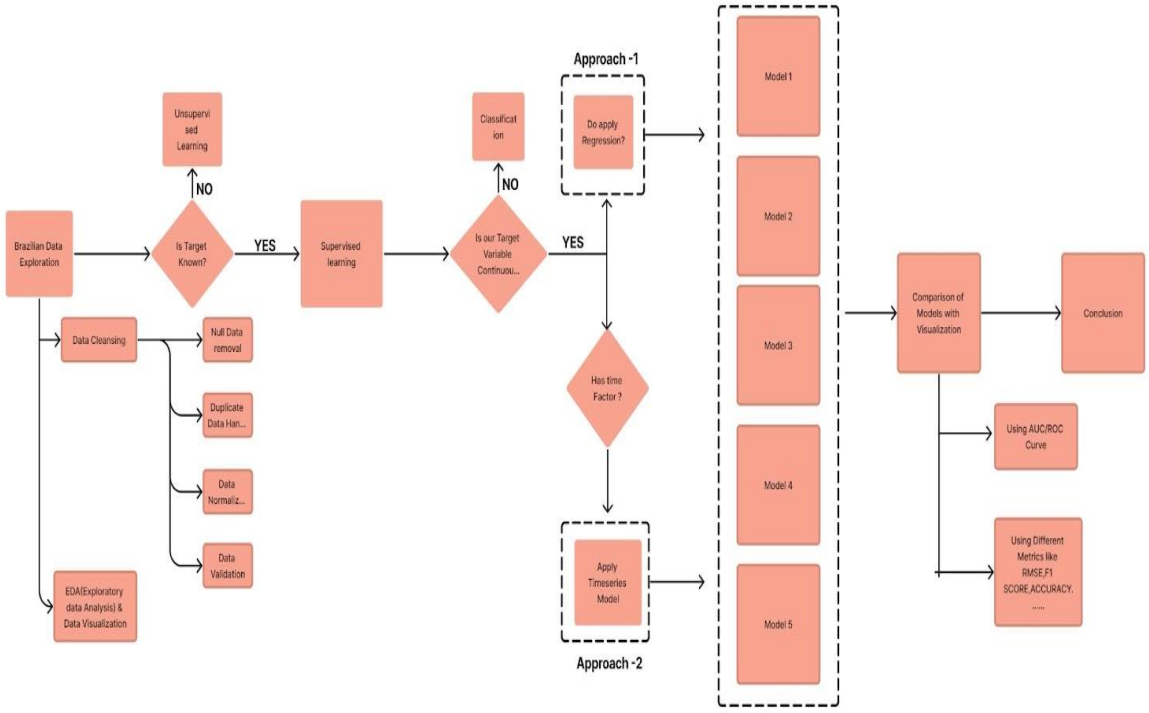


Fig. 10

6. Machine Learning Method

6.1 Catboost

Model selection and description: CatBoost Regressor is a gradient boosting framework designed for regression tasks, meaning it predicts continuous target values. Here's a quick introduction:

- Handles diverse data: CatBoost thrives on mixed data types, including categorical features, without needing extensive pre-processing.
- Boosting power: It combines decision trees in a sequential manner, improving accuracy with each iteration.
- Customization: A rich set of hyperparameters allows tailoring the model to specific tasks and data sets.

Model Implementation: The model is divided into training and test datasets, with 80% of the data allocated to the training dataset and the remaining 20% to the test dataset. GridSearchCV is employed to determine the optimal hyperparameters for the model. After training the model, the output is predicted and visually represented through a plot. The Mean Squared Error (MSE) and R-squared (R2) values are also calculated to assess model performance. Subsequently, the time series data is stored in X, while the feature is dynamically changed with each iteration in y. The predicted values are then saved to a CSV file and visualized using appropriate visualization tools.

Hyperparameter Tuning for Optimal Performance: For hyperparameters tuning in Catboost we have includes the number of weak learners (n_estimators), the learning rate for each learner and max depth. The tuning process aims to find a balance between model complexity and avoiding overfitting.

Results: As it is time series forecasting and we are taking each feature at a time and forecasting so for each we have R-square value and Root Mean Square Error. Few features R-square value and Root Mean Square Error are shown below.

```
MAE TEST: 88.97538279253622
R2 TEST: -1.9511056516751681
RMSE: 103.49678876252321
```

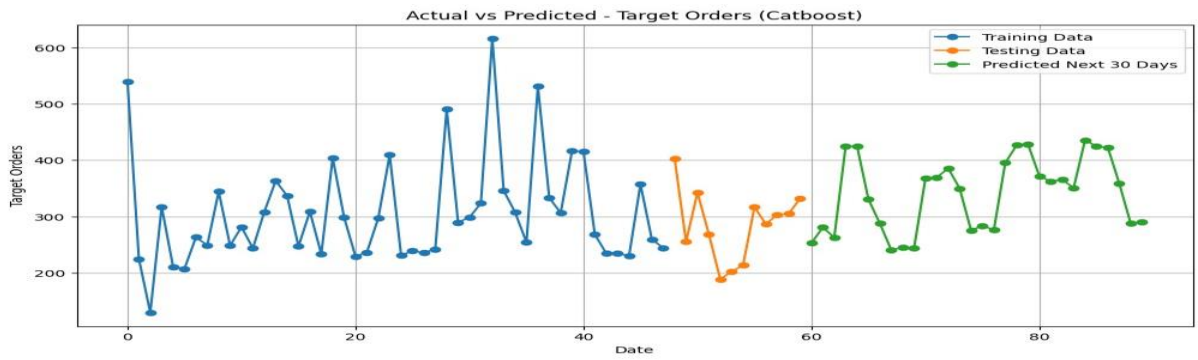


Fig. 11

6.2 Adaboost

Model Selection: The AdaBoostRegressor was chosen for forecasting based on its ability to handle complex relationships in data and its flexibility in working well with various base models. Our problem statement is related to forecasting so we will use AdaBoost regressor. It is an estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

Model Description: AdaBoostRegressor is an ensemble learning technique that combines multiple weak learners, often decision trees in the case of regression. Each weak learner is trained sequentially, with more emphasis on the misclassified instances from the previous learners. Initially, equal weights are assigned to all data points. A weak regression model, often a decision tree, is trained on the data, and its performance is evaluated. Weighted errors are calculated, considering the importance of each data point based on its prediction accuracy.

This adaptive process continues for several iterations, constructing an ensemble model that combines the predictions of individual models, with each model's contribution weighted by its accuracy. The final prediction results from the weighted sum of these individual model predictions.

Model Implementation: The dataset is partitioned into training and test sets, with 80% of the data designated for training and the remaining 20% for testing. The optimal hyperparameters for the model are determined using GridSearchCV. Following model training, predictions are generated and presented graphically. To evaluate model performance, Mean Squared Error (MSE) and R-squared (R2) values are computed. The time series data is stored in X, with the feature dynamically modified in each iteration as y. Model is tuned to forecast 30 days data for each feature of dataset. Then new datasets are created and predicted values are saved to a CSV file and visualized using suitable visualization tools.

Hyperparameter Tuning: For hyperparameters tuning in AdaBoostRegressor we have include the number of weak learners (n_estimators), the learning rate for each learner and max depth. The tuning process aims to find a balance between model complexity and avoiding overfitting.

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.1, 0.5, 1.0],
    'base_estimator__max_depth': [3, 4, 5]
}
```

Fig. 12

Conclusion: As it is time series forecasting and we are taking each feature at a time and forecasting so for each we have R-square value and Root Mean Square Error. Few features R-square value and Root Mean Square Error are shown below.

Feature Variable	R-square	RMSE
Urgent Order	0.32	16.81
Order type A	0.23	17.73
Order type C	-1.12	36.96
Banking Order (2)	0.66	20.35
Target (Total Order)	-1.58	96.86

Table: Few variables result.

Below figure shows the prediction line chart of actual data and forecasted data of Target (Total order).

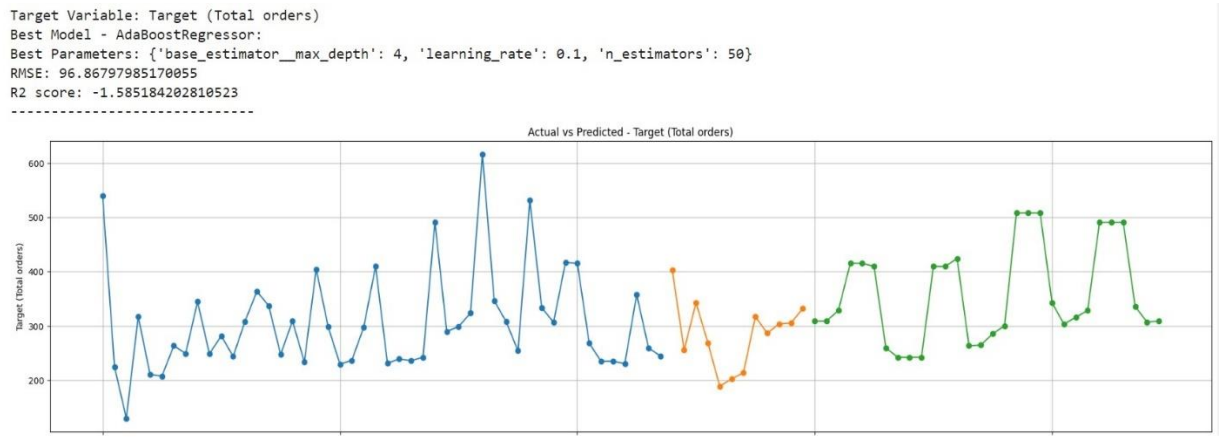


Fig. 13

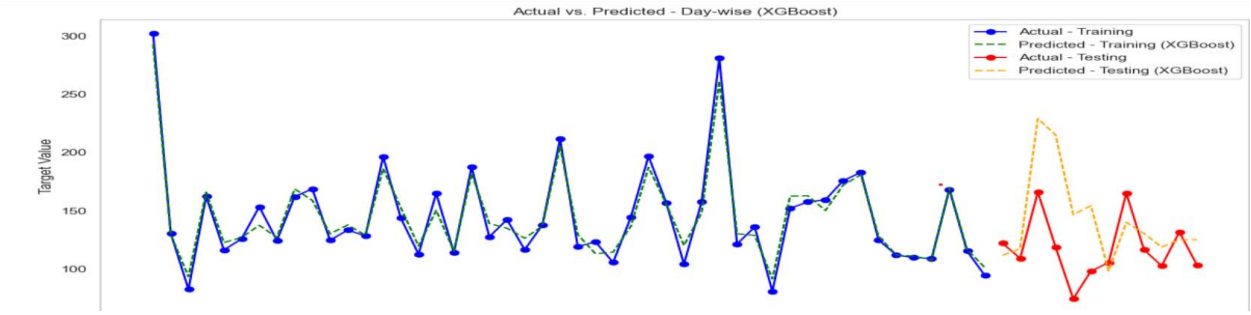
6.3. XG Boost:

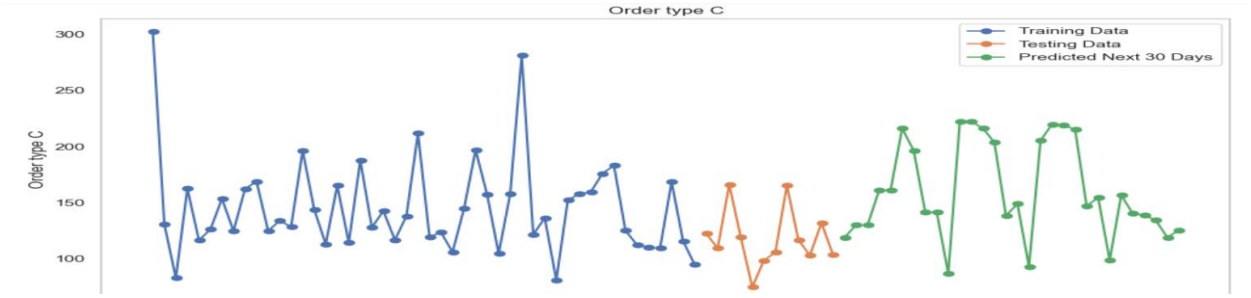
Model Selection:

Model Selection Criteria: XGBoost is a balance between complexity and interpretability while demonstrating superior performance in capturing complex patterns in time series data. Its ability to sequentially add weak learners, optimize a regularized objective function, and handle non-linear relationships makes it well-suited for time series forecasting tasks.

Model Description:

Architecture: XGBoost is an ensemble learning algorithm that combines the predictions of multiple decision trees through boosting.





Mathematical Foundations: It minimizes a regularized objective function, consisting of a loss term measuring prediction errors and a regularization term preventing overfitting. The algorithm adds decision trees sequentially to correct errors made by previous trees.

Underlying Assumptions: XGBoost assumes that individual decision trees (weak learners) are relatively simple models. The ensemble of these weak learners creates a robust and accurate model capable of capturing intricate patterns in time series data.

Hyperparameter Tuning: The hyperparameters tuned for the XGBoost model include:

- n_estimators: The number of boosting rounds or trees: [10, 20, 30, 40, 50, 100, 300]
- Learning_rate: The step size shrinkage used to prevent overfitting: [0.01, 0.05, 0.1]
- Max_depth: The maximum depth of each decision tree: [1, 2, 3, 5]
- Subsample: The fraction of samples used for building each tree: [0.2, 0.3, 0.4, 0.6, 0.8, 1.0]

The tuning process involves exploring different values for these hyperparameters to find the combination that minimizes the R^2 score. This aims to enhance the model's generalization capabilities and its ability to capture underlying patterns in the time series data.

Cross-Validation Strategy: The code utilizes a 5-fold cross-validation technique (cv=5 in GridSearchCV). For each target variable in Column_to_split, the XGBoost model is trained on four subsets of the time series data and validated on the remaining subset. This ensures a fair comparison of the model's performance across different time periods, considering the temporal structure of time series data.

6.4 Lasso Regression:

Model Selection Criteria: After analyzing the data, we came to know that data is continuous so we can use predictive model. A predictive model is a mathematical or computational representation that is trained on historical data to make predictions about future events or outcomes. The main purpose of using the predictive model is to train the data on past or historical data and then use them to predict the future data. Few reasons for selecting this model is mentioned below:

- Overfitting of data as less data is given.
- Also, there is data with nonlinear relationships.
- It is better than linear regression as it's ensemble learning approach so it works well with overfitting data, and we can also tune the hyperparameters here for better model results.

Model Description:

Objective function of Lasso regression consists of combination of loss function of the standard linear regression and a penalty term.

Objective function:

$$\sum_{i=1}^{n}(y_i - \beta_o 0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} | \beta_j |)$$

Where:

- y_i is the observed value for the i_{th} data point.
- β_o is the intercept term.
- β_j is the coefficient for the j_{th} independent variable x_{ij}.
- n is the number of data point
- p is the number of independent variables.
- λ is the regularization parameter that controls the strength of the penalty term.

The objective is to minimize the sum of squared differences between the predicted and actual values while also penalizing the absolute values of the coefficients.

The penalty term ($\lambda \sum_{j=1}^{p} | \beta_j |$)encourages sparsity in the model. It drives some coefficients to exactly zero effectively performing feature selection.

Mathematical Foundations:

It consists of L1 Regularization (i.e. penalty term) which is the sum of the absolute values of the coefficients.

Penalty term expression = $\lambda \sum_{j=1}^{p} | \beta_j |$

It also involves finding the intercept and slope value to optimize the objective function.

Hyperparameter Tuning:

For tuning, alpha (often denoted as λ, lambda) is a hyperparameter that controls the strength of the regularization applied to the model. Alpha is a regularization parameter that balances the trade-off between fitting the data well and keeping the model simple by discouraging overly complex models.

Model Implementation and Training:

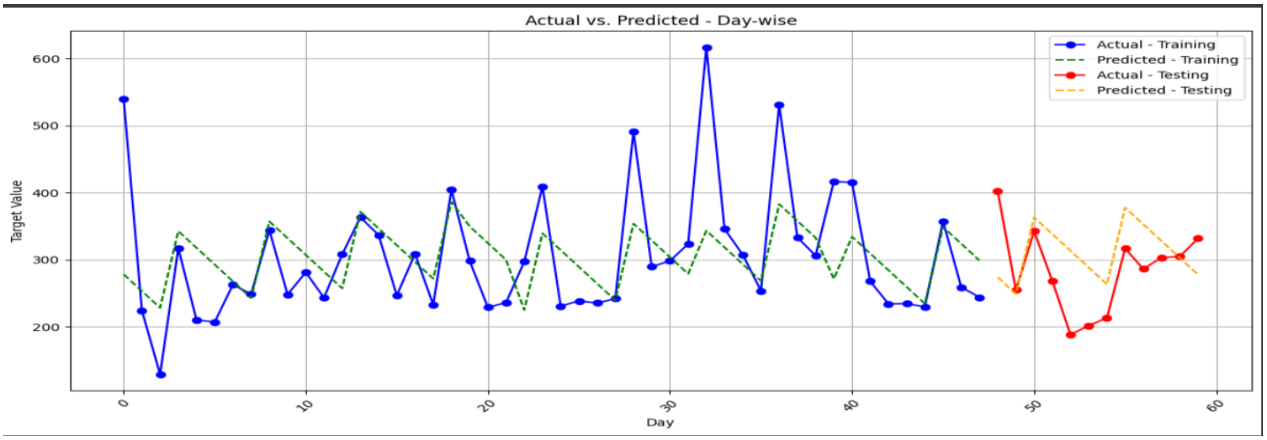
Model Implementation: For implementing a machine learning model, I follow these steps, data preparation, model selection, training, evaluation, and potentially hyperparameter tuning. For training the model, data was split into train and test data where train data was 80% of total data. In x_ test and x_ train week and days were taken while in y-test & y_train Columns different target features name was taken as forecasting different features are required.

Evaluation Metrics and Results:

R2 square (coefficient of determination) is measure of how well the independent variable(s) explain the variability in the dependent variable in a regression model. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It varies from 0 to 1, where 0 means model does not explain the variability and 1 indicate that it explains variability of the response data around its mean.

RMS Error (Root mean Square) is a measure of the average magnitude of the errors between predicted and observed values. It is calculated as the square root of the mean of the squared differences between predicted and observed values.

```
RMSE - Train: 81.05758645346033
R2 score - Train: 0.2600480490473105
RMSE - Test: 70.18774794731193
R2 score - Test: -0.35723054019603495
```

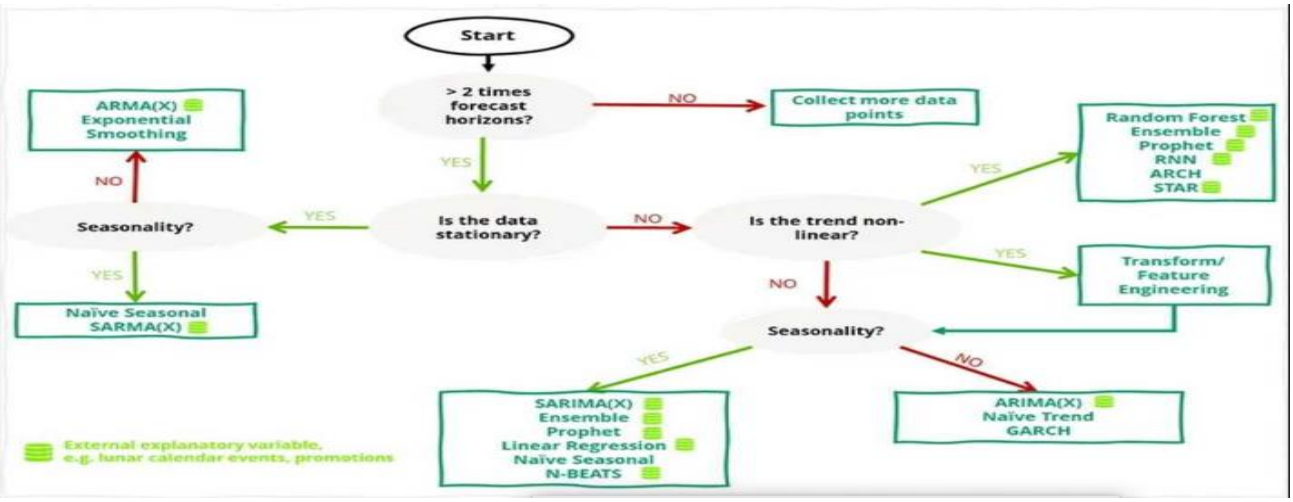


Summary: R2 score of all the value are negative which implies that lasso regression data is not good for prediction on given data set because we have taken only 3 features (Week_of_the_month, Day_of_the_week(Monday_to_Friday) & month) into account. Except for Banking Order (2) whose R2 score is close to 0.5. For RMSE, Fiscal Sector Orders have 100 while Orders from Traffic controller sector, Banking Orders (1), Banking Orders (2),Banking Orders (3) are having values above 1000. Others have values less than 100. RMSE values should be minimum.

Approach 2

Time Series Forecasting Models

1. ARIMA
2. SARIMAX
3. SARIMAX with Regressors
4. Prophet
5. Prophet with Regressors



6.5 Arima: -ARIMA stands for Autoregressive Integrated Moving Average.

Auto Regressive:

- These are time series models that predict values based on their past values.
- For example, if today is sunny, we might predict that tomorrow will also be sunny.

Integrated:

- ARIMA models work with **stationary data, where the mean and standard deviation remain relatively constant over time.**
- Suppose we have the stock price of Tesla, which is increasing every day. We can say that it is not stationary since its mean keeps increasing.
- To make such a series stationary, we can subtract the current value from the previous value. This transforms the series into a stationary one.

Moving Average:

- Moving averages help identify short-term trends in data by calculating the average of data within a specific timeframe.

ARIMA Model Usage:

Understanding ARIMA Parameters: The key points to understand here are **p**, **d**, and **q**. Let's dive into their meanings:

- **p**: Autoregressive Order. This is the number of past observations considered for making future predictions.
- **q**: Moving Average Order. It accounts for a specific number of previous residuals when making future predictions.
- **d**: Integration Order. It determines the number of differences needed to make the time series stationary *(T(q) — T(q-1))*.

Pros of ARIMA

- It can capture Short-Term trend (using Auto-regressive Order) as well as Long-Term trends (using Moving Average Order)
- ARIMA model does not make any assumption about data distribution, so it can work well with any distribution.
- ARIMA model has small number of parameters, so there it is easy to use as less time waste to find best turned parameters.

Cons of ARIMA

- ARIMA does not perform well with complex data pattern, it assumes that relationship between future values and past value is linear or near linear.
- ARIMA takes lot of time to train on huge dataset as compared to other time series models.
- ARIMA can be sensitive to outliers and extreme values, potentially leading to inaccurate forecasts.

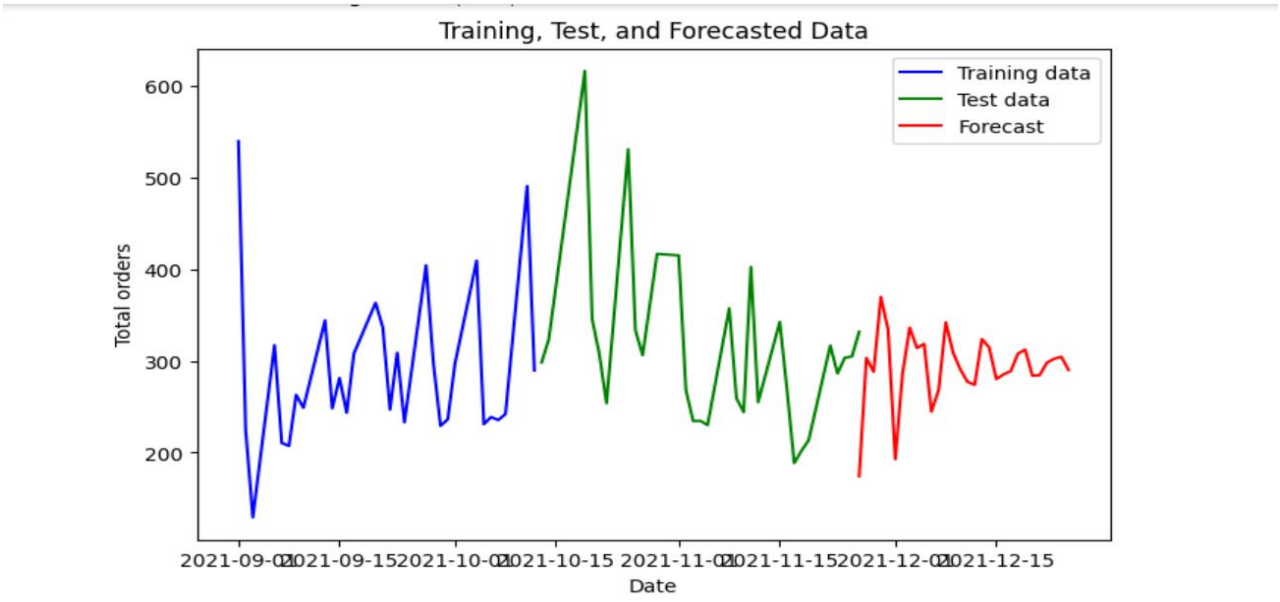
Code Representation: -

```
# Fitting ARIMA model
model = ARIMA(train['Target (Total orders)'], order=(5,1,0))
model_fit = model.fit()

# Forecasting for the next 30 days
forecast = model_fit.forecast(steps=30)
```

Metrics Result: -

```
R-squared (R2) Score: -0.2290
Root Mean Squared Error (RMSE): 101.2321
Mean Absolute Error (MAE): 70.1288
Mean Absolute Percentage Error (MAPE): 22.3970%
```

Plotting of Train, Test and Forecasted Data :-



6.6 Sarimax: - SARIMAX (Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors) is a generalization of the ARIMA model that **considers both seasonality and exogenous variables**. SARIMAX models are among the most widely used statistical models for forecasting, with excellent forecasting performance.

In the SARIMAX model notation, the parameters p, d, and q represent the autoregressive, differencing, and moving-average components, respectively. P, D, and Q denote the same components for the seasonal part of the model, with m representing the number of periods in each season.

- **p** is the order (number of time lags) of the autoregressive part of the model.
- **d is** the degree of differencing (the number of times the data have had past values subtracted).
- **q** is the order of the moving-average part of the model.
- **P** is the order (number of time lags) of the seasonal part of the model
- **D is** the degree of differencing (the number of times the data have had past values subtracted) of the seasonal part of the model.
- **Q** is the order of the moving-average of the seasonal part of the model.
- **m** refers to the number of periods in each season.

When two out of the three terms are zeros, the model may be referred to based on the non-zero parameter, dropping "AR", "I" or "MA" from the acronym describing the model**. For example, ARIMA (1,0,0) is AR(1) , ARIMA(0,1,0) is I(1) , and ARIMA(0,0,1) is MA(1).**

Libraries Imported: -

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

Data Preparation & Splitting: -

Train Percentage (train_percentage): Represents the proportion of the dataset allocated for training. Here, it's set at 80%.

Train Final Index (train_final_index): Calculates the index position for the end of the training data based on the specified percentage of the dataset.

Splitting the Data:

Train Data: Contains the initial portion (80%) of the dataset, from index 0 to the calculated train_final_index. This data is used for training the forecasting model.

Test Data: Consists of the remaining portion (20%) of the dataset, starting from the train_final_index to the end. This data remains unseen during training and is used to evaluate the model's performance.

```
demand_df=df
# Log scaling the 'Target (Total orders)' column
df_log = np.log(demand_df['Target (Total orders)'])

# Splitting data into train and test sets (80-20 split)
train_percentage = 80
train_final_index = round(len(df_log) * (train_percentage / 100))
train_data, test_data = df_log[0:train_final_index], df_log[train_final_index:]
```

SARIMAX Model Implementation: -

This below function builds and trains a Seasonal Autoregressive Integrated Moving Average (SARIMA) model for time series forecasting. Here's a concise breakdown:

SARIMA_model Function:

Inputs: Takes in the training and test datasets for time series forecasting.

Model Initialization: Initializes a SARIMAX model with specific parameters (order=(1, 1, 1), seasonal_order=(1, 1, 1, 5)).

Model Fitting: Fits the SARIMAX model to the training data.

Model Persistence: Saves the trained model using joblib to a file named "timeseries_sarima_model.pkl" for future use.

Prediction: Uses the trained model to make predictions for the test data (forecasting future values).

Back-Transformation: Reverts the predictions from the log-transformed scale to the original scale (exponential transformation) for better interpretability.

Output: Returns the trained model (model_fit) and a DataFrame (res) containing predicted values (Pred) and actual values (Act) for evaluation.

```
# SARIMA model with seasonal parameters
def SARIMA_model(train, test):
    model = SARIMAX(train, order=(1, 1, 1), seasonal_order=(1, 1, 1, 5))
    model_fit = model.fit()

    joblib.dump(model_fit, "timeseries_sarima_model.pkl")
    yhat = model_fit.predict(start=len(train), end=len(train) + len(test) - 1)
    res = pd.DataFrame({"Pred": yhat, "Act": np.exp(test)})  # Back-transforming predictions
    return model_fit, res
```

Evaluation of Models by using different Metrics: -

```
RMSE: 71.26669953648661
R2 score: -0.3992789497503215
Mean Absolute Error (MAE): 53.4735
Mean Absolute Percentage Error (MAPE): 18.7813%
```

Forecasting for the next 30 days: -

So, we are forecasting for the next 30 days using a SARIMA model (model_sarima) and prepares a DataFrame (forecast_sarima_df) to store these forecasted values.

Forecasting Dates:

Creates a list of dates for the next 30 days (forecast_dates) starting from the last date in the test data plus one day (test_data.index[-1] + d for d in range (1,31)).

Forecast DataFrame Preparation:

Initializes an empty DataFrame (forecast_sarima_df) with the forecast_dates as the index.

Forecasting Using SARIMA Model:

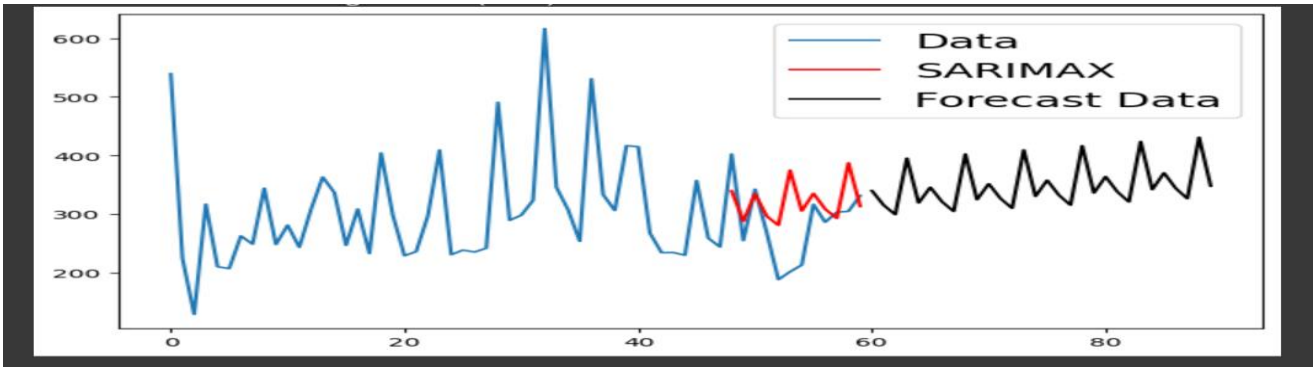
Utilizes the trained SARIMA model (model_sarima) to predict the total orders for the next 30 days.

Populates the 'Target (Total orders)' column in the forecast_sarima_df DataFrame with the predicted values.

Essentially, this code snippet generates a time series of forecasted total orders for the next 30 days using the SARIMA model and stores these predictions in a DataFrame for further analysis or visualization.

```
# Forecasting for the next 30 days
forecast_dates = [test_data.index[-1] + d for d in range (1,31)]
# forecast_dates = pd.date_range(start=test_data.index[-1] + pd.Timedelta(days=1), periods=7, freq='D')
forecast_sarima_df = pd.DataFrame(index=forecast_dates)
forecast_sarima_df['Target (Total orders)'] = model_sarima.predict(start=forecast_sarima_df.index[0], end=forecast_sarima_df.index[-1])
```

Plotting the Train, Test and Forecasted Data: -



SARIMAX With Exogenous Variables: -

This below code segment fits a SARIMAX model by considering both the endogenous variable ('Target (Total orders)') and multiple exogenous variables ('Urgent order', 'Order type B', 'Order type C', 'Orders from the traffic controller sector', 'Banking orders (2)') for better prediction accuracy.

Model Fitting (SARIMAX):

Utilizes the SARIMAX model to incorporate the target variable and additional exogenous variables for modeling the time series data.

The 'order' parameter specifies the autoregressive (AR), differencing (I), and moving average (MA) components.

The 'seasonal_order' parameter sets the seasonal ARIMA components.

Forecasting for Next 30 Days:

Uses the fitted SARIMAX model to forecast the next 30 days.

Includes exogenous variables from the test dataset ('Urgent order', 'Order type B', 'Order type C', 'Orders from the traffic controller sector', 'Banking orders (2)') to make these predictions.

Retrieves the predicted mean values for the forecasted period.

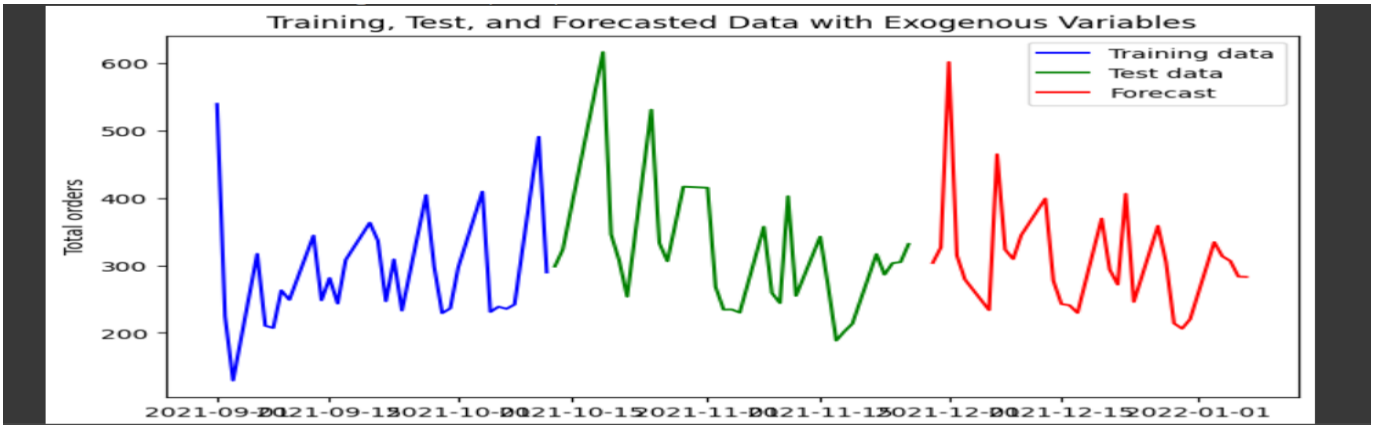
In summary, this code builds a SARIMAX model considering both endogenous and exogenous variables, and then generates a 30-day forecast based on this model, incorporating the provided exogenous variables for prediction.

```
# Fitting SARIMAX model
model = SARIMAX(endog=train[ 'Target (Total orders)'],
                exog=train[[ 'Urgent order', 'Order type B', 'Order type C',
       'Orders from the traffic controller sector',
    'Banking orders (2)']],
                order=(1, 1, 1),  # Example ARIMA order, can be tuned
                seasonal_order=(1, 1, 1, 5))  # Example seasonal order, can be tuned
model_fit = model.fit()

# Forecasting for the next 30 days
forecast = model_fit.get_forecast(steps=30, exog=test[[ 'Urgent order', 'Order type B', 'Order type C',
       'Orders from the traffic controller sector', 'Banking orders (2)']])
forecast_values = forecast.predicted_mean
print(forecast_values.values)
```

Model Evaluation Metrics & Model's Result: -

```
[303.71280434 326.61866898 601.52969965 314.82292334 279.2444246
 233.75445306 464.78915579 323.52243457 309.51939975 344.83073279
 399.22697711 277.22774409 242.80948284 240.97178973 229.62643942
 369.53603768 293.94912117 271.69751674 406.23159388 246.02150646
 358.68001829 305.17446688 214.19120056 206.30788436 220.24927712
 334.57121289 313.69342905 306.61497602 283.52660062 283.01414354]
R-squared (R2) Score: 0.9161
Root Mean Squared Error (RMSE): 26.4505
Mean Absolute Error (MAE): 19.6302
Mean Absolute Percentage Error (MAPE): 6.2693%
```

Plotting of Train, Test and Forecasted Data for next 30 days:-



6.7 Prophet Model Experimentation

Prophet Model Implementation: -

This below code initializes a forecasting model using the Prophet library, a tool designed for time series forecasting with added flexibility and ease of use.

Prophet Model Initialization:

Growth Parameter: Set to either 'linear' or 'logistic,' depending on the nature of the data's growth. 'linear' assumes linear growth, while 'logistic' accommodates non-linear growth patterns.

Seasonality Mode: Configured as 'multiplicative,' implying that seasonal effects will be proportionate to the trend. Other options might include 'additive,' assuming constant seasonal effects.

Seasonality Components: Specifies various seasonal components like daily and weekly patterns. For instance, 'daily_seasonality=True' and 'weekly_seasonality=True' will activate these patterns.

Yearly Seasonality: Currently disabled ('yearly_seasonality=False') since we have no yearly data

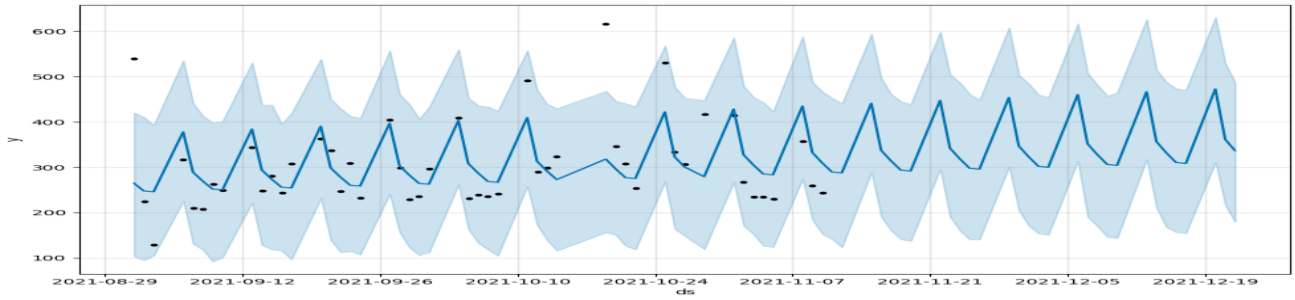
Changepoint Prior Scale: Controls the flexibility of trend changes. A lower value makes the model less sensitive to abrupt changes.

Interval Width: Sets the width of the uncertainty intervals for the forecast. Here, it's at a 95% confidence level.

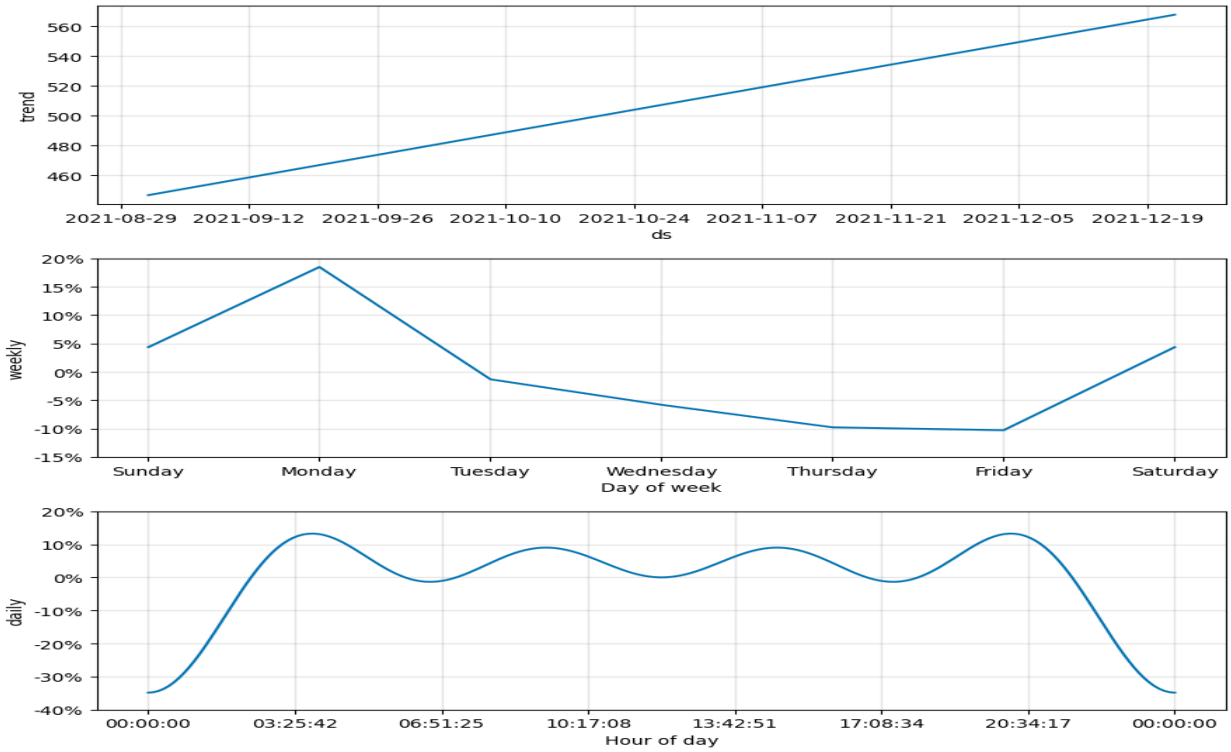
This initialization establishes a Prophet model with specified parameters, allowing for time series forecasting while accommodating various trends and seasonal patterns present in the data.

```
# Define and fit the model
model_prophet = Prophet(
    growth='linear',  # Choose 'linear' or 'logistic' based on your data's growth
    seasonality_mode='multiplicative',
    daily_seasonality=True,
    weekly_seasonality=True,
    yearly_seasonality=False,  # Adjust as needed
    changepoint_prior_scale=0.01,  # Adjusting the flexibility of trend changes
    interval_width=0.95  # Confidence interval
)

# Fit the model
model_prophet.fit(train_data_prophet)
```

Prophet Model Visualization of Data and Forecasted data for 30 days date wise with Total orders:-



Plotting the selected Components (Trend, Weekly & Daily): -



Prophet Model Implementation with Regressors and LASSO: -

Model Initialization and Configuration:

Prophet Model Setup:

Initializes a Prophet model (model) with specified configurations: Seasonality settings: Configures daily and weekly seasonality while disabling yearly seasonality. Adds a custom monthly seasonality.Includes regressors (exogenous variables) related to order types, urgent orders, sectors, etc.

Data Preparation and Training:

Data Splitting: Splits the data into features (X) and the target variable (y). Divides the data into training and test sets using train_test_split.

Model Training: Fits the Prophet model (model.fit) using the training data, concatenating features and the target variable.

Forecasting and Evaluation:

Model Prediction: Predicts on the test data (model.predict), generating forecasted values.

Model Visualization: Plots the forecasted values and components (trend, weekly) using plot and plot_components.

Model Evaluation: Calculates error metrics such as RMSE and R2 score (mean_squared_error, r2_score) to assess the model's performance.

Forecasting Future Data:

Future Data Preparation: Creates a future DataFrame (future) for the next 30 business days to forecast future values.

Individual Prophet Models: Trains separate Prophet models for each column except the timestamp ('ds').Predicts future values for each feature using these separate models and appends these predictions to the future Data Frame.

Combining Predictions and Error Calculation:

Forecast Combination: Uses the main Prophet model to generate forecasted values (model.predict) for future data.Utilizes a pre-trained Lasso regression model (lasso_model.predict) to make predictions on the same future data.

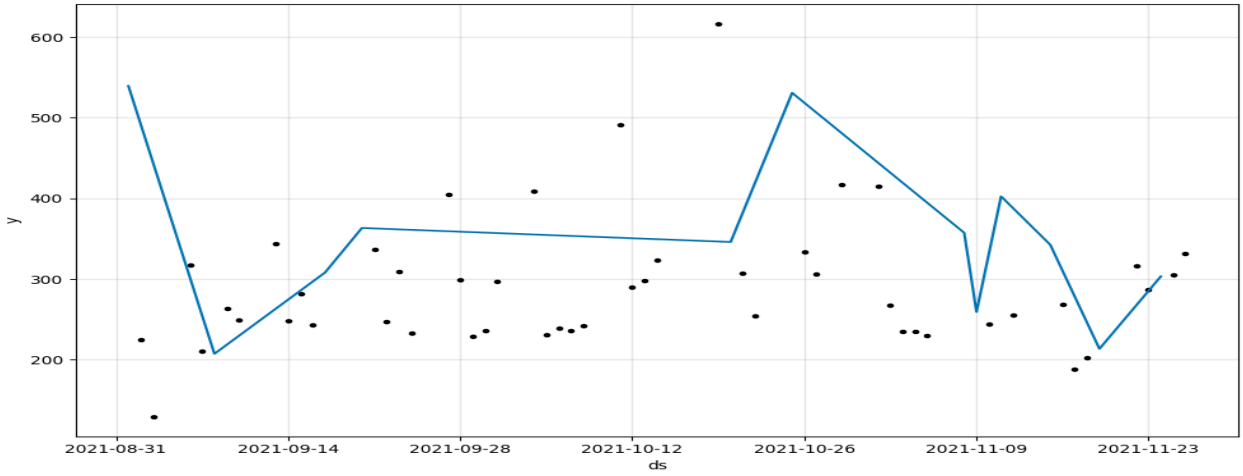
Error Assessment: Calculates RMSE and R2 score between the forecasted and predicted values to evaluate and compare their performance.

```
model = Prophet(growth='linear',
    seasonality_mode='multiplicative',
    daily_seasonality=False,
    weekly_seasonality=True,
    yearly_seasonality=False,
    changepoint_prior_scale=0.001,
    interval_width=0.95)


model.add_seasonality(
        name='weekly',
        period=30.5,
        fourier_order=5
    )

# # Adding regressors to the model
model.add_regressor('Non-urgent order')
model.add_regressor('Urgent order')
model.add_regressor('Order type A')
model.add_regressor('Order type B')
model.add_regressor('Order type C')
model.add_regressor('Fiscal sector orders')
model.add_regressor('Orders from the traffic controller sector')
model.add_regressor('Banking orders (1)')
```

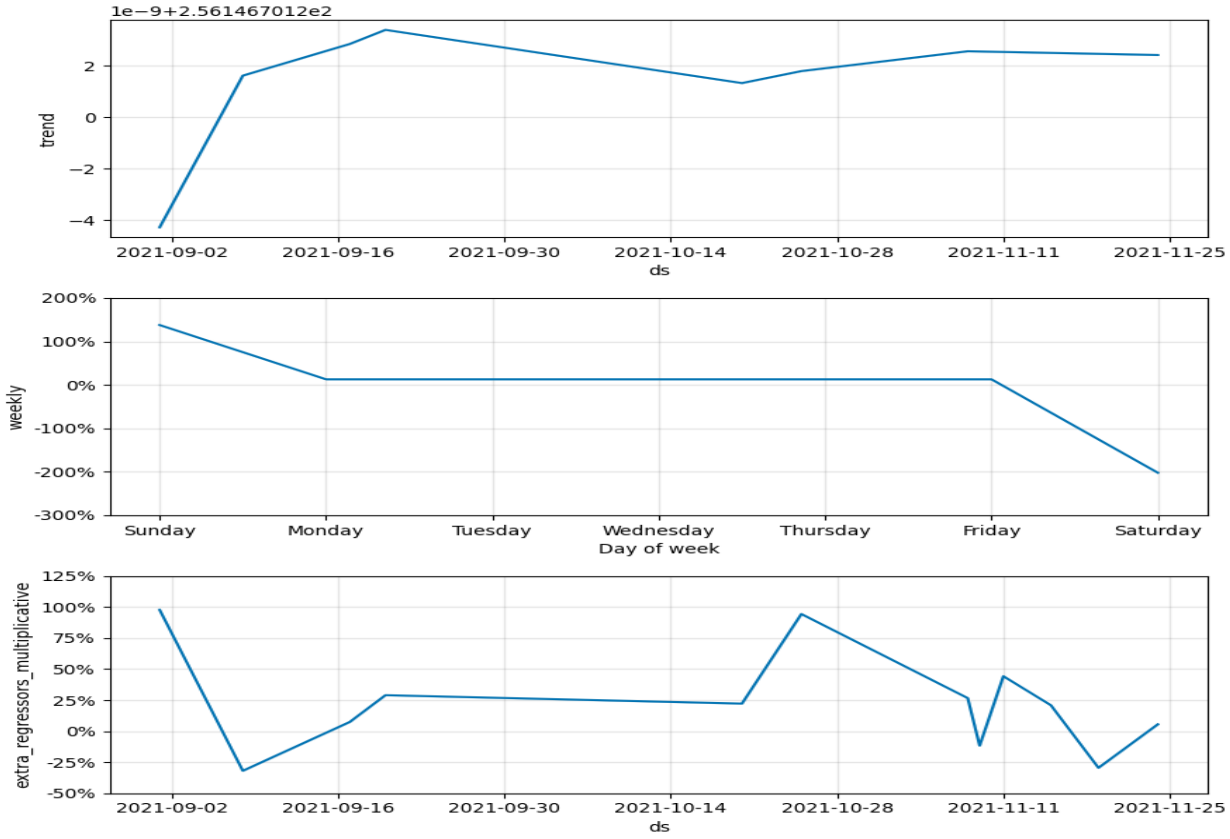
Splitting of Data: -

```
X = df_final.drop(columns=['y'])  # Features
y = df_final['y']  # Target variable
# Splitting data into train-test (adjust test_size and random_state as needed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

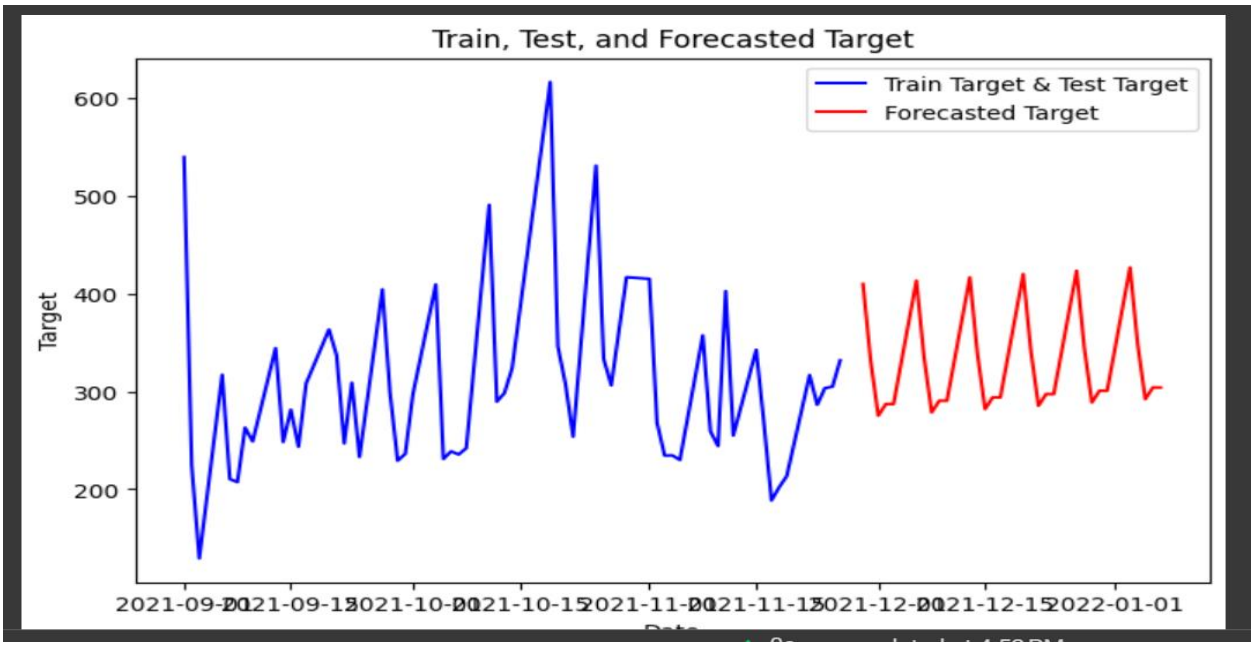
Prophet Model Visualization: -



Prophet Model Trend Analysis over the given data:-



Visualization of Forecasted data:-

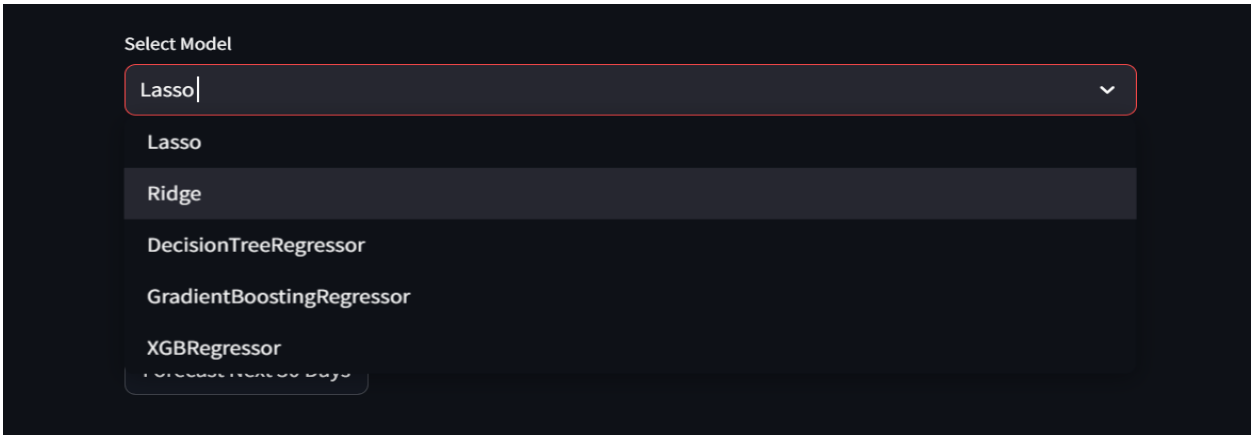


Visualization & Deployment:-

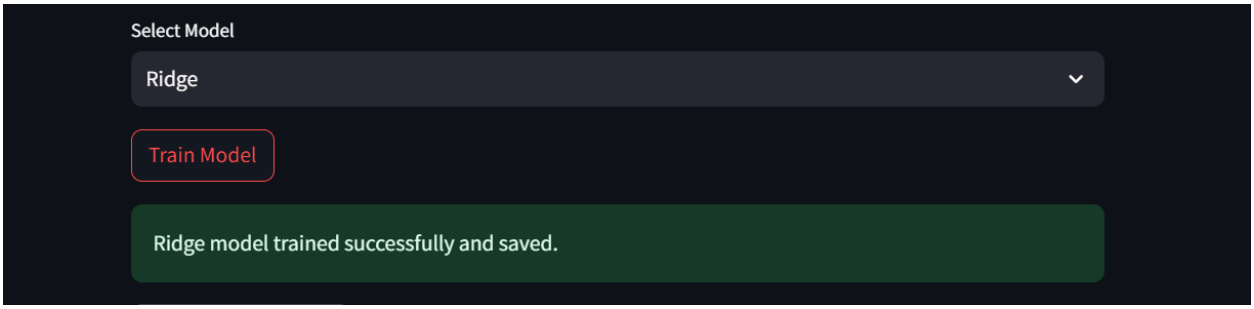
We have used the Streamlit framework to visually depict every stage of the Machine Learning Lifecycle, starting from Data loading to the final step of forecasting data.

Load Data Phase: -

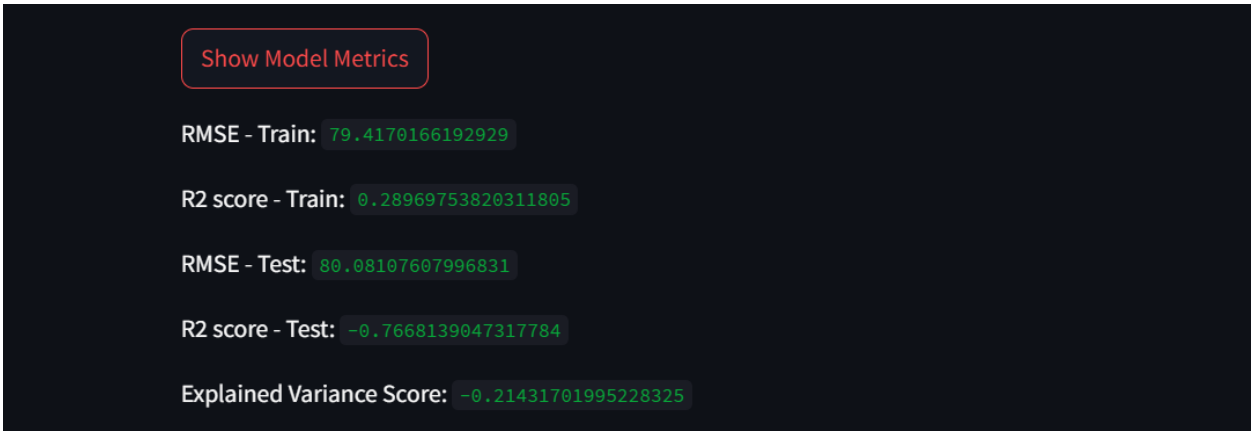
Dropdown Button for all regressor Models: -



Let's say we selected the Ridge Model here and when we will click the Button "Train Model" to train it will train the Model in the backend and will pop up the notification "Model Trained Successfully and saved": -



Then we are showing all the Metrics for the selected Model: -



We are plotting the graph of Actual & Predicted for the Train and the Test data: -
Forecasting plot for the next 30 days: -

Streamlit Link :- https://tidy-apples-reply.loca.lt

7.Evaluation Metrics: Metric used for evaluation of model is Root Mean Square Error (RMSE), R square score, variance score and MAPE (Mean Absolute Percentage error)

		RMSE	R2-Score	Variance Score	MAPE(Mean Absolute Percentage Error)
			Machine Learning Approach		
Models	**Linear regression**	9.568	1	1	2.751

Lasso Regression	4.748	0.993	0.994	24.924
Ridge Regression	0.302	0.999	0.999	0.039
Decision Tree Regressor	44.867	0.445	0.498	24.548
Gradient Boosting Regressor	42.293	0.507	0.507	22.711
XGBoost Regressor	43.956	0.467	0.499	22.675
Adaboost	96.8	-1.5	-	84.63
Catboost	88.97	-1.9	-	88.97

Deep Learning Approach

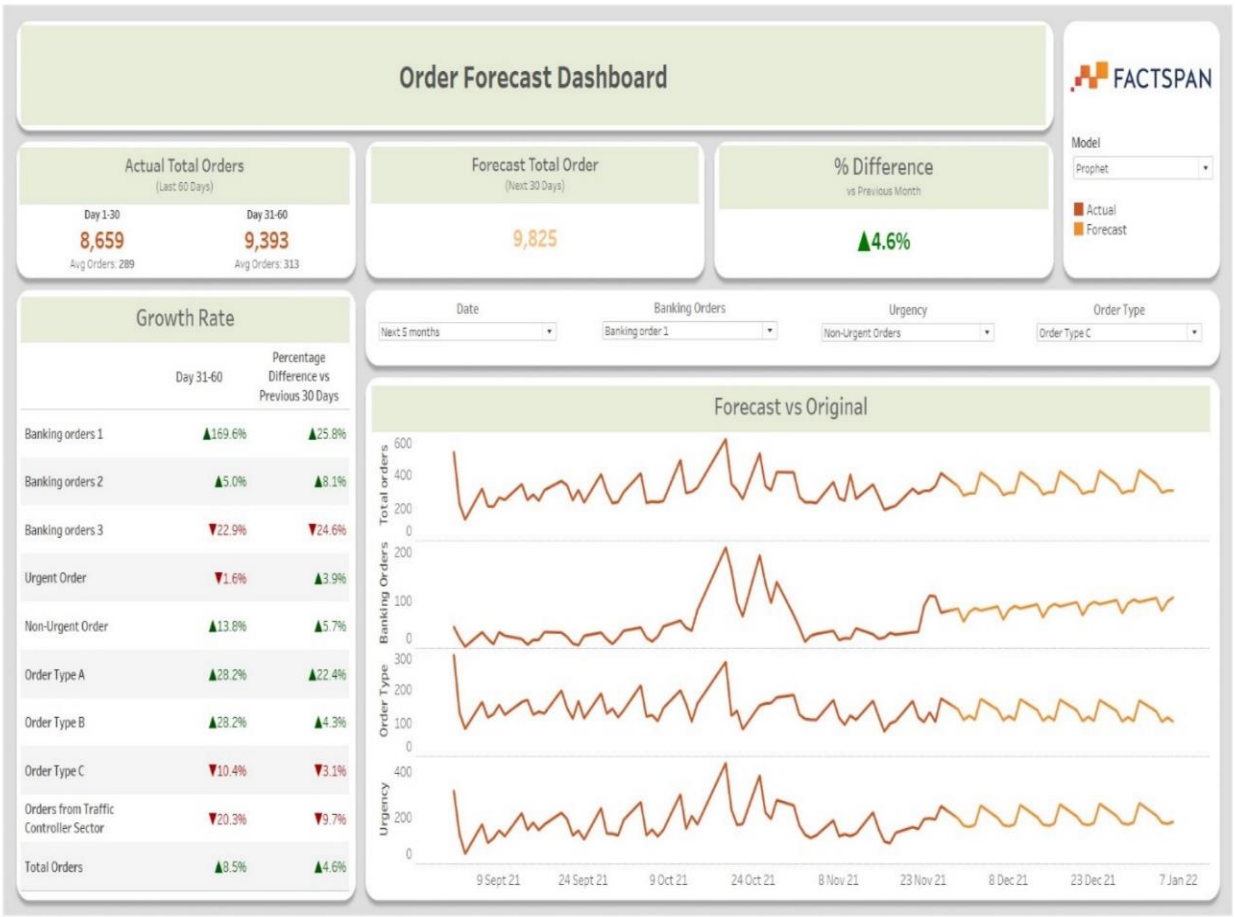
MLP(Multi Layer Perceptron)	2.322	0.998	0.998	25.187
MLP Regressor with 7 hidden layers, 25 neurons per layer, and 1000 iterations	11.89	0.961	0.963	24.305

Time Series Forecasting Models

	RMSE	**R2-Score**	**Variance Score**	**MAPE(Mean Absolute Percentage Error)**
ARIMA	-0.229	101.232	70.128	22.397
HWES	-0.03	61.433	30.452	12.569
SARIMAX	-0.39	71.266	53.475	18.781
SARIMAX with Exogenous variable	0.916	26.45	19.63	6.269
Prophet Model	-5.507	175.72	141.043	49.538

Prophet Model with Regressor	0.8755	21.25	18.465	8.564

8.Visualizations and comparative analysis/Result



th Rate | Forecast vs Original | Sheet 6 | Distribution of Banking Orders (... | Distribution of Order Type (Fore... | Distribution based on Urgency (... | Total Orders Distribution for 90 ... | Sheet 11 | Dashboard 2 | Dashboard 3 | Capstone Group 3

Tableau link :- [Capstone Group 3(1) | Tableau Public]

9.Conclusions

We tried lots of ways to predict demand for the Brazilian Logistic Company. Two methods stood out: SARIMAX with Regressors added and Prophet with Regressors. These methods were good at guessing demand accurately.

Why They're Good:

SARIMAX with Regressors and Prophet with Regressors did the best job because they could include more info, like outside factors that affect demand. This helped them make better predictions compared to other methods we tested.

Why They Matter for Logistics:

For the logistic company, these methods are super useful. They're great at handling the complicated nature of demand in logistics, where many things can influence how much stuff is needed.

Integration into GitHub for Version Control and Enhanced Project Management:

GitHub Link :- https://github.com/Hasnain-Factspan/Capstone_Project.git

Upon concluding our extensive analysis and selection of SARIMAX and Prophet models with regressors for demand forecasting in the logistics domain, we have taken a strategic step toward improving project management and ensuring systematic version control by integrating our work into GitHub.

Version Control and Collaboration:

- GitHub offers a robust version control system, allowing us to keep track of changes made throughout the project. This means we can easily revert to previous versions if needed and monitor modifications made by different team members.
- Collaboration is streamlined as GitHub provides a platform where multiple contributors can work on the project simultaneously, facilitating seamless teamwork and avoiding conflicts in code changes.

Enhanced Project Transparency and Documentation:

- By hosting our project on GitHub, we ensure transparency and accessibility. Anyone involved in the project can see the progress, including the analyses conducted, model evaluations, and the rationale behind selecting SARIMAX and Prophet.
- Detailed documentation, including README files, can be included to guide users through the project setup, data sources, model implementation, and interpretations, fostering clarity and understanding for all stakeholders.

Facilitating Project Management and Iterative Improvements:

- GitHub's issue tracking system enables the identification and assignment of tasks, bugs, or enhancements within the project. This promotes structured project management and prioritization of improvements or optimizations for SARIMAX and Prophet models.
- Through branches and pull requests, we can experiment with improvements or modifications to the models while ensuring a controlled and systematic approach. This iterative process allows us to refine the models over time based on new insights or requirements.

Ensuring Continual Model Maintenance and Performance Monitoring:

- Regular updates and refinements to the SARIMAX and Prophet models are vital for maintaining their accuracy and relevance. GitHub's platform allows us to schedule and track these updates, ensuring the models remain effective amidst changing market dynamics.

- Performance monitoring, achieved through continuous integration and testing, can be implemented to automatically check the models against new data, ensuring they consistently meet the expected standards.

In Summary:

The integration of our demand forecasting project, particularly the SARIMAX and Prophet models, into GitHub serves multiple purposes, including version control, enhanced collaboration, transparency, systematic project management, iterative model improvements, and continual model maintenance. This integration marks a significant step in ensuring the efficiency, accessibility, and ongoing enhancements of our predictive models for the logistics company.

This elaborative explanation emphasizes how GitHub integration contributes to various aspects of project management, collaboration, transparency, and ongoing model improvements, aligning with the chosen SARIMAX and Prophet models for demand forecasting in the logistics domain.