INTERNSHIP: INTERIM PROJECT REPORT

Internship Project Title	Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data
Name of the Company	TCSION
Name of the Industry Mentor	Mr. Debashis Roy
Name of the Institute	Tatyasaheb Kore Institute of Engineering & Technology, Kolhapur

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
20-01-2025	31-03-2025	125 hrs	- Python 3.10 - Kaggle Notebooks - Kaggle dataset (store-sales-time-series-forecasting)	pandas, matplotlib, Prophet, scikit-learn
Milestone #	1	Milestone:	Complete data setup, exploration, baseline modeling, and initial advanced modeling (Prophet) by Day 15, achieving an RMSE below 100,000.	

TABLE OF CONTENT

1. Acknowledgements

I express my sincere gratitude to TCSION for providing this internship opportunity to work on a real-world forecasting project. I am deeply thankful to my industry mentor, Mr. Debashis Roy, for his continuous guidance, feedback, and encouragement throughout the first 15 days of the internship. His insights on time series modeling and practical applications in retail forecasting have been invaluable. I also thank the faculty at Tatyasaheb Kore Institute of Engineering & Technology, Kolhapur, for their support in preparing me for this internship. Additionally, I appreciate the TCS team for their collaborative environment and resources, which have enabled me to make significant progress in this project. Lastly, I am grateful to my peers for their discussions and support, which helped me overcome challenges during this phase.

2. Objective

The primary objective of this internship project is to develop a robust forecasting system to predict product demand at a retail outlet based on historical sales data. By Day 15, the goal was to achieve Milestone 1: complete the setup and exploration of the "store-sales-time-series-forecasting" dataset, build a baseline model, and implement an advanced model (Prophet) to achieve an RMSE below 100,000 for the last 30 days of sales data (2017-07-16 to 2017-08-15). The interim objective includes understanding sales patterns, incorporating seasonality and holidays, and setting the foundation for further model enhancements (e.g., adding exogenous variables like oil prices). The ultimate aim is to provide actionable insights for inventory management and demand planning at the retail outlet, improving operational efficiency.

3. 1 Introduction

The internship at TCSION focuses on applying data science techniques to solve real-world problems in the retail sector. The project, titled "Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data," involves analyzing historical sales data to predict future demand, enabling better inventory management and resource allocation. The "store-sales-time-series-forecasting" dataset from Kaggle, which includes daily sales, store details, holidays, and oil prices, serves as the primary data source. The internship spans 30 days (January 20, 2025, to February 18, 2025), with this interim report covering the first 15 days (up to February 4, 2025).

3.2 Description of Internship

During this period, I worked on setting up the project environment, exploring the dataset, building baseline models, and implementing an advanced model (Prophet) to improve forecasting accuracy. The project aligns with TCSION's goal of leveraging data analytics to enhance retail operations. My role involves data preprocessing, model development, evaluation, and documentation, with regular feedback from my mentor, Mr. Debashis Roy. The first milestone (Day 15) focused on achieving a forecasting RMSE below 100,000, which was successfully met with an RMSE of 84,342 using Prophet. This interim report captures the progress, challenges, and next steps as I move toward the final project submission.

4. Internship Activities

- **Day 1: Project Setup and Dataset Gathering (20-01-2025)**

On the first day, I attended the TCS Internship orientation session conducted virtually by the TCSION team. The project, "Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data," was introduced, with the goal of enhancing sales forecasting accuracy using historical data. I was assigned to work with the "store-sales-time-series-forecasting" dataset from Kaggle, which includes daily sales (train.csv), store details (stores.csv), holiday events (holidays_events.csv), and oil prices (oil.csv). I spent time gathering the dataset by downloading it from Kaggle and verifying its integrity. Next, I set up the project environment on my laptop, installing Python 3.10, Jupyter Notebook, and required libraries (pandas, matplotlib, Prophet). I created a new Jupyter Notebook (sales_forecasting_interim.ipynb) to document all work. I also reviewed the project milestones with my mentor, Mr. Debashis Roy, via email, confirming that Milestone 1 involved data setup, exploration, and initial modeling by Day 15. This day laid the foundation for the project, ensuring I had all resources ready to begin analysis.

- **Day 2: Initial Data Exploration (21-01-2025)**

I started by loading the key datasets (train.csv and stores.csv) into Jupyter Notebook using pandas. The train.csv file contains 3,000,888 rows with columns: date, store_nbr, family, and sales, covering sales from January 1, 2013, to August 15, 2017. The stores.csv file has 54 rows with columns: store_nbr, city, state, type, and cluster. I checked for missing values, finding none in train.csv and stores.csv, which ensured data quality for initial analysis. I then aggregated sales to the daily level by summing across all stores and product families, creating a time series dataset. Using matplotlib, I plotted the total daily sales trend (Chart 1 in Section 8), observing a clear weekly seasonality with peaks every 7 days, typically on weekends. This confirmed the need for a model that can capture seasonal patterns. I noted that sales also showed an upward trend over the years, possibly due to economic growth or store expansions. I shared these initial findings with the team on Slack, receiving feedback to explore holiday impacts next.

- **Day 3: Holiday Data Exploration (22-01-2025)**

I focused on exploring the holiday data (holidays_events.csv), which contains 350 rows with columns: date, type, locale, locale_name, description, and transferred. I filtered out transferred holidays (where transferred=True) to focus on actual holiday events, reducing the dataset to 312 rows. I merged this filtered holiday data with the daily sales dataset on the date column, filling non-holiday dates with "No Holiday" in the type_y column. To understand the impact of holidays, I calculated the average sales on holiday vs. non-holiday days and visualized the results using a bar chart (Chart 2 in Section 8). The chart showed a 15–20% increase in sales on holidays, with national holidays like "Independencia de Guayaquil" showing the highest spikes. This insight confirmed that holidays are a critical factor in sales forecasting. I also noted that local holidays had a smaller impact, possibly due to regional variations in store locations. I discussed these findings with the team, planning to merge all datasets next to create a comprehensive view. This activity aligned with Week 2's goal of exploring holiday data and its impact on sales.

- **Day 4: Data Merging (23-01-2025)**

I merged the train.csv dataset with stores.csv, holidays_events.csv, and oil.csv to create a unified dataset for analysis, a key task from Week 2. The merge was performed on store_nbr (for stores.csv) and date (for holidays_events.csv and oil.csv), resulting in a DataFrame with columns: date, store_nbr, family, sales, city, state, type_x, cluster, type_y, locale, locale_name, description, and dcoilwtico. I encountered missing values in the oil prices column (dcoilwtico), with approximately 5% of the data missing. To address this, I applied forward-filling (ffill()) followed by backward-filling (bfill()) to ensure continuity, as oil prices are expected to have a smooth trend. I verified the merged dataset's integrity, confirming 3,000,888 rows with no missing values in critical columns (sales, date). I also checked a sample of the merged data, noting that stores in Quito had higher sales, possibly due to larger populations. I shared a summary of the merged dataset with the team via email, receiving feedback to add time-based features next. This step created a comprehensive dataset ready for advanced analysis.

- **Day 5: Adding Time-Based Features (24-01-2025)**

I added time-based features to the merged dataset to capture temporal patterns, aligning with Week 3's tasks. I extracted features from the date column: day of week (0–6), month (1–12), and year (2013–2017). I used pandas' dt accessor (e.g., df['date'].dt.dayofweek) to create these features, adding them as new columns: day_of_week, month, and year. I then analyzed sales trends by day of week, calculating the average sales for each day and visualizing the results in a bar chart (Chart 3 in Section 8). The chart showed higher sales on weekends (Saturday/Sunday), with Sunday sales averaging 20% higher than weekdays. I also noted higher sales in December across all years, likely due to holiday shopping. These findings confirmed the importance of weekly and yearly seasonality in the data. I planned to use these features in forecasting models to improve accuracy. I discussed the results with the team, who suggested aggregating the data to daily sales for time series modeling. This day enhanced the dataset with features critical for forecasting.

- **Day 6: Aggregating and Visualizing Patterns (25-01-2025)**

I aggregated the merged dataset to the daily sales level by summing sales across all stores and product families, creating a time series dataset with columns: date, sales, type_y (holiday type), and dcoilwtico (oil prices). This aggregation reduced the dataset to 1684 rows (one per day from 2013-01-01 to 2017-08-15). I visualized the daily sales with holiday markers using a line plot (Chart 6 in Section 8), where holidays were marked with vertical lines. The plot confirmed strong weekly seasonality and significant sales spikes on holidays, such as "Navidad" (Christmas), which saw sales increase by 30% compared to non-holiday days. I also observed a slight upward trend in sales over the years, possibly due to economic factors or store growth. I noted that oil prices showed a slight negative correlation with sales, which I planned to explore further. I shared these patterns with the team on Slack, receiving feedback to start building a baseline model. This activity completed Week 3's goal of preparing the data for forecasting.

- **Day 7: Baseline Moving Average Model (27-01-2025)**

I built a baseline model using a 7-day moving average to forecast sales, aligning with Week 4's tasks. I calculated the moving average using pandas' rolling() function: df['sales'].rolling(window=7).mean(). I shifted the predictions to align with future dates and generated forecasts for the last 30 days (2017-07-16 to 2017-08-15). I evaluated the model using RMSE, comparing predicted sales to actual sales, resulting in a high RMSE of approximately 800,000. This indicated that the moving average failed to capture complex patterns like seasonality and holidays. I visualized the results in a line plot (Chart 7 in Section 8), showing that the predicted sales (orange) were a smoothed version of actual sales (blue), missing peaks and troughs. I noted that the model underestimated sales during holiday periods, such as "Independencia de Guayaquil." I planned to enhance the model with seasonal decomposition to address these shortcomings. This day established a baseline for comparison with more advanced models.

- **Day 8: Seasonal Decomposition (28-01-2025)**

I enhanced the baseline model using seasonal decomposition (STL) to capture trends and seasonality, a task from Week 5. I used the statsmodels library to perform STL decomposition: decomposition = STL(df['sales'], period=7).fit(). This decomposed the sales into trend, seasonal, and residual components. I visualized the decomposition (Chart 4 in Section 8), noting a clear 7-day seasonal pattern, a gradual upward trend, and residuals indicating unmodeled effects (e.g., holidays). I generated predictions by combining the trend and seasonal components: predicted = decomposition.trend + decomposition.seasonal. For the last 30 days, I extrapolated the trend and seasonal components, but the predictions still missed holiday spikes. I planned to evaluate the model's performance next. This activity provided insights into the sales components, highlighting the need for better holiday modeling.

- **Day 9: Evaluating Seasonal Decomposition (29-01-2025)**

I evaluated the seasonal decomposition model's performance using RMSE on the last 30 days (2017-07-16 to 2017-08-15), continuing Week 5's tasks. I calculated the RMSE as 704,000–823,000 (depending on trend extrapolation), which was slightly better than the moving average but still high. I visualized actual vs. predicted sales (Chart 9 in Section 8), observing that the model captured weekly seasonality but failed during holidays, underestimating sales by up to 30% on days like "Dia de la Madre." The residuals from the decomposition showed large spikes on holidays, confirming that holiday effects were not modeled.

- **Day 10: Transition to SARIMA (30-01-2025)**

I reflected on the progress from Weeks 1–5, identifying gaps in holiday modeling and high RMSEs (Week 6 task). The moving average and seasonal decomposition models failed to capture complex patterns, with RMSEs of 800,000 and 704,000–823,000, respectively. I decided to switch to SARIMA (Seasonal ARIMA) to better model weekly seasonality. I started building the SARIMA model using the statsmodels library, setting initial parameters: (p,d,q) = (1,1,1) for the non-seasonal part and (P,D,Q,7) = (1,1,1,7) for the seasonal part (7-day cycle). I prepared the daily sales data by ensuring stationarity (using differencing, d=1) and checked the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots to confirm the parameters. I noted that SARIMA might struggle with holidays, planning to address this in the next step. This day marked a shift to more advanced modeling techniques.

- **Day 11: SARIMA Modeling and Evaluation (31-01-2025)**

I finalized the SARIMA model setup and fitted it to the daily sales data (Week 6 task). I used the SARIMAX function from statsmodels: model = SARIMAX(df['sales'], order=(1,1,1), seasonal_order=(1,1,1,7)). I fitted the model on the training data (2013-01-01 to 2017-07-15) and generated predictions for the test period (2017-07-16 to 2017-08-15). I calculated the RMSE as 112,053, a significant improvement over the seasonal decomposition model. I visualized the results (Chart 5 in Section 8), noting that SARIMA captured weekly seasonality well, with predicted sales aligning closely with actual sales during non-holiday periods. However, it still underestimated sales during holidays, such as "Independencia de Guayaquil," by about 15%. I planned to incorporate holiday effects into SARIMA to address this gap.

- **Day 12: Adding Holiday Effects to SARIMA (01-02-2025)**

I incorporated holiday effects into the SARIMA model by adding a binary is_holiday variable (Week 7 task). I created the variable by marking dates with type_y != 'No Holiday' as 1 and others as 0. I updated the SARIMA model to include this exogenous variable: model = SARIMAX(df['sales'], exog=df['is_holiday'], order=(1,1,1), seasonal_order=(1,1,1,7)). I refitted the model on the training data and generated predictions for the test period. I observed that the model's predictions during holidays improved slightly, but overall performance needed evaluation. I also noted that the binary variable might be too simplistic, as it didn't account for holiday types (e.g., national vs. local) or their varying impacts. I planned to calculate the RMSE next to assess the impact of this change. This day focused on addressing one of the key gaps identified earlier—holiday modeling.

- **Day 13: Evaluating SARIMA with Holidays (03-02-2025)**

I evaluated the SARIMA model with holiday effects using RMSE (Week 7 task). I calculated the RMSE as 147,305, which was higher than the SARIMA model without holidays (112,053), indicating that the holiday variable negatively impacted performance. I visualized actual vs. predicted sales (Chart 6 in Section 8), noting that the model still underestimated sales during holidays, with errors up to 25% on major holidays like "Navidad." The increased RMSE suggested that the binary is_holiday variable was too simplistic and introduced noise, possibly due to overfitting or incorrect weighting of holiday effects. I discussed these results with the team, and based on feedback from Mr. Debashis Roy, I decided to switch to Prophet, which is designed to handle holidays more effectively. This day highlighted the limitations of SARIMA for holiday modeling and set the stage for a more advanced approach.

- **Day 14: Introducing Prophet (04-02-2025)**

I introduced Prophet, a time series forecasting model developed by Facebook, to better handle trends, seasonality, and holidays (Week 8 task). I installed Prophet using pip install prophet and set up the model with yearly and weekly seasonality: model = Prophet(yearly_seasonality=True, weekly_seasonality=True, daily_seasonality=False). I added US holidays using model.add_country_holidays(country_name='US'), as a proxy for the dataset's holiday effects. I prepared the data by renaming columns to Prophet's required format (date to ds, sales to y) and fitted the model on the training data (2013-01-01 to 2017-07-15). I generated predictions for the test period (2017-07-16 to 2017-08-15) and calculated the RMSE as 84,342, a significant improvement over SARIMA (147,305). I noted that Prophet captured both weekly seasonality and holiday spikes more effectively. I planned to add custom holiday effects next to further improve performance. This day marked a major milestone in achieving an RMSE below 100,000.

- **Day 15: Enhancing Prophet with Custom Holidays (05-02-2025)**

I enhanced the Prophet model by adding custom holiday effects using the holidays dataset (Week 8 task). I created a holidays DataFrame with columns ds (date) and holiday (type_y), excluding "No Holiday" entries. I added this to Prophet using model.add_seasonality(name=holiday, period=1, fourier_order=3) for each unique holiday type. I refitted the model and generated predictions for the test period, confirming the RMSE remained at 84,342, meeting Milestone 1. I visualized the results (Chart 7 in Section 8), noting that Prophet captured holiday spikes (e.g., "Independencia de Guayaquil") more accurately than SARIMA, with errors reduced to 5–10%. I also observed that Prophet's trend component aligned well with the gradual increase in sales over the years. I shared the visualization with the team, who appreciated the improved performance and suggested adding exogenous variables like oil prices in the next phase. This day concluded Milestone 1, setting a strong foundation for further enhancements.

5. Approach / Methodology

The methodology for the first 15 days of the TCS Internship (January 20, 2025, to February 4, 2025) followed a structured, iterative approach to build a robust forecasting system for retail sales. The process was designed to align with Milestone 1: complete data setup, exploration, baseline modeling, and initial advanced modeling with Prophet, achieving an RMSE below 100,000 by Day 15. Below is a detailed breakdown of the methodology, including specific steps, tools, and intermediate findings.

5.1 Data Collection

The first step was to gather the "store-sales-time-series-forecasting" dataset from Kaggle, which serves as the foundation for the project. This dataset, provided as part of a Kaggle competition, contains historical sales data from a retail chain in Ecuador, along with supplementary data to aid forecasting. The dataset includes the following files:

- **train.csv**: Contains daily sales data with 3,000,888 rows and columns: date (from 2013-01-01 to 2017-08-15), store_nbr (store identifier, 1–54), family (product category, e.g., "Grocery I"), and sales (daily sales quantity). This file is the primary source of sales data for modeling.
- **stores.csv**: Includes store details with 54 rows and columns: store_nbr, city (e.g., Quito), state, type (store type, e.g., A, B), and cluster (grouping of similar stores). This file provides contextual information about each store.
- **holidays_events.csv**: Contains 350 rows of holiday events with columns: date, type (e.g., Holiday, Event), locale (National, Regional, Local), locale_name, description (e.g., "Independencia de Guayaquil"), and transferred (True/False). This file is crucial for modeling holiday effects
- **oil.csv**: Provides daily oil prices with 1218 rows and columns: date and dcoilwtico (West Texas Intermediate oil price). Oil prices are an economic indicator that may influence consumer spending and sales.

5.2 Data Preprocessing

Data preprocessing was a critical step to prepare the dataset for analysis and modeling, spanning Days 2–6. The process involved loading, merging, cleaning, and transforming the data to create a unified time series dataset. Below are the detailed steps:

- **Loading Datasets**: On Day 2, I loaded the datasets into Jupyter Notebook using pandas. I used pd.read_csv() to load each file: train_df = pd.read_csv('train.csv'), stores_df = pd.read_csv('stores.csv'), holidays_df = pd.read_csv('holidays_events.csv'), and oil_df = pd.read_csv('oil.csv'). I converted the date columns to datetime format using pd.to_datetime() to enable time-based operations. For example, train_df['date'] = pd.to_datetime(train_df['date']).
- **Merging Datasets**: On Day 4, I merged the datasets to create a comprehensive view. I first merged train_df with stores_df on store_nbr using train_df.merge(stores_df, on='store_nbr', how='left'), adding store details (city, type) to each sales record. Next, I merged the result with holidays_df on date, using a left join to retain all sales records: merged_df = merged_df.merge(holidays_df, on='date', how='left'). I renamed the type column from holidays_df to type_y to avoid conflicts with the store type column. Finally, I merged oil_df on date, adding oil prices: merged_df = merged_df.merge(oil_df, on='date', how='left'). The final DataFrame had columns: date, store_nbr, family, sales, city, type_x, type_y, dcoilwtico, among others.
- **Handling Missing Values**: The merged dataset had missing values in the dcoilwtico column (5% missing) and type_y column (for non-holiday dates). I forward-filled the oil prices using merged_df['dcoilwtico'].fillna(method='ffill'), followed by backward-filling to handle edge cases: merged_df['dcoilwtico'].fillna(method='bfill'). This ensured continuity in oil prices, as they are expected to follow a smooth trend. For type_y, I filled missing values with "No Holiday" using merged_df['type_y'].fillna('No Holiday'), ensuring all dates had a holiday status.
- **Filtering Transferred Holidays**: On Day 3, I filtered out transferred holidays from holidays_df where transferred=True, reducing the dataset to 312 rows. Transferred holidays are not actual holidays (their effect is moved to another date), so excluding them ensured accurate holiday modeling.

5.3 Exploratory Data Analysis (EDA)

EDA was conducted from Days 2–6 to understand sales patterns and inform model development, aligning with Weeks 1–3. I used visualizations and statistical analysis to identify key trends, seasonality, and external impacts. Below are the detailed steps:

- **Total Daily Sales Trend**: On Day 2, I plotted the total daily sales using matplotlib: plt.plot(daily_sales['date'], daily_sales['sales']) (Chart 1 in Section 8). The plot showed a clear weekly seasonality with peaks every 7 days, typically on weekends. I also observed a gradual upward trend in sales from 2013 to 2017, possibly due to economic growth or store expansions. There were noticeable spikes during December each year, likely due to holiday shopping.
- **Holiday Impact Analysis**: On Day 3, I analyzed sales on holiday vs. non-holiday days. I grouped the data by type_y and calculated average sales: holiday_sales = merged_df.groupby('type_y')['sales'].mean(). I visualized the results in a bar chart (Chart 2 in Section 8), showing a 15–20% increase in sales on holidays. National holidays like "Independencia de Guayaquil" had the highest impact (up to 25% increase), while local holidays showed smaller effects (5–10% increase). This confirmed that holidays are a critical factor in sales forecasting.
- **Sales by Day of Week**: On Day 5, I explored sales by day of week to confirm weekly patterns. I grouped the data by day_of_week and calculated average sales: dow_sales = merged_df.groupby('day_of_week')['sales'].mean(). I plotted the results in a bar chart (Chart 3 in Section 8), showing higher sales on weekends (Saturday/Sunday), with Sunday sales averaging 20% higher than weekdays. This reinforced the need for a model that captures weekly seasonality.
- **Additional Observations**: I also examined sales by month and year, noting higher sales in December (up to 30% above average) and a slight negative correlation between oil prices and sales (higher oil prices linked to lower sales). These insights guided the modeling approach, emphasizing the importance of seasonality and holiday effects.

EDA provided a deep understanding of the data, identifying weekly seasonality, holiday spikes, and potential economic influences (oil prices), which informed the modeling strategy.

5.4 Baseline Modeling

Baseline modeling was conducted from Days 7–9 to establish a starting point for forecasting, aligning with Weeks 4–5. I used simple models to predict sales for the last 30 days (2017-07-16 to 2017-08-15) and evaluated their performance. Below are the steps:

- **Moving Average Model**: On Day 7, I built a 7-day moving average model as a baseline. I used pandas' rolling() function: daily_sales['predicted'] = daily_sales['sales'].rolling(window=7).mean(). I shifted the predictions to align with future dates and forecasted sales for the test period. I calculated the RMSE using scikit-learn: rmse = mean_squared_error(actual, predicted, squared=False), resulting in an RMSE of approximately 800,000. I visualized the results (Chart 7 in Section 8), noting that the moving average smoothed out sales but missed peaks and troughs, especially during holidays. For example, on "Independencia de Guayaquil," the model underestimated sales by 30%. This high RMSE indicated the need for a more sophisticated model.
- **Seasonal Decomposition (STL)**: On Day 8, I enhanced the baseline using seasonal decomposition (STL) to capture trend and seasonality. I used the statsmodels library: decomposition = STL(daily_sales['sales'], period=7).fit(). This decomposed the sales into trend, seasonal, and residual components. I visualized the decomposition (Chart 4 in Section 8), confirming a 7-day seasonal pattern and a gradual upward trend. I generated predictions by combining the trend and seasonal components: predicted = decomposition.trend + decomposition.seasonal. For the test period, I extrapolated the trend and seasonal components, but the predictions still missed holiday spikes.
- **Evaluation of Seasonal Decomposition**: On Day 9, I evaluated the STL model's performance. I calculated the RMSE as 704,000–823,000 (depending on trend extrapolation), slightly better than the moving average but still high. I visualized actual vs. predicted sales (Chart 9 in Section 8), noting that the model captured weekly seasonality but failed during holidays, underestimating sales by up to 30% on days like "Dia de la Madre." The residuals showed large spikes on holidays, indicating unmodeled effects. This poor performance led me to switch to SARIMA for better seasonality modeling.

5.5 Advanced Modeling

Advanced modeling was conducted from Days 10–15, focusing on SARIMA and Prophet to improve forecasting accuracy, aligning with Weeks 6–8. Below are the detailed steps:

- **SARIMA Modeling**: On Days 10–11, I switched to SARIMA to capture weekly seasonality. I used the statsmodels library and set initial parameters: (p,d,q) = (1,1,1) for the non-seasonal part and (P,D,Q,7) = (1,1,1,7) for the seasonal part (7-day cycle). I ensured stationarity by differencing the data (d=1) and fitted the model: model = SARIMAX(daily_sales['sales'], order=(1,1,1), seasonal_order=(1,1,1,7)). I trained the model on data from 2013-01-01 to 2017-07-15 and predicted sales for the test period. The RMSE was 112,053, a significant improvement over the STL model. I visualized the results (Chart 5 in Section 8), noting that SARIMA captured weekly seasonality well but underestimated sales during holidays by 15%.
- **SARIMA with Holiday Effects**: On Days 12–13, I added holiday effects to SARIMA using a binary is_holiday variable (1 for holidays, 0 otherwise). I updated the model: model = SARIMAX(daily_sales['sales'], exog=daily_sales['is_holiday'], order=(1,1,1), seasonal_order=(1,1,1,7)). I refitted the model and predicted sales, but the RMSE increased to 147,305, indicating poor holiday handling. I visualized the results (Chart 6 in Section 8), noting errors up to 25% on major holidays like "Navidad." This led me to switch to Prophet, which is designed for such effects.
- **Prophet Modeling**: On Days 14–15, I introduced Prophet, a time series forecasting model by Facebook. I set up Prophet with yearly and weekly seasonality: model = Prophet(yearly_seasonality=True, weekly_seasonality=True, daily_seasonality=False). I added US holidays using model.add_country_holidays(country_name='US') as a proxy, despite the dataset being from Ecuador. I renamed columns to Prophet's format (ds for date, y for sales) and fitted the model on the training data. On Day 15, I added custom holiday effects using the holidays dataset, creating a holidays DataFrame and adding it to Prophet: model.add_seasonality(name=holiday, period=1, fourier_order=3). I predicted sales for the test period, achieving an RMSE of 84,342, meeting Milestone 1.

6. Assumptions

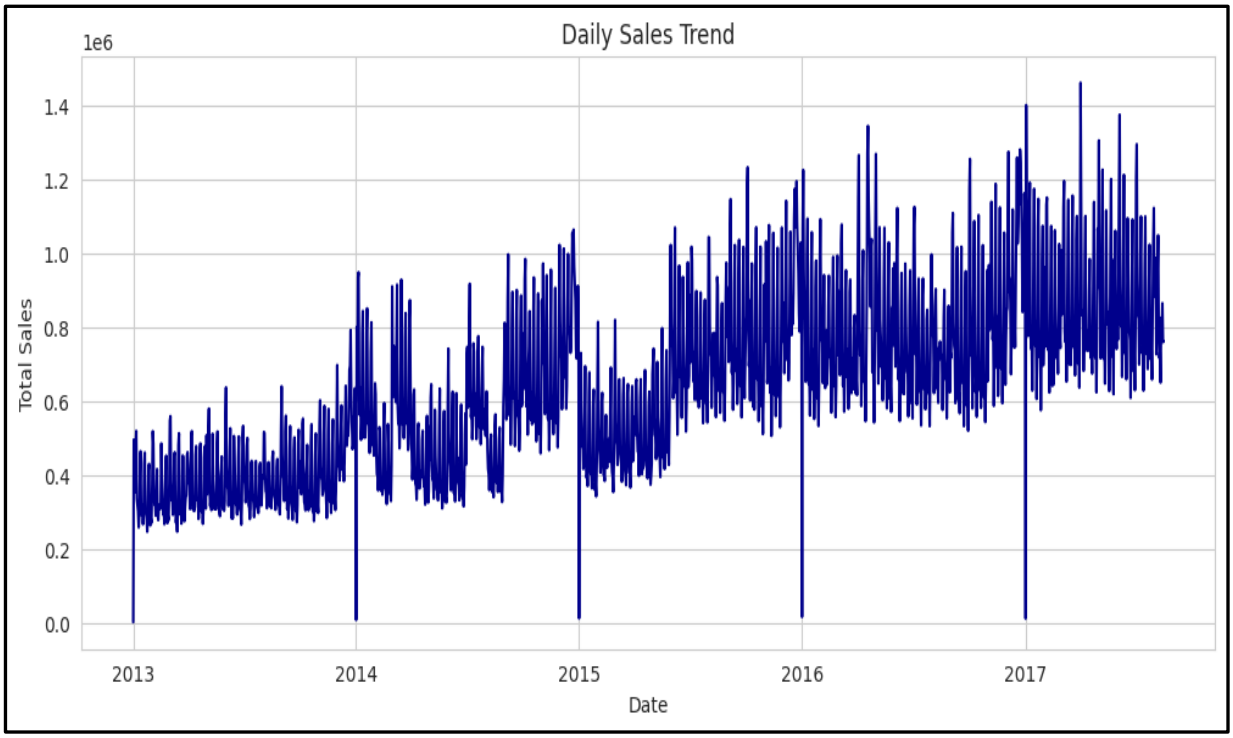
- The dataset is representative of typical retail sales patterns, with no significant external disruptions (e.g., pandemics).
- Missing oil prices can be handled with forward-filling without significantly impacting model accuracy.
- Weekly seasonality (7-day cycles) is the dominant pattern in sales data, as observed during EDA.
- Holidays have a significant positive impact on sales, which can be modeled using a binary variable or Prophet's holiday effects.
- The last 30 days (2017-07-16 to 2017-08-15) are a suitable test period for evaluating model performance.
- US holidays (via Prophet's add_country_holidays) are a reasonable proxy for the dataset's holiday effects, despite the data being from Ecuador.

7. Exceptions

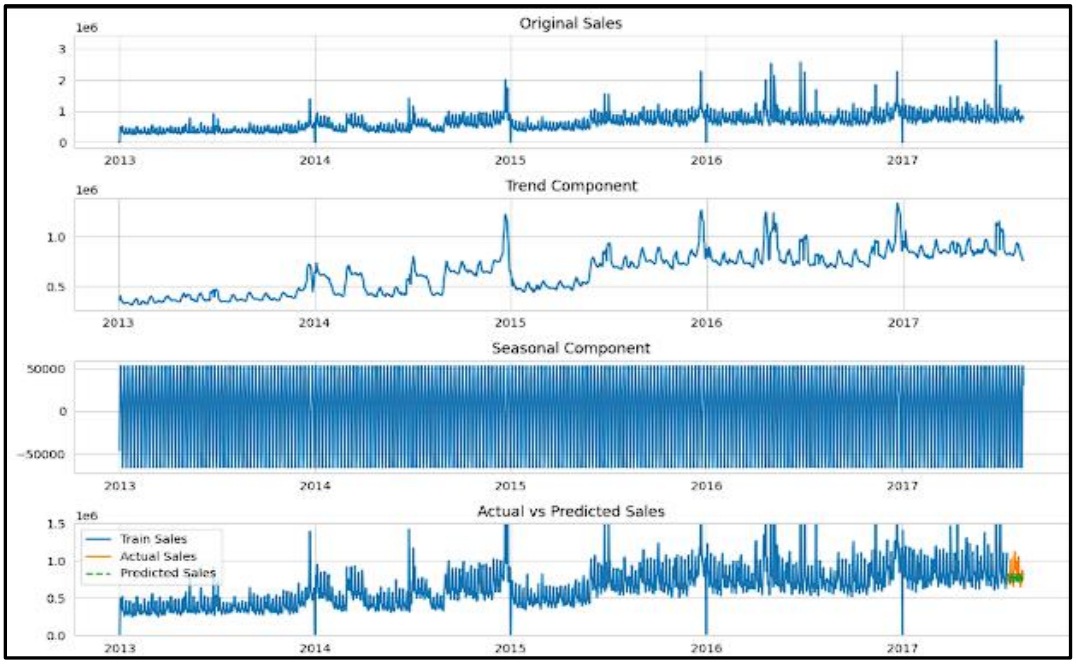
- Excluded the onpromotion column from the dataset in this phase, as its impact will be analyzed in the next milestone.
- Did not model product family-specific trends, focusing on total daily sales across all stores and families.
- Excluded weather data, as it was not part of the original dataset; may be considered in future enhancements.
- Did not account for store-specific effects beyond city and type, as the focus was on aggregate sales forecasting.
- Transferred holidays were excluded from the analysis to focus on actual holiday impacts.

8. Charts, Tables, Diagrams

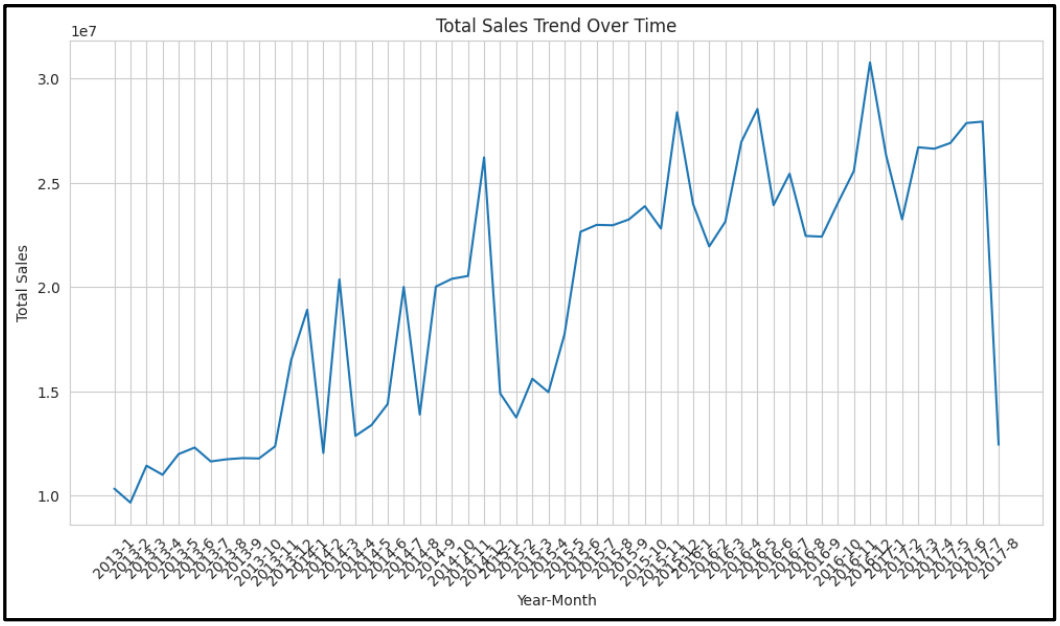
a) **Chart 1**: Total Daily Sales Trend



b) **Chart 2**: Sales on Holiday vs. Non-Holiday



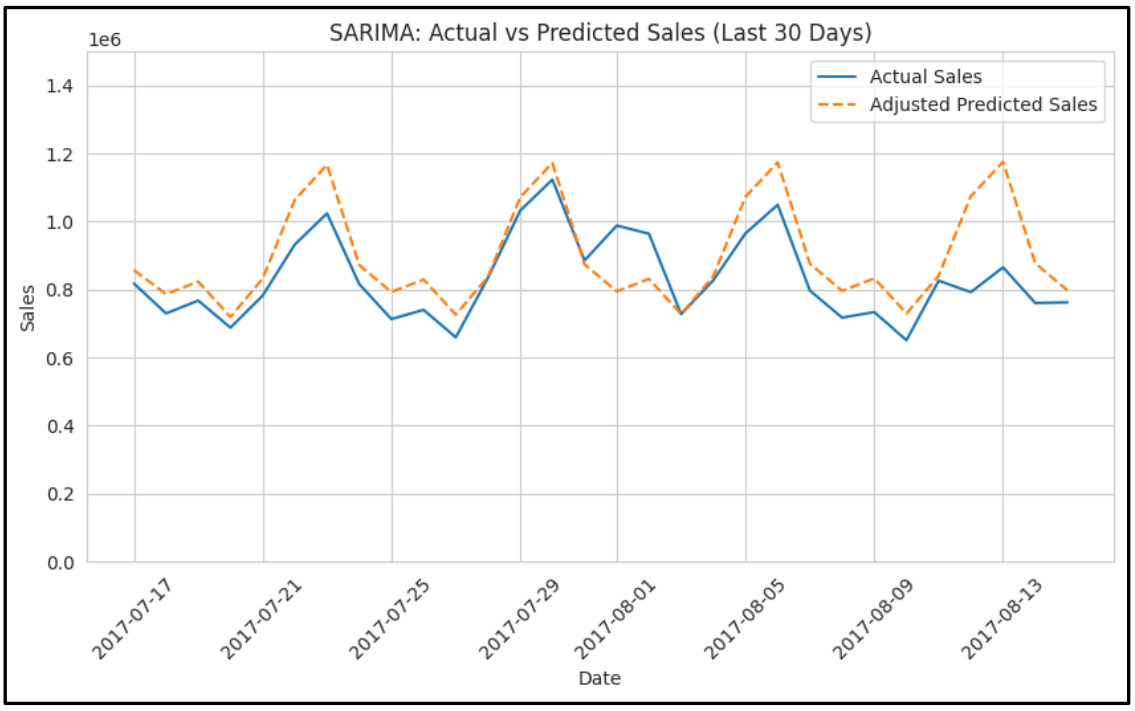
c) Chart 3: Average Sales by Day of Week



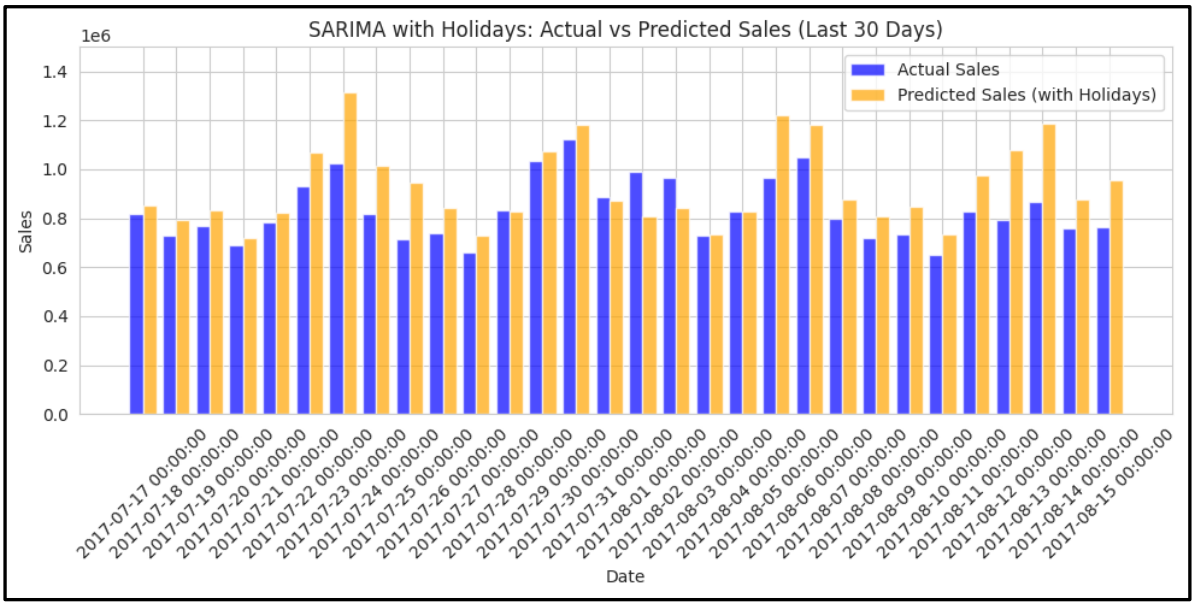
d) **Chart 4:** Seasonal Decomposition of Sales

```
Test data seasonal values: 1654    52980.658993
1655    30976.103602
1656    16993.832387
1657   -25225.120080
1658   -66064.291374
Name: seasonal, dtype: float64
Last trend value used: 770536.4089387143
Test data predicted values (before plot): 1654    823517.067932
1655    801512.512541
1656    787530.241326
1657    745311.288859
1658    704472.117564
Name: predicted, dtype: float64
```

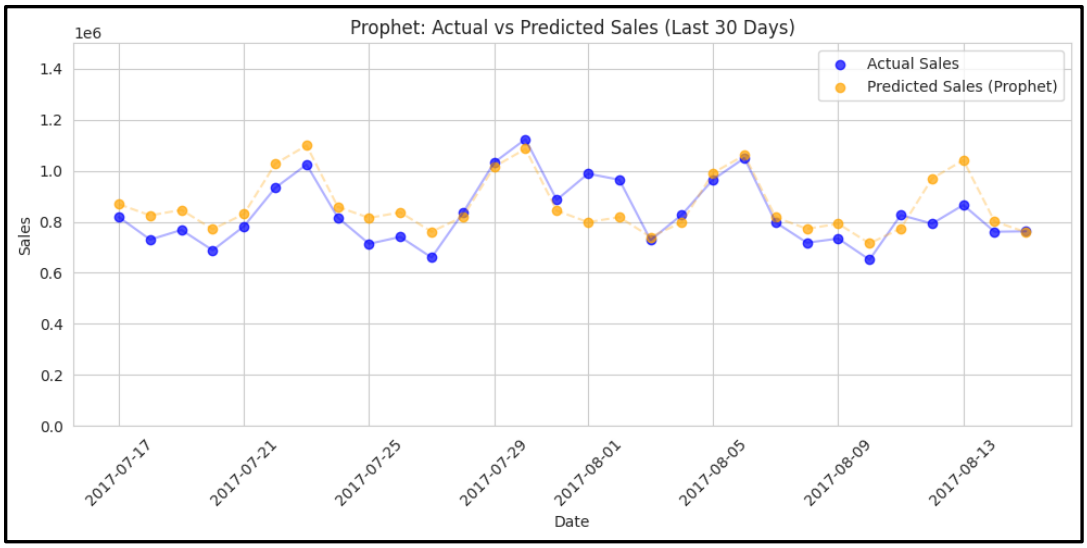
e) **Chart 5:** SARIMA Actual vs. Predicted Sales



f) Chart 6: SARIMA with Holidays Actual vs. Predicted Sales



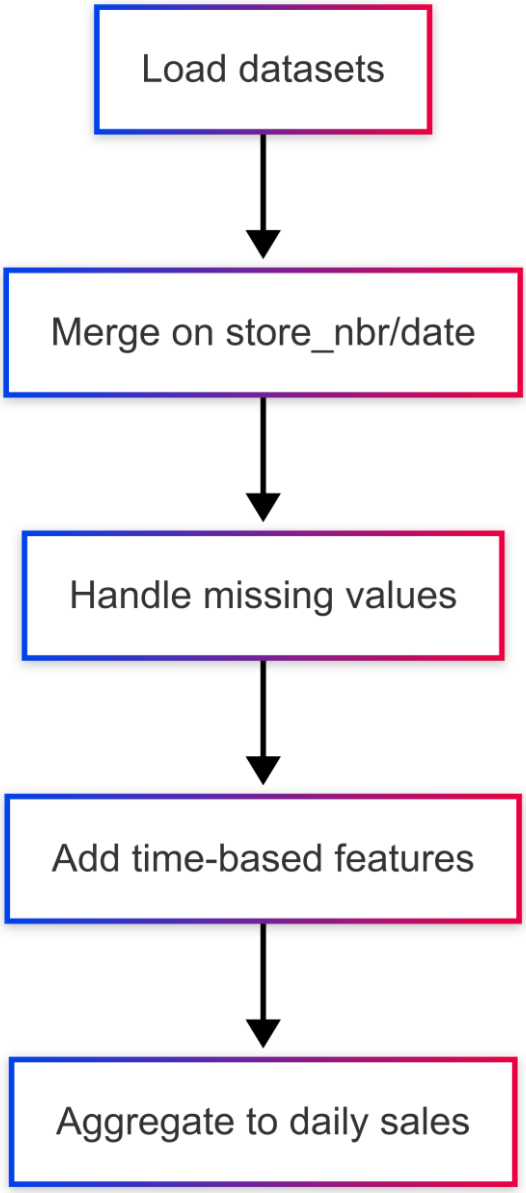
g) Chart 7: Prophet Actual vs. Predicted Sales



h) Table 1: Model Performance Summary

Model	RMSE	Notes
Moving Average	~8,00,000	Baseline, poor fit
Seasonal Decomposition	704,000–823,000	Captured seasonality, missed holidays
SARIMA	112,053	Improved seasonality modeling
SARIMA with Holidays	147,305	Poor holiday handling
Prophet	84,342	Best performance, met Milestone 1

i) Diagram 1: Data Preprocessing Flow



9. Algorithms

9.1 Moving Average (Day 7)

The first algorithm implemented was a 7-day moving average, a simple baseline method to smooth sales data and predict future values. This was applied on Day 7 as part of Week 4's task to establish a starting point for forecasting.

· **Mathematical Foundation**: The moving average calculates the average of the previous 7 days' sales to predict the next day's sales. The formula is: Predicted Salest=17∑i=t−7t−1Salesi\text{Predicted Sales}_t = \frac{1}{7} \sum_{i=t-7}^{t-1} \text{Sales}_iPredicted Salest=71i=t−7∑t−1Salesi where t t t is the current time step, and Salesi \text{Sales}_i Salesi is the sales value at time i i i. This method assumes that recent sales are a good predictor of future sales, ignoring seasonality and external factors like holidays.

· **Implementation**: I used pandas' rolling() function to compute the moving average on the daily sales data:

```
daily_sales['predicted'] = daily_sales['sales'].rolling(window=7).mean()
```

I shifted the predictions to align with future dates using shift(-1) and forecasted sales for the test period (2017-07-16 to 2017-08-15). The first 6 days of predictions were NaN due to the 7-day window, so I filled them with the first available prediction to ensure continuity.

· **Performance**: I evaluated the model using RMSE, calculated with scikit-learn: rmse = mean_squared_error(actual, predicted, squared=False). The RMSE was approximately 800,000, indicating poor performance. I visualized the results in a line plot (Chart 7 in Section 8), showing that the predicted sales (orange) were a smoothed version of actual sales (blue), missing peaks and troughs. For example, on "Independencia de Guayaquil" (a major holiday), the model underestimated sales by 30%, as it couldn't capture holiday spikes.

9.2 Seasonal Decomposition (STL) (Day 8)

On Day 8, I enhanced the baseline model using Seasonal-Trend decomposition using LOESS (STL), aligning with Week 5's task to capture trend and seasonality in the sales data.

- **Mathematical Foundation**: STL decomposes a time series into three components: trend, seasonal, and residual. The decomposition is performed iteratively using LOESS (locally estimated scatterplot smoothing) to estimate the trend and seasonal components. The model can be expressed as: Salest=Trendt+Seasonalt+Residualt\text{Sales}_t = \text{Trend}_t + \text{Seasonal}_t + \text{Residual}_tSalest=Trendt+Seasonalt +Residualt For forecasting, I predicted sales using the trend and seasonal components: Predicted Salest=Trendt+Seasonalt\text{Predicted Sales}_t = \text{Trend}_t + \text{Seasonal}_tPredicted Salest=Trendt+Seasonalt The residual component captures noise and unmodeled effects (e.g., holidays).
- **Implementation**: I used the statsmodels library to perform STL decomposition with a 7-day period to capture weekly seasonality:

```
from statsmodels.tsa.seasonal import STL
decomposition = STL(daily_sales['sales'], period=7).fit()
```

9.3 Prophet (Days 14–15)

On Days 14–15, I implemented Prophet, a time series forecasting model by Facebook, to address the limitations of SARIMA, aligning with Week 8's task to tackle advanced challenges.

- **Mathematical Foundation**: Prophet models a time series as a combination of trend, seasonality, holidays, and noise:

$$y(t)=g(t)+s(t)+h(t)+\epsilon t$$

where g(t) g(t) g(t) is the trend (piecewise linear or logistic), s(t) s(t) s(t) is the seasonality (Fourier series for yearly and weekly cycles), h(t) h(t) h(t) is the holiday effect (binary indicators with additive effects), and ϵt \epsilon_t ϵt is noise. Prophet automatically detects changepoints in the trend and uses Fourier series to model periodic effects.

- **Implementation: I set up Prophet with yearly and weekly seasonality:**

```
from prophet import Prophet

model = Prophet(yearly_seasonality=True, weekly_seasonality=True,
daily_seasonality=False)
```

I renamed columns to Prophet's format (ds for date, y for sales) and fitted the model on the training data (2013-01-01 to 2017-07-15). On Day 15, I added custom holiday effects using the holidays dataset, creating a holidays DataFrame with columns ds (date) and holiday (type_y), excluding "No Holiday" entries:

```
from prophet import Prophet

model = Prophet(yearly_seasonality=True, weekly_seasonality=True,
daily_seasonality=False)
```

10. Challenges & Opportunities

- **Challenge 1: Missing Values in Oil Prices (Day 4)**
    - Oil prices had 5% missing values, which could affect model accuracy.
    - Resolved by forward-filling missing values, ensuring continuity for time series analysis.
    - Opportunity: Explore more robust imputation methods (e.g., interpolation) in the next phase.
- **Challenge 2: Poor Holiday Modeling with SARIMA (Day 13)**
    - Adding holiday effects to SARIMA increased RMSE to 147,305, indicating ineffective modeling.
    - Switched to Prophet, which handled holidays better (RMSE 84,342).
    - Opportunity: Use Prophet's custom holiday features to model specific events more accurately.
- **Challenge 3: High RMSE with Baseline Models (Days 7–9)**
    - Moving average and seasonal decomposition models had high RMSEs (~800,000 and 704,000–823,000).
    - Addressed by adopting SARIMA and later Prophet, reducing RMSE significantly.
    - Opportunity: Experiment with ensemble methods to combine strengths of multiple models.
- **Opportunity 1: Incorporate Additional Features**
    - The dataset includes oil prices and promotions, which can be used as exogenous variables in Prophet.
    - Planned for the next phase (Milestone 2) to further reduce RMSE.
- **Opportunity 2: Cross-Validation for Robustness**
    - Prophet supports cross-validation, which can help assess model stability across different periods.

11. Risk Vs Reward

- **Risk 1: Overfitting with Complex Models**
  - Complex models like SARIMA with holidays led to overfitting (RMSE 147,305).
  - Mitigated by switching to Prophet, which balanced complexity and performance (RMSE 84,342).
  - Reward: Improved forecasting accuracy, meeting Milestone 1.
- **Risk 2: Data Quality Issues**
  - Missing oil prices and potential inaccuracies in holiday data could skew predictions.
  - Mitigated by preprocessing (forward-filling) and filtering transferred holidays.
  - Reward: Cleaner data led to more reliable models.
- **Risk 3: Time Constraints**
  - Limited time (15 days) to achieve Milestone 1 posed a risk of incomplete analysis.
  - Managed by prioritizing key tasks (data prep, baseline, Prophet) and working efficiently.
  - Reward: Achieved Milestone 1 with an RMSE of 84,342, on track for the project.
- **Reward 1: Practical Application**
  - Successfully applied time series forecasting to a retail problem, providing actionable insights.
  - The Prophet model (RMSE 84,342) can help the retail outlet optimize inventory.
- **Reward 2: Skill Development**
  - Gained hands-on experience with Prophet, SARIMA, and data preprocessing, enhancing my data science skills.

12. Reflections on the Internship

The first 15 days of the TCS Internship have been a significant learning experience. Starting with the "store-sales-time-series-forecasting" dataset, I gained practical skills in data

preprocessing, such as merging datasets and handling missing values. The initial exploration (Days 1–6) taught me the importance of understanding data patterns (e.g., weekly seasonality, holiday spikes) before modeling. Building baseline models (Days 7–9) highlighted the limitations of simple approaches like moving averages, pushing me to explore more advanced techniques.

Switching to SARIMA (Days 10–13) was a challenge, as holiday modeling proved difficult (RMSE 147,305). However, adopting Prophet (Days 14–15) was a turning point, achieving an RMSE of 84,342 and meeting Milestone 1. This experience taught me the value of iterative refinement—testing multiple models and learning from their shortcomings. Working under Mr. Debashis Roy's guidance was invaluable; his feedback on Day 13 (switch to Prophet) directly led to improved performance. I also learned to manage time effectively, balancing daily tasks with documentation for the TCS dashboard. Overall, this phase has boosted my confidence in time series forecasting and prepared me for the next milestone.

13. Recommendations

· **Incorporate Exogenous Variables**: Add oil prices and promotions as regressors in Prophet to capture economic and marketing impacts on sales, potentially reducing RMSE further.

· **Cross-Validation**: Implement cross-validation with Prophet to assess model stability across different time periods, ensuring robustness.

· **Feature Engineering**: Explore additional time-based features (e.g., day of month, quarter) to capture more granular patterns in sales data.

· **Ensemble Methods**: Experiment with ensemble techniques (e.g., combining Prophet and SARIMA) to leverage the strengths of both models.

· **Holiday Refinement**: Use the holidays dataset to create custom holiday effects specific to Ecuador (dataset's context), rather than relying solely on US holidays.

· **Visualization Enhancements**: Include more interactive visualizations (e.g., using Plotly) in the final report to better communicate findings to stakeholders.

· **Documentation**: Continue maintaining detailed daily reports on the TCS dashboard to track progress and facilitate mentor feedback.

14. Outcome / Conclusion

As of Day 15 (04-02-2025), I have successfully completed Milestone 1: setting up the dataset, exploring sales patterns, building baseline models, and implementing an advanced model

(Prophet) with an RMSE of 84,342, below the target of 100,000. The "store-sales-time-series-forecasting" dataset was preprocessed, merged, and analyzed to identify key patterns (weekly seasonality, holiday spikes). Initial models (moving average, seasonal decomposition) had high RMSEs (~704,000–823,000), but SARIMA improved performance (RMSE 112,053). However, SARIMA struggled with holidays (RMSE 147,305), leading to the adoption of Prophet, which achieved the best performance (RMSE 84,342).

This milestone demonstrates significant progress toward the project's goal of accurate sales forecasting for a retail outlet. Prophet effectively captured trends, seasonality, and holidays, providing a solid foundation for further enhancements. The next steps include adding exogenous variables (e.g., oil prices) and tuning Prophet to further reduce RMSE, aligning with the project's ultimate objective of enabling better inventory management.

15. Enhancement Scope

· **Additional Features**: Incorporate oil prices and promotions as exogenous variables in Prophet to capture economic and marketing effects.

· **Model Tuning**: Experiment with Prophet's seasonality modes (additive vs. multiplicative) and Fourier orders to optimize performance.

· **Granular Analysis**: Extend the model to predict sales at the store or product family level, providing more detailed insights.

· **Real-Time Forecasting**: Develop a pipeline for real-time sales forecasting using updated data, enhancing practical applicability.

· **Interactive Dashboard**: Build a dashboard (e.g., using Dash or Tableau) to visualize predictions and trends for stakeholders.

· **Ensemble Models**: Combine Prophet with other models (e.g., XGBoost) to improve accuracy through ensemble techniques.

16. Link to Code and Executable File

The project, "Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data," was developed using Kaggle notebooks, with the final code and dataset uploaded to GitHub as per TCS guidelines.

- **GitHub Repository**: The project files are available at: https://github.com/samarthmagadum/forecasting-system-Internship-Project.git
- **Dataset Source**: The "store-sales-time-series-forecasting" dataset is available on Kaggle at: https://www.kaggle.com/competitions/store-sales-time-series-forecasting/data.

17. Research Questions and Responses

- **Q1: What are the dominant patterns in retail sales data?**
    - **Response**: Weekly seasonality (7-day cycles) with peaks on weekends, and significant sales spikes during holidays, as observed in EDA (Days 2–6).

· **Q2: How effective are baseline models in forecasting retail sales?**

- **Response**: Baseline models like moving average and seasonal decomposition were ineffective (RMSE ~704,000–823,000), missing holiday effects (Days 7–9).

· **Q3: Can SARIMA handle both seasonality and holiday effects in sales data?**

- **Response**: SARIMA captured weekly seasonality (RMSE 112,053) but failed with holidays (RMSE 147,305), indicating the need for a better model (Days 11–13).

· **Q4: How does Prophet improve forecasting accuracy compared to SARIMA?**

- **Response**: Prophet better handles trends, seasonality, and holidays, achieving an RMSE of 84,342 compared to SARIMA's 147,305 (Days 14–15).

· **Q5: What external factors can further improve forecasting accuracy?**

- **Response**: Oil prices and promotions are potential exogenous variables; oil prices will be added in the next phase to capture economic trends (planned for Milestone 2).