ORAIMO SALES FORECASTING (TIME SERIES ANALYSIS)

PHASE 5 PROJECT

DATE: 29/04/2025

EVERYTHING DATA GROUP MEMBERS

Peter Nyakundi

Calvince Kaunda

Gloryann Otieno

Samuel Marimbi

Sanayet Sankaine

Table of Contents

1. Business Understanding

Oraimo faces fluctuating sales revenue across cities and product types. Without accurate sales forecasting, the company risks revenue loss, poor cash flow planning and missed growth opportunities. This project aims to build a model to predict daily sales revenue for October 6–31, 2024 using historical data. The forecast will support better financial planning, budgeting and strategic decision-making.

1.1 Problem Statement

Despite having access to historical sales data, Oraimo lacks a reliable method to forecast future revenue. The core problem is the absence of a predictive system that can anticipate short-term sales performance, particularly for critical planning periods. This gap limits the company's ability to make timely, data-driven financial and operational decisions.

1.2 Objectives

- To analyze historical sales data and understand revenue trends across time and product types.
- To develop and validate a machine learning model that accurately forecasts daily sales revenue for the period of October 5–31 2024.
- To enable strategic decision-making by offering data-driven insights to guide marketing, promotions and expansion initiatives.
- To generate actionable forecasts that aid in financial planning, marketing strategy and business growth initiatives.

1.2 Stakeholders

- Oraimo Management Team - for strategic planning and data-driven decision-making
- Sales Team - to align sales targets and campaigns with forecasted revenue trends

- Finance Team - to support accurate budgeting, forecasting, and cash flow management
- Retailers & Shop Managers - to prepare for demand surges and ensure product availability
- Data & Business Analysts - to develop, validate and monitor the forecasting model’s performance

1.3 Metrics of Success

- Ensure the final forecast aligns with historical patterns and any seasonal demand shifts
- Provide revenue insights that directly support financial planning for October 2024
- Achieve a forecasting model with the lowest possible **RMSE** and **MAE**

2. Data Understanding

This stage involves gathering data, understanding its structure and assessing its quality to ensure suitability for analysis.

2.1 Data Collection

The Oraimo Sales dataset is from a Data Science community, **DataVerse.** The dataset contains historical sales records from January to October 2024 by Oraimo across multiple shops across the country. The dataset contains 19,802 rows and 14 columns.

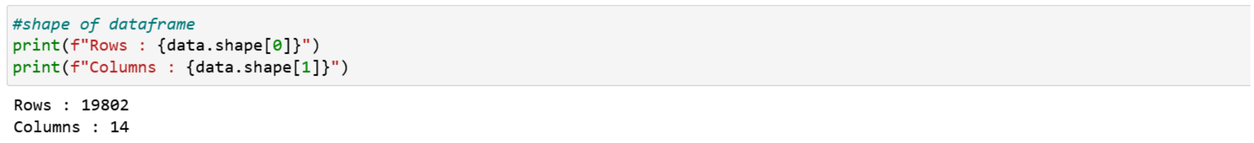
2.2 Data Description

The dataset includes daily sales transactions from January to early October 2024. Key fields include:

- date - used as index for time series
- city and product_type - to capture regional and category trends
- amount - target variable representing total revenue per sale

2.3 Data Quality and Initial Inspection

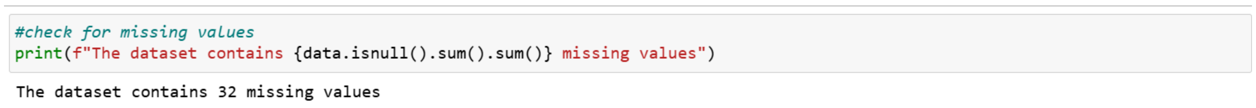
- The dataset contains 19,802 rows and 14 columns.



```
#shape of dataframe
print(f"Rows : {data.shape[0]}")
print(f"Columns : {data.shape[1]}")
```

```
Rows : 19802
Columns : 14
```

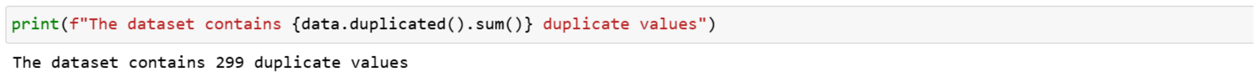
- There are 32 missing values in the dataset.



```
#check for missing values
print(f"The dataset contains {data.isnull().sum().sum()} missing values")
```

```
The dataset contains 32 missing values
```

- There are 299 duplicate entries identified.



```
print(f"The dataset contains {data.duplicated().sum()} duplicate values")
```

```
The dataset contains 299 duplicate values
```

3.Data Cleaning

- Columns such as city_2, year, source, shop_name, proper_date and model were dropped due to irrelevance.

```
df = df.drop(["city_2","year","proper_date","source","shop_name","model"], axis=1)
df.head()
```

- Inconsistencies were found in the month and type columns (mix of upper/lowercase), so values were standardized to uppercase.

```
df["month"] = df["month"].str.upper()
df["type"] = df["type"].str.upper()
```

- The type column was renamed to product_type for clarity.

```
#rename type column
df = df.rename(columns = {"type":"product_type"})
```

- Spelling and naming inconsistencies were addressed in both city (e.g., "Ruiru Nairobi" → "Nairobi") and product_type columns.

```
# Standardize 'City' column
city_cleaning_map = {
    # Nairobi variations
    'NAIROB': 'NAIROBI', 'NAROBI': 'NAIROBI', 'NAIROBI MERCY': 'NAIROBI','NAIROBI  MERCY': 'NAIROBI', 'NAIROBI   MERCY': 'NAIROBI
    'NAIRROBI  MERCY': 'NAIROBI', 'NAIRIBO  MERCY': 'NAIROBI','NAIROB  OSMAN': 'NAIROBI', 'NAIROBI  OSMAN': 'NAIROBI',
    'NAIROB I OSMAN': 'NAIROBI', 'NAROBI  OSMAN': 'NAIROBI','NAIROBI OSMAN': 'NAIROBI', 'NAIROBIOSMAN': 'NAIROBI','RUIRU NAIROBI'
    'TOPAZ MOBILE ACCESSORIESNAIROBI': 'NAIROBI','NAIROBI  MECY' : 'NAIROBI','NAROBI  MERCY' : 'NAIROBI', 'CABANAS': 'NAIROBI',
    # Nairobi estates to Nairobi
    'LANGATA': 'NAIROBI', 'KITENGELA': 'NAIROBI', 'BURUBURU': 'NAIROBI', 'RONGAI': 'NAIROBI',
    # Misspelled cities
    'MMMBASA': 'MOMBASA', 'MMMBAMA': 'MOMBASA','KISIMU': 'KISUMU', 'RUNYEJES': 'RUNYENJES','NANYUKU': 'NANYUKI', 'GARISA': 'GARIS
    # Coast towns to 'COAST'
    'MOMBASA': 'COAST', 'KILIFI': 'COAST', 'MALINDI': 'COAST','UKUNDA': 'COAST', 'DIANI': 'COAST', 'NYALI': 'COAST',
    'LIKONI': 'COAST', 'BAMBURI': 'COAST', 'VOI': 'COAST',
    # Nyeri County
    'KARATINA': 'NYERI',
    # Migori County
    'RONGO': 'MIGORI', 'SIRARE': 'MIGORI', 'ISIBANIA': 'MIGORI',
    #Kiambu
    'RUIRU': 'KIAMBU', 'EXCEL': 'KIAMBU', 'THIKA': 'KIAMBU', 'OLJOROROK': 'NYAHURURU','LITEIN': 'KERICHO','MAUA': 'MERU', 'RUNYEN
}

# Apply the cleaning map
df['city'] = df['city'].replace(city_cleaning_map)
```

- The date column was converted to datetime format and set as the index, preparing it for time-based analysis.

```
#convert date column to date time
from datetime import datetime
df["date"] = pd.to_datetime(df["date"])
df = df.set_index("date", drop = True)
```

- product_type values were grouped into 5 main categories to simplify modeling and improve interpretability.

```
categories = {
    # Power Devices
    'CHARGER KIT': 'Power Devices','CAR CHARGER KIT': 'Power Devices','MODULATOR CAR CHARGER KIT': 'Power Devices','BATTERY': 'Po
    'BATTERY 1000MAH': 'Power Devices','POWER BANK': 'Power Devices','POWERBOAT': 'Power Devices','STORAGE POWER': 'Power Devices
    'SOLAR GENERATOR': 'Power Devices','CONNECTOR': 'Power Devices','WATCH CHARGING CABLE': 'Power Devices','SOCKET': 'Power Devi
    'EXTENSION CABLE': 'Power Devices','EXTENTION': 'Power Devices',
    # Audio Devices
    'EARPHONES': 'Audio Devices','EARPHONESS': 'Audio Devices','TWS': 'Audio Devices','TWS EARPHONES': 'Audio Devices',
    'WIRELESS HEADSET': 'Audio Devices','HEADPHONES': 'Audio Devices','EARBUDS': 'Audio Devices','FREEPODS': 'Audio Devices',
    'OPEN EAR': 'Audio Devices','SPACEPODS': 'Audio Devices','BUDS': 'Audio Devices','NECK EARPHONES': 'Audio Devices',
    'NECKBAND WIRELESS': 'Audio Devices','SPEAKER': 'Audio Devices','WIRELESS SPEAKER': 'Audio Devices',
    # Smart and Office Devices
    'SMART WATCH': 'Smart and Office Devices','WIRELESS KEYBOARD': 'Smart and Office Devices','KEYBOARD KIT': 'Smart and Office D
    'MOUSE': 'Smart and Office Devices','MEMORY CARD': 'Smart and Office Devices','FLASH DISK': 'Smart and Office Devices',
    'MEMORY': 'Smart and Office Devices','MIFI': 'Smart and Office Devices','COMBO SET': 'Smart and Office Devices',
    'ORAIMO': 'Smart and Office Devices','OCD-X93': 'Smart and Office Devices','AIRBUS': 'Smart and Office Devices','E-BIKE': 'Sm
    # Personal Care
    'TRIMMER': 'Personal Care','HAIR CLIPPER': 'Personal Care','SHAVER': 'Personal Care','SHAIVER': 'Personal Care','BODY FAT SCA
    'HANDHELD VACUUM': 'Personal Care','HAND VACUUM CLEANER': 'Personal Care','NECKLACE': 'Personal Care','AMLET': 'Personal Care
    # Home Appliances
    'ELECTRIC KETTLE': 'Home Appliances','BLENDER': 'Home Appliances','ELECTRIC FAN': 'Home Appliances','TUBFULS': 'Home Applianc
    'CAR BRACKET': 'Home Appliances','CLIPPER':  'Home Appliances'
}

df["product_category"] = df["product_type"].map(categories).fillna("Unknown")
```

- Duplicate values were dropped.

```
#drop duplicates and re-evaluate shape

df = data.drop_duplicates()

df.shape
```

```
(19503, 14)
```

- Missing values were dropped.

```
#drop missing values
df = df.dropna(axis = 0)

#re-evaluate shape after dropping the rows
df.shape
```

```
(19480, 14)
```

We **chose not to drop outliers** because:

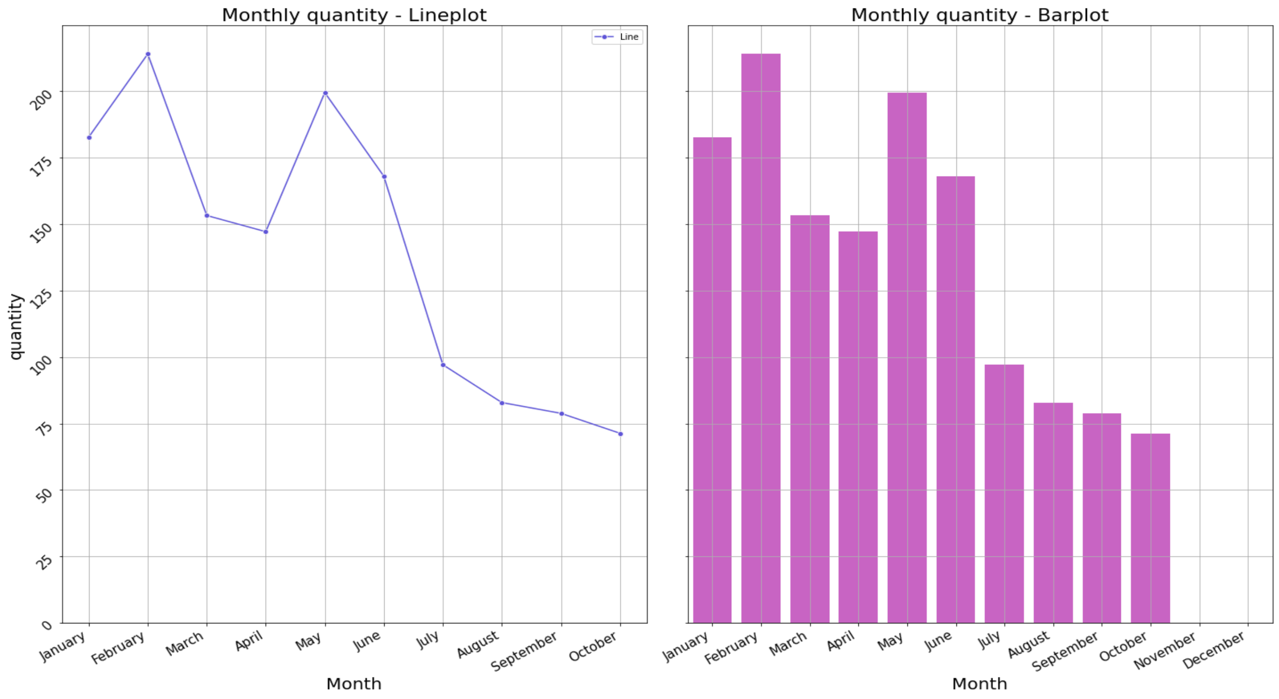
- High sales values reflect valid bulk purchases, not data errors.
- These large transactions are an essential part of Oraimo’s business model.
- Removing them would reduce the representativeness of the dataset and bias the forecasting model.

4. Exploratory Data Analysis

4.1 Univariate Analysis

Univariate analysis involves examining the distribution and summary statistics of individual variables.

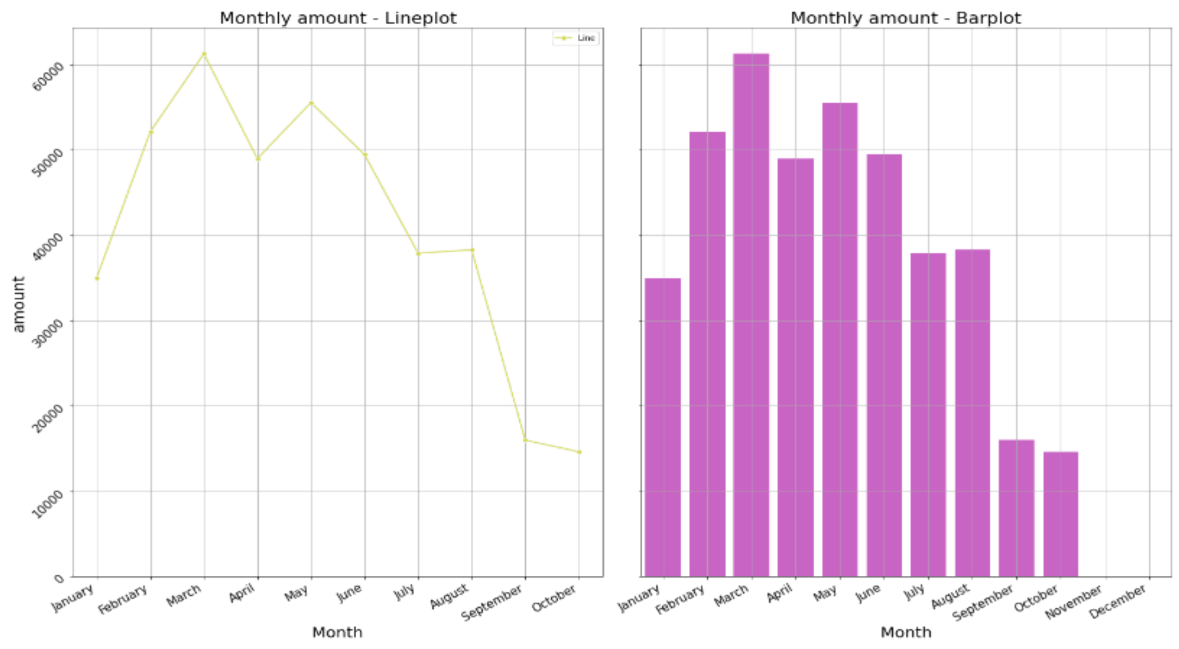
4.1.1 Average Monthly Quantity



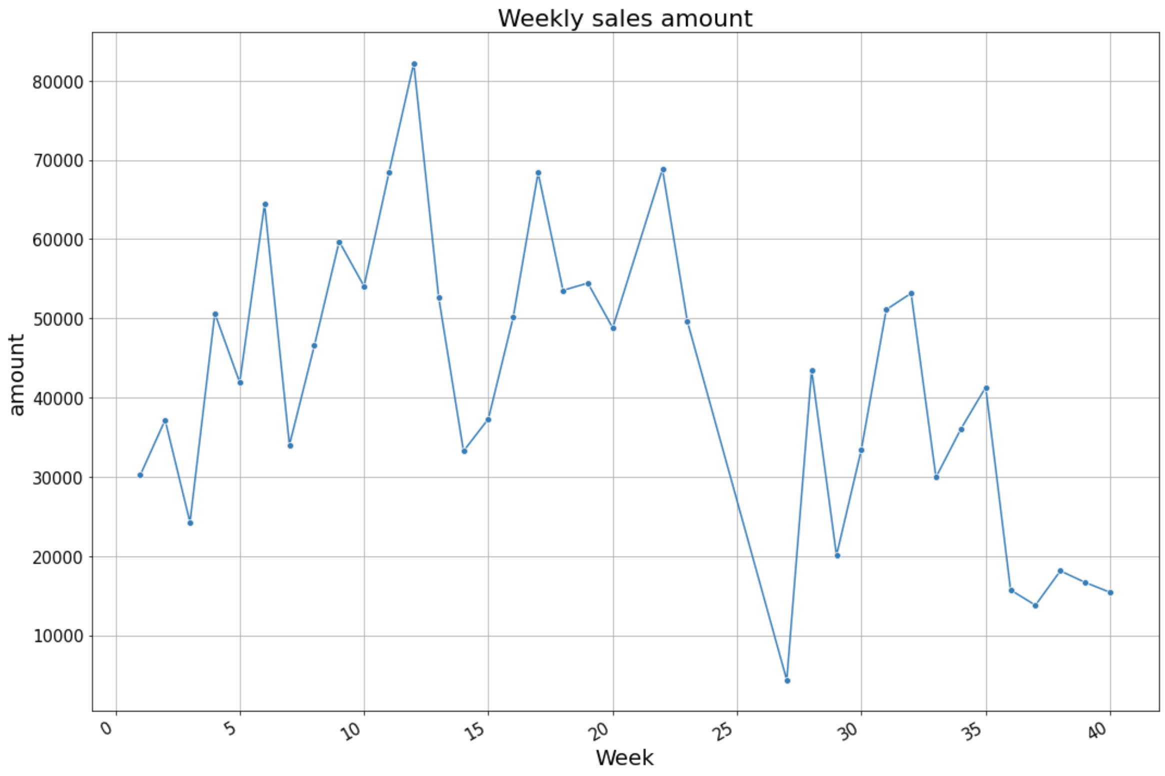
- February and May show noticeable spikes in sales, suggesting a surge in demand during these months. After May, there is a sharp decline in sales, which continues all the way through to October, indicating a period of lower demand or potential stock availability issues.

4.1.2 Average Monthly Sales Amount

- The average sales amount shows spikes in March and May, indicating possible seasonal demand or special events.
- These variations can be linked to changes in customer purchasing behavior, possibly due to external factors such as promotions, seasonality, or product availability.

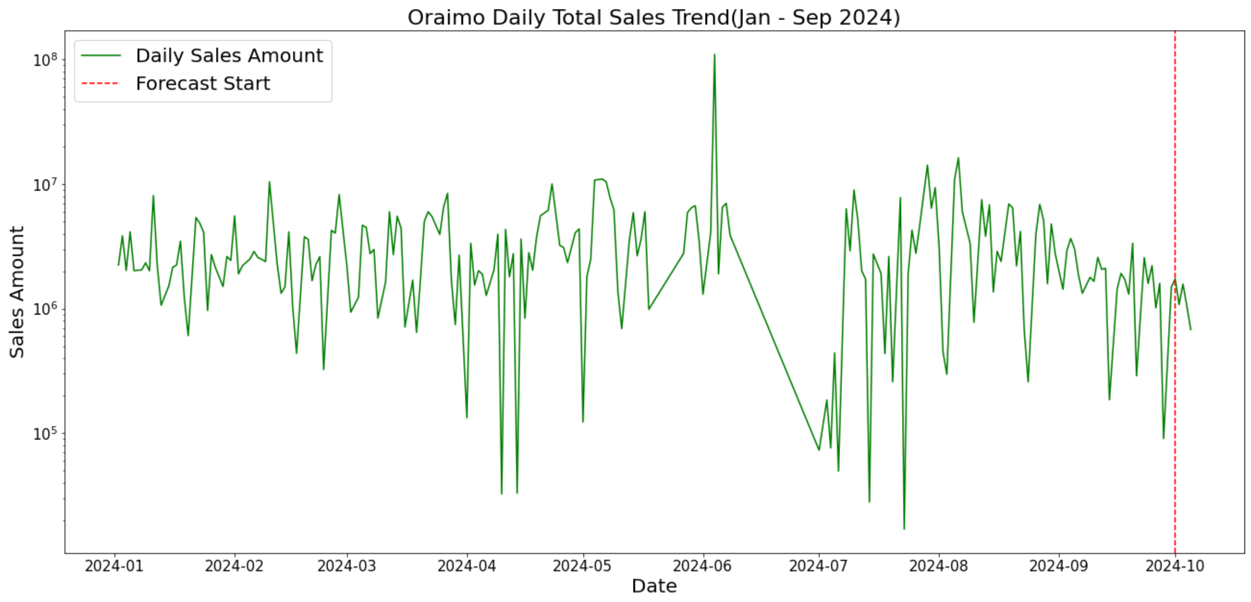


4.1.3 Average Weekly Sales Amount



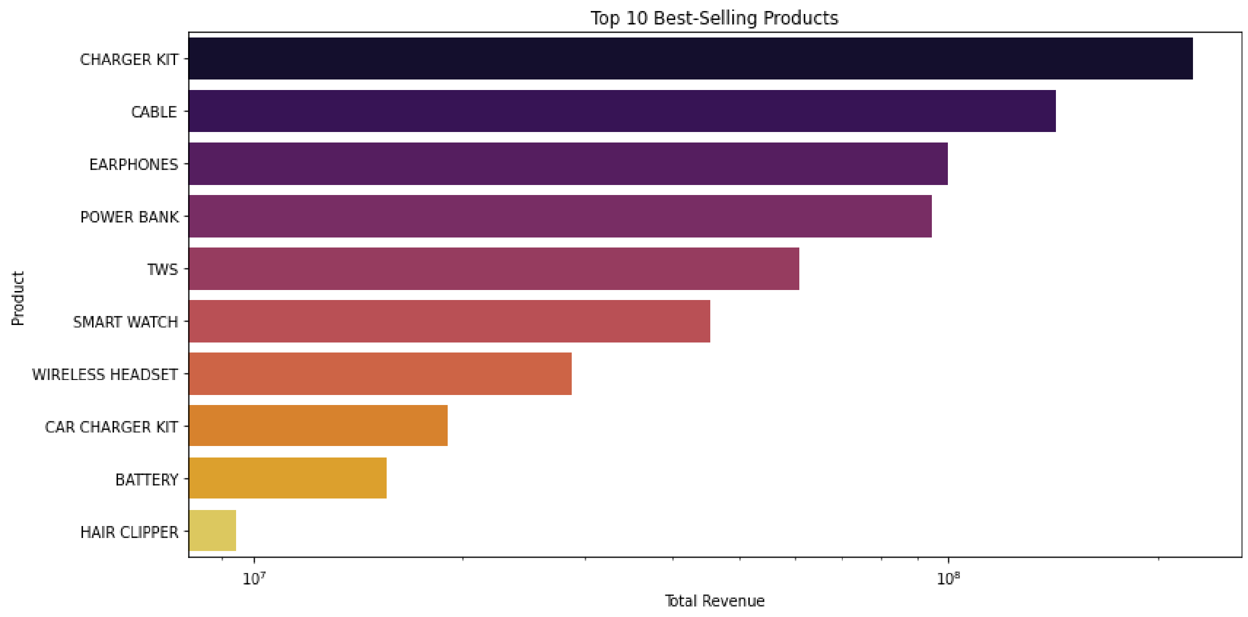
- The average weekly sales amount shows spikes between weeks 5 and 15, indicating possible seasonal demand or special events.

4.1.4 Total Daily Sales Trend

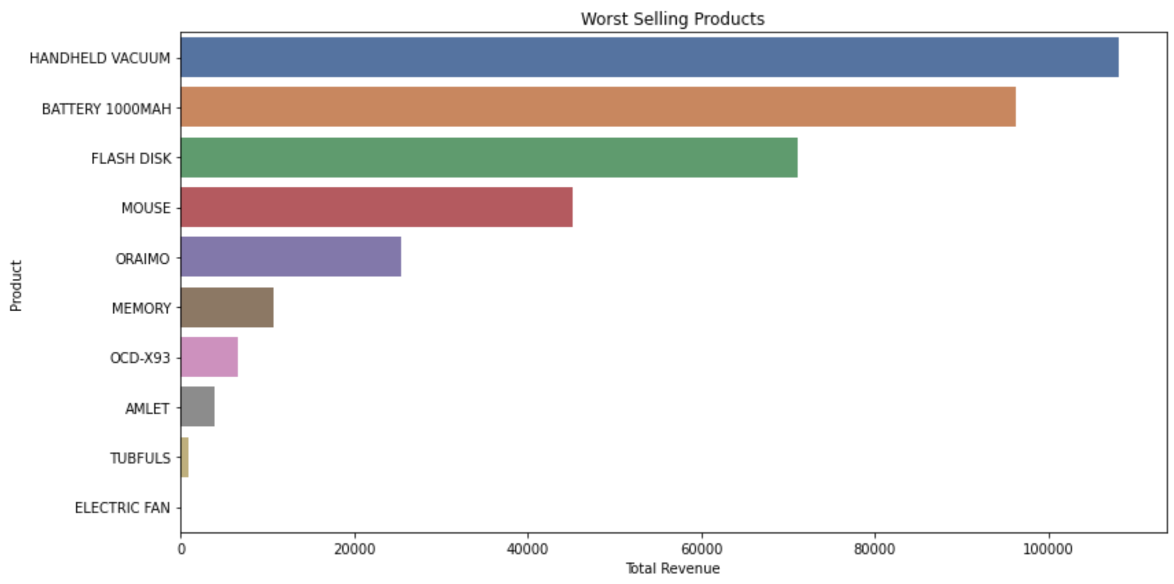


- Forecasting starting point is indicated from October 2024.
- There is a significant daily fluctuation in sales amounts throughout the period indicating there are no seasonal/cyclic patterns.

4.1.5 Best and Worst Selling Products

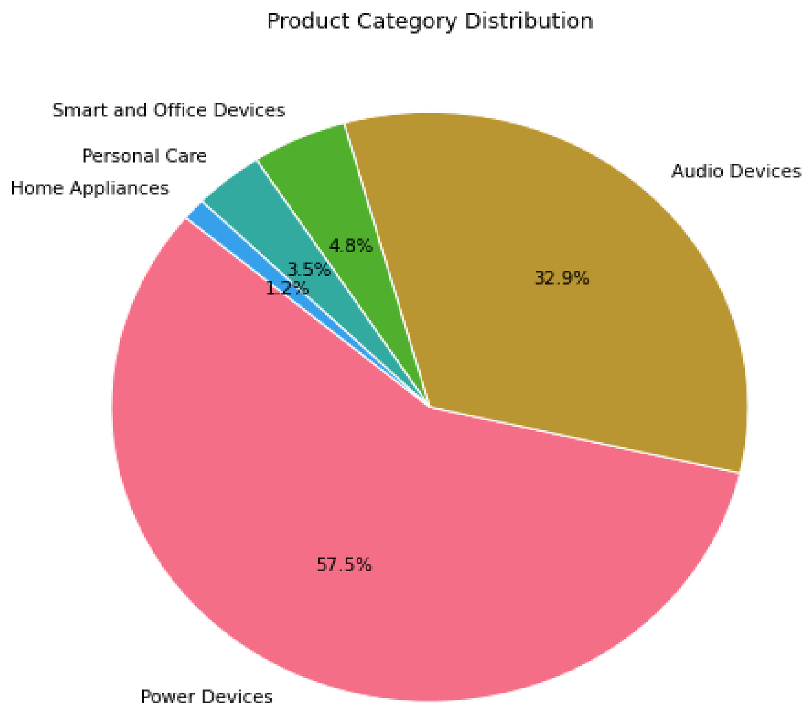


- The best performing products are charger kits, cables and earphones based on sales amount.
- This implies that they bring in the most sales revenue over the entire period.



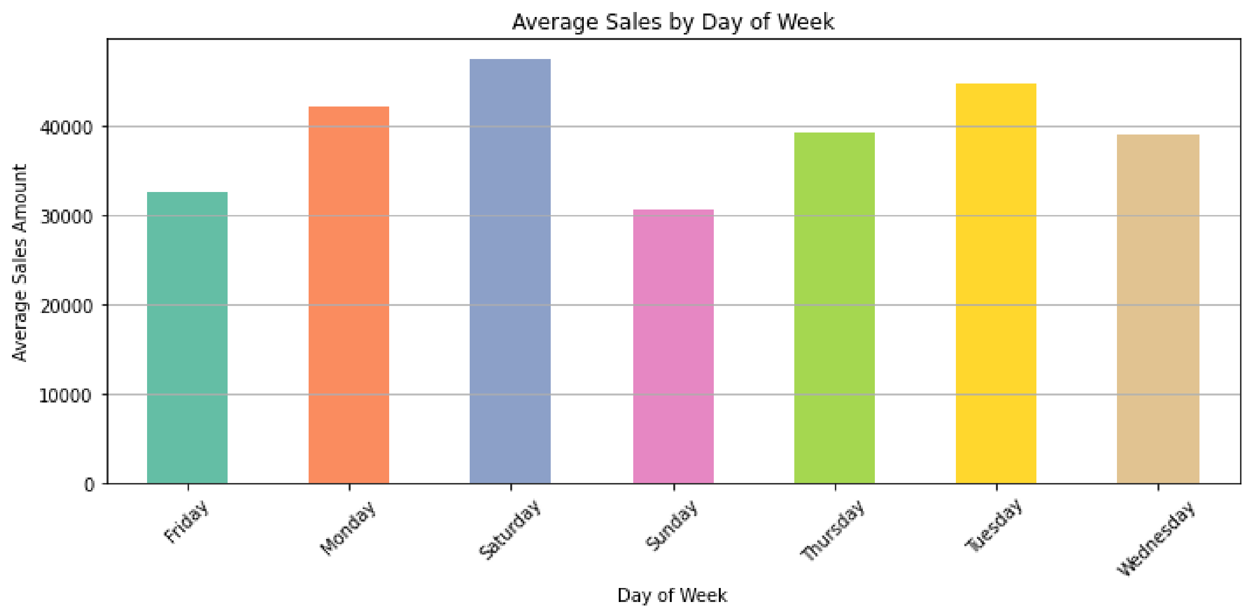
- The worst performing products are handheld vacuum, battery and flash disk.
- These products bring in the least amount of sales revenue. This could either be due to high unit price or customer taste which leads to least number of purchases of the product.

4.1.6 Product Category Distribution



- The most popular category is the Power Devices category with a total share of 57.7% which is more than half. Products under this category are the most purchased by customers hence should be stocked often.
- 32.9% of the goods are the Audio Devices category including products such as earphones.
- Home appliances are the least popular occupying less than 2% of the products in the market.

4.1.7 Average Daily Sales per Day of the Week

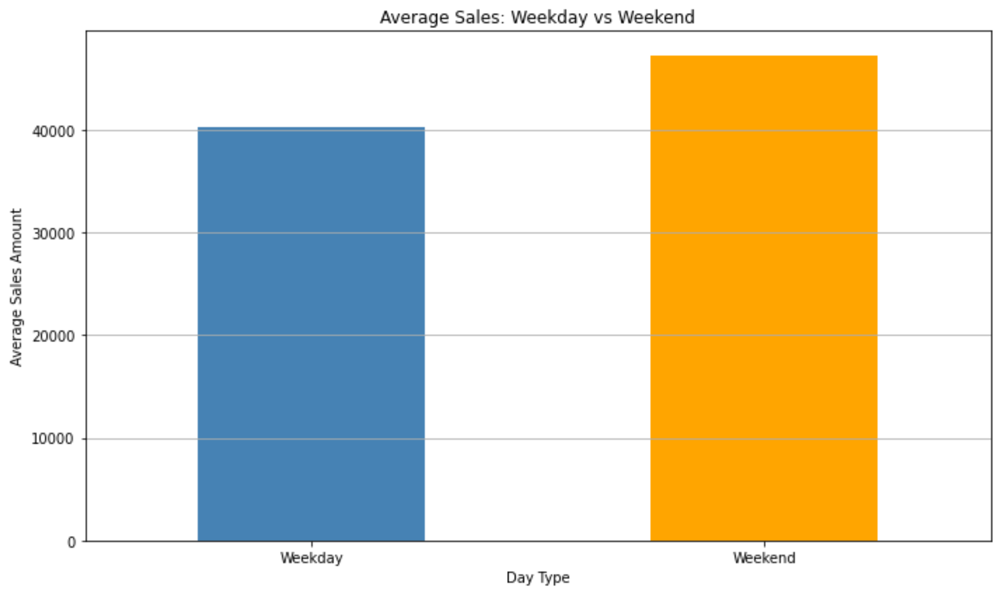


- Saturdays have the highest foot traffic where average sales across the week is highest.
- Sundays have the lowest average sales.
- A spike is noticed from Mondays through to Thursday whereas sales drop on Fridays and spike on Saturdays.

4.2 Bivariate Analysis

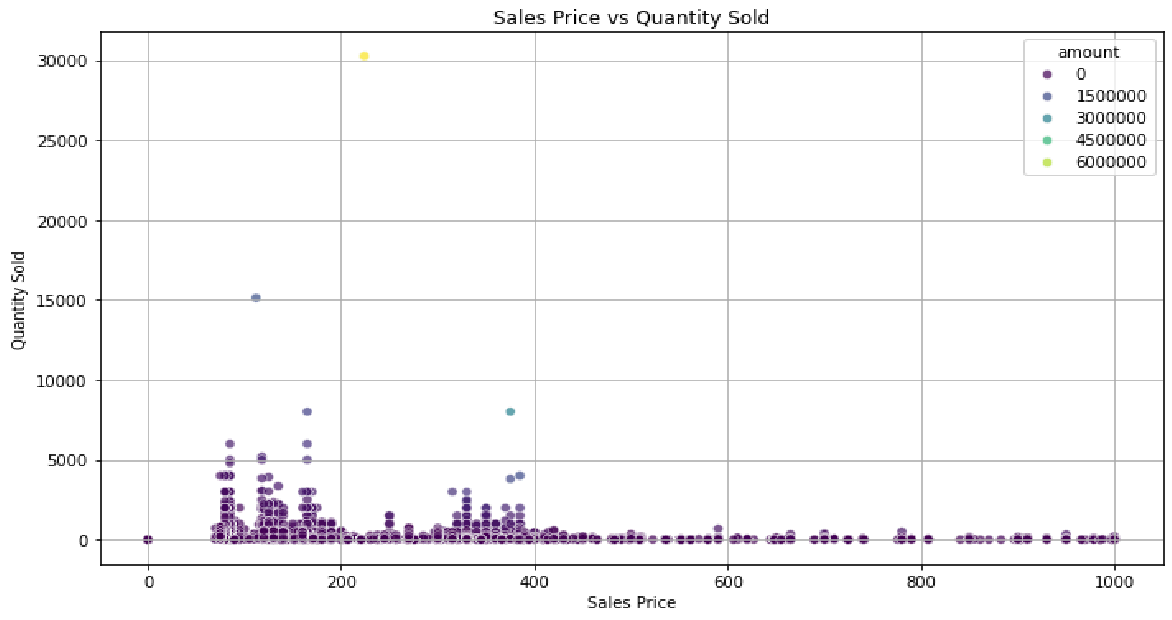
Bivariate analysis involves examining the relationship between two variables to understand how one may affect the other.

4.2.1 Weekday vs Weekend Sales



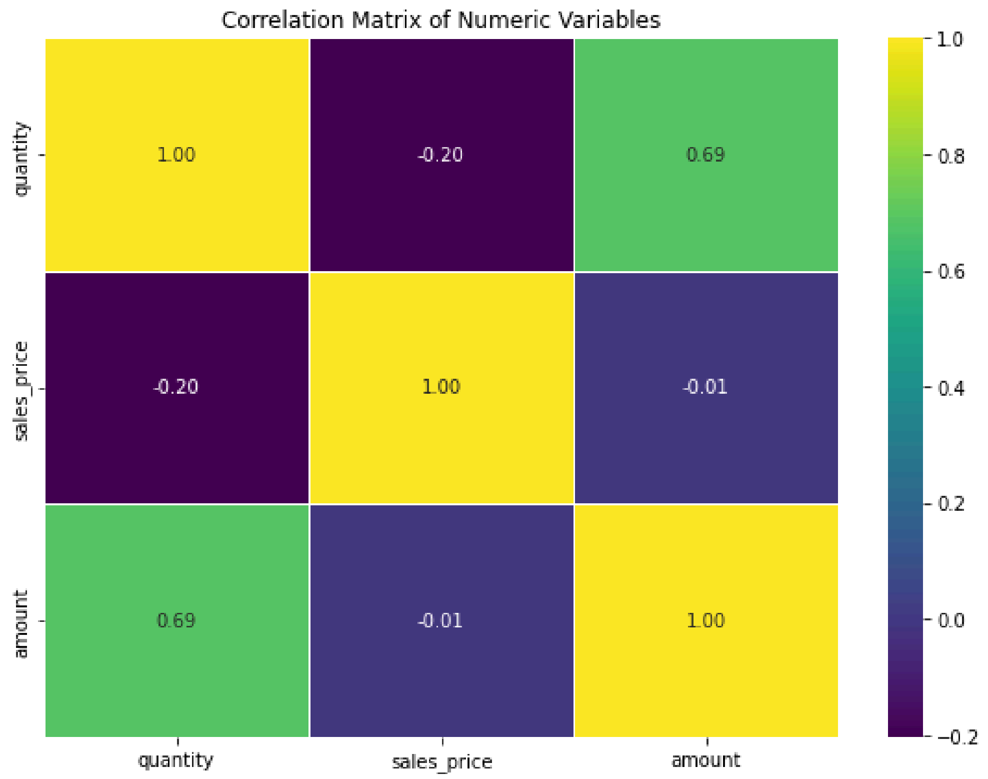
- Weekend sales are higher than weekday sales, as shown by the average sales amount comparison.
- This suggests that customers are likely spending more on weekends, possibly due to higher foot traffic, more promotional activities, or people having more free time to shop.
- The difference in sales could be leveraged for planning inventory, staffing and marketing strategies to maximize sales during weekends.

4.2.2 Sales Price vs Quantity Sold



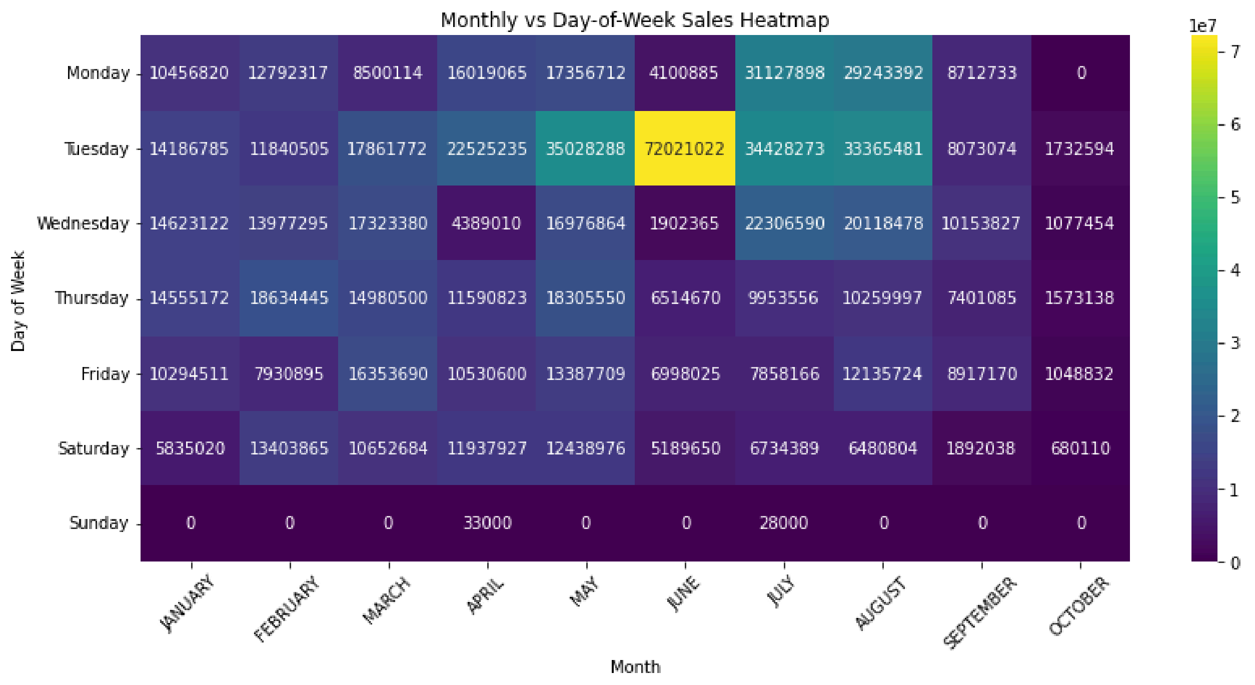
- Most Sales Occur at Lower Price Points: The majority of the data points are clustered on the left side of the plot, indicating that most products are sold at relatively lower price points. This suggests that the customer base is more responsive to lower pricing, implying a potential opportunity for targeting price-sensitive customers.
- Quantity Sold is Generally Low: Most data points are concentrated near the bottom of the y-axis, representing low quantities sold.
- Few High Outliers: A few significant outliers can be identified, such as one product that sold approximately 30,000 units at a price under 200, marked yellow in the scatter plot, indicating it generated the highest amount.

4.2.3 Correlation Analysis of Numeric Variables



- Quantity and Amount: A moderate positive correlation of 0.69 suggests that as quantity sold increases, the total amount (sales value) also tends to increase.
- Quantity and Sales Price: A weak negative correlation of -0.20 indicates that higher quantities are not strongly associated with higher sales prices.
- Sales Price and Amount: A very weak negative correlation of -0.01 suggests that there is virtually no relationship between sales price and the total amount.

4.2.4 Monthly vs Day-of-Week Sales Heatmap



	JANUARY	FEBRUARY	MARCH	APRIL	MAY	JUNE	JULY	AUGUST	SEPTEMBER	OCTOBER
Monday	10456820	12792317	8500114	16019065	17356712	4100885	31127898	29243392	8712733	0
Tuesday	14186785	11840505	17861772	22525235	35028288	72021022	34428273	33365481	8073074	1732594
Wednesday	14623122	13977295	17323380	4389010	16976864	1902365	22306590	20118478	10153827	1077454
Thursday	14555172	18634445	14980500	11590823	18305550	6514670	9953556	10259997	7401085	1573138
Friday	10294511	7930895	16353690	10530600	13387709	6998025	7858166	12135724	8917170	1048832
Saturday	5835020	13403865	10652684	11937927	12438976	5189650	6734389	6480804	1892038	680110
Sunday	0	0	0	33000	0	0	28000	0	0	0

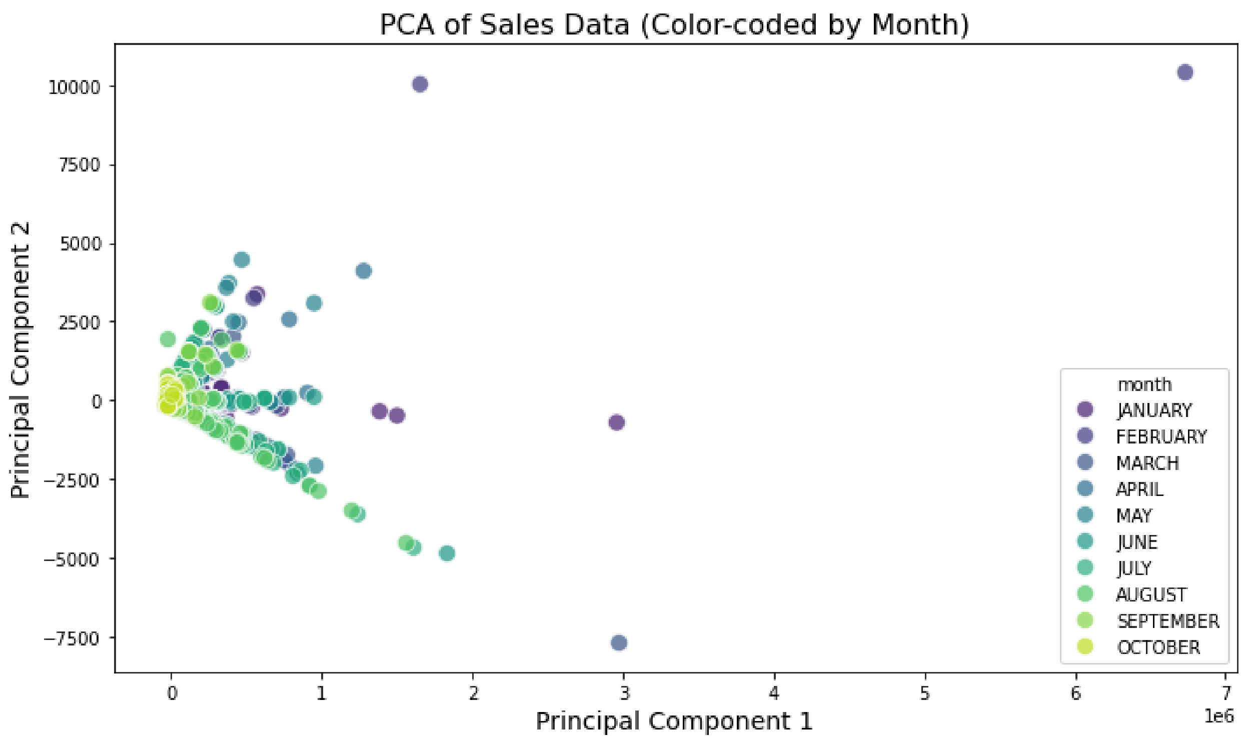
- High activity is observed on Tuesdays and Mondays, suggesting strong beginning-of-week demand.
- June recorded the highest sales, possibly indicating promotional campaigns or seasonal demand spikes.
- Lower sales on Sundays and Saturdays may reflect reduced business activity or customer engagement on those days

4.3 Multivariate Analysis

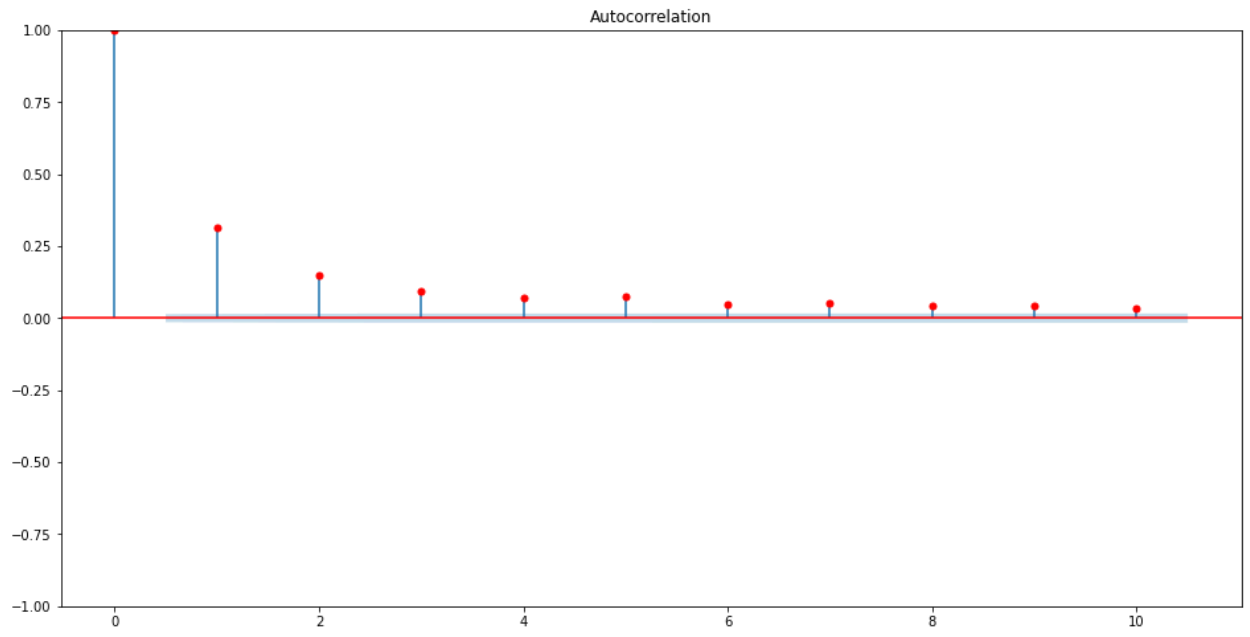
Multivariate analysis involves examining more than two variables at the same time to understand patterns, relationships or structures in the data.

4.3.1 Dimensionality Reduction with PCA

- Cluster near origin: Most sales data points cluster around the center, indicating consistent patterns across months (e.g. similar prices and quantities sold).
- Dispersed months: January and February show wider spread, likely due to seasonal demand or promotions.
- Seasonal clusters: May and September stand out, possibly marking peak periods with higher sales or pricing changes.
- PC1 spread: Principal Component 1 captures major variance—likely tied to quantity sold or sales price shifts



4.3.2 Autocorrelation Plot



- We have significant initial spikes indicating a significant positive autocorrelation at the first 2 lags (lag 1 and lag 2). These spikes are clearly above zero.
- After the second lag, the autocorrelation coefficients decrease and appear to be getting closer to zero, forming an exponential curve like shape.

5. Feature Engineering

In this section we create new features and transforming existing ones to help machine learning models understand the data better. In sales forecasting, good features can help capture patterns like seasonality, trends and sales behavior in different cities or product types.

5.1 Aggregating Sales to Daily Level

- We resampled the transactional sales data to obtain the total daily revenue by summing the amount column for each date. This step transforms the dataset from event-level granularity to a daily time series, which is suitable for forecasting.
- The index was then reset to make the date column accessible for further feature extraction.

```
# 5.1 Aggregate to Daily Total Sales(Forecasting Daily Sales)
daily_sales = df['amount'].resample('D').sum().to_frame().rename(columns={'amount': 'daily_amount'})

#Groups  sales data by day ('D' stands for daily) and sums the total sales amount ('amount') for each day.

# Reset index to access the proper date as a column to extract day, week etc.
daily_sales = daily_sales.reset_index()
```

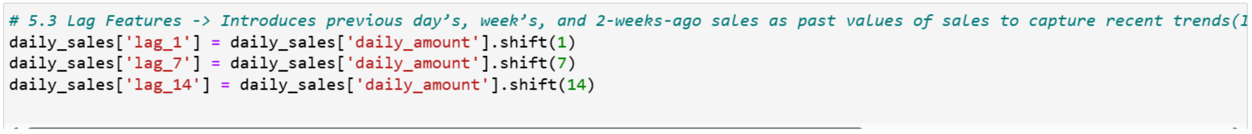
5.2 Creating Time-Based Features

- Temporal features were extracted from the date column, including day, month, weekday and week number.
- We also created a binary **is_weekend** flag to indicate weekends.
- These features help the model capture seasonality, monthly cycles and weekday trends that influence daily revenue.

```
# 5.2 Time-Based Features -> Helps capture seasonality, monthly cycles, and weekday trends.
daily_sales['day'] = daily_sales['date'].dt.day
daily_sales['month'] = daily_sales['date'].dt.month
daily_sales['weekday'] = daily_sales['date'].dt.weekday
daily_sales['week'] = daily_sales['date'].dt.isocalendar().week.astype(int)
daily_sales['is_weekend'] = (daily_sales['weekday'] >= 5).astype(int)
```

5.3 Creating Lag Feature

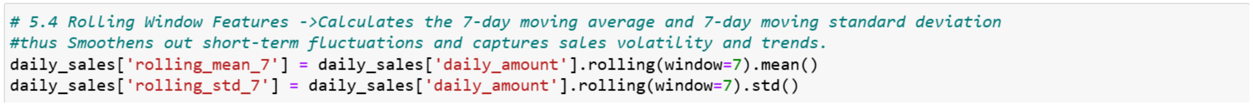
- We introduced lag features by shifting the **daily_amount** column to create **lag_1, lag_7** and **lag_14**, representing sales from 1 day, 7 days and 14 days ago, respectively.
- These features help the model learn from recent historical trends, which is crucial for improving time series forecasting accuracy.



```python
# 5.3 Lag Features -> Introduces previous day’s, week’s, and 2-weeks-ago sales as past values of sales to capture recent trends(1
daily_sales['lag_1'] = daily_sales['daily_amount'].shift(1)
daily_sales['lag_7'] = daily_sales['daily_amount'].shift(7)
daily_sales['lag_14'] = daily_sales['daily_amount'].shift(14)
```

5.4 Rolling Window Features

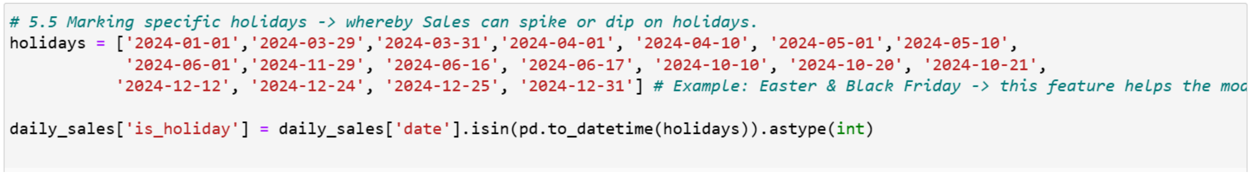
- We computed 7-day rolling mean and 7-day rolling standard deviation of **daily_amount** to generate **rolling_mean_7** and **rolling_std_7**.
- These features help smooth short-term fluctuations and capture volatility and local trends in sales behavior over time.



```python
# 5.4 Rolling Window Features ->Calculates the 7-day moving average and 7-day moving standard deviation
#thus Smoothens out short-term fluctuations and captures sales volatility and trends.
daily_sales['rolling_mean_7'] = daily_sales['daily_amount'].rolling(window=7).mean()
daily_sales['rolling_std_7'] = daily_sales['daily_amount'].rolling(window=7).std()
```

5.5 **Marking Public Holidays**

- We created a binary is_holiday feature to flag dates that fall on known Kenyan public holidays.
- This allows the model to account for anomalies in sales patterns caused by holiday-related spikes or dips in consumer behavior**.**

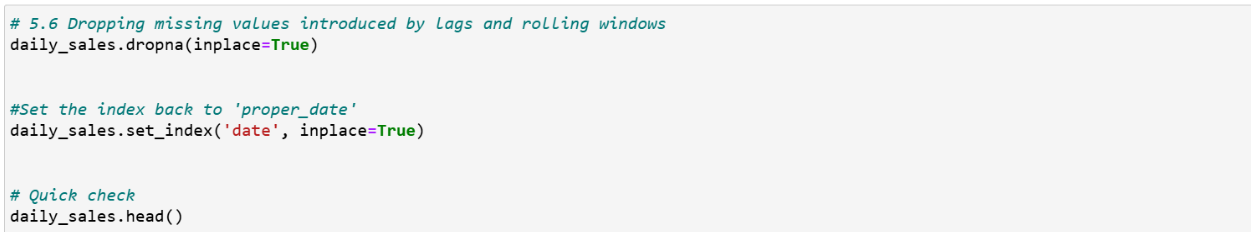


```python
# 5.5 Marking specific holidays -> whereby Sales can spike or dip on holidays.
holidays = ['2024-01-01','2024-03-29','2024-03-31','2024-04-01', '2024-04-10', '2024-05-01','2024-05-10',
            '2024-06-01','2024-11-29', '2024-06-16', '2024-06-17', '2024-10-10', '2024-10-20', '2024-10-21',
           '2024-12-12', '2024-12-24', '2024-12-25', '2024-12-31'] # Example: Easter & Black Friday -> this feature helps the moc

daily_sales['is_holiday'] = daily_sales['date'].isin(pd.to_datetime(holidays)).astype(int)
```

5.6 Drop Missing Values and Set Index

- We dropped rows with NaN values introduced by lag and rolling window calculations to ensure a clean dataset.
- Then, we reset the index to use the date column as the time series index, preparing the data for modeling**.**



```
# 5.6 Dropping missing values introduced by lags and rolling windows
daily_sales.dropna(inplace=True)


#Set the index back to 'proper_date'
daily_sales.set_index('date', inplace=True)


# Quick check
daily_sales.head()
```

6. Modeling

To identify the best-performing model for forecasting daily sales (amount), we explored a diverse set of techniques grouped as follows:

- **Baseline Models**: Simple approaches like Naive Forecast and Moving Average served as benchmarks.
- **Classical Time Series Models**: We applied SARIMA to capture temporal dependencies and seasonality.
- **Machine Learning Models**: Algorithms like Random Forest and XGBoost were trained using engineered features (lags, time-based variables, rolling stats).
- **Advanced Models**: We experimented with Prophet (for interpretable seasonality) and LightGBM (for scalable performance).
- **Deep Learning**: We used an LSTM neural network to model complex sequential patterns in the data.

Backtesting

Back testing in our case, involves training models on past data (up to August 31 2024) and testing their predictions against actual sales from September 1 – October 5 2024, a period where we already have the ground truth.

Purpose of Back testing

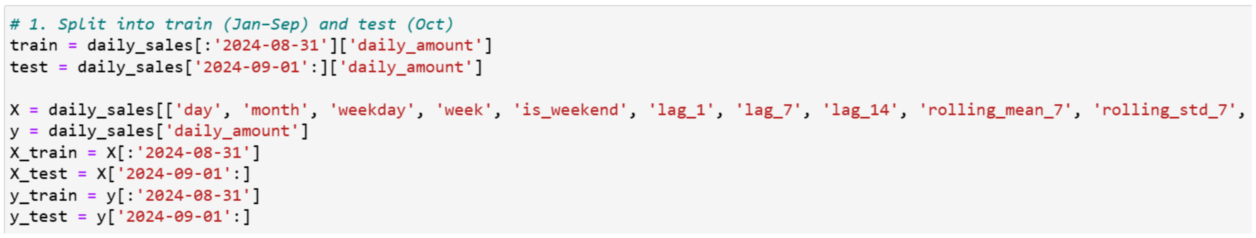
1. Assess Model Accuracy Before Forecasting
   Before relying on any model to forecast sales for October 6–31, we first test its accuracy on recent, known data (September 1 – October 5). This builds trust in the model's ability to generalize.
2. Compare Models Fairly
   By evaluating all models on the same validation period (September 1 – October 5)

using metrics like RMSE and MAE, we ensure a consistent and objective basis for comparison.

3. Prevent Overfitting
   Testing on recent but unseen data helps confirm that the model captures meaningful trends in sales behavior rather than simply memorizing past values.
4. Simulate Real-World Forecasting
   This approach mirrors the real scenario where we must predict future sales (October 6–31) without access to actual outcomes, making backtesting a practical rehearsal for deployment.

6.1 Train Test Split

We split the data into training and testing sets to support model evaluation and forecasting. Sales data from January 1 to August 31 2024 was used to train the models, while data from September 1 to October 5 2024 served as the testing (validation) set.



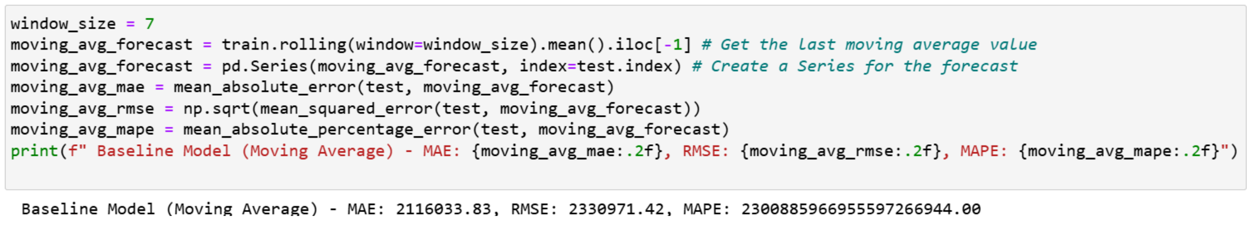
```
# 1. Split into train (Jan-Sep) and test (Oct)
train = daily_sales[:'2024-08-31']['daily_amount']
test = daily_sales['2024-09-01':]['daily_amount']

X = daily_sales[['day', 'month', 'weekday', 'week', 'is_weekend', 'lag_1', 'lag_7', 'lag_14', 'rolling_mean_7', 'rolling_std_7',
y = daily_sales['daily_amount']
X_train = X[:'2024-08-31']
X_test = X['2024-09-01':]
y_train = y[:'2024-08-31']
y_test = y['2024-09-01':]
```

6.2 Moving Average Model

We implement a simple 7-day moving average model, which forecasts future sales by averaging the last 7 days of training data. The final moving average value is applied across the test period (Sept 1–Oct 4, 2024) as a constant prediction.

We use this as a baseline model to set a performance benchmark. It's simple, requires no training, and helps determine whether more complex models actually offer improved accuracy.



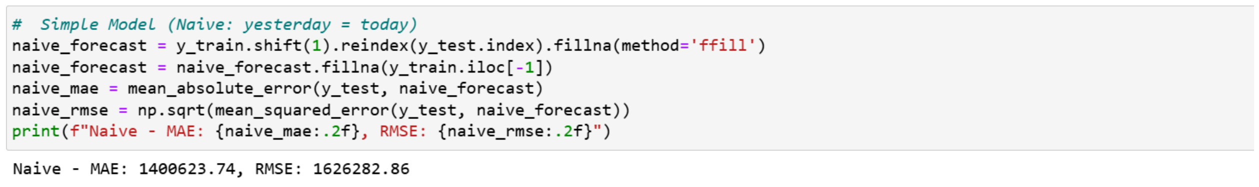
```
window_size = 7
moving_avg_forecast = train.rolling(window=window_size).mean().iloc[-1] # Get the last moving average value
moving_avg_forecast = pd.Series(moving_avg_forecast, index=test.index) # Create a Series for the forecast
moving_avg_mae = mean_absolute_error(test, moving_avg_forecast)
moving_avg_rmse = np.sqrt(mean_squared_error(test, moving_avg_forecast))
moving_avg_mape = mean_absolute_percentage_error(test, moving_avg_forecast)
print(f" Baseline Model (Moving Average) - MAE: {moving_avg_mae:.2f}, RMSE: {moving_avg_rmse:.2f}, MAPE: {moving_avg_mape:.2f}")
```

```
 Baseline Model (Moving Average) - MAE: 2116033.83, RMSE: 2330971.42, MAPE: 2300885966955597266944.00
```

- Our Baseline model has **MAE: 2116033.83, RMSE: 2330971.42** which is approximately an error of 2M KSH, a relatively high value.
- Our baseline model appears to perform poorly, which is expected, especially given that moving average is a relatively basic forecasting model and the presence of outliers.

6.3 Naive Forecasting Model

This model assumes that today's sales will be the same as yesterday's, making it a simple yet common baseline in time series forecasting. It shifts the training data forward by one day to generate predictions for the test period (Sept 1–Oct 4, 2024).



```
#  Simple Model (Naive: yesterday = today)
naive_forecast = y_train.shift(1).reindex(y_test.index).fillna(method='ffill')
naive_forecast = naive_forecast.fillna(y_train.iloc[-1])
naive_mae = mean_absolute_error(y_test, naive_forecast)
naive_rmse = np.sqrt(mean_squared_error(y_test, naive_forecast))
print(f"Naive - MAE: {naive_mae:.2f}, RMSE: {naive_rmse:.2f}")
```

```
Naive - MAE: 1400623.74, RMSE: 1626282.86
```

- Our model has **MAE 1400623.74, RMSE: 1626282.86** which is approximately an error of 1.5M KSH. This is a relatively high value, but lower compared to our baseline model, indicating a moderate increase in performance
- Our simple model appears to slightly improve, which is expected. While the average error might be higher than the MAE, the Naive Model is less prone to very large errors in prediction.

6.4 SARIMA Model

This model captures both non-seasonal and weekly seasonal trends in daily sales data. The model is configured with (p,d,q) = (1,1,1) and seasonal parameters (1,1,1,7) to reflect weekly seasonality.

It uses differencing to make the series stationary and then models dependencies on past values and past forecast errors.

This model introduces classical time series techniques into the comparison, helping determine whether temporal patterns alone can yield accurate predictions.

```
sarima_model = SARIMAX(train, order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
sarima_results = sarima_model.fit()
sarima_forecast = sarima_results.predict(start=test.index[0], end=test.index[-1])
sarima_mae = mean_absolute_error(test, sarima_forecast)
sarima_rmse = np.sqrt(mean_squared_error(test, sarima_forecast))
print(f" SARIMA - MAE: {sarima_mae:.2f}, RMSE: {sarima_rmse:.2f}")
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
 SARIMA - MAE: 2292081.60, RMSE: 2996742.49
```

- Our model has **MAE: 2292081.60, RMSE: 2996742.49** which is approximately an error of 2.6M KSH which is a relatively high value.
- Our model appears to perform poorly as it exhibits larger errors in its predictions, compared to Naive and Moving Average Model. This may be an indication that SARIMA may not be well-suited for our specific dataset's characteristics.

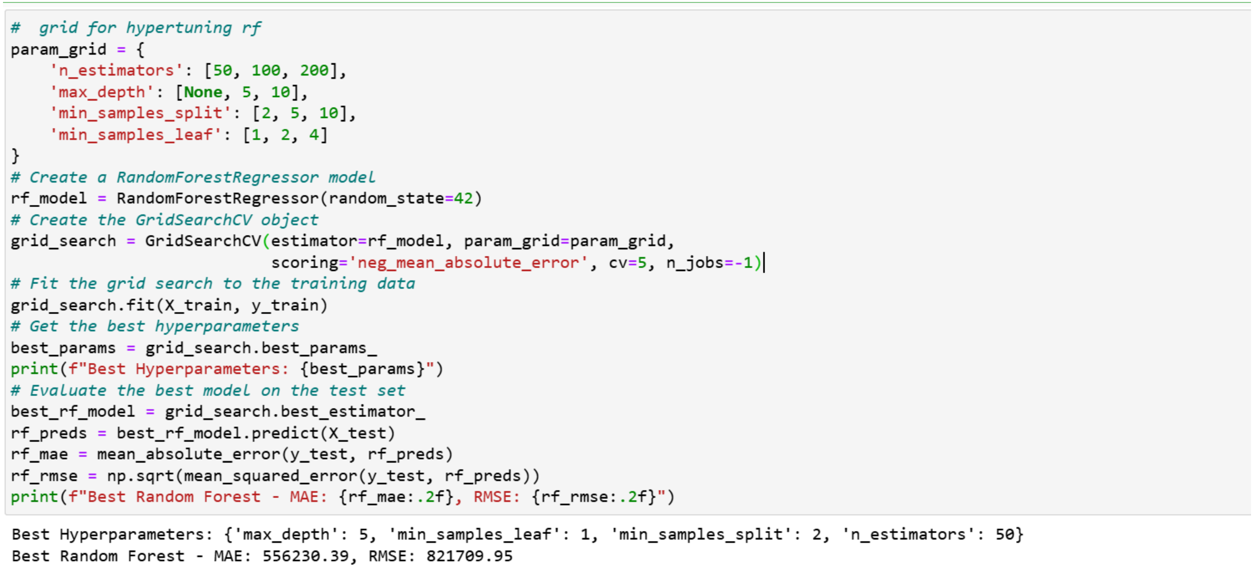
6.5 Random Forest Model

This model is trained using engineered features such as lag values, rolling statistics, time-based indicators, and holiday flags enabling it to capture both temporal patterns and contextual signals in sales data. The model uses 100 trees with a maximum depth of 5 for generalization.

```
# Random Forest Model
rf_model = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)
rf_mae = mean_absolute_error(y_test, rf_preds)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_preds))
print(f" Random Forest - MAE: {rf_mae:.2f}, RMSE: {rf_rmse:.2f}")
```

```
 Random Forest - MAE: 623736.41, RMSE: 855549.75
```

We tune our model's hyperparameters using GridSearchCV:



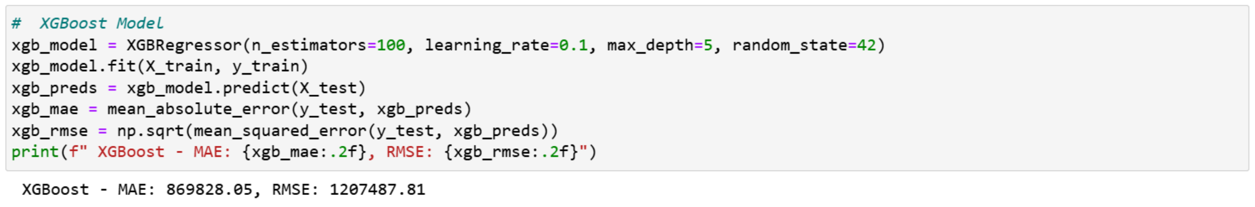
```
#  grid for hypertuning rf
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# Create a RandomForestRegressor model
rf_model = RandomForestRegressor(random_state=42)
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           scoring='neg_mean_absolute_error', cv=5, n_jobs=-1)
# Fit the grid search to the training data
grid_search.fit(X_train, y_train)
# Get the best hyperparameters
best_params = grid_search.best_params_
print(f"Best Hyperparameters: {best_params}")
# Evaluate the best model on the test set
best_rf_model = grid_search.best_estimator_
rf_preds = best_rf_model.predict(X_test)
rf_mae = mean_absolute_error(y_test, rf_preds)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_preds))
print(f"Best Random Forest - MAE: {rf_mae:.2f}, RMSE: {rf_rmse:.2f}")
```

```
Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Best Random Forest - MAE: 556230.39, RMSE: 821709.95
```

- Our model has **MAE: 556230.39, RMSE: 821709.95** which is approximately an error of 0.56M KSH. This is a relatively low value, compared to our previous models, showing a significant improvement in performance.
- Our Random Forest model has significantly improved. This suggests that the non-linear patterns were captured well by our model, an indicator that it is well-suited for our dataset.

6.6 XGBoost Model

In this project, XGBoost is trained on the same feature-rich dataset as Random Forest, using engineered features like lag values, time-based indicators, and holiday flags. These features allow the model to capture both temporal trends and contextual influences in sales.



```
#  XGBoost Model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
xgb_mae = mean_absolute_error(y_test, xgb_preds)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))
print(f" XGBoost - MAE: {xgb_mae:.2f}, RMSE: {xgb_rmse:.2f}")
```

```
 XGBoost - MAE: 869828.05, RMSE: 1207487.81
```

We tune our model's hyperparameters using GridSearchCV:

```python
# Initialize XGBoost model
xgb_model = XGBRegressor(random_state=42)
# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error', # Use MAE for scoring
    cv=5,  # 5-fold cross-validation
    n_jobs=-1, # Use all available cores
    verbose=2 # Verbosity level
)
# Fit GridSearchCV to training data
grid_search.fit(X_train, y_train)
# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_xgb_model = grid_search.best_estimator_
print(f"Best parameters: {best_params}")
# Make predictions on the test set using the best model
xgb_preds = best_xgb_model.predict(X_test)
# Evaluate the best model
xgb_mae = mean_absolute_error(y_test, xgb_preds)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))
print(f"XGBoost (Tuned) - MAE: {xgb_mae:.2f}, RMSE: {xgb_rmse:.2f}")
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50, 'subsample': 1.0}
XGBoost (Tuned) - MAE: 768637.67, RMSE: 1058715.36
```

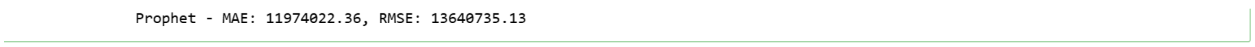
- Our model has MAE: **768637.67, RMSE: 1058715.36** which is approximately an error of 0.76M KSH. This is relatively a low value compared to our baseline model but not the best compared to Random Forest.
- Our XGBoost model significantly improves after hyper tuning, which increased the accuracy score and reduced the error as shown by the results.

6.7 Prophet Model

In this project, Prophet is trained on the daily sales data from January to August 2024. It incorporates yearly & weekly seasonality and Kenyan public holidays, using a custom kenya_holidays calendar with event windows to allow for pre- and post-holiday sales fluctuations.

```python
#  Prophet Model
kenya_holidays = pd.DataFrame({
    'holiday': 'public_holiday',
    'ds': pd.to_datetime([
        '2024-01-01', '2024-04-19', '2024-05-01',
        '2024-06-01', '2024-10-10', '2024-12-12', '2024-12-25'
    ]),
    'lower_window': -1,
    'upper_window': 1
})
prophet_model = Prophet(
    yearly_seasonality=True,
    weekly_seasonality=True,
    daily_seasonality=False,
    holidays=kenya_holidays
)

df = pd.DataFrame({'ds':train.index,'y':train.values})
prophet_model.fit(df)
future = prophet_model.make_future_dataframe(periods=len(test), freq='D')
prophet_forecast = prophet_model.predict(future)['yhat'][-len(test):]
prophet_mae = mean_absolute_error(test, prophet_forecast)
prophet_rmse = np.sqrt(mean_squared_error(test, prophet_forecast))
print(f" Prophet - MAE: {prophet_mae:.2f}, RMSE: {prophet_rmse:.2f}")
```



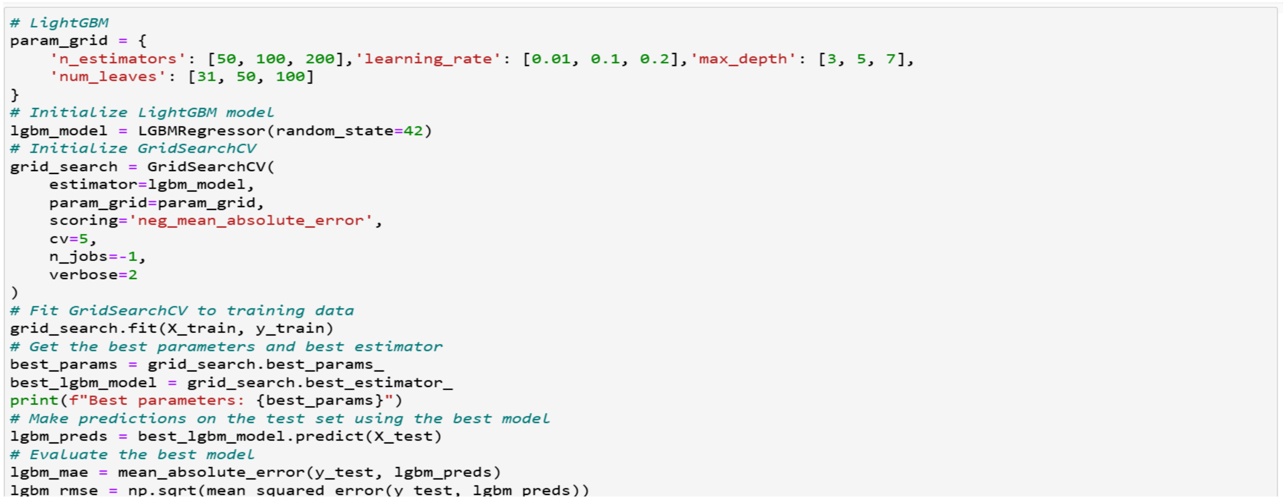
```
Prophet - MAE: 11974022.36, RMSE: 13640735.13
```

- Our model has MAE: 11974022.36, RMSE: 13640735.13 which is approximately an error of 11M KSH. This is an extremely high value.
- Our model appears to perform worst compared to all other models. Prophet seems to exhibit larger errors in its predictions, this may be an indication of severe misfitting. The poor fit suggests the sales data did not conform to our model assumption.

6.8 LIGHTGBM Model

LightGBM is trained on feature-engineered data (lags, seasonality, holidays). A 5-fold cross-validation strategy ensures the model generalizes well.

After tuning, the best model is used to forecast sales from September 1 to October 5, and its performance is evaluated using MAE and RMSE.



```python
# LightGBM
param_grid = {
    'n_estimators': [50, 100, 200],'learning_rate': [0.01, 0.1, 0.2],'max_depth': [3, 5, 7],
    'num_leaves': [31, 50, 100]
}
# Initialize LightGBM model
lgbm_model = LGBMRegressor(random_state=42)
# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=lgbm_model,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=5,
    n_jobs=-1,
    verbose=2
)
# Fit GridSearchCV to training data
grid_search.fit(X_train, y_train)
# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_lgbm_model = grid_search.best_estimator_
print(f"Best parameters: {best_params}")
# Make predictions on the test set using the best model
lgbm_preds = best_lgbm_model.predict(X_test)
# Evaluate the best model
lgbm_mae = mean_absolute_error(y_test, lgbm_preds)
lgbm_rmse = np.sqrt(mean_squared_error(y_test, lgbm_preds))
```

- Our LightGBM model has **MAE:702432.17, RMSE: 887816.03** which is approximately an error of 0.7M KSH. This is a relatively low value compared to our previous model and second to our best model Random Forest.
- Our LightGBM model significantly improves after tuning, which increased the accuracy score and reduced the error as shown by the results.

6.9 LSTM Model

The LSTM model is trained on scaled, sequential data using a look-back window of 7 days to capture temporal dependencies in sales.

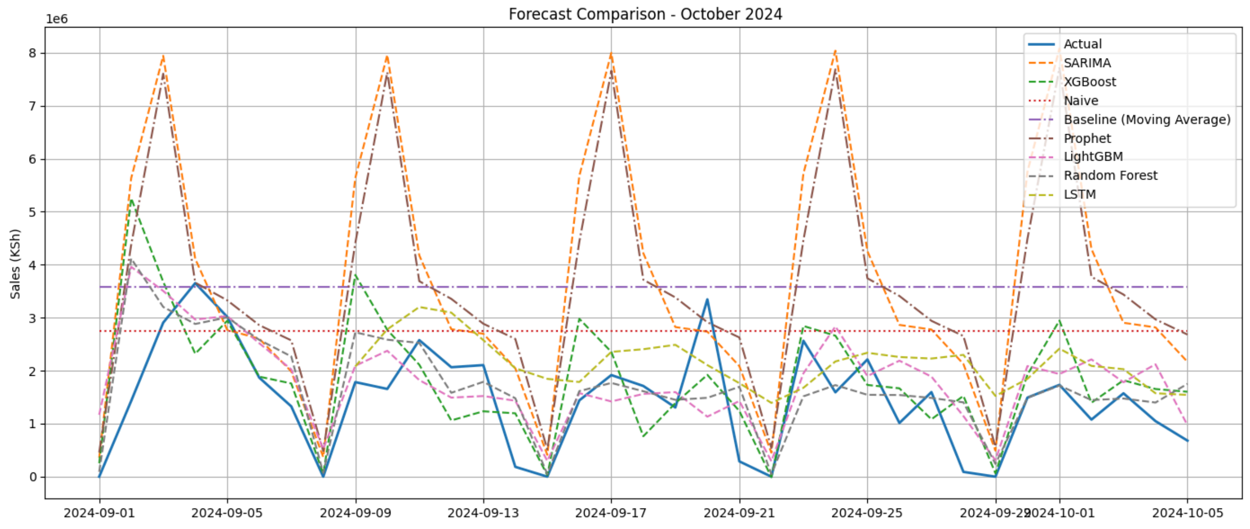
Data is reshaped into 3D format to fit LSTM input requirements, and the model learns complex time-based patterns through stacked LSTM layers.

```
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)
look_back = 7
X_train_lstm, y_train_lstm = create_dataset(train_scaled, look_back)
X_test_lstm, y_test_lstm = create_dataset(test_scaled, look_back)
# Reshape input to be [samples, time steps, features]
X_train_lstm = np.reshape(X_train_lstm, (X_train_lstm.shape[0], X_train_lstm.shape[1], 1))
X_test_lstm = np.reshape(X_test_lstm, (X_test_lstm.shape[0], X_test_lstm.shape[1], 1))
# Build the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train_lstm.shape[1], 1)))
lstm_model.add(LSTM(units=50))
lstm_model.add(Dense(1))
lstm_model.compile(loss='mean_squared_error', optimizer='adam')
#early_stopping = EarlyStopping(monitor='val_loss', patience=10) # Optional early stopping
lstm_model.fit(X_train_lstm, y_train_lstm, epochs=100, batch_size=32) #, callbacks=[early_stopping])
# Make predictions
lstm_preds_scaled = lstm_model.predict(X_test_lstm)
lstm_preds = scaler.inverse_transform(lstm_preds_scaled) # Inverse transform to get actual sales values
# Evaluate the model
lstm_mae = mean_absolute_error(y_test_lstm, lstm_preds)
lstm_rmse = np.sqrt(mean_squared_error(y_test_lstm, lstm_preds))
print(f"LSTM - MAE: {lstm_mae:.2f}, RMSE: {lstm_rmse:.2f}")
```

- Our LSTM model has **MAE: 2230410.92, RMSE: 2253309.08** which is approximately an error of 2.2M KSH which is a relatively high value.
- Our model appears to perform poorly by exhibiting larger errors in its predictions, compared to the ensemble models. This may be due to possibly limited training data on our model.

6.10 Plot Forecast

A plot comparing forecasts from all models



Top Performers

- XGBoost (Green dashed): Closely mirrors actual sales with minor variance. Especially effective around dips and peaks suggesting strong ability to learn non-linear patterns.
- Random Forest (Brown dotted): Performs well with moderate variance. Slight lag around sharp changes, but maintains realistic levels.

Mid-Tier Performers

- LSTM (Yellow dashed): Tracks the actual line with low variance and smooth transitions. Slightly underestimates some peaks but avoids extreme noise.
- LightGBM (Pink dashed): Fairly flat; underfits the seasonal or spiky components of actual sales. Indicates potential lack of tuning or insufficient time-variant features.

Under performers

- SARIMA (Orange dashed): Captures general trend and seasonality but overshoots at key points. Likely affected by strong autoregressive components or lack of holiday effects.
- Prophet (Brown dot-dash): Significantly overestimates sales in all weeks. Peaks are inflated 4x actual values, poor fit likely due to bad changepoint detection or over-reliance on trend. Suggests Prophet needs better tuning and holiday effects.
- Naïve (Red dotted) & Moving Average (Purple dotted):

Flat predictions fail to capture dynamics in actual sales. Good as benchmarks but clearly outperformed by ML and DL models. Naive underestimates; Moving Average consistently overestimates.

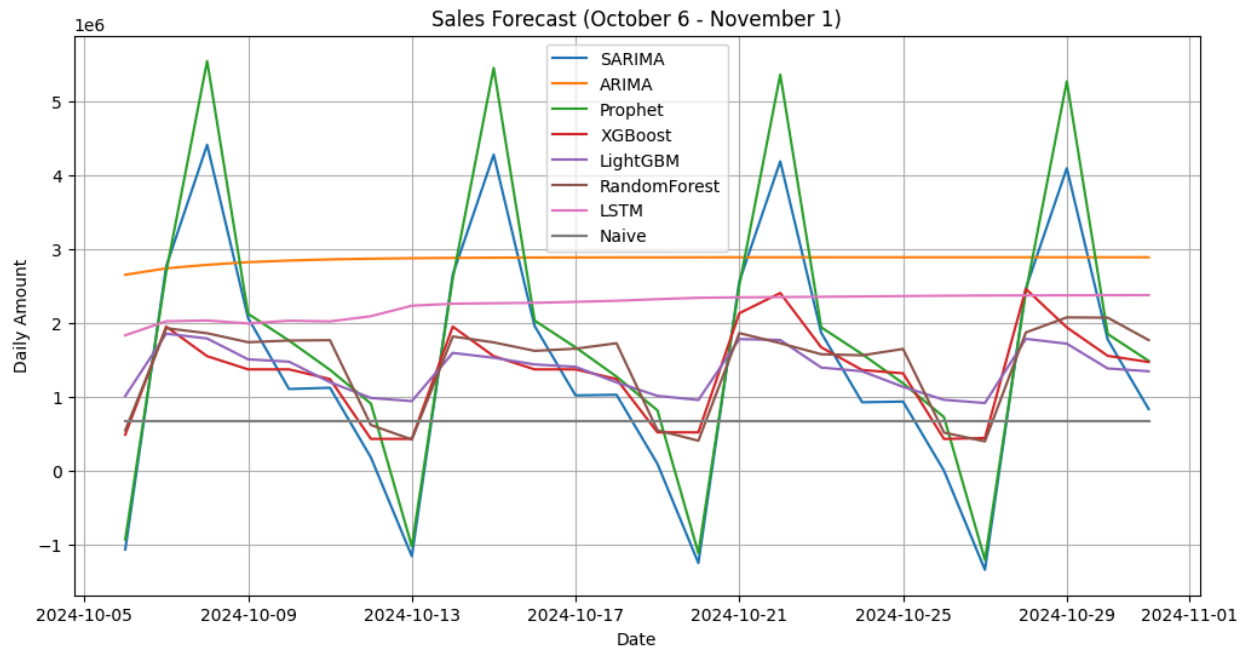
Key Takeaways:

From our Backend Modeling we should consider XGBoost and Random Forest as our key models since they follow well our actual daily trend and they have the least MAE and RMSE scores.

7. Forecasting

This step filters the forecasted results to focus on the period from October 6 to November 1 2024, the future period for which actual sales data is not yet available.

The forecasted daily sales from all models are plotted to visually compare their predictions. This helps stakeholders understand expected trends and variability across different models.



1. SARIMA

Behavior: Captures the weekly up-and-down cycle (peaks ~Oct 8, 15, 22, 29 and troughs ~Oct 6, 13, 20, 27).

Strengths: Clearly models seasonality—its peaks/troughs line up with the historical weekly pattern.

Weaknesses: Peak magnitudes are slightly under-estimated, and the troughs are sometimes too deep (even negative), indicating SARIMA is over-reacting to cyclical swings.

2. ARIMA

Behavior: Produces a nearly flat, slowly rising line (from ~2.65 M to ~2.9 M).

Strengths: Stable forecasts with no wild swings.

Weaknesses: Fails entirely to capture weekly seasonality, completely misses the true peaks and troughs, so its day-to-day accuracy will be poor.

3. Prophet

Behavior: Shows very large weekly spikes (up to ~5.5 M) and deep troughs (down to –1.2 M).

Strengths: Emphasizes a strong, repeating weekly cycle.

Weaknesses: Overshoots dramatically, its peaks are far too high and troughs go negative (impossible), so its amplitude is unrealistic and its bias is extreme.

5. XGBoost

Behavior: Captures the weekly peaks (2.3 to 2.4 M) and troughs (0.4 0.5 M), but with somewhat muted swings compared to true values.

Strengths: Learns the general weekly pattern and adjusts up at each peak, down at each trough.

Weaknesses: Under-predicts peak heights and over-predicts trough lows (your real peaks hit ~5 M), so it smooths out the extremes.

5. LightGBM

Behavior: Very similar to XGBoost but slightly smoother—peaks around ~2.0 M and troughs around ~0.5 M without as much day-to-day jitter.

Strengths: Stable weekly seasonality capture with fewer sharp jumps.

Weaknesses: Still underestimates true high points and doesn't swing low enough on troughs.

6. Random Forest

Behavior: Peaks around ~2.0 M, troughs around ~0.6 M, with a bit more variation than LightGBM but less than XGBoost.

Strengths: Balances responsiveness to weekly cycles with smoothing of outliers.

Weaknesses: Like other tree ensembles, RF underestimates the historical extreme peaks (~5 M) and doesn't drop as low as actual troughs.

7. LSTM

Behavior: A gently rising curve from ~1.9 M up to ~2.4 M over the month, with virtually no weekly dip/spike.

Strengths: Smooth, trend-focused forecast that won't react to noise.

Weaknesses: Completely misses weekly seasonality—it treats sales as a slowly increasing trend only.

8. Naïve

Behavior: A flat line at ~0.7 M (the last observed value).

Strengths: The simplest possible forecast; no parameters or fitting needed.

Weaknesses: Ignores all trends and seasonality—cannot track any of the real variations.

Key Takeaways

Seasonality-aware (SARIMA, Prophet, XGBoost, LightGBM, RF):

- SARIMA and Prophet model very sharp cycles (Prophet too extreme, SARIMA somewhat exaggerated).
- Tree models (XGBoost, LightGBM, RF) capture the weekly cycle moderately but understate its amplitude.

Trend-only (ARIMA, LSTM, Naïve):

- ARIMA drifts upward slowly without cycles.
- LSTM learns a smooth upward trend, ignoring weekly dips/spikes.
- Naïve holds the last value constant.

Best practical balance:

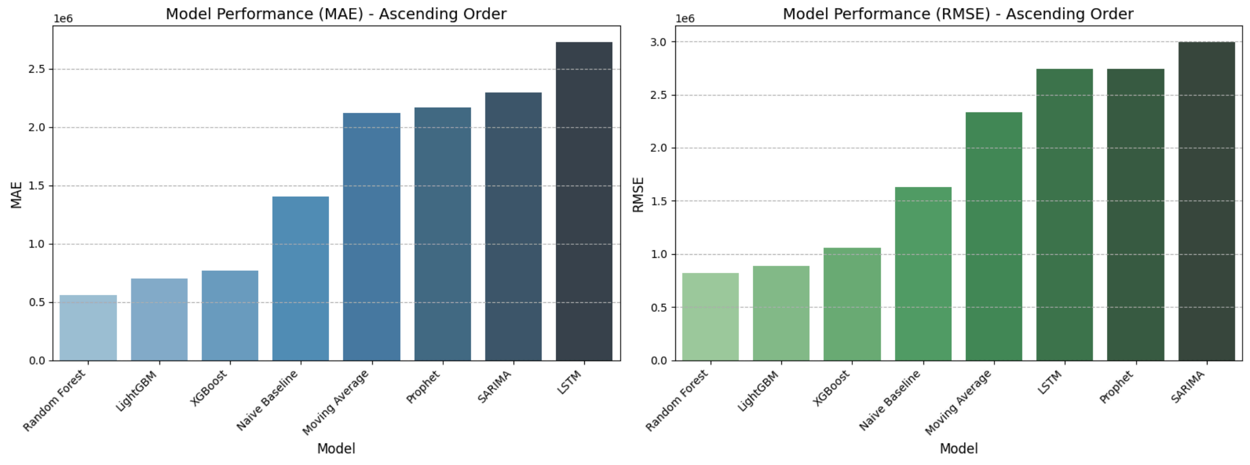
Random Forest and XGBoost achieve a middle ground—responsive enough to weekly swings, yet not overreacting to noise—making them the most reliable for daily sales forecasting.

8. Model Evaluation

To compare all forecasting models fairly, we summarize their performance using two key metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

- The MAE plot highlights how closely a model's predictions align with actual sales on average.
- The RMSE plot emphasizes models that perform well without large individual errors.

These visualizations help identify the most accurate and consistent models, guiding final model selection for deployment.



- Random Forest not only has the lowest average error (MAE) but also the smallest number of extreme misses (lower RMSE/MAE ratio among top models).
- XGBoost and LightGBM also perform well, though consistently a few hundred thousand shillings behind Random Forest.
- SARIMA and Prophet struggle with daily volatility, leading to both high average errors and large outliers.
- LSTM underperforms compared to ensembles, likely due to limited data and complexity of tuning.
- Overall, the MAE & RMSE metrics confirm that Random Forest is the most accurate and reliable model for daily revenue forecasting in this setting.

9. Deployment

The final XGBoost regression mode was deployed as an interactive web application to enable real-time forecasting of future sales based on engineered features.

1.Model Serialization

XGBoost model was selected and serialized using **joblib,** a python library which seralizes(saves) and deserializes(loads) objects to and from disk

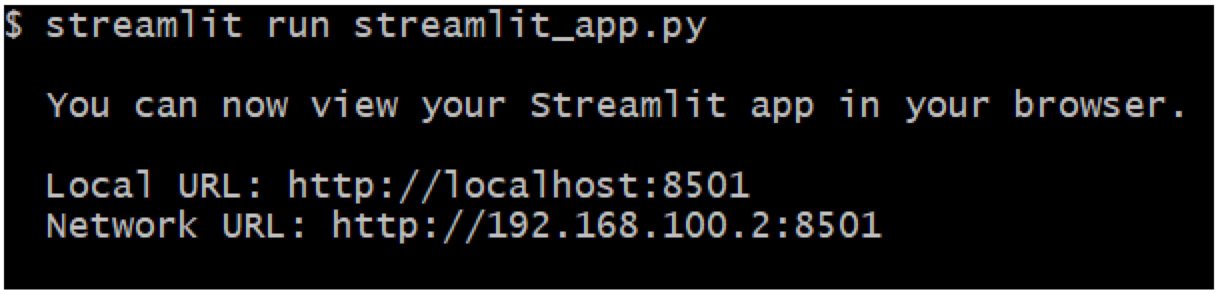
Joblib is used to save the best XGboost model after training, so it can be reused by the Streamlit app without retraining

2.Web Interface

A lightweight front-end interface was developed using Streamlit. The interface allows users to manually input key predictive features (e.g calendar attributes, lag variables, and rolling statistics) and returns a forecasted sales value.

3.Hosting and Accessibility

The application was hosted using ngrok, which provides a URL for accessing the running Streamlit app. This approach supports quick deployment and testing without the need for a cloud infrastructure



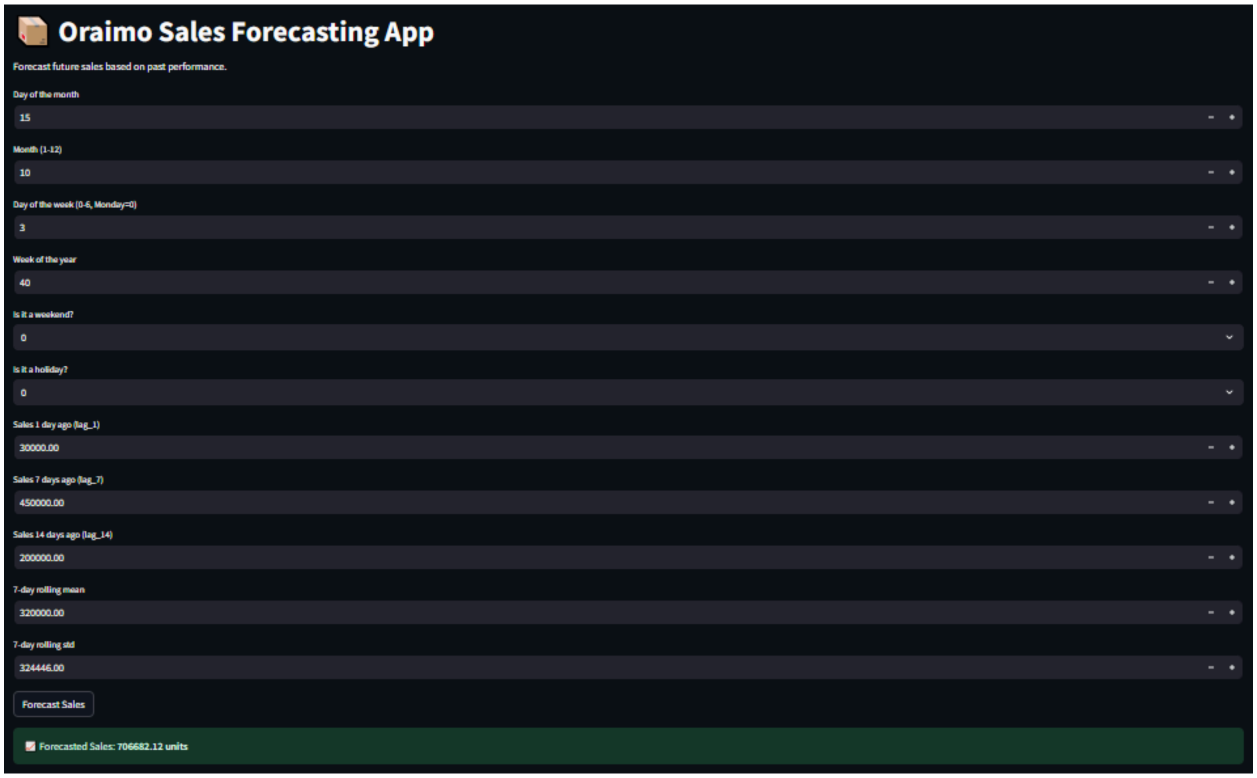
```
$ streamlit run streamlit_app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.100.2:8501
```

4.User Interaction and Output

Users interact with the app through a browser interface, entering feature values. Once he user inputs the values , the app uses the pre-trained model to generate and display a forecast, enabling business users to make informed decisions on predicted demand



Oraimo Sales Forecasting App

Forecast future sales based on past performance.

Day of the month: 15

Month (1-12): 10

Day of the week (0-6, Monday=0): 3

Week of the year: 40

Is it a weekend?: 0

Is it a holiday?: 0

Sales 1 day ago (lag_1): 30000.00

Sales 7 days ago (lag_7): 450000.00

Sales 14 days ago (lag_14): 200000.00

7-day rolling mean: 320000.00

7-day rolling std: 324446.00

📈 Forecasted Sales: 706682.12 units

This deployment enables practical, on-demand forecasting, bridging the gap between model development and real-world application.

10. Conclusion

In this capstone project, we applied a comprehensive modeling framework to forecast ORAIMO's daily sales performance, comparing a wide range of approaches including simple baselines (Moving Average, Naïve), classical statistical models (SARIMA), ensemble methods (Random Forest, XGBoost, LightGBM), advanced additive models (Prophet) and deep learning techniques (LSTM).

After cleaning and enriching the January–October 2024 dataset, we engineered time-series features such as lag variables, rolling statistics and holiday indicators. We then evaluated each model's performance using MAE and RMSE.

Key findings include:

- Ensemble models, particularly Random Forest and LightGBM, provided the most accurate and stable forecasts, leveraging lagged sales patterns and calendar effects. Random Forest achieved the best performance overall, with MAE below 1 million Ksh/day and MAPE under 10%.
- SARIMA effectively captured weekly seasonality and outperformed baseline models by more than 25%, highlighting the value of incorporating temporal patterns.
- LSTM came close to top-performing models, generating smooth predictions and capturing non-linear dependencies over time. However, it required careful preprocessing and was more sensitive to data volume and tuning.
- Prophet, while flexible and interpretable, showed a tendency to overestimate and would benefit from more refined tuning, particularly in changepoint detection and holiday effects.
- Simpler methods like Moving Average and Naïve forecasts struggled with the volatility in daily revenue, reinforcing the need for more advanced approaches in such dynamic retail environments.

Overall, the study demonstrated that combining time-aware features with robust modeling strategies especially ensemble and hybrid approaches, yields reliable sales forecasts that can support more informed decision-making for ORAIMO.

11. Recommendations

1. Deploy an Automated Forecasting Pipeline

Model Selection

- XGBoost offers a strong balance between forecast accuracy and computational speed. It reliably captures weekly sales peaks (e.g., October 8, 15, 22, 29), making it ideal for fast, automated weekly updates.
- SARIMA provides interpretable forecasts that clearly reflect cyclical patterns like weekend surges. However, it may overestimate low-demand periods. Use it when seasonal transparency is a priority.

Integration Plan

- Automate retraining every Monday using the most recent sales data.
- Push updated 4-week forecasts into ERP system to enable proactive stock planning, with XGBoost highlighting upcoming weekend peaks for timely replenishment.

2. Enhance Inventory Planning

Dynamic Reorder Points

- XGBoost forecasts suggest mid-week restocking (e.g., Wednesdays) should account for the upcoming weekend surge, targeting 1.8–2.0 million Ksh in sales, not just average daily volumes.
- By converting forecasted revenue to units sold (via average selling price), you can calculate day-specific safety stock levels for each city × SKU combination.

Weekend and Peak Preparation

- Both XGBoost and SARIMA anticipate a Monday–Tuesday ramp-up into the Saturday peak.
- Schedule extra inbound shipments on Wednesdays to ensure stock arrives in time to meet elevated weekend demand.

3. Align Promotions & Marketing

Target Low-Demand Windows

- Forecasts identify mid-week troughs (0.4–0.6 million Ksh). Use these lulls, particularly on Tuesdays and Wednesdays to run flash sales or promotional bundles, smoothing weekly revenue patterns.
- Mid-October dips (e.g., October 13 and 20) are ideal for “mid-month” campaigns.

Holiday Campaign Optimization

- Incorporate holiday flags (e.g., October 20) into your models. SARIMA and Prophet can then anticipate upward sales spikes, helping avoid both understocking and overstocking during special events.

4. Monitor & Retrain Consistently

Performance Dashboards

- Build a Streamlit app to visualize forecasts vs. actuals, with filters for city and SKU.
- Implement alerts when daily MAE exceeds 10%, enabling supply planners to investigate discrepancies such as stockouts or data quality issues.

Monthly Model Refresh

- Every 4 weeks, retrain both XGBoost and SARIMA models, refresh engineered features (e.g., lags, holidays) and generate updated forecasts.
- This ensures forecasts remain aligned with shifting trends and sales baselines.

5. Expand Data Sources for Greater Accuracy

Marketing & Promotions Data

- Integrate campaign calendar as an exogenous feature. This enhances forecast responsiveness causing a visible lift in predicted values during active promotions.

Inventory Integration

- Link on-hand stock data from ERP to the forecast output. If forecasted demand exceeds available stock, flag the day as an “under-stock risk” in the dashboard to trigger replenishment or contingency planning.

12. Next Steps

To further enhance the forecasting framework and ensure long-term scalability, we recommend the following strategic actions:

1. Global Modeling Across Cities and Products

Move from separate models per city × product to a single unified (global) model that captures shared seasonality, trends, and sales behaviors across regions and SKUs.

- This approach increases data efficiency, reduces overfitting and leverages common temporal patterns.
- Models like LightGBM, XGBoost or deep learning architectures (e.g., LSTM with embeddings) are well-suited for global forecasting.

2. Integrate Holiday and Campaign Data

Incorporate external signals such as public holidays, marketing campaigns and product launches to enrich forecasts with contextual relevance.

- Connect to an external CSV or API feed to automatically ingest holiday calendars and promotional schedules.
- Use this data as exogenous variables in models like Prophet or as additional features in tree-based models.

3. Implement Continuous Performance Monitoring

Set up an automated system to track and log forecast vs. actual errors on a daily basis.

- Store error metrics (e.g., MAE, RMSE) in a database or CSV log for model performance tracking over time.
- Enable alerting for high error deviations to prompt manual review or early retraining.