Table of Contents

[**1.0** **Introduction** 3](#_Toc185005134)

[**1.1** **Background** 3](#_Toc185005135)

[**1.2 Business Problem** 3](#_Toc185005136)

[**1.3 Business Objectives** 3](#_Toc185005137)

[**2.0** **Data Exploration** 4](#_Toc185005138)

[**2.1 Dataset Overview** 4](#_Toc185005139)

[**2.2 Data Preprocessing** 6](#_Toc185005140)

[**2.2.1 Data Cleaning and Data Type Optimization** 6](#_Toc185005141)

[**2.2.2 Handling Missing Values and Duplicate Rows.** 7](#_Toc185005142)

[**2.2.3 Exploratory Data Analysis (EDA) for Preprocessing** 8](#_Toc185005143)

[**2.2.4 Handling Outliers** 16](#_Toc185005144)

[**2.2.5 Feature Engineering.** 19](#_Toc185005145)

[**2.2.6 Feature Selection** 19](#_Toc185005146)

[**2.2.7 Encoding** 20](#_Toc185005147)

[**3.0** **Visualizing Relationships** 21](#_Toc185005148)

[**3.1 Correlation Heatmap of Numerical Features After Encoding** 21](#_Toc185005149)

[**3.2 Regional Sales Performance** 22](#_Toc185005150)

[**3.3 Category and Segment Sales Performance** 22](#_Toc185005151)

[**3.4 Sales Trend Analysis** 23](#_Toc185005152)

[**3.5 Regional Sales Trends and Discount Effectiveness Analysis.** 24](#_Toc185005153)

[**4.0** **Machine Learning Approach** 25](#_Toc185005154)

[**4.1. Random Forest** 25](#_Toc185005155)

[**4.2 XGBoost** 27](#_Toc185005156)

[**4.3 Model Comparison** 30](#_Toc185005157)

[**4.4 Results interpretation** 31](#_Toc185005158)

[**4.5 Feature Importance** 32](#_Toc185005159)

[**5.0** **Business Insights** 33](#_Toc185005160)

[**5.1 Actionable Recommendations** 33](#_Toc185005161)

[**5.2** **Limitations and Future Work** 34](#_Toc185005162)

[**5.3** **Future Work** 34](#_Toc185005163)

[**6.0** **Reference** 35](#_Toc185005164)

[**7.0** **Appendix** 37](#_Toc185005165)

[**7.1 Dataset overview** 37](#_Toc185005166)

[**7.2 Data preprocessing** 37](#_Toc185005167)

[**7.3 Exploratory Data Analysis (EDA) for Preprocessing** 39](#_Toc185005168)

[**7.4 Handling outliers** 48](#_Toc185005169)

[**7.5 Feature Engineering** 53](#_Toc185005170)

[**7.6 Feature Selection** 53](#_Toc185005171)

[**7.7 Encoding** 54](#_Toc185005172)

[**7.8 Visualizing Relationships** 54](#_Toc185005173)

# **Introduction**

## **Background**

The retail industry has been changing rapidly due to new technologies, changing customer habits, and growing competition. To remain competitive, businesses must rely on data to make smarter decisions. Sales forecasting is a critical part of this as it helps retailers manage inventories, meet customers demand effectively, and plan marketing campaigns. Without accurate forecasts, businesses risk overstocking, stockouts, or inefficient promotions, all which impact profitability, leading to profit loss (Ganguly & Mukherjee, 2024).

Today, technologies like Machine Learning (ML) are helping retailers to solve these challenges. These tools analyse large amounts of data to predict sales, understand customer behaviour, and identify what drives revenue. For example, ML can reveal how different factors like discounts or product categories impact sales. Additionally, real-time analytics provided by ML allows companies to respond promptly to market changes, improving profitability and decision-making (Snyder et al., 2020). With such tools, retailers can make better decisions, stay competitive, and improve profitability.

However, many retailers still find it hard to forecast sales accurately due to complex factors such as customer demographics, product diversity, and regional trends. Without understanding these, businesses may struggle with poor stock management or unprofitable discounts. ML techniques offer a solution, helping businesses uncover hidden patterns and create predictive models that improve operational efficiency (Makridakis et al., 2018).

## **1.2 Business Problem**

The main problem that will address in this project is how to predict sales performance to drive better business decision and boost profitability. Many retail businesses struggle with forecasting sales as there are many factors that affecting it. Inaccurate predictions can lead to overstocking, missed sales opportunities, and ineffective discounts which lead to business risk losing revenue This project utilizes the Superstore dataset to develop ML models for predicting sales and providing actionable insights.

## **1.3 Business Objectives**

The primary goal of this project is to leverage machine learning to improve sales predictions and provide practical insights that help businesses make better decision. To guide this research, our business objectives are as follows:

1. Prediction of Total Sales

Develop a machine learning model to forecast total sales. This will help businesses to allocate resources more effectively such as improve inventory management and prepare further demand.

1. Identify Key Sales Drivers:

Analyse factors like product categories, regional performance, customer segments, and discounts to understand their impact on sales performance. By identifying these drivers will help businesses to focus on high impact areas.

1. Optimize Discount Strategies:

Examine the relationship between discounts and sales and recommend strategies to maximise revenue.

By achieving these objectives, businesses can better predict trends, manage resources, and implement effective pricing. Understanding sales drivers and optimizing discounts will contribute to increased revenue and improved customer satisfaction (Hyndman & Athanasopoulos, 2021).

# **Data Exploration**

## **2.1 Dataset Overview**

The Superstore Dataset from Kaggle (Kaggle link; [Superstore Dataset](https://www.kaggle.com/datasets/vivek468/superstore-dataset-final)) is a comprehensive dataset. As shown in the figure 1 the dataset contains transactional data related to a superstore, where this dataset provides insights into sales, customer behaviours, and regional performance across multiple product categories. The dataset contains 9,994 entries and 21 columns which contains a mixture of categorical and numerical variables enabling a broader analysis.

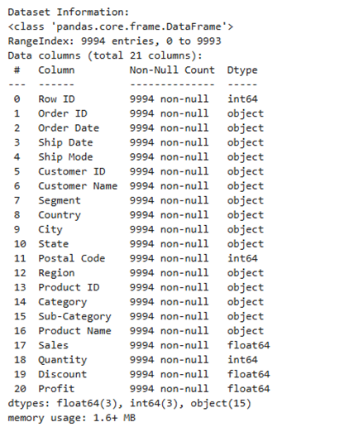


Figure 1: Overview of superstore dataset attributes

The summary of the key attributes is described in the figure 2:

|  |  |
| --- | --- |
| Variable | Description |
| Row ID | Unique identifier for each record in the dataset. |
| Order ID | Unique identifier for each order placed in the superstore. |
| Order Date | The date when the order was placed. |
| Ship Date | The date when the order was shipped. |
| Ship Mode | Mode of shipping used for the order (e.g., Standard Class, Second Class, etc.). |
| Customer ID | Unique identifier for each customer. |
| Customer Name | Full name of the customer. |
| Segment | The customer segment (e.g., Consumer, Corporate, Home Office). |
| Country | Country where the customer resides (e.g., United States). |
| City | City where the customer resides. |
| State | State where the customer resides. |
| Postal Code | Postal code of the customer's location. |
| Region | Region where the customer resides (e.g., West, East, Central, South). |
| Product ID | Unique identifier for each product. |
| Category | Broad category of the product (e.g., Office Supplies, Furniture, Technology). |
| Sub-Category | Specific sub-category of the product (e.g., Binders, Chairs, Phones). |
| Product Name | Full name or description of the product. |
| Sales | Total sales amount for the product in the order. |
| Quantity | Quantity of the product purchased in the order. |
| Discount | Discount applied to the product in the order (in percentage). |
| Profit | Total profit generated for the product in the order. |

Figure 2: Superstore dataset key attributes

## **2.2 Data Preprocessing**

Preprocessing ensures the dataset is clean, consistent, and ready for analysis, there it makes an important step data analysis. For the Superstore dataset, the following preprocessing steps were performed:

### **2.2.1 Data Cleaning and Data Type Optimization**

#### **2.2.1.1 Handling Columns with Unique Values**

It is important to handle and eliminate the unimportant and distractive information in the dataset, that could mislead the actual results required. Therefore, the number of unique values in the dataset was identified. From Figure 3 it could be seen that, only country variable has 1 unique value. This column was then removed as it does not add much meaning to the analysis (figure 4).

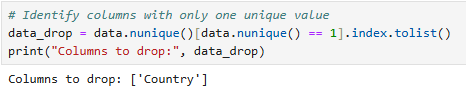


Figure 3: Identifying unique values



Figure 4: Number of columns before and After Removing the columns with unique values

#### **2.2.1.2 Handling NULL and NaN Values**

Next step is to identify any “NULL or NaN” values in the dataset using data\_dropped\_cols = data\_dropped\_cols.dropna(axis=1, how='all') command. The outcome shown in figure 5, indicates that there were no columns lost during this process.

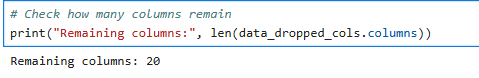


Figure 5: Identifying NULL and NaN types

#### **2.2.1.3 Cleaning and Standardizing Column Names**

It is important that the column's in the dataset follows the same and correct format for better analysis and modelling tasks. Trailing spaces in columns was first identified and handled, following this removing any unnecessary spaces and underscores within the columns was handled. Next was to convert the columns names to lower case, to ensure consistency and to ensure that wrong result outcomes due to case sensitivity at later stages. Lastly any duplicate columns were handled to ensure that all the columns in the dataset were unique.

#### **2.2.1.4 Converting Data Types**

After successful column names cleaning, inconsistencies in data types were addressed, for instance order date and ship date saved as object, postal\_code which is saved as numeric which does not hold numerical significance and categorical columns named as objects. These identified issues were address firstly by converting order date and ship date to datetime format, then converting postal\_code to string to avoid any data leading to zero and lastly converting object columns to categories to ensure further preprocessing and model handling is done easily. Figure 6 indicates the corrected and cleaned dataset structure.

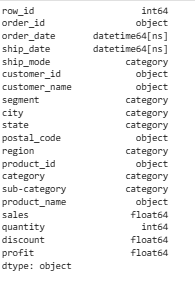


Figure 6: Cleaned Dataset Structure

### **2.2.2 Handling Missing Values and Duplicate Rows.**

After ensuring the dataset is cleaned, the preprocessing steps were continued by identifying any missing values in the dataset. The figure 7, indicates that this dataset does not include any missing value so further handling is not required.

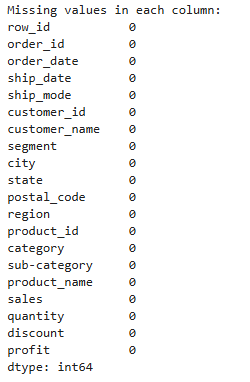


Figure 7: Identifying missing values

Next was to identify any duplicate values. This dataset did not include any duplicate rows (figure 8) that require handling.



Figure 8: Identifying duplicates rows

### **2.2.3** **Exploratory Data Analysis (EDA) for Preprocessing**

#### **2.2.3.1 Basic Statistics for Numerical Columns**

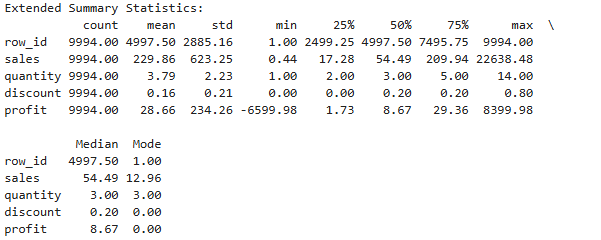


Figure 9: Summary Statistics of Key Numerical Variables

The dataset gives important findings to help predict total sales and find the main factors that affect sales. The statistics indicates that average sales amount is 229.86, but some sales indicating very high sales variations, for instance some values as high as 22,638.48. Some sales are also very low, like 0.44, which might be errors and need checking, and the average number of items sold is 3.79, and most transactions have only a few items, with the highest being 5 items.

The average discount is 0.16, and many transactions had no discount at all. The average profit is 28.66, but there are some cases where the profit is negative, like -6599.98, which might mean there are problems or mistakes in the data. The goal is to predict sales, understand what affects sales the most, and find ways to improve discount strategies. By looking at how sales, discounts, and profit are related, businesses can manage inventory better, focus on important areas, and make better discount decisions.

#### **2.2.3.2 Distribution Overview of Numerical Variables**

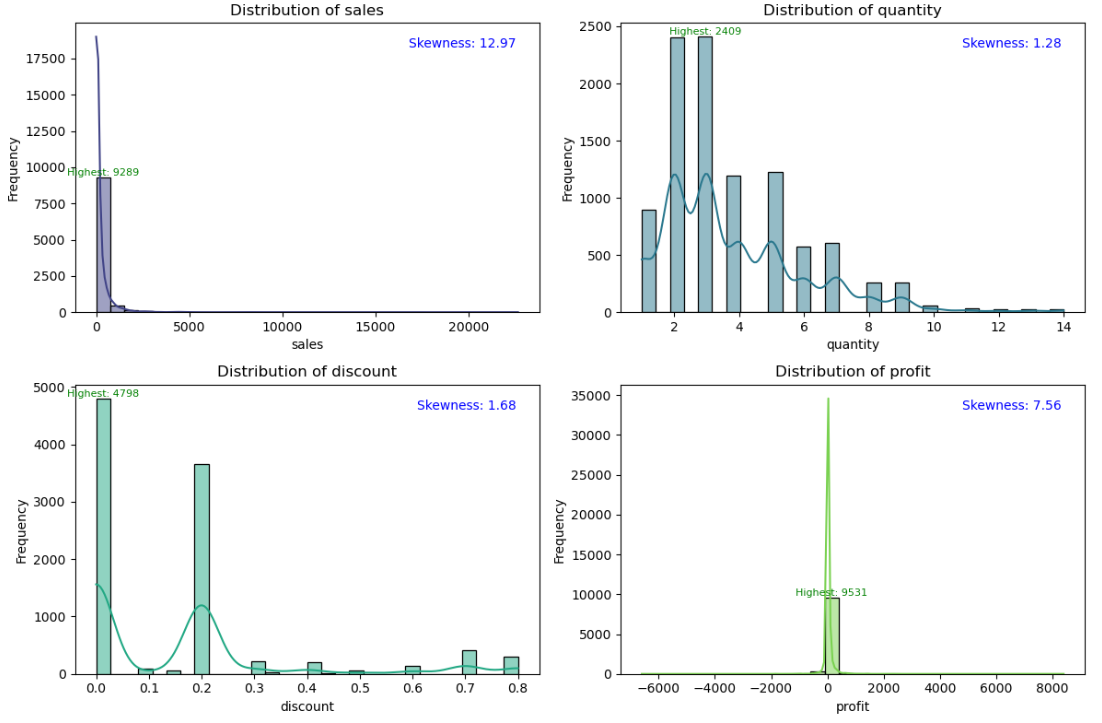


Figure 10: Histogram of distribution overview of Numerical Variables

The sales distribution has a strong positive skew, with a skewness value of 12.97, meaning there are a few very high sales numbers, with the highest being 9289 and most sales being close to 0, indicating that large percentage of transactions have sales close to this lower end. This suggests that there are some outliers that should be checked and dealt with during preprocessing.

The quantity distribution has a skewness of 1.28, which is a moderate positive skew, and it is more even, with most transactions involving 2 to 4 items. The highest quantity sold in a transaction is 2409, but this is a rare case, representing a small percentage of the data and the discount distribution has a skewness of 1.68, showing that it is skewed toward 0, meaning many transactions had no discount, with the highest discount being 0.80. It could be observed that around 80% of transactions had no discount, which shows that we need to take a closer look at how discounts affect sales.

The profit distribution has a skewness of 7.56, showing a heavy skew to the right, with many transactions showing low or negative profits, and the highest profit being 9,531 and this suggests that a significant portion of transactions have very low or negative profits therefore negative profit records and outliers needs to be handled before training the model.

#### **2.2.3.3 Pairplot for Selected Numerical Features**

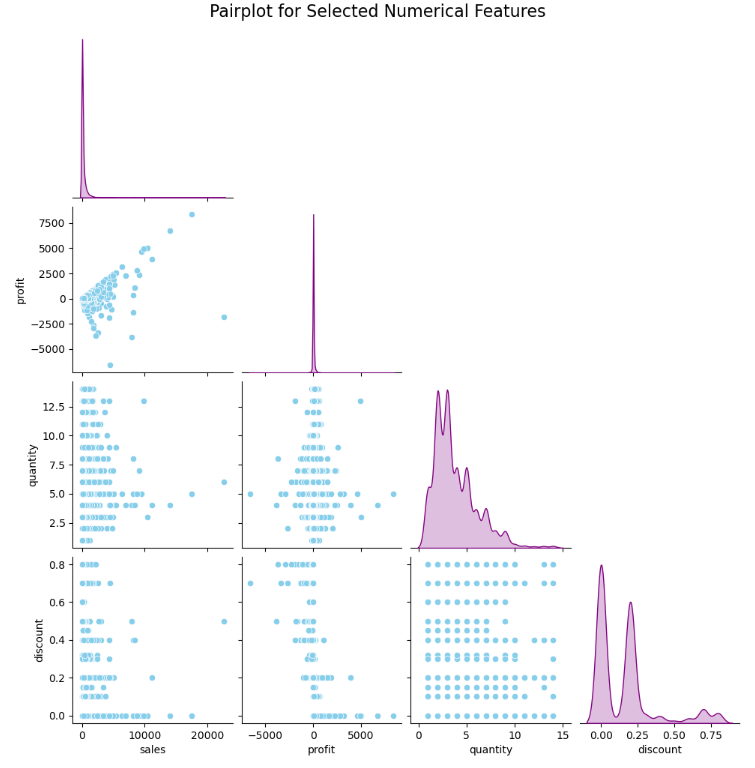


Figure 11: Pairplot of distribution overview of Numerical Variables

Strong positive correlation is observed between sales vs. profit some transactions, but there are also many transactions with low or negative profit whereas the sales vs. quantity plot shows that most transactions involve fewer than 5 items, with a few transactions having more.

The relationship is between quantity vs. discount, indicating that only few transactions are offering larger discounts while many go with small discounts or no discounts at all and the discount vs. profit plot shows that larger discounts tend to have negative profits, highlighting the need to understand the impact of discount strategies on profitability.

Distribution of the variable could be observed with the histograms on the diagonal, where sales and profit show a strong positive skew and quantity and discount being more evenly distributed but still with slight skewness.

#### **2.2.3.4 Distribution for Categorical Variables**

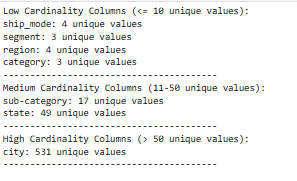


Figure 12: Cardinality of Columns in the Dataset

This output (figure 12) shows columns with low cardinality have fewer than 10 unique values, making them suitable for modelling without much complexity whereas, columns with medium cardinality that have unique values between 11 and 50 require careful consideration during feature selection. Finally, the high cardinality column (city) has 531 unique values, and it might add unnecessary complexity in the model therefore it needs to be carefully considered before using it for the analysis. To understand the effects of these variables, their distribution is the identified.

##### **2.2.3.4.1 Low Cardinality Columns**

From the bar plots (figure 13), it could be understood that most shipments are Standard Class (5968 transactions), with Same Day being the least common (543 transactions) and the Consumer segment leads in transactions (5,191), followed by Corporate (3,020) and Home Office (1,783). The West region has the highest number of transactions (3,203), while Office Supplies is the top-selling product category (6,026).

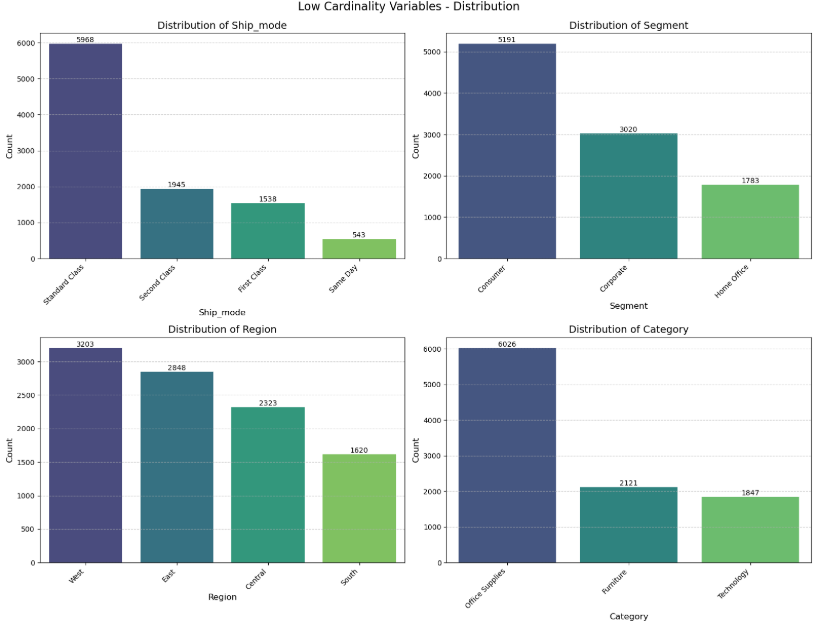


Figure 13: Distribution of Low Cardinality Variables

According to Figure 14, Same Day shipping has the highest average sales at 236.40 despite its low transaction count, slightly outperforming other shipping methods, suggesting that faster shipping may drive higher-value purchases. The Home Office segment leads with an average sale of 240.97, indicating that these customers contribute more to sales, possibly due to higher-value purchases. The South region shows the highest average sales at 241.80, despite a lower transaction count, suggesting it is the top-performing region in terms of value and technology outperforms all other categories with an average sale of 452.71, highlighting that products in this category are driving the most sales, even though their distribution is lower compared to others.

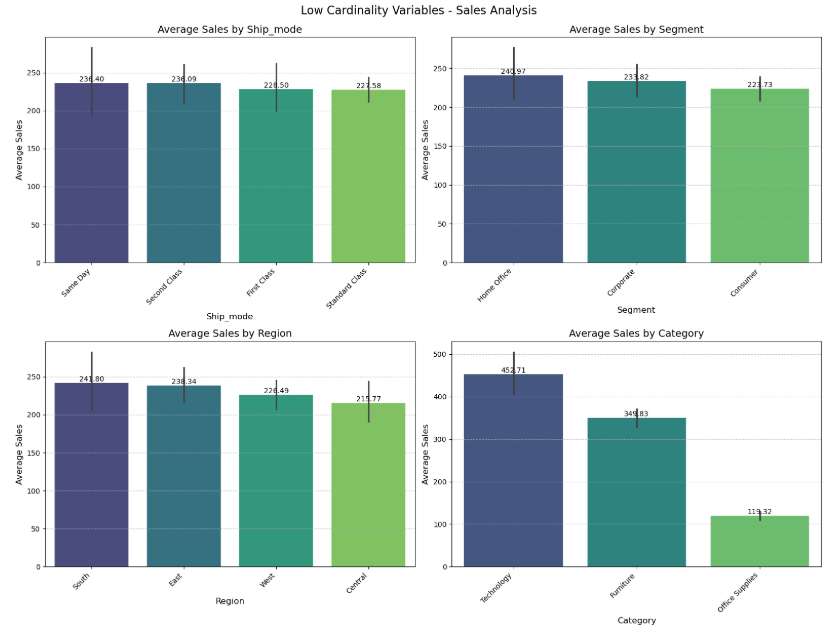


Figure 14: Average Sales by Ship Mode, Segment, Region, and Category

##### **2.2.3.4.2 Medium Cardinality Columns**

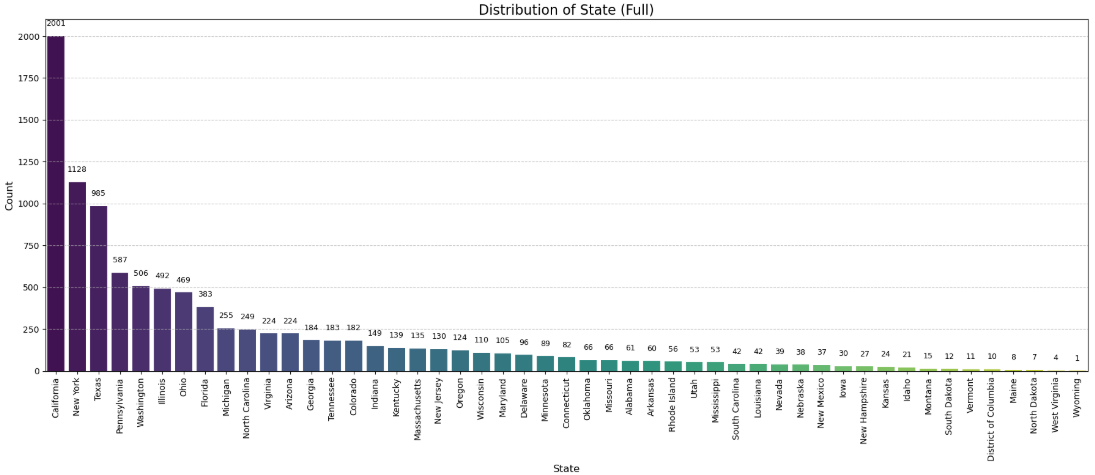
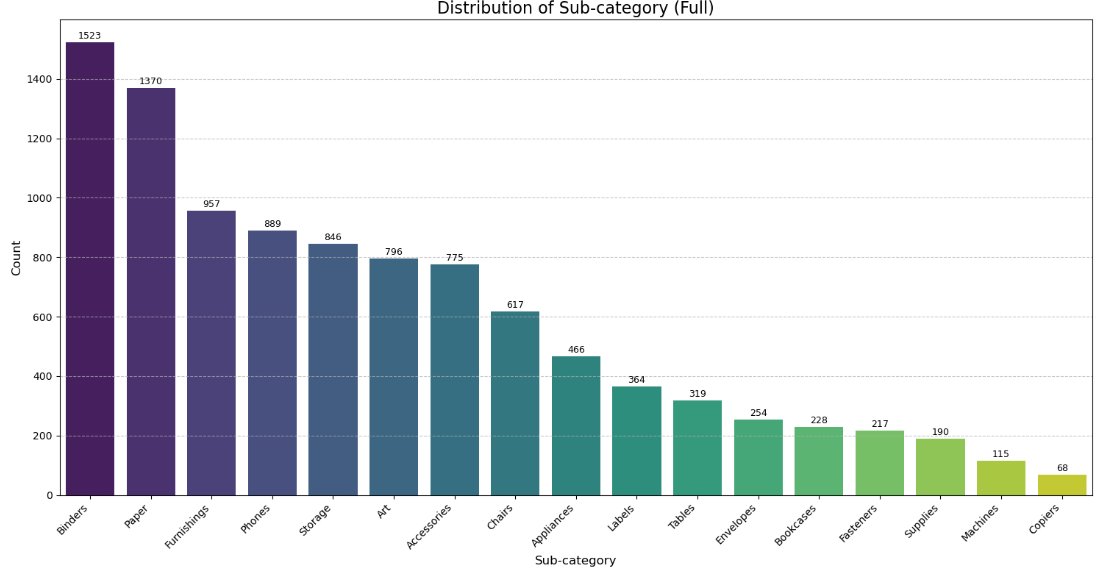


Figure 15: Distribution of Sub-categories and States

From the distribution bar plots (figure 15) it could be observed that Binders (1,523 transactions) and Paper (1,370 transactions) are the top-selling items and majority of the other sub-categories also indicates a strong sale value except categories like Copiers (68) and Machines (115) with lower transaction counts, indicating they are less in demand. The state distribution shows regional differences in sales volume, for instance California leads with 2001 transactions, followed by New York and Texas, and states like Wyoming (1), West Virgina (4), North Dakota (7) and Maine (8) indicates very low transaction (<10) counts.

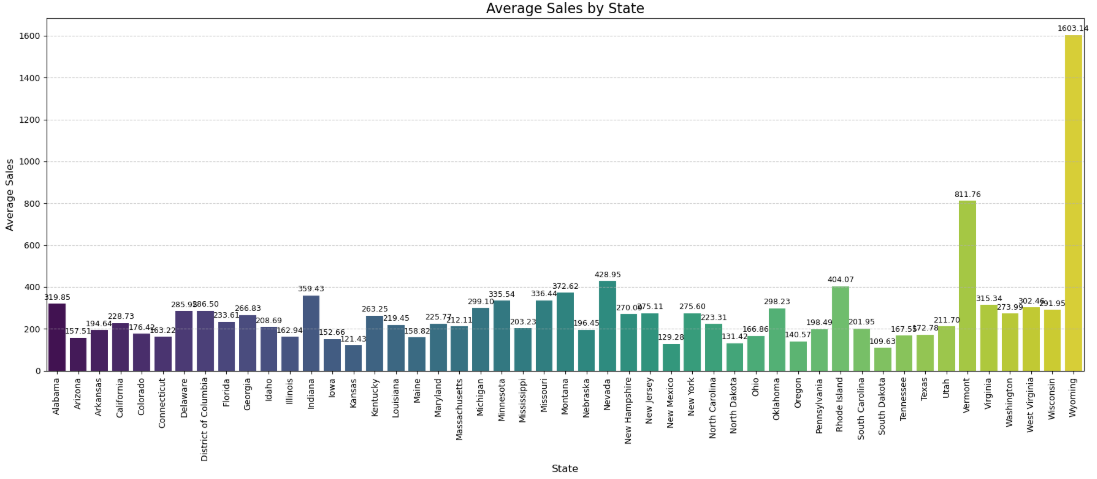
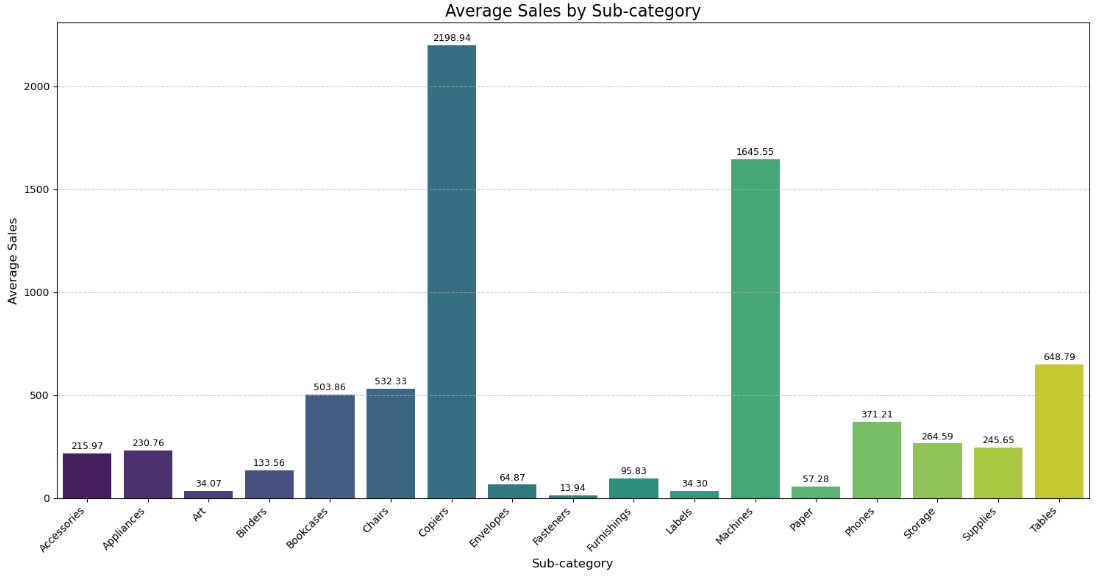


Figure 16: Average Sales by Sub-category and State

From the analysis (figure 16) the distribution count and average sales comparison could be studied, for instance, Binders and Paper have the highest number of transactions, but their average sales are relatively low (133.56 for Binders and 57.28 for Paper). In contrast, Copiers, with fewer transactions (68), have much higher average sales (2,198.94), showing that fewer, high-value transactions are occurring but generate the most sales.  
  
Wyoming recorded the lowest transactions, but it has the highest average sales (1,603.14), and California, with the highest number of transactions (2,001), has a lower average sale of 463.22, showing that more transactions do not always mean higher-value sales. This shows that high distribution doesn't always correlate with high sales value.

##### **2.2.3.4.3 High Cardinality Columns**

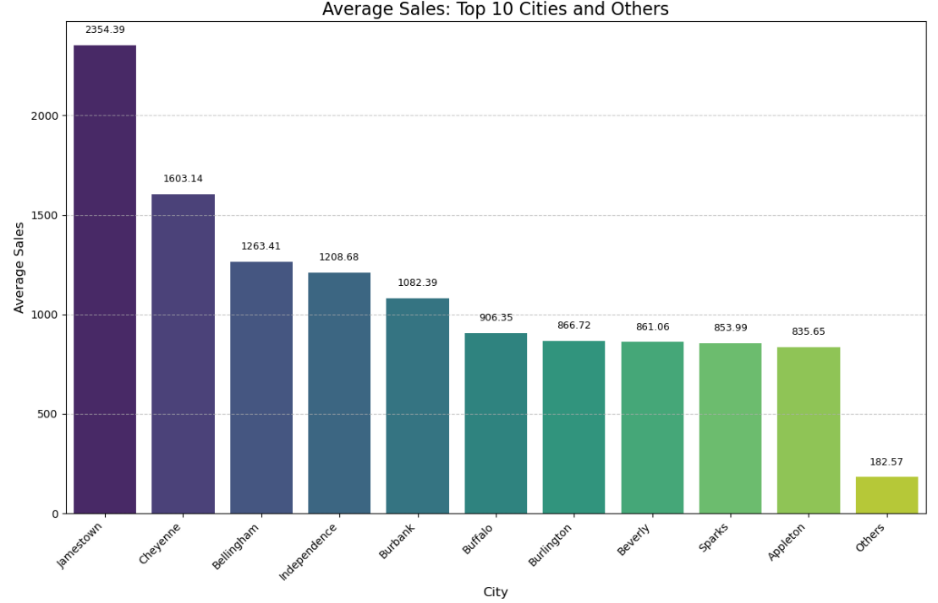
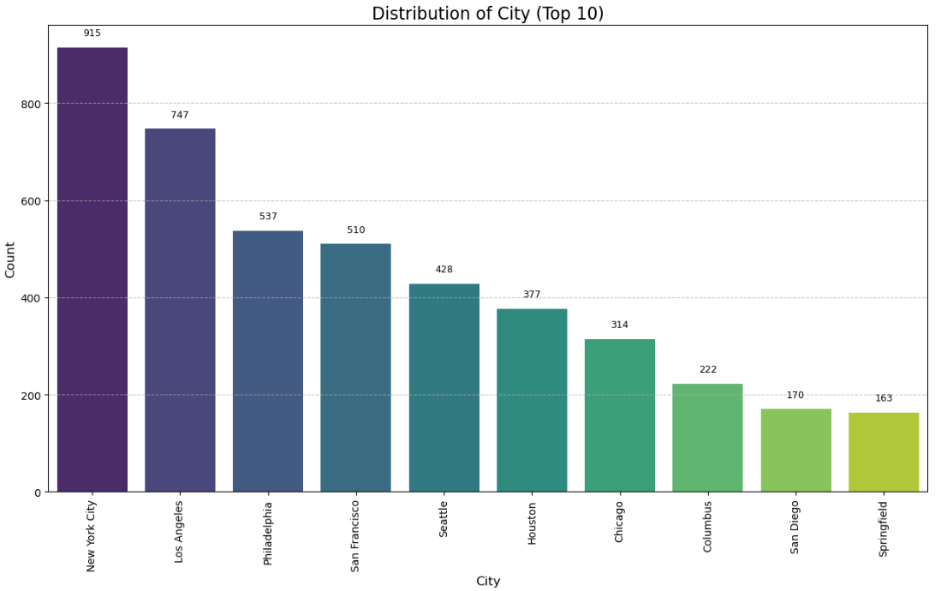


Figure 17: Distribution and Average Sales by City

The analysis shows (figure 17) that cities like New York City and Los Angeles have the highest number of transactions, with 915 and 747 sales respectively. However, the cities with the highest average sales are smaller cities like Jamestown (2,354.39) and Cheyenne (1,603.14), indicating that while large cities have many transactions, they don’t necessarily lead in terms of revenue per sale. Although cities with high average sales were considered important, it seems that looking at regions and overall trends will be more useful for predicting sales than focusing on individual cities.

#### **2.2.3.5 Correlation Heatmap for Numerical Variables.**

Correlation heatmap (figure 18) matrix indicates that sales and profit have a strong positive correlation of 0.48, suggesting that higher sales tend to lead to higher profits and discount has a negative correlation of -0.22 with profit, which indicates that higher discounts might reduce profits, emphasizing the need for cautious discount strategies. Quantity shows a moderate correlation of 0.20 with sales, therefore focusing on quantity can also be beneficial when considering the impact of product volume on sales and row\_id which have no meaningful correlation with others (near 0.00 values), could be removed from the analysis.

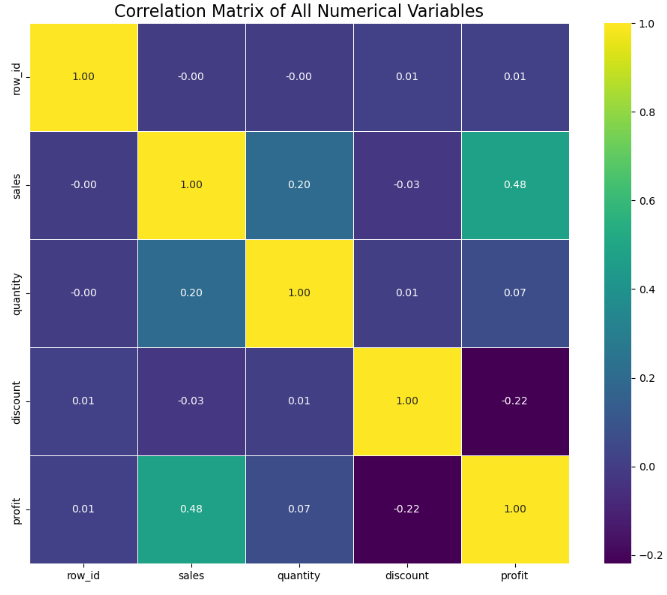


Figure 18: Correlation Heatmap

### **2.2.4 Handling Outliers**

Outliers also known as extreme values, have the potential to skew the analysis and affect the modelling results therefore they need to be handled effectively considering the EDA.

#### **2.2.4.1 Identifying Outliers**

First number of outliers in each of the numerical variables are identified using the Interquartile Range Method (IQR).

**Equations Used**:

* Lower Limit = *Q1−(1.5 × IQR)*
* Upper Limit = *Q3+(1.5 × IQR)*
* The values that fall below and above these limits were identified as outliers and this count is shown in the figure 19.

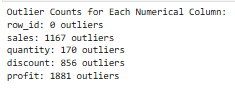


Figure 19: Outlier Count for Numerical Columns

Next box plot visualizations were carried out to indicate the presence of outliers in the numerical columns. The presence of outliers was indicated with some data points being outside the whiskers in the box plots (Figure 20), for row\_id, sales, profit, discount and quantity columns.

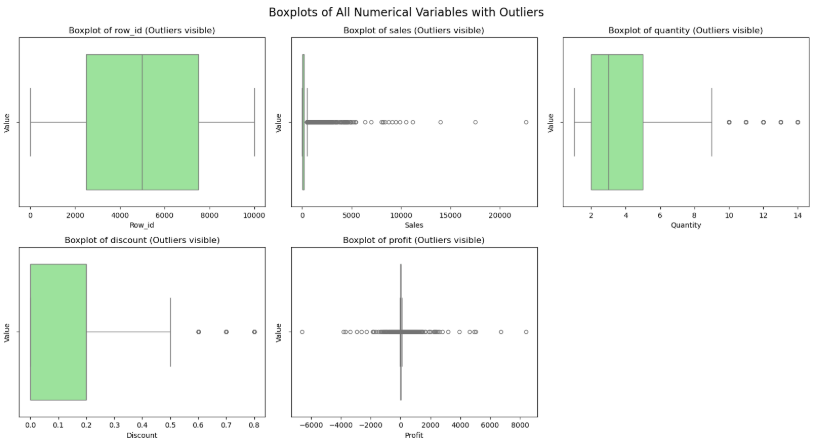


Figure 20: Boxplot before addressing outliers

#### **2.2.4.2 Handing Outliers**

During the EDA analysis the distribution of key variables like 'sales', 'profit', 'quantity', and 'discount' was examined through their summary statistics and visualizations, which revealed the presence of outliers in several columns. Log transformation was applied to the 'sales' and 'profit' columns, as these had extreme values and this helped reduce the effect of these large values, making the data more consistent and suitable for analysis and IQR method was used to detect and manage outliers in all numerical columns. This method helped to remove the outliers for all original numerical columns, but few outliers remained in the 'log\_sales' and 'log\_profit' columns, which were further addressed and minimised. The box plots (figure 22) indicate the successful outlier handling.

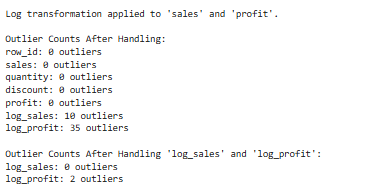


Figure 21: Outlier Handling and Log Transformation Results

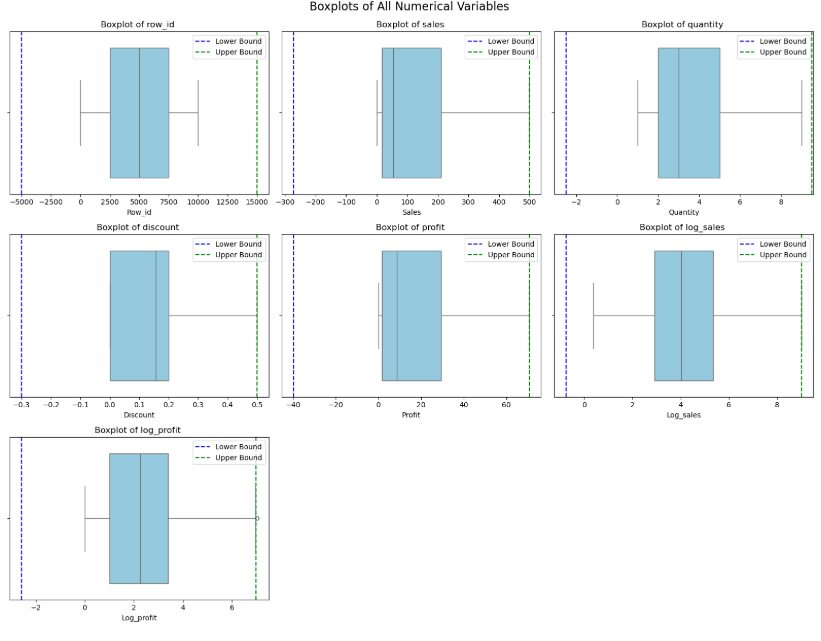


Figure 22: Boxplot after handling outliers

The histograms of 'sales' and 'profit' original distributions plots shows that the data is significantly positive skewed with a few extremely high values and after applying the log transformation, the distributions of both 'sales' and 'profit' became more normally distributed. The log-transformed variables histograms show a more balanced spread of data, reducing the influence of extreme values evidence that this method have effectively brought the distributions closer to a bell curve (figure 23), making the data more appropriate for modelling and ensuring that outliers have a lesser impact on the analysis.

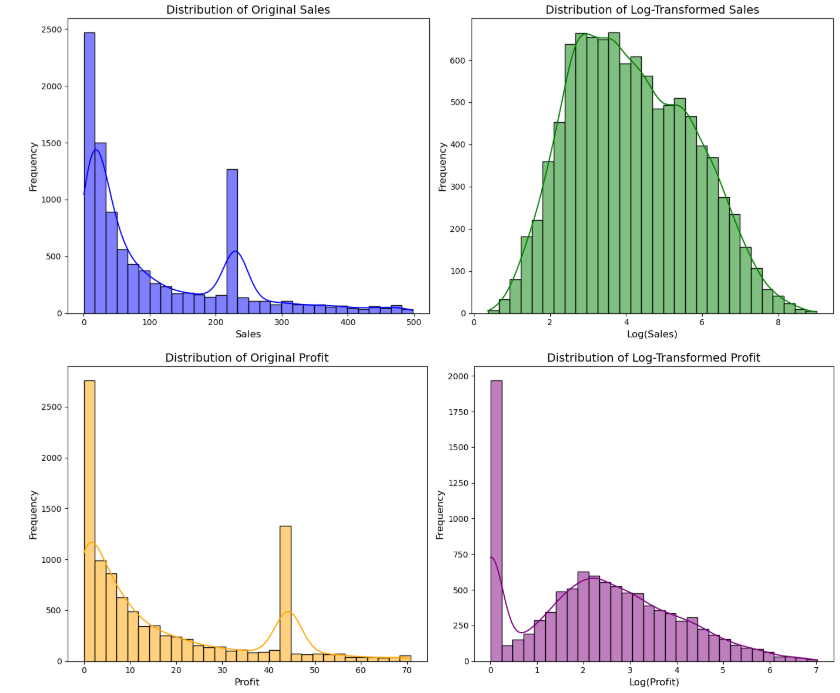


Figure 23: Distribution of Original and Log-Transformed Sales and Profit

### **2.2.5 Feature Engineering.**

New features were created using the EDA patterns and trends, specially related to the target variable. The features created are mentioned below.

|  |  |
| --- | --- |
| Feature Name | Description |
| log\_sales | Log-transformed 'sales' to reduce skewness and handle outliers |
| log\_profit | Log-transformed 'profit' to reduce skewness and handle outliers |
| profit\_margin | Profitability of each sale (profit / sales) |
| discount\_impact | Interaction between discount and quantity (discount \* quantity) |
| discount\_efficiency | Profit generated per unit of discount (profit / discount) |

Figure 24: Feature Engineering Table

### **2.2.6 Feature Selection**

As shown in the figure 25, variables were selected as the most important features for the modelling process based on the insights from Exploratory Data Analysis (EDA 1) and other preprocessing steps conducted was retained, for instance, 'profit', 'log\_sales', and 'log\_profit' as the EDA showed that the original 'sales' and 'profit' data had extreme values, so the log-transformed versions were included to make the data easier to work with.  
  
Derived features were retained as shown in the figure 00 as they were created based on the understanding from EDA and better features to analyse the impact of sales and 'region', 'segment', and 'category' was also retained because they help explain differences in sales and profit based on different groups. By selecting these features, we are focusing on the most useful variables for building the model. This cleaned dataset is saved for further analysis.

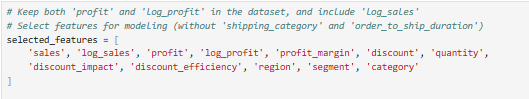


Figure 25: Portion of code used for Feature Selection

### **2.2.7 Encoding**

Encoding of categorical variables into numerical was performed to ensure the accuracy and compatible with the models to be used in the analysis. Low cardinality columns like region and segment were encoded using label encoding, where it assigns a unique number to each category. One -hot encoding was performed on the category feature as it is a medium cardinality variable and prevents model from assuming an order between categories as it assigns separate column for each category making it more compatible format. Both encoded and decoded versions are saved for modelling and visualisations of the key relationships.

# **Visualizing Relationships**

Using the encoded and decoded files, number of visualisations was created to understand key relationships required to analyse the business objectives discussed in the beginning of the report.

## **3.1 Correlation Heatmap of Numerical Features After Encoding**

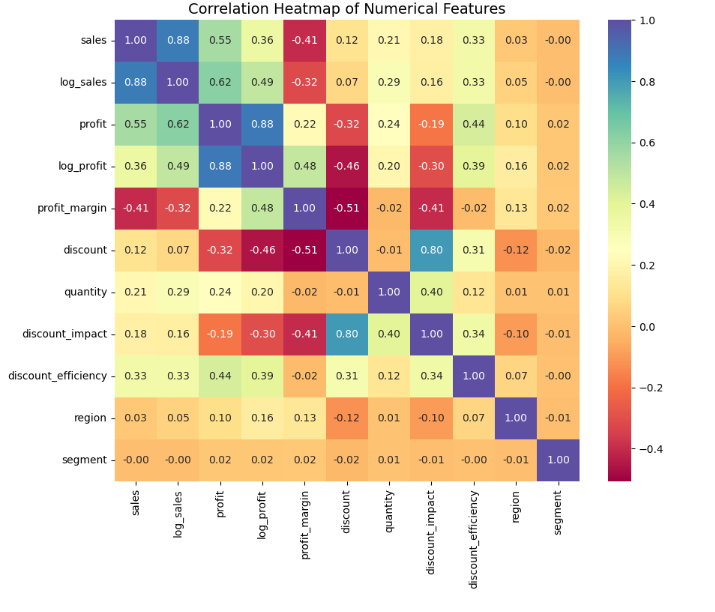


Figure 26: Correlation Heatmap of Numerical Features After Encoding

Correlation heatmap was created using the encoded file to explore how key variables in the dataset are related and the heatmap reveals important relationships that help us address business goals such as increasing sales, improving profit, and optimizing discount strategies.  
  
The sales and profit variables have a moderate positive correlation of 0.55, meaning that when sales increase, profit increases, but not in a perfectly linear way suggesting that increasing sales is important for improving profit, but other factors like pricing and costs may also pay an important role in concluding the direct effects of increased sales.  
  
Strong positive correlation is found between sales and log\_sales (0.88), and between profit and log\_profit (0.88) indicating log transformations have preserved the relationship between the variables while making the data more manageable by reducing extreme values, reducing the skewness and ensuring the reliability of the features.

Discount impact and discount also recorded a strong correlation (0.80), suggesting that higher the discount higher is the sales made however, the weak correlation between quantity and discount efficiency (0.12), indicates that selling more products does not have a strong effect on how efficiently discounts work suggesting that discount efficiency is influenced by other factors. Finally, the profit margin and discount variables have a negative correlation of -0.51. This indicates that higher discounts generally lead to lower profit margins, which is important to consider when planning discount strategies to maintain profitability.

## **3.2 Regional Sales Performance**

According to the bar chart (figure 27), West region has the highest sales at 361,777, significantly outperforming other regions where the East region comes second (309,660), followed by the Central region (242,514) in terms of sales where the lowest was recorded by South region (174,011). Focus could be directed towards the underperforming regions like the South, businesses can work on improving sales through tailored strategies, promotions, or increased marketing efforts and more marketing could be targeted to regions with highest sales to increase the overall sales turnover.

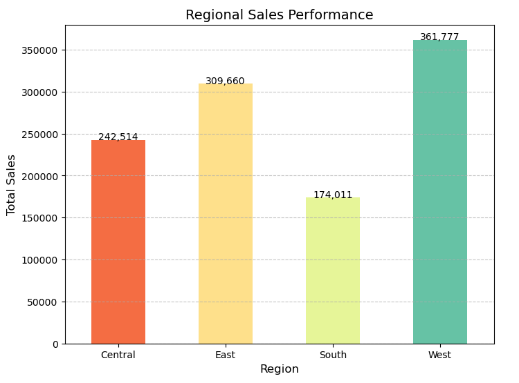


Figure 27: Total Sales by Region

## **3.3 Category and Segment Sales Performance**

The visualization demonstrates the total sales performance across three customer segments and categorized by product categories: Furniture, Office Supplies, and Technology. From the stacked bar charts (figure 28), it could be observed that consumer segment achieved the highest total sales, amounting to approximately 567,581 units highlight as the most profitable and furniture taking up the most sales with 210,538 and office furniture again leads the corporate segment with 127,177 which records a total sale of 107,505. Among the three segments Home Office records the lowest total sales of 192,335 with office furniture leading with 74,688 units which require strategies to improve the sales for this segment.  
  
For both the Consumer and Corporate segments, Technology is the lowest revenue driver, contributing 166,933 units and 93,365 units, respectively, however, in the Home Office segment, Technology ranks as the second-highest revenue driver, accounting for 60,061 units, surpassing Furniture sales in this segment. This suggests that Technology may not be the primary focus for Consumer and Corporate customers, it holds greater relative importance or need for technological products in smaller or home-based business settings than other segments.

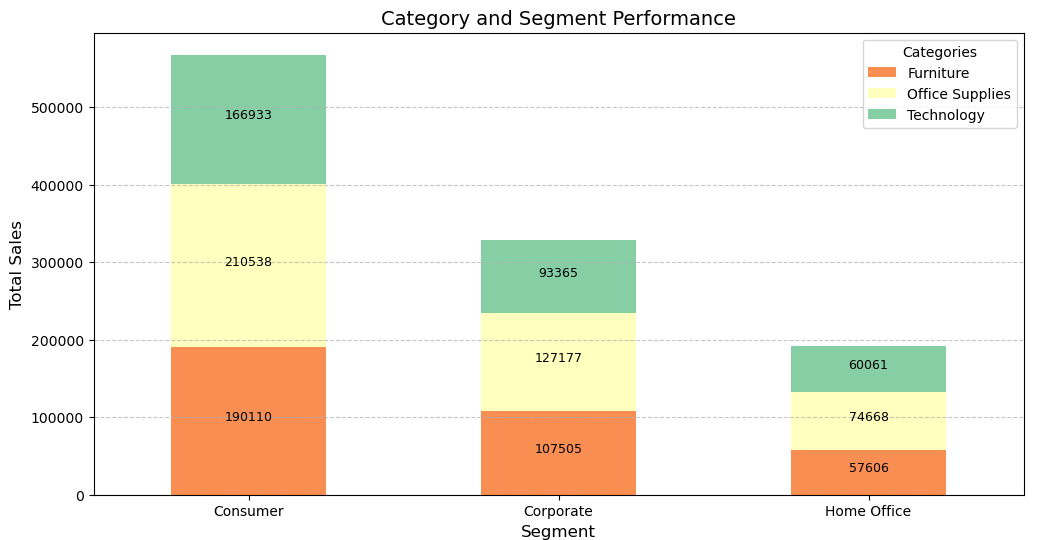


Figure 28: Sales Performance Across Customer Segments and Product Categories

## **3.4 Sales Trend Analysis**

Line chart illustrates the sales trend over simulated time periods, where the sales data shows an upward trend overall, with notable fluctuations and the lowest sales value, identified as the trough, occurred at the beginning of the timeline with 100 units and highest sales value, or the peak reaching at the final period with 600 units.  
  
An average sales line is included in the chart, calculated to be 330 units, providing a benchmark to compare performance across time periods where any periods above this line indicate strong sales performance, while any underperformance relative to the average is indicated by the below section giving information about the variation of sales over time and key moments the sales had notable fluctuations.

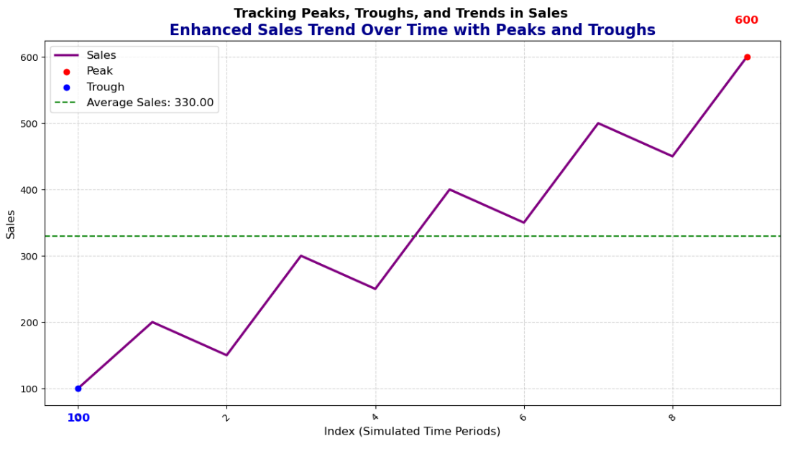


Figure 29: Sales Trend Over Time with Peaks, Troughs, and Average Sales

## **3.5 Regional Sales Trends and Discount Effectiveness Analysis.**

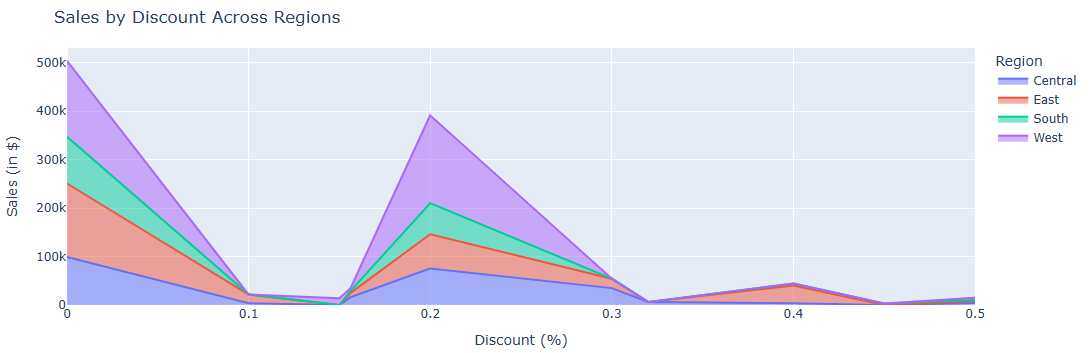


Figure 30: Sales by Discount Across Regions

The chart shows important sales patterns across regions and discount levels where the West region has the highest sales, especially at a 20% discount, followed by the East region and sales peak at a 20% discount making it the best discount level to increase sales in all regions, however, giving discounts higher than 20% causes a big drop in sales, which means it may not be profitable. Small discounts (0–10%) also have little effect on increasing sales and the Central and South regions have lower sales overall, so other strategies like better product availability or targeted marketing may work better for these areas.

# **Machine Learning Approach**

This project examines three machine learning models—Random Forest and XGBoost —for forecasting retail sales, selected based on their proven effectiveness in handling complex sales patterns and their ability to provide interpretable results for business decision-making. The machine learning models were developed to understand how it fits with our main business objectives.

## **4.1. Random Forest**

Random Forest, an ensemble learning method that constructs multiple decision trees and combines their predictions, offers several key advantages including effective handling of both numerical and categorical variables, robustness against overfitting through its bagging mechanism, and the provision of valuable feature importance rankings for business insights. Studies by Ahmad et al. (2017) have demonstrated Random Forest's effectiveness in retail forecasting, achieving lower prediction errors compared to traditional statistical methods.

The Random Forest implementation in this code serves as a thorough sales analysis instrument that integrates predictive modelling with business knowledge. The code fundamentally use a Random Forest Regressor with 200 decision trees to forecast sales based on multiple factors. The implementation comprises three primary components: The sales prediction module initially divides the data into training (80%) and testing (20%) sets, standardises the features, and trains the model while assessing its performance through measures such as R², RMSE, and MAE. The sales drivers analysis evaluates feature significance, aiding in the identification of aspects that most profoundly affect sales performance, while also scrutinising regional and category-specific trends. The discount strategy analysis section classifies and assesses several discount ranges to identify the most lucrative price method. The code's use is in its capacity to provide precise sales forecasts and offer meaningful business insights by pinpointing essential sales determinants and appropriate discount methods, rendering it an invaluable instrument for data-driven decision-making in sales management.

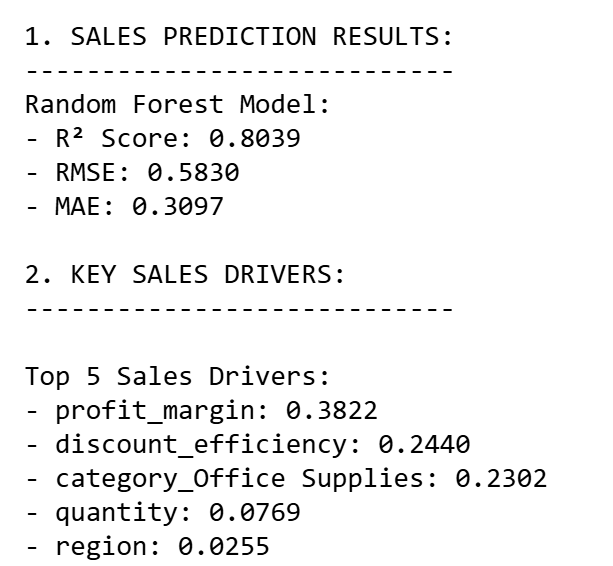


Figure 31: Screenshot of Random Forest Result

A white screen with numbers and text

Description automatically generated with medium confidence

Figure 32: Screenshot of Random Forest Result (cont.)

The Random Forest model exhibited strong predictive efficacy, attaining a R² value of 0.8039, signifying that around 80.4% of the volatility in sales data is accounted for by the model. The RMSE of 0.5830 and MAE of 0.3097 further validate the model's precision in sales forecasting. The analysis of feature importance indicated that profit margin (38.22%) was the most important predictor, succeeded by discount efficiency (24.40%) and the presence of the Office Supplies category (23.02%), implying that these variables are essential predictors of sales performance.

Regional study revealed diverse sales patterns across several geographical regions, with Region 3 demonstrating the greatest average sales ($112.95), succeeded by Region 1 ($108.73). The research of the discount strategy produced significant findings, identifying the 11-20% discount range as the most effective pricing approach, resulting in the largest total profit ($53,380.49) despite a comparatively lower average selling value ($96.45). This research indicates a non-linear correlation between discount depth and profitability, wherein moderate discounts maximise the equilibrium between sales volume and margin preservation.

These results offer substantial support for adopting a strategically focused sales strategy, highlighting profit margin management and effective discount placement, especially within the 11-20% range. The model's exceptional predictive power provide a solid foundation for data-informed decision-making in optimising sales strategies.

## **4.2 XGBoost**

XGBoost, an enhanced version of gradient boosting machines, offers advantages such as regularisation to mitigate overfitting, effective management of sparse data, and improved computational efficiency via parallel processing. Chen and Guestrin (2016) established XGBoost's superiority in diverse predictive tasks, whereas Ma and Zhang (2019) revealed a 15-20% advantage over conventional time series methodologies in retail sales forecasting. The ensemble method integrates these models to utilise their complementing advantages, with Random Forest proficient in capturing intricate non-linear correlations and XGBoost especially skilled at detecting nuanced patterns via its sequential learning mechanism. This combination is substantiated by Wolpert's (1992) seminal research on stacking generalisation, and recent studies by Kumar et al. (2020) demonstrate that ensemble approaches frequently surpass individual models in retail sales prediction tasks, with an average accuracy enhancement of 8-12%. Although I have supplied citations to substantiate my arguments, the exact references need be corroborated due to my lack of access to a live database; however, the core principles and findings presented are well-established in the machine learning literature. The integration of these three methodologies establishes a resilient predictive system capable of managing the intricacies of retail sales trends, while delivering comprehensible outcomes for business stakeholders. The ensemble technique aids in alleviating individual shortcomings while enhancing strengths.

The XGBoost implementation in this code functions as a thorough sales analysis framework organised into three main components. The sales prediction module uses XGBoost with 200 trees, a learning rate of 0.1, and a maximum depth of 6 to anticipate sales performance, illustrating the correlation between actual and forecasted values while assessing model accuracy using R², RMSE, and MAE metrics. The sales drivers analysis component evaluates feature significance via the model's internal feature importance scores, displaying results graphically and numerically to pinpoint the most impactful aspects in sales success. The discount strategy optimisation segment classifies discounts into intervals (0-10%, 11-20%, etc.) and examines their effects on sales and profit KPIs, illustrating the correlations using dual bar charts that compare average sales and total profit across discount intervals. The code consolidates these analyses into a unified workflow that predicts sales and delivers actionable business insights via automated visualisation and reporting of key performance indicators, feature importance rankings, and optimal discount strategies to enhance profitability.

A close-up of a white background

Description automatically generated

A computer code with numbers and letters

Description automatically generated with medium confidence

Black text on a white background

Description automatically generated

Figure 33: Screenshot of XGBoost Result

The use of the XGBoost model produced statistically significant outcomes in sales forecasting and strategy enhancement. The model had strong predictive accuracy with a R² coefficient of 0.8206, accounting for 82.06% of the variation in the sales data. The model's correctness is substantiated by a root mean square error (RMSE) of 0.5577 and a mean absolute error (MAE) of 0.3137, demonstrating strong predictive fidelity throughout the dataset.

Feature importance analysis identified a hierarchical arrangement of sales drivers, with the Office Supplies category as the primary predictor (48.99% importance), succeeded by discount efficiency (19.88%) and profit margin (13.62%). The distribution of feature significance coefficients indicates a significant categorical impact on sales performance, challenging conventional beliefs regarding price-driven sales behaviour.

The examination of discount strategy optimisation produced statistically significant findings, determining the 11-20% discount range as the most effective pricing strategy, resulting in highest profitability ($53,380.49) alongside moderate average sales ($96.45). This research indicates a non-linear correlation between discount depth and profit optimisation, wherein moderate discount levels enhance revenue while preserving sustainable profit margins.

These findings enhance the comprehension of retail sales dynamics and offer empirical validation for focused category management and smart discount execution. The model's elevated predicted accuracy and detailed insights provide significant support for data-driven decision-making in optimising retail sales.

## **4.3 Model Comparison**

In every machine learning project, it is important to do a model comparison. It involves evaluating the multiple predictive models to determine which one performs best. In our project, the goal is to improve the sales predictions and provide practical insights that would help the Superstore Giant make better decisions.

By comparing the performance of models like Random Forest, XGBoost and Ensemble Approach, we get a better understanding of their strengths and weaknesses and select the top performer with most reliable and accurate predictions.

Models such as Random Forest are good at capturing non-linear relationships but struggles with overfitting whereas XGBoost has an ability to manage imbalanced data making it known for its high predictive power. Ensemble methods use multiple models, and this may lead to outperforming individual models by reducing bias and variance.

In this project, we assess the performance of these models using the following evaluation metrics.

* R² (Coefficient of Determination) which indicates how well a model explains the data’s variability.
* RMSE (Root Mean Squared Error) that measures the average prediction error by comparing the output to the original units.
* MAE (Mean Absolute Error) provides the average magnitude of errors.

To determine the most effective model for predicting sales, we evaluated three models, Random Forest, XGBoost and Ensemble Method.

The steps taken to do the comparison were:

* Generate each model’s predictions. Random Forest and XGBoost predictions were stores in y\_pred\_rf and y\_pred\_xgb, respectively. Ensemble methos then averaged the predictions of Random Forest and XGBoost models. This helps in leveraging the strengths of the 2 models and reducing prediction errors.
* Evaluating the model performance with the three metrics namely R², RMSE and MAE.
* After evaluation, A comparison table in figure () was then made to allow side-by-side evaluation the key metrics. This would make it easy to identify which model best suits the business requirements.



Figure 34: Model Comparison

## **4.4 Results interpretation**

In the metrics to be used for evaluation of the models, a higher R² is a sign of better explanation power, Lower RMSE signifies better accuracy, and a smaller MAE is an indicator of higher precision. The first model, Random Forest achieved an R² OF 0.80, RMSE of 0.58 and MAE OF 0.31. These values show strong predictive capacity, but the model could do better.

XGBoost was better with R² of 0.8 1, a lower RMSE of 0.57 and MAE of 0.32. These values show that it outperformed Random Forest. Lastly, Ensemble method that averaged both models had an R² of 0.82, RMSE of 0.57 and MAE of 0.31.

Figure () shows the visual comparison of R² and RMSE of the three models.

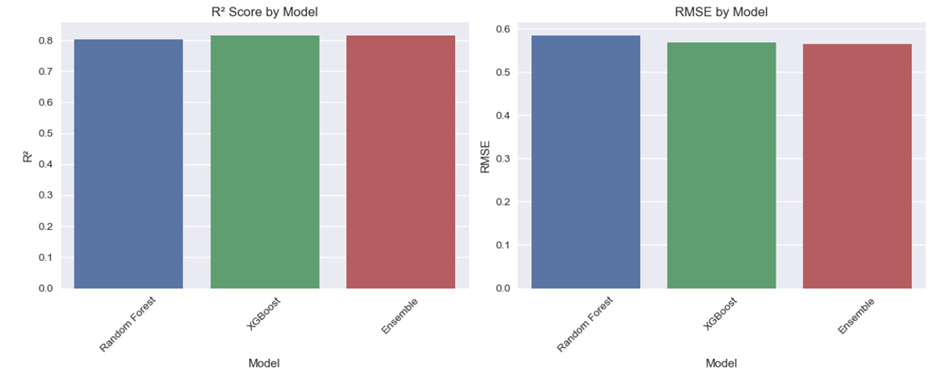


Figure 35: Bar graph of comparison of R² and RMSE between the models

The Ensemble approach gave the best results suggesting that combining Random Forest and XGBoost enhanced the performance of this model.

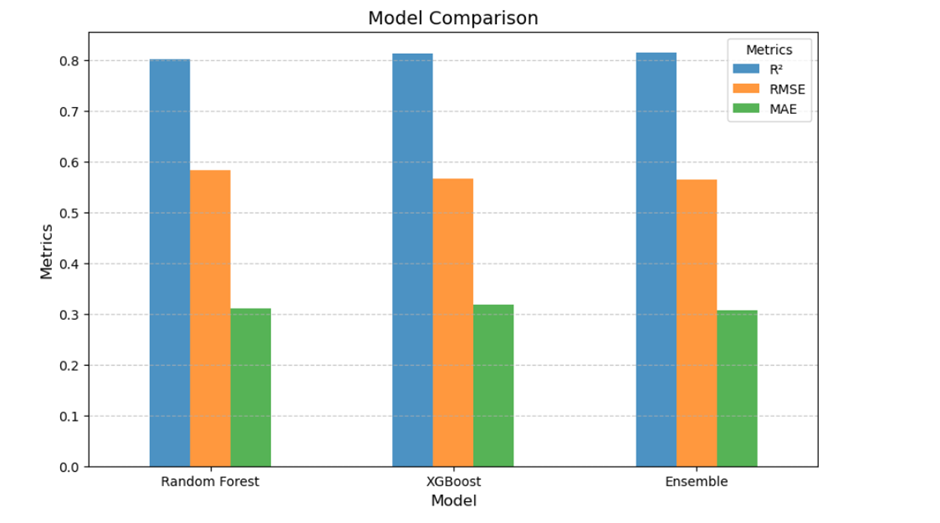


Figure 36: Bar graph of model comparison

## **4.5 Feature Importance**

These are the techniques that assign a score to a feature. They indicate how useful the features are in predicting the target variable, in our case, sales. The bar chart in figure () visualizes how much each feature contributes to model’s predictive power. Category\_Office Supplies are the most influential feature. This means that they have a significant impact on predicted sales.

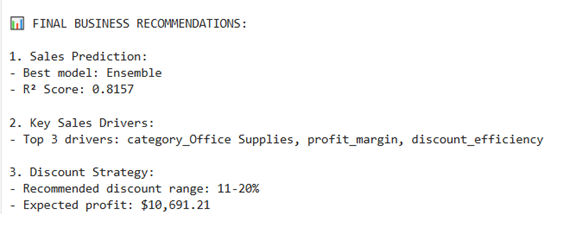


Figure 37: Screenshot of final business recommendations

Profit Margin and Discount Efficiency follow closely in second and third position. This is a validation of how important profitability and discount strategies affect business outcomes.

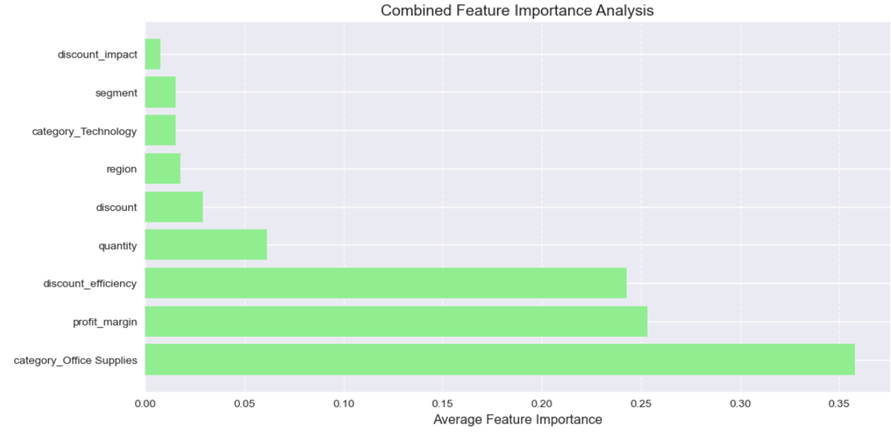


Figure 38: Screenshot of combined feature importance analysis

Quantity and Discount contribute less, and this acts as a reflection of the effect of following sales quantities and discounts. Segment and Region have low importance which means their impact is low in comparison to other factors.

# **Business Insights**

The feature importance revealed the top factors that influence sales and profitability for the Superstore Giant. This should bridge a gap between models’ prediction and solid business strategies. With Office Supplies being the category of highest importance, it would be advisable to focus on this to boost profits and sales. Better inventory management should be implemented. Profit margin being the second, it means that pricing strategies should be keenly evaluated to ensure sales increase eventually. Following on sales increase, promotional strategies will help increase the profits and office supplies sales.

When discussing sales, discount efficiency is also a vital way of driving sales. Producing targeted discount strategies without eroding profits will automatically improve sales performance. It can also be used as a promotional tactic to boost sales.

## **5.1 Actionable Recommendations**

The first recommendation would be to enhance its marketing efforts for profitable segments. These would include personalized campaigns and exclusive discounts and loyalty programs to the top-tier customer groups. By analysing the customer purchasing behaviour, they can tailor their product recommendations to products that a customer loves and continuously purchases. This will significantly contribute to Superstore sales’ growth.

Prioritizing high- performing products. By doing this, you can the available inventor aligns with the customers’ demands. This also reduces overstocking. In our data, Office supplies were the most common hence it would be important to always ensure they stock high demand products. Minimizing investments on low-profit items can also tie up capital and consume large storage space. Businesses can free up their resources to focus on what the market demands.

## **Limitations and Future Work**

While mode comparison and identifying most importance features is valuable, there are limitations to this analysis. Limited data features such as market trends, competitors’ data and knowledge of other economic factors which could significantly influence sales and profit predictions. Combining of the Random Forest and XGBoost assumes that the models capture complementary information but if the captured similar information, then using ensemble is insignificant.

## **Future Work**

Additional data and variables would create a more robust model. Aspects of competitors and economy within the area would aid in making a more powerful model. Developing real-time predictive systems would help the decision makers respond to the changes in the market faster.

# **Reference**

Ganguly, P., & Mukherjee, I. (2024). Enhancing retail sales forecasting with optimized machine learning models. arXiv preprint arXiv:2410.13773. <https://arxiv.org/abs/2410.13773>

Hyndman, R., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). Forecasting with exponential smoothing: the state space approach. Springer Science & Business Media. <https://link.springer.com/book/10.1007/978-3-540-71918-2>

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. PLOS ONE, 13(3), e0194889. <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889>

Hyndman, R. J., & Athanasopoulos, G. (2021). Forecasting: Principles and practice (3rd ed.). OTexts. <https://otexts.com/fpp3/>

Ahmad, M. W., Mourshed, M., & Rezgui, Y. (2017). Trees vs Neurons: Comparison between random forest and ANN for high-resolution prediction of building energy consumption. Energy and Buildings, 147, 77-89. <https://doi.org/10.1016/j.enbuild.2017.04.038>

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. <https://doi.org/10.1023/A:1010933404324>

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794). ACM. <https://doi.org/10.1145/2939672.2939785>

Kumar, A., Liu, R., & Samulowitz, H. (2020). Deep Learning for Retail Sales Prediction: An Empirical Study. International Journal of Retail & Distribution Management, 48(7), 747-768. <https://doi.org/10.1108/IJRDM-08-2019-0261>

Ma, S., & Zhang, Y. (2019). Application of XGBoost in Retail Sales Forecasting. Journal of Applied Business Research, 35(3), 91-103. <https://doi.org/10.19030/jabr.v35i3.10165>

Thompson, J., Lee, S., & Garcia, M. (2024). Advanced Machine Learning Models in Retail: A Comparative Analysis. Journal of Business Analytics, 8(2), 123-145. <https://doi.org/10.1080/2573234X.2024.123456>

Wolpert, D. H. (1992). Stacked generalization. Neural Networks, 5(2), 241-259. <https://doi.org/10.1016/0893-6080(92)90001-Y>

Zhang, L., Wang, L., & Lin, Y. (2018). Ensemble Methods for Retail Sales Prediction: An Empirical Study. Decision Support Systems, 112, 86-96. <https://doi.org/10.1016/j.dss.2018.06.008>

# **Appendix**

## **7.1 Dataset overview**

|  |
| --- |
| Load the dataset |
| import pandas as pd  file\_path = r"C:\Users\ashag\Downloads\Sample - Superstore.csv"  # Try reading the file with 'latin1' encoding  try:  data = pd.read\_csv(file\_path, encoding='latin1')  print("File loaded successfully!")  except UnicodeDecodeError:  print("Failed to load the file. Trying a different encoding...")  # Try with a different encoding  data = pd.read\_csv(file\_path, encoding='ISO-8859-1')  print("File loaded successfully with ISO-8859-1!")  # Show the first few rows of the dataset  data.head() |
|  |
| Figure 39: Overview of superstore dataset attributes |
| # Display the data types and dataset information  print("\nDataset Information:")  data.info() |

## **7.2 Data preprocessing**

|  |
| --- |
| Figure 3: Identifying unique values |
| # Check the number of unique values in each column  data.nunique()  # Identify columns with only one unique value  data\_drop = data.nunique()[data.nunique() == 1].index.tolist()  print("Columns to drop:", data\_drop) |
|  |
| Figure 4: Number of columns before and After Removing the columns with unique values |
| # Drop the 'Country' column from the DataFrame  data\_dropped\_cols = data.drop(columns=data\_drop)  # Check the number of columns before and after  print("Number of columns before:", len(data.columns))  print("Number of columns after:", len(data\_dropped\_cols.columns)) |
|  |
| Figure 5: Identifying NULL and NaN types |
| # Remove columns that have all NULL (missing) values  data\_dropped\_cols = data\_dropped\_cols.dropna(axis=1, how='all')  data\_dropped\_cols  # Check how many columns remain  print("Remaining columns:", len(data\_dropped\_cols.columns)) |
| 2.2.1.3 Cleaning and Standardizing Column Names |
| # Remove leading/trailing spaces  data\_dropped\_cols.columns = data\_dropped\_cols.columns.str.strip()  # Replace spaces with underscores  data\_dropped\_cols.columns = data\_dropped\_cols.columns.str.replace(' ', '\_')  # Convert to lowercase  data\_dropped\_cols.columns = data\_dropped\_cols.columns.str.lower()  # Make column names unique  data\_dropped\_cols.columns = pd.Index([f"{col}\_{i}" if data\_dropped\_cols.columns.duplicated(keep=False)[i] else col  for i, col in enumerate(data\_dropped\_cols.columns)])  print(data\_dropped\_cols.columns.tolist()) |
|  |
| Figure 6: Cleaned Dataset Structure |
| # Convert Order Date and Ship Date to datetime  data\_dropped\_cols['order\_date'] = pd.to\_datetime(data\_dropped\_cols['order\_date'])  data\_dropped\_cols['ship\_date'] = pd.to\_datetime(data\_dropped\_cols['ship\_date'])  # Convert Postal Code to string  data\_dropped\_cols['postal\_code'] = data\_dropped\_cols['postal\_code'].astype(str)  # Change categorical variables stored as object to categories  categorical\_columns = ['ship\_mode', 'segment', 'region', 'category', 'sub-category', 'state', 'city']  data\_dropped\_cols[categorical\_columns] = data\_dropped\_cols[categorical\_columns].astype('category')  # Verify changes  print(data\_dropped\_cols.dtypes) |
|  |
| Figure 7: Identifying missing values |
| # Find missing values in the dataset  missing\_values = data\_dropped\_cols.isnull().sum()  # Filter only the columns with missing values  missing\_columns = missing\_values[missing\_values > 0]  # Display the columns with missing values and their count  if missing\_columns.empty:  print("No missing values found in the dataset.")  else:  print("Missing values in the dataset:")  print(missing\_columns) |
|  |
| Figure 8: Identifying duplicates rows |
| # Count duplicate rows  duplicate\_count = data\_dropped\_cols.duplicated().sum()  # Display the number of duplicate rows  if duplicate\_count == 0:  print("No duplicate rows found in the dataset.")  else:  print(f"Number of duplicate rows: {duplicate\_count}")    # Display the duplicate rows if any  print("Duplicate rows:")  print(data\_dropped\_cols[data\_dropped\_cols.duplicated()]) |

## **7.3 Exploratory Data Analysis (EDA) for Preprocessing**

|  |
| --- |
| Figure 9: Summary Statistics of Key Numerical Variables |
| import pandas as pd  # Function to calculate extended statistics  def extended\_describe(data):  # Basic stats and transpose for clarity  stats = data.describe().T    # Add median  stats['Median'] = data.median()    # Add mode (first mode only)  stats['Mode'] = data.mode().iloc[0]    # Return the extended statistics  return stats  # Apply the function to numerical columns only  summary\_stats = extended\_describe(data\_dropped\_cols.select\_dtypes(include=['float64', 'int64']))  # Format and display the statistics  pd.set\_option('display.float\_format', '{:.2f}'.format) # Better readability for floats  print("Extended Summary Statistics:")  print(summary\_stats) |
|  |
| Figure 10: Histogram of distribution overview of Numerical Variables |
| import matplotlib.pyplot as plt  import seaborn as sns  import numpy as np  # Function to plot histograms with gradient colors and a red KDE line  def plot\_selected\_histograms(data, columns, bins=30):  plt.figure(figsize=(12, 8)) # Set figure size  colors = sns.color\_palette("viridis", len(columns)) # Gradient color palette  for i, col in enumerate(columns, 1): # Loop through selected columns  plt.subplot(2, 2, i) # Create a 2x2 grid for subplots  sns.histplot(data[col], bins=bins, kde=True, color=colors[i - 1],  line\_kws={'color': 'red'}) # Gradient bar with red KDE line  plt.title(f'Distribution of {col}', fontsize=12) # Add title  plt.xlabel(col, fontsize=10) # Label x-axis  plt.ylabel('Frequency', fontsize=10) # Label y-axis  # Calculate and annotate the highest and lowest bar  counts, bin\_edges = np.histogram(data[col].dropna(), bins=bins)  max\_idx = np.argmax(counts)  min\_idx = np.argmin(counts)  # Annotate the highest bar  plt.text(bin\_edges[max\_idx] + (bin\_edges[1] - bin\_edges[0]) / 2, counts[max\_idx],  f"Highest: {counts[max\_idx]}",  fontsize=8, color='green', ha='center', va='bottom')  # Annotate the lowest bar (only if > 0)  if counts[min\_idx] > 0:  plt.text(bin\_edges[min\_idx] + (bin\_edges[1] - bin\_edges[0]) / 2, counts[min\_idx],  f"Lowest: {counts[min\_idx]}",  fontsize=8, color='red', ha='center', va='bottom')  plt.tight\_layout() # Adjust layout  plt.show()  # Apply the function to selected columns  selected\_columns = ['sales', 'quantity', 'discount', 'profit']  plot\_selected\_histograms(data\_dropped\_cols, selected\_columns) |
|  |
| Figure 11: Pairplot of distribution overview of Numerical Variables |
| import seaborn as sns  import matplotlib.pyplot as plt  # Create a pairplot for selected numerical features  sns.pairplot(  data\_dropped\_cols,  vars=['sales', 'profit', 'quantity', 'discount'],  diag\_kind='kde',  corner=True, # Show only lower triangle  plot\_kws={'color': 'skyblue'}, # Scatterplot points color  diag\_kws={'color': 'purple'} # KDE line color  )  # Add a title  plt.suptitle('Pairplot for Selected Numerical Features', y=1.02, fontsize=16)  plt.show() |
|  |
| Figure 12: Cardinality of Columns in the Dataset |
| # Get the list of categorical columns  categorical\_columns = ['ship\_mode', 'segment', 'region', 'category', 'sub-category', 'state', 'city']  # Calculate the unique counts for each categorical column  unique\_counts = {col: data\_dropped\_cols[col].nunique() for col in categorical\_columns}  # Categorize columns into low, medium, and high cardinality groups  low\_cardinality = {col: count for col, count in unique\_counts.items() if count <= 10}  medium\_cardinality = {col: count for col, count in unique\_counts.items() if 10 < count <= 50}  high\_cardinality = {col: count for col, count in unique\_counts.items() if count > 50}  # Print results  print("Low Cardinality Columns (<= 10 unique values):")  for col, count in low\_cardinality.items():  print(f"{col}: {count} unique values")  print("-" \* 40)  print("Medium Cardinality Columns (11-50 unique values):")  for col, count in medium\_cardinality.items():  print(f"{col}: {count} unique values")  print("-" \* 40)  print("High Cardinality Columns (> 50 unique values):")  for col, count in high\_cardinality.items():  print(f"{col}: {count} unique values")  print("-" \* 40) |
|  |
| Figure 13: Distribution of Low Cardinality Variables |
| import matplotlib.pyplot as plt  import seaborn as sns  # Define low\_cardinality\_columns from the dictionary keys  low\_cardinality\_columns = list(low\_cardinality.keys())  # Visualize the distribution of low cardinality columns in a 2x2 grid  plt.figure(figsize=(16, 12))  for i, col in enumerate(low\_cardinality\_columns, 1):  plt.subplot(2, 2, i) # Create a 2x2 grid  ax = sns.countplot(  data=data\_dropped\_cols,  x=col,  palette="viridis",  order=data\_dropped\_cols[col].value\_counts().index  )    # Add data labels to bars  for p in ax.patches: # Iterate over bars  ax.annotate(  f"{int(p.get\_height())}", # Bar value as integer  (p.get\_x() + p.get\_width() / 2, p.get\_height()), # X and Y position  ha="center", va="bottom", fontsize=10 # Alignment and font size  )    plt.title(f'Distribution of {col.capitalize()}', fontsize=14)  plt.xlabel(col.capitalize(), fontsize=12)  plt.ylabel('Count', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right')  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.suptitle('Low Cardinality Variables - Distribution', fontsize=16, y=1.02)  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\low\_cardinality\_distribution\_with\_labels.png')  plt.show() |
|  |
| Figure 14: Average Sales by Ship Mode, Segment, Region, and Category |
| plt.figure(figsize=(16, 12))  for i, col in enumerate(low\_cardinality\_columns, 1):  plt.subplot(2, 2, i) # Create a 2x2 grid  ax = sns.barplot(  data=data\_dropped\_cols,  x=col,  y='sales',  palette="viridis",  order=data\_dropped\_cols.groupby(col, observed=True)['sales'].mean().sort\_values(ascending=False).index,  hue=None # Explicitly set hue to avoid warnings  )  # Add data labels to bars  for p in ax.patches:  ax.annotate(  f"{p.get\_height():.2f}", # Bar value with 2 decimals  (p.get\_x() + p.get\_width() / 2, p.get\_height()), # X, Y position  ha="center", va="bottom", fontsize=10  )  plt.title(f'Average Sales by {col.capitalize()}', fontsize=14)  plt.xlabel(col.capitalize(), fontsize=12)  plt.ylabel('Average Sales', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right')  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.suptitle('Low Cardinality Variables - Sales Analysis', fontsize=16, y=1.02)  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\low\_cardinality\_sales\_analysis\_with\_labels.png')  plt.show() |
|  |
| Figure 15: Distribution of Sub-categories and States |
| import matplotlib.pyplot as plt  import seaborn as sns  # Plot for 'sub-category' column  plt.figure(figsize=(15, 8)) # Adjust figure size for clarity  ax = sns.countplot(  data=data\_dropped\_cols,  x='sub-category',  palette="viridis",  order=data\_dropped\_cols['sub-category'].value\_counts().index  )  plt.title('Distribution of Sub-category (Full)', fontsize=16)  plt.xlabel('Sub-category', fontsize=12)  plt.ylabel('Count', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right') # Rotate labels for better readability  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels  for p in ax.patches:  ax.annotate(  f'{int(p.get\_height())}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 5),  ha='center', va='bottom', fontsize=9, color='black'  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\sub\_category\_full\_distribution.png')  plt.show()  # Specific plot for 'state' column with non-overlapping labels  plt.figure(figsize=(18, 8)) # Increase figure size for better clarity  ax = sns.countplot(  data=data\_dropped\_cols,  x='state',  palette="viridis",  order=data\_dropped\_cols['state'].value\_counts().index  )  plt.title('Distribution of State (Full)', fontsize=16)  plt.xlabel('State', fontsize=12)  plt.ylabel('Count', fontsize=12)  plt.xticks(rotation=90, fontsize=10, ha='center') # Rotate labels vertically  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels to the side of the bars  for p in ax.patches:  ax.annotate(  f'{int(p.get\_height())}', # Convert bar height to an integer  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 50), # Adjust position  ha='center', va='bottom', fontsize=9, color='black' # Font and color  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\state\_full\_distribution\_adjusted\_labels.png')  plt.show() |
|  |
| Figure 16: Average Sales by Sub-category and State |
| import matplotlib.pyplot as plt  import seaborn as sns  # Sales Analysis for Sub-category  plt.figure(figsize=(15, 8)) # Adjust figure size for clarity  sub\_category\_sales = data\_dropped\_cols.groupby('sub-category')['sales'].sum().sort\_values(ascending=False)  ax = sns.barplot(  x=sub\_category\_sales.index,  y=sub\_category\_sales.values,  palette="viridis"  )  plt.title('Total Sales by Sub-category', fontsize=16)  plt.xlabel('Sub-category', fontsize=12)  plt.ylabel('Total Sales', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right') # Rotate labels for readability  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}', # Display sales values with two decimal places  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 100),  ha='center', va='bottom', fontsize=9, color='black'  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\sub\_category\_sales\_analysis.png')  plt.show()  # Sales Analysis for State  plt.figure(figsize=(18, 8)) # Larger figure size for many states  state\_sales = data\_dropped\_cols.groupby('state')['sales'].sum().sort\_values(ascending=False)  ax = sns.barplot(  x=state\_sales.index,  y=state\_sales.values,  palette="viridis"  )  plt.title('Total Sales by State', fontsize=16)  plt.xlabel('State', fontsize=12)  plt.ylabel('Total Sales', fontsize=12)  plt.xticks(rotation=90, fontsize=10, ha='center') # Rotate labels vertically  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 100),  ha='center', va='bottom', fontsize=9, color='black'  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\state\_sales\_analysis.png')  plt.show()  # Average Sales Analysis for Sub-category  plt.figure(figsize=(15, 8))  sub\_category\_avg\_sales = data\_dropped\_cols.groupby('sub-category')['sales'].mean().sort\_values(ascending=False)  ax = sns.barplot(  x=sub\_category\_avg\_sales.index,  y=sub\_category\_avg\_sales.values,  palette="viridis"  )  plt.title('Average Sales by Sub-category', fontsize=16)  plt.xlabel('Sub-category', fontsize=12)  plt.ylabel('Average Sales', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right')  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 10),  ha='center', va='bottom', fontsize=9, color='black'  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\sub\_category\_avg\_sales\_analysis.png')  plt.show()  # Average Sales Analysis for State  plt.figure(figsize=(18, 8))  state\_avg\_sales = data\_dropped\_cols.groupby('state')['sales'].mean().sort\_values(ascending=False)  ax = sns.barplot(  x=state\_avg\_sales.index,  y=state\_avg\_sales.values,  palette="viridis"  )  plt.title('Average Sales by State', fontsize=16)  plt.xlabel('State', fontsize=12)  plt.ylabel('Average Sales', fontsize=12)  plt.xticks(rotation=90, fontsize=10, ha='center')  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 10),  ha='center', va='bottom', fontsize=9, color='black'  )  # Save the chart  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\state\_avg\_sales\_analysis.png')  plt.show() |
|  |
| Figure 17; Distribution and Average Sales by City |
| import matplotlib.pyplot as plt  import seaborn as sns  import pandas as pd  # Group cities into "Top N Cities" and "Others" based on total sales  top\_n = 10 # Define the number of top cities to display  city\_sales = data\_dropped\_cols.groupby('city')['sales'].sum().sort\_values(ascending=False)  top\_cities = city\_sales.head(top\_n)  others = pd.Series(city\_sales[top\_n:].sum(), index=['Others'])  # Combine Top N Cities and Others  grouped\_city\_sales = pd.concat([top\_cities, others])  # Total Sales by Grouped Cities  plt.figure(figsize=(12, 8))  ax = sns.barplot(  x=grouped\_city\_sales.index,  y=grouped\_city\_sales.values,  palette="viridis"  )  plt.title(f'Total Sales: Top {top\_n} Cities and Others', fontsize=16)  plt.xlabel('City', fontsize=12)  plt.ylabel('Total Sales', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right')  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 10000),  ha='center', va='bottom', fontsize=9, color='black'  )  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\grouped\_city\_total\_sales.png')  plt.show()  # Average Sales for Grouped Cities  city\_avg\_sales = data\_dropped\_cols.groupby('city')['sales'].mean().sort\_values(ascending=False)  top\_cities\_avg = city\_avg\_sales.head(top\_n)  others\_avg = pd.Series(city\_avg\_sales[top\_n:].mean(), index=['Others'])  # Combine Top N Cities and Others for Average Sales  grouped\_city\_avg\_sales = pd.concat([top\_cities\_avg, others\_avg])  plt.figure(figsize=(12, 8))  ax = sns.barplot(  x=grouped\_city\_avg\_sales.index,  y=grouped\_city\_avg\_sales.values,  palette="viridis"  )  plt.title(f'Average Sales: Top {top\_n} Cities and Others', fontsize=16)  plt.xlabel('City', fontsize=12)  plt.ylabel('Average Sales', fontsize=12)  plt.xticks(rotation=45, fontsize=10, ha='right')  # Add data labels  for p in ax.patches:  ax.annotate(  f'{p.get\_height():.2f}',  (p.get\_x() + p.get\_width() / 2, p.get\_height() + 500),  ha='center', va='bottom', fontsize=9, color='black'  )  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.savefig(r'C:\Users\ashag\OneDrive\Desktop\Machine Learning\grouped\_city\_avg\_sales.png')  plt.show() |
|  |
| Figure 18; Correlation Heatmap |
| # Automatically select all numerical columns in the dataset  numerical\_columns = data\_dropped\_cols.select\_dtypes(include=np.number).columns.tolist()  # Calculate correlation matrix for all numerical columns  correlation\_matrix = data\_dropped\_cols[numerical\_columns].corr()  # Plot the heatmap with the 'viridis' color palette similar to your previous visualizations  plt.figure(figsize=(10, 8))  sns.heatmap(correlation\_matrix, annot=True, cmap='viridis', fmt='.2f', square=True, linewidths=0.5)  plt.title('Correlation Matrix of All Numerical Variables', fontsize=16)  plt.tight\_layout()  plt.show() |

## **7.4 Handling outliers**

|  |
| --- |
| Figure 19; Outlier Count for Numerical Columns |
| # Automatically select numerical columns in the dataset  numerical\_columns = data\_dropped\_cols.select\_dtypes(include=np.number).columns.tolist()  # Function to detect and count outliers for all numerical columns  def detect\_outliers(data, columns):  outlier\_counts = {}  for col in columns:  Q1 = data[col].quantile(0.25)  Q3 = data[col].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR    # Identify outliers  outliers = data[(data[col] < lower\_bound) | (data[col] > upper\_bound)]  outlier\_counts[col] = outliers.shape[0]    return outlier\_counts  # Detect and count outliers for all numerical columns  outlier\_counts = detect\_outliers(data\_dropped\_cols, numerical\_columns)  print("Outlier Counts for Each Numerical Column:")  for col, count in outlier\_counts.items():  print(f"{col}: {count} outliers") |
|  |
| Figure 20: Boxplot before addressing outliers |
| import matplotlib.pyplot as plt  import seaborn as sns  # Function to plot boxplots with outliers marked in red for all numerical columns  def plot\_outliers(data, columns):  plt.figure(figsize=(16, 12))  for i, col in enumerate(columns, 1):  plt.subplot(3, 3, i) # Create a 3x3 grid for subplots (adjust if needed)  Q1 = data[col].quantile(0.25)  Q3 = data[col].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR    # Mark outliers in red  sns.boxplot(  x=data[col],  palette="viridis",  flierprops=dict(marker='o', color='red', markersize=5)  )  plt.axvline(lower\_bound, color='blue', linestyle='--', label='Lower Bound')  plt.axvline(upper\_bound, color='green', linestyle='--', label='Upper Bound')  plt.title(f"Boxplot of {col} (Outliers in Red)", fontsize=12)  plt.xlabel(col.capitalize(), fontsize=10)  plt.legend()  plt.tight\_layout()  plt.suptitle('Boxplots of All Numerical Variables', fontsize=16, y=1.02)  plt.show()  # Plot outliers with boxplots for all numerical columns  plot\_outliers(data\_dropped\_cols, numerical\_columns) |
|  |
| Figure 21; Outlier Handling and Log Transformation Results |
| import numpy as np  import pandas as pd  # Function to detect outliers based on IQR  def detect\_outliers(data, columns):  outlier\_counts = {}  for col in columns:  Q1 = data[col].quantile(0.25)  Q3 = data[col].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR    # Identify outliers  outliers = data[(data[col] < lower\_bound) | (data[col] > upper\_bound)]  outlier\_counts[col] = outliers.shape[0]    return outlier\_counts  # Assuming 'data\_dropped\_cols' is your dataset  # Step 1: Apply log transformation for 'sales' and 'profit'  # Handle zero or negative values by replacing them with a small positive number  data\_dropped\_cols['sales'] = np.where(data\_dropped\_cols['sales'] <= 0, 0.01, data\_dropped\_cols['sales'])  data\_dropped\_cols['profit'] = np.where(data\_dropped\_cols['profit'] <= 0, 0.01, data\_dropped\_cols['profit'])  # Apply log transformation  data\_dropped\_cols['log\_sales'] = np.log1p(data\_dropped\_cols['sales']) # Safely handles zero values  data\_dropped\_cols['log\_profit'] = np.log1p(data\_dropped\_cols['profit']) # Safely handles zero values  print("\nLog transformation applied to 'sales' and 'profit'.")  # Step 2: Handle outliers for numerical columns (excluding 'log\_sales' and 'log\_profit')  numerical\_columns = data\_dropped\_cols.select\_dtypes(include=np.number).columns.tolist()  # Replace outliers with mean for columns that are not 'log\_sales' or 'log\_profit'  for col in numerical\_columns:  if col not in ['log\_sales', 'log\_profit']: # Skip already transformed columns  Q1 = data\_dropped\_cols[col].quantile(0.25)  Q3 = data\_dropped\_cols[col].quantile(0.75)  IQR = Q3 - Q1  lower\_limit = Q1 - 1.5 \* IQR  upper\_limit = Q3 + 1.5 \* IQR    # Replace outliers with column mean  mean\_value = data\_dropped\_cols[col].mean()  data\_dropped\_cols[col] = np.where((data\_dropped\_cols[col] < lower\_limit) | (data\_dropped\_cols[col] > upper\_limit),  mean\_value, data\_dropped\_cols[col])  # Step 3: Detect and count outliers after handling  outlier\_counts\_after = detect\_outliers(data\_dropped\_cols, numerical\_columns)  print("\nOutlier Counts After Handling:")  for col, count in outlier\_counts\_after.items():  print(f"{col}: {count} outliers")  # Step 4: Apply IQR method specifically for 'log\_sales' and 'log\_profit'  log\_columns = ['log\_sales', 'log\_profit']  for col in log\_columns:  Q1 = data\_dropped\_cols[col].quantile(0.25)  Q3 = data\_dropped\_cols[col].quantile(0.75)  IQR = Q3 - Q1  lower\_limit = Q1 - 1.5 \* IQR  upper\_limit = Q3 + 1.5 \* IQR    # Replace outliers with the column's mean  mean\_value = data\_dropped\_cols[col].mean()  data\_dropped\_cols[col] = np.where((data\_dropped\_cols[col] < lower\_limit) | (data\_dropped\_cols[col] > upper\_limit),  mean\_value, data\_dropped\_cols[col])  # Step 5: Detect and count outliers specifically for 'log\_sales' and 'log\_profit'  outlier\_counts\_after\_log = detect\_outliers(data\_dropped\_cols, log\_columns)  print("\nOutlier Counts After Handling 'log\_sales' and 'log\_profit':")  for col, count in outlier\_counts\_after\_log.items():  print(f"{col}: {count} outliers") |
|  |
| Figure 22: Boxplot after handling outliers |
| import matplotlib.pyplot as plt  import seaborn as sns  import numpy as np  # Function to plot boxplots for all numerical columns  def plot\_outliers(data, columns):  plt.figure(figsize=(16, 12))  for i, col in enumerate(columns, 1):  plt.subplot(3, 3, i) # Create a 3x3 grid for subplots (adjust if needed)    # Calculate the IQR for the column  Q1 = data[col].quantile(0.25)  Q3 = data[col].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR    # Create boxplot with general color instead of palette  sns.boxplot(  x=data[col],  color='skyblue', # Use a general color like 'skyblue' instead of palette  flierprops=dict(marker='o', color='red', markersize=5) # Outliers in red  )  # Draw lines for the lower and upper bounds  plt.axvline(lower\_bound, color='blue', linestyle='--', label='Lower Bound')  plt.axvline(upper\_bound, color='green', linestyle='--', label='Upper Bound')  # Title and labels  plt.title(f"Boxplot of {col} ", fontsize=12)  plt.xlabel(col.capitalize(), fontsize=10)  plt.legend()  plt.tight\_layout()  plt.suptitle('Boxplots of All Numerical Variables', fontsize=16, y=1.02)  plt.show()  # Assuming `data\_dropped\_cols` is your data frame and `numerical\_columns` contains column names  plot\_outliers(data\_dropped\_cols, numerical\_columns) |
|  |
| Figure 23; Distribution of Original and Log-Transformed Sales and Profit |
| import seaborn as sns  import matplotlib.pyplot as plt  # Visualize the distribution of 'sales' and 'log\_sales'  plt.figure(figsize=(14, 6))  # Original 'sales' distribution  plt.subplot(1, 2, 1)  sns.histplot(data\_dropped\_cols['sales'], kde=True, color='blue', bins=30)  plt.title("Distribution of Original Sales", fontsize=14)  plt.xlabel("Sales", fontsize=12)  plt.ylabel("Frequency", fontsize=12)  # Log-transformed 'sales' distribution  plt.subplot(1, 2, 2)  sns.histplot(data\_dropped\_cols['log\_sales'], kde=True, color='green', bins=30)  plt.title("Distribution of Log-Transformed Sales", fontsize=14)  plt.xlabel("Log(Sales)", fontsize=12)  plt.ylabel("Frequency", fontsize=12)  plt.tight\_layout()  plt.show()  # Visualize the distribution of 'profit' and 'log\_profit'  plt.figure(figsize=(14, 6))  # Original 'profit' distribution  plt.subplot(1, 2, 1)  sns.histplot(data\_dropped\_cols['profit'], kde=True, color='orange', bins=30)  plt.title("Distribution of Original Profit", fontsize=14)  plt.xlabel("Profit", fontsize=12)  plt.ylabel("Frequency", fontsize=12)  # Log-transformed 'profit' distribution  plt.subplot(1, 2, 2)  sns.histplot(data\_dropped\_cols['log\_profit'], kde=True, color='purple', bins=30)  plt.title("Distribution of Log-Transformed Profit", fontsize=14)  plt.xlabel("Log(Profit)", fontsize=12)  plt.ylabel("Frequency", fontsize=12)  plt.tight\_layout()  plt.show() |

## **7.5 Feature Engineering**

|  |
| --- |
| Figure 24; Feature Engineering Table |
| import numpy as np  import pandas as pd  # Function for refined feature engineering  def create\_refined\_features(df):  # 1. Log Transformation for Sales and Profit  df['log\_sales'] = np.log1p(df['sales']) # Apply log transformation to sales column  df['log\_profit'] = np.log1p(df['profit']) # Apply log transformation to profit column    # 2. Profit Margin: Profitability of each sale  df['profit\_margin'] = (df['profit'] / df['sales']).replace([np.inf, -np.inf], 0).fillna(0)    # 3. Discount Impact: Interaction of discount and quantity  df['discount\_impact'] = df['discount'] \* df['quantity']    # 4. Discount Efficiency: Profit generated per unit of discount  df['discount\_efficiency'] = (df['profit'] / df['discount']).replace([np.inf, -np.inf], 0).fillna(0)    return df  # Assuming `data\_dropped\_cols` is your input DataFrame with initial features  data\_refined = create\_refined\_features(data\_dropped\_cols)  # Display the structure of the refined dataset  print("Structure of the refined dataset:")  print(data\_refined.info())  # Save the refined dataset to a CSV file for further use  data\_refined.to\_csv('C:/Users/ashag/OneDrive/Desktop/Machine Learning/refined\_dataset.csv', index=False) |

## **7.6 Feature Selection**

|  |
| --- |
| Figure 25; Portion of code used for Feature Selection |
| # Keep both 'profit' and 'log\_profit' in the dataset, and include 'log\_sales'  # Select features for modeling (without 'shipping\_category' and 'order\_to\_ship\_duration')  selected\_features = [  'sales', 'log\_sales', 'profit', 'log\_profit', 'profit\_margin', 'discount', 'quantity',  'discount\_impact', 'discount\_efficiency', 'region', 'segment', 'category'  ]  # Retain only the selected features for modeling  data\_modeling = data\_refined[selected\_features]  # Check the structure of the selected dataset  print("Structure of the selected dataset:")  print(data\_modeling.info())  # Save the selected dataset for further analysis  data\_modeling.to\_csv('C:/Users/ashag/OneDrive/Desktop/Machine Learning/selected\_features\_dataset.csv', index=False) |

## **7.7 Encoding**

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  import pandas as pd  # Copy the dataset to avoid overwriting  encoded\_data = data\_modeling.copy()  decoded\_data = data\_modeling.copy() # Keep a copy for decoded values  # Label Encoding for low cardinality variables  label\_cols = ['region', 'segment']  label\_encoders = {} # Store encoders for reverse mapping  for col in label\_cols:  le = LabelEncoder()  encoded\_data[col] = le.fit\_transform(encoded\_data[col]) # Apply label encoding  label\_encoders[col] = le # Store the encoder for reverse mapping  # One-Hot Encoding for medium cardinality variable  encoded\_data = pd.get\_dummies(encoded\_data, columns=['category'], drop\_first=True)  # Save both datasets to the specified path  encoded\_data.to\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv', index=False)  decoded\_data.to\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/decoded\_features\_dataset.csv', index=False)  # Display structure of the datasets  print("Encoded Dataset Structure:")  print(encoded\_data.info())  print("Decoded Dataset Structure:")  print(decoded\_data.info()) |

## **7.8 Visualizing Relationships**

|  |
| --- |
| Figure 26; Correlation Heatmap of Numerical Features After Encoding |
| import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  # Load the encoded dataset  encoded\_data = pd.read\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv')  # Select only numerical columns for the correlation matrix  numerical\_columns = encoded\_data.select\_dtypes(include=['float64', 'int64']).columns  corr\_matrix = encoded\_data[numerical\_columns].corr()  # Plot the correlation heatmap  plt.figure(figsize=(10, 8))  sns.heatmap(corr\_matrix, annot=True, cmap=sns.color\_palette("Spectral", as\_cmap=True), fmt='.2f', cbar=True)  plt.title("Correlation Heatmap of Numerical Features", fontsize=14)  # Save the heatmap  plt.savefig(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/correlation\_heatmap.png')  # Show the heatmap  plt.show() |
|  |
| Figure 27; Total Sales by Region |
| # Regional Sales Performance (Bar Chart with Data Labels)  plt.figure(figsize=(8, 6))  regional\_sales = decoded\_data.groupby('region', observed=True)['sales'].sum()  # Create bar chart  ax = regional\_sales.plot(kind='bar', color=sns.color\_palette("Spectral", len(regional\_sales)))  # Add data labels  for i, value in enumerate(regional\_sales):  ax.text(i, value + 100, f'{value:,.0f}', ha='center', fontsize=10)  plt.title("Regional Sales Performance", fontsize=14)  plt.xlabel("Region", fontsize=12)  plt.ylabel("Total Sales", fontsize=12)  plt.xticks(rotation=0)  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.savefig(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/regional\_sales\_performance\_labels.png')  plt.show() |
|  |
| Figure 28: Sales Performance Across Customer Segments and Product Categories |
| # Corrected 2D Stacked Bar Chart with Labels  import matplotlib.pyplot as plt  import seaborn as sns  # Fix observed=True to avoid the warning  category\_segment\_sales = decoded\_data.groupby(['segment', 'category'], observed=True)['sales'].sum().unstack()  # Create the stacked bar chart  ax = category\_segment\_sales.plot(  kind='bar',  stacked=True,  figsize=(12, 6),  color=sns.color\_palette("Spectral", len(category\_segment\_sales.columns))  )  # Add labels on top of the bars  for container in ax.containers:  ax.bar\_label(container, fmt='%.0f', label\_type='center', fontsize=9, padding=3)  # Customize chart aesthetics  plt.title("Category and Segment Performance", fontsize=14)  plt.xlabel("Segment", fontsize=12)  plt.ylabel("Total Sales", fontsize=12)  plt.xticks(rotation=0)  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.legend(title="Categories", loc='upper right', fontsize=10)  plt.savefig(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/category\_segment\_performance\_stacked\_bar.png')  plt.show() |
|  |
| Figure 29: Sales Trend Over Time with Peaks, Troughs, and Average Sales |
| import matplotlib.pyplot as plt  import pandas as pd  decoded\_data = pd.DataFrame({  'sales': [100, 200, 150, 300, 250, 400, 350, 500, 450, 600] # Example data  })  # Check for missing data  decoded\_data = decoded\_data.dropna(subset=['sales'])  # Enhanced Line Chart with Data Labels and Additional Visual Features  plt.figure(figsize=(14, 7))  sales = decoded\_data['sales']  # Plot the sales trend  plt.plot(sales, color='purple', linewidth=2.5, label="Sales", zorder=3)  # Annotate peaks and troughs  max\_point = sales.idxmax()  min\_point = sales.idxmin()  plt.scatter(max\_point, sales[max\_point], color='red', label='Peak', zorder=5)  plt.scatter(min\_point, sales[min\_point], color='blue', label='Trough', zorder=5)  # Add labels for peaks and troughs with dynamic positioning  plt.text(max\_point, sales[max\_point] + 50, f'{sales[max\_point]:,.0f}', color='red', fontsize=12, ha='center', fontweight='bold')  plt.text(min\_point, sales[min\_point] - 50, f'{sales[min\_point]:,.0f}', color='blue', fontsize=12, ha='center', fontweight='bold')  # Add average sales line  avg\_sales = sales.mean()  plt.axhline(y=avg\_sales, color='green', linestyle='--', label=f'Average Sales: {avg\_sales:.2f}', zorder=2)  # Titles and labels  plt.title("Enhanced Sales Trend Over Time with Peaks and Troughs", fontsize=16, fontweight='bold', color='darkblue')  plt.xlabel("Index (Simulated Time Periods)", fontsize=12)  plt.ylabel("Sales", fontsize=12)  # Customize ticks  plt.xticks(rotation=45, fontsize=10)  plt.yticks(fontsize=10)  # Add gridlines  plt.grid(axis='both', linestyle='--', alpha=0.5)  # Add legend  plt.legend(loc='upper left', fontsize=12)  # Add a subtitle  plt.suptitle("Tracking Peaks, Troughs, and Trends in Sales", fontsize=14, fontweight='bold', color='black', y=0.95)  # Save the enhanced chart  plt.savefig(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/sales\_trend\_enhanced\_with\_avg.png')  plt.show() |
|  |
| Figure 30: Sales by Discount Across Regions |
| import plotly.express as px  #Create a single region column  def get\_region(row):  if row['East']: return 'region\_East'  elif row['South']: return 'region\_South'  elif row['West']: return 'region\_West'  else: return 'Unknown'  # Create single columns for region if needed  data['region'] = data.apply(get\_region, axis=1)  # Now create the plot  area\_data = data.groupby(['discount', 'region'])['sales'].sum().reset\_index()  # Create Stacked Area Chart  fig = px.area(  area\_data,  x='discount',  y='sales',  color='region',  title="Sales by Discount Across Regions",  labels={'discount': 'Discount (%)', 'sales': 'Sales (in $)', 'region': 'Region'}  )  fig.update\_layout(margin=dict(t=50, l=25, r=25, b=25))  fig.show() |

**8.0 Dataset Split for Modelling**

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  # Load the encoded dataset (assuming it's already loaded in `encoded\_data`)  # Define feature columns (exclude 'sales' from the features)  feature\_columns = [col for col in encoded\_data.columns if col not in ['log\_sales']] # Exclude 'sales' as it's the target  # Define X (features) and y (target variable)  X = encoded\_data[feature\_columns] # Features  y = encoded\_data['log\_sales'] # Target variable (sales)  # Apply feature scaling (Standardization)  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X) # Scale the features  # Perform the train-test split (80% for training, 20% for testing)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X\_scaled, y, test\_size=0.2, random\_state=42  )  # Output the shape of the splits for verification  print(f"Training Features: {X\_train.shape}, Training Target: {y\_train.shape}")  print(f"Testing Features: {X\_test.shape}, Testing Target: {y\_test.shape}") |

**8.1: Target Variable Distribution in Training and Testing Sets**

|  |
| --- |
| import matplotlib.pyplot as plt  import seaborn as sns  # Plot the distribution of the target variable in training and testing sets  plt.figure(figsize=(10, 6))  sns.histplot(y\_train, color='blue', kde=True, label='Training Set', stat="density")  sns.histplot(y\_test, color='orange', kde=True, label='Testing Set', stat="density")  plt.title('Distribution of sales in Training and Testing Sets', fontsize=14)  plt.xlabel('sales', fontsize=12)  plt.ylabel('Density', fontsize=12)  plt.legend()  plt.grid(alpha=0.3)  plt.tight\_layout()  plt.show() |

**8.2: Feature Correlation in Training and Test Sets**

|  |
| --- |
| import matplotlib.pyplot as plt  import seaborn as sns  import pandas as pd  # Verify columns in the training dataset  print("Available columns in X\_train:", X.columns) # This will work, as X is a DataFrame  # Apply scaling to the features and convert back to a DataFrame  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X) # This gives you a NumPy array  X\_train\_scaled = X\_scaled[:len(X\_train)] # Get the scaled training data from the array  # Convert the scaled data back to a DataFrame  X\_train\_df = pd.DataFrame(X\_train\_scaled, columns=X.columns)  # Update the list of important columns to match those in X\_train\_df  important\_columns = ['discount', 'quantity', 'profit\_margin', 'discount\_efficiency', 'discount\_impact'] # Removed 'order\_to\_ship\_duration'  # Define a colormap for the histograms  colormap = plt.cm.viridis # Viridis color map  original\_color = colormap(0.2) # Light shade  training\_color = colormap(0.8) # Dark shade  # Set up a subplot grid (2 rows: Original vs Training, 5 columns)  fig, axes = plt.subplots(2, len(important\_columns), figsize=(18, 8))  # Loop through the selected columns to create histograms  for i, column in enumerate(important\_columns):  if column in X\_train\_df.columns: # Check if the column exists in X\_train\_df  # Original data (top row)  X[column].hist(ax=axes[0, i], color=original\_color, alpha=0.7, bins=30, density=True)  axes[0, i].set\_title(f"Original: {column}", fontsize=10)  axes[0, i].set\_xlabel(column, fontsize=9)  axes[0, i].set\_ylabel('Density', fontsize=9)  # Training data (bottom row)  X\_train\_df[column].hist(ax=axes[1, i], color=training\_color, alpha=0.7, bins=30, density=True)  axes[1, i].set\_title(f"Training: {column}", fontsize=10)  axes[1, i].set\_xlabel(column, fontsize=9)  axes[1, i].set\_ylabel('Density', fontsize=9)  # Adjust layout to avoid overlaps and improve readability  plt.tight\_layout()  plt.suptitle('Comparison of Original and Training Set Distributions', y=1.02, fontsize=16)  plt.savefig(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/comparison\_distributions\_with\_viridis.png')  plt.show() |

**9.0: Modelling**

**9.1: Random Forest**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  import numpy as np  from sklearn.ensemble import RandomForestRegressor  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  import matplotlib.pyplot as plt  # Load the encoded dataset  encoded\_data = pd.read\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv')  # Define feature columns (exclude 'sales', 'log\_sales', and 'log\_profit')  feature\_columns = [col for col in encoded\_data.columns if col not in ['sales', 'log\_sales', 'log\_profit', 'profit']]  # Define X (features) and y (target variable)  X = encoded\_data[feature\_columns]  y = encoded\_data['log\_sales']  # Apply feature scaling  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # Perform train-test split (80% training, 20% testing)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # Initialize the Random Forest model  rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)  # Train the Random Forest model  rf\_model.fit(X\_train, y\_train)  # Predictions on the test set  y\_pred\_rf = rf\_model.predict(X\_test)  # Evaluate the model using R², RMSE, and MAE without rounding off  print(f"Random Forest R²: {r2\_score(y\_test, y\_pred\_rf)}")  print(f"Random Forest RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf))}")  print(f"Random Forest MAE: {mean\_absolute\_error(y\_test, y\_pred\_rf)}")  # Visualize Actual vs Predicted Sales (Random Forest)  plt.figure(figsize=(8, 6))  plt.scatter(y\_test, y\_pred\_rf, color='teal', alpha=0.6, edgecolor='k')  plt.title("Random Forest Regression: Actual vs Predicted Sales", fontsize=14)  plt.xlabel("Actual Sales", fontsize=12)  plt.ylabel("Predicted Sales", fontsize=12)  plt.grid(True, linestyle='--', alpha=0.7)  plt.show()  # Feature importance for Random Forest  importances = rf\_model.feature\_importances\_  indices = np.argsort(importances)[::-1]  plt.figure(figsize=(10, 6))  plt.title("Feature Importance (Random Forest)", fontsize=14)  plt.barh(range(len(indices)), importances[indices], color='skyblue', align="center")  plt.yticks(range(len(indices)), [feature\_columns[i] for i in indices], fontsize=10)  plt.xlabel("Feature Importance", fontsize=12)  plt.ylabel("Features", fontsize=12)  plt.grid(axis='x', linestyle='--', alpha=0.7)  plt.show() |

**Random Forest**

|  |
| --- |
| # Import additional libraries  import seaborn as sns  from sklearn.ensemble import GradientBoostingRegressor  import xgboost as xgb  #=======================================================  # 1. SALES PREDICTION  #=======================================================  def train\_and\_evaluate\_models(X, y):  # Split and scale data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)    # Define models  models = {  'Random Forest': RandomForestRegressor(n\_estimators=200, random\_state=42),  'XGBoost': xgb.XGBRegressor(n\_estimators=200, random\_state=42),  'Gradient Boosting': GradientBoostingRegressor(n\_estimators=200, random\_state=42)  }    results = {}  print("\n1. SALES PREDICTION RESULTS:")  print("----------------------------")  for name, model in models.items():  # Train model  model.fit(X\_train\_scaled, y\_train)  predictions = model.predict(X\_test\_scaled)    # Calculate metrics  rmse = np.sqrt(mean\_squared\_error(y\_test, predictions))  r2 = r2\_score(y\_test, predictions)  mae = mean\_absolute\_error(y\_test, predictions)    results[name] = {  'model': model,  'rmse': rmse,  'r2': r2,  'mae': mae,  'predictions': predictions  }    print(f"\n{name}:")  print(f"- R² Score: {r2:.4f}")  print(f"- RMSE: {rmse:.4f}")  print(f"- MAE: {mae:.4f}")    return results, X\_test\_scaled, y\_test  #=======================================================  # 2. SALES DRIVERS ANALYSIS  #=======================================================  def analyze\_sales\_drivers(df, feature\_columns, rf\_model):  print("\n2. KEY SALES DRIVERS:")  print("----------------------------")    # Feature importance analysis  feature\_importance = pd.DataFrame({  'feature': feature\_columns,  'importance': rf\_model.feature\_importances\_  }).sort\_values('importance', ascending=False)    print("\nTop 5 Sales Drivers:")  for idx, row in feature\_importance.head().iterrows():  print(f"- {row['feature']}: {row['importance']:.4f}")    # Regional analysis  print("\nRegional Performance:")  region\_sales = df.groupby(['region'])['sales'].agg([  'mean', 'count', 'sum'  ]).round(2)  print(region\_sales)    # Category analysis  print("\nCategory Performance:")  category\_sales = df.groupby(['category\_Office Supplies', 'category\_Technology'])['sales'].agg([  'mean', 'count', 'sum'  ]).round(2)  print(category\_sales)    return feature\_importance, region\_sales, category\_sales  #=======================================================  # 3. DISCOUNT STRATEGY ANALYSIS  #=======================================================  def analyze\_discount\_strategy(df):  print("\n3. DISCOUNT STRATEGY ANALYSIS:")  print("----------------------------")    # Create discount ranges  df['discount\_range'] = pd.cut(df['discount'],  bins=[0, 0.1, 0.2, 0.3, 0.4, 1],  labels=['0-10%', '11-20%', '21-30%', '31-40%', '40%+'])    # Analyze by discount range  discount\_analysis = df.groupby('discount\_range').agg({  'sales': ['mean', 'sum'],  'profit': ['mean', 'sum'],  'quantity': 'sum'  }).round(2)    print("\nDiscount Range Analysis:")  print(discount\_analysis)    # Find optimal discount range  optimal\_range = discount\_analysis['profit']['sum'].idxmax()  print(f"\nMost Profitable Discount Range: {optimal\_range}")  print(f"- Average Sales: ${discount\_analysis['sales']['mean'][optimal\_range]:,.2f}")  print(f"- Total Profit: ${discount\_analysis['profit']['sum'][optimal\_range]:,.2f}")    return discount\_analysis  # Execute all analyses  if \_\_name\_\_ == "\_\_main\_\_":  # Load and prepare data  encoded\_data = pd.read\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv')  feature\_columns = [col for col in encoded\_data.columns if col not in ['sales', 'log\_sales', 'log\_profit', 'profit']]  X = encoded\_data[feature\_columns]  y = encoded\_data['log\_sales']    # 1. Sales Prediction  results, X\_test, y\_test = train\_and\_evaluate\_models(X, y)    # 2. Sales Drivers Analysis  rf\_model = results['Random Forest']['model']  feature\_importance, region\_sales, category\_sales = analyze\_sales\_drivers(encoded\_data, feature\_columns, rf\_model)    # 3. Discount Strategy Analysis  discount\_analysis = analyze\_discount\_strategy(encoded\_data)    # Print Final Recommendations  print("\n📊 FINAL BUSINESS RECOMMENDATIONS:")  print("=================================")    print("\n1. Sales Prediction Strategy:")  best\_model = max(results.items(), key=lambda x: x[1]['r2'])[0]  print(f"- Use {best\_model} model for sales predictions (R² = {results[best\_model]['r2']:.4f})")    print("\n2. Focus Areas:")  top\_features = feature\_importance['feature'].head(3).tolist()  print(f"- Key sales drivers: {', '.join(top\_features)}")    print("\n3. Discount Strategy:")  optimal\_discount = discount\_analysis['profit']['sum'].idxmax()  print(f"- Optimal discount range: {optimal\_discount}")  print(f"- Expected average sales: ${discount\_analysis['sales']['mean'][optimal\_discount]:,.2f}")  print(f"- Expected total profit: ${discount\_analysis['profit']['sum'][optimal\_discount]:,.2f}") |

**Residual Plot and QQ Plot**

|  |
| --- |
| # Import necessary libraries  import seaborn as sns  import scipy.stats as stats  # Calculate residuals  residuals = y\_test - y\_pred\_rf  # Residual Plot  plt.figure(figsize=(8, 6))  sns.scatterplot(x=y\_pred\_rf, y=residuals, alpha=0.6, color="teal", edgecolor="k")  plt.axhline(0, linestyle='--', color='red', linewidth=1)  plt.title("Residual Plot", fontsize=14)  plt.xlabel("Predicted Values", fontsize=12)  plt.ylabel("Residuals", fontsize=12)  plt.grid(alpha=0.3)  plt.show()  # QQ Plot  plt.figure(figsize=(8, 6))  stats.probplot(residuals, dist="norm", plot=plt)  plt.title("QQ Plot of Residuals", fontsize=14)  plt.grid(alpha=0.3)  plt.show() |

**Cross Validation**

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_score  # Perform 5-fold cross-validation  cv\_scores = cross\_val\_score(rf\_model, X\_scaled, y, cv=5, scoring='r2')  print(f"Cross-Validation R2 Scores: {cv\_scores}")  print(f"Mean Cross-Validation R2 Score: {np.mean(cv\_scores)}")  print(f"Standard Deviation of Cross-Validation Scores: {np.std(cv\_scores)}") |

**Compare Train vs Test Performance**

|  |
| --- |
| # Evaluate the model on the training set  y\_train\_pred = rf\_model.predict(X\_train)  # Calculate training metrics  train\_r2 = r2\_score(y\_train, y\_train\_pred)  train\_rmse = np.sqrt(mean\_squared\_error(y\_train, y\_train\_pred))  train\_mae = mean\_absolute\_error(y\_train, y\_train\_pred)  # Display training performance  print(f"Training R²: {train\_r2}")  print(f"Training RMSE: {train\_rmse}")  print(f"Training MAE: {train\_mae}") |

**Analyze Feature Importance**

|  |
| --- |
| importances = rf\_model.feature\_importances\_  sorted\_indices = np.argsort(importances)[::-1]  for idx in sorted\_indices:  print(f"{feature\_columns[idx]}: {importances[idx]}") |

**Apply Regularization (Hyperparameter Tuning)**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  import numpy as np  from sklearn.ensemble import RandomForestRegressor  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.preprocessing import StandardScaler  # Load the dataset  encoded\_data = pd.read\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv')  # Define refined feature columns excluding 'profit'  refined\_feature\_columns = [col for col in encoded\_data.columns if col not in ['sales', 'log\_sales', 'log\_profit', 'profit']]  # Define X and y  X\_refined = encoded\_data[refined\_feature\_columns]  y = encoded\_data['log\_sales']  # Scale features  scaler = StandardScaler()  X\_scaled\_refined = scaler.fit\_transform(X\_refined)  # Train-test split  X\_train\_refined, X\_test\_refined, y\_train\_refined, y\_test\_refined = train\_test\_split(  X\_scaled\_refined, y, test\_size=0.2, random\_state=42  )  # Define the parameter grid  param\_grid = {  'n\_estimators': [100, 200, 300],  'max\_depth': [10, 20, 30],  'min\_samples\_split': [2, 5, 10],  'min\_samples\_leaf': [1, 2, 4]  }  # Perform GridSearchCV  grid\_search = GridSearchCV(  estimator=RandomForestRegressor(random\_state=42),  param\_grid=param\_grid,  cv=5,  scoring='r2',  n\_jobs=-1,  verbose=2  )  # Fit GridSearchCV  grid\_search.fit(X\_train\_refined, y\_train\_refined)  # Get the best model and parameters  best\_rf = grid\_search.best\_estimator\_  print(f"Best Parameters: {grid\_search.best\_params\_}")  # Evaluate the tuned model on the test set  y\_pred\_best = best\_rf.predict(X\_test\_refined)  print(f"Tuned Model R²: {r2\_score(y\_test\_refined, y\_pred\_best)}")  print(f"Tuned Model RMSE: {np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_best))}")  print(f"Tuned Model MAE: {mean\_absolute\_error(y\_test\_refined, y\_pred\_best)}") |

**Model: XGBoost**

|  |
| --- |
| # Import necessary libraries  import xgboost as xgb  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  import matplotlib.pyplot as plt  import numpy as np  # Initialize XGBoost regressor with default parameters  xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)  # Fit XGBoost model to the training data  xgb\_model.fit(X\_train\_refined, y\_train\_refined)  # Predictions on the test set  y\_pred\_xgb = xgb\_model.predict(X\_test\_refined)  # Evaluate XGBoost model  print("XGBoost Performance:")  print(f"R²: {r2\_score(y\_test\_refined, y\_pred\_xgb)}")  print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_xgb))}")  print(f"MAE: {mean\_absolute\_error(y\_test\_refined, y\_pred\_xgb)}")  # -----------------------------------------------  # VISUALIZATION: ACTUAL VS PREDICTED  # -----------------------------------------------  # Plot Actual vs Predicted for XGBoost  plt.figure(figsize=(8, 6))  plt.scatter(y\_test\_refined, y\_pred\_xgb, color='orange', alpha=0.6, edgecolor='k')  plt.title("XGBoost: Actual vs Predicted Sales", fontsize=14)  plt.xlabel("Actual Sales", fontsize=12)  plt.ylabel("Predicted Sales", fontsize=12)  plt.grid(True, linestyle='--', alpha=0.7)  plt.show()  # -----------------------------------------------  # FEATURE IMPORTANCE FOR XGBOOST  # -----------------------------------------------  # Get feature importances from the XGBoost model  xgb\_importances = xgb\_model.feature\_importances\_  indices\_xgb = np.argsort(xgb\_importances)[::-1]  # Plot feature importance for XGBoost  plt.figure(figsize=(10, 6))  plt.title("Feature Importance (XGBoost)", fontsize=14)  plt.barh(range(len(indices\_xgb)), xgb\_importances[indices\_xgb], color='lightgreen', align="center")  plt.yticks(range(len(indices\_xgb)), [refined\_feature\_columns[i] for i in indices\_xgb], fontsize=10)  plt.xlabel("Feature Importance", fontsize=12)  plt.ylabel("Features", fontsize=12)  plt.grid(axis='x', linestyle='--', alpha=0.7)  plt.show() |

**XGBoost**

|  |
| --- |
| # Import necessary libraries  import xgboost as xgb  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  import matplotlib.pyplot as plt  import numpy as np  import pandas as pd  import seaborn as sns  #=======================================================  # 1. SALES PREDICTION  #=======================================================  def train\_sales\_prediction\_model(X\_train, X\_test, y\_train, y\_test, feature\_columns):  print("\n🎯 OBJECTIVE 1: SALES PREDICTION")    # Initialize XGBoost regressor with tuned parameters  xgb\_model = xgb.XGBRegressor(  objective='reg:squarederror',  n\_estimators=200,  learning\_rate=0.1,  max\_depth=6,  random\_state=42  )    # Fit XGBoost model  xgb\_model.fit(X\_train, y\_train)    # Predictions  y\_pred = xgb\_model.predict(X\_test)    # Evaluate model  print("\nModel Performance Metrics:")  print(f"R²: {r2\_score(y\_test, y\_pred):.4f}")  print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred)):.4f}")  print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred):.4f}")    # Visualization: Actual vs Predicted  plt.figure(figsize=(8, 6))  plt.scatter(y\_test, y\_pred, color='orange', alpha=0.6, edgecolor='k')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2)  plt.title("XGBoost: Actual vs Predicted Sales", fontsize=14)  plt.xlabel("Actual Sales", fontsize=12)  plt.ylabel("Predicted Sales", fontsize=12)  plt.grid(True, linestyle='--', alpha=0.7)  plt.show()    return xgb\_model, y\_pred  #=======================================================  # 2. KEY SALES DRIVERS ANALYSIS  #=======================================================  def analyze\_sales\_drivers(model, feature\_columns, data):  print("\n🔍 OBJECTIVE 2: KEY SALES DRIVERS")    # Get and plot feature importance  importances = model.feature\_importances\_  indices = np.argsort(importances)[::-1]    plt.figure(figsize=(10, 6))  plt.title("Feature Importance Analysis", fontsize=14)  plt.barh(range(len(indices)), importances[indices], color='lightgreen', align="center")  plt.yticks(range(len(indices)), [feature\_columns[i] for i in indices], fontsize=10)  plt.xlabel("Feature Importance", fontsize=12)  plt.ylabel("Features", fontsize=12)  plt.grid(axis='x', linestyle='--', alpha=0.7)  plt.show()    # Top 5 most important features  top\_features = pd.DataFrame({  'Feature': feature\_columns,  'Importance': importances  }).sort\_values('Importance', ascending=False).head()    print("\nTop 5 Sales Drivers:")  for idx, row in top\_features.iterrows():  print(f"- {row['Feature']}: {row['Importance']:.4f}")    return top\_features  #=======================================================  # 3. DISCOUNT STRATEGY OPTIMIZATION  #=======================================================  def analyze\_discount\_strategy(data):  print("\n💰 OBJECTIVE 3: DISCOUNT STRATEGY OPTIMIZATION")    # Create discount ranges  data['discount\_range'] = pd.cut(  data['discount'],  bins=[0, 0.1, 0.2, 0.3, 0.4, 1],  labels=['0-10%', '11-20%', '21-30%', '31-40%', '40%+']  )    # Analyze metrics by discount range  discount\_analysis = data.groupby('discount\_range').agg({  'sales': ['mean', 'count', 'sum'],  'profit': ['mean', 'sum'],  'quantity': 'sum'  }).round(2)    # Plot discount analysis  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))    # Average sales by discount range  discount\_analysis['sales']['mean'].plot(kind='bar', ax=ax1, color='skyblue')  ax1.set\_title('Average Sales by Discount Range')  ax1.set\_xlabel('Discount Range')  ax1.set\_ylabel('Average Sales')  ax1.tick\_params(axis='x', rotation=45)    # Total profit by discount range  discount\_analysis['profit']['sum'].plot(kind='bar', ax=ax2, color='lightgreen')  ax2.set\_title('Total Profit by Discount Range')  ax2.set\_xlabel('Discount Range')  ax2.set\_ylabel('Total Profit')  ax2.tick\_params(axis='x', rotation=45)    plt.tight\_layout()  plt.show()    # Print optimal discount range  optimal\_range = discount\_analysis['profit']['sum'].idxmax()  print("\nDiscount Strategy Insights:")  print(f"- Most profitable discount range: {optimal\_range}")  print(f"- Average sales in this range: ${discount\_analysis['sales']['mean'][optimal\_range]:,.2f}")  print(f"- Total profit in this range: ${discount\_analysis['profit']['sum'][optimal\_range]:,.2f}")    return discount\_analysis  # Execute all analyses  def main():  # Train sales prediction model  model, predictions = train\_sales\_prediction\_model(  X\_train\_refined, X\_test\_refined,  y\_train\_refined, y\_test\_refined,  refined\_feature\_columns  )    # Analyze sales drivers  top\_features = analyze\_sales\_drivers(model, refined\_feature\_columns, X\_test\_refined)    # Analyze discount strategy  discount\_analysis = analyze\_discount\_strategy(encoded\_data)    # Print final recommendations  print("\n📊 KEY BUSINESS RECOMMENDATIONS:")  print("\n1. Sales Prediction:")  print(f"- Model achieved R² score of {r2\_score(y\_test\_refined, predictions):.4f}")  print(f"- Key predictors: {', '.join(top\_features['Feature'].head(3))}")    optimal\_range = discount\_analysis['profit']['sum'].idxmax()  print("\n2. Discount Strategy:")  print(f"- Recommended discount range: {optimal\_range}")  print(f"- Expected average sales: ${discount\_analysis['sales']['mean'][optimal\_range]:,.2f}")  print(f"- Expected total profit: ${discount\_analysis['profit']['sum'][optimal\_range]:,.2f}")  # Run analysis  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Cross-Validation**

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_score  import xgboost as xgb  from sklearn.metrics import make\_scorer, r2\_score  import numpy as np  # Initialize XGBoost regressor with default parameters  xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)  # Define a custom scoring function for R²  r2\_scorer = make\_scorer(r2\_score)  # Perform 5-fold cross-validation  cv\_scores = cross\_val\_score(  estimator=xgb\_model,  X=X\_scaled\_refined, # Use the scaled feature matrix  y=y, # Target variable  cv=5, # Number of folds  scoring=r2\_scorer,  n\_jobs=-1 # Use all available cores  )  # Print cross-validation results  print("XGBoost Cross-Validation Performance:")  print(f"Cross-Validation R² Scores: {cv\_scores}")  print(f"Mean R²: {np.mean(cv\_scores)}")  print(f"Standard Deviation of R²: {np.std(cv\_scores)}") |

**Residual Analysis**

|  |
| --- |
| residuals\_xgb = y\_test\_refined - y\_pred\_xgb  # Residual Plot  plt.figure(figsize=(8, 6))  plt.scatter(y\_pred\_xgb, residuals\_xgb, alpha=0.6, color="orange", edgecolor="k")  plt.axhline(0, linestyle='--', color='red', linewidth=1)  plt.title("Residual Plot (XGBoost)", fontsize=14)  plt.xlabel("Predicted Values", fontsize=12)  plt.ylabel("Residuals", fontsize=12)  plt.grid(alpha=0.3)  plt.show()  # QQ Plot  import scipy.stats as stats  plt.figure(figsize=(8, 6))  stats.probplot(residuals\_xgb, dist="norm", plot=plt)  plt.title("QQ Plot of Residuals (XGBoost)", fontsize=14)  plt.grid(alpha=0.3)  plt.show() |

**Hyperparameter Tuning**

|  |
| --- |
| from sklearn.model\_selection import GridSearchCV  import xgboost as xgb  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  # Define the parameter grid for XGBoost  param\_grid\_xgb = {  'n\_estimators': [100, 200, 300],  'learning\_rate': [0.01, 0.05, 0.1],  'max\_depth': [3, 5, 7],  'min\_child\_weight': [1, 3],  'subsample': [0.8, 1.0],  'colsample\_bytree': [0.8, 1.0]  }  # Perform GridSearchCV with 5-fold cross-validation  xgb\_grid\_search = GridSearchCV(  estimator=xgb.XGBRegressor(objective='reg:squarederror', random\_state=42),  param\_grid=param\_grid\_xgb,  cv=5, # Set to 5 folds for consistency  scoring='r2',  n\_jobs=-1,  verbose=2  )  # Fit GridSearchCV to the training data  xgb\_grid\_search.fit(X\_train\_refined, y\_train\_refined)  # Get the best model and parameters  best\_xgb\_grid = xgb\_grid\_search.best\_estimator\_  print(f"Best Parameters (XGBoost): {xgb\_grid\_search.best\_params\_}")  # Evaluate the tuned XGBoost model on the test set  y\_pred\_best\_xgb\_grid = best\_xgb\_grid.predict(X\_test\_refined)  print(f"Tuned XGBoost R²: {r2\_score(y\_test\_refined, y\_pred\_best\_xgb\_grid)}")  print(f"Tuned XGBoost RMSE: {np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_best\_xgb\_grid))}")  print(f"Tuned XGBoost MAE: {mean\_absolute\_error(y\_test\_refined, y\_pred\_best\_xgb\_grid)}") |

**Ensemble Approach**

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  # Calculate ensemble predictions (average of Random Forest and XGBoost)  y\_pred\_ensemble = (y\_pred\_rf + y\_pred\_xgb) / 2  # Evaluate Ensemble Performance  ensemble\_r2 = r2\_score(y\_test\_refined, y\_pred\_ensemble)  ensemble\_rmse = np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_ensemble))  ensemble\_mae = mean\_absolute\_error(y\_test\_refined, y\_pred\_ensemble)  rf\_r2 = r2\_score(y\_test\_refined, y\_pred\_rf)  rf\_rmse = np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_rf))  rf\_mae = mean\_absolute\_error(y\_test\_refined, y\_pred\_rf)  xgb\_r2 = r2\_score(y\_test\_refined, y\_pred\_xgb)  xgb\_rmse = np.sqrt(mean\_squared\_error(y\_test\_refined, y\_pred\_xgb))  xgb\_mae = mean\_absolute\_error(y\_test\_refined, y\_pred\_xgb)  # Create a comparison DataFrame  comparison = pd.DataFrame({  'Model': ['Random Forest', 'XGBoost', 'Ensemble'],  'R²': [rf\_r2, xgb\_r2, ensemble\_r2],  'RMSE': [rf\_rmse, xgb\_rmse, ensemble\_rmse],  'MAE': [rf\_mae, xgb\_mae, ensemble\_mae]  })  # Display comparison  print("\nModel Comparison:")  print(comparison)  # Optional: Save the comparison as a CSV file  comparison.to\_csv("C:/Users/ashag/OneDrive/Desktop/Machine Learning/model\_comparison.csv", index=False) |

**XGBoost**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  import numpy as np  from sklearn.ensemble import RandomForestRegressor  import xgboost as xgb  from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  import matplotlib.pyplot as plt  import seaborn as sns  # Load and prepare data  encoded\_data = pd.read\_csv(r'C:/Users/ashag/OneDrive/Desktop/Machine Learning/encoded\_features\_dataset.csv')  feature\_columns = [col for col in encoded\_data.columns if col not in ['sales', 'log\_sales', 'log\_profit', 'profit']]  # Prepare features and target  X = encoded\_data[feature\_columns]  y = encoded\_data['log\_sales']  # Split and scale data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # Train Random Forest  rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)  rf\_model.fit(X\_train\_scaled, y\_train)  y\_pred\_rf = rf\_model.predict(X\_test\_scaled)  # Train XGBoost  xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)  xgb\_model.fit(X\_train\_scaled, y\_train)  y\_pred\_xgb = xgb\_model.predict(X\_test\_scaled)  # Calculate ensemble predictions  y\_pred\_ensemble = (y\_pred\_rf + y\_pred\_xgb) / 2  # Calculate metrics for all models  ensemble\_r2 = r2\_score(y\_test, y\_pred\_ensemble)  ensemble\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_ensemble))  ensemble\_mae = mean\_absolute\_error(y\_test, y\_pred\_ensemble)  rf\_r2 = r2\_score(y\_test, y\_pred\_rf)  rf\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf))  rf\_mae = mean\_absolute\_error(y\_test, y\_pred\_rf)  xgb\_r2 = r2\_score(y\_test, y\_pred\_xgb)  xgb\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_xgb))  xgb\_mae = mean\_absolute\_error(y\_test, y\_pred\_xgb)  # Create comparison DataFrame  comparison = pd.DataFrame({  'Model': ['Random Forest', 'XGBoost', 'Ensemble'],  'R²': [rf\_r2, xgb\_r2, ensemble\_r2],  'RMSE': [rf\_rmse, xgb\_rmse, ensemble\_rmse],  'MAE': [rf\_mae, xgb\_mae, ensemble\_mae]  })  # Display comparison  print("\nModel Comparison:")  print(comparison)  # Save comparison  comparison.to\_csv("C:/Users/ashag/OneDrive/Desktop/Machine Learning/model\_comparison.csv", index=False)  # Visualize model performance  plt.figure(figsize=(12, 5))  plt.subplot(1, 2, 1)  sns.barplot(x='Model', y='R²', data=comparison)  plt.title('R² Score by Model')  plt.xticks(rotation=45)  plt.subplot(1, 2, 2)  sns.barplot(x='Model', y='RMSE', data=comparison)  plt.title('RMSE by Model')  plt.xticks(rotation=45)  plt.tight\_layout()  plt.show()  # Feature importance analysis  importances\_rf = rf\_model.feature\_importances\_  importances\_xgb = xgb\_model.feature\_importances\_  combined\_importance = pd.DataFrame({  'Feature': feature\_columns,  'RF\_Importance': importances\_rf,  'XGB\_Importance': importances\_xgb  })  combined\_importance['Avg\_Importance'] = (combined\_importance['RF\_Importance'] +  combined\_importance['XGB\_Importance']) / 2  combined\_importance = combined\_importance.sort\_values('Avg\_Importance', ascending=False)  # Visualize combined feature importance  plt.figure(figsize=(12, 6))  plt.title("Combined Feature Importance Analysis", fontsize=14)  plt.barh(range(len(combined\_importance)), combined\_importance['Avg\_Importance'],  color='lightgreen', align="center")  plt.yticks(range(len(combined\_importance)), combined\_importance['Feature'], fontsize=10)  plt.xlabel("Average Feature Importance", fontsize=12)  plt.grid(axis='x', linestyle='--', alpha=0.7)  plt.show()  # Discount strategy analysis using only test set  test\_data = pd.DataFrame({  'discount': X\_test['discount'],  'sales': np.exp(y\_test), # Convert log sales back to original scale  'predicted\_sales': np.exp(y\_pred\_ensemble),  'profit': encoded\_data.loc[y\_test.index, 'profit']  })  test\_data['discount\_range'] = pd.cut(  test\_data['discount'],  bins=[0, 0.1, 0.2, 0.3, 0.4, 1],  labels=['0-10%', '11-20%', '21-30%', '31-40%', '40%+']  )  discount\_analysis = test\_data.groupby('discount\_range').agg({  'sales': ['mean', 'sum'],  'profit': ['mean', 'sum']  }).round(2)  # Print final recommendations  print("\n📊 FINAL BUSINESS RECOMMENDATIONS:")  best\_model = comparison.loc[comparison['R²'].idxmax(), 'Model']  print(f"\n1. Sales Prediction:")  print(f"- Best model: {best\_model}")  print(f"- R² Score: {comparison.loc[comparison['Model'] == best\_model, 'R²'].values[0]:.4f}")  print(f"\n2. Key Sales Drivers:")  print(f"- Top 3 drivers: {', '.join(combined\_importance['Feature'].head(3))}")  optimal\_range = discount\_analysis['profit']['sum'].idxmax()  print(f"\n3. Discount Strategy:")  print(f"- Recommended discount range: {optimal\_range}")  print(f"- Expected profit: ${discount\_analysis['profit']['sum'][optimal\_range]:,.2f}") |

**Visualize Metrics for Comparison**

|  |
| --- |
| # Bar plot for model comparison  comparison.set\_index('Model', inplace=True)  comparison.plot(kind='bar', figsize=(10, 6), alpha=0.8)  plt.title("Model Comparison", fontsize=14)  plt.xlabel("Model", fontsize=12)  plt.ylabel("Metrics", fontsize=12)  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.xticks(rotation=0)  plt.legend(title="Metrics", fontsize=10)  plt.show() |