HARNESSING MACHINE LEARNING FOR DEMAND FORECASTING IN SUPPLY CHAINS

A PREDICTIVE ANALYTICS APPROACH

Jothiba Chavan

Candidate Number: 245044

Written under the Supervision of Prof. Ozren Despic

BDM200J - MSc Business Analytics Business Project

January 2025

Master of Science (MSc) in Business Analytics

This project is submitted in partial fulfillment of the requirements for the master's in business Analytics at Aston University, dated January 27th, 2025

Declaration

Acknowledgment

I would like to express my deepest gratitude to my dissertation advisor, Prof. Ozren Despic, for his invaluable guidance and expertise throughout this research endeavor. I am sincerely thankful to Aston University for offering the necessary resources and support that made this study possible. I also wish to acknowledge all the module teachers whose shared knowledge greatly contributed to my analysis. My heartfelt thanks go to my colleagues and friends for their thoughtful discussions and collaboration. I am grateful to the open-source Python community for creating the indispensable libraries that served as the backbone of this analysis. Finally, I extend my profound appreciation to my family and friends for their unwavering encouragement and support. This dissertation stands as a reflection of the collective efforts and contributions of everyone mentioned.

Abstract

This study explores the application of advanced machine learning (ML) techniques to improve demand forecasting for order item quantities in supply chain management. Using the "DataCo Smart Supply Chain" dataset, the research identifies critical variables influencing demand, such as product pricing, discounts, and departmental activities. Four ML models—Linear Regression, Decision Trees, Random Forest, and Gradient Boosting—were implemented and evaluated using performance metrics like RMSE and R^2. Among these, Gradient Boosting emerged as the most effective, achieving the lowest RMSE (0.8489) and highest R^2 (0.6589), showcasing superior capability in capturing non-linear trends and complex interactions within the data.

The study highlights Gradient Boosting's effectiveness in optimizing inventory management, minimizing operational inefficiencies, and enhancing decision-making processes. It recommends leveraging enriched datasets, advanced modeling techniques, and real-time analytics for continuous improvement. While emphasizing the significant influence of pricing and discount strategies on demand patterns, the research underscores the potential of ML models to transform modern supply chains, fostering sustainable growth and efficiency. Future work could include expanding datasets, exploring neural networks, and integrating real-time data for further advancements in demand forecasting.

Aston Business School
Birmingham

List of Figures

List of Tables

Table of Contents

CHAPTER ONE : INTRODUCTION

1.Introduction

1.1 Background Information

Supply Chain Management (SCM) is the foundation of modern businesses, encompassing the planning, production, transportation, and distribution of goods. Central to SCM is inventory management, which ensures that the right products are available at the right time and place to meet customer demand. Effective inventory management reduces excess stock, prevents stockouts, and minimizes operational costs, directly impacting customer satisfaction and profitability. A key enabler of inventory management is demand forecasting, which helps businesses predict future demand trends. Accurate forecasting allows companies to allocate resources effectively, avoid waste, and maintain operational efficiency. However, traditional forecasting approaches often struggle to meet the complexities of dynamic and data-driven modern supply chains.

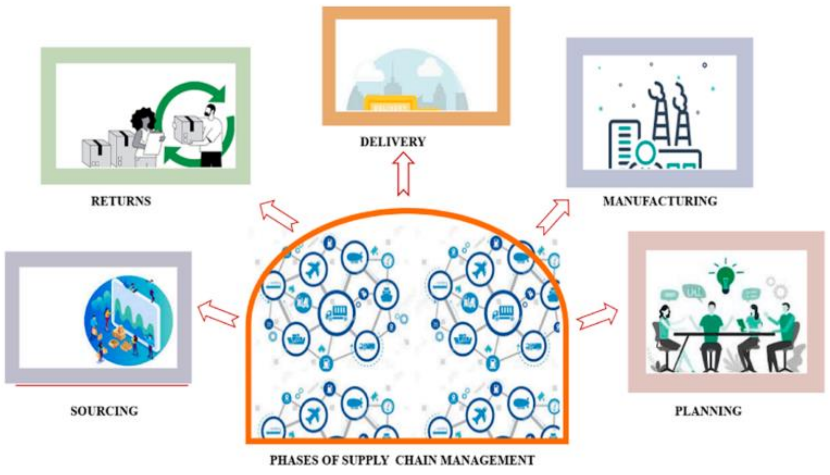


Figure 1: Different phases of SCM (Khader, A. M., & Rani, S. (2024)

Historically, businesses have relied on statistical methods such as Linear Regression (LR) and ARIMA (Auto-Regressive Integrated Moving Average) for demand forecasting due to their simplicity and interpretability. LR is effective for modeling stable, linear relationships, while ARIMA excels in capturing trends and seasonality in time-series data. However, these models have significant limitations, particularly in handling nonlinear relationships, high variability, and sudden market disruptions. For instance, ARIMA often falls short in predicting demand spikes caused by external factors such as geopolitical events or economic shifts. These limitations result in inefficiencies such as overstocking, stockouts, and missed sales opportunities, underscoring the need for more advanced and adaptable forecasting solutions.

Machine Learning (ML) has emerged as a powerful alternative to address the shortcomings of traditional methods. ML models, such as Decision Trees, Random Forest, and Gradient Boosting, excel at capturing nonlinear patterns, processing high-dimensional data, and adapting to real-time market changes. Decision Trees are effective for modeling categorical and numerical data, while Random Forest reduces overfitting by aggregating predictions from multiple decision trees. Gradient Boosting iteratively improves accuracy by minimizing prediction errors, making it particularly suited for complex and dynamic datasets. These advancements in ML enable businesses to forecast demand more accurately, optimize inventory levels, reduce costs, and enhance customer satisfaction. By leveraging ML, companies can overcome the challenges of traditional approaches, ensuring that their supply chains remain resilient and competitive in today's fast-paced market environment.

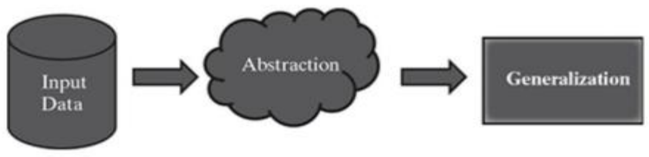


Figure 2: Process of Machine Learning

1.2 Problem Statement

Demand forecasting is a critical component of effective supply chain management (SCM) as it directly impacts inventory optimization, resource allocation, and operational efficiency. However, traditional statistical methods such as Linear Regression (LR) and ARIMA, while widely used, have notable limitations. These models often fail to account for the complexities of modern supply chains, including nonlinear relationships, fluctuating demand patterns, and the influence of external factors like market trends and economic conditions. This results in forecasting inaccuracies that lead to inefficiencies such as overstocking, stockouts, and increased operational costs.

Modern supply chain environments require solutions that are both adaptive and precise, capable of handling vast datasets with diverse variables such as shipping schedules, customer demographics, and purchase behavior. Machine learning (ML) models, including Decision Trees, Random Forest, and Gradient Boosting, have emerged as powerful tools for addressing these challenges. Despite their potential, there is a need to systematically evaluate their effectiveness in improving demand forecasting outcomes. Furthermore, there is a gap in understanding how ML models can optimize inventory management while minimizing forecasting errors.

This study seeks to address these challenges by exploring the performance of various ML models in demand forecasting, identifying their limitations, and determining the most effective techniques for enhancing forecasting accuracy. By leveraging advanced ML techniques, this research aims to contribute to the development of more resilient and efficient supply chain systems, ultimately improving decision-making and operational performance.

1.3 Research Objectives

Aim

The primary aim of this study is to enhance demand forecasting for order item quantities in supply chain management using Machine Learning (ML) models. By identifying key variables and evaluating the performance of various ML algorithms, the study seeks to provide actionable insights for improving inventory management, reducing operational inefficiencies, and optimizing resource allocation.

Objectives

1. To identify the key variables that significantly impact the demand forecasting of order item quantities.
2. To adapt and implement various Machine Learning algorithms, including Linear Regression, Decision Trees, Random Forest, and Gradient Boosting, for demand forecasting.
3. To compare the performance of these ML algorithms and determine the most suitable model for accurate and reliable demand forecasting.
4. To utilize the insights from demand forecasting to enhance inventory management, minimizing overstocking and stockouts, and improving overall operational efficiency.

1.4 Significance of the Study

This study is pivotal for advancing supply chain efficiency by exploring machine learning models' capabilities in demand forecasting. It addresses the limitations of traditional methods, such as their inability to handle non-linear relationships and dynamic datasets and highlights how techniques like Gradient Boosting optimize inventory management. By identifying key variables like product pricing and discounts, this research enhances forecasting accuracy, minimizes inefficiencies such as overstocking and stockouts, and supports resource allocation. Ultimately, it empowers businesses to make data-driven decisions, improve customer satisfaction, and remain competitive in fast-evolving market conditions.

1.5 Methodology Framework

The methodology of this study focused on improving demand forecasting for order item quantities in supply chain management using advanced machine learning (ML) techniques. The study employed the "DataCo Smart Supply Chain" dataset, containing 180,519 records from January 2015 to January 2018, to examine supply chain dynamics. Data preprocessing involved handling missing values, feature engineering, and stratified sampling, ensuring data integrity and relevance. Key variables such as product price, discounts, and monthly department activity were identified as influential factors.

Four ML algorithms Linear Regression, Decision Trees, Random Forest, and Gradient Boosting were implemented. Among these, Gradient Boosting demonstrated the highest R^2 and lowest RMSE, indicating superior predictive performance compared to other models. The methodology highlighted the role of ML in capturing complex, nonlinear relationships, enabling precise forecasting. This approach supports enhanced inventory management, strategic decision-making, and reduced operational inefficiencies in modern supply chains.

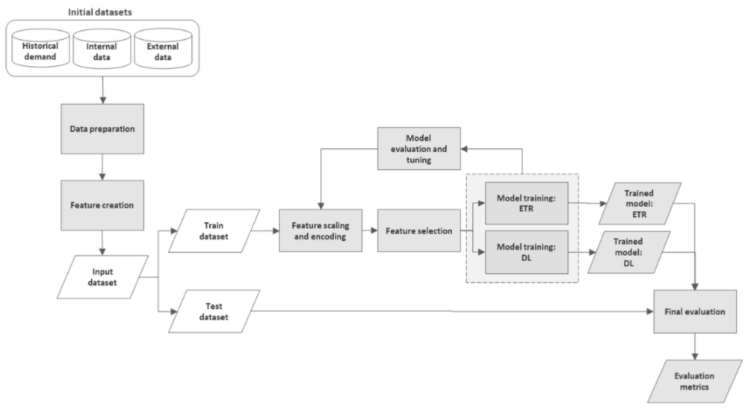


Figure 3: Workflow for Machine Learning Model Training and Evaluation

1.6 Structure and Flow of the Dissertation

This dissertation provides a thorough examination of demand forecasting in supply chains. The "Introduction" sets the stage by outlining the background, problem statement, and research objectives, specifically addressing the limitations of traditional methods such as Linear Regression and ARIMA. The "Literature Review" then explores existing literature on both traditional statistical approaches and machine learning models, including Decision

Trees, Random Forest, and Gradient Boosting, analyzing their respective strengths and weaknesses. The "Methodology" section details the dataset used (sourced from Kaggle), the preprocessing steps applied, and the implementation of the four chosen machine learning models. Performance is evaluated using RMSE and R^2 metrics. The "Data Analysis and Results" section presents the findings, highlighting Gradient Boosting as the most effective model for forecasting order item quantities. Finally, the "Conclusion" summarizes the research, and the "Recommendations" section offers actionable insights for stakeholders to leverage machine learning for improved demand forecasting, inventory optimization, and enhanced operational efficiency.

CHAPTER TWO: LITERATURE REVIEW

2.1 Literature Review

Demand forecasting is a cornerstone of effective supply chain management (SCM). It ensures inventory optimization, minimizes waste, and enhances overall operational efficiency. Traditional statistical models like Linear Regression (LR) and ARIMA have historically been utilized for this purpose, offering simplicity and interpretability. However, they often struggle to adapt to the dynamic and nonlinear nature of modern supply chains. This review delves deeper into the evolving role of machine learning (ML) models, specifically Linear Regression, Decision Trees, Random Forest, and Gradient Boosting, in addressing these challenges. The review also explores the comparative advantages of ML over traditional methods and deep learning techniques, its applications in demand forecasting, and its critical role in SCM.

2.2 Traditional Approaches in Demand Forecasting

Traditional statistical models have long been the mainstay of demand forecasting, primarily because of their simplicity and well-established methodologies. Among these, Linear Regression and ARIMA are the most prominent. Linear Regression (LR) is widely regarded for its straightforward implementation and interpretability. It establishes a linear relationship between independent and dependent variables, making it effective for datasets where patterns and trends are stable and linear. For example, LR has been employed to model the relationship between sales and pricing or seasonal demand patterns. However, its reliance on linearity limits its ability to account for more complex relationships in data. As a result, LR often fails to capture the nuances of non-linear patterns, outliers, and dynamic variations commonly observed in supply chains (Zheng, 2024). This model's simplicity becomes its drawback when datasets feature high variability or intricate interdependencies, such as sudden spikes in demand caused by external factors.

The Auto-Regressive Integrated Moving Average (ARIMA) model is a cornerstone in time-series forecasting, particularly for capturing trends and seasonality. Its structured approach—consisting of autoregressive terms, differences to stabilize data, and moving averages—makes it suitable for stationary time-series data. Nevertheless, ARIMA exhibits several shortcomings, including its inability to manage high variability and its lack of adaptability to rapid shifts in demand. For instance, ARIMA often struggles with datasets that experience sudden fluctuations or irregular patterns, such as those caused by external market dynamics or geopolitical events. These constraints often lead to suboptimal results, especially in modern supply chains where demand patterns can be volatile (Terrada et al., 2022). While these traditional approaches have merit, their limitations necessitate the exploration of advanced techniques, particularly those rooted in machine learning.

2.3 Machine Learning in Demand Forecasting

Machine learning has revolutionized the field of demand forecasting by offering sophisticated tools capable of handling complex and dynamic datasets. The flexibility, scalability, and adaptability of ML models make them particularly suited for the challenges of modern SCM. Among the diverse array of ML models, this review focuses on Linear Regression, Decision Trees, Random Forest, and Gradient Boosting. Linear Regression, while often considered a traditional statistical method, finds its place in machine learning through advanced implementations such as feature scaling, regularization techniques like Ridge and Lasso, and cross-validation. These enhancements mitigate overfitting and improve generalization, although LR remains best suited for structured numerical datasets and scenarios with predominantly linear relationships (Shakir & Modupe, 2023). Decision Trees, on the other hand, excel in capturing non-linear relationships by recursively splitting data into subsets based on decision rules. They are particularly effective in modeling categorical and numerical data, such as predicting seasonal demand patterns. However, Decision Trees are prone to overfitting, which can reduce their predictive accuracy unless techniques like pruning or ensemble methods are employed (Zheng, 2024).

Random Forest, as an ensemble learning method, combines multiple Decision Trees to improve predictive accuracy and robustness. By aggregating the predictions of individual trees, Random Forest reduces overfitting and provides reliable results. Its ability to handle missing values, outliers, and complex data structures makes it a versatile tool for demand forecasting. For example, RF has been applied to model multi-dimensional relationships in retail and inventory optimization (Vairagade et al., 2019). Its scalability also allows it to process large datasets efficiently, making it particularly useful for global supply chains. Similarly, Gradient Boosting models, such as XGBoost, LightGBM, and CatBoost, have gained popularity for their high accuracy and ability to handle diverse data types. These models iteratively improve weak learners by minimizing prediction errors, making them particularly effective for time-series data and scenarios with non-linear relationships. Gradient Boosting is computationally intensive but delivers unparalleled precision in complex forecasting tasks (Panda & Mohanty, 2023). For instance, Gradient Boosting has been used to predict demand for seasonal products by dynamically adjusting weights to minimize forecasting errors.

2.4 Applications of ML in Demand Forecasting

Machine learning models have transformed the landscape of demand forecasting by enabling more accurate predictions, better resource allocation, and optimized decision-making. Inventory optimization is one of the key applications, where ML models help businesses maintain optimal inventory levels by accurately forecasting demand. This reduces excess stock and minimizes stockouts, ensuring a smoother supply chain (Nasseri et al., 2023). By leveraging predictive analytics, organizations can reduce holding costs and avoid penalties associated with stock shortages. Another significant application is dynamic pricing, where ML algorithms analyze historical data and market trends to assist in real-time pricing strategies. For example, ML has been instrumental in e-commerce platforms that adjust prices dynamically based on demand elasticity, improving profitability and customer satisfaction (Douaioui et al., 2024). Additionally, ML enables the identification of bottlenecks and inefficiencies, facilitating better resource allocation and reducing lead times (Zhang, 2024). By using real-time tracking and forecasting, ML enhances decision-making processes, ensuring timely product delivery and reduced transportation costs.

2.5 Benefits of ML for Inventory Management in Demand Forecasting

Inventory management is a critical aspect of supply chain operations, and ML-based demand forecasting has emerged as a transformative tool for enhancing its efficiency. By leveraging ML models, organizations can align inventory levels with real-time demand, reducing both excess stock and stockouts. Machine learning algorithms such as Random Forest and Gradient Boosting analyze historical sales data, seasonal trends, and external factors such as economic conditions and weather to generate precise demand forecasts. This ensures that inventory levels are optimized to meet actual market needs (Nasseri et al., 2023). Excess inventory, which ties up capital and incurs storage costs, can be significantly reduced with ML-driven forecasting, thereby improving cash flow and overall operational efficiency (Shakir & Modupe, 2023). On the other hand, stockouts, which lead to lost sales and diminished customer trust, can be minimized as ML models identify trends and anomalies in demand patterns, enabling businesses to proactively replenish inventory and meet customer expectations (Douaioui et al., 2024). Advanced ML models like Gradient Boosting and Random Forest incorporate real-time data from sensors, point-of-sale systems, and market trends, allowing businesses to adjust inventory levels dynamically and respond to sudden demand changes (Zhang, 2024). Moreover, ML tools facilitate better communication and coordination with suppliers by providing accurate forecasts, ensuring timely procurement of raw materials and aligning production schedules with demand forecasts (Panda & Mohanty, 2023).

2.6 Advantages of Machine Learning in SCM

Machine learning (ML) offers transformative advantages in supply chain management (SCM), particularly in enhancing demand forecasting accuracy. One significant advantage of ML is its ability to process complex, high-dimensional datasets, which are typical in supply chains. Unlike traditional methods, ML models like Random Forest and Gradient Boosting handle non-linear relationships efficiently, enabling more accurate predictions of demand patterns, even in volatile markets (Douaioui et al., 2024).

ML's scalability is another critical advantage. Models such as Gradient Boosting can manage vast datasets encompassing historical demand, pricing trends, and external factors like weather or economic conditions. This scalability allows businesses to integrate diverse data sources, resulting in more robust and reliable forecasts (Panda & Mohanty, 2023). Furthermore, ML models excel in real-time forecasting, allowing supply chains to adapt dynamically to sudden changes in demand. This real-time capability minimizes inefficiencies, such as the Bullwhip Effect, and optimizes resource allocation across the supply chain (Zhang, 2024).Moreover, ML improves inventory management by aligning stock levels with precise demand forecasts. By minimizing overstocking and stockouts, ML not only reduces holding costs but also enhances customer satisfaction through timely product availability (Shakir & Modupe, 2023). Overall, ML's adaptability, efficiency, and predictive accuracy position it as an indispensable tool for modern SCM.

2.7 Limitations of Machine Learning in SCM

Machine learning (ML) has emerged as a critical tool in supply chain management (SCM), but its adoption is not without limitations. One significant challenge is the computational demands of advanced ML models like Gradient Boosting and Support Vector Machines (SVM), which often require high computational resources and specialized hardware, such as GPUs, to handle large datasets and complex algorithms efficiently (Zheng, 2024). For smaller organizations, these requirements can make implementation cost-prohibitive, limiting accessibility and scalability.

Another limitation is the potential for overfitting, particularly in models like Decision Trees and Gradient Boosting. When ML models are overly tuned to training data, they may struggle to generalize to unseen data, leading to inaccurate predictions and reduced reliability in real-world applications (Douaioui et al., 2024). This issue highlights the need for careful regularization and robust cross-validation techniques to ensure model stability.

Interpretability is also a major concern. While simpler models like Linear Regression provide clear insights into variable relationships, more complex ML models, such as Random Forest and XGBoost, often operate as "black boxes," making it difficult for stakeholders to understand the decision-making process (Panda & Mohanty, 2023). This lack of transparency can hinder trust and acceptance, especially in high-stakes SCM environments.

Lastly, ML models often require large volumes of high-quality, labelled data for training, which may not always be available in supply chain contexts. Data scarcity or poor data quality can compromise model performance, emphasizing the need for robust preprocessing and data augmentation strategies (Nasseri et al., 2023). Despite these challenges, ML continues to offer transformative potential in SCM, provided its limitations are carefully managed.

2.8 Comparison with Deep Learning

Machine learning (ML) models and deep learning (DL) approaches, while both pivotal in demand forecasting, cater to distinct challenges and opportunities within supply chain management (SCM). ML models such as Random Forest, Gradient Boosting, and Linear Regression provide a balance of interpretability, computational efficiency, and adaptability. These models are highly effective for structured datasets and non-linear relationships, offering robust performance in scenarios requiring lower computational resources and faster implementation (Douaioui et al., 2024).

Deep learning models, particularly Long Short-Term Memory (LSTM) networks, are designed for time-series data and excel in capturing sequential dependencies and non-linear patterns over long durations. For instance, LSTMs significantly outperform traditional ML models like ARIMA and Random Forest in dynamic environments involving extensive datasets, as they can learn complex temporal behaviors and interactions (Terrada et al., 2022). These capabilities make DL approaches especially valuable for datasets with strong temporal variability, such as retail demand influenced by weather or seasonal events (Panda & Mohanty, 2023).

However, the application of DL in real-world SCM comes with challenges. Deep learning models require substantial computational resources, including specialized hardware such as GPUs, and extensive training datasets to avoid issues like overfitting or underperforming in smaller, noisier datasets (Zhang, 2024). This limitation restricts their practicality for organizations with limited technological infrastructure. Conversely, ML models like Random Forest and Gradient Boosting deliver high accuracy even with modest datasets and are more accessible to businesses aiming for cost-effective yet precise forecasting solutions (Nasseri et al., 2023).

Another critical differentiator is interpretability. ML models such as Linear Regression and Random Forest allow for greater transparency in decision-making. For example, feature importance rankings in Random Forest or the coefficients in Linear Regression help explain model predictions, fostering stakeholder trust (Vairagade et al., 2019). On the other hand, DL models, particularly LSTMs, are often viewed as "black boxes" due to their complex architectures, making it challenging to trace the decision-making process. This lack of interpretability can hinder adoption in industries where clear, explainable models are preferred (Douaioui et al., 2024).

While deep learning models like LSTMs offer unparalleled accuracy in handling sequential, high-dimensional data, their resource-intensive nature and limited interpretability restrict their widespread applicability in SCM. ML models strike a balance between precision, efficiency, and explainability, making them more practical for organizations with varying levels of technological capacity. Therefore, the choice between ML and DL approaches should be guided by the specific requirements of the dataset, organizational resources, and the need for interpretability in decision-making processes (Zheng, 2024).

2.9 Machine Learning vs. Traditional Methods

Traditional forecasting methods such as Linear Regression (LR) and ARIMA have been extensively used in supply chain management (SCM) due to their simplicity and well-established frameworks. However, these methods often struggle with the complexity and variability of modern supply chains. Linear Regression, for instance, is efficient in capturing linear relationships between variables, making it suitable for structured datasets with consistent patterns. Yet, its inability to model non-linear dependencies limits its effectiveness in dynamic environments where demand fluctuations are influenced by multiple interdependent factors (Zhang, 2024). Similarly, ARIMA excels in time-series forecasting by modeling trends and seasonality in stationary data. However, it falls short in scenarios with significant variability or sudden shifts in demand, often resulting in higher error rates and reduced accuracy (Terrada et al., 2022).

Machine learning (ML) models address many of these limitations by offering greater adaptability and robustness. Unlike traditional models, ML approaches can handle non-linear relationships, high-dimensional data, and dynamic demand patterns. Decision Trees (DT), for example, are capable of modeling non-linear dependencies by segmenting data based on decision rules, although they may overfit without proper regularization (Zheng, 2024). Ensemble methods like Random Forest (RF) improve upon DTs by aggregating predictions from multiple trees, thereby enhancing accuracy and reducing overfitting. RF's versatility in handling missing values and outliers makes it particularly effective for real-world SCM applications (Vairagade et al., 2019).

Gradient Boosting (GB) models, including XGBoost and LightGBM, further enhance ML's capability by iteratively optimizing weak learners to minimize errors. These models are highly effective in complex forecasting tasks, such as capturing seasonal trends or demand spikes influenced by external variables like weather or economic factors (Panda & Mohanty, 2023). In comparison, traditional methods like ARIMA and Linear Regression lack the flexibility to incorporate such external variables effectively, often resulting in less accurate predictions.

Another key advantage of ML over traditional methods lies in scalability and real-time adaptability. ML models can process vast datasets and dynamically adjust to changing demand patterns, enabling businesses to respond more effectively to market fluctuations. For example, ML algorithms can integrate real-time sales data and external factors to optimize inventory levels, a capability that traditional models like ARIMA cannot match (Douaioui et al., 2024). Additionally, ML models are capable of reducing inefficiencies such as the Bullwhip Effect by improving forecast accuracy across multiple stages of the supply chain.

While traditional models remain valuable for their simplicity and interpretability, ML approaches significantly outperform them in terms of accuracy, scalability, and adaptability. Studies have demonstrated that ML models consistently achieve lower error rates and higher R^2 values compared to traditional methods, making them better suited for modern SCM challenges. For instance, a comparative study found that Random Forest achieved an R^2 of 0.83, outperforming both Linear Regression and ARIMA in dynamic retail environments (Vairagade et al., 2019).

While traditional methods provide a foundational approach to demand forecasting, machine learning offers advanced capabilities that address the complexities of modern supply chains. By leveraging ML models, organizations can achieve more accurate forecasts, optimize resource allocation, and enhance overall supply chain efficiency. Future research should explore hybrid models that combine the strengths of ML and traditional approaches to further enhance scalability and performance (Zheng, 2024).

2.10 Conclusion

This review highlights the transformative role of ML models Linear Regression, Decision Trees, Random Forest, and Gradient Boosting in demand forecasting for SCM. While deep learning models like LSTM provide unparalleled accuracy for sequential data, their resource intensity makes them less practical in many settings. ML models offer a pragmatic balance between accuracy, interpretability, and computational efficiency, making them ideal for addressing modern supply chain challenges. By adopting these models, this study aims to enhance forecasting accuracy, optimize inventory management, and improve decision-making processes, paving the way for more resilient and efficient supply chains.

CHAPTER THREE: METHODOLOGY

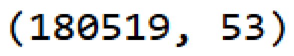
3. Methodology

3.1.Data Source

We sourced our data from the "**DataCo Smart Supply Chain for Big Data Analysis**" dataset, a publicly available resource on Kaggle. Kaggle is a well-known hub for data science and machine learning datasets, making it a reliable source for this type of research. This particular dataset is designed to explore big data analysis within the realm of supply chain management.

3.1.1 Dataset Overview

The dataset, with 180,519 records across 53 fields, offers a detailed view of supply chain operations from January 2015 to January 2018. It includes order types, shipping schedules, customer purchases, delivery statuses, late delivery risks, product categories, customer demographics, and logistical data, making it a valuable resource for analyzing supply chain dynamics.



```
(180519, 53)
```

Figure 4: Dataset Dimensions

3.1.2 Data Description

The dataset consists of 53 variables that capture different aspects of supply chain activity. The following table explains each variable, along with its corresponding data type and classification.

Table 1: Variable Descriptions

SI.No	Variable Name	Description	Data Type	Type
1	Type	Type of transaction	Object	Categorical
2	Days for shipping (real)	Actual shipping time in days	Integer	Numerical (Discrete)
3	Days for Shipment (scheduled)	Scheduled shipping time in days	Integer	Numerical (Discrete)
4	Benefit per order	Profit per order	Float	Numerical (Continuous)
5	Sales per customer	Sales generated by each customer	Float	Numerical (Continuous)

6	Delivery Status	Status of the delivery	Object	Categorical
7	Late_delivery_risk	Binary indicator for late delivery	Integer	Numerical (Discrete)
8	Category Id	Unique identifier for category	Integer	Numerical (Discrete)
9	Category Name	Name of the category	Object	Categorical
10	Customer City	City of the customer	Object	Categorical
11	Customer Country	Country of the customer	Object	Categorical
12	Customer Email	Email address of the customer	Object	Categorical
13	Customer Fname	First name of the customer	Object	Categorical
14	Customer Id	Unique identifier for the customer	Integer	Numerical (Discrete)
15	Customer Lname	Last name of the customer	Object	Categorical
16	Customer Password	Customer password	Object	Categorical
17	Customer Segment	Segment classification of the customer	Object	Categorical
18	Customer State	State of the customer	Object	Categorical
19	Customer Street	Street address of the customer	Object	Categorical
20	Customer Zipcode	Zip code of the customer	Integer	Numerical (Discrete)
21	Department Id	Unique identifier for the department	Integer	Numerical (Discrete)
22	Department Name	Name of the department	Object	Categorical
23	Latitude	Geographical latitude	Float	Numerical (Continuous)
24	Longitude	Geographical longitude	Float	Numerical (Continuous)
25	Market	Market classification	Object	Categorical
26	Order City	City of the order	Object	Categorical
27	Order Country	Country of the order	Object	Categorical
28	Order Customer Id	Customer ID for the order	Integer	Numerical (Discrete)
29	order date (DateOrders)	Date the order was placed	Object	Categorical
30	Order Id	Unique identifier for the order	Integer	Numerical (Discrete)

31	Order Item Cardprod Id	Product ID for the order item	Integer	Numerical (Discrete)
32	Order Item Discount	Discount on the order item	Float	Numerical (Continuous)
33	Order Item Discount Rate	Discount rate on the order item	Float	Numerical (Continuous)
34	Order Item Id	Unique identifier for the order item	Integer	Numerical (Discrete)
35	Order Item Product Price	Price of the product in the order	Float	Numerical (Continuous)
36	Order Item Profit Ratio	Profit ratio of the order item	Float	Numerical (Continuous)
37	Order Item Quantity	Quantity of items ordered	Integer	Numerical (Continuous)
38	Sales	Total sales value	Float	Numerical (Continuous)
39	Order Item Total	Total cost for the order item	Float	Categorical
40	Order Profit Per Order	Profit for the order	Float	Categorical
41	Order Region	Region of the order	Object	Categorical
42	Order State	State of the order	Object	Numerical (Continuous)
43	Order Status	Status of the order	Object	Numerical (Discrete)
44	Order Zipcode	Zip code of the order	Float	Numerical (Discrete)
45	Product Card Id	Unique identifier for the product	Integer	Numerical (Continuous)
46	Product Category Id	Category ID for the product	Integer	Categorical
47	Product Description	Description of the product	Float	Categorical
48	Product Image	Image of the product	Object	Numerical (Continuous)
49	Product Name	Name of the product	Object	Numerical (Discrete)
50	Product Price	Price of the product	Float	Categorical
51	Product Status	Status of the product	Integer	Categorical
52	shipping date (DateOrders)	Date the product was shipped	Object	Categorical
53	Shipping Mode	Mode of shipping	Object	Categorical

3.1.3 Programming Environment

Jupyter Notebook was the primary platform for developing and testing Python scripts in this analysis, offering modular code development, debugging, and immediate insights through data visualization. As an open-source environment supporting over 40 programming languages, it integrates live code, visualizations, and text, making it ideal for data analysis and experimentation (Kluyver et al., 2016).

3.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial first step in any data analysis project. It involves examining datasets using statistical techniques (e.g., means, medians, standard deviations) and visual methods (e.g., histograms, box plots, scatter plots) to uncover key features, identify outliers, generate hypotheses, and assess data quality. EDA reveals data distribution, correlations, and variability, guiding analysts in creating well-fitting models. In our study, we will conduct EDA to ensure data integrity, investigate patterns, and prepare for effective modeling and machine learning.

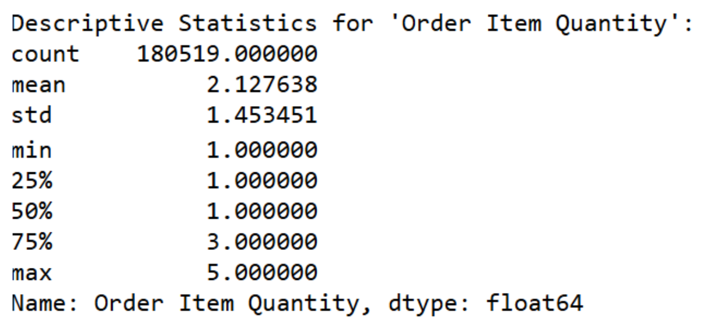
3.2.1 Target Variable

The dependent variable, Order Item Quantity, reflects the number of items a customer buys per order. It is crucial for understanding consumer behavior and identifying demand trends, seasonality, product types, pricing, shipping methods, and delivery performance. Analyzing these relationships enables more accurate demand forecasting and informed business decision-making, making it central to this study.

3.2.2 Descriptive statistics

Descriptive statistics give us a snapshot of our data. They tell us about the typical values, how spread out the data is, and where most of the values lie.

For our **Order Item Quantity**, we looked at 180,519 actual orders. On average, customers ordered about 2 items (2.13 to be precise), but there was some variation – the standard deviation of 1.45 tells us that orders weren't all clustered tightly around that average. The smallest order was just 1 item, and the largest was 5. Interestingly, if we look at where the data clusters, we see that 75% of all orders contained 3 items or fewer. This shows us that most people tend to buy smaller quantities in each order.



```
Descriptive Statistics for 'Order Item Quantity':
count    180519.000000
mean          2.127638
std           1.453451
min           1.000000
25%           1.000000
50%           1.000000
75%           3.000000
max           5.000000
Name: Order Item Quantity, dtype: float64
```

Figure 5: Descriptive Statistics for Order Item Quantity

3.2.3 Uni-variate Analysis

Univariate analysis is concerned with finding the average, definition, and actual representation of a particular variable and will often represent this distribution with a histogram or a boxplot. It should normally show common values, variability, and general distribution.

1. **Distribution of Order Item Quantity**

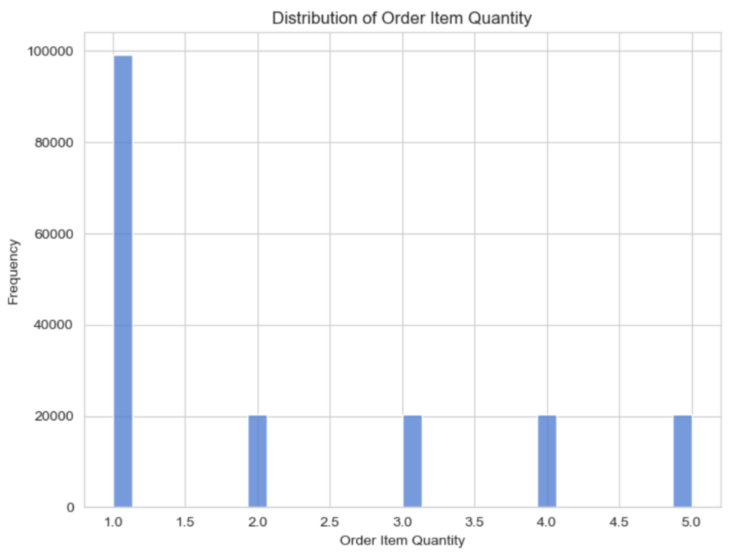


Figure 6: Frequency Distribution of Order Item Quantity

Description: The frequency distribution of Order Item Quantity is shown in this bar chart.

Insights: In terms of the order sizes, single-item orders were highly favored. There were around 100,000 of such orders. Large orders-four or five items-were quite infrequent, with less than 20,000 each. This data suggests that by handling small orders efficiently, one must also simultaneously analyse the various strategies for increasing an order's profitability.

2. Customer Segment Distribution

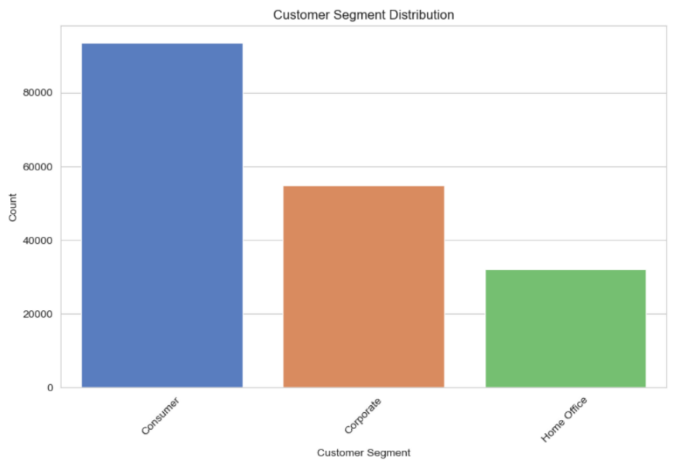


Figure 7: Customer Segment Distribution

Description: The bar graph shows how orders were divided among three different customer segments.

Insights: From the data shown in the bar chart, it can be seen that consumers are our biggest customer type, indicating about 90,000 orders. Corporate clients come next as our second-largest customer segment with about 60,000 orders, while home offices are the smallest group with only approximately 30,000 orders. For now, consumers are our biggest focus.

3. Transaction Types and Their Frequency

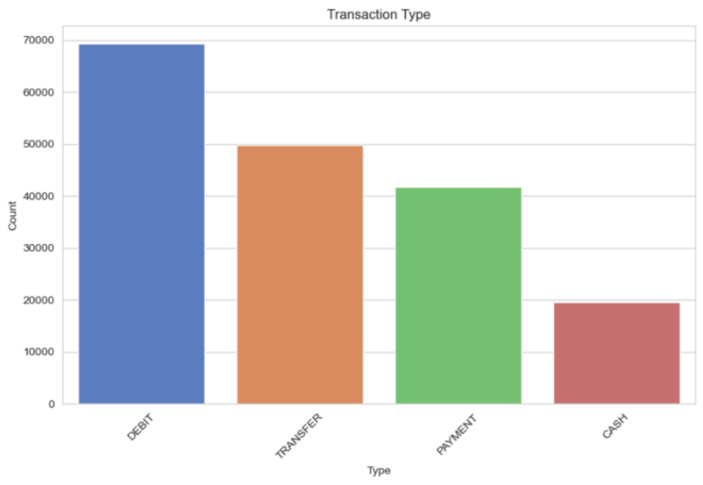


Figure 8: Transaction Type Distribution

Description: A bar chart showing the frequency of transactions across four types.

Insights: DEBIT transactions are the most popular, with 70,000 recorded, showing they are the preferred choice for most people. TRANSFER transactions come next, with 50,000, followed closely by PAYMENT at 45,000, highlighting their growing usage. CASH transactions, however, are the least common, with only 25,000, indicating that fewer people rely on cash these days. This shift toward digital payments reflects changing habits, with more users favouring convenience and efficiency over traditional methods.

3. **Frequency of Late Delivery Risk**

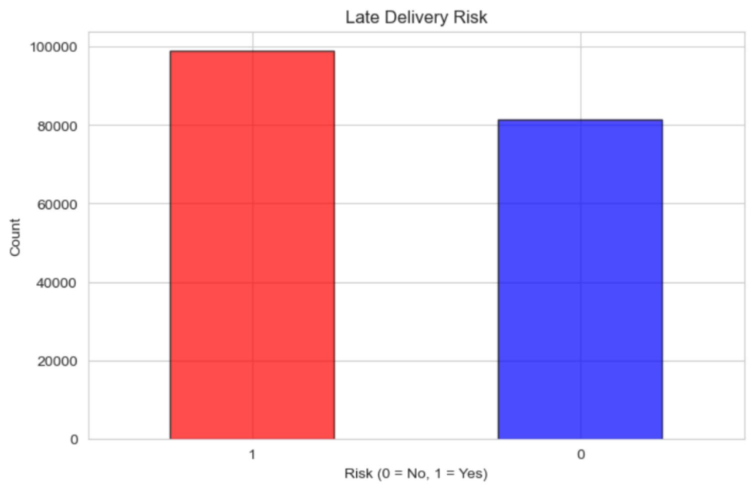


Figure 9: Late Delivery Risk Analysis

Description: A bar chart showing the frequency of orders segregating late delivery risk.

Insights: From this chart, we interpret that 0 means there is no risk in a late delivery while 1 means there is late delivery risk: about 90,000 orders are categorized as having a late delivery risk (1), implying that there is definitely a big problem with delivery times. Whereas 80,000 orders are delivered on time (0). These statistics indicate that improvements in logistics and other operations are needed so that any late deliveries are minimized in order to catch up with the prematurely existing time threshold of deliveries.

3.2.4 Bi-variate Analysis

Bivariate analysis is useful for investigating the relationship between two variables and plotting them as scatter plots or bar charts. It finds trends, associations, correlations, and differences in groups or categories.

1. Sales Trends by Month/Year

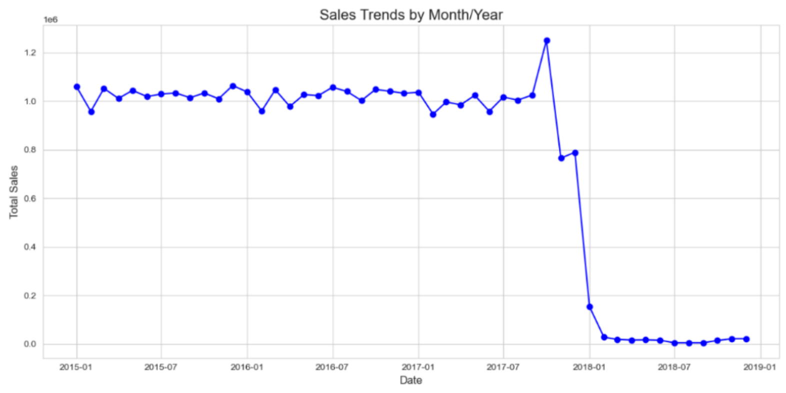


Figure 10: Sales Trends by Month and Year

Description: A line graph showing the total monthly sales over time.

Insights: The line graph shows relatively stable sales at approximately 1 million per month from 2015 to mid-2017, indicating consistent performance. However, there is a noticeable spike in sales in mid-2017, reaching a peak of 1.2 million in July 2017. Following this, sales sharply declined to near-zero by early 2018. This decline likely reflects insufficient data records in the dataset rather than an actual drop in sales. Therefore, data beyond 2018 can be disregarded due to incomplete records.

2. Top Product Departments by Order Item Quantity

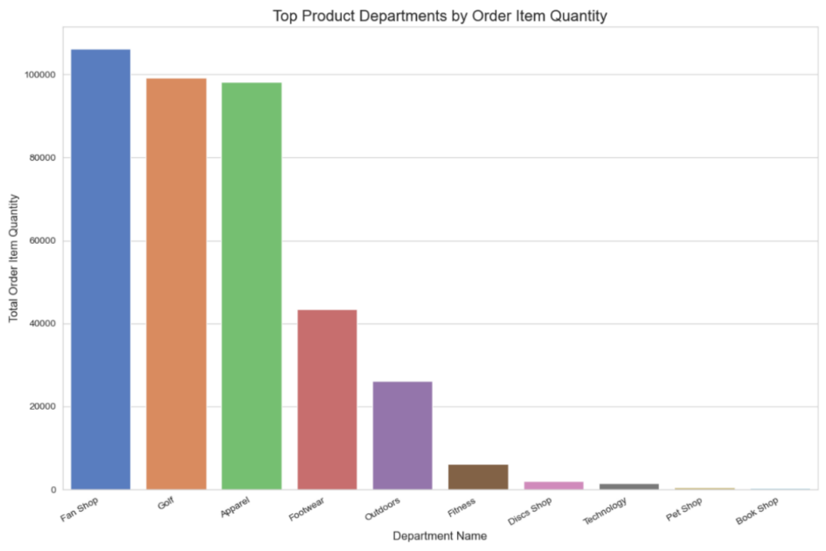


Figure 11: Top Product Departments by Order Item Quantity

Description: A bar chart showing the total order quantities across various departments.

Insights: The bar chart shows that the **Fan Shop**, **Golf**, and **Apparel** departments are our top performers, each with over **90,000 items ordered**. On the other hand, departments like **Pet Shop** and **Book Shop** have much lower order numbers. This suggests we should focus on strengthening and expanding the success of the high-demand departments, while also finding ways to boost sales in the smaller, underperforming categories.

3. Sales Analysis Across Order Quantities

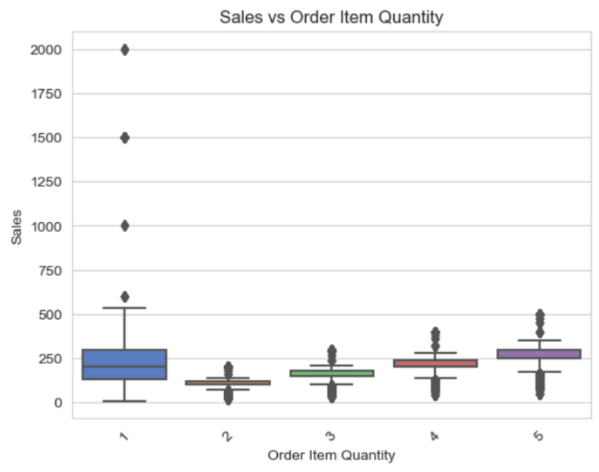


Figure 12: Sales vs. Order Item Quantity

Description: The boxplot provides a visual representation of how sales amounts vary across order quantities, ranging from 1 to 5 items.

Insights: Single-item orders tend to have the highest and most unpredictable sales, often involving higher-priced products. As the number of items in an order increases (from 2 to 5), sales amounts become more consistent, with slightly lower average values, possibly due to discounts or lower-priced bulk purchases. While there are occasional high-value outliers across all order sizes, these are most common with single-item orders. Overall, smaller orders are less predictable, while larger orders show more stability in sales.

4. **Analysis of Late Delivery Risk Across Shipping Modes**

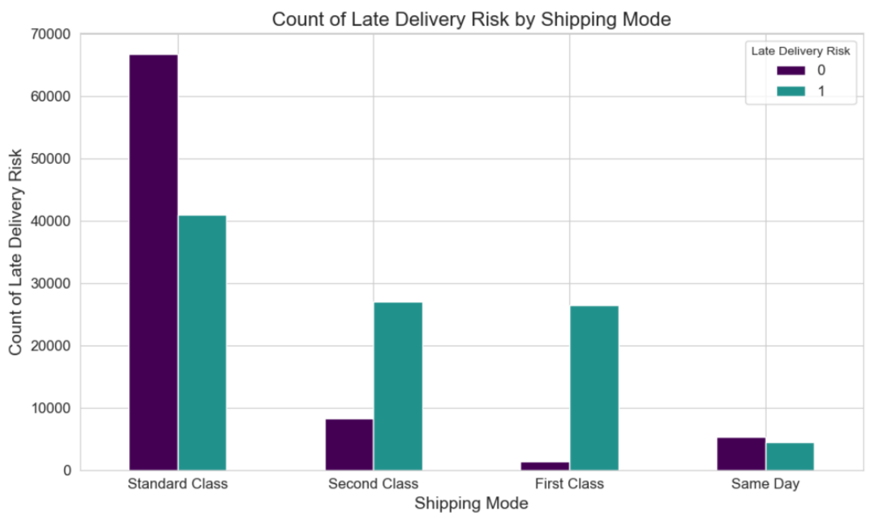


Figure 13: Count of Late Delivery Risk by Shipping Mode

Description: This bar graph illustrates the count of late delivery risks (categorized as "0" for no risk and "1" for risk) across four shipping modes.

Insights: The bar graph indicates that Standard Class is the most reliable shipping mode, with over 60,000 orders delivered on time, although it also has approximately 40,000 late deliveries, likely due to its high volume of usage. Second Class and First-Class shipping modes face more reliability challenges, with about 30,000 late deliveries each compared to fewer than 10,000 on-time deliveries. Same Day shipping, despite having the smallest overall usage with fewer than 10,000 orders, shows a high proportion of late deliveries, highlighting inefficiencies in this premium option. The data suggests that faster shipping modes, such as Same Day and First Class, often struggle with delays due to the pressure of meeting tight delivery deadlines.

3.2.5 Multi-variate Analysis

Multivariate analysis examines relationships between three or more variables, uncovering patterns, connections, and interactions in complex data.

Order Item Quantity Distribution by Region and Shipping Mode

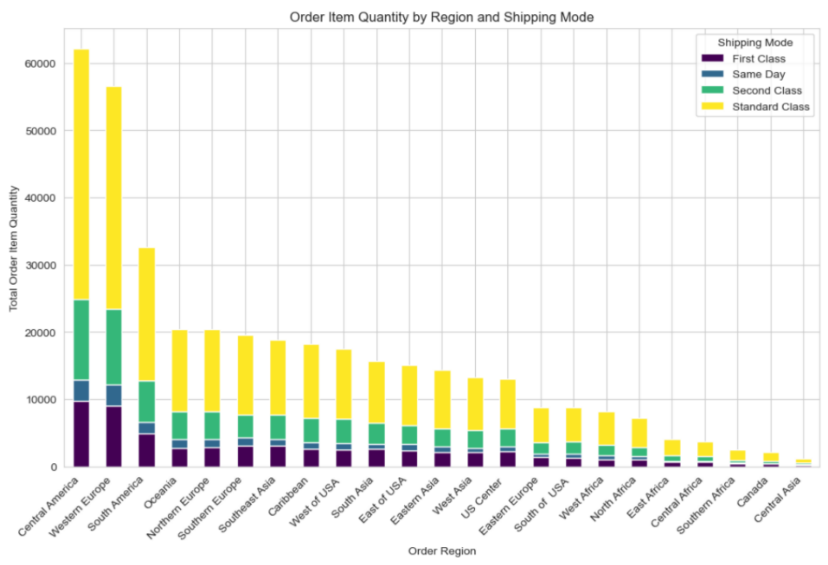


Figure 14: Order Item Quantity by Region and Shipping Mode

Description: This multivariate analysis graph illustrates the distribution of order quantities across different regions, grouped by shipping modes.

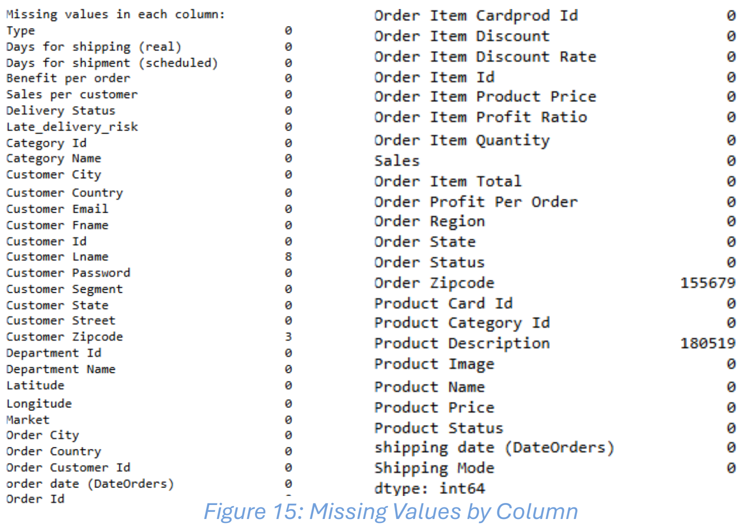
Insights: The stacked bar chart displays total order item quantities across various regions, categorized by shipping modes: "First Class," "Same Day," "Second Class," and "Standard Class." Central America and Western Europe record the highest order quantities, exceeding 60,000, with "Standard Class" being the most utilized shipping mode. Regions like South America, Oceania, and Northern Europe follow with order quantities ranging from 20,000 to 40,000, also dominated by "Standard Class." Regions such as Africa and Central Asia report less than 10,000 orders. "Same Day" and "First Class" shipping modes are rarely used, highlighting potential growth opportunities in low-volume regions like Africa and Central Asia.

3.3. DATA PRE-PROCESSING

Data preprocessing is the process of cleaning and preparing raw data so it can be used effectively in a machine learning model. Raw data often has issues like missing values, inconsistent formats, or irrelevant information, and preprocessing fixes these problems.

3.3.1 Handling Missing values

In machine learning, **missing values** refer to the absence of data in a dataset. This occurs when no value is recorded for a particular feature in a row.



```
Missing values in each column:
Type                                  0
Days for shipping (real)              0
Days for shipment (scheduled)         0
Benefit per order                     0
Sales per customer                    0
Delivery Status                       0
Late_delivery_risk                    0
Category Id                           0
Category Name                         0
Customer City                         0
Customer Country                      0
Customer Email                        0
Customer Fname                        0
Customer Id                           0
Customer Lname                        8
Customer Password                     0
Customer Segment                      0
Customer State                        0
Customer Street                       0
Customer Zipcode                      3
Department Id                         0
Department Name                       0
Latitude                              0
Longitude                             0
Market                                0
Order City                            0
Order Country                         0
Order Customer Id                     0
order date (DateOrders)               0
Order Id
```

```
Order Item Cardprod Id                0
Order Item Discount                   0
Order Item Discount Rate              0
Order Item Id                         0
Order Item Product Price              0
Order Item Profit Ratio               0
Order Item Quantity                   0
Sales                                 0
Order Item Total                      0
Order Profit Per Order                0
Order Region                          0
Order State                           0
Order Status                          0
Order Zipcode                    155679
Product Card Id                       0
Product Category Id                   0
Product Description              180519
Product Image                         0
Product Name                          0
Product Price                         0
Product Status                        0
shipping date (DateOrders)            0
Shipping Mode                         0
dtype: int64
```

Figure 15: Missing Values by Column

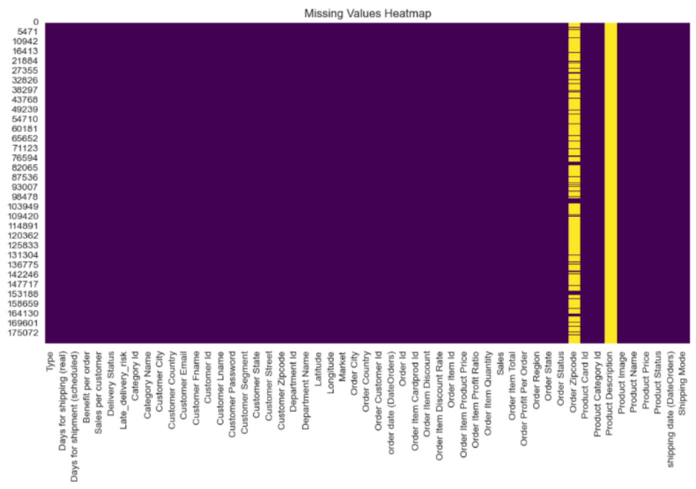
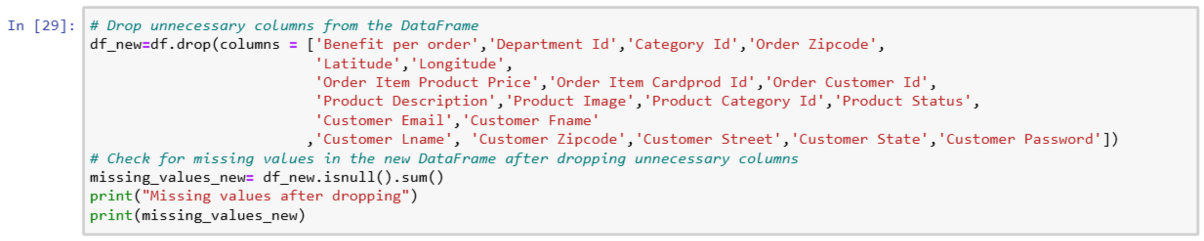


Figure 16: Heatmap of Missing Values

The above heatmap is a type of chart used to visualize the presence of missing values in a dataset. The yellow colour in the heatmap indicates missing values. In our dataset, the variables **"Order Zip Code"** and **"Product Description"** have 155,679 and 180,519 missing values, respectively. Since these variables do not have a significant influence on our target variable, we have decided to exclude them from our analysis.

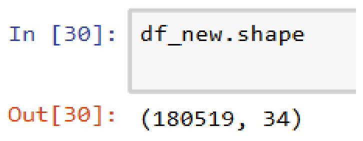
3.3.2 Dropping unnecessary columns



```
In [29]: # Drop unnecessary columns from the DataFrame
         df_new=df.drop(columns = ['Benefit per order','Department Id','Category Id','Order Zipcode',
                                   'Latitude','Longitude',
                                   'Order Item Product Price','Order Item Cardprod Id','Order Customer Id',
                                   'Product Description','Product Image','Product Category Id','Product Status',
                                   'Customer Email','Customer Fname'
                                  ,'Customer Lname', 'Customer Zipcode','Customer Street','Customer State','Customer Password'])
         # Check for missing values in the new DataFrame after dropping unnecessary columns
         missing_values_new= df_new.isnull().sum()
         print("Missing values after dropping")
         print(missing_values_new)
```

Figure 17: Dropping Unnecessary Columns

Some of the dropped variables are, Benefit per order, Customer Detail, Product Description, Order Item Product Price, and Location Data: latitude, longitude, etc. These variables either do not add to the actual estimation of order quantities or are identifiers, Customer ID, Product ID, or those providing redundant or possibly irrelevant qualification information, Product Images, Customer Password. This allows for the modification of the data in order to enhance analysis by removing such variables.



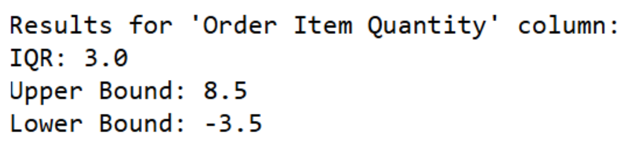
```
In [30]: df_new.shape

Out[30]: (180519, 34)
```

Figure 18: Dataset Dimensions After Cleaning

Further, with these columns deleted, a total of 34 variables are left to examine with respect to modeling and forecasting demand for Order Item Quantity.

3.3.3 Handling outliers

Outliers are essentially data points that are very different from the rest of the data. We need to identify and address them, as they can be caused by errors in data entry. We use the IQR method to find and deal with outliers, which are like the oddballs in our data. This helps us ensure our analysis is accurate and our models are reliable

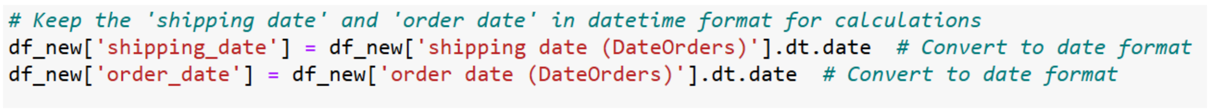


```
Results for 'Order Item Quantity' column:
IQR:  3.0
Upper Bound:  8.5
Lower Bound:  -3.5
```

Figure 19: IQR and Boundaries for 'Order Item Quantity'

The 'Order Item Quantity' column contains 5 distinct values: 1, 2, 3, 4, and 5, representing valid item quantities. The Interquartile Range (IQR) was calculated to be 3.0, leading to theoretical bounds of -3.5 (lower) and 8.5 (upper). However, as all values fall within the range of 1 to 5, there are no outliers present in the data. The calculated bounds are not applicable to this dataset due to its discrete and limited value range. No further action is required for handling outliers in this column.

3.3.2 Data Transformation



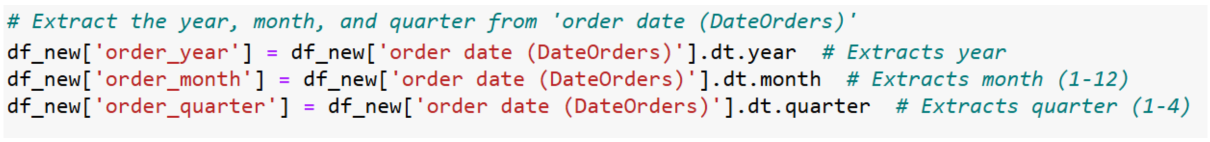
```
# Keep the 'shipping date' and 'order date' in datetime format for calculations
df_new['shipping_date'] = df_new['shipping date (DateOrders)'].dt.date  # Convert to date format
df_new['order_date'] = df_new['order date (DateOrders)'].dt.date  # Convert to date format
```

Figure 20:Converting Date Columns to Datetime Format

In our dataset, we have two columns: Shipping Date (DateOrders) and Order Date (DateOrders). The data includes both the date (in the format **month/day/year**) and time (hours and minutes), stored as strings (object format). To perform analysis, we need to convert these columns into a proper datetime format, ensuring the dates are standardized as **year/month/day**.

3.3.5 Feature Engineering

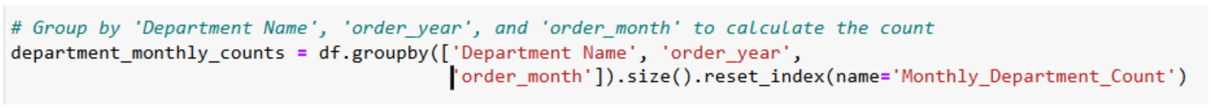
To improve how well machine learning models work, we use feature engineering to create new input variables, select the most relevant ones, or modify existing ones from the raw data. This process combines understanding of the data's meaning with technical data manipulation skills.



```
# Extract the year, month, and quarter from 'order date (DateOrders)'
df_new['order_year'] = df_new['order date (DateOrders)'].dt.year  # Extracts year
df_new['order_month'] = df_new['order date (DateOrders)'].dt.month  # Extracts month (1-12)
df_new['order_quarter'] = df_new['order date (DateOrders)'].dt.quarter  # Extracts quarter (1-4)
```

Figure 21:Extracting Year, Month, and Quarter from Order Date

we have created three new features: Order_Year, Order_Month, and Order_Quarter. From the Order Date, we extracted the year to identify when the order was placed and the specific month of the order. Furthermore, we derived the Order_Quarter, which categorizes months into their respective quarters (e.g., January belongs to Quarter 1, February to Quarter 2, and so on). Since we are conducting demand forecasting, order_year and order_month are key variables for our analysis.



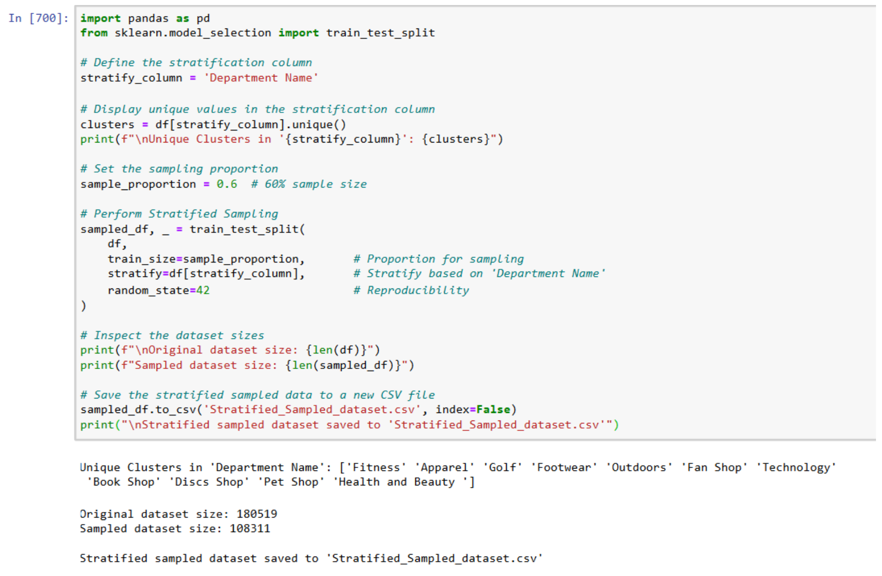
```
# Group by 'Department Name', 'order_year', and 'order_month' to calculate the count
department_monthly_counts = df.groupby(['Department Name', 'order_year',
                                        'order_month']).size().reset_index(name='Monthly_Department_Count')
```

Figure 22:Monthly Order Counts by Department

We are going to create another important column in our analysis, that is department_monthly_counts. This code calculates the number of orders (counts) for each department in each month of each year present in our data. The resulting department_monthly_counts DataFrame will have columns like 'Department Name', 'order_year', 'order_month', and 'Monthly_Department_Count'. Analyzing historical monthly order counts per department generated by grouping data by department, year, and month, enables demand forecasting, allowing businesses to optimize stock levels by increasing inventory for periods of high anticipated demand and reducing it during low-demand periods, preventing both stockouts and costly overstocking, ultimately improving cash flow and customer satisfaction.

3.3.6 Sampling

Sampling involves choosing a smaller, representative portion of a larger dataset to make analysis more manageable without losing key information. There are several ways to sample data including random, stratified, systematic, and cluster sampling. For our analysis, we're using stratified sampling because our dataset has distinct groups that need to be represented proportionally. This approach helps us ensure diversity in our sample, reduces variability in our results, and gives us a more accurate picture of the whole dataset, making it ideal for what we're trying to achieve.



```
In [700]: import pandas as pd
          from sklearn.model_selection import train_test_split

          # Define the stratification column
          stratify_column = 'Department Name'

          # Display unique values in the stratification column
          clusters = df[stratify_column].unique()
          print(f"\nUnique Clusters in '{stratify_column}': {clusters}")

          # Set the sampling proportion
          sample_proportion = 0.6  # 60% sample size

          # Perform Stratified Sampling
          sampled_df, _ = train_test_split(
              df,
              train_size=sample_proportion,       # Proportion for sampling
              stratify=df[stratify_column],       # Stratify based on 'Department Name'
              random_state=42                     # Reproducibility
          )

          # Inspect the dataset sizes
          print(f"\nOriginal dataset size: {len(df)}")
          print(f"Sampled dataset size: {len(sampled_df)}")

          # Save the stratified sampled data to a new CSV file
          sampled_df.to_csv('Stratified_Sampled_dataset.csv', index=False)
          print("\nStratified sampled dataset saved to 'Stratified_Sampled_dataset.csv'")
```

```
Unique Clusters in 'Department Name': ['Fitness' 'Apparel' 'Golf' 'Footwear' 'Outdoors' 'Fan Shop' 'Technology'
 'Book Shop' 'Discs Shop' 'Pet Shop' 'Health and Beauty ']

Original dataset size: 180519
Sampled dataset size: 108311

Stratified sampled dataset saved to 'Stratified_Sampled_dataset.csv'
```

Figure 23:Stratified Sampling by Department

The code employs stratified sampling on the "Department Name" column to create a smaller dataset (60% of the original), preserving proportional representation of each department. "Department Name" was selected for its balance between granularity and relevance, avoiding the complexity of more specific columns like "Product Name." The original dataset of 180,519 rows was reduced to 108,311 rows, ensuring key departments (e.g., Fitness, Apparel, Golf) remain proportionally represented. This approach minimizes bias, supports fairness, and facilitates departmental-level analysis. It ensures methodological rigor, reproducibility, and robust findings, making it ideal for dissertation research.

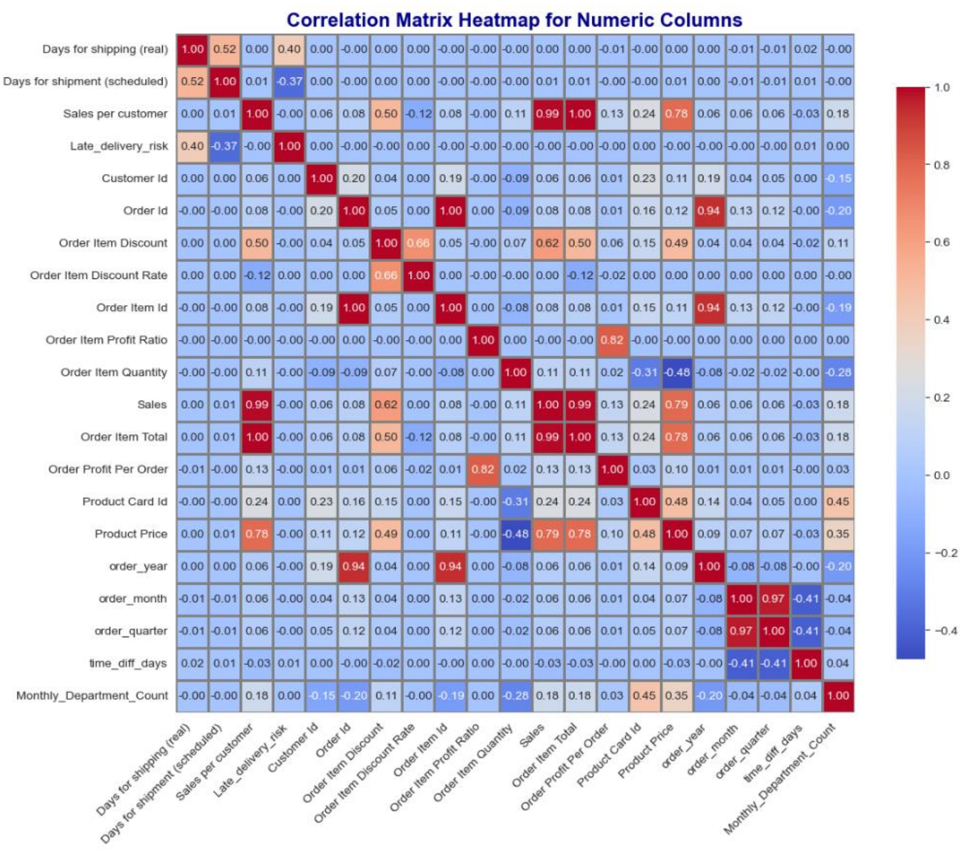
3.4 Correlation Matrix and Cramér's V Analysis

3.4.1 Purpose of analysis

We used correlation and Cramér's V to explore how different factors relate to the number of items ordered. These methods helped us pinpoint which numerical and categorical factors have a strong connection to order quantity, allowing us to choose the most relevant variables for our analysis.

3.4.2 Correlation Matrix Analysis

The correlation matrix is a statistical tool used to measure the strength and direction of linear relationships between numerical variables in a dataset. In this study, we used a correlation matrix to identify the numerical variables that have the greatest impact on **Order Item Quantity**.



	Days for shipping (real)	Days for shipment (scheduled)	Sales per customer	Late_delivery_risk	Customer Id	Order Id	Order Item Discount	Order Item Discount Rate	Order Item Id	Order Item Profit Ratio	Order Item Quantity	Sales	Order Item Total	Order Profit Per Order	Product Card Id	Product Price	order_year	order_month	order_quarter	time_diff_days	Monthly_Department_Count
Days for shipping (real)	1.00	0.52	0.00	0.40	0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.01	-0.00	0.00	0.00	-0.01	-0.01	0.02	-0.00
Days for shipment (scheduled)	0.52	1.00	0.01	-0.37	0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	0.01	0.01	-0.00	-0.00	0.01	0.00	-0.01	-0.01	0.01	-0.00
Sales per customer	0.00	0.01	1.00	-0.00	0.06	0.08	0.50	-0.12	0.08	-0.00	0.11	0.99	1.00	0.13	0.24	0.78	0.06	0.06	0.06	-0.03	0.18
Late_delivery_risk	0.40	-0.37	-0.00	1.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.01	0.00
Customer Id	0.00	0.00	0.06	0.00	1.00	0.20	0.04	0.00	0.19	-0.00	-0.09	0.06	0.06	0.01	0.23	0.11	0.19	0.04	0.05	0.00	-0.15
Order Id	-0.00	-0.00	0.08	-0.00	0.20	1.00	0.05	0.00	1.00	0.00	-0.09	0.08	0.08	0.01	0.16	0.12	0.94	0.13	0.12	-0.00	-0.20
Order Item Discount	0.00	0.00	0.50	-0.00	0.04	0.05	1.00	0.66	0.05	-0.00	0.07	0.62	0.50	0.06	0.15	0.49	0.04	0.04	0.04	-0.02	0.11
Order Item Discount Rate	0.00	0.00	-0.12	0.00	0.00	0.00	0.66	1.00	0.00	-0.00	-0.00	0.00	-0.12	-0.02	0.00	0.00	0.00	0.00	0.00	0.00	-0.00
Order Item Id	-0.00	-0.00	0.08	-0.00	0.19	1.00	0.05	0.00	1.00	0.00	-0.08	0.08	0.08	0.01	0.15	0.11	0.94	0.13	0.12	-0.00	-0.19
Order Item Profit Ratio	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	1.00	0.00	-0.00	-0.00	0.82	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00
Order Item Quantity	-0.00	-0.00	0.11	-0.00	-0.09	-0.09	0.07	-0.00	-0.08	0.00	1.00	0.11	0.11	0.02	-0.31	-0.48	-0.08	-0.02	-0.02	-0.00	-0.28
Sales	0.00	0.01	0.99	-0.00	0.06	0.08	0.62	0.00	0.08	-0.00	0.11	1.00	0.99	0.13	0.24	0.79	0.06	0.06	0.06	-0.03	0.18
Order Item Total	0.00	0.01	1.00	-0.00	0.06	0.08	0.50	-0.12	0.08	-0.00	0.11	0.99	1.00	0.13	0.24	0.78	0.06	0.06	0.06	-0.03	0.18
Order Profit Per Order	-0.01	-0.00	0.13	-0.00	0.01	0.01	0.06	-0.02	0.01	0.82	0.02	0.13	0.13	1.00	0.03	0.10	0.01	0.01	0.01	-0.00	0.03
Product Card Id	-0.00	-0.00	0.24	0.00	0.23	0.16	0.15	0.00	0.15	-0.00	-0.31	0.24	0.24	0.03	1.00	0.48	0.14	0.04	0.05	0.00	0.45
Product Price	0.00	0.01	0.78	-0.00	0.11	0.12	0.49	0.00	0.11	-0.00	-0.48	0.79	0.78	0.10	0.48	1.00	0.09	0.07	0.07	-0.03	0.35
order_year	0.00	0.00	0.06	-0.00	0.19	0.94	0.04	0.00	0.94	0.00	-0.08	0.06	0.06	0.01	0.14	0.09	1.00	-0.08	-0.08	-0.00	-0.20
order_month	-0.01	-0.01	0.06	-0.00	0.04	0.13	0.04	0.00	0.13	0.00	-0.02	0.06	0.06	0.01	0.04	0.07	-0.08	1.00	0.97	-0.41	-0.04
order_quarter	-0.01	-0.01	0.06	-0.00	0.05	0.12	0.04	0.00	0.12	0.00	-0.02	0.06	0.06	0.01	0.05	0.07	-0.08	0.97	1.00	-0.41	-0.04
time_diff_days	0.02	0.01	-0.03	0.01	0.00	-0.00	-0.02	0.00	-0.00	0.00	-0.00	-0.03	-0.03	-0.00	0.00	-0.03	-0.00	-0.41	-0.41	1.00	0.04
Monthly_Department_Count	-0.00	-0.00	0.18	0.00	-0.15	-0.20	0.11	-0.00	-0.19	0.00	-0.28	0.18	0.18	0.03	0.45	0.35	-0.20	-0.04	-0.04	0.04	1.00

Figure 24:Correlation Matrix Heatmap for Numeric Columns

The Correlation matrix analysis, **Product Price** has a very good negative correlation, -0.48 with quantities ordered. From this, as product prices would increase, most customers tend to order fewer numbers of items-this is the natural negative price elasticity of demand: customers tend not to buy quantities when prices have increased. Conclusion The findings of the quantitative analysis emphasize just how crucial this aspect of ordering volume can actually be influenced in terms of by pricing strategies adopted. Similarly, **Order Item Discount** has a weak positive correlation of 0.07, which indicates that as the percentage of discount increases, customers do increase their purchase amount, but only in a very limited extent. It is not a major contributing factor itself, but it operates in concurrence with other factors such as price and product category.

The **Order Year** has a very weak negative correlation of -0.08, which indicates that quantities ordered start to depreciate over time. It may be associated with market trends, customers' behavior, or business practice inside. Comparably, **Order Month** has a very weak positive correlation of 0.02, and seasonal fluctuations do not have a direct impact on order quantities. Nevertheless, it is connected because seasonal patterns might interact with other factors like promotions or availability of a product in having an indirect effect on order behavior.

The **Monthly Department Count** is moderately negatively correlated at -0.28, indicating that the more departments that are transacted, the lower the average quantity ordered per transaction. This could be an effect of operational practice where orders are spread out over departments, creating smaller, more frequent transactions instead of larger bulk orders.

Sales and **Product card ID** are not included in the analysis. Although the order quantity has a very weak positive correlation of 0.11 with sales, it is calculated directly by the formula Sales=Product Price×Order Item Quantity. Adding it would create redundancy and sometimes overlap, making the effects of Product Price and Order Item Quantity harder to read from the model. Product card ID alone doesn't give much insight into customer behavior or trends in orders; the name of the product was substituted in for clarity and its improvement.

3.4.3 Cramér's V Analysis

Cramér's V is the statistical measure used for the estimation of the strength of association between categorical variables. It ranges from 0 to 1. The former signifies no association while the latter implies a strong association. It is also used in finding the dependency of categorical variables on the target variable, which Order Item Quantity in this analysis.

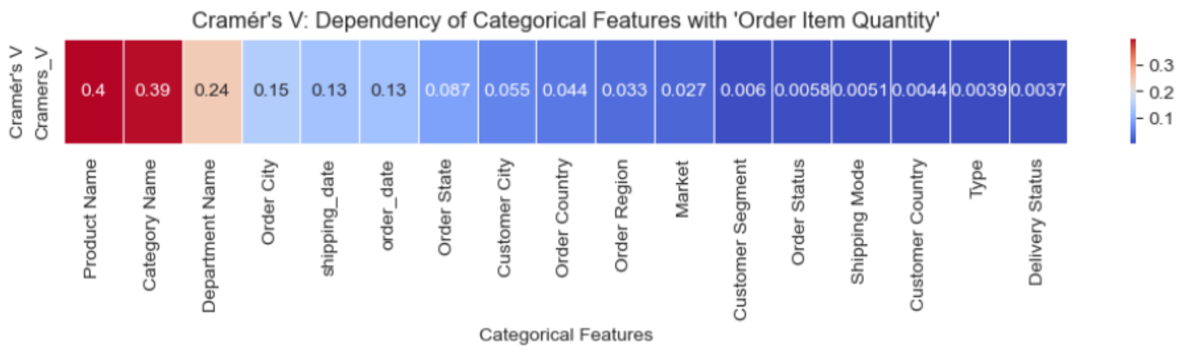
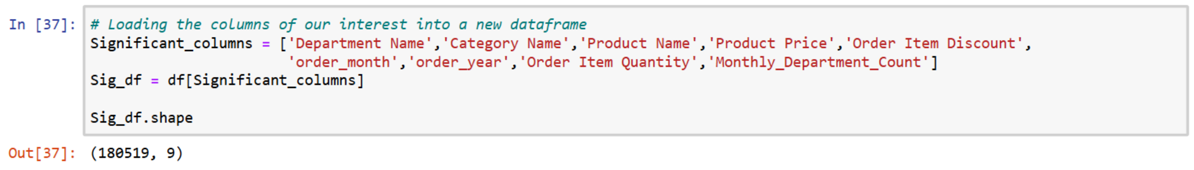


Figure 25: Cramér's V Analysis: Dependency of Categorical Features with 'Order Item Quantity'

Product Name is strongly associated with a value of 0.40, indicating that customers order specific products in different quantities, hence showing various preferences of customers. Similarly, **Category Name** has a moderately strong dependence with a value of 0.39, thereby underlining the very important role that product categories play in determining order volume. **Department Name** has a moderate association with a value of 0.24, therefore showing its influence on the quantity of items ordered. In contrast, other variables like Order Status, Shipping Mode, and Delivery Status showed weak associations with Cramér's V values near 0, meaning those have minimal influence on the target variable and hence should be eliminated from further analysis.

3.5 Key Influential Variables



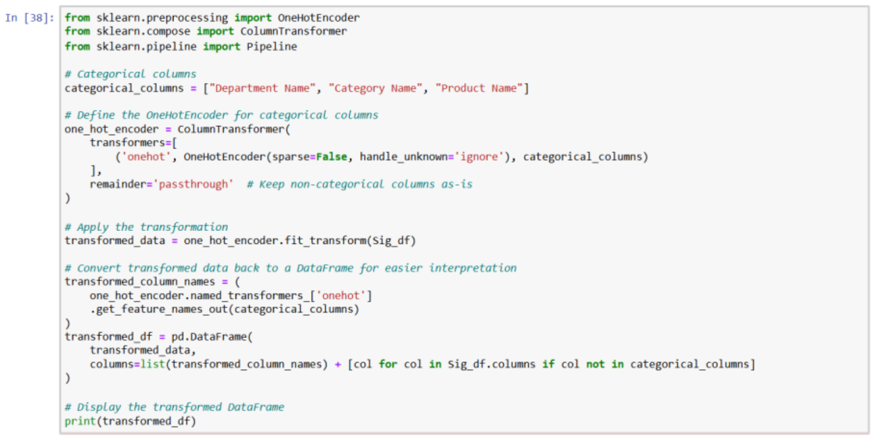
```
In [37]: # Loading the columns of our interest into a new dataframe
         Significant_columns = ['Department Name','Category Name','Product Name','Product Price','Order Item Discount',
                                'order_month','order_year','Order Item Quantity','Monthly_Department_Count']
         Sig_df = df[Significant_columns]

         Sig_df.shape

Out[37]: (180519, 9)
```

Figure 26: Selection of Key Influential Variables

Analysis of the correlation matrix and Cramer's V identified nine key variables influencing order item quantities: Department Name, Category Name, Product Name, Product Price, Order Item Discount, Order Month, Order Year, Order Item Quantity, and Monthly Department Count. These variables form the basis for demand forecasting, offering insights to optimize inventory, enhance supply chain efficiency, and align strategies with customer demand, enabling practical and accurate decision-making.

3.6 One Hot Encoder

One-Hot Encoding is a method that allows machines to understand categorical variables by turning them into numbers. For machine learning embedding models, significance is assigned to perceiving a category versus not perceiving a category by creating binary columns across categories. Encoding enables models to use this information without biasing the results. Transforming text-based data into categories is critical because most machine learning models fail to understand the text categories. It enables machines to use this information while avoiding any bias introduced by these variables.



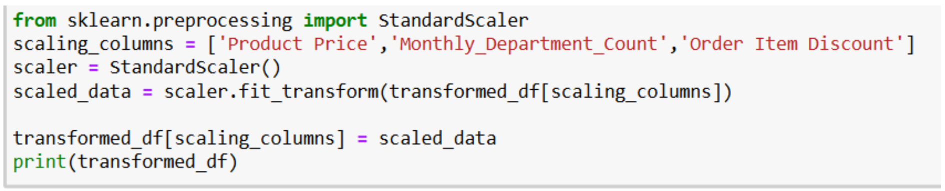
```
In [38]: from sklearn.preprocessing import OneHotEncoder
         from sklearn.compose import ColumnTransformer
         from sklearn.pipeline import Pipeline

         # Categorical columns
         categorical_columns = ["Department Name", "Category Name", "Product Name"]

         # Define the OneHotEncoder for categorical columns
         one_hot_encoder = ColumnTransformer(
             transformers=[
                 ('onehot', OneHotEncoder(sparse=False, handle_unknown='ignore'), categorical_columns)
             ],
             remainder='passthrough'  # Keep non-categorical columns as-is
         )

         # Apply the transformation
         transformed_data = one_hot_encoder.fit_transform(Sig_df)

         # Convert transformed data back to a DataFrame for easier interpretation
         transformed_column_names = (
             one_hot_encoder.named_transformers_['onehot']
             .get_feature_names_out(categorical_columns)
         )
         transformed_df = pd.DataFrame(
             transformed_data,
             columns=list(transformed_column_names) + [col for col in Sig_df.columns if col not in categorical_columns]
         )

         # Display the transformed DataFrame
         print(transformed_df)
```

Figure 27:One-Hot Encoding of Categorical Variables

In this code, Department Name, Category Name, and Product Name were categories that were converted into binary vectors through the ColumnTransformer that was OneHotEncoded into this code, and other columns remained unchanged (remainder = 'passthrough'). For enhanced visualization and analysis, the modified data was transformed into a DataFrame after the transformation.

3.7 Scaling

Scaling involves converting numerical variables into a common scale such as 0 mean's and 1 standard deviations. Variables with greater numerical ranges will not unduly affect the outcome and a fair comparison can be made, and the effectiveness of the model will be enhanced in this manner.



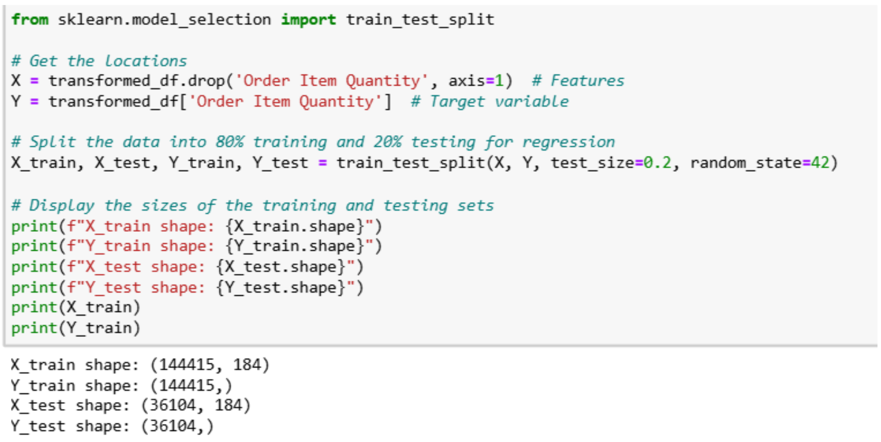
```
from sklearn.preprocessing import StandardScaler
scaling_columns = ['Product Price','Monthly_Department_Count','Order Item Discount']
scaler = StandardScaler()
scaled_data = scaler.fit_transform(transformed_df[scaling_columns])

transformed_df[scaling_columns] = scaled_data
print(transformed_df)
```

Figure 28:Standard Scaling of Numerical Features

Numerical variables like Product Price, Monthly Department Count, and Order Item Discount were scaled using StandardScaler for consistent ranges and better modeling. The target variable, Order Item Quantity, was not scaled to preserve its relationship with independent variables. Order Year and Order Month were also excluded, as scaling temporal data could distort their interpretability and impact. This preprocessing approach ensures the integrity of key variables, supporting reliable and accurate outcomes in predictive modeling and enhancing the evaluation process.

3.8 Test-Train Split

Test-Train Split it is an important step in machine learning thought process in which the model performance is thoroughly evaluated. In the process, the entire dataset is split into two subsets: a **training set** which is used to train the model and a **test set** which is used to evaluate the model with new data not seen before by the model. This also deals with occurrence of overfitting which is when a model is seen to work well on training data but gives poor output on new data. So, by splitting the data we are able to determine the ability of the model to generalize.



```
from sklearn.model_selection import train_test_split

# Get the locations
X = transformed_df.drop('Order Item Quantity', axis=1)  # Features
Y = transformed_df['Order Item Quantity']  # Target variable

# Split the data into 80% training and 20% testing for regression
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Display the sizes of the training and testing sets
print(f"X_train shape: {X_train.shape}")
print(f"Y_train shape: {Y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"Y_test shape: {Y_test.shape}")
print(X_train)
print(Y_train)
```

```
X_train shape: (144415, 184)
Y_train shape: (144415,)
X_test shape: (36104, 184)
Y_test shape: (36104,)
```

Figure 29:Train-Test Split

Here, the above code executes a test-train split followed by the target variable and feature split and then uses the previously split data for training and testing. **X** is a feature set which is derived by omitting the target variable **Order Item Quantity** of the DataFrame, and **Y** is the target variable, which in this case is **Order Item Quantity**, the value the model aims to predict. A function called train_test_split is employed wherein 80% of the data is set aside for training and 20% meant for testing. The parameter test_size = 0.2 assigns the testing data, while random_state=42 allows the split to be performed several times with no change in results. The output of this process is as follows: the training set consists of 144,415 rows and 184 columns (features) in **X_train** along with the corresponding target values in **Y_train**, while the testing set includes 36,104 rows and 184 columns in **X_test** with their respective target values in **Y_test**.

3.9 Implementation of Machine Learning Models

In this research, we choose four machine learning algorithms for demand forecasting Linear Regression, Decision Trees, Random Forest, and Gradient Boosting. They have been implemented on the basis of literature-reviewed knowledge and their practical reliability in conducting forecasting tasks of different complexity and non-linearity degrees.

1. **Linear Regression (Baseline Model)** is a straightforward, yet effective statistical technique used to model the relationship between a dependent variable and one or more independent variables. It operates under the assumption of a linear relationship between inputs and outputs. Serving as a baseline model, linear regression offers a point of reference for assessing the performance of more advanced algorithms. Douaioui et al. (2024) point out that while linear regression works well with structured numerical datasets, it struggles with non-linear and dynamic demand patterns. Although it is simple and easy to interpret, its limitations in modeling non-linear dependencies render it less suitable for the complex and variable patterns often seen in demand forecasting.

2. **Decision trees** are non-parametric models that divide the dataset into subsets based on feature values, forming a tree-like structure. They are both interpretable and effective at capturing non-linear relationships in data. Zhang (2024) highlights the ability of decision trees to manage non-linear relationships, making them particularly useful in dynamic supply chain contexts. Decision trees are favoured for their interpretability and their capacity to handle non-linear and categorical variables, which are crucial for recognizing patterns in order-item quantity data.

3. **Random forest** is an ensemble technique that constructs multiple decision trees and merges their outputs to enhance accuracy and minimize overfitting. It is robust and performs well with complex datasets. Vairagade et al. (2019) discovered that random forest is practical and efficient, often surpassing artificial neural networks in large-scale applications. The robustness and scalability of random forest make it well-suited for capturing the diverse and intricate patterns of demand in supply chains, particularly when working with high-dimensional data.

4. **Gradient Boosting** is a step-by-step ensemble method that focuses on minimizing errors at each iteration to improve predictions. It is particularly well-suited for working with

structured data that involves complex interactions between features. According to Panda and Mohanty (2023), Gradient Boosting techniques, such as XGBoost, outperform traditional statistical models by effectively capturing non-linear relationships. Its strength lies in managing intricate patterns and prioritizing error reduction, making it an excellent tool for enhancing forecasting accuracy.

3.10 Advantages of Selected Techniques Compared to Regression

For the demand forecasting of order item numbers, we used the most powerful machine learning methods because it was able to identify complex patterns, analyze large-scale data and make accurate predictions. These are especially useful when it comes to supply chains with varying dynamics seasonality, promotions, economic indicators.

1.**Non-Linear Dependencies in Demand Prediction** Standard linear models are insufficient for modelling non-linear dependencies, which often occur in demand forecasting applications. Advanced algorithms can account for these details and then be more predictive by accounting for the actual facts of the data.

2. **Managing Missing and Noisy Data** Supply chain data contains missing or noisy information because of real-world disparities. Advanced algorithms like ensemble are resistant to such situations and predicts even if the dataset is imperfect can still be reliable and stable.

3. **Data Acquisition of Nonlinear Relationships** Within supply chains, many forces seasonal fluctuations, advertising campaigns, economic cycles – behave non-linearly. These complex interactions are detected and modelled by powerful machine learning methods, which is why they are essential to a precise demand prediction. They are essential to a precise demand prediction.

4. **Incremental Error Reduction** Some methods improve the predictions iteratively by reducing errors on subsequent runs. This incremental procedure improves the model's ability to extract small patterns and relationships for a better forecast.

5. **Strength Against Outliers** Supply chain datasets might have outliers due to an unexpected event or reporting mistake. High-end techniques can be made to less sensitive to these deviations so that predictions are stable and accurate.

6. **Scalability and Performance** The high throughput required for the supply chain data requires algorithms capable of dealing with a lot of information without sacrificing performance. They're designed to be scalable and scale well over large datasets.

3.11 Prediction Accuracy Measures

The key performance metrics that the study relies on in evaluating machine learning models in demand forecasting are **R^2** (Coefficient of Determination) and **RMSE** (Root Mean Squared Error).

R^2 gives the proportion of variation in the dependent variable explained by the model. Close to 1 means strong predictive power, hence making sure that the model correctly represents data relationships. That would be important in inventory management and resource allocation.

RMSE is the average deviation of the predictions from actual values, the smaller the better. This is very good at picking up major outliers and provides dependable and communicable forecasts that will help lower the chances of overstocking and understocking.

These metrics are highly useful for evaluating model accuracy and reliability in supply chains according to Douaioui et al. (2024), and Panda and Mohanty (2023). The two together provide a complementary look at predictive performance to support operational efficiency, decision-making, and minimize supply chain inefficiencies such as stockouts or overproduction.

CHAPTER FOUR: MODEL ANALYSIS AND RESULTS

4. Performance Assessment and Results

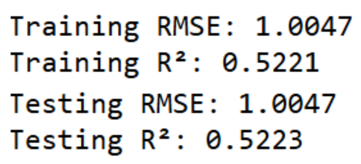
4.1. Linear Regression

4.1.1 Evaluation of Linear Regression

Linear regression is one of the simplest models it was decided to start with a look at its ability to predict the target variable Order item quantity in the context of the demand forecast. The code here initializes a linear regression model, trains it on the provided dataset , and assesses the performance on both the training and testing data by generating predictions to review the results in standard metrics, Root Mean Squared Error(RMSE) and R² score. This is a good baseline of how well a simple, interpretable model can perform with the given data.

4.1.2 Performance Metrics and Insights

The performance of the linear regression model was measured using RMSE and R² score. The RMSE for both the training and testing datasets was 1.0047, indicating that the predictive accuracy is consistent across both datasets. This consistency suggests that the model generalizes well to unseen data and does not suffer from overfitting. However, the R² scores of training and testing are 0.5221 and 0.5223, respectively, which means the model explains about 52% of the variance in the target variable. While this indicates moderate predictive power, it also highlights significant unexplained variability, suggesting that the linear regression model might be insufficient for capturing the complexities of the data.



```
Training RMSE: 1.0047
Training R²: 0.5221
Testing RMSE: 1.0047
Testing R²: 0.5223
```

Figure 30:Model Performance Metrics for Training and Testing - Linear Regression

4.1.3 Graph Analysis: Actual vs. Predicted Values

The graph below plots the actual(blue line) versus the predicted (orange dashes). It shows how the linear regression model aligns reasonably with the actual but cannot catch the extreme peaks and troughs, because it underpredicts high demands and deviates during low demand periods. This deficiency arises because the model cannot capture non-linear trends. Advanced models such as decision trees and gradient boosting can thus be recommended to enhance the forecast accuracy of the order item quantity.

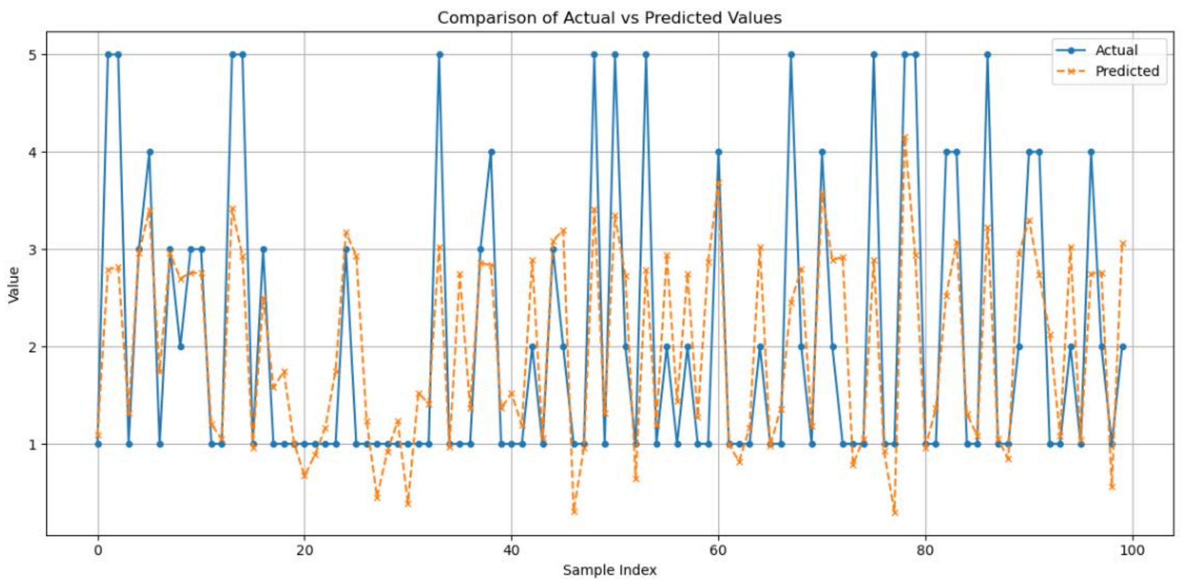


Figure 31:Line chart showing Predicted Vs Actual Values - Linear Regression

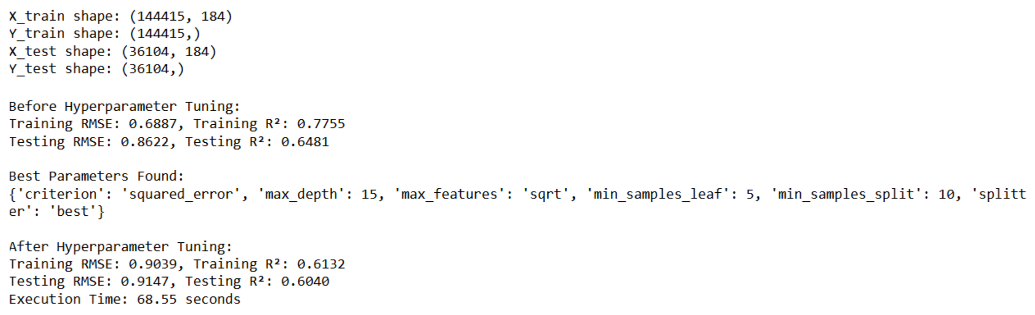
In our previous machine learning algorithm, Linear Regression, we saw that the model did not handle non-linear trends well and gave poor performance while predicting complex patterns in the data. To address this advanced model such as Decision Trees were tested to improve the forecast of Order Item Quantity.

4.2 Decision tree

4.2.1 Evaluation of Decision tree

The Decision Tree Regressor was evaluated in two stages, before and after the hyperparameter tuning. Before tuning, the model gave us a training RMSE of 0.6887 and an R^2 of 0.7755, which meant that the model explained 77.55% of the variance in the training data. It achieved RMSE and R^2 of 0.8622 and 0.6481 respectively, quantifying 64.81% of variance in the testing set. These numbers imply decent performance but signal overfitting observing the gap between training and test metrics.

The best parameters found after tuning hyperparameters were {Criterion: squared_error, max_depth: 15, max_features: sqrt, min_samples_leaf: 5, min_samples_split: 10, splitter: best}. This resulted in a RMSE of 0.9039 and R^2 of 0.6132 on the training set and 0.9147 and 0.6040 on the testing set. While the overall performance indicators saw a slight drop, the closer correlation between training and testing metrics indicated reduced overfitting and improved generalization as we introduced regularization by controlling things like maximum tree depth and minimum sample size.



```
X_train shape: (144415, 184)
Y_train shape: (144415,)
X_test shape: (36104, 184)
Y_test shape: (36104,)

Before Hyperparameter Tuning:
Training RMSE: 0.6887, Training R²: 0.7755
Testing RMSE: 0.8622, Testing R²: 0.6481

Best Parameters Found:
{'criterion': 'squared_error', 'max_depth': 15, 'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 10, 'splitt
er': 'best'}

After Hyperparameter Tuning:
Training RMSE: 0.9039, Training R²: 0.6132
Testing RMSE: 0.9147, Testing R²: 0.6040
Execution Time: 68.55 seconds
```

Figure 32: Model Performance Before & After Hyperparameter tunning - Decision Tree

4.2.2 Performance Metrics and Insights

The performance metrics highlight the Decision Tree's effectiveness. While RMSE slightly increased after tuning due to regularization, it led to a more robust model for unseen data. The R² scores decreased but showed better balance between training and testing. Unlike Linear Regression, which failed to capture non-linear patterns, Decision Trees effectively modelled the data's complexity. Initial overfitting in the default model was mitigated through hyperparameter tuning.

4.2.3 Graph Analysis: Actual vs. Predicted Values

The graph comparing actual and predicted values demonstrates the Decision Tree's ability to capture non-linear trends. The alignment between actual and predicted values shows the model captures general patterns better than Linear Regression. However, the model struggles with some extreme values, as seen in the peaks and valleys. Predicted values are smoother, especially at extreme ranges, due to regularization constraints. Overall, the Decision Tree provides a significant improvement over Linear Regression in forecasting accuracy, though further refinement is possible. To enhance performance, ensemble methods such as **Random Forests** or **Gradient Boosting** could be considered. This analysis demonstrates that Decision Trees are a meaningful step forward in improving predictive accuracy while addressing the limitations of simpler models like Linear Regression.

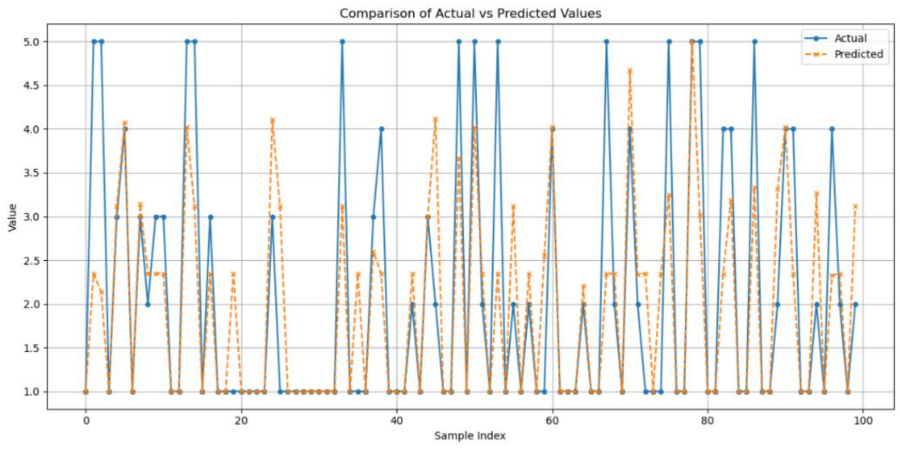


Figure 33: Line chart showing Predicted Vs Actual Values - Decision Tree

4.2.4 Feature Importance

Feature importance analysis shows how much each feature impacts a model's predictions, helping identify the most important factors for the target variable.

According to the feature importance analysis, **Product Price (0.56485) and Order Item Discount (0.28831)** are the most important variables for determining order quantities, highlighting strong price sensitivity and responsiveness to discounts. Monthly Department Count (0.05794) and order_month (0.03192) indicating some influence from departmental activity and seasonal trends. The other features like order_year, Department Name, Category Name, Product Name etc., do not really have much effect. Overall, pricing-related factors dominate, contributing to over 80% of the predictive power, suggesting that focusing on pricing and discounts can significantly improve predictions of order item quantities.

```
Feature: Product Price, Score: 0.56485
Feature: Order Item Discount, Score: 0.28831
Feature: order_month, Score: 0.03192
Feature: order_year, Score: 0.01431
Feature: Monthly_Department_Count, Score: 0.05794
Feature: Department Name, Score: 0.01197
Feature: Category Name, Score: 0.01262
Feature: Product Name, Score: 0.01809
```

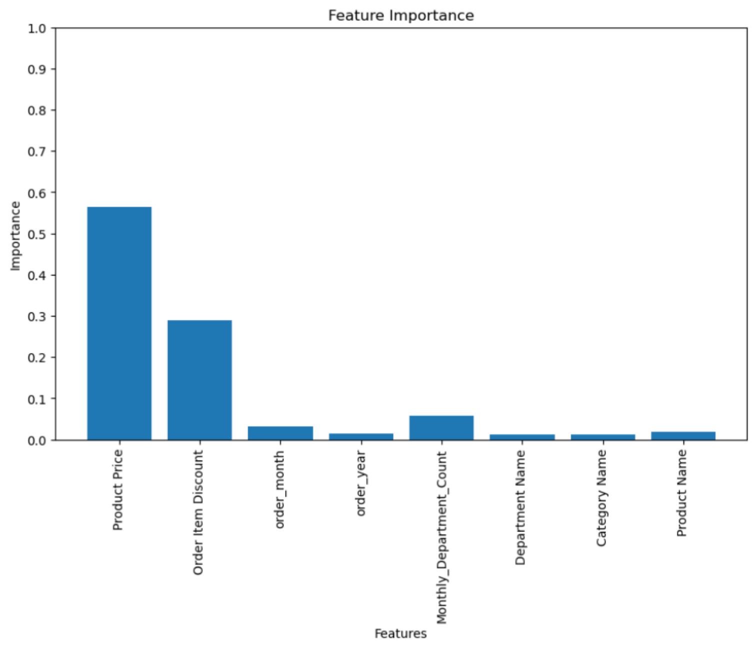


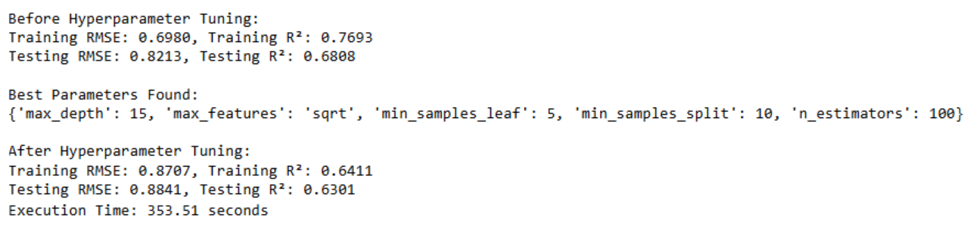
Figure 34: Feature Importance for Predicting Order Item Quantity - Decision tree

4.3 Random Forest

Earlier, we used the Decision Tree Regressor and found that it was able to capture non-linear trends, but it also overfits and is sensitive to outliers. To overcome these problems, we examined Random Forest Regressor, an ensemble method which constructs various decision trees and averages their predictions to make them more robust.

4.3.1 Evaluation of Random Forest

The Random Forest-based model recorded a training RMSE of 0.6980, R^2 of 0.7693, testing RMSE of 0.8213, and R^2 of 0.6808 before tuning, resulting in a model that is able to capture the end of the distribution with some overfitting, as expected in non-linear-based models. The best parameters after hyperparameter tuning were {max_depth: 15, max_features: 'sqrt', min_samples_leaf: 5, min_samples_split: 10, n_estimators: 100}, which gave a training RMSE of 0.8707, R^2 of 0.6411, and a testing RMSE of 0.8841 and R^2 of 0.6301. This hyperparameter tuning decreased overfitting and improved generalization, although it did cause a slight decrease in the predictive accuracy of the model on the training set. This means Random Forest is more stable and generalized with respect to a single Decision Tree because it is an ensemble method.



```
Before Hyperparameter Tuning:
Training RMSE: 0.6980, Training R²: 0.7693
Testing RMSE: 0.8213, Testing R²: 0.6808

Best Parameters Found:
{'max_depth': 15, 'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

After Hyperparameter Tuning:
Training RMSE: 0.8707, Training R²: 0.6411
Testing RMSE: 0.8841, Testing R²: 0.6301
Execution Time: 353.51 seconds
```

Figure 35: Model Performance Before & After Hyperparameter tunning - Random Forest

4.3.2 Performance Metrics and Insights

RMSE values quantify the average error, showing slightly higher values after tuning due to regularization, while the R² scores decreased but achieved better training-testing balance. Random Forest's ability to capture non-linear relationships makes it superior to simpler models like Linear Regression, and it provides greater robustness compared to a single Decision Tree. The trade-off introduced by regularization resulted in a more generalizable model, even though predictive accuracy slightly declined. For better performance, exploring Gradient Boosting could be a promising next step, as it often performs well on complex datasets by iteratively minimizing prediction errors.

4.3.3 Graph Analysis: Actual vs. Predicted Values

The graph shows that predictions align well with actual values, but the model struggles with extreme peaks and exhibits a smoothing effect due to its ensemble nature. While the Random Forest effectively captures non-linear patterns, the smoothing reduces noise but can underestimate sharp spikes in actual values. Overall, the graph reflects the model's strength in capturing general trends but highlights opportunities for improvement in handling variability. Trying **Gradient Boosting** may address some of these limitations by boosting performance on under-predicted data points.

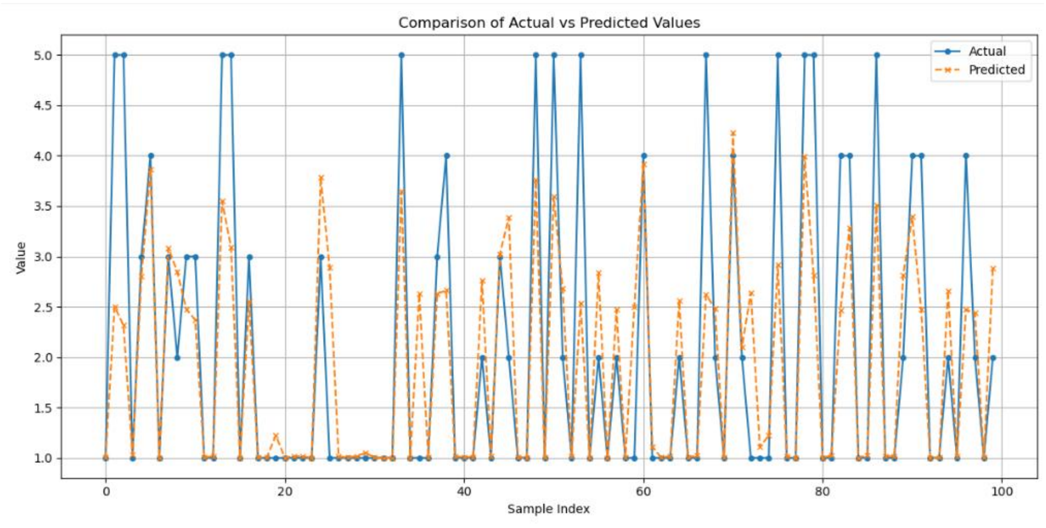
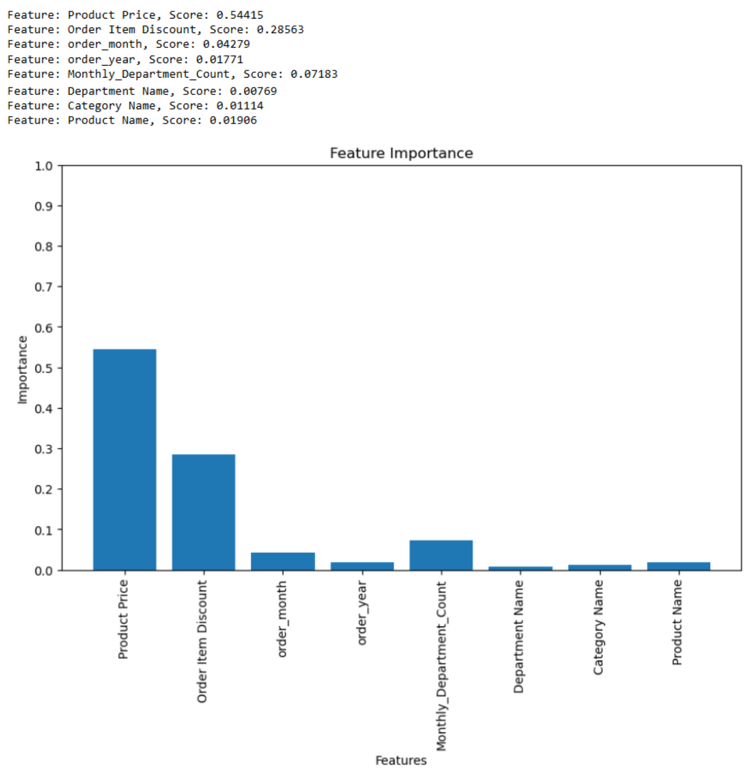


Figure 36: Line chart showing Predicted Vs Actual Values - Random Forest

4.3.4 Feature Importance

The most impactful features for the Order Item Quantity features are Product Price (0.54415) and Order Item Discount (0.28563) have the most significant impact, which goes without saying. Approximately less important features are order_month, order_year, Product Name, Department Name, Category Name (0.07183) and are of moderate importance. Random Forest makes price-related features weigh the most, but adds small features without much impact, so it learns more about character features, so it considers a more comprehensive range of features.



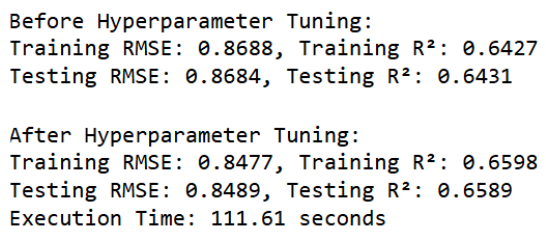
```
Feature: Product Price, Score: 0.54415
Feature: Order Item Discount, Score: 0.28563
Feature: order_month, Score: 0.04279
Feature: order_year, Score: 0.01771
Feature: Monthly_Department_Count, Score: 0.07183
Feature: Department Name, Score: 0.00769
Feature: Category Name, Score: 0.01114
Feature: Product Name, Score: 0.01906
```

Figure 37: Feature Importance for Predicting Order Item Quantity - Random Forest

4.4 Gradient Boosting

4.4.1 Evaluation of Gradient Boosting

Previously, we tried the **Random Forest**, which performed well in capturing non-linear trends but struggled with balancing training and testing performance. To address this, we evaluated the **Gradient Boosting**, which iteratively improves predictions by correcting errors from previous models. Before hyperparameter tuning, the Gradient Boosting model achieved a training RMSE of 0.8688 and R^2 of 0.6427, with a testing RMSE of 0.8684 and R^2 of 0.6431, showing reasonable performance with good training-testing balance. After tuning, the model improved slightly, achieving a training RMSE of 0.8477 and R^2 of 0.6598, and a testing RMSE of 0.8489 and R² of 0.6589, with an execution time of 111.61 seconds. Tuning optimized parameters like learning rate, number of estimators, and subsampling, enhancing generalization and predictive performance.



```
Before Hyperparameter Tuning:
Training RMSE: 0.8688, Training R²: 0.6427
Testing RMSE: 0.8684, Testing R²: 0.6431

After Hyperparameter Tuning:
Training RMSE: 0.8477, Training R²: 0.6598
Testing RMSE: 0.8489, Testing R²: 0.6589
Execution Time: 111.61 seconds
```

Figure 38: Model Performance Before & After Hyperparameter tunning - Gradient Boosting

4.4.2 Performance Metrics and Insights

The model's RMSE decreased slightly after tuning, indicating improved accuracy, while the R^2 scores increased, with the testing R^2 rising from 0.6431 to 0.6589, showing the model better explains the variance in **Order Item Quantity**. Gradient Boosting effectively captures complex relationships in the data, refining predictions with each iteration and reducing residual errors. The tuned model achieves better training-testing balance compared to the default model and Random Forest, making it more reliable for unseen data.

4.4.3 Graph Analysis: Actual vs. Predicted Values

The graph of actual vs. predicted values shows that predictions closely align with actual values, reflecting the model's ability to capture general trends effectively. It performs well in predicting peaks and valleys, with better accuracy and fewer mismatches compared to the Decision Tree and Random Forest models. While slight mismatches are observed in extreme cases, Gradient Boosting outperforms the other two models in handling variability and non-linear trends. The predictions also exhibit a smoothing effect, characteristic of Gradient Boosting's regularization, which reduces noise but may underestimate sharp spikes in order quantities. Overall, Gradient Boosting delivers the best performance among the three models, offering superior generalization and accuracy, though further optimization or feature engineering could improve its handling of outliers.

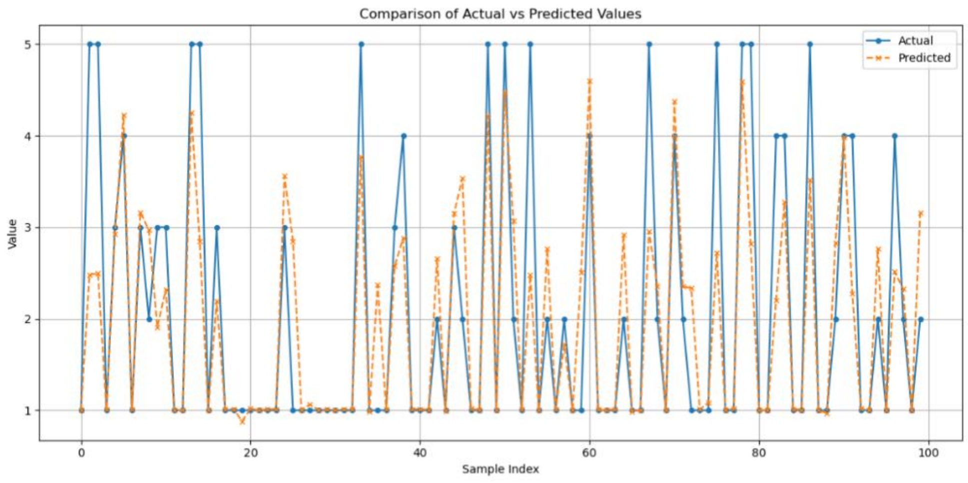


Figure 39: Line chart showing Predicted Vs Actual Values - Gradient Boosting

4.4.4 Feature Importance

Product Price (0.69728) is the most dominant predictor of order item quantity, followed by Order Item Discount (0.27905). Other factors, such as Monthly Department Count, order_year, order_month, Product Name, Department Name, and Category Name, have minimal impact on the prediction. Gradient Boosting relies heavily on the top predictors, focusing on simplicity and efficiency by minimizing the influence of less relevant features. This makes it highly effective for scenarios where top-performing variables, such as price and discounts, drive the primary decisions.

```
Feature: Product Price, Score: 0.69728
Feature: Order Item Discount, Score: 0.27905
Feature: order_month, Score: 0.00005
Feature: order_year, Score: 0.00404
Feature: Monthly_Department_Count, Score: 0.00148
Feature: Department Name, Score: 0.01160
Feature: Category Name, Score: 0.00119
Feature: Product Name, Score: 0.00533
```

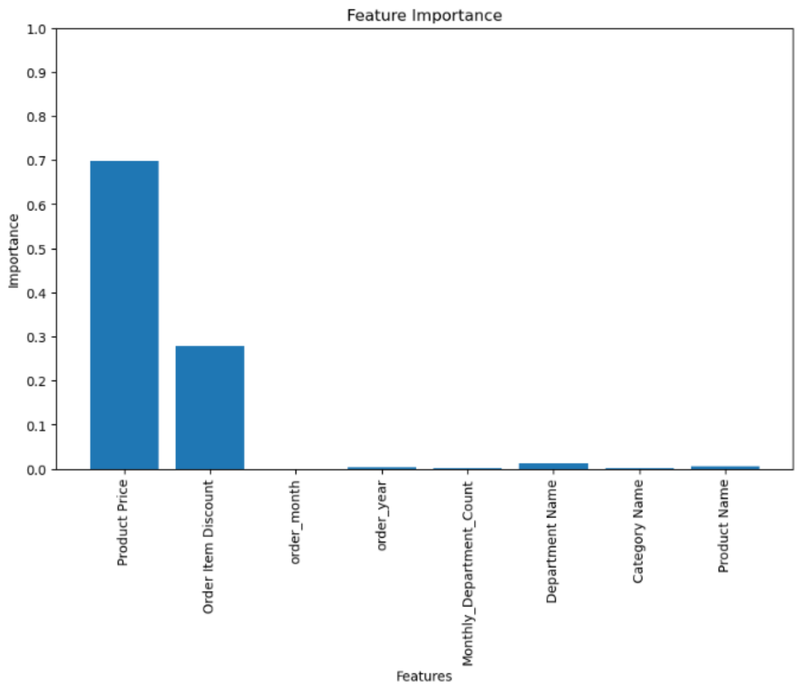


Figure 40: Feature Importance for Predicting Order Item Quantity - Gradient Boosting

4.5 Evaluation metrics

In this section, we will look into the performance of evaluation metrics for the ML algorithms used to predict order item quantity, with Linear Regression considered as the baseline model. The important variables include Department Name, Category Name, Product Name, Product Price, and Order Item Discount, while the derived variables are Order Month, Order Year, and Monthly Department Count.

Table 2: Model Performance Comparison Based on Evaluation Metrics

Evaluation metrics	**Linear regression**	**Decision tree**	**Random forest**	**Gradient boosting**
RMSE	1.0047	0.9147	0.8841	0.8489
R^2	0.5223	0.6040	0.6301	0.6589

Among the models, **Gradient Boosting** achieves the best performance with the lowest RMSE (0.8489) and the highest R^2 (0.6589), making it the most accurate and reliable model. It effectively captures complex relationships between variables by combining predictions from multiple weak learners, which results in superior performance. **Random Forest** comes close with an RMSE of 0.8841 and R^2 of 0.6301 but falls short of Gradient Boosting's refinement. **Decision Tree** shows moderate performance, and **Linear Regression**, the baseline, has the least accuracy. Gradient Boosting's ability to handle non-linear relationships and minimize prediction errors makes it ideal for this task.

CHAPTER FIVE: SUMMARY OF FINDINGS

5.Summary

In the previous section of the study, we evaluated four machine learning algorithms Linear Regression, Decision Tree, Random Forest, and Gradient Boosting to forecast order item quantity. Linear Regression was chosen as the baseline model due to its simplicity and ease of interpretation.

Linear Regression, as the baseline, provided moderate predictive power, with an RMSE of 1.0047 and R^2 of 0.5223, but struggled to capture non-linear relationships in the data. **Decision Tree** improved upon this, leveraging its ability to handle non-linearity, yielding an RMSE of 0.9147 and R^2 of 0.6040. However, its initial overfitting was addressed through hyperparameter tuning.

Random Forest, an ensemble method, outperformed Decision Tree with an RMSE of 0.8841 and R^2 of 0.6301, offering greater robustness and generalization. Yet, it lacked the refinement needed to fully capture the intricate relationships in the data.

Gradient Boosting emerged as the best model, achieving the lowest RMSE (0.8489) and highest R^2 (0.6589). Its iterative nature allows it to correct errors from previous predictions, effectively handling complex patterns and non-linear interactions. Gradient Boosting's reliance on the most significant predictors ensures its precision while minimizing the influence of less impactful features. This combination of accuracy, generalization, and interpretability makes Gradient Boosting the most suitable model for demand forecasting of order item quantity, aligning perfectly with the aim of providing reliable predictions to optimize inventory management and supply chain decisions.

5.1 Demand Forecasting by the best model

After evaluating four machine learning models, Linear Regression, Decision Tree, Random Forest, and Gradient Boosting. It is clear that **Gradient Boosting** is the most suitable algorithm for demand forecasting of order item quantity in this study.

Gradient Boosting delivers the best performance with the lowest RMSE (0.8489) and the highest R^2 (0.6589), making it the most accurate and reliable model for this task. Its ability to effectively handle complex, non-linear relationships within the supply chain data, while capturing the nuanced impact of critical variables like product price, discounts, and seasonal trends, sets it apart. This capability ensures precise forecasting and actionable insights, essential for improving inventory management and reducing operational inefficiencies.

Using these predictions for inventory management can revolutionize operations by ensuring optimal stock levels. For instance, the model’s ability to identify demand spikes in high-performing categories like "Fan Shop" allows businesses to prepare inventory in advance, ensuring customer satisfaction during peak seasons. Conversely, it can also identify underperforming departments, enabling proactive adjustments to avoid overstocking. By incorporating real-time data, such as recent sales trends and customer feedback, the model’s predictions can be refined further, allowing dynamic inventory adjustments. This not only minimizes operational costs but also enhances customer satisfaction, ultimately driving sustainable growth and efficiency in the supply chain ecosystem.

CHAPTER SIX: CONCLUSIONS

6.Conclusion

This study showcases the potential of such models while addressing limitations, offering recommendations, and exploring opportunities for improvement.

6.1 Limitations

The dataset used in this study spans from 2015 to 2018, which, while valuable, excludes recent trends and may not fully capture current supply chain dynamics. Missing values in critical columns, such as "Order Zip Code" and "Product Description," reduced the dataset’s completeness, potentially affecting prediction accuracy. Additionally, the absence of key variables like customer sentiment and seasonal trends limits the depth of insights derived from the analysis.

From a modelling perspective, Linear Regression struggled to capture non-linear dependencies, making it less effective for complex relationships. Decision Trees and Random Forest models showed promise but initially overfitted the training data, though hyperparameter tuning mitigated this to some extent. Gradient Boosting emerged as the best performer but underestimated extreme peaks and troughs in demand, highlighting the need for further refinement. Computational constraints were another challenge, as advanced models like Gradient Boosting required significant resources, particularly during hyperparameter optimization. Lastly, the limited impact of certain features, such as "Order Year" and "Monthly Department Count," underscores the need for better feature selection and engineering.

6.2 Recommendations

To improve demand forecasting, several actionable steps can be taken. First, data enrichment and quality improvements are essential. Including recent and comprehensive data, such as customer reviews, ratings, and economic indicators, would provide a broader context for analysis. Addressing missing values through advanced imputation techniques can enhance data completeness, ensuring the dataset's reliability. Additionally, model performance can be improved by exploring advanced techniques such as neural networks, hybrid ensemble models, and stacking, which can capture complex patterns and relationships more effectively. Feature engineering remains a critical area, with potential improvements including the addition of variables related to promotional campaigns, competitor pricing, and real-time inventory levels, as well as the development of interaction terms and polynomial features to better represent non-linear relationships.

Operationally, real-time analytics systems should be implemented to facilitate continuous data collection and model retraining, ensuring that forecasts remain adaptive to changing conditions. Pricing and discount strategies should be prioritized, given their significant influence on demand. Targeted marketing campaigns can be designed to maximize revenue in high-performing categories, such as apparel and sports equipment, while focusing on strategies to improve sales in underperforming areas like books and beauty products. Insights from our study emphasize the importance of factors such as product pricing, order discounts, and monthly department activity, which should guide strategic decision-making. High-performing categories, such as "Fan Shop" and "Apparel," provide significant opportunities for targeted inventory optimization, while real-time analysis of high-demand products can further enhance customer satisfaction and revenue.

6.3 Future Scope

Future research and development should prioritize expanding datasets to cover at least five years, incorporating variables like weather patterns and market trends to improve model accuracy and relevance. Advanced modelling techniques, including time-series approaches such as LSTMs or transformers, should be explored to better capture dynamic trends and patterns. Reinforcement learning also holds potential for optimizing supply chain decisions in real-time by adapting to changes in demand and supply conditions.

Moreover, developing scalable, real-time demand forecasting systems capable of processing streaming data can revolutionize inventory management by enabling just-in-time stocking and reducing wastage. Expanding research to include regional testing while factoring in transportation infrastructure and cultural preferences will enhance the generalizability and applicability of the models to diverse markets. These initiatives will ensure that forecasting systems remain robust, scalable, and capable of addressing the complex challenges of modern supply chains.

6.4 Concluding Perspectives

Effective demand forecasting directly supports inventory management by aligning stock levels with customer demand and reducing the costs of overstocking or stockouts. By addressing the limitations identified in our study, implementing the proposed recommendations, and exploring future opportunities, businesses can achieve more accurate and actionable insights. These improvements will enhance operational efficiency, strategic decision-making, and customer satisfaction, ultimately fostering sustainable growth

REFERENCES

Douaioui, K., Oucheikh, R., Benmoussa, O., and Mabrouki, C. (2024) *Machine Learning and Deep Learning Models for Demand Forecasting in Supply Chain Management: A Critical Review*. Applied System Innovation, 7(93).

Available at: https://doi.org/10.3390/asi7050093

Zhang, Y. (2024) *Machine Learning Applied in Demand Forecasting and Supply Planning*. Master's Thesis, Metropolia University of Applied Sciences.

Available at: https://metropolia.fi

Panda, S. K. & Mohanty, S. N. (2023) *Time Series Forecasting and Modeling of Food Demand Supply Chain Based on Regressors Analysis*. IEEE Access.

Available at: https://doi.org/10.1109/ACCESS.2023.3266275

Nasseri, M., Falatouri, T., Brandtner, P. and Darbanian, F. (2023) *Applying Machine Learning in Retail Demand Prediction—A Comparison of Tree-Based Ensembles and Long Short-Term Memory-Based Deep Learning*. Applied Sciences, 13(11112).

Available at: https://doi.org/10.3390/app131911112

Vairagade, N., Logofatu, D., Leon, F. and Muharemi, F. (2019) Demand Forecasting Using Random Forest and Artificial Neural Network for Supply Chain Management. In: N. T. Nguyen et al. (Eds.), Lecture Notes in Artificial Intelligence, vol. 11683. Springer, pp. 328–339.

Available at: https://doi.org/10.1007/978-3-030-28377-3_27

Terrada, L., El Khaili, M. and Ouajji, H. (2022) *Demand Forecasting Model using Deep Learning Methods for Supply Chain Management 4.0*. International Journal of Advanced Computer Science and Applications, 13(5), pp. 704-711.

Available at: https://doi.org/10.14569/IJACSA.2022.0130510

Shakir, T. & Modupe, A. (2023) *Enhancing Demand Forecasting in Retail Supply Chains: A Machine Learning Regression Approach*. Global Journal of Management and Business Research, 23(8), pp. 1-14.

Available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Zhang, Y., Zhu, H., Wang, Y., and Li, T. (2022) *Demand Forecasting: From Machine Learning to Ensemble Learning*. IEEE Conference on Telecommunications, Optics, and Computer Science (TOCS), pp. 461-466.

Available at: https://doi.org/10.1109/TOCS56154.2022.10015992

Benkachcha, S., Benhra, J., and El Hassani, H. (2014) *Demand Forecasting in Supply Chain: Comparing Multiple Linear Regression and Artificial Neural Networks Approaches*. International Review on Modelling and Simulations (IREMOS), 7(4), pp. 1-15.

Available at: https://doi.org/10.1109/IREMOS.2014.269984405

Zheng, Y. (2024) *Application of Machine Learning Algorithms in Enterprise Supply Chain Demand Forecasting*. Proceedings of the 2nd International Conference on Design Science (ICDS), pp. 1-12.

IEEE. Available at: https://doi.org/10.1109/ICDS62420.2024.10751682

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., & Jupyter Development Team, 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In: F. Loizides & B. Schmidt, eds. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, pp. 87–90.

Available at: https://eprints.soton.ac.uk/403913/

Khurana, R., and Kaul, D., 2022. AI-Driven Optimization Models for E-commerce Supply Chain Operations: Demand Prediction, Inventory Management, and Delivery Time Reduction with Cost Efficiency Considerations. *International Journal of Social Analytics*, 7(12), pp.1–20.

Available at: https://www.researchgate.net/publication/386734670

Babaee Tirkolaee, E., Sadeghi, S., Mansoori Mooseloo, F., Rezaei Vandchali, H., and Aeini, S., 2021. Application of Machine Learning in Supply Chain Management: A Comprehensive Overview of the Main Areas. *Mathematical Problems in Engineering*, 2021, Article ID 1476043.

Available at: https://doi.org/10.1155/2021/1476043

Krishnamoorthy, G., Jeyaraman, J., Konidena, B.K., and Reddy, A.P., 2024. Machine Learning for Demand Forecasting in Manufacturing. *International Journal for Multidisciplinary Research*, 6(1), pp.1–15.

Available at: https://www.researchgate.net/publication/380030267

Farzana, S., and Prakash, N., 2020. Machine Learning in Demand Forecasting: A Review. In: *Second International Conference on IoT, Social, Mobile, Analytics & Cloud in Computational Vision & Bio-Engineering (ISMAC-CVB 2020)*.

Available at: https://ssrn.com/abstract=3733548

Aamer, A.M., Yani, L.P.E., and Priyatna, I.M.A., 2021. Data Analytics in the Supply Chain Management: Review of Machine Learning Applications in Demand Forecasting. *Operations and Supply Chain Management*, 14(1), pp.1–13.

Available at: https://doi.org/10.1155/2021/1476043

Adewusi, A.O., Komolafe, A.M., Ejairu, E., Aderotoye, I.A., Abiona, O.O., and Oyeniran, O.C., 2024. The Role of Predictive Analytics in Optimizing Supply Chain Resilience: A Review of Techniques and Case Studies. *International Journal of Management & Entrepreneurship Research*, 6(3), pp.815–837.

Available at: https://www.fepbl.com/index.php/ijmer

Khedr, A.M., and Rani, S.S., 2024. Enhancing Supply Chain Management with Deep Learning and Machine Learning Techniques: A Review. *Journal of Open Innovation: Technology, Market, and Complexity*, 10, p.100379.

Available at: https://doi.org/10.1016/j.joitmc.2024.100379

Seyedan, S., 2023. Development of Predictive Analytics for Demand Forecasting and Inventory Management in Supply Chain Using Machine Learning Techniques. PhD Thesis. Concordia University.

Available at: https://spectrum.library.concordia.ca/

Wilson, G., Johnson, O., and Brown, W., 2024. The Role of Machine Learning in Predictive Analytics for Supply Chain Management.

Available at: https://doi.org/10.20944/preprints202408.0343.v1

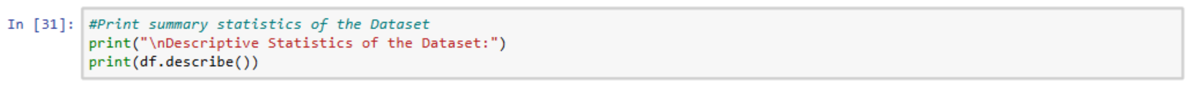
Seyedan, S., and Mafakheri, F., 2020. Predictive Big Data Analytics for Supply Chain Demand Forecasting: Methods, Applications, and Research Opportunities. *Journal of Big Data*, 7(53), pp.1–22.

Available at: https://doi.org/10.1186/s40537-020-00329-2

APPENDICES

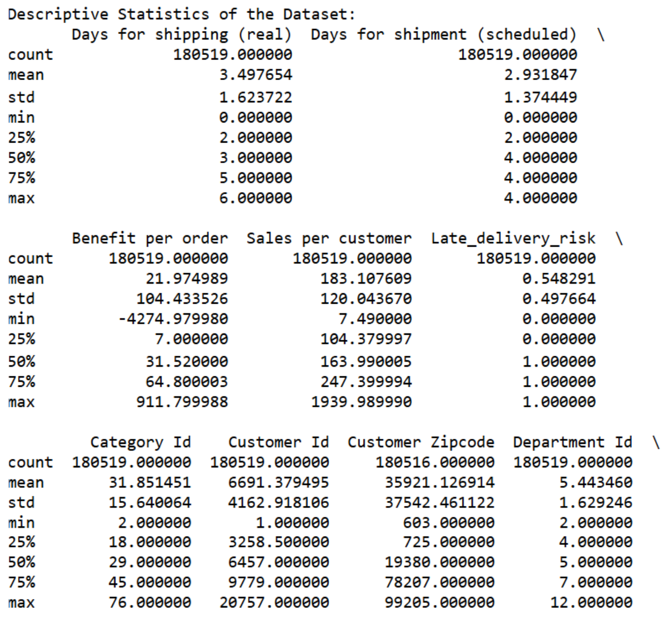
Appendix A- Descriptive Statistics

1. Descriptive statistics Code

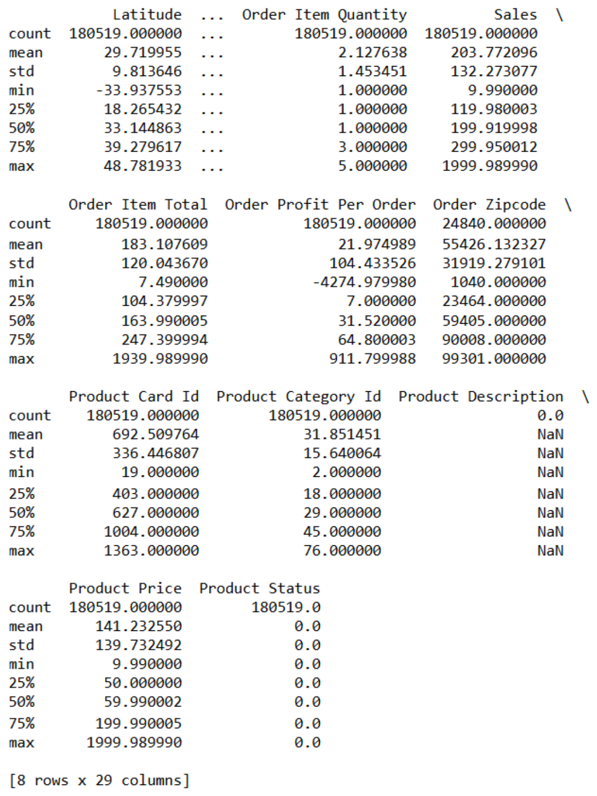


```
In [31]: #Print summary statistics of the Dataset
         print("\nDescriptive Statistics of the Dataset:")
         print(df.describe())
```

2. Descriptive statistics of all variables



```
Descriptive Statistics of the Dataset:
       Days for shipping (real)  Days for shipment (scheduled)  \
count             180519.000000                  180519.000000
mean                   3.497654                       2.931847
std                    1.623722                       1.374449
min                    0.000000                       0.000000
25%                    2.000000                       2.000000
50%                    3.000000                       4.000000
75%                    5.000000                       4.000000
max                    6.000000                       4.000000

       Benefit per order  Sales per customer  Late_delivery_risk  \
count      180519.000000       180519.000000       180519.000000
mean           21.974989          183.107609            0.548291
std           104.433526          120.043670            0.497664
min         -4274.979980            7.490000            0.000000
25%             7.000000          104.379997            0.000000
50%            31.520000          163.990005            1.000000
75%            64.800003          247.399994            1.000000
max           911.799988         1939.989990            1.000000

         Category Id    Customer Id  Customer Zipcode  Department Id  \
count  180519.000000  180519.000000     180516.000000  180519.000000
mean       31.851451    6691.379495      35921.126914       5.443460
std        15.640064    4162.918106      37542.461122       1.629246
min         2.000000       1.000000        603.000000       2.000000
25%        18.000000    3258.500000        725.000000       4.000000
50%        29.000000    6457.000000      19380.000000       5.000000
75%        45.000000    9779.000000      78207.000000       7.000000
max        76.000000   20757.000000      99205.000000      12.000000
```

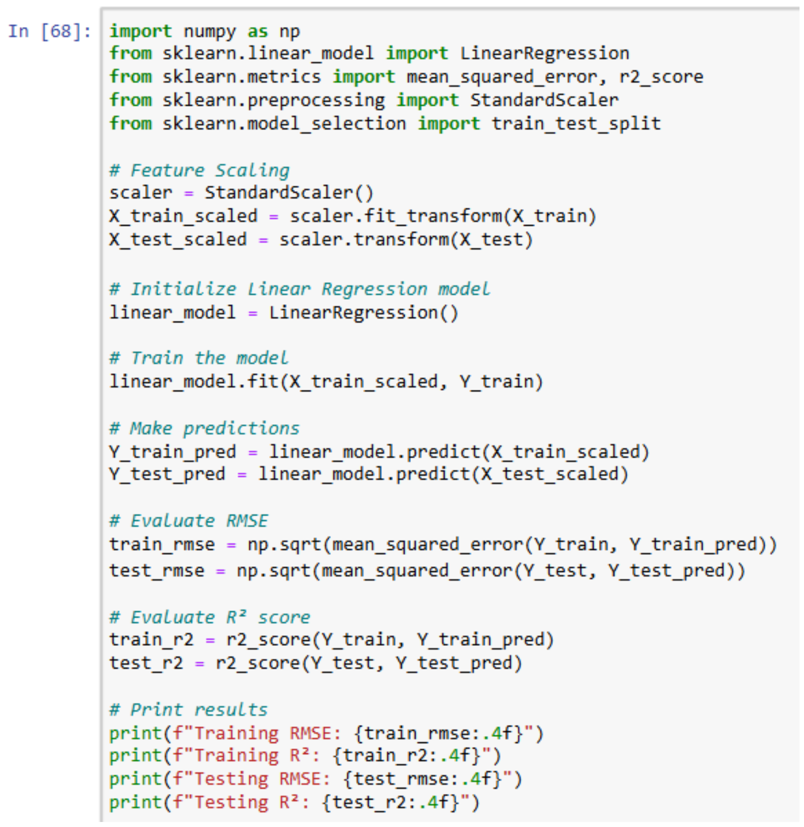


```
            Latitude  ...  Order Item Quantity          Sales  \
count  180519.000000  ...        180519.000000  180519.000000
mean       29.719955  ...             2.127638     203.772096
std         9.813646  ...             1.453451     132.273077
min       -33.937553  ...             1.000000       9.990000
25%        18.265432  ...             1.000000     119.980003
50%        33.144863  ...             1.000000     199.919998
75%        39.279617  ...             3.000000     299.950012
max        48.781933  ...             5.000000    1999.989990

       Order Item Total  Order Profit Per Order  Order Zipcode  \
count     180519.000000           180519.000000   24840.000000
mean         183.107609               21.974989   55426.132327
std          120.043670              104.433526   31919.279101
min            7.490000            -4274.979980    1040.000000
25%          104.379997                7.000000   23464.000000
50%          163.990005               31.520000   59405.000000
75%          247.399994               64.800003   90008.000000
max         1939.989990              911.799988   99301.000000

       Product Card Id  Product Category Id  Product Description  \
count    180519.000000        180519.000000                  0.0
mean        692.509764            31.851451                  NaN
std         336.446807            15.640064                  NaN
min          19.000000             2.000000                  NaN
25%         403.000000            18.000000                  NaN
50%         627.000000            29.000000                  NaN
75%        1004.000000            45.000000                  NaN
max        1363.000000            76.000000                  NaN

       Product Price  Product Status
count  180519.000000        180519.0
mean      141.232550             0.0
std       139.732492             0.0
min         9.990000             0.0
25%        50.000000             0.0
50%        59.990002             0.0
75%       199.990005             0.0
max      1999.989990             0.0

[8 rows x 29 columns]
```

Appendix B - Model Performance, Value Comparison, and Feature Importance

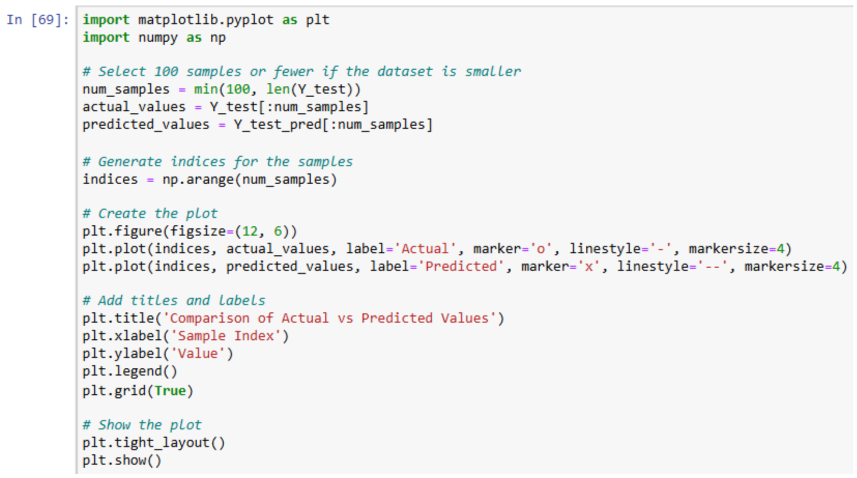
1. Linear Regression

1.1. Model Performance Code



```
In [68]: import numpy as np
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error, r2_score
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split

         # Feature Scaling
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         # Initialize Linear Regression model
         linear_model = LinearRegression()

         # Train the model
         linear_model.fit(X_train_scaled, Y_train)

         # Make predictions
         Y_train_pred = linear_model.predict(X_train_scaled)
         Y_test_pred = linear_model.predict(X_test_scaled)

         # Evaluate RMSE
         train_rmse = np.sqrt(mean_squared_error(Y_train, Y_train_pred))
         test_rmse = np.sqrt(mean_squared_error(Y_test, Y_test_pred))

         # Evaluate R² score
         train_r2 = r2_score(Y_train, Y_train_pred)
         test_r2 = r2_score(Y_test, Y_test_pred)

         # Print results
         print(f"Training RMSE: {train_rmse:.4f}")
         print(f"Training R²: {train_r2:.4f}")
         print(f"Testing RMSE: {test_rmse:.4f}")
         print(f"Testing R²: {test_r2:.4f}")
```

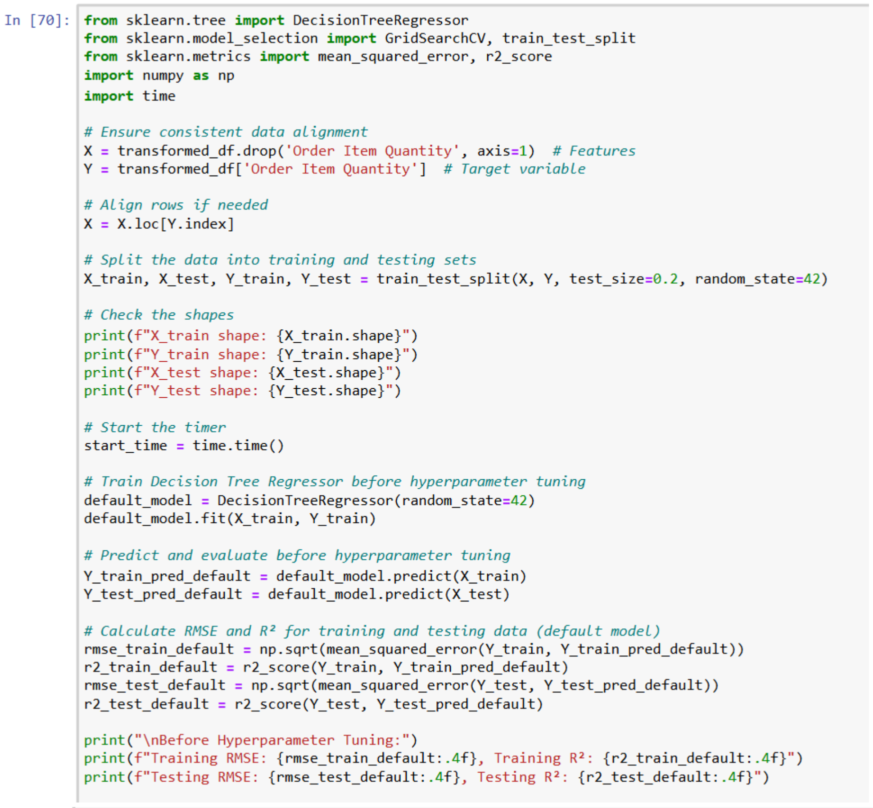
1.2. Code for Line Chart of Predicted vs. Actual Values



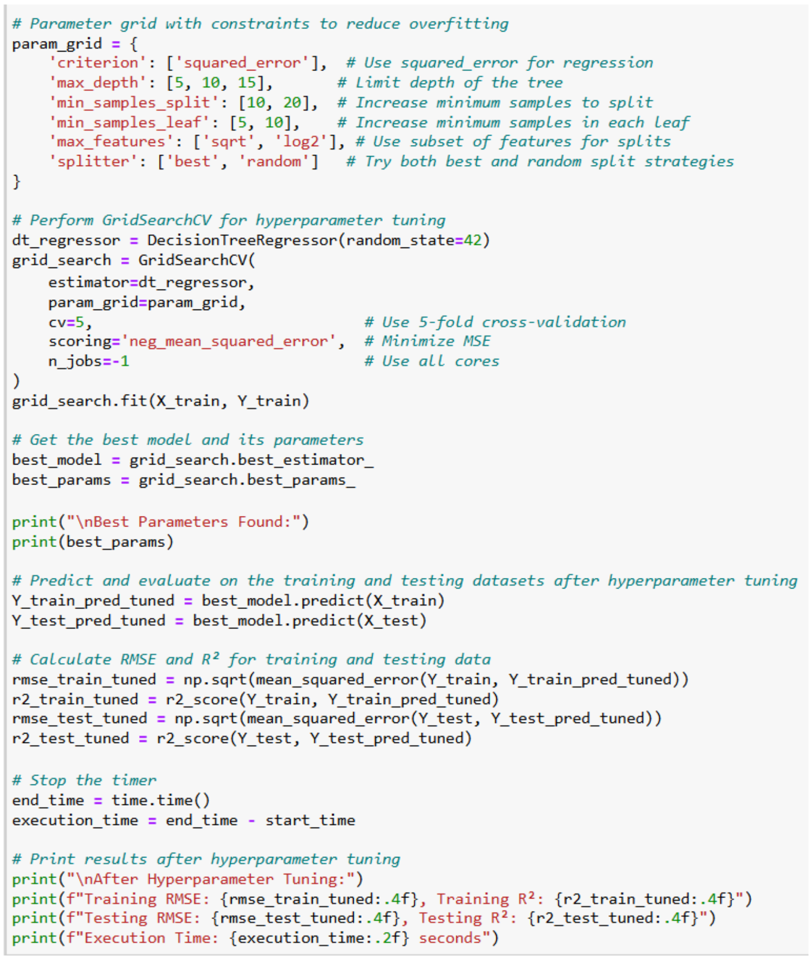
```
In [69]: import matplotlib.pyplot as plt
         import numpy as np

         # Select 100 samples or fewer if the dataset is smaller
         num_samples = min(100, len(Y_test))
         actual_values = Y_test[:num_samples]
         predicted_values = Y_test_pred[:num_samples]

         # Generate indices for the samples
         indices = np.arange(num_samples)

         # Create the plot
         plt.figure(figsize=(12, 6))
         plt.plot(indices, actual_values, label='Actual', marker='o', linestyle='-', markersize=4)
         plt.plot(indices, predicted_values, label='Predicted', marker='x', linestyle='--', markersize=4)

         # Add titles and labels
         plt.title('Comparison of Actual vs Predicted Values')
         plt.xlabel('Sample Index')
         plt.ylabel('Value')
         plt.legend()
         plt.grid(True)

         # Show the plot
         plt.tight_layout()
         plt.show()
```

2. Decision tree

2.1. Model Performance Code

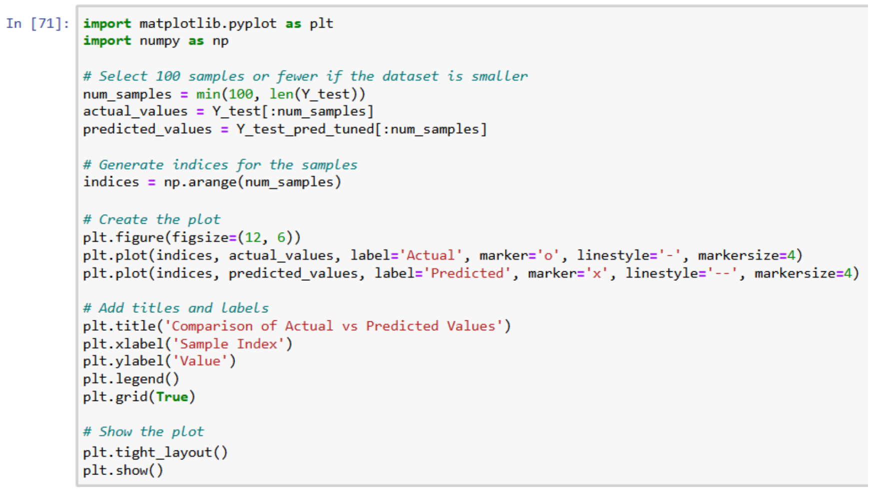


```
In [70]: from sklearn.tree import DecisionTreeRegressor
         from sklearn.model_selection import GridSearchCV, train_test_split
         from sklearn.metrics import mean_squared_error, r2_score
         import numpy as np
         import time

         # Ensure consistent data alignment
         X = transformed_df.drop('Order Item Quantity', axis=1)  # Features
         Y = transformed_df['Order Item Quantity']  # Target variable

         # Align rows if needed
         X = X.loc[Y.index]

         # Split the data into training and testing sets
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

         # Check the shapes
         print(f"X_train shape: {X_train.shape}")
         print(f"Y_train shape: {Y_train.shape}")
         print(f"X_test shape: {X_test.shape}")
         print(f"Y_test shape: {Y_test.shape}")

         # Start the timer
         start_time = time.time()

         # Train Decision Tree Regressor before hyperparameter tuning
         default_model = DecisionTreeRegressor(random_state=42)
         default_model.fit(X_train, Y_train)

         # Predict and evaluate before hyperparameter tuning
         Y_train_pred_default = default_model.predict(X_train)
         Y_test_pred_default = default_model.predict(X_test)

         # Calculate RMSE and R² for training and testing data (default model)
         rmse_train_default = np.sqrt(mean_squared_error(Y_train, Y_train_pred_default))
         r2_train_default = r2_score(Y_train, Y_train_pred_default)
         rmse_test_default = np.sqrt(mean_squared_error(Y_test, Y_test_pred_default))
         r2_test_default = r2_score(Y_test, Y_test_pred_default)

         print("\nBefore Hyperparameter Tuning:")
         print(f"Training RMSE: {rmse_train_default:.4f}, Training R²: {r2_train_default:.4f}")
         print(f"Testing RMSE: {rmse_test_default:.4f}, Testing R²: {r2_test_default:.4f}")
```



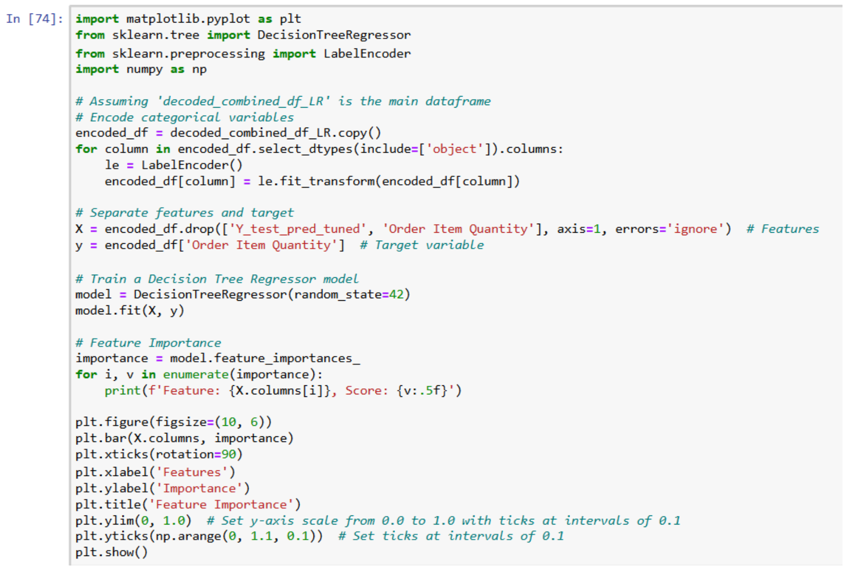
```
# Parameter grid with constraints to reduce overfitting
param_grid = {
    'criterion': ['squared_error'],  # Use squared_error for regression
    'max_depth': [5, 10, 15],        # Limit depth of the tree
    'min_samples_split': [10, 20],   # Increase minimum samples to split
    'min_samples_leaf': [5, 10],     # Increase minimum samples in each leaf
    'max_features': ['sqrt', 'log2'], # Use subset of features for splits
    'splitter': ['best', 'random']   # Try both best and random split strategies
}

# Perform GridSearchCV for hyperparameter tuning
dt_regressor = DecisionTreeRegressor(random_state=42)
grid_search = GridSearchCV(
    estimator=dt_regressor,
    param_grid=param_grid,
    cv=5,                             # Use 5-fold cross-validation
    scoring='neg_mean_squared_error',  # Minimize MSE
    n_jobs=-1                         # Use all cores
)
grid_search.fit(X_train, Y_train)

# Get the best model and its parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

print("\nBest Parameters Found:")
print(best_params)

# Predict and evaluate on the training and testing datasets after hyperparameter tuning
Y_train_pred_tuned = best_model.predict(X_train)
Y_test_pred_tuned = best_model.predict(X_test)

# Calculate RMSE and R² for training and testing data
rmse_train_tuned = np.sqrt(mean_squared_error(Y_train, Y_train_pred_tuned))
r2_train_tuned = r2_score(Y_train, Y_train_pred_tuned)
rmse_test_tuned = np.sqrt(mean_squared_error(Y_test, Y_test_pred_tuned))
r2_test_tuned = r2_score(Y_test, Y_test_pred_tuned)

# Stop the timer
end_time = time.time()
execution_time = end_time - start_time

# Print results after hyperparameter tuning
print("\nAfter Hyperparameter Tuning:")
print(f"Training RMSE: {rmse_train_tuned:.4f}, Training R²: {r2_train_tuned:.4f}")
print(f"Testing RMSE: {rmse_test_tuned:.4f}, Testing R²: {r2_test_tuned:.4f}")
print(f"Execution Time: {execution_time:.2f} seconds")
```

2.2. Code for Line Chart of Predicted vs. Actual Values



```
In [71]: import matplotlib.pyplot as plt
         import numpy as np

         # Select 100 samples or fewer if the dataset is smaller
         num_samples = min(100, len(Y_test))
         actual_values = Y_test[:num_samples]
         predicted_values = Y_test_pred_tuned[:num_samples]

         # Generate indices for the samples
         indices = np.arange(num_samples)

         # Create the plot
         plt.figure(figsize=(12, 6))
         plt.plot(indices, actual_values, label='Actual', marker='o', linestyle='-', markersize=4)
         plt.plot(indices, predicted_values, label='Predicted', marker='x', linestyle='--', markersize=4)

         # Add titles and labels
         plt.title('Comparison of Actual vs Predicted Values')
         plt.xlabel('Sample Index')
         plt.ylabel('Value')
         plt.legend()
         plt.grid(True)

         # Show the plot
         plt.tight_layout()
         plt.show()
```

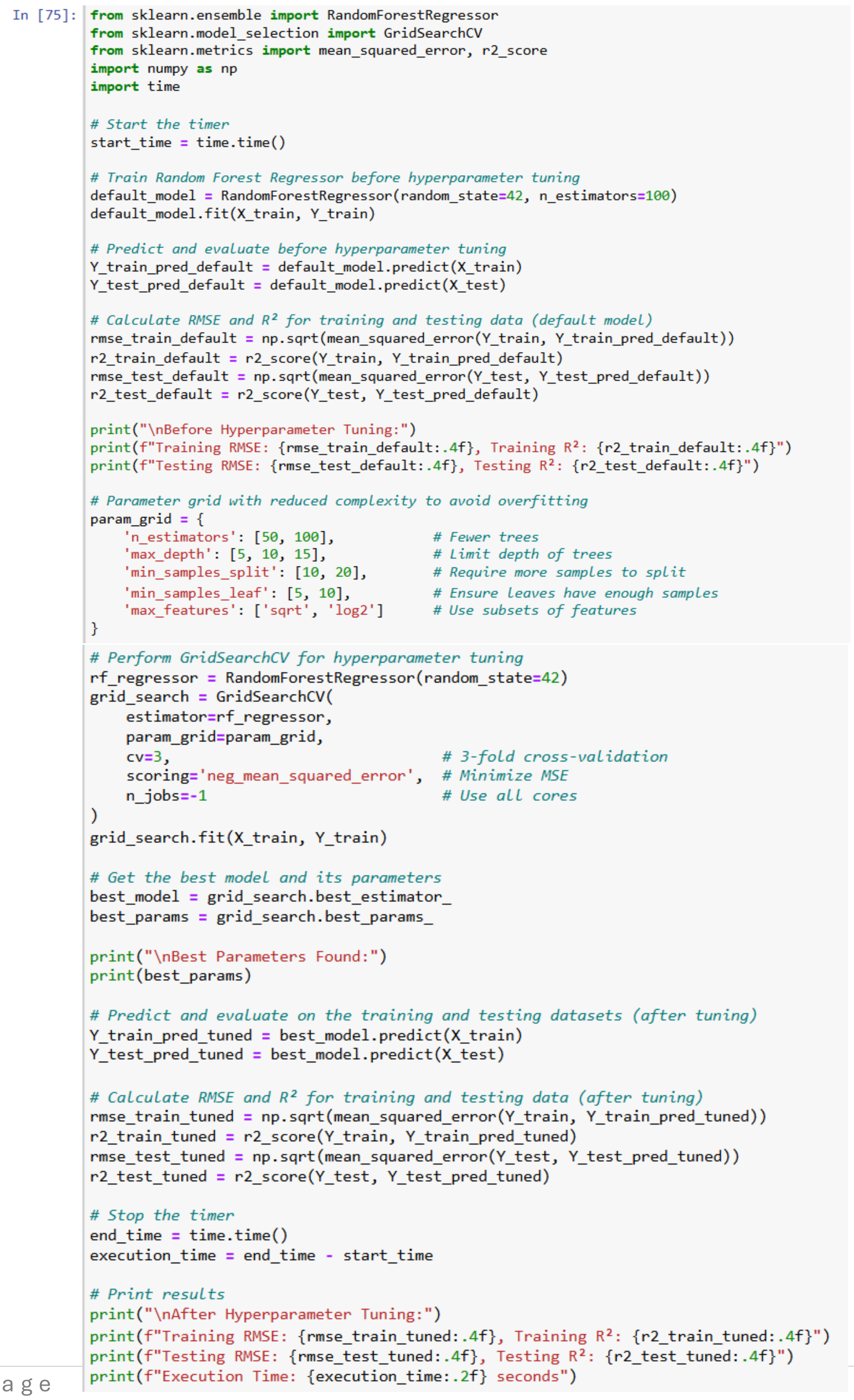
2.3. Code for Feature Importance



```
In [74]: import matplotlib.pyplot as plt
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.preprocessing import LabelEncoder
         import numpy as np

         # Assuming 'decoded_combined_df_LR' is the main dataframe
         # Encode categorical variables
         encoded_df = decoded_combined_df_LR.copy()
         for column in encoded_df.select_dtypes(include=['object']).columns:
             le = LabelEncoder()
             encoded_df[column] = le.fit_transform(encoded_df[column])

         # Separate features and target
         X = encoded_df.drop(['Y_test_pred_tuned', 'Order Item Quantity'], axis=1, errors='ignore')  # Features
         y = encoded_df['Order Item Quantity']  # Target variable

         # Train a Decision Tree Regressor model
         model = DecisionTreeRegressor(random_state=42)
         model.fit(X, y)

         # Feature Importance
         importance = model.feature_importances_
         for i, v in enumerate(importance):
             print(f'Feature: {X.columns[i]}, Score: {v:.5f}')

         plt.figure(figsize=(10, 6))
         plt.bar(X.columns, importance)
         plt.xticks(rotation=90)
         plt.xlabel('Features')
         plt.ylabel('Importance')
         plt.title('Feature Importance')
         plt.ylim(0, 1.0)  # Set y-axis scale from 0.0 to 1.0 with ticks at intervals of 0.1
         plt.yticks(np.arange(0, 1.1, 0.1))  # Set ticks at intervals of 0.1
         plt.show()
```

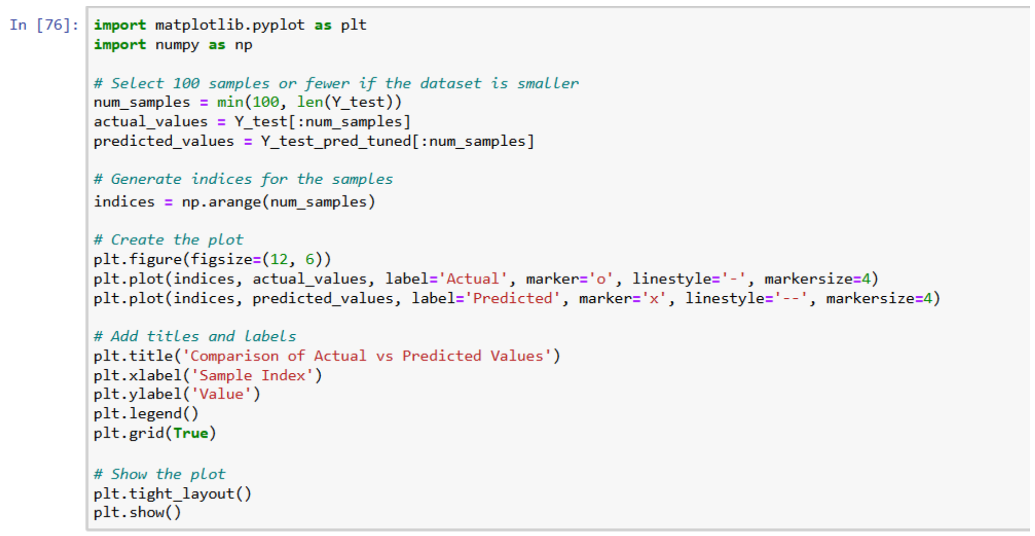
3. Random Forest

3.1. Model Performance Code



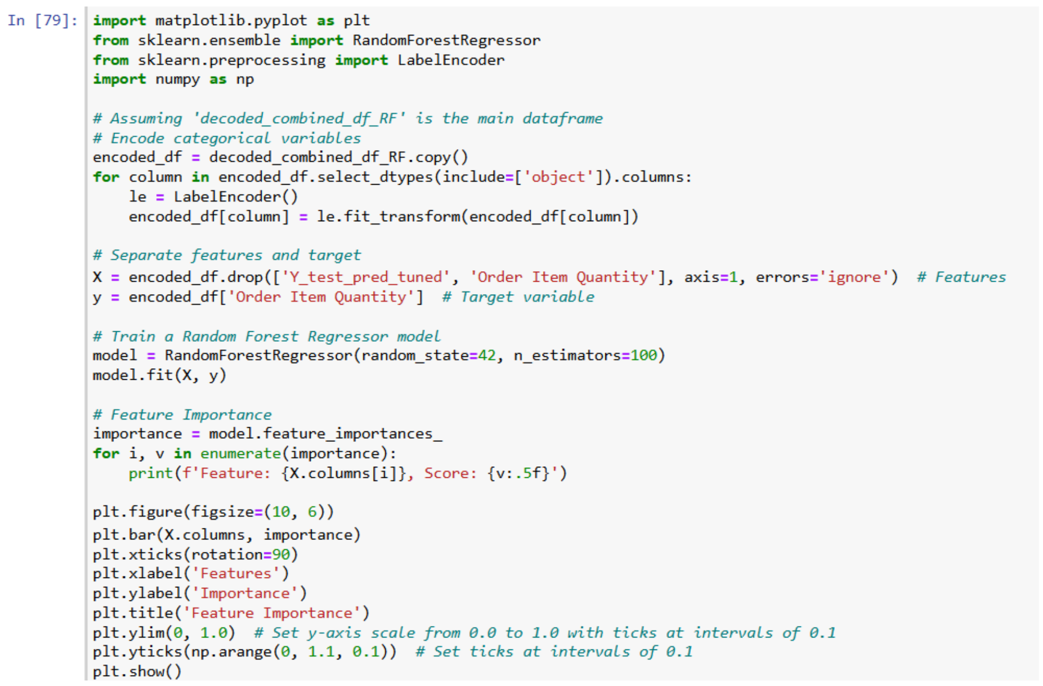
```
In [75]: from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import mean_squared_error, r2_score
         import numpy as np
         import time

         # Start the timer
         start_time = time.time()

         # Train Random Forest Regressor before hyperparameter tuning
         default_model = RandomForestRegressor(random_state=42, n_estimators=100)
         default_model.fit(X_train, Y_train)

         # Predict and evaluate before hyperparameter tuning
         Y_train_pred_default = default_model.predict(X_train)
         Y_test_pred_default = default_model.predict(X_test)

         # Calculate RMSE and R² for training and testing data (default model)
         rmse_train_default = np.sqrt(mean_squared_error(Y_train, Y_train_pred_default))
         r2_train_default = r2_score(Y_train, Y_train_pred_default)
         rmse_test_default = np.sqrt(mean_squared_error(Y_test, Y_test_pred_default))
         r2_test_default = r2_score(Y_test, Y_test_pred_default)

         print("\nBefore Hyperparameter Tuning:")
         print(f"Training RMSE: {rmse_train_default:.4f}, Training R²: {r2_train_default:.4f}")
         print(f"Testing RMSE: {rmse_test_default:.4f}, Testing R²: {r2_test_default:.4f}")

         # Parameter grid with reduced complexity to avoid overfitting
         param_grid = {
             'n_estimators': [50, 100],           # Fewer trees
             'max_depth': [5, 10, 15],            # Limit depth of trees
             'min_samples_split': [10, 20],       # Require more samples to split
             'min_samples_leaf': [5, 10],         # Ensure leaves have enough samples
             'max_features': ['sqrt', 'log2']     # Use subsets of features
         }
         # Perform GridSearchCV for hyperparameter tuning
         rf_regressor = RandomForestRegressor(random_state=42)
         grid_search = GridSearchCV(
             estimator=rf_regressor,
             param_grid=param_grid,
             cv=3,                                # 3-fold cross-validation
             scoring='neg_mean_squared_error',    # Minimize MSE
             n_jobs=-1                            # Use all cores
         )
         grid_search.fit(X_train, Y_train)

         # Get the best model and its parameters
         best_model = grid_search.best_estimator_
         best_params = grid_search.best_params_

         print("\nBest Parameters Found:")
         print(best_params)

         # Predict and evaluate on the training and testing datasets (after tuning)
         Y_train_pred_tuned = best_model.predict(X_train)
         Y_test_pred_tuned = best_model.predict(X_test)

         # Calculate RMSE and R² for training and testing data (after tuning)
         rmse_train_tuned = np.sqrt(mean_squared_error(Y_train, Y_train_pred_tuned))
         r2_train_tuned = r2_score(Y_train, Y_train_pred_tuned)
         rmse_test_tuned = np.sqrt(mean_squared_error(Y_test, Y_test_pred_tuned))
         r2_test_tuned = r2_score(Y_test, Y_test_pred_tuned)

         # Stop the timer
         end_time = time.time()
         execution_time = end_time - start_time

         # Print results
         print("\nAfter Hyperparameter Tuning:")
         print(f"Training RMSE: {rmse_train_tuned:.4f}, Training R²: {r2_train_tuned:.4f}")
         print(f"Testing RMSE: {rmse_test_tuned:.4f}, Testing R²: {r2_test_tuned:.4f}")
         print(f"Execution Time: {execution_time:.2f} seconds")
```

3.2. Code for Line Chart of Predicted vs. Actual Values



```
In [76]: import matplotlib.pyplot as plt
         import numpy as np

         # Select 100 samples or fewer if the dataset is smaller
         num_samples = min(100, len(Y_test))
         actual_values = Y_test[:num_samples]
         predicted_values = Y_test_pred_tuned[:num_samples]

         # Generate indices for the samples
         indices = np.arange(num_samples)

         # Create the plot
         plt.figure(figsize=(12, 6))
         plt.plot(indices, actual_values, label='Actual', marker='o', linestyle='-', markersize=4)
         plt.plot(indices, predicted_values, label='Predicted', marker='x', linestyle='--', markersize=4)

         # Add titles and labels
         plt.title('Comparison of Actual vs Predicted Values')
         plt.xlabel('Sample Index')
         plt.ylabel('Value')
         plt.legend()
         plt.grid(True)

         # Show the plot
         plt.tight_layout()
         plt.show()
```

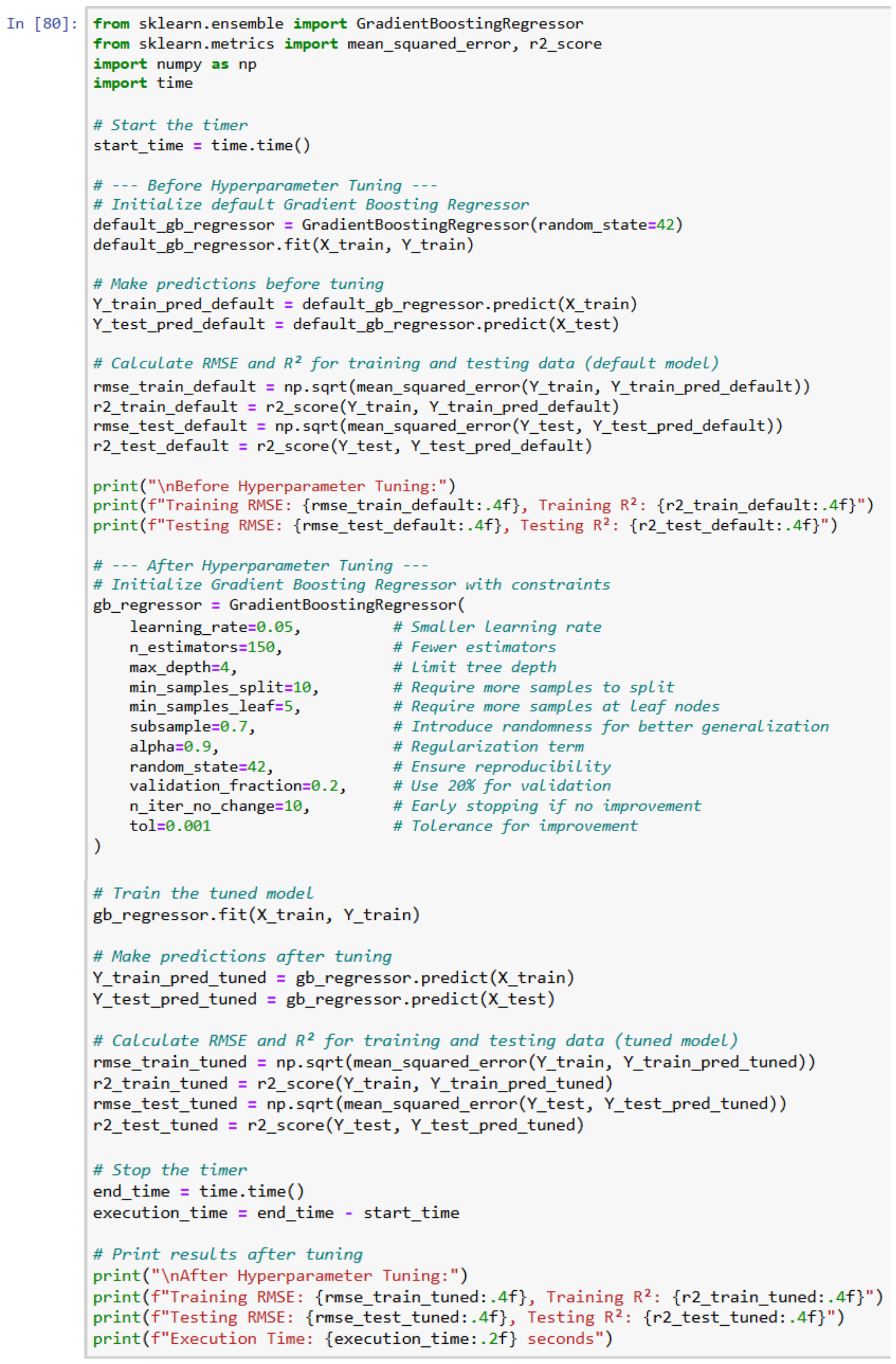
3.3. Code for Feature Importance



```
In [79]: import matplotlib.pyplot as plt
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.preprocessing import LabelEncoder
         import numpy as np

         # Assuming 'decoded_combined_df_RF' is the main dataframe
         # Encode categorical variables
         encoded_df = decoded_combined_df_RF.copy()
         for column in encoded_df.select_dtypes(include=['object']).columns:
             le = LabelEncoder()
             encoded_df[column] = le.fit_transform(encoded_df[column])

         # Separate features and target
         X = encoded_df.drop(['Y_test_pred_tuned', 'Order Item Quantity'], axis=1, errors='ignore')  # Features
         y = encoded_df['Order Item Quantity']  # Target variable

         # Train a Random Forest Regressor model
         model = RandomForestRegressor(random_state=42, n_estimators=100)
         model.fit(X, y)

         # Feature Importance
         importance = model.feature_importances_
         for i, v in enumerate(importance):
             print(f'Feature: {X.columns[i]}, Score: {v:.5f}')

         plt.figure(figsize=(10, 6))
         plt.bar(X.columns, importance)
         plt.xticks(rotation=90)
         plt.xlabel('Features')
         plt.ylabel('Importance')
         plt.title('Feature Importance')
         plt.ylim(0, 1.0)  # Set y-axis scale from 0.0 to 1.0 with ticks at intervals of 0.1
         plt.yticks(np.arange(0, 1.1, 0.1))  # Set ticks at intervals of 0.1
         plt.show()
```

4. Gradient Boosting

4.1. Model Performance Code



```
In [80]: from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.metrics import mean_squared_error, r2_score
         import numpy as np
         import time

         # Start the timer
         start_time = time.time()

         # --- Before Hyperparameter Tuning ---
         # Initialize default Gradient Boosting Regressor
         default_gb_regressor = GradientBoostingRegressor(random_state=42)
         default_gb_regressor.fit(X_train, Y_train)

         # Make predictions before tuning
         Y_train_pred_default = default_gb_regressor.predict(X_train)
         Y_test_pred_default = default_gb_regressor.predict(X_test)

         # Calculate RMSE and R² for training and testing data (default model)
         rmse_train_default = np.sqrt(mean_squared_error(Y_train, Y_train_pred_default))
         r2_train_default = r2_score(Y_train, Y_train_pred_default)
         rmse_test_default = np.sqrt(mean_squared_error(Y_test, Y_test_pred_default))
         r2_test_default = r2_score(Y_test, Y_test_pred_default)

         print("\nBefore Hyperparameter Tuning:")
         print(f"Training RMSE: {rmse_train_default:.4f}, Training R²: {r2_train_default:.4f}")
         print(f"Testing RMSE: {rmse_test_default:.4f}, Testing R²: {r2_test_default:.4f}")

         # --- After Hyperparameter Tuning ---
         # Initialize Gradient Boosting Regressor with constraints
         gb_regressor = GradientBoostingRegressor(
             learning_rate=0.05,           # Smaller learning rate
             n_estimators=150,             # Fewer estimators
             max_depth=4,                  # Limit tree depth
             min_samples_split=10,         # Require more samples to split
             min_samples_leaf=5,           # Require more samples at leaf nodes
             subsample=0.7,                # Introduce randomness for better generalization
             alpha=0.9,                    # Regularization term
             random_state=42,              # Ensure reproducibility
             validation_fraction=0.2,      # Use 20% for validation
             n_iter_no_change=10,          # Early stopping if no improvement
             tol=0.001                     # Tolerance for improvement
         )

         # Train the tuned model
         gb_regressor.fit(X_train, Y_train)

         # Make predictions after tuning
         Y_train_pred_tuned = gb_regressor.predict(X_train)
         Y_test_pred_tuned = gb_regressor.predict(X_test)

         # Calculate RMSE and R² for training and testing data (tuned model)
         rmse_train_tuned = np.sqrt(mean_squared_error(Y_train, Y_train_pred_tuned))
         r2_train_tuned = r2_score(Y_train, Y_train_pred_tuned)
         rmse_test_tuned = np.sqrt(mean_squared_error(Y_test, Y_test_pred_tuned))
         r2_test_tuned = r2_score(Y_test, Y_test_pred_tuned)

         # Stop the timer
         end_time = time.time()
         execution_time = end_time - start_time

         # Print results after tuning
         print("\nAfter Hyperparameter Tuning:")
         print(f"Training RMSE: {rmse_train_tuned:.4f}, Training R²: {r2_train_tuned:.4f}")
         print(f"Testing RMSE: {rmse_test_tuned:.4f}, Testing R²: {r2_test_tuned:.4f}")
         print(f"Execution Time: {execution_time:.2f} seconds")
```

4.2. Code for Line Chart of Predicted vs. Actual Values

```
In [81]: import matplotlib.pyplot as plt
         import numpy as np

         # Select 100 samples or fewer if the dataset is smaller
         num_samples = min(100, len(Y_test))
         actual_values = Y_test[:num_samples]
         predicted_values = Y_test_pred_tuned[:num_samples]

         # Generate indices for the samples
         indices = np.arange(num_samples)

         # Create the plot
         plt.figure(figsize=(12, 6))
         plt.plot(indices, actual_values, label='Actual', marker='o', linestyle='-', markersize=4)
         plt.plot(indices, predicted_values, label='Predicted', marker='x', linestyle='--', markersize=4)

         # Add titles and labels
         plt.title('Comparison of Actual vs Predicted Values')
         plt.xlabel('Sample Index')
         plt.ylabel('Value')
         plt.legend()
         plt.grid(True)

         # Show the plot
         plt.tight_layout()
         plt.show()
```

4.3. Code for Feature Importance

```
In [84]: import matplotlib.pyplot as plt
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.preprocessing import LabelEncoder
         import numpy as np

         # Assuming 'decoded_combined_df_GBR' is the main dataframe
         # Encode categorical variables
         encoded_df = decoded_combined_df_GBR.copy()
         for column in encoded_df.select_dtypes(include=['object']).columns:
             le = LabelEncoder()
             encoded_df[column] = le.fit_transform(encoded_df[column])

         # Separate features and target
         X = encoded_df.drop(['Y_test_pred_tuned', 'Order Item Quantity'], axis=1, errors='ignore')  # Features
         y = encoded_df['Order Item Quantity']  # Target variable

         # Train a Gradient Boosting Regressor model
         model = GradientBoostingRegressor(random_state=42, n_estimators=100, learning_rate=0.1)
         model.fit(X, y)

         # Feature Importance
         importance = model.feature_importances_
         for i, v in enumerate(importance):
             print(f'Feature: {X.columns[i]}, Score: {v:.5f}')

         plt.figure(figsize=(10, 6))
         plt.bar(X.columns, importance)
         plt.xticks(rotation=90)
         plt.xlabel('Features')
         plt.ylabel('Importance')
         plt.title('Feature Importance')
         plt.ylim(0, 1.0)  # Set y-axis scale from 0.0 to 1.0 with ticks at intervals of 0.1
         plt.yticks(np.arange(0, 1.1, 0.1))  # Set ticks at intervals of 0.1
         plt.show()
```