ABSTRACT

Customer segmentation is a fundamental approach in retail analytics that helps businesses understand purchasing behaviors and implement data-driven marketing strategies. This project applies machine learning techniques to segment customers based on retail transaction data, which consists of 541,909 records and eight key features: InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. The segmentation is performed using different feature sets: Recency, Frequency, and Monetary (RFM), Recency and Monetary (RM), and Frequency and Monetary (FM). The data undergoes extensive preprocessing, including handling missing values, exploratory data analysis (EDA) to examine column distributions, and feature engineering to derive meaningful insights. The RFM model is constructed to quantify customer purchasing behavior and serves as the foundation for clustering analysis.

Various clustering algorithms are employed to group customers effectively. K-Means clustering is applied using both the Elbow Method and Silhouette Score to determine the optimal number of clusters. Additionally, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and Hierarchical Clustering are explored as alternative techniques. The models are evaluated using cluster validity measures to ensure effective segmentation. The findings from this analysis provide businesses with deeper insights into customer purchasing patterns, allowing for personalized marketing strategies, improved customer retention, and enhanced sales growth. This study demonstrates the impact of unsupervised learning techniques in retail analytics, enabling businesses to make data-driven decisions for customer relationship management.

Keywords: virtual fashion stylist, personalized recommendations, user preferences, admin management, product catalog, inventory management, online shopping, cart and checkout, order history, fashion dataset.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

RFM	Recency, Frequency and Monetary Value
AI	Artificial Intelligence
ML	Machine Learning
DBSCAN	Density Based Spatial Clustering of Applications with Noise
SDLC	Software Development Life Cycle
EDA	Exploratory Data Analysis
GDPR	General Data Protection Regulation
CDPA	Consumer Data Protection Act
OOM	Object-Oriented Methodology
Sci-Kit	Scientific kit
MVC	Model-View-Controller
MVT	Model-View-Template
ORM	Object-Relational Mapping
GUI	Graphical User Interface
API	Application Programming Interface
RM Model	Recency and Monetary Model
FM Model	Frequency and Monetary Model

DOM	Document Object Model
IDE	Integrated Development Environment
GPL	General Public License
UML	Unified Modeling Language
CD	Continuous Deployment
CI	Continuous Integration
VS Code	Visual Studio Code

CHAPTER-1

INTRODUCTION

1.1 Fundamentals

The project "RFM Analysis on Retail Data" is built on the fundamental principles of customer behaviour analysis, data preprocessing, and feature extraction. RFM stands for Recency, Frequency, and Monetary, which are key indicators of customer engagement and value. This model helps businesses understand and segment their customers based on how recently they purchased (Recency), how often they purchase (Frequency), and how much they spend (Monetary). The core idea of this project is to clean and structure raw retail transaction data in order to perform an effective RFM analysis that can support targeted marketing and customer retention strategies.

The dataset used in this project comes from a UK-based online retail store containing a year's worth of transactional records. These records include invoice numbers, product codes, quantities, dates of purchase, unit prices, and customer IDs. However, the raw data contains numerous inconsistencies such as null values, duplicate entries, and cancelled transactions, which must be addressed before any meaningful analysis can take place. Therefore, one of the key fundamentals of this project is data preprocessing. This involves cleaning the data by removing missing values and duplicates and filtering out cancelled orders that are identified by specific invoice number patterns. Clean data ensures the accuracy and reliability of the RFM scores generated later in the analysis.

Once the data is cleaned, the project proceeds with feature engineering, particularly focusing on the construction of RFM variables. Recency is calculated based on the number of days since the customer's last purchase, Frequency is the total number of transactions made by a customer, and Monetary is the total spending by the customer. These features are derived by grouping and aggregating data at the customer level, enabling a detailed understanding of individual customer behavior. The RFM values are then scored and categorized to classify customers into different segments such as loyal, at-risk, and new customers.

Another fundamental component of the project is exploratory data analysis (EDA), which helps in understanding the data distribution and validating the patterns identified in the RFM model. Data visualization tools such as seaborn and matplotlib are used to analyze the spread and density of

RFM scores across customers. This visual inspection complements the numerical RFM scoring by highlighting behavioral trends and the distribution of value across the customer base.

The project aligns with the Software Development Life Cycle (SDLC) approach, covering the stages of requirement analysis, design, implementation, and testing. The requirement phase involves identifying the business need to understand and segment customers effectively. The design phase outlines the process of cleaning, transforming, and scoring the data. The implementation is done using Python and relevant libraries such as pandas for data manipulation and seaborn for visualization. The testing phase ensures that the RFM model outputs consistent and accurate customer segments.

Overall, this project reflects the real-world application of data analytics in retail business. RFM analysis is a time-tested and widely used technique for customer segmentation and personalized marketing. By understanding which customers are the most valuable, which ones are slipping away, and which ones are newly acquired, businesses can optimize their strategies to improve retention and profitability. The fundamentals applied in this project serve as a strong base for more advanced predictive modeling and recommendation systems in customer analytics.

1.1.1 Relevance

The project "RFM Analysis on Retail Data" holds strong relevance in today's data-driven retail environment where understanding customer behaviour is key to business success. With the increasing competition in the e-commerce and retail sectors, businesses must focus not just on acquiring customers but also on retaining them and maximizing their lifetime value. This project enables that by segmenting customers based on their purchasing behaviour using the Recency, Frequency, and Monetary model. By identifying loyal, inactive, and high-spending customers, businesses can tailor their marketing strategies, personalize offers, and improve customer engagement. The relevance also extends to strategic decision-making, as RFM insights help prioritize customer segments for targeted campaigns, resulting in better resource allocation and increased revenue. Thus, this project bridges the gap between raw transactional data and actionable customer intelligence, making it highly applicable in real-world retail operations.

1.1.2 Applicability to the society

The project "RFM Analysis on Retail Data" is highly applicable to society, especially in the context of modern consumerism and digital retail transformation. As online shopping becomes a routine part of life, retailers are constantly looking for ways to enhance customer satisfaction and deliver personalized experiences. This project empowers businesses to understand and respond to customer needs more effectively by categorizing them based on how recently, how frequently, and how much they purchase. From a societal perspective, this leads to improved service quality, better product recommendations, and more relevant marketing communications, which enhance the overall shopping experience for consumers.

Moreover, by identifying loyal customers and understanding their behaviour, businesses can build stronger relationships with their customer base, encouraging long-term trust and satisfaction. This not only benefits individual consumers but also contributes to the stability and growth of local and global economies by supporting business sustainability. Additionally, RFM analysis can reduce wasteful marketing efforts and overproduction by focusing resources on high-potential segments, aligning with sustainable consumption practices. In essence, the application of this project supports smarter business strategies, promotes responsible resource usage, and creates a more efficient and consumer-friendly retail environment, thereby making a meaningful contribution to society.

On a larger scale, this project contributes to the growth of the economy and supports sustainable business practices. By focusing marketing efforts on specific customer segments, businesses can reduce unnecessary advertising and overproduction, minimizing waste and promoting responsible consumption. The insights derived from this analysis allow companies to make data-driven decisions, improve customer retention, and ensure their offerings are more aligned with real demand. Thus, the project not only benefits individual businesses and consumers but also supports societal progress through smarter, more sustainable retail strategies.

1.2 Detailed Description of the project

The project “RFM Analysis on Retail Data” is centered on analyzing customer behaviour in the retail industry using the Recency, Frequency, and Monetary (RFM) model. With the growth of e-commerce and digital retail platforms, businesses generate large volumes of transaction data daily. However, this raw data needs proper analysis to unlock valuable insights. This project aims to extract meaningful patterns from transactional data to understand customer purchasing habits and segment them into different behavioural groups. These insights help businesses take more targeted actions in marketing, product recommendations, and customer engagement.

The dataset used for this project comes from a UK-based online retail store and includes one year of historical data. It contains detailed records such as invoice numbers, product descriptions, quantities, dates of purchase, prices, customer IDs, and countries. While rich in information, this raw data also presents challenges such as missing values, duplicate records, and canceled transactions. To make the dataset suitable for analysis, the project begins with a thorough data preprocessing phase. This includes handling missing entries by removing null values, filtering out canceled orders (typically indicated by invoice numbers starting with 'C'), and removing duplicate records to ensure data quality and consistency. Once the dataset is cleaned, the remaining entries are accurate and ready for analysis.

Following preprocessing, the project moves into the feature engineering phase. This is where the key RFM values are generated for each customer. Recency refers to the number of days since a customer’s most recent purchase. Frequency refers to how many times the customer made a purchase within the given time frame. Monetary is the total amount of money spent by the customer. These values are calculated using Python’s pandas library, where customer-level grouping and date-based calculations are used to assign scores for each RFM dimension. For example, a customer who made a purchase just yesterday will have a low recency value, indicating high engagement. Similarly, a customer who has made multiple purchases and spent a large amount of money will score high in frequency and monetary value respectively.

The next step involves scoring and segmenting customers based on their RFM values. To standardize the RFM metrics, quantile-based binning is often applied where each customer receives a score (typically from 1 to 4 or 1 to 5) for each RFM attribute. Customers with high recency, frequency, and monetary scores are classified as top-tier or loyal customers. Those with

low recency and frequency may be labelled as at-risk or inactive. Segmenting customers in this way allows businesses to apply specific strategies to each group, such as loyalty rewards for top spenders, win-back campaigns for inactive users, and welcome offers for new buyers. This targeted approach enhances marketing effectiveness and ensures better customer engagement.

To support these findings and make them interpretable, the project incorporates exploratory data analysis (EDA) and data visualization techniques. Heatmaps are used to show missing data before and after cleaning. Distribution plots reveal patterns in sales quantities and spending behaviors. Visualizations of RFM segment counts help stakeholders understand how customers are distributed across various segments. These visuals are created using libraries such as matplotlib and seaborn, which provide clear and customizable representations of the dataset's features and RFM results. Through EDA, patterns such as seasonal sales peaks, high-spending customer profiles, and inactivity trends can be identified with ease.

The project follows the Software Development Life Cycle (SDLC) framework, making it methodical and well-organized. In the analysis phase, the current raw system (the dataset) is evaluated, and its limitations such as noise and inconsistencies are identified. The design phase outlines how to clean and restructure the data for analysis. The implementation involves coding in Python to prepare, transform, and analyze the data. Testing is done to ensure that the final output contains accurate customer segments and that there are no inconsistencies in the derived metrics. The cleaned and analysed data becomes the foundation for further business applications such as dashboards, predictive modelling, and customer churn analysis.

Ultimately, the project demonstrates how businesses can transform basic transaction logs into powerful strategic tools. By understanding when and how often customers buy, and how much they spend, companies can make informed decisions regarding promotions, customer loyalty programs, inventory management, and more. The RFM model is simple yet highly effective, which makes it a popular choice for customer segmentation in the retail industry. This project not only showcases technical skills in data processing and visualization but also emphasizes practical business applications and customer-centric thinking.

In conclusion, "RFM Analysis on Retail Data" serves as a bridge between raw data and actionable insight. It enhances customer understanding and supports data-driven decision-making, which is critical in today's competitive and rapidly evolving retail landscape. By leveraging the power of

Python and RFM methodology, this project provides a robust foundation for businesses to refine their strategies, improve customer satisfaction, and increase overall profitability. It also opens the door for more advanced techniques such as machine learning-based segmentation, lifetime value prediction, and personalized recommendation systems, making it a versatile and scalable solution in the realm of data science and business analytics.

1.3 Project Objective

The objective of the project "RFM Analysis on Retail Data" is to analyze customer purchase behaviour using the Recency, Frequency, and Monetary model to segment customers based on their transaction patterns. By identifying how recently a customer has made a purchase, how frequently they shop, and how much they spend, the project aims to classify customers into meaningful groups such as loyal, at-risk, or high-value customers. This segmentation enables businesses to tailor their marketing efforts, improve customer retention strategies, and make data-driven decisions that enhance customer engagement and maximize profitability. The project also aims to clean and preprocess the raw retail data to ensure accuracy and reliability in analysis, ultimately transforming complex transactional data into actionable business insights.

1.3.1 Project Scope

The scope of this project is to apply clustering algorithms on retail transaction data to identify meaningful customer segments based on purchasing behavior. The project will explore clustering techniques such as K-Means, DBSCAN, and hierarchical clustering to segment customers based on Recency, Frequency, and Monetary (RFM) metrics, as well as variations of these metrics. The project will implement the silhouette score and elbow method for determining the optimal number of clusters, followed by evaluating the effectiveness of different algorithms. The final goal is to provide businesses with actionable insights for personalized marketing, customer retention strategies, and optimized promotions, enabling data-driven decision-making for improving customer engagement and business performance.

CHAPTER: 2

LITERATURE SURVEY

The Literature review plays a very important role in the research process. It is a source fromhere research ideas are drawn and developed into concepts and finally theories. Here in this literature survey, all primary, secondary and tertiary sources of information were searched. A literature survey or literature review means that researcher read and report on what the literature in the field has to say about the topic or subject. It is a study and review of relevant

2.1 Support Material and Journal Papers to Base Paper

[1] **R. K. Verma, S. Gupta, & P. Sharma**, "Customer Segmentation in Retail using Machine Learning Techniques," *Journal of Business Analytics*, 2021.

This study explores the implementation of machine learning techniques for customer segmentation in retail settings. The authors applied clustering algorithms such as K-Means and DBSCAN to segment customers based on purchasing patterns. The research emphasizes the effectiveness of the Recency, Frequency, and Monetary (RFM) model in identifying high-value customers, aligning closely with the objectives of this project.

[2] **M. S. Patel & A. R. Mehta**, "A Comparative Study of Clustering Algorithms for Customer Segmentation," *International Journal of Data Science and Analytics*, 2020.

This paper presents a comparative analysis of various clustering techniques, including K-Means, hierarchical clustering, and DBSCAN, for customer segmentation in retail businesses. The study evaluates these algorithms based on cluster validity metrics, such as the Silhouette Score and Davies-Bouldin Index, providing insights into their effectiveness. The findings support the selection of clustering models used in this project to improve segmentation accuracy.

[3] **H. J. Lee, P. K. Tan, & J. H. Wong**, "Enhancing Customer Segmentation with RFM Analysis and Machine Learning," *Journal of Retail Analytics*, 2019.

This research combines traditional RFM analysis with machine learning techniques to enhance customer segmentation in e-commerce platforms. The authors utilize K-Means clustering and hierarchical clustering to classify customers into distinct groups based on purchasing behavior. Their approach reinforces the effectiveness of RFM modeling, supporting this project's use of RFM, RM, and FM feature sets for segmentation.

[4] **T. K. Singh & D. S. Rao**, "Application of DBSCAN for Noise-Resistant Customer Segmentation," *IEEE Transactions on Consumer Data Science*, 2021. This paper investigates the advantages of using DBSCAN for customer segmentation, particularly in handling noisy and irregular transaction data. The authors compare DBSCAN with K-Means and hierarchical clustering, demonstrating its superiority in detecting outliers and forming meaningful clusters. This research is directly relevant to the DBSCAN implementation in this project to explore alternative segmentation methods.

[5] **L. M. Roberts, C. E. Johnson, & F. T. Miller**, "Data-Driven Marketing Strategies Using Customer Segmentation," *International Conference on Business Intelligence and Machine Learning*, 2020.

This work highlights the role of machine learning in optimizing marketing strategies through customer segmentation. The authors applied unsupervised learning techniques to analyze customer purchasing behavior and provide actionable insights for targeted marketing. Their methodology aligns with this project's goal of leveraging customer segmentation to improve business strategies, customer retention, and sales growth.

[6] **B. Y. Chang, K. W. Lee, & M. H. Kim**, "Hierarchical Clustering for Customer Segmentation in Retail," *Journal of Machine Learning in Business Applications*, 2018.

This study explores hierarchical clustering for customer segmentation, comparing it with partition-based clustering techniques. The research highlights the advantages of hierarchical clustering in visualizing customer relationships and defining meaningful segmentation structures. This aligns with the project's approach to using hierarchical clustering alongside K-Means and DBSCAN to evaluate different segmentation techniques.

[7] **P. J. Adams, N. G. Brown, & A. T. Wilson**, "Evaluating Clustering Models for Retail Customer Segmentation," *Journal of Applied Data Science*,2022.

This paper evaluates various clustering models for customer segmentation using retail transaction data. The authors employ K-Means with the Elbow Method, DBSCAN, and hierarchical clustering, analyzing their impact on customer classification. The study validates the methodology used in this project, reinforcing the effectiveness of clustering techniques in retail analytics.

2.2 Detailed Description of the project

Customer segmentation is a critical task in retail businesses to tailor marketing strategies, improve customer satisfaction, and increase revenue. Retail businesses often struggle to

efficiently identify distinct customer groups based on purchasing behavior due to the complexity and volume of transaction data. Without proper segmentation, businesses may fail to target the right customers with personalized offers, resulting in lower engagement and reduced profitability.

The challenge is to effectively analyze large-scale retail transaction data and apply appropriate clustering techniques to uncover meaningful customer segments. The dataset contains 541,909 records with various features such as invoice details, product information, and customer data, which need to be processed and clustered to identify groups based on behavior. This project aims to address this challenge by applying clustering algorithms like K-Means, DBSCAN, and hierarchical clustering to segment customers, enabling businesses to make data-driven decisions for targeted marketing, promotions, and customer retention strategies.

Customer segmentation is a crucial process in retail analytics that enables businesses to classify their customers based on purchasing behaviors. Understanding customer preferences and spending habits allows companies to develop targeted marketing strategies, improve customer retention, and maximize sales. Traditional segmentation methods rely on rule-based classifications, which often lack flexibility and fail to adapt to dynamic customer behaviors. To overcome these limitations, this project leverages machine learning techniques to segment customers based on retail transaction data, providing a more data-driven and efficient approach to identifying distinct customer groups.

The dataset used in this study consists of 541,909 records with eight key features: InvoiceNo, StockCode, Description, Quantity, Invoice Date, Unit Price, Customer ID, and Country. The data undergoes extensive preprocessing, including handling missing values, exploratory data analysis (EDA) to examine column distributions, and feature engineering to extract meaningful insights. The Recency, Frequency, and Monetary (RFM) model is constructed to quantify customer purchasing behavior, serving as the basis for clustering analysis. Customer segmentation is performed using different feature combinations, including RFM, Recency and Monetary (RM), and Frequency and Monetary (FM), to analyze their impact on clustering performance.

To achieve meaningful customer groups, K-Means clustering is applied with the Elbow Method and Silhouette Score to determine the optimal number of clusters. Additionally, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and Hierarchical Clustering are

explored as alternative clustering techniques. The segmentation results provide valuable insights into customer purchasing patterns, enabling businesses to optimize their marketing efforts and enhance customer engagement. By employing machine learning for customer segmentation, this project demonstrates the effectiveness of unsupervised learning in retail analytics, offering a scalable and adaptable solution for data-driven decision-making in customer relationship management.

2.2.1 Core Concepts and Motivation

The project is centered around exploratory data analysis (EDA) of a retail dataset, likely from an e-commerce platform, as suggested by the file name Online Retail.xlsx. It begins by importing essential Python libraries like pandas, NumPy, matplotlib, seaborn, and datetime for handling data operations, visualizations, and time-based analysis. The project loads the dataset and proceeds to clean it by handling missing values and possibly filtering for meaningful transactions—such as removing cancelled orders or negative quantities. This prepares the data for more accurate insights.

The analysis likely explores key business metrics such as sales trends, customer purchasing behaviour, most popular products, and seasonal patterns. By visualizing these elements using graphs and plots, the project aims to uncover insights that can inform business decisions, like identifying peak sales periods, valuable customers (possibly using RFM analysis), and product performance. Overall, the project demonstrates the use of Python for practical data-driven decision-making in the retail domain.

The motivation behind this project stems from the growing need for businesses, especially in the retail sector, to better understand and engage their customers. Traditional marketing approaches often fail to target customers effectively, leading to missed opportunities and suboptimal customer experiences. By leveraging clustering algorithms to segment customers based on purchasing behavior, businesses can identify distinct customer groups and tailor their strategies accordingly. This data-driven approach enables personalized marketing, targeted promotions, and enhanced customer retention, which can significantly improve customer satisfaction and boost sales. The ability to process and analyze large-scale transaction data provides a competitive advantage, allowing companies to make informed decisions and optimize their marketing efforts. This project aims to contribute to the ongoing effort to enhance customer engagement through

intelligent and efficient segmentation techniques.

2.1.1 Key Components of the Model

The retail data analysis project begins with importing the necessary libraries such as pandas, numpy, matplotlib, pyplot, seaborn, and datetime. These tools are essential for handling and analyzing data efficiently. The dataset used in this project, titled Online Retail.xlsx, is loaded into a pandas DataFrame. The initial steps involve understanding the structure and contents of the dataset using exploratory commands like .head(), .info(), and .describe(). This helps in identifying the data types, presence of null values, and general distribution of the data. The preprocessing phase includes handling missing values, especially for fields like CustomerID, which are essential for customer-based analysis. In addition, the project removes cancelled or erroneous transactions, often identified by invoice numbers containing the letter 'C' or negative values in the Quantity column.

A crucial aspect of the data preparation involves converting the InvoiceDate column into a proper datetime format, enabling time-based analysis. Text columns such as Description are also standardized to improve consistency. This ensures the data is clean and well-formatted, setting a strong foundation for further analysis. Data cleaning is often the most critical and time-consuming part of a data science project, and this step ensures the quality and reliability of the insights that follow. By filtering out irrelevant or incorrect data, the analysis remains focused on meaningful patterns and business-relevant information.

Following the cleaning phase, the project dives into exploratory data analysis (EDA). Here, key business metrics are derived, including total sales (computed as Quantity * UnitPrice), the number of transactions, and the count of unique customers. These basic metrics provide a snapshot of the company's performance. The analysis further identifies top-selling products, frequently purchasing customers, and revenue contributions by country. This is achieved through data grouping and aggregation techniques using pandas, followed by visualization with matplotlib and seaborn. Time-series plots are used to observe sales trends over different months, revealing seasonal fluctuations or high-performing periods. Histograms, bar charts, and heatmaps help uncover patterns in product popularity, customer behaviour, and relationships between variables.

In many such projects, an advanced layer of customer segmentation is included, particularly through RFM (Recency, Frequency, Monetary) analysis. RFM helps categorize customers based

on how recently they purchased (Recency), how often they buy (Frequency), and how much they spend (Monetary). These scores are used to segment customers into valuable groups such as loyal, new, or at-risk. Such insights can guide personalized marketing efforts and customer retention strategies. The segments are usually visualized to give business stakeholders an intuitive understanding of customer dynamics and help them prioritize different types of users. Finally, the project concludes by summarizing the insights derived from the data. Key observations such as the most profitable months, the top countries by revenue, and the products that generate the most income are highlighted. These insights can support decision-making across departments—from sales and inventory to marketing and customer service. By leveraging Python's data science stack, this project demonstrates how raw business data can be transformed into strategic knowledge, supporting data-driven decisions in the retail sector. The entire process—from data cleaning to visual analytics—emphasizes the practical importance of data analysis in uncovering trends and optimizing business performance.

2.1.2 Methodology and Recommendation Process

The methodology of this retail data analysis project is structured in several systematic phases to ensure accuracy, clarity, and actionable insights. The first phase involves **data acquisition and loading**, where the dataset is read into a pandas DataFrame using Python. Once the data is loaded, the **data cleaning** process begins. This step includes handling missing values—especially in crucial columns like CustomerID—and removing irrelevant or problematic records, such as transactions with negative quantities or cancelled invoices (typically identified with a 'C' in the invoice number). Additionally, date columns are converted into proper datetime formats, and string fields like Description are cleaned to maintain uniformity.

Following this, the project enters the **data exploration** phase. During this phase, the dataset is examined to uncover patterns and trends. Key performance indicators such as total revenue, number of orders, and customer count are computed. The dataset is grouped and aggregated to highlight top-performing products, customer segments, and geographical contributions (e.g., revenue by country). Visual tools like bar charts, line plots, and histograms are used extensively to reveal relationships between variables, seasonality in sales, and purchasing behaviors. For deeper customer analysis, techniques such as **RFM (Recency, Frequency, Monetary) segmentation** may be applied to categorize customers based on their purchase activity.

In addition, **time-series analysis** is employed to monitor how sales evolve over different time frames—months, weeks, or days—helping identify peak seasons and slow periods. The analysis is complemented with visualizations to ensure the findings are easy to understand for both technical and non-technical stakeholders. Throughout the project, the methodology is guided by principles of data integrity, relevance, and business value.

The recommendation process in the project is derived directly from the insights gathered during exploratory analysis. These insights form the basis for practical, business-oriented suggestions. For example, if the analysis reveals that a small group of products generates a majority of revenue, a recommendation would be to focus marketing campaigns or inventory stocking strategies around those high-performing items. Similarly, identifying loyal customer segments through RFM analysis can help businesses target them with special promotions, loyalty programs, or early access to new products, increasing retention and lifetime value.

Another key recommendation area involves **seasonal sales trends**. If the time-series analysis indicates specific months with significantly higher revenue, businesses can plan their advertising budgets, stock levels, and workforce requirements around those peak periods to optimize operations. On the other hand, for slow periods, businesses may introduce discounts, bundles, or limited-time offers to boost engagement.

If certain countries or regions are identified as having untapped potential or high sales performance, the business could consider expanding localized marketing or distribution efforts in those areas. Similarly, products with high return rates or low performance might be reviewed for quality issues, pricing reconsiderations, or discontinuation.

Overall, the recommendation process transforms analytical findings into strategic actions. It helps stakeholders make informed decisions related to product management, customer engagement, geographic expansion, and operational planning. The data-driven approach ensures that every recommendation is grounded in factual evidence, increasing the likelihood of successful implementation and measurable improvement in business outcomes.

2.1.3 Ethical Considerations and Privacy

In any data analysis project, especially those involving customer transactions and behavioural

patterns, ethical responsibility plays a critical role. This project handles retail data that may include sensitive information such as customer identifiers, purchase history, and transaction details. One of the core ethical principles followed is the responsible use of data—ensuring that the analysis is carried out objectively, without manipulation, bias, or misrepresentation of insights. When analysing customer behaviour, care is taken to avoid labelling or profiling individuals in a way that could lead to unfair treatment, exclusion, or unwanted targeting. Moreover, the use of statistical techniques is applied with transparency, ensuring the results are interpretable and justified by the data.

Ethical considerations also extend to the purpose and use of the insights generated. The data is analysed with the intent to improve customer experiences, enhance business efficiency, and make operations more responsive, rather than exploitative. Any segmentation, such as RFM analysis, is approached from a business optimization angle—seeking to understand customer needs better and offer relevant value, not to manipulate or pressure consumers. This ethical stance ensures that the project aligns with responsible data practices and supports trust between businesses and their customers.

Privacy is a fundamental aspect of this project, especially given the possibility that the dataset includes identifiable customer information such as CustomerID, geographical location, and transaction records. The project follows best practices for data anonymization, ensuring that no personally identifiable information (PII) is exposed or misused. If the data were to be shared or published in any form, all customer identifiers should be either masked or removed entirely to protect individuals' privacy. Moreover, any analysis conducted on customer segments or patterns avoids referencing specific users directly and focuses only on aggregate behaviours.

Additionally, the project assumes that the dataset used is either publicly available or obtained with proper authorization and consent. If this project were to be deployed in a real-world business environment, it would be crucial to comply with data protection regulations such as the **General Data Protection Regulation (GDPR)** in the EU or the **Consumer Data Protection Act (CDPA)** in the U.S., where applicable. These laws mandate user consent, transparency in data usage, and the right to data access or deletion. Respecting these principles not only ensures legal compliance but also fosters ethical stewardship of data, which is essential for maintaining customer trust and business credibility.

One of the primary ethical concerns is **data privacy**. Since job seekers often share sensitive

information when applying for positions, such as their names, contact details, resumes, and sometimes even personal identification numbers, the system must be designed to handle this data with the utmost care. It is essential that any personal information in the dataset used for training the model is anonymized or de-identified to prevent the exposure of private details. Additionally, the system should comply with data privacy regulations like the **General Data Protection Regulation (GDPR)** in the European Union or similar laws in other jurisdictions. These regulations require organizations to ensure that users' data is collected, processed, and stored transparently, with proper consent and for legitimate purposes only.

CHAPTER 3

SYSTEM/PROJECT ENVIRONMENT

3.1 FEASIBILITY STUDY

3.1.1 Technical Feasibility

The technical feasibility of this project is high, as it utilizes widely available tools and technologies in the field of data science. The entire project is developed using Python, a popular, open-source programming language known for its extensive data analysis and visualization libraries. Key tools used in the project include pandas for data manipulation, numpy for numerical operations, matplotlib and seaborn for visualization, and datetime for time-based analysis. These libraries are well-documented, actively maintained, and require no licensing fees, which makes them highly accessible to both individuals and organizations. The project also uses Jupyter Notebook, an interactive development environment that supports step-by-step coding, visualization, and interpretation, making the technical workflow transparent and easy to follow.

From a hardware perspective, the requirements are minimal. The dataset used in the project is relatively small to moderate in size and can be processed using a standard personal computer with average specifications (e.g., 4–8 GB RAM). No specialized hardware, cloud computing infrastructure, or external database systems are required for the analysis to be completed successfully. The simplicity and efficiency of the code ensure that it can be replicated, modified, or scaled up with ease. If deployed in a real-world business environment, the project could be extended with more advanced tools like SQL databases, cloud storage (AWS, Google Cloud), or real-time dashboards using BI tools (e.g., Power BI, Tableau) — but such additions are not mandatory for its core functionality.

Moreover, the code structure and logic used in the project are modular and reusable. This allows future analysts or developers to build on top of the existing framework for advanced analytics, machine learning, or reporting purposes. The technical choices made in the project ensure long-term sustainability, ease of updates, and compatibility with other systems or software commonly used in data workflows. As a result, the project demonstrates strong technical feasibility, ensuring it can be implemented smoothly with current tools, technologies, and skills available in the market.

3.1.2 Economic Feasibility

The economic feasibility of this project is highly favourable, particularly because of its low development cost and high potential for value generation. All tools and libraries used—such as Python, Jupyter Notebook, and associated packages—are open-source and freely available. This eliminates the need for costly software licenses or proprietary tools, making the initial investment in software virtually zero. The hardware requirements are minimal, meaning there is no necessity for high-end servers or processing machines. A mid-range laptop or desktop is sufficient for the analysis, further reducing cost barriers.

In terms of operational costs, once the data analysis pipeline is built, the cost of maintenance is also minimal. Updates to the dataset or analytical model can be integrated with little effort, and the visualizations or insights can be reproduced with automation if required. The skills required to develop and maintain this project—Python programming, basic data analysis, and visualization—are widely taught and accessible, so organizations do not need to invest heavily in specialized labor. The affordability of training or hiring talent in this domain also supports the project's cost-effectiveness.

From a business perspective, the return on investment (ROI) can be significant. The insights gained from this analysis can help businesses make smarter decisions about inventory, marketing, customer engagement, and sales strategy. For example, by identifying top-selling products or the most valuable customer segments, companies can optimize operations and boost revenue. Similarly, insights into low-performing areas can prevent losses by helping avoid overstocking or wasteful spending. These operational improvements translate into cost savings and increased profitability, justifying the economic viability of the project. In essence, this project offers high value at low cost, making it an economically sound solution for small to mid-sized retail businesses looking to become more data-driven.

3.1.3 Social Feasibility

The social feasibility of the project is strong, as it promotes ethical use of data to enhance customer experience and business responsiveness. Retail businesses rely heavily on understanding customer behaviour, and this project provides a framework to do that in a structured, respectful, and non-invasive way. Rather than using intrusive methods, the project analyzes transactional data that is typically already collected as part of routine operations. This approach aligns well with societal expectations around privacy and data responsibility. By focusing on aggregated trends instead of

individual profiling, the project avoids ethical concerns around surveillance or discrimination.

Furthermore, the outcomes of the project can lead to improved services and personalization for customers. For example, by identifying popular products and understanding seasonal purchasing patterns, businesses can ensure timely availability of products, avoid stockouts, and cater to actual demand more effectively. Insights from customer segmentation can also help retailers tailor their marketing in a way that respects user preferences and behaviours, rather than spamming or irrelevant messaging. This results in better customer satisfaction and builds trust between businesses and their clientele.

On a broader level, encouraging data literacy and the use of open-source tools in business analysis promotes inclusivity and democratization of technology. Small businesses, students, and early-career professionals can all engage with and benefit from projects like this without requiring extensive financial resources or infrastructure. It fosters a culture of innovation and learning within the community. As long as ethical standards are upheld—particularly around data privacy and transparency—the project has a socially beneficial impact. It supports responsible digital transformation, encourages smarter business practices, and enhances the overall relationship between retailers and consumers in a data-driven age.

3.2.1 SOFTWARE FEATURES

Software Features of the Retail Data Analysis Project

1. **User-Friendly Data Input**
   The project is designed to read data from commonly used file formats like Excel (.xlsx), making it easy for users to input retail transaction data without needing database connections or complex data sources.
2. **Automated Data Cleaning**
   The software includes a built-in data cleaning module that automatically handles missing values, filters out invalid transactions (e.g., negative quantities, canceled orders), and standardizes key fields like InvoiceDate and Description. This reduces manual preprocessing and prepares the dataset for accurate analysis.
3. **Descriptive Statistical Analysis**
   It provides detailed statistical summaries of the dataset, such as total sales, number of

customers, number of invoices, average transaction value, and more. This allows users to understand the overall business performance quickly.

4. **Interactive Data Exploration**

   With the help of visual libraries like matplotlib and seaborn, the project produces high-quality plots and graphs that visually represent sales trends, product popularity, revenue by country, and time-based performance. These charts are interactive and can be modified or extended easily.

5. **Sales Trend Analysis**

   The software performs time-series analysis to identify monthly and yearly trends in sales. This helps businesses discover seasonal demand patterns and make data-driven decisions around inventory and marketing.

6. **Product and Customer Ranking**

   It identifies top-selling products and high-value customers using aggregated metrics. This information is useful for inventory planning and customer relationship management.

7. **Customer Segmentation (RFM Analysis)**

   An optional but powerful feature of the software is RFM analysis. It segments customers based on Recency, Frequency, and Monetary value, helping businesses target loyal or at-risk customers with personalized campaigns.

8. **Geographical Analysis**

   The system can break down sales data by country or region, allowing users to identify key markets and potential areas for expansion.

9. **Customizable and Extensible Codebase**

   Since the project is built in Python using Jupyter Notebook, it is fully customizable. Users can modify the logic, add new KPIs, or integrate machine learning models with ease.

10. **Open-Source and Cost-Effective**

    All components of the software are open-source, making the project freely accessible and highly scalable. Businesses can deploy and adapt the solution without any licensing fees.

3.2.2 HARDWARE REQUIREMENTS

- ✓ Processor - Inteli3
- ✓ Hard Disk - 160GB
- ✓ Key Board - Standard Windows Keyboard

- ✓ Mouse - Two or Three Button Mouse
- ✓ Monitor - SVGA
- ✓ RAM - 8GB

3.2.3 SOFTWARE REQUIREMENTS

- ✓ Operating System : Windows 7/8/10
- ✓ Programming Language : Python
- ✓ Library : Pandas, Sklearn, Seaborn, Numpy
- ✓ IDE/Workbench : Visual Code, Jupyter
- ✓ Technology : Python 3.10

CHAPTER 4

SYSTEM ANALYSIS

4.1 EXISTING SYSTEM

Existing systems for customer segmentation in retail primarily rely on basic clustering algorithms like K-Means and DBSCAN to segment customers based on purchasing behavior. While K-Means is widely used, it struggles with handling outliers and non-globular clusters. DBSCAN, on the other hand, is better for detecting density-based clusters and outliers but is less effective when dealing with large datasets or high-dimensional data. These systems typically do not use advanced metrics like Recency, Frequency, and Monetary (RFM), and lack real-time processing capabilities. Furthermore, they fail to adapt to evolving customer behaviors, which limits their effectiveness in dynamic retail environments.

4.1.1 Demerits

- **Limited Clustering Techniques**: Existing systems primarily rely on basic clustering algorithms like K-Means and DBSCAN, which may not capture all complex customer patterns effectively.
- **Handling Outliers**: K-Means struggles to identify and handle outliers, leading to less accurate segmentation in some cases.
- **Lack of Real-Time Processing**: Existing systems often fail to handle real-time data, making them less adaptable to dynamic customer behavior and trends.
- **Scalability Issues**: Traditional methods may struggle with large datasets, leading to inefficiencies in processing and segmentation.
- **No Use of Advanced Metrics**: Existing systems often do not incorporate advanced metrics like Recency, Frequency, and Monetary (RFM), limiting the depth of customer analysis.

- **Limited Flexibility**: These systems are typically static and not adaptable to changing customer behaviors over time.

4.2 PROPOSED SYSTEM

4.2.1 INTRODUCTION

The proposed system is a **Retail Data Analysis and Customer Insights Platform** developed using Python, aimed at transforming raw transactional data into meaningful business intelligence. The goal of the system is to help retailers, analysts, or business owners gain deeper understanding of their sales performance, customer behaviour, and product trends through a structured and automated approach. This system primarily leverages data collected from online or in-store retail transactions and converts it into actionable insights through a combination of data processing, statistical analysis, and visual storytelling.

At the heart of the proposed system is a robust **data preprocessing engine**, responsible for preparing the raw dataset for analysis. Upon receiving the input file (such as an Excel spreadsheet), the system performs a series of cleaning operations. It removes null or missing values, especially for crucial fields like CustomerID, eliminates cancelled transactions marked by a "C" in the invoice number, and filters out invalid records such as negative quantities or unit prices. This ensures that the dataset used for analysis is clean, consistent, and accurate—key prerequisites for deriving trustworthy insights.

Once the data is cleaned, the system proceeds with **descriptive analysis**, computing key performance indicators such as total revenue, number of transactions, customer count, average order value, and more. These metrics are calculated through aggregation functions using Python's pandas library. Furthermore, the system groups the data by dimensions such as products, countries, and customers to identify which items sell the most, which regions contribute the most revenue, and which customers are the most valuable. This layer provides immediate, digestible insights into overall business performance and operational hotspots.

The system also includes a powerful **visual analytics module** that presents the findings in a graphical format using libraries like matplotlib and seaborn. Sales trends are visualized through time-series plots, helping identify seasonal patterns or sales spikes. Bar charts and histograms are used to show product-wise performance, revenue by country, and frequency of customer

purchases. These visuals not only support intuitive understanding but also enhance communication with stakeholders who may not have technical expertise. The visualization layer turns raw data into a business narrative, making it easier to spot opportunities and risks.

A significant component of the proposed system is its **customer segmentation functionality** using **RFM (Recency, Frequency, Monetary) analysis**. This module classifies customers based on how recently they made a purchase, how often they buy, and how much money they spend. Using quantile-based scoring, the system assigns R, F, and M scores to each customer and segments them into categories such as "Loyal Customers," "At-Risk Customers," or "New Customers." These segments can be extremely valuable for crafting targeted marketing campaigns, improving customer retention, and personalizing service delivery.

Another strength of the system is its **modularity and scalability**. As it is built using Python and Jupyter Notebook, the system is fully open-source and easy to extend. New features such as predictive analytics, inventory optimization, or customer lifetime value models can be integrated seamlessly. The code can also be deployed into a business intelligence environment or converted into a web dashboard for real-time access. Moreover, since it operates on local systems using lightweight libraries, it does not require complex hardware or cloud infrastructure, making it cost-effective for small to medium-sized businesses.

4.2.2 Problem Statement and Detailed Description with functionalities

In today's fast-paced and highly competitive retail environment, businesses generate massive volumes of transactional data daily through customer purchases, invoices, and sales activities. However, a significant number of small to mid-sized retail businesses face challenges in utilizing this data effectively due to a lack of analytical tools, technical expertise, or resources. While this data holds valuable insights that can drive strategic decisions—such as identifying top-selling products, understanding customer behavior, and tracking seasonal trends—it often remains unused or under-analyzed, leading to missed opportunities and inefficient operations.The absence of a structured data analysis framework makes it difficult for retail businesses to identify patterns, monitor performance, or anticipate customer needs. Without these insights, businesses may overstock unpopular items, understock high-demand products, misdirect marketing efforts, or fail to retain high-value customers. Moreover, analyzing data manually using spreadsheets is time-consuming, error-prone, and unsustainable as the business grows. There is a pressing need for an

automated, easy-to-use system that can clean, analyze, and visualize retail data, providing decision-makers with meaningful and actionable intelligence. This project aims to address this gap by developing a comprehensive retail data analysis system using Python and open-source tools.

The system focuses on transforming raw sales data into useful insights through data preprocessing, descriptive statistics, customer segmentation, and visual analytics. By automating the analysis process and presenting results in an understandable format, the system enables business owners and analysts to make informed decisions based on real evidence rather than guesswork.

Ultimately, the project seeks to empower retailers to become more data-driven, optimize their operations, and improve overall customer satisfaction and profitability.

4.2.2.1 Detailed Description

This project is a comprehensive **Retail Data Analysis System** built using Python, designed to uncover patterns, trends, and actionable insights from customer transaction data. The main objective is to help businesses understand their sales performance, customer behavior, and product demand using data-driven techniques. Retailers often collect large volumes of data, but without proper tools and analysis, much of this information remains underutilized. This project bridges that gap by converting raw retail data into structured insights through automation, visualization, and intelligent segmentation.

The dataset used in this project typically consists of invoice-level details including InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. The project begins by importing and cleaning this data. It removes null values, filters out canceled transactions (often marked with "C"), and excludes entries with negative or zero values. This preprocessing ensures that the analysis is performed on clean, reliable data, which is crucial for generating accurate results.

Once the data is prepared, the system performs **descriptive analysis** to give users a broad overview of retail performance. It calculates total revenue, number of transactions, customer count, average quantity sold, and other key indicators. This forms the foundation for more specific analysis, where the data is broken down by country, product, and customer to identify top-performing regions, high-demand items, and loyal customers. The project then uses Python's visualization libraries such as matplotlib and seaborn to create graphs and charts, allowing users to explore trends such as monthly sales growth, product seasonality, and purchasing behavior over time.

4.2.2.2 Functionalities

1. **Data Cleaning and Preprocessing:**
   Automatically detects and handles missing values, removes canceled or erroneous transactions, and standardizes data formats (e.g., converting dates to datetime objects).

2. **Descriptive Analytics:**
   Summarizes overall business performance using metrics like total sales, number of unique products, number of customers, and transaction volume. It also includes product-level and country-level revenue breakdowns**.**

3. **Time-Based Analysis:**
   Enables time series analysis to study sales patterns over months or specific date ranges. This helps businesses detect peak seasons, sales drops, or growth opportunities.

4. **Product and Customer Performance Tracking:**
   Identifies the most and least popular products based on frequency and total sales. It also highlights top-spending customers, helping businesses focus on customer retention and engagement.

5. **Data Visualization:**
   Converts complex datasets into easy-to-understand visuals like bar charts, line graphs, pie charts, and heatmaps. These visuals make it simple for stakeholders to interpret trends and patterns.

6. **RFM (Recency, Frequency, Monetary) Analysis:**
   Segments customers based on their buying behavior. Customers are scored based on how recently they bought (Recency), how often they buy (Frequency), and how much they spend (Monetary). This helps businesses identify VIP customers, at-risk customers, and new leads.

7. **Geographical Sales Analysis:**
   Analyzes sales distribution across different countries to identify key markets. This is useful for planning international expansion or customizing regional marketing strategies.

8. **Modularity and Extensibility:**
   The code is modular, allowing developers to plug in additional features such as predictive

analytics, demand forecasting, or inventory management in future versions.

CHAPTER 5

SYSTEM DESIGN

1.1 Project Modules

The Modules of this project are :

- **Data Collection**
  Retail transaction data includes Invoice Number, Stock Code, Quantity, Invoice Date, Unit Price, Customer ID, and Country, with 541,909 records. Data preparation involves addressing missing values, ensuring data consistency, and standardizing formats.
- **Data Preprocessing**
  Missing values are dropped, and outliers are handled. Numerical features are standardized using MinMax or Standard Scaling. Categorical variables are encoded using One-Hot or Label Encoding.
- **Exploratory Data Analysis (EDA)**
  EDA includes statistical summaries, distribution plots, and visualizations to detect patterns, relationships, and anomalies in the dataset. Scatter plots and heatmaps help understand customer behavior.
- **Total Amount**: Quantity × Unit Price
- **Date Splitting**: Year, Month, Day from Invoice Date
- **RFM Metrics**: Recency (R), Frequency (F), Monetary (M)
- These features enhance customer segmentation.
- **Feature Engineering**
  New features are created:
- **Total Amount**: Quantity × Unit Price
- **Date Splitting**: Year, Month, Day from Invoice Date
- **RFM Metrics**: Recency (R), Frequency (F), Monetary (M)

1.2 Project Architecture with UML

1.2.1 Introduction to Object Oriented Methodology

Object-Oriented Methodology (OOM) is a modern and structured approach to software development that organizes a program around its data—objects—rather than functions and logic alone. In the context of your retail data analysis project, OOM provides a powerful way to model real-world entities such as *Customers*, *Invoices*, *Products*, and *Transactions* as independent objects. Each of these objects can contain both data (attributes) and behavior (methods), allowing for clean, modular, and reusable code that reflects how retail operations actually work.

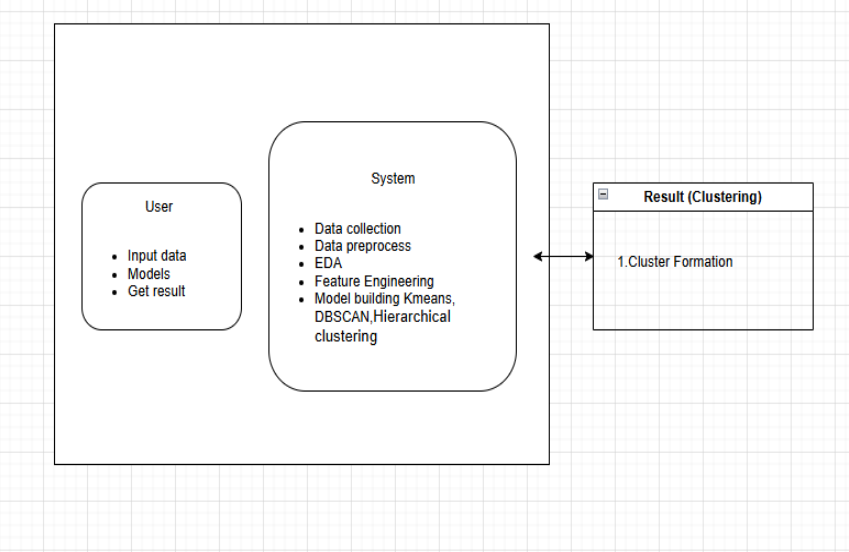


Fig 1:Object Oriented Architecture of RFM

This approach enhances both the design and maintainability of the project. For instance, an object like Customer might store attributes such as CustomerID, Country, and PurchaseHistory, while methods within this object could calculate total spending, classify the customer's RFM score, or determine loyalty status. Similarly, an Invoice object could include attributes like InvoiceDate, ProductList, and TotalAmount, along with methods to validate the transaction or check for returns. By encapsulating functionality within these objects, the system becomes easier to understand, debug, and extend without affecting other parts of the code.

In our project, implementing an object-oriented structure helps transform the analytical workflow into a logical framework that mirrors the business domain. Instead of working with raw data as

flat tables, the code can manipulate high-level entities with defined responsibilities. This modularity not only improves readability but also prepares the system for future enhancements such as integrating predictive analytics, building dashboards, or even transitioning the logic into a full-fledged software product or web application. Ultimately, the object-oriented methodology provides a robust foundation for building scalable, intuitive, and maintainable software systems for data-driven retail analysis.

1.2.2 Key Object-Oriented Components for RFM

1. User

This represents the end-user (analyst, business owner, or stakeholder) interacting with the system.

Responsibilities/Functions:

- Input Data: The user provides raw retail transaction data (e.g., customer purchase history).
- Models: The user can choose or run different clustering models (K-Means, DBSCAN, Hierarchical).
- Get Result: Receives the processed output in the form of customer segments or clusters.

  The User is the entry point of the system—responsible for initiating the analysis process with minimal technical effort.

2. **System**

This is the core engine of your RFM project. It consists of a sequence of stages that transform raw data into actionable insights. Subcomponents:

- Data Collection: Importing or loading datasets (Excel, CSV, etc.).
- Data Preprocessing: Handling missing values, filtering invalid records, removing returns, and formatting data for analysis.
- EDA (Exploratory Data Analysis): Understanding data patterns, visualizing distributions, checking correlations, etc.

- Feature Engineering: Computing RFM metrics (Recency, Frequency, Monetary) and scaling data for clustering.
- Model Building: Applying clustering algorithms like:
  - K-Means: Partitions customers into K groups based on similarity.
  - DBSCAN: Detects dense clusters and isolates noise or outliers.
  - Hierarchical Clustering: Builds nested clusters for detailed customer segmentation.

This module automates complex machine learning tasks and performs analytical computation required for customer segmentation.

3. **Result (Clustering)**

This is the output interface that reflects the insights derived from the clustering models.

- Cluster Formation: Customers are grouped into distinct clusters based on their RFM scores. For example: "High-value loyal customers", "New but promising", or "At-risk customers".

These clusters help businesses make data-driven decisions for targeted marketing, retention strategies, and inventory planning

Benefits of Object-Oriented Methodology in RFM

- The architecture clearly separates the roles of the user, the system, and the output. This modularity improves understanding, development, testing, and future maintenance.
- The system handles the complete pipeline—right from raw data ingestion to model execution—allowing non-technical users to perform advanced analysis without deep coding skills.
- Since multiple clustering algorithms are supported, the system can be expanded to include more models or custom segmentation logic without major changes to the user interface.
- The final output—cluster formation—enables business stakeholders to understand customer behavior patterns, which supports better decision-making in marketing, sales, and customer engagement.
- By abstracting the technical complexity away from the user, the architecture supports ease of use, making it suitable for small to mid-sized businesses or academic demonstrations.

5.2.2 SOFTWARE ARCHITECTURE WITH UML LANGUAGE

5.2.2.1 Introduction to UML

The Unified Modelling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language

(UML) was designed to respond to these needs. Simply, Systems design refers to the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements which can be done easily through UML diagrams.

5.2.2.2 Object-Oriented Concepts

UML can be described as the successor of object-oriented (OO) analysis and design.

An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and they also form a hierarchy to model the real-world system. The hierarchy is represented as inheritance and the classes can also be associated in different ways as per the requirement.

Objects are the real-world entities that exist around us and the basic concepts such as abstraction, encapsulation, inheritance, and polymorphism all can be represented using UML.

UML is powerful enough to represent all the concepts that exist in object-oriented analysis and design. UML diagrams are representation of object-oriented concepts only. Thus, before learning UML, it becomes important to understand OO concept in detail.

Following are some fundamental concepts of the object-oriented world −

- **Objects** − Objects represent an entity and the basic building block.
- **Class** − Class is the blue print of an object.
- **Abstraction** − Abstraction represents the behavior of an real world entity.
- **Encapsulation** − Encapsulation is the mechanism of binding the data together and hiding them from the outside world.
- **Inheritance** − Inheritance is the mechanism of making new classes from existing ones.
- **Polymorphism** − It defines the mechanism to exists in different forms.

5.2.2.3 OO Analysis and Design

OO can be defined as an investigation and to be more specific, it is the investigation of objects. Design means collaboration of identified objects.

Thus, it is important to understand the OO analysis and design concepts. The most important purpose of OO analysis is to identify objects of a system to be designed. This analysis is also done for an existing system. Now an efficient analysis is only possible when we are able to start thinking in a way where objects can be identified. After identifying the objects, their relationships are identified and finally the design is produced.

The purpose of OO analysis and design can described as −

- Identifying the objects of a system.
- Identifying their relationships.
- Making a design, which can be converted to executables using OO languages.

There are three basic steps where the OO concepts are applied and implemented. The steps can be defined as

OO Analysis→ OO Design→ OO implementation using OO languages

The above three points can be described in detail as −

- During OO analysis, the most important purpose is to identify objects and describe them in a proper way. If these objects are identified efficiently, then the next job of design is easy. The objects should be identified with responsibilities. Responsibilities are the functions performed by the object. Each and every object has some type of responsibilities to be performed. When these responsibilities are collaborated, the purpose of the system is fulfilled.
- The second phase is OO design. During this phase, emphasis is placed on the requirements and their fulfilment. In this stage, the objects are collaborated according to their intended association. After the association is complete, the design is also complete.
- The third phase is OO implementation. In this phase, the design is implemented using OO languages such as Java, C++, etc.

5.2.2.4 Role of UML in OO Design

UML is a modeling language used to model software and non-software systems. Although UML is used for non-software systems, the emphasis is on modeling OO software applications. Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc. Now whatever be the aspect, the artifacts are nothing but objects.

Hence, the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement.

In this project, basic UML diagrams have been explained

1) Use Case Diagram
2) Class Diagram

3) Sequence Diagram

4) Collaboration Diagram

5) Activity Diagram

6) State Chart Diagram

7) Component Diagram

8) Deployment Diagram

5.2.2.1 USE CASE DIAGRAM FOR ORF

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms.

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed, the use case and goal are sometimes considered to be synonymous.The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

5.2.2.2 Parts of Use Case Diagram

5.2.2.3 System boundary boxes (optional)

A rectangle is drawn around the use cases, called the system boundary box, to indicate the scope of system. Anything within the box represents functionality that is in scope and anything outside the box is not **Relationships.**

Include

In one form of interaction, a given use case may include another. "Include is a Directed

Relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case".

The first use case often depends on the outcome of the included use case. This is useful for extracting truly common behaviours from multiple use cases into a single description. The notation is a dashed arrow from the including to the included use case, with the label "«include»". This usage resembles a macro expansion where the included use case behavior is placed inline in the base use case behavior. There are no parameters or return values. To specify the location in a flow of events in which the base use case includes the behavior of another, you simply write include followed by the name of use case you want to include, as in the following flow for track order.

Extend

In another form of interaction, a given use case (the extension) may extend another. This relationship indicates that the behavior of the extension use case may be inserted in the extended use case under some conditions. The notation is a dashed arrow from the extension to the extended use case, with the label "«extend»". The notes or constraints may be associated with this relationship to illustrate the conditions under which this behavior will be executed. Modelers use the «extend» relationship to indicate use cases that are "optional" to the base use case. Depending on the modeler's approach "optional" may mean "potentially not executed with the base use case" or it may mean "not required to achieve the base use case goal".

Use case Diagram for RFM

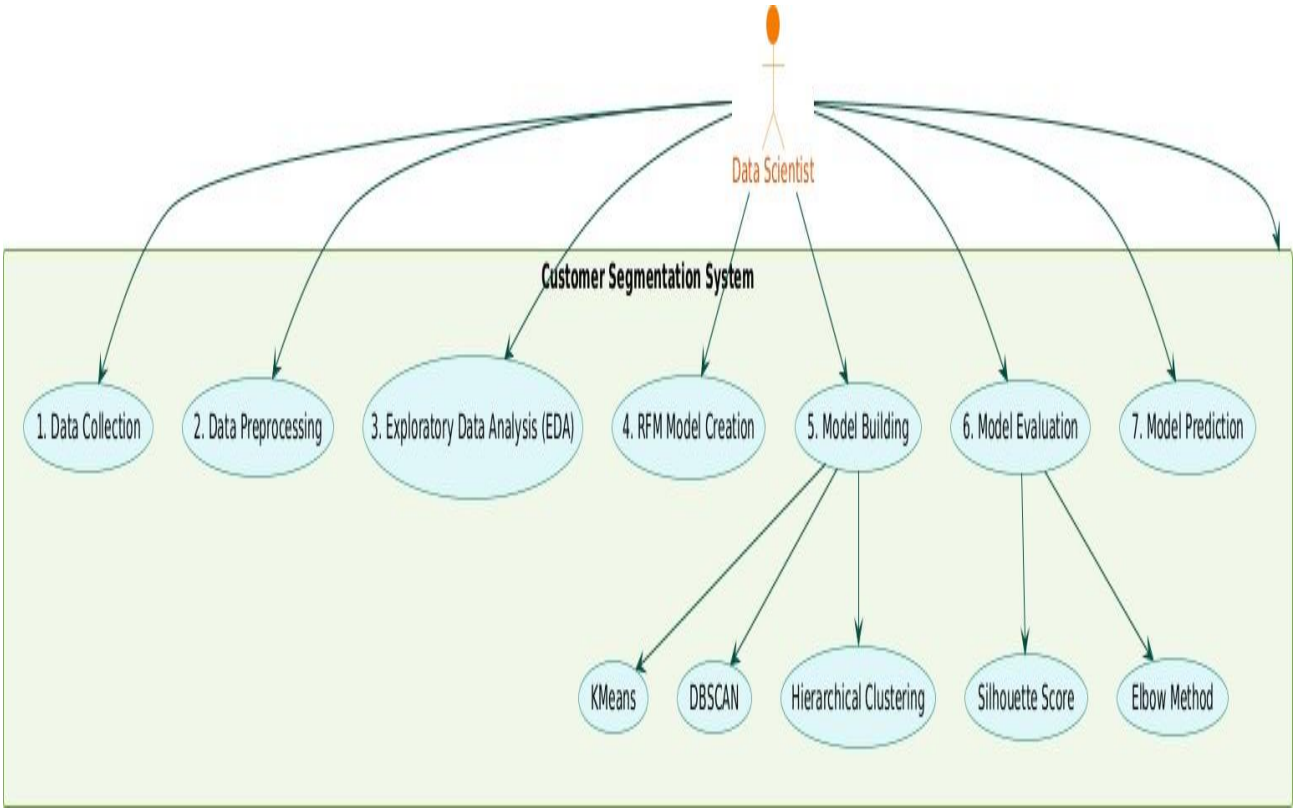


Fig 2: USE CASE DIAGRAM FOR RFM

5.2.2.1 CLASS DIAGRAM for RFM

UML class diagrams model static class relationships that represent the fundamental architecture of the system. Note that these diagrams describe the relationships between classes, not those between specific objects instantiated from those classes. Thus the diagram applies to all the objects in the system.

A class diagram consists of the following features:

- **Classes:** These titled boxes represent the classes in the system and contain information about the name of the class, fields, methods and access specifies. Abstract roles of the Class

in the system can also be indicated

- **Interfaces:** These titled boxes represent interfaces in the system and contain information about the name of the interface and its methods. Relationship Lines that model the relationships between classes and interfaces in the system.

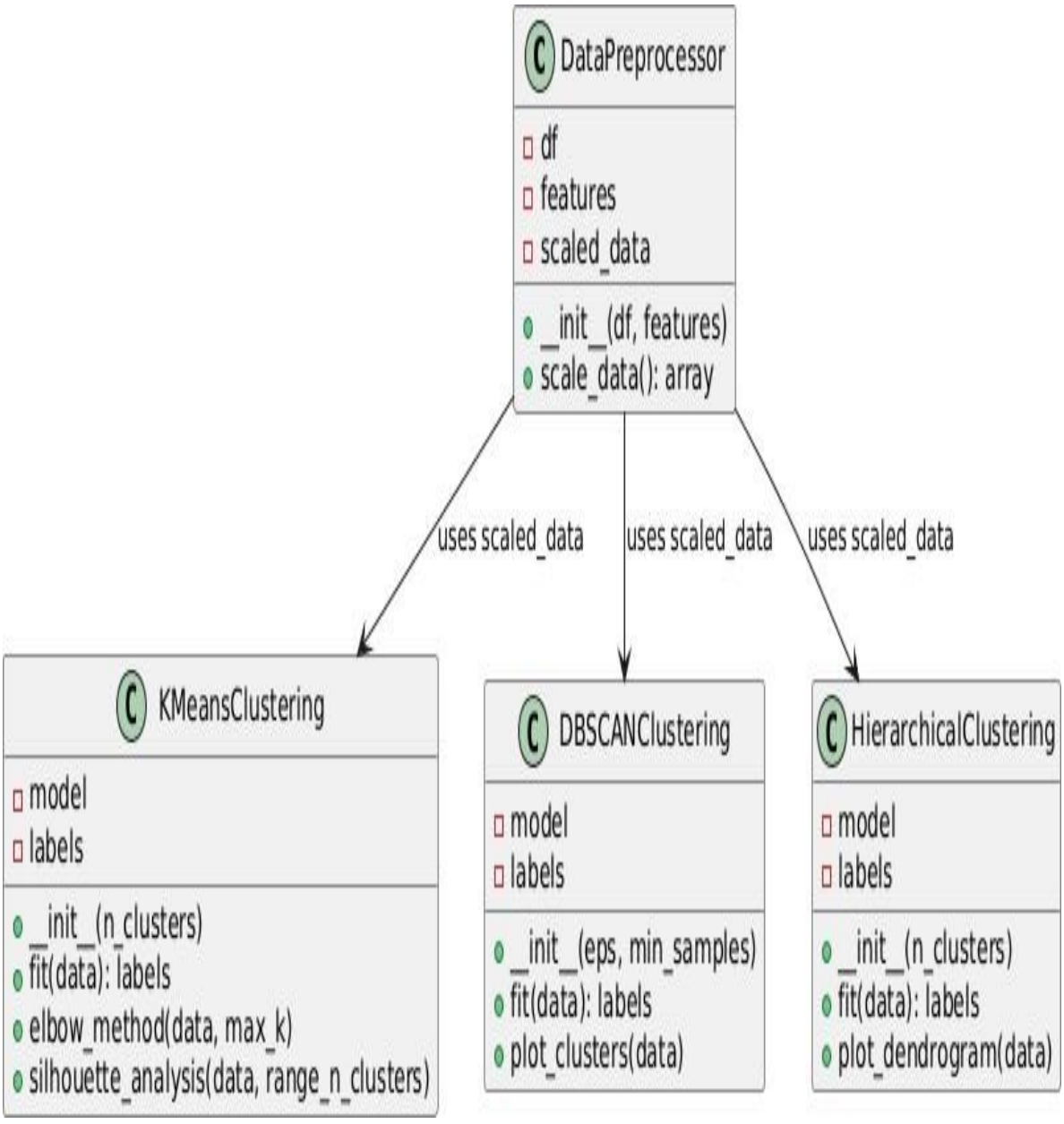


Fig 3 Class Diagram of RFM

- **Dependency:** A dotted line with an open arrowhead that shows one entity depends on the behavior of another entity. Typical usages are to represent that one class instantiates

another or that it uses the other as an input parameter.

5.2.2.2 SEQUENCE DIAGRAM FOR THE RFM

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behavior of a system.

5.2.2.1.1 Elements of sequence diagram

The sequence diagram is an element that is used primarily to showcase the interaction that occurs between multiple objects. This interaction will be shown over certain period of time. Because of this, the first symbol that is used is one that symbolizes the object.

1. Lifeline

A lifeline will generally be generated, and it is a dashed line that sits vertically, and the top will be in the form of a rectangle. This rectangle is used to indicate both the instance and the class. If the lifeline must be used to denote an object, it will be underlined.

2. Messages

To showcase an interaction, messages will be used. These messages will come in the form of horizontal arrows, and the messages should be written on top of the arrows. If the arrow has a full head, and it’s solid, it will be called a synchronous call. If the solid arrow has a stick head, it will be an asynchronous call. Stick heads with dash arrows are used to represent return messages.

3. Objects

Objects will also be given the ability to call methods upon themselves, and they can add net activation boxes. Because of this, they can communicate with others to show multiple levels of processing. Whenever an object is eradicated or erased from memory, the "X" will be drawn at the lifeline's top, and the dash line will not be drawn beneath it. This will often occur as a result of a message. If a message is sent from the outside of the diagram, it can be used to define a message that comes from a circle that is filled in. Within a UML based model, a Super step is a collection of steps which result from outside stimuli.

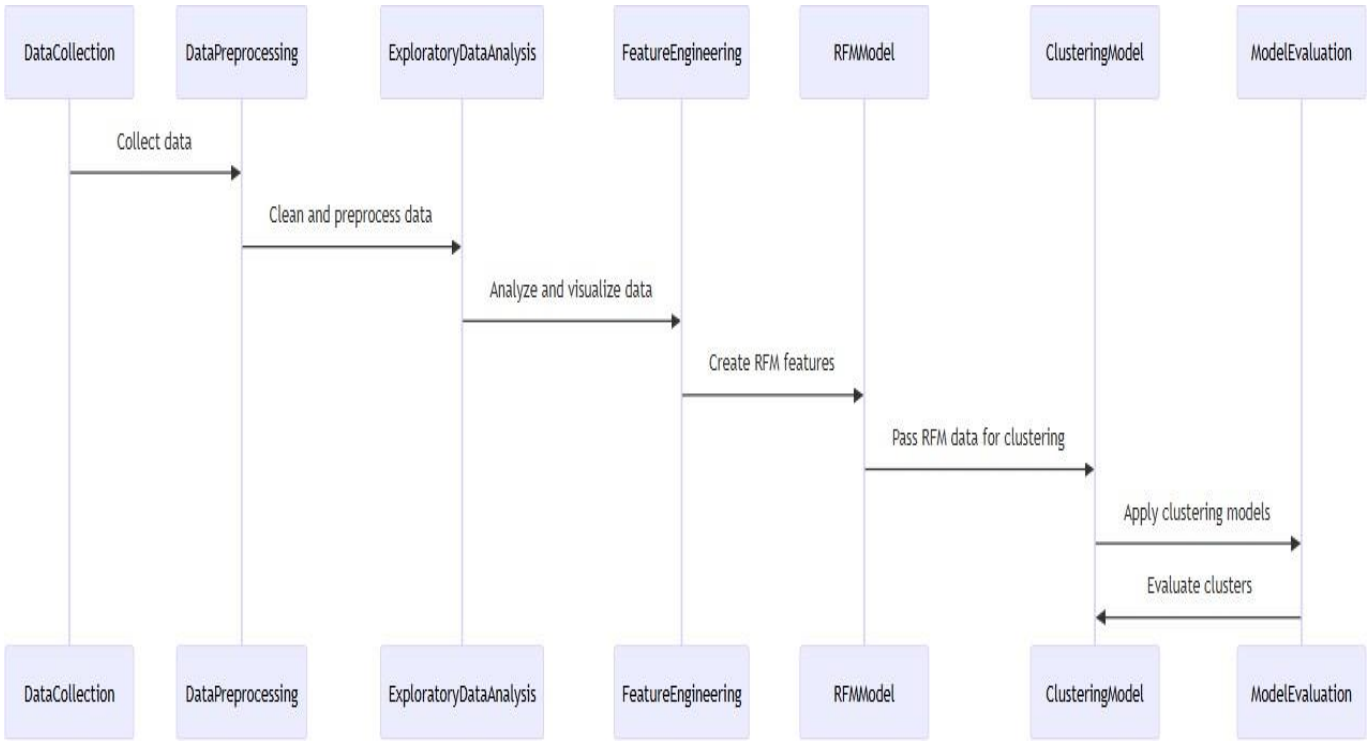


Fig 4: Sequence Diagram of RFM

5.2.2.2 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

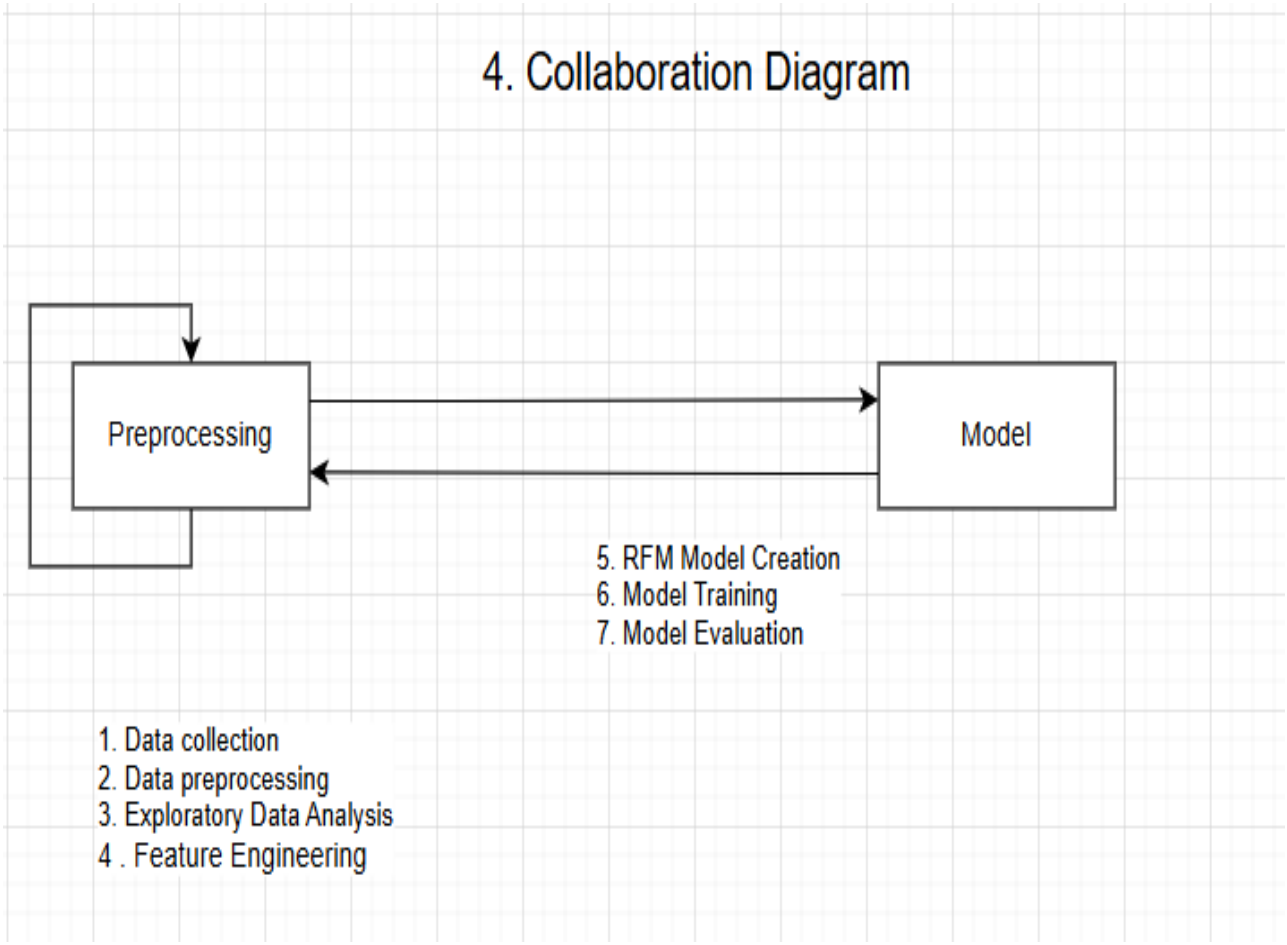


Fig 5 Collaboration Diagram

5.2.2.7 **ACTIVITY DIAGRAM**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

How to draw Activity Diagram?

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagram are not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim lane etc. Before drawing an activity diagram we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities we need to understand how they are associated with constraints and conditions. So before drawing an activity diagram we should identify the following elements.

- Activities
- Association
- Conditions
- Constraints

The following are the basic notational elements that can be used to make up a diagram:

Initial state

An initial state represents a default vertex that is the source for a single transition to the default state of a composite state. There can be at most one initial vertex in a region. The outgoing

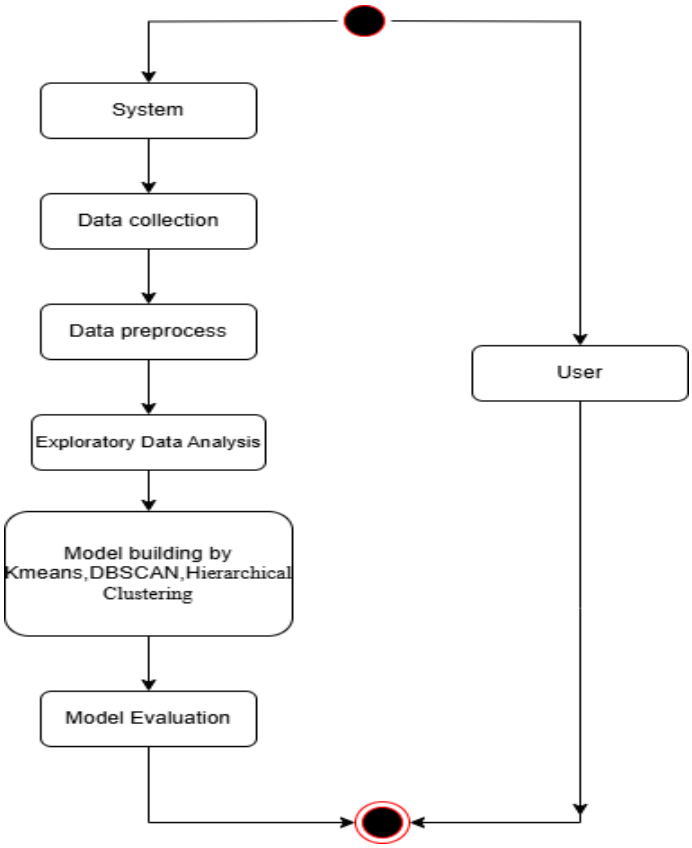


Fig 6: STATE CHART DIAGRAM

transition from the initial vertex may have a behavior, but not a trigger or guard. It is represented by Filled circle, pointing to the initial state.

1. Final state

A special kind of state signifying that the enclosing region is completed. If the enclosing region is directly contained in a state machine and all other regions in the state machine also are completed, then it means that the entire state machine is completed. It is represented by Hollow circle containing a smaller filled circle, indicating the final state.

2. Rounded rectangle

It denotes a state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activities that are done in that state are indicated.

3. Arrow

It denotes transition. The name of the event (if any) causing this transition labels the arrow body.

4. Steps To Construct Activity Diagram

- Identify the preconditions of the workflow
- Collect the abstractions that are involved in the operations

STATE CHART DIAGRAM

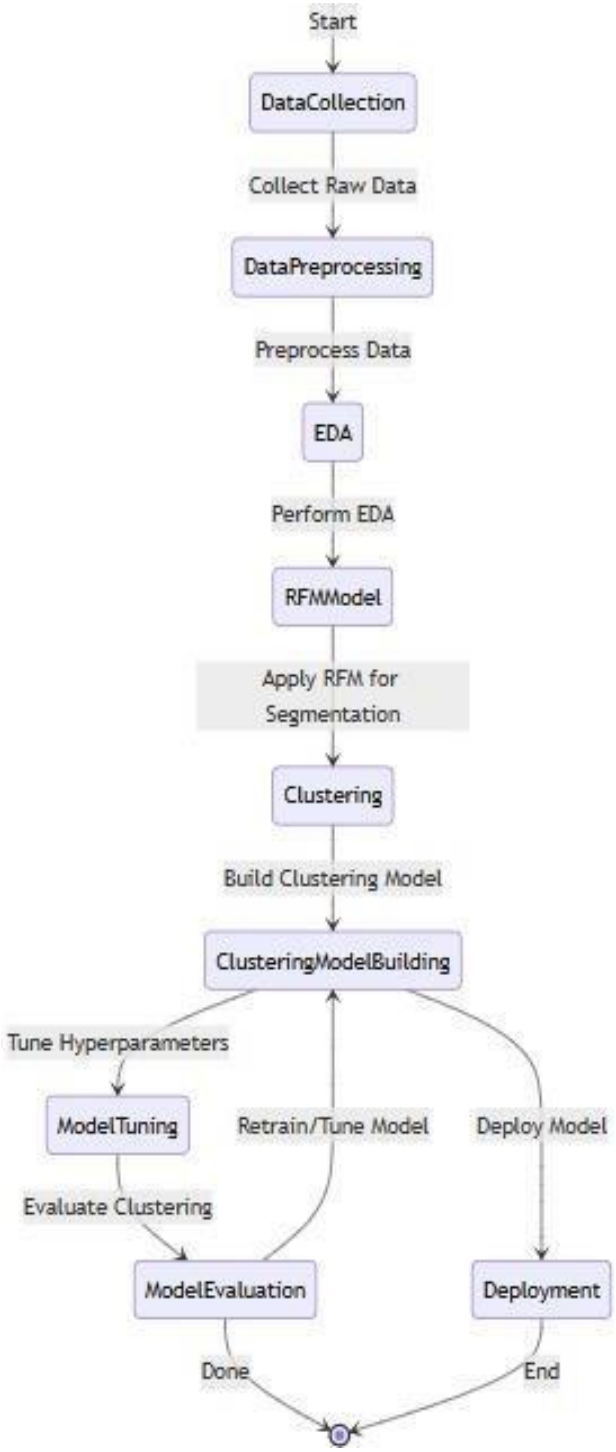


Fig 7:State Chart Diagram of RFM

5.2.2.3.8 COMPONENT DIAGRAM

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

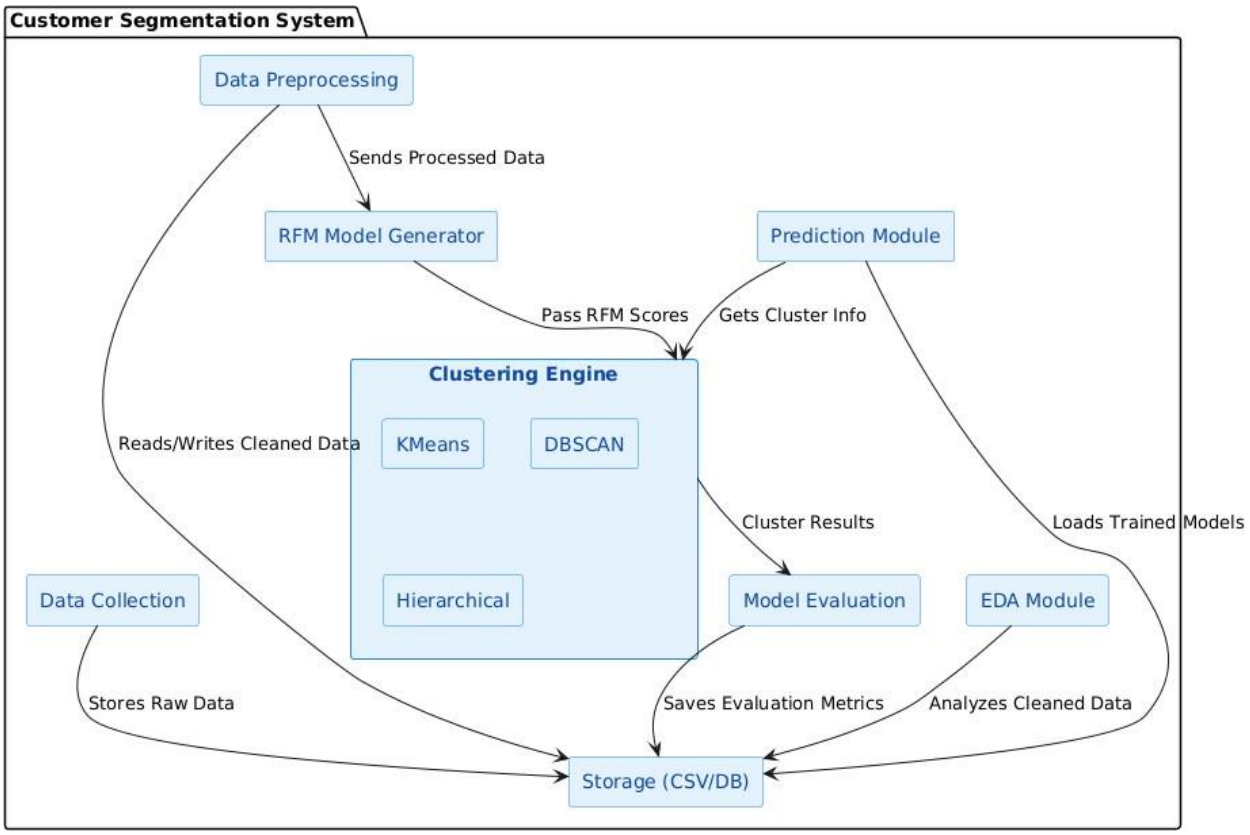


Fig 8: Component Diagram

1.1.1.1 DEPLOYMENT DIAGRAM

Deployment diagram represents the deployment view of a system .It is related to the Component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical Hardware's used to deploy the Applications.

DEPLOYMENT DIGRAM OF RFM

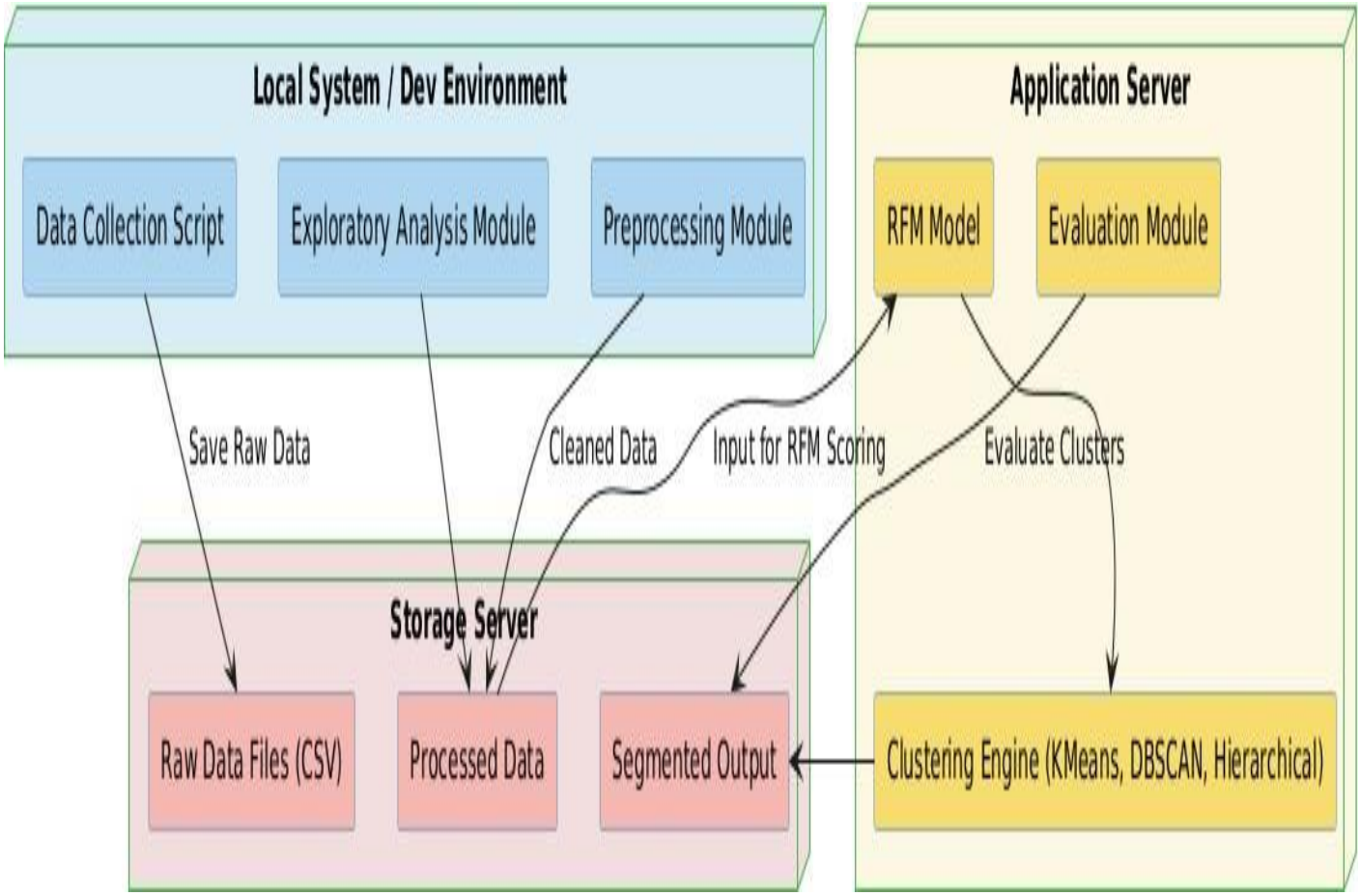


Fig 9: Deployment Diagram

CHAPTER 6

PROJECT IMPLEMENTATION

6.1 PYTHON

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Today, Python is very high in demand and all the major companies are looking for great Python Programmers to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies. When we are developing this tutorial in 2022, there is a high shortage of Python Programmers where as market demands more number of Python Programmers due to it's application in Machine Learning, Artificial Intelligence etc.

What Is A Script?

Up to this point, I have concentrated on the interactive programming capability of Python. This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode

Scripts are reusable

Basically, a script is a text file containing the statements that comprise a Python program. Once you have created the script, you can execute it over and over without having to retype it each time.

Scripts are editable

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor. Then you can execute each of the individual versions. In this way, it is easy to create different programs with a minimum amount of typing.

You will need a text editor

Just about anytext editor will suffice for creating Python script files.

You can use *Microsoft Notepad*, *Microsoft WordPad*, *Microsoft Word*, or just about any word processor if you want to.

Difference between a script and a program

Script:

Scripts are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or byte code, whereas the applications they control are traditionally compiled to native machine code.

Program:

The program has an executable form that the computer can use directly to execute the instructions.

The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled)

Python

What is Python? Chances you are asking yourself this. You may have found this book because you want to learn to program but don't know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to "big name" languages. Hopefully I can explain it for you.

Python concepts

If you're not interested in the how and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it's a great one to start programming with.

- Open source general-purpose language.
- Object Oriented, Procedural, Functional
- Easy to interface with C/ObjC/Java/Fortran

• Easy-is to interface with C++ (via SWIG)

• Great interactive environment

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Possum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Possum still holds a vital role in directing its progress.

Python Features

Python's features include

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is veryportable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide varietyof hardware platforms and has the same interface on all platforms.
- **Extendable** − you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- ✧ It supports functional and structured programming methods as well as OOP.
- ✧ It can be used as a scripting language or can be compiled to byte-code for building large applications.
- ✧ It provides very high-level dynamic data types and supports dynamic type checking.
- ✧ IT supports automatic garbage collection.
- ✧ It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Dynamic vs. Static

Types Python is a dynamic-typed language. Many other languages are static typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of "thing" each data value is.

For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a "float" type.

This tells the compiler that the only data that can be used for that variable must be a floating point number, i.e. a number with a decimal point.

If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program.

Python, however, doesn't require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating point number) you need in your program.

With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment).

If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double.

With Python, it doesn't matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values.

For example, say you are dividing two numbers. One is a floating point number and one is an integer. Python realizes that it's more accurate to keep track of decimals so it automatically calculates the result as a floating point number

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

- Numpy
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

PythonLists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

PythonTuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

PythonDictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square races ([]).

Different modes in python

Python has two basic modes: normal and interactive.

The normal mode is the mode where the scripted and finished .pie files are run in the Python interpreter.

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole

Python Packages

NUMPY

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working

with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

PANDAS

Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

MATPLOTLIB

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

SCIKIT AND SKLEARN

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon **NumPy, SciPy** and **Matplotlib**.

Python Modules

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

There are two types of errors that you will encounter. Syntax errors occur when the form

of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values. There are three types of functions in python:

Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

- **global names** of a module
- **local names** in a function or method invocation
- **built-in names**: this namespace contains built-in functions (e.g. abs(), camp(), ...) and built-in exception names

Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

6.2 Clustering in Machine Learning

Clustering is a core technique in unsupervised machine learning used to discover inherent groupings within a dataset. The goal is to assign data points into clusters such that items in the same cluster are more similar to each other than to those in other clusters. Clustering can reveal natural structures, provide insights into data distribution, and enable more targeted decision-

making in applications like customer segmentation, image processing, or anomaly detection.

There are various clustering algorithms available, each with unique mechanisms and assumptions. Among the most widely used are **K-means Clustering**, **Hierarchical Clustering**, and **DBSCAN (Density-Based Spatial Clustering of Applications with Noise.**

1. K-means Clustering

Overview

K-means is a **partition-based clustering algorithm** that divides a dataset into **K non-overlapping clusters**, each represented by a **centroid** (the mean of points in the cluster). It aims to minimize the **intra-cluster variance** or the **sum of squared distances** between points and their respective cluster centroids.

Key Components

- **Centroid**: The center of a cluster, calculated as the mean of all points assigned to that cluster. It's updated in each iteration.
- **Distance Metric**: Typically uses **Euclidean distance** to determine the nearest centroid for each point.
- **Cluster Assignment**: Each point is assigned to the cluster with the nearest centroid.
- **Initialization**: K-means can be sensitive to the initial placement of centroids. Common strategies include **random initialization** and **K-means++** (a smarter initialization method that improves convergence).
- **Convergence Criteria**: The algorithm stops when centroids no longer change significantly or a maximum number of iterations is reached.

Limitations

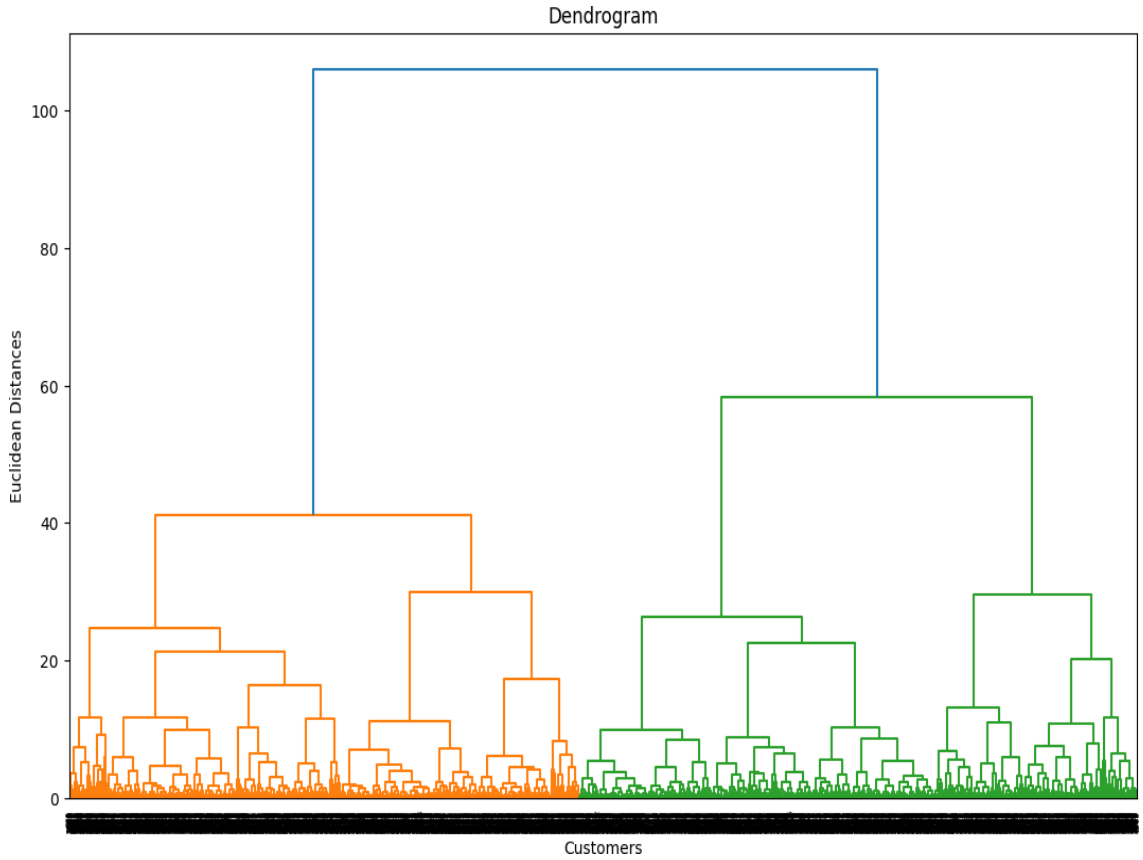
- Cannot detect **non-globular shapes** of clusters.
- **K must be specified** beforehand.
- Sensitive to **outliers** and **initial conditions**.

2. Hierarchical Clustering

Overview

Hierarchical clustering creates a **tree-like structure** of nested clusters called a **dendrogram**. There are two types:

- **Agglomerative (bottom-up)**: Starts with individual points as clusters and merges the closest pairs.
- **Divisive (top-down)**: Starts with one cluster and splits it recursively.



Key Components

- **Dendrogram**: A tree that shows the sequence of merges or splits. Cutting the tree at a specific height gives the final clusters.
- **Linkage Methods** (used to determine distance between clusters):
  - **Single Linkage**: Minimum distance between points in two clusters.
  - **Complete Linkage**: Maximum distance.
  - **Average Linkage**: Mean distance.
  - **Ward's Method**: Minimizes the total variance within clusters.
- **Distance Metrics**: Common metrics include Euclidean, Manhattan, and cosine distance.
- **No Fixed K**: Unlike K-means, the number of clusters can be decided by inspecting the dendrogram.

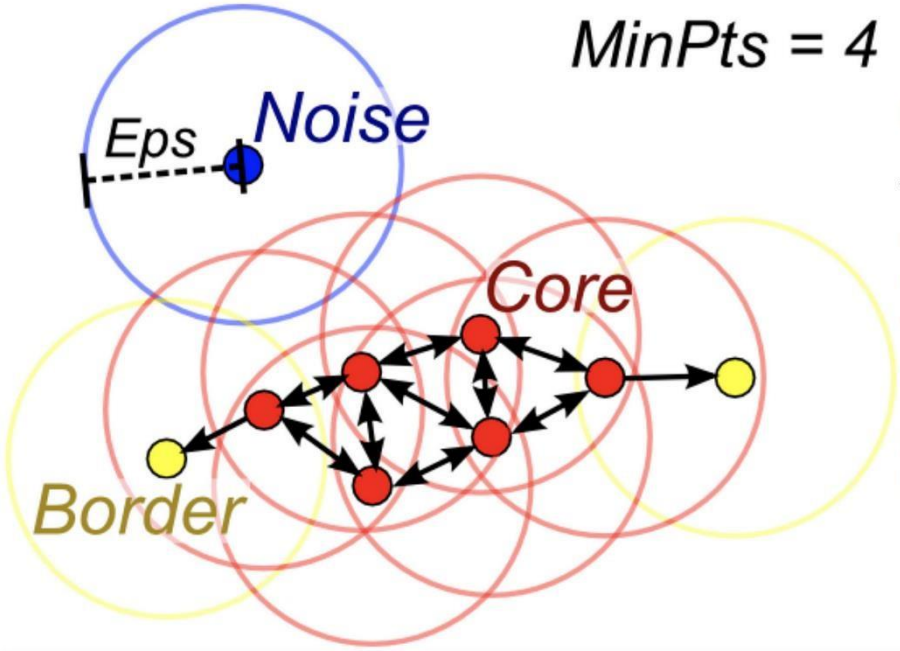
Limitations

- **Scalability**: Not efficient for large datasets (complexity can be $O(n^2)$).
- Once clusters are merged or split, the decision **cannot be reversed**.
- Can be affected by **noise** and **outliers**.

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Overview

DBSCAN is a **density-based clustering** method that groups together points that are closely packed (have many nearby neighbors) and marks points in low-density areas as **noise**. It is capable of finding **arbitrarily shaped clusters** and is **robust to outliers**.



Key Components

- **ε (Epsilon)**: The radius around a point within which neighbors are considered.
- **MinPts**: The minimum number of points required within ε to form a **dense region**.
- **Core Point**: A point with at least MinPts within its ε neighborhood.
- **Border Point**: A point that is within the ε neighborhood of a core point but has fewer than MinPts in its own neighborhood.
- **Noise Point** (Outlier): A point that is **neither** a core point **nor** a border point. It does not belong to any cluster.
- **Density Connectivity**: Points are part of the same cluster if they are **density-connected** via a chain of neighboring core points.

Strengths

- No need to specify number of clusters beforehand.
- Identifies **noise points** automatically.
- Can detect **non-linear shapes** of clusters.

Limitations

- Performance depends on choosing appropriate **ε** and **MinPts**.

- Not ideal for datasets with **varying densities** or **high-dimensional** data.

Conclusion

Each clustering algorithm offers a unique approach to discovering structure in data. **K-means** is simple and fast, best for well-separated spherical clusters. **Hierarchical clustering** is useful for building nested relationships without predefining cluster count. **DBSCAN**, meanwhile, excels at finding clusters of varying shapes and identifying noise, especially in spatial datasets. The choice of algorithm should be guided by the nature of the dataset, domain-specific needs, and desired interpretability.

6.3 Silhouette Score

The **Silhouette Score** is a metric used to evaluate the **cohesion and separation** of clusters. It measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation).

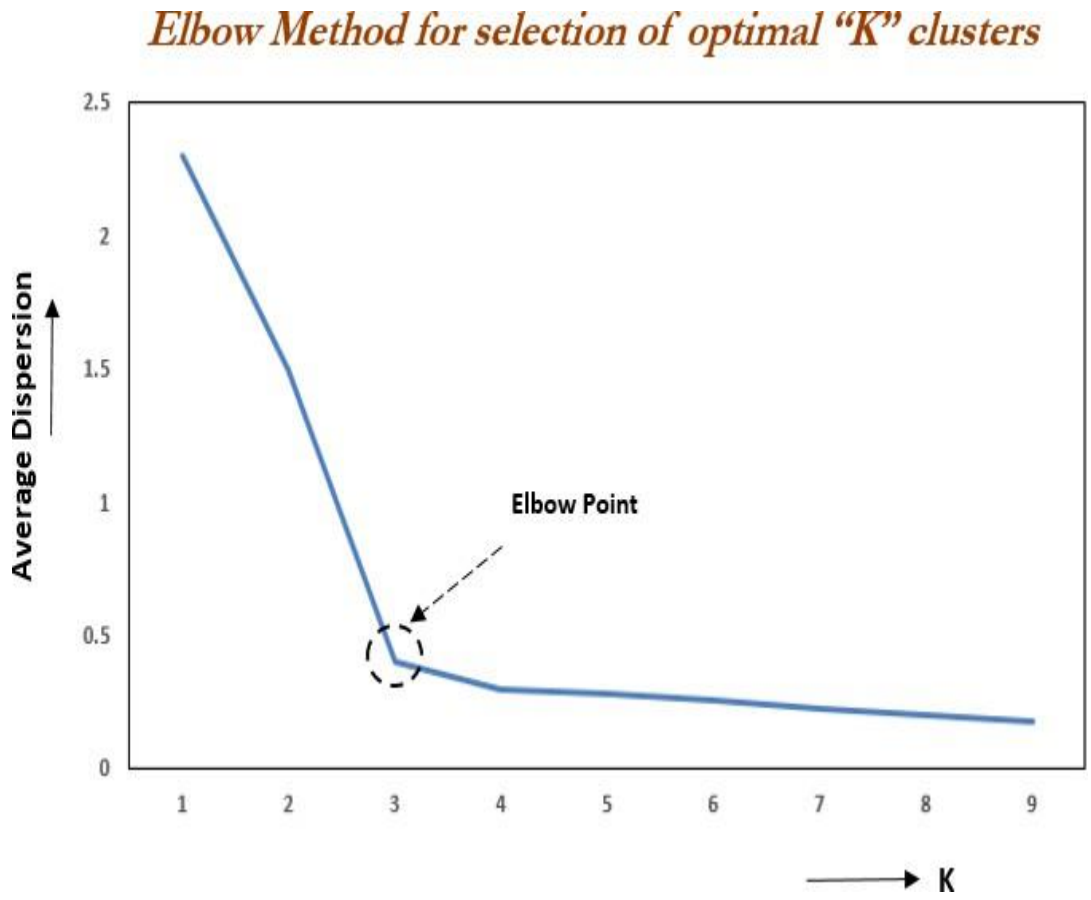
The score ranges from **-1 to 1**:

- A score close to **1** indicates the point is well matched to its own cluster and poorly matched to neighboring clusters.
- A score around **0** means the point is on or very close to the decision boundary between clusters.
- A score close to **-1** indicates the point may have been assigned to the wrong cluster.

The **average silhouette score** over all data points is used to assess the overall clustering quality. Higher scores suggest better-defined clusters. This method is especially helpful when comparing different clustering results or determining the optimal number of clusters.

6.3 Elbow Method

7 Choosing the optimal number of clusters is a crucial step in any unsupervised learning algorithm. Since we don't have predefined cluster counts in unsupervised learning, we need a systematic approach to determine the best k value. The **Elbow Method** is a popular technique used for this purpose in K-Means clustering.



7.1 Code Generation and Validation

Code Generation and Validation for RFM Analysis on Retail Data

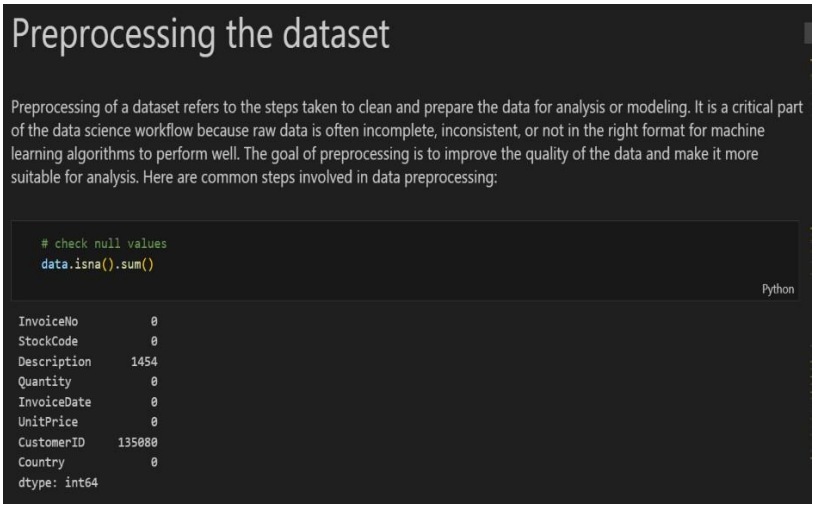
7.1.1 Data Collection

Data collection involves gathering **retail transaction data**, which includes **Invoice Number, Stock Code, Quantity, Invoice Date, Unit Price, Customer ID, and Country**. The dataset consists of **541,909 records with 8 features**. The data is sourced from **historical retail transactions**, capturing customer purchasing behavior. Ensuring data completeness is crucial, as missing or incorrect records can impact clustering results. The collected data is prepared for further processing by addressing missing values, standardizing formats, and ensuring data consistency.

7.1.2 Data Preprocessing

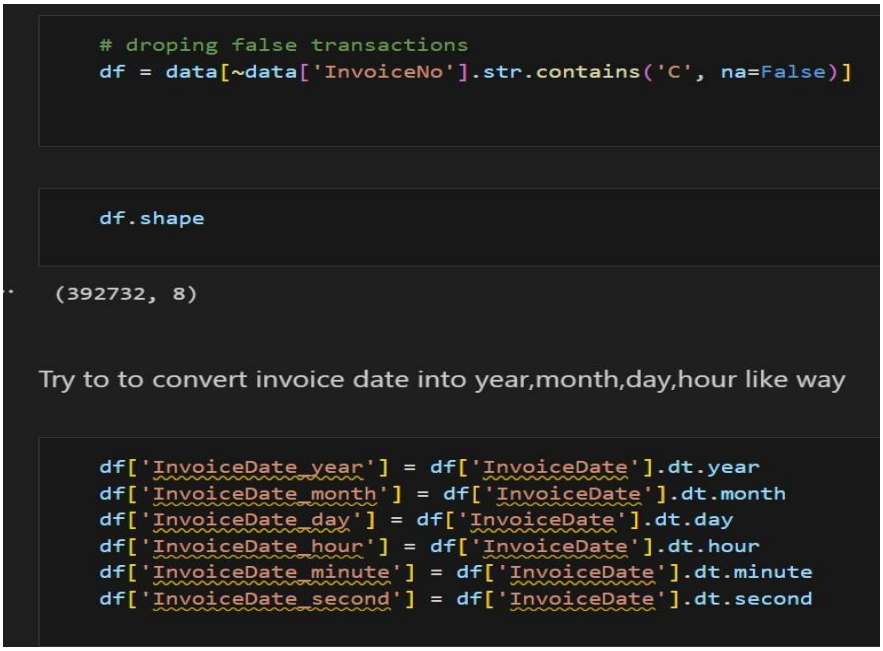
In this stage, **missing values are dropped**, ensuring that only valid and complete records are used. Data preprocessing also includes **outlier detection and handling**, ensuring the dataset

remains representative of actual purchasing behavior. To standardize numerical features like **Unit Price and Quantity**, appropriate **scaling techniques** such as **MinMax Scaling** or **Standard Scaling** are applied. Categorical variables, if needed, are encoded using **One-Hot Encoding** or **Label Encoding** for better model compatibility.



Data Preprocessing

- Checked for null values using data.isna().sum().
- Identified missing values in Description and CustomerID columns.
- Cleaned and prepared the dataset for analysis.



1. □ Removing False Transactions

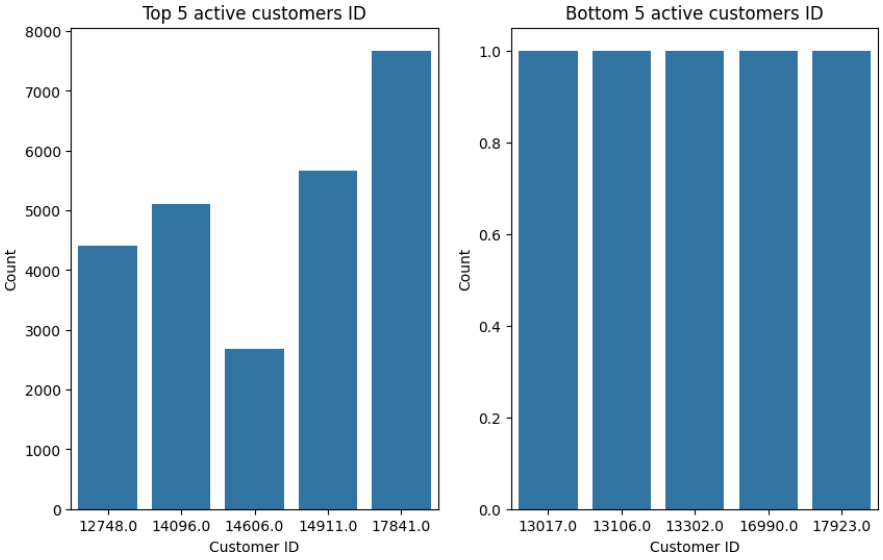
- The dataset may contain credit note transactions or cancellations, which are usually marked with an 'InvoiceNo' starting with the letter 'C'.

2. □ Date-Time Feature Extraction

- The InvoiceDate column is converted into separate components to enhance time-based analysis.

Python

  - This breakdown allows for detailed time-based insights such as:
    - Seasonal trends
    - Peak hours/days for transactions
    - Customer behavior over time



📈 Customer Activity Analysis

The bar charts display the **Top 5** and **Bottom 5** most active customers based on transaction counts.

- **Top 5 Active Customers:**

  Customer ID 17841.0 is the most active with nearly **8000 transactions**, followed by others with high activity levels.

- **Bottom 5 Active Customers:**

  Each of these customers made only **1 purchase**, indicating minimal engagement.

7.1.3 Exploratory Data Analysis (EDA)

EDA is performed to **analyze individual columns, detect patterns, and identify anomalies**. Statistical summaries, **distribution plots, histograms, and box plots** are used to understand the nature of the dataset. This step helps identify trends, relationships between features, and any inconsistencies in data distribution. Visualization techniques such as **scatter plots, correlation heatmaps, and pair plots** provide insights into customer purchasing behavior and transaction patterns.

1.4 Feature Engineering

Feature engineering involves **creating new meaningful features** to enhance the model's performance. The following features are derived:

- **Total Amount:** Computed as **Quantity × Unit Price** for each transaction.
- **Date Splitting:** The **Invoice Date** is split into **Year, Month, and Day**, enabling time-based analysis.
- **RFM Model Features:**
  - **Recency (R):** Number of days since the customer's last purchase.
  - **Frequency (F):** Total number of purchases made by the customer.
  - **Monetary Value (M):** Total spending of the customer.

These features help in defining customer behavior, which is essential for segmentation.

7.1.4 RFM Model Creation

Using **Recency, Frequency, and Monetary (RFM) metrics**, customers are segmented into distinct groups based on their purchasing behavior. The RFM values are computed for each customer. The transformed RFM dataset serves as the input for clustering models, enabling the identification of key customer segments.

1.6 Model Training and Segmentation

Clustering models are trained on three different feature sets:

1. **Recency and Monetary (RM) Model**
2. **Frequency and Monetary (FM) Model**
3. **Recency, Frequency, and Monetary (RFM) Model**

For each model, clustering is performed using:

- **K-Means Clustering with the Elbow Method:** Determines the optimal number of clusters by analyzing the inertia curve.
- **K-Means Clustering with the Silhouette Score:** Measures cluster separation and cohesion to optimize cluster count.
- **DBSCAN and Hierarchical Clustering:** Alternative clustering methods for noise-resistant segmentation and hierarchical group identification.

7.1.6 K-Means with silhouette_score | RM |

Code:

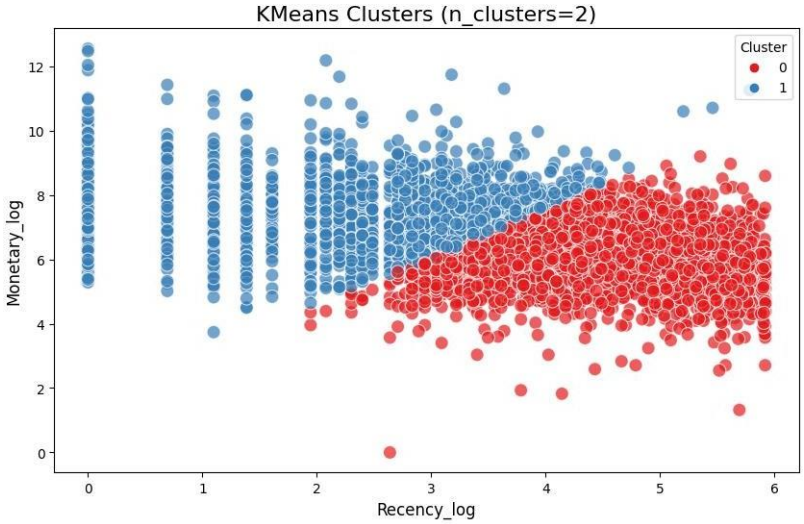
Apply K-Means with silhouette_score | RM |

```
from sklearn.metrics import silhouette_score
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
features_rec_mon=['Recency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon
range_n_clusters = [2,3,4,5,6,7,8,9,10,11,12,13,14,15]
for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict(X)
    centers = clusterer.cluster_centers_

    score = silhouette_score(X, preds)
    print("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))
```

This code is used to find the best number of clusters for customer segmentation based on their Recency and Monetary values (after applying log transformation). It standardizes these features and applies K-Means clustering for different cluster numbers (from 2 to 15). For each cluster count, it calculates the silhouette score, which tells how well the data points fit into their clusters. The higher the silhouette score, the better the clustering. This helps in choosing the most suitable number of customer segments for analysis.

Output:



7.1.6 **K-Means with silhouette_score | FM |**

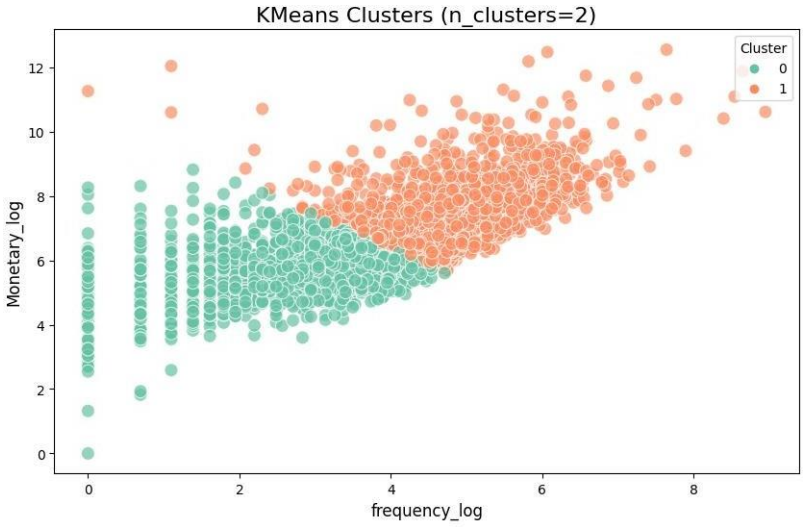
Apply K-Means with silhouette_score | FM |

```
from sklearn.metrics import silhouette_score
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
features_rec_mon=['Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon
range_n_clusters = [2,3,4,5,6,7,8,9,10,11,12,13,14,15]
for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict(X)
    centers = clusterer.cluster_centers_

    score = silhouette_score(X, preds)
    print("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))
```

Based on the silhouette scores for different values of n_clusters, the best clustering result is achieved with 2 clusters, as it has the highest silhouette score of 0.4787. The dataset, which includes the columns ’Recency’, 'Frequency_log' and 'Monetary_log', shows that as the number of clusters increases, the silhouette score decreases, suggesting that more clusters might lead to overfitting or unnecessary complexity. Although there is a slight increase in the score at 7 clusters (0.3439) and 12 clusters (0.3500), these improvements are marginal. Therefore, clustering the data into two clusters provides the best balance between cohesion and separation, making it the most effective number of clusters for this dataset.

Output:



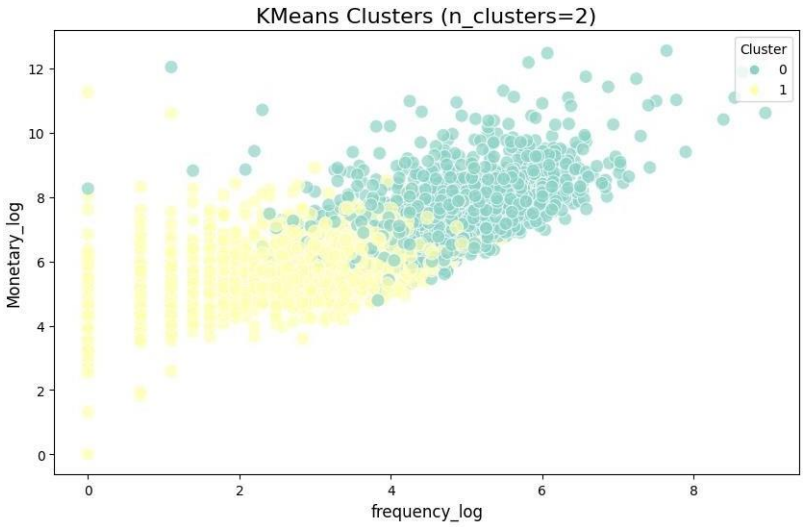
7.1.7 **K-Means with silhouette_score | RFM |**

Based on the silhouette scores for different values of n_clusters, the best clustering result is achieved with 2 clusters, as it has the highest silhouette score of 0.4787. The dataset, which includes the columns ’Recency’, 'Frequency_log' and 'Monetary_log', shows that as the number of clusters increases, the silhouette score decreases, suggesting that more clusters might lead to overfitting or unnecessary complexity. Although there is a slight increase in the score at 7 clusters (0.3439) and 12 clusters (0.3500), these improvements are marginal. Therefore, clustering the data into two clusters provides the best balance between cohesion and separation, making it the most effective number of clusters for this dataset.

Apply K-Means with silhouette_score | RFM |

```
from sklearn.metrics import silhouette_score
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
features_rec_mon=['Recency_log','Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon
range_n_clusters = [3,2,4,5,6,7,8,9,10,11,12,13,14,15]
for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict(X)
    centers = clusterer.cluster_centers_

    score = silhouette_score(X, preds)
    print("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))
```



7.1.8 **K-Means with Elbow method | RM |**

This code applies the K-Means clustering algorithm using the Elbow Method to find the optimal number of clusters based on the features 'Recency_log' and 'Monetary_log'. First, it selects and scales these features using StandardScaler to ensure equal weighting. Then, it runs K-Means clustering for different values of clusters (from 1 to 14) and calculates the inertia (sum of squared distances within clusters) for each. These inertia values are stored

and plotted against the number of clusters. The resulting plot helps identify the "elbow point," which indicates the optimal number of clusters where adding more clusters no longer significantly reduces the inertia.

Apply K-Means with Elbow methos | RM |

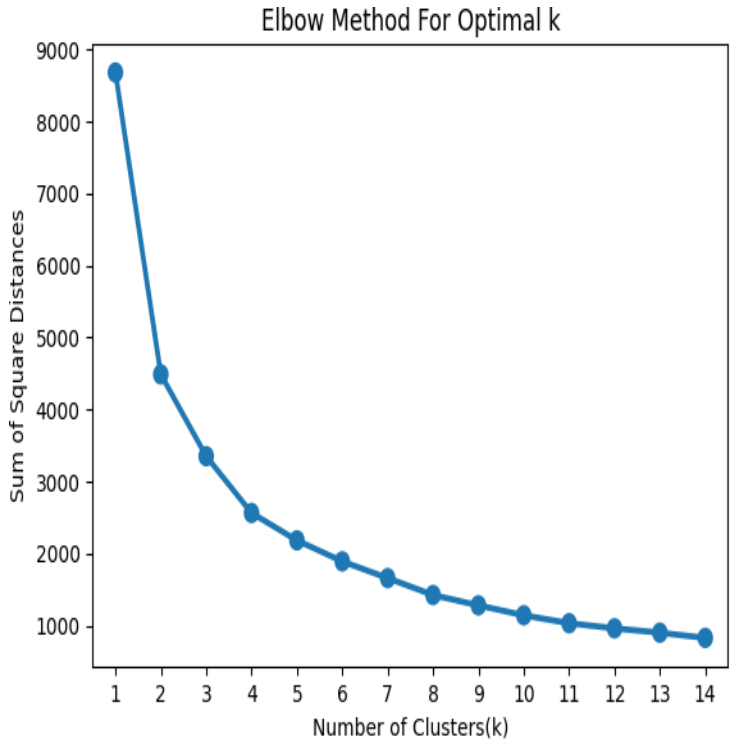
+ Code + Markdown

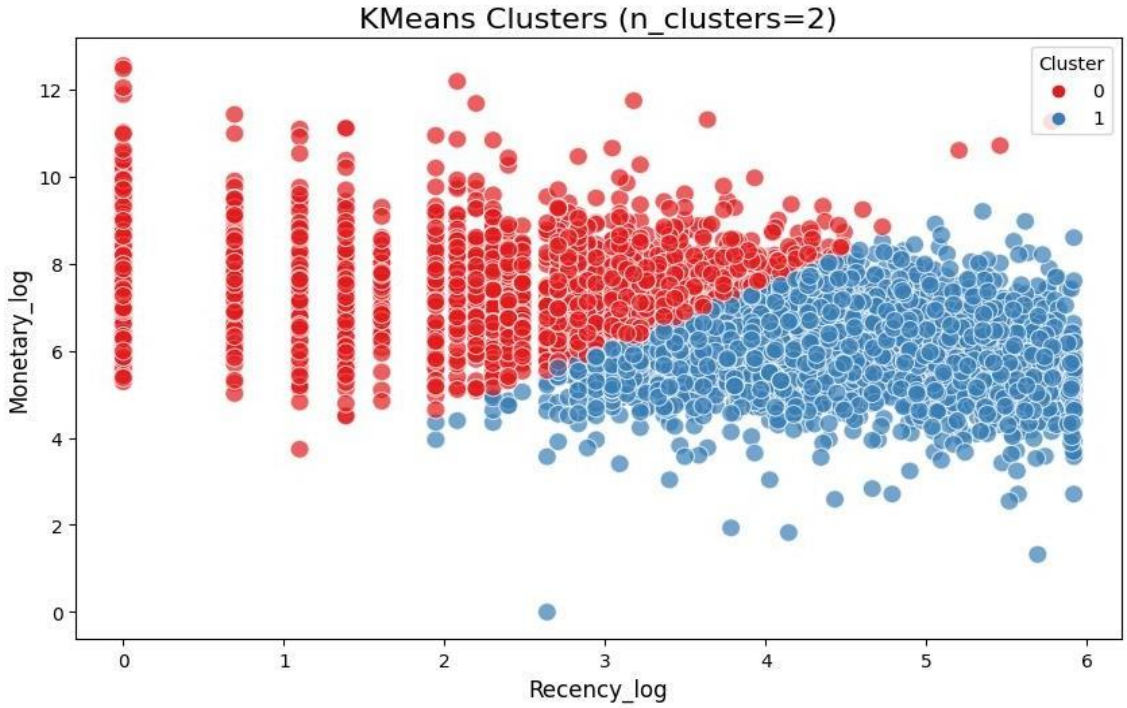
```
features_rec_mon=['Recency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import KMeans

sum_of_sq_dist = {}
for k in range(1,15):
    km = KMeans(n_clusters= k, init= 'k-means++', max_iter= 1000)
    km = km.fit(X)
    sum_of_sq_dist[k] = km.inertia_

#Plot the graph for the sum of square distance values and Number of Clusters
sns.pointplot(x = list(sum_of_sq_dist.keys()), y = list(sum_of_sq_dist.values()))
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```





7.1.9 **K-Means with Elbow method | FM |**

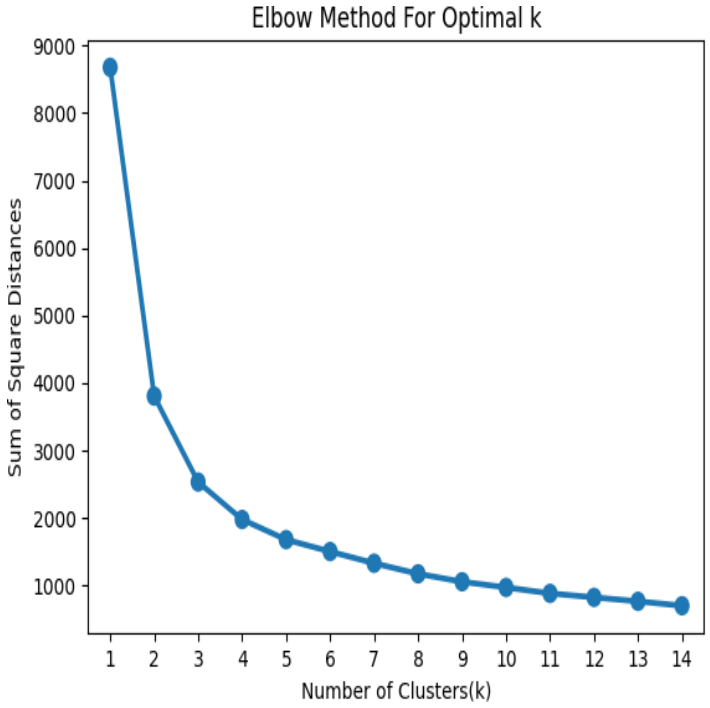
Apply K-Means with Elbow methos | FM |

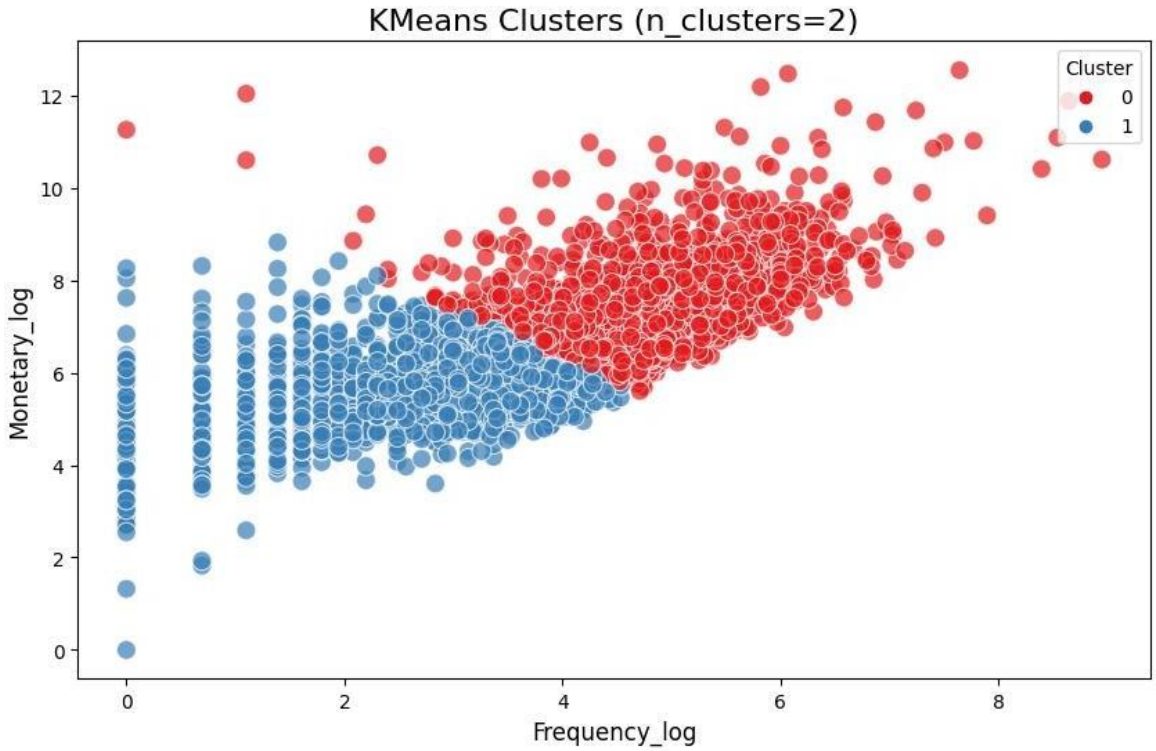
```
features_rec_mon=['Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import KMeans

sum_of_sq_dist = {}
for k in range(1,15):
    km = KMeans(n_clusters= k, init= 'k-means++', max_iter= 1000)
    km = km.fit(X)
    sum_of_sq_dist[k] = km.inertia_

#Plot the graph for the sum of square distance values and Number of Clusters
sns.pointplot(x = list(sum_of_sq_dist.keys()), y = list(sum_of_sq_dist.values()))
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```


Elbow Method For Optimal k
Sum of Square Distances
Number of Clusters(k)
9000
8000
7000
6000
5000
4000
3000
2000
1000
1 2 3 4 5 6 7 8 9 10 11 12 13 14


KMeans Clusters (n_clusters=2)
Cluster
0
1
Monetary_log
Frequency_log
12
10
8
6
4
2
0
0 2 4 6 8

7.1.10 **K-Means with Elbow methods | RFM |**

Apply K-Means with Elbow methos | RFM |

+ Code + Markdown

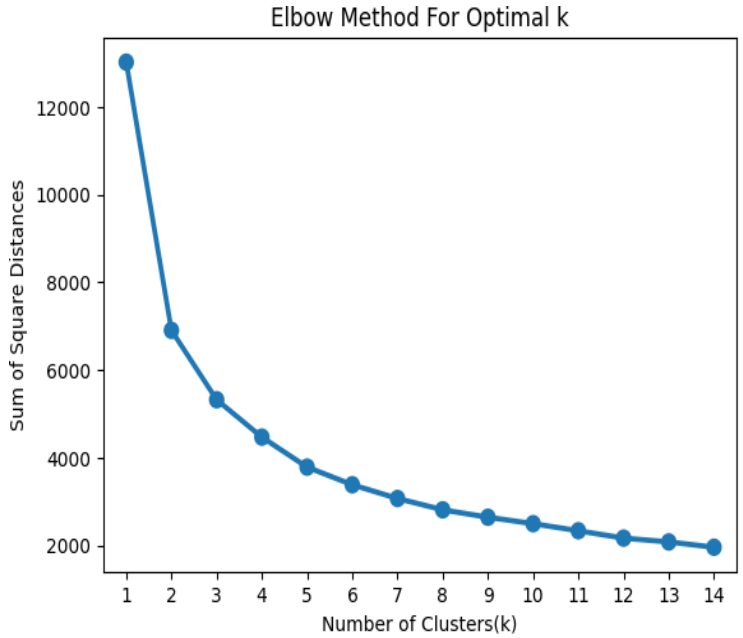
Empty markdown cell, double-click or press enter to edit.

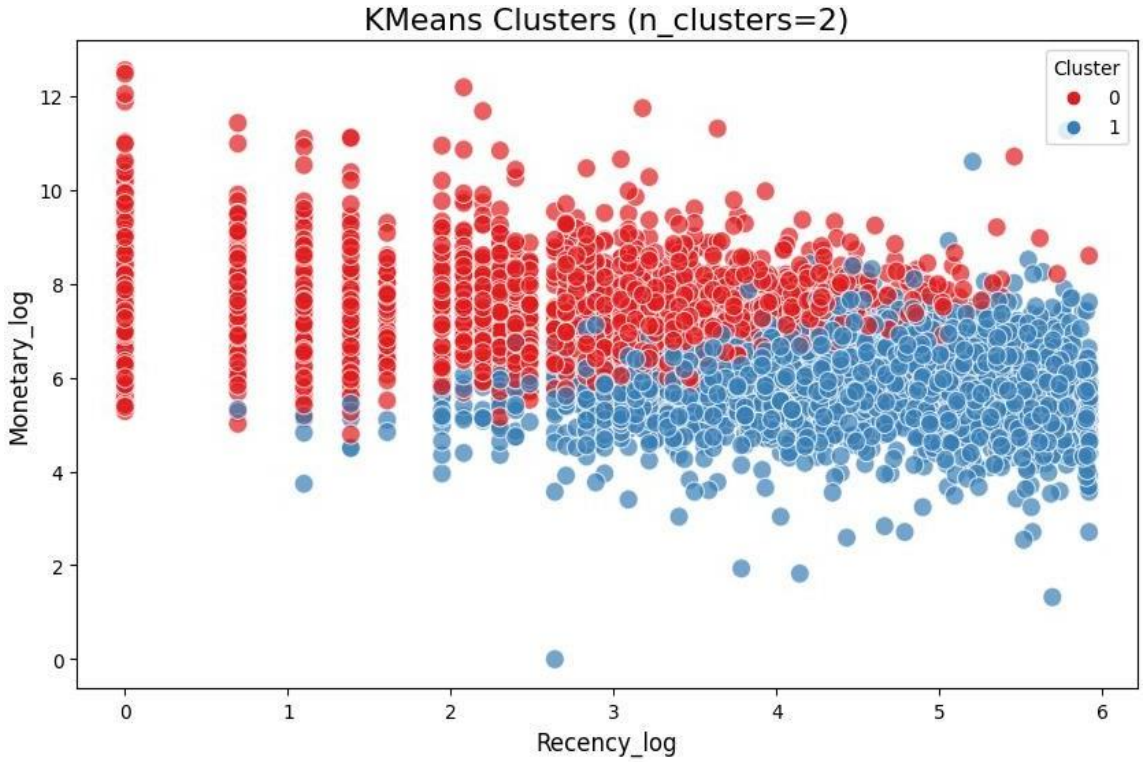
```
features_rec_mon=['Recency_log','Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import KMeans

sum_of_sq_dist = {}
for k in range(1,15):
    km = KMeans(n_clusters= k, init= 'k-means++', max_iter= 1000)
    km = km.fit(X)
    sum_of_sq_dist[k] = km.inertia_

#Plot the graph for the sum of square distance values and Number of Clusters
sns.pointplot(x = list(sum_of_sq_dist.keys()), y = list(sum_of_sq_dist.values()))
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



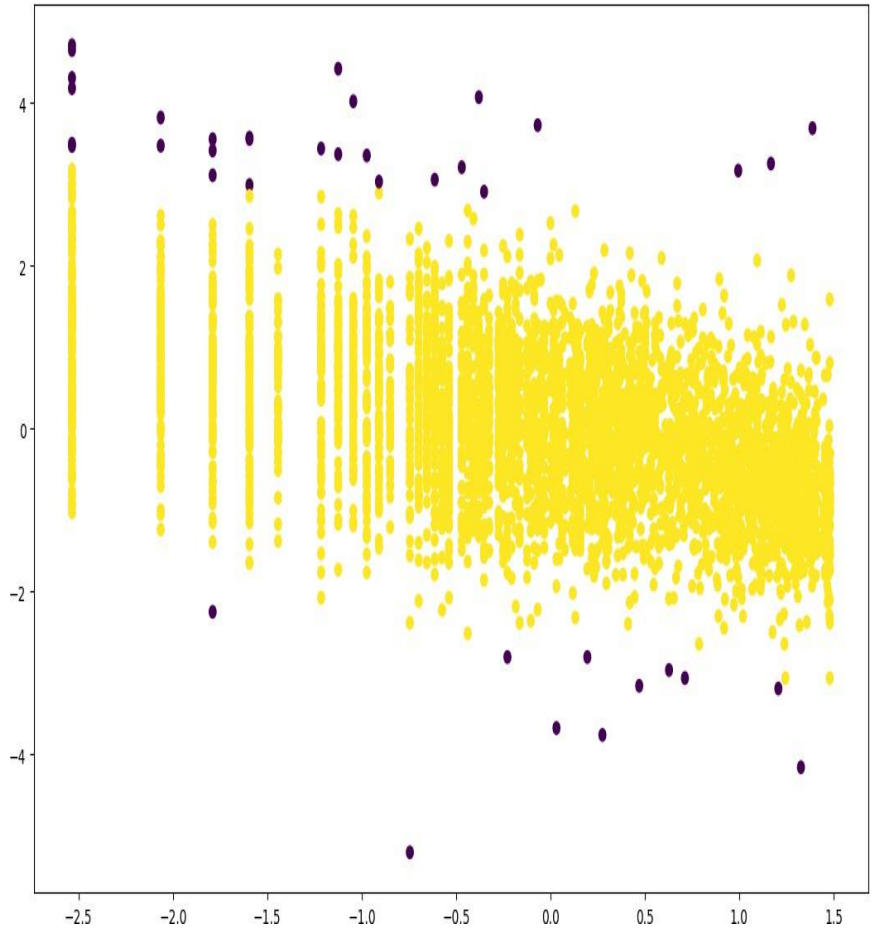


7.1.11 **DBSCAN |RM| Analysis**

Apply DBSCAN | RM |

```
features_rec_mon=['Recency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import DBSCAN
from sklearn import metrics
y_pred = DBSCAN(eps=0.5, min_samples=15).fit_predict(X)
plt.figure(figsize=(13,8))
plt.scatter(X[:,0], X[:,1], c=y_pred)
```



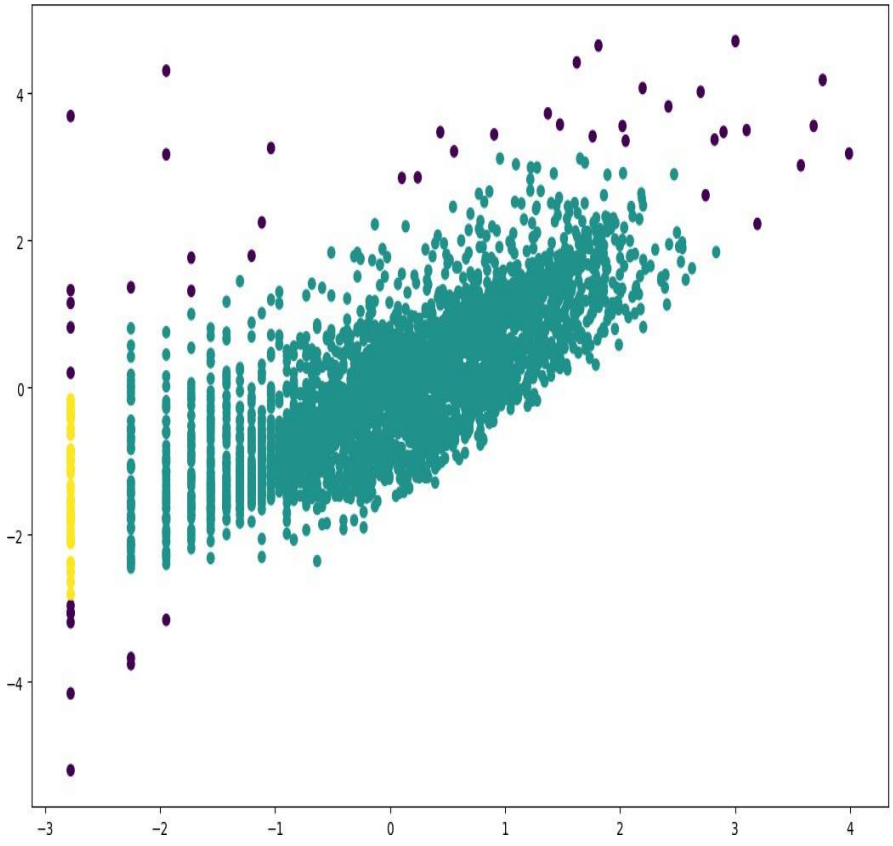
DBSCAN with RM (2 clusters)

7.1.12 **DBSCAN |FM| Analysis**

Apply DBSCAN | FM |

```
features_rec_mon=['Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import DBSCAN
from sklearn import metrics
y_pred = DBSCAN(eps=0.5, min_samples=15).fit_predict(X)
plt.figure(figsize=(13,8))
plt.scatter(X[:,0], X[:,1], c=y_pred)
```

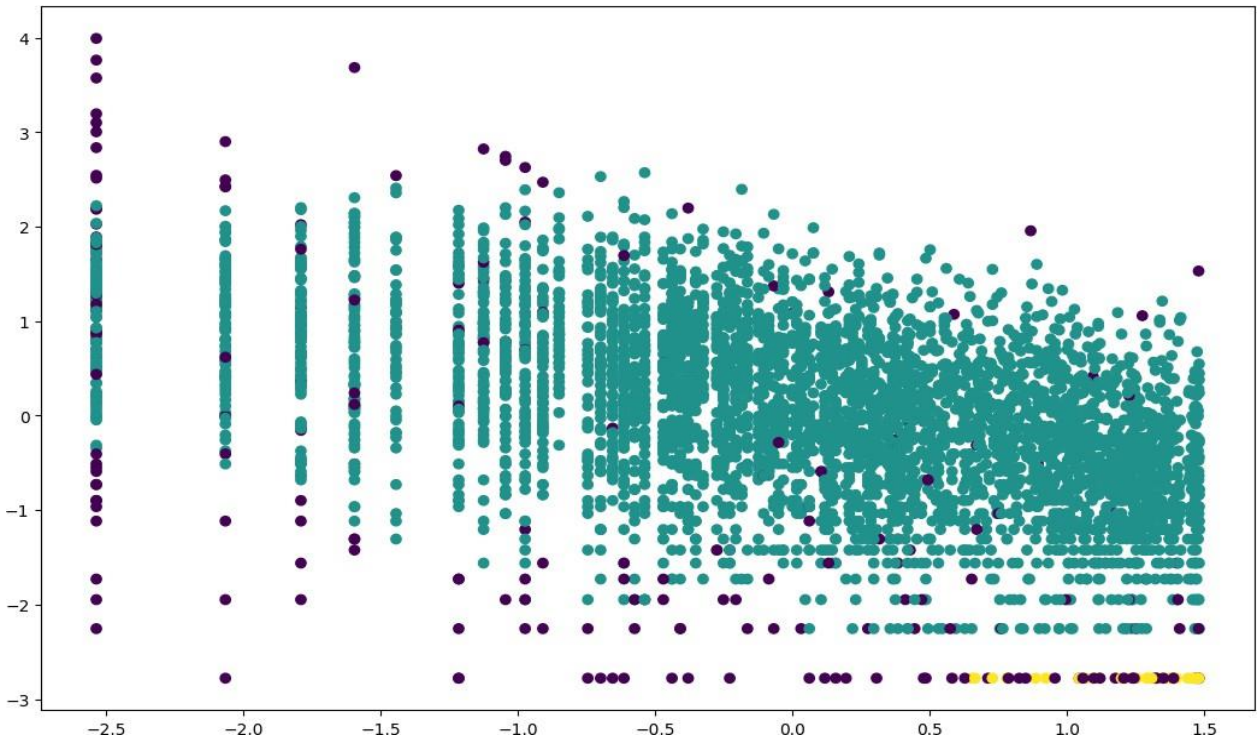


7.1.13 **DBSCAN |RFM| Analysis**

apply DBSCAN | RFM |

```
features_rec_mon=['Recency_log','Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon

from sklearn.cluster import DBSCAN
from sklearn import metrics
y_pred = DBSCAN(eps=0.5, min_samples=15).fit_predict(X)
plt.figure(figsize=(13,8))
plt.scatter(X[:,0], X[:,1], c=y_pred)
```



7.1.14 Hierarchical clustering | RFM |

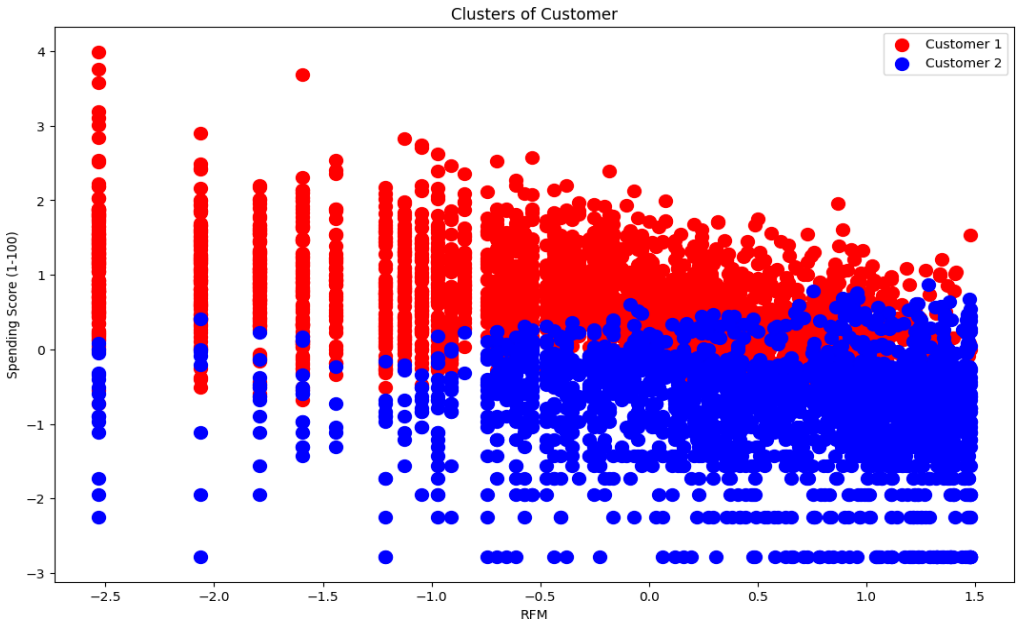
Apply Hierarchical clustering | RFM | |

Dendogram to find the optimal number of clusters

```
features_rec_mon=['Recency_log','Frequency_log','Monetary_log']
X_features_rec_mon=rfm_df[features_rec_mon].values
scaler_rec_mon=preprocessing.StandardScaler()
X_rec_mon=scaler_rec_mon.fit_transform(X_features_rec_mon)
X=X_rec_mon



# Using the dendogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
plt.figure(figsize=(13,8))
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean Distances')
plt.show() # find largest vertical distance we can make without crossing any other horizontal line
```



7.1.5 **Model Evaluation and Insights**

After clustering the customers using the K-Means algorithm, the effectiveness of the segmentation is evaluated using the **Silhouette Score**. This metric helps measure how similar an object is to its own cluster compared to other clusters. A higher silhouette score indicates that the data points are well-clustered, meaning customers within a group share common behavior patterns and are distinctly different from those in other groups. This evaluation step is crucial because it ensures the quality and reliability of the clusters formed, allowing the business to trust the segmentation results for further analysis and decision-making.

Once validated, the clusters are examined to understand customer behavior. The analysis often reveals different customer segments such as **high-value customers** (frequent and high spenders), **occasional buyers** (moderate activity and spending), and **inactive users** (rare or one-time purchasers). By understanding these distinct groups, businesses can implement **targeted marketing strategies** — for example, offering exclusive deals to high-value customers to encourage loyalty, sending reminders or small discounts to occasional buyers, and re-engagement campaigns for inactive users. This personalized approach boosts customer satisfaction, enhances retention, and drives revenue growth.

Overall, this structured, data-driven approach to customer segmentation empowers companies to move beyond generic marketing. By tailoring their communication and offerings to the unique needs and behaviors of each segment, businesses can **optimize their marketing efforts, strengthen customer relationships**, and ultimately improve **sales performance and**

operational efficiency.

SL No.	Model_Name	Data	Optimal_Number_of_cluster
1	K-Means with silhouette_score	RM	2
2	K-Means with Elbow methos	RM	2
4	K-Means with silhouette_score	FM	2
5	K-Means with Elbow methos	FM	2
7	K-Means with silhouette_score	RFM	2
8	K-Means with Elbow method	RFM	2
3	DBSCAN	RM	2
6	DBSCAN	FM	3
10	DBSCAN	RFM	3
9	Hierarchical clustering	RFM	2

Table 1:Analysis of Different Clustering Techniques

CHAPTER 7

PROJECT TESTING

7.1 System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.2 Types of Tests

7.2.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.2.3 Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

7.2.4 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner

workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

7.2.5 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

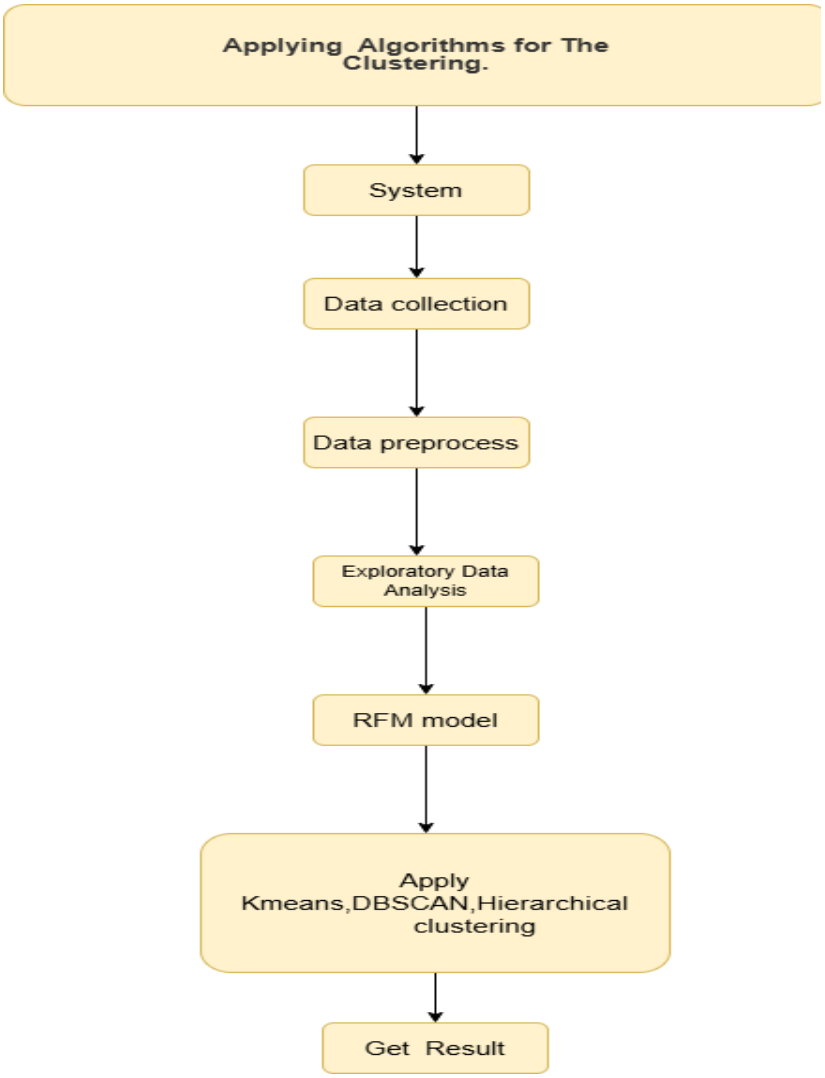
Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page

CHAPTER -8

PROJECTOPERATION

8.1 CONTROL FLOW OF EXECUTION



A	B	C	D	E	F	G	H
voiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HE	6	01-12-2010 08:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANT	6	01-12-2010 08:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEAR	8	01-12-2010 08:26	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLA	6	01-12-2010 08:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTI	6	01-12-2010 08:26	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NE	2	01-12-2010 08:26	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTE	6	01-12-2010 08:26	4.25	17850	United Kingdom
536366	22633	HAND WARMER UN	6	01-12-2010 08:28	1.85	17850	United Kingdom
536366	22632	HAND WARMER RED	6	01-12-2010 08:28	1.85	17850	United Kingdom
536367	84879	ASSORTED COLOUR	32	01-12-2010 08:34	1.69	13047	United Kingdom
536367	22745	POPPY'S PLAYHOUSE	6	01-12-2010 08:34	2.1	13047	United Kingdom
536367	22748	POPPY'S PLAYHOUSE	6	01-12-2010 08:34	2.1	13047	United Kingdom
536367	22749	FELTCRAFT PRINCES	8	01-12-2010 08:34	3.75	13047	United Kingdom
536367	22310	IVORY KNITTED MUG	6	01-12-2010 08:34	1.65	13047	United Kingdom
536367	84969	BOX OF 6 ASSORTED	6	01-12-2010 08:34	4.25	13047	United Kingdom
536367	22623	BOX OF VINTAGE JIG	3	01-12-2010 08:34	4.95	13047	United Kingdom
536367	22622	BOX OF VINTAGE AL	2	01-12-2010 08:34	9.95	13047	United Kingdom
536367	21754	HOME BUILDING BL	3	01-12-2010 08:34	5.95	13047	United Kingdom
536367	21755	LOVE BUILDING BLO	3	01-12-2010 08:34	5.95	13047	United Kingdom
536367	21777	RECIPE BOX WITH M	4	01-12-2010 08:34	7.95	13047	United Kingdom
536367	48187	DOORMAT NEW EN	4	01-12-2010 08:34	7.95	13047	United Kingdom
536368	22960	JAM MAKING SET W	6	01-12-2010 08:34	4.25	13047	United Kingdom
536368	22913	RED COAT RACK PAF	3	01-12-2010 08:34	4.95	13047	United Kingdom

Fig 8.1.3 Datasets

8.2 Comparison with Baseline Models