POLITECNICO DI TORINO

MASTER's Degree in DATA SCIENCE AND ENGINEERING

MASTER's Degree Thesis

Dyna ML - A Machine Learning Approach to Sales Forecasting and Product Recommendations

Supervisor

Prof. Giuseppe RIZZO

Candidate

Sevde BEKDIK

Company Supervisor

Massimiliano COLOMBO

April 2025

Summary

This thesis presents a comprehensive machine learning framework for forecasting product sales and generating product recommendations in a retail context using the ContosoRetailDW dataset. The investigation begins with an extensive exploration of the data stored in a Microsoft SQL Server environment. Relevant numeric and categorical attributes are merged and refined, bringing important features such as product identifiers, location information, and temporal details. To capture the underlying complexity of real-world sales transactions, detailed feature engineering steps became essential, such as target encoding for high-cardinality variables. This preprocessing phase ensures that the dataset remains both indicative of retail trends and effectively structured for further steps. The sales prediction section includes gathering a range of regression models, such as Ridge Regression, Random Forest, Gradient Boosting, Support Vector Regression, and XGBoost. Although initial experiments suggest that Ridge Regression performs well as a baseline, after a thorough hyperparameter tuning phase with cross-validation, the most reliable predictor, XGBoost, achieves the lowest root mean squared error on unseen data.

The two primary methods that are examined in the product recommendation section are content-based filtering and collaborative filtering. The Surprise library's algorithms are implemented to provide collaborative filtering, which considers past user-item interactions. The best performing collaborative model is determined by comparing rating predictions across multiple approaches, ultimately selecting a KNN-based algorithm that provides the highest accuracy in user preference prediction. Contrarily, content-based filtering relies on textual product descriptions encoded using TF-IDF vectorization and categorized using KMeans clustering. This allows suggestions to remain in line with textual similarity even when comprehensive user feedback is absent. The final result is integrated into a production-ready pipeline. FastAPI endpoints are designed to process prediction and recommendation requests, and a Gradio interface provides an accessible web-based front-end.

This comprehensive solution successfully brings together data ingestion, feature engineering, model training, hyperparameter tuning, and practical deployment, demonstrating the transformative potential of machine learning in sales forecasting and product recommendation within the retail sector.

Acknowledgements

This journey has been an incredible learning experience for me, filled with challenges, growth, and invaluable lessons. First and foremost, I extend my deepest gratitude to my mother, whose unwavering support and endless encouragement have been my greatest source of motivation. Her presence has been my strongest pillar of strength, and no words can ever fully express my appreciation. I am also grateful to my beloved family, whose constant support I have felt every step of the way.

To my dearest husband, who has been by my side through every challenge, lifting me up when I stumbled and inspiring me to continue pushing forward—without you, this journey would have been immeasurably more difficult. Facing hardships together, shoulder to shoulder, has been a priceless experience, and I am forever thankful for your unconditional love and support.

To my dear friends, you have made this journey not only enriching but also enjoyable. Your presence has turned even the toughest moments into shared victories, and I am grateful for the privilege of working and growing alongside you. The bond we built through hard work and perseverance will always be a source of delight for me.

Additionally, I would like to sincerely thank Professor Giuseppe Rizzo, my supervisor, for providing valuable guidance throughout this project. His expertise and insights have been instrumental in shaping my work.

I would also like to express my gratitude to my colleague Massimiliano Colombo, whose generous sharing of knowledge and experience has been an inspiration throughout this process. Acting as both a mentor and a friend, he has helped me navigate every challenge with his wisdom and patience. I also extend my thanks to all my colleagues, whose collaboration and encouragement have made this journey all the more rewarding.

This work is the result of collective efforts, encouragement, and continuous support from those around me, and for that, I will always be deeply grateful.

Sevde Bekdik

Table of Contents

List of Figures

List of Tables

Acronyms

ALS	Alternating Least Squares.
API	Application Programming Interface.
CF	Collaborative Filtering.
CV	Cross-Validation.
EDA	Exploratory Data Analysis.
ETL	Extract, Transform, Load.
KNN	K-Nearest Neighbors.
MAE	Mean Absolute Error.
ML	Machine Learning.
NMF	Non-negative Matrix Factorization.
RMSE	Root Mean Squared Error.
SQL	Structured Query Language.
SVR	Support Vector Regression.
SVD	Singular Value Decomposition.
TF-IDF	Term Frequency–Inverse Document Frequency.
UI	User Interface.
XGBoost	eXtreme Gradient Boosting.

Chapter 1

Introduction

This thesis presents Dyna ML - A Machine Learning Approach to Sales Forecasting and Product Recommendations Prediction, a company-driven research initiative aimed at addressing critical challenges encountered by retailers. Accurate product sales forecasts and personalized product suggestions have become crucial elements for attaining long-term business success in the dynamic and competitive retail industry.

Sophisticated data-driven strategies are necessary to sustain profitability and improve customer satisfaction in an environment of rapid advances in technology, evolving consumer behaviors, and changing market trends. By analyzing historical sales data, customer demographics, promotional activity, and external influencing factors, sales prediction estimates future sales quantity. Retailers could improve profitability by proactively planning promotional activities and optimizing inventory management with the help of accurate sales forecasting. However traditional forecasting techniques, which frequently rely on simplistic linear assumptions, find it difficult to identify the complex patterns and nonlinear relationships present in actual sales data, especially when there are a large number of categorical and numerical variables.

Product recommendation systems have become essential in e-commerce and retail, enhancing user experience by proposing relevant items based on prior customer interactions (collaborative filtering) or specific item characteristics (content-based filtering). Efficient recommendation systems can markedly enhance customer satisfaction and loyalty by precisely aligning product offerings with consumer preferences, hence improving purchase likelihood and promoting long-term client retention. This research clearly addresses the interrelated challenges of accurately forecasting sales quantity and delivering relevant, customized product suggestions.

The primary objectives of this project are twofold: first, to employ advanced machine learning techniques, including Random Forest, Gradient Boosting, and XGBoost, to enhance sales prediction accuracy beyond traditional regression models; second, to implement and assess recommendation systems via collaborative filtering

methods, utilizing historical customer interactions and content-based filtering approaches that incorporate detailed product metadata. This study aims to provide a comprehensive and reliable solution that can increase forecast accuracy and produce substantial commercial value through useful product suggestions by combining these approaches into a single analytical framework.

The significance of this research lies in its innovative integration of diverse machine learning approaches specifically tailored to the complexities of retail data. Through systematic evaluation and benchmarking of various predictive and recommendation algorithms, this thesis aims to provide both academic contributions and practical applications, enabling retailers to adopt effective data-driven decision-making strategies.

Chapter 2

Literature Review

Sales Prediction

Sales prediction is an an essential procedure of analysis that enables retailers to foresee consumer demand and manage inventory effectively, guaranteeing optimized allocation of resources and maximized profitability. Because of their accessibility and interpretability, traditional forecasting methods such as exponential smoothing, moving averages, and linear regression have been frequently adopted. However, given the existence of high-dimensional categorical variables along with additional influencing factors like promotions, seasonality, and economic conditions, these methods frequently fall short when dealing with the complexity and non-linearity present in modern-day retail data.

The latest advances in ML are resulting in sophisticated predictive models that significantly improve forecasting precision. For the reason they can detect complex, non-linear correlations and interactions between variables, tree-based ensemble approaches such as Random Forests and Gradient Boosting algorithms have achieved exceptional performance in sales forecasting [27][29]. XGBoost, a variant of Gradient Boosting, further boosts prediction accuracy and computational efficiency by incorporating regularization terms and improved handling of missing values.

The superiority of ML approaches over traditional statistical methods in sales forecasting has been demonstrated by numerous research. In order to estimate retail sales, for instance, Pavlyshenko (2018) [39] shows that ensemble approaches perform better than standard statistical models because of their greater generalization capabilities and flexibility in modeling nonlinear interactions. Similar to this, a study by Ma and Fildes (2021) [40] highlights the significance of advanced feature engineering, which includes efficiently encoding categorical variables and greatly enhances model performance while lowering forecast errors.

Product Recommendation Systems

Technologies and methods known as recommender systems make recommendations for products that consumers would find useful [55]. In both online and offline retail settings, they have greatly increased client engagement and sales and have proven essential in providing individualized customer experiences. Two main approaches have been applied in this project: content-based filtering and collaborative filtering.

In order to suggest products based on the similarity of consumers or products, Collaborative Filtering (CF) utilizes user-item interactions, including ratings and past purchases. Basic collaborative filtering algorithms were presented by Sarwar et al. (2001), who showed that they were successful in providing customized recommendations solely based on past interactions [57]. Recent developments have expanded these conventional techniques using complex algorithms such as nearest neighbor methods (KNN variations) and matrix factorization (SVD, NMF), which achieve superior performance thanks to their capacity to learn latent user-item interactions and handle sparse data efficiently.

In contrast, Content-Based Filtering utilizes product attributes, typically derived from product descriptions and metadata, to recommend products similar to those the user has already liked or interacted with. TF-IDF vectorization and clustering methods, such as KMeans, are widely applied in content-based systems to group products with similar features and textual characteristics [56][47]. This method is especially effective for recommending new or less popular products lacking sufficient interaction history (cold-start problem).

Hybrid recommendation systems, which integrate collaborative and content-based methods, are frequently proposed as the best solutions, as they utilize the advantages of both approaches while mitigating their respective limitations [58]. Nevertheless, hybrid techniques add complexity and can present computational difficulties, requiring thorough planning of the methodology.

Contributions and Gaps

While extensive research has explored sales prediction and recommendation systems separately, integrated studies that address both challenges simultaneously within a unified analytical framework remain limited. Existing studies often neglect the complexities introduced by real-world retail scenarios, such as high cardinality categorical variables, missing data, and highly imbalanced consumer-product interactions. Furthermore, practical deployment considerations such as computational efficiency, real-time prediction requirements, and interpretability are often inadequately ad-

dressed in the current literature.

This study bridges this gap by presenting a comprehensive analytical framework that integrates advanced ML techniques for sales prediction with robust recommendation methodologies. By leveraging pipeline structures that include sophisticated encoding strategies (target encoding, one-hot encoding), robust cross-validation, and hyperparameter optimization, this research aims to enhance predictive performance. Furthermore, the integration of collaborative and content-based filtering methodologies within a single, practical application further distinguishes this study from the existing literature.

Through this integrated approach, this thesis seeks to provide meaningful contributions to academic literature and industry practice, allowing retail organizations to benefit from more accurate forecasts and actionable personalized recommendations, ultimately supporting informed and strategic decision-making.

Chapter 3

Methodology

3.1 Data Description and Acquisition

This research utilizes the **ContosoRetailDW dataset**, a comprehensive and publicly available retail dataset provided by Microsoft, specifically designed to demonstrate business intelligence and analytical tasks in the retail industry [1] [2].

Figure 3.1: Contoso DB Dataset

The dataset includes rich and diverse data sources, which makes it particularly suited for advanced analytical projects such as sales forecasting and personalized product recommendation.

The dataset comprises several fact and dimension tables that collectively provide a comprehensive analytical foundation. Our analysis focused mainly on two fact tables, FactSales and FactOnlineSales, which contain detailed transaction-level data, including sales quantities, pricing, and promotional details, and many dimension tables such as DimCustomer, DimProduct, DimStore, DimPromotion, DimGeography, and DimProductSubcategory offer contextual and descriptive attributes that enrich feature engineering and enhance predictive capability.

To optimize the accuracy and reliability of analyses, two specialized views, V_Sales and V_ProductRecommendation, were meticulously developed through exploratory data analysis and sophisticated feature engineering processes. The

V_Sales view integrates essential columns from multiple tables, including FactSales, DimStore, DimGeography, DimProduct, and DimPromotion, combining strategically selected variables identified as relevant predictors of sales volumes.

Similarly, the V_ProductRecommendation view aggregates meaningful features from FactSales, DimCustomer, and DimProduct, effectively addressing the needs of recommendation systems by capturing detailed interactions and product attributes.

Table 3.1: Tables and Views Used in Contoso DB for Dyna ML

Fact Tables	**Dimension Tables**	**Custom Views**
FactSales	DimStore	V_Sales
FactOnlineSales	DimProduct	V_ProductRecommendation
	DimPromotion	
	DimGeography	
	DimProductSubcategory	
	DimDate	
	DimProductCategory	
	DimCustomer	

While the complete ContosoRetailDW dataset encompasses around 10 million transaction records, this project strategically employs a randomly selected subset of 10,000 records. This decision was made to ensure computational feasibility and efficiency without compromising the diversity and representativeness necessary for rigorous model development and validation of the models. A dedicated ETL (Extract, Transform, Load) pipeline was implemented to process these data. Irrelevant columns were excluded, dates were transformed into actionable temporal characteristics (year, month, weekday), categorical variables were appropriately encoded (one-hot encoding for low cardinality features and target encoding for high cardinality features), and numeric data were standardized using robust scaling techniques. These comprehensive preprocessing steps substantially elevated the performance of analytical models, effectively addressing the intricate and dynamic nature of retail data.

3.2 Machine Learning Overview and Theoretical Background

The term machine learning (ML) describes computational techniques and algorithms that enable computers to learn from data patterns on their own and make predictions or judgments without direct guidance [25]. Supervised, unsupervised, and reinforcement learning are the three main categories into which ML techniques fall [26]. In order to effectively forecast target variables, this research primarily use supervised learning, where algorithms learn from labeled data.

Regression and classification tasks are two further subcategories of supervised learning, where regression involves continuous target variables and classification involves categorical target variables. Regression techniques were used in this project to forecast sales quantities and other continuous numerical outcomes. The supervised regression models that were chosen are SVR, XGBoost, Random Forest, Gradient Boosting, and Ridge Regression.

Ridge regression is a regularized linear regression approach that introduces an L2 penalty to reduce model complexity, thus mitigating multicollinearity and improving predictive stability [16][26]. Random Forest, an ensemble learning method, constructs multiple decision trees during training and produces the average predictions, effectively reducing variance and overfitting issues [37][15][26]. Gradient Boosting iteratively builds weak learners, usually decision trees, optimizing predictive accuracy by minimizing a differentiable loss function; this method excels in capturing complex non-linear relationships within the data [35]. XGBoost extends the gradient boosting framework with additional regularization terms and advanced optimization techniques, providing superior computational efficiency, and improved prediction accuracy, which is particularly beneficial for large-scale datasets [36][12].

Two main approaches were investigated for recommendation systems: content-based filtering and collaborative filtering. Using latent connections discovered by algorithms like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF), collaborative filtering approaches make product recommendations based on user-item interaction patterns [45]. Content-based filtering, on the contrary, uses product attributes and descriptions, typically represented using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization and clustering techniques, such as KMeans, to recommend similar products based on their inherent features [56] [46].

3.3 Technologies and Tools

The implementation and deployment phases of this research rely heavily on a meticulously selected set of technologies, libraries, and platforms that form a robust analytical ecosystem. Each component was strategically chosen based on its capability to fulfill specific requirements in data handling, model experimentation, and effective dissemination of results. This section provides a comprehensive overview of all the major technologies and libraries utilized, the rationale behind their selection, and their respective roles within the project.

Database and Data Storage

Microsoft SQL Server forms the foundational data layer of this project. It hosts the ContosoRetailDW dataset, providing efficient storage and quick retrieval through customized SQL views (*V_Sales*, *V_ProductRecommendation*). Its robust querying capabilities facilitated rapid data exploration and sophisticated feature engineering, which are crucial to preparing data for subsequent stages of ML modeling [2][1][5]. To interact programmatically with this database in Python, **SQLAlchemy** was employed as the core SQL toolkit, ensuring reliable connections, intuitive abstractions for queries, and streamlined operations for both small- and large-scale data manipulations. This maintainable database interaction served as a strong foundation for iterative experimentation.

Figure 3.2: Technologies and Tools Utilized for the Data Management

Python Ecosystem and Development Environments

Python Serves as the primary programming language for data pre-processing, model building and deployment tasks. Python's expansive ecosystem of packages and its easy readability accelerated the overall implementation. Additionally, it offered an extensive standard library for core functionalities such as system interactions and math operations, reducing development overhead [24].

Jupyter Notebook and Visual Studio Code Both environments played a pivotal role in supporting different stages of the data science workflow [17][10]. Jupyter Notebook was particularly suited for exploratory data analysis (EDA), interactive visualizations, and iterative experimentation, enabling quick data inspection and real-time result interpretation. In parallel, Visual Studio Code offered an integrated and extensible environment for tasks such as advanced debugging, Git version control, project organization, and end-to-end model pipeline development. This combination ensured both flexibility in experimentation and rigor in code maintenance.

Figure 3.3: Technologies and Tools Utilized in the Python Development

Data Manipulation and Numerical Computation

Pandas Used as the primary tool for loading, cleaning, transforming, and preparing datasets [4]. Its DataFrame structure, with methods for grouping, merging, reshaping, and statistical summaries, significantly simplified the process of handling large tabular data. Additionally, Pandas integrates seamlessly with libraries such as NumPy and scikit-learn, thus serving as the central data structure throughout the modeling process.

NumPy Serves as the foundational library for numerical operations in Python [3]. Many other libraries, including Pandas and scikit-learn, build upon NumPy arrays for performance and memory-efficiency reasons. Its vectorized operations, linear algebra routines, and broadcasting rules made it vital for large-scale rapid numerical computations, especially when training models or performing complex transformations on multidimensional arrays.

Figure 3.4: Technologies and Tools Utilized for Data Manipulation

Machine Learning and Model Development

scikit-learn (sklearn) The core toolkit for general purpose ML tasks, providing a uniform API for regression, classification, and clustering. It was widely used to create end-to-end pipelines, perform hyperparameter tuning via *GridSearchCV*, and evaluate model performance using cross-validation. scikit-learn's feature engineering utilities, such as *OneHotEncoder*, *StandardScaler*, and custom *TransformerMixin* classes, streamlined data processing and integration into reproducible, robust experimentation pipelines.

XGBoost A high-performance library for gradient boosting that often yields competitive results in structured data tasks [36]. Its parallelized tree-building, effective handling of missing data, and robust parameter tuning options made it particularly attractive for capturing non-linear relationships. Using advanced boosting strategies, XGBoost typically offered strong predictive capabilities in tabular data, which proved valuable for our sales quantity predictions.

joblib Enables efficient serialization and deserialization of Python objects, including entire scikit-learn pipelines. This was particularly crucial when it came to saving trained models (along with their pre-processing transformations) so they could be reloaded for inference without retraining from scratch, thereby accelerating deployment workflows [11].

Figure 3.5: Technologies and Tools Utilized in Model Development

Recommendation System Approaches

Surprise A specialized Python library for building and analyzing recommender systems. It focuses on collaborative filtering (e.g., user–item rating predictions) and thus provides an intuitive framework for implementing techniques like SVD, KNNBasic, or SlopeOne with minimal overhead [9]. Using Surprise simplified the process of exploring a variety of collaborative filtering methods under a common interface. Its built-in tools for cross-validation and performance reporting (e.g., RMSE and MAE) on sparse rating matrices also helped efficiently compare algorithm variants.

Figure 3.6: Surprise: A Python scikit for recommender systems

Content-Based Filtering Built primarily through scikit-learn but in conjunction with *TF-IDF* vectorization and *KMeans* clustering. Product descriptions and related metadata from the database were transformed into numerical feature vectors, which allowed the system to recommend items based on text similarity. This method

proved particularly valuable when user–product interactions were limited or needed supplementary item-centric analysis.

Data Visualization

matplotlib Serves as the foundational Python library for creating static, animated, and interactive visualizations. Used extensively for fundamental charting operations, such as scatter plots to examine the relationships between features, bar plots to compare model performance, and histograms to illustrate data distributions.

seaborn Built on top of matplotlib, Seaborn offers advanced statistical plotting aesthetics and concise syntax. During EDA, it helped quickly reveal data patterns, correlations, and potential outliers through more detailed and visually appealing plots, such as pairplots, heat maps, and regression plots [18].

Figure 3.7: Technologies and Tools Utilized for Data Visualization

API Development and Deployment

FastAPI Chosen for its performance, modern design, and straightforward integration with asynchronous Python features [6]. FastAPI exposed our trained models (both predictive and recommender) via RESTful endpoints, enabling real-time predictions and recommendations. This framework provided automatic data validation, documentation, and error handling, thus reducing the likelihood of runtime failures due to malformed requests.

Pydantic Works in conjunction with FastAPI to validate incoming requests against well-defined schemas. It enforced strict typing, ensuring that only well-structured data reached the core inference logic. Using Pydantic models, errors could be caught at the request layer and quickly relayed to end-users or client applications [23].

requests A convenient and human-readable HTTP library that can interact with our FastAPI endpoints. It simplified testing various scenarios programmatically and allowed other parts of the code (e.g., Gradio applications) to make prediction or recommendation requests internally without complicated networking logic [22].

Figure 3.8: Technologies and Tools Utilized for API Development

Interactive Interfaces and Testing Tools

Gradio A user-friendly toolkit to quickly create interactive web interfaces. By attaching our Python functions for sales forecasting or product recommendations, to Gradio UI components, non-technical stakeholders could directly input features or specify parameters in a simple browser-based interface [8]. This accelerated feedback loops and facilitated demonstrations of model behavior.

Postman Used extensively to test, debug, and document FastAPI endpoints. By mimicking real HTTP requests (GET, POST, etc.), Postman ensured robust verification of all API functionalities and simplified rapid iteration during development.

Figure 3.9: Technologies and Tools Utilized for Interface and Testing

Summary of Integration

Collectively, these technologies, libraries, and tools form a comprehensive ecosystem for modern data science workflows. Python's scientific stack (Pandas, NumPy, and scikit-learn) anchors data transformations and traditional predictive modeling. SQL Server and SQLAlchemy provide high-performance data ingestion capabilities. Surprise and TF-IDF-based methods address collaborative and content-based recommendation scenarios, respectively, while XGBoost extends predictive performance on structured data. Finally, model exposure to end-users and stakeholders is enabled through FastAPI for production-grade endpoints, Gradio for interactive exploration, and Postman for thorough testing and monitoring. By employing this carefully orchestrated suite of technologies, the project ensures robustness, scalability, and ease of future enhancements.

3.4 Model Evaluation Metrics

The Root Mean Squared Error (RMSE) is an extensively utilized statistic for evaluating the prediction accuracy of a model when the outcome is a continuous numerical

variable. The value provides an indication of the average amount of the prediction errors between the estimated values of the model and the corresponding true (observed) values [29]. Formally, if y_i represents the true value and $\hat{y}_i$ represents the predicted value for the i-th sample, then the RMSE over N samples is given by

$$\text{RMSE} \;=\; \sqrt{\frac{1}{N}\sum_{i=1}^{N}\big(\hat{y}_i \;-\; y_i\big)^2}. \tag{3.1}$$

The differences $\hat{y}_i - y_i$ in the summation represent the individual prediction errors.These errors are first squared and then averaged, indicating that that larger errors incur a most significantly higher penalty. In practical or business applications, taking the square root makes the measure easy to understand because it returns it to the same unit as the target variable.

Due to the fact that it squares each error before averaging, RMSE penalizes big individual variations more severely than smaller ones, which makes it very useful. RMSE is a suitable option in situations where lowering the frequency of significant outliers or mispredictions is the aim. The Mean Absolute Error (MAE), on the other hand, is one alternative statistic that does not heavily penalize particular significant deviations. As a result, in time-series forecasting, rating prediction tasks, and regression issues, RMSE frequently acts as a common comparison benchmark.

RMSE also provides an intuitive understanding of where those predictions might be, on average, when the target variable's size is important to stakeholders. For example, when a product sales forecast model's RMSE is 5.0, it means that estimates usually vary by five units sold from the actual quantity sold. Depending on the necessities of the business, this threshold may or may not be acceptable.

RMSE additionally allows it to be possible for internal and external stakeholders to assess model accuracy by offering a single, simply comprehensible result. RMSE is essential for assessing and justifying the models used for sales predictions and recommendation system user preference prediction when paired with other metrics or domain-specific performance indicators.

RMSE in Sales Prediction

In the context of predicting product sales, each $\hat{y}_i$ denotes the model's estimated number of product units sold during a specific period, and each y_i denotes the actual sales observed. If N test samples are evaluated, the RMSE of Equation 3.1 indicates how accurately the model captures overall demand patterns.

An RMSE value must be interpreted as being about the typical scale of the time-series or the seasonal sales range. For example, if a product regularly sells around 100 units daily, an RMSE of 4.19 implies a modest level of error compared

to daily volumes. On the other hand, if a product commonly sells only 10 units a day, an RMSE of 4.19 constitutes a substantial error proportionally.

When multiple models (for instance, Linear Regression, Random Forest, Gradient Boosting, and XGBoost) are compared, presenting both the RMSE of cross-validation and RMSE of the final test set is a clear way to convey the performance generalization of each candidate. This presentation demonstrates which method offers the smallest RMSE, thus providing a quick comparison that is both numerical and intuitive.

RMSE in Product Recommendation Systems

Many collaborative filtering algorithms internally handle the recommendation task as a *rating prediction* problem, despite the fact that recommendation systems are frequently focused on ranking or classification approaches. When a user–item pair is not observed, the system tries to estimate the user's unknown rating $\hat{r}_{ui}$. This can be compared to the true rating r_{ui} if the pair is ever observed in a test set [54].

The fundamental metric in rating prediction is the RMSE. In the test set, every data record represents a user–item interaction for which the user gave an actual rating r_{ui}. The rating $\hat{r}_{ui}$ predicted by the system might differ from this ground truth. The average error magnitude of the system can be expressed as a single value by adding up and averaging the squared deviations, then taking the square root as in Equation 3.1. This is useful when the objective is to give extremely precise numerical predictions of how a user would rate specific items or products.

For example, in a collaborative filtering experiment employing algorithms like K-Nearest Neighbors (KNNBaseline), Non-negative Matrix Factorization (NMF), and Singular Value Decomposition (SVD), one may evaluate the RMSE of each approach to see which yields the best accurate rating estimations. The system is often forecasting ratings that are closer to what people would really assign when the RMSE is smaller. If KNNBaseline produces an RMSE of 0.2731 and SVD produces 0.3147, it implies that KNNBaseline yields lower average errors for rating predictions.

It is important to keep in mind that, despite being a reliable statistic for numerical predictions, RMSE does not fully convey the quality of a recommender system. Other performance metrics that concentrate on ranking accuracy include Precision@k and Normalized Discounted Cumulative Gain (NDCG). Nonetheless, RMSE continues to be a consistent and comprehensible statistic when the rating value itself is important.

Chapter 4

Sales Prediction

4.1 Initial Data Exploration and Feature Engineering

The sales prediction component of the Dyna ML project began with an extensive exploratory data analysis (EDA) using the primary fact table, FactSales. Initial investigation revealed that the raw variables provided by FactSales were insufficient to adequately portray the complexity and unpredictability inherent in the sales data.

As a result, a comprehensive feature engineering process was carried out by examining related dimension tables (DimStore, DimProduct, DimPromotion, DimGeography, DimProductSubcategory, DimDate, DimProductCategory, DimCustomer) to identify additional meaningful attributes that could improve predictive accuracy. V_Sales is a custom SQL view created using advanced SQL joins and aggregation techniques. This view included essential data such as promotional details, product subcategories, geographical information, and a variety of temporal features (year, month, weekday), allowing for more exact and reliable sales forecasts.

4.2 Attributes and Target Variable

The target variable for the sales prediction task was defined explicitly as **SalesQuantity**, a numeric measure that indicates the number of products sold. The predictor variables were divided into two main categories:

- **Categorical Features**: `PromotionKey, StoreKey, ProductKey, channelKey, RegionCountryName, ProductSubcategoryKey, PoductCategoryKey, year, weekday, month`.
- **Numerical Features**: `UnitCost, UnitPrice, ReturnQuantity,`

`ReturnAmount`, `DiscountQuantity`, `DiscountAmount`, `TotalCost`, `SalesAmount`, `DiscountPercent`.

Since ML algorithms need numerical input to process data, categorical variables in predictive analytics typically need to be encoded. Discrete labels or text are frequently used to convey categorical information that algorithms cannot automatically comprehend. For models to comprehend and take advantage of the underlying correlations between these features and the target variable, encoding techniques translate these categorical labels into numerical forms.

4.3 Data Preprocessing

In order to guarantee model stability and forecast accuracy, these features needed to be preprocessed effectively. Various kinds of pre-processing techniques were used meticulously:

Time-based Features

The original date attribute (`DateKey`) was converted to a datetime format and subsequently decomposed into more detailed and instructive components:

- `year`: capturing the annual sales cycle.
- `month`: reflecting seasonal variations.
- `weekday`: revealing weekly purchasing patterns.

Categorical Feature Encoding

Different cardinalities required specific encoding techniques for categorical features. The following are some notable effects of high cardinality (features with many unique values) versus low cardinality (features with fewer unique values) on the choice of encoding method.

One-Hot Encoding:

Categorical variables are transformed into binary columns using one-hot encoding, each of which indicates whether a specific category is present or not. Categorical variables with low cardinality are best suited for this encoding since it avoids the introduction of unintentional ordinal relationships and guarantees the simplicity and interoperability of the algorithm [34].

Motivation: For categorical variables with comparatively few unique values, the one-hot encoding works quite well and avoids arbitrarily establishing numerical correlations between categories.

Table 4.1: Low-cardinality Categorical Features

Feature	**Unique Values**
year	3
channelKey	4
weekday	7
ProductCategoryKey	8
month	12
PromotionKey	28
ProductSubcategoryKey	32
RegionCountryName	35

One-hot encoding enables a categorical feature with k unique categories to be transformed into k distinct binary indicators, making it clear which specific category exists for each instance. Because the resulting design matrix is still computationally manageable, this method works particularly well for features where k is quite small. Formally, each observation with $X = c_j$ is mapped to the vector if a categorical feature X can take values $x \in \{c_1, c_2, \ldots, c_k\}$.

$$\big(0,\, 0, \ldots,\, 1, \ldots, 0\big),$$

where the j-th position is 1 and the remaining positions are 0. One-hot encoding guarantees that the model doesn't derive any arbitrary ordering among categories in tasks involving linear or distance-based learners. However, the newly formed binary columns may add sparsity to the data and greatly increase dimensionality if the categorical variable has a high cardinality.

Target Encoding:

The mean of the target variable for each category is substituted for the category in the target encoding [50]. In cases where one-hot encoding would provide an unreasonably high number of features, resulting in sparsity and computational inefficiency, it is especially helpful for high-cardinality categorical data. KFold cross-validation, which calculates the encoding using training folds and applies the smoothed encoding to the validation folds, was used to implement target encoding in order to prevent overfitting and information leaking.

Motivation: Target encoding provides numerical representations that improve model performance, especially for high-cardinality categorical variables, while also

dramatically reducing dimensionality and accurately capturing category-target correlations.

Table 4.2: High-cardinality Categorical Features

Feature	**Unique Values**
StoreKey	306
ProductKey	2298

Target encoding provides an alternative for categorical features that demonstrate high cardinality [33]. This approach gives each category a numerical value that represents a statistical summary of the target variable rather than creating a large number of new columns. Target encoding in its most basic form is described as follows:

$$\text{TargetEncode}(c_j) = \frac{\sum_{\text{instances } i \text{ where } X=c_j} y_i}{\text{number of instances where } X = c_j},$$

where y_i is the target for instance i. However, overfitting may result from depending only on this naive mean, particularly when dealing with sparse or unbalanced data. KFold cross-validation is used to reduce overfitting by ensuring that the mean used to encode every observation is only calculated from folds that do not include that observation. When a category has a limited sample size, it is common practice to add an extra smoothing factor to bring the per-category mean closer to the global average. The smoothing can be shown as follows:

$$\text{Smoothed}(c_j) = \frac{n_j\,\overline{y_j} + \alpha\,\overline{y}}{n_j + \alpha},$$

where n_j is the count of category c_j, $\overline{y_j}$ is the mean target for that category, $\overline{y}$ is the global target mean, and α is a positive parameter.This regularization helps prevent overly deflated or inflated target-encoded values from being acquired by categories with small sample sizes. Target encoding reduces the possibility of information leakage while capturing the relationship between each categorical value and the target by using such smoothing and integrating cross-validation in the transformation step.

Numeric Feature Scaling

Numeric features `UnitCost, UnitPrice, ReturnQuantity, ReturnAmount, DiscountQuantity, DiscountAmount, TotalCost, SalesAmount, and DiscountPercent` were standardized using the **StandardScaler** method from scikit-learn.

The objective of standard scaling is to reduce bias resulting from significant variations in numerical ranges by transforming each numeric feature to have a zero mean and unit variance. The sample mean μ is subtracted from each data point x_i in this method, and the result is divided by the standard deviation σ, which can be written mathematically as:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Standardization guarantees that learning is not dominated by features with huge sizes by transforming all numerical variables into a comparable scale. When each dimension has a similar scale, techniques like distance-based algorithms and gradient-based optimizers frequently converge more rapidly and consistently. Additionally, standard scaling improves overall model performance and offers more consistent training dynamics for algorithms that are sensitive to feature magnitude. The `StandardScaler` from scikit-learn is a popular first choice for preprocessing numerical features due to its inherent simplicity and effectiveness, especially when the goal is to normalize the influence of features across all numeric variables and the underlying distribution of values is not extremely heavy-tailed.

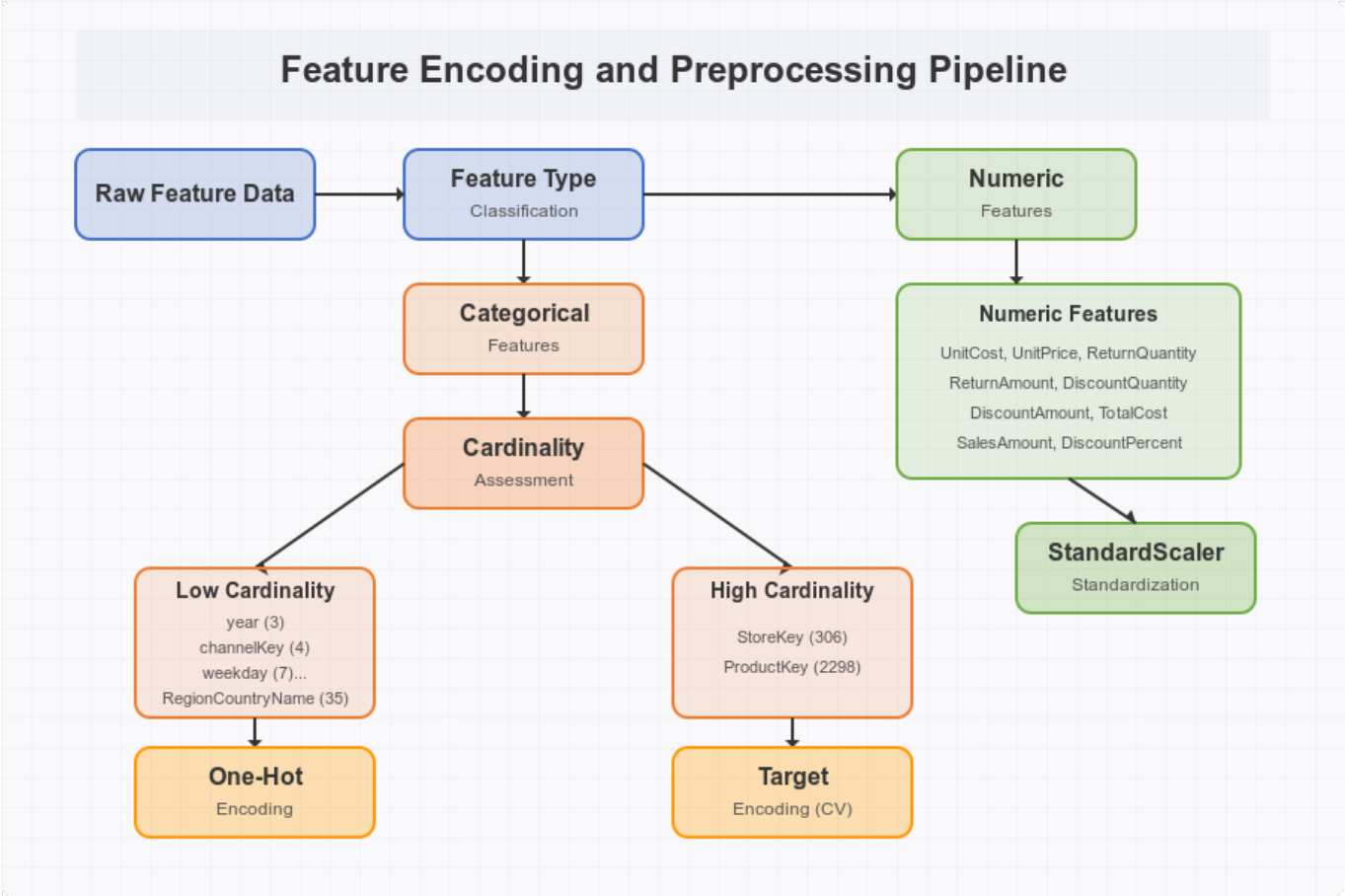

Figure 4.1: Feature Encoding and Preprocessing Pipeline

The data was successfully prepared using this thorough and strategically justified preprocessing pipeline, ensuring that the ML models could fully utilize feature-target correlations for accurate sales prediction.

4.4 Baseline and Model Selection

Initially, several supervised ML methods were examined in order to tackle the sales prediction challenge methodically. The process of choosing an appropriate model began with a baseline to set a performance standard, and then more complicated models were evaluated to better describe the dataset's underlying complexities.

Ridge Regression (Baseline Model) Ridge regression is a linear regression variation that includes L2 regularization. Ridge regression successfully addresses multicollinearity among predictor variables by adding a penalty term appropriate to the squared size of the coefficients. This strengthens coefficient estimates and improves generalization performance on unseen data. It is the ideal baseline model because of its ease of use and computational effectiveness.

Random Forest The ensemble technique, Random Forest, builds several decision trees during training and averages their results (for regression problems) to generate a stable and reliable prediction [37]. Because of its integrated ensemble averaging, the approach is particularly effective in handling high-dimensional data, managing noisy datasets, and capturing nonlinear relationships. However, if tree complexity is not properly controlled, it could overfit training data.

Gradient Boosting Another powerful, iterative ensemble approach for successively creating predictive models is gradient boosting. Particularly for datasets with intricate, nonlinear interactions, each successive model (often shallow decision trees) is built to reduce residual errors from earlier trees, producing incredibly precise predictions [35]. Although gradient boosting is renowned for its precision and adaptability, it usually necessitates meticulous hyperparameter adjustment to prevent overfitting and preserve generalization.

XGBoost Extreme Gradient Boosting, or XGBoost, is a sophisticated gradient boosting algorithm that is renowned for its speed, scalability, and enhanced regularization methods [12]. XGBoost is a popular approach in predictive modeling challenges because it provides many regularization parameters (such L1 and L2), improved computation through parallel processing, and effective handling of sparse data. When properly adjusted, its robustness and performance efficiency frequently lead to higher predictive skills [36].

Support Vector Regression (SVR) Regression difficulties are addressed by Support Vector Regression (SVR), which applies the ideas of Support Vector Machines (SVM). SVR efficiently captures intricate correlations by using kernel functions to map input data into higher-dimensional regions. SVR is computationally demanding and extremely sensitive to hyperparameter selection, which makes it difficult to use for very large or high-dimensional datasets, despite its theoretical stability and efficacy in nonlinear modeling [28] [27].

4.5 Initial Model Performance Comparison

An initial evaluation of these models provided the following cross-validation (CV) and test set performances are shown in Table 4.3:

Table 4.3: Initial Model Performance Comparison (RMSE)

Model	CV RMSE	Test RMSE
Ridge Regression (Baseline)	19.96 ± 5.15	**15.57**
Random Forest	10.76 ± 4.17	16.74
Gradient Boosting	8.16 ± 2.99	21.52
XGBoost	10.34 ± 4.53	17.44
SVR	28.57 ± 9.07	28.88

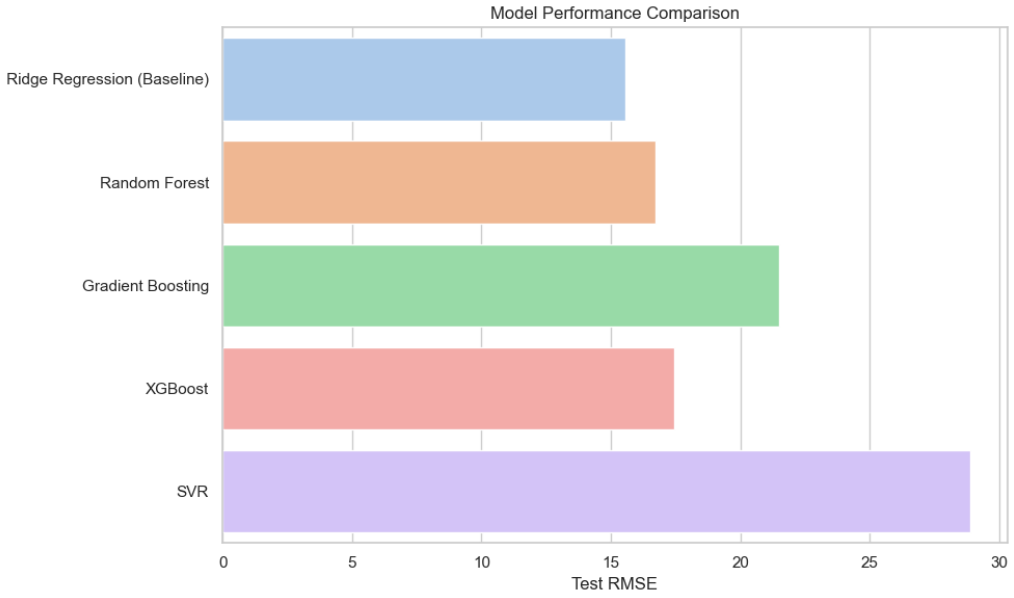

Figure 4.2: Initial Model Performance Comparison (RMSE)

The simpler Ridge Regression model was initially the best-performing model on the test set, despite expectations that more sophisticated, tree-based ensemble models may perform better because of their capacity to capture nonlinearities and complicated feature interactions.

Our baseline model, the Ridge Regression model, uses L2 regularization to manage complexity and decrease feature multicollinearity. The dataset's underlying relationships may primarily display linear characteristics, or at the very least, not complex enough to justify more complex, nonlinear modeling at first attempt, according to its strong generalization capability in our initial tests (achieving a Test RMSE of 15.57). Ridge Regression obtained the lowest test RMSE among initial choices because of its built-in regularization capacity, which also successfully reduced the risk of overfitting to the training data and allowed it to retain consistent predictions on unseen observations.

On the other hand, because of their versatility in simulating complex, non-linear

relationships seen in retail sales data, ensemble approaches like Random Forest, Gradient Boosting, and XGBoost showed noticeably better cross-validation results. Their higher cross-validation results, however, were not reflected in their prediction performance in the test dataset, which is a clear indication of overfitting. This disparity between test and cross-validation performance in ML typically indicates either a lack of regularization or a lack of hyperparameter tuning, leading to models that overadapt to the oddities of the training subsets and fail optimal generalization.

Specifically, the Gradient Boosting and XGBoost models, despite their promising cross-validation outcomes, revealed an important discrepancy between training and testing errors. This pattern suggests that these models might have performed poorly when tested on completely unseen data because they may have picked up on fine details or noise unique to the training data subsets used during cross-validation. In the same way, the Random Forest model displayed a smaller but still significant difference, suggesting that while ensemble approaches are capable of effectively utilizing complicated relationships, careful hyperparameter tuning is necessary to achieve robust generalization.

Another nonlinear model, SVR, had the highest error of all the models that were evaluated. This suggests that either the dataset's essential properties, such as categorical features encoded through linear-target relationships, made SVR less than ideal for this predictive scenario, or that its default hyperparameters were inadequate to capture underlying complexities.

Critical insights were obtained from this preliminary investigation, which showed that although complicated ensemble models have a lot of potential, they need to be systematically optimized for hyperparameters before they can be deemed superior. Therefore, we made the decision to carry out thorough hyperparameter optimization in order to fully utilize the sophisticated models and produce forecasts that are more accurate. In order to properly balance the bias-variance trade-off and guarantee increased performance and generalization capabilities on unseen retail sales data, this tuning procedure was crucial. As explained in the following sections, subsequent tuning attempts confirmed this theory and significantly shifted the performance rankings in favor of ensemble-based models.

4.6 Hyperparameter Tuning for Improved Performance

The predictive accuracy and generalization capacity of the models are greatly impacted by the settings of the many hyperparameters that are commonly included in ML methods. The complexity, learning dynamics, and general behavior of ML

algorithms are controlled by hyperparameters, which are external parameters that are defined prior to the learning process. Finding the ideal hyperparameter values is essential since inadequate settings can lead to overfitting, poor performance, or an inability to generalize well to new data.

We used GridSearchCV to conduct a thorough hyperparameter tuning procedure in order to resolve the performance issues found during the first modeling stage. To find the best settings, this approach methodically tests an established number of hyperparameter combinations. GridSearchCV evaluates every hyperparameter setting by conducting a thorough search and cross-validating the results across the training set. This method's primary benefit is its comprehensiveness and rigorousness in covering the parameter space, which guarantees the selection of hyperparameters that minimize the prediction error by best generalizing to unknown data.

In this study, the specific objectives and anticipations of hyperparameter tuning were:

- To significantly decrease the model prediction error (RMSE) on test data that is unseen.
- To enhance the models' capacity to capture the complicated relationships and nonlinearities present in retail sales data.
- To improve model robustness and reduce overfitting by managing complexity appropriately.

4.6.1 GridSearchCV Hyperparameter Tuning

Cross-validation and a predefined hyperparameter search space were used to thoroughly fine-tune each model. Using cross-validation to assess each set, GridSearchCV is a hyperparameter optimization method that thoroughly examines every possible combination of parameters within a preset grid. Based on the selected scoring criteria, in this case, the RMSE, it chooses the parameter combination that produces the best performance. GridSearchCV was selected because of its extensive and systematic approach, which guarantees a thorough investigation of the hyperparameter space and finds the most reliable model configuration. In particular, each model's hyperparameters were optimized as follows:

- **Ridge Regression:** The regularization strength (`alpha`), which balances the magnitude of the coefficients, was optimized with the objective of handling multicollinearity and controlling complexity in order to avoid overfitting and enhance stability.

- **Random Forest:** The maximum depth (`max_depth`) and the number of trees (`n_estimators`) were adjusted. While controlling tree depth keeps each tree from fitting extremely specific patterns, increasing the number of trees improves generalization by reducing variance through prediction averaging.
- **Gradient Boosting and XGBoost:** The number of estimators (`n_estimators`), learning rate (`learning_rate`), and maximum depth (`max_depth`) were among the hyperparameters that were adjusted. Each tree's rate of error correction from previous trees is determined by its learning rate. Optimizing the number of trees and tree depth at the same time balances complexity and effectively reduces residual errors in the model.
- **Support Vector Regression (SVR):** The purpose of optimizing hyperparameters like epsilon, regularization (`C`), and kernel types was to improve the model's ability to capture complex nonlinear interactions.

4.6.2 Tuned Model Results

Hyperparameter optimization led to substantial performance improvements, as summarized clearly in Table 4.4 and illustrated visually in Figure 4.4.

Table 4.4: Model Performance Comparison (RMSE) Before and After Tuning

Model	Test RMSE (Before Tuning)	Test RMSE (After Tuning)
Ridge Regression	15.57	16.21
Random Forest	16.74	5.14
Gradient Boosting	21.52	5.57
XGBoost	17.44	**4.19**
SVR	28.88	23.36

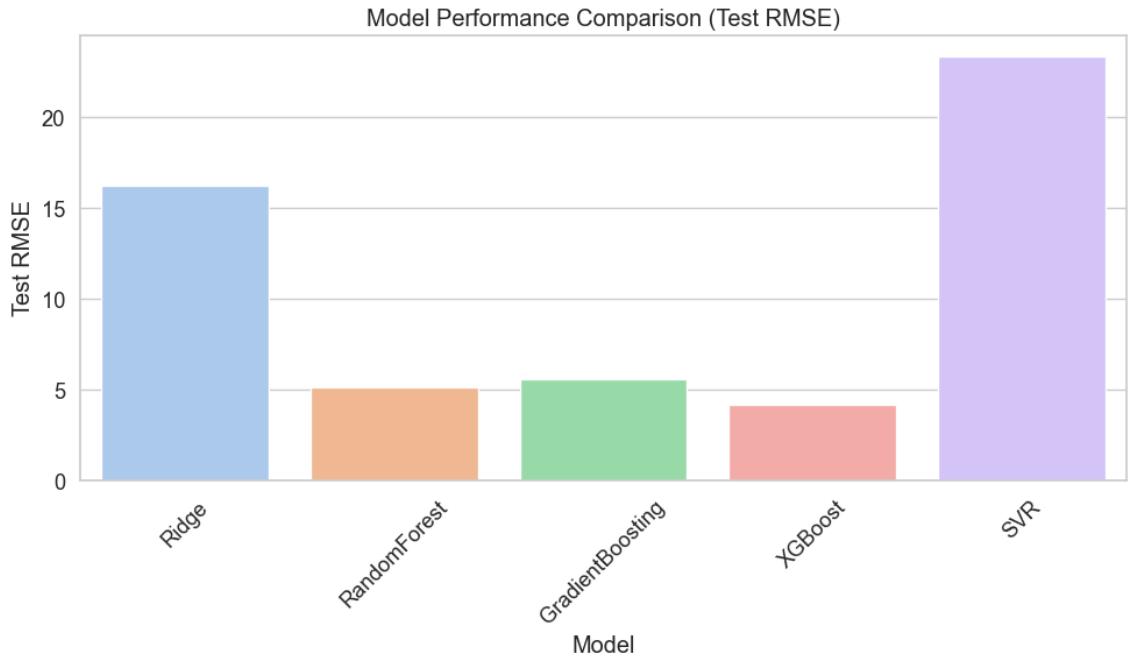

Figure 4.3: Model Performance after Hyperparameter Tuning (RMSE)

The ensemble-based approaches (Random Forest and XGBoost) showed the most notable increases, confirming that hyperparameter tuning is an effective technique for improving generalization and prediction accuracy.

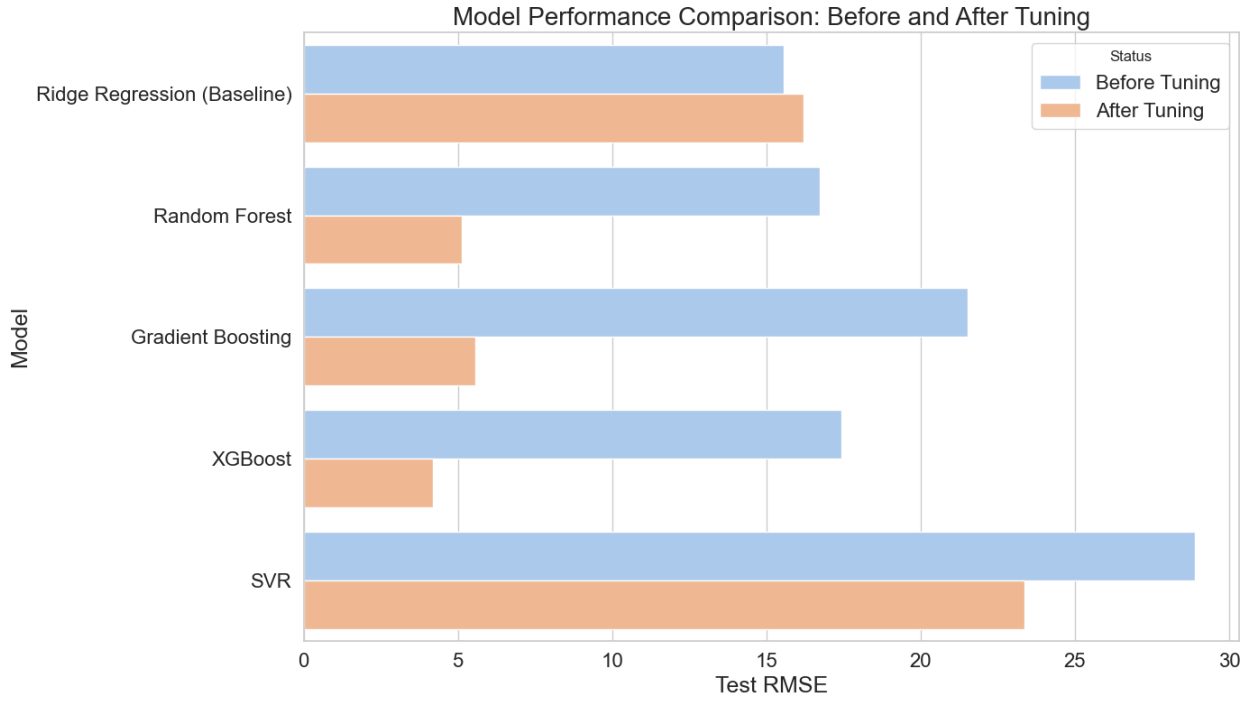

Figure 4.4: Bar Plot of Test RMSE Comparison (Before and After Hyperparameter Tuning)

4.7 Interpretation and Final Model Selection

The comparative model performance landscape was significantly changed by the hyperparameter tuning stage. Out of all the tuned models, **XGBoost** outperformed the rest by achieving the lowest test RMSE (4.19). The advanced gradient-boosting framework XGBoost iteratively creates decision trees to reduce residual errors, successfully capturing the complicated feature interactions and elaborate nonlinear patterns common in retail sales datasets.

Random Forest additionally showed substantial improvement, achieving a competitive test RMSE of 5.14. Its small underperformance in comparison to XGBoost, however, despite its great generalization, demonstrated XGBoost's advantage in modeling complex relationships between a large number of numerical and categorical parameters.

Gradient Boosting closely matched Random Forest and demonstrated significant improvement (RMSE = 5.57). Although its iterative residual correction capacity greatly improved predictions, it was still marginally less successful than XGBoost, perhaps as a result of minor variations in optimization approach or sensitivity to hyperparameter selection.

Ridge Regression, given its simplicity and interpretability, did not improve considerably after tuning (RMSE = 16.21). The need for more adaptable models was highlighted by the fact that its underlying linear assumption was insufficient to describe the intricate, non-linear interactions present in retail data.

SVR, remained the least effective even after hyperparameter tuning (RMSE = 23.36), probably because of its inherent computational limitations when handling high-dimensional data with significant categorical encoding, as well as its sensitivity to feature scaling and kernel selection.

XGBoost was chosen as the final model because of its outstanding generalization to unknown data and better predicted accuracy. Highly accurate and actionable sales predictions are now achievable mainly to the systematic saving of this finely trained model (as `best_model.joblib`) and its integration into the Dyna ML analytical framework.

4.7.1 Final Model and Practical Implications

Implementing the improved XGBoost model yields precise and trustworthy sales projections, which are crucial for strategic decision-making, inventory optimization, focused marketing campaigns, and operational effectiveness. In retail settings, its remarkable predictive powers guarantee significant, data-driven insights that boost profitability and customer satisfaction.

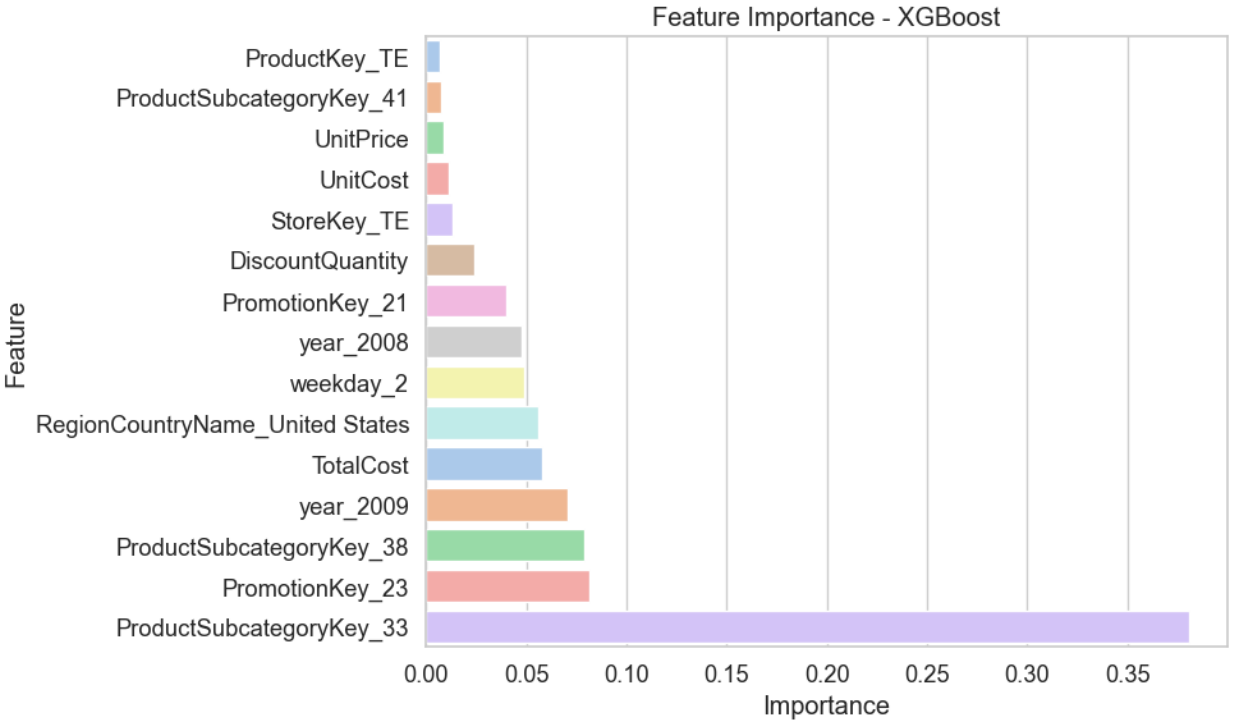

Figure 4.5: Top 15 Feature Importance in XGBoost Model

Knowing how much each feature contributes to the final predictions is a crucial part of predictive modeling. The top 15 most significant features identified by the tuned XGBoost mode based on the calculated significance scores are shown in Figure 4.5. The degree to which each attribute helps reduce the loss function when employed in the model's ensemble of decision trees is commonly used for evaluating feature relevance in XGBoost.

Notably, the one-hot encoded `ProductSubcategoryKey_33` is the most influential feature with an approximate importance score of 0.35, indicating that the specific product subcategory it represents captures a substantial portion of the variance in sales behavior. Following this, `PromotionKey_23` exhibits an importance of around 0.10, underscoring the significant impact of certain promotional campaigns on sales volume. Additionally, `ProductSubcategoryKey_38` contributes with an importance score of approximately 0.09, highlighting the value of granular product-

level segmentation in capturing critical sales trends. Temporal indicators further play a pivotal role, as shown by `year_2009` with an importance value between 0.07 and 0.08 and `year_2008` with an importance of about 0.04, which together capture the influence of historical context and macroeconomic conditions on sales performance. The feature `TotalCost`, with an importance score of roughly 0.06 to 0.07, likely reflects the effect of production or acquisition costs on pricing strategies and overall profitability.

The geographic indicator `RegionCountryName_United States` (approximately 0.05) and the temporal feature `weekday_2` (also around 0.05) illustrate the impact of regional market characteristics and day-of-the-week effects on sales. Economic variables, including `DiscountQuantity` (around 0.02), `UnitCost` (approximately 0.01), and `UnitPrice` (approximately 0.01), highlight the sensitivity of customer demand to pricing strategies. Moreover, `StoreKey_TE`, a target-encoded identifier for stores with an importance of roughly 0.02, captures location-specific historical sales patterns, whereas `ProductSubcategoryKey_41` (with an importance below 0.01) and `ProductKey_TE` (approximately 0.00) contribute minimally to the predictive performance of the model.

The overall findings confirm that a combination of finely grained product details, strategic promotional tracking, and both temporal and regional contextual data is fundamental for generating accurate and actionable sales forecasts. These insights validate the feature engineering and encoding strategies employed in this study, demonstrating that focusing on key product subcategories, effective promotions, and localized as well as historical trends can substantially improve inventory management, resource allocation, and operational efficiency in retail analytics.

The analysis reveals that the dominant predictors in our model are product-related features, particularly the product subcategory indicators `ProductSubcategory Key_33` and `ProductSubcategoryKey_38`, which capture the most significant variance in sales results. In contrast to conventional high-cardinality features such as `ProductKey_TE` and `StoreKey_TE`, whose contributions are minimal in this context, these subcategory features, along with targeted promotional signals like `PromotionKey_23` and `PromotionKey_21`, highlight the superior predictive strength of more aggregated and contextually enriched representations. This finding reinforces the effectiveness of our feature engineering and encoding strategy, which transforms granular categorical data into robust numeric predictors that drive model accuracy.

The comparatively lower importance of generalized categorical features, such as broad temporal or geographic indicators not ranked among the top predictors, underscores that the fine-grained product subcategories and specific promotional variables exert a more pronounced influence on sales volume predictions. In our results, while certain temporal features (for example, `year_2009` and `year_2008`)

and regional markers (for example, `RegionCountryName_United States`) play a supportive role, it is the detailed product and promotion signals that capture the nuanced dynamics of consumer behavior, thereby confirming that specificity in feature representation is critical for forecast accuracy.

By offering precise and useful data for the retail strategy, this comprehensive feature importance study verifies our feature engineering methodology. In order to improve inventory management, enhance promotional campaigns, and more efficiently allocate resources, retailers can prioritize product subcategories with historically strong sales signals and use targeted promotional and temporal indicators. In order to improve financial performance and operational efficiency, the results indicate a strategic focus on data-driven, precisely customized features that reflect market-specific and product-specific dynamics.

Chapter 5

Product Recommendation

Recommendation systems contribute significantly to business success by improving customer experience and driving sales through personalized suggestions [55]. The two main approaches used in this project, Collaborative Filtering and Content-Based Filtering, are covered in detail in this section. The emphasis focuses on the rationale behind the implemented rating system, detailed preprocessing steps, and how these systems leverage the unique data view created specifically for optimal recommendation performance.

Despite their strength, recommendation systems face a number of obstacles and limitations when used in practical settings. Among these difficulties, the most prominent ones are as follows:

- **Cold-Start Problem:** Insufficient information about new users or new products in a system might lead to this situation [51][48]. New users or items with limited interactions result in poor recommendations because collaborative filtering depends mainly on user-item interaction histories.
- **Data Sparsity:** Users frequently rate or buy only a small portion of the items in huge catalogs, which leads to extremely sparse user-item matrices. Traditional neighborhood-based techniques and matrix factorization strategies may become less successful due to sparsity.
- **Diversity & Serendipity:** To maintain high user engagement, systems need to achieve a balance between offering popular items that users are likely to enjoy and making new or less evident recommendations.

The cold-start problem is the most noticeable of these difficulties. A common scenario is when a new user registers for a service and the system does not have enough interaction history to provide precise, specific recommendations. Similarly, because there are little to no user-item interaction data, recently released products also have limited exposure in collaborative filtering models [51][48][52].

In order to address the cold start problem, platforms frequently use hybrid strategies that combine content-based features with collaborative signals. By implementing a *content-based* strategy for products with inadequate interaction data and a *popularity-based* recommendation layer for new clients, our project can partially address the *cold-start problem*. Users with similar interests may still be recommended new or less-rated products based on product descriptions. Even in instances where collaboration signals are scarce or nonexistent, this method preserves the overall quality of the proposal.

5.1 Collaborative Filtering

An effective technique in recommendation systems is collaborative filtering, which takes advantage of past user-product interactions to forecast future references [53][55]. This project uses collaborative filtering to create personalized product recommendations based on past purchases made by users. Collaborative filtering's main advantage is its capacity to make recommendations based only on users' prior interactions, without the need for specific details on the characteristics of the products. It finds patterns of similarity between users or items and recommends products with which similar users have had positive experiences in the past [57].

Collaborative filtering was chosen because it has been shown to be successful in identifying user preferences and making appropriate product recommendations, especially when there is enough historical data available. This method immediately benefits from a robust rating system that precisely evaluates user-product interactions, greatly impacting the level of quality and applicability of the suggestions produced.

5.1.1 Initial Data Preparation and Rating System

The sales, customer, and product tables each have useful but insufficient information that cannot be used solely to accurately estimate client preferences. It was crucial to develop a thorough and unified data representation in order to achieve its full predictive potential. In order to compile the required dimensions and metrics from several relevant columns, a specialized SQL view (V_ProductRecommendation) was created. The final dataset constructed through this view contains the following columns: `ProductCategoryName, ProductSubcategory, Product, ProductKey, CustomerKey, DateKey, SalesQuantity, SalesAmount, ReturnQuantity, ReturnAmount`.

This enriched dataset guarantees a comprehensive representation of consumer behavior and makes meaningful recommendation modeling appropriate. The constructed view offers consolidated customer-product interaction data, including prod-

uct categories, subcategories, aggregated sales quantities, and financial figures related to sales and returns. This enriched dataset allows for an extensive and efficient analysis of customer behavior and product performance, greatly enhancing recommendation outcomes.

5.1.1.1 Development of Rating System

For collaborative filtering to be effective, user preferences must be precisely determined. Direct, trustworthy ratings were difficult to extract due to the transactional nature of the data, which is characterized by intrinsic skewness and variation. As a result, we created a custom rating system that accurately captures the user-product engagement.

In order to determine net sales figures that accurately reflected real client interest, return quantities, and amounts were first subtracted from total sales. These metrics are computed formally as follows:

$$\text{NetSalesQuantity} = \text{SalesQuantity} - \text{ReturnQuantity}, \tag{5.1}$$

$$\text{NetSalesAmount} = \text{SalesAmount} - \text{ReturnAmount} \tag{5.2}$$

We used a square-root transformation since the net sales data showed a high degree of skewness:

$$\begin{aligned} \text{NormalizedQuantity} &= \sqrt{\text{NetSalesQuantity}}, \\ \text{NormalizedAmount} &= \sqrt{\text{NetSalesAmount}} \end{aligned} \tag{5.3}$$

Following the square-root transformation, these metrics were further scaled onto a standard range of 0–1 by dividing by their corresponding maximum values:

$$\text{NormalizedQuantity} = \frac{\text{NormalizedQuantity}}{\max(\text{NormalizedQuantity})}, \tag{5.4}$$

$$\text{NormalizedAmount} = \frac{\text{NormalizedAmount}}{\max(\text{NormalizedAmount})} \tag{5.5}$$

Lastly, a weighted linear combination was used to calculate the ratings, with a greater emphasis on the financial value (sales amount):

$$\textbf{Rating} = 1 + 4 \times (0.4 \times \textbf{NormalizedQuantity} + 0.6 \times \textbf{NormalizedAmount}) \tag{5.6}$$

To better understand the distribution of the generated ratings, the following table (Table 5.1) shows the frequency of products within each rating category:

Table 5.1: Distribution of Products by Rating Category

Rating Category	Number of Products
1 – 1.5	0
1.5 – 2	2114
2 – 2.5	4851
2.5 – 3	2143
3 – 3.5	346
3.5 – 4	126
4 – 4.5	6
4.5 – 5	0

Furthermore, the top-rated products according to the average calculated rating are displayed in Table 5.2, highlighting the efficiency and reliability of the created rating system in capturing customer preferences and product popularity.

Table 5.2: Top 10 Products by Average Rating

Product Name	Average Rating
SV 16xDVD M360 Black	224
Adventure Works 26" 720p LCD HDTV M140 Silver	200
A. Datum SLR Camera X137 Grey	98
Contoso Telephoto Conversion Lens X400 Silver	90
SV Keyboard E90 White	81
Contoso In-Line Coupler E180 Silver	78
Contoso Optical USB Mouse M45 White	69
Cigarette Lighter Adapter for Contoso Phones E110 Red	65
Contoso 4G MP3 Player E400 Silver	63
Reusable Phone Screen Protector E120	62

These comprehensive assessments confirm the reliability of our rating system and guarantee that it accurately reflects customer preferences and product performance, providing a solid basis for later collaborative filtering recommendation modeling.

5.1.2 Model Development and Evaluation

After building a strong rating system, we used the *Surprise* library, a well-known Python toolkit for recommender systems, to apply collaborative filtering algorithms to the data. In general terms, there are two types of collaborative filtering techniques: matrix factorization methods and neighborhood-based approaches [45]. Finding the best predictive model for our dataset is the basic justification for comparing several collaborative filtering strategies.

Recommender model creation and benchmarking are made easier by Surprise's many benefits [9]. First, it has a standardized data loading system that enables to use of built-in datasets or create rating datasets from `pandas` DataFrames. Second, the library reduces redundant code and encourages reproducible experiments by automating hyperparameter tuning and cross-validation. Third, by offering uniform metrics (for example, RMSE and MAE) and consistent API calls for training, testing, and predictions, it makes it easier to compare various collaborative filtering algorithms. These architectural approaches allow researchers and practitioners to focus on model selection instead of technical details.

Encapsulating user-item-rating triplets, the `Dataset` class is the foundation of Surprise. By defining the `rating_scale` and column ordering, a `Reader` object controls the parsing of raw rating data. After loading, typical splitting or cross-validation procedures can be employed to instantiate test and train sets. Because of the library's modular design, experimenting with different recommendation algorithms can be done rapidly. As illustrated in the listing below, a typical workflow includes:

1. Loading or building a `Dataset` from a built-in dataset or `pandas` DataFrame.
2. Selecting an algorithm from the supported algorithms (for example, SVD, KNNBaseline, NMF).
3. Evaluating model performance using cross-validation based on MAE or RMSE.
4. The final model is trained using the complete dataset and produces top-N recommendations or predictions.

The `model_selection` methods enable cross-validated parameter searches, and the `accuracy` module calculates RMSE and other metrics for evaluation.

A wide range of *matrix factorization* and *neighborhood-based* algorithms, as well as baseline and naive predictors, are implemented by the Surprise library [9]. The following classes were essential to our project:

- `SVD`, `SVD++`, and `NMF` for factorizing the user-item rating matrix into latent representations.
- `KNNBasic`, `KNNWithMeans`, `KNNBaseline`, and `KNNWithZScore` for neighborhood based approaches.
- `BaselineOnly` for modeling global baseline ratings derived from user/item biases.
- `SlopeOne`, `CoClustering`, and `NormalPredictor` as baseline or substitute techniques that are appropriate for comparison.

This variety of techniques allows for a comprehensive comparison of accuracy, scalability, and conceptual complexity, guaranteeing that the chosen model best suits the properties of the dataset.

Surprise was the main engine for collaborative filtering experimentation in our recommendation pipeline. A `pandas` DataFrame with three crucial columns—`CustomerKey`, `ProductKey`, and `Rating`—was created using the data from our SQL-based rating system. We used cross-validation procedures to compare the performance of the chosen algorithms after enclosing these data in a Surprise `Dataset`. The primary criterion for evaluating accuracy was the RMSE, which is consistent with best practices for assessing prediction tasks.

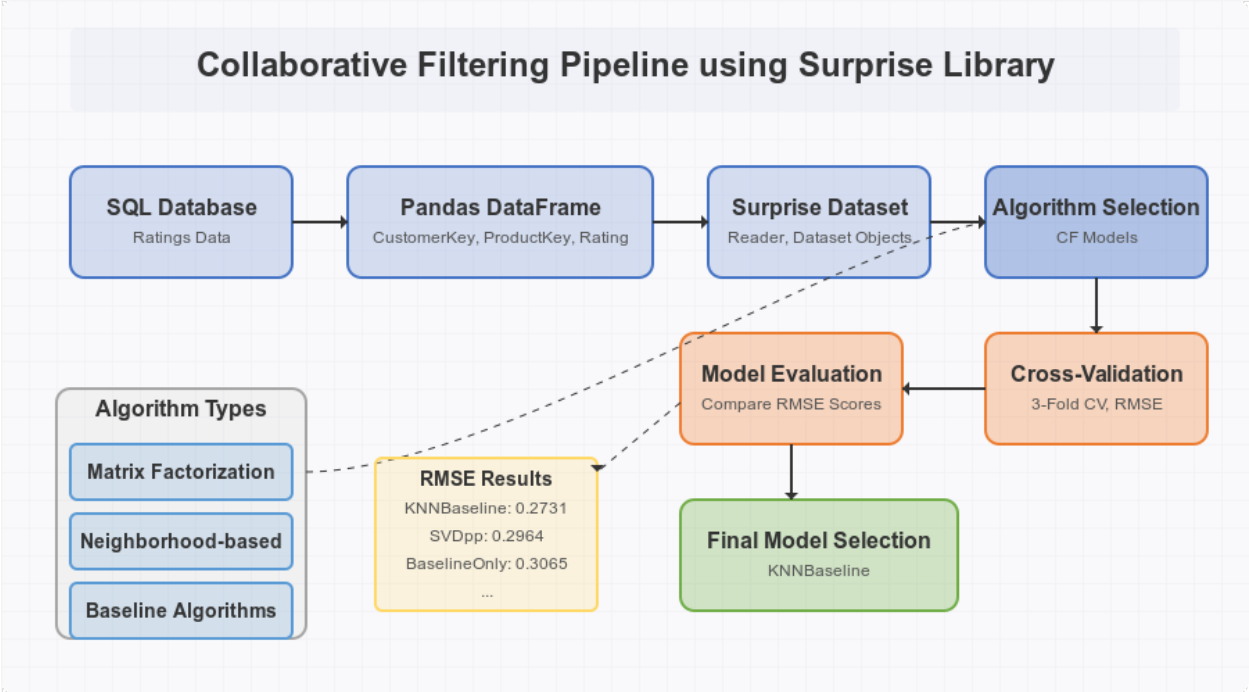

Figure 5.1: Conceptual Overview of the Surprise Library Framework

Figure 5.1 provides a simplified overview of how the Surprise library fits into our recommendation workflow, demonstrating how raw rating data is transformed into trainable datasets and subsequently used by a variety of collaborative filtering algorithms to generate predictions.

The RMSE, which is generally considered as a reliable indicator of the accuracy of rating prediction, was the evaluation parameter used to compare these algorithms. To ensure that performance estimations are reliable, we conduct cross-validation. The compared performance metrics are outlined in Table 5.3.

Based on the benchmarking results, **KNNBaseline** emerged as the optimal algorithm, achieving the lowest RMSE score of 0.2731. Therefore, KNNBaseline was selected as the final collaborative filtering model for further training and deployment.

Table 5.3: Collaborative Filtering Algorithms Benchmarking Results

Algorithm	**Mean RMSE**
KNNBaseline	0.2731
SVDpp	0.2964
BaselineOnly	0.3065
NMF	0.3109
SVD	0.3147
KNNBasic	0.3180
KNNWithMeans	0.3409
KNNWithZScore	0.3439
SlopeOne	0.4727
NormalPredictor	0.5421
CoClustering	0.5875

5.1.3 Final Model Selection and Evaluation

KNNBaseline's significant performance advantage is due to its sophisticated method of calculating similarities. KNNBaseline specifically uses Alternating Least Squares (ALS) optimization to adjust the baseline ratings according to user-item deviations from the global average, resulting in more accurate and reliable predictions.

SVD and NMF, two matrix factorization-based techniques, also demonstrated commendable performance, demonstrating their ability to successfully capture latent factors [29][27]. However, because of the unique features of our dataset, like sparsity and variability in user-item interactions, they were slightly less accurate than KNNBaseline. Although they offered useful comparative insights, baseline approaches lacked the granularity and depth of more sophisticated algorithms.

The performance of KNNBasic and other less complex neighborhood techniques was average. The lower performance of CoClustering and NormalPredictor indicates that these algorithms are less appropriate for datasets with skewed interactions and high sparsity, which are characteristics of real-world sales data.

KNNBaseline was chosen as the final model for deployment after these thorough tests, re-trained on the entire training dataset, and then serialized to be employed in producing dynamic recommendations in real-time scenarios.

This comprehensive benchmarking procedure not only validated our model selection, but also described the benefits and drawbacks of each method, providing useful data for future study and practical applications in related domains.

Personalized product recommendations are capable of being dynamically generated by the trained collaborative filtering algorithm. Estimated ratings for products with which each client has never interacted with before are used to generate recommendations. These anticipated ratings are then used to choose the best products,

therefore, customizing the recommendations.

Figures illustrating the distribution of the calculated ratings and the top products recommended to customers are presented in Figures 5.2 and 5.3, respectively.

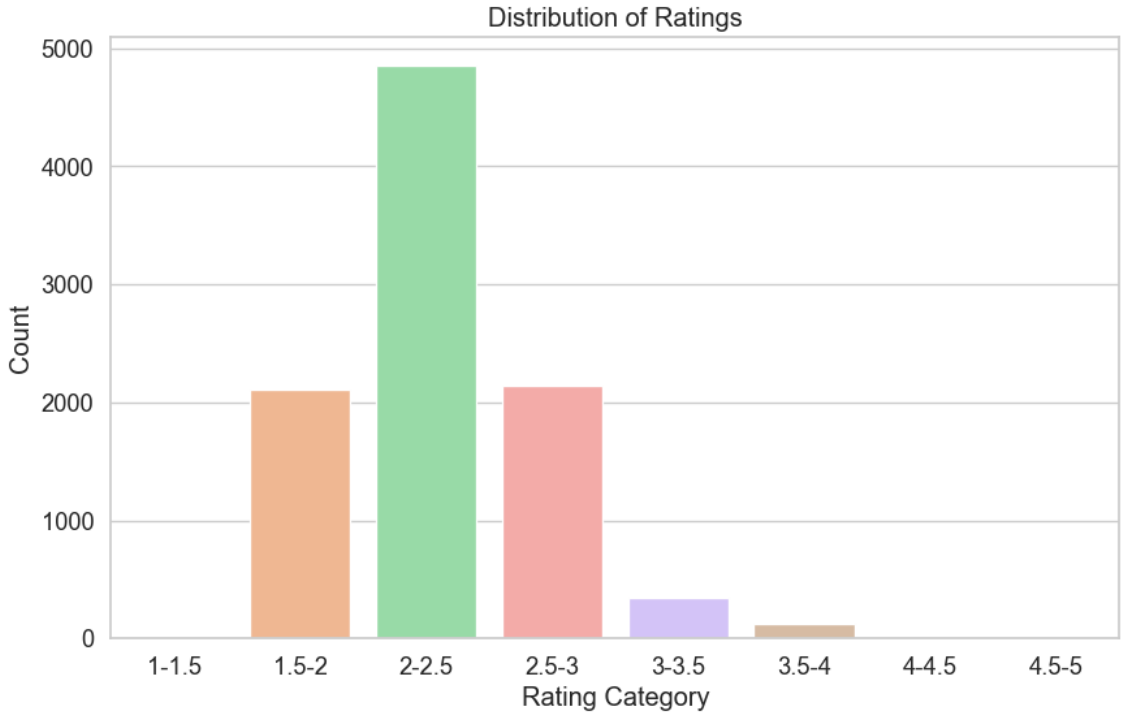

Figure 5.2: Distribution of Ratings

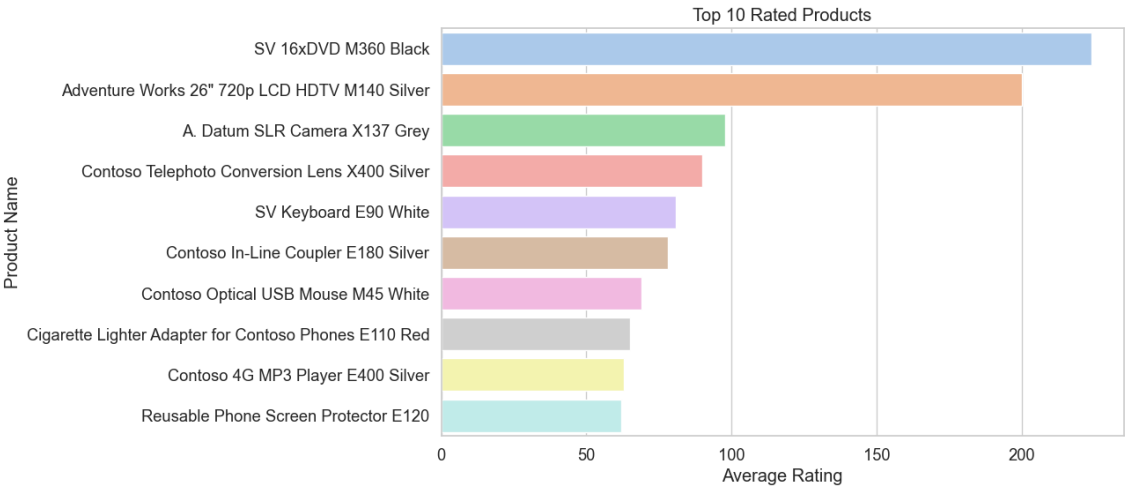

Figure 5.3: Top 10 Products by Average Rating

These visualizations demonstrate how well the system can differentiate between products according to consumer preferences, confirming the rating system's applicability and efficacy.

In conclusion, the meticulous algorithm selection procedure and the customized grading system ensured that collaborative filtering generated high-quality personalized suggestions. To increase the precision and scope of recommendations, future studies could investigate the combination of this process with other approaches, including hybrid recommendation systems.

5.2 Content-Based Filtering

Content-based filtering suggests products that are similar to items that customers have liked or engaged with positively in the past by using item-specific characteristics or descriptions. The cold-start problem, where there is limited or no historical user interaction data, can be successfully handled via content-based methods rather than collaborative filtering. Additionally, they are not dependent on significant amounts of user interaction. This method works particularly well when descriptive textual data or particular product characteristics are readily available.

In this study, textual product descriptions have been utilized to implement content-based recommendations leveraging clustering methods and Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. In particular, we adopted the KMeans clustering approach due to its scalability and robustness.

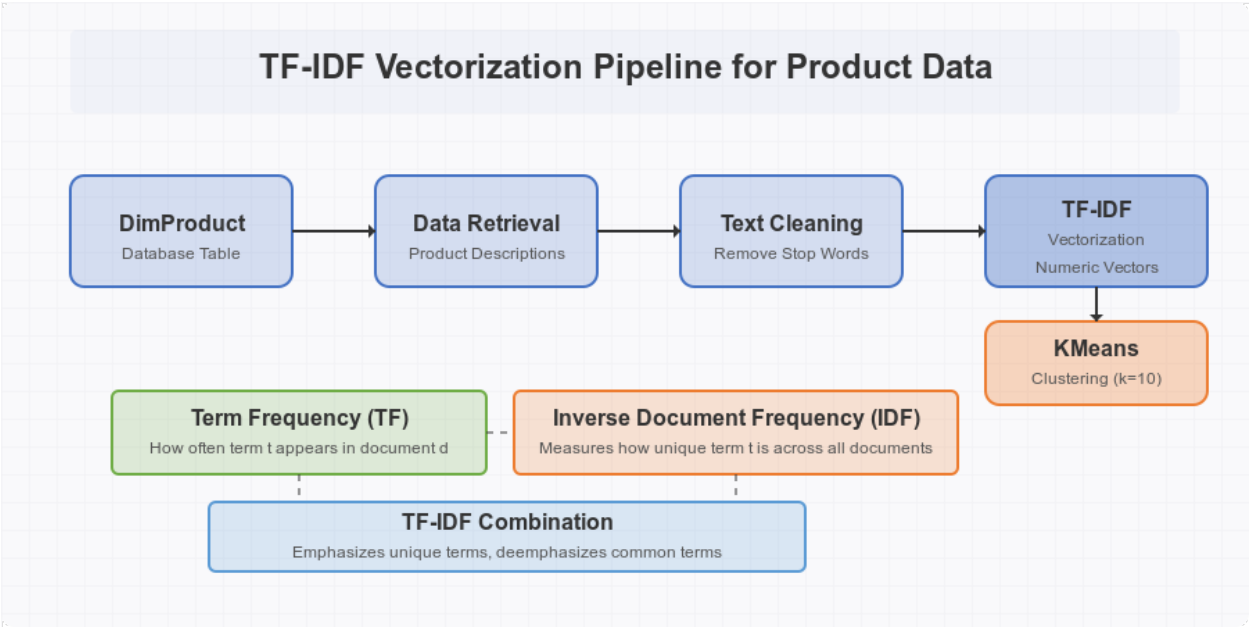

Figure 5.4: TF-IDF Vectorization Pipeline for Product Data

5.2.1 Data Preparation and Text Vectorization

The product data, including detailed descriptions, were retrieved from the 'DimProduct' database table. Textual descriptions went through a cleaning process to remove stop words, followed by the application of the TF-IDF algorithm.

Since it offers a method based on mathematics for determining the corresponding importance of terms in a document, TF-IDF has proven to be a key component of text mining and information retrieval. TF-IDF highlights terms that are particularly prevalent in certain documents while minimizing words that are common to most documents by recording both the frequency of a term in a single document (Term Frequency, TF) and the term's distribution throughout the entire corpus (Inverse Document Frequency, IDF) [47].

Formally, let t be a term, d a document, and D the corpus. The TF-IDF weight of t in d is computed as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \ \times \ \log\Big(\frac{N}{|\{\, d' \in D : t \in d'\}|}\Big), \tag{5.7}$$

where $\text{TF}(t, d)$ represents the frequency of t in d, N is the number of documents in D, and $\{\, d' \in D : t \in d'\}$ denotes the set of documents containing. Terms that appear frequently in a limited number of papers are given higher weight by this formula, which makes them more representative of the content of those documents. In contrast, words that are extensively used (such as articles or stop words exclusive to a given domain) have lower weights and have less of an effect on the representation.

For the product descriptions in the `DimProduct` table, common stop words were eliminated, and text was normalized (converted to lowercase, removed from punctuation). The specific preprocessing stages depend on linguistic and domain-specific requirements, however, stemming or lemmatization procedures could be implemented to optionally reduce morphological variability. Each product description was tokenized into terms after normalization, and Equation (5.7) was used to convert these terms into TF-IDF vectors. This conversion produces sparse, high-dimensional vectors that are ideal for tasks involving similarity or clustering and efficiently capture distinctive textual properties.

The primary benefit of TF-IDF is derived from it being able to bring attention to unique noticable terms of a product while minimizing the importance of ubiquitous terms. This design improves the representational quality of textual descriptions, leading to more precise assessments of similarity and superior subsequent recommendations. Sparse representations also facilitate efficient computing, which is beneficial for extensive e-commerce catalogs.

As depicted in Figure 5.4, each product undergoes a text-cleaning process, tokeniza-

tion, and TF-IDF weight assignment, resulting in a numeric representation conducive to effective clustering and search. The next section illustrates how these TF-IDF vectors feed into a KMeans clustering model, enabling the grouping of semantically similar products into coherent clusters that serve as the basis for content-based recommendations.

5.2.2 KMeans Clustering Implementation

We utilized the KMeans clustering algorithm for grouping products into separate clusters in order to generate the TF-IDF vector representation. Because of its computational efficiency and ease of use, K-means is one of the most widely used clustering algorithms in data mining and ML. Data points are separated into k distinct clusters using this method, each of which is defined by the mean (centroid) of its points. K-Means aims to minimize overall within-cluster variance, also known as the sum of squared errors or within-cluster sum of squares (WCSS), by iteratively updating the points allocated to clusters and the centroids' locations.

Formally, if $S_1, S_2, \ldots, S_k$ denote the k clusters, then K-Means seeks to solve:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2, \tag{5.8}$$

where $\mathbf{x}$ represents a data point, $\boldsymbol{\mu}_i$ is the centroid (mean vector) of the cluster S_i, and k is the prescribed number of clusters. The algorithm typically proceeds in the following iterative manner:

1. **Initialization:** Choose k initial centroids, usually by random sampling from the dataset.

2. **Assignment Step:** By applying a selected distance metric (which is typically Euclidean), assign each data point to the cluster whose centroid it is closest to.

3. **Update Step:** Take the mean of all the points that have been allocated to each cluster to recalculate its centroid.

4. **Repeat:** Switch between the assignment and update phases until convergence, which usually occurs when the cluster's assignment stops changing or reaches a maximum iteration threshold.

K-Means is a default clustering technique in many situations due to its computational efficiency and relative simplicity of use. In this study, we employed K-Means to cluster TF-IDF vectors derived from product descriptions. Each document vector $\mathbf{x}$ corresponds to the textual characteristics of a product, and the algorithm groups

products into semantically coherent clusters. After experimenting with different values of k, we fixed $k = 10$ to achieve an interpretable balance between cluster compactness and thematic coherence among products. The chosen value of k allowed different product categories to emerge within each cluster, simplifying the interpretation and visualization of the groups.

By mapping new items or keywords to the nearest cluster centroid, the algorithm successfully recognizes similar products, thereby allowing content-based recommendations [57]. This approach is not only scalable but also straightforward: items that share particular descriptive characteristics are more likely to lean toward the same cluster, improving the quality and reliability of recommendations in cases where user interaction data may be limited or scarce.

$$k = 10 \quad \Longrightarrow \quad \text{(Balance between interpretability and cluster cohesion)} \tag{5.9}$$

Throughout the procedure of assignment and update steps, Euclidean distance was utilized as the principal similarity measure, which is commonly used for textual feature vectors in TF-IDF format. Content-based recommendations, which concentrated on the cold-start problem by recommending newly released products even in the absence of any noteworthy feedback from customers or interactions, were based on the latter clusters.

5.2.3 Cluster Analysis and Interpretation

The interpretability of each cluster was enhanced by analyzing the top terms within them, obtained from their respective TF-IDF vectors. Table 5.4 illustrates representative terms for each cluster, facilitating understanding of their semantic content.

Table 5.4: Top Terms per Product Cluster

Cluster	Top Terms
0	pedestal, duet, drawer, washer, dryer, inch, metal, construction, oscillating, fan
1	definition, high, standard, 720p, 1080i, vga, 1080p, contrast, angle, 47
2	cubic, foot, capacity, watt, refrigerator, oven, microwave, 1100, sized, design
3	camera, digital, oz, compact, SLR, product, type, weight, mode, dimensions
4	inches, 15, pounds, 17, 10, fax, printer, 11, 23, jet
5	battery, rechargeable, hour, flip, swivel, degree, 180, screen, coffee, life
6	LCD, 16, GB, screen, widescreen, brightness, touch, control, aspect, ratio
7	wash, programs, dry, tumble, cleaning, rinse, temperature, washing, action, steam
8	home, power, watts, theater, cooking, channel, total, speakers, interior, defrosting
9	memory, usb, light, feet, air, rooms, conditioner, btu, square, channel

Each cluster was characterized by a set of top terms derived from the centroid's

highest TF-IDF values, providing clear interpretability of product groupings and improving user trust in recommendations.

5.2.4 Keyword-Based Recommendation

The ability to promptly provide recommendations based on a user-inputted term is an essential aspect of our content-based strategy. The trained KMeans model is then used to allocate the term to a cluster once it has first been transformed into its matching TF-IDF vector. In order to ensure that product recommendations are appropriate and in line with user intent, recommendations are then derived from inside this specified cluster.

Figure 5.5: Word Cloud of Product Names Weighted by Ratings

In Figure 5.5, frequently occurring and highly rated product-related terms are displayed more prominently, reflecting their relative significance in the corpus of item descriptions. Larger words, such as *"digital camera," "720p,"* and *"cubic foot,"* indicate popular or distinctive features that tend to appear in highly rated items. For instance, *"digital camera"* suggests a frequent and well-rated category of products focused on photography, whereas *"720p"* and *"LCD HDTV"* point to a group of television or display items that users have positively rated. The term *"cubic foot"* refers to capacity or volume, hinting at product categories such as refrigerators or microwaves with defining size attributes.

Crucially, this word cloud illustrates the variety of textual features that the TF-IDF representation captures. Terms related to consumer electronics (for example, *"hard drive," "HDMI," "widescreen,"* and *"touch screen"*) appear alongside more general concepts (for example, *"foot capacity," "rechargeable battery,"* and *"wireless"*). Such diversity indicates the system's ability to differentiate between product specifications and domains, ultimately helping in the *keyword-based recommendation* step. When a user inputs a keyword (for instance, *"camera"*), the content-based approach takes advantage of these TF-IDF-weighted terms to identify similar items within the same cluster, even if specific collaborative filtering data are lacking for those products.

Content-based filtering produced useful recommendations by meticulous textual data preparation, vectorization, and clustering; it was highly effective where there was none or limited previous user data. This strategy, which exhibits accuracy and versatility in conjunction with collaborative filtering approaches, is essential for tackling the cold-start problem associated with recommendation systems.

Chapter 6

Deployment

The deployment phase is a critical component of ML projects, involving the transition of developed predictive models and recommendation systems into practical, user-accessible services. The primary goal of this phase is to provide end users with actionable insights generated by complex analytical processes. This thesis presents a comprehensive deployment approach that uses FastAPI for API generation, Postman for meticulous API testing, and Gradio for intuitive and interactive user interface development [6][7][8]. These platforms and tools were specifically chosen for their robustness, ease of use, scalability, and compatibility with the Python programming ecosystem. This section provides detailed explanations and justifications for each technology used, along with custom methodologies applied to handle specific deployment requirements.

6.1 API Development with FastAPI

FastAPI is a cutting-edge, high-performance web framework for Python RESTful API development. Its choice was motivated by its superior performance, ease of use, automatic documentation capabilities, and excellent support for asynchronous programming, which collectively ensure robust and efficient API operations [6]. Furthermore, FastAPI integrates seamlessly with Pydantic for effective request validation, serialization, and error handling, significantly improving data integrity and API reliability. Three primary API endpoints were developed:

Prediction Endpoint (`/predict`): Provides predictions of sales quantities based on user-input transactional data.

Collaborative Filtering Endpoint (`/recommend/collaborative`): Offers personalized product recommendations leveraging collaborative filtering techniques.

Content-Based Filtering Endpoint (`/recommend/content-based`): Provides recommendations based on keyword input, using content clustering methods.

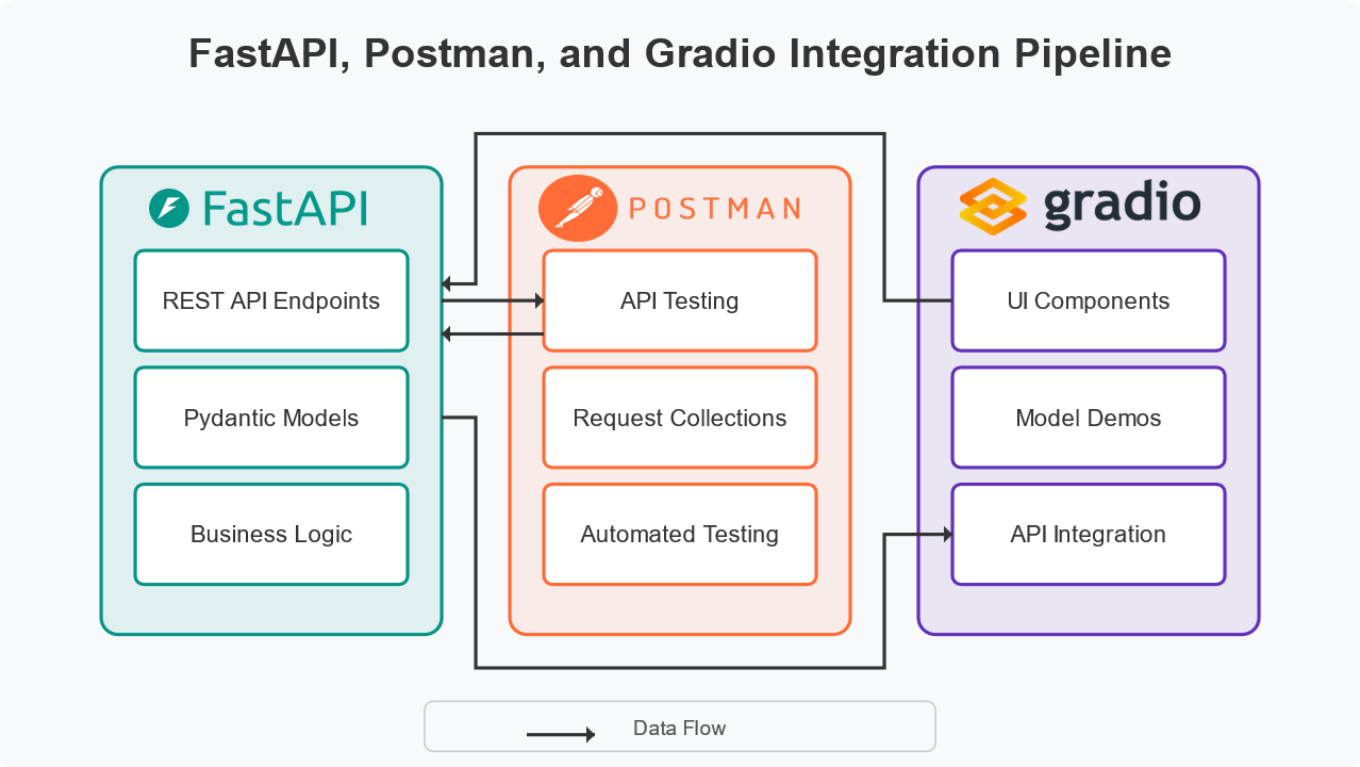

Figure 6.1: FastAPI, Postman, and Gradio Integration Pipeline

The Sales Prediction API includes specialized preprocessing steps distinct from the initial training preprocessing. These differences stem primarily from the need to handle the `RegionCountryName` field in real-time inference scenarios. Custom transformers, such as `DropAndDateTransformer` and `SinglePassTargetEncoder`, were developed to handle high-cardinality categorical variables effectively, ensuring that the model could perform accurate predictions in various geographical regions. The prediction pipeline was serialized using Joblib to enable rapid loading and execution during API calls.

Pipeline-Based Preprocessing with Custom Transformers

Making sure that the entire pre-processing procedure is reliable and repeatable is crucial in contemporary predictive modeling, particularly when working with high-dimensional data that comes from feature engineering and category encoding. Instead, we combined all data transformation processes into a single, cohesive *scikit-learn* pipeline, eliminating the need to manually construct a multitude of features (in our instance, 132) from raw input data. In terms of automation, maintainability, and consistency between the training and inference stages, this method has clear benefits.

Manually creating and maintaining an extensive array of features is not always

feasible when dealing with high-cardinality and multi-level categorical data. Any discrepancy between the feature engineering in the training phase and the process applied during inference risks leading to critical prediction errors. In contrast, encapsulating the entire workflow in a pipeline guarantees that the transformations applied to new (unseen) data match precisely those performed during training. Similarly, during the implementation of the sales prediction system, we encountered a critical mismatch between the training and inference stages: although the model was trained on a dataset with 132 encoded features, the API received only 18 unprocessed features at inference time. This discrepancy resulted in a runtime error indicating that the model expected 132 input features while being presented with far fewer. To address this, we adopted a unified *scikit-learn* pipeline that consolidates all data preprocessing steps into a single workflow, ensuring that inference data undergoes the same transformations as those used during training.

The deployed pipeline includes specialized preprocessing transformations like `DropAndDateTransformer` for date feature extraction like `DateKey` and dropping unnecessary columns, and `SinglePassTargetEncoder` for encoding high-cardinality categorical variables (e.g., `StoreKey`, `ProductKey`). The choice of these custom transformers enhances the model's predictive performance and allows accurate inference across various geographical regions.

By incorporating preprocessing steps and model training into a single pipeline, we streamline the workflow in three important ways. First, the pipeline accepts raw input data and applies all transformations automatically, eliminating the need for the manual construction of numerous encoded or scaled features. Second, consistency across training and inference is maintained through a unified procedure that minimizes the likelihood of data leakage or transformation errors. Third, saving the pipeline as a single object (for example, using `joblib`) substantially eases deployment: the same processing steps are reloaded at inference time, ensuring reproducible results.

In summary, the integration of custom transformers within a unified pipeline represents a robust and maintainable strategy for complex feature engineering in sales prediction tasks. This design choice not only simplifies the deployment process, particularly when interfacing with services like FastAPI or tools such as Postman and Gradio, but also enhances the reliability and performance of the predictive system as a whole.

6.2 API Testing with Postman

Postman is extensively employed in this project to test the robustness and accuracy of the API endpoints developed [7]. Its capabilities for detailed request-response

inspection, automated testing scripts, and widespread usage within the software development industry underpin its selection.

For the Prediction API, collaborative filtering, and content-based recommendation APIs, comprehensive testing scenarios were created and validated. The tests confirm endpoint responsiveness, prediction accuracy, and relevance of the recommendation. Example scenarios used during the tests are included in the documentation to demonstrate successful API interactions and the reliability of the deployed services [7].

The following are sample tests executed in Postman to verify API reliability and correctness:

Sales Prediction API Postman Test:

Request:


```
{
  "DateKey": "2009-03-04",
  "StoreKey": 307,
  "ProductKey": 1114,
  "channelKey": 2,
  "PromotionKey": 1,
  "RegionCountryName": "United States",
  "ProductCategoryKey": 4,
  "ProductSubcategoryKey": 24,
  "UnitCost": 153.59,
  "UnitPrice": 334.00,
  "ReturnQuantity": 0,
  "ReturnAmount": 0.00,
  "DiscountAmount": 0.0,
  "TotalCost": 4607.70,
  "SalesAmount": 10020.00,
  "DiscountPercent": 0
}
```


Response:


```
{"predicted_sales_quantity":13.956844577223194}
```


Collaborative Product Recommendation API Postman Test:

Request:

```
{
    "customer_id": 333
}
```

Response:

```
{
  "customer_id": 333,
  "recommended_products": [
    {
      "ProductKey": 1860,
      "Product": "NT Washer & Dryer 21in E2100 Blue"
    },
    {
      "ProductKey": 249,
      "Product": "Contoso Home Theater System 2.1 Channel E1220 Black"
    },
    {
      "ProductKey": 1817,
      "Product": "MGS MechCommander 2009 E173"
    },
    {
      "ProductKey": 1686,
      "Product": "SV Hand Games for students E40 Yellow"
    },
    {
      "ProductKey": 1668,
      "Product": "MGS Hand Games for students E400 Black"
    }
  ]
}
```

Content-Based Recommendation API Postman Test:

Request:

```
{"keyword": "photo"}
\end{minted}

\textbf{Response:}
\begin{lstlisting}[language=json]
{
  "keyword":"photo",
  "predicted_cluster":6,
  "top_terms":["definition","high","720p","standard","channel",
```

```
                "usb","1080i","watts","wash","coffee"],
  "recommended_products":[
    {
      "ProductKey":248,
      "ProductName":"Contoso Home Theater System 5.1 Channel M1530
    Black"
    },
    {
      "ProductKey":173,
      "ProductName":"SV 8xDVD E120 Black"
    },
    {
      "ProductKey":2106,
      "ProductName":"Contoso Water Heater 4.3GPM M1250 Grey"
    },
    {
      "ProductKey":914,
      "ProductName":"SV 160GB USB2.0 Portable Hard Disk M65 Grey"
    },
    {
      "ProductKey":1644,
      "ProductName":"Contoso DVD External DVD Burner M200 Blue"
    },
    {
      "ProductKey":232,
      "ProductName":"Litware Home Theater System 4.1 Channel M413 Brown
    "
    },
    {
      "ProductKey":1365,
      "ProductName":"Contoso 2-Line Corded Cordless Telephone M202
    White"
    },
    {
      "ProductKey":222,
      "ProductName":"Litware Home Theater System 5.1 Channel M515
    Silver"
    }
  ]
}
```


6.3 User Interface Development with Gradio

The thesis utilizes Gradio, a user-friendly interface building tool, chosen specifically for its ability to quickly prototype and showcase complex ML models through intuitive

and interactive web interfaces without extensive front-end development knowledge [8]. Gradio was instrumental in achieving the goal of making the developed models directly accessible and easily interpretable by users. The resulting Gradio application interfaces directly with the FastAPI endpoints and provides clear, immediate feedback for sales prediction and personalized recommendations [8].

The interface is structured into three distinct functional tabs:

1. **Sales Prediction**: Users can enter transaction-specific details and instantly receive a prediction for sales quantities.

2. **Collaborative Recommendations**: Users input their customer ID to receive personalized product suggestions.

3. **Content-Based Recommendations**: Users enter keywords to obtain relevant product recommendations based on content similarity.

To demonstrate how users interact with the deployed services, the Gradio interface provides separate tabs for sales prediction, collaborative filtering, and content-based filtering. In the sales prediction section, users can enter essential parameters such as date, store information, product details, country of sale, and cost/price figures. After receiving these inputs, the system computes and displays a predicted sales quantity in real time, employing a trained model and deployment-specific preprocessing steps.

In the collaborative recommendation section, users are prompted to input a customer ID. This input is used to generate a list of recommended products derived from analyzing purchasing patterns between multiple users. In contrast, the content-based recommendation section allows a user to submit a specific keyword, which is then compared against product descriptions through TF-IDF vectorization and KMeans clustering. The interface subsequently returns items that align the most closely with the user keyword, reflecting the system's capability to identify relevant products based on textual similarity.

Chapter 7

Conclusion

This thesis presented a comprehensive exploration of ML methodologies applied to sales forecasting and product recommendation problems within the retail sector. Utilizing the ContosoRetailDW dataset, we successfully developed, optimized, and deployed predictive models capable of accurately forecasting sales quantities and generating personalized product recommendations. The research contributes to practical industry applications, especially in handling the real-world complexities inherent in retail analytics.

We first carried out extensive exploratory data analysis (EDA) in the sales prediction field, which was followed by precise preprocessing steps. Notably, our models' predictive ability was greatly increased by the application of strategic feature engineering. The intricate correlations in retail sales data were successfully captured by the targeted approach to categorical variable encoding, specifically one-hot encoding for low cardinality variables and target encoding with cross-validation for high cardinality variables. Furthermore, the model's performance was further stabilized and enhanced by standard scaling of numerical attributes.

Numerous prediction algorithms, such as Ridge Regression, Random Forest, Gradient Boosting, XGBoost, and SVR, were methodically benchmarked and assessed. According to the first assessments, ensemble approaches had a lot of potential but were hampered by overfitting in the absence of adequate regularization. The resilience and prediction accuracy of ensemble models were greatly improved by thorough hyperparameter tweaking using GridSearchCV and cross-validation techniques, which finally resulted in the choice of XGBoost as the final model. With the lowest RMSE of 4.19, the optimized XGBoost model showed exceptional predictive accuracy, confirming its ability to handle complex nonlinear connections that are common in retail data.

Two different methods, Collaborative Filtering and Content-Based Filtering, were thoroughly applied and assessed for product recommendations. Using advanced

algorithms from the Surprise library, the collaborative filtering approach made use of past customer-product interactions. With the lowest RMSE of 0.2731 and the best ability to suggest appropriate items based on previous user behavior, KNNBaseline became the best model. Textual data preprocessing using KMeans clustering and TF-IDF vectorization allowed the content-based recommendation system to produce precise and significant recommendations, particularly in cases in which data was sparse or the cold-start problem was present.

The practical effectiveness and understanding of the models were greatly improved by combining these approaches into a single, FastAPI-powered RESTful API and by using Gradio to provide an easy-to-use, interactive user interface. Through an easily available web-based platform, stakeholders may now directly facilitate strategic and well-informed business decisions by utilizing the deployed models.

7.1 Future Works

The work described in this thesis can be expanded upon in the future by investigating a number of promising directions. First, adding real-time streaming data may improve the prediction models' ability to adapt to changing consumer trends and dynamic market situations. Retail organizations could receive instant, actionable information by using real-time analytics platforms like Apache Spark Streaming or Apache Kafka.

Additionally, investigating deep learning methods like Transformer-based models, Long Short-Term Memory (LSTM) networks, and Recurrent Neural Networks (RNN) may reveal more intricate temporal patterns and correlations in sales data. These techniques could greatly improve prediction accuracy, especially in cases with noticeable temporal dynamics, even if they demand bigger datasets and more processing power.

The creation of hybrid recommendation systems that combine collaborative and content-based methodologies to use the advantages of both approaches is another interesting area of research. Furthermore, adding contextual information like real-time feedback could greatly improve the recommendations' accuracy and customization.

Lastly, addressing deployment environments' scalability and efficiency is a crucial area for further study. Utilizing containerization technologies (like Docker and Kubernetes) and cloud services (like AWS, Azure, and Google Cloud) will enable scalable, dependable deployments, guaranteeing the smooth and effective functioning of ML models in production.

In conclusion, this thesis has offered a thorough methodological framework together with real-world applications of cutting-edge ML techniques for product recommendations and sales forecasting. In addition to achieving notable improvements

in prediction performance, the provided research presents an important foundation for future advancements in data-driven decision-making in the retail industry by filling in existing gaps in the literature and addressing practical problems.

Bibliography

[1] *ContosoRetailDW Dataset*. 2024 (cit. on pp. 6, 9).

[2] SQLBI. *Contoso Data Generator*. Last Accessed: 2025-03-17 (cit. on pp. 6, 9).

[3] *NumPy Documentation*. Last Accessed: 2025-03-17 (cit. on p. 10).

[4] *Pandas Documentation*. Last Accessed: 2025-03-17 (cit. on p. 10).

[5] *Microsoft SQL Server Documentation*. Last Accessed: 2025-03-17 (cit. on p. 9).

[6] *FastAPI Documentation*. Last Accessed: 2025-03-17 (cit. on pp. 12, 46).

[7] *Postman API Testing*. 2024 (cit. on pp. 46, 48, 49).

[8] *Gradio Official Documentation*. Last Accessed: 2025-03-17 (cit. on pp. 13, 46, 52).

[9] *Surprise Library for Recommender Systems*. Last Accessed: 2025-03-17 (cit. on pp. 11, 36).

[10] *Visual Studio Code Documentation*. Last Accessed: 2025-03-17 (cit. on p. 9).

[11] *Joblib Documentation*. 2025. URL: `https://joblib.readthedocs.io/` (cit. on p. 11).

[12] *XGBoost Documentation*. Last Accessed: 2025-03-17. URL: `https://xgboost.readthedocs.io/` (cit. on pp. 8, 22).

[13] *Support Vector Regression*. Last Accessed: 2025-03-17. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html`.

[14] *Gradient Boosting Documentation*. Last Accessed: 2025-03-17. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html`.

[15] *Random Forest Regressor*. Last Accessed: 2025-03-17. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html` (cit. on p. 8).

[16] *Ridge Regression*. Last Accessed: 2025-03-17. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html` (cit. on p. 8).

[17] *Jupyter Notebook Documentation*. Last Accessed: 2025-03-17. URL: https://jupyter-notebook.readthedocs.io/en/stable/ (cit. on p. 9).

[18] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021 (cit. on p. 12).

[19] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[20] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[22] *Requests: HTTP for Humans*. Last Accessed: 2025-03-17. URL: https://requests.readthedocs.io/en/latest/ (cit. on p. 12).

[23] *Pydantic*. Last Accessed: 2025-03-17. URL: https://docs.pydantic.dev/latest/ (cit. on p. 12).

[24] Python Software Foundation. *Python Foundations*. Accessed: 2025-03-18. URL: https://docs.python.org/3/tutorial/index.html#tutorial-index (cit. on p. 9).

[25] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997 (cit. on p. 7).

[26] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014 (cit. on pp. 7, 8).

[27] D.P. Kroese, Z.I. Botev, T. Taimre, and R. Vaisman. *Data Science and Machine Learning: Mathematical and Statistical Methods*. Chapman & Hall/CRC machine learning & pattern recognition. Boca Raton: CRC Press, 2019. ISBN: 9781138492530. URL: https://people.smp.uq.edu.au/DirkKroese/DSML/ (cit. on pp. 3, 22, 38).

[28] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012. ISBN: 9780262018258. URL: https://books.google.it/books?id=maz6AQAAQBAJ (cit. on p. 22).

[29] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020 (cit. on pp. 3, 14, 38).

[30] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[31] R.J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018. ISBN: 9780987507112. URL: https://books.google.it/books?id=_bBhDwAAQBAJ.

[32] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[33] *sklearn.preprocessing.TargetEncoder*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.TargetEncoder.html (visited on 03/17/2025) (cit. on p. 19).

[34] *sklearn.preprocessing.OneHotEncoder*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder (visited on 03/17/2025) (cit. on p. 17).

[35] J. H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29 (2001), pp. 1189–1232 (cit. on pp. 8, 22).

[36] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: Aug. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785 (cit. on pp. 8, 11, 22).

[37] L Breiman. "Random Forests". In: *Machine Learning* 45 (Oct. 2001), pp. 5–32. DOI: 10.1023/A:1010950718922 (cit. on pp. 8, 22).

[38] Jeff Heaton. "An empirical analysis of feature engineering for predictive modeling". In: *SoutheastCon 2016*. 2016, pp. 1–6. DOI: 10.1109/SECON.2016.7506650.

[39] Bohdan Pavlyshenko. "Using Stacking Approaches for Machine Learning Models". In: *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*. 2018, pp. 255–258. DOI: 10.1109/DSMP.2018.8478522 (cit. on p. 3).

[40] Shaohui Ma and Robert Fildes. "Retail sales forecasting with meta-learning". In: *European Journal of Operational Research* 288.1 (2021), pp. 111–128. DOI: 10.1016/j.ejor.2020.05.03. URL: https://ideas.repec.org/a/eee/ejores/v288y2021i1p111-128.html (cit. on p. 3).

[41] Waddah Saeed. "Frequency-based ensemble forecasting model for time series forecasting". In: *Computational and Applied Mathematics* 41 (Mar. 2022), p. 66. DOI: 10.1007/s40314-022-01765-x.

[42] D.C. Montgomery, C.L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. Wiley Series in Probability and Statistics. Wiley, 2015. ISBN: 9781118745151. URL: https://books.google.it/books?id=Xeh8CAAAQBAJ.

[43] Sahith Krishna. *12 Sales Forecasting Methods You Need for Accurate Predictions*. Last Accessed: 2025-03-17. 2023. URL: https://www.luru.app/post/sales-forecasting-methods.

[44] Diego Monti and Giuseppe Rizzo. "A systematic literature review of multicriteria recommender systems". In: *Artificial Intelligence Review* 54 (Jan. 2021). DOI: 10.1007/s10462-020-09851-4.

[45] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42 (2009), pp. 30–37 (cit. on pp. 8, 35).

[46] KAREN JONES. "Jones, K.S.: A Statistical Interpretation of Term Specificity and its Application in Retrieval. Journal of Documentation 28(1), 11-21". In: *Journal of Documentation* 28 (Dec. 1972), pp. 11–21. DOI: 10.1108/eb026526 (cit. on p. 8).

[47] Karen Spärck Jones. "A Statistical Interpretation of Term Specificity and its Application in Retrieval". In: *Journal of Documentation* 60.5 (2004). Accessed: 2025-03-17, pp. 493–502. URL: https://www.cl.cam.ac.uk/archive/ksj21/ksjdigipapers/jdoc04.pdf (cit. on pp. 4, 41).

[48] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. "Collaborative Filtering and Deep Learning Based Recommendation System For Cold Start Items". In: *Expert Systems with Applications* 69 (Oct. 2016). DOI: 10.1016/j.eswa.2016.09.040 (cit. on p. 32).

[49] Elodie Bouzekri, Alexandre Canny, Camille Fayollas, Célia Martinie, Philippe A. Palanque, Eric Barboni, Yannick Déléris, and Christine Gris. "A List of Pre-Requisites to Make Recommender Systems Deployable in Critical Context". In: *EnCHIReS@EICS*. 2017. URL: https://api.semanticscholar.org/CorpusID:51957059.

[50] Samy Baladram. *Encoding Categorical Data Explained: A Visual Guide with Code Example for Beginners*. Accessed: 2025-03-20. 2024. URL: https://towardsdatascience.com/encoding-categorical-data-explained-a-visual-guide-with-code-example-for-beginners-b169ac4193ae/ (cit. on p. 18).

[51] Xuan Lam, Thuc Vu, Trong Le, and Duong Duc. "Addressing cold-start problem in recommendation systems". In: Jan. 2008, pp. 208–211. DOI: 10.1145/1352793.1352837 (cit. on p. 32).

[52] Poonam B.Thorat, R. Goudar, and Sunita Barve. "Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System". In: *International Journal of Computer Applications* 110 (Jan. 2015), pp. 31–36. DOI: 10.5120/19308-0760 (cit. on p. 32).

[53] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. "Collaborative Filtering Recommender Systems". In: *Found. Trends Hum.-Comput. Interact.* 4.2 (Feb. 2011), pp. 81–173. ISSN: 1551-3955. DOI: 10.1561/1100000009. URL: https://doi.org/10.1561/1100000009 (cit. on p. 33).

[54] Jon Herlocker, Loren Terveen, John C.s Lui, and T. Riedl. "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems* 22 (Jan. 2004), pp. 5–53. DOI: 10.1145/963770.963772 (cit. on p. 15).

[55] F. Ricci, L. Rokach, and B. Shapira. *Recommender Systems Handbook*. 2nd. Springer, 2015 (cit. on pp. 4, 32, 33).

[56] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008 (cit. on pp. 4, 8).

[57] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. "Item-based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of ACM World Wide Web Conference* 1 (Aug. 2001). DOI: 10.1145/371920.372071 (cit. on pp. 4, 33, 43).

[58] Robin Burke. "Hybrid Recommender Systems: Survey and Experiments". In: *User Modeling and User-Adapted Interaction* 12 (Nov. 2002). DOI: 10.1023/A:1021240730564 (cit. on p. 4).

[59] Paul Resnick and Hal R. Varian. "Recommender systems". In: *Communications of The ACM* 40 (1997), pp. 56–58. URL: https://api.semanticscholar.org/CorpusID:265885880.