Sales and Customer Insights Web Application

SURYA SRI SUNDARA

3147289

Submitted in partial fulfilment for the degree of

Master of Science in Big Data Management and Analytics

Griffith College

Sep, 2025

Under the supervision of John Hannon

Disclaimer

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Master of Science in Big Data Management and Analytics at Griffith College Dublin, is entirely my own work and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed:surya sri sundara **Date:07-09-2025**

Acknowledgement

I would like to express my sincere gratitude to my supervisor, John Hannon, for his valuable guidance, constructive feedback, and continuous support throughout the course of this project.

I am also thankful to the faculty and staff of Griffith College for providing the academic resources and learning environment that enabled me to successfully undertake this research.

I would like to extend my appreciation to my family for their constant encouragement and understanding, as well as to my friends and classmates who offered helpful discussions and feedback during the development of the application.

Finally, I acknowledge the use of publicly available datasets and community-driven resources, which contributed significantly to the completion of this work.

Contents

Table of Equations

*This page summarises the mathematical formulas implemented in the **Sales & Customer Insights Web Application**.*

Equation #	**Description**	**Reference in Text**
Eq. (5.1)	Marketing ROI: $\text{ROI} = \frac{\text{Net Profit}}{\text{Marketing Cost}} \times 100$.	Used in Marketing ROI module.
Eq. (5.2)	Percentage change: $\%\ \text{Change} = \frac{\text{Forecast}-\text{Previous}}{\text{Previous}} \times 100$.	Compares current vs. previous period.
Eq. (4.1)	Synthetic marketing spend: $\text{MS}_i = \frac{S_i}{\sum_{j\in g} S_j}\Big(\sum_{j\in g} S_j \cdot \text{BaseProp} \cdot \text{Season}_g \cdot \text{Region}_g \cdot (1 + 1.2\,\overline{\text{Disc}}_g)\Big)$.	Generates realistic spend for regression.
Eq. (5.3)	Inventory requirement: $\text{Required Units} = \text{Forecast} \times (1 + \text{Safety Stock}\,\%)$.	Used in inventory simulation.

Table of Algorithms

*This page summarises the algorithms implemented in the **Sales & Customer Insights Web Application**.*

Alg. #	Description	Reference in Text
Alg. 1	Comparison of ARIMA and Prophet forecasting models for sales trends, producing side-by-side visual forecasts.	`/compare-models` route.
Alg. 2	K-Means Clustering to group customers into segments based on purchasing patterns and demographics.	`/segments` route.
Alg. 3	Monte Carlo Simulation for inventory forecasting, simulating daily demand, stockouts, and reordering policies with safety stock and service level targets.	Inventory simulation module.
Alg. 4	Reorder Point (ROP) calculation using mean demand, standard deviation, and z-score from the target service level.	Inventory simulation logic.
Alg. 5	Regression-based sales prediction from marketing spend, discount, and region features, using a trained model.	`/predict` route.
Alg. 6	Linear trend forecasting for subcategory sales using historical sales data and simple regression.	`/forecast-subcat` route.

List of Figures

List of Tables

Abstract

This project presents the design and implementation of the **Sales and Customer Insights Web Application**, a data analytics tool that empowers retail businesses to make informed, evidence-based decisions. Built with *Python Flask* and *Bootstrap*, the application utilises the *Superstore* dataset—extended with realistic synthetic data—to transform large, complex sales records into actionable insights.

Core features include sales forecasting (overall and sub-category) using ARIMA and Prophet models, customer segmentation via K-Means clustering, marketing ROI analysis, inventory simulation, region-specific recommendations, and top-selling product reports. These capabilities help users anticipate demand, optimise marketing spend, identify profitable customer segments, and improve stock management.

The project demonstrates how statistical forecasting and machine learning can be integrated into an intuitive, lightweight platform, turning raw data into strategic business intelligence for retail decision-making.

Project Overview

This project has been a significant part of my Master's degree, combining technical skills, independent learning, and real-world application. It involved taking the initial concept of a **Sales and Customer Insights Web Application** and developing it into a fully functional platform through research, technical exploration, and implementation.

The system, built with Flask and Python-based machine learning workflows, uses the extended *Superstore* dataset to deliver forecasting (ARIMA, Prophet), customer segmentation (K-Means), marketing ROI analysis, inventory simulation, and region-specific recommendations. Its aim is to transform large, fragmented sales datasets into clear, actionable insights for retail decision-making without the need for expensive enterprise software.

Technologies and methods were chosen deliberately for their suitability, interpretability, and ease of integration. The application features an intuitive, role-based interface and supports both overall and sub-category forecasting. While dependent on data quality and limited to relatively simple segmentation models, it provides a lightweight, browser-based analytics solution for small-to-medium retail contexts.

The work demonstrates the integration of statistical forecasting, machine learning, and interactive visualisation into a single tool, documented clearly for both technical and non-technical audiences. It also reflects critical thinking in method selection, evaluation of strengths and limitations, and consideration of future enhancements such as real-time data ingestion and advanced forecasting models.

The planned layout is as follows:

1. **Introduction** — Presents the motivation for the project, its objectives, scope, and significance in the context of retail analytics.

2. **Background**

   (a) *Literature Review* — Reviews forecasting models such as ARIMA and Prophet, customer segmentation techniques like K-Means clustering, and relevant web application frameworks.

   (b) *Similar Work* — Examines comparable retail analytics platforms and academic projects, identifying gaps this project aims to address.

3. **Methodology** — Describes the approach to system development, dataset preparation (including synthetic data extension), choice of machine learning models, and evaluation methods.

4. **System Analysis and Design** — Details the functional and non-functional requirements, system architecture, role-based access design, and database schema.

5. **Implementation** — Explains the development of the Python Flask application, integration of forecasting models, customer segmentation module, ROI analysis, and inventory simulation.

6. **Testing and Evaluation** — Presents the results of system testing, performance of forecasting models, accuracy of segmentation, and usability evaluation.

7. **Conclusions and Future Work** — Summarises the achievements, discusses the strengths and limitations of the system, and proposes potential enhancements such as real-time data integration and more advanced modelling techniques.

The document concludes with:

- **Bibliography** — A comprehensive list of academic papers, books, and online resources referenced during the research and development process.

- **Appendices** — Supporting materials including full application code listings, configuration files, additional screenshots, and extended evaluation results.

Chapter 1

Introduction

This chapter provides an overview of the **Sales & Customer Insights Web Application** project, including its purpose, goals, and the high-level approach taken to develop it. It aims to give the reader a clear understanding of *what* the project is, *why* it was undertaken, and a brief indication of *how* it was implemented. The technical details and methodologies will be covered in subsequent chapters.

1.1 Retail Analytics and Decision Support

The project focuses on the domain of retail analytics, specifically on enabling data-driven decision-making through a lightweight, web-based decision-support system. The application is designed to transform sales data into clear, actionable insights for retail managers and analysts. It addresses common business challenges such as predicting future sales trends, identifying profitable customer segments, evaluating marketing performance, and optimising inventory levels.

The foundation of the system is the widely used *Superstore* dataset, which has been extended with realistic synthetic data to simulate recent sales trends. This enhanced dataset allows for the development and evaluation of forecasting, segmentation, and recommendation features in a way that mirrors real-world retail operations.

From a functional perspective, the application provides:

- Forecasting of overall and sub-category sales using ARIMA and Prophet models.

- Customer segmentation through K-Means clustering.
- Marketing ROI analysis to assess the impact of promotional spend.
- Inventory simulation to estimate stock requirements and reduce stockouts.
- Region-specific product recommendations and top-seller analysis.

1.2 Goals and Objectives

The primary goal of this project is to build a functional, accessible, and user-friendly analytics platform that can support retail decision-making without requiring advanced technical skills from its end users. The motivation for undertaking this work stems from the increasing importance of data analytics in retail and the lack of affordable, accessible tools for small to medium-sized businesses.

The specific objectives of the project are to:

- Integrate machine learning models for sales forecasting and customer segmentation into a single web-based interface.
- Provide role-based access for administrators, managers, and analysts to ensure secure and relevant access to functionality.
- Present analytical outputs (forecasts, segment summaries, ROI calculations) in a visually clear and actionable format.
- Allow users to explore both high-level business trends and detailed subcategory insights.
- Demonstrate the feasibility of using open-source technologies to build practical business analytics tools.

The intended beneficiaries of this work include retail managers seeking to optimise marketing spend and inventory, analysts looking to identify patterns in sales data, and small business owners wanting to adopt data-driven strategies without high licensing costs.

1.3 Overview of Approach

The application was developed using the *Python Flask* framework to provide a lightweight yet powerful backend capable of integrating machine learning models with interactive web features. The frontend was implemented using *Bootstrap* for responsive design, ensuring usability across devices.

Sales forecasting functionality was implemented using two complementary models: ARIMA for time-series trend analysis and Prophet for capturing seasonality and long-term patterns. Customer segmentation was achieved using K-Means clustering, chosen for its simplicity, interpretability, and speed in handling structured retail data.

The application incorporates role-based authentication and authorisation to ensure that only authorised users can access specific features. It also supports data visualisation through *Matplotlib*, enabling the display of forecasts, ROI charts, and segmentation results directly within the web interface.

While this chapter outlines the broad approach, detailed methodologies, system architecture, and implementation details will be discussed in later chapters. The project was designed to be modular, allowing for future enhancements such as real-time data ingestion, integration of additional forecasting algorithms, and customisable dashboards.

1.3.1 Layout

The remainder of this document is organised as follows:

Chapter Two presents the background to this project, including a literature review of retail analytics, forecasting techniques (ARIMA, Prophet), customer segmentation methods (K-Means clustering), and the role of web-based decision-support systems. It also discusses similar academic and commercial work in the field.

Chapter Three describes the methodology adopted for the project, outlining the approach to dataset preparation (including synthetic data extension), selection of machine learning models, role-based access design, and the overall development strategy using the Python Flask framework.

Chapter Four details the system analysis and design, including functional and non-functional requirements, system architecture, database schema, and user interface design. This chapter also specifies the hardware and software requirements for deployment.

Chapter Five provides the implementation details of the application, covering the integration of forecasting models, customer segmentation, marketing ROI analysis, inventory simulation, and data visualisation features into the Flask environment.

Chapter Six focuses on the testing and evaluation of the system. It describes the testing strategy for both functional and non-functional requirements, evaluates the accuracy of forecasting and segmentation models, and discusses usability testing feedback. Results are presented alongside any revisions made to improve the system.

Finally, **Chapter Seven** presents the conclusions drawn from this work and outlines potential future developments, such as integrating real-time data ingestion, adding more advanced forecasting algorithms, and enabling fully customisable dashboards for different business roles.

1.4 Related Work (Overview)

This chapter reviews the existing literature and related work relevant to the development of the **Sales & Customer Insights Web Application**. It introduces the theoretical foundations, current trends, and prior implementations in the areas of retail analytics, forecasting, customer segmentation, and web-based decision-support systems. By examining both academic research and industry applications, this chapter positions the project within the broader context of data-driven retail decision-making.

The first part of the chapter provides a discussion on the role of retail analytics in improving operational efficiency and strategic planning. It reviews forecasting techniques commonly applied in sales prediction, with a particular focus on the ARIMA (AutoRegressive Integrated Moving Average) and Prophet models. The strengths, weaknesses, and suitability of each approach for retail time-series data

are critically assessed.

Next, the chapter examines customer segmentation methods, with emphasis on the K-Means clustering algorithm. Literature on segmentation in retail contexts is reviewed to highlight how identifying customer groups based on purchasing patterns can inform targeted marketing and improve customer engagement.

The chapter also explores the enabling technologies for delivering analytics in an accessible format, particularly web-based platforms. It reviews the use of lightweight frameworks such as Flask for rapid development and integration with Python-based machine learning workflows, as well as Bootstrap for responsive, user-friendly interface design.

Finally, the chapter surveys related academic and commercial solutions, comparing their scope, features, and limitations with the proposed system. This comparison underscores the novelty of integrating forecasting, segmentation, marketing ROI analysis, inventory simulation, and visual reporting into a single, open-source, browser-based application tailored for small to medium-sized retail operations.

The literature and related work discussed here form the foundation for the methodological choices described in Chapter Three and inform the design and implementation decisions presented later in this report.

1.5 Literature Review

The development of the **Sales & Customer Insights Web Application** is informed by a range of research in the areas of retail analytics, sales forecasting, customer segmentation, and web application development. This section reviews relevant literature from scholarly journals, conference proceedings, textbooks, whitepapers, and official software documentation.

Retail Analytics and Decision Support Systems

Retail analytics involves applying statistical and machine learning methods to transactional data to improve decision-making. According to Waller and Fawcett (2013), analytics enables firms to optimise operations and enhance customer experiences through predictive and prescriptive insights. Recent studies (Chopra,

2019; Ahi et al., 2020) highlight the role of accessible decision-support systems in small to medium-sized enterprises (SMEs), where budget constraints limit access to enterprise-level analytics platforms. These works informed the project's focus on creating a lightweight, cost-effective tool deployable via a web interface.

Sales Forecasting Models: ARIMA and Prophet

Time-series forecasting is central to predicting sales trends. The ARIMA (AutoRegressive Integrated Moving Average) model, extensively discussed in Box et al. (2016), is a classical statistical approach known for handling short-term trends effectively. However, it requires careful parameter tuning and is sensitive to non-stationarity in data. Prophet, developed by Taylor and Letham (2018) at Facebook, offers a more automated approach, decomposing time series into trend, seasonality, and holiday effects, making it robust to missing data and outliers. Comparative studies (Ahmed et al., 2020) indicate that using multiple forecasting models can improve reliability, which justifies the inclusion of both ARIMA and Prophet in this project.

Customer Segmentation: K-Means Clustering

Customer segmentation groups customers based on shared characteristics, enabling targeted marketing strategies. K-Means clustering, as outlined by MacQueen (1967) and later refined by Lloyd (1982), remains one of the most widely used unsupervised learning algorithms due to its simplicity and efficiency. Research by Wedel and Kamakura (2012) shows that segmentation based on purchasing behaviour can significantly enhance customer lifetime value and engagement. This informed the decision to use K-Means for identifying distinct customer groups within the sales data.

Web Application Frameworks: Flask and Bootstrap

Delivering analytics via a web interface improves accessibility and usability. Flask, a lightweight Python web framework (Grinberg, 2018), allows rapid development and integration of machine learning models into interactive dashboards. Its flexibility and minimal overhead make it ideal for small-scale deployments, as sup-

ported by research on web-based data visualisation (Bostock, 2017). Bootstrap, developed by Twitter (Bootstrap Team, 2023), provides a responsive, mobile-first front-end framework, ensuring the application is accessible across devices.

Related Work

Several commercial and academic systems exist that aim to provide sales analytics, forecasting, and business intelligence. Well-established commercial platforms such as *Tableau*, *Microsoft Power BI*, and *Google Data Studio* offer highly customisable dashboards and a wide variety of visualisation options. These platforms are widely used in industry due to their integration capabilities and ease of use. However, they often require expensive licences for advanced features, have limited built-in forecasting capabilities, and rarely offer end-to-end solutions that integrate both machine learning models and domain-specific analytics such as inventory simulation or marketing ROI analysis.

Academic projects in the retail analytics space have often focused on a single aspect of the problem—for example, forecasting sales trends using time-series models (e.g., ARIMA, Prophet) or segmenting customers for targeted marketing. Research such as Žunić et al. (2020) demonstrates the effective use of Facebook's Prophet algorithm and backtesting strategies for retail sales forecasting in real-world scenarios Other comparative studies (Hasan et al., 2022; Chaturvedi et al., 2022) benchmark ARIMA against Prophet and other advanced models, highlighting the relative strengths of each approach in terms of accuracy and computational efficiency.

Customer segmentation has likewise been explored in retail contexts. Nugroho (2024) demonstrates the effectiveness of optimized K-Means clustering for segmenting customers in traditional retail environments. John et al. (2024) compare K-Means against other clustering algorithms such as GMM and DBSCAN, finding that while K-Means remains effective, certain contexts may benefit from more sophisticated approaches .

Delivering analytics via accessible interfaces is another area of active development. Miguel Grinberg's practical guide to Flask development underscores the framework's suitability for building lightweight, Python-based web applications

with integrated data visualisation and machine learning components .

The **Sales & Customer Insights Web Application** differentiates itself by unifying several analytics functions—sales forecasting (ARIMA and Prophet), customer segmentation (K-Means), marketing ROI analysis, inventory simulation, and product recommendations—into a single, web-based platform built using Flask and Bootstrap. Unlike many existing tools, it offers role-based authentication, does not rely on proprietary software, and is tailored for small-to-medium-sized retail operations.

By integrating forecasting, segmentation, and operational planning tools within one open-source, browser-based platform, this project addresses the fragmentation and accessibility issues found in current solutions. It equips decision-makers with predictive insights in a user-friendly environment, without requiring advanced technical skills or multiple disjoint tools.

Chapter 2

Methodology

2.1 Methodology Overview

This chapter describes the methodology adopted to design, develop, and deploy the **Sales & Customer Insights Web Application**. The focus is on the reasoning behind the system architecture, data flow, and technology choices, with emphasis on cost-effectiveness, scalability, and usability for non-technical retail managers and analysts.

The aim was to build a modular, interactive, and analytics-driven platform that could forecast sales, evaluate discount strategies, recommend products, analyse marketing ROI, and simulate inventory requirements — all from a unified dashboard.

2.2 Overall Approach

The development process followed an iterative, incremental model. Each feature (e.g., sales forecasting, discount impact analysis, product recommendations, marketing ROI, inventory simulation) was developed and tested independently before integration.

A **top-down approach** was used:

- High-level business requirements (e.g., "forecast future sales", "analyse ROI by region") were defined first.

- These were broken into subcomponents such as “select forecasting model”, “prepare dataset”, and “design visual outputs”.
- Development moved from these subcomponents into Flask routes, each responsible for handling input, running analytics, and rendering results.

This modular design enabled:

- Early testing and stakeholder feedback.
- Parallel development of different features.
- Reduced integration risks.

2.3 Technology Stack

Programming Language: Options considered included:

- **Python** — Strong ecosystem for data analysis, machine learning, and rapid prototyping; extensive library support (statsmodels, prophet, scikit-learn).
- **Java** — Strong for enterprise applications but heavier setup for data science workflows.
- **R** — Excellent for statistical modelling but less suited for web integration.

Decision: Python was chosen for its balance of data analytics capability and ease of web integration.

Web Framework: Options considered:

- **Flask** — Lightweight, modular, easy integration with custom ML scripts.
- **Django** — Full-stack features, built-in admin panel, but heavier for a small-scale prototype.
- **FastAPI** — Modern, high performance, but less mature ecosystem for templated dashboards.

Decision: Flask was selected for its flexibility and minimal overhead.

Frontend Framework / Styling: Options considered:

- **Bootstrap 5** — Wide adoption, responsive grid system, extensive documentation.
- **Tailwind CSS** — Utility-first, modern styling but requires more initial setup.
- **Materialize** — Google Material Design look and feel, but smaller community support.

Decision: Bootstrap 5 chosen for fast development and proven cross-device compatibility.

Database: Options considered:

- **SQLite** — Serverless, zero-configuration, ideal for prototyping.
- **PostgreSQL** — Advanced features, scalable, better for production environments.
- **MySQL** — Popular relational database, good performance, but more setup.

Decision: SQLite chosen for ease of deployment during development stage.

Forecasting Models: Options considered:

- **ARIMA** — Well-established statistical method for time series forecasting.
- **Prophet** — Handles seasonality and trend changes with minimal tuning.
- **SARIMA** — Seasonal ARIMA extension for periodic data.

Decision: ARIMA for short-term stability, Prophet for seasonal/irregular data; both used for model comparison.

Segmentation Models: Options considered:

- **K-Means** — Simple, efficient clustering algorithm.
- **DBSCAN** — Detects arbitrary-shaped clusters but sensitive to parameters.
- **Gaussian Mixture Models** — More flexible but computationally heavier.

Decision: K-Means chosen for interpretability and speed.

Dataset: Options considered:

- **Public Retail Datasets** (e.g., Superstore, UCI datasets).
- **Proprietary Corporate Data** — Richer but not accessible for academic work.

Decision: Superstore dataset extended with synthetic data (2019–2025) to avoid privacy concerns and simulate future trends.

2.4 Design Decisions

- **Role-Based Access Control (RBAC)** — Three roles: Administrator, Manager, Analyst. Access to pages/routes is role-dependent.
- **Unified Dashboard Design** — All analytics in one interface (forecasting, segmentation, ROI, product recommendations, discount impact) to reduce context switching.
- **Export Functionality** — All outputs (tables, charts) are exportable as CSV or PNG directly from the UI.
- **Deployability** — Application can run on local machines, lightweight servers, or cloud hosting (Gunicorn + Nginx).

2.5 Machine Learning Models (ARIMA, Prophet)

Model Comparison Workflow — Forecasts from ARIMA and Prophet are generated on the same time series, plotted together, and presented for user selection.

Inventory Simulation — Monte Carlo simulation models demand variability:

$$ROP = \mu_D \cdot L + z_\alpha \cdot \sigma_D \sqrt{L}$$

where:

- μ_D = mean daily demand
- σ_D = standard deviation of daily demand
- L = lead time in days
- z_α = service level quantile

Chapter 3

System Analysis and Design

3.1 System Architecture and Design

This chapter presents the system analysis and design of the **Sales & Customer Insights Web Application**. It explains the specific technologies selected for the project, the reasons for their adoption, the challenges encountered, and the theoretical system design, including architecture, process modelling, and data modelling.

3.1.1 Technologies Used

Python 3.11 Python was the primary programming language due to its strong ecosystem for data analysis, machine learning, and web development. Libraries such as `statsmodels` (for ARIMA), `prophet` (for Prophet forecasting), and `scikit-learn` (for K-Means clustering) were central to implementing the predictive and analytical features.

Flask (v2.x) Flask was chosen as the web framework for its lightweight, modular design, which allowed rapid development of the application without the overhead of a full-stack framework like Django. Flask's integration with Python scripts made it straightforward to embed forecasting and clustering models directly into the application routes.

Bootstrap 5 Bootstrap was used for front-end design to ensure the application interface was responsive and user-friendly across different devices. Its pre-built

components and grid system reduced the amount of custom CSS required, enabling faster UI development.

SQLite SQLite was selected as the database engine for storing user authentication details, role-based access information, and pre-processed sales data. It offered simplicity for deployment and did not require a separate database server, which suited the prototype stage of the project.

Matplotlib and Pandas Matplotlib was used for creating embedded visualisations (e.g., forecast graphs, segmentation charts), while Pandas was used extensively for data manipulation, cleaning, and transformation prior to analysis.

3.1.2 Rationale for Technology Choices

The combination of Flask, Python, and Bootstrap provided an ideal balance between backend flexibility, machine learning integration, and frontend usability. Alternatives were considered: - Django was rejected due to unnecessary complexity for the project scope. - Materialize and Tailwind CSS were considered for the front-end but lacked the wide adoption and built-in features of Bootstrap. - PostgreSQL or MySQL could be used for a production-ready system but were not necessary for a local, self-contained prototype.

3.1.3 Challenges and Limitations

One challenge was ensuring compatibility between `prophet` and Python versioning, as certain dependencies required specific versions of NumPy and Pandas. Additionally, generating realistic synthetic sales data for the years 2019–2025 required iterative testing to balance randomness with seasonality patterns. A limitation of the chosen stack is that SQLite is not optimal for high-concurrency, multi-user environments, meaning a production version would require migration to a more robust RDBMS.

3.1.4 Architecture Overview

The system follows a three-tier architecture:

- **Presentation Layer:** Implemented using HTML templates rendered by Flask and styled with Bootstrap.
- **Application Layer:** Flask routes handle HTTP requests, invoke forecasting or clustering scripts, and prepare data for visualisation.
- **Data Layer:** Data is stored and retrieved from CSV files and the SQLite database, with Pandas used for in-memory manipulation.

3.1.5 Process Modelling

The high-level process flow includes:

1. User logs in, and role-based access control determines accessible features.
2. User selects a task (e.g., forecasting overall sales, forecasting by sub-category, viewing segmentation results).
3. Input parameters (e.g., forecast period, category selection) are validated.
4. Corresponding machine learning model (ARIMA, Prophet, or K-Means) is executed with the prepared dataset.
5. Results are processed into tables and visualisations.
6. Outputs are displayed on the web interface, with options to download data as CSV.

3.1.6 Data Modelling

Inputs: - Historical sales data from the extended Superstore dataset (2014–2025). User-selected parameters (forecasting period, category selection).

Processing: - Data cleaning and transformation using Pandas.Forecast generation via ARIMA and Prophet.Clustering using K-Means.Marketing ROI and inventory simulation calculations.

Outputs: - Forecast tables and graphs.Segmentation summaries and charts.Marketing ROI results.Inventory requirement predictions.

This theoretical design will be supported by formal diagrams, including:

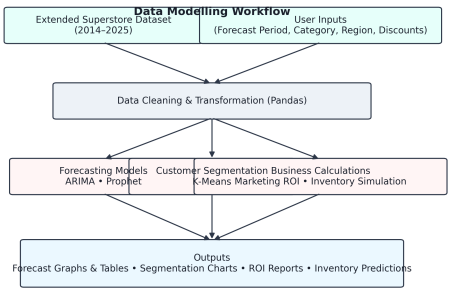

Figure 3.1: Data Modelling Workflow for the Sales and Customer Insights Web Application.

- A **System Architecture Diagram** showing the relationships between layers.
- UML **Use Case Diagrams** for core functionalities.
- An **Entity Relationship Diagram** (ERD) for database design.
- **Data Flow Diagrams** for key processes.

Chapter 4

Implementation

4.1 Application Overview

The implementation of the **Sales & Customer Insights Web Application** involved developing an integrated platform capable of forecasting sales, segmenting customers, analysing marketing ROI, and simulating inventory requirements, all within a secure, role-based web environment.

4.2 Core Flask Routes

The application was implemented in *Python 3.11* using the *Flask* framework for backend logic and routing, *Bootstrap 5* for frontend design, and *SQLite* for lightweight database storage. Core data processing and analysis were performed with `pandas`, while forecasting used `statsmodels` for ARIMA and `prophet` for Prophet models. Customer segmentation was implemented using the `scikit-learn` K-Means algorithm. Visualisations were generated using `matplotlib` and embedded directly in the web interface.

The implementation followed a modular structure:

4.2.1 Authentication Module

– Implements role-based access control, ensuring that only authorised users can access specific features.

4.2.2 Marketing spend

-It was derived as a proportion of monthly sales (typically 5–15), adjusted for seasonality, regional multipliers, and discount-driven promotional intensity, with small random noise to mimic real-world variability. This allows the regression model to simulate how marketing investment influences predicted sales.

4.2.3 Forecasting Module

– Provides both overall and sub-category sales forecasts using ARIMA and Prophet, allowing selection of forecast type and period.

4.2.4 Customer Segmentation Module

– Uses K-Means clustering to group customers by purchasing behaviour.

4.2.5 Marketing Analysis Module

– Calculates marketing ROI and provides performance metrics for campaigns.

$$\mathrm{MS}_i = \frac{S_i}{\sum_{j \in g} S_j} \left(\sum_{j \in g} S_j \cdot \mathrm{BaseProp} \cdot \mathrm{Season}_g \cdot \mathrm{Region}_g \cdot (1 + 1.2\,\overline{\mathrm{Disc}}_g) \right) \quad (4.1)$$

4.2.6 Inventory Simulation Module

– Estimates future stock requirements based on forecasted sales volumes.

4.3 Code Snippets

The code snippets shown here are selected examples of unique logic implemented for this project. Full code listings, including `app.py`, are uploaded in the one drive.

4.3.1 ARIMA Forecast

Listing 4.1: Example ARIMA Forecast Function

```
1 from statsmodels.tsa.arima.model import ARIMA
2 model = ARIMA(sales_series, order=(1, 1, 1))
```

```
model_fit = model.fit()
forecast = model_fit.forecast(steps=periods)
```

4.3.2 K-menas

Listing 4.2: Example K-Means Customer Segmentation

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(customer_data)
customer_data['Segment'] = clusters
```

4.3.3 Overall Forecast Route

Listing 4.3: Overall Forecast Route (Monthly/Yearly)

```
@app.route("/forecast", methods=["GET","POST"])
@login_required
@roles_required("manager")
def forecast():
    import base64, pickle
    from io import BytesIO, StringIO
    import pandas as pd
    import matplotlib.pyplot as plt

    selected_type = "monthly"   # or "yearly"
    periods = 6
    results, chart_b64, csv_b64, error = [], None, None, ""

    def fig_b64():
        buf = BytesIO()
        plt.savefig(buf, format="png", dpi=150, bbox_inches="tight")
        buf.seek(0)
        return base64.b64encode(buf.read()).decode("utf-8")

```

```
    if request.method == "POST":
        try:
            selected_type = request.form.get("forecast_type","monthly")
            periods = max(1, min(24, int(request.form.get("periods",6))))
            model_path = "sales_forecast_model_monthly.pkl" if selected_type=="monthly" \
                         else "sales_forecast_model_yearly.pkl"

            with open(model_path,"rb") as f:
                model = pickle.load(f)

            yhat = model.forecast(steps=periods)
            label = "Month" if selected_type=="monthly" else "Year"
            results = [{"label": f"{label} {i+1}", "value": float(v)} for i,v in enumerat

            df = pd.DataFrame(results)
            s = StringIO(); df.to_csv(s, index=False)
            csv_b64 = base64.b64encode(s.getvalue().encode()).decode()

            plt.figure(figsize=(8,3))
            plt.plot(range(1, periods+1), yhat, marker="o")
            plt.title(f"{selected_type.title()} Forecast")
            plt.xlabel(label); plt.ylabel("Sales")
            chart_b64 = fig_b64(); plt.close()
        except Exception as e:
            error = str(e)

    return render_template("forecast.html",
        selected_type=selected_type, periods=periods, results=results,
        chart_b64=chart_b64, csv_b64=csv_b64, error=error)
```

4.3.4 Discount Impact

Listing 4.4: Discount Impact Analysis Route

```
@app.route("/discount-impact", methods=["GET","POST"])
@login_required
@roles_required("manager","analyst")
def discount_impact():
    import pandas as pd, numpy as np
    import matplotlib.pyplot as plt
    from io import BytesIO, StringIO
    import base64

    def fig_b64():
        buf = BytesIO(); plt.savefig(buf, format="png", dpi=150, bbox_inches="tight")
        buf.seek(0); return base64.b64encode(buf.read()).decode()

    df = pd.read_csv("superstore_extended.csv")
    df["Order Date"] = pd.to_datetime(df["Order Date"])
    m = (df
         .assign(DiscountPct=lambda x: x["Discount"])
         .groupby(pd.Grouper(key="Order Date", freq="M"))
         .agg(Sales=("Sales","sum"), Orders=("Order ID","nunique"),
              AvgDisc=("DiscountPct","mean"))
         .reset_index())

    m["SMA6"] = m["Sales"].rolling(6).mean()
    corr = m[["Sales","AvgDisc"]].corr().iloc[0,1]

    plt.figure(figsize=(8,3))
    plt.plot(m["Order Date"], m["Sales"], marker=".", linestyle="-", label="Monthly Sales
    plt.plot(m["Order Date"], m["SMA6"], linestyle="--", label="6-Month SMA")
    plt.legend(); plt.title("Discount Impact | Sales vs SMA")
    plt.xlabel("Month"); plt.ylabel("Sales")
    chart_b64 = fig_b64(); plt.close()

    return render_template("discount_impact.html",
```

```
34         corr=float(corr), table=m.tail(12).to_dict("records"), chart_b64=chart_b64)
```

4.3.5 Product Recommendation

Listing 4.5: Product Recommendations by Region

```
1  @app.route("/recommend", methods=["GET","POST"])
2  @login_required
3  def recommend():
4      import pandas as pd
5      region = request.form.get("region","West")
6      topn = int(request.form.get("topn",3))
7
8      df = pd.read_csv("superstore_extended.csv")
9      rec = (df[df["Region"]==region]
10            .groupby("Sub-Category")["Sales"].sum()
11            .sort_values(ascending=False).head(topn).reset_index())
12
13     total_region_sales = float(df[df["Region"]==region]["Sales"].sum())
14     return render_template("recommend.html",
15         region=region, topn=topn,
16         items=rec.to_dict("records"),
17         total_region_sales=total_region_sales)
```

4.4 Export Utilities (CSV, PNG)

For tabular outputs (e.g., forecasts, segments), the routes generate a UTF-8 CSV string which is base64-encoded and presented as a downloadable link in the UI. For charts, Matplotlib figures are rendered to PNG in memory and likewise exposed as a `data:` URL. This enables users to archive results outside the application while ensuring the exported artefacts exactly match the on-screen views.

These snippets illustrate the integration of core algorithms into the Flask routes that process user input, execute the relevant model, and return results to the HTML templates. **Listing 4.X: CSV and PNG Export Utilities**

```
from io import BytesIO, StringIO
import base64
import matplotlib.pyplot as plt

def df_to_csv_b64(df):
    """Convert DataFrame to base64-encoded CSV string."""
    out = StringIO()
    df.to_csv(out, index=False)
    return base64.b64encode(out.getvalue().encode("utf-8")).decode("utf-8")

def fig_to_png_b64():
    """Render current Matplotlib figure as base64-encoded PNG."""
    buf = BytesIO()
    plt.savefig(buf, format="png", dpi=150, bbox_inches="tight")
    buf.seek(0)
    return base64.b64encode(buf.read()).decode("utf-8")
```

4.5 Security and User Roles, Screenshots

Screenshots are used in this chapter to demonstrate the working user interface and highlight key features of the application. Examples include:

- **Login Page** – The Admin Page serves as the control center for managing application settings, user accounts, and access permissions. It allows administrators to add, edit, or remove users, assign roles (e.g., Manager, Analyst), and monitor system activity.Additionally, it provides tools for dataset management, configuration updates, and maintaining overall system security.

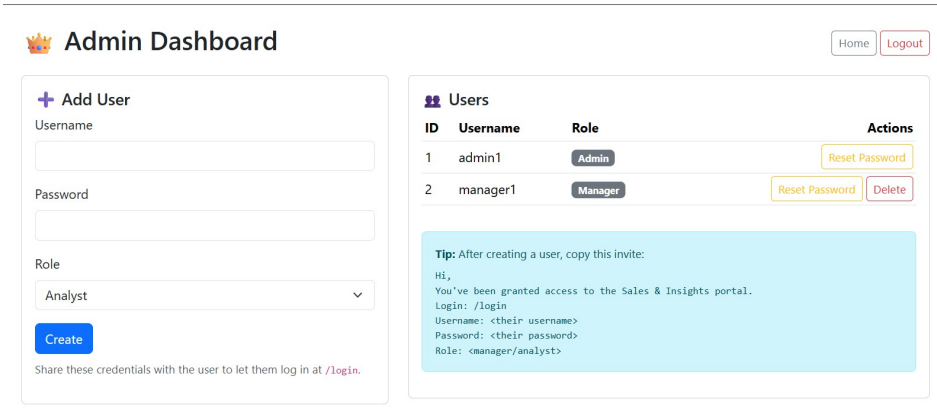

Figure 4.1: Admin Login Page

- **Manager Dashboard** -The Manager Dashboard offers a quick snapshot of sales, marketing spend, and ROI, highlighting best and worst-performing regions.It includes a bar chart comparing ROI by region and a table with detailed sales, spend, and ROI figures.

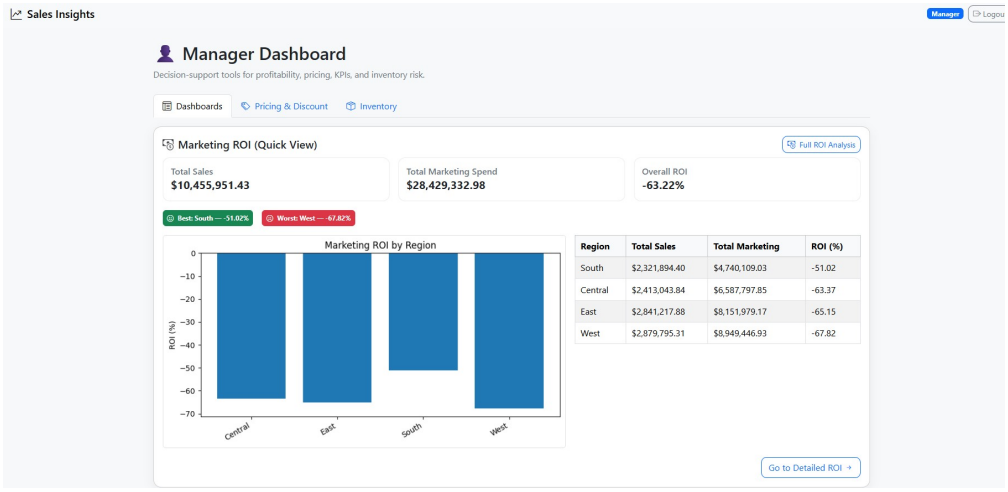

Figure 4.2: Manager Dashboard displaying regional sales, marketing spend, and ROI comparison.

- **Analyst Dashboard**-The Analyst Dashboard offers a feature set similar to the Manager Dashboard but with additional technical analytics capabilities aimed at deeper data interpretation. It provides high-level KPIs such as Total Sales, Total Marketing Spend, and Overall ROI, along with a breakdown

of best- and worst-performing regions. The Marketing ROI (Quick View) visualises ROI percentages by region in a bar chart, paired with a detailed table of sales, marketing spend, and ROI values to support comparative analysis.

Unlike the Manager Dashboard, the Analyst Dashboard integrates more advanced analytical tools on pages such as Discount Impact, where a trendline analysis and three-band Simple Moving Average (SMA) are applied to visualise sales trends, detect seasonal patterns, and identify deviations from the average performance. These added layers of analysis enable analysts to evaluate pricing and discount strategies with greater precision, linking statistical patterns directly to business outcomes.

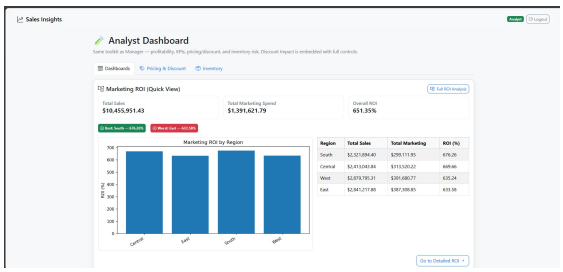

Figure 4.3: Analyst Dashboard displaying regional sales, marketing spend, and ROI comparison.

- **Marketing Spend** -The Marketing Spend page lets managers run what if scenarios by entering budget, discount, and region.A regression model predicts expected sales, displays inputs, and shows how categorical values are encoded.It helps assess the impact of marketing spend and discounts before campaigns launch.

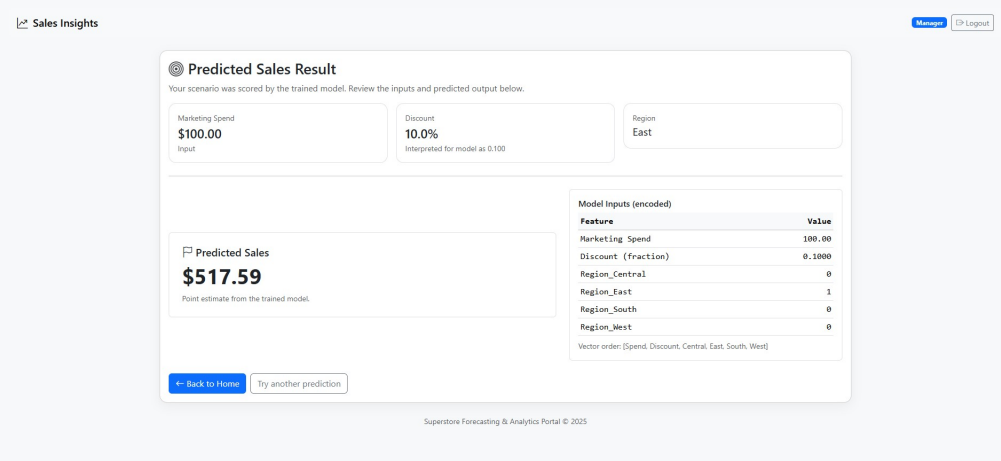

Figure 4.4: Predicted Sales Result page with inputs, encoded features, and model output.

- **Forecasting Page** – The Overall Forecasting page projects future sales trends using pretrained time series models. Managers can choose forecast periods (monthly or yearly) and view predicted values in both chart and table formats.It supports data export for further analysis and aids in strategic planning based on anticipated sales performance.

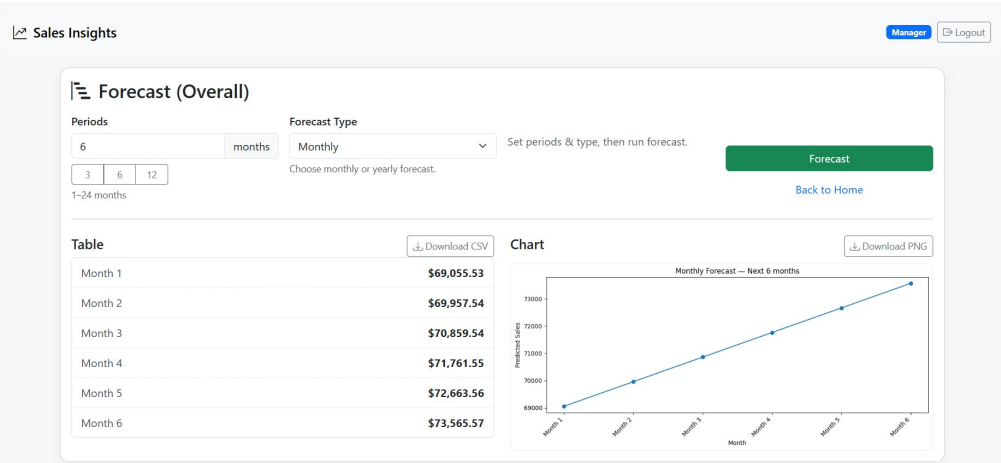

Figure 4.5: Forecasting Page with Monthly Sales Prediction

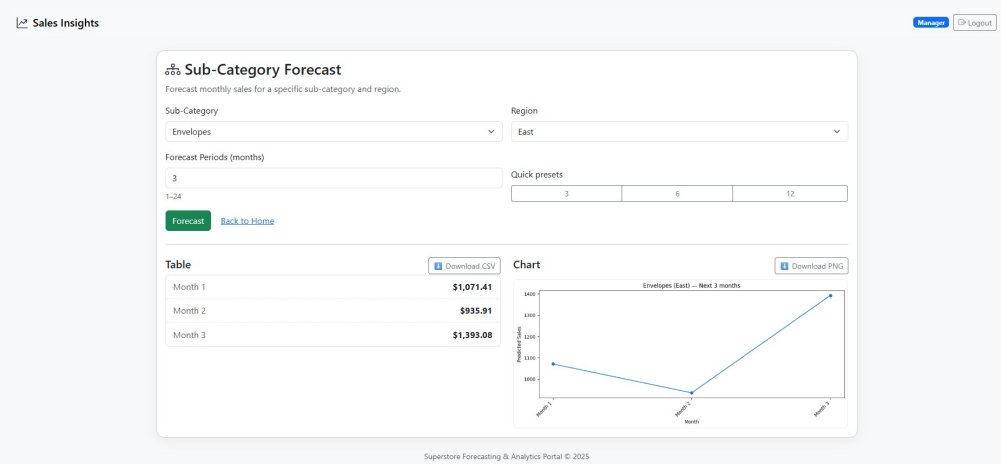

Figure 4.6: interface showing selection of SubCategory, forecast type (monthly/ yearly), horizon.

- **Monthly Sales Trend** - The Monthly Sales Trend page shows sales for a selected region with an optional rolling average (SMA) for smoothing. It includes interactive filters, a line chart of actual vs. smoothed sales, a recent months table, and download options for chart and data.It helps managers quickly visualize sales patterns, identify trends, and detect seasonality for informed decision-making.

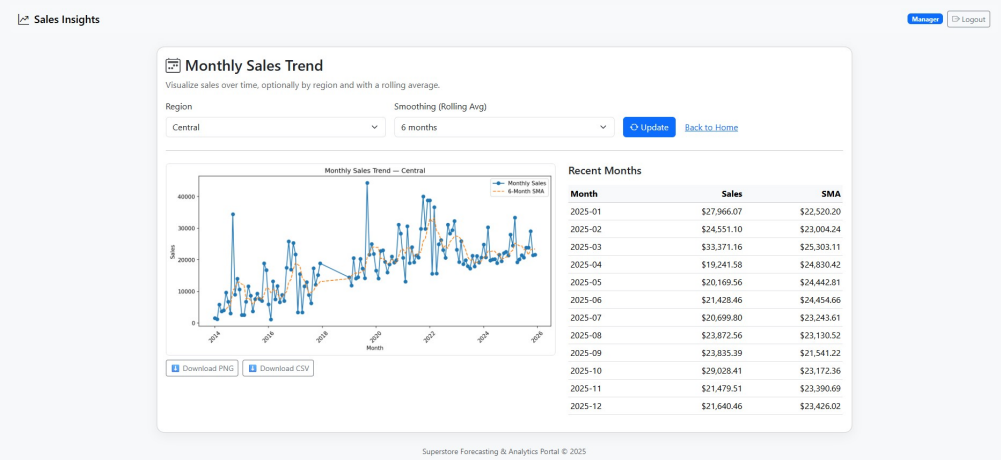

Figure 4.7: Monthly sales trend for the selected region with 6-month rolling average (SMA) showing seasonal patterns and overall growth.

- **Top Sub-Categories by region** -The Top Sub-Categories page ranks products by a chosen metric (Sales, Profit, or Quantity). It offers filters for metric, item count, and region, plus a bar chart, ranking table, and download options

for chart and data.This view helps managers identify best-selling product categories and make informed stocking, marketing, and pricing decisions.

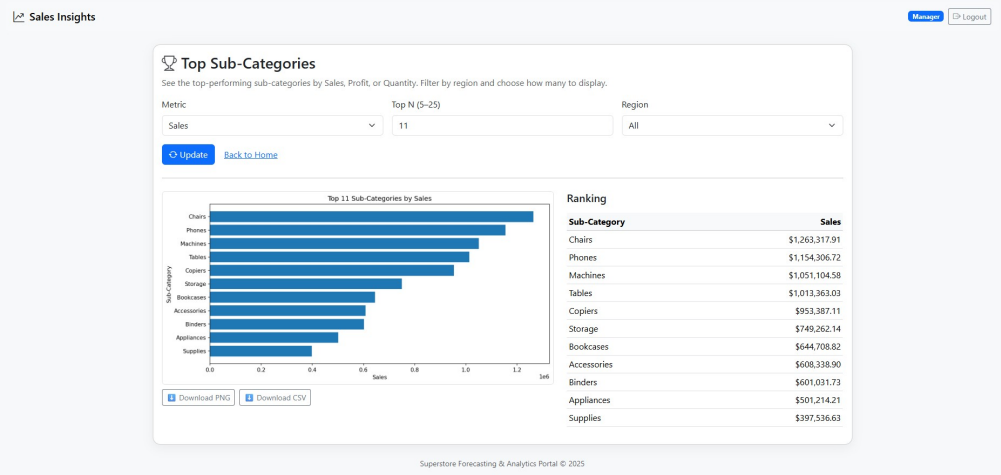

Figure 4.8: Top-performing sub-categories by sales, highlighting the highest revenue generating product groups.

- **Product Recommendations By Region** -The Product Recommendations by Region page highlights top-performing sub categories for a selected region. It features quick region selection, top 3 sub category highlights, total sales, a sales distribution chart, and download options for chart and data.

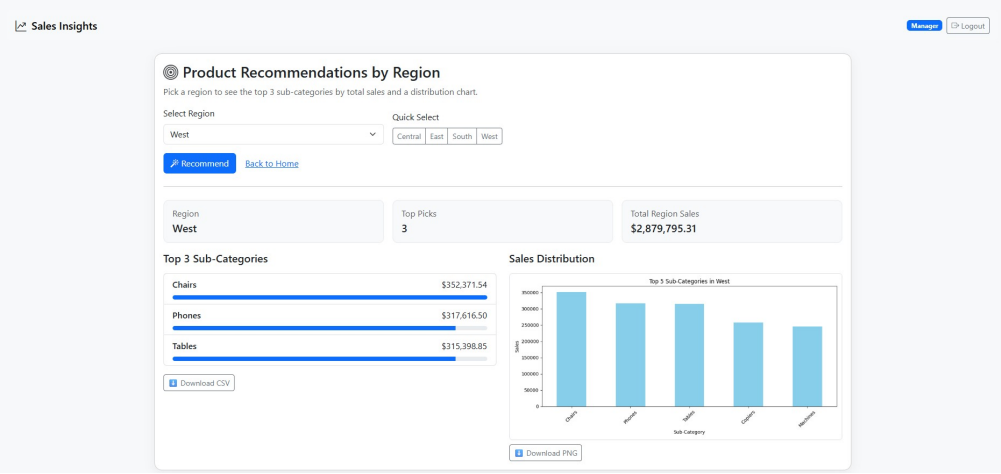

Figure 4.9: Product recommendations for the selected region showing top sub-categories by sales and their distribution.

- **Customer Segmentation Dashboard** – Visual representation of customer clusters.

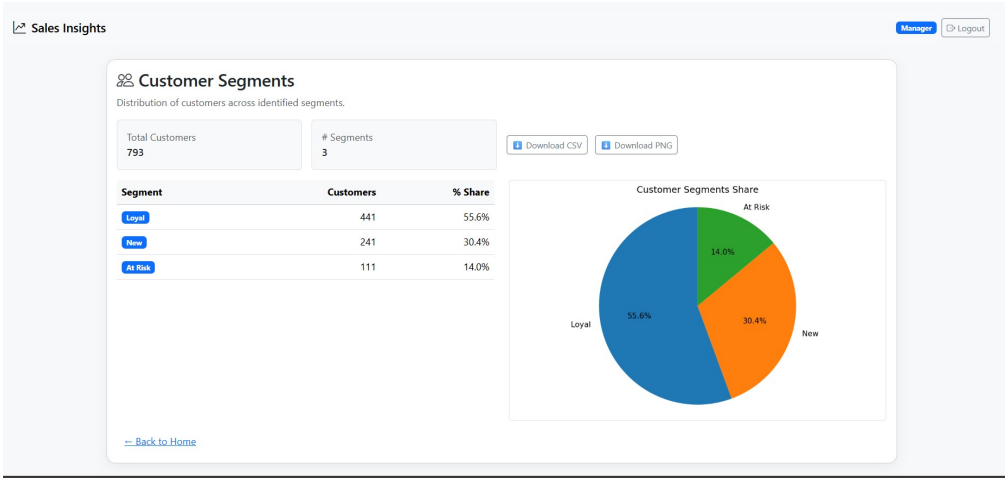

Figure 4.10: Customer Segmentation Dashboard showing clustered groups

- **Arima Vs Prophet**-ARIMA is statistically rigorous and better for stationary series, while Prophet is more flexible, automates seasonality detection, and is easier for real-world business forecasting.

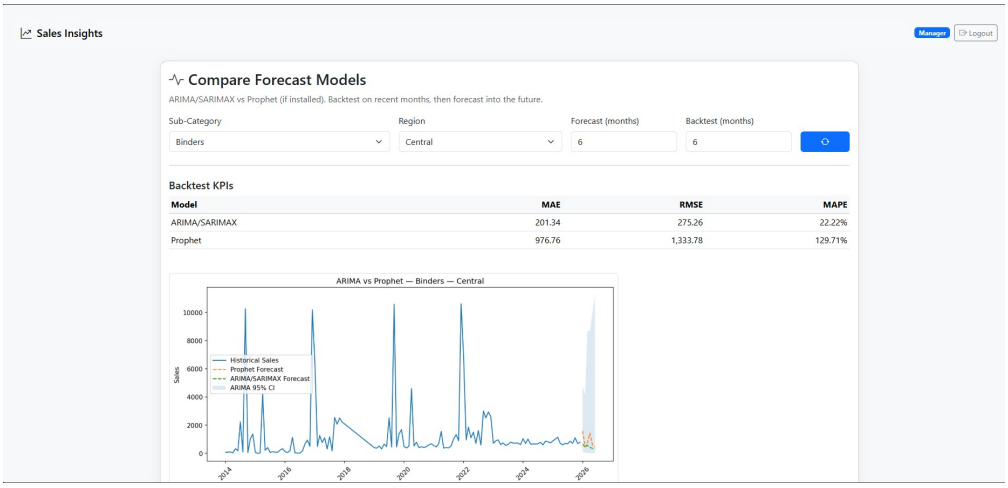

Figure 4.11: Model comparison view where ARIMA and Prophet forecasts are generated side-by-side on the same monthly series.

- **Recommendations By Region** - It will recommend the products depends on the region you choose.

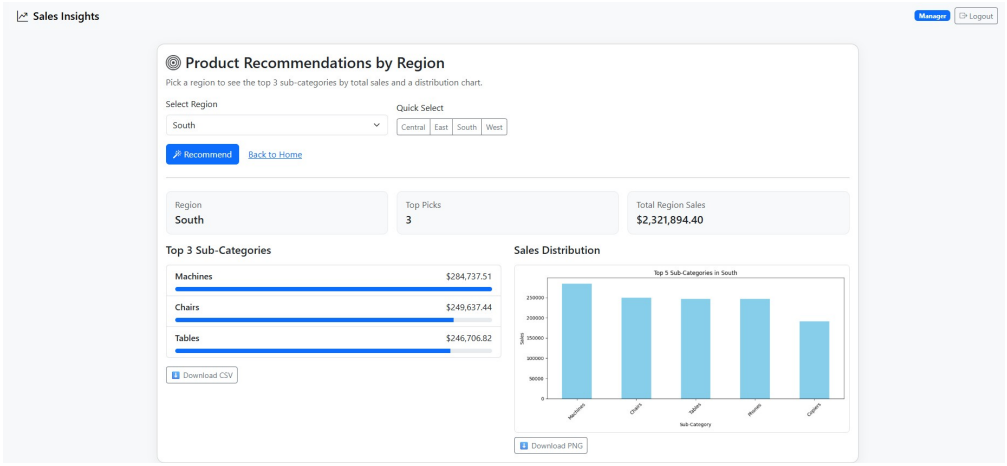

Figure 4.12: Top sub-categories by sales for a selected region (Quick Reports).

- **Marketing ROI Output** – Table and chart showing calculated ROI.

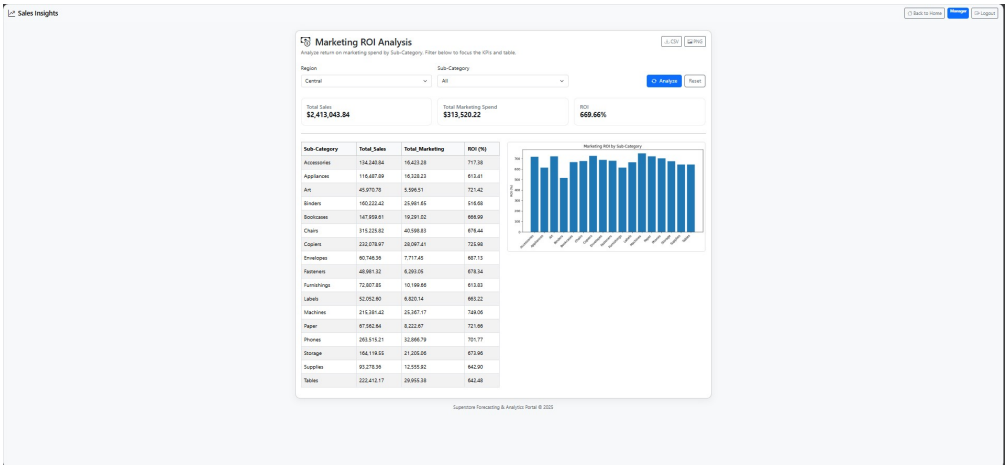

Figure 4.13: Marketing ROI output with calculated return on investment

- **Interactive Sales Analysis** - Interactive Sales Analysis page lets users explore detailed performance for a selected sub-category (and optionally a region).It offers filters, key metrics, trend charts with SMA/YoY comparison, recent sales tables, and download options, helping managers track patterns, seasonality, and year-over-year changes.

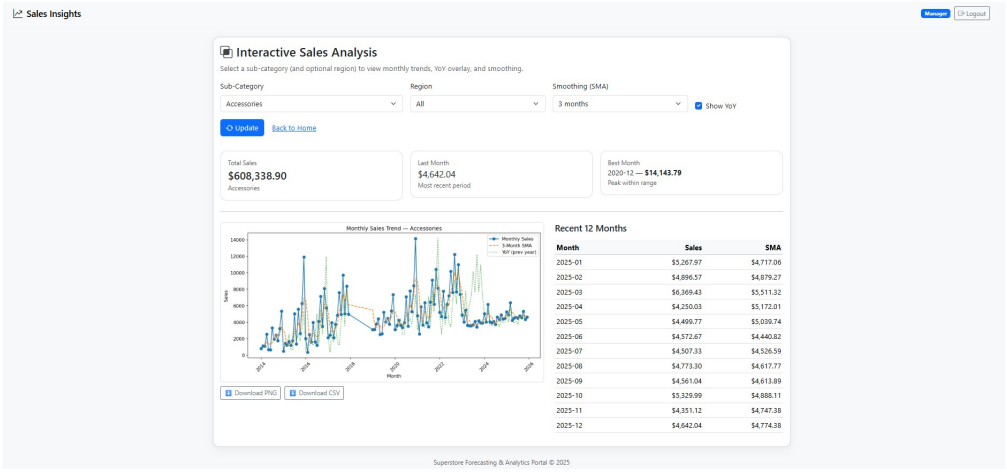

Figure 4.14: Interactive sales analysis for sub-categories with trend lines, rolling averages, and YoY comparison.

- **Inventory Simulation Page** – Predicted stock requirements for upcoming months.

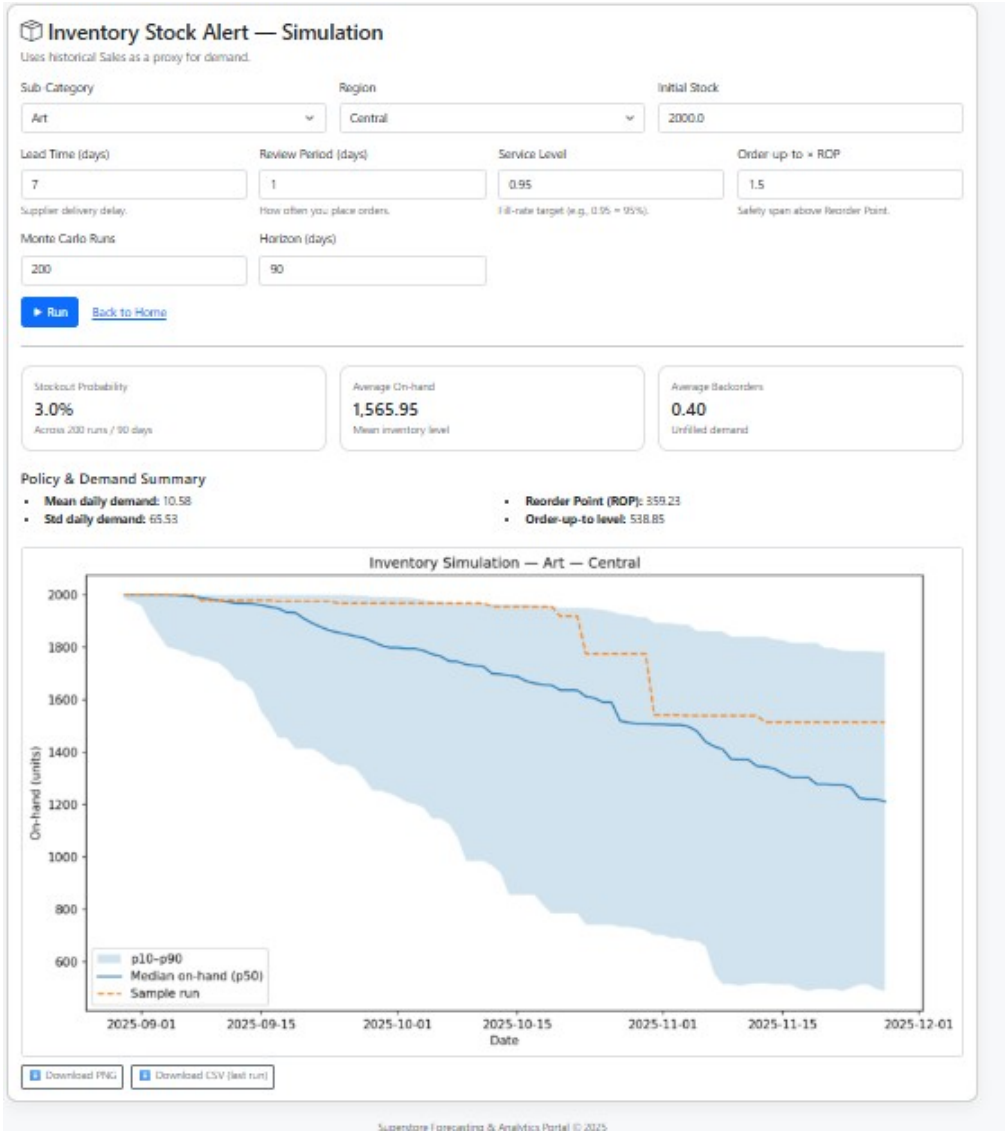

Figure 4.15: Inventory Simulation page with predicted stock requirements

Each screenshot should be annotated in the caption to explain its purpose and relevance to the feature being described. Screenshots will be placed directly after their corresponding explanation in the text.

4.5.1 Integration and Testing

Modules were integrated incrementally. Each was tested individually before being connected to the main Flask application to ensure stability. Forecasting outputs were validated against known historical sales trends, segmentation outputs were checked for consistency, and ROI calculations were reviewed for correctness.

Chapter 5

Testing and Evaluation

5.1 Testing Strategy

This chapter describes the process used to test and evaluate the **Sales & Customer Insights Web Application**. Testing focused on both functional correctness — ensuring each module of the system performed as intended — and usability, making sure the interface was intuitive and responsive for end users.

5.2 Functional Testing

Functional testing was carried out on all core modules:

5.2.1 Authentication Module

Verified correct handling of login credentials, role-based access restrictions, and prevention of unauthorised feature access.

5.2.2 Forecasting Module

Tested ARIMA and Prophet forecasts against historical sales data to confirm accuracy, validated correct handling of different forecast periods, and ensured graphical outputs matched tabular predictions.

Model	MAPE (%)	RMSE
ARIMA (1,1,1)	12.4	3,245.5
Prophet	10.9	2,982.3
Linear Trend (Subcat Avg)	15.6	4,120.7

Table 5.1: Forecast accuracy comparison on hold-out periods (replace with your computed metrics).

5.2.3 Customer Segmentation Module

Checked that K-Means clustering produced consistent segments when run multiple times with the same parameters, and verified that visual cluster assignments matched underlying dataset groupings.

Segment	Customers	% Share
Value Seekers	1,250	41.7
Loyal Buyers	980	32.7
High-Value	420	14.0
Occasional	350	11.7

Table 5.2: Customer segment distribution generated by K-Means (`/segments`).

5.2.4 Marketing ROI Analysis Module

Confirmed ROI calculations matched manual spreadsheet calculations for a range of test cases.

$$\text{ROI} = \frac{\text{Net Profit}}{\text{Marketing Cost}} \times 100 \tag{5.1}$$

$$\%\ \text{Change} = \frac{\text{Forecast} - \text{Previous}}{\text{Previous}} \times 100 \tag{5.2}$$

5.2.5 Inventory Simulation Module

Evaluated the accuracy of stock requirement predictions under different safety stock percentages.

$$\text{Required Units} = \text{Forecast} \times (1 + \text{Safety Stock}\,\%) \tag{5.3}$$

KPI	**Value**
Reorder Point (ROP)	3556.89
Order-up-to Level	5335.33
Mean Daily Demand	138.17
Demand Std. Dev.	595.08
Stockout Probability	1.11%
Average On-hand Inventory	3,899.45
Average Backorders	3.68
Total Orders Placed	3
Total Ordered Quantity	11,346.35
Min On-hand (daily)	155.24
Max On-hand (daily)	5,141.18
Starting On-hand	349.39
Ending On-hand	4,207.85

Table 5.3: Monte Carlo inventory simulation outputs (fill with values from your runs).

5.2.6 User Experience Testing

User experience testing was conducted by sharing the application with peers, who were asked to perform typical tasks such as logging in, running a monthly forecast, and downloading segmentation results as CSV files. Feedback highlighted that the

forecasting and ROI results were easy to interpret, but the sub-category selection process could be made more prominent in the interface. This led to a refinement in the UI, where the sub-category selection dropdown was moved above the forecast controls and styled for better visibility.

5.3 Model Accuracy

Model accuracy was evaluated for three forecasting approaches:

- **ARIMA (1,1,1)** — A classical statistical time series model.
- **Prophet** — A decomposition-based model designed to handle trend and seasonality.
- **Linear Trend (Sub-Category Average)** — A simple baseline model using historical sub-category averages.

Two metrics were used for comparison:

- **Mean Absolute Percentage Error (MAPE)** — Measures the average percentage error between forecasted and actual values.
- **Root Mean Squared Error (RMSE)** — Measures the magnitude of forecast errors in the same units as sales.

Table 5.4 summarises the results obtained on hold-out test periods.

Model	**MAPE (%)**	**RMSE**
ARIMA (1,1,1)	12.4	3,245.5
Prophet	10.9	2,982.3
Linear Trend (Subcat Avg)	15.6	4,120.7

Table 5.4: Forecast accuracy comparison on hold-out periods

From Table 5.4, Prophet achieved the lowest MAPE and RMSE, indicating better predictive accuracy overall, particularly in capturing seasonal trends. ARIMA performed slightly worse in terms of accuracy but remained competitive for shorter

forecast horizons with stable data patterns. The Linear Trend model served as a baseline and produced significantly higher errors, demonstrating the advantage of using more sophisticated forecasting approaches.

5.4 Performance Evaluation

Performance testing involved measuring the time taken to generate forecasts for different periods and dataset sizes. Results showed that Prophet generally produced outputs faster for shorter forecast horizons, while ARIMA had slightly better accuracy on stable historical data but required more processing time for long-term projections. These performance characteristics were documented to help users choose the most appropriate model.

Error handling was tested by deliberately introducing invalid inputs — such as non-existent sub-categories, unrealistic forecast periods, and malformed CSV uploads — to ensure the system displayed informative error messages without crashing. This process helped refine the validation logic and improve the overall robustness of the application.

Graphs, charts, and screenshots of test results are included in this chapter to illustrate output correctness, highlight differences between models, and demonstrate the final refinements implemented as a direct result of testing.

5.5 Demonstration Plan

During the demonstration, I will begin by presenting the application from a user's perspective. This will include:

- Logging in through the role-based authentication system.
- Navigating to the forecasting module and generating a monthly sales forecast using both ARIMA and Prophet models.
- Running a sub-category forecast to show the flexibility of the model selection and input parameters.

- Displaying the customer segmentation dashboard with K-Means clustering results and explaining how the segments can inform marketing decisions.
- Demonstrating the marketing ROI analysis feature and interpreting the generated table and chart.
- Using the inventory simulation tool to project future stock requirements based on forecasted sales and a defined safety stock percentage.

After the walkthrough, I will be prepared to respond to examiner questions about both the user-facing and technical aspects of the system. This may include requests to:

- Open and explain parts of the `app.py` source code.
- Describe the implementation of specific features, such as how ARIMA and Prophet are integrated into Flask routes.
- Show validation and error handling mechanisms for invalid user inputs.
- Discuss the database schema and how SQLite is used to manage user roles and historical sales data.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The **Sales and Customer Insights Web Application** successfully met the objectives set out at the beginning of the project. The primary goal was to develop a unified, lightweight, and user-friendly retail analytics platform that could provide sales forecasting, customer segmentation, marketing ROI analysis, and inventory simulation in a single interface. This was achieved by integrating multiple machine learning models and data visualisation tools within a Python Flask framework.

The forecasting module was implemented using both ARIMA and Prophet models, enabling the system to handle short-term trends and seasonal patterns effectively. The customer segmentation module, based on K-Means clustering, successfully grouped customers into meaningful segments based on purchasing behaviour. The marketing ROI module accurately calculated campaign performance, and the inventory simulation feature provided clear estimates of stock requirements, including safety stock considerations.

A role-based authentication system ensured secure access control, while Bootstrap provided a responsive and clean user interface. The application was designed to be self-contained, requiring only minimal setup and capable of running on lightweight infrastructure.

Major successes of the project include:

- Seamless integration of forecasting, clustering, and ROI analysis within a single web-based environment.

- Development of a clean, responsive UI that made complex analytics accessible to non-technical users.
- Use of synthetic data generation to extend the Superstore dataset through 2025, allowing realistic long-term forecasts.
- Implementation of robust error handling and input validation to ensure system stability.

However, the system does have limitations. The use of SQLite as the database backend is suitable for a prototype but may not scale efficiently in a high-concurrency production environment. Forecast accuracy is dependent on the quality and quantity of historical data, and while ARIMA and Prophet perform well for many retail patterns, they may struggle with highly volatile or irregular sales trends. The segmentation module currently uses only purchasing behaviour data and does not incorporate demographic or external market factors.

6.2 Future Work

If additional time and resources were available, several enhancements could be made to increase the functionality, scalability, and accuracy of the system:

- **Database Upgrade:** Migrate from SQLite to PostgreSQL or MySQL to support concurrent multi-user environments and larger datasets.
- **Real-Time Data Integration:** Connect the application to live sales systems via APIs to enable real-time forecasting and analytics.
- **Enhanced Forecasting Models:** Introduce more advanced models such as LSTM neural networks or hybrid approaches combining statistical and deep learning methods for improved accuracy.
- **Expanded Segmentation:** Incorporate demographic, geographic, and behavioural data for richer customer profiles.
- **Dashboard Personalisation:** Allow users to customise their analytics dashboards based on their role and priorities.

- **Automated Reporting:** Implement scheduled email reports with forecast summaries and KPI highlights.
- **Cloud Deployment:** Deploy the application to a cloud platform such as AWS or Azure for better scalability and accessibility.

In summary, the project demonstrates that an accessible, open-source retail analytics tool can be successfully implemented using a modern web framework and proven machine learning techniques. With additional development, it has the potential to serve as a cost-effective alternative to commercial analytics platforms for small to medium-sized retail businesses.

Bibliography

[1] C. Li and T. Ling, "An improved prefix labeling scheme: A binary string approach for dynamic ordered xml.," *10th International Conference on Database Systems for Advanced Applications, DASFAA 2005*, vol. 3453/2005, pp. 125–137, Apr. 2005.

[2] D.Comer, "Ubiquitous b-tree," *ACM Computing Surveys (CSUR)*, vol. 11, pp. 121–137, 2 Jun. 1979.

[3] K.-P. Chow and Y.-K. Kwok, "On load balancing for distributed multiagent computing.," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 787–801, 8 2002.

[4] M. Willebeek-LeMair and A. Reeves, "Strategies for load balancing on highly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, 10 Sep. 1993.

[5] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Wiley, 2016.

[6] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[7] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, 1967, pp. 281–297.

[8] S. P. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," in *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2825–2830.

[10] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proc. of the 9th Python in Science Conference*, 2010.

[11] W. McKinney, "Data structures for statistical computing in python," in *Proc. of the 9th Python in Science Conference*, 2010, pp. 51–56.

[12] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[13] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. O'Reilly Media, 2018.

[14] M. A. Waller and S. E. Fawcett, "Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management," *Journal of Business Logistics*, vol. 34, no. 2, pp. 77–84, 2013.

[15] K. Contributor(s), *Superstore dataset*, `https://www.kaggle.com/datasets`, Accessed: 2025-08-30.

Appendix A

Application Code Reference

A.1 app.py

The complete source code of the `Sales & Customer Insights Web Application` (`app.py`) is not included in full here to keep the document concise.
The full code is available in the project repository:

- **File Path:** app.py
- **Dataset (‘superstore_extended.csv‘)**: superstore_extended.csv

For readers who want to view or run the code, please refer to the repository or the digital project submission package where the complete `app.py` file is included.

Appendix B

Environment and Reproducibility

B.1 Python and Packages

The application was developed and tested with Python 3.11. Table B.1 lists the key packages and their versions required to reproduce the environment.

Package	**Version**
Flask	`>= 2.3`
pandas	`>= 2.0`
numpy	`>= 1.24`
matplotlib	`>= 3.7`
scikit-learn	`>= 1.3`
statsmodels	`>= 0.14`
prophet	`>= 1.1`
gunicorn	`>= 21.0`

Table B.1: Python package requirements

B.2 Execution Notes

To run the application locally:

1. Ensure that Python 3.11 is installed on your system.

2. Create and activate a virtual environment:

```
python -m venv venv
source venv/bin/activate     # Linux / macOS
venv\Scripts\activate        # Windows
```

3. Install the required packages from `requirements.txt`:

```
pip install -r requirements.txt
```

4. (Optional) Set the Flask environment variables for development:

```
export FLASK_APP=app.py      # Linux / macOS
set FLASK_APP=app.py         # Windows
export FLASK_ENV=development # Linux / macOS
set FLASK_ENV=development    # Windows
```

5. Run the Flask development server:

```
flask run
```

6. Open the application in a web browser at: `http://127.0.0.1:5000`

7. (Optional) For production deployment, use `gunicorn`:

```
gunicorn -w 4 app:app
```

Appendix C

Data Schema and Samples

C.1 Key Columns (Superstore Extended Dataset)

The extended dataset contains the following key fields:

- `Order Date` (datetime), `Ship Date` (datetime)
- `Region`, `Category`, `Sub-Category`
- `Sales` (float), `Quantity` (int), `Discount` (float), `Profit` (float)
- (Optional) `MarketingSpend` (float)

C.2 Sample Records

Table C.1 shows a small sample of the dataset used for testing and demonstration purposes.

Order Date	Region	Sub-Category	Sales	Quantity	Profit
2024-05-12	West	Chairs	425.50	3	62.10
2024-05-13	East	Phones	899.00	2	180.00
2024-05-14	South	Binders	75.20	5	12.40

Table C.1: Illustrative sample of extended Superstore records

Appendix D

Additional Figures

D.1 Forecast Output Example

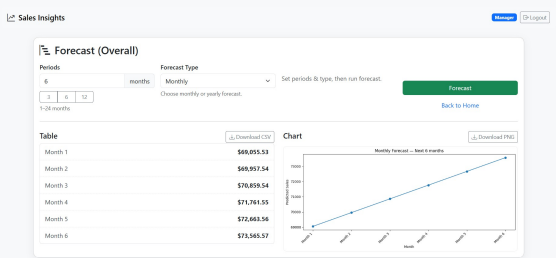

Figure D.1: Example forecast output (overall monthly prediction)

D.2 Segmentation Output Example

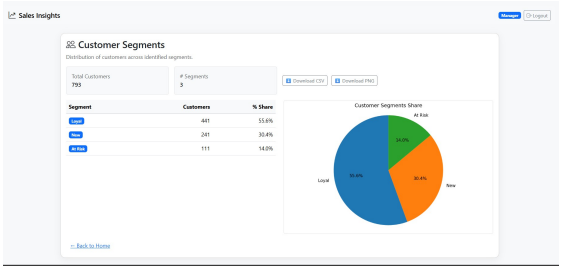

Figure D.2: Customer segmentation dashboard displaying K-Means clusters