**Final Exam**

Instructor: Qasim Ali

Inventory Optimization

**Group Name: Group B**

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Dhanush Chandar Sivakumar | 500228646 |
| Rithwik Prem | 500220919 |
| Sweatha Palani | 500221321 |
| Oluwakanyinsola Adebanjo | 500228268 |
| Rajesh Jayaraman | 500228177 |

Table of Contents

[Abstract 3](#_Toc184341784)

[Introduction 3](#_Toc184341785)

[Case Study 1: Initial Planning & Design 3](#_Toc184341786)

[Problem Definition 3](#_Toc184341787)

[Data Requirements 3](#_Toc184341788)

[System Architecture 4](#_Toc184341789)

[Database Schema 5](#_Toc184341790)

[GitHub Setup 5](#_Toc184341791)

[Midterm: Development Phase 7](#_Toc184341792)

[Data Preprocessing 7](#_Toc184341793)

[Forecasting Model Development (ARIMA) 9](#_Toc184341794)

[Continuous Integration with Jenkins 11](#_Toc184341795)

[Case Study 2: Advanced Development with Ansible 13](#_Toc184341796)

[Optimization Algorithms 13](#_Toc184341797)

[Deployment Automation with Ansible 14](#_Toc184341798)

[Testing and Validation 16](#_Toc184341799)

[Final Exam: Production Deployment 18](#_Toc184341800)

[Deployment Pipeline Design 18](#_Toc184341801)

[Monitoring and System Performance 18](#_Toc184341802)

[Challenges and Resolutions 19](#_Toc184341803)

[Conclusion 20](#_Toc184341804)

[Recommendations 20](#_Toc184341805)

# Abstract

Design and creation of a more advanced system for the optimal control of inventory based on AI helping on-demand predictions and restocking. This system lowers the inventory holding costs and increases stock availability by incorporating demand forecasting via ARIMA and optimization algorithms configured for this purpose. Jenkins & Ansible – IT automation Reducing the manual steps during development, testing and deployment phase. These new techniques provide an example of the possibilities when it comes to AI and automation in retail inventory management.

# Introduction

All retailers need to do is make the process of inventory management efficient by keeping costs and customer satisfaction in control. The purpose of this project is to create an AI system for demand forecasting that results in efficient restocking. The model fuses prediction modes with optimization models and devices of computerization to give a comprehensive solution about inventory that concerns overstocking and stockouts.

# Case Study 1: Initial Planning & Design

## Problem Definition

Retailers often face inefficiencies due to:

* **Overstocking**: Leads to high storage costs and wastage.
* **Understocking**: Results in missed sales opportunities and dissatisfied customers.

**Solution**:  
A predictive analytics AI system analyzing historical sales data to determine how many products should be stocked at any one time.

## Data Requirements

The project requires the following data:

* Sales Data: Past transactions to see a history of purchase patterns.
* Product Information- product categories and pricing.
* Seasonal Data: When do the demands spike and at what time of year.
* Inventory Levels: On-time stock notifications which leads to inventory arrangements.

A screenshot of a computer

Description automatically generated

Figure 1: Sample of raw data screenshots from csv

## System Architecture

The architecture integrates:

* Data Input: Sales and inventory data ingestion.
* Forecasting Model: ARIMA for demand prediction.
* Optimization Algorithm: Linear programming for restocking suggestions.

A diagram of a software development process

Description automatically generated

## Database Schema

Key tables:

* Products Table: Where product info is stored (ID, Name, Category) •categories
* Sales Table: Order details (sale ID, product ID, quantity ==> transaction Log)
* Inventory Table: Maintains the current stock levels and restocks history.

## GitHub Setup

* Repository Structure: [Github Repository link-GroupB](https://github.com/GroupB-DeploymentOfAISolution/Group-B)
  + **/data**: Sample datasets.
  + **/models**: Forecasting and optimization scripts.
  + **/docs**: Project documentation.
* Version Control: Branches for individual contributions and a main branch for production.

A screenshot of a computer

Description automatically generated

Figure 2: GitHub respository overview

# Midterm: Development Phase

## Data Preprocessing

* Data Cleaning:
  + Duplicates were removed using data.drop\_duplicates() to ensure the dataset contained unique records.
  + Handled missing values in key columns like Description, Quantity, Price, and CustomerID using data.dropna().
* Outlier Detection:
  + Identified and removed extreme values in Quantity and Price to improve data quality.
  + Box plots were used to visualize and filter outliers using the IQR method (1.5 \* IQR).
* Feature Engineering:
  + Created new features like:
    - Total Price: Calculated as Quantity \* Price for each transaction.
  + Monthly Sales Trends: Aggregated sales data to observe patterns over time.
  + Visualized data trends, such as top products, top countries by sales, and sales distribution over time, for exploratory insights.

A screenshot of a computer

Description automatically generated

Figure 3: Cleaned dataset on jupyter notebook

A screenshot of a computer

Description automatically generated

Figure 4:Screenshots of some EDA performed on the cleaned dataset

## Forecasting Model Development (ARIMA)

**Model Selection**:

* ARIMA Model:
  + - The ARIMA model was selected due to its effectiveness in time-series forecasting.
    - The Model Building section shows detailed performance metrics and output, including the AIC and BIC scores, which were likely used for hyperparameter tuning.

**Validation:**

* **Data Splitting and Forecasting:**
  + Historical sales data was used for model validation, as indicated in the Data Preprocessing section. Outliers and inconsistencies in sales data (e.g., negative values) were removed during preprocessing.
  + The Model Forecasting section includes a comparison graph of forecasted sales versus actual sales, demonstrating model accuracy visually.
* **Evaluation Metrics:**
  + The Model Evaluation section shows metrics like:
    - Mean Squared Error (MSE): 6981501284.1
    - Mean Absolute Error (MAE): 383153.875153534
  + These metrics validate the performance of the model.

A screenshot of a computer

Description automatically generated

Figure 5: ARIMA Model Building

A screen shot of a computer

Description automatically generated

Figure 6:ARIMA Model Training

A screenshot of a computer

Description automatically generated

Figure 7: ARIMA Model evaluation

## Continuous Integration with Jenkins

* **Pipeline Automation:** The pipeline script automates key steps like setting up the Python environment, cloning the GitHub repository, and installing dependencies.
* **Build Monitoring:** Jenkins tracks build status and history, providing immediate feedback on success or failure.
* **Automated Testing:** pytest is integrated into the workflow to validate code changes, ensuring reliability and preventing regressions.
* **Efficient Workflow:** Integration with GitHub allows Jenkins to fetch code updates, run tests, and prepare the application automatically, minimizing manual intervention.

This setup highlights a robust CI process ensuring efficient code integration, testing, and feedback.

A computer screen shot of a computer

Description automatically generated

Figure 8: Build status in Jenkins

A screenshot of a computer

Description automatically generated

Figure 9: Pipline Configuration

A screenshot of a computer

Description automatically generated

Figure 10: Job Overview in jenkins

A screenshot of a computer

Description automatically generated

Figure 11: Automated testing output

# Case Study 2: Advanced Development with Ansible

## Optimization Algorithms

* **Linear Programming:**
  + Utilized scipy.optimize.linprog to compute optimal restocking quantities.
* **Inputs:**
  + Predicted demand (e.g., [100, 200, 150] as shown in the code).
  + Costs (e.g., [5, 4, 6] for three products).
* **Constraints:**
  + Total quantities restricted by storage capacity (e.g., sum ≤ 500).
  + Supplier limits for each product (bounds [0, ∞]).
* **Output:** Optimal restocking quantities are calculated and displayed.

A screenshot of a computer

Description automatically generated

Figure 12:Linear Progamming Optimization Model

## Deployment Automation with Ansible

* **Ansible Playbooks:**
  + Created playbooks to automate testing and deployment:
    - setup\_testing\_environment.yml:
      * Installed Python3 and pip.
      * Deployed required Python packages based on requirements.txt.
  + **Inventory File:**
    - Defined testing and staging servers in inventory.ini.
    - Example: Testing server at 192.168.1.10.
* **Execution Issues:**
  + SSH connectivity to the testing-server failed (UNREACHABLE error in the playbook run). This needs network troubleshooting.

A screenshot of a computer

Description automatically generated

Figure 13: Setting Up Testing Environment

A screenshot of a computer

Description automatically generated

Figure 14:Ansible Inventory File

A screenshot of a computer

Description automatically generated

Figure 15:Ansible Playbook Execution and Error

## Testing and Validation

* Unit Testing:
  + Example: test\_optimization.py validated the restocking algorithm.
  + Ensured that restocking quantities respect constraints (e.g., quantities between 0 and 200).
* Integration Testing:
  + Verified interaction between input constraints, cost calculations, and result generation.
* Performance Testing:
  + Example: *test\_performance.py* benchmarked execution times using *pytest-benchmark* to ensure the algorithm's efficiency.

**A screenshot of a computer

Description automatically generated**

Figure 16:Unit Testing Script

A screenshot of a computer

Description automatically generated

Figure 17: Integration Script

A screenshot of a computer

Description automatically generated

Figure 18:Performance Testing Script

# Final Exam: Production Deployment

## Deployment Pipeline Design

* **Text**: Description of Jenkins-Ansible integration for production deployment.
* **Screenshots**: Jenkins pipeline logs for production deployment.

## Monitoring and System Performance

* **Text**: Tools and metrics used for monitoring.
* **Screenshots**:
  + Monitoring dashboards from Prometheus or Grafana.
  + System performance graphs (e.g., response times, resource usage).

# Challenges and Resolutions

|  |  |  |  |
| --- | --- | --- | --- |
| Challenge | |  | | --- | | **Details** | | Solution |
| Data quality and missing values | 1. **Missing Values and Duplicates**: Missing values (NaN) and duplicate rows in the dataset. 2. **Outliers in Data**: Extreme values in Quantity and Price affected model performance. | |  | | --- | | 1. Used dropna() to handle missing values and drop\_duplicates() to remove duplicates. 2. Applied the IQR method to filter outliers using 1.5 \* IQR threshold. |  |  | | --- | |  | |
| Model Performance | **High Error Metrics**: Forecasting model had high MSE (~6.9 billion) and MAE (~383,000). | Fine-tuned ARIMA/SARIMA hyperparameters. Added external variables like seasonality or promotions. |
| Deployment errors with Ansible | |  | | --- | |  |   **Unreachable Testing Server**: Playbook execution failed due to SSH connectivity issues | Verified SSH credentials, server IPs, and firewall settings. Ensured proper SSH keys in ansible.cfg. |
| Optimization Model Constraints | |  | | --- | |  |   Ensuring the model respected stock limits and non-negativity constraints. | Implemented unit tests (test\_optimization.py) to validate these constraints. |

# Conclusion

The project successfully developed an AI-based inventory optimization system addressing overstocking and understocking challenges. Key achievements include:

* Accurate demand forecasting using ARIMA.
* Cost-efficient restocking through linear programming.
* Automation with Ansible and Jenkins for seamless deployment.
* Robust testing and performance monitoring using Prometheus and Grafana.

This system demonstrates the potential of AI and automation to optimize inventory management and improve retail efficiency.

# Recommendations

1. **Expand Data Inputs**: Include external factors like promotions and real-time inventory data for improved predictions.
2. **Advanced Models**: Explore LSTM and ensemble models for better forecasting accuracy.
3. **Scalability**: Extend to multi-store operations and use Kubernetes for deployment.
4. **Optimization Upgrades**: Integrate dynamic pricing and multi-objective optimization.
5. **User Interface**: Build an interactive dashboard for better visualization and parameter adjustments.
6. **Security**: Ensure compliance with data protection regulations and encryption.
7. **Feedback Loop**: Automate model retraining with new sales data to enhance accuracy