Supply Chain Performance Optimization and Analytics

Table of Contents

1. Introduction

Project Overview:

This project focuses on performing advanced analytics on supply chain data to uncover insights that can help in optimizing inventory management, improving supplier performance, and forecasting future demand. By leveraging machine learning and statistical techniques, this project provides actionable solutions to improve the efficiency and effectiveness of supply chain processes.

Objectives:

- Perform demand forecasting to predict the number of products sold.
- Optimize inventory levels using the Economic Order Quantity (EOQ) model.
- Evaluate and improve supplier performance based on metrics like lead time and defect rates.
- Segment suppliers using clustering techniques for better decision-making.

2. Technologies Used

- **Python**: For data analysis, machine learning, and optimization.
- **Pandas**: For data manipulation and analysis.
- **Scikit-learn**: For machine learning models and clustering.
- **Matplotlib and Tableau**: For data visualization.
- **Numpy**: For numerical operations and optimization calculations.

3. Data Source

The dataset contains various supply chain metrics such as product sales, stock levels, lead times, defect rates, and supplier performance. It includes 100 records and 24 features, such as:

Key columns include:

- Product type, SKU, Price, Availability
- Number of products sold, Revenue generated, Stock levels
- Lead times, Shipping times, Shipping costs
- Supplier name, Location, Manufacturing costs
- Defect rates, Transportation modes, Routes

4. Methodology

Data Loading and Preparation:

The dataset was loaded using **Pandas** and explored to understand the structure, check data types, and identify missing values.

```
import pandas as pd
# Load the dataset
supply_chain_data = pd.read_csv('supply_chain_data.csv')
```

```
# Inspect the data
supply_chain_data.info (), supply_chain_data.head()
```

Data Cleaning:
The dataset was checked for missing values, outliers, and inconsistencies. All columns were complete with no missing values, ensuring the data was ready for analysis.

```
# Check for missing values
print(supply_chain_data.isnull().sum())
```

Data Analysis:
Several key supply chain metrics were calculated, including:

- **Inventory Turnover**: Measures how efficiently inventory is sold.
- **Supplier Performance**: Combines lead time and defect rates for evaluating supplier reliability.
- **Cost to Revenue Ratio**: Helps in analysing the cost efficiency of products.

```
# Calculate key metrics

supply_chain_data['Inventory Turnover'] = supply_chain_data['Number of products sold'] / supply_chain_data['Stock levels']

supply_chain_data['Supplier Performance'] = (1 / supply_chain_data['Lead time']) * (1 / supply_chain_data['Defect rates'])

supply_chain_data['Cost to Revenue Ratio'] = supply_chain_data['Costs'] / supply_chain_data['Revenue generated']
```

5. Key Findings

Supplier Performance:

- Top Suppliers: Suppliers with high performance were in Chennai and Mumbai, showing low lead times and defect rates.
- Bottom Suppliers: Suppliers in Bangalore and Kolkata showed low performance, indicating room for improvement.

Cost Efficiency:

- Products such as cosmetics in Mumbai have the lowest cost-to-revenue ratios, indicating efficient cost management.
- Products in Delhi have higher costs relative to revenue, requiring cost reduction strategies.

Inventory Optimization:

- High-turnover products like skincare in Kolkata need consistent restocking to meet demand.
- Low-turnover products like haircare in Mumbai should have reduced stock levels to minimize holding costs.

Predictive Analytics:

- A Random Forest model was built to predict product demand based on features like price, stock levels, and availability.
- The initial model had a **Mean Absolute Error (MAE)** of 309.33 and an **R-squared** value of -0.43, suggesting improvements could be made by adding more features or external data.

Economic Order Quantity (EOQ):

- The EOQ calculation suggests that ordering 960 units would minimize the total costs associated with inventory management.

Segmentation:

- Suppliers were segmented into three clusters using K-Means clustering, revealing high-performing, mid-performing, and low-performing supplier groups.

6. Conclusion

This project successfully demonstrated how to apply data analysis, machine learning, and optimization techniques to improve supply chain processes. By using predictive analytics for demand forecasting, inventory optimization, and supplier segmentation, the project highlights key areas where businesses can improve efficiency and reduce costs.

7. Recommendations

- **Enhance Demand Forecasting:** Improve the demand prediction model by adding more features, such as seasonality, promotions, or market trends.
- **Optimize Supplier Performance:** Work with suppliers in Cluster 3 (low performance) to improve lead times or defect rates or consider switching to better-performing suppliers.
- **Inventory Management:** For low turnover products, reduce stock levels to minimize holding costs and avoid overstocking.
- **Cost Efficiency:** Focus on products with high cost-to-revenue ratios to identify cost-saving opportunities, such as optimizing manufacturing or shipping processes.

8. Appendix

Python Code:

```
# Load the dataset

import pandas as pd

supply_chain_data = pd.read_csv('supply_chain_data.csv')


# Data Cleaning and Key Metrics Calculation
```

```
supply_chain_data['Inventory Turnover'] = supply_chain_data['Number of products sold'] / supply_chain_data['Stock levels']

supply_chain_data['Supplier Performance'] = (1 / supply_chain_data['Lead time']) * (1 / supply_chain_data['Defect rates'])

supply_chain_data['Cost to Revenue Ratio'] = supply_chain_data['Costs'] / supply_chain_data['Revenue generated']


# Predictive Analytics

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, r2_score


X = supply_chain_data[['Price', 'Stock levels', 'Lead time', 'Availability']]

y = supply_chain_data['Number of products sold']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)

print (f"MAE: {mean_absolute_error(y_test, y_pred)}, R-squared: {r2_score(y_test, y_pred)}")


# EOQ Calculation

import numpy as np

annual_demand = supply_chain_data['Number of products sold'].sum()

ordering_cost = 100

holding_cost = 10

EOQ = np.sqrt((2 * annual_demand * ordering_cost) / holding_cost)

print (f"Optimal EOQ: {EOQ}")


# Segmentation
```

```
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

X = supply_chain_data[['Supplier Performance', 'Cost to Revenue Ratio', 'Defect rates']]

kmeans = KMeans(n_clusters=3, random_state=42)

supply_chain_data['Segment'] = kmeans.fit_predict(X)


plt.scatter(X['Supplier Performance'], X['Cost to Revenue Ratio'], c=supply_chain_data['Segment'])

plt.xlabel('Supplier Performance')

plt.ylabel('Cost to Revenue Ratio')

plt.title('Supplier Segmentation')

plt.show()
```