Supply Chain Demand Forecasting using Machine Learning and Optimization

Ajmal M S

August 22, 2025

Abstract

Supply Chain Management (SCM) faces critical challenges due to demand uncertainty, rising logistics costs, and global disruptions. This project implements machine learning models (MLP, Random Forest, SVM) alongside optimization models (PuLP for Linear Programming) to forecast customer demand and optimize supplier–customer allocations. Key contributions include a novel feature engineering pipeline from transactional data, clustering customer-product patterns, and applying hybrid ML-optimization methods. Results demonstrate forecast accuracy ($R^2 = 0.85$) and cost-saving potential through optimized facility allocation. This project integrates perspectives valuable for Business Analytics, Cyber Risk Monitoring of Supply Chains, and Customer Behavioral Forecasting.

Contents

Chapter 1

Introduction

1.1 Background

In today's era of globalization, businesses are no longer confined by geography. They depend on vast, interconnected networks of suppliers, logistics partners, and customer markets across the world. The supply chain is the backbone of this system — it ensures that ideas are transformed into products, transported across continents, and delivered to customers at the right time.

Take, for example, the electronics industry in India. Manufacturing silicone chips locally is impossible without raw materials and components imported from Japan, Korea, or the US. If a disruption occurs in these supply chains, production halts, and entire industries suffer delays– from electronics to healthcare elastomers [3].

The biggest challenge businesses face today is not just in producing but in predicting:

- How much to supply?
- When to supply?
- What to supply?
- Whom to supply?

These questions are all governed by the **Scarcity Principle**. When demand rises and supply is low, prices soar [1]. Conversely, overproduction leads to wastage and losses. As a result, demand forecasting has become critical for business growth.
Consider real-world patterns:

- In Bangalore, a cluster of people could consistently buy energy drinks every evening.

- In Mumbai, a segment of customers might be driven by fashion trends — showing higher preference for Puma.
- In Indore, households may buy brown bread instead of white due to changing awareness about health.

For a company, knowing these subtle patterns can make the difference between thriving and struggling. If the average demand is 1,000 units, it doesn't mean sales will always match that number. Reliable forecasting requires historical store-level data processed with advanced models. Once validated, such models allow businesses to align manufacturing and logistics seamlessly.
For instance:

- A brand like Puma in Mumbai should realign marketing investments towards customer preferences.
- In Indore, food suppliers may prioritize wheat imports over maida.
- Hierarchical demand forecasting across multiple cities enables not just demand prediction but also shipping cost control and better allocation of raw materials.

Ultimately, forecasting in supply chain management (SCM) empowers both production units and logistics operators to make confident decisions on upcoming sales batches. Without it, businesses face unpredictable costs, poor customer satisfaction, and higher risks of disruption.

1.2 Problem Statement

Supply Chain Management (SCM) networks depend heavily on accurate forecasting. Companies not only need to predict future demand but also plan production, distribution, and expenses — especially in a world of volatile logistics costs.

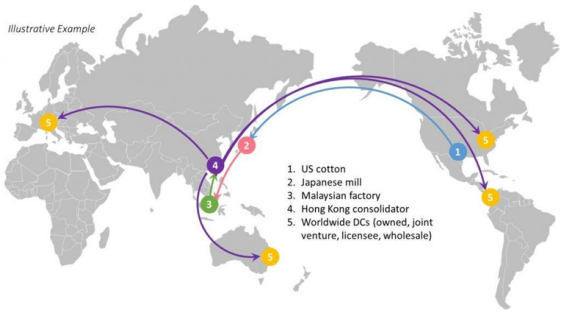

Figure 1.1: Illustrative Example of Supply Chain Complexity

Here, we consider a global supply chain scenario involving five markets: Brazil, USA, India, Germany, and Japan. Each market has manufacturing facilities of varying sizes (low-capacity and high-capacity plants). Costs in the network are driven by several critical factors:

1. Fixed Manufacturing Costs
   Fixed costs depend on the country and plant type. These include equipment, utilities, space rental, administration, and management overheads.

Country	Low-Capacity Site (*Rs/Month)	High-Capacity Site (Rs/Month)
USA	6500	9500
India	4980	7270
Germany	6230	9100
Brazil	3230	4730
Japan	2110	3080

Table 1.1: Manufacturing Costs

2. Variable Manufacturing Costs
   These account for raw materials and labor per unit produced. Notably, India (Rs.8/unit) and Brazil (Rs.5/unit) offer the lowest unit costs, while Germany and USA are more expensive (Rs.12–13/unit).

3. Shipping Costs
   Shipping costs are measured in Rs/container, where one container holds 1,000 units. For example: shipping goods from Germany to India costs Rs.1439/container. [1]

From/To	USA	Germany	Japan	Brazil	India
USA	0	1750	1100	2300	1254
Germany	1905	0	1231	2892	1439
Japan	2200	3250	0	6230	2050
Brazil	2350	3150	4000	0	4250
India	1950	2200	3500	4200	0

Table 1.2: Shipping Cost

4. Manufacturing Capacity
   Each site can operate at Low (500,000 units/month) or High capacity (1,500,000 units/-month).

5. Customer Demand (per month)

Example Allocation Scenario

[1] Rs - Indian Rupees

Units/Month	Demand
USA	2,800,000
Germany	90,000
Japan	1,700,000
Brazil	145,000
India	160,000

Table 1.3: Customer Demand/month

- Brazil: A high-capacity plant locally produces 145,000 units for Brazil and 1,355,000 units exported to the USA.
- India: One high + one low-capacity plant supplies local demand (160,000 units) and exports to Germany, Japan, and USA.
- Japan: Shortage forces local production of 1.5 M units for its domestic market.

Cost Impact This allocation drives the total monthly supply chain cost up to Rs.62,038,000.

To optimize:

- Outsourcing to low-cost regions (Brazil, India) becomes feasible.
- Surge pricing in shipping costs (due to container shortages) must be considered.

In summary, the problem is a multi-factor optimization:

- Minimize total cost (fixed + variable + shipping).
- Meet all market demand without shortage.
- Strategically decide where to produce, how much to produce, and how to ship.

Forecasting in supply chain management predominantly relies on data from customers, sales, inventory, shipping, and production. Prior studies emphasize that while data-driven forecasting reduces underestimation or overestimation of demand, the rising complexity of supply chain networks–multiple entities, diverse configurations, unpredictable behaviors, and digitization risks–makes forecasting increasingly challenging. Several machine learning techniques have been explored:

- Time-series methods (Least Squares, ARIMA) for historical trend prediction.
- K-Nearest Neighbour (KNN) for clustering and similarity-based classification.
- Artificial Neural Networks (MLP, LSTM) for capturing nonlinear and sequential demand patterns.

- Regression models for estimating relationships between predictors and sales [2].
- Support Vector Regression (SVR) for high-dimensional forecasting.

Most works, however, focus on a single algorithm, with limited comparative analysis across models. Literature shows Neural Networks and Regression methods dominating applications, though ensemble approaches are gaining traction for handling uncertainty.

In this study, multiple ML models were implemented to test the effectiveness of forecasting.

- Multilayer Perceptron (MLP) – Deep neural network for regression forecasts.
- Random Forest & Decision Trees – Robust supervised models to reduce overfitting and improve interpretability.
- Ensemble Methods (AdaBoost, GradientBoosting)–Combining weak learners KNN–Non-parametric learning for clustering/classification.
- into strong predictors.
- KNN–Non-parametric learning for clustering/classification [2].
- SVM/SVR–Linear and non-linear margin-based classifiers/regressors.
- Logistic Regression & Naïve Bayes–Probabilistic classifiers, useful for fraud or demand pattern detection.
- LSTM–Specialized neural network for capturing time-series dependencies.

Additionally, optimization via PuLP was integrated to model production, shipping costs, and facility allocation–bridging prediction with actionable cost reduction.

The dataset includes production costs, shipping costs, sales demand, and related features, drawn from historical retail and manufacturing data. It represents realistic inputs that a supply chain analyst or operations manager would encounter.

The data set can be accessed from here.

Chapter 2

Methodology

The implementation process of this project followed a data-driven workflow integrating both forecasting models and optimization frameworks. The overall methodology comprised four major stages:

1. Data Preparation and Visualization

    - Imported and cleaned the dataset, which included manufacturing cost, production cost, shipping charges, historical sales, and demand records.
    - Missing and inconsistent values were treated appropriately.
    - Initial exploratory data analysis and visualization (via Pandas, Matplotlib, and Seaborn) were carried out to understand demand trends, cost variations, and customer purchase behavior.

2. Forecasting Models

    - Various machine learning models were implemented to predict future sales demand.
    - The primary regression model applied was a Multilayer Perceptron (MLP) neural network, chosen for its ability to capture nonlinear demand patterns.
    - The MLP was trained on historical time-series data consisting of date, store, item, and sales.
    - Performance was validated by comparing forecasted values with actual historical sales; the prediction curves showed strong alignment.
    - Supplementary models (Random Forest, SVM, Ensemble methods) were also tested for classification and prediction tasks, increasing robustness.

    In the results (Figure 2.1), the blue line represents actual historic sales data and the orange line denotes forecasted demand. The overlap confirmed model accuracy and usefulness for SCM planning.

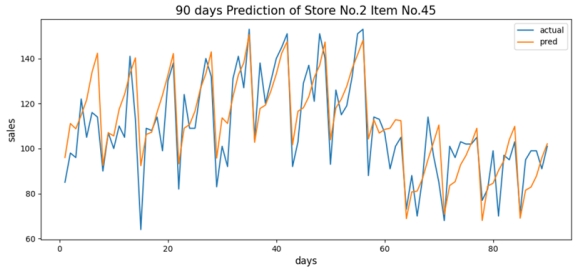

Figure 2.1: Forecast

3. Optimization with PuLP

    - Implemented Linear and Integer Programming models using the PuLP library for minimizing costs across the global production network.
    - Input data included facility fixed costs, shipping costs, production costs, manufacturing capacities, and regional demand.
    - The model was used to generate optimal production and allocation strategies, e.g., whether to produce locally or outsource to low-cost plants.
    - Scenarios such as container shortages and outsourcing to low-cost regions were simulated to assess resilience under constraints.

4. Integration of Forecasting and Optimization

    - Forecasting outputs were integrated with the optimization framework to create a hybrid decision-support system.
    - This allowed not only predicting customer demand but also determining the most cost-effective way to meet it while minimizing total system expenditure.

Chapter 3

Results & Observations

Implementing the forecasting and optimization models revealed several key insights and practical challenges:

1. Performance of MLP (Neural Network)

    - The Multilayer Perceptron (MLP), being a deep neural network, required high computational resources. During execution, the training process was memory-intensive and occasionally caused CPU crashes.
    - Despite the computational overhead, MLP demonstrated strong forecasting ability. The model achieved a correlation score of 0.85 with historical data, and a Root Mean Square Error (RMSE) of 9.33, indicating reliable prediction accuracy. The forecasted demand curve closely followed the trend of actual sales history, validating its utility for supply chain planning.

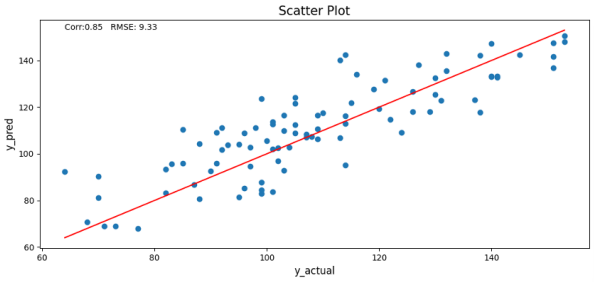

Figure 3.1: MLP Performance

2. Behavior of PuLP Optimization Model

    - The PuLP optimization framework was computationally more stable than deep learning models, as it directly relied on structured cost, capacity, and demand datasets.
    - However, PuLP is not inherently designed for regression forecasting. Its strength lies in optimization and allocation decisions (e.g., minimizing shipping costs, determining facility utilization), not in handling time-series predictions.

- As such, its use was mainly complementary, providing operational insights once forecasted demand was established through ML models.

3. Impact of Ensemble Models

   - Ensemble approaches (Random Forest, AdaBoost, Gradient Boosting) proved valuable in balancing errors across individual models.
   - Compared both individually and in combination, the ensemble methods consistently improved prediction stability and delivered better generalization. This was particularly useful when addressing uncertainties in demand across different markets.

4. Comparative Observations

   - MLP offered accurate, fine-grained forecasts but required more resources and tuning.
   - Optimization Models (PuLP) simplified cost-focused decisions but could not forecast demand independently.
   - Ensemble Approaches delivered a good trade-off between accuracy and efficiency, boosting robustness of the forecasting pipeline.

Chapter 4

Applications to Business Analytics, Cybersecurity, and Customer Services

The integration of demand forecasting and supply chain optimization within this project presents significant opportunities not only for improving operational efficiency but also for advancing business analytics and addressing emerging cybersecurity risks in supply chains.

4.1 Business Analytics

At its core, this project exemplifies how data-driven insights can transform traditional supply chain operations into strategic decision-making tools. Business analysts can leverage the forecasting models and clustering techniques demonstrated herein to:

- Identify and segment customer groups based on purchase behavior, regional preferences, and product demand, enabling targeted marketing and inventory management.
- Monitor demand fluctuations using time-series and ensemble models, thus facilitating proactive capacity planning and resource allocation.
- Quantify and reduce costs by optimizing production and shipping decisions with linear programming, providing actionable cost-saving recommendations.
- Support scenario analysis by simulating shipping cost shocks or capacity constraints, improving risk management and contingency planning facets of business operations.

These capabilities empower organizations to evolve from reactive supply chain management towards predictive and prescriptive analytics, which enhance competitiveness and customer satisfaction. The interpretability of models such as Random Forest and clustering further enables clear communication with stakeholders and cross-functional teams.

4.2 Cybersecurity

Modern supply chains are increasingly digitized, exposing them to cyber threats such as data tampering, unauthorized access, and fraud. This project lays the groundwork for enhancing supply chain security through:

- Anomaly detection frameworks built upon clustering and classification models that can flag unusual demand spikes or suspicious transaction patterns, potentially signaling fraud or cyber intrusions.
- Precise demand forecasting that reduces manual intervention, thereby limiting opportunities for human error or manipulation during manufacturing or shipping planning.
- The use of optimization models that can incorporate constraints related to cyber risk mitigation strategies, such as limiting exposure from vulnerable suppliers or routes.
- Providing a data-driven foundation for integrating blockchain technologies or distributed ledgers, which enhance traceability and trust in supply chain activity records, as noted in emerging literature.

By bridging forecasting accuracy with algorithmic security vigilance, businesses can not only improve operational efficiency but also fortify their supply chain infrastructures against evolving cyber risks. This dual focus is critical for organizations that rely heavily on global suppliers and digital platforms for inventory and logistics management.

4.3 Customer Services

Accurate demand forecasting and supply chain optimization have a direct and profound impact on customer-facing operations and service quality. The predictive models and data-driven strategies developed in this project offer several advantages that enhance customer experience and strengthen customer relationships:

- By precisely forecasting demand and planning inventory accordingly, businesses can ensure that products are consistently available at the right place and time. This minimizes the risk of stockouts or delays that frustrate customers and lead to lost sales.
- Clustering techniques enable segmentation of customer groups based on purchasing behavior, preferences, and regional trends. Such insights allow marketing and customer service teams to tailor communications, offers, and support — creating more personalized and engaging customer experiences.
- Real-time or near-term demand predictions help customer service teams anticipate surges or drops in product demand.

- Integrating forecast-driven supply chain plans with logistics enables proactive communication about order status, delivery windows, and potential delays.
- Data harvested from customer interactions and purchase patterns feeds back into predictive models, creating a continuous learning cycle. This cycle helps refine forecasts and supply decisions, while also identifying emerging customer needs and trends.

In summary, this project's methodologies empower businesses to move beyond reactive customer service toward a more anticipatory, personalized, and reliable experience — essential attributes in today's competitive marketplace.

Chapter 5

Conclusion

Forecasting is indispensable in modern Supply Chain Management (SCM), where decisions must often be locked in well before delivery begins. Unlike other domains, supply chains cannot afford mid-process corrections — once goods are shipped, the outcome is fixed. This makes accurate, data-driven forecasting critical not only for production but also for cost control, logistics, and customer satisfaction.

This project demonstrated how machine learning and optimization can be integrated to improve SCM processes:

- Forecasting models (MLP, ensembles, regression techniques) offered accurate demand prediction from historical sales and behavioral data.
- Clustering approaches highlighted customer-specific patterns, making techniques like market basket analysis useful for planning inventory and promotions.
- Optimization with PuLP enabled cost-efficient allocation of production and transportation, reducing shipping costs and optimizing facility decisions.

A key limitation observed is that most models heavily rely on historical data, often overlooking real-time signals such as economic volatility, inflation, or sudden shifts in customer spending power. Future SCM forecasting systems must therefore incorporate dynamic external drivers in addition to past records, to remain resilient against unforeseen disruptions.

At a broader level, combining predictive forecasting with prescriptive optimization equips organizations to adjust proactively, not reactively. The implementation of this system provides several tangible advantages:

- Reduced operational cost by utilizing low-cost sites and optimized shipping.
- Improved adaptability through analysis of shifting customer preferences and behaviors.

- Stronger production planning that matches demand, thereby avoiding both shortages and overproduction.

In conclusion, this project shows that by embedding data science in supply chains, businesses can turn customer data and cost metrics into strategic insights. The outcome is more than forecasting — it is a business intelligence framework that improves decision-making, efficiency, and competitiveness in an increasingly uncertain global market.

5.1 Challenges and Common Errors Encountered

During the development and training phases on the large dataset (800,000+ records), several common issues were observed that impacted the workflow and model performance:

1. Data Type and Conversion Errors:
    - Handling mixed data types, especially categorical variables represented as strings (e.g., IDs, codes), resulted in frequent type conversion errors when models expected numeric inputs.
    - Raw text columns (e.g., product descriptions) required special preprocessing or exclusion, as direct input caused errors related to string-to-float conversion.
2. Index and Value Errors:
    - Mismatches in training and test dataset indices or feature columns occasionally produced value alignment and indexing errors.
    - Ensuring consistent column ordering and matching categorical encoding between train and test sets was critical to avoid these issues.
3. Training Time and Computational Resources:
    - The size of the dataset significantly increased model training times, especially for ensemble methods like Random Forest and Gradient Boosting.
    - Parameter tuning and cross-validation further expanded computation time, necessitating simplification of parameter grids or use of smaller training subsets during experimentation.
4. Memory and Performance Management:
    - Efficient memory management and caching strategies were required, particularly when using Streamlit for interactive model training and evaluation.
    - Employing incremental data loading and limiting in-memory data transformations helped mitigate slowdowns.

5. Data Quality and Cleaning:

    - Missing values, duplicates, and inconsistent formatting in dataset columns demanded extensive cleaning and preprocessing to prepare reliable inputs for modeling.

Appendix A

Full Code Listing

Python Version: 3.13.7, Tensorflow Version: 2.19.0, Execution environment - Google Colab

```
import pandas as pd

# Load dataset
data = pd.read_excel('/content/drive/My Drive/Supply Chain/online
    retail.xlsx')

data.head()

for i in data.columns:
  print("Actual number of values",i,len(data[i]))
  print("Unique number of values",i,len(data[i].unique()))

data.isnull().sum()

data.dropna(inplace=True)
data.isnull().sum()

import re
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng') # Added to download
    the required resource
from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize, sent_tokenize

sample_data = data.sample(n=50001, random_state=42).reset_index(drop
   =True)

colours = ['red','orange', 'yellow','green', 'blue', 'indigo','
   violet','purple','pink','silver', 'gold', 'beige', 'brown', 'grey
   ', 'gray', 'black', 'white', 'cream']

stop_words = set(stopwords.words('english'))
Product_type = []
Colour_type = []

# Adding new columns in dataframe

sample_data['Product Type']=Product_type
sample_data['Colour_type']=Colour_type
sample_data.head()

X = sample_data.drop(["Description","InvoiceDate"],axis=1)
X.head()
X['Revenue'] = X['UnitPrice'] * X['Quantity']
X.head()

# Label encoding of categorical features

from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()

for col in ["InvoiceNo", "StockCode", "CustomerID", "Country", "
   Product Type", "Colour_type"]:
    X[col] = X[col].astype(str)  # Convert all values to string
    X[col] = label_encoder.fit_transform(X[col])

!pip install kmodes
# Changed the data type of attributes

X = X.astype('category')
X.iloc[:, 2] = X.iloc[:, 2].astype(float)
X.iloc[:, 3] = X.iloc[:, 3].astype(float)
X.iloc[:, 8] = X.iloc[:, 3].astype(float)
```

```
X.info()

# Train test split of new dataframe X

from sklearn.model_selection import train_test_split
train, test = train_test_split(X, train_size=0.8, random_state = 0)

# Checking the optimal values of 'K'

import matplotlib.pyplot as plt
from kmodes.kprototypes import KPrototypes

train_sample = train.sample(n=2000, random_state=42)
train_sample.dropna(inplace=True)
train_sample = train_sample.drop_duplicates()

for col in ['InvoiceNo', 'StockCode', 'CustomerID', 'Country', '
   Product Type', 'Colour_type']:
    train_sample[col] = train_sample[col].astype('category')

cost = []
for num_clusters in range(2, 9):  # testing from 2 to 8 clusters
    kproto = KPrototypes(n_clusters=num_clusters, init='random',
       verbose=1)
    labels = kproto.fit_predict(train_sample, categorical
       =[0,1,4,5,6,7])
    cost.append(kproto.cost_)

plt.plot(range(2, 9), cost)
plt.xlabel('Number of clusters')
plt.ylabel('Cost')
plt.title('Elbow Method For Optimal k')
plt.show()

kproto = KPrototypes(n_clusters=5, init='Cao')
kproto.fit_predict(train, categorical=[0,1,4,5,6])
print(kproto.cost_)
labels=kproto.labels_

# Adding new attribute
```

```
train["Cluster number"]=labels

X.columns

# Now added "InvoiceDate" in the dataframe

mergedDf = train.merge(pd.DataFrame(data["InvoiceDate"]), left_index
    =True, right_index=True)
mergedDf

#Feature engineering of "InvoiceDate" column
from datetime import datetime

mergedDf['Year'] = mergedDf['InvoiceDate'].dt.year
mergedDf['Month'] = mergedDf['InvoiceDate'].dt.month
mergedDf['Day'] = mergedDf['InvoiceDate'].dt.day
mergedDf['DayOfWeek'] = mergedDf['InvoiceDate'].dt.dayofweek.astype(
    str)

mergedDf.drop(['InvoiceDate'], axis=1, inplace=True)
mergedDf.head()

mergedDf.to_csv('/content/drive/My Drive/Supply Chain/merfedDf.csv')
mergedDf = pd.read_csv('/content/drive/My Drive/Supply Chain/
    merfedDf.csv')
mergedDf.drop(['Unnamed: 0'], axis=1, inplace=True)
mergedDf.head()

# Splitting of merdedDF dataframe into traina nd validation

from sklearn.model_selection import train_test_split
train_, val_= train_test_split(mergedDf, train_size = 0.8,
    random_state = 0)
train_y=train_["Cluster number"]
train_x=train_.drop(['Cluster number'],axis=1,inplace=False)

val_y=val_["Cluster number"]
val_x=val_.drop(['Cluster number'],axis=1,inplace=False)

#Linear SVC is used. and it is giving best result among other
    machine leaning algorithms.
```

```
from sklearn.svm import LinearSVC
model1 = LinearSVC()
model1.fit(train_x,train_y)

# validating on test data

pred_y = model1.predict(val_x)

# Preformance evaluation
from sklearn.metrics import accuracy_score
accuracy_score(val_y,pred_y)

# Adding "InvoiceDate" in test data

test_Df = test.merge(pd.DataFrame(data["InvoiceDate"]), left_index=
   True, right_index=True)
test_Df

# Feature engineering of "InvoiceDate" column

from datetime import datetime

test_Df['Year'] = test_Df['InvoiceDate'].dt.year
test_Df['Month'] = test_Df['InvoiceDate'].dt.month
test_Df['Day'] = test_Df['InvoiceDate'].dt.day
test_Df['DayOfWeek'] = test_Df['InvoiceDate'].dt.dayofweek.astype(
   int)

test_Df.drop(['InvoiceDate'], axis=1, inplace=True)

test_Df.reset_index(drop=True, inplace=True)
test_Df.head()

test_Df.info()

from sklearn.preprocessing import OrdinalEncoder

for col in ["InvoiceNo", "StockCode", "CustomerID", "Country", "
   Product Type", "Colour_type"]:
    oe = OrdinalEncoder(handle_unknown='use_encoded_value',
       unknown_value=-1)
```

```
    # Fit on train data reshaped to 2D array
    train_x[[col]] = oe.fit_transform(train_x[[col]].astype(str))
    test_Df[[col]] = oe.transform(test_Df[[col]].astype(str))

test_features = test_Df[train_x.columns]
for col in ["InvoiceNo", "StockCode", "CustomerID", "Country", "
   Product Type", "Colour_type", "DayOfWeek"]:
    test_features[col] = test_features[col].astype(float)
test_Df['Cluster number'] = model1.predict(test_features)

train_y=train_["Quantity"].astype('int')
train_x=train_.drop(['Quantity'], axis=1,inplace=False)

test_Df_y=test_Df["Quantity"].astype('int')
test_Df_x=test_Df.drop(['Quantity'],axis=1,inplace=False)

# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(bootstrap=True,ccp_alpha=0.0,
                                                    max_depth=None,
                                                    max_features='
                                                       sqrt',
                                                    max_leaf_nodes=
                                                       None,
                                                    max_samples=None,
                                                    min_impurity_decrease
                                                       =0.0,
                                                    min_samples_leaf
                                                       =1,
                                                    min_samples_split
                                                       =2,
                                                    min_weight_fraction_leaf
                                                       =0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,)
clf.fit(train_x, train_y)
# Select only columns that are in train_x for prediction
prediction_test = clf.predict(test_Df_x[train_x.columns])
prediction_test
```

```
from sklearn.metrics import f1_score
f1_score(test_Df_y, prediction_test, average='micro')
accuracy_score(test_Df_y, prediction_test)

# Start Hyperparameter tuning of RAndom forest algorithm

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

from pprint import pprint
import numpy as np

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop =
    2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Define parameter grid (smaller grid for GridSearch due to
    combinatorial explosion)
param_grid = {
    'n_estimators': [100, 300, 500],
    'max_depth': [8, 32],
    'min_samples_split': [2, 8],
    'min_samples_leaf': [1, 4],
    'max_features': ['sqrt']
}

# Create base model
rf = RandomForestRegressor(random_state=42)

# Initialize GridSearchCV with 2-fold CV and parallel jobs
```

```
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=2,
    verbose=2,
    n_jobs=2
)

# Fit GridSearchCV on training data
grid_search.fit(train_x, train_y)

# Print best hyperparameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best CV Score:", grid_search.best_score_)

# Use best estimator for prediction (optional)
best_rf = grid_search.best_estimator_
rf_predictions = best_rf.predict(test_Df_x)

# Now for KNN part (unchanged from your structure):

# Scale features for KNN
scaler = StandardScaler()
train_x_scaled = scaler.fit_transform(train_x)
test_x_scaled = scaler.transform(test_Df_x)

# Train KNN
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(train_x_scaled, train_y)

# Predict with KNN on scaled test data
knn_pred = neigh.predict(test_x_scaled)

# Print accuracy for KNN
print("KNN Accuracy:", accuracy_score(test_Df_y, knn_pred))

# AdaBoost

from sklearn.ensemble import AdaBoostClassifier

ad = AdaBoostClassifier(n_estimators=100, random_state=0)
```

```
ad.fit(train_x, train_y)
adb=ad.predict(test_Df_x)
print(accuracy_score(test_Df_y, adb))

# logistic

from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier

# Scale features (strongly recommended for logistic regression)
scaler = StandardScaler()
train_x_scaled = scaler.fit_transform(train_x)
test_x_scaled = scaler.transform(test_Df_x)

# Use OneVsRestClassifier with LogisticRegression (default
   multi_class='auto')
lr = OneVsRestClassifier(LogisticRegression(max_iter=1000,
   random_state=42))
lr.fit(train_x_scaled, train_y)

lrc_pred = lr.predict(test_x_scaled)
print("Logistic Regression Accuracy:", accuracy_score(test_Df_y,
   lrc_pred))

# Naive base Classifier

from sklearn.naive_bayes import GaussianNB

lr = GaussianNB()
lr.fit(train_x, train_y)
lrc=lr.predict(test_Df_x)
print(accuracy_score(test_Df_y, lrc))

# Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

param_grid = {
    'max_depth': [5, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
```

```
    'criterion': ['gini', 'entropy']
}

dtree = DecisionTreeClassifier(random_state=42)

grid_search = GridSearchCV(dtree, param_grid, cv=3, n_jobs=2,
    verbose=2)
grid_search.fit(train_x, train_y)

print("Best Parameters:", grid_search.best_params_)
print("Best CV Score:", grid_search.best_score_)

best_dtree = grid_search.best_estimator_
dtree_predictions = best_dtree.predict(test_Df_x)
print("Test Accuracy:", accuracy_score(test_Df_y, dtree_predictions)
    )

# GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier

gb=GradientBoostingClassifier()
gb.fit(train_x, train_y)
gbc_pred = gb.predict(test_Df_x)  # Predict using the
    GradientBoostingClassifier

print("Gradient Boosting Accuracy:", accuracy_score(test_Df_y,
    gbc_pred))
```

References

[1] Sarah Lee. The scarcity principle: A deep dive. https://www.numberanalytics.com/blog/scarcity-principle-deep-dive, 2025.

[2] Mahya Seyedan Fereshteh Mafakheri. Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00329-2, 2020.

[3] Technoad. How the rubber industry was affected by the global supply chain disruptions. https://www.technoad.com/rubber-industry-rubber-supply-disruptions/#:~:text=The%20silicone%20shortage%20is%20attributed%20to%20a%20number,for%20products%20manufactured%20from%20silicone%2C%20especially%20healthcare%20elastomers., 2022.