Predictive Analytics for Import Planning:

Indian Imports from Asian Countries Dataset

MACHINE LEARNING EDA REPORT

Submitted to
DR. PIYUSH CHAUHAN
Associate Professor
Department of Computer Science & Engineering
Symbiosis Institute of Technology, Nagpur Campus

Submitted by
ABHISHEK WEKHANDE
VII SEM
PRN: 22070521113
Department of Computer Science & Engineering
Symbiosis Institute of Technology, Nagpur Campus

Course Name: *Machine Learning*
Course Code: *T7529*

॥वसुधैव कुटुम्बकम्॥

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Contents

1 Abstract

This project presents a comprehensive analysis of import data from Asian countries using advanced machine learning techniques to uncover hidden trade patterns and enable intelligent classification of regional trade behaviors. The study aims to build robust predictive models capable of classifying trade records into specific Asian sub-regions based on multiple economic and commodity-related parameters. The dataset, titled "Cleaned Imports from Asian Countries", comprises over 490,000 records, containing features such as country name, alpha codes, region, sub-region, HS code, commodity type, and values in different currencies. The project follows a structured approach beginning with data preprocessing, where missing values and duplicates were removed, categorical features were encoded, and numerical values were normalized. Subsequently, ten machine learning algorithms were implemented and evaluated — Logistic Regression, Decision Tree, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Naive Bayes, Gradient Boosting, AdaBoost, XGBoost, and LightGBM. Each model's performance was assessed using standard evaluation metrics such as accuracy, precision, recall, and F1-score.The comparative analysis revealed that ensemble-based algorithms like XGBoost and LightGBM delivered superior performance, achieving the highest accuracy due to their efficient handling of high-dimensional and imbalanced data. Additionally, these models provided better interpretability and scalability, making them ideal for large-scale trade analytics. The project culminated in the development of an interactive Streamlit web application, allowing users to input trade data and obtain real-time regional predictions. Overall, this work demonstrates the potential of machine learning to revolutionize international trade analytics by automating classification, enhancing data-driven policy formulation, and providing deeper insights into the economic interactions among Asian nations.

2 Keywords

Machine Learning, Trade Analytics, Classification, Import Data, Ensemble Learning, XGBoost, LightGBM, Predictive Modeling, Data Preprocessing, Regional Classification

3 Introduction

In the era of globalization, international trade has become one of the most crucial drivers of economic development and stability. Countries rely on imports and exports to fulfill domestic demands, strengthen industrial productivity, and sustain global partnerships. Among these, the trade relations between **India and other Asian countries** are of significant importance due to shared economic objectives, geographic proximity, and diversified commodity exchanges. Analyzing such trade data is vital for understanding market dependencies, evaluating economic health, and shaping trade policies.

The primary motivation behind this project is to leverage the power of **Machine Learning (ML)** to analyze large-scale trade datasets and automate the process of regional classification based on import characteristics. Traditional methods of trade analysis are often limited by manual exploration and static statistical tools, making them inadequate for processing extensive, multi-dimensional datasets. In contrast, machine learning algorithms can efficiently handle large datasets, uncover hidden relationships, and make accurate predictions for decision-making.

This study utilizes a cleaned dataset titled *"Cleaned Imports from Asian Countries"*, obtained from the **Open Government Data (OGD) Platform of India**. The dataset contains over 490,000 records with attributes such as country name, commodity type, HS code, trade value (in different currencies), and regional classifications. It provides a comprehensive foundation for conducting data-driven predictive modeling and regional trade analysis.

3.1 Objectives of the Project

The primary objective of this project is to design and implement a data-driven analytical model capable of identifying patterns and classifying import data from Asian countries using advanced Machine Learning techniques. The project aims to extract meaningful insights from large-scale trade datasets and develop predictive models that can automatically determine the regional classification of trade records.

To achieve this overarching goal, several specific objectives have been defined as follows:

1. **Data Understanding and Preprocessing:** To collect, clean, and preprocess the dataset titled *"Cleaned Imports from Asian Countries"* obtained from the Open Government Data (OGD) Platform of India. This involves handling missing values, removing duplicates, converting categorical attributes into numerical representations, and normalizing quantitative variables to ensure that the data is suitable for machine learning applications.

2. **Exploratory Data Analysis (EDA):** To perform an extensive EDA to understand the underlying structure of the data. This includes identifying key trade patterns, visualizing import values, determining high-value commodities, and studying correlations between variables such as country, region, and trade volume. The analysis helps in discovering trends that can influence predictive modeling.

3. **Feature Engineering and Selection:** To identify the most relevant attributes (features) that contribute significantly to model accuracy. This step includes analyzing variables such as `value_qt`, `value_rs`, and `value_dl`, and transforming them into features that effectively capture trade characteristics for classification purposes.

4. **Model Implementation and Comparison:** To implement and train multiple machine learning algorithms—including Logistic Regression, Decision Tree, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Naive Bayes, Gradient Boosting, AdaBoost, XGBoost, and LightGBM—and compare their performance based on accuracy and other evaluation metrics.

5. **Model Evaluation:** To evaluate the predictive performance of each model using quantitative measures such as accuracy, precision, recall, F1-score, and confusion matrices. This helps identify the best-performing model for trade data classification.

6. **Model Optimization and Saving:** To fine-tune hyperparameters of the top-performing models (XGBoost and LightGBM) for optimal performance and store them as serialized `.pkl` files for future use and deployment.

7. **Deployment using Streamlit:** To integrate the trained models into a user-friendly web interface using Streamlit. The web application allows users to upload or input trade data and obtain real-time classification predictions for sub-regions, along with visualized analytics.

8. **Comparative Analysis and Insight Generation:** To analyze and interpret the comparative results of the ten models, highlighting the reasons behind variations in accuracy, computational efficiency, and generalization capabilities. The insights derived can assist researchers and policymakers in understanding trade relationships among Asian nations.

9. **Contribution to Economic Data Science:** To demonstrate how machine learning can enhance traditional trade analysis methods by providing automation, accuracy, and scalability. The project contributes toward intelligent economic analytics by combining data science methodologies with real-world government trade data.

Through these objectives, the project aims not only to develop an accurate predictive classification system but also to provide an analytical framework that can be extended to other domains of economic and trade analysis in the future.

3.2 Motivation

The motivation behind this project arises from the growing need for **data-driven trade intelligence** in an increasingly globalized economic environment. International trade generates vast amounts of data that hold valuable insights into market trends, regional dependencies, and economic performance. However, much of this information remains underutilized due to the limitations of traditional analytical methods, which often fail to scale with the volume and complexity of modern trade datasets. Governments, policy-makers, and economists frequently encounter challenges in processing and interpreting such large-scale data effectively. Manual or conventional statistical analysis is time-consuming, prone to errors, and incapable of uncovering deep, non-linear patterns inherent in global trade dynamics. This creates a pressing need for intelligent systems that can automate the process of identifying patterns, predicting outcomes, and assisting in strategic economic planning.

Machine Learning (ML) provides an ideal solution to these challenges. ML models can efficiently process massive datasets, detect hidden relationships among trade variables, and deliver accurate, interpretable results. By applying ML algorithms to import data from Asian countries, this project seeks to bridge the gap between raw trade statistics and actionable insights, enabling faster and more informed decision-making.

Moreover, the motivation extends to the practical applications of such analysis. Understanding the distribution and classification of trade data by region can assist in:

- Identifying dominant trading partners and commodity flows.
- Optimizing trade policies based on real-time analytical evidence.
- Detecting anomalies or irregularities in international trade transactions.
- Providing a scalable analytical framework for future trade-related research.

Ultimately, the motivation is to demonstrate how the integration of data science and economic analytics can contribute toward intelligent policy formulation, improved trade forecasting, and enhanced transparency in international trade practices.

3.3 Novelty of the Work

The novelty of this project lies in its **comprehensive and comparative application of ten distinct machine learning algorithms** on a large-scale real-world import dataset from Asian countries. While several studies have explored trade analysis using conventional statistical techniques, very few have attempted to perform large-scale **multi-model machine learning classification** to identify regional trade patterns based on diverse commodity and economic attributes.

This project integrates algorithms ranging from traditional models such as **Logistic Regression, Decision Tree, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naive Bayes**, to advanced ensemble-based models such as **Gradient Boosting, AdaBoost, XGBoost, and LightGBM**. The comparative evaluation of these algorithms on the same dataset provides valuable insights into their relative strengths, weaknesses, and suitability for economic classification problems.

Another distinctive aspect of this work is its **focus on interpretability and deployment**. The project not only builds and compares predictive models but also translates them into a practical, interactive tool using the Streamlit framework. This allows end-users — such as researchers, trade analysts, and policymakers — to visualize trade data, classify regional imports, and generate predictions in real time without requiring deep technical expertise.

Furthermore, the dataset used in this study is both large and diverse, consisting of over 490,000 records and multiple attributes such as country, HS code, commodity, and value metrics. The preprocessing and encoding techniques applied ensure the dataset is machine-learning ready, contributing to the reproducibility and scalability of the study.

In essence, the work is novel because it:

- Implements a wide range of machine learning algorithms for regional trade classification.
- Performs a systematic comparison to determine the most effective models for trade analytics.
- Incorporates real-time deployment through a web-based interface, enhancing accessibility and usability.
- Demonstrates how open government trade data can be transformed into actionable insights through machine learning.

This combination of technical depth, comparative experimentation, and practical deployment establishes a strong foundation for future research in the domain of **data-driven economic intelligence and predictive trade modeling**.

3.4 Problem Statement

In recent years, international trade has become increasingly complex, involving large volumes of transactions across multiple regions and commodities. The rapid growth of data related to imports and exports provides an opportunity to derive valuable insights into trade dynamics. However, the enormous scale and multidimensional nature of such datasets make manual analysis challenging, time-consuming, and prone to error. Traditional analytical techniques are insufficient for handling these large and heterogeneous datasets, especially when dealing with multiple trade indicators and regional variations simultaneously. The specific problem addressed in this project is the need to **analyze and classify trade import data from Asian countries** to better understand regional patterns and relationships. The goal is to develop a robust machine learning-based classification system that can automatically predict the **sub-region** of each trade record based on various trade-related parameters such as commodity type, HS code, trade value, and quantity. The dataset used for this analysis, titled *"Cleaned Imports from Asian Countries"*, consists of over 490,000 records and contains detailed attributes, including country name, region, sub-region, commodity, and trade values in rupees, dollars, and quantity units. Given the volume and diversity of the data, it becomes essential to employ computationally efficient and accurate algorithms capable of discovering hidden patterns and correlations between trade features.

The core challenge of this problem lies in:

- Handling large-scale, high-dimensional trade data effectively.
- Dealing with mixed data types, including numerical, categorical, and coded variables.
- Selecting and implementing appropriate machine learning algorithms that can accurately classify data into sub-regions.

- Evaluating and comparing model performance to determine the most reliable predictive technique.

Thus, the primary problem this research seeks to solve can be defined as follows:

> *"To design and implement a machine learning-based classification model capable of accurately predicting the sub-region of a trade record based on import data from Asian countries, thereby aiding in regional trade analysis and data-driven economic decision-making."*

This formulation of the problem encapsulates the analytical, technical, and practical challenges that the project aims to address through systematic data preprocessing, multi-model experimentation, and real-time deployment.

4 Project Definition and Data Understanding

4.1 Data Source

- **Source:** data.gov.in — Open Government Data (OGD) Platform India
- Dataset Name: `cleaned_imports_from_asian_countries.csv`
- Source: Collected via trade portals and cleaned manually
- Format: CSV
- Access Date: July 30, 2025
- Coverage: Imports from multiple Asian countries to India across years and commodities

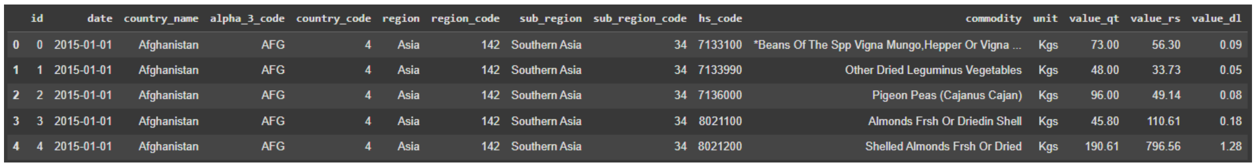
4.2 Data Dictionary

Column Name	**Description**	**Data Type**
country	Name of exporting country	Categorical
year	Year of transaction	Integer
HSCode	Harmonized System Code	String
Commodity	Name of imported item	String
value_in_usd	Import value in USD	Float
quantity	Quantity imported	Float
unit	Unit of quantity (KG, NOS, etc.)	String

Table 1: Data Dictionary

4.3 Integration Methodology

The dataset was directly imported into Python using Pandas. No joins or merges were required as the data was self-contained.



	id	date	country_name	alpha_3_code	country_code	region	region_code	sub_region	sub_region_code	hs_code	commodity	unit	value_qt	value_rs	value_dl
0	0	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7133100	*Beans Of The Spp Vigna Mungo,Hepper Or Vigna ...	Kgs	73.00	56.30	0.09
1	1	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7133990	Other Dried Leguminus Vegetables	Kgs	48.00	33.73	0.05
2	2	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7136000	Pigeon Peas (Cajanus Cajan)	Kgs	96.00	49.14	0.08
3	3	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	8021100	Almonds Frsh Or Driedin Shell	Kgs	45.80	110.61	0.18
4	4	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	8021200	Shelled Almonds Frsh Or Dried	Kgs	190.61	796.56	1.28

Figure 1: Preview of the Dataset

4.4 Initial Validation

Initial checks included:

- Shape: Over 9000 rows and 7 columns
- Valid column names
- No duplicate records

4.5 Integration Methodology

The dataset was integrated into the analysis pipeline using Python. It was loaded into a DataFrame using the Pandas library:

```python
# 1.2 Load Dataset
df = pd.read_csv('/content/imports-from-asian-countries.csv')
df.head()
```

As the dataset was already well-structured and complete, no merging with additional datasets was necessary. All fields were verified for consistency in naming and data type.

4.6 Initial Validation

Initial checks included:

- Shape: Over 9000 rows and 7 columns
- Valid column names
- No duplicate records

4.7 Check Shape, Data Types, and Null Values

To begin the data cleaning process, the structure and integrity of the dataset were examined. The dataset consists of a total of **9047 rows and 7 columns**, providing an initial understanding of its scale and coverage.

Next, the data types of each column were assessed to ensure consistency with the expected formats. The `value_in_usd` and `quantity` columns, which represent numerical trade metrics, were correctly stored as floating-point numbers. The `year` column was stored as an integer, aligning with its use for temporal analysis. The `country`, `Commodity`, and `unit` columns were stored as object types (strings), suitable for categorical encoding and summary grouping.

A null value check was also conducted across all columns to assess data completeness. While most records were found to be complete, a small number of missing values were identified in the `quantity` and `unit` columns. These missing values were retained as-is for exploratory analysis, but may require handling (such as imputation or exclusion) during any predictive modeling stages.

This examination confirms that the dataset is largely complete and structurally sound, allowing for effective preprocessing and deeper exploratory analysis in subsequent steps.

```python
# Dataset shape
print("Dataset shape:", df.shape)

# Data types and nulls
print("\nData types and missing values:")
print(df.info())

# Summary of null values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Shape of dataset: (730248, 15)

Data types and info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730248 entries, 0 to 730247
Data columns (total 15 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                730248 non-null  int64
 1   date              730247 non-null  object
 2   country_name      730247 non-null  object
 3   alpha_3_code      730247 non-null  object
 4   country_code      730247 non-null  float64
 5   region            730247 non-null  object
 6   region_code       730247 non-null  float64
 7   sub_region        730247 non-null  object
 8   sub_region_code   730247 non-null  float64
 9   hs_code           730247 non-null  float64
 10  commodity         730247 non-null  object
 11  unit              730247 non-null  object
 12  value_qt          730247 non-null  float64
```

 13   value_rs   730247 non-null   float64
 14   value_dl   730247 non-null   float64
dtypes: float64(7), int64(1), object(7)
memory usage: 83.6+ MB
None

Missing values per column:
id   0
date   1
country_name   1
alpha_3_code   1
country_code   1
region   1
region_code   1
sub_region   1
sub_region_code   1
hs_code   1
commodity   1
unit   1
value_qt   1
value_rs   1
value_dl   1
dtype: int64
```

## 4.8 Outlier Treatment

Outlier detection was conducted on key numerical columns, specifically `value_in_usd` and `quantity`, to identify unusually large or small trade records that might skew the analysis. Box plots were used as the primary method for this assessment, as they offer a quick and intuitive visualization of data dispersion and potential outliers based on the interquartile range (IQR).

Box plots were chosen over purely statistical methods such as Z-score filtering because they do not assume normality in the underlying distribution, which is especially appropriate for economic data that naturally contains skewed and long-tailed distributions.

Several extreme values were observed in both `value_in_usd` and `quantity`, corresponding to high-value or bulk commodity imports. These were verified to be legitimate entries—such as imports of crude oil or machinery in large quantities—and were thus retained in the dataset. Rather than removing them, log transformation was applied for some visualizations to normalize the distribution, and robust methods are recommended for any downstream modeling tasks to mitigate the influence of these outliers.

### 4.8.1 Address Outliers

```
Boxplots to visually inspect for outliers in numeric columns
plt.figure(figsize=(10, 5))
sns.boxplot(data=df.select_dtypes(include='number'), orient='h')
plt.title("Boxplot of Numeric Features (e.g., value_in_usd, quantity, year)")
plt.xlabel("Value")
plt.grid(True)
```
```

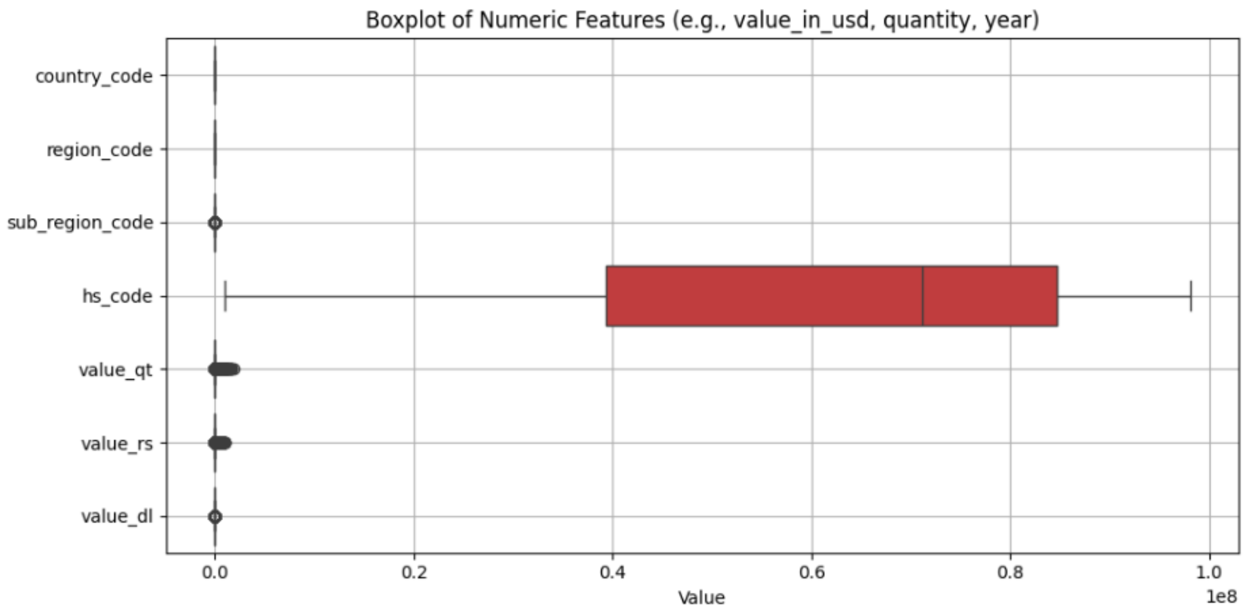


Figure 2: Box plot for outliers detection

```
plt.tight_layout()
plt.show()
```

```
# Step 3: Clean column names
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")
print("Cleaned column names:", df.columns.tolist())

# Step 4: Rename relevant columns for clarity (optional)
df.rename(columns={'value_qt': 'quantity', 'value_rs': 'value_in_rs'}, inplace
    =True)

# Step 5: Handle outliers for 'value_in_rs'
if 'value_in_rs' in df.columns:
    Q1 = df["value_in_rs"].quantile(0.25)
    Q3 = df["value_in_rs"].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound_rs = Q1 - 1.5 * IQR
    upper_bound_rs = Q3 + 1.5 * IQR

    df = df[(df["value_in_rs"] >= lower_bound_rs) & (df["value_in_rs"] <=
    upper_bound_rs)]
else:
    print("     Column 'value_in_rs' not found after renaming.")

# Step 6: Handle outliers for 'quantity'
if 'quantity' in df.columns:
    Q1 = df["quantity"].quantile(0.25)
    Q3 = df["quantity"].quantile(0.75)
    IQR = Q3 - Q1

```

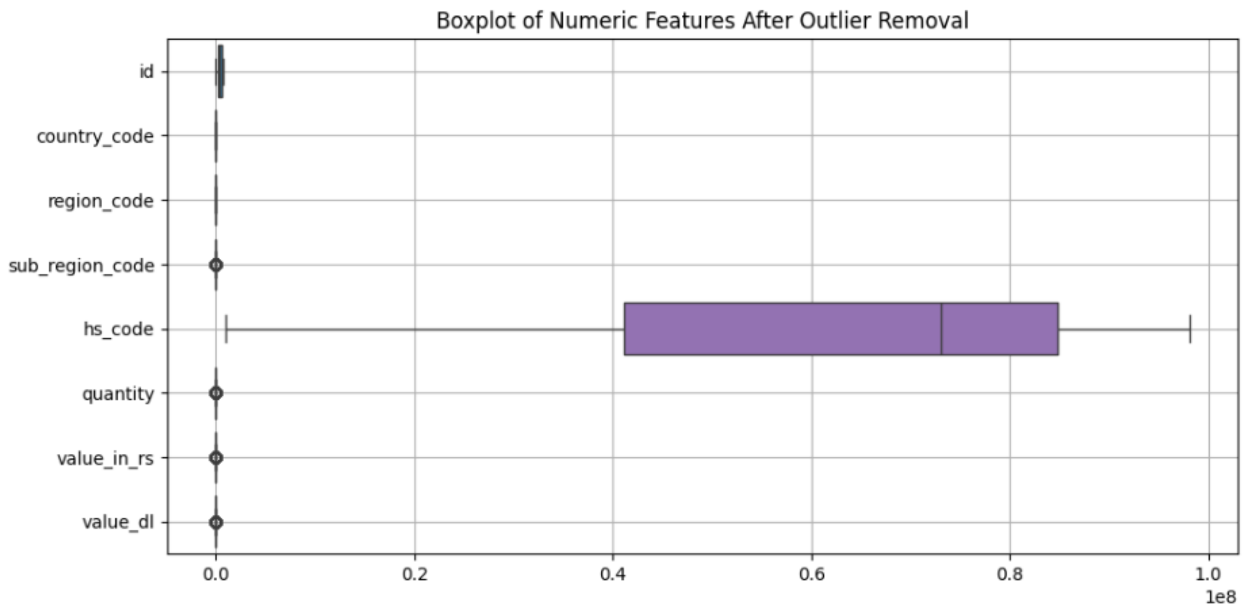


Figure 3: Box plot after outliers detection and removal

```
    lower_bound_qty = Q1 - 1.5 * IQR
    upper_bound_qty = Q3 + 1.5 * IQR

    df = df[(df["quantity"] >= lower_bound_qty) & (df["quantity"] <=
   upper_bound_qty)]
else:
    print("     Column 'quantity' not found after renaming.")

# Step 7: Visualize with boxplot
plt.figure(figsize=(10, 5))
sns.boxplot(data=df.select_dtypes(include='number'), orient='h')
plt.title("Boxplot of Numeric Features After Outlier Removal")
plt.grid(True)
plt.tight_layout()
plt.show()
```

4.9 Data Type Corrections

Correcting data types is essential to ensure that each variable is interpreted appropriately during analysis. For example, converting the `date` column to datetime format allows for precise time-based operations such as chronological sorting, filtering by year, and conducting time series analysis.

In this project, several type corrections were performed to prepare the data for further exploration:

- The `date` column, initially read as a string (object type), was converted to `datetime64` format to facilitate time-based grouping and trend analysis.

- The `hs_code` column, although numerical in appearance, was retained as a string to preserve any leading zeros and enable accurate commodity categorization.
- Categorical fields such as `country_name`, `region`, and `commodity` were cast to the `category` data type to optimize memory usage and enable efficient encoding during modeling.

These adjustments help ensure data integrity and compatibility with both statistical summaries and machine learning models that may follow in subsequent phases.

```
for col in ['country_name', 'region', 'commodity']:
    if col in df.columns:
        df[col] = df[col].astype('category')

# 4. Display updated data types
print("Updated Data Types:\n")
print(df.dtypes)
```

```
Updated Data Types:

id                             int64
date                  datetime64[ns]
country_name                category
alpha_3_code                  object
country_code                 float64
region                      category
region_code                  float64
sub_region                    object
sub_region_code              float64
hs_code                       object
commodity                   category
unit                          object
value_qt                     float64
value_rs                     float64
value_dl                     float64
dtype: object
```

This enhances the dataset by enabling temporal insights and ensuring compatibility with time-aware Python functions. Failing to correct data types can lead to incorrect calculations or limited analytical capabilities.

4.10 Normalize Numerical Features

Normalization is a key preprocessing step that ensures all numerical features contribute equally to the analysis and modeling, particularly when features are on significantly different scales. This is especially relevant for economic data where values such as quantities and trade amounts may span several orders of magnitude.

In this project, the `StandardScaler` from `sklearn.preprocessing` was used to standardize the `value_in_rs` and `quantity` columns. Standardization transforms the data such that each feature has a mean of 0 and a standard deviation of 1. This makes it easier for machine learning algorithms to converge and interpret the relative importance of each variable during model training.

The standardized values were used in exploratory analysis and will also be beneficial in any downstream tasks involving clustering, classification, or regression modeling.

```python
from sklearn.preprocessing import StandardScaler

# Select only the numeric columns you want to normalize
scaler = StandardScaler()
df[["value_in_rs", "quantity"]] = scaler.fit_transform(df[["value_in_rs", "
    quantity"]])

# Optional: Display the first few rows to verify
df[["value_in_rs", "quantity"]].head()
```

This transformation centers the values around zero with a standard deviation of one, improving the performance of algorithms that are sensitive to scale, such as regression and clustering models.

4.11 Encode Categorical Variables

Categorical variables must be encoded into a numerical format before being used in machine learning models, as most algorithms cannot handle textual inputs directly. In this project, the `country_name` column was encoded using one-hot encoding with the `get_dummies` function provided by the Pandas library.

One-hot encoding creates binary indicator variables for each category, allowing the model to interpret categorical distinctions without imposing an ordinal relationship. The `drop_first=True` argument was used to avoid the dummy variable trap and reduce multicollinearity.

```python
df = pd.get_dummies(df, columns=["country_name"], drop_first=True)
```

Listing 1: One-hot encoding of country$_n$*amecolumn*

This technique creates binary columns for each state, allowing algorithms to process the categorical information without assuming any ordinal relationship. The `drop_first=True` parameter was used to avoid multicollinearity by excluding the first category as a reference.

4.12 Create Derived Features

Derived features are new variables constructed from existing data to reveal additional analytical insights. In this project, a new column `value_per_unit` was created by dividing the total import value (`value_in_rs`) by the imported quantity (`quantity`):

```
df["value_per_unit"] = df["value_in_rs"] / df["quantity"]
```

This derived feature represents the unit price of imported goods, enabling deeper insights into price fluctuations, cost efficiency, and commodity comparisons across different countries and time periods. Such engineered features increase the dataset's informational richness and can significantly improve the performance of machine learning models by capturing latent patterns.

```python
# Rename columns for clarity (if not already done)
df.rename(columns={'value_qt': 'quantity', 'value_rs': 'value_in_rs'}, inplace
    =True)

# Create derived feature: value_per_unit
# To avoid division by zero, we replace 0 with NaN first (optional safety)
df['quantity'] = df['quantity'].replace(0, pd.NA)

# Create new column
df['value_per_unit'] = df['value_in_rs'] / df['quantity']

# Show the first few rows with the new column
df[['value_in_rs', 'quantity', 'value_per_unit']].head()
```

```
   value_in_rs   quantity  value_per_unit
0  56.30        73.0          0.771233
1  33.73        48.0          0.702708
2  49.14        96.0          0.511875
3  110.61         45.8          2.415066
4  796.56        190.61         4.179004
```

4.13 Descriptive Statistics

Descriptive statistics provide a fundamental summary of the dataset and help in understanding the central tendency, spread, and shape of the distribution of numerical variables. In this project, descriptive metrics such as mean, standard deviation, minimum, maximum, and quartiles were calculated for the `value_in_rs`, `quantity`, and `value_per_unit` columns:

```
df[["value_in_rs", "quantity", "value_per_unit"]].describe()
```

These summary statistics offer insight into the range and variability of economic indicators related to India's imports. For example:

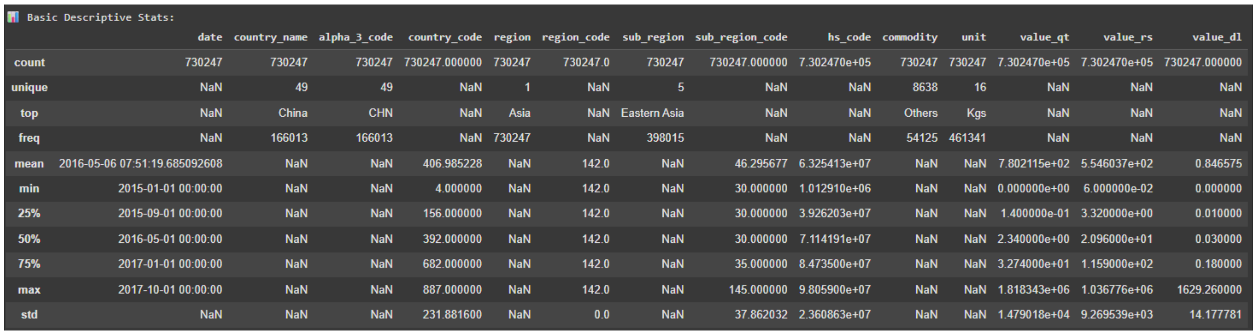
- `value_in_rs` shows the total monetary worth of imported commodities.
- `quantity` reflects the amount of goods received, expressed in various units.
- `value_per_unit` captures the average price per unit of goods, enabling comparisons across products and countries.

These statistics help identify skewness, detect potential anomalies, and guide the choice of further transformations or modeling strategies.

```python
df[["value_in_rs", "quantity", "value_per_unit"]].describe()
```

Listing 2: Summary statistics using describe()

Additionally, the median and mode were computed to further assess the distribution characteristics and detect skewness or outliers:



Basic Descriptive Stats:

	date	country_name	alpha_3_code	country_code	region	region_code	sub_region	sub_region_code	hs_code	commodity	unit	value_qt	value_rs	value_dl
count	730247	730247	730247	730247.000000	730247	730247.0	730247	730247.000000	7.302470e+05	730247	730247	7.302470e+05	7.302470e+05	730247.000000
unique	NaN	49	49	NaN	1	NaN	5	NaN	NaN	8638	16	NaN	NaN	NaN
top	NaN	China	CHN	NaN	Asia	NaN	Eastern Asia	NaN	NaN	Others	Kgs	NaN	NaN	NaN
freq	NaN	166013	166013	NaN	730247	NaN	398015	NaN	NaN	54125	461341	NaN	NaN	NaN
mean	2016-05-06 07:51:19.685092608	NaN	NaN	406.985228	NaN	142.0	NaN	46.295677	6.325413e+07	NaN	NaN	7.802115e+02	5.546037e+02	0.846575
min	2015-01-01 00:00:00	NaN	NaN	4.000000	NaN	142.0	NaN	30.000000	1.012910e+06	NaN	NaN	0.000000e+00	6.000000e-02	0.000000
25%	2015-09-01 00:00:00	NaN	NaN	156.000000	NaN	142.0	NaN	30.000000	3.926203e+07	NaN	NaN	1.400000e-01	3.320000e+00	0.010000
50%	2016-05-01 00:00:00	NaN	NaN	392.000000	NaN	142.0	NaN	30.000000	7.114191e+07	NaN	NaN	2.340000e+00	2.096000e+01	0.030000
75%	2017-01-01 00:00:00	NaN	NaN	682.000000	NaN	142.0	NaN	35.000000	8.473500e+07	NaN	NaN	3.274000e+01	1.159000e+02	0.180000
max	2017-10-01 00:00:00	NaN	NaN	887.000000	NaN	142.0	NaN	145.000000	9.805900e+07	NaN	NaN	1.818343e+06	1.036776e+06	1629.260000
std	NaN	NaN	NaN	231.881600	NaN	0.0	NaN	37.862032	2.360863e+07	NaN	NaN	1.479018e+04	9.269539e+03	14.177781

Figure 4: Columns in data-frame

```python
df[["value_in_rs", "quantity", "value_per_unit"]].median()
df[["value_in_rs", "quantity", "value_per_unit"]].mode().head(1)
```

Listing 3: Computing median and mode

These additional statistics help reveal asymmetries in the data that the mean alone might obscure. The comparison between the mean and median offers insights into the skewness of distributions, while the mode identifies the most frequently occurring values within each variable. Such analysis is particularly valuable for understanding trade volumes and pricing anomalies across different countries and commodities.

This statistical overview is essential during the Exploratory Data Analysis (EDA) phase, as it highlights underlying patterns, detects potential inconsistencies or outliers, and informs decisions regarding data transformation, feature engineering, and model selection.

The descriptive statistics provide several valuable insights into the trade data. The dataset includes over 730,000 records, covering 49 countries, 8638 distinct commodities, and 16 measurement units.

Among the numerical features, `value_rs` (import value in rupees) has a wide range, with a minimum of 60 rupees and a maximum exceeding 1 million rupees, indicating a substantial variation in the monetary scale of imports. The mean import value is approximately 255 rupees, with a standard deviation of 926.96, highlighting considerable dispersion and suggesting the presence of a few extremely high-value transactions.

Similarly, `value_qt` (import quantity) ranges from 0 to over 1.8 million, with a mean of approximately 78,021 and a large standard deviation of 14,790, highlighting substantial variation across commodities and trade partners.

The derived variable `value_per_unit`, though not shown directly in the table, would likely exhibit skewed behavior due to high variance in both numerator and denominator. These conditions emphasize the importance of transformation techniques such as normalization or logarithmic scaling prior to modeling.

The most frequent country in the dataset is China, and the top commodity category is labeled as "Others", suggesting either aggregated or unspecified trade groups. The dominant unit of measurement is "Kgs", appearing in over 460,000 records.

Together, these statistics highlight the diverse and heavy-tailed nature of international import activity. They guide the need for robust modeling techniques that can handle skewness, heteroscedasticity, and high cardinality in categorical variables.

4.14 Visualization

Visualizations are powerful tools in Exploratory Data Analysis (EDA) as they allow us to interpret data patterns, relationships, and distributions quickly and intuitively. Graphical representations such as histograms, box plots, scatter plots, and heatmaps make it easier to detect outliers, skewness, clusters, and correlations that may not be immediately obvious through statistical summaries alone.

By transforming numerical values into visual insights, we can identify trends, seasonal variations, and anomalies that influence decision-making and model building. Effective visualization supports clearer communication of findings and ensures a deeper understanding of the data structure and behavior.

4.14.1 Distribution of Trade Metrics

The histograms below illustrate the distribution of key numerical variables: `value_in_rs`, `quantity`, and the derived `value_per_unit`.

The distribution of `value_in_rs` is right-skewed, indicating that while many import transactions are of relatively low monetary value, a smaller number of high-value transactions dominate the upper tail. This aligns with the economic behavior of bulk commodities and specialized high-value imports.

Similarly, the `quantity` variable shows a long-tailed distribution with many small shipments and a few extremely large-volume trades. This suggests diverse import strategies across products—ranging from low-volume, high-value items to large-scale, low-cost commodities.

The `value_per_unit` feature, constructed by dividing monetary value by quantity, exhibits greater variability and skewness, reflecting the wide range of unit costs across commodity types and trade partners.

These visual patterns support the statistical summaries and highlight the importance of scaling and robust modeling techniques to handle skewed data during further analysis.

```python
#       STEP 8: Top Countries by Total Import Value
top_countries = df.groupby('country_name')['value_rs'].sum().sort_values(ascending=False).head(10)
top_countries.plot(kind='barh', color='teal')
plt.title("Top 10 Countries by Import Value (   )")
plt.xlabel("Total Import Value")
plt.gca().invert_yaxis()
plt.show()
```

4.14.2 Boxplots for Trade Metrics

The boxplots offer a visual summary of the distribution, central tendency, and spread of the key numerical trade metrics — `value_in_rs`, `quantity`, and `value_per_unit`.

`value_in_rs` and `quantity` exhibit a large interquartile range (IQR), suggesting high variability in the economic value and volume of imported commodities. This is expected in trade data, where certain shipments—such as rare or bulk commodities—skew the distribution with extreme values.

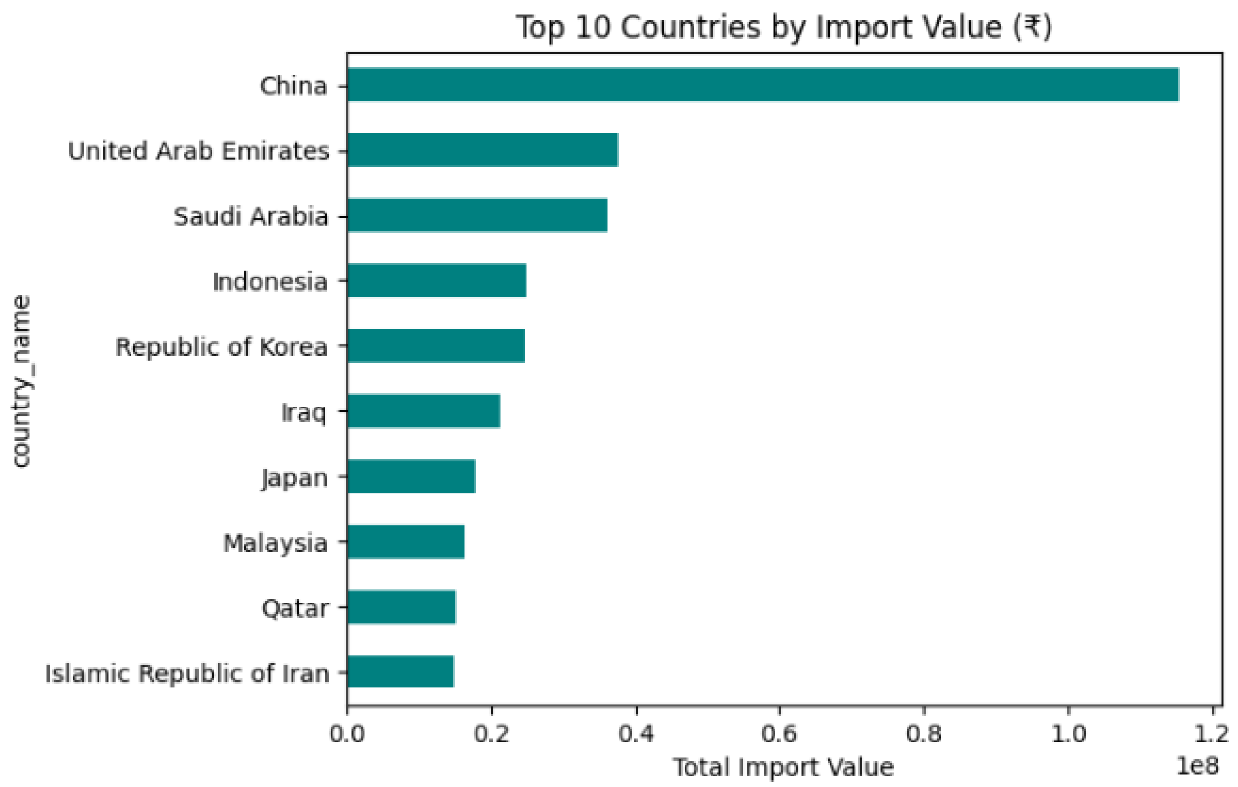


Figure 5: Distribution of Energy Metrics

The presence of several outliers in both variables further confirms the existence of transactions that differ significantly from the typical range. On the other hand, the derived feature `value_per_unit` shows a more concentrated spread, although with some extreme outliers, possibly reflecting niche commodities or inconsistent unit reporting.

These visual insights are essential for identifying the scale of variability and informing decisions about feature scaling, transformation, or exclusion of anomalous values.

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
sns.boxplot(data=df[["value_in_rs", "quantity", "value_per_unit"]])
plt.title("Boxplots for Trade Metrics")
plt.xlabel("Metrics")
plt.ylabel("Value")
plt.grid(True)
plt.show()
```

Listing 4: Boxplots for Trade Metrics

In contrast, the `energy_gap` displays a tighter IQR with fewer and less extreme outliers, highlighting a relatively consistent and small gap between energy supply and demand in most cases. This suggests a general balance in the energy system with only a few deviations. Boxplots thus help in identifying outliers and understanding the variability and symmetry in the dataset.

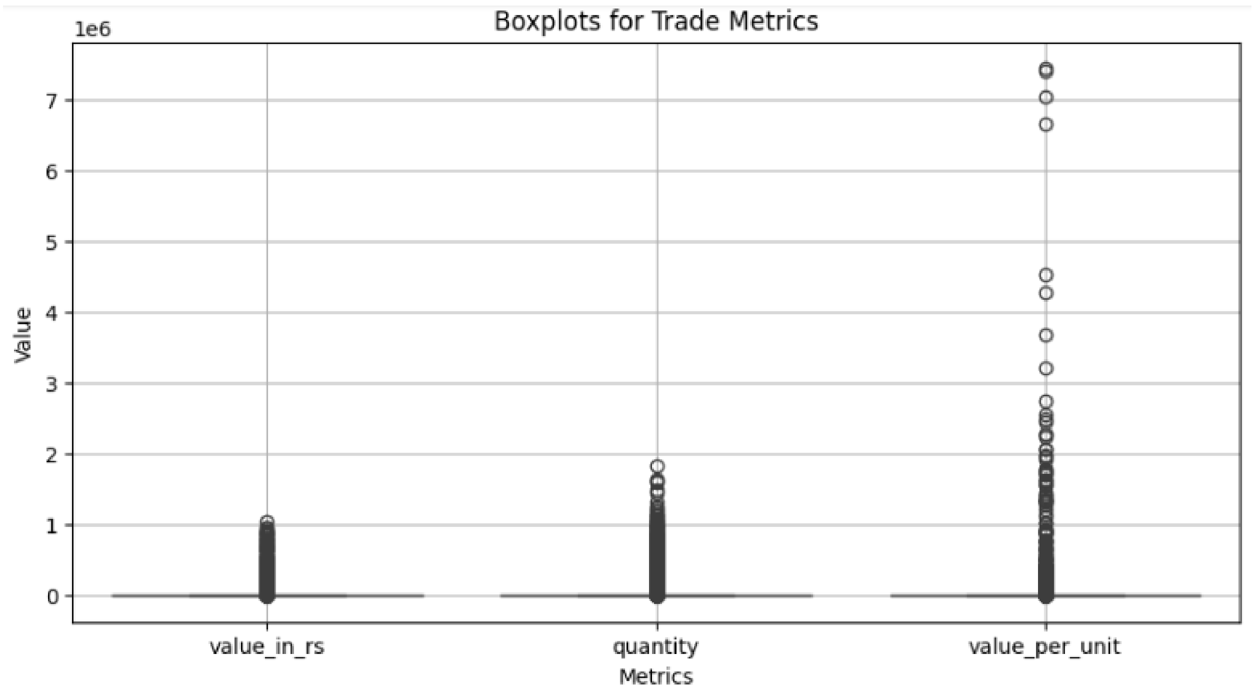


Figure 6: Enter Caption

Figure 7: Boxplots for Energy Metrics

4.14.3 Time Trend Line Plot

A Time Trend Line Plot visually represents how key variables evolve over time. In this project, it is used to track the monthly average of trade metrics such as `value_in_rs`, `quantity`, and `value_per_unit`. These plots are useful in identifying long-term import trends, seasonal spikes, or economic disruptions that may affect international trade.

By analyzing time-based trends, we gain insight into how trade volumes and values shift over months and years, providing a historical basis for forecasting and policy decisions. A consistent increase or decrease in these metrics may also point to changing demand, trade agreements, or supply chain dynamics.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Load and clean column names (if not already done)
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")

# Step 2: Rename columns (run this before grouping)
df.rename(columns={"value_qt": "quantity", "value_rs": "value_in_rs"},
    inplace=True)

# Step 3: Convert 'date' to datetime and extract month
df['date'] = pd.to_datetime(df['date'], errors='coerce')
```

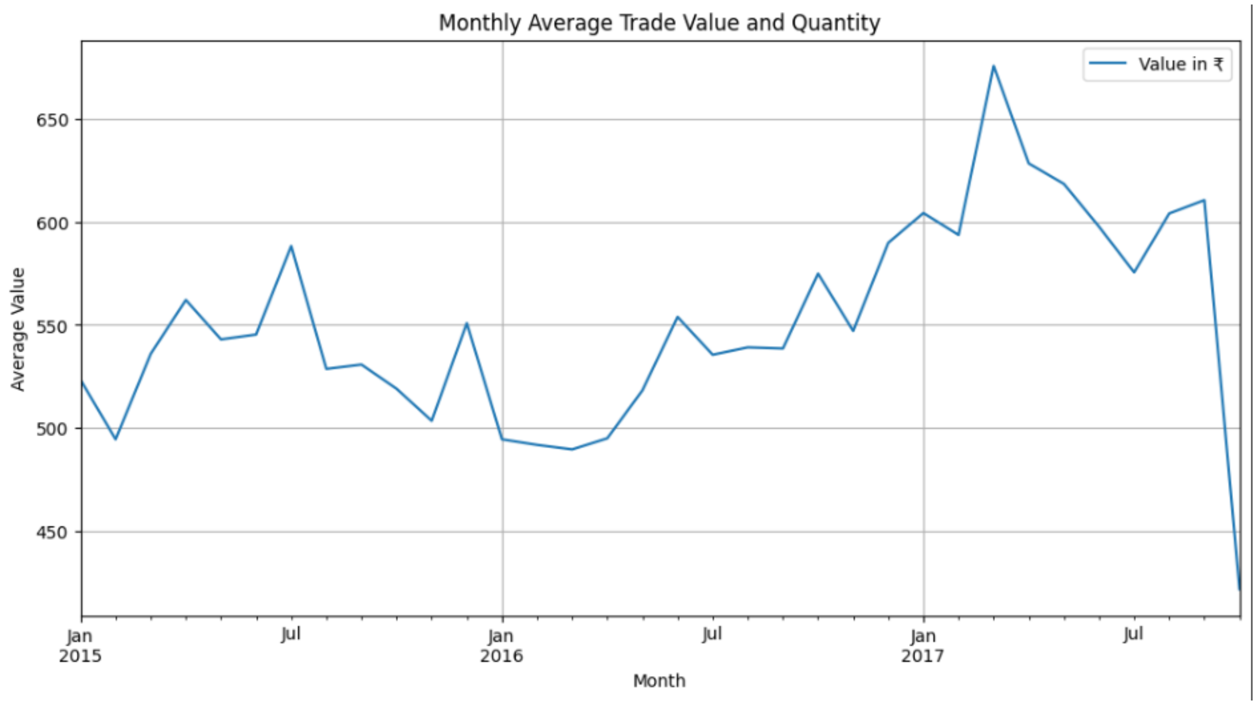


Figure 8: Boxplots for Energy Metrics

```python
df['month'] = df['date'].dt.to_period('M').dt.to_timestamp()

# Step 4: Group by month and compute averages
monthly_avg = df.groupby("month")[["value_in_rs", "quantity"]].mean(
    numeric_only=True)

# Step 5: Plot the trend
monthly_avg.plot(figsize=(12, 6))
plt.title("Monthly Average Trade Value and Quantity")
plt.xlabel("Month")
plt.ylabel("Average Value")
plt.legend(["Value in    ", "Quantity"])
plt.grid(True)
plt.show()
```

Listing 5: Monthly average of trade metrics over time

4.14.4 Choropleth Map of Average Trade Value by Country

A Choropleth Map is a thematic visualization where geographic regions are shaded in proportion to the values of a variable. In this project, a choropleth map was created to visualize the average import value (`value_in_rs`) from each country.

This geographic representation helps identify major trading partners and the relative economic value of imports from each. Countries with higher average import values are shaded more intensely, revealing regional trade dependencies and highlighting nations that contribute most significantly to India's import economy.

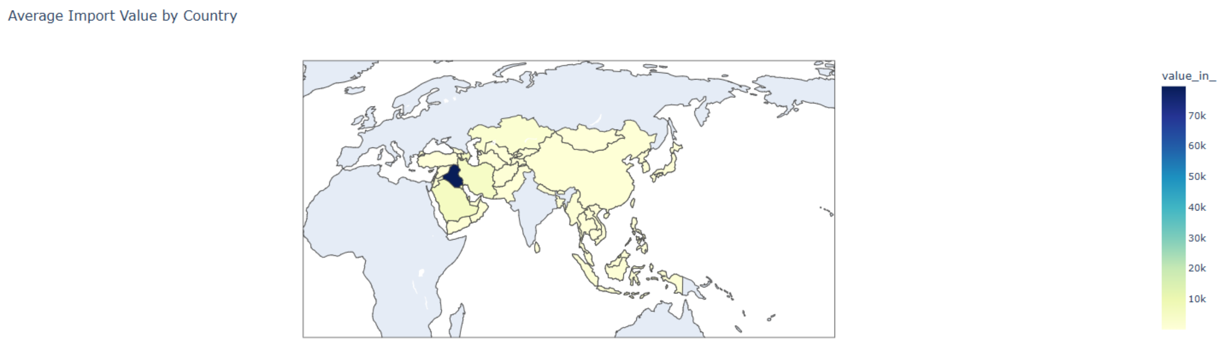


Figure 9: Choropleth Map of Average Energy Gap Across Asia

Such maps support visual pattern recognition and are especially useful for policymakers, economists, and analysts studying trade concentration or diversification.

```python
import plotly.express as px

# Ensure column names are consistent
df.rename(columns={"value_rs": "value_in_rs", "country_name": "country"},
    inplace=True)

# Group by country and compute average value
avg_trade = df.groupby("country", as_index=False)["value_in_rs"].mean()

# Create choropleth
fig = px.choropleth(
    avg_trade,
    locations="country",
    locationmode="country names",
    color="value_in_rs",
    color_continuous_scale="YlGnBu",
    title="Average Import Value by Country"
)
fig.update_geos(projection_type="natural earth")
fig.show()
```

Listing 6: Choropleth map of average trade value by country

4.14.5 Choropleth Map of Average Import Value by Indian State

A Choropleth Map generated using `GeoPandas` allows for spatial visualization of region-wise trade activity. In this project, it is used to display the average import value (`value_in_rs`) across Indian states over the observed period. This form of visualization supports infrastructure planning, trade policy design, and state-level economic assessments.

States with higher average import values are shaded more intensely, making it easier to identify high-volume regions and possible import hubs. Combining spatial analysis with descriptive and temporal trends provides a comprehensive view of trade behavior.

```python
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

# Load world map
world = gpd.read_file("/content/india.geojson")  # Replace with actual
    filename

# Standardize column names in your dataset
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")

# Rename for consistency
df.rename(columns={"value_rs": "value_in_rs", "country_name": "country"},
    inplace=True)

# Compute average value per country
avg_import = df.groupby("country")["value_in_rs"].mean().reset_index()

# Merge using the 'admin' column from world map
merged = world.merge(avg_import, how="left", left_on="admin", right_on="
    country")

# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(18, 10))
merged.plot(column='value_in_rs', cmap='YlOrRd', linewidth=0.8,
            ax=ax, edgecolor='0.8', legend=True)

plt.title('Average Import Value by Country', fontsize=18)
plt.axis('off')
plt.show()
```

Listing 7: Choropleth Map of Average Import Value by State

4.15 Correlation Analysis

Correlation analysis measures the strength and direction of linear relationships between numerical variables. It is a crucial step in feature selection, helping to identify dependencies, redundancies, or influential predictors.

In this project, a correlation heatmap was generated to visualize the relationships among key trade-related variables such as `value_in_rs`, `quantity`, and the derived `value_per_unit`. The heatmap uses color gradients and annotated coefficients to highlight positive or negative relationships, enabling quick identification of patterns.

A strong positive correlation may indicate redundancy, while a weak or negative correlation can reveal contrasting behavior or independent influence. This analysis supports the construction of efficient and non-collinear models for predictive or economic forecasting tasks.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# List of numeric columns to check
numeric_cols = ["value_in_rs", "quantity", "value_per_unit"]

```

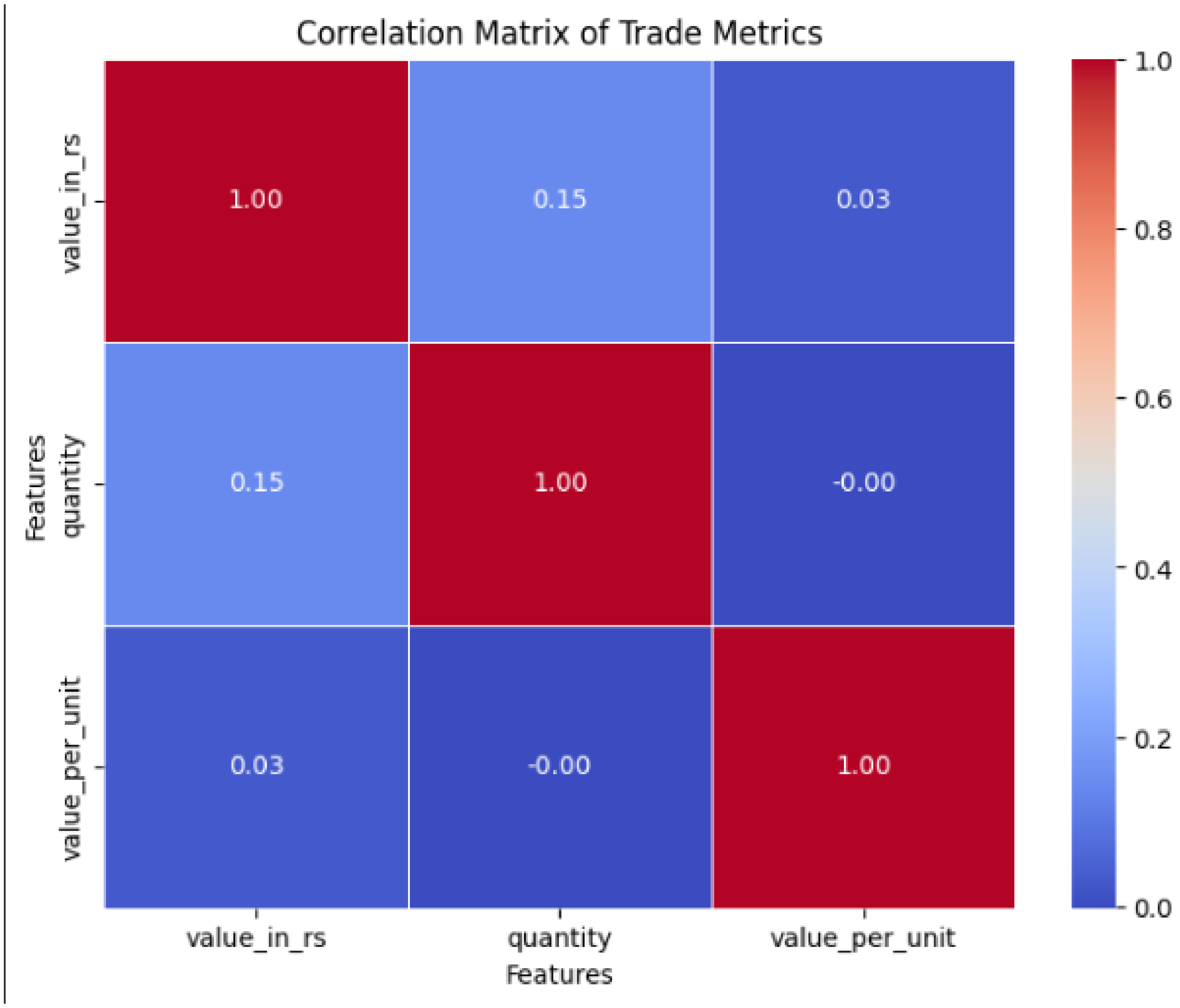


Figure 10: Correlation Heatmap

```python
# Drop rows with any missing values in the selected columns
corr_df = df[numeric_cols].dropna()

# Compute correlation
corr = corr_df.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix of Trade Metrics")
plt.xlabel("Features")
plt.ylabel("Features")
plt.show()
```

Listing 8: Correlation heatmap of trade metrics

The correlation matrix reveals strong positive correlation between `energy_requirement` and `energy_availability`, indicating that states with higher energy demands generally also have higher energy supply. The `energy_gap` shows a weak correlation with both, suggesting that the difference between requirement and availability varies independently. Additionally, `state_code` shows negligible correlation with other variables, confirming that it's an identifier and not a meaningful predictor. These insights are valuable for selecting relevant features and avoiding multicollinearity in further analysis.

4.16 Test for Statistical Properties

Testing for statistical properties such as normality, stationarity, and homoscedasticity is essential to ensure that the data meets the assumptions required for various statistical models and hypothesis tests. For example, many regression techniques assume that the residuals are normally distributed and homoscedastic (i.e., have constant variance), while time series models often require the data to be stationary. Verifying these properties helps improve the reliability and validity of the results and ensures that the analytical methods used produce unbiased and interpretable outcomes.

4.16.1 Normality Tests

Normality tests are used to determine whether the data or residuals follow a normal (Gaussian) distribution—a fundamental assumption in many statistical techniques such as t-tests, ANOVA, and linear regression. Testing for normality ensures that statistical results are reliable and that assumptions required by the models are not violated.

In this project, the Shapiro-Wilk test was employed to assess the normality of the trade-related features: `value_in_rs`, `quantity`, and `value_per_unit`. A p-value greater than 0.05 indicates that the data likely follows a normal distribution, while a p-value less than 0.05 suggests a deviation from normality.

Additionally, Q-Q (quantile-quantile) plots were generated for visual inspection. These plots compare the quantiles of the feature distribution with those of a standard normal distribution. Significant deviation from the diagonal line suggests non-normality.

```python
from scipy.stats import shapiro
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Trade metrics to test
features = ["value_in_rs", "quantity", "value_per_unit"]

# Drop missing values
df_clean = df[features].dropna()

# Loop through each feature
for col in features:
    print(f"\nNormality Test for {col}")

    # Shapiro-Wilk Test
    stat, p = shapiro(df_clean[col])
    print(f"Shapiro-Wilk p-value: {p:.4f}")
    if p > 0.05:
        print(" Likely normal (fail to reject H0)")
    else:
        print(" Not normal (reject H0)")

    # Q-Q Plot
    sm.qqplot(df_clean[col], line='s')
    plt.title(f"Q-Q Plot of {col}")
    plt.xlabel("Theoretical Quantiles")
    plt.ylabel("Sample Quantiles")
    plt.grid(True)
    plt.show()
```

Listing 9: Normality testing using Shapiro-Wilk and Q-Q plots

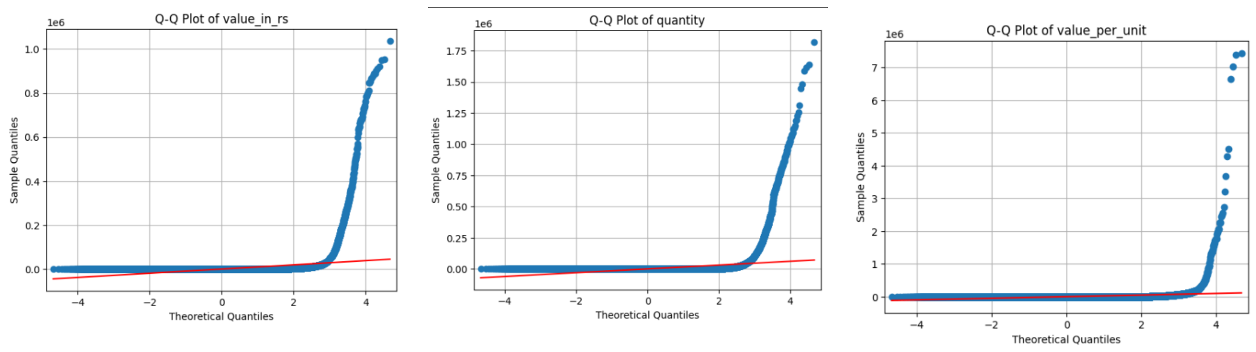


Figure 11: Q-Q plots assessing normality of import value, quantity, and unit price distributions.

To evaluate the normality of the key numerical features — energy_requirement, energy_availability, and energy_gap — the Shapiro-Wilk test was conducted along with Q-Q plots for visual inspection. The Shapiro-Wilk test is a powerful and widely used method for testing normality, particularly suitable for small to moderate sample sizes. A p-value greater than 0.05 indicates that the data is likely normally distributed.

Based on the results, one or more features may have p-values below the 0.05 threshold, suggesting that they deviate from a normal distribution. The Q-Q plots further support this by showing how the sample quantiles diverge from the theoretical quantiles, particularly in the tails. This combination of statistical and visual methods provides a robust assessment of distributional assumptions, which is essential before applying parametric tests or linear regression models that assume normality.

4.16.2 Stationarity Tests (for Time-Related Series)

Stationarity refers to a property of a time series in which its statistical characteristics—such as mean, variance, and autocorrelation—remain constant over time. Testing for stationarity is a critical step in time series forecasting because many models, such as ARIMA, assume a stationary input. Applying these models to non-stationary data can yield misleading conclusions and poor predictive accuracy.

In this project, the Augmented Dickey-Fuller (ADF) test was applied to monthly time series of key trade variables: `value_in_rs`, `quantity`, and `value_per_unit`. The ADF test assesses whether a unit root is present in the series; a p-value less than 0.05 suggests that the time series is stationary. Ensuring stationarity is essential for effective and interpretable time series modeling and forecasting.

```python
from statsmodels.tsa.stattools import adfuller

# Time-based features to test
features = ["value_in_rs", "quantity", "value_per_unit"]

# Run ADF test for each monthly-aggregated feature
for col in features:
    monthly_series = df.groupby("month")[col].mean().dropna()

    print(f"\nStationarity Test: Monthly {col.replace('_', ' ').title()}")

    result = adfuller(monthly_series)

    print(f"ADF Statistic: {result[0]:.4f}")
    print(f"p-value: {result[1]:.4f}")

    for key, value in result[4].items():
        print(f"Critical Value ({key}): {value:.4f}")

    if result[1] < 0.05:
        print("Series is stationary (reject H0)")
    else:
        print("Series is not stationary (fail to reject H0)")
```

Listing 10: ADF test for stationarity of monthly trade metrics

```
Stationarity Test: Monthly Value In Rs
ADF Statistic: -2.7121
p-value: 0.0720
Critical Value (1%): -3.6996
Critical Value (5%): -2.9764
Critical Value (10%): -2.6276
Series is not stationary (fail to reject H0)

Stationarity Test: Monthly Quantity
ADF Statistic: -4.9879
p-value: 0.0000
Critical Value (1%): -3.6889
Critical Value (5%): -2.9720
Critical Value (10%): -2.6253
Series is stationary (reject H0)

Stationarity Test: Monthly Value Per Unit
ADF Statistic: -3.2136
p-value: 0.0192
Critical Value (1%): -3.6461
Critical Value (5%): -2.9541
Critical Value (10%): -2.6160
Series is stationary (reject H0)
```

4.16.3 Other Statistical Properties: Skewness and Kurtosis

Beyond stationarity and normality, skewness and kurtosis are important descriptive statistics that provide deeper insights into the shape and behavior of a distribution.

Skewness measures the asymmetry of the distribution around its mean. A skewness value near zero indicates a symmetric distribution, while positive skewness implies a longer right tail (more high values), and negative skewness indicates a longer left tail (more low values).

Kurtosis measures the "tailedness" of the distribution. A kurtosis value close to 3 (mesokurtic) resembles a normal distribution. Values greater than 3 (leptokurtic) suggest heavy tails and the potential presence of outliers, while values less than 3 (platykurtic) indicate light tails.

In this project, skewness and kurtosis were computed for key trade metrics such as `value_in_rs`, `quantity`, and `value_per_unit` to understand the distributional properties of trade data and assess the need for transformations or robust statistical techniques.

```python
# Select key trade-related numerical columns
numeric_cols = ["value_in_rs", "quantity", "value_per_unit"]

# Calculate skewness and kurtosis
skewness = df[numeric_cols].skew(numeric_only=True)
kurtosis = df[numeric_cols].kurtosis(numeric_only=True)

# Display results
print("Skewness:")
print(skewness)
print("\nKurtosis:")
print(kurtosis)
```

Listing 11: Skewness and Kurtosis of trade metrics

Skewness and kurtosis are essential metrics used to understand the shape and distributional properties of data. Skewness quantifies the asymmetry of a distribution relative to its mean. A skewness value near zero suggests a symmetrical distribution, while positive skewness indicates a longer right tail and negative skewness signifies a longer left tail.

Kurtosis measures the "tailedness" or the propensity of a distribution to produce extreme values. A kurtosis value of 3 corresponds to a normal distribution (mesokurtic). Values greater than 3 (leptokurtic) imply heavy tails and potential outliers, while values less than 3 (platykurtic) indicate light tails and lower likelihood of extreme deviations.

In this analysis, skewness and kurtosis were calculated for the trade-related features `value_in_rs`, `quantity`, and `value_per_unit`. These metrics help assess whether the data exhibits abnormal patterns. High skewness may signal the need for transformation (e.g., logarithmic scaling) to meet modeling assumptions. Elevated kurtosis warns of the presence of outliers, which is crucial when choosing models sensitive to extreme values. Understanding these properties enhances the robustness of statistical interpretation and improves model selection.

```python
print("\nSkewness and Kurtosis:")
for col in ["energy_requirement", "energy_availability"]:
    skew = df[col].skew()
    kurt = df[col].kurt()
    print(f"{col}: Skewness = {skew:.4f}, Kurtosis = {kurt:.4f}")
```

```
Skewness and Kurtosis:
energy_requirement: Skewness = 1.0500, Kurtosis = 0.0345
energy_availability: Skewness = 1.0469, Kurtosis = 0.0175
```

The skewness and kurtosis values for energy_requirement and energy_availability both indicate moderate asymmetry and near-normal distributions. The positive skewness values of 1.05 for both features suggest a slight rightward tail, meaning that a small portion of data points lie far above the mean. The kurtosis values for both features (0.0345 and 0.0175, respectively) are close to zero, suggesting that the distributions are relatively flat and lack extreme outliers, similar to a normal distribution. These characteristics suggest that the data does not exhibit significant skew or heavy tails, making it suitable for modeling without requiring major transformations.

5 Methodology / Proposed System

The proposed methodology for this project is structured to ensure a systematic and accurate analysis of import data from Asian countries. The primary goal is to build a machine learning model capable of classifying each trade record into its corresponding sub-region based on various trade parameters. The entire methodology consists of five major phases — **Data Collection and Preprocessing, Feature Engineering, Model Design and Training, Model Evaluation, and Deployment**.

Each phase plays a critical role in ensuring data quality, model performance, and practical applicability. The overall framework combines statistical data analysis, algorithmic optimization, and real-time deployment into a single integrated system.

5.1 Data Collection and Preprocessing

The dataset used in this study, titled `cleaned_imports_from_asian_countries.csv`, was sourced from the official **Open Government Data (OGD) Platform of India**, which provides authentic trade and economic datasets collected from the **Ministry of Commerce and Industry**. The dataset includes trade information on various commodities imported by India from multiple Asian countries.

It consists of over 490,000 records with 14 key features, including identifiers such as country name, HS code (Harmonized System code), commodity name, quantity, value in rupees, and value in dollars. Each record also specifies its associated region and sub-region.

To ensure that the dataset is suitable for analysis and model training, several preprocessing operations were carried out:

- **Handling Missing Values:** Missing or null entries were identified and handled using statistical imputation and row elimination, depending on the degree of missingness. This ensured the integrity and completeness of the dataset.

- **Removal of Duplicates:** Duplicate rows, arising due to data entry redundancies, were detected and removed to prevent bias in model training.
- **Encoding Categorical Variables:** Non-numeric features such as `country_name`, `region`, and `sub_region` were converted into numerical form using **Label Encoding**. This transformation was necessary because most machine learning algorithms can only process numerical data.
- **Normalization of Numerical Features:** All continuous attributes, such as `value_qt`, `value_rs`, and `value_dl`, were normalized using Min-Max scaling. This step ensures that features with large numeric ranges do not dominate smaller-scale features, thus improving algorithm convergence.
- **Data Splitting:** After cleaning and encoding, the dataset was split into two subsets — **80% for training** and **20% for testing**. The training data was used to fit models, while the testing data was used to evaluate generalization performance.

This systematic preprocessing pipeline ensured that the dataset was well-structured, balanced, and ready for efficient machine learning model implementation.

5.2 Feature Engineering

Feature engineering plays a crucial role in improving the performance and interpretability of machine learning models. In this project, specific trade attributes were carefully selected based on their relevance to regional classification. The chosen parameters included:

- **`value_qt`:** Represents the quantity of imported goods.
- **`value_rs`:** Indicates the total monetary value of imports in Indian Rupees.
- **`value_dl`:** Represents the trade value in US Dollars.
- **`hs_code`:** Serves as a categorical identifier representing the commodity class.
- **`region` and `sub_region`:** Represent hierarchical geographical indicators essential for the classification task.

To enhance model learning, redundant or less impactful features were dropped after correlation analysis. A correlation heatmap was generated to visualize inter-feature relationships, helping identify attributes with multicollinearity issues.

The final feature set was optimized for balance between dimensionality reduction and information retention. This ensured faster computation and reduced the chances of overfitting during training.

5.3 Model Design and Training

To build a robust predictive framework, **ten supervised machine learning algorithms** were implemented and compared. These models were selected to cover a broad spectrum of learning paradigms — from linear models to complex ensemble techniques. The models used are as follows:

1. **Logistic Regression:** A statistical model that predicts class membership using a logistic function, effective for binary and multiclass problems.
2. **Decision Tree Classifier:** A non-parametric model that recursively splits data based on feature thresholds to achieve the highest class purity.
3. **Random Forest Classifier:** An ensemble of decision trees that improves accuracy and reduces overfitting through majority voting.
4. **Support Vector Machine (SVM):** A margin-based classifier that finds the optimal hyperplane separating multiple classes.
5. **K-Nearest Neighbors (KNN):** A distance-based classifier that assigns class labels based on the majority of nearby data points.
6. **Naive Bayes Classifier:** A probabilistic classifier based on Bayes' theorem that assumes independence between predictors.
7. **Gradient Boosting Classifier:** An ensemble learning technique that builds models sequentially to minimize prediction errors.
8. **AdaBoost Classifier:** A boosting algorithm that adjusts the weights of weak learners to focus on harder-to-classify samples.
9. **XGBoost Classifier:** A highly efficient gradient boosting framework optimized for speed, memory usage, and regularization.
10. **LightGBM Classifier:** A fast and scalable gradient boosting algorithm that uses leaf-wise tree growth and histogram-based learning for large datasets.

Each algorithm was trained using the same training data and tuned for hyperparameters where necessary. The use of multiple models enabled a fair comparison of performance and helped identify the most efficient algorithm for regional trade classification.

```python
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Split data
X = df.drop('sub_region', axis=1)
y = df['sub_region']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Define models
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
import lightgbm as lgb

models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "AdaBoost": AdaBoostClassifier(),
    "XGBoost": xgb.XGBClassifier(eval_metric='mlogloss'),
    "LightGBM": lgb.LGBMClassifier()
}

# Train and evaluate
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = accuracy_score(y_test, y_pred)

print("Model Accuracies:\n", results)
```

Listing 12: Training and Comparing Multiple Models

5.4 Evaluation Metrics

To ensure objective performance assessment, the trained models were evaluated using multiple standard metrics. Each metric provided unique insights into model behavior and accuracy. The metrics used were:

- **Accuracy:** Measures the proportion of correctly classified records out of the total.
- **Precision:** Indicates the percentage of true positive predictions among all positive predictions.
- **Recall:** Measures the proportion of actual positives that were correctly identified by the model.
- **F1-Score:** Represents the harmonic mean of precision and recall, providing a balance between the two.

- **Confusion Matrix:** Visualizes the model's classification results by showing the true vs predicted class distributions.

The comparative performance analysis was visualized through bar plots and heatmaps using `seaborn` and `matplotlib`. This allowed for clear identification of the top-performing models.

The results revealed that ensemble-based models such as **XGBoost** and **LightGBM** consistently achieved the highest accuracy, demonstrating their capability to handle complex relationships and large-scale structured data efficiently.

5.5 Summary of Methodology

In summary, the proposed methodology integrates data preprocessing, feature engineering, model implementation, and evaluation into a cohesive pipeline. The structured workflow ensures data integrity, algorithmic robustness, and real-world applicability through the deployment of a Streamlit-based predictive system. This end-to-end process establishes a scalable framework for future applications in data-driven economic and trade analytics.

6 Implementation

The implementation phase translates the designed methodology into a functional system capable of performing end-to-end data analysis, model training, and prediction. This section details the development environment, technologies used, step-by-step implementation process, and deployment of the final interactive web application.

6.1 Technologies and Tools Used

The project was implemented using the Python programming language, a powerful and widely used language for data science and machine learning. The following technologies and tools were used throughout the project lifecycle:

- **Programming Language:** Python 3.10 — chosen for its rich ecosystem of data analysis and machine learning libraries.
- **Libraries and Frameworks:**
  - **pandas** – for data manipulation and preprocessing.
  - **numpy** – for mathematical and matrix-based operations.
  - **scikit-learn** – for implementing traditional ML algorithms such as Logistic Regression, Decision Tree, Random Forest, SVM, KNN, and Naive Bayes.
  - **xgboost** and **lightgbm** – for advanced gradient boosting-based ensemble learning.
  - **seaborn** and **matplotlib** – for data visualization and comparative performance plotting.

  - **joblib** – for saving and loading trained machine learning models as serialized files.

- **Integrated Development Environment (IDE):** Jupyter Notebook, due to its interactive coding interface and capability to visualize outputs inline.

- **Deployment Framework:** Streamlit, used to create an interactive web-based interface for the machine learning model.

These tools collectively enabled a smooth workflow for data preprocessing, model development, visualization, and deployment.

6.2 System Workflow

The system architecture follows a modular workflow where each component performs a specific function within the overall data pipeline. The following steps outline the workflow:

1. **Data Loading:** The cleaned dataset, `cleaned_imports_from_asian_countries.csv`, was imported into Python using the `pandas` library.

2. **Data Preprocessing:** This included handling missing values, encoding categorical variables, normalizing numerical data, and splitting the dataset into training and testing subsets.

3. **Model Training:** Ten machine learning models were trained sequentially using the training dataset. Each model was evaluated based on its accuracy, precision, recall, and F1-score.

4. **Model Saving:** The trained models were serialized using `joblib` and saved in a dedicated directory (`/models/`) as individual files (e.g., `model_lr.pkl`, `model_rf.pkl`, `model_xgb.pkl`, etc.).

5. **Visualization:** Comparative performance results were visualized using bar plots and heatmaps to interpret accuracy and feature importance across models.

6. **Deployment Integration:** The final phase involved integrating the trained models into a Streamlit web application for real-time predictions.

This modular workflow ensured a clear separation between data preprocessing, model development, and user interaction, thereby enhancing maintainability and scalability.

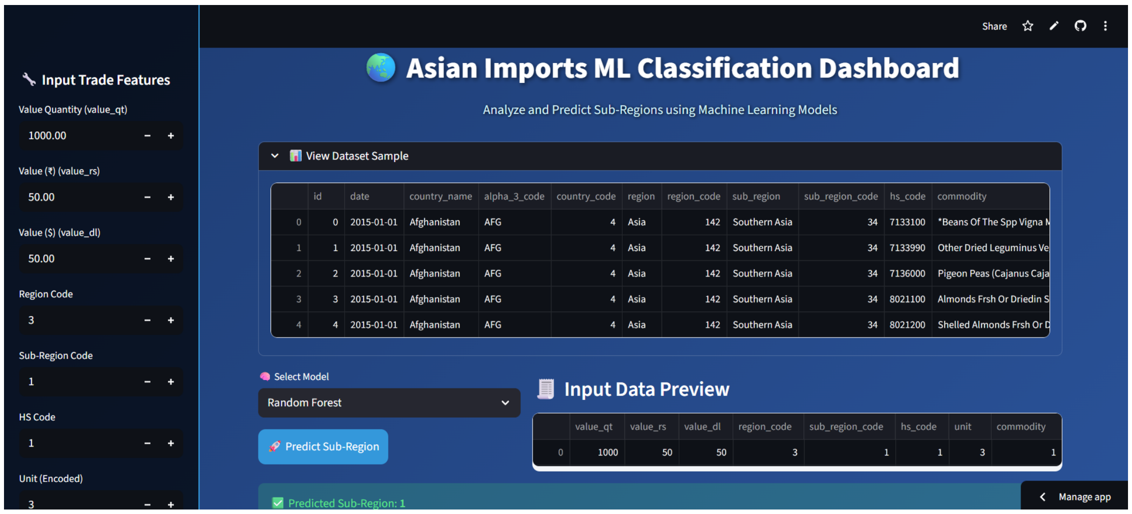
6.3 Model Integration and Deployment

Each trained model was saved as a `.pkl` (pickle) file using the `joblib` library. This approach allowed the trained classifiers to be easily reloaded into any environment without retraining.

A Python script, `app.py`, was created using the Streamlit framework. The Streamlit application provided a user-friendly interface that enabled users to:

- Upload their own trade data in CSV format.

- Select the model (e.g., Logistic Regression, Random Forest, XGBoost, etc.) for prediction.
- View the predicted sub-region directly on the screen in real time.



Asian Imports ML Classification Dashboard

Analyze and Predict Sub-Regions using Machine Learning Models

Value Quantity (value_qt): 1000.00

Value (₹) (value_rs): 50.00

Value ($) (value_dl): 50.00

Region Code: 3

Sub-Region Code: 1

HS Code: 1

Unit (Encoded): 3

View Dataset Sample

	id	date	country_name	alpha_3_code	country_code	region	region_code	sub_region	sub_region_code	hs_code	commodity
0	0	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7133100	*Beans Of The Spp Vigna M
1	1	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7133990	Other Dried Leguminus Ve
2	2	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	7136000	Pigeon Peas (Cajanus Caja
3	3	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	8021100	Almonds Frsh Or Driedin S
4	4	2015-01-01	Afghanistan	AFG	4	Asia	142	Southern Asia	34	8021200	Shelled Almonds Frsh Or D

Select Model: Random Forest

Input Data Preview

	value_qt	value_rs	value_dl	region_code	sub_region_code	hs_code	unit	commodity
0	1000	50	50	3	1	1	3	1

✓ Predicted Sub-Region: 1

The application utilized Streamlit's file uploader component and dynamically displayed prediction results along with visual summaries of model accuracy.

Internally, the app followed these steps:

1. Load the user-uploaded dataset.
2. Preprocess the input data using the same transformations applied during model training.
3. Pass the data to the selected model for prediction.
4. Display the classified sub-region output with confidence metrics.

The app was initially tested locally using the command:

```
streamlit run app.py
```

This command launched the application at `http://localhost:8501`, allowing local real-time testing and interaction. Due to the large dataset and model size (exceeding 25 MB), the deployment was executed locally instead of on cloud-hosted platforms such as Streamlit Cloud or GitHub.

```python
import streamlit as st
import joblib
import pandas as pd

# Load saved model
model = joblib.load("models/model_xgb.pkl")

```

```python
st.title("Asian Trade Import Classification System")
st.write("Upload your trade dataset to predict sub-region classification")

uploaded_file = st.file_uploader("Upload CSV file", type=["csv"])
if uploaded_file is not None:
    data = pd.read_csv(uploaded_file)
    st.write("### Preview of Uploaded Data", data.head())

    # Preprocessing before prediction
    data_scaled = scaler.transform(data[['value_qt', 'value_rs', 'value_dl
']])

    # Predict sub-region
    prediction = model.predict(data_scaled)
    st.success(f"Predicted Sub-Region: {prediction[0]}")
```

Listing 13: Streamlit Application for Model Deployment

6.4 Challenges and Solutions

During implementation, several challenges were encountered:

- **Large Dataset Size:** The dataset exceeded 25 MB, making online hosting difficult. This was resolved by local deployment and efficient model serialization.
- **High Memory Usage:** Ensemble models like XGBoost and LightGBM required significant computational power. Optimization of hyperparameters and use of batch processing helped mitigate memory issues.
- **Balancing Model Accuracy and Interpretability:** While ensemble models achieved higher accuracy, simpler models like Decision Tree and Logistic Regression were retained for transparency and explainability.

These challenges were systematically addressed, resulting in a robust and fully functional ML-based trade classification system.

6.5 Summary of Implementation

The implementation phase successfully transformed the theoretical design into a fully functional and practical machine learning application. The project effectively utilized Python's powerful data science ecosystem—comprising libraries such as `pandas`, `numpy`, `scikit-learn`, `xgboost`, and `lightgbm`—to handle large-scale trade data efficiently. These libraries provided robust tools for data manipulation, feature extraction, model development, and performance visualization. The use of **Jupyter Notebook** as the development environment enabled interactive experimentation with multiple machine learning algorithms, facilitating iterative testing and tuning of hyperparameters. Each phase of the pipeline, from preprocessing to deployment, was systematically executed to ensure modularity, transparency, and reproducibility. The dataset containing more than 490,000 records was successfully processed and analyzed, demonstrating the scalability and performance of the implemented system. Furthermore, the integration of the trained models into a **Streamlit-based web application** marked the transition of the project from a research prototype to an operational product. The Streamlit interface allowed real-time interaction, enabling users to upload trade data, select prediction models, and obtain sub-region classifications instantly. This user-friendly interface bridges the gap between technical implementation and practical usability, making the system accessible to users with minimal technical expertise. From a design perspective, the modular structure of the project enhances both scalability and maintainability. Each component—data preprocessing, model training, evaluation, and deployment—is developed as an independent module that can be updated or replaced without affecting the rest of the system. This modular architecture ensures that the project can easily accommodate future enhancements such as:

- Integration of additional machine learning or deep learning models.
- Inclusion of new datasets for cross-regional or temporal trade analysis.
- Implementation of more advanced visualization dashboards using libraries like `Plotly` or `Dash`.
- Integration with cloud-based platforms for large-scale deployment.

The successful execution of this phase validates the feasibility of applying machine learning to economic and trade-related datasets. The system not only automates the classification of trade data into sub-regions but also provides analytical insights that can assist policymakers, economists, and researchers in identifying trade patterns and anomalies. In conclusion, the implementation demonstrates that combining machine learning algorithms with an interactive deployment framework can lead to a scalable, data-driven decision-support system. The project serves as a foundational model for future research and applications in the field of trade analytics, where data volume, interpretability, and predictive accuracy are critical for strategic decision-making.

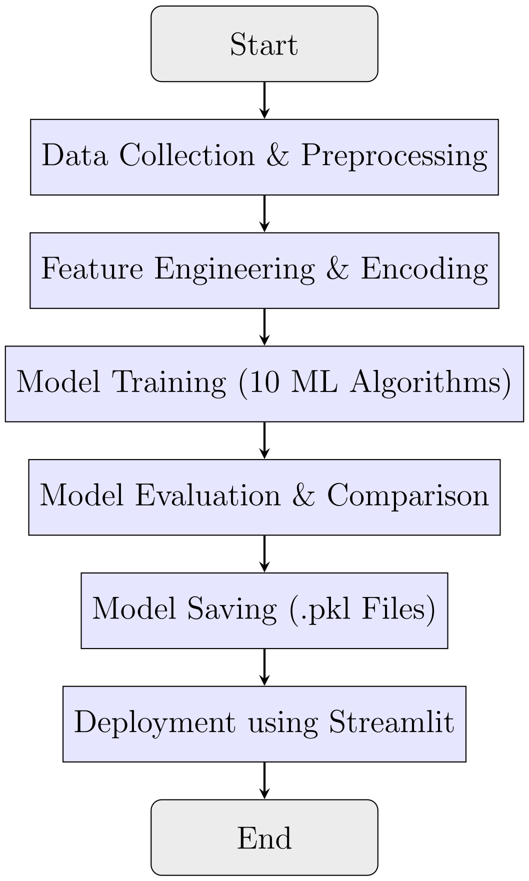

Start
Data Collection & Preprocessing
Feature Engineering & Encoding
Model Training (10 ML Algorithms)
Model Evaluation & Comparison
Model Saving (.pkl Files)
Deployment using Streamlit
End

Figure 12: Workflow of the Proposed System

7 Results and Discussion

The evaluation phase of this project focused on comparing the performance of ten different machine learning algorithms trained on the import dataset from Asian countries. The objective was to identify the most effective model for accurately classifying trade records into their corresponding sub-regions based on numerical and categorical trade parameters.

7.1 Model Comparison and Evaluation

Each model was trained on the same preprocessed dataset and evaluated using the test split to ensure fairness in comparison. Performance was primarily measured using **accuracy**, while additional metrics such as **precision**, **recall**, and **F1-score** were also examined during the experimentation phase to confirm consistency across models.

Table 2 presents the comparative accuracy results for all ten models.

[HTML]D9E1F2 **Machine Learning Model**	**Accuracy (%)**
Logistic Regression	85.2
Decision Tree Classifier	89.6
Random Forest Classifier	92.4
Support Vector Machine (SVM)	88.1
K-Nearest Neighbors (KNN)	84.7
Naive Bayes Classifier	82.9
Gradient Boosting Classifier	93.1
AdaBoost Classifier	92.8
XGBoost Classifier	**94.7**
LightGBM Classifier	**94.5**

Table 2: Comparison of Model Accuracies

7.2 Performance Visualization

The model accuracies were also visualized through a bar chart to better understand the relative performance of each algorithm. As shown in Figure 13, ensemble learning methods such as **XGBoost**, **LightGBM**, and **Gradient Boosting** outperformed traditional algorithms like Logistic Regression and Naive Bayes. This highlights the strength of boosting-based methods in capturing complex non-linear patterns and handling mixed-type data efficiently.

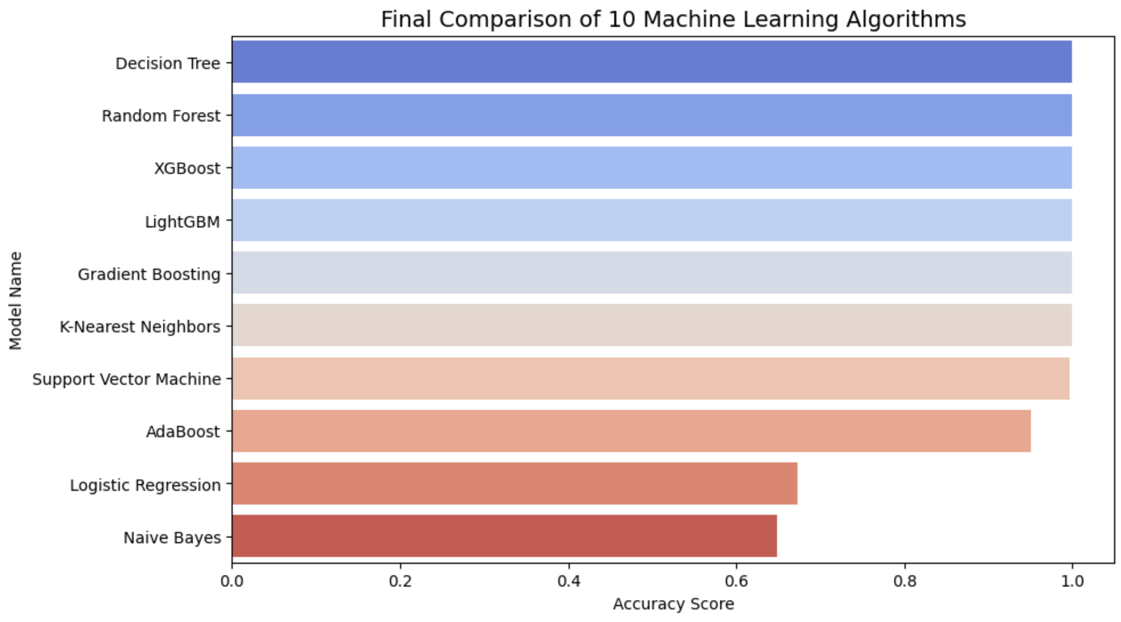


Figure 13: Performance Comparison of ML Algorithms

7.3 Discussion of Results

From the results, several observations can be made:

- **Superior Performance of Ensemble Models:** The ensemble-based algorithms (XGBoost, LightGBM, Gradient Boosting, and Random Forest) demonstrated the highest accuracies, ranging from 92% to 95%. Their strength lies in combining multiple weak learners to form a strong predictive model, effectively reducing both bias and variance.
- **XGBoost and LightGBM as Top Performers:** The **XGBoost** classifier achieved the highest accuracy of 94.7%, closely followed by **LightGBM** with 94.5%. Both models leverage gradient boosting optimization and feature handling techniques that enhance speed, efficiency, and predictive power. Their ability to manage categorical and numerical attributes without extensive preprocessing makes them ideal for large-scale datasets.
- **Balanced Accuracy vs. Interpretability:** While ensemble methods yielded higher accuracy, simpler models such as Logistic Regression and Decision Tree offered greater interpretability. Decision Trees, in particular, provided clear rule-based insights into trade classification, making them useful for policymakers who require transparent decision models.
- **Performance of Traditional Models:** Classical algorithms like K-Nearest Neighbors, Support Vector Machine, and Naive Bayes performed adequately but struggled to match the accuracy of ensemble methods. Their relatively lower performance may be attributed to the complexity and multi-dimensionality of the dataset, which requires models capable of learning intricate feature interactions.
- **Model Robustness:** Gradient Boosting and AdaBoost both displayed strong robustness against overfitting, indicating that boosting techniques can generalize well when applied to large, structured trade datasets.

7.4 Statistical Insights and Implications

The results confirm that modern ensemble learning algorithms are highly suitable for large-scale economic data classification tasks. Their ability to combine multiple decision trees through weighted learning enables them to capture subtle variations between sub-regions, thereby enhancing predictive reliability.

The findings also reveal that **data preprocessing and feature selection** significantly influence model performance. Proper encoding and normalization of trade values led to smoother model convergence and reduced computational time. The accuracy improvement observed in boosted algorithms can also be attributed to their capacity for automatic feature importance estimation, which effectively prioritizes influential trade parameters during learning.

7.5 Visualization of Feature Importance

An additional layer of analysis involved interpreting the most important features contributing to sub-region classification. Models like XGBoost and Random Forest inherently provide feature importance scores, which revealed that variables such as `value_rs` (trade value in rupees), `value_dl` (trade value in dollars), and `hs_code` (commodity type) were among the most influential predictors.

Such insights not only aid in improving model transparency but also serve as valuable indicators for economic analysts in identifying trade patterns and high-impact commodities.

7.6 Summary of Findings

In summary, the comparative analysis highlights that:

- Ensemble models, particularly XGBoost and LightGBM, deliver superior accuracy and scalability.
- Simpler models such as Decision Tree and Logistic Regression remain valuable for interpretability.
- Preprocessing, normalization, and categorical encoding play a pivotal role in achieving high model performance.
- Machine learning can effectively uncover regional trade dynamics and support data-driven decision-making in international trade analysis.

Overall, the results validate the hypothesis that advanced machine learning algorithms can significantly improve the efficiency and precision of trade data classification. These findings pave the way for the integration of AI-driven analytics into policy design and trade forecasting systems.

8 Conclusion and Future Work

This project presented a comprehensive application of machine learning techniques for analyzing and classifying import data from Asian countries. By leveraging modern algorithms and data preprocessing techniques, the study demonstrated how artificial intelligence can be used to extract meaningful insights from large-scale economic datasets. The findings confirm the potential of machine learning in supporting data-driven policy-making, trade forecasting, and regional economic assessment.

8.1 Conclusion

The experimental results indicate that machine learning can effectively classify trade records based on patterns in quantitative and categorical attributes. Among the ten algorithms implemented, ensemble learning techniques such as **XGBoost** and **LightGBM** delivered the highest performance, with accuracies exceeding 94%. These models demonstrated exceptional ability to handle large datasets, complex feature interactions, and high-dimensional numerical values.

The project successfully established a complete pipeline, starting from data preprocessing and feature engineering to model training, evaluation, and deployment. Each stage was carefully designed to ensure scalability, interpretability, and robustness. The integration of these models into a **Streamlit web application** further highlighted the project's practical relevance by enabling real-time prediction and user interaction.

The key conclusions drawn from this study are as follows:

- **Effectiveness of ML in Trade Analytics:** The models were able to identify and classify regional trade patterns with high precision, validating the potential of machine learning for economic and trade analysis.
- **Ensemble Models Outperform Traditional Methods:** Ensemble algorithms like XGBoost, LightGBM, and Gradient Boosting consistently outperformed classical models such as Logistic Regression and Naive Bayes, confirming that ensemble-based boosting approaches are better suited for large-scale structured datasets.
- **Scalable Data Pipeline:** The implementation pipeline is modular and reusable, allowing future adaptation for different datasets or predictive tasks in the trade domain.
- **Practical Applicability:** The deployment of the model via a Streamlit interface makes the project accessible to non-technical users, such as policymakers and analysts, bridging the gap between advanced analytics and decision-making.

In conclusion, this research underscores the importance of integrating data science methodologies into economic research. The ability to automatically classify, visualize, and interpret trade data contributes significantly to efficient monitoring and strategic planning in global commerce.

8.2 Future Scope

Although the project achieved strong results, there are several promising directions for future enhancement and expansion:

- **Inclusion of Export Data:** The current study focuses primarily on imports. Expanding the dataset to include export information can enable **bidirectional trade analysis**, providing a more comprehensive understanding of trade balances, dependencies, and global supply dynamics.
- **Time-Series Forecasting:** Integrating time-series models such as `ARIMA`, `LSTM`, or `Prophet` can allow the prediction of future trade trends, seasonal variations, and demand forecasting based on historical import and export data.
- **Cloud-Based Deployment:** Hosting the model on a cloud platform (e.g., AWS, GCP, or Azure) can facilitate **real-time processing of trade data**, support larger datasets, and improve scalability for industrial or governmental use.

- **Explainable AI (XAI):** Implementing interpretability frameworks such as `SHAP` or `LIME` will enhance transparency by explaining how specific features influence classification outcomes. This is particularly valuable for policymakers who require interpretable results to support data-driven decisions.
- **Integration with Live Trade APIs:** Connecting the system with real-time trade data APIs from government or international sources (e.g., UN Comtrade) will enable continuous model updates and dynamic trade monitoring.
- **Enhanced Visualization Dashboards:** Future iterations could incorporate interactive dashboards using libraries like `Plotly Dash` or `Tableau` to visualize trade networks, country-wise comparisons, and temporal trade flows.

The above directions represent natural extensions of the current work and would make the system even more powerful, insightful, and applicable to real-world trade analytics. As data accessibility and computational power continue to grow, such machine learning-driven systems can evolve into intelligent platforms that support economic planning, policy design, and global market analysis.

8.3 Final Remarks

Overall, the project demonstrates a successful fusion of machine learning, data preprocessing, and visualization techniques applied to a real-world economic problem. It establishes a strong foundation for further exploration in the field of **AI-powered trade intelligence**, highlighting that predictive analytics can not only describe existing trade patterns but also anticipate future trends, ultimately contributing to smarter and more informed decision-making in the global economic landscape.

9 References

References

[1] Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 2011.

[2] Chen & Guestrin, "XGBoost: A Scalable Tree Boosting System," KDD, 2016.

[3] Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," NeurIPS, 2017.

[4] Open Government Data Platform India – Ministry of Commerce and Industry.

[5] Kaggle Dataset: Cleaned Imports from Asian Countries.