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 Project Definition and Data Understanding

1.1 Problem Statement

India imports various commodities from Asian countries. Understanding the trend, value, and volume of these imports is crucial for policymaking, trade negotiations, and economic planning. This project performs a detailed Exploratory Data Analysis (EDA) on the imports dataset to identify:

- Major source countries
- Trends over time
- Dominant commodities
- High-value trade flows

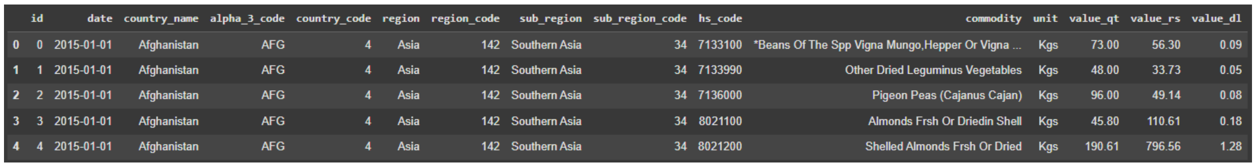
1.2 Data Source

- **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

1.3 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



	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

1.4 Integration Methodology

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

1.5 Initial Validation

Initial checks included:

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

1.6 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.

1.7 Initial Validation

Initial checks included:

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

1.8 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
```

1.9 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.

1.9.1 Address Outliers

```python
# 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)
```

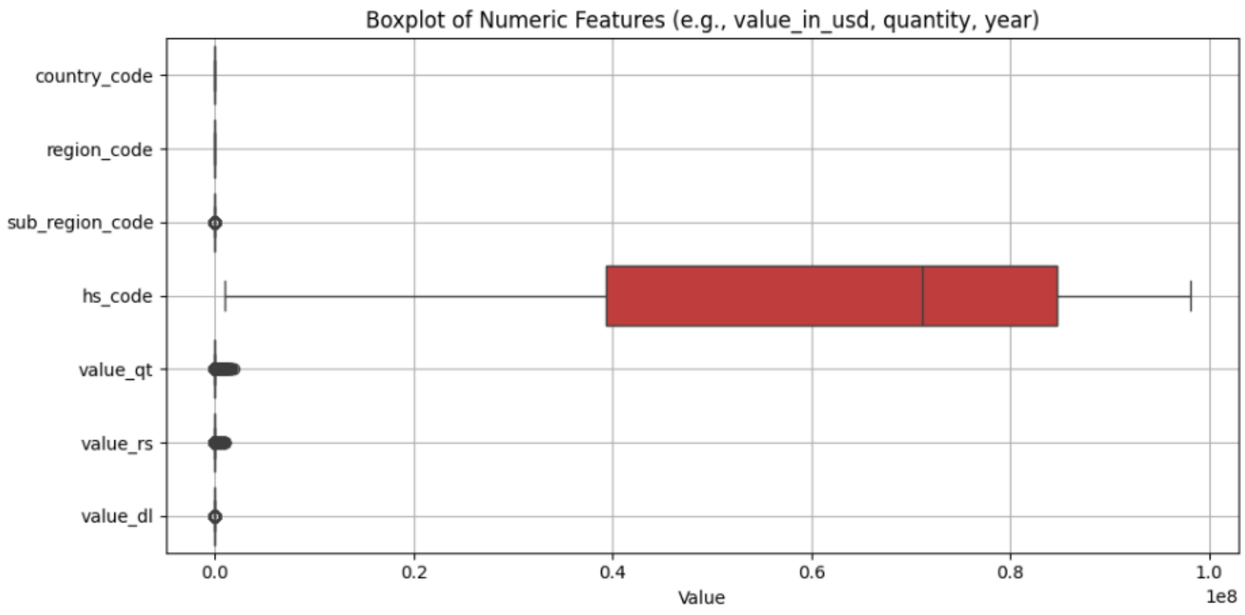


Figure 2: Box plot for outliers detection

```
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

```

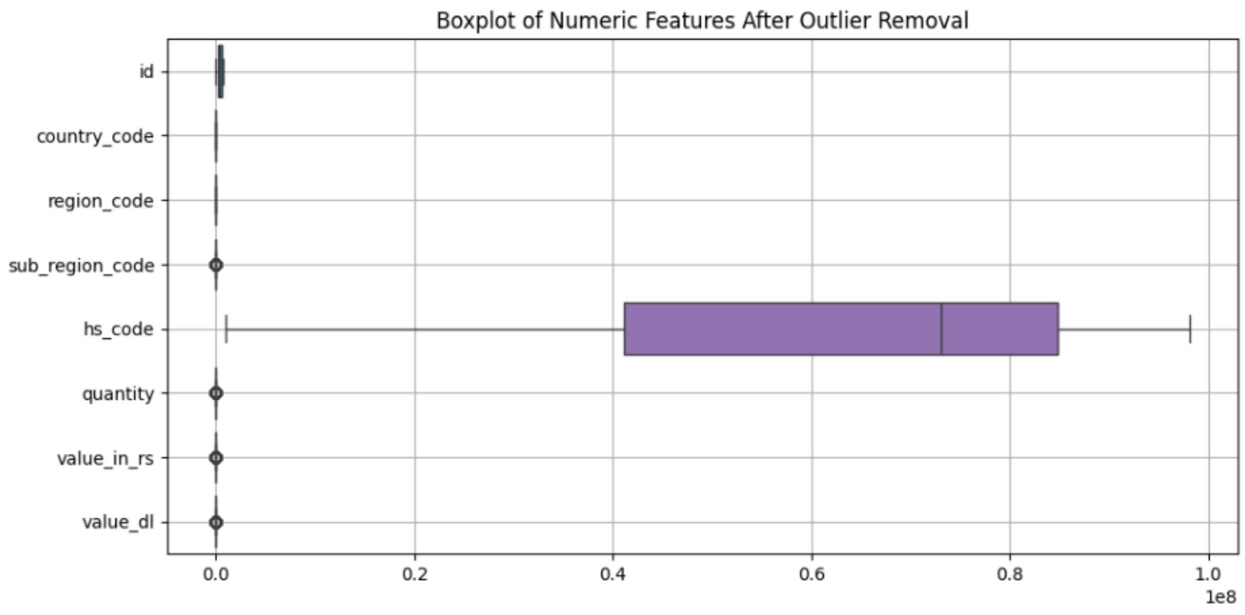


Figure 3: Box plot after outliers detection and removal

```python
    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()
```

1.10 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.

```python
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.

1.11 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.

1.12 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.

1.13 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`):

```python
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.

```
# 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
```

1.14 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:

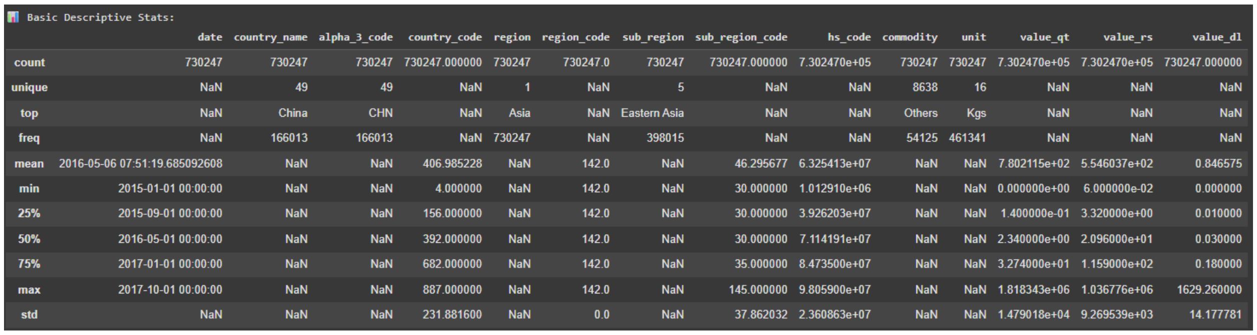
- `value_in_rs` shows the total monetary worth of imported commodities.
- `quantity` reflects the amount of goods received, expressed in various units.
- `value_per_unit` captures the average price per unit of goods, enabling comparisons across products and countries.

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

```
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.

1.15 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.

1.15.1 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()
```

1.15.2 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.

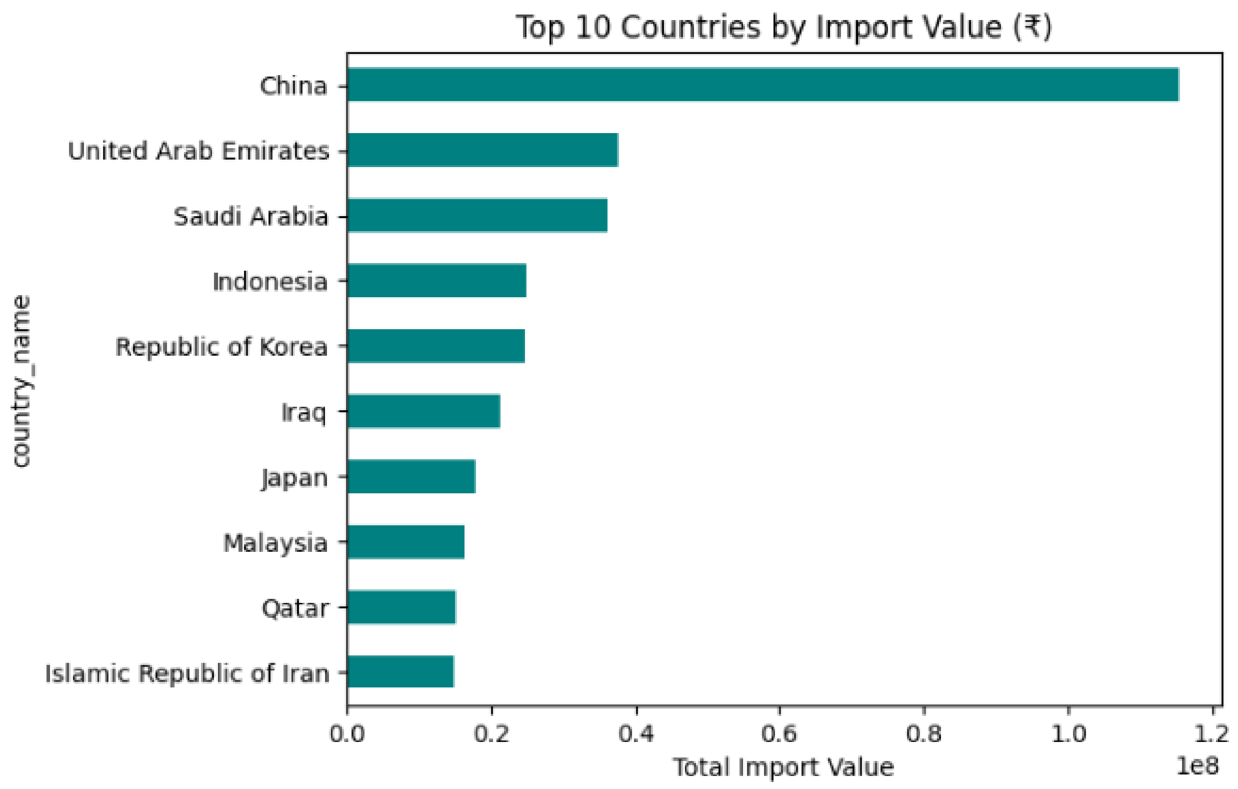


Figure 5: Distribution of Energy Metrics

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.

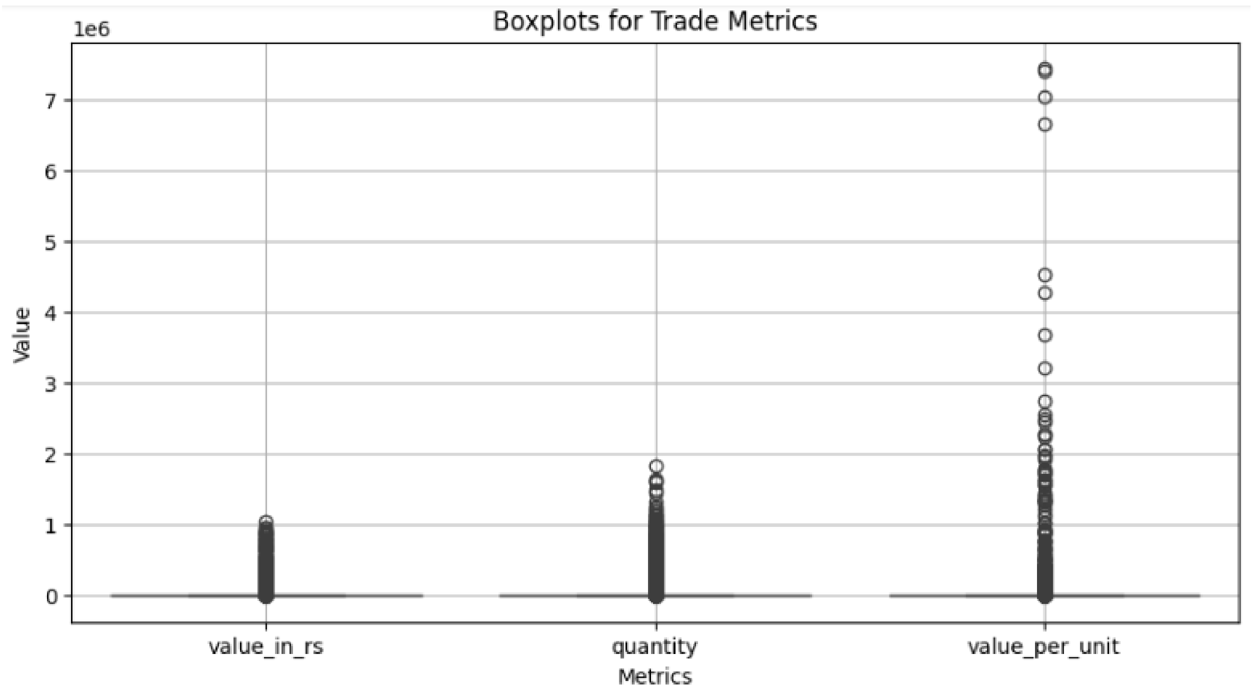


Figure 6: Enter Caption

Figure 7: Boxplots for Energy Metrics

1.15.3 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.

```
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')
```

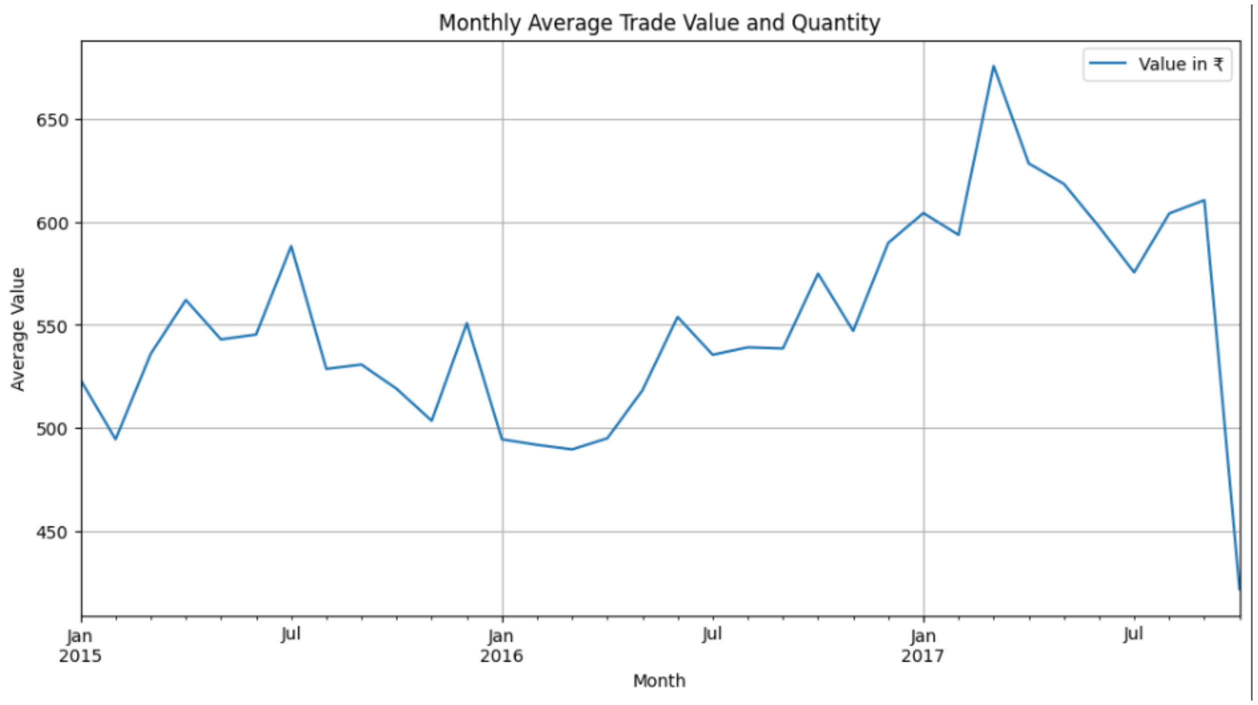


Figure 8: Boxplots for Energy Metrics

```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

1.15.4 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.

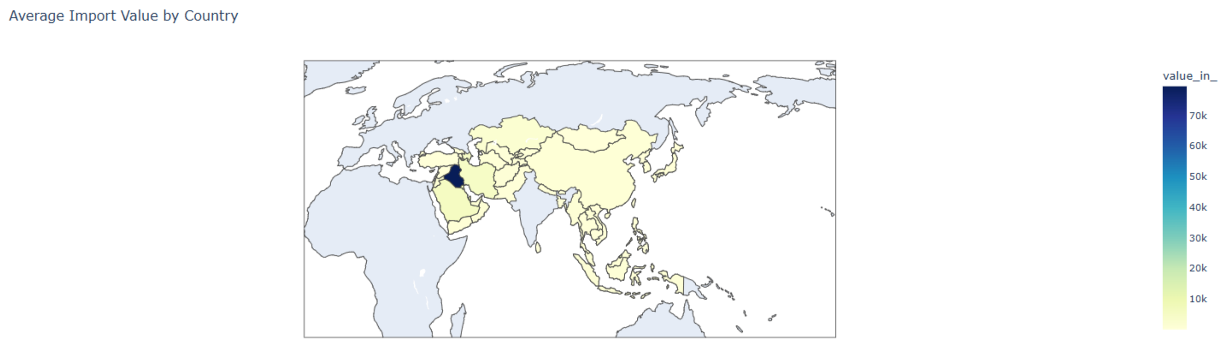


Figure 9: Choropleth Map of Average Energy Gap Across Asia

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

1.15.5 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

1.16 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"]

```

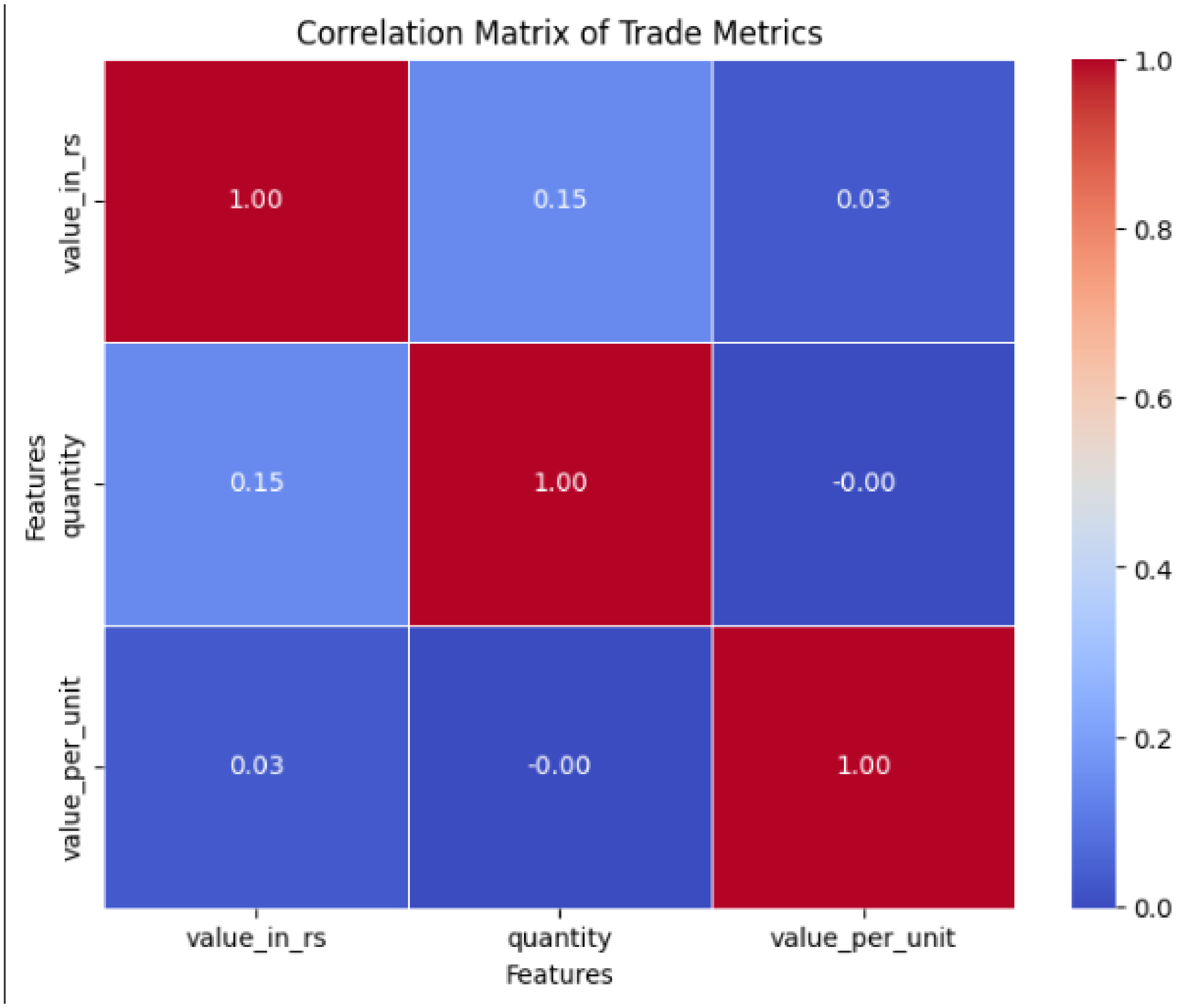


Figure 10: Correlation Heatmap

```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.

1.17 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.

1.17.1 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

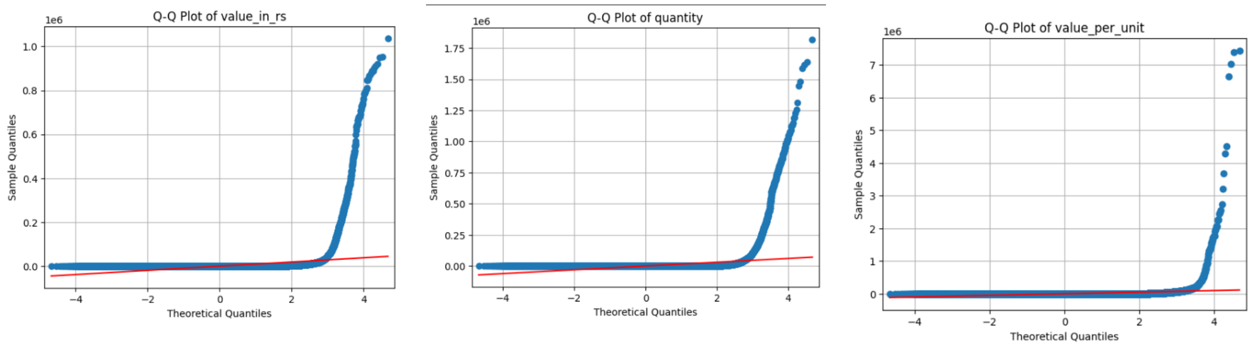


Figure 11: Q-Q plots assessing normality of import value, quantity, and unit price distributions.

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.

1.17.2 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)
```

1.17.3 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.

2 Machine Learning Algorithms Overview

This section outlines the machine learning algorithms applied to the trade dataset. The objective is to model and predict key trade metrics, such as `value_in_rs` or `value_per_unit`, based on features including commodity type, country, quantity, and time (month, year).

The following regression algorithms were selected for experimentation due to their suitability for numerical prediction and interpretability:

- **Linear Regression:** Acts as the baseline model. It assumes a linear relationship between input variables and the target variable and provides easily interpretable coefficients.
- **Ridge Regression:** An extension of linear regression that incorporates L2 regularization to mitigate multicollinearity and enhance generalization on unseen data.
- **Lasso Regression:** Uses L1 regularization, which can shrink some coefficients to zero, effectively performing variable selection and improving model simplicity.
- **Random Forest Regressor:** A powerful ensemble method that builds multiple decision trees and aggregates their results to improve prediction accuracy and reduce variance.

Model Performance Comparison

To assess the effectiveness of each model, performance metrics such as the R^2 score and Mean Squared Error (MSE) were calculated on the test set. The R^2 score indicates the proportion of variance explained by the model, while MSE penalizes large errors and provides a direct measure of prediction accuracy.

This comparative evaluation facilitates the selection of the most appropriate model for trade value prediction. The chosen model balances accuracy, interpretability, and robustness to overfitting, ensuring reliable performance for economic forecasting or policy planning.