India Import and Export Analysis (1997-2022)

1. Project Overview

This project provides a comprehensive analysis of India's import and export data from 1997 to 2022. The data was sourced, cleaned, and transformed using Python (Pandas and NumPy), stored in an SQLite database, and finally visualized using Power BI for interactive insights. The entire project, including the data and scripts, is managed using Git and hosted on GitHub.

Technologies Used:

Python: Pandas, NumPy, SQLite3

Database: SQLite

Business Intelligence: Microsoft Power BI

Version Control: Git, GitHub

2. Data Source and Initial State

The raw data was provided in a CSV format, named EAI.csv. This dataset contains information on country-wise exports, imports, total trade, and trade balance across various financial years.

Initial Data Snippet (EAI.csv):

Country	Export	Import	Total Trade	Trade Balance	Financial Year (start)	Financial Year (end)
AFGH →	21.25	10.7	31.95	10.55	1997	1998
AFGH →	12.81	28.14	40.95	-15.33	1998	1999
AFGH →	33.2	21.06	54.26	12.15	1999	2000
AFGH →	25.86	26.59	52.45	-0.73	2000	2001
AFGH →	24.37	17.52	41.89	6.85	2001	2002

3.Data Cleaning and Preprocessing (Python - Pandas/NumPy)

The raw data required significant cleaning to handle missing values, inconsistent data types, and prepare it for analysis. The EAI_Cleaned.py script was used for this purpose.

Key Cleaning Steps:

- Load Data: The EAI.csv file was loaded into a Pandas DataFrame.
- Handle Missing Country Names: Rows where the 'Country' column was entirely missing were dropped.

- Convert Numeric Columns: 'Export', 'Import', 'Total Trade', and 'Trade Balance' columns contained commas and were of object type. They were converted to string, commas were removed, and then coerced to numeric type. Any errors during conversion were replaced with NaN and then filled with 0.
- Fill Remaining NaNs: All other remaining NaN values across the DataFrame were filled with 0.
- Create 'Year' Column: A new 'Year' column was extracted from 'Financial Year(start)', ensuring it was an integer type.
- Add India's Perspective Columns: To provide a clear perspective from India's viewpoint, the following columns were added:India_Imports: Represents other countries' exports to India (which are India's imports). This was derived from the 'Export' column in the original dataset.

India_Exports: Represents other countries' imports from India (which are India's exports). This was derived from the 'Import' column in the original dataset.

IndiaTrade_Balance: Calculated as India_Exports - India_Imports.

Save Cleaned Data: The processed data was saved to a new CSV file named Complete_India_Trade_Data_All_Countries_Filled.csv.

Python Code for Data Cleaning (EAI_Cleaned.py):

df.to csv(output filename, index=False)

```
Python
import pandas as pd
import numpy as np
# Load the complete dataset
df = pd.read csv('EAI.csv')
# Data Cleaning - keep all rows, only remove if Country is completely missing
df = df.dropna(subset=['Country'], how='all')
# Convert numeric columns - fill blanks with 0 and handle commas
numeric_cols = ['Export', 'Import', 'Total Trade', 'Trade Balance']
for col in numeric cols:
# Convert to string, replace commas, then to numeric
df[col] = df[col].astype(str).str.replace(',', ")
# Convert to numeric, forcing errors to NaN, then fill with 0
df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)
# Fill any remaining NaN values with 0 (for non-numeric columns if any)
df = df.fillna(0)
# Create Year column from Financial Year(start) and fill blanks with 0
df['Year'] = df['Financial Year(start)'].astype(str).str.extract('(\\d+)')[0].fillna(0).astype(int)
# Add India's perspective columns
df['India Imports'] = df['Export'] # Other countries' exports to India (India's imports)
df['India Exports'] = df['Import'] # Other countries' imports from India (India's exports)
df['India Trade Balance'] = df['India Exports'] - df['India Imports']
# Save the complete dataset with all countries and all years
output filename = 'Complete India Trade Data All Countries Filled.csv'
```

Verification

print(f"Total records processed: {len(df)}")

print(f"Total unique countries: {df['Country'].nunique()}")
print(f"Years covered: {df['Year'].min()} to {df['Year'].max()}")

Cleaned Data Snippet (Complete_India_Trade_Data_All_Countries_Filled.csv):

Coun try	Expor t	Impor t	Total Trade	Trade Balan ce	Finan cial Year (start)	Finan cial Year (end)	Year	India _Imp orts	India _Exp orts	India _Trad e_Bal ance
A •	21.25	10.7	31.95	10.55	1997	1998	1997	21.25	10.7	-10.55
A	12.81	28.14	40.95	-15.33	1998	1999	1998	12.81	28.14	15.33
A •	33.2	21.06	54.26	12.15	1999	2000	1999	33.2	21.06	-12.14
A •	25.86	26.59	52.45	-0.73	2000	2001	2000	25.86	26.59	0.73
A •	24.37	17.52	41.89	6.85	2001	2002	2001	24.37	17.52	-6.85

4. Database Creation (Python - SQLite)

The cleaned data was then loaded into an SQLite database (india_trade.db) for efficient querying and management. The EAI_DATABASE.py script handles the database creation and data insertion.

Key Database Steps:

Connect to Database: An SQLite connection was established, creating the india_trade.db file if it didn't exist.

Create Table: A table named trade_data was created with appropriate columns to store the cleaned trade information.

Insert Data: The DataFrame containing the cleaned data was directly inserted into the trade_data table.

Create Views (for easier querying):

india_yearly_summary: Aggregates India's total exports, imports, and trade balance per year.

top_trading_partners: Provides a summary of total trade volume, exports, imports, and trade balance for each country across all years, ordered by total trade volume.

Python Code for Database Creation (EAI_DATABASE.py):

Python import pandas as pd import numpy as np

```
import sqlite3
from pathlib import Path
# Load and preprocess the data
print("Loading and preprocessing data...")
df = pd.read csv('EAI.csv')
# Data Cleaning (as described in section 3)
df = df.dropna(subset=['Country'], how='all')
numeric cols = ['Export', 'Import', 'Total Trade', 'Trade Balance']
for col in numeric cols:
  df[col] = df[col].astype(str).str.replace(',', ")
  df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)
df = df.fillna(0)
df['Year'] = df['Financial Year(start)'].astype(str).str.extract('(\\d+)')[0].fillna(0).astype(int)
df['India_Imports'] = df['Export']
df['India Exports'] = df['Import']
df['India_Trade_Balance'] = df['India_Exports'] - df['India_Imports']
# SQLite Database Setup
db_path = 'india_trade.db'
print(f"\nCreating SQLite database at {db path}...")
# Connect to SQLite (this will create the database if it doesn't exist)
conn = sqlite3.connect(db_path)
cursor = conn.cursor()
# Drop table if it already exists (for clean runs)
cursor.execute("DROP TABLE IF EXISTS trade data")
# Create table with all necessary columns
cursor.execute(""
CREATE TABLE trade_data (
  Country TEXT,
  Export REAL,
  Import REAL,
  "Total Trade" REAL,
  "Trade Balance" REAL,
  "Financial Year(start)" TEXT,
  "Financial Year(end)" TEXT,
  Year INTEGER,
  India Imports REAL,
  India Exports REAL,
  India_Trade_Balance REAL
```

```
# Insert data from DataFrame into the table
df.to sql('trade data', conn, if exists='append', index=False)
# Create views for easier analysis
# 1. India Yearly Summary
cursor.execute(""
CREATE VIEW IF NOT EXISTS india yearly summary AS
SELECT
  year,
  SUM(india exports) AS total india exports,
  SUM(india_imports) AS total_india_imports,
  SUM(india trade balance) AS total india trade balance
FROM trade data
GROUP BY year
ORDER BY year;
# 2. Top Trading Partners
cursor.execute(""
CREATE VIEW IF NOT EXISTS top_trading_partners AS
SELECT
  country,
  SUM(india exports) AS total exports to country,
  SUM(india_imports) AS total_imports_from_country,
  SUM(india trade balance) as total trade balance,
  SUM("Total Trade") as total trade volume,
  COUNT(*) as years_of_data
FROM trade data
GROUP BY country
ORDER BY total_trade_volume DESC;
"")
# Commit changes and close connection
conn.commit()
conn.close()
print(f"\nDatabase successfully created with:")
print(f"- {len(df):,} total trade records")
print(f"- {df['Country'].nunique()} unique countries")
print(f"- Data from {df['Year'].min()} to {df['Year'].max()}")
print("\nSample queries you can run:")
print("1. Get India's trade with China:")
print(' sqlite> SELECT * FROM trade_data WHERE country = "CHINA P RP" ORDER BY year;')
print("\n2. View India's yearly trade summary:")
print(' sqlite> SELECT * FROM india_yearly_summary;')
print("\n3. View top 10 trading partners:")
print(' sqlite > SELECT * FROM top trading partners LIMIT 10;')
```

print("\n4. Find countries with trade surplus with India:")
print(' sqlite> SELECT country, total_trade_balance FROM top_trading_partners WHERE total_trade_balance > 0
ORDER BY total_trade_balance DESC;')

Sample SQLite Queries:Retrieve all data for a specific country (e.g., CHINA P RP):

SQL

SELECT * FROM trade data WHERE Country = 'CHINA P RP' ORDER BY Year;

View India's yearly trade summary:

SQL

SELECT * FROM india yearly summary;

Get top 10 trading partners by total trade volume:

SQL

SELECT * FROM top_trading_partners LIMIT 10;

Find countries with a trade surplus with India:

SQL

SELECT Country, total_trade_balance FROM top_trading_partners WHERE total_trade_balance > 0 ORDER BY total_trade_balance DESC;

Extract Data from SQLite to CSV (using Python is common):

Install necessary libraries: If you don't have them, you'll need pandas and sqlite3.

```
pip install pandas
```

Python Script: Write a Python script to connect to your SQLite database, query the data from the desired tables or views, and then save that data to a CSV file.

Python

```
import sqlite3
import pandas as pd
db path = 'india trade.db' # Replace with your .db file name
output csv path = 'trade data from db.csv' # Desired output CSV file name
conn = sqlite3.connect(db path)
# Example: Querying a specific table or view
# You can change 'trade_data' to 'india_yearly_summary' or 'top_trading_partners'
query = "SELECT * FROM trade data"
try:
    df = pd.read sql query(query, conn)
    df.to csv(output csv path, index=False)
   print(f"Data successfully exported to {output csv path}")
except Exception as e:
   print(f"An error occurred: {e}")
finally:
conn.close()
```

this script: Execute the Python file. This will create a CSV file containing your database's data.

5. Data Visualization and Analysis (Power BI)

Power BI was used to create interactive dashboards and visualizations to explore India's import and export trends. The trade_data.xls file contains the Power BI report.

Steps to Connect Power BI to SQLite Database:

Open Power BI Desktop: Launch Power BI Desktop application.

Get Data: Click on "Get Data" from the Home tab.

Browse for excel File: In the SQLite database dialog box, browse and select the trade_data.xls file created earlier. Click "Open".

Navigator: In the Navigator window, you will see all the tables and views from your SQLite database. Select trade_data, india_yearly_summary, and top_trading_partners.

Load Data: Click "Load" to import the selected tables/views into Power Bl.

Data Modeling (if needed): Ensure relationships between tables are correctly identified (though for this setup, direct loading of views might be sufficient for most analyses).

Power BI Visualizations and Screenshots:

Here are some screenshots from the Power BI dashboard, showcasing key insights into India's trade:

Overview of India's Trade (Imports, Exports, Trade Balance over Time)

This slide provides a high-level overview of India's import and export performance and the resulting trade balance across the years.

LINE CHARTS:

Power BI Line Chart – India Import vs Export (1997–2022)

1. Purpose

- To analyze trade trends over 25 years.
- Compare the growth and gaps between imports and exports.
- Identify trade surplus or deficit patterns.

2. Data Setup

- X-axis: Year (1997 to 2022).
- Y-axis: Value in USD Billion (or INR Crore).
- Legend: Import, Export (two lines).

3. Key Insights You Can Show

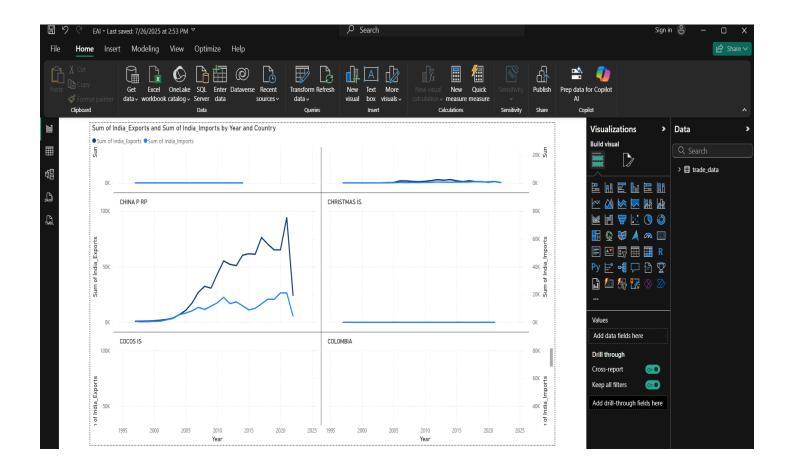
- Trend Analysis: How trade volume evolved.
- Trade Deficit Periods: When imports > exports.
- Trade Surplus/Improvement: Years when the gap reduced.
- Important Events Impacting Trade:
 - o 2008 Global Recession
 - o 2016 Demonetization
 - o 2020 COVID-19
 - 2022 Russia-Ukraine war impact on oil import bills

4. Visual Features to Use

- Dual-line chart: One for Exports, one for Imports.
- Markers: Highlight major events (tooltip or annotation).
- Data labels (optional): For peak values.
- Custom tooltip: Show year-wise details on hover.

• 5. Customizations

- Format Y-axis with currency (₹ or \$).
- Color Exports (e.g., green) and Imports (e.g., red).
- Add slicer for filtering by decade or government era.



TREEMAP CHARTS:

***** Treemap Chart – Power BI Overview (India Trade Data)

• 1. Purpose

- Visualize hierarchical or categorical data.
- Show proportions within a whole (e.g., top exporting sectors).
- Quickly compare the size of each category.

2. Data Setup (India Trade Example)

Use Case	Treemap Setup Example
Top Export Commodities	Category: CommodityValues: Export Value
Top Importing Countries	Category: CountryValues: Import Value
Sector-wise Trade Share	Group: SectorCategory: SubsectorValues: Trade value

3. Key Insights You Can Sho

- Which sectors dominate India's exports (e.g., Petroleum, Textiles, IT).
- Which countries are India's biggest trade partners.
- S Compare import vs export share by category.

4. Features

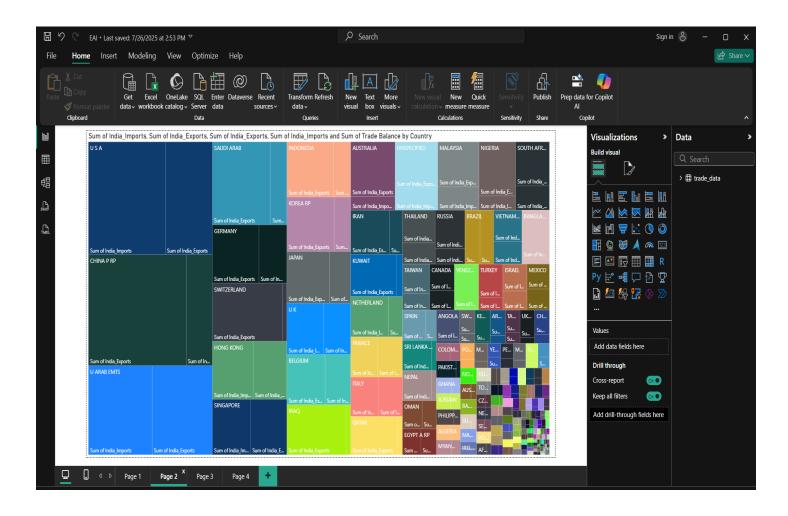
- Size of rectangle = value (e.g., export amount).
- Color can show:
 - Different categories (by default)
 - o Conditional formatting (e.g., trade surplus in green, deficit in red).
- Tooltips show detailed data on hover.

5. Best Practices

- Use for relative comparison, not for trends.
- Combine with slicers (e.g., Year, Type: Import/Export).
- Limit the number of categories shown to avoid clutter.

6. Example Analysis Titles

- Country-wise Share of India's Imports (1997–2022)"
- Variable "Sectoral Contribution to India's Export Growth"



WATERFALL CHARTS:

Waterfall Chart – Power Bl Overview (India Trade Data)

• 1. Purpose

- Shows incremental changes in values over time or across categories.
- Helps identify how a value increased or decreased step-by-step.
- Great for showing contributions to a total (e.g., trade balance).

2. Structure

- Start column: Beginning value (e.g., trade balance in 1997).
- Rising bars: Positive changes (e.g., export growth).
- Falling bars: Negative changes (e.g., import spikes).
- End column: Final total (e.g., trade balance in 2022).

3. Use Cases for India's Trade

Use Case	Description
Year-over-Year Trade Balance	how trade surplus/deficit changed yearly.
Import Cost Change Breakdown	total change into sectors (e.g., oil, gold, electronics).
Export Revenue Growth Breakdown	Identify which sectors contributed most to export increase.

4. Key Features

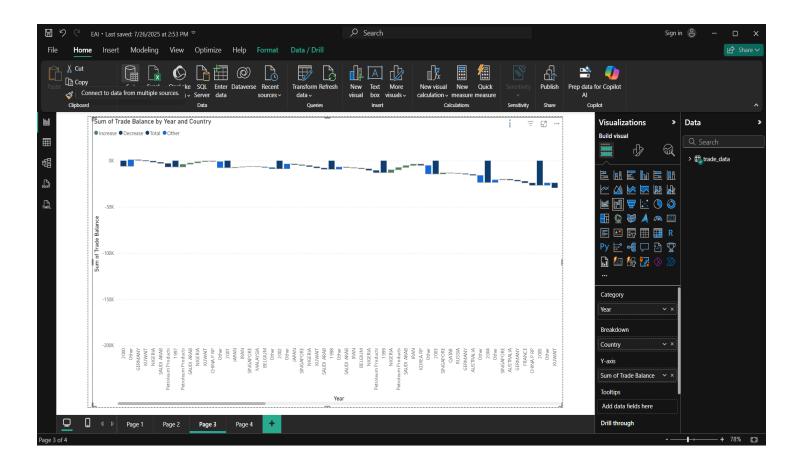
- Automatically calculates increases and decreases.
- Breakdown field: Allows category-wise contribution (e.g., sector-wise).
- Tooltips: Show exact values and deltas on hover.
- Total bars: Can be toggled on/off.

5. Best Practices

- Keep category names short for clarity.
- Use distinct colors for increase (green) and decrease (red).
- Highlight key categories or years with data labels.

6. Example Analysis Titles

- V "How India's Trade Deficit Widened from 1997 to 2022"
- Contribution of Import Categories to Total Cost − 2022"
- V "Year-wise Impact on Export Revenue Growth"



MATRIX CHARTS:

Matrix Chart – Power Bl Overview (India Trade Data)

• 1. Purpose

- Displays data in tabular format with the power of pivot tables.
- Allows you to show hierarchies, row/column groups, and aggregates.
- Ideal for detailed comparisons across multiple dimensions.

2. Structure

- Rows: Typically a category like Year or Country.
- Columns: Another category (e.g., Import/Export, Sector).
- Values: Numerical data (e.g., trade amount, growth %, etc.).

Use Case	Setup Example		
Year-wise Export/Import Summary	YearColumns: Trade TypeValues: USD		
Ocuntry-wise Trade with India	CountryColumns: Import/ExportValues: Amount		
Sector Contribution to Exports (by year)	SectorColumns: YearValues: Export Value		
Trade Balance Comparison	YearColumns: Import vs ExportValues: Difference (Calculated Measure)		

4. Key Features

- Drill-down: Expand/collapse rows or columns for hierarchy (e.g., Sector → Subsector).
- Conditional Formatting: Add color scales or data bars.
- Totals/Subtotals: Automatically displayed and customizable.
- Sorting: By rows, columns, or values.
- Tooltips: Show detailed info on hover.

5. Best Practices

- Use column/row formatting to highlight key data.
- Hide unnecessary totals for clarity.
- Use hierarchical fields to reduce clutter.
- Avoid overcrowding with too many columns—consider slicers or filters.

6. Example Titles

- V "India's Year-wise Import & Export Matrix (1997–2022)"
- Country-Wise Trade with India: A Matrix View"
- V "Sector-Wise Export Trends Over Time"

