2021

Big Data Architecture & Governance



**Student Names:**

Dileep Ravindranath Holla  
Shaunak Prabhu

**Assignment Name:**Team 6 - Group Assignment  
Global Trade Statistics

Contents

[2. Assignment 3](#_Toc70108115)

[2.1. Case 3](#_Toc70108116)

[2.2. Assignment Goals 3](#_Toc70108117)

[2.2.1. Visualization Deliverables 3](#_Toc70108118)

[2.2.2. Other deliverables 4](#_Toc70108119)

[3. Documentation 5](#_Toc70108120)

[3.1. Introduction 5](#_Toc70108121)

[3.2. Vision Diagram 5](#_Toc70108122)

[Vision diagram 5](#_Toc70108123)

[Graph Data Model 7](#_Toc70108124)

[3.3. Data Profiling 8](#_Toc70108125)

[Pandas Profiling Report 8](#_Toc70108126)

[Pandas Profiling warnings 8](#_Toc70108127)

[Column Profiling 9](#_Toc70108128)

[3.4. Data Wrangling and Cleansing 11](#_Toc70108129)

[3.5. Database Installation 13](#_Toc70108130)

[Mac Installation (dmg) 13](#_Toc70108131)

[Windows 13](#_Toc70108132)

[Create and Configure Database 13](#_Toc70108133)

[3.6. Data Mapping and Integration 14](#_Toc70108134)

[Data Mapping 14](#_Toc70108135)

[Data Integration 14](#_Toc70108136)

[Workflow 15](#_Toc70108137)

[Cypher To load Neo4j 17](#_Toc70108138)

[Graph Data Model in Neo4j: 19](#_Toc70108139)

[3.7. Business and Technical Metadata 19](#_Toc70108140)

[Technical Metadata 19](#_Toc70108141)

[Business Metadata 20](#_Toc70108142)

[3.8. Data Validation and Data Visualization 20](#_Toc70108143)

[Data Validation 20](#_Toc70108144)

[Data visualization 22](#_Toc70108145)

[key insights derived from the visualizations 23](#_Toc70108146)

[Metrics being assessed and visualized. 23](#_Toc70108147)

[Data Validation for Visualizations 33](#_Toc70108148)

[Codebase for visualization 36](#_Toc70108149)

[3.9. System Integration and User Acceptance Testing 36](#_Toc70108150)

[System integration tests 36](#_Toc70108151)

[3.10. Challenges Encountered 41](#_Toc70108152)

[3.11. End User Instructions 43](#_Toc70108153)

[End user instructions for neo4j 43](#_Toc70108154)

[End user instructions for Data Loading & Ingestion 43](#_Toc70108155)

[End user instructions for Viewing the dashboard 44](#_Toc70108156)

[4.0 Velero Screenshots 44](#_Toc70108157)

[Project Milestones/Tasks 44](#_Toc70108158)

[Timesheet 45](#_Toc70108159)

# Assignment

## Case

Each team should select a dataset to analyze and build an analytical dashboard as a Proof-of-concept to illustrate the value of data driven analytics. You need to present your dataset.

## Assignment Goals

To work with datasets, Perform/Create:

* Group Assignment/Project on Velero with below mentioned activities:
  + Tasks/ Project Short/ Long Form/ Group Allocation/ Timesheet/ Issues & Risks.
* Data Profiling – Using Python profiling library, describe your understanding of the data.
* Data Wrangling and Cleansing - Pandas/Alteryx/XSV
  + Filtering and aggregating if needed.
  + Missing value handling.
  + Deriving additional columns from existing datasets if needed.
  + Cleaning (removing blank spaces, formatting dates, Capitalizing etc.).
* Database Installation: Install NEO4J database.
* Data Mapping and Integration to your Database for the Entire Dataset.
* Business and Technical Metadata – develop business term list describing all the data elements available in the file.
* Data Validation and Data Visualization using Python – Validate the data using python and provide a dashboard using python visualization libraries.
* System Integration and User Acceptance Testing - Test Cases – describe your validation & testing process.
* Risks/Issues of project.
* Describe challenges encountered and how you resolved them.
* End User Instructions (Steps to run your Dashboard) – provide a full description how to run your process:
  + Database Creation and load.
  + Visualization interpretation - describe information regarding your findings.

### Visualization Deliverables

Once you wrangle/clean/join/integrate the data, import the data into **NEO4J** and illustrate how to use the appropriate graph to illustrate various aspects of analysis.

Questions to consider:

* Columns used for dimensions, and columns that are used for measurement.
* How would you generate new dimensions?
* Who would use this dashboard and how they benefit from your dashboard?
* What value would be generated using this dashboard?

### Other deliverables

* Presentation of the entire work from the first step till the dashboards including the Velero screenshots.
* Business and technical metadata presentation – Identifying all available business terms and extracting related technical metadata.
* Complete explanation of the dashboard and usability.
* Complete instruction as how to implement and run the database load, technical meta data extraction, and dashboard.

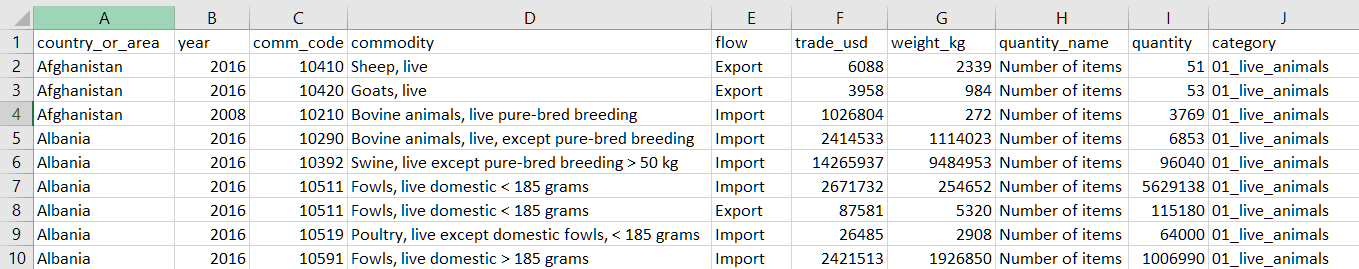
# Documentation

## Introduction

The dataset is the Global Commodity Trade Statistics dataset published by the United Nations. This dataset covers import and export volumes for 5,000 commodities across most countries on Earth over the last 30 years. The dataset was published by the United Nations Statistics Division on the UN Data site and has 10 fields and contains over 8 million records.

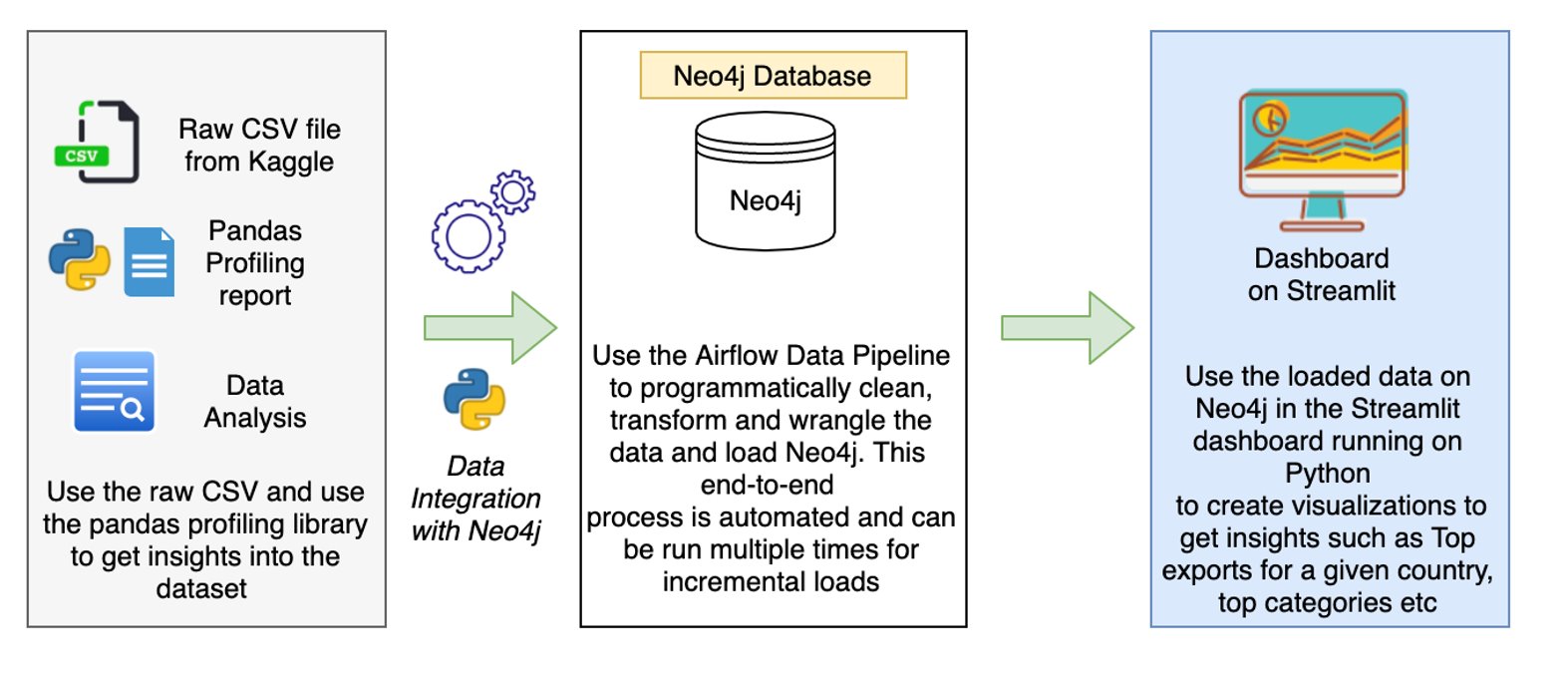
This dataset can be used to draw interesting insights on which countries trade on which commodities as well as the trade weights, trade values and how trades increased or decreased in nations over time.

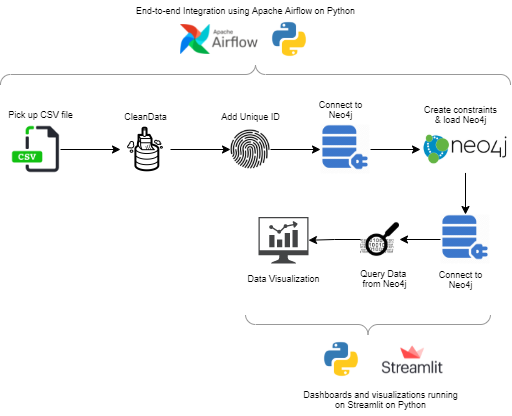
**Sample head of the data fields:**



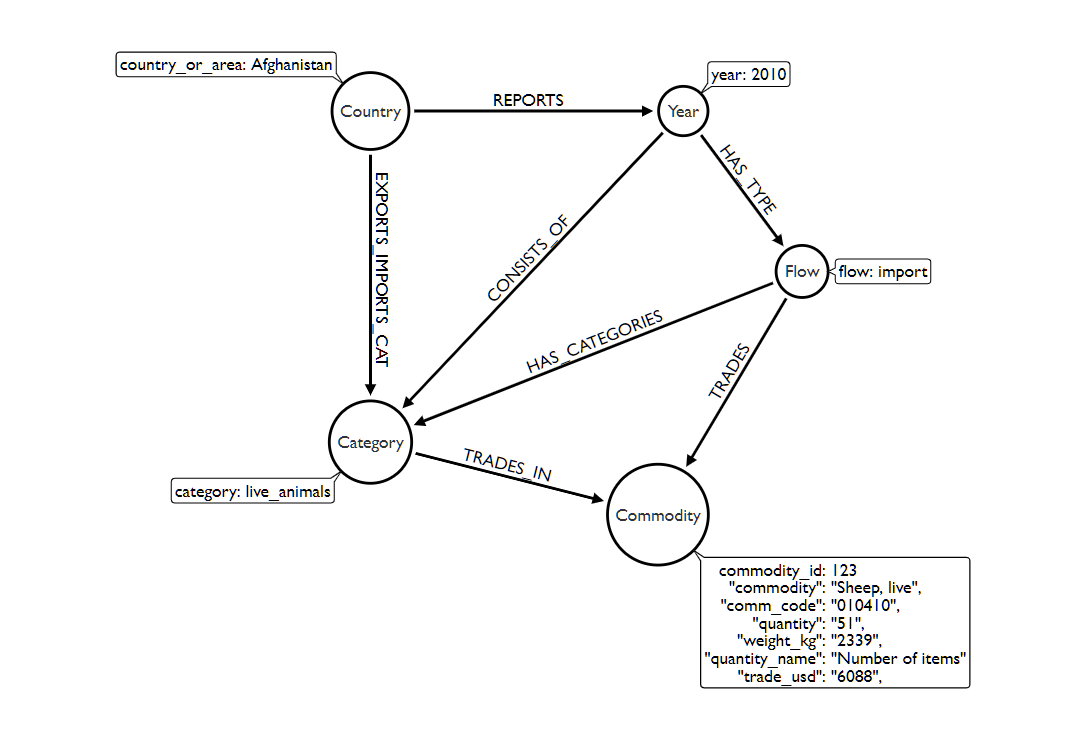
## Vision Diagram

### Vision diagram





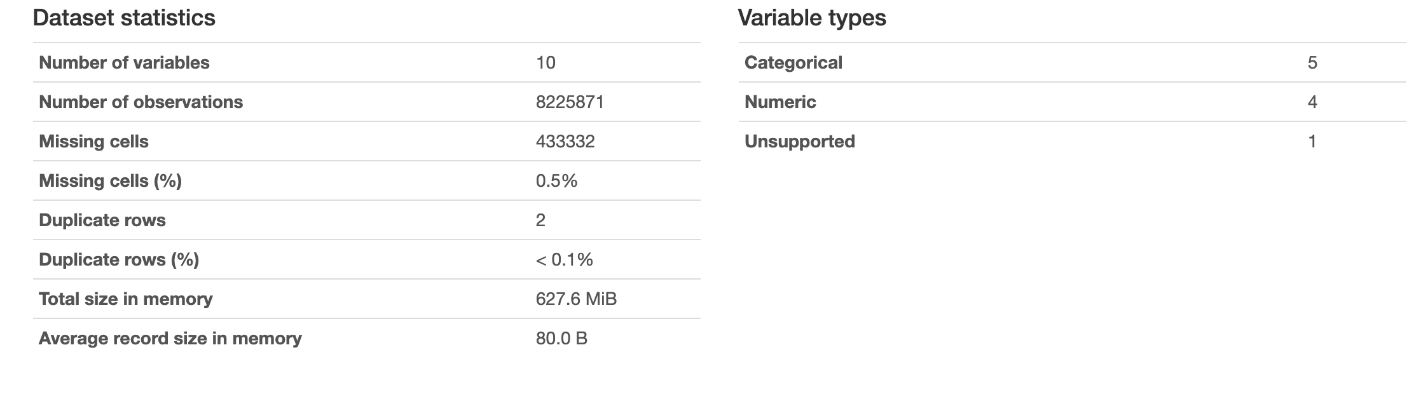
### Graph Data Model



## Data Profiling

Pandas profiling library was used to analyze the raw csv file to get a better understanding of each of the columns as well as what steps would be required to wrangle and clean the data before inserting it into neo4j.

### Pandas Profiling Report



From the summary above we get a rough overview of the tables and the data consistency. The dataset has 0.5% of fields with missing values which can’t be dropped as these are over 430,000 records.

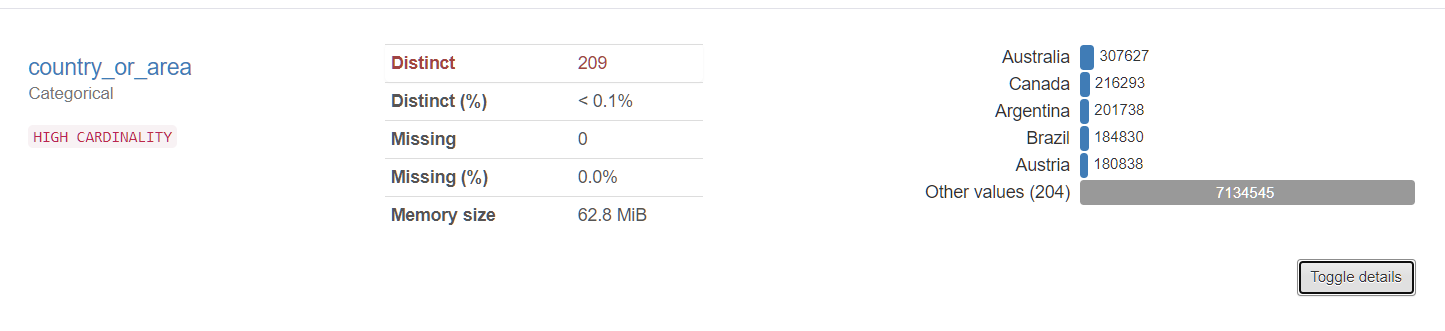
### Pandas Profiling warnings



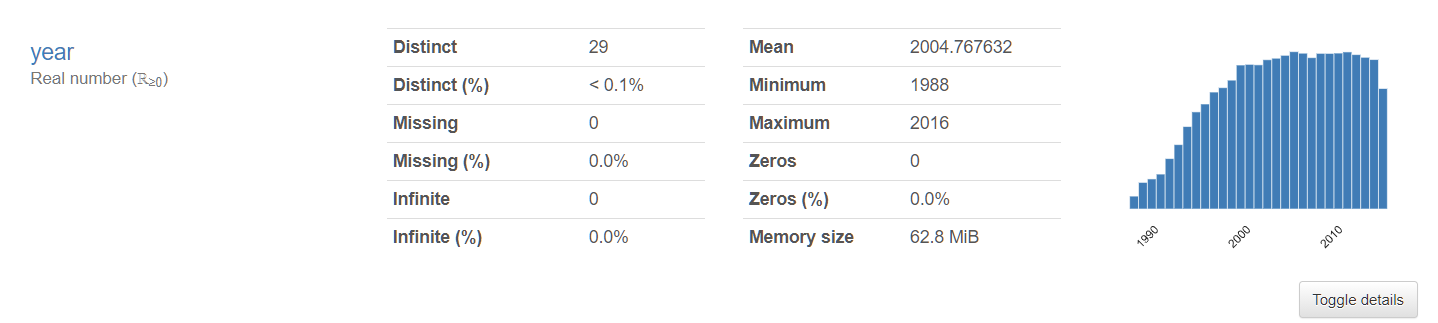
As the dataset is exceptionally large it has a high degree of cardinality which may cause issues when inserting into neo4j if not done properly. There are missing values primarily in the ‘weight\_kg’ column as well as the ‘quantity’ column.

### Column Profiling

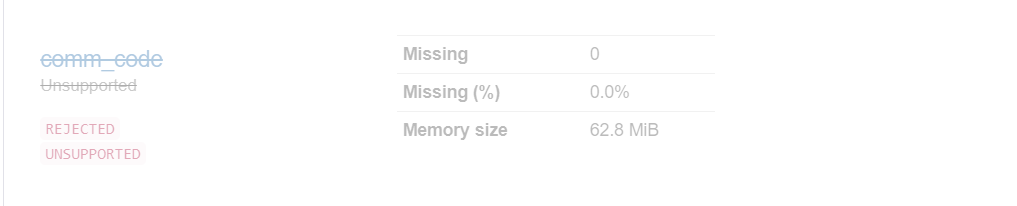
1. **Country:** There are 209 unique values with zero missing values therefore no cleaning is required for this field.



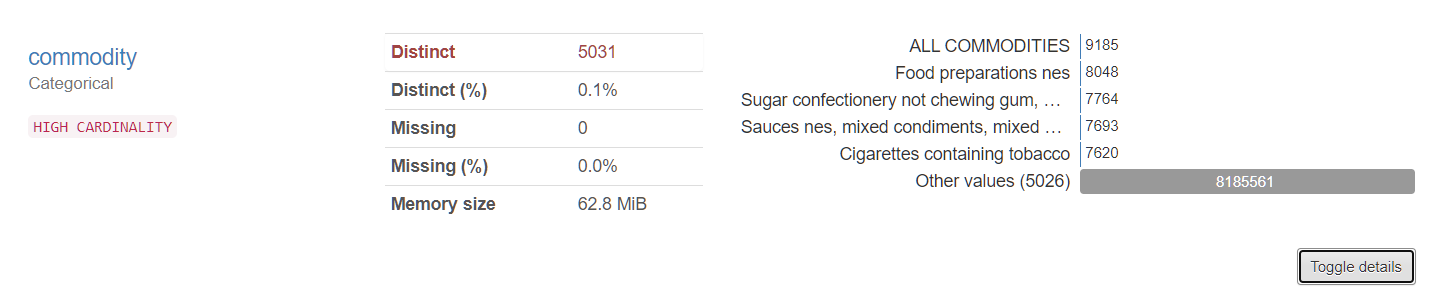
1. **Year:** The dataset spans 29 years from 1988 to 2016. Trade can be seen to increase in quantity until the early 2000’s after which it flatlines in volume.



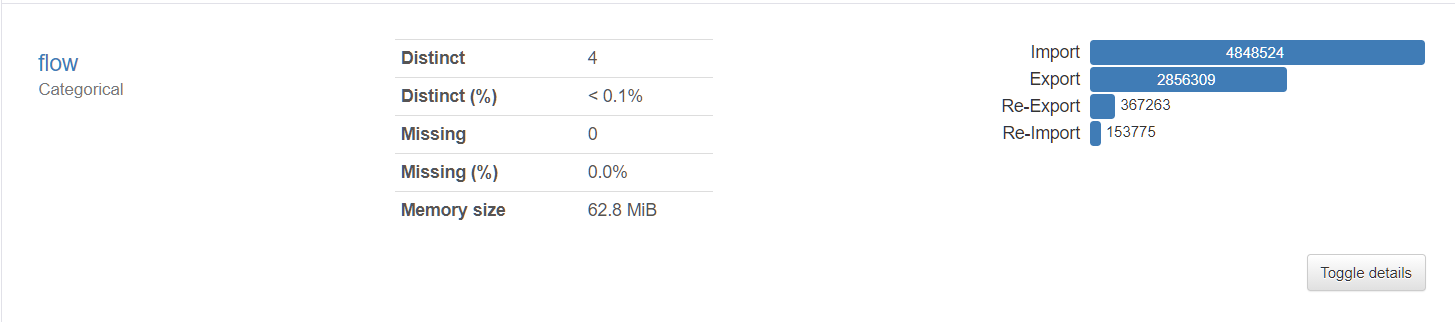
1. **Comm\_code:** This field seems to be of a different data type but could still be inserted into neo4j as the DB by default converts all fields to string data type on loading the csv.



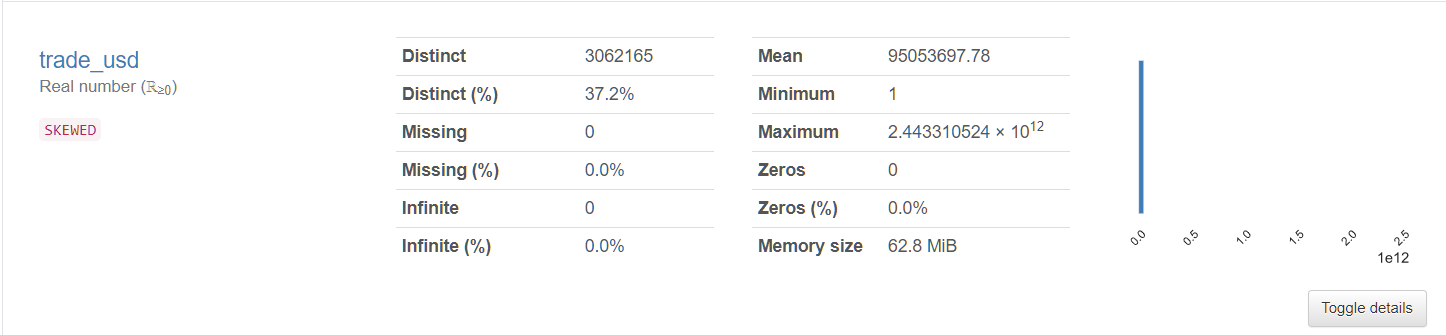
1. **Commodity:** Each commodity is part of a category. Has high cardinality.



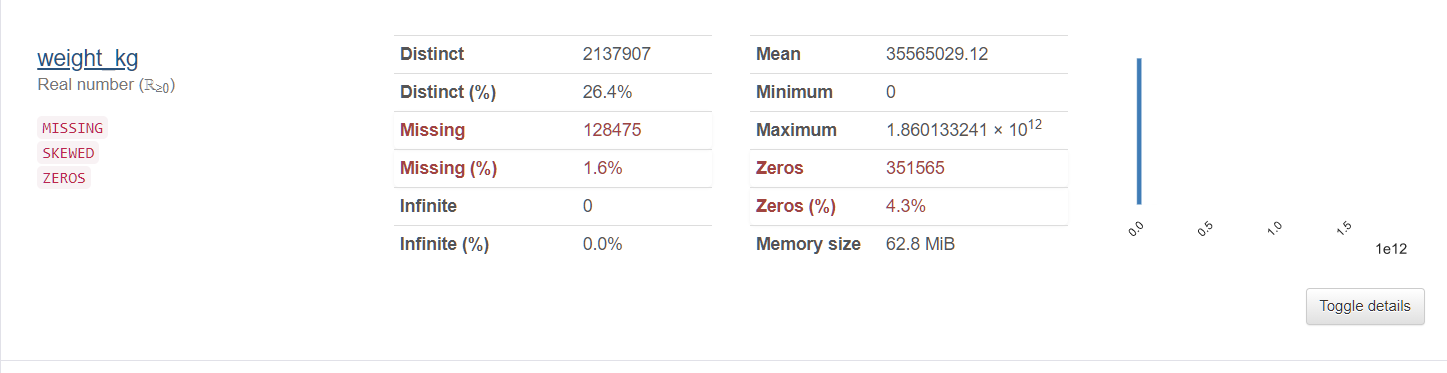
1. **Flow:** Here we can see the trades that were categorized as import, export, re-import, and re-export



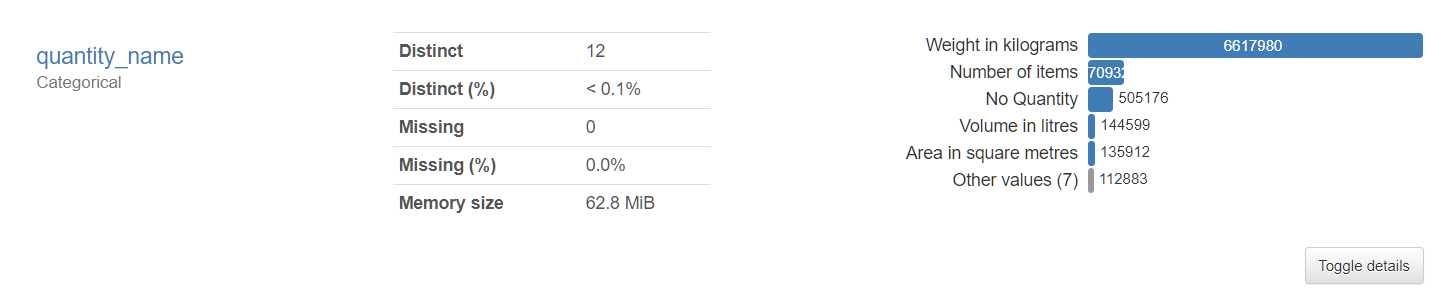
1. **Trade\_usd:** The trade column does not require further cleaning as there are no null values.



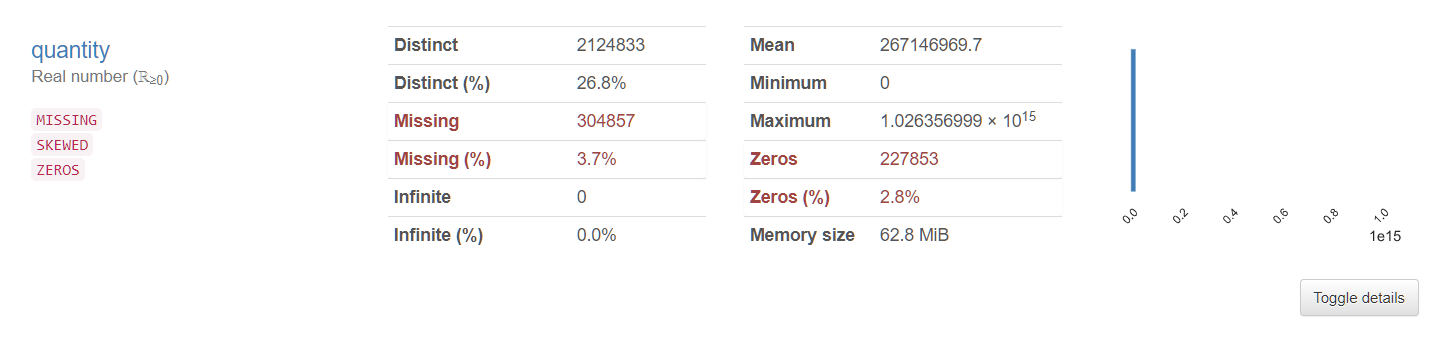
1. **Weight\_kg:** This column contains the weight of each trade done in kg’s. This field contains a majority of the missing values.



1. **Quantity\_name**: Specifies the trade measurement.



1. **Quantity:** Number of units of each trade according to the quantity name specified. Has large percent of missing values.



1. **Category:** Has 98 distinct values. No cleaning required.

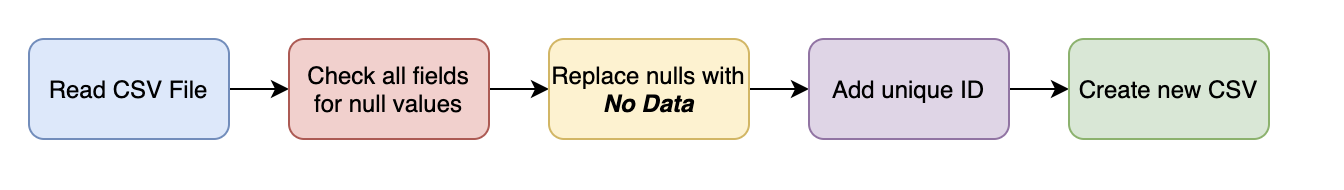


## Data Wrangling and Cleansing

After profiling the data, we can move ahead with cleaning the data according to the columns. The dataset already has a high level of usability, but a certain degree of wrangling and cleaning must be done before inserting the data into the database. The following steps were taken to ensure this:

* Taking care of the fields quantity and weight\_kg due to null or empty values.
* As an added check – all columns were checked for null/empty values. All empty values were replaced with ***No Data.***
* As part of the data wrangling and cleansing process – a unique identifier was added to each record in the file.
* The other fields in the dataset were clean and did not require any added data wrangling or cleansing.
* The Python code for data wrangling is available in the **dataCleansing\_loadNeo4j.ipynb** file.
* Additionally – this entire process is done as part of the data pipeline on Apache Airflow. This is covered in the next section.

**General workflow for the Data Cleansing process:**



**Code Snippets:**

The Python code for data wrangling is available in the **dataCleansing\_loadNeo4j.ipynb** file. Additionally, this entire data cleaning and wrangling process has been a part of **the Airflow Data pipeline** and the standalone script is available in the Jupyter notebook.

Code to read the file, remove all nulls and add unique identifier to the data:



## Database Installation

### Mac Installation (dmg)

The installer includes the Java version needed for running Neo4j.

* Open the dmg file you just downloaded.
* Drag the Neo4j icon into your Applications folder. Watch the video to see this in action.
* Open Neo4j from your Applications folder. You might need to acknowledge that you downloaded the application from the Internet.
* Click on the Start button to start the Neo4j server.
* Open the provided URL in your local web browser.
* Change the password for the neo4j account.

### Windows

* The installer includes the Java version needed for running Neo4j.
* Launch the installer you just downloaded. You might have to give the installer permission to make changes to your computer.
* Follow the prompts and choose the option to Run Neo4j.
* Click on the Start button to start the Neo4j server.
* Open the provided URL in your local web browser.
* Change the password for the neo4j account.

### Create and Configure Database

* Open Neo4j
* From the Neo4j Desktop home screen, create a new project.
* Give the new project a name and click on the + icon to create a new Local DBMS.
* Provide the Database name and password to create the new database.
* Once the database is created – start the database and open the Neo4j Browser which will allow you to run queries.

## Data Mapping and Integration

After cleaning and wrangling the raw dataset – the data is loaded to Neo4j using a data pipeline running on Apache Airflow.

### Data Mapping

The below table illustrates the fields in the source CSV fields and their mapping to Neo4j nodes and corresponding properties for the respective nodes.

|  |  |  |
| --- | --- | --- |
| Raw CSV Field | Node | Property |
| comm\_code | Commodity | CommodityCode |
| quantity | Commodity | Quantity |
| quantity\_name | Commodity | QuantityName |
| trade\_usd | Commodity | TradeValue |
| commodity | Commodity | CommodityName |
| weight\_kg | Commodity | Weight |
| category | Category | category |
| country\_or\_area | Country | country |
| flow | Flow | flow |
| year | Year | year |

### Data Integration

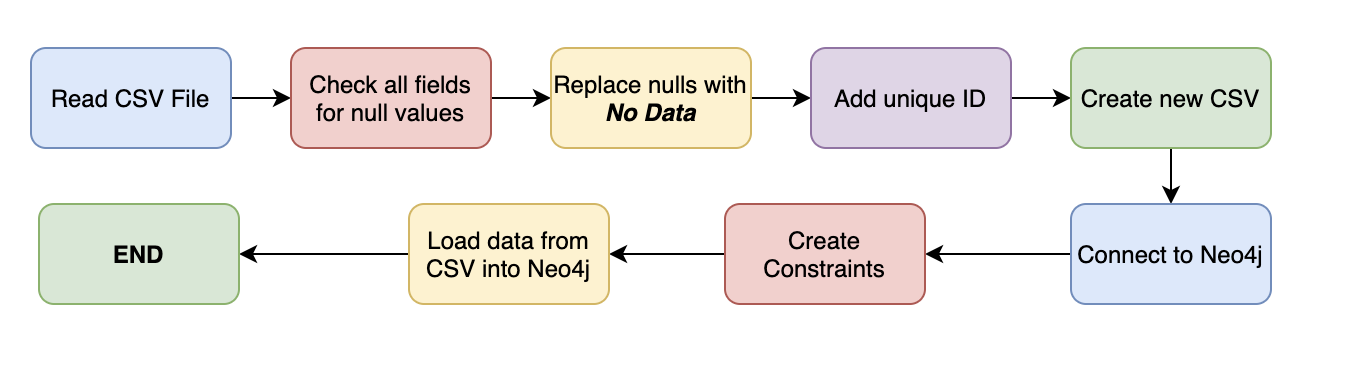
**Method of Integration: Data Pipeline**  
**Language: Python**  
**Data Integration Framework: Apache Airflow**

* Complete end-to-end integration is made possible with the help of a data pipeline.
* Apache Airflow was chosen as the preferred data pipelining tool since it is open-source and free to use.
* Apache Airflow also offers a very convenient user-interface from where the data ingestion pipeline can be monitored and triggered on demand.
* The Airflow webserver was deployed on a local machine running on OSX – although it is compatible with Windows.
* The code for Airflow pipeline is available in **team6\_pipeline.py** Python script.
* To integrate both the cleansing and loading process, the pipeline takes the source file as the input, cleans, and transforms the data and loads the Neo4j database.

**Advantages of using Apache Airflow:**

* Data pipeline is highly modular and can be re-used to load Neo4j with other datasets.
* UI can track multiple incremental/delta loads over time.
* Ability to track and report any failures during the data ingestion or cleansing process.

### Workflow



**Airflow Login Screen:**

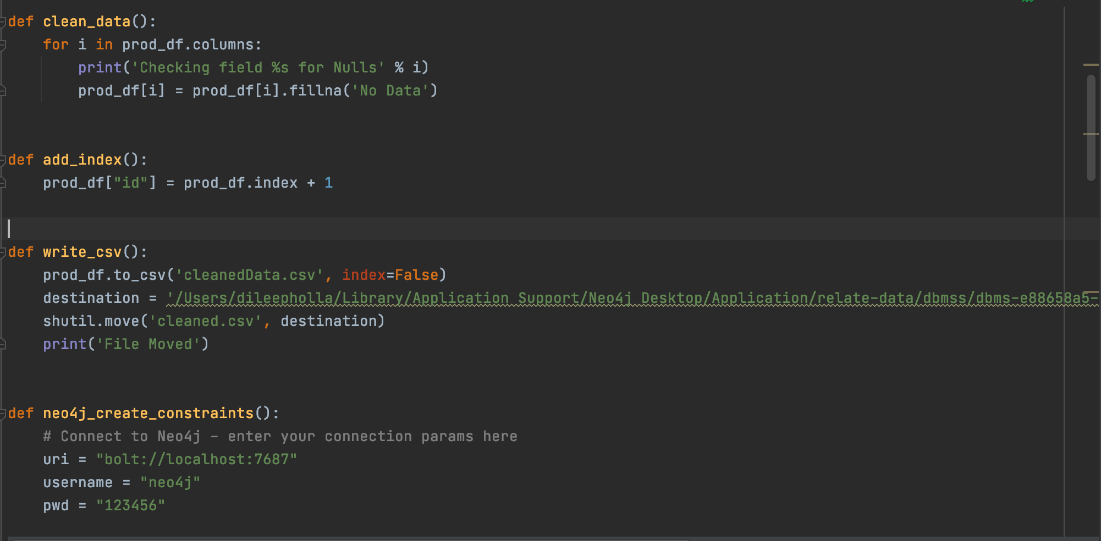


**Data Pipeline for cleaning and data ingestion into Neo4j:**



**Code Snippets – for Airflow Orchestration:**

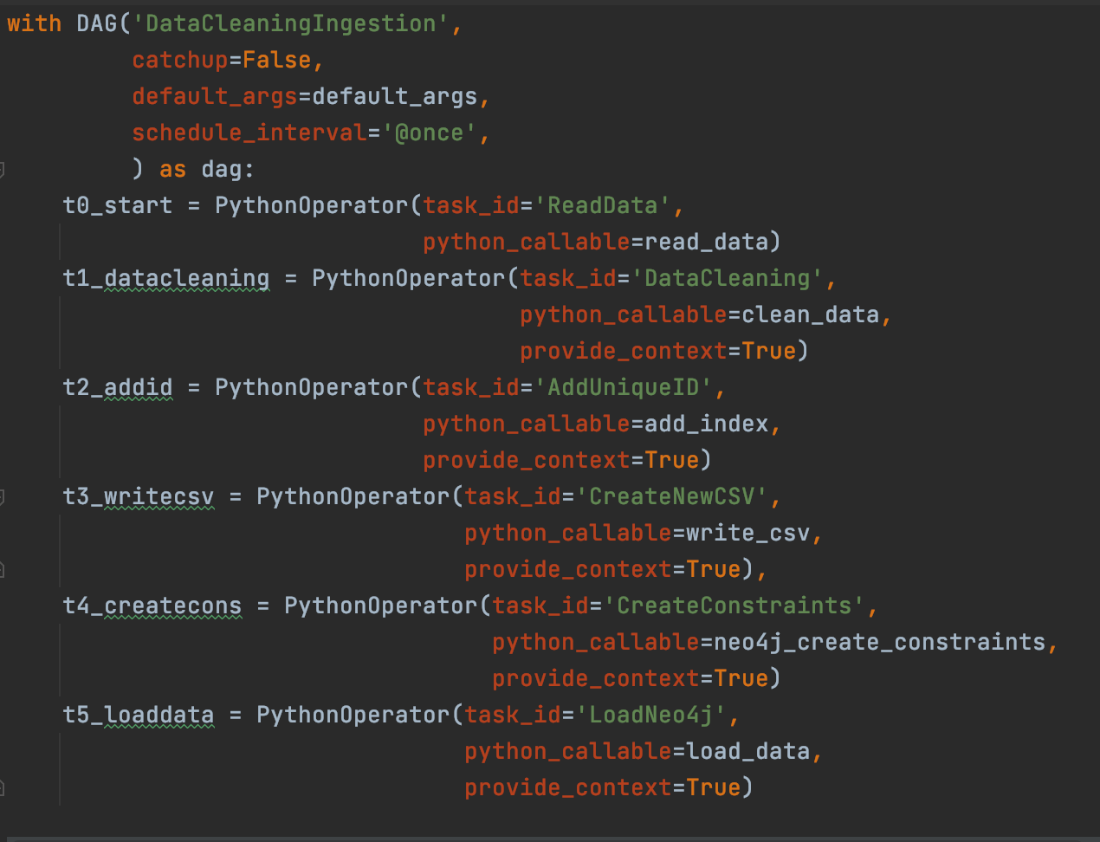
Individual tasks that are orchestrated together using Airflow:







**Task Orchestration:**



### Cypher To load Neo4j

**//Creating constraints for the data**

CREATE CONSTRAINT com IF NOT EXISTS ON (com:Commodity) ASSERT com.id IS UNIQUE;  
CREATE CONSTRAINT flow IF NOT EXISTS ON (flo:Flow) ASSERT flo.flow IS UNIQUE;  
CREATE CONSTRAINT cat IF NOT EXISTS ON (cat:Category) ASSERT cat.category IS UNIQUE;  
CREATE CONSTRAINT country IF NOT EXISTS ON (cou:Country) ASSERT cou.country\_or\_area IS UNIQUE;  
CREATE CONSTRAINT year IF NOT EXISTS ON(yea:Year) ASSERT yea.year IS UNIQUE;

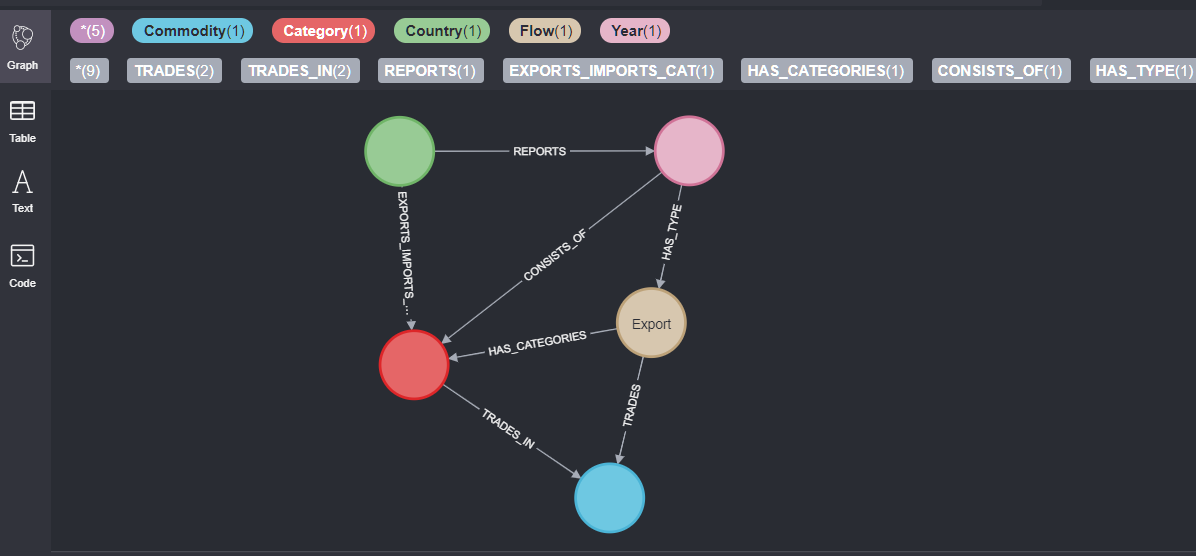
CREATE CONSTRAINT com IF NOT EXISTS ON (com:Commodity) ASSERT com.id IS UNIQUE;  
CREATE CONSTRAINT flow IF NOT EXISTS ON (flo:Flow) ASSERT flo.flow IS UNIQUE;  
CREATE CONSTRAINT cat IF NOT EXISTS ON (cat:Category) ASSERT cat.category IS UNIQUE;  
CREATE CONSTRAINT country IF NOT EXISTS ON (cou:Country) ASSERT cou.country\_or\_area IS UNIQUE;  
CREATE CONSTRAINT year IF NOT EXISTS ON(yea:Year) ASSERT yea.year IS UNIQUE;

**// Loading the csv file into neo4j**  
:auto using periodic commit   
LOAD CSV WITH HEADERS FROM "file:///cleaned2.csv" AS row

**//Performing a merge on the nodes**  
MERGE(com:Commodity{id:row.id})  
ON CREATE SET com.CommodityName=row.commodity, com.CommodityCode=row.comm\_code, com.Quantity= row.quantity, com.Weight= row.weight\_kg, com.TradeValue= row.trade\_usd, com.QuantityName=row.quantity\_name  
MERGE (cat: Category {category: row.category})   
MERGE (cou: Country {country: row.country\_or\_area})   
MERGE (flo: Flow {flow: row.flow})

**//Creating relationships between each of the nodes**  
MERGE (yea: Year {year: row.year})MERGE (cou)-[:REPORTS]->(yea)   
MERGE (cou)-[:EXPORTS\_IMPORTS\_CAT]->(cat)   
MERGE (yea)-[:CONSISTS\_OF]->(cat)   
MERGE (yea)-[:HAS\_TYPE]->(flo)   
MERGE (flo)-[:HAS\_CATEGORIES]-> (cat)   
MERGE (flo)-[:TRADES]-> (com)   
MERGE (cat)-[:TRADES\_IN]->(com);

### Graph Data Model in Neo4j:



## Business and Technical Metadata

The technical metadata was extracted from the Neo4j database using **export\_metadata.py** script. This can be found in the metadata directory.

### Technical Metadata

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| counts | label | property | dbName | relationships |
| 8225871 | Commodity | CommodityCode | CommodityDB | TRADES,BELONGS\_TO |
| 8225871 | Commodity | Quantity | CommodityDB | TRADES,BELONGS\_TO |
| 8225871 | Commodity | QuantityName | CommodityDB | TRADES,BELONGS\_TO |
| 8225871 | Commodity | TradeValue | CommodityDB | TRADES,BELONGS\_TO |
| 8225871 | Commodity | CommodityName | CommodityDB | TRADES,BELONGS\_TO |
| 8225871 | Commodity | Weight | CommodityDB | TRADES,BELONGS\_TO |
| 98 | Category | category | CommodityDB | BELONGS\_TO,EXPORTS\_IMPORTS\_CAT,HAS\_CATEGORIES |
| 209 | Country | country | CommodityDB | EXPORTS\_IMPORTS\_CAT,REPORTS |
| 4 | Flow | flow | CommodityDB | HAS\_TYPE,TRADES,HAS\_CATEGORIES |
| 29 | Year | year | CommodityDB | REPORTS,HAS\_TYPE |

Can also be found here: <https://docs.google.com/spreadsheets/d/1rTQeW2DbqpxNK0CBypdsx4ZgK3Kz0E8xk2FgXo37HqQ/edit#gid=684619117>

### Business Metadata

|  |  |  |
| --- | --- | --- |
| Domain Name | Business Term Name | Term Description |
| GlobalTradeStats | Country | Area or country taking part in exports/imports |
| GlobalTradeStats | Year | Year in which country exported/imported a given commodity |
| GlobalTradeStats | Commodity Code | Unique identifier for a commodity |
| GlobalTradeStats | Commodity | Commodity/Item being exported/imported by a country |
| GlobalTradeStats | Flow | Flow of trade i.e. Export, Import |
| GlobalTradeStats | Trade Value | Value of the trade in US Dollars |
| GlobalTradeStats | Weight | Weight of the commodity in Kilograms |
| GlobalTradeStats | Quantity Name | A description of the quantity measurement type given the type of item (i.e. Number of Items, Weight in KG) |
| GlobalTradeStats | Quantity | Count of the quantity of a given item based on the Quantity Name |
| GlobalTradeStats | Category | Category to identify commodity. |

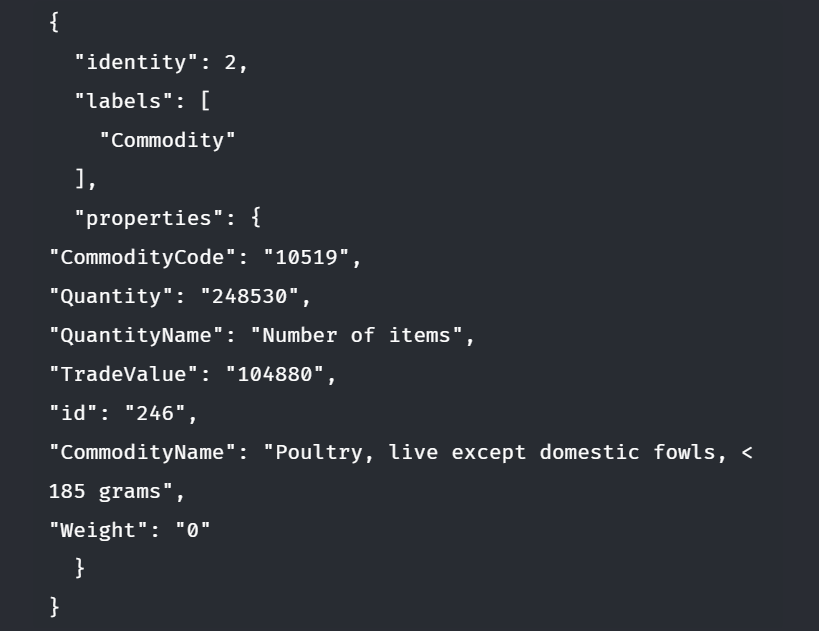
Can also be found here: <https://docs.google.com/spreadsheets/d/18YdO9N_KmMV57-l_2utxQ-i5AH2A66XCEnDV1Lyb_zs/edit?usp=sharing>

## Data Validation and Data Visualization

### Data Validation

Data Validation was done by comparing the data loaded on Neo4j with the data available in the raw CSV files.

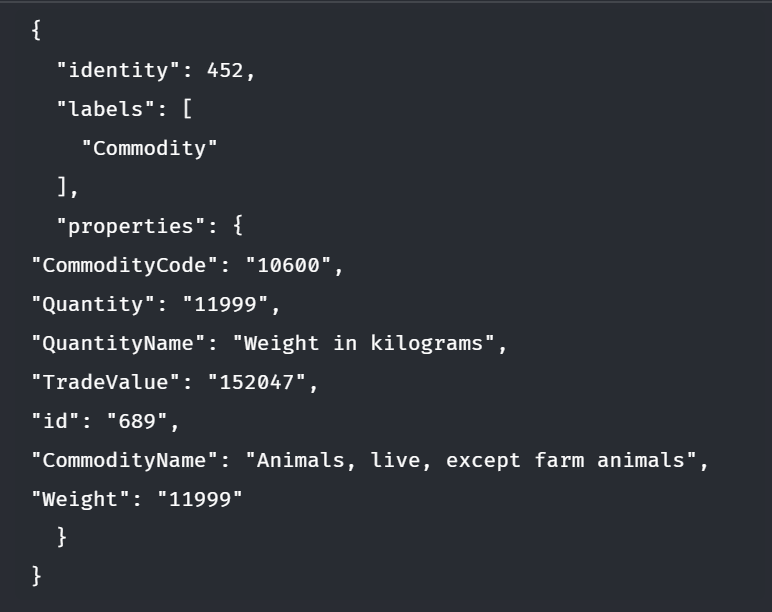
**Commodity Node on Neo4j for ID: 246**



**Corresponding Data on CSV:**

|  |  |
| --- | --- |
| id | 245 |
| comm\_code | 10511 |
| commodity | Fowls, live domestic < 185 grams |
| trade\_usd | 446192 |
| weight\_kg | 0 |
| quantity\_name | Number of items |
| quantity | 422122 |

**Commodity Node on Neo4j for ID: 689**



**Corresponding Data on CSV:**

|  |  |
| --- | --- |
| id | 689 |
| comm\_code | 10600 |
| commodity | Animals, live, except farm animals |
| trade\_usd | 152047 |
| weight\_kg | 11999 |
| quantity\_name | Weight in kilograms |
| quantity | 11999 |

### Data visualization

**Language: Python**  
**Data Visualization Framework: Streamlit (**[**https://streamlit.io/**](https://streamlit.io/)**)**

* The dashboard supplies value to potential business-users who would want to track and visualize global trade statistics in a single web application.
* The end-users would be business analysts who would want to track how a country or a product category is being traded around the world.
* The visualizations are written in Python by using Plotly, matplotlib and other visualization libraries. The visualizations are rendered by Streamlit on a web-browser.
* Business users need not write Python code or run Python scripts to get insights from the data – deploying the entire visualization and reporting framework as a web-application ensures that any business users can just visit the application on their browsers and view metrics.
* The dashboard/visualization offers multiple ways to slice and filter the data.
* In most of the visualizations – there is functionality to filter data by country, flow (exports, imports etc.), product categories etc.
* Filters in the visualizations allow users to slice the data as required.
* Filters are contextual, and change based on the selected visualization type.

**Why Streamlit?**

* Streamlit makes it extremely easy to deploy a stand-alone visualization as a web application.
* Easy to code and uses Python as the programming language.
* Extremely easy to integrate with Pandas dataframes.

### key insights derived from the visualizations

* China has emerged as a key player in the global trade market with its exports constantly on the rise.
* China is the dominating exporter in global trade today, and that it experienced a spectacular growth in exports after the WTO accession in 2001.
* EU’s biggest imports continue to be petroleum products and minerals.
* Among the tariff goods, Aircraft related products has the highest trading value, more that 97.5%. It carries almost the whole import market for USA.
* We can see Chinese import goods is more diversified, Auto, Aluminum and aircraft has high ratios.
* Petroleum products and fuel account for the highest imports across all categories.

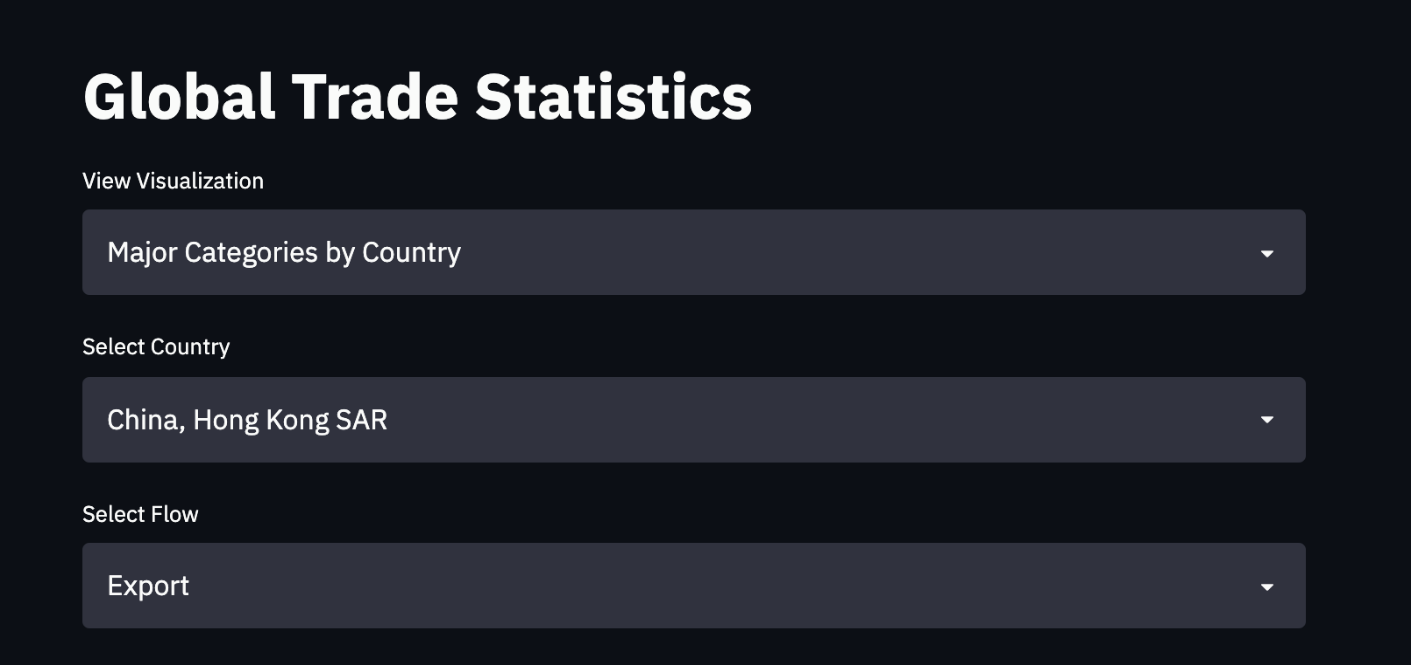
### Metrics being assessed and visualized.

|  |  |  |
| --- | --- | --- |
| **Metrics** | **Description** | **New Measures** |
| Total trade by country | Gives the total percentage of exports, imports, re-exports, re-imports by a given country. | New measure created with a sum of all trades by a given flow (exports/imports) |
| Major Categories by Country | Gives the breakdown of exports/imports by top categories for a selected country. | Measure created to aggregate data by a given country and category. |
| Exports/Imports by Country | Time series visualization of trade value against year for a given country. | New measure created with aggregates of total trade value by country. |
| Exports/Imports by Weight | Time series visualization of total weight against year for a given country. | New measure created with aggregates of all total weight by country and flow. |
| Exports/Imports by Quantity | Time series visualization of total units against year for a given country. | New measure created with aggregates of all total quantity by country and flow. |
| Global Trade by Product Category | Total value of exports/imports by category over time | New measure created with aggregated trade value by country and category. |
| Top 10 Countries | Top 10 Countries by exports/imports |  |
| Top 10 Categories | Top 10 categories by exports/imports |  |

**Visualization Options**



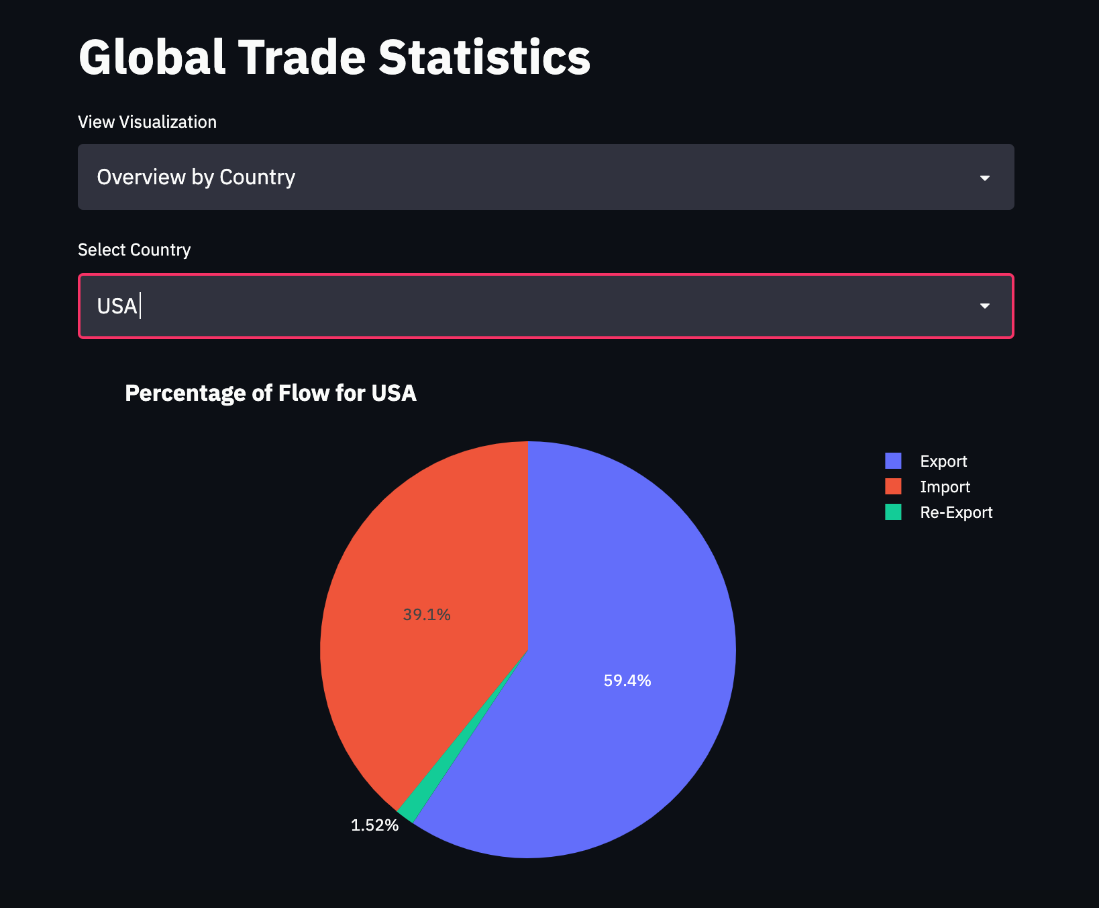
**Filters – the filters are contextual, and change based on the selected visualization type:**



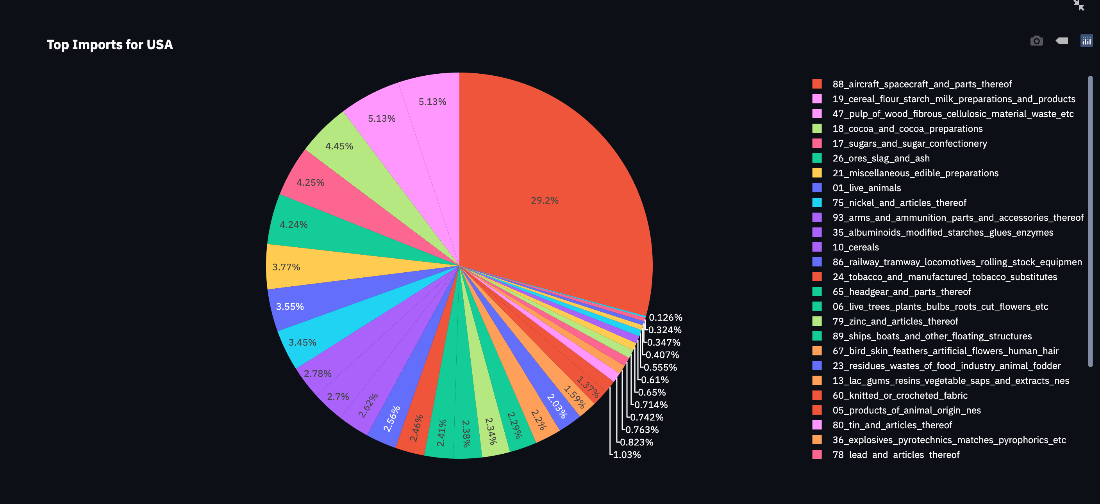
Using the visualizations constructed above along with the filters we can construct a large array of visualizations that are interactive and can be adjusted using the drop-down menus above.

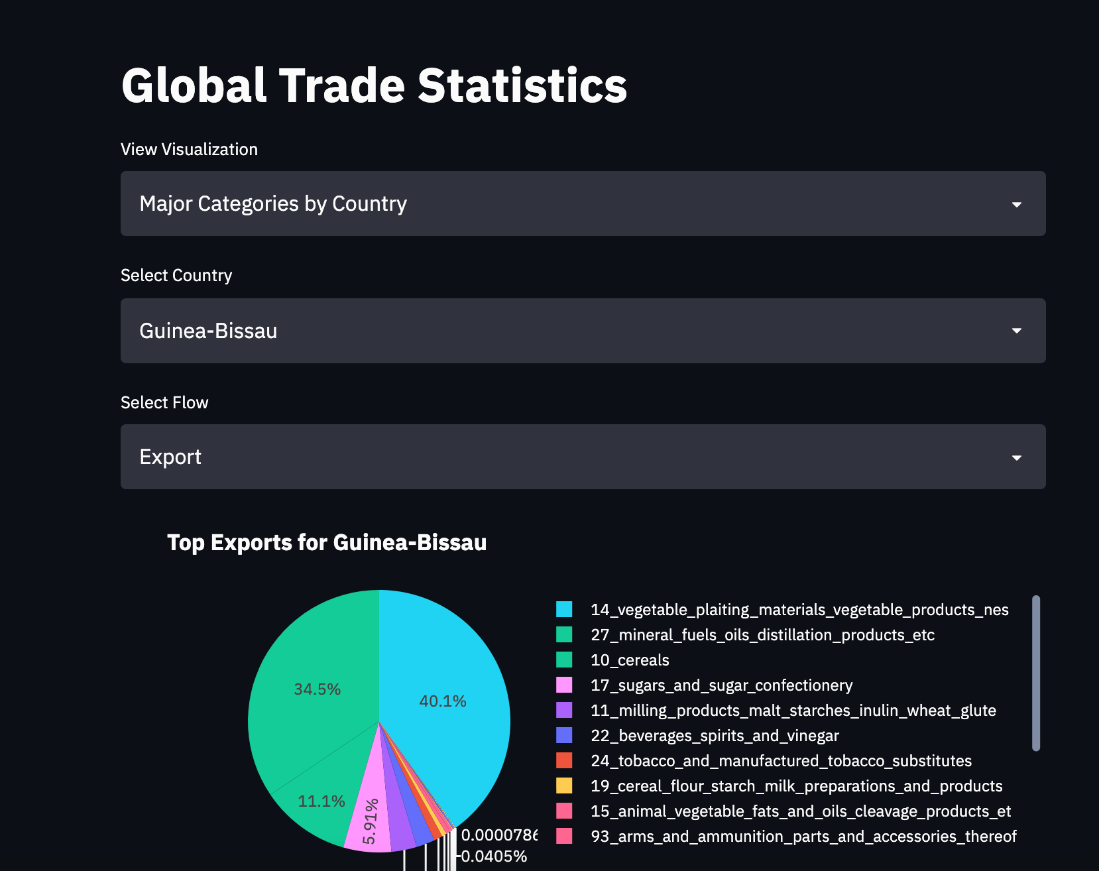
Some sample visualizations have been constructed below through which we can generate some valuable inferences:

1. **Total percentage of exports, imports, re-exports, re-imports by a given country.**



1. **Breakdown of exports/imports by top categories for a selected country.**

c



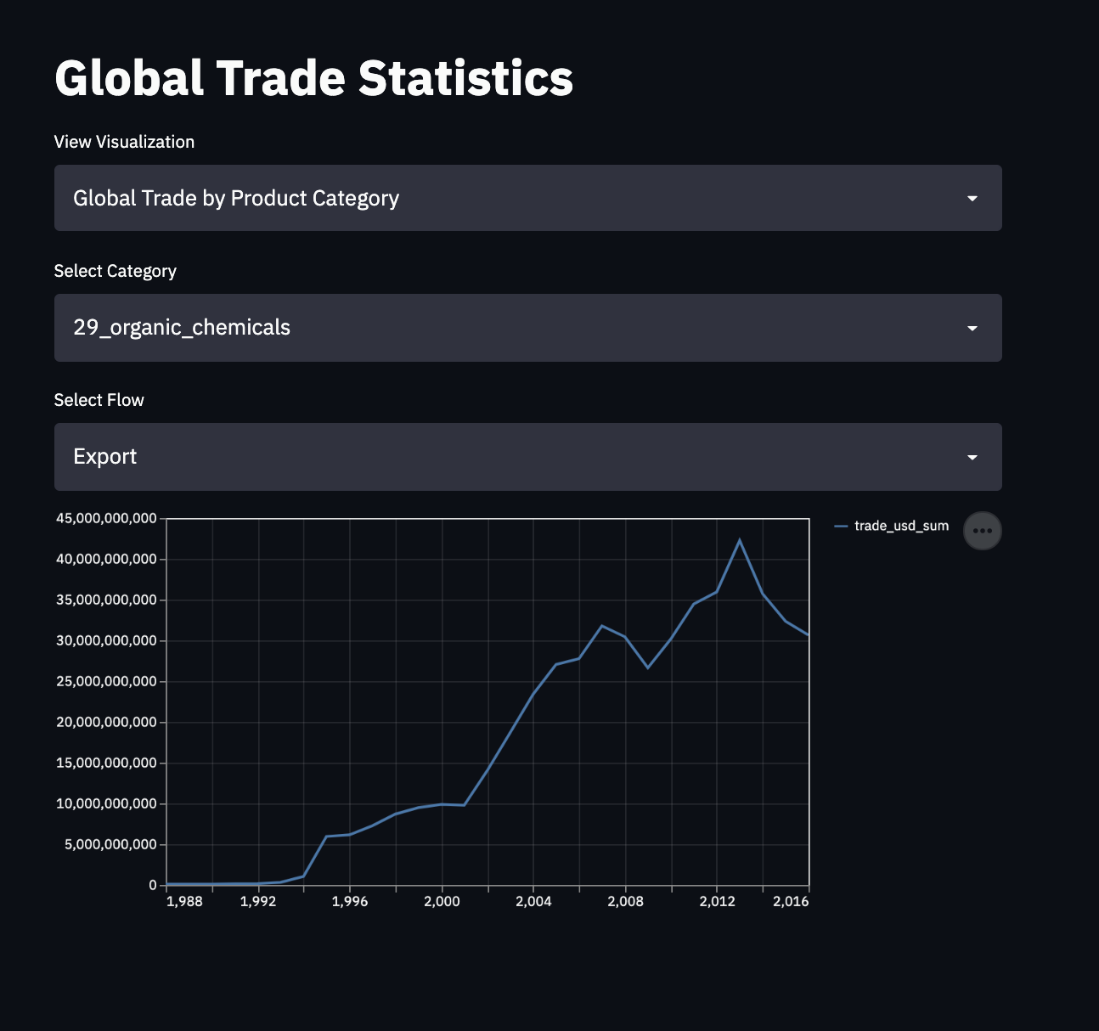
1. **Time series visualization of trade value against year for a given country.**



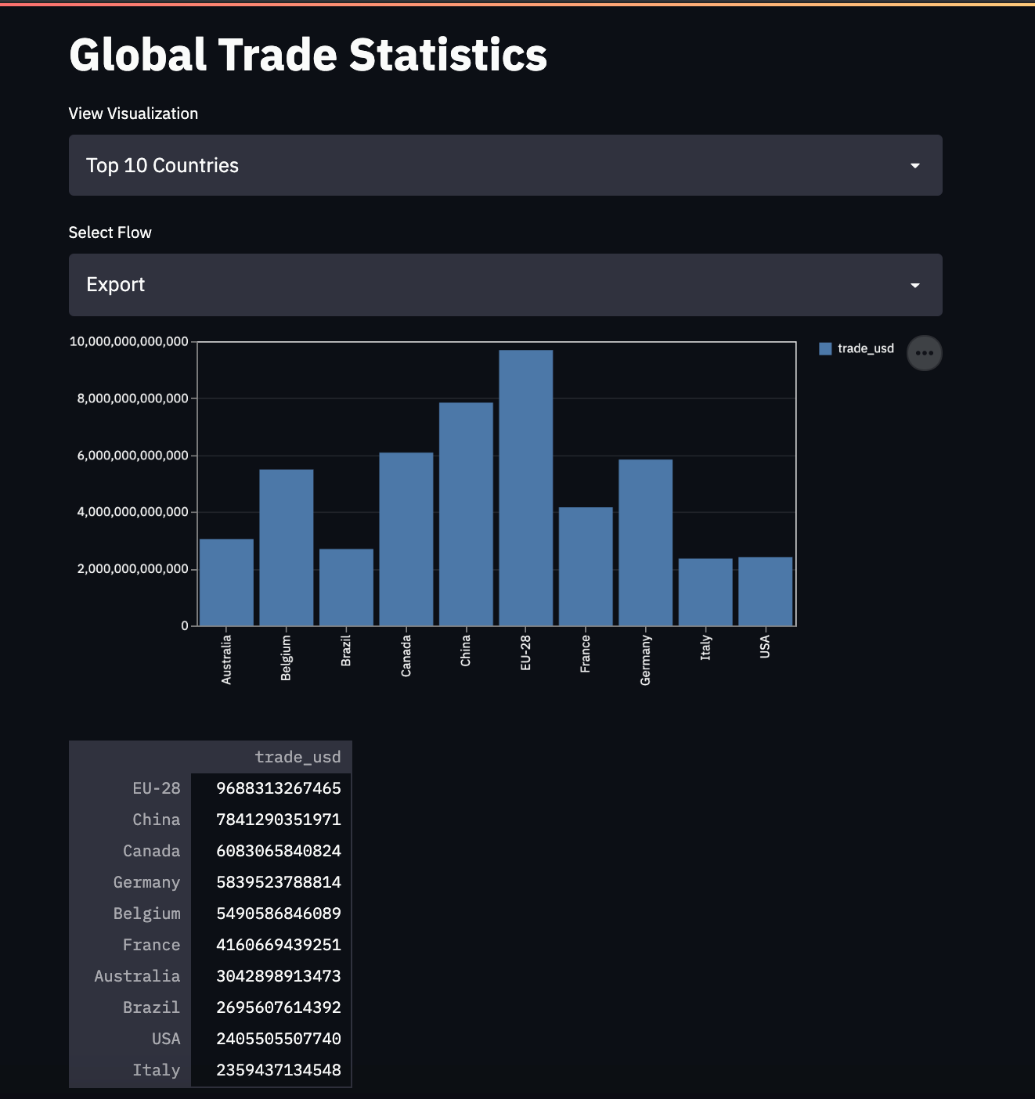
1. **Visualization showing the exports/imports for a given country over time.**



1. **Total value of exports/imports by category over time**



1. **Top 10 Countries for Exports**



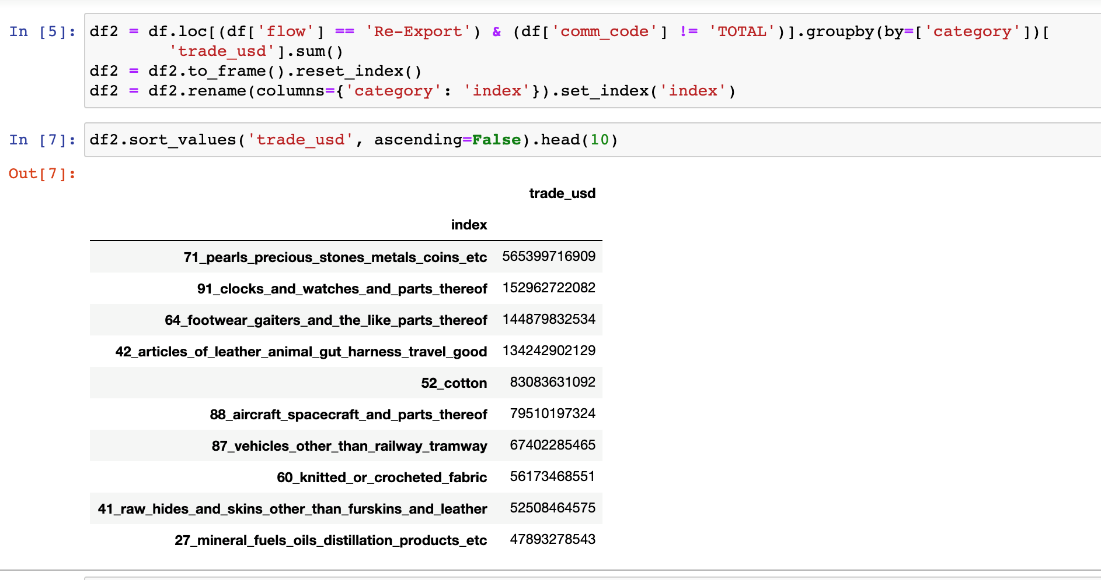
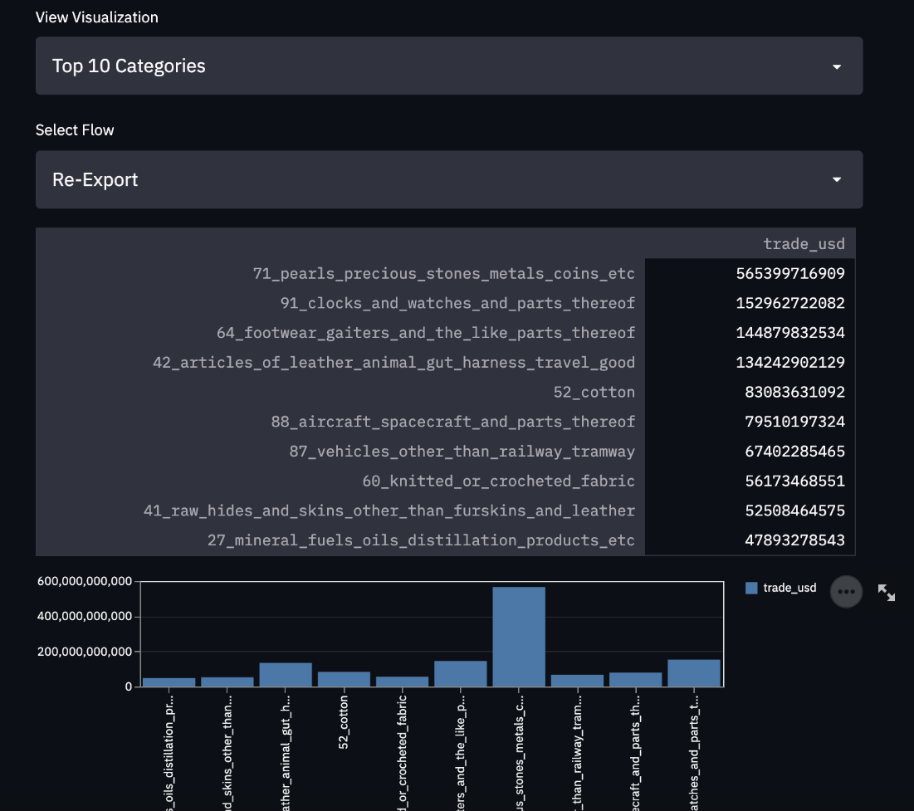
1. **Top 10 Categories for Imports**



### Data Validation for Visualizations

To ensure data consistency and accuracy among visualizations with the original dataset, the following test cases have been used:

**Case 1**



**The metrics from the visualization match the actual values from the source dataset.**

**Case 2:**





**The metrics from the visualization match the actual values from the source dataset.**

### Codebase for visualization

The code for the Python Streamlit Visualization is available in **viz/app.py**

## System Integration and User Acceptance Testing

### System integration tests

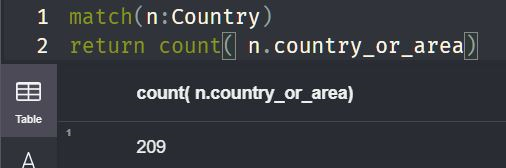
System integration tests were performed on the entire data pipeline and the front-end visualization.

|  |  |  |
| --- | --- | --- |
| Test Case | Description | Test Case Passed/Failed? |
| CSV File in the source directory | The CSV file should be picked up by the first stage in the pipeline and the pipeline should transition to the next stage. | PASS  CSV file is picked up by the pipeline and the pipeline transitions from **ReadData** to **DataCleaning** |
| Missing CSV file in the data pipeline | The csv file is removed from the source location, where the pipeline is expected to pick it up. | **PASS**  The pipeline transitions to Failed state since the CSV file is not found. |
| Wrong CSV file provided. | Wrong CSV file is provided. The pipeline should work as expected but data load to Neo4j should fail since the data in the file does not match the Cypher to load Neo4j. | **PASS**  The pipeline transitions to Failed state in the LoadNeo4j task. |
| Database is not started. | Neo4j database is not started. The pipeline is unable to connect to Neo4j to execute Cypher statements. | **PASS**  The pipeline transitions to Failed state in the Create Constraints task since the database is not active. |
| Database active. | Neo4j database is started and active. The pipeline should work as expected and load the data. | **PASS**  The pipeline cleans data, creates constraints, and loads the data to Neo4j. |
| Database is not started. | The front-end dashboards running on Streamlit should not display any data/visualizations. | **PASS**  The applications fail to load and does not show any visualizations/insights from the database. |

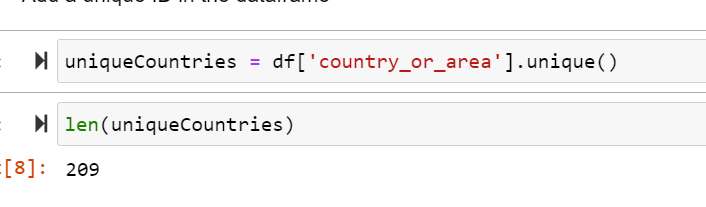
**USER ACCEPTANCE TESTING**

* This was carried out by ensuring the data in the database could be accessed by others to retrieve information accurately.
* When ensuring the data can be accessed and used, the database should be able to return query results on one table as well as on joins.
* The data returned should correspond accurately to the standards of data integrity and data consistency. Below are a few of the queries used to test the Commodity database:

1. Counting the unique country names

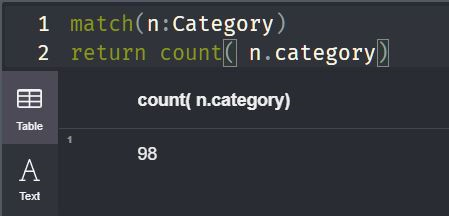


Unique Country names in the processed CSV file:

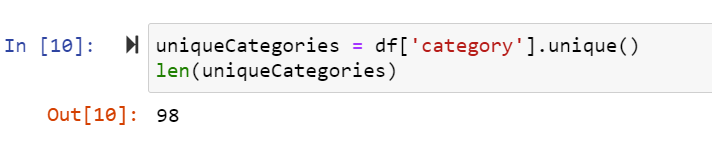


**Test Case: PASSED**

1. Counting the distinct categories

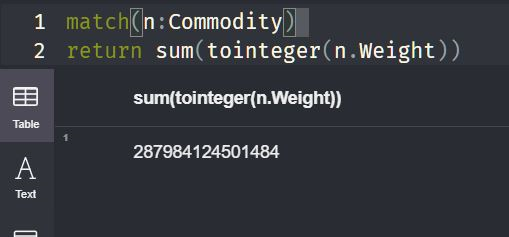


Unique Categories in the processed CSV file:

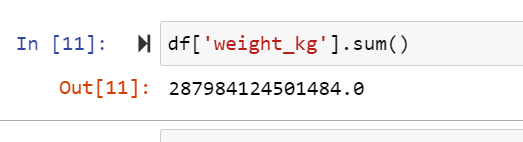


**Test Case: PASSED**

1. Adding sum of all the weights

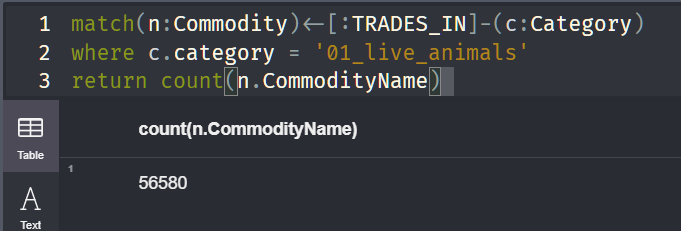


Sum of all weights on the processed CSV file:

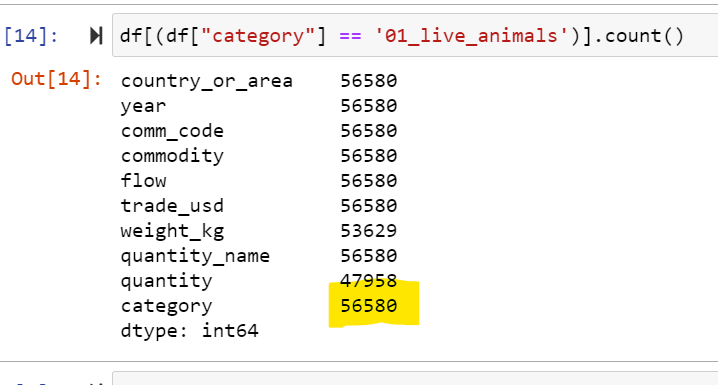


**Test Case: PASSED**

1. Querying number of commodities in a particular category



Counts of Categories on the processed CSV file:

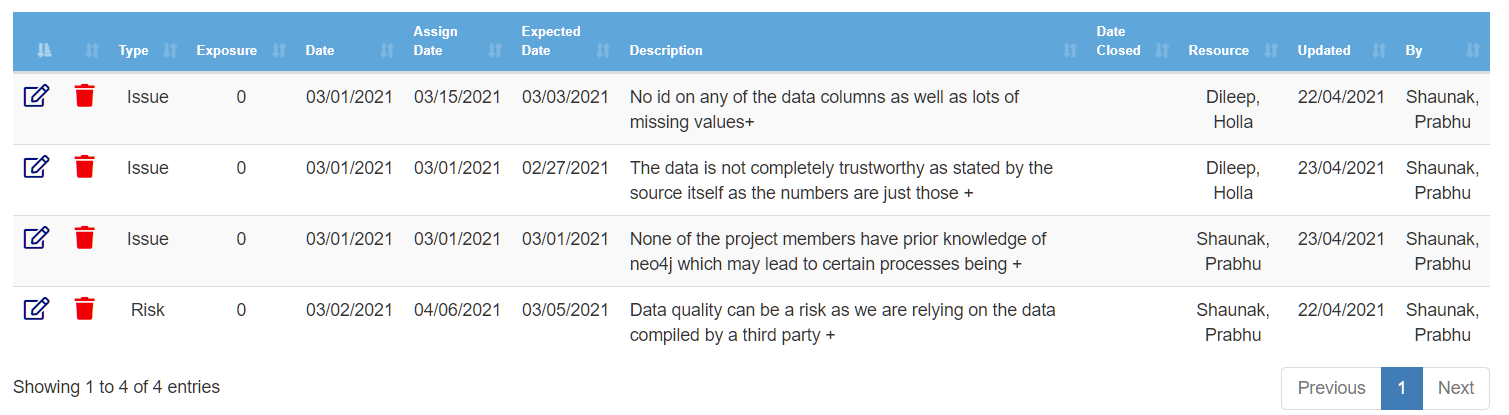


**Test Case: PASSED**

## Challenges Encountered

* No prior experience with Neo4j: Graph databases were a relatively new concept and there was a major learning curve to understand the differences between relational and graph databases. ​
* Method of querying is also different from the usual ANSI SQL syntax and was a challenge. ​
* The dataset provided had over 8 million records – reducing load times to Neo4j was one of the major challenges.​
* End-to-end integration using Python to clean, wrangle and load the database was a huge learning curve as well – since Apache Airflow was used to integrate the entire process. ​
* Manpower: Only team with 2 members. Hence, there was more work distributed between team members.

**RISKS AND ISSUES**



1. **Issue:** No id on any of the data columns as well as lots of missing values in the dataset which had to be handled in a manner that does not reduce the data and keeps data integrity. This was mitigated by changing null values and ‘NaN’ to ‘No Data’ to ensure data consistency and by creating a new column for ‘id’.
2. **Issue:** The data is not completely trustworthy as stated by the source itself as the numbers are just those which the country has shared with the UN. Cannot be mitigated but analysis must be done only on commodities that are more likely to have trustworthy data.
3. **Issue:** None of the project members have prior knowledge of neo4j which may lead to certain processes being done inefficiently. It may also lead to poorer usage of the database tool and neo4j's cypher for querying. This can be mitigated by better studying the intricacies of neo4j and its cypher language as well as studying better practices when loading and accessing data.
4. **Risk:** Data quality can be a risk as we are relying on the data compiled by a third party. Can be mitigated by profiling the UN data individually although the probability of this is low as the data has simply been scraped but there is chances of error.

## End User Instructions

### End user instructions for neo4j

* Download Neo4j for your OS (Operating System) from here: <https://neo4j.com/download/>
* Follow the on-screen prompts and install the application.
* Create a new database and use this database in the following steps to load data.

### End user instructions for Data Loading & Ingestion

**Python version:** Python 3.7+  
**Libraries:** py2neo, pandas, apache-airflow, pandas-profiling

The data can be cleaned and loaded in two ways:

* Use the Airflow data pipeline.
* Run Jupyter notebook: dataCleansing\_loadNeo4j.ipynb

**Run instructions:**

* The contents of the data ingestion pipeline are present in the **data-ingestion** directory.
* Download and install Python from here: <https://www.python.org/downloads/release/python-370/>
* Install the dependencies by using the requirements.txt file by running the following command: **pip install –r requirements.txt**
* Once all dependencies are installed, set up the Airflow instance. Initialize the instance by running the following commands:  
  **# Use your present working directory as# the airflow home**  
    
  **export AIRFLOW\_HOME=~(pwd)**  
    
  **# export Python Path to allow use# of custom modules by Airflowexport PYTHONPATH="${PYTHONPATH}:${AIRFLOW\_HOME}"**  
    
  **# initialize the database**  
    
  **airflow db init**  
    
  **airflow users create \**   
  **--username admin \**   
  **--firstname <YourName> \**   
  **--lastname <YourLastName> \**   
  **--role Admin \**   
  **--email** [**example@example.com**](mailto:example@example.com)
* Copy the contents of dags directory into the dags directory on your machine.
* Start the Airflow webserver by running **airflow webserver –D**
* Start the Airflow scheduler by running **airflow scheduler**
* Visit <http://127.0.0.1:8080/> to view the Airflow UI and trigger the pipleine: ***DataCleaningIngestion***

### End user instructions for Viewing the dashboard

**Python version:** Python 3.7+  
**Libraries:** streamlit, pandas

**Run instructions:**

* Install the required libraries by running **pip install streamlit** and **pip install pandas**
* The Python file is present in viz/app.py directory.
* Start the application by running **streamlit run app.py**
* This should start the streamlit application.
* View the dashboard by visiting <http://127.0.0.1:8501/> on your browser.

## 4.0 Velero Screenshots

Project Milestones/Tasks



Timesheet