Overview

This document will highlight the process of taking raw UN Comtrade data and transforming it into the visualizations included in the report.

Data Cleaning

The purpose of this section is to detail how the raw data was transformed into data frames to be more easily and efficiently used in the work environment.

*Note: The following process are specifically detailed for quarterly partitioned data instead of annual, but the methodology remains very similar between the two

Step 1: Locating and partitioning the data

- The raw data is stored at the following location in our repository: '/project/bi_dpi/data/UN_Comtrade/bulk/' in thousands of .arrow files
- The associated report was compiled with quarterly data; therefore we will partition our data by quarter (can also be done yearly, monthly, etc.) The following code allows us to get all of the .arrow files from the bulk folder that are associated with each quarter within a given year returned as a list with four entries (one for each quarter in the inputted year) (to be ran in the *console*): **(quarterly)partitionData.md**
- Now that we have a way to partition the data, we need to **filter** it, **condense** it, and **map** it

Step 2: Filtering the data

- There are 20+ total columns in the raw data, and we want to filter out the columns that aren't relevant to our analysis. For the disruption calculations, the columns pull are:
  - reporterCode = country reporting the trade
  - partnerCode = country being traded with
  - cmdCode = identifier for commodity being traded
  - flowCode = identifier for direction and type of trade
  - qtyUnitCode = identifier for units of the qty column
  - qty = quantity in units being traded
  - primaryValue = value of trade in dollars (USD $)
- Now we run another script in the console which will filter the raw data for the relevant columns and then write the data into quarterly files in a specified output directory: **(quarterly)filterData.md**
- This script takes in the list of partitioned quarterly raw data from the script we executed in the previous step and will write a file for each corresponding entry in that list. (For example, if the year was 2025, and only 1 quarter of raw data was available, only 1 file

would be written, but if it was 2024 with all data reported, then 4 files would be written from one execution of this script)

- Now that we've partitioned and filtered the data, we need to **condense** and **map** it
- *As a note steps 1-2 will need to be repeated for each year (in that order) being analyzed until all relevant quarters have files written for them

Step 3: Condensing the data

- Condensing the data is a vital step for the efficiency of our future visualization generations.
- What we mean by condensing the data is that currently, after partitioning and filtering, each row in a quarterly file represents **one** trade between two trading countries for a specified quantity in the time frame. Since we are concerned about the aggregate amount of trade between countries in the given time frame, we condense the data based on the mutable columns. The only mutable columns are "qty" and "primaryValue", and we hold the rest of the columns constant (the code takes the mode for qtyUnitCode but this is irrelevant for what we are describing here)
- Here is an illustration of what the condensing looks like at a very high level (using abridged data, we haven't done the mapping yet so you would actually see reporter, partner, and flow codes)
    - 2020Q1 – All trades for HS code 285000
        - US → China, $2000, 10 units
        - US → China, $4000, 20 units
        - US → China, $3000, 15 units
        - China → US, $1500, 10 units
        - China → US, $3000, 20 units
    - After condensing:
    - 2020Q1 (condensed) – All trades for HS code 285000
        - US → China, $9000, 45 units
        - China → US, $4500, 30 units
- As stated previously, this condensing of the data greatly improves efficiency down the line by reducing the file sizes
- The script we use to condense is at: **(quarterly)condenseData.md**
- This script, ran in the console, only needs to be called once and iteratively condense each quarterly file from the previous steps that exist in one specified directory.
- After execution, each quarter will have two files of data
    - One partitioned and filtered file
    - One partitioned, filtered, and condensed file
- Separating the data step by step is done so to troubleshoot each step independently instead of filtering, condensing, and mapping all in the same step (which could also be done)

- Now that the data is partitioned, filtered, and condensed, we need to **map** it.

Step 4: Mapping the data

- As stated in **Step 2**, the data doesn't explicitly state which country is the exporter and which is the importer – instead we have a reporter (reporting the trade), a partner (trading partner of the reporter) and a flow code (direction and type of trade). When we map the data, we will replace these 3 columns with 2, just "exporter" and "importer" which will make the data much more usable. The partners and reporters are also represented by numerical codes so we will also map them to ISO-3 codes again to improve understandability.
- The flowCode dictionary is as follows

flowCode	flowDescription	flowCategory
FM	Foreign Import	M
M	Import	M
MIP	Import of goods for inward processing	M
MOP	Import of goods after outward processing	M
RM	Re-import	M
DX	Domestic Export	X
RX	Re-export	X
X	Export	X
XIP	Export of goods after inward processing	X
XOP	Export of good for outward processing	X

- For our intents and purposes, the script we use to map the data groups the flow codes by their category (M or X – import/export) and essentially doesn't distinguish between types of imports and exports
- The script we use (execute once in the console after specifying range of years to pull quarters from) can be found at **(quarterly)mapData.md**
- After execution, each quarter will have **three** files of data
  - One partitioned and filtered file
  - One partitioned, filtered, and condensed file
  - One partitioned, filtered, condensed, and mapped file
- Again, separating the data step by step is done so to troubleshoot each step independently instead of filtering, condensing, and mapping all in the same step (which could also be done)
- Now that the data is partitioned, filtered, condensed, and map, it is ready to be loaded into data frames in the notebooks where we will make our data visualizations

Step 5: Loading the data

- Now that our data has fully been cleaned, all we need is something to convert it from .txt files into data frames that are much more easily usable in our workspace
- The code that is used for this is in the notebooks where the data visualizations will be made (ran once per workplace session (if kernel restarts need to run it again): **(quarterly)loadData.md**
- After this is ran, each quarter will have its own data frame

Max Disruption Data Visualization

The purpose of this section is to detail how to take the cleaned data and use it to generate the visualizations seen in the report. The two types of functions we will explain further are referred to as "single-k" and "all-k" where the single-k function calculates the set of size k exporters with the largest disruption to the importer of a commodity and the all-k function calculates all of the sets up to size k exporters with the largest disruptions to the importer of a commodity

*Note: for consistency purposes, both functions filter out rows of data where the importer/exporter is unknown (ISO-3 code "W00" in Comtrade), and where the exporter and importer are the same (a country trading with itself)

*Note: there is logic at the top of each function that maps countries to a color. Some countries colors have been hardcoded either because they are the focal point of the study, or because the random color assignment gave them colors too similar to other countries

MaxDisruption(single k)

- This function is fairly simple and it can be referenced in the repository at **(quarterly)maxDisruption(single k)**
- The function works by taking a set of commodity codes, and an importing country, and then brute forces (loops through every possibility) the set of countries that provide the largest disruption to the importer across a trade network from any of the given codes in the codes set. The function does this for both metrics of value and qty and displays the largest disrupting sets by time stamp over the entirety of the time period.

MaxDisruption(all k)

- This function can be referenced in the repository at **(quarterly)maxDisruption(all k)**
- This function works similarly to the first one, but requires a metric type as a parameter. Based on the metric type, this function calculates the largest disrupting sets at all sizes up to a specified k and displays them side by side (3 pars per time period)