BSD2343 DATA WAREHOUSING

GROUP NAME: Focus YEAH

PROJECT TITLE: Food Supply Chain Monitoring and Optimization System

NAME	MATRIC ID	SECTION
TAN CHEK CHENG	SD21031	02G
MITRAA A/P KOLANTHAI	SD21013	01G
NUR ADLINA ADILAH BINTI AZIMAN	SD21003	02G
NUR SYAZREEN BINTI ISMAIL	SD21050	02G
NORFATINI BINTI MUSTAFA	SD21057	02G

Group Focus YEAH's photo members

TABLE OF CONTENT

1.0 Background Project

The Sustainable Development Goal of Zero Hunger (SDG 2) is one of the 17 global goals by the United Nations in 2015 as part of the 2030 Agenda for Sustainable Development. SDG 2 aims to defeat hunger and develop sustainable agriculture worldwide. For our project, the title we have chosen is the Food Supply Chain Monitoring and Optimization System which is closely related to the Sustainable Development Goal of Zero Hunger (SDG 2). This project recognizes the urgent need for establishing an efficient and sustainable food supply chain in order to address issues with food security.

Our project's main goal is to create a complete system that analyses and improves the food supply chain. To do this, we make use of a variety of datasets that offer crucial data for efficient monitoring and optimization. The worldwide nation dataset, which acts as the project's fact table, is one important dataset used in this project. Among the attributes in this dataset are the nation (primary key), the region, the population, and the area (sq). We may learn important things about the expanding populations and how they affect food consumption in various nations by utilising this dataset. Understanding these dynamics enables us to predict food demand and make informed plans to ensure everyone has access to food more accurately.

Additionally, we make use of the global food consumption dataset, which includes information on the country, year, population, and particular food consumption values (such as those for beef(kg/capita), poultry(kg/capita), sheep (kg/capita), soy, maize (1000 tonnes), rice (1000 tonnes), soy(1000 tonnes) and wheat (1000 tonnes). We may examine consumption trends and preferences across nations using this dataset. We can make sure there is a sufficient range of healthy food alternatives available to fulfill the various nutritional demands of different locations by detecting these trends and monitoring changes in food preferences. We can address any shortfalls in food availability and make wise decisions to improve food security thanks to this study.

We also pay close attention to the global food supply port transportation dataset, which offers important details on the transportation component of the food supply chain. Data on nations, port names, ships in port, typical departure and arrival times, type, region local coverage, and area global coverage are all included in this dataset. We can find transportation system inefficiencies and boost the overall effectiveness of food distribution by analyzing this dataset. By streamlining the transport procedure, we lower waste and minimize damage, guaranteeing that food is delivered effectively and safely.

Additionally, the information on global weather conditions is crucial for agricultural planning for keeping an eye on the food supply chain. The features of this dataset contain information about the nation, city, latitude, longitude, average air temperature, average wind direction (in degrees), average wind speed (in km/h), and average sea level air pressure (in hPa). By taking into account these variables, we can determine how weather conditions affect crop development and guarantee a steady supply of food. Understanding weather patterns enables us to foresee problems and take the necessary action to lessen their impact on agricultural productivity, food supply and monitor the transportation condition at the same time.

The worldwide crop yield dataset, which contains data on parameters including country, food code, food item, year, unit, and yield value, is the last one we use. We can examine production levels and trends across multiple countries and categories of food thanks to this information. We can analyze agricultural performance, pinpoint low-productivity areas, and improve crop production methods by analyzing this data. These observations help to improve overall productivity, decrease food waste, and increase the effectiveness of the food supply chain.

In conclusion, the project Food Supply Chain Monitoring and Optimization System is created to solve the issues of food security. Our initiative intends to improve the efficiency, sustainability, and responsiveness of the whole food supply chain by utilizing data-driven insights and applying optimization methodologies. We aim to significantly affect the eradication of hunger and the establishment of a more sustainable and secure global food system by helping to accomplish the Sustainable Development Goals, particularly SDG 2 of Zero Hunger.

1.1 Problem need to be solved

The Food Supply Chain Monitoring and Optimization System project aims to address food security by developing an approach that leverages data-driven insights and optimization strategies to enhance the effectiveness, sustainability, and adaptability of the entire food supply chain. The project aims to make significant contributions towards achieving the Sustainable Development Goal of Zero Hunger (SDG 2).

The problem to be solved:

1. How can consumption patterns and preferences of different countries be analyzed to gain insights into demand trends and optimize the food supply chain accordingly?
2. What measures can be implemented to streamline the distribution process, reduce losses, and improve overall supply chain performance in order to enhance efficiency in food distribution systems?
3. How can weather data be utilized to understand the impact of weather conditions on crop transportation of food item worldwide to ensure a reliable and resilient food supply in the face of changing weather patterns and extreme events?

These problems highlight the need for data analysis and optimization strategies to improve various aspects of the food supply chain, including crop selection, distribution systems, weather impact, and understanding consumption patterns. By addressing these challenges, the project aims to contribute towards achieving SDG 2 and promoting sustainable and efficient food supply chains globally.

1.2 Objective

The objective of this project are:

1. To improve food supply chain operations based on consumption patterns.
2. To enhance efficiency in food distribution systems.
3. To utilize country weather data for reliable food supply.

1.3 Data Schemas

A database schema defines how data is organized within a relational database including table names, fields, data types, and the relationships between these entities. A schema describes the organization and storage of data in a database and defines the relationship between various tables. This dataset of this project contains five tables which are Global Country acts as facts table and the other datasets are Global Crop Yield, Global Food consumption, Global Weather Condition, and Global Food Supply Port Transportation. In this project, we used 4 library packages which are pandas to display the data schema with python code in Jupyter as shown in Figure 1.3.1.

```
import pandas as pd
import psycopg2 as ps
import pandas.io.sql as sqlio
import missingno as msno
```

Figure 1.3.1: Libraries were used to find data schema

The figure below shows the

```
1 df_facts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46 entries, 0 to 45
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Country       46 non-null     object
 1   Region        46 non-null     object
 2   Population    46 non-null     int64 
 3   Area (sq km)  46 non-null     int64 
dtypes: int64(2), object(2)
memory usage: 1.6+ KB
```

1.3.2: Data Schema Global Country Tables

The Figure 1.3.2 shows the Global Crop Yield Table from the Food Supply Monitoring and Optimization System. This table has 56717 rows and 6 columns in total such as Country, FoodCode, FoodItem, Year, Unit and YieldValue. It shows that there are 3 integer (int64) columns and 3 columns with object data type which are typically represented as string.

```
: df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56717 entries, 0 to 56716
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Country     56717 non-null  object
 1   FoodCode    56717 non-null  int64
 2   FoodItem    56717 non-null  object
 3   Year        56717 non-null  int64
 4   Unit        56717 non-null  object
 5   YieldValue  56717 non-null  int64
dtypes: int64(3), object(3)
memory usage: 2.6+ MB
```

Figure 1.3.3: Data Schema of Global Crop Yield Table.

Additional, Global Food Consumption Table is shown in Figure 1.3.4. Based on the result, this table consists of 10 columns in total which are Country, Year, Population, Beef (kg/capita), Poultry(kg/capita), Sheep(kg/capita), Maize(1000 tones), Rice (1000 tones), soy (1000 tonnes), and lastly is Wheat (1000 tonnes). There are 3 datatypes in this data schema. These data types are string, integer and float. From the figure below, we have 8 float attributes, and 1 for each integer and string attributes.

```
1 df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 957 entries, 0 to 956
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Country             957 non-null    object
 1   Year                957 non-null    int64
 2   Population          957 non-null    float64
 3   Beef(kg/capita)     957 non-null    float64
 4   Poultry(kg/capita)  957 non-null    float64
 5   Sheep(kg/capita)    957 non-null    float64
 6   Maize(1000 tonnes)  957 non-null    float64
 7   Rice(1000 tonnes)   957 non-null    float64
 8   soy (1000 tonnes)   957 non-null    float64
 9   Wheat(1000 tonnes)  957 non-null    float64
dtypes: float64(8), int64(1), object(1)
memory usage: 74.9+ KB
```

1.3.4: Data Schema Global Food Consumption Tables

Furthermore, the Global Weather Condition Table is another table for our project. This table contains 10 columns altogether. The names for each column are Date, Country, City, Latitude, Longitude, AverageAirTemperature(celcius), MinimumAirTemperature(celcius), MaximumAirTemperature(celcius), AverageWindDirection(celcius), AverageWindSpeed(km/h) and AverageSeaLevelAirPressure (hPa). This table consists of only 2 data types such as string and float. Based on the result from figure below, this table has 3 string attributes and 8 float attributes as shown in Figure 1.3.5.

```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 324647 entries, 0 to 324646
Data columns (total 11 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Date                              324647 non-null  object 
 1   Country                           324647 non-null  object 
 2   City                              324647 non-null  object 
 3   Latitude                          324647 non-null  float64
 4   Longitude                         324647 non-null  float64
 5   AverageAirTemperature(celcius)    314963 non-null  float64
 6   MinimumAirTemperature(celcius)    312284 non-null  float64
 7   MaximumAirTemperature(celcius)    312269 non-null  float64
 8   AverageWindDirection(degree)      283937 non-null  float64
 9   AverageWindSpeed(km/h)            302400 non-null  float64
 10  AverageSeaLevelAirPressure(hPa)   289416 non-null  float64
dtypes: float64(8), object(3)
memory usage: 27.2+ MB
```

1.3.5: Data Schema Global Weather Condition Tables

Lastly, the last table for our database is Food Supply Port Transportation in our project. This data frame contains a total of 8 columns. Among these, 3 columns have the data type 'int64', representing integer values, while the remaining 5 columns have the data type 'object', indicating string or categorical values.

```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Country             480 non-null    object
 1   Port Name           480 non-null    object
 2   Vessels in Port     480 non-null    int64 
 3   Average_Departures  480 non-null    int64 
 4   Average_Arrivals    480 non-null    int64 
 5   Type                480 non-null    object
 6   Area Local          480 non-null    object
 7   Area Global         480 non-null    object
dtypes: int64(3), object(5)
memory usage: 30.1+ KB
```

1.3.6: Data Schema Food Supply Port Transportation Tables

2.0 Architecture and ETL Pipeline

2.1 Pipeline structure

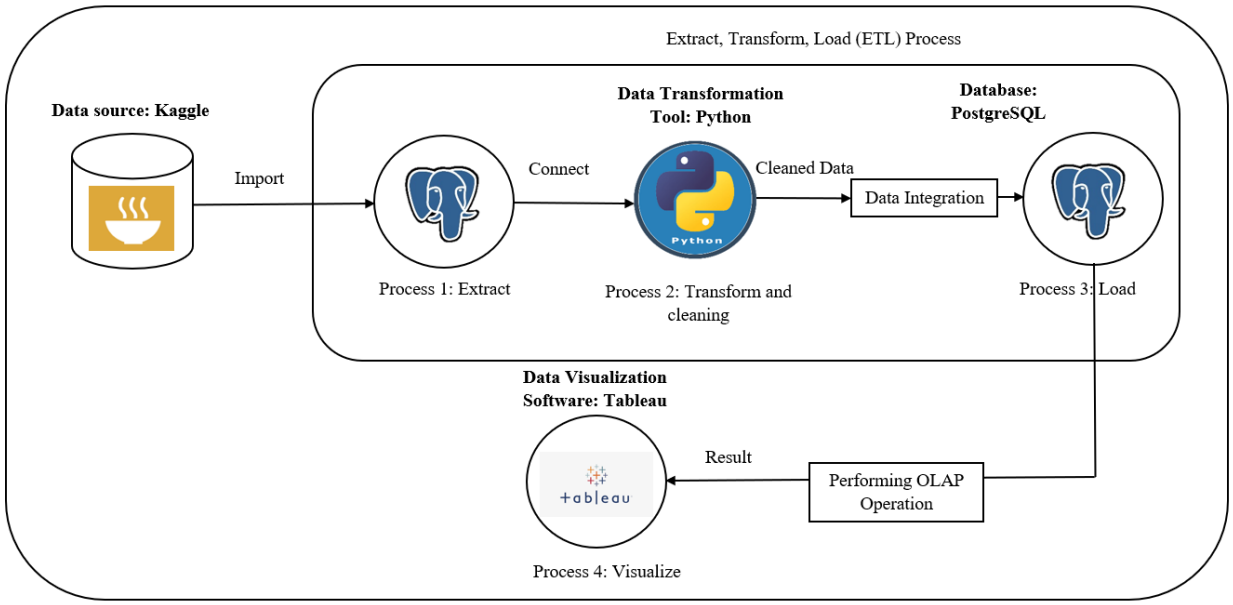


Figure 2.1

Based on Figure 2.1, for data collection and preparation, utilize Kaggle as a platform to access relevant datasets related to Global Country, Global Crop Yield, Global Food Consumption, Global Weather and Food Supply Port Transportation. Use Python to download and preprocess the datasets. Python provides libraries such as Pandas and NumPy that facilitate data manipulation, cleaning, and transformation.

For database design and management, install and set up PgAdmin to design the database schema based on the ERD. Create the five tables Global Country, Global Crop Yield, Global Food Consumption, Global Weather and Food Supply Port Transportation with the appropriate fields and relationships. Use SQL statements to define the tables, primary keys, foreign keys, and constraints.

For data loading and transformation, use Python to load the preprocessed data into the PostgreSQL database using the psycopg2 library. Convert the datasets into appropriate formats for example and insert them into the respective tables using SQL queries. Perform any necessary data transformations or manipulations within the database using SQL statements. This can include joining tables, aggregating data, and creating derived columns.

For data warehouse construction, once the data is loaded and transformed, the PostgreSQL database serves as the foundation for the data warehouse. Design and implement appropriate indexes, views, and materialized views in the database to optimize data retrieval and analysis.

For data visualization and analysis, connect Tableau to the PostgreSQL database. Tableau supports various data connectors, including PostgreSQL, and can easily access and visualize data from the data warehouse. Create interactive dashboards and visualizations in Tableau to explore the data across the five tables. We can leverage Tableau's drag-and-drop interface to create insightful visualizations, such as charts, graphs, and maps, to gain actionable insights and communicate them effectively.

2.2 Data Warehouse Architecture

The Architecture of our project is close to Kimball's apporach. Our group is starting the project by understanding and documenting the most critical problem from one of the 17 Sustainable Development Goals (SDGs), we extract the idea about the project needs, and project questions being asked. Zero Hunger is chosen as our main focus and the idea is about defeating hunger by developing a system to ensure a constantly sufficient food supply worldwide.

Then we document all data sources available from the online resource like kaggle throughout the project. After gathering all the required data ,we build ETL pipelines that extract from kaggle, transform by jupyter through python, and load data from the data sources into a denormalized data model in postgresql. The dimensional model is built as a star schema with central fact tables surrounded by dimension tables as shown in figure 3.1 of our database part.Lastly ,we proceed by enabling user-friendly data visualization and OLAP operation analysis.

All these aspects align with the Kimball approach, which focuses on delivering user-centric data warehouse solutions with bottom-up procedure. Kimball's approach focuses on a bottom-up procedure.

2.3 ETL Pipeline

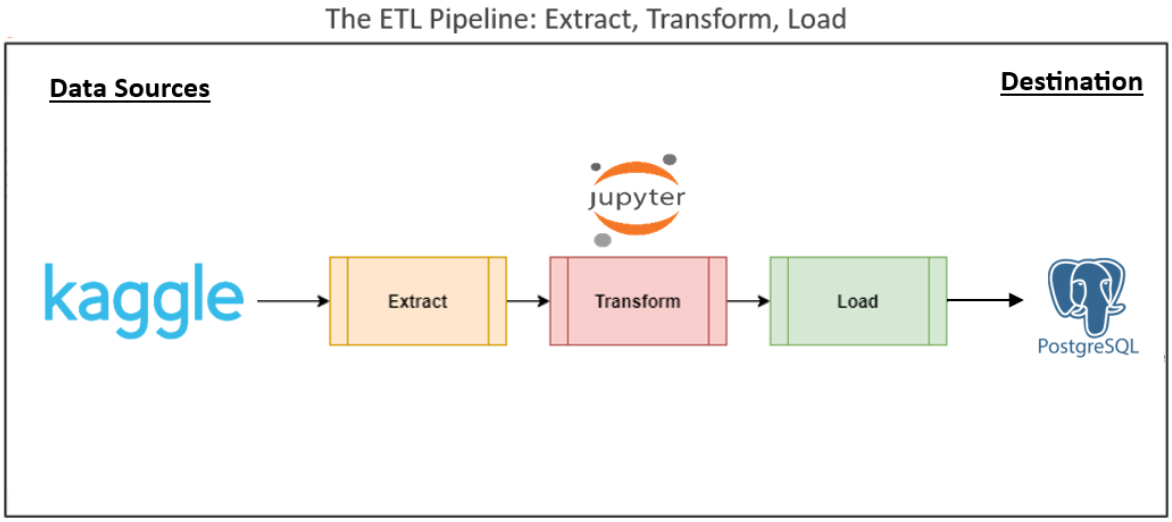


Figure 2.3

The dataset is downloaded from Kaggle. The dataset is composed of five tables which are Global Country, Global Crop Yield, Global Food Consumption, Global Weather and Food Supply Port Transportation. The table has been extracted and the process data cleaning and data integrating using Jupyter Notebook. After the cleaned data has been provided, the data will transform to a new csv file before loaded to PostgreSQL database to create visualization using Tableau.

2.3.1 Step to Extract and Transform (Python)

Refer To Appendices

2.3.2 Step to Load Data into Database (PostgreSQL)

Refer To Appendices

2.3.3 Step to Load OLAP Data into Visualization Tool (Tableau)

Refer To Appendices

3.0 Database

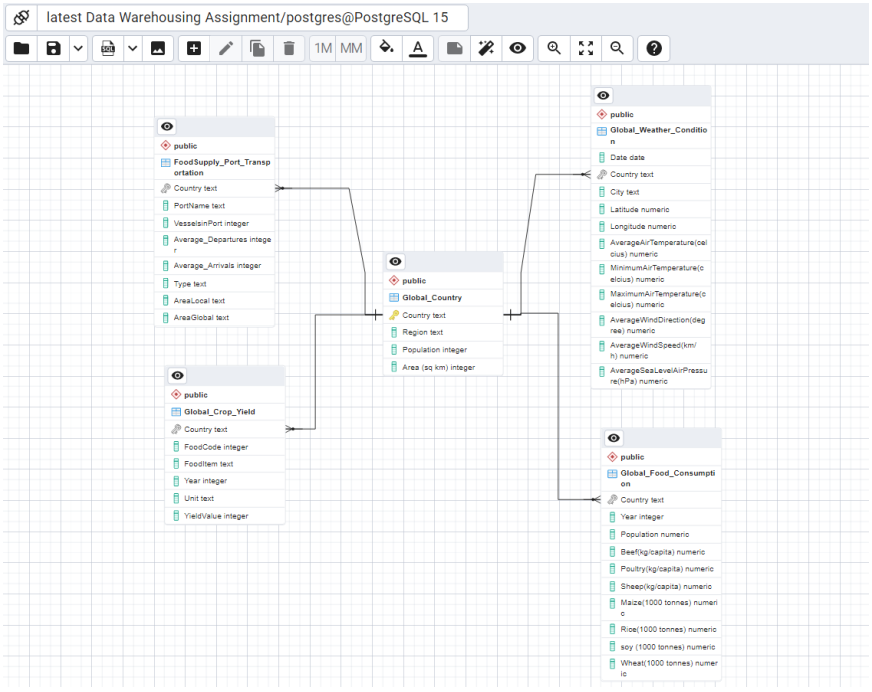


Figure 3.1 : Data schema from PostgreSQL

The figure 3.1 showed the relational model of our database by loading the 5 tables including Global_Country, Global_Crop_Yield, Global_Food_Consumption, Global_Weather_Condition and FoodSupply_Port_Transportation.The Entity Relationship Diagram(ERD) is clearly showing that our data warehouse schema is a star schema .Star schema is the fundamental schema among the data mart schema and it is simplest.

The data model is said to be star schema as its physical model resembles to the star shape having Global fact table at its center and the dimension tables like Global_Crop_Yield, Global_Food_Consumption, Global_Weather_Condition and FoodSupply_Port_Transportation at its peripheral representing the star's points. Based on Figure 3.1, the fact table in a star schema has one to many relationships with dimension tables. The relationship between the Global_Country(fact table) and Global_Crop_Yield(dimension table) is one-to- many relationship with foreign key country in the Global_Crop_Yield table.Same goes to other relationship between fact and dimension tables are all one-to-many relationship with the country as foreign key to the Global_Country table.

Overview of database relationship:

Fact Table:

1. Global_Country: Country (Primary Key), Region, Population, Area (sq km)

Dimension Tables:

1. FoodSupply_Port_Transportation: Country(Foreign Key), PortName, VesselsinPort, Average_Depatures, Average_Arrivals, Type, AreaLocal, AreaGlobal
2. Global_Crop_Yield: Country(Foreign Key), FoodCode, FoodItem, Year, Unit, YieldValue
3. Global_Food_Consumption: Country(Foreign Key), Year, Population, Beef(kg/capita), Poultry(kg/capita), Sheep(kg/capita), Maize(1000tonnes), Rice(1000tonnes), soy(1000tonnes), Wheat(1000tonnes)
4. Global_Weather_Condition: Country(Foreign Key), Date, Latitude, Longitude, AverageAirTemperature(celsius), MinimumAirTemperature(celsius), MaximumAirTemperature(celsius), AverageWIndDirection(degree), AverageWindSpeed (kn/h) ,AverageSeaLevelAirPressure(km/h)

4.0 Result and Data Analysis

4.1 OLAP Operation

4.1.1 Slicing

```
SELECT "Global_Country"."Country",
"Global_Crop_Yield"."FoodItem",ROUND(AVG("Global_Crop_Yield"."YieldValue"),2
) as "AverageYieldValue"
FROM "public"."Global_Country"
INNER JOIN "public"."Global_Crop_Yield"
ON public."Global_Country"."Country" =public."Global_Crop_Yield"."Country"
GROUP BY "Global_Country"."Country", "Global_Crop_Yield"."FoodItem"
HAVING  ROUND(AVG("Global_Crop_Yield"."YieldValue"),2) >10000
ORDER BY "Global_Country"."Country", "Global_Crop_Yield"."FoodItem";
```

	Country text	FoodItem text	AverageYieldValue numeric
1	Afghanistan	Maize	17194.61
2	Afghanistan	Potatoes	136327.96
3	Afghanistan	Rice, paddy	22194.14
4	Afghanistan	Wheat	12729.98
5	Algeria	Maize	20211.66
6	Algeria	Potatoes	132693.70
7	Algeria	Rice, paddy	22642.84
8	Algeria	Sorghum	37275.58
9	Angola	Cassava	62410.21
10	Angola	Potatoes	61504.39
11	Angola	Rice, paddy	10885.84
12	Angola	Sweet potatoes	78162.21
13	Angola	Wheat	10186.04
14	Argentina	Cassava	103245.41
15	Argentina	Maize	41597.32
16	Argentina	Potatoes	202888.89
17	Argentina	Rice, paddy	46650.59
18	Argentina	Sorghum	33220.29
19	Argentina	Soybeans	20215.63
20	Argentina	Sweet potatoes	127468.55
21	Argentina	Wheat	20165.96

Total rows: 241 of 241 Query complete 00:00:00.103

Figure 4.1: Output Slicing

The output of slicing of OLAP operation in shown in Figure 5a. This operation filters the unnecessary portions.Two tables involved in this operation that are Global_Country and Global_Crop_Yield. We conduct slicing operation by selecting some attributes through inner join of two tables and conditions using <HAVING>,<GROUP BY> and<ORDER BY> and aggregation on some attribute by AVG("Global_Crop_Yield"."YieldValue") with total 241 rows of result table. Through Slicing operation, we can filter the AverageYieldValue of different FoodItem from all the countries.For example, we can observe the Algeria country has the crop of Maize, Potatoes and Rice,paddy with their respective AverageYieldValue.

4.1.2 Pivot

```
SELECT * FROM crosstab(
'SELECT "Country", "Year", "Rice(1000 tonnes)"
FROM public."Global_Food_Consumption"
ORDER BY 1,2'
)AS Country("Country" text, "1990" numeric,"1991" numeric,"1992" numeric, "1993" numeric,
        "1994" numeric,"1995" numeric,"1996" numeric,"1997" numeric,"1998" numeric,"1999" numeric,
        "2000" numeric,"2001" numeric,"2002" numeric,"2003" numeric,"2004" numeric,"2005" numeric,
        "2006" numeric,"2007" numeric,"2008" numeric,"2009" numeric,"2010" numeric,"2011" numeric,
        "2013" numeric,"2014" numeric,"2015" numeric,"2016" numeric,"2017" numeric,"2018" numeric);
```

	Country text	1990 numeric	1991 numeric	1992 numeric	1993 numeric	1994 numeric	1995 numeric	1996 numeric	1997 numeric	1998 numeric	1999 numeric
1	Argentina	291.0	236.6	498.4	413.4	413.2	629.7	670.5	819.4	687.5	1127.4
2	Australia	740.0	957.0	858.0	1042.0	1016.0	966.0	1255.0	1324.0	1362.0	1084.0
3	Brazil	5338.6	6698.0	6769.0	6635.0	7050.4	7529.5	6724.8	6381.1	5670.2	7875.8
4	Canada	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	Chile	91.5	78.7	89.8	88.2	89.5	98.2	103.0	72.3	70.2	41.0
6	China	127231.1	123522.3	125698.5	120837.4	119986.3	126882.6	133646.2	137503.5	136117.7	135964.3
7	Colombia	1380.0	1008.6	1017.0	1068.6	1050.0	1185.9	1131.5	1244.4	1567.3	1485.9
8	Egypt	2185.5	2379.1	2697.3	2869.7	3161.6	3304.4	3380.3	3781.2	3087.1	4013.7
9	Ethiopia	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.5
10	India	74291.3	74679.8	72860.4	80300.3	81810.1	76980.3	81730.1	82540.1	86080.3	89679.8
11	Indonesia	30721.7	30389.2	32803.2	32763.8	30783.7	32828.4	33216.3	31107.5	30996.0	32147.3
12	Iran	1307.5	1484.9	1486.8	1437.0	1423.2	1449.0	1691.6	1480.5	1745.7	1479.2
13	Israel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	Japan	9512.1	8701.2	9579.1	7097.6	10854.8	9737.7	9371.7	9082.6	8117.8	8312.6
15	Kazakhstan	40.0	343.5	311.5	268.8	188.8	123.0	150.9	170.4	157.5	132.9
16	Malaysia	1215.1	1241.7	1298.2	1357.1	1379.0	1372.6	1438.6	1368.2	1256.8	1316.9
17	Mexico	200.0	190.0	200.0	189.5	246.6	242.2	260.1	309.8	302.4	231.8

Figure 4.2: Output Pivot

Based on Figure 4b, the output pivot indicates the purpose is to pivot the information from the "Global_Food_Consumption" table, specifically focusing on rice consumption data with a unit of 1000 tonnes. The resulting output will present the countries as rows and each year's rice consumption as separate columns with a total 33 rows after pivoting. By organizing the data in this manner, it becomes easier to compare and analyze the rice consumption trends across different countries and years. The query utilizes the crosstab function in PostgreSQL, which is a useful tool for transforming data from a row-based structure into a column-based format

4.1.3 Roll Up

```
SELECT "Country", "Type", sum("VesselsinPort")
FROM public."FoodSupply_Port_Transportation"
GROUP BY ROLLUP (1, 2)
ORDER BY 1, 2;
SELECT CASE WHEN "Country" IS NULL
THEN 'TOTAL' ELSE "Country" END,
CASE WHEN "Type" IS NULL
THEN 'TOTAL' ELSE "Type" END,
TotalPort_of_Vessels
FROM(SELECT "Country", "Type", sum("VesselsinPort") as TotalPort_of_Vessels
FROM public."FoodSupply_Port_Transportation"
GROUP BY ROLLUP (1, 2)
ORDER BY 1, 2) AS x;
```

	Country text	Type text	totalport_of_vessels bigint
1	Angola	Port	55
2	Angola	TOTAL	55
3	Argentina	Anchorage	69
4	Argentina	Port	199
5	Argentina	TOTAL	268
6	Australia	Marina	173
7	Australia	Port	856
8	Australia	TOTAL	1029
9	Bahrain	Port	244
10	Bahrain	TOTAL	244
11	Bangladesh	Anchorage	75
12	Bangladesh	Port	64
13	Bangladesh	TOTAL	139
14	Belgium	Anchorage	82
15	Belgium	Port	1138
16	Belgium	TOTAL	1220
17	Brazil	Anchorage	136
18	Brazil	Port	310
19	Brazil	TOTAL	446
20	Bulgaria	Port	66
21	Bulgaria	TOTAL	66

Total rows: 178 of 178 Query complete 00:00:00.168

Figure 4.3: Output Roll Up

The output for roll up operation is shown in Figure 4c. This operation is zooming out on the data cube . Roll -up is used to provide abstract level details to user . It performs further aggregation of data by reduction in dimension or by stepping up a concept hierarchy for adimension.One table (FoodSupply_Port_Transportation) only is involved in this operation. Abstract level like the overview of TotalPort_of_Vessels in port and anchorage of each country is displayed in table by using sum("VesselsinPort") with total 178 rows in result table.

4.1.4 Dicing

```
SELECT "Global_Country"."Country", "Global_Country"."Population",
ROUND(AVG("Global_Weather_Condition"."AverageWindSpeed(km/h)"),2)  AS
Average_Wind_Speed
FROM public."Global_Country"
JOIN public."Global_Weather_Condition" ON public."Global_Country"."Country" =
public."Global_Weather_Condition"."Country"
GROUP BY "Global_Country"."Country", "Global_Country"."Population"
HAVING ROUND(AVG("Global_Weather_Condition"."AverageWindSpeed(km/h)"),2)
> 5 OR "Global_Country"."Population">10000
ORDER BY "Global_Country"."Country" ASC;
```

	Country [PK] text	Population integer	average_wind_speed numeric
1	Algeria	44616624	10.41
2	Angola	33933610	10.93
3	Argentina	45376763	13.70
4	Austria	9152066	12.10
5	Belgium	11589623	12.47
6	Canada	37505437	12.12
7	China	1409517397	9.38
8	Croatia	4031249	7.35
9	Denmark	5824677	16.56
10	Egypt	100075480	14.60
11	Finland	5576003	15.17
12	Greece	10423054	7.79
13	Hungary	9769526	9.27
14	Iceland	356991	18.01
15	India	1366417754	6.05
16	Indonesia	276361783	6.76
17	Israel	9390084	10.79
18	Jamaica	2961167	18.66
19	Japan	125360000	10.14
20	Kazakhstan	18948816	9.16
21	Morocco	36910560	9.34

Total rows: 31 of 31 Query complete 00:00:00.504

Figure 4.4: Output Dicing

The output of dicing of OLAP operation is shown in Figure 4d. This operation emphasizes two or more dimensions from a cube given and suggests a new sub-cube.Two tables involved in this operation are Global_Country and Global_Weather_Condition. We conduct dicing operation by selecting some attributes through inner join of two tables and conditions using <HAVING>,<GROUP BY> and<ORDER BY> and aggregation on some attribute by AVG("Global_Weather_Condition"."AverageWindSpeed(km/h)") with total 31 rows in result table.The main difference compared to slicing operation is that our condition apply logical operator like <AND> and <OR> to filter more detail information from different dimensions.Through dicing operation, we can filter the Average_Wind_Speed of different countries that is greater than 5 and population of all countries that is greater than 10000 with requirement to our objectives.

4.2 Visualization

4.2.1 Food Item Produced by Each Country Based on Average Yield Value

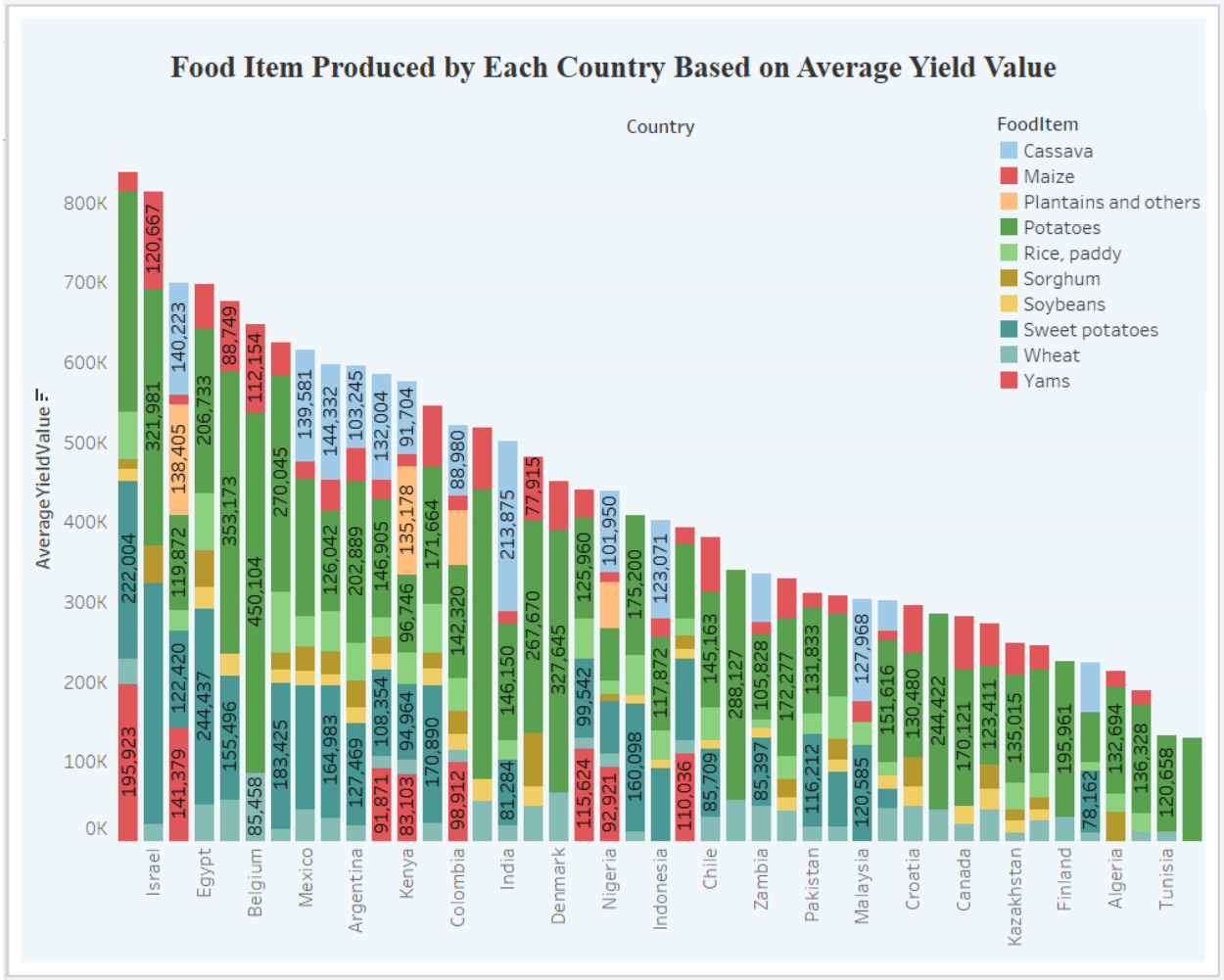


Figure 4.2.1: Food Item Produced by Each Country Based on Average Yield Value

The objective of improving food supply chain operations based on consumption patterns can be achieved by examining the rankings of countries in terms of average yield value for food items and the specific production quantities of different food items. Japan, with the highest average yield value above 800k, and Israel, ranking second around 800k, exemplify efficient food supply chain operations. Jamaica follows closely with an average yield value of 700k. Analyzing their strategies can inform improvements in other regions. Additionally, Belgium leads in potato production, generating 450,104, while Iceland focuses solely on potatoes but has the lowest average yield value. By studying these patterns and relationships, targeted measures can be implemented to optimize food supply chains and enhance overall efficiency.

4.2.2 Top 10 Highest Country Based on Average Wind Speed

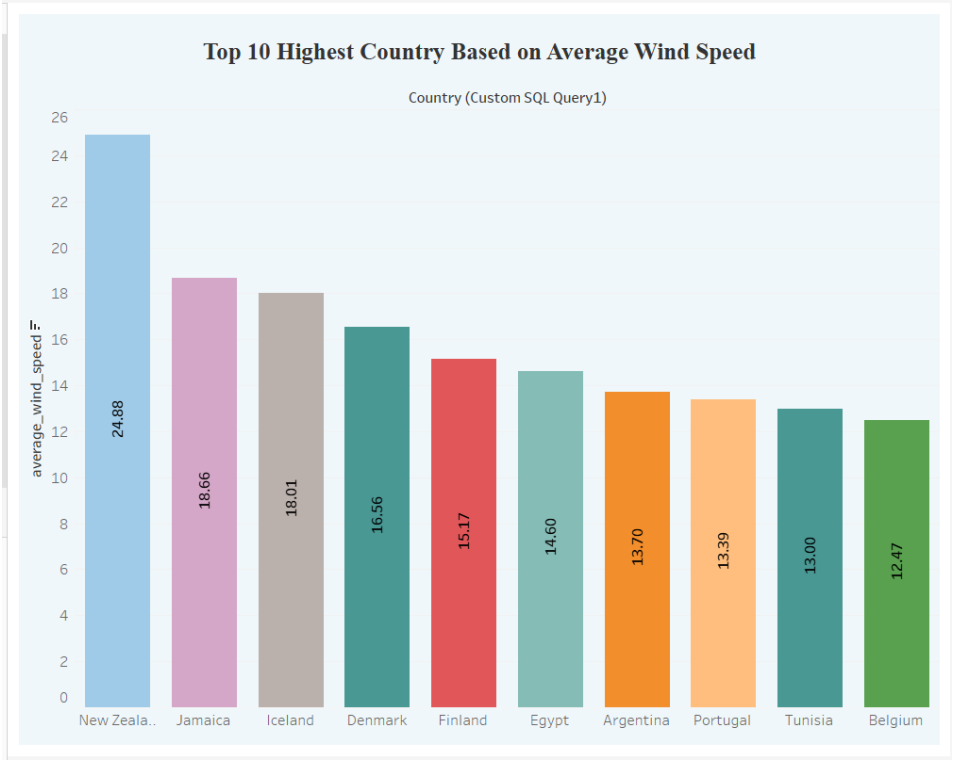


Figure 4.2.2: Top 10 Highest Country Based on Average Wind Speed

Based on Figure 4.2.2, vertical bar shows the visualization of top 10 highest country based on average wind speed. New Zealand has the highest average wind speed which is 24.88 km/h may experience more significant challenges in transporting crops due to potentially stronger winds. This information can help policymakers, transportation logistics professionals, and stakeholders in the agricultural industry to understand the specific risks and plan appropriate measures for crop transportation in windy conditions. Followed by Jamaica at 18.66 km/h, Iceland at 18.01 km/h, Denmark at 16.56 km/h, Finland at 15.17 km/h, Egypt at 14.60 km/h, Argentina at 13.70 km/h, Portugal at 13.39, Tunisia at 13.00 km/h and lastly in Belgium at 12.47 km/h may also face transportation challenges due to wind-related factors. By considering weather data and average wind speeds in these regions, stakeholders can take proactive steps to mitigate risks, such as reinforcing transportation infrastructure, optimizing logistics routes, or adjusting transportation schedules to minimize the impact of adverse weather conditions.

4.2.3 Top 10 Average Wind Speed Effect The Food Supply on Each Country Based On Their Population.

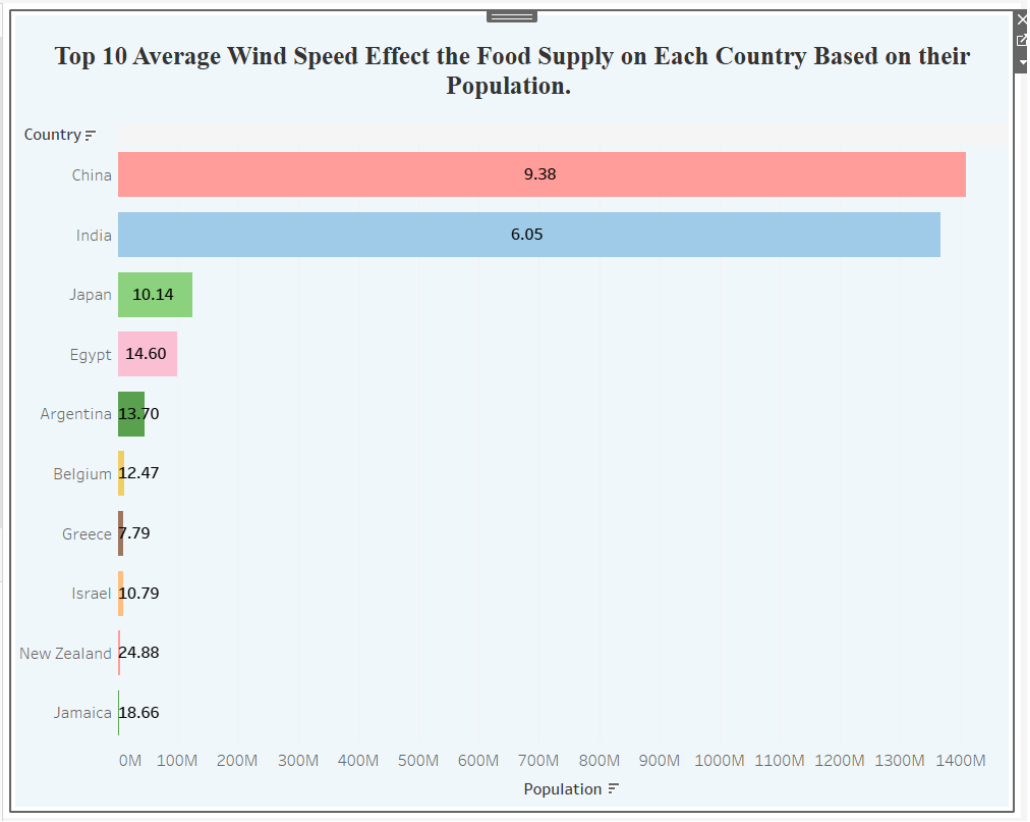


Figure 4.2.3: Top 10 Average Wind Speed Effect The Food Supply on Each Country Based On Their Population.

Wind Force	Description	Wind Speed		
		km/h	mph	knots
0	Calm	<1	<1	<1
1	Light Air	1-5	1-3	1-3
2	Light Breeze	6-11	4-7	4-6
3	Gentle Breeze	12-19	8-12	7-10
4	Moderate Breeze	20-28	13-18	11-16

Table 4.2.3: Safe wind Force Index.

The objective of this visualization is to utilize weather data, specifically average wind speeds, to ensure a reliable food supply. China has the highest population, exceeding 1.4 billion, followed by India with 1.3 billion people, and Japan with over 100 million. As population size increases, so does the demand for food, necessitating a larger food supply. However, the transportation of food items by sea is influenced by average wind speeds, which can affect the quality and condition of the goods. Fortunately, all three countries are strategically located in regions with safe average wind speeds for exporting and importing food items. These safe average wind speeds can be verified by referring to Table 4.2.3. By considering this vital weather data, stakeholders can make well-informed decisions to ensure a consistent and secure food supply for these densely populated nations.

4.2.4 Total Port of Vessels for Each Country

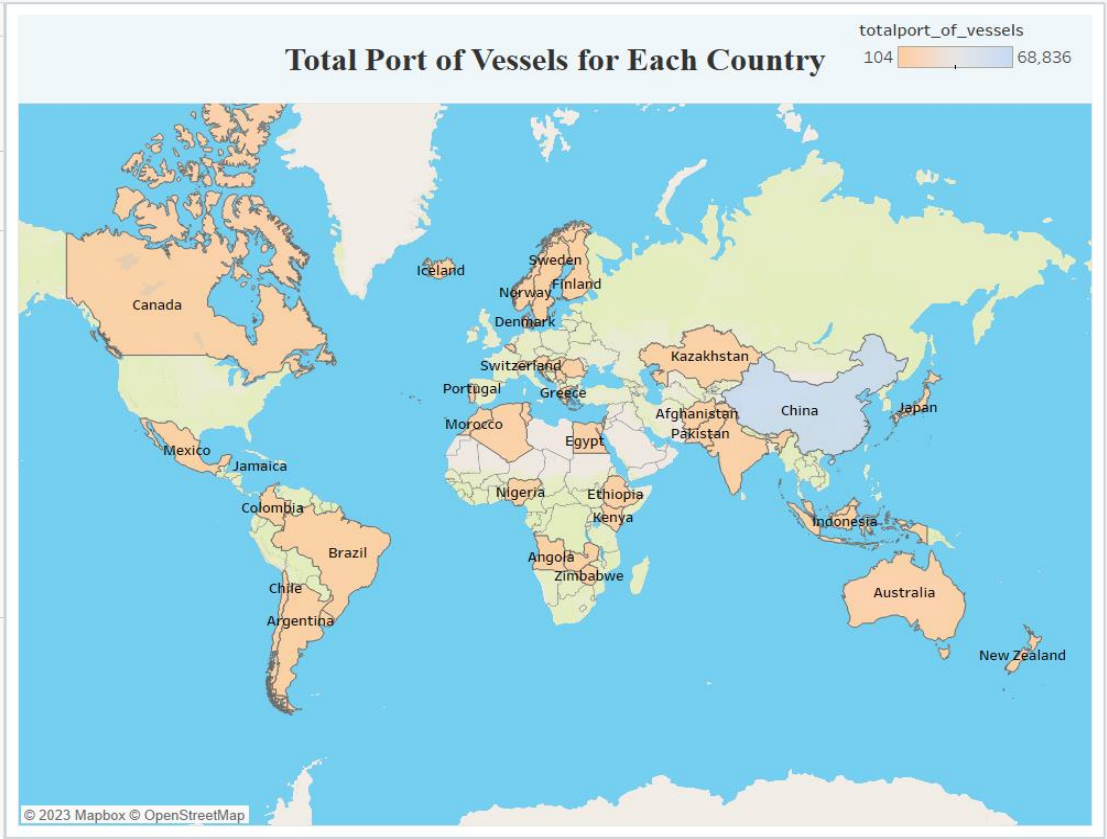


Figure 4.2.4: Total Port of Vessels for Each Country

Based on the figure 4.2.3. China has the highest total port of vessels for each country which is 68836. Followed by Belgium, Indonesia, and Australia at 2440, 2084 and 2058 respectively. The lowest total port of vessels goes to Chile with only 142 number of ports. This represents that China is the highest total port of vessels since China has the highest population which is 1,409,517,397 billion as shown in the figure 4.2.3 above. The significant presence of numerous vessels in Chinese ports highlights the necessity for efficient food distribution systems to meet the consumption patterns and requirements of its large population. By prioritizing enhancements in food supply chain operations aligned with consumption patterns and optimizing the efficiency of food distribution systems, we can ensure a dependable and efficient flow of food, effectively meeting the demands of China's population while also contributing to global food security.

5.0 Conclusion

In conclusion, the 5 different table from different data sources were loaded into a data warehouse and lastly loaded into a database. The ETL process is carried out so that cleaned datasets are obtained. The transforming process was done by using Python to clean the data then loading into postgresql to perform OLAP operation. Data visualization is practiced according to the result of OLAP operation.

Summary of data visualization result of OLAP operation is showing that analyzing consumption patterns, yield values, production quantities, and weather data provides valuable insights to enhance food supply chain operations. Countries like Japan, Israel, and Jamaica exhibit efficient supply chains, while Belgium excels in potato production. Considering average wind speeds helps identify transportation challenges, and proactive measures can be taken. China's large population necessitates efficient distribution systems for food security. By leveraging this information, stakeholders can optimize supply chains and meet the demands of densely populated regions.

Each group member encountered various challenges during the project. One major difficulty was finding suitable datasets that covered all the required questions and aligned with our system. We found it difficult to find the ideal datasets to cover those question especially when we tried to be looking for a dataset that consist of all the table that match our system. Another challenge was sharing the transformed data within the group's database. Each member had to update the database regularly, which consumed time and required careful attention to avoid any errors and catch up the progress. Furthermore, relating the project objectives to OLAP operations posed another challenge. We needed a comprehensive understanding of the database to determine if OLAP operations would provide efficient outputs to solve our project's problem. Lastly, we faced difficulties in establishing a connection between the database and visualization tools. We had to carefully evaluate whether PowerBI or Tableau would be the most suitable tool for visualization, considering that different group members were more proficient in different tools.

6.0 References

Goal 2: Zero Hunger - The Global Goals. (n.d.). The Global Goals. https://globalgoals.org/goals/2-zero-hunger/

How to Connect To PostgreSQL Database Server Using Python. (n.d.). How to Connect to PostgreSQL Database Server Using Python - CommandPrompt Inc. https:///education/how-to-connect-to-postgresql-database-server-using-python/

Pandas Merging 101. (2018, December 6). Stack Overflow. https://stackoverflow.com/questions/53645882/pandas-merging-101

Star Schema in Data Warehouse modeling - GeeksforGeeks. (2018, October 12). GeeksforGeeks. https://www.geeksforgeeks.org/star-schema-in-data-warehouse-modeling/

PostgreSQL Connector - Tableau. (n.d.). PostgreSQL Connector - Tableau - Tableau. https://help.tableau.com/current/pro/desktop/en-us/examples_postgresql.htm

Tableau and PostgreSQL: How To Make Them Work Together. (n.d.). Tableau and PostgreSQL: How to Make Them Work Together. https://blog.panoply.io/tableau-and-postgresql-two-ways-of-working-together

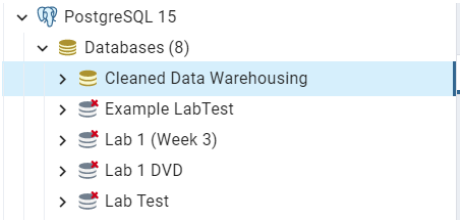
Pandas DataFrames - Reading from and writing to PostgreSQL table | Pythontic.com. (n.d.). Pandas DataFrames - Reading From and Writing to PostgreSQL Table | Pythontic.com. https://pythontic.com/pandas/serialization/postgresql

Sustainable Development Goal 2 - Wikipedia. (2022, June 15). Sustainable Development Goal 2 - Wikipedia. https://en.wikipedia.org/wiki/Sustainable_Development_Goal_2

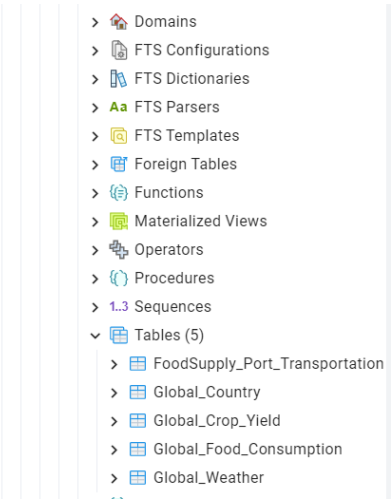
7.0 APPENDICES

Steps to load data into database (PostgreSQL)

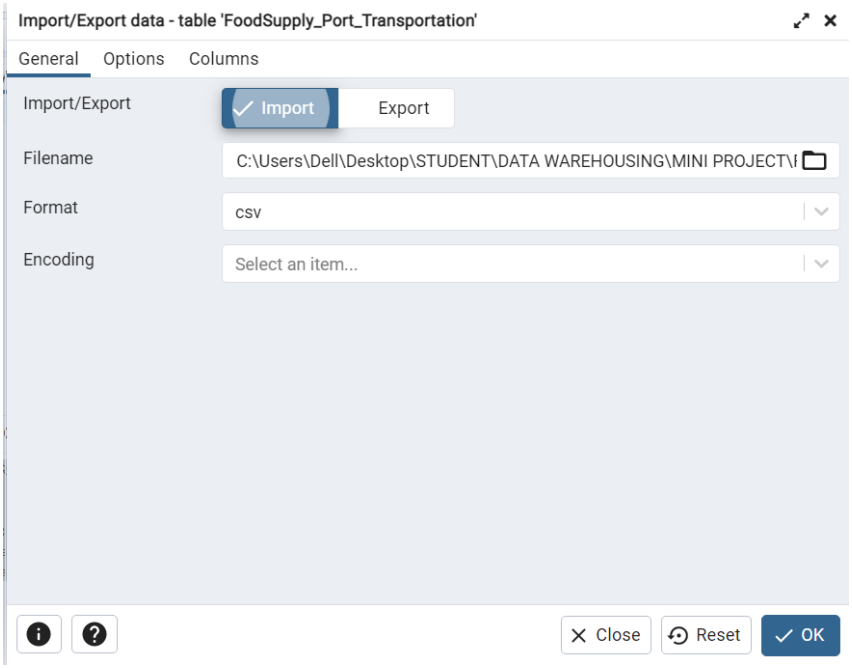
1. Created database ‘Clean Data Warehousing’ PostgreSQL



2. Created all of the five table ‘Global_Country’, ‘Global_Crop_Yield’, ‘Global_Food_Comsumption’, ‘Global_Weather’ and ‘FoodSupply_port_Transportation’



3. Import dataset ‘Global_Country’, ‘Global_Crop_Yield’, ‘Global_Food_Comsumption’, ‘Global_Weather’ and ‘FoodSupply_port_Transportation’ into table ‘Global_Country’, ‘Global_Crop_Yield’, ‘Global_Food_Comsumption’, ‘Global_Weather’ and ‘FoodSupply_port_Transportation’



Step to extract and transform

```
! pip install ipython-sql
! pip install sqlalchemy
! pip install psycopg2
! pip install python-sql
! pip install pandas-sql
! pip install sql-queries
! pip install missingno
```

Step to extract and transform (Python) These packages need to be installed before run the codes

1. pip install ipython-sql
2. pip install sqlalchemy
3. pip install psycopg2
4. pip install python-sql
5. pip install pandas-sql
6. pip install sql-queries
7. pip install missingno

```
import pandas as pd
import psycopg2 as ps
import pandas.io.sql as sqlio
import missingno as msno
```

These libraries need to be imported before run the codes:

```
%reload_ext sql
```

The code %reload_ext sql is a command to reload or activate the SQL extension.

```
from sqlalchemy import create_engine
```

The code from sqlalchemy import create_engine imports the create_engine function from the sqlalchemy library.

```
conn2=ps.connect(dbname="Uncleaned Data Warehousing Assignment ",
                 user="postgres",password= "1234",host="localhost",
                  port="5432")
```

dbname="Uncleaned Data Warehousing Assignment": Specifies the name of the database to connect to.

user="postgres": Specifies the username to use for authentication in PostgreSQL setup.

password="1234": Specifies the password for the user specified above.

host="localhost": Specifies the host or IP address where the PostgreSQL database is running.

port="5432": Specifies the port number on which the PostgreSQL database is listening.

```
sql="""SELECT * FROM pg_catalog.pg_tables"""
```

The query is a SELECT statement that retrieves all columns (*) from the pg_tables table in the pg_catalog schema of the PostgreSQL database.

The pg_catalog.pg_tables table contains information about tables in the database, such as table names, schema names, and other metadata.

By assigning the SQL query to the variable sql, the code prepares the query for execution or further processing.

Global_Crop_Yield table cleaning

```
sql="""SELECT *  FROM "Global_Crop_Yield"
"""
```

We used the SQL query to selects all columns and rows from the table named "Global_Crop_Yield". It retrieves the complete dataset stored in the table, allowing for further analysis or manipulation of the data.

```
df1=sqlio.read_sql_query(sql,conn2)
df1
```

Then, we use the DataFrame to reads the result of the SQL query into a pandas DataFrame called "df1" using the sqlio library. The DataFrame contains the data retrieved from the "Global_Crop_Yield" table in the database.

```
df1=df1.drop(['Area Code','Domain Code','Domain','Element Code','Element', 'Year Code'], axis=1)
df1
```

We drop the attributes which are Area Code, Domain Code, Domain, Element Code, Element and Year Code. And the table above shows the current of our dataset after drop the columns.

```
newattribute_names = {
    'Area': 'Country',
    'Item': 'FoodItem',
    'Item Code' :'FoodCode',
    'Value' : 'YieldValue'}

df1=df1.rename(columns=newattribute_names)
df1
```

Next, we need to rename old column to new columns names. For example, 'Area': 'Country' means that the old column name "Area" will be renamed to "Country" in the new DataFrame.

```
df1.info()
```

Then, we check the datatypes for dataframe above.The df_yield DataFrame contains 56717 entries (rows) and 6 columns. The columns include both numerical and object (string) data types, with three columns being integers and three columns being objects.

```
# Save the modified DataFrame back to a CSV file
df1.to_csv('cleaned_yield.csv', index=False)
```

Lastly, The code saves the DataFrame "df1" to a CSV file named "cleaned_yield.csv". The resulting CSV file will contain the cleaned and modified data from the DataFrame, and the "index=False" parameter ensures that the row indices are not included in the saved file.

Global_Food Consumption table cleaning

```
sql="""SELECT *  FROM "Global_Food_Consumption"
"""
```

The query is a SELECT statement that retrieves all columns (*) from the "Global_Food_Consumption" table.

```
df2=sqlio.read_sql_query(sql,conn2)
df2
```

The code df2 = sqlio.read_sql_query(sql, conn2) executes a SQL query using the read_sql_query function from the pandas.io.sql module.

The SQL query specified by the sql variable is executed using the conn2 connection object, which represents the connection to a PostgreSQL database.

The result of the query is returned as a Pandas DataFrame, which is assigned to the variable df2.

```
df2.isnull().sum()
```

The isnull() function checks each element of df2 to determine if it is null (missing).

The sum() function then adds up the number of null values for each column.

By executing this code, you get a count of how many missing values are present in each column of df2.

```
df2.info()
```

The info() method provides a summary of the DataFrame, including the number of rows, the number of columns, and the data types of each column.

It also provides additional details such as the memory usage of the DataFrame and the count of non-null values in each column.

By calling info() on the DataFrame df2, you can obtain a concise overview of the dataset's structure and the presence of missing values.

```
df2 = df2.drop(['pig (kg/capita)'], axis =1)
df2
```

The drop() method is used to remove columns from the DataFrame.

The column to be dropped is specified as a list within the function call.

The axis=1 parameter indicates that the column is being dropped (as opposed to dropping rows).

After executing the drop() method, the modified DataFrame without the specified column is assigned back to the variable df2.

```
# Rename the columns
new_column_names = {
    'country': 'Country',
    'year': 'Year',
    'population' : 'Population',
    'beef (kg/capita)' : 'Beef(kg/capita)',
    'poultry (kg/capita)' : 'Poultry(kg/capita)',
    'sheep (kg/capita)' : 'Sheep(kg/capita)',
    'maize (1000 tonnes)' : 'Maize(1000 tonnes)',
    'rice (1000 tonnes)' : 'Rice(1000 tonnes)',
    'soy (1000 tonnes)  ' : 'Soy(1000 tonnes)  ',
    'wheat (1000 tonnes)' : 'Wheat(1000 tonnes)',
}


df2 = df2.rename(columns=new_column_names)
df2
```

The df2.rename(columns=new_column_names) function is used to rename the columns of the DataFrame df2 based on the mapping provided in new_column_names.

After executing the rename() method, the modified DataFrame with renamed columns is assigned back to the variable df2.

The resulting DataFrame df2 now has column names that have been updated according to the mapping specified in new_column_names.

```
newRegion = {
    'ARG' : "Argentina",
    'AUS' : "Australia",
    'BRA' : "Brazil",
    'CAN' : "Canada",
    'CHL' : "Chile",
    'CHN' : "China",
    'COL' : "Colombia",
    'EGY' : "Egypt",
    'ETH' : "Ethiopia",
    'IDN' : "Indonesia",
    'IND' : "India",
    'IRN' : "Iran",
    'ISR' : "Israel",
    'JPN' : "Japan",
    'KAZ' : "Kazakhstan",
    'KOR' : "South Korea",
    'MEX' : "Mexico",
    'MYS' : "Malaysia",
    'NGA' : "Nigeria",
    'NZL' : "New Zealand",
    'PAK' : "Pakistan",
    'PER' : "Peru",
    'PHL' : "Philippines",
    'PRY' : "Paraguay",
    'RUS' : "Russia",
    'SAU' : "Saudi Arabia",
    'THA' : "Thailand",
    'TUR' : "Turkey",
    'UKR' : "Ukraine",
    'USA' : "United States",
    'VNM' : "Vietnam",
    'WLD' : "World",
    'ZAF' : "South Africa"
}

df2.Country=df2.Country.replace(newRegion)
df2
```

The country codes in the 'Country' column are replaced with their corresponding country names using the replace() method.

After executing this code, the 'Country' column of df2 will have the country names instead of the country codes.

The resulting DataFrame df2 will reflect the updated country names in the 'Country' column.

```
# Round the columns to one decimal place
df2['Population'] = df2['Population'].round(1)
df2['Beef(kg/capita)'] = df2['Beef(kg/capita)'].round(1)
df2['Poultry(kg/capita)'] = df2['Poultry(kg/capita)'].round(1)
df2['Sheep(kg/capita)'] = df2['Sheep(kg/capita)'].round(1)
df2['Maize(1000 tonnes)'] = df2['Maize(1000 tonnes)'].round(1)
df2['Rice(1000 tonnes)'] = df2['Rice(1000 tonnes)'].round(1)
df2['soy (1000 tonnes)'] = df2['soy (1000 tonnes)'].round(1)
df2['Wheat(1000 tonnes)'] = df2['Wheat(1000 tonnes)'].round(1)

df2
```

The .round(1) method is applied to the columns 'Population', 'Beef(kg/capita)', 'Poultry(kg/capita)', 'Sheep(kg/capita)', 'Maize(1000 tonnes)', 'Rice(1000 tonnes)', 'soy (1000 tonnes)', and 'Wheat(1000 tonnes)'.

By using the .round() method with a parameter of 1, the values in these columns are rounded to one decimal place.

After executing this code, the DataFrame df2 will contain the rounded values in the specified columns.

```
df2.to_csv('cleaned_Global_Consumption.csv', index=False)
```

The CSV file will be named "cleaned_Global_Consumption.csv".

The index=False parameter indicates that the index column should not be included in the exported CSV file.

By executing this code, the DataFrame df2 will be saved as a CSV file with the specified name and without the index column.

Global_COuntry table cleaning

In [6]:
```
sql="""SELECT *  FROM "Global_Country"
"""
```

We import the Global_Country from postgresql database

In [22]:
```
df3=sqlio.read_sql_query(sql,conn2)
df3
```

```
C:\Users\acer\anaconda3\lib\site-packages\pandas\io\sql.py:762: UserWarning: pandas only support SQLA
lchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2
objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

Out[22]:

ıtion	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climate	Birthrate	De
3997	647500	48,0	0,00	23,06	163,07	700.0	36,0	3,2	12,13	0,22	87,65	1	46,6	
1655	28748	124,6	1,26	-4,93	21,52	4500.0	86,5	71,2	21,09	4,42	74,49	3	15,11	
0091	2381740	13,8	0,04	-0,39	31	6000.0	70,0	78,1	3,22	0,25	96,53	1	17,14	
7794	199	290,4	58,29	-20,71	9,27	8000.0	97,0	259,5	10	15	75	2	22,46	
1201	468	152,1	0,00	6,6	4,05	19000.0	100,0	497,2	2,22	0	97,78	3	8,71	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
0492	5860	419,9	0,00	2,98	19,62	800.0	None	145,2	16,9	18,97	64,13	3	31,67	
3008	266000	1,0	0,42	None	None	NaN	None	None	0,02	0	99,98	1	None	
3188	527970	40,6	0,36	0	61,5	800.0	50,2	37,2	2,78	0,24	96,98	1	42,89	
2010	752614	15,3	0,00	0	88,29	800.0	80,6	8,2	7,08	0,03	92,9	2	41	
3805	390580	31,3	0,00	0	67,69	1900.0	90,7	26,8	8,32	0,34	91,34	2	28,01	

In [8]:
```
# Drop multiple columns
df3= df3.drop(['Pop. Density (per sq. mi.)', 'Coastline (coast/area ratio)', 'Net migration','Infant mo
                ,'Phones (per 1000)','Arable (%)','Crops (%)','Other (%)','Climate','Birthrate','Deat
              'Agriculture','Industry','Service','GDP ($ per capita)'],axis =1)

df3
```

Out[8]:

	Country	Region	Population	Area (sq. mi.)
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500
1	Albania	EASTERN EUROPE	3581655	28748
2	Algeria	NORTHERN AFRICA	32930091	2381740
3	American Samoa	OCEANIA	57794	199
4	Andorra	WESTERN EUROPE	71201	468
...	...	...	...	...
222	West Bank	NEAR EAST	2460492	5860
223	Western Sahara	NORTHERN AFRICA	273008	266000
224	Yemen	NEAR EAST	21456188	527970
225	Zambia	SUB-SAHARAN AFRICA	11502010	752614
226	Zimbabwe	SUB-SAHARAN AFRICA	12236805	390580

227 rows × 4 columns

```
Then all the columns except Country,Region,Population and Area(sq.mi.) are dropped.
```

In [9]:
```
newattribute_names = {
    'Area': 'Country',
    'Item': 'FoodItem',
    'Area (sq. mi.)' :'Area(sq km)'}

df3=df3.rename(columns=newattribute_names)
df3
```

Out[9]:

	Country	Region	Population	Area(sq km)
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500
1	Albania	EASTERN EUROPE	3581655	28748
2	Algeria	NORTHERN AFRICA	32930091	2381740
3	American Samoa	OCEANIA	57794	199
4	Andorra	WESTERN EUROPE	71201	468
...	...	...	...	...
222	West Bank	NEAR EAST	2460492	5860
223	Western Sahara	NORTHERN AFRICA	273008	266000
224	Yemen	NEAR EAST	21456188	527970
225	Zambia	SUB-SAHARAN AFRICA	11502010	752614
226	Zimbabwe	SUB-SAHARAN AFRICA	12236805	390580

227 rows × 4 columns

In [ ]:
```
Rename the attributes for better understanding of table
```

```
In [11]: df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Country      227 non-null    object
 1   Region       227 non-null    object
 2   Population   227 non-null    int64 
 3   Area(sq km)  227 non-null    int64 
dtypes: int64(2), object(2)
memory usage: 7.2+ KB
```

check the data info like data type of table

```
In [27]: df3.to_csv('new_country.csv', index=False)
```

```
In [ ]: Export the cleaned csv file
```

Global_Weather_Condition table cleaning

```
sql="""SELECT *  FROM "Global_weather_Condition"
"""
```

The query is a SELECT statement that retrieves all columns (*) from the "Global_weather_Condition" table.

```
df4=sqlio.read_sql_query(sql,conn2)
df4
```

The SQL query specified by the sql variable is executed using the conn2 connection object, representing the connection to a PostgreSQL database.

The result of the query is returned as a Pandas DataFrame, which is assigned to the variable df4.

The DataFrame df4 contains the data retrieved from the database based on the executed SQL query.

```
df4.info()
```

The info() method provides a summary of the DataFrame, including the number of rows, the number of columns, and the data types of each column.

It also provides additional details such as the memory usage of the DataFrame and the count of non-null values in each column.

By calling info() on the DataFrame df2, you can obtain a concise overview of the dataset's structure and the presence of missing values.

```
df4.columns = ['Date','Country', 'City', 'Latitude', 'Longitude', 'AverageAirTemperature(celcius)',
               'MinimumAirTemperature(celcius)', 'MaximumAirTemperature(celcius)', 'AverageWindDirection(degree)',
               'AverageWindSpeed(km/h)', 'AverageSeaLevelAirPressure(hPa)']
df4.columns
df4
```

After executing this code, the columns of df4 will be renamed according to the provided list.

The statement df4.columns returns the updated column names of df4.

By evaluating df4 after this code, you will see the DataFrame with the new column names applied.

```
# Round the columns to one decimal place
df4['Latitude'] = df4['Latitude'].round(1)
df4['Longitude'] = df4['Longitude'].round(1)
df4['AverageAirTemperature(celcius)'] = df4['AverageAirTemperature(celcius)'].round(1)
df4['MinimumAirTemperature(celcius)'] = df4['MinimumAirTemperature(celcius)'].round(1)
df4['MaximumAirTemperature(celcius)'] = df4['MaximumAirTemperature(celcius)'].round(1)
df4['AverageWindDirection(degree)'] = df4['AverageWindDirection(degree)'].round(1)
df4['AverageWindSpeed(km/h)'] = df4['AverageWindSpeed(km/h)'].round(1)
df4['AverageSeaLevelAirPressure(hPa)'] = df4['AverageSeaLevelAirPressure(hPa)'].round(1)

df4
```

By using the .round() method with a parameter of 1, the values in these columns are rounded to one decimal place.

After executing this code, the DataFrame df4 will contain the rounded values in the specified columns.

```
df4.to_csv('cleaned_weather.csv', index=False)
```

The CSV file will be named "cleaned_weather.csv".

The index=False parameter indicates that the index column should not be included in the exported CSV file.

By executing this code, the DataFrame df4 will be saved as a CSV file with the specified name and without the index column.

FoodSupply_Port_Transportation table cleaning

```
sql="""SELECT *  FROM "FoodSupply_Port_Transportation"
"""
```

We provided SQL code that selects all columns from the "FoodSupply_Port_Transportation" table in a database, retrieving all the data from that table.

```
df5=sqlio.read_sql_query(sql,conn2)
df5
```

We set the code to read the result of an SQL query stored in the variable 'sql' from a database connection 'conn2' and assigns it to a DataFrame called 'df5'.

```
df5 = df5.drop(['Index','UN Code','Expected Arrivals','Also known as'], axis=1)
df5
```

We removed the columns named 'Index', 'UN Code', 'Expected Arrivals', and 'Also known as' from the DataFrame 'df5', and the updated DataFrame is still referred to as 'df5'.

```
# Rename the columns
new_column_names = {
    'Departures(Last 24 Hours)': 'Average_Departures',
    'Arrivals(Last 24 Hours)': 'Average_Arrivals'}

df5 = df5.rename(columns=new_column_names)
df5
```

We renamed the columns in the DataFrame 'df5' to 'Average_Departures' and 'Average_Arrivals' from their original column names 'Departures(Last 24 Hours)' and 'Arrivals(Last 24 Hours)', respectively. The updated DataFrame is still referred to as 'df5'.

```
newattribute_names = {
    'Port Name': 'PortName',
    'Vessels in Port': 'VesselsinPort',
    'Area Local' :'AreaLocal',
    'Area Global' : 'AreaGlobal'}

df5=df5.rename(columns=newattribute_names)
df5
```

We renamed specific columns in the DataFrame 'df5' to new attribute names, such as 'Port Name' to 'PortName', 'Vessels in Port' to 'VesselsinPort', 'Area Local' to 'AreaLocal', and 'Area Global' to 'AreaGlobal'. The updated DataFrame with the renamed columns is assigned to a new DataFrame called 'd45', while 'df5' remains unchanged.

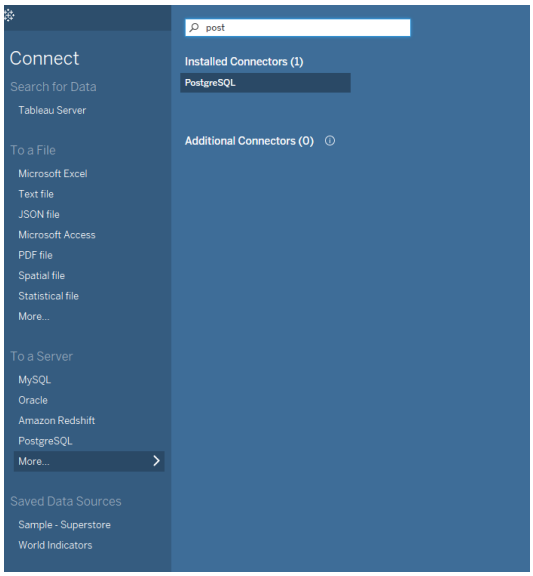
```
df5.info()
```

We displayed information about the DataFrame 'df5', including the data types of columns, the number of non-null values, and memory usage.

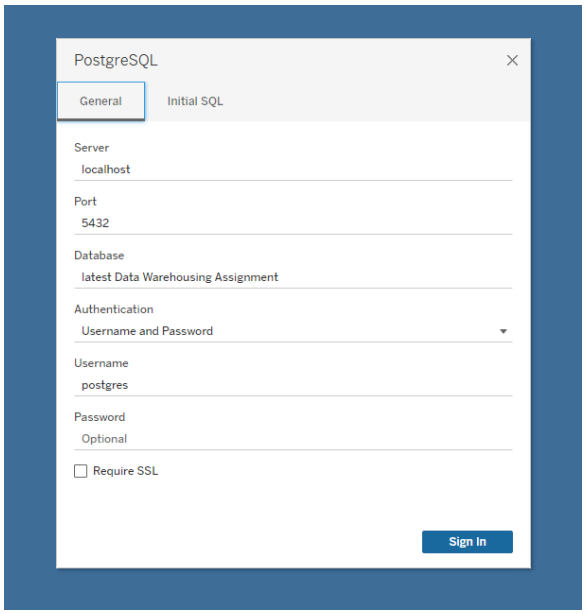
```
df5.to_csv('cleaned_shippingport.csv', index=False)
```

We saved the DataFrame 'df5' as a CSV file named 'cleaned_shippingport.csv' in the current directory, excluding the index column in the CSV file.

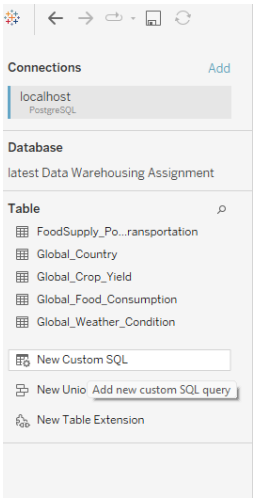
Step to Load OLAP Data into Visualisation Tool (Tableau)



Launch Tableau and navigate to the Connect option. From there, choose PostgreSQL as the desired connection.

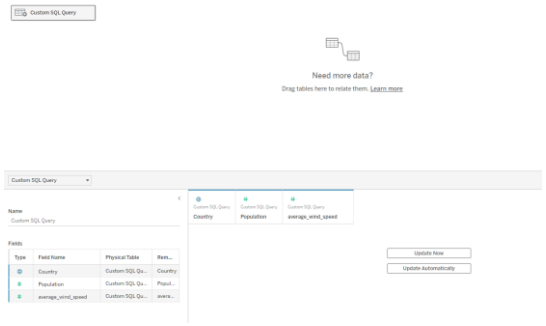


To establish a connection in Tableau, provide the server name and port where the database is hosted. Additionally, input the name of the specific database you wish to access. Next, choose your preferred method of signing in to the server, whether it's through Integrated Authentication or by providing a username and password. If the server requires a password and you're not operating in a Kerberos environment, you'll need to enter the appropriate credentials. Finally, click on the Sign In option to initiate the connection process.

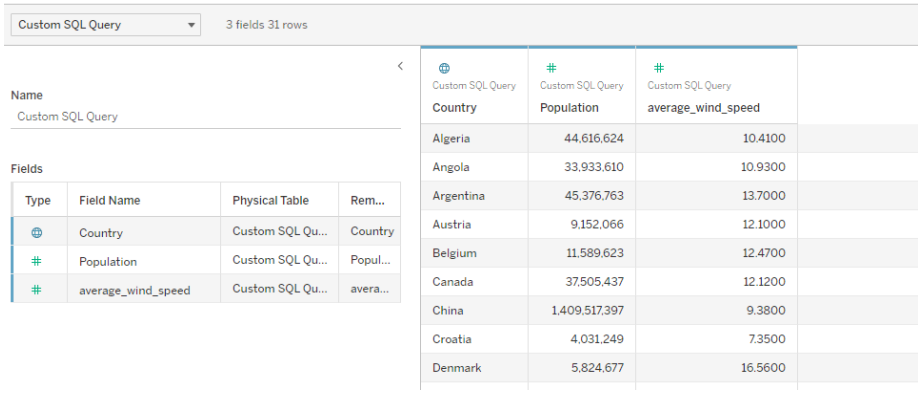


After this we can create the table directly based on sql coding of OLAP operation.Double click the New Custom SQL at the Table part of data source after connected to Tableau.

Then we are asked to copy and paste the SQL coding from postgresql and then click OK button. In this case, we copy the dicing operation from postgresql and paste in tableau custom sql.



Then one custom SQL query is created and we can click the button update now to check the result of SQL coding.



Custom SQL Query | 3 fields 31 rows

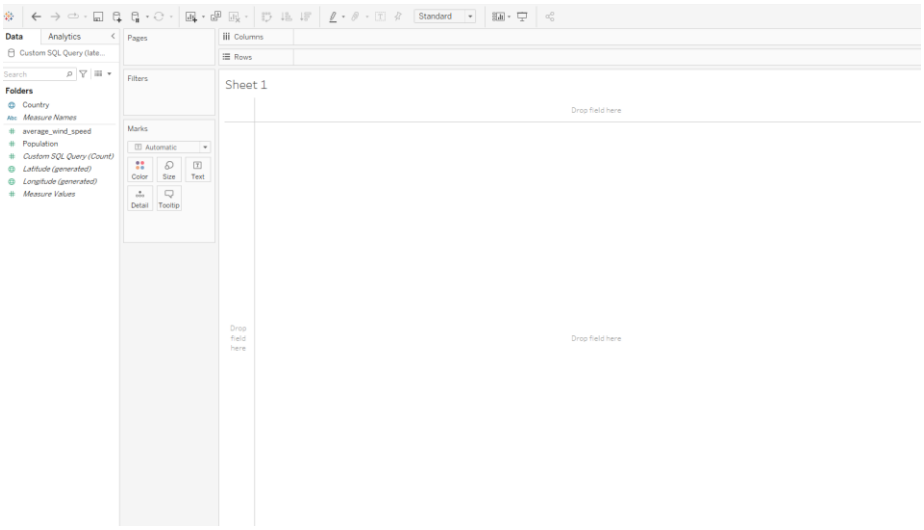
Name: Custom SQL Query

Fields

Type	Field Name	Physical Table	Rem...
Country	Country	Custom SQL Qu...	Country
Population	Population	Custom SQL Qu...	Popul...
average_wind_speed	average_wind_speed	Custom SQL Qu...	avera...

Custom SQL Query Country	Custom SQL Query Population	Custom SQL Query average_wind_speed
Algeria	44,616,624	10.4100
Angola	33,933,610	10.9300
Argentina	45,376,763	13.7000
Austria	9,152,066	12.1000
Belgium	11,589,623	12.4700
Canada	37,505,437	12.1200
China	1,409,517,397	9.3800
Croatia	4,031,249	7.3500
Denmark	5,824,677	16.5600

So we can see the table now, we can double check the result of table with the postgresql.The output should be exactly the same between them like number of rows and attribute.This means that we are successfully load the OLAP data from postgresql to tableau.



We can now start to visualise the graph based on the attribute of table result created just now.The process same goes to other OLAP operation as well.