An Analysis of Covid-19 Statistics

Group Members: Jared Silva, Lauren Matos, Grant Ritzwoller, Dylon Wilson, Calum Hewson

Overview

- **Data Engineering Track:** For this project, we implemented a ETL pipeline, specifically designed for the data engineering track. This pipeline allows us to extract, transform, and load complex Covid-19 data into a structured format.

- **Purpose:** The main objective was to use a range of coding techniques to transform raw Covid-19 data into a readable and insightful format, allowing us to draw meaningful conclusions.

- **Tools Used:** We used PostgreSQL for database storage, Jupyter Notebook for our coding environment, and Python as our primary programming language.

- **Data Source:** *Our World in Data* (OWID), a globally recognized platform that aggregates and publishes reliable, up-to-date COVID-19 statistics from sources such as the World Health Organization (WHO) and other reputable health organizations, ensuring a comprehensive view of the pandemic's impact across countries.

PostgreSQL Integration (Lauren)

```sql
-- SQL script for setting up the COVID-19 ETL Project database and tables
-- This script should be run in a PostgreSQL environment.
-- If the database already exists, skip the CREATE DATABASE line.

-- Step 1: Create the database (uncomment this line if the database is not yet created)
CREATE DATABASE covid19_data;

-- Step 2: Connect to the covid19_data database
\c covid19_data;

-- Step 3: Create the Covid_19_full_data table
CREATE TABLE IF NOT EXISTS Covid_full_data (
    date DATE,
    location VARCHAR,
    new_cases INT,
    new_deaths INT,
    total_cases INT,
    total_deaths INT,
    weekly_cases INT,
    weekly_deaths INT,
    biweekly_cases INT,
    biweekly_deaths INT,
    PRIMARY KEY (date, location)
);

-- Step 4: Create the OWID_Covid table
```

- **Creating the Database and Tables:** A PostgreSQL database was set up to store our Covid-19 data. This involved creating the key table: Covid_19_full_data
- **Schema Design:** The design for the schema focuses on essential fields like location, date, new_cases, and total_deaths. These fields were chosen for their relevance in analyzing the Covid-19 spread and trends over time.
- **Foreign Key Relationship:** To maintain data integrity and enable advanced querying, Lauren established a foreign key relationship between the tables. This linkage allows us to cross-reference data accurately across different sources.

ETL Workflow: Extract and Transform (Jared)

```
def extract_data():
    # Load CSV files into pandas DataFrames
    covid_full_data = pd.read_csv(covid_full_data_path)
    owid_data = pd.read_csv(owid_data_path)
    return covid_full_data, owid_data


def transform_data(covid_full_data, owid_data):
    # Transform data based on the available fields

    # Cases and deaths from Covid-19 full data
    cases_and_deaths_full_data = covid_full_data[
        ["date", "location", "new_cases", "new_deaths", "total_cases", "total_deaths"]
    ].copy()
    cases_and_deaths_full_data["date"] = pd.to_datetime(
        cases_and_deaths_full_data["date"]
    )
```

- **Extract Phase:** In the first step of the ETL pipeline, Jared extracted the raw Covid-19 data from CSV files. Using the extract_data() function, he loaded the Covid-19 full data.csv file into a Pandas DataFrame.

- **Transform Phase:** Transformation is a critical step to make the data usable for analysis. For the Covid-19 full data set, Jared cleaned and standardized the data by converting dates into a consistent format and selecting only essential fields like date, location, new_cases, new_deaths, total_cases, and total_deaths.

ETL Workflow: Load Phase (Jared)

```python
def load_data(transformed_data, covid_full_data, owid_data):
    # Load each DataFrame into its respective table in the database
    transformed_data["cases_and_deaths_full_data"].to_sql(
        "cases_and_deaths_full_data", engine, if_exists="replace", index=False
    )
    transformed_data["cases_and_deaths_owid"].to_sql(
        "cases_and_deaths_owid", engine, if_exists="replace", index=False
    )
    transformed_data["death_rates"].to_sql(
        "death_rates", engine, if_exists="replace", index=False
    )
    transformed_data["regional_deaths"].to_sql(
        "regional_deaths", engine, if_exists="replace", index=False
    )

    # Load raw CSV data directly into respective tables
    covid_full_data.to_sql("covid_full_data", engine, if_exists="replace", index=False)
    owid_data.to_sql("owid_data", engine, if_exists="replace", index=False)

    print("Data loaded successfully.")


def main():
    # ETL Workflow
    covid_full_data, owid_data = extract_data()
    transformed_data = transform_data(covid_full_data, owid_data)
    load_data(transformed_data, covid_full_data, owid_data)


if __name__ == "__main__":
    main()
```

- **Load Phase:** After transforming the data, Jared loaded it into our PostgreSQL database. Using SQLAlchemy's to_sql() method, he saved the cleaned and structured Covid-19 full data DataFrame into a table called cases_and_deaths_full_data within the database. This table serves as a core source for further analysis, enabling us to query and visualize trends effectively.

Dataframe Creation Using Pandas (Dylon)

```python
from sqlalchemy import create_engine
import pandas as pd

DATABASE_URL = 'postgresql+psycopg2://postgres:postgres@localhost:5432/covid19_data'
```

Python

```python
engine = create_engine(DATABASE_URL)
```

[3] Python

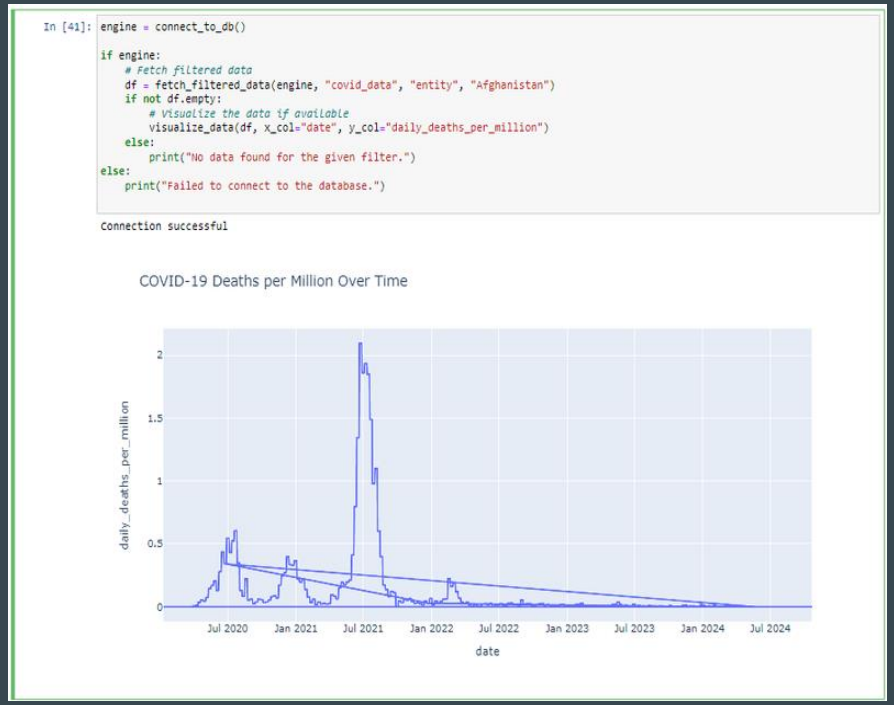
```python
# Query to select data from a table
query = "SELECT * FROM covid_full_data LIMIT 5;"

# Load data into a DataFrame
df = pd.read_sql(query, engine)

# Display the DataFrame
df
```

[6] Python

	date	location	new_cases	new_deaths	total_cases	total_deaths	weekly_cases	weekly_deaths	biweekly_cases	biweekly_deaths
0	2020-01-05	Afghanistan	0.0	0.0	0	0	None	None	None	None
1	2020-01-06	Afghanistan	0.0	0.0	0	0	None	None	None	None
2	2020-01-07	Afghanistan	0.0	0.0	0	0	None	None	None	None
3	2020-01-08	Afghanistan	0.0	0.0	0	0	None	None	None	None
4	2020-01-09	Afghanistan	0.0	0.0	0	0	None	None	None	None

- **Connecting to the Database:** Using SQLAlchemy, Dylon created a connection to the PostgreSQL database by utilizing the create_engine() function. This function allows seamless interaction between our Jupyter Notebook and the database.
- **Executing SQL Queries:** With the connection established, He wrote SQL queries to pull data from the Covid_full_data table and load it directly into a Pandas DataFrame.

User Driver Interaction (Grant)



- **Filtered Data Extraction:** In this step, Grant provides users with the ability to extract specific data from the database based on filters, allowing targeted analysis. Using SQL queries, we can retrieve data by specific conditions such as country or date range.

- **Visualization Insight:** Once the filtered data is retrieved, it's loaded into a Pandas DataFrame, making it easier to analyze and visualize. Here, Grant used Plotly to generate a line graph that depicts COVID-19 deaths per million people over time. This graph shows how deaths per million fluctuated during the pandemic, highlighting major peaks and trends.

Thank you for listening!