Global COVID Data Tracker

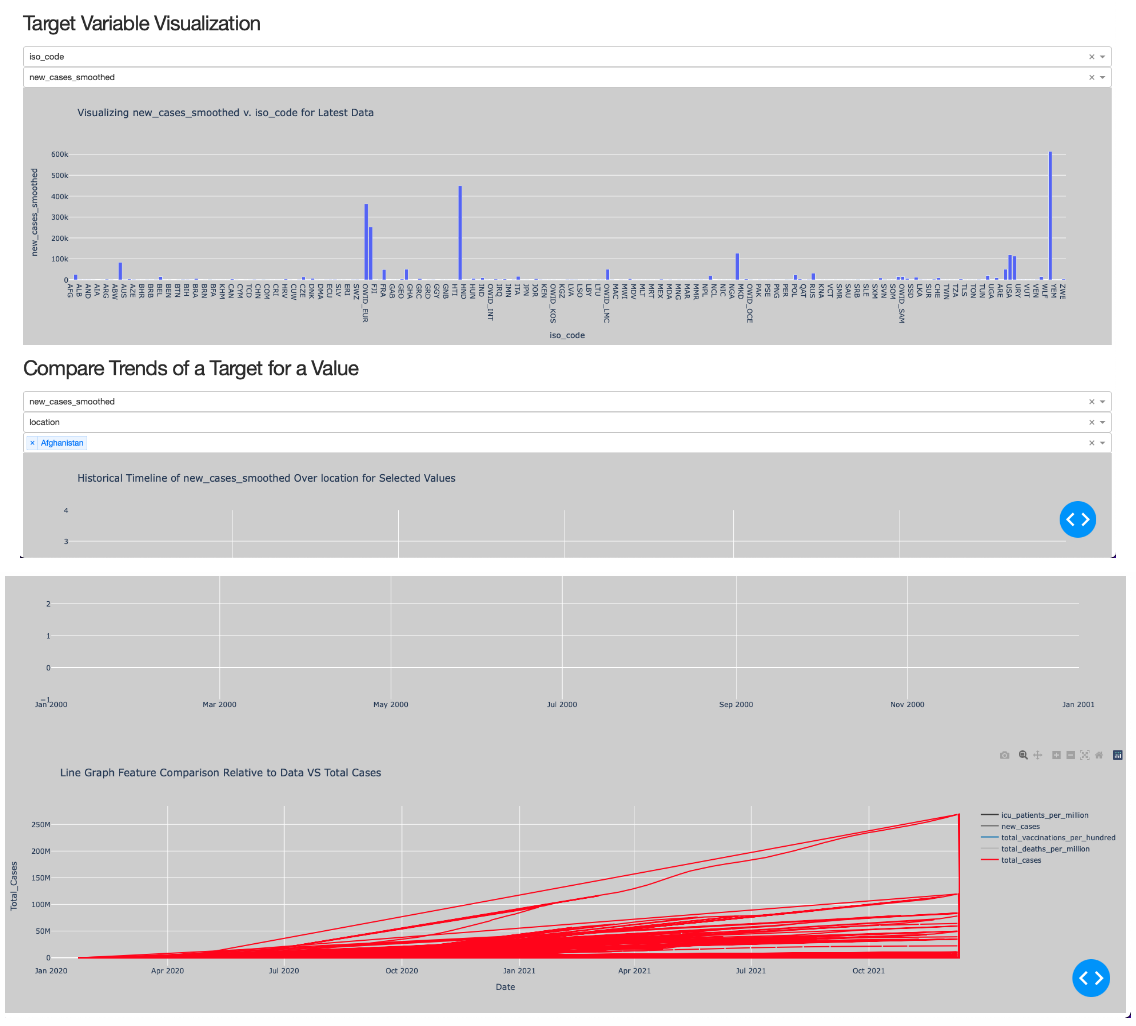
Project by: Samaye Lohan, Shubham Makharia, Yash Mehta, and Logan Heft

Setup for starting this project on GitPod:

1. Fork this project into your own GitHub account

2. Start a GitPod container on your forked repository by pasting the following URL into your web-browser:
    a. https://gitpod.io/#https://github.com/makhas/DATA1050

If you connect to port 1050 you should see a page similar to this:



Target Variable Visualization

iso_code

new_cases_smoothed

Compare Trends of a Target for a Value

new_cases_smoothed

location

× Afghanistan

Visualizing new_cases_smoothed v. iso_code for Latest Data
new_cases_smoothed
iso_code
Historical Timeline of new_cases_smoothed Over location for Selected Values
Jan 2000
Mar 2000
May 2000
Jul 2000
Sep 2000
Nov 2000
Jan 2001
Line Graph Feature Comparison Relative to Data VS Total Cases
Total_Cases
Date
icu_patients_per_million
new_cases
total_vaccinations_per_hundred
total_deaths_per_million
total_cases

Data Engineering Component

We are using the following CSV files as our base data:

1.https://covid.ourworldindata.org/data/owid-covid-data.csv
2.https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/latest/owid-covid-latest.csv

In order to create a system to store the data and update it at regular intervals, we set up a Cloud SQL instance (Postgres) on the Google Cloud Platform and created tables to represent the data from each file.

We setup a separate GCP VM instance to perform the following tasks:
1. Fetch the CSV files using Pandas
2. Infer the table structure from the CSV and create the necessary tables
3. Insert the data into the tables
4. For future updates, for historical data, check what has changed from existing data and only add those records. For the latest data table, replace the entire table with the most recent data.

Some of the challenges associated with this process were:
1. Designing tables to store the data. We have used a primitive approach of exactly mimicking the file structure since we didn't have the time to create 'good' DB design, however this can be easily fixed with some time investment. We used the dataframe dtypes to infer column data types and create the tables.
2. Inserting data into the DB. This was challenging since there are 67 fields in the file and writing an insert query was challenging and we couldn't get SQLAlchemy working with the GCP setup. We wrote a function to generate insert statements based on data.

3. Connecting the VM instance and the dash application to the GCP SQL DB. This was rather inconvenient, but we made the GCP SQL DB public, allowed all IPs to access it by setting an appropriate range, and stored the credentials to access with the web app and on the VM in an encrypted fashion.

Launching The App

This webapp runs through GCP. In order to run this from a local workspace or in GitPod, contact one of the contributors to this repo for Key_CREDENTIALS to access the database instance and run the dashboard.

Once key_CREDENTIALS has been secured, run the following command to launch the webapp.

export KEY="key_CREDENTIALS" && pip install -r requirements.txt && python app.py

Project Findings and Analysis

While designing the components of the dashapp, we approached the problem from the ground up. We thought about the kind of relationships we were interested in visualizing, and decided to focus on analyzing relationships between any two variables, and relationships between time and the values of a particular variable. The obstacles we encountered mostly came from difficulty translating the idea of a visualization to the component/callback framework used in dash apps. We knew to use callbacks to update a visualization, and that we wanted to create interactive plots that users could zoom in on. Specifically, we created three interactive visualizations that ranged from target variable visualization to feature comparison relative to a specific variable. The first visualization allows the user to select a feature for the x-axis and a feature for the y-axis which ultimately displays a bar graph that they can interact with. The visualization was restricted to certain features which added an additional layer of complexity because not every feature can portray a relationship with another feature. The second visualization displays a solid-line graph that allows the user to compare the trend of any two limited feature values with respect to a location attribute. For example, the user can select “new_cases_smoothed” and “iso” for the location “Canada” and “United States” and visualize that relationship. The last visualization represented a more statically dynamic visualization where the user was not given an option to select features, which is done by initiating a callback sequence in the backend, but instead we pre-selected certain features that could be stacked into a line graph. Furthermore, users can select and deselect certain features to get the appropriate aggregation of results.

During our exploratory data analysis phase of the project, many of the charts, tables, and queries served as the inspiration for what kind of interactive figures we wanted to make. Plotly was great for making interactivity easy, but making the plots update required a bit of debugging. Most of these obstacles seemed like natural growing pains that come with learning a new type of functionality like dashboarding. One problem we uncovered while exploring our data was that depending on the COVID data infrastructure in place in each country, the data that is being reported daily by each country is far different with many undeveloped countries reporting very little COVID data.

Citations

Mathieu, E., Ritchie, H., Ortiz-Ospina, E. *et al.* A global database of COVID-19 vaccinations. *Nat Hum Behav* (2021). https://doi.org/10.1038/s41562-021-01122-8

Hasell, J., Mathieu, E., Beltekian, D. *et al.* A cross-country database of COVID-19 testing. *Sci Data* 7, 345 (2020). https://doi.org/10.1038/s41597-020-00688-8