CORONASAFE

Current Version: 2.0.1 | Developed in Python

CoronaSafe is a python-based application that provides easy access to a COVID contraction risk rating for any global address. It provides easy access to a COVID contraction risk rating for any global address given by the user. CoronaSafe does this by analyzing live foot traffic data and calculating urban density (with a time weight), giving it the potential to work for any viruses that spread through close proximity and respiratory fluids.

Additional Feature: CoronaSafe creates interactive heat/choropleth maps for live US and State COVID-19 case data taken from the live New York Times .csv file.

Coming Soon: Server hosting backend → Website, Desktop/iOS/Android App Versions

Known Issues: State COVID case maps currently not working

FILE STRUCTURE BREAKDOWN

1. Final Product/Frontend:
    - `coronasafe_v2` → `coronasafe_v2_ui.py`
        - Contains frontend for CoronaSafe.
        - Interacts with `coronasafe_v2_backend.py`
2. Coronasafe Backend:
    - `coronasafe_v2` → `coronasafe_v2_backend.py`
        - Contains places search function, master risk calculation algorithm, and COVID case maps constructor caller.
        - Calls:
            a. `local_risk_calculator.py`
                - Contains local risk calculation algorithm.
            b. `surrounding_risk_calculator.py`
                - Contains surrounding risk calculation algorithm (factoring in urban density and a time of day weight).
            c. `heat_maps.py`
                - Contains US and State heat map constructor functions.

APIs USED

`Google Geocoding API`

`Google Places API`

Prerequisite: `Google API Key`

NON BUILT-IN LIBRARIES USED

`requests`, `livepopulartimes`, `plotly`, `pandas`, `ssl`, `argparse`, `python-dotenv`, `pathlib`

DOCUMENTATION

Main Functions

1. COVID-19 risk calculator based on a search query (can be something like "starbucks near me" or an address)
    - Parameters: search query
    - Functionality Breakdown:
        - Takes a search query in input field from user and runs it through a novel COVID-19 contraction risk calculation algorithm
        - Search Places: Takes in the search query, passes through the Google Maps Places API and outputs results to user as a dropdown
            - User then selects one of the addresses
        - COVID-19 Contraction Risk Calculation Algorithm: Generates risk level bar graph (through matplotlib) for the user chosen address

            a. Local Risk Calculation Algorithm:

            - Uses livepopulartimes library to get current live popular times (works best for named places such as a mall)
                - If the live popular times isn't available, it cross references with the average popularity at the current day and time
            - Outputs a local risk rating between 0 - 100

b. Surrounding Risk Calculation Algorithm:

- Uses Google Geocoding API to transform address into coordinates that get passed through the Google Places API to output named places in a 0.5 mile radius around the given address
- Generates a list of "popular" locations within the search radius such as airports and mall
- Factors in the number of "popular" locations within the search radius and time-weights based on the current time of day
- Outputs a surrounding risk rating between 0 - 100

c. Master Risk Calculation Algorithm:

- Combines both inputs (if available) to give a cumulative COVID-19 contraction risk rating between 0 - 100
  - Math: `cumulative_risk_rating = (local_risk_rating * 0.8) + (surrounding_risk_rating * 0.2)`

2. Real time COVID-19 maps that update every 24 hours

- Functionality Breakdown:
  - Uses pandas to read the following csv files:
    - `https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv`
    - `https://raw.githubusercontent.com/jasonong/List-of-US-States/master/states.csv`
    - `https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv`
  - SSL is imported to connect plotly with the CSV files through this code:
    - `ssl._create_default_https_context = ssl._create_unverified_context`
  - Two maps are created:

    a. A heat map that collects the sum of all COVID-19 cases in each state.

    b. A choropleth map that collects the sum of all COVID-19 cases in each county at a particular state that the user inputs.

a. Choropleth map:

- Pandas iterates through all the columns that contain the keyword for a particular state:
  - A manually coded example (doesn't take into account for user input)': `df_Maryland = df[ df['state'] == "Maryland"]`
- Pandas is used to set the date to the current date by using the built-in function `max()` function and then is used to iterate through all COVID-19 cases for the current date in that particular state:
  - `last_date = df['date'].max()`
  - `df = df_Maryland[ df_Maryland['date'] == last_date]` (manually coded example)
- Pandas is also used calculate the sum of all COVID-19 cases and deaths for a particular state through the built-in function `sum()`:
  - `df['cases'].sum()`
  - `df['deaths'].sum()`
  - This iterates through all cases and deaths for each county for that state in the csv file.
- Plotly is used to make a choropleth map that iterates through the COVID-19 data for all counties in order to create the map:
  - `fig = px.choropleth(df, geojson=counties, locations='fips', color='cases', color_continuous_scale="Viridis", range_color=(0, 20000) )`

b. Heat map:

- Pandas is again used set the date to the current date by using the built-in function `max()` function and then is used to iterate through all COVID-19 cases for the current date in that particular state:
  - `last_date = df['date'].max()`
  - `df = df[ df['date'] == last_date]`
- Pandas is also used to calculate the sum of all COVID-19 cases and deaths in the U.S. through the built-in function `sum()`:
  - `df['cases'].sum()`
  - `df['deaths'].sum()`
  - This iterates through all cases and deaths for for each state in the csv file which gets added up to calculate the sum of all COVID-19 cases and deaths in the U.S.
- Pandas is used to calculate the sum of all COVID-19 cases and deaths in each U.S. state through the built-in function `sum()` and `to_frame`:
  - `df = df.groupby('state')['cases'].sum().to_frame()`
- Plotly is used to make a heat map that iterates through the COVID-19 data for all states in order to create the map:
  - `fig = px.choropleth(df, locations=df['Abbreviation'], color=df['cases'], locationmode="USA-states", color_continuous_scale="hot", range_color=(0, 4500000), scope="usa"))`

THE INSPIRATION

Since 2020, we saw and felt the internal panic that people experience every time they go out during the pandemic. The question "Is today the day I get sick?" is always in the back of everyone's minds. So, we wanted to create an application that allowed people to be more informed about the world around them. We wanted people to be able to find how busy a place would be with extreme ease and to make it even more convenient, give them a simple "threat" bar to look at. The goal behind the app was an incredibly simple UI with a powerful backend so that the user had a seamless experience while the high-quality backend development ensured quality results every time. To add to this, since our goal was to provide convenient access to information, we also included a button that opens an interactive heat map of live US COVID-19 case data. It sources the data directly from the constantly updating New York Times COVID cases .csv file.

THE TEAM

V2 Team: Ainesh Chatterjee and Botond Parkanyi

V1 Team: Ainesh Chatterjee, Lorenz Driza and Salamun Nuhin

Special thanks to Charles Wang for the awesome logo!

THE JOURNEY

- V1 of CoronaSafe was born during the 2021 MocoHacks Hackathon: Lorenz Driza, Salamun Nuhin, and Ainesh Chatterjee built the first version of the app with just the surrounding risk calculation algorithm (we developed this at the last minute because the livepopulartimes library was not working at the time) and the heat maps, and wrapped it in a Tkinter GUI

- V2 of CoronaSafe was created for the 2021 Congressional App Challenge: Ainesh Chatterjee reused the backend code for the heat maps and the surrounding risk calculation, added the local risk calculation algorithm, built the master risk calculation, and wrapped it in a new Kivy Python GUI

- Botond Parkanyi developed a C# version of the V2 GUI but it had trouble getting data from the Python backend