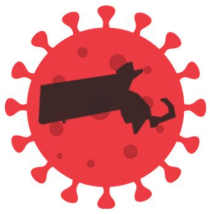
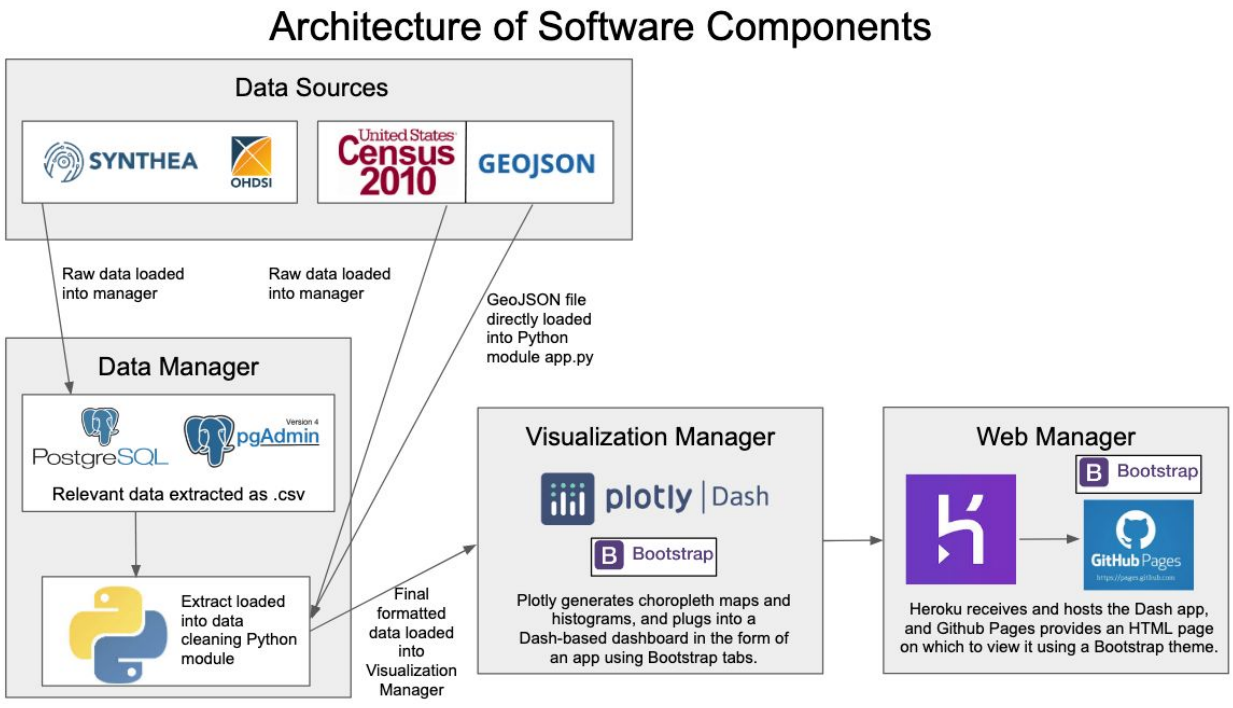
CO-MAP-V: Visualizing synthetic COVID-19 data in Massachusetts
Component Specification
CSE 583
December 17, 2020
Aja Sutton, Andrew Teng, Jason Thomas, Nanhsun Yuan



Software components:
The data we are using for this project is synthetic data mimicking the initial outbreak of the novel coronavirus (COVID-19) in the state of Massachusetts. The data was created by Synthea and is in the Observational Medical Outcomes Partnership (OMOP) common data model format. The data is separated out into individual tables, which are listed in the functional specification document. In terms of geographical data, a GeoJSON file represents the outline of the geographic area and geographic levels of interest for this project (discussed in the Functional Specifications).

Our software may be conceptualized in three components (Fig 1): a data manager, a visualization manager, and a web manager.

Fig 1. Software Components



Data manager:

- *What it does: Clean data obtained from querying via PostgreSQL and from US 2010 Census*
- *Input: Raw data and US 2010 Census*
- *Output: Cleaned, combined data of raw data and US 2010 Census to be used in Visualization Manager*

We have used PostgreSQL and PGAdmin to query the data for our specific needs. We created a local database and imported the synthetic data. These tools were chosen mainly for their ease of use and ubiquity in the field. We considered using R or python to merge and join these columns as well, but neither of those options were as intuitive. Additionally, because our team has prior experience writing queries in SQL, we felt using PostgreSQL was a suitable option.

Once imported, we were able to perform data exploration to explore which tables and columns were needed for our goals. After writing the query, we obtained a CSV file which was imported into a Python module. This CSV file is formatted with the main identifying column as County, as this format is best optimized for GeoJSON and Plotly to visualize and create dynamic choropleth maps. Python is then used to further separate the data and calculate mathematical features, e.g. positive rates, death rates, percentages by population. Correspondingly, a GeoJSON file containing vector line data corresponding to the geographic shapes of counties in Massachusetts is directly imported into the data manager (Python module app.py) as inputs. We have deployed Python modules for data cleaning (data_cleaning.py) and for integrating our visualizations into an application (app.py) that can be integrated as a Dash app (more below). These data are stored on our team Github Repository.

Visualization manager:

- *What it does: Creates Plotly-based choropleth maps and histograms based on data; integrates into a Dash-based app which is hosted by Heroku (next manager)*
- *Input: Cleaned data from Data Manager; GeoJSON file (integrated in app.py module)*
- *Output: Plotly visualizations; Dash app containing visualizations and app code integrated into app.py file*

Our Python modules contain calls to Plotly using our data to create dynamic visualizations. These visualizations rely on importing the pre-cleaned data (above) representing extracted feature counts (.csv format) by time (month) and geographic marker (i.e. county) from our team Github Repository and implementing these through the Plotly code library (specifically plotly.express). We have generated visualizations including a zoomable map with a time slider representing months, epidemiologic curves with a time slider representing months, and line charts (with similar time sliders) corresponding to our data categories. There are by-county choropleth maps and histograms of COVID-19 death counts, positive COVID-19 case counts, as well as a choropleth map of the population density of each county in Massachusetts according to the 2010 US Census. The time slider feature is created by calling the “animation_frame” variable within the choropleth code, initializing “condition_month” as the

variable. These Plotly visualizations are exportable as png/jpg files, re-centerable, and zoomable, amongst other intuitive features that are built-in with Plotly visualizations.

Dash is an open source library for creating web applications using only Python code. It requires little-to-no knowledge of Javascript or HTML, though these may be useful for more technical endeavours. For our purposes, Dash's basis in Python allows it to integrate well with other Python libraries, such as Plotly because Dash is in part built on Plotly. In order to import Plotly visualizations to Dash, the Plotly visualizations must be assigned variable names in a module or notebook and then called as input figures or data within the Dash application code. Dash then calls these when the dashboard application is loaded through the web browser. Each button/tab representing a different visualization of the data within the Dash app and implemented through Bootstrap is loaded at the time of calling the application.

Web manager:

- *What it does: Allows Plotly/Dash dashboard app to be hosted and embedded on an online website*
- *Input: Dash application (conglomeration of Plotly and Python) from Visualization Manager to Heroku; Github Pages for receiving Heroku code for embedding*
- *Output: App code hosted by Heroku in the cloud, which is embedded in a team website on Github Pages*

We used Dash to create a dashboard that features various dynamic visualization options for users, which is hosted on a Github Pages website and utilizes our team's Github repository for hosting our data and documentation. However, Github Pages is unable to host analytic or dynamic code such as we have within our functions described above. As such, our online visualization manager uses Bootstrap in order to integrate our Dash app into Heroku. Heroku is a web-based cloud application platform that is able to host and run our code for free. This approach provides an easily accessible storage and hosting solution for our application dashboard, which then integrates into our simple HTML-based Github Pages website. Our dashboard is visible in two "views" deployed through Bootstrap: comparison view and single view. Comparison view allows the user to click through two side-by-side but identical versions of our dashboard in order to compare the data. Single view allows the user to look at an individual copy of our dashboard. All views (3 simultaneous versions of our dashboard: i.e. 2 for comparison view, 1 for single view) are loaded at the time of loading the website. This provides some visualization lag, but ultimately better user experience when clicking through visualizations as it eliminates lag during actual user interaction with the website.

Use Cases and Interactions:

Use case #1

Description:

A health analytics personnel is interested in exploring the density of COVID-19 cases on the county level for the state of Massachusetts.

Interactions:

In a web browser, the user will open the website. This generates a call to the Python-based app code hosted by Heroku. Specifically, the Python application module is stored on the team Github Repository, and the computing of the application module hosted by Heroku is a series of Plotly and Dash calls which generate choropleth maps (combining the GeoJSON data of counties in Massachusetts) or histograms, and the dashboard interface, respectively. Dash creates a HTML version of the dashboard application, which is then loaded on the HTML Github pages website. When the website loads, the user will immediately be presented with a comparison view of two identical dashboards side-by-side depicting choropleth maps with time sliders, but these visualizations are changeable according to the user's needs based on tab-buttons that switch the Plotly visualization output to e.g. another map or histogram. The user also has the option to select single view, in order to view these visualizations in a slightly larger format, and without comparison and explore the synthetic COVID-19 data for the state of Massachusetts by county for each month between January and March, 2020: population density based on the 2010 US Census, death counts, positive case counts. The dataset was previously curated and extracted and therefore there is no communication between the database and the tool, rather the Plotly code will call the clean data and the GeoJSON, then present visualizations. Furthermore, the user can click and hover elements (hover provides a built-in Plotly pop-up that presents data categories for the element selected by the user), zoom in and out, export charts/maps, adding levels of interactivity and functionality.

Use case #2

Description:

A health analytics personnel or public health enthusiast wants to visualize COVID-19 data from the same source as our data based on their own selections using our visualization tool/dashboard (i.e. they want to "plug in" their own information).

Interactions:

The user will first download or clone our project Github Repository containing the scripts and queries necessary for the dashboard (e.g. app.py and data_cleaning.py). They will then follow our basic guidance we provide in our ReadMe file to set up a local database using Postgres and PGAdmin to import their CSV files that are in the OMOP common data model format. Using our provided SQL query, they will extract the necessary elements and features that will be visualized. They must provide their own GeoJSON file based on their geographic area and level of interest (widely available online with a quick search engine query). The user

then can create a web-based dashboard using a HTML hosting site, similar to our GitHub Pages + Heroku configuration, or host locally without Heroku on their own machine if they prefer. The user will be able to explore the data visually through an interactive choropleth map and epidemiological charts. By default, this transitions into a use case #1 interaction (above).

Preliminary plan:

- Gather data and ensure that it is in the correct format that is easily readable by Plotly. Ensure GeoJSON is correct and readable. Host these on Github Repository.
- Plot a choropleth and histogram/line chart/etc. using one feature of interest (e.g. case count by county).
- Begin coding within a Jupyter Notebook for ease of collaboration, and then transition to Python modules for the final product.
- Import these visualizations into our Dash dashboard.
- Integrate repository with Travis for continuous integration.
- Add a dropdown menu to the dashboard for the various complementary charts we have created in order to allow the user to click through these and compare with the map.
- Host the site through Github.
- Test site and visualizations for bugs or errors.
- Write up detailed instructions for use case #2 users, ensure code and scripts are pylint >8.