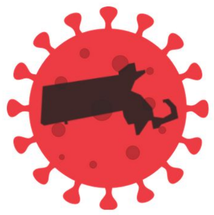
Visualizing synthetic COVID-19 data in Massachusetts

CSE 583 - Functional Specification
November 20, 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: a data manager, a visualization manager, and a dashboard manager.

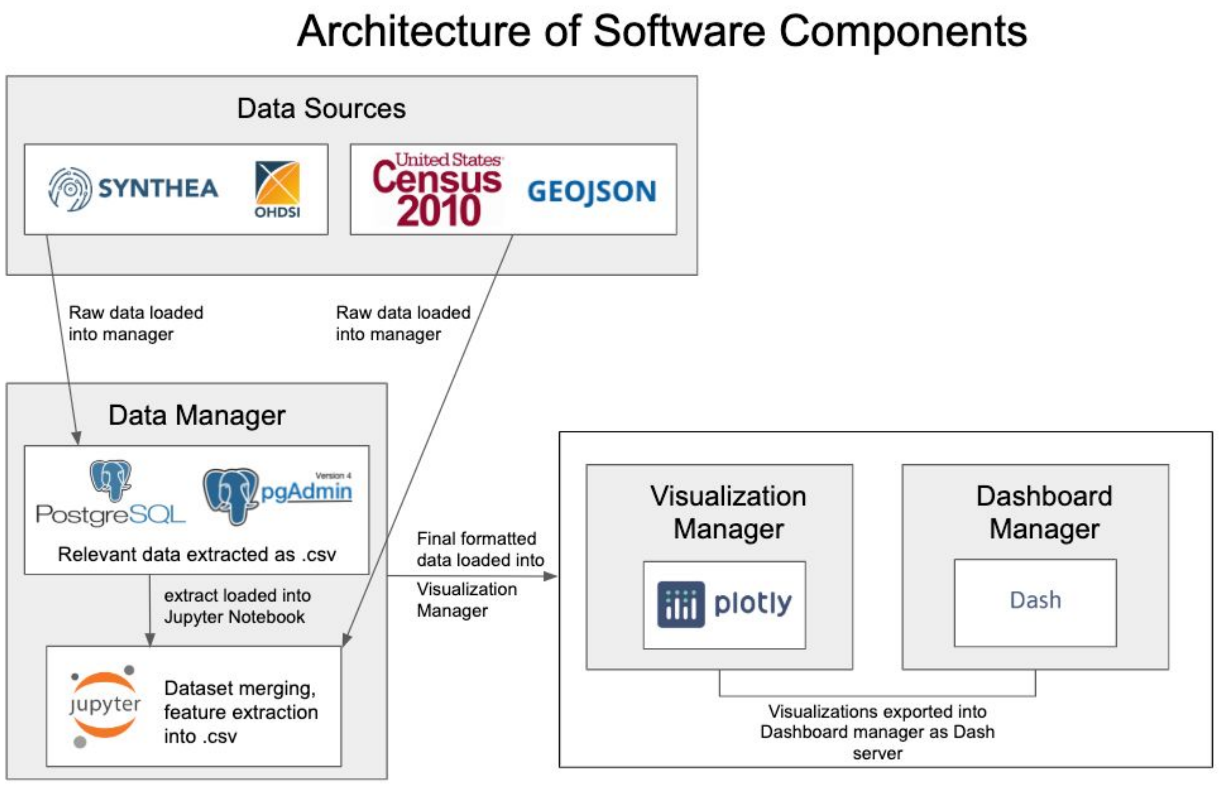
Data manager:

We are using 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 will then be imported in python, initially a Jupyter notebook as it provides better feedback and responsiveness, two features highly valued during our development phase. This CSV file will be formatted with the main identifying column as County, as this format is best optimized for GeoJSON and Plotly to visualize and create dynamic choropleths. Python will be used to further separate the data and calculate mathematical features, e.g. positive rates, death rates, percentages by population. Fig. 1 (on the following page below) visually depicts our data management workflow.

Visualization manager:

We intend to import Plotly visualizations into Dash. Specifically we will generate 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 (outlined above). These have the ability to export png/jpg files amongst other intuitive built-in features that are built-in with Plotly visualizations. These visualizations will rely on importing pre-cleaned data representing extracted feature counts (.csv format) by time and geographic marker (e.g. county name or Zip code) from our team github repository and implementing these through the Plotly code library. Similarly, our Plotly map will require Massachusetts counties' GeoJSON file and data extracted from the data manager as inputs.

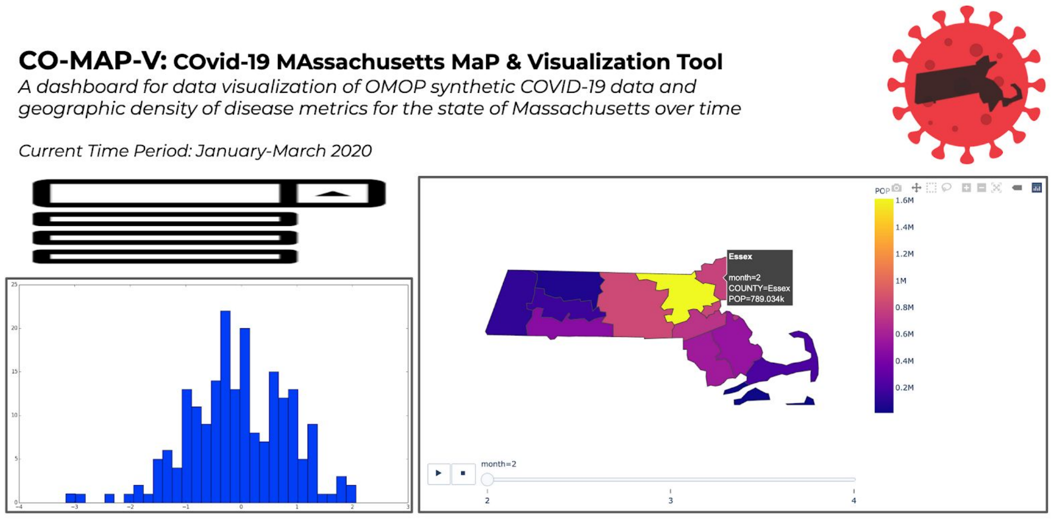
Fig 1. Software Components


Architecture of Software Components
Data Sources
SYNTHEA
OHDSI
United States Census 2010
GEOJSON
Raw data loaded into manager
Raw data loaded into manager
Data Manager
PostgreSQL
Version 4
pgAdmin
Relevant data extracted as .csv
extract loaded into Jupyter Notebook
jupyter
Dataset merging, feature extraction into .csv
Final formatted data loaded into Visualization Manager
Visualization Manager
plotly
Dashboard Manager
Dash
Visualizations exported into Dashboard manager as Dash server

Dashboard manager:

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 dropdown menu item in the dashboard is called on-demand through Plotly by Dash based on the user's selection. We will be using Dash to create a dashboard that features various dynamic visualization options for users, which will be hosted on a github.io website and utilize our team's Github repository for hosting our data and documentation. This is beneficial to the functionality of the dashboard in that this approach provides an easily accessible and free storage and hosting solution. Our dashboard will represent an exploratory visualization tool which will integrate visualizations built in Plotly. Plotly is a flexible browser-based graphing library for Python. It has the ability to draw numerous kinds of visualizations, from simple static line charts to choropleths and other map plots with time sliders. The figures below (fig. 2, fig. 3) represent a loose interpretation of the dashboard and a time-series histogram we hope to generate.

Fig 2. A mock-up of our Dash + Plotly dashboard hosted through Github webpages.

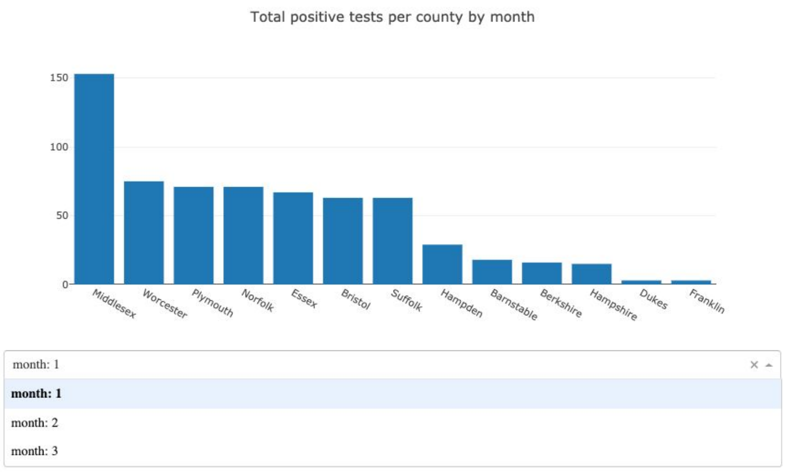


CO-MAP-V: COvid-19 MAssachusetts MaP & Visualization Tool

A dashboard for data visualization of OMOP synthetic COVID-19 data and geographic density of disease metrics for the state of Massachusetts over time

Current Time Period: January-March 2020

Fig 3. An example of a simple time-series bar chart created in Plotly.



month: 1

month: 1

month: 2

month: 3

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 dashboard which will immediately present a choropleth and chart created using Plotly and GeoJSON and displayed on the HTML website using Dash. 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 present a visualization. The user will then be able to select various features of interest from a dropdown menu (e.g. number of cases, number of deaths, etc.) and explore the synthetic COVID-19 data for the state of Massachusetts. 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. Additionally, because the choropleth also has point data for hospital locations pinpointed, users can also see the types and sizes of hospitals in various counties.

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 repository containing the scripts and queries necessary for the dashboard. They will then follow our basic guidance we provide in our ReadMe file to setting 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. Then they will run the python script which performs various mathematical calculations to obtain rates, e.g. death rate, case positivity rate. 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). Then using a HTML hosting site, similar to our GitHub pages website, the user will be able to explore the data visually through an interactive choropleth map and epidemiological charts. This by default 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.