A Global Database of Cholera

CS 4624 Multimedia, Hypertext & Information Access
Instructor: Dr. Edward A. Fox

Virginia Tech
Blacksburg, VA 24061
6 May 2020

Hemakshi Sharma, Gabby Alcantara, Michael Roberto, Andrés García Solares, Emily Croxall

Client: Dr. Luis Escobar

Table of Contents

1. Abstract

This paper describes the process and implementation details of work toward a database of Cholera records from 2010 – 2020. The WHO repository was used to extract and normalize data to build CSV files. Each year where data is available has a CSV file containing location and total number of cases in the location. The ProMED repository was used to collect data for the same timeframe. The data was extracted, condensed, and tagged for easier manual viewing. Data for all years available is given in one CSV file.

Data from WHO can be viewed in logarithmically colored maps based on the number of cases in each location. These visualizations are produced for each year in the study. The data from ProMED can be viewed in bar graphs which graph the number of articles that occur and in what weeks the articles are written for each country. These visualizations can be seen or downloaded at choleradb.cs.vt.edu. Additionally, all the CSV files of data produced are available for download on our website.

Due to the complexity of NLP and the inconsistencies in the ProMED articles, our data is not completely normalized and requires some manual work. Unforeseen circumstances, including the COVID-19 crisis, slowed the project's progress. Therefore, the ProMED data extraction did not proceed further, other data repositories have not been explored, and interactive visualizations have not been built.

The results of this project are compiled datasets and data visualizations from the WHO and ProMED repositories. These are useful to our client for future analysis as well as anyone else who may be interested in the trends of Cholera outbreaks. The results of data collection are formatted for easy analysis and reading. The graphics provide a simple visual for those who are more interested in higher level analysis. This project can be useful to developers who are working on data extraction and representation in the field of epidemiology or other case based global studies.

In the future, more repositories can be explored for more extensive results. Additionally, further work can be done with the ProMED set developed in order to condense it further and eliminate the need for any manual analysis after our program is run. The results of this project are all available publicly on choleradb.cs.vt.edu, including for download. All code is open source and available on Gitlab.

2. Introduction

2.1 Background

Cholera is an infectious disease that has global and simultaneous transmission, making it an ongoing pandemic. Cholera is a bacterium that can kill within hours, mainly due to a lack of fluids. There are an estimated 4 million cases annually worldwide and about 143,000 cases result in death. It is spread through food or water that is unclean or contaminated. It is especially prevalent in coastal areas and places with poor sanitation practices. The bacteria thrive in an environment with a mixture of fresh and saltwater that is standing still, often in deltas or other similar areas.

2.2 Client

Our client, Dr. Luis Escobar, is part of the Fishing and Wildlife Conservation Department at Virginia Tech and has a laboratory that works with cholera epidemics. They work to understand the temporal, spatial, and climatic pulses in past years cases. The laboratory has been a pioneer in disentangling the potential effects of climate change on this water-borne disease. Dr. Escobar has built a database of cases in 2014 to track start and end times of outbreaks, where outbreaks occur (not including the outliers), and how many cases are counted in every outbreak. This database was built by collecting and verifying data by hand.

2.3 Objective

Our problem was to develop a way of collecting data by web scraping and other automatic means in order to create a digital epidemiology. This means the automated collection, curation, storage, and analysis of the disease. Data will originate from two main sources, ProMED and the World Health Organization (WHO). From each source, the data will be collected, accounting for date ranges, number of cases, and location. The data will be analyzed to assess whether cases are stand-alone or part of an outbreak, as stand-alone cases are beyond this project's scope.

The WHO data repository contains data about the number of cases, deaths, and fatality rate per year per country. Such coarse-grained data will only give a general idea of where cholera has been more prevalent over the years. But, due to its yearly precision, it won't shed much light on the seasonal nature of the outbreaks, which is something our client is interested in. To process the WHO data, a Python tool has been developed that allows filtering and clustering of the data in various ways. This allows the WHO data to be collected in many different and meaningful ways. Using this data, choropleth maps of cholera across the world have been produced. For this purpose, another Python tool was developed. It takes in a data file and outputs a map as an image file, which enables easy visualization of the data.

ProMED is a collection of articles submitted by different sources that are regarded as a reputable source of data. The data in these articles is much more in-depth than the data provided by the WHO, giving specific numbers for certain cities/localities, down to the week. This is the primary data source that is used to gather our information about the trends of seasonal outbreaks. This data, however, does not have as much structure as the WHO data. There are many different ways

that the same data can be conveyed since the English language can be very complicated. Due to this, processing is broken into multiple steps where regular expressions and natural language processing are used to extract important data. Another Python script developed collects, analyzes, and processes the data to return a file that contains: the location, city, and country if applicable; the number of cases; the date of the outbreak; and the latitude/longitude of the location.

To allow others to learn about our project and access our results, a public website is available. It will contain basic information about our project, some data visualization elements such as maps, and download links for the data collection produced. There will also be a way of sorting or filtering the data, because users may be interested only in a specific country or time range. The website is hosted by VT and deployed in a virtual server with a public IP address and name, which has been requested and granted. This server is expected to last for at least a year.

The efforts from this project will be used to analyze the impacts of climate change and increasing water temperatures on cholera outbreaks. The database will be added to an international effort to inform policy to prevent human mortality around the world.

3. Requirements

Our system requirements have been defined by our client, Dr. Escobar. In order to be a continuation of his previous work, the solution needs to follow similar guidelines as he had. The main deliverable is a website that has data available for download. The data needs to be outbreaks of Cholera for the last 10 years, worldwide. The data is expected to include the number of cases per outbreak, the location of the outbreak, and the time range of the outbreak with emphasis on the start date. The website will allow anyone to access the data collected, in a well formatted database. For ease-of-use, the data is returned in the form of CSV files. The website is required, at a bare minimum, to have the data available year-by-year. Additional goals include the website having visuals for the number of cases and where the cases are located, and interactive maps that may be clickable.

The data collected needs to be from reputable and verifiable sources. Because of this, professional sources and databases are required such as WHO and ProMED. Another potential data repository is HealthMap.org; however, the administrators for this data were unresponsive. This is discussed further in *9. Lessons Learned*. The data collection also needs to be automated so that no manual data processing is needed. Therefore, one of the requirements is to create a program to get the data, clean it, and enter it into a database for accessing.

Another requirement that is important to our client is documentation of the process. The process that needs to be highlighted most is the web scraping and automated data collection. The documentation needs to be detailed and descriptive enough that someone from a non-tech background could operate and utilize this program for their own research. The goal is that Dr. Escobar can use code created in the future to compile data, and that the code could be applied to data collection for other diseases.

4. Design

Our system's general design is to pull data from online repositories, generate maps and other visual representations of the data, organize the data in a database, and post this information to our website. The data collection happens through two programs that get data from the ProMED and WHO websites, clean up the data, and export normalized numbers into CSV files. The organization of the data follows the required pieces of information: location of outbreak, number of cases, and start week (and end week if available). The data will be in a normalized form where locations will have a country if not a specific location, the number of cases will be an integer number, and weeks will be an integer number. If the end week is found it will accompany the start week. Raw data will also be available so that those interested can see where data comes from, such as seeing a precise date instead of simply a week number.

The maps generated assign colors to numbers on a logarithmic scale for visualization purposes. The countries with more cases per outbreak are colored in increasingly darker colors. These maps serve as a basic visualization tool in order to see area trends and how they change across years. These maps do not include any time information such as date ranges from outbreaks beginning and ending, or trends in outbreak seasons; they just include the year.

On our website, the homepage has visual representations of the data. There is another page where users can go to access the data downloads. Downloads are available as CSV files which will make accessing their contents and readability simple. Our website will also feature an About page with our team's contact information in case any user has follow up questions or comments. It also contains a description of this project and our client's contact information. This site is hosted on the VT CS servers and will be available to the public for access at choleradb.cs.vt.edu. For more details and screenshots, see *7. User's Manual*.

Our system is simplistic yet effective in data collection. Our main purpose is to gather information and make our project replicable. For our client's goal of usability, our design is consistent with his past project.

5. Implementation

5.1 WHO Data Gathering

The WHO data is processed with a Python script called who.py that reads from a CSV file (formatted like the ones in the WHO data repository). See *8. Developer's Manual Table 2. WHO Directory Details* for descriptions of each file. At a high level, the program re-organizes the data based on the arguments that were specified by the user. More about these arguments will be explained in *7. User Manual*. The algorithm the script performs will iterate over the entire input file ($\Omega(n)$). Each row (which is formatted as Country, Year, Value) will be filtered out if it doesn't fall into the year range specified by the user, or if the country doesn't match the one specified by the user. Otherwise, a test is made regarding whether the value for the country needs to be added to an existing row in the output (in case the user is adding up the values for different years), or if this is the first time the program encounters this country in the input. Next, the program sorts the list resulting from the previous step. The sorting algorithm used is Selection Sort ($O(n^2)$), and the order of sorting is from greater to lesser, to prioritize the most relevant ones. As it is sorted, the list is truncated to the maximum number of countries specified by the user. After that, the program outputs the list as a CSV file or prints it to the console, depending on whether the user specified an output file.

There's another script which contains a batch of calls to who.py. It is called batch_run_who.py. It is useful when the user needs to systematically retrieve exhaustive information from the WHO files.

5.2 Map Creation

The maps are generated with the make_map.py script, which heavily relies on matplotlib, numpy, and Pandas. The way the script works is, it takes in the output from who.py and creates a Pandas dataframe to facilitate access. It then creates an array with the bins for the map legend and assigns each country a bin based on the value for the country. That assignment is done in a logarithmic way because the distribution of cholera cases across the countries is roughly exponential, with a few countries containing most of the cases, so this logarithmic scale compensates for that and the result is a uniform distribution of countries across the bins. Then, a continuous color gradient is generated and discretized, so each bin will have an associated color hue. The map is drawn using a file containing the shape information, with matplotlib, and for each country in the Pandas dataframe, that country in the map is colored with the corresponding color. Lastly, the legend and other texts (header and footer) are added to the image, and it's saved to a PNG file.

There's another script called batch_run_map.py which, similarly to batch_run_who.py, can be used to batch-run the map.py script.

5.3 ProMED Data Gathering

The ProMed data is much more complex to deal with. There is no specific API that allows data collection from the website, so HTTP POST requests are used to retrieve the HTML. There are 2

different POST requests required. One request is to get the article IDs for the current page, and one request is to retrieve the actual article. Multiple requests are performed in order to get all the article IDs because they are spread across multiple pages. A Keep-Alive socket is set up to use one TCP connection to gather all the data. First, a page that lists all the ProMed articles is fetched. The page is then iterated over to get the ID for each individual article. Finally, using the ID number, an HTTP request is made to get the HTML. See *8. Developer's Manual Table 3. dataGatherer.py Details* for details about each function.

After each individual article is collected, the HTML is parsed to extract the raw data. However, before extracting the data, the HTML needs to have preprocessing done. Some of the older records from 2010 and 2011 have formatting problems where there are
 (break lines) in the middle of the sentence which, when run through an HTML parser, leads to analysis of partial sentences. So, the function break_removal() is called to clean up the HTML data before it is sent through the parser. The BeautifulSoup library is used for the parsing. BeautifulSoup takes the HTML as input and provides a list of the stripped strings that represent the article. Each entry in this list constitutes a paragraph. There is now a list of paragraphs in the article which can contain "noise" or irrelevant data. Therefore, filtering of the noise is required before analysis begins. Using a predefined list of keywords (found in keyWords.txt), sentences are analyzed to decide relevancy. If enough keywords are found, the paragraph is kept. If not, the paragraph is deemed irrelevant and not included in the condensed article.

One thing to note is there can be multiple articles within a single ProMed article. Sometimes articles discuss about multiple locations, so a separate ProMed record is created for each of the locations mentioned. There is a specific pattern that separates each of these articles within the article. The flow diagram can be found in Figure 12. Anything that takes the form of something like "[1] Cholera - Somalia" represents a change in the article discussion and signifies that a new ProMed Record should be created for the next data.

There is important metadata stored in the beginning of the article including the archive number, the source of the article, the general location that the article references, and the date the article was published. This data can be useful for analysis. Therefore, it is found in parsing and added to the promedRecord object. The condensed article data is part of this object as well, allowing all necessary information to be stored together.

Once the article is condensed, one final intermediate step occurs; potential cases within each article are "highlighted." Name Entity Recognizers (NERs) have a highlighting scheme that can be used for natural language parsing. A tagging scheme was created to mark potential cases in each article. The motivation is that manual analysis will be simpler and more straightforward. Alternatively, future development of this project can run the tagged data produced through natural language processing routines. Before tagging is done, the elements that need to be tagged are determined. This is done by training an NER model using the Spacy library within Python. The code repository has a file called train.json which contains 200 training sentences created by NER to help find what should be tagged. The goal is to tag the cases and the dates within each article. With this training data, a blank Spacy model is trained over 20 iterations so a model that can catch all important information is built. The program runs through 20 iterations so little loss would occur. This means the model will not miss any phrases, which can happen when a model

is over trained or under trained. After running the training and manually analyzing the results manually, it was found that 20 iterations provides a good model to maximize the number of results found. Once the list of tagged data is built, tags are highlighted within the condensed article. The structure of highlighting is as follows: “There have been <CASE> 32 new cases of cholera </CASE> in <DATE> week 2 of 2019 </DATE>”. This allows the text to stand out and as well as makes it easy to parse through this data for potential future use. Using this tagged format, useful data such as total number of cases per record and number of reported cases per record are easily pulled from the article. With this information, a new field called rankSeverity is added. The rank is calculated by sorting each record based on total number of cholera cases in descending format. This is done by looking for case tags in the data just created. When a case tag is found, the number of cases found in this document is incremented by 1 and then by a search for a number within the case tag. If a number is found, it is added to a total cases variable for that article. Once all articles have an associated number of cases and total number of cases, they are sorted by number of cases and then generate a rank severity for each one. This is done by walking down the list incrementally and giving a case a sequential number. So, the case with the highest number of total cases, which will be at the front of the list, will have a rank severity of 1, the next highest will have a rank severity of 2, and so on.

Finally, important case information is extracted from condensed articles and moved to CSV format. Information extraction is done using 14 regular expressions; see Appendix A for more details. These regular expressions are complex and attempt to capture the different ways that someone can say the same phrase such as: “There are 20 new cases”. But there are many different expression types that can occur. The program attempts to match a case to a location and date; however, these are not always specified in the current sentence being analyzed. Because only one sentence is analyzed at a time, context is lost in each iteration. Therefore, it is necessary to check each sentence of an article for location and date information. Most of the processing is done in the analyzeData() function. If a last known location and a last known date can be found, possible correlations could be found to fill in missing data. This may not be the most accurate method. Some analysis needs to be done to see if results are accurate.

In the generateRecords() function, the case, location, and date data are combined together. It is possible that for a single case, 2 dates could be applied which classify as a date range.

Finally, normalized data is converted into the final data format. This involves stripping any words out of the date and cases data, converting the date to a week format (week 12 of 2019) and converting the location to a latitude and longitude format. To strip the extra words, more regular expressions are used. To convert the location, the Google Maps API is used. The location and the 2-letter country alpha code (if known) is fed to the API, to get the latitude and longitude coordinates back. Once this data is collected, it is added to the final CSV data file. This is done in the normalizeData() function.

5.4 Report Graph Creation

This module, named graph.py, generates a set of graphs given a collection of reports in JSON format. The format of the file should be a JSON object in which each report is an object with at

least a “location” and “datePublished” attribute. This is the kind of file that the dataGatherer module can generate from the ProMED data.

It works by first loading the JSON file into a Python dictionary. The keys will be the report ID and the values, the report object containing the attributes. It iterates through the dictionary keys, and for each value, it extracts its location and its publication date. The publication date format is extremely heterogeneous, so the code accounts for a wide variety of formats. These are some of the date formats encountered:

- day - three letter month - year
- day - full textual month - year
- three letter month - day - year
- month - year

There are also some erroneous or misspelt dates that the program tries to correct, which are mainly due to entry errors. Finally, each date is converted to a Python datetime object, and all dates and locations are put into a Pandas dataframe. A list of locations is built to facilitate iteration. Locations with only 1 report are discarded due to 2 reasons: it doesn’t make any sense to draw a graph with only one data point, and most of the time it corresponds to an erroneous/misspelt location.

The plots are drawn using matplotlib and the Pandas dataframe. First, two graphs (Figures 1 and 2) are created for the complete set of reports (global): one of them groups the reports by month and the other one by week. Then, by iterating over the locations list, two graphs for each location are drawn (one monthly and one weekly). Only the rows of the dataframe that match each location are considered for each graph.

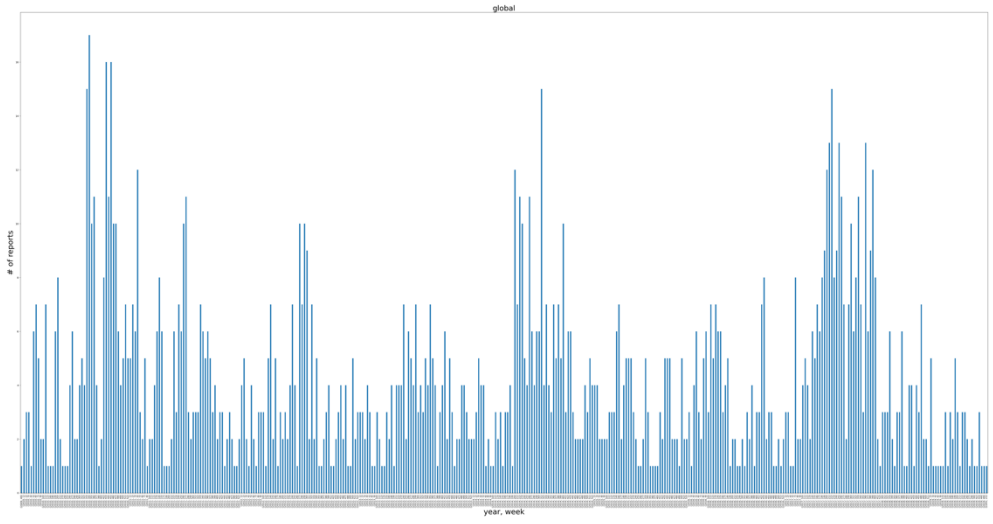

Figure 1. Weekly Global Graph

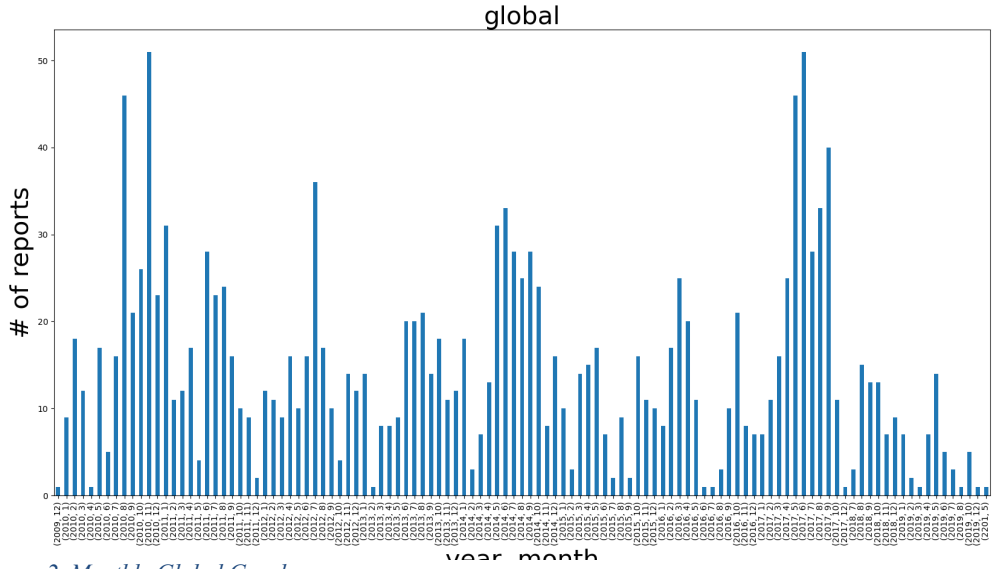

Figure 2. Monthly Global Graph

5.5 Website

The site is a relatively simple PHP + HTML website. Each one of the three pages (Figures 3, 4, and 5 in section *7.1 Tutorial of Use*) is a PHP file and makes use of a “header” and “footer” HTML for code clarity. The searching functionality is implemented with the help of AJAX requests. The downloads page allows users to filter the files through the form and sends a get request to search.php. Then based on the form input, the search is resolved, and the file iterates over the “/downloads” directory, picking the relevant files. Links to those files are assembled in an HTML format and sent back to the view (the downloads.php page). Additionally, the /downloads/ directory tree is publicly accessible from the browser.

6. Testing, Evaluation, Assessment

The data collected from the repositories is not dynamic. To test the method of pulling data, manual verification was done to ensure the articles had relevant information to cholera cases or outbreaks. Through testing, it was discovered that most of the early ProMed articles were formatted in a way that was not compatible with the current method of parsing. These early articles contained
 tags between every line, whereas a new paragraph was indicated by two consecutive
 tags. This caused the current parsing method to believe these
 separated lines are their own separate paragraphs. Formatting of these earlier articles was fixed using a preprocessing method.

A user test was done for usability of the website. A paper prototype test was conducted in which each page of the website was printed on separate pieces of paper and tasks were outlined for a test subject to complete on the site. The tasks were

1. Locate the map for WHO data of cases in 2014 per country
2. Download the fatality rate per country WHO data for 2011
3. Download the monthly graph for Kuwait from the ProMed data
4. Locate the team members' contact information

As the research subject tried to navigate the pages to perform the tasks, they were given the next subsequent printed page of the website based on the button they "clicked."

The research subject was Taylor Casarotti, a Virginia Tech student who is interested in learning more about Cholera, which is one of the identified user types. During the task run-through, she completed task #1 quickly and efficiently, and located the correct map displayed on the homepage carousel. In task #2, she immediately clicked the "downloads" tab on the website. She then noted that it was difficult to locate the exact data she wanted from the list of downloadable files and suggested some sort of sorting feature. This was noted and has been implemented in the current website design. She had the same complaint with task #3 but was still able to locate the desired information. She also completed task #4 quickly, navigating to the "about" tab in the website. In conclusion, she found the website easy to navigate and found each task to be clear. Through this test, the website design was improved by implementing a search feature on the data.

7. Users' Manual

7.1 Tutorial of Use

Navigate to the website at choleradb.cs.vt.edu. Once here, the homepage is automatically displayed, see *Figure 1*. This features maps that were developed from the WHO data. The maps themselves run in a carousel that rotates automatically. The title above the map will tell what year the map correlates with. At the bottom there is a message detailing how the maps are logarithmic. It is important to read the legend on the map to understand the case number range that each color represents, as it is logarithmic and not linear. This was done for visualization purposes as the case numbers have large ranges.

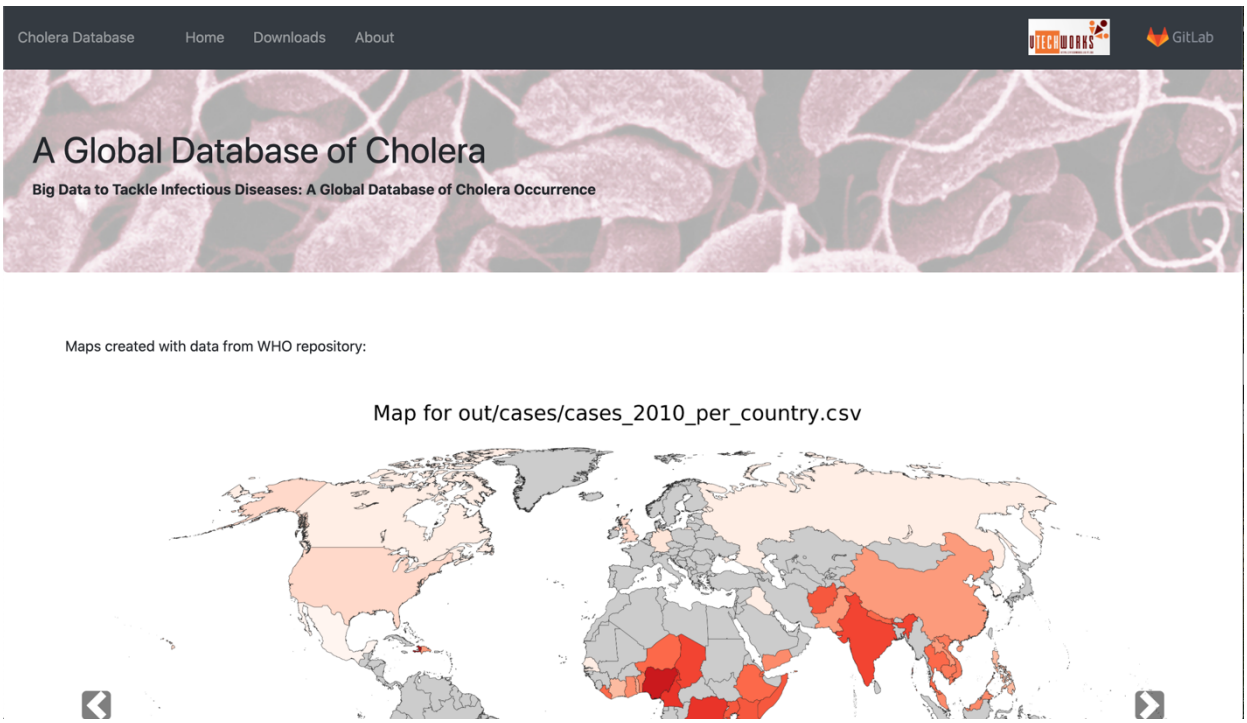

Figure 3. Website Home Page

By clicking on the Downloads tab along the top of the page, all data can be viewed. There are three lists of data displayed on this page (see *Figure 2*). The first column is data from WHO that was collected. For ease of use, this list is searchable by year. The second column is data from ProMED that was collected. This column is also searchable by year. Finally, the third column is graphs developed from the ProMED data. These graphs feature the number of cases over a certain time frame for various locations; see *8. Developer's Manual Graph Module* for more information. This column is searchable by location and grouping type (any, month, or year). All of the data that is on this page is easily downloaded by clicking the link desired.

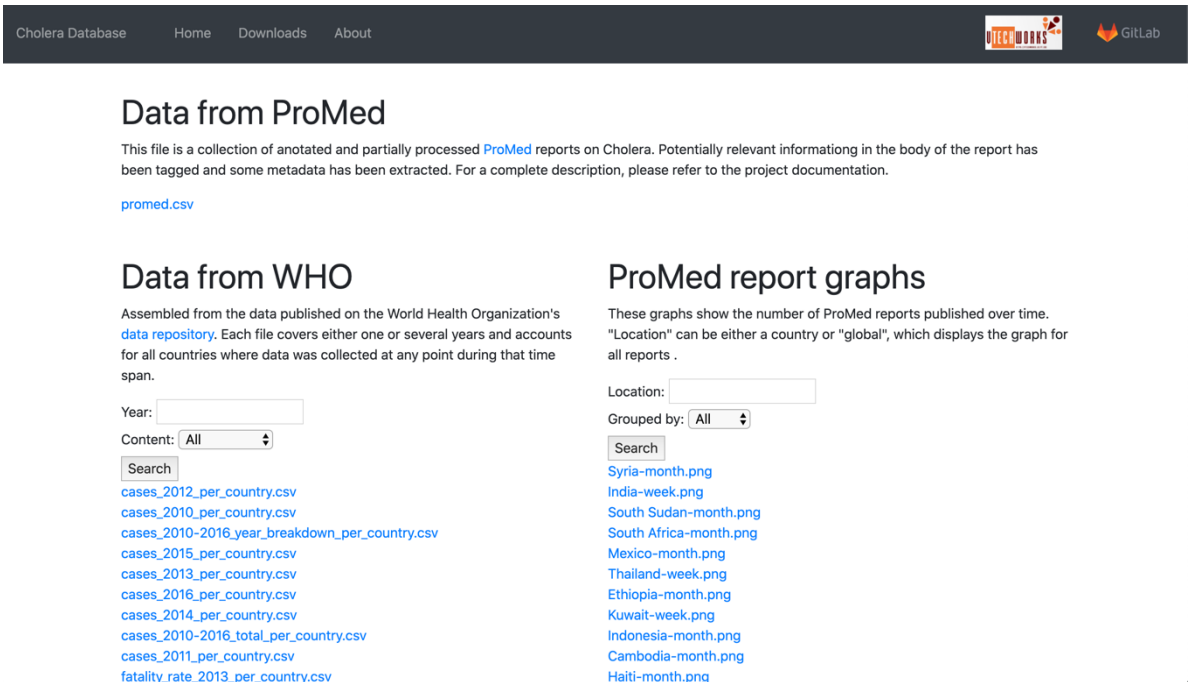

Figure 4. Website Downloads Page

The final page on the website is About; see *Figure 3*. This page features information and contact information for our client as well as all team members. There is information about this project and other projects done by the client.

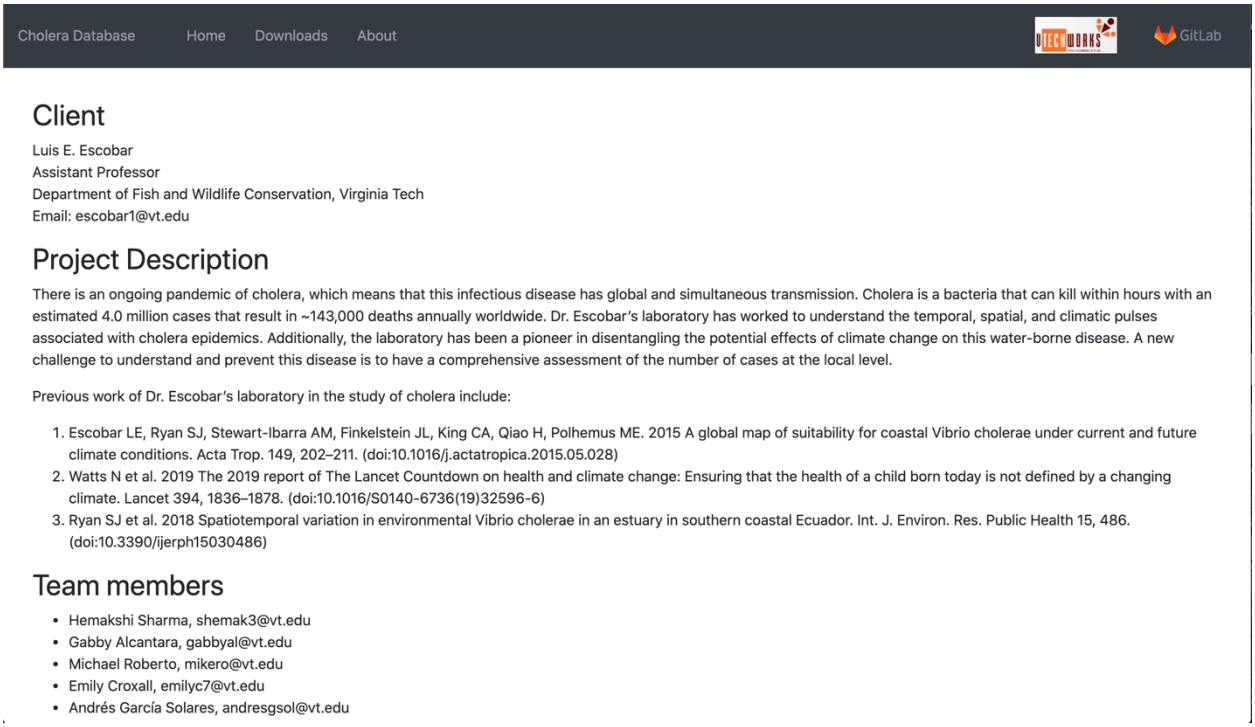

Figure 5. Website About Page

7.2 User Goals & Use Cases

This project accounts for and supports various users with end goals. For each user, a goal was developed, with use cases to achieve those goals to help develop a user-friendly application. The first user group supported is epidemiology researchers or trackers whose end goal may be to see data presented visually and by year and country. Another user group with a similar goal is Cholera experts. The third user group includes people who are interested in learning about Cholera. For this user, use cases provided include those for the goal of seeing where outbreaks are most common and at what time of year outbreaks occur, as well as finding out how many cases are common per year and per country in an outbreak. The fourth and final user type is a developer who is looking for open source code to develop a project similar or based on this project. The end goal for this user is to find code that is replicable in a web-scraping process.

Table 1. User Types and Descriptions

User Type	Short Description	General Goals
1. Epidemiology Researchers or Trackers	Researchers and disease trackers who use past cases and databases to map the spread of their disease. They may utilize digital epidemiology.	They can utilize our project for their own diseases by changing some of the input data. They can use our data to view the spread of Cholera worldwide over multiple years.
2. Cholera Experts	This user is anyone who specializes in studying Cholera. It could include doctors or researchers.	This group can use our data to view the spread of Cholera worldwide over multiple years. They can also build on our data and expand on the years and precision of our data.
3. People Interested in Cholera	Anyone who is generally interested in Cholera would fit into this user type, including the members of our group.	A goal for this user group is to see where and when Cholera outbreaks occur most in the world and to find out how many cases are typical in an outbreak.
4. Developers (see *8. Developer's Manual*)	Software developers who are interested in finding open source code to develop similar projects.	This user's goals include getting access to open source code that they can utilize and replicate.

For each goal, tasks and subtasks were developed in a user-centric form for how a goal is accomplished. Figures 6-10 are task diagrams with an accompanying description of the process.

Goal 1:

This goal (see *Figure 6*) consists of viewing general data about Cholera in a map. Researchers may want to get a general idea without having to dive into our huge files, but they still want to see numbers together with the visual elements (colors).

a. Users access the website via their browser (the site has a public domain name).
b. Maps are presented on the Home page and users can browse them using clickable arrows.
c. View the numbers on the map and interpret using the legend.

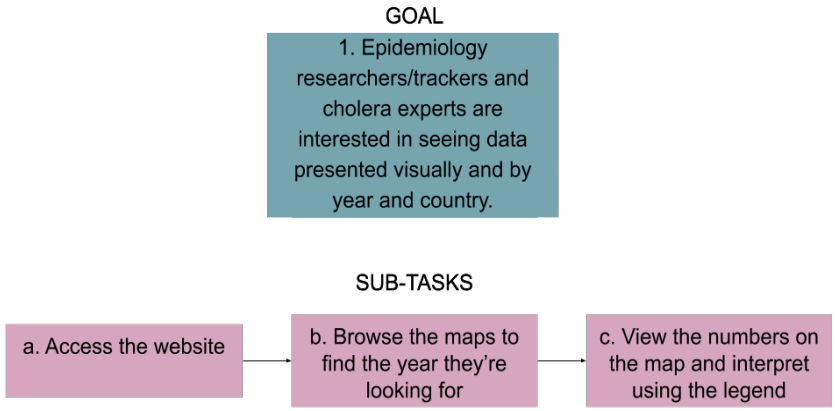

Figure 6. Goal 1 Subtasks

Goal 2:

Goal 2, illustrated in *Figure 7*, consists of using the data the Cholera website provides for different purposes such as tracking patterns over years and by country and making predictions. Researchers will need data organized by year and country for their analysis purposes.

a. Users access the website via their browser (the site has a public domain name).
b. Users may search through the downloadable data by navigating to the Downloads tab. Here they will find the search bar where they can search by year and country using the dropdown menu.
c. After filtering the data, they may download the relevant data by simply clicking on the CSV file name.

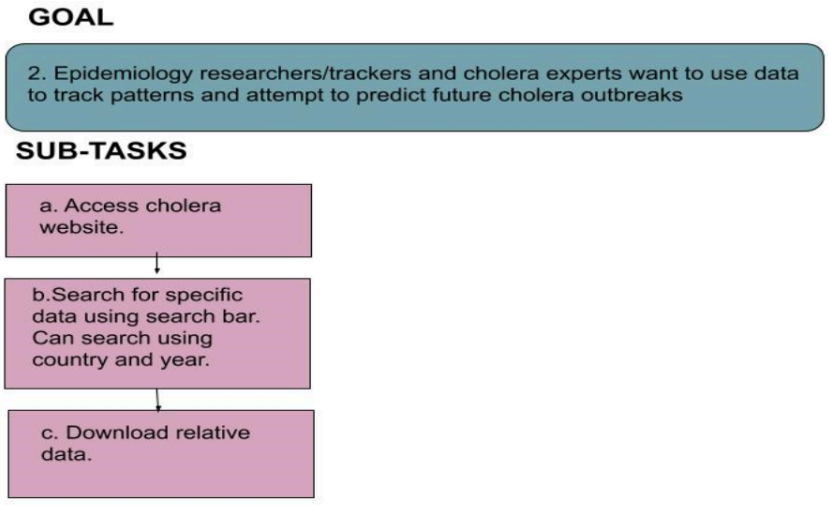

Figure 7. Goal 2 Subtasks

Goal 3:

The users with this goal (see *Figure 8*) will be regular people trying to learn about Cholera and its seasonal patterns, who are interested in getting to the files. It's likely that they will look at the maps first, to get an idea of what the general situation of Cholera is.

a. Users access the website via their browser (the site has a public domain name).
b. Maps are presented on the Home page and users can browse them using clickable arrows.
c. Based on what they learn from the map, they can then navigate to the "downloads" section, filter by country/year, and download the files they're interested in.

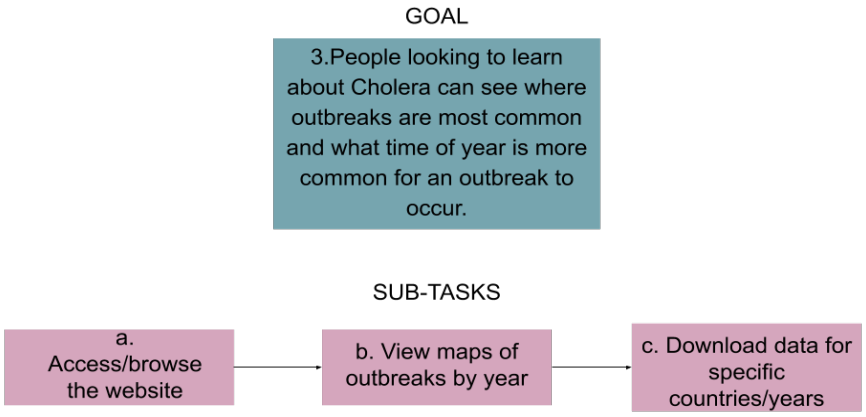

Figure 8. Goal 3 Subtasks

Goal 4:

This goal (see *Figure 9*) is similar to goal 1 in the sense that users are interested mainly in the maps. However, the users are not professional researchers, but normal people, trying to get an idea of which countries have a higher prevalence of Cholera, how it has evolved over time, etc. So, they value a powerful visualization and don't care that much about the numbers in the legend.

a. Users access the website via their browser (the site has a public domain name).
b. Maps are presented on the Home page and users can browse them using clickable arrows.
c. The legend is at the bottom of the image, but users will probably get more meaningful information from the colors, which allow for an easy comparison.

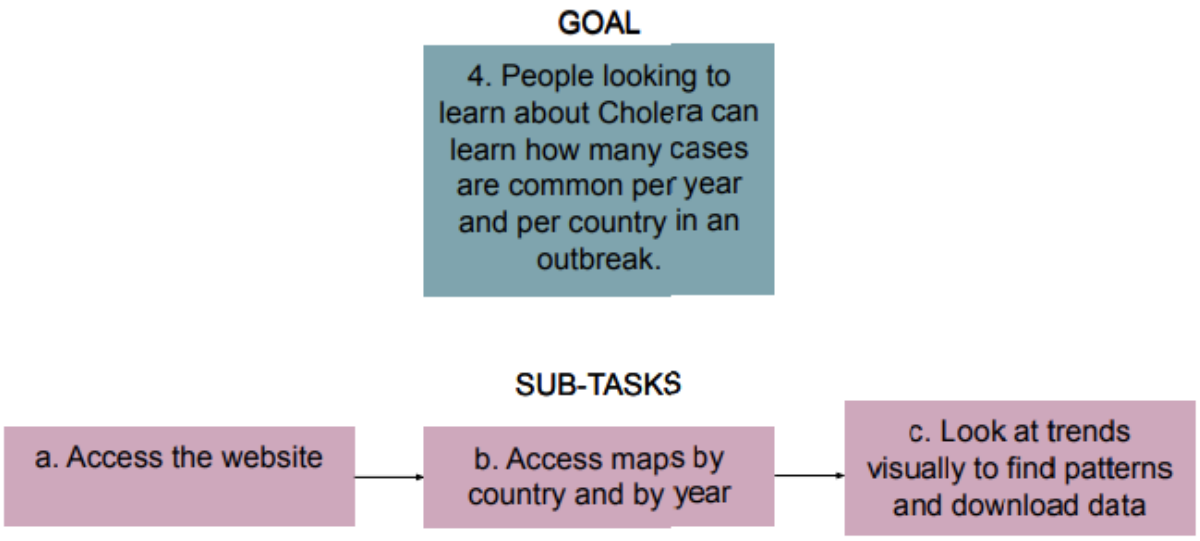

Figure 9. Goal 4 Subtasks

Goal 5:

This last goal (see *Figure 10*) relates to the "workflow" aspect of our project, i.e., the process followed to reach the results in this project, and project reutilization. Along with the results, the code and documentation for this project will be published. Other developers may want to build on the code or adapt it to their needs (for which they will appreciate the documentation and the description of our process).

a) Our website will include a link to a public repository, but developers may reach it from other sources as well.
b) Since the repository will be public, they will be able to download its content. If they are trying to modify it for their own projects, "forking" will probably be their best option.
c) Documentation of the code and the workflow will be available publicly as well, probably in the website and/or the repository.
d) This last task is pretty much up to them and out of the scope of our project.

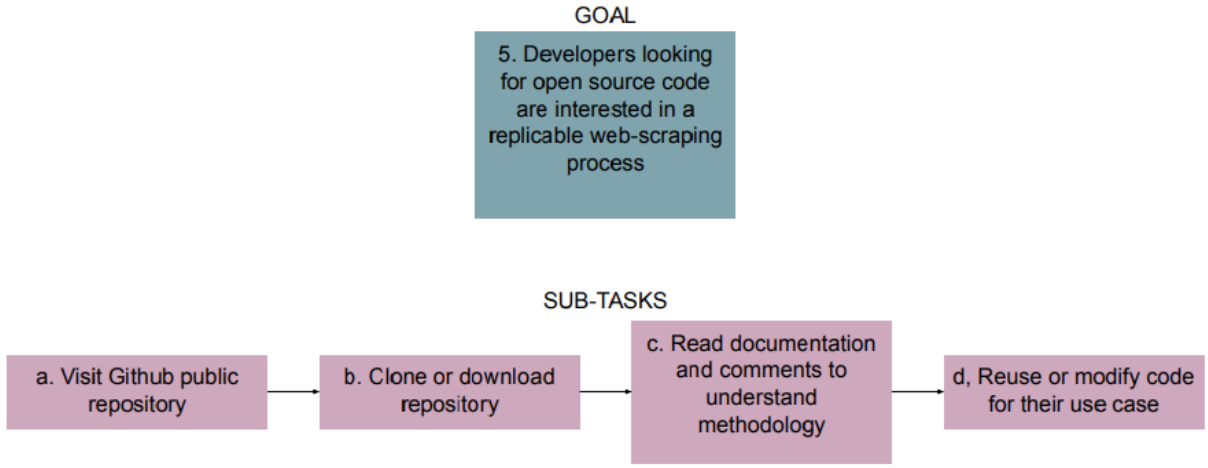

Figure 10. Goal 5 Subtasks

8. Developer's Manual

In *7.2 Users Goals & Use Cases*, developers will find a helpful guide to the users this system was designed for and what use cases were considered in the development. There are useful diagrams to show the workflow of each use case.

8.1 Python Requirements

To run the Python modules, developers need to make sure they have the correct dependencies installed. A list of all dependencies can be found in *Appendix B*. A requirements.txt file can be found in the repository, along with the rest of the code:
https://git.cs.vt.edu/mikero/choleradatabase

8.2 WHO Modules

Table 2. WHO Directory Details

File Name	Dependencies	Inputs	Outputs	Description
who.py	argparse, csv	User-provided arguments (for more info, see *7. User Manual)*	CSV file or console print, depending on user arguments	Performs the bulk of the WHO data processing, as explained above
batch_run_who.py	argparse, who	None	A set of CSV files	This script invokes the who-py module with a variety of predefined arguments in order to automatically produce a set of CSV files.
make_map.py	argparse, matplotlib, numpy, Pandas, mpl_toolkits	User-provided arguments (for more info, see *7. User Manual)*	PNG image containing the map	Performs the map plotting operation, as explained above
batch_run_map.py	argparse, make_map	None	A set of PNG images	Invokes the make_map module with a variety of predefined arguments in order to automatically produce a set of maps

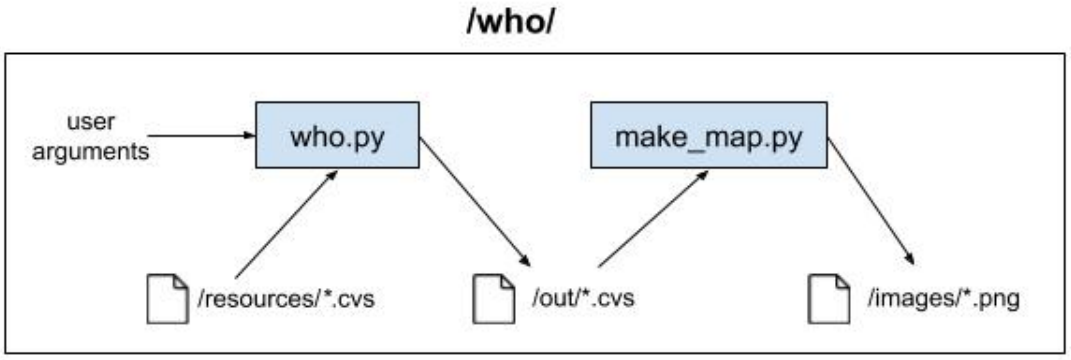

Figure 11. WHO Directory Workflow

The who.py module can be invoked with a wide variety of command line arguments, which, when combined, can produce different results. Here is a list of all of them:

- filename (positional): path to the source CSV file containing the WHO data. Typically, it will be "who/resources/cases.csv"
- -m, --minyear (optional): minimum year (included) for the time range that will be used to filter the WHO data.
- -x, --maxyear (optional): maximum year (included) for the time range that will be used to filter the WHO data (when minyear == maxyear, only that year will be considered when gathering the data).
- -n, --numcountries (optional): maximum number of countries to display after the data has been sorted. Only the n countries with the top values will be displayed.
- -c, --country (optional): if specified, only data from that country will be gathered. If absent, every country will potentially be included in the list.
- -b, --breakdown (optional switch): Boolean value. If present, a year by year breakdown of the values will be displayed for each country. If absent, values across the time range will be added up for each country.
- -f, --fileoutput (optional): name of the file where the output will be written to. If absent, the output will be printed to the terminal.

The make_map.py module takes in some arguments as well:

- filename (positional): path to the source CVS file containing the result of a who.py invocation. Typically, it will be a file under "who/out/".
- fileoutput (positional): path/name to the output image that will be generated. Typically, it will be under "who/images".

8.3 Graph Module

As mentioned in *5. Implementation*, a variety of date formats are considered when interpreting the report date, and some possible errors are accounted for. However, this is an ad-hoc implementation that works for our data collection but could fail if a new, non-standard date format is plugged in. In that case, the datetime module will most likely throw an error.

8.4 Website

The website was created with a LAMP stack in mind, although it makes no use of the database. A Windows XAMPP environment was used for development.

The production version runs on a virtual machine provided by the Virginia Tech Computer Science department. Its name is choleradb.cs.vt.edu and its public address is 128.173.237.71. Developers can access it via SSH connections, but only from the VT network or the VT VPN.

8.5 DataGatherer

The data gatherer module can be found in the dataGatherer.py file. Much like the WHO module, make sure that you have all the dependencies installed. A list of all dependencies can be found in Appendix B. A requirements.txt file can be found in the repository, along with the rest of the code: https://git.cs.vt.edu/mikero/choleradatabase.

Table 3. dataGatherer.py Details

Function Name	Dependencies	Inputs	Output	Description
getSingleProMedData	None	ID Number as integer Socket as a Requests Socket	List of PromedRecord objects	Retrieves a ProMED record by using HTTP, grabs important information and condenses the article text
getAllProMedData	None	None	A dictionary of PromedRecords that are indexed by a number	Access all the ProMED articles related to Cholera and call getSingleProMedData to create a list of condensed data
normalizeRecords	None	List of caseRecords Last known location as string Date published as string	List of formatted records	Take the data collected from generateRecords and normalize the data so that cases are a number, and the location has a latitude and longitude
getMonth	None	Month as string	Month as int	This function takes the date string and outputs the month as

				an int for normalizing the data.
generateRecords	None	List of caseRecords containing only cases List of caseRecords containing only dates List of caseRecords containing only location Last known date as string Last known location as string Date published as a string	List of caseRecords that are completely filled out with the case, date, and location	This takes the partial information obtained from analyzeData and creates full caseRecords. By using the sizes of each of the list inputs, generateRecords tries to create the correct records that were listed in the full article.
parseNormal	None	Current line as string Last known location as string Date published as string	List of caseRecords that are completely filled out with the case, date, and location	parseNormal uses regular expressions (see *Appendix A*) to parse out the cases, date, and location. It creates partial caseRecords that are then passed into generateRecords
parseSpecial	None	Current line as string List of partial caseRecords	List of caseRecords that are completely filled out with the case, date, and location	parseSpecial is for some special types of expressions that appear in ProMED records like “Uganda (53)” to express the number of cases.

		dateFound as Boolean Last known location as string Date published as string		Cases are sent see if a location and date for each special case is found, then pass those incomplete caseRecords into generate Records
analyzeData	None	A ProMED article as string	List of caseRecords that are completely filled out with case, date, and location	analyzeData uses all the defined regular expressions (see *Appendix A*) to send current sentence to parseNormal or parseSpecial
createSynonyms	A list of stop words already defined	None	A list of the most common words in the article sorted	Searches through all the ProMED articles to find the most common words that can be used to identify the most important parts of the article. Meant to be analyzed manually
expandSynonyms	NLTK corpus Lemma names	None	A list of our synonyms and synonyms for those	Takes the synonyms defined in createSynonyms and expands the list to include synonyms for those words
convert_dataturks_to_spacy	None	File path and a correctly formatted JSON	A correctly formatted JSON file that spacy can use to train	Takes a JSON file with tags and converts it to a JSON file that can be used to train an empty spacy model for use.
train_spacy	A correctly formatted JSON file Spacy	A correctly formatted JSON file	None	Uses the JSON file to train a spacy model over 20 iterations to find cases and dates for tagging
break_removal	getSingleProMedData	html_data as string	html_data as string	This function takes raw text pulled from ProMED and will

				remove excessive breaks that interfere with analysis.
insert_tag	None	Text as the full body of text that is being tagged Start as the start character offset End as the end character offset Tag as the string that will be used as the tag	The full article with an additional tag that highlights the important data found from the spacy model	This function allows us to add any tag to an article given a start index, an end index, a body text and the tag to be inserted
tag_records	None	Record as the ProMED record that will be tagged	A taggedData object that contains the ProMED record and the tagged data	This function takes a promedRecord object and calls nlp() on the provided text to allow spacy to find the tags based on the given model, which is then run through insert_tag to insert the data into the text.
generate_CSV	None	Dictionary of taggedData objects	A file named output.csv containing all the taggedData object data in CSV format	Converts a set of taggedData objects into CSV format

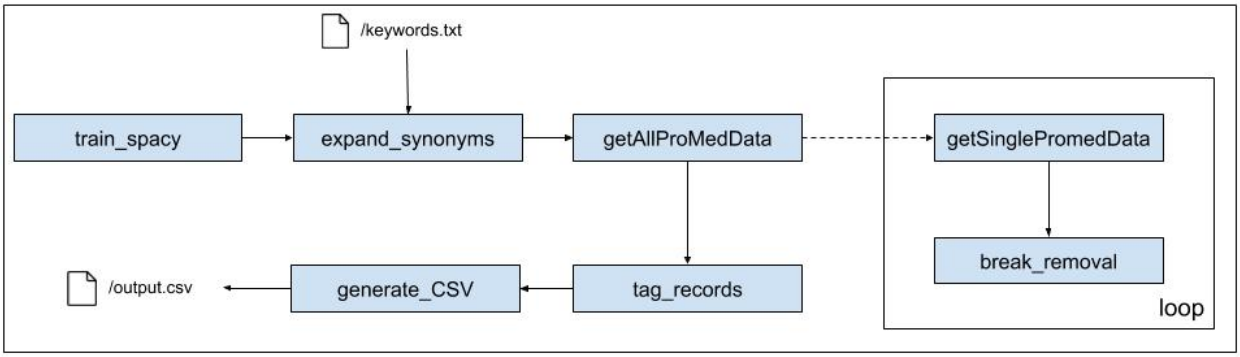

Figure 12. dataGatherer.py Workflow

The dataGatherer module can be invoked by calling: python3 dataGatherer.py. There are no parameters that need to be passed in. The dataGatherer module does not use every function listed above. The main functions that are left out are analyzeData, parseNormal, parseSpecial, generateRecords, and normalizeRecords. These methods are not included the main method of the module because there is not confidence in the output produced. While it has functionality, more work needs to be done on these methods to provide a more consistent output. All of these functions can be invoked by running analyzeData.

To run analyzeData, a file needs to be provided in the directory called key.py. Google Maps Geocoding API is used, which requires an API key. For security reasons, this team's API key is not displayed in the file for public use, so the separate file was created to store the key, then import that file to the dataGatherer module. The only line that needs to be in key.py is "API_KEY=[YOUR API KEY]". This will allow the dataGatherer module to use the Google Maps to access the API that allows for latitude and longitude lookup. To get an API key, go to https://developers.google.com/maps/documentation/javascript/get-api-key. Please note that there may be some payment required. The API used in this project has a fee of $5 per 1000 requests made. In testing, 100 requests were made. There are 1600 articles available with approximately 2 locations per article, needing about 3200 requests. We did not have to pay for usage of this API because we were able to use Google's free trial credit that was given to us when we signed up.

9. Lessons Learned

While there were some successes in this project, many obstacles and roadblocks were encountered; many focused on the natural language processing part. None of this team had any experience with NLP parsing and name entity recognition before, so this was a new topic. It was quickly discovered that English is a very complicated language with a lot of ambiguity. When looking through the articles, there were many different ways to say the same thing. One article could say “There are 20 new cases in week 10” where another article could say “In EW (epidemiological week) 10, the number of new cases was 20”. Both sentences mean the same thing, but that is expressed in different ways. This made it very difficult to make regular expressions to cover all the different ways to express something. It is something that we should have been considered beforehand, and then analyzed more articles to see the differences.

Another lesson, perhaps the most important, is the idea of intermediate steps. As the project progressed, the main goal was to try and deliver the finished product to our client by the end of the semester. However, this project, while it sounds simple, is quite complicated, and will take time to complete correctly and thoroughly. Originally, no intermediate steps were planned. The program would go directly from condensed articles to the final CSV output. This is not friendly to others who may want to use this data and it is inappropriate if the project is not complete at the end of the semester. So, intermediate steps were incorporated. These steps are to translate the data into a different form, slowly changing it from the normal text to the final output. This is very important because even if the final product our client wanted is not exactly produced, another team will be able to finish the work started, or someone can use the condensed form of the data for a different analysis. The main takeaway is handing in unfinished work that is usable by others is better than turning in bad, inaccurate, and unusable data as a final product.

9.1 Schedule

- 1/23: Reach out to client about meeting time to talk about project specs
- 1/28: Finalize project proposal
- 2/12: Data collection from one repository fully functional and a list of repositories to use
- 2/13: Initial Project Presentation -- proposal and deliverables
- 3/5: Prototype website with data from one online repository
- 3/31: Secondary Project Presentation -- work so far and future plans
- 4/1: Data collection from at least one more repository fully functional
- 4/14: Data collection from all repositories completed; website is being built and refined
- 4/28: Final Project Presentation -- work completed
- 4/30: User views finalized, data is downloadable
- 5/5: Reports and all other deliverables completed
- 5/6: Project Due

9.2 Problems

One of our goals in this project was to get data from multiple repositories. According to the client, ProMED had the most comprehensive data and was where data collection should be focused. There is a 3rd party website called HealthMap.org which displays data about outbreaks and diseases in an interactive map. This site collects a lot of data from ProMED and will distribute the data on request. As issues arose with data parsing through the ProMED articles, HealthMap seemed to be a useful alternative. Multiple attempts to contact the HealthMap team received no reply. Future work in this area could be to request the data at a time with less extenuating circumstances in order to retrieve comprehensive data from them.

The client expressed that he would like the data to be displayed visually on the website as well as be interactive. Specifically, he suggested a map that allowed the user to click on a country and have the information about that country's outbreaks as well as the downloadable data. As parsing issues arose, the focus of the project shifted to data collection rather than the interactive display of information. Data visualizations became a stretch goal in the project. This decision was made under the notion that the main goal of this project was to collect data and have it as downloadable for the user. Non-interactive visualizations of the data have been provided on the website, which is an alternative substitute for the interactive visualizations.

One final problem encountered was something out of our control. In March of 2020, the COVID-19 outbreak resulted in campus being closed down and classes being moved to an online format. Because campus was now closed, several of our team members went back home to social distance. This made working together difficult because there was upwards of a 14-hour time zone difference between members. Planning meetings and communicating between partners became much more difficult as someone may have questions for another member but are unable ask because it is 3 A.M. in the other time zone. It really tested the team's communication skills.

10. Acknowledgements

This section is to acknowledge our client, Dr. Luis Escobar.
Dr. Escobar is a member of the Fishing and Wildlife Conservation Department at Virginia Tech. He specializes in the distribution of biodiversity, including parasites and pathogens at global scales, and under past, current, and future environmental conditions. He looks into how climate change and land use conditions have affected the spread and seasons of outbreaks. Dr. Escobar's laboratory has worked to understand the temporal, spatial, and climatic pulses associated with cholera epidemics. Additionally, the laboratory has been a pioneer in disentangling the potential effects of climate change on this water-borne disease. A new challenge to understand and prevent this disease is to have a comprehensive assessment of the number of cases at the local level.

Dr. Escobar's contact information
Phone number: (540) 232-8454
Email: escobar1@vt.edu
Personal VT website: https://fishwild.vt.edu/faculty/escobar.html
Lab website: https://ecoguate2003.wixsite.com/escobar

11. References

[1] Doctorj, *googlemaps 1.0.2*, PyPI, Oct. 17, 2009. Available: https://pypi.org/project/googlemaps/1.0.2/ (accessed 25 April 2020).

[2] J. Geissinger, T. Long, J. Jung, J. Parent, and R. Rizzo, "Big Data Text Summarization - Hurricane Harvey," *VTechWorks*, Dec. 2018. http://hdl.handle.net/10919/86358 (accessed 25 April 2020).

[3] J. Hunter, D. Dale, E. Firing, M. Droettboom, *Matplotlib*, 2012. Available: https://matplotlib.org/index.html (accessed 25 April 2020).

[4] K. Reitz, *Requests: HTTP for Humans*, 2019. Available: https://requests.readthedocs.io/en/master/ (accessed 25 April 2020).

[5] L. Richardson, *Beautiful Soup Documentation*, 2020. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (accessed 25 April 2020).

[6] M. Honnibal, *spaCy Industrial Strength Natural Language Processing*, March 12, 2020. Available: https://spacy.io/ (accessed 25 April 2020).

[7] Numpy Developers, *Numpy*, 2005. Available: https://numpy.org/index.html (accessed 25 April 2020).

[8] *ProMED International Society for Infectious Disease*, 2020. Available: promedmail.org (accessed 25 April 2020).

[9] S. Bird, L. Tan, *NLTK 3.5 documentation*, Apr. 3, 2020. Available: nltk.org (accessed 25 April 2020).

[10] The Pandas Development Team, *Pandas*, "pandas-dev/pandas: Pandas 1.0.3". Zenodo, March 18, 2020. Available: https://pandas.pydata.org/ (accessed 25 April 2020).

[11] *World Health Organization*, 2020. Available: who.int (accessed 25 April 2020).

[12] Y. Martinez Palenzuela, *geotext 0.4.0*, PyPI, July 7, 2018. Available: https://pypi.org/project/geotext/ (accessed 25 April 2020).

12. Appendices

Appendix A: Regular Expression Patterns

Table 4. Regular Expression Patterns

Pattern Name	Regular Expression
Case Extractor	`(\d+\s\d*(?!Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Dec))+`
Cases	`((\d+\s*\d*)+\s(new|confirmed|suspected|reported|new\ssuspected)\s(cases|case))|((\d+\s*\d*)+\s(cases|case))|((\d+\s*\d*)+\s(new|confirmed|suspected|reported|new\ssuspected)\scholera\scases)|((\d+\s*\d*)+\s(\snew|confirmed|suspected|reported|new\ssuspected)*\scholera\s*\/\s*AWD\scases)|cases\s(\w\s*)*\sis\s(\d+\s*)+`
Cumulative	`(cumulative)*\stotal\sof\s((\d+\s*)+)\s(\w\s*)*cases|total(\snumber)?\sof(\scases)?\s(\w\s*)+(\d+\s*)+\scases|(cumulative\s)*total\snumber\sof\s(\w\s*)*\scases\s(\w\s*)*\sis\s(\d+\s*)+`
Date	`(\d+\s(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|June|July|August|Semptember|October|November|December)\s\d+)|(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|June|July|August|Semptember|October|November|December)(\s\d+)*\s\d+|(beginning\sof\s\d+)|during\s(\d+)`
Date (this week)	`this week|the week under review|this year|the week in review`
Deaths	`((\d+\s*\d*)+(new|confirmed|suspected|deported|new\ssuspected)\s(deaths|death))|((\d+\s*\d*)+\s(deaths|death))`
First Check	`(\S+\s\((\d+\s*)+\))|(\S+\s\(\d+\.\d+%;\s(\d+\s*)+\))`
Last Known Location	`\S+:`
Location	`((include|include;|includes:|including)\s(\w+,\s|and\s\w*)+)|\w*\s(region|district)`
Number (to find numerical values)	`\(\d+.\d+%;\s\d+\)|\(\d+\)`
Record Splitting	`\[\d|10|11|12\s\w+.+`
Second Check	`(\d+\s*)+\sin\s\w+`
Week	`(week(s*)\s\d+)`
Week Number	`(?<!\w)\d+`

Appendix B: Python Libraries

This is a list of Python Libraries that are required to run our program:

- argparse
- beautifulsoup
- csv
- datetime
- geotext
- googlemaps
- json
- key
- logging
- matplotlib
- mpl_toolkits.basemap (Linux only)
- nltk
- nltk.corpus
- numpy
- operator
- pandas
- pycountry
- random
- re
- requests
- spacy

Appendix C. Table of Figures

Table 5. Table of Figures

Appendix D. Table of Tables

Table 6. Table of Tables