CSC110 Project Report
Educational Crisis - A Closer Examination on the Correlations Between COVID-19 and School Closures Around the Globe

Shouyi (Ray) Hung, Yuhan (Charlotte) Chen, Mengyuan (Alyssa) Li, and Fuyang (Scott) Cui

December 10, 2021

Contents

1 Introduction

COVID-19 profoundly impacted students' learning environment and strategies. **Therefore, we are curious about how this global pandemic correlates with school closures all around the world as time passes, which is one of the main influencing factors that entirely changed our way of learning and living.** We will compare the levels of School Closure with the severity of COVID-19 of different countries in a specified time frame.

As a group of students, COVID-19 changed our way of learning from face-to-face to online for quite a long time. In 2021, results from the National Survey of Public Education's Response to COVID-19 had shown that more than 46%[1] of the students in the US across all grades are studying remotely. However, after COVID-19 eased a little bit, some of our schools reverted to the traditional in-person learning classes. A very prevalent issue amongst students is frequently switching between online and in-person learning. Under such circumstances, learning became increasingly hard as time passed.

Besides, as international students, we were energetic and excited about future university life. However, everything became harsh and unpredictable after the emergence of COVID-19. We are now bothered by expensive flight tickets, personal safety issues, and potential school closures as a result of the pandemic. For example, two of our group members are currently living in China because of COVID-19 and are troubled by the inconveniences of timezone differences.

Therefore, we aim to discover a general relationship between COVID-19 and school closures. With the help of the observed correlation, we could be more prepared in countering the impacts of COVID-19 as individuals. For example, we could reasonably predict the next virus outbreak based on our project and switch to online classes beforehand.

Additionally, from a broader scope, our project could provide intuitions to educational institutions about the trend of school closures and COVID-19 cases. Therefore, they could identify whether they made a correct decision of closing/opening schools during the pandemic, and draft plans to minimize the impacts in the future. In other words, we could utilize our project as a guide to help prevent future impacts that could rain onto the educational sectors that suffered during the current pandemic.

[1] National Survey of Public Education's Response to COVID-19 Infographic, https://www.air.org/sites/default/files/2021-07/infographic-results-national-covid-survey-june-2021_1.pdf

2 Dataset Description

We have identified two main datasets that will be relevant for our project's implementation.

These are:

1. Global School Closures for COVID-19[2] – Obtained from The Humanitarian Data Exchange, compiled by Saleh Ahmed Rony, sourced from UNESCO.
2. COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University[3] – Obtained from GitHub, compiled by JHU, sourced from WHO, ECDC, DXY, US CDC, etc.

Both datasets will be stored in a Comma Separated Value file, which will allow us to read from them easily through Python's csv library.

Furthermore, both datasets are very credible as they are sourced from multiple sites, including but not limited to WHO, ECDC, and US CDC. Furthermore, these datasets are also licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0), which allows us to utilize these data for our own needs.

The datasets that we have downloaded and utilized in this project were chosen because they were compiled in a way that allows easy access and manipulation. By using datasets that are already organized could improve the efficiency and robustness of our program.

The Global COVID-19 Dataset (Time series) has the following structure:

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	...
	Afghanistan	33.93911	67.70995	0	0	0	...
	Albania	41.1533	20.1683	0	0	0	...
...	...	...	...	...	...	...	...

The headers extend up until November 1, 2021 for the dataset that we will be using in our application.

Starting from column 5 and onwards (for the Global COVID-19 Dataset) contains the amount of cumulative cases for the specified country in the date shown in the header row.

The School Closure Dataset has the following structure:

Date	ISO	Country	Status	Note
17/02/2020	CHN	China	Partially open	
17/02/2020	MNG	Mongolia	Closed due to COVID-19	
...	...	...	...	...

The data is organized by entries of different country each day.

Some of the country names in our datasets contain characters that are not part of the standard ASCII table. An example would be ”Curaçao”, where ”ç” is a French (Latin script) letter.

Since these letters are not in the standard ASCII Table, displaying and encoding issues may arise. So for convenience, we will filter them out.

For the purpose of this project, we will simplify the standard ASCII Table to only contain characters from ”a-z”, ”A-Z”, ”0-9”, ”!-~”.

[2] Global School Closures COVID-19, https://data.humdata.org/dataset/global-school-closures-covid19
[3] One of the most used COVID-19 Cases Database, github.com/CSSEGISandData/COVID-19

3 Computational Overview

We divide our project into 3 main parts: data pre-processing, Graphical User Interface (GUI) implementation, and interactive data manipulations (filtering, aggregation, searching, etc).

3.1 Data Pre-Processing

3.1.1 Resource Initialization

In this part of the implementation, we have utilized `requests`, `os`, and `hashlib` to download required resources (data sets, icons, etc) and verify their completeness.

The `requests` library allows our application to get the contents of certain URLs. We used this library in combination with `os` to download and save those resources into the working directory, so our application can run with the specified data. Specifically, we used requests.get, which allows us to retrieve the information contained in the specified URL, and `os.makedirs`, which allows us to create directories to store our files in. Also, we set up a public GitHub repository to hold all of our codes and resources, so our application can visit those permanent raw links and download all of the resources.

We also implemented a MD5 Hash function[4] to calculate the unique identifiers of our resources. For those newly downloaded resources, our application will compare their hash values with the predefined valid hash values to ensure that the files downloaded are what we expected.

To put all of these together, we implement a module `resource_manager.py`, containing a `Resource` class and a bunch of other helper functions and constants. An instance of the `Resource` class represents a resource file of our application. Each resource instance contains a remote path, a unique identifier, and a local path so we can download and verify the files. Also, we build the whole resource manager in an extendable way: By maintaining a `dict` that maps resource name to the `Resource` instance, we could simply register, or add, a new resource to the `dict` if needed. Then, our application will automatically handle all downloading and file manipulating tasks without any further modifications.

3.1.2 Algorithms

To simplify our data manipulations, we abstract out few generic algorithms such as sorting, grouping, searching, etc. in a new module: `algorithms.py`.

All of the algorithms in `algorithms.py` are generic, meaning they could handle all data types with the help of some callable parameters. To be more specific, we used a generic type `T` to represent any type of arguments passed into our algorithms. However, not all data types support the operations an algorithm needed (such as compare operation in sorting algorithm). Thus, our algorithms need another callable parameter to specify how the operations are performed on the passed data type.

Additionally, we implemented multiple sorting algorithms for the purposes of showing the impact of algorithm running time complexity:

- Bubble Sort: $\mathcal{O}(n^2)$
- Selection Sort: $\mathcal{O}(n^2)$
- Insertion Sort: $\mathcal{O}(n^2)$
- Merge Sort: $\mathcal{O}(n \log(n))$

Upon the initialization of the application, the user should choose a sorting algorithm for the whole application and then initialize the data. This is achieved by assigning a new sorting function reference (defined in `algorithms.py`) to the project sorting function defined in `settings.py`. That is, our project will always call the sorting function defined in `settings.py` instead of the sorting functions in `algorithms.py`.

Also, we implemented some grouping and searching algorithms for data manipulations like grouping all data by a categorical variable or searching for a date in all the date values we have.

[4] Implementation referenced from StackOverFlow, https://stackoverflow.com/questions/3431825/generating-an-md5-checksum-of-a-file

3.1.3 Data Initialization

We read all the data into Python with help of the `csv` library in the module `data.py`.

For the COVID-19 Dataset, we only used the columns with the headers "Country" and "Dates", as they provide the information for us to visualize.

For the School Closure Dataset, we used all of the columns aside from the "Note" column, as it provides irrelevant and hard-to-process information.

Our program created the following data classes:

- `ClosureStatus` - An `Enum` class that maps the different status of school closure to an int value
- `Location` - A class that represents a location.
- `Country` - A class that represents a country, inherited from `Location`.
- `Province` - A class that represents a province, inherited from `Location`.
- `BaseData` - A basic class that represents a piece of data.
- `TimeBasedData` - A class that represents a time-based data, inherited from `BaseData`.
- `CovidCaseData` - A class that represents an entry in COVID data, inherited from `TimeBasedData`.
- `SchoolClosureData` - A class that represents an entry in Closure Data, inherited from `TimeBasedData`.

Our program generates the following GLOBAL CONSTANTS:

- `ALL_COVID_CASES` - A list of all `CovidCaseData` objects read from the data set, including country-wide and provincial data.
- `COUNTRIES_TO_COVID_CASES` - A dictionary mapping from a country name to all of its respective `CovidCaseData`.
- `GLOBAL_COVID_CASES` - A list of all `CovidCaseData` that contains COVID19 cases summed across all countries on a certain date.
- `ALL_SCHOOL_CLOSURES` - A list of all `SchoolClosureData` that is read from the data set.
- `COUNTRIES_TO_SCHOOL_CLOSURES` - A dictionary mapping from a country name to its respective `SchoolClosureData`.
- `GLOBAL_SCHOOL_CLOSURES` - A list of `SchoolClosureData` with only dates and the status of the majority of the schools in that date.
- `COUNTRIES` - A set of `Country` objects that are read from the data set.
- `SORTED_COUNTRIES` - A list of sorted `Country` objects by country name.

Then, we converted the raw data read from the data set into the classes specified above, so we could easily utilize them in practice. Also, by filtering, grouping, and sorting the data we have loaded into Python, we obtain the GLOBAL CONSTANTS defined above that could be utilized by other modules (`gui_main.py`, etc).

This brings us to the next part of our project, which is the GUI Implementation.

3.2 Graphical User Interface

We utilized `PyQt5` to generate an interactive graphical user interface and `Matplotlib` to plot graphs.

We have implemented the following functionalities:

- A fully functional Graphical User Interface with menu bar, multiple buttons, input date edits, sliders, etc.
- A progress bar on the status bar displaying the progress of data initialization.
- Panning and zooming functionalities on the graph by left click, drag, and scrolling.
- COVID-19 cumulative cases scatter plot and school closures visualizations.
- Customizable line color, style, and marker.
- And many more details...

3.2.1 Utilities

We create a module `gui_utils.py` that contains many helper functions, classes, and constants.

Basically, we inherit all PyQt widgets used to extend their functionalities:

- `StandardLabel`: A standard text label.
- `StandardPushButton`: A standard button.
- `StandardComboBox`: A standard combo box.
- `StandardDateEdit`: A standard date editor.
- `StandardCheckbox`: A standard checkbox.
- `StandardRadioButton`: A standard radio button.
- `StandardProgress Bar`: A standard progress bar.
- `StandardGroupBox`: A standard group box.
- `StandardMenuBar`: A standard menu bar.
- ...

3.2.2 Main Window

Our main window is defined in module `gui_main.py`, which consists of a "frontend" and a "backend".

The "frontend" is the `MainWindowUI` class that is responsible for creating widgets, initializing widgets' attributes, and managing the layout of all widgets. Also, it represents the "true" window itself and defines all the attributes of the window.

The "backend" is the combination of multiple classes including `PlotCanvas` and `MainWindow`. Technically, all the codes besides the "frontend" part are responsible for running our "backend".

The `PlotCanvas` class represents our plots and is responsible for handling all user's inputs to the plot such as mouse moving or dragging. We implemented the full zooming and panning functionalities by ourselves because we did not like how `matplotlib` handle them. Also, we constructed a cross-hair that is always centered on the data that are the closet to the cursor position.

All the user's inputs on the main window, such as clicking a button, selecting a country, or sliding a slider are handled by the `MainWindow` class. We utilized `PyQt5` signals and slots mechanism to achieve most of the functionalities. For example, when a button is clicked, then the button sends a signal to a callback function to perform some actions. The callback functions are the slots, and the signals are the information passed into the callback functions.

The interaction logics of our GUI is that our visualization (plot) will update as the user update the filters (a country or a data range).

3.3 Misc

Aside from the libraries mentioned, we also employed `sys`, `logging`, `time`, `typing`, and `ctypes`.

The libraries `sys` and `logging` are used to log the information, warnings, or errors in the console to inform the user what our application is doing.

The `time` library is used to evaluate the time that it takes for us to initialize the data and to output the time in the console log.

We used `typing` to specify the types of inputs that our functions will be taking and some generic types mentioned earlier.

Lastly, we used `ctypes` to specifically set an identifier for our application when running in Windows platform, so the icon will be displayed normally on the task bar when the application window is active.

4 Instructions

Before everything start, please make sure you have a stable connection with the Internet, specifically, GitHub.

1 Download the zipped file we provided on MarkUs. The zipped file should contain “”`algorithms.py`, `data.py`, `gui_main.py`, `gui_utils.py`, `main.py`, `resource_manager.py`, `settings.py`, and `requirements.txt`.”

2 Unzip the file in an empty directory (It is better to make sure the path to the directory only contains English letters).

3 Boot up PyCharm and open ”requirements.txt” in the directory you just unzipped the files in. Click ”Install requirements” to automatically download all the libraries needed.

   Alternatively, you could open the terminal in the directory and run “pip install -r requirements.txt.”

4 Either use PyCharm or a terminal to run ”main.py” and you should be able to see a window popup.

5 Select a ”Sorting algorithm” and click the initialize button. After few seconds, our application should be ready.

6 You should be able to set the start and end dates for the plot, pan and zoom on the plot, select countries and plot the result again, and utilize the menu bar for more customizable configurations.

7 Feel free to explore the application from this point onwards!

4.1 Notes

- We only tested our application in Windows 10/11 platform with Python 3.10.1. Although we were trying our best to make it compatible with others, we could not guarantee the behaviors under other operating systems.
- It can run on a M1 Mac with an extensive setup. (You need to run Terminal on Rosetta and install the x86_64 version of all libraries used through pip)
- We are unsure of the performance of our application in an Intel Mac.
- We haven’t tested our application on any Linux distributions.

4.2 Remedy

If our application failed to download all **required** resources, please download this compressed file and unzip it in the same directory as the `main.py`. Make sure that the `resources` folder is in the same directory as the `main.py`. TODO: Add the link before we submit.

Or you could clone our repository and everything will be right in the correct place. TODO: Add github.

5 Changes from the Proposal

- Added MLA Style Footnote in multiple occasions.
- Added more details in Computational Overview.
- Added features to download dataset and write in working directories.
- Added features to hash a file in MD5 to ensure its completeness.
- Removed some unused references and added new ones.
- Removed the US Covid Dataset as it is unnecessary for our analysis purposes.

6 Discussion

6.1 Results and Conclusion

Our application can be utilized to visually determine if there are any correlations between COVID-19 Cases of a certain country and School Closure status of the certain country. This would be enough for us to answer the research question, because we can clearly identify a trend between COVID-19 Cases and School Closure statuses.

Furthermore, we can also utilize this application to generate plots about data around the world that can be shared to raise awareness, which could allow educational sectors to be more prepared for future events similar to the COVID-19 Pandemic.

We can also identify some of the "Outliers" from the general trend for some countries. These may exist because their country officials decided for schools to continue operating amidst the severe conditions posed.

6.2 Limitations

- A major limitation of our application is the fact that our data is not a 100% accurate data from all around the globe. School Closure Data being used is uploaded by an individual. The COVID-19 Data may also contain inaccurate information.
- Furthermore, we recklessly removed some of the country names in our datasets because they contain non-ASCII characters, which limits the amount of data.
- We also removed some country names because they were not present in both datasets, making the actual data presented in our application even more limited.
- We did not introduce the vaccination data and this impacts our scope of analysis. That is, most countries resume in-person learning after most individuals were vaccinated.

6.3 Obstacles

- Our application crashes all day, all night, 24 * 7, non-stop. Making us want to drop out of CSC110. But nevertheless, we still overcame the difficulties and produced a "Functioning" application.
- One of the most impressive challenges is the implementation of progress bar.

  If we want to display a progress bar monitoring the progress of something like initialization, we need three tasks to be performed simultaneously, or asynchronously:

  1. The initialization process itself.
  2. Display the current progress (GUI).
  3. Continuously monitor the current progress and send the progress to the GUI to display.

  Therefore, we need 2 more threads, in addition to the main GUI thread, to make the progress bar work as intended. But here comes the problems, we don't know anything about threading and specifically, threading with `PyQt5`.

  Then, we searched on the Internet, read the full documentation of `PyQt5` which is in C++, study others' examples, and experiment tons of times.

  We firstly used `threading` module to run the progress bar on another window on another thread, but after many crashes without error message, we found that `PyQt5` doesn't like the Python native `threading` module. After running `QApplication`, any operations from `threading` module will block the main thread, which is totally not what we expected.

  Then, we switched to `QThread` which is preferred by `PyQt5`. However, another problem raised that as we know, `PyQt5` is thread-safe which means that it doesn't allow any modifications from another `QThread`, so we could not simply run initialization on another `QThread` and directly report the progress to the `PyQt5`.

  When we nearly gave up implementing the progress bar, the angels came. We remembered the `PyQt5` had the signals and slots mechanism (we explained earlier), and we could actually create a new signal in our monitor thread and send this signal to the main gui thread every time the progress updated. In this way, we utilized the built-in mechanism of `PyQt5` and prevented any thread-unsafe operations.

  Retrospectively, because we learned many valuable knowledge from this challenge and now we know how to tackle it, the progress bar is not very difficult. However, although the progress bar itself is trivial, the processes involved in overcoming the challenge are pivotal to us.

- The other challenge is about optimizations.

  In our plot, we implemented a cross-hair that centered on the closet point in the line from the cursor. We successfully implemented it with binary search and many data transformations and calculations. However, the frame rate per second (FPS) was very low when moving the cross-hair.

  To investigate the root of the problem, we analyzed the running time of our algorithms but it turned out that the running time was good ($\mathcal{O}(n)$).

  Then, we used `time` module and divided the codes into many piece of codes to calculate the real times each piece took. Finally, we found that the root of the evil is actually the `matplotlib#Canvas.draw()` function call. But here comes the issues, we cannot modify the code from `matplotlib` to make it faster.

  After many researches, we found out that we could solve this issue with a relatively advanced method: blit. Since we already had many experiences from the previous challenge, by reading through the documentation, experimenting with many small examples, and updating the whole implementation of `PlotCanvas`, we successfully implement blit to fix the issue.

  We firstly draw the figure, excluding the cross-hair, and save the figure background to cache it. Then, every time the cursor moves, we only draw the cross-hair and restore the figure from cache. In this way, the previous cross-hair disappears because the figure background does not contain the cross-hair - we draw the figure before the cross-hair, and we don't need to redraw the background frequently.

  This largely improve the FPS and now the cross-hair is very smooth.

6.4 The Future

- We will be carrying the skills we developed in this project into other projects in the future. We have now acquired the ability to download, process, and hash data. These abilities would be necessary to create many more applications in the future.
- We also acquired one of the most important skills in the Computer Science Industry, which is to design a Graphical User Interface. This is the ultimate version of "Designer to User" contract, which provides a very "Easy to use" interface so that users without any knowledge in Computer Science can also use this application to their advantage.
- This project will be held public on GitHub repository for anyone to fork and create an updated version of the application. We have included docstrings and comments everywhere around our code.
- We will also claim copyright, specifically MIT License 2.0 in the GitHub repository, so no harm will be done to the University of Toronto's CSC110Y1 Fall's instructing team.

7 References

"Coronavirus." World Health Organization, WHO, 10 Jan. 2020, www.who.int/health-topics/coronavirus#tab=tab_1.

"CSV File Reading and Writing — Python 3.10.0 Documentation." Python Documentation, docs.python.org/3/library/csv.html. Accessed 30 Oct. 2021.

Johns Hopkins University. "GitHub - CSSEGISandData/COVID-19: Novel Coronavirus (COVID-19) Cases, Provided by JHU CSSE." GitHub, CSSEGISandData, github.com/CSSEGISandData/COVID-19. Accessed 30 Oct. 2021.

Kaggle — Global School Closures for COVID-19. Saleh Ahmed Rony, www.kaggle.com/salehahmedrony/global-school-closures-covid19. Accessed 30 Oct. 2021.

https://stackoverflow.com/questions/3431825/generating-an-md5-checksum-of-a-file