Project 1

Group 1

March 1st, 2021

# Introduction

# Methods

The first step in analyzing the two conditions was to extract the required data from ClinicalTrials.gov. Of the two options (downloading the data or parsing AACT), our group decided the most efficient method for analyzing the dataset was to extract the data directly from ClinicalTrials.gov. On the site, there are multiple options for exporting data. Most of our analyses could be completed by downloading the table contents in a 1 row per study xml file. However, for more advanced portions of the project, such as querying adverse event data, we needed to download the full study XML records. This option exports a zip file containing a folder storing every trial in a separate XML file. For basic search operations this step is unnecessary, but for more in depth analyses this method is required.

Once the data was exported, the next step was to parse the files and extract all relevant information. We decided on using Python for our parsing language. Rather than parsing the XML contents directly, we decided on converting the data into a JSON file. This method was an extra step, but it allowed for easier data parsing which saved both time and effort. Utilizing Python’s json, os, and xmltodict libraries, the downloaded XML file was converted to JSON with a simple function.

## Section A:

With the data stored in an efficient format for parsing, we began extracting relevant data for part A. The dataset that was returned by ClinicalTrials.gov when searching by both ‘COVID-19’ and ‘Hepatitis A’ provided trials whose conditions did not include the corresponding diseases. Because of this, an extra data parsing step had to be implemented to analyze only trials with Hepatitis A or COVID-19 listed in the condition section.

Using the Python json library, we loaded the data file into our code. In total, 13 functions were created and utilized to extract the relevant data as outlined in the project specs file for part A. Our main function loaded the JSON file into a dictionary and called the remaining functions: getStudyDuration, getStudyResults, getAgeGroups, getMinMaxAge, getGender, getStudyType, getInterventionStatus, getActivityStatus, getStudyDesignData, getPhaseData, getLocationCount, and getEnrollmentCount. Each function had a single parameter input of the data object dictionary and returned a dictionary with the relevant extracted data. The general format of each procedure was as follows

1. Create a dictionary for storing the information.
2. Iterate over the dataset.
3. Extract the desired data from the current study.
4. If that item was not stored in the new dictionary, add the item as a key with a value of 1
5. If that item was stored in the new dictionary, add one to the value stored at the matching key.
6. Return the dictionary storing all the keys and corresponding value representing the data found in each study for a given procedure and the frequency at which they occurred.

Some of the functions required more advanced parsing methods.

* getLocationCount extracted data regarding the locations for each study. This data was stored in the dictionary object under the key ‘locations’. Some files had no location information available, and thus we had to filter through studies to add those with a NoneType value for the locations key to the dictionary key counting studies with 0 locations. For the remaining studies, locations could either be stored as a Unicode string or a list of Unicode strings. Those with a list had to be iterated over again to count the total number of locations found. Those with a Unicode string represented studies with only one location provided.
* getPhaseData extracted data regarding the phase of each study. Phases were either stored as a NoneType, meaning no data was available, a Unicode type meaning one phases was listed, or as a list which represented a trial with multiple phases provided. The NoneType and Unicode studies were simply added to the dictionary to count frequency. The list types were converted to a string joining the two phases together. For instance, [‘Phase 1’,’Phase 2’] was converted to ‘Phase 1/Phase 2’. Once converted, these objects were added to the dictionary containing the phase data.
* getStudyDesignData

For each function returning a dictionary, the information then had to be saved to a CSV file to allow for diagram creation and more in-depth analysis. This required the use of the Python library, CSV.

# Results

## Section A:

Phase of Study

Activity Status

Type of Trial

Eligibility Criteria

Trial Results

Design Aspects

Study Duration

Enrollment

Locations

Limitations

Conclusions