**Section 2: Implementation Details**

This project has two code files: The executable Jupyter notebook file and a query named pm\_query where the functions called in the Jupyter notebook file are stored. Breaking the code into functions this way greatly improves the readability of the code and allows specific functions to be used multiple times for different purposes

**Modules**

The following modules must be imported to run these user-created functions:

yaml: YAML is a flexible, data-oriented serialization standard. It is used here only for the secret manager function used to by our team member to store passwords their personal API key.

json: JSON is an abbreviation for JavaScript object notation syntax that is often used in data interchange. It is used here to convert data gathered in the function get\_data from a python dictionary to a JSON file and then into a csv file in the first part of this project.

os: The python os module provides miscellaneous functions for interfacing with the operating system. It is used here only to check if a current saved file already exists. This allows for the user to optimize disk space and time by not creating redundant copies of a large data file.

sqlalchemy: This is one of many database modules for python. We use this specific module for the create engine function and compatibility with sqlite dialect.

pandas: This is the primary dataframe module used throughout this project, allowing users to manipulate the structure of the data and create summary statistics.

plotly.express: This is the graphing library used for the data visualization portion of this project

Also, the Entrez package must be imported from biopython. This is the text search and retrieval package necessary for the operation of the data scapper in this project.

**Part1**

In part 1 of this project, we are tasked with creating a scrapper module that can collect the paper title, author list publication time and abstract from pubmed for keyword HIV within a pre-specified time window of 1/1/2020-8/30/2020. The retrieved data must be saved in CSV format.

The code begins with the initial parameters of the scrapper module, with the email address of the team member who created the code. The function pq.secret\_manager reads a yaml file containing the passwords and api keys necessary for running this module without hardcoding them into the python script. The search term 'HIV' is assigned the variable name 'search'. The personal API key from NCBI allows 10 queries per second as opposed to the default of three.

**Get\_Pmid Function**

The Entrez utility package allows efficient access of NCIB data over the web. Using the Entrez search term, the get\_pmid function queries the eSearch endpoint of the Entrez api to retrieve the corresponding pmids and join them to the input dataframe.

We begin with an empty list called for\_efetch, where will store the pmids initially. Next, we calculate the total number of records that need to be retrieved that match the search term between the max and min dates specified in the assignment. In this example, we are searching for articles published on the subject ‘HIV’ between 1/1/2020 and 8/30/2020, but the function is written generally to allow flexibility and reuse for later queries.

Since the default for the retmax parameter is 20 UIDs, we will need to adjust this parameter to retrieve the amount of data our scrapper needs to collect. At the same time, we want the program to run efficiently and only run the number of iterations required for our specific search. Therefore, the function calculates the number of pmids that meet our search requirements and uses this number as the value of the retmax parameter.

After we have the total number of records, the data scrapper gets all pmids that fit our search criteria and appends them into our list for\_efetch. Finally, the output for\_efetch is changed from a list into a one dimensional array using pandas.series and our input dataframe has been set up.

**Get\_Data Function**

Using the pmids retrieved in the get\_pmids function, the get\_data function queries the eFetch endpoint to retrieve the details for the corresponding citation as a list of dictionaries. This function also uses the python time sleep function to add 60 seconds of delay after every 600 iterations of the for loop and prints the total number of records retrieved, allowing the user to periodically check the scrapper’s status without interfering too much with performance and execution time.

The data gathered is then converted from a python dictionary into a JSON-encoded object and saved as hiv\_records.json

**Clean\_Data Function**

This function uses a list of dictionaries (that contains all previously gathered citation data for the dataset), on a per citation basis, it extracts the specified information about each record.

First, the data is sorted by whether the retrieved record is an article or a book publication and the record is then cleaned according to the corresponding Entrez keywords for the data type using a series of if/else arguments. Author’s name is similarly cleaned and standardized using if/else arguments depending on whether the author is an individual or a company and if either first or last name is missing before storing to the corresponding dictionary in the standardized format.

The extracted information is saved as a list of dictionaries which is then converted to a Pandas dataframe with labeled columns for pmid, title, abstract, dates, and author(s). The data is entered into the dataframe after the first iteration of the loop using the pandas.concat function.

**Keep\_Cleaning Function**

Using the pandas dataframe from the previous function, the keep\_cleaning function performs additional cleaning on the data by resetting the index of the dataframe, converting the pmid variable to an integer data type and formatting the dates into the %Y-%m-%d’ format and the columns for title and abstract are joined by index. Standardizing the datetime format this way not only will lead to increased legibility of our results, but will make it easier to perform the calculations by month in part 3 of this project. We now have our completed and cleaned dataframe.

**File\_Downloader Function**

This function performs the final step in part one of the assignment by converting the newly created data frame into a csv file. The file downloader function also takes the additional step to determine whether we have a current, existing file of this same name and file path.

**Part 2**

In this part of the assignment, we are asked to create a database module that can import the CSV file to SQLite, build database automatically, and then implement SQL code to query the publication by author’s name

**Csv\_bnb Function**

The csv\_bnb function reads the csv file created by the data crawler and reads it using the pandas read\_csv function. This data is then reformatted for use with sqlite and saved as a new csv file called hiv\_csv.

**Sqlite\_out Function**

The sqlite\_out function then takes the file hiv\_csv and uses the create engine function included in sqlalchemy to automatically build a database from the aforementioned file, specifying sqlite as the database dialect.

**Sql\_author\_query Function**

Using similar syntax and commands as before, we use a similar create engine function included in sqlalchemy to automatically build a database, specifying sqlite as the database dialect. In this case, we want to restrict results to those with a name similar to the input name, which is achieved using the pandas read\_sql function to search to column author(s) name for entries similar to what the user specifies in the function.

Given the amount of data in our dataframe, we only return the first 10 results from the SQL author query using the head function.

**Part3**

In this portion of the project, we are tasked with creating a visualization module that can read the CSV file, display the number of publications in each month, and visualize the trend of the publication numbers over time We are then asked to generate and visualize the summary statistics of the publication numbers per month.

**Sql\_draw\_graph Function**

This function creates the graphs where the user can display number of publications in each month as a bar graph, visualize the trend of the publications over time as a line graph or view both simultaneously as the line graph overlays the bar graph.

First, a copy of the dataframe is created, which is always a good idea when manipulating data, as it ensures we have an unaltered version of the original dataframe in the event that an error occurs. Next, the publication date is changed into the categorical variable “months” using the pandas to\_datetime and CategorigalIndex functions. In each of the three available graph options, the x-axis will display the month of publication and the y axis will display the number of publications. Each graph is designated with a title as well. The legend is removed from the graph that displays both the line and bar graph for legibility and appearance.

**Summary\_stats Function**

This function, as the name suggests, creates and displays the summary statistics by month. Similar to in the draw\_graph function, we begin by making a copy of the dataframe and use the pandas to\_datetime function to create a categorical month variable. Next, the number of publications per month are summed using the pandas value\_counts function. Using the distribution created by this function, we create summary statistics using the pandas function describe and then sending these results to the data frame.

Before displaying the summary statistics, the count of publications is dropped from the data frame. The remaining fields are displayed when called in this function by specified month.