**Section 4: User Manual/Guide**

**Description**

This program contains scraper, database, and visualization modules meant to assist the user with various tasks. The tasks of each included module are:

1. Scraper: retrieving paper title, author list, publication time, and abstract from PUBMED for a given keyword within a pre-specified time window and saving the retrieved data in the CSV format.
2. Database: importing the CSV file to SQLite to build a database automatically and implementing SQL code to query publications by author name.
3. Visualization: reading the CSV file, showing the number of publications in each month, generating and visualizing the summary statistics for the publication numbers per month, and visualizing the trend of the publication numbers over time.

**Prerequisites**

Load up the prerequisites into your Python environment. The code shown below is meant to be added into a Python file or run in a Jupyter Notebook cell. **pm\_query** is the program query where the necessary module functions are stored. This must be imported to use any module functions.

**import** **yaml**

**import** **json**

**import** **os**

**import** **sqlalchemy** **as** **sql**

**import** **pandas** **as** **pd**

**import** **plotly.express** **as** **px**

**import** **pm\_query** **as** **pq**

**from** **Bio** **import** Entrez

**pd.options.mode.chained\_assignment** **= None** *# Stop set copy on slice warning*

Install any other missing prerequisite modules or packages. These are listed, with versions specified, in **requirements.txt**

An empty yaml file **apikeys\_project.yaml** has been provided. Use this to input your NCBI personal API keys and passwords. This is necessary to maximize the efficiency of the scraper module. Use of the yaml file also allows the user to avoid having to hardcode personal data into the Python script.

**Documentation**

**pq.secret\_manager(yaml\_file)**

Reads a yaml file containing the passwords and API keys of the user.

Parameters:

* **yaml\_file**: the string path leading to or the string name of the yaml file containing your NCBI API keys and passwords.

Example:

keys = pq.secret\_manager("api\_keys\_file.yaml")

**pq.get\_pmid(contact, key, term, \*\*dates)**

Queries the eSearch endpoint of the Entrez API to retrieve the corresponding pmids and join them to the input dataframe.

Parameters:

* **contact**: the string email address used for the pmid query.
* **key**: the yaml file containing your NCBI API keys and passwords.
* **term**: the string search term used for the query.
* **\*\*dates**: input your desired time window as **mindate** (start date) and **maxdate** (end date) using the yyyy/mm/dd format.

Example:

email = "rachit.sabharwal@uth.tmc.edu"

search = "HIV"

hiv\_pmids = pq.get\_pmid(contact=email, key=keys["apikeys"]["ncbikey"]["key"], term=search, mindate="2020/01/01", maxdate="2020/09/01")

**pq.get\_data(pmid\_list, contact, key)**

Queries the eFetch endpoint of the Entrez API to retrieve the details for the corresponding citation as a list of dictionaries.

Parameters:

* **pmid\_list**: the retrieved pmids from the get\_pmid function.
* **contact**: the string email address used for the pmid query.
* **key**: the yaml file containing your NCBI API keys and passwords.

Example:

hiv\_records = pq.get\_data(pmid\_list=hiv\_pmids, contact=email, key=keys["apikeys"]["ncbikey"]["key"])

**pq.clean\_data(records)**

Using a list of dictionaries, on a per citation basis, this function extracts the following information about the citations where possible: title, abstract, date, authors. The extracted information is saved as a list which is then converted into a dataframe.

Parameters:

* **records**: a list of dictionaries.

Example:

hiv\_clean = pq.clean\_data(hiv\_records)

**pq.keep\_cleaning(df)**

Used in coordination with the **clean\_data** function, this function performs additional cleaning on the data by resetting the index of the dataframe, converting the pmid variable to an integer data type, formatting the dates into the %Y-%m-%d’ format, and joining the columns for title and abstract by index.

Parameters:

* **df**: the converted dataframe following use of the **clean\_data** function.

Example:

hiv\_clean = pq.keep\_cleaning(hiv\_clean)

**pq.file\_downloader(file\_name, df)**

Converts the information from the cleaned dataframe following use of **clean\_data** and **keep\_cleaning** functions into CSV format.

Parameters:

* **file\_name**: desired string name of the dataframe converted into CSV format.
* **df**: the cleaned dataframe following use of **clean\_data** and **keep\_cleaning** functions.

Example:

pq.file\_downloader("hiv\_csv\_clean.csv", hiv\_clean)

**pq.csv\_bnb(file\_path)**

The function reads the CSV file created by the data crawler via the **pandas** **read\_csv** function. This data is then reformatted, converting pmids to int type, dates to datetime type, and unlisting author names to prevent nested lists in the dataframe. The function ultimately returns a cleaned dataframe.

Parameters:

* **file\_path**: the string path leading to or the string name of the CSV file created by the data crawler **file\_downloader** function.

Example:

hiv\_csv = pq.csv\_bnb("hiv\_csv\_clean.csv")

**pq.sqlite\_out(df\_name, db\_name, tbl\_name)**

The function is used to take an input dataframe and two arguments, a database name and a table name, with SQLite established as the database dialect. This function ultimately involves two checks, first is a database check and second is a table check.

At the first check, the function will check for an existing database with the specified database name parameter. If a database with the name exists, then the function will proceed to check for the specified table name parameter. If the table exists in the database, then the function will replace it. If the table does not exist, then the function will create a new table with the specified name.

If a database with the name does not already exist, however, then the function will create a new database with the specified name and then create the specified table within the newly created database.

Parameters:

* **df\_name**: the input dataframe.
* **db\_name**: the specified string database name.
* **tbl\_name**: the specified string table name.

Example:

pq.sqlite\_out(hiv\_csv, "HIV\_Records", "PubMed\_Query")

**pq.sql\_author\_query(author\_name, db\_name, tbl\_name)**

This function is used for an author query of a specified database and table, restricting results to those with a similar author name within a specified database and table by using the **pandas read\_sql** function. This function does not create a new database or table; it queries based on already-existing parameters.

Parameters:

* **author\_name**: the desired string author name.
* **db\_name**: the specified string database name.
* **tbl\_name**: the specified string table name.

Example:

sql\_df = pq.sql\_author\_query("Mary", "HIV\_Records", "PubMed\_Query")

**pq.draw\_graph(df, graph\_type)**

To either 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, call on this function. The created graph is interactive.

Parameters:

* **df**: the dataframe containing the data being visualized.
* **graph\_type**: an optional string used to specify the desired graph type. The default is “line”, but users can input “bar” or “both” to change the graph type.

Example:

pq.draw\_graph(hiv\_csv, ”both”)

**pq.summary\_stats(df, calendar\_month)**

Creates and displays the summary statistics by month.

Parameters:

* **df**: the dataframe containing the data being visualized.
* **calendar\_month**: the string used to specify the desired month for the function. This parameter is case-insensitive.

Example:

summary\_stats = pq.summary\_stats(hiv\_csv, "january")