PH 1975, Fall 2020: Capstone Project

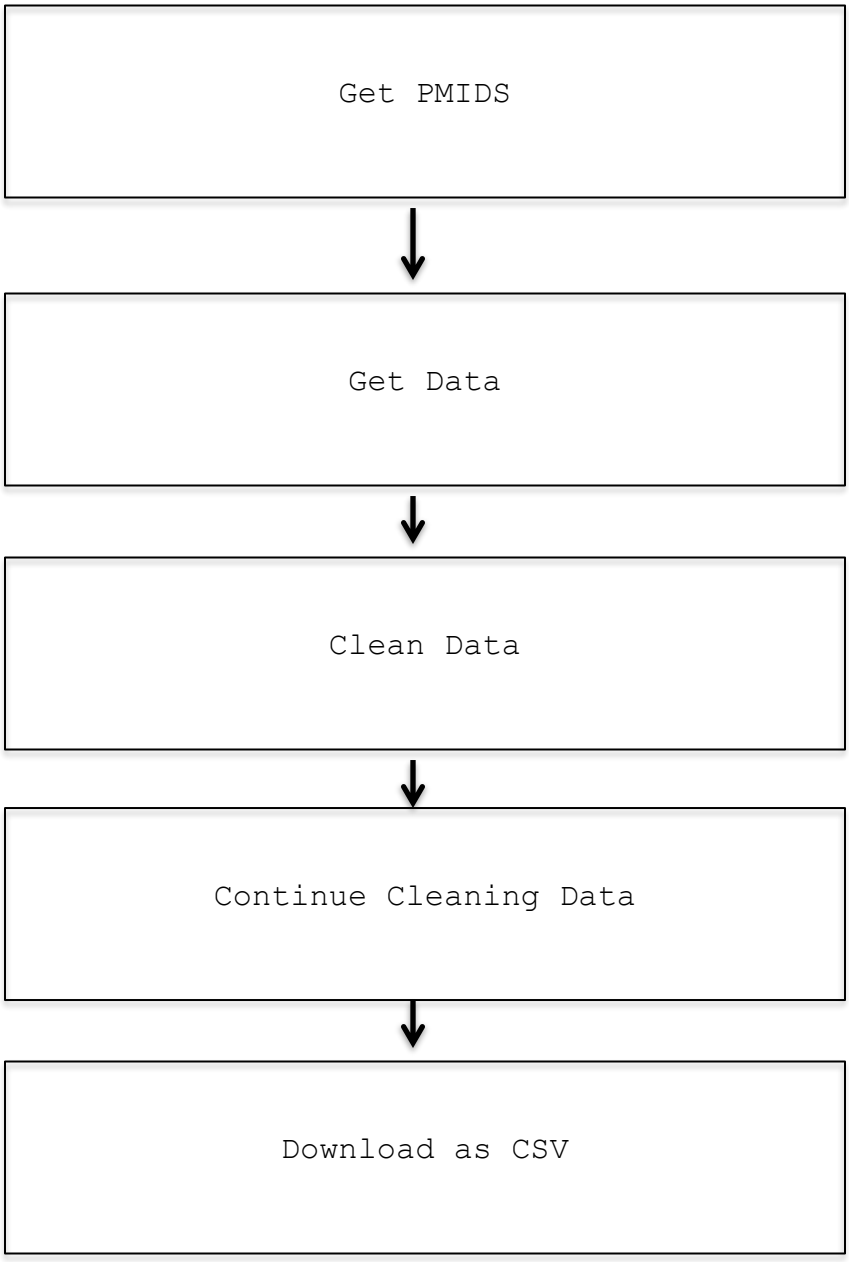
By Julie Liu, Lee Minnetti & Rachit Sabharwal

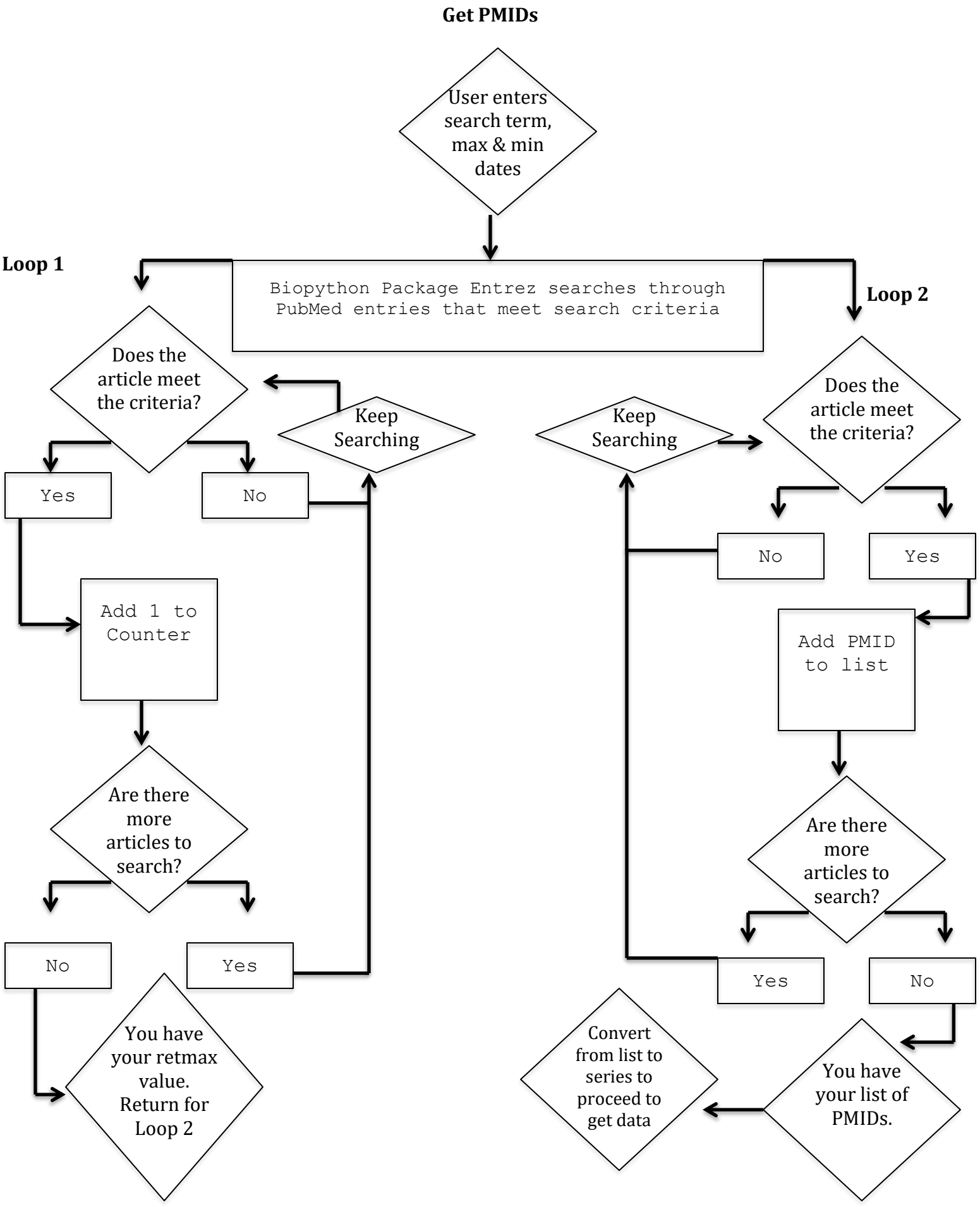
Section 1: Program Design

This project is divided into three modules: A data scraper module, a SQL module and a data visualization model. The following pages contain the workflow diagrams detailing the structure and function of these modules.

Data Scraper

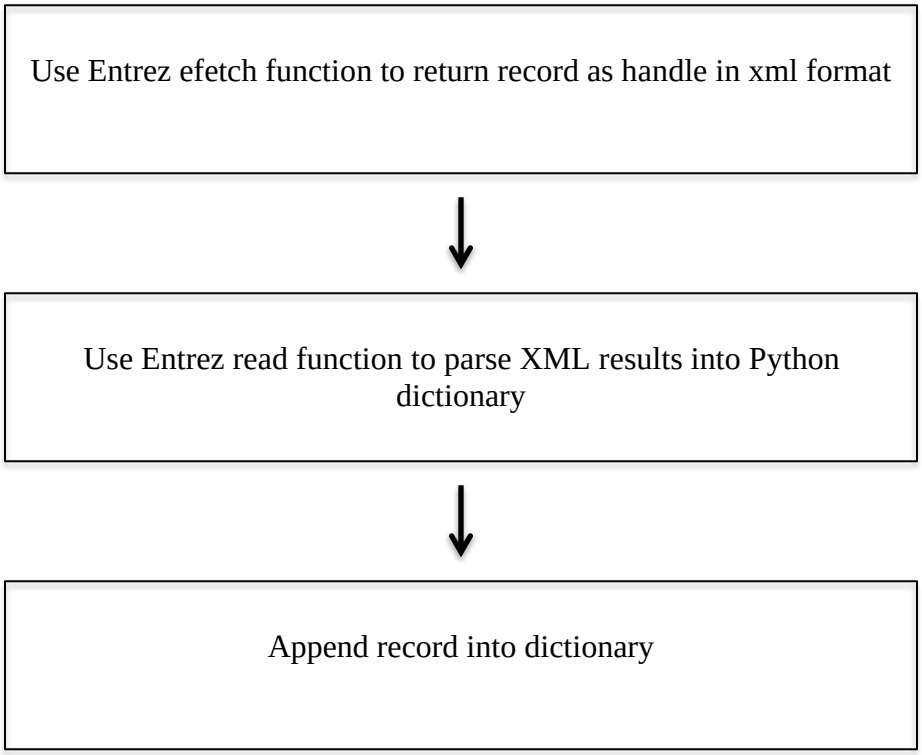
These are the five main functions performed in the scraper module, each of which has it's own detailed workflow diagram in the subsequent pages:




Get PMIDs
User enters search term, max & min dates
Loop 1
Biopython Package Entrez searches through PubMed entries that meet search criteria
Loop 2
Does the article meet the criteria?
Keep Searching
Yes
No
Add 1 to Counter
Are there more articles to search?
No
Yes
You have your retmax value. Return for Loop 2
Keep Searching
Does the article meet the criteria?
No
Yes
Add PMID to list
Are there more articles to search?
Yes
No
You have your list of PMIDs.
Convert from list to series to proceed to get data

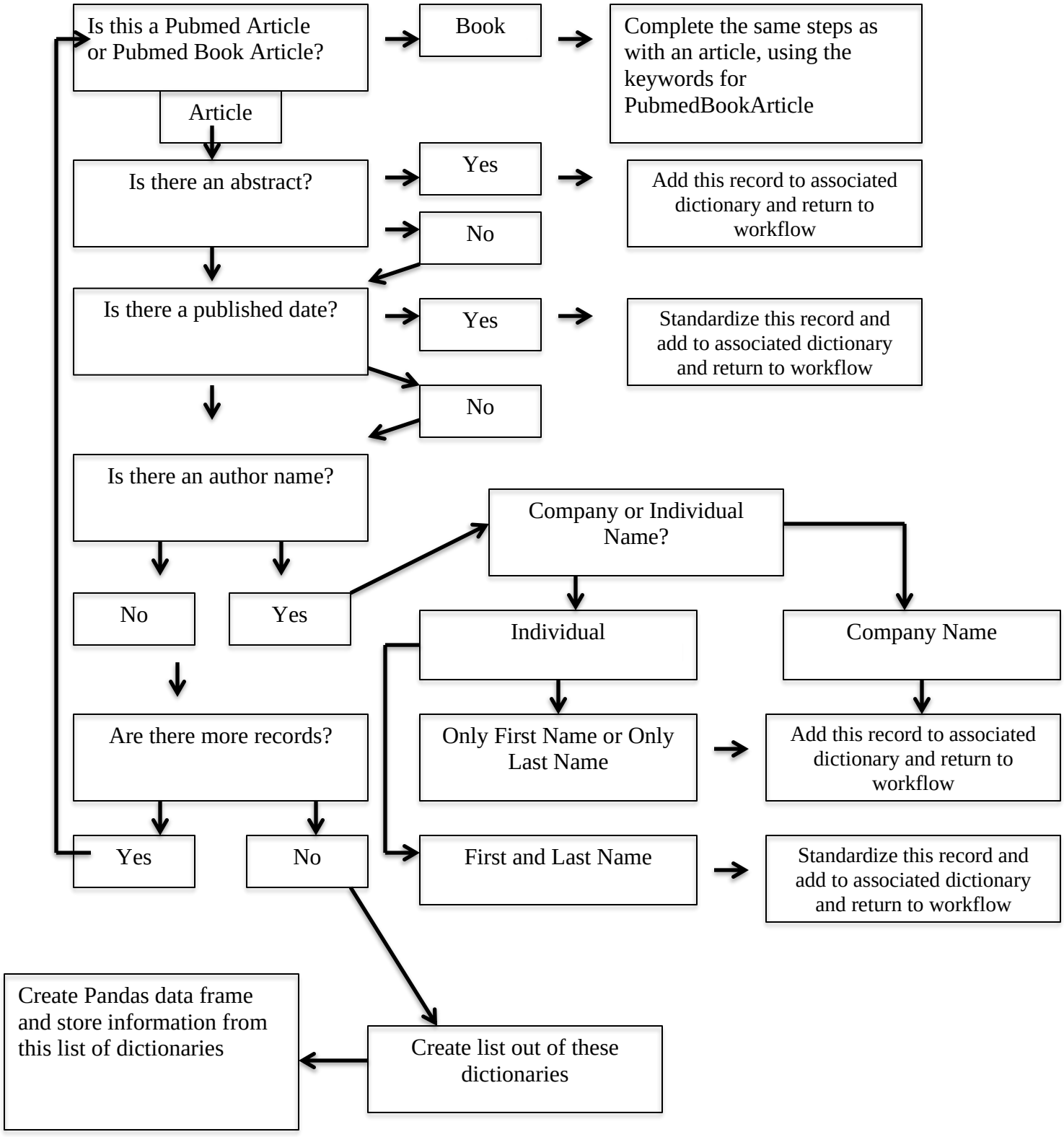
Get Data

For each PMID in series created in previous step:



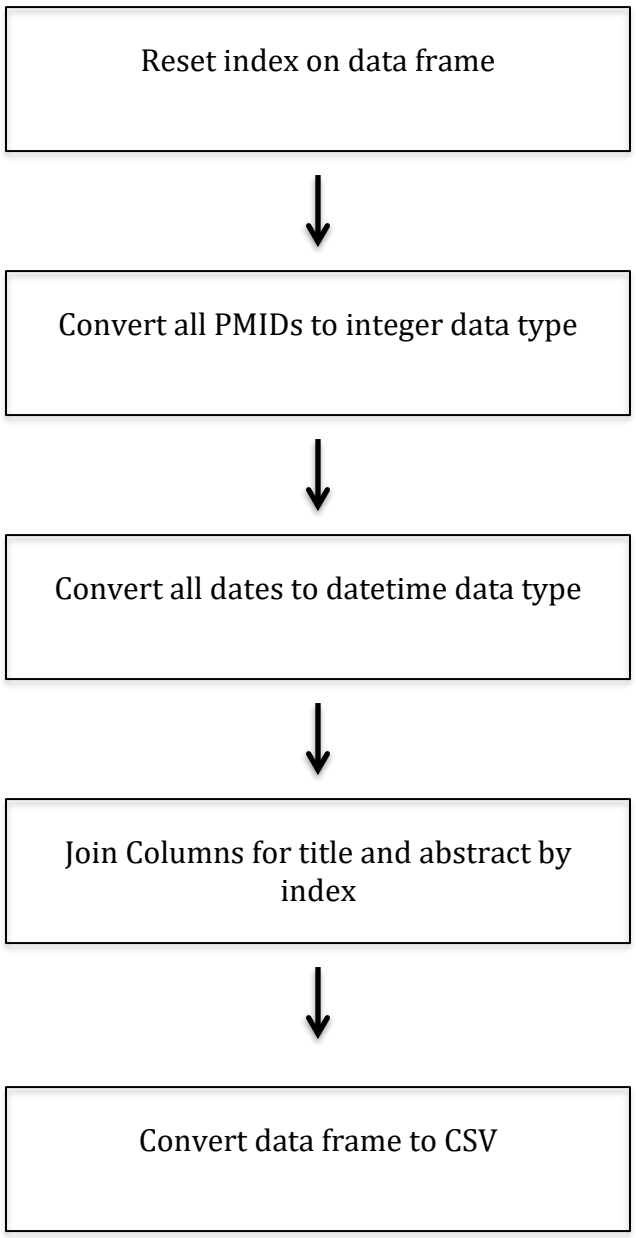
Clean Data

This function uses a list of dictionaries to store all citation data for the dataset. The following is performed for each previously retrieved record:

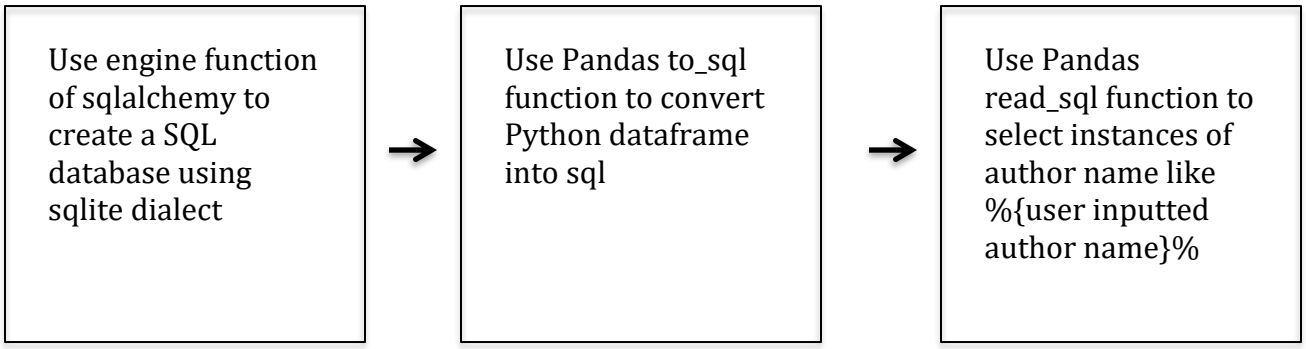


Continue Cleaning Data & Convert to CSV

Begin with dataframe from previous workflow:

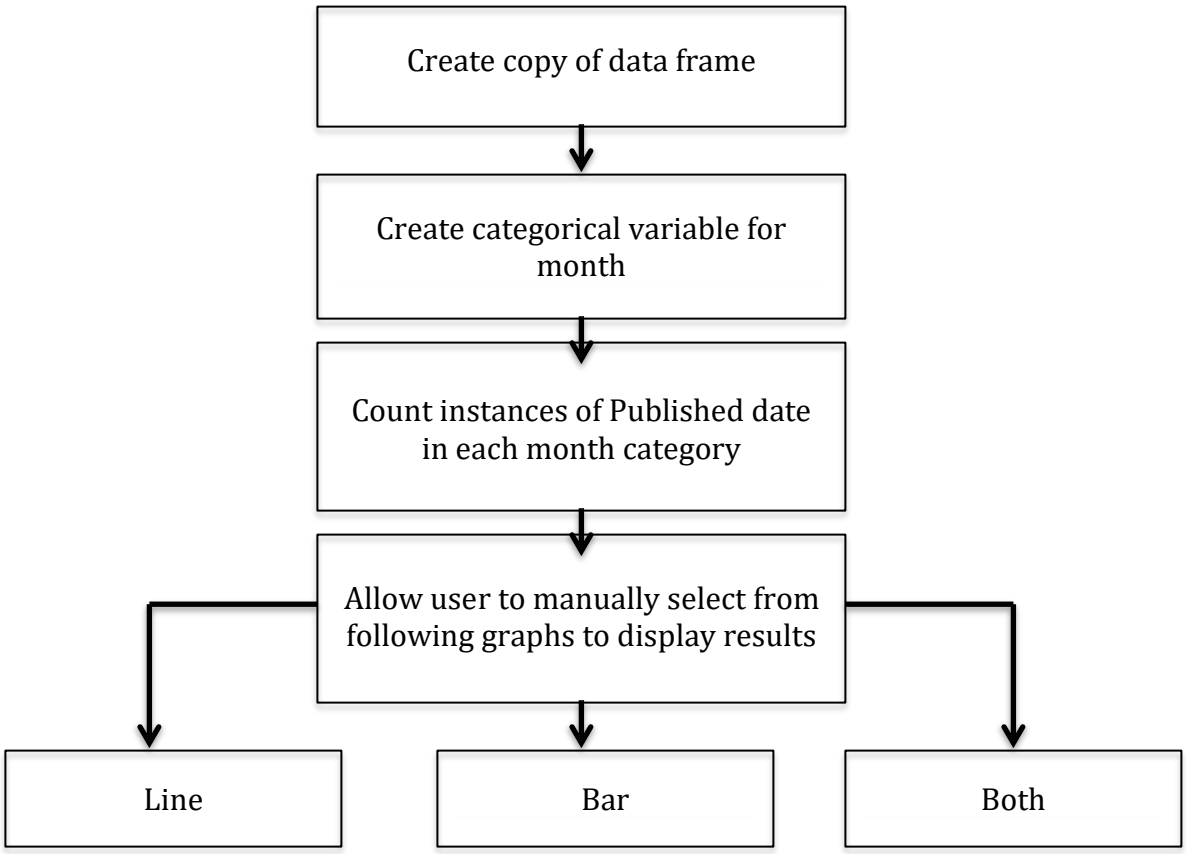


SQL Module

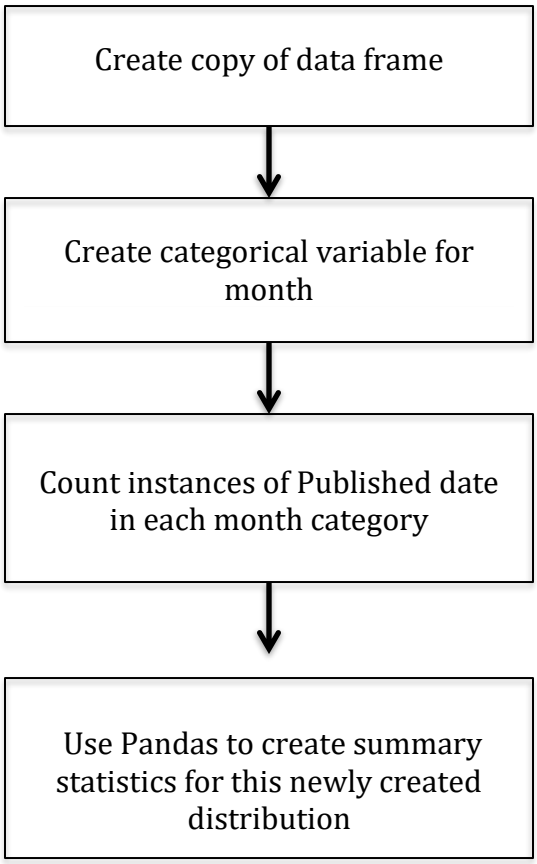


Visualization Module

Draw Graph



Summary Statistics



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 by our team member to store personal NCBI API keys and passwords.

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 preventing the creation of 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 scraper in this project.

Part1

In Part 1 of this project, we are tasked with creating a scraper 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 by assigning the initial parameters of the scraper module: keys, email, and search. Steps for assigning values to these parameters are discussed below:

- keys: 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. Use of the personal API key from NCBI allows 10 queries per second as opposed to the default of three.
- email: this has been assigned as the email address of the team member who created the module.
- search: this has been assigned as the search term "HIV".

Get_Pmid Function

The Entrez utility package allows efficient access of NCBI 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 scraper 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 scraper retrieves 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 is 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 scraper'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), and 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. Following this, the record is 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 then saved as a list of dictionaries, which is 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: 1) resetting the index of the dataframe, 2) converting the pmid variable to an integer data type, 3) formatting the dates into the %Y-%m-%d' format, and 4) joining the columns for title and abstract on index. Standardizing the datetime format this way not only allows for increased legibility of our results, but also makes it more efficient to perform the summary statistic calculations by month in Part 3 of this project.

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 a database automatically, and then implement SQL code to query the publication by author's name

Csv_bnb Function

Using the csv_bnb function, the CSV file created by the data crawler is read 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.

Sqlite_out Function

The sqlite_out function is used to take the hiv_csv file and two arguments, a database name (assigned as "HIV_Records") and a table name (assigned as "PubMed_Query"), 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 name "HIV_Records". If a database with the name exists, then the function will proceed to check for the specified table name "PubMed_Query". If the table exists within "HIV_Records", then the function will replace it. If the table does not already exist, then the function will create a new table with the specified name.

If "HIV_Records" does not already exist as a database, however, then the function will create a new database with the specified name and then create "PubMed_Query" as a table within "HIV_Records".

Sql_author_query Function

Using similar syntax and commands as before, function sql_author_query is then used for an author query, restricting results to those with a similar author name within a specified database and table by using the pandas read_sql function. SQLite is specified as the database dialect, and query results are restricted to those that contain the string input for the author name parameter. This function does not create a new database; it queries the already created database.

Given the amount of data in our dataframe, we only print the first 5 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 also 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. Users can input the optional string parameter "graph_type" to specify for the desired graph type of "line" (the default), "bar" or "both". The created graph is interactive.

First, a copy of the dataframe is created, which is beneficial 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 displays the month of publication and the y-axis displays 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 dataframe. The remaining fields are displayed when called in this function by specified month.

In [1]:
```
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
```

Part 1: Scraper

These are the initial parameters of the scraper module: keys, email, and search. For the purpose of this demo, these parameters have already been assigned. The user should reassign these parameters for individual-use. Descriptions for assigned demo parameters are provided below:

keys: 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. An empty YAML file has been included for use. Add your own API key to increase polling rate.

email: this has been assigned as the email address of the team member who created the module.

search: this has been assigned as the search term "HIV".

In [8]:
```
keys = pq.secret_manager("apikeys_project.yaml")

email = "rachit.sabharwal@uth.tmc.edu"
search = "HIV"
```

```
Read Successful
```

In the below cell, we gather the data for the final data frame. The get_pmid function queries the eSearch endpoint of the Entrez API to retrieve the corresponding pmids and join them to the input dataframe. 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. The data gathered is then converted from a Python dictionary into a JSON-encoded object and saved as hiv_records.json

For the purpose of time, do not run the cell. We have provided our output JSON file needed to continue the demo past this point.

In [ ]:
```
hiv_pmids = pq.get_pmid(contact=email, key=keys["apikeys"]["ncbikey"]["key"], term=search, mindate="2020/01/01", maxdate="2020/09/01")

hiv_records = pq.get_data(pmid_list=hiv_pmids, contact=email, key=keys["apikeys"]["ncbikey"]["key"])

with open('hiv_records.json', 'w') as outfile:
    json.dump(hiv_records, outfile)
```

In this section, the retrieved data is cleaned by executing the clean_data and keep_cleaning functions.

The keep_cleaning function performs additional cleaning on the data by: 1) resetting the index of the dataframe, 2) converting the pmid variable to an integer data type, 3) formatting the dates into the %Y-%m-%d' format, and 4) joining the columns for title and abstract on index.

Finally, the information from the dataframe is converted into CSV format.

In [2]:
```
with open('hiv_records.json', 'r') as outfile:
    hiv_records = json.load(outfile)

hiv_clean = pq.clean_data(hiv_records)
hiv_clean = pq.keep_cleaning(hiv_clean)

pq.file_downloader("hiv_csv_clean.csv", hiv_clean)
```

```
Your CSV is already up to date
```

Part 2: Database

Using the csv_bnb function, the CSV file created by the data crawler is read 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.

The head function is then used to display the first 5 results from the query.

In [3]:
```
hiv_csv = pq.csv_bnb("hiv_csv_clean.csv")
hiv_csv.head()
```

Out[3]:

	pmid	title	abstract	dates	author(s)
0	32866934	The prevalence and risk factors for systemic h...	Diabetes and hypertension are common chronic d...	2020-08-18	Almobarak Ahmed Omer, Badi Safaa, Siddiq Samar...
1	32866611	Expression, purification and crystallization o...	Cdc-like kinase 1 (CLK1) is a dual-specificity...	2020-08-29	Dekel Noa, Eisenberg-Domovich Yael, Karlas Ale...
2	32866436	COVID-19 pneumonia in an HIV-positive woman on...	COVID-19 pandemic has been a problem worldwide...	2020-08-26	Cipolat Murillo Machado, Sprinz Eduardo
3	32866396	Acute supplementation with beetroot juice impr...	Human immunodeficiency virus (HIV) is associat...	2020-08-31	Nogueira Soares Rogerio, Machado-Santos Ana Pa...
4	32866318	Model Informed Development of VRC01 in Newborn...	VRC01 is a first-in-class, potent, broadly neu...	2020-08-31	Li Jerry, Nikanjam Mina, Cunningham Coleen K, ...

The sqlite_out function is used to take the hiv_csv file and two arguments, a database name (assigned as "HIV_Records") and a table name (assigned as "PubMed_Query"), 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 name "HIV_Records". If a database with the name exists, then the function will proceed to check for the specified table name "PubMed_Query". If the table exists within "HIV_Records", then the function will replace it. If the table does not already exist, then the function will create a new table with the specified name.

If "HIV_Records" does not already exist as a database, however, then the function will create a new database with the specified name and then create "PubMed_Query" as a table within "HIV_Records".

In [4]:
```
pq.sqlite_out(hiv_csv, "HIV_Records", "PubMed_Query")
```

Function sql_author_query is then used for an author query, 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; it queries the already created database.

As seen in the cell below, the desired name for the query has been set as "Mary", the specified database is "HIV_Records", and the specified table is "PubMed_Query".

The head function is again used to display the first 5 results from this query.

In [5]:
```
sql_df = pq.sql_author_query("Mary", "HIV_Records", "PubMed_Query")
sql_df.head()
```

Out[5]:

	pmid	title	abstract	dates	author(s)
0	32866256	Nursing Considerations for Patients With HIV i...	Infection with HIV is a chronic condition that...	None	Graham Lucy, Makic Mary Beth Flynn
1	32864388	COVID-19 in Hospitalized Adults With HIV.	The spread of SARS-CoV-2 and the COVID-19 pand...	2020-08-01	Stoeckle Kate, Johnston Carrie D, Jannat-Khah ...
2	32860699	Risk factors for COVID-19 death in a populatio...	Risk factors for COVID-19 death in sub-Saharan...	2020-08-29	Boulle Andrew, Davies Mary-Ann, Hussey Hannah,...
3	32859191	Understanding long-term HIV survivorship among...	Persons living with HIV (PLWH) are living long...	2020-08-28	Freeman Robert, Gwadz Marya, Wilton Leo, Colli...
4	32852363	Brief Report: Increased Cotinine Concentration...	There is a strong link between cigarette smoki...	None	Diaz Philip T, Ferketich Amy, Wewers Mary E, B...

Part 3: Visualization

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 the draw_graph function. The created graph is interactive.

The default graph drawn is a line graph, which we have shown below. Users can input the optional string parameter "graph_type" to specify for the desired graph type of "line" (the default), "bar" or "both".

EX: pq.draw_graph(df,"both")

In [6]:
```
pq.draw_graph(hiv_csv)
```

Call on the summary_stats function to display the summary statistics by month. Change the string value to output summary statistics for different months. The current input month value is "january". This string is case-insensitive.

In [7]:
```
summary_stats = pq.summary_stats(hiv_csv, "january")
summary_stats
```

Out[7]:

	January (Publications per Month)
mean	27.580645
std	13.065921
min	3.000000
25%	18.000000
50%	32.000000
75%	37.500000
max	46.000000

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["ap
ikeys"]["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")
```