**­­­­**

**PH1975 Introduction to Data Science**

**Final Group Project**

**Team members:**

Maha Almohamad

Nsikak Jackson

Dania Mofleh

Garret Munoz

Chiu-Feng Yap

**The University of Texas Health Science Center at Houston**

**December 16, 2020**

**Section 1. Program design.** Please describe your ideas, draw your workflow diagram, and show the architecture design of your program.

The main idea of the Python program is to be able to take inputs, such as the key term (“hiv’), minimum date of the publication date (01/01/2020), and the maximum date of the publication date (09/01/2020), through Biopython, a set of tools for computational molecular biology, bioinformatics use, and research, to receive data from PubMed and exported the data as a CSV file. Entrez databases are used to integrate access to PubMed, which contains bibliographic records. MEDLINE is then used as a parser. The outputs (PMID, title of the publication, full author list, date of publication, and abstract) are then being exported/appended into a .csv file. Using the output CSV file, sqlite3, a C-language library is implemented to import the CSV file and build a database query the publications by author’s name using SQL code.

In addition, the last part of the program includes a visualization module that can i) read the CSV file, ii) show the number of publications in each year, iii) visualize the trend of the publication numbers over time, and iv) generate and visualize the summary statistics for the publication number per year, including mean, SD, range, median, 1st to 3rd quartile.

**Section 2. Implementation details.** Please describe all important implementation details (e.g., modules, classes/functions, 3-rd party packages and tools).

[**ast**](https://github.com/python/cpython/blob/3.9/Lib/ast.py)- The ast module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change each Python release; this module helps to find out programmatically what the current grammar looks like.

[**ast.literal.eval()**](https://docs.python.org/3/library/ast.html#ast.literal_eval) **-** evaluates an expression node or a string containing a Python literal or container display. The string or node provided may only consist of the following Python literal structures: strings, bytes, numbers, tuples, lists, dicts, sets, booleans, and None. This can also be used for safely evaluation strings containing Python values from unstructured sources without the need to parse the values oneself. It is not capable of evaluating arbitrarily complex expressions, for example involving operators or indexing.

[**csv**](https://docs.python.org/3/library/csv.html)- The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. CSV format was used for many years prior to attempts to describe the format in a standardized way in [**RFC 4180**](https://tools.ietf.org/html/rfc4180.html). The lack of a well-defined standard means that subtle differences often exist in the data produced and consumed by different applications. These differences can make it annoying to process CSV files from multiple sources. Still, while the delimiters and quoting characters vary, the overall format is similar enough that it is possible to write a single module which can efficiently manipulate such data, hiding the details of reading and writing the data from the programmer.

The [csv](https://docs.python.org/3/library/csv.html#module-csv) module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

The [csv](https://docs.python.org/3/library/csv.html#module-csv) module’s [reader](https://docs.python.org/3/library/csv.html#csv.reader) and [writer](https://docs.python.org/3/library/csv.html#csv.writer) objects read and write sequences. Programmers can also read and write data in dictionary form using the [DictReader](https://docs.python.org/3/library/csv.html#csv.DictReader) and [DictWriter](https://docs.python.org/3/library/csv.html#csv.DictWriter) classes.

[**Sqlite3**](https://docs.python.org/3/library/sqlite3.html) **-** SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. To use the module, a connection object must first be created that represents the database.

Import sqlite 3

Conn = sqlite3.connect(‘example,db’)

Once the connection is established, a cursor object can be created and called its execute() method to perform SQL commands.

[**Pandas**](https://pandas.pydata.org/docs/user_guide/index.html) **-** pandas is a PYthon package provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

[**Re**](https://docs.python.org/3/library/re.html) **-** A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Regular expressions can be concatenated to form new regular expressions; if *A* and *B* are both regular expressions, then *AB* is also a regular expression. In general, if a string *p* matches *A* and another string *q* matches *B*, the string *pq* will match AB. This holds unless *A* or *B* contain low precedence operations; boundary conditions between *A* and *B*; or have numbered group references. Thus, complex expressions can easily be constructed from simpler primitive expressions.

[**Numpy**](https://numpy.org/doc/stable/) **-** NumPy (Numerical Python) is an open source Python library that’s used in almost every field of science and engineering. It’s the universal standard for working with numerical data in Python, and it’s at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

The NumPy library contains multidimensional array and matrix data structures (you’ll find more information about this in later sections). It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

[**Biopython**](https://biopython.org/) **-** Biopython is a set of freely available tools for biological computation written in [Python](http://www.python.org/) by an international team of developers. It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the [Biopython License](https://github.com/biopython/biopython/blob/master/LICENSE.rst), which is extremely liberal and compatible with almost every license in the world.

[**Entrez**](http://www.ncbi.nlm.nih.gov/Entrez/)- Entrez provides code to access NCBI over the WWW. This module provides a number of functions like efetch (short for Entrez Fetch) which will return the data as a handle object. This is a standard interface used in Python for reading data from a file, or in this case a remote network connection, and provides methods like .read() or offers iteration over the contents line by line. Unlike a handle to a file on disk from the open(filename) function, which has a .name attribute giving the filename, the handles from Bio.Entrez all have a .url attribute instead giving the URL used to connect to the NCBI Entrez API.

All the functions that send requests to the NCBI Entrez API will automatically respect the NCBI rate limit (of 3 requests per second without an API key, or 10 requests per second with an API key) and will automatically retry when encountering transient failures (i.e. connection failures or HTTP 5XX codes). By default, Biopython does a maximum of three tries before giving up, and sleeps for 15 seconds between tries. You can tweak these parameters by setting Bio.Entrez.max\_tries and Bio.Entrez.sleep\_between\_tries.

The Entrez module also provides an XML parser which takes a handle as input.

Functions used in the demo include:

[**efetch()**](http://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.EFetch) **-** retrieves records in the requested format from a list of one or more primary IDs or from the user’s environment

[**esearch()**](http://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch) **-** searches and retrieves primary IDs (for use in efetch, elink, and esummary) and term translations and optionally retains results for future use in the user’s environment

[**Medline**](https://biopython.org/docs/1.75/api/Bio.Medline.html)- Code to work with Medline from the NCBI. It contains classes that record a dictionary holding Medline data as well as functions that read() that reads one Medline record and parse() that iterates over a bunch of Medline records.

[**Matplotlib**](https://matplotlib.org/) - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

[**pyplot**](https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.html)- matplotlib.pyplot is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting. Pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation.

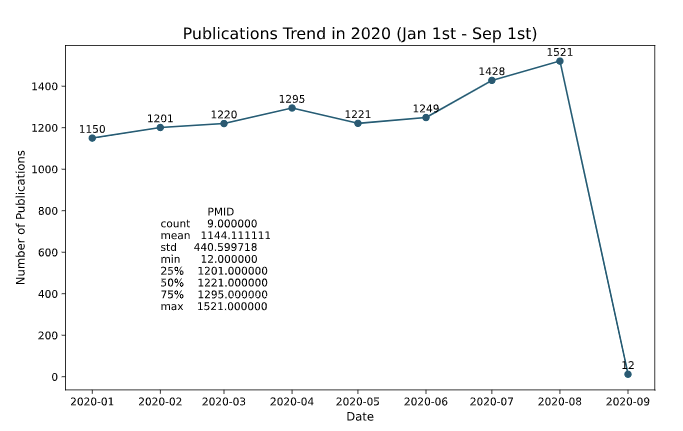
[**seaborn**](https://seaborn.pydata.org/#:~:text=Seaborn%20is%20a%20Python%20data,attractive%20and%20informative%20statistical%20graphics.) **-** seaborn is a Python data visualization library baked on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Section 3. Results.** Please include all required outputs (figures, tables, and/or numbers) in this section.









**Section 4. User manual/guide.**

Working with Jupyter Notebooks in [Visual Studio Code](https://code.visualstudio.com/).

Jupyter is an open-source project that allows users to easily combine Markdown text files and executable Python source code on one notebook.

To work with Jupyternote books in Visual Studio Code, the Anaconda environment in VS Code must be activated. Another option would be activating Python environment in which you have installed the Jupyter package. To select an environment, use the **Python: Select Interpreter** command from the Command Palette.

Once the environment is activated, create and open a Jupyterbook Notebook, connect to a remote Jupyter server from running code cells of the demo.ipynb file, and export a Jupyter Notebook as a Python file.

To open the demo.ipynb Jupyter notebook:

1. Open and select demo.ipynb in a new workspace.
2. When you select the ipynb file, the Notebook Editor is launced and will allow you to edit and run code cells in the demo ipynb file.
   1. Run code cell chunk 1 for the crawler module that can collect paper title, author list, publication time, and abstract from **PUBMED** for a given keyword (i.e., **HIV**) within a pre-specified time window (that is, **01/01/2020 – 09/01/2020**), and the retrieved data should be saved in the CSV format.
   2. Run code cell chunk 2 for the database module that can import the CSV file to SQLite to build a database automatically and query the publications by author’s name (i.e., input an author’s name and find out and return all his/her publications)
   3. Run code cell chunk 3 for the visualization module that can i) read the CSV file, ii) show the number of publications in each year, iii) visualize the trend of the publication numbers over time, and iv) generate and visualize the summary statistics for the publication number per year, including mean, SD, range, median, 1st to 3rd quartile.