

Tweet Collections

Automatic Population

of

Metadata Spreadsheet

By

Brandon Chang, Kirk Chenault, Chris Keener, Joseph Widrig

Virginia Tech, Blacksburg, VA 24061

CS4624: Multimedia, Hypertext, and Information Access

May 2, 2018

Clients

Liquing Li & Ziqian Song

TABLE OF CONTENTS

1. Table of Figures	2
2. Abstract.....	4
3. Introduction.....	6
4. Requirements.....	7
5. Design.....	8
6. Implementation.....	10
7. Testing and Evaluation.....	12
8. User's Manual.....	13
9. Developer's Manual.....	18
10. Lessons Learned.....	25
11. Acknowledgements.....	27
12. References.....	28

TABLE OF FIGURES

1. Populating an input table with only a category as a keyword	6
2. Diagram showing the architecture and data flow of the software	9
3. Running a Flask application on Ubuntu to load a HTML template to localhost	11
4. Example of ontology category 'Storm' [visualized].....	12
5. Python download page	13
6. Setting up Python	14
7. The Python Tkinter [3] window when first loaded [Version 1]	15
8. The Python Tkinter [3] window after given input [Version 1]	15
9. This will happen when it is running the program. Do not close the window, or you will lose progress.	16
10. This is the result when the program finished running. Click "Finish" to close the window.	16
11. . Example output from an error file.	17
12. Installing pip	18
13. Windows System Settings	19
14. System Properties and Environment Variables	19
15. Viewing updated status of the program running in developer mode	20
16. Example output from running the csv_parser script. Given the argument 'Collection Terms', this will return the values in the column for the specified rows	21

17. Part 1 of 2 of the Wikipedia [1] data retrieval script outlining how Wikipedia page and their content will be retrieved.	22
18. Code for extracting categories from the OWL file to cross reference content for spreadsheet entries.	23

ABSTRACT

This is a report generated to explain the details of the Tweet Collections project. The report contains both a user manual and developer manual, as well a “lessons learned” section. Throughout the process of completing this project, there have been many things to take into consideration. These are documented so that any future readers will be able to avoid pitfalls our team faced.

Over the past decade, social media use has grown exponentially. More and more people are using social networks to connect and communicate with one another, which has given birth to a new source of data: social media analysis. Since Twitter is one of the largest platforms for text based user input, many tools have been created to analyze data from this social media network.

The TweetCollections project is designed to analyze large amounts of tweet collection metadata, and provide additional information that makes the tweet collections easy to categorize and study. Our clients, Liuqing Li and Ziqian Song, have provided our team with a set of tweet collections and have asked us to assign metadata to them so that future researchers are able to easily find relevant collections. This includes assigning tags and categories, as well as a description with an accompanying source. Formerly, this process had been done by hand. While this improves the accuracy of the data collected, it is too expensive and time consuming to maintain. Our team has been tasked with speeding up the process, using scripts to find information for these fields and fill them out.

The majority of technology used in our approach has been concentrated on Python and its many libraries. Python has made it easy to quickly parse through our tweet collection data by treating the input as an Excel file, as well as pulling other relevant information from third party sources like Wikipedia. The driver will create a new, updated Excel file with the additional data, categories, and tags. Additionally, an ontology will be produced and serve as reference for categorizing topics listed in the fields from the input.

The GETAR team has created over 1400 tweet collections, containing over two billion tweets. To help categorize this data, they also store metadata about these collections in a Comma Separated Value (.csv) file. This project will result in a product that will take in a CSV file of the archive of tweet collections metadata as input, with the required fields (such as “Keyword” and/or “Date”) filled in, and produce a separate Comma Separated Value file as output with missing fields filled in. The overarching problem is that each category term is rather vague, and more data will need to be pulled out of this term. Additionally, an ontology will be produced and serve as reference for categorizing topics listed in the fields from the input. The completed project contains three Python scripts: `csv_parser.py`, `search_wikipedia.py`, and `GUI.py`. Together, these create a program that can take in an input CSV file and integer range for which lines to run, and then return a new CSV file with the additional metadata filled in. Also included with the deliverable is a populated Excel file, with over 150 additional entries of metadata, and an error file

containing recommendations for the ontology. These recommendations are generated from any results our driver determines as 'low relevance', and returns options with a higher term frequency.

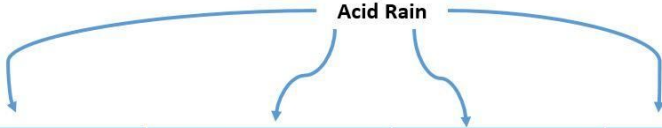
INTRODUCTION

The Global Events and Trend Archive Research (GETAR) team, funded by the National Science Foundation (NSF), has been collecting large amounts of data from Twitter for research purposes. Through many of their projects related to tweet activity and webpage analysis, the GETAR team has collected in total over 2 billion tweets. These are organized into roughly 1500 different tweet collections. These collections can be about trends, like climate change, or events, such as Hurricane Irma. The names of the tweet collections are of three types: keyphrases/keywords, mentions, and hashtags.

A spreadsheet [11] has been generated to describe each tweet collection. In each row of the spreadsheet, there are fields for the tweet collection title, a detailed description of what the title means, where to find more information regarding the title, tags, categories related to the title, starting date, and finally the total tweet count (Figure 1).

Previous personnel who were tasked with populating this data decided to work on the spreadsheet manually. This has led to a partially filled out spreadsheet. Our group has been tasked with creating an automated method where we could populate the spreadsheet with the desired information. This will help lower the amount of time as well as the cost needed to fill out the sheet manually.

In addition to working on the unfinished spreadsheet, our team will also work on extending the ontology (an extended taxonomy) for events and trends. With the massive amounts of data that GETAR is collecting, it's important that there exist some more comprehensive ontology to help understand the data. Our team will apply techniques that will help extend the ontology by recommending new terms and phrases relevant to various tweet collections.



Database	ID	Source	Wikipedia	Description	Category	Event Name	Count	Latest Date
GIVEN	GIVEN	GIVEN	Link to an article about 'acid rain'	The most comprehensive and relevant sentence found in the Wikipedia article	Search pre-existing categories for the most relevant for 'acid rain'	Title of the Wiki page	GIVEN	GIVEN

Figure 1: Populating an input table with only a category as a keyword

REQUIREMENTS

Our team was presented a spreadsheet that contained the metadata for the tweet collections that our client is archiving. When we met with the client to discuss what was expected of our group, there were talks of completing the entire spreadsheet, whether it be manually or automatically. Our client expressed concern, stating that they did not believe that we would be able to fully complete the document during the allotted time period. After discussing what would be a reasonable amount of work for our group, our client decided that we should complete at least 150 rows worth of data. Our solution would include a packaged program that could be used in the future in order to generate information for any possible future entries. The programming environment was left for the team to decide due to the many different ways that this project could be completed. Our client has been very helpful and willing to communicate on a frequent basis in order to address any issues that come up. Currently our requirements are to complete the work already assigned, but it is possible we may readdress these requirements and try to tackle some more advanced issues within the project.

DESIGN

The design process for this project was long and tedious. When we were first presented the project, our group first considered using C as the core system for our program. This was the language that our group was most comfortable with, considering we have had many classes that used C. Once we settled on a language we started to discuss the pros and cons of the language related to our project. C is a very powerful language and it would be capable of completing our project, however some members expressed concern over the amount of work needed to add all of the features. We met again and brainstormed which language would be the easiest to manage all of the features, as well as a language that would be easy for future developers to modify. One of our members suggested Python as a language because it is extremely modular and many users are familiar with it. Once we changed the foundation of our project to Python, we found it much easier to proceed.

Python offers many libraries that support Comma Separated Value file parsing. For a larger dataset, this should reduce run time since most of the computational work can be done within the module. We also know that Python has a Wikipedia [1] library available, since several of our members have used it before. Because the main source of data for our spreadsheet will be pulled from Wikipedia [1], this provides more evidence for choosing Python over another language. We can grab any page from Wikipedia [1], or grab a series of keywords from a page, all in just one line of code. Another advantage for data population with this language is the simplicity of making HTTP requests. It was brought up from our client that not every entry in our Comma Separated Value file dataset will have a Wikipedia [1] page. This brings up the concern that we will have to pull data from an alternate website if Wikipedia [1] does not contain any relevant information. Since we all have access to Virginia Tech's rlogin cluster, we all have access to Python version 2.

We decided that the project should include some sort of front end user interface to make the project easier to use. This way, not everything would have to be done from a command line. For the front end GUI, we discussed several possibilities. Since we had decided on using Python at this point, our front end would have to be Python compatible. (Any front ends built should have minimal effort connecting to the Python scripts.)

The resulting product contains the following files: CollectionTable.csv, csv_parser.py, CTRontology.owl, errors.txt, GUI.py, OutputCollectionTable.csv, README, search_wikipedia.py, and stoplist.txt. The CollectionTable.csv is an example spreadsheet to be filled in by our product, while the OutputCollectionTable.csv is an example result of running our product. The errors.txt file is generated from our product when there is a field that is too ambiguous as a search term, and it will recommend adding more ontology categories. The stoplist.txt is a simple stoplist containing common words that do not need to be searched. This file can be modified if the user notices a

need for a more restrictive list. Lastly, the Python scripts run the program with the files aforementioned.

The Python scripts are implemented with a front-end GUI.py script calling upon the csv_parser.py module. Inside the csv_parser.py module, it iterates through the given input CSV file and then pulls the search terms. The csv_parser.py module then calls on the search_wikipedia.py to bring back a page from wikipedia so it can parse the results and cross reference it with the ontology. The csv_parser module then takes that page and processes it, returning a word count map and a summary sentence. Finally, this script puts the resulting data into the output file and continues on to the next line. The complete flow of our product can be seen in Figure 2.

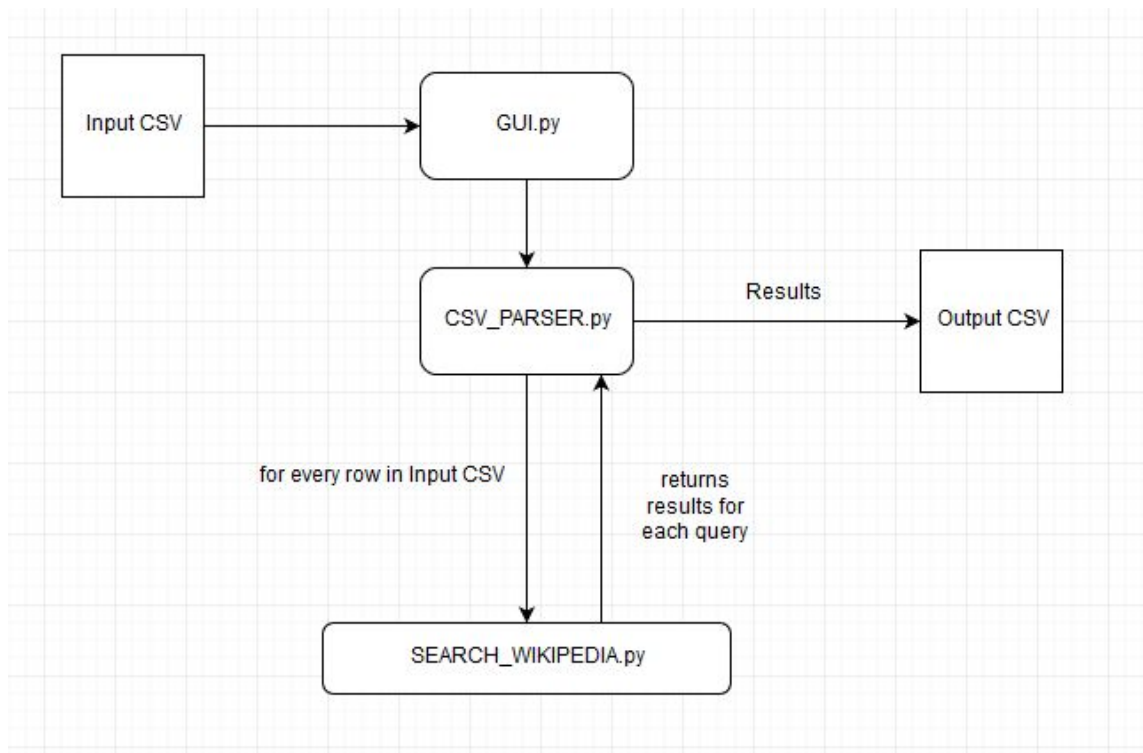


Figure 2: Diagram showing the architecture and data flow of the software

IMPLEMENTATION

The hardest challenge for executing our design ideas was figuring out how to create our Python sandbox environment. Since we are all VT students in the computer science department, we have access to both rlogin (a Linux cluster) as well as VTGitlab (Git for version control). We soon realized this would not be enough, since rlogin does not give us root access, for security reasons, and prevents us from experimenting with other libraries. Rlogin only lets us test Python libraries that are already installed on the cluster. One example of a Python library relevant to this project is the CSV Python library module, which lets us parse/update a CSV file.

Other libraries like Wikipedia [1] and Flask [2] need to be installed using Python's pip command. Because we do not have root access on rlogin, this means we needed another environment that supports Python. We decided to use a virtual machine running Ubuntu Linux. This gives us an easy to use environment as well as quick access to download any third party tools.

For our front end, the first option would be using something like a GUI library for Python. Tkinter [3] provides a simple GUI for Python applications. We looked into using this library at first, but the options are fairly limited. If this project were to be used in the long run, we decided a nicer GUI would be the best option. Several members of the group are familiar with HTML, so it was in heavy consideration. A web-based application would certainly provide the best looking GUI with the set of tools we have. Simply using Twitter's Bootstrap [4], along with editing different CSS elements, should make the most user friendly interface. The next problem would be connecting the Python scripts with the front end.

Several of our group members have some experience with web design, and the Flask [2] framework. Flask [2] is a Python microframework for creating Python based web applications. In short, Flask [2] could do all the complicated server-client communication and computation in the back-end, connecting our Python scripts to the front end HTML page. The other added bonus is that if this project were to become used on a larger scale, instead of using a localhost to display the HTML, you could set up a server to allow others to access the program and use the tools we created (Figure 3).

After much debate, the team decided to revert back to Tkinter [3] due to complications with Flask [2]. It was decided that our front end did not need to have much functionality other than input and output, therefore we turned back to Tkinter [3] due to its simplicity.

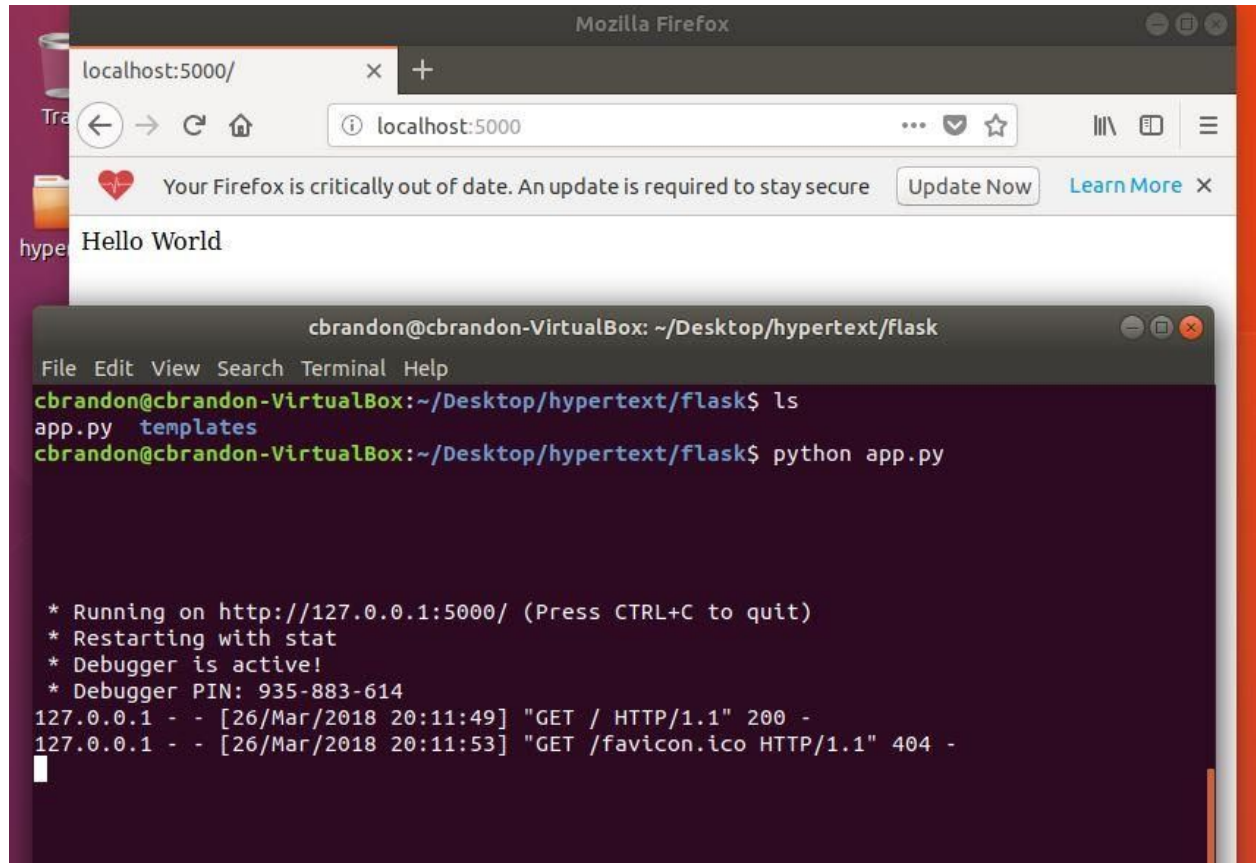


Figure 3: Running a Flask application on Ubuntu to load a HTML template to localhost

TESTING AND EVALUATION

For testing purposes, we have created a separate sandbox in rlogin to traverse and manipulate spreadsheet data. This is connected on VTGitlab so that all users can access and run various scripts on the spreadsheet data, with the option to discard their changes at the end of their session.

As we worked on implementation of pulling content from Wikipedia [1], we began testing different algorithms to pull descriptive sentences. We ran several tests on mock input CSV files to ensure that we would not alter the original file. Upon updates with our client, we have determined to make copies of all data generated, to ensure validity.

Another aspect of our project evaluation involves the updated ontology. Protege [10] is a data visualization tool recommended by the client. This will allow you to visualize relationships between categories in our OWL file, as seen in Figure 4. Our code does not update the OWL file directly, rather, in the errors.txt, it will make suggestions for new categories to be manually added through Protege. Our client has requested this approach be done in order to minimize the number of erroneous entries being added to the OWL file.

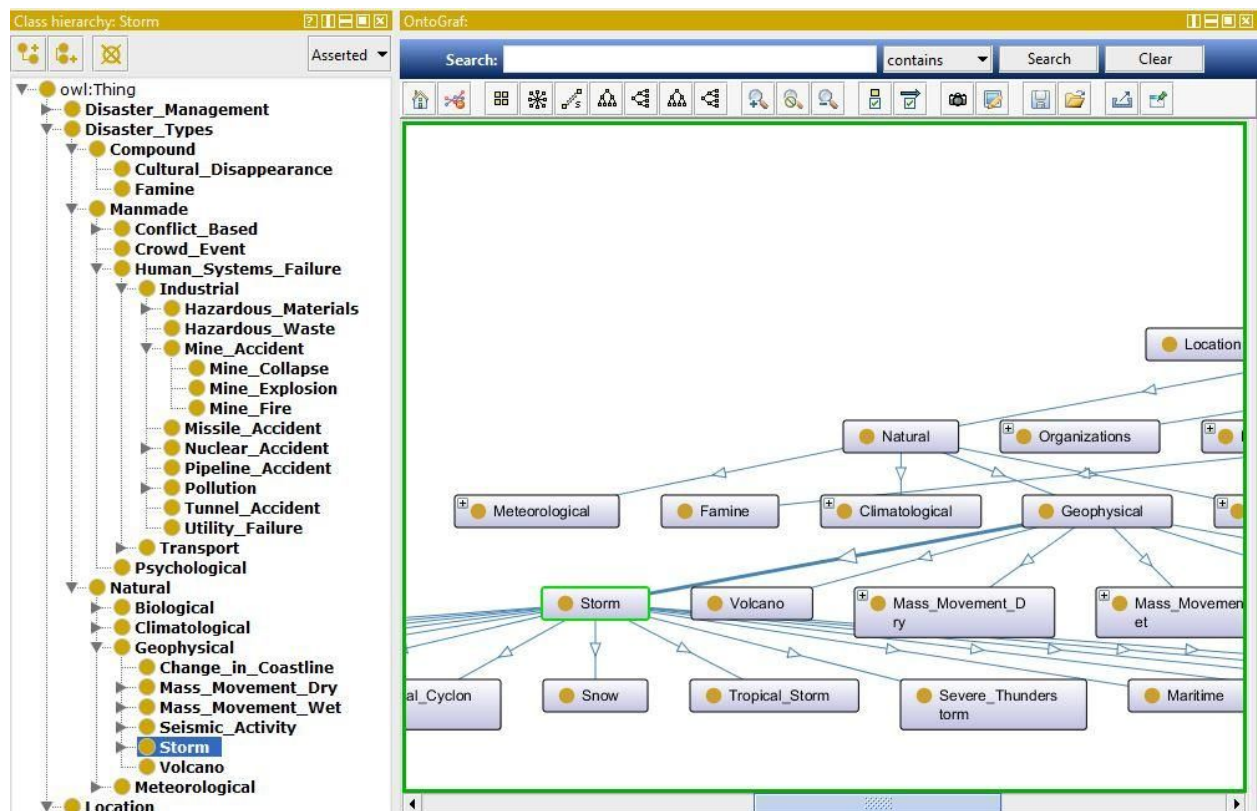


Figure 4: Example of ontology category 'Storm'

USER'S MANUAL

Starting with Python

Once you download the files that accompany this project, there is some setup that must be done to ensure that your machine is ready to begin. This project uses Python, a programming language that is not guaranteed to be native to your machine. If you are not sure if Python is installed, the next few paragraphs will walk you through setting up Python so that this project is ready to run.

To begin the installation process you must first download a Python installer. Our group used the installer from <https://www.python.org/downloads/> (Figure 5). Once you download this installer and begin running it, you will select all of the default options until you see a screen that appears similar to Figure 6.

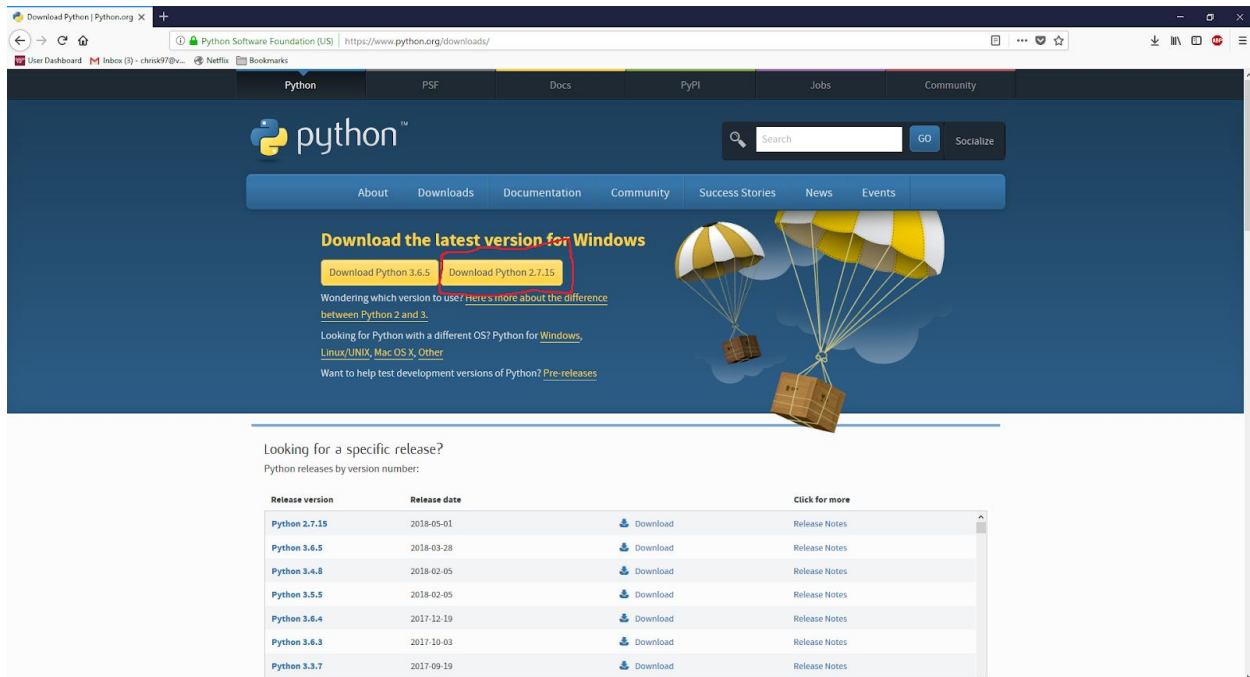


Figure 5: Python download page



Figure 6: Setting up Python

This part of the installation asks what sections of Python you wish to install on your machine. To ensure that everything works correctly, it is advised that you install all of these. There is an important change that must be made before proceeding. In Figure 6 there is a string of text that is highlighted that says “Add python.exe to Path”. You must scroll down until you can select this line and make sure that it is installed correctly. This will allow you to execute Python files without any issue. Once this has been updated, you can continue through the installation by selecting the default options until it finishes. Now Python is installed on your machine and can be used to execute .py files. Before our project is ready to run you must first open up a terminal. You can use either the command line if you are on Windows or terminal if you are on Linux. Once you have the terminal open, run the command “pip install wikipedia”. This is required for our project to access the wikipedia library. After pip installs the wikipedia package into your Python folder, you will be able to run our project. In order to run the project, open the folder “TweetCollections” and right click on the file “GUI.py”. Select the option “open with” and you will be prompted to select a program to open the file. If the Python installation was successful, you should be able to open with the Python terminal and a window will appear.

The window that opens when the script executes can be seen in Figure 7. The “Browse” button will prompt you to input the file you wish to run through our program. Once the user has provided the input file, a “Calculate Run” option appears (see Figure 8). When the user clicks the “Finish” button, it will close both the window and the file generated (in the same directory as where you pulled the input file).

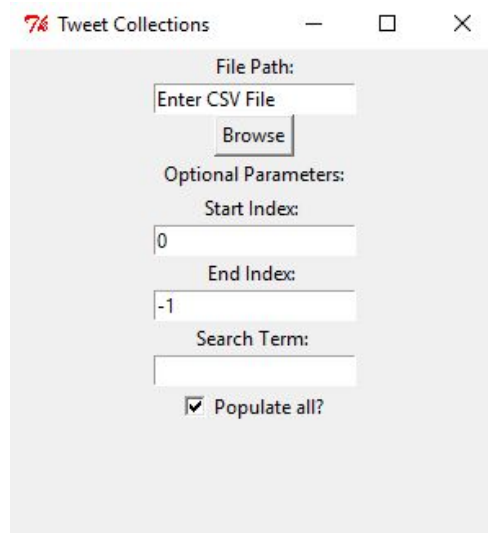


Figure 7: The Python Tkinter [3] window when first loaded [Version 1]

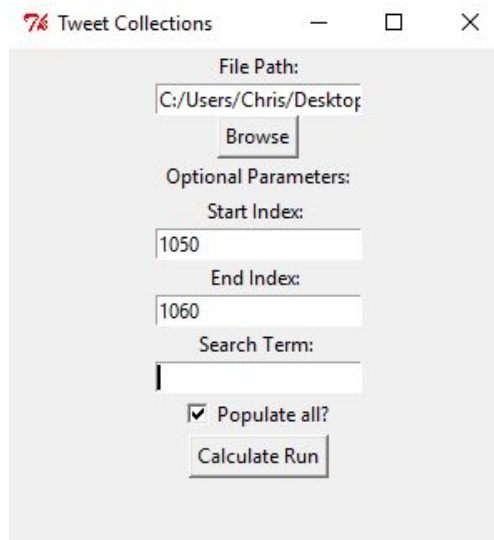


Figure 8: The Python Tkinter [3] window after given input [Version 1]

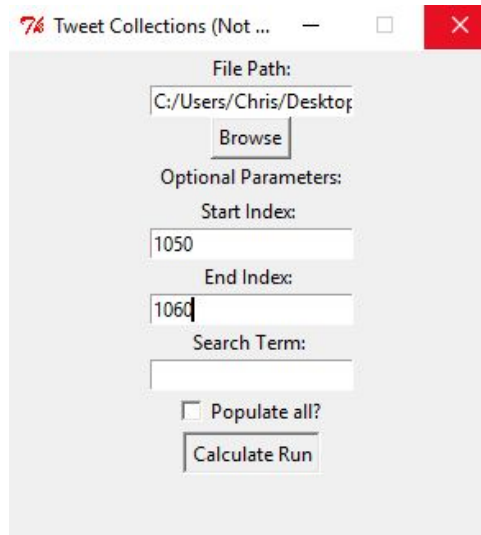


Figure 9: This will happen when it is running the program. Do not close the window, or you will lose progress.

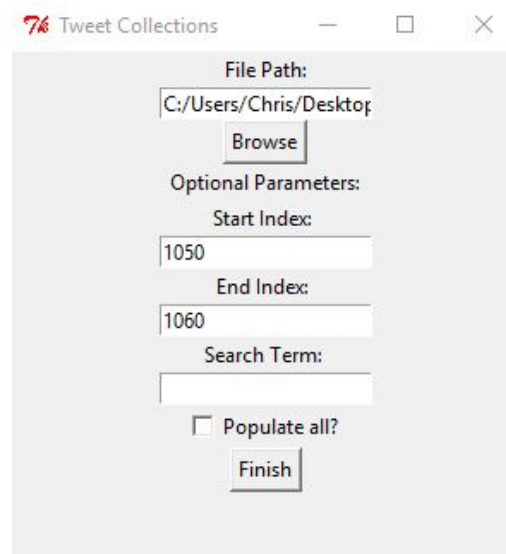


Figure 10: This is the result when the program finished running. Click “Finish” to close the window.

Discussions of the Use Environment

On the top of the interface, provide an input file through file selection via the ‘Browse’ button. At the very bottom, you receive the computed file through a ‘Finish’ button. In between, there are optional parameters for the user that provide more flexibility. The ‘Start Index’ text box will take a positive integer, representing the starting line that the user wishes to begin updating the CSV file. Note that the line number is the same as it will appear in the CSV input file. The ‘End Index’ text box will take a positive integer number. This represents the line the user would like to stop editing the input CSV file. The search term text box is an additional search parameter. When this box is

non-empty, it will tag this search term in the Wikipedia search query. This will help increase the relevance of the returned results, if the generic search was too ambiguous. Leave this box empty to perform a generic search. The last optional field is the “Populate All” checkbox. Leave a checkmark in the box if you wish to run the whole document, otherwise if left unchecked, it will run from the starting index to the ending index. See Figure 1 for a list of the fields.

Use Case

Providing an input file and optional parameters given to the product, where the specified fields in the file that are filled out will be run through our algorithm, and then putting the found information into a receiving file.

Tutorials on Use

When on the interface, simply click the “Browse” button and then select the file from your computer. Fill out any optional parameters if you wish. Once you click “Calculate Run”, this will process the file you submit with our Python driver. If the file is incompatible, the “Calculate Run” button will not appear. Once the file has been processed (Figure 9), a “Finish” button will appear at the bottom of the window. Click the “Finish” button to close the window (Figure 10).

Results

After running the program, two files are created. The first file is ‘OutputCollectionTable’. This is the updated CSV file, complete with Wikipedia links, tags, descriptions, and categories. The original input file will not be altered, to preserve accuracy. The second file is ‘errors.txt’ (Figure 11). This will contain any flags, or errors that popped up during the run. Current flags that we catch include failure to find a page, disambiguation errors within the Wikipedia library, and low ontological relevance. Any issues with the file that is generated should be placed in this file so that the users will be able to see them.

```
Error in searching for wikipedia page '#nrv 2012'
PLEASE CONSIDER SEARCHING FOR THE TERMS:

[[
"NRV" may refer to:
Net realizable value
New Revised Standard Version
New River Valley
Nintendo Revolution
Non-return valve
UPX
Valmet Nr I
Norddeutscher Regatta Verein
]]

FLAG: Article has low ontological relevance for search for "foursquare " using known keyword from ontology: "General" with results 2 entries.
Here are some suggestions: [('foursquare', 76), ('users', 42), ('location', 32), ('new', 28), ('user', 27)]
[Collect_YTK][34] No Page Found for '@ReadydotGov '
[Collect_YTK][35] No Page Found for '@craigatFEMA '
```

Figure 11: Example output from an error file

DEVELOPER'S MANUAL

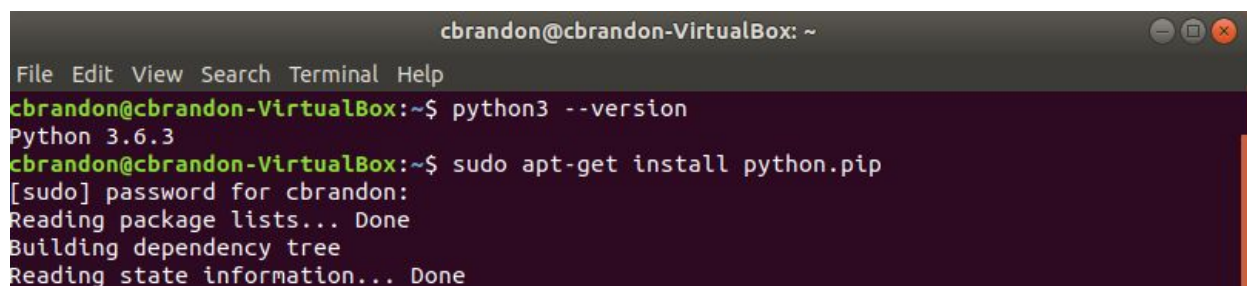
This project was developed in Ubuntu version 16.04.4, and Windows 10. Note that we used a shell that supports Unix/Linux commands for Windows, so all the commands and examples demonstrated in the manual are Linux/Unix based.

For Windows: Git Bash provides a nice shell that uses Linux/Unix based commands as well as supporting Git for future project changes (see: <https://gitforwindows.org/>).

For Mac: Mac is a Unix-based operating system, and as such, most of the bash functionality outlined in the Ubuntu sections is present natively in the MacOS shell.

Python:

The source code for our project is built using Python version 2.7.14. To run the script, you must have Python installed (see User's Manual for details). Linux/Unix users should already have Python installed. You can check Python's version by using 'python --version' or 'python -V' in the command line. You will also need 'pip' in order to install the necessary packages and libraries we have used in this project. See Figure 12 on how to install pip via linux.

A terminal window titled 'cbrandon@cbrandon-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
cbrandon@cbrandon-VirtualBox:~$ python3 --version
Python 3.6.3
cbrandon@cbrandon-VirtualBox:~$ sudo apt-get install python.pip
[sudo] password for cbrandon:
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 12: Installing pip

If you use a Windows operating system, 'pip' should already be installed upon installing Python 2.7.14, and can be found at <https://www.python.org/downloads/> (See User's Manual for how to install Python for Windows.)

In order to execute Python programs you must have Python in your path variable. You can check your path variable in Windows by visiting system settings as seen in Figure 13. Once you are in system settings you must click on "Advanced System Settings". This will bring you to a screen that can be seen in Figure 14. By clicking on "Environment Variables" you will see a screen that contains many different variables. By navigating to the Path variable you can verify that Python is correctly appended and ready to run.

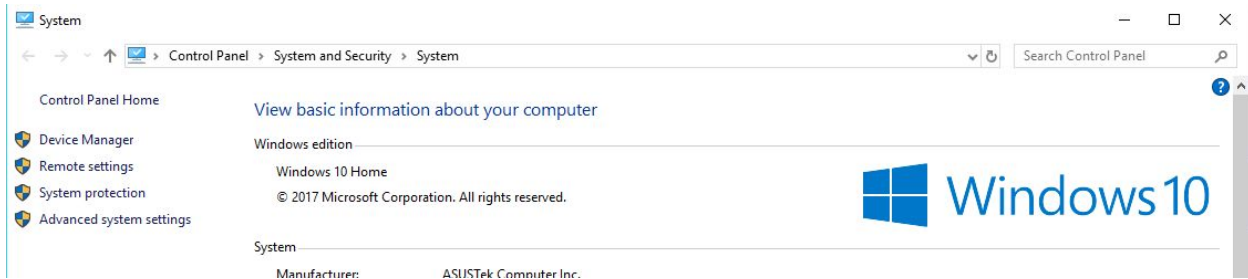


Figure 13: Windows System Settings

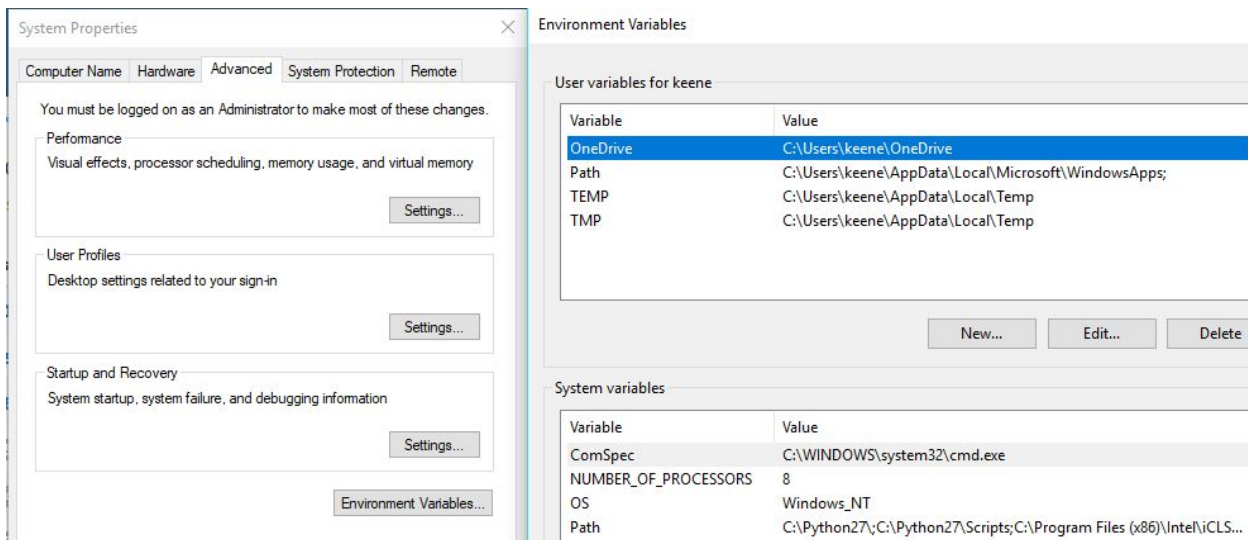
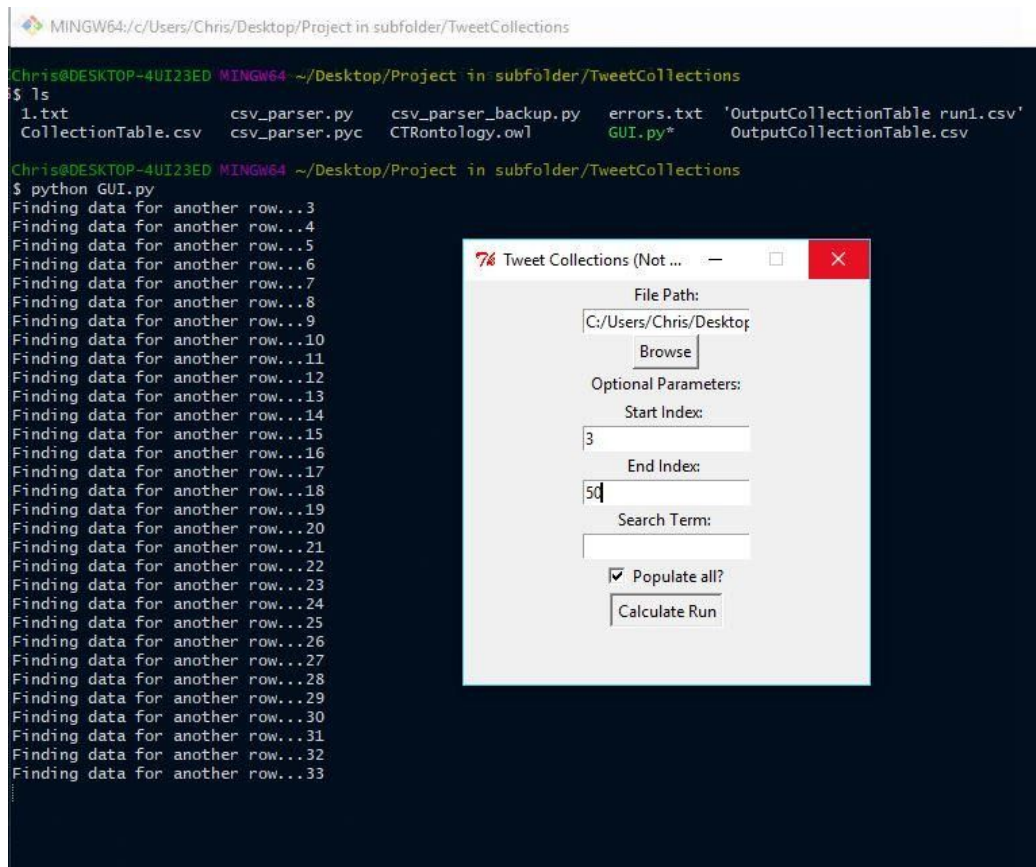


Figure 14: System Properties and Environment Variables

Running the Program in Developer Mode

This project supports running a version where output is tracked to the console. This involves constant updates on each row of the CSV being updated, as well as any exceptions, errors, and warnings that have been caught by the program. This mode can be used for tracking bugs and making changes to the program. To run in developer mode, open the project folder. Via command line, run 'python GUI.py'. This will launch the Python module responsible for opening up the GUI. Run the program in the usual manner (explained in the User Manual).

You can track the status of the program running in the terminal, as demonstrated in Figure 15.



```
MINGW64/c:/Users/Chris/Desktop/Project in subfolder/TweetCollections
Chris@DESKTOP-4UI23ED MINGW64 ~/Desktop/Project in subfolder/TweetCollections
$ ls
1.txt          csv_parser.py  csv_parser_backup.py  errors.txt  'OutputCollectionTable run1.csv'
CollectionTable.csv  csv_parser.pyc  CTRontology.owl      GUI.py      OutputCollectionTable.csv

Chris@DESKTOP-4UI23ED MINGW64 ~/Desktop/Project in subfolder/TweetCollections
$ python GUI.py
Finding data for another row...3
Finding data for another row...4
Finding data for another row...5
Finding data for another row...6
Finding data for another row...7
Finding data for another row...8
Finding data for another row...9
Finding data for another row...10
Finding data for another row...11
Finding data for another row...12
Finding data for another row...13
Finding data for another row...14
Finding data for another row...15
Finding data for another row...16
Finding data for another row...17
Finding data for another row...18
Finding data for another row...19
Finding data for another row...20
Finding data for another row...21
Finding data for another row...22
Finding data for another row...23
Finding data for another row...24
Finding data for another row...25
Finding data for another row...26
Finding data for another row...27
Finding data for another row...28
Finding data for another row...29
Finding data for another row...30
Finding data for another row...31
Finding data for another row...32
Finding data for another row...33
```

Figure 15: Viewing updated status of the program running in developer mode

CSV Parsing:

CSV (Comma Separated Values) is a type of file our program uses such that it can be used to traverse elements from the tweet collections datasheet. By creating a module that can access data elements from the spreadsheet, we will be easily able to insert data into each column, as well as pull information by column. We start by extracting the keyword from an input file using our `csv_parser.py` file. Figure 16 shows some output of our code detecting keywords from the original file.


```
[cbrandon@boxelder TweetCollections]$ ./csv_parser.py
The keywords are:
The keywords are: LAX shooting
The keywords are: #Isaac
The keywords are: hurricane sandy
The keywords are: hurricane
The keywords are: hurricane isaac
The keywords are: @NOAA
The keywords are: California storms
The keywords are: #egypt
The keywords are: #jan25
The keywords are: Nigerian school attack
The keywords are: Nigeria school attack
The keywords are: Connecticut shooting
The keywords are: connecticut school shooting
The keywords are: kentucky shooting
The keywords are: christiansburg mall shooting
The keywords are: santa monica shooting
The keywords are: atlanta school shooting
```

Figure 16: Example output from running the csv_parser script. Given the argument 'Collection Terms', this will return the values in the column for the specified rows

Wikipedia:

The primary source for event descriptions will be Wikipedia. We have installed a library entitled “Wikipedia” for use in retrieving pages and their contents (source: <https://pypi.python.org/pypi/wikipedia>). The contents of each article will be compared against the ontological categories in the ontology (.owl) file to estimate the relevance of the content found within the article. Additionally, it can be used to isolate the relevant sentences to become the description in the spreadsheet (Figure 17). This is accomplished by calculating the term frequency of the words found in the ontology, and determining which sentence contains the highest score.

```
In [33]: # import urllib2
owl_filename = 'CTRontology-080414 - update.owl'
import wikipedia
# temp = urllib2.urlopen("https://en.wikipedia.org/w/api.php?action=query&titles=Main%20Page&prop=revisions&rvprop=cont
# temp = wikipedia.search("Pizza")
# temp = wikipedia.page("Pizza").content
pages = wikipedia.search("9/11")

pages

# HTTP GET and scrape existing links
# wikipedia.page.content new queries

Out[33]: [u'September 11 attacks',
u'Fahrenheit 9/11',
u'9/11 conspiracy theories',
u'11/9',
u'9/11 (2017 film)',
u'Hijackers in the September 11 attacks',
u'Casualties of the September 11 attacks',
u'National September 11 Memorial & Museum',
u'9/11 Truth movement',
u'9/11 (2002 film)']

In [34]: page = wikipedia.page(pages[0])
content = page.content
content

to the attacks were still in the court system. On October 17, 2006, a federal judge rejected New York City's refusal
to pay for health costs for rescue workers, allowing for the possibility of numerous suits against the city. Governme
nt officials have been faulted for urging the public to return to lower Manhattan in the weeks shortly after the atta
cks. Christine Todd Whitman, administrator of the EPA in the aftermath of the attacks, was heavily criticized by a U.
S. District Judge for incorrectly saying that the area was environmentally safe. Mayor Giuliani was criticized for ur
ging financial industry personnel to return quickly to the greater Wall Street area.\nThe United States Congress pass
ed the James L. Zadroga 9/11 Health and Compensation Act on December 22, 2010, and President Barack Obama signed the
act into law on January 2, 2011. It allocated $4.2 billion to create the World Trade Center Health Program, which pro
vides testing and treatment for people suffering from long-term health problems related to the 9/11 attacks. The WTC
Health Program replaced preexisting 9/11-related health programs such as the Medical Monitoring and Treatment Program
and the WTC Environmental Health Center program.\n\n=== Economic ===\n\nThe attacks had a significant economic impa
```

Figure 17: Part 1 of 2 of the Wikipedia [1] data retrieval script outlining how Wikipedia pages and their content will be retrieved

Usage and Accessing Page Content:

Our usage of the Wikipedia [1] library is as follows: first, a search is made using the `wikipedia.search(string)` function, where the string is designated by driver code written in Python for each Twitter data collection. This search returns an array of page names for wikipedia pages containing relevant information. Our current iteration of the program chooses the first search result to be the page we check for relevance. Relevance is determined by pulling categories out of the ontology file and using it as a cross reference (Figure 18). If no relevance to the keyword is found, the script will then check the subsequent pages. Relevance is estimated after the page's content is accessed in the next step, where we return the page object to be stored in a temporary variable using the `wikipedia.page(string)` function. The content is then accessed by the `page.content` function.

```
In [32]: 1 f = open(owl_filename, 'r')
          2 lines = f.readlines()
          3 #lines

In [55]: for line in lines:
          line = line.replace('_', ' ')
          if 'Class IRI=' in line:
              line = line.replace('<Class IRI=\"#', ' ')
              line = line.replace('\\"/>', ' ')
              if line.upper() in content.upper():
                  print line
              #print line.upper()
```

Figure 18: Code for extracting categories from the OWL file to cross reference content for spreadsheet entries

Front End:

The front end of the project uses Tkinter [3] to support communication between the Tkinter [3] window and the Python driver. Note: Tkinter is included with the standard Microsoft Windows and Mac OS X installation of Python.

Because the Tkinter [3] window directly interacts with the Python driver, each input and output will be on a one-to-one mapping. This results in simplicity and could be expanded upon in future implementations to allow the interface to run multiple files in one session. To learn more about Tkinter [3], see <https://wiki.python.org/moin/TkInter>.

Please note that the code will not change the input file, but only will create a new version of the original file with the missing fields within it filled in.

LESSONS LEARNED

Throughout the process of working on this project, our group has overcome many obstacles. With each issue that we ran into, we made sure to take note of what went wrong and how we could have avoided them in the future. Many of these issues can be avoided with proper communication and planning.

When our team gathered together to discuss our milestones we were very confident in our ability to get a working solution quickly. We set our milestones up in such a way that a majority of the work was scheduled to be done early in the semester. Once we started working on the project, we quickly found that we would need more time in order to get a proper product. To split up the work evenly, our team decided to divide each of the problems into modules so that each member could progress on their own solution without holding back another member from working. Once each module is complete, we will be able compile all of the files together using a driver in Python. This solution has allowed us to address the timeline issue without causing problems.

Our group encountered other issues with the project regarding the programming environment. We originally wanted to use rlogin as the host environment because we all had access to it, and it is also hosted by Virginia Tech. We quickly found out that we would not be able to download the packages that we needed, so we decided to move to a local environment. Our group decided to utilize Ubuntu because it has Python pre-installed and would be easy to set up. We created our virtual machine and started designing our front end using Flask [2]. After our first few meetings we scheduled another meeting to work on the user interface. Unfortunately the virtual machine that contained the files had become corrupted, and we could not retrieve the files. Even though the loss was not very large because the UI was still in beta, our team learned the lesson of creating multiple backups in case of data loss. Due to this loss, we moved to the simpler interface, Tkinter [3], since it provided enough functionality to complete our objective.

Another issue that arose while creating this project was the realization of the complexity needed for our algorithms to determine how to best describe events and identify valid sources. Since we only have a single collection term and date to cross reference each entry, this brings up a lot of ambiguity. In order to counter this, we have several countermeasures including a hardcoded threshold value, but more advanced metrics should be taken with further versions of this project.

Since this project was created as a prototype for our client, there are several recommendations we have for future edits. The first edit would be to polish off the user interface. We have created a simple design, creating a seamless experience for the user, but if more complex heuristics are added, the GUI should be updated as well. As for the heuristics, there is room for added levels of complexity. To find relevant articles from Wikipedia, our algorithm just searches the name, date, and a keyword (optional) and uses the pre-existing ontology to determine relevance. The ontology provided is

not finished, so this method of comparison may be too naive or become outdated in the future.

ACKNOWLEDGEMENTS

Client:



Liuqing Li: liuqing@vt.edu
Ziqian Song: ziqian@vt.edu

Special thanks to the National Science Foundation (NSF [5]) for Global Event and Trend Archive Research (GETAR [6]) grant IIS-1619028

REFERENCES

- [1] Wikipedia Foundation, accessed 29 April 2018 www.wikipedia.org
- [2] Armin Ronacher, Flask, accessed 29 April 2018 <http://flask.pocoo.org/>
- [3] Andy Salnikov, Introduction to Tkinter, accessed 29 April 2018
<https://wiki.python.org/moin/TkInter>
- [4] Mark Otto, Bootstrap, accessed 28 April 2018 <http://getbootstrap.com/2.3.2/>
- [5] National Science Foundation, accessed 30 April 2018 <https://www.nsf.gov/index.jsp>
- [6] Global Event and Trend Archive Research, accessed 30 April 2018
<http://www.ctrnet.net>
- [7] Traversy Media, Python Flask From Scratch, accessed 25 April 2018
<https://www.youtube.com/watch?v=zRwy8gtgJ1A>
- [8] Installing Ubuntu inside Windows using Virtualbox, 20 October 2012, accessed 29 April 2018 <http://www.psychocats.net/ubuntu/virtualbox>
- [9] Ronacher, Armin, Flask web development one drop at a time, accessed 26 March 2018 <http://flask.pocoo.org/>
- [10] Protege, accessed 2 May 2018 https://protegewiki.stanford.edu/wiki/Main_Page
- [11] GETAR, TweetCollectionsSpreadsheet.csv, accessed 5 February 2018
<https://docs.google.com/spreadsheets/d/13wUfD-BI49Wklog8ZezfqTuuwQV0PKIIS9um0RoM80/edit#gid=0>