Natural Language Processing with Twitter to Assist Internet Outage Detection

Max G. Sydow

Western Governors University

Table of Contents

Summary

The overall goal of this project was to provide alerts for possible internet outages by analyzing content from customers posted on Twitter. The result was a visual dashboard showing counts of observed word groupings related to possible outage types. This dashboard was incorporated into the ISPs existing outage reporting system, and was used to trigger email alerts to relevant work groups to aid in diagnosing and resolving a common cause issue.

Directed Tweets were already being responded to by technical support representatives, and this same set of Tweets was chosen to analyze. Python's TwitterScraper package was used to extract and filter Tweet content. The remains were exported to comma separated files (CSVs). Spurious content such as stop words, articles, and names were filtered out. Different forms of the same words were also grouped together in a process known as lemmatization. For example, "talk", "talking", and "talked" are different verb forms of the same word. Once the content of Tweets was cleaned, counts of the most frequently occurring words and 2-3 word phrases were extracted. The sets of most frequent words and phrases were then organized into dataframes using Python's Pandas data analytics package.

From here word groupings were identified using a form of machine learning called natural language processing (NLP). Lists of words that may describe internet outages and those that don't were identified and used to test for similarity to form the word groupings from NLP. These lists were agreed upon during a brainstorming session involving all team members. The Word2Vec NLP model was used to obtain the groupings, and cosine similarity scores were computed to measure relatedness of groupings to the list of outage related terms. A t-SNE plot to visually represent the degrees of relatedness of words in Tweets to outage descriptors was made. Each word appears as a point on a 2-dimensional plane, color coded according to grouping. The larger the point, the stronger the word is related to an outage.

The ISPs existing outage reporting system was used as well. Outage categories, durations, and locations were extracted. The ISP uses an Oracle database to store the information from this system so SQL was used to make these queries. TwitterScraper was used again on each word grouping for the times that each outage actually occurred. If any frequency of word groupings coincided with the duration of an outage it was noted along with the number of words and phrases in each group. Since TwitterScraper does not provide exact locations, only cities that users identified were used to compare with real outage locations.

For some groupings it was clear that a higher spike in word groupings were related to outages, for others statistical T-tests could be performed. These T-tests allowed for rejection of a null hypothesis that the mean word frequencies for coincidence is less than or equal to the mean for non-coincidence. For coinciding groups the mean frequency minus standard deviations were computed to use as a

threshold indicator of possible outage. This threshold was divided by the time of outage duration to obtain a scalable metric.

Now that word groupings and thresholds of occurrences were identified, their figures were summarized graphically in Tableau. Each outage related word grouping was color coded and the counts for outage durations were displayed in histogram form. This visual dashboard was then added to the existing outage reporting system, and functionality to adjust time intervals was added.

Twitter's API provides geolocations, but can only be used to extract data going back one month. The same process of comparing with real outages carried out above was performed for the previous month. Deviations in corresponding word group frequencies were found to be negligible during that interval, so the metrics found from the longer duration analysis were kept.

Scripts to trigger the code to extract and analyze Tweets using the API were made. These scripts were set to run every hour. The output of word group counts were then updated to the dashboard with each iteration. The threshold of frequencies was indicated as a solid vertical line in each histogram column to visualize if/when the counts indicate a possible outage. If the height of a bin exceeded that threshold an alert was triggered. Scripts to monitor that were made, which then triggered sending emails to relevant work groups. The relevant work groups and content of emails were determined during an early brainstorming Scrum meeting.

A representative from each work group was chosen to respond to follow up to confirm that alerts and emails were received during the last day of work. Representatives from each organization that uses the dashboard were also consulted to make sure they were able to access it. After these meetings it was determined that further training on the use of these new features was needed.

Review of Other Work

In order to extract key words and phrases the content of Tweets need to be parsed and processed. Having to filter each word of each post in order to mine key terms could be very time consuming. Storing all content may also not the best use of memory for a database. In *Internet Outages, the Eyewitness Accounts: Analysis of the Outages Mailing List*, techniques from text mining and NLP to perform data preprocessing in a slightly different context were described. This white paper examines key words from a publicly available internet outages mailing list to categorize type of outage and entity involved.

One of the first things that the authors of this paper did to narrow down content was to remove stop-words. Stop-words include articles, prepositions, and pronouns. The SMART informational retrieval system, which is an early NLP system developed at Cornell University in the 1960s, was used to identify such stop-words (Banerjee et al., p. 4). This functionality can be performed by Python packages, which will be discussed later. Punctuation was also removed.

Lemmatization is the process of grouping together different forms of words. The verb “to talk” can be grouped along with “talk”, “talking”, “talked”, etc. Again, the authors used an older tool that can be supplanted by Python modules for this function. The Natural Language Tool Kit, or NLTK, package can be imported into Python code to perform lemmatization. Names of people, and organization names were also removed. I would argue that organization names such as “level 2” or “engineering” not be removed, as a knowledgeable customer’s expression of intuition may be a useful indicator.

Another group whose paper is closely related to the one described above describes data processing a little differently. In *Network Outages Analysis and Real-Time Prediction,* some more content extraction criterion was discussed. Links to other websites and emails were filtered out, as were abbreviated words such as ICS, and ISP. Apparently, output from Traceroutes were found in these emails, and they were also removed. I wouldn’t expect to see those in customer Tweets, but if they do appear, they would be more useful to networking experts than as a means of classifying language describing services being down.

The above sources describe data pre-processing, which is necessary before any sentiment analysis using NLP can be performed. Avlani (2019) describes how NLTK can be used to perform these tasks. Word tokenization breaks text down into individual words, but stop words are not removed yet. Python’s NLTK package includes a module that can filter stop words in different languages; only English content was examined. Lexicon normalization and stemming was also performed. This involves removing suffixes and prefixes of different forms of the same word. Lemmatization goes further to categorize root words that may not seem so obvious. Such base or root words are called ‘lemmas’. According to Avlani (2019) “The word “better” has “good” as its lemma.”

Applying natural language processing on Tweets was discussed in an article on the towardsdatascience.com site titled *Can We Use Social Media to Locate Legitimate Power Outages?* Although the purpose of the article was focused on identifying electrical power outages, many of the same techniques can be applied to identifying internet outages. The author discusses using Twitter’s API and TwitterScraper to extract semantic content.

As in the previous projects discussed, common stop-words were removed before looking at frequency of relevant words. Tweets from several of the largest US cities were scraped covering a time span of 5 years. The author indicates a preference for using TwitterScraper as opposed to Twitter’s API. While the API includes geolocations, it only provides one month of historical data. For ongoing analysis Twitter’s API may be used.

A set of key words and phrases such as “outage”, “power failure”, “electricity out” etc. were used to identify Tweets in targeted cities from which to further examine. Hill only mentions spending time exploring how people talk about power online. There would seem to be a great deal of subjectivity here, as there likely will be when describing internet outages. One would have to trust that if someone Tweets that they are experiencing a “power failure”, then there exists a reason beyond the premise for cause – i.e. an outage. For the purposes of this project, a basic common sensical list can be comprised. Further research on variations of vocabulary used by customers and networking professionals may help to serve as future goals to enhance the efficacy of possible side projects which may grow from this one.

The most frequently used words and 2-3-word long phrases were extracted from the Tweet set and composed into a data frame. Some further cleaning was performed on the data frames before modelling. The Word2Vec model was used "because of the way it focuses on the relationship of words and gives weight to that value". (Hill, Apr. 2016, para. 12). This model can be trained to learn conceptual relationships between words. Machine learning classification models are trained using sets of positive and negative labeled word sets. This is where the list of words that describe outages which were brainstormed come in. Since Word2Vec involves computations on higher dimensional vector spaces, a t-SNE model was used for dimensional reduction for purposes of visualization in 2-dimensions. The author then uses a cosine similarity measure to numerically gauge how closely the most frequent words and phrases correspond to targeted key words.

Hill provides a link to her GitHub page that includes Python code to accomplish the goals described in her article, [Hill, May 2019] This code can be adapted to be used for detecting internet outages. Different cities were used and Tweet selection criteria needed to be changed and different extraction time intervals should be used. Names of CSVs, dataframes, and other objects and variables required alteration, and file paths need to be modified. Positive and negative word lists need changing. Some functions to split timestamps needed altering as the lambda functions she used didn't work well with the adaptations that were made.

Her code included portions to compare with weather extracted from the National Oceanic and Atmospheric Administration (NOAA) data using their API. She also include some other neat functionality to create live maps. These aspects were not needed for the purposes of this project, but could be used for interesting further exploration. Overall, her code was easily adaptable, and performed what was desired for the portion of the project that analyzed historical data. The adaptation is included in Appendix A.

It is possible to schedule automatic refreshes in Tableau's data sets to update the graphs. The Tableau Online Help page shows how to do this. Windows Scheduler allows one to automate the Python scripts that extract, analyze, and update the csv's. Honeycut [2013] explains how to do this in his blog post. Task Scheduler can be accessed via the Control Panel. The option to 'Start a Program' should be selected under 'Action', and the name the script can be entered into a text box or browsed for exact file path. The scheduler can use batch file that references the file path for the Python scripts instead of pointing directly to them. This is useful if more than one Python script saved as different files need to be executed. Options for when to trigger the scripts can be designated, such as every hour or x-minutes.

It is also possible to embed Python code directly into a Tableau workbook to create a calculated field. In a Clearpeaks.com article by Marc G, the TabPy Tableau Python Server is discussed. This API can be installed and set up with Anaconda Navigator, or by using a simple 'pip install' command in an Anaconda prompt. Once installed the server is connected to by selecting options in the help menu on an open Tableau workbook. When editing a calculated field a Python script can be used. The scripts are contained within a 'SCRIPT_REAL()' tag. The Twitter scraping code which tabulates Tweet counts based on key word indicators could be incorporated in this manner.

Changes to Project Environment

The Tweets coming in used by support representatives to address issues were not being stored. The entire content of each Tweet is not needed, and would likely take up too much space. A table in the existing Oracle outage reporting database was created to store the filtered content consisting of relevant words in groupings. Time-stamp, and location fields were used in this table as well. The contents of the new table were exported to csv using Oracle SQL Developer. The SQL queries for this action are called a 'job', and Oracle's scheduler can be configured to run these jobs at regular intervals. The csv's were stored in OneDrive to allow access for multiple workgroups. The analytic Python scripts were also stored on a OneDrive folder. Another table in the database was added to store outage cause, location, and duration fields and linked to the word grouping database via location field.

The existing outage report site did not include any form of analytics for customer input from social media. The Tableau dashboard was published online, and a link to it was added to the site. The threshold of word group occurrences for which a possible outage may be identified was used to trigger email alerts to relevant workgroups. This task was also automated by triggering a checking Python script using Windows Task Scheduler. Relevant groups targeted included Network Operations Center (NOC), central office engineers, field managers, and Technical Support Operations (TSO).

Methodology

The Scrum project implementation methodology was used to carry this project out. Scrum breaks a project down into sprints, which involve team members working in a focused manner on a specific objective. Scrum meetings are scheduled to communicate progress on achieving benchmarks. Little scope creep or deviation is allowed, which keeps teams focused on their goals defined by the sprints. The path from extracting Tweets to presenting a functional dashboard can be broken down into manageable tasks.

A team of analysts, developers, a data scientist, a product owner, and Scrum Master were assembled. The product owner and Scrum Master are roles specifically designated to manage a Scrum project. The product owner served as liaison between the project group and stakeholders. Stakeholders include the relevant workgroups identified above, as well as some senior management. This role is also responsible for scheduling and managing any backlog of objectives that may not have met the planned timeline. The Scrum Master serves as a team leader in the project. They work closely with team members to keep the project on schedule and communicate any needs of the team with the product owner. The scrum master also conducts sprint meetings and documents progress in achieving objectives.

The analysts were heavily involved with the earlier objectives of the project including. These objectives include:

1. Scraping data from Tweets by username going back 5 yrs.
2. Cleaning data – remove stop words, etc.
3. Preparing data frames and conduct NLP
4. Extracting data from existing outage reporting system going back 5 yrs.

The data scientist was then responsible for performing the machine learning aspects. This included:

1. Performing correlation study between frequently appearing words and actual outage occurrence
2. Grouping high frequency words into training and testing feature sets
3. Coding and running Word2Vec algorithm

Developers played a crucial role in the latter portions of the project including:

1. Managing locations of scripts and csv's, and creating batch files for them
2. Scheduling execution of scripts, and email alerts

Analysts were also responsible for creating the Tableau dashboard, while developers linked it to the existing outage reporting site.

Project Goals and Objectives

For an ISP operating in the tri-state area of Texas, Oklahoma, and New Mexico, Tweets were extracted according to major cities that are served. These included: Austin, Dallas, Fort Worth, Houston, San Antonio, Oklahoma City, Tulsa, Albuquerque, and Santa Fe. Python's twitterscraper package was used, as well as other analytics packages including: pandas, NumPy, matplotlib, sklearn, json, regex, datetime, nltk, seaborn, codecs, and glob. A CSV for each city was written to before combining them. This master CSV was used to create the tweets_df dataframe.

The dataframe was then further cleaned. In addition to using the nltk package stopwords module and using regular expressions to filter content, a custom list of stop words was created. It was found that several unrelated words appeared in high frequency that don't really indicate much. These include articles, and transition verbs such as get, into, such, have, and, was, those, etc. The full list appears in code cell 12 of Appendix A. The remaining words in the dataframe were then tokenized to find frequency of occurrences. The filtered words and their corresponding frequencies are now much more related to internet issues. The same was applied to 2-3 word phrases. Top 10 words and phrases are tabulated by frequency in code cells 17 and 21 of Appendix A.

The Word2Vec model was then instantiated, and testing and training sets were applied. The list of words describing internet outages that were brainstormed were used at this point to train the Word2Vec model. Words related to typical outage categories were included in the positive set. These included terms that describe internet equipment such as router, and fiber; general descriptions of services such as google, twitter, website, wifi; appliances such as laptop, alexa, and phone; and natural causes like storms, and earthquakes. Words related to cellular data, electrical outages, and positive feelings regarding internet service were chosen for the negative set. Code cell 49 of Appendix A shows the full lists used. Cosine similarity scores were found, and a t-SNE model was created to visualize the strengths of words as they relate to internet outages and can be found in code cell 61 of Appendix A.

The top 10 words that indicate internet outages according to the NLP processing were found to be:

{*video, delay, equipment, modem, due, internet, youtube, website, netflix, facebook*}

These words were then used to extract Tweets again during intervals for which actual outages occurred as found from the ISPs outage reporting database. Each actual outage category corresponded to high occurrences of subsets of the above list of words. The correspondences were found to be much more pronounced than anticipated. Either there was a significantly noticeable spike in such words, or hardly any co-occurrence at all. Using T-tests to compare average frequencies of occurrence with non-occurrence was deemed unnecessary. Using the corresponding high frequency words as basis to define positive and negative classification sentiment for NLP was deemed enough to satisfy the correlation requirement. The average Tweet count per category was computed and divided by minute of outage duration to obtain a scalable metric. The standard deviations for each category and mean minus standard deviation was also computed and multiplied by 60 to get an hourly threshold for which to alert for possible outage.

These same high frequency indicator word subsets were then re-applied to the existing Twitter scraper for ongoing analysis. Begin and end date and times were changed to hourly intervals. For example, the words {delay, internet, youtube, website, facebook, and Netflix} were found to correspond to the category of outages caused by network congestion. Tweet count for each category was then tabulated and compared to the threshold described above. These Tweet counts were exported to CSVs which were then uploaded to Tableau.

The scripts for ongoing Tweet extraction were triggered to each hour using Windows Scheduler. Tableau refreshes were set to update every hour once the CSVs from each interval of Tweet extraction was completed. The Tableau workbook was published online and a link to it was incorporated into the ISPs existing outage reporting sited. Another Windows Scheduler event was created to trigger Python code to check if thresholds have been exceeded each hour and if so to send emails to relevant workgroups. Python’s smtplib package was used and the ISPs smtp server was needed for this requirement.

Project Timeline

Some objectives were scrapped as discussed in the Unanticipated Requirements section. The Python code was found to be easily adaptable from other sources. Completing the milestones associated with it was completed quicker than anticipated. At least half of the first day was spent brainstorming selection criteria, positive and negative word lists, and naming conventions for adapting code. The ongoing application of the code was quickly completed, since that only required slight modification of a portion of what was already made.

Creating the visual dashboard in Tableau also did not take as long as anticipated. However, the scripts to send emails needed several updates. It took several days of determining who exactly should be alerted. This required a lot of leg work for the project leader who enlisted others in the group to assist in communicating with others to make sure emails were received and content accurate.

Timeline Table

Milestone	Planned Duration	Planned Start Date	Planned End Date	Actual Start Date	Actual End Date
Goal 1: Cleaned Data Frame	½ Day	2/11/2020	2/11/2020	2/11/2020	2/11/2020 (3hrs)
Goal 2: Table from existing outage DB	½ Day	2/11/2020	2/11/2020	Scrapped	Scrapped
Goal 3: Summary of word groupings from NLP	½ Day	2/12/2020	2/12/2020	2/12/2020	2/12/2020 (5hrs)
Goal 4: T-test comparisons	½ Day	2/12/2020	2/12/2020	Scrapped	Scrapped
Goal 5: Dashboard with histogram summaries	½ Day	2/13/2020	2/13/2020	2/12/2020	2/12/2020 (2hrs)
Goal 6: Automation of continued analysis: scripts to run ETL and NLP, and send emails	2 Days	2/13/2020	2/14/2020	2/12/2020	2/14/2020

Unanticipated Requirements

Not having to perform T-tests to validate correspondences of actual outages with word frequencies was a nice perk. However, although the initial word lists used as criteria for Tweet extraction was agreed upon everyone involved believed it could be expanded. The same was thought of the positive and negative comparison lists for NLP application. Starting from the beginning and examining high frequency word for each interval of outage occurrence may lead to more accurate positive and negative lists of words, thereby improving accuracy. Doing so would take too much time and prolong completion, but serves as a good start for a separate related project.

In lieu of the previous discussion it was also decided to not create a new table in the DB for outage indicting word occurrences. At this time it didn't seem necessary. If or when the side project mentioned above led to more accurate categorizations then this objective may be revisited.

Embedding the Tableau workbook directly into the outage reporting site was something the developers had a tricky time doing. A PDF copy of the dashboard could be added, but the site did not want to update well along with the updates on the workbook. It was decided to simply have the workbook published online via Tableau, and include a link to it on the site.

Automatically triggering Tableau to refresh after the other triggers were executed revealed another snag. It was decided to set Tableau refreshes to occur 10 minutes after the execution of each new Tweet extraction. While it is possible to have Tableau run the Python scripts to create it's calculated fields it was decided to not do that. Connections to the TabPy server dropped at times and needed to be manually re-established. This would not be good if it were to occur during hourly refreshes.

Conclusions

The main goals of this project were realized. Some objectives were deemed unnecessary, and some minor issues were easily fixed. The Python code to extract Tweets, filter content, apply NLP, and summarize outage relevant words was long but easily adaptable from other sources. Additions to the existing Oracle DB was postponed, and performing T-tests was deemed unnecessary. This sped up the timeline of project completion. Making sure the right people were included in alerts took longer than anticipated. Good planning allowed for the flexibility to work through this challenge by the end of the week.

Other machine learning algorithms could be tried other than just Word2Vec. The initial Tweet extraction word list could be expanded or altered, as well as the positive and negative word lists. These factors give rise to possible side projects. Other social media platforms could definitely be used with a project like this as well. In many ways this was just a starting point for a much larger initiative to use machine learning and social media to become more pro-active with internet outage management.

References

Banerjee, R., Razaphpahah, A., Chiang, L., Mishra, A.,, Sekar, V., Choi, Y., & Gill, P. (Nov 16, 2015). Internet Outages, the Eyewitness Accounts: Analysis of the Outages Mailing List.

Retrieved from https://users.ece.cmu.edu/~vsekar/papers/pam15_outages.pdf

Zhu, G., Wei-Ting, L., & Sun. Z. (n.d.). Network Outages Analysis and Real-Time Prediction

Retrieved from http://zhuguanyu.github.io/fundamental_of_network/documents/[2]Network_Outage_Analysis_and_Real-Time_Prediction.pdf

Hill, Jen. (Apr 29, 2019). Can We Use Social Media to Locate Legitimate Power Outages? *towardsdatascience.com*

Retrieved from https://towardsdatascience.com/can-we-use-social-media-to-locate-legitimate-power-outages-7b7409708447

Hill, Jen. (May 20, 2019). Power_Outage_Identification. *gitbub.com*

Retrieved from https://github.com/jenrhill/Power_Outage_Identification/tree/b1bebb6afd61d007f026805ff5edcf6e8c521659

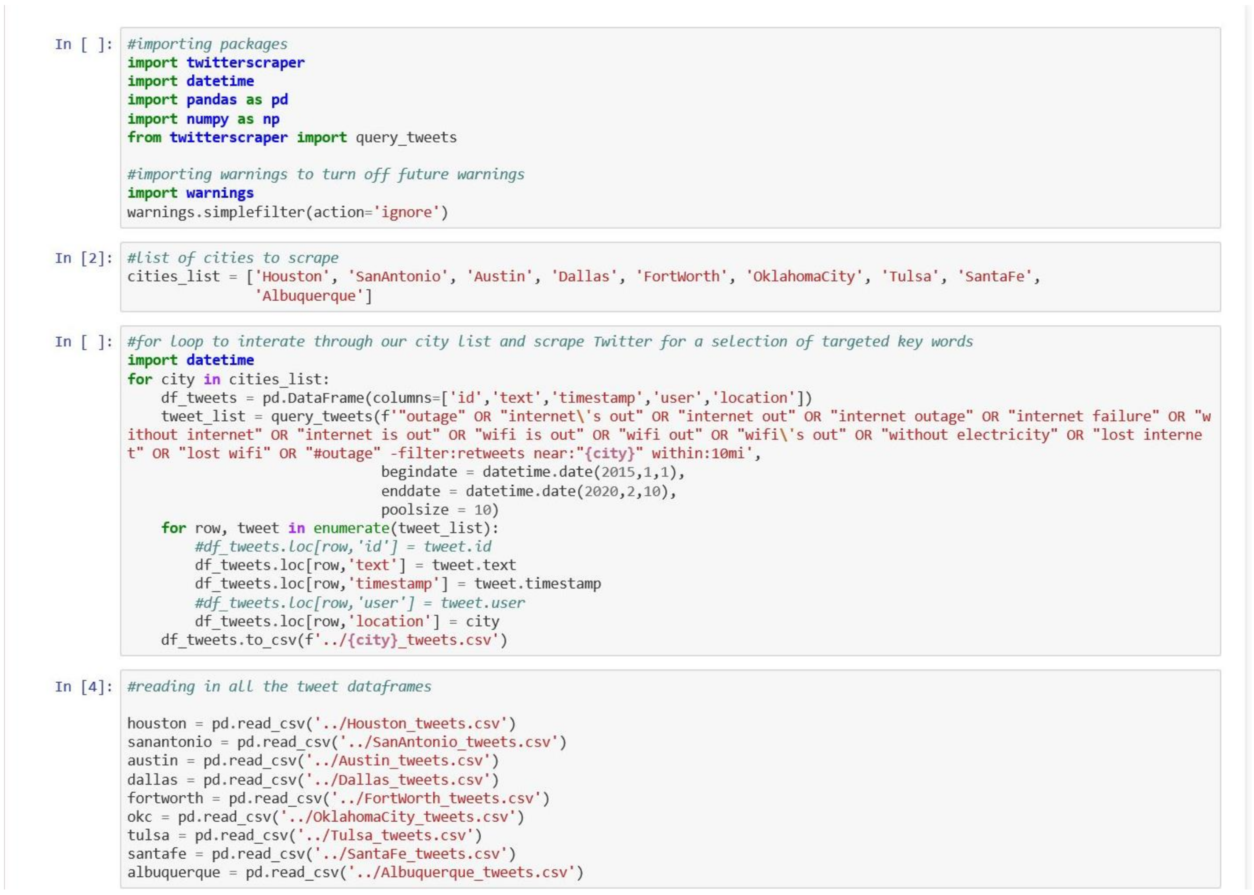
Marc, G [Apr 24, 2019]. Advanced Analytics: Tableau + Python.

Retrieved from. https://www.clearpeaks.com/tableau-python/

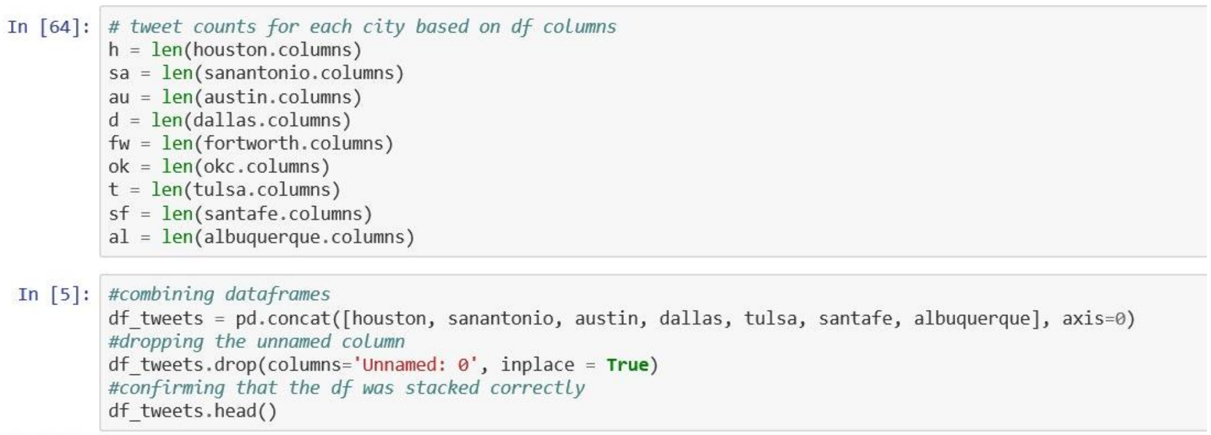
Honeycutt, Dale. [Jul 30, 2013]/ Scheduling a Python script or model to run at a prescribed time. Retrieved from. https://www.esri.com/arcgis-blog/products/product/analytics/scheduling-a-python-script-or-model-to-run-at-a-prescribed-time/

Project Deliverables

Appendix A: Python code for Tweet extraction and NLP



```
In [ ]: #importing packages
        import twitterscraper
        import datetime
        import pandas as pd
        import numpy as np
        from twitterscraper import query_tweets

        #importing warnings to turn off future warnings
        import warnings
        warnings.simplefilter(action='ignore')

In [2]: #list of cities to scrape
        cities_list = ['Houston', 'SanAntonio', 'Austin', 'Dallas', 'FortWorth', 'OklahomaCity', 'Tulsa', 'SantaFe',
                       'Albuquerque']

In [ ]: #for loop to interate through our city list and scrape Twitter for a selection of targeted key words
        import datetime
        for city in cities_list:
            df_tweets = pd.DataFrame(columns=['id','text','timestamp','user','location'])
            tweet_list = query_tweets(f'"outage" OR "internet\'s out" OR "internet out" OR "internet outage" OR "internet failure" OR "w
        ithout internet" OR "internet is out" OR "wifi is out" OR "wifi out" OR "wifi\'s out" OR "without electricity" OR "lost interne
        t" OR "lost wifi" OR "#outage" -filter:retweets near:"{city}" within:10mi',
                                      begindate = datetime.date(2015,1,1),
                                      enddate = datetime.date(2020,2,10),
                                      poolsize = 10)
            for row, tweet in enumerate(tweet_list):
                #df_tweets.loc[row,'id'] = tweet.id
                df_tweets.loc[row,'text'] = tweet.text
                df_tweets.loc[row,'timestamp'] = tweet.timestamp
                #df_tweets.loc[row,'user'] = tweet.user
                df_tweets.loc[row,'location'] = city
            df_tweets.to_csv(f'../{city}_tweets.csv')

In [4]: #reading in all the tweet dataframes

        houston = pd.read_csv('../Houston_tweets.csv')
        sanantonio = pd.read_csv('../SanAntonio_tweets.csv')
        austin = pd.read_csv('../Austin_tweets.csv')
        dallas = pd.read_csv('../Dallas_tweets.csv')
        fortworth = pd.read_csv('../FortWorth_tweets.csv')
        okc = pd.read_csv('../OklahomaCity_tweets.csv')
        tulsa = pd.read_csv('../Tulsa_tweets.csv')
        santafe = pd.read_csv('../SantaFe_tweets.csv')
        albuquerque = pd.read_csv('../Albuquerque_tweets.csv')
```



```
In [64]: # tweet counts for each city based on df columns
         h = len(houston.columns)
         sa = len(sanantonio.columns)
         au = len(austin.columns)
         d = len(dallas.columns)
         fw = len(fortworth.columns)
         ok = len(okc.columns)
         t = len(tulsa.columns)
         sf = len(santafe.columns)
         al = len(albuquerque.columns)

In [5]: #combining dataframes
        df_tweets = pd.concat([houston, sanantonio, austin, dallas, tulsa, santafe, albuquerque], axis=0)
        #dropping the unnamed column
        df_tweets.drop(columns='Unnamed: 0', inplace = True)
        #confirming that the df was stacked correctly
        df_tweets.head()
```

```
In [6]: #seeing how many rows we have
        df_tweets.shape

Out[6]: (1223, 5)
```

```
In [7]: #saving the file to a csv
        df_tweets.to_csv('df_tweets.csv')
```

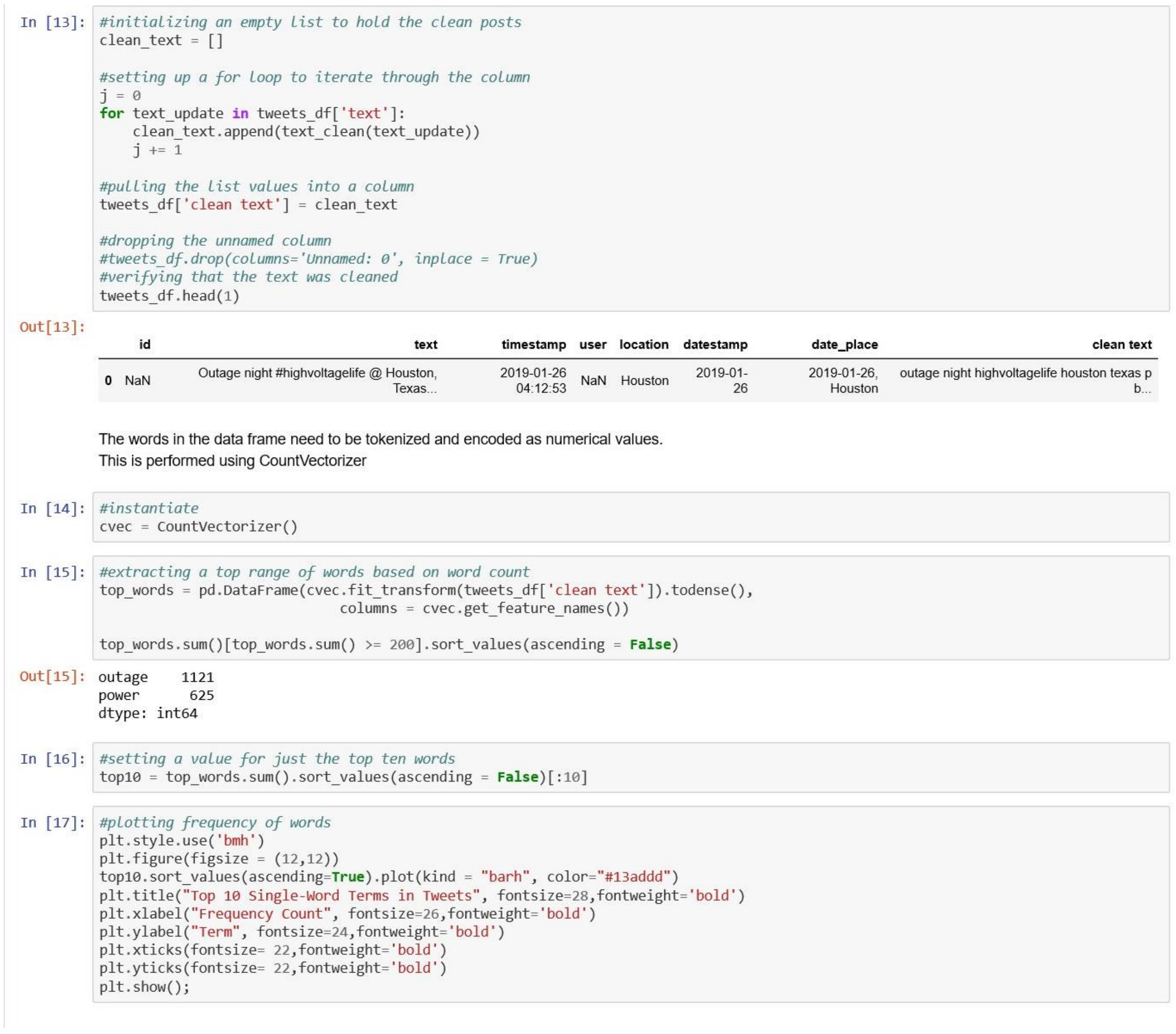
```
In [8]: #importing more packages to be used for further exploration and cleaning
        import json
        import regex as re
        import matplotlib.pyplot as plt
        from datetime import datetime
        from cdo_api_py import Client
        from pprint import pprint
        from sklearn.feature_extraction.text import CountVectorizer
        from nltk.corpus import stopwords
```

```
In [11]: #creating a new colume that combines date and location
         tweets_df = pd.read_csv('df_tweets.csv',)
         tweets_df.drop(columns=['Unnamed: 0'], inplace = True)
         tweets_df['datestamp'] = pd.to_datetime(tweets_df['timestamp']).dt.date
         tweets_df['date_place'] = tweets_df[['datestamp','location']].apply(lambda row: ', '.join(row.values.astype(str)), axis=1)
         tweets_df.head()
```

Out[11]:

	id	text	timestamp	user	location	datestamp	date_place
0	NaN	Outage night #highvoltagelife @ Houston, Texas...	2019-01-26 04:12:53	NaN	Houston	2019-01-26	2019-01-26, Houston
1	NaN	Outage night #highvoltagelife @ Houston, Texas...	2019-01-26 04:12:53	NaN	Houston	2019-01-26	2019-01-26, Houston
2	NaN	Unfortunately Pimlico and surrounding areas ar...	2019-01-22 23:19:55	NaN	Houston	2019-01-22	2019-01-22, Houston
3	NaN	It's been a while but looks like an outage nig...	2018-07-28 01:39:30	NaN	Houston	2018-07-28	2018-07-28, Houston
4	NaN	It's been a while but looks like an outage nig...	2018-07-28 01:39:30	NaN	Houston	2018-07-28	2018-07-28, Houston

```
In [12]: #cleaning dataframe text column and filtering stop words
         def text_clean(text):

             # removing non-letters
             letters_only = re.sub("[^a-zA-Z]", " ", text)

             # converting to lower case, split into individual words
             words = letters_only.lower().split()

             # creating a custom list of stopwords then converting it to a set to run faster
             stop_words = ['dlvr', 'get', 'got', 'go', 'bubly', 'us', 'http', 'www', 'https', 'com', 'ourselves', 'hers', 'between',
                           'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out',
                           'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do',
                           'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off',
                           'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the',
                           'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his',
                           'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this',
                           'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to',
                           'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them',
                           'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves',
                           'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did',
                           'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where',
                           'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom',
                           't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it',
                           'how', 'further', 'was', 'here', 'than', 'twitter', 'instagram', 'node',
                           'toplocalnow', 'cityandpress', 'pic', 'bit', 'ly', 'utm', 'source',
                          'ig', 'share', 'igshid']
             stops = set(stop_words)

             # removing stop words
             meaningful_words = [w for w in words if not w in stops]

             # joining the words back into one string separated by space,
             # and return the result.
             return(" ".join(meaningful_words))
```



```
In [13]: #initializing an empty list to hold the clean posts
         clean_text = []

         #setting up a for loop to iterate through the column
         j = 0
         for text_update in tweets_df['text']:
             clean_text.append(text_clean(text_update))
             j += 1

         #pulling the list values into a column
         tweets_df['clean text'] = clean_text

         #dropping the unnamed column
         #tweets_df.drop(columns='Unnamed: 0', inplace = True)
         #verifying that the text was cleaned
         tweets_df.head(1)
```

Out[13]:

	id	text	timestamp	user	location	datestamp	date_place	clean text
0	NaN	Outage night #highvoltagelife @ Houston, Texas...	2019-01-26 04:12:53	NaN	Houston	2019-01-26	2019-01-26, Houston	outage night highvoltagelife houston texas p b...

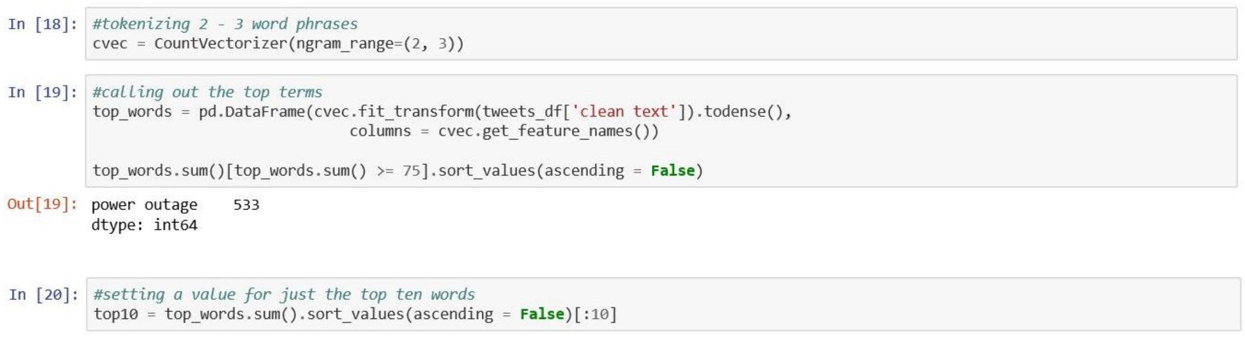
The words in the data frame need to be tokenized and encoded as numerical values.
This is performed using CountVectorizer

```
In [14]: #instantiate
         cvec = CountVectorizer()
```

```
In [15]: #extracting a top range of words based on word count
         top_words = pd.DataFrame(cvec.fit_transform(tweets_df['clean text']).todense(),
                                  columns = cvec.get_feature_names())

         top_words.sum()[top_words.sum() >= 200].sort_values(ascending = False)

Out[15]: outage    1121
         power      625
         dtype: int64
```

```
In [16]: #setting a value for just the top ten words
         top10 = top_words.sum().sort_values(ascending = False)[:10]
```

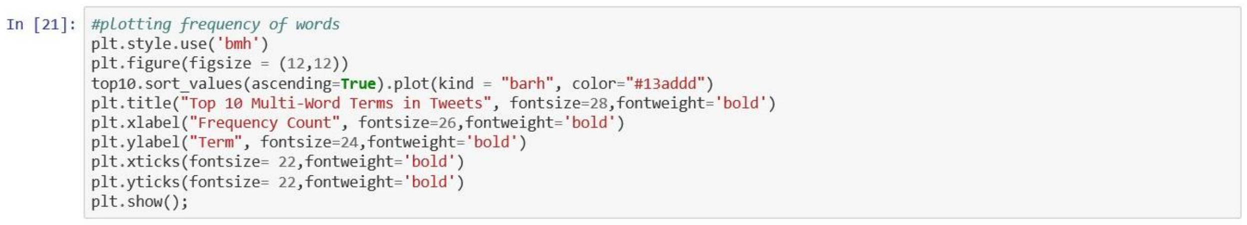
```
In [17]: #plotting frequency of words
         plt.style.use('bmh')
         plt.figure(figsize = (12,12))
         top10.sort_values(ascending=True).plot(kind = "barh", color="#13addd")
         plt.title("Top 10 Single-Word Terms in Tweets", fontsize=28,fontweight='bold')
         plt.xlabel("Frequency Count", fontsize=26,fontweight='bold')
         plt.ylabel("Term", fontsize=24,fontweight='bold')
         plt.xticks(fontsize= 22,fontweight='bold')
         plt.yticks(fontsize= 22,fontweight='bold')
         plt.show();
```



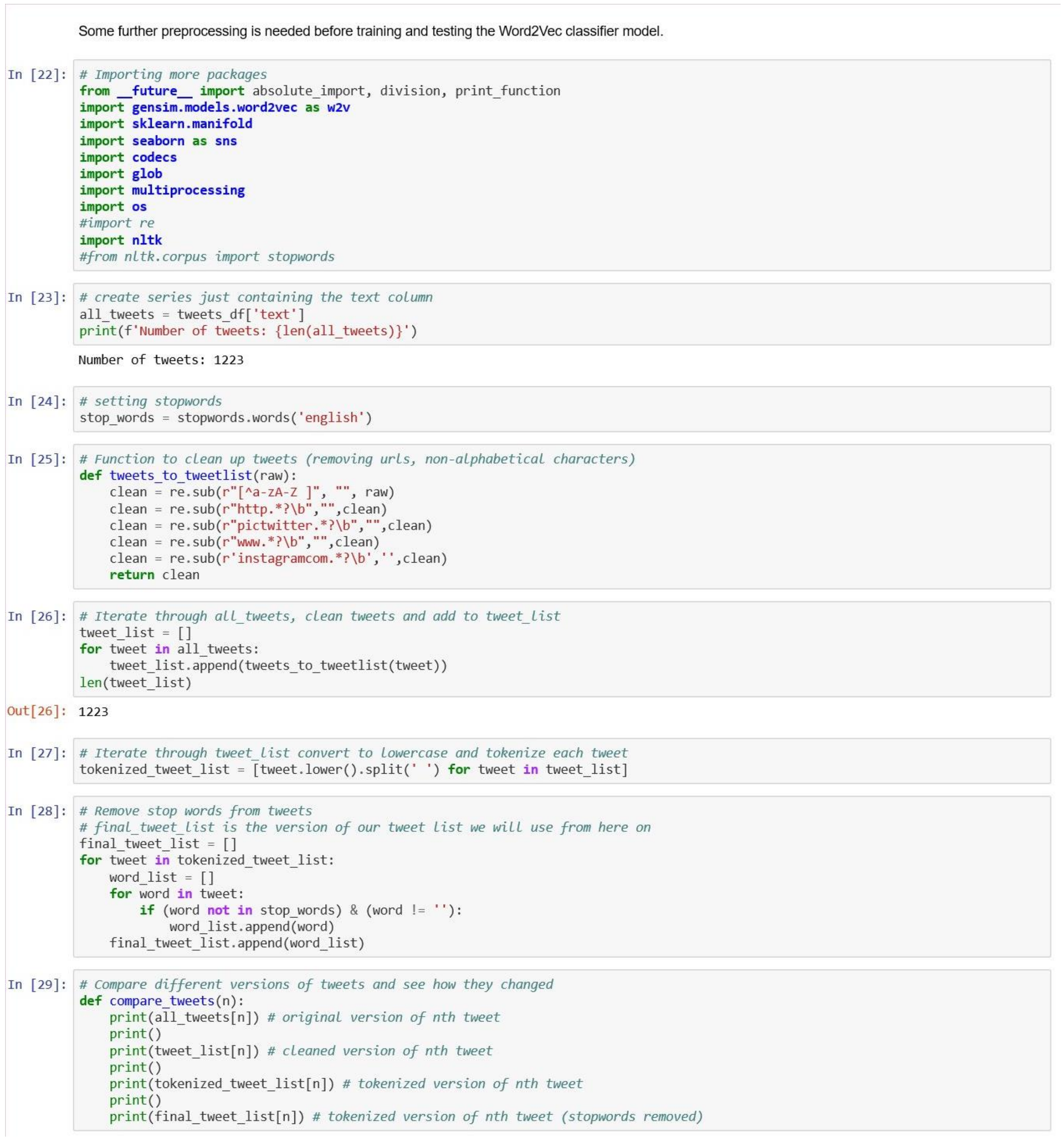
```
In [18]: #tokenizing 2 - 3 word phrases
         cvec = CountVectorizer(ngram_range=(2, 3))
```

```
In [19]: #calling out the top terms
         top_words = pd.DataFrame(cvec.fit_transform(tweets_df['clean text']).todense(),
                                  columns = cvec.get_feature_names())

         top_words.sum()[top_words.sum() >= 75].sort_values(ascending = False)

Out[19]: power outage    533
         dtype: int64
```

```
In [20]: #setting a value for just the top ten words
         top10 = top_words.sum().sort_values(ascending = False)[:10]
```

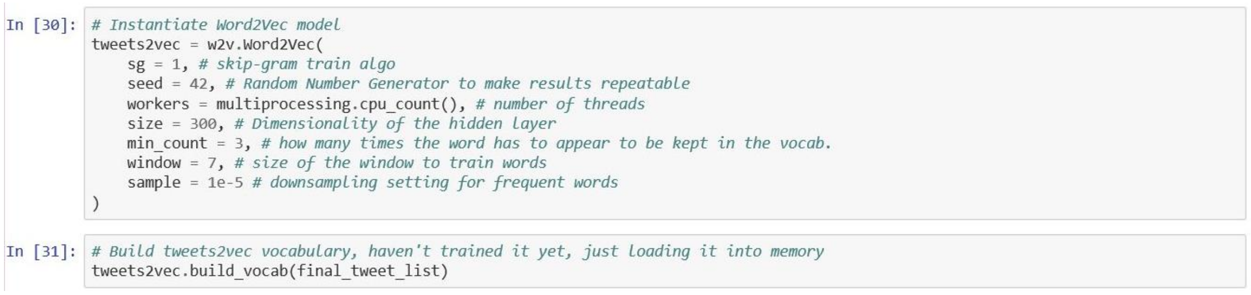


```
In [21]: #plotting frequency of words
         plt.style.use('bmh')
         plt.figure(figsize = (12,12))
         top10.sort_values(ascending=True).plot(kind = "barh", color="#13addd")
         plt.title("Top 10 Multi-Word Terms in Tweets", fontsize=28,fontweight='bold')
         plt.xlabel("Frequency Count", fontsize=26,fontweight='bold')
         plt.ylabel("Term", fontsize=24,fontweight='bold')
         plt.xticks(fontsize= 22,fontweight='bold')
         plt.yticks(fontsize= 22,fontweight='bold')
         plt.show();
```

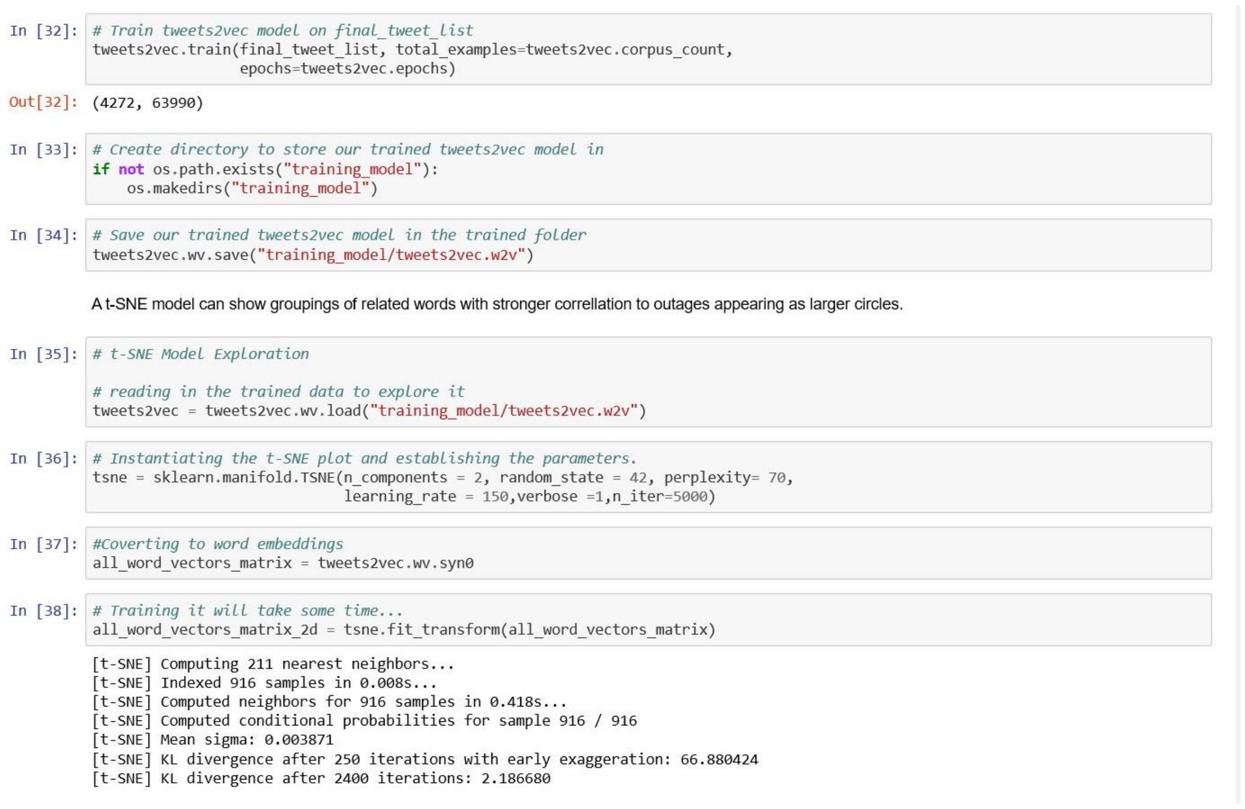


Some further preprocessing is needed before training and testing the Word2Vec classifier model.

```
In [22]: # Importing more packages
         from __future__ import absolute_import, division, print_function
         import gensim.models.word2vec as w2v
         import sklearn.manifold
         import seaborn as sns
         import codecs
         import glob
         import multiprocessing
         import os
         #import re
         import nltk
         #from nltk.corpus import stopwords

In [23]: # create series just containing the text column
         all_tweets = tweets_df['text']
         print(f'Number of tweets: {len(all_tweets)}')

         Number of tweets: 1223

In [24]: # setting stopwords
         stop_words = stopwords.words('english')

In [25]: # Function to clean up tweets (removing urls, non-alphabetical characters)
         def tweets_to_tweetlist(raw):
             clean = re.sub(r"[^a-zA-Z ]", "", raw)
             clean = re.sub(r"http.*?\b","",clean)
             clean = re.sub(r"pictwitter.*?\b","",clean)
             clean = re.sub(r"www.*?\b","",clean)
             clean = re.sub(r'instagramcom.*?\b','',clean)
             return clean

In [26]: # Iterate through all_tweets, clean tweets and add to tweet_list
         tweet_list = []
         for tweet in all_tweets:
             tweet_list.append(tweets_to_tweetlist(tweet))
         len(tweet_list)

Out[26]: 1223

In [27]: # Iterate through tweet_list convert to lowercase and tokenize each tweet
         tokenized_tweet_list = [tweet.lower().split(' ') for tweet in tweet_list]

In [28]: # Remove stop words from tweets
         # final_tweet_list is the version of our tweet list we will use from here on
         final_tweet_list = []
         for tweet in tokenized_tweet_list:
             word_list = []
             for word in tweet:
                 if (word not in stop_words) & (word != ''):
                     word_list.append(word)
             final_tweet_list.append(word_list)

In [29]: # Compare different versions of tweets and see how they changed
         def compare_tweets(n):
             print(all_tweets[n]) # original version of nth tweet
             print()
             print(tweet_list[n]) # cleaned version of nth tweet
             print()
             print(tokenized_tweet_list[n]) # tokenized version of nth tweet
             print()
             print(final_tweet_list[n]) # tokenized version of nth tweet (stopwords removed)
```



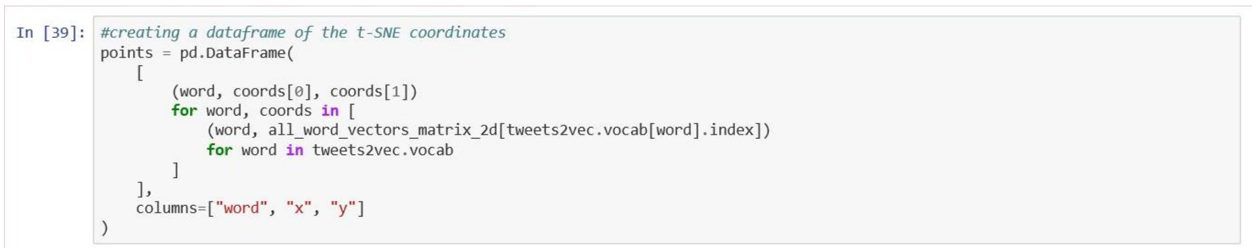
```
In [30]: # Instantiate Word2Vec model
         tweets2vec = w2v.Word2Vec(
             sg = 1, # skip-gram train algo
             seed = 42, # Random Number Generator to make results repeatable
             workers = multiprocessing.cpu_count(), # number of threads
             size = 300, # Dimensionality of the hidden layer
             min_count = 3, # how many times the word has to appear to be kept in the vocab.
             window = 7, # size of the window to train words
             sample = 1e-5 # downsampling setting for frequent words
         )

In [31]: # Build tweets2vec vocabulary, haven't trained it yet, just loading it into memory
         tweets2vec.build_vocab(final_tweet_list)
```



```
In [32]: # Train tweets2vec model on final_tweet_list
         tweets2vec.train(final_tweet_list, total_examples=tweets2vec.corpus_count,
                          epochs=tweets2vec.epochs)

Out[32]: (4272, 63990)

In [33]: # Create directory to store our trained tweets2vec model in
         if not os.path.exists("training_model"):
             os.makedirs("training_model")

In [34]: # Save our trained tweets2vec model in the trained folder
         tweets2vec.wv.save("training_model/tweets2vec.w2v")
```

A t-SNE model can show groupings of related words with stronger correlation to outages appearing as larger circles.

```
In [35]: # t-SNE Model Exploration

         # reading in the trained data to explore it
         tweets2vec = tweets2vec.wv.load("training_model/tweets2vec.w2v")

In [36]: # Instantiating the t-SNE plot and establishing the parameters.
         tsne = sklearn.manifold.TSNE(n_components = 2, random_state = 42, perplexity= 70,
                                      learning_rate = 150,verbose =1,n_iter=5000)

In [37]: #Coverting to word embeddings
         all_word_vectors_matrix = tweets2vec.wv.syn0

In [38]: # Training it will take some time...
         all_word_vectors_matrix_2d = tsne.fit_transform(all_word_vectors_matrix)

         [t-SNE] Computing 211 nearest neighbors...
         [t-SNE] Indexed 916 samples in 0.008s...
         [t-SNE] Computed neighbors for 916 samples in 0.418s...
         [t-SNE] Computed conditional probabilities for sample 916 / 916
         [t-SNE] Mean sigma: 0.003871
         [t-SNE] KL divergence after 250 iterations with early exaggeration: 66.880424
         [t-SNE] KL divergence after 2400 iterations: 2.186680
```



```
In [39]: #creating a dataframe of the t-SNE coordinates
         points = pd.DataFrame(
             [
                 (word, coords[0], coords[1])
                 for word, coords in [
                     (word, all_word_vectors_matrix_2d[tweets2vec.vocab[word].index])
                     for word in tweets2vec.vocab
                 ]
             ],
             columns=["word", "x", "y"]
         )
```

```
In [40]: # Define vectorization function
         def vectorize_corpus(keyword_list):

             # Instantiate counter for number of words in keyword_list that exists
             n_words = 0

             # Create template for cumulative corpus vector sum
             corpus_vec_sum = np.zeros((1,300))

             # Scan through each word in list
             for word in keyword_list:
                 if word in tweets2vec.vocab:
                     word_vec = tweets2vec.word_vec(word)
                     n_words +=1
                     corpus_vec_sum = corpus_vec_sum + word_vec

             # Compute average vector by taking cumulative vector sum and dividing it by number of words traced
             corpus_avg_vec = corpus_vec_sum/n_words

             # Squeeze this N-dimensional nested array object into a 1-D array to streamline future processing
             corpus_avg_vec = np.squeeze(corpus_avg_vec)

             return(corpus_avg_vec)
```

```
In [41]: #defining cosine similarity function
         def cos_sim(vector_1, vector_2):
             dp = np.dot(vector_1, vector_2)
             magnitude_v1 = np.sqrt(np.dot(vector_1,vector_1))
             magnitude_v2 = np.sqrt(np.dot(vector_2,vector_2))
             return(dp/(magnitude_v1*magnitude_v2))
```

```
In [49]: #creating pos/neg keyword lists

         # List of words that may describe an internet outage
         internet_out = ['damage', 'cut', 'equipment', 'router', 'modem', 'ont' 'computer', 'website',
                         'storm', 'flooding', 'failure', 'cable', 'buffering', 'slow',
                         'netflix', 'streaming', 'video', 'email', 'facebook','earthquake', 'speed',
                         'congestion', 'phone', 'tablet', 'laptop', 'firestick', 'roku', 'wireless',
                         'wifi', 'broadband', 'dsl','amazon', 'google', 'instagram', 'twitter',
                         'hulu', 'alexa', 'camera', 'feed', 'service', 'network','youtube',
                         'internet', 'fiber', 'delay', '5G', 'port', 'ethernet', 'pole', 'wire', 'data']

         # List of words that likely are not associated with an internet outage
         not_out = ['good', 'pleased', 'fast', 'cellular', 'data', 'transformer', 'candle', 'lights',
                    'solar', 'charge','generator', 'substation', 'energy', 'electrical', '4G',
                    'apple', 'microsoft', 'monitor', 'outlet', 'refrigerator', 'lamp', 'clear',
                    'sunny', 'happy', 'love', 'recommend', 'microwave', 'fine', 'power', 'electricity']
```

```
In [50]: #vectorizing the keyword lists
         int_out_vec = vectorize_corpus(internet_out)
         not_out_vec = vectorize_corpus(not_out)
```

```
In [51]: #scoring words against our keywords
         points['int_out_cs'] = [cos_sim(tweets2vec.word_vec(word),int_out_vec) for word in points['word']]
         points['not_out_cs'] = [cos_sim(tweets2vec.word_vec(word),not_out_vec) for word in points['word']]
         points['int_out_label'] = np.where(points['int_out_cs'] >= points['not_out_cs'],'internet_out','not_out')
         points.head()
```

Out[51]:

	word	x	y	int_out_cs	not_out_cs	int_out_label
0	outage	0.767890	-0.101621	0.019542	0.073687	not_out
1	night	0.571323	1.574581	0.044637	-0.027141	internet_out
2	highvoltagelife	0.559318	-0.650164	-0.033985	-0.076112	internet_out
3	houston	0.152850	0.305594	-0.075634	-0.014513	not_out
4	texas	-0.372509	2.046023	-0.030317	-0.062772	internet_out

```
In [52]: #getting the counts for each label
         points['int_out_label'].value_counts()

Out[52]: internet_out    466
         not_out         450
         Name: int_out_label, dtype: int64
```

```
In [53]: #sorting to see what words appear up top for legitimate internet outages
         points[points['int_out_label'] == 'internet_out'].sort_values(by = 'int_out_cs',ascending = False).head(10)
```

Out[53]:

	word	x	y	int_out_cs	not_out_cs	int_out_label
556	video	-1.578168	1.635520	0.315977	-0.041190	internet_out
497	delay	-0.303637	0.802018	0.280189	-0.035587	internet_out
30	equipment	-0.514592	1.297351	0.267866	0.094864	internet_out
830	modem	-0.541725	0.373455	0.260245	0.049859	internet_out
29	due	-0.164139	0.123715	0.247240	-0.051659	internet_out
71	internet	-1.988128	0.312822	0.246165	0.073955	internet_out
542	youtube	-1.458736	0.805571	0.244322	0.053591	internet_out
811	website	-0.185817	0.944106	0.243216	-0.032083	internet_out
305	netflix	-0.520431	0.573648	0.228365	0.037804	internet_out
579	facebook	0.281340	0.810242	0.215657	0.074805	internet_out

```
In [54]: #seeing what the top are for not a legitamate internet outage
         points[points['int_out_label'] == 'not_out'].sort_values(by = 'not_out_cs',ascending = False).head(10)
```

Out[54]:

	word	x	y	int_out_cs	not_out_cs	int_out_label
598	lights	-2.288865	0.495534	-0.000042	0.437955	not_out
553	happy	1.016000	0.323567	-0.000061	0.349115	not_out
107	data	0.300680	0.799691	0.162450	0.329636	not_out
256	energy	-2.275940	-0.293185	0.035946	0.320510	not_out
773	transformer	-1.359230	-0.748184	0.003969	0.319025	not_out
8	power	-1.579908	-0.063433	0.058167	0.256880	not_out
254	electricity	-1.221049	1.154637	0.086767	0.255237	not_out
351	good	1.216639	0.200396	-0.021925	0.250118	not_out
441	fine	-0.444107	0.196404	-0.034617	0.247752	not_out
891	sunny	0.155963	0.515473	0.028662	0.221940	not_out

```
In [55]: #checking out what words are most similar to the word "internet"
         tweets2vec.most_similar("internet", topn=10)

Out[55]: [('airlines', 0.189323291182518),
          ('actually', 0.17067837715148926),
          ('customs', 0.1678716391324997),
          ('net', 0.1600463092327118),
          ('almost', 0.15606530010700226),
          ('thats', 0.14820802211761475),
          ('wont', 0.14021992683410645),
          ('todays', 0.13982287049293518),
          ('att', 0.1373782902956009),
          ('get', 0.13704143464565277)]
```

```
In [57]: #checking out what words are most similar to the word "outage"
         tweets2vec.most_similar("outage")

Out[57]: [('play', 0.20596256852149963),
          ('ac', 0.1617257446050644),
          ('dont', 0.15673181414604187),
          ('offline', 0.14364472031593323),
          ('tonight', 0.13697656989097595),
          ('power', 0.13692809641361237),
          ('hrs', 0.13625654578208923),
          ('id', 0.13333970308303833),
          ('inconvenience', 0.13173605501651764),
          ('car', 0.13120682537555695)]
```

```
In [ ]:  #taking a look at each tweet with both scores so we can review what word groups were used to create the score
         #this is helpful for fine tuning the model
         for tweet in final_tweet_list:
             print(tweet)
             tweet_avg_vec = vectorize_corpus(tweet)
             print(f'Internet Out CS: {cos_sim(tweet_avg_vec,int_out_vec)}')
             print(f'Not Out CS: {cos_sim(tweet_avg_vec,not_out_vec)}')
```

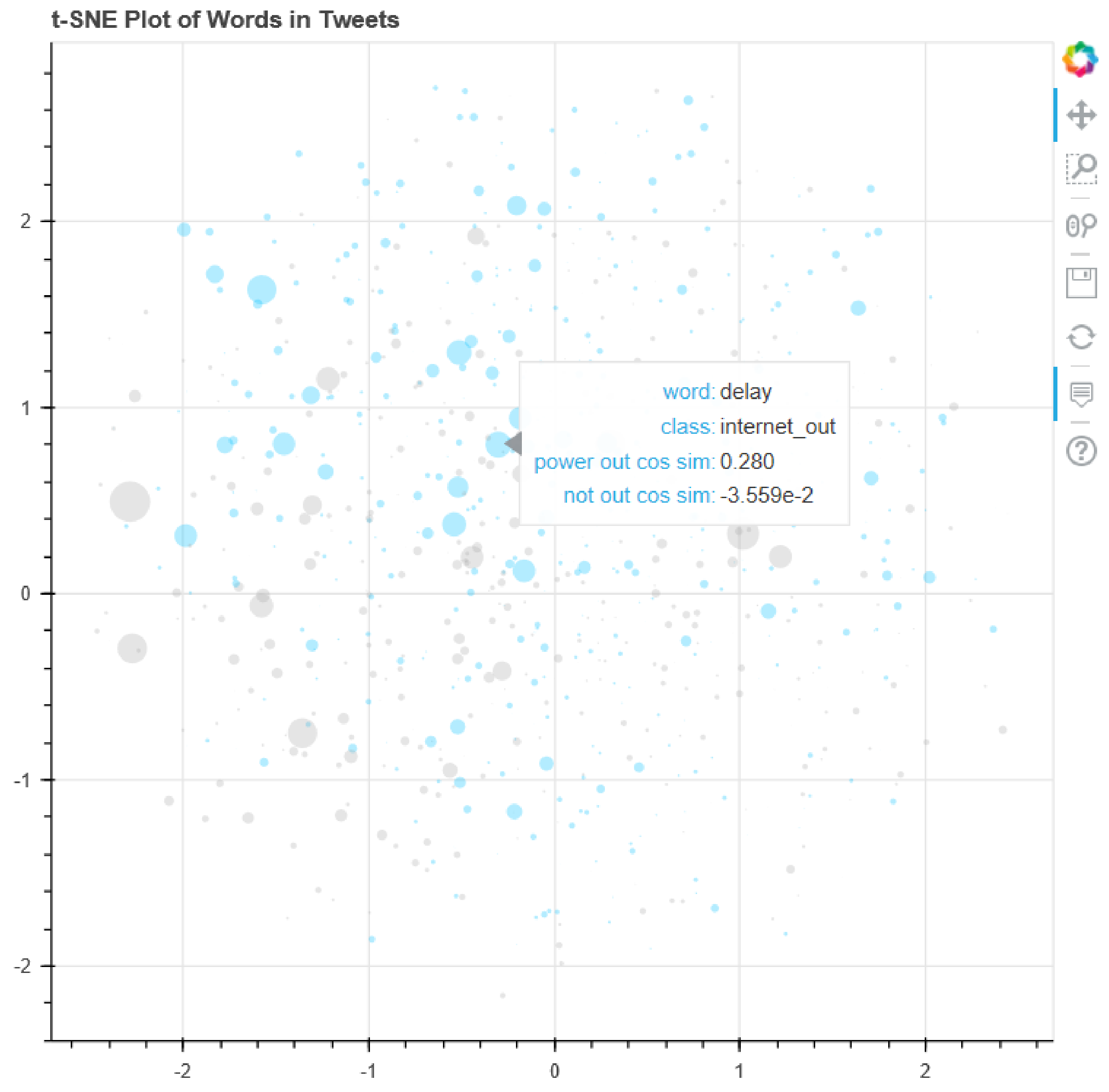
```
In [59]: #creating a dataframe of the scores, including noting areas of blackout
         tweets_df['score_int_out']     = [cos_sim(vectorize_corpus(tweet),int_out_vec) for tweet in final_tweet_list]
         tweets_df['score_not_out'] = [cos_sim(vectorize_corpus(tweet),not_out_vec) for tweet in final_tweet_list]
         tweets_df.head()
```

Out[59]:

	id	text	timestamp	user	location	datestamp	date_place	clean text	score_int_out	score_not_out
0	NaN	Outage night #highvoltagelife @ Houston, Texas...	2019-01-26 04:12:53	NaN	Houston	2019-01-26	2019-01-26, Houston	outage night highvoltagelife houston texas p b...	-0.034959	-0.050093
1	NaN	Outage night #highvoltagelife @ Houston, Texas...	2019-01-26 04:12:53	NaN	Houston	2019-01-26	2019-01-26, Houston	outage night highvoltagelife houston texas p b...	-0.034959	-0.050093
2	NaN	Unfortunately Pimlico and surrounding areas ar...	2019-01-22 23:19:55	NaN	Houston	2019-01-22	2019-01-22, Houston	unfortunately pimlico surrounding areas curren...	0.007947	0.070330
3	NaN	It's been a while but looks like an outage nig...	2018-07-28 01:39:30	NaN	Houston	2018-07-28	2018-07-28, Houston	looks like outage night highvoltagelife housto...	-0.030997	-0.034920
4	NaN	It's been a while but looks like an outage nig...	2018-07-28 01:39:30	NaN	Houston	2018-07-28	2018-07-28, Houston	looks like outage night highvoltagelife housto...	-0.030997	-0.034920

```
In [60]: #saving the csv
         tweets_df.to_csv('final_scored_data.csv')
```

```
In [61]: #plotting the t-SNE
         plt.figure(figsize = (150,100))
         colors = {'internet_out':'deepskyblue', 'not_out':'darkgrey'}
         for i,word in enumerate(points['word']):
             x = points['x'][i]
             y = points['y'][i]
             color = points['int_out_label'].apply(lambda x: colors[x])[i]
             plt.scatter(x, y, color=color, s = 10)
             plt.text(x+0.01, y+0.01, word,color = color, fontsize=30)
         plt.show()

         # to see the plot please zoom in.
```

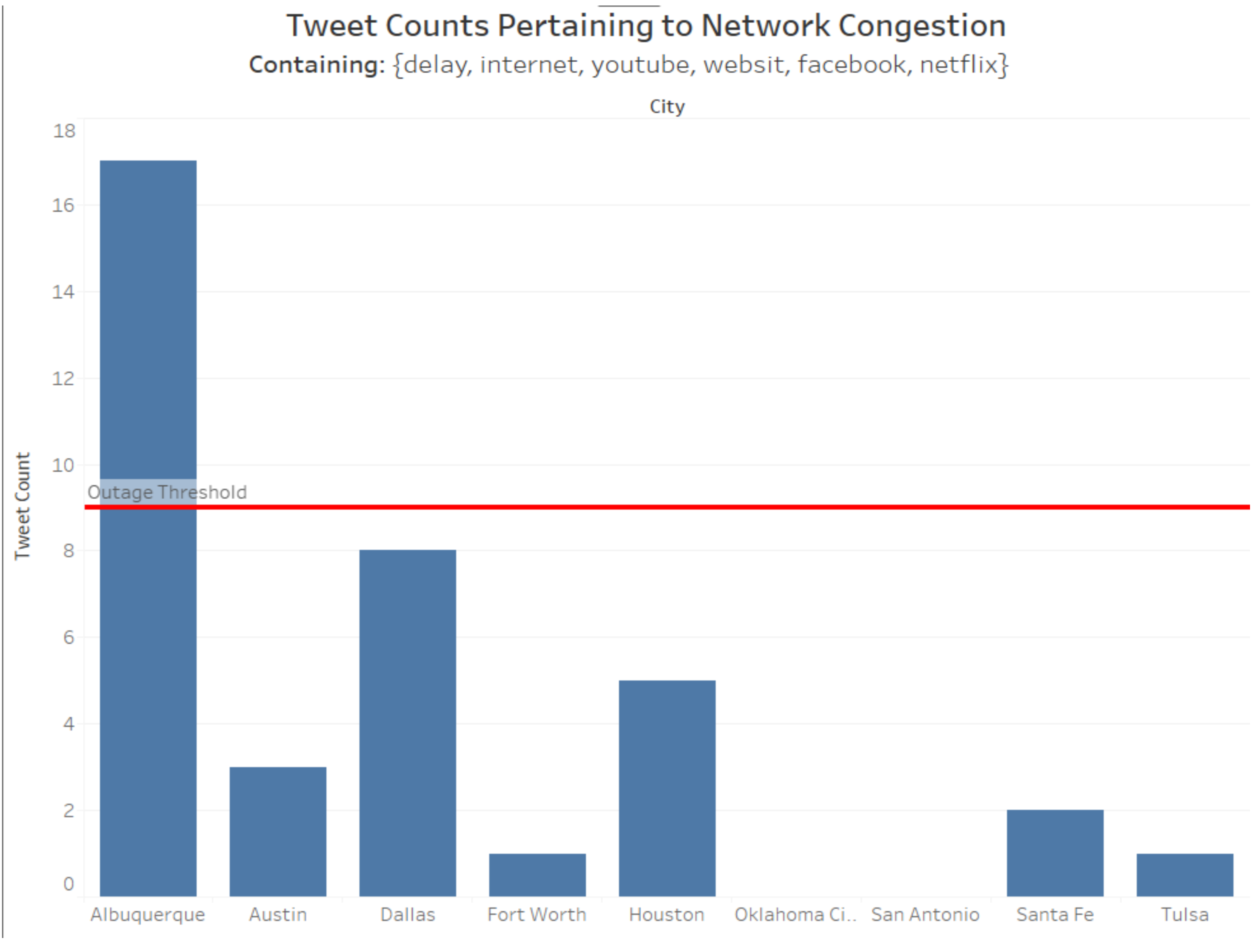


The descriptor hovering over one of the large blue circles indicates that the word ‘delay’ is more related to positive outage words than other words that appear as smaller blue circles. The grey circles indicate words that relate to negative outage words.

Appendix B: Python code to send emails

```python
SERVER = "smtp.example.com"
FROM = "yourEmail@example.com"
TO = ["listOfEmails"] # must be a list

SUBJECT = "Subject"
TEXT = "Your Text"

# Prepare actual message
message = """From: %s\r\nTo: %s\r\nSubject: %s\r\n\

%s
""" % (FROM, ", ".join(TO), SUBJECT, TEXT)

# Send the mail
import smtplib
server = smtplib.SMTP(SERVER)
server.sendmail(FROM, TO, message)
server.quit()
```

Appendix C: Example histogram for Tweet counts



This view shows Tweet counts related to outages caused by network congestion by city. The threshold for which a possible outage may be occurring is indicated by the red line.