Networks, Data Mining, and APIs

Josiah Manning

ID # 1566780

Biola University

Writing Competency

11/3/19

**Abstract**

This paper will discuss network graph theory and how it applies to data mining in social network APIs for Twitter, Instagram, and LinkedIn. *Graph theory* is the study of how network structures are represented. Data mining is the process of building and organizing databases from bulk data extracted from APIs. *APIs* (*Application Program Interfaces*) are sets of routines, protocols, and tools used to build applications as components to a larger program. A *RESTful* API stands for **RE**presentational **S**tate **T**ransfer, and follows the 6 attributes described on [restfulapi.net](https://restfulapi.net/). A *node* is defined as an individual member object of a network. An *edge* is an arbitrary link or connection between two nodes. Two nodes are *neighbors* “if they are connected by an edge” (Easley and Kleinberg 23). *Graphs* are a visual technique used to display network structures, and consist of nodes and edges, typically demonstrated as labeled circles or points connected by lines. *Edit distance* is “measure of how many insertions, deletions, and replacements it would take to convert one string into another” (Russell and Klassen 164). *Jaccard distance* is the measure of the similarity between two sets.

**1. Introduction**

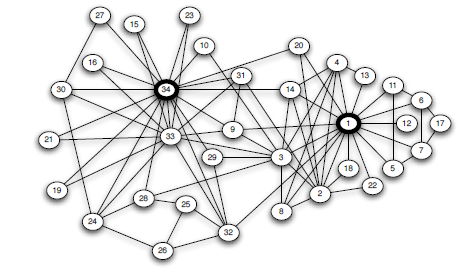
Social media websites like Twitter, Instagram, LinkedIn represent more than just places to post status updates for your followers, share cat videos and selfies, or maintain personal or professional relationships. They are networks of data interactions that can be traced and graphed by means of the website’s Application Program Interface (API). Data analysts can use these APIs to obtain and catalogue bulk data and apply it to graph theory, which can be used to identify and predict network groupings and relationships, an essential strategy to many internet-marketed industries.

While often restricted by individual sites, much data is open to the public, such as hashtags, keywords, and locations in Twitter’s case, image content identified by neural networks, tags, and descriptions in Instagram, and job titles, names, and locations in LinkedIn. The specifics and methods of API access and overviews of the top social media sites’ network structures are detailed in *Mining the Social Web* 3rd Edition by Matthew A. Russell and Mikhail Klassen, and network theory with specific attention to graph theory is covered in *Networks, Crowds, and Markets: Reasoning About a Highly Connected World* by David Easley and Jon Kleinberg.

In this paper, first we will cover aspects of graph theory according to Easley and Kleinberg, explaining how social media data structure can be categorized by graph theory concepts to identify new network information. Next we will discuss data mining and API concepts with reference to Appendix A and B. Then we will cover data similarity and clustering techniques, which can define groupings and trends beyond what can be inferred visually from graphs. Finally we will explain the technical code of Appendix A from start to finish.

**2. Graph Theory**

Graphing the structure of a network in various ways can reveal visual insights not explicitly described from the network’s attributes. When edges are given numerical costs or distances, various search algorithms can be performed across them to determine the length of different paths, such as breadth-first search (Easley and Kleinberg 32). When these edges are represented to scale based on distance or quantity, nodes are physically distributed based on location, and the graph gives a visual depiction of the relationships between each node.



**Figure 1: Easley and Kleinberg reference a network of a 34-person karate club, with nodes #34 and #1 representing two key figures, and edge distance representing quantity of interactions, with shorter edges meaning more interactions, and longer lines meaning fewer (p2).**

As you can see in Figure 1, there are two distinct groupings surrounding #1 and #34 which are not connected, nor many of their immediate connections directly connected to the other. As Easley and Kleinberg mention, “this pattern of non-interacting clusters was the most visible symptom of a conflict between them and their factions that ultimately splintered the group into two rival karate clubs” (9). This information was inferred based on the quantities of interaction, and relevant graphing procedure, without interviews or any other factors.

Many different kinds of graphs exist for different purposes. Collaboration graphs display networks of joined collaborators in different settings like members of teams, actor in a movie, or Wikipedia editors. These networks are interesting to sociologists because “they form detailed, pre-digested snapshots of a rich form of social interaction that unfolds over a long period of time” (55). Who-talks-to-Whom graphs record interactions between individuals, and Figure 1 is an example of this graph. Inferences can be drawn about the nature of a relationship based on the number of interactions, as well as the length of each interaction. Regrettably to data miners but fortunate to most of humanity, this data is always protected behind privacy screens in the largest social networks, at least from those outside. Information linkage graphs display the connectedness of web pages via links, and search engines make use of this data to determine the relevancy of websites, among other factors such as text and usage data(427). Web-based businesses depend on maintaining the relevancy of their links in order to maximize their connections in the web.

**3. Data Mining and APIs**

Merriam-Webster defines data mining as “noun: the practice of searching through large amounts of computerized data to find useful patterns or trends.” Social media websites record large amounts of such data, which can be accessed in large amounts by means of iteratively accessing dump information from their respective APIs, according to guided heuristics. Many large sites have one or more APIs for creating apps, and Twitter’s API has the fewest restrictions among the sites described in Russell and Klassen, so it’s the one I decided to use for Appendix A, for a more enlightening study of data mining.

Russell and Klassen walk readers through the APIs of social media sites Twitter, Facebook, Instagram, and LinkedIn. Facebook used to have a fairly open API, but due to privacy concerns, they restricted the API to basic information, and the API can’t retrieve user data without the user’s express permission. Instagram’s API is fairly accessible, with similar hashtag information as Twitter, but restricted by users with privacy settings. Russell and Klassen discuss how neural networks can be employed across the pixels of images in order to identify picture content, and Instagram uses these neural networks to identify explicit or illegal material. Google’s Vision API is suggested as an optional neural network, albeit at industry pricing (p129). All of them use the OAuth authentication process detailed in lines 7-10 of Appendix A, and the keys are generated from the app created on the developer sides of their respective websites.

**4. Twitter’s Search API**

Twitter’s RESTful API is accessed as most in the industry with the OAuth 2.0 authentication protocols, and is included in the python packages *twitter* and *tweepy.* For Appendix A, I used Twitter’s developer site to create an app on the site linked to my personal Twitter account, with a unique keys generated by the app for identifying and verifying the server-side app from the client-side python code, with the *twitter* package. Twitter’s API allows for a number of keyword arguments for GET and POST queries relating to the various member variables in a tweet, including hashtags, urls, favorited, retweeted\_status, and even follower\_count, profile\_link\_color, and time\_zone (Russell and Klassen 40-44). Tweet metadata is stored in json strings, with kwargs to organize and retrieve variables. Python’s *twitter* package provides functions and data structures for user-friendly API access, and these are further demonstrated and detailed in Section 5.

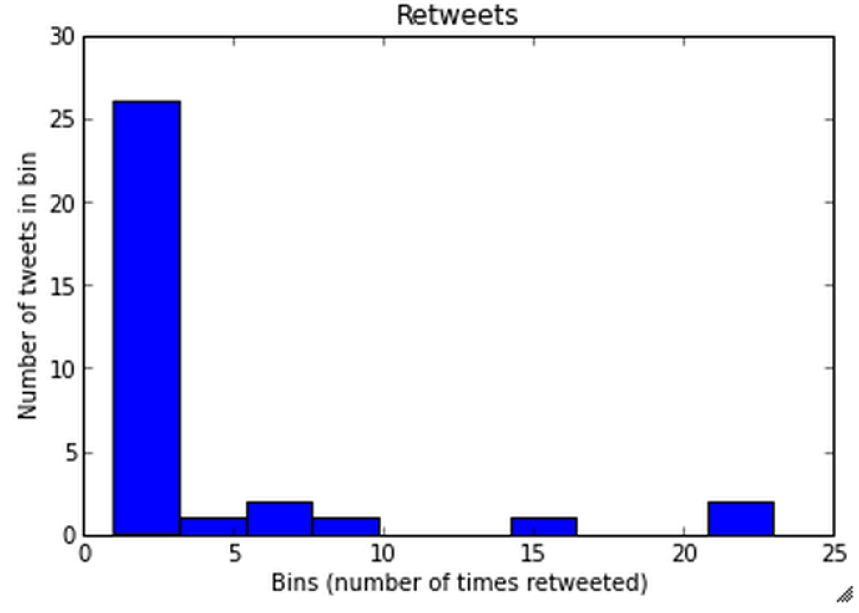
Appendix B demonstrates tables of word counts, screen names, hashtags, and locations of the top 30 tweets related to the hashtag #epsteinmurder, which demonstrates an original example of data mining the Twitter Search API with Python scripting. With normalization, we can see in the Locations table that Twitter users in Florida, Texas, and California list their location at a much higher ratio than others, and most simply don’t include their location. Additionally, without any other information concerning the Epstein murder case, we can see from the Screen Names table that names such as ABC news, Bill and Hilary Clinton, and the user opeoluwa2k appear most frequently in tweets that contain the hashtag #epsteinmurder. By looking at the Hashtags list, we see tags also posted in tweets containing #epsteinmurder, such as #epsteindidntkillhimself, #epsteincoverup, #ExposeABC, and further down the list #ClintonBodyCount, as well as different spellings of each other. I considered normalizing the hashtags, but because each hashtag leads to a different page of data, I elected to leave the hashtags as they were. Either way, we can identify trends by their accompanying hashtags, word usage, and locations, and we can identify participants by screen name popularity. This relates to data mining because all this information was inferred by grouping and graphing bulk data without any prior preconceptions about the issue.

**4. Data Organization Techniques: Similarity Calculation and Clustering**

When hardcoded normalization falls short, algorithms can be employed to calculate the similarity of strings. Russell and Klassen introduce the edit and Jaccard distances as means to calculate the similarity of different data structures, which can then be applied through the nltk Python library for frequency distribution (p164-166). Normalization is a necessary prerequisite to clustering, whether hardcoded or by means of algorithm. The transforms list of tuples in Appendix A shows the arbitrary normalization transforms applied to the top 10 pages of entries.

Clustering is “unsupervised machine learning technique [that] involves taking a collection of items and partitioning them into smaller collections” (Russell and Klassen p148). These collections follow a heuristic to identify common traits like location, genre, or income range, which is essential when working with immense quantities of bulk data. Greedy clustering is generated by means of a “thresholded Jaccard similarity metric” (p166), and registers an object in the first eligible cluster it finds, and hierarchical/k-means clustering forms a tree structure to identify distances between items in the cluster (p173). Appendix B demonstrates normalized tables, reducing multiple spellings of the same into single entries. Unfortunately, the location member variable of tweets are optional, and this is represented by the 411 tweets without a listed location. Additionally, since locations are unverified, certain entries such as “Collapsistan”, “Earth”, “USA”, and “Blue Planet ~ Darug Country,” are unhelpful at best and nonexistent at worst.

Russell and Klassen also discuss how frequency data can be graphed in histograms and other graphing methods using the python matplotlib library (55-61). They explain the benefits of using histograms as able to “group together data values into bins that correspond to a range of frequencies” (58). These groupings can help identify which words make up the majority of a tweet.



**Figure 2: Russell and Klassen demonstrate a histogram of retweets based on their similar code for the hashtag #MentionSomeoneImportantForYou (p60). Regrettably, matplotlib was not currently up to date with the latest python release at the time of this paper, so my attempts to do a similar display led to kernel crashes and .dll errors that I couldn’t resolve.**

As mentioned in Figure 2, I was unable to demonstrate a similar graph for visual clarity of Appendix A, but was able to display counts of each object, which will have to suffice for the purposes of this study, and hopefully their numerical analysis is helpful for understanding. I explained the information inferred from Appendix B at the end of Section 3. Similar graphs can be generated from other numerical data.

**5. API Source Code Walkthrough**

In this section we will analyze Appendix A, which is an original demonstration of accessing the Twitter Search API in order to mine data related to the #epsteinmurder hashtag. The first 4 lines import a few of python’s twitter API, display organization, and counting packages. The next section titled #API AUTHENTICATION VARIABLES defines the OAuth tokens, generated by the Twitter developer app I made on their website, each of which is parameters for the twitter package’s function twitter.oauth.OAuth(). These tokens provide identification and security for the app, as well as usability for developers. This function creates an auth object which is then passed as a parameter to create a twitter\_api object, which establishes a connection with the Twitter Search API app with aforementioned OAuth tokens.

The next section titled #IDENTIFY TRENDING TOPIC POSTS defines the hashtag and the number of search results to retrieve, which it passes into the search\_results object by means of twitter\_api.search.tweets(), and also defines the statuses by accessing the keyword argument ‘statuses.’ The twitter\_api object is a json string, which identifies member data with kwargs, which can be found by means of a json.dumps() command, or through the package’s documentation. A simple loop in lines 24-28 retrieves search metadata for the first ten results, with exception handling in the event of invalid results. This loop also creates a dictionary on line 30, which then organizes the json metadata into statuses within a list.

In the section titled #TWEET ANATOMY on line 36, we break the json metadata into lists of status\_texts, screen\_names, hashtags, locations, and words, which are each generated by iterating through the keyword arguments of the statuses object in lines 37-53. Then we define a list of string mappings on lines 56-70 in order to normalize all the location names, because the locations variable of each tweet is manually entered by the user, so spelling and semantics differ. Lines 72-74 apply the transforms to the list of locations. Tweets contain many more possible keywords which can be used to mine other details.

Finally, the data is ready to be displayed. The program outputs four tables using the PrettyTable package, organizing the tables by their usage within the top 30 results, and printing the results to an ‘output.txt’ file, as well as printing it to the console. As you can see in Appendix B, the tables are displayed as printed, and the locations are normalized to prevent repeated rows of similar names. The normalization was hardcoded arbitrarily, but as previously mentioned in Section 4, a more sophisticated program could apply an edit and Jaccard distance algorithm to apply normalization automatically. You may also notice a discrepancy between the Words table counts and the Hashtag tables counts, and this is because the Word table only records the raw text words and not the hashtag data, while the Hashtag table records both.

**6. Conclusion**

APIs are powerful tools that give access to large quantities of bulk data for analysis, and the analysis is a science. We discussed the fundamentals of graph theory and how groupings can identify trends. We examined the concepts of data mining social media networks by accessing their APIs in Python, and talked briefly about several major APIs. Then we explained the functions of Twitter’s API in greater depth, referencing the source code and output for the details of Appendix A. Then we discussed data organization and explained how similarity algorithms work, as well as clustering techniques, and how normalization was implemented in Appendix A’s code. Finally, we did an explanation of OAuth 2.0 and the python twitter package, and how Appendix A’s code works. These are essential topics to master for any data miner, or company interested in web-based market trends.

**7. Addendum: TF-IDF and Cosine Similarity**

As a final comment on the functions of data mining, we’ve covered raw data analysis, normalization, and organization techniques, but we can also make sense of complete commentary with natural language processing (NLP), using packages like Natural Language Toolkit (NLTK) and concepts like Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF is a method ”used to query a corpus (collection) by calculating normalized scores that express the relative importance of terms in the documents” (p190). In our experiment we identified important terms manually, but algorithms can be used to identify them as well, and streamline the data mining process. TF-IDF works by identifying and scoring the frequency of normalized terms in documents, while implementing a stopword metric for rigid normalization. Rare words of low TF-IDF scores are registered as stopwords, and are removed from potential normalization. Additionally, phrases can be scored; not just words, and document content can be classified by the high TF-IDF scores of bigram phrases. Bigrams contain far more useful information than simple tokens. When these scores are plotted on a graph, the cosine between the two points approaches 1 as the documents become more similar, which can allow for a simple clustering mechanic on a full document scale. However, TF-IDF doesn’t take word order or context into consideration, which is much more difficult to parse, and requires artificial intelligence algorithms to implement.

**8. Operations Research Review and Project Summary**

I spent approximately 60 hours researching, experimenting, and programming for this project. I read 90% of *Mining the Social Web* and about 50% of *Networks*. During the course of this project, I wrote Python programs in Jupyter Notebook which accessed the APIs of Twitter, Facebook, Instagram, and LinkedIn, and used matplotlib to graph data mined from them. For this report, I composed an original program which endeavored to accomplish a wider degree of analysis than the steps instructed in *Mining the Social Web*. I identified concepts in *Networks* to apply to my studies. As a result of this project, I demonstrated data mining fundamentals and network concepts by means of normalizing and organizing data mined from Twitter regarding the #epsteinmurder hashtag. From this hashtag I identified connections in the form of keywords, screen names, other hashtags, and locations, and theorized implementing Google Geocoder API in order to plot tweet locations. I retrieved relevant information, and made conjectures concerning the #epsteinmurder hashtag based exclusively on network observations, and no external insights. Finally, the study touched on information retrieval theory and natural language processing for contextual conjectures in data mining. This report therefore demonstrates industry practices of data scientists and analysts.

**Appendix A: Twitter Data Mining Python Script**

|  |
| --- |
| 1. import twitter 2. from urllib.parse import unquote 3. from prettytable import PrettyTable 4. from collections import Counter 5. #API AUTHENTICATION VARIABLES--------------------------------------- 6. CONSUMER\_KEY = 'oDrAwJCwApR36whEvYaA1F4LF' 7. CONSUMER\_SECRET = 'ExbTVWrz2ji9TNACeDwJKnfsuTIPopMEnNDrRxHfKJebLq7shs' 8. OAUTH\_TOKEN = '1175128015296065536-hH4p3rcZlFrlBr7KE1y3IOq3CQOT8l' 9. OAUTH\_TOKEN\_SECRET = 'CHAQos4iEK0VVHUN4gYMfeX5LUwtbn12i5KjrnaO0JQco' 10. auth = twitter.oauth.OAuth(OAUTH\_TOKEN, OAUTH\_TOKEN\_SECRET, 11. CONSUMER\_KEY, CONSUMER\_SECRET) 12. twitter\_api = twitter.Twitter(auth=auth) 13. #IDENTIFY TRENDING TOPIC POSTS------------------------------------ 14. q = '#epsteinmurder' 15. count = 100 16. search\_results = twitter\_api.search.tweets(q=q, count=count) 17. statuses = search\_results['statuses'] 18. for \_ in range(10): 19. try: 20. next\_results = search\_results['search\_metadata']['next\_results'] 21. except KeyError as e: # No more results when next\_results doesn't exist 22. break 24. kwargs = dict([ kv.split('=') for kv in unquote(next\_results[1:]).split("&") ]) 26. search\_results = twitter\_api.search.tweets(\*\*kwargs) 27. statuses += search\_results['statuses'] 28. #TWEET ANATOMY---------------------------------------------------- 29. status\_texts = [ status['text'] 30. for status in statuses ] 31. screen\_names = [ user\_mention['screen\_name'] 32. for status in statuses 33. for user\_mention in status['entities']['user\_mentions'] ] 34. hashtags = [ hashtag['text'] 35. for status in statuses 36. for hashtag in status['entities']['hashtags'] ] 37. locations = [ status['user']['location'] 38. for status in statuses] 39. words = [ w 40. for t in status\_texts 41. for w in t.split() ] 42. #NORMALIZE LOCATIONS 43. transforms = [ 44. ('TEXAS', 'Texas, USA'), 45. ('Texas', 'Texas, USA'), 46. ('Southern Florida', 'Florida, USA'), 47. ('South Florida', 'Florida, USA'), 48. ('Middle Georgia', 'Georgia, USA'), 49. ('MI', 'Michigan, USA'), 50. ('phoenix', 'Phoenix, AZ'), 51. ('Hollywood, Los Angeles', 'Los Angeles, CA'), 52. ('Anywhere USA', 'USA'), 53. ('AnywhereUSA', 'USA'), 54. ('Ohio', 'Ohio, USA'), 55. ('United States of America', 'USA'), 56. ('United States', 'USA'), 57. ('USA ', 'USA')] 58. for i, \_ in enumerate(locations): 59. for transform in transforms: 60. locations[i] = locations[i].replace(\*transform) 61. #DISPLAY TABLE OF DATA-------------------------------------------- 62. with open('output.txt','a') as myfile: 63. for label, data in (('Word', words), 64. ('Screen Name', screen\_names), 65. ('Hashtag', hashtags), 66. ('Location', locations)): 67. pt = PrettyTable(field\_names=[label, 'Count']) 68. c = Counter(data) 69. [ pt.add\_row(kv) for kv in c.most\_common()[:30] ] #top 30 results 70. pt.align[label], pt.align['Count'] = 'l', 'r' 71. print(pt) 72. table\_txt = pt.get\_string() 73. myfile.write(table\_txt) 74. myfile.write('\n') |

**Appendix B: Output of Appendix A code**

|  |  |
| --- | --- |
| **+--------------------------+-------+**  **| Word | Count |**  **+--------------------------+-------+**  **| RT | 816 |**  **| #EpsteinMurder | 722 |**  **| the | 553 |**  **| #EpsteinCoverup | 512 |**  **| #EpsteinSuicideCoverUp | 476 |**  **| Epstein | 331 |**  **| up | 267 |**  **| #ExposeABC | 261 |**  **| cover | 259 |**  **| News | 253 |**  **| #Epstein… | 253 |**  **| helped | 252 |**  **| @opeoluway2k: | 250 |**  **| .@ABC | 250 |**  **| scandal. | 250 |**  **| Disney | 168 |**  **| ABC | 151 |**  **| @HillaryClinton | 149 |**  **| was | 146 |**  **| a | 140 |**  **| #EpsteinSuicide | 135 |**  **| #EpsteinCoverUp | 132 |**  **| IS | 131 |**  **| #Epsteindidntkillhimself | 131 |**  **| THE | 126 |**  **| WHAT | 122 |**  **| @MarkTJay3: | 121 |**  **| .@BillClinton | 121 |**  **| ACTUAL | 121 |**  **| F-CK | 121 |**  **+--------------------------+-------+** | **+--------------------------+-------+**  **| Hashtag | Count |**  **+--------------------------+-------+**  **| EpsteinMurder | 735 |**  **| EpsteinCoverup | 521 |**  **| EpsteinSuicideCoverUp | 481 |**  **| ExposeABC | 262 |**  **| Epsteindidntkillhimself | 146 |**  **| EpsteinCoverUp | 139 |**  **| EpsteinSuicide | 135 |**  **| Epsteindidnotkillhimself | 88 |**  **| Arkancide | 82 |**  **| Epstein | 62 |**  **| PrinceAndrew | 46 |**  **| JeffreyEpstein | 44 |**  **| epstiencoverup | 36 |**  **| TuesdayThoughts | 30 |**  **| Bombshell | 27 |**  **| BillBarr | 27 |**  **| epstein | 24 |**  **| EpsteinIsland | 19 |**  **| EpsteinCoverupCoverup | 18 |**  **| ClintonBodyCount | 17 |**  **| LionelNation | 17 |**  **| QAnon | 16 |**  **| ClintonCrimeFamily | 15 |**  **| EPSTEINMURDER | 15 |**  **| EPSTEINGATE | 14 |**  **| Pedogate | 10 |**  **| EPSTEINSUICIDE | 10 |**  **| FakeNewsMedia | 10 |**  **| maga | 10 |**  **| kag | 10 |**  **+--------------------------+-------+** |
| **+-----------------+-------+**  **| Screen Name | Count |**  **+-----------------+-------+**  **| ABC | 315 |**  **| opeoluway2k | 251 |**  **| HillaryClinton | 153 |**  **| BillClinton | 133 |**  **| MarkTJay3 | 121 |**  **| FollowsTruth | 120 |**  **| Jewel4Trump | 31 |**  **| Dougsjourney7 | 28 |**  **| Mac72Terry | 19 |**  **| LionelMedia | 17 |**  **| realDonaldTrump | 16 |**  **| arobach | 15 |**  **| ignitehealing | 12 |**  **| abcnews | 11 |**  **| SITSSHOW | 11 |**  **| Project\_Veritas | 10 |**  **| JamesOKeefeIII | 10 |**  **| ShawnHomerMusic | 10 |**  **| Deplorable80210 | 8 |**  **| Dani06548474 | 8 |**  **| thefreerifleman | 8 |**  **| hammy413 | 8 |**  **| jamesgoldston | 7 |**  **| brianstelter | 7 |**  **| mostlikedtees | 6 |**  **| JEFFKANGSTA | 6 |**  **| AgoristN | 6 |**  **| chetbtester | 6 |**  **| VerseCannon | 6 |**  **| jkbjournalist | 5 |**  **+-----------------+-------+** | **+-----------------------------+-------+**  **| Location | Count |**  **+-----------------------------+-------+**  **| <no location> | 411 |**  **| USA | 88 |**  **| Florida, USA | 26 |**  **| Texas, USA | 23 |**  **| California, USA | 13 |**  **| Georgia, USA | 10 |**  **| Los Angeles, CA | 9 |**  **| Michigan, USA | 5 |**  **| New Jersey, USA | 5 |**  **| San Antonio, TX | 5 |**  **| New York, USA | 5 |**  **| Blue Planet ~ Darug Country | 5 |**  **| San Diego, CA | 5 |**  **| Nigeria | 4 |**  **| Oregon, USA | 4 |**  **| Massachusetts, USA | 4 |**  **| Las Vegas, NV | 4 |**  **| Pennsylvania, USA | 4 |**  **| Stillness in the Storm | 3 |**  **| Phoenix, AZ | 3 |**  **| Louisiana, USA | 3 |**  **| Atlanta, GA | 3 |**  **| Colorado, USA | 3 |**  **| Tennessee, USA | 3 |**  **| Nashville, TN | 3 |**  **| Connecticut, USA | 3 |**  **| Earth | 3 |**  **| Ohio, USA | 3 |**  **| Collapsistan | 3 |**  **+-----------------------------+-------+** |

**References**

Easley, David, and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 2017.

Russell, Matthew A., and Mikhail Klassen. *Mining the Social Web Data Mining Facebook, Twitter, Linkedin, Instagram, Github, and More*. 3rd ed., O'Reilly, 2019.

Talada, Rob, et al. “REST API Tutorial.” *What Is REST – Learn to Create Timeless REST APIs*, 19 Aug. 2019, restfulapi.net/.