Data Mining Human Reasoning: Vaccine Hesitancy in the USA

**Taha A. Kashaf**

**ID: 180020889**

**AC40001 Honours Project**

**BSc (Hons) Computing Science**

**University of Dundee, 2022**

**Supervisor: Prof. J. Lawrence**

*The purpose of this project was to use Machine-Learning powered Sentiment Analysis and Natural Language Processing techniques to classify sentiment about vaccination from textual data in the form of tweets.*

*The project endeavoured to present the results of this analysis in a clear, easy-to-approach way. Allowing users/readers to be able to view the results alongside other statistical data and see correlations and comparisons for themselves.*

*The goal being to facilitate an understanding of potential underlying reasons for vaccine hesitancy, to be able to better address it in the future.*

*While many parts of the project were successful – data collections, formatting, cleaning, pre-processing and presenting. Ultimately, the Machine-Learning powered Sentiment-Analysis model wasn’t accurate enough to draw useful conclusions from – however, this was primarily due to the lack of training data, and the model can be improved with simply more training.*

# Introduction

Vaccinations and vaccines have become a controversial talking point in this day and age. In the midst of the worlds best-documented pandemic: COVID-19. Many of us saw the development of COVID-19 vaccines as a fantastic feat of collaboration between the global medical and scientific communities. We believed that with the help of the vaccines and large-scale inoculation, multiple years of lockdowns, restrictions, and sacrifices would come to an end.

However, while that may be the predominant opinion, many people also look at vaccinations and vaccines as a bad thing. Hence the talking point of vaccines being one that brings controversy to the table.

Anti-vaccination rhetoric has been around since long before COVID-19[1]. However, in recent years, and especially with the advent of social media, the visibility of the movement has grown dramatically.

In pre-COVID times, anti-vaccination rhetoric was primarily something we’d hear about and shake our heads at. For most, it did not have any tangible real life effect, it wasn’t likely to cause any changes to our day-to-day lives.

However, currently, in the midst of a pandemic. With reports of hospital urgent care wards being filled with primarily unvaccinated covid patients[2] the effects of the anti-vax movement are bigger and closer than ever. Where even a vaccinated individual may be unable to receive care due to hospital beds being occupied by those who choose not to take vaccines.

The problem is an obvious one: large groups of people carry a negative sentiment towards vaccinations. The goal of this project was simple: To use Machine-Learning powered Sentiment Analysis, in combination with Natural Language Processing techniques to aid in producing a data-driven solution that would help us understand *why* people have the opinions they do regarding vaccines.

This is obviously not a solution to vaccine hesitancy in itself. With a topic as complex as human reasoning, and why people choose the things they do. First, you need a comprehensive understanding of *why*. Only then, once that understanding has been built, can you begin to tackle the problem itself. This project is an attempt at understanding the reasoning behind the problem.

# Background

My background research was split into three distinct categories, Data Analysis, Sentiment Analysis Techniques and Previous Attempts at Classifying Vaccine Sentiment on social media.

## Data Analysis

The field of data analysis is a large and still rapidly expanding one. One that has also come to the forefront of discussion in the last few years, as the power of data-analysis and data-driven solutions has been made clear to the general public – often in a negative light.

Perhaps the most famous example of how data analysis has changed the world is the role it played in recent large-scale political campaigns[3]. These successes have even led to legal debate regarding data privacy, and the uses of data analysis. The most well known result of such debate being the passing of legislation such as the General Data Protection Regulation (GDRP).

As we can see, there is no question about the importance or research potential within big data. The ability to analyse and extract value or meaning from large-scale unsorted data is an ever-growing field with proven real-world results.

## Sentiment Analysis Techniques

For the purposes of this project, I made use of Sentiment Analysis, a form of data analysis on written text. Sentiment Analysis comes in different forms[4], primarily one of two: Lexicon based approaches and Machine-Learning based approaches. Machine-Learning based approaches are then also split further into supervised, unsupervised, and semi-supervised.

Lexicon based sentiment analysis approaches are techniques that make used of a predefined list of words that have been assigned a specific association. These predefined sentiment lexicons are then used to assign a polarity value to each text document (for example, a tweet) by following a basic algorithm[5]. Predefined sentiment lexicons are often referred to as corpora or dictionaries.

Examples of popular lexicon-based sentiment analysis models include the “Valence Aware Dictionary and Sentiment Reasoner”, known simply as VADER[6]. VADER is even specifically tuned to sentiments expressed in social media, however, was not suitable for the project, as discussed in *Implementation and Testing*

Machine-Learning based sentiment analysis techniques do not use any form of pre-defined corpora. Instead, in the case of supervised machine-learning, models are trained using full text documents (tweets) that have been hand labelled into certain categories. In the case of sentiment analysis, these are often positive/negative/neutral. This was the approach I decided to take myself and is again discussed further in *Implementation and Testing.*

Semi-supervised, or hybrid approaches to sentiment analysis use a combination of lexicon, and machine-learning based techniques.

## Previous Attempts at Classifying Vaccine Sentiment

My project is not the first attempt at classifying vaccine sentiment, it is not even the first attempt at classifying vaccine sentiment on twitter specifically. In recent times, similar studies have been carried out[4][7][8]. However, these studies primarily focus on classifying data and presenting the classification results in a purely academic format, such as this report. While this provides an excellent base for that classification data to be used or extended for further study, this was not the direction I wanted to take my project in.

These previous projects and studies typically used a Semi-Supervised Machine-Learning methodology. Meaning a combination of Lexicon-Based and Supervised Machine-Learning techniques. This approach, combined with a high amount of training data lead to a high level of accuracy. In the case of one study, with 10,500 points of training data, a >85% accuracy was achieved in classification[4].

However, even with multiple studies with high levels of classification accuracy, my primary takeaway was that none of the information gathered was presented in a wide and easily accessible scope. It remained within the realms of academia, as mentioned earlier. This is where I wanted my project to be different, deciding from early on that I would present my results in a easily accessible, graphical format for anyone to be able to find and use. I discuss this further in the following *Specification* section.

# Specification

I split my project into four broad, high-level sections early on:

1. Data Collection
2. Data Formatting
3. Data Classifying
4. Data Presenting

Each of these sections also had distinct sub-sections, some of which I was aware of at the outset of the project, others I discovered as the project went on.

Due to this, the nature of the project was very exploratory for me. I was learning what was required as I was progressing through the stages. This made it very difficult to nail down a development methodology early on in the project. I discuss this further in subsection *Development Methodology*.

So, while parts of my specification were pre-planned right from the outset, as the project evolved and I learned more, the initial plans rapidly fell to the wayside. This led to me spending less time in the pre-planning specification writing stages, and more time just working. As time spent planning was often time wasted due to how quickly reality deviated from plans.

## Data Collection

The first part of any Data-Driven solution is to acquire the data. For this project, I would be using the Twitter API[9] for my data collection purposes. The project plan for this section broke down as the following:

1. Acquire Twitter Developer Account
2. Create Twitter API Application (needed for endpoints access)
3. Learn how to use the API 2.0 Endpoints
4. Use API to Collect Data.

However, as stated earlier in *Specification*, reality rarely went to the original specification, so the Data Collection part of the project went more like the following:

1. Acquire Twitter Developer Account
2. Create Twitter API Application (needed for endpoints access)
3. Escape from Twitter Spam Filter detection
4. Learn how to use API 2.0 Endpoints
5. Collect Data
6. Learn how to use API 1.1 Endpoints
7. Collect Data

The details as to why reality deviated from the original specification so dramatically are discussed thoroughly in *Implementation and Testing.*

## Data Formatting

The data formatting section was planned as the following, with no changes during implementation:

1. Data “Cleaning”
   1. Discarding extra information returned by the API that was unnecessary for the project goals.
2. Data Structure “Formatting”
   1. Splitting the tweets into individual text files, organizing them into a two-level folder structure that the Machine Learning model could use.
   2. This stage was done in conjunction with hand-labeling data for training
3. Data “Pre-Processing”[10]
   1. Remove all special characters
   2. Remove all single characters
   3. Remove single characters from start
   4. Replace multiple spaces with single spaces
   5. Removing pre-fixed “b” from loading function
   6. Converting to lowercase
   7. Lemmatization
   8. Covert Text to Numbers
      1. Done using “Bad of Words” Model.
   9. Removing Stopwords
   10. Finding TFIDF – (Term Frequency – Inverse Document Frequency)

## Data Classifying

Once all the data has been adequately formatted and pre-processed, the final classifying stage is fairly simple. Only consisting of the following:

1. Train Model
2. Evaluate Model
3. Save (“Pickle”) Model
4. Use Model to Classify remaining large-scale data

A few snags were hit in this area, and are discussed in detail in *Implementation and Testing*.

## Data Presenting

The final part of the project involved Data Presenting. This was making a full front-end website that users could browse to see the results of the project. As well as to see some comparisons of sentiment data alongside other statistics. I arrived at the decision to implement this part of the project based on background research. This was what would set my project apart from previous studies into Vaccine Hesitancy – a clear, non-academic, easily accessible way to view the classified data from the project.

This was further motivated by my first meeting with my project supervisor – Prof. John Lawrence, where we discussed how the project would be split on a back-end/front-end level (see Appendix A).

This would also be used as an opportunity for me to learn and become more familiar with JavaScript, and JavaScript libraries such as React.js[11]. While I had done front end development before, it had not been with simpler technologies, so this was a chance expand my skillset. As a result, I arrived at the following (loose) plan/specification for the website.

1. Learn React.js
2. Create landing page
3. Create “Comparisons” Pages
4. Present data graphically using Graph.js[12]
5. Achieve full deployment for website.

## Development Methodology

While *Development Methodology* is not explicitly related to *Specification* it is important in my case to understand the development methodology I implemented, as it was directly responsible for the very loose style of specification I arrived at and gave myself for the purposes of this project.

The initial plan for the project, before I ever had my first meeting or read my project brief, was to adopt an Agile Methodology to development. I had in previous projects adopted Agile practices to great success, and was a fan of how Agile allowed flexibility and rapid prototyping during development.

However, after my first meeting with my project supervisor, it became apparent to me that I knew very little about the space in which my project was going to be developed. I had never worked with the Twitter API, with Machine-Learning, with Sentiment Analysis or with Natural Language Processing techniques, even my plan to use React.js was so that I could have the opportunity to learn it. Every single stage of my research and development process would be like uncovering a fog-of-war on a map – I could only see what’s in front of me as I came up to it.

As a result, although I did initially plan sprints (see Appendix B) I quickly decided it wasn’t a good investment of time to generate detailed requirements or user stories, as these requirements would quickly be overridden during development – when I actually learned what is required, rather than guessing without much knowledge.

# Design

My design choices were across a number of areas, what languages to adopt, what frameworks to use, what tools would I be able to use, what constraints would I be working with and what systems would I be using for backup and version control.

Each of these decisions was considered for each of the main areas detailed earlier in *Specification* (Data Collecting, Formatting, Classifying, and Presenting).

However, before any of that could be considered, the first and simplest decision needed to be made regarding backup, code storage, and version control.

## Backup, Code Storage & Version Control

This was the easiest design decision out of them all. While consideration was given to services such as SourceForge[13], BitBucket[14], GitLab[15] and even just locally saving all my work. The final decision was very quickly to use the industry standard: GitHub[16].

Although I had been recommended the alternatives by my friends, and even initially wanted to just work locally from my own machine. I had prior experience with GitHub and decided that even thought the project was a solo one, and did not require any code-collaboration, I would use GitHub just because it provides a form of backup – just in case something was to go wrong. (To good effect too, as I did lose all my local work during the year due to a system malfunction on my main machine that required a full re-install)

## Data Collection

For Data Collection, right from the outset I had decided to use the Twitter API. This wasn’t much of a choice, although techniques such as JavaScript Web Scrapping exist for data collection, the Twitter API offers the most control and easiest level of access to Twitter Data.

Within the Twitter API, a decision has to made between using the version 2.0 Endpoints, and the version 1.1 Endpoints.

Initially, I decided on using the 2.0 Endpoints, as the version 1.1 Endpoints were both deprecated, poorly documented and no longer maintained.

To use the 2.0 Endpoints, as recommended by the Documentation I used the Postman API Platform[17] tool. The Postman API Platform is a tool for developers to design, build, test and iterate their APIs. It allows user to make requests to APIs from a graphical user interface. This was perfect for me, as I had never used the Twitter API before and a GUI-centric way of accessing it would be the most simple.

Twitter also supplied a pre-made collection[18] that made access to the Twitter API 2.0 Endpoints from postman very simple. While this worked, due to reasons discussed in detail in *Implementation and Testing* I was forced away from using the Twitter API 2.0 Endpoints, and had to pay for access to the 1.1 Endpoints, and use those.

The version 1.1 Endpoints provided by Twitter are both deprecated, and poorly documented. This meant there was no Postman collection available to allow the API to be accessed graphically. This lead to a steep learning curve as I was forced to adopt Ubuntu (via Windows Subsystem for Linux – WSL[19]) and cURL[20] (a command line tool for transferring data) as my main technologies for Data Collection.

## Data Formatting

The next design choice to be made was what technologies would I use for the expansive amount of data formatting that was required – in fact, this was probably one of the most key parts of the project. As such, the design choices made here would be consequential.

Luckily, these choices were somewhat made for me. During my first meeting with my project supervisor. I was strongly advised to use Scikit-Learn[21] for when I eventually moved on to the Machine-Learning and classifying stages. Scikit-Learn is a Python[22] library. On top of this, my own background research showed that Python was considered the go-to language for entry-level Machine-Learning and Data Analysis. So, Python seemed like the obvious choice, if I used it now during the Data Formatting stages, I would be more familiar with it during the Data Classifying stages.

However, although Python seemed like the obvious choice, I was also going to be using JavaScript later during the Data Presenting stage, when building my front-end. The Data returned from the Twitter API 1.1 Endpoints was also returned as JSON files – JavaScript Object Notation (see Appendix C). So, I was initially tempted to use JavaScript for all the Data Formatting purposes.

Despite this, my final decision was to use Python. As I learned that the Data Formatting and Data Classifying stages would be heavily intertwined, and switching languages between them would just be extra work for no benefit.

Within Python, a number of packages were used to aid in the Data Formatting. These include:

* NumPy[23], a library that adds support for large, multi-dimensional matrices and arrays, as well as a large collection of high-level mathematical functions to operate on those arrays.
* re[24], a base python module that provides regular expression matching operations
* Natural Language Toolkit (NLTK)[25], a suite of libraries and programs for natural language processing for English.
* pickle[26], a base python module for serializing and de-serializing Python object structure – essentially granting the ability to “save” trained Machine-Learning models to a file.

## Data Classifying

As stated in the previous subsection *Data Formatting*, the advice given during the initial project meeting (see Appendix A) was to use Scikit-Learn, and by extension, Python, for the Machine-Learning aspects of the project. My own research also showed that Python was by far and away the most popular language for entry-level Machine-Learning.

However, during my research, I also discovered alternatives to Scikit-Learn. Primarily in the form of TensorFlow[27], TensorFlow being a free and open-source library developed by Google that is available in Python. It can be used for a wide range of tasks but has particular focus on training and inference of deep neural networks and machine-learning.

TensorFlow also had a Sentiment-Analysis tutorial available in their own documentation, whilst Scikit-Learn did not. However, upon discussing with my project supervisor and being provided a number of resources (see Appendix D) for Sentiment-Analysis using Scikit-Learn, I did finally settle on that.

For the purposes of the model itself, I settled on using a Random Forest Classifier algorithm. While the Random Forest Classifier algorithm creates models that are slow to train and require higher processing power. This wasn’t a large problem on my end, as the model would only need to be trained once, before being pickled. Random Forest Classifiers are also good at working on large datasets, my project was designed to be easily extendable, so I would need a model that could cater to a growing dataset, which the Random Forest Classifier was suited for. The following reasons, as well as Random Forest Classifiers being recommended by various tutorials and articles, led to that algorithm being the one I very quickly settled on for training and classification purposes.

## Data Presenting

For the Data Presenting portion of my project, I was certain from the start that I would be making a website as my front end. I also knew that I wanted to use this as an opportunity to learn and work in JavaScript. As a result, although a number of JavaScript frameworks such as Angular.js[28] and Vue.js[29] are considered “better” or “easier”[30], I decided on using React.js[11].

This was because regardless of ease of use, or being told other frameworks were “better”, React.js is by far the most popular web framework in use today[31]. Which means it is a valuable technology to be proficient in as a developer.

As a result, it didn’t get given much further considering, I would be using React.js for my website/front-end. This was not the only technology used however, several packages were used alongside React.js to develop the final website, including:

1. Bootstrap[32], a free and open-source CSS framework for responsive front-end web development
2. Chart.js[12], a free and open-source JavaScript library for data visualization
3. React-Router[33], a library for routing in React.
4. React-Twitter-Embed[34], a library for easily embedding tweets, allowing you to show tweets without violating Twitters style guide/policy.
5. React GitHub Pages[35], scripts that allow deployment of React Apps to GitHub Pages. Allowing a fully-deployed website.
6. React-Select[36], a flexible input select control library for React.js

# Implementation and Testing

Since the project is easily divided into the four main sections seen previously in *Specification* and *Design*, the *Implementation and Testing* section will follow the same structure, starting with Data Collection.

## Data Collection

While this should have been a very simple part of the project, due to the clear idea of the type of data I wanted to be collecting, it ended up being the section that took the largest period of time, both because of the learning curve associated with using the Twitter API for the first time, and also with factors that occurred outside of my own control with the API, and with my API access.

### The Trouble with Twitter

Whilst the Twitter API puts a large sea of data right at the fingertips of developers, accessing this data is a challenge beyond just the technical aspects of interacting with the API.

I first made my application for a Twitter Developer account in March of 2020, well before my 4th year or before this project. I include this piece of information as that application was not approved until over 6 months later. This should give a clue about how quickly things move at Twitter.

This became a severe issue during implementation when I received the following email seen below, in *figure 1*.

Graphical user interface, text, application, email

Description automatically generatedFigure 1: Email from Twitter informing that my API Access had been suspended.

This email began an approximately 6-week period in which little to no development work was able to be done, as I had been locked out of being able to complete step one for a data driven project: collect my data.

I read the developer policy, and the API Policy back-to-front to make sure I wasn’t in violation of any rules. Which fortunately I wasn’t. However, I already knew this as my API access was suspended without hours of creating an application and receiving API keys – before I’d ever even sent one request.

With little else to do, I filled out an Application Appeal, and waited. After a few days had gone by, I had received no response, not even an acknowledgement of my appeal being received. At this point I deleted the Application that had been suspended, and tried making another one – just in case I had done something wrong in the creation process.

This changed nothing, and once again, within hours of creating the Application and receiving API Keys, before ever sending a request, I received another email identical to the first. Detailing that my Application had been suspended from accessing the API. Once again, I filled out the Appeal, once again I received no acknowledgement of any kind from Twitter.

At this point, knowing how long it took my Developer Account to be approved in the first place, I decided to take matters into my own hands (with what little I could do), and search for other people who had similar issues with Twitter, hoping someone had found a solution or explanation.

After weeks of deep diving the Twitter Developer Forums, which were filled with complaints of the same nature, but no solutions, I found one random comment in one random thread explaining that Twitter automatically suspends all applications they expect are coming from bot accounts, and does not consider the any appeals for these applications, or these accounts.

At this point, I put two-and-two together. Realizing that my Twitter Developer Account was linked to my normal Twitter Social Media account (as all Developer Accounts need to be). My normal account had no activity on it, as I do not use Twitter. No tweets, no comments, no followers, no activity of any kind. So, as a result, I was being considered a “bot” or “ghost” account, and having my applications suspended and my appeals ignored.

The solution for this, quite painfully, was for me to resurrect long since dead and inactive social media accounts, so that I could beg people I know to follow me. All in the hopes that my Twitter Account would meet the bare minimum threshold required to not be considered as spam. These embarrassing messages are captured in *Figures 2* and *3*.

Text

Description automatically generated Figure 2: I beg for followers on Discord.

Text

Description automatically generatedFigure 3: I beg for followers on Facebook

After these messages, I waited patiently for a number of days, to allow people to see them and follow me before I attempted again to submit an Appeal. When I did finally submit the Appeal to have my API Access restored, I did this time receive an acknowledgement that my appeal had been received, not an unsuspension, merely an acknowledgement (see Appendix E).

Even after that, it took a further eight days, till the 12 of January for my Application to finally be unsuspended, and for my API access to be returned (see Appendix F). Overall, this ordeal during the *Implementation* stage cost me approximately 6 weeks of development time, from early December to almost mid January.

### Twitter API v2.0 Endpoints

After receiving API access back, it was time to rapidly move on and begin collecting Data. I had settled on a plan to collect Tweets from each state in the USA that contained one of the following key words: “Vaccination”, “Vaccine”, “Vaccinated”, “Vaxed” or “Vaxxed”.

Graphical user interface, text, application, email

Description automatically generatedFigure 4: The GUI Presented by Postman to simplify making requests to the API

Whilst I was successful in using Postman (as seen in *figure 4*) to make requests to the API and to retrieve data, I hit a snag that unfortunately I was unable to solve itself. The API access that I had been given was akin to a “Student” Level account, elevated from normal API access, but still without all the bells and whistles available to me. This led to two issues. The second of which I would only become aware of further down the line.

The first issue, and one that was initially very worrying was that I did not have access to any form of Geographical Data. I was able to query Tweets that contained specific words, like vaccine, but I was unable to query for those tweets to also be limited to a certain State – like New York. This put a massive hole in my plans, as the entire idea behind the project was to present comparisons of Vaccine Hesitancy alongside other statistics, which I had planned on doing my comparing and contrasting different American states.

While this first issue was never resolved itself, as that would require “Researcher” level access to the API (which I did apply for, and still have not received any sort of response), it could be circumvented. This circumvention/solution was provided by the project supervisor, Prof. John Lawrence, during one of our meetings. Whilst I was unable to restrict tweet sources geographically, I could still restrict them based on what was IN the tweet. So the solution because to simply search for tweets that mentioned both one of the vaccine related key words, and the name of the state I was collecting data for.

Text

Description automatically generatedFigure 5: 7-Day Results for Arkansas provided by the v2.0 Endpoint

This was when I was presented with the second of my two issues with the API v2.0 Endpoints. Once again, due to my limited access level to the Twitter API, I was unable to access the “full-archive” search option that was provided, I was only able to query the “7-Day Archive”. I didn’t initially think this was a huge problem, as my goal was only to collect 500 tweets from each state, for a total of 25,000 data points. I assumed far more than 500 tweets about vaccines (containing state names) would be made in such a time period. However, I was very wrong. The query that returned the most results was for the District of Columbia, and even that contained only a measly 98 tweets. For most states, the results contained less than 10 Tweets, and for some, no tweets were found at all. *Figure 5* shows the results obtained for Arkansas – a grand total of 2 Tweets. The full data collected using this method can also be accessed in *Appendix G*.

At this point I realized that the v2.0 endpoints would not be able to provide what was necessary for this project. I would have to collect my data using either external sources (not the Twitter API), or figure out the v1.1 endpoints, which were both deprecated and poorly documented.

### Collecting Data Externally

Before attempting to figure out the v1.1 endpoints, I decided to look online to see if an existing data set was easily accessible for my purposes. I didn’t want to use a Data set that I didn’t collect myself, as a key part of this project for me was doing everything manually – and being unable to complete the data collection step and simply using existing data sets from the Internet was not part of my plan.

However, I needed to swallow my pride and have a backup ready before I invested time into the v1.1 endpoints, as at this point I did not know if that would work either, and if it didn’t I would be left at square one, with no data, no project, and half the time I had spent simply fighting with Twitter with nothing to show for it.

Luckily, I was able to find a passable dataset[37] on Kaggle[38], an online community of Data Scientists and Machine Learning practitioners (Kaggle also happened to be a subsidiary of Google).

This dataset wasn’t ideal, as it contained tweets from all over the world, some geo tagged, some not geo tagged, so I was unsure if I’d be able to filter out enough tweets from each individual state to suit my purposes. However, it did contain over 350,000 tweets (*see Appendix H*), so I was satisfied with having it as a last minute fall-back, should my efforts with the v1.1 Endpoints fail.

### Twitter API v1.1 Endpoints

The data that I finally did acquire, and use was from the Twitter API v1.1 Endpoints. As has been mentioned a few times, these endpoints are no longer maintained, and are not well documented.

A further issue during implementation with the v1.1 Endpoints was due to their deprecated nature – no existing collection existed for use with Postman. The API could only be accessed from the command line. However, this was not an insurmountable problem, merely a learning curve with regards to using cURL[20].

The issues that were presented with the v1.1 Endpoints were much like the issues with the v2.0 Endpoints. Primarily – I still could not access any for of geographical data, and still could not access the Full-Archive search feature. The search time was however increased from 7 days to 30 days.

For the geographical data, I was planning on using the same workaround as previously, however, as I still could not access the Full-Archive search, the ability to collect enough data was still a worry.

Luckily, after spending time learning the v1.1 Endpoints, I became aware of a potential solution that was not present with the v2.0 Endpoints. For the v2.0 Endpoints, the only way to access to the higher-level functions was to be approved for a Research or Enterprise level account, neither of which I qualified for (although I did apply for a Research account).

However, the v1.1 Endpoints offered the ability to purchase premium access, at quite a steep price. I was able to purchase full-archive search permissions for $100 USD/Month *(see Appendix I)*. This was both expensive, time-limited to one month, and still only offered 100 total requests to the API. So any bad requests, errors, or other faults would be slowly chipping away at the limited resources I had available.

After all of what I’ve described, the v1.1 Endpoints and paid premium access was what I ended up using, after spending a large amount of time searching through outdated documentation, the cURL requests that I needed for my purposes were figured out, and example of which can be seen below, in *figure 6*. The API Access token have of-course been redacted.

Text

Description automatically generatedFigure 6: cURL Script Used to Retrieve Tweet Data.

The script includes ‘“maxResults”: 500’, as this was the maximum amount tweet objects that the API allowed to be returned in one request.

Due to only having 100 requests total, and some of them being wasted whilst I was learning the correct syntax, I could only allocate 1 request per state – which led to 500 tweets per state. Or 25,000 total tweets collected. These tweets were returned in large JSON files – one for each state. An example of which can be found in *Appendix C*.

All the tweet data I received was stored on GitHub – alongside everything else in my project. This led to an ethical concern, as the data returned by the v1.1 Endpoint differed to the v2.0 Endpoint in that each Tweet was returned as a full Tweet Object. Meaning, all the information about the Tweet Author, and their account was also being returned. An example of just some of the extra data returned can be seen below in *figure 7*.

Text

Description automatically generatedFigure 7: An example of the extra user data returned with each tweet

I was worried about all the extra user data that was being returned and stored publicly in the project repository (*see Appendix J*). I raised this point during a meeting with my project supervisor, and the discussion agreed to the point that all the data being saved was also publicly available on the given Twitter accounts or Tweets. So, as long as I was not violating Twitter Policy, I was not ethically in any danger – as the data being saved is actually public data, not private.

## Data Formatting

Now that I had all my data, and I was satisfied that it was being ethically stored. I was able to move onto the second stage of my project – Data Formatting, making sure that the data I had collected could be used by my Machine-Learning model.

As discussed in the *Design* section, this fell into three subcategories. Which I dubbed Data “Cleaning”, this was parsing the raw JSON files returned by the API, and only keeping the few fields that would be useful, such as tweet text and ID, compared to the almost 100 fields that were actually returned for each tweet.

The second step was what I called Data “Formatting”, which is a very broad term, and indeed one that encompasses this entire section. However, in this case, I used it to define the actions taken to split the tweets into individual text files and arrange them in a two-level folder substructure. This was required as the Machine-Learning algorithm used this format, with the folder names being used as the supervised signal labels that the model would be trained with.

This also meant that the step of hand-labeling a certain amount of the data to be used to train the model had to be done at this step. Since the folder names were being used as the supervised signal labels (positive/negative/neutral), the files placed into these folders needed to be classified as well.

The third step was the Data “Pre-Processing”, this was finally taking the text data and performing a certain number of operations on it to prepare the data before it was fed to the Machine-Learning Model. These individual operations have already been named in subsection *3.2 Data Formatting* under section *3 Specification*. However, they will be discussed in more detail later in this section.

### Data “Cleaning”

Taking the files returned from the API and discarding the extra fields returned for each tweet object was a simple enough task.

This was handled with a quick Python script that can be found in *Appendix K*. The script very simply picked out the tweet ID field, and either the full tweet text field, or the tweet text field (depending on the length of the tweet). It took the data from these fields and simply printed it to a .txt file for clearer access for the rest of the project. An example of the resulting .txt file can also be found in *Appendix K*.

### Data Structure “Formatting”

After the tweets had been “cleaned” and placed into a .txt file for each state. They still needed to be split into individual text files, one per tweet. This was necessary as the Machine-Learning model required the data to be structured in this way.

Apart from being in individual text files, the data also needed to be organized in a two-level folder structure, using the folder names as supervised signal label names. An illustration of this can be found below, in *Figure 8*.

Text

Description automatically generatedFigure 8: Explanation of folder structure, from Scikit-Learns "load\_data()" function[39]

Due to the files not only needing to be split, but to be organized into the correct categories, I also took this as the opportunity to hand-label 100 tweets from each state (so 5000 total tweets) that I would then use as my training data.

This was once again done using a simple Python script. It would read the data from the .txt files created by the script described in the previous section and present the first 100, line by line in the default output to be classified, before creating and placing them in the right folder/file based on the user classification – given as a simple one letter input to the console.

The script used for this can be found in *Appendix L*, and the resulting generated “training sets” structured as described in *Figure 8* can be found in *Appendix M*.

The act of hand-classifying my data was a key part of the project – and one were I restarted part way through, as can be seen in the commit messages[40] on GitHub around March 26th. For the first 600 tweets I hand-labelled. I simply decided on positive/negative. However, this proved to be very difficult, as often, no obvious sentiment was present, and other times, a tweet would have a very negative sentiment, but still be pro-vaccines.

This was problematic as the point of the study for me wasn’t to classify simply the sentiments of Tweets, but to classify sentiment towards Vaccines specifically. A tweet that had a negative sentiment overall, but a positive outlook about vaccinations needed to be classified as positive – not negative.

As a result, I abandoned the Tweets I had classified, and started over (as seen in the commit messages). This time around, the classification methodology I adopted while labelling the training data was: “Does this tweet show that the person who wrote it is pro-vaccine, anti-vaccine, or can I not tell?”

Upon adopting this, I also changed my classification categories from just positive and negative, to positive, negative, and neutral. The process of preparing the training data took about 5 days – as I would label 1000 tweets a day, or 100 tweets from 10 states out of 50 total.

Moving on from preparing the training data. A slight modification of the same script was also used to split the tweet data not being used for training into individual text files as well. This script omitted the first 100 tweets in each .txt, as these tweets had been used for training, and as such would not be re-used for classification by the model later, this very slightly modified script can be found in *Appendix N*.

### Data Pre-Processing

The data pre-processing stage was an essential part of the project that involved preparing the individual text files and text data that had been cleaned and formatted. Pre-processing involved several steps, summarized briefly as:

1. Data “Pre-Processing”[10]
   1. Remove all special characters
   2. Remove all single characters
   3. Remove single characters from start
   4. Replace multiple spaces with single spaces
   5. Removing pre-fixed “b” from loading function
   6. Converting to lowercase
   7. Lemmatization
   8. Converting Text to Numbers
      1. Done using “Bad of Words” Model.
   9. Removing stopwords
   10. Finding TFIDF – (Term Frequency – Inverse Document Frequency)

As described earlier in the *Design* section, the pre-processing stage required the use of the Python “re” (regular expressions/RegEx) library, as well as the external Natural Language Toolkit (NLTK) library.

The steps involved in pre-processing occur in a number of files across the program, including when training the model (*see Appendix O*), evaluating the model (*see Appendix P*), and using the model to classify new data (*see Appendix Q*). This was because in all these cases, the new data that was loaded in needed to be pre-processed before being handed to the Machine-Learning model.

The implementation for the “Pre-Processing” steps A, B, C, D, E and F was all done primarily using regular expressions and can be seen in *Figure 9*.

Text

Description automatically generatedFigure 9: Data Pre-Processing Steps

Step G was “lemmatizing” the data. In lemmatization, words are reduced into the words dictionary root form. For example, "elephants" would be converted into "elephant". This is done in order to avoid creating features that are semantically similar but syntactically different. For instance, we don't want two different features named "elephant" and "elephants", which are semantically similar, therefore we perform lemmatization[10].

For the purposes of implementation, lemmatization was achieved using the “WorldNetLemmatizer” provided by the Natural Language Toolkit library, the import as well as implementation for this can be seen in *Figure 9*.

Step H was to convert the text data into numbers. Since the machine learning model cannot understand raw text, only numbers. There are a number of different approaches to doing this, including the World Embedding Model[41] and the Bag of Words Model[42]. For the purposes of this project, I used the Bag of Words Model.

The implementation for the bag of words model was done using the CountVectorizer function from Scikit-Learn. This can be seen in *figure 10*.

Text

Description automatically generatedFigure 10: Bag of Words Implementation

There are some important parameters to be considered when using the CountVectorizer function. The first parameter is “max\_features”. When you convert text to numbers using bag of words. All the unique words in all the input text are converted into “features”. Thousands of input tweets can contain tens of thousands of unique words. However, words which don’t occur very often are usually not a good parameter for classifying. So we use max\_features to use only the 1500 most occurring words as features for our classifier.

Other parameters include min\_df, and max\_df. This corresponds to the minimum and maximum number of tweets a word can occur in. So, we only include words that occur in at least 5 tweets, and words that occur in less than 70% of tweets. Words that occur in almost every single tweet are not useful for classification as they tend not to provide any unique information about the tweet.

Lastly, the CountVectorizer is also passed the a stop\_words parameters defined as stopwords.words(‘english’). Stop words are words which are filtered out before the processing of natural language. There is however, no universal list of stop words, nor any agreed upon rules for identifying stopwords[43]. For the purposes of this project, I used the “English” stopwords list provided by the Natural Language Toolkit, this completes step I, leaving just step J.

The one drawback of the Bag Of Words approach is that in assigns scores to words based on occurrence in one particular tweet. It doesn’t take into account that the word in question may also occur frequently in other tweets as well. Step J, or “TFIDF” solved this issue by multiplying the term frequency of a word by the inverse document frequency. While this sounds complicated, it is handled by the TfidTransformer function from Scikit-Learn. The implementation of which can be seen in *figure 11*.

Text

Description automatically generatedFigure 11: TFIDF Implementation

## Data Classifying

After all the previous steps, the actual act of training the model, and then using the model to classify data was actually very simple.

Text

Description automatically generatedFigure 12: Training and saving the Machine-Learning Model

As seen in *figure 12*, the “classifier” was defined as using the Random Forest Algorithm, as discussed in the *Design* section, and set to have n\_estimators of 1000, this just means the algorithm was set to have 1000 “Trees” in the forest.

Following this, a simple .fit() function was used to train the Model. In this case, X\_train holds the actual data from the tweets (however, this is not held as text, since it has been pre-processed, as discussed previously) and y\_train holds the supervised signal label names (positive, negative and neutral). So that the Model knows what to associate each tweet with during training.

Once the model has been trained, we use the base Python “Pickle” library to save the model to a file, allowing us to load it and use it for classifying without having to re-train it each it.

Loading the model and using it for training can be seen in *Figure 13*.

Text

Description automatically generatedFigure 13: Loading the model, and using it to predict.

The model saves its prediction in the NumPy array y\_pred. During implementation, this caused some headaches at the y\_pred array was seemingly randomized, making it hard to track that classification back to an actual tweet (as the array itself simply held a 0, 1, or 2 for negative, neutral and positive respectfully). However, this turned out to not be an issue with the prediction, but with the load\_files() function discussed earlier than was used to load the data (held in variable X).

The load\_files function by default shuffles the data as it is loaded, which was something I did not realize and took far to long to figure out. This was eventually solved simply by adding a “shuffle=false” parameter (as can be seen in the code, available at *Appendix Q)*.

The ”for” loop that can be seen at the bottom of the Classifier.py script (*Appendix Q*) was used three-level folder substructure for storing the classified data as positive, negative, or neutral for each individual state. This was done to have the data be in an easily readable function for the final part of the implementation. The front-end React.js App used to display the results of classification.

## Data Presenting

HERE

You should describe the important aspects of implementation, testing, and debugging that you went through to produce your system. You can structure this in different ways, depending on the development methodology adopted and the needs of your project. You may wish to start with a review and overview of the main features to be implemented and a general, architectural overview of the system. You may then wish to walk through the major features, components, or sub-systems that were created, one after another. These could be sub-sections in your report, e.g., Feature X, Feature Y, etc. Or you may wish to present a time-based review of the implementation process, according to the stages you went through in your project plan. Indeed, if you have adopted an Agile approach, you may wish to structure your discussion around the various Sprints that were undertaken. In your discussion, highlight any important features that were implemented, any major problems that were encountered, and the workarounds that you produced. Your aim is to convince the reader that you are technically competent and that you are capable of problem solving and adapting to needs of the project. The amount / extent of technical contribution is also being assessed and the extent to which you have been able offer original ideas of your own. Regarding the amount of technical contribution. For example, a basic website, with a few, static pages is likely to be rated somewhat poorly. Instead, one would expect dynamic content, a database, more complex code and problems being solved, additional considerations for accessibility, usability, security, etc.

Regarding the implementation section. You may wish to illustrate your discussion with diagrams, or code snippets, that offer additional insights into your work or

achievements. You may wish to emphasize user-centred processes, where applicable, and how the system evolved during implementation. For technically oriented projects, it is understood that you may wish to focus more on the performance, accuracy, reliability, or precision in your outcomes, including benchmarking against the work of others. For an additional layer of sophistication, any project can consider additional non-functional aspects of the system which are applicable, e.g., security, scalability, performance, usability, accessibility.

Later in your report, there is a related section: Description of Final Product. This later section is focused around providing a summary overview of your finished product. In contrast, the implementation section focuses on the stages that you went through to achieve and deliver it. There may be some areas of overlap, e.g., when you discuss the implementation of a particular user interface component, and you wish to use a screenshot to highlight the implementation choices made. Meanwhile, it turns out that a similar screenshot is necessary later in the Description of Final Product section, where you are simply presenting what the key aspect of the interface looks like. That is OK. There is just a difference of emphasis here.

For additional sophistication in your implementation, you should consider the use of software testing techniques, e.g., unit testing or similar. If so, the markers would need to see evidence of their use, e.g., in your source code or similar. In addition, you could consider traceability back to your original requirements, and verification or validation that they have been achieved.

As noted above, you may wish to include snippets of code in your report, to accompany your discussion of the implementation. Commonly, these may be included as screenshots of the relevant portions of code. It is best to keep these focused on specific areas of the code, e.g., it may be a specific method or a section of a method. For example, we are developing a web-based system which has a sequence of code for iterating through groups of product items. There is perhaps some reason why this code is noteworthy, e.g., it illustrates a novel approach or solves a tricky problem or is just something you are pleased with. Having discussed the feature, we wish to show a code snippet too. An example of this is below, e.g., please see Figure 2. Code Snippet. Iterating product options. below which illustrates the routine that was implemented to address this challenge. In the code snippet, you will see how the product items are iterated to complete the relevant basket page for the user.

# Evaluation / Testing

You must evaluate your system. This will be done in different ways depending on the project. For example, if you are developing a web application or app, it is common to do user testing and you may wish to seek feedback and comments from end users through interviews or questionnaires. If your project has a technical, non-user-based focus, your testing may focus more on benchmarking, comparing different algorithms or parameters, measuring performance or precision, etc. For any type of project, you can consider additional criteria where applicable, e.g., security, performance, accessibility, and computational efficiency. In the case of Cloud-based applications or services, one could also consider the cost implications (e.g., 'x' pence per query) and whether this has influenced the design and testing of the application.

Regardless of the project, you must describe the evaluation or testing of your system in your report, and this must include the following: a presentation of any relevant data; a discussion and analysis of the data; a discussion of the significant results and outcomes you have found. Ideally, you should consider any limitations in your evaluation and the extent to which your outcomes can be generalized to a wider ‘population’, or not.

Consider what you want to evaluate or test, and how you will achieve it. Develop the necessary evaluation plans / materials / methods, and make sure these are described in your report. Be mindful of ETHICS where required and make sure that the relevant Ethics documents are utilised, and it is clear where and how ethics has been adopted in your evaluation. Describe how your tests or evaluations were conducted. You can include the materials you have used in your appendices, e.g., test plans, evaluation checklists or tasks, copies of questionnaires used. Present and discuss the data in your report. You can include copies of the data in the Appendices too. Discuss the main outcome or findings from your evaluation / testing.

# Description of the final product

You should provide a clear description of what the final product looks like and what it does. You do not have to explore every minute detail of the system, you should attempt to convey the key, major areas of functionality. In some ways, you could consider this section to be a cut-down version of a user manual. Even in systems where there is no user interface, there may still be some general aspects that you can mention. However, if it is the case that this section of the report is just not relevant to your project, please just state that or omit this section.

When you are writing your report, you may find that the content of this section could overlap with earlier content in the report too, such as the implementation section. We want to avoid repetition in the report. At the same time, a degree of overlap is OK, bearing in mind that it is other people who are reading your report and they may benefit from a reminder, and a focused overview of what the final product looks like. As noted earlier, this section provides an overview of your finished product whereas earlier sections such as the implementation focus more on how you got to that point, i.e., the stages you went through, the decisions you made, and the problems you had to solve along the way.

# Appraisal

Provide a critical appraisal of the project. The question that I would pose to you here is as follows: if you were doing the whole project again, what would you do differently, what would you do the same, what advice would you give to others if they were doing the same project? Here you should reflect on the entirety of your project including your choice of technologies, your implementation decisions, and the project plan. With the benefit of hindsight, what are the lessons learned during the project and the evaluation of the final product and the process of its production (including a review of the plan and any deviations from it). Also consider what have been the most useful learning aspects for you.

NOTE: the appraisal section could potentially occur after the Summary and Conclusions below, or even as a sub-section within the Summary and Conclusions. See what works best for you and your advisor.

# Summary and Conclusions

Summarise the main points of what your project was and what the report has provided. Provide a summary. Describe the conclusions and outcomes that you have found.

# Future Work

What recommendations do you have for future work? Are there more features that need to be included? More testing? More evaluations? Are there follow-on projects or ideas that could be explored? Do you plan to do any more with the project yourself? Please discuss this here.

NOTE: this section could possibly appear as a sub-section within the preceding Summary and Conclusions.

## Acknowledgments

You can provide acknowledgements here to anyone who has been helpful in your project, or beyond. In some cases, the licensing of certain software products you have used may require you to acknowledge them here, e.g., in return for free use.

# References

[1] Durbach, N. They might as well brand us: Working class resistance to compulsory vaccination in Victorian England. The Society for the Social History of Medicine. 2000;13:45-62.

[2] Article on Unvaccinated Patients: https://northeastlondonccg.nhs.uk/news/almost-90-of-patients-admitted-to-intensive-care-units-innorth-east-london-are-not-fully-vaccinated/

[3] Cambridge Analytica Swinging Elections: https://www.theguardian.com/news/2018/may/06/cambridge-analytica-how-turn-clicks-into-voteschristopher-wylie

[4] Piedrahita-Valdés H, Piedrahita-Castillo D, Bermejo-Higuera J, et al. Vaccine Hesitancy on Social

Media: Sentiment Analysis from June 2011 to April 2019. Vaccines (Basel). 2021;9(1):28. Published 2021

Jan 7. doi:10.3390/vaccines9010028

[5] Almatarneh, Sattam, and Pablo Gamallo. "A lexicon based method to search for extreme opinions." PloS one 13.5 (2018): e0197816.

[6] VADER: https://github.com/cjhutto/vaderSentiment

[7] Neha Puri , Eric A. Coomes , Hourmazd Haghbayan & Keith Gunaratne (2020) Social media and

vaccine hesitancy: new updates for the era of COVID-19 and globalized infectious diseases, Human

Vaccines & Immunotherapeutics, 16:11, 2586-2593, DOI: 10.1080/21645515.2020.1780846

[8] Muric G, Wu Y, Ferrara E. COVID-19 Vaccine Hesitancy on Social Media: Building a Public Twitter Data

Set of Antivaccine Content, Vaccine Misinformation, and Conspiracies. JMIR Public Health Surveill. 2021

Nov 17;7(11):e30642. doi: 10.2196/30642. PMID: 34653016; PMCID: PMC8694238

[9] Twitter API: Twitter API Documentation | Docs | Twitter Developer Platform

[10] Stack Abuse: Text Classification with Python and Scikit-Learn: https://stackabuse.com/text-classification-with-python-and-scikit-learn/

[11] React.js, A JavaScript Library for building user interfaces: https://reactjs.org/

[12] Chart.js, Simple yet flexible JavaScript charting for designers and developers: https://www.chartjs.org/

[13] SourceForge, Compare, Download & Develop Open Source & Business Software: https://sourceforge.net/

[14] Bitbucket, The Git solution for professional teams: https://bitbucket.org

[15] Git Lab, iterate faster, innovate together: https://gitlab.com/

[16] GitHub, Where the world builds software: (https://github.com/)

[17] Postman API Platform: https://www.postman.com/

[18] Postman API Twitter 2.0 Endpoints Collection: https://github.com/twitterdev/postman-twitter-api

[19] Microsoft Windows Subsystem for Linux (WSL): https://docs.microsoft.com/en-us/windows/wsl/about

[20] cURL, a command line library for transferring data: https://curl.se/

[21] Scikit-Learn: Machine Learning in Python: https://scikit-learn.org/

[22] Python Programming Language: https://www.python.org/

[23] NumPy, https://numpy.org/

[24] re, https://docs.python.org/3/library/re.html

[25] Natural Language Toolkit (NLTK), https://www.nltk.org/

[26] pickle, https://docs.python.org/3/library/pickle.html

[27] TensorFlow, an end-to-end open-source platform for machine learning: https://www.tensorflow.org/

[28] Angular.js: https://angular.io/

[29] Vue.js: https://vuejs.org/

[30] Frameworks Comparison Video, Fireship: https://www.youtube.com/watch?v=cuHDQhDhvPE

[31] Most Popular Frameworks, according to Statista: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

[32] Bootstrap, https://getbootstrap.com/

[33] React-Router: https://reactrouter.com/

[34] React-Twitter-Embed:

https://www.npmjs.com/package/react-twitter-embed

[35] React-GitHub-Pages:

https://github.com/gitname/react-gh-pages

[36] React-Select: https://react-select.com

[37] Covid Tweets Dataset from Kaggle:

https://www.kaggle.com/datasets/kaushiksuresh147/covidvaccine-tweets

[38] Kaggle: https://www.kaggle.com/

[39] sklearn.datasets.load\_files(): https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\_files.html

[40] Link to Github Commit History set to March 26th:

https://github.com/GriffinKF/Y4-HonorsProject/commits/main?before=ca99b369b74722275ff1898c965f2819f2c29282+70&branch=main

[41] Word Embedding Model:

https://en.wikipedia.org/wiki/Word\_embedding

[42] Bag of Words Model:

https://en.wikipedia.org/wiki/Bag-of-words\_model

[43] Stopwords: https://en.wikipedia.org/wiki/Stop\_word

# Appendices

1. Hand-Written meeting log for first meeting with John Lawrence.
2. Gantt Chart for Semester 2
3. Example of JSON Returned by API
4. Email Containing Resources for Scikit-Learn provided by supervisor (John Lawrence)
5. Twitter Acknowledges Application Appeal
6. Twitter Returns API access
7. 7-Day Endpoint Result for Each State
8. Tweet Dataset Obtained from Kaggle
9. Premium Search Invoice
10. Link to Full Project Repository
11. Tweet Cleaning Script and Result
12. Manual Classifier Script
13. Training Sets
14. Tweet Splitter
15. Model Training
16. Model Evaluating
17. Classifying New Data