CM2305 - Group Project (Group 7) - “Misinformation Dashboard” Final Report

Méabh Carragher (2042336), Samuel Carrington (21067509), Harry Franklin (21056720), Charlie Howard (21058496), Jake Samuels (21068235), William Weston (21113347), Hayley Yau (21050400)

Client: Alun Preece
Supervisor: Miah George-Thompson

Introduction

Our client requested that we design and implement a system that could be used to monitor the propagation of misinformation over social media. Following our initial ideas and requirements presentation, we set out to create a dashboard which would allow a user to view trends in misinformation and conspiracy theories on the social media site Twitter. Over the past 5 months, we have made significant progress and have successfully completed this task. Despite several setbacks throughout the project, we have managed to realise the specifications of the client into a functional system that retrieves relevant data from Twitter, analyses it for trends and displays it on a web dashboard in several novel ways. This report will discuss our progress from the beginning of the spring semester to the final presentation with our client and the competition of our project.

Our initial plan was to use the Twitter API to gather data about tweets which fit into several predetermined categories of misinformation. We would then analyse these tweets and store the resulting data in a database, to be displayed on the dashboard for the user. Ideally, this could be done in real time and the data we gathered would be displayed in a variety of different ways. Following our meetings with the client, we agreed on trying to organise the data in three ways:

1. Firstly, line graphs comparing the popularity of different categories over a given period of time
2. Secondly, world map where countries were highlighted to indicate the proportion of Tweets of a specific category that originated from that region
3. Lastly, word-clouds that showed what words were popular in conspiracy communities at different times.

As a result, our first tasks were to determine what categories we could use to group misinformation and to determine how we would assess the popularity of each category.

Categorisation of Misinformation

We decided that our categories should focus on grouping tweets based on a specific concept or event within misinformation and conspiracy communities, as this would allow us to observe which specific ideas were the most popular at any time, as well as allowing us to observe correlations between the rise and fall of different categories. The plan was to search for relevant tweets and organise them based on keywords that related to specific ideas within conspiracy theory lore. We also ensured that one tweet can occupy several categories at once, as it is possible that someone could discuss multiple topics in a single tweet. We eventually decided upon 16 categories, which are as listed in the appendix.

Methods of Gathering Misinformation

Throughout the development process, we were forced to use several different methods to gather tweets from Twitter. Initially, we intended to use Twitter's API to retrieve relevant tweets, however, on the 2nd of February it was announced that Twitter's API would be moving to a paid-only model. As no details of how the new pricing model would work were released at the time, we chose to try and find an alternative option, as it is better to be safe than sorry. Also, we concluded that this was an especially volatile period for Twitter, and relying on it so heavily would only cause further setbacks. Our shift to an alternative option of tweet gathering had the benefit of potentially removing the restraints of the Twitter API, which was limiting the number of tweets we could potentially acquire per month, amongst other factors.

Following some research into alternative options, we eventually decided to use a web scraper instead. This allowed us the use of Twitter's search function to request tweets that matched certain keywords, which we could then organise and store in our database. This option had no limit on the amount of tweets we could retrieve each month, but it did come with some downsides. Most notably, without the Twitter API we no longer had the ability to retrieve most of the location data on Twitter, as only authorised users have access to. The only location data we could access was what users had added to their public profile. This data was unreliable and impossible to verify. Therefore, we had to drop the location-based partition of our data visualisation plans. In addition, the Twitter search function has a limit on the number of characters used in a single search, meaning that our keyword-based searches needed to be kept to below 500 characters each. This was initially a problem but we solved it by using a set of general keywords to find a large set of potentially relevant tweets, then sorting them into categories internally. Another unintentional benefit of this method was that it allowed us to retrieve tweets that were considerably older than what was possible with the API, as the API only provides the most recent 30 days of tweets, whereas this option provided tweets as far back as 2018.

However, Twitter made further (this time unannounced) changes which made this method unviable as well. Twitter suspended the use of its search function to unauthorised users on the 21st of April. This presented a significant problem, as our final presentation to our client was on the 3rd of May, and we had been left with no way of gathering new Tweets.

During this time period, we found a new solution which relied on the nature of Twitter's data organisation surrounding a user and their tweets. While we could no longer request a search using a web scraper, we could request the data of a single user. This contains their most recent 3,200 tweets. Each of these tweets contains the data about users who have interacted with it, specifically anyone who has commented, quote tweeted, or was mentioned in the original tweet, more on this later. The process can then be repeated for these new users, allowing us to gather tweets. The advantage of this method is that it allows us to trawl through Twitter in a more logical, web-like manner, as this process can be started with a user we already know to be a source of misinformation, and therefore is likely to interact with other users with similar views. This results in an overall increase in the ratio between the total number of tweets checked and the number of relevant tweets we obtain, increasing our tweet gathering facility's efficiency, satisfying requirements set out by ourselves early in the project - as we aimed to pull a certain percentage of relevant tweets. Additionally, we could now retrieve tweets as far back as Twitter's release, assuming that the user had only tweeted less than 3,200 times before that point. This additional timeframe allowed us to observe many more trends than would have been possible with only Twitter's API, or the first version of the web scraper. This provides us with a more complete view of how misinformation has propagated across Twitter over the past 17 years. However, due to the limited number of tweets per user, which prioritised newer tweets, it would be increasingly hard to find tweets based on how old they were. For most accounts, this is a lesser problem, though for particularly 'spammy' accounts, it is more likely that our new system would miss older tweets.

Due to the limited time we had to develop, implement and test this method, we were only able to have it run for around 8 hours on a single machine before our presentation. Despite this, we managed to retrieve 628,598 tweets.

Overall, despite the difficulties we encountered, the process of gathering tweets was a huge success. We also gathered enough tweets to properly demonstrate the purpose and benefits of our product, through the use of graphs, such that they had meaning to them. However, we could have improved our implementation by allowing for tweets to be gathered in real-time.

Live Data Gathering

One of the original suggestions of the client was that data could be gathered in real time. While this is possible using our current system, the disruption caused by Twitter's policy changes meant that we never had time to implement it properly for any of our data gathering methods. It would be possible to check the status of each tweet in our database on a regular basis, in order to update engagement scores or to remove the tweet if it had since been deleted, though this would reduce the rate at which we could gather tweets and ultimately it was decided the project would be best served without such a feature, as our focus could lie elsewhere.

Assessing the Popularity of Each Category

Using research completed early in the semester, we formed our own formula for creating a score for each Tweet. Our solution is more simple than the ones presented in the researched papers[1,2], though we feel ours is more than adequate to convey enough meaning on our visualisation tools:

$$Score = (1 * views) + (2 * likes) + (3 * retweets) + (5 * quoteTweets) + (5 * comments)$$

We decided that comments, quote tweets and retweets were the most valuable metrics, because they create more content that can spread, meaning more people could be influenced by them, and so on. Therefore, quote tweets and retweets are weighted the highest.

Web Team

As we were using an agile methodology, it was important to the website team to ensure that we always had a working prototype. The team used GitLab, a git-based source code repository, which enabled the team to focus on this working prototype idea and aim to work on the project at high velocity and with high efficiency. GitLab allowed us to utilise a version control system. This meant that each member in the team could work concurrently on the project without having to worry about the problems that can occur, as Git stores each version of the code and deals with clashes. As a result of each member in the team having their own working copy which they can work on, this caused the team to collaborate much better towards the project, and avoided starvation issues. GitLab is also self-hosted meaning that we did not need to worry about hosting of DevOps service. Using DevOps meant that we could build, test, and release prototypes of our product in a continuous way, and streamline the process.

As a group we concluded that our colour scheme should match the Twitter blue to create a sense of familiarity and consistency. To make the website user friendly, we decided on a navigation bar to move between the web pages. The decision to include dynamic buttons that have the ability to hover was so that it would entice the users and make the site more inviting and user friendly. We also incorporated the words “Hello, Guest!” where “Guest” will change to the user’s name once they log in. This again, invites the user to the website and gives them a personalised feel.

From the client requirements, the aim of our website was to present visualisations in order to help the user understand misinformation trends, and this was the main problem to tackle from the website team's perspective. As a result, we subdivided these requirements into 3 smaller sub-requirements to decompose the problem into smaller and more manageable tasks.

---

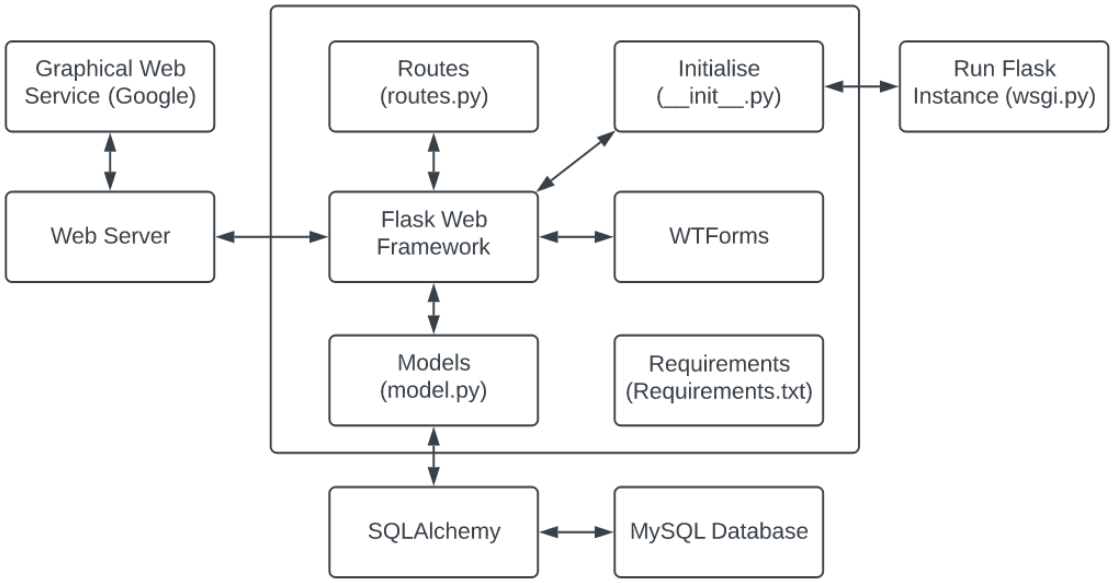
[1] Silver N, Kierstead E, Kostygina G, Tran H, Briggs J, Emery S, Schillo B. The Influence of Provaping "Gatewatchers" on the Dissemination of COVID-19 Misinformation on Twitter: Analysis of Twitter Discourse Regarding Nicotine and the COVID-19 Pandemic.

[2] Fang, Z., Costas, R. & Wouters, P. User engagement with scholarly tweets of scientific papers: a large-scale and cross-disciplinary analysis.

These tasks could then be carried out by respective members of the team, in conjunction with our Kanban style scheduling system to enable ourselves to work within our agile methodology. These 3 subproblems are: A Website Framework, Database Integration, and Visualisation Tools.

Website Framework using Flask

The Website and its framework would act as the platform for users to access the platform, log in or register an account, select the type of visualisation tools that they would like to use, and select how the data should be filtered. We chose to use flask as our framework for a number of reasons. Namely, Flask is a microframework that each member of the team has had previous experience in from a module we completed in first year. The team's prior experience allowed for a quicker process of creating code which was more robust, as we were already familiar with the system. The team created the flask website using this Website Deployment Diagram, which outlined how we expected the website and its components to integrate together to get the final website operational. This diagram involves a Central Flask App (Flask Web Framework) with various supporting systems (Routes, Form Handling, Database Modeling, Initialisation Files, etc.).



Flask websites use a system known as routing, in which specific functions are mapped to each URL and are responsible for any logic that is happening under the hood of that URL. These routing functions were responsible for most of the functionality of the website, and through a render_template variables are able to be passed directly into a HTML page such as data arrays and forms. HTML templates are rendered by the routing function at the particular URL using the render template. These routing functions are very flexible and can be assigned to multiple different URLs in the app through the use of python decorators, as well as generate dynamic routes.

One future aspiration of this product was to expand on the user system, in which dynamic routing could've been implemented to generate URLs corresponding to each user. This would enable users to have their own respective and private page on the website. Some ideas were to enable people to have their own misinformation categories tracked to then be displayed on these custom pages. Ultimately, this is an example of a feature dropped towards the end of the project, so we could adequately finish the more important aspects of the project, as laid out by the Trello board/requirements set out by the client.

The forms were implemented through a system known as WTForms, and were used in particular due to their built in form rendering system as well as data validation. This library was trustworthy and overall very useful and solid in getting this aspect of the web development working. In order to insert these variables passed by the functions into our render templates, we need to use a Web Templating Engine, which enables quick rendering of data sent to it by the server side/backend. The variable referenced in the render template is mentioned in the template file (home.html for example), where it is to be replaced by the actual data. Utilising this concept, we were able to make designs for our website using both HTML and CSS for which the data is to be placed into. This process of forms enables for a quick and rapid development process, as forms are able to be cast and passed from routing functions easily into HTML templates, which follow a very object oriented approach.

Using a templating engine like this was especially helpful for our first time working on a large scale product like this and carried many benefits. One major benefit is the use of template inheritance, in which particular templates "extend" a base template. We used this in our website in the Navigation Bar system, in which every single page of the website extended this. Using Jinja2 also lets us iterate over an item and perform simple condition checks such as IF statements, so that for example if a user is logged in, to display that on the Navigation Bar.

Database Integration using Pandas

Creating a Pandas Dataframe on initialisation of the website from the SQL table was really useful for this project, as it enabled us to manipulate the big data of the database much more powerfully. Pandas Dataframes utilise columns with specific names, such as the 'categories' which were listed earlier in this report. These give us the opportunity to filter the data, by matching values in the categories table to the category number the user selected on the interface. Not only did utilising Pandas allow more powerful methods of searching, but it made the solution more simple and readable compared to other methods. In the website team we had briefly discussed developing our own solutions to the program with build in SQL queries through our engine, however due to the nature of the project and its unforeseen circumstance we deemed that creating a system like this could be more complicated and cause problems that would take more time to solve. Using a Pandas Library that had already been tested and is reputable, was a more robust and simpler solution. It seemed to fit our problem especially well as there were built-in functions that were able to read SQL-Tables through our own engine. Pandas was also a frequently used library in first year, as well as an optional module this year, which saved us time as there was less of a learning curve to start using it effectively.

Visualisation Tools

When discussing with our client in our requirements presentation, it was clear that the client wanted three types of visualisation graphs: A Temporal Visualiser, A Semantic Visualiser, and a Spatial Visualiser. We aimed to include all three types of visualisation tools in the final project, however, as discussed above, it became apparent over time that using a Spatial Visualiser would not be feasible. This realisation was disappointing, as a GeoChart would have been a particularly exciting tool. The user would be able to see how these topics were spreading globally, and where each topic was most popular. However, due to the changes in collecting data,we were left with only two visualisation graphs, temporal and semantic.

The Temporal Visualisation Tool was responsible for comparing either one or two misinformation categories over a time period to see its temporal trend. On our home page, this was implemented as a quick recap, showcasing the activity from the chosen misinformation category over the previous 30 days. The compare page was responsible for using two categories over a set of user-inputted dates, using two date pickers. When meeting with the client at our interim meeting, we had not yet integrated the website with the database. We were able to implement test data to use these graphs to provide the client with a visual representation of our vision for the website. By supplying these pre-built charts with dummy data we fabricated, we were able to create a prototype of the website, further showing our strong connection to the agile development methodology, as we prioritised working prototypes over fully fleshed out code.

In order to display the categories trend over time, we had to create a system in which our database data could be filtered by a given date range as well as a category. When a user submits a form, the category selected is able to be gathered by the routing function in which the data is able to be filtered by. From here we are able to also gather the dates selected by the user, and create a range of datetime objects between the two given dates.

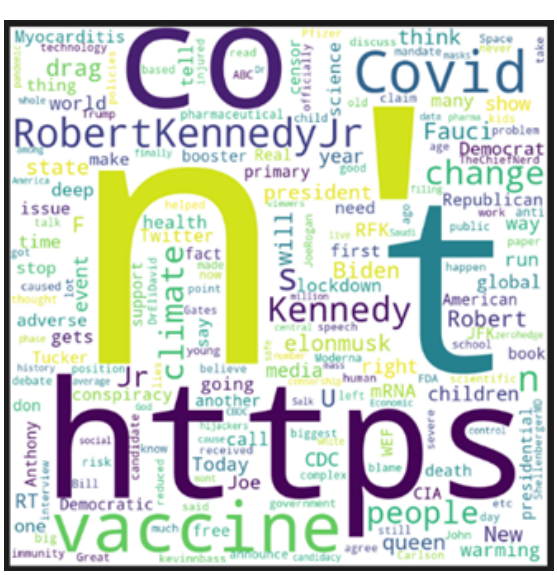
JavaScript (along with JQuery) enabled our website to respond dynamically to our user. To create visualisations using the data we have selected through filtering, we would need to render them either Server Side or Client Side. The approach we went for was to use Google Charts, a Web Service that creates client-side charts through the use of vector based graphics. By using a client-side approach, it means that there is less strain and load on the website. This was of particular consideration to the team as in our requirements we had the objective of having multiple users running the website smoothly simultaneously. Due to some of the data varying by order of magnitude, such as ‘9/11’ peaking massively each September, we decided to use a logarithmic scale to plot our data onto. In this way we could follow both macro and micro changes to date over a course of time, and be able to represent these large ranges of values in a much more compact and human-readable way.

One issue that we had with the Graphs being generated client-side however is that they did not adjust dynamically to the website. In order to combat this we used the Document Object Model Manipulator (JQuery) in order to create an event for each time the website had been resized. Then, each time the website was resized and this event happened, we were able to recall the JavaScript function that was responsible for drawing the chart, but this time to the right spacial requirements needed. As per the Kanban board's tasks, one task that was of particular importance was the appearance of the website/aesthetics. So therefore it was important to us within the website team to make careful considerations that affected overall aesthetics such as this.

As we were using Google Charts for the other forms of visualisation, we researched using the Word Cloud tool from that library, but in terms of using forms and pandas to collect the necessary data from the database deemed the task too difficult. As mentioned below, we went with the option of generating the word clouds, then storing them as Portable Network Graphics images. These were passed on to the web team in a zip file, where each file was a word cloud representing a month. For the other visualisation tools, forms were used to allow the user to select what data they would like to see. We had initially hoped to use a selecting tool for the user to input the month and year of word cloud they would like to see. However, as the images of the word clouds were not stored in a database, it would be hard to implement the forms tool to allow for selection in this way. We settled on a compromise of allowing the user to select which word cloud they would like to see in a simple drop-down box listing which word clouds were available.

Word-Clouds

One of our visualisation tools is the word-cloud. This took a couple weeks and a few cycles of iteration - inline with our Agile development methodology. The first version suffered from exceptionally long processing times, which would have not been suitable at all. The second version was much quicker but didn't display much useful information. After removing stopwords, our final version was hundreds of times faster than v1 and displayed much more useful content. Libraries were used to remove the stopwords, along with manually combing through the word-clouds and removing common terms which have no meaning. Twitter stores letters representing different objects, for example, "t.co" represents a link shortener. This symbol, along with "RT" and "n " needed to be removed for the word clouds to be more relevant and easier to read. The before and after of the word-clouds can be seen below:



As you can see, the 'before' image conveys very little meaning, as large portions of it are taken up by non-useful terms. In contrast, the 'after' image conveys much more meaning, and you can easily tell what was being talked about in a misinformative manner, and even what month's data it uses. As a group, we are really happy with the final outcome for the word-clouds, and the group who worked on it learned a lot from it.

Website Security

One library that was useful for implementing effective website security was Werkzeug, a Web Server Gateway Interface. Werkzeug has a security feature responsible for the hashing of our user passwords, so that user passwords are never actually dealt with in the MySQL Database. This is an important feature due to the fact that our users' sensitive credentials would be safe against being compromised. The way we are able to check if a user entered a correct password is to compare the hashes against one another, instead of comparing the raw password information. This system is integrated into the Flask User Model, which uses flask_login's UserMixin to provide default implementation for the methods expected of user objects.

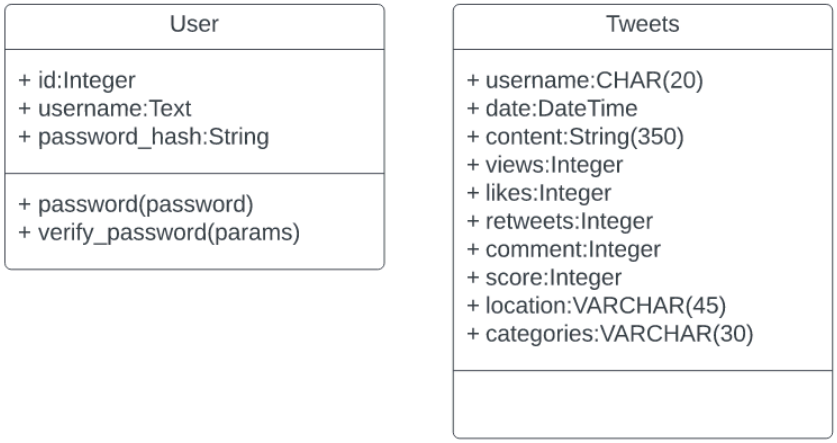
Database Team

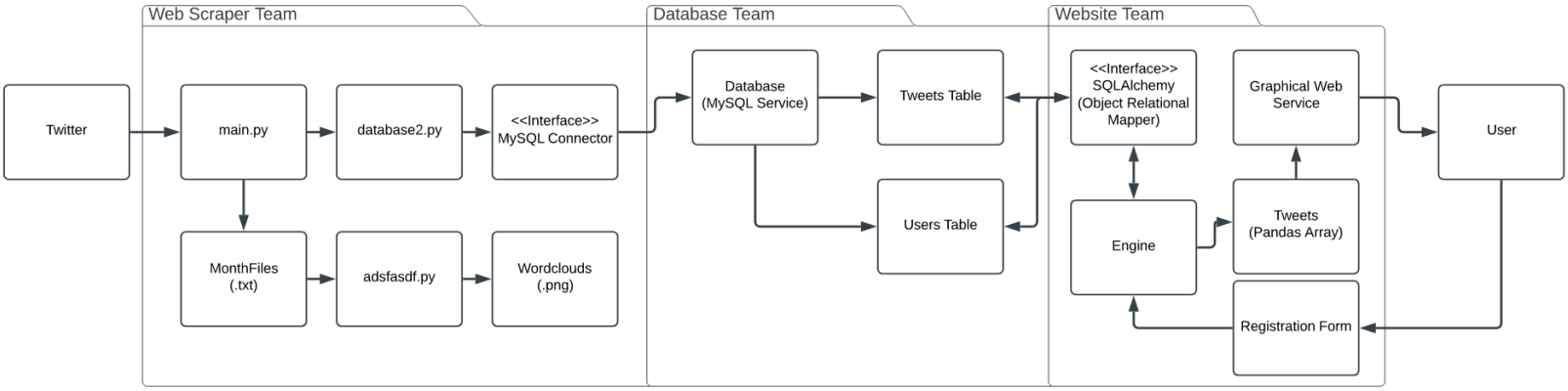
Our main role was to design an efficient way to store the data collected by the API/scraper team and have it readily available for the web team to retrieve and display. Our team consisted of two members, but due to lack of contribution from one member, the responsibility of designing the database fell solely on one member. Nonetheless, our team worked together to distribute the workload effectively.

To begin, we collaborated with the API/scraper team to determine what data was being pulled and what data we could use. After reviewing the data, we decided to create two tables for the database: "Tweets" and "date". The "Tweets" table would hold information on each tweet, and the "date" table would hold information on the scores of each category on a given day. Certain fields, such as location, were set to nullable because the location information was only available for a small number of tweets. All other fields were set to not null to ensure that every tweet had a value.

Once we had determined how to store the selected data, we created a Python script that would connect to the database, insert tweet information into the "Tweets" table, and retrieve the date, score, and category information for each tweet. At first, the date table was named "date", but this caused syntax errors in python because the name conflicted with a set variable in the datetime module. To avoid this, we renamed the table to "dateTable" and the column to "tweet_date". During the integration phase, the API team discovered that the date table was redundant, so we used the tweet table to calculate the date-by-date score, which was much more efficient. We provided the API team with the insertion script to integrate it with the scraper to store the data in the database.

To integrate the data into the website we worked closely with the website team, using SQLAlchemy and MySQL libraries to enable the website to pull data from the database. This is done through the use of an engine, which is able to facilitate the communication between the python script in the website files, and the MySQL Server Service that runs locally on a dedicated port. In order to do this however, the engine needs to be able to map the Database to a Object Relational Mapper. This was done by declaring a model with Flask-SQLAlchemy using a class with a parameter of db.Model. Understanding the steps and how to configure these was not easy and required both the database and website team to spend lots of time reading and understanding module documentation. However once we were able to connect to the database, we converted the website team's CSV insertion script to the SQL server. As well as this, we could redirect the URI of the user's database that was being hosted to the same database under a different table, integrating the users.





Our Development Methodology

Before Christmas, we decided to follow an Agile development methodology, which served us well. Every member of this team was important, so the productivity of our group could be hugely impacted due to a small change, such as an illness, or people dropping out. Therefore, an agile methodology allowed our productivity to flourish through interactions such as pair programming, working software over extensive documentation, and responding calmly to changes in our original plan.

Pair programming allowed teams to work collaboratively with other teams, allowing us to efficiently optimise code and troubleshoot problems. Timekeeping was another important skill we utilised to ensure that we met project deadlines and did not create bottlenecks. We prioritised getting code to other teams when they needed it to prevent delays in the project.

As seen in the report submitted in January, we had developed a Gantt chart as a timeplan. This was integrated with our Trello board, which helped us as a team to organise our tasks and manage our time effectively.
Our Trello board was separated into areas of importance, related to our requirements set out by Alun, and requirements set out by ourselves. We aimed to constantly have working software, so we could easily develop and integrate requirements without having to wait for a functional website, for example. The Trello board also served as a way for us to adopt the Kanban system, where we could see what was being worked on, what was done and what needed to be done. Kanban was a particularly useful tool as it complemented our Agile development methodology nicely. Instead of looking at the project as a whole, Kanban enabled us to decompose the project into smaller and more maintainable tasks. We then listed these subgoals to the Trello board, allowing each member of the group to see the current progress of each task in the system. The goal of this approach was to allow each member the opportunity to delegate themselves a task. This reduces the chance of two members working on the same task unnecessarily. This approach increased the transparency in the team, as progress by each member was displayed clearly. We learnt throughout this project that the priority of each task would change. This could be depending on productivity of team members, which tasks had been completed or changes to

the twitter application. Through using Kanban and the Trello board, however, we were able to keep up with the changing priority of tasks. This system also allowed for transparency between teams, which was particularly useful when it came to integrating the different components of the project, for example, the integration of the database with the website.

By following this methodology, we were able to reduce the impact of unexpected issues, such as the changes in data collection we had to make, and a group member dropping out.

As a team, we are proud of the level of communication we were able to achieve by practising traits from agile methodology. Frequent interactions between subteams and within each subteam allowed for better organisation, and therefore a more successful product.

Legal, Social, Ethical and Professional Issues

The majority of our ethical concerns surrounded the methods we used to gather tweets, and the potential risk of obtaining sensitive information. This also raised some legal concerns, as we could have potentially violated GDPR by acquiring or mishandling personal information. At first, we were concerned about our gathering of location data through the Twitter API, however this became irrelevant once we dropped that data visualisation method. After this point, we exclusively gathered information found in a user's profile or public tweets, which Twitter users must agree to allowing before they can create an account, as Twitter's privacy policy states:

> *"Twitter content, including your profile information (e.g., name/pseudonym, username, profile pictures), is available for viewing by the general public. The public does not need to be signed in to view much of the content on Twitter. They may also find Twitter content off of Twitter, for example from search query results on Internet search engines."*

As such, we operated on the assumption that any of the information we had access to, the user must have consented to posting. Additionally, if a user deleted a tweet or made it private, we would no longer have the ability to access it. In this way, our system was incapable of gathering any information that someone didn't want to be available to the public. Ethically, this also fell within the universities' ethics standards, as we had consent from anyone from whom we gathered data.

There were also some ethical and legal concerns surrounding our database. We ensured that our data storage was compliant with the relevant legislation by ensuring that the data we were keeping was accurate, secure and would not be kept for an unreasonable amount of time. Additionally, we only stored data that was relevant and necessary for our project to function. The only problem we had with this was that if a user deleted a tweet after we had added it to our database we would still have access to it. We solved this issue by wiping our database after each attempt at gathering tweets, to ensure that the next dataset we created would account for deleted tweets. In addition, storing now-deleted tweets poses very little risk to a user's privacy due to the way we display data on the dashboard.

Because the dashboard can only display data about tweets as a collective, and has no way of displaying the Tweets or profile information of a specific user, there is no way to use our systems to find a tweet that has since been deleted. We also accepted that having tweets that are now deleted in our dataset does not discredit the results we obtain using it, in the same way that many of the tweets in our dataset have had more engagement since we obtained them, it simply reflects that our data gathering process is not live. If we were using a live data gathering system, we would be able to completely remove this problem entirely.

Finally, we used several data preservation and security methods to keep our database secure. This included: RAID-5 data storage to prevent data loss in the event of a hard drive failure, Database password protection, and SQL injection prevention techniques in our program.

Risk Assessment

The risk assessment we completed (and showed in our interim report) ensured we knew how to handle errors and issues if/when they arrived. As discussed, we encountered issues throughout the project and overall handled them well. Mostly because we had plans in place to help us avoid such issues from crippling our project. For example, we saw the possibility of the API not working any more. Our contingency plan for such was to use the provided sample data. This contingency was in place and ready to be deployed if Jake's second and third versions of the web-scraper proved fruitless. Thankfully, we didn't have to use the sample data, and our product is of a much higher quality as a result of which, as we have tweets dating back much further, and dating further forward, than the provided data set would allow. The knowledge of having a 'back-up plan' after the Twitter API got shut down allowed us to focus our time on other project areas, as we knew we would still have tweets.
Fortunately, our data management process meant that once we had a functioning database, we had no problems with storing or preserving the data we gathered. In addition, we managed to minimise the amount of data we stored in order to minimise hard drive usage and therefore maximise the number of tweets we could store. We estimate that we could store the required data of over 10 billion tweets if we needed to in the future. This shows the project has potential for expansion and is more than just a 'proof-of-concept'.

Another issue we dealt with successfully, due to a contingency laid out in our risk assessment, was handling dwindling group numbers. Due to our organisation and communication with each other and our supervisor, this impacted us to a lesser degree than it would have done had we not thought about such a possibility earlier in the project's development. Lachlan left so early on in the project that it did not have a significant impact (aside from the reduced weekly productivity), and while Sam's contributions only started diminishing after christmas, he had a teammate (Charlie) who was able to finish the database system without him.

Our communications systems (primarily the Trello board) allowed us to effectively monitor our time between deadlines, so that we were always prepared for them ahead of time. Our decision to split the project into several teams also helped in this regard, as it meant that the other teams could continue to work even if a different group was experiencing problems. The only time when a deadline played a significant factor in our project was when we had a very limited amount of time to develop the third and final method of gathering tweets. This time constraint was out of our control, and if we had additional time we would have been able to gather more tweets before our presentation, but ultimately we still managed to deliver an effective demonstration of our project on schedule.

By using the Trello board to track and monitor what needed to be done, we had very few issues with scope creep. The clearly defined requirements of our project allowed us to easily see what we had originally deemed to be most important (especially as each task was organised based on its priority). This ensured that we did not waste time by trying to introduce features that were not key to our original vision.

We initially had some concerns surrounding the team's capabilities in terms of implementing some parts of the project. We were primarily worried that we would lack the skill set required to complete some of the more technical parts of the project, especially areas like the web design and database as we did not have any experience outside of other modules. However, we had enough time that we could learn how to use these systems as well as adapt them to the specific needs of our project.

We were also concerned about receiving approval from the ethics board, however this was not a problem as we first confirmed that our actions fell within the universities' ethics guidelines and the relevant legal guidelines before applying for permission with the ethics board. This ensured that we would not need to make any significant changes at a later date in the event that we did not receive approval. As such, we scrape, process and store as little data about an individual as possible, to ensure security and safety, in line with our risk assessment.

As a result of our effective management strategies and agile development method, we were able to meet the requirements of the client effectively, and on time. Our risk assessments allowed us to anticipate potential problems and allowed us to devise solutions and mitigation strategies to minimise the potential risk. This ensured that we were able to complete the project. The testing done late in development shows this.

Testing

Overall, we feel more than happy with the outcome of testing we completed upon software finalisation. This is a testament to how well we have worked as a group, as well as how we mitigated risks through cautious planning and assessment earlier in the project. There are four main tests that we planned and executed earlier in the project, based on our risk assessment and key features that needed to go smoothly. After our program was coded, we ran these tests and the results were very promising - all tests were successful. A large factor behind how successful our tests were, was due to the development and/or use of contingency plans as described in our risk assessment.

One example of such is our contingency for the tweet gathering, which was needed in the event that we could no longer use the API or any other of our developed tweet gathering solutions. This contingency allowed us to focus on other things when that event did in fact come about, and we feel that is reflected in the testing completed. This is analysed further in the Appendices, under testing. Two of the goals we had were ensuring the user could access data older than one year ago, and that accessing older data was as quick as accessing new data. Both of these goals were tested and successful, demonstrating our overall cohesion and dedication to the project.

Conclusion

Our project achieved the requirements of the client, providing a dashboard that can be used to view trends in different areas of misinformation on social media. Our platform provides a way for users to easily understand how misinformation and conspiracy theories are created, grow and eventually fade by displaying the data we have gathered in a variety of ways. Despite the significant difficulties we have experienced, we have created a way to gather relevant tweets from Twitter, in a time when the platform has faced continuous, significant disruption. Overall, we managed to achieve this by working effectively, both as team members and as programmers to develop solutions to problems as they arose, ultimately leading to a successful outcome within the required timeframe.

In the future, there are several ways that our project could be expanded upon even further. Most obviously, more time could be spent using our current data gathering method to expand the size of our dataset. We could also consider expanding the algorithm we created to determine if a tweet was relevant to our project aims, to create new categories or to improve upon existing ones. We could also change our algorithm for scoring the engagement of tweets, to observe how a different ranking system of each form of engagement would influence our results. Additionally, we could introduce new ways of displaying our data on the dashboard to further improve the user experience.

Finally, one area for improvement could be more extensive testing, however, we feel for the current scope of the project, the testing completed is sufficient and demonstrates the functionality of each key aspect of the project, as laid out in the requirements presentation. One example of this would be testing User Accessibility using SUS Scoring (System Usability Scale) to measure the usability of the system. SUS is an industry wide standard used and would give us a good insight into how well our product performs in accessibility compared to other systems.

Appendices

Categories of Misinformation

1. General Conspiracies - the basic idea that a secret organisation is in control of the world, or a significant part of it. (for example, the “globalists” or “cabal”)

2. Covid-19/Vaccines - the miasma of ideas surrounding covid-19 and vaccinations for it and other diseases. (for example, the idea that the MMR vaccine causes autism in children)

3. JFK - the idea that John F. Kennedy’s death was somehow faked or that the real shooter was never caught. (for example, the CIA killed him and used Lee Harvey Oswald as a scapegoat).

4. 9/11 - misinformation surrounding the events of 9/11 (for example the idea that the world trade centres were destroyed by bombs placed within the towers rather than the planes)

5. White genocide - the conspiracy that minorities are moving into majority-white countries in order to replace and eventually eradicate white people

6. False flags - the idea that public tragedies (such as mass shootings) are staged by the government to push their agenda.

7. Qanon - the belief that an anonymous poster on the messaging board 4Chan (and later 8chan and 8kun) was a high-level intelligence operative working with Donald Trump to arrest and prosecute members of a global conspiracy.

8. Russia/Ukraine war - ideas relating to the war in Ukraine, such as the idea that the war is not anywhere near as bad as western media has reported.

9. Antisemitism - conspiracies about the influence of jewish people in global politics (such as the idea that the world is controlled by a “Zionist Occupied Government”)

10. Economics - the belief that the global financial system is controlled by a single organisation (such as the world economic forum)

11. Secret Space Program - ideas surrounding the true nature of human space exploration, aliens and advanced technology

12. Climate Change - discussions of climate change as a fictional concept

13. Satanic panic - the belief that Satanic forces (evil spirits or the devil itself) are involved in global politics in some way (for example the belief that certain politicians are demons)

14. MH370 - the belief that the crashing of planes MH370 and MH17 was intentional

15. LGBTQ+ - conspiracies surrounding the LGBTQ+ community.

16. Flat earth - the belief that the earth is flat + divine creator belief

Tweet Gatherer Code Snippet

```
while True:
    num = random.randint(0,len(people)-1)
    k = people[num]
    del people[num]
    print("looking at tweets of: " + k)
    try:



        for tweet in sntwitter.TwitterProfileScraper(k).get_items():
            newpeople = []

            tweetsLookedAt = tweetsLookedAt + 1
            if tweetsLookedAt % 1000 == 0:
                print("tweets looked at: " + str(tweetsLookedAt))

            if hasattr(tweet, 'inReplyToUser'):
                if tweet.inReplyToUser is not None and tweet.inReplyToUser.username not in people and tweet.inReplyToUser.username not in newpeople:
                    newpeople.append(tweet.inReplyToUser.username)

            if hasattr(tweet, 'mentionedUsers'):
                if tweet.mentionedUsers is not None:
                    for l in tweet.mentionedUsers:
                        if l is not None and l.username not in people and l.username not in newpeople:
                            newpeople.append(l.username)
            if hasattr(tweet, 'rawContent'):
                if "scamdemic" or "plandemic" or "election" or "politics " or " the cabal" or "Shadow government" or "mass shooting" or "deep state "
                    if tweet.user.username is not "elonmusk":
                        tweetCheck(tweet)
                    if newpeople:
                        for x in newpeople:
                            people.append(x)

    except Exception as e: #has to be here as there is a bug in the web scraper library that will break the program otherwise.
        print(e)
```

Word Cloud Generator Code Snippet

```
def make_cloud(lists, topics):
    with open("clouds.txt", "w") as f:
        for week_totals in lists:
            line_items = []
            for topic, count in zip(topics, week_totals):
                line_items.extend([topic] * count)
            f.write(", ".join(line_items) + "\n")
    f.close()

    with open('clouds.txt', 'r') as file:
        lines = file.readlines()

    # create a WordCloud for each line and save it to a file
    for i, line in enumerate(lines):
        # create the WordCloud object for the current line
        wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = [], min_font_size = 10).generate(line)

        # plot the WordCloud image
        plt.figure(figsize = (8, 8), facecolor = None)
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.tight_layout(pad = 0)

        # save the WordCloud image to a file with a name based on the line number
        plt.savefig(f'week{i}.png')
        file.close()

    return f
```

Web Security Code Snippet

```
class user(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.Text, unique=True, nullable=False)
    password_hash = db.Column(db.String(128))

    def __repr__(self):
        return f"'{self.id}', '{self.username}'"

    @property
    def password(self):
        raise AttributeError('Password is not readable.')

    @password.setter
    def password(self,password):
        self.password_hash=generate_password_hash(password)

    def verify_password(self,password,x):
        return check_password_hash(self.password_hash,x)
```

Database Code Snippet

```
class user(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.Text, unique=True, nullable=False)
    password_hash = db.Column(db.String(128))

    def __repr__(self):
        return f"'{self.id}', '{self.username}'"

    @property
    def password(self):
        raise AttributeError('Password is not readable.')

    @password.setter
    def password(self,password):
        self.password_hash=generate_password_hash(password)

    def verify_password(self,password,x):
        return check_password_hash(self.password_hash,x)

@login_manager.user_loader
def load_user(user_id):
    return user.query.get(int(user_id))



class tweets(db.Model):
    username = db.Column(db.CHAR(20), nullable=False, primary_key = True)
    date = db.Column(db.Date, nullable=False, primary_key = True)
    content = db.Column(db.String(350), nullable=False)
    views = db.Column(db.INTEGER ,nullable=False)
    likes = db.Column(db.INTEGER, nullable=False)
    retweets = db.Column(db.INTEGER, nullable=False)
    comment = db.Column(db.INTEGER, nullable=False)
    score = db.Column(db.INTEGER, nullable=False)
    location = db.Column(db.VARCHAR(45), server_default=text('none'))
    categories = db.Column(db.VARCHAR(30), nullable=False)

#Test Data
CSV = pd.read_csv("../DATA.csv")

#New Database
df = pd.read_sql_table(table_name = "tweets", con = engine)
print(df)
CSV = CSV.iloc[: , 1:]
```

Web WTForms Forms Snippet

```
Categories = ["General Conspiracies","COVID-19/Vaccines","JFK","9/11","White Genocide","False Flags"
titles = ['2007-02.png', '2009-01.png', '2009-07.png', '2009-08.png', '2009-09.png', '2009-10.png',

class RegistrationForm(FlaskForm):
    username_new = StringField('Username',validators=[DataRequired()])
    password_new = PasswordField('Password',validators=[DataRequired(),EqualTo("password_confirm")])
    password_confirm = PasswordField('Confirm',validators=[DataRequired()])
    submit = SubmitField('Register')

    def validate_username(self, username):
        user = user.query.filter_by(username=username.data).first()
        if user is not None:
            raise ValidationError('Username already exist. Please choose a different one.')

class LoginForm(FlaskForm):
    username = StringField('Username',validators=[DataRequired()])
    password = PasswordField('Password',validators=[DataRequired()])
    submit = SubmitField('Login')

class CompareForm(FlaskForm):
    Cat1 = SelectField('Category1',choices=Categories)
    Cat2 = SelectField('Category2',choices=Categories)
    StartDate = DateField('Start Date', format='%Y-%m-%d',validators=[DataRequired()])
    EndDate = DateField('End Date', format='%Y-%m-%d',validators=[DataRequired()])
    Compare = SubmitField('Compare')

class CloudForm(FlaskForm):
    CatAll = SelectField('title',choices= titles)
    cloudSubmit = SubmitField('cloudSubmit')

class HomeForm(FlaskForm):
    Cats = SelectField('title', choices = Categories)
    homeSubmit = SubmitField('Submit')
```

A Risk Assessment (also in Interim Report)

Our risk assessment covers both the risks to the completion of our project, as well as the potential risks of changes within Twitter that make parts of our project difficult or impossible, which we have identified as a possibility given the recent change of ownership.

Risk	Likelihood (high, medium, low)	Impact before action	Response strategy	Actions required	Result	Impact after action
Problems with the Twitter API	Medium- due to recent complications with Musk's takeover.	Medium	Actively accept risk- use a contingency plan, but only if the risk happens.	If this were to happen, we could use the sample data provided by the client to simulate the tweets as if they were being received in real-time.	This would impact our ability to retrieve recent data, impacting how current the trends are.	Low
Problems with the database	Medium	High	Mitigate impact	We will use RAID-5 technology to back-up our database. RAID-5 stores the same data across multiple drives.	We would be able to recover any data lost.	Low
Issues with team members attendance	Medium	Medium	Passively accept risk/ transfer risk	If team members are not attending any team meetings or completing any work, all we can do is let our supervisor know.	We would be one team member down, so we would have to take on more work.	Medium
Deadline issues	Medium	Low	Mitigate impact	By using an agile approach to development, we should always have a working prototype. By making sure that we reach our milestones in time, we should not have a problem reaching the deadline.	If we do not complete the project by the deadline, our current prototype should at least meet all of our functional requirements, if not a few of the non-functional requirements too.	Low
Scope creep	Medium	Medium	Mitigate impact	We will refer back to our requirements presentation if we feel like we are going off track. Any new ideas should be shared at weekly meetings and we will decide if it is necessary/ achievable.	We will stick to the original requirements agreed to by the client, focusing on the completion of the functional requirements before expanding the scope.	Low
Issues with the client- unmet expectations	Low	Low	Avoid risk	We will avoid this risk by communicating with the client frequently and demonstrating prototypes. We have set clear goals for the project that the client has agreed to.	Any feedback from the client will be acknowledged and worked on.	Low
Overestimating the team's abilities	Medium	High	Mitigate risk	Set achievable goals and communicate any worries about ability to finish these goals.	Work as a team to help meet targets.	Low
Not getting approved by the ethics board	Low	High	Mitigate risk	Alter project to suit needs of ethics board and then reapply	Push back the timeline as we would have to re-work the fundamentals of the project	Medium

Testing (also shown in presentation)

Requirement:	Test:	Result we were hoping for:	Result:
Information should be presented to the user live.	Two of us tested the website simultaneously.	Both users will be seeing the same information instantaneously	Passed
Collate tweets into a class to store on a database	Create the functionality of a class and database and test it using the sample test data given by the client	Each tweet as a class is stored into a database that can be pulled and pushed onto the database	Passed
The user must be able to switch between the topic that is shown on the graph	A function must be created that can produce a graph based on data given to it. This data must be given through the user input in the website	The function returns a graph based on the topic of data the user requests.	Passed
The user must be able to switch between which graph type is showing the data	A function must be created that can produce a different types of graph based on data given to it. This data must be given through the user input in the website	The function returns a type of graph based on the type of data the user requests.	Passed, for the range of graphs we have implemented