COMP90024 2022 SEMESTER 1
ASSIGNMENT 2 REPORT

TEAM 9

Chenyang Dong
Student ID: 1074314
doncd@student.unimelb.edu.au

Lang Chen
Student ID: 1181506
lachen1@student.unimelb.edu.au

Un Leng Kam
Student ID: 987838
ukam@student.unimelb.edu.au

Ying Zhu
Student ID: 1174301
ying.zhu4@student.unimelb.edu.au

Zhuoya Zhou
Student ID: 1366573
zhuoyaz@student.unimelb.edu.au

May 26, 2023

ABSTRACT

This is the paper for *ELEFEEL*, a cloud-based solution developed to visualise trends between the sentiment of Australian Tweets discussing politics and Federal Election results as part of Group 9 for COMP90024's Assignment 2. The report will discuss methods and choices made in the process of *ELEFEEL*'s development, as well as their limitations.

Keywords Cloud Computing · Melbourne Research Cloud · SUDO · Twitter · CouchDB · Mastodon · Social Media · Sentiment Analysis · Ansible · Docker · Flask · Vue · Google Map API

Contents

1 Introduction

Elections are an important part of political life in Australia, held at least once every three years (Commission [2023a]) to elect new Members of Parliament and Senators, determining the governing Party at national level as well as key Federal Leadership Positions such as the Prime Minister.

In recent years, technological advancement in areas such as the internet, web and mobile devices have increased usage of social media amongst the population in developed nations like Australia. As such, data associated with social media posts - including but not limited to the post content, time of post and geolocation - have become a powerful dataset. With Gen Z being a key force in the evergrowing user-base of social media services like Twitter (Pietsch [2021]), and gradually reaching the age of mandatory voting in Australia, the Australian social media dataset is becoming ever more valuable in analysing election patterns and has the potency for predicting outcomes.

Though not all Australians will use and express themselves on social media, it is believed that as time moves on and more young people becoming eligible to take the mandatory vote, the social media dataset will only ever increase its ability in being representative of Australia's voice, which will in turn largely correlates with election results. Thus, the key problem that this project attempts to tackle is: **to what extent do social media sentiment data currently correlate with recent election results, through building a web application that allows rapid visualisation and analysis of relations between sentiment statistics from Twitter and Mastadon and the 2022 election results**.

ELEFEEL, the principal outcome of this project, builds towards the potential for social media sentiment analysis to replace traditional Think Tank surveys or Newspaper polls that reflect present political trends, fostering a new branch of political analysis that leverages rapid technological advancements.

The application was built on the Melbourne Research Cloud (MRC), utilising free resources provided by the subject COMP90024. Other key components of the web application includes Apache CouchDB, Flask, Gunicorn, Vue and Ansible and Docker to serve as the database, back-end, front-end and deployment tools respectively. Data were sourced from the Australian Electoral Commission (AEC) and the Spatial Urban Data Observatory (SUDO), which were processed using Python. R was also used to perform data and scenario analysis using outputted results.

2 User Guide

Our project source code repository is located at the GitHub repository `https://github.com/doncd-p/COMP90024-23S1-A2-Australia-Social-Media-Analytics.git`. The project structure is allocated as follows:

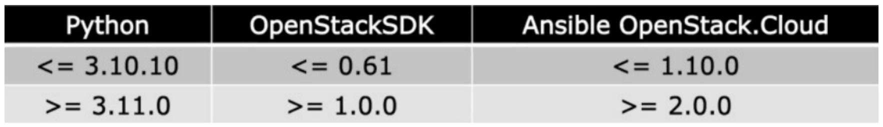
```
├── ansible/ ........................ Ansible scripts for orchestration
├── data/ ............................ Data used in the project
├── doc/ ............................. Documentation files
├── harvest/ ......... Scripts for harvesting data from Twitter and Mastodon APIs
├── scripts/ ................. Additional scripts used in the project
├── src/ ................... Jupyter notebooks used for data analysis
│   ├── backend/ ............ Back-end code for the web application
│   └── frontend/ .......... front-end code for the web application
├── .gitignore
├── LICENSE
└── README.md
```

A video demonstration of the web application can be found here https://www.youtube.com/playlist?list=PLdCTupRD-v9g_HvDFUE4nTTCGT8tjNqZ7.

2.1 Requirements

2.1.1 Package Requirements

In order to deploy the system, please ensure you have installed the following packages on local host depending on your python version.



Python	OpenStackSDK	Ansible OpenStack.Cloud
<= 3.10.10	<= 0.61	<= 1.10.0
>= 3.11.0	>= 1.0.0	>= 2.0.0

Figure 1: Ansible version requirements

To install Ansible on the localhost, view the following installation guide on `https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html` to install Ansible.

2.1.2 Directory Requirements

Clone the project's repository from `https://github.com/doncd-p/COMP90024-23S1-A2-Australia-Social-Media-Analytics.git` and change directory into `ansible`.

2.2 System Deployment

2.2.1 Create Instances

Instances, volume, security groups are created on Melbourne Research Cloud via the shell script `create_mrc_instances.sh` which utilises the Ansible playbook `create_mrc_instances.yaml` and `mrc_settings.yaml` located under the directory `vars`. An inventory file `instance_addresses.ini` is created after the script with saves the IP addresses of instances. The roles of the playbook is explained in table 1:

Role	Functionality
`openstack-common`	Install required dependencies on host (openstack)
`openstack-images`	Show all available images for user
`openstack-volume`	Create volumes as per defined on `mrc_settings.yaml`
`openstack-security-group`	Create security groups as per defined on `mrc_settings.yaml`
`openstack-instance`	Cloud instances creation according to demand as per defined on `mrc_settings.yaml`

Table 1: Functionality of Roles of `create_mrc_instances.yml`

Run `./create_mrc_instances.sh` in terminal to create instances, assuming you are in the `ansible` directory.

2.2.2 Configure Instances

For this task, instances are configured with correct service by installing Docker and Docker Compose onto the created instances and cloning the git repository on to the instances. The shell script to configure the instances is `configure_mrc_instances.sh` which utilise the playbook `configure_mrc_instances.yml`. The role of the playbook is explained in table 2:

Role	Functionality
`openstack-config`	Install Docker and Docker Compose and pull the respective git repository

Table 2: Functionality of Role of `config_mrc_instances.yml`

Create a `git_credential.yaml` under the directory `vars`, using `ansible-vault create git_credentials.yml`, enter your vault password. Run `./config_mrc_instances.sh` in terminal to config instances, assuming you are in the `ansible` directory.

2.2.3 Generate Hosts for application

For this task, a python script was ran to update the inventory file `instance_addresses.ini` to allocate applications to instances according to our teams system architecture design.

```
cd inventory
python3 configure_addresses.py
```

2.2.4 Deploy CouchDB Cluster

For this task, instances that are allocated to host the CouchDB database are being setup via Ansible to create Docker containers with the CouchDB images and they were then further configure to form a cluster. The shell script to deploy the cluster is `deploy_couchdb_cluster.sh` which utilises the playbook `deploy_couchdb_cluster.yml`. The role of the playbook is explained in table 3.

Role	Functionality
`deploy_couchdb_cluster`	1. Create a Docker bridging network 2. Create Docker containers with images of CouchDB 3. Configure CouchDB cluster 4. Add instances to the CouchDB cluster 5. Check if cluster is setup

Table 3: Functionality of Role of `deploy_couchdb_cluster.yml`

Run `./deploy_couchdb_cluster.sh` in terminal to create the database cluster, assuming you are in the `ansible` directory.

2.2.5 Deploy Mastodon Crawlers

For this task, Mastodon crawlers are deploy to instances defined by the user through configuring the `mastodon_info.ini` inventory file, where the IP address of the desired instance of deployment is set, the Mastodon URL of interest, and the token of the Mastodon server are enter under the heading `[CRAWLERS]`. Crawlers that want to be disabled can be done through entering the name of the user defined crawler under the heading `[DISABLE_CRAWLERS]` for the inventory file. The shell script to deploy crawlers is `deploy_crawler.sh` which utilises the playbook `deploy_crawler.yml`. The role of the playbook is explained in table 4.

Role	Functionality
`deploy_crawler`	1. Docker remove previous crawlers 2. Generate Docker compose files for crawlers which passes the Mastodon URL and token to the crawler 3. Create the Docker containers for the crawlers 4. Disable any crawler containers as desired by the user

Table 4: Functionality of Role of `deploy_crawler.yml`

Run `./deploy_crawler.sh` in terminal to deploy crawlers, assuming you are in the `ansible` directory.

2.2.6 Deploy Back-end

For this task, instances that are allocated to host the back-end are being setup via Ansible to create a Docker container with the `python:3.11.3-slim-bullseye` image. The shell script to deploy the back-end is `deploy_backend.sh` which utilises the playbook `deploy_backend.yml`. The role of the playbook is explained in table 5.

Role	Functionality
`deploy_backend`	1. Docker remove the previous back-end container 2. Generate Docker compose file for back-end to pass in the CouchDB IP addresses, and credentials 3. Create the Docker container for the back-end

Table 5: Functionality of Role of `deploy_backend.yml`

Run `./deploy_backend.sh` in terminal to deploy back-end, assuming you are in the `ansible` directory.

2.2.7 Deploy front-end

For this task, instances that are allocated to host the front-end are being setup via Ansible to create a Docker container where it is first build with `Node.js` and `NGINX` was used to host the build of the front-end. The shell script to deploy the front-end is `deploy_frontend.sh` which utilises the playbook `deploy_frontend.yml`. The role of the playbook is explained in table 6.

Role	Functionality
`deploy_frontend`	1. Create a `.env` file which contains the backend IP address to allow the front-end to request the correct API 2. Docker remove the previous front-end container 3. Create the Docker container for the front-end

Table 6: Functionality of Role of `deploy_frontend.yml`

Run `./deploy_frontend.sh` in terminal to deploy frontend, assuming you are in the `ansible` directory.

3 Cloud System Architecture and Deployment

3.1 System Architecture Design Diagram

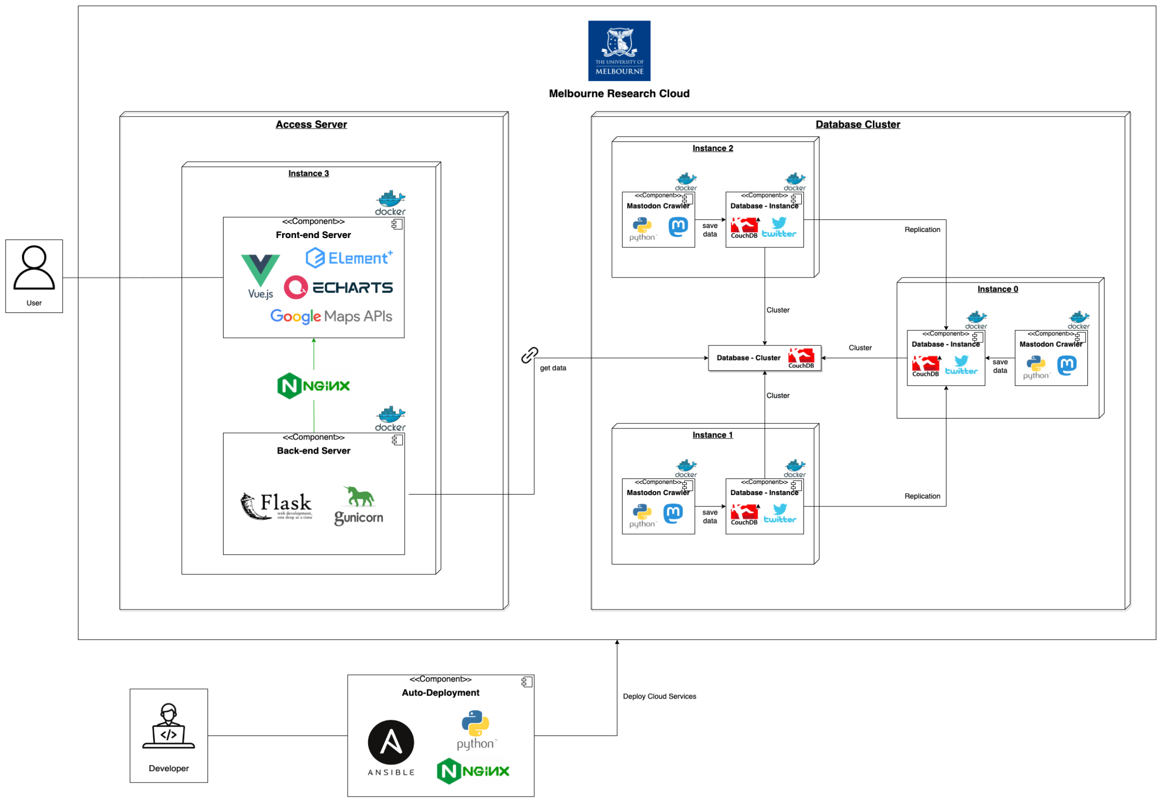


Figure 2: System Design Architecture

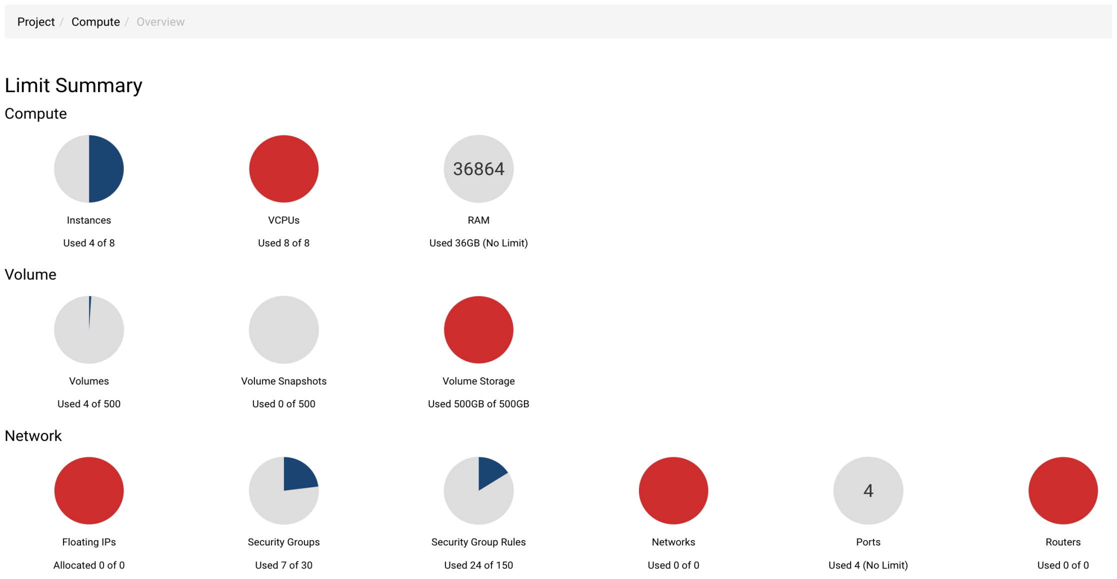
This design diagram describes the overall structure of this project's cloud system. We have decided to create four instances, where three of the instances will form our CouchDB cluster storing our twitter data and SUDO data, where also Mastodon crawlers will be deploy on each instance crawling toots into the database, the remaining instance will hold our front-end and back-end that will be host using `NGINX` and `gunicorn`. The deployment of the system is through Python and Ansible scripts which will help in creating Docker containers on the instances where the applications will run.

3.2 Melbourne Research Cloud

Our solution is implemented via the Melbourne Research Cloud (MRC), which offers a free on-demand computing resources to researchers at the University of Melbourne. MRC uses a private cloud deployment model and provides Infrastructure-as-a-Service (IaaS) cloud computing service, this service makes it easy for researchers to quickly access scalable computational power as their research grows, without the overhead of spending precious time and money setting up their own computing environment. TheUniversityOfMelbourne.

3.2.1 Allocated Resources

Our team were allocated with 8 instances, 8 vCPUs, 36 GB of RAM, and 500 GB of volume. For our system's implementation, we decided to create 4 instances each with 2 VCPUs and 9 GB of RAM, as shown in Figure 3



Limit Summary

Compute

Instances Used 4 of 8
VCPUs Used 8 of 8
RAM Used 36GB (No Limit)

Volume

Volumes Used 4 of 500
Volume Snapshots Used 0 of 500
Volume Storage Used 500GB of 500GB

Network

Floating IPs Allocated 0 of 0
Security Groups Used 7 of 30
Security Group Rules Used 24 of 150
Networks Used 0 of 0
Ports Used 4 (No Limit)
Routers Used 0 of 0

Figure 3: Instance Usage

3.2.2 Advantages of Melbourne Research Cloud

Portability

Since MRC is based on OpenStack, which contains many open source software, the deployment of this project is compatible with any hosting provider that supports the OpenStack platform. This facilitates ease of migration in the future, if ever required.

Security

MRC offers high-level security for our project, ensuring that allocated cloud resources are exclusive to our group. This allows us to configure the cloud infrastructure and systems as per our security needs. Besides, we can trust MRC as it adheres to data protection principles outlined in relevant legal legislation.

Performance

The private deployment model and the organisation's intranet-based firewall that ensures high network performance for users. Furthermore, the cloud server's location within the Melbourne University data center enables efficient data transfer between instances.

Adaptability

MRC offers easy customisation, flexible resource allocation, and avoids vendor lock-in issues that are prevalent in public clouds. MRC allows users to easily customise the hardware and other resources, providing a scalable virtual environment.

Compliance

Utilising MRC guarantees compliance. As long as we follow MRC's guidelines, legal constraints for storing sensitive data on the cloud are easily manageable.

Cost

The MRC's cost-effectiveness is a notable benefit. It eliminates the substantial initial expenditure associated with on-premise infrastructure, including hardware costs. Furthermore, by using MRC, the administrative responsibilities are significantly reduced. This involves negating electricity bills and procurement costs related to applications and infrastructure. Additionally, MRC is free of charge to use for us students.

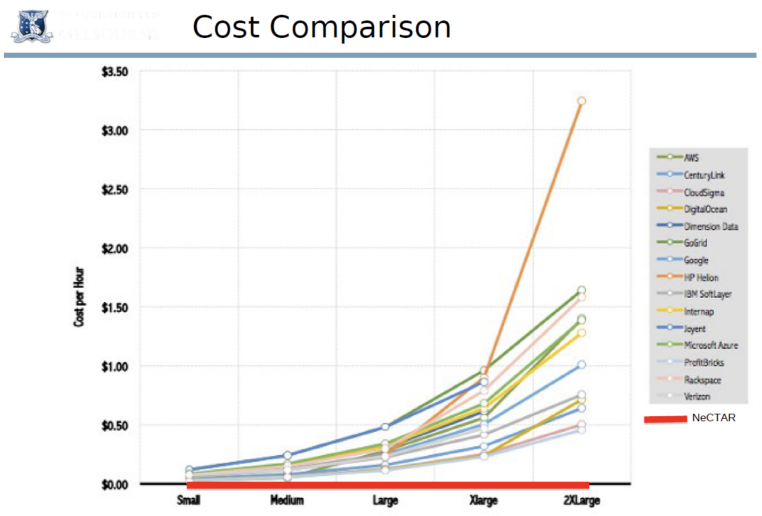


Figure 4: Cloud Service Price Comparison

3.2.3 Disadvantages

Responsiveness

Compared to many popular public cloud platform such as Amazon AWS and Microsoft AZURE, a "ticket" needs to be placed for the cloud administrator to check into problems on MRC, and the time of response is not guaranteed. Thus, if bugs arises with MRC it could impact development for a certain period of time.

Technical Knowledge

Despite the demonstrations of MRC delivered by the COMP90024 Teaching team, it was still time-consuming for the team to take time and get familiar with MRC, and hence reduced the development time spent on implementing the solution.

Accessibility

The access of MRC requires using the network within the University of Melbourne, making deployment of instances on MRC unavailable through public IP addresses.

3.3 Ansible

Ansible is an IT automation engine that simplifies cloud provisioning, configuration management, and other IT needs. In this project, Ansible supports the implementation of 'dynamic deployment' by communicating with both physical and virtual remote hosts through SSH. Ansible's rich APIs, compatible with the OpenStack platform utilised by MRC, streamline the deployment process and reduce the complexity of managing multiple remote hosts. Below lists the deployment of our system:

1. Create instances and configured resources for the instances according to our system design
2. Install dependencies require each instance and clone code from Git and mount the corresponding volume.
3. Deploy CouchDB containers on instances and create a database cluster.
4. Deploy Mastodon harvester on each CouchDB instances.
5. Deploy the back-end.
6. Deploy the front-end.

3.3.1 Detailed Explanation of Deployments

MRC Instance Creation and Configuration

We first have to ensure correct installation of Python and relevant resource packages for Ansible playbook operation. This then allow us for the creation of security groups, which defines network access rules done by opening necessary ports to the instance. Volumes for each instance is also generated as per defined by the system design, instance 0, 1, 2 is given 150 GB for storage, and instance 3 the access server is given 50 GB. Afterwards instance flavour, network, key-pair is set up for the instances and hence the creation of the instances.

We then wait for the establishment of the instance, while we wait we can create a git credential file `git_credentials.yml` under `vars` directory, where the git username and password will be enter by the user. It was further encrypt using `ansible-vault encrypt git_credentials.yml` and a vault password was entered to ensure safety., once instances are created, it will install the require dependencies onto instances such as Docker, python and pip. It will then clone the git repository on to the instances which allows future deployment.

Deploy CouchDB Cluster

1. A internal Docker network of "172.25.0.0/16" is created
2. Create Dokcer containers on each instances, with the `couchdb:3.2.0` image, and the CouchDB credentials are set up by passing Docker environment variables. Ensure the following ports are opened to ensure visual UI of CouchDB, internal communication of CouchDB, and the erlang port.
   - 5984
   - 9100
   - 4369
3. Sent request to the CouchDB interface to enable cluster
4. For each node in the 3 node set up add it to rest of the 2 nodes to set up the cluster.
5. Create the `_users` and `_replicator databases`
6. Check if the cluster is set up by sending request to see the `_membership` of the nodes.

Deploy Mastodon Harvester

First create an inventory file call `mastodon_info.ini`, where it should a group `[CRAWLERS]` should be defined, here the user define:
`<name of crawler> ansible_host=<IP address of desire deployment> mastodon_server=<URL of server> mastodon_token=<Server token> couchdb_ip=<IP of CouchDB server>`
A Dockerfile was created to run the harvester, running on the `python:3.9` image and also install the required python libraries.

The steps of deployment are as follow:

1. Docker Compose down any existing crawler containers, ignore errors as they may be no Docker Compose file
2. Generate Docker Compose files on the instance according to the define group `[CRAWLERS` and pass in the variable as defined as environment variables using the docker-compose.j2 template
3. Run the harvester through Docker Compose
4. Disable any crawlers defined in the `DISABLE_CRAWLERS` in the `mastodon_info.ini`

Deploy Back-end

First create a `gunicorn` configure file call `gunicorn_config.py`, where the user configure the number of workers and settings, usually 2-4 times the number of CPUs. A Dockerfile was created to run the back-end, running on the `python:3.11.3-slim-bullseye` image and also install the required python libraries. The steps of deployment are as follow:

1. Docker Compose down any existing back-end containers, ignore errors as they may be no Docker Compose file
2. Read the `intance_addresses.ini` file, and convert the contents to string

3. Extract the IP addressees of the CouchDB cluster, and concatenate them into a string of `<IP address1>,<IP address2>,<IP address3>` and register as a Ansible fact
4. Generate Docker Compose files on the instance according to the `backend_couchdb_settings.ini` and the previous set fact where they are passed to the back-end as environment variables
5. Run the back-end through Docker Compose

Deploy Front-end

First create a `NGINX` configure file call `nginx.conf`. A Dockerfile was created to run the front-end, running on the `node.js` image and was split into a build and production stage. During build stage the static `VUE` application was build and ensure the `.env` file which contains the back-end IP address was copied. In the production stage, `NGINX` was use to host the front-end

The steps of deployment are as follow:

1. Create a .env file which stores the back-end IP address
2. Docker Compose down any existing front containers
3. Run the front-end through Docker Compose

3.4 Containerisation

Containerisation, a lightweight variant of full machine virtualisation, encapsulates an application within a container accompanied by its operating system.

3.4.1 Isolation

This characteristic enables isolation between applications and their dependencies, avoiding potential disputes and ensuring the uniform application behavior across varied environments.

3.4.2 Portability

Containerisation packages the application along with its environment, which includes all the dependencies the application needs to run. This means the application can run on any machine that has the containerisation software regardless of the underlying operating system and hardware.

3.4.3 Efficiency

Compared to complete virtual machines, containers are resource-efficient and lightweight as they utilise the host system kernel instead of demanding a operation system.

3.4.4 Microservices Architecture

Each microservice can be encapsulated within its own container, ensuring that it runs in a well-defined environment and that it is isolated from other services. This aids in creating a loosely coupled architecture which allows individual microservices to be developed, deployed, and scaled independently.

3.4.5 Why Docker?

Docker is one of the most popular containerisation platforms due to its ease of use, widespread support, and comprehensive documentation. Docker containers are based on open standards, enabling containers to run on all major infrastructures and the cloud.

3.4.6 Why Docker Compose?

Docker Compose is a tool for defining and running multi-container Docker applications. With Docker Compose, you use a YAML file to configure your application's services. With a single command, we can start the containers, define their dependencies, and manage their lifecycle which greatly simplifies the deployment process.

3.5 Dynamic scaling

Dynamic scaling is a key aspect of our system design, allowing for flexibility and optimisation in terms of resource allocation and utilisation.

3.5.1 System loose coupling design

The system architecture we've implemented is based on a loosely coupled model. Loose coupling enables greater scalability as each component can be scaled up or down independently based on specific needs or resource constraints. Since each module operates independently, problems or changes in one module are less likely to impact others. This isolation can increase system stability and simplify debugging. Also, changes can be made to individual modules without requiring a total system overhaul. This can increase development speed and enable more rapid deployment of new features or improvements.

3.5.2 Ansible

Ansible's functionality inherently supports the expansion or reduction of system components as needed. Ansible allows for the creation of custom playbook of roles using YAML, a language designed to be easily human-readable and write able. These roles provide a way to reuse code and execute common tasks, which is essential for scalability. For example, if you need to deploy a new instance you can simply use a pre-defined role. Also, unlike a static inventory, which requires manual updates, a dynamic inventory in Ansible allows for the real-time management which frees the need of user to manually change variable contents for future deployment

3.5.3 CouchDB Cluster

When constructing our system, we opted to utilise a CouchDB cluster instead of a single node database setup. As the database load increases, new nodes can be added into the cluster to distribute the load more evenly. CouchDB's built-in replication and conflict resolution mechanics ensure the data remains consistent across all nodes. The multi-node setup allows for load balancing and failover support, enhancing the robustness and reliability of the system. If one node experiences an issue, the other nodes can continue to function, minimising potential system downtime.

3.5.4 Mastodon Harvester

The deployment of harvesters, essentially exhibits dynamic scaling. Depending on the requirements and resources available, users can adjust the number of active harvesters, thus modulating the rate of toot collection. However, this elasticity is limited by the available Mastodon servers and the possession of access tokens. With these conditions met, the system can adapt to the fluctuating needs of the user.

3.5.5 Docker

As mentioned in section 3.4, Docker is utilised due to isolation, portability, efficiency, and the aid of creating a microservice architecture ensures scaling of the system depending on load and user demand.

3.6 Error Handling

In the deployment process, for every component of the system (front-end, back-end, harvester, CouchDB cluster) it was first deployed locally for testing, then unit testing and integrating testing was performed. It was then deployed to the cloud where once again thorough testing was conducted, including smoke testing to ensure critical component was working. Logging was kept and monitored to provide insights if anything goes wrong. Also, wait time was set up to ensure application was running to ensure there are no inconsistency.

3.7 Challenges

3.7.1 Cluster Configuration

According to the CouchDB documentation, after the cluster is set up, a request should be sent to the API to check for completion, however, sometimes it would return an error indicating that credential are incorrect. Yet, using the same credentials to access the database has no problems, it was later discovered that it seems to be a bug of CouchDB 3, and instead we asked for `_membership` from CouchDB to see if under `all_nodes` all the IP addresses of the CouchDB instances are included.

3.7.2 Crawler Deployment

The approach taken was to dynamic create `docker-compose.yaml` files as required. However, this only worked when we first ran of the deploy script, it was found out that without specifying a Docker network to be used, Docker compose will create a new Docker network every-time, causing the container to run into the conflict of not knowing which Docker network to use as there will be multiple networks with the same name. Hence, it was resolved by specifying a Docker network name that will be used for each specific crawler and the Docker network created if it does not exists.

4 Dataset and Wrangling

4.1 Election Data

For reference, Australian voters vote for a member of parliament in their local electorate, with the winner contributing one "win" towards the Federal (National) level tally. The Party/Coalition of Parties that wins more than 76 out of the 151 available electorates becomes the national governing Party/governing Coalition, and the leader of the governing party/governing coalition becomes the Prime Minister. Regardless of which Party becomes the national governing Party, the local governing party is determined solely by the original votes.

4.1.1 Retrieval

Although the analysis is performed on the 2022 Australian Election, having taken place on the 21st of May 2022, both the election results from the 2022 and 2019 election were retrieved from the AEC website (Commission [2023b]) so analysis on change of governing party in each electorate and pre-election sentiment analysis in electorates governed by different parties could be performed. The key data downloaded were the "Two Candidate Preferred (TCP) votes by party by division" for both 2019 (Commission [2019]) and 2022 (Commission [2022a]) elections.

For reference, the Australian Labour Party (ALP) won the 2022 election whilst the Liberal National Coalition (consisting the Liberal Party (LP), National Party of Australia (NP) and the Liberal National Party of Queensland (LNP)) won the 2019 election.

The 2022 Australian Electorate Boundaries shapefiles were also retrieved (Commission [2022b]). Election boundaries are slightly adjusted by the Electoral Commission before each election so to ensure each electorate approximately has the same number of voters and thus maintain election integrity, but it was decided to only use the electorate boundaries of the election that is being analysed (in this case 2022) and not also the 2019 (i.e. for aggregating sentiment or number of tweets pre-election) because it would make the pre-election statistics incomparable against the 2022 election results.

4.1.2 Wrangling

Both the 2022 and 2019 TCP contained 302 rows of data, containing features such as the absolute number of preferential first votes received, the candidate's affiliated party, the electorate and other relevant columns for both candidates of 151 electorates. The data fields extracted overall from each of the two TCP files included:

- *2022 winning party*
- *2022 winning party's winning percent*
- *2019 winning party*
- *Whether there was a change in winning party in 2022*

These statistics were aggregated from iterating through each pair of candidates in Python's Pandas package, grouped by electorate, and performing ArgMax or division. The "whether electorate saw a change in winning party in 2022" statistic was created after merging the cleaned 2019 and 2022 DataFrames. The cleaned data was a DataFrame with 151 rows and the aforementioned columns, with their electorate unique identifiers. The 2022 winning party column in the data will enable denotation on the application of whether the majority preferential vote-getting party* (refer to limitation in section 9.1.1) in the electorate was the national governing Party. The Polygons/Multipolygons denoting the electorate boundaries were merged onto this original DataFrame using the Python GeoPandas package.

The centroid of the geolocations were also extracted from each Polygon for the front-end to add icons on the map denoting whether the majority vote-getting party in an electorate was the national governing party. If an electorate contained a MultiPolygon (often including island territories), then the largest polygon within the MultiPolygon would first be extracted before it's centroid was extracted. This algorithm solves the problem of some centroids being placed within the ocean, or other electorate centroids not being a good geographic representation of the region.

These data were converted into JSON formats for storage on CouchDB, after merging with SUDO data.

4.2 Census Data from SUDO

4.2.1 Retrieval

Three national census datasets (Highest Level of Education, Weekly Household Income, Population by Age) were retrieved from the SUDO website (e Research Group [2018]), with data were grouped by Local Government Area (LGA).

4.2.2 Wrangling

The education level was determined by the average highest level of high school education completed by the population in each LGA in Australia. The raw data provides the count of population in each LGA who completed 8 to 12 Years of schooling. The average "Highest Level of Education Completed" was calculated by dividing the counts for each level of education by the total LGA population.

For the "Age" dataset, as data were categorised by binning, an algorithm was used to recover an estimate for the average age in the LGA. Simply speaking, the age of every individual in a bin was assumed to be the median between the bin lower and upper bound, and a mean of the estimated recovered age of all individuals in the LGA was used as the estimate of the average age in the electorate. For example, there were 1848 people aged 20-24 in the dataset, so the algorithm assumes all 1,848 people were 22 years old. All individuals in the special case of the age bracket "80 or over" take an estimated recovered age of 80. The age group 15-19 is excluded from calculation as the individuals in this group are not eligible to vote.

The average income for all households in each LGA is calculated from the income dataset by the same algorithm as Age. The average income for households with a negative or nil income is estimated to be 0, while the average income for households with an income over AU$4000 is estimated to be $4000. The migration count (both intrastate, interstate and overseas between 2020 and 2021) contained in the Age dataset was also aggregated in this way.

As a result, four features for each LGA were extracted from the SUDO dataset:

- *Average Education Level*
- *Average Age of Voters*
- *Average Weekly Income*
- *Internal and Overseas Migration*

As election results are grouped by geographic unit of electorates rather than LGAs - the geographic granularity of the census data a proportioning algorithm [1] was employed to estimate the aggregated statistics by electorate level.

4.3 Twitter Data

4.3.1 Source and Retrieval

As Twitter changed its API policy in February 2023, data for this project was provided by the COMP90024 teaching team in the form of a 63GB JSON file containing 52.5 million tweets posted in Australia.

From Exploratory Data Analysis only 3.2 million tweets contained geolocation location of posting, given in the form of bounds of latitude and longitude, forming a rectangular bounding box which obfuscates the precise location that the tweet was made to achieve K-anonymity. It was decided to only perform analysis on tweets that had geolocation (as it was essential to differentiate which electorate the tweet came from). PySpark - the Python API for Apache Spark - was employed to tackle this big data problem, as 63GB was much too large to fit on the local RAM. PySpark performs transformations of data of size larger than RAM size by iterative reading the file into the RAM in chunks; it is also optimised under the hood via algorithms which re-orders the execution of a sequence of transformations on a Resilient Distributed Dataset when multiple actions are performed. For this dataset, transformation were executed in 471 stages.

As one of the important metrics to collect data from tweets on was the text's sentiment, it was imperative for the texts to be in English. All non-English tweets were thus discarded and only 2.5 million tweets remained for analysis.

4.3.2 Wrangling

The initial pre-processing was performed using PySpark, to reduce the data size both row-wise and column-wise to a level capable of being handled in the local RAM on Python Package Pandas. Each tweet contained lots of information which were deemed unimportant for this analysis; to save space on CouchDB and time for uploading/creating indexes and views, the only fields from the raw data that remained were:

- *tweet id (unique id)*
- *account id of tweeter*
- *text of post*
- *time of post (date)*
- *time of post (week relative to election)*

The "week" document field was generated in data processing for the purpose of view creation on CouchDB, as the flexibility around the Map function on CouchDB is much less than that on a Python environment.

To further perform analysis, the SentimentIntensityAnalyser from Python's NLTK package (NLTK [2016]) was invoked and applied to each tweet's text. The rule based sentiment analyser returns three scores:

- *negative score*
- *neutral score*
- *positive score*

which are all within [0, 1] and sums to 1, thus acting as pseudo-probabilities for each the sentiment of the tweet's text. A further *"compound score"* is also returned as part of the output, and is an aggregation of sentiments of each word in the sentence. Whilst the official formulation was not published by NLTK, given the prominence of the package in NLTK, this score is more reliable than finding a way to aggregate the three aforementioned scores arbitrarily. In practice, this analyser is also highly feasible when dealing with big data, capable of processing close to 7000 tweets per minute (115 per second). This is one of the key advantages of rule-based analyser compared to Deep Neural models like Bidirectional Encoder Representations from Transformers (BERT), along with that it doesn't require further training/fine-tuning.

A label is also added to each tweet if it contains keywords which are related to Australian Politics, according to political research paper ***Keywords in Australian Politics*** (Smith et al. [2006]). This allows for analysis to be performed either on "politically related Tweets", or all Tweets in Australia, but in the end only the analysis on political tweets were retained in the final design to prevent over-complicating application features.

Finally, to create the feature denoting which electorate the tweets are from, an algorithm [2] which first finds the centroid of the bounding box and then finds which electorate lies within is applied. The processed tweets were then uploaded to CouchDB (see section 5.1).

4.4 Mastodon Data

4.4.1 Harvesting

Refer to section 3.3.1.

4.4.2 Wrangling

Mastadon toots are processed on-the-fly when they are harvested by harvesters, and immediately stored into CouchDB.

As mastodon toots do not come with a geolocation, and are not always posted in Australia, the use case for them in our analysis is limited. However, we still processed them almost the exact same way as tweets (without the *electorates* field), and used them in one comparison analysis with the Twitter data.

Mastodon Toots also mandates the specification of keywords when querying. Since all toots are thus either full or partial matches of political keywords, the tag *"partial match"* was recorded instead of *"is political"*. The processed toots were then uploaded to CouchDB (see section 5.1).

5 Database

CouchDB was chosen as the database for this project due to its NoSQL properties (i.e. not requiring a strict schema) and its in-built MapReduce paradigm of functions, making it suitable for storing and querying Big Data.

5.1 Schemas and LoadIn

Three databases were created on CouchDB to store the social media data: two non-partitioned database for tweets and toots, and a partitioned database on the 151 electorates to store twitter data.

All mastodon toot data were put into the database as they were crawled in, with cleaning occurring on the fly. Cleaned tweets were bulk uploaded to CouchDB via a python script, uploaded in batches of 10,000, taking roughly 20 minutes to upload 2.5 million. The unique tweet/toot id were deliberately stored in each document with key *"_id"*, so CouchDB will use it as its unique document key in the database. This way, problems like duplicated toots from the same toot being harvested by different Mastodon Harvesters are overcome.

The same corpus of tweets were stored in separate partitioned and non-partitioned databases because it was found queries for documents with top positive and negative scores were far slower on non-partitioned databases to the point of infeasibility, whilst the partition mechanism allows for rapid retrieval. Meanwhile, back-end APIs often required data from all electorates, and partitioned databases which requires users to specify the partition name (electorates) in every request will not serve this purpose.

The CouchDB is also distributed on a cluster across three nodes, creating deliberate redundancy which seeks to improve availability of the data as any node is able to answer requests, increasing its fault-tolerancy. All replications happen automatically when data is loaded into one node's CouchDB, and the re-balancing of shards (horizontal partitions of a database which improves performance through parallelising computation on different nodes) as a result of additional/deleted nodes are completed automatically in the background by CouchDB. However, due to the sharding mechanism, if two out of the three nodes fail, then between a quarter and a half of the data will be inaccessible, as each node stores different shards of the dataset and the remaining node will certainly not store all the shards.

5.2 Views and Indexes

Views with compound keys of $[time, electorate]$ were produced by deliberately designed Map functions, allowing swift aggregation of data either by time or by electorate or both in the back-end. In particular, *"date"* allows aggregation by both days and month.

The non-partitioned tweets database had 4 Views: *(pseudo-)number of tweets with each sentiment tag by day*, *average political sentiment by day*, *average political sentiment by week*, *number of political tweets*.

- *(Pseudo-)number of political tweets with each sentiment tag by electorate by day* returns $\{[date, electorate] : [pos\_score, neu\_score, neg\_score]\}$. This is used for the pie charts that appear when each electorate is clicked. The map function is defined in appendix and the reduce function is the default *"_stat"*. Wrangling and aggregation of View's API output is performed in the back-end to get data into this state.

• *Average political sentiment by day* returns $\{[date, electorate] : avg\_sentiment\}$, where sentiment is determined by the average compound scores of all docs with this key with *"is_political"* field = 1. This is used for the heatmap, line graph when each electorate is clicked and multiple dashboard features. The map function is defined in appendix and the reduce function is the default *"_stat"*. Wrangling and aggregation of View output is performed in the back-end to get data into this state.

• *Average political sentiment by week* is same as the previous View, except keys are by week rather than by month to support the alternative granularity.

• *Number of political tweets* returns: *{null: {percentage of political tweets, count of political tweets, count of all tweets} }* . This is used for the comparison table of tweets and toots and uses the reduce function.

• Three Indexes were built on this database: *["is_political"]*, *["date", "electorate", "is_political"]*, *["week", "electorate", "is_political"]* to support fast retrieval of the View via API, as reduce functions rely heavily on Indexes for fast performance.

The partitioned tweets database had no Views as its sole purpose was for Mango Queries of top five highest positive sentiment score and negative sentiment score tweets within each electorate.

• It had two Indexes: one on *["positive score"]*, one on *["negative score"].*

The non-partitioned toots database had 1 View: number of partially matched toots

• *Number of partially matched toots*: returns *{null: percentage of partially matched toots, count of partially matched toots, count of all toots }* . This is used for the comparison table of tweets and toots.

• It had one Index on *["partial_match"].*

5.3 Challenges

Refer to section 9.2.

6 Back-end

Deployed on Instance 3, as shown in Fig.2, our back-end server is the cornerstone of our system's functionality. Developed using the Python language with the Flask web development framework, it facilitates access to our services via RESTful API. It retrieves the necessary data, stored in the JSON format within our CouchDB database, and promptly returns it for front-end visualisation.

6.1 Flask

We choose Flask for the back-end of our system since it provides a lightweight and flexible framework for building web applications. In comparison to other web frameworks like Django or Pyramid, Flask stands out due to its simplicity and modularity. This allows us to start small and add more complexity when necessary, offering a high degree of flexibility in expansion. Its easy-to-use routes and endpoints are one of its key features that enable rapid data handling and efficient request-response cycles. It facilitates the separation of concerns by allowing us to organise the codebase based on URL patterns and the associated functions. This promotes clean and maintainable code structures, making it easier to manage different parts of the application's functionality such as map and dashboard in our system.

6.2 Gunicorn

Gunicorn is a Python HTTP server that interfaces with our Flask application, acting as a Web Server Gateway Interface (WSGI). It enables the app to serve multiple concurrent requests without blocking, enhancing its overall performance. We chose to deploy our Flask app with Gunicorn due to its efficient handling of worker processes. For our application, we have configured Gunicorn with 4 worker processes considering both the requirements and instance configuration. This allows the server to manage and satisfy the largest possible number of concurrent requests at one time from the front-end, optimising the responsiveness and robustness of our application.

6.3 RESTful API

In our system, the RESTful APIs are implemented using the Flask-RESTful extension. This framework not only simplifies API creation but also encourages best practices with its streamlined setup. Also, it empowers us to structure our resources through a class-based approach, specifically utilising the *Resource* class from Flask-RESTful. This class-based structure, combined with the *api.add_resource* method to add resources to their respective endpoints, provides an intuitive and clean way to define and manage our API routes. By using Flask-RESTful, it offers a significant edge over alternative approaches especially on greater flexibility and easier error handling.

6.4 System Initialisation and Resilience

On each API call, our custom CouchDB client is initialised. During this initialisation, the client attempts to connect to all the CouchDB servers specified in the configuration. Each server's availability is then verified; only those servers that are operational are selected for potential use. By checking the availability of servers with each API call, we ensure that our system remains resilient, capable of continually operating even if some servers experience downtime.

6.5 Load-Balancing

To enhance our back-end server's efficiency and ensure its scalability, we've adopted an intelligent load-balancing strategy encapsulating in the CouchDB client. This strategy is designed to evenly distribute network traffic across multiple CouchDB server, ensuring that no single server becomes a bottleneck while reducing the risk of server failure. Whenever a data operation needs to be performed, such as fetching data from a partition or retrieving a database, our CouchDB client randomly selects one server from the available pool. This randomised selection ensures that the load is evenly spread across all servers, preventing any single server from being overburdened.

6.6 Error Handling

One unique challenge we encountered for back-end was related to CouchDB's data replication process. When data is replicated from a local server to a remote server in CouchDB, an anomaly could occur where the database name must be changed slightly (such as additional characters being added) to bypass an unsolvable CouchDB error. To tackle this, we've implemented a mechanism in our custom CouchDB client to find the best match for a given database name. When trying to access a database on a particular server, our client searches for the database with a name that most closely matches the intended name. If a match is found, the client proceeds with this database; if not, it raises an error indicating that the database is unknown. This measure enhances our system's resilience by preventing potential mismatches in database names from disrupting the service. In addition to this, our client also includes error handling at various other stages, such as during server connection, partition finding, and database access.

6.7 Back-end Structure

- `back-end` .... The Main back-end Directory
  - `app` .... Main Flask Application Directory
    - `api` .... Holds the API Resources
      - `__init__.py` .... Initialises the API Module
      - `boardPolitical.py` .... Resources for the Political Dashboard
      - `electorateData.py` .... Resources for Handling Electorate Data
      - `topTweets.py` .... Resources for Retrieving Top Tweets
      - `tweetsMeta.py` .... Resources for Tweets Metadata
      - `...` .... Additional Resources
    - `utils` .... Utility Files and Helpers
      - `cors.py` .... Cross-Origin Resource Sharing (CORS) Setup
      - `couchdb_client.py` .... Client Setup for Interacting with CouchDB
    - `__init__.py` .... Initialises the Flask Application
    - `app.config.py` .... Configuration Settings for the Flask Application
  - `docker-compose.yml` .... Docker Compose File for Deployment
  - `Dockerfile` .... Configuration File for Docker Deployment
  - `guincorn_config.py` .... Configuration Settings for Guincorn
  - `main.py` .... The Entry Point for the Flask Application
  - `README.md` .... Documentation and Instructions for the back-end
  - `requirements.txt` .... Lists Python Dependencies Required by the App

7 Front-end

JavaScript was chosen as the front-end language due to its ability to facilitate the swift creation of a prototype, effectively showcasing our development progress. Additionally, the language's flexibility greatly aids in collaborative efforts among team members, making it the preferred choice.

7.1 Components

7.1.1 Vue JS

Vue JS, commonly referred to as Vue, is an open-source, progressive JavaScript framework created as an alternative to jQuery. The decision to adopt Vue is driven by the following reasons.

Firstly, Vue provides a progressive and component-based development approach, which allows for seamless integration into projects based on specific requirements without extensive learning or refactoring. This grants us the flexibility to selectively leverage Vue's features and combine it with other frameworks or libraries, while also improving developers' understanding and maintainability of the codebase.

Secondly, Vue's implementation of two-way data binding, which automatically updates the view when data changes, is particularly advantageous for our project. Since our project involves on-the-fly calculations, this feature ensures that any changes in the data will be promptly reflected in the corresponding view, providing real-time and dynamic updates.

Thirdly, Vue leverages a virtual Document Object Model(DOM) mechanism and optimised update algorithms, resulting in exceptional performance. This advantage plays a crucial role in our website development, especially when dealing with large data sets used to generate heatmaps. It enables us to accelerate the rendering of dynamic content and deliver a smooth user experience, ensuring efficient processing and visualisation.

An additional advantage of Vue is its vibrant ecosystem and robust community support, providing us with an array of valuable resources and tools. Regardless of the specific requirements, such as routing management, state management, UI component libraries, or build tools, Vue's ecosystem offers suitable solutions.

7.1.2 Element UI

Element UI is a UI component library that is built on Vue.js, offering a comprehensive collection of reusable UI components and styles. This library significantly speeds up the development process of applications by providing a wide range of pre-built components. With its extensive component library and responsive design, Element UI simplifies the creation of complex elements on our web application, such as data tables, form pages, pop-ups, and navigation menus.

In the *ELEFEEL* web application, a significant portion of the interactive components are built using Element UI as the foundation, including form pages, pop-ups, navigation menu, selection controls etc. By leveraging Element UI, we can ensure consistent and intuitive user experiences while efficiently creating and integrating these interactive elements into the *ELEFEEL* web application.

7.1.3 ECharts

Within our application's Dashboard page, we employ ECharts, an open-source data visualisation charting library developed by Baidu, to implement dynamic and interactive charts. ECharts is a robust and versatile library that offers a wide range of interactive chart displays, empowering users to effectively comprehend and analyse data. With its customisable and extensible nature, we can tailor and expand ECharts to meet the specific requirements of our project, enabling us to achieve various impactful dashboard visualisations to provide insight of different scenarios.

7.1.4 Google Map API

For the Map page of the *ELEFEEL* web application, we have chosen to utilise the Google Maps API for map rendering. Google Maps API is a platform developed by Google that enables developers to integrate Google Maps into their websites seamlessly. Another alternative considered was Mapbox, which also offers map rendering capabilities. However, Mapbox's map data and API services are not as extensive as those provided by Google Maps. More importantly, Mapbox lacks direct data retrieval and real-time updating capabilities, which are essential for our project's requirements. As our application involves retrieving real-time data from the back-end and dynamically displaying it on the map, the Google Maps API proves to be the optimal choice in terms of map rendering tools. Its rich features, comprehensive data, and real-time updating capabilities align perfectly with our project's needs.

7.2 Front-end Structure

The structure of front-end is as follows:

```
frontend ........................................................ Root
  └ [node_modules] ..................................... Dependency Packages
  └ public ................................................ Public Resources
  └ Dockerfile ........................... Configuration File for Docker Deployment
  └ src .................................................... Source Code
     └ assets .............................................. Static Files
     └ layout ................................ Reusable Vue Layout Components
     └ views ............................................. Routing Pages
     └ routers ................................... Route Configuration Files
     └ store ..................................... State Management Store
     └ utils .................................... Tool Functions and Constants
     └ App.vue .............................. Root Component of the Application
     └ main.js ................................................ Entry File
  └ docker-compose.yml ...................... Docker Compose File for Deployment
  └ .env ............................................ Environment Variables
  └ nginx.conf ................................ Nginx Server Configuration File
  └ vue.config.js .................... Vue Project Build and Configuration Settings
  └ Package.json ...................... Node Project Dependencies and Configuration
  └ Package-lock.json .................... Locked Versions of Project Dependencies
  └ README.md ....................... Front-end Project Documentation and Instructions
```

7.3 Challenges

We encountered several challenges during front-end development, but through collaborative problem-solving and continuous improvement, we were able to overcome them.

Firstly, we faced an issue with the navigation bar and sidebar highlighting on the dashboard page, particularly when navigating to the Ranking, Distribution, and Comparison sub-pages. The highlighting did not accurately correspond to the displayed content after refreshing the page. To address this, we introduced a new method called "activeMenu" within the computed life-cycle of the navbar and sidebar's settings files. This method retrieves the current page path and modifies it to return the appropriate parent path. By utilising *computed* property in Vue components, we were able to perform relevant operations and rendering logic based on the current route's path, ensuring the correct highlighting of the navigation bar and sidebar.

The second challenge we encountered was the initial rendering of the Map. We wanted to put together static data such as coordinates and computed data such as sentiment for each electorate, and represent the sentiment values using a heatmap. To solve this, we consulted Google's documentation and found that we could first load in the static data in *geojson* format using *map.data.loadGeoJson()*, return the calculated values through another API and convert them to hexadecimal colors, and set their colors using the *map.data.setStyle()*, so that the heatmap is displayed.

7.4 Errors Handling

One error we encountered was updating data when changing filter options using the *onclick* event listener in Vue.js. Initially, the data was not being updated upon modifying the filters. To solve this error, we implemented a solution by switching to using the *watch* in Vue. By utilising the *watch*, we were able to monitor the values of the filters. Whenever a change was detected, we would re-request the data and trigger a re-render of the relevant components. This approach allowed us to effectively synchronise the changes in filter options with the corresponding data updates, ensuring accurate and up-to-date information was displayed to the users.

Additionally, we faced a problem in rendering comprehensive information for each electorate on the map. When users clicked on a region in the map page to view its details, not all the information would appear. To overcome this, asynchronous operations were implemented as a solution. By executing all the data request operations asynchronously, the data would be rendered and displayed once all the necessary information became available. This approach guaranteed that all the information would be visible to users once it was retrieved.

8 Scenario Analysis

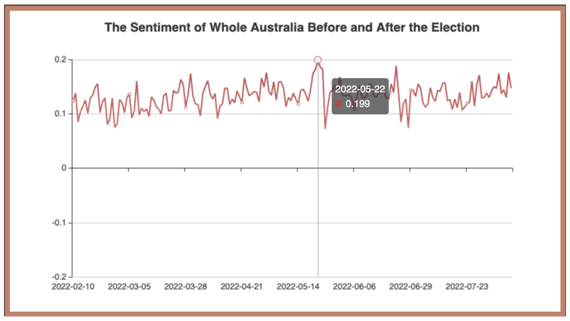
8.1 Social Media Political Activeness in Australia

In an attempt to gauge the level of (English Language) political discussion on social media in Australia compared to international levels, a comparison was made between the 'political tweets' percentage in the Australian tweets against the percentage in Mastadon toots, which are crawled from servers all over the world. As Mastodon only allows harvesting on specific keywords rather than any topic, the percentage of exact match political English language toots in all exact and partially matched political toots are used for comparison with tweets. With this assumption in mind, as well as that the geolocation containment of tweets is not correlated (biased) towards any tweeted topic, we can see that the Toots' 23.7% (at the time of drafting of this report) is higher than Tweets' 12.2% by 9.5%, and thus we can say Australia's political discussion is not that high compared to the international English speaking community.

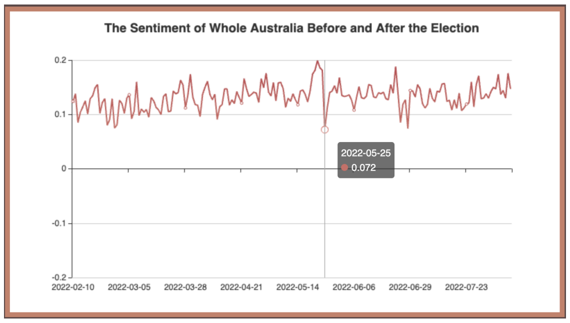
However, it should be noted that 12.2% of tweets being politically related is quite a significant number, given the vast number of topics that one could discuss on social media.

8.2 Sentiment Fluctuations in Political Discourse around Election Times

Another key finding is the fluctuation of political discussion sentiments on Twitter in Australia around the election day, with average sentiment reaching an all time high for the observed period on the day after the election (22nd of May) at 0.199, rising rapidly from the 0.123 on the 18th of May, but then sharply decreasing on the 25th to an all new low of 0.072 (figure 5). A possible explanation is that voters with all different political opinion holds high hope for their favoured Party winning before the election, and the day after the election supporters of ALP were vocal about their positivity on Twitter, hence the unusual levels of optimism before the 22nd; but following the election, supporters of the losing major party (which should be substantial) would be cynical about the future direction of the nation, hence the sudden dip.



(a) 22nd of May



(b) 25th of May

Figure 5: Daily National Political Sentiment on Social Media

Otherwise, there is a general growing trend pre-election as the election date gets closer, and a general decreasing trend after the election when observing monthly aggregations (figure 6). Whilst this is counter-intuitive with respect to the result of the election which saw a change in government, an explanation is that although the general public sentiment felt at the time were against the governing Coalition, the political discussions were optimistic and excited about the ever-nearing opportunity to vote them out; however, finding post-election that change cannot come rapidly and significantly, the sentiment subsequently decreased.

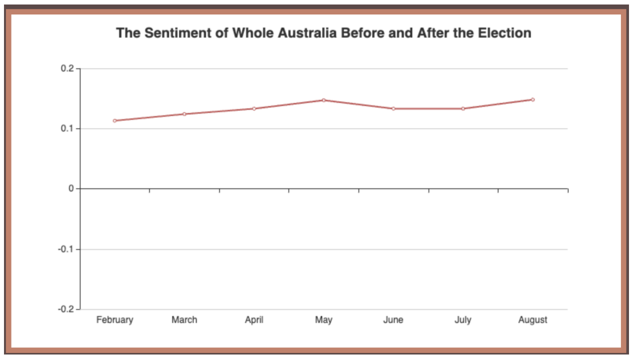


Figure 6: Monthly Aggregated Sentiment with Weak Increasing Trend pre-election and Weak Decreasing Trend post-election

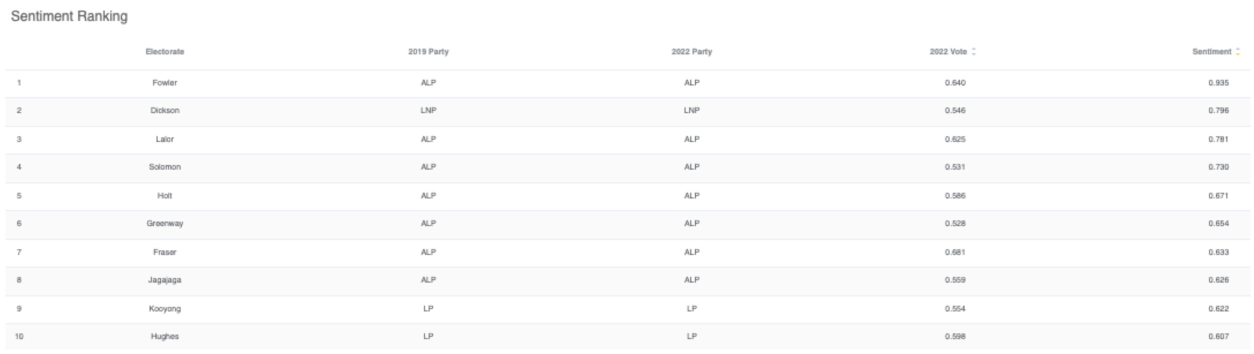
8.3 General Positive Sentiments in Australia

Other findings include that the Australian sentiment when discussing politics is generally positive for the observed period, as seen on the daily aggregated national sentiment line graph which never dipped into negative values over the 6-month period (figure 5). Though the magnitude of the positivity is difficult to interpret as different Sentiment models may use different scales, this observation is derived from the sign of the sentiment score, a statistic which provides a much more straightforward interpretation.

8.4 One-Month Sentiment Ranking

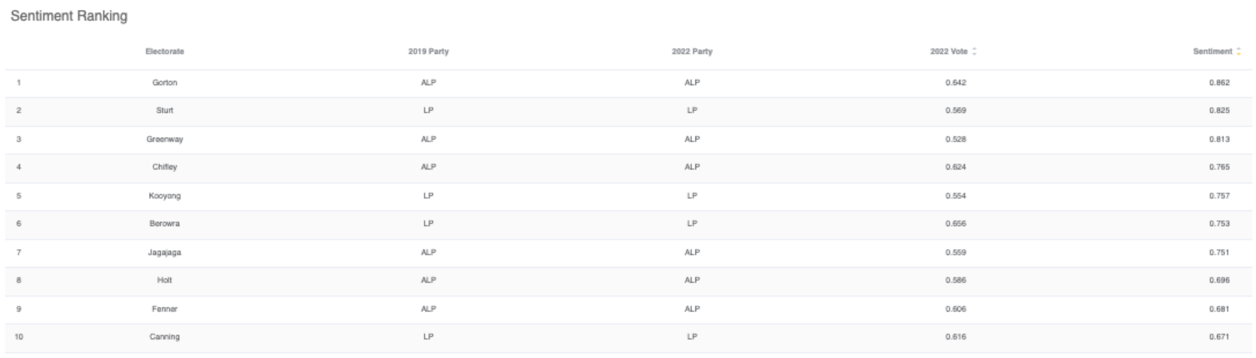
It was observed that for positive sentiment ranking of political discussion in each electorate one month before the election, 7 of the top 10 (in fact 7 out of the top 8) ranking electorates at the time were held by the ALP (the eventual 2022 national election winner) (figure 7a). Meanwhile, one month after the election, both ALP and LP had a pretty equal share up the top of the sentiment ranking (6 ALP in top 10, where LP held the 2nd, 5th and 6th highest sentiment ranking electorates) (figure 7b).

Thus, the one month pre-election rankings could potentially be a crude qualitative indicator of the winner of the national election, though further qualitative analysis must be undertaken with more data on different elections to test this relationship. Meanwhile, the one month post election rankings suggests that people in each electorate are generally happy with the government they just recently elected. Another perhaps more ambitious explanation of this observation is that the positive sentiment in ALP governed electorates were not really due to difference in governance that changed living standards in the electorate, rather that people felt more positive about living in an electorate governed by the ALP because of the widespread discontent towards the Liberals Coalition Federal Government at the time; with this aura disappearing post-election, a truer sentiment is revealed. Thus, an alternative conclusion is that political sentiment is not really affected by which Party governs the electorate. This should be investigated by looking at further previous elections to see if it is a recurrent theme.


Sentiment Ranking

	Electorate	2019 Party	2022 Party	2022 Vote	Sentiment
1	Fowler	ALP	ALP	0.640	0.935
2	Dickson	LNP	LNP	0.546	0.796
3	Lalor	ALP	ALP	0.625	0.781
4	Solomon	ALP	ALP	0.531	0.730
5	Holt	ALP	ALP	0.586	0.671
6	Greenway	ALP	ALP	0.528	0.654
7	Fraser	ALP	ALP	0.681	0.633
8	Jagajaga	ALP	ALP	0.559	0.626
9	Kooyong	LP	LP	0.554	0.622
10	Hughes	LP	LP	0.598	0.607

(a) Pre-Election Top 10 Sentiment Ranking


Sentiment Ranking

	Electorate	2019 Party	2022 Party	2022 Vote	Sentiment
1	Gorton	ALP	ALP	0.642	0.862
2	Sturt	LP	LP	0.569	0.825
3	Greenway	ALP	ALP	0.528	0.813
4	Chifley	ALP	ALP	0.624	0.765
5	Kooyong	LP	LP	0.554	0.757
6	Berowra	LP	LP	0.656	0.753
7	Jagajaga	ALP	ALP	0.559	0.751
8	Holt	ALP	ALP	0.586	0.696
9	Fenner	ALP	ALP	0.606	0.681
10	Canning	LP	LP	0.616	0.671

(b) Post-Election Top 10 Sentiment Ranking

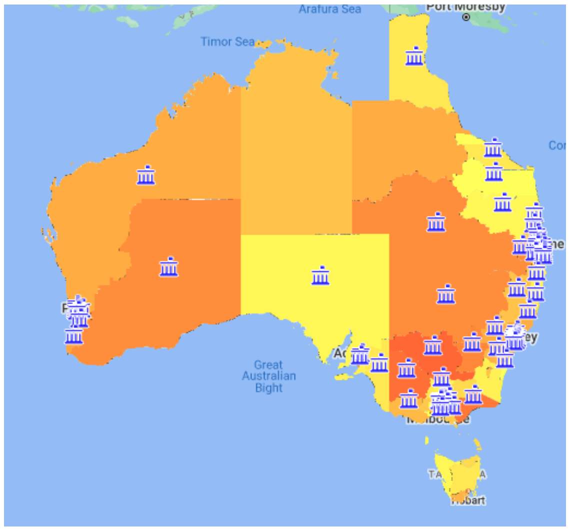
Figure 7: Electorate Ranking by Sentiment

8.5 Distribution of Electorates Won in the 2019 and 2022 Election

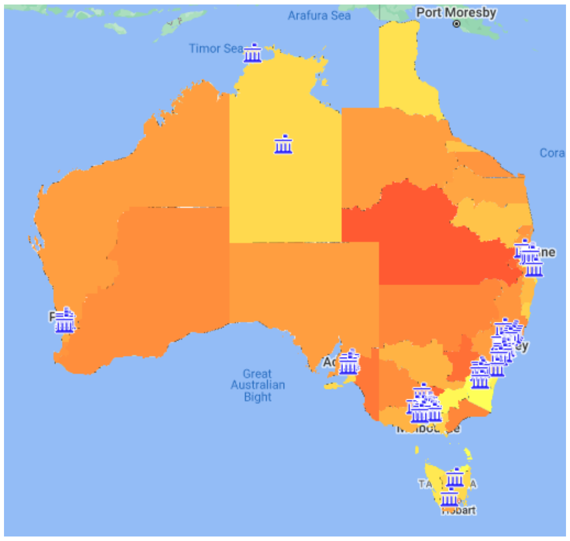
In the 2022 election, most of the electorates won by ALP (the national election winning Party) was in the Greater Capital City areas (figure 8b), whilst in 2019 the Liberal Nationals Coalition had a balanced share of rural and metropolitan figure (8a) - perhaps due to the National Party's rural roots.

8.6 Comparison of Post-Election sentiments of Electorates that changed parties

It was observed that in the two weeks (figure 9a) after election, the electorates that changed governing Parties increased in sentiment to achieve a higher mean sentiment compared to electorates without change of governing Party. However, as time continued (1 month, 2 months, 3 months post election (figure 9b])), the average sentiment of this group of electorates toppled to be significantly lower than that of the electorates which didn't change governing Party. A possible explanation is that the people of this electorate were dissatisfied with the pre-election governing Party (hence the change in Party), and felt initially positive about voting in a new Party to govern their electorate. However, as time went on they felt that the new Party didn't bring substantial change to their lives as political problems that relate to lifestyle such as public infrastructure takes months if not years to solve, hence the dip at the end.

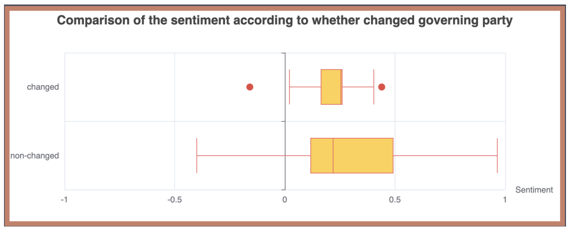


(a) Pre-Election National Governing Party (LP/NP/LNP) held electorates distribution

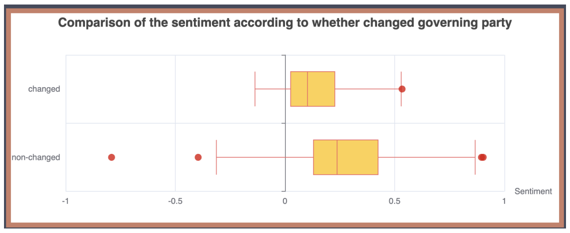


(b) Post-Election National Governing Party (ALP) held electorates distribution

Figure 8: Distribution of Governed Electorates of National Governing Parties



(a) 2 weeks post election



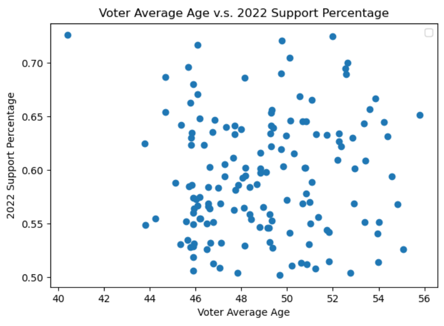
(b) 3 months post election

Figure 9: Box and Whisker plot of political sentiment of electorates that changed vs did not change governing Party

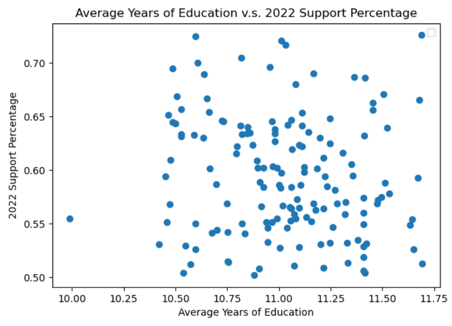
8.7 Census Data vs Sentiment Data

Observing the scatter plots of Census Data attributes (i.e. *average voter age*, *average years of education*, *average weekly household income*, *migration*) and sentiment as well as percentage of votes won, no trend is observed between any of the census and election data (figure 10). However, there seem to be some trend between *average voter age*, *average years of education*, *average household income* and sentiment, and further analysis demonstrated that the weak relationship seems most reasonable in *log(census data)* v sentiment form (figure 11).

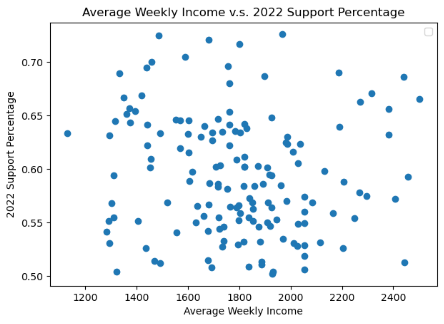
As observed, *log(average voter age)* has negative correlation to sentiment, whilst *log(average years of education)* and *log(average weekly household income)* has positive relations. If in future Linear Regression is used to rigorously analyse relationships between sentiment and election results, *log(average voter age)*, *log(average years of education)* and *log(average weekly household income)* should be included in the regression as control variables as they are correlated to the explanatory variable and hence could cause omitted variable bias.



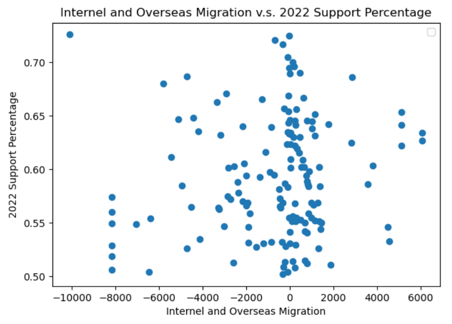
(a) Age vs 2022 Election



(b) Education vs 2022 Election

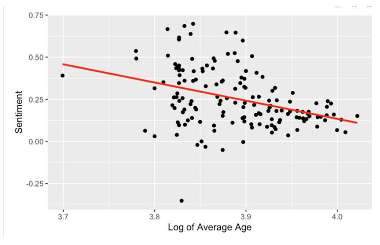


(c) Income vs 2022 Election

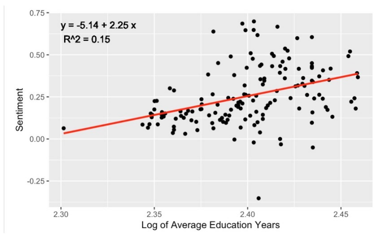


(d) Migration vs 2022 Election

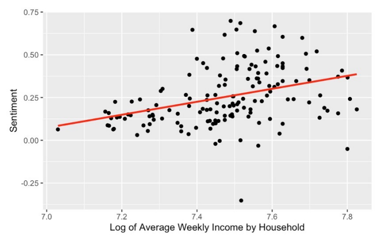
Figure 10: Relationship between Census Data and 2022 Election Percentage votes won



(a) Log Age vs Sentiment



(b) Log Education vs Sentiment



(c) Log Income vs Sentiment

Figure 11: Relationship between Log(Census Data) and Sentiment

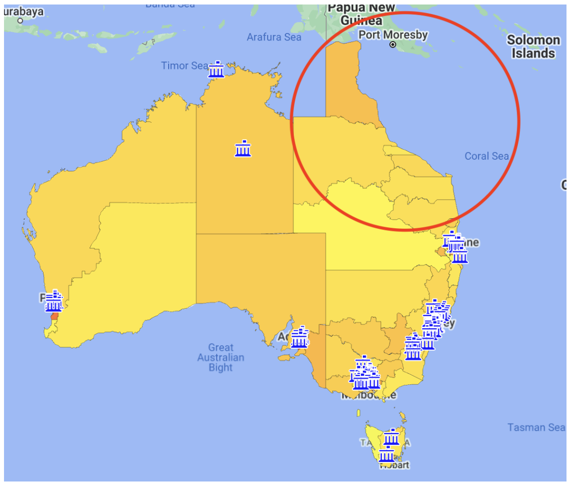
9 Limitations and Future Improvements

9.1 Limitations on Dataset

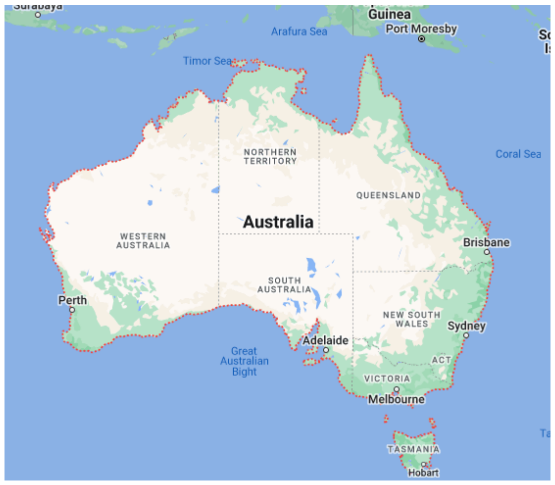
9.1.1 Elections Data

Shapefile Error

When using the shapefile to plot on the Map of Australia using the WGS84 standard (World Geodetic System 1984), the electorate boundaries of several coastal Queensland electorates expanded severely beyond the landmass. Researching solutions to the problem on the AEC website and other forums did not provide an answer, so this problem was left unsolved as it is mainly an aesthetics problem on the application. It should be noted however that several Queensland electorates social media data aggregates may be inaccurate as a result of the potentially incorrect boundaries provided by the AEC 12.



(a) Australian Map from AEC shapefile



(b) Australian Map with Correct Boundaries

Figure 12: Problem of Incorrect Queensland Coastal Borders in AEC Shapefile

Incomplete Election Data

Another data problem was that the party with the most TCP votes did not fully match the official winning parties in each electorates according to AEC (Commission [a]). This is because Australia's voting system is different from the intuitive "first past the post" - where the Party whose candidate has the most first preference wins - but rather uses preferential voting, as explained in (Commission [b]). In short, each voter not only puts down the first vote but also orders their votes on candidates from 1 to n (for n candidates in their electorate). If no candidate can get more than half of the first count votes in the electorate, then a second round of counting occurs where the candidate with the least first preference votes are removed from the race, and the votes originally amassed by them are converted as "first preference votes" to other candidates according to each voter's second preference. If still no candidate has more than 50% (pseudo-)first preference votes, then the current candidate with lowest (pseudo-)first preference votes is removed and the process is repeated. Hence, a winner may be determined before the final round when only two candidates are left, and so the candidate with most TCP votes may not have actually "won" the electorate. As the AEC website only offers the First Preference votes and TCP (i.e. the “first round” and “last round” statistics, it is not possible to recover the *%first preference votes* at the round that the race for this electorate was decided, nor know whether an electorate where the winner according to TCP data indeed matched the official winner actually won their race in the TCP (last) round. Hence, to keep the percentage of votes won by the winning party consistent, it was decided to analysis this stat based on the TCP round percentage of majority votes won, as it was the most accessible and reliable source of data.

From an analysis perspective, this is also the best source of reflection of the people’s preferences, compared to previous rounds with more than two candidates where votes are more diluted. At the very least, even if the correct terminating round’s statistics were obtainable, data that ended in different rounds are not comparable due to the mechanism of preferential voting. Thus, the best choice was made in spite of this limitation, and users should simply keep it in mind when interpreting observations.

9.1.2 Census Data

Estimation Errors in de-Aggregation and Proportionation

Using two estimation algorithms back-to-back to recover estimated aggregated statistics on an Electorate level will lead to potentially large inaccuracies. In particular, the proportionation which only takes into account landmass overlap without considering real geographical distribution of population (e.g. via house locations) leads to biases and inaccurate representations. A better algorithm should use the number of houses in the overlapped areas to help determine overlapping percentages that more accurately represent the distribution of population, instead of just the overlap of geographical areas.

Data Staleness

Another (tho much smaller) limitation of the data is its staleness (some collected in 2016) compared to the 2022 election for which relations are attempted to be mined - a 6 year outdate. The analysis would be much more valuable using newer data (i.e. 2021 census data which is currently not fully and freely available).

9.1.3 Twitter

Rule-based Sentiment Analyser and Sarcasm

The inability of the compound score to detect sarcasm is another limitation, as it computes sentence scores on a word-to-word basis, and would be insensitive to capturing semantic meaning immersed in word ordering. Neural methods such as Long Short Term Memory models which are not as computationally expensive as BERT should be attempted in future for better performance.

9.2 Limitations on Database

Custom Reduce Functions Infeasible

One significant disadvantage of CouchDB is that manual reduce functions are extremely slow when API to views are requested, as they are implemented in JavaScript - an interpreted language, compared to default reduce functions that are implemented in the (much faster) compiled programming language *C*. Though the default *"_sum"*, *"_stats"*, *"_count"* are powerful, this limits the ability to perform more advanced aggregations on the database to leverage its efficiency using just views alone.

MapReduce Inability to Perform Advanced Aggregations

Another significant disadvantage of CouchDB is its inability to group over just the later values of the MapReduced documents' keys. For example, performing a second *groupby* over electorates in the *Average political sentiment by day* view is impossible because the group function must also involve prior key values (in this case dates) - a result of the underlying B-tree structure that the view has. This again forces back-end to perform remaining aggregations, which are suitable for a problem which only has 151 electorates and 181 days, but less so for longer time periods and problems that involve i.e. sals for which Australia has more than 60,000.

Indexes and Views Slow to Build

On our moderate database of 2.5 million documents, the time between when the view/index is created and complete (useable) often take 1+ hours. Whilst this may be due to the restriction of 2 vCPUs per node, as the process is much faster on local devices with much more computing power, it is not a challenge once these design documents are completely created, with quick responses for views and queries provided the right indexes are present. However, this may become a problem if database observes a high volume of data entry, as each design document is recalculated at the next request based on the updated documents since the last call, under such circumstances which more computing resources should be allocated to the CouchDB nodes.

Docker Swarm Cluster Configuration

Docker Swarm was utilised in the creation of the cluster, however, although we ensured all the ports are open and CouchDB was set up in the instances and UI of each CouchDB shows that the cluster is set up, when we asked for`_membership` in the instances, under `all_nodes` only the respective IP addresses of the CouchDB instances are included. This took a week to debug but nothing seem to work, hence, using the Docker Swarm approach was abandoned.

9.3 Deployment issues on MRC

When setting up instances, although a "wait for instances connection" was set, the instances sometimes will not allow ssh connections even all ports are opened. This led to removal of the instance and redeployed. Also, the Docker Swarm cluster setup led to instances crashes for which we had no idea why, thus again leading to the removal of the instance and redeployed; this became another reason of the abandon of the Docker Swarm approach.

9.4 Limitations on Front-end

When developing the sentiment chart that changes over time on the dashboard page, the original intention was to include an additional vertical line representing the election day, specifically on May 21st, as shown in Figure 13. This line would serve as a visual indicator, making it easier for users to observe the sentiment change before and after the election. However, due to the limitations of the chosen charting library, ECharts, it was challenging to incorporate conditional logic directly into the provided options. This limitation highlights the constraints associated with using pre-built libraries. While these libraries adhere to standards and offer customisation documentation, certain functionalities may not be easily achievable or the particular chart that we want may not be available.

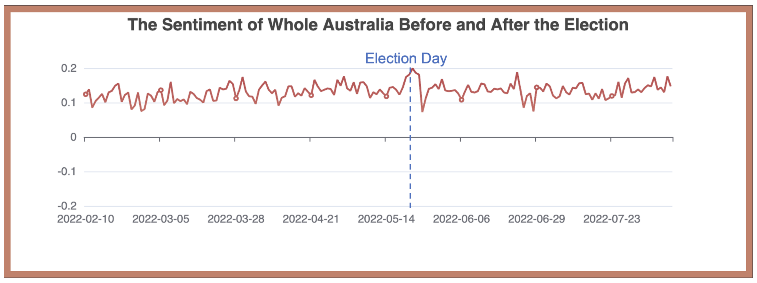


Figure 13: Ideal sentiment chart with an additional blue vertical line at Election Day, 21st May

10 Development Process

10.1 Division of Responsibilities

Our team worked in a manner such that most key components of the project had one member primarily responsible. The team followed the Agile development method with sprint length of 3-4 days, due to the complexity of the project under a tight 7 week deadline. An Agile Ceremony was held weekly to discuss progress and set new sprint goals and a standup every three days, with cross-component correspondence meetings convened when necessary between members involved (i.e. Data and Front-end meetings to settle design, Front-end and Back-end meetings, Database and Back-end meetings etc).

The team also held non-technical roles such as Scrum master to support the efficient functioning of the team.

Team Member	Technical Component Responsible	non-Technical roles
Chenyang Dong	Back-end and assistance to Deployment	Git Manager
Lang Chen	Database and Data	Scrum Master, Minute Taker
Un Leng Kam	Deployment and Tester	Documentation Manager
Ying Zhu	Data and assistance to Front-end	External Communications
Zhuoya Zhou	Front-end	Google Drive Data Manager

Table 7: Team Responsibility

10.2 Technologies and Techniques Used

Code collaboration and version control were achieved through GitHub, with each feature branching out from development branch, which when tested to be bug-free is merged into the main branch. Large data files were transferred using a shared Google Drive to prevent slow GitHub push and pulls due to clogging.

All meetings except for ceremonies were held virtually on Zoom, and regular communications made over a group chat or direct messaging on a social media app previously familiar to all team members.

Cross-component meetings

Cross-component meetings were an important technique employed to increase team efficiency, where related group members attended short intense meetings to get confirm their understanding of what needs to be done, and settle on decisions that jointly affect their component's work.

Peer Programming

Peer programming was also employed as a technique to prevent errors at time of coding. Although it seems expensive to expend two people's time for one piece of work, it mitigates the more time consuming efforts to return to the work later for debugging, as well as other delays due to errors on critical parts of the development pipeline.

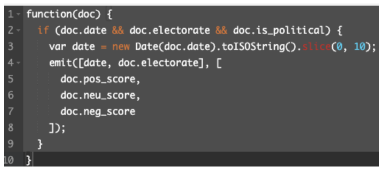
11 Appendix

Algorithm 1 Proportioning Statistics from LGA to Electorates

```
1: $n \leftarrow$ number of LGAs in Australia
2: for $i \leftarrow 1$ to number of electorates do
3:     $StatElectorate_i \leftarrow 0$
4:     for $j \leftarrow 1$ to $n$ do
5:         $AreaOverlap \leftarrow LGA_j AreaOverlapElectorate_i / AreaElectorate_i$
6:         $StatLGA_j \leftarrow$ statistic value of LGA $j$
7:         $StatElectorate_i \leftarrow StatElectorate_i + (AreaOverlap \times StatLGA_j)$
8:     end for
9: end for
```

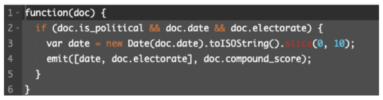
Algorithm 2 Assign electorates to Tweets via coordinates

```
1: function GET_ELECTORATE_FOR_TWEET(tweet)
2:     coord $\leftarrow$ Point(tweet["$long$"], tweet["$lat$"])
3:     for each polygon in polygons do
4:         if polygon.geometry.contains(coord) then
5:             return polygon['Electorate_Name']
6:         end if
7:     end for
8:     return None
9: end function
```



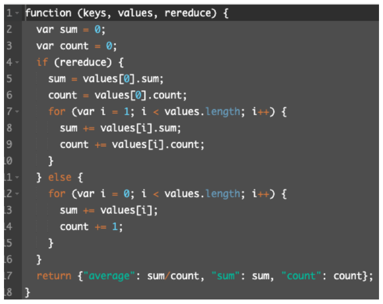
```
function(doc) {
  if (doc.date && doc.electorate && doc.is_political) {
    var date = new Date(doc.date).toISOString().slice(0, 10);
    emit([date, doc.electorate], [
      doc.pos_score,
      doc.neu_score,
      doc.neg_score
    ]);
  }
}
```

(a) Map function for *(Pseudo-)number of political tweets with each sentiment tag by electorate day*



```
function(doc) {
  if (doc.is_political && doc.date && doc.electorate) {
    var date = new Date(doc.date).toISOString().slice(0, 10);
    emit([date, doc.electorate], doc.compound_score);
  }
}
```

(b) Map function for *Average political sentiment by day returns*



```
function (keys, values, rereduce) {
  var sum = 0;
  var count = 0;
  if (rereduce) {
    sum = values[0].sum;
    count = values[0].count;
    for (var i = 1; i < values.length; i++) {
      sum += values[i].sum;
      count += values[i].count;
    }
  } else {
    for (var i = 0; i < values.length; i++) {
      sum += values[i];
      count += 1;
    }
  }
  return {"average": sum/count, "sum": sum, "count": count};
}
```

(c) Reduce function for *Number of political tweets*

Figure 14: Relationship between Log(Census Data) and Sentiment

References

Australian Electoral Commission. When elections are held, 2023a. URL https://www.aec.gov.au/learn/election-timetable.htm. Accessed: 2023-04-25.

Rob Pietsch. There's gen z. then there's gen z on twitter., 2021. URL https://marketing.twitter.com/en/insights/gen-z-twitter-trends. Accessed: 2023-05-10.

TheUniversityOfMelbourne. Melbourne research cloud documentation. https://docs.cloud.unimelb.edu.au/.

Australian Electoral Commission. Australian electoral commission, 2023b. URL https://www.aec.gov.au/. Accessed: 2023-04-25.

Australian Electoral Commission. 2019 federal election downloads and statistics, 2019. URL https://www.aec.gov.au/Elections/federal_elections/2019/downloads.htm#:~:text=Historical%20two%20candidate%20preferred%20(TCP)%20votes%20by%20party%20by%20division%20%5BCSV%2017KB%5D. Accessed: 2023-04-25.

Australian Electoral Commission. 2022 federal election downloads and statistics, 2022a. URL https://www.aec.gov.au/Elections/federal_elections/2022/downloads.htm#:~:text=Historical%20first%20preference%20votes%20by%20party%20by%20division%20%5BCSV%2051KB%5D. Accessed: 2023-04-25.

Australian Electoral Commission. 2022 electoral boundaries, 2022b. URL https://www.aec.gov.au/electorates/gis/gis_datadownload.htm#:~:text=ESRI%20(.shp)%20%5BZIP%2020.68MB%5D. Accessed: 2023-04-25.

Melbourne e Research Group. Spatial urban data observatory, 2018. URL https://sudo.eresearch.unimelb.edu.au/. Accessed: 2023-04-25.

NLTK. Sentimentintensityanalyser, 2016. URL https://www.nltk.org/howto/sentiment.html. Accessed: 2023-04-25.

Rodney Smith, Ariadne Vromen, and Ian Cook. Keywords in australian politics. *Keywords in Australian Politics*, pages 15–18, 2006. Accessed: 2023-04-25.

Australian Electoral Commission. House of represenatives - final result, a. URL https://results.aec.gov.au/27966/Website/HouseDefault-27966.htm. Accessed: 2023-04-25.

Autralian Electoral Commission. Preferential voting, b. URL https://www.aec.gov.au/learn/preferential-voting.htm. Accessed: 2023-05-15.