COMP90024 Assignment 2:

Australia Social Media Analytics

AOBO LI	Pavith Samarakoon	Zhihao Liang	Jiqiang Chen
1172339	1297058	1367102	1171420

Yijun Liu

1132416

Contents

1 Introduction

In 2023, social media has grown significantly and plays a critical role in shaping society. In this project, we aim to conduct an in-depth research of three interesting topics on social media in Australia. In order to analysis the large amount of social media data, this project leverage the resouce on the Unimelb Research Cloud which offers free on-demand computing resources to researchers at the University of Melbourne.

1.1 Melbourne Research Cloud

Melbourne Research Cloud(MRC) operates based on an open-source software OpenStack, which is an OpenStack private cloud that offers a subset of the services OpenStack offers, providing a platform for University of Melbourne's researchers for cloud computing. It provides similar functionality to commercial cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. This section discusses advantage and disadvantage of MRC.

Advantages:

- MRC is a free Cloud services, no cost to access the MRC for us unlike other cloud services such as AWS and Azure
- MRC has much more computational resources compare to our PC
- MRC is easy to facilitate collaboration as it is cloud-based
- MRC has high flexibility to set up the server environment
- MRC is prone to accident mistake as it has snapshot mechanism that enables recovery

Disadvantages:

- MRC has a limited resource per research group, unlike commercial IAAS services that are

highly scalable

- MRC may become unavailable as some group's servers are down
- MRC security group mechanism require VPN support which may incur problems in different local environments

2 Design and Architecture

2.1 High-Level System Architecture

Our cloud-based data analysis system is deployed on four instances within the Melbourne Research Cloud (MRC). These instances consist of one master node and three slave instances. The master node is tasked with setting up the slave instances, deploying frontend and backend applications, initializing a mastodon crawler, and setting up HAproxy as a reverse proxy.

The other three slave instances play a critical role in deploying a CouchDB cluster. Each of these instances hosts a Docker container that in turn hosts a CouchDB. To facilitate the process of development and continuous improvement, we have chosen GitHub as a platform for sharing our development progress. GitHub action is our choice for continuous integration and continuous deployment (CI/CD). The process is triggered when a push request is made. GitHub action proceeds to create a virtual instance, then accesses the master instance to execute a shell script. This script contains commands for redeploying the project.

Ansible is our chosen tool for deploying the CouchDB cluster. The process begins by installing dependencies, including OpenStack, in the master instance. Once this is completed, instances are created with specific properties such as volume, model, and the operating system in use. Following the creation of the slave instances, SSH keys are set up, allowing Ansible to access the new instances. Next, we install required dependencies in the slave instances, then set up a CouchDB container for each one. One of the slave instances is chosen to establish a connection with the other

two, thus forming the CouchDB cluster.

To balance the load among the slave instances, we set up HAproxy as a reverse proxy in the master instance. The overall architecture of our cloud system is illustrated in figure 1.

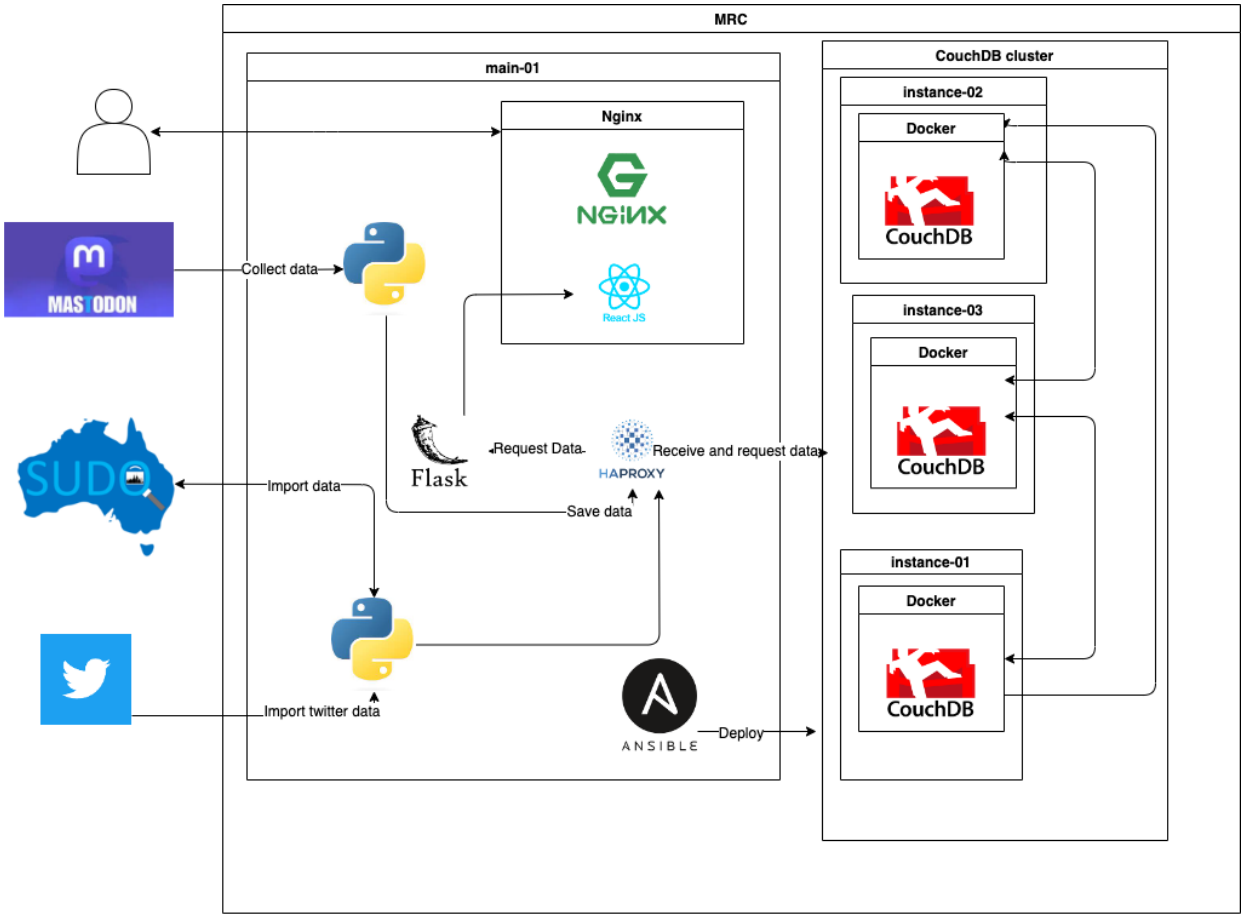


Figure 1: Diagram of the Architecture

2.2 Individual Components

2.2.1 Docker Containers

Docker is essential for deploying the CouchDB cluster in our setup. It helps us to isolate each CouchDB in its container, offering both scalability and isolation.

2.2.2 Ansible Playbooks

Ansible playbooks are used to automate the deployment and orchestration of our system. They manage the installation of dependencies on the local host, configure SSH keys, install required dependencies in the slave instances, and set up the CouchDB cluster.

2.2.3 CouchDB Database Structure

Our CouchDB cluster offers a strong and distributed data storage solution. The cluster is made up of three nodes, situated on separate server instances. These instances have the following IP addresses:

```
couchdb@172.26.128.91
couchdb@172.26.129.201
couchdb@172.26.133.51
```

This distribution ensures that data is resilient and highly available. We've further strengthened our system's resilience by using a HAProxy as a reverse proxy to efficiently manage network traffic to and from these nodes.

Each node in the cluster has a large storage capacity of 135 GiB, enabling us to handle vast amounts of data effectively.

We've configured our cluster with specific parameters to maintain data integrity and ensure robust data distribution. For instance, we've set the number of shards ('q') to 2. Sharding is a method that allows us to distribute data across multiple machines. In addition, we've set the number of replicas ('n') to 3. Replication enhances data availability by maintaining multiple copies of every document across the nodes.

2.2.4 GitHub repository

We use GitHub as a version control system and for collaborative development. It acts as the central repository for our codebase, allowing developers to work together efficiently and keep track of all changes. The repository also integrates with GitHub Actions for seamless CI/CD pipelines.

For the deployment of our instances, we follow specific configurations for compatibility with the Melbourne Research Cloud:

```
Availability_zone: melbourne-qh2-uom
Instance_network: qh2-uom-internal
Instance_flavor: uom.mse.2c9g
```

By adhering to these configurations, we ensure optimal performance and resource allocation for our components. Figure 2 below is the dashboard of MRC after deployment.

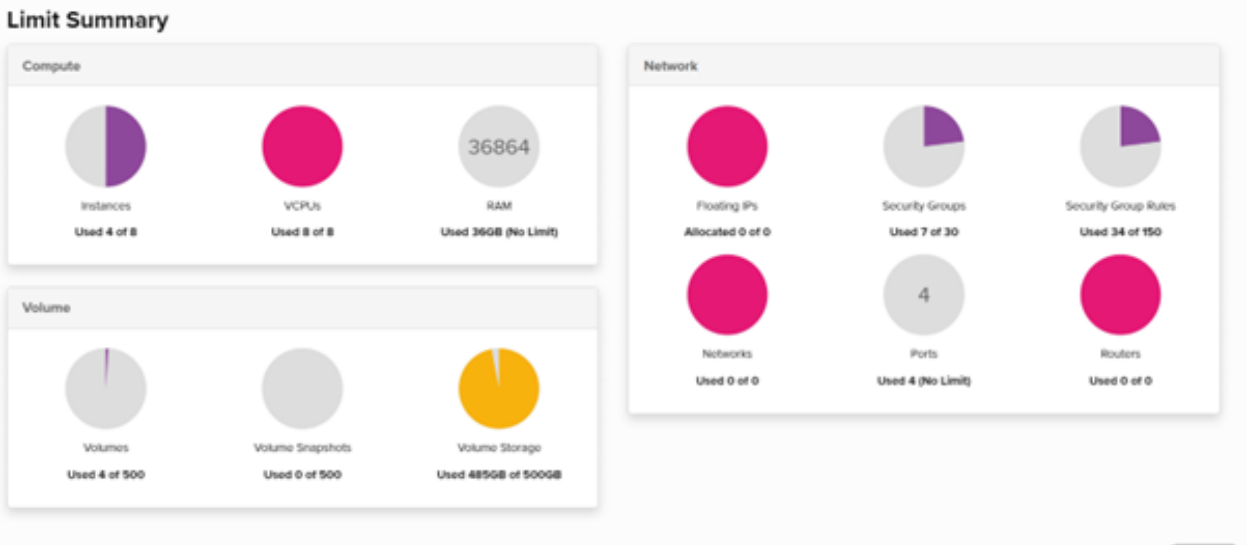


Figure 2: dashboard of MRC

3 Scenarios Introduction

In our analysis of Australia social media, our research directions are split into three different scenarios. For each scenarios, we used information collected from both Twitter and Mastodon, combine with the SUDO data to carry out a deep analysis. This section contains a brief description of the three scenarios and the reason why we choose them.

3.1 Scenarios 1 - Russian and Ukraine War

On February 24, 2022, Russia invaded Ukraine provoking the most serious military conflict in Central Europe since 1945. [1]The impact of this crisis extends to the global economy and politics. For our analysis, we aim to extract information from Australian social media. Our goal is not only to determine the general public's thoughts on this war but also to understand how opinions vary among different individuals. In order to obtain a well-rounded result, we conducted the study on several different dimensions, including gender, location, political views, and more.

3.2 Scenarios 2 - LGBT community

LGBT or LGBTQIA+ is an abbreviation for lesbian, gay, bisexual, transgender, queer or questioning, intersex, asexual, and more. [2]These terms are used to describe a person's sexual orientation or gender identity. The LGBT community has been growing rapidly in recent years, and more and more Australians are becoming aware of this community. As Australia claims to be an inclusive country, we are attempting to gather different opinions from people and analyze social media data to determine how inclusive Australians are towards the LGBT community.

3.3 Scenario 3: General Analysis of Australian Tweets and Mastodon toots

Topic likes Scenario 1 and 2 are interesting but we also want to conduct research on social media that are more general. So for the last scenario, we zoom out the analysis from a specific topic to all Australian social media users. We aim to find the habit of the general users on both Twitter and

Mastodon platform, also trying to investigate if the habit varies among different individuals.

4 Data Pre-processing/Collection

4.1 Twitter Preprocessing

We will describe how we process the vast amount of data from Twitter and extract relevant details for our analysis in this section. For each tweet, we want to extract scenario information, location information and other general information such as times or sentiment. Scenario and proper location information are not store in the Twitter data, so we have to develop methods to obtain this information. Figure 3 is a flowchart that demonstrates our preprocessing steps.

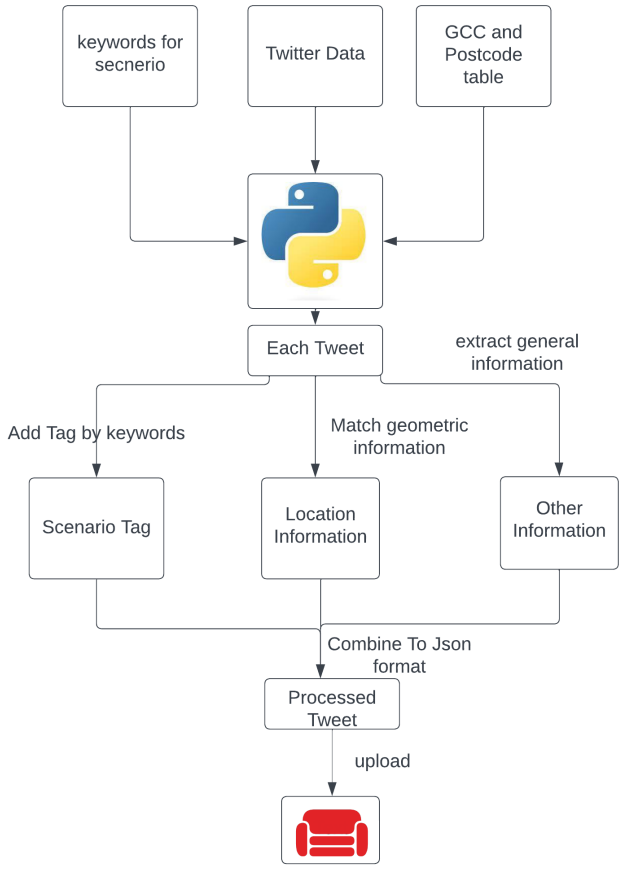


Figure 3: Diagram of the preproccess

4.1.1 Scenarios Detection

We purposed a keyword detection method to finding the relevant scenarios for our analysis. A list of keywords is created by our prior knowledge to the topic, then we will match the tweet text that contains the keywords. We also tested the algorithm on a subset of the Twitter dataset to validate its effectiveness. This helped us discover additional potential keywords for matching. We discovered that to identify tweets related to the LGBT community, using a single keyword from our list provided relatively accurate results. However, to identify tweets related to the Russian war, we found that two or more keywords from our list were necessary to achieve an accurate match.

This scenarios is more efficiency focused rather than accuracy. However for the matching tweets, the result is relative accuracy, there are only few unrelated tweets.

4.1.2 Location matching

For our analysis purpose, we needed to extract the state, Greater Capital City (GCC) and postcode from the location information of the tweet. A matching algorithm is purposed based on two tables: GCC table and POSTCODE TABLE. We also stored a dictionary of high-frequency suburbs for direct matching to increase efficiency. Below is a diagram of our location-matching algorithm.

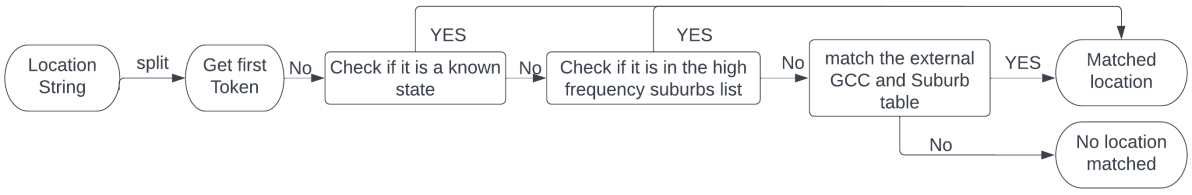


Figure 4: Diagram of the location matching

4.2 Mastodon streaming

Mastodon [3], a popular decentralized social media platform, is used to provide post comparison with Twitter data. The social media platform offers Representational State Transfer (REST) API

that enables developers to access all Mastodon posts, and by utilizing HTTP requests, toots can be streamed live from the Mastodon server using Mastodon's python libraries and retrieved in JSON format using an access token. In the case of data collection from mastodon.au, the number of toots is not excessively high, leading to a single Mastodon streamer on our master node to keep streaming data and stored into our CouchDB instance nodes.

```
nohup python3 stream_mastodon.py &
```

Script is then deployed on our main node using the nohup command from above, with the output of the script stored in nohup.out file in the same directory as the streaming script. When inspecting the nohup.out file when stream_mastodon.py is terminated, some errors are noted.

4.2.1 Error Handling

MarkupResemblesLocatorWarning: Certain toot's content is not in html format, which will cause MarkupResembles-LocatorWarning when processing with beautifulsoup, below are some examples of toot content that create this issue.

```
why is /HSG/ home stuck general the most active thread on my forum
Why a Makefile you may ask, and the answer is that `make &lt;task name&gt;` works on both macOS and Linux without the need to set up anything in advance.
So most likely, I can git clone and then run make setup / make dev right away.
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught MastodonNetworkError: Could not connect to streaming server: Bad Gateway
they should invent more precious stones so trans girls can have more names to choose from and/or hoard
*I eat the post above/below this one whole*
Context switching was brutal at work today. The day went in a haze spread between a myriad of small tasks/meetings. I'm pretty sure I did stuff but I'd ne
ed some effort to remember exactly what.
I have been rewatching InuYasha a bunch lately, and I think they missed the opportunity for InuYasha themed/branded menstrual products.
آخرین آمار کرونا در ایران/ فوت ۸ معوطن در شبانه روز گذشته
#livestreaming Relaxed Play with DknBluez Ep. 30 | Coffee time! Back at my Veecraft SMP Megabase! #minecraft #veecraftsmp http://twitch.tv/deaconstable 10
am EST!
from the appearance, it looks like it's UIKit with a bunch of custom controls. sorry to all the people who wanted AppKit/ProKit on iOS
```

Figure 5: examples of toot content

Using python's built-in function "warning", we skipped those toots while not terminating the program, keeping it streaming on cloud.

```
# Parse the HTML content using BeautifulSoup, toots are formatted in html form
# some toots may raise MarkupResemblesLocatorWarning, skip these toots as they might create
    problems
with warnings.catch_warnings(record=True) as w:
    soup = BeautifulSoup(status["content"], 'html.parser')
    for warning in w:
        if isinstance(warning.message, MarkupResemblesLocatorWarning):
            print(status["content"])
            return
```

Figure 6: actual error handling code

Mastodon streaming error:

Streaming toots also encountered several errors, below are some errors occurred while streaming Mastodon toots. The very first error we encountered is: Caught MastodonNetworkError: Could not connect to streaming server: Bad Gateway, where Mastodon's RESTful API has connectivity issues. To deal with this problem, we implemented an exception handling while loop using try&except to check current streaming status and used Mastodon's MastodonNetworkError module from the python library they provided to handle this specific error. To deal with future unknown errors, another except block is added to deal with general errors. Implemented code is shown below:

```
while True:
    try:
        listener = Listener()
        m.stream_public(listener)
    except MastodonNetworkError as e:
        print(f"Caught MastodonNetworkError: {e}")
        # sleep for a short period of time before attempting to reconnect
        time.sleep(15)
        # re-initiate mastodon api in case of an error
        m = Mastodon(
            api_base_url='https://mastodon.au',
            access_token='Ji9hWsGcTpCyzly8ibuNyCvcimle0P6SMQDDzTdvJjk'
            )
    except Exception as e:
            print(f"Caught Exception: {e}")
            time.sleep(15)
            # re-initiate mastodon api in case of an error
            m = Mastodon(
                api_base_url='https://mastodon.au',
                access_token='Ji9hWsGcTpCyzly8ibuNyCvcimle0P6SMQDDzTdvJjk'
                )
```

Figure 7: actual error handling code

We intended to “log” back on using Mastodon’s access token just to ensure that no further error will affect this streaming script. Nohup.out recorded all the errors the script faced since then, and script has been continuously running since our last deploy on 6th of May [Figure 9]. Below are the errors we faced but hasn’t affected live streaming of the script.

```
Caught Exception: [Errno 101] Network is unreachable
Caught MastodonNetworkError: Could not connect to streaming server: Bad Gateway
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: [Errno 101] Network is unreachable
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: [Errno 101] Network is unreachable
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught MastodonNetworkError: Could not connect to streaming server: Bad Gateway
Caught Exception: [Errno 101] Network is unreachable
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: [Errno 101] Network is unreachable
Caught MastodonNetworkError: Could not connect to streaming server: Bad Gateway
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: [Errno 101] Network is unreachable
Caught Exception: [Errno 101] Network is unreachable
Caught Exception: _ssl.c:980: The handshake operation timed out
Caught Exception: [Errno 101] Network is unreachable
```

Figure 8: error message

```
root      198352  0.0  0.2 949896 26560 ?        Ssl  May01   1:27 /usr/lib/snapd/snapd
ubuntu    252894  0.5  1.6 178784 150704 ?       S    May06 145:48 python3 stream_mastodon.py
ubuntu    371819  0.0  0.0   8944  5340 pts/52   Ss   May16   0:00 /bin/bash --init-file /home/u
ubuntu    378817  0.0  0.3 1294164 31052 pts/52  Sl   May16   0:00 npm run start 8000
```

Figure 9: Mastodon Streaming status

4.3 Mastodon data preprocessing


```
{'id': 110417509936173698, 'created_at': datetime.datetime(2023, 5, 23, 10, 24, 51, tzinfo=tzutc()), 'in_reply_to_id': None, 'in_reply_to_account_id'
: None, 'sensitive': False, 'spoiler_text': '', 'visibility': 'public', 'language': 'en', 'uri': 'https://mastodonapp.uk/users/id1om/statuses/1104175
09859046674', 'url': 'https://mastodonapp.uk/@id1om/110417509859046674', 'replies_count': 0, 'reblogs_count': 0, 'favourites_count': 0, 'edited_at':
None, 'content': '<p>Goodness.</p><p>Is that the time?</p><p>Daily greetings from this insignificant account in the milieu.</p>', 'reblog': None, 'ac
count': {'id': 109370920495667365, 'username': 'id1om', 'acct': 'id1om@mastodonapp.uk', 'display_name': 'Ian Davis', 'locked': False, 'bot': False, '
discoverable': False, 'group': False, 'created_at': datetime.datetime(2022, 11, 7, 0, 0, tzinfo=tzutc()), 'note': '<p>Harmless Drudge</p>', 'url': 'h
ttps://mastodonapp.uk/@id1om', 'avatar': 'https://o.mastodon.au/cache/accounts/avatars/109/370/920/495/667/365/original/212571f8e2f7b7ce.jpg', 'avata
r_static': 'https://o.mastodon.au/cache/accounts/avatars/109/370/920/495/667/365/original/212571f8e2f7b7ce.jpg', 'header': 'https://o.mastodon.au/cac
he/accounts/headers/109/370/920/495/667/365/original/93d96ae81b9a5277.jpg', 'header_static': 'https://o.mastodon.au/cache/accounts/headers/109/370/92
0/495/667/365/original/93d96ae81b9a5277.jpg', 'followers_count': 62, 'following_count': 181, 'statuses_count': 2269, 'last_status_at': datetime.datet
ime(2023, 5, 23, 0, 0), 'emojis': [], 'fields': []}, 'media_attachments': [], 'mentions': [], 'tags': [], 'emojis': [], 'card': None, 'poll': None, '
filtered': []}
```


Figure 10: sample of a toot

Here is a sample of a toot obtained through Mastodon’s RESTful API. It reveals that the toot contains some valuable information, although many details of the toot may not be as helpful. We would need to preprocess this JSON formatted data for future analysis.

Content, or posts from the pulled toot is in html format, therefore we'll need to convert this content to normal sentences. This step is done using beautifulsoup. Since we plan to compare Twitter user's sentiment against Mastodon user, we'll need a sentiment score of each toot. A sentiment score is calculated by ADO for Twitter data; however, Mastodon do not contain such value. Pre-trained library from textblob is used to calculate polarity of a toot, which lies in between [-1,1], where negative float indicates a negative sentiment, 0 means neutral and positive float for positive sentiment.

During this process of calculating sentiment score, something strange is spotted. All languages except English have a sentiment score of 0, indicating that sentiment of toot is not correctly calculated. It is found that textblob cannot provide sentiment polarity of non-english languages, therefore another python library is implemented. Googletrans library uses Google Translate Ajax API to access google translate's translate and detect functions. Language tag is included in Mastodon's toot data, non-english toots can be translated, and then employ textblob for sentiment polarity calculation.

The month and hour of toot is extracted, words in toot are also lemmatized for topic selection. Toots will then undergo the same topic selection procedure for as we did for Twitter's tweets. The preprocessed toot is then saved to our CouchDB database using CouchDB's python library. We can create new CouchDB database or stream into existing ones by simply checking whether the indicated database name exists or not.

4.4 SUDO and geospatial data

Datasets:

1. Government of the Commonwealth of Australia - Australian Electoral Commission, (2019): AEC - Federal Election - Polling Places (Point) 2019 [4]

2. Government of the Commonwealth of Australia - Australian Bureau of Statistics/Grattan

Institute, (2012): GI - Working Age Employment and Income (Suburb) 2011 [5]

3. Australian Statistical Geography Standard (ASGS) digital boundaries – Australian Bureau of Statistics (2021) [6]

The Spatial Urban Data Observatory (SUDO) provides access to many open-source datasets from a range of different organisations. This project used SUDO to access AEC - Federal Election - Polling Places (Point) 2019 and GI - Working Age Employment and Income (Suburb) 2011. The former provides voting data at polling stations and the location (longitude and latitude coordinates) of these polling stations for the 2019 federal election. The latter provided median income, employment percentages (male and female), and education (percentage of residents with university degrees) data for suburbs in Australia. SUDO provides a spatialise dataset tool to generate shape files for datasets, however, for GI - Working Age Employment and Income (Suburb) 2011 the spatialise tool failed to execute. Therefore, the necessary ASGS boundary shapefiles for states and suburbs were sourced from the Australian Bureau of Statistics. Boundary shapefiles for postcodes and greater capital cities were also used during preprocessing, however, the data obtained from these were not used in the final analysis/visualisations (see 9.1 for further information). The shapefiles were then processed through Mapshaper to reduce the boundary granularity to produce smaller file sizes for the GeoJSONs extracted from the shapefiles. This results in lower precision in boundaries (especially noticeable for suburbs), however, was a necessary step to reduce file size for better performance.

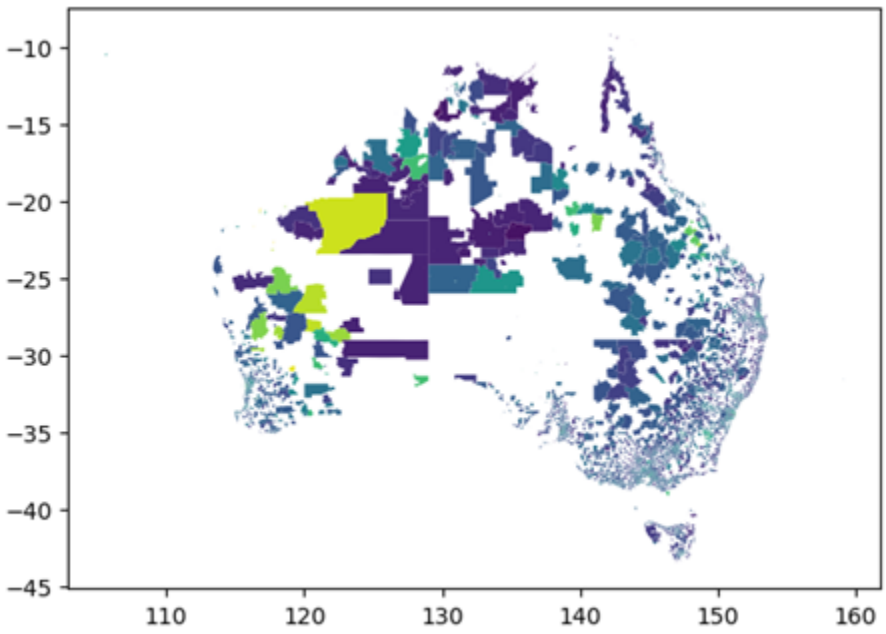


Figure 11: Choropleth of Suburb Median Income

The SUDO data was then combined with the GeoJSONs using GeoPandas, an open source Python library for geospatial data by extending the functionality of the Pandas library using Shapely. The election data was aggregated into states and suburbs by matching with the boundary enclosing the coordinate for the polling station, and the employment/income/education data was matched with the corresponding boundaries by matching suburb names. For the later case, some data loss/inaccuracy is expected as the GI - Working Age Employment and Income (Suburb) 2011 dataset was collected 10 years prior to the ASGS boundaries in the shapefiles/GeoJSONs, so some suburb names may not match and some boundaries may have changed over time. The median income from dataset obtained by merging GI - Working Age Employment and Income (Suburb) 2011 with the ASGS suburb boundaries is shown in Figure above.

5 Result gathering - Couch DB MapReduce

CouchDB utilizes a support feature of design documents called Views, allowing us to define functions and extract useful information from CouchDB's database. These views are created using MapReduce functions, which are stored and executed within CouchDB. CouchDB employs an incremental MapReduce View approach, capable of updating view results as new data is added or updated in the database, which makes it specifically useful since our Mastodon data is consistently being pulled from Mastodon server.

When views are created, all our CouchDB node and their corresponding shards will response to the same task, and we can access the view result from any CouchDB instance when all shards finished processing to ensure View result consistency. An example of our MapReduce View and the corresponding result for LGBT scenario about the monthly state's sentiment proportion is shown below, where we iterated key values of CouchDB database's files and emit the information we need for data visualisation and analysis. Reduce function will gather all emitted data with the same key provide an overview of the dataset.


```
{"key":["2022-02","australian capital territory","negative"],"value":{"count":12,"sentiment":1.221471029295401,"average_magnitude":0.10178925244128341}},
{"key":["2022-02","australian capital territory","neutral"],"value":{"count":23,"sentiment":0,"average_magnitude":0}},
{"key":["2022-02","australian capital territory","positive"],"value":{"count":17,"sentiment":2.60047518947043,"average_magnitude":0.15296912879237823}},
{"key":["2022-02","new south wales","negative"],"value":{"count":319,"sentiment":38.413042526736746,"average_magnitude":0.1204170612123409}},
{"key":["2022-02","new south wales","neutral"],"value":{"count":353,"sentiment":0,"average_magnitude":0}},
{"key":["2022-02","new south wales","positive"],"value":{"count":418,"sentiment":61.197621331815554,"average_magnitude":0.14640579265984582}},
{"key":["2022-02","northern territory","negative"],"value":{"count":4,"sentiment":0.8224879943973855,"average_magnitude":0.20562199859934638}},
{"key":["2022-02","northern territory","neutral"],"value":{"count":3,"sentiment":0,"average_magnitude":0}},
{"key":["2022-02","northern territory","positive"],"value":{"count":11,"sentiment":1.3944509554345403,"average_magnitude":0.1267682686758673}},
```


Figure 12: sample result of map reduce function

Map function

```
function (doc) {
  var month = doc.date;
  if (doc.LGBT === true && doc.state) {
    var sentimentCategory;
    if (doc.sentiment > 0) {
      sentimentCategory = 'positive';
    } else if (doc.sentiment < 0) {
      sentimentCategory = 'negative';
    } else {
      sentimentCategory = 'neutral';
    }
    emit([month, doc.state, sentimentCategory], {count:1, sentiment:doc.sentiment});
  }
}
```

Reduce (optional)

CUSTOM

Custom Reduce function

```
function (keys, values, rereduce) {
  var result = {
    count: 0,
    sentiment: 0
  };
  for (var i = 0; i < values.length; i++) {
    result.count += values[i].count;
    result.sentiment += Math.abs(values[i].sentiment);
  }
  return {
    count: result.count,
    sentiment: result.sentiment,
    average_magnitude: result.sentiment/result.count,
  };
}
```

Figure 13: sample map reduce function

One problem we faced is that our initial mastodon streaming script provided Russian and Ukraine war tag and rental tag (which we initially planned to analyse but without enough data). We then integrate tweet's scenarios detection procedure in CouchDB's map function 14. Even though this process takes longer as CouchDB has to iterate over all toots, we can still gather toots that relates to LGBT without the tag in streamed data.

```
var keywords_lgbt = ['questioning','Gay','Same-sex marriage','lesbian','LGBT','Gender identity','ase
  'Transphobia','bisexual','intersex','Coming out','Non-binary','Queer','Rainbow flag','trans','tran
  'heterosexuality','sexual orientation'];
var content = doc.content;
var language = doc.language;

var LGBT = false;
for (var i = 0; i < keywords_lgbt.length; i++) {
  var keyword = keywords_lgbt[i];
  // make it exact match so trans don't match transition
  var pattern = new RegExp('\\b' + keyword + '\\b', 'i');
  if (pattern.test(content)) {
    LGBT = true;
    break;
  }
}
```

Figure 14: sample map function with scenarios detection procedure

View result can be accessed through CURL command using RESTful API for our backend to do some more process for frontend delivery.

6 Back-end

6.1 Processing the result from CouchDB

Back-end applications use RESTful APIs to access views in CouchDB and retrieve data from CouchDB. It also processes the data into the format desired by the front end, such as JSON and GeoJSON formats. In our project, the front-end needed some GeoJSON-formatted data that needed to be mapped on the map. Another part of the data needs to be responded to in JSON format. For GeoJSON format, the back-end needs to store the data in a dictionary whose key is Australia state or Australia suburb. The values corresponding to keys are some sentiment mean values or sentiment proportion, etc., which need to be analysed. The values from the dictionary are then added to the SUDO data that has been converted to a GeoJSON file for easy retrieval and analysis. With JSON format, the back-end just needs to group by unique keys, such as language, state, and so on, and finally sends the data back to the front-end in JSON.

6.2 Flask

By using Flask to develop a back-end server, we benefit from the following advantages. Firstly, because we decided to use python to develop the front and back ends, and Flask is based on Python. Because the Python ecosystem is very strong and rich. Flask applications can be extended and enhanced by the various libraries and tools that Python provides. This includes libraries for database operations, data processing, testing, and more. And Flask is a lightweight framework that is easy to learn and use. It provides basic functionality and tools that make developing web applications simple and intuitive. Flask also provides flexible routing and view definition, allowing developers to customise according to application requirements.

However, Flask is a small web services framework that doesn't work well for large applications. Although Flask can be used to develop applications of all sizes, large and complex applications may require a more powerful framework or a more complex architecture like Django. Flask's flexibility and simplicity can lead to performance challenges when dealing with large numbers of concurrent requests or high loads. All in all, we benefit from using Flask to build a back-end server. Flask's simplicity and flexibility is an advantage for a small-scale application like ours.

Because the back-end needs to respond to the front-end with data for different scenarios, Blueprint using Flask provides a centralised registry entry for extensions. With Blueprints, applications can be managed in the Flask layer, share configurations, and register to change application objects on demand. So, by registering Blueprints from different scenarios in a file that we want to run, we can launch the app once and get all the data. And it also helps test responses to front-end data and facilitates later deployment.

6.3 RESTful API

RESTful API follows a set of conventions and principles that enable Web applications to communicate in a uniform, extensible, and easy to understand manner. We use RESTful APIs to transfer data between users and remote resources. The back-end application will use the corresponding

RESTful APIs to access views in CouchDB and retrieve data from CouchDB. One advantage of the RESTful API is that they use a uniform interface and standard HTTP methods, making API design simple and easy to understand. Therefore, it has good scalability, allowing new resources and endpoints to be added on demand. On the other hand, RESTful API can support multiple data formats, such as JSON and XML, making data exchange more flexible and universal. Contrast this with other APIs, such as the SOAP API, which uses XML as a message format and relies on additional protocols and standards (such as WSDL, UDDI, and so on). RESTful APIs use HTTP for communication. Therefore, RESTful APIs are easier to learn and use. And RESTful API takes full advantage of HTTP's caching mechanism to improve performance and efficiency.

However, RESTful API have some limitations, such as the possibility that certain requirements may not be met through RESTful APIs because RESTful APIs adhere to a set of specifications and constraints, requiring extension or adoption of other types of APIs. Another point is that when communicating using RESTful APIs, complete HTTP headers need to be transferred for each request and response, which can lead to additional performance overhead in the case of a large number of requests. Overall, for our project, we can benefit from using RESTful API to build back-end servers.

7 Front-end

7.1 JS and React JS

The frontend was developed using JavaScript, primarily using React JS. This enabled easier development via reusable components which could be utilised across different parts of the frontend, for UI components as well as data visualisation where map layers and plots could be developed as React components to be efficiently reusable with different datasets. Using React also enabled the use of Material UI, a React library which provides a wide range of UI tools for styling and layout of elements, this simplified the implementation of UI while also being relatively easily customisable

through MUI's built-in theme customisation tools.

Create React App was used to set up the web app as well as manage dependencies/packages automatically, this streamlined the development process as well as providing tools for straightforward optimisation and deployment of the production build through Docker to a NGINX server.

React-Router was used for navigation in the web-app, while simple to implement routing on the frontend using this method, it should be noted that it requires modification of the NGINX config file to prevent 404 errors when refreshing pages as NGINX will not recognise some routes, as discussed further here on stackoverflow.

7.2 Visualisations and Data/Query optimisations

For geospatial data visualisation, the library React-Leaflet was used. React-Leaflet provides React component focused methods to use Leaflet, a JavaScript library for implementing interactive maps. The GeoJSON components were used to visualise attributes for suburbs and states from the Twitter and SUDO datasets in choropleth layers, with layer control enabling a single map to be used for displaying information from multiple attributes based on the user's selection. Choropleth maps visualise geospatial data by colouring geometries based on some attribute value, typically with a continuous or discrete colour scale.

A discrete colour scale was chosen due to being more resistant to outliers since many suburbs had small sample sizes for Twitter data making extreme values (outliers) more common. An advantage of leaflet was the readily available functionality for interchangeable baselayers for maps, as choropleth maps can become difficult to interpret if the colours representing attributes become hard to distinguish from elements in the baselayer (e.g. green forests/hills causing incorrect perception of choropleth colour). Allowing users to toggle between baselayers enables them to find one such that they can more easily interpret the information according to personal preference.

As discussed in 'Section 4.2 SUDO and geospatial data', the GeoJSON boundaries were simplified for performance, this was due to the fetching of GeoJSON data being a major limitation on

the responsiveness of the frontend as large files would cause significant delays in loading visualisations. To further limit the impact of the GeoJSON files on performance, all necessary properties for states and suburbs were written into a single GeoJSON (one states GeoJSON and one suburbs GeoJSON), to avoid duplicate geometry information being fetched (as geometries account for the vast majority of file size), and the frontend queried the backend API immediately when users first enter the web-app, such that the GeoJSON data for maps is available sooner when users select a scenario.

The remaining data visualisations were implemented using Recharts, a composable charting/plotting library based on React components. Recharts provided a wide variety of charts/plots which can be modified simply using its component-based structure. The presence of interactive components such as brush bars enabled the implementation of interactive filtering of data, to provide more dynamic and customisable visualisations of data. As any additional datasets required for these visualisations were significantly smaller than the GeoJSON files, the frontend only queries the backend for this data when users select the corresponding scenario, to prevent unnecessary queries.

8 Scenario Data Visualisations and Analysis

Three scenarios were analysed using a combination of sentiment analysis performed on Twitter/-mastodon text, election behaviour, median income, education, and employment. The first and second scenarios, the Ukraine-Russia war (scenario 1) and LGBTQ (scenario 2), investigated the sentiments of users on Twitter and mastodon when discussing/mentioning keywords or phrases associated with the topic, while the third scenario investigated the activity of users over time (scenario 3) based on different user/text characteristics.

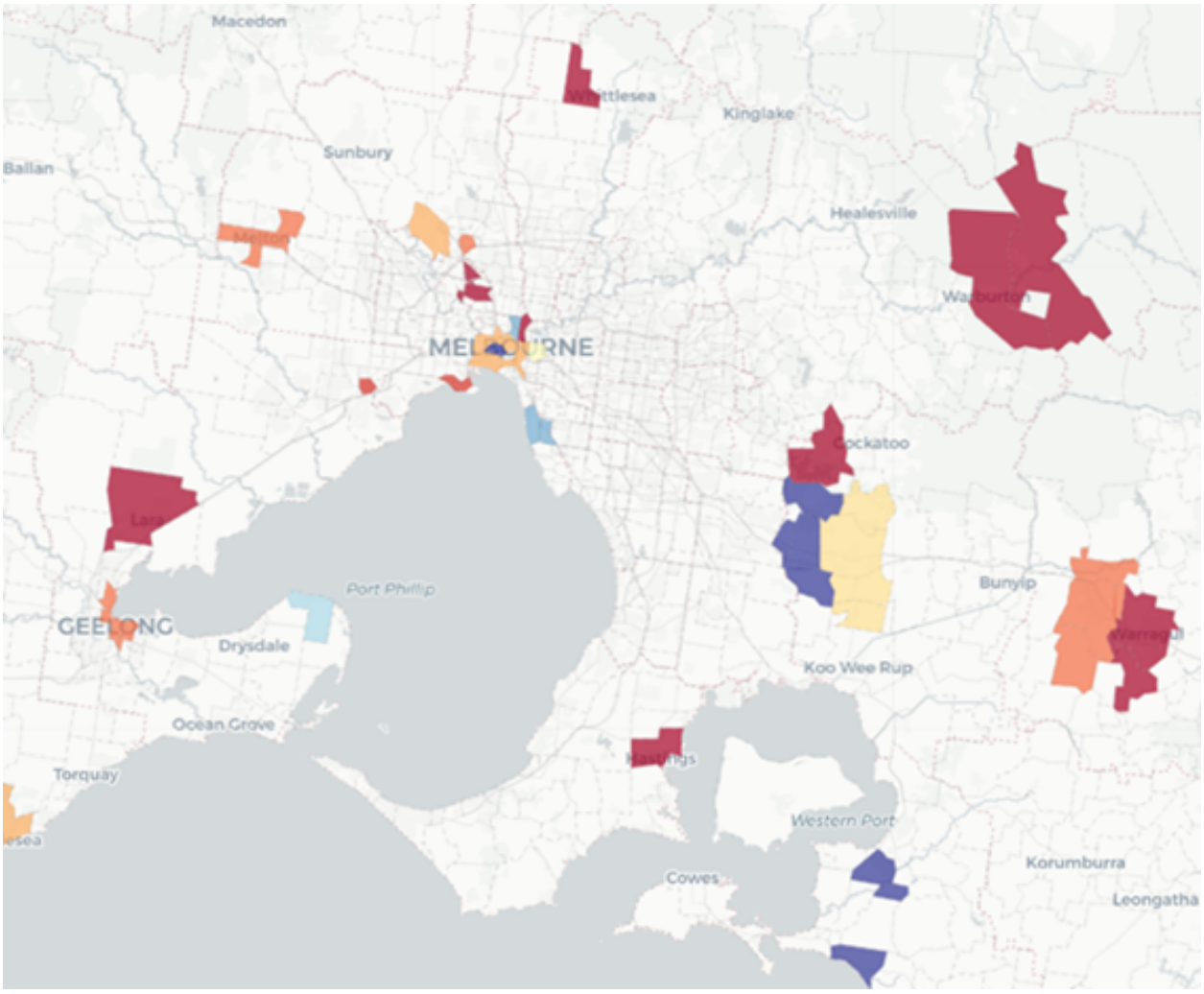


Figure 15: Sentiment of tweets about the Ukraine-Russia war in suburbs

For scenario 1 and 2, geospatial data was obtained from Twitter and SUDO. Each of these variables was visualised through the use of choropleth maps, an example of these visualisations is presented in figure 14. Unfortunately, only a small proportion of suburbs were able to be sampled from the available Twitter data, however, approximately 700 suburbs were able to be sampled. The Mastodon data did not contain any location information, so geospatial data could not be obtained for users/toots on the server.

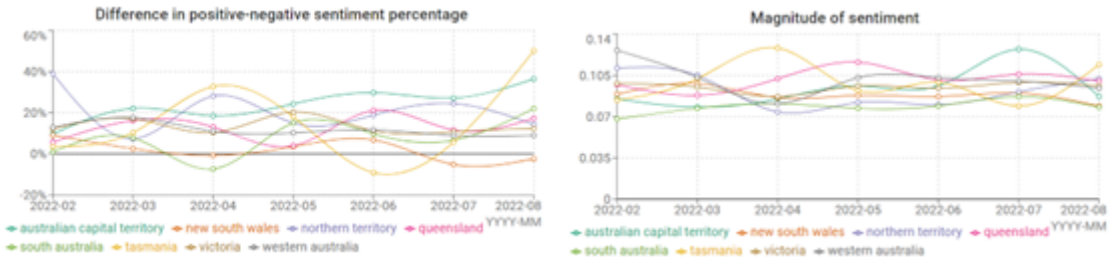
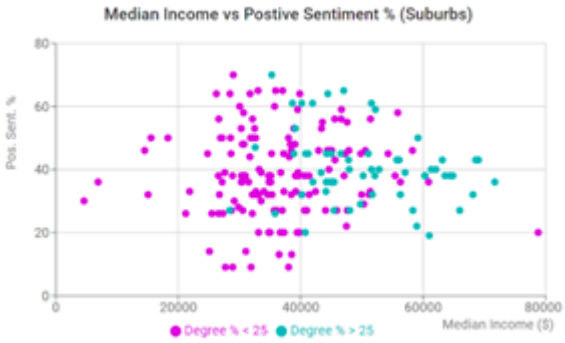


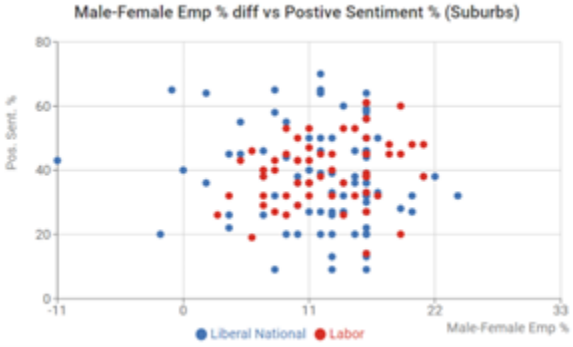
Figure 16: Percentage difference in positive and negative sentiment (left) and mean magnitude of sentiment (right) on Twitter

To investigate potential changes in sentiment over time towards these topics, the Twitter data was aggregated by time and state, the results of this analysis for scenario 2 (LGBTQ) are presented in figure 15. It was found that sentiment towards the Ukraine-Russia war was predominantly negative across all states/territories, although there was a slight trend towards more balanced sentiment over time. In contrast sentiment towards LGTBQ was predominantly positive across all states/territories, with the sentiment of states/territories seeming to diverge over time. However, the mean magnitude of sentiment for both scenarios remained relatively stable throughout this period. States/territories with smaller populations seemed to have greater fluctuations in sentiment, however, this is likely due to small sample sizes for tweets from these regions, rather than significant evidence of major changes between each month.

The overall sentiment and sentiment among the most popular languages on Twitter and Mastodon was also analysed, showing a much higher proportion of positive sentiment when discussing the Ukraine-Russia war on Mastodon compared to Twitter, and also a much higher proportion of positive sentiment when discussing LGBTQ topics on Mastodon compared to Twitter.



(a) Positive sentiment proportion and median income grouped by university degree percentage



(b) positive sentiment proportion and male-female employment difference grouped by political party preference

Figure 17: figures for scenario 2 (LGBTQ).

To observe potential correlations with characteristics of the suburb populations, scatter plots as shown in figure 16 were used for both scenario 1 and 2. No clear correlations between factors/variables was observed, except for median income being correlated with education (although this is a well known correlation). However, for scenario 2, there seems to be some indication of a decrease in sentiment proportion variance among suburbs which voted Labor as the male-female employment difference decreases. There may also be some positive correlation between positive sentiment and both education and median income for scenario 2, however, this is ambiguous. For scenario 1 there does not appear to be any weak or strong correlations between any factors/variables (other than the previously observed correlation between education and income).

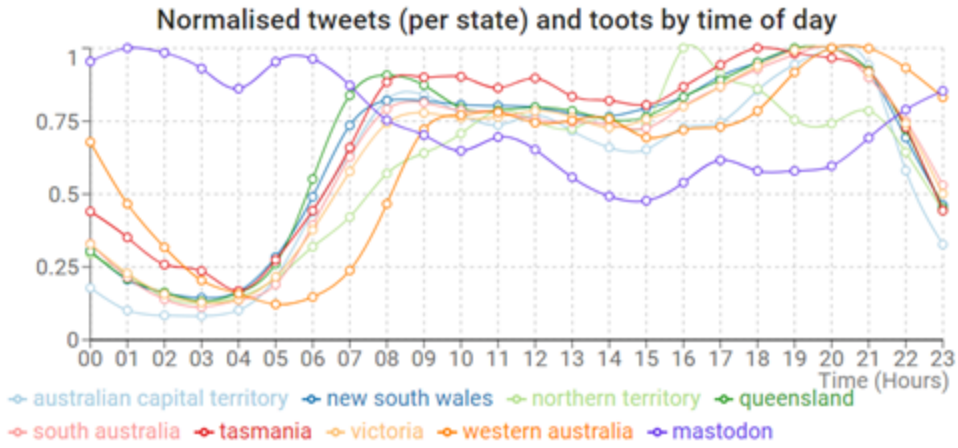


Figure 18: Tweets by state and toots on Mastodon server by time of day (AEST).

For scenario 3, line charts were used to determine trends in activity throughout an average day on Twitter and Mastodon. Twitter activity across states displayed similar trends (just shifted in time due to time zone differences), highest activity during 6-9pm and very little activity between 12-5am, as expected since this corresponds with when most people have finished work/school and have more free time to be active on social media, and also a drop in activity during the time when most people are sleeping. Victoria and NSW had the most activity by a large margin, due to having the largest populations. Mastodon shows a very different trend, with highest activity during 11pm-6am, when australians would be sleeping, and in strong contrast with Twitters activity for australians, suggesting that there is likely a large proportion of international users active in this australian mastodon server. While the presence of international users in the Mastodon dataset is unsurprising since location filtering could not be applied, such a large international presence on the Australian server was unexpected.

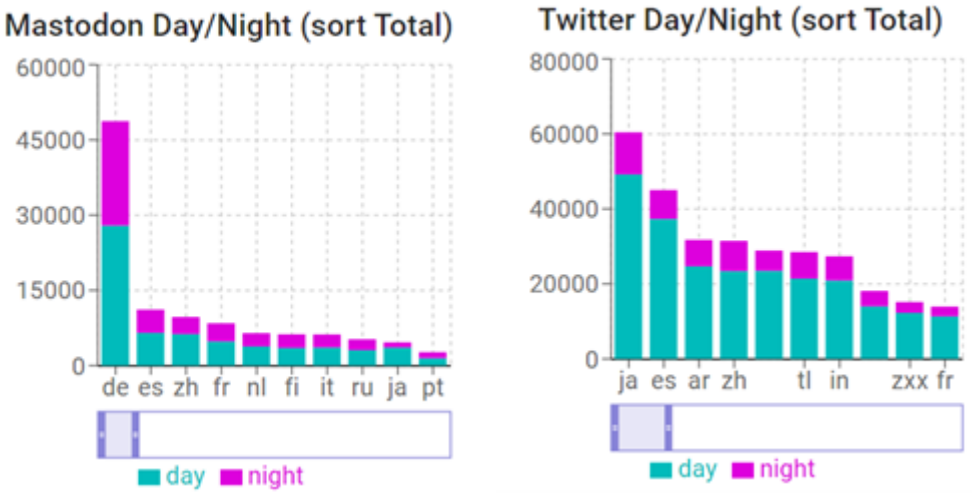


Figure 19: Day and Night activity on Twitter and Mastodon for most popular languages on each platform.

Languages, excluding English were analysed based on the time a tweet/toot was posted. Japanese is the language tweeted most on Twitter, whereas German is the most frequent language in Mastodon's toots. Spanish and Chinese top the list of posts on both social media, which is understandable given the popularity of both languages. All languages on Mastodon seemed to have a higher proportion of night time activity when compared with Twitter.

Generally speaking, Mastodon had a higher proportion of toots posted during night time (10pm-5am) compared to Twitter. We propose two possible reasons for this finding. Mastodon is a new social media platform launched in 2016, so there might be a higher proportion of young users staying up late. Another possible reason is that since the Mastodon dataset did not contain post location information, anyone around the world could post at Australia's server and be included in the analysis, which could cause this misleading result.

9 Project GitHub and Demonstrations links

Project GitHub: `https://github.com/yijun-github/ccc`

Youtube Demonstration Videos:

Couchdb cluster Set up: `https://youtu.be/D1nRjd29T2c`

Frontend/UI: `https://youtu.be/ZwgfauaIFXg`

10 Team Collaboration

Table 1: Team responsibilities

Team Member	**Responsibilities**
Aobo Li	Couchdb map reduce Mastodon harvesting Scenario analysis
Pavith Samarakoon	Frontend development SUDO/Geospatial data processing Data/Scenario Analysis
Zhihao Liang	Setup development environment Deploy Couchdb cluster Import Twitter data
Jiqiang Chen	Twitter data preprocessess
Yijun Liu	Backend development

References

[1] A. Kurapov, V. Pavlenko, A. Drozdov, V. Bezliudna, A. Reznik, and R. Isralowitz, "Toward an understanding of the russian-ukrainian war impact on university students and personnel," *Journal of Loss and Trauma*, vol. 28, no. 2, pp. 167–174, 2023.

[2] *Defining LGBTQIA+ — gaycenter.org*, https://gaycenter.org/about/lgbtq/, [Accessed 23-May-2023].

[3] *Mastodon dataset*, https://mastodon.au/public, [Started Streaming from 06-May-2023], 2023.

[4] *Australian electoral commission: Aec - federal election - polling places (point) 2019*, [Retrieved May 2, 2023, from https://sudo.eresearch.unimelb.edu.au/].

[5] *Australian bureau of statistics/grattan institute: Gi - working age employment and income (suburb) 2011*, [Retrieved May 2, 2023, from https://sudo.eresearch.unimelb.edu.au/].

[6] *Australian bureau of statistics - australian statistical geography standard (asgs) digital boundaries*, [Retrieved May 5, 2023, from https://www.abs.gov.au/statistics/standards/australian-statistical-geography-standard-asgs-edition-3/jul2021-jun2026/access-and-downloads/digital-boundary-files], 2021.