# COMP90024 Assignment 2 Report
# Group 46


Kunal Patel
Student ID: 1291822
Email: kppatel@student.unimelb.edu.au

Mayank Yadav
Student ID: 1403092
Email: mayanky@student.unimelb.edu.au

Sophie von Doussa
Student ID: 1064884
Email: svondoussa@student.unimelb.edu.au

Harsh Mangla
Student ID: 1418017
Email: hmangla@student.unimelb.edu.au

Maxson Stephen Mathew
Student ID: 1428525
Email: matmm@student.unimelb.edu.au

# Contents

# 1 Introduction

Huge volumes of data are being generated every second in the era of social media dominance, offering priceless insights into many facets of human life. This project seeks to analyse the massive Twitter corpus and examine its correlation with official data from the Spatial Urban Data Observatory (SUDO) by utilising the capabilities of cloud technologies and data analytic techniques. By merging these two comprehensive sources of knowledge, we can learn more about Australian culture and unearth amazing tales buried within the social media environment.

The project starts with the gathering of Twitter data from the Mastodon API and the Australian Data Observatory (ADO). The larger Twitter corpus will be supplemented by postings from Mastodon APIs, which students will have the chance to collect from these networks. The collected data will be merged into a CouchDB database along with the datasets offered by the SUDO platform, creating a comprehensive resource for study and comparison.

**Video links**
Architecture: https://youtu.be/le-mBZZM7e4
Dashboard: https://youtu.be/J9P2MBzP1Wo

**GitHub Link** To establish the technical requirements for the implementation of a version-control system designed for the purpose of sharing and managing code: https://github.com/bulkpanda/cloud-computing-project

# 2 System Architecture and Design

## 2.1 System Architecture Diagram

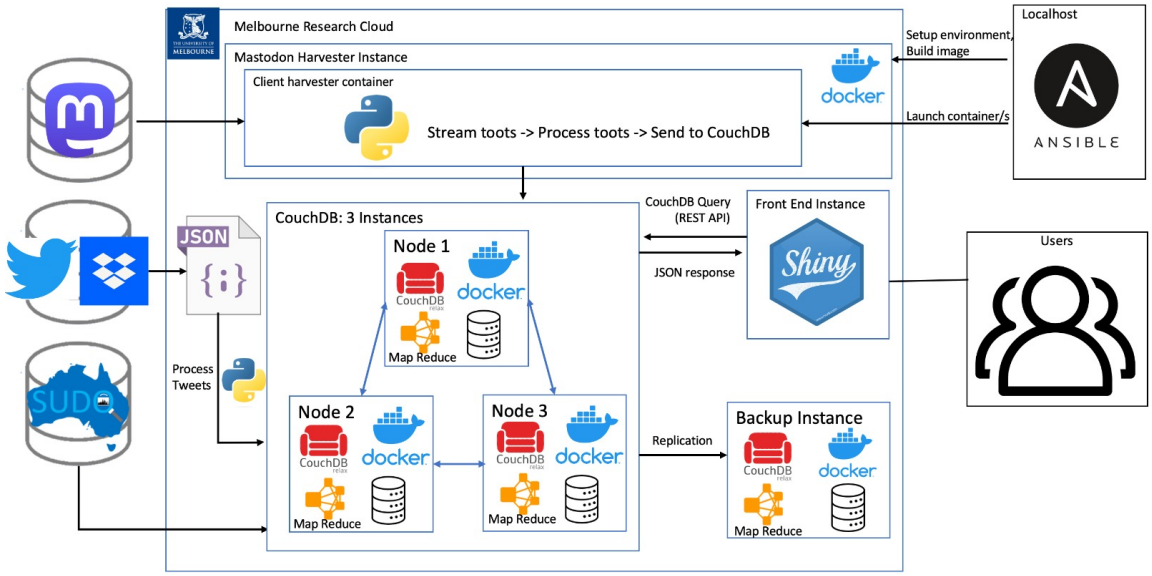



Figure 1: System Architecture Design Diagram

The architecture uses six instances, four for the database servers, one for web-server and one for mastodon harvesters. Detailed description is given in the later sections.

## 2.2 Melbourne Research Cloud

The University of Melbourne in Melbourne, Australia, offers a high-performance computer and data storage infrastructure called the Melbourne Research Cloud (MRC). By providing academics with access to strong computer resources and cutting-edge data storage capabilities, it is intended to encourage research and cooperation across a wide range of fields.

Researchers can use the MRC's scalable and adaptable platform to analyse big datasets, run intricate simulations, and carry out computations that require a lot of data. To meet various research needs, it provides a range of computing solutions, such as virtual machines, high-performance clusters, and specialised hardware accelerators.

Researchers can manage their computing resources, start and control virtual machines, save and retrieve data, and more by gaining access to the Melbourne Research Cloud through a web-based interface or command-line tools.

It gives University of Melbourne academics access to a broad range of on-demand virtualized computing resources (such servers and storage) using Infrastructure as a Service (IaaS) cloud computing. The service saves researchers the effort and expense of setting up their own compute environment and enables them to instantly access scalable computational capability as their research expands.

### 2.2.1 Resource Allocation

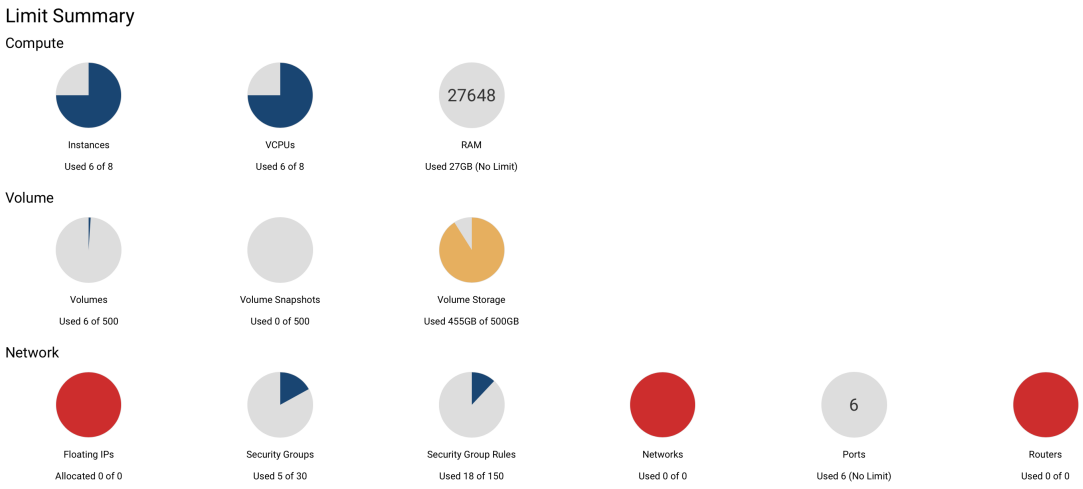



Figure 2: MRC Allocation

We are allocated with 6 servers (instances) with 6 virtual CPUs (36GB memory total) on the Melbourne Research Cloud.

## 2.3 MRC Advantage

### 2.3.1 Costs

A significant benefit of utilising the UniMelb Research Cloud is its low price. We have no hardware costs, to begin with. We can benefit from this for our work, as an on-premise infrastructure requires a substantial initial hardware investment. As a consequence, we do not have to worry about the cloud system's hardware and infrastructure costs. Using the MRC can also reduce our administrative load. On the one hand, we do not need to pay a utility bill for the UniMelb Research Cloud system, which reduces our daily maintenance expenses. On the contrary, the use of cloud services is complimentary. Therefore, there is no service charge for the undertaking. Moreover, we do not have to pay for the use of cloud services in the MRC, in contrast to the expensive cost of services hosted by prominent cloud providers, such as AWS and AZURE.

### 2.3.2 Security

Utilising the UniMelb Research Cloud's security is a significant advantage. First, a high level of security is established. As the allocated cloud resources belong to a single consumer, which is our organisation, we have access to them. Thus, the infrastructure and systems of the cloud can be configured by us to provide a high level of security. For instance, we can log in to the cloud instance using a pair of encrypted keys, and we can modify the data volume attachment, etc. Second, we can have faith in the cloud service provider. As stated in the Privacy Policy (MPF1104)1, the MRC ensures compliance with the legal data protection principles for consumers. Consequently, we can confidently store the data in the cloud. Thirdly, we can customise the security group settings. By modifying the rules in the security group, we can design the port utilisation for each instance at will. Consequently, the security of each cloud instance is enhanced.

1. Access Controls: The MRC implements stringent access controls to guarantee that only authorised researchers can access and administer their resources. To verify user identities, authentication mechanisms such as usernames and passwords are employed.

2. Encryption of Data: Data transmission and storage are encrypted within the MRC to prevent unauthorised access. Encryption algorithms are used to secure data during transmission over networks and storage.

### 2.3.3 Scalability

The Melbourne Research Cloud (MRC) is designed to be extremely scalable, allowing researchers to scale their computing resources according to their individual requirements. The infrastructure of the Medical Research Council provides a variety of scalability options to accommodate varying computational demands and duties.

1. Allocation of Resources: The MRC utilises a flexible resource allocation model, allowing researchers to request and allocate computing resources on-demand. Researchers are able to modify their resource allocations as necessary, scaling up during times of high computational demand and scaling down when resources are not actively required. This assures optimal utilisation of resources and cost effectiveness.

2. Storage Scalability: The MRC offers storage options that are scalable, enabling researchers to store and manage large volumes of research data. The MRC offers a variety of storage solutions, including network-attached storage (NAS), object storage, and archival storage. Researchers can scale their storage capacity based on their data requirements.

### 2.3.4 Flexibility

The UniMelb Research Cloud's adaptability is a significant benefit. Firstly, the MRC is easily customizable due to the fact that the hardware and other resources can be easily modified by the development team. We can send request e-mails to the MRC customer team to meat our cloud customization requirement.Secondly, we can take advantage of the MRC's flexibility to reallocate resources. Users are provided with a virtual machine by the MRC. A virtual machine can be scaled up or down without interruption, unlike a physical machine. And when you own all the virtual machines, you can dynamically deploy them wherever they are required the most. Thirdly, the issue of lock-in is not cause for concern. The URC is a private cloud, making it difficult to transition from AWS to Azure.

### 2.3.5 Reliability

The Melbourne Research Cloud (MRC) offers simple, on-demand access to scalable, virtualized computing capacity for storing, managing, processing, analysing, and backing up research data. It enables you to create virtual machines and servers on-demand without the hassle of purchasing and maintaining hardware. With the Melbourne Research Cloud, you can bring computers online in a matter of minutes and release them without having to dispose of tangible hardware.

### 2.3.6 Availability

High availability is a significant advantage of using the UniMelb Research Cloud. The MRC makes server creation accessible to all researchers and support staff, removing the need to plan, purchase, maintain, or dispose of tangible hardware devices. The MRC is accessible 24 hours a day, seven days a week, both within and outside the University of Melbourne, allowing researchers to work from any location. The platform offers comprehensive tutorials for all operating system users, making it user-friendly and simple to navigate. In addition, the MRC provides rapid deployment of software applications via Ansible scripts, thereby eliminating the need for complex configuration steps and excessive clicking on UI management interfaces. In addition, the ability to archive volumes in the cloud guarantees data recoverability and provides an additional layer of data security.

## 2.4 MRC Disadvantages

### 2.4.1 Technical Support Limitations

There is no set standard for the availability or responsiveness of technical support services. When trying to receive timely support for difficult technical challenges or troubleshooting, researchers could run against delays or other obstacles.

### 2.4.2 Resource sharing

The Melbourne Research Cloud (MRC) has a number of drawbacks, one of the most notable being the need to share resources. The MRC is a shared computing environment, therefore its computer resources are distributed among a number of different users. This indicates that the available resources, such as the central processing unit, memory, and storage space, are shared among the various researchers who are working at the same time. Researchers can face poor performance or limited availability during peak usage periods, which occur when there is a large demand for resources and a limited supply of those resources. This is due to the fact that the resources are required to be shared across all of the users, and when there is excessive concurrent usage, the resources may become stretched or confined.

### 2.4.3 Availability

Major disadvantages of utilising the UniMelb Research Cloud may include availability issues. Figure 2 demonstrates that we have limited resource allocation. On the cloud, we cannot store as much data as we would like, for instance. The second concern is the capacity ceiling. There may be a server or storage capacity restriction imposed by the service provider's physical hardware limitations. Thirdly, we lack tangible access to the data, making it impossible to physically transfer the data from one location to another. Even though we have extensive documentation and a tutorial on how to use the MRC, we still need to invest time in using the MRC professionally.

## 2.5 Database

### 2.5.1 CouchDB

We are using CouchDB database service to fetch and store the data needed. CouchDB is being used because it is an open source software designed for the web architecture and is convenient for big data analysis because of its powerful MapReduce function.

Fauxton interface is handy and powerful and can be directly accessed through browser, it allows a better way to manage CouchDB setup using GUI instead of using command lines. It also allows MapReduce function to be done on the interface and MapReduce's results can be checked immediately. It provides simple to use http API to process queries. Writing data and reading data from CouchDB simultaneously does not lead to any conflict and error message. Multi-version concurrency control is used in data management system of CouchDB so we do not have to stop the harvester to read the data for analysis.

All of the instances use docker containers to run CouchDB servers, couchdb3 image by ibmcom is being used here.

Python package CouchDB is used to connect to CouchDB server for exporting the mastodon data from harvesters and for exporting the tweets from tweets-corpus. Fauxton interface provides a better way to manage CouchDB set up and check for data updates.

### 2.5.2 Database Server Architecture

The database architecture contains a cluster of three couchDB servers, each being deployed on a different instance. The mastodon harvesters and the web server serially try to connect to all the servers. So, if there is an issue with one instance, the frontend and the mastodon harvesters can still work with other clusters, this provides resilience to single instance failure.

Beyond that, a backup server is also created on a fourth instance with full data replication from the clusters. In case someone accidentally deletes a database from the cluster setup, the backup server will still have it. The data can then be replicated back to the cluster setups for re-use.

The data in the database server is being stored in the mounted volume attached to each instance, so if an instance fail we can simply attach the volume to another instance. The volume being allocated to each instance is 100GB which is more than enough for this project.

### 2.5.3 Databases

There are 5 databases on the couchDB cluster:

1. tweets: Corpus of all the tweets that contain geographical information
2. tweets-covid: Tweets that are covid related
3. income-data: Information regarding regions and their median income
4. population-data: Information regarding regions and their population
5. mastodon-toots: The data coming from mastodon harvesters.

Out of these the toot database is dynamic, i.e, it changes as more live data from toots come in. The other databases are static, i.e., there is no live data being sent to them.

### 2.5.4 CouchDB Setup

Instances are created in flavor of uom.mse.1c4g. Network used is qh2-uom-internal. Source is NeCTAR Ubuntu 20.04 LTS (Focal) amd64. Security groups ssh, http, rdp and default are used.
In this setup, other VMs in the MRC can access each other but are not open to external world except for port 22 and 80.

Volume is then attached to the instance with same source, then we need to run the script to mount the volume via /data directory.
Docker engine is then installed on the VM.
Since, we need to store the data on the mounted volume. We need to change the default volume creation path for docker to /home/ubuntu/data/docker.
We then pull ibmcom/couchdb3 docker image and use then create the couchdb server. Ports 5984, 4369 and 9100-9200 are linked to the Machine's port.
Cluster creation is then done. The current servers run on 172.26.135.183, 172.26.130.94, 172.26.131.200 with backup server on 172.26.136.119.
Shell script createcouchDB.sh in database_creation repository contains the commands needed to create a cluster after docker installation.

### 2.5.5 Database creation

To create tweet database, the files createcoviddb.py and createprofanitydb.py should be run, the file path to twitter data and ip address to couchDB server should be set accordingly.
To create views, python is used as it gives easy to understand code. All the view files in the createviews file should be run after creating the databases to create MapReduce views.

### 2.5.6 Database creation

To create tweet database, the files createcoviddb.py and createprofanitydb.py should be run, the file path to twitter data and ip address to couchDB server should be set accordingly.
To create views, python is used as it gives easy to understand code. All the view files in the createviews file should be run after creating the databases to create MapReduce views.

## 2.6 Mastodon Harvester Clients

In our system architecture, we have a dedicated instance for streaming Mastodon Toots. This instance leverages the power of Ansible and Docker to effortlessly create multiple harvester clients. These clients can be dynamically scaled in both directions, providing a flexible and straightforward approach to managing our data streaming capabilities.

### 2.6.1 Mastodon Harvester Instance Setup

Instances are created in flavor of uom.mse.1c4g. Network used is qh2-uom-internal. Source is NeCTAR Ubuntu 20.04 LTS (Focal) amd64. Security groups ssh, http, rdp and default are used.

No volume is attached to this instance as it exclusively handles streaming data. The data is processed in an intermediary step without being written to disk, serving as a pure stream.

The IP address of the instance then needs to be added to the mastodon-harvester/inventory/inventory file, along with the private key used for ssh connection. The current mastodon harvester instance is running on 172.26.128.250.

### 2.6.2 Ansible

Ansible is a powerful open-source configuration management tool which can assist in the creation and deployment of software systems. Ansible has powerful OpenStack cloud capabilities to create and configure cloud instances at scale. However, during the lifetime of this project many reoccurring issues with the OpenStack reduced the scope of Ansible in this assignment. In this project, Ansible has been used to facilitate the dynamic deployment of Mastodon toot harvester clients.

While Ansible playbooks allow a certain level of freedom in their structure, they are best organised in a hierarchical manner. For this project, the recommend best practices were followed from the official Ansible documentation, https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html, leading to the following directory structure.

```
mastodon_harvester
├── Dockerfile
├── config.ini
├── inventory
│   └── inventory
├── mastodonStreamer.py
├── requirements.txt
├── roles
│   ├── build-docker-image
│   │   └── tasks
│   │       └── main.yaml
│   ├── common
│   │   └── tasks
│   │       └── main.yaml
│   ├── install-docker
│   │   └── tasks
│   │       └── main.yaml
│   ├── start-docker-container
│   │   └── tasks
│   │       └── main.yaml
│   └── stop-docker-container
│       └── tasks
│           └── main.yaml
├── run-docker-container.sh
├── run-docker-container.yaml
├── run-mastodon-setup.sh
├── run-mastodon-setup.yaml
├── stop-docker-container.sh
├── stop-docker-container.yaml
├── tootProcessor.py
└── variables
    ├── delete-client.yaml
    ├── mastodon-client-1.yaml
    ├── mastodon-client-2.yaml
    └── mastodon-client-3.yaml
```

Once the harvester instance is created on the Melbourne Research Cloud, the inventory file must specify the harvester instance IP and the private key file to use for the ssh connection. There are three yaml playbooks, which can each be executed from their corresponding shell script. Each shell script can be executed from localhost.

**1. run-mastodon-setup.yaml**

This playbook will configure the instance environment by running three main roles which are located inside the roles directory. First, common will install the required dependencies and update pip. Second, install-docker will ensure the latest version of Docker is installed. Lastly, build-docker-image will copy the required files to build the image from localhost into the instance, and will build the mastodon-harvester-image from the Dockerfile.

**2. run-docker-container**

This playbook will start a container with the newly made image. It will take variables from a specified variable file, mastodon-client-x.yaml. Herein lies the ability to dynamically scale the harvester functionality. Within the specified variable file, a base_url and MASTODON_ACCESS_TOKEN must be specified for any desired server. Running this playbook will then create a Docker container which will run the toot streamer and processor for the specified server.

**3. stop-docker-container**

This playbook takes a container name variable from the file delete-client.yaml variable file and will stop any docker container under this name. This allows the harvester to be dynamically scaled down as required.

### 2.6.3 Docker

The deployment of a containerised architecture through the use of Docker in this context gives very powerful system advantages. Docker facilitates the highly scalable, simplistic, efficient and manageable deployment of Mastodon client harvesters. The Docker image created by the run-mastodon-setup.yaml playbook installs all the dependent Python modules, meaning they only need to be downloaded once to then be utilised by multiple containers. This design is simplistic, as clients can be dynamically scaled up or down through simple configuration of the appropriate variable files. Containerisation ensures consistency between the

different client harvesters and facilitates portability across machines. Docker also manages CPU usage of the running containers, further abstracting complexity away from the user.

#### 2.6.4 Mastodon streamer & Toot processor

When a Mastodon client Docker container is instantiated, it is given the command:

"*python*3","*mastodonStreamer.py*","*base_url*","*mastodon_access_token*"

The container will run the mastodonStreamer.py file with command line arguments specifying the server base_url and access token.

The configuration file, config.ini, contains the couchDB configurations; that is the three IP addresses of the clustered database, and the database name, mastodon_toots. Leveraging benefits of a clustered database, the client application will attempt to establish a connection with each CouchDB node until it is successful. This approach eliminates the vulnerability of a single point of failure, ensuring uninterrupted access to the database even if one of the nodes becomes unavailable.

The Python package Mastodon.py provides the essential functionality for streaming toots from Mastodon. It serves as an API that enables seamless connection to Mastodon and facilitates toot streaming using the base_url and mastodon-access-token specified.

Each incoming toot is handled by the stream listener and passed to the TootProcessor. The TootProcessor plays a crucial role in extracting the required fields from the toot, converting the content from HTML to plain text, language detection, and filtering out non-English toots. Additionally, it also calculates the occurrence of hotwords and generates a sentiment score for further analysis. Once processed, the toots are transmitted to the connected CouchDB node, with their corresponding Toot ID serving as their ID in the database. This mechanism ensures that duplicate toots are not stored within the database.

# 3 User guide

## 3.1 CouchDB

- Instances are created in flavor of uom.mse.1c4g. Network used is qh2-uom-internal. Source is NeCTAR Ubuntu 20.04 LTS (Focal) amd64. Security groups ssh, http, rdp and default are used.
- In this setup, other VMs in the MRC can access each other but are not open to external world except for port 22 and 80.
- Volume is also created with NeCTAR Ubuntu 20.04 LTS (Focal) amd64 image is then attached to the instance, then we need to run the script to mount the volume via /data directory. Use mountvolume.sh script to mount the volume
- Docker engine is then installed on the VM. Instructions are at , or you can run dockerinstall.sh script.
- Since, we need to store the data on the mounted volume. We need to change the default volume creation path for docker to /home/ubuntu/data/docker. Follow the instructions at change volume root path for docker
- We then pull ibmcom/couchdb3 3.2.1 docker image and use then create the couchdb server. Ports 5984, 4369 and 9100-9200 are linked to the Machine's port.
- Cluster creation is then done. The current servers run on 172.26.135.183, 172.26.130.94, 172.26.131.200 with backup server on 172.26.136.119. They can be accessed via coucdDB1.txt key.
- Shell script createcouchDB.sh in database_creation repository contains the commands needed to create a cluster after docker installation. You will need to modify some parts like ip-address of the clusters.

## 3.2 Frontend

The frontend application can be accessed through this URL: http://172.26.132.13/ after running the commands provided below on the MRC instance.

### 3.2.1 Commands To run in MRC instance

1. To connect to your MRC instance run the following ssh command :

$$"ssh - icouchDB1.pemubuntu@172.26.132.13"$$

2. Next go to R files location and then to git folder using cd:

$$"cd\ r - files","cd\ git"$$

3. Pull the latest R application code from the git repository:

$$"git\ pull\ origin\ master"$$

4. Change directory to UI2:

$$"cd\ UI2"$$

5. Run the following command to launch web application:

$$"R\ -e\ "library(shiny) : shiny :: runApp('app.R', port = 3838)""$$

## 3.3 Mastodon Harvester

- Instance is created manually on MRC in flavor of uom.mse.1c4g. Network used is qh2-uom-internal. Source is NeCTAR Ubuntu 20.04 LTS (Focal) amd64. Security groups ssh, http, rdp and default are used.
- Ensure instance IP and private key for ssh connection are specified in /mastodon_harvester/inventory/inventory file
- Ensure config.ini contains the updated CouchDB IP addresses and database name for toots
- To configure the instance, from localhost run:

$$./run - mastodon - setup.sh$$

- To deploy a client harvester, specify the server name, base url and mastodon access token within a /variables/mastodon-client-X.yaml. Ensure this variable file is specified in run-docker-container.yaml under vars_files.
- To deploy container, run:

$$./run - docker - container.sh$$

- These steps can be repeated for any number of Mastodon server clients.
- To remove a client harvester, specify the server name in /variables/delete-client.yaml and run:

$$./stop - docker - container.sh$$

# 4 Frontend Web Application

To visualise our analysis effectively on a web application we used R shiny to develop our frontend application.

## 4.1 R Shiny Web App

R Shiny is an open source R package which helps us in building a powerful and elegant web framework for web applications using R programming language. Shiny also helps in turning analysis into interactive web applications, dashboards and data visualizations. It has become a popular framework in the data science community as the visualization in Shiny is easy and does not require much knowledge of coding frameworks beforehand. It has different blocks for user interface and server-side logic. In the user interface block we define the layout and components whereas in the server, it manages the data processing and logic. The only limitation that exists is the considerations for large-scale applications. We are using R version 4.3.0 in our application.

## 4.2 Why R shiny?

The very reason to use R shiny to develop our frontend application are the below mentioned key features that helped us to effectively visualize our analysis.

1. **Reactive Programming:** This feature of shiny allows user to create dynamic and responsive web applications based on input changes.
2. **UI Components:** There are a lot of libraries(e.g., plotly, leaflet) in shiny that can be used to render different types of visualizations or components that we want to render. Also, many input types can also be added to get inputs from user using different input controls(e.g., sliders, dropdowns).
3. **Server-side Processing:** Shiny also allows server-side processing, which makes complex calculations, data manipulations, and integration with external APIs and databases easy.
4. **Customizations and Styling:** It also provides an easy way to manipulate the appearance and styling of the application using CSS and HTML.

## 4.3 Packages Used

Various non default packages were put to use in order to display maps and graphs in the web applications. These packages and their functionalities are explained as follows.

1. **'leaflet':** We have used a leaflet package which helps us to deploy interactive maps. There are various map layers inside leaflet like tile layers, polygons, markers, etc. It can be added based on the appearance and behavior that is needed. It provides various controls, such as zoom control, scale control and layer control.
2. **'plotly':**It helps us to create interactive and dynamic visualizations in R Shiny. Plotly offers a wide range of graph types whether it be scatter plots, bar charts, line charts and many more. It also provides features such as zooming, panning and hovering. Users can interact with the graphs in real-time and update plots based on the results that the user wants to see.
3. **'jsonlite':**This package provides functions for working with JSON data in R. It offers us a set of functions to convert JSON data to R objects and vice-versa. It makes parsing, manipulation and analyzing easy within the R environment. It also provides efficient methods for handling large JSON datasets, minimizing memory usage and improving performance.
4. **'httr':**This package helps us for making HTTp requests and interacting with web APIs. Communicating with external APIs is a lot easier using this package. It also provides additional functionality of passing query parameters, form data, or custom headers to the HTTp requests. This package also includes support for secure communications using SSL/TLS protocols.
5. **'ggplot2':**This package is used for generating visually appealing and interactive statistical analysis. It can also be used for filtering and summarizing data, which can be useful for creating meaningful visualizations in R Shiny.

6. **'dplyr':** This package provides a set of functions for data manipulation and transformation. Also, it offers a concise and persistent syntax for performing common data manipulation tasks.

7. **'stringr':** This package offers a set of functions for working with strings. It provides a consistent and intuitive syntax for performing various string manipulation tasks.

8. **'wordcloud':** This package offers various functions for creating word clouds. It helps to create word frequency calculation as well. It can also be set in different shapes, such as circle, square, etc. It can be used to preprocess text, topic modeling, perform sentiment analysis as well.

9. **'plotrix':** This package provides a set of functions for creating plots and charts. It can be used to lot 3D plots, polar plots, bar charts and many more.Additionally, there are many other customizations available as well which can be added to the specific need.

### 4.4 RESTful API

In our project we have used REST APIs for communication between web-based applications and couchDB clusters deployed on the cloud. The data that is being fetched is real-time in case of mastodon toots and REST APIs provide an easy way to communicate with the cluster and fetch data. The package "httr" is used for making a GET request to the REST API. In the endpoint it takes some parameters as well which define which database, design and view it needs to fetch data from. The data is received in JSON format. Error handling is also performed in case any of the couchDB server accidentally stops or crashes. If one IP address fails to execute it checks for the other mentioned IPs and try to connect to them and retrieves the data successfully. The data fetched was then used for data visualizations, analysis between different parameters and calculations.

## 5 Data Pre-processing and Analysis

### 5.1 Pre-processing

We were given a tweets file called "twitter-huge.json" that had a size of 63.2 gigabytes and contained tweets sent within Australia. From this file, we removed tweets that included locations and saved them in a file called "twitter-place-data.json," which helped reduce the amount of data to 4.5 gigabytes. After that, we processed this data and changed it into a new JSON file with a structure. Place name : Year : Month : Date : [[tweet,spacy token],....]] before saving it to the couchDB server.
Similarly while collecting toots, unnecessary information is removed and only fields of date, text, id, profanity are kept. This way we only store important information and unimportant data does not occupy the memory.

**MapReduce**
Several views are created to support our scenarios and get only the data that is needed. Using python or doing data processing at the backend would be slow, couchDB's views are very fast to query thus they are used. Views like (GCC,profanity), (Date, sentiment) etc. are queried at different levels to get relevant data fast. Query examples are given in queries.txt file.

### 5.2 Scenarios

For data analysis aspect we considered 2 scenarios, which are mentioned as follows.

1. Covid Analysis

2. Profanity Analysis

3. Mastodon toots analysis

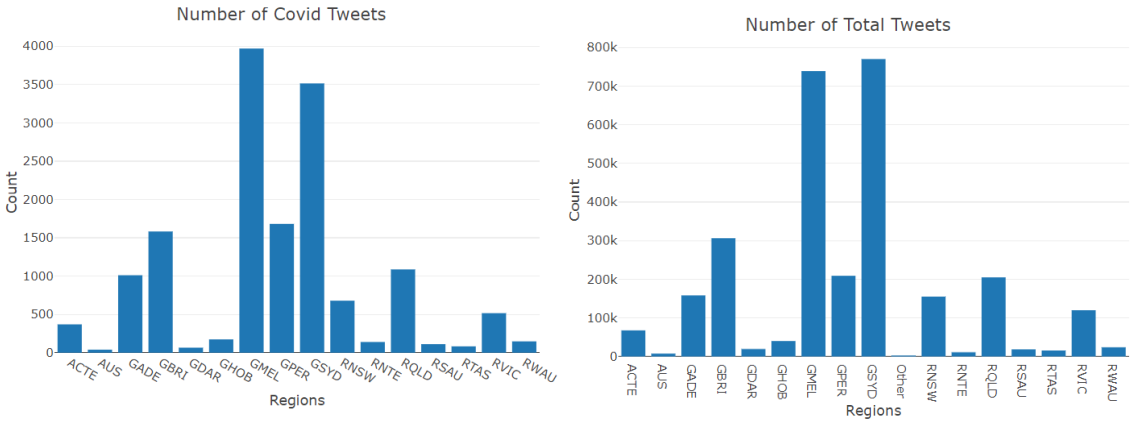
The scenarios and the analysis for each scenario is explained in further sections.


(a) Example of tweet data (b) Example of a query output

Figure 3: Bar graph displaying the number of tweets by region

### 5.2.1 Covid Analysis

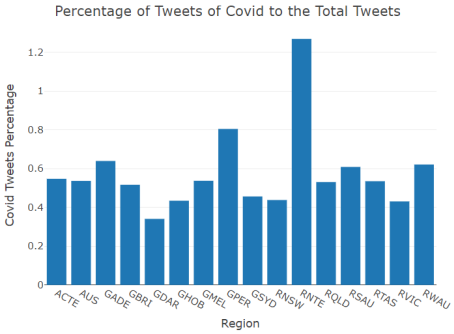
In this section we elaborate on analysis we did on the twitter data related covid. Covid has been a hot topic since 2020 due to the havoc it wreaked in past few years, thus making it an ideal topic for analysing how people react to the issue as well as what opinions different sections of the society have towards pandemic.



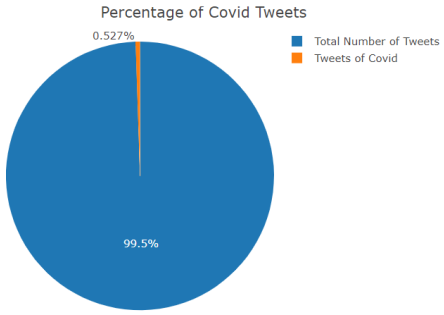

(a) Number of Covid Tweets (b) Total number of tweets

Figure 4: Bar graph displaying the number of tweets by region

Figure 4(a) represents a bar graph that shows the number of tweets related to covid from every region in Australia, in and around the cities and inside the regions like Greater Melbourne (GMEL), Rural New South Wales (RNSW). Similarly in Figure 4(b) we can see total number of tweets made from different regions across Australia. From the given graphs we can observe that most number of tweets about covid was made in Greater Melbourne Region.



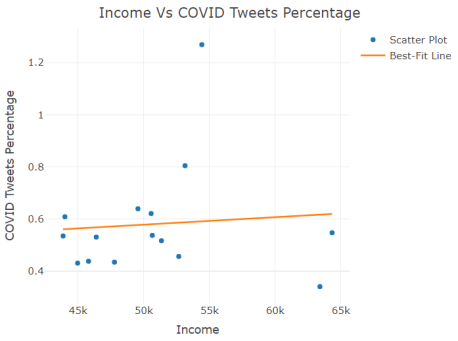

(a) Percentage of tweets made region wise



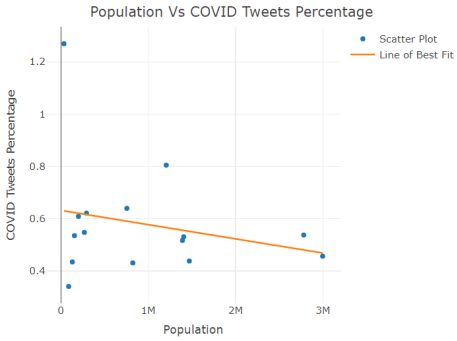

(b) Pie chart showing percentage of tweets made related to covid

Figure 5: Percentage comparisons of covid related tweets

In Figure 5(a) we can see a bar graph illustrating the percentage of tweets made related to covid split region wise. From the given graph we can observe that most number of tweets were made from Rural Northern Territory which accounts for about 1.2 percent. The pie chart in Figure 5 (b) showcases the percentage of tweets made regarding covid out of total number of tweets in data which barely accounts for 0.5 percent. Thus we can conclude as the covid cases decreased number of tweets on the tropic also dropped significantly ove a period of time.




(a) Income Vs Covid Tweets by percentage




(b) Population vs Covid Tweets by percentage

Figure 6: Scatter Plot with best fit lines by region

Figure 6 (a) showcases a scatter plot of percentage of covid tweets made by the different economic section of the society with a best fit line. From this graph we can concur that people between the pay bracket of 50 thousand to 55 thousand have tweeted more regarding the pandemic. This section of the society account for the middle class population in the country.

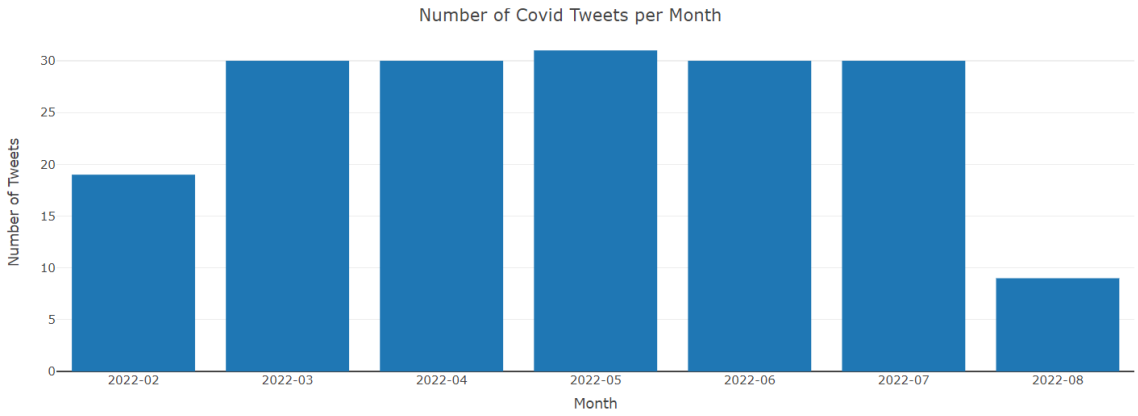
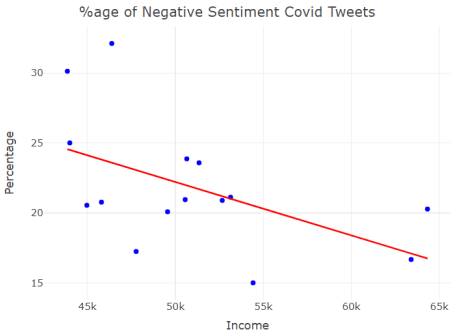
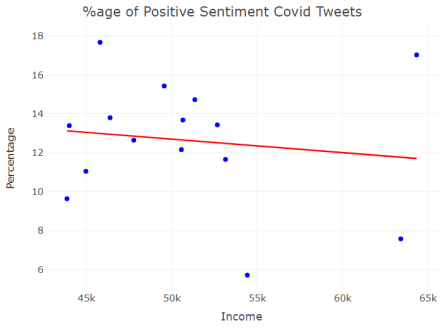



Figure 7: Number of covid tweets per month

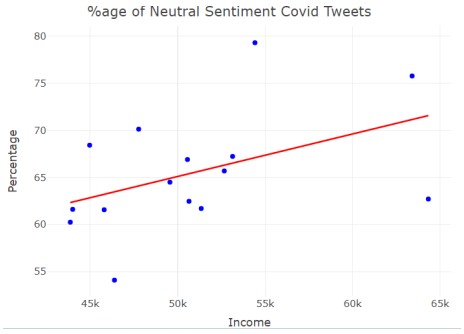
In figure 7 we can see number of covid related tweets made per month for a period of 7 months from February 2022 to August 2022. Here we can observe the number of tweets increasing gradually from February and reaching the highest in the month of May and then eventually decreasing with a sudden decrease in the month of August.




(a) Percentage of negative sentiments




(b) Percentage of positive sentiments




(c) Percentage of neutral sentiments

Figure 8: Sentiment Analysis of covid via scatter plot

Figures 8(a),(b) & (c) illustrates percentage of negative, positive and neutral tweets respectively made by different economic sections of the society. From these scatter plots we can infer that people who belong to lower economic sections in various regions tend to have more negative sentiment towards covid, while the wealthier section of society tends to hold a neutral opinion about the pandemic situation.

### 5.2.2 Profanity Analysis

The next scenario which we analysed was profanity in tweets made by people. Profanity stands for obscene gestures or swear words used by people while tweeting. We have used various parameters to analyse the profanity in tweets and the same have been elaborated below.

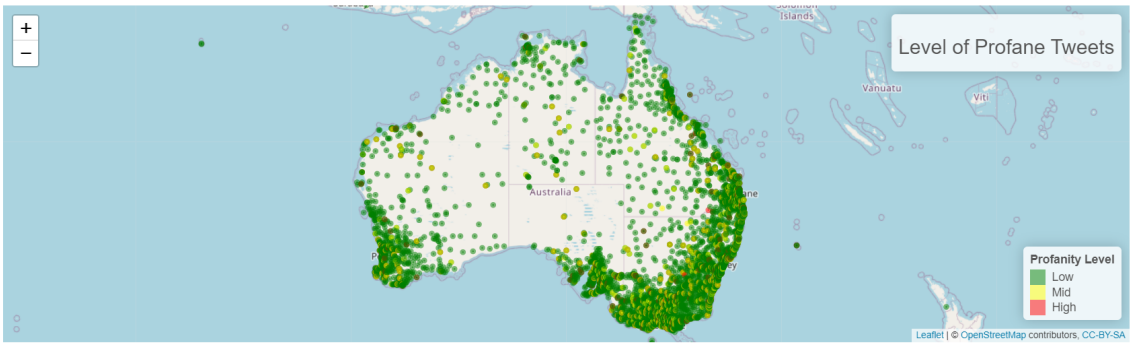
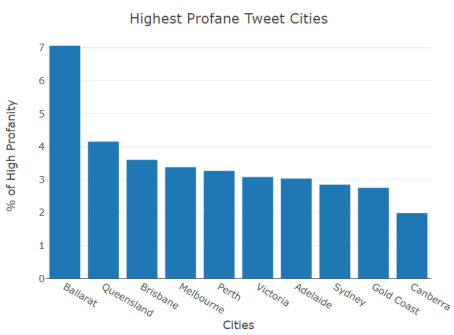
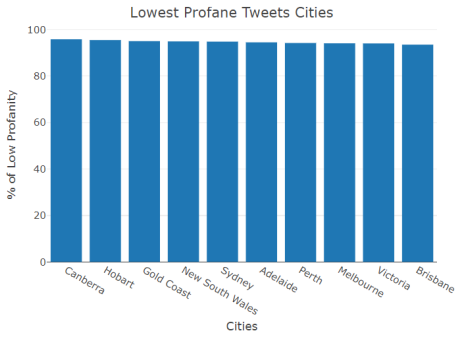



Figure 9: Levels of profanity in tweets

Figure 9 is plotted using leaflet library and it displays the profanity of tweets in different regions of Australia. Furthermore, there is a scale that indicates the color coding according to the profanity level at the bottom right corner of the map. From the map we can see that mostly profanity is low generally throughout the continent but most of the profane tweets are made in the western part of Australia.



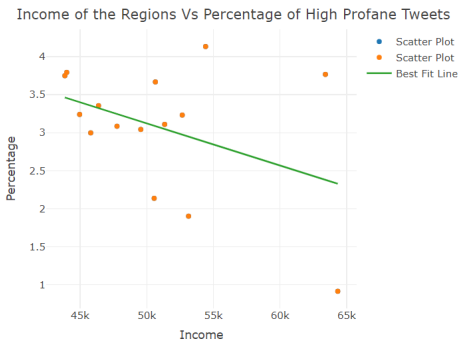

(a) Percentage of highest profane tweets



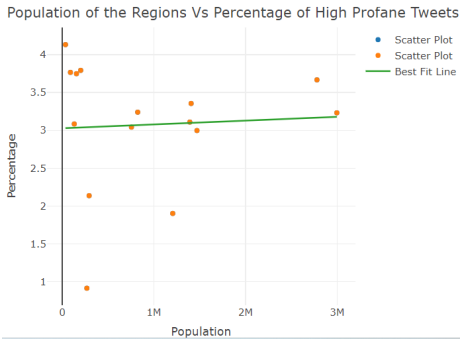

(b) Percentage of lowest profane tweets

Figure 10: Profanity analysis in percentage

Graphs in figure 10 (a) and (b) depicts the regions that have the highest and lowest profane tweets percentage. The profanity of the specific region is calculated from the high/low profanity tweets of that area to the total tweets of that region. We can observe that highest profane tweets are made from Ballarat.




(a) High profane tweet vs income of that region




(b) High profane tweet vs population of that region

Figure 11: Scatter plots for profanity analysis

Here figure 11 (a) has plotted a scatter plot of income of a specific region to the high profane tweets of that region. When we hover on the point in the scatter plot it shows the region name, average income value and the percentage of high profane tweets of that area. Also, a best fit line is plotted which tells us the trend in the plot and from here we can observe that as the income of a region increases the profanity in the tweets of that region decreases. Similarly, figure 12 (b) shows a scatter plot that depicts the population of a region to the percentage of high profane tweets of a specific region. The dots on the scatter plot represent different regions. Here also, a best fit line is plotted which tells us about the trend in the data. There is not much of a difference according to the best fit line in the trend with population.

### 5.2.3 Mastodon Toots Analysis

In this section of our web application we analyse the live toots we get from mastodon server and display hot words, or the most used words in the toots using a word map. We also plot a bar graph showing how many toots contain these hot words.

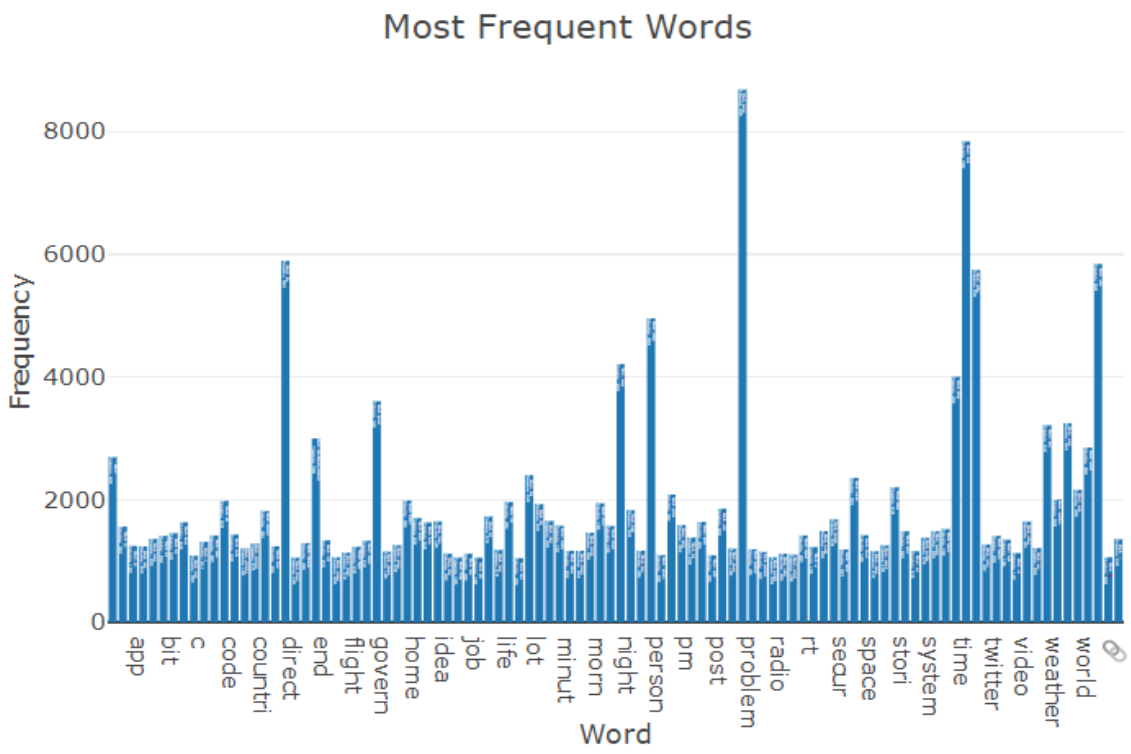



Figure 12: Most Frequent Words

Figure 12 illustrates a graph that shows the most frequent words after analysis of the data. As the size of the graph is small it cannot display every word name on the x-axis of the graph but when we hover over the spikes of the graph it tells us about the word name, number of times it appeared in the data. This is plot of top 100 common words not the all of them in mastodon toots.


Figure 13: Word Map

Figure 13 displays a word map that shows us the most frequent word in the overall mastodon toots and

all of this has a dynamic working if any word occurrence count increases it will be updated and displayed according to its frequency. To keep it clean we have taken the top 100 common words, not all of them to plot on word cloud.

# 6 Error Handling

**Sentiment analysis:** Sentiment analysis may not be accurate for some data. It is very difficult to detect sarcasm.
**CouchDB setup:** For creating couchDB clusters inside containers opening ports(5984, 4369, 9100+) to the machine and other VMs is important.
**Region wise data**: Getting the region code using place's name has some issues, there are a lot places in tweets like Victoria that are outside Australia, like in Britain. Filtering such places is difficult, they are few in numbers and have no effect on analysis so no such cases are covered.
**Frontend:**Error handling is also performed in case any of the couchDB server accidentally stops or crashes. If one IP address fails to execute it checks for the other mentioned IPs and try to connect to them and retrieves the data successfully
As the mastodon data is added on the couchDB server, views don't change unless a query is called, so querying the view continuously at some intervals is necessary.
**Mastodon toots:**As most frequent word analysis is used, all toots are passed through a language detector. If they are not in English they are passed over and not included in the database.

# 7 Contributions

| Name | Contribution |
|---|---|
| Kunal Patel | CouchDB Setup, Docker, View-creation, Data Preprocessing |
| Mayank Yadav | Data Preprocessing & Analysis, Frontend development , Latex administrator |
| Harsh Mangla | Frontend application development, Data Processing, Video Demonstration |
| Sophie von Doussa | Mastodon harvester, Toot processor, Ansible, Docker, Video Demonstration |
| Maxson Stephen Mathew | Frontend application development, Latex Administration, Video Demonstration |

# 8 References


1. UniMelb Research Cloud documentation: https://docs.cloud.unimelb.edu.au/
2. Leaflet documentation for R Shiny web application: https://rstudio.github.io/leaflet/