Report for CCC Assignment 2

Team 24

Jiyuan Chen

1323889

Jiyuanc1@student.unimelb.edu.au

Jiakang Li

1287117

jiakangl@student.unimelb.edu.au

Xueqing Li

1340489

xueqingl4@student.unimelb.edu.au

Yuqing Li

1042408

yuqing5@student.unimelb.edu.au

Kejie Xu

1342103

kejiex@student.unimelb.edu.au

2023/05/21

Abstract

This is the report for the COMP90024 Group 24 project. In this project, we aim to develop a cloud-based solution for analyzing Twitter data related to user behavior and industries. In Section 2, we will provide a brief guide on using the system and explain the core functionalities of our scripts, as well as the reasons for choosing this solution. Section 3 will provide documentation on the system architecture and design. Section 4 will describe the methods for data filtering, preprocessing, and sentiment analysis. In Section 5, we will present the scenarios describing the selection of system functionalities and graphical results. Section 6 will discuss the problems and challenges we encountered in this project and the solutions we implemented. In Section 7, we will provide links to the deployment demonstration and frontend video. Section 8 will include a link to the GitHub repository where our code is hosted. Section 9 will outline the minutes of team meetings, and Section 10 will detail the contributions of each team member to the project

Content

1 Introduction

Twitter has become an indispensable part of our daily lives and is considered one of the influential social media platforms. It has transformed into a powerful force that impacts our society, politics, and business environment. In our research, we focus on exploring the relationship between each industry and Twitter user behavior. By studying Twitter user behavior within specific industries, we aim to gain insights into how users interact, engage, and contribute within their respective fields of interest. This analysis allows us to understand the dynamics of user participation and its relevance to different industry sectors. Such knowledge can provide valuable information for businesses, policymakers, and researchers seeking to leverage the power of Twitter and understand its impact on various domains.

Through our research, we aim to uncover patterns, trends, and correlations that shed light on the relationship between industries and Twitter user behavior. This exploration can contribute to a deeper understanding of how social media platforms like Twitter influence and reflect the activities and interests of individuals within different sectors.

To accomplish this, we utilized Twitter's dataset and two crucial datasets provided by Sudo [1]: "sudo_industry_type" and "sudo_industry_worktime," along with Mastodon data. On our web interface, we designed various scenarios and employed visualization techniques to gradually delve into the details, starting from the broader context of Twitter and narrowing down to specific insights derived from the Sudo datasets. This approach enables a better understanding of the data and reveals the level of user engagement in Australia and the impact of different industries on Twitter. Additionally, we employed NLP (Natural Language Processing) techniques to perform sentiment analysis on all the collected data. This allowed us to gain insights into the emotional tone and sentiment expressed by users within different industries, further enriching our analysis.

Furthermore, we leveraged Docker and Ansible for project deployment. Each component of our project was encapsulated and packaged into containers, allowing us to bundle the application and its dependencies as a standalone executable unit. This approach ensures that our project can run consistently in different environments, promoting portability and ease of deployment.

By combining data analysis, visualization, sentiment analysis, and containerization techniques, our project provides a comprehensive exploration of user engagement and industry impact on Twitter. It offers valuable insights into the dynamics of social media behavior and its relationship with various industries, contributing to a better understanding of the role Twitter plays in our digital landscape.

[1] Sudo data from https://sudo.eresearch.unimelb.edu.au/

2 User Guide

Our project will be developed and used based on the Melbourne Research Cloud. The process and implementation of the project will be described below.

2.1 localhost instance Deploy and Use

First, we will manually create an instance in the Melbourne Research Cloud (MRC) and set up its source, flavour, networks and security groups for subsequent external access and use. The key pair will also need to be set up. After the instance has been set up and successfully started, connect to the instance via external ssh and the generated key.

When the instance has been started, the ansible dependencies should be downloaded:

sudo apt-get update
sudo apt-get install software-properties-common
sudo apt-get install ansible

After that, we need to manually download the ansible file from GitHub. The Github link about ansible is described in Chapter 8. Place the ansible file in the */project* location in the instance. The file structure of ansible is described in Table 1. In the ansible file, *openrc.sh* is obtained from the OpenStack file in MRC, which records user information and the configuration of the relevant MRC system, enabling us to access MRC in instance for the task of automatically deploying instances. *cloud_key.pem* is the instance key, which facilitates the establishment of communication between instances. Finally, there are several ansible auto-deploy scripts and .sh scripts that will detail how to deploy our system in the current instance in the following sections.

localhost: /project/ansible	**description**
host_vars/config.yaml	variable value
inventory/hosts.ini	localhost address option
roles/	deploy roles (docker, backend, frontend, git, harvester)
ansible-backend.sh	setup backend
ansible-frontend.sh	setup frontend
ansible-harvester.sh	setup harvester
cloud_key.pem	ssh key
openrc.sh	MRC openrc script
run.sh	setup whole system
setup-backend.yaml	backend yaml script
setup-frontend.yaml	frontend yaml script
setup-harvester.yaml	harvester yaml script
setup.yaml	whole system setup script

Table 1 Ansible files

2.2 Backend Deploy and Use

When deploying the back-end environment, the following commands need to be executed:

./ansible-backend.sh

The command will run *setup-backend.yaml*. ansible will require the command about installing Python, pip, openstacksdk and other relevant dependencies. The backend repository will be fetched from GitHub, the GitHub link about the backend is described in Chapter 8. The backend files will be generated in the default file path. In *roles/deploy-docker*, the docker and associated repository will be downloaded, and a *my-network* will be created to facilitate requests and fetches between the backend and frontend later. After the docker has been downloaded and installed, in *roles/deploy-backend*, it will first determine whether the backend's container already exists, and then delete and rebuild it to prevent duplicate docker builds. Finally, create the docker image in the specified path, run the container, and add *my-network*.

This is backend's URL: http://172.26.133.42:8000

2.3 Frontend Deploy and Use

When deploying the back-end environment, the following commands need to be executed:

./ansible-frontend.sh

The command will run *setup-frontend.yaml*. The logic of the ansible script is essentially the same as the Backend deployment, pulling the corresponding repository from GitHub, resetting docker and deploying the frontend to docker. Finally, add frontend to the *my-network* as well. The GitHub link about the frontend is described in Chapter 8.

This is frontend URL: http://172.26.133.42:3000

2.4 CouchDB Deploy and Use

For CouchDB, a new instance is created for deployment. The method of creating this instance is the same as that of the front and back instances. In MRC, we created a 30G volume and mounted it on the instance to prevent the data in CouchDB from running out of the instance's storage space.

In addition, with the latest version of CouchDB, the username and password needs to be set in */var/snap/couchdb/current/etc/local.ini* on CouchDB. On our system, the username and password are *user* and *pwd* respectively.

This is CouchDB URL: http://172.26.134.204:5984

2.5 Harvester Deploy and Use

For the harvester task, we will create an instance automatically. The harvester task needs to constantly fetch new data in the Mastodon servers and store it in CouchDB, which is relatively independent compared to other tasks and requires constant interaction with external servers. The harvester task is relatively independent compared to the other tasks and needs to interact with external servers constantly, so it is done by dynamically creating instances.

When deploying the back-end environment, the following commands need to be executed:

./ansible-harvester.sh

The command will run *setup-harvester.yaml*. The ansible script will fetch the harvester repository from GitHub, the GitHub link about the harvester is described in Chapter 8. The ansible script will then use the OpenStack script file to get the MRC-related configuration. In the *deploy-harvester-instance* file, the new instance will be configured, including its flavor, image and security groups, and will then wait for the new instance to be created. To connect to the new instance, you need to use the *cloud_key.pem* file, which is the key to connecting to the new instance. When created successfully, the host of the new instance will be created and remembered in localhost. At this point, the new instance is successfully running and in setup-*harvester.yaml*, the hosts will be switched from localhost to *harvester_instance* to run harvester-related tasks.

2.6 Automatic Deploy

On the current localhost, we also provide the following command:

./run.sh

The command will execute *setup.yaml*, the ansible script that does all of the above directly:

1. Install the relevant dependencies
2. Clone the GitHub repository
3. Build dockers
4. Configure front-end and back-end dockers
5. Create a new instance
6. Execute the harvester task in the new instance

3 System Design and Architecture

In this section we will show the design and structure of the system and explain some of the reasons for the design and the process.

3.1 System Design Structure

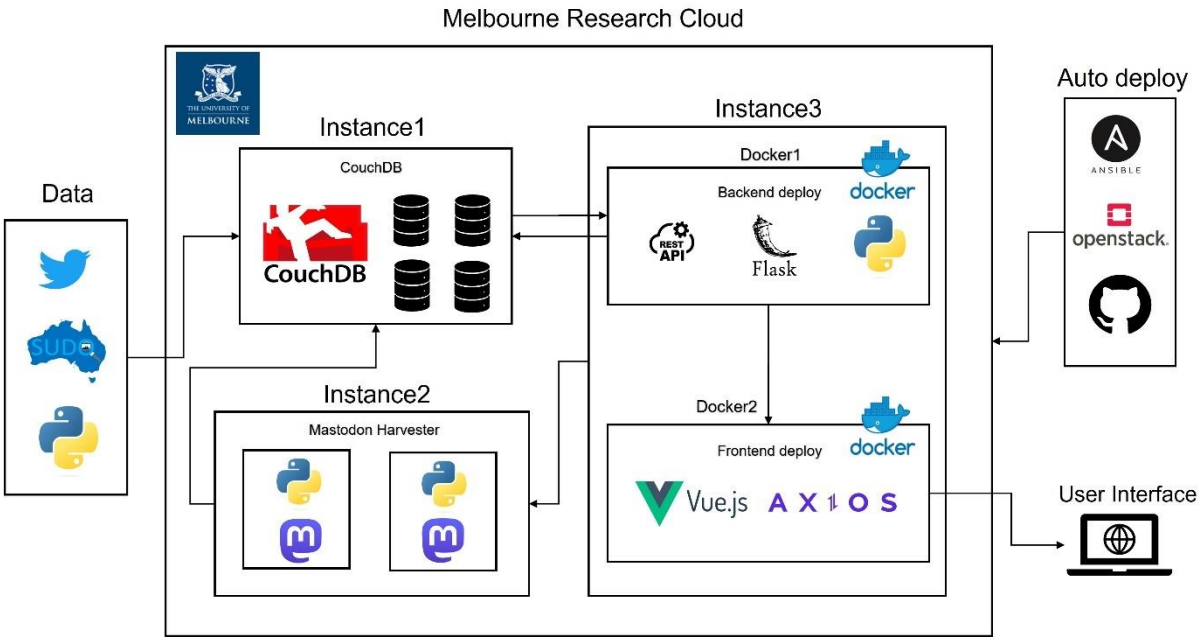

Figure 1 - System Architecture Diagram

The architecture diagram of our entire system is shown in Figure 1. The entire system will rely on the three instances in MRC. The configuration is as follows:

1. *availability_zone: melbourne-qh2-uom*
2. *network: qh2-uom-internal*
3. *flavor: uom.mse.2c9g*
4. *image name: NeCTAR Ubuntu 22.04 LTS (Jammy) amd64*
5. *key pair: cloud_key.pem*
6. *security groups: couchdb_port, http, rdp, ssh*

In Security Groups, *couchdb_port* sets the new ingress direction, exposing ports 8000,6984,3000, corresponding to the backend, CouchDB and frontend ports needed for subsequent deployments. Then we present the design principle of our entire system:

Instance 1: This instance is used by our CouchDB component, which is a NoSQL database used to hold Document type data. Data is a very important and sensitive part of any scenario, so in the system, the CouchDB component is deployed in a separate instance to ensure the persistence and availability of data when other parts of the system fail, thus reducing the risk of the system and satisfying the single responsibility principle.

Instance 2: This instance is used by our Mastodon harvester component, which constantly gets the data it needs from the Mastodon servers and stores it in CouchDB. And it has a high demand on the instance load and is a separate component from the others. When there is a problem with the harvester component, there is no impact on the whole system.

Instance 3: This instance contains our backend and frontend components. The backend will use Flask and the REST API to get the CouchDB data, while the frontend will use Vue to build the frontend interface and use the Axios method to get the data processed by the backend. As the backend and frontend components are strongly related, deploying them in the same instance will simplify the complexity of deployment and management. However, the backend and frontend may have conflicting dependencies and setting up two dockers allows for their independence and isolation. Each container can have its independent

runtime environment, dependencies and configuration, allowing the two projects to be developed, tested and deployed independently without interfering with each other.

3.2 UniMelb Research Cloud Deployment Details

The Melbourne Research Cloud (MRC) operates on a private cloud deployment model, providing users with free and on-demand computing resources. As an Infrastructure-as-a-Service (IaaS) it offers essential IT infrastructure components such as computers, networking, and storage. Within this environment, a subset of OpenStack services is available to meet the diverse needs of researchers and developers. The MRC empowers users with the flexibility to utilize these resources as required for their specific computing and research requirements.

The Melbourne Research Cloud comprises almost 20,000 virtual cores, available across a range of virtual machine sizes. It also includes specialist resources such as GPGPUs, private networking, load balancing, and DNS[2].

3.2.1 Resource Allocation

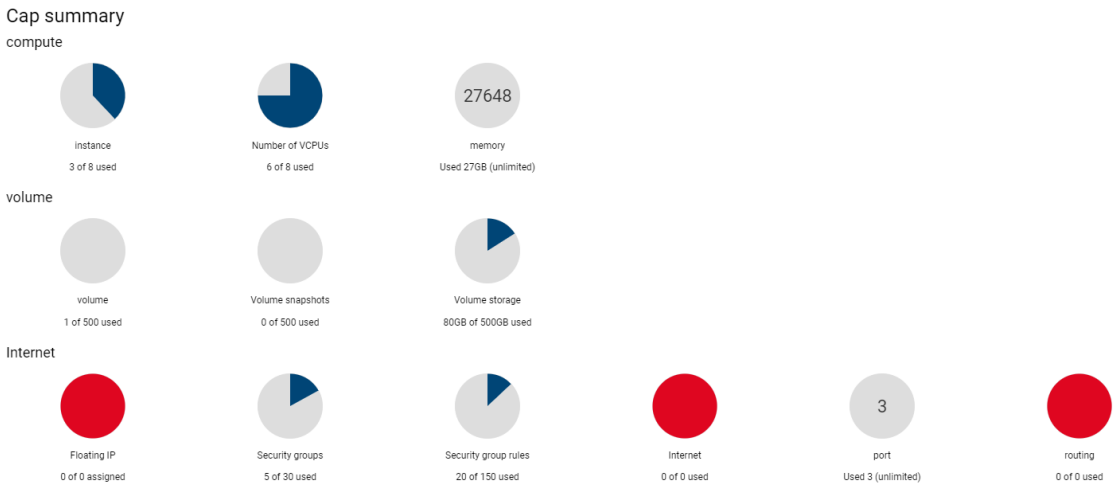

Figure 2 - Resource Allocation

In our deployment on the Melbourne Research Cloud, we have allocated three instances with specific configurations. The first instance, called "couchdb_instance," is responsible for hosting CouchDB and storing all our data. The second instance, named "frontend_and_backend_instance," serves as the frontend and backend for our application. Finally, the third instance, "harvester_instance," is used for running the data harvester.

To ensure the security of our instances, we have customized our own security groups in addition to the default and SSH security groups. While the default security group allows outgoing connections from the instance and the SSH security group opens port 22 for SSH connections, we have customized our security groups to allow incoming connections on specific ports. Port 3000 is open for the frontend, port 5984 for CouchDB, and port 8000 for the backend.

For the operating system image, we have selected the NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 version. We chose this image because we are familiar with the Ubuntu environment, and 22.04 is the latest version of Ubuntu available. It should be noted that we did not choose an image with Docker pre-installed because the Docker version in those images is outdated.

In addition to the instance configurations, we have also added a volume to the "couchdb_instance." This volume allows us to store and manage our data effectively within the instance, providing the necessary storage capacity for our application.

[2] These words are from Melbourne Research Cloud Documentation (unimelb.edu.au)

By configuring our instances, customizing security groups, and selecting the appropriate operating system image, we have established a robust and secure environment for our application on the Melbourne Research Cloud.

3.2.2 Advantages

Cost:

As the MRC is free service to all UniMelb researchers, there are no cost for us to use MRC. By utilizing the MRC, we can access computing resources without incurring additional costs typically associated with commercial platforms like Amazon AWS or other similar services.

Security:

The Secure Shell (SSH) protocol is commonly used to establish secure communication with instance deployed on the MRC. To utilize SSH, we generate an SSH key pair consisting of a public key and a private key.

The public key is configured on the instance, allowing it to authenticate and establish a secure connection with us. On the other hand, we securely hold the private key, which is used to authenticate ourselves and decrypt the data transmitted between us and the instance.

By employing this approach, all data communication between our machine and the instance is fully encrypted. This ensures the confidentiality and integrity of the exchanged information. Only individuals possessing the private key are capable of decrypting and accessing the data, providing an added layer of security. This utilization of SSH and key-based authentication enhances the overall security posture of the communication between us and the instance, safeguarding against unauthorized access or interception.

Furthermore, by selectively opening only the required ports, we have complete control over traffic access to our instances. This approach significantly reduces the risk of unauthorized access by potential hackers. By carefully managing and limiting the open ports, we can strengthen the security measures in place and mitigate the chances of security breaches or unauthorized intrusions. This proactive approach to network security helps safeguard the integrity and confidentiality of the data and resources hosted within the MRC.

In addition, the utilization of private IP addresses contributes to enhancing the security of the instance deployed on the MRC. By employing private IP addresses, the instance is shielded from direct public access, reducing its exposure to potential security risks.

To access the instance, a VPN (Virtual Private Network) service is required. This adds an extra layer of security by establishing an encrypted connection between the user's device and the MRC infrastructure. By leveraging the VPN service, authorized users can securely access the instance, ensuring that communication and data transmission remain protected from unauthorized interception or malicious activities.

Overall, the combination of private IP addresses and VPN usage enhances the security posture of the instance, providing a secure and controlled environment for accessing and managing resources on the MRC.

Recovery:

The Melbourne Research Cloud (MRC) offers us the capability to select the popular image and create snapshots of our cloud instances and provides efficient recovery options. Creating snapshots serves as a backup mechanism, allowing us to restore our instance to a previous state if necessary. In the event of system failures, data corruption, or other unforeseen issues, we can rely on the MRC's snapshot functionality to quickly recover and restore our instance to a known working state.

Overall, the MRC's snapshot and recovery capabilities enhance the resilience and reliability of our cloud instances. By leveraging this feature, we can confidently deploy applications and services, knowing that we have a reliable backup and recovery solution in place.

3.2.3 Disadvantages

Setting up and using the Melbourne Research Cloud (MRC) may require some initial knowledge about cloud computing. While there are documentation and resources available to assist users, it can still be

challenging for newcomers to understand the process and functionalities. Learning how to properly configure and utilize the MRC may involve a significant investment of time and effort.

Additionally, accessing the MRC can be cumbersome as it requires the installation of a VPN and logging in to the UniMelb network. The security measures in place, such as using private keys for authentication, are designed to enhance instance security but may add complexity to the access process.

The web interface of the MRC may not be intuitive for users, making it difficult to locate specific features or options. For example, when setting up automatic deployment options, finding the "reset password" button for OpenStack can be challenging as the documentation may not provide clear instructions. Only those who have attended classes might be familiar with the location of this button, leaving newcomers unaware of how to reset their password.

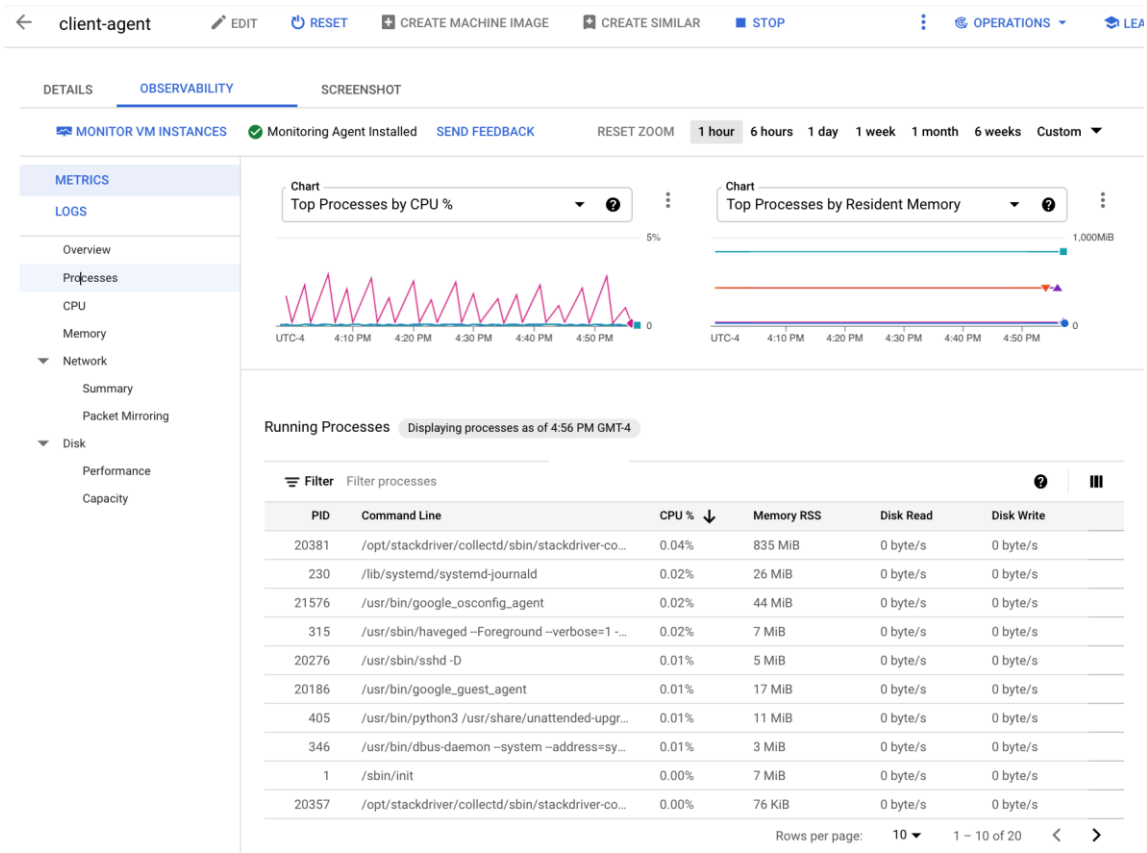

Figure 3 - Google Cloud Instance Monitor

Furthermore, the MRC lacks a real-time monitoring feature for individual instances in the dashboard like shown in Figure 3[3]. This means that users do not have a convenient way to monitor the current resource usage or occupancy of their instances. This can be inconvenient compared to other platforms like Google Cloud or similar services that provide real-time monitoring capabilities.

3.3 Backend Details

The backend refers to the part of an application that is responsible for handling business logic and data processing. It plays a crucial role between the frontend and the database. The frontend is responsible for displaying data and interacting with users, sending requests to the backend and receiving responses. Communication between the frontend and backend is done through network protocols. The database is a system used for storing and managing data. It is used to persist the application's data for subsequent read, update, and delete operations. The backend is responsible for interacting with the database, executing queries and update operations, retrieving data from the database, and processing it.

[3] Image from https://cloud.google.com/monitoring/agent/process-metrics

In our project, I used the Flask framework for the backend. Two files were utilized: "couchDButi.py" for connecting to the database and "app.py" for handling frontend requests and returning database data to the frontend. Flask's built-in interface design is based on the RESTful API style.

In "couchDButi.py," I set up the username and password for the database and integrated five datasets using the "tablename" method. Since our map-reduce operations were already performed within the database, no further data processing was required on the backend.

In the "app.py" file, The Flask route decorator, @app.route(), maps the requested URL to its corresponding handling function. By specifying the HTTP methods within the route decorator, such as methods=['GET', 'POST'], you can restrict the allowed request methods for the API, aligning with the method conventions of RESTful APIs.URL specification and resource naming: Flask supports the use of variables and parameters to construct dynamic URLs, enabling the inclusion of a resource's unique identifier as part of the URL. By utilizing suitable URL naming and conventions, API endpoints become more intuitive and easier to comprehend. I defined GET and POST requests. GET requests pass data through URL parameters as key-value pairs. Since no further processing of the data was required, I used the GET method for HTTP requests. GET requests are more suitable for data retrieval operations and not intended for modifying server-side data. I processed the data retrieval for the five views in the "twitter_all_new" dataset, ten views in the "industry_twitter_new" dataset, two views in the "sudo_industry_type" dataset, two views in the "sudo_industry_worktime" dataset, and two views in the "mastodon_all_servers" dataset.

By connecting to the database and running the application, data was successfully returned to the web pages when accessed through a browser. In conclusion, the backend acts as a bridge between the frontend and the database, handling data processing, and facilitating communication between the two.

3.4 Frontend Details

Our front-end use Vue.js application consists of five distinct pages, all of which interact with APIs to provide data-driven outputs represented as interactive ECharts.

Home Page: main entry page of the application, introducing our group and project, "Analysis of the Impact of Twitter Users and Industry in Australia". The page displays a brief introduction and two related images, found in the "src" directory.

Twitter Page: The Twitter page fetches and represents Twitter data in various categories. Users can view charts depicting the number of tweets based on city, language, and month. The page offers an interaction model that clicking on buttons triggers specific data fetches, which are then represented in chart form.

Jobs Page: The Twitter Job page fetches and represents Twitter data in four types of jobs. Users can view charts depicting the number and emotions of tweets in technology, art, education, finance area. The page also offers an interaction model that clicking on buttons triggers specific data fetches.

SUDO Page: It provides an overview of SUDO jobs' distribution based on categories like job type, number of jobs per city, gender distribution, and work time. Four different charts represent these aspects. User interaction, such as clicking on buttons, triggers the data fetch from the server, and the corresponding chart is updated accordingly.

Mastodon Page: This page focuses on data from Mastodon, a decentralized social network. It displays the monthly count of posts and their sentiment polarity. The data is represented in an interactive bar chart that updates based on user-triggered actions.

Throughout the development of these pages, various technologies and techniques were employed:

Bootstrap: This was used to enhance the UI/UX of the application, providing a more user-friendly and responsive design.

Axios: Axios was employed to perform HTTP requests from the frontend to the backend server. Axios's promise-based structure is advantageous when dealing with asynchronous operations, which were required for fetching data from the server without blocking the UI.

Vue.js Lifecycle Hooks: Particularly, the mounted lifecycle hook was used. This hook executes after the Vue instance has been created, and in this application, it was used to call the Axios functions to fetch the initial data.

Async/Await: JavaScript's async/await syntax was used to simplify the process of working with promises, improving the readability and maintainability of the code. For instance, the await keyword was used to pause the execution of the async function, waiting for the Promise (Axios request) to resolve before moving on to the next line of code.

3.5 Harvester Details

We use Message Passing Interface for Python (mpi4py) to enable parallel execution of multiple Mastodon Harvesters. Each harvester processor is responsible for crawling data from a single Mastodon server, and all processors store the collected data into a shared CouchDB dataset. Communication between the harvester processors is not necessary for our specific requirements.

Each harvester processor operates independently, continuously fetching real-time data and processing it promptly. The data undergoes several stages of processing. Firstly, we filter the content by searching for specific keywords within the Mastodon posts. Next, we perform sentiment analysis on the filtered content. Finally, we extract the relevant information, including the month, content, and sentiment analysis results, which are then stored in the CouchDB dataset.

This approach allows for efficient and parallel processing of the Mastodon data, enabling us to gather valuable insights and analysis from multiple sources simultaneously.

To customize the harvester configuration, you can edit the global constant variables MASTODON_ACCESS_TOKENS and SERVERS_URLS. These variables allow you to add your own access tokens and the corresponding URLs of the Mastodon servers you want to crawl. It's important to note that the number of harvester processes should align with the number of servers you want to crawl.

3.6 CouchDB Design

Our system uses a single node CouchDB as our database. It is able to store processed tweet, sudo and mastodon data easily and securely. In addition, it can generate views directly using the MapReduce function to help us with scenario analysis.

3.6.1 Advantages of Couchdb

Firstly, CouchDB uses a document-based data model that allows you to organise and store data in a free-form format. Each document is a complete, independent unit of data that can be represented using the JSON format. This adapts perfectly to the format of the Twitter dataset, making the data model very flexible and easy to understand and manipulate.

Secondly, CouchDB supports MapReduce-based query and indexing mechanisms. You can define Map functions and Reduce functions to create views that can be used to quickly retrieve and aggregate data. This type of querying makes CouchDB ideal for complex data analysis and report generation.

Thirdly, CouchDB's official administration interface, Fauxton, provides an intuitive, easy-to-use interface with no additional development and cross-platform support. With Fauxton, you can intuitively view and edit documents, execute queries, create views and see output as well as manage database and security settings, further enhancing ease of administration and operation.

Finally, CouchDB has the ability to work offline and synchronise data. It can cache data locally on the client and continue to work when offline or disconnected from the network. Once the connection is restored, it can automatically synchronise local changes with the remote database, ensuring data consistency.

In summary, by choosing Couchdb as our database, our system can quickly design and query MapReduce views and facilitate the integration of different programming languages and frameworks, features that make CouchDB our top choice for handling our data needs.

3.6.2 Data upload in CouchDB

For the processed tweet dataset to be stored locally, the transfer time required for the large amount of data is long, so an iterative approach is used to speed up the transfer of data. First, a connection to the CouchDB instance is established via the couchdb module. Then, specifying the name of the database into which the data is to be imported, the ijson module is used to read the data row by row in an iterative fashion. For each JSON object read, the necessary data processing is performed. The processed JSON objects are then

added to the bulk document list, and a counter is added. When the number of documents in the bulk document list reaches a certain threshold, the db.update() method is used to save the bulk documents to the database.

The above steps are performed using the Python API provided by CouchDB to transfer the data. Data is saved to CouchDB in bulk to increase efficiency and reduce communication overhead. The data transfer and import process is completed by reading and processing the JSON data row by row and then saving the bulk documents to the database using the db.update() method. The same import method is used for Sudo data.

Mastodon differs slightly from Twitter and Sudo in terms of data storage in CouchDB. With our real-time Mastodon harvester, we process and upload the data as soon as it becomes available. This ensures that the information is promptly accessible for analysis and further utilization.

3.6.3 Database MapReduce

CouchDB provides the MapReduce function function for creating views. The views allow us to filter and consolidate tweet data and perform various calculations on it, such as summing and averaging. Depending on the scenario, we created multiple views, each extracting specific data fields of interest to us from the tweets. Using Flask on the back end, the view data is fetched from CouchDB and processed and analysed accordingly to the business requirements.

3.7 Docker

Both docker and Virtual Machine (VM) are two virtualization technologies that can create multiple virtual logical entities on a single physical computer, enabling them to run different operating systems, applications or services independently. With virtualization, physical resources (such as processors, memory, storage and networks) can be partitioned into multiple virtual pools of resources, each of which can be allocated for use by different virtual entities. However, docker containers are lightweight virtualization units that share the host's operating system kernel, so they take up fewer resources and start up faster than traditional virtual machines[4].

In our projects, we have multiple developers working on different environments, each of which may be different, such as Windows, Linux, MacOS, etc. When our locally executed code is placed in another operating environment, some of the dependencies may cause problems, which is not conducive to co-development. When our locally executed code is put into another operating environment, some of the dependency libraries may cause problems in a way that is not conducive to co-development. Docker provides a portable way of packaging and deployment. We can package our application and its dependencies into a standalone container and run it in any Docker's environment without having to worry about differences in the underlying operating system and environment.

In addition, the use of docker containers allows for the isolation of the application environment. For example, in our frontend and backend components, which have a complex library of dependencies on the environment and are prone to problems, Docker's environment isolation allows applications to be isolated from each other and reduces problems caused by conflicting dependencies or changes in the environment.

Moreover, we often have problems when deploying dockers. But docker provides a wealth of tools and commands that make container management and deployment easy and convenient. For example, the *docker ps -a* command can directly output the status of each docker, and the *docker logs container_name* command can directly output the docker's logs, allowing us to quickly know the cause of errors. Among other things, the *docker compose* method can define and run multiple dockers at the same time, define multiple container configurations, network configurations and mounted volumes directly through a single configuration file, and provide simplified command line tools to start, stop, configure and manage these containers, mainly for managing multiple containers on a single host. Although this project did not end up using the *docker compose* feature, the importance of *docker compose* can be felt when the number of docker containers becomes large. As our project only used two dockers, we built a *my-network* by hand

[4] https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/

and manually added it directly to the network when creating the dockers. The need for a *docker compose* is not great.

3.8 Dynamically Scale Out

In the system architecture, our harvester is deployed in a separate instance. This instance is created automatically by ansible scripts. The IP address of each newly created instance is stored in *ansible/inventory/hosts.ini*. We can dynamically deploy any number of harvester servers, and the IP address of the created instance will be marked. Even if there is a problem with an extended instance, this does not affect the overall functionality.

4 Data Progressing

4.1 Twitter Data Pre-processing and Analysis

Lemmatization:

Word morphology reduction of words using the WordNet morphogenetic reducer in NLTK, which facilitates the subsequent sentiment analysis of tweets.

Text Preprocessing:

We use regular expressions to remove extraneous information from the text. Then, we keep only the characters that contain letters and convert the text to lowercase to remove the case difference. Next, we split the text into individual words using the word splitting function provided by the NLTK library. To improve the accuracy of our analysis, we also removed common stop words. To further accurately represent words, we word-ordered each word using the word-ordering function defined earlier to better capture their meaning in subsequent analyses. Through these pre-processing steps, we cleaned and prepared the text data to make it more suitable for subsequent tasks such as sentiment analysis and text classification.

Sentiment Analysis :

In the Sentiment Analysis phase, we use the TextBlob library and extract sentiment information from the text. By analysing the text for sentiment, we calculate the polarity of the sentiment, which is used to indicate how positive or negative the sentiment is. We classify the sentiment labels as 'Positive', 'Negative' and 'Neutral', and by returning the sentiment labels and polarity values, we can better understand and describe the sentiment tendencies in the text data. This helps with further tasks such as sentiment analysis, emotion recognition and sentiment classification.

Data Processing:

We read the Twitter file, check whether each item belongs to the predefined eight cities and extract the relevant fields. For the items that match, the function performs the sentiment analysis and text pre-processing defined earlier and constructs a new dictionary containing the processed information. This step allows us to process and analyse the Twitter data efficiently, obtaining the cleaned, sentiment analysed and pre-processed data.

These pre-processing steps ensure that the Twitter data is cleaned, normalised and additional fields are added for further analysis.

4.2 Industry Twitter data filter

After pre-processing the tweet data, we need to filter it for tweets related to industries. Firstly, we define a list of keywords related to the four major industries (education, arts, technology and finance) and then use the ijson library to iteratively read each item in the tweet file, extracting the key field information for each item. The type of industry the tweet belonged to was then determined based on whether the text contained keywords from the four industries, so that tweet items related to the four industries could be filtered from the tweet data.

It is worth mentioning that due to the large volume of tweet data, the mpi multi-process approach was used to filter the data in order to speed up the filtering process. This allows for parallel processing by splitting the tweet data into multiple parts, with each process independently processing the data segments it is responsible for. By using multiple processes, the performance of multi-core processors can be fully utilised to speed up the processing of Twitter data. Each process can perform different tasks at the same time, increasing processing efficiency. Especially for large Twitter data sets, the use of multiple processes can reduce the processing time considerably.

5 System Functionality

This section includes four scenarios that progressively unfold from a larger scope to smaller ones. They involve the analysis of Twitter data based on cities and months, sentiment analysis, the impact of job categories on tweet counts, and gender-based analysis within job categories. Lastly, it includes the implementation of data crawling and sentiment analysis for Mastodon data.

1. In the first scenario, we analyze Twitter data based on different cities and months.

2. Moving on, we delve into the impact of job categories on tweet counts and perform sentiment analysis.

3. In the subsequent smaller scenario, we further analyze the relationship between job categories, gender, and tweet counts.

4. Finally, we explore Mastodon data by crawling and conducting sentiment analysis to determine positive or negative sentiment.

Through the progression of these scenarios, we gain a comprehensive understanding of the interconnections between various data analysis results, starting from a broader perspective and gradually zooming in to examine specific details. This approach provides us with intriguing insights and observations. Following are some interesting findings based on the charts we have accessed.

5.1 First scenario

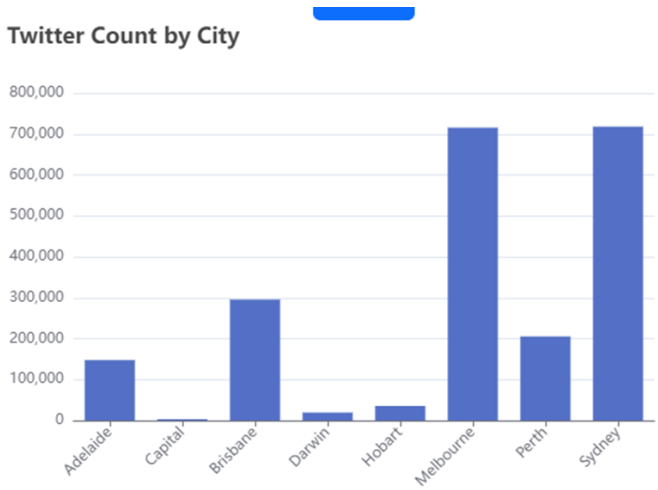

Figure 4 - Twitter Count by City

First, we compared the number of tweets posted in the eight major cities in Australia. In Figure 4, it can be observed that Adelaide had 147,533 tweets, Capital had 3,101 tweets, Brisbane had 295,440 tweets, Darwin had 19,303 tweets, Hobart had 35,384 tweets, Melbourne had 715,480 tweets, Perth had 205,325 tweets, and Sydney had 718,033 tweets. Melbourne and Sydney had a higher number of tweets, while

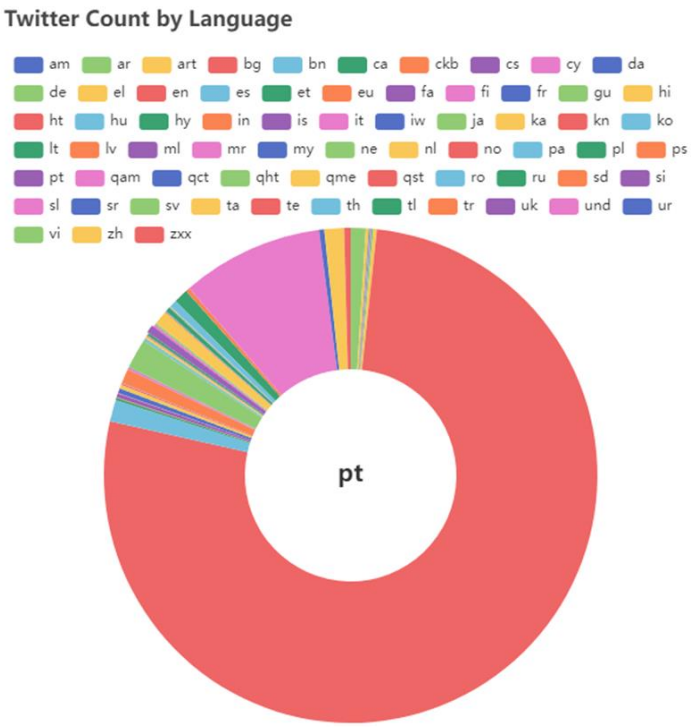

Figure 5 - Twitter Count by Language

Capital and Darwin had a lower number of tweets. The difference between the highest value, Sydney, and the lowest value, Capital, is 714,930 tweets.

In the Twitter language statistics (Figure 5), we found that the number of tweets in English was 1,643,545, accounting for 76.82% of the total. The next highest category is "und," which stands for "undefined" or "unknown," indicating that the language of the text cannot be determined. It accounts for 9.29% of the total tweets and may include content such as emojis or patterns.

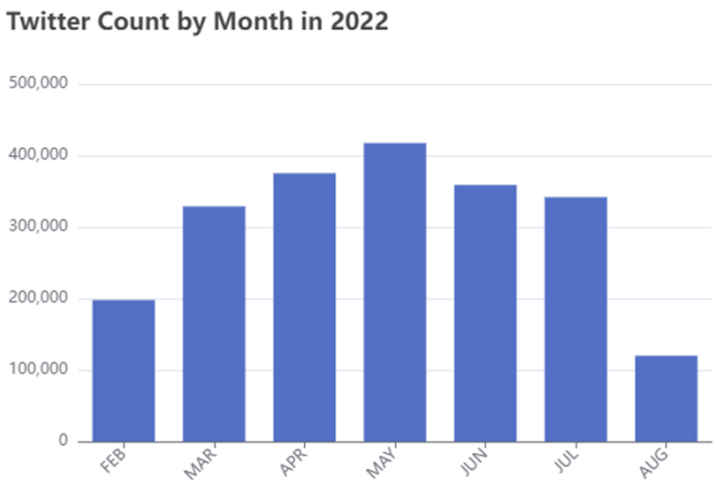

Figure 6 - Twitter Count by Month

In the monthly statistics for 2022 (Figure 6), based on the available Twitter data from February to August, we found that the highest number of tweets was in May, with 417,381 tweets, accounting for 19.51% of the total. The second highest was in April, with 375,240 tweets, accounting for 17.54%. Following that,

June had 358,828 tweets, accounting for 16.77%. The lowest number of tweets was in August, with 119,865 tweets, accounting for 5.60% of the total.

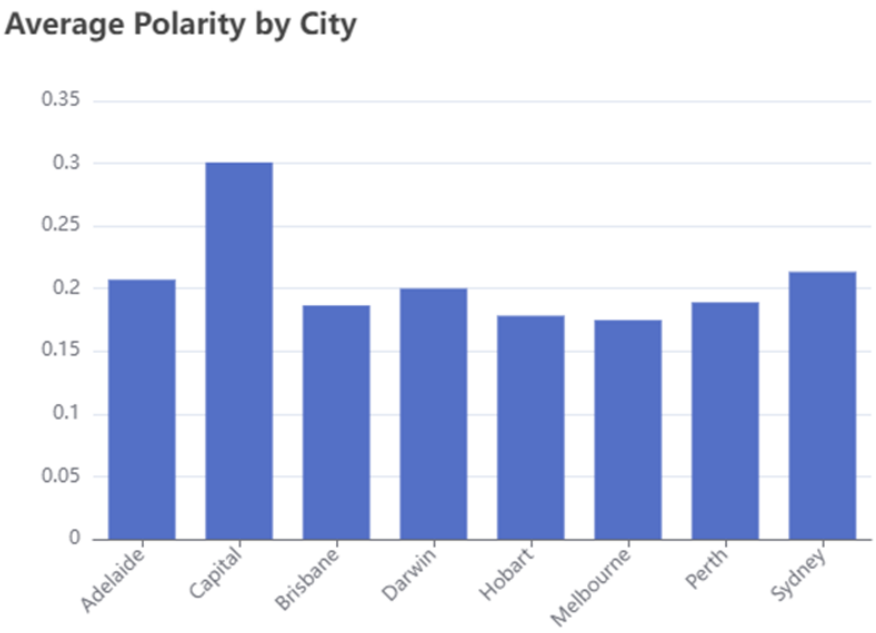

Figure 7 - Average Polarity by City

Based on Figure 7, we conducted sentiment analysis on the tweets from different cities and assigned a score to each tweet. By averaging the scores for each region, we calculated the polarity. From Figure 4, it can be observed that Capital had the highest polarity score, approximately 0.3001. On the other hand, Melbourne had the lowest polarity score, which was 0.1748.

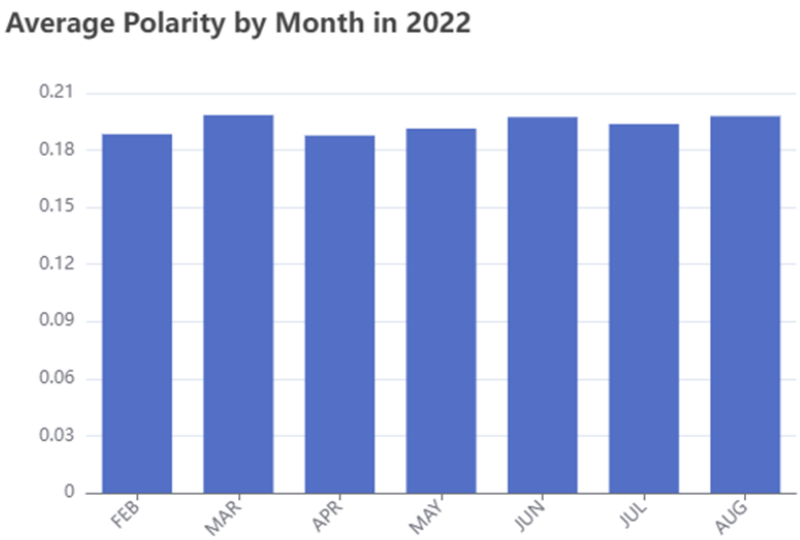

Figure 8 - Average Polarity by Month in 2022

According to Figure 8, we conducted sentiment analysis on tweets from different months, assigning a score to each tweet and calculating the average polarity score for each month. It can be observed from Figure 8 that the highest polarity score was in March, approximately 0.1984. The lowest polarity score was in April, with a score of 0.1877.

5.2 Second scenario

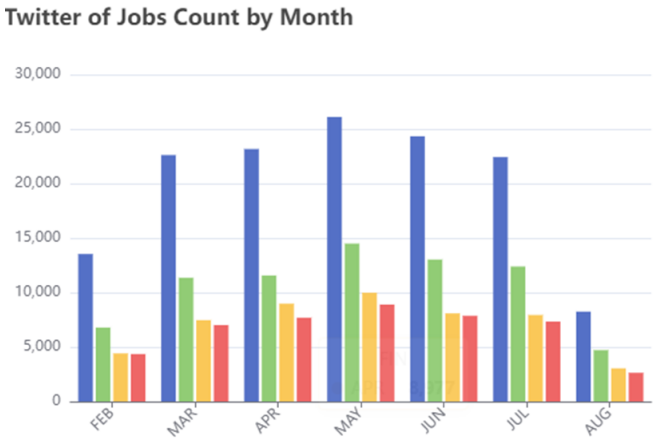

Figure 9 - Twitter of Jobs Count by Month

Figure 10 - Twitter of Jobs Count by Month in 2022

In Figure 9, we classified different jobs into categories. The blue color represents art jobs, the green color represents EDU jobs, the yellow color represents FIN jobs, and the red color represents Tech jobs. For Figure 10, across different months, we can observe that art jobs consistently had the highest number of tweets each month, while FIN and Tech jobs had relatively fewer tweets. May had a higher number of tweets compared to other months, while August had a lower number of tweets.

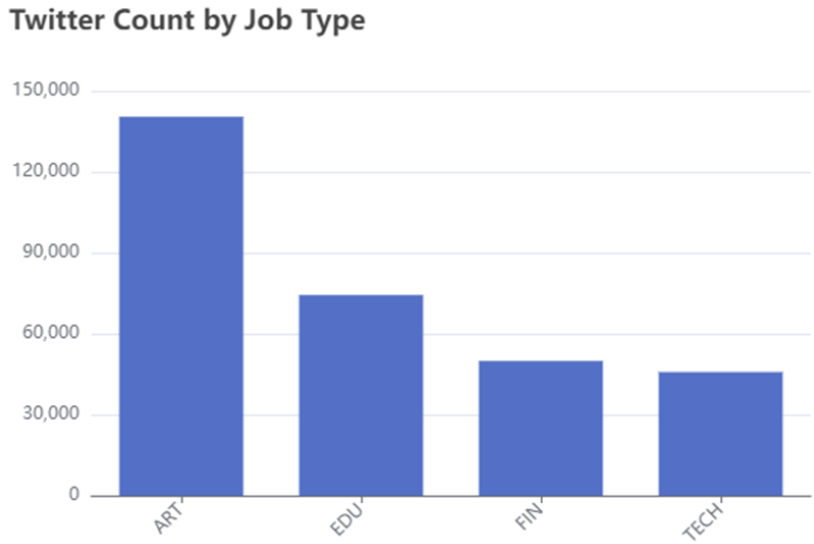

Figure 11 - Twitter Count by Job Type

In Figure 11, we conducted a statistical analysis based on four job types. It can be observed that the number of tweets for Art jobs was 140,427, accounting for the highest percentage at 45.23%. Edu jobs had 74,327 tweets, accounting for 23.94%. Fin jobs had 49,900 tweets, accounting for 16.07%. The lowest number of tweets was for Tech jobs with 45,787 tweets, accounting for 14.75%.

5.3 Third scenario

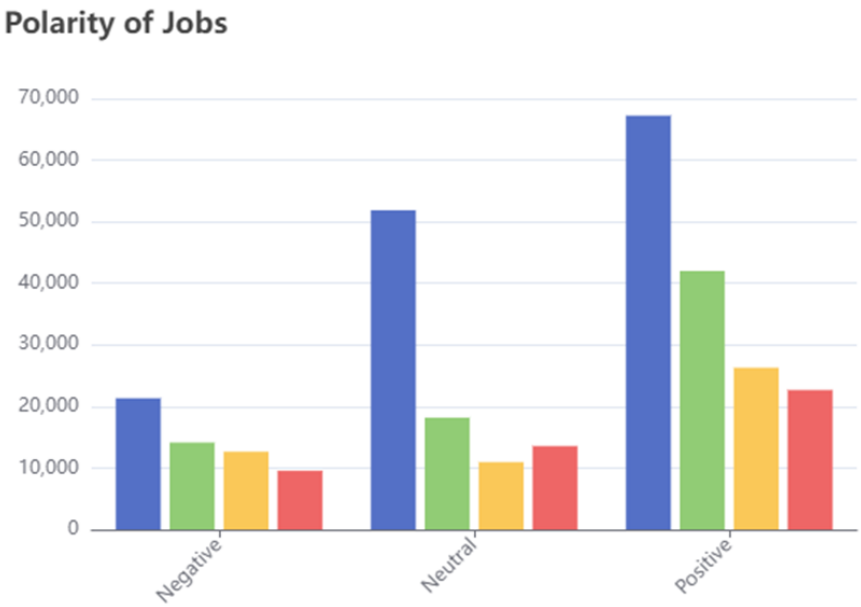

Figure 12 - Polarity of Jobs

In Figure 12, we categorized the polarity into negative, neutral, and positive sentiments, represented by the colors blue for art jobs, green for EDU jobs, yellow for FIN jobs, and red for Tech jobs. In the negative and positive categories, we can observe that art jobs had the highest number of tweets with the highest polarity scores. However, in the neutral category, Tech jobs had a higher number of tweets compared to Fin jobs. This is an interesting phenomenon worth noting.

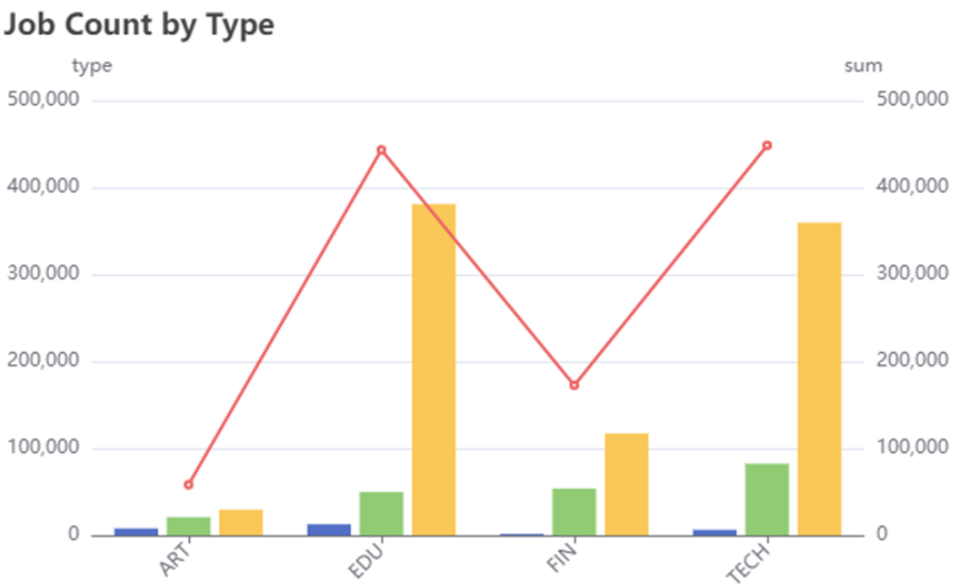

Figure 13 - Job Count by Type

In Figure 13, we conducted a statistical analysis of different job categories, where blue represents labourers, green represents managers, and yellow represents professionals. We can observe that Tech jobs had the highest number of tweets with 448,521 tweets, accounting for 39.95% of the total. EDU jobs had 443,450 tweets, accounting for 39.50%. Fin jobs had 172,580 tweets, accounting for 15.37%. Art jobs had 58,166 tweets, accounting for 5.18%.

When comparing different fields within each industry, we can see that professionals had a higher number of tweets compared to labourers and managers. The highest difference was observed in the Fin industry, where professionals tweeted approximately 70.7 times more than labourers. In the Art industry, professionals tweeted approximately 3.7 times more than labourers.

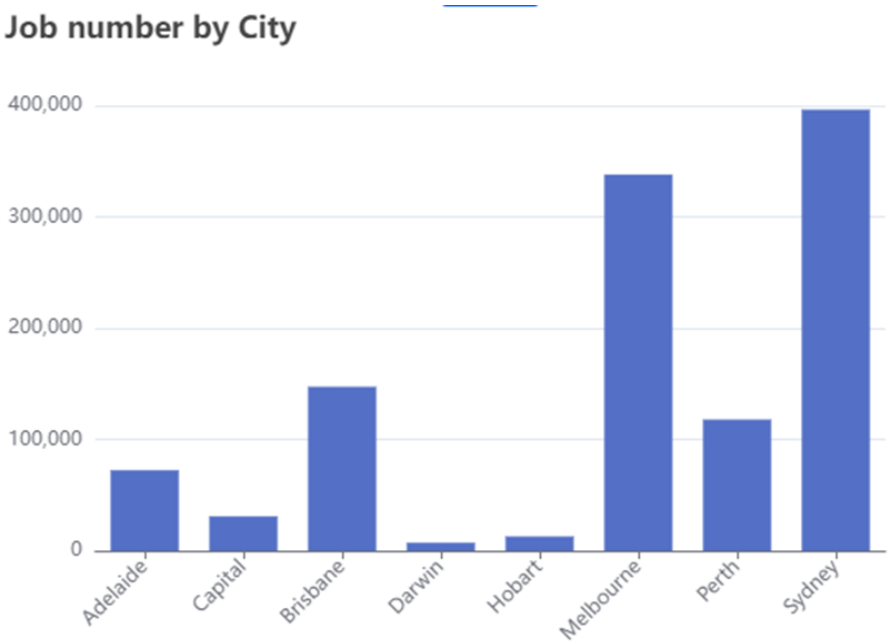

Figure 14 - Job number by City

In Figure 14, it can be observed that Sydney has the highest number of job listings, with a count of 396,354. Melbourne has 338,031 job listings, Brisbane has 147,247, Perth has 117,866, Adelaide has 72,306, Capital has 30,852, Hobart has 12,800, and Darwin has 7,161. Sydney has approximately 12.85 times more job listings than Capital.

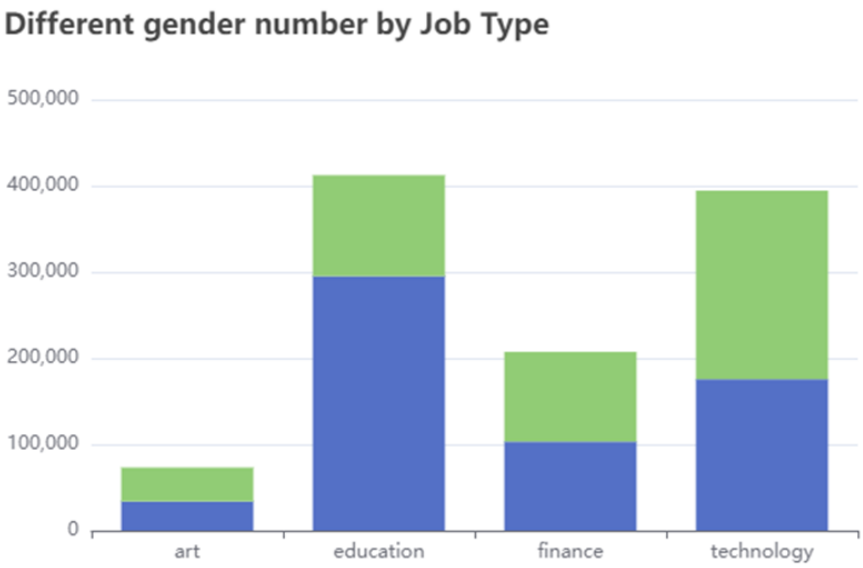

Figure 15 - Different Gender number by Job Type

In Figure 15, we also conducted an analysis of gender in different industries. In the EDU industry, there were 295,161 female individuals and 117,089 male individuals. This corresponds to a percentage of 71.60% for females and 28.40% for males. In the Tech industry, there were 175,598 female individuals and 218,736 male individuals. This corresponds to a percentage of 44.53% for females and 55.47% for males. In the Art and Fin industries, the gender ratios are relatively balanced, with both industries having around 50% male representation.

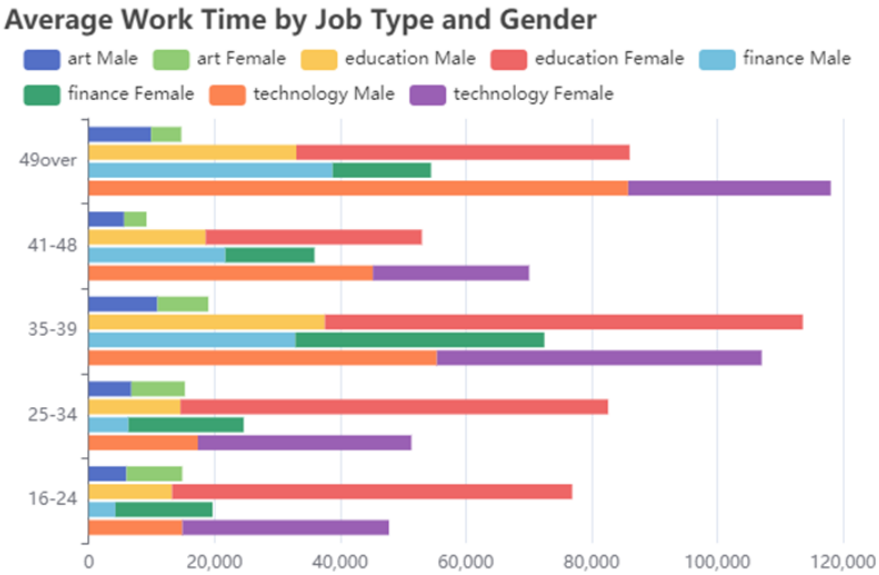

Figure 16 - Average Work Time by Job Type and Gender

In Figure 16, which utilizes a stacked horizontal bar chart, we categorized the data by different age groups. We found that the age groups of 35-39 and 49 and above had a higher number of tweets, while the age group of 16-24 had a lower number of tweets.

In the Art industry, for the age group of 49 and above, males had a higher number of tweets compared to females. However, in the age groups of 16-24 and 25-34, females had a higher number of tweets compared to males.

In the EDU industry, females had a higher number of tweets compared to males across all age groups. However, as the age increases, the proportion of male tweets becomes higher.

In the Fin industry, it was evident that females had a higher number of tweets compared to males below the age group of 35-39. Conversely, in the age group of 39 and above, males had a higher number of tweets compared to females.

Similarly, in the Tech industry, it can be observed that females had a higher number of tweets compared to males in the age group below 34. However, in the age group of 39 and above, males had a higher number of tweets compared to females.

5.4 Forth scenario

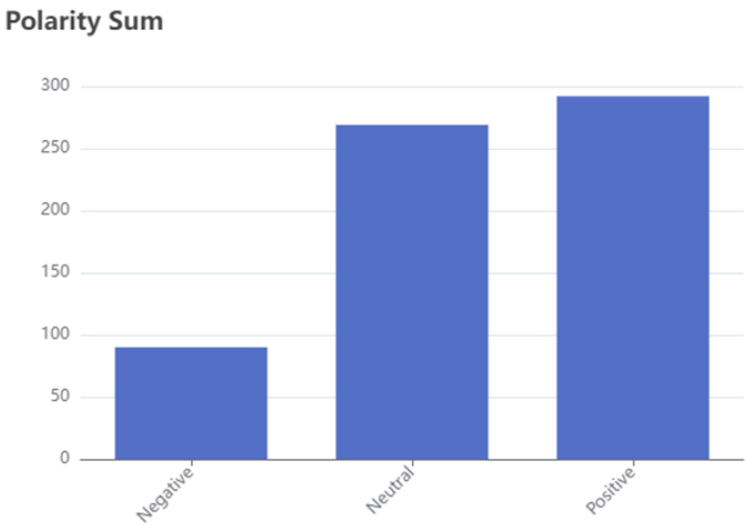

Figure 17 - Mastodon Polarity Sum

In Figure 17, we present the results obtained after filtering and conducting sentiment analysis on the data collected from Mastodon. From the figure, it can be observed that the majority of the topics selected by Mastodon users exhibit positive sentiment (292 posts) or neutral sentiment (269 posts), accounting for 44.85% and 41.32% respectively. Only a small number of posts showed negative sentiment (90 posts), accounting for 13.82%.

5.5 Summary

Based on the analysis of the 13 charts and findings, we can summarize several interesting phenomena and patterns:

1. In the language statistics of Twitter data, English is the predominant language, followed by "undefined" or "unknown" types of tweets, which may contain emojis or patterns.
2. In the analysis of tweet counts by month, May has the highest number of tweets, while August has relatively fewer tweets.
3. When analyzing the sentiment of tweets from different cities, Sydney and Melbourne have lower sentiment scores but higher tweet counts.
4. There are variations in tweet counts among different job categories, with the art industry having the highest tweet count and the finance and technology industries having lower counts.

5. Different job categories exhibit varying sentiment analysis results. For example, the art industry has higher scores in both negative and positive sentiments, while the technology industry has a higher count of neutral sentiment tweets.

6. In the analysis of gender within job categories, the education industry has more female tweets compared to male tweets, while the technology industry has more male tweets. Other industries show a relatively balanced gender ratio.

7. When examining tweet counts across different age groups, the 35-39 and 49+ age ranges have higher tweet counts, while the 16-24 age group has fewer tweets.

5.6 Key Observations

From Scenario 1, as seen in Figure 4 and Figure 7, we observe that Sydney (718,033 tweets) and Melbourne (715,400 tweets) have the highest tweet counts from February to August. This can be attributed to the larger population and rapid economic development of these cities. Surprisingly, despite having the lowest tweet count, the capital city exhibits the highest average sentiment value. We speculate that smaller cities may experience less work-related stress and competition, leading to more positive sentiments expressed in tweets compared to larger cities.

From Scenarios 2 and 3, as depicted in Figure 11 and Figure 13, an intriguing finding is that the art industry has the highest tweet count among the four major job categories, surpassing the others by a significant margin. This is despite having a smaller number of professionals in the art industry. One possible explanation is that artists rely more on social media to showcase their work and promote their art, leading to a higher frequency of art-related tweets.

From Scenario 3, specifically Figure 15 and Figure 16, we observe that in the education industry, female tweet counts surpass male tweet counts, whereas in the technology industry, male tweet counts outweigh female tweet counts. In the art, finance, and technology industries, as males age, their proportion of tweet counts increases. This observation may be connected to the mid-career crisis and higher life pressures experienced by males after the age of 35.

In Scenario 4, we find that most topics on Mastodon exhibit positive or neutral sentiment, while negative sentiment tweets are relatively scarce.

6 Issue and challenges

6.1 Deployment Issue

Port and network configuration for docker. When I first deployed the frontend and backend using the docker container, I was unable to access the frontend or backend URLs correctly, but after doing some research, I found that the docker container itself came with a port, and in the *Dockerfile*, I needed to expose the required port and map it to the local port correctly. So that services within the container can be accessed from a specific port on the host. In addition, when accessing the frontend port on the host after the current backend configuration was completed, I found that I could not get the back-end data and it displayed a timeout. after a period of research, I found that different dockers have their separate network configurations, so I needed to set up the containers to communicate with each other on the same network when building the containers.

Docker port security. To make other components accessible via all network interfaces, not just the local host, we set *host='0.0.0.0'* in the backend implementation. It will listen for requests on all network interfaces, thus allowing access from the external network. This is useful if you are running an application in a container and want to access it from the outside. However, this approach also exposes docker to potential attacks and security risks and could be a potential challenge.

Automatic instance creation does not get the IP. When creating a new instance of harvester, *item.server.access_ipv4* does not correctly get the correct IP of the instance. in fact, the instance's network is using the *qh2-uom-internal* internal network, while *item.server.access_ipv4* is the IP of the public network.

6.2 CouchDB & Data Pre-processing Issues

6.2.1 CouchDB upload speed issues

The initial upload process took the form of processing the data item by item and saving it to the database, which we found took a lot of time due to the volume of data in the tweet. So we improved the code and the new approach uses adding data to a bulk document list and performing a bulk save after a certain number of documents have been uploaded, which has the advantage of reducing the number of interactions with the database and the bulk save allows the data to be written to the database more efficiently. Compared to the original code, the batch processing of documents greatly reduces the overhead of database operations and thus speeds things up.

6.2.2 Streamlined data

The initial data pre-processing we saved all the data containing geographical information, which resulted in a large amount of data being transferred slowly, and it was found that many geographical coordinates were not needed for subsequent scenario analysis. To facilitate this, we first pre-processed the data locally by picking out the tweet dataset with the top eight cities and then uploaded it to CouchDB, which not only accelerated the transfer speed but also facilitated our subsequent analysis.

6.3 Backend Issue

Flask-Cors offers more granular configuration options, allowing customization of Cross-Origin Resource Sharing (CORS) based on specific requirements and security considerations. Here are a few ideas for fine-grained configurations:

1. **origins**: Allows specifying the allowed sources (domains) for cross-origin requests. It can be a single string, regular expression, list, or function. By configuring **origins**, you can restrict requests to specific sources, enhancing security.

2. **expose_headers**: Sets the exposed response headers accessible to the client. Cross-origin responses may include custom response headers. By configuring **expose_headers**, you can specify which response headers can be accessed by the client.

In addition to utilizing CouchDB's MapReduce functionality for data processing, we have implemented further improvements on the backend to optimize the data processing workflow. We have introduced additional data processing steps to enhance data accuracy, consistency, and availability. We have focused on performance optimization and scalability throughout the data processing process. By employing parallel processing and distributed computing techniques, we are able to handle large-scale data more efficiently and adapt to the growing volume of data. These backend improvements enable us to process data comprehensively and accurately, providing valuable analytical results and a solid foundation for our research and insights.

6.4 Frontend Issue

The development process was filled with several challenges that required meticulous solutions:

Real-Time Data Fetching: Fetching real-time data from different APIs was a crucial challenge faced during the development. The latency in response could potentially disrupt the user experience, necessitating an optimization of the data fetching process. This was addressed by using Vue.js's asynchronous methods and Axios for promise-based HTTP requests. By incorporating these tools, the application can handle API requests in a non-blocking manner, ensuring smooth user experience.

Data Visualization: The task of presenting data in a user-friendly format was a considerable challenge. The application needed to convert raw data into meaningful and dynamic visualizations. Vue.js's reactivity system and ECharts were used to handle this challenge. Vue's reactivity system ensures that any changes in data are automatically reflected in the UI. Additionally, ECharts, a powerful charting and visualization library, was employed to create intuitive, interactive, and highly customizable data visualizations.

UI/UX: Ensuring a user-friendly and aesthetically pleasing interface was a significant challenge. To address this, Bootstrap, a renowned CSS framework, was used, providing pre-designed components and responsive layout utilities, thereby enhancing the application's UI/UX design. However, despite these improvements, the team acknowledges that there is still considerable room for improvement and refinement. As the application grows and evolves, further exploration and integration of a wider range of components and resources available on platforms like GitHub could enhance the aesthetics and user interaction. This is viewed not as a shortcoming, but as an opportunity for continuous learning and enhancement of the application's UI/UX design in the future.

6.5 Harvester Issue

As we utilize multiple parallel mastodon harvesters to gather mastodon data into a single CouchDB dataset, a *FileExistError* occurs during the step where the processor checks if the target dataset already exists. This happens because all processors assume simultaneously that the dataset does not exist and attempt to create it. However, due to slight latency differences between the processors, all except the first processor realize that the dataset already exists in the CouchDB when they try to create it, resulting in the *FileExistError*.

To resolve this issue, we have implemented a solution that ensures only one processor creates the dataset while the others wait for it to be created. Although this approach slightly sacrifices the harvesting speed, the impact is negligible and does not significantly affect the overall performance of our mastodon harvesters. By synchronizing the dataset creation process, we have successfully mitigated the *FileExistError* and improved the reliability of our data gathering operation.

7 Video Link

7.1 Deploy the Whole System by Ansible

https://youtu.be/a68ELHguSYE

7.2 Frontend demo

https://youtu.be/04BCZEOVpMU

8 GitHub Link

8.1 Ansible Link

https://github.com/Comp90024-Group24/ansible.git

8.2 Frontend Link

https://github.com/Comp90024-Group24/Frontend.git

8.3 Backend Link

https://github.com/Comp90024-Group24/Backend.git

8.4 Data Preprocessing Link

https://github.com/Comp90024-Group24/data_preprocessing.git

8.5 Harvester Link

https://github.com/Comp90024-Group24/harvester.git

8.6 Whole system Link

https://github.com/Comp90024-Group24/Combine-project.git

9 Trello Link

https://trello.com/invite/b/FkSkTR4f/ATTI46645e05508e20d15496b3e99779dea06D600222/project-schedule

10 Individual contribution

Name	Contribution
Jiyuan Chen	The Whole system design and deployment with ansible script and dockers. Manage the MRC system. Twitter data and SUDO data pre-processing. Manage GitHub repository. Communicate closely with all team members to confirm the technical stack. Prepare the deployment section of the presentation. Record ansible deployment video. Write the report.
Kejie Xu	Create and transfer data to CouchDB CouchDB view creation for different scenarios Creating the initial scenario idea Write report Prepare the web demonstration section of the presentation
Xueqing Li	Frontend Design and Development Decide which views of data needed Record front end demo video. Hold zoom meeting Write report Design and write presentation slides
Yuqing Li	Mastodon Harvester Mastodon data pre-processing Prepare the web demonstration section of the presentation Construct the template of the report Write the report
Jiakang Li	Backend framework construction Communicator between frontend and database Meeting summary recorder Prepare the web demonstration section of the presentation Write the report

A Appendices