COMP90042 Cluster and Cloud Computing

Assignment 2 Report

Group20

Bowen Fan

Melbourne 1035162

Guanqin Wang

Melbourne 1074138

Junran Lin

Melbourne 1068324

Tianqi Wang

Melbourne 1045939

Yi (Eunice) Yao

Melbourne 1288896

May 2023

Contents

ABSTRACT

This report demonstrates the implementation of a software system built on Melbourne Research Cloud (MRC). The project focuses on performing sentiment analysis on various topics extracted from social media, combined with data from Spatial Urban Data Observatory (SUDO), at the Statistical Area Level 4 (SA4) scale in Australia. The document provides a comprehensive user guide, followed by an exposition of Ansible's role in automating deployment and scaling. The report illustrates the system's architecture, RESTful API design, and the advantages and disadvantages of MRC. Docker's utilization for one-click deployment of the web app is also presented. The report details the data used for the project, along with data processing and analysis procedures. It describes the function of Mastodon Harvester and the incorporation of CouchDB. The report concludes with the presentation of selected scenarios, providing valuable insights into the narratives of Australia, and an in-depth description of the system's front-end and back-end, incorporating design structure and implementation. This report establishes a robust system for conducting large-scale sentiment analysis.

Keywords Ansible · Big data analysis · CouchDB · Deployment · Docker · Mastodon · MRC · SA4 · Sentiment Analysis · SUDO · Twitter

1 User Guide

1.1 Deployment

This section provides a comprehensive guide on how to initiate, deploy, and execute our web application. Our system utilizes an Infrastructure as Code (IaC) methodology to effectively manage the DevOps processes dynamically. Ansible, an automated software tool, facilitates the interaction with instances or Virtual Machines (VMs) allocated on the Melbourne Research Cloud (MRC). The absence of a graphical interface fosters automated, efficient, and scalable deployment of the system.

Upon adhering to the outlined steps, you should successfully operationalize the application to a fully functional state and gain access to all its features.

1.1.1 Prerequisite

Before initiating the deployment process, ensure the following prerequisites are satisfied:

1. Possess an active Melbourne Research Cloud (MRC) account and have access to the project dashboard.
2. Clone the Git repository and navigate to the `ansible/` directory.
3. Download the `openrc.sh` file from MRC and place or replace it in the `/ansible/openrc/` directory, renaming it to `grp-20-openrc.sh`.
4. Position the private key `group_20.pem` in the `ansible/config` directory, which is essential for logging into MRC instances.

1.1.2 Auto Deployment

Once the prerequisites are fulfilled, execute the command outlined below to deploy the system. This procedure automatically manages authentication, dependency installation, environment configuration, and service initialization across all created MRC instances.

1. Initiate a terminal session within the `ansible/` directory.
2. Execute `chmod +x one-click-auto-deploy.sh` to modify the shell script's executability.
3. Launch `./one-click-auto-deploy.sh` to commence the automation process.

1.1.3 Auto Scaling

Scalability is paramount for modern applications. Given the dynamic nature of demand and resources, the ability to scale services both up and down is necessary to maintain user experience

and optimize resource utilization. The project demonstrates the auto-scaling process of the Mastodon Harvester. Although demand listening is not incorporated due to the insufficient data stream on the Mastodon server, it can easily be integrated into our system for large-scale applications.

1. Initiate a terminal session within the `ansible/` directory.
2. Execute `chmod +x auto-scale-up.sh` to modify the shell script's executability.
3. Launch `./auto-scale-up.sh` to enhance the harvester's scale. Utilize the `--new-ip` option to scale it to a new IP address sourced from an unused MRC instance. Alternatively, use the `--exist-ip` option to scale it to a pre-established node designated for data harvesting.

To auto-scale down the application, replicate the steps above but replace the shell script file name with `auto-scale-down. sh`.

1.2 System Navigation and Testing

The system should operate continuously on the Melbourne Research Cloud (MRC) instances. If the system is deployed from scratch following the procedure described above, once deployed you should be able to access and interact with our applications after the process completes. For each access, the URLs are listed in the appendix section.

The frontend pages serve as the visual representation of our analysis of the data corresponding to our investigative scenarios. RESTful design principles guide the backend construction, providing URL endpoints for data to be fetched. They are documented in the backend section of this report.

Testing the Ansible automation can be a complicated and potentially troublesome process due to the complexities of the dynamic environment and dependencies. We suggest a more accessible way using the `./auto-scale.sh --exist-ip` command for testing. This command is easy to use and ensures that the system can successfully dynamically scale and automate.

2 Ansible

2.1 Design Details

In this project, Ansible is used for auto-deployment and scaling but not for the creation of new instances on MRC due to some issues and limitations we currently experiencing with the cloud. Because the functionality of the system is rather complex, we delegate tasks to different roles and organize the Ansible directory in the structure shown in figure 1.

```
ansible
├── README.md
├── run-playbook.sh
├── playbook.yaml
├── config
│   └── group_20.pem
├── host_vars
│   └── config.yaml
├── inventory
│   └── hosts.ini
├── openrc
│   └── grp-20-openrc.sh
└── roles
    ├── clone-repo
    ├── install-dependency
    ├── run-mastodon-harvester
    ├── run-webapp
    └── scale-up-mastodon-client
```

Figure 1: Ansible directory

The structured Ansible directory enhances the manageability and clarity of the project's components. These components include role assignments for repository cloning, dependency installation, the execution of Mastodon Harvester and the web application, and the scaling of the Mastodon client. (Note 'run-playbook.sh', and 'playbook.yaml' is a representations of all the shell scripts and Ansible-playbook in this directory.)

The 'one-click-deploy.yaml' file comprises the core functionalities of our project. The file is divided into three sections, each corresponding to a specific task, and operates on a set of hosts characterized by their purpose: all-instances, mastodon-harvester, and web app.

```yaml
# Environment initialization on all instances
- hosts: all-instances
  vars_files:
    - host_vars/config.yaml
  gather_facts: true
  roles:
    - role: install-dependency
    - role: clone-repo

# Mastodon Harvester Setup
- hosts: mastodon-harvester
  vars_files:
    - host_vars/config.yaml
  gather_facts: true
  roles:
    - role: run-mastodon-harvester

# Web application setup
- hosts: webapp
  vars_files:
    - host_vars/config.yaml
  gather_facts: true
  roles:
    - role: run-webapp
```

Figure 2: Ansible code

In the first section, the instructions will be executed in all instances. This portion of the playbook ensures the installation of dependencies and cloning of repositories on all instances. Variable files (`vars_files`) from `host_vars/config.yaml` are loaded for each host.

The second section focuses on the setup of the Mastodon Harvester. It follows the same structure, but the execution is limited to hosts designated as `mastodon-harvester`. The role `run-mastodon-harvester` is assigned, initiating the Mastodon Harvester on these hosts, and allowing dynamic storing of raw and processed data into CouchDB.

Finally, the third section caters to the web application setup on the hosts labelled `webapp`. The role `run-webapp` is invoked which leverages Docker containers technology to establish both the front-end and back-end application and assigned necessary environment variables to the containers.

Our implementation presents several advantages. It provides clarity and easy navigation of tasks due to its structured format. This ensures efficient use of resources by running tasks on relevant hosts only, eliminating unnecessary operations. Nevertheless, it promotes re-usability and consistency through the use of roles. It indeed provides benefits to our system, as the auto-scaling functionality reuses the services from `run-mastodon-harvester`.

2.2 Limitations and Improvements

However, some drawbacks also surface. To ensure the playbook will run smoothly, we heavily rely on a well-structured inventory file to distinguish between various host groups. If the properties of instances change dynamically, for example, the utilization of floating IP, then we need to manually and constantly maintain the inventory file.

Another pitfall of the implementation is the absence of automation for CouchDB and its volume management. This process poses complex challenges, which influenced the decision not to implement it within the current scope of the project. Automated deployment of CouchDB must account for intricate considerations, such as clustering configurations for scalability, compaction and replication settings for performance, and robust fail start-over and backup strategies for reliability.

Furthermore, volume management with CouchDB involves the control of data storage, which requires interaction with the OpenStack APIs. Automating these tasks could introduce substantial risks if not done correctly, such as corruption of instances, or loss of valuable processed data. Therefore, considering the complexity and potential risks, it is not automated by Ansible. Instead, we use a shell script that can effectively map CouchDB data to the attached volumes that allow data backup.

Despite these potential challenges, adopting best practices, inventory and volume management can ensure the effective execution of the automation deployment and scaling processes.

3 Architecture

3.1 General Infomation

The architecture of the system is straightforward. A user accesses the front-end via HTTP links, with the front-end being accessible through port 3000. Both the front and back-end, hosted on the same instance, come with Docker-files and are built concurrently through a Docker Compose script at the click of a button. Frontend requests data from backend via a RESTful API, and backend, in turn, retrieves this data from a CouchDB database.

CouchDB is robustly configured, setting across several instances to ensure data safety even in case of a node failure. One instance is primarily responsible for storing pre-processed data, including a vast amount of 60GB Twitter data and supplementary data contained geographic information from Spatial Urban Data Observatory (SUDO). The remaining instances are tasked with harvesting data from the Mastodon platform. This setup allows us to handle an increasing stream of data effectively, with a backup harvester instance that can be scaled up dynamically.

The system deployment, including the data processing code, onto instances from a local setup, is managed through an Ansible script. All necessary environment will be initialized by Ansible. In case of a surge in the data stream, we can effortlessly deploy additional harvesters onto the backup instance using the same script. The necessary code will be deployed by cloning from git.

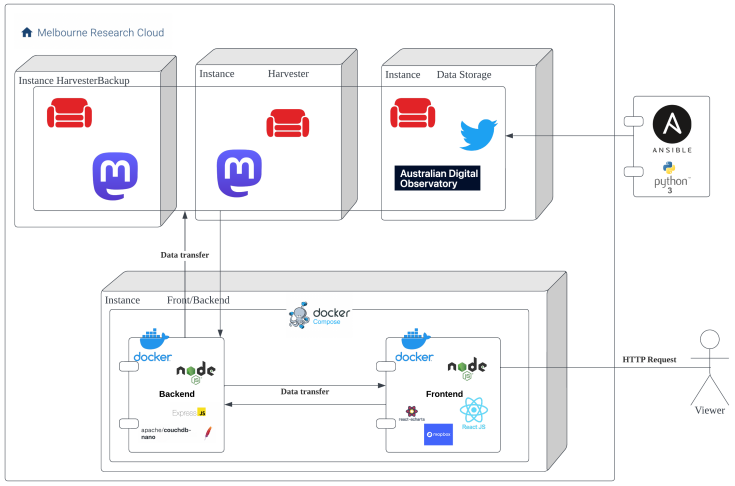


Figure 3: Architecture Diagram

3.2 RESTful API Design

Adhering to the RESTful API communication framework has been a strategic decision in the development of this whole project. The reasons for choosing to use RESTful APIs are most notably

due to their simplicity, scalability, performance, and compatibility. RESTful APIs follow a stateless, client-server architecture, where the client is responsible for the user interface and user experience, while the server is tasked with data management and operations. This separation allows developers to focus on distinct areas without impacting the other, enhancing both development speed and product quality.

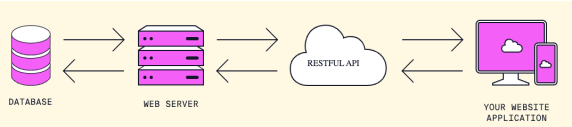


Figure 4: RESTful API

RESTful APIs are language-agnostic, that is they can be written in any programming language and interact with any other system that also uses HTTP protocols, which significantly broadens the compatibility. For our project, this is particularly beneficial as it facilitates diverse team collaboration, leveraging different skill sets without being confined to a specific programming language. For instance, we can use React to implement complex graphical interfaces and advanced chart features, while the backend is using Python Flask. The use of RESTful APIs allows effortless and efficient collaboration between the Mastodon data harvester, the data classifier, and analyzer, the CouchDB database, and the web application front and backend.

One particular advantage of RESTful APIs in our context is their inherent compatibility with CouchDB, which also communicates via REST. This compatibility ensures seamless communication between different components of the system, fostering an efficient and streamlined development process. Along with the MapReduce functionality provided by CouchDB, our system can access and manipulate complex data structures merely by passing URLs. This functionality simplifies data operations and, combined with RESTful APIs' stateless nature, ensures that the server does not need to retain excessive data or session details between requests, resulting in an overall performance increase. The RESTful APIs are built around HTTP protocol, and methods like GET, POST, PUT, and DELETE, each conveying a specific meaning of what an API call does. This increases the maintainability of the code and is extremely useful when performing CRUD operations with our CouchDB.

Furthermore, scalability is another key aspect of RESTful APIs that holds particular crucial for our project. As our system expands, RESTful APIs can easily scale to accommodate increased loads without significant alterations in the existing codebase. As we are required to dynamically scale up and down part of our system, employing RESTful APIs can allow us to distribute across multiple servers, ensuring the system remains responsive and resilient even as the demand grows.

The utilization of RESTful APIs within our project offers significant benefits, including simplicity, scalability, performance, and compatibility. Despite the challenges posed by our system's requirements, these APIs offer a robust and efficient means of managing diverse components that make up our project's architecture. With RESTful APIs at the core of our communication framework, we are able to deliver a reliable, high-performing, and scalable application.

4 MRC

4.1 Introduction

The Melbourne Research Cloud (MRC) operates as a private cloud service, functioning uninterruptedly around the clock. This service enables users to deploy scripts for continuous execution. Utilizing the intuitive user interface provided at https://dashboard.cloud.unimelb.edu.au/, instances on MRC can be configured and set up within a short time frame. The platform also offers an extensive selection of operating system images to cater to varied user requirements.

However, working with MRC instances can pose significant challenges, particularly for individuals unfamiliar with Linux-based systems, remote control operations, and deployment procedures. Nevertheless, these challenges present excellent learning opportunities. The complexity and intricacies involved in navigating the MRC environment provide a conducive environment for users to acquire practical skills and deepen their understanding of these systems.

4.2 Advantages of MRC

The Melbourne Research Cloud (MRC) offers several substantial benefits that contribute to its effectiveness and suitability for diverse computational tasks. These advantages can be broadly classified into three categories: scalability, efficiency, and security.

4.2.1 Scale-ability

The MRC framework provides seamless scalability, enabling users to easily create, revoke, and halt instances as per their workload requirements. This flexibility facilitates optimal resource allocation in line with the volume of work, promoting operational efficiency. Moreover, the MRC supports Ansible for setup, which further enhances scalability. Ansible scripts not only automate the deployment tasks for multiple instances but can also establish the instances themselves. This automated setup drastically simplifies the scaling process, facilitating swift and easy adaptation to increasing or decreasing workloads.

4.2.2 Efficiency

Utilizing multiple instances on the MRC can significantly improve task execution efficiency. By dividing a large computational workload amongst several instances, the system can concurrently process different parts of the task, leading to quicker completion times. This parallel processing capability enables the MRC to handle large-scale, complex tasks effectively, enhancing overall system productivity.

4.2.3 Security

MRC's security infrastructure is another noteworthy advantage. Each instance on the MRC is accessed and managed using a specific key pair. Without this key pair, no modifications are allowed on the instance, effectively preventing unauthorized access or changes. This strict authentication protocol maintains the integrity and confidentiality of the data and tasks managed within the instances, providing a secure environment for users to execute their computational tasks.

4.3 Disadvantages

While the Melbourne Research Cloud (MRC) offers numerous advantages, it is not without its shortcomings. One notable disadvantage pertains to the requirement of a Virtual Private Network (VPN). Unless a user is operating the project from the university's Wi-Fi network, they must connect to the Cisco Unimelb VPN to access instances via their respective IP addresses.

The VPN requirement introduces a layer of complexity and potential instability to the user experience. VPN connections can be unstable or inconsistent, often causing delays in accessing the MRC. In some instances, users might fail to establish a connection to the MRC entirely. This necessitates visiting the MRC dashboard to check logs and diagnose the cause of the failure, thereby consuming more time and potentially disrupting workflows.

More critically, the VPN-related instability can have detrimental effects on tasks performed through Ansible from the local host. A sudden loss of VPN connection could cause a currently running Ansible task to fail, further complicating operations. This requirement for constant VPN connectivity can thus contribute to inefficiency, operational disruptions, and an overall decline in user experience.

To circumvent these challenges, future improvements could explore alternatives to VPN connectivity. Developing a more stable and reliable connection mechanism would enhance the platform's usability and efficiency, especially for users working outside the university network. Moreover, ensuring that running Ansible tasks can gracefully handle temporary network disruptions could also help maintain operational continuity despite VPN instability.

In conclusion, the Melbourne Research Cloud (MRC) presents a versatile and robust platform for managing diverse computational tasks. Its scalability, efficiency, and stringent security offer a highly capable, adaptable, and secure environment. While the MRC provides a robust and scalable platform for managing computational tasks, its VPN requirement presents some challenges.

4.4 Setup

All instances are manually set up prior to use. The setup process, which is relatively straightforward, initiates from the MRC dashboard at MRC Dashboard. Firstly, the correct project must be selected as demonstrated in figure 5.

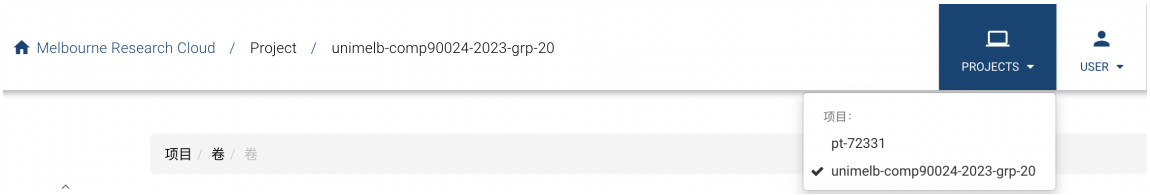


Figure 5: MRC project

Following this, a key pair is generated and must be saved securely. This is the sole method of authentication to gain access to the instances. The subsequent steps align with the guidelines provided in the "Creating VMs (MRC).pdf" document available on Canvas.

We have established a total of four instances at present, the specifics of which are outlined in the table below. Notably, we have recognized that the web application does not necessitate the volume size initially allocated. This observation underscores an opportunity for future improvements to optimally allocate resources.

	IP	Volume	Open Port
Data Storage	172.26.128.48	200G	5984
Web-app	172.26.128.31	150G	3000, 5984
Harvester	172.26.132.6	60G	5984
Harvester-backup	172.26.136.154	60G	5984

Table 1: Instances detail

Furthermore, we acknowledge the potential to automate instance setup using Ansible scripts, an enhancement worth considering for future iterations. For the Mastodon data harvester instance, deployment is executed locally through a one-click Ansible playbook script. This process encompasses setting up the environment, cloning necessary code from the GitHub repository, deploying the harvester, and collating raw and processed data into CouchDB. Remarkably, these operations can be performed using a concise shell script of less than ten lines, executed locally without needing to SSH onto the instance. A more detailed explanation will be provided in section 2.

5 Docker

5.1 Introduction

Docker is an open-source platform that leverages containerization technology to automate the deployment, scaling, and management of applications. IBM (nd) Containers encapsulate an application along with its entire run-time environment, including libraries, system tools, and code, allowing it to run consistently across different computing environments. Gillis (2020) A salient advantage of Docker is its isolation capability. Each container operates independently, ensuring that varying environments for different applications do not interfere with each other. This characteristic is particularly relevant in the project, where the back-end and front-end are built and deployed on the same instances. To prevent potential conflicts between the two environments, Docker is utilized. To better deal with multi-container situations, docker-compose is employed.

5.2 Docker Compose

Docker Compose is a tool provided by Docker that simplifies the process of managing multi-container Docker applications. It allows users to define and run complex applications using multiple containers with a single file - the docker-compose.yml file. With this file, users can specify the services required for the application to function, including the relationship between the services, the networks and volumes they will use, and other configurations. By simply executing Docker Compose commands, users can create and start all services from their configuration. It promotes effective organization, smooth integration, and easy scale-ability of multi-container applications.

5.3 Docker Swarm

Docker Swarm, also known as Docker Engine in Swarm mode, is a clustering and orchestration tool built into Docker. It transforms a group of Docker nodes into a single, coherent, and distributed system, or a 'Swarm', thereby facilitating the management and deployment of services across multiple nodes. Docker Swarm provides robust capabilities for service discovery, load balancing, service replication, fault tolerance, and more. Docker Swarm allows users to manage their application across multiple nodes as if they were handling a single system, simplifying the process of maintaining large-scale Docker deployments. Its emphasis on scalability, decentralization, and secure design makes it a potent tool for managing and scaling containerized applications.

5.4 Implementation

The construction of the frontend backend components of this project is facilitated by invoking a singular shell command, namely "sudo docker-compose up". This action initiates the execution of

the docker-compose file, which, in turn, builds frontend and backend separately, each according to its respective Dockerfile.

The Dockerfile, essentially a script containing a series of instructions, governs the creation of the Docker image. The general structure of the Dockerfile used in this project comprises the following steps:

- Retrieving the Base Image: The Dockerfile commences by pulling the base image, in this case, node:20.1.0-alpine3.16. This lightweight image is particularly suited to the structure of our front-end and back-end components owing to its compact size and compatibility with Node.js applications.
- Updating and Installing: The next step involves updating the APK package manager native to Alpine Linux and installing bash. Bash is incorporated to facilitate scripting and interactive command-line interface tasks.
- File Copying: Subsequently, the Dockerfile instructions entail copying the necessary files into the working directory within the Docker image. This ensures all requisite application resources are included in the image.
- Environment Setup: A shell script containing instructions for installing the required environment is copied into the Docker image and given the necessary execution permissions (chmod +x).
- Script Execution: Following the setup, the shell script is executed. This step installs any additional software or packages necessary for the application's operation.
- Port Exposure: Finally, the requisite ports are exposed to facilitate communication between the application and external services. In this instance, port 8000 is designated for the backend, while port 3000 is assigned to the front-end.

Upon successful execution of the Dockerfile and subsequent creation of the Docker image, the web application becomes operational. The utilization of Docker and Docker Compose ensures a streamlined and efficient build process, enabling the concurrent setup of both frontend and backend environments. This approach reduces potential conflicts and increases the overall development and deployment efficiency, thereby making Docker a crucial element of our project infrastructure.

5.5 Improvement

The existing architecture leaves room for enhancements that can strengthen its fault tolerance, portability, and overall performance. A couple of key areas of improvement are outlined as follows:

1. The utilization of Docker Swarm can elevate the robustness of the infrastructure significantly. "Docker Swarm can transform a group of Docker nodes into a ‘Swarm’, which is a single, distributed system." Docker (2019) With Swarm mode, every instance could potentially serve as a web-app server, thus improving failure tolerance and increasing portability.

2. Currently, updates to the frontend are carried out by cloning the repository from GitHub and rebuilding the Docker image, a process that will lead to downtime for the server. A more streamlined method would involve pushing the updated Docker image to Docker Hub and subsequently pulling it for deployment. Such an approach would enable a ‘rolling update’, a strategy that applies updates to individual nodes or services within the swarm sequentially. This approach would ensure that some nodes remain operational during the update process, thereby minimizing disruptions to the service. Moreover, the rolling update strategy can also revert updates if anomalies are detected during the update process, thereby adding an extra layer of safety.

6 Data

As the scenarios concentrate on sentiment analysis in the domains of Food, Traffic, and Sport, the data will be manipulated in ways specified in the coming sections.

6.1 Data Source

6.1.1 Australian Data Observatory (ADO)

From ADO, a large amount of data from Twitter whose size is about 60GB is introduced to extract important insights specified for exact topics of the scenarios related to Life in Australia. The Twitter data in the form of JSON format comprises the following information:

1. ID: Unique identifier of the tweet
2. Date of the tweet being posted
3. Content of the Tweet in the form of tokens or original text
4. Author ID
5. Users mentioned in the tweet (name, user id, and positions in the tweet)
6. Language
7. Public metrics
8. Sentiment
9. Geo-information (the name of places, bounding boxes)

To be exploited for scenarios, *Content*[1] represented by *Tokens*[1], *Language*[1], *Sentiment*[1], and *Bounding Box*[1] are chosen to be preprocessed, which is illustrated in the following section.

6.1.2 Spatial Urban Data Observatory (SUDO)

From SUDO, many kinds of interesting data within a particular geometry level can be exported for the research of scenarios. Specifically, the datasets in Statistical Area 4 Level which contains the largest sub-state regions in the main structure of the ASGS are chosen as follows:

1. Weekly Median Personal Income in 2021, from AU-Govt-ABS-Census (2021b)
2. Number of Vehicles per Dwelling in 2021, from AU-Govt-ABS-Census (2021a)
3. Population by Age Group in 2021, from AU-Govt-DESE (2021)
4. Statistical Areas Level 4 - 2021 - shapefile, from Australian Bureau of Statistics (2021)

[1] keys from twitter's JSON file

The reasons to select those datasets are as follows: First, the dataset related to population is used to average aggregated statistics by the number of persons in SA4-Level regions to avoid the effect of geometrical imbalanced population structure. Second, under specific topics, the datasets related to income and vehicle can be utilized for comparison against the statistical information extracted from Twitter data. Lastly, the shapefile of SA4 is designed for classifying each tweet to the SA4-Level area it is in.

6.1.3 Mastodon

Posts (Toots) are harvested from several servers by Mastodon APIs to observe whether the same kind of information exists when compared to Tweets. The context of Toots is preprocessed for sentiment analysis and topic classification.

6.2 Preprocessing

6.2.1 Filter the Tweets

The size of original Twitter data is around 60GB and those tweets may lose some sort of information (e.g. lack of Geo-information). Besides, there is a variety of languages used in tweets. However, since the topics focus on Australian Life, tweets in English is the primary condition for filtering. Furthermore, as the data from SUDO is connected with related geo-locations, the second filter condition should be that those tweets contain bounding boxes.

Figure 6: Workflows

As shown in Figure 6, two workflows to filter the original Twitter Data have been illustrated. Two important factors are considered for the selection of workflows: Time and Process Efficiency (i.e. Cores that can be utilized for Parallel Programming). First of all, the strategy of Parallel Program

applied for filtering is Single Program Multiple Data (SPMD) as initially, the 60GB file comprises about 100 million Tweets, which can be regarded as being large. Thus separating those Tweets for the same job is expected to decrease the processing time by the number of divisions of data which relies on the number of cores that can be used. By observing the blue rectangles in the Figure 6, the amount of data that needs to be transmitted is less in Workflow 2 than in Workflow 1. Depending on the speed of transmissions, Workflow 2 can obviously save more time. As for the Process Efficiency, only 8 cores in total are provided on the MRC. Nevertheless, in the owned local machine, 14 cores can be utilized for parallel processing by mpi4py in Python. Considering that solely using the number of cores as the standard, realizing Workflow 2 in the specific local machine will save considerable time. Consequently, Workflow 2 was implemented to filter the Tweets by their language and the existence of geo-information. After the workflow, 2.5 million Tweets are chosen for further preprocessing.

6.2.2 Classify each Tweet to its belonging SA4-level Area

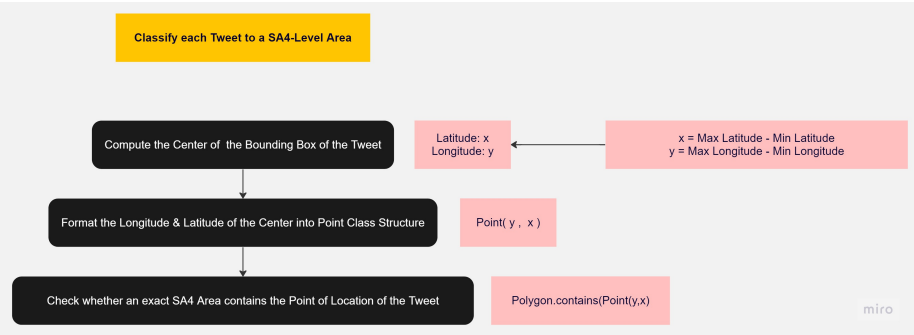


Figure 7: Location Classification

Figure 7 illustrates the process of the SA4 Area classification. It should be noticed that in the shapefile provided, some areas do not contain the related polygons. Those areas are assumed not to be presented in the further sections. For the process, each tweet goes through all the polygons of areas to confirm that the center of the bounding box is located in any one of them. If so, the location of the tweet will be labeled by the code of the SA4 area. For this task, the Parallel Programming Strategy SPMD is also applicable to reduce the time required for a total of 2.2 million tweets and 89 polygons.

6.2.3 Topic Classification

As stated in Figure 8, topic classification is mainly completed by key token matching after lower-casing the tokens of the tweet. Each topic collects the tweets that belongs to it, while some tweets may belong to multiple topics. It is noted that the Wordnet in NLTK library in Python is utilized to generate the key tokens which are hyponymy of the Word describing the topics (e.g. food).

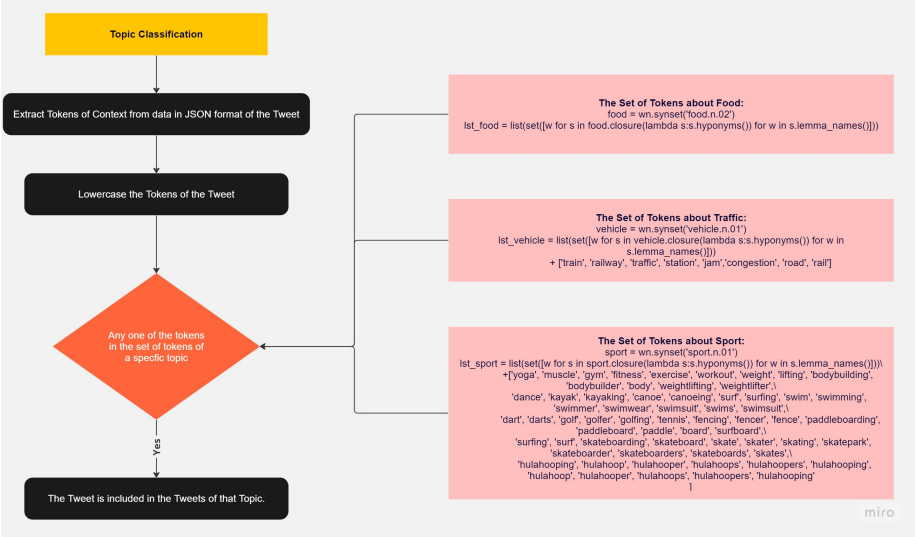


Figure 8: Topic Classification

6.2.4 Limitation

Ambiguous sense of a word/token

For instance, one of the senses of the word ‘dog’ which is in the set of Food is “Synset(‘frank.n.02’): a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll” as returned by the Wordnet. However, normally and generally, the ‘dog’ in tweets relates to the topic of Pet. Therefore, as there are some other words with senses not usually used, this method includes many tweets unrelated to the topic. It is possible to solve the problem by manually inspecting the tokens and removing the tokens that literally do not belong to particular topics.

Phrases in the Set of Tokens of Topics

In the set of tokens of topics, there are phrases represented by “word1_word2” that are not handled by this method. A solution is, first, building a bi-gram language model of the tokens in the tweet. After that, for those phrases, the underline score is replaced with space and then if they appear in the bi-gram model, it will be considered as a match. Nevertheless, the solution pays for the price of consuming more computation overhead.

Normalization of tokens by Lemmatization

For the tokens of the tweet, they are likely to be inflected in English. For example, the ‘egg’ is inflected to its plural ‘eggs’, making the method fail for matching. Without changing the meaning of the tokens, only the inflectional morphology is considered to be removed by using ;emmatization which turns the tokens into an uninflected form. Therefore, by lemmatizing both the tokens of the tweet and the set of topics, higher performance of matching can be achieved.

A better approach will be using a fine-tuned pre-trained BERT language model. The model takes the complete sentence of the tweet as the input and returns the topic label of the text. Once the specific topic labels are chosen, the model is able to perform better as the contextual representation in the last layer of the BERT model can better reflect the meaning of the context in the tweet rather than naively matching keywords.

6.2.5 Topics Modelling by LDA

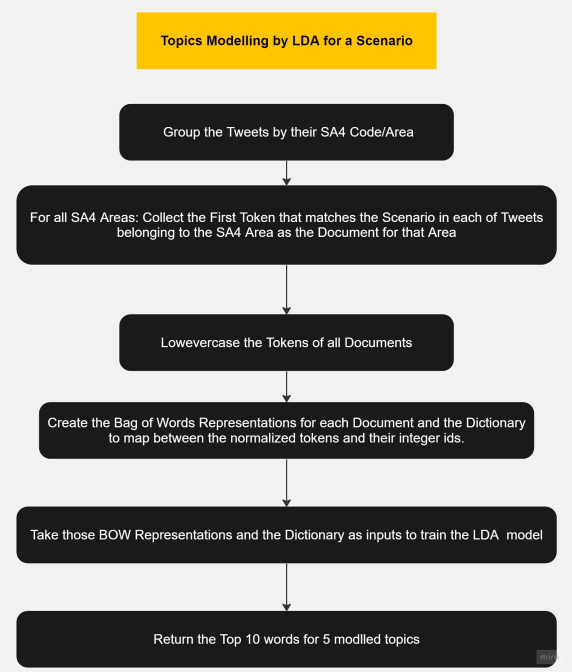


Figure 9: Topics Modelling LDA

In this step, for the sake of observing any interesting patterns that would appear in the topics from the tweets belonging to each of the SA4 Areas, the LDA model used for topics modeling is implemented as shown in the Figure 9 by exploiting the Gensim library in Python. It is assumed that there are 5 topics related to the documents of all SA4 areas, which may also produce different results if this hyperparameter is changed. Out of the purpose of illustration, the random state and the number of topics to be modeled are fixed in this case. Those 5 topics for each scenario are presented in the form of Word Cloud in the web application and further analyzed while discussing the scenarios.

6.2.6 Process of Mastodon Data

The data from Mastodon contains far less entries compared to Twitter and is not able to be geocoded correctly to Australia as well as scarce the sentiment of the toots. For this case, only the text of toots is used for research on the distribution of sentiment and the topics of scenarios. The processing of the text of toots includes two tasks as follows:

The first one is Topic Classification as mentioned above. The only difference is that the text of toots is tokenized first, after which those tokens are used for checking whether it belongs to a specific topic. Nevertheless, the limitations and possible improvements are the same as discussed before.

The second one is Sentiment Prediction based on a fine-tuned pre-trained RoBERTa model for predicting the sentiment of a sentence. The text of the toots is first preprocessed as required for the model, after which the preprocessed text is taken as the inputs to the model. The model will return the texts' possibilities of being positive, negative or neutral. And the label with the highest possibility will be chosen as the sentiment of the toot.

7 Mastodon Harvester

7.1 Introduction

This section presents a detailed overview of the Mastodon harvester, with further information on the deployment of multiple harvesters available in Section 2.

7.2 Harvesting

The primary goal of the harvester in this project is to capture live streaming data from Mastodon, focusing primarily on two Mastodon servers: Mastodon AU (`https://mastodon.au/`) and AU Social (`https://aus.social/`). This harvesting operation is achieved through a well-structured architecture consisting of four integral files as depicted in Figure 10.

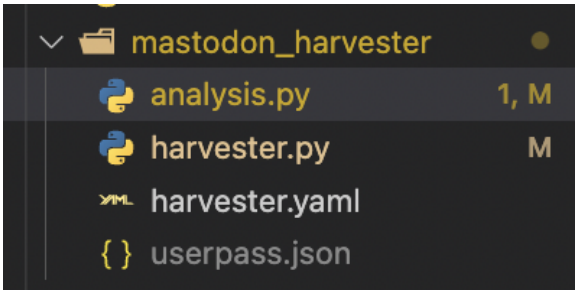
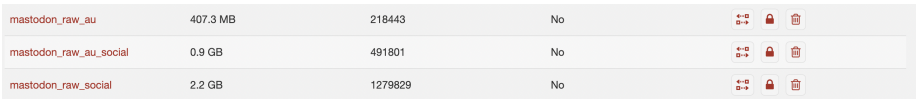


Figure 10: Harvester structure

- *analysis.py*: This file encompasses the analytical functions responsible for processing incoming data. It features advanced algorithms to parse and interpret the data, preparing it for subsequent uses.
- *harvester.py*: This file is the cornerstone of data collection. It employs scripts to fetch updates from the Mastodon servers continually. This file utilise the Mastodon.py library.
- *harvester.yaml*: This file includes an Ansible script purposed to facilitate the setup process of the harvester.
- *userpass.json*: This file retains the authentication tokens required for access to the Mastodon servers and their respective URLs.

By integrating a stream listener, the harvester is capable of monitoring any updates on the Mastodon servers in real-time. The harvested data is then stored directly in a CouchDB database as raw data.

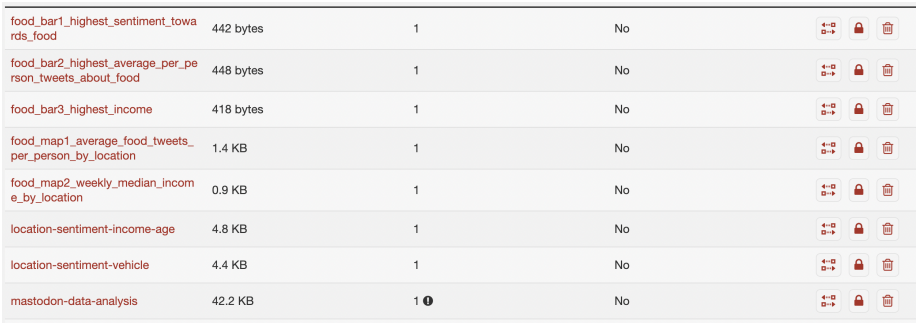
Following this, the analytical function is invoked to process the raw data and store the processed data, as shown in Figure 12. At present, the processed data intended for display is stored in separate files. While this approach works, it may not be the most efficient or scalable solution. An improvement



mastodon_raw_au	407.3 MB	218443	No	
mastodon_raw_au_social	0.9 GB	491801	No	
mastodon_raw_social	2.2 GB	1279829	No	

Figure 11: Raw data in CouchDB

to consider could be integrating a more robust, scalable, and efficient data management solution, ensuring optimal performance as the volume of harvested data increases.



food_bar1_highest_sentiment_towards_food	442 bytes	1	No	
food_bar2_highest_average_per_person_tweets_about_food	448 bytes	1	No	
food_bar3_highest_income	418 bytes	1	No	
food_map1_average_food_tweets_per_person_by_location	1.4 KB	1	No	
food_map2_weekly_median_income_by_location	0.9 KB	1	No	
location-sentiment-income-age	4.8 KB	1	No	
location-sentiment-vehicle	4.4 KB	1	No	
mastodon-data-analysis	42.2 KB	1	No	

Figure 12: Analysed data in CouchDB

7.3 Data Process

The detail of how to process the data refers to Section 6.2.6.

7.4 Single-point Failure

Single point failure refers to a situation where the failure or malfunctioning of a single component, system, or process can lead to the complete failure or disruption of an entire system or operation. When harvesting, a single failure of converting to JSON file or failing to save to the database might eventually lead to a complete shut down of the whole harvesting system. As a result, a try-except frame is used to avoid possible failure when harvesting the stream data to avoid single-point failure causing the whole harvester down.

7.5 Improvement

Several potential enhancements can be incorporated into the existing system to optimize the performance, particularly focusing on how we manage, store, and retrieve the analyzed results.

The current system of managing the analyzed data relies on continual modifications and overwrites to a single file. Although this method is functional, it lacks efficiency and has potential issues concerning data integrity and concurrent data access. Moreover, it requires substantial computational resources, especially as the data volume grows.

An effective alternative solution could be the integration of a MapReduce function into the system. By implementing a MapReduce function, the backend could easily access the necessary data by invoking the associated API. This method would bring about the following benefits:

1. Reduced file modifications: The need to consistently revise a single file would be eliminated, as the data can be accessed and manipulated efficiently using the MapReduce function.
2. Conflict avoidance: Potential conflicts that may arise due to simultaneous file access can be mitigated, ensuring data integrity and consistent performance.
3. Resource Optimization: The computational resources can be more effectively utilized, thereby reducing the overall system load and promoting better performance.

As such, shifting towards a MapReduce-based architecture could offer a scalable, efficient, and reliable solution for handling and processing large volumes of analyzed data.

8 CouchDB

CouchDB plays a vital role in this project, not only for its friendly user interface "Fauxton", but also due to the capability to directly and easily access through various languages such as Python and JavaScript. Moreover, CouchDB's capability to handle vast volumes of data while ensuring high-speed retrieval significantly contributes to the project's efficiency and reliability.

8.1 Clustering

Cluster CouchDB is a remarkable feature that improves the database's reliability, scalability, and performance. It allows for the seamless replication of data across multiple nodes, resulting in a distributed architecture that provides high availability and fault tolerance.

The main concept of this function is horizontal scaling - the ability to increase capacity by adding more servers to the network as opposed to upgrading the hardware capabilities of a single machine, also known as vertical scaling. A crucial aspect of CouchDB's clustering is the conflict resolution mechanism. It allows all writes to proceed successfully and leaves the conflict resolution to be done at read time or during replication. This is a result of its "eventually consistent" design principle, which values availability and partition tolerance over immediate consistency.

In addition to its benefits for reliability and scalability, CouchDB's clustering also provides a strong safeguard against data loss. If one node in the cluster fails, the data can still be accessed from another node.

8.2 CouchDB Replication

Replication is another key capability to make the database fault-tolerant. In the case of streaming Mastodon data, replication is employed to store the same documents on different nodes, ensuring data resilience and fault tolerance.

8.3 Map-Reduce

The MapReduce capability offered by CouchDB is employed to improve the efficiency of data analysis. To be specific, it is utilized to extract information from social media data and calculate the summary statistics of the extracted information.

Views can be created from the design documents for a specific database. By specifying the map function, data with specified characteristics can be mapped and grouped. The reduce function therefore enables the calculation of required statistical values. Once the view is specified, data processing takes place within the database, minimizing the impact on network resources. Additionally,

the clustering and parallel computing nature of the MapReduce are implemented across nodes to optimize data processing efficiency.

8.3.1 Implementation: MapReduce to capture essence of life in Australia

In our analysis, a significant portion of sentiment data analysis and integration is accomplished through MapReduce. We classify the Twitter data based on the geographical information, and then utilize the mapping process. An example of the execution code is provided in the accompanying images figure 13 and 14.

Map function

```
function (doc) {
  if (doc.doc.includes.places[0].full_name) {

    var sentiment = doc.doc.data.sentiment;
    var sentimentLabel = "neutral";
    if (sentiment > 0) {
      sentimentLabel = "positive";
    } else if (sentiment < 0) {
      sentimentLabel = "negative";
    }

    emit(doc.doc.includes.places[0].full_name, {
      positive: sentimentLabel === "positive" ? 1 : 0,
      negative: sentimentLabel === "negative" ? 1 : 0,
      neutral: sentimentLabel === "neutral" ? 1 : 0
    });

  }
}
```

Figure 13: Map Function Demo

Custom Reduce function

```
function (keys, values, rereduce) {
  var result = {
    positive: 0,
    negative: 0,
    neutral: 0
  };
  if (rereduce) {
    values.forEach(function (value) {
      result.positive += value.positive;
      result.negative += value.negative;
      result.neutral += value.neutral;
    });
  } else {
    values.forEach(function (value) {
      result.positive += value.positive;
      result.negative += value.negative;
      result.neutral += value.neutral;
    });
  }
  return result;
}
```

Figure 14: Reduce Function Demo

The MapReduce view can be accessed and queried using the RESTful API provided by CouchDB, so the processed data can be accessed by both the frontend and backend, which enhances the efficiency of data transmission. Furthermore, once views are created, subsequent retrieval of the views is significantly faster, thereby improving the overall data extraction process.

9 Scenarios

This project explores interesting stories of daily life in Australia by focusing on sentiment analysis related to three common topics: Food, Traffic, and Sport.

For better comparison and analysis, the regions of the study are separated according to the Statistical Area Level 4 (SA4) classification. Each SA4 area serves as a distinct unit of analysis, allowing for the exploration of different characteristics within each region. The characteristics include sentiment and popularity towards specific topics, and the overall economic conditions of the region. By comparing and analyzing these features, the project aims to uncover intriguing insights and draw meaningful conclusions.

9.1 Scenario 1: Food

Among the selected data, 6% are related to the topic of food, and 46% of them show a positive sentiment, which implies a general sense of happiness when discussing food in Australia. In specific, three key statistics related to food are explored in each area. First, the average sentiment towards food is examined to provide insights about the overall attitude expressed by individuals. Second, the average frequency of food-related posts by individuals is measured to offer an indication of the level of engagement with food-related topics. Last, the weekly median personal income is analyzed to establish the economic context and explore potential correlations between income levels and sentiment towards food.

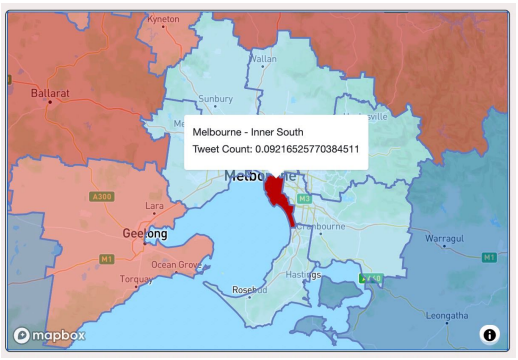
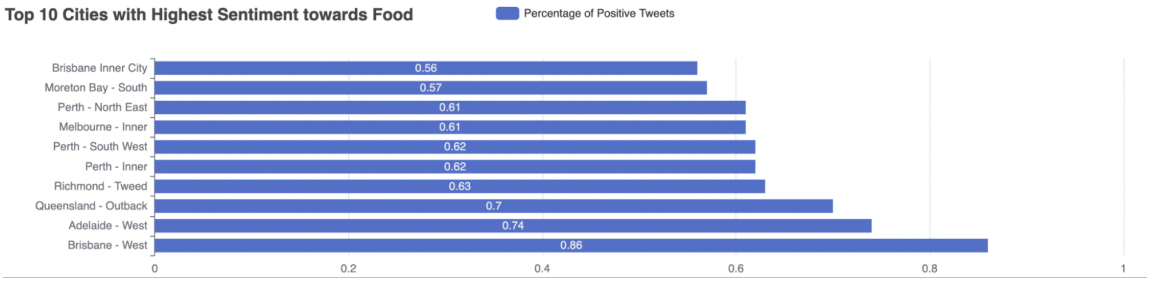


Figure 15: Average Number of Tweets About Food per Person by Location

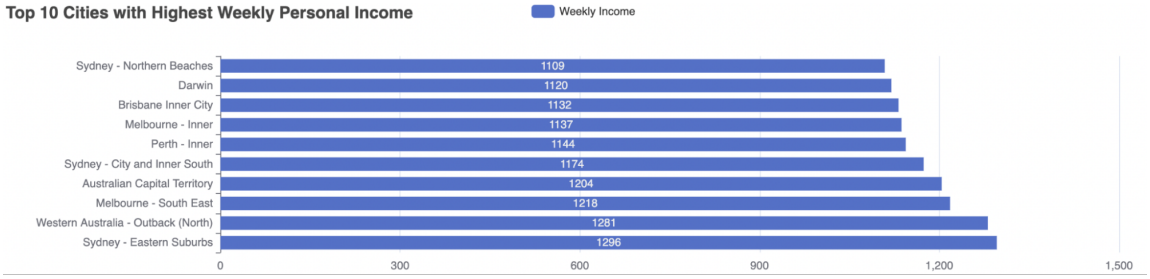
Figure 16: Average Weekly Median Personal Income by Location

By examining the distribution of post frequency for areas around Melbourne (figure 20), it is observed that Geelong and Inner South display a relatively high average frequency of food-related posts. For example, out of 1000 individuals in Inner South Melbourne, approximately 92 tweets were found to mention food. Further investigation indicates that these two locations are popular vacation places within the region. For example, St.Kilda in Inner South and Great Ocean Road

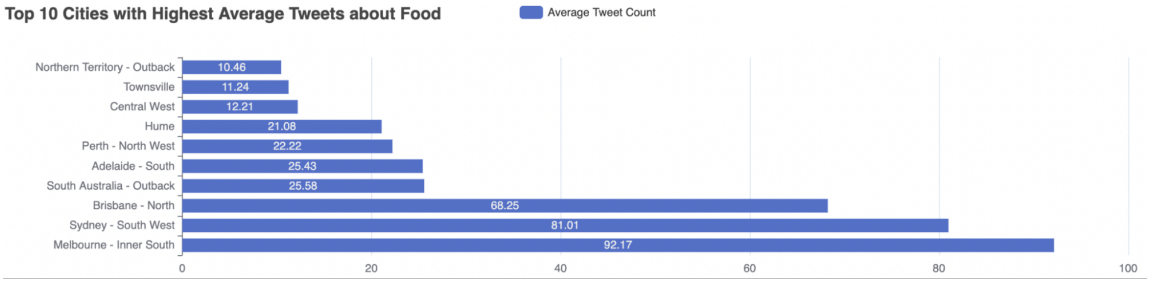
in Geelong are famous attractions. Hence, the higher frequency of posts in these areas could be attributed to the special characteristics and appeal of these places.



(a) Top 10 cities with highest sentiment towards food



(b) Top 10 cities with highest average tweets about food



(c) Top 10 cities with highest weekly personal income

Figure 17: Top 10 Cities with highest Sentiment, Tweet Frequency, and Income

Furthermore, by analyzing the bar plots regarding places with the top sentiment and frequency of post, it is found that individuals in major cities or popular vacation destinations tend to have more positive attitudes and higher posting frequencies regarding food.

However, when analyzing the income distribution in both bar plot and map, no significant patterns are observed, which means the average income levels in major cities do not differ a lot. Therefore, there is no evidence to support that people's attitudes towards food are related to their income levels.

In conclusion, two interesting stories emerge from the analysis of food-related tweets. First, places with certain characteristics, such as popular attraction places or major cities, will have a stronger affinity towards food, as evidenced by the higher posting frequency and more positive sentiments. Second, people's attitudes towards food are not correlated with their income levels. Higher income doesn't necessarily result in increased posting frequency or higher sentiment towards food.

9.2 Scenario 2: Vehicle (Traffic)

Traffic and vehicles are topics that are closely related to daily life. Among English tweets containing location information, 3.2% of them discuss vehicles. Similarly, three key statistics are selected to facilitate the analysis of this topic: average sentiment towards vehicles, average frequency of vehicle-related posts and the average number of motor vehicles per dwelling (SUDO, 2020). The inclusion of the average number of motor vehicles per dwelling aids in uncovering the underlying factors influencing people's attitudes towards traffic. These statistics collectively provide valuable insights into people's perceptions regarding vehicles and traffic.

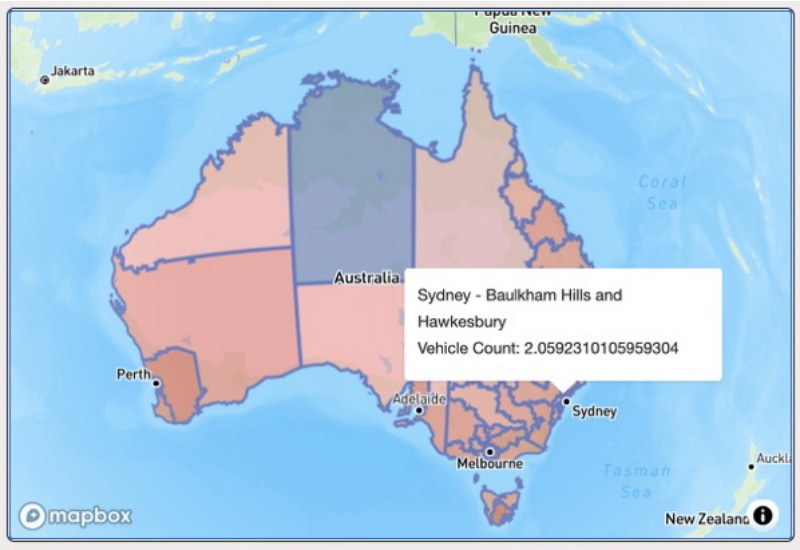


Figure 18: Average number of motor vehicles per dwelling by location

The distribution of the average number of motor vehicles on the map aligns with the overall economic development of Australia, with coastal areas reflecting higher levels of development and consequently having a higher average number of motor vehicles.

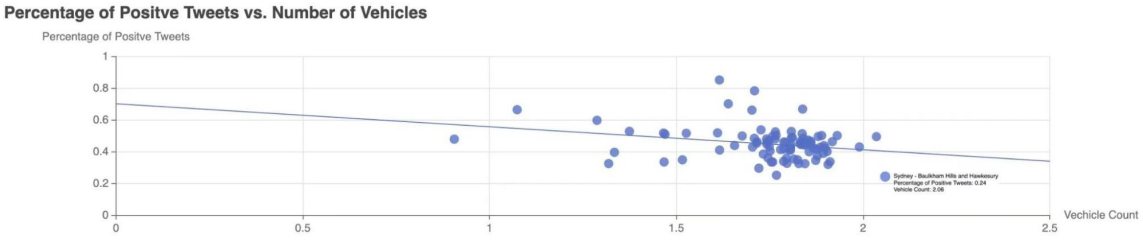


Figure 19: Percentage of positive tweets vs. Number of vehicles

A scatter plot is derived from the analysis, with the x-axis being the average number of motor vehicles, and the y-axis representing the sentiment towards vehicles or traffic in each SA4 area.

Overall, although the correlation is not strong, with a majority of data clustering around an average of 1.6-1.8 vehicles, there is still an approximately negative relationship between the two features.

In conclusion, this key finding suggests that as the number of motor vehicles increases, people tend to become more pessimistic about traffic. This could be attributed to the fact that individuals with more vehicles are more concerned about the traffic situation, leading to a greater likelihood of expressing complaints towards traffic conditions.

9.3 Scenario 3: Sport

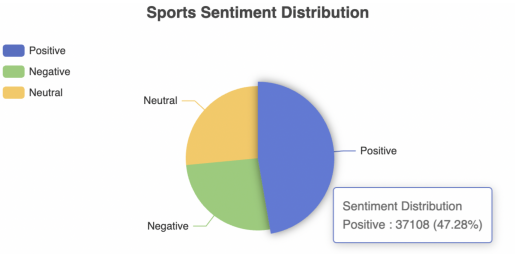


Figure 20: Sport Sentiment Distribution

Figure 21: Word Cloud of Sport

Last, sport is another popular topic to discuss, and there are approximately 3.1% of tweets mentioning sport. The utilization of LDA topic modeling also reveals several popular sports-related topics in Australia, such as skateboarding, kayaking, and surfing. Furthermore, among these sport-related tweets, an intriguing finding reveals that nearly half of them, 47.28%, exhibit a positive sentiment. This implies that when people mention sport, there is a trend towards positivity, suggesting that sport has the ability to evoke positive emotions and energy among individuals.

9.4 Mastodon Scenario: Sentiment Comparison between Toots and Tweets

To gain deeper insights into the extent of discussions on various topics among Australians, data from different social media platforms, Mastodon, was analyzed. Unlike pre-collected Twitter data, Mastodon toots data is continuously streamed. Therefore, direct comparison of data between the two social media platforms in terms of quantity is challenging due to the difference in time frame. However, the percentage of sentiment regarding different topics between the two platforms can still be compared.

The distribution of sentiments between Mastodon toots and Twitter tweets are vastly different. Figure 22 represents the sentiment distribution of toots and figure 23 shows the sentiment distribution of tweets.

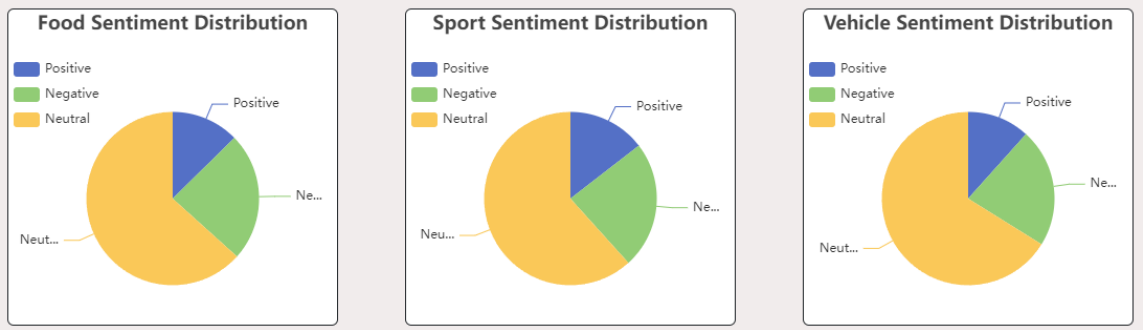


Figure 22: Mastodon Toots Sentiment Distribution

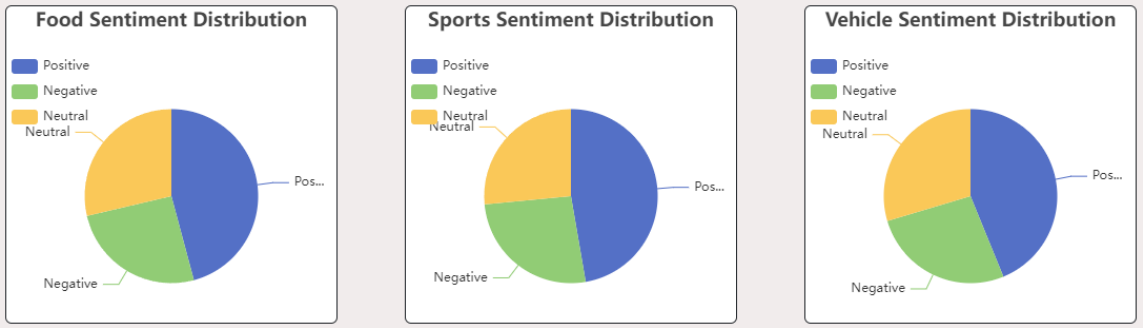


Figure 23: Twitter Tweets Sentiment Distribution

As a general trend, people on Mastodon tend to express more neutrally than people on Twitter, with over 60% of toots being neutral. Moreover, regarding non-neutral sentiments, Mastodon users tend to express more negatively, with almost twice negative toots than positive ones for each topic. These findings suggest that there are distinct differences in sentiment expression between the two platforms. Mastodon users lean towards a more neutral or negative sentiment, while Twitter users exhibit a greater tendency towards positive sentiment. The reasons behind these differences need further investigation and analysis.

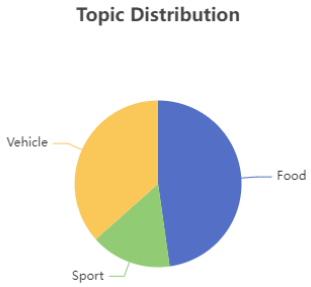


Figure 24: Mastodon Topic Distribution

Overall, the topic distribution of the three scenarios is outlined in the pie chart (Figure 19). Food-related toots constitute the largest proportion in mastodon, with vehicles the second. The sport takes the smallest proportion of Mastodon discussion at around 15%. This overall trend gives an

understanding of what information the data conveys and in what distribution regarding our concerned scenarios.

10 Frontend

10.1 Frontend Architecture

ReactJS is used as the main frontend framework with package manager pnpm. Many packages are used to aid the development and the visualization of the data collected. The main packages included are:

- React Bootstrap
- React Echarts
- React Wordcloud
- React Mapbox GL
- Axios for Restful API

Other supporting and less important packages are not listed here and a full list of packages for frontend can be found in "frontend/package.json". A general overview of the important skeleton of frontend code structure is as follows:

Figure 25: Frontend Directory Structure

10.2 Restful Infrastructure

Frontend maintains the passing and receiving of data with the Restful design. As some of the display data is streamed from real time collection, there is a need to constantly retrieve resources from CouchDB. For example, in this project, Mastodon toots are constantly being collected by the mastodon harvester and the toots information in CouchDB is constantly being updated. Frontend communication is managed by Axios, a common frontend API that adopts the Restful design

philosophy. Requests are sent from the frontend by users to the backend and asks for data such as the total sentiment distribution of Mastodon toots. Backend will handle requests accordingly and return the desired resources. Backend has some data stored within its infrastructure and some retrieved from CouchDB on requests.

Frontend, when requesting resources, will always reach to the backend and never CouchDB directly. The reason behind this decision is security. CouchDB authentication requires a username and password pair. Storing this key pair in the frontend directly is insecure. Making users enter them is inconvenient and inappropriate for a data visualization webpage. Backend acts as a middleware in this case that defines user actions rigidly and protects the application from inappropriate user usages.

10.3 Mapbox

Mapbox is a perfect tool for visualizing data with geolocation. During the development process, it is discovered that it also has strict requirements of the input data structure. There are many specific data preprocessing endeavors to make map data to display at its best performance and clarity.

The data for Statistical Area Level 4 (SA4) region data provided by the Australian Bureau of Statistics comes in shapefile format. This data needs to be converted into geojson format to fit the need of twitter data location display on the map. The geopandas python package is used to convert the shapefile data into geojson data.

Geojson data structure consists of different fields, one most important field is the "geometry" field that defines the polygon on which the SA4 region is positioned at. The geometry data is used to draw the bounding boxes on Mapbox that clearly separated each region from another.

Furthermore, to visualize the data, each SA4 region needs its data value stored in the "properties" field. This field originally consists of the SA4 code name, text name, and many others. To add data onto the map, extra data needs to be inserted into the "properties" field. One example of final processed geojson at the properties field looks like Figure 26.

The last 6 entries are numbers analyzed from the twitter data that needs to be displayed on Mapbox accordingly, with higher values represented with a darker color of the statistical area on the map.

One more discovery during implementation is that the map data often takes a long time to load. This is likely due to the size of the file. The geojson file alone consists of hundreds of megabytes of data. Mapbox real time rendering of this data is a huge burden and significantly slows down the data loading process and makes the navigation at frontend feel unsmooth. The solution adopted was to use the Visvalingam weighted area algorithm to make the appearance smoother by potentially removing over-detailed points at the vertex with acute angles. The resizing parameter is set to be 5%. After resizing, the resized geojson file is compressed to around 3 megabytes, while still maintaining

```
"properties": {
  "SA4_CODE21": "109",
  "SA4_NAME21": "Murray",
  "CHG_FLAG21": "0",
  "CHG_LBL21": "No change",
  "GCC_CODE21": "1RNSW",
  "GCC_NAME21": "Rest of NSW",
  "STE_CODE21": "1",
  "STE_NAME21": "New South Wales",
  "AUS_CODE21": "AUS",
  "AUS_NAME21": "Australia",
  "AREASQKM21": 97796.4898,
  "LOCI_URI21": "http://linked.data.gov.au/dataset/asgsed3/SA4/109",
  "food_map1": 0.0024798387096774195,
  "food_map2": 724,
  "sport_map1": 0.0012298387096774193,
  "sport_map2": 0.4672131147540984,
  "vehicle_map1": 0.001310483870967742,
  "vehicle_map2": 1.7652042613874674
}
```


Figure 26: Geojson File Properties

the overall details of the polygons. As we can see, less detail is present in Figure 28 compared to Figure 27, while the general shape is almost identical.

Figure 27: SA4 Border Around Melbourne Pre-Conversion

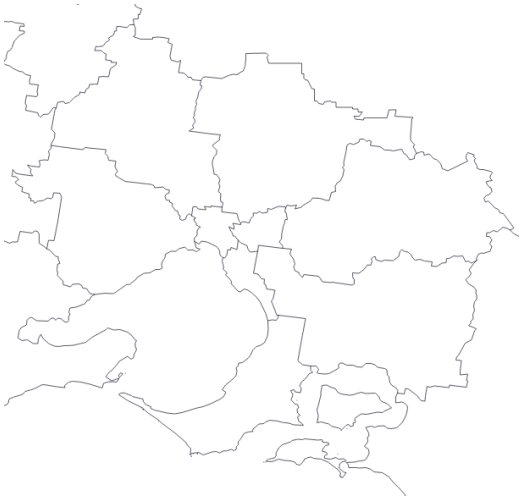


Figure 28: SA4 Border Around Melbourne Post-Conversion

11 Backend

11.1 Architecture

In this project, the backend plays a relatively simple role. It is merely serving as a bridge to orchestrate various system components. Its fundamental ability is to route requests and provide APIs for data and pages to clients. Express.js, a fast and minimalist web framework for Node.js, is our chosen framework, primarily due to its compatibility with the frontend and its ability to facilitate seamless communication with other components, adhering to the RESTful API. A crucial component facilitating interaction with CouchDB is the Nano package, which is officially supported by the CouchDB and Node.js teams. This underscores its reliability and ensures a smooth interface between our backend and the database.

A core feature of our backend architecture is the adoption of the Model-View-Controller (MVC) design pattern, renowned for its clear separation of concerns. The MVC pattern is a software design pattern that partitions the related program logic into three interconnected elements. The Model represents the application's data structure, typically in terms of the database schema and operations. The View corresponds to the user interface and visualizations (In our case, it is not used). The Controller controls the interaction between the data from the Model module and the rest of the system. Employing the MVC has several benefits for our backend implementation. It provides a clear structure and separation of concerns, which simplifies the process of understanding and maintaining the codebase. Each component has its specific role, and changes in one rarely impact the others. This decoupling results in increased modularity, leading to enhanced scalability.

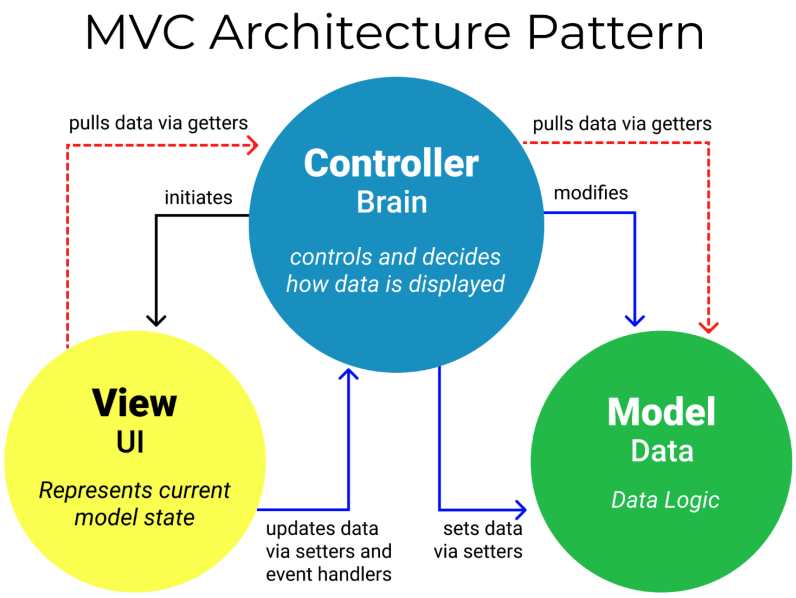


Figure 29: Backend

11.2 Infrastructure

Given that this project implements a comprehensive suite of Python programs for data harvesting, preprocessing, and analysis, there is minimal need for intensive Create, Read, Update, and Delete (CRUD) operations to be performed by the backend. Instead, it primarily focuses on reading the data and providing APIs. Listening for requests from other components on a designated port, underpinning the overall operation of our system.

12 Functionality

The functionalities of the web application can be categorized into two main aspects: Data Analysis based on the pre-recorded ADO and SUDO data with respect to each of the topics and Comparison of data from Twitter and live-streaming data from Mastodon.

As for Data Analysis of Australian Life, a detailed example is shown in section 9. However, the theme and the outcome of the analysis can be varied according to the purpose of the viewer. For instance, since the map boxes are able to exhibit data with respect to each SA4 area, the viewer can easily observe patterns or differences between values of particular features by the color of the regions on the map (e.g. Income level of an area and number of tweets per person about the scenario). Besides, the overall trends and top ten areas with respect to particular features also provide obvious useful information. One limitation worth mentioning is that for the topics modeled by LDA in the word cloud, the number of topics should be freely tuned by the viewer and the word cloud should present the result after the LDA model is trained according to the number of topics. This helps the viewer captures more insights from the keywords in those topics. And the same improvement is also applicable for the Top N areas with respect to particular features where N is the hyperparameter. Moreover, it should also allow users to plot the scatter plot according to their requirements.

The crucial achievement of the live-streaming data from Mastodon is that the distribution of sentiment related to each topic and the number of toots classified to each topic from Mastodon can vary from time to time. The main server to be listened to is Aus_social and the harvesters are able to scale up and listen to other servers by the Ansible scripts. For illustration, one example is that the viewer can save the data of Mastodon from the web at some time first. After a period of time, if some significant sports events are held, the number of toots mentioning sport may increase obviously compared to the other two topics and the distribution of sentiment for Sports may also vary in a relatively large degree compared to previously. Therefore, the viewer can research the degree to which people actively participate in the discussion of the events and their potential attitudes towards the events.

13 Limitation

13.1 Ansible

Ansible limitation see section 2.2

13.2 Docker

Docker limitation see section 5.5

13.3 Data

Topic Classification limitation see section 6.2.4

13.4 Mastodon Harvester

Mastodon Harvester limitation see section 7.5

14 Appendix

Github: https://github.com/terrance2630/CCC_A2.git

Youtube video: https://youtu.be/oCYwG_Q5i6k

Front end: http://172.26.128.31:3000/ (Access with Unimelb VPN)

15 Individual Contribution

Name	Contribution
Bowen Fan	Data Analyst Front-end developer
Guanqin Wang	Ansible Back-end developer Fault Tolerance
Junran Lin	Data Analyst NLP preprocesser Scenario writer
Tianqi Wang	MRC Ansible Docker Mastodon Harvester
Yi Yao	Data Preprocess CouchDB Scenario writer

Figure 30: Individual Contribution

References

AU-Govt-ABS-Census (2021a). Number of motor vehicles by dwellings. https://sudo.eresearch.unimelb.edu.au.

AU-Govt-ABS-Census (2021b). Selected medians and averages. https://sudo.eresearch.unimelb.edu.au/.

AU-Govt-DESE (2021). Dese - labour market - population by age group (sa4) december 2021. https://sudo.eresearch.unimelb.edu.au.

Australian Bureau of Statistics (2021). Australian statistical geography standard (asgs): Volume 1 - main structure and greater capital city statistical areas, july 2021. Accessed: 2023-05-25.

Docker (2019). Swarm mode overview. https://docs.docker.com/engine/swarm/.

Gillis, A. (2020). What are containers (container-based virtualization or containerization)? *SearchITOperations*.

IBM (n.d.). What is docker? https://www.ibm.com/topics/docker.