2023 Cluster & Cloud Computing Assignment 2
Australia Social Media Analytics on the Cloud

Social Sense Dashboard

Zongchao Xie[1], Xuan Wang[2], Wei Zhao[3], Runqiu Fei[4], & Sunchuangyu Huang[5]

[1]Student ID: 1174047, Email: zongchao.xie@student.unimelb.edu.au
[2]Student ID: 1329456, Email: xuan.wang19@student.unimelb.edu.au
[3]Student ID: 1118649, Email: zhao.w2@student.unimelb.edu.au
[4]Student ID: 1166093, Email: runqiu.fei@student.unimelb.edu.au
[5]Student ID: 1118472, Email: sunchuangy.huangh@student.unimelb.edu.au

Content

1 Introduction

The advent of digital transformation has presented an immense opportunity for the development and understanding of various societal aspects. The convergence of cloud computing, social media analytics, and data science has opened up exciting vistas for insight generation and policy formulation. Social media platforms, such as Twitter and Mastodon, constantly capture the pulse of society, embodying a real-time and organic reflection of public sentiment, preferences, and attitudes. In parallel, other significant repositories of structured data like the Spatial Urban Data Observatory (SUDO) can augment and enrich these social media insights, providing a multifaceted view of societal realities.

This project aims to leverage these digital resources - Twitter corpus, Mastodon data, and data from the SUDO platform, dive into the life in Australia. Specifically, the project will explore how the combined these sources can inform our understanding of the country's educational status and income distributions. Use UniMelb Research Cloud (MRC) to develop a cloud-based solution capable of harvesting, storing, and analysing large scale data to draw meaningful inferences.

1.1 Motivation

In this project, we are focusing on two key scenarios:

- The first involves studying how a person's age, sex, and income level interact with each other. By analyzing tweets related to income and job satisfaction, we hope to gain a better understanding of the complexities of these relationships. We're particularly interested in seeing how these discussions vary between areas with high and low median incomes.
- The second area involves investigating the relationship between crime rates and discussions about crime and safety on Twitter. We aim to see if the sentiment of these tweets changes based on the crime rate in the location from which they were sent.

Income is a major factor that influences a person's quality of life and social standing. It's tied to social mobility, trends in the job market, and economic policy. By studying discussions about financial and income related topics on Twitter and Mastodon, we can learn more about the realities such as income distribution in society.

Meanwhile, crime rates and perceptions of safety have a big impact on society. By studying the relationship between crime rates and discussions about crime and safety on Twitter and Mastodon, we can gain valuable insights into how people perceive safety in their communities.

This project aims to use social media data to gain a deeper understanding of the relationship between income and crime in Australia. By analyzing the sentiment of tweets related to income and crime, we aim to provide a detailed and data-informed view of these important societal issues. The insights we gain from this work could be very valuable for policymakers and other stakeholders as they make decisions to address these issues.

1.2 Objectives

In line with the motivation highlighted, our primary goal in this project is to construct a comprehensive cloud computing solution using the IaaS provided by MRC. This solution is geared towards effectively harnessing, analyzing, and visualizing the insights gleaned from social media pertaining to education and income scenarios in Australia.

The proposed cloud solution comprises five main components:

- A backend server developed using Flask, a popular Python web framework, which will handle data processing and communication tasks.
- A frontend server built with React.js to facilitate interactive and dynamic data visualization.
- A NoSQL database established with CouchDB to store and manage the social media data. The database will leverage MapReduce, a programming model designed for handling big data sets, to execute analytics tasks.
- A data processing and analysis module built using the Message Passing Interface (MPI), which will aid in managing the concurrent tasks and optimizing the performance of the solution.
- Docker, a platform that employs containerization technology, to ensure each part of our solution is neatly packaged, thus enhancing its portability and scalability.
- Ansible, an open-source automation tool, to streamline the deployment of our solution and effectively manage the scalability issue across the cloud infrastructure.

By integrating these components, we aim to deliver a conceptual product, which provide a scalable and efficient, all in one web based service that can process, analyze, and present large scale social media data, thereby offering valuable insights into societal patterns in Australia.

1.3 Team Roles

In our project team, each member carries out specific roles aligned with their major and areas of expertise, fostering a collaborative and efficient workflow.

- **Xuan Wang (Database)**: Xuan will be primarily responsible for managing the database. He will use his software engineering expertise to handle the storing, retrieval, and management of data in CouchDB. Xuan will also take on the responsibility of using Ansible to automate the deployment and management of the entire cloud-based system.
- **Wei Zhao (Ansible & Backend)**: Wei will focus on the development of the backend server using Flask. This involves establishing the server, managing data processing tasks, and ensuring effective communication between the frontend, backend, and the database. Wei will also be working with Ansible, similar to Xuan, ensuring the seamless deployment and scalability of the system.
- **Zongchao Xie (Scenario Analysis)**: Zongchao will work on scenario analysis. His role involves deriving valuable insights from the collected data. By applying his data science skills, he will analyse and interpret the social media data to better understand the societal implications regarding education and income in Australia. He will also assist in the data processing tasks.
- **Runqiu Fei (Data Processing)**: Runqiu will be focused on data processing, which includes cleaning, transforming, and preparing the data for analysis. This role is crucial to ensure the quality of the data being analyzed. Additionally, Runqiu will also be assisting Xuan in managing the database, specifically in ensuring the data is correctly stored and easily retrievable.
- **Sunchuangyu Huang (Frontend)**: Sunchuangyu will be responsible for building the frontend server using React.js. His role will be vital in creating interactive and dynamic data visualizations for the end users. In addition to this, Sunchuangyu will also be working on data processing tasks, building customised API tools, aiding in the preparation and transformation of the data for analysis.

1.4 Report Structure

This report begins with a thorough appraisal of the tools utilized in our project, spotlighting both their strengths and limitations, with particular focus on two SUDO scenarios, income and crime. Following this, we describe the key functionalities of our system and elaborate on its underlying design and architecture. Next, we explain the data processing to handle the social media data across different resources. Afterward, we present a user guide to help users effectively engage with and benefit from our cloud service. The report further showcases certain supported scenarios, providing real-world examples of how our system can be used for analysis. In the conclusion, we summarised the essence of our work, its potential limitations, and future possibilities for development.

2 Melbourne Research Cloud

The Melbourne Research Cloud (MRC) stands as a cloud service provider, hosted by the University of Melbourne. It offers an array of on-demand computing resources, extending its services not only to students and researchers, but also to various business partners. Like other cloud platforms, like Amazon Web Services and Google Cloud Platform, functioning as an Infrastructure as a Service (IaaS) provider that facilitates the development of customized cloud solutions for a variety of tasks, such as data analysis and web hosting.

The MRC service has about 20,000 virtual cores, advanced GPU capabilities, private networking, and load balancing. This solid foundation ensures that it can meet diverse requirements and deliver a consistently high-quality experience for its users.

The primary advantage of the MRC lies in its cost-effectiveness. Unlike other commercial cloud platforms such as AWS or Google Cloud, where pricing models can be confusing and subject to change, MRC maintains a transparent and stable cost structure. Being a publicly-funded service and part of the Australian Research Cloud (Nectar), it maintains accessibility and affordability for a wide spectrum of research groups. This non-commercial stance greatly reduces the risk of sudden price surges, allowing researchers to concentrate on their work without the distraction of unpredictable costs.

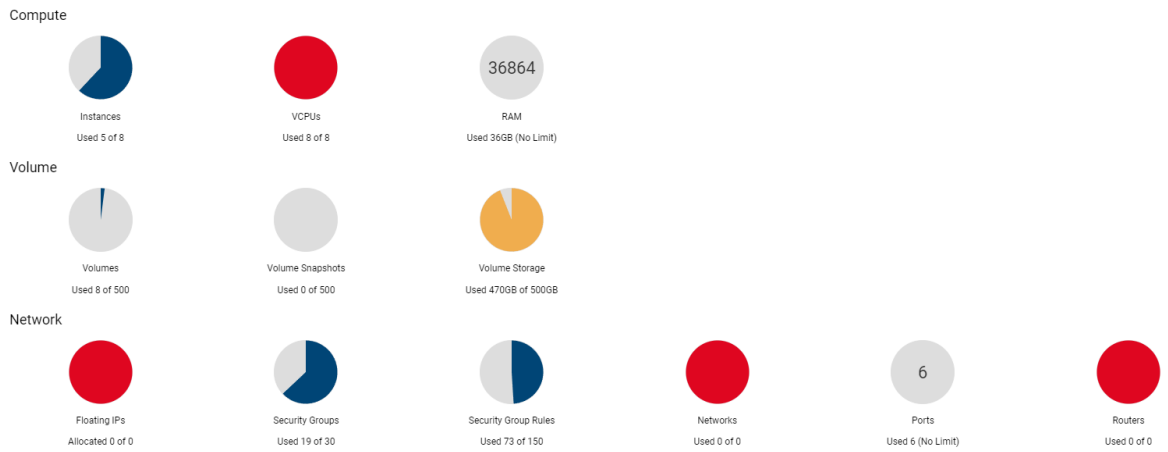

Figure 1: Melbourne Research Cloud COMP90024 2023 Group 57 Security Overview

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
db-worker 2	NeCTAR Ubuntu 22.04 LTS (Jammy) amd64	172.26.128.114	uom.mse.1c4g	cloud_backend	Active	melbourne-qh2-uom	None	Running	16 hours, 9 minutes	Create Snapshot
db-worker 1	NeCTAR Ubuntu 22.04 LTS (Jammy) amd64	172.26.133.214	uom.mse.1c4g	cloud_backend	Active	melbourne-qh2-uom	None	Running	1 day, 18 hours	Create Snapshot
db-master	NeCTAR Ubuntu 22.04 LTS (Jammy) amd64	172.26.134.248	uom.mse.2c9g	cloud_backend	Active	melbourne-qh2-uom	None	Running	5 days, 18 hours	Create Snapshot
swarm-worker1	NeCTAR Ubuntu 22.04 LTS (Jammy) amd64	172.26.134.216, 172.26.128.118	uom.mse.2c9g	cloud_backend	Active	melbourne-qh2-uom	None	Running	4 weeks	Create Snapshot
swarm-master1	NeCTAR Ubuntu 22.04 LTS (Jammy) amd64	172.26.130.83	uom.mse.2c9g	RunqiuFei	Active	melbourne-qh2-uom	None	Running	4 weeks, 1 day	Create Snapshot

Figure 2: Melbourne Research Cloud COMP90024 2023 Group 57 Instances Configuration

Usability is another strength of MRC. It features an intuitive user interface and boasts a supportive user community. Users can effortlessly create virtual instances either by engaging with the web application or by automating the process using IT automation tools, such as Ansible. This simplicity not only facilitates user engagement but also enhances the overall productivity of the cloud platform.

Lastly, MRC showcases remarkable efficiency in resource allocation and scalability. The platform empowers users with the capability to scale their computing resources in alignment with their specific needs, ensuring optimal utilization of the cloud environment. Such flexibility help users to effectively manage their resources while reducing wastage.

2.1 Resource and Instances

In accordance with the project specification, our team has been granted access to 8 virtual CPUs and 500GB of storage space. For the development of this project, we strategically allocated all CPUs and approximate 470GB of storage capacity to 6 instances. All resources could be leveraged by Ansible automation scripts.

As indicated in Table 2 and Figure 1, we have assigned 4 CPUs for the CouchDB clustered database, which also employs 400GB of storage volume. The master database instance is assigned 2 CPUs and 1 CPUs for two database slave instances, they all share the designated storage space. The flask backend has been allocated 2 CPUs, and 2 CPU for React.JS frontend. This balanced allocation of resources aids in maintaining efficient performance across different components of our project while also ensuring we have the flexibility for scalability testing.

2.2 Security Groups

Per the instructions laid out in the project guidelines, we granted remote instance access by opening port 80. To align with customary settings and optimize the functioning of various components, we have the following ports configuration:

- For ReactJS, a widely used JavaScript library for building user interfaces, we have opened port 3000.
- For Flask, a lightweight Python web framework, we have designated port 8080.
- For CouchDB, a scalable and high-performance open-source document-oriented database, we have allocated port 5984.

These port settings ensure smooth communication between the various parts of our system and facilitate the overall execution of our project.

Direction	Ether Type	IP Protocol	Port Range	Description
Ingress	IPv4	TCP	80 (HTTP)	-
Ingress	IPv4	TCP	443 (HTTPS)	-
Ingress	IPv4	TCP	3000	frontend/react
Ingress	IPv4	TCP	5984	database/couchdb
Ingress	IPv4	TCP	8080	backend/api

Table 1: Melbourne Research Cloud COMP90024 2023 Group 57 Security Groups

3 System Design and Architecture

The team is comprised of two students from diverse major fields, thereby allowing us to leverage the strengths of each major in designing our cloud-based system. In order to meet the challenging requirements, we employ Ansible for the automatic deployment of frontend, backend, and database. This deployment is equipped with a scaling capability, tailored to user-defined requirements.

The system uses RESTful requests to manage data transfers and trigger another mastodon harvester. Given this requirement, we opted for Flask - a lightweight Python framework - owing to its suitability for handling these tasks. React.JS was chosen as the frontend because of our extensive familiarity with it.

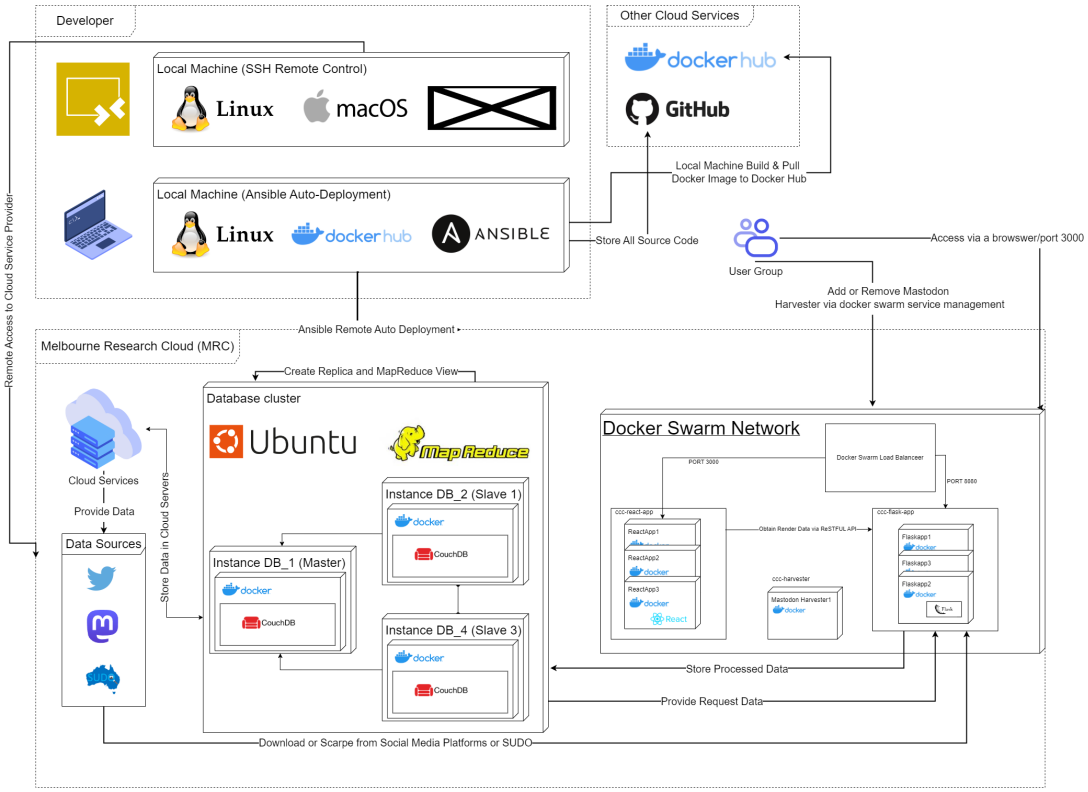

Figure 3: Group 57 Social Sense Dashboard System Design Graph

Our codebase is stored on GitHub, adhering to conventional practices for Continuous Integration and Continuous Deployment (CI/CD). The summarised design of the system can be seen in figure 3.

3.1 CouchDB

CouchDB, an Apache-developed NoSQL database, is recognized for its scalability and multi-master replication capabilities. It employs a document-oriented model, housing data in JSON-like documents. This offers an adaptable, intuitive mode of data representation. Paired with its powerful map-reduce views and peer-based replication, CouchDB is engineered for user convenience.

Compare it with other NoSQL database like MongoDB, CouchDB's distinct advantages lie in its exceptional conflict resolution mechanism that proves crucial in distributed systems, as well as its intrinsic multi-master synchronization feature. Additionally, CouchDB utilizes a RESTful API, permitting direct interaction over HTTP with all its database features.

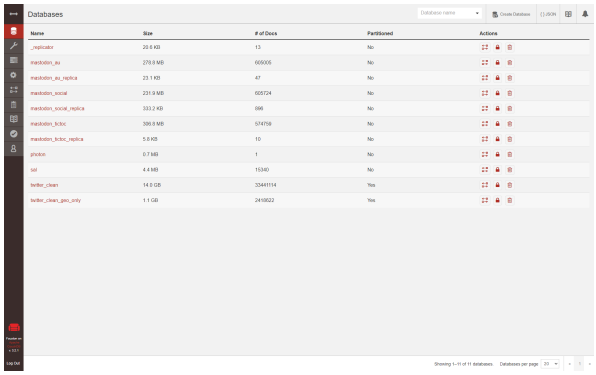

Figure 4: CouchDB Fauxton

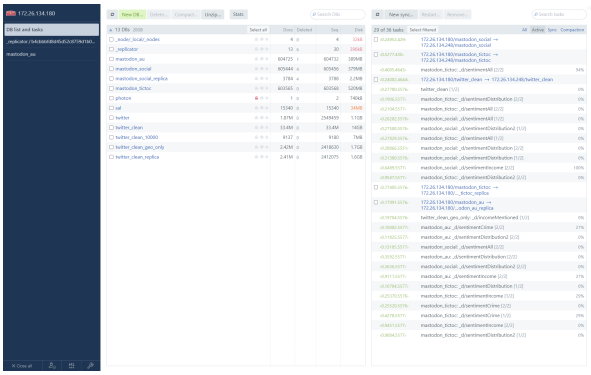

Figure 5: CouchDB Photon

The primary shortcoming of CouchDB is its slower data retrieval speed, particularly for larger datasets, as opposed to MongoDB. To counteract these limitations, we adopt the map-reduce strategy, a programming model and computational technique for processing and generating large datasets. This approach strategically breaks down the data, paralleling the distributed nature of NoSQL databases, thereby assisting in efficient querying and data processing. The employment of this divide-and-conquer methodology is particularly advantageous for our clustered database, optimizing our handling of large datasets.

3.1.1 MapReduce

As indicated earlier, we leverage the map-reduce programming model in our data processing pipeline, particularly as the first level of data processing on the complete database. Map-reduce plays a critical role in processing large volumes of data and contributes significantly to the overall efficiency of data management in CouchDB.

In CouchDB, map-reduce is utilized in the creation of views or secondary indexes. The 'map' function filters and sorts the data, creating an index of documents that satisfy certain conditions. The 'reduce' function then processes these indexed documents to provide summarized results. Although the initial creation of the map-reduce views can be time-consuming, once the view is built, subsequent updates and retrievals are significantly faster as the view is incrementally updated with changes in the data. The processing speed is therefore drastically increased, especially for large volumes of data.

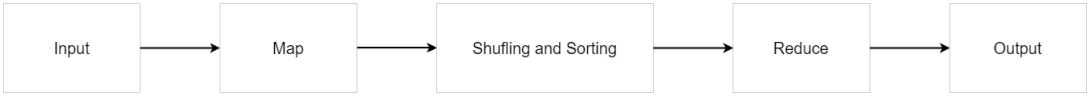

Figure 6: MapReduce Process

The integration of the map-reduce model is a strategic choice for our system, especially given the large volumes of data we are handling. Despite the time-consuming view creation process, the resulting increase in data processing speed is a worthwhile trade-off, making map-reduce an essential component in our data processing pipeline.

3.2 Data Security

To uphold data security, we leverage CouchDB's default backup strategy which inherently establishes 2 shards and 3 replicas. This strategy not only ensures robust data security but also implies a significant storage space requirement. Given that data is stored on each instance, it can easily occupy around 400GB of storage space (including original data and replicas).

In CouchDB, the concepts of shards and replicas are fundamental to the storage and replication strategy, enabling not just data security but also fault tolerance and enhanced performance. Sharding, a technique of distributing data across multiple machines, divides a database into smaller parts known as shards. Each shard operates independently, holding a portion of the database's data. On the other hand, replicas are essentially copies of the shards. In our case, CouchDB creates 3 replicas, meaning that each shard is replicated three times.

The primary purpose of these replicas is to ensure data durability and availability. In case of a node failure or network issues, having multiple copies of data ensures that data is always accessible. Therefore, these replicas contribute significantly to data security, but also require additional storage capacity.

3.3 Clustered Database

In our project, we employ a clustered database architecture, comprising of one 2-core master and two 1-core slave instances (table 2). This configuration brings forth multiple advantages including enhanced performance, improved availability, and more effective utilization of resources.

A clustered database effectively divides tasks among the instances, which significantly reduces the load on each individual server and bolsters overall performance. The master-slave arrangement also provides robust fault tolerance. If one node fails, the others can take over, ensuring data remains accessible and services uninterrupted.

To form a cluster database, we use containerization method to facilitates the setup. Containers encapsulate the database instances along with their dependencies, making it easier to deploy across multiple servers. It simplifies the replication of instances, scaling up, and maintaining a consistent environment. To tackle the scaling issue, we could simply create a new instance based on DockerFile, then join the cluster via Ansible automation scripts.

Regarding data security issues, in the cluster database, each instances only holds a fraction of the total data because of MapReduce and sharding strategy spread data across the nodes. Moreover, replicas ensure data security to prevent events like node failure.

3.4 External Tool: Photon

For the administration of the database, we employ a third-party tool named Photon. Compared to the native dashboard, Photon simplifies database configuration and provides a more user-friendly interface to monitor backend processes. This allows us to efficiently manage the database and maintain a keen eye on the system's performance, thereby ensuring smooth operation of our services.

3.5 Flask Backend

The Flask App serves as our backend, offering key advantages that are pivotal to our project's success: it is lightweight, flexible, and Python-based. As a Python-based framework, Flask allows for seamless integration with an array of Python tools and libraries. This is crucial in our case as we make use of database connectors, data processing libraries, and JSON processing tools, which together form a robust support system for our backend services.

The design of Flask permits developers to conduct experiments and create prototypes within the Jupyter Notebook environment. This approach simplifies the development process as these codes can then be readily transformed into functions within the Flask App, enabling an uncomplicated transition from data processing to web services.

Our backend service,concentrates on three main tasks: retrieving data from the database, processing this data, and creating JSON data required for graphics rendering. The resulting JSON data is subsequently forwarded to the front end for rendering.

In pursuit of enhanced efficiency and minimized network bandwidth usage, we employ gzip to compress the generated JSON files before storing them locally. When a request is received from the front end, we first check if the relevant compressed data is already available locally. If it is, we directly transmit this data. If it's not, we initiate the corresponding data generation function. This strategy not only significantly reduces network bandwidth consumption but also promotes faster server response times.

3.6 React Web Application

In the frontend, we have opted for the React.JS library due to its reputation for efficiency and flexibility in building interactive user interfaces. Our goal for the frontend is to allow users to select summarized information provided by the team and interact with various plots. To accomplish this, we have incorporated the use of Plotly, a graphing library that supports multiple programming languages such as Python and JavaScript.

To streamline the process, we have chosen to generate informative plots using Python and store the graph JSON data. Instead of processing the data on the frontend, we utilize the backend to pass the data to the frontend for display.

On the left panel, we designed to provided summary statistic from Mastodon, Twitter and SUDO, each section will have summary statistics plot and a summary section. On the right panel, we have two map plots for twitter and sudo data. Since no geo-location could found from any Mastodon servers hence we only display map for two data sources.

The user interface has been designed with a two-panel layout. The left panel displays summary statistics from Mastodon, Twitter, and SUDO. Each section in this panel includes a plot showcasing the summary statistics as well as a corresponding summary section.

In the right panel, we have incorporated two map plots, one for Twitter data and another for SUDO data. Due to the absence of available geolocation data from Mastodon servers, we are only able to display maps for the two mentioned data sources.

We aim to provide users with an all-in-one experience on the frontend, eliminating the need to navigate between different web pages or programs. Through a single dashboard, users can access and analyze the most relevant and valuable information pertaining to the problem at hand.

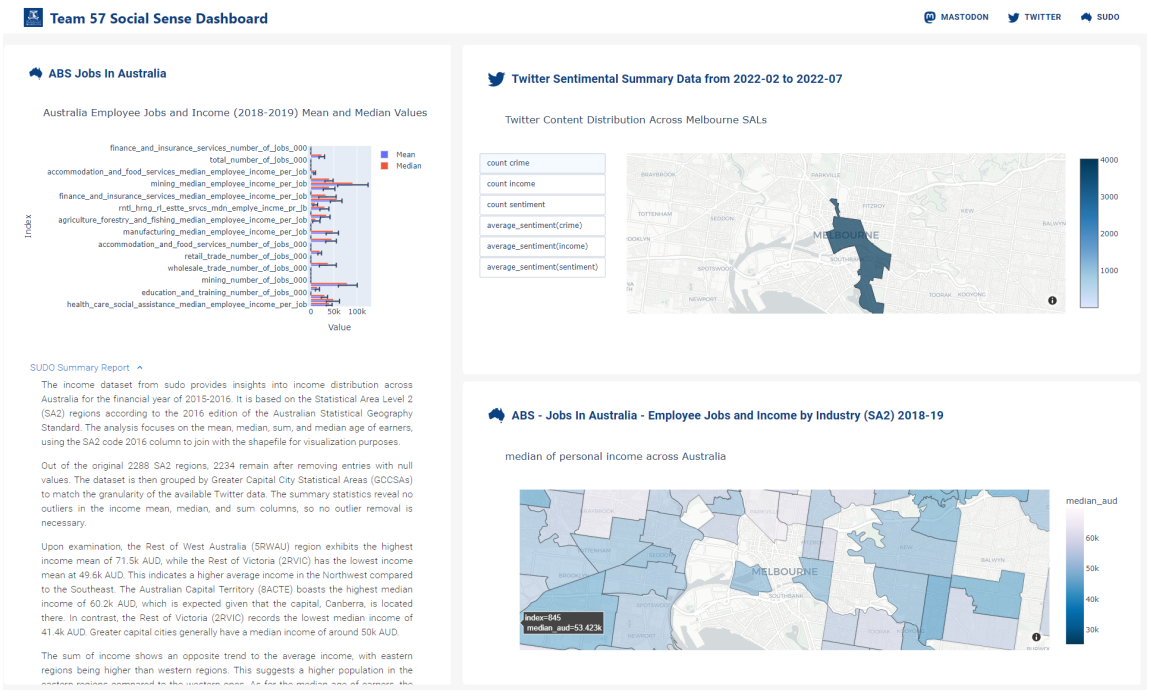

Figure 7: React.JS Frontend Design

3.7 IT Automation with Ansible

This section provides a detailed explanation of how our team leverages the combination of Ansible and Docker container technology to enhance application deployment in MRC.

Using Ansible, we utilize our local machines as control nodes to control the behavior of remote servers through pre-defined playbooks. Meantime, Docker container technology helps us achieve application portability and scalability.

3.7.1 Dockerization and Docker Swarm

During the application development process, Docker plays crucial role in it. The most important advantage to apply Docker in the project are:

- **Consistent**: Docker provides a highly consistent development and testing environment. The Docker image, built by developers based on the current repository's code, contains all the necessary environment for running our application, whether it is front-end, back-end, or database. Docker ensures that the application launched from the same Docker image runs consistently as expected, whether it is on the devices of developers or testers, or in the production environment.
- **Quick deployment**: By uploading the Docker image to Docker Hub and pulling the Docker image on other machines with Docker installed, our team can quickly launch the application on different devices.
- **Simplify**: Docker technology allows us to bypass traditional cumbersome steps to launch our application. Instead of manually installing dependencies and running the program through specific commands, which can sometimes lead to errors due to compatibility issues with dependency packages, we have described and defined the dependencies, libraries, and configuration files for our frontend, backend, and database in the Dockerfile of our project.

It's important to note that the Docker container's execution environment must match the system architecture used during the image construction. Since MRC utilizes the amd64 system, it is important to

Application	Image Name	Build On
Frontend	redpeony159/myreactapp:latest	amd64 & x86
Backend	redpeony159/myflaskapp:latest	amd64 & x86
CouchDB	redpeony159/couchdb:1.0	amd64 & x86
Harvester	redpeony159/harvester:latest	amd64 & x86

Table 2: Docker Image Table

ensure that any image built for deployment is compatible with this architecture. If building the image on an M1 or M2 MacBook with ARM architecture, the resulting image may not run properly on the MRC.

For frontend and backend applications, each application have a unique Dockerfile at their root directory in our codebase. These files outline the necessary environment for Docker image creation and provide instructions for launching the application. The Docker images are built and uploaded to Docker Hub once the development is complete.

Docker Swarm is an orchestration tool built into the Docker platform. We deploy the appliaction within a docker swarm network. It automatically create services to nodes in the cluster, ensuring the services are always up and running. Therefore we could our application has high availability and fault tolerance.

For database, the Docker image is derived from the ibmdb/couchdb:3.2.1 version image. We achieved this by adding configuration files and using the docker commit command.

3.7.2 Automation

By integrating Ansible's powerful automation capabilities with Docker Swarm's robust orchestration features, we can achieve seamless automation of deploying, scaling, and managing containerized applications with enhanced efficiency and consistent reproducibility. Using Ansible simplified our deployment process. In our project, the ansible scripts will cover and achieve the following tasks automatically.

- Create instances, setup security group, volumes attachment from config file.
- Create Docker Swarm cluster and deploy the frontend, backend, mastodon harverster applications.
- Create and configure CouchDB, including setting up the cluster.
- Scale up/down the services by adjusting the replicas in docker swarm network.

3.8 Scalability

To address the scalability issue, we have implemented several approaches. Firstly, to resize the instance configuration, such as adjusting the number of CPUs, users can effortlessly perform this operation through the MRC dashboard. This provides a convenient way to adapt the system resources to meet the growing demands of the application.

In terms of database scaling, users have two options. They can either create a new instance and integrate it into the existing CouchDB cluster using command line tools. Alternatively, if users prefer to set up the cluster from scratch, they need to update the parameter settings in the Ansible roles section. This flexibility allows users to expand the database capacity horizontally by adding more instances, ensuring efficient storage and retrieval of data.

For scaling the backend and frontend components, our applications are containerized using Docker, which simplifies the scaling process. By leveraging the DockerFile, users can easily scale the backend

and frontend components based on the specific requirements of their application. This ensures that the application can handle increased traffic and user load, maintaining optimal performance.

In the case of the Mastodon data harvester, our current instance configuration supports both horizontal and vertical scaling. Horizontal scaling is achieved by adding more harvester API classes, allowing for parallel processing of data and enhancing overall throughput. Additionally, if the need arises, vertical scaling can also be implemented to increase the resources allocated to each harvester, further optimizing the data harvesting process.

By employing these various approaches to scalability, we provide users with the flexibility to adapt their system resources, expand their database cluster, scale the application components, and enhance the data harvesting capabilities. This comprehensive and adaptable framework ensures that our system can efficiently handle increasing workloads and evolving requirements.

4 Data Processing

This section outlined our strategies for processing and managing diverse datasets from Twitter, Mastodon, and SUDO. We implemented parallel processing techniques, data harvesting strategies, and ensured the integration of sentiment analysis and language detection for Twitter and Mastodon datasets. No need to processing SUDO data due to it provided well-structured dataset. Notably, our approach addressed the issue of scalability. We've employed a vertical scaling strategy for Twitter data and a horizontally scalable architecture for Mastodon data processing. Our methods ensure both effective data processing and the scalability necessary for the system's future growth and expansion.

4.1 Huge Twitter Data Processing

Our team was tasked with the processing of a huge dataset, consisting of 57GB of Twitter data. Each entry in the provided JSON file represented a single tweet, and our objective was to extract key information such as the unique tweet ID, the author, the time of publication, the geographical location, and the content. Additionally, we aimed to compute the sentiment score and detect the language of each tweet using Natural Language Processing (NLP) techniques.

Processing this considerable amount of data presented significant challenges. We approached this task using a divide and conquer strategy. Leveraging the Message Passing Interface (MPI) enabled us to parallelly process the dataset. We utilised regular expressions to extract pertinent information from each tweet, which was then stored as a 'Tweet' object in temporary storage. Whenever this temporary space reached a capacity of 1,000 records, a customised CouchDB API was invoked to upload the bulk data.

Once we extracted the content, we normalised the text using the `nltk.stem.WordNetLemmatizer` library. We further enriched our dataset by calculating the sentiment score of each tweet with `nltk.sentiment.SentimentIntensityAnalyser`. This score, which ranged from 0 to 1, helped us gauge the emotional tone of the tweet, from extremely negative to extremely positive. We also leveraged the `langdetect` library, powered by TensorFlow, to identify the language of the tweet.

For tweets containing geographical data, we first normalised the location string by removing punctuation and converting it to lowercase. Subsequently, we attempted to match this location with the 2021 Statistical Area Level (SAL) data from a previous assignment. If the location didn't directly match the SAL data, we converted the location string into a list of ngram words for a second attempt at matching. Once a match was found, the SAL code was stored for future use, particularly for plotting.

A consideration during this process was the CRUD lock created by CouchDB during data streaming and processing. If a thread was engaged in data uploading, other threads had to wait for this task to complete. Consequently, the data upload rate significantly impacted the processing speed of other threads. We strived to maintain a reasonable bulk size of between 801 and 1,500 records to mitigate long waiting times.

Our methodology proved effective. We successfully processed approximately 15.9 GB of data, encompassing 37,823,414 tweets, 2,418,621 of which contained geographical data. To facilitate future tasks, we split the data into two databases—one for all tweets and another exclusively for geographically tagged tweets. For most studies, we predominantly utilise tweets that contain geographical information.

Currently, the data processing is conducted locally. However, we have also developed a RESTful API that allows users to parse any Twitter file. In terms of scalability, we've implemented a vertical scaling strategy by augmenting the number of processing units for the MPI task. Given our present system design, the processing of this sizeable Twitter dataset using three nodes is time-consuming. As per Amdahl's law, increasing the number of processing units should enhance the efficiency of threaded tasks. Thus, a vertical scale-up of the system is expected to result in more efficient processing.

4.2 Mastodon Toot Data Processing

Unlike Twitter, bulk Mastodon toot data isn't readily available for download. Consequently, we resorted to using the Mastodon API to manually scrape data from targeted servers. Given our project's focus on aspects of related to life, we selected to harvest data from `Mastodon.Social`, `Mastodon.AU`, and `Mastodon.Tictoc` servers.

We faced certain limitations with the Mastodon API. Each harvester could only obtain 300 toots every five minutes with a single access token and IP address. Moreover, the API doesn't support retrieving toots from a specific date unless a starting toot id from the desired date is provided. As our team decided to start the harvest from 12-31-2021, we found the initial toot id using a custom binary search function. To manage harvesting from three different Mastodon servers, we implemented a backend scheduler running at 10-minute intervals. Each cycle was assigned to a single thread which managed a list of three Mastodon server APIs. Each iteration fetched 40 toots from each API and streamed them to separate databases according to the Mastodon server name.

Nevertheless, manual provision of the entry toot id for each API request wasn't feasible. As a workaround, before each harvest cycle, the system would check the id of the latest toot and fetch the next 40 available records succeeding that id. The harvested toots were well-structured and stored in an object. Like our Twitter data processing approach, we extracted key information such as the toot id, author id, creation date, and content from this object. Using NLP techniques, the content was then normalised and a sentiment score calculated. Upon processing, the data was streamed to the respective database.

To address scalability while considering the efficiency and resource usage of the harvester, we have adopted a single-thread model for the harvesting job. Generally, the scheduler is set to execute the harvester task across the three servers every 10 minutes. As the task progresses, our system has been designed to handle scale efficiently.

We have opted for a horizontal scaling strategy by incorporating a ReSTful API into the backend. This feature enables users to expand the scope of the harvester by adding more valid Mastodon servers to the scheduler. In simple terms, a user can send a GET request to activate a valid harvester or deactivate an existing one. This method allows us to manage and utilise resources effectively, ensuring the scalability of our system while maintaining performance efficiency. This architecture not only supports our current data processing needs but also provides ample flexibility for future expansion and scaling.

Till 18-05-2023, we have harvested 757.9 MB of data, comprising 1,659,690 toots from the period of 12-31-2021 to 22-10-2022 from three Mastodon servers. This toots data serves as a valuable resource for our project, offering further insights into social sentiments and trends for our scenario analysis.

4.3 Spatial Urban Data Observatory (SUDO)

The SUDO datasets are well-structured and well-formatted, hence no need for further processing. In accordance with our focus scenarios, we utilized two primary datasets:

- ABS - Jobs In Australia - Employee Jobs and Income by Industry (SA2) 2018-19
- VIC CSA - Crime Statistics - Criminal Incidents by Principal Offence (LGA) 2010-2019

To generate meaningful MapBox plots, we incorporated the Statistical Area Level 2 and Local Government Area shapefiles provided by the Australian Bureau of Statistics (ABS) for future analyses. Consequently, we generated map plots and summary reports, which were stored on the frontend for quick user access. Given our current focus on only two SUDO datasets for visualization, we store the data on the frontend. However, in future, all SUDO data is planned to be stored as a blob in CouchDB.

There is no Scalability design for SUDO data. Given the nature of SUDO data, distinct processing methods are required based on the selected scenario topics. In an ideal scenario, users would be able to upload the processed SUDO data via the backend RESTful API, and see visualizations on the frontend.

5 User Guide

This section provides a concise user guide outlining the steps to effectively run and utilize our application. Before running any of the following commands, please make sure you have connected with the University Melbourne VPN and clone required code from GitHub[1]. The detailed instructions are list below.

5.1 Flask Backend

Flask application is a REST API that provides several endpoints to access processed json data for front-end visualization: crime sentiment, income sentiment, and Twitter statistics. This guide provides a high-level overview of each endpoint. The flask app is running as a service on our docker swarm network, it could be access on every single node: `http://172.26.130.83:8080, http://172.26.128.118:8080`

1. GET /ping This endpoint is used to test the connection to the backend server. Upon accessing, it returns a string indicating the IP address, container ID, and the timestamp of access.
2. GET /api/sudo/crime/ This endpoint provides access to crime sentiment data, which is returned as a zipped JSON file. If the file does not exist, the endpoint first generates the data and compresses it before sending.
3. GET /api/sudo/income/ This endpoint provides access to income sentiment data, which is returned as a zipped json file. Similar to the crime sentiment endpoint, if the file does not exist, the endpoint first generates the data and compresses it before sending.
4. GET /api/twitter/ This endpoint generates or fetches previously generated Twitter statistics data and returns it as a compressed JSON file.

Error Handling In case of an error during data generation or retrieval, the server returns a json message with a status code to indicate the type of error.

[1]This repository will be public available after 26th May 2023

5.2 React Frontend

For universal access, users can connect to the front-end web application from any web browser by navigating to http://172.26.130.83:3000/. Upon successful rendering, users will be greeted with an interface as shown in Figure 7. The web page comprises three main components: a navigation bar at the top, a summary statistics panel on the left, and two interactive maps on the right.

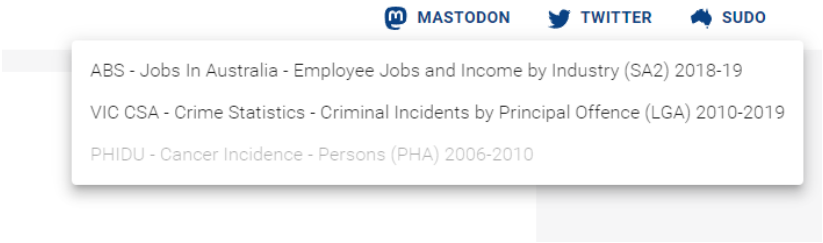

Figure 8: Navigation Section

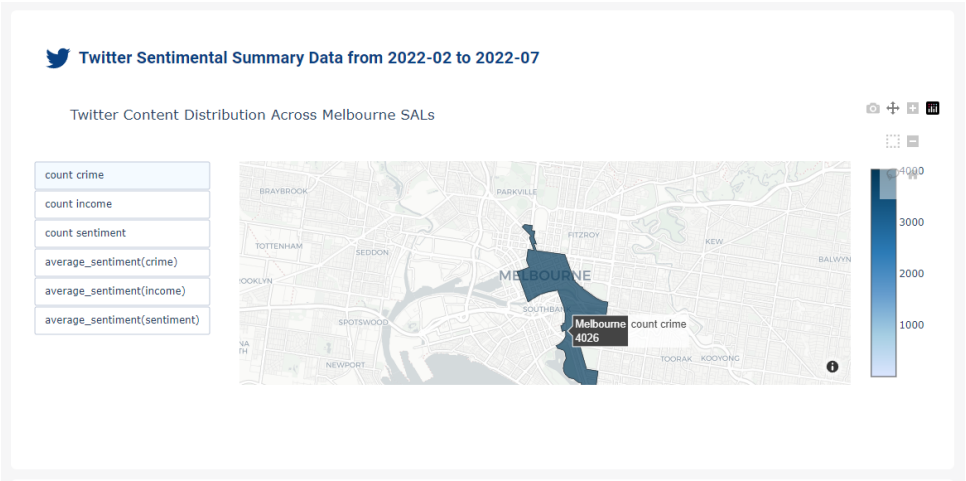

Figure 9: Interactive Map

To navigate through the application:

1. Users first select their target data source, as illustrated in Figure 8. Data from different Mastodon servers, Sudo data, and Twitter summary statistics are all available.
2. Both panels on the left and right will update the plots based on user selections.
3. Users can then analyze the data by interacting with the plots and maps.

For instance, to analyze crime-related scenarios, users would navigate to the Sudo section, switch the data source from 'Job Incomes' to 'Crime Statistics', and then interact with the map by selecting from different groups, zooming, or hovering over the map to view summary information.

```
# Navigating to working directory
# install required pacakge and run development server
npm -i && npm start
```

For local development, Node.js version 12 or above is required. For deploying the statistical website, Node.js version 14 is a mandatory requirement. To run the development server, use the commands above and start on your development journey.

5.3 CouchDB Database

The end users do not need to interact with the database directly. Therefore, this part of the guide aims to show the steps that how to deploy the CouchDB cluster and how to use it for the developers.

5.3.1 CouchDB Cluster Deploy - Manually

1. Instance Setup: The instances were set up in the Melbourne Research Cloud (MRC), selected according to the requirements of the project. For further details on this process, visit the MRC's official page.
2. Volume Setup: Given the limited storage of 30GB in the instances, the data needed to be stored in mounting volumes. Volumes were applied for and set up in MRC, then subsequently attached to each instance. These volumes were then mounted to the path "/mnt/" and formatted accordingly. Unique identifiers (UUIDs) for the volumes were recorded for reference.
3. Permission Setting: Appropriate permissions were granted to the Docker image, enabling it to access the path "/mnt/couchdb".
4. Docker Installation: Docker was installed on the instances following the official Docker manual.
5. Docker-Compose Installation: Docker-Compose was installed using the command:
```
sudo apt install docker-compose
```
6. Configuration File Setup: A Docker-Compose file ("docker-compose.yml") was created, wherein the IP addresses were set to the instances' allocated IPs.
7. Docker Container Setup: Docker containers were run with the command:
```
sudo docker-compose up -d
```
8. Cluster Configuration: The CouchDB cluster was configured with specific nodes, usernames, passwords, and a defined cookie.
9. Photon Web-Admin Addition: The Photon web-admin tool was added to the CouchDB cluster for easier management and administration of the database.
10. Database Creation: A test database was created on one node of the cluster, confirming that it is replicated across all other nodes.

5.3.2 CouchDB cluster access

1. Accessing the database via CouchDB Fauxton (figure 4) or Photon (figure 5) Dashboard.
2. Accessing the database via code, for example, Python library couchdb or use team defined python CouchDB API class.

5.4 Ansible Automation

Prerequisite

- Ensure that your local machine is connected to the University of Melbourne's internal network.
- Have the `openrc.sh` configuration file downloaded from the MRC dashboard.
- Have the password for the MRC dashboard.

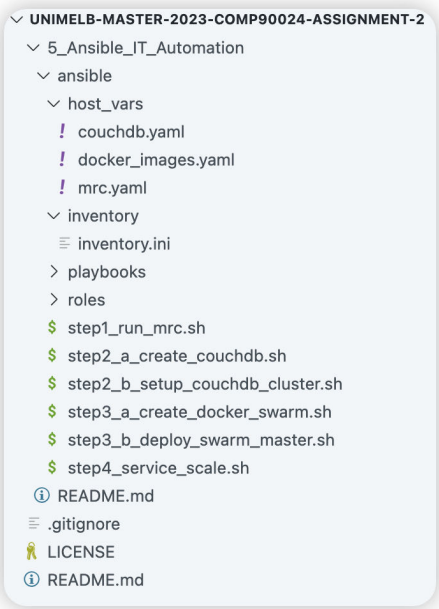

Figure 10: Ansible-Directory-Tree

Instance Creation

We start by creating instances based on the configuration information set in `hosts_vars/mrc.yaml`. These information includes the system image, the instance name, creation of data volumes, and attaching volumes to each instances.
Then the script `step1_run_mrc.sh` would Complete the tasks defined in `mrc.yaml` on MRC.

Setting up inventory.ini

After creating the instances, we proceed to copy the IP addresses of the created instances from the MRC dashboard and fill them in the `inventory/inventory.ini` file. We then assign tasks to the instances according to our needs.

Database Deployment and Cluster Setup

- First, we set the IP address information of the database node in the `inventory/inventory.ini` and fill in the node information in `ansible/roles/couchdb/templates/init_db.sh`. This script is responsible for establishing the cluster.
- Then, we run `step2_a_create_couchdb.sh` to deploy CouchDB containers on all database nodes using Docker, completing the mapping of data volume mounts.
- Finally, we run `step2_b_setup_couchdb_cluster.sh` to set up the cluster.

Frontend, Backend and Harvester Deployment

- In our project, for example, we set 172.26.130.83 as the Swarm master node and 172.26.128.118 as the swarm worker. By running the `step3_a_create_docker_swarm.sh` script, we create a Swarm network based on the node information in `inventory.ini`.
- After this, we run `step3_b_deploy_swarm_master.sh` to deploy the frontend React app, the backend Flask app, and the Mastodon Data Harvester into the Swarm network. The docker image that we created and deployed are stored in the `host_vars/docker_images.yaml`.

Scaling Service

The script `step4_service_scale.sh` is set to control the number of replicas of the services in the docker swarm, which can flexibly adjust the scale of the service.

Figure 11: Run Scale Script

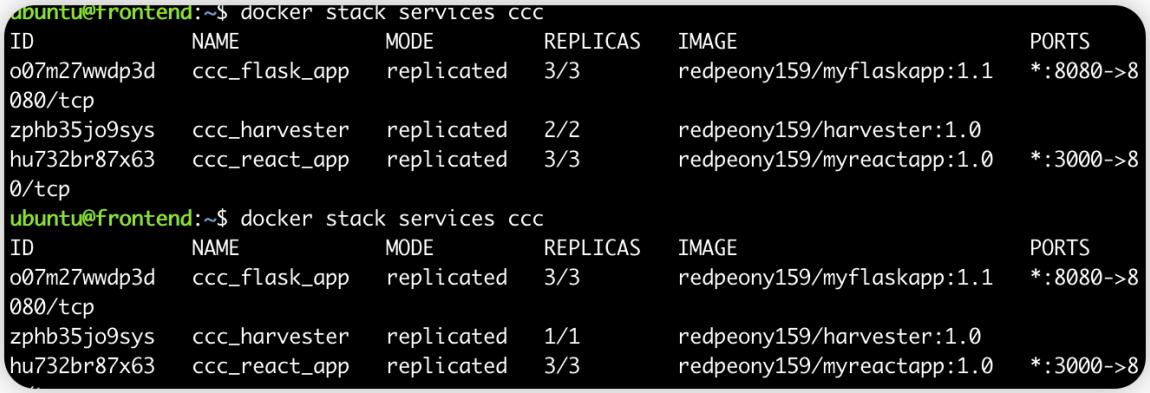

Figure 12: Replica Changes from 2 to 1

5.5 Data Processing

To streamline data processing needs, we have encapsulated most of our code within Jupyter notebooks. This includes functionalities for generating plots, experimenting with the Mastodon toot fetch API, and more.

For handling large Twitter corpora, we modified Assignment Part 1 and adapted the messaging for parallel computing. To execute MPI programming, please follow the instructions below.

```
# make sure virtual environment and mpi4py installation
mpiexec -np [NUMBER OF PROCESSOR] python3 [Twitter Processing Scripts] -t [Huge
    Twitter File]
```

In addition, we provide several useful utility tools for other sections, such as the CouchDB API.

6 Supported Scenarios and Analysis

The project focuses on sentiment analysis based on the Twitter dataset. We examine how the distribution of Twitter sentiment scores varies across various geographical locations and under different social scenarios.

Specifically, we focus on the twitters sent from different geo-locations within Victoria, especially geo-locations around Melbourne. And we investigate the social scenario effects based on the following two typical scenarios.

- **Income**: Examine the distribution of tweets discussing income inequality, financial struggles, or job satisfaction across different income brackets. Identify locations with higher median incomes and compare the sentiment of tweets in these areas to those with lower median incomes.
- **Crime**: Analyze the relationship between criminal incidents by principal offence and the frequency of tweets discussing crime or safety in specific locations. Investigate how the sentiment of these tweets varies across areas with different crime rates.

To do sentiment analysis more comprehensively, we also consult other social media data from Mastodon servers, and official data from SUDO. Especially for the social scenario analysis, we hope that official social science statistics from SODU help us better understand the reason behind various sentiment score distributions.

6.1 Overall Sentiment Score Distribution

Overall, the sentiment score ranges from 1 (extremely negative sentiment) to 9 (extremely positive sentiment). The sentiment score distribution across the entire Twitter dataset is shown in Figure 18. The scores are distributed densely at the centre of score 5, and the distribution is slightly right-skewed in general (i.e., there tends to be more posts with positive sentiments than with negative sentiments).

For Mastodon data, interestingly, while the two servers Mastodon AU and Mastodon TicToc have sentiment distributions similar to that of the Twitter data, the Mastodon Social server actually has sentiment scores distributed extremely dense at the mean score of 5, with extremely small variance (as shown in Figure 14.

As for the geo-location effect, no obvious pattern can be observed at this overall data aggregation stage. The majority of regions exhibit an average sentiment score ranging from 4 to 7. For other regions with low average sentiment scores (less than or equal to 4) or high average sentiment scores (greater than 7), a visual representation is constructed (see Figure 15). However, it is noteworthy that these regions are dispersed randomly, lacking discernible clusters. We will keep investigating the geographical distribution of the sentiment scores by grouping the data under different social scenarios and see whether certain patterns can be further discovered.

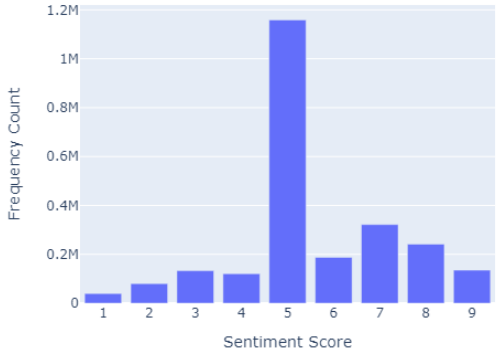

Figure 13: Twitter Sentimental Summary Data from 2022 02-07 Overall Sentiment Score

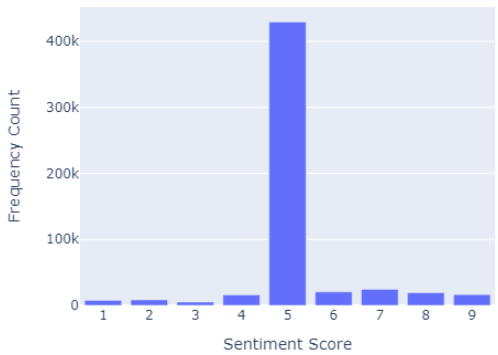

Figure 14: Mastodon Social Overall Sentiment Score

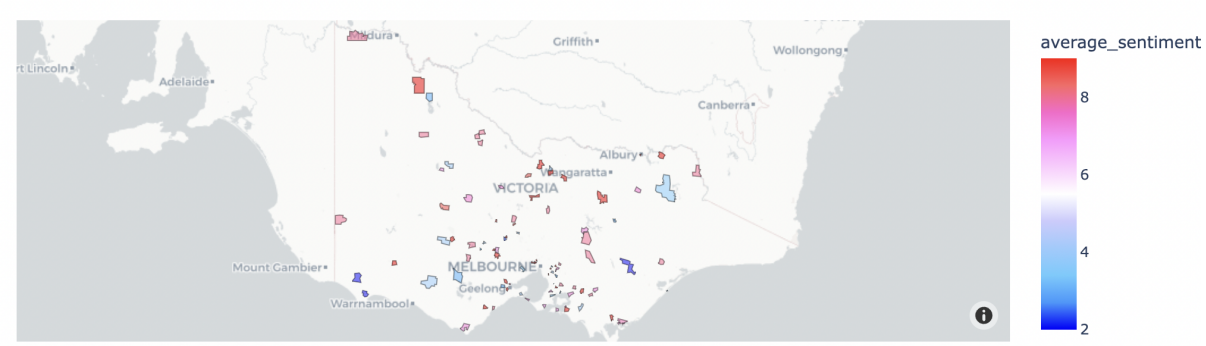

Figure 15: Low and High Average Sentiment Areas

6.2 Scenario 1: Income

6.2.1 Insights Derived from the Distribution of SUDO Data

After removing the outliers, the median income across 420 SA2 regions within Victoria ranges from a minimum of 28.996k AUD in Merbein to a maximum of 62.029k AUD in Sydenham. Overall, the regions with relatively high median incomes are primarily concentrated near Melbourne (as indicated by Figure 16), with most areas near Melbourne having a median income of around 50k, which surpasses 50th quantile of 45.8k AUD for Victoria as a whole. Furthermore, the map also demonstrates that median income is generally slightly higher in coastal areas than inland areas.

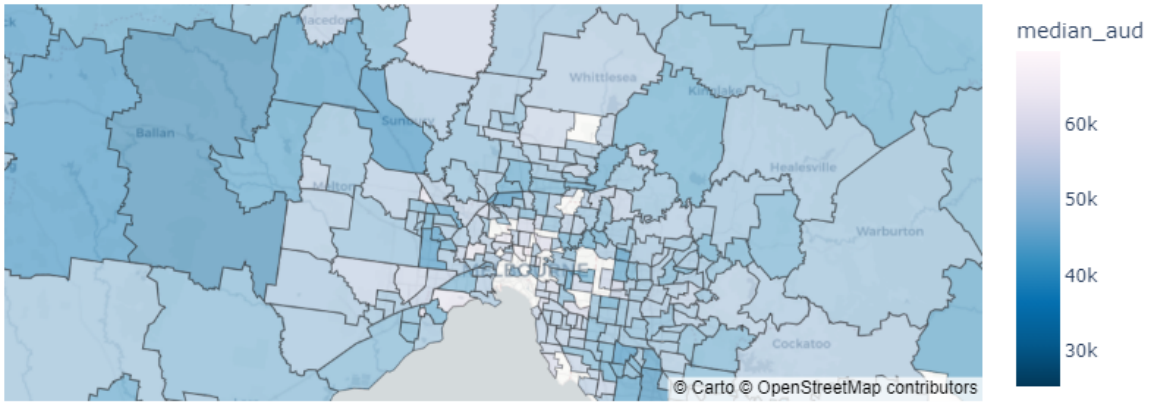

Figure 16: Income Levels Distribution around Melbourne

6.2.2 Insights Derived from the Distribution of Twitter Corpse

Across Victoria, the number of tweets mentioning income related terms in each SA2 region is counted and found that it was highly unevenly distributed. 73% of the regions have only 1-10 tweets mentioned income related terms, 22% of the regions have 10-100 such tweets, only 4.7% of the regions has more than 100 tweets contained income related terms. Notably, Melbourne stands out with a significant concentration of income-related tweets, totaling 23.281k tweets, which greatly exceeds the count in other regions. While the SUDO data also shows a trend for high median income regions to be clustered around Melbourne, they are not overwhelmingly more numerous than other areas like Twitter data is.

However, though the income levels and the frequencies of income-related topics vary across different geo-locations, there seems to be no related geographical patterns for the distribution of sentiment scores

(e.g., see Figure 17 for the geographical distribution of income-topic-related sentiment scores around Melbourne).

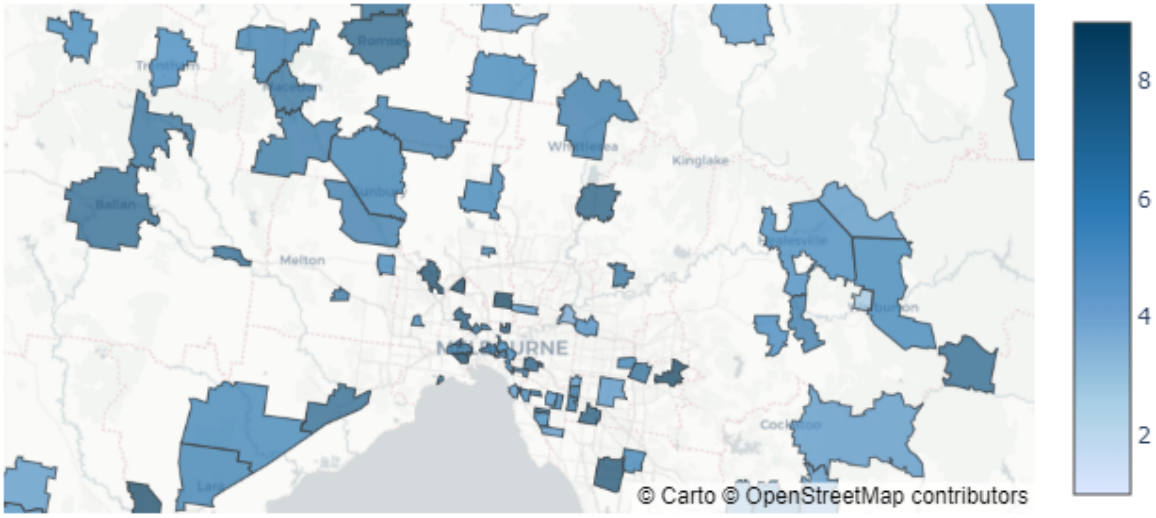

Figure 17: Income-topic-related Twitter Sentimental Scores Geographical Distribution around Melbourne

6.2.3 Numerical Distribution of Sentiment Scores in Mastodon

For Mastodon data, we discuss the numerical distribution of the sentiment scores. Interestingly, the income-related sentiment scores from the Mastodon servers tend to follow double or triple modal distributions (unlike that from the Twitter data which follows a single modal distribution around the score of 5). Typically, for the Mastodon Social server, we see that most income relating posts are either too high (≥ 7) or too low (≤ 3) in their sentiment scores, as shown in Figure 18. This may reflect that people tend to post to either show off their luxurious life (with sentiment ≥ 7), or to complain about their financial difficulties (with sentiment ≤ 3).

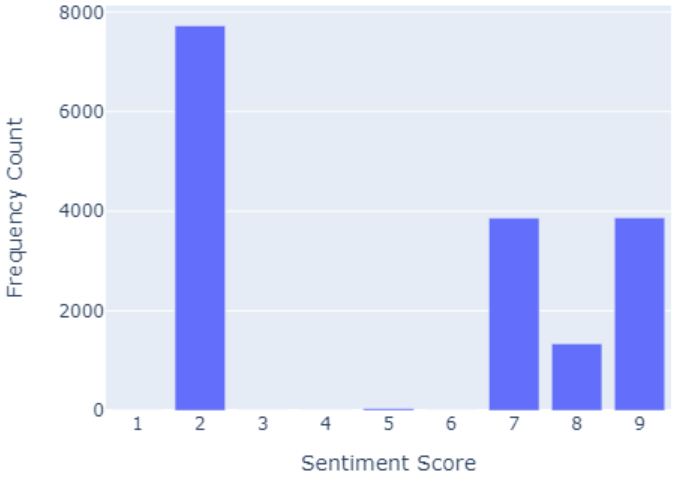

Figure 18: Mastodon Social Income Sentiment Score

6.2.4 Scenario 1 Summary

The analysis of both SUDO and Twitter data provides interesting insights into the distribution of median incomes and the mention of income-related terms across the SA2 regions in Victoria. The SUDO data suggest that regions with higher median incomes tend to cluster around Melbourne, with a slight tendency for coastal areas to have slightly higher median incomes compared to inland areas. However, this pattern is not as pronounced in the Twitter data. On the other hand, the Twitter data reveals an uneven distribution of tweets mentioning income-related terms, with a significant concentration of such tweets in Melbourne compared to other areas. This indicates that income-related discussions on Twitter are more prevalent in Melbourne than in other regions.

However, there is no clear indication that people in higher or lower median-income areas are tweeting more or less frequently about income-related topics. Further, the income level and the income discussion frequency seem to have little impact on the geographical distribution of sentiment scores.

Interestingly, most income-related posts on Mastodon tend to either show off people's luxurious lives (with sentiment ≥ 7), or to complain about people's financial difficulties (with sentiment ≤ 3).

6.3 Scenario 2: Crime

6.3.1 Insights Derived from the Distribution of SUDO Data

Among the 72 LGA regions in Victoria, Brimbank has the highest number of cases across all types of offenses. Overall, several suburbs near the CBD, such as Wyndham, Melton, and Whittlesea, have a relatively high number of cases, which is likely due to their large population size. Additionally, other suburbs with a significant number of offenses include Mildura, Ballarat, Greater Bendigo, Greater Shepparton, Latrobe, Frankston, and Mornington Peninsula (i.e., see dark blue areas in Figure 19). These areas are more likely to be considered dangerous districts.

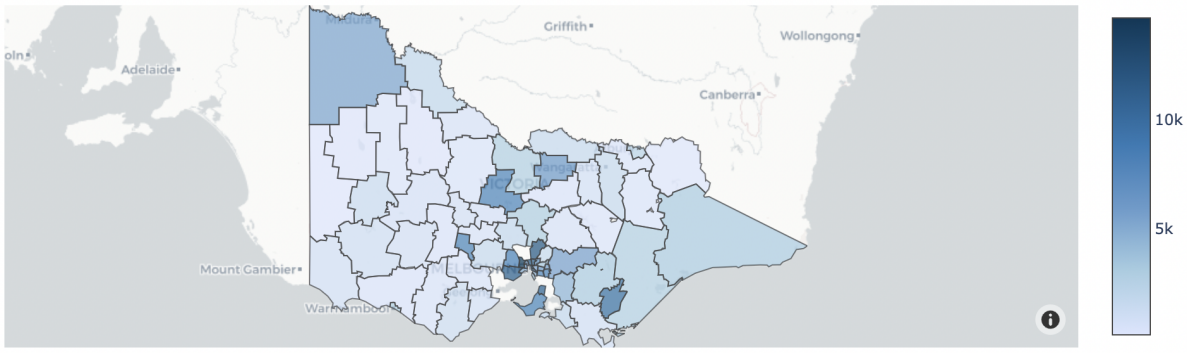

Figure 19: SUDO Total Offences Distribution across Victoria

6.3.2 Insights Derived from the Distribution of Twitter Corpse

Similar to the analysis of income, an examination of the number of tweets mentioning crime-related terms in each LGA region across Victoria demonstrates a highly uneven distribution. Approximately 79% of the regions have only 1-10 tweets mentioning crime-related terms, while 18% have 10-100 such tweets. Merely 3% of the regions contain over 100 tweets with crime-related terms. Once again, Melbourne stands out with a significantly higher count compared to other regions, totaling 4,026 tweets (see Figure 20). It is worth noting that, Ballarat follows as the region with the second highest number of tweets about crime, with 485 tweets. This observation aligns with the speculation based on the SUDO data that

Ballarat could be a genuinely dangerous region. More interestingly, we find that areas with high crime-discussion frequencies actually tend to have more negative sentiment scores related. This is reasonable in the sense that crime-related topics and events tend to cause more disruptions to people's lives in general.

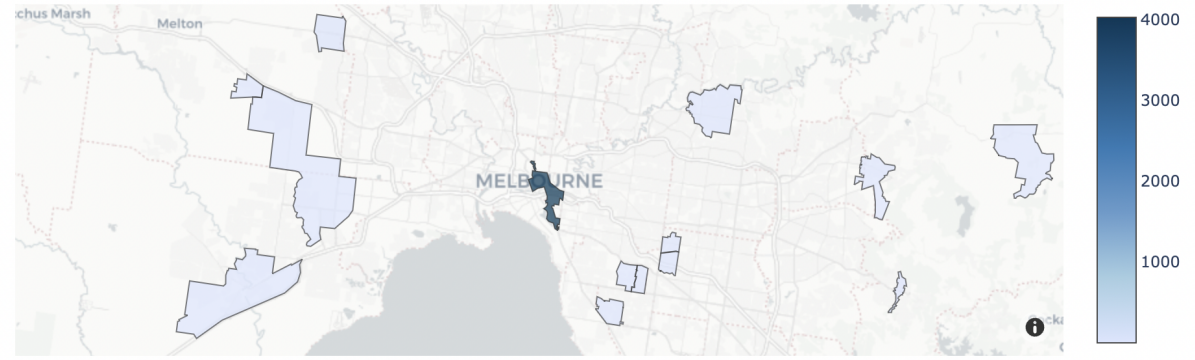

Figure 20: Crime-topic-related Tweets Frequency Distribution around Melbourne

Looking at the numerical distribution of the crime-topic-related sentiment scores, the distribution is dramatically left-skewed, with the mean centred around the score of 3, and with a mode at the most negative sentiment score of 1, as shown in Figure 21. This pattern is actually consistent with people's common sense, and it reflects individuals' tendency to express dissatisfaction about crime-related topics or events on social media platforms such as Twitter.

Figure 21: Twitter Sentimental Summary Data from 2022-02 to 2022-07 Crime Mentioned Sentiment Score

6.3.3 Scenario 2 Summary

Analysis of SUDO data and Twitter data provides valuable insight into crime in Victoria. Brimbank has the highest number of all types of crime, indicating a clear presence of crime. suburbs near the CBD show a relatively high number of cases, but this may be due to their larger population size. In addition, other areas such as Mildura, Ballarat, Greater Bendigo, Greater Shepparton, Latrobe, Frankston and Mornington Peninsula show high levels of crime, indicating a greater likelihood of being considered dangerous areas. On the other hand, Twitter data shows that the number of crime-related tweets varies greatly by LGA area. Melbourne stood out with the highest count of 4,026 crime-related tweets, followed

by Ballarat with 485 tweets. This is consistent with SUDO's data. In addition, we find that the discussion about crime topics or events in nature inherits negative sentiments, and more frequent crime-topic discussions tend to lead to more negative sentiments in general.

Note that we are not considering Mastodon data for this scenario due to the lack of crime-related discussions in the Mastodon servers we sampled from.

7 Conclusion

In conclusion, our team has successfully conceptualized and created a social sense dashboard web application. This application uses social media resources such as Twitter corpus, Mastodon data, and data from the SUDO platform to gain insights into crime rates and income distributions in Australia. We leverage the cloud-based solution provided by the Melbourne Research Cloud (MRC) to harvest, store, and analyse large scale data. This addresses the limitations of SUDO, such as the inability to explore real-time data from social media and to compare data from various sources in a consolidated platform.

The result of our effort is a comprehensive cloud computing application, utilizing the IaaS provided by MRC. Our cloud solution encompasses a complete ecosystem, from a backend server to frontend visualisation, supported by a NoSQL CouchDB database and a data processing and analysis module. All these components are neatly packaged using Docker and exhibit automation and scalability through Ansible. This system facilitates a better understanding of the relationships between income levels and crime rates, and their related discussions on Twitter.

Looking ahead, we acknowledge the need for further enhancements to our system. Our focus would be on developing more precise system interaction logics. For instance, the backend could be optimized to handle customized data queries. We also image a future where users can input their data into our application for a one-stop analysis purpose. Furthermore, incorporating advanced machine learning algorithms could greatly enhance the analytical capabilities of our system, providing even more nuanced and valuable insights, and our application become good data analysis tool.

8 Appendix

8.1 External Links

Description	URL
Resources	
GitHub	https://github.com/rNLKJA/Unimelb-Master-2023-COMP90024-Assignment-2
Ansible Demonstration Video	https://youtu.be/qermcMn7x1M
Overleaf Report	https://www.overleaf.com/read/fvkvxvsrtsxs
Presentation Slides	https://www.canva.com/design/DAFjo4avJwE/wclPChIlSh_BJLDBhiELpw/edit?utm_content=DAFjo4avJwE&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
Melbourne Research Cloud (MRC)	
MRC Flask Backend	http://172.26.128.118:8080
MRC React.JS Frontend	http://172.26.130.83:3000
MRC CouchDB Fauxton	http://172.26.134.248:5984/_utils#
MRC CouchDB Photon	http://172.26.134.248:5984/photon/_design/photon/index.html#
Spatial Urban Data Observatory (SUDO)	https://sudo.eresearch.unimelb.edu.au/
Mastodon	
Mastodon Social	https://mastodon.social
Mastodon AU	https://mastodon.au
Mastodon Tic Toc	https://tictoc.social

8.2 Team Information

Name	Major	Roles	
Xuan Wang	Software Engineering	Database	Ansible
Wei Zhao	Software Engineering	Backend	Ansible
Zongchao Xie	Data Science	Scenario Analysis	Data Processing
Runqiu Fei	Data Science	Data Processing	Database
Sunchuangyu Huang	Data Science	Frontend	Data Processing