COMP90024 PROJECT2 REPORT

A Cloud-based Solution to Social Media Sentiment Analysis

Group 67

Yun-Chi Hsiao, SID:1074004[1], Jiahe Liu, SID:1214235[2], Jongho Park, SID: 1152502[2], Benjamin Murdoch, SID: 1434075[2], and YongLi Qin, SID: 1174036[1]

[1]School of Computing and Information Systems, The University of Melbourne
[2]School of Mathematics and Statistics, The University of Melbourne

Abstract

Our study performed a could-based solution to social media sentiment analysis. We leveraged the Melbourne Research Cloud for the establishment of instances required for data gathering, data processing, data administration, and website deployment. We developed Ansible scripts to support automatic deployment and dynamic scaling of our project. For our analysis scenarios, we performed Natural Language Processing (NLP) study investing the happiness expressed on social media in Australia and globally. Scenarios include happiness variations by time, location, and and influential factors for happiness expressed on social media. Machine learning model was developed to classify potential reasons for happiness.

Contents

1 User Guide 4

1.1 Manual Deployment 4

1.1.1 Deploying Instances 4

1.1.2 Configuring Instances 4

1.2 Dynamic Deployment Using Ansible 4

1.2.1 Deploying CouchDB 4

1.2.2 Deploying Data Crawlers 4

1.2.3 Deploy Back-end Server 5

1.2.4 Start Front-end Application 5

2 System Architecture and Design 5

2.1 Rationale for System Design 5

2.2 Allocated Resources 6

2.3 Instances 6

2.3.1 Instance 1 (mastodon-harvester) 6

2.3.2 Instance 2 (twitter-harvester) 6

2.3.3 Instance 3 (couchdb-master) 7

2.3.4 Instance 4 (web-server) 7

2.4 Application (functionalities) 7

2.4.1 Mastodon Harvester 7

2.4.2 Twitter Processor 8

2.4.3 Front-end 8

2.4.4 Back-end 8

2.5 Ansible 9

2.5.1 Hosts 9

2.5.2 Initial Setup 10

2.5.3 Volume Setup 10

2.5.4 Applications Setup 10

2.5.5 Starting Monitoring 10

2.6 Containerisation 10

2.6.1 Docker 11

2.7 Mastodon Dynamic Scaling 11

2.7.1 Install Backup Mastodon Harvester 11

2.7.2 Remove Backup Mastodon Harvester 11

2.8 Server Dynamic Scaling 11

2.8.1 Server Side 11

2.8.2 Client Side 11

2.8.3 Crontab 12

2.9 CouchDB Database 12

3 Data Management 13

3.1 Data Collection 13

3.1.1 Twitter Data 13

3.1.2 Mastodon Data 13

3.1.3 Additional Dataset 13

3.1.4 Spatial Urban Data Observatory (SUDO) Data 13

3.2 Data Filtering 13

3.3 Data Preprocessing 14

3.4 Feature Engineering 14

3.5 Data Management with CouchDB 14

3.5.1 MapReduce Functions 14

3.6 Data Management with Web APP 14

3.6.1 Data Delivered from Database to Back-end 15

3.6.2 Data Delivered from Back-end to Front-end 15

4 Data Analysis **16**
4.1 Related Work . . . 16
4.2 Algorithm Design . . . 16
4.2.1 Quantifying Happiness by NLP . . . 16
4.2.2 Happiness Classification by Machine Learning . . . 17

5 Scenarios **17**
5.1 Exploring Happiness Variations by Time . . . 17
5.1.1 Happiness Trend from February 2022 to August 2022 . . . 17
5.1.2 Happiness Trend by Hour of a Day . . . 18
5.1.3 Happiness Trend by Day of Week . . . 18
5.2 Exploring Happiness Variations by Location . . . 18
5.2.1 Happiness Score Across Australian States . . . 19
5.2.2 Happiness Score Across Greater Capital Cities and Suburb and Localities . . . 19
5.3 Influential Factors of Happiness . . . 20
5.3.1 Social Factors . . . 20
5.3.2 Economic Factors . . . 20
5.3.3 Classify Reasons For Happiness . . . 21

6 Challenges **22**
6.1 Error Handling . . . 22
6.1.1 Error Handling with Preprocessing . . . 22
6.1.2 Error Handling within the Virtual Machines . . . 22
6.1.3 Error Handling with CouchDB . . . 22
6.1.4 Error Handling with Web Server . . . 22
6.1.5 Error Handling with Data Pipeline . . . 22
6.2 Limitations . . . 23
6.2.1 Ansible and MRC limitations . . . 23
6.2.2 Data Limitations . . . 23

7 Discussion (Pros and Cons) **24**
7.1 MRC . . . 24
7.1.1 OpenStack and IaaS . . . 24
7.1.2 Private cloud setup . . . 25
7.2 System Scalability . . . 25
7.2.1 Mastodon Harvester . . . 25
7.2.2 Server . . . 25
7.3 Tools and Processes for Image Creation and Deployment . . . 25
7.3.1 Virtual Machines . . . 26
7.3.2 Docker . . . 26

8 Innovation **26**

9 Video Link **27**

10 GitHub Link **27**

11 References **27**

1 User Guide

In this section, we detail how our project was deployed to the Melbourne Research Cloud (MRC) in a manner that allows for scalablility.

1.1 Manual Deployment

As is detailed in *section 5* of the report, were we unable to create a script to deploy the instances due to issues with OpenStack, and as a result these sections were done manually.

1.1.1 Deploying Instances

We created four instances using the snapshot detailed in *section 1.1.2* each with a volume attached that stores the data. The size of these volumes vary depending on the role of the instance.

1.1.2 Configuring Instances

The snapshotted image used in the creation of our running instances ran the following steps:

With the instances that we created, we added unique public and private key pairings for each member of the group during the configuring stage. This was done such that each member can use their own private key to access the instances with the appropriate security implemented. In order to do this, we added an authorised keys file to the `/home/ubuntu/.ssh` directory

Here, we also installed some specific dependencies to the instances in order to have the software run appropriately. We ran general linux updates with `sudo apt update` and `sudo apt-get upgrade`.

We then installed docker manually through the following steps:

1. installing dependencies such as curl, gnupg, ca-certificates and lsb-release
2. configuring a docker repository
3. Starting the Docker Daemon
4. Installing the following packages: docker-ce, docker-ce-cli, containerd.io, docker-compose-plugin

After this stage, a snapshot was taken under the name `docker installed`

On our Ansible master instance, we also installed Ansible with `python3 -m pip install --user Ansible`.

1.2 Dynamic Deployment Using Ansible

In order to run this following section, simply run the code below in the Ansible master instance (instance 1). This code runs each of our Ansible scripts that setup the instances as required to run the

```
. /home/ubuntu/Ansible/initial-setup.sh
```

1.2.1 Deploying CouchDB

CouchDB is deployed to run on instance one, two and three, with instance three being the master node.

This stage consists of two roles:

1. couchdb-setup, this step creates a docker container on each of the machines running CouchDB.
2. cluster-setup, this step creates a cluster of the CouchDB nodes created in the previous step.

1.2.2 Deploying Data Crawlers

Deploying the docker instances that held the CouchDB databases is based on the Ansible script(`deploy-twitter-harvester`). The docker container with an image(`jonghop/twitter-docker`) is created and launched on the virtual machine (`twitter-harvester`) with a CouchDB container. The condition of the twitter harvesting is locating the raw tweets data in volume which attached in the same virtual machine. If it not exists, it will periodically check the file and give warning messages.

1.2.3 Deploy Back-end Server

The task of deploying back-end server is handled by the Ansible script(`deploy-backend`) when the main Ansible script(`initiate`) is executed. The docker container with an image(`jonghop/flask-docker`) is created and launched on the virtual machine (`web-server`). The environment file which records `COUCH_PASSWORD, MASTER_NODE, LAST_SERVER, THRESHOLD` is stored at (`/data/flask-setting`).

1.2.4 Start Front-end Application

Similar to the process of deploying back-end server, the front-end server would be deployed by the Ansible script(`deploy-frontend`) when the main Ansible script(`initiate`) is executed. The docker container with an image(`jonghop/react-docker`) is created and launched on the virtual machine (`web-server`). The environment file which records `REACT_APP_BACKEND_URL, REACT_APP_BACKEND_URL_2` is stored at (`/data/react-setting`).

2 System Architecture and Design

Our model is designed around a connected system of Virtual Machines on the Melbourne Research Cloud that take advantage of the communication between themselves to distribute the overall load of more intense operations. As demonstrated in Figure 1, our model is designed such that each instance does not use more than two docker containers at a time, and when required, the website can scale to get extra help.

2.1 Rationale for System Design

We designed our system with careful considerations in terms of automatic deployment, instance workload balancing, dynamic scaling, database fault tolerance with reasons listed in Table 1.

Table 1: System Design Considerations

Component	**Explanation**
Ansible	• Our goal is to realize user-friendly deployment of our application, and the utilisation of Ansible scripts greatly simplifies the deployment process.
Instances	• We aimed to achieve a balanced and optimal distribution of workload across each instance, resulting in the decision to create four instances with 2 CPUs. This choice strikes a balance between manageability and CPU power. We opted against selecting eight 1-CPU instances due to potential management challenges and concerns regarding CPU power adequacy for task performance.
Dynamic Scaling	• Our objective is to implement dynamic scaling for both the Mastodon harvester and the back-end service, ensuring our application can handle increased demand for Mastodon data and user traffic.
Database	• To enhance the reliability of our database, we leverage fault tolerance measures. Specifically, we have created two additional replicas on instance 1 and instance 2, ensuring our application continues to function even in the event of unexpected shutdowns of one or two instances or Docker containers.

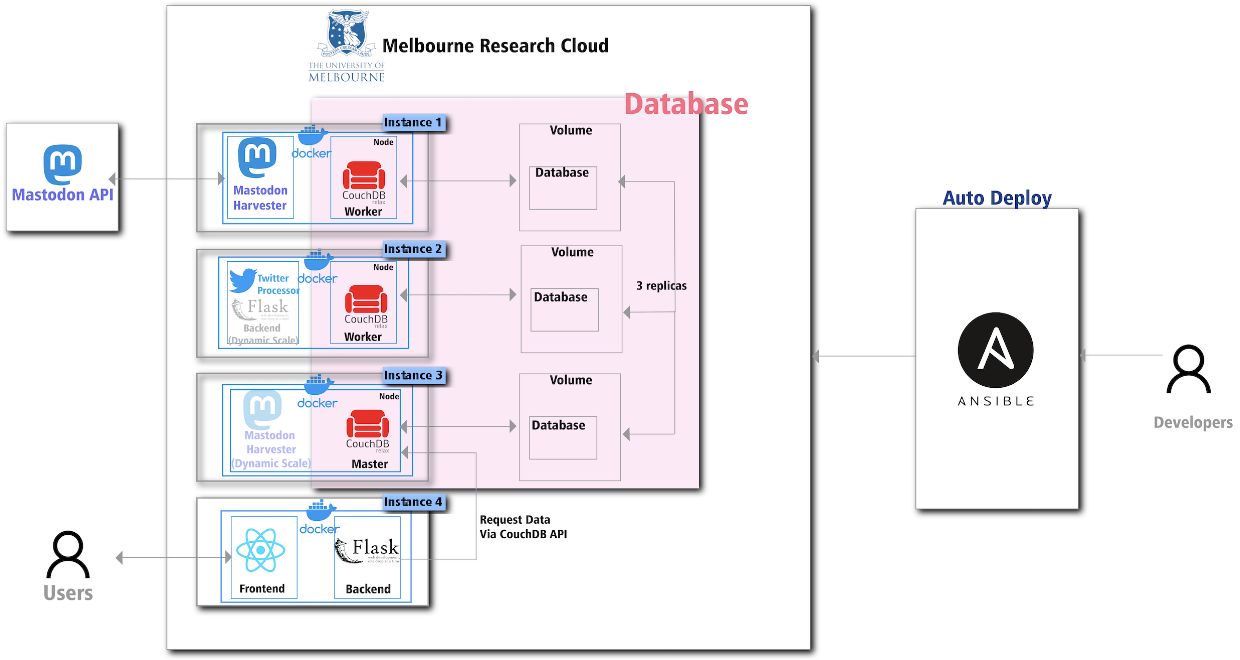


Figure 1: System Architecture

2.2 Allocated Resources

We utilise four instances on the Melbourne Research Cloud [1], each of these are built with 2 Virtual CPUs and 9GB of RAM. Meaning that, in total we are using 8VCPUs and 36GB of RAM on the MRC.

For our volumes, we used three volumes to store our data. Our CouchDB master instance has a 150GB volume attached, and our other two instances have 100GB volumes attached. This means that we have 350GB of space used in the MRC for this project.

2.3 Instances

Our model consists of four instances that each serve a specific purpose:

2.3.1 Instance 1 (mastodon-harvester)

Instance 1 is our Ansible master node that calls the dynamic deployment of our project. This instance runs the initial-setup.sh script, which divides the tasks amongst the virtual machines appropriately. This instance contains a node in the CouchDB cluster, which stores its data on the attached volume at `/volume/CouchDB`. Additionally, when the deploy-mastodon-harvester role is called, it does so on Instance 1, the details of how this harvester functions can be seen in *section 3.1.2* . This means that once the initial setup is complete, Instance 1 is running two Docker containers, one containing a CouchDB node, and the other containing the Mastodon Harvester.

2.3.2 Instance 2 (twitter-harvester)

Instance 2 has a Docker container that holds a CouchDB node. This node's volume is attached at `/volume/CouchDB/`. This instance also hosts the Twitter processor. This processor has two tasks, one which gets the raw Twitter data and processes it to a smaller, more usable form. It then has a second task which is to upload the processed data to our CouchDB cluster. When scaling is required for our back-end, this node actually has the capacity to take on an extra Docker container that runs the back-end for our website in order to take some traffic away from our website master instance. It is a simple deployment of a container that runs the Flask back-end of the website, which can then be taken down when the demand for the website is back to a manageable state for the fourth instance.

2.3.3 Instance 3 (couchdb-master)

Our third instance's primary responsibility is to be the CouchDB master node. This means that when the CouchDB-setup script and cluster-setup script are run, they are done so from the third instance and the master node Docker container for the CouchDB cluster is on this node. In addition, this instance also has the capacity to take on extra container as the Mastodon harvester for scale data retrieving to other country server.

2.3.4 Instance 4 (web-server)

The fourth instance in our system is the website instance. When our initial setup script is run, the second last thing it does before running monitoring is deploying the web server. This means setting up the back-end which is coded with Flask, and deploying the front-end which is written with React.

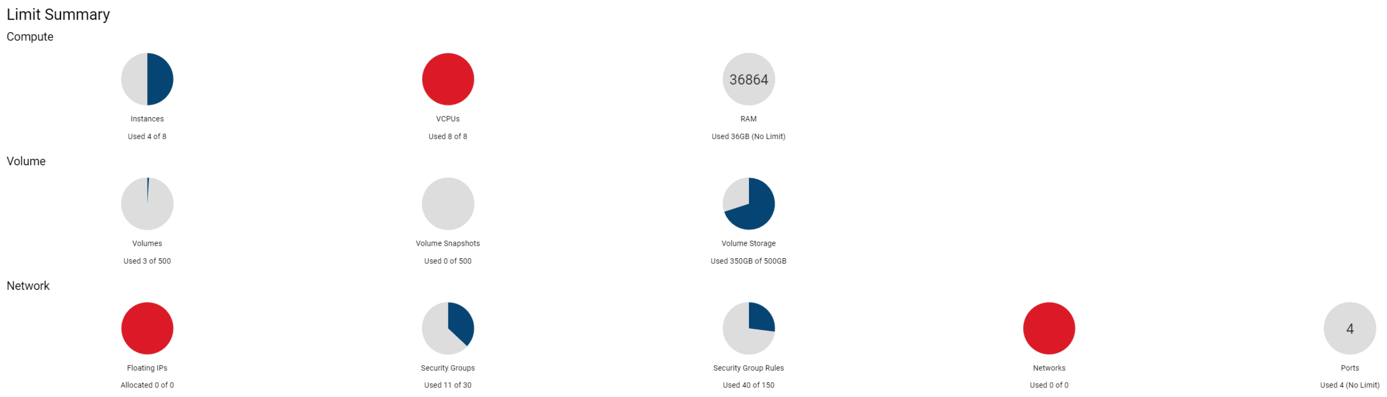


Figure 2: MRC Usage

2.4 Application (functionalities)

In this section, the functionality and information of each application utilised in this project would be introduced. Functionalities of Mastodon harvester, Twitter processor, front-end, and back-end will be introduced.

2.4.1 Mastodon Harvester

The Mastodon server facilitates API services, enabling operations such as data retrieval and content publishing. it can be accessed only real-time streaming content related to data retrieval. Therefore, it is important to handle errors that may occur in API requests and maintain reliable functionality.

First, when searching for live content, there may be times when no one has posted it at a certain time. As a result, the response may be delayed, exceeding the request timeout. This may result in an error causing the harvesting system to stop. Therefore, it is essential to identify and properly deal with this problem within the harvesting system. Specifically, the program need to capture errors so that the system can continue to request content after a certain period of time. In addition, if the collection instance's Internet connection is unstable or if the Mastodon server rejects the instance's request due to too many requests, the harvester must wait for a period of time before trying again.

Second, the stable functions of the Mastodon harvester include a variety of other tasks, such as natural language processing, happiness scores, decision-making based on happiness behavior, and database updates. It is important that these functions do not interfere with the harvest process. This requires the appropriate identification and management of errors arising from these features, taking into account various scenarios. In addition, the processing speed of these functions should be reasonable or allow multiprocessing interfaces. For happiness mining, scoring algorithms and logistic regression should be performed within a reasonable processing time range, which means that content retrieval should be fast enough to avoid delays. For natural language processing, when creating docker containers for mastodon harvesters, the packages required for stop words and streaming are downloaded. Tokenisation applies to all Mastodon content and can also be completed within a reasonable period of time. The amount of time it takes to upload data to CouchDB can vary because of the heavy load of internal database functionality or response to other requests. As a result, processed data is collected and uploaded in bulk to CouchDB, reducing the number of upload requests. In addition, if the delay exceeds a certain time limit, the harvester refuses to respond.

2.4.2 Twitter Processor

The processor would analyse the tweets from 60GB raw JSON data (since it is unlikely that Twitter data will be accessible through the APIs, a large volume of data will be provided directly). The process mainly focuses on extracting the information that would be used for the data analysis and identify the tweet's location (Great Capital City) based on the `sal.json` form the Assignment 1.

As the result, the 60GB raw data is shrunk to 2GB data. The processed data would be directly imported into CouchDB. Instead of saving the raw data into CouchDB directly and consuming the data storage, storing the processed data is a better choice.

Unfortunately, the Twitter processor is having single thread and the processing speed is slow. To improve that, we might introduce parallel programming to leverage the processing speed.

2.4.3 Front-end

We selected React as our front-end framework due to its various features. Front end is developed by React.js framework. React is a widely-used JavaScript library for building user interfaces, especially single-page applications. I React uses componentisation to organise the code, making the code easy to understand and maintain. Componentisation also means we can reuse the previous code in other part of application or even in other applications. Additionally, React has lots of third-party libraries can be used to help create various types of data visualisations, such as D3.js, Chart.js, Recharts, etc.

2.4.4 Back-end

We chose Flask for the development of our back-end service due to its lightweight features and its compatibility with our front-end. Firstly, our project doesn't have high requirements for the back-end, so Flask, as a micro-framework, is the most suitable. We can add and configure various plugins. Also, it is easy to use, and Flask's API is designed to be intuitive and user-friendly. Because it's a Python framework, understanding a basic Python is sufficient to comprehend the code. Additionally, it can easily integrate with other Python data processing libraries like Numpy and Pandas. Flask can be connected with React, using RESTful requests.

However, it has its downsides. As a micro-framework, its performance is not as good as Node.js or Go. Also, in terms of functionality, it doesn't compare to full-featured frameworks like Django.

In conclusion, Flask is a versatile and user-friendly framework for back-end development, particularly when used in data visualisation projects. Its simplicity and easy integration with Python's extensive data processing libraries make it an excellent choice. However, it's essential to consider the project requirements thoroughly. While Flask offers ample functionality for lighter, more straightforward applications, for more complex projects demanding higher performance or comprehensive built-in features, more robust frameworks like Node.js or Django might be more suitable.

2.5 Ansible

The reason for using Ansible is attributed to automated deployment, containerisation support, and improved fault tolerance. Details are listed in Table 2.

Table 2: Reasons and Explanations

Reason	**Explanation**
Automated Deployment	• Ansible plays a pivotal role in streamlining the deployment process by enabling automatic deployment of our application.
Containerizsation Support	• Ansible scripts allow us to leverage containerisation, particularly in setting up Docker containers. This facilitates efficient management and deployment of our application components.
Improved Fault Tolerance	• Ansible contributes to enhancing the fault tolerance of our application. In case of unforeseen errors or issues, Ansible scripts enable us to easily reset and restore the system to a stable state.

Ansible on our system consists of two core components the initial setup dynamic deployment, and the scaling. The following section will describe these two components in detail.

2.5.1 Hosts

The information of hosts (the four instances we have) is written in `/home/ubuntu/Ansible/host.ini` which allows Ansible to manage the tasks across the machines. To give the permission for Ansible, the ssh key `mrc_key.pem` also has been provided. To avoid the key being too open and having security concerns, the key file is set by `chmod 400` which is read only.

Below are the hosts and Groups in Ansible:

Instances

- `[mastodon] Ansible_host=172.26.130.198`
- `[twitter] Ansible_host=172.26.133.155`
- `[CouchDB] Ansible_host=172.26.128.127`
- `[web] Ansible_host=172.26.134.134`

Database

- `mastodon`
- `twitter`
- `CouchDB`

WorkerDB

- `mastodon`
- `twitter`

2.5.2 Initial Setup

The initial-setup script runs all of the Ansible scripts in order.

- `volume-setup`
- `CouchDB-setup`
- `cluster-setup`
- `deploy-mastodon-harvester`
- `deploy-twitter-harvester`
- `deploy-backend`
- `deploy-frontend`
- `start-monitoring`

2.5.3 Volume Setup

Even though the volume setup is set up on MRC, the volume still need to be attached on the virtual machine which would be ran by the following commands:

Reformat volume

- `sudo mkfs.ext4 /dev/vdb`

Mount

- `sudo mount /dev/vdb /volume`
- `sudo chmod 777 /volume`
- `sudo mkdir /volume/CouchDB/`
- `sudo chmod 777 /volume/CouchDB`
- `sudo mkdir /volume/data/`
- `sudo chmod 777 /volume/data`

These commands create the file system for the volume, and would also create the directories for mounting to CouchDB docker container and make sure the `/volume/data` exists for storing environment files. As we detail in *section 6.1.2*, Docker requires very open permissions for a folder to be used as a Docker Volume.

2.5.4 Applications Setup

The applications in the project including Mastodon Harvester, Twitter processor, front-end and back-end server would be deployed by the Ansible script. Every application would need a environment file (in .yaml or .env) to be set up, therefore the Docker image would not need to be rebuild if the hosts or password are changed. Furthermore, placing the environment files in the right directory is important since the the file would be mounted by `(--volume)` while the Docker container starts.

2.5.5 Starting Monitoring

The script is going to replace the Crontab of the system and it is allocated the task of constantly checking the primary server. It will be further discussed in *section 2.8.3*

2.6 Containerisation

Containerisation is used for Separation of the environments for examples, front-end use react based packages however back-end is running by flask. The containerisation prevents collision of the two running environment and isolation of those component give scalability and convenience to replace parts in case of trouble.

2.6.1 Docker

Docker is one of the most widely used containerisation tools. Docker can build virtual environment using Docker file that contains setting of image of environment and instruction for installations and other setting. Docker allows opening ports and forwarding ports, also the container can connect to external storage and share the resources with other containers.

In our services, docker is used for containerise our micro-services such as front-end, back-end, twitter harvester and mastodon-harvester. Each of container can be scalable with configuration in yaml file.

2.7 Mastodon Dynamic Scaling

To collect more data on Mastodon, installing a new Mastodon harvester is necessary. The backup harvester is set on instance 3 by default.

2.7.1 Install Backup Mastodon Harvester

By executing the Ansible script(`backup-mastodon-harvester`), a new harvester would be deployed automatically in the docker container that is on instance 3.

2.7.2 Remove Backup Mastodon Harvester

To remove the backup `mastodon-harvester`, it just needs to execute the Ansible script(`remove-mastodon-harvester`).

2.8 Server Dynamic Scaling

Apart from dynamic scaling with Mastodon harvester, we also implemented dynamic scaling with our back-end service to accommodate increased user traffic as demonstrated in Figure 3.

2.8.1 Server Side

Since the cost time about fetching data from CouchDB would be expensive, we attempt to optimise the speed of responding on the server side to provide a better user experience. To deal with overloading requests when there are too many visitors browsing the website, an additional back-end server would be deployed when the number of the request's counter exceeds the threshold that set in the environment file. The counter would record every request and it would be refreshed every minute. While the number is over the default threshold, the server will return {`forwarding:true`} to the client side, so the client would be leaded to fetch the data from the back up server and therefore the client would not keep waiting on the server where is already struggle to deal with tons of requests.

2.8.2 Client Side

The front-end web would fetch the information from the primary server. Unless the server response announces that it is not taking any more requests, the client side would visit the second server and the IP address of the two servers are stored in the front-end environment file. Moreover, there is a potential connection failure while the backup server is deploying and the client side is trying to send the request to it. To handle the situation, the front-end will re-fetch the information automatically from the default server again after thirty seconds, and if the server is still overloading, the client would send the request to secondary server which should be deployed already. The error-handling is expected to improve the user experience.

2.8.3 Crontab

The Crontab is the system within Linux used to run scheduled tasks. We used this in conjunction with Ansible as the backbone of the server dynamic scaling. The Crontab would execute the Ansible script, which checks the current loading status of the primary server every minute. By reading the Boolean value from the response, the script would decide if a new server should be deployed or the backup server should be removed based on the server demand.

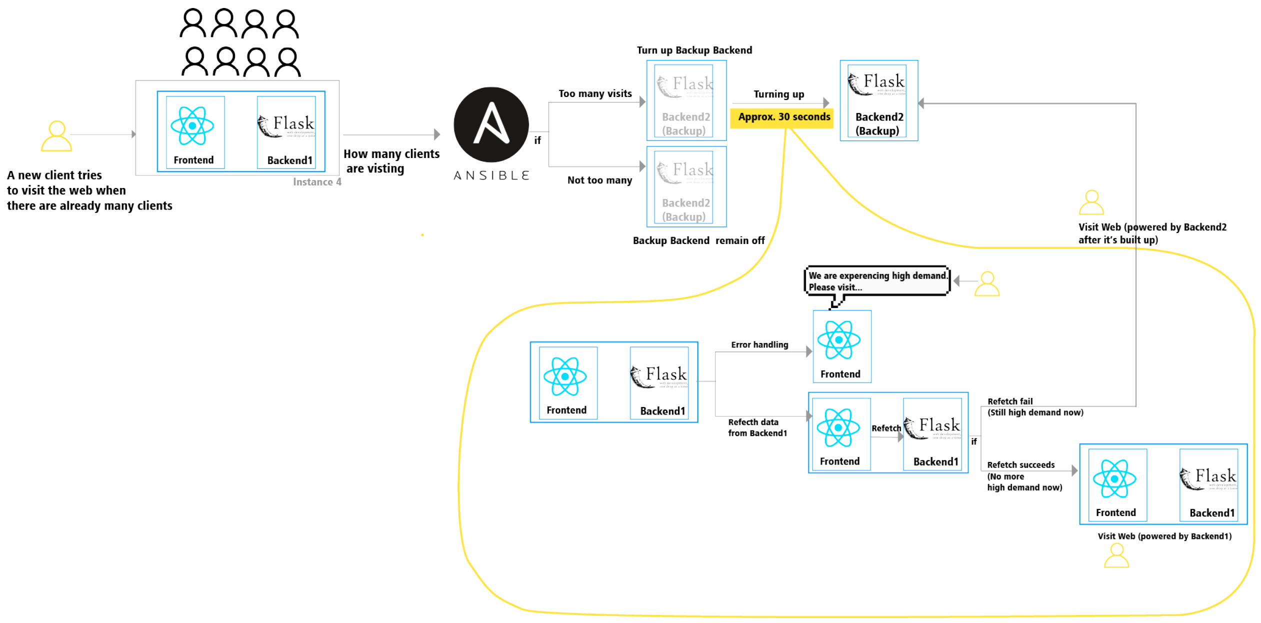

A new client tries to visit the web when there are already many clients
Frontend
Flask
Backend1
Instance 4
How many clients are visting
ANSIBLE
if
Too many visits
Not too many
Turn up Backup Backend
Backend2 (Backup)
Backup Backend remain off
Turning up
Approx. 30 seconds
Visit Web (powered by Backend2 after it's built up)
We are experencing high demand. Please visit...
Error handling
Refecth data from Backend1
Refetch
Refetch fail (Still high demand now)
Refetch succeeds (No more high demand now)
Visit Web (powered by Backend1)

Figure 3: Dynamic Scale

2.9 CouchDB Database

CouchDB container is set up by two-step of Ansible script(CouchDB-setup and cluster-setup). First the three CouchDB container are created in each virtual machine with volume attached. The Ansible script allocates the data storage of CouchDB to external volume. After creating CouchDB nodes, the second step of Ansible script will decide the master node and connect worker nodes as clusters. For CouchDB's default settings, three replicas are set up on the database system to increase availability and reliability preventing data loss in the event of two nodes failing. Shrading in Couchdb is set to two for scalable options when the extra node is set up and join into the cluster. With three replicas and two shards, the node can be added up to six nodes.

3 Data Management

This section will illustrate the data management process in terms of data collection, data filtering, data preprocessing, feature engineering, for Twitter, Mastodon, and SUDO data.

3.1 Data Collection

In this study, we collected data from Twitter, Mastodon to perform comparative analysis on the difference between Twitter users and Mastodon users sentiments. In order to deliver a more comprehensive and precise assessment of happiness displayed on social media, we utilised additional datasets to form a happiness index. We also collected SUDO data to perform correlation analysis to find influential factors on happiness score.

3.1.1 Twitter Data

A 63G dataset was provided from COMP90024 for this study. However, since our project focuses on the data with geo location, only data with location will be studied.

3.1.2 Mastodon Data

The Mastodon data is archived by API requests. The data is live-streaming and does not contain location information. Compared to tweets data, the Mastodon does not provide tokenisation of contents or sentiment of them. So it needs to be processed to tokenise the content for calculating happiness score and identifying happiness behaviours.

3.1.3 Additional Dataset

Two crowdsourced dataset was used support our data analysis as additional. The first dataset was used to engineer "happiness score" feature for each tweet and mastodon. The second dataset was used to built a machine learning model to classify reasons for happiness. Details will be discussed in *4.1 Related Work*.

3.1.4 Spatial Urban Data Observatory (SUDO) Data

To analyse social and economic factors that would impact the regional happiness score, we collected data from SUDO [2] with GCCSA aggregation level.

Social Factors: Population, median age, number of people not married, household size, number of people participating higher education (post graduate, bachelor, graduate diploma) for each GCCSA were selected as potentially influential social factors on happiness.

Economic Factors: Unemployment rate, labor force participation, weekly rent, weekly income by family, and weekly income by individual for each GCCSA were considered as potentially influential economic factors.

3.2 Data Filtering

As our analysis focus on geo-located English tweets with Greater Capital City Statistical Areas (GCCSAs) as granularity, we filtered for data with location recorded as `"full_name"`. Among the 63G Twitter data, 1.8G (3,226,684) data was found to be attached with a location. The GCC code and Suburb and Location (SAL) code was attached to the tweet by matching `"full_name"`. However, due to the mismatch between the location name on Twitter and the GCCSA, a total of 615,698 (19%) data entries could not be associated with a corresponding GCCSA or SAL code.

Additionally, as our analysis focused on English tweets, among the 1.8G data 2,512,076 (77.8%) English tweets were filtered.

3.3 Data Preprocessing

A happiness score was calculated for each tweet and mastodon with the algorithm illustrated in the *4.2 Algorithm Design* section. 1,807,963 and around 270,000 data were attached with a non-zero happiness score for Twitter and Mastodon data respectively.

For Mastodon Data, unlike tweets data, it does not contain location information and it only allowed live streaming retrieving. The Mastodon content is tokenised using nltk package in Python. For tokens in Mastodon, the happiness algorithm is applied to calculate happiness score for each content and using machine learning, token is vectorised as input of the machine learning model and identify the happiness behavior of each content.

For SUDO Data, we performed standardisation to some feature by dividing the population for the GCCSA. For instance, we calculated the percentage of people not get married by dividing number of people not get married by the population. This process was also performed for the percentage of people participating high education.

3.4 Feature Engineering

"Happiness Score" and "Happiness Behaviour" was engineered for each tweet and mastodon. Happiness Score is an integer ranged from 1 (saddest) to 9 (happiest) indicating how happy the text is. Each tweet and mastodon was label with a "Happiness Behaviour" in terms of achievement, bonding, affection, enjoy-the-moment, leisure, exercise, nature. This label can tell us the reason for happiness.

The score calculation and behavior labelling process is discussed in detail in *4.2 Algorithm Design*.

3.5 Data Management with CouchDB

For CouchDB, since version 3.0, The default setting of shards is set up to 2 and 3 replicas with 3 nodes. It allows to have availability and partition tolerance. There are two databases on CouchDB: Twitter and Mastodon. All of data is processed before uploading to CouchDB with happiness score and happiness behaviours. Map-reduce functions are uploaded before the data uploading so the map-reduce functions can be processed while the data is uploading.

3.5.1 MapReduce Functions

The utilization of MapReduce functions simplifies the processing of large-scale data. In our analysis of Twitter data, we have implemented 18 MapReduce functions to examine the data from various perspectives. These functions primarily involve mapping pairs of index combinations and reducing them to generate statistical information. The derived statistics include counts, sums, and averages of happiness scores, as well as counts related to happiness behaviors such as behavior-dow, behavior-gcc-dow, behavior-gcc-month, behavior-gcc-hour, behavior-gcc, behavior-hour, behavior-month, behavior, score-dow, score-gcc-dow, score-gcc-hour, score-gcc-month, score-gcc, score-hour, score-month, score-sal, score-state, and score.

However, in the context of Mastodon, the location index is not provided, resulting in a reduction of the number of MapReduce functions to eight functions.

3.6 Data Management with Web APP

In this part we will describe how data is delivered from database to back-end and front-end. And how to use those data to visualised to the user.

3.6.1 Data Delivered from Database to Back-end

For the back-end side, we write different routers for accept RESTful api request. When the back-end receive the request from front-end, the router will access the CouchDB server and enter the database that we asking for (`mastodon/Twitter`).After that, back-end will access the name of the data and wrap those data into JSON file. And now back-end is ready to send the data to the Front-end

3.6.2 Data Delivered from Back-end to Front-end

For sending the data from back-end to front-end, we use RESTful api get request to separate different data that we used in different scenarios. On the front-end side, we will fetch the data from back-end and receive a JSON file that is contains the data we need. Also, if the we can't get any data we will have error handling to handle this situation(more detail will be discussed on 5.1).

4 Data Analysis

This section will describe how we quantified happiness of texts in terms feature engineering. Happiness score and happiness label were engineered based on scoring algorithm and logistic regression model.

4.1 Related Work

Studies have developed algorithm to quantify happiness based on a piece of text using human evaluations by crowdsourcing [3][4]. Dodds et al. used Amazon Mechanical Turk to ask the crowd to rate the most frequently used words on Twitter, Google, New York Times, and Lyrics in terms of happiness. In total, 10,223 words were rated from the range of 1 (saddest) to 9 (happiest) where one word will be evaluated by 50 individuals and for an averaged score [5]. With this measurement index, average happiness score of a corpus or text can be calculated by summing the weighted happiness score of each words in the corpus or rext.

Additionally, an NLP study classified happiness into 7 categories: achievement, affection, bonding, enjoy the moment, exercises, leisure, and nature. HappyDB project crowdsourced 100,000 happy moments and labelled each moments with the proposed categories [6].

4.2 Algorithm Design

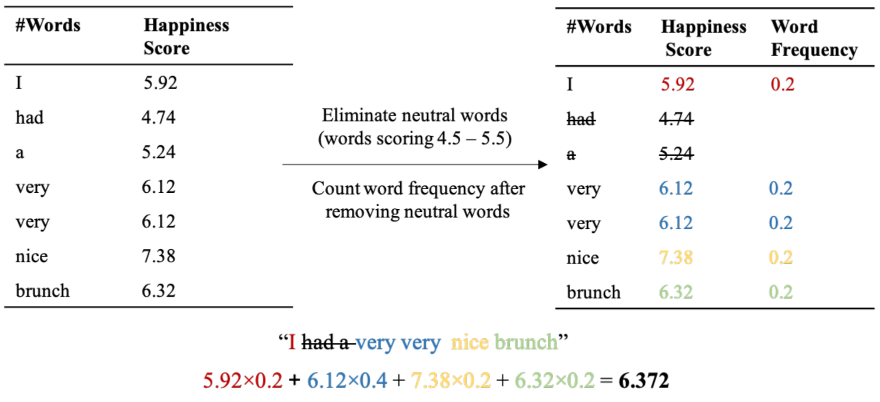
Based on previous studies, out project used the LabMIT happiness index [5] to quantify the happiness score for each tweet. Additionally, we built a logistic regression model for happiness category prediction.

4.2.1 Quantifying Happiness by NLP

Happiness score of each tweet is calculated by the following equation where where $h_{avg}(w_i)$ is the happiness score for a word and p_i is the frequency of the word in the tweet.

$$h_{avg}(T) = \sum_{i=1}^{N} h_{avg}(w_i) \cdot p_i$$

However, each tweet contains many neutral words, we eliminated words with happiness score between 4.5 and 5.5. A example of calculating happiness score of "I had a very very nice brunch." will be demonstrated as the following:



#Words	Happiness Score
I	5.92
had	4.74
a	5.24
very	6.12
very	6.12
nice	7.38
brunch	6.32

Eliminate neutral words (words scoring 4.5 – 5.5)
Count word frequency after removing neutral words

#Words	Happiness Score	Word Frequency
I	5.92	0.2
~~had~~	~~4.74~~	
~~a~~	~~5.24~~	
very	6.12	0.2
very	6.12	0.2
nice	7.38	0.2
brunch	6.32	0.2

"I ~~had a~~ very very nice brunch"

$5.92\times0.2 + 6.12\times0.4 + 7.38\times0.2 + 6.32\times0.2 = \mathbf{6.372}$

Figure 4: The Happiness Score of "I had a very very nice brunch." is 6.372

4.2.2 Happiness Classification by Machine Learning

Our study fitted a logistic regression model that labels the text in to achievement, affection, bonding, enjoy the moment, exercises, leisure. This model is built with the labelled data from HappyDB [6] and testing accuracy of our model is around 0.83.

5 Scenarios

The web system provides functionalities to analyse happiness expressed on social media platforms, focusing on tweets and Mastodon posts. It supports three scenarios: users can explore happiness trends by time and different regions and identify reasons for happiness.

Additionally, to gain insights on the difference on happiness expresses on Australia and global social media platforms, we compared the happiness scores derived from Australian tweets with global Mastodon posts.

To provide a benchmark for comparisons, the average happiness score was bench marked to be 5.91526 and approximately 5.709499 for Australia and global respectively. However, as the Mastodon-harvesting process is dynamic, the average happiness score for Mastodon is changing constantly.

The system presents graphical results, including charts, graphs, and maps, to facilitate understanding and interpretation. These functionalities are valuable for policymakers and stakeholders interested in promoting happiness and well-being.

5.1 Exploring Happiness Variations by Time

In the first scenario, analysis was conducted to determine "when happiness occurs" by discovering happiness trends based on the month, hour of the day, and day of the week for tweets and mastodons.

5.1.1 Happiness Trend from February 2022 to August 2022

The happiness score demonstrated an upward trend from February to July 2022, as depicted in Figure 5. This may be attributed to the growing economic, as Australian Bureau of Statistics (ABS) suggests that Gross Domestic Product (GDP) rose 0.7% in the March quarter 2022 and 2.7% through the year [7]. National celebrations and events during this period including Easter, ANZAC Day may foster a sense of community and happiness among the population.

In addition, seasonal factors may contribute to the declined happiness trend. As August is the last month of winter, and the transition from winter to spring may cause seasonal affective disorder and affect people's mood.

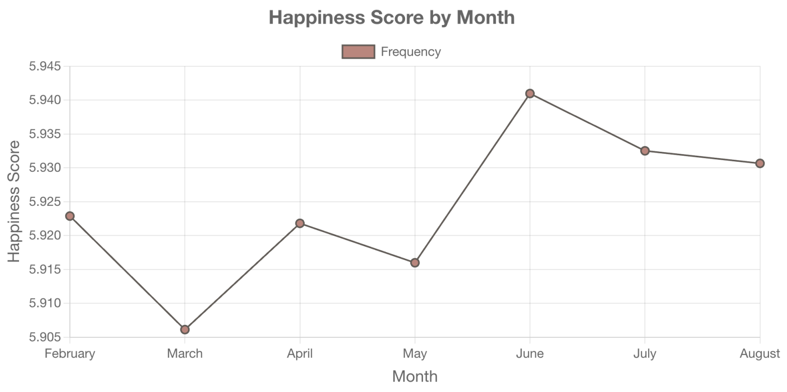
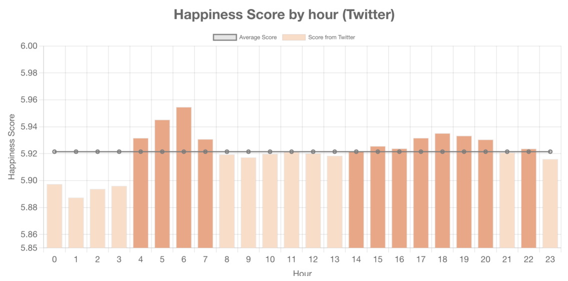


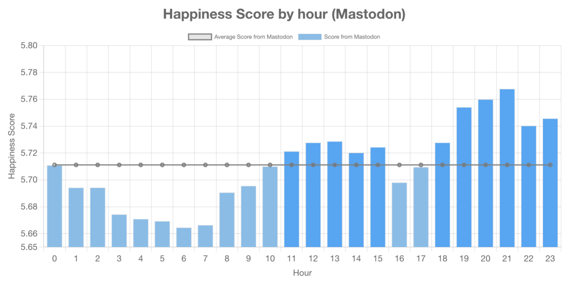
Figure 5: Twitter's Happiness Score by Month

5.1.2 Happiness Trend by Hour of a Day

The happiness score reached its highest point in the early morning and exhibited a gradual decline throughout the day. Additionally, a notable increase in happiness was observed at approximately 6 PM, resulting in a minor peak as demonstrated in Figure 6. This could be attributed to various factors such as the anticipation of leisure time, reconnecting with loved ones after work or school, or engaging in recreational activities.



(a) Twitter's Happiness Score by Hour

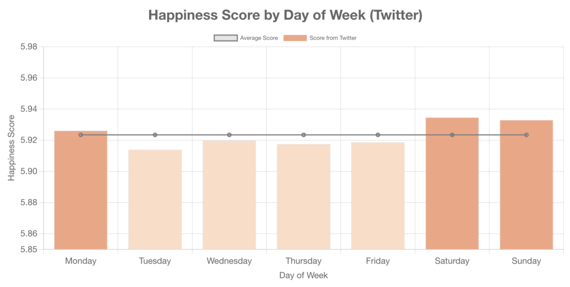


(b) Mastodon's Happiness Score by Hour

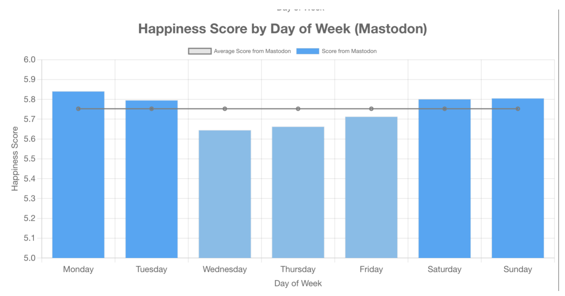
Figure 6: Comparison of Happiness Scores

5.1.3 Happiness Trend by Day of Week

As demonstrated in Figure 7, Twitter users indicate more happiness during the weekend and happiness reached the peak on Friday. Mastodon users also revealed higher happiness score among weekends. This can be attributed to factors such as relief from work or school, anticipation of leisure time, increased social interactions, and engagement in enjoyable activities. The perception of weekends as a time for relaxation and happiness may also contribute to this trend.



(a) Twitter's Happiness Score by Day of Week



(b) Mastodon's Happiness Score by Day of Week

Figure 7: Comparison of Happiness Scores by Day of Week

5.2 Exploring Happiness Variations by Location

The second scenario focused on determining "where happiness is found" by examining happiness levels across different regions in Australia, including Australian states, Greater Capital Cities (GCC), and Suburbs and Localities (SAL). By analysing regional variations, the section aimed to identify areas exhibiting higher or lower happiness levels.

5.2.1 Happiness Score Across Australian States

As demonstrated in Figure 8, Tasmania exhibited the highest level of happiness among the states, whereas Western Australia had the lowest level of happiness. Victoria, Queensland, and Western Australia were found to have below-average happiness scores. It is important to note that these findings are based on social media data and may not fully represent the overall happiness of individuals in each state.

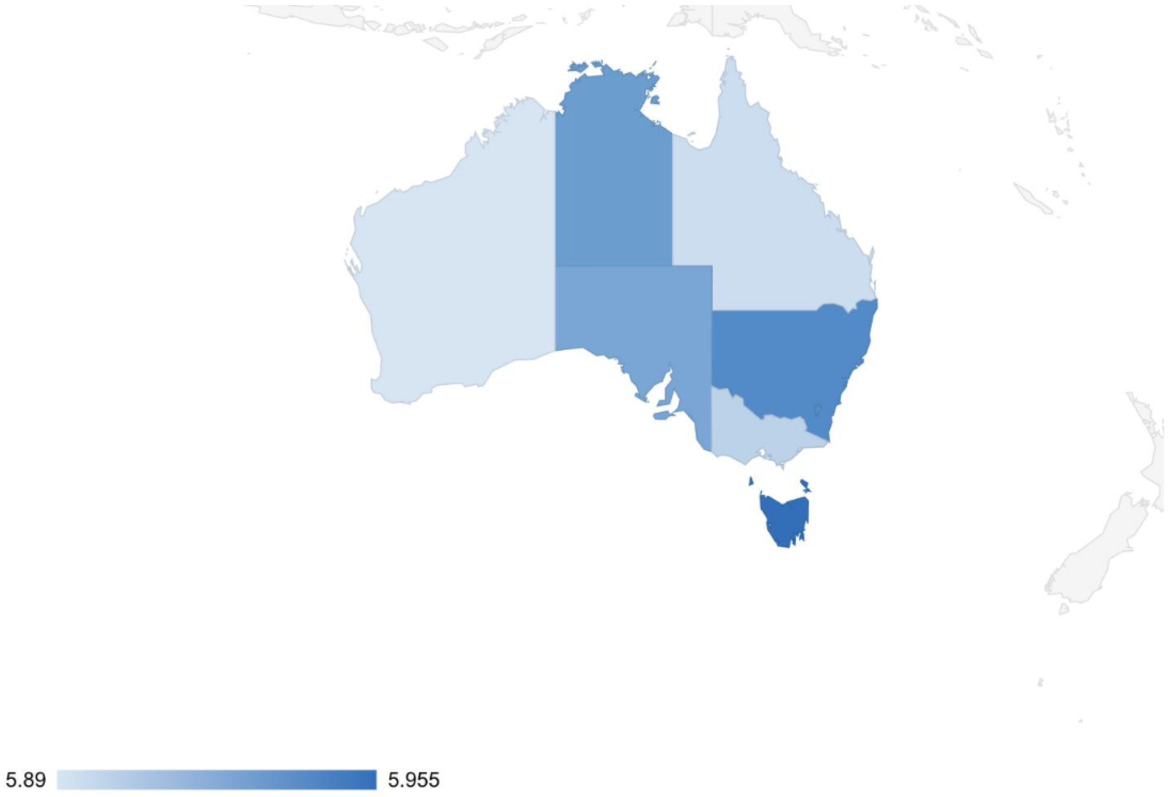


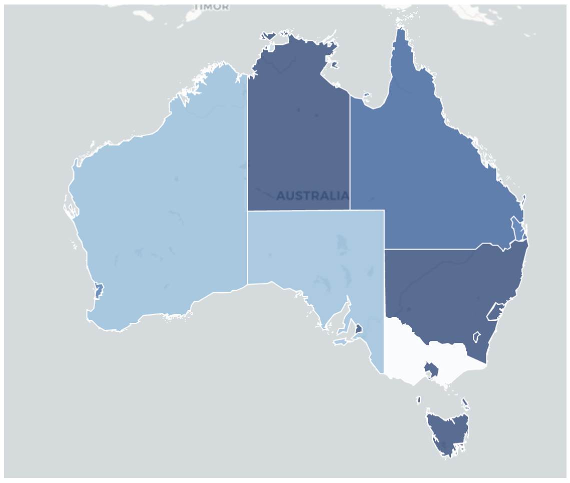
Figure 8: Twitter's Happiness Score by States

5.2.2 Happiness Score Across Greater Capital Cities and Suburb and Localities

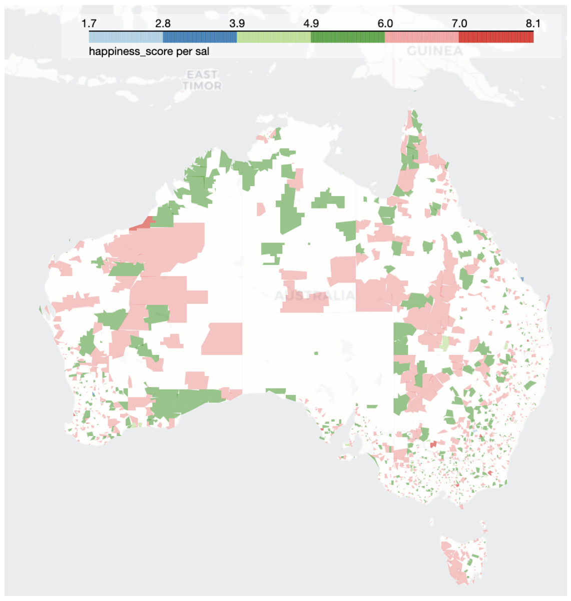
It was discovered that the Northern West region tends to exhibit higher levels of happiness compared to the Southern East region as demonstrated in Figure 8. This disparity can be attributed to various factors, such as regional climate, economic conditions, social cohesion, and cultural influences.

In addition, this study investigates the variations in happiness levels between Greater Capital Cities and Rest Areas within each state. As suggested by Figure 9, apart from Darwin, Greater Capital Cities tend to exhibit higher levels of happiness compared to Rest Areas across the country. This finding suggests that urban areas characterised by better access to resources, employment opportunities, and social infrastructure may contribute to enhanced well-being. However, additional research is necessary to explore the underlying mechanisms and relevant correlation analysis will be underlined in *4.3.3 Influential Factors*.

The utilisation of SAL provides a finer level of granularity, enabling the observation of more specific trends. Figure 9 highlights the sparsity of data, particularly in South Australia, which limits our ability to draw comprehensive conclusions for that region. Notably, the result suggests that the central region of Australia and the western part of Tasmania exhibit higher levels of happiness.



(a) Twitter's Happiness Score by GCC



(b) Twitter's Happiness Score by SAL

Figure 9: Twitter's Happiness Score by Statistical Area Level

5.3 Influential Factors of Happiness

In the third scenario, a correlation analysis was performed to identify the social and economic factors significantly influencing happiness levels in different regions of Australia. By examining various indicators such as income, education, employment, and household size etc., the study aimed to uncover the underlying determinants of happiness disparities.

To mitigate the impact of extreme values, we opted to utilise the median values for age, weekly rent, weekly family income, and weekly personal income variables. By using the median, we aimed to obtain a representative measure that is less influenced by outliers and provides a more balanced representation of the central tendency within the dataset.

5.3.1 Social Factors

Our study focused on several social factors, including population, median age, the percentage of the unmarried population, household size, and the percentage of the population participating in higher education, to explore their correlations with happiness levels. The findings revealed significant relationships between these factors and happiness as demonstrated in Figure 10. Specifically, a strong positive correlation was observed between the percentage of the unmarried population and happiness, indicating that regions with a higher proportion of unmarried individuals tended to have higher happiness levels.

Additionally, there was a notable correlation between household size and happiness, with larger households being associated with lower levels of happiness. The analysis also revealed a small positive correlation between the percentage of the population participating in higher education and happiness. However, it is important to note that age demonstrated a negative correlation with happiness, suggesting that regions with a higher median age tended to have lower happiness levels.

5.3.2 Economic Factors

The study examined several economic factors and their correlations with happiness scores. The findings revealed a strong positive correlation between the unemployment rate and happiness. On the

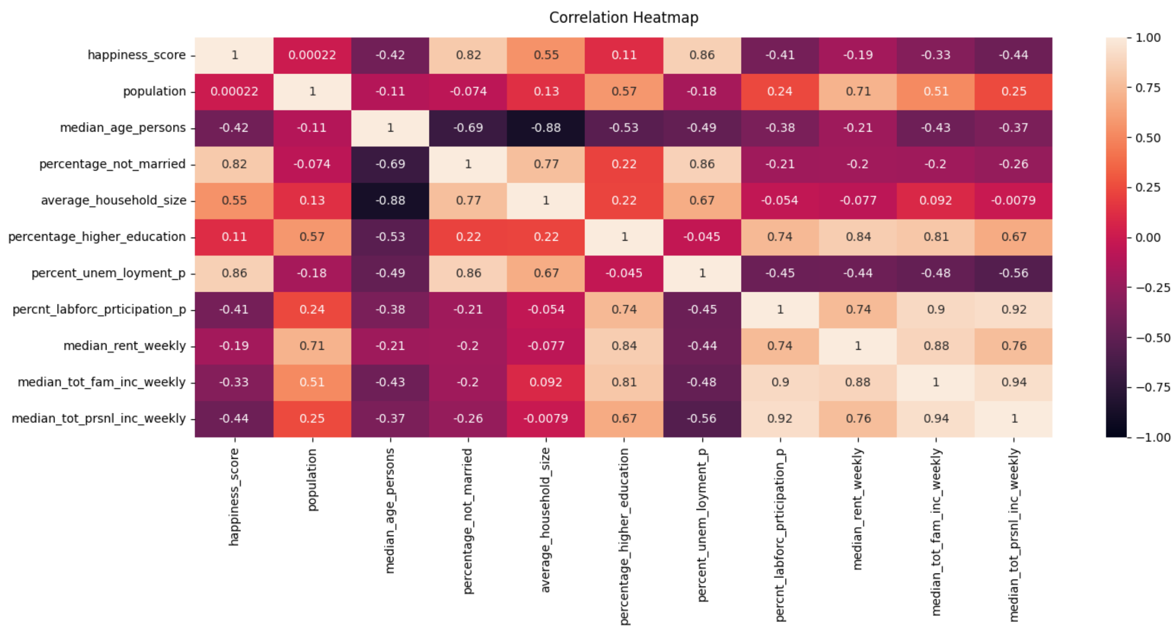


Figure 10: Correlation Map

other hand, labor force participation, weekly rent, weekly income by family, and weekly income by individual demonstrated negative correlations with happiness scores.

The positive correlation between unemployment and happiness may be explained by factors such as increased leisure time, reduced job-related stress, or the presence of social support systems during periods of unemployment. Conversely, the negative correlations with labor force participation, rent, and income highlight the potential negative impact of financial strain and work-related stress on happiness levels.

5.3.3 Classify Reasons For Happiness

Our study utilised classification to assign labels to each Mastodon post, including achievement, affection, bonding, enjoying the moment, exercise, leisure, and nature as indicated in Figure 11. Analysing the real-time Mastodon data, we observed a consistent pattern in the number of posts throughout the week. Specifically, there was an increase in Mastodon posts from Monday to Thursday, followed by a gradual decline from Thursday to Sunday. Furthermore, our analysis revealed that the majority of posts were labeled as “achievement,” while “nature” accounted for the lowest proportion. This indicates that people tend to share their achievement via social media.

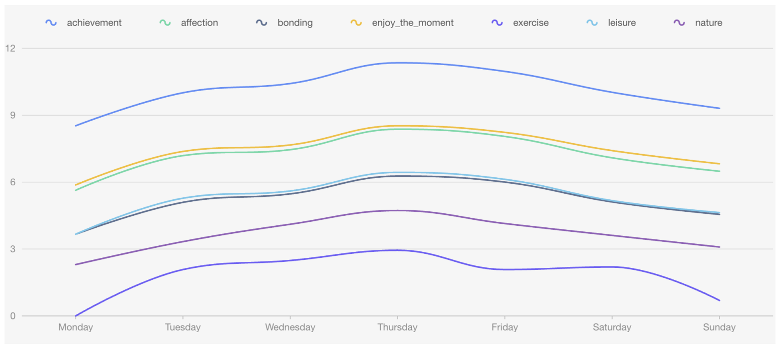


Figure 11: Classification on Reasons of Happiness by Day of Week (Mastodon)

6 Challenges

This section will demonstrate challenges we encountered as we tried to leverage the fault tolerance of our application. Additionally, data limitations will also be mentioned.

6.1 Error Handling

This section demonstrates the challenges and error handling process adopted in our project. We encountered issues arising from complex data structures, inconsistent data formats, network connectivity, and potential hardware problems. We will detail our error handling strategies during data preprocessing, CouchDB operation, front and back end development, and data pipeline process.

6.1.1 Error Handling with Preprocessing

During data filtering for data with geo location, error handling steps were taken. For each tweets with the location is presented as `"full_name":"Sydney,New South Wales"`. Due to the complex multi-level structure of JSON, `"full_name"` only exists when there is a `"place_id"` in `"geo"`. However, the format of the tweets are inconsistant and the `"geo"` key may be missing in some tweets. Hence, we used `try` and `catching error` for the inconsistency.

6.1.2 Error Handling within the Virtual Machines

Docker requires very open permissions for a folder to be used as a Docker Volume, and as such, we created a deliberate system in our Ansible setup script that creates appropriate directories for our Docker volumes to be stored. This is detailed in *section 2.5.3*.

6.1.3 Error Handling with CouchDB

From data pipeline to CouchDB, the `CouchDB package` is used to connect with Restful design. However, poor internet connection between CouchDB instance and data pipeline system cause uploading data to CouchDB. In this scenario, there is a maximum wait time set to 3 minutes after request, if the pipeline system did not get the response or CouchDB fails to save data. The pipeline system wait for 10 seconds and retires periodically.

There is possibility that CouchDB instance is broke for some reason such as docker container faulty or hardware faulty. For preventing losing data from such cases, CouchDB store data external space from docker container. It provides scalable storage options and data safety from container faulty. Furthermore it allows flexibility that possible to stop, restart and relaunch CouchDB without data loss.

6.1.4 Error Handling with Web Server

Front End. Error will happen on front end when connection between front end and back end is lost. To handle this error, if the RESTful api request get 500 or 404 error from back end, it will have a pop up window display the error message to the users.

Back End. For the Back End error handling, as we use RESTful design and Flask, the request from front end are required specific parameter to access different scenarios so the system will not have any conflict with parameter request. On the other hand, when the back end can't access the Couch DB,it will catch the error and send the 500 error to front end and let front-end know that we can't access the data from database.

6.1.5 Error Handling with Data Pipeline

Twitter consists of large JSON formats it has various forms of posts depends on many materials such as user status and device, app versions. Some of tweets do not have location data and some of them have additional futures such as bounding box. So understanding various form of structure in tweets and finding common data structures are important. After finding common structures, it need to handle error when the different structure of tweets come up. In this case, the error will be caught on python script and move to next post. In additional, it is possible to count unstructured posts and conducting additional process to match with the common structure of tweets.

We included time-out error handling with Mastodon data. The Mastodon platform is accessed through a web API, where the response time can vary depending on the content of users' posts, potentially resulting in timeouts. If time out happens, the python script will catch the error and wait for 10 seconds and requests posts periodically.

6.2 Limitations

The limitations of Ansible, MRC and data will be discussed in this section.

6.2.1 Ansible and MRC limitations

During the time when we were learning how to use Ansible and how it interacts with the MRC, we were limited by Ansible being unable to create, and then access Virtual Machines on the cloud through automation. The solution to this, as can be seen *section 1* , was to create the Virtual Machines manually and input our static data before running the automated parts of our project.

6.2.2 Data Limitations

Limitations data collection involved access restriction to Twitter's API. Mastodon data was used as an alternative but it lacked location variables and required manual preprocessing. The granularity of the social economic index provided by SUDO may have been insufficient. There was also a temporal mismatch between the available SUDO data and the Twitter data. Furthermore, the findings on the correlation between unemployment and happiness differed from conventional knowledge, possibly due to the indirect measurement of happiness through social media and biases in the Twitter dataset.

6.2.2.1 Twitter

Unfortunately, when the project was being set up, it became apparent that Twitter was not going to allow new users access to their API, meaning that we were not able to stream tweets to our system and have the live data be sourced from Twitter. Instead, we were able to source our live data from Mastodon, which is much smaller in scale, when compared to Twitter, and provides a different set of information when being streamed. This created challenges in how we approached the planning stage of our task, because the Mastodon data provides a much smaller sample size when training our models in comparison to Twitter.

6.2.2.2 Mastodon

Mastodon data was pulled in a different form compared to that of the Twitter data we were supplied. Our Mastodon data does not contain location variables like Twitter does. Moreover, for the logistic regression classification model, the training dataset was unbalanced, as more texts were labelled with "achievement". This may be attributed to our unbalanced prediction results where achievement was the majority label.

Additionally, Mastodon did not provide tokenisation for each post. As Mastodon post includes many hyperlinks, the format appears disordered. Hence, necessary preprocessing and tokenisation steps were taken. It is important to acknowledge that these manual steps could introduce potential biases in the analysis.

6.2.2.3 SUDO

SUDO is unique compared to Twitter and Mastodon because it presents social economic index with a specific level of granularity (GCCSA). However, due to the relatively large statistical level of GCCSA, the granularity may not be fine enough for analysing correlations. For example, certain variables like Crime Rate, which are recorded at a more detailed level in Statistical Area Level 2 (SA2), cannot be utilised in this context.

Furthermore, it should be noted that the latest available data for analysis is from 2021, which does not align with the Twitter data collected from the period of 2022.2 to 2022.8. Although there may be minor changes in the variables over the course of one year, we made the assumption that the correlation between the variables and happiness remains valid.

Additionally, our findings contradict conventional studies regarding the correlation between happiness and the unemployment rate. This can be attributed to the indirect measurement of happiness through social media content. Additionally, unemployed individuals may be underrepresented, causing potential biases in the Twitter dataset. Therefore, caution is necessary when interpreting these results.

7 Discussion (Pros and Cons)

In this section, we will critically analyse the pros and cons of MRC, system scalability design, and tools and processes for image creation and deployment.

7.1 MRC

The MRC (Melbourne Research Cloud) runs on using the IaaS (Infrastructure as a Service) model, meaning that aside from the visualisation, servers, storage and networking, the management is left to the user. The MRC is a very useful platform for creating an accessible environment with which teams can create and use scalable systems. Although this platform is imperfect, it provides a balanced software package that benefits those with experience in development. The MRC is based on OpenStack, which groups together computing, storage systems and networking resources to provide the IaaS model previously mentioned [8].

7.1.1 OpenStack and IaaS

Using the OpenStack design means that there is a relatively small overhead price for the user as the physical element of the resources is managed with OpenStack software on many physical servers, outside of the user's control.

The IaaS model leaves much in the hands of the user, meaning that people without much experience in controlling cloud systems or using Linux machines might struggle initially with development on the MRC. This can been seen as a positive, however, because experienced personnel have the capacity to be less restricted in their approach to designing their system. Teams without much experience in resource management might prefer to use a PaaS (Platform as a Service) systems, where they can put their focus into their applications and data.

Unfortunately, we as a group experienced significant issues with Ansible in our usage of the MRC. More specifically, there were updates occurring on the cloud which were causes errors when users were creating and accessing images and volumes in the MRC. These sorts of errors where issues that are occurring are out of the users' hands are not possible with on-premises management of hardware and software. When the users are in control of their servers, storage and networking, however, the resources required in terms of software, hardware and management are significantly higher.

7.1.2 Private cloud setup

The use of private cloud software in the MRC provides a very reliable security design, where the users have much control over who can interact with their cloud. This security system, however, does require extra resources and time in order to maintain high confidence in the security setup.

Using a private cloud setup, when compared to a public cloud such as Amazon AWS or Google GCP is an interesting discussion of the value of control and whether that is worth the resources required for an individual. Using a public cloud service puts the consumer in a somewhat powerless position in terms of the security of their data.

Public cloud architecture also likely forces a user's commitment to a particular platform. Due to these companies being for-profit, their incentives are designed around producing as large a profit as possible, and hence, the businesses benefit both from getting as much money from users as they will pay, but also ensuring that they will not migrate to a rival platform. Utilising a private cloud IaaS system in the MRC provides cheaper access as its costs does not run on a per hour basis, and instead is supported somewhat through government project funding.

7.2 System Scalability

Mastodon Harvester and the server infrastructure will be discussed in terms of their limitations, and potential improvements to enhance efficiency and resource management.

7.2.1 Mastodon Harvester

Currently, the processes of scaling up and scaling down of Mastodon harvester are commenced by the user which is not complete automation, however, it does provide autonomy for the user to decide when the harvester should be deployed. There are some alternatively plans to allow the Ansible script to decide the timing of scaling up based on the user's preference. For instance, if the amount of the streaming data from AU server is less than ten per minute, another harvester targeting *world* would be deployed to collect more data. Overall, the logic of scaling up and scaling down could be added and applied to achieve full automation in this context.

7.2.2 Server

The server scalability is mainly dependant on the threshold of the requests counter provided by the primary server. Yet, the limitation of the server is unknown so far, so the current threshold is not perfect. Therefore, it might cause the two unwanted scenarios:

1. The secondary server is deployed too early when the primary server still possesses the potential to handle more requests. Therefore, the backup server is wasting resources when it could be running more on its primary task.
2. The server is having too many pending requests (or crushed) while the backup has not been deployed it. It would let the client wait the response for exceeded time and the user experience would be bad.

Since the scaling is intuitive and experimental, the scalability could be improved by using Docker Swarm or having another server as a load balancer. Either way would enhance the performance of load balancing of the server.

7.3 Tools and Processes for Image Creation and Deployment

In our case, we created the Virtual machines manually and any scaling and automation was achieved utilising the containerisation capabilities of Docker.

In our manual creation, we created blank instances using the MRC and once we had installed our requirements for Docker, we used the snapshot functionality to save these instances and create updated ones when required.Deploying these snapshots then required significantly less effort than creating a fresh instance installing the dependencies each time. This practice could have been utilised if we had used automation when creating our instances, as a means of reducing deployment time.

7.3.1 Virtual Machines

Running virtual machines in the MRC provides a malleable option in terms of how much can be achieved with them. At least one virtual machine is needed in order to run Docker applications and so there is a minimum requirement for their use. Using virtualisation to run multiple virtual machines with Ansible is an option that can provide significant use, but the scaling of this through creating new virtual machines is significantly slower than that of containerisation with the likes of Docker. Creating and running virtual machines in Ansible means needing a hypervisor running multiple virtual machines hosting guest operating systems. This is a structure that has a relatively big overhead cost because of the hypervisor and much of the base installations into a virtual machine can be unnecessary.

7.3.2 Docker

Docker is a containerisation tool that facilitates encapsulation of the technology stack. virtual environments can be created in the form of Docker images by providing an independent environment from the host operating system. These images are version-controlled, enabling reproducibility and scalability in production environments. Docker containers provide environmental isolation, increased security, application conflict mitigation, and resource utilisation optimisation. Docker container processes are also lighter than virtual machines, minimizing overhead. Leveraging the kernel of the host operating system reduces startup time, reduces memory consumption, and improves overall performance.

Managing and coordinating multiple containers in a production environment can create complexity and overhead. Setting up networking and storage management may require additional effort and expertise. As Docker emphasises isolation, it is essential to safely configure containers to prevent potential vulnerabilities. Ignoring proper configuration, access control, or using unsafe container images can pose a security risk. By default, Docker containers are designed to be stateless and do not retain data or modifications if the container is persisted. Managing persistent data in docker container requires additional considerations, such as leveraging external volumes or specialised containerisation solutions. Therefore, the suitability of the docker should be carefully evaluated, taking into account the specific characteristics and requirements of the project.

8 Innovation

Our project is innovative in our dynamic scaling and data analysis approach. We performed two-way dynamic scaling where we have enabled the augmentation of computational power for our back-end server and our Mastodon data harvester, allowing them to adapt and scale up according to the demands placed upon them.

Regarding data analysis, our method involves the use of a crowdsourced dataset to derive happiness scores. This allows us to direct our investigation specifically towards “happiness” rather than broad classifications like positive, neutral, or negative sentiments, typically found in general sentiment analyses.

Further, by utilising a machine learning classification model in our study, we’re able to label specific reasons leading to happiness. This has allowed us to comprehend the most common types of happiness expressed on social media more effectively. As such, our research provides more detailed insights into how happiness is expressed in the online world.

9 Video Link

To view our video demonstration, please visit https://www.youtube.com/watch?v=5SL0uL2fwLc.

10 GitHub Link

To view our code, please visit https://github.com/is0xjh25/social-media-analytics-on-cloud.

References

[1] The University of Melbourne, “Melbourne Research Cloud Documentation.” [Online]. Available: https://docs.cloud.unimelb.edu.au

[2] ——, “Spatial Urban Data Observatory.” [Online]. Available: https://sudo.eresearch.unimelb.edu.au

[3] M. R. Frank, L. Mitchell, P. S. Dodds, and C. M. Danforth, “Happiness and the Patterns of Life: A Study of Geolocated Tweets,” *Scientific Reports*, vol. 3, no. 1, p. 2625, Sep. 2013.

[4] L. Mitchell, M. R. Frank, K. D. Harris, P. S. Dodds, and C. M. Danforth, “The Geography of Happiness: Connecting Twitter Sentiment and Expression, Demographics, and Objective Characteristics of Place,” *PLoS ONE*, vol. 8, no. 5, p. e64417, May 2013.

[5] P. S. Dodds, K. D. Harris, I. M. Kloumann, C. A. Bliss, and C. M. Danforth, “Temporal Patterns of Happiness and Information in a Global Social Network: Hedonometrics and Twitter,” *PLoS ONE*, vol. 6, no. 12, p. e26752, Dec. 2011.

[6] A. Asai, S. Evensen, B. Golshan, A. Halevy, V. Li, A. Lopatenko, D. Stepanov, Y. Suhara, W.-C. Tan, and Y. Xu, “HappyDB: A Corpus of 100,000 Crowdsourced Happy Moments,” 2018.

[7] Australian Bureau of Statistics, “Australian National Accounts: National Income, Expenditure and Product.” [Online]. Available: https://www.abs.gov.au/statistics/economy/national-accounts/australian-national-accounts-national-income-expenditure-and-product/latest-release

[8] Openstack, “Openstack Software.” [Online]. Available: https://www.openstack.org/software/