COMP90024 S1 Cluster and Cloud Computing

Project 2 Report
Melbourne the Most Livable City…?

Group 08

Jiahui Luo: 1282009

Qichang Li: 1215907

Shengze Zhu: 955127

Yebin Ge: 1212734

Zihao Guo: 931278

Cluster and Cloud Computing

THE UNIVERSITY OF MELBOURNE

May 2022

Content

Abstract

This is the report of Group 08 for the COMP90024 Cluster and Cloud Computing subject. This report first describes the architecture of the whole system and the technology stack used. Then the data sources of the project and the scenarios for analysis are introduced. Also, the crawler, frontend, backend and database design of the project web application are presented. Finally, the Melbourne Research Cloud (MRC) resources utilised in this project are described, and a user guide is provided for users to deploy this project and how to use the web application interface.

Keywords: Melbourne Research Cloud, Crawler, Frontend, Backend

1 Introduction

This project is to develop a web application based on the Melbourne Research Cloud (MRC) platform to derive the livability index of Melbourne in various aspects by analysing Twitter and Australian Urban Research Infrastructure Network (AURIN) data and developing some scenarios.

The main architecture of this project uses four instances on MRC, and instance 1 includes three containers that run the crawler application, the backend application, and the frontend application, respectively. Simultaneously, a 3-node CouchDB runs on instance 2, instance 3 and instance 4 to store the data collected from tweets and AURIN.

This paper aims to analyze and visualize the relationship between the Rental Affordability Index (RAI) and several aspects (housing, engagement, health, neighbourhood, environment, opportunity, transport) that people care about on Twitter in order to determine the livability.

A crawler program to collect data from Twitter and AURIN and a web application to showcase the analyzed scenarios were developed. The crawler collects data while performing NLP sentiment analysis on the data and stores the results in the database. The web application consists of a CouchDB database, a backend built with the flask framework, and a frontend that uses Google maps to show the attitudes of different districts in Melbourne towards a variety of aspects.

The entire project is deployed and running on MRC, a cloud platform that provides scalable and on-demand computational resources to researchers at the University of Melbourne. This cloud platform has the advantage of being free and providing real-time computational resources, allowing researchers to focus on their research without having to think about resource overhead.

A user guide is also provided to guide developers on how to deploy the project's GitHub repository code to the MRC platform through Ansible automation, as well as to guide users on how to use the project's frontend interface.

2 System Design and Architecture

2.1 System Architecture Diagram

The architecture of the whole system is shown in Figure 1. The system uses four instances, the frontend, backend, and crawler are running in the container in the instance 1. The database consists of 3 nodes cluster of CouchDB. For the stability of the database, each node is running in a different instance of the container respectively.

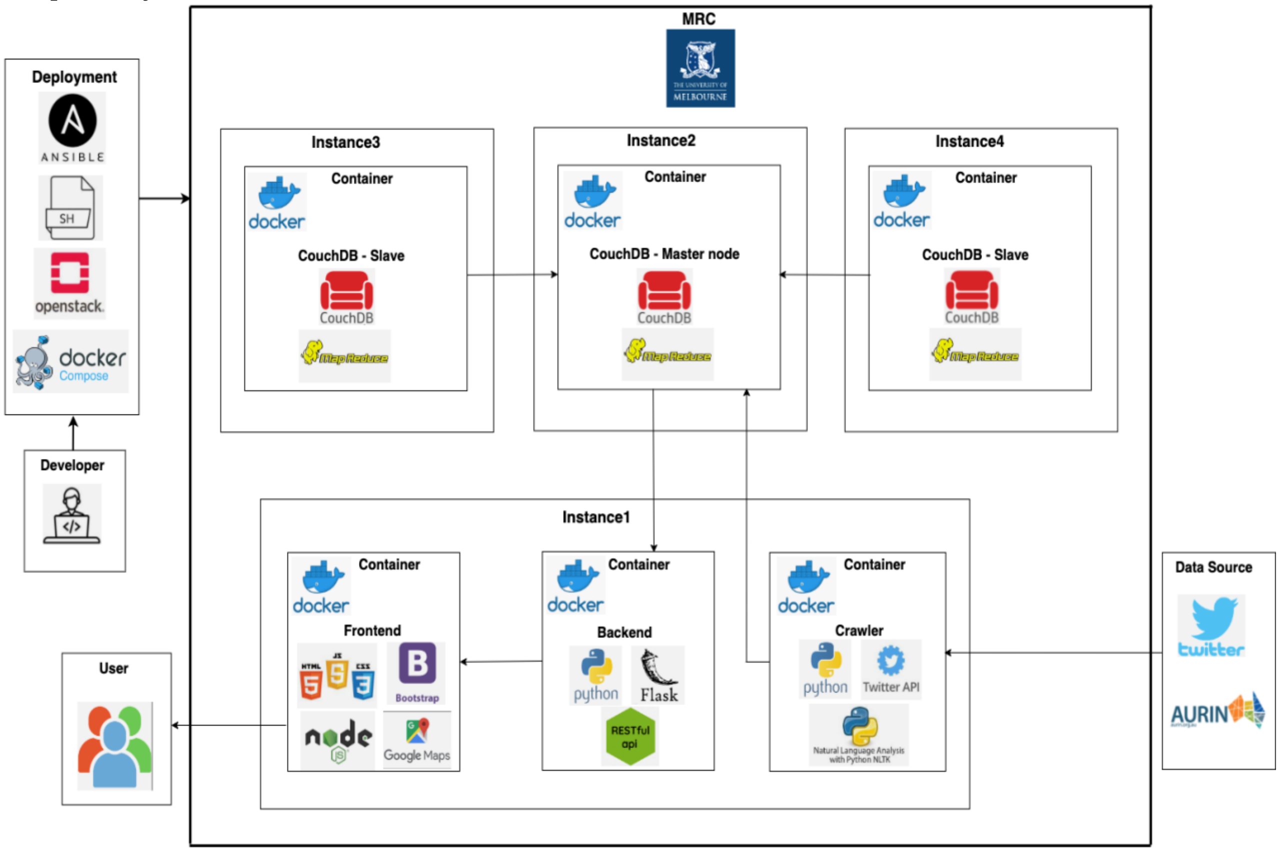


Figure 1. System Architecture

2.2 System Technology Stack

The different technology stacks are applied in the frontend, backend, crawler, and deployment accordingly.

The frontend mainly used HTML5, CSS and JavaScript, in order to increase the user experience, also used some bootstrap components and google maps API, and finally used node for packaging.

The backend is mainly written in the python programming language, and the back-end system is built by flask framework, and the endpoints are provided to the front-end with RESTful API technology.

The data for the entire system comes from Twitter and the Australian Urban Research Infrastructure Network (AURIN) platform. The crawler is mainly written in the python language and requests data from these two platforms through the corresponding APIs. The data is stored in the database after sentiment analysis by using the python natural language processing (NLP) module NLTK.

The database uses a NoSQL database CouchDB. In order to improve the security and stability of the data, a 3-node distributed cluster is established, and MapReduce technology is applied to process the large-scale distributed data and pass the aggregate results to the backend for further manipulation.

The whole system is deployed to MRC mainly through ansible and OpenStack technologies, and shell scripts are also used. To improve the development efficiency, the docker container technology is used to make the whole system run on the same environment locally and on the MRC.

3 Scenarios and Data Sources

3.1 Scenario Overview

Australia is one of the most immigrant-heavy countries, and for many Australians, rent is their most significant expense. As a result, the Rental Affordability Index (RAI) can be used to assess the liveability of a place in Australia. In this research, the cloud-based program aims to analyse the relationship between RAI and people's attitudes regarding the different topics of the tweet for each suburb in Victoria. The research is based on two datasets: the first is an official dataset concerning RAI in Melbourne released on AURIN by the Australian agency. The second dataset was crawled from Twitter using different keyword filters.

3.2 AURIN Data

The AURIN data about the RAI contains the feature RAI, longitude, latitude, and correlated region. However, the region doesn't feature any types of suburbs. Thus, an additional file, "suburb-VIC.geojson", which contains the bounding box of each suburb, is used to compare the point of the area. After processing the geoinformation, the output file of AURIN data includes each suburb in Victoria and their correlated RAI.

3.3 Twitter Data

There are two types of Twitter data used for this program.

Streaming data:

The first type applies Twitter streamer for the real-time streaming of Twitter data, as our program aims to analyse people in Victoria. Thus, only the Twitter data with the geolocation in Victoria are crawled.

For each crawled twitter in the dataset, there are two useful information text of the tweet and the geolocation for the tweet. And that information is being used to extract features of the tweet.

The text of the tweet is being used for the NLP sentiment analysis. To extract features from the text, the first step is text cleaning and preprocessing, in which all the tweets are clean to the pure text. And then, those texts are used for the sentimental analysis using the Python API text blob. Three types of sentimental are associated with the tweet, which is positive, negative or neutron. Also, the positive rate (number of positive tweet/ numbers of total tweet) is MapReduce for comparison with the RAI.

As for the geolocation, the crawled tweeter comes with the geolocation of the bounding box. However, to find the relationship between the positive rate of people's tweets and RAI in Victoria, the feature suburb of the tweet is needed. To extract the feature suburb from the bounding box, an extra file "suburb-VIC.geojson" is used; based on the bounding box of each suburb, the feature suburb is added in each tweet.

After preprocessing the tweet, each tweet is stored in the CouchDB, and those data are map reduced based on the keyword for future analysis.

Old data:

The second type applies the Twitter searcher for the old tweets; analyzing the streaming data demonstrates that the housing-related tweets can have the strongest relation with the RAI. Thus, to find more correlation between the housing-related tweet and RAI, the old tweet containing housing-related keywords from 2017 to 2022 are crawled. Same as the steaming data, each tweet is processed having the same features.

3.4 Result and Analysis

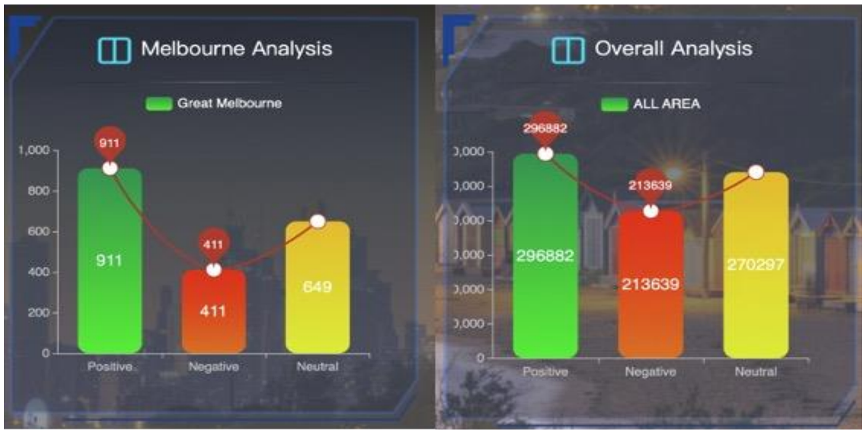


Figure 2 Overall and Great Melbourne area sentimental analysis

The bar chart above shows that the proportion of negative sentiment in the Melbourne region is much smaller than the proportion of overall tweets, which means the positive ratio of the great Melbourne is 41% higher than the positive ratio of the overall tweets, which is 38%.

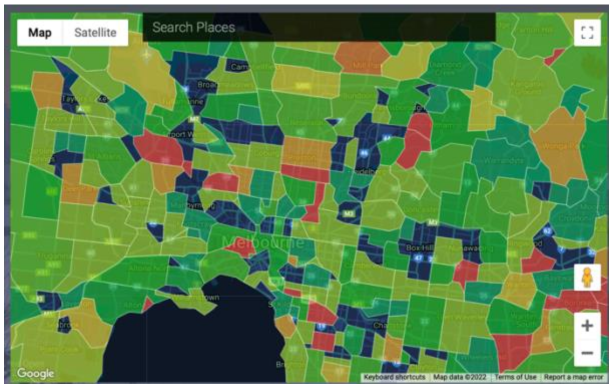


Figure 4 Housing related sentimental Map

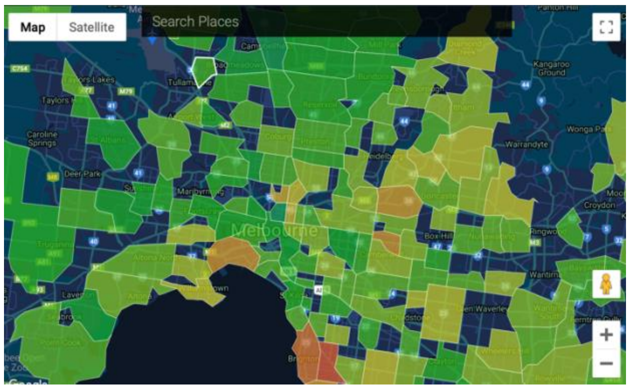


Figure 3 Rental Affordability Index Map

According to the streaming data, the positive rate of the housing-related tweets in each suburb can match AURIN's RAI, which tends to be greener in Melbourne City's area as shown in the map above. As for the other keyword related tweets, their positive rate could be lower than expected from RAI.

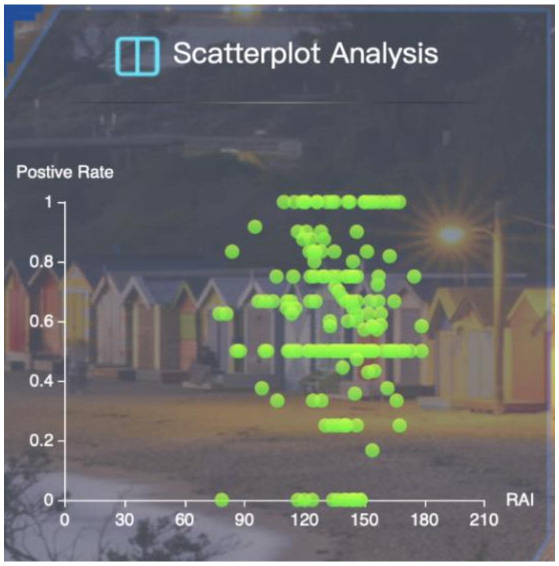


Figure 5 Relationship between Housing related tweet Rental Affordability Index

Thus, the old tweets related to housing are compared with AURIN's RAI data from 2021. As the Scatter plot shows above, the optimistic point tends to be denser with the increasing RAI. Overall, it can be concluded that the people's attitude about the housing topic in each area can correlate to the RAI in such an area.

4 Web Application

4.1 Crawler

In this project, two tweet harvesters were developed, including a Tweet Streamer and a Tweet Searcher. Both use the Python programming language. Tweepy library (Tweepy, 2022) was used In order to access Twitter API methods and CouchDB library (CouchDB, 2022) in order to store collected tweets to CouchDB, which is implemented on MRC.

Tweet Streamer

The Tweet Streamer uses standard Twitter API V1.1 to harvest streaming tweets. In standard Twitter API V1.1, a JSON response contains geo info, including bounding boxes and point coordinates which means an estimated suburb could be obtained from that information.

The overall tweets harvesting process for tweet Streamer can be summarized as follows:

1. The user passes the arguments to start the streaming process:

   i. Streaming harvest tweets by the keywords “Melbourne” to best help searching tweets in the Melbourne area. Also, the “retweet” tweets are filtered, to ensure **no duplicate tweets**
   ii. A geo bounding box in longitudes and latitudes which the streamer will cover during the harvesting process
   iii. A file name which will log the streaming process status and errors.
   iv. If no argument was passed by the user, the streamer will set the default value of the bounding box to the region of Melbourne and output the log to a filename “streamlog.txt”
   v. Use command like “python3 TwitterStreamer/tweetStreamer.py -l [144, -38.4, 145.5, -37.5] “to specify the bounding box

2. The Tweet Streamer creates an API object, Tweet Listener object and uses a pre-defined consumer key and access token to connect to the Twitter server. Then create a connection to CouchDB in order to store tweets data.
3. Tweet Listener continues harvesting tweets within the specified bounding box using pre-defined key works.
4. If a satisfactory tweet is found, it is processed into a document and structured in the following format:
   i. ID: Tweet ID the tweet’s unique ID
   ii. User: the user who creates the tweets
   iii. Text: the text of the current tweet
   iv. Hashtags: hashtags that are used in this tweet
   v. Date: the time when the tweet is created
   vi. Geo: precise point coordinate empty if the user does not turn on the precise location
   vii. Bounding box: the bounding box which contains this tweet
   viii. Estimated coordinate: an estimated point coordinate generated using the bounding box
   ix. Suburb: the suburb in which the tweet is sent in Victoria, not found if it is not located in Victoria

x. Sentiment: The sentiment (positive, neutral, negative) of the tweet text. Generated via an NLP model.

5. The document is stored in the "processed tweets" database and if an error is raised print the error information, write the error message and the document to the stream log file and sleep the process for thirty seconds.
6. After the document is stored in the nominated database, the streaming process goes back to step three and keeps looping until it is shut down by the user.

Tweet Searcher

The Tweet Searcher uses academic Twitter API V2 to harvest all tweets starting from 2017 to 2022 in order to make comparison to AURIN data. In Twitter API V2, a JSON response contains geo info, including a place id and a place name which means no exact coordinates and bounding box could be obtained from that information. However, Tweet Searcher could still harvest tweets within Melbourne without breaking them into exact suburbs.

The overall tweets harvesting process for tweet Searcher can be summarized as follows:

1. The Tweet Searcher creates a client object and uses a bearer token to connect to the Twitter server. Then create a connection to CouchDB to store tweets data.
2. The Tweet Searcher builds a search query using a set of pre-defined keywords and a point radius filter to filter out tweets that are located twenty-five miles away from the central point of Melbourne.
3. The Tweet Searcher uses a paginator to search tweet starting from "2017-01-01" to "2022-05-01" and use pointers of pages to **avoid duplicated data**. Each page contains 500 tweets
4. Carry out a sentiment analysis which is the same as Tweets Streamer and then build a document using the text of tweets and sentiment
5. Steps 3 and 4 are repeated until the pre-defined limit of desired tweets is achieved.
6. The Tweet Searcher will sleep for fifteen minutes once the rate limit is reached. And keep harvesting tweets once the Twitter API is back to use.

Handle duplication

We have set the Twitter streamer and searcher to only return unique tweets. In addition, we store the searching and streaming tweets separately in different databases and use them for different functionalities, so there are no duplicate tweets.

4.2 Database

CouchDB

Apache CouchDB, which is a distributed document-based NoSQL database, is chosen to store and persist the data for this project. It is a great choice for the project with bit data since the distributed database is able to distribute the computing load across multiple servers and therefore increase the availability and efficiency. In addition, CouchDB is easy to set up and communicate with either HTTP ReSTful API or Fauxton User Interface. Furthermore, the JSON format document used in CouchDB is easy to process in any language. To improve the capacity and availability, our CouchDB is set to a 3-node cluster database that runs in 3 server instances.

Database

There are 3 main databases. And the first one for the searched tweets with only text and sentiment. And the first one for the real time harvested tweets, which contains extra location information. And the AURIN data is also stored in the database to reduce the size of our source code.

MapReduce Views

MapReduce is an algorithm useful for parallel computing, it firstly distributes the computing across machines and then hierarchically summarizes them to get the results. CouchDB has good support for MapReduce by using views, which can generate aggregated data from the database and retrieve them.

We mainly build the views to group for the tweets with different emotions, from different suburbs, and containing different keywords. For example, build views to maintain how many positive tweets in each suburb containing keywords related to housing.

Fault Tolerant

Our database is fault tolerant by using 3-cluster nodes across different instances. So even one server is unexpected shutdown, the database could still run normally.

4.3 Backend

The backend server is built with Flask Framework and follows the ReST architectural style.

Flask

In this project, the major responsibility of the backend is to retrieve data from the database, process and format the data, and respond to the frontend. Due to the low workload required, we choose to use Python for backend, which is simple to use, despite sacrificing a few performances. For the framework, we considered both Flask and Django. Though Django is more mature, scalable, and maintainable, it is more suitable for full stack development with multiple pages and does not naturally support API. Thus, Flask is chosen finally due to its lightweight and good support for the ReSTful API.

ReSTful API

ReSTful API standards for the Application Programming Interface with Representational State Transfer style. It is stateless and uses the URI to locate the data and uses the methods (GET, POST, PUT, etc) to state the action. It is more concise and simpler compared to other styles, like SOAP (Simple Object Access Protocol), by making full use of the protocol. In addition, the JSON content type is used for the ReSTful API, which is lighter and more readable compared to XML.

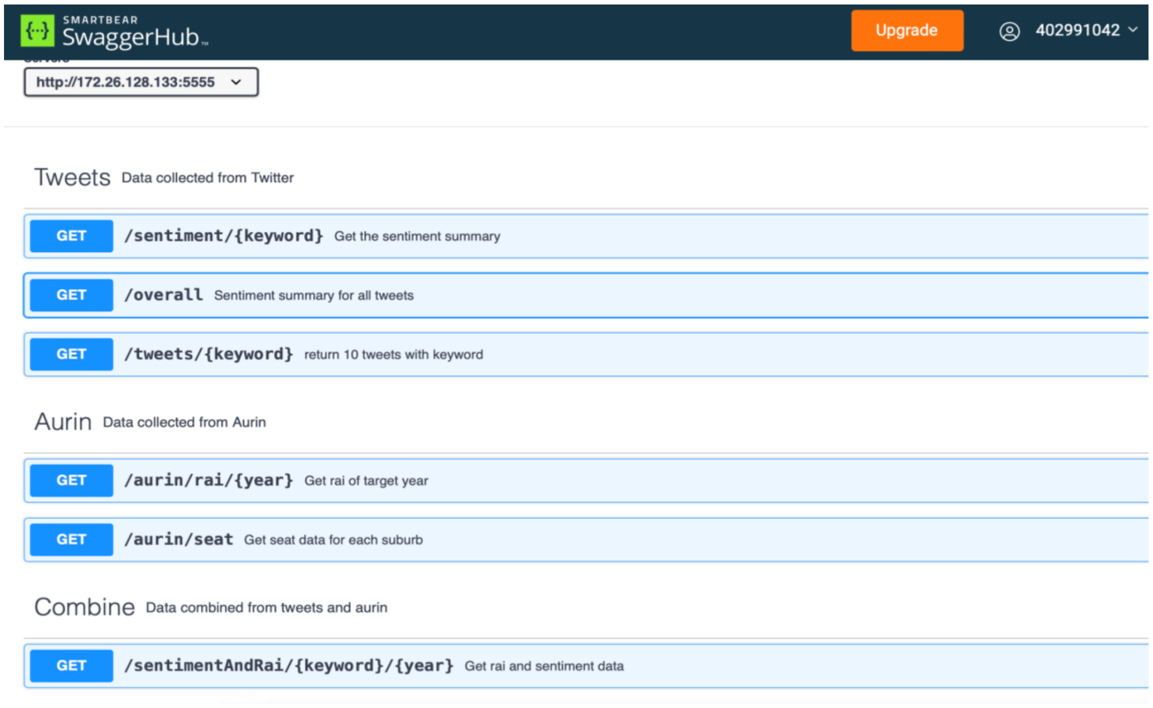
In our program, the *flask_restful* package is used to help with the build of the ReSTful API. There are several endpoints, including：

/sentiment/<keyword>, /AURIN/rai/<db_name>, /sentimentAndRai/<keyword>/<year> and so on.

However, since we only analyze and display the data, only GET methods are implemented.

Swagger

We use Swagger to document our API. Here is a snapshot of all the API calls, the details of request parameters and response format for each endpoint can be found here.



Tweets Data collected from Twitter

GET /sentiment/{keyword} Get the sentiment summary

GET /overall Sentiment summary for all tweets

GET /tweets/{keyword} return 10 tweets with keyword

Aurin Data collected from Aurin

GET /aurin/rai/{year} Get rai of target year

GET /aurin/seat Get seat data for each suburb

Combine Data combined from tweets and aurin

GET /sentimentAndRai/{keyword}/{year} Get rai and sentiment data

Figure 6 Swagger Document Overview

4.4 Frontend

4.4.1 Start frontend application

4.4.1.1 Unimelb user

Via unimelb WiFi or Cisco VPN, the frontend web application can be accessed through the following URL: http://172.26.128.133:3000/.

4.4.1.2 Local user

Use web browser Direct to our GitHub repository, download the code package. Tying the following shell command in your terminal to initialize the frontend application in your localhost web server. (Node.js required)

1. Run cd /file_path_to_github_package/FrontEnd
2. Run npm i
3. Run npm start

Now, the web application will be displayed on your localhost by http://localhost:3000/.

4.4.2 Frontend Component

Front end is implemented by node.js, apply Google Map API and echarts.js library to display the data analysis results, and apply bootstrap.js library for page decoration. Frontend allows users to select relevant parameters to fetch filtered data from backend and search the geographic location for local data analysis.

4.4.2.1 Web Implementation

HTML, page structure technique for visualizing web applications to users. CSS, style information, styling the web. JavaScript, the interface of the backend, provides functions to the web application. Npm helps to build the web application on the local server.

4.4.2.2 Google map Platform

Google map Platform allows users to implement google map functionality to their own web. For this web application, Google map API has been used. The map of the great Melbourne is loaded to the web by Google map API. Also, searching function, allowing users to search an exact location or point out their favorite places. Moreover, loading geographic information such as the bounding box polygons of Melbourne suburb. As such, the Google Maps API supports visualization of tweets data for any suburb in Melbourne. As users choose their desired scenario, a heatmap for each Melbourne suburb will be loaded.

Therefore, users can use their mouse to hover over any sub-region of Melbourne in the map for statistical analysis.

4.4.2.3 echarts.js

Apache echarts is an open-source JavaScript visualizer that runs smoothly on PCs and mobile devices. It is compatible with most modern web browsers such as IE9/10/11, Chrome, Firefox, Safari, etc. echarts relies on the graphics rendering engine ZRender to create intuitive, interactive, and highly customizable charts. The basic chart types supported by echarts include line series, bar series, scatter series, and pie charts for statistics. And it's very easy to combine them with echarts. In this web application, pie charts, hill charts, and bar series are created by echarts.

4.4.2.4 bootstrap.js

Bootstrap is an open-source potent front-end framework used to create modern websites and web apps. Some elements in this web are decorated by the UI interface design of bootstrap.

5 MRC Infrastructure

5.1 Resources

The Melbourne Research Cloud Platform uses OpenStack private cloud deployment paradigm. It offers academics at the University of Melbourne OpenStack infrastructure-as-a-service (IaaS) cloud computing, giving them access to a strong set of on-demand computational capabilities. This service allows researchers to quickly acquire scalable computational resources according to their demands (Sinnott, 2022).

For this project, 4 instances, 8 VCPUs, and an unlimited amount of RAM and 500Gb of volumes are allocated. The total allocated computational resources and the used computational resources are shown below.

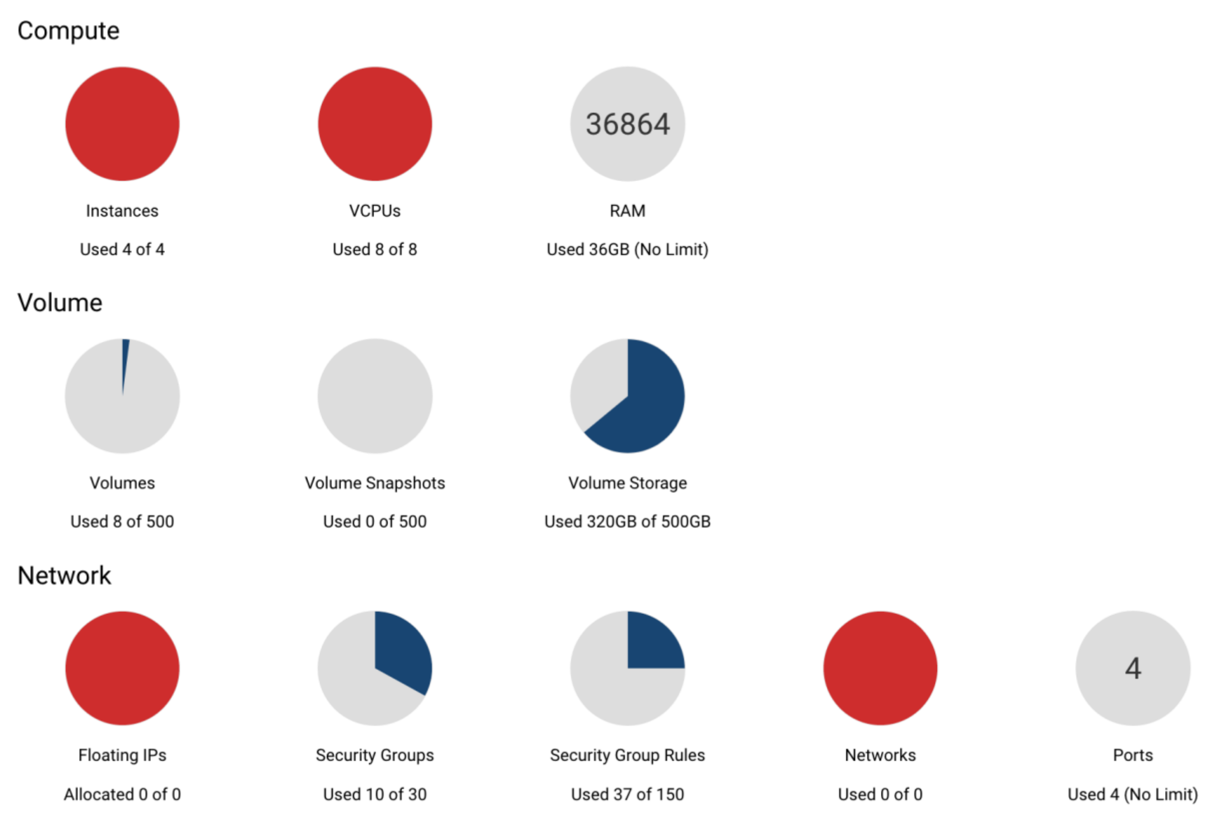


Figure 7 Computational resources overview

5.2 Ansible

Since deploying the complex cloud system requires multiple steps and much duplicate work, such as creating instances, mounting volumes, installing docker, and so on. Some work needs to be done in each virtual machine, while some work requires to be redone every time there is an update. To reduce duplication of effort, document the complete process and reduce manual errors, we chose to use an automation tool to configure and manage our servers.

Ansible is an automation tool released in 2012, and it was chosen due to its simplicity and minimal requirements. Full usage of our Ansible playbooks can be found here.

5.3 Docker

To better manage our multiple servers, Docker is used to containerize our whole application. There are many other tools that have similar functionalities; however, Docker is the most popular and successful one in recent years. It is lightweight but very useful. Firstly, each docker can be seen as an independent machine, so it is naturally simple to build a different environment for different servers. In addition, by using docker image, we could restart our servers with just a one-line script, and even control the version of our servers. Furthermore, with the use of volume mapping, the data inside each container is easy to move or backup.

Meanwhile, to make our deployment easier, Docker Compose is used to orchestrate our multiple containers. With Docker file and docker-compose, all the containers in one virtual machine can be managed at the same time. Though we could simply run all containers with one docker-compose, only the containers with multiplication will be rebuilt and restarted, which is exactly what we want to do.

Docker SWARM is another orchestration tool that could run containers as a cluster of multiple nodes across many machines and can scale the containers as needed. However, since our CouchDB naturally runs as a cluster, and besides this, we only have 3 or 4 containers for the frontend, backend, harvester, and the search engine (not always running). And the workload for these servers is not very high. So we decided to run these servers in a single instance and use docker-compose to manage them.

5.4 Pros and Cons

The MRC platform has many advantages. First, it is free of charge, so scholars can focus on their projects or research without having to consider the resource overhead. Secondly, it provides scalable and on-demand computational resources, so it can greatly improve the efficiency of project DevOps. Finally, MRC allows projects to run on it 24/7, so crawlers can get data much more efficiently when they are deployed on it than when running locally.

However, the MRC platform also has some disadvantages such as it can only be used within the University of Melbourne network, but because of the Covid-19, learning becomes a remote online mode, logging into MRC requires the use of a VPN and therefore may be slow. Also, the project which is deployed on MRC cannot be shown to recruiters when looking for a job. In addition, it is also because MRC is free and the group cannot occupy these resources for a long time, so the project deployed on it may not exist for very long, for example, it may not be available after the subject is over.

6 User guide

6.1 System Deployment

To deploy the whole system from scratch, firstly, the developer needs to install ansible on a local computer, and the version of ansible used in this project is 2.12.4. Secondly, the developer should clone the project's repository on GitHub and cd into the ./Ansible directory. Finally, the developer needs to download the OpenStack RC file of the project on MRC and set the API access password. The deployment of the entire project involves the following six steps.

6.1.1 Deploy instances

This step is performed by running the "bash create_instances.sh" command in the terminal, and the playbook content is executed on the local host.

The main purpose of this step is to create four 2 cores and 9 gigabytes of RAM virtual machines (instances) with ubuntu installed, named Server 1, Server 2, Server 3, and Server 4, and these virtual machines also install some dependencies such as python3, python3-pip, openstack sdk and so on. Meanwhile, eight volumes are created, and each virtual machine is bundled with two volumes, one for the application files and one for the data files. In order to ensure secure access to these virtual machines, some security groups need to be created, such as SSH access security group and HTTP access security group.

The entire content of this step is listed in the following table.

Command: bash create_instances.sh		
Hosts: Localhost		
	roles	name
1	common	- Gather facts of remote host - Install dependencies - Update pip
2	create-volume	- Create volume - Create a list of volume Ids
3	security-group	- Create a security group - Create a list of security group names - Create security group rules
4	instances	- Create an instance - Wait for connection - Add host

Table 1. Deploy instances roles

6.1.2 Mount volumes

This step is performed by running the "bash mount_volumes.sh" command in the terminal, and the playbook content is executed on the 4 virtual machines on MRC.

The main purpose of this step is to mount each virtual machine's “/dev/vdb” and “/dev/vdc” directories to the volumes created in the first step.

The entire content of this step is listed in the following table.

Command: bash mount_volumes.sh		
Hosts: Instances		
	roles	name
1	mount-volumes	- Install dependencies - Make file system - Checking folders - Create directory - Mount device

Table 2. Mount volumes roles

6.1.3 Install docker

This step is performed by running the "bash install-docker.sh" command in the terminal, and the playbook content is executed on the 4 virtual machines on MRC.

The main purpose of this step is to install docker and docker-compose on each virtual machine, in preparation for building containers later.

The entire content of this step is listed in the following table.

Command: bash install-docker.sh		
Hosts: Instances		
	roles	name
1	install-docker	- Uninstall old versions of docker - Install dependencies - Add Docker apt repository key - Add Docker apt repository and update apt cache - Install docker - Install docker-compose

Table 3. Install docker roles

6.1.4 Deploy CouchDB

This step is performed by running the "bash create_couchdb.sh" command in the terminal, and the playbook content is executed on three node servers of CouchDB, server 2, server 3 and server 4, where CouchDB on server 2 is the master node and the playbook that finally creates the cluster runs on the master node server 2.

The main purpose of this step is to create a container running a single node CouchDB on Server 2, Server 3 and Server 4, and then connect the master node CouchDB on Server 2 to the CouchDB on Server 3 and Server 4 to create a 3-node CouchDB cluster.

The entire content of this step is listed in the following table.

Command: create_couchdb.sh		
Hosts: couchdbnodes (couchdb-container) Hosts: couchdbmasternode (couchdb-cluster)		
	roles	name
1	couchdb-container	- Start a couchdb container
2	couchdb-cluster	- couchdb cluster setup one - couchdb cluster setup two - couchdb cluster setup finish

Table 4. Deploy couchDB roles

6.1.5 Clone repository

This step is performed by running the "bash git-clone.sh" command in the terminal, and the playbook content is executed on server 1, which frontend, backend and crawler applications will be deployed on top.

The main purpose of this step is to clone the project repository on Github to server 1 so that the frontend, backend and crawler can be deployed later on the server by running docker-compose.

In addition, if there are updates to the project repository on Github later, it is possible to push the code to server 1 by running "bash git-pull.sh" command in the terminal and then re-running docker-compose to update the corresponding application.

The entire content of this step is listed in the following table.

Command: git-clone.sh Command: git-pull.sh		
Hosts: instance1		
	roles	name
1	git-clone	- git clone
2	git-pull	- git pull

Table 5. Clone repository roles

6.1.6 Deploy web application

This step is performed by running the "bash deploy-webapp.sh" command in the terminal, and the playbook content is executed on server 1.

The main purpose of this step is to create three containers to run the frontend, backend and crawler applications respectively by running the docker-compose.yml file.

The entire content of this step is listed in the following table.

Command: deploy-webapp.sh		
Hosts: instance1		
	roles	name
1	deploy-webapp	- deploy Docker Compose stack

Table 6. Deploy web application roles

6.2 Font-end usage

6.2.1 Tweet analysis

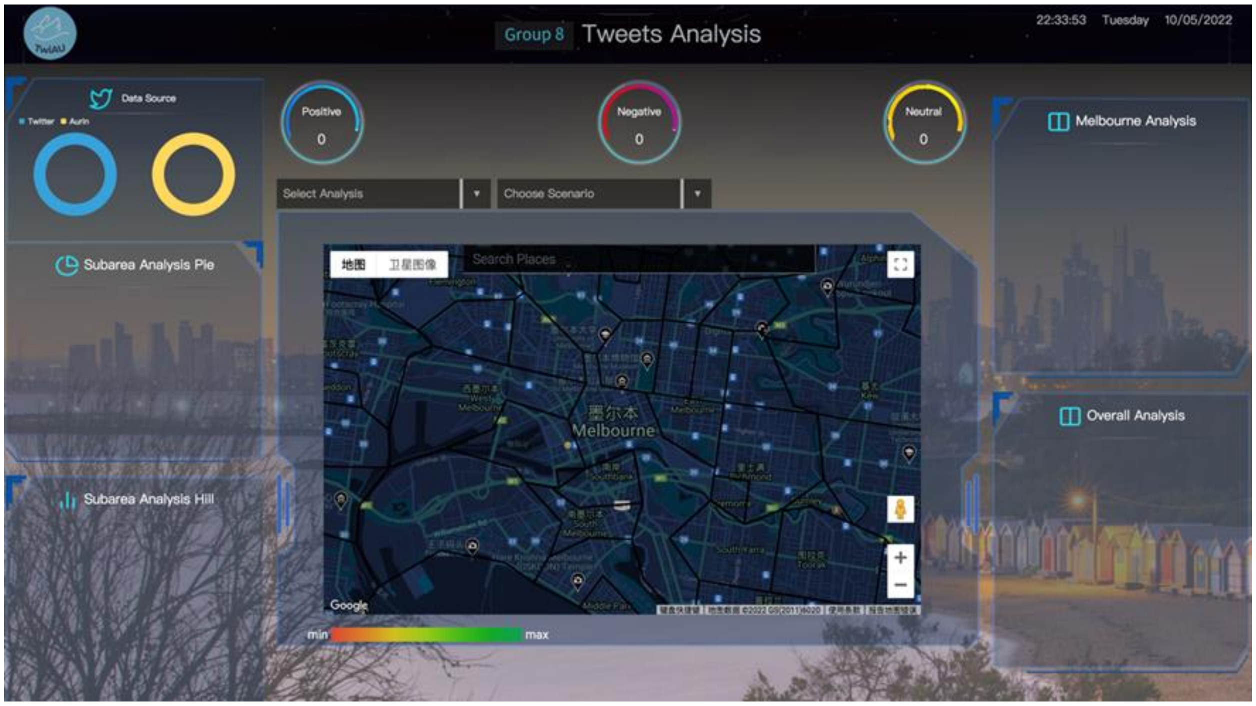


Figure 8 Tweet analysis page

In Figure 8, the main page of the web application is shown. The header at the top of the page displays the web's logo and page's theme. The main page is used to analyse the tweets data.

6.2.1.1 Button and Search Bar

At the top left of the main page, users can **click the hollow circle** to jump to different data analysis pages. For example, the current page is Tweet Analysis, and the user may want to view the analysis of AURIN data by clicking on the yellow circle.

Users are available to select a <u>data analysis</u>.

Click the first dropdown menu in the middlebox. There are two types of analysis of tweet data. One is a heatmap of the tweet counts in the Victoria state sub-region, and the other is a heatmap of the sentiment classification of tweets in the Victoria sub-region.

Users are available to click their interest <u>scenarios</u>.

Click the second dropdown menu in the middlebox. There are seven scenarios that users can choose, which are Housing, Engagement, Health, Neighborhood, Environment, Opportunity, and Transportation. Different scenarios lead to different tweet data. Different analyses result in different heatmaps and their corresponding evaluation matrices.

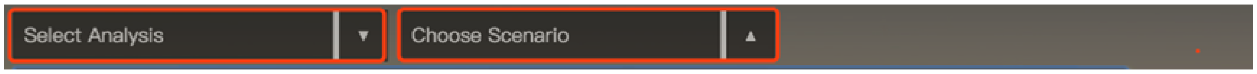


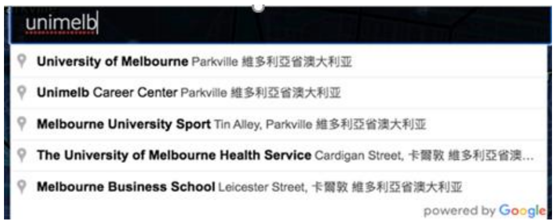
Figure 9 Dropdown menu

Additionally, users can find their location of interest on the map in the middlebox.

1. **Enter location information or keywords into the search bar**, and the map will generate several corresponding locations.
2. **Click on one of the suggested options** to navigate to a specific geolocation.

For example, in figure 10, users enter "unimelb" to the search bar, and the search bar returns five geographic locations about "unimelb", and the user can select one of them.

After the user sets up the two buttons, data visualizations are generated.


unimelb

University of Melbourne Parkville 維多利亞省澳大利亚
Unimelb Career Center Parkville 維多利亞省澳大利亚
Melbourne University Sport Tin Alley, Parkville 維多利亞省澳大利亚
The University of Melbourne Health Service Cardigan Street, 卡爾敦 維多利亞省澳...
Melbourne Business School Leicester Street, 卡爾敦 維多利亞省澳大利亚

powered by Google

Figure 10 Geolocation search bar

6.2.1.2 Heatmap and Charts in tweets count

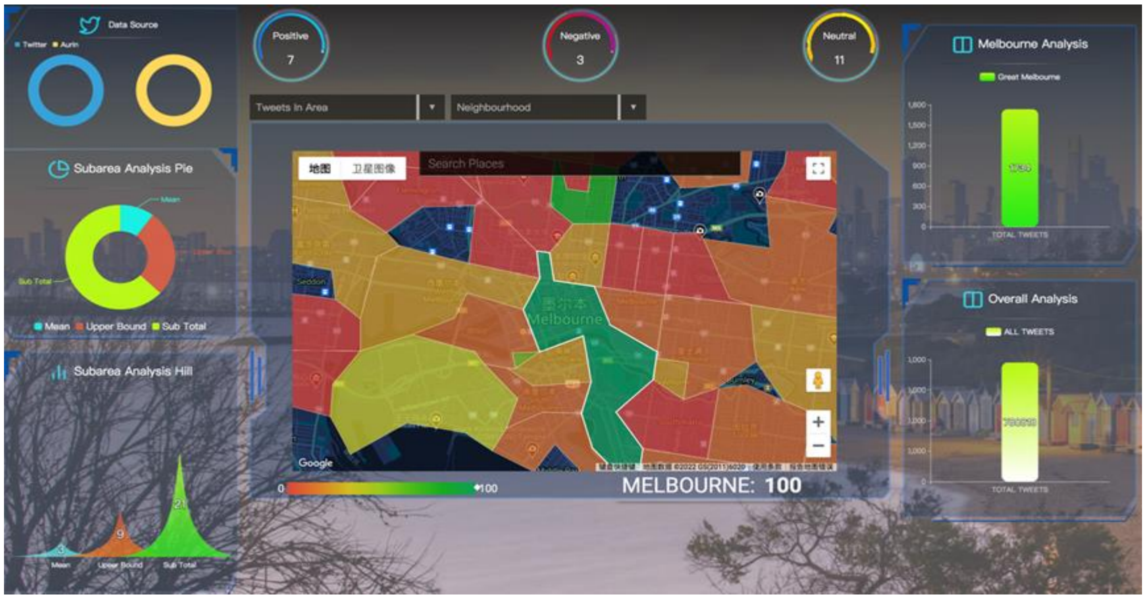


Figure 11 Heatmap of tweets count

In Figure 11, Analysis of tweet count and neighbourhood scenario is selected.

6.2.1.2.1 Pie chart

The pie chart in the left middle will be a**utomatically formed when users select a region in the heatmap**.

It shows the numerical proportions of three elements, the average number of tweets, the upper bound of tweets received in regions, and the number of tweets in the current region. This relationship is also reflected on the map. As shown in figure 11, the proportion of the Melbourne area exceeds the proportion of the upper bound, the value presented in the heat map is 100, and the area's colour is green.

6.2.1.2.2 Hill chart

The hill chart at the left bottom will be **automatically formed when users select a region in the heatmap**.

It compares the number of three elements like the pie chart. The pie chart illustrates the proportion, and the hill chart demonstrates the number. The exact number is shown in the graph.

6.2.1.2.3 Heat Map

The heatmap in the middlebox presents how good a region is in the entire Victoria state.

Users **select different regions using their mouse to hover a region in the heatmap**.

Evaluation Metrics shown at the right bottom of the map are accessed via the formula,

$$\frac{current\ -\ min}{upper\ bound\ -\ min}$$

current -- Tweets received in the current region

min -- Tweets received in the region with the lowest number of tweets

upper bound -- the IQR is the Q3 of Tweets received in a region – Q3 of Tweets received in a region. Values outside this range in statistics are called extreme values. The reason for not using real max value is that real max greatly exceeds other values, making most regions look terrible.

The evaluation matrix of tweet counts is a min-max scaler. So, a number between 0-1 is received, which is suitable for styling the heatmap. The higher the ratio, the more people (more tweets) live in the area.

As figure 11 shows, the number of tweets about housing in the Melbourne region is large, therefore, the color of the region is green. However, its neighbors do not produce many tweets about housing, and their colors are red or brown.

6.2.1.2.4 Melbourne Analysis

The bar chart at the right top will be **automatically formed when users select a scenario**. It counts the number of tweets received in the great Melbourne with location information.

6.2.1.2.5 Overall Analysis

The bar chart at the right top will be **automatically formed when users select this analysis**. It counts the number of tweets received in the whole world.

6.2.1.3 Heatmap and Charts in sentiment analysis

In Figure 12, an Analysis of sentiment classification and the neighborhood scenario are selected.

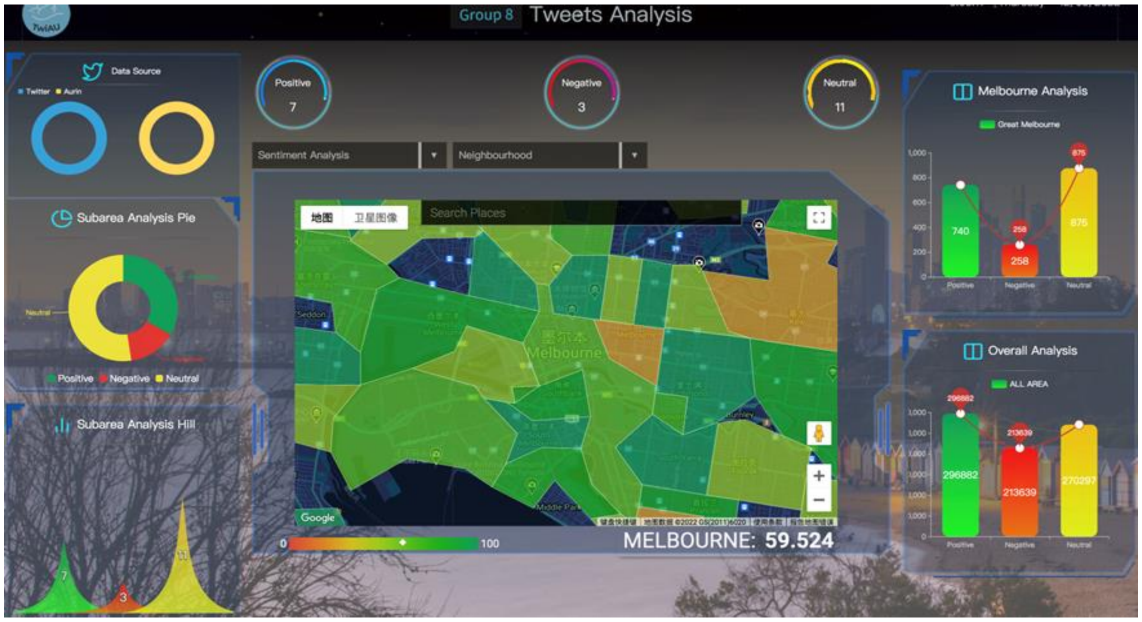


Figure 12 Heatmap of sentiment classification

6.2.1.3.1 Pie chart

The pie chart in the sentiment analysis is similar to the one in the tweets count.

6.2.1.3.2 Hill chart

Also, the Hill chart in the sentiment analysis is similar to the one in the tweets count.

6.2.1.3.3 Heat Map

Evaluation Metrics shown at the right bottom of the map is accessed via the following formula

$$positive\ rate = \frac{positive + 0.5 \times neutral}{total}$$

positive -- Tweets received in the current region are classified as positive sentiment

neutral -- Tweets received in the current region are classified as positive sentiment

total – Number of Tweets received in the current region

The output number is between 0 – 1, which is used to assess whether people in an area have positive emotions. The higher the ratio, the higher the positive sentiment among people in the region.

As shown in the figure 12, the Melbourne region has three negative tweets, seven positive tweets, and eleven neutral tweets. Therefore, by the evaluation metrics, the output value is about 60. The color shown in the graph is chartreuse (yellow green).

6.2.1.3.4 Melbourne Analysis

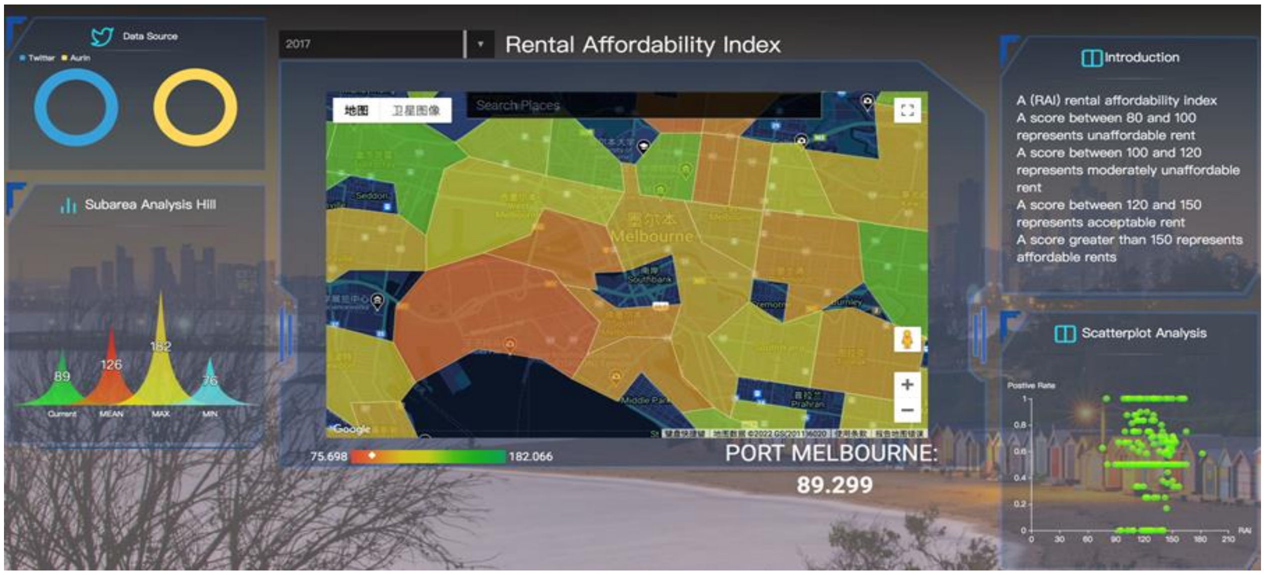
Melbourne Analysis counts the different sentiments received in the great Melbourne with exact geolocation information.

6.2.1.3.5 Overall Analysis

Overall Analysis counts the different sentiments received in the whole world.

6.2.2 AURIN analysis

In Figure 13, the AURIN page of the web application is shown. The AURIN page analyses RAI (Rental Affordability Index) data from AURIN and compares it to housing scenarios of the tweet data.



A (RAI) rental affordability index
A score between 80 and 100 represents unaffordable rent
A score between 100 and 120 represents moderately unaffordable rent
A score between 120 and 150 represents acceptable rent
A score greater than 150 represents affordable rents

Figure 13 AURIN analysis page

6.2.2.1 Button and Search Bar

Click the dropdown menu in the middlebox. There are five years that can be chosen from 2017 to 2021.

Usage of the search bar is the same as on the main page.

6.2.2.2 Heatmap and Charts

6.2.2.2.1 Hill chart

The hill chart in the AURIN page will be **automatically formed when users select a region in the heatmap**. It compares four numbers: RAI of the current region, lowest RAI of the great Melbourne, highest RAI of the great Melbourne, and mean value of RAI.

6.2.2.2.2 Heatmap

Heatmap at the middlebox, RAI in selected areas compared to other sub-regions in Greater Melbourne.

For example, the min value of RAI in 2017 is 75.7, which is represented as red color. And the max value of RAI in 2017 is 182.1, which is represented as green color. Port Melbourne had an RAI of 89.3 in 2017, which is closer to the min value shown in reddish-brown.

6.2.2.2.3 Introduction

A brief introduction to RAI is at the right top.

6.2.2.2.4 Scatter Plot

A scatter plot shows how the tweet data (housing scenario) compares with RAI. The X-axis is the RAI. Y-axis is the positive rate.

7 Project Management

7.1 Role of Members

Jiahui Luo	Backend & Database & MRC
Qichang Li	Crawler & Data analysis & Frontend
Shengze Zhu	Crawler & Data analysis
Yebin Ge	Backend & Database & MRC
Zihao Guo	Frontend & Crawler

7.2 Problem Encountered and Solution

7.2.1 Apply for Twitter API

The first problem we encountered in this project is applying for Twitter API. Since Twitter just updated their API this year and after that only Academic or Elevated accounts could retrieval location information of tweets, which is what we need. However, they are hard to apply, with much more information needed, like the academic profile, which we do not have yet. To solve this problem, we let everyone in our team to apply for the developer accounts using all accounts. Some of team members have been rejected at the first step. Fortunately, two of our team members successfully applied for Academic or Elevated accounts.

7.2.2 CouchDB hard to setting up

During the setting of the CouchDB cluster across different instances, since the documentation of CouchDB are not very detailed, and there are very few relevant resources online like tutorial. We encounter several issues and cost two members two days to solve them finally.

7.2.3 Lack of tweets with location information

In Twitter API, the tweets include location information only if the users choose to, and most users do not do so. And we want to group our data based on suburbs, which need coordinate information. In addition, the location information could only be retraveled using streamer. As a results, it’s very hard to get enough data. To solve this, we run our harvester as early as possible. However, tough the harvester has been run for nearly 3 weeks, the data we have currently are still not enough, some suburbs have only few tweets.

7.3 Team Reflection

In total, our team did a great work in this project. A good aspect worth mentioning is the distribution of work in our group. We decide the role of each member at the start and followed it throughout the entire project. Every member only needs to focus on own parts, which reduced the workload and ensured the efficiency. In addition, each role has assigned to two members, which ensure that they can discuss and help each other, and one of them would be the head of this role. And every member has been assigned to two or three roles, with one of them focused. By doing so, we have a good collaboration during the project.

However, there are shortcomings. The most important point is the lack of detailed documentation, like requirements, user stories, etc. The work we needed to do was decided more through discussions in meetings, but without writing them down. This caused some misunderstandings and made some issues not considered in time.

8 Conclusion

In conclusion, we have collected the data from Twitter API and integrated it with RAI data from AURIN to create this visualisation website to explore Melbourne's livability. We analysed the content of tweets for sentiment and grouped them by suburb and keywords. The results showed some correlation between the emotion contained in housing-related tweets and the RAI data. In addition, the visualized data also indicates that tweets from the Great Melbourne area had a more positive sentiment than the entire Victoria state, which may reflect its livability.

Appendix

YouTube Link

Github Link

References

Ansible documentation. (2022, May 05). Retrieved May 14, 2022, from https://docs.ansible.com/ansible/latest/index.html

Djc. (n.d.). *DJC/couchdb-python: Python library for working with couchdb*. GitHub. Retrieved May 14, 2022, from https://github.com/djc/couchdb-python/

Docker documentation. Docker Documentation. (2022, May 12). Retrieved May 14, 2022, from https://docs.docker.com/

Features. Features - Apache ECharts. (n.d.). Retrieved May 14, 2022, from https://echarts.apache.org/en/feature.html

Flask-restful.readthedocs.io. 2022. Flask-RESTful — Flask-RESTful 0.3.8 documentation. Retrieved May 14, 2022, from https://flask-restful.readthedocs.io/en/latest/

Mark Otto, J. T. (n.d.). *Bootstrap*. Bootstrap · The most popular HTML, CSS, and JS library in the world. Retrieved May 14, 2022, from https://getbootstrap.com/

Sinnott, R. (2022). Cluster and Cloud Computing COMP90024 Week 5 Lecture Semester 1-2022[PowerPoint slides].

Tweepy. (n.d.). Tweepy Project Website. Retrieved May 14, 2022, from http://www.tweepy.org/