

An Investigation on the livability of Melbourne

COMP90024 - Team2

Claire Zhang, Yi Yang, Hengzhi Qiu
Yifeng Pan, Yonghao Hu

COMP90024 Cloud and Cluster Computing

The University of Melbourne

May 17, 2022

Abstract

This report presents the system and application architecture that was implemented for assignment 2, COMP90024 Cluster and Cloud Computing at the University of Melbourne. This is an investigation conducted on Twitter social media platform about the livability of Melbourne. The scenario that was chosen for this assignment is related to the topic of language diversity, health and income. We attempt to link these subjects with user activity on Twitter and learn about the city's true opinion on the Opportunity and Health component of livability. Here we discuss each component of the architecture, as well as how team members collaborated during development.

Public access to the source codes for this assignment is available. Please see:

<https://github.com/COMP90024-G2/COMP90024-project2/tree/main>

student ID: 1080915 - wazhang1@student.unimelb.edu.au

student ID: 1253748 - hengzhiq@student.unimelb.edu.au

student ID: 1074365 - yyyang4@student.unimelb.edu.au

student ID: 1049814 - yonghaoh1@student.unimelb.edu.au

student ID: 955797 - yifengp@student.unimelb.edu.au

Content Table

Abstract	1
1 Introduction	3
1.1 Scenario Overview	3
1.2 Report Structure	3
2 System architecture	4
2.1 General Design	5
2.2 System Instance Deployment	6
3 Cloud Infrastructure	6
3.1 MRC	7
3.1.1 Resource Allocation	7
3.1.2 MRC - Pros and Cons	8
3.2 Ansible Cloud Orchestration	8
3.3 Docker	9
3.3.1 System Deployment	9
3.3.2 Docker - Pros and cons	10
4 Data Collection and Storage	11
4.1 Twitter API Harvesting	11
4.1.1 Streamer	12
4.1.2 Searcher	13
4.2 Aurin Data Mining	13
4.2.1 Aurin Data Searching	13
4.2.2 Aurin Data Preprocessing	14
4.3 CouchDB Storage	14
4.3.1 CouchDB Set-up	14
4.3.2 Database Overview	14
4.3.3 CouchDB - Pros and Cons	15
5 Data Analysis and Visualization	16
5.1 Change in Melbourne's general livability views	16
5.2 Livability and national identity	17
5.3 Public perception on opportunity and health indicators	18
5.4 Geo-spatial difference on the livability indicators	19
6 Web Development	20
6.1 Backend - Flask	21
6.1.2 Flask - Pros and Cons	21
6.2 Frontend	21
7 Interface Guide	22
8 Conclusion	24
8.1 General Conclusion	24
8.2 Issues and challenges	24
8.2.1 Twitter API	24
8.2.2 MRC	24
8.2.3 Database	25
8.2.4 Aurin	25
8.2.5 Analysis	25
8.2.6 Scalability	26
9 Team Collaboration	27

1 Introduction

Melbourne, Australia's second-largest city, is known as Australia's cultural capital for its elegant natural environment and a high degree of urban greenery. Anyone who lives in Melbourne knows that Melbourne has won the title of 'the most livable city in the world many times. Since 2002, the Economist Intelligence Unit has ranked Melbourne in the top three most livable cities in the world. Online social media networks like Twitter have grown in importance over the last decade, and there has been a lot of discussion and information sharing in such networks, making them a good reflection of the opinions of citizens.

1.1 Scenario Overview

The objective scenario is to investigate the livability of Melbourne based on a set of indicators that includes housing, neighborhood, transportation, environment, health, engagement, and opportunity. We have chosen to study the indicators of **opportunity** and **health** by focusing on the scenarios of **diversity, health and opportunity**. In particular, we have chosen to:

- analyze whether there is diverse language usage in social media, which is regarded as an **opportunity** indicator;
- analyze the attitudes associated with general health and career planning topics (positive, negative, neutral etc.), which is directly related to **opportunity and health** indicators.

1.2 Report Structure

By comparing Melbourne citizens' attitudes towards these topics, we hope to preliminarily explore how satisfied people are with life in Melbourne. In [section 2](#) we will discuss the system that has been developed to achieve this.

The general system architecture consists of various components and is deployed on the Melbourne Research Cloud (MRC). The MRC infrastructure will be further explained in [section 3](#). Using Ansible scripting and Docker techniques, the MRC is highly scalable and can be fully orchestrated.

For the analysis, we have harvested a large amount of tweets that are extracted from Twitter. We filtered the tweets with regards to the selected keywords and location so that they all say something about the topics and are tweeted in Melbourne. We will explain the detailed implementation of the Twitter harvester in [section 4](#). The application makes use of Twitter's Search and Stream API and stores the extracted tweets in CouchDB. Apache CouchDB is a document-oriented database management system. It provides a REST interface to operate on JSON as a data format.

Most importantly, we performed a series of processes on the data we collected, including counting tweets in different languages, and Natural Language Processing(NLP) to infer emotional information about tweet content. The processing steps are presented in [section 5](#), along with the visualization method. To learn more about the data authenticity, we compare the results with the statistical data from the Australian Urban Research Infrastructure Network (AURIN). The AURIN organization provides more than 5,500 datasets associated with urban and regional research, planning and infrastructure. We find some of them relevant to the subject of our investigation.

To present the visualized results to users, we developed an HTML website and deployed it on MRC. The website is a basic Flask application that renders visualizations of three topics. The discussion of the front and back ends takes place in [section 6](#).

At the end of this paper, a user guide on deployments will be provided in [section 7](#), which covers deployments for all the components (harvesters, database etc.) specifically. In [section 8](#) we will conclude our major discoveries as well as the issues and challenges we encountered during development. Lastly, in section 9 we explain how we work together as a team.

2 System architecture

System Workflow Architecture Reference

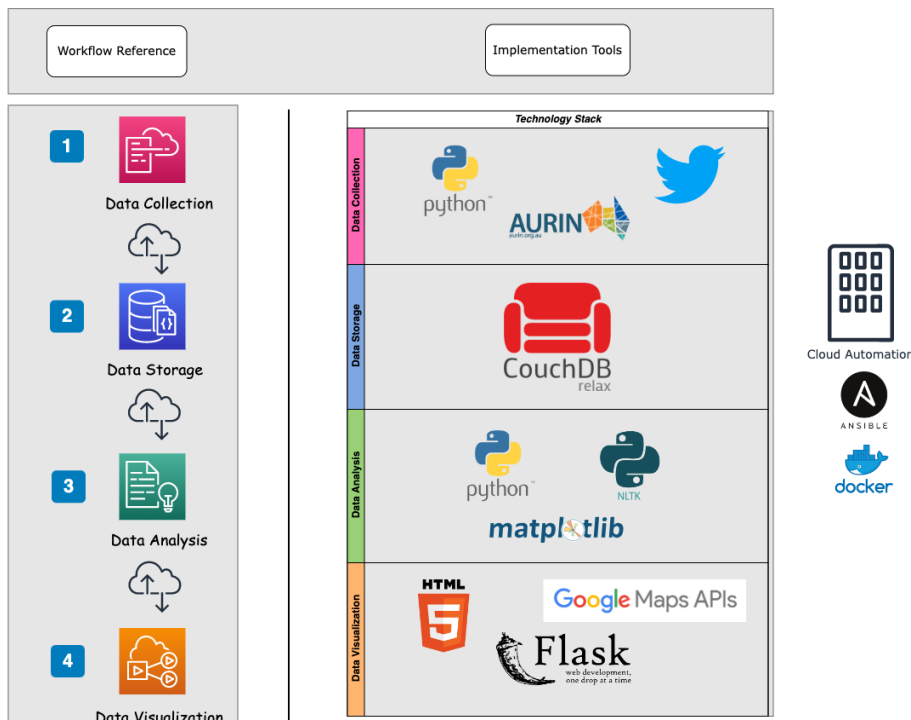


Figure 1: System Architecture

In this section we will go through the system we designed for this assignment, which consists of a 4-step workflow:

1. Data Collection
2. Data Storage
3. Data Analytics
4. Data Visualization

For every architectural layer, there are respective technology stacks (techniques and programming frameworks) chosen. The elements are shown in figure 1 and are further described throughout upcoming sections.

2.1 General Design

The core of this system design is data flow control. In our system, CouchDB is the foundation for Data Collection, Analytics, and Visualization applications. First of all, we start the Twitter Harvester with pre-defined parameters(run-time, keywords etc.), and the captured tweets go straight into the database. We have pre-built three corresponding databases for three scenarios as well as for AURIN data. The data processing component extracts tweets from the raw data storage, processes them and stores the resulting information in a new database ‘output’. Finally, the front-end website retrieves the ‘output’ data and visualizes it to the user.

Data Flow Control

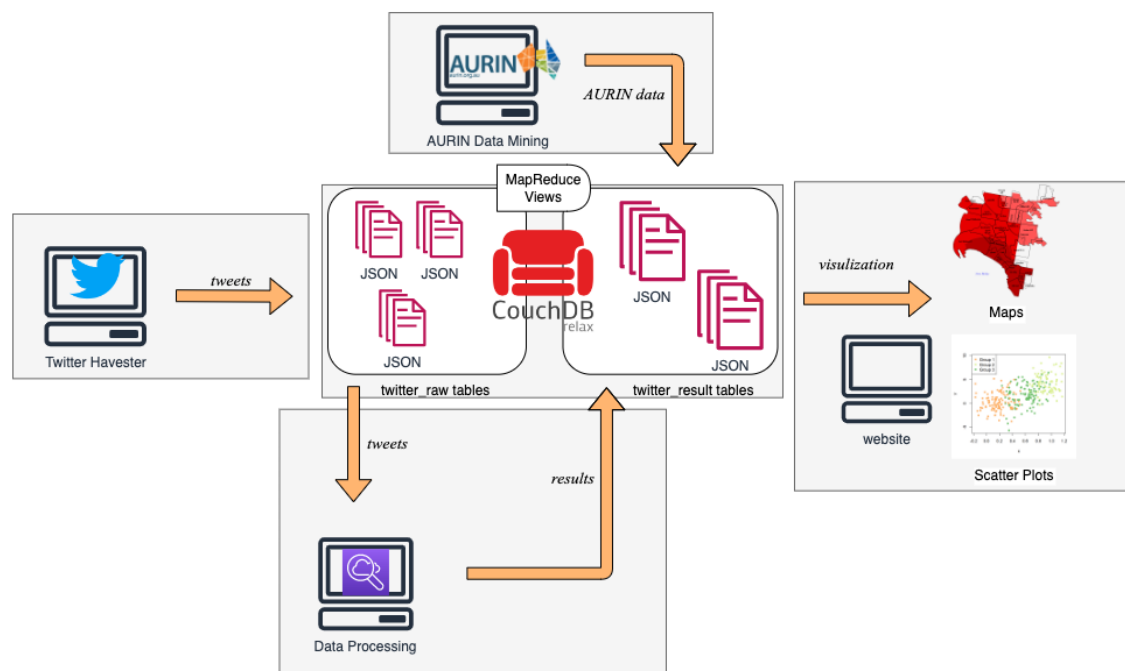


Figure 2: Data Flow Control

2.2 System Instance Deployment

The entire system is deployed on a total of four computing instances on the Melbourne Research Cloud. Three of the four instances act as the ‘Data Node’, including Master-node, subnode-1, and subnode-2 (master here just for simple and clear deployment when using Ansible to create CouchDB cluster, actually three nodes in the cluster have the same status), we deploy CouchDB on these nodes in a Docker container environment and bind them together as a CouchDB cluster. Depending on the cluster, data replication and load balance can be performed automatically. As for these data nodes, each one runs an instance of the Twitter Harvester (including tweet searcher and tweet streamer) and the master data node runs an instance of the Data analyzer. Besides, the remaining instance is designated as the ‘Main Server Node’ and we deploy the Flask-based web application in a Docker container environment on this node.

In the Cloud Infrastructure section, we will discuss it all in detail.

3 Cloud Infrastructure

The system architecture and the corresponding technology stacks are shown in figure 3.

The whole process of setting up the computing instances and the deployment of the system is based on the Melbourne Research Cloud. Besides, Ansible is used to facilitate the setup of server instances, dependencies installation and building docker images to simplify the deployment of system components.

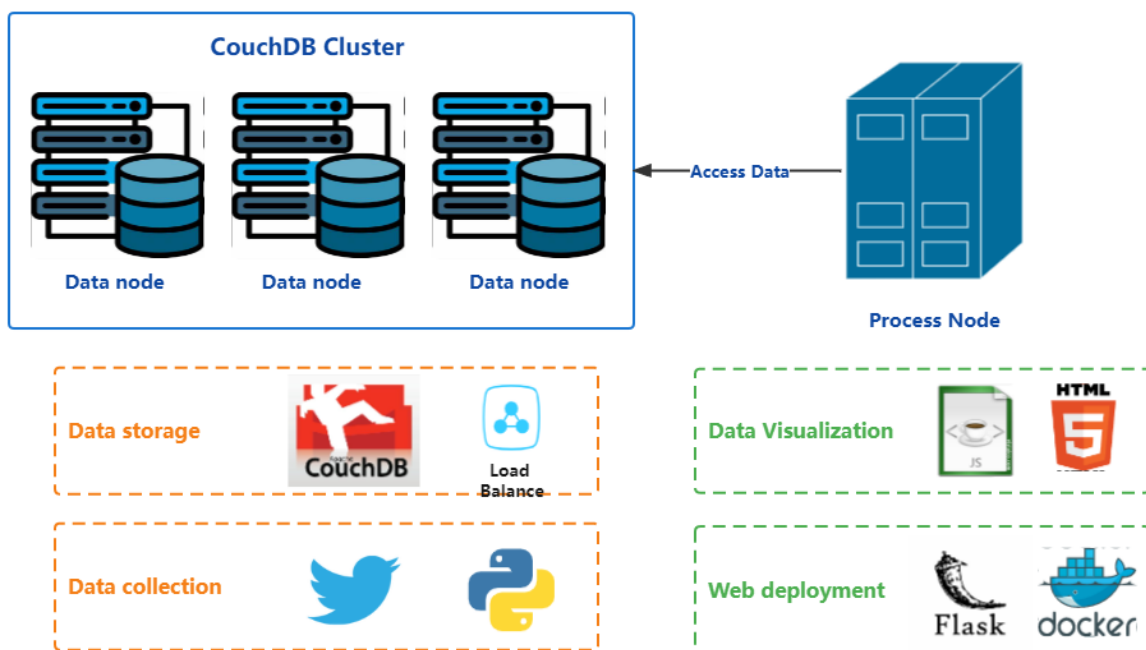


Figure 3: Architecture and technology stack

3.1 MRC

MRC is a private cloud developed and maintained by the University of Melbourne. It contains nearly 20,000 virtual cores and is suitable for a range of instances of different sizes. In addition, since the cloud infrastructure itself is built on The OpenStack Foundation, it allows the use of existing libraries to access OpenStack application programming interfaces (APIs). Good scalability brings convenience to cloud application deployment. No doubt that MRC is powerful but there are still some shortcomings.

3.1.1 Resource Allocation

In this project, four instances are set up in MRC, including one process node and three data nodes. Each instance is allocated 2 cores and 9 GB memory (uom.mse.2c9g) and sets up based on image NeCTAR Ubuntu 20.04 LTS (Focal) amd64.

As for volumes, two volumes with a combined capacity of 50 GB are created and attached to each data node. One of them (40 GB) acts as a data volume, which stores raw tweets or results after tweets processing. The other one (10 GB) acts as a docker volume, which works for docker itself and docker images.

3.1.2 MRC - Pros and Cons

According to the user experience on the MRC, we summarize some pros and cons which are listed below:

Pros

- 1) High availability: The cloud provides a simple and easy-to-use UI interface. Users do not need to master the underlying principles, and do not need to be proficient in operation and maintenance deployment. What they need to do is to click and select resources, and MRC will do all the things for them to complete the construction of cloud instances.
- 2) Security: The MRC can only be accessed through the campus network, so only on-campus personnel (including off-campus students who connect to the campus network through the campus network proxy) can access and use the resources of the MRC. The establishment of firewalls can effectively resist attacks from the Internet, which greatly improves the security of MRC.
- 3) Portability: All projects successfully deployed on MRC can be ported to any OpenStack-based cloud server because the interface standard is the same.

Cons

- 1) Access restrictions: As mentioned above, the establishment of firewalls brings security to MRC. But on the other hand, if this cloud infrastructure is only available for University networks, it can hardly expand the number of potential users.
- 2) Slow maintenance: Unlike large commercial public clouds, MRC does not have a dedicated cloud service provider to respond quickly. Therefore, it is difficult to provide timely fixes when there is a problem in the cloud.

3.2 Ansible Cloud Orchestration

Ansible is an open-source IT automation tool that automates provisioning, configuration management, application deployment, orchestration, and many other manual IT processes. The most powerful part of Ansible is that users can automate deployment by running written Ansible playbooks, including development environment configuration and cloud infrastructure construction.

Our project deployment consists of several parts, including dependency installation, OpenStack instances and volumes setup, docker and CouchDB setup and our system deployment. The role list in the playbook is shown in figure 4.

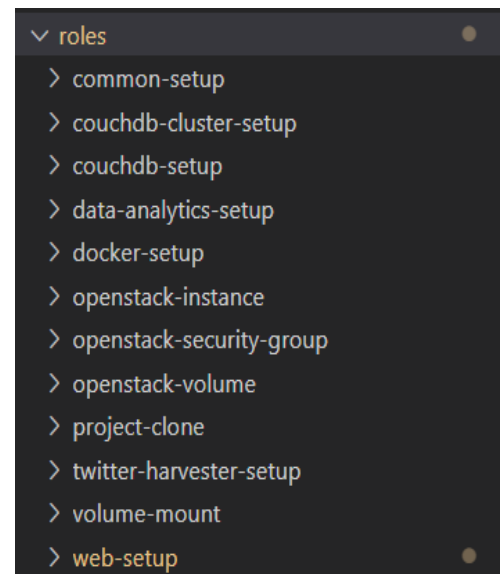


Figure 4: Architecture and technology stack

And the roles are executed in the following order:

1. Cloud resources provision, including instances, volumes and security groups
2. Mount volumes to instances
3. Install dependencies and docker
4. Build CouchDB images and run docker instances on data nodes
5. Construct CouchDB cluster
6. Clone the whole project to the Master Data Node
7. Deploy tweet searcher and streamer on the Master Data Node in the docker environment
8. Download the big tweet_melb file in the Data analytics directory and deploy the data analyzer in the Master Data Node (without docker)
9. Clone the whole project to the Process Node
10. Deploy web application for visualization on the Process Node

Something worth mentioning is that we use the built-in package `add_host` to store the address of all the instances in the memory after creating all the instances we need. Therefore, we can get rid of the extra `host.ini` file which is used to save instance addresses for subsequent execution.

The most intuitive feeling of using Ansible for deployment is to get rid of the cumbersome interactive configuration. Using the written playbook only needs to input an extra line of code to complete the deployment work repeatedly and accurately. Of course, there is no free lunch in the world, Ansible has a certain learning curve, and it takes a lot of time to learn to write efficient and concise deployment code.

3.3 Docker

Docker is a software/container platform that uses virtualization technology (cgroups, namespaces) to achieve resource isolation and limitation of the operating system. For developers, container technology provides a sandbox environment for application deployment. We can run and manage application processes in independent containers. The abstraction layer provided by Docker enables developers to maintain a relatively consistent development environment, avoiding conflict.

3.3.1 System Deployment

In this project, we deploy all the system components in the docker environment except for the data analyzer. To run the system components as a docker container, an extra Dockerfile is needed. Because building a working image in the traditional way, we may need to use a base image, install packages in the image, write configuration files in the image, write code in the image, etc. But using Dockerfile can record all the operations in the image in the form of multiple command lines. If we need to change the operations in the image, we only need to modify the Dockerfile directly, which greatly improves the sharing and maintainability of the image. Besides, since we use Dockerfile to install dependencies like python packages, we need a file to store all the package names and their version numbers, which is named `requirement.txt`.

And here we take tweet-harvester deployment as an example to explain the whole process of using docker to deploy system components.

Tweet harvester (searcher and streamer):

- 1) Prepare Dockerfile and `requirement.txt`, saved in the working directory
- 2) Clone the whole project to the Master Data Node
- 3) Target the tweet-harvester directory and build a docker image using the given Dockerfile.
The whole process is controlled by an ansible-playbook.
- 4) Stop all the running tweet-harvest containers (if any)
- 5) Create and start a new tweet-harvest container

However, compared to purely running the *main.py* in tweet harvester, the deployment of our Flask-based web application is a little bit tricky. Since the Flask application is a Python application that conforms to the WSGI specification, it cannot be run independently (a method similar to `app.run` is only suitable for development mode), and it needs to rely on other components to provide server functions. Therefore, we need a library called Gunicorn to provide the server functionality and use library `gevent` to support asynchronous processing of requests and improve throughput. Besides, an extra configuration file is needed to Set server access parameters, including concurrent access volume, port range, etc. And the start command is shown in figure 5.

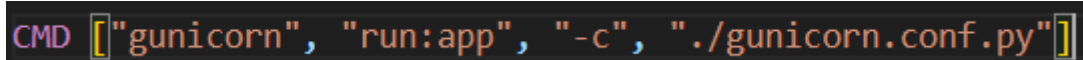
A terminal window showing a command to run a flask-based application. The command is: `CMD ["gunicorn", "run:app", "-c", "./gunicorn.conf.py"]`. The text is displayed in a monospaced font with a dark background and light-colored text.

Figure 5: run flask-based application

3.3.2 Docker - Pros and cons

Undoubtedly, docker is a powerful tool for deployment. It takes a long time to set up a development environment or an application production environment, and a small issue may take a long time to solve. With docker, these become very easy, and the development environment is just the address of one or several Docker container images. Developers only need to write an execution script that controls the deployment process, download the packaged image and start the container, and the environment setup is complete. In addition, using docker makes it simple for developers to quickly roll back the system to the history version because each docker image is a natural historical version.

However, docker is not perfect, and we think the main disadvantages of docker are as follows:

- 1) Isolation is not as good as hypervisor-based virtualization technology. The hypervisor-based virtualization is completely virtualized on system hardware resources. When a system-level problem occurs in a virtual machine, it usually does not spread to other virtual machines on the same host. But docker containers share the same operating system kernel and other components. Therefore, when a docker container is attacked, it tends to implicate other containers through the underlying operating system.
- 2) Docker is not suitable for persistent storage scenarios. In terms of data storage, the solution provided by the Docker container is to use the volume interface (storage volume) to form data mapping and transfer to achieve the purpose of data persistence. However, this will also result in a waste of some resources and more interactions. Whether it is mapped to the host or the network disk, it is not an efficient solution.

In summary, docker is not suitable for some scenarios like data storage. But in this project, docker brings great convenience to our application deployment. More importantly, for flask applications, the

use of docker deployment avoids the inconvenience of terminating the system when terminating the running command. As long as the project container is started, our deployed application can continue to run on MRC.

4 Data Collection and Storage

This section explains how data collection and storage are implemented. Section 4.1 describes the Twitter harvester setup while section 4.2 describes the way of mining AURIN data. Section 4.3 further introduces the CouchDB which we use to store both raw data and processed data.

4.1 Twitter API Harvesting

The Twitter harvester is developed in Python language. The Twitter API supports programmatic access to Twitter in a fast and advanced way. The interface digs into the core elements of Twitter including Tweets, Direct Messages, Spaces, Lists, users and more. In order to access this data, we would need a developer account. Tweepy is a specialized module for working with the Twitter API in Python. It is simple to use, eliminating the need to manually develop a Twitter crawler. The key to capturing Twitter data is that Twitter requires all requests to be OAuth authenticated, and Tweepy provides a dedicated Auth function that makes authentication easy.

In this project, we implemented two main methods to harvest tweets with regards to the topic we were concerned about, which exploit the Streaming API and Searching API respectively. Since this project aims to focus on the two specific domains of opportunity and health, both APIs have used keyword retrieval to extract tweets relevant to the interested domains.

The key words are generated by expanding the given root words list to increase the harvester's coverage, however the initial set of root words are manually chosen. This might be a potential downside since this approach would overlook many relevant and potentially indicative information. The alternative NLP topic analysis approach might yield a better result in classifying and capturing the tweets' underlying topics, however was not able to be implemented in the application due to time limitations. Nevertheless, the current method comprises the data coverage to trade for easy implementation and provides a fairly good outcome. The keywords are expanded by the wordnet feature from the Python NLTK package, which could return the synonyms and antonyms of a given word. Hence, in this way the Harvester is able to extract relevant tweets with unspecified sentiment for further analysis.

4.1.1 Streamer

The Twitter Streaming API allows us to live stream public tweets from the platform so we can store them and display basic metrics about them. Unlike the REST API, the streaming API pushes messages to persistent sessions instead of pulling data from Twitter. By using the streaming API, more data can be downloaded in real-time than would be possible with the REST API. There are three steps to stream tweets:

- Create a Stream
A data source (e.g., collection, array) retrieves a stream. We created a *tweepy.Stream* object with custom settings(e.g. search only Melbourne-based tweets), keywords etc.
- Intermediate Operation
An intermediate chain of operations that processes and stores tweets from the data source e.g. filtering out the geocode/text information from a tweet.
- Termination Operation
A termination operation that performs an intermediate chain of operations and produces a result. In our implementation we terminate the data stream with a pre-defined run time, i.e. after some running time, the data stream will terminate automatically.

Streamer's performance is outstanding, capturing more than 10 Melbourne-based tweets per second with a specified keyword. However, with such strong performance, we need to be careful about stream timing so that we don't quickly exhaust the maximum search volume. There's another important benefit to streaming. Because we are streaming in a unique time frame every time, that is, the streamer will only work for some time after we started it. Each captured tweet will not appear twice unless we start two streamers at the same time.

During practical streaming, we have noticed a performance drop-down when searching with null keywords. This might be due to the streaming algorithm in the API. To mitigate this we decided to implement a Searcher using Twitter's Search API.

4.1.2 Searcher

Unlike Stream API, Twitter's Search Tweets API returns results encoded in JSON using an HTTP-based RESTful API. For matching Tweets of interest, all requests require a query parameter that contains filtering syntax. There are also various operators that match keywords, hashtags, URLs, and other Tweet attributes.

In Tweepy, we implement tweet search by calling the *API.search_tweets* function. The method searches and returns tweets for the last 7 days with a maximum of 100 tweets per request. In order to

get tweets older than just the last 7 days, we will need to use the *search_all_tweets* method, which however is only available if we upgrade to the academic research product track or other elevated access levels. In this project, we only searched tweets from Melbourne within the last seven days because of this limitation. The 7-day data from the search will work as a supplement to the large historical geo-coded tweet data provided by Richard.

Whenever a Twitter API error (mostly hitting the rate limit) is reached or no more data is available, the harvester will go to sleep and continue after sleeping for 15 min, since the rate limit is in the form of counts/15 min.

It is worth mentioning that if we search with the same settings (keywords, locations etc.) multiple times, the searcher is bound to return tweets that have come up before. Therefore, tweets are stored in the database with the corresponding tweet ID as the record ID. However, before storing, a check is executed to see whether a record with the same ID already exists in the database or not. This way, it can be ensured that no duplicate data is stored.

4.2 Aurin Data Mining

4.2.1 Aurin Data Searching

The AURIN workbench is a powerful and useful tool that allows researchers, academia, and industry at all levels to access. It provides an easy way to analyze and visualize high-quality data from many perspectives to enhance the research on specific countries or cities. The Aurin data has several spatial levels which need to be considered when selecting the data since it might affect the Twitter data analysis. Therefore, we spend time looking for proper Aurin datasets from the Aurin portal at the start after the topics of analysis are determined.

4.2.2 Aurin Data Preprocessing

According to the selected topic, all of the datasets we found from the Aurin portal are based on SA3 code (Statistical Areas Level 3) which means it divides Melbourne into 40 regions. Since the Aurin portal provides an attribute selection window, we are able to find the relative statistics to suit our Twitter analysis and check the missing data from there. We also use spatialization tools in the portal to generate the bounded coordinates for the analysis and mapping. As a result, we write preprocessing python scripts for each dataset to extract and combine the selected attributes according to different regions and output them as JSON files.

4.3 CouchDB Storage

The data storage layer for this system was built using the CouchDB database technology. CouchDB is a distributed database that can distribute storage systems across multiple physical nodes and coordinate and synchronize data read and write consistency between nodes. CouchDB stores data as non-relational JSON documents, which is why we pass in captured tweets as JSON objects.

4.3.1 CouchDB Set-up

As said before, three of the four instances act as ‘data nodes’ i.e. our system uses 3 nodes that are deployed on instances 1, 2, 3 to cluster the database. Each of the instances has the same status and stores the same data.

4.3.2 Database Overview

The CouchDB database is an integral part of the various components of the system. In this assignment, the database is responsible for:

- Storing raw tweets that were collected by Twitter Harvester
- Providing Map-reduce queries for processing parallelizable problems across large datasets.
- Storage of AURIN data for later scenario analysis
- Storage of final results generated by analysis script for visualization

The general view of our database is as Figure 3:



Name	Size	# of Docs
_replicator	3.3 KB	1
_users	3.3 KB	1
aurin_income	1.8 MB	40
aurin_lang	1.8 MB	40
aurin_medi	1.8 MB	40
diversity_search	1.5 MB	3459
diversity_stream	113.4 KB	257
employment_search	143.8 KB	297
employment_stream	45.1 MB	101804
health_search	90.9 KB	189
health_stream	39.2 MB	90580
output	26.6 MB	1 

Figure 3: Database Overview

4.3.3 CouchDB - Pros and Cons

First of all, CouchDB is open source so we can check the official document and peruse the source code. Secondly, CouchDB is easy to set up a cluster and provides Fauxton GUI for users to monitor the database status. In order to connect with the remote database, we only need to provide the node IP address and port number to make HTTP calls then the cluster will start working. It is very convenient and saves lots of time for the database connection and cluster set-up. With the help of the Fauxton GUI, the output of the MapReduce Views can be checked easily.

Thirdly, CouchDB can remove the duplicated tweet based on the document id. It will generate a unique id on its own or allow the users to specify the id. In our CouchDB database, each tweet that was crawled through Twitter API is stored as one document and it automatically uses the tweet id as its unique document id. Therefore, there would be no duplicated tweets in the database.

However, compared to MongoDB there are fewer tutorial resources that we can find online. The official document does not help a lot when we meet some problems such as local database set-up or database operation. Most of the resources focus on how to store and retrieve data from CouchDB instead of how to set up as well as connect with CouchDB so it takes lots of time for us to figure it out.

5 Data Analysis and Visualization

The data that is used to assist analysis is composed of three parts, the large melbourne twitter data material provided by the school, the 2016 Aurin data on Melbourne area under the SA3 standard on aspects of income, diversity and medical expenditure, and the newly harvested twitter data by the above metrics. The main focus of the analysis is to use the available data to determine the following questions:

- **How has the livability of Melbourne changed over the years by the opinions of the residents?**
- **How would the livability standard change given a resident's national identity?**
- **What is the public's perception on opportunity and health indicators?**
- **What is the geospatial difference of Melbourne on these livability indicators?**

The analysis is done based on using correlations and NLP sentiment analysis features from the TextBlob package based on a few assumptions. The general assumption made here is that a person would use twitter to express his/her feelings towards a certain topic, which if the livability of Melbourne is high, the sentiment within texts relating to focused topics would be boosted by the good livability background and hence produce a more positive sentiment score and vice versa.

5.1 Change in Melbourne's general livability views

For this section, the approach that we have taken was to use the streaming function from the Twitter Harvester to get the most updated tweets and store it in a separate database to isolate the tweets from the tweets harvested much earlier. Then the streamed data is analyzed and compared to the historical twitter data in terms of the content's sentiment. The assumption made here is that over a large number of tweets, it will gradually diminish the random fluctuations of emotion within each tweet, and by combining the generated sentiment scores for analysis, it could provide us with an overview of how the underlying sentiment has changed over time. The figure 4 below is the generated result:

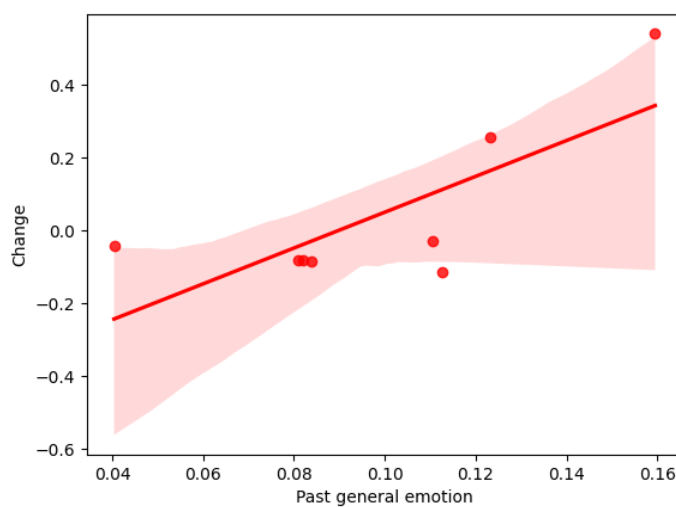


Figure 4: Change in the tweets overall sentiment

The sentiment score ranges from -1 to 1, with negative values representing how pessimistic a tweet's text is and positive values indicating the positivity within that tweet's text. In the process of gathering tweets for analysis, it is found that most streamed tweets have the geolocation feature disabled and hence were not able to identify whether the tweet indeed is from the Melbourne area, and were excluded from the further analysis.

Among the few tweets that were able to identify the tweeted location, we compared the gap in sentiments between current time and the past. Each dot in figure 4 corresponds to a real Melbourne district under SA3 criteria. It can be observed that although the graph has predicted a positive trend among the districts, meaning that the districts with more positive past sentiment is likely to have an increase in the text sentiment score, among the 7 districts presented, 5 of them show a decrease in the overall sentiment of text being tweeted. Combining the assumption that sentiment could reflect the livability in a way, it is reasonable to infer that the overall livability of Melbourne has decreased over

the years, and such conclusion is sensible given the background of disruption to normal lives in Melbourne brought by the COVID-19 pandemic.

5.2 Livability and national identity

In this section we are concerned with the question of whether the native citizens perceive the livability of Melbourne the same as those non-native residents. This topic would reveal the inclusiveness and diversity tolerance of Melbourne as an international metropolis. Here we made the assumption that a person tweeting in a language other than English in Melbourne would imply that this person has a foreign nationality background. However, in reality this would be an oversimplified assumption since nowadays more people are capable of bilingual or even multilingual abilities. Figure 5 and 6 investigated the opportunity aspect relating to non-native residents.

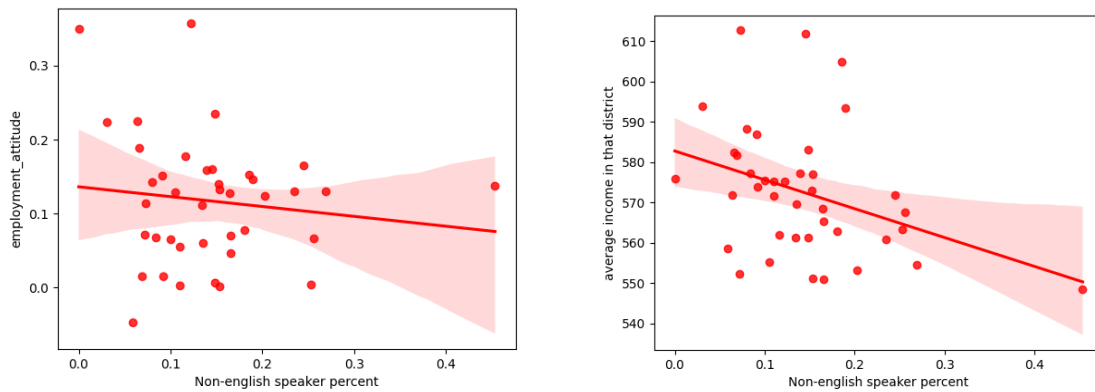


Figure 5 & 6: Plotting the percentage of non-english speakers to the opportunity indicator. Figure 5 on the left demonstrates the employment attitude against the percentage of non-english tweets within areas of Melbourne. This figure is less indicative since the sentiment analysis was performed on the English tweets relating to that topic instead of the tweets sent by the non-English users which would explicitly express their opinions, but it can still show a relative trend. The trend projection line is almost very much close to horizontal, indicating that there is no significant pattern corresponding to the non-native speakers here. However, looking at some extreme values, there is a trivial trend of districts with lower non-English percentages to an extent have more positive sentiment on the employment topic. But the conclusion here should be reached with caution since correlations do not necessarily imply causations.

The figure 6 on the right, on the other hand, suggests a more obvious trend. The y-values are yielded from the Aurin data 2016 Census and hence the accuracy is guaranteed to an extent. The percentage of non-English speakers in the area is negatively correlated with the average income with significance.

Therefore, it could be concluded from this graph that for non-English speakers, the opportunity aspect of the liveability might fall short of that experienced by the native English-speaking residents.

5.3 Public perception on opportunity and health indicators

The figure 7 and 8 below shows the general and health-specific sentiment on the income as an aspect of opportunity indicator and average medical-expenditure as an aspect of health indicator. Both of the following plots are not significant in terms of the distribution pattern. The potential issue causing a non-significant result may be due to the noises within the text categorization. In future attempts, using a more accurate topic categorization scheme might produce a better, more obvious trend for this topic, but here we can only come to the conclusion that the following two graphs are inconclusive.

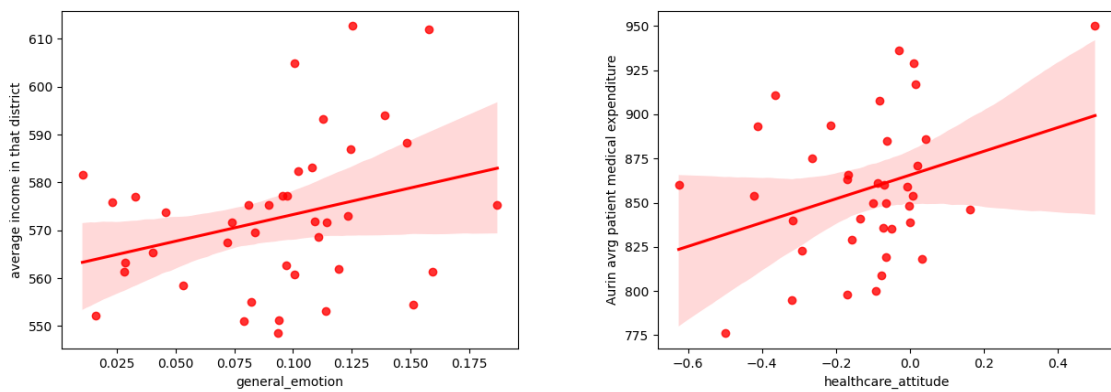
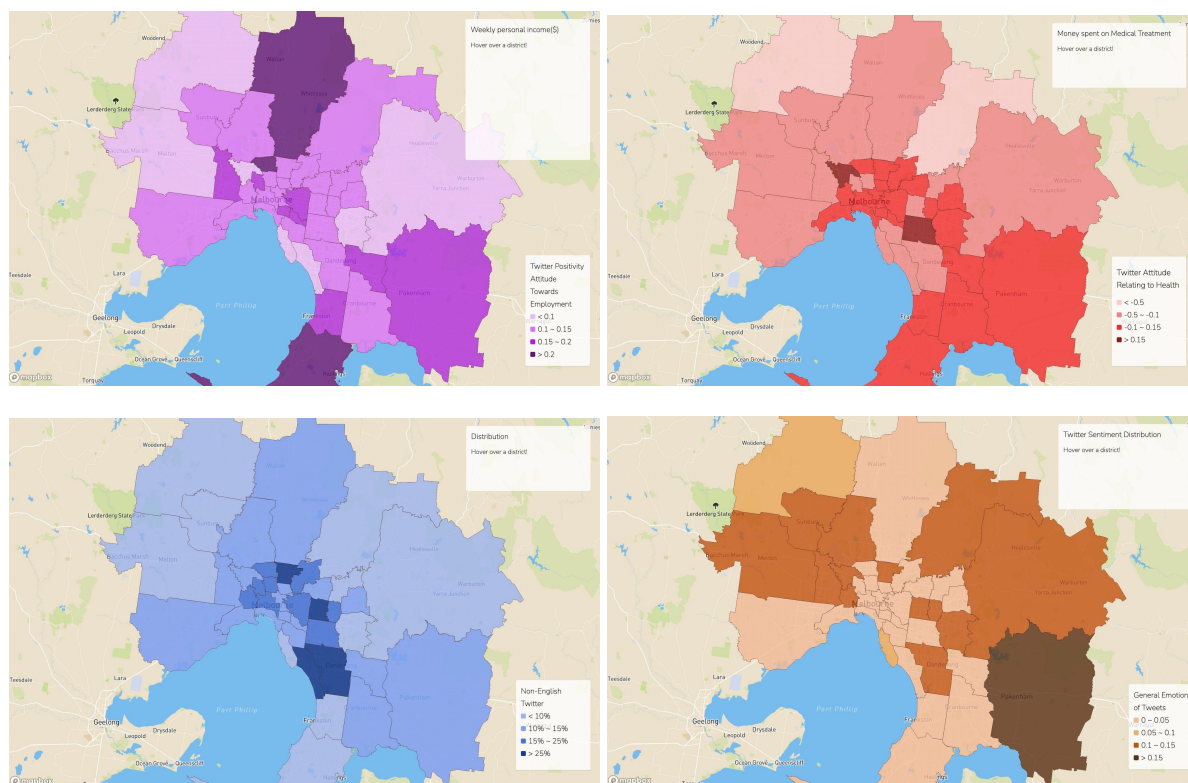


Figure 7 & 8: Plotting the emotion against the opportunity and health indicator

5.4 Geo-spatial difference on the livability indicators

The following 4 plots map the percentage of non-English speakers, sentiment on the topics of employment and health as well as the general sentiment to the districts of Melbourne to produce a more intuitive view. The plots are available on the website produced by the project for more detailed information.



The purple and the blue plot very well shows a negative correlation which is derived by the figure 6 shown above. For the health indicators, although the figure 8 failed to reach some conclusive statement, from the red geospatial plot on the top right, we can see that there exists some kind of disparity of health resources between regions of Melbourne, with the central and south-east areas reports more positive reactions on health related topics and the north-eastern areas reveals more negative perceptions on the healthcare. However, on the bottom right plot showing overall sentiment in regions, the blue and red properties are only partially preserved in this graph, with the city area of Melbourne somehow becoming more neutral given this area reports relatively high positivity in opportunity and health, which is a topic worth researching in the future.

6 Web Development

The web application was developed using flask as the framework and uses HTML and CSS to construct the front-end. Part of the web styles refers to a free open source template from W3School. 6.1 talks about the architecture used in the backend development, and 6.2 focuses on the frontend end visualization.

6.1 Backend - Flask

Flask is a web framework that is used for developing our web application. It is very easy to extend and provides a clear structure for the backend while creating the web app. It contains several libraries and modules that make the development process easier for new developers.

The backend uses Flask to build the application framework and the data was retrieved from CouchDB. The overall file structure of the flask framework includes statics, template and python script for running the web applications. The *run.py* script includes all of the routers which define the logic of the backend. When the different routes of the URL have been queried, the backend will send the request and render the HTML on the corresponding websites. The template folder contains all of the HTML files that need to be rendered. In the statics, all of the images or other resources of the frontend are stored here.

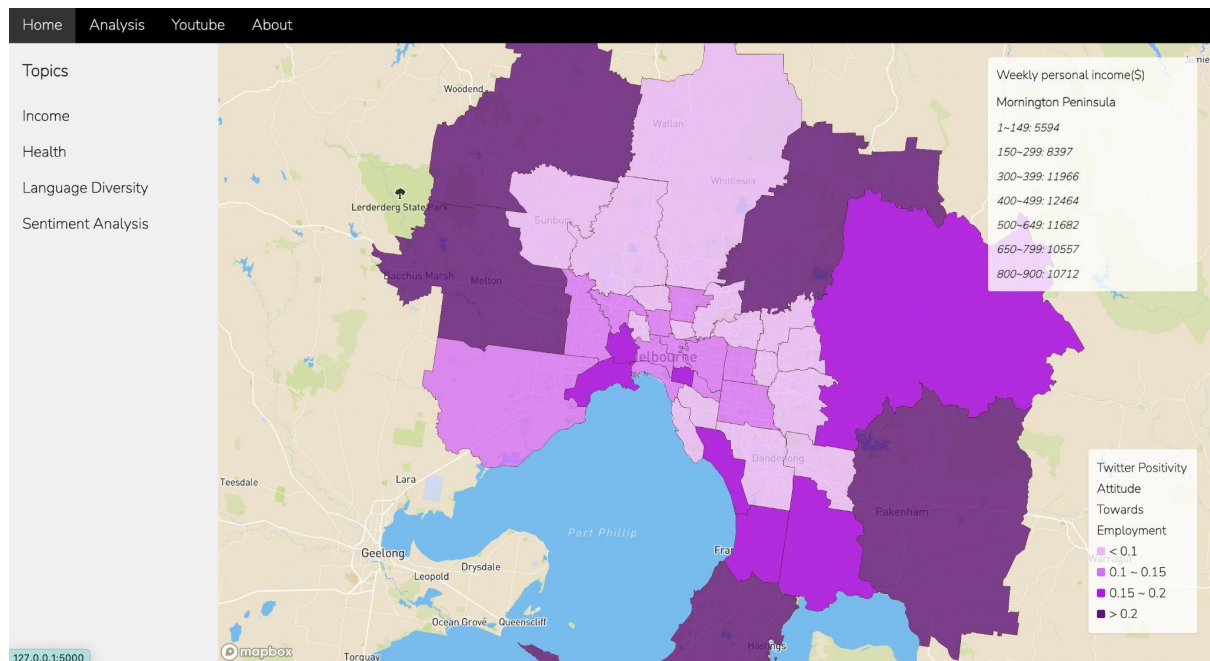
6.1.2 Flask - Pros and Cons

By using flask to develop our web application, we can benefit from several advantages. Firstly, constructing our web application with Flask is simple. It is a python based microframework which means that we can use it to produce a web app very easily and efficiently. There is the least amount of coding as compared to other frameworks. Moreover, The structure of the flask framework is easy to follow and less coding rules to learn if users understand it well in python. Compared to the “Django” framework, Flask has better performance since it is much more explicit when defining the request handlers and views.

However, it also has some limitations which are that Flask does not have a large toolbox. More extensions, such as libraries, must be manually added by developers if we want to add some functions or expand this project. Due to a large number of requests, it may cause the app to slow down. Furthermore, Flask has lower flexibility than Django but it does not bring the effects on our web application since we only use a small route in this development.

6.2 Frontend

The frontend is built mainly based on HTML and CSS. Mapbox API with some JavaScript commands has been used to present the data. There are four main parts to our web: Home, Analysis, Youtube and About as shown in the navigation bar. The UI interface of the web application as shown below:



The home page presents the maps showing different topics: Income, Health, Language Diversity and Sentiment Analysis which can be seen in the topic tabs at the left of the screen. The map has been produced by the MapBox API and it divides the map into different regions according to the SA3(Statistical Area Level 3) shapefiles from AURIN. And the GeoJSON data are provided in the form of polygon coordinates. MapBox renders the property data including the AURIN as well as the extracted Twitter data of each region from CouchDB. Each map demonstrates the attitudes of tweets towards a specific topic in various regions. The attitude of tweets is distinguished by different RGB colours where the legends are on the bottom right, darker colour stands for more positive emotion. By hovering over a region, the corresponding data from AURIN will appear at the top right corner with the name of that zone. In that way, the audience is able to view and compare each variable in different places, as well as observe the relationship between AURIN and Twitter data.

The reason for generating colour maps by MapBox in the front-end component is that Mapbox studio allows the user to customize the map to suit our own Twitter analysis. We can locate the region needed and can zoom in and zoom out smoothly.

Move on the Analysis button, there are some scatter plots for each topic. The buttons there are for controlling to go to the previous or next plot. On the Youtube page, there is a Youtube video that demonstrates how to use the web application and introduces some features. In the end, there is an About page that briefly describes our project and contains the contact information of each group member involved in this project.

7 Interface Guide

Source Code Repository:

<https://github.com/COMP90024-G2/COMP90024-project2.git>

App Demonstration:

<https://172.26.129.178>

Video Demonstration:

<https://www.youtube.com/watch?v=7l9lCaYvbdI&t=1s>

7.1 System deployment

Prerequisite :

Before the system deployment, make sure

- 1) Install ansible and ansible is available on the computer
- 2) Linux terminal is available (for Windows, install **WSL - Windows Subsystem for Linux**, which can be easily found in the Microsoft store)
- 3) Login to [Melbourne Research Cloud](#), then reset the password and save it in a text file.
- 4) Download the OpenStack RC file and rename it as **openrc.sh**

To deploy the system on MRC, the user need to follow the following steps:

- 1) Clone the project. Locate to a specific project folder and open git bash or any Linux terminal, then input the command:

```
git clone https://github.com/COMP90024-G2/COMP90024-project2.git
```
- 2) Use visual studio code or other software development IDE to open the project folder
- 3) In the terminal, (for Windows, open a WSL terminal first) input the following command to navigate to the deployment directory:

```
cd deployment
```
- 4) Move file openrc.sh to the deployment directory and move the private key team2.pem to
~/**.ssh** folder
- 5) Input the following command and wait for the deployment to complete:

```
./run.sh
```

7.2 Start web application

Open a browser and type the following address into the address bar:

<https://172.26.129.178>

8 Conclusion

8.1 General Conclusion

For investigating the livability of Melbourne, we have taken the approach of topic-specific sentiment analysis to investigate the opportunity and health components of the livability on top of a cloud-based solution that utilized University's MRC facility, CouchDB for data storage and the twitter API for data harvesting, and analyzed the results with reference to the geospatial locations and dataset provided by AURIN. Overall, it was found that there exists a disparity based on the sentiment analysis of tweets relating to employment and health both with respect to the national identity of an individual and the geospatial location based on some assumptions. For Melbourne to become the most livable city, it should provide good and equal experiences for all residents irrespective of their backgrounds. Therefore, we could not agree on the expression that it is the most livable city since there are many aspects for improvement.

8.2 Issues and challenges

In this section, we will talk about all the issues and challenges we encountered during the development of each component.

8.2.1 Twitter API

The main issues and challenges in developing Twitter Harvester are caused by API limitations. For example, in the search API, we were not able to search tweets with regional bounding boxes, due to this query operation being only available in higher-level access. For that reason, we have to manually set up mining circles (a selected geographical center with a radius). Also, each of the twitter APIs uses rate limits in different ways. Furthermore, in the process of streaming, we have noticed a performance drop-down when searching tweets with keywords set to null, i.e. the data captured per second has decreased. This might be due to the streaming algorithm.

8.2.2 MRC

While using Melbourne Research Cloud, we find that once we delete an instance and try to create a new one immediately, it always raises an "Exceeded maximum number of retries" error. Then, we cannot use the new instance.

There is one more question worth documenting about the MRC connection. Windows users outside the University network will fail to connect to the MRC through the WSL terminal after the Cisco proxy is enabled. At first, we thought it was a bug of MRC. Then, through the ping command, we

found that the two virtual hosts of WSL and Windows could not communicate with each other, that is, there was no network interconnection between these two. After a series of network experiments, we found that it was a Cisco proxy issue. The default setting of the proxy is that all network traffic must go through the proxy node before it can be sent to the destination, but WSL does not know the address of the proxy node before accessing the Windows virtual host, which constitutes an infinite loop. Finally we solved the problem by installing WSL-VPNKIT to build a virtual LAN between WSL and Windows. Of course, the most effective way is to change Cisco's traffic settings so that requests from virtual nodes do not preferentially flow through proxy nodes, but as a student user, we are not able to do that.

8.2.3 Database

As mentioned before in section 4.3.3, the duplicated tweets can be removed based on the unique document id. The Couchdb database stores each tweet as a single document and uses tweet id as its unique document id. Therefore, the twitter harvester will ask couchdb to check whether the same id occurs and discards the tweet that its id has existed in the database.

When we try to download the result data from couchdb database by using the “curl GET” method, it always downloads a html file instead of a correct json file. After we found that “/utils” is pointed at the Fauxton web-admin not to the database resource, we are able to successfully retrieve data from the database to the local.

8.2.4 Aurin

When we search the Aurin data from the Aurin portal, there is lots of data with different spatial levels and attributes. It takes a lot of time to search for the proper datasets since we have to compare with the twitter data. Missing data can be avoided since the Aurin portal provides an attribute window to preview the data. In addition, we choose the Aurin data based on Statistical Areas Level 3 in order to unify the geographical area in the combined data analysis.

8.2.5 Analysis

The validity of the assumptions made in the analysis part is unchecked, therefore some of them might be too generous and not reflecting the real underlying characteristics of the topic that has been investigated. For example, the assumption that tweets' sentiment is a reflection of the people's feeling of the livability of Melbourne might not be concise since people might tend to exaggerate their feeling in an online environment. Besides, for some analysis sections, due to the API limitation, not enough data is generated to produce an unarguable and meaningful result. Therefore, to reinforce the validity

of this research, the methods used in the report should be used alongside with other methods, such as real-life surveys, to provide a more general view for interpretation.

8.2.6 Scalability

Initially when harvesting APIs would not generate a sufficient number of outputs for analysis, we would amend it by adding historical tweets provided in the 10G Melbourne tweet dataset which provided approximately 100 million entries. As the development continued, we discovered that such data still served a very functional role in the data analysis and therefore this feature is retained in the final script. Hence, when no sufficient entries of data could be harvested by twitter API, it can always be amended by loading more entries locally. Due to the computing power limitations, the application would have a significant performance reduction over a larger read of the local dataset, which required approximately 30 minutes to process up to 100 thousand entries. As for online scalability, it has not been comprehensively studied since the Twitter API currently does not possess the ability to generate such vast harvests that would significantly influence on the performance. However, it is also because of this limitation, the application did not perform any optimizations for large reading of dataset from the Couch DB, therefore potentially resulting in an issue for drastic up-scaled size of input data.

9 Team Collaboration

This section mainly introduces the responsibilities of each team member. During the development process, each team member is assigned an independent task. However, due to the cross-functionality nature of this project, some development is completed with mutual efforts. The specific tasks of the team members are as follows:

Member	Task	
Hengzhi Qiu <i>1253748</i>	<ul style="list-style-type: none">- Implementation of the Twitter Harvester (both Searcher and Streamer)- Assistance in tweet analytics and debugging	
Claire Zhang <i>1080915</i>	<ul style="list-style-type: none">- Backend and Frontend web development- Make Youtube Demo- Retrieve data from CouchDB	
Yi Yang <i>1074365</i>	<ul style="list-style-type: none">- Backend and Frontend web development- Searching and preprocessing Aurin data- Make Youtube Demo	
Yifeng Pan <i>955797</i>	<ul style="list-style-type: none">- Implemented of the Twitter Data Analytics and program debugging- Implemented of the Couch DB Harvest Data retrieval and database result storage	
Yonghao Hu <i>1049814</i>	<ul style="list-style-type: none">- Implement ansible playbook to deploy cloud infrastructure and set up CouchDB cluster- Deploy the tweet harvester and Web application	