An Investigation into the Liveability of Melbourne through the Voices of the People

Xinhao Chen
1230696
xinhchen1@student.unimelb.edu.au

Junjie Xia
1045673
juxia@student.unimelb.edu.au

Weimin Ouyang
340438
wouyang@student.unimelb.edu.au

Tianqi Yu
1221167
Tyyu2@student.unimelb.edu.au

Yuling Zheng
954408
Yulingz3@student.unimelb.edu.au

Abstract

A scalable system was deployed on instances across the Melbourne Research Cloud for the COMP90024 Cluster and Cloud Computing course at the University of Melbourne. The system has two components: a Twitter harvester and a web application. It supports analytic scenarios exploring the liveability of Melbourne. All components of the system were containerised by Docker and the deployment was fully automated using Ansible. Tweet data along with data from Australian Urban Research Infrastructure Network and other sources were used to develop the analytic scenarios. This report includes details and in-depth discussion of the architecture, deployment, and components of the system, as well as the analytic scenarios supported by the system.

Table of Contents

1. Introduction

Being awarded the world's top city for seven consecutive years until 2017 by the Economist Intelligence Unit's Global Liveability Index, Melbourne had long enjoyed the crown of 'the most liveable city in the world'. In addition, the city had been consistently ranked among the world's top three most liveable cities ever since the index began in 2002 (Global Victoria, n.d.). In the last few years, however, there was a drop and some fluctuations in Melbourne's liveability rankings (Eddie, Preiss & Estcourt, 2021).

While the liveability of a city is typically determined by a set of indicators including housing, neighbourhood, transportation, environment, health, engagement and opportunity, each tapping into a particular category containing metrics and policies (AARP, n.d.), voices of the people living there were usually left unheard during the assessment.

The current project therefore explored into aspects of the liveability of Melbourne through the voices of people. Scenarios were developed for chosen liveability indicators of opportunity, housing, transportation and neighbourhood, with trending hashtags and cost of living as additions. Given the proliferation of online social media platforms such as Twitter, Tweet data were primarily employed as the evidence basis, as tremendous amount of relevant discussions were expected to be discovered there. Data from the Australian Urban Research Infrastructure Network (AURIN) were also employed to assist data analyses for further insights.

The final deliverable of the current project was a system deployed, using Docker containers, on instances across the Melbourne Research Cloud (MRC). Components of the system included a Twitter harvester that collected and saved Tweet data into the CouchDB in real time, and a web application that visualised selected scenarios through analyses of the Tweet and the AURIN data retrieved from the CouchDB.

The general system architecture of the final deliverable was designed to operate in a cloud environment with high scalability, while the deployment of its components on the MRC was fully automated with the use of Ansible (Red Hat Ansible, n.d.). In addition, management of the deployment of the system components across the instances on the MRC was achieved by utilising the Docker technologies (Docker, n.d.). Sections 2 and 3 respectively describe in detail the system architecture, and the deployment of the system on the MRC.

Tweet data that supported the development of analytic scenarios were primarily collected through the Twitter APIs accessed by the Tweepy Python library (Tweepy, n.d.), and were stored

in the CouchDB (Apache CouchDB, n.d.) hosted on the MRC. Part of the provided historic Tweet data, as well as processed AURIN and other data, were also imported into the CouchDB to complement analyses. Section 4 details the design logic of the Twitter harvester and the process of data collection.

The CouchDB adopted a clustered database setup to enjoy performance advantages. The capabilities of its incremental MapReduce system significantly eased data processing and supported data analyses. An overview of the databases in the CouchDB and the MapReduce views that were created for data processing is provided in Section 5.

Data analyses were performed on the Tweet, the AURIN and other data to support the development of the selected scenarios. Data analyses functions drew on the MapReduce views predefined and saved in the CouchDB to retrieve data from the databases, and exploited available Python libraries and packages. Results of the analyses were visualised with Highcharts wrapper for React (Highcharts, 2021) and Mapbox GL JS (React-map-gl, n.d.) using React wrapper to provide a React API on the web page frontend. Section 6 covers the detail on the data analyses steps and how selected scenarios were supported by analyses performed. The design and the functionalities of the frontend, as well as the actual data visualisation results are presented in Section 7.

To achieve the final deliverable, resources of 4 instances with 8 virtual CPUs and 500 GB of volume storage were provided. A breakdown of the development activities involved are as follows:

- Using Ansible to set up instances on the MRC
- Using Ansible to mount the volumes on individual instances
- Using Ansible to set up an internally replicate CouchDB cluster across all instances
- Creating Twitter harvester to collect tweet data and save to CouchDB
- Using Docker to set up the Twitter harvester across all instances
- Developing data analytic scenarios with Twitter and AURIN data, employing the MapReduce technique supported by the CouchDB
- Creating the frontend of the web application
- Dockerising the frontend

Information regarding how these activities were split among the team members and the project management methodologies adopted are outlined in Section 8.

Section 9, in addition, provides a simple user guide for testing.

2. System Architecture & Cloud Infrastructure

2.1 System Architecture

The system architecture of the current project was designed with an emphasis on scalability and reliability. There are two core components of the system, the Twitter harvester, and the web application, each with their subcomponents. They together served to support the core focus of the current project, namely the analytic scenarios exploring the liveability of Melbourne.

Given the large data volume involved in data analytics, leveraging computing power to process the data was one of the challenges the system faced. Allocating limited resources wisely hence was important. Figure 1 and 2 shows how provided resources were allocated for the current project.

The core components of the Twitter harvester and the web application were both containerised using one or more Docker containers to enjoy the benefits of easy deployment and isolation of dependencies. Figure 3 illustrates the architecture of the system and the interactions between different components and subcomponents.

Figure 1

Summary of the Utilised Resources

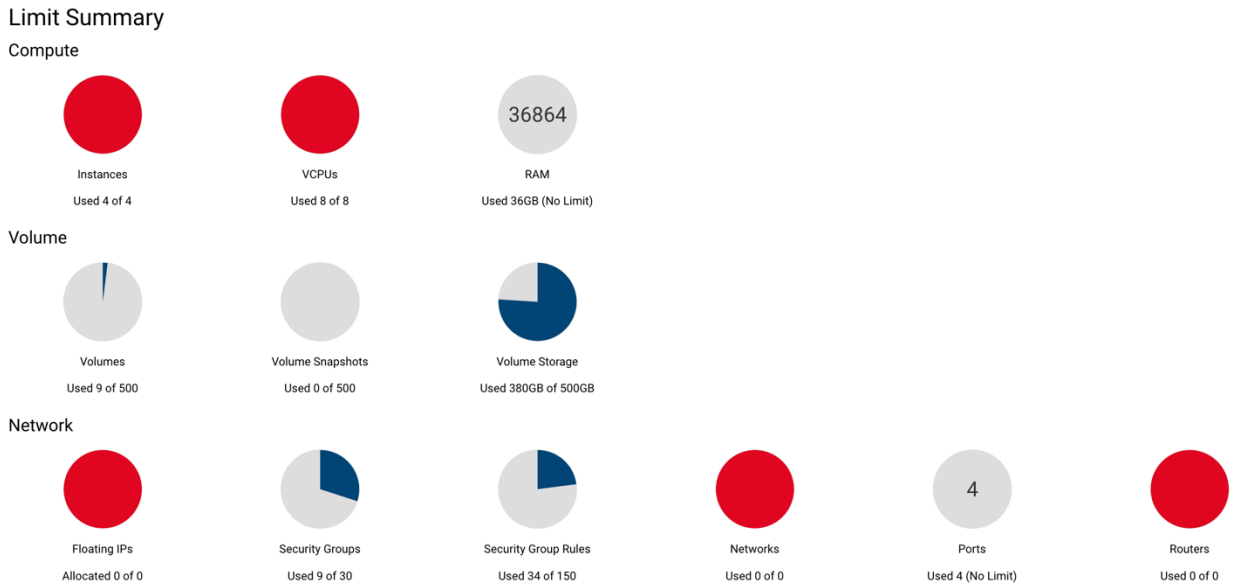


Figure 2

Summary of the Instances Created

Usage

Download CSV Summary

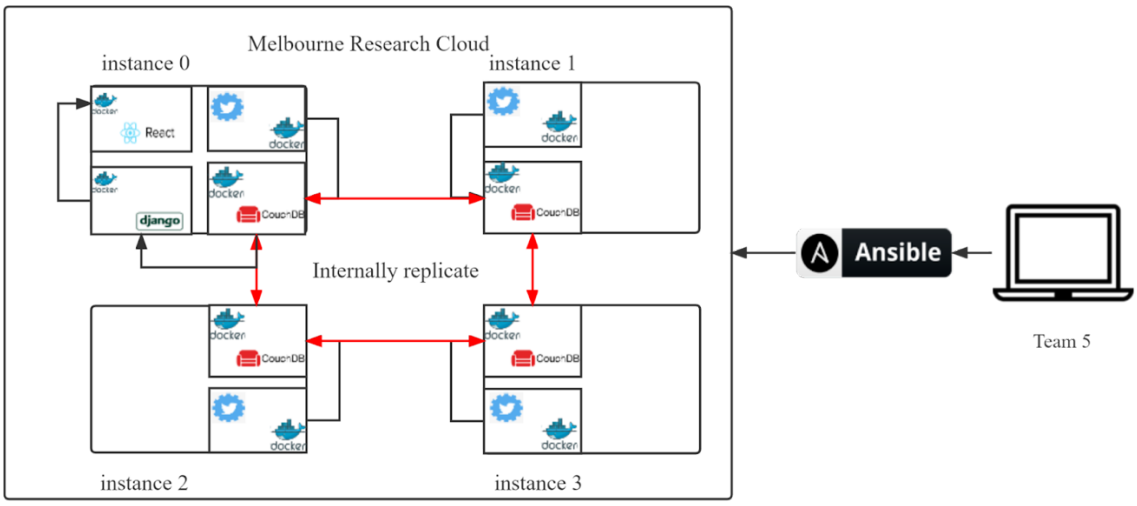
Displaying 4 items

Instance Name	VCPUs	Disk	RAM	Age
server2	2	30GB	9GB	2 days, 2 hours
server1	2	30GB	9GB	2 days, 2 hours
server0	2	30GB	9GB	2 days, 2 hours
server3	2	30GB	9GB	2 days, 2 hours

Displaying 4 items

Figure 3

System Architecture



2.1.1 Twitter Harvester Component

Twitter Harvester. The Twitter harvester was responsible for collecting Tweet data for analytic uses. It was deployed across all instances on the MRC with a Docker image. The Tweet data it collected were saved into the CouchDB database.

CouchDB. CouchDB served the data storage functionality of the current project. It was containerised and deployed across all four instances on the MRC. The CouchDB database adopted a clustered design to improve system scalability and reliability.

Scalability. With a clustered design, the database can be scaled dynamically depending on the actual volume of data involved. The CouchDB cluster would continuously add more capacity to increase the size of the cluster without taking it down by its unique strategy of oversharding (Apache CouchDB, 2022).

Reliability. With a clustered design, the CouchDB internally replicates with each other, which provided great fault tolerance. That is, in unfavourable situations where some of the instances fail, the CouchDB would still not crash but working properly as long as at least one of the instances that hosted the CouchDB cluster is still running (Apache CouchDB, 2022). As such, possible failure of instances can be overcome automatically.

2.1.2 Web Application Component

The web application component was consisted of the frontend and the backend. Users of the web application could visit the web page frontend to browse through and interact with data analytic charts. The data presented on the web page were firstly queried and retrieved from the CouchDB, processed through data analyses functions, and then sent to the frontend by the backend.

Frontend. The frontend of the web application was responsible for displaying data analyses results using graphs and maps. Axios was used to fetch data from the backend.

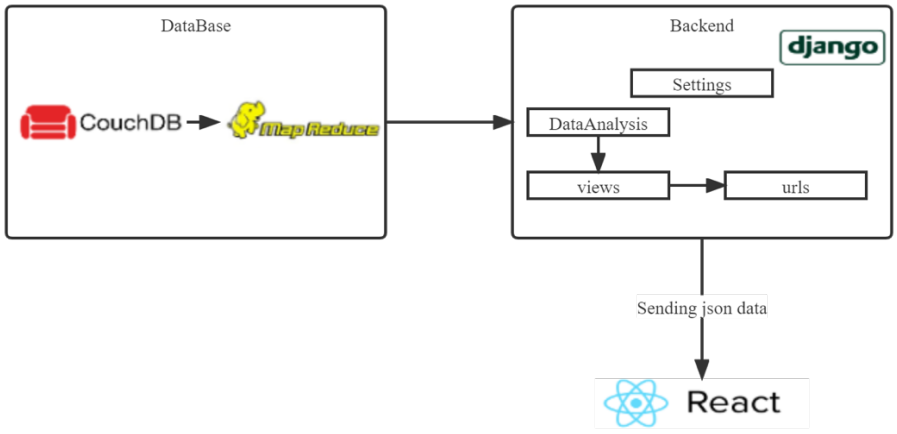
Backend. The backend of the web application was responsible for integrating data analyses functions to process and package data, and for creating connection and transmitting information between the backend and the frontend. It adopted a ReSTful design.

The Python web framework Django was used to establish the development of the backend. Django was selected primarily because it is a heavy-weighted framework with many APIs and interfaces to work with (Django, n.d.), which could drastically relieve the burden for development. Figure 4 demonstrates the working process between the frontend and the backend.

Model Views Controller is the basic design pattern the Django framework used. Generally, Django integrated data analyses functions to first retrieve raw data from the CouchDB through generated MapReduce views. Django views then received processed data and rendered them into the JSON format, packaged together with request and response methods and status codes to generate URLs. URLs represented directions to the Django views, from which the frontend could receive data and eventually present to the users. During this process, dependencies and collisions, such as cross-domain conflicts, were all well-handled in settings.

Figure 4

Working Process Between the Frontend and the Backend



2.2 Melbourne Research Cloud

The MRC was the platform on which we deployed our instances. It contained about 20,000 virtual cores and have available instances of multiple sizes (Melbourne Research Cloud, n.d.). The MRC is based on the open-source software collectively known as the OpenStack (Openstack, n.d.), which enables users to create multiple instances and access the OpenStack Application Programming Interface (API). Multiple advantages and disadvantages of the MRC have been identified throughout the development of the current project.

Advantages

Easy deployment. Users of the MRC could quickly deploy the system with the use of an Ansible script which is also reusable in the future. Furthermore, the system provides several types of operating systems which greatly benefit the users in terms of both the availability of choices and efficiency.

Resource saving. As a general benefit of the Cloud, service providers could share the resources with more people with a Cloud infrastructure. By contrast, one computer could only be used by one person at a time, which is a great waste of resources.

Flexibility. Users could apply for the resources they need on the MRC without paying extra to buy a server.

Easy collaboration. Using the MRC was convenient for students completing group projects, because every team member could use ssh to visit the instance on the MRC. However, if the system

is deployed on the personal computer of a student, others may not be able to visit the IP address easily.

Disadvantages

Limitation on system access. The system could only be visited within the University of Melbourne or by connecting to the Cisco VPN, making it difficult for the users.

Limitation on system resources. The MRC does not have some software to accelerate project development or improve team collaboration as in AWS or other Cloud infrastructure, which offer a large variety of more sophisticated services. It only provides basic functionalities of the Cloud.

Low maintenance. The MRC is only supported and maintained by a small number of staff in the University of Melbourne. As a result, when the system goes down, it may stay down for a few days. In fact, during the development of the current project, we have experienced a 2-day system downtime due to the failure of a hypervisor. That caused delay in our progress.

3. System Deployment

3.1 Ansible

Ansible is an IT automation engine that automates cloud provision, configuration management, and application deployment. Ansible scripts are called playbooks, written as YAML files. YAML is a human-readable language that makes automation jobs easier to write and read as it approaches plain English (Red Hat Ansible, n.d.).

An ansible module is usually structured in a simple folder hierarchy with 4 parts: the variables, the inventory, the roles and the playbook. The variable file defines any variables to be used during the execution of the roles. The inventory stores the list of instances/hosts in the infrastructure. The roles state the tasks that are going to be executed in the module. While the playbook finally assigns the roles to the hosts in the infrastructure.

In the current project, the creation of the instances, and the deployment of the Twitter harvester and the web application on the MRC were done using a single shell script that executed ansible modules. The structure of the automated deployment is shown in Figure 5.

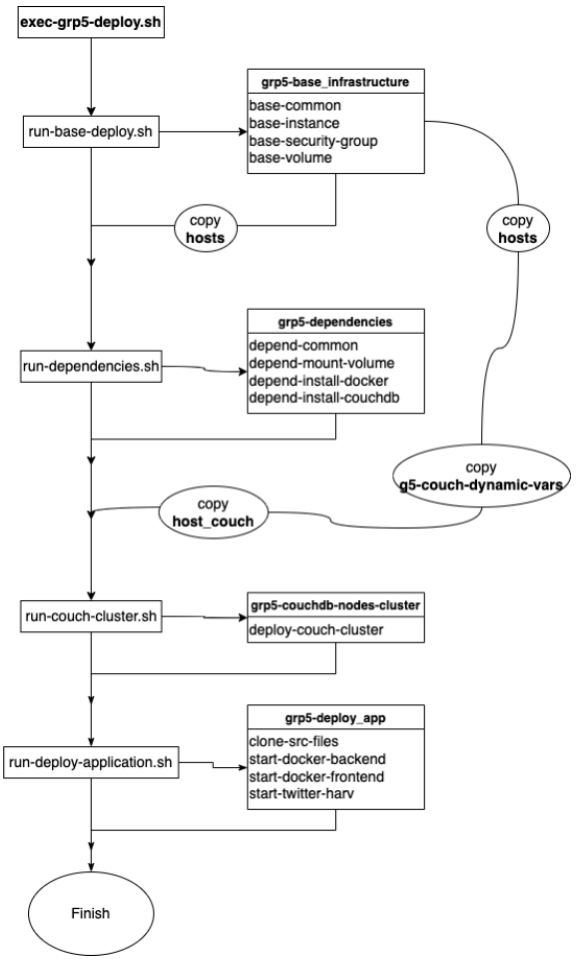
Automation with ansible enhanced scalability of the system. The traditional way to deploy software applications on cloud-based servers would require the use of Unix-like commands to interact with the remote servers. It would be inconvenient for developers to deploy different

applications with a variety of requirements and dependencies, let alone the differences on each local machine. With ansible, the system can be deployed from any Unix-like machines to the remote servers on the MRC by simply executing the ansible scripts. There is no need to know the required configurations and dependencies of the system.

As shown in Figure 5, the complete ansible script consisted of four ansible modules, and these modules were managed by 'exec-grp5–deploy.sh'. The shell script was the entry point of the scripted deployment, it was responsible for executing each ansible module in a serial execution order, and creating and moving the inventory files to subsequent modules dynamically. The execution flow, in addition, started from creating the basic infrastructure on the MRC, installing required dependencies, deploying the CouchDB cluster on the instances created and finally deploying and running the web application.

Figure 5

Structure of the Automated Deployment



3.1.1 Basic Infrastructure

The first step of deployment was to create the instances on the MRC. As an entry point of deployment, this part was executed on the localhost to create and configure the basic infrastructure on the MRC.

The variable file 'base_var.yaml' defined the following variables: the volumes, the security groups and their rules, as well as the instances; while the following roles were executed in order.

base-common. This role installed the Openstack Python SDK, which was required to interact with the MRC. Other packages such as pip were also installed for subsequent tasks.

base-volume. This role created the volumes and named them on the MRC according to the IDs and names defined in the base_var.yaml.

base-security-group. This role created and named the security groups and set the rules according to the variables defined in the base_var.yaml. Ports 3000 and 8000 were opened for later deployment of the frontend and the backend of the web application.

base-instance: This role created the instances on the MRC using the volumes, the security groups, and the other relevant configurations such as the availability zone and the flavour that have already been created in previous roles.

Upon the completion of the creation of the basic infrastructure, four instances had been created and started running on the MRC. The module then created the following inventory files for subsequent tasks:

hosts. This file stored the IP addresses for all four instances on the MRC. They were categorised under the name 'Webservers'. The first instance was assigned as the master and was named 'Webserver'.

hosts_couch. This file stored the IP address of the master instance, which was to be used as the master node in the CouchDB cluster.

g5-couch-dynamic-vars.yaml: Similar to **hosts**, this file stored the IP addresses of all instances, created to be used as the variable file in the deployment module for the CouchDB cluster.

3.1.2 Dependencies

The next step of deployment was to install dependencies to the instances.

The variable file 'depend_var.yaml' defined the mount points for mounting the volumes. Two volumes were mounted on each instance at two separate paths with the larger volume

designed to provide storage for the CouchDB and the other for storing the Docker images and containers.

The following roles were executed in order:

depend-common. This role installed required dependencies such as apt and curl for subsequent tasks.

depend-mount-volume. This role first created a file system on each instance, and hence the paths for various directories. The volumes were then mounted at the paths as specified in the variable file.

depend-install-docker. This role downloaded Docker from the Internet using apt-get, then installed both Docker and Docker-compose on each instance.

depend-install-couchdb. After Docker was installed, this role installed the CouchDB on each instance by building the Docker images for the CouchDB.

3.1.3 CouchDB Cluster

The CouchDB cluster deployment module enabled the CouchDB cluster after the creation of the docker image for the CouchDB.

The variable file 'g5-couch-dynamic-vars.yaml' containing the IP addresses of all instances was copied from the last section by the main script 'exec-grp5–deploy.sh'. The CouchDB cluster were deployed on all instances.

There was one role executed:

deploy-couch-cluster. This role first registered the cluster nodes with the master node using a curl command. The username and password for the CouchDB were created and passed into the command. It then added each node into the cluster by iterating through the variable file. Finally, it completed the cluster setup using curl as well (Apache CouchDB, 2022).

3.1.4 Applications

The application deployment module deployed the Twitter harvester and the web application after the CouchDB cluster was set up on all instances.

The Twitter harvester was deployed on all the instances referred to as 'Webservers', while the web application was deployed only on the master instance referred to as 'Webserver'.

The following roles were executed in order:

clone-src-files. This role executed the git clone command to clone the latest source files from the main branch of the GitHub repository for the current project into the destination repository created on each instance.

start-twitter-harv. This role first stopped and removed any existing Docker container and the image of the Twitter harvester on each instance, then built new Docker images and started them in the background.

start-docker-backend. This role was executed only on the master instance. It stopped and removed any existing Docker container and the image for the backend of the web application, and then built a new Docker image for the backend.

start-docker-frontend. This role was executed only on the master instance. Similar to the last role, it stopped and removed any existing Docker container and the image for the frontend of the web application, and then built a new Docker image for the frontend. After that, the Docker-compose command was executed to start both the frontend and the backend Docker images in the background.

Up until this stage, the deployment of the current project had been completed. Figure 6 shows the output after the execution of all four ansible modules.

3.2 Docker

Docker is a containerisation platform. Containerisation entails placing a software component and its environment, dependencies and configuration into an isolated unit called Docker container (Docker, n.d.). This enables developers to deploy applications faster and consistently on both on-premises and cloud-based environments, hence enhances system scalability.

When deploying, shipping, and running applications on remote instances, instead of installing required dependencies and running applications manually on the instance, developers can simply create a Docker container for the application and run the application by starting the container.

In this project, all the CouchDB, the Twitter harvester and the web application were dockerised into images and run inside containers. To run the application inside the container, a Dockerfile is required in order to create a Docker image. The Dockerfiles are attached in Appendix A, while a screenshot of the running Docker containers on the master instance is included in Appendix B.

Figure 6

Deployment Output

```
TASK [start-docker-backend : remove backend docker container] ******************************************
changed: [172.26.134.129]

TASK [start-docker-backend : remove backend docker image] **********************************************
changed: [172.26.134.129]

TASK [start-docker-frontend : remove frontend docker container] ****************************************
changed: [172.26.134.129]

TASK [start-docker-frontend : remove frontend docker image] ********************************************
changed: [172.26.134.129]

TASK [start-docker-frontend : run docker compose] ******************************************************
changed: [172.26.134.129]

PLAY RECAP *********************************************************************************************
172.26.129.71              : ok=8    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.26.130.21              : ok=8    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.26.134.129             : ok=14   changed=10   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.26.134.2               : ok=8    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

3.3 Challenge and Refinement

While with Ansible the whole system can be deployed through the script automatically, during the process of deployment, any mistakes and bugs will not be discovered until the ansible module finishes executing. More importantly, the output of the ansible execution can only tell which task failed, making locating bugs challenging as the mistakes and bugs may not only be in the ansible script but also be from the source codes.

Challenge 1

When deploying the web application, port 3000 was designed to be the address for the frontend, and port 8000 was for the backend. However, although the ansible script had been successfully executed and the containers for the web application were running properly on the master instance, the connection to the web application was still refused.

Solution

In order to access the web application on the instance, all required ports need to be opened on the instance. More importantly, it's not recommended to just simply open all ports from 0.0.0.0/0 for any reason, as this would lead to security vulnerabilities. Therefore, a security group was created for the deployment of the web application where ports 3000 and 8000 were opened from 0.0.0.0/0 within this group only (Figure 7). After correctly setting the security group, the connection to the web application was successful.

Figure 7

Security Groups for the Web Application

```
- name: g5_http
  protocol: tcp
  port_range_min: 3000
  port_range_max: 3000
  remote_ip_prefix: 0.0.0.0/0
- name: g5_http
  protocol: tcp
  port_range_min: 8000
  port_range_max: 8000
  remote_ip_prefix: 0.0.0.0/0
```

Challenge 2

One of the key Ansible modules was for the deployment of the CouchDB cluster on all instances on the MRC. Although the CouchDB cluster could be set up manually, decision was made to build it using ansible for further automation and scalability. The challenge was related to integrating the required CouchDB configuration with the ansible script. The CouchDB configuration consisted of setting up the manager account, configuring the file permissions and exposing the relevant ports for public access. Figuring out the way to complete the configuration using ansible script was difficult and buggy.

Solution

Considering the inconvenience of building the native version of the CouchDB, the containerisation technique was used. As mentioned in section 3.1.2, Docker images were used for building and starting the CouchDB on each instance. The detailed script for building the Docker image for the CouchDB is shown below (Figure 8).

Figure 8

Building the Docker image for the CouchDB

```
# Create new docker container for CouchDB and start container
- name: Create and start CouchDB Docker container
  become: yes
  docker_container:
    name: couchdb
    image: "ibmcom/couchdb3"
    state: started
    recreate: true
    ports:
      - "5984:5984"
      - "4369:4369"
      - "9100-9200:9100-9200"
    volumes:
      - /data
    pull: yes
    env:
      COUCHDB_USER: "{{ user }}"
      COUCHDB_PASSWORD: "{{ pass }}"
      COUCHDB_SECRET: "{{ cookie }}"
      ERL_FLAGS: "-setcookie \"{{ cookie }}\" -name \"couchdb@{{ inventory_hostname }}\""
```

4. Data Collection

4.1 Twitter Harvesting

A Twitter harvester was developed using the Python programming language to collect Tweet data. The Twitter harvester collected tweets through Twitter APIs, while the Tweepy Python library was employed to access the methods that Twitter API offers. A number of considerations were taken into account when designing and implementing the harvester.

4.1.1 Design Considerations

Geographical location. Provided the objective of the current project was to explore the liveability of Melbourne through the voices of people, the Twitter harvester must be able to identify and gather tweets from Melbourne alone.

This requirement on geographical location implied that only Twitter API endpoints with enabled geo-related searching or filtering could be employed for the Twitter harvester. While in theory it is possible to first harvest tweets from any geographical locations before subsequent geo-related filtering for tweets from Melbourne, such approach would be extremely inefficient. Given the chance of a randomly selected tweet happening to be geo-tagged with 'Melbourne', either with a coordinate point, or a place id, is extremely low, this approach would result in an incredible amount of collected tweets to be discarded and wasted. Such a waste is both a waste of time, computing power, as well as the monthly tweet collection quota of the Twitter developer accounts, and thus should be avoided in the first place.

Twitter API access levels and versions. Twitter offers different access levels for developer accounts. The access levels available had implications on the design of the Twitter harvester and the choice of the Twitter API endpoints employed.

With Essential access, developers can access the upgraded API v2, and has a quota of 50,000 tweets per month to collect. Accounts with Elevated access, on the other hand, are allowed access to both the Twitter API v2 and v1.1 and have a monthly quota of 2,000,000 tweets (Twitter, 2022). However, the Twitter API v2 does not support geo-related searching or filtering for Essential or Elevated access, while the Twitter API v1.1 comes at least with enabled geo-filtering for the streaming endpoint (Twitter, 2022). Provided the consideration regarding geographical location discussed previously, the use of the Twitter API v1.1 became a must and hence the elevated access.

Elevated access was therefore applied to enable access to the API v1.1 and hence to gain geo-related filtering in the streaming endpoint. Further, the 2,000,000 monthly quota that came with Elevated access implied that the design of the Twitter harvester could set aside worries on exceeding quota as it is unlikely to be exceeded.

Twitter API endpoints. Two Twitter API endpoints were employed for harvesting tweets for the current project: the streaming endpoint and the user Tweet timeline endpoint. The streaming endpoint allows streaming of public tweets from the platform in real-time (Twitter, 2022), while the user Tweet timeline endpoint provides access to tweets published by a specific Twitter account (Twitter, 2022). As discussed previously, the use of the streaming endpoint was warranted by the consideration on geographical location, while the purpose of employing the user Tweet timeline endpoint was to improve the overall efficiency of the harvester. Further detail on the selection of these two endpoints are elaborated in section 4.1.2 where the harvesting process is covered.

Time range. The time range of the Twitter harvester was set to from 1st Jan 2018, so only tweets posted from 2018 onwards were collected. The decision was made based on exploration of the provided 10 GB historic tweet data. Preliminary exploration of the historic tweet data revealed that they contain tweets posted between the years 2014 and 2017, with more than 3,000,000 from the year 2017 alone. Setting the time range to from 2018 therefore best avoided collecting duplicate tweets and hence saved on the computing power and the monthly collection quota.

Duplication of tweets. The project requirements specify that the system should be designed so that duplicate tweets will not arise. To achieve this, the unique identifying id strings of the tweets were used as the document keys when they were saved into the CouchDB. Specifically, when a new tweet was collected, its id string would firstly be checked against the existing document keys in the CouchDB, and it would only be saved if there was not a duplicate.

In a similar vein, after the tweets from a user's timeline were collected, the user id string would be saved into a user database in the CouchDB as the document key. Then when a new user id is fetched, it would firstly be checked against the existing document keys in the user database, and the tweets in the user's timeline would only be gathered if the id string was not already there.

Duplication of tweets therefore would not arise, while a same user would not be searched twice, so that no monthly quota was wasted on collecting duplicates.

4.1.2 Twitter Harvester Design and Harvesting Process

Keeping in mind the aforementioned design considerations, the Twitter harvester was designed and operated as the following:

1. By specifying the bounding box of Melbourne with the streaming endpoint, the harvester streamed real-time tweets posted in Melbourne. Tweets posted outside of the bounding box of Melbourne would automatically be discarded, not counting towards the monthly quota.

   In addition, as the Twitter harvester was deployed across all instances on the MRC, the bounding box of Melbourne was divided into four smaller boxes according to the names of the instances. The harvester on each instance was only responsible for harvesting tweets made inside the smaller bounding box, while together they make up the whole Melbourne. This approach further reduced the probability that duplicate tweet would be streamed and thus the number of tweets to be discarded, saving on the computing power and the monthly quota.

2. Once a tweet from Melbourne was streamed, the user id string of the user posted that tweet was captured and fed into the user Tweet timeline endpoint. While the user Tweet timeline endpoint searched all tweets a particular user posted in 2018 or later, only tweets geo-tagged with Melbourne's place id were saved to the CouchDB to ensure all tweets gathered were from Melbourne.

   The logic behind such design was that if a user posted a tweet in Melbourne, that user is more likely a person living in Melbourne. Therefore, there would be a higher probability that other tweets the user had posted before were also geo-tagged with Melbourne. This way, the user Tweet timeline search significantly improved the overall efficiency of the Twitter harvester, and managed to gather a greater number of tweets of interests than streaming. Streaming, on the other hand, essentially acted as a continuous seeding process.

3. Tweets collected from streaming and user Tweet timeline search were both saved into the 'raw_tweets' database in the CouchDB. As discussed previously, the id strings of the tweets were used as document keys to prevent duplication, while the user id strings were saved into the 'user_list' database to avoid double searching of the same user.

   Because the target 'raw_tweets' and 'user_list' databases need to firstly exist in the CouchDB for the Twitter harvester to save data in, a database connection function was

implemented at the initialisation of the harvester to ensure that. Calling the database connection function build connections with the target databases, or create them if they don't already exist, so that the harvester could immediately start gathering tweets at running.

No search terms were used in the harvesting of tweets and the collected tweets were not trimmed down before storing into the CouchDB. The choice was made to better preserve information tweets provide, so to allow more possibility for later data analyses in supporting the development of the selected scenarios.

4.1.3 Error Handling

The use of the Tweepy library eased handling errors such as hitting the rate limits, as there is an additional parameter to pause and wait until the time is up. Other than that, the Twitter harvester did not encounter any error at running.

4.2 AURIN and Other Data

The current project used the 2016 census data on country of birth and language spoken at home retrieved from the AURIN portal to support the analytic scenario for the opportunity indicator. Precisely, two datasets, 'SA1-P09 Country of Birth by Sex-Census 2016' and 'SA3-P13 Language Spoken at Home by Sex-Census 2016' were found by keyword searching with keywords such as 'language' or 'birth country' and were downloaded as csv files.

Since our geographical location of interest was Melbourne alone, 'Greater Melbourne' was selected in the Area panel on the AURIN portal to adjust the data retrieved. No further geographical processing was thus needed.

In addition, data on percentage changes of the residential property price of Melbourne were collected from the Australian Bureau of Statistics (ABS) website to support the analytic scenario for the housing indicator.

5. Data Storage and Processing

5.1 CouchDB Overview

As the data storage component of the system architecture, the CouchDB was responsible for:

1. Storing raw tweets and searched user ids for the Twitter harvester.
2. Supporting easy query of stored data for analytical processing using MapReduce views.

3. Storing processed tweets for direct retrieval by the backend
4. Storing processed AURIN data for comparison in scenario analyses

Figure 9 shows on overview of the databases created in the CouchDB.

The ‘birthcountry’ and ‘homelang’ databases, in addition, stored processed AURIN data, while the ‘langcode’ database held information on converting language codes into language names. These three databases together supported data analyses that dived into the analytic scenario for the opportunity indicator. The ‘cost_text’, ‘housing_text’ and ‘transportation_text’ databases stored processed tweet texts related to topics of cost of living, housing and transportation, which were used to explore into the housing and transportation indicators and the cost of living in Melbourne. Further, the ‘housingprice’ database stored processed data related to housing prices in Melbourne and supported the scenario for the housing indicator. Finally, the ‘top_lat_long_live_hist’ database contained the latitude and longitude information used for map-based visualisation that supported the analytic scenario for the neighbourhood indicator.

5.2 MapReduce Views

Multiple MapReduce views were created to support easy query and retrieval of data for data analyses and web application development. They were primarily saved in the ‘raw_tweets’ database.

Figure 10 shows the list of views that were created in the ‘raw_tweets’ database.

Figure 9

Overview of Databases in the CouchDB

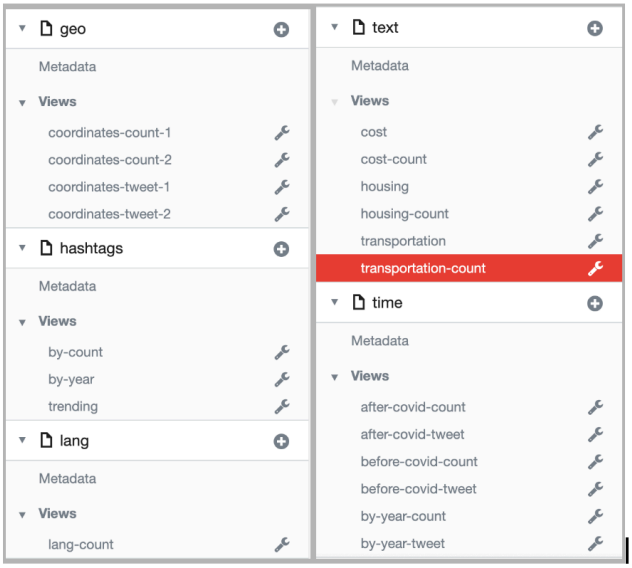
Databases

Database name | Create Database | {} JSON

Name	Size	# of Docs	Partitioned	Actions
birthcountry	9.7 KB	29	No	
cost_text	196.9 KB	7	No	
homelang	11.3 KB	34	No	
housing_text	2.1 MB	10	No	
housingprice	5.5 KB	10	No	
langcode	28.1 KB	125	No	
raw_tweets	1.6 GB	936870	No	
top_lat_long_live_hist	14.6 MB	96129	No	
transportation_text	2.7 MB	10	No	
user_list	0.5 MB	3584	No	

Figure 10

Overview of MapReduce Views in the 'raw_tweets' Database



Geo-related views mapped collected tweets by geo coordinates as the keys, with the tweet text (coordinates-tweet-1/2) or count (coordinates-count-1/2) as the values, and supported geo-based data analyses and map-based visualisations. Language related views mapped collected tweets by the languages they were made as the keys, with count as the values, and supported analyses into the opportunity indicator together with processed AURIN data. Text related and time related views, in addition, mapped collected tweets by the years they were made as the keys, with the tweet texts or count as the values. These views supported analyses concerning the housing and transportation indicators as well as the cost of living.

Other views, such as those related to hashtags or covid, while may not have directly contributed to the final analytic scenarios, still assisted with data exploration and creation of a coherent story for the current project.

6. Data Analyses

To develop analytic scenarios for the current project, count-based, text-based, and geo-based analyses were conducted.

6.1 Count-based Analyses

Count-based analyses directly exploited the MapReduce views created in the CouchDB. Analyses that belonged to this category included trending hashtags, top-tweeted languages other than English, and the trends of topic-related tweets.

6.1.1 Trending Hashtags

Hashtags used in tweets posted within the recent 14 days were firstly mapped, then reduced by their frequencies using the reduce option '_sum', and saved as a CouchDB view (hashtags/trending). After retrieving data in this view, top 10 of those hashtags were selected and visualised using a word cloud. The MapReduce function for trending hashtags can be found in Appendix C.

Though the hashtags by themselves did not bear any direct link to the analytic scenarios concerning liveability indicators, they provided a general overview of the specific themes or topics people living in Melbourne are interested in at the moment. If there were any liveability related topics of concern, they would be captured. In this sense, trending hashtags could provide a quick and general indication of whether any liveability related topics were of particular interests by the people in Melbourne at a particular time.

6.1.2 Top Tweeted Languages Other Than English

Languages used to make the tweets were mapped and reduced by their frequencies using the reduce option '_sum', and saved as a CouchDB view (lang/lang-count, Appendix C). Tweets made using languages that could not be identified by Twitter were omitted. After direct retrieval of data from this view, the percentage of tweets made in languages other than English was calculated, while the top 10 of those languages were visualised and compared with the analytic results from complimentary AURIN data, to support the scenario for the opportunity indicator.

6.1.3 Trends of Topic-related Tweets

Tweets tweeted about topics of housing, transportation, or cost of living for each year from 2014 to 2022, as well as the total number of tweets collected for each of those years, were mapped and reduced by their frequencies using the reduce option '_sum' and saved as CouchDB views ('text/cost-count', 'text/housing-count', and 'text/transportation-count' for topic related tweets and 'time/by-year-count' for total number of tweets, Appendix C). These data supported the calculation of percentage of topic-related tweets to the total number of tweets for each year from 2014 to 2022.

The percentages of topic related tweets were expected to reflect people's general attitudes toward those topics. It was proposed that the more people are bothered by those aspects of living, the more they would tend to tweet about it and thus the greater percentage of tweets about those topics there would be. The trends from 2014 to 2022, in addition, revealed how people's attitudes changed from year to year.

These count-based topic-related analyses were complemented with text-based topic-related sentiment analyses to tell a more coherent story. Together they supported the analytic scenarios for the housing and the transportation indicators.

6.2 Text-based Analyses

6.2.1 Topic-related Sentiment Analysis

Sentiment analyses on topic-related tweets were conducted to complement count-based topic-related trend analyses. Tweets related to topics of interest for each year from 2014 to 2022 were mapped and saved in the CouchDB views of 'text/cost', 'text/housing' and 'text/transportation' (Appendix C). Upon retrieval of data from those views, text pre-processing including removal of hashtags, punctuations, English stopwords, and any non-alphanumeric characters were applied to the topic-related tweets before they were joined together to form the topic-related corpuses on which sentiment analyses were performed.

The sentiment analyses were conducted using the Python TextBlob library which calculates the polarity of textual data (TextBlob, 2020). Polarity is a value that lies between -1 and 1, with -1 indicating a negative sentiment and 1 indicating a positive sentiment (TextBlob, 2020). Sentiment analyses on topic-related tweets hence helped to identify the general feelings of people living in Melbourne toward the topics in question. They supported the development of analytic scenarios on the housing and the transportation indicators, together with count-based topic-related trend analyses discussed previously.

6.2.2 Challenge and Refinement

During test implementation, the processing time for the sentiment analyses were found to be slow due to the massive amount of text pre-processing involved. To optimise the processing time, decision was made to separate the text pre-processing from the sentiment analyses. Specifically, the text pre-processing was designed to be pre-loaded, so it could be completed in advance and saved the updated topic-related corpuses into the CouchDB in one of the 'housing_text', 'transportation_text' and 'cost_text' databases. Sentiment analyses were then

performed on the already pre-processed texts directly retrieved from the corresponding topic-related databases.

In addition, the text pre-processing was designed to only update topic-related corpuses for the years 2018 to 2022, because the 2014 to 2017 Tweet data were solely from the provided historic data and hence are static. Only Tweet data from 2018 onwards were constantly added due to the running of the Twitter harvester. Consequently, every time the text pre-processing function was called to pre-load text pre-processing, it firstly deleted the previously saved topic-related corpuses for the years 2018 to 2002 in the topic-related databases. It then retrieved updated topic-related tweets to include the ones freshly collected, performed text pre-processing, and saved the updated topic-related corpuses back into the topic-related databases for query and retrieval by the sentiment analyses functions.

6.3 Geo-based Analyses

6.3.1 Mostly Checked-in Geographical Locations

Geographical locations checked-in as indicated by longitude and latitude coordinates information in the tweets were mapped and reduced by their frequencies using the reduce option '_sum', and saved as CouchDB views (geo/coordinates-count-1, geo/coordinates-count-2, Appendix C). After retrieving data from these views, the top 300 checked-in locations were extracted and used for map-based visualisations.

The top 300 checked-in geo-locations were expected to reflect the most popular places in Melbourne. It was proposed that these locations would mostly be parks, markets, shopping centres, restaurants etc to reflect that people in Melbourne live with proximity to locations that make local life interesting, so that the development of analytic scenario on neighbourhood could be supported.

6.3.2 Challenge and Refinement

As the loading of geo-location data was time consuming, the extracted top 300 locations were saved into the database of *top_lat_long_live_hist* in the CouchDB for direct retrieval of the backend. The latitudes and longitudes of the top 300 locations were mapped and saved as a CouchDB view (geoLocation/new-view) to support the retrieval of data. Since the visualisation layers in Mapbox GL JS took into consideration the count of each location, the reduce function was not necessary. The Map function for the view is documented in Appendix C.

6.4 Analyses of AURIN and Other Data

The collected and downloaded AURIN and other data were in the format of csv files. They were loaded, explored and analysed using the Python Numpy and Pandas libraries.

For the AURIN data on country of birth, only information for countries of birth of non-English speaking countries were retained. The total number of people who were born in a non-English speaking country were summed according to their countries of birth, and the percentages of the country totals to the total population of Melbourne were calculated. The country names, country totals and percentages were then saved into the 'birthcountry' database in the CouchDB for query and retrieval by the web application.

Similarly, for the AURIN data on languages spoken at home, only information for the languages other than English were retained. The total number of people speaking a non-English language at home were summed according to the language they speak, and the percentages of the language totals to the total population of Melbourne were calculated. The language names, language totals and percentages were then saved into the 'homelang' database in the CouchDB for query and retrieval by the web application.

For the housing price data collected from the ABS website, only information for Melbourne for the years 2014 to 2021 were retained. The percentage change of residential housing prices for those years were calculated and were saved into the 'housingprice' database in the CouchDB for query and retrieval by the web application.

7. Frontend Data Visualisation

7.1 Frontend

Frontend was built by React (React, 2022). React was selected as it provided easily accessible APIs for the frontend visualisation applications. Specifically, Highcharts was used for interactive graph-based visualisations, and Mapbox was chosen for map-based visualisations.

The frontend web page consisted of six sections: 'Now Trending', 'Opportunity', 'Housing', 'Transportation', 'Cost of Living' and 'Map'. Visualisations of analyses results for each section could be accessed by clicking on the interested topics at the Navigation bar on the left side of the page. An illustration of the web page can be found in Figure 11.

Figure 11

Homepage of the Web Page

7.1.1 Graphs

Highcharts provided a supported wrapper for React. Stacked column and pie charts were used for the opportunity related scenario. Trend and sentiment analyses for topic-related tweets were presented on the same graph. Bars and lines were used to show trends and sentiments of tweets respectively. These graphs supported the scenarios related to housing, transportation and cost of living.

Data used to create the graphs were JSON formatted by the backend, and were fetched using axios.

7.1.2 Maps

Geo-location data were displayed on a map using Mapbox GL JS, which allowed easy visualisation of GeoJSON data with highly customisable styling and complexity. The map contained components such as zoomControl, Scale Control, Marker, Popup and Cluster, for better user interaction. The map featured a Mapbox Dark style (Mapbox, n.d.), to highlight data layers on the map and reduce distraction from the map background. The original data retrieved from the CouchDB views were converted into GeoJSON format in the backend, while the frontend fetched the GeoJSON-formatted data from the backend for visualisation.

The tweet data were displayed with two layers: the cluster and the circle layers, to provide an overview of the tweet density at the top 300 extracted locations. Radio groups were used for users to select a layer to explore. Markers were used to show popularly checked-in places in Melbourne. By hovering over the markers, detailed information of that place would be displayed.

7.2 Visualisation of Analytic Scenarios

7.2.1 Now Trending

Figure 12 demonstrates a wordcloud of trending hashtags in the 14 days preceding 15 May 2022. The wordcloud shows that election related topics are currently trending ('auspol', 'ausvotes'), which was expected as the election is approaching. The increasing counts of 'victraffic', in addition, indicates it's been an eventful fortnight in terms of traffic, as this hashtag is generally associated with road closures, roadworks and accidents.

In general, in can be seen that this analysis does pick up topics of common interests and concerns, and therefore can be expected to capture liveability related topics should they arise.

7.2.2 Opportunity

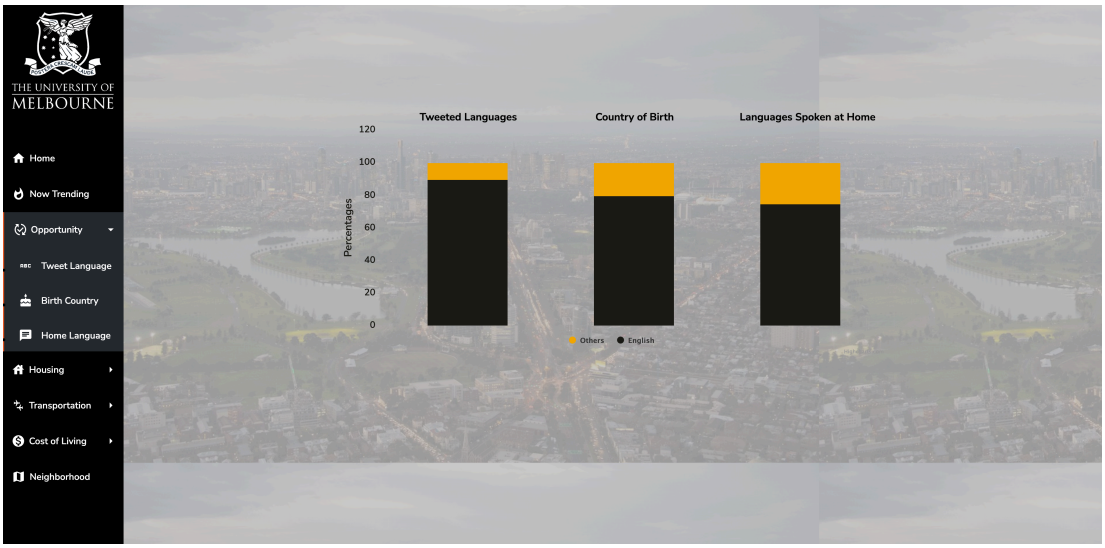
Stacked column charts and pie charts were employed to illustrate the analytic scenario for the opportunity indicator. The stacked column charts display the proportion of tweets posted in languages other than English, the proportion of population in Melbourne that were born in a non-English speaking country and the proportion of population in Melbourne that speak a language other than English at home (Figure 13).

Figure 12

Now Trending

Figure 13

Stacked Column Charts for the Opportunity Indicator

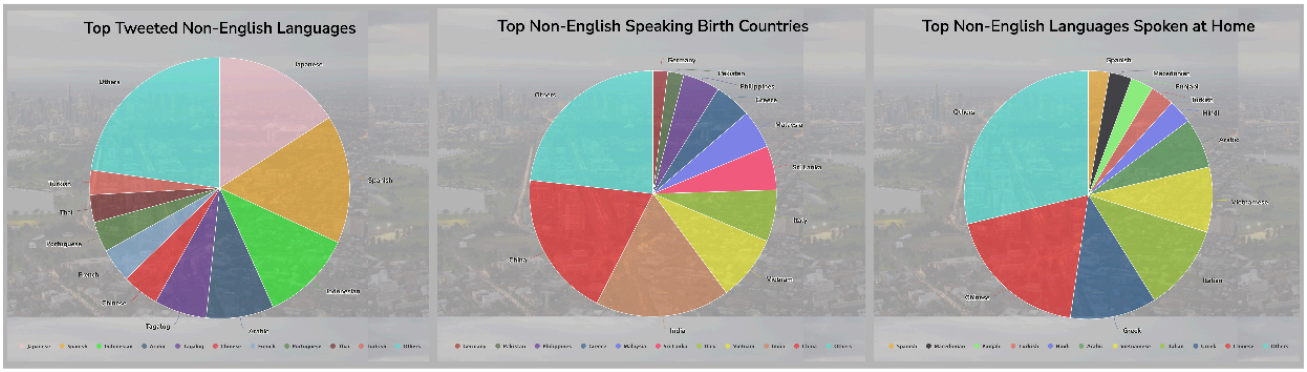


It can be seen that approximately 1 in 5 people in Melbourne was born in a non-English speaking country, more than 1 in 4 people in Melbourne speak a language other than English at home, while roughly 1 in 10 tweets posted in Melbourne used a language other than English, as of 15 May, 2022.

In addition, pie charts illustrate in detail top tweeted languages other than English, top non-English speaking birth countries and top non-English languages spoken at home. Figure 14 shows an example generated on 15 May 2022.

Figure 14

Pie Charts for the Opportunity Indicator



It is clear that people living in Melbourne were originally born in a wide variety of countries and speak a diversity of languages. In addition, parallels can be drawn between the three pie charts, suggesting culturally and linguistically diverse people in Melbourne tend to happily present themselves on the online social media platforms with their diversity.

These stacked column and pie charts together demonstrate that Melbourne is indeed a multicultural city in which people from all over the world live and earn a living. The multiculturalism of Melbourne provides solid evidence that this city is inclusive, welcoming and celebrates diversity, therefore, opportunistic indeed.

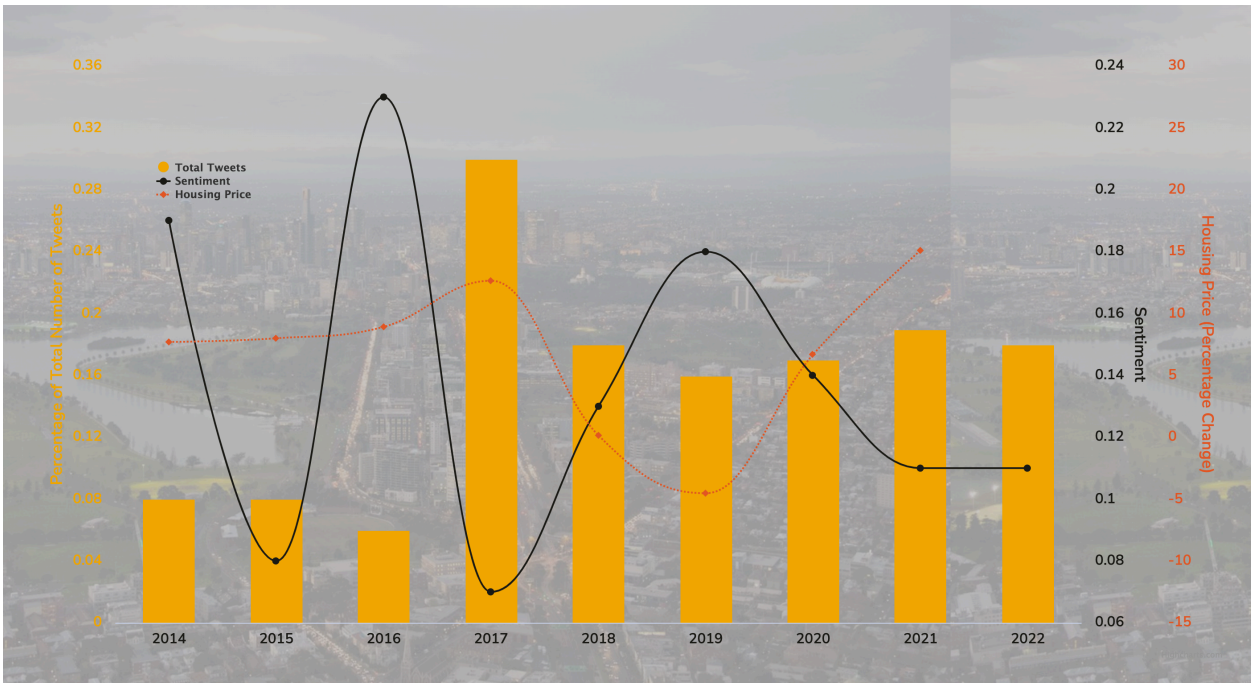
7.2.3 Housing

The analytic scenario for housing was visualised with a combination of lines and column graphs. An example generated on 15 May 2022 can be seen below (Figure 15).

The percentages of housing-related tweets for the years 2014 to 2022 are represented as the columns in the graph, while the sentiment scores for those tweets over those years are represented with the black solid line. In addition, the dotted orange line represents the percentage change in housing price for Melbourne from 2014 to 2021.

Figure 15

Analytic Graph for the Housing Indicator



Generally speaking, the percentages of housing-related tweets appear to be negatively correlated with the sentiment scores for those tweets, supporting the assumption that people tend to tweet more about the topics they are concerned with. This is most evident for the years 2016 and 2017. As can be seen, the most positive sentiment score across the 9 years happened in the year 2016, and it was associated with the least percentage of housing-related tweets across the 9 years. On the other hand, 2017 witnessed a dramatic increase in the percentage of housing-related tweets, while the sentiment score reached the lowest among the 9 years.

Relating the amount of tweets for a given topic people make, to the amount of concern people have for that topic enables us to gain further insights into the housing aspect of living in Melbourne. For example, there was an increase in the percentages of housing-related tweets from the year 2017 onwards, as compared to the percentages in years before 2017, suggesting raised concern towards housing since 2017. This might be due to the fluctuation of housing prices in Melbourne since 2017, as indicated by the dotted orange line. Because, intuitively, people tend to prefer a steadily growing housing market, while fluctuations tend to cause concerns. This fluctuation in housing prices, and the amplified concerns it was associated with, may, to some extent, have contributed to the slip of Melbourne's liveability ranking after 2017.

By and large, however, the sentiment scores towards housing were all positive across the 9 years, suggesting that housing wasn't a pressing issue that was significantly concerned by people living in Melbourne. From this perspective, Melbourne is performing satisfactorily for the housing indicator for liveability.

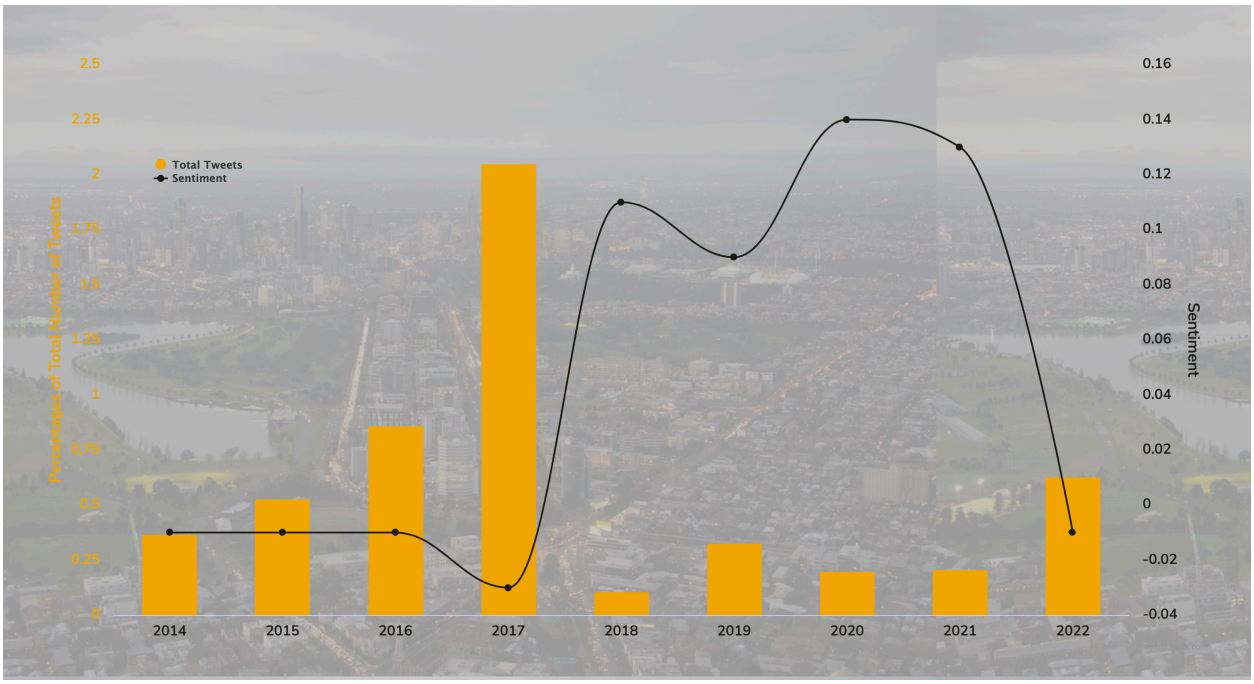
7.2.4 Transportation

The analytic scenario for transportation was visualised with a dual axes line and column graph. An example generated on 14 May 2022 can be seen below (Figure 16).

The percentages of transportation-related tweets for the years 2014 to 2022 are represented as the columns in the graph, while the sentiment scores for those tweets over those years are represented with the line.

Figure 16

Analytic Graph for the Transportation Indicator



According to this graph, people in Melbourne generally hold a negative attitude towards transportation in Melbourne, as suggested by the negative sentiment scores for most of the years since 2014. The only years when positive sentiment scores were observed were years 2018 to 2021. While in the years 2020 and 2021 the sentiment scores soared up high. However, these were the years when Melbourne was mostly in a lockdown due to the COVID-19 pandemic. The improved sentiment scores, therefore, may bear no relation to improvements to the transportation in Melbourne, but be more likely related to the fact that there was not much transportation during the lockdowns.

The liveability of Melbourne associated with the transportation indicator, therefore, isn't very satisfactory according to the opinions of its people.

As a sidenote, similar to the trend and sentiment for housing-related tweets, a negative correlation can be obviously perceived between the percentages of transportation-related tweets and the sentiment scores for those tweets. Once again, the assumption on the relation between the amount of tweets people make for a given topic and the amount of concern they have for that topic is supported.

7.2.5 Cost of Living

The analytic scenario for cost of living was visualised with a dual axes line and column graph. An example generated on 14 May 2022 can be seen below (Figure 17).

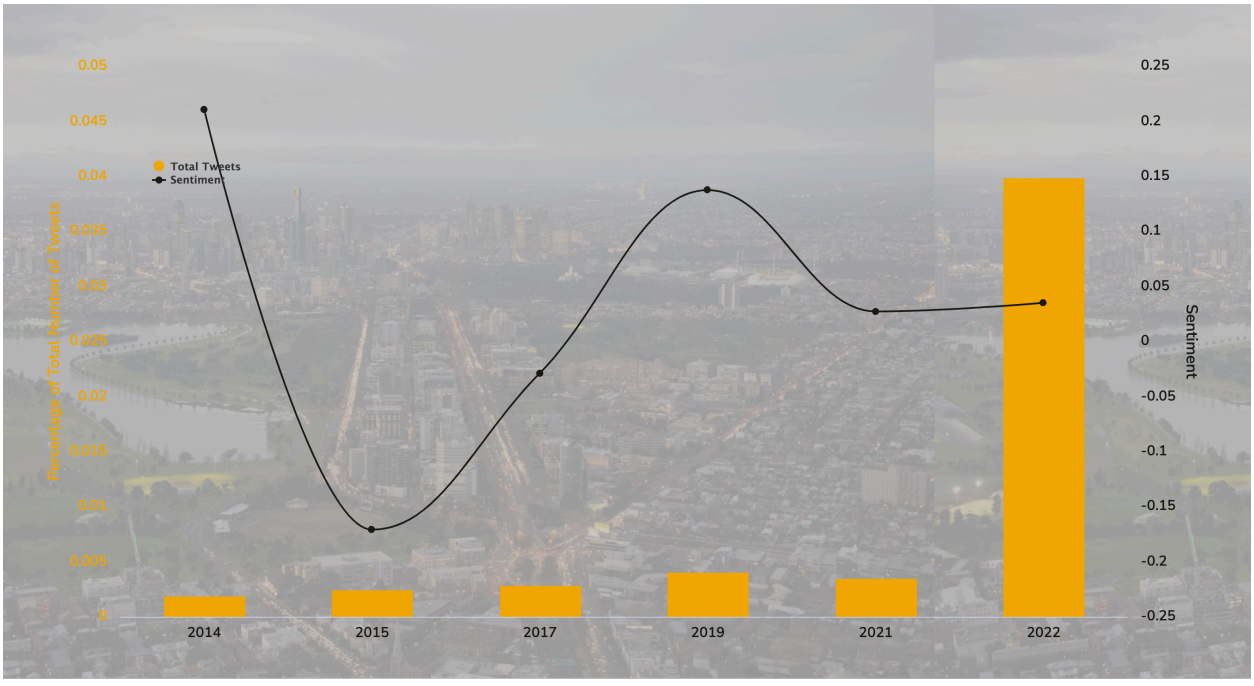
The percentages of cost-of-living-related tweets for the years 2014 to 2022 are represented as the columns in the graph, while the sentiment scores for those tweets over those years are represented with the line.

Although there isn't a cost-of-living indicator for liveability, cost of living was included in the analytic scenarios as we believe it is closely related to liveability.

In terms of the trend for the percentages of the cost-of-living-related tweets, there is a drastic increase in 2022, suggesting a recent raised concern about cost of living. The trend perfectly reflects the reality as the petrol and the grocery prices skyrocketed recently. It is surprising that the sentiment score isn't heading down to reflect this distress. A possible explanation might be that as the federal election is approaching, the leaders are making cost-of-living-related promises, which may have slightly driven up the sentiment score. However, it remains doubtful as to what extent those promises would be realised. From this perspective, Melbourne isn't currently doing so well for the cost-of-living aspect of liveability.

Figure 17

Analytic Graph for Cost of Living

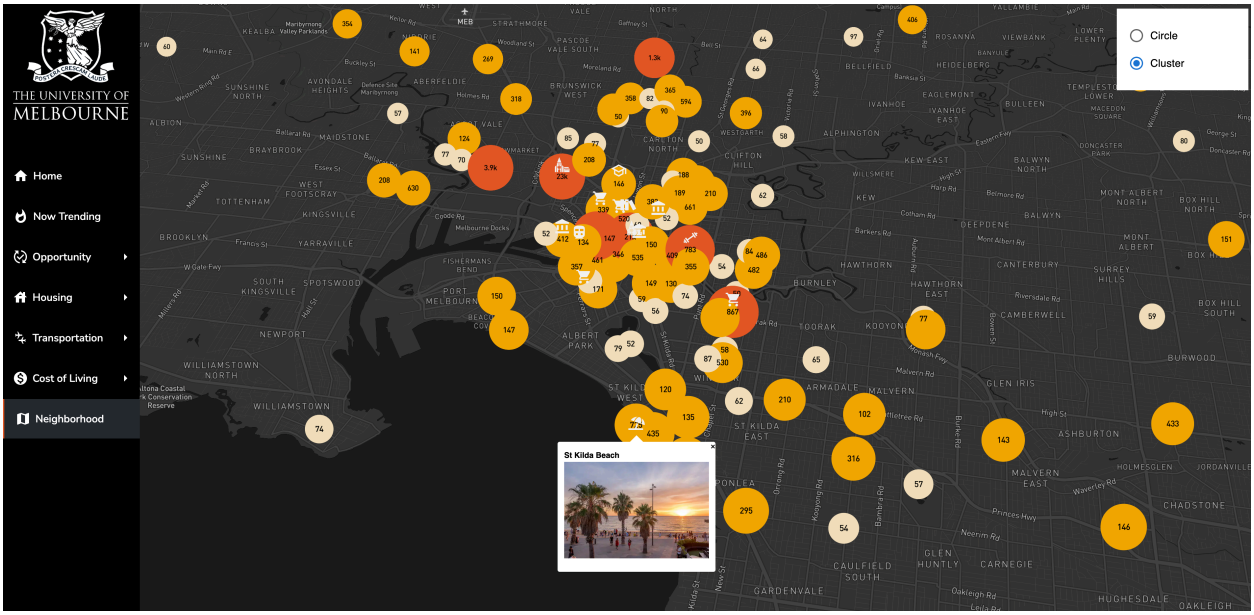


7.2.6 Neighbourhood

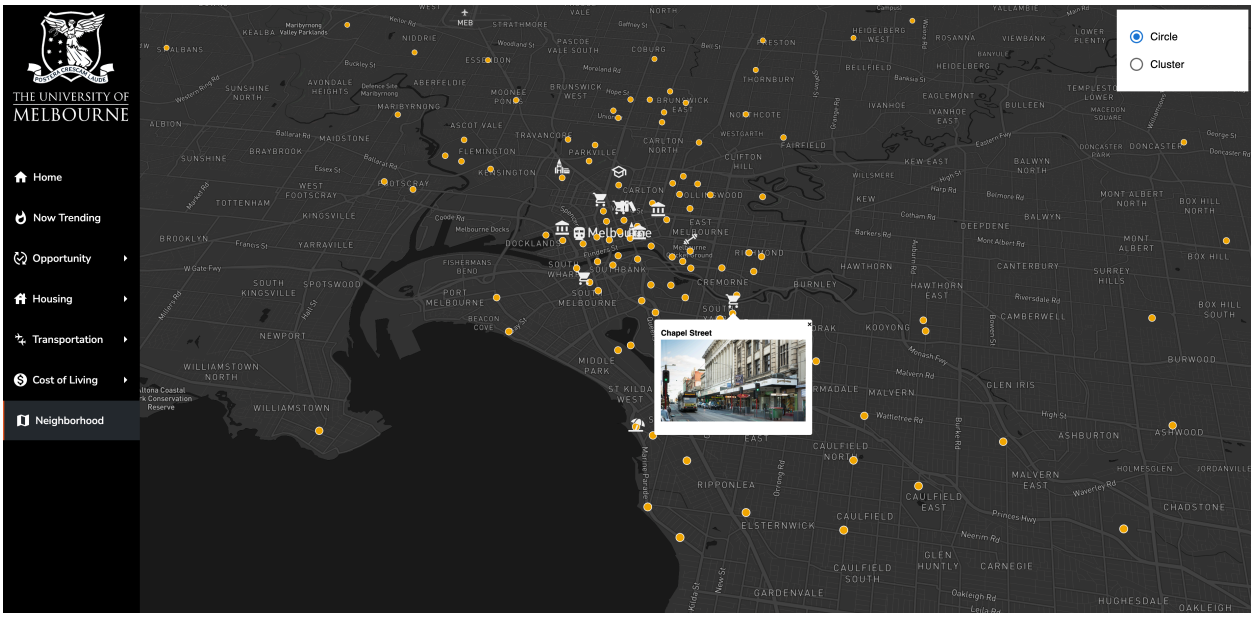
The analytic scenario for the neighbourhood indicator was visualised by a map with different layers. An example generated on 16 May 2022 can be seen below (Figure 20).

Figure 20

Analytic Map for the Neighbourhood Indicator (Cluster Layer)



Analytic Map for the Neighbourhood Indicator (Circle Layer)



The two different layers, the cluster layer, and the circle layer, were used to illustrate the density for the top 300 extracted tweeted locations. The default layer is the cluster layer. Clusters in darker colours with larger sizes represent higher densities of tweeted locations. Clusters in light yellow with small sizes, solid yellow with medium sizes and solid orange with large sizes present clusters with less than 100 tweets, from 100 to 750 tweets and above 750 tweets respectively. Icons were used to label some of the dense clusters. It can be seen that many of the dense clusters are parks, gyms, shopping centres, historic sites, beaches, stadiums and libraries, such as St Kilda beach, Chapel St and Burke St shopping districts. The clusters of tweeted locations illustrate the typical life of people living in Melbourne. For example, they enjoy going shopping, going to the beach and watching sport games. These reflect that people in Melbourne live with proximity to locations that make local life interesting.

It is interesting to see that Ss Peter & Paul Ukrainian Catholic Cathedral has the densest cluster with around 23,000 tweets. The surge of this tweet location suggests that going to church may be part of many Melbournians' lives. Indeed, it might also be related to the Russian invasion of Ukraine, so people go and join prayer services to pray for peace and pray for Ukraine.

In addition, the circle layer shows that tweet locations are spread all over Melbourne, illustrating that there are many places that people could explore in the Greater Melbourne. They also suggest that people in Melbourne love going around to explore different places.

The two layers together, clearly demonstrate that Melbourne has a diversified neighbourhood with activities, services, and shops for people to explore and enjoy, and they spread out all over the Greater Melbourne. Therefore, Melbourne is indeed performing well in terms of the neighbourhood indicator according to tweet locations.

8. Project Management

The project was a collaborative work by all members in the team. A GitHub repository with multiple branches was created initially for all members to work on. Upon the completion of the project, sub-branches were merged into the main. The link to the GitHub repository can be found below:

https://github.com/CCC-Team5/CCC_A2/tree/main

Each of the team members was responsible for a separate set of tasks while also assisted with others' tasks wherever they could. Table 1 outlines the key responsibilities of each team member.

A variety of communication and project management tools were leveraged during the implementation period, including WeChat for instant messaging, Zoom for teleconferences and GitHub project Kanban for task tracking.

The team communicated with each other in the team's WeChat chat group, especially when there were questions or issues that required immediate attention or response. In addition, the team met virtually using Zoom on a weekly basis to update each other with the progresses made, to demo works done and to make plans for the next week. Backlogs were lodged and assigned to team members on the GitHub project Kanban using task cards and were dragged long the columns as the project progressed. A screenshot of the GitHub project Kanban is shown in Figure 20. These tools significantly smoothed the workflow and the management of the project, especially when two of the team members were overseas.

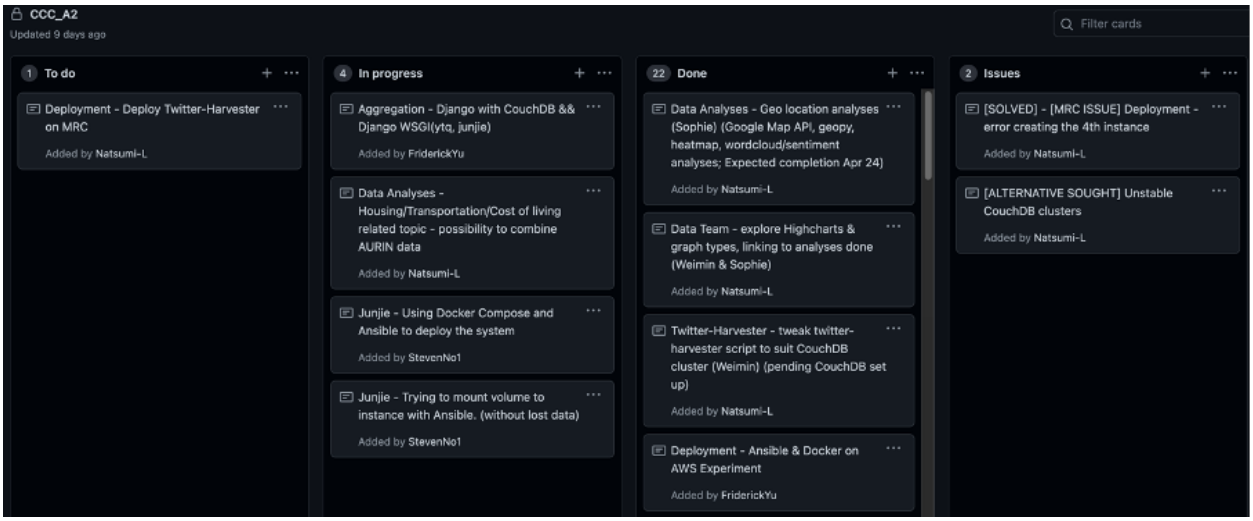
Table 1

Key Responsibilities of Team Members

Team Member	Key Responsibiities
Xinhao Chen	Deployment of the whole system, including the cloud infrastructure, the CouchDB, the Twitter harvester and the web application; Data Visualisation and implementation of the frontend
Weimin Ouyang	Implementation of the Twitter harvester; AURIN data mining; Managing the CouchDB database and data processing using MapReduce views; Analysing tweet and AURIN data
Tianqi Yu	Dockerising Python Django Web Application; Implementation of middleware and connection to backend; Implementation of the backend
Junjie Xia	Dockerising Python Django Web Application; Implementation of middleware and connection to backend; Implementation of the backend
Yuling Zheng	Analysing tweet and AURIN data; Data Visualisation and implementation of the frontend

Figure 20

Screenshot of the GitHub project Kanban



9. User Guide

9.1 Installation and Deployment Guide

The whole system can be deployed using ansible and was designed to be operated on the MRC. Therefore, the follow prerequisites need to be satisfied in order to deploy and interact with the system:

1. Having an eligible MRC account.
2. Having a Linux terminal.
3. Having Ansible installed by input command in the Linux terminal: `python -m pip install --user ansible`.
4. Connecting to Unimelb using the Cisco VPN (if you are using a unimelb account).

Once logged into the MRC dashboard successfully, the installation and deployment can be done in the following steps:

1. Download the 'OpenStack RC file' from the dashboard and rename it to 'grp5-openrc.sh'. A password is required to access the MRC using the OpenStack API. It is recommended to go to the account settings and reset the password, and then copy and save the new password somewhere as users will be asked to input the password several times during the execution of the script.

2. Create a new SSH key pair in the MRC dashboard. A '.pem' file would then be automatically downloaded. Rename the file to 'grp5_key.pem' and put it together with the 'grp5-openrc.sh' downloaded in Step 1 in the CCC_A2/deployment/serverfiles.
3. Open the terminal in the 'CCC_A2/deployment' folder. We've created a main script 'exec-grp5-deploy.sh' for managing the whole process of deployment.
4. Execute the command 'sh exec-grp5-deploy.sh' in the Linux terminal. It will first ask for the sudo password, then ask for the password generated in Step 1.
5. Input the OpenStack password when necessary. After the whole execution flow is completed, the whole system (the twitter harvester and the web application) should be up and running.
6. To interact with the system, find the IP address of the master instance in the 'hosts' file under the 'Webserver' category. Enter the IP address at the port 3000 in the web browser.

9.2 Visualisation Frontend Guide

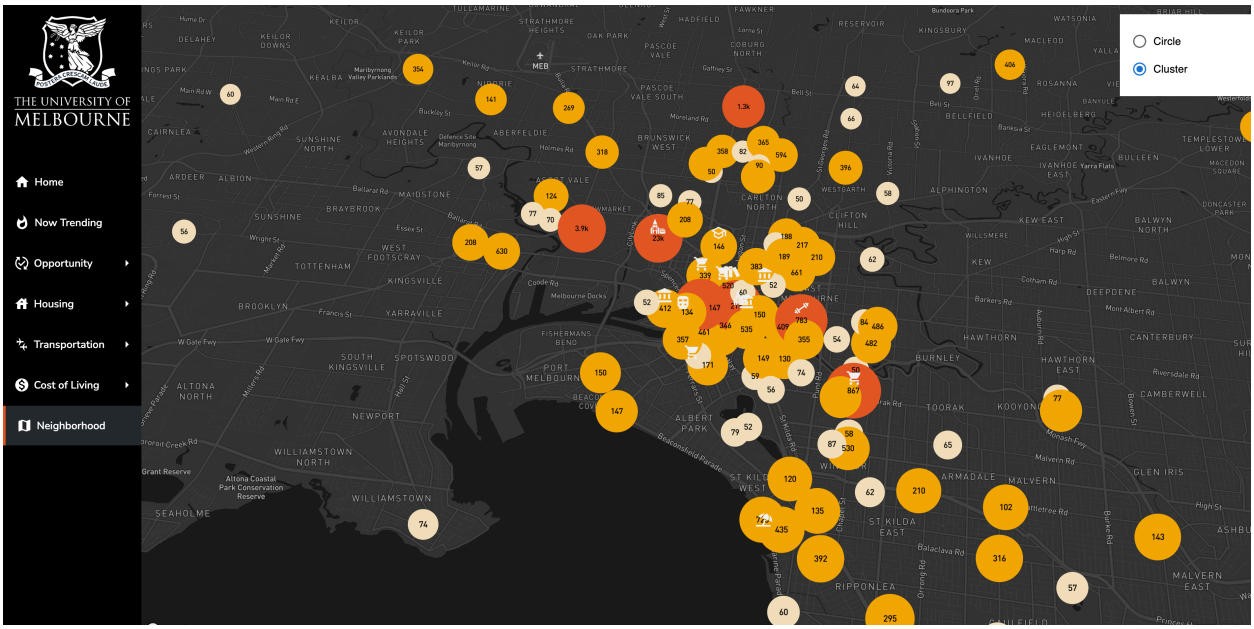
Landing at the home page of the web application, users can navigate through the website using the navigation panel on the left. The navigation panel lists the subpages that contain analytic scenarios supported by the web application.

Clicking at buttons on the navigation panel, users can explore the trending hashtags, the trend and sentiment analyses for the cost-of-living aspect of living in Melbourne, and the analytic scenarios for the opportunity, housing, transportation, and neighbourhood indicators of liveability on the subpages. An example of this is the user clicking on the 'Neighbourhood' button and they will be taken to the map illustrating the neighbourhood aspect of liveability of Melbourne as seen in Figure 21.

On each of the subpage, users can interact with the visualisations by hovering over aspects of interests. On the neighbourhood page in particular, users can further zoom in and out of the map to explore more closely the frequently checked-in places in Melbourne or click on the icons for extra information for selected representative spots.

Figure 21

Neighbourhood Page



References

AARP. (n.d.) *Scoring*. https://livabilityindex.aarp.org/scoring

Apache CouchDB. (2022). Apache CouchDB 3.2.0 documentation. https://docs.couchdb.org/en/stable/.

Apache CouchDB. (n.d.). *CouchDB relax*. https://couchdb.apache.org/

Django. (n.d.). *Django overview.* https://www.djangoproject.com/start/overview/

Docker. (n.d.). *Docker project website*. https://www.docker.com

Eddie, R., Preiss, B., & Estcourt, D. (2021, June 9). Melbourne drops to 8th on world liveability index. The Age. https://www.theage.com.au/national/victoria/melbourne-drops-to-8th-on-world-liveability-index-20210609-p57zcs.html

Global Victoria. (n.d.) *One of the world's most liveable cities.* https://global.vic.gov.au/victorias-capabilities/why-melbourne/one-of-the-worlds-most-liveable-cities#:~:text=The%20Economist%20Intelligence%20Unit's%202017,the%20index%20began%20in%202002.

Highcharts. (2021). *GitHub repository.* https://github.com/highcharts/highcharts-react

Mapbox. (n.d.). *Mapbox Dark*. https://www.mapbox.com/maps/dark

Melbourne Research Cloud. (n.d.). *Melbourne Research Cloud documentation.* https://docs.cloud.unimelb.edu.au/

Openstack. (n.d.). Open-source cloud computing infrastructure - Openstack. https://www.openstack.org/

React. (2022). *React - A JavaScript library for building user interfaces.* https://reactjs.org/

React-map-gl. (n.d.) *REACT-MAP-GL.* https://visgl.github.io/react-map-gl/

Red Hat Ansible. (n.d.). *Ansible project website*. https://www.ansible.com/

TextBlob. (2020). *TextBlob: Simplified Text Processing.* https://textblob.readthedocs.io/en/dev/

Tweepy. (n.d.). *Tweepy project website*. http://www.tweepy.org/

Twitter. (2022). *Getting started with the Twitter API*. https://developer.twitter.com/en/docs/twitter-api

Appendix A

Dockerfiles

Dockerfile for React Frontend

```
# pull the base image
FROM node:lts-alpine

# make the 'app' folder the current working directory
WORKDIR /app

# copy both 'package.json' and 'package-lock.json' (if available)
COPY package*.json ./

# install project dependencies
RUN npm install

# copy project files and folders to the current working directory (i.e. 'app' folder)
COPY . .

EXPOSE 3000

CMD [ "npm", "start"]
```

Dockerfile for Twitter Harvester

```
FROM python:latest

WORKDIR /usr/src/app
COPY . /usr/src/app

# set python environment variable
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

COPY requirements.txt ./
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

CMD [ "python", "main.py"]
```

Dockerfile for Django Backend

```
# get python3 environment
FROM python:3
# set domain name
# http_prox haven not setted yet, thererfore, we do not include this commond.
#ENV http_proxy http://wwwproxy.unimelb.edu.au:8000
#ENV https_proxy http://wwwproxy.unimelb.edu.au:8000

# set python environment variable
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
# create code directory and set it to work directory
RUN mkdir /code
WORKDIR /code

COPY requirements.txt /code/
RUN pip install --upgrade pip
RUN pip install -r requirements.txt
ADD . /code/

EXPOSE 8000

CMD [ "python3","manage.py", "runserver", "0.0.0.0:8000"]
```

Appendix B

Summary of Running Docker Containers on the Master Instance

```
ubuntu@server0:~$ sudo -s
root@server0:/home/ubuntu# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS                                                                                  NAMES
3d4225e3e33f   tweet_backend      "python3 manage.py r…"   4 minutes ago   Up 4 minutes   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp                                              ccc_a2_backend_container_1
0824b84fa97b   tweet_frontend     "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp                                              ccc_a2_frontend_container_1
5431081b2668   twitter_harv       "python main.py"         7 minutes ago   Up 7 minutes                                                                                          harvester_container
d542aae71d12   ibmcom/couchdb3    "/docker-entrypoint.…"   31 hours ago    Up 31 hours    0.0.0.0:4369->4369/tcp, 0.0.0.0:5984->5984/tcp, 0.0.0.0:9100-9200->9100-9200/tcp      couchdb
```

Appendix C

MapReduce functions used for CouchDB views

```
// 'hashtags/trending'
function (doc) {
    if(doc.tweet.entities.hashtags.length > 0 && Date.parse(doc.tweet.created_at) >= new Date().setDate(new Date().getDate() - 14)){
        for (var i = 0; i < doc.tweet.entities.hashtags.length; i++){
          if (doc.tweet.entities.hashtags[i].tag){
            emit(doc.tweet.entities.hashtags[i].tag, 1);
          }
          else {
            emit(doc.tweet.entities.hashtags[i].text, 1)
          }
        }
    }
}
// reduce: _sum


// 'hashtags/by-year'
function (doc) {
    if(doc.tweet.entities.hashtags.length > 0){
        for (var i = 0; i < doc.tweet.entities.hashtags.length; i++){
          if (doc.tweet.entities.hashtags[i].tag){
            emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.entities.hashtags[i].tag);
          }
          else {
            emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.entities.hashtags[i].text)
          }
        }
    }
}


// 'lang/lang-count'
function (doc) {
    if (doc.tweet.lang && doc.tweet.lang != 'und') {
      emit(doc.tweet.lang, 1);
    }
}
// reduce: _sum


// 'geo/coordinates-count-1'
// stream & historic format
function (doc) {
    if (doc.tweet.coordinates.coordinates) {
        emit(doc.tweet.coordinates.coordinates, 1);
    }
}
// reduce: _sum


// 'geo/coordinates-count-2'
// timeline format
function (doc) {
    if (doc.tweet.geo.coordinates.coordinates) {
        emit(doc.tweet.geo.coordinates.coordinates, 1);
    }
}
// reduce: _sum
```

```
// 'geo/coordinates-tweet-1'
// stream & historic format
function (doc) {
    if (doc.tweet.coordinates.coordinates) {
        emit(doc.tweet.coordinates.coordinates, doc.tweet.text);
    }
}


// 'geo/coordinates-tweet-2'
// timeline format
function (doc) {
    if (doc.tweet.geo.coordinates.coordinates) {
        emit(doc.tweet.geo.coordinates.coordinates, doc.tweet.text);
    }
}


// 'time/before-covid-tweet'
function (doc) {
    if(Date.parse(doc.tweet.created_at) <= Date.parse("Apr 01 00:00:00 +0000 2020")){
        emit(doc._id, doc.tweet.text);
    }
}


// 'time/before-covid-count'
function (doc) {
    if(Date.parse(doc.tweet.created_at) <= Date.parse("Apr 01 00:00:00 +0000 2020")){
        emit(doc._id, 1);
    }
}
// reduce: _sum


// 'time/after-covid-tweet'
function (doc) {
    if(Date.parse(doc.tweet.created_at) > Date.parse("Apr 01 00:00:00 +0000 2020")){
        emit(doc._id, doc.tweet.text);
    }
}


// 'time/after-covid-count'
function (doc) {
    if(Date.parse(doc.tweet.created_at) > Date.parse("Apr 01 00:00:00 +0000 2020")){
        emit(doc._id, 1);
    }
}
// reduce: _sum


// 'time/by-year-tweet'
function (doc) {
    if(doc.tweet.created_at){
        emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.text);
    }
}
```

```
// 'time/by-year-count'
function (doc) {
    if(doc.tweet.created_at){
        emit(new Date(doc.tweet.created_at).getFullYear(), 1);
    }
}
// reduce: _sum


// 'text/housing'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/housing|house price|\brent|property|real estate/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.text);
    }
  }


// 'text/housing-count'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/housing|house price|\brent|property|real estate/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), 1);
    }
  }
// reduce: _sum


// 'text/transportation'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/transport|traffic|roadwork/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.text);
    }
  }


// 'text/transportation-count'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/transport|traffic|roadwork/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), 1);
    }
  }
// reduce: _sum


// 'text/cost'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/cost of living|petrol price|grocery price|utility bill/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), doc.tweet.text);
    }
  }


// 'text/cost-count'
function(doc) {
    if (doc.tweet.text.toLowerCase().match(/cost of living|petrol price|grocery price|utility bill/)) {
      emit(new Date(doc.tweet.created_at).getFullYear(), 1);
    }
  }
// reduce: _sum
```