Cluster and Cloud Computing

Assignment 2

Melbourne Suburb Data Analyse Application

GROUP 6

Haoyang Cui - 886794

Xin Wu - 947753

Dongming Li - 1002971

Ziyue Wang - 1014037

Mayan Agarwal - 973957

Introduction

With a huge amount of data inside, Twitter is a very interesting and rich data source for many different analyses. In this project, we built an entire system from data-harvest, data analysing to visualisation and dynamic deployment to display some interesting scenarios and results we found from tweet data. These scenarios includes:

- Find out recent hot words (e.g. stayhome, socialdistancing) on Twitter, and further explore people's sentiment when they mention these words, and how these hotword-related tweets distributed on Map
- Find out people's sentiment in each suburb, how many people have positive attitude in their tweets and how many are negative
- Which suburbs are more 'closed' with one suburb? That is, the more the number of users tweet on both suburbs, the more closed these two suburbs will be. By this. We want to explore the normal living regions (mostly includes several suburbs) of people in each suburb.
- Explore many suburb related information from both Aurin data and twitter data, including unemployment rate, middle income, number of non-English tweets, and so on.
- Users could also get some temporal features of people sending tweets, e.g. when people send the most tweets in one day.

The entire system is organised with these components/parts as follows:

Component	Description
Data Harvest	Implementation of scripts with tweepy Stream and Search api to collect relevant data from Twitter.
Data Analysis	Scripts and Map-Reduce function for all data analysis, including analysing original tweet data, sentiment analysis via NLP library and formatting data results from Aurin.
Backend Application	Including CouchDB and Spring Backend, implemented Rest API for communication
Middleware Redis Cluster	As we found the querying couchdb speed can be quite slow, A redis cluster as middleware to alleviate burdens on couchdb.
Frontend Application	Visualisation of multiple scenarios and components, mainly includes a functional Map view component and Chart View component. Nginx is used to listen on port 80 for the frontend and pass backend requests to port 8080.
Ansible Deployment	The deployment script for configuring and building systems with one command. Including parts like configuring instances, managing docker, CouchDB clustering and executing Backend & Frontend components.

Other Components	We also applied some other components for this project's completeness. E.g. We also deployed some components for instance status monitoring and container status monitoring.

The relevant links of Github repository, Video demo and System link are listed below:

GitHub: https://github.com/HaoyangCui0830/CCCProject2

Demo:

System Architecture Design

Ansible Deployment

Key Scenarios Demo

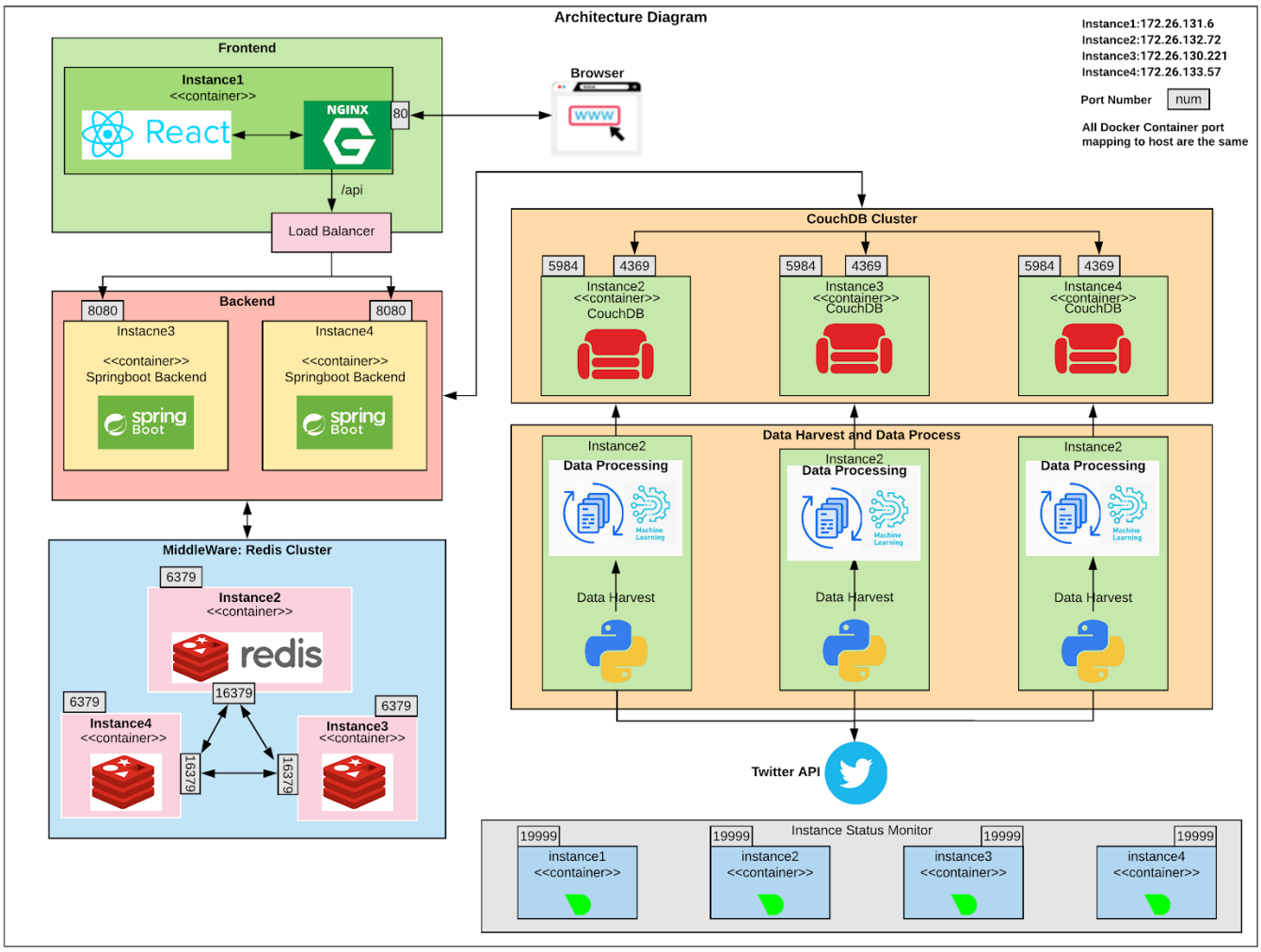
Home Page: http://172.26.131.6

The responsibilities of all team members are as follows:

Name	Responsibility
Haoyang Cui	• Implementation of Data Harvest and Data Handling Scripts • Implementation of MapReduce functions • Deployment script of all CouchDB related components
Xin Wu	• Implement Backend and auto generated API documentation by Swagger module • Configure cluster couchdb and cluster redis; • Backend load balance by Nginx; • Optimization query speed by adding Redis middleware; Partial Ansible script.
Dongming Li	• Implemented frontend application which includes using google map api to visualise data on the map, • handling data format from the backend • using rechart to display data in charts. Help with Nginx configuration.
Ziyue Wang	• Data handling work for aurin data
Mayan Agarwal	• Implementation of some frontend application features and functions

System General Design

The overall system architecture diagram looks like below:



Explanation

Here are some explanations of system architecture and components design:

Frontend & Nginx:

We use Nginx as a HTTP server and store those frontend static resources to provide frontend functionalities. The Nginx server listens to port 80 to handle requests from the browser(client). Nginx will pass all backend requests starting with /api/ to one of those two upstream backend servers by the configuration of proxy_pass and use Round Robin load balance policy to assign which backend processes the request.

We use React to build up our frontend as a single page application(SPA), so to provide routing between components, we use BrowserRouter to parse the url. And to avoid the hassle caused by unnecessary pass state and data through intermediary components, we use React Context to provide a central and global store.

Backend:

Nginx will assign requests for a specific backend by loading balance strategy. When a backend gets a request, it will validate the request first and process the request if it is legal, abandon it if it is illegal. Backend query redis first and if the target data in redis then quickly response to nginx and nginx sends the data to the browser. If target data is not in redis, then query couchDB and response the data besides, store the data into redis in case next time query.

Middleware Redis Cluster:

Lots of query requests are repeated and CouchDB responds to the same data, which let CouchDB do the same query and waste computing resources. Redis is an in-memory database and the speed is overwhelming. Middleware redis cluster can help couchDB Reduce the burden and improve response speed.

Data Harvest&Processing:

Data harvester collects data from Twitter API(both Search API and Stream API) and processor will further complete sentiment analysing using NLP library and Geo info recognising.

CouchDB Cluster:

Handled data will be written into CouchDB cluster with MapReduce views, backend could request relevant data from different views. We clustered three different instances as one couchDB cluster. In our test, the DB read/write performances are improved a lot through clustering.

Instance Status Monitor:

We deployed netData for monitoring instance status on all instances, including usage of CPUs, Disk and network. This could give us clear views about any issue on Server, and provide an efficient visualisation of workload in our system

System Components & Detailed Design

Ansible Deployment

The System used ansible to automatically deploy all required components with one command. The detailed ansible playbook structure looks like below:

```
#Playbook Structure
.
├── openrc.sh
├── couchdb_cluster.sh
├── couchdb_cluster.yaml
├── set_environment.sh
├── set_environment.yaml
├── xinwu-key.pem
├── host_vars
│   ├── nectar.yaml
│   └── remote.yaml
├── DeployAllApplications.sh
├── inventory
│   └── inventory.ini
├── DeployAllApplications.yaml
├── roles
│   ├── openstack-common
│   │   └── tasks
│   │       └── main.yaml
│   ├── openstack-instance
│   │   └── tasks
│   │       └── main.yaml
│   ├── openstack-security-group
│   │   └── tasks
│   │       └── main.yaml
│   ├── openstack-volume
│   │   └── tasks
│   │       └── main.yaml
│   ├── couchdb-subnode1
│   │   └── tasks
│   │       ├── main.yaml
│   │       └── subnode1.sh
│   ├── couchdb-subnode2
│   │   └── tasks
│   │       ├── main.yaml
│   │       └── subnode2.sh
│   ├── install-docker
│   │   └── tasks
│   │       └── main.yaml
│   ├── mount-volumes
│   │   ├── defaults
│   │   │   └── main.yaml
│   │   └── tasks
│   │       └── main.yaml
│   ├── couchdb-masternode
│   │   └── tasks
│   │       ├── masternode.sh
│   │       └── main.yaml
│   ├── set-proxy
│   │   └── tasks
```

```
│   │       └── main.yaml
│   ├── MapReduce_Setting
│   │   └── tasks
│   │       ├── main.yaml
│   │       └── setCouchDB.sh
│   ├── backend
│   │   └── tasks
│   │       ├── backendDeploy.sh
│   │       └── main.yaml
│   ├── netdata-instance-visualization
│   │   └── tasks
│   │       ├── main.yaml
│   │       └── netdataDeploy.sh
│   ├── nginx-frontend
│   │   └── tasks
│   │       ├── nginx.conf
│   │       ├── main.yaml
│   │       └── nginx_frontend.sh
│   └── redis-cluster
│       └── tasks
│           ├── main.yaml
│           ├── redis.conf
│           └── redisScript.sh
├── README.md
├── create_VMs.sh
├── create_VMs.yaml
└── run.sh
```

To execute it, you can simply run the following commands on the terminal after connecting to the Uni VPN.

```
./run.sh
```

The detailed **steps/processes** that the playbook will execute are as follows:

- **Define Variables & Create instances**

  All variable information is set in a variable file previously, including:

Image	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64
Flavor	uom.mse.2c9g
Availability Zone	melbourne-qh2-uom
Volumns	20GiB (each instance)
Instances	instance1, instance2, instance3, instance4,
Security Group	Egress Port: all Ingress Port: 22 (ssh), 80 (HTTP), 5984, 4369, 9100-9200(CouchDB cluster) , 2377(docker swarm), 3000(React), 19999(instance status visualization), 9000(docker visualisation) 8080(backend) 6379(redis) 16379(redis cluster)

Then the script will create instances following the variables.

- **Apply Proxy**

  As the public IP address issue, some proxy is required to be applied firstly on the system. This step will set the proxy address inside the environment variables, after that, the system will reboot one time to make sure all proxy settings work for the current session.
- **Add Essential Dependencies**

  The next step is install all required dependencies, including vim, python3, curl, git and so on. Update & Upgrade command will be firstly executed and all packages will be the latest version.
- **Install Docker**

  Docker will be widely used in this project because of its conveniences. In this step, the scripts will firstly check whether any previous version of docker exists in the system to avoid any conflicts. After that, docker will be installed across all instances and proxy settings will be executed after that.
- **CouchDB Cluster Setting**

  In this project, we applied three nodes as a couchDB cluster, two of them as sub-nodes and one of them as master node. In this step, we previously provided three different scripts using docker to install couchDB clusters across instances. The three scripts will be used to build master node, subnode one and subnode two. They will be firstly uploaded by the ansible procedure and then be executed. After execution completion, three nodes will be built as one cluster. Any error message raised in this process will be returned and displayed on the terminal.
- **Collect Project from Github**

  The next step is collecting source code from GitHub. The script will execute “git clone” with all git user information pre-set.
- **Deploy all components**

  As we use spring for backend and React.js for frontend, all relevant packages will be essential to automatically running the project. In this step, the script will automatically complete the following steps:
- **Redis Cluster**

  Three redis are run in different instance’s docker container by a Shell script and configure the three redis into a cluster.
- **Deploy Backend**

  Two backends are deployed in two different instance’s docker containers by a Shell script.
- **Deploy Nginx and Frontend**

  This step has two main responsibilities, deploying nginx and frontend. Besides deploying Nginx, the configuration for back end load balance and proxy and configuration for mapping frontend path by nginx.conf file.
- **Set up MapReduce Views**

  We used “grunt” as the auto-runner for the deployment of all MapReduce documents. All dependencies and other essential steps (e.g. build Database) could be completed by execution one .sh script.
- **Execution of Data Harvest Script**

  After deploying all MapReduce files, the script will execute different Python scripts on different instances for Data Harvest using Stream and Search API.

Data Harvest

Tweepy package is used in the project to collect data from twitter, there are mainly two APIs in tweepy for this tweet collecting procedure.

Stream API:

Tweepy's stream API could download twitter messages in real time. It is useful for obtaining a high volume of tweets, or for creating a live feed using a site stream or user stream. Location limitation is used inside the filter object to collect data only for the Melbourne area. We override some functionalities inside the Listener object (e.g. on_error), to allow Tweepy to reconnect for some or all codes, using the backoff strategies recommended in the Twitter Streaming API Connecting Documentation.

We also used the stream API with some filters of track like "covid", "coffee", "toilet paper" and so on to collect more data for exploring some topic-related scenarios.

Search API:

Tweepy's search API could return a collection of relevant Tweets matching a specified query. Currently some attributes are not supported to be searched or filtered in this endpoint. We set some available attributes in Search API, including Geo information, result type (we applied "mixed" mode to include both popular and real time results in the response) and so on. After getting results, we will apply some operations including attaching Suburb information and checking duplication before save into Database.

Collect Data from Aurin

As there are some kinds of information which will be hard to collect from Twitter, like unemployment rate, quality of education or the number of hospitals, we used aurin as a complement for data and scenarios. As the original data export from Aurin cannot be used directly by our system, we implement some scripts to further handle all data from Aurin.

We mainly collect data from sub-level "sa2" and the data handler script will recognise relevant regions and convert into the standard format used in our system.

History Data

Because of the speed limitation of Twitter API for tweets collecting, we also used the history tweets provided, so that some of our scenarios could be supported by more sufficient data.

Data Analysis & Handling

There are many operations on data after collection, including sentiment analysis, tag Geo information, MapReduce and so on.

Sentiment analysis

For sentiment analysis, we used a NLP library named “TextBlob”, which will analyse the polarity of sentiment based on the main text of each tweet. After analysis, the results will be recorded in two regions: “sentiment” which will record all original results generated by the NLP library and “attitude” which will record whether the attitude of the text is “positive”, “negative” or “neural”, and these results will be further used by the Map-Reduce function to explore scenarios.

Recognise & Tag Geo information:

The limitation of Geo information from tweet data is a big challenge for us at the first stage. Most tweets won’t contain any Geo information, and for the parts that contained, there is usually no detailed suburb information inside. That means we need to recognise the Geo information and map tweets to suburbs by ourselves.

According to Twitter Developer Documentation, there are mainly two positions in each tweet result that may provide Geo information -- “**coordinates**” and “**place**”.

For “**coordinate**”, its coordinate data without any detailed suburb information, and we will need to find out which suburb each coordinate belongs to. The solution is: we collect coordinate-suburb data of Melbourne from the government website, and we use the coordinate information in tweets to compare with this data to make sure which suburb each tweet comes from. In this process we used a Python package called shapely to build Polygon based on the coordinates list, and check which Polygon each coordinate point belongs to.

Also, for the “**place**” data, there are several different “place_type”:

- **Poi** - location is a specific point (such as a scenic spot on the map)
- **Neighborhood** - location is a specific suburb
- **City** - location is a specific city
- **Admin** - location is a specific province
- **Country** - location is a specific country

As we aimed to analyse based on different suburbs in Melbourne, we mainly used “place” data when the “place_type” is “neighborhood”.

After Geo recognition, we will put the suburb information in a separate position named “suburb” for further usage.

Map-Reduce

For Map-Reduce, we implemented multiple pairs for analysis of many scenarios, some main parts are as follows:

- ***Hot words - Sentiment -Suburb***: We collected the times each topic mentioned in Twitter data by analysing the texts and hashtags. After that we filtered tweets with the hotest words, like “covid”, “victraffic”, “streetartmelbourne” and so on, to explore people’s attitudes when they are talking about these topics. We will also collect the spatial features of these words and we could use the results to discuss some scenarios

further, e.g. which suburb mentioned “victraffic” mostly with negative attitude, the results may represent bad traffic situations in that suburb.

- ***Sentiment Analysis of Suburbs***: we calculate the average sentiment value of each suburb to get the sentiment situation map of Melbourne. And this data is used for further exploration after combining the results of education, unemployment rate and so on.
- ***Number of tweets by Time***: we separate and group tweets based on the created time to explore which time people tend to post more tweets. This result is very interesting to display people’s daily life.
- ***Non English Tweet - Suburb***: we compared Geo information with the kinds of languages used in non English tweet, to see whether we could catch some location distribution features based on the languages.

More detailed scenario information will be covered in the “Scenario” part later.

Backend Application

The main responsibility of the backend is to offer a RESTful API with High Availability, High Concurrency and quick response. SpringBoot framework is applied. SpringBoot reduces lots of development time and increases productivity and avoids writing lots of boilerplate code. Ektorp is integrated into the backend to connect CouchDB. It is a Java Persistence API, making it easy to communicate between backend and CouchDB.

Deliverable API

The table below lists the core API for our project. See whole api and more details, please visit our api documentation (you must be under unimelb network), which is generated by Swagger.

URL (prefix: http://172.26.131.6)	Method	Description
/api/attitude	GET	Get each suburb’s attitude
/api/sentiment	GET	Get each suburb’s sentiment
/api/time	GET	Get the number of tweets in each hour
/api/no_english	GET	Get the number of no english tweets in each suburb
/api/hotword	GET	Get a list of hot words with sentiment info
/api/hotword/suburb?word=covid	GET	Query a specific hotword in each suburb distribution

Load Balance

The increase in the number of users in future need to be considered, so high concurrency and high availability are a part of backend design goals. At this moment, two backends are deployed in different instances and use Round Robin load balance policy by nginx. If one backend crashes, another backend is

still alive and can support service, so it improves high availability. If the number of users is increased and requires high concurrency, just need to increase the number of instances and docker containers to run more backends, which embrace the main feature of cloud computing, elastic computing.

Load Test Result by JMeter

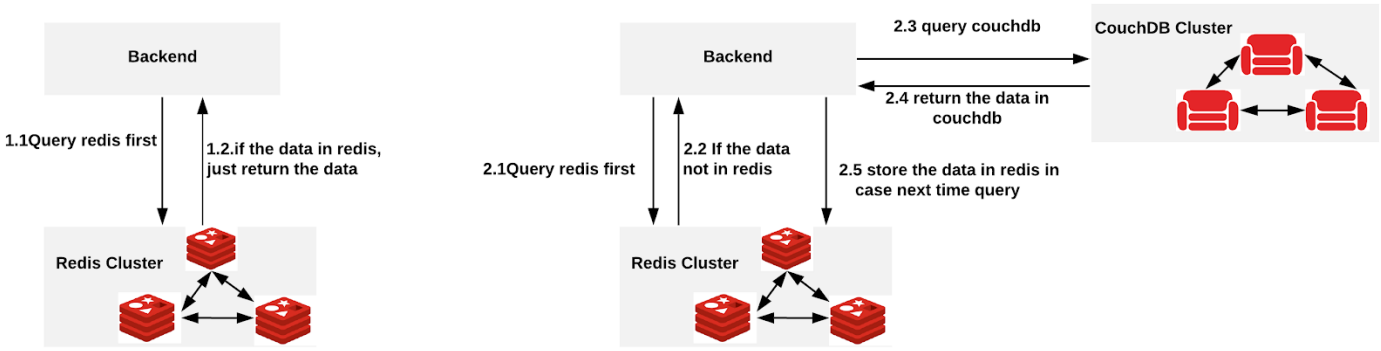
The test runs in a MacBook Pro 2015 13 inch with CPU 2.7GHz Intel Core i5-5257U and Memory 8GB DDR3 SDRAM 1,866MHz and Network NBN 50 Mbps. The result is below table.

API	Throughput (num/sec)	Error (%)	Received (Kb/sec)	Sent (Kb/sec)
/api/attitude	3.3	4.10	29.85	0.38
/api/sentiment	3.3	3.92	22.52	0.38
/api/time	3.6	4.32	21.84	0.36
/api/no_english	3.5	3.41	14.18	0,39
/api/hotword	3.4	3.24	7.78	0.36
/api/hotword/suburb?word=covid	3.3	3.82	7.32	0.39

The throughput in all api is only around 3.4 per second, which is very slow. It can be improved by adding more high performance instances and running more backends in docker containers, but the computing resources of this project are limited.

Middleware Cluster Redis

In order to speed up response time, a middleware is essential. With system evolution, CouchDB may become a bottleneck, because many requests hit CouchDB and it may slow down CouchDB performance. Lots of query requests are repeated and CouchDB responds to the same data, which let CouchDB do the same query and waste computing resources. Redis is an in-memory data structure database, so the speed is overwhelming. Before querying CouchDB, query redis first, if there is no data in redis then query couchdb and store the data into redis as well, so next time the same query can be found in redis. The figure below shows data flow of how backend query data with redis middleware.

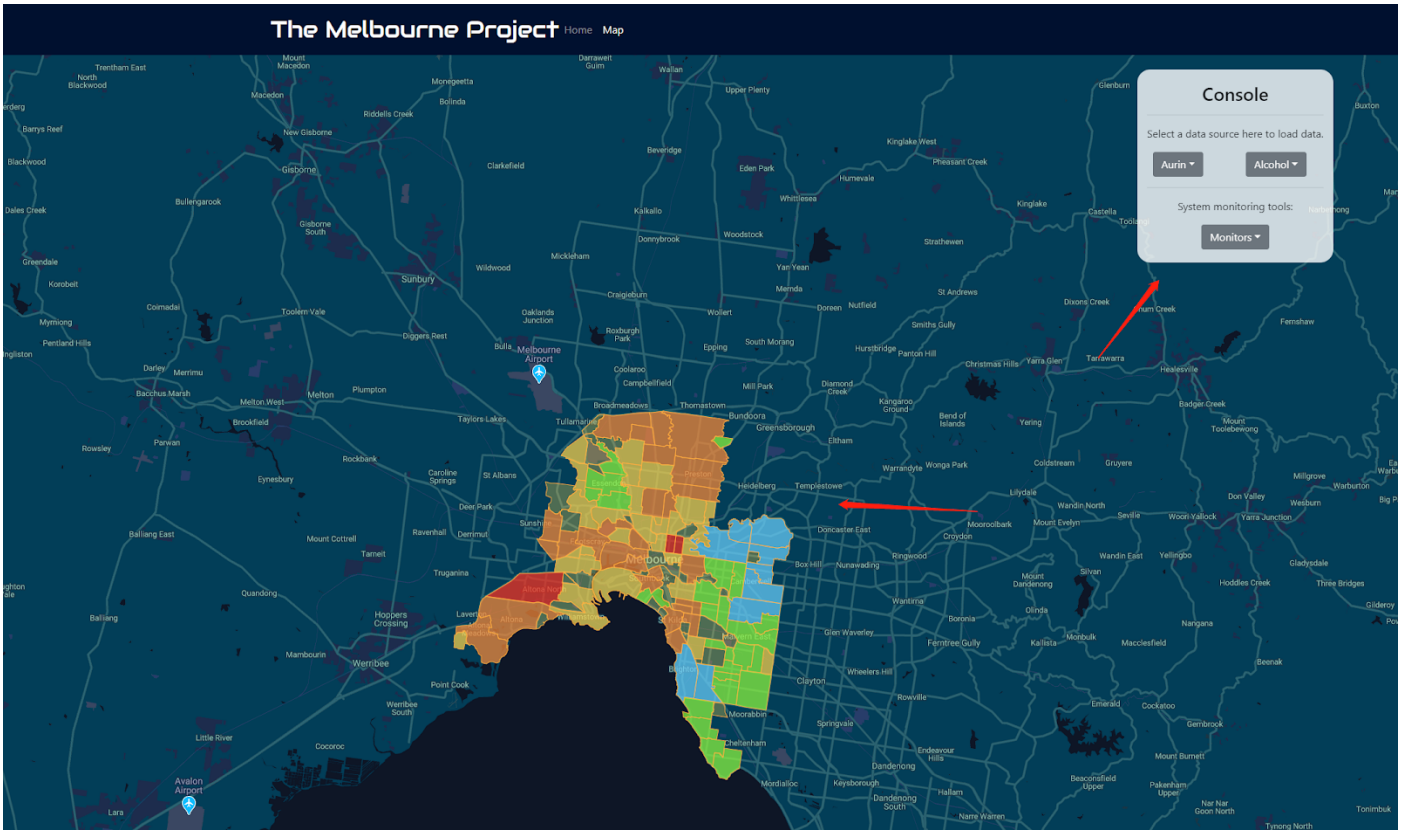


The data in redis **The data Not in redis**

To make redis more reliable, building a cluster of redis is needed. Three redis are deployed and configure them into a cluster in our project.

Frontend Application

Given the data processed by the backend, we are aiming to provide a friendly and neat user interface, which helps to better demonstrate the data. Our front-end application is built by **React**, and the reason is that react is a light-weight but powerful library which we can start with quickly. The simple data flow makes it easier to manage complex data and there are tons of third party libraries we can easily reuse to achieve our design goals.



Generally, the frontend application includes a home page and a map page. The home page provides some basic information and the map page is the main stage for us to display. The map occupies most of the page, and there is a console in the upper right corner to allow us to select a data source or use monitoring tools. There will be a chart right below the map after selecting a data source.

Frontend functions mainly include communication with the backend to fetch data, visualise data on the map, and display data by charts. We used Nginx as a http server, which stores frontend static resources, listens to frontend requests on port 80, and passes all backend requests start with /api/ to upstream backend nodes as mentioned before.

Frontend to Backend communication

To get the Twitter data from the backend, we use a Promise based http client called axios to send asynchronized requests to the backend and then get data back. After receiving the data, we use our implemented dataFormater to reformat the data once again to accomplish the needs of the front end. We also use React.Context to store data globally and pass it through to the component tree which simplify the data flow and ensure the data consistency as well as reducing duplicated querying methods.

Google Map

To better display the relationship between tweets and location, we tried to display geo-related data on the map. We chose to use the original GoogleMap API (Google Maps JavaScript API V3) which enables loading GeoJSON on the map as a data layer. With the help with the data layer, we are capable of adding interactive events on the map, such as displaying info windows with charts after selecting a region, or rendering the map with different colors. According to the data, we rendered the map as a heat map. More specifically, we use blue to indicate the lowest amount or index, followed by green, yellow, orange, and the highest is red. With customised google map style, we managed to implement the map that is in line with our design style./

Chart

To demonstrate the data, we chose to use a versatile third-party library called recharts, which allows us to render different kind of data by formatting the given data to the required data format. We are able to modify the display of charts as well as adding onClick or hover functions to make our charts more informative.

Error Handling & Tolerance

Avoid Tweets Duplication:

As in the process of using Twitter APIs, there might be some duplicated tweet return by twitter that we don't want to add into the Database, some duplication avoiding methodologies needs to be applied. The solution

used in this project is very straightforward, as we noticed that the id_str attribute in tweets is uniquely related to one single, we used that attribute as the _id attribute of the tweet objects in Database. In this way, whenever a new tweet becomes, we could avoid duplication by checking whether the same id originally existed in the database. If the tweet already existed, we won't write it into the Database so we could make the data avoid duplication from the beginning.

Tolerance one backend crash:

Two backends are deployed in different instances to do load balance, with a bonus that improves High Availability(HA). If one instance hosted a backend crash, the backend still can provide service from another instance. For example, a instance has a crash possibility 0.01 in 24 hours, if we only deploy backend in one instance, there is 0.01 possibility backend down, if backend is deployed in two different instances, there is only 0.01*0.01=0.0001 possibility backend down in 24 hours, almost impossible lost backend service.

Analysis of Pros and Cons of the UniMelb Research Cloud

Pros:

As Unimelb Research Cloud is based on OpenStack, one main advantage is its easy to use and manage feature. This includes many aspects like: we could easily manage many resources in a centralised dashboard, we could easily set up the instances, security group, keys and so on. This helps us to start the process quickly with intuitive ideas of how everything works.

Also, its open source feature helps us a lot, as there are many resources and technologies we could use to implement our application.

Cons:

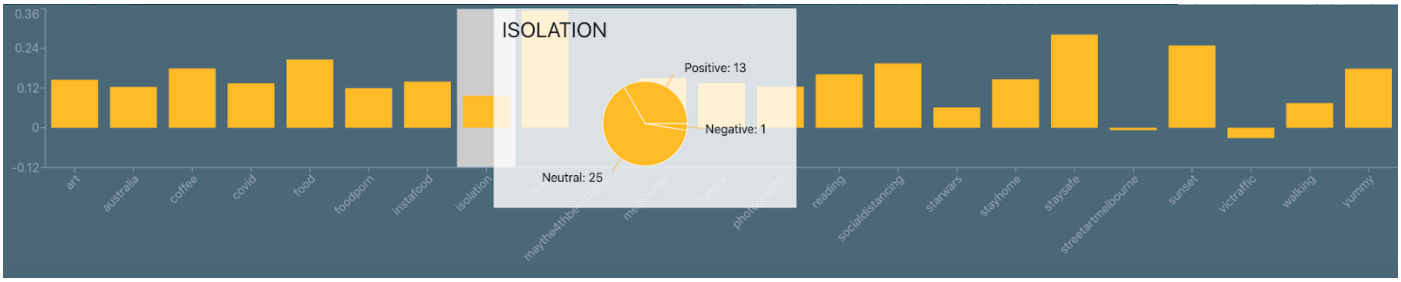
One main issue bothering us during the developing process is the instability of the cloud, especially the individual instance. The instance crashed several times while we were posting data to CouchDB or while we edited the MapReduce functionalities. This results in some loss of time as we have no time to store all data before crashing. We have to deploy some backup instances in later processes so that some important data won't lose, but the data sync becomes another thing we have to handle many times as we used different instances.

No Floating IP resources for this project. The overall architecture of this system is robust except for the nginx part. If nginx is down, the whole system will be out of service. We definitely want to fix it and have tried to use Keepalived + multiple Nginx and Linux Virtual Server(LVS) + multiple Nginx, but both Keepalived and LVS need to use virtual IP, which means floating ip resource is required, but unfortunately we are not allowed to use floating ip in Unimelb Research Cloud.

Scenarios

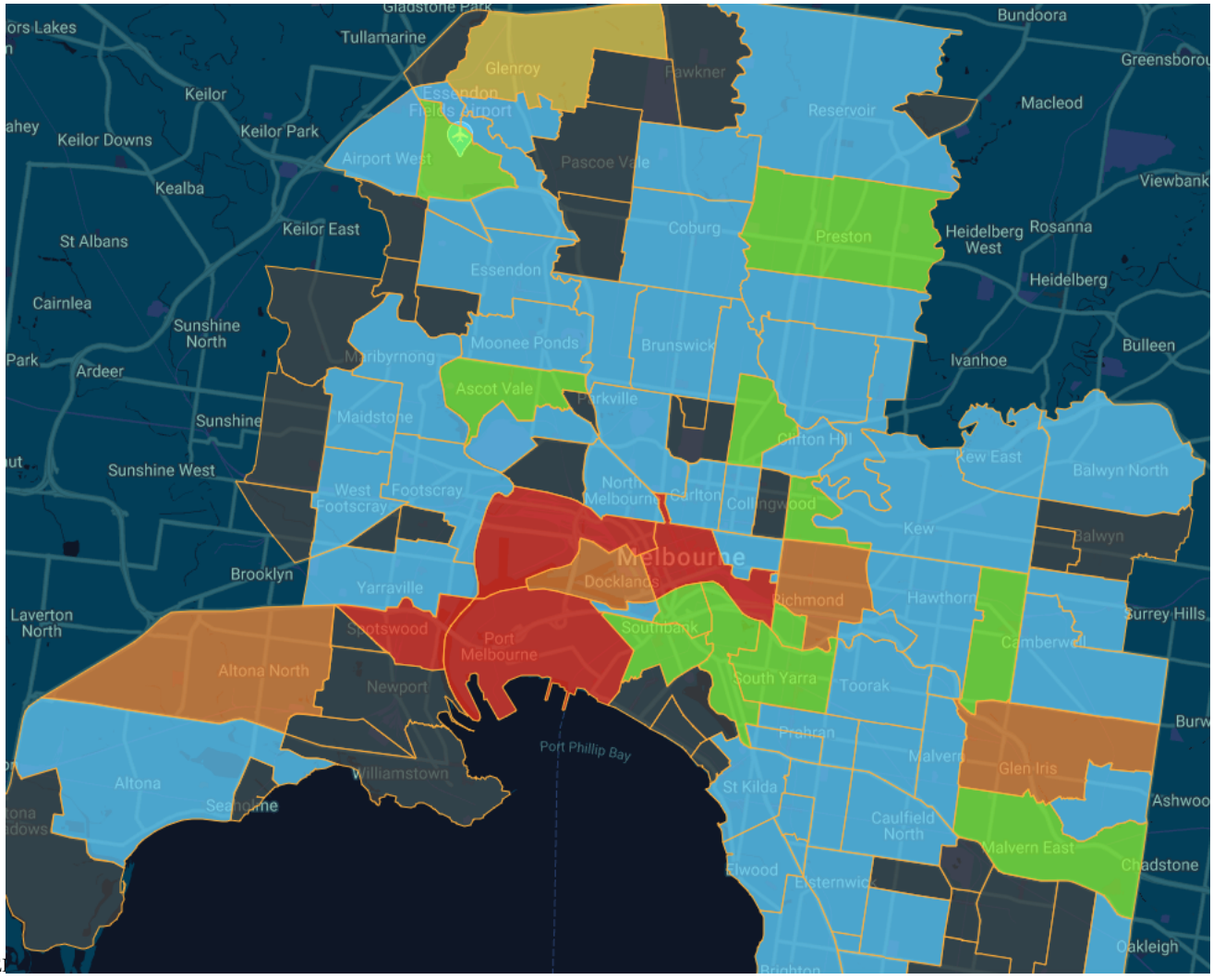
Key Scenario 1: Hot words

We collected some hot words and topics from twitter data. By selecting Twitter -> Hot words, there will be a list of all hot words.



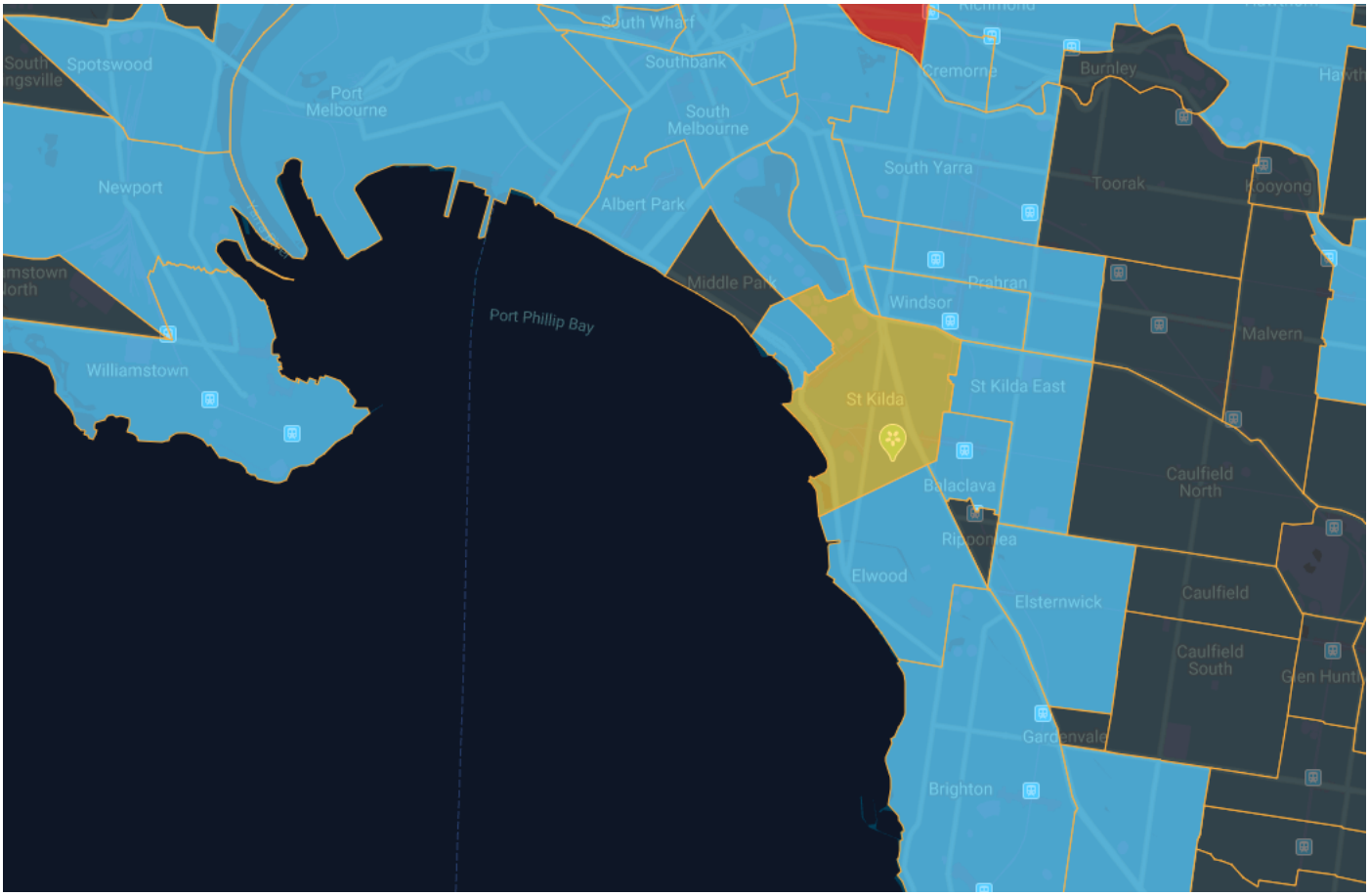
Here the bar chart represents the average sentiment of each hotword's related twitters. The higher the more positive. From the chart we could see hot words like "Love", "sunset", "food" often appear with more positive attitudes. In contrast, words like "Victraffic" have more negative related tweets.

Apart from these sentiment data, there is also geography information for each hot word. For example, by clicking the word "**victraffic**", the map will reload and the map will show. The color on the map represents the number of times that this word is mentioned in each suburb. By clicking "Victraffic", the map will looks like this:



Suburbs like Dockland, City, Richmond have deeper colors meaning more times of "victraffic" related tweets. This may represent the traffic situation in these areas are worse compared with other suburbs.

Also, users could explore other words for some other interesting scenarios. For example, by clicking "**Sunset**", the map will change to this:

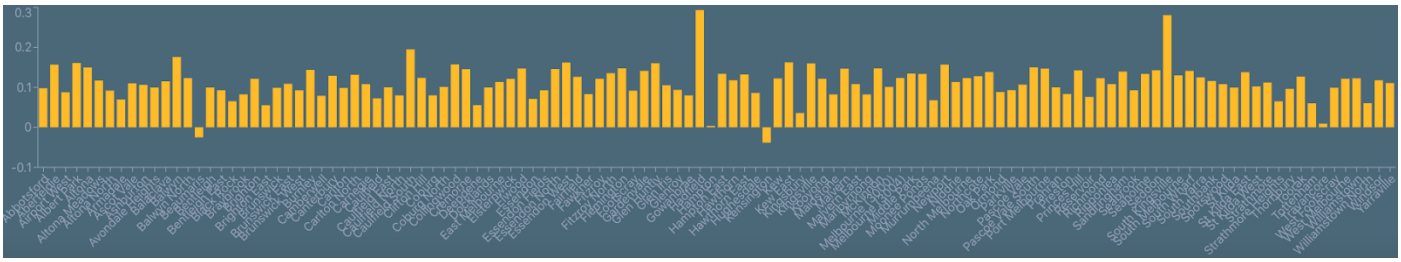


We could see St Kilda has much more "sunset" related tweets than other regions, this make sense considering St Kilda's beautiful sunset most of the time.

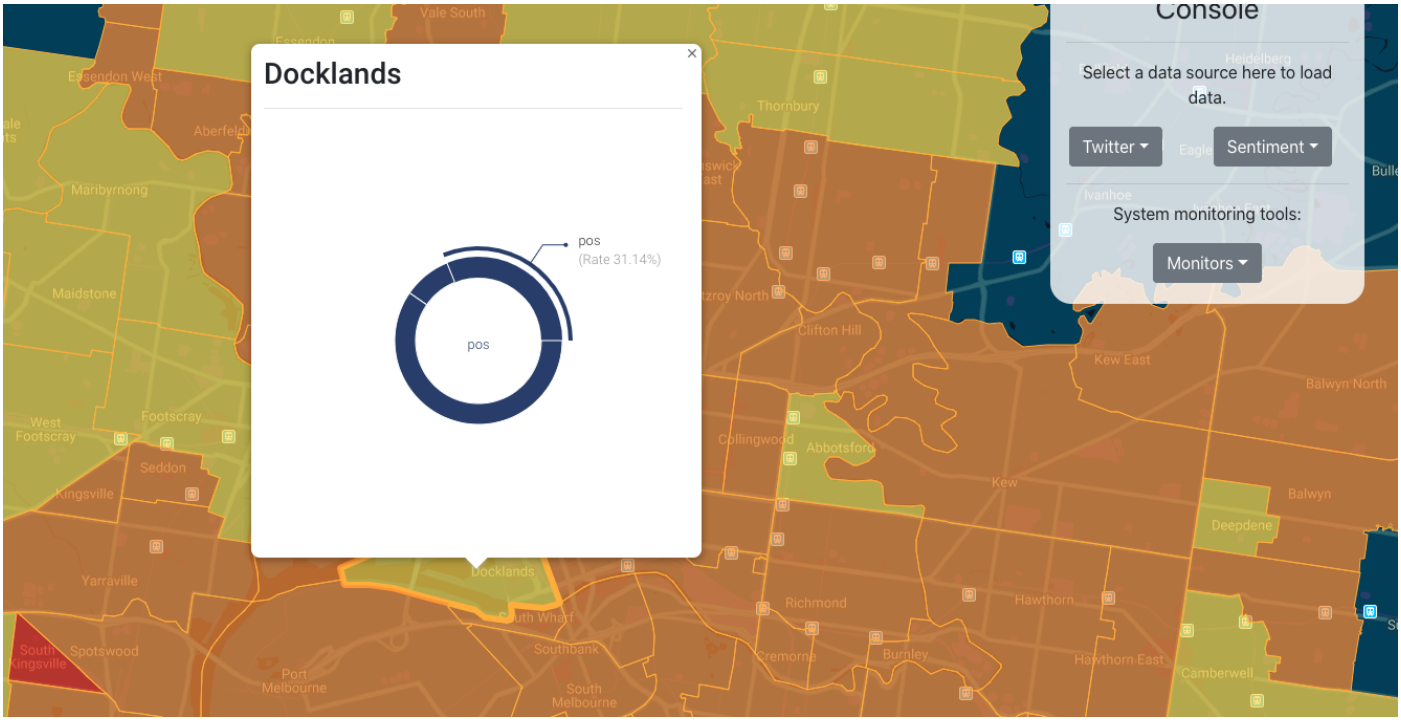
Key Scenario 2: Suburb - Sentiment

In this part, users could explore the sentiment situation in each suburb. The data is collected by analysing all tweets' text using a NLP library. By selecting Twitter -> Sentiment, Map will reload with different colors, representing different levels of sentiment data as follows:

By scrolling down, there will also be a bar chart for more accurate average sentiment data.



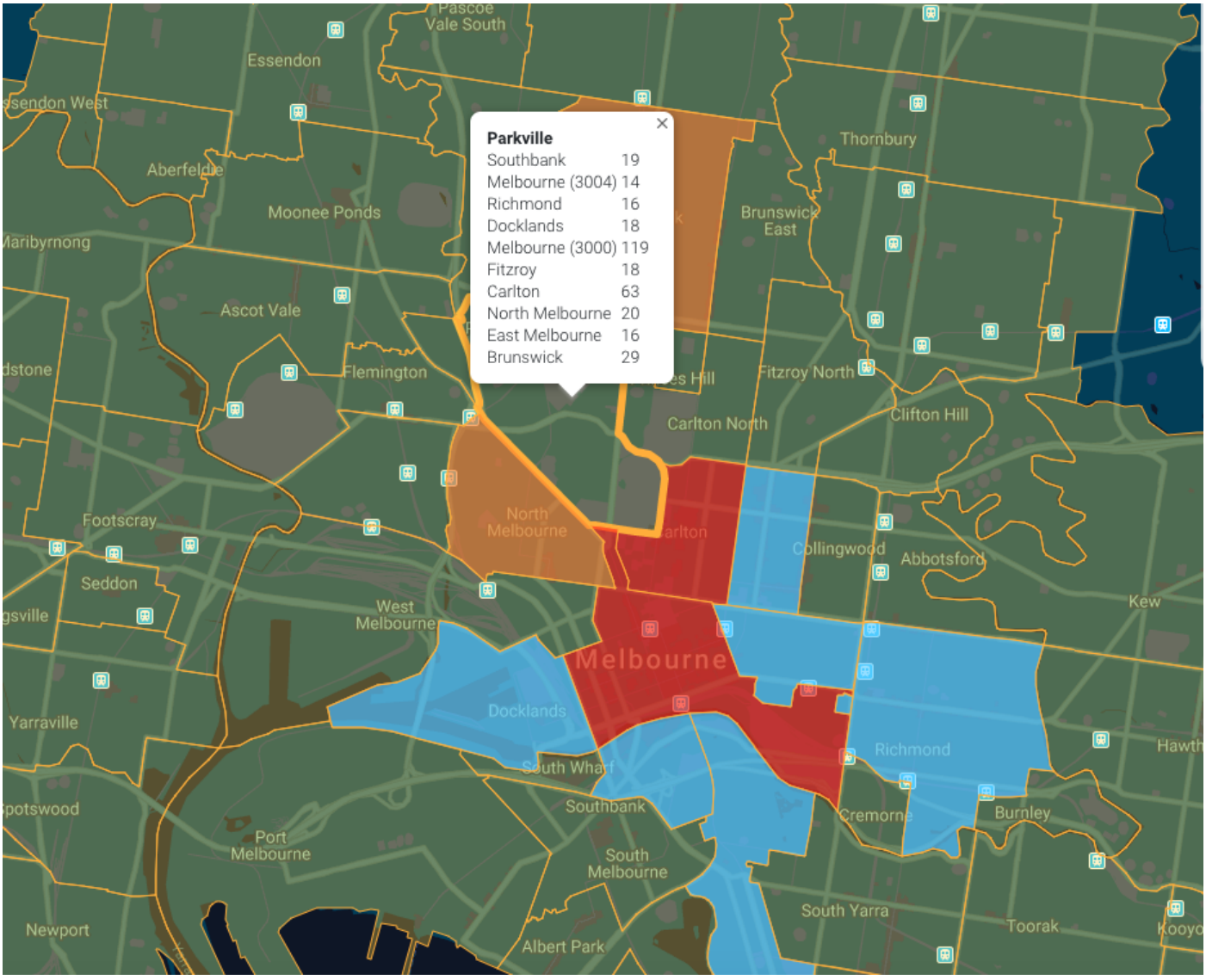
Apart from there, by clicking one suburb on the map, there will be a pie chart displaying how many tweets have positive/negative/neutral attitudes.



This sentiment data is used for further exploring other scenarios by combining with other data, e.g. does people in lower unemployment rate regions have more positive feelings? What's the sentiment data's relationship with Medium income data? What about smoking rate? What about Alcohol sales?

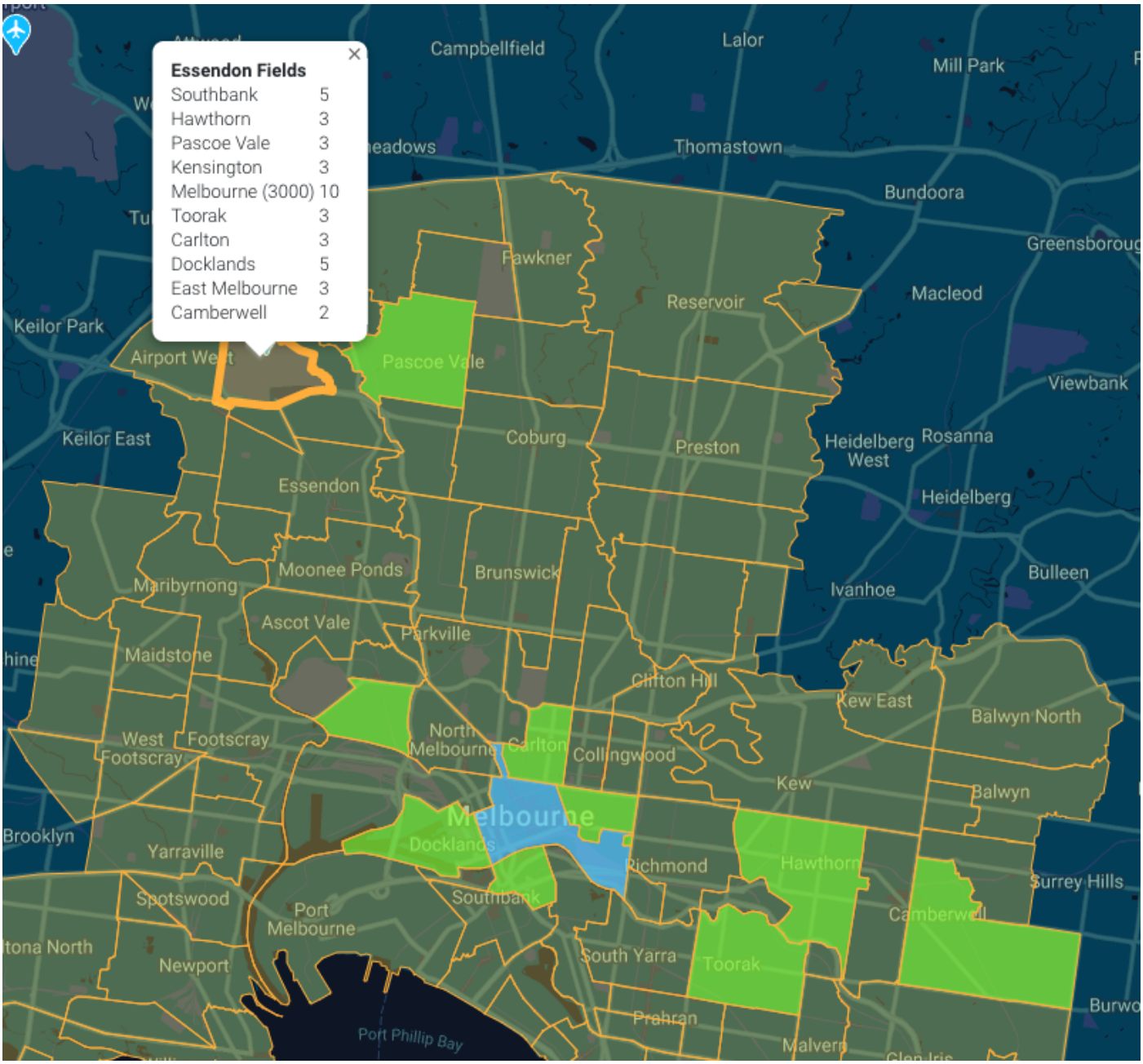
Key scenarios 3: Living Region

This data is collected by collecting Twitter users' information. Firstly, we track one single user's tweets and find out the geo information inside. If one single user posts tweets in two different regions, we will set one connection between these two suburbs. Apparently, the most "connections" there are between two different suburbs, the most closed they should be. After counting and analysing, for each suburb, we will find its TOP 10 most closed suburbs will display on the Map. Still, the colors represent different numbers of connections between suburbs. The map will look like this if we select "Parkville":



We could see the closest regions for people in **Parkville** are City and Carlton, and the colored suburbs in the map together with Parkville become a common "living region" for people in Parkville.

More interesting, if we select the suburb where **airport** located, the map will look like this

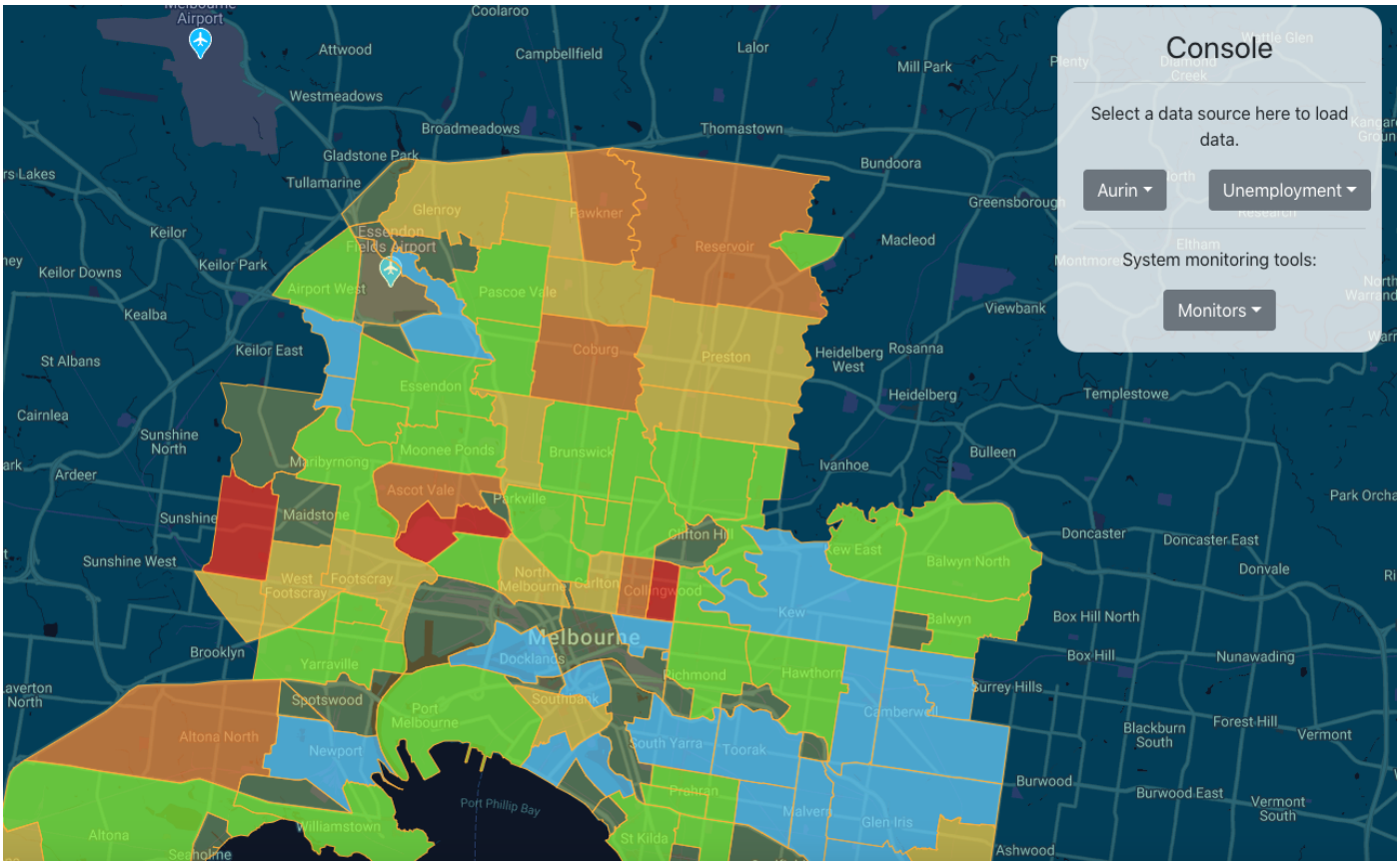


We can see the living regions of the airport are all very far away from it and are very separated. This might be because of the fact that the main flow of people to this suburb are people heading to the airport from different regions. Hence, the living regions for the airport become no longer geographically adjacent.

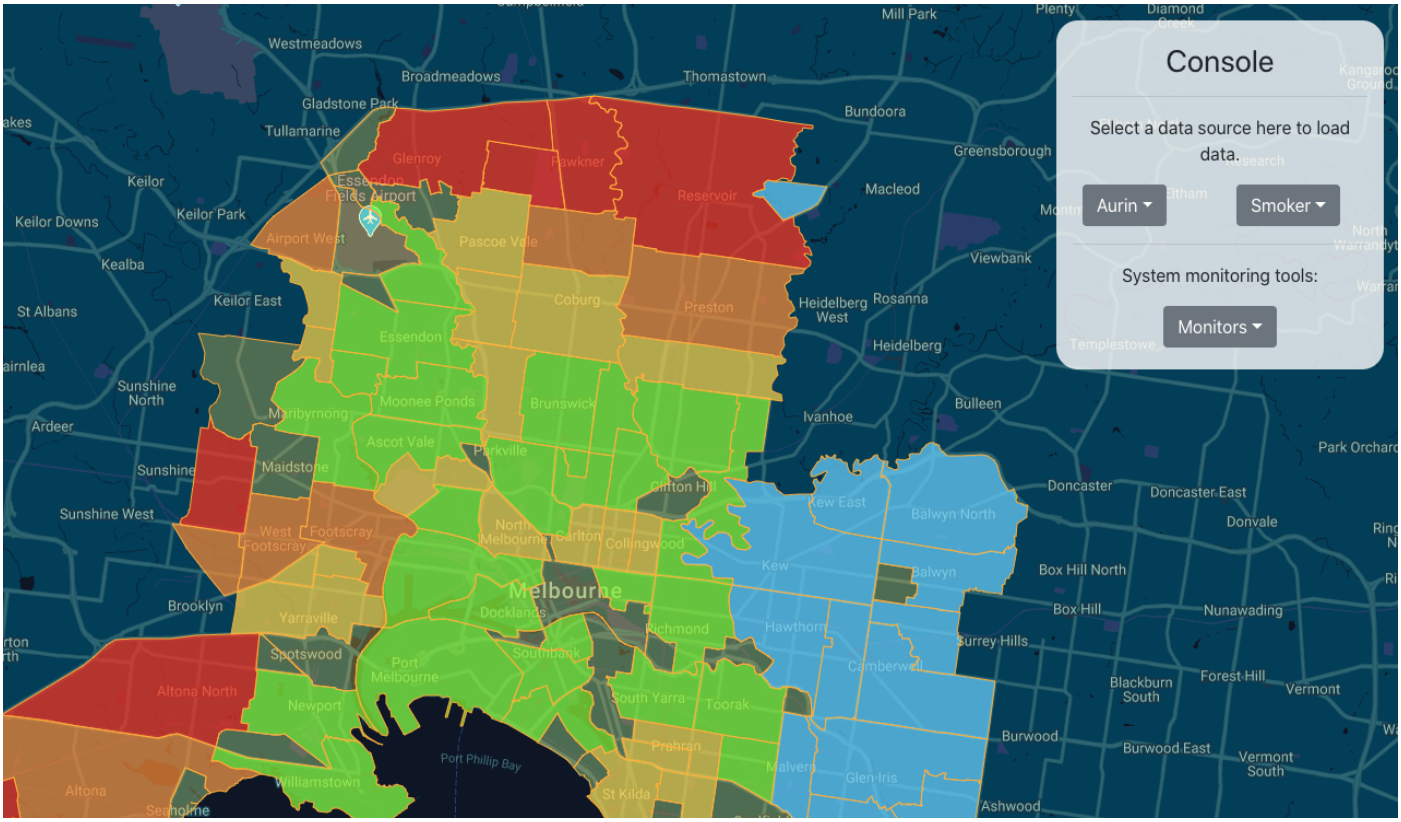
Other Scenarios

There are also many other interesting scenarios that users could explore. Here are some of them:

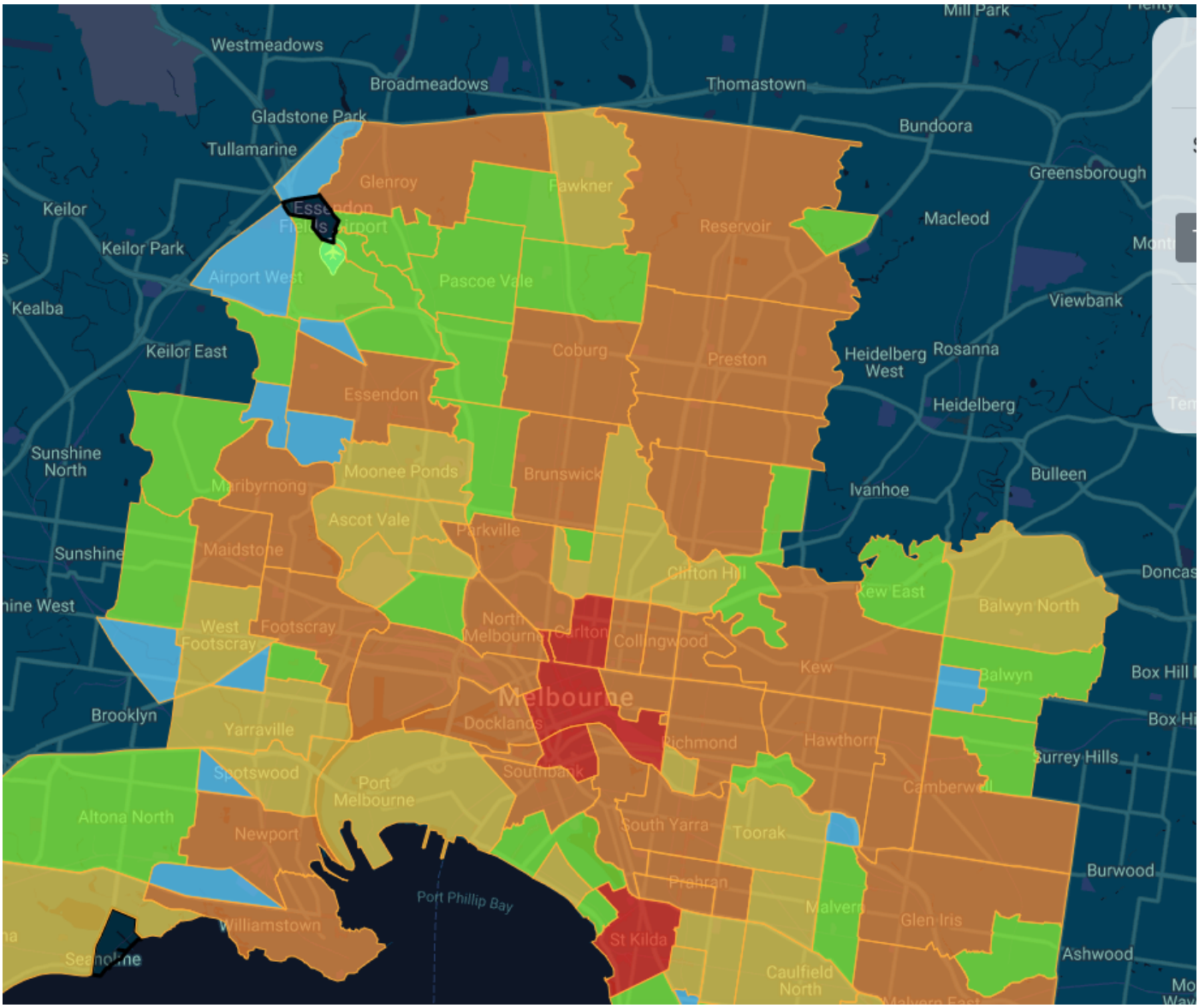
Unemployment Rate:



Smoker Rate:



Geo distribution of non-English Tweets:



The number of tweets people post on different **time** in one day

By combining these results, we could explore some interesting scenarios. E.g. We found suburbs with higher medium income usually have lower unemployment rates, and these suburbs are also more positive based on Twitter data. Also, we found the distribution of smokers maps are very similar to alcohol sales maps.

Conclusion

In this project, we build a cloud based system using data collected from Twitter. We explored many interesting scenarios and displayed all of them. In this process, we learned and used many technologies, including DB cluster, ansible, nginx, React, springboot, NLP, redis and so on. We met many challenges and issues in the process but finally we are satisfied with the outcome. It's a great opportunity for us to get more experience about cloud computing and implementation of an entire system.

References

React – A JavaScript library for building user interfaces. Retrieved 24 May 2020, from https://reactjs.org/

Overview | Maps JavaScript API | Google Developers. Retrieved 25 May 2020, from https://developers.google.com/maps/documentation/javascript/tutorial

GeoJSON. Retrieved 25 May 2020, from https://geojson.org/