COMP90024 Cloud and Cluster Computing
--Assignment 2 Australian Social Media Analytics

Team Information

Name	StudentID	E-mail
Yiru Pan	889832	yirup@student.unimelb.edu.au
Lu Chen	883241	lchen12@student.unimelb.edu.au
Wenyi Zhao	882335	Wenyiz4@student.unimelb.edu.au
Peng Kuang	779206	pkuang@student.unimelb.edu.au
Fazard Vazirnia	942555	f.vazkrnia@student.unimelb.edu.au

1. Introduction

Big data has been a buzz word for many years, mining the information and knowledge behind these data could deliver huge business and research values to companies and organizations. Twitter is one of the most popular social media among people, from breaking news and entertainment to sports, politics, and everyday interests, what's more, people could see all sides of the story, join the conversation, watch streaming events, etc. People are more connected by twitter. These data can be used as research materials in human emotional and behavioural researches. This project aims to use public cloud platform Nectar(IaaS) to handle a large amount of social media data provided by twitter, and implement sentiment analysis behind the big data, then visualize the data in a web based app. Our team used program written in Python to harvest tweets from both Twitter Streaming API and Twitter Search API, and gained about 3 million data of Melbourne. For data set, 2.8 million is generated by our harvester, and 0.2 million is provided data [1]. We used TextBlob, a NLP Python package, to implement sentiment analysis. In order to explore the potential meaning behind the big data, we prepared several scenarios related to "happiness" and will find the underlying relationship between these contributing factors and "happiness". MapReduce functions were used to transform tweet data into statistics data and frontend will display these data as charts. By doing this, we could acquire a clear view of the sense of "happiness" among people in Melbourne.

Our team is composed of 5 members, the main responsibilities are as below:

- Yiru Pan- System Administrator (Nectar cloud setup, CouchDB setup, Ansible/Boto Automation, architecture design)
- Lu Chen- Backend Developer (twitter harvester)
- Fazard- Backend Developer (twitter harvester, Map Reduce, Django)
- Wenyi Zhao- Frontend Developer
- Peng Kuang- Frontend Developer

2. System Functionalities

This system is a web app which could display different scenarios related to "happiness" in Melbourne, the main functionalities are as below:

- System Automation: setup cloud environment in Nectar automatically by Boto, all software and dependencies are installed by Ansible

- Tweet Collection: a harvester program using Twitter Search API and Twitter Streaming API to collect data in Melbourne.
- View Generation: use couchDB Map Reduce functions to generate views for data collected by harvester.
- Data Visualization: frontend displays views as statistic charts, present audience intuitional data.

2.1 System Design and Architecture

This section is about design and architecture of this project. The whole system is a normal 3-tier model, including presentation layer (frontend data visualization), application layer (business logic, webserver) and data layer (harvester, couchDB database).

The 3-tier architecture is as diagram illustrates:

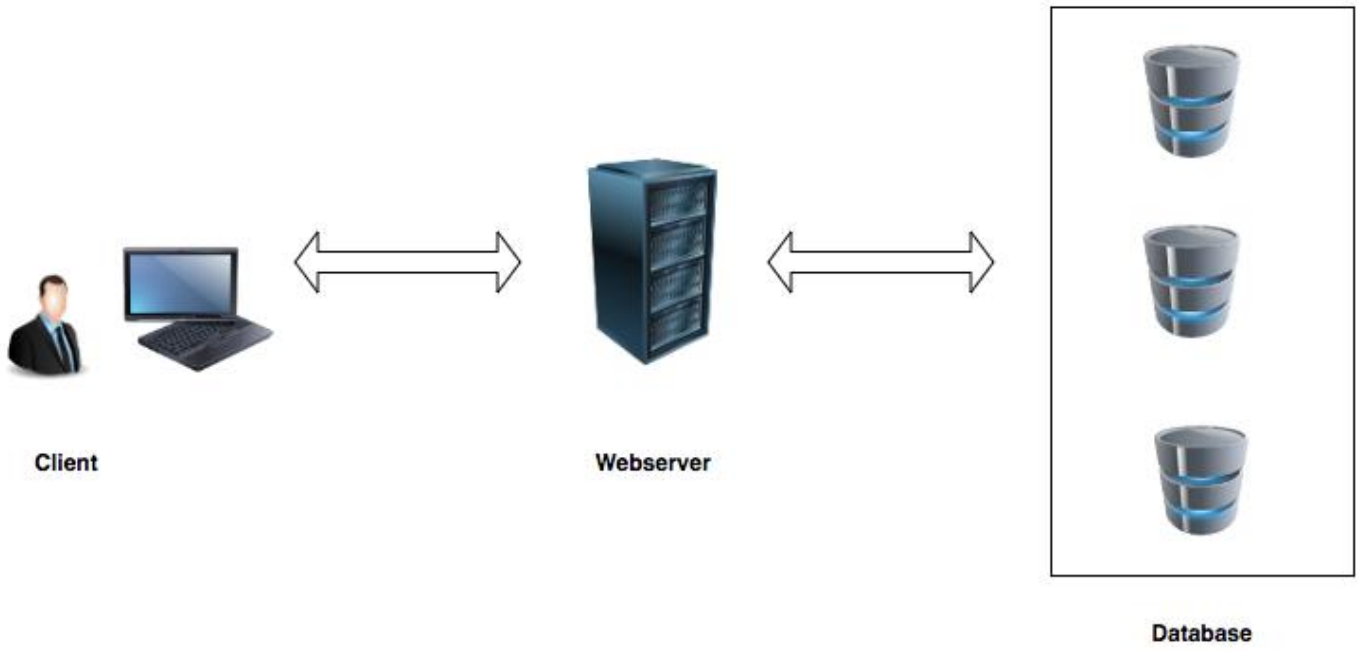


Figure 2.1-1

Frontend used jQuery Ajax to request RestAPI in webserver, and webserver will query database and provide response to frontend. In our architecture, we used Django webserver and couchDB as database. 1 instance in Nectar is used as webserver, and 3 instances in Nectar are used as couchDB instances. 3 couchDB instances have been set up as replicator to each other, each data node has a full copy data of two other nodes, which aims at backing up data when a node encounters failures accidently. The advantages of adopting 3-tier architecture are as below:

- It makes the logical separation between presentation layer, business layer and database layer.
- Each tier is independent, it's possible to enable parallel development without interfering other tiers.
- Database layer will be more secured since application layer is in the middle, client will not have direct access to the database.
- New rules can be defined anytime, and changes made to middle layer will not affect presentation layer.

As for business logic in webserver, we used Django as framework which is compatible with our python harvester. Webserver is in charge of querying database, and providing json view to frontend.

The detailed design diagram is as below:

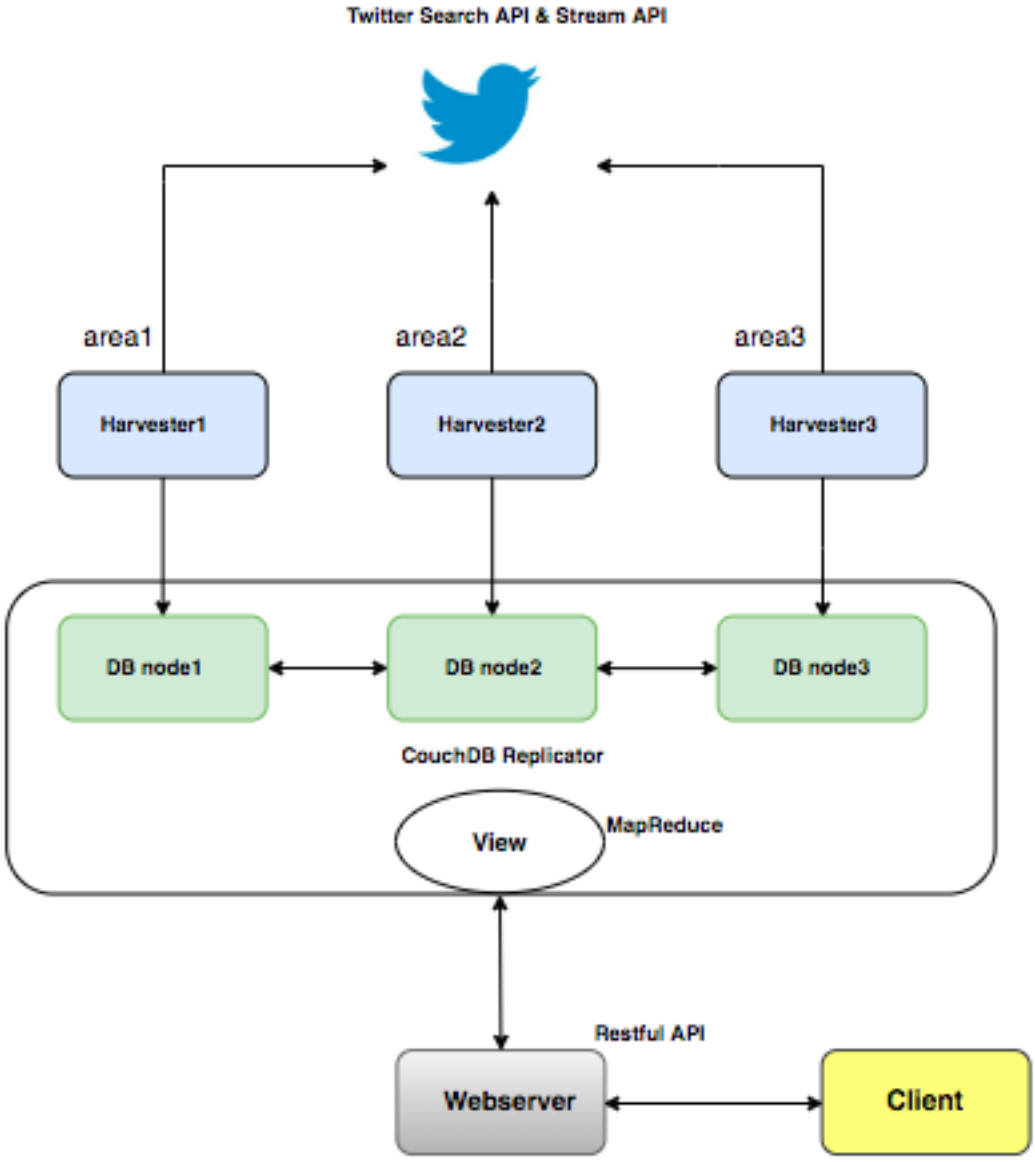


Figure 2.1-2

2.2 Nectar Cloud Environment

This project used Nectar Research Cloud and deployed a cloud-based solution using the given 4 instances, 8 CPUs and 250GB volumes. 3 instances are used as database nodes to manage harvesting and tweet storage, 1 instance is used as webserver to provide web service.

2.2.1 System Setup

The detailed configuration is as below:

Nectar Cloud Instances

	Image Name	IP Address	Size	Availability Zone	CPU	RAM	Permanent Volume	Attached Volume
Webserver	Ubuntu 16.04	115.146.85.254	m1.large	melbourne-qh2	4	16GB	10GB	0
DBserver1	Ubuntu 16.04	115.146.86.136	m1.small	melbourne-qh2	1	4GB	10GB	30GB+30GB
DBserver2	Ubuntu 16.04	115.146.84.192	m1.small	melbourne-qh2	1	4GB	10GB	30GB+60GB
DBserver3	Ubuntu 16.04	115.146.85.230	m1.small	melbourne-qh2	1	4GB	10GB	30GB

Figure 2.2-1

The instance can be created by boto, use ec2_conn.run_instances(), and volume can be created by ec2_conn.create_volume(), then the volume can be attached on instance by ec2_conn.attach_volume(). Boto package provides bunch of functions, which facilitate automation deployment on Nectar EC2(Amazon).

We use default security group and add several rules, the detailed configuration is as below:

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
Egress	IPv6	Any	Any	::/0	-	Delete Rule
Ingress	IPv6	Any	Any	-	default	Delete Rule
Ingress	IPv4	Any	Any	-	default	Delete Rule
Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	TCP	5984	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	TCP	5986	0.0.0.0/0	-	Delete Rule

Displaying 8 items

Figure 2.2-2

Browser can access webserver by port 80, port 5984 and 5986 are mapping to couchDB. Terminal can access host by SSH. Our key pair is “cloud”, we need to input `ssh -i ~/.ssh/cloud.key ubuntu@hostname` to access host.

During the deployment, we found no matter we chose m1.large or m1.small, the root volume(/dev/vda) is always 10GB. For webserver, since it doesn’t need to store big data, extra attached volume is not necessary. For DBservers, firstly we attached 30GB volume for each of them. The attached volumes are on /dev/vdc, and mount in root directory /database. However, with the data increasing, DBserver1 and DBserver2 already exceeded the expected volume, so we extended the storage and attached extra volumes to them, DBserver1 extra volume is on /dev/vdd, mount in root directory /database1. DBserver2 extra volume is on /dev/vde, mount in root directory /database1. After we extended the storage, DBserver1 and DBserver2 were running normally.

After setup all instances, related software and dependencies are supposed to be installed. Ansible is a very useful tool for configuring software packages. It combines multi-node software deployment, we can run a single yaml and install softwares on different nodes. In our project, there are several roles to install different dependencies. For example, common is for all nodes, the tasks in common include format file system, mount volume, install curl, docker, all python dependencies for harvester and couchDB. What's more, there are specific roles for different servers, such as setup couchDB replicator, each database server has its own "replica" role and tasks.

Once all dependencies are ready, the harvester program can be executed on DB nodes, and from couchDB websites, we could see the data amount is increasing.

Here are several technical highlights in our deployment:

- Use Docker to run couchDB container-command:

```
sudo docker run -d -p 5984:5984 -e COUCHDB_USER=admin -e COUCHDB_PASSWORD=team38 –v /database:/opt/couchdb/data --name couchnode apache/couchdb --log-opt max-size=10m --log-opt max-file=3
```

The localhost port 5984 is mapping to couchDB default port 5984, and all data is stored in /database. Besides, we set the log max-size, since we found replicator generated a large amount of log files in /dev/vda, which causes the lack of storage. By setting the size limit, it will mitigate the storage issues. For DBserver1 and DBserver2, since we attached extra volumes, so the port mapping has been changed to localhost:5986, and we created another container for new volumes.

- Set couchDB replicator-command:

```
curl -H 'Content-Type: application/json' -X POST http://admin:team38@115.146.86.136:5984/_replicator -d ' {"source": "http://115.146.86.136:5984/tweet","target": "http://admin:team38@115.146.84.192:5984/tweet","create_target": false, "continuous": true}'
```

This is a replicator from DBserver1 to DBserver2, since replicator in couchDB is uni-direction, in order to achieve full copy between three database nodes, we set total six replicators, and each node has two replicators which points to another two nodes.

Detailed deployment code and user guide are in Appendix.

Please refer Anisble directory for automation part. Source code in Automation is for group instances, source code in Demo Code is for private instances. Logic of these two folders' code is same, differences are instances quantity and replicator quantity.

The automation video link is: https://vimeo.com/267390442

We used two private instances to demonstrate and record this video, since our team instances are running and processing data, we cannot delete or do other operations on them. Deployment logic is same, instead of running one command, we added several intervals and interact with

browser and terminal to verify if our deployment is correct. Therefore, 1 python command and 3 shell scripts should be executed.

2.2.2 Pros and Cons of Nectar Cloud

Nectar Research Cloud is an open source cloud platform which utilizes openStack, it offers free and open-source software platform for cloud computing(IaaS). Components of openStack are interrelated, including compute, storage, network, image, orchestration, etc.

During our deployment process, we found several advantages of Nectar:

- Use for free, we don't need to pay any money and can access instances to process big data. Students can implement research on this open-source and free platform.
- It's very convenient to use a web-based dashboard. In Nectar dashboard, we can see a list of Compute, Network, Orchestration, Database, etc. It's easier for a green hand to kick off learning path of openStack.
- It also supports RESTful API. In this project, we utilized EC2 API and boto(python interface for Amazon EC2) to set up cloud environment automatically.

Some cons are listed as below:

- The time of deleting an instance is relative long, sometimes we found the instance was already deleted in "Compute→Instances", however, "Compute→Overview" hasn't been updated.
- Private instances don't have volume, need to apply for approval. Waiting process is long.

2.3 Deployment Issues

As for system part, we experienced "storage lack" issue. Since docker generates a large amount of logs, which occupied permanent storage /dev/vda, and it's nearly full. Our strategy is that remove the current docker container, and run a new container with "log-opt" parameter to limit the size of log file. By this way, system returned to a normal state.
What's more, due to several versions of documents in couchDB, our attached volumes are not enough as well, so we extended volume by adding two extra volumes, mounted them on different root directories, mapped them on new ports and create new couchDB containers. New attached volumes work well.

2.4 System Fault Tolerant and Scalability

Assuming single node may crash during running process, a fault tolerant strategy is necessary to deal with system failure and data loss. In our project, in order to avoid data loss or other system failure, we come up with a backup plan—create snapshot for each volume. By doing this, our data has been cloned in snapshot, and won't be lost in accident.

Scalability can be achieved by adding or deleting instances dynamically. Boto ec2 API provides related functions to achieve this goal. For example, if more instances are to be created, just modify parameter in ec2_conn.run_instances(). If more instances need to be terminated, use ec2_conn.terminate_instances(). Automation tool makes deploying complex cloud system programmable[2], we only need to modify our code then the system will be changed accordingly.

3 Backend Architecture and Fault Tolerance Design

As it can be seen in the diagram below (Figure 3-1), the only point of using a three-tier architecture instead of direct calls to the databases is to decouple front-end code from the database technology and the number of replicas. This approach allows us to call the desired endpoints using a HTTP request to the application server and get the data in any arbitrary format independent of the database technology or response format. In our case, most of the calls done by the application server to the databases, do not need any further processing and the JSON format produced by CouchDB satisfies our needs.

By replicating data in three couchDB instances and using continuous master/master replication, even if one of them fails the availability of the system does not get affected as the application server attempts to call a different database if its attempt to the first one fails.

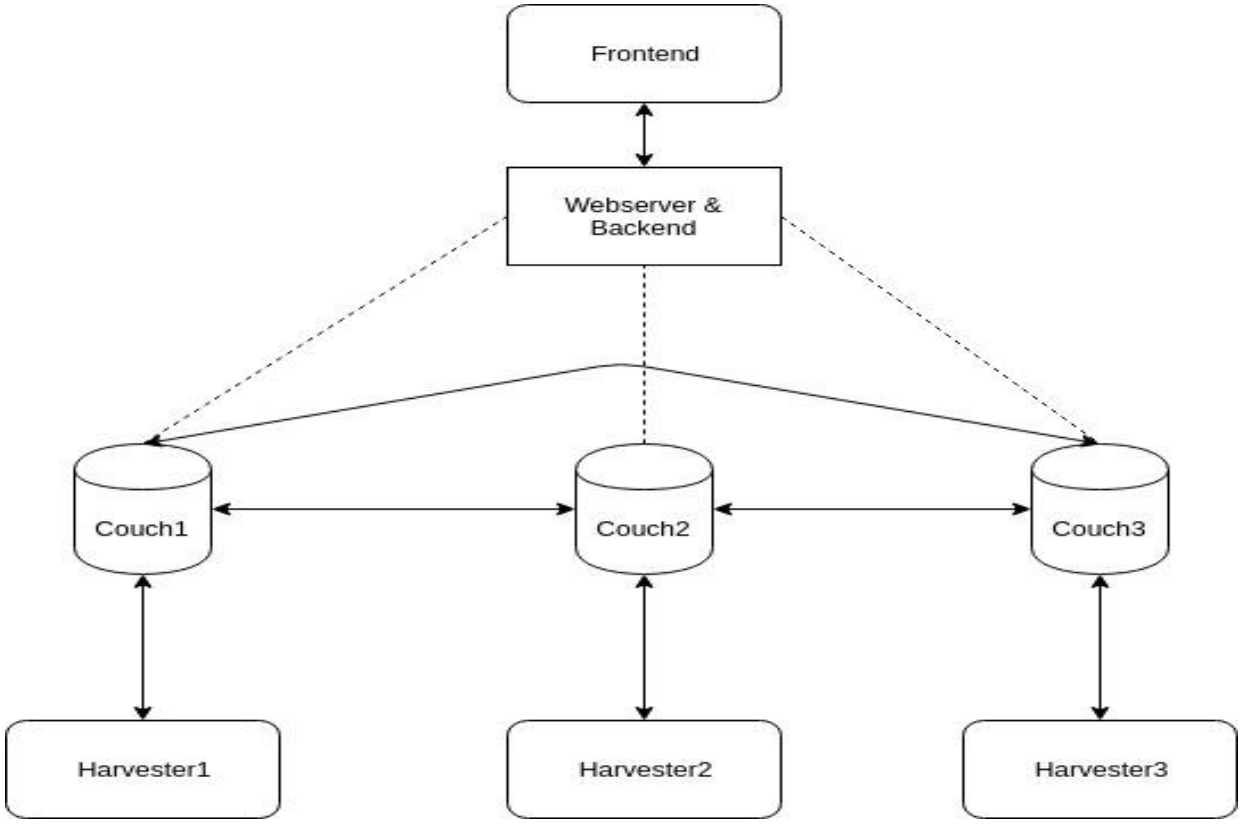


Figure 3-1

However, since the databases are on separate virtual machines, there could be case that they are located on the same physical machine, so if that machine fails, none of the databases would be accessible by the application server (In our case Django). This issue can be resolved by doing the replication on the virtual machines that are in different availability zones to ensure that a single machine failure does not limit access to the data.

When running the harvester processes contentiously to gather twitter data in real-time, a crash in one of them can lead to “missed” tweets that could not be captured within the timespan that the process was down. Yet, this issue does not affect the availability of the system and merely causes temporarily falling behind within the area that the harvester was operating against. When the harvester recovers from the crash it uses “since_id” of the latest tweet it had gathered before the failure to resume capturing tweets.

The current system is prone to shut down in situations that the application server fails. At the first glance, we may come up with the idea of replicating the application servers and use a cluster of reverse proxies for load balancing between the application servers and redirecting the calls from the client to the live application servers. Although it can be helpful in a production environment in which we expect a huge number of client requests, following this approach does not solve the issue of single point of failure, as if the reverse proxy node(s) fails, again the data could not be requested from the databases.

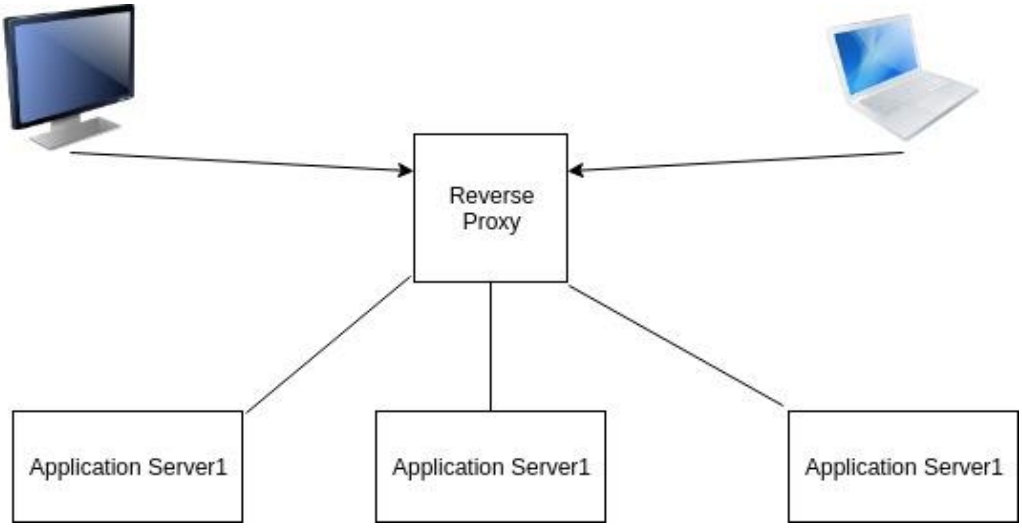


Figure 3-2

3.1 Simple User Guide for Harvester

This harvester script should be run on three NeCTAR servers separately and parallelly.

```
ubuntu@r-cciqgw8g-0:~$ cd harvester/
ubuntu@r-cciqgw8g-0:~/harvester$ nohup python harvesterManager.py 2
```

Figure 3.1-1

You can run it using "**nohup python harvesterManager.py 2**". 'nohup' means this script will continue to run at backend even if you close your terminator. The first parameter '2' indicates that this is the harvester No.2. Thus, this harvester will utilize the second group of information in config files, including different twitter API authentication keys and different searching ranges.

```
ubuntu@r-cciqgw8g-0:~$ cd harvester/
ubuntu@r-cciqgw8g-0:~/harvester$ nohup python harvesterManager.py 2 990000000000
000000
```

Figure 3.1-2

You can also run it using "**nohup python harvesterManager.py 2 990000000000000000**". The second parameter is the since_id for Search API. Harvesters will not search tweets whose ids are smaller than this since_id. It's useful when you collect data for the second round and you probably don't want to search for the tweets earlier than the latest tweet from last round.

The harvester will run after you type in this command. Tweets gathered from Search API as well as Stream API will be written into the database continuously.

3.2 Harvester Workflow

(1) We divide the Melbourne area into three parts. For Search API, filtered areas are shaped as circles and for Stream API, filtered areas are shaped as rectangles. Each harvester processes data from one area for Search API and one area for Stream API.

(2) If a tweet is located within the area of interest, the harvester will try to write it into the database on the corresponding node.

(3) Additionally, if this tweet has ‘coordinates’ attribute, harvester will check in Melbourne SA2 code json file and append the SA2 code to this tweet if it belongs to a valid SA2 area.

(4) Since three couch DBs are kept consistent all the time, whenever a tweet with duplicate id is written into any node, couch DB will raise an exception and this tweet will not be written into the database.

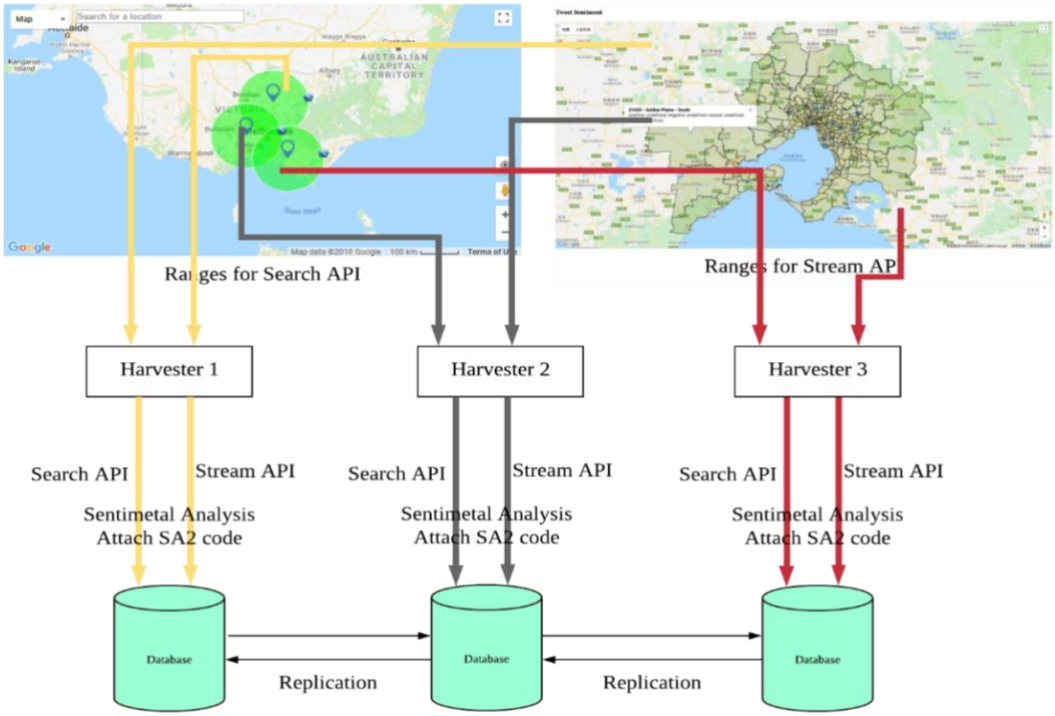


Figure 3.2-1

3.2.1 Config File

```
# processor_1 information_Lu
processor_1 = {}
processor_1['couchdb-admin-username'] = 'admin'
processor_1['couchdb-admin-password'] = 'team38'
processor_1['couchdb-address'] = 'localhost:5986/'
processor_1['couchdb-db-name'] = 'tweet'
processor_1['consumer_key'] = '1RcIwPa92JvKPegrsRzSWzXht'
processor_1['consumer_secret'] = 'AFceGbD4TzRGmqRz8oq65ovHIsAuG7WlwkzfsqEvz5WgYJDoUw'
processor_1['access_token'] = '987908747375726593-yp9g2tc5FJMfr54eIc8mI8mkboX9VNC'
processor_1['access_token_secret'] = 'hH5K46mLTHk5Yn51gesInrq3YYUN5vGOBDjWx8PdBbdEm'
# example:  37.781157 -122.398720 1mi
processor_1['twitter-geo-latlngrad'] = '-36.816,145.137,100.00km'
# Rec starts from SW corner
processor_1['twitter-geo-rec'] = [143.582020, -37.6421353, 145.873154, -37.17361]
```

Figure 3.2.1-1

Inside the config file, we keep our tweet API keys, couch DB authentication information and range of area for each harvester. We have three different groups of config information for three harvesters. We use these configurations in other files of our harvester project.

3.2.2 Harvester Manager File

This is the python script that invokes the whole harvester. The main function of this file will be illustrated as follows:

(1) Handle user’ s input. We require users to type in a harvester number (1, 2, or 3) to indicate which harvester is about to be run. Also, the second parameter can be used to tell Search API the minimum tweet id it is about to search for.
(2) Connect to couch DB on its own node.
(3) Start Search API thread
(4) At the meantime, start Stream API thread

3.2.3 Search API and Stream API

How to work within the quota imposed by the Twitter APIs:

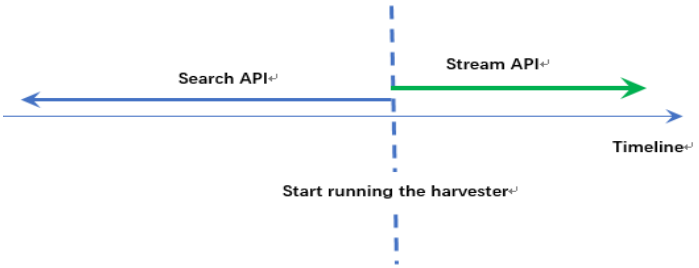


Figure3.2.3-1

We utilize two twitter APIs to gather data, including Search API and Stream API. The main difference is that search API searches for the tweets before the time you run it, while Stream API fetches the real-time tweets after the time you run it.

We pre-process these tweets before writing them into the database. We first filter out tweets outside the range. Then we check whether this tweet covers ‘coordinates’ attribute. If so, we continue to check which SA2 code it belongs to and add the valid SA2 information to this data. For each tweet within the range, we will do the sentimental analysis for the text and append the sentimental polarity value to the data as well.

For tweets without location(above)
For tweets with location(right)


```
"process-info": {
  "sentiment": 0.8,
  "processor-id": "1"
},
```

```
"process-info": {
  "processor-id": "1",
  "sentiment": 0,
  "properties": {
    "SA2_NAME16": "Glen Waverley - West",
    "GCC_NAME16": "Greater Melbourne",
    "SA3_CODE16": "21205",
    "SA3_NAME16": "Monash",
    "SA2_5DIG16": "21322",
    "STE_CODE16": "2",
    "SA2_MAIN16": "212051322",
    "SA4_CODE16": "212",
    "STE_NAME16": "Victoria",
    "AREASQKM16": 7.6758,
    "SA4_NAME16": "Melbourne - South East",
    "GCC_CODE16": "2GMEL",
    "sentiment": 0
  }
},
```


Figure 3.2.3-2

Both APIs set limit for the number of requests every 15 minutes per user. However, we applied some methods to collect more useful data within limited time

(1) For Search API, we use ‘Application Only Auth’ instead of ‘Access Token Auth’. Thus, we can search at a rate of 45,000 tweets/15 mins. If we use Access Token Auth, we can only get 15,000 tweets/15 mins.

```
auth = tweepy.OAuthHandler(self.twitter_consumer_key, self.twitter_consumer_secret)

self.api = tweepy.API(auth, wait_on_rate_limit=True,
                      wait_on_rate_limit_notify=True)
```

Figure 3.2.3-3

(2) For Stream API, if we find a tweet has ‘coordinates’ attribute, we will trace this user’s timeline and get this user’s most recent 100 tweets. This help us get more tweets with information of location.

```
if tweet and tweet['coordinates'] and tweet['coordinates']['coordinates']:

    # This is a way to get the 100 recent tweets of the people who have included their geolocation in
    # their tweets and stores them as well
    resp = self.api.user_timeline(user_id=tweet['user']['id'], count=100)
    for status in resp:

        status_json = status._json
```

Figure 3.2.3-4

(3) Also, our three harvesters run at the same time. Each harvester has different twitter API key, which means three Search APIs and three Stream APIs can work independently on the cloud to collect data. Thus, we can fetch data at a faster speed.

3.3 Map Reduce

As mentioned in the previous sections, a harvester process checks if the tweet contains coordinates field and adds an extra field with the following format before storing it to its associated database instance:


```
"process-info": {
  "processor-id": "1",
  "properties": {
    "sentiment": 0,
    "GCC_NAME16": "Greater Melbourne",
    "SA2_NAME16": "Mitcham (Vic.)",
    "AREASQKM16": 6.6271,
    "GCC_CODE16": "2GMEL",
    "STE_CODE16": "2",
    "SA2_MAIN16": "211041270",
    "SA4_CODE16": "211",
    "STE_NAME16": "Victoria",
    "SA3_NAME16": "Whitehorse - East",
    "SA4_NAME16": "Melbourne - Outer East",
    "SA3_CODE16": "21104",
    "SA2_5DIG16": "21270"
  },
  "sentiment": 0
}
```


Figure 3.3-1

These fields then will be queried using CouchDB map functions and built-in Erlang reduce functions to give us the desired data.

The following map function emits SA2 code of each tweet as a key and the sentiment as the value.

```
function (doc) {
  if(doc['process-info']['properties'] && doc['coordinates']){
    emit(doc['process-info']['properties']["SA2_5DIG16"], doc['process-info']['sentiment']);
  }
}
```

Figure 3.3-2

Built-in _stats reduce function together with setting group =True then can be used to get sentiment sum of each area as well as the number of tweets within the district and the client code can then calculate the average sentiment of the tweets within that neighborhood by dividing them.

```
function(head, request) {
    var polarity_count = {};
    var determine_polarity = function(val) {
        if (val < -0.1) {
            return 'negative';
        } else if (val >= -0.1 && val <= 0.1) {
            return 'neutral';
        } else {
            return 'positive';
        }
    };
    while (row = getRow()) {
        var polarity = determine_polarity(row.value);
        if (polarity_count[row.key]) {} else {
            polarity_count[row.key] = {
                'positive': 0,
                'negative': 0,
                'neutral': 0
            };
        }
        polarity_count[row.key][polarity] += 1;
    }
    send(JSON.stringify(polarity_count));
}
```

Figure-3.3-3

The following list function can be applied and fed with the previous view results by setting parameter reduce=False to yield the count of each “neutral”, “positive”, and “negative” categories within each area.

- General Mood Discovery

This map function categorizes tweets based on the polarity of their content to three classes of “neutral”, “positive”, and “negative”. Again, the optimized _count reduce function lets us aggregate the results about the number of each category. In contrast to the previous list function that was employed to return polarity for tweets containing coordinates, this query operates on all tweets no matter if they include the coordinates or not. Since the harvester parameters had been set to just gather tweets within Melbourne city area, the output would give us a more general idea of Melbourne residents moods.

```
function (doc) {
   if (doc['process-info']['sentiment']){
     if(doc['process-info']['sentiment'] < 0.1 && doc['process-info']['sentiment'] > -0.1){
        emit('neutral' , 1);
     }
     else if(doc['process-info']['sentiment'] < 0.1){
       emit('negative', 1)
     }
     else{
       emit('positive', 1)
     }
  }
}
```

Figure 3.3-4

- Timely Data

The following two map functions allow us to check the sentiments of the tweets based on the time of the day and the weekday on which the tweet was posted. _stats reduce function helps us to extract the average mood of people based on the time of the day and different weekdays easily.

```
function (doc) {
  if (doc.created_at && doc['process-info']['sentiment']){
     var d = new Date(doc.created_at);
    emit(d.getHours() , doc['process-info']['sentiment']);
  }
}
```

Figure 3.3-5

```
function (doc) {
 if (doc.created_at && doc['process-info']['sentiment']){
     var d = new Date(doc.created_at);
    emit(d.getDay() , doc['process-info']['sentiment']);
  }
}
```

Figure 3.3-6

However, an alternative approach to the previous two queries is to define a single view that its keys are consisted of [weekday, hour] pair.

```
function (doc) {
  if(doc.created_at && doc['process-info']['sentiment']){
  var d = new Date(doc.created_at);
    emit([d.getDay(), d.getHours()] , doc['process-info']['sentiment']);
  }
}
```

Figure 3.3-7

By setting group_level=1 parameter in the request header sent to CouchDB, we may reach the same results as the previous separate view for weekdays. Nevertheless, reaching the result similar to that of the hours view cannot be done via setting a group_level. So as to utilize the

same output, a list function can be defined to process the output of the map function and yield the same information:

```
function (head, request){
  var counts= {};
  var hours = {};
  while(row=getRow()){
    if(hours[row.key[1]]){
      hours[row.key[1]] += row.value.sum;
      counts[row.key[1]] += row.value.count }
    else{
      hours[row.key[1]] = row.value.sum;
      counts[row.key[1]] = row.value.count;}
  }
  var res = {'hours_sum':hours, 'counts': counts };
  send(JSON.stringify(res));
}
```

Figure 3.3-8

The aforementioned list function mimics the behaviour of _stats function with two fields of sum and count only using the second field of the key (hours of day) and sends a JSON string containing the sum and count of the tweets sentiments for each hour. This data can be then used for determining the average sentiment for every particular point of time.

3.4 Aurin Data Collection

We checked the available datasets in Aurin portal and selected the scenarios we were interested in. We downloaded the data on **average income per person per week, medium age, how many people living with children, the number of volunteers and how many people received high education**. All the data are counted according to SA2 district. Then, we calculated the values we tended to use and merged the useful data into one json file. The key is SA2 code. We uploaded this json file onto couch DB as well.


```
{
  "key": "212031302",
  "value": {
    "medium_age": 31,
    "higher_education_percentage": "10.3231%",
    "average_income_per_person_per_week": 668,
    "having_children_percentage": "85.2738%",
    "volunteer_total_num": 1903
  }
},
```


Figure 3.4-1 Scenario Analysis Attributes

3.5 Error Handling and Reflections

3.5.1How do we handle duplicate tweets?

Our harvester is designed to catch as many possible exceptions as we can. Whenever an error occurs, we write the error message to logs and try not to affect other part of application. For example, when the harvester tries to write a duplicate tweet into the database and the database

throws an error, the harvester will catch it and write this information into logs. But it will not stop the application, the harvester continues to read and write next tweet. Also, if any tweepy error occurs during the run time, it will leave a log message and fail gracefully. For some error, like user's input error, we print out a message like 'Processor Number should be given' to remind users to add necessary parameters.

There are indeed some single points of failure in this harvester. But they are harmless and the whole application cannot proceed with these errors. For example, if you type in the wrong couch DB ip address, this harvester will stop running and raise an error message. However, it is tolerable since it is not a web server application, which requires high reliability and availability.

3.5.2 Limitation of Language Processing

Sentimental analysis software tools like TextBlob utilize machine learning, statistics to analyse the natural language. They often base the analytic results on large data from web pages, online groups, social media, reviews, etc. However, there are some limitations on these tools. So far, computer systems cannot easily recognize negations, exaggerations, jokes and sarcasm in the natural language. They make errors from time to time since their understandings are always too naïve and straightforward. There exist several solutions for the future. The first approach is to take into account the relevant context. So, when the computer tries to analyse the sentence, it will consider the sentences before or after it. The second is to gather an even larger number of natural language data. Thus, these data can offer clues to see whether a sentence is sarcastic or not. Additionally, sentiment is inherently subjective to person. Thus, for our project, a small number of data cannot tell anything about the mood of people in the district. Only the aggregation of big data matters.

3.5.3 Issues we encountered

When we implemented harvesters, we faced some problems. Fortunately, we solved them at last. At first, we didn't know we should divide different districts using SA2 code. We used postcode to distinguish a certain district. Thus, when we pre-processed data, we appended postcodes to all tweets with 'coordinates' attribute. We collected data for the first round and got around 1.3 million tweets. Then we discovered this problem and had to update our database since we should append SA2 code instead of postcode. We ran the updating script for about two days to update all the data, which was frustrating and time-consuming.

Secondly, since we got some old versions of data in database, storage space for database became full immediately when we started to gather data for the second round. And we tried to compact the database. However, there had already been no free space for compacting. Finally, we had to attach more volume to database and then ran compact command to release space on databases.

Thirdly, our target was to gather 3 million data. We managed to gather 2.8 million data using our harvesters. We had to use 0.2 million data from lecturer's couch DB. In order to keep data format consistency, we had to append SA2 code attribute to those data before storing them to our own database.

3.5.4 Where we worked well

We started to design the framework early and we got around two weeks to gather data. Thus, we collected a huge number of tweets (2.8 million) using our harvesters. Also, when we faced problems in databases, we reacted quickly, and those problems didn’ t affect our results.

4 Frontend Architecture

4.1 Architecture Design

The web architecture of our tweets analytics application falls in to a client-server design. The front end employs multiple powerful data visualisation technologies and JavaScript libraries including Google Map API, Plotly.js, Bootstrap, jQuery and AJAX. The team chose the Google Map API because this application processes geo-special data at a million level. In particular, the team has made their decision to focus on the data visualisation and analytics in the wide Melbourne area. Given the detailed geo-spatial data provided by AURIN, the team consented that employing Google Map API with its outstanding visualisation capabilities would feed intuitive information to the potential users of the application. It can convey and establish in-depth insights and knowledge about the Twitter users in Melbourne.

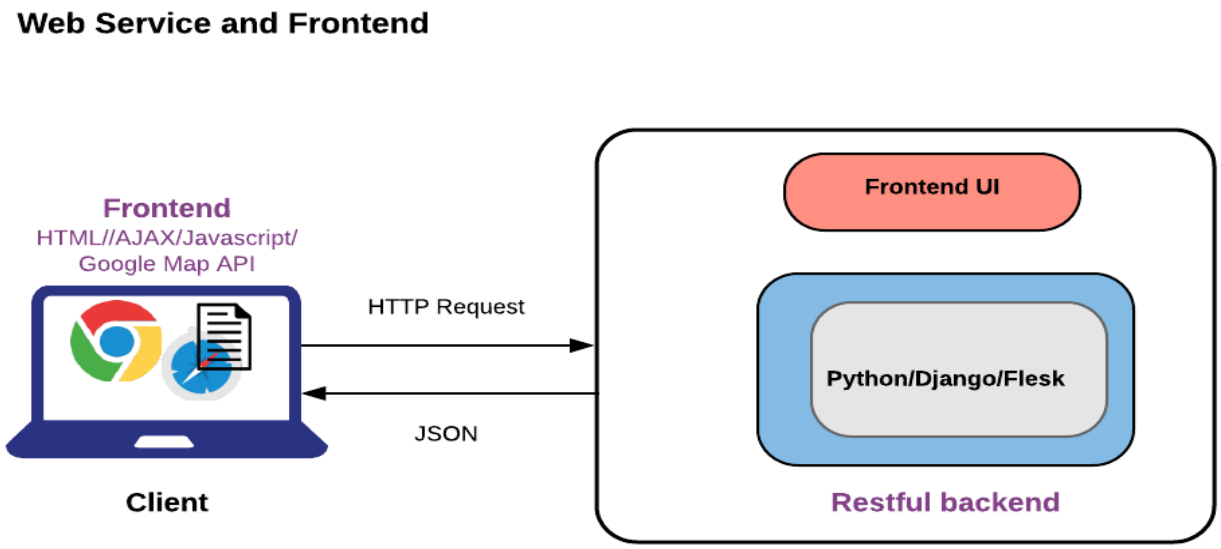


Figure 4.1-1

In addition to that, team also found that Plotly glows with its extraordinary support towards in-depth data analysis and beautiful yet practical visualisation. Particularly, Plotly speaks for itself with its outstanding capabilities of performing scientific analysis and big data visualisation given we are aiming to handle millions of tweets in an active manner. Bootstrap was chosen by the developers because of its strong ability to quickly prototype and always keep HTML elements organised. Besides that, the team also adopted jQuery and AJAX [3]for view-side manipulation, smoothing, caching and thus greater user experience.

```
var totalJson = $.ajax({
    url: "http://115.146.85.254/total_sentiment/",
    async: true
});

var hourJson = $.ajax({
    url: "http://115.146.85.254/hours/",
    async: false
});

var weekJson = $.ajax({
    url: "http://115.146.85.254/weekdays/",
    async: false
});
```

Figure 4.1-2

Our tweet analytics application adopted the RESTful design in multiple ways. Firstly, the web architecture employs the client-server model. By separating the user interface concerns from server & data storage concerns, we greatly increased the portability of the view components and gave both the user interfaces and the server components great flexibility and scalability. Secondly, our client communicates with the server in a stateless manner. The server treats each client request independently. No state information will be stored by the server and in each request the client provides all the information necessary to service its request. Thirdly, we flagged the cacheability in our responses from the server to the client. This reduces the unnecessary interactions between the server and the client. Therefore, scalability and performance on either end were greatly improved. Fourthly, we implemented the web server as an intermediary component within our entire design. Hence, the system became 3 layered and encompasses much flexibility for scaling with load balancing and shared caching abilities. It also enforces the security in a sense. Fifthly, we realised the capability of the server to send back executable code to a client by implementing the Django and Flesk frameworks on the server end. Lastly, our tweet analytics web application exposed our data to any potential client via a RESTful design. The interfaces are uniform, and no authentication or authorisation is required. In addition, the resources are encoded with sufficient self-descriptive messages/metadata for clients to manipulate.

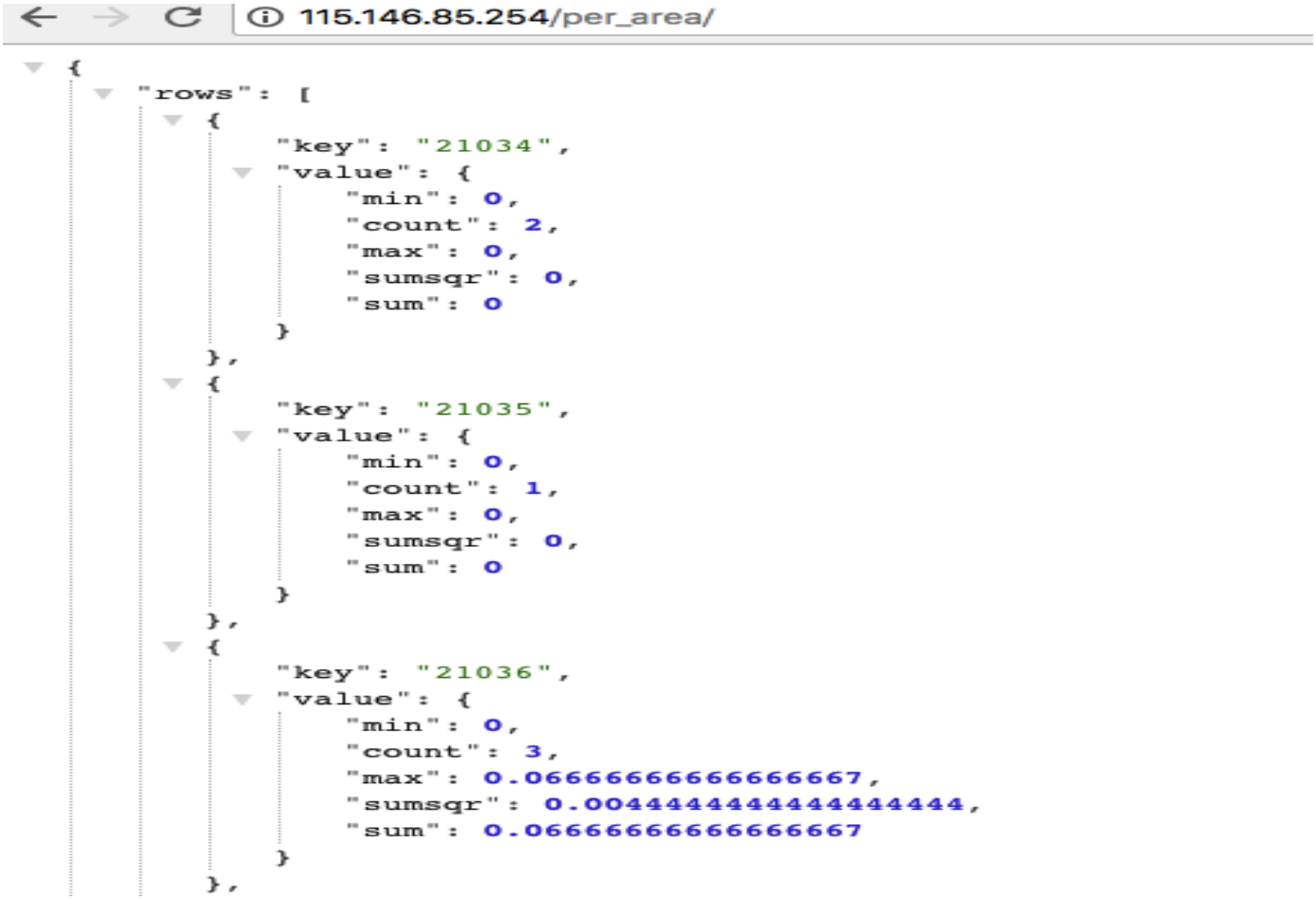


Figure 4-3

4.2 Google Map API

To demonstrate our sentiment measurement results and five scenarios directly, Google Maps API is used to show them on maps.

To draw the polygons of each district, first download Australia State Boundary Shapefile from official website (https://data.gov.au/dataset/sa-state-boundary-psma-administrative-boundaries). And then, load it into QGIS, which is a professional GIS Software. As is shown in Figure4.2-1, the green part is the whole Australia data. And by doing filtering away untargeted areas with the process shown in Figure 4.2-2, the targeted Melbourne area is obtained as shown as the small purple area in Figure 4.2-1. Finally, as shown in Figure 4.2-3, transfer the Shapefile into Geojson file, which could be used for loading the Google Maps.

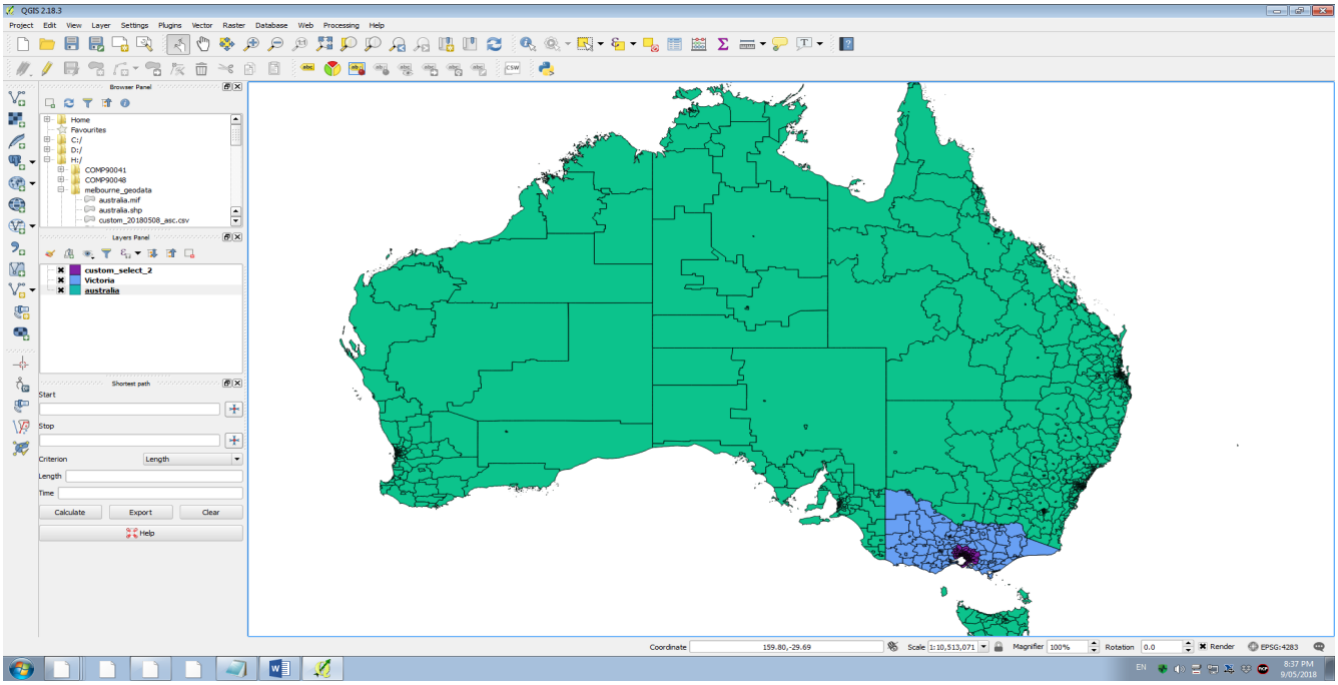


Figure 4.2-1

. Australia (green), Victoria State (blue) and Melbourne (purple) shown in QGIS

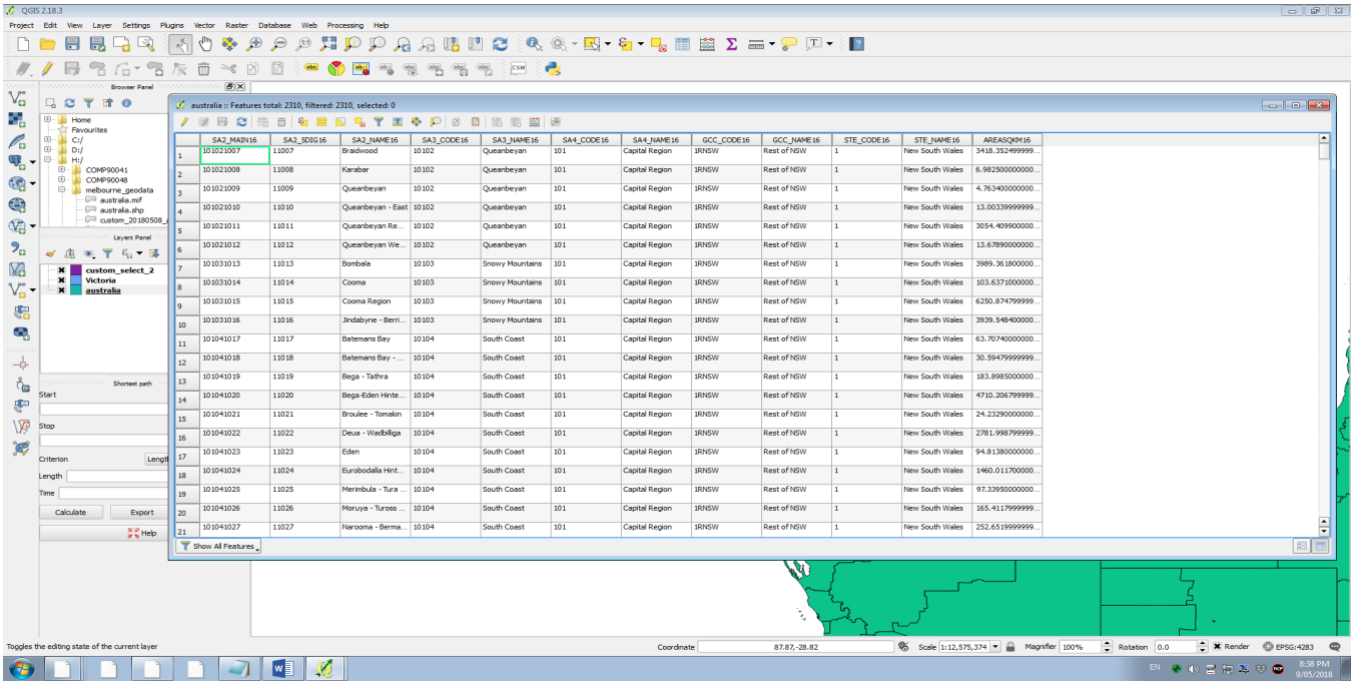


Figure 4.2-2

Filtering away untargeted areas using attributes

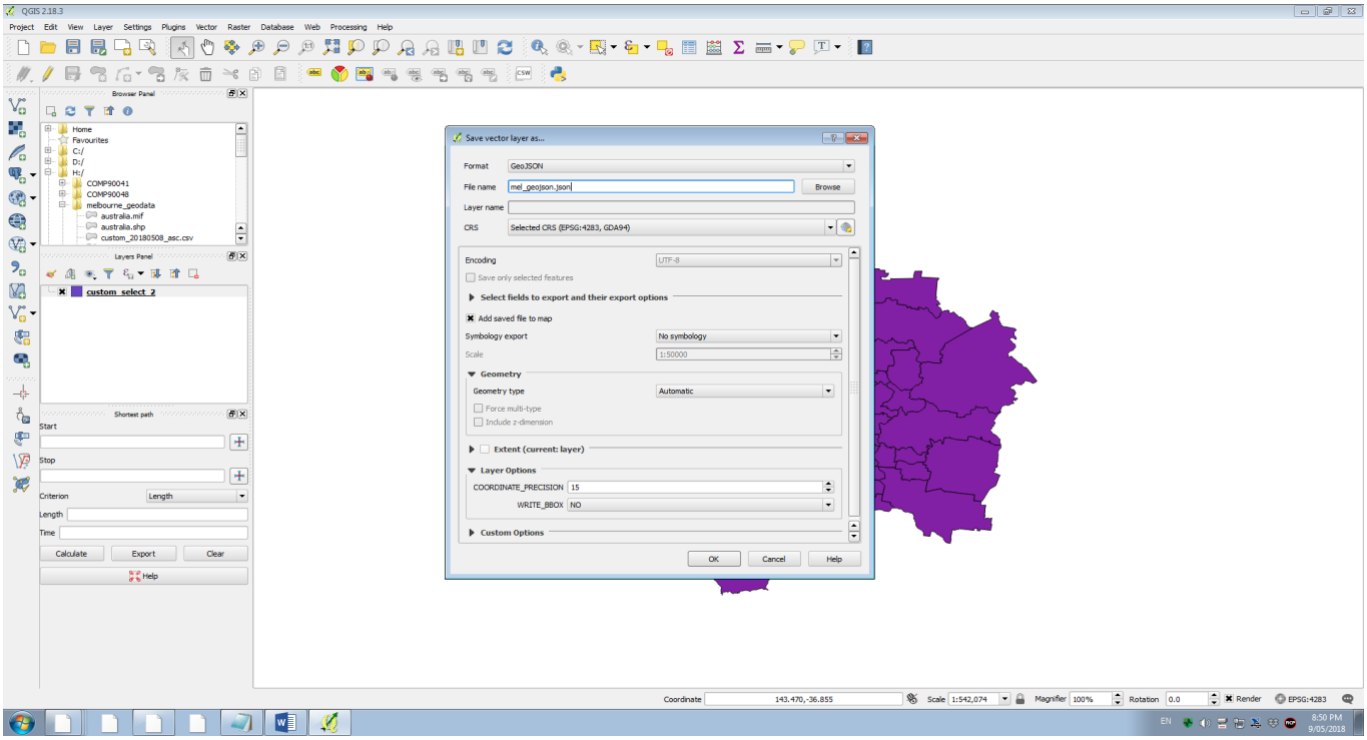


Figure 4.2-3 Transfer the Shapefile into Geojson file

There are two data sources, one for ‘Sentiment’, with another for the five scenarios (‘Income’, ‘Age’, ‘Education’, ‘Children’, ‘Volunteer’).

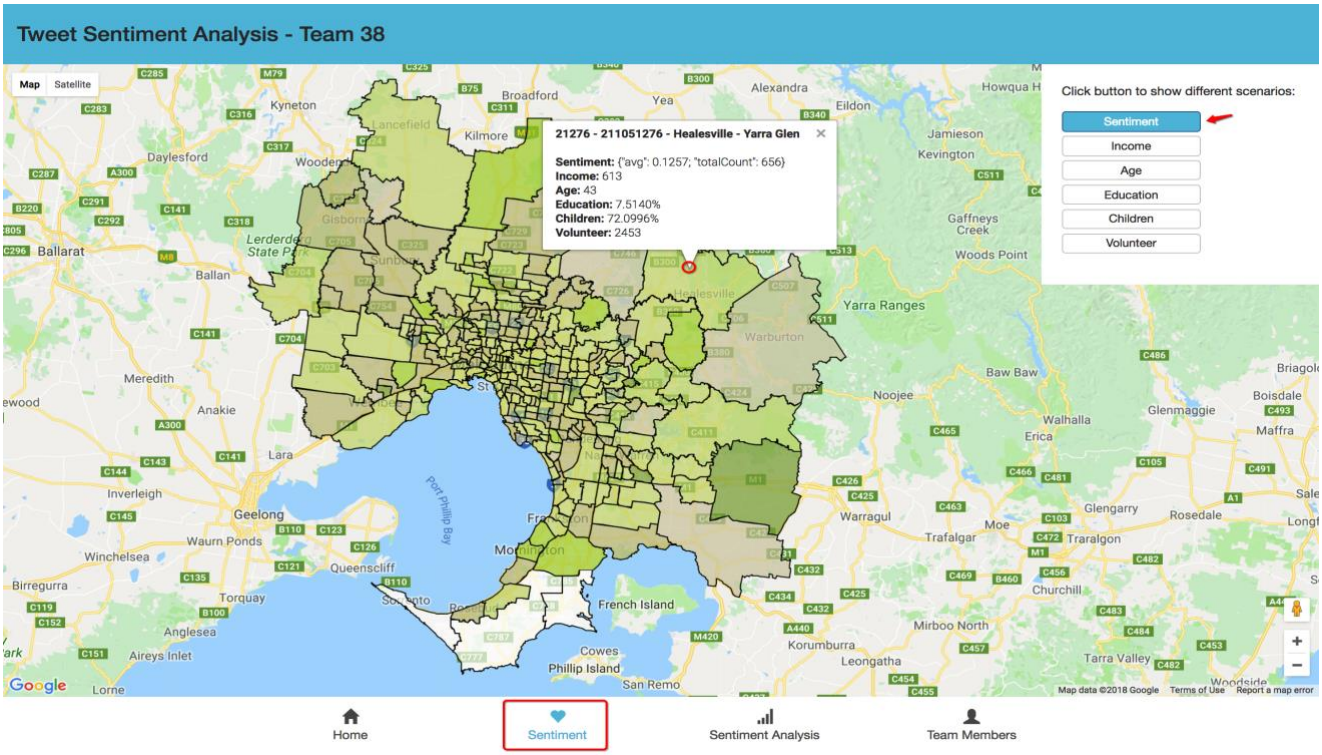


Figure 4.2-4 Map for ‘Sentiment’

As is shown above graph, click the ‘Sentiment’ button on the bottom navigation bar of the page, firstly the map for sentiment results is shown. The colored areas are where we chose to

study and different colors indicate the average sentiment for each district. The darker the color is, the higher (which means more positive) average sentiment the district has. The white color means there is no data gathered in this district. Move the computer mouse above the colored areas, a popup message box will come out, which shows the all the detailed information in the district.

By clicking the six buttons on the top right corner of the page, users could toggle between maps for sentiment results and five scenarios. For each map, the indication of color shading is the same as that in ‘Sentiment’ map. Also, computing mouse hanging over colored areas will stimulate the popup message box.

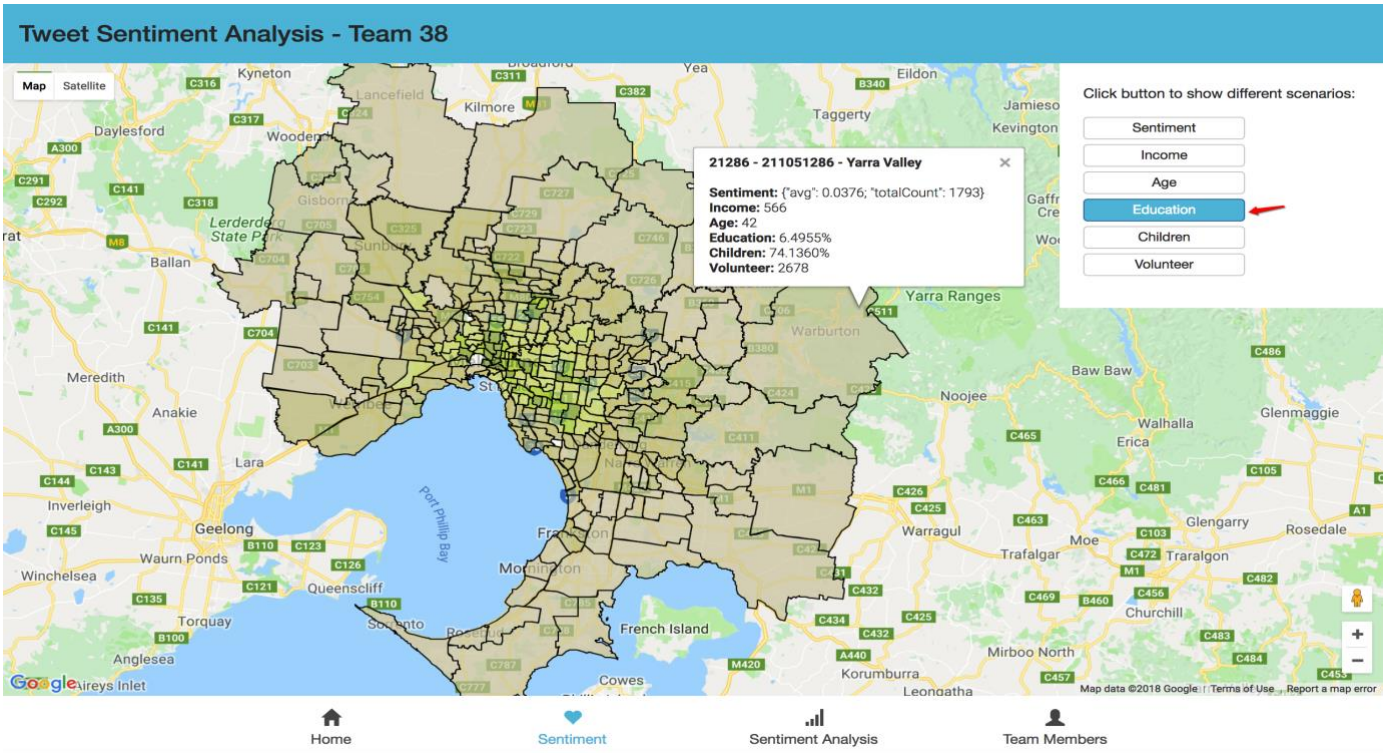


Figure 4.2-5 Map for ‘Education’ Scenario

Take ‘Education’ Scenario for example, which is shown above graph. It is apparently shown in the map that compared to people who are near the CBD of Melbourne, less people in outskirts has received higher education.

5 Scenario Analysis

We selected 5 scenarios, and explored relationship between happiness and other contributing factors like income, age, children, education and volunteer. The data source is as below:

Scenario	Aurin Data
Income	sa2_g02_selected_medians_and_averages_census_2016
Age	sa2_i04_selected_medians_and_averages_census_2016
Children	sa2_p25_family_composition_census_2016
Education	sa2_g15_type_of_edu_institute_by_age_by_sex_census_2016
Volunteer	sa2_g19_voluntary_work_by_age_by_sex_census_2016

5.1 Statistic Overview

- Pie Chart

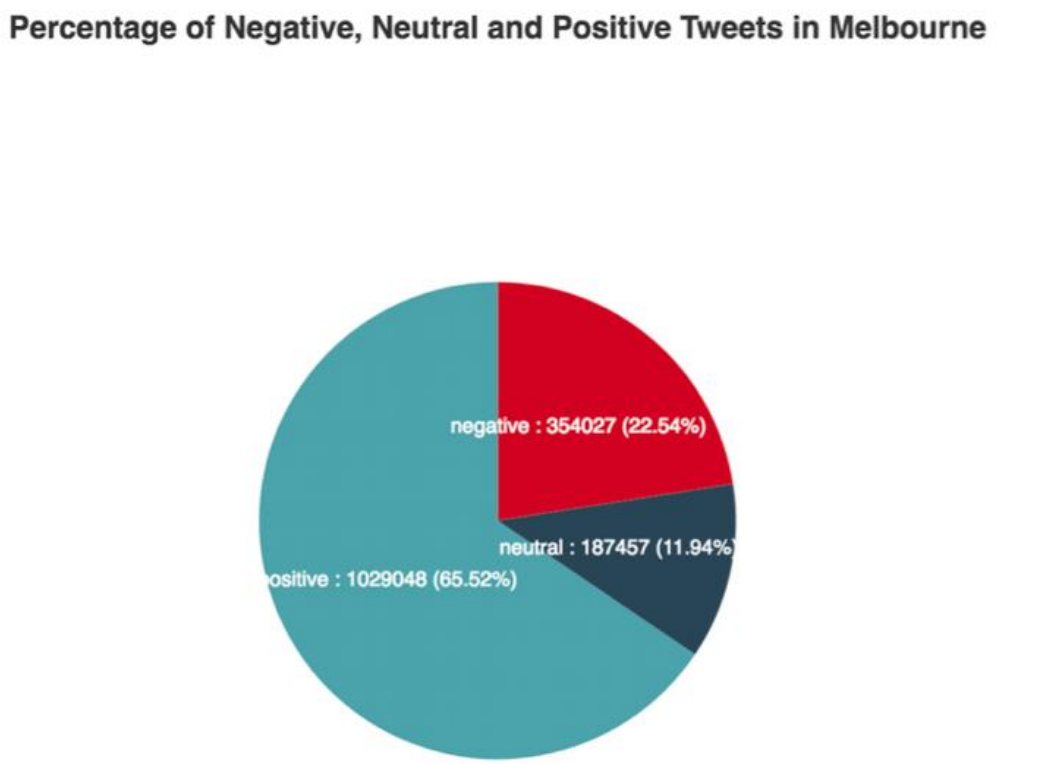


Figure 5.1-1

This pie chart shows the percentage of negative, neutral and positive tweets in Melbourne. Most tweets (65.5%) in Melbourne reflect a positive mood. Only 22.5% show a negative mood, which account for 1/4 of the total tweets.

- Hourly Sentiment

In this chart, the line shows the hourly sentimental change in Melbourne while the bars indicate the count of tweets by hour.

Sentiment polarity value for each hour is relatively stable, ranging from 0.18 to 0.21. We can infer that most people have a positive mood in Melbourne. No wonder Melbourne is the most liveable city in the world! As we can see clearly from the chart, there are slight differences on people’s mood from hour to hour. Sentiment polarity value slightly declines from 12am to 12pm. People at 12 pm have the worst mood of a day. Then they seem to become happier and happier from 12pm to 8pm. People’s sentiment polarity reaches a peak at 8pm. We can infer that they may have the best mood when they start to enjoy night life at 8pm.

As to the tweet count per hour, we can see the trend clearly from those bars. From 12am to 12pm, there are approximately 85,000 tweets per hour. However, the count drops sharply from 1pm to 6pm. There are only 20,000 tweets at 6pm. This is probably because people are busy at work and have no time to send tweets. After 6pm, tweet count increases slowly again.

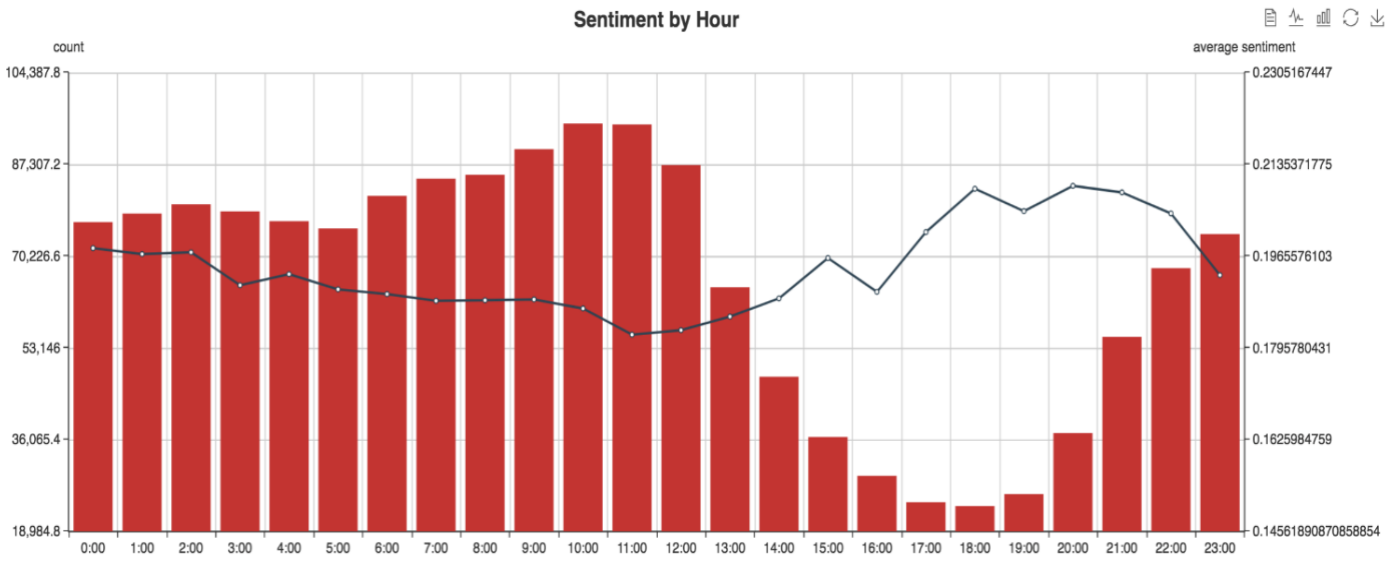


Figure 5.1-2

- Weekly Sentiment

In this chart, the line shows the daily sentimental change in Melbourne while the bars show the tweet count every day in a week.

We can see people’s moods are relatively stable from Monday to Sunday, ranging from 0.17 to 0.2. There is a slight decrease from Wednesday to Thursday. And people become happier again from Thursday to Friday. It is very interesting to see that people are happier on Friday than Saturday. But their moods reach a peak on Sunday. It seems that people enjoy Sunday and Friday most!

Meanwhile, tweet count is relatively stable but low from Tuesday to Thursday, which is around 180,000 per day. However, from Friday to Sunday, tweet count increases significantly. Tweet daily count reaches a peak at around 260,000 on Sunday and Monday. We can infer that people in Melbourne tend to send more tweets during the weekend. People also send a lot of tweets on Monday!

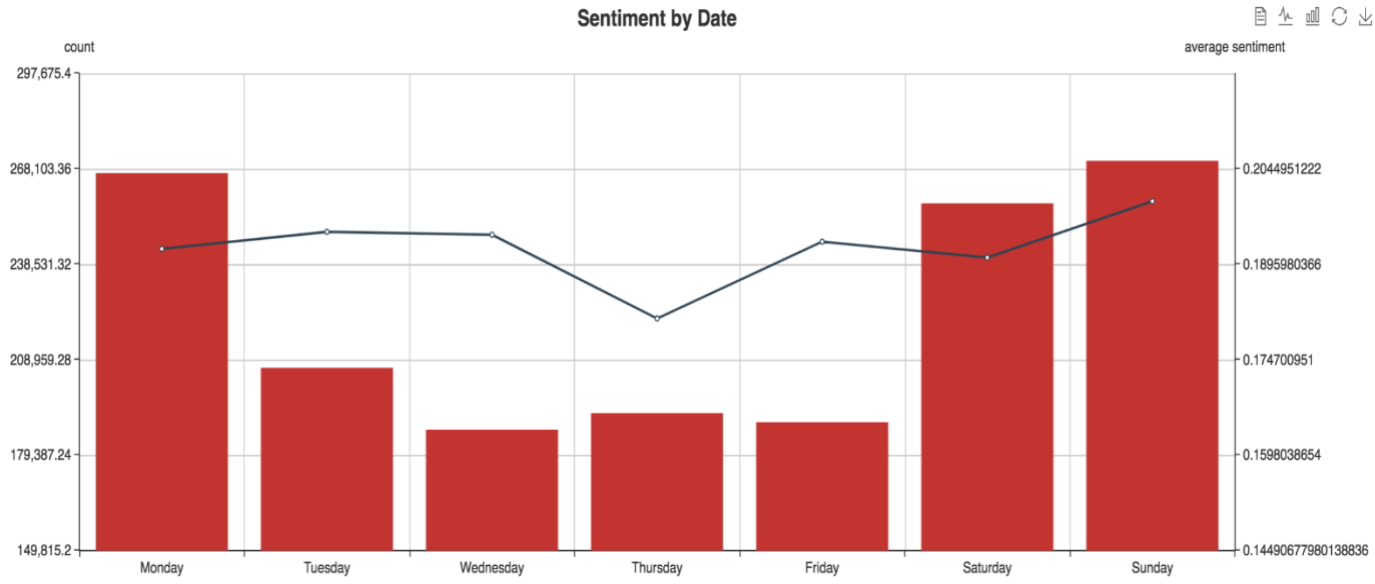


Figure 5.1-3

5.2 Income

The reason why we want to explore the relationship between wealth and happiness is in common sense, people who are with a large amount fortune may be happier than people who earn a modest income. Because rich people don’ t need to worry about money, and they can enjoy life as they want. Conversely, poor people need to make a budget for everything, and their life is limited by income, which may cause unhappiness.

Big data is a good tool for us to explore this question. We implemented sentiments analysis for 3 million tweets in whole Melbourne during last 2 weeks, compared these data with Aurin, and generate the scatter plot as below, the spot stands for each SA area:

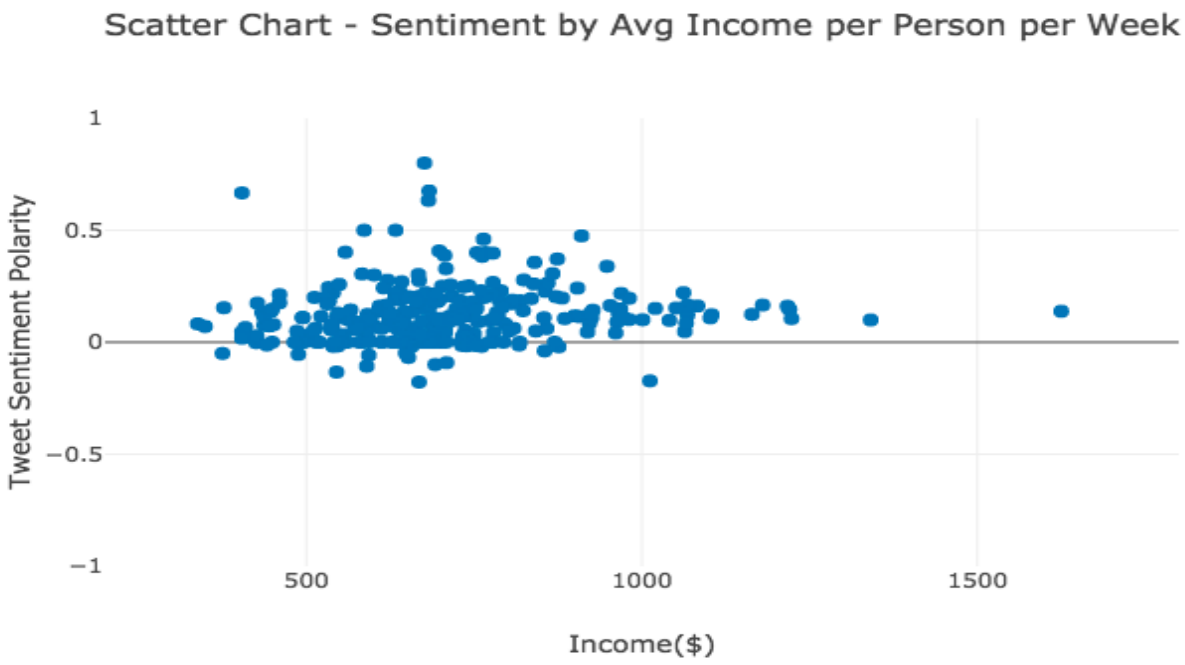


Figure 5.2-1

From chart, most spots distribute between income 500-1000. We can get a conclusion that the sentiment scores of areas which income range is from 500 to 1000 are relative higher.
We calculated the average income and sentiment in Melbourne, data is as below:

region	average income	average sentiment
melbourne	697	0.12

Figure 5.2-2

The average income is 697(per week), from chart we can see the densest part is in average income, around $700, most sentiment scores of this region are between 0 to 0.5 . The highest value is also generated in this range, there are some negative points in this range as well. Thus, the average income range contains various polarity situation.

We ranked the source data by income and sentiment, and list top10 areas respectively.

top 10 area (rank by income)		
area	**income**	**sentiment**
Port Melbourne Industrial	1625	0.1381
East Melbourne	1341	0.0993
Port Melbourne	1223	0.1054
South Yarra - West	1220	0.1463
Toorak	1217	0.1606
Albert Park	1180	0.1659
Elwood	1164	0.1244
South Yarra - East	1104	0.1226
Richmond (Vic.)	1102	0.1089

Figure 5.2-3

This result demonstrates the sentiment is not in proportion to income, and the highest sentiment score doesn’t appear in the rich area.

top 10 area(rank by sentiment)		
area	**income**	**sentiment**
Epping - West	676	0.8
Croydon South	683	0.675
Broadmeadows	404	0.6667
Templestowe	682	0.6333
Box Hill North	633	0.5
Moorabbin Airport	0	0.5
Mill Park - South	586	0.5
Surrey Hills (West) - Canterbury	910	0.475
Niddrie - Essendon West	764	0.4603

Figure 5.2-4

We found the highest score 0.8 is in Epping-West, which has an average income. This finding is compatible with the scatter plot.

From the data, we speculate rich people may be under huge pressure, and easy to be stressed out, which may hardly trigger a higher mood. In contrast, people who earn an average income are ordinary people, they focus on family more, and have time to accompany with family and friends, which may provide them a good mood, and they are easier to show happiness in tweet.

5.3 Age

We chose age as a scenario, because we want to explore if the happiness is in proportion to age, which group of people are happier, younger or older.

From big data, we generated scatter chart as below:

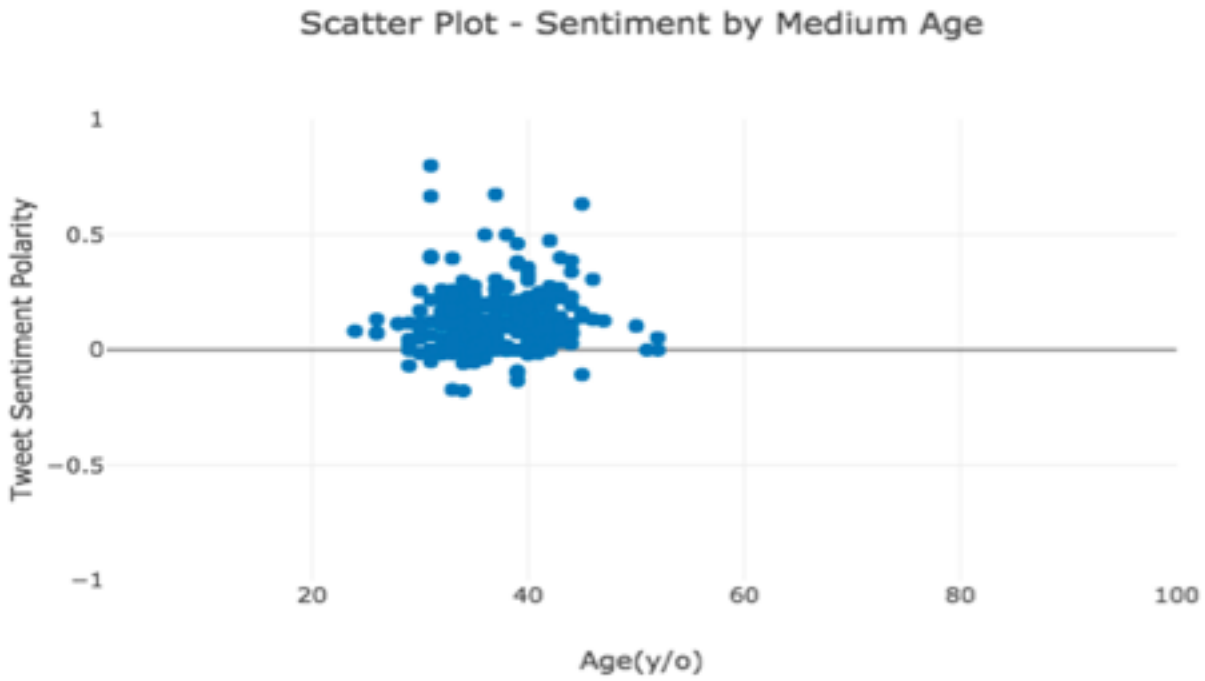


Figure 5.3-1

Most spots distributed between age range from 30 to 50, and we get the conclusion that the majority people in Melbourne areas are middle-aged people, and their mood is various from negative to positive, but most areas with middle-aged people are happy, and sentiment scores are above average 0.12.

We ranked the data by sentiment, and list top 10 areas:

top 10 areas(rank by sentiment)		
	age	sentiment
Epping - West	31	0.8
Croydon South	37	0.675
Broadmeadows	31	0.6667
Templestowe	45	0.6333
Box Hill North	38	0.5
Moorabbin Airport	0	0.5
Mill Park - South	36	0.5
Surrey Hills (West) - Canterbury	42	0.475
Niddrie - Essendon West	39	0.4603
Burnside Heights	31	0.4074

Figure 5.3-2

From data, we can see areas where medium age between 30 to 50 are easier to generate high sentiment score.

The average status of Melbourne is as below:

region	medium age	average sentiment
Melbourne	36.6	0.12

Figure 5.3-3

We speculate middle-aged people have stable family, income, and like to involve in social activities, they are easier to have good mood.

5.4 Children

We want to study this scenario since we are interested in the relationship between happiness index and people living with children percentage in each district. It is interesting to know whether people are happier or not when they live with their kids. There are several factors behind this scenario which may affect people's mood. For example, if the children are too naughty or noisy at home, people may have a worse mood. On the other hand, if the children are cute and well-behaved, people may have a better mood.

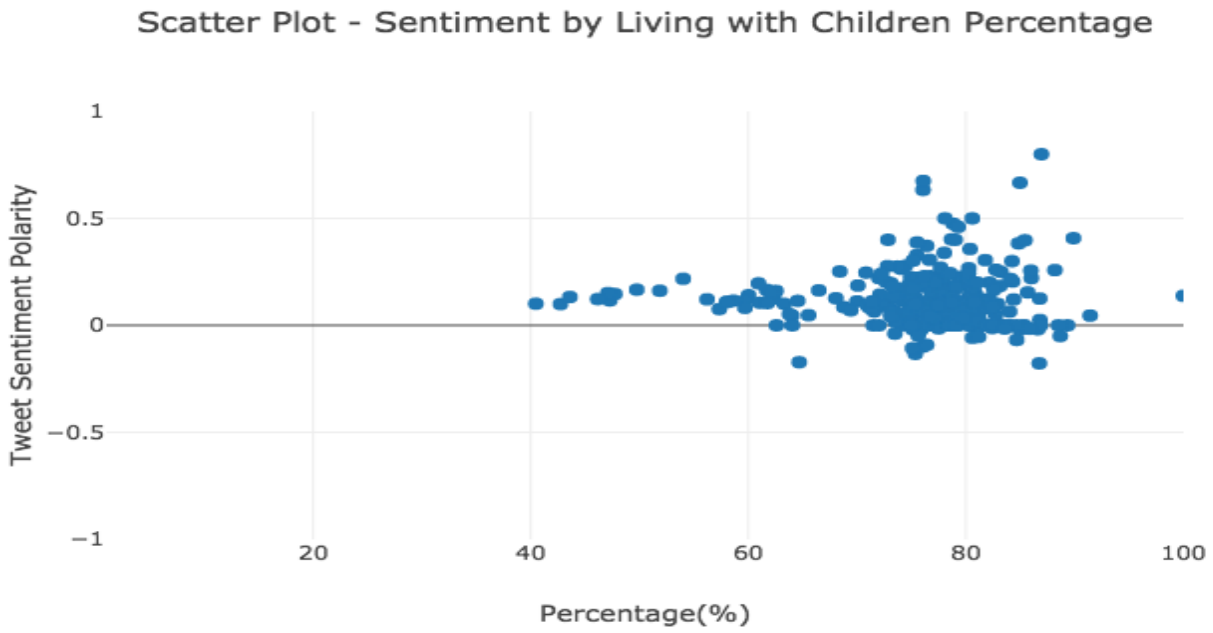


Figure 5.4-1

As we can see from this scatter plot, 60% to 90% of people are living with kids. For most areas, around 80% of people have children at home. For some other areas, only 40% to 60% people have children at home. In districts with more people living with children, people's average sentimental polarity fluctuates a lot, ranging from -0.2 to 0.9. Meanwhile, in districts with less people living with children, people's average sentimental polarity tends to be more stable, ranging from 0.1 to 0.3. We can infer that when people live with kids, their mood appears a lot more unstable than people living without children. The reason is probably that people's mood changes a lot according to kids' different behaviours, which are unstable inherently.

5.5 Education

Our interest in this scenario originates from the usage of Twitter in Donald Trump's election campaign and presidency. The news media often say that President Trump won his election because he maximized Twitter's viral power in his campaign to reach the broad white working class who are potentially his strongest supporters. Even after the election, as a

president, Donald Trump still uses Twitter frequently to promote his strategies and opinions. We are interested in knowing the composition of the regular Twitter users, their sentiment when using Twitter and the correlation between their usage of Twitter and their education level.

Overall, the percentage of the population with higher education in Melbourne areas varies significantly from 2% to 71%. At the same time, the sentiment amongst these areas also demonstrates a sparse distribution, which ranges from -0.18 to 0.8.

In most areas, the percentage of the population with higher education is between 5% and 40% as per the graph. It indicates that the distribution of this population in most areas is not largely different. However, in a few areas, the percentage of this higher educated population can take up to around 71%, which is extremely rare in most residential areas. Nonetheless, from the sentiment perspective, people in those areas do not seem to be living a happier life but perhaps a more secure and peaceful one; their sentiment is even below the average. On the other hand, people in districts with a higher education percentage of 15% to 35% tend to have severer emotional fluctuations. Their sentiment has a wide range, with -0.18 the lowest and 0.8 the highest. They seem to live a more dynamic and turbulent life that affects their mood greatly.

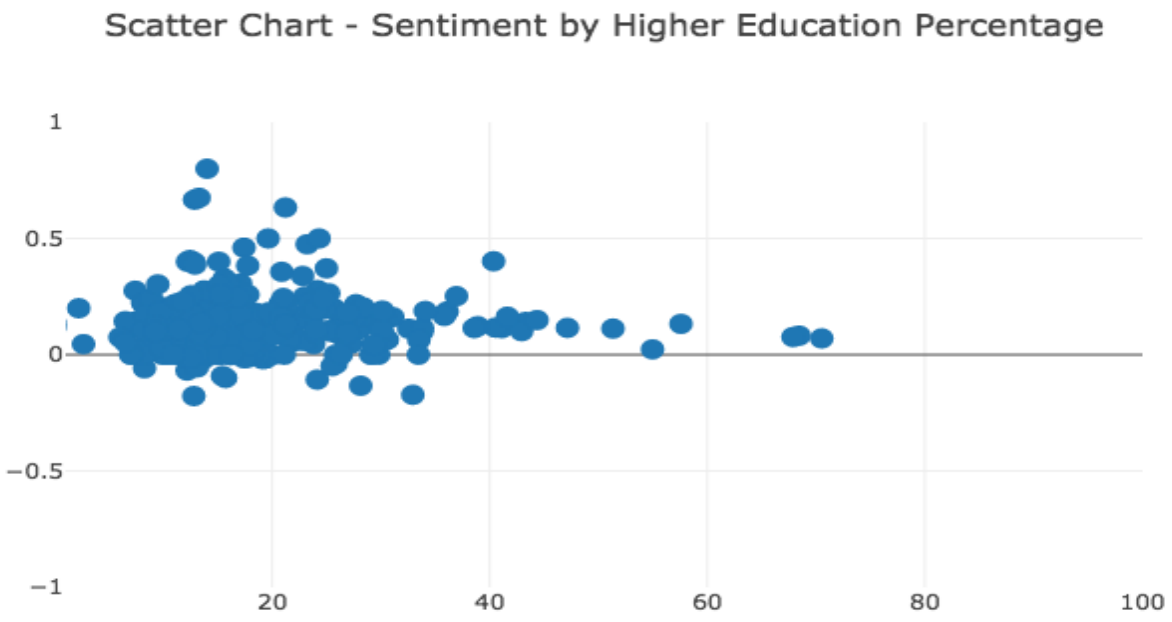


Figure 5.5-1

region	avg higher education %	average sentiment
Melbourne	0.18992724	0.122500262

Figure 5.5-2

In order to further examine the in-depth reason behind the table presented above and to better understand these Twitter users, we generated some additional data in csv/excel format. By looking into the data combining with the specific location, we interestingly found some patterns. Firstly, the users who hold a higher education degree appear more often in areas such as Burwood, Parkville, Carlton and Clayton. This is quite self-explanatory as these areas all

have university campuses or student residential buildings. People there are busy with study or other academic commitments. Therefore, their sentiment would be lower and more even. Secondly, people who expressed sparsely diverse feelings through their tweets are usually based in places like Moorabbin Airport, Box Hill North, Mill Park – South, Bundoora – East, Kensington and Narre Warren South (East). Some of these areas are comfortable residential areas with parkland, shopping centers and other amenities, which apparently explains why users there are generally happier. In contrast, the other places are outer suburban and more isolated; they are generally quieter and provide fewer chances to meet or interact with other people. It is no surprise people there would demonstrate a relatively lower sentiment.

top 10 area (rank by higher education)		
area	higher education %	sentiment
Clayton	70.52%	0.0702
Carlton	68.44%	0.082
Parkville	67.88%	0.0761
Melbourne	57.58%	0.1326
Burwood	54.96%	0.023
Kingsbury	51.33%	0.1125
North Melbourne	47.14%	0.1154
Carlton North - Princes Hill	44.34%	0.149
Brunswick	43.29%	0.1408
Southbank	42.94%	0.1012

top 10 high education areas

bottom 10 area (rank by higher education)		
area	higher education %	sentiment
Wallan	7.17%	0.1167
Hastings - Somers	7.10%	0
Point Nepean	6.99%	0
Rosebud - McCrae	6.53%	0.0524
Yarra Valley	6.50%	0.1441
Koo Wee Rup	6.01%	0.075
Melbourne Airport	2.65%	0.0455
Rockbank - Mount Cottrell	2.22%	0.2
Port Melbourne Industrial	0.00%	0.1381
Braeside	0.00%	0.1179

top 10 low education areas

Figure 5.5-3

To sum up, about 20% of the Tweeter users in Melbourne hold a higher education qualification. Their average sentiment is about 0.12. When people are in places with entertainment facilities and are away from their study or work places, they generally feel happier. When people are in location where they have less opportunities to interact with others, their mood would go further down than usual.

5.6 Volunteer

Regarding the total number of volunteers and its impact on general mood of the residents' tweets, It can be seen that there is a slight positive correlation between the two factors. However, the happiest suburbs are the ones with less than 1000 volunteers, a counterintuitive observation that might be because of the small size (and therefore less population and less number of volunteers) of that neighborhood.

The areas with more than 2000 volunteers, compared to the ones with less than this number have an obvious higher average that confirms the positive influence of volunteer work in persons' mood. Overall, it can be seen that as the number of volunteers increases in a district, the average sentiment moves from neutral towards positive and at least in Melbourne this observation defends the general belief about volunteering and happiness.

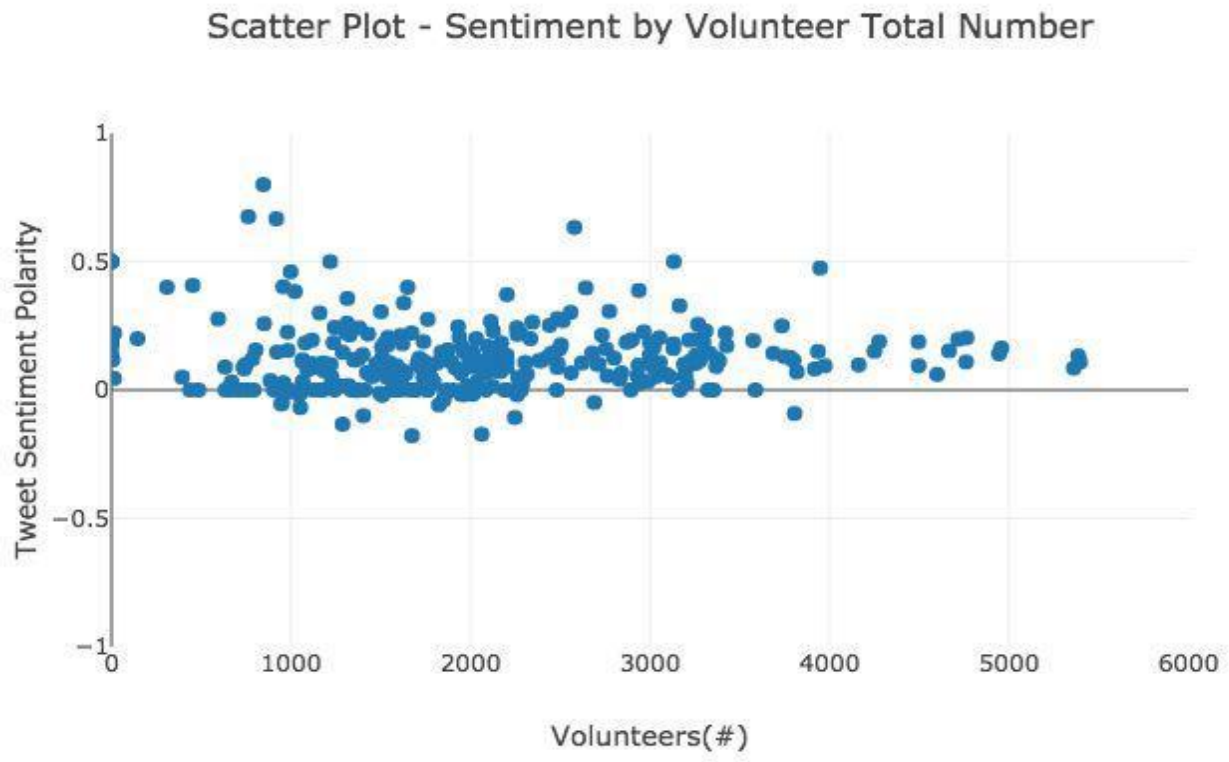


Figure 5.6-1

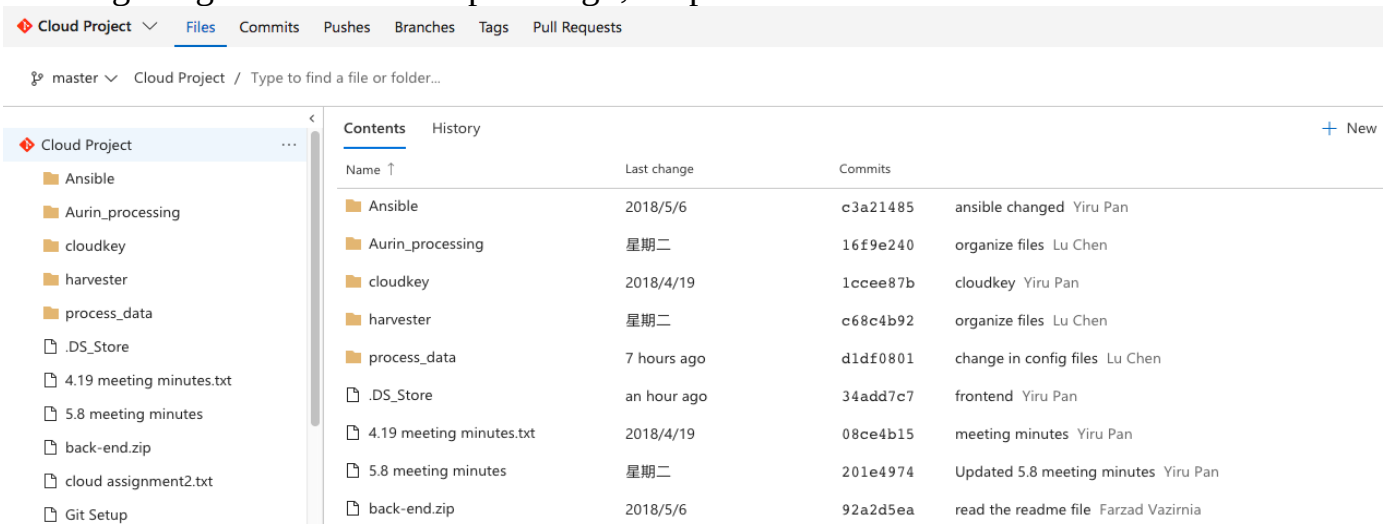
Reference

1. Twitter Analysis--data provided by Richard Sinnott
2. Sinnott, R.(2017) Cluster and Cloud Computing[Lecture Notes 5, pp.32]
3. Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

Appendix

1. Github

Our original git is visualstudio private git, snapshot is as below:



We migrate all source code to a public repo in github for review purpose.
Public Repository Link:

https://github.com/Panyiru/Cloud-Project2-Australian-Social-Media-Analysis

2. Automation Video Link:
https://vimeo.com/267390442

3. Visualization Video Link:
https://youtu.be/xMl2KJHaHys