COMP90024 Assignment 2:
City Analytics on the Cloud

Team 72

Siyuan Wu (1110062)

Dongfang Wang (906257)

Decheng Wang (812203)

Jian Liu (1010361)

Table of Contents

1. Introduction

Coronavirus disease 2019 (COVID-19) is a terrible infectious disease which has spread global and caused an ongoing pandemic. It has changed people’s normal lives and aroused the attention of people. Under the global pandemic, many people expressed their anxiety and concern about COVID-19 on Twitter. And the most important thing concerned by people is medical resources.

2. Functionalities and Scenarios

Functionalities:

This system contains 4 major functions which are Data Harvest connect with clustered CouchDB database, Data Analysis, and Web Visualization. Harvest's function is gathering tweets data from Twitter. CouchDB is used to store data. Data Analysis is used to analyse tweets and data from AURIN. Flask Web Server is used to build the webpage and create charts. Automation is used to deploy the system.

Scenarios:

In this project, we focus on exploring the relationship between the degree of people’s attention on COVID-19 and the hospital's numbers in their city. The hypothesis is that people will put more attention on it when there are less hospitals in their city, which means fewer medical resources they can reach. As the less medical resources people can reach, the more anxious they will be, and the more tweets about COVID-19 they will send since their anxiety.

We used front-end separation frame in order to implement live-updated data on web, which through the visualisation tools such as Line chart, pie charts and map to present the tendency and density of the COVID-19 relative data in different regions.

3. User Guide

3.1 System

3.1.1 Infrastructure

The infrastructure deployment is in terms of the creating key pairs, instances, attached volume on NeCTAR cloud. All the budding process is done by ansible playbooks and all the relative delivered result data are collected in json files.

3.1.2 Software

Software deployment process is shown below:

<u>Database:</u>

1. Create instances on cloud server
2. Create clustered CouchDB nodes on instances

   Open port (TCP):

   Database server: 5984

   Cluster communication: 4369, 9100-9200

<u>Web:</u>

1. Create instance on cloud server

   Open port (TCP):

   Main server: 80
2. Using “Search.py” to harvest the data from twitter and save into database
3. Filtering and analysis the data, save results in database
4. Flask server get results from database
5. Web request relative data from Flask server and done the visualisation

4.System Architecture and Design

4.1 Overall Architecture

In order to apply functionality on the Cloud Server, the overall architecture of the NeCTAR can be determined. In this project, the computational budget has 8 CPU cores, 36 GB of RAM and 250 GB usable storage. There are four instances created on the cloud server. All the instances are using CentOS 7 * 86 _64 as basic operating systems, within “uom.mse.2c9g” as flavour. Three of these instances are connected to the CouchDB database and the rest one implemented as a web server. For the instances which connect to the database have 70 GB storage capacity, since they are designed to accomplish the main computation tasks while analysing the harvest data. Hence, the web server instance only has 40 GB to run the server and achieve the data visualisation and automation tasks. In addition, the CouchDB database are clustered together which has three nodes in order to store the large amount of data, and they are full replicated in terms of backup.

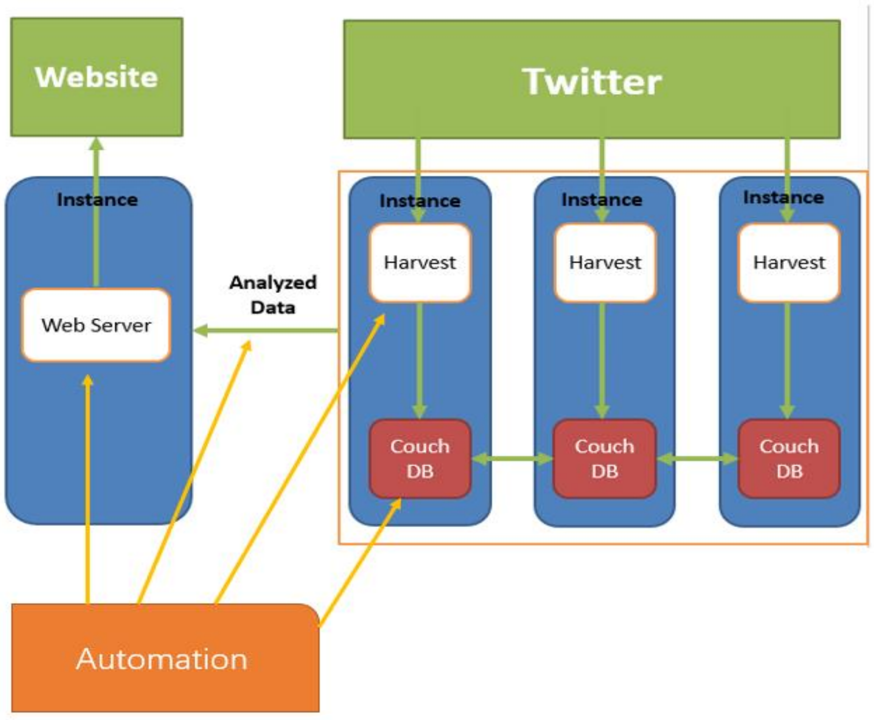


Figure 4.1.1 Over Structure

4.2 Unimelb Research Cloud

Unimelb Research cloud service is a cloud platform base on the OpenStack, it applies widely range of research tools and data for researchers in majority fields. In the project we found the following advantages and shortcoming.

Pros:

Kindly user interface: It has a kindly user interface that allows researchers to manipulate the relative features such as instances, columns, networks and so on. In addition, it is convenient to make security changes.

Cons:

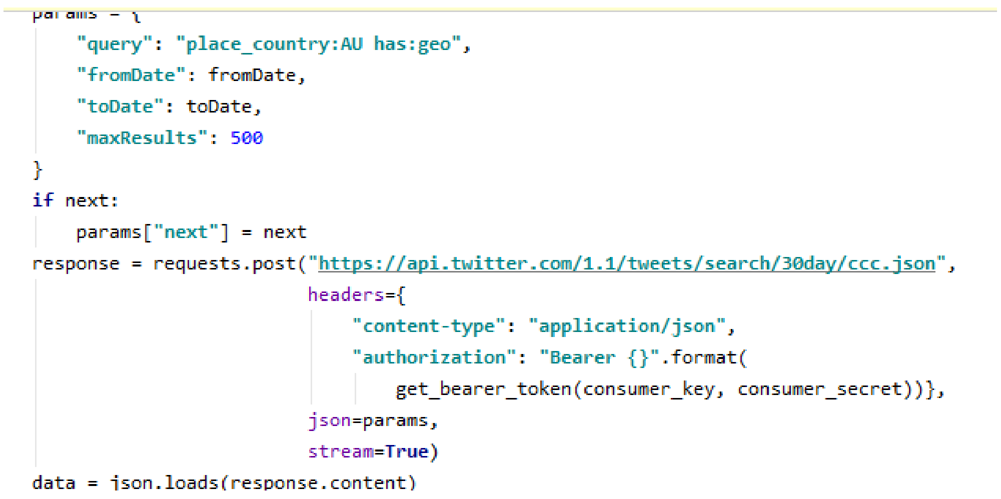
Processing speed: Due to the reason that the limit of the bandwidth, the processing speed of loading and transferring is not really fast.

4.3 Clustered CouchDB

In order to set up the CouchDB environment, the first thing is connecting the instances which are created above to the CouchDB database. After changing the setup of the address in instances, we can access the database from another machine. The next step is to cluster these three nodes, within CouchDB. different from the master-slave structure, it is more similar to a peer-to-peer structure, because they are the same level. After clustering, the data in CouchDB would change replicated to the other nodes, which make the following flask web connection more effective and convenient.

5.Harvest

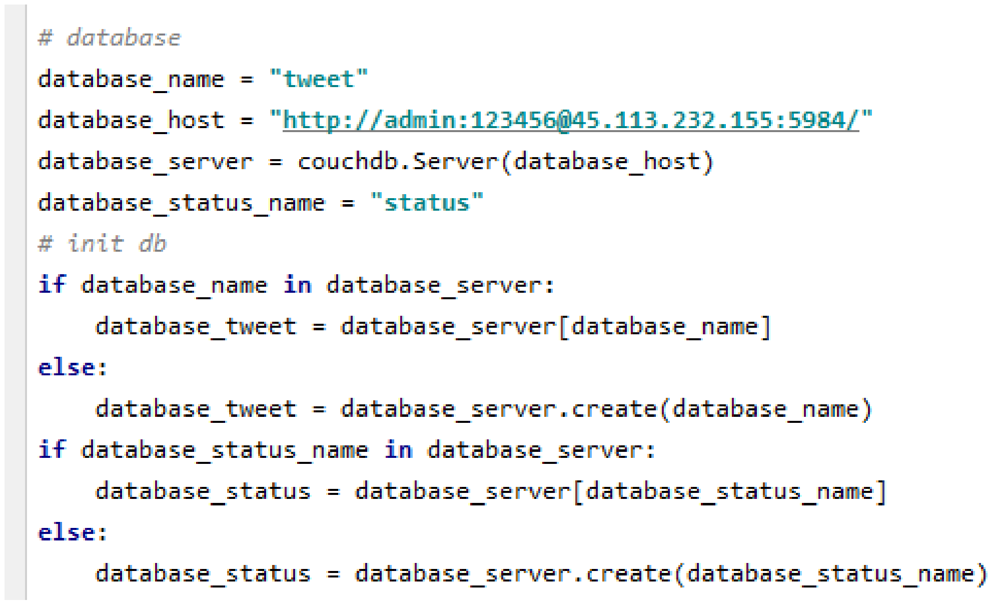
To collect data on twitter, the program uses twitter API to request the data. Since the limitation on developer accounts, we collected tweets in 30 days for analysis, this case also limited the amount of our tweet data. As twitter API can be used to control the range of data collection, there is no need to use extra methods to achieve this goal.



```
params = {
    "query": "place_country:AU has:geo",
    "fromDate": fromDate,
    "toDate": toDate,
    "maxResults": 500
}
if next:
    params["next"] = next
response = requests.post("https://api.twitter.com/1.1/tweets/search/30day/ccc.json",
                         headers={
                             "content-type": "application/json",
                             "authorization": "Bearer {}".format(
                                 get_bearer_token(consumer_key, consumer_secret))},
                         json=params,
                         stream=True)
data = json.loads(response.content)
```

Figure 5.1 Tweets Request

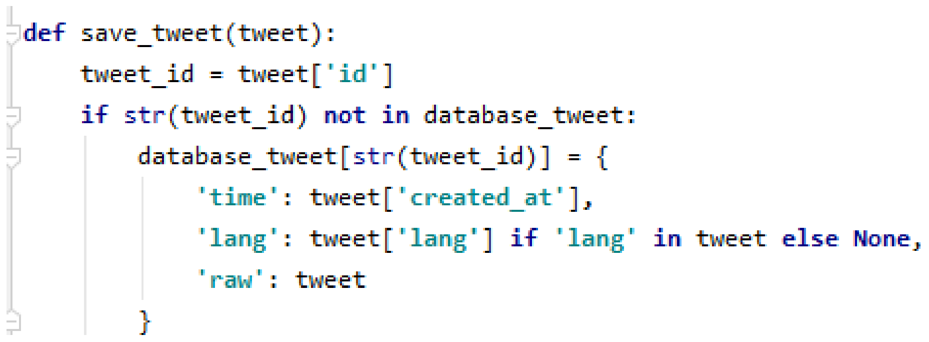
Before reading data, we should create a database in CouchDB for saving data and status.



```
# database
database_name = "tweet"
database_host = "http://admin:123456@45.113.232.155:5984/"
database_server = couchdb.Server(database_host)
database_status_name = "status"
# init db
if database_name in database_server:
    database_tweet = database_server[database_name]
else:
    database_tweet = database_server.create(database_name)
if database_status_name in database_server:
    database_status = database_server[database_status_name]
else:
    database_status = database_server.create(database_status_name)
```

Figure 5.2 Create Database

After receiving the tweets data, we save the data in the database ‘tweet’. In this process, id of tweet is used to check if the program gets duplicate data. Because of the replication function in CouchDB, the harvest on three instances can get the same result when checking the list of tweets in the database. This can prevent the program from saving duplicate data.



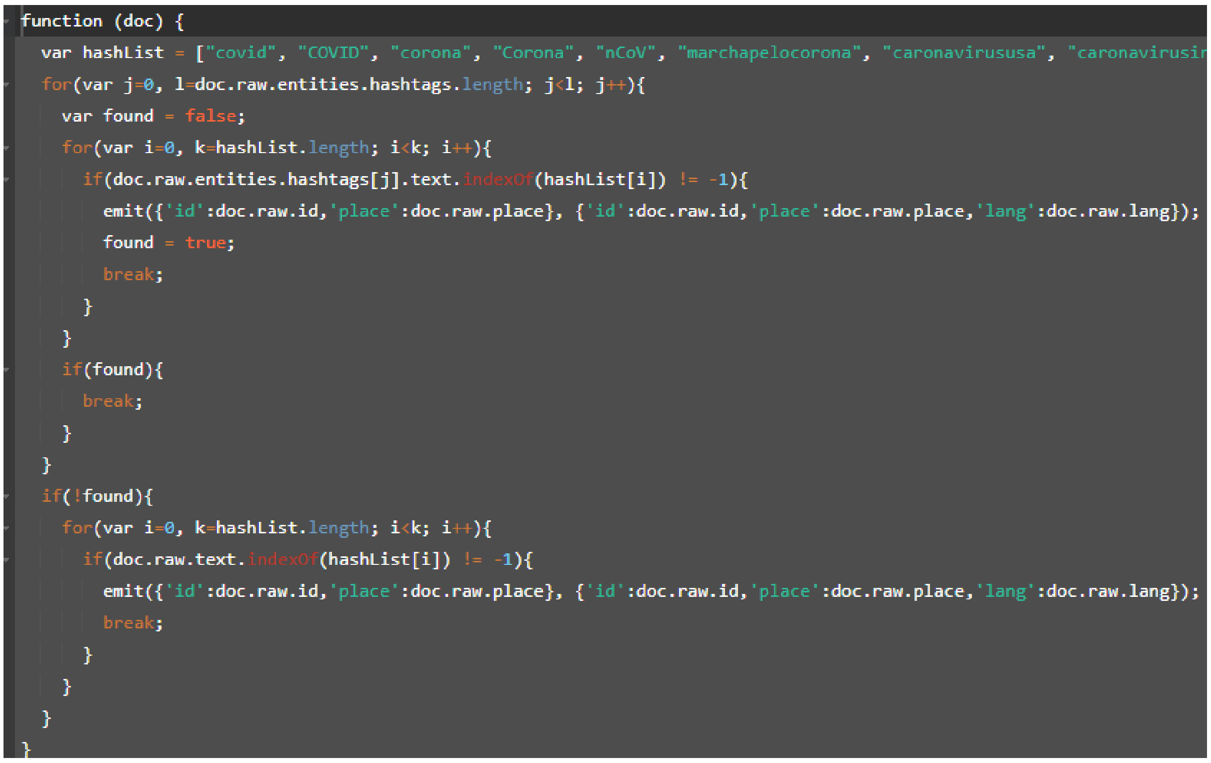
```
def save_tweet(tweet):
    tweet_id = tweet['id']
    if str(tweet_id) not in database_tweet:
        database_tweet[str(tweet_id)] = {
            'time': tweet['created_at'],
            'lang': tweet['lang'] if 'lang' in tweet else None,
            'raw': tweet
        }
```

Figure 5.3 Save Data

6. Data Analysis

6.1 CouchDB MapReduce

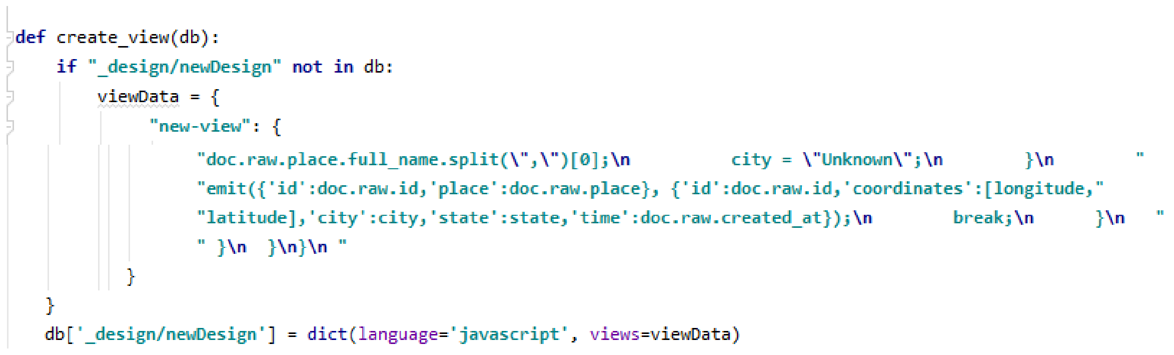
Before starting the data analysis, we need to select the tweets mentioned COVID-19. To achieve the goal, a view with map function is added to CouchDB. The map function will check whether the tweet has hashtags about COVID-19 and will also check if the content of tweet mentions COVID-19 without any hashtags.
After filtering out the tweets, the function will clean the data, which means it only returns critical necessary data for analysis, such as the ‘id’ and ‘place’ attributions in tweets. As the MapReduce function can build a filtered index in a short time, compared with cleaning data after reading them into the program, using MapReduce function will save much time and memory storage in this step.
This function is implemented in JavaScript since the default language for MapReduce function in CouchDB is it.



```
function (doc) {
  var hashList = ["covid", "COVID", "corona", "Corona", "nCoV", "marchapelocorona", "caronavirususa", "caronavirusi
  for(var j=0, l=doc.raw.entities.hashtags.length; j<l; j++){
    var found = false;
    for(var i=0, k=hashList.length; i<k; i++){
      if(doc.raw.entities.hashtags[j].text.indexOf(hashList[i]) != -1){
        emit({'id':doc.raw.id,'place':doc.raw.place}, {'id':doc.raw.id,'place':doc.raw.place,'lang':doc.raw.lang});
        found = true;
        break;
      }
    }
    if(found){
      break;
    }
  }
  if(!found){
    for(var i=0, k=hashList.length; i<k; i++){
      if(doc.raw.text.indexOf(hashList[i]) != -1){
        emit({'id':doc.raw.id,'place':doc.raw.place}, {'id':doc.raw.id,'place':doc.raw.place,'lang':doc.raw.lang});
        break;
      }
    }
  }
}
```

Figure 6.1.1 Map Function in Tweets database

And to make sure the view works normally in this process, we add the view into the 'tweets' database for data cleaning before starting analysis.

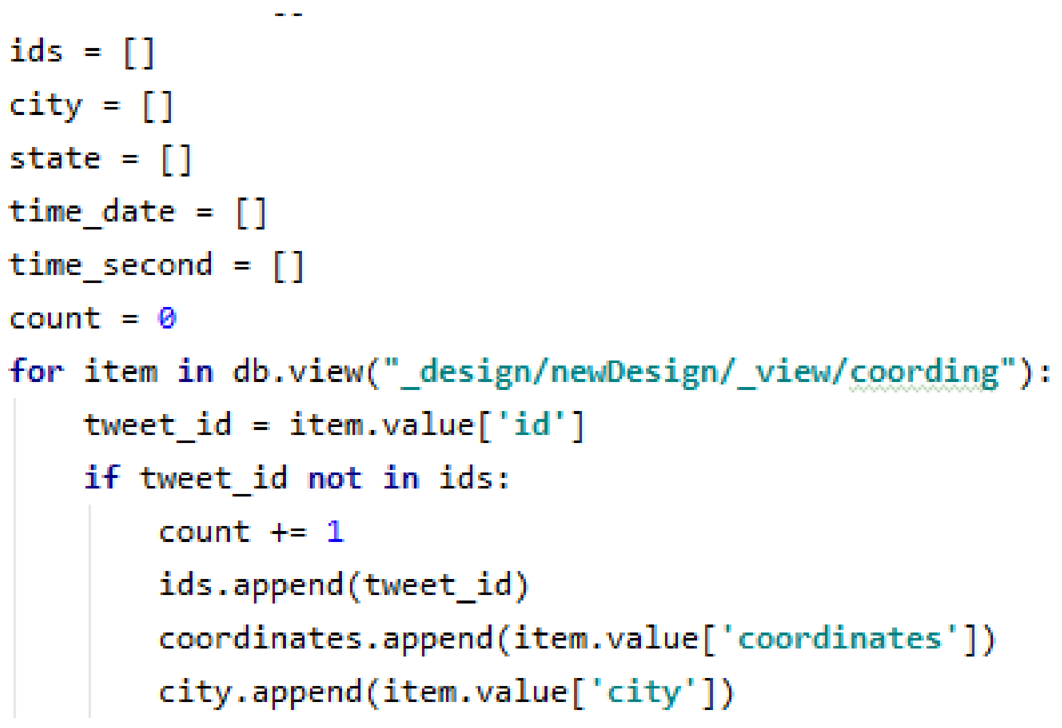


```
def create_view(db):
    if "_design/newDesign" not in db:
        viewData = {
            "new-view": {
                "doc.raw.place.full_name.split(\",\")[0];\n          city = \"Unknown\";\n        }\n        "
                "emit({'id':doc.raw.id,'place':doc.raw.place}, {'id':doc.raw.id,'coordinates':[longitude,"
                "latitude],'city':city,'state':state,'time':doc.raw.created_at});\n        break;\n      }\n    "
                " }\n  }\n}\n "
            }
    }
    db['_design/newDesign'] = dict(language='javascript', views=viewData)
```

Figure 6.1.2 Create View (Part of Code)

6.2 Data Analysis

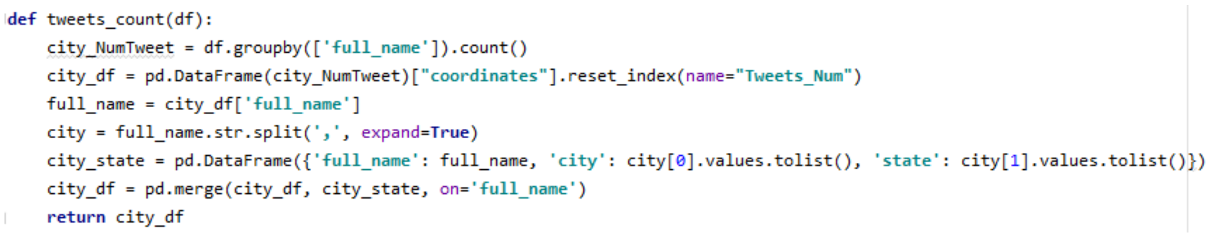
Since there may be some duplicate tweets read from the database, the program needs to check if a tweet is read in the process. So, we decide to save tweets' id in a list and check if tweets' ids are in the list before using the data in analysis.



```
ids = []
city = []
state = []
time_date = []
time_second = []
count = 0
for item in db.view("_design/newDesign/_view/coording"):
    tweet_id = item.value['id']
    if tweet_id not in ids:
        count += 1
        ids.append(tweet_id)
        coordinates.append(item.value['coordinates'])
        city.append(item.value['city'])
```

6.2.1 Read Tweets

The twitter user’s location is important data for analysis, however, not all users put their location online. Some of the tweets do not have an accurate city location, and it will cost unacceptable time if we try to use their coordinates to find which city they stay in. Though the ‘geopy’ package can find an address using the coordinates, the huge number of tweets and query time decide that it does not suit this case. Thus, we decide to use the ‘full_name’ in tweet data and regard those who do not have accurate city location in an ‘Unknown’ city.
And the data analysis process uses the 'groupby()’ method to count how many tweets in a city, then saves this data and tweets’ coordinates, cities, states in a data frame for the next step.

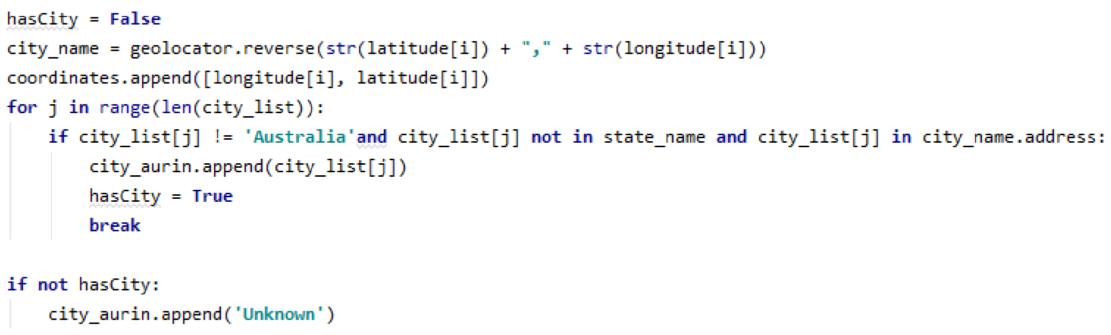


```
def tweets_count(df):
    city_NumTweet = df.groupby(['full_name']).count()
    city_df = pd.DataFrame(city_NumTweet)["coordinates"].reset_index(name="Tweets_Num")
    full_name = city_df['full_name']
    city = full_name.str.split(',', expand=True)
    city_state = pd.DataFrame({'full_name': full_name, 'city': city[0].values.tolist(), 'state': city[1].values.tolist()})
    city_df = pd.merge(city_df, city_state, on='full_name')
    return city_df
```

6.2.2 Count Tweets in One City

6.3 AURIN Data

For further analysis, AURIN data about hospitals in Australia is needed. We use the AIHW-National Hospital Statistics from AURIN as the data for hospitals because it is the most complete dataset we can find.
However, there are only abbreviations of states, longitude and latitude to show the hospitals’ location. As the data will be grouped by cities, further processing is necessary for this process. Since the number of hospitals in the dataset is limited, we decided to use the ‘geopy’ package to find the address of the hospital.



```
hasCity = False
city_name = geolocator.reverse(str(latitude[i]) + "," + str(longitude[i]))
coordinates.append([longitude[i], latitude[i]])
for j in range(len(city_list)):
    if city_list[j] != 'Australia'and city_list[j] not in state_name and city_list[j] in city_name.address:
        city_aurin.append(city_list[j])
        hasCity = True
        break

if not hasCity:
    city_aurin.append('Unknown')
```

6.3.1 Find City Name

There are two problems in this process. First, City names from ‘geopy’ are in different styles between tweet and geopy. For example, Melbourne will be ‘The Melbourne City’ in ‘geopy’, but it will be ‘Melbourne’ in tweets.
This case makes the analysis process more difficult. To deal with this case, the method tries to find if the city name from tweets can be found in results from geopy. If there is no corresponding string, the method will mark the city name as ‘Unknown’. Second, the city name in results from geopy does not have a certain index. We have to deal with it as a string. This makes the final result not accurate and many hospitals will have no city name. Thus, we need record hospitals’ coordinates for further analysis on map.

6.4 Generate Json File

After counting the hospitals in cities, the data should be combined and saved for further using. So, at the end of this process, we combine these data by the ‘full_name’, which means the city and its state, and save them in CouchDB as a json file.

6.5 Annalise Result

According to the result, we found that twitter users in states with more hospitals pretend to tweet about COVID-19. Though there are only some of the hospitals that have a certain city name since the limitation of geopy, we found the tweet number and hospital number are in positive correlation.

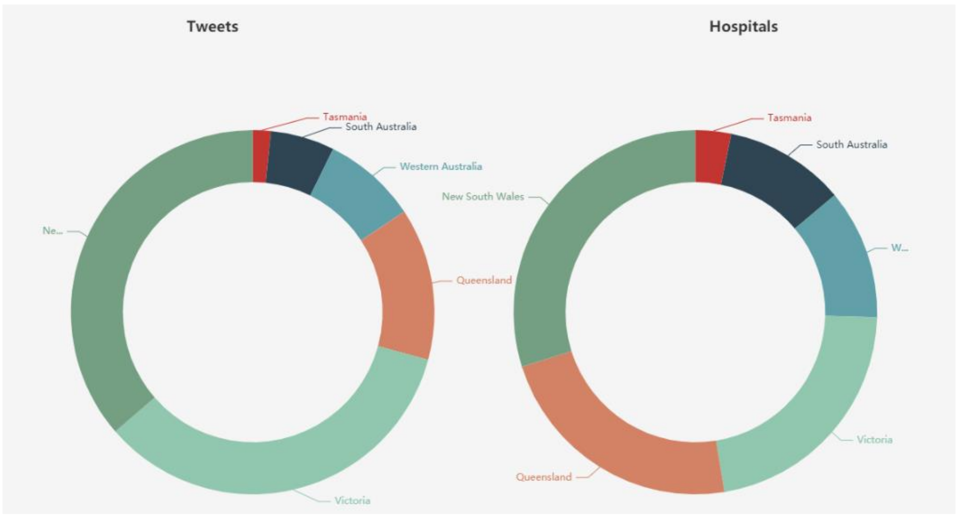


Figure 6.5.1 Tweets and Hospitals in States

Compared with the result on the map, it’s clearer that those areas with more hospitals will have more tweets about COVID-19. It may be caused by that people are more likely to meet news, even cases, about COVID-19 in areas where have more medical resources, and they will put more attention on it.

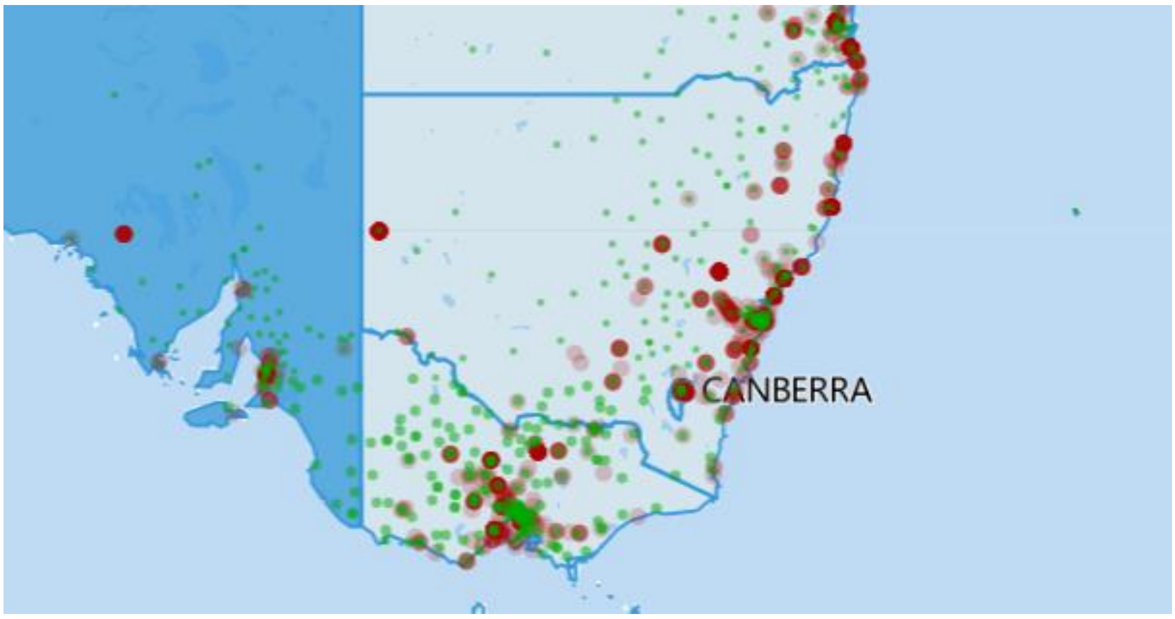


Figure 6.5.2 Tweets and hospitals on Map

7. Web Server

7.1 Restful API with Flask

In order to synchronise the analysed data to the web and help the web to present the visualising result.
This API is designed with these rules, it should have a separation between the client and server, and communication between server and client would be standardised in the way which allows the negotiator to respond to the requests. In addition, while the client communicates to the server, the way of communication should be uniform.
After setting up the flask environment, process of building web service with flask is simple, in this project, the server connects to the database host "http://admin:pwd@45.113.235.44:5984/" GET data from database and return the corresponding json file to the path.

7.2 Server

Different from the classic front-end frame, we used a front-end separation frame. The classic front-end frame is that after rendering the file, it will return a html file which is used for the front web, and in our project, the end server is applying the API data, and the front web will accomplish the visualisation tasks, they work separately.
After connecting to the database, we can design the server at the next step. The first thing to design a web server is to set up the root URL which is "http://localhost/", where the local host will be changed to "0.0.0.0." for accessing by different machines. Then as described as above, in order to select resources which will be exposed by the resource, the HTTP method 'GET' will be used in all the applications later. As for the applications, the details have been listed as table below.

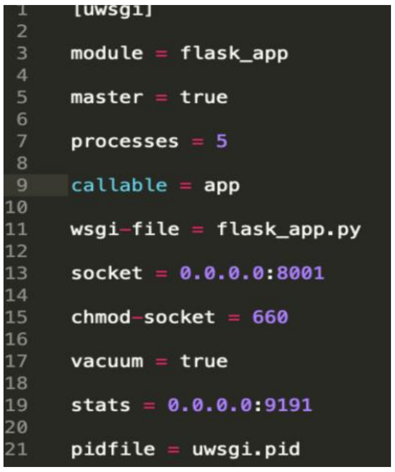
'

API path	Relative database data	Main Output
/timeline_data	day_state_tweets	“Date” :[D1 ,D2, D3 …] “Data”: [{“name”: “STATE_1”: “data” : [NUM1, NUM2,NUM3...]} [{“name”: “STATE_2”: “data” : [NUM1, NUM2,NUM3...]}....]
/city_data	results	“Data”: [{ “name”: “STATE_1”, “tweets” : {CITY_1, NUM1}, {CITY_2, NUM2}...}... “hospitals”:{CITY1,NUM3}, {CITY_2, NUM4}...}...]
/lang_data	city_lang_results	“Data”: [{“name”: Language 1, “value”: NUM1}, {“name”: Language 2, “value”: NUM2}...]
/dot_data	tweet_coord	“tweets”: [{“coordinate”: COORD1}, {“coordinate”: COORD2},...] “hospitals”:[{“coordinate”: COORD3}, {“coordinate”: COORD4},...]

Table 7.2.1 Server Input and Output

7.3 uWSGI Deploy

In this project, we use uWSGI to deploy the flask application, which makes multi requests that can be processed at the same time. The configuration is shown below.



```
[uwsgi]

module = flask_app

master = true

processes = 5

callable = app

wsgi-file = flask_app.py

socket = 0.0.0.0:8001

chmod-socket = 660

vacuum = true

stats = 0.0.0.0:9191

pidfile = uwsgi.pid
```

Figure 7.3.1 uWSGI Deploy

7.4 Web Design

7.4.1 Timeline (Line Chart)

At this part the top button can choose to present the data of each state or in total, X axis is the time shaft in May, the Y axis is the number of tweets that are relative to COVID-19. This chat can show that the attention tendency of people changes over the time. In addition, it can present the attention tendency at different states, which depends on the population density or the situation that is affected by COVID-19, or the decree that is published by the state government and so on.

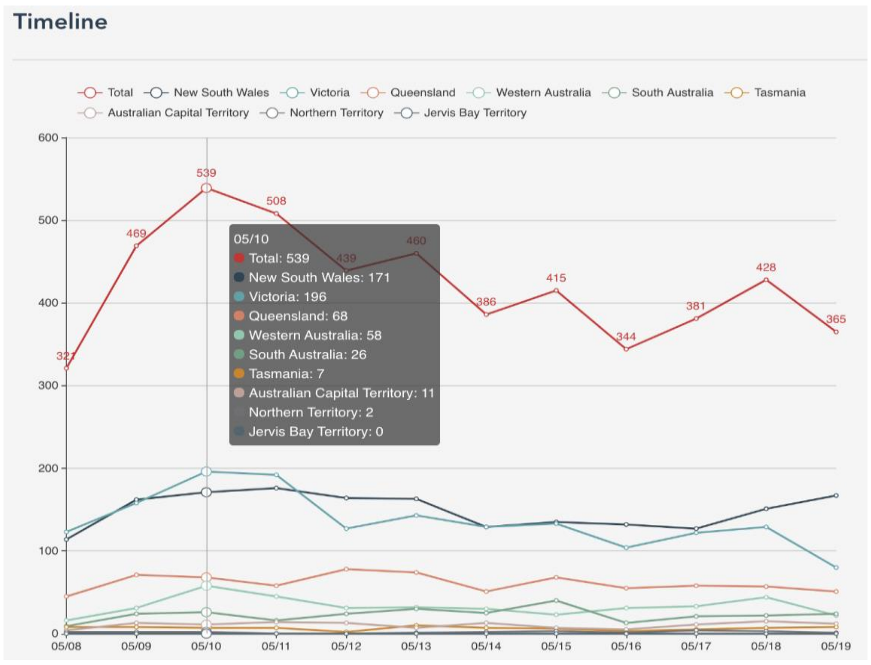


Figure 7.4.1.1 Tweets Timeline

7.4.2 States & Cities (Pie Chart)

We presented the tendency above, as for states and cities, we focus more about comparing the number of relative COVID-19 tweets between states and cities in states. In other words, we only care about the attention from people in such states or cities. In addition, we also present the amount of hospitals in the cities and states, which aims to analyse if the number of the hospitals affect the attention. The top bar can select the state to see the details of the city, which is shown below.

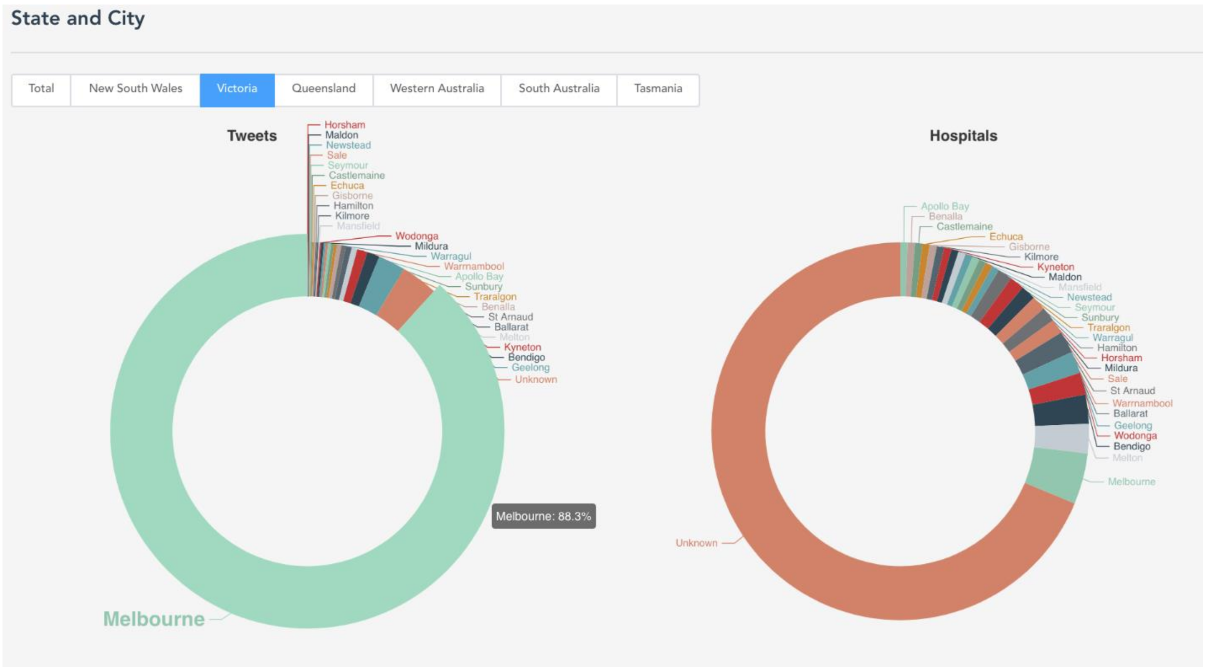


Figure 7.4.2.1 States & Cities

7.4.3 Language (Pie Charts)

Depending on the different communities and cultures background, we are also interested in if it will affect the attention for the COVID-19. Due to the English tweets as the main language, we also add another pie chart to compare the rest other languages, the charts are shown below.

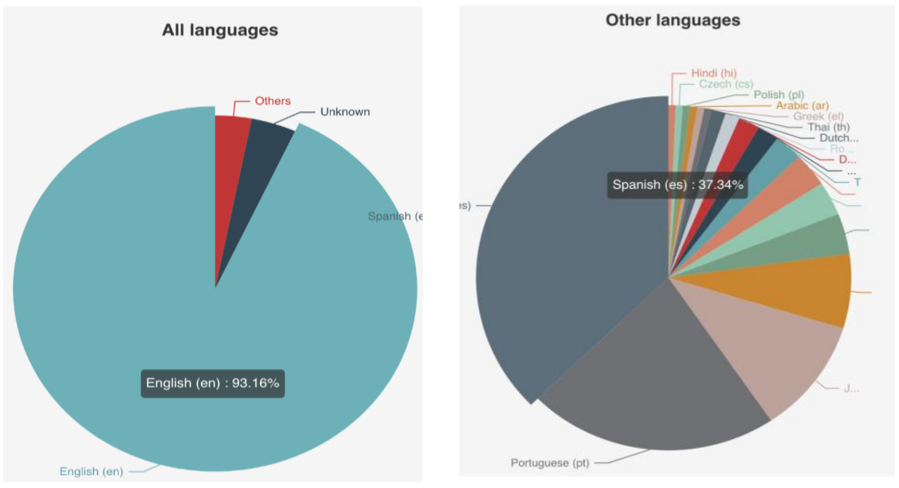


Figure 7.4.3.1 Languages

7.4.4 Map

The map presents the statistic number of tweets data for each state, which is the other way to show where the tweets are from and to see the distribution during the time period.

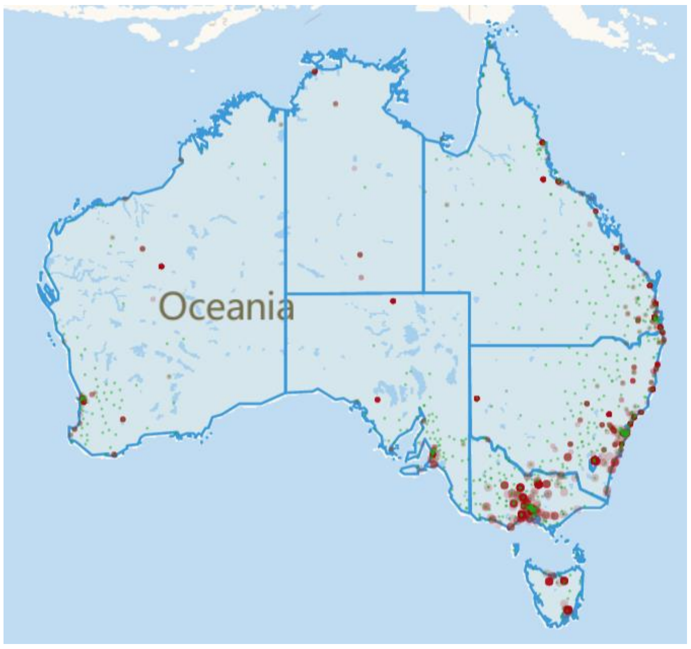


Figure 7.4.4.1 Map

8. Error Handling

8.1 Twitter API

While we search the twitter API, since the network fluctuation problems, the request might fail which will lead to the data loss. In order to solve this problem, we will use the “NEXT” parameter which from the result of the request,when we send the request, it will respond with 500 requests and the “NEXT” parameter, which indicate the range of these 500 requests. If we send another request, it will start from the end of the data which the “NEXT” flag indicates. Hence, we add the “NEXT” parameter into the search algorithm, if the request fails, the parameter can help us avoid losing data and get replicate data.

8.2 Front-End Request

When the web requests data from the flask server, it may fail, and the web will crash. For solving this, we added the resend button while unsuccess to get data from the server.

9. Conclusion

According to the result from the project, We found that among the people who care about COVID-19, apart from the number of people who speak English, a considerable number of people who use other languages (such as Spanish, Portuguese, etc.) are affected by COVID-19, which shows that COVID-19 has an impact on the lives of people with different cultural backgrounds. The epidemic is a problem we face together.

The number of sending tweets about COVID-19 is related to many factors, the local population, and the severity of the epidemic and so on. All in all, these areas need more medical resources allocation, we want to know whether these medical resources are met. Comparing the tweets distribution with the distribution data of hospitals from AURIN, we know that regions with higher Twitter traffic have higher density of hospital distribution, which shows that the government's planning for hospitals is reasonable.

The good news is that from the trend of related tweets, the trend of people sending related tweets is gradually decreasing, which shows that people are gradually adapting to the government's related anti-epidemic measures and the inconvenience of life caused by the COVID-19, which also shows that the COVID-19 is being effectively controlled.

10. Future Improvement

In this project, the project can be improved in three points.

First, the data size is limited since the limitation of account. There are around 250000 tweets collected in the project. If we have more data, the chart on the web page will perform better and the relationship between the number of tweets and hospitals will be clearer.

Second, we can use machine learning in the project to analyse twitter users' emotions when they mention COVID-19. It will show how the amount of hospitals influence their attitude to COVID-19, and how people's attitudes change as time goes by.

Third, we can try to use more AURIN data and GeoJson to make the map on the web page clearer and more accurate.

11. Reference List

[1] Geopy Contribution (2018) Geopy Documentation. Retrieved from: https://geopy.readthedocs.io/en/stable/#

[2] Twitter, Inc (2020) API reference index. Retrieved from: https://developer.twitter.com/en/docs/api-reference-index

[3] Apache Software Foundation (2020) Apache CouchDB® 3.1.0 Documentation. Retrieved from: https://docs.couchdb.org/en/stable/

[4] Red Hat, Inc (2019) Ansible Documentation. Retrieved from: https://docs.ansible.com/ansible/latest/index.html

12. Appendix Link

Server: http://45.113.233.244/

Github: https://github.com/liujl3/CCCTeam72

System Demo Video:
https://www.youtube.com/watch?v=8MdzP73Syk0