City Analytics on the Cloud

THE UNIVERSITY OF MELBOURNE

Group 31

Bilei Zhu, 1185748

Tianyi Mo, 875556

Yan Xu, 777481

Yitong Cheng, 1195544

Yusheng Huang, 1257211

Table of Contents

1. Introduction/Abstract

In the context of the information age, social media has become very popular in people's daily lives, such as Twitter and Instagram. The longer people use them, the higher the status of social media in daily life.

This Assignment builds and deploys applications on a cloud platform and runs them with the help of the Unimelb Research Cloud. The app collects public tweets from people in all capital cities in Australia through the Twitter API. They were then screened and analyzed. At the same time, we also combined the data of the Australian Urban Research Infrastructure Network(AURIN) and analyzed the results of the data in CouchDB.For our data analysis, we focused on the happiness index of several major Australian cities. The Scenario is mainly considered:

a. Whether people's happiness is positively correlated with the size of their city and its GDP?
b. Does the number of tweets containing COVID-19 sent by people in each city correlate with the number of outbreaks in that city in the past year? How did they feel about COVID-19?
c. If people send their tweets with a location, does that mean they're happier (assuming they're willing to share their location publicly for travel and other reasons)?
d. If the sentiment score of tweets on topic 'income' is related to the local salary level?
e. If the number of tweets under 'beer' or 'junk food' has correlation with the local obesity rate？

Finally, we show the results of data analysis in the form of a map and other charts.

2. Melbourne Research Cloud(MRC)

2.1 Pros of using MRC

The nature of cloud computing is distributed computing, also known as network computing. It can achieve a very powerful network service, in a very short time to complete the processing of tens of thousands of data. Cloud computing divides the processing program of data computation into many small programs, and processes these small programs in parallel through multiple servers and then outputs the results.

2.1.1 Scalability

The Melbourne Research Cloud (MRC) is an on-demand computing resource provided by the University of Melbourne that functions similar to the commercial cloud. Scalability is one of the most important advantages of using the MRC in this task. The couchDB database, web server can easily scale up and down to meet the demand. For example, if we need to store and process more data, we can request for allocating more resources and increase the capacity of the system.

2.1.2 Rapid Deployment

Another advantage of MRC is rapid deployment of the system. With the help of ansible and other automation tools, the entire system including frontend, backend, data harvester and database can be fully functional in several minutes including. The users of MRC do not need to maintain the infrastructures or visit the data center to deploy the system. Thus the developer can focus on the core component of the system rather than buying and maintaining the IT infrastructures.

2.1.3 Performance

Melbourne Cloud Resource is great support for Twitter's data processing. Efficient communication between Instances makes data transfer smoother. The smooth interaction between the frontend and backend means that data can be transmitted to the frontend and displayed in real-time after the backend has processed it.

2.2 Cons of Using MRC

2.2.1 Access Speed

Unfortunately, when we used the Melbourne Resource Cloud, we found that it had a number of limitations. First, we need to connect to the Virtual Private Network(VPN) to access the MRC resources. Accessing unimelb's VPN is very slow in China, which increases the latency of transmitting data between instance and our computers. This makes the terminal less responsive while connecting the MRC virtual machines using ssh and reduces the efficiency of deployment. The same thing happened while visiting the web server of the deployed system from using the VPN, it takes about 5 seconds to load the webpage because of the slow network speed.

2.2.2 Limited control and flexibility

While it is convenient to not maintain all IT infrastructure, it also means that the infrastructure is completely owned and maintained by the service provider which is the University of Melbourne. There are few options on the type of the instances and operating system provided. For example, we only have two choice of instance types which are "1core4g" and "2core9core", we cannot have virtual machines that have more virtual CPUs and stronger performance. In addition, the service provider has some management policies that might impose limits on what customers can do with their deployments.

3. System Design and Architecture

3.1 System Architecture

The system under current configuration consists of 3 instances. The data harvester and couchDB are deployed on the first and second instance. The data analyzer and web server are deployed on the third instance. The rest instance is reserved for testing purposes.

The data harvesters continuously receive tweets through the twitter api. After preprocessing and keeping the necessary fields, the tweets are stored in the CouchDB database. As shown in Figure 1, we have three data harvesters “Search Data Harvester”, “User Timeline Data Harvester”. However, since we only have two twitter developer accounts, only two data harvesters can run simultaneously. The details for our data harvester will be discussed in section 4.2. The details for CouchDB will be discussed in section 4.3.

The data analyzer is designed for pulling tweets stored in several databases in CouchDB, performing several analyses every 60 minutes and storing the results in json formats . This is because the analysis is computationally expensive, it needs to run for a few minutes to generate the result. The data analyzer will be discussed in section 5.

The web server then reads the result generated by the data analyzer, these two processes share the results by sharing the volume between their containers. While the web server receives the HTTP request, it will send the response message with the corresponding html body. The web page designed will be shown at section 6.

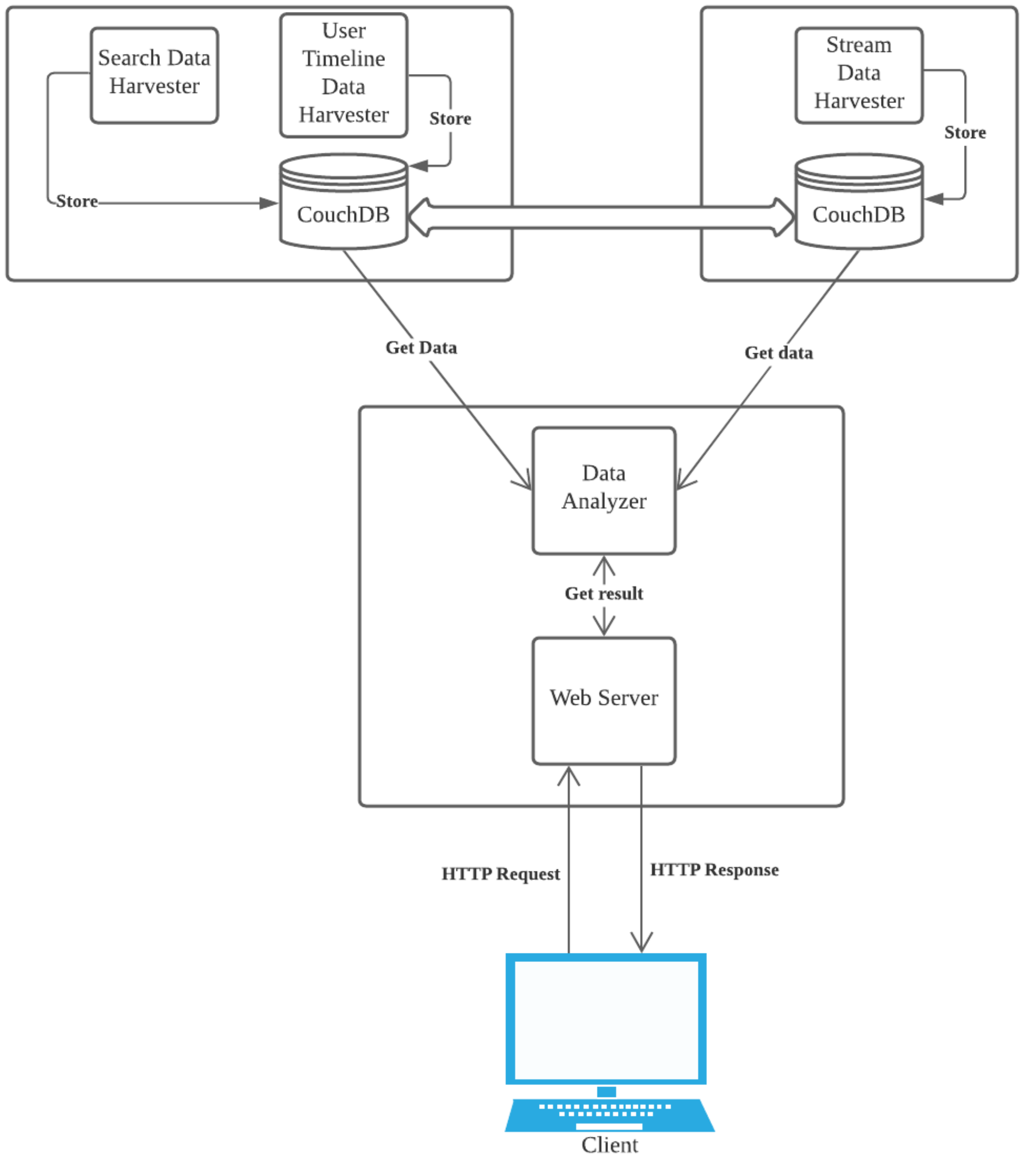

Search Data Harvester
User Timeline Data Harvester
Store
Store
CouchDB
Stream Data Harvester
Store
CouchDB
Get Data
Get data
Data Analyzer
Get result
Web Server
HTTP Request
HTTP Response
Client

Figure. 1

3.2 Instances Allocation

Instance Name	Usage	Availability Zone	Image name	IP Address	Storage
instance1	CouchDB, Data Harvester	melbourne-qh2-uom	NeCTAR Ubuntu 20.04 LTS (Focal) amd64	172.26.133.138	uom.mse.1c4g
instance2	CouchDB, Data Harvester	melbourne-qh2-uom	NeCTAR Ubuntu 20.04 LTS (Focal) amd64	172.26.128.223	uom.mse.1c4g
instance3-webserver	Web Server, Data Analyzer	melbourne-qh2-uom	NeCTAR Ubuntu 20.04 LTS (Focal) amd64	172.26.131.90	uom.mse.2c9g

Table. 1

4. Development

The following figure shows the interaction of project resources between the local and cloud infrastructure. The ansible management node uses ssh to communicate with other nodes. The images are built and pushed to the docker hub with tags and they can be pulled on the virtual machine on the Melbourne Research Cloud and run as containers. All components in the system are deployed using docker containers to ensure each process can have consistent and isolated environments. This can simplify the deployment process on heterogeneous operating systems as each image can manage its dependencies.

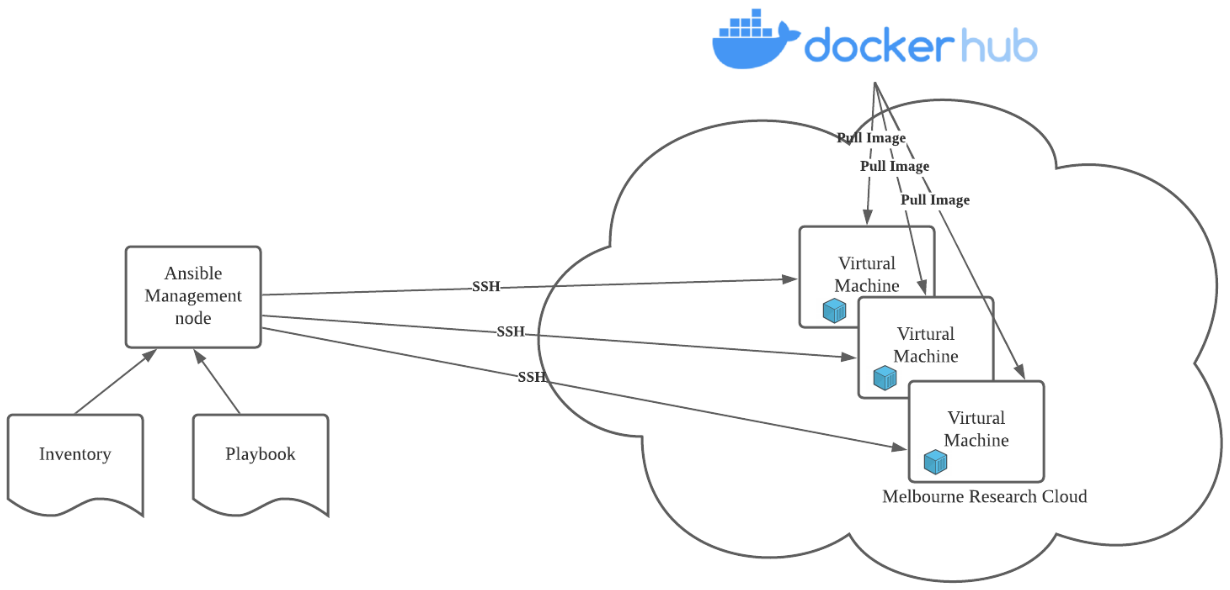


Figure. 2

4.1 Ansible

In this project, we use Ansible as the deployment tool. Ansible is a Python-based automated operation and maintenance tool that can implement system configuration, program deployment and running commands in batches (Mohaan & Raithatha, 2014). Ansible allows us to deploy complex cloud systems with only few commands, all the configurations and steps are stored in yaml playbooks and make the deployment more reproducible and less error-prone. With the help of ansible, if we want to change the configuration of the system, we can simply update the configuration file and execute the playbooks again.

We used ansible to automatically do the following tasks:

a. Setting up security group rules in the project
b. Managing the volumes and instances creation on the Melbourne Research Clouds
c. Installing dependencies and tools on the instances
d. Setting up and launching the CouchDB cluster.
e. Launching the Data Harvester.
f. Launching the Data Analyzer.
g. Launching the Web Server.

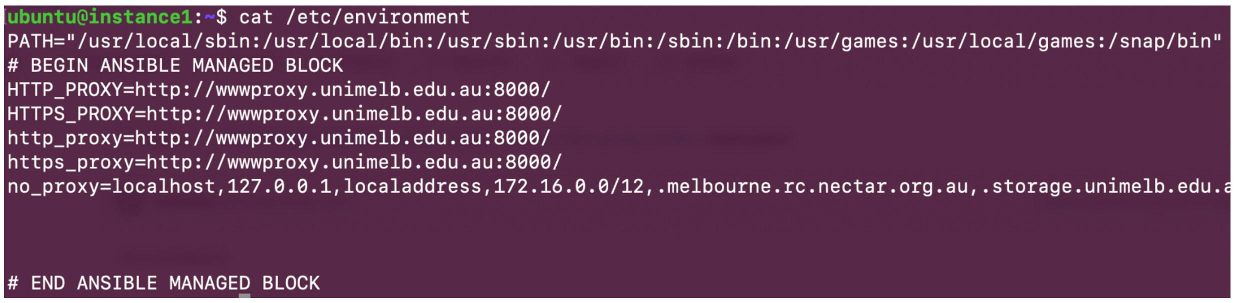
4.1.1 Ansible Roles

4.1.1.1 Automatically saving the IP address

The IP address for the created instances are stored so that that can be used in subsequent tasks without manually specifying the ip address. This is extremely important if we need to manage a large number of instances and the saved file is used as the inventory file when executing other ansible playbooks.

4.1.1.2 Modifying Proxy on the instances

Ansible is also used to modify the http_proxy, https_proxy and no_proxy environment variables on the instances so that the virtual machine can have access to the Internet to update software or pull docker images. Since the virtual machines are behind the unimelb's HTTP or HTTPS proxy server, it is essential to add the configuration in the Docker systemd service file using Ansible. Figure. 3 shows that the proxy variables are in the ANSIBLE MANAGEMENT BLOCK.



```
ubuntu@instance1:~$ cat /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
# BEGIN ANSIBLE MANAGED BLOCK
HTTP_PROXY=http://wwwproxy.unimelb.edu.au:8000/
HTTPS_PROXY=http://wwwproxy.unimelb.edu.au:8000/
http_proxy=http://wwwproxy.unimelb.edu.au:8000/
https_proxy=http://wwwproxy.unimelb.edu.au:8000/
no_proxy=localhost,127.0.0.1,localaddress,172.16.0.0/12,.melbourne.rc.nectar.org.au,.storage.unimelb.edu.a


# END ANSIBLE MANAGED BLOCK
```

Figure. 3

4.1.1.3 Configuration CouchDB cluster

The complex setting up process which consists of logging on each couchDB node and sending some HTTP requests is replaced with 3 ansible roles which are "deploy-couchDB", "deploy-couchDB-cluster" and "deploy-couchDB-grunt". By default, it will make the first instance in the database node instance list as the master node. The "deploy-couchDB" role runs a docker container on each instance, "deploy-couchDB-cluster" role sets up the cluster and then verifies the setup is successful and "deploy-couchDB-grunt" role sets up all views in the couchDB database.

4.1.1.4 Managing Docker Container

In our project, all processes are run in the Docker containers. We also use ansible to pull images, manage the creation of containers. If the container with given name is running, it will first stop current and start the new container

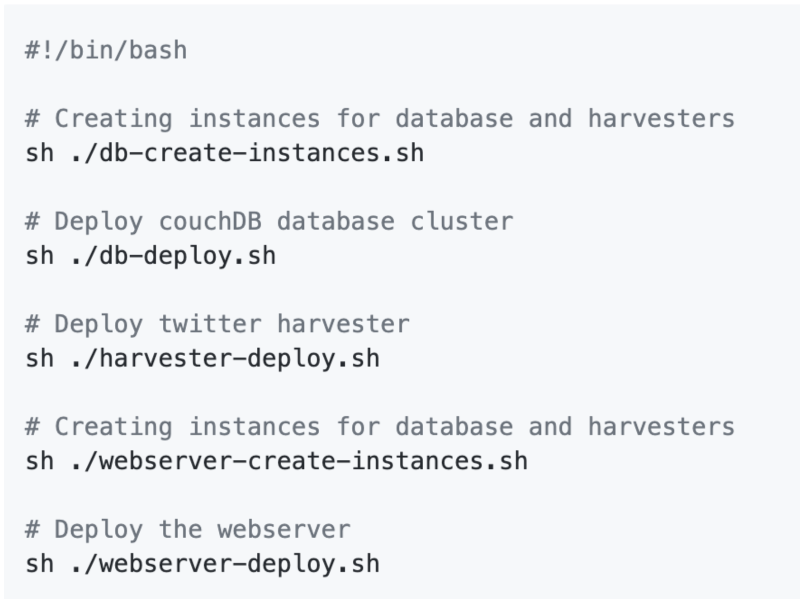
4.1.2 Deployment using Ansible

Before deploying the system, you need to download the OpenStack RC File from the MRC Dashboard and get your user password. This file contains the Environment variables that are necessary to run OpenStack command-line clients.

To start the system, you can execute the following command from the “ansible” directory. This will create and set up all instances and set all components including the webserver, couchdb database, data harvester and data analyzer.

```
sh ./deployall.sh
```

The figure 4 shows the command will run in “deployall.sh”. On some occasions, for example if the instance is already created, you can run commands to do specific tasks. Each script will execute the playbook of the same with the corresponding inventory file. The playbooks consist of several reusable roles defining the operation will be done or bash command will be executed . The detailed documentation of all playbooks can be found on our github repository.



```
#!/bin/bash

# Creating instances for database and harvesters
sh ./db-create-instances.sh

# Deploy couchDB database cluster
sh ./db-deploy.sh

# Deploy twitter harvester
sh ./harvester-deploy.sh

# Creating instances for database and harvesters
sh ./webserver-create-instances.sh

# Deploy the webserver
sh ./webserver-deploy.sh
```

Figure. 4

4.2 Twitter Harvesting

Twitter provides several APIs that can be used to harvest tweet objects based on different strategies and criteria set by developers. For the purpose of harvesting a great amount of tweets which are useful for the analytics scenarios we choose in a short time, we utilize three different Twitter APIs, which are "stream", "search" and "user_timeline". As each of them has its own advantages and limitations in practice (Table 2):

	Advantages	Limitations And disadvantages
Search API	1) Fast. Can return a big amount of tweets in a short time. 2) Can set criteria such as keywords and locations for searching.	1) May return lots of duplicates if search frequently 2) Reach rate limit after harvesting 2500 tweet objects in practice. It will recover in 10 minutes.

Stream API	1) All tweets streamed are real-time. 2) Hardly return duplicated tweets. 3) Can set criteria such as keywords and locations for streaming.	1) Slower than the other two APIs. Can only return about 12000 tweets per day if we only stream tweets in Australia in practice. The speed will be even much slower if more criteria is set.
User_timeline API	1) Fast. Able to return a big amount of tweets in a short time. 2) Can find years ago tweets theoretically.	1) Need user ids for searching tweets. 2) 900 requests per 15 mins as maximum 3) Not able to set keyword for searching

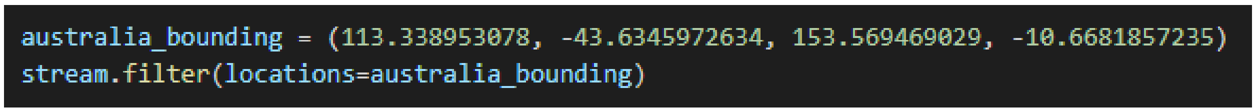
Table. 2

4.2.1 Twitter APIs selected and the reasons

we apply each of them for carrying out just the appropriate jobs it is good at for maximizing the revenue.

Stream API:

Stream API is used for streaming real-time public tweets. It provides a function named "filter" for helping us only harvest the tweets we need under a set of criteria, including keywords, locations, and languages. In practice, if we set too strict conditions for the filter, the streaming flow will be very low, as there might be few tweets on a certain topic posted from a certain location. Therefore, we apply it to monitor and stream tweets posting across Australia under no other condition (Figure 5).



```
australia_bounding = (113.338953078, -43.6345972634, 153.569469029, -10.6681857235)
stream.filter(locations=australia_bounding)
```

Figure. 5

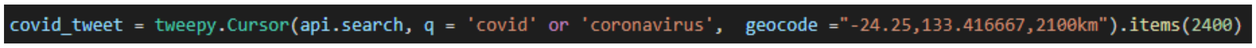
The tweets harvested by streaming are used for doing the general tweets information analysis, including sentiment comparison between Australia's big cities, and the percentages of tweets talking about the topics we are interested in out of all tweets.

User_timeline API:

As stream API can only provide us with around 12000 tweets posting across Australia per day, we also take advantage of user_timeline API, for harvesting a bigger amount of tweets within a short period. Although there is a rate limit set for user_timeline API (900 requests per 15 mins), it can help us harvest around 200000 tweets per day in practice, which is much larger than the amount of tweets that stream API can harvest per day. It is noteworthy that user_timeline needs user ids for harvesting, it can obtain 20 tweets per user as maximum. Therefore, stream API needs to be used for obtaining a big amount of user ids before applying user_timeline. The user ids used for user_timeline search should also be chosen cautiously for different analytics scenarios, as biased lists of users may result in tweets harvested biased, which may not be a good sample for reflecting the population.

Search API:

Although the combination of stream and user_timeline API can already allow us to crawl a large number of tweets, the tweets on the topics we are interested in is still not enough. Therefore, search API is used in our twitter harvesting process as well, in order to crawl enough tweets on certain topics (Figure 6).



```
covid_tweet = tweepy.Cursor(api.search, q = 'covid' or 'coronavirus',  geocode ="-24.25,133.416667,2100km").items(2400)
```

Figure. 6: search tweets on “covid” topic in Australia

4.2.2 Architecture of the Twitter Harvester

The harvester is composed of 3 individual harvesters, which are search harvester, stream harvester and user_timeline harvester. Each of them is deployed in a different docker and can crawl tweet objects, extract the attributes of tweet objects we need and write the extracted tweet objects into CouchDB independently. The design is fault tolerant, because even if one harvester shuts down, others are still able to do the harvesting job. And the design can also allow us to decide which harvesters to turn on and which harvesters to turn down without limitations.

The flowchart of our Twitter harvester is demonstrated in Figure 7:

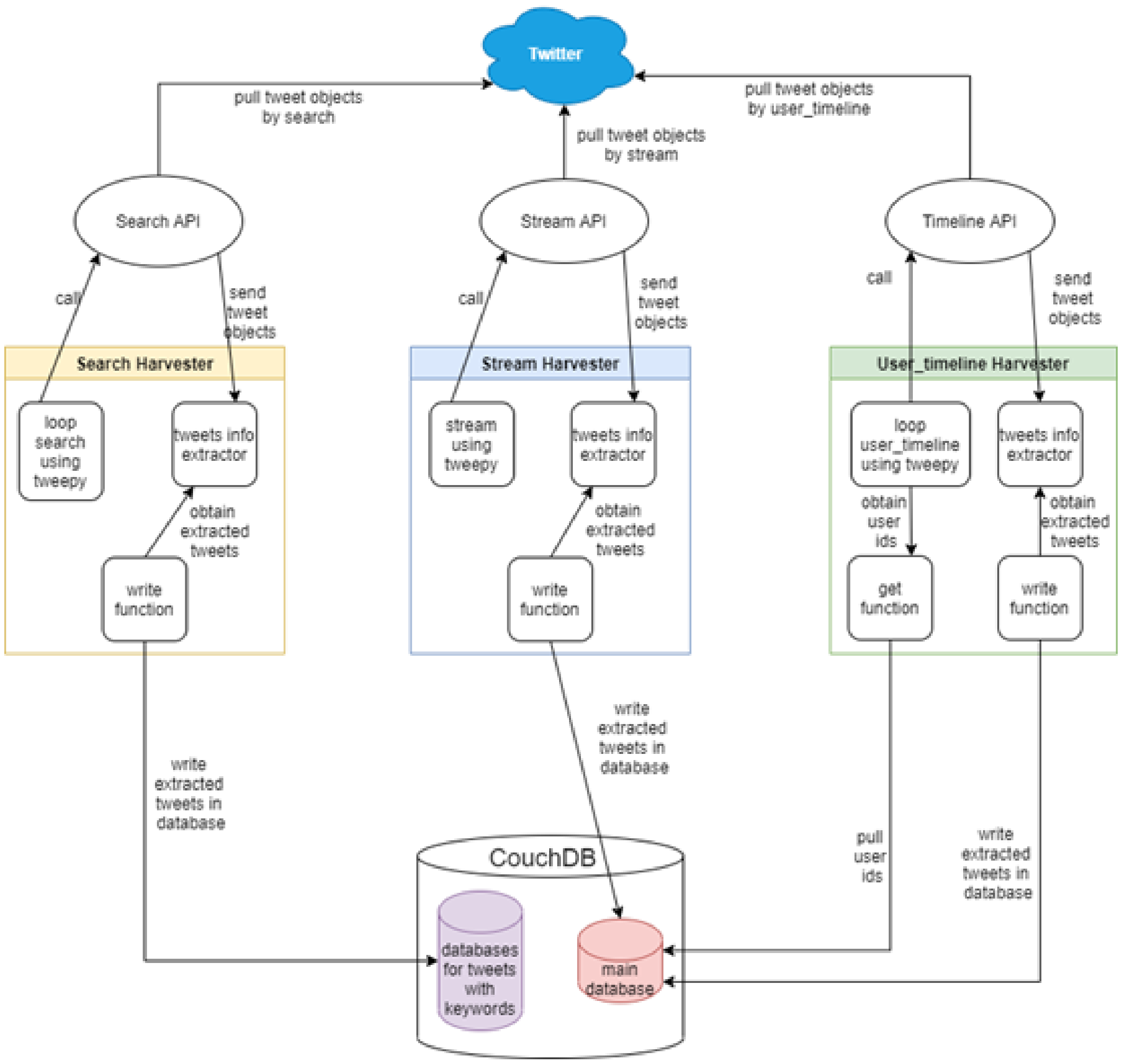


Figure. 7

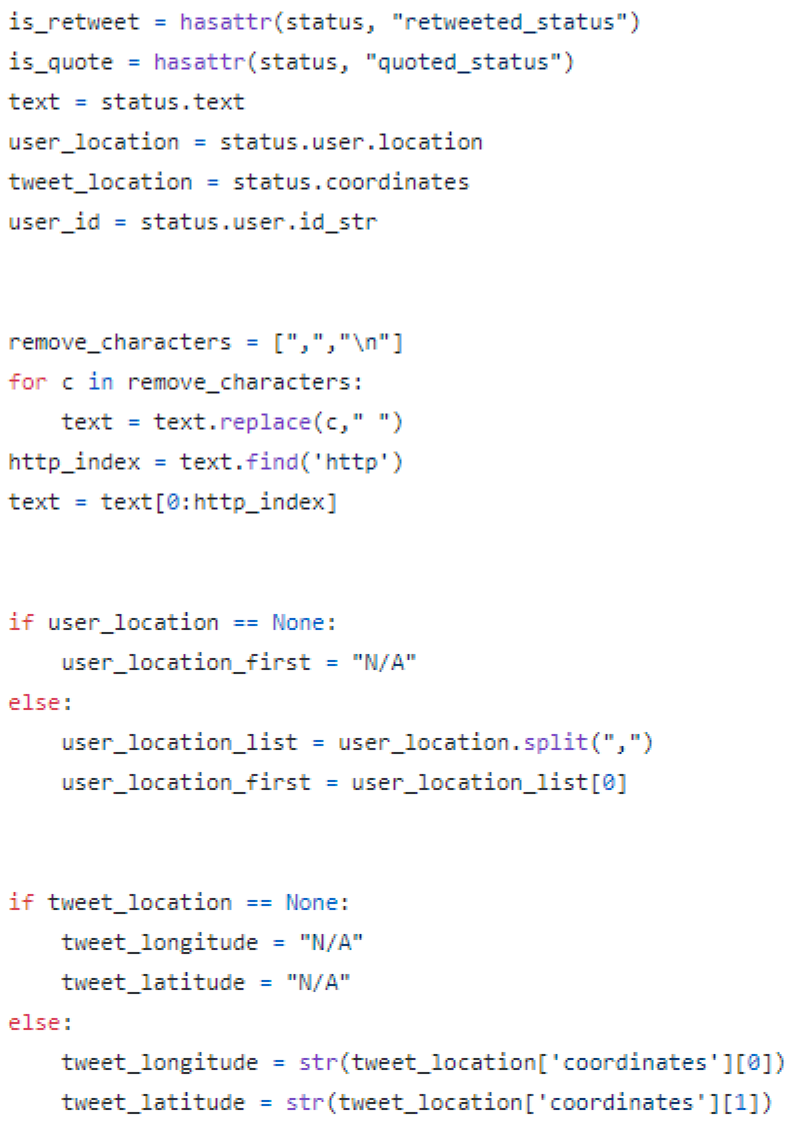
4.2.2.1 Tweet Objects Extraction and Processing

The tweet objects are large data dictionaries with lots of attributes. Most of the attributes are not useful for the analytics scenarios, therefore, we only keep 12 attributes of each tweet object for 3 reasons:

1) Extracted tweet objects take up less storage space of CouchDB.
2) The write in speed and pull speed of extracted tweet objects are both faster.
3) Easier for the Data Analyzer module to deal with.

The 12 attributes we keep are: tweet.text, tweet.coordinates, user.location, user.id_str, tweet.retweet_count, tweet.favorite_count, user.followers_count, user.friends_count, retweeted_status, quoted_status, tweet.created_at.

After extracting the attributes we need, The 3 harvesters will also process some of the attributes following the same steps before injecting them into CouchDB, in order to make the data suitable for storing in json format and easier to operate after pulling by data analyzer module (Figure 8):



```python
is_retweet = hasattr(status, "retweeted_status")
is_quote = hasattr(status, "quoted_status")
text = status.text
user_location = status.user.location
tweet_location = status.coordinates
user_id = status.user.id_str


remove_characters = [",","\n"]
for c in remove_characters:
    text = text.replace(c," ")
http_index = text.find('http')
text = text[0:http_index]


if user_location == None:
    user_location_first = "N/A"
else:
    user_location_list = user_location.split(",")
    user_location_first = user_location_list[0]


if tweet_location == None:
    tweet_longitude = "N/A"
    tweet_latitude = "N/A"
else:
    tweet_longitude = str(tweet_location['coordinates'][0])
    tweet_latitude = str(tweet_location['coordinates'][1])
```

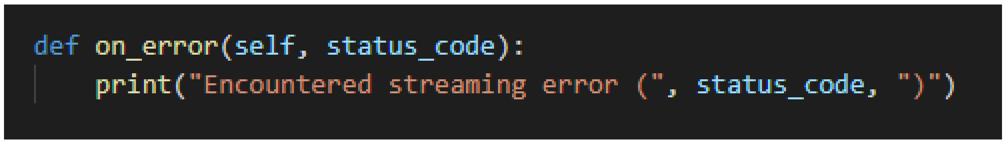
Figure. 8

4.2.2.2 Write Extracted Tweets to CouchDB

The 3 harvesters all have independent writers, which make use of the python HTTP library called "Requests" . Providing the ip address of the CouchDB, user name, password, name of the database, "Requests" can help us inject tweets in certain databases of CouchDB.

4.2.2.3 Error handling for the stream harvester

Although the stream harvester can continue running on MRC naturally, there are still a lot of errors which can cause the stream harvester to shut down. The on_error function demonstrated in Figure 9 can help us target the error occurred so that it will be easier for us to fix it:



```python
def on_error(self, status_code):
    print("Encountered streaming error (", status_code, ")")
```

Figure. 9

4.2.2.4 Loop and error handling for search and user_timeline harvesters

Unlike stream API, search API and user_timeline API will stop when finish the task assigned to it or reach the rate limit. Therefore, if we want to continue harvesting tweets using search API and user_timeline API, we will need to loop sending requests and handle the error that occurs when reaching the rate limit.

For the search harvester, we loop the search task assigned to it to allow it continuously running on MRC. After each time the task is completed, the harvester will sleep for 5 mins to avoid reaching the rate limit when doing the next round of searching. However, if somehow the harvester still reaches the time limit, our design can help catch the error and let the harvester sleep for 1 min waiting for the rate limit recovery (Figure 10).

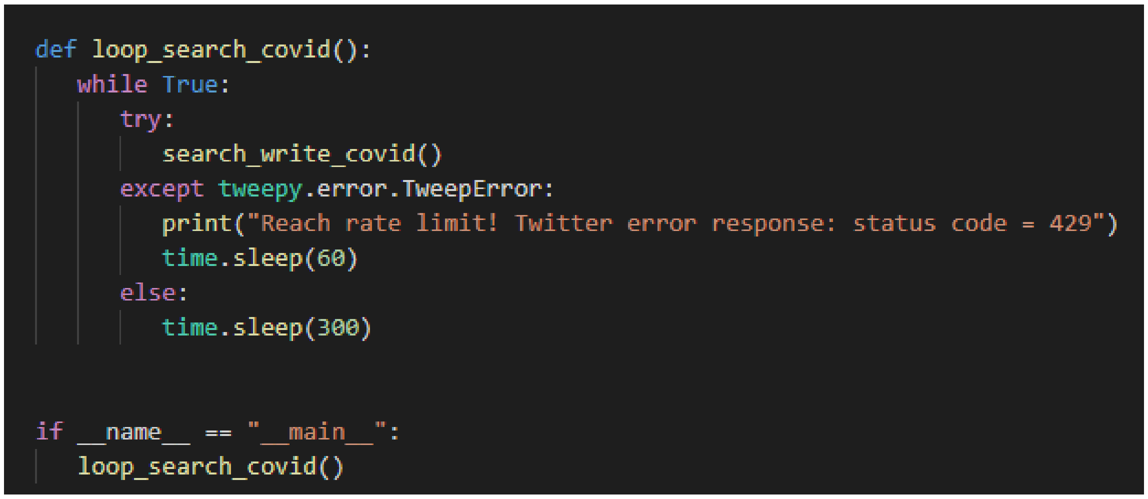

```python
def loop_search_covid():
    while True:
        try:
            search_write_covid()
        except tweepy.error.TweepError:
            print("Reach rate limit! Twitter error response: status code = 429")
            time.sleep(60)
        else:
            time.sleep(300)


if __name__ == "__main__":
    loop_search_covid()
```

Figure. 10

For the user_timeline harvester, we also loop sending requests using the user ids list pulled from CouchDB and let the harvester sleep for 15 mins when reaching the time limit. The design can also handle the error when the given user id is not found or the user id is not an integer (Figure 11).

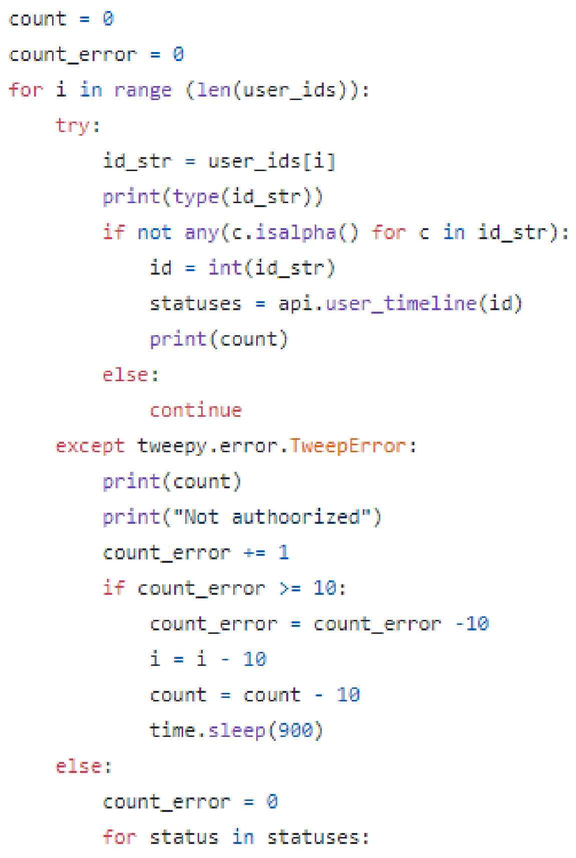
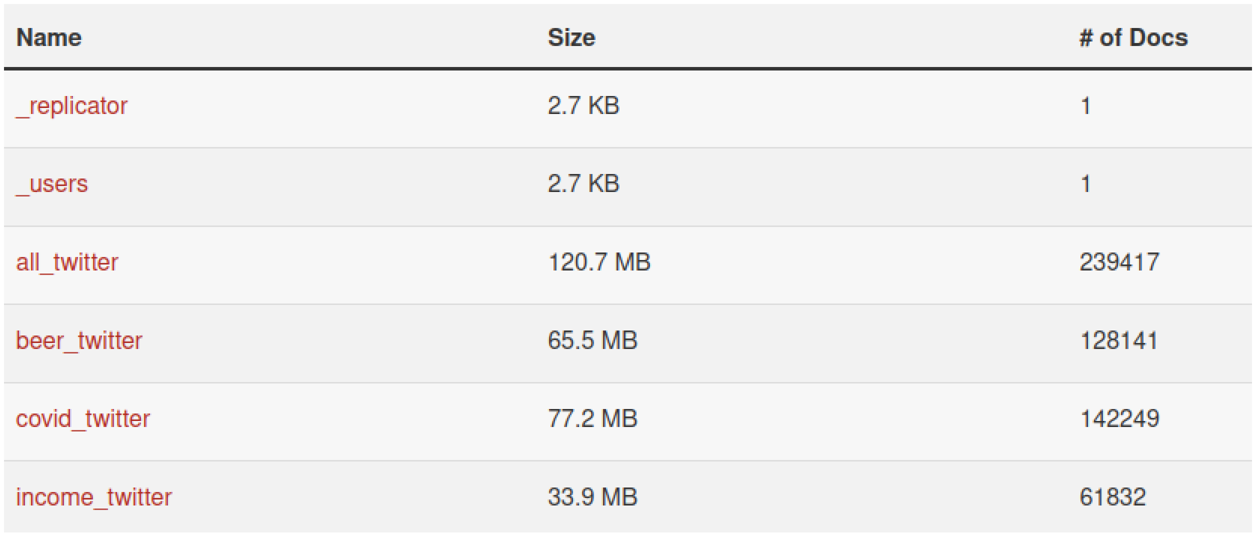

```python
count = 0
count_error = 0
for i in range (len(user_ids)):
    try:
        id_str = user_ids[i]
        print(type(id_str))
        if not any(c.isalpha() for c in id_str):
            id = int(id_str)
            statuses = api.user_timeline(id)
            print(count)
        else:
            continue
    except tweepy.error.TweepError:
        print(count)
        print("Not authoorized")
        count_error += 1
        if count_error >= 10:
            count_error = count_error -10
            i = i - 10
            count = count - 10
            time.sleep(900)
    else:
        count_error = 0
        for status in statuses:
```

Figure. 11

4.3 CouchDB

CouchDB is an open-source document-oriented database management system that can be accessed through the RESTful JavaScript Object Notation (JSON) API. The term "Couch" is the acronym Of "Cluster Of Unreliable Commodity Hardware". It reflects that the goal Of CouchDB is highly scalable and provides high availability and reliability. This is true even when running on failure-prone hardware. In this assignment, we chose CouchDB as the data storage and management platform due to its focus on data redundancy. It can avoid possible information loss.

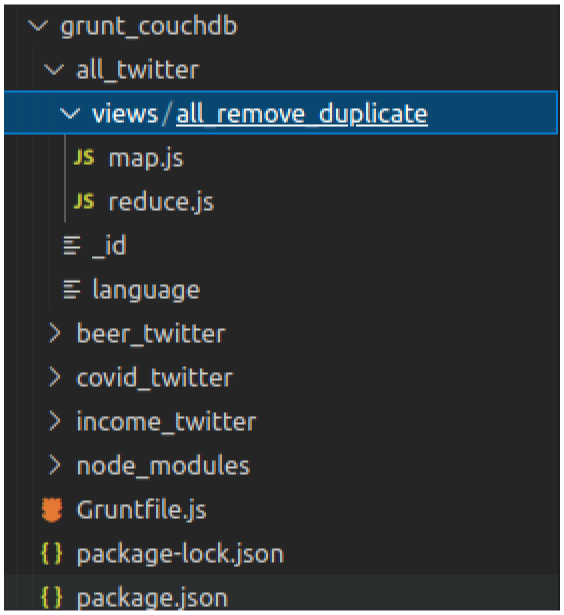
There are 571,639 tweets in total in couchDB up to now; 239,417 raw tweets, 128,141 beer related tweets, 142,249 covid related tweets and 61832 income related tweets. (Figure. 12)



Name	Size	# of Docs
_replicator	2.7 KB	1
_users	2.7 KB	1
all_twitter	120.7 MB	239417
beer_twitter	65.5 MB	128141
covid_twitter	77.2 MB	142249
income_twitter	33.9 MB	61832

Figure. 12

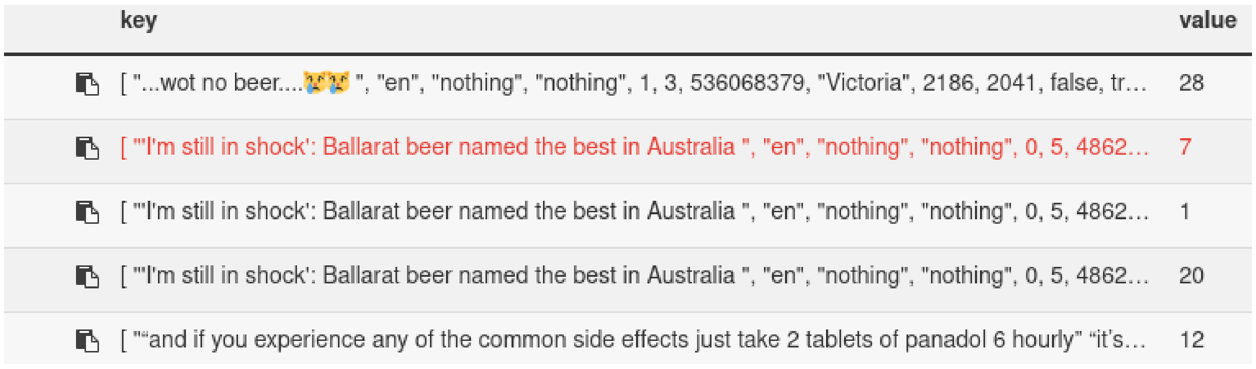
Grunt-couch is used to automatically deploy the design documents on couchdb databases. Four databases are created to store twitter data for different scenario analysis. To automatically deploy these databases, the Couch Directory Tree is designed as the structure in Figure. 13. The Gruntfile.js defines the “couch-compile” task, which processes the Couch Directory Tree, and the “couch-push” task, which pushes the compiled documents to our couchdb database on MRC instances.



```
grunt_couchdb
  all_twitter
    views/all_remove_duplicate
      map.js
      reduce.js
    _id
    language
  beer_twitter
  covid_twitter
  income_twitter
  node_modules
  Gruntfile.js
  package-lock.json
  package.json
```

Figure. 13

Because Twitter harvesters can get some duplicated tweets. Mapreduce functions in each database are used to reduce duplication. As shown in Figure. 14., The map function maps each tweet into {key:value} pairs where the key is the tweet and the value is 1. The "_count" reduce function counts the number of duplicate tweets. Thus, when fetching data from couchdb, the redundant tweets will be filtered.

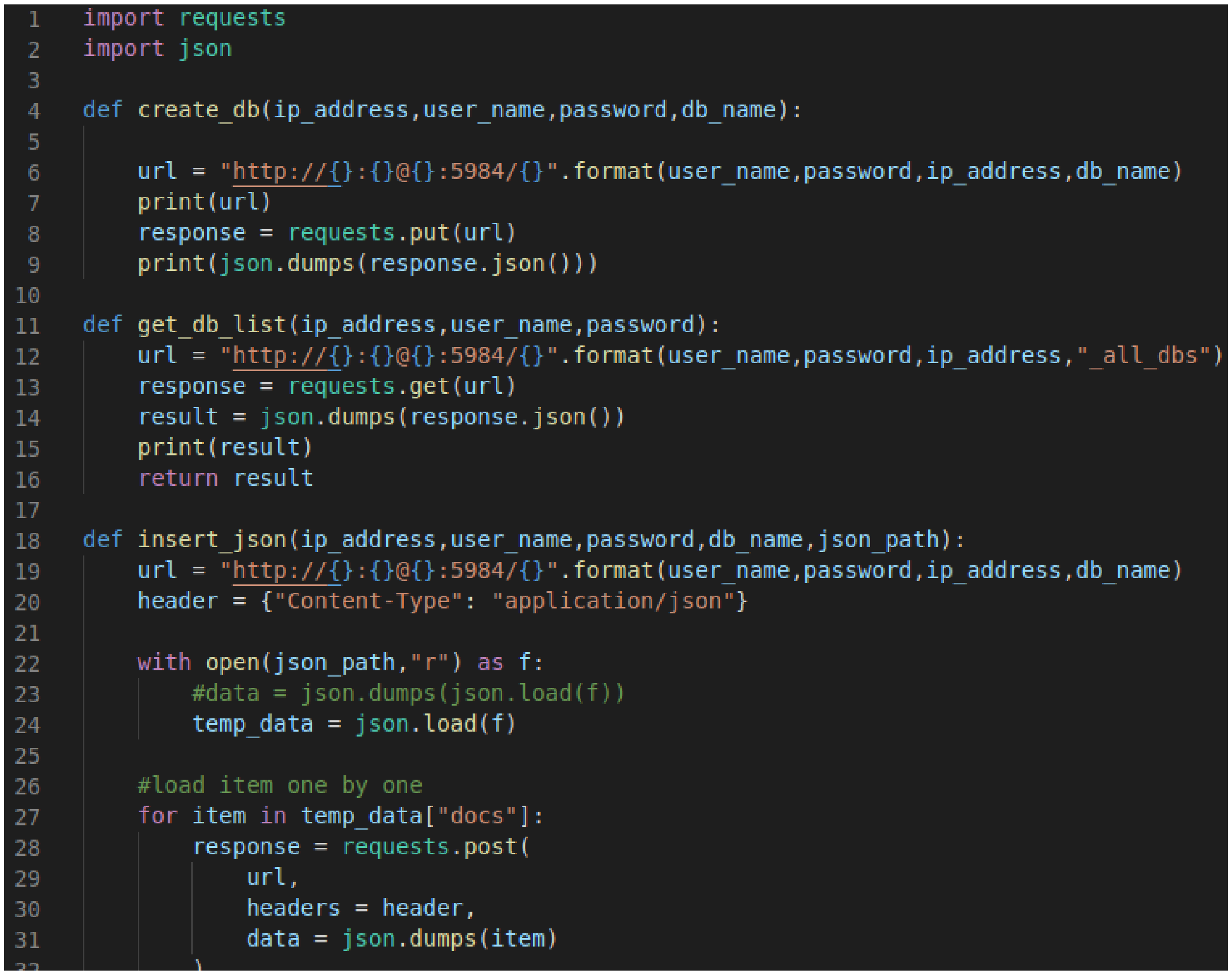


key	value
["...wot no beer....😿😿 ", "en", "nothing", "nothing", 1, 3, 536068379, "Victoria", 2186, 2041, false, tr…	28
["'I'm still in shock': Ballarat beer named the best in Australia ", "en", "nothing", "nothing", 0, 5, 4862…	7
["'I'm still in shock': Ballarat beer named the best in Australia ", "en", "nothing", "nothing", 0, 5, 4862…	1
["'I'm still in shock': Ballarat beer named the best in Australia ", "en", "nothing", "nothing", 0, 5, 4862…	20
["“and if you experience any of the common side effects just take 2 tablets of panadol 6 hourly” “it's…	12

Figure. 14

CouchDB has two ways of communication, CouchDB cURL and CouchDB Fauxton. In this project we use CouchDB cURL to communicate with the database. For example, the command "curl -XPUT "http://user_name:user_pwd@ip_address:5984/db_name" " creates a database named db_name on the specified couchDB server. And the command "curl -XPOST "http://user_name:user_pwd@ip_address:5984/db_name/_bulk_docs" --header "Content-Type:

application/json" --data data.json ” inserts a json file to the specified database. To interact with couchDB databases in an easier way, we encapsulate the curl commands in a python file (Figure. 15).



```
import requests
import json

def create_db(ip_address,user_name,password,db_name):

    url = "http://{}:{}@{}:5984/{}".format(user_name,password,ip_address,db_name)
    print(url)
    response = requests.put(url)
    print(json.dumps(response.json()))

def get_db_list(ip_address,user_name,password):
    url = "http://{}:{}@{}:5984/{}".format(user_name,password,ip_address,"_all_dbs")
    response = requests.get(url)
    result = json.dumps(response.json())
    print(result)
    return result

def insert_json(ip_address,user_name,password,db_name,json_path):
    url = "http://{}:{}@{}:5984/{}".format(user_name,password,ip_address,db_name)
    header = {"Content-Type": "application/json"}

    with open(json_path,"r") as f:
        #data = json.dumps(json.load(f))
        temp_data = json.load(f)

    #load item one by one
    for item in temp_data["docs"]:
        response = requests.post(
            url,
            headers = header,
            data = json.dumps(item)
```

Figure. 15

4.4 Docker

Docker is an open-source platform that can make the development and deployment process easier. It can run applications. The application and its associated resources are packaged as a container in Docker (Rad, Bhatti, & Ahmadi, 2017). The Container is deployed to run on a cloud platform. Container technology has been around for many years, while Docker is considered by far the most successful containerization technology. We packaged the application (Tweets collected using the Twitter API in our project) in a lightweight Docker container and deployed it to a cloud-based environment, the Melbourne Cloud Resource (Rad, Bhatti, & Ahmadi, 2017).

In this project, both the twitter harvester and the twitter analyzer are developed locally and run in a docker container on the MRC instance. The method we used to deploy the docker container is shown in Figure.

16. We built a docker image based on the dockerfile and pushed the image to docker hub. Then, we pulled the docker images from docker hub and ran a docker container based on that image.

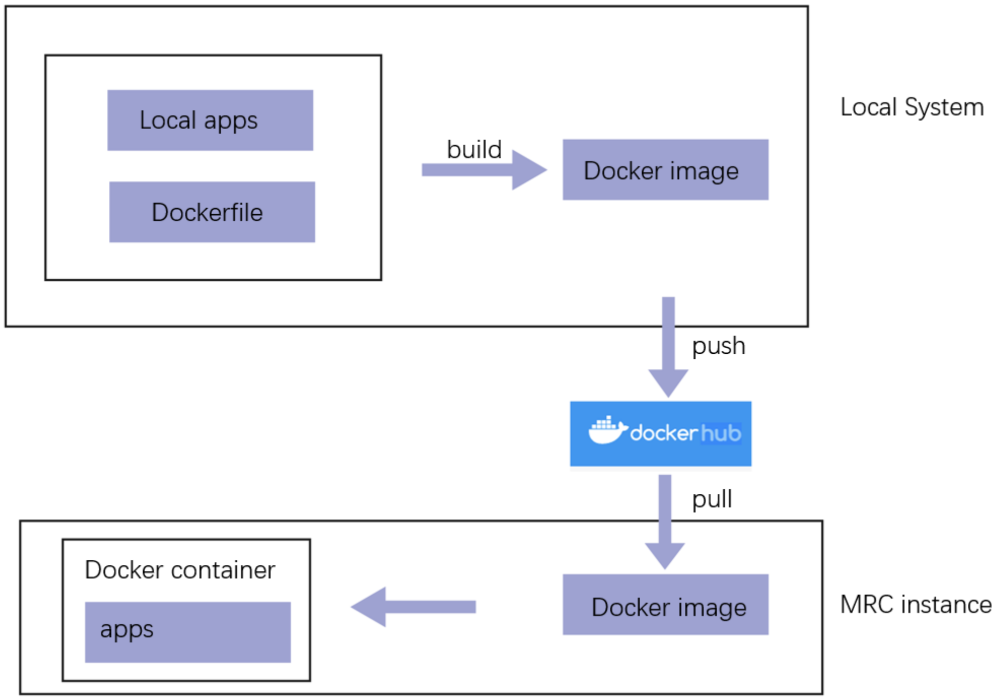
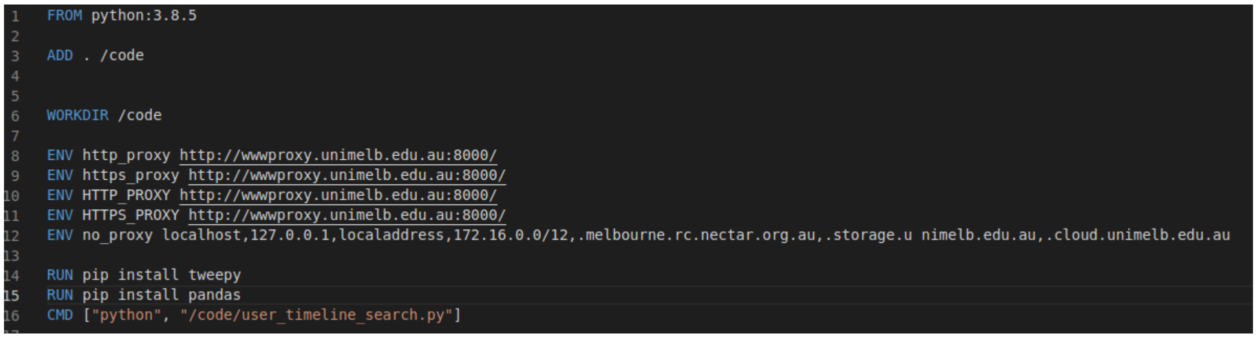


Figure. 16

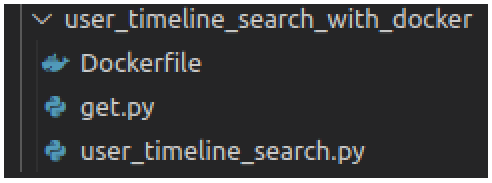
A dockerfile describes the operating environment, the working dictionary, the network environment and the repository needed in the docker container. The CMD command executes the following commands in the docker container once the container is started (Figure. 17).



```
FROM python:3.8.5

ADD . /code


WORKDIR /code

ENV http_proxy http://wwwproxy.unimelb.edu.au:8000/
ENV https_proxy http://wwwproxy.unimelb.edu.au:8000/
ENV HTTP_PROXY http://wwwproxy.unimelb.edu.au:8000/
ENV HTTPS_PROXY http://wwwproxy.unimelb.edu.au:8000/
ENV no_proxy localhost,127.0.0.1,localaddress,172.16.0.0/12,.melbourne.rc.nectar.org.au,.storage.u nimelb.edu.au,.cloud.unimelb.edu.au

RUN pip install tweepy
RUN pip install pandas
CMD ["python", "/code/user_timeline_search.py"]
```

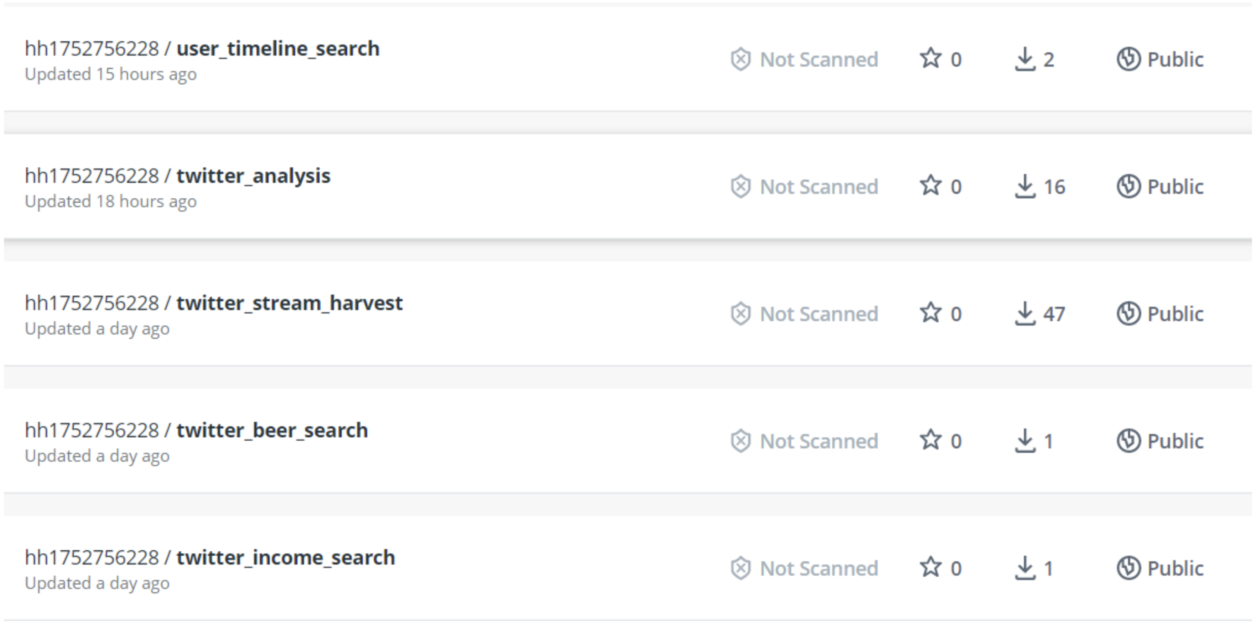
Figure. 17

The docker file and the python files must be under the same dictionary to create a docker image (Figure. 18). And the command “docker build -t image_name” will create a docker image based on the descriptions in the docker file. After that, we login docker hub with command “docker login” and command “docker push image_name:tag” to push the local docker images to docker hub. The images used in our project can be seen in Figure. 19.



```
user_timeline_search_with_docker
  Dockerfile
  get.py
  user_timeline_search.py
```

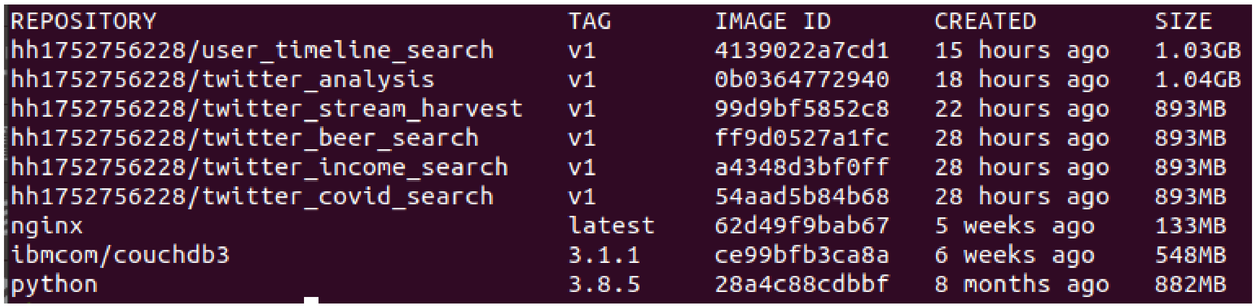
Figure. 18



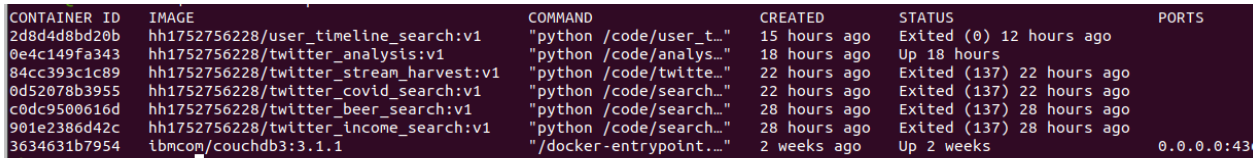
hh1752756228 / **user_timeline_search** Updated 15 hours ago	Not Scanned	0	2	Public
hh1752756228 / **twitter_analysis** Updated 18 hours ago	Not Scanned	0	16	Public
hh1752756228 / **twitter_stream_harvest** Updated a day ago	Not Scanned	0	47	Public
hh1752756228 / **twitter_beer_search** Updated a day ago	Not Scanned	0	1	Public
hh1752756228 / **twitter_income_search** Updated a day ago	Not Scanned	0	1	Public

Figure. 19

To run our code on the MRC instance, we pulled the images from docker hub with the command "sudo docker pull image_name:tag" and built the container with the command "sudo docker run -d --name=container_name image_name:tag". The parameter "-d" means run the container in the backend. The docker images and docker containers on the MRC instance can be seen in Figure. 20. In Particular, a "-v" parameter is used when running the twitter_analysis container. The command "sudo docker run -d -v ~/data:/code/data --name=twitter_analysis image_name" associates the local folder named data in MRC instance to the folder named data in the docker container. Thus, the analysis result can be visited locally on the MRC instance and the front end can get the analysis data easily.



```
REPOSITORY                          TAG       IMAGE ID       CREATED         SIZE
hh1752756228/user_timeline_search   v1        4139022a7cd1   15 hours ago    1.03GB
hh1752756228/twitter_analysis       v1        0b0364772940   18 hours ago    1.04GB
hh1752756228/twitter_stream_harvest v1        99d9bf5852c8   22 hours ago    893MB
hh1752756228/twitter_beer_search    v1        ff9d0527a1fc   28 hours ago    893MB
hh1752756228/twitter_income_search  v1        a4348d3bf0ff   28 hours ago    893MB
hh1752756228/twitter_covid_search   v1        54aad5b84b68   28 hours ago    893MB
nginx                               latest    62d49f9bab67   5 weeks ago     133MB
ibmcom/couchdb3                     3.1.1     ce99bfb3ca8a   6 weeks ago     548MB
python                              3.8.5     28a4c88cdbbf   8 months ago    882MB
```



```
CONTAINER ID   IMAGE                                   COMMAND                  CREATED        STATUS                       PORTS
2d8d4d8bd20b   hh1752756228/user_timeline_search:v1    "python /code/user_t…"   15 hours ago   Exited (0) 12 hours ago
0e4c149fa343   hh1752756228/twitter_analysis:v1        "python /code/analys…"   18 hours ago   Up 18 hours
84cc393c1c89   hh1752756228/twitter_stream_harvest:v1  "python /code/twitte…"   22 hours ago   Exited (137) 22 hours ago
0d52078b3955   hh1752756228/twitter_covid_search:v1    "python /code/search…"   22 hours ago   Exited (137) 22 hours ago
c0dc9500616d   hh1752756228/twitter_beer_search:v1     "python /code/search…"   28 hours ago   Exited (137) 28 hours ago
901e2386d42c   hh1752756228/twitter_income_search:v1   "python /code/search…"   28 hours ago   Exited (137) 28 hours ago
3634631b7954   ibmcom/couchdb3:3.1.1                   "/docker-entrypoint.…"   2 weeks ago    Up 2 weeks                   0.0.0.0:43
```

Figure. 20

5. Data Analyzer

The data analyzer is designed for pulling tweets stored in several databases in CouchDB, analyzing the tweets every 60 mins and storing or renewing the results of json format in the instance which both the data analyzer and the web server allocated in. Therefore, the web server can get the results and display them at the front end easily.

The flow chart and architecture of the data analyzer is demonstrated in Figure 21:

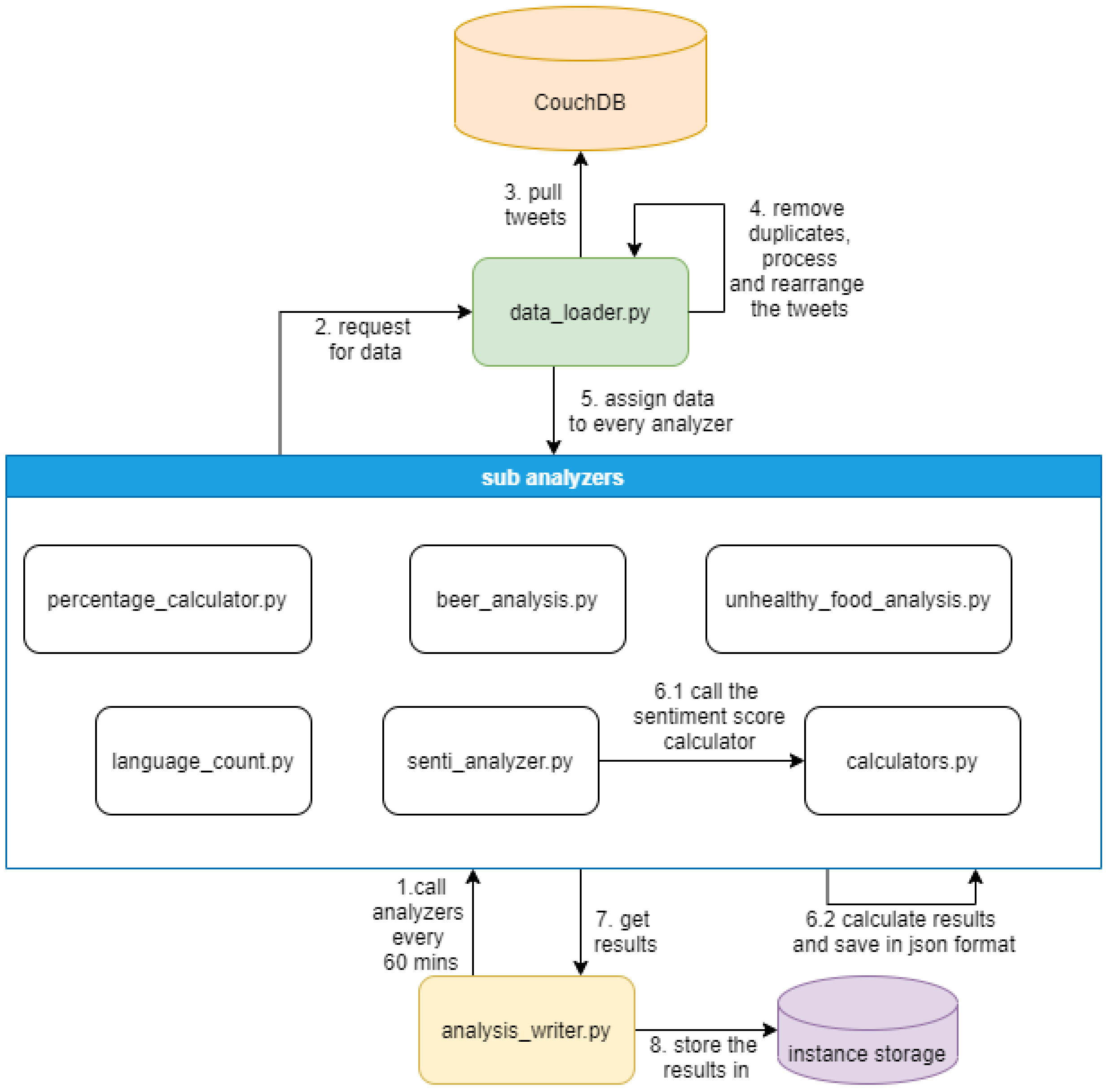

CouchDB
3. pull tweets
4. remove duplicates, process and rearrange the tweets
data_loader.py
2. request for data
5. assign data to every analyzer
sub analyzers
percentage_calculator.py
beer_analysis.py
unhealthy_food_analysis.py
language_count.py
senti_analyzer.py
6.1 call the sentiment score calculator
calculators.py
1.call analyzers every 60 mins
7. get results
6.2 calculate results and save in json format
analysis_writer.py
8. store the results in
instance storage

Figure. 21

The data analyzer is composed of 3 parts, which are data loader, sub analyzers and analysis writer. The tweets are pulled and assigned to sub analyzers by the data loader. The analysis writer will execute every 60 mins for waking the sub analyzers for calculating new results and store the results in instance storage. The design of the data analyzer allows us to display analysis results in the front end dynamically. It also makes it easy for us to add new analysis functions to the data analyzer system or modify existing functions.

6. Frontend Visualization

In order to display the data distribution of different scenarios and the data comparison between cities more intuitively and clearly, we developed a front-end Web application that is mainly based on maps and supplemented by pie charts and other charts. This page realizes the data visualization of this project. It uses Vue as the front-end architecture and Flask as the back-end support to display the sorted processed data in the Web application.

6.1 Flask

Flask is a lightweight framework. Because Flask implements only core functionality, it allows you to create Web applications in a short period of time and flexibly add the functionality you need. The great thing about Flask is that it's simple and flexible, but that's why it's not suitable for large projects. (Ghimire, 2020) Flask does not include an abstraction layer for the database because it is only a prototype framework. At the same time, it does not include any form of validation.

6.2 Visualization

The following show the visualization of our front-end web application. The web application can be access using following link:

http://172.26.131.90:8887/

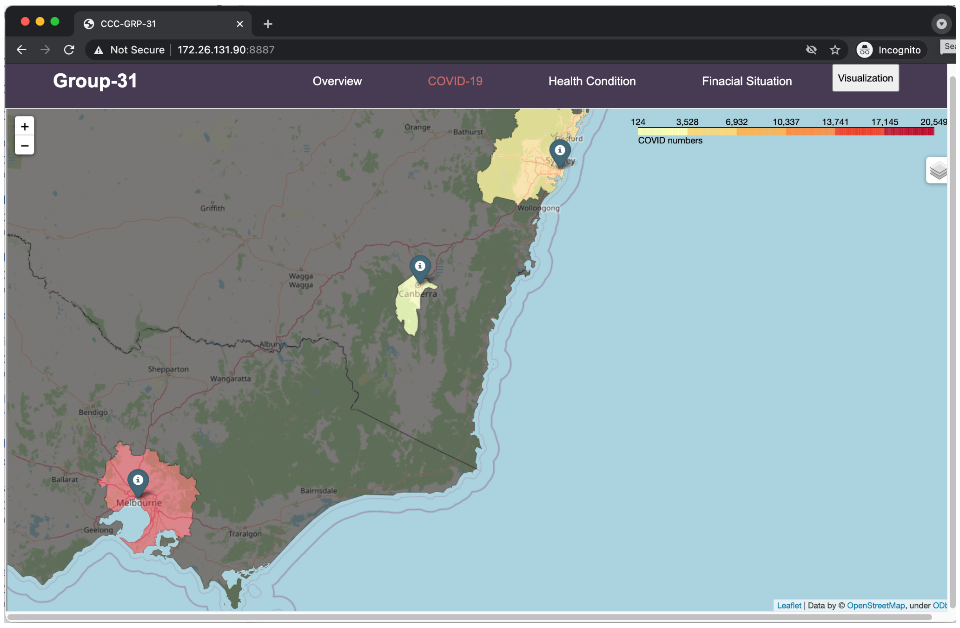


Figure 22

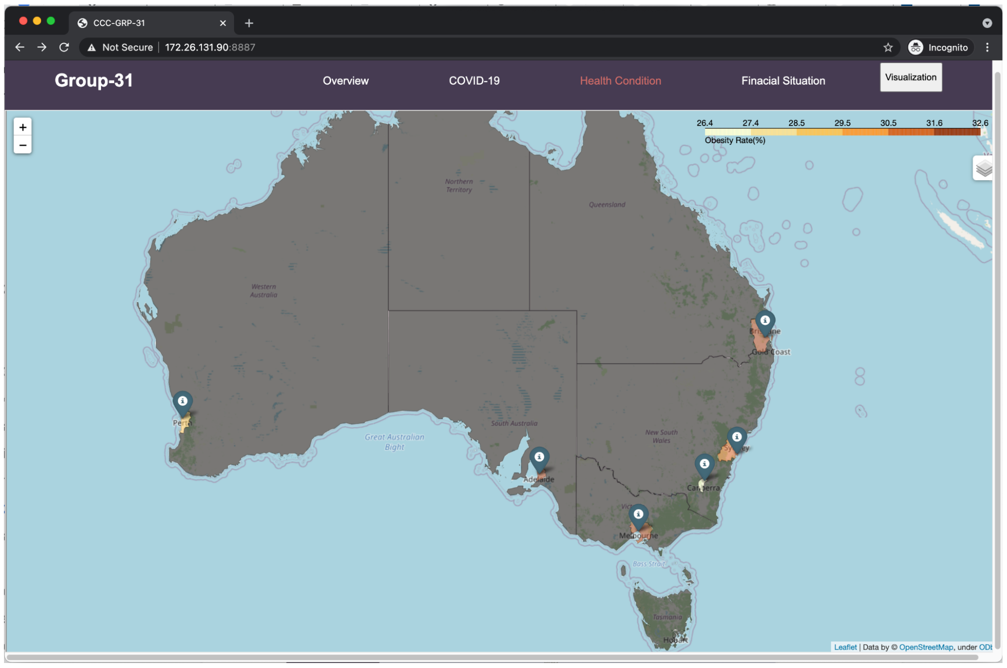


Figure 23

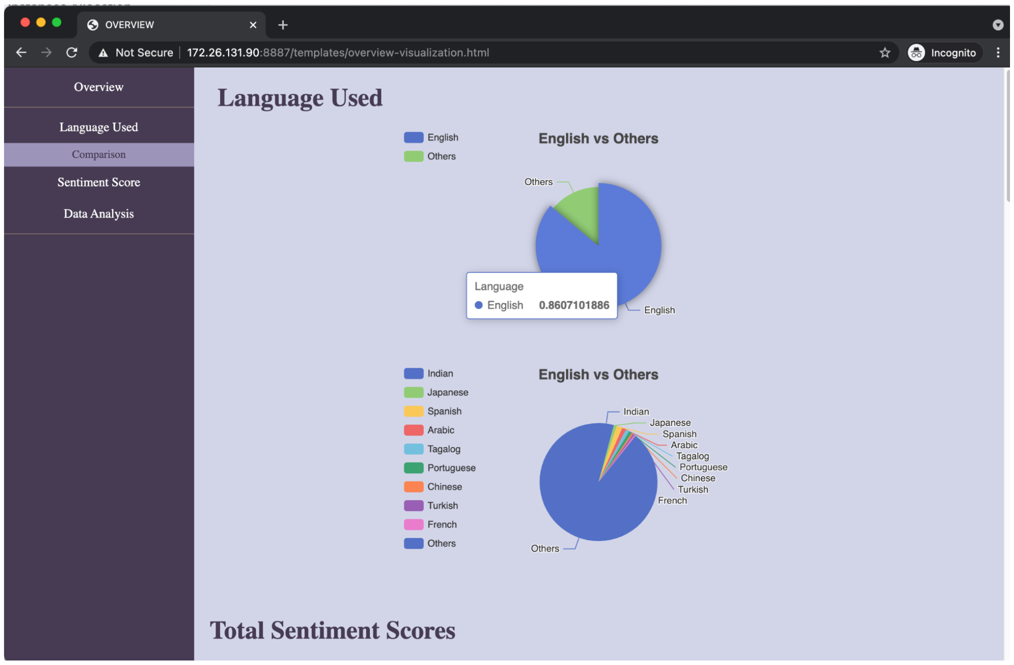


Figure 24

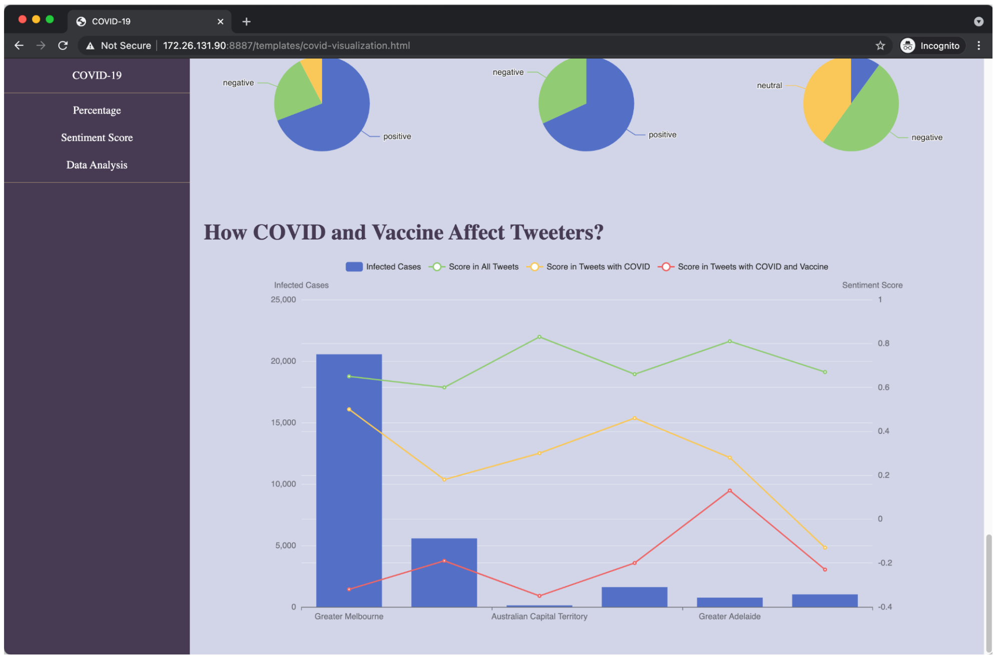


Figure 25

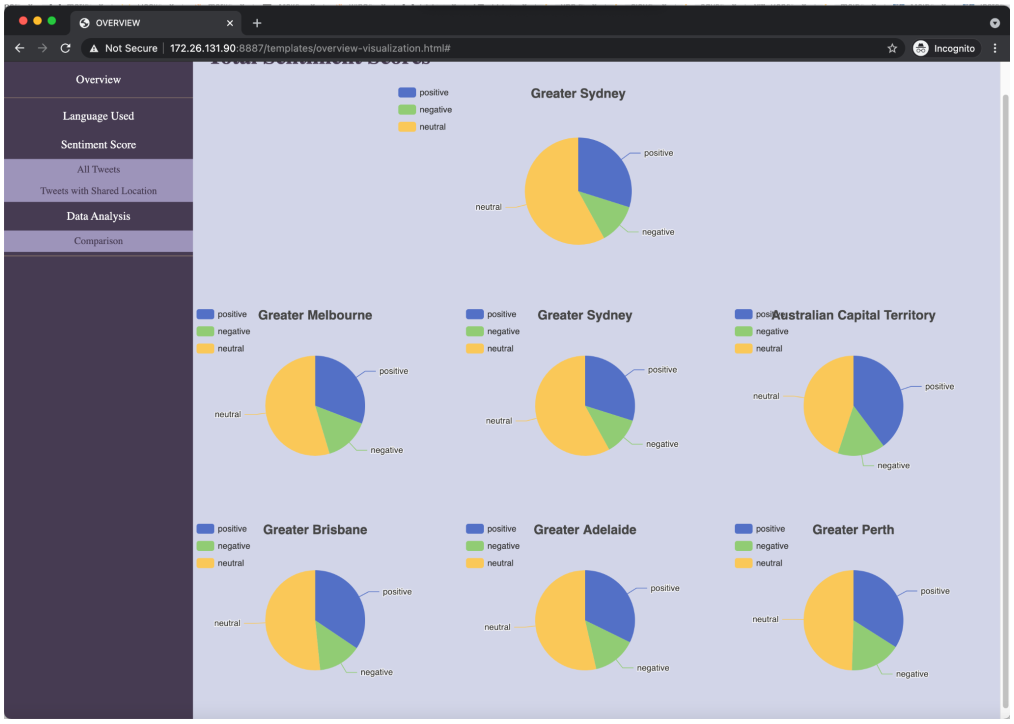


Figure 26

7. Data analysis

We conducted an emotional analysis on the collected tweets, aiming to complete tasks including calculate the happiness index of Twitter users by city, so as to conduct further analysis.

7.1 Sentiment analysis and comparisons between different cities and tweets on different topics

Text sentiment analysis is also called opinion mining, orientation analysis, and so on. To put it simply, it is to analyze, process, and summarize specific texts.

The sentiment analysis approach we select is to calculate sentiment score for each tweet based on the AFINN-111 dictionary. Although it is not a state-of-art approach, it can calculate sentiment scores for texts fast with a relatively high accuracy. We conduct sentiment analysis for tweets streamed with no keyword setting, tweets with geo location revealed, tweets on "covid-19" topic, tweets on "covid-19" and "vaccine' topic and tweets on "income" topic, we get the percentages of negative, neutral and positive tweets in 6 cities across Australia (including Sydney, Melbourne, Brisbane, Adelaide, Perth and Canberra) demonstrated in Figure 22, 23, 24, 25, 26:

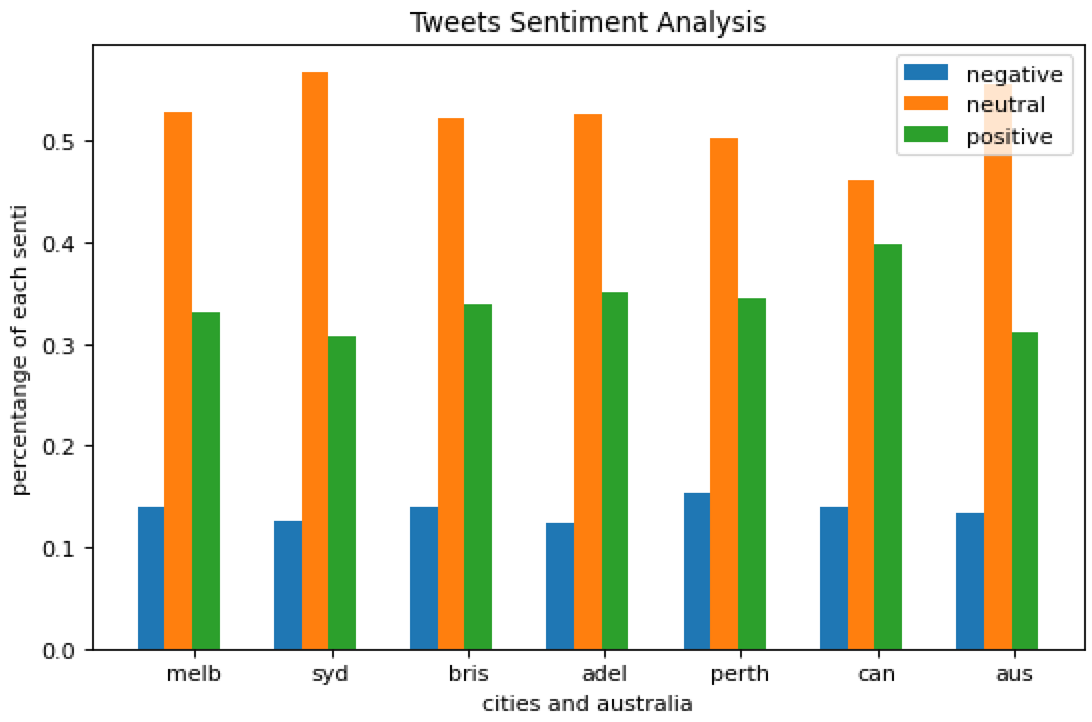


Figure 27

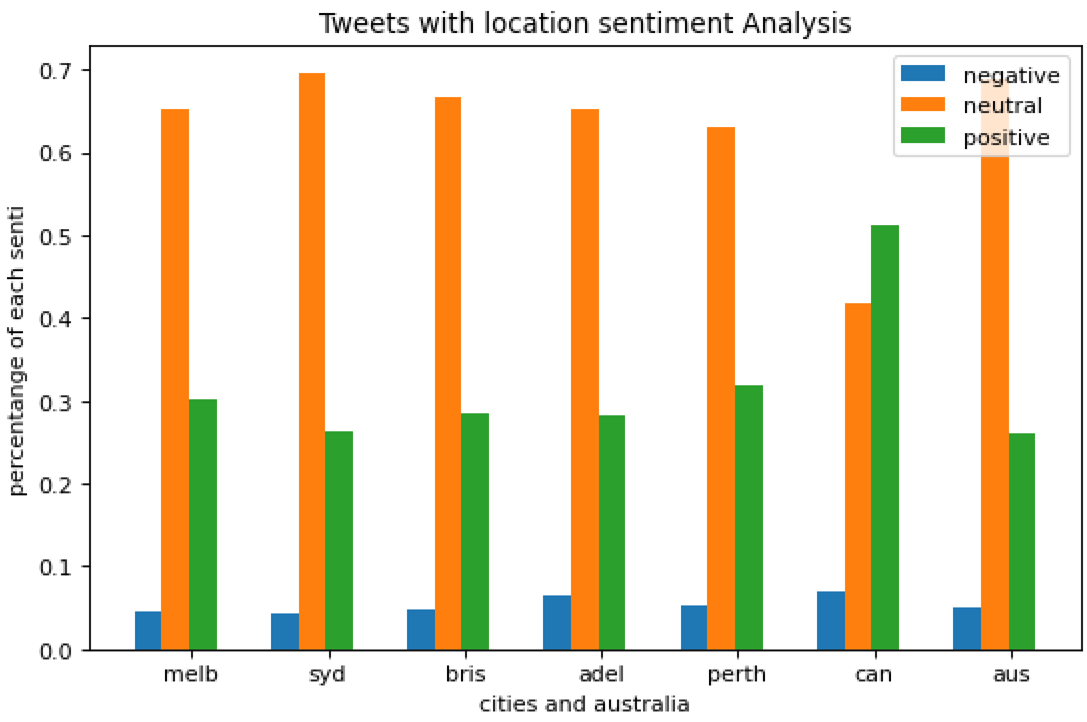


Figure 28

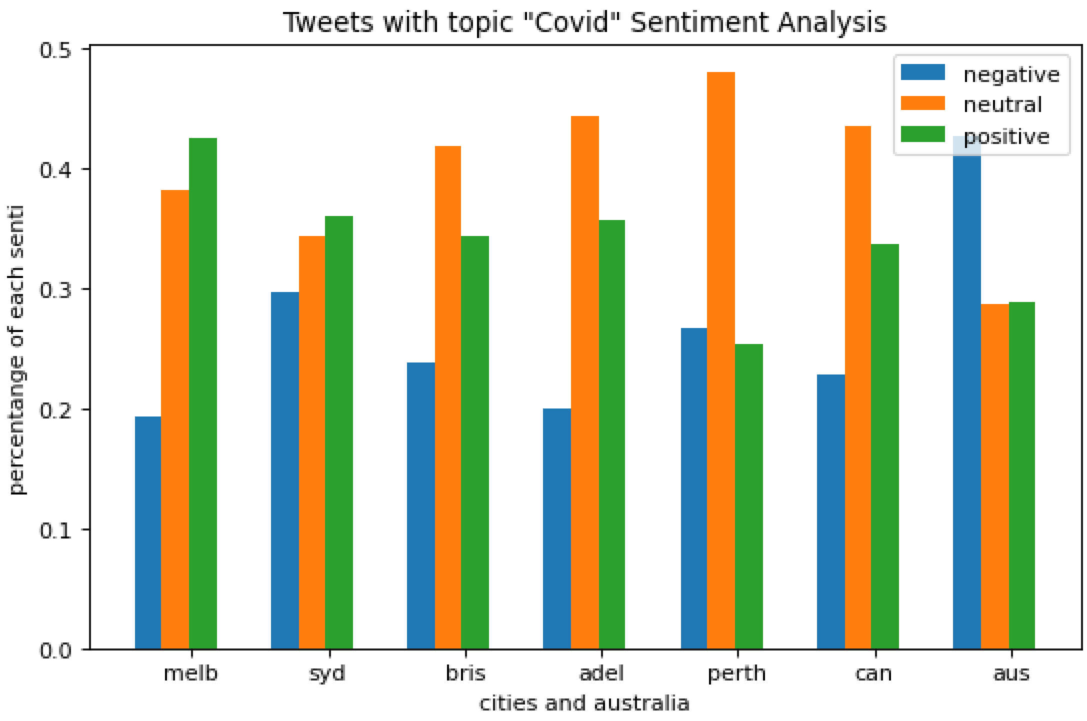


Figure 29

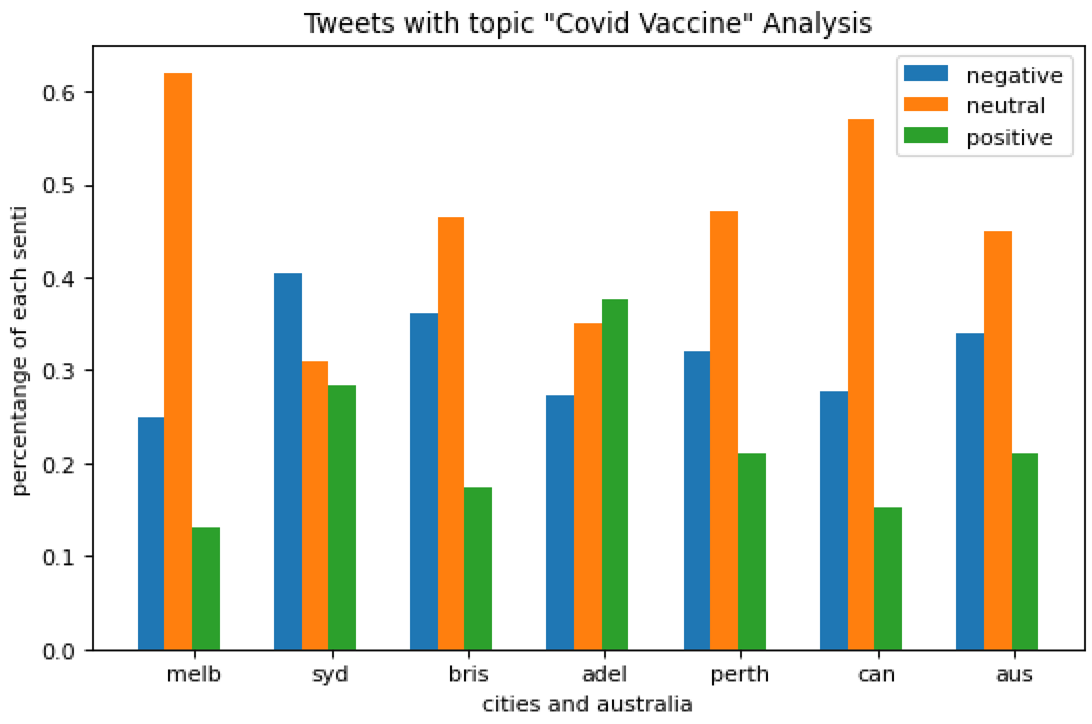


Figure 30

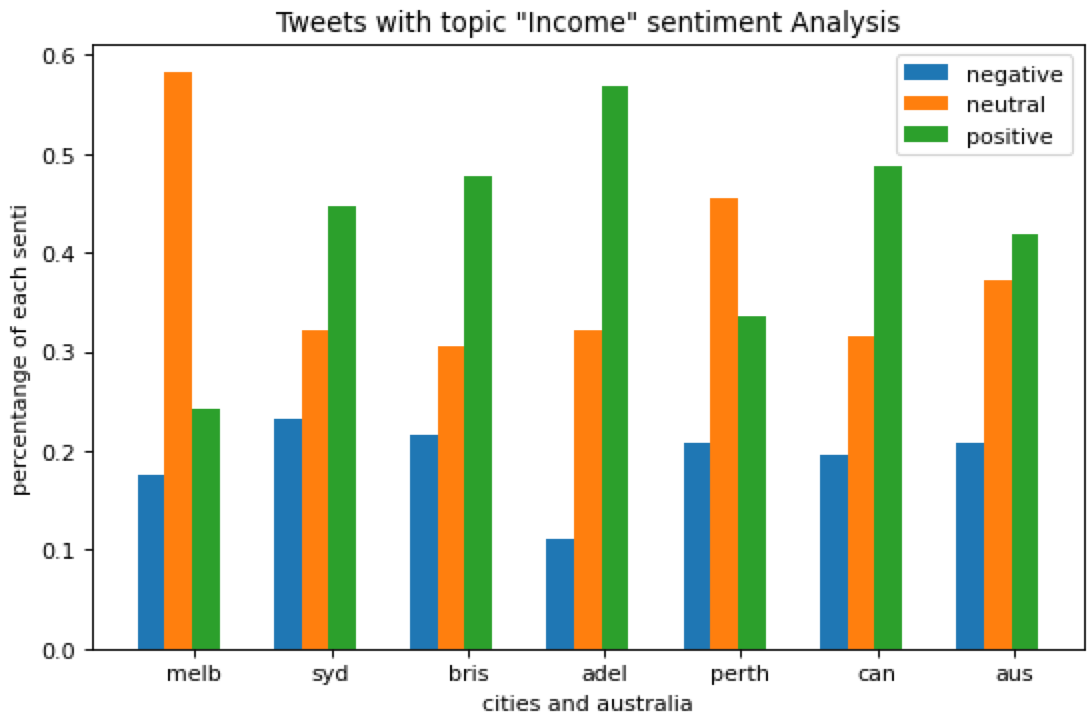


Figure 31

By comparing the percentages of positive tweets and negative tweets between the 6 Australian cities, we can see that users who live in smaller cities with lower GDP(such as Canberra) tend to post more positive tweets than users who live in bigger cities with higher GDP (such as Sydney).

7.1.1 Tweets with geolocation disclosed

Comparing the average sentiment scores between normal tweets and tweets with geo location available (Figure 32),

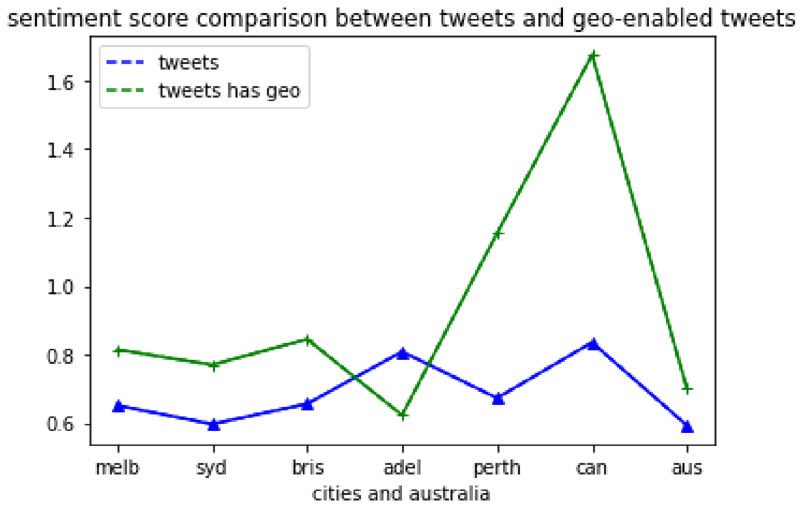


Figure 32

The figure shows that tweets with geolocation disclosed have higher sentiment scores in most cities. This might be due to users being more willing to share where they are when they are travelling, eating food at good restaurants, or just having some pleasant experiences.

7.1.2 Tweets on “covid’ topic and “covid vaccine” topic

Comparing average sentiment scores between normal tweets, tweets on “covid’ topic and “covid vaccine” topic (Figure 33):

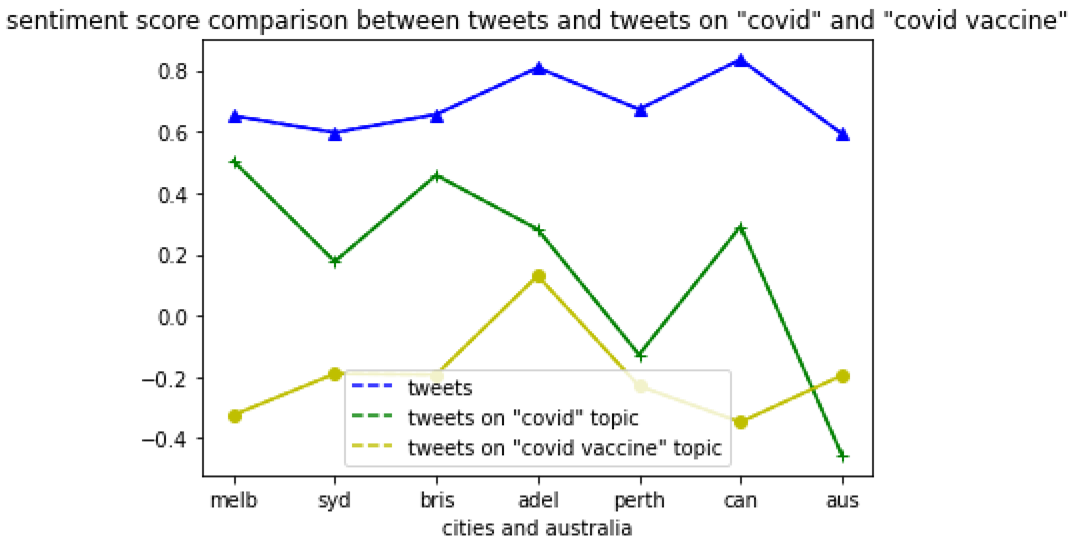


Figure 33

We can find that tweets on “covid” topic and “covid vaccine” topic have lower average sentiment score than random tweets. And it seems that unlike people living in small towns, people living in the 6 big cities of Australia are not optimistic about vaccines.

7.1.3 Tweets on “income” topic

Comparing average sentiment scores between normal tweets and tweets on “income” topic (Figure 34):

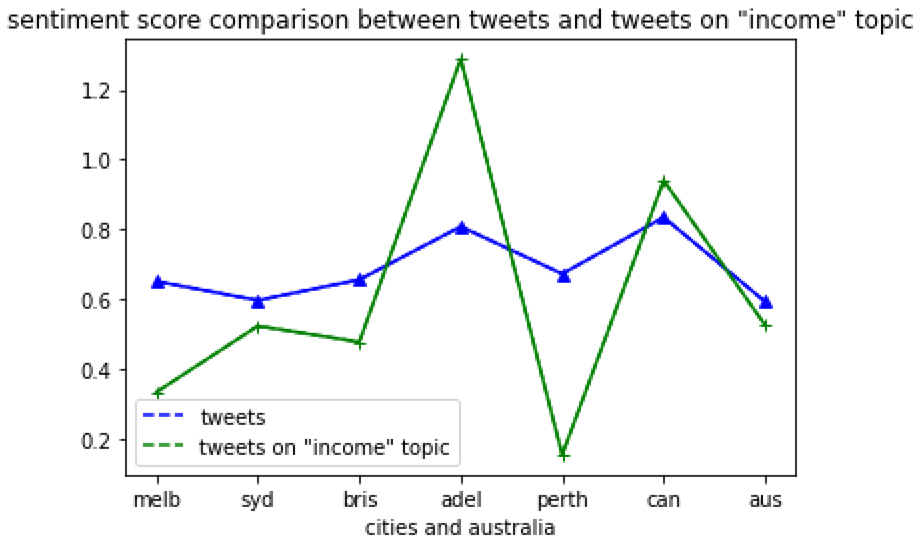


Figure 34

The line chart illustrates that people living in the top 2 largest cities of Australia are dissatisfied with their income. People living in smaller cities, such as Adelaide and Canberra, are more satisfied with their income. This might be because big cities have higher prices for commodities and greater competitive pressure, compared to smaller cities.

7.2 Sentiment scores of tweets on "covid" topic, local infection and mortality

The correlation between the sentiment score of tweets on "covid" topic and the local infection and mortality are demonstrated in Figure 35:

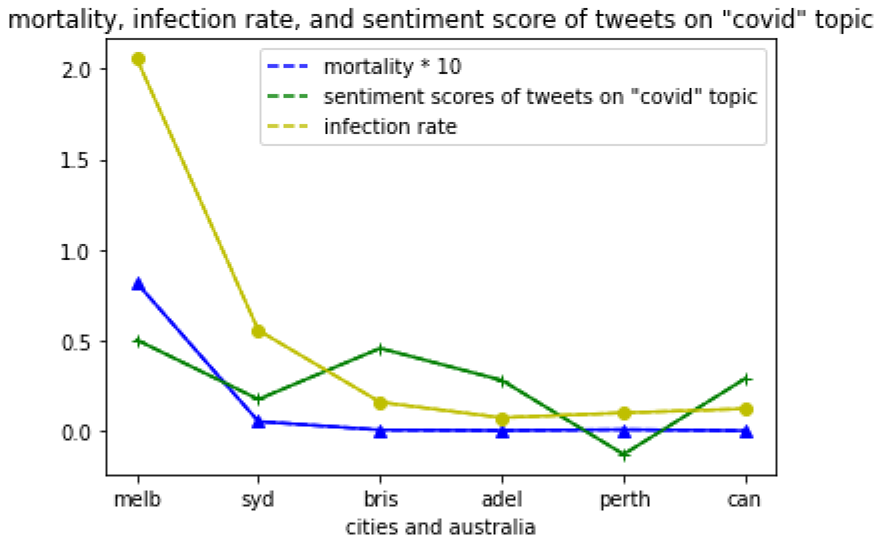


Figure 35

The line chart indicates that there is no correlation between people's sentiment towards covid-19 and the local infection and mortality. People living in Australia's worst-affected city, Melbourne, even have the most positive sentiment towards Covid-19 among the 6 cities.

7.3 Which is more correlated with obesity rate. Beer or unhealthy food?

The correlations between the probability of unhealthy foods (restaurants) or beers being mentioned in tweets and the local obesity rate are demonstrated in Figure 36:

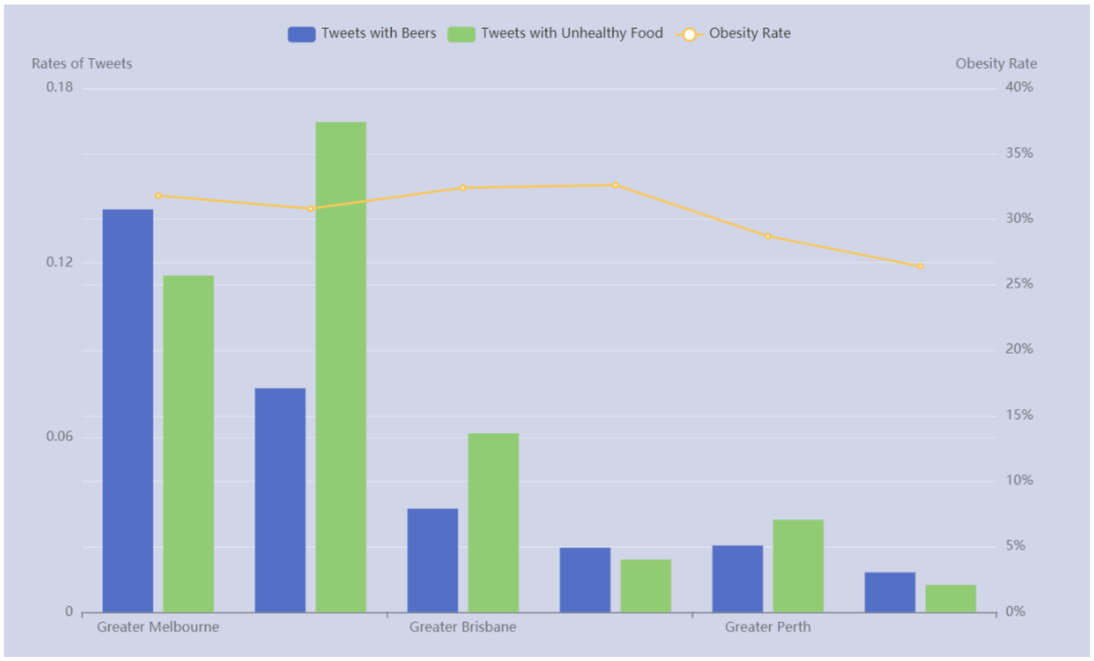


Figure 36

The figure shows that beer has a higher correlation with the local obesity rate, compared to fast food. In cities with lower “beer” mention rates, the obesity rates of them tend to be lower as well.

7.4 Discussion of the results

We predicted that the more prosperous the city, the higher its GDP, the happier people would be. Surprisingly, however, the opposite is true. And as we expected, the mood at the mention of COVID-19 was relatively grim. In addition, mentions of COVID-19 are higher on people's public tweets in cities that have experienced severe outbreaks, such as Sydney and Melbourne. The authors of tweets who were willing to disclose their location also reported higher levels of happiness.

8. Limitation

Our web has the possibility of a single point of failure because the frontend is deployed on only one instance. This means that if this instance fails, our webpage will be completely inaccessible until the web server restarts. When we were choosing scenarios, because of the limited public datasets available on AURIN and the Internet, we had to define the scenarios based on the data sets we found. In addition, the Twitter API has limitations in terms of speed and data volume, although we have collected a large amount of tweets, they are still not enough to get accurate results. Due to the influence of COVID-19, our team members can only work online. Even though we meet and communicate almost every day, the efficiency of communication and collaboration is still not as good as that of Face-to-Face meetings.

9. Conclusion

In this project, we learned about cloud computing concepts and how to deploy the whole system on a cloud platform using automation tools. We also learned to create virtual machines with Ansible, build and start Docker containers on virtual machines, and process a large amount of data with CouchDB. It is clear that cloud computing makes it more convenient to dynamically scale up the system to meet the demand.

Reference

CouchDB, A. Apache CouchDB. *URL https://couchdb. apache. org*.

Ghimire, D. (2020). Comparative study on Python web frameworks: Flask and Django.

Mohaan, M., & Raithatha, R. (2014). *Learning Ansible*. Packt Publishing Ltd

Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 228.

Appendix

Github repository: https://github.com/motianyi/CCC-Assignment2/

Deployment documentation:

https://github.com/motianyi/CCC-Assignment2/blob/main/ansible/README.md

Deployed System(require VPN to access): http://172.26.131.90:8887/

The video of our system and web demo: https://www.youtube.com/watch?v=nx2HKSYtUIQ