COMP90024 Cluster and Cloud Computing
Assignment 2 Report
Australian Gluttony Tweets Analysis

Team37
Chaoxin Wu 828707
Defang Shi 929036
Rongxiao Liu 927694
Xudong Ma 822009
Zijie Shen 741404

Table of content

Abstract

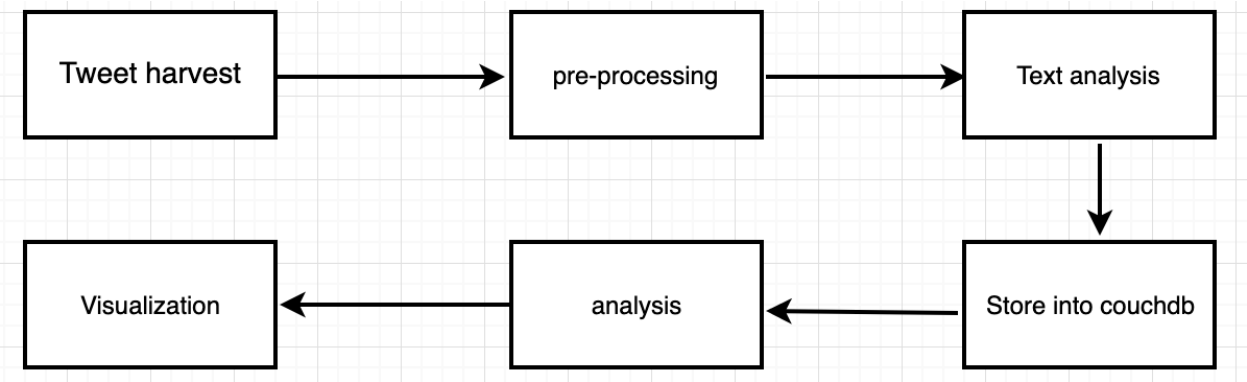
This Report includes all original works of team 37 for Assignment 2 (COMP90024 Cluster and cloud computing), which is about building corroboratory instances on Nectar research cloud and based on which we retrieve information from Twitter in cities across Australia. Finally, we would analyze these data combined with data from the AURIN platform, in order to dig out interesting stories, particularly in the scenario of Gluttony, one of the seven deadly sins.

1. Introduction

Twitter is one of the most common ways for people to express opinions and record daily life. its relatively short context and clear tags are perfect to be stored into a database and analyzed. Twitter, as a kind of social media, has a nature of huge volume and rapid velocity, therefore a perfect material for big data analyzing.

In this project we build a big data analytic system, telling interesting stories about seven deadly sins in the context of twitters within major cities across Australia. We used ansible to automatically deploy four instances on Nectar cloud, which harvest twitters synchronously and store data into one couchdb cluster. We distill information from harvested data and compare them with data on AURIN, in order to tell several interesting stories.

2. Work breakdown



After analyzing the assignment requirements, we break the assignment into several tasks:

System Environment Deployment, CouchDB Deployment, Docker, Tweets Harvesting, MapReduce, Web design, Visualization

These tasks above can be divided into 3 major groups:

(1) Deployment Group (2) Tweets Group and (3) Web Group.

Consider that our team have 5 members, we decide to allocate 2 people into deployment group, 2 people into tweets group; 1 person into web group:

Deployment Group (2)	*Xudong Ma, Defang Shi*
Tweets Group (2)	*Rongxiao Liu, Chaoxin Wu*
Web Group (1)	*Zijie Shen*

Dependency and delivery of each group:

	Dependency	*Delivery*
Deployment Group	*/*	*Deployed system environment*
Tweets Group	*Deployed system environment*	*Processed data*
Web Group	*Deployed system environment & Processed data*	*Web service (map and charts)*

In order to finish the assignment on time, although there are dependency between tasks, we still parallelized these tasks by using our local computer to harvest and process tweets / design web front and visualize demo data. After the previous stage finished, the next group would be able to apply their deliveries and merge the work immediately.

Task allocation

A more specific work allocation shows below:

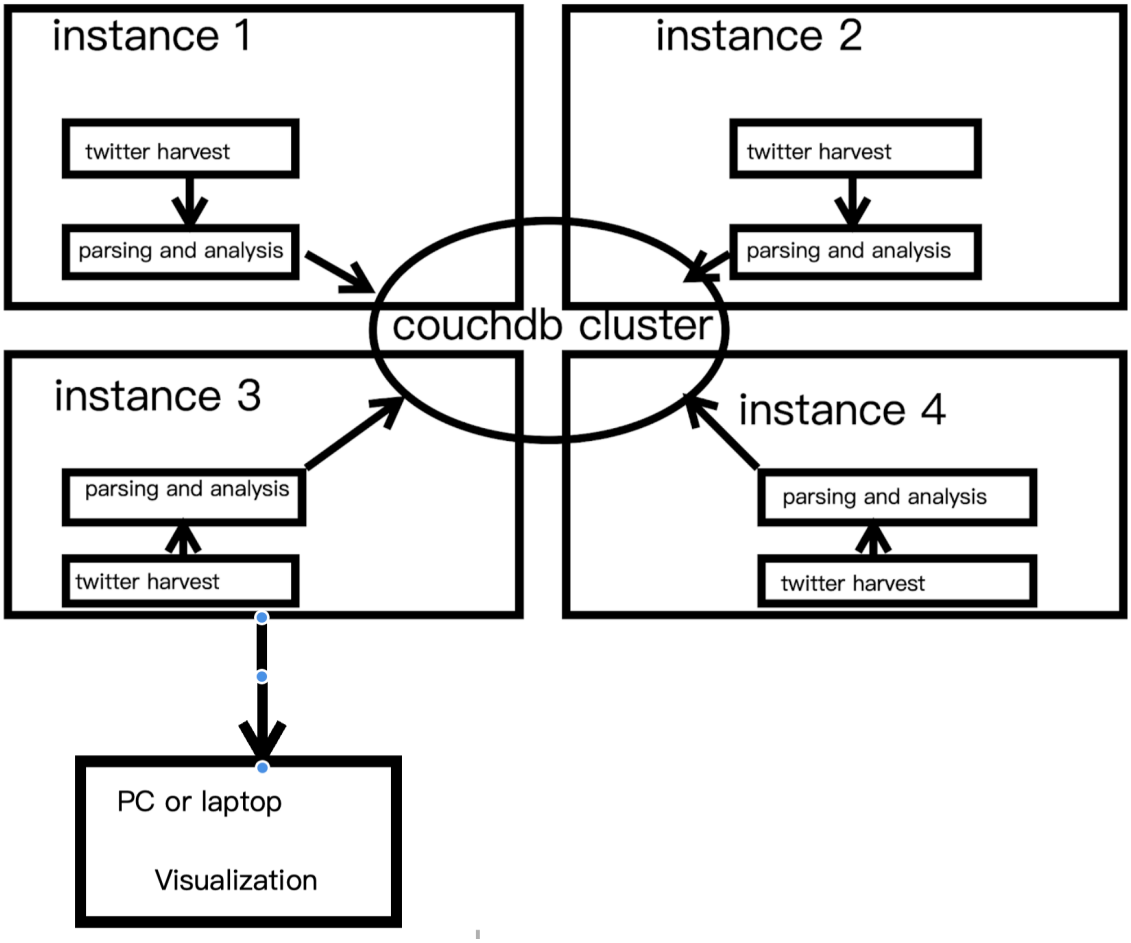
Team Member	*Tasks*
Xudong Ma	*The automation of instances creation, Couchdb cluster deployment, Web API deployment (ansible and docker code)*
Defang Shi	*Couchdb cluster deployment and report writing*
Rongxiao Liu	*Tweets harvesting, Data cleaning and Processing*
Chaoxin Wu	*Tweets harvesting, Data analysis and MapReduce*
Zijie Shen	*Front end deployment*

Team work

Working files (e.g. codes) are shared among team members through the repository in Github (https://github.com/DefangS/CCC-ass2). Team members push their update when they finish a stage of delivery.

In average, we meet every 3 days at project room or booth in unimelb libraries. In each meeting, each member will report their progress from last meeting and decide what need to be done before next meeting (allocate new tasks).

3. System Design



3.1 Automatically Deployment

Majority of our system run on four instances based on Nectar research cloud, which is based on OpenStack.

Compared with other cloud service provider such as AWS and Google cloud, Unimelb research cloud's pro is free, and most of the manipulating operations and system setup procedures are same with other more common cloud, therefore it is a nice tool for students to practice on. One drawback of Unimelb research cloud is that the instances are unstable, namely the access of them may fail sometimes, which would threaten the safety of our data.(Thus we need duplication or backup for important data) Our group is assigned four instances, eight cpus, 36 Gb RAM and 250 Gb volume storage on Nectar(National eResearch Collaboration Tools and Resources) for this

project. We allocate these resources evenly on each virtual machine, namely each with two cpus, 9Gb RAM and 60 Gb volume storage, because the tasks of capturing, querying (through MapReduce function) and visualization of data are also separated evenly across the instances.

Instead of manually deploy instances by clicking on the website, ansible scripts are applied to automatically deploy them on nectar, including couchdb installation and required environment for latter twitter crawling.

Displaying 4 items

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
mytest4	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.38.174	uom.mse.2c9g	xudongm	Active	melbourne-qh2-uom	None	Running	13 hours, 21 minutes	Create Snapshot
mytest3	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.37.189	uom.mse.2c9g	xudongm	Active	melbourne-qh2-uom	None	Running	13 hours, 22 minutes	Create Snapshot
mytest2	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.37.241	uom.mse.2c9g	xudongm	Active	melbourne-qh2-uom	None	Running	13 hours, 22 minutes	Create Snapshot
mytest1	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.38.153	uom.mse.2c9g	xudongm	Active	melbourne-qh2-uom	None	Running	1 day, 16 hours	Create Snapshot

Displaying 4 items

3.2 CouchDB

CouchDB is selected for implementing the system, where multi-instances across nectar cloud would work cooperatively to crawl data together. We apply four instances as dbservers. These instances' couchdbs are in the same cluster and the data are stored in a duplicated way, which means any node's failing will not influence the safety of the whole system. Also, CouchDB cluster has a natural of transparency, namely application could fetch or upload data on any of the instances, without knowing where exactly the data are stored.

More details about mapReduce functions applied are listed in latter sections.

3.3 Tweets Harvester & Processing

In this part, we plan to harvest tweets which are related to foods (Gluttony) from 4 Australian cities (Sydney, Melbourne, Brisbane and Perth). 4 harvesting programs will be running at 4 instances at the same time.

Twitter provides two official APIs - Search and Stream for developers to harvest tweets. The programming language we choose is Python3. The package used to harvest tweets is 'tweepy 3.7.0' and the package used to save tweets to couchdb is 'CouchDB 1.2'.

To used the Twitter API, we first register 4 apps in Twitter Developer website. These apps include keys and tokens that used to authorize our access to tweets.

Methods and Data sources

(1) user_timeline (excluded)

The first method to harvest tweets that we considered is to search by user_timeline. However, this method will be much biased because one user's location and preference may be fixed and not random. It is not suitable to be used to represent one city's performance.

(2) Stream API (excluded)

The second possible method is to harvest through Stream API provided by Twitter. We can collect real-time posted tweets through Stream API. However, it must filter the tweets posted all over the world, which results in a very limited number of tweets returned every minutes.

(3) Search API (applied)

The next harvesting approach is through a time period by Twitter Search API. It can quickly harvest tweets in one area. Although we can only access to tweets in previous one week, the amount of tweets is enough for us to analyze.

We set parameters in Cursor object that used to get tweets that we want. Considering that a part of tweets may be truncated due to their length, we set parameter tweet_mode='extended' to show full_text. Another limit of search API is request limit. We can only send 1500 requests per 15-minute, thus we set wait_on_rate_limit as True so that when the limit reached, the program will wait until 15 minutes rather than throwing an exception.

Rather than saving the original tweets into CouchDB, we choose some attributes that may contribute our analysis and build a new dictionary to save into CouchDB. To avoid saving duplicate tweets that are already in the CouchDB, the primary key "_id" of doc is set as the value of tweets ID "str_id".

(4) provided historical data in database (applied)

After we harvest recent tweets, two main problems occurs. The first problem is that the latest data in AURIN about our topic is in 2014, while we can only collect tweets in 2019 through Twitter API. Another problem is that most of tweets do not contain geo location information (only 1/1000 have 'geo'). Then we find that Appendix in the end of assignment requirement provides us access to historical tweets from 2014 to 2018. Therefore, we download the tweets in 2014 and 2015 and then upload then to CouchDB as historical tweets. Furthermore, these tweets are rebuilt as the same format as recent tweets that collected from the previous stage.

Topic recognition

When harvesting tweets, we need to analyze whether it is related to our topic (Gluttony). We assume that only if there is a food related word in text of this tweet, it is a gluttony-related tweet. Following this idea, we built a food list that includes 1100+ food-related words. Furthermore, we processes the original text to achieve a higher accuracy. After recognizing these food-related word, they will be recorded as an list of word and be a new attributes "foods" in this tweet. If there is no food-related words been found, "foods" will be an empty list.

Data cleaning & formatting

Since the speed of twitter capture is relatively slower than the speed of parsing and analyzing harvested data, each tweet is analyzed before it being stored into couchdb, rather than storing them first and analysis them all afterward. Namely, once a tweet is captured by one of our instance, it will be parsed and analyzed whether it is or not relevant to our topic, gluttony.

The final attributes in a json file includes following parts.The useful attributes are circled in red.

"_id" is the unique identifier for each doc, which is also an important method to avoid recording duplicate tweets.

```
{
    "_id":"1126626584788516874",
    "_rev":"1-f2630537f163f9fdf9162ee1338cfbc4",
    "created_at":"Thu May 09 23:14:44 +0000 2019",
    "full_text":"RT @itsnaderi: 808's are my mental health therapy",
    "entities":{ },
    "source":"<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Android</a>",
    "user":{ },
    "geo":null,
    "coordinates":null,
    "place":null,
    "lang":"en",
    "retweet_count":6,
    "favorite_count":0,
    "is_quote_status":false,
    "quoted_status":null,
    "city":"Sydney",
    "weekday":"Thu",
    "month":"May",
    "day":"09",
    "hour":"23",
    "year":"2019",
    "foods":[ ]
}
```


Fig. rebuilt tweet and useful attributes

"user" include all information about the user who posts this tweet. We may want to know something behind each user.

"geo" and "coordinates" are the significant evidences that we use to visualize amount of tweets in each areas on maps.

"city" identify which city the tweet was posted.

"weekday", "month", "day", "hour" and "year" are hierarchies of time, which is convenient for us to analyze phenomenon in a certain period.

The last key is "foods". This is a new key we added when harvesting, which is the most important evidence to show result.

Finally, we collect tweets in 4 cities from two source: (1) recent 3-day tweets by Search API and (2) 2014-2015 tweets in given database. They are saved in 8 databases in CounchDB: "Sydney_past", "Sydney_recent", "Melbourne_past", "Melbourne_recent", "Brisbane_past", "Brisbane_recent", "Perth_past" and "Perth_recent"

Remote Commands

Considering that copying, logining and running files in 4 instances are too complex, we decide to run one shell script to achieve remote operations through ansible. The shell script to start harvesting called "run.sh" . To execute it, we need to run "./run.sh (database_IPaddress)" on remote instances. After that, harvesting works will start.

When finishing harvesting, the databases in CouchDB looks like the following:

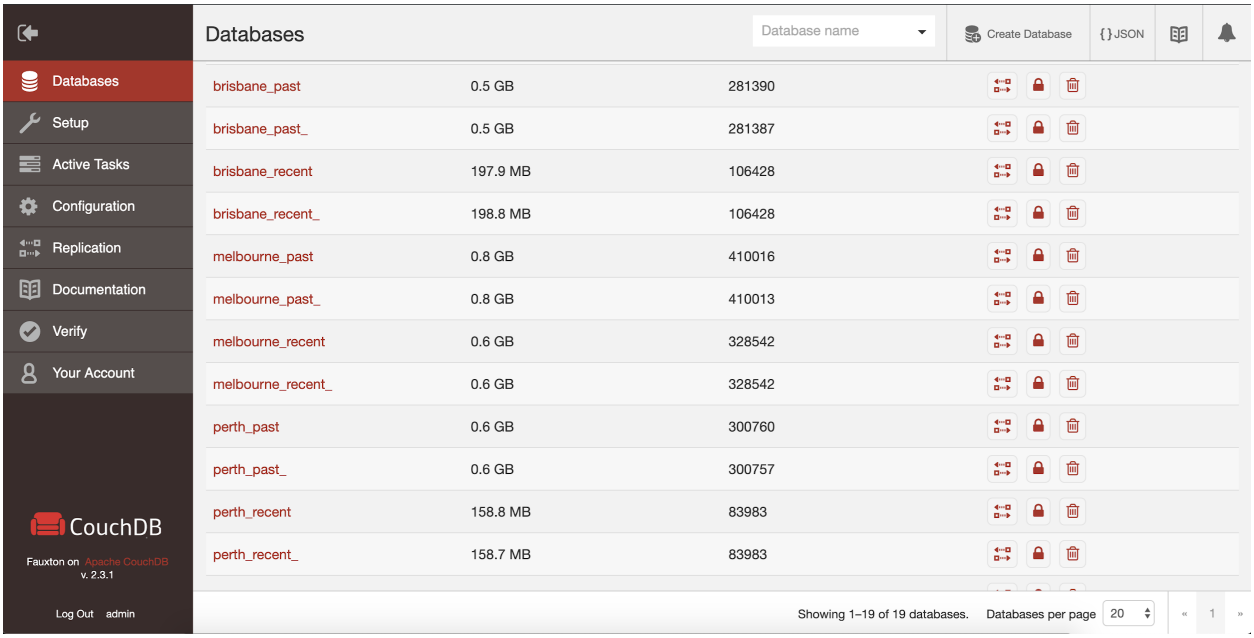
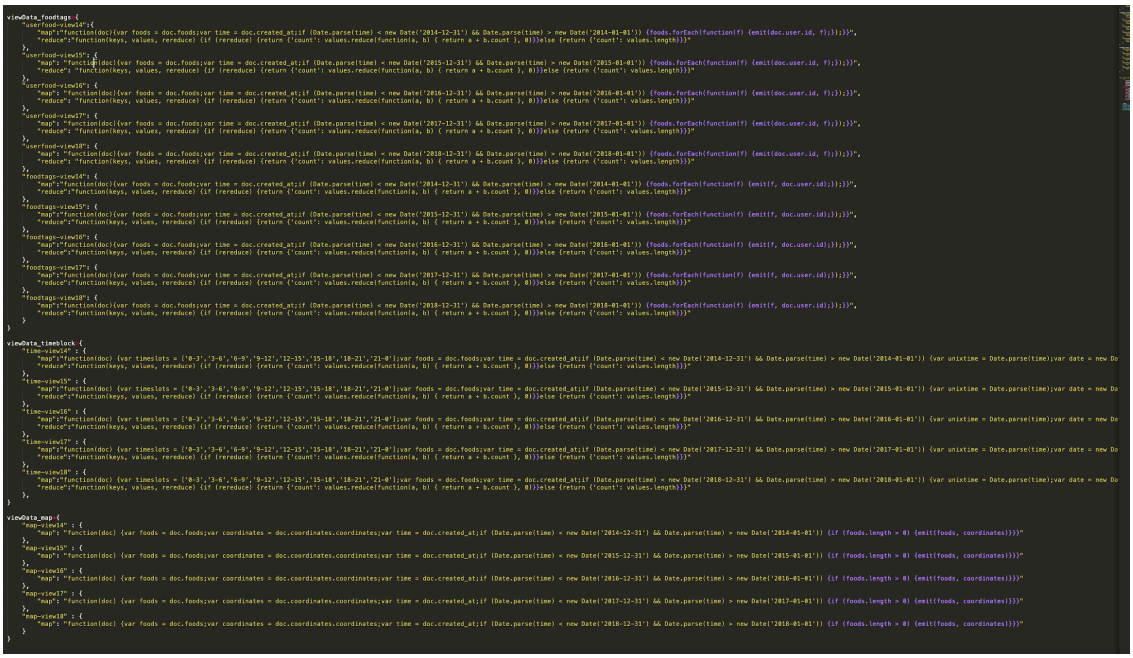


Fig. Databases in CouchDB

3.4 MapReduce

We apply mapReduce functions to generate views, in order to retrieve relevant information we demand from couchdb. Map functions is to allocate certain information into database and reduce function is to gather required information and generate a view for front end website to use.

As below is some of our mapReduce functions used in the project.



3.5 Web and Data Visualization

3.5.1 Main Tools and Libraries

The web application is built upon ReactJS (a javascript library that allows reusability of front-end elements), Redux (a store for states for better management of them when the application scales up) and Bootstrap (a toolkit for designing responsive user interface). The main reason why ReactJS is chosen to be the main tool involved in this project is due to the fact ReactJS utilizes component-based approach for the convenience of allowing smaller scale of elements on a page to be used in other pages and there will be many small components need to be reused over and over again in this project such as popup windows, clustered points and statistical diagrams. Additionally, ReactJS is useful in improving response time when complex interactions might be needed because of its single page feature and this project may need some amount of interactions on the map page. Therefore, it is beneficial to use ReactJS in this project for the better development and performance

Also, webpack, which is a module bundler, is used to minify all relates files such is, css and html for improving the performance of the website. And when displaying map views, this application take advantages of a wide range of functionalities provided by mapbox which features in adding every dataset as a layer and allows user to select

preferred dataset to be shown. By doing this, the analyzed data can be demonstrated in a clearer way.

3.5.2 Main Layers on the map

- *Number of people who are obese in 2014 provided by AURIN.*
- *Number of people who are overweight in 2014 provided by AURIN.*
- *Clustered gluttony related tweets in Melbourne, Sydney, Brisbane and Perth tweeted in different time periods including 0-6am, 6am-12pm, 12pm-18pm, 18pm-24am.*

3.5.3 Charts

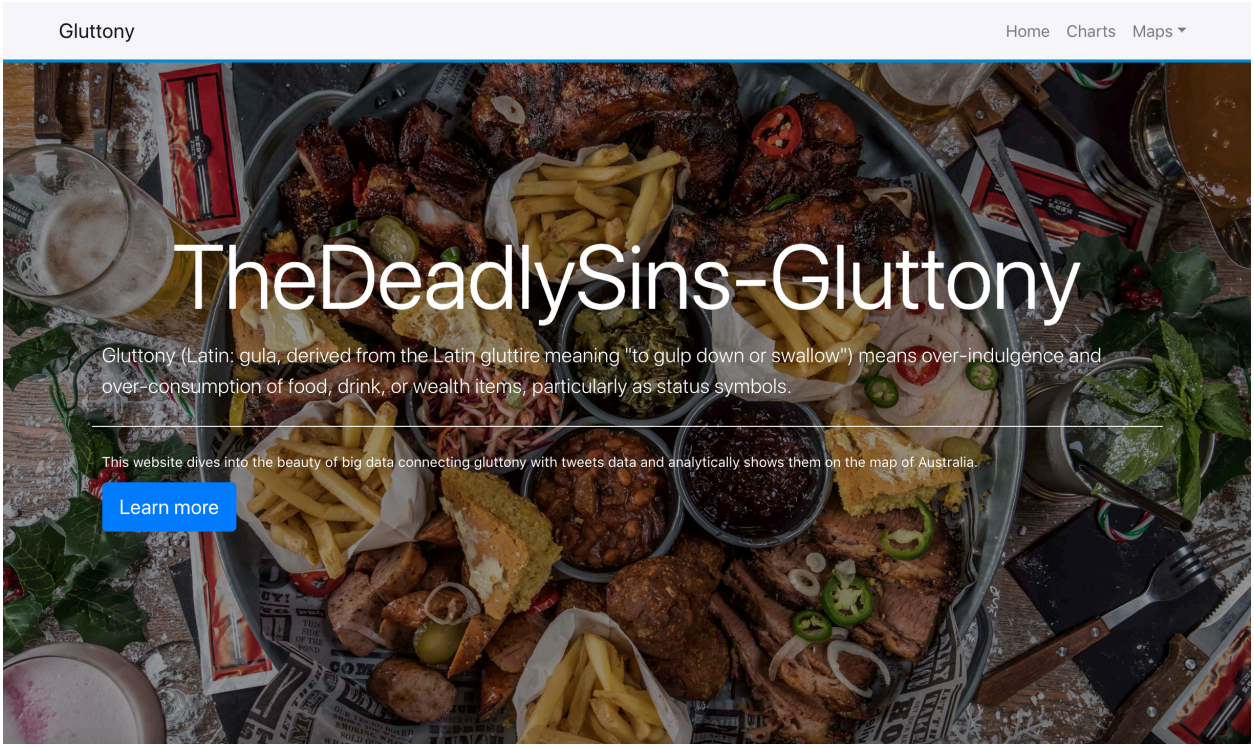
- *Number of people who are considered related to obesity according to the tweets for different cities in 2014, 2015 & 2019*
- *Top 5 food tags for different cities in 2014, 2015 & 2019*
- *Number of gluttony tweets for different cities in different time periods in 2014, 2015 & 2019*

4. User Guide

1. *Firstly, you need to create instances, we use ansible to do this process automatically. You just need to run the line regarding **instances_create.yml** inside run.sh.*
2. *Secondly, you need to download docker for each instance. You just need to run the line regarding **docker_install.yml** inside run.sh (as the instances need two or three minutes to prepare for visiting so it is better to wait several minutes after the instances were created successfully).*
3. *Thirdly, you need to establish a couchdb cluster. Just run the line regarding **couchdb_install.yml** inside run.sh .*
4. *After that, you need to do the data harvest. Just run the line regarding **data_havest.yml** inside run.sh. As we use a large amount of data to do analyze, this process may cost several hours to finish.*
5. *The last step is to publish our web service. You just need to run the line regarding **services_deploy.yml** inside run.sh.*

6. *Finally, Open your browser and type **'http://ip-of-the-webserver:80'** into the address blank. (As we already deploy the whole system so you can just use own webserver's ip to access the service.*

   Now you should see the website below:



The homepage is a description of our team. You could also click the buttons at the top right corner to get statistic result as well as graphs, maps and other cool things we have got.

5. Discussion about Interesting stories

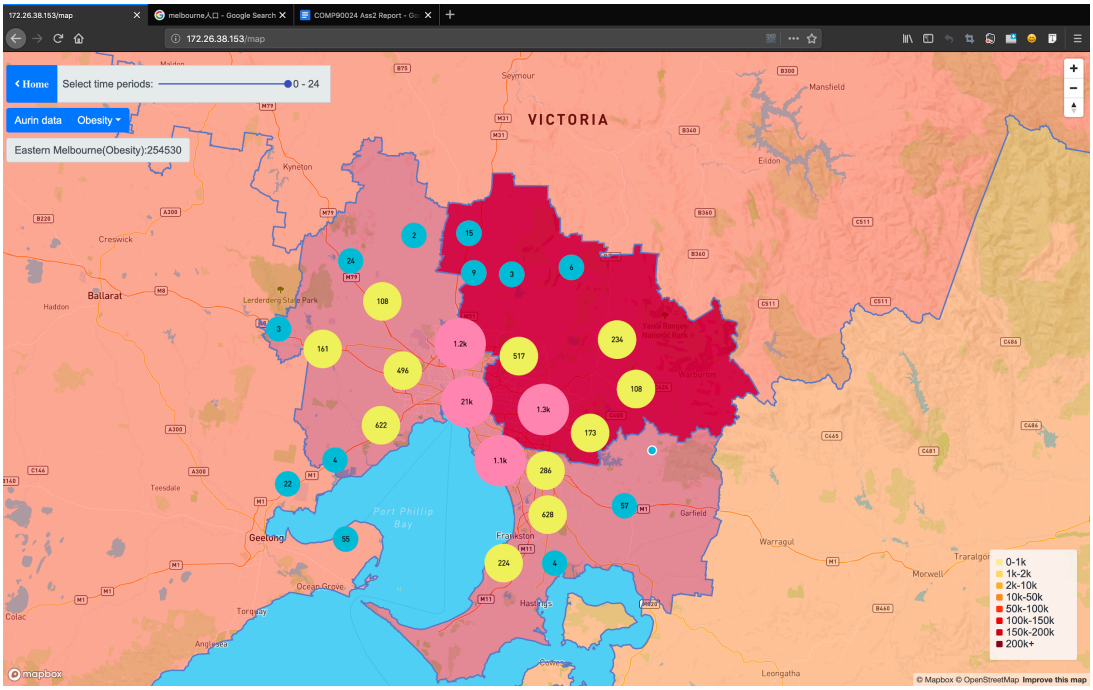
The identifier we applied to judge whether a tweet is relevant to gluttony or not is relatively straight. We firstly crawl 1000 popular food names in Australia, then build a keywords set containing all of them. Any tweet that includes any of these keywords are considered relevant.

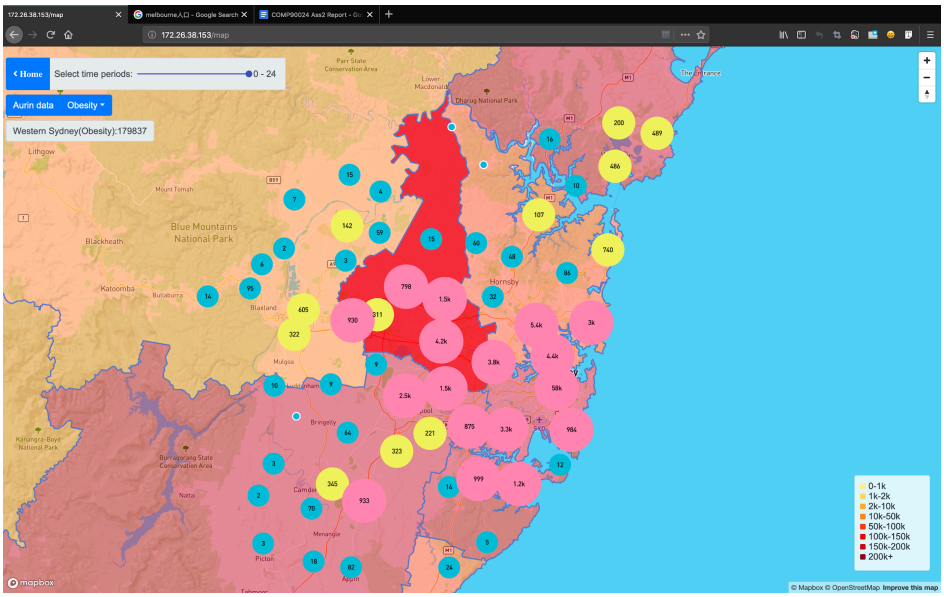
1. *The more a district with fixed acreage has tweets about food are, the higher rate of obesity/overweight it has.*

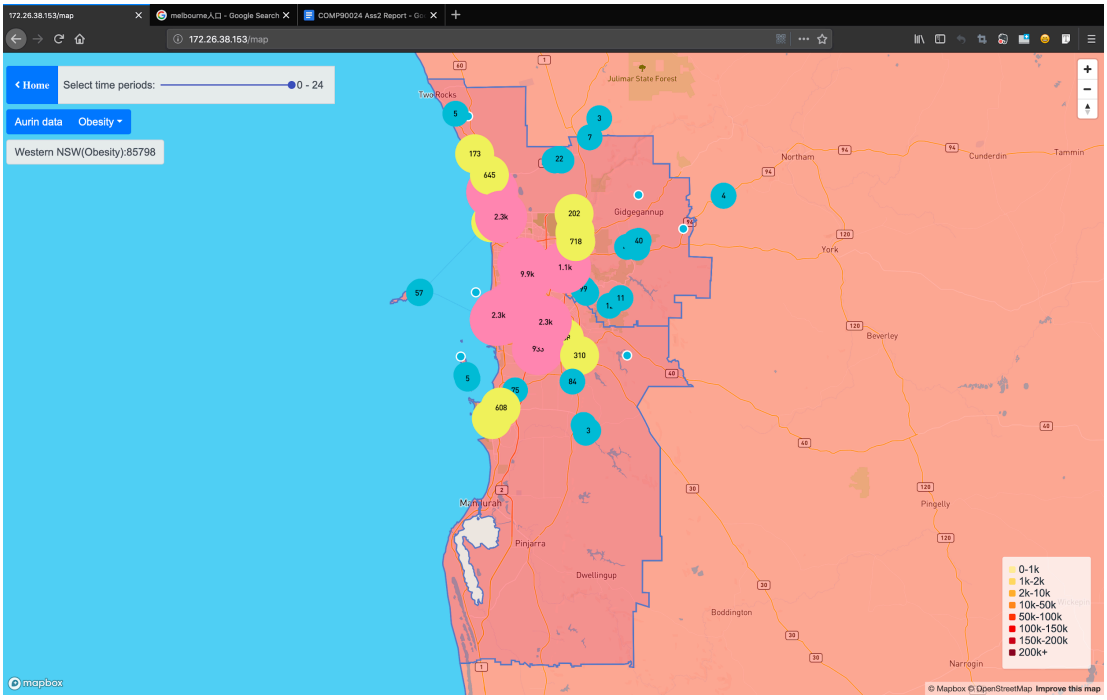
This opinion is relatively obvious yet theoretically a foundation and argument for latter analysis(namely 5.2, 5.3 and 5.4).

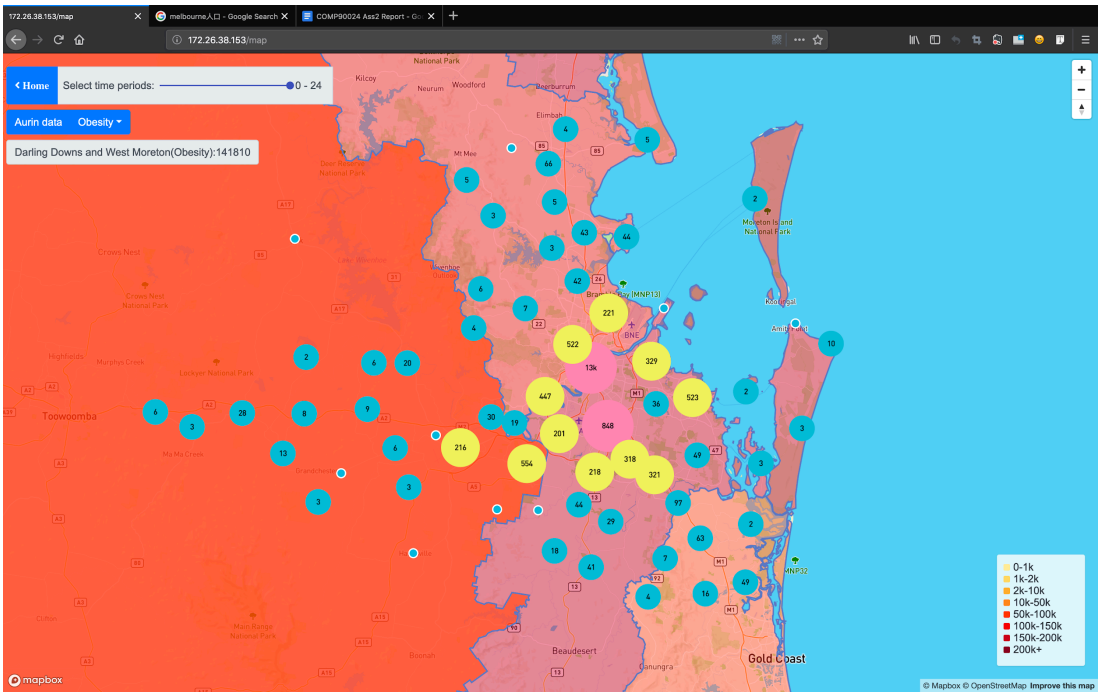
According to the statistic result, In 2014 Melbourne has the highest rate of 'gluttony' tweet among the four cities, while Sydney has the second highest. Perth is the third and Brisbane is the fourth. (picture here)

We have visualized the obesity population among the four cities.




Home
Select time periods:
0 - 24
Aurin data
Obesity
Western Sydney(Obesity):179837
0-1k
1k-2k
2k-10k
10k-50k
50k-100k
100k-150k
150k-200k
200k+
mapbox


Home
Select time periods:
0 - 24
Aurin data
Obesity
Western NSW(Obesity):85798
0-1k
1k-2k
2k-10k
10k-50k
50k-100k
100k-150k
150k-200k
200k+
mapbox

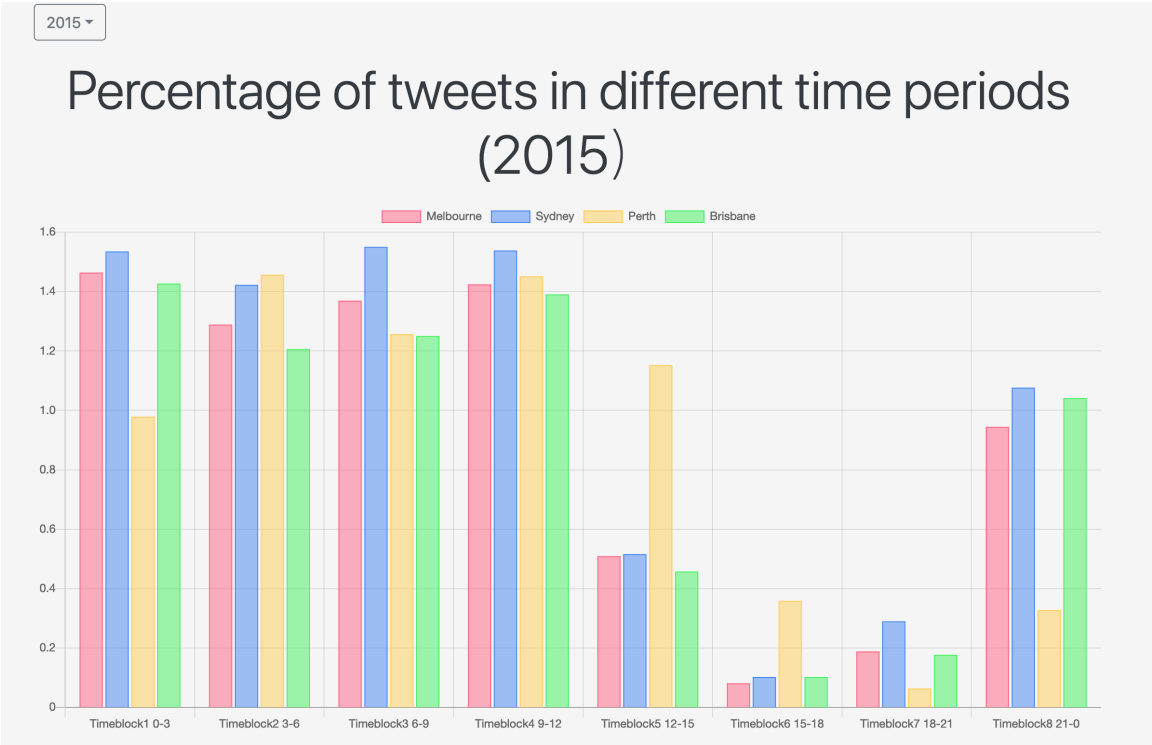


Along with the population of these four cities in 2014, we are able to calculate the obesity rate of these four cities.

City	*Melbourne*	*Sydney*	*Perth*	*Brisbane*
Obesity Rate	*19.3%*	*18.4%*	*16.4%*	*16.0%*

We now could find out the relative order of obesity rate is exactly the same to the the order of rate of 'gluttony' tweet, which confirms our guess. (And also suggest that our method to identify 'gluttony' tweet is reliable.)

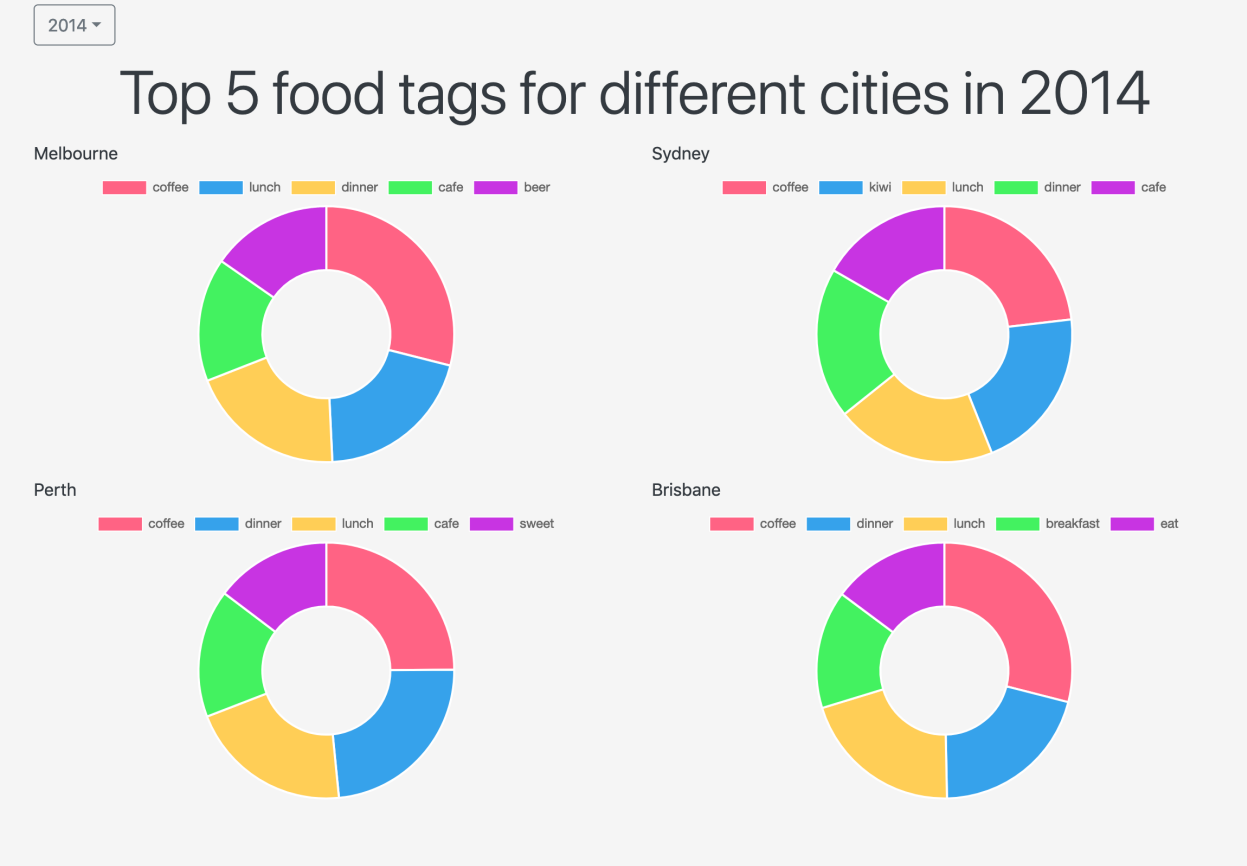
2. *Not having meals punctually will lead to obesity.*

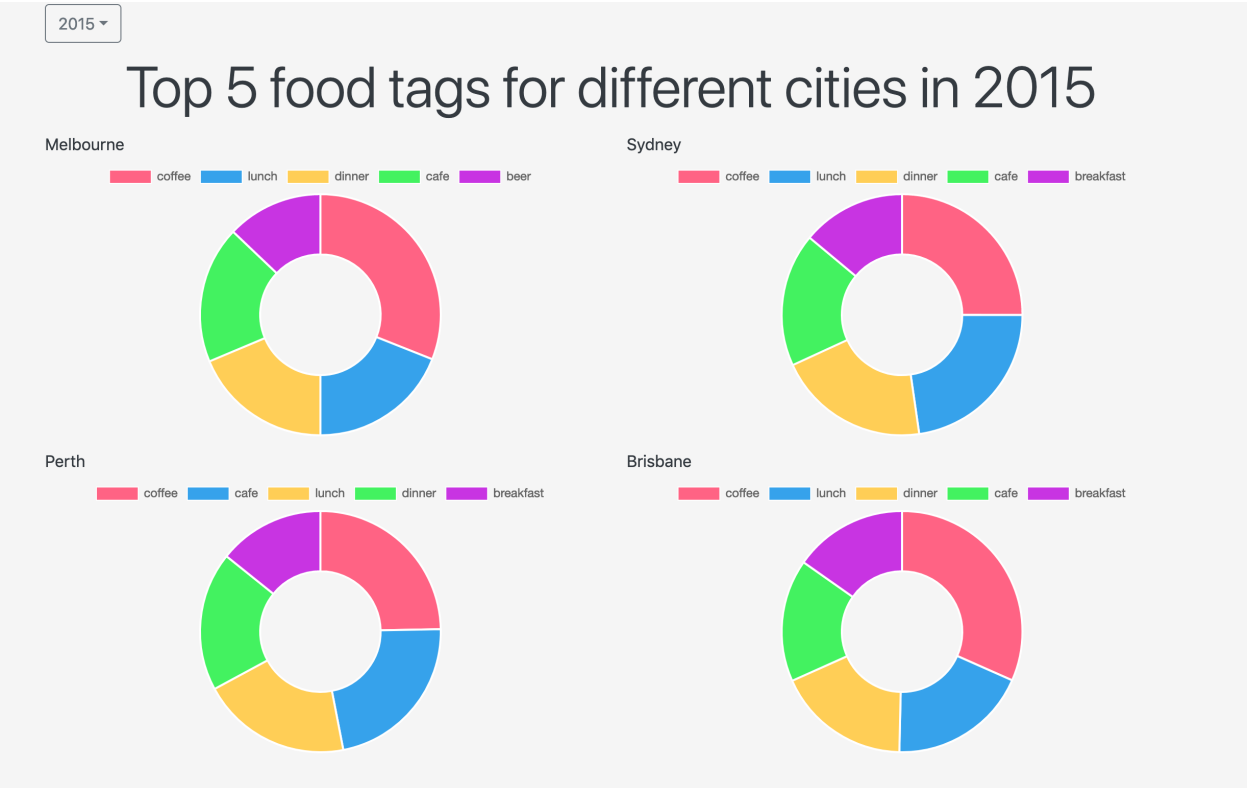


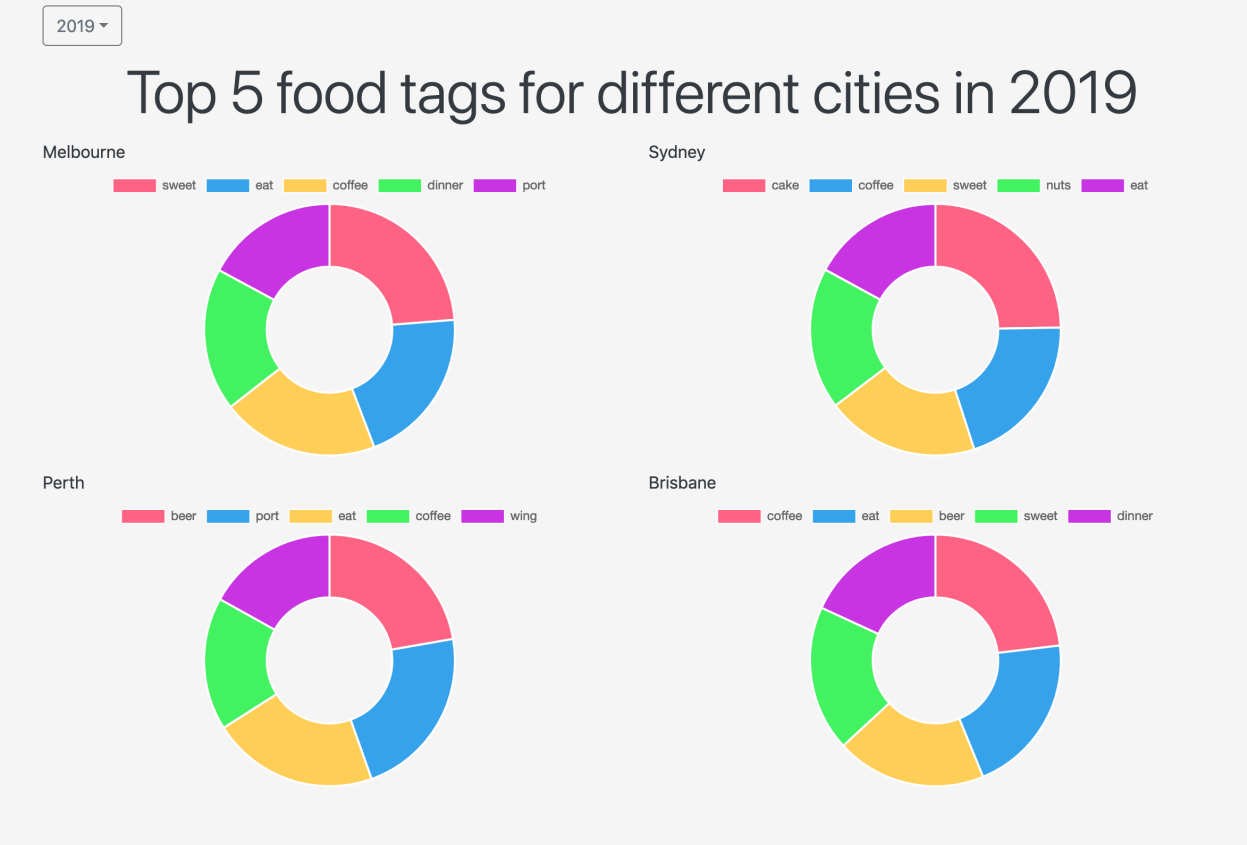
Punctually having meals is important for health keeping and avoiding gluttony.

We calculate the rate of 'gluttony' tweet in each time period among a day. Let's pay attention to the timeblock 21-0, as night snack is a common known factor of obesity. As can be seen in the graph, the order Melbourne>Sydney>Perth is same as obesity rate. However Brisbane is slightly higher than Perth. This may be because that Brisbane has more travelers in 2014, who enjoy Brisbane's night snack and tweet them, but bot be counted as local population in Brisbane.

3. *Popularity of coffee decreases over years .*







As shown in the graphs above, In 2014 and 2015, more than 30 percents of Melbourne's, Sydney's and Brisbane's tweets about food are relevant to coffee, while in 2019, this percentage decreases by 5 percentages.

Same things happens in Perth as well. In 2014 and 2015, coffee dominates ¼ of food tweets, yet in 2019, it is not even one of the top 5 tags.

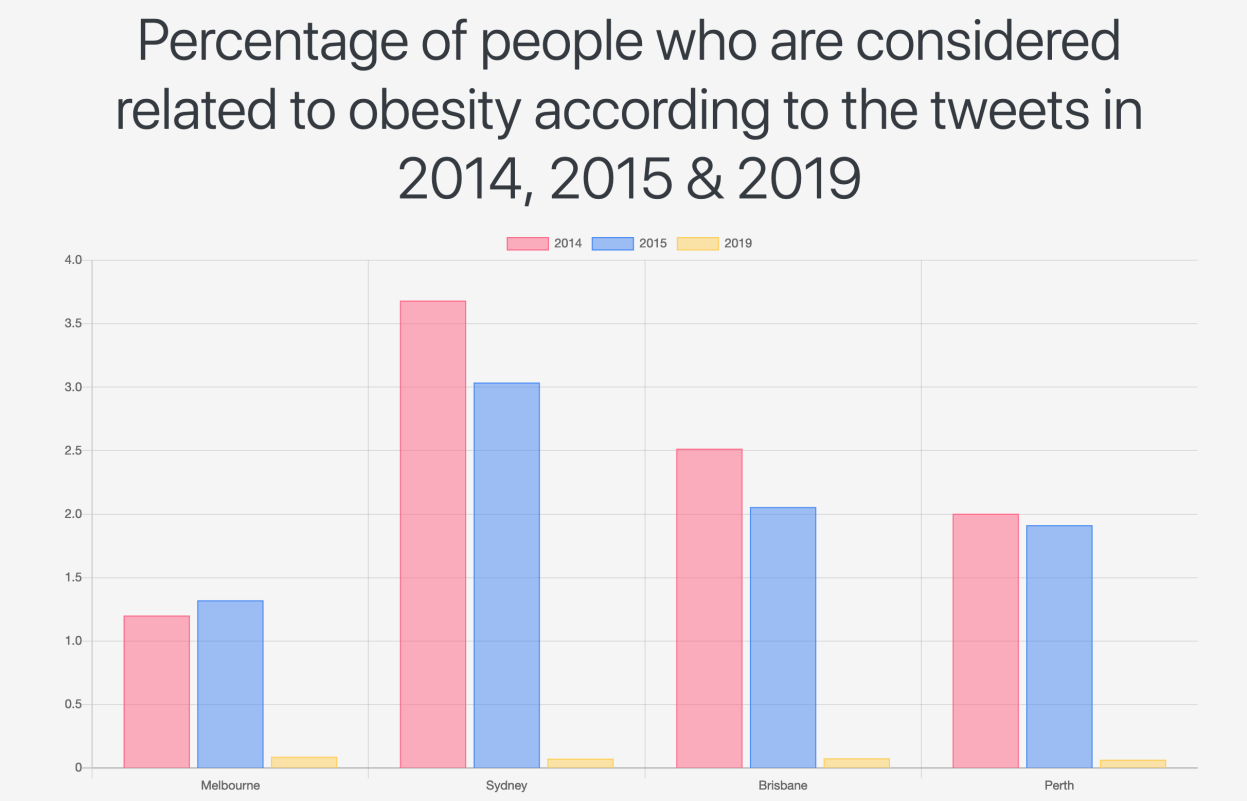
In all these four cities, coffee is not that popular as four or five years ago. In the meantime, beer has a trend of growing and occupies top 5 tags in more than one cites. This may be because the life rhythm of Australian has been accelerated, thus more people prefer drink like beer rather than enjoying a whole afternoon with a cup of coffee.

4. Predict obesity trend of year 2019

From AURIN platform we have retrieved information such as obesity rate in Sydney and Brisbane in 2014 and 2015, but not newest data of 2019.

	Sydney	*Brisbane*
2014	*171227*	*172726*
2015	*129530*	*128661*

In both cities, the obesity rate has decreased from 2014 to 2015. This trend confirms our conjecture, because rate of food-relevant tweets has decreased too, as shown in the graph below.



We could see that in 2019 the rate of food-relevant tweets has decreased even more. Therefore we could make a rough prediction that nowadays In the four main cities the obesity rate has decreased a lot.

6. Error handling

(1) Unstable instances on Unimelb Research cloud

The instances are unstable, namely the access of them may fail sometimes, which would threaten the safety of our data.

(2) removal of duplicate data

Our 4 tweet harvesters all search tweets from different city and different time period. It is impossible to get the same tweet in database. However, sometimes we may run the program more than once. In this case, checking whether a twitter is already in

the database becomes vital and necessary to prevent redundancy, which may slow down the process and waste storage space.

Each twitter harvested has a unique twitter id. Whenever an instance's crawler captures a twitter, it will check whether the twitter with the specific id is in CouchDB cluster already. If it is already in the database, an exception will be thrown. Thus duplicate twitter will not be stored again.

We also consider retweet. However, we find that when a user quotes a tweet, the new tweet still has a unique id that is different from the original tweet. Therefore, we did not remove these quote. But in order to figure out which is retweeted, we save "is_quote_status" and "quoted_status".

(3) request rate limit

Twitter official API has a limitation of 180 Requests per 15 mins window when we use user authentication. When the limit reaches, there will be an exception. Then I catch the exception each time it is thrown and wait 15 minutes. However, it is not the most efficient way because sometimes we only need to wait 10 minutes while we have already spent 5 minutes to harvest. After that, I found that there is a parameter called "wait_on_rate_limit" which can control the rest time to wait automatically when it is set as "True". In reality, we harvested around 2500 tweets/15 minutes. It is much faster than stream API already. Then we found a faster way than this Access Token Auth method. That is App Only Auth method, which can reach the speed of 6000 tweets/15 minutes (2.5 times as previous one).

(4) keywords matching problem

To match keywords in tweet text, it will be inaccurate when we simply search the words in text. In our collection of food-related keywords, some keywords like "tea" will match "teacher" which is not related to food. This situation can be solved by splitting the whole text. However, new problem occurs after splitting. Some combined words like "hot dog" will not match. Also, there is some symbol like "#", "!", "?", "@" may appear at the beginning or end of the words. Thus, we come up with an idea to cope these situations. Before that, Some regular preprocessing like "lower()" is obviously applied. First, replace all non-alphabetic and non-numeric character as space (" "). Second, add space (" ") at head and end of the text and keywords. Finally, match them directly.

For example:

"apple" => " apple "

"I love #apple! and #banana" => " i love apple and banana "

Match " apple " in " i love apple and banana "

Our Github clone address: https://github.com/DefangS/CCC-ass2.git (We have already invited Professor Sinnott as contributor of this private repository. If more invitation is needed, please feel free to contact defangs@student.unimelb.edu.au)

Demo video: https://www.youtube.com/watch?v=SaF67tXHJNI&feature=youtu.be

7 Data sources

1. *Data from AURIN platform https://aurin.org.au/*
2. *Twitter API https://developer.twitter.com/en/docs.html*

3. 2014-2015 historical tweets from provided database in "Appendix – Access to Twitter and/or Instagram Data" - http://45.113.232.90/couchdbro/twitter/_design/twitter/_view/summary