COMP90041 CLUSTER AND CLOUD COMPUTING

ASSIGNMENT 2

HOUSING PRICE ANALYSIS THROUGH TWEETS

GROUP 41- Melbourne

Akriti Sharma (1037120)

Oliver Costa (320420)

Raelene Huang (694488)

Tiffany Geng (695035)

Zhichao Zhong (1016516)

Table of Contents

1. Introduction

This report details how we leveraged the UniMelb Research Cloud to explore the evolving relationship between social media use and housing prices in Australia’s major cities. Housing prices in Australia, particularly those in Melbourne and Sydney are often a hot topic of social and political discourse. The landscape regarding domestic investment policy whether it be negative gearing or the first homeowners grant, as well as constant changes in immigration and overseas investment policy means housing prices are frequently in the Australian news cycle. This phenomenon was no more apparent than during the COVID-19 pandemic the impact on residential real estate was a mainstay in many publications both print and online. As the expansion of our major cities continues, fewer individuals can attain the coveted goal of homeownership and thus the discussion regarding homeownership and house prices in our cities continues at an increasing rate, with higher stakes and lasting social and political impact. It made sense then, to further explore the mechanisms behind Australia’s relentless rise in housing prices given the importance of the topic to Australian life and its continued impact on the future.

Taking advantage of the availability of both real-time and historical Twitter data, combined with data available from the Australian Urban Research Infrastructure Network (AURIN) platform, we will use the UniMelb Research Cloud to analyze the differences in social media use between the inner Statistical Area Level 3 (SA3) regions in Melbourne. This analysis aims to determine how more affluent areas, those with higher average median house prices interact on social media as opposed to those with a lower average median house price. This analysis will not only examine the current differences in the use of social media but will use historical social media data to examine the change in use over time when compared to the median house prices, thereby yielding predictive power in observing house price trends into the future. Ultimately, we seek to better understand the differences in social media use across varying socioeconomic regions to understand house price movements. This report will focus on 13 key inner SA3s in Melbourne, encompassing the CBD and surrounding northern, eastern, and western suburbs.

To best utilize the findings of our analysis a web frontend will enable users to examine an overview of key terms used in Twitter, as well as the ability to research the distribution of any term of interest. This endeavor aims to allow users of our software the freedom to explore how the frequency of words and phrases used on Twitter correspond to housing price trends in the greater Melbourne SA3s. We hope to provide not only an economic insight into the current housing trends but also shed light on the influence certain demographics have on an area’s housing affordability. This report will detail how we have deployed the UniMelb Research Cloud with a detailed solution that the user may recreate with the attached code. The code enables readers to deploy a scalable solution that may be run across any node of the UniMelb Research Cloud to harvest and store tweets while also scaling up and down as required to meet demand.

2. System Functionalities

2.1 System Overview

To achieve the goal of analyzing twitter data among SA3 regions we will be dividing the task into two main components, the backend infrastructure, and the frontend infrastructure. The frontend will be comprised of a responsive react.js application that fetches processed data from our database. The backend will be comprised of four virtual machines hosted in the UniMelb research Cloud. Of these virtual machines, three are used as database servers and one is operated as our webserver. The database nodes are connected to a cluster that pushes analytical results to our webserver via our API.

2.2 Data Collection

From Twitter, we harvested tweets for May 2020 based on geographic locations of individual SA3 regions around VIC by manually supplying longitude and latitude. We are also fortunate to collect tweets from the second half of 2014 from Richard. These tweets were also separated based on geographic locations of each SA3 location around VIC by radius, longitude, and latitude.

From the AURIN platform, we chose Timeseries property data (SA3 - ASGS) as our base data to compare with data harvested from Twitter. Since we are comparing housing prices, we chose *Average House Price* (going forward house price) of the SA3 region as our feature, and for a valid comparison with Twitter data, we only use house prices from the ending months of 2014 and starting months of 2020.

2.3 Data Processing

To look at the topic of each tweet content, text pre-processing needed to be done. Twitter users typically post a variety of contents, which include text, emojis, URLs, mentions, and hashtags. Since emojis, URLs and mentions cannot provide useful information, these were removed as a first step. The second step was to remove stop words from the text - the common words in English that are not useful in determining the underlying meaning of the text such as “him”, “be”. The third step was to stem the words to the root word. All the aforementioned steps were handled by python natural language processing packages *nltk* and *re*.[1]

2.3.1 Gains Ratio

To tackle the task of identifying the top influential topics, we created a feature vector that listed all unique topics, for each tweet. If the tweet contains a particular topic, the topic in the feature vector will record 1, else the element will be 0. An average housing price was assigned to each tweet, according to the SA3 region this tweet came from.

[1]https://pypi.org/project/nltk/

https://docs.python.org/3/library/re.html

We aim to find the most influential terms that would provide information about the housing prices of each location, hence gain ratio comes into play. The gain ratio looks at the correlation between each topic attribute and the class (average housing price). The information gain is equal to the total entropy for an attribute if for each of the attribute values a unique classification can be made for the result attribute. It then weights the information gain of each attribute by the total. The higher the gain ratio, the more information it provides about the class.[2]

The feature vectors and average housing price was constructed into a *pandas* data frame, and the gain ratio for each feature was calculated using the *info_gain* package.

There are a few limitations to this approach. Even though we have pre-processed the text, it still doesn't remove unless terms such as "the" or useless nones like "today" which doesn't inform anything about the content of the tweet. The terms picked by the gain ratio was then manually skimmed through, to allow for better analysis between the term and the reason it would correlate to average housing prices of each SA3 region.

2.3.2 Term Frequency Visualization

Our main idea in the front end was to visualize the relationship between a word/topic and the average housing price in each SA3 region. The most straightforward way to visualize this is to plot a bar graph between each region/average housing price against the document frequency of the query topic/term. Document frequency is calculated by the number of tweets the term appears in over the number of total tweets.

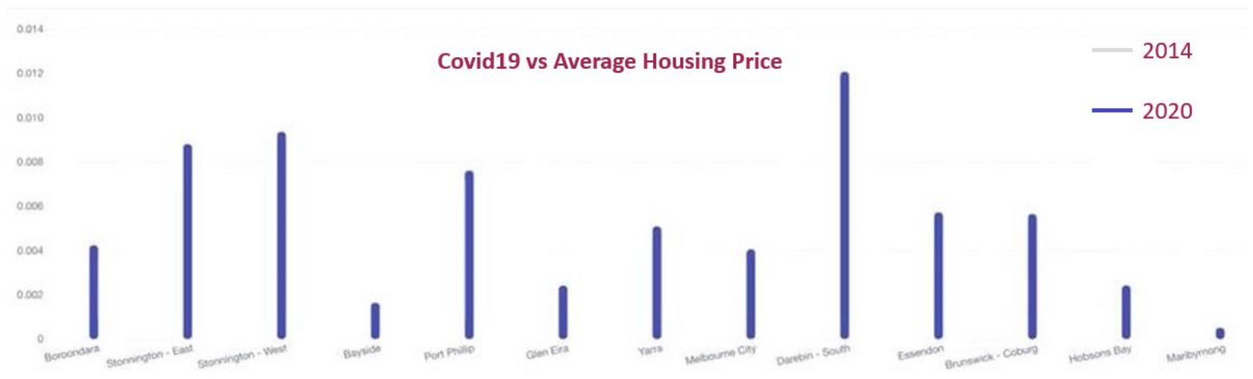
2.4 Scenario Analysis

In the following scenarios, the x-axis denotes average housing prices range from highest to lowest as we go from left to right. The y-axis denotes the document frequency of the term.

Scenario 1: COVID19

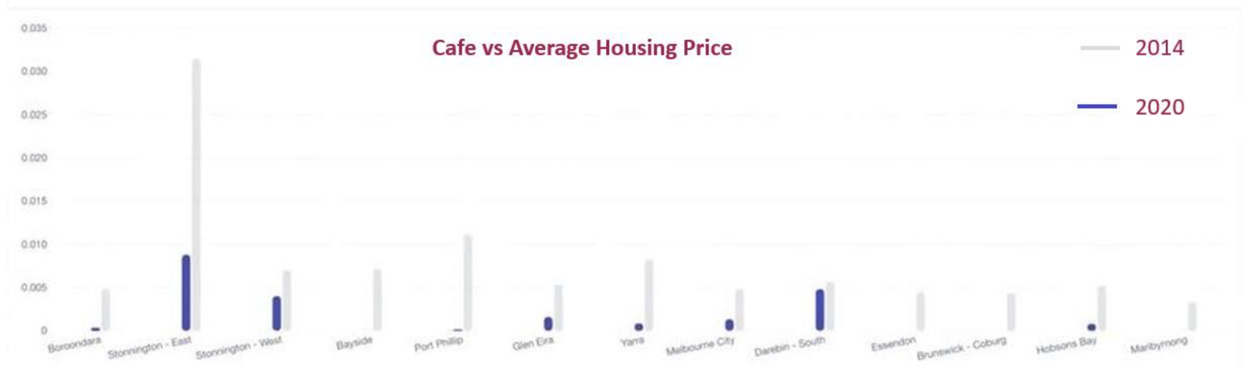
Consider the graph which shows a correlation between *COVID19* and the average house price of an SA3 region. As we can see there is no tweet with the term *COVID19* in 2014 because the term simply did not exist in that year. For the year 2020, the SA3 region with the high average housing price the frequency of the term is also high as compared to those with low average housing prices since people living in richer SA3 region tends to travel more for business or vacations and are more concerned for their safety and well-being from the virus as they are more likely to come in contact with it. The exception to this is *Darebin-South*, where the frequency is highest a possible reason can be because most of the native Asians live in this region and since the virus originated from an Asian country, COVID19 is more likely to be the topic of discussion.

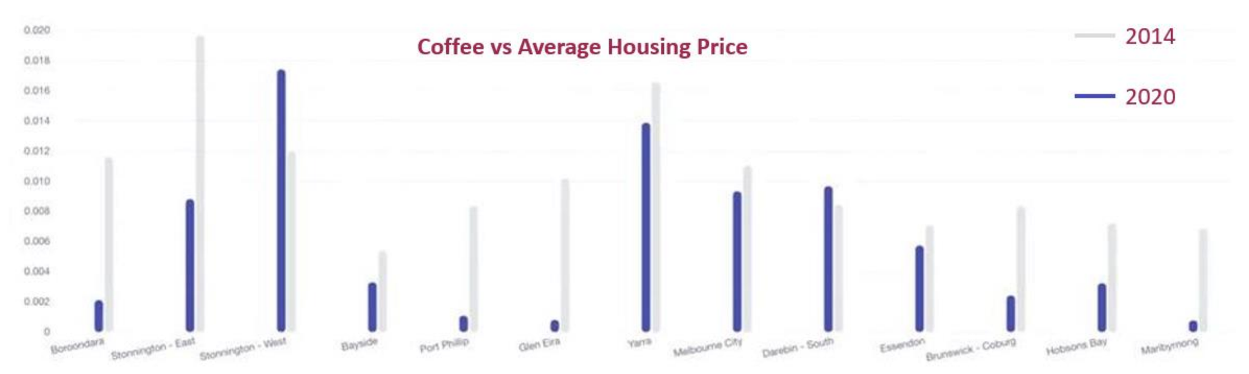
[2]https://www.mitre.org/sites/default/files/pdf/harris_biases.pdf



Scenario 2: COFFEE VERSUS CAFE

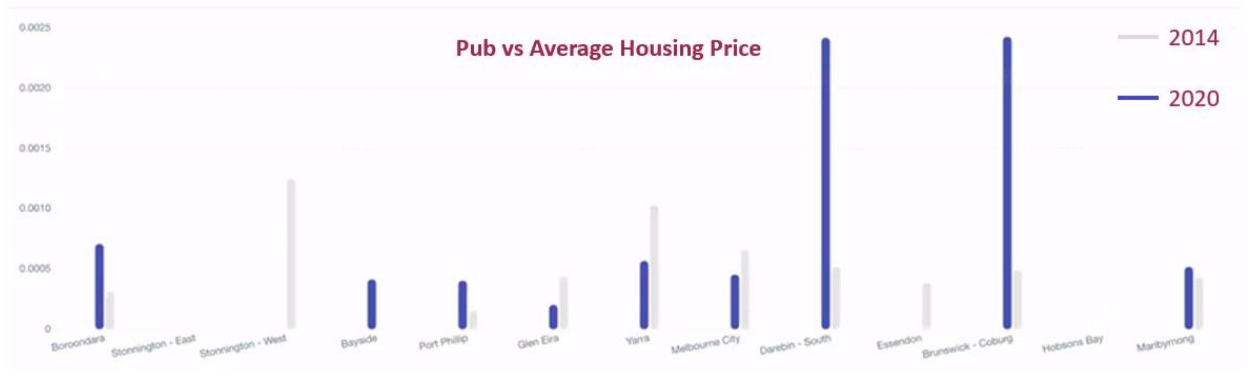
Let us look at the two graphs below, where the frequency of terms *cafe* and *coffee* are compared with average housing prices. For the term *coffee*, it is equally used for tweets originating from a high-level to low-level housing prices because it is a popular beverage among people of all ages. The term *cafe* however is used only in high housing price regions even though it is just another term for the beverage. This can be because the term *cafe* is also often associated with food and comes under the category of luxury experience hence people living in richer regions are most likely to discuss it.





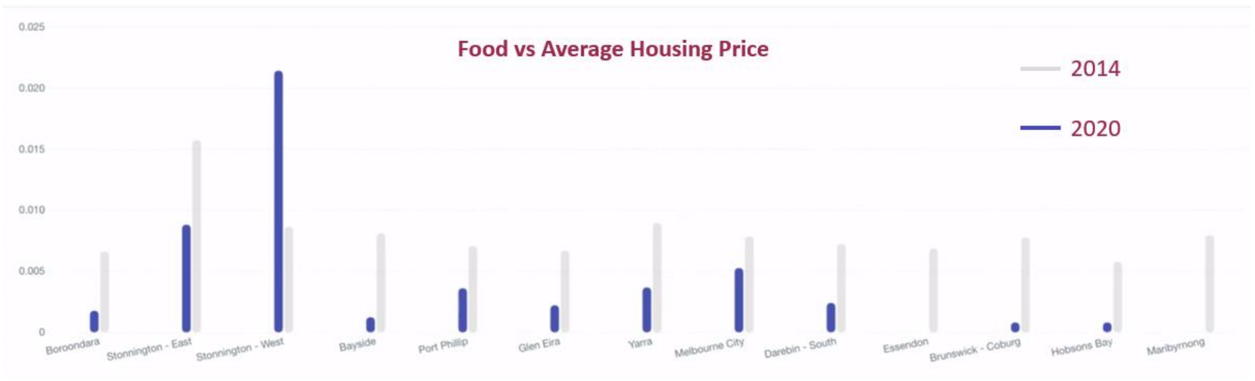
Scenario 3: PUB

Consider another scenario where we look at the frequency of term *pub* and find if there is any correlation with average housing price. The region with mid-level to low-level housing prices has a high number of tweets originating with the highest frequency in *Darebin-South* and *Brunswick-Coburg* regions because the number of pubs in these is high and hence the discussion around this topic will also be more.

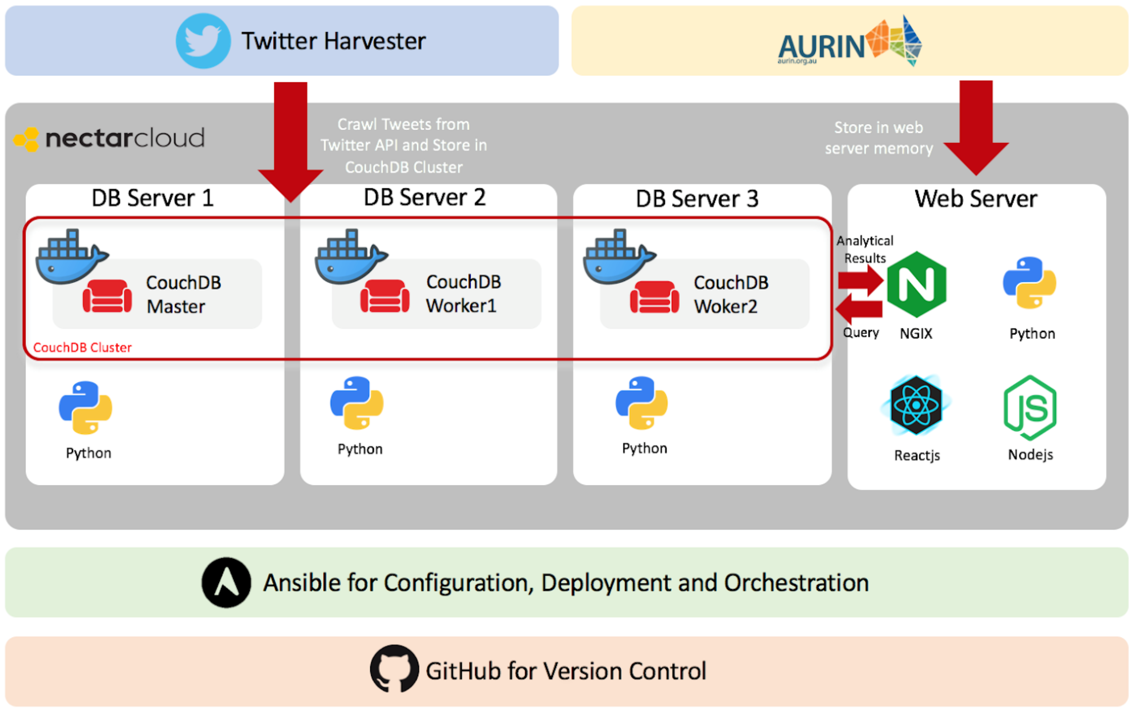


Scenario 4: FOOD

Consider another example of our scenario, where we look at the term *food* and the average housing price of an SA3 region. On average, the number of tweets with *food* in them was more in 2014 as compared to 2020. The obvious reason here is due to lockdown restrictions, the number of restaurants operating is less and hence there is less discussion around this topic. *Stonnington-West* has the highest number of tweets than any other region for the year 2020, even more than the number of tweets in 2014 from that region. This can be because people from the upper class can afford food from expensive restaurants. Another reason can be the biggest shopping center of Australia is in this region and hence it is intuitive that people will be discussing *food* in this region.



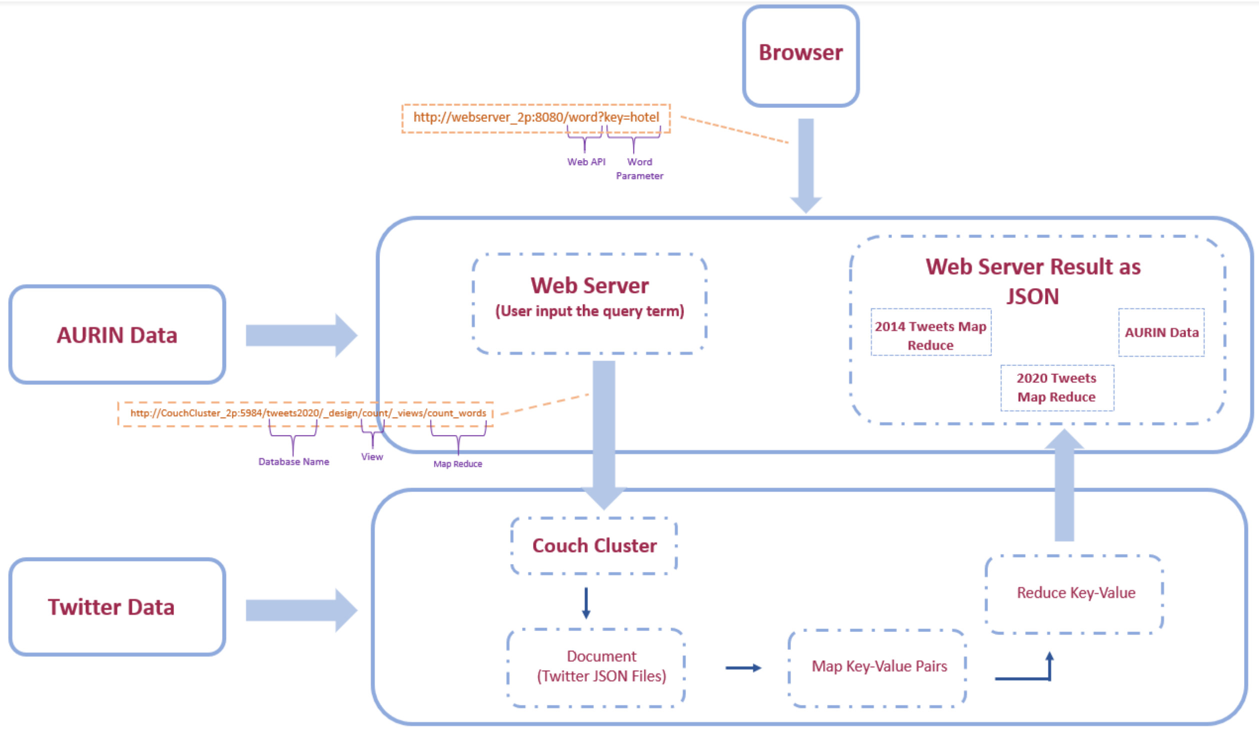
3. Architecture



The architecture provides a blueprint for the high-level logical components overlaid with technology and tools. This architecture has four medium-sized virtual machines (VMs) on the NeCTAR cloud platform, three of which are database servers and one operated as a web server. In each database server, a CouchDB node is configured in a Docker container for data storage, capture analysis, and transfer. To leverage CouchDB's shard replication features, the three nodes are connected into a cluster that pushes analytical results into the webserver. This web server contains Nginx, Python, Reactjs, and Nodejs to generate analytical results into front end visualization.

The architecture supports the automatic ingestion of semi-structured JSON documents into the CouchDB cluster from Twitter API through a python harvester script. AURIN data is extracted manually into JSON files and stored in the web server memory. Ansible is then used for system configuration management, deployment, and orchestration that can easily handle creating task dependencies and automation challenges. Additionally, GitHub is included for version control and collaboration within the development team.

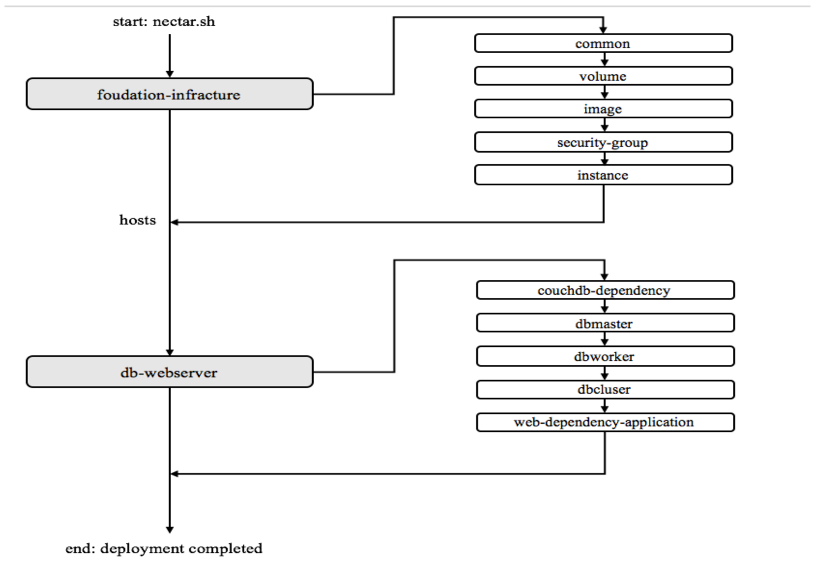
3.1 System


Browser
http://webserver_2p:8080/word?key=hotel
Web API
Word Parameter
AURIN Data
Web Server
(User input the query term)
Web Server Result as JSON
2014 Tweets Map Reduce
AURIN Data
2020 Tweets Map Reduce
http://CouchCluster_2p:5984/tweets2020/_design/count/_views/count_words
Database Name
View
Map Reduce
Twitter Data
Couch Cluster
Reduce Key-Value
Document (Twitter JSON Files)
Map Key-Value Pairs

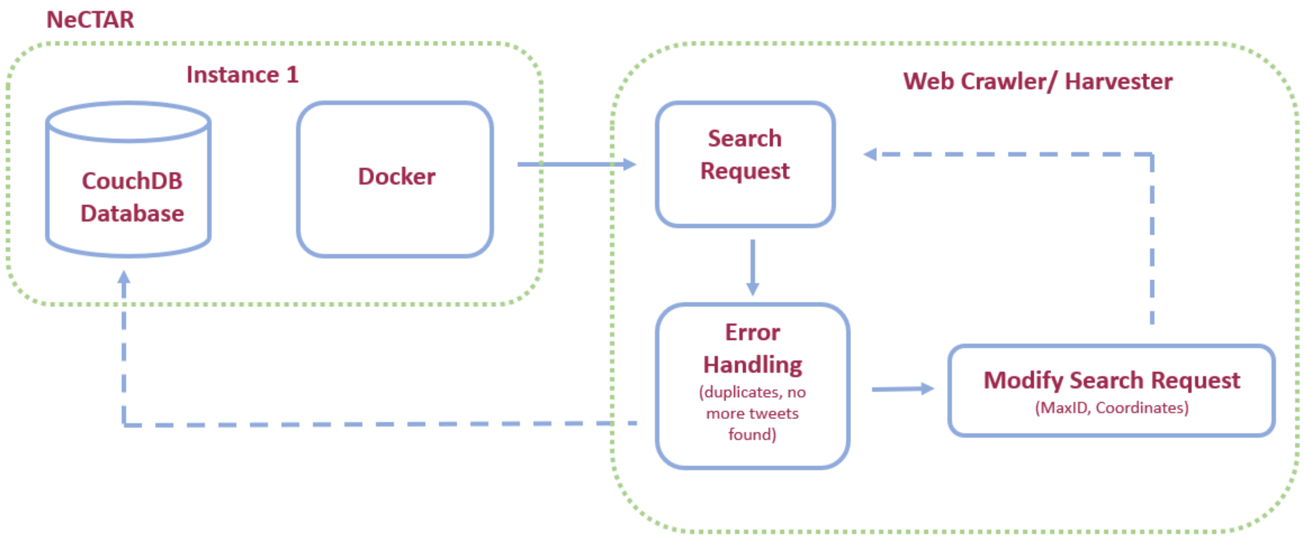
3.2 Ansible

Ansible is open-source configuration management, application deployment, and orchestration tool. In this project, we used the Ansible playbook to automatically deploy the entire system including basic infrastructure, software installations, and application configurations. As illustrated below, the process starts with creating four instances along with their security groups, additional volume, and other environment details. We then set up all dependencies and configurations for database servers and the web server by invoking the run nectar.sh file.

Since our system needs to scale and update rapidly in complex technology environments, automation is crucial for project management and system administration. By using Ansible, configurations are consistent across all instances and applications. If the system only has one instance and a limited amount of dependencies, it is possible to manage manually. However, we have multiple virtual machines with completely different dependencies on each of them. It is better to use an automation tool and here Ansible provides a simple solution for configuration management. Additionally, Ansible playbook performs all functions easily over SSH and it does not require any agent installation.



3.3 Twitter Harvester/API



3.3.1 Twitter API Endpoints

The python *tweepy* package was used to authenticate Twitter developer access and connect to the Twitter API. There are several key Twitter API endpoints available for gathering tweets, however only one allows for geo-location search, which is the ***Search API*** - searches for tweets within the last 7 days per request.

The ***Search API*** endpoint also allows for empty query terms. Since our topic of interest is to find correlations between tweets and average housing prices, we are not searching for any specific terms in the tweet queries. It is ideal to gather as many tweets as possible.

3.3.2 Geographical Limitations

The purpose of this analysis is to evaluate tweets via geolocations, so the tweet harvester must be able to identify this. We used the *search* API endpoint to request tweets, because this specific method allows coordinates (longitude and latitude) and a radius from the specified coordinate parameters, so the tweets returned came from the specified radius.

The specific coordinates of each SA3 location and radius were visualized manually from AURIN and passed as a parameter in the *search* request to Twitter API in separate iterations. As most SA3 regions do not resemble an exact circle, tweets from these coordinate and radius requests may not represent each SA3 region exactly, rather it is an approximate region.

3.3.3 Removal of Duplicates

Since each *search* request only returns 100 (most recent) tweets, we utilized the *MaxID* parameter and set it as the smallest tweet unique identifier from the last call, so that the next request would ask for the next 100 tweets posted earlier. In this case, there will be no duplicates between each request.

Besides, to make sure there are no duplicates among each location and requests from separate days, both tweet unique identifiers and abbreviations of SA3 locations have been used as the document key for unique storage in the CouchDB database. Hence the harvester will only store each tweet once. If the tweet already exists in the database, the CouchDB API will return a 409-request status, which is handled by exception and the tweet will not be stored again in the database.

3.3.4 Rate Limits

Twitter API enforces a rate limit that only allows a maximum number of 180 requests per 15-minute window. We implemented two approaches to overcome this issue and maximize our harvesting efficiency. First, we requested multiple Twitter Developer access accounts (one from each member). Second, letting the harvester sleep when the rate limit has reached, and restart when the time has passed. This method can be automatically set in the *tweepy.API()* method.

Since we are interested in specific SA3 regions around Victoria only, it turns out that there are not many tweets around Melbourne Victoria in May 2020. So, with the sleep on rate limit implementation, one Twitter Developer access account is sufficient for the harvester.

3.3.5 Error Handling

Apart from the rate limit issue, which was handled by tweepy, we encountered two other errors that needed to be handled. First, was to prevent putting duplicate tweets in the database. The 409-status error was handled by an exception as mentioned before.

The second concern was to handle the search request returning an empty JSON file, which means there are no more tweets available in this search. This was handled by an \`if\` statement to check the file before pushing it to the database. Other errors from *tweepy* search requests (tweepy.TweepError) were handled with the exception at the end of each request, but this type of error was rarely encountered.

3.3.6 Web Crawler Design

The harvester was written in a python script but executed and set up through docker. It runs on one instance only since we only require one Twitter developer access account. The proxy server was set up to match the proxy of the instance.

The harvester script was set to run once every day (i.e. 23 hours). The script will send search requests in loops, until it reaches one of the two exit conditions. First, when it encounters “no more tweets” and exits through an exception. Second, it will encounter the 409-request status when uploading to the CouchDB database. This is a key error that means the tweet has already been stored in the database. Once this error is encountered for all SA3 regions, it means all new tweets have been stored successfully, and it is starting to loop through the old tweets already stored. In this case, the harvester will exit.

3.3.7 Search Space Limitations

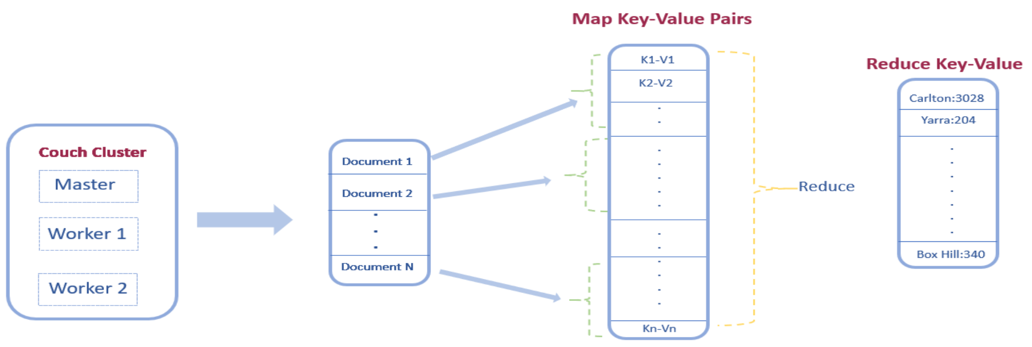
Our initial idea was to look at the correlation between tweets and housing prices over time, which requests tweets over several years. Eventually leading to a prediction of overvalued or undervalued areas. However, due to the search space limitation of the Twitter API endpoint, we weren’t able to get tweets over the years, rather than only for May 2020. This shifted our analysis to pouring looking at the correlation between the content of tweets and the average housing prices in each SA3 region. In addition to tweets collected from the harvester, we also collected tweets provided by Richard over the second half-year of 2014. This provides a contrast of analysis between 2020 tweets and 2014 tweets.

3.4 AURIN Dataset

Time Series property data (SA3 - ASGS) is manually retrieved as a .JSON file from the Australian Urban Intelligence Network (AURIN) portal and directly stored in the webserver. This file contains both Victoria property prices and their geolocation and provides an additional layer of information for scenario analysis.

3.5 CouchDB

CouchDB is an open-source document-oriented NoSQL database that provides the capability to store large semi-structured datasets and utilize MapReduce function to query views in a multi-node environment.



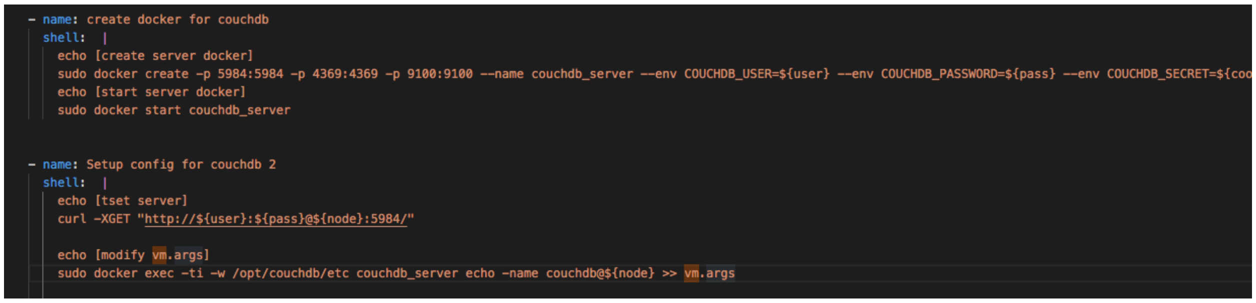
3.5.1 Installation

In the beginning, all relevant dependencies for CouchDB such as NodeJs, QGIS, jq, and npm are installed and executed in three database servers using Ansible. Additionally, Docker containers that bundle software and its configurations into a standardized unit are also used for CouchDB setups. By doing this, the system uses far less memory than virtual machines and the system is more portable and reliable.

3.5.2 Configuration

To create docker containers from their images, we need to edit the configuration file to update the proper management and instance proxy first.
Steps are as follows :

1. Create the admin account
2. Seed default Cookie secret
3. Name the CouchDB name with the instance IP, which will be the proxy of CouchDB container
4. Bind the ports from instance to docker:
    - 5984: Standard cluster port for HTTP API
    - 4369: Erlang port mapper daemon
    - 9100: Cluster nodes communication
5. Update the argument file "/opt/couchdb/etc/vm.args" with "-name couchdb@instance_IP_address"



```
- name: create docker for couchdb
  shell: |
    echo [create server docker]
    sudo docker create -p 5984:5984 -p 4369:4369 -p 9100:9100 --name couchdb_server --env COUCHDB_USER=${user} --env COUCHDB_PASSWORD=${pass} --env COUCHDB_SECRET=${coo
    echo [start server docker]
    sudo docker start couchdb_server


- name: Setup config for couchdb 2
  shell: |
    echo [tset server]
    curl -XGET "http://${user}:${pass}@${node}:5984/"

    echo [modify vm.args]
    sudo docker exec -ti -w /opt/couchdb/etc couchdb_server echo -name couchdb@${node} >> vm.args
```

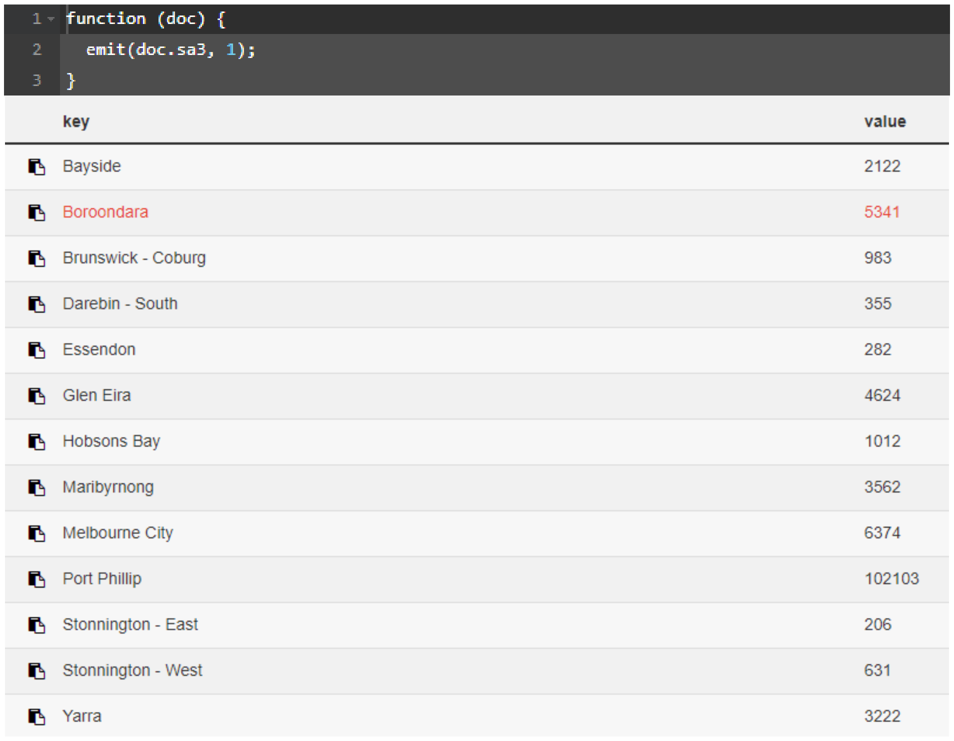
3.5.3 Cluster Setup

Upon completion of CouchDB dependencies and node configurations, all three nodes including one master node and two worker nodes are connected into a CouchDB cluster through ports, meaning all three nodes can communicate with each other. By using CouchDB in a clustered mode, it supports both horizontal partitioning and replication to create an easily managed solution for balancing both read and write loads. In our system, each database has two shards, three replicas for each shard, and a total of six shard files. Those files are horizontally scaled into three CouchDB nodes. In principle, the more copies of a shard, the more we can scale out. This design is very durable and reliable. As long as there is at least one copy of every shard that is still online, all documents can be read and written.

3.5.4 Databases, Views, and MapReduce Function

In the database, each JSON file only contains only one tweet. Two databases are initialized to meet the analytical requirements of this study. ‘tweets2014’ and ‘tweets2020’ store JSON files from 2014 and 2020 within relevant SA3 locations. For each database, there are two map-reduce views to count the number of documents and the number of words, respectively. Since we do not update the document contents, the key-value pair results from the map function will be saved and reused. When we call the reduce function, only the updated tweets will be mapped into key-value pairs, which means only the reduce function uses the old data. This strategy optimizes the processing speed and releases more traffic.

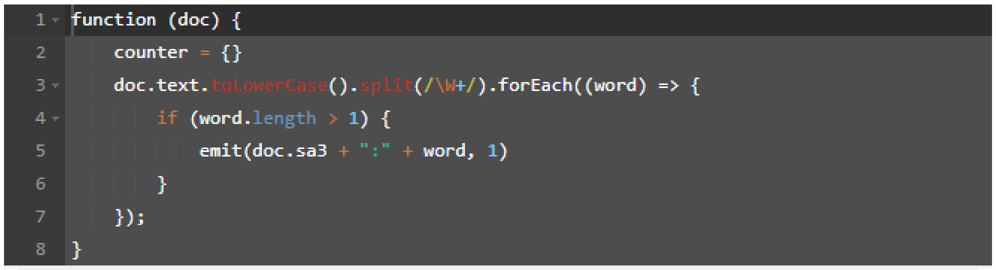
Count-doc view shows the number of documents for each SA3 location (e.g. "melbourne_city": 6031). Map function returns SA3 key and document count (“sa3:1”). The reduce function corresponding to this provided by CouchDB is _count function. This view can be found by HTTP request with URI: “http://node-IP:5984/db-name/_design/count/_view/count-doc”.



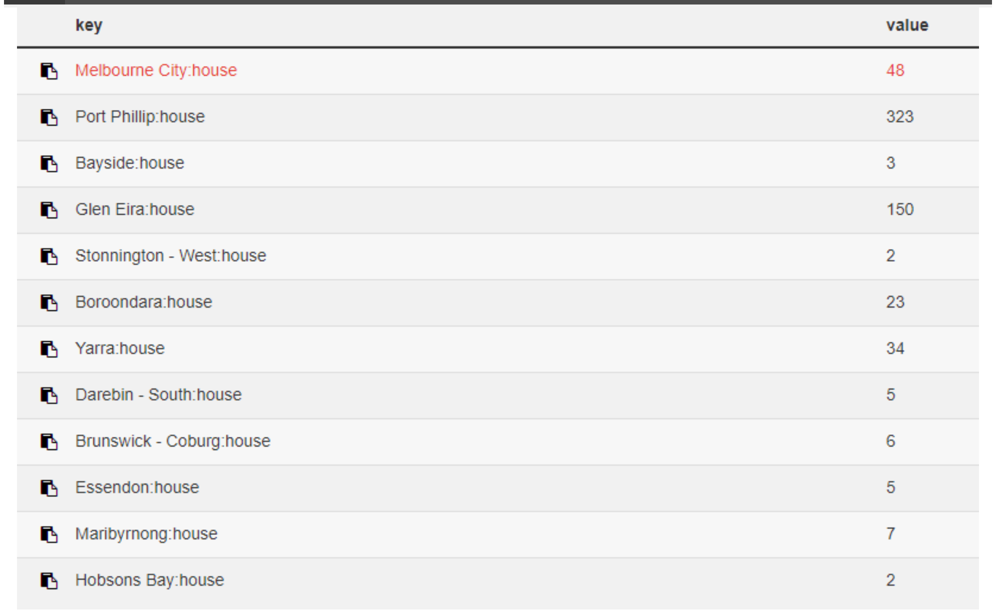
```
function (doc) {
  emit(doc.sa3, 1);
}
```

key	value
Bayside	2122
Boroondara	5341
Brunswick - Coburg	983
Darebin - South	355
Essendon	282
Glen Eira	4624
Hobsons Bay	1012
Maribyrnong	3562
Melbourne City	6374
Port Phillip	102103
Stonnington - East	206
Stonnington - West	631
Yarra	3222

Word-doc view shows the number of words for each SA3 location (e.g. "melbourne_city:you" : 204). Map function returns SA3 key with keyword and word count. The reduce function corresponding to this provided by CouchDB is _sum function. This view can be checked by an HTTP request with URI: “http://node-IP:5984/db-name/_design/count/_view/count-word?key=searching-word”.



```
function (doc) {
    counter = {}
    doc.text.toLowerCase().split(/\W+/).forEach((word) => {
        if (word.length > 1) {
            emit(doc.sa3 + ":" + word, 1)
        }
    });
}
```



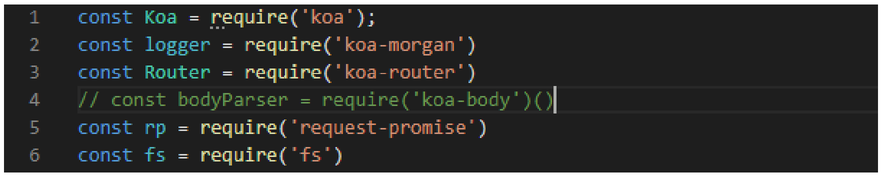
key	value
Melbourne City:house	48
Port Phillip:house	323
Bayside:house	3
Glen Eira:house	150
Stonnington - West:house	2
Boroondara:house	23
Yarra:house	34
Darebin - South:house	5
Brunswick - Coburg:house	6
Essendon:house	5
Maribyrnong:house	7
Hobsons Bay:house	2

3.6 Back-End

We implemented a NodeJs server to collect data from CouchDB and a responsive front end with an analysis of twitter data. This structure simplifies the client query format and avoids direct interaction between users and the database.

3.6.1 Environment Setup

To build the node server properly, some dependencies are installed by npm. Also, another package pm2 is used to run this server in the background.

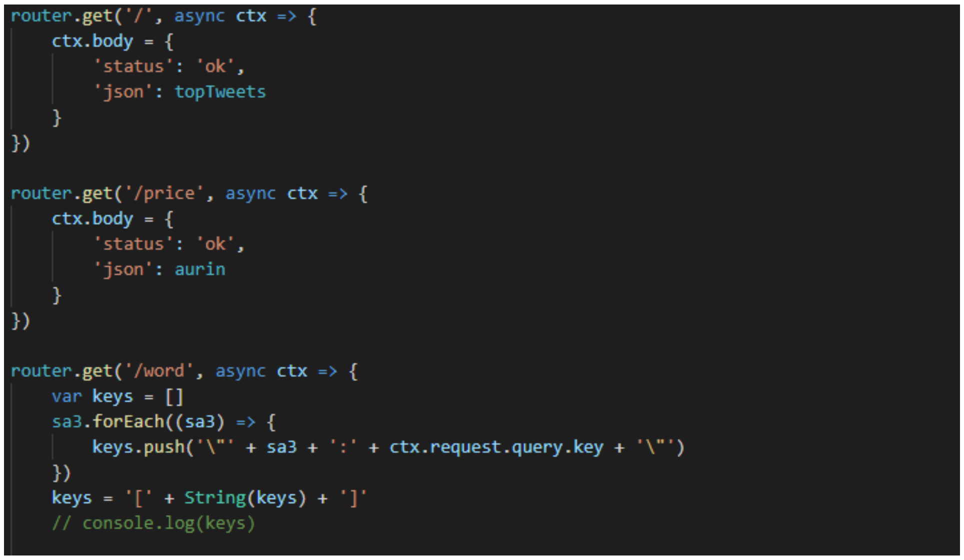


```
const Koa = require('koa');
const logger = require('koa-morgan')
const Router = require('koa-router')
// const bodyParser = require('koa-body')()
const rp = require('request-promise')
const fs = require('fs')
```

3.6.2 Route

This web server provides three APIs, including words with high information gain ratios, house price rank, and commonly searched words. These APIs respond to user queries and then send twitter analytical results in a .JSON file to the webserver.
URI is as follows:

- Words with high information gain ratios: http://server-ip:port/
- House price rank: http://server-ip:port/price
- Commonly searched words: http://server-ip:port/word?key=word

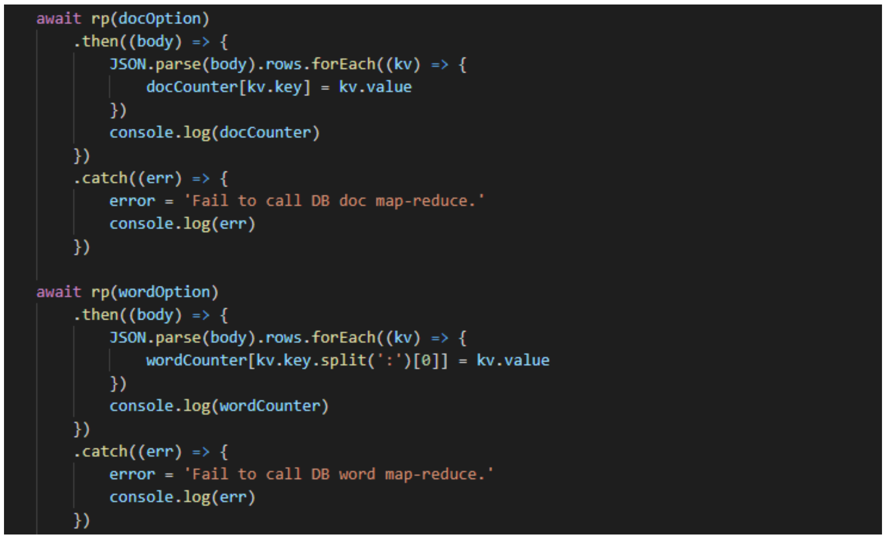


```
router.get('/', async ctx => {
    ctx.body = {
        'status': 'ok',
        'json': topTweets
    }
})

router.get('/price', async ctx => {
    ctx.body = {
        'status': 'ok',
        'json': aurin
    }
})

router.get('/word', async ctx => {
    var keys = []
    sa3.forEach((sa3) => {
        keys.push('\"' + sa3 + ':' + ctx.request.query.key + '\"')
    })
    keys = '[' + String(keys) + ']'
    // console.log(keys)
```

3.6.3 System details

To handle the concurrent requests, we implement a Koa app. This application attaches asynchronous functions, which takes charge of client requests with the smallest memory consumption.

As we need to query the database, the package ‘request-promise’ can request the CouchDB view API and return a Promise object. The keyword ‘await’ is used to wait until the promise is done. This is essential, otherwise, Javascript will skip this block. We use the ‘request-promise’ method to collect the reduced results from two databases.



```
await rp(docOption)
    .then((body) => {
        JSON.parse(body).rows.forEach((kv) => {
            docCounter[kv.key] = kv.value
        })
        console.log(docCounter)
    })
    .catch((err) => {
        error = 'Fail to call DB doc map-reduce.'
        console.log(err)
    })

await rp(wordOption)
    .then((body) => {
        JSON.parse(body).rows.forEach((kv) => {
            wordCounter[kv.key.split(':')[0]] = kv.value
        })
        console.log(wordCounter)
    })
    .catch((err) => {
        error = 'Fail to call DB word map-reduce.'
        console.log(err)
    })
```

3.7 Front-End Web Application

The front-end web application enables users to generate customized, real-time analysis on harvested tweets via a responsive React.js framework. Upon entering a term of the users' choice, the application will send a request to our couch cluster. The response from the cluster will be a JSON object which holds the values required to populate the graph plotting the weighted term-document frequency vs descending median house price. This graph is the central analysis tool on the application and will update whenever the JSON object holding its data is updated, this allows the user to enter numerous terms in succession without leaving the page.

The two tables at the bottom of the application both query our couch cluster on application launch and are populated with their respective JSON responses. The responses have been set to return the terms which best demonstrate segregation between areas of higher median house price and areas of lower median house price via the information gain metric. Results are displayed in these tables from our two datasets, both 2020 and 2014. While the 2014 response is unlikely to change as it is unlikely, we will add to this dataset, the 2020 dataset is being updated which means that its response will change over time. A new trend may appear on social media that better shows the differences in usage of higher-priced areas and lower-priced areas. If this discrepancy does appear, then the data in the 2020 table will reflect this new behavior pattern. The application itself has been developed to be responsive and mobile-friendly, with a grid structure that dynamically scales to suit the user's web browser. This is possible through utilizing a popular react friendly User Interface framework called Material UI where objects displayed in the application are treated as part of the overall grid of the application and displayed according to the resolution afforded to them on the client's browser.

3.8 Unimelb Research Cloud

This segment discusses the advantages and disadvantages of Unimelb Research Cloud.

3.8.1 Advantages

1. The platform provides cloud-computational tools for researchers, free of cost.
2. Support of volumes with snapshot alternatives is provided that can be easily removed or added to any virtual instance based on the requirement.
3. Setup is based on the OpenStack framework because of which commands to execute operations on the cloud are easily accessible using Ansible playbook in turn making deployment undemanding.
4. On-Demand computation and storage utilization feature provides ease of access and removes the unnecessary management of personal hardware as well as facilitating different architectural complexities.
5. Customized templates can be uploaded by the user to deploy stacks which further helps in implementing entire infrastructure on the cloud using a single configured template.
6. Supports floating IPs which makes it possible to access the instances from outside and allows systematic access control through security groups and authentication procedures.

3.8.2 Disadvantages

1. Technical glitches like unresponsive web-interface create hindrance in task executions and hence fail to provide notifications about the task on time.
2. Immediate availability of new IPs for the instances is not provided which delays in completion of the overall task.
3. The possibility of data loss from the root disk creates uncertainty due to which it is difficult to use the 30GB storage provided by it.

4. User Guide

4.1 Installation Guide

This guide aims to provide users clear instructions to deploy the entire system through Ansible. Detailed steps are shown below:

1. Download the open rc.sh file from the NeCTAR Research Cloud dashboard and generate a new OpenStack API password for first-time users.
2. Create a key-pair on the NeCTAR Research Cloud and store private key .pem file in the local .ssh directory
3. Install Ansible on your local machine.
4. In the terminal, find the Ansible folder directory and trigger command **./run-nectar1.sh** to set up foundation infrastructure and then trigger command **./runn-ectar2.sh** for system deployment
5. sudo password that is generated from step 1, when the system prompt to enter OpenStack API password

4.2 Back-End Guide

1. Confirm the default port for NodeJs is port: 3000
2. In the node directory, the index.js file is the backend server loader. This file should run (node index.js args) with arguments: couchdb_username, couchdb_password, couchdb_IP_address, and the two database names.
3. The koa app server will take the preprocessed JSON files from aurin and tweet harvester.

4.3 Front-End UI Guide

UI capabilities and features:

- <u>SA3 region map</u>: this map visualizes the SA3 regions we are interested in for the tweet analysis. The average housing price (from AURIN) for each region is also displayed for visual clarity.
- <u>Search query</u>: users can search for a query term. It will search through the database and plot a bar graph showing the term-document frequency against SA3 regions, in descending order of average housing prices. This allows the user to visualize if there are any correlations between the usage of the query term and the average housing prices.
- <u>Highest gain ratio terms</u>: this provides an overview of the top terms returned by gain ratio analysis, which the user can put into the query and examine the correlations.

5. Improvements

While our system functions perfectly, there is plenty of room for improvement in the future. Our twitter harvester utilizes the search API endpoint and searches for tweets within the past 7 days in every run. There are some redundancies in this approach which is not the best method in the long term. Instead, we should use streaming API to process and record tweets in real-time as they are posted. This would provide a more efficient tweet crawling process.

When collecting tweets, our twitter harvester would almost always exit on the “no more tweets” condition. This could be worked around if we modified our interested regions to a larger scale, or tweaking the twitter search. A possible approach was to not only search for tweets from each SA3 region but to get tweets from the entire user timeline for these tweet users as well. It is more likely that users will tweet from the same location, and it would also provide a small scale of tweets that are more spread out across time.

As mentioned in previous sections, we are only able to access tweets within a small time range. This largely limits our analysis between tweets and housing prices. A better analysis approach is to look at the correlations between tweets and changes in housing prices over time if time permits.

6. Reference Links

6.1 Application UI:

http://172.26.130.208:8080/dashboard

Note: CORS extension required for some browsers, available below.

https://chrome.google.com/webstore/detail/allow-cors-access-control/lhobafahddgcelffkeicbaginigeejlf?hl=en

6.2 Github Repository:

https://github.com/aakritisharmaa/comp90024-assignment-2-group-41.git

6.3 YouTube Link:

https://www.youtube.com/watch?v=gpXPpurj9ec&feature=youtu.be