Australian City Analytics

Melbourne Migration Sentiment

Assignment II Report

Group 34

Github link: https://github.com/dazajuandaniel/twitterProject/

Video Link: https://youtu.be/5IfhEPdbToA

Website Link: http://115.146.93.162:5000/index

Ansible showcase: https://youtu.be/MFyQSehA3L0

Juan Daniel Daza – 846887
Junwen Zhang – 791773
Mingyan Wei – 280744
Sai Sree Vatsav – 905157
Wen Zhang - 769355

Contents

I. Introduction

Social media has opened the opportunity to researchers and developers to analyze user generated content. Twitter is one of these social networks which has data accessible to the public to analyze and process through various Application Program Interfaces (API). In this project, we aim to understand the demographics of suburbs who have either positive or negative sentiment towards migration topic by analyzing thousands of tweets generated across the Victoria region.

This is report covers the development of a cloud-based Twitter harvester and analyzer solution. Using Twitter's feed, we managed to harvest more than half a million tweets which we then processed using MAP/REDUCE in CouchDB. We tagged individual tweets and combined the data by leveraging Aurin's extensive datasets. We will cover the architecture chose as well as further development aspects.

Our main programming language used was Python due to its flexibility, power and easiness of use. We developed a basic front end application using flask and a combination of javascript, HTML and CSS as well as the Highchart libraries. In the backend, we set up a CouchDB database and took advantage of the MAP/REDUCE functionalities as well as the fact that it has an HTTP/JSON Api.

We automated the deployment of additional harvester servers in NeCTAR cloud by using Ansible and Boto and created mechanisms to tolerate failure at various levels.

II. System Functionalities

General Architecture

This system is composed by different parts. We have a harvester that utilizes both the Search API and the Stream API functionalities from Twitter. The information is then stored in a CouchDB database which is replicated to a different instance for security measures. In addition to this, we have a VM dedicated to performing analytics and to provide the web service for the instances.

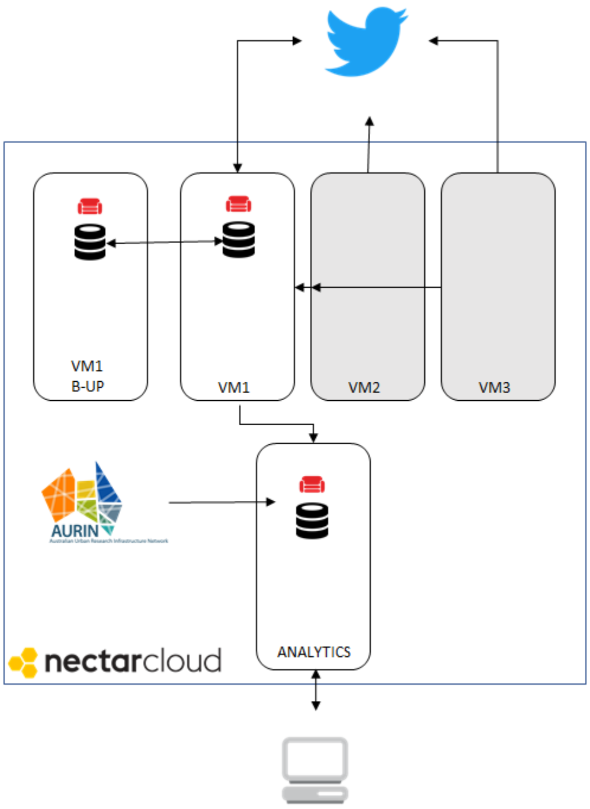


Figure 1. General Architecture

Tweet Harvester

Given the nature of the project, a large number of Tweets are required to be Harvested. To accomplish this, initially we set up three different instances to capture Tweets by using two different methods, later on, as the project deadline approached, we removed the additional instances. We leveraged the development of the Python library called Tweepy [1] this library provides most of the required tools to meet Twitter's rules and restrictions.

We created different applications in Twitter to generate multiple access codes and tokens to accelerate the capture of Tweets. This was a possibility due to the benefits of cloud computing where we could get a different IP Address for each virtual machine and capture tweets in parallel using various mechanisms

- **Search API:** Using this Twitter functionality, we were able to capture Tweets relating to a specific topic. We defined a list of keywords, which according to our personal experience and judgement, related to the following topics:
  a. Artificial Intelligence
  b. Food
  c. Politics

    d. Migration - Main aspect of interest

  In addition to this we set more parameters to try and filter and get more relevant tweets. The following parameters were adjusted to fit our needs:

  - **lang**: Only Tweets in English as our Sentiment Libraries are tuned for English language.
  - **geo**: We created a radio of 5000 miles around the Melbourne's CBD to capture relevant Tweets.

  The purpose of having more than one topic, was to have the required options to perform analysis. At first, we were unsure if we would have enough data about one specific topic so we decided to harvest multiple topics. To perform this correctly we were required to abide by the rules imposed by Twitter and take into account the general purpose of our project.

  There are API restrictions and limitations for Twitter, such as a limit of 18.000 tweets per 15 minutes or 100 calls every 15 minutes. Our system had rules in place to abide by these rules and make the required pauses when necessary. We had special counters and kept track of time elapsed since the last sleep timer to see if either more than 18.000 tweets had been collected or if more than 100 calls have been made within the 15 minutes. To make our system more fault tolerant and not quit the process when an error was found, we captured exceptions thrown by Tweepy and instead of quitting, we gave it a cooling off period of 15 minutes and restarted all variables as to reset the system. This proved to be efficient and kept the system running with minor interruptions and interactions from us.

- **Stream API:** To get a real-time feed of the Tweets we used the Stream API. With this API, we were able to get Tweets as soon as they occurred. To avoid capturing Tweets from areas which we were not interested such as countries different to Australia or cities such as Melbourne. We set different parameters in the API.

  - location: We created a grid that consisted of the most southeastern and northwestern point in Melbourne with coordinates:

    locations=[141.157913,-38.022041,146.255569,-36.412349]

  Since we do not have a particular topic with the Stream API we wrote a script that looks for a relevance of the tweet text with our topic of interest. Using the easiness of the Python language to handle *strings,* we wrote a script that looks for migration related keywords in the text and labels the tweet accordingly to perform queries easier and faster.

In addition to this, we used different libraries to label the sentiment of the Tweets before storing them in the Database. This will be covered in a future section of this report. Over the entire developing period, we managed to accumulate more than half a million tweets.

Duplicate Removal Process

Given the nature of the APIs provided by Tweeter many duplicates could be present. For instances, while searching for Tweets with the keyword "*migration*" and the keyword *"migration visa"* we had a high probability of getting duplicates. At first, we decided to capture all the tweets and then deal with the duplication removal process. However, after doing initial scripts to remove duplicates we found out that it was easier to validate for duplicates as the Tweets stream arrived.

To do this, we set up the unique identifier ID in our cloud instance of CouchDB to be the Tweet's ID. We did not allow CouchDB to generate a random ID but instead set the ID of the document to be stored in the database to be the unique identifier Twitter gives to its Tweets. If a duplicate was found, we skipped the addition of that particular tweet. This proved to be much more efficient than creating a view and the removing the duplicates using a script with Python. Nevertheless, since we had harvested tweets without the initial validation, we were required to remove duplicates from our database. To overcome this, we created a new database which included tweets without duplicates. To do this, we leveraged the power of the Couchdb Python Library [2]. We set up a script that read from the Raw Tweets database and inserted into the Clean Database. To speed up this process we created a view in the Raw Tweets database that mapped the duplicates tweets thus reducing the amount of computation Python required to do.

CouchDB Configuration

CouchDB is a NoSQL database design for reliability [3]. It has many advantages and benefits which is the main driver behind using it for this project. It speaks natively JSON and supports MAP/REDUCE natively through its views functionality. The initial setup and configuration is straightforward.

- **Accessible:** We provided open access to the database by changing the *bind* parameter to access the database remotely.
- **Security:** We set up an admin role to prevent unauthorized access or retrieval of information.
- **Replication:** Given that the Tweet data is the most valuable asset to develop this project, we set up a replication to a **different** virtual machine which also had CouchDB installed. We decided to do this, since we wanted to minimize the risk of data loss.

In addition to the mentioned configuration aspects, we changed the storage of the database files to the mounted drive of the instance. At first, this step was omitted and it caused issues due to the nature and volume of the Tweets. We realized this was an issue when we began using the set views and started seeing that we had no response from the server. We checked the log files from CouchDB and saw that suddenly there were no more additions to the database. After careful debugging, we discovered that even though the volume was attached, CouchDB was not using the additional storage which is why we had to migrate the data and reconfigure the database. We had to do this same process for the replication database set up in the different virtual machine.

The project was made using version 1.6.

Fault Tolerance & Error Handling

The system we proposed has certain levels of fault tolerance in the different components.

Twitter Harvesting:

As explained in the section above, whenever an error is received via the Tweepy interface, we handle it by cooling off the system during a 15-minute break and resetting the variables automatically. In addition to this, since we have multiple streamers running automatically, if one fails, since they are all independent, the other can keep harvesting Tweets even if one machine stops working.

Database:

We have replication in place which allows us to backup the data in a different VM in case the main one fails. The scripts were also created to tolerate write errors by dismissing the write and report it to a text file hence being able to tolerate write errors.

Scale up the system by Ansible and Boto

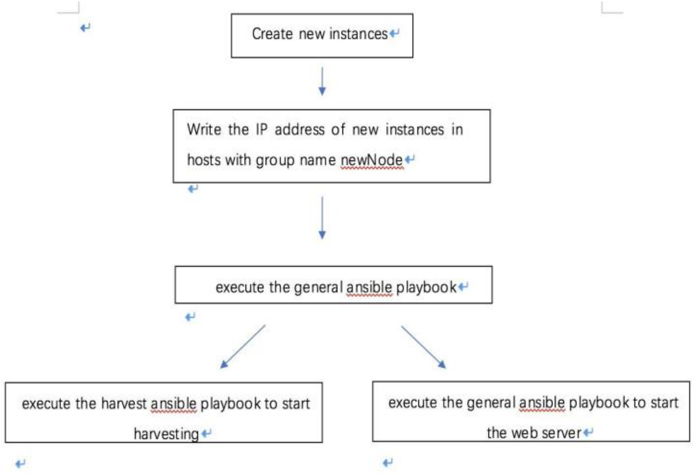

Create new instances
Write the IP address of new instances in hosts with group name newNode
execute the general ansible playbook
execute the harvest ansible playbook to start harvesting
execute the general ansible playbook to start the web server

Figure 2. Boto and Ansible

The system we need to create should have a good scalability which means the system should be easily extended by add more nodes to process the job on a parallel basis. We use Boto to launch and terminate the instances and Ansible to set up the environment for new instances and run the processes. To execute boto, we must first install it on the monitor computer which is our own computer. and import in into the python file we created for launching or terminating the instances. There are three arguments for executing this python file, one is “-operation” for determining launching or terminating operation. another one is “-n” for determining the number of instances that going to be created. The third one is “-id” for determining the id of instance that going to be terminated. According to the parameters of the arguments, the python file will execute the corresponding operations. If the operation is launching, the python file will check the status of the new instances and when the status turns into running, it will print out the details of the instances. In this project, we only created one instance and did not attach volume to it since this instance does not need a database. All we need is to print out the IP address of the instances, which will then be used in ansible.

To execute ansible, we must first install it on the monitor computer which will be used to manage all the deployment tasks, and establish an SSH connection with all the instances create on Nectar. After the inventory file is created and stored in /etc/ansible/, we then created three playbooks to automate a series of tasks for our deployment and configuration on cloud, namely “general.yml”, “harvest.yml” and “web.yml”. The purpose of the “general” playbook is to set up general environment and get the private key and source code from monitor computer and github.

- Install python-setuptools: this is required for packaging and distribution of python projects
- Install pip: pip is a package management system to install and manage software packages written in python. We will need it to install other libraries needed for our project.
- Install required packages-git: we need to install git on remote machines so that we can pull the repository which contains all the source code.
- System cache update: to update the system

The next playbook, “harvest.yml”, is to set up the environment for tweet harvester on a new instance and then run the harvester, and connect all of this with CouchDB. Majority of the tasks included in this playbook are installing CouchDB and various python libraries that are needed for our application to run. The remaining two tasks are to keep running the harvester code at the background on the new instance.

The final playbook is to set up the environment for web server. First, we would need to install flask, CouchDB, CouchDB_flask and simpleJson, and then run the web server at the background.

After creating the new instances with Boto and getting the IP addresses of the new instances. we wrote these IP address into the hosts within the group named newNode. If the new instances are created for harvester, then we will execute general.yml and harvest.yml. After this, the new instances will start to harvest the tweets and store tweets into the CouchDB. If the new instances are created for web server, then we will execute general.yml and web.yml. In this case, the new instances will run the web server and the web pages is accessible by any public IP address. Essentially, we only need one instance to store and run the web server to avoid IP address conflict. In this project, we created three instances manually and created one instance for harvester with Boto and Ansible.

Twitter Sentiment Analysis

Twitter has been widely used to perform sentiment analysis due to various reasons. One of them is the fact that, to some extent, information is publicly available unlike other social media sites such as Facebook or LinkedIn. Sentiment Analysis deals with the task of determining positive, negative or neutral polarity in the given text. It can be extended to understand other types of emotions but it is out of scope for the development of our project.

Many tools and techniques have been developed to complete this task. We decided to compare two out-of-the-box tools which are Textblob [4] and the database of labeled words used in AFINN [5]

The first step was to trim the tweets by using regular expressions, and we did the following tasks:

1. Change the text to lowercase letters
2. Convert links and URLs to the text “URL”
3. Convert all usernames (@example) to “USERNAME”
4. Remove any additional whitespaces
5. Replaces hash-tagged words (#word to word)
6. Remove any special character

Once we had cleaned all the tweets, and they have been normalized, we passed this text to the respective libraries. To assess the performance of these libraries and tools we manually labeled a random set of 150 Tweets (to our own judgement) and compared the outcome of both libraries.

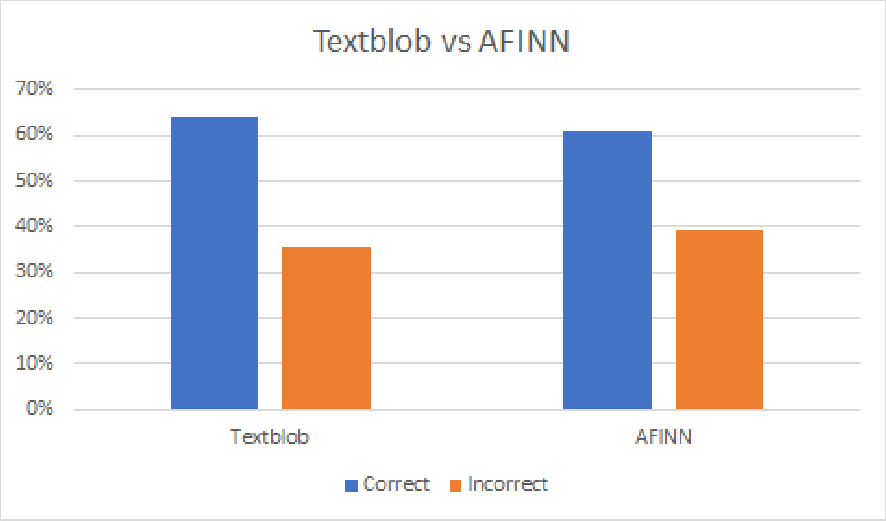


Figure 3. Textblob AFINN Comparison

As we can see from the figure above, Textblob slightly outperforms AFINN for this particular subset of words. Given this, in conjunction with the fact that AFINN has a large but limited set of words which are labeled we decided to use Textblob in the development of this project.

As a future endeavor, we could utilize different strategies to improve the development of the sentiment analysis such as Lemmatization.

Descriptive Data

To query the data, we created multiple views by using the MAP/REDUCE functionality in CouchDB.

Scenario 1 - Tweets by Time of Day

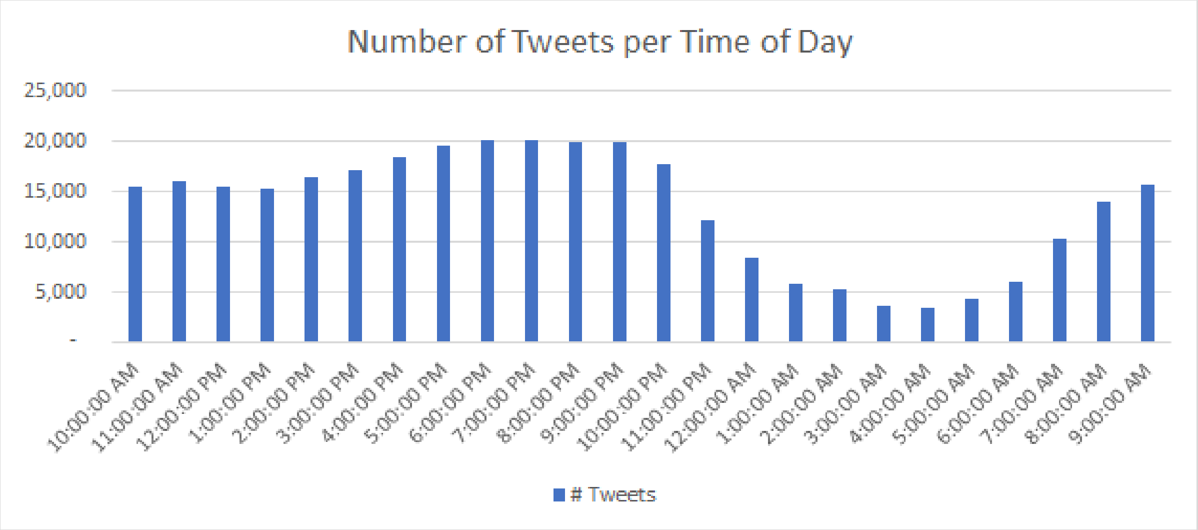


Figure 4. Number of Tweets per Time of Day

In our sample, the majority of Tweets were created during evening peak hours. Probably when people get off work and get into their social activities. We came to this conclusion since the peak of tweets we received was from 5pm to 9pm. It is worth noticing that Twitter delivers the creation time in an UTC format, it is up to us to convert it to local Australian time. As expected during the early morning the amount of Tweets is the lowest since most people are asleep.

Scenario 2 - Sentiment Tweets by Time of Day

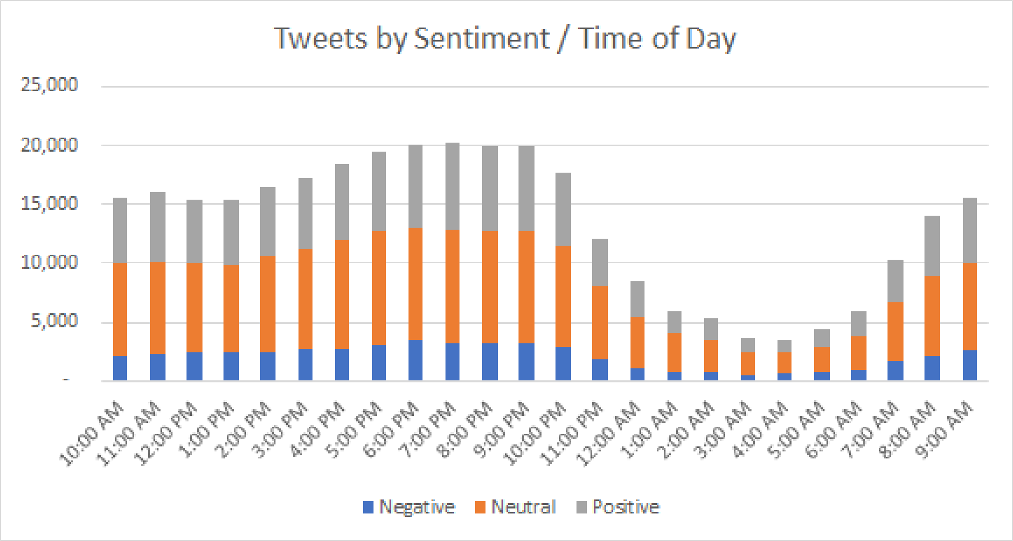


Figure 5. Tweet Sentiment

Drilling down and building on our previous scenario we determined the sentiment of the tweets we harvested across the day. As we can see from the graph, most of the Tweets are labeled as “Neutral” or having no sentiment. This is partly due to some constraints in the Textblob library and partly due to the fact that many tweets have an informative nature. However, an interesting pattern is found. Four our set of Tweets, the proportion of Tweets that have positive sentiment is higher than those that have negative sentiment across the day.

The next section contains information relevant to tweets with the label “Migration”, which as explained earlier, were extracted using a list of keywords curated by the members of the group. To determine correct results, we used suburbs which had received tweets from all sentiment options (Positive, Neutral and Negative) as well as more than 20 tweets per suburb. We aimed to find any correlation between the strongest sentiment in the suburbs and demographics such as income level, educational background and employment ratio.

Scenario 3 - Sentiment Towards Migration vs Employment Ratio

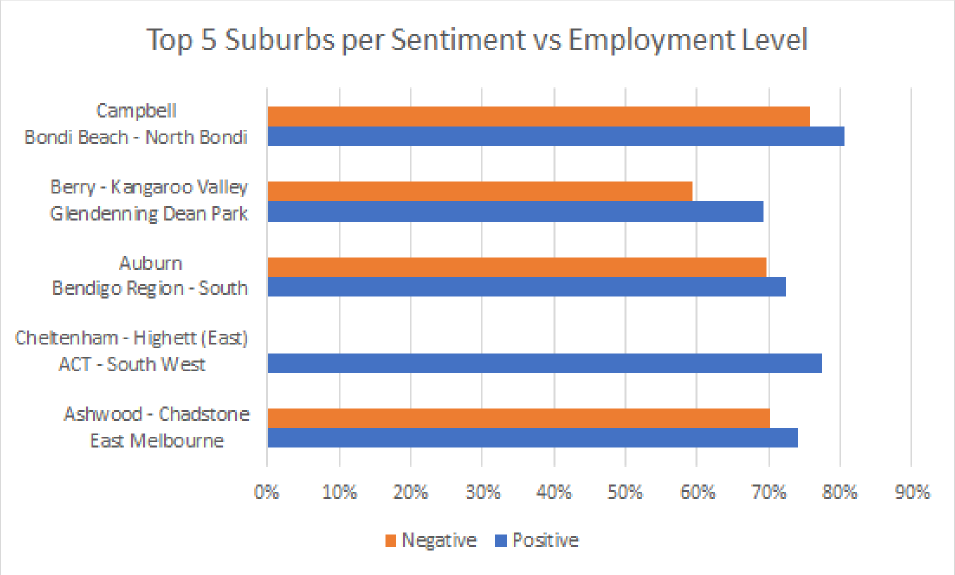


Figure 6. Suburbs per Employment Level

In terms of Employment level, the correlation is not evident. Even though there is a slight correlation between the level of employment and the sentiment towards migration (a suburb has higher employment ratio would feel slightly more positive towards migration). The difference is still not significant enough to make a final decision. Our conclusion at this point is that there is no direct correlation between employment level and the sentiment of people towards migration.

Scenario 4 - Sentiment Towards Migration vs Education Level

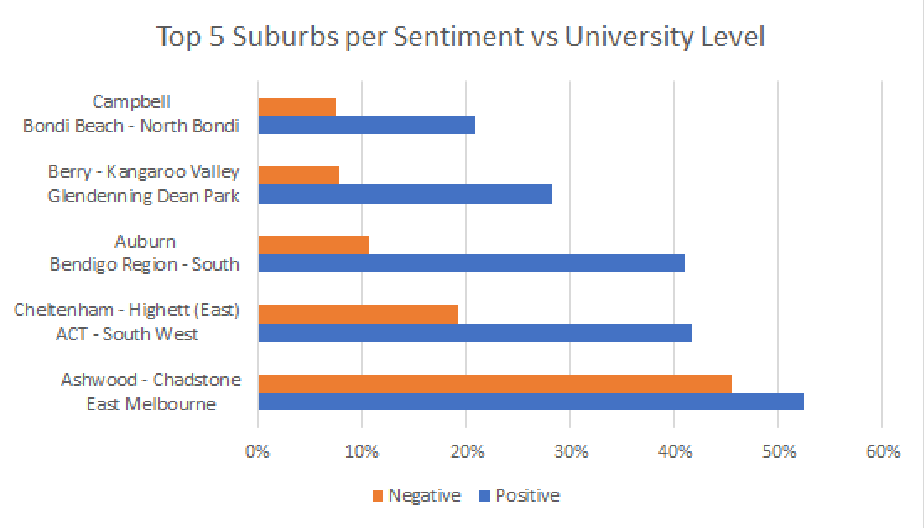


Figure 7. Suburbs per University Level

The graph above shows for the top 5 suburbs in amount of tweets the percentage of people who have at least completed a Bachelor Degree and the overall sentiment. This graph can be read in the following way. The suburb Campbell has an overall Negative sentiment towards Migration (its negative percentage is higher than positive), and approximately 9% of the working-age population have attended a University.

Higher education level positively correlates with the overall sentiment towards migration. For Suburbs that have high percentages of University level education, the general sentiment towards migration is more positive than suburbs where the percentage of University level of education is lower.

Scenario 5 - Sentiment Towards Migration vs Income Level

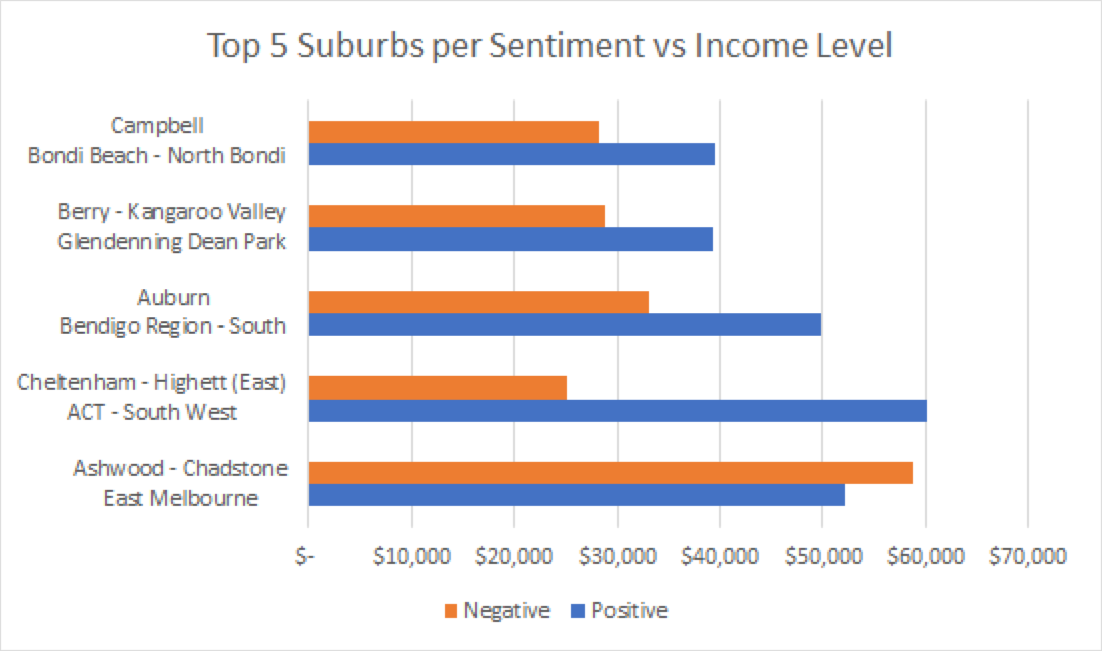


Figure 8. Suburbs per Income Level

All in all, there are correlations between the sentiment towards migration and the demographics of suburbs. If we could analyze at a deeper level this could lead to interesting insights such as the ones showed in this post published by The Guardian [6] in which there are correlations between people who voted for and against the “Brexit”.

Aurin Data

Aurin provides an exhaustive set of datasets for researchers to utilize and use to augment results. For this project, we wanted to understand what is the demographics of the suburb of which positive or negative tweets are sent that are related to migration. The dataset we used is “Suburb Working Age Employment and Income 2011”, created by Grattan Institute for the productive cities report and published in 2013. This dataset is collected from working-age (between 25 and 65 years old) population from different suburbs throughout Australia (all data relies on census data collected by ABS). It provides us different demographic variables of importance such as employment ratio and educational background. More specifically, pertinent to our data analysis we have the following values:

- Employment: Percentage of the population that are employed,
- Education: Percentage of the population which have attained a Degree or a Certificate up to any of the categories listed below:
  - University - proportion of population with a Bachelor Degree or higher
  - TAFE - proportion of population with TAFE qualification
  - High School - proportion of population with no post-school qualification (may include those who did not complete high school)
- Income: The median income of individuals aged 25 to 65.

The main link between Twitter data and Aurin data is the geopositioning of the Tweets. This is a challenge since not every Tweets is geocoded which means that only a small percentage of the Tweets are actually useful to augment with Aurin. Using these different datasets where were able to create different polygons per suburb based on the data from AURIN. By leveraging from the Python library Shapely, we managed to easily create one polygon per suburb and determine, given the coordinates from the available Tweets, if it belonged to a specific Suburb.

Given that this process is computationally expensive, since we have to iterate multiple times through multiple files, we created a database that stored this demographic data. Due to the nature of the data, it does not change very often since government census or state released data have long cycles. Using a Python Script, we did the initial load and labelling of readily available data and subsequent analysis can be performed from the CouchDB database.

This was used to correlate the sentiment of the tweets with different demographics such as wealth (income), education level, age and employment rate.

To extract and create the polygons we would first need to create a shapefile for our target dataset. This can be easily done by utilizing the Spatial Data Manipulation tools available on AURIN portal. Once the shapefile is generated, we could process it with QGIS (an open source geographic information system) and its plugin application MMQGIS, which can produce two csv files for each shapefile. One file contains all the attributes from the original dataset, and the other contains all the coordinates for each geographical region, and all figures from these two files can be joined via “shapeID”, which is used for identifying each suburb area. With the information of “shapeID” and the coordinates, we can then create a polygon for each suburb that appears in our dataset.

User Guide

User Invocation – Ansible

Boto

Using the following commands to execute the boto.py file and make sure that boto has been installed on the computer:

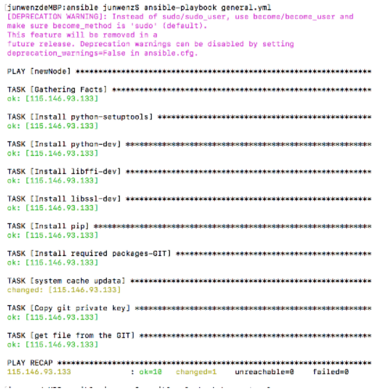
python boto.py -operation launch -n 1 or *python boto.py -operation terminate*

Ansible:

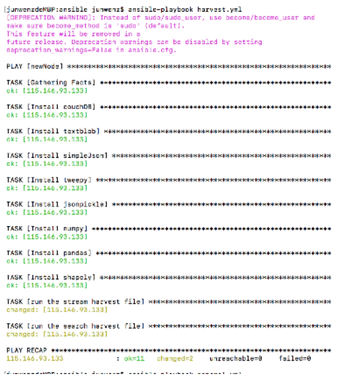
a) Using the following commands to execute the ansible files. Make sure that ansible has been installed on the computer and playbook and inventory files are stored in */etc/ansible/.* The key to *ssh* the instances should be stored in *~/.ssh.* The key to access the github should be stored in */etc/ansible/.*

ansible-playbook playbook_name.yml (ansible-playbook harvest.yml in our project)

The successful running result is:



```
[junwenzdeMBP:ansible junwenz$ ansible-playbook general.yml
[DEPRECATION WARNING]: Instead of sudo/sudo_user, use become/become_user and
make sure become_method is 'sudo' (default).
This feature will be removed in a
future release. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.

PLAY [newNode] ****************************************************************

TASK [Gathering Facts] ********************************************************
ok: [115.146.93.133]

TASK [Install python-setuptools] **********************************************
ok: [115.146.93.133]

TASK [Install python-dev] *****************************************************
ok: [115.146.93.133]

TASK [Install libffi-dev] *****************************************************
ok: [115.146.93.133]

TASK [Install libssl-dev] *****************************************************
ok: [115.146.93.133]

TASK [Install pip] ************************************************************
ok: [115.146.93.133]

TASK [Install required packages-GIT] ******************************************
ok: [115.146.93.133]

TASK [system cache updata] ****************************************************
changed: [115.146.93.133]

TASK [Copy git private key] ***************************************************
ok: [115.146.93.133]

TASK [get file from the GIT] **************************************************
ok: [115.146.93.133]

PLAY RECAP ********************************************************************
115.146.93.133             : ok=10   changed=1    unreachable=0    failed=0
```



```
[junwenzdeMBP:ansible junwenz$ ansible-playbook harvest.yml
[DEPRECATION WARNING]: Instead of sudo/sudo_user, use become/become_user and
make sure become_method is 'sudo' (default).
This feature will be removed in a
future release. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.

PLAY [newNode] ****************************************************************

TASK [Gathering Facts] ********************************************************
ok: [115.146.93.133]

TASK [Install couchDB] ********************************************************
ok: [115.146.93.133]

TASK [Install textblob] *******************************************************
ok: [115.146.93.133]

TASK [Install simpleJson] *****************************************************
ok: [115.146.93.133]

TASK [Install tweepy] *********************************************************
ok: [115.146.93.133]

TASK [Install jsonpickle] *****************************************************
ok: [115.146.93.133]

TASK [Install numpy] **********************************************************
ok: [115.146.93.133]

TASK [Install pandas] *********************************************************
ok: [115.146.93.133]

TASK [Install shapely] ********************************************************
ok: [115.146.93.133]

TASK [run the stream harvest file] ********************************************
changed: [115.146.93.133]

TASK [run the search harvest file] ********************************************
changed: [115.146.93.133]

PLAY RECAP ********************************************************************
115.146.93.133             : ok=11   changed=2    unreachable=0    failed=0
```

Figure 9. Ansible Run

Web Application

Introduction and general architecture

Web service for this project is a comprehensive system which dynamically reads data from couchDB and visualizes the data analytics scenarios results as charts or graphs to provide a more straightforward way for potential users of our system.

The famous lightweight web server named Flask has been adopted in our project to help create a micro web server which retrieves data from both CouchDB database documents and CouchDB database views. However, in fact, we did a lot of research and study among different web APIs before doing this project to decide which web server library to use for this specific project. We tried Django, which is a relatively more mainstream web API frameworks in industry compared to Flask. However, we found out, although Django may offer us a more comprehensive function and has less expected latency, it has a very steep learning curve for beginners. Furthermore, it seems to be a little bit clumsy for a small project like ours. In the end, given the fact that we are dealing with a small project with no complicated URL handling or templates rendering, we decided to apply Flask instead of Django.

As a matter of fact, structures in Flask are strictly defined by its source code in which folder named *templates* are referred as to directory containing, and only containing html files; folder named static are referred as to directory including CSS files, JavaScript files and all other images the html files may call.

To route Json data to corresponding templates, we set up route() decorator and after a few steps of data reformatting, we assign the Json data to a new variable, and render the templates. As you may interest, all steps mentioned above can be found in *main2.py* of my our folder.

Theoretically speaking, one server connection would be enough. However, to minimize the possibility that server connection crushing can cause the whole web server to shut down, namely the fault tolerance of our system, we decided to create a new server connection for each route. Also, local host is set as 0.0.0.0 to listen to all public IPs.

Test

All web service related files can be found on instance named Analytics Web whose IP address is 115.146.93.162 with Ansible setting all needed configurations.

By entering http://115.146.93.162:5000/index, users can visit the index page with all other pages being linked through it.

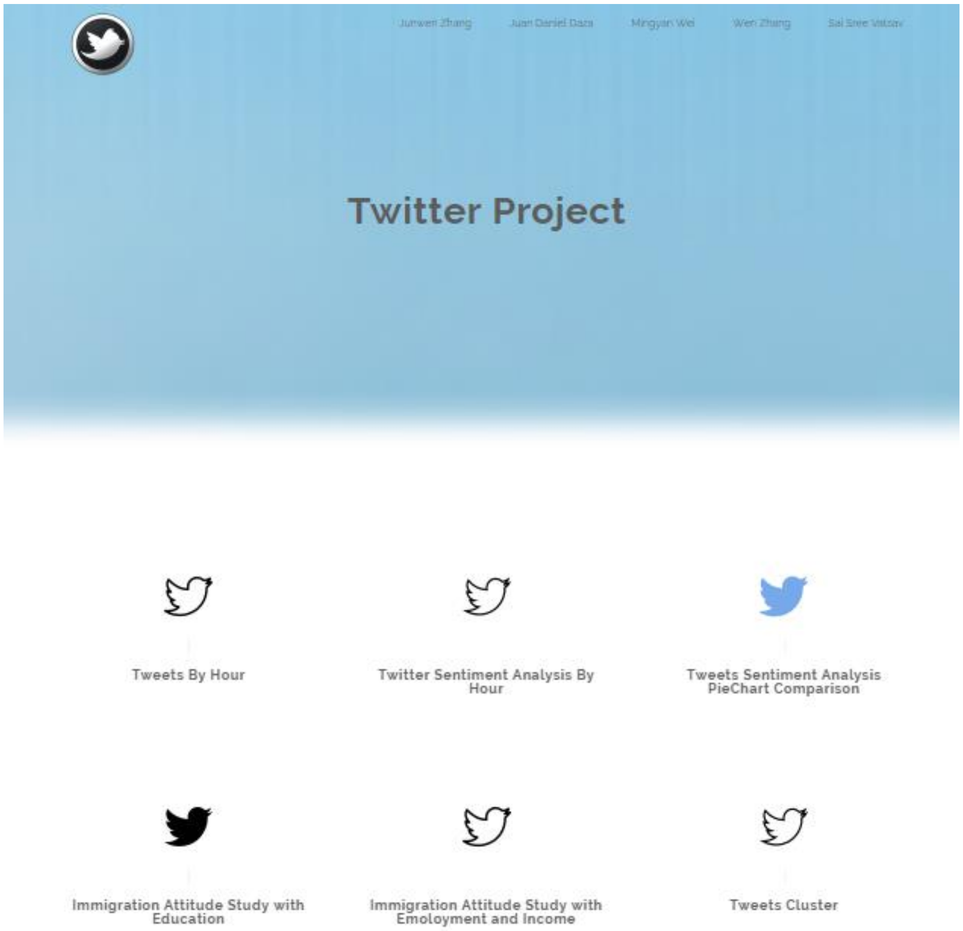

Junwen Zhang
Juan Daniel Daza
Mingyan Wei
Wen Zhang
Sai Sree Vatsav
Twitter Project
Tweets By Hour
Twitter Sentiment Analysis By Hour
Tweets Sentiment Analysis PieChart Comparison
Immigration Attitude Study with Education
Immigration Attitude Study with Emoloyment and Income
Tweets Cluster

Figure 10. Web Page

Error Handling

the Socket error: the address is already in use is frequently met when debugging. That is caused by the previous processes occupying IP address: 0.0.0.0:5000. Hence, we need to kill those processed and get this IP address free. Command: sudo losf -I:5000, therefore, can be really helpful.

RESTful API

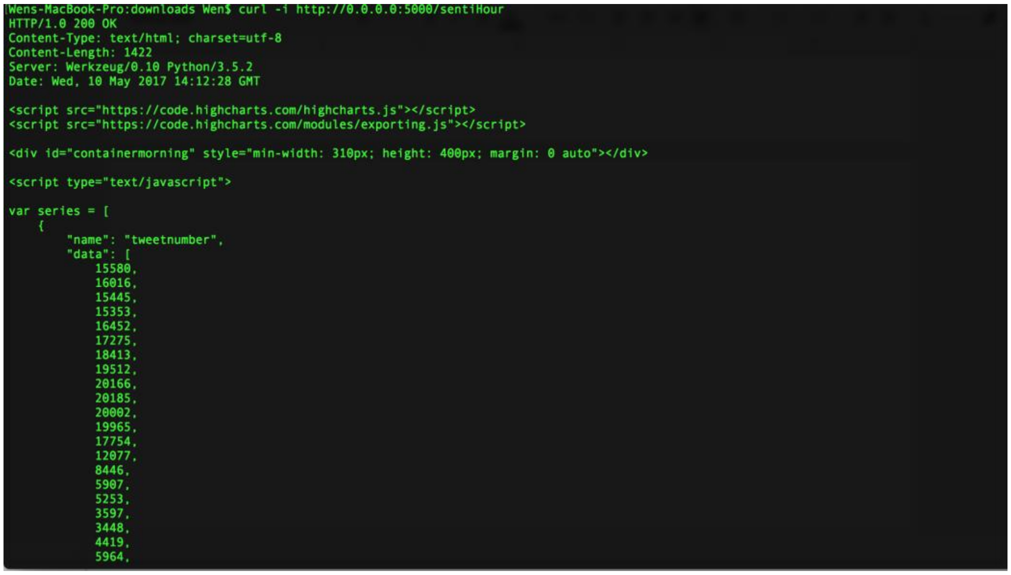
Essentially speaking, Rest API defines a completely new architecture for web, and it is extremely helpful when nowadays web is more viewed as a comprehensive, interactive and complex application online instead of simply delivering static data to users. As long as both clients and servers agree on HTML, they can interact in a variety of ways based on REST API

Flask, the web API library obeying REST API, requires the server-side users, in this case, our team, to carefully provide everything a potential client-side user would ask for.

Four important verbs are defined as one of HTTP standards, namely GET, POST, PUT AND DELETE.

In this specific project, every route entry point that is associated with the url is the GET HTTP method. So as you can see in main.py file, `@app.route('/..', methods=['GET'])` is used for most of our functions. All those responses would be HTML data.

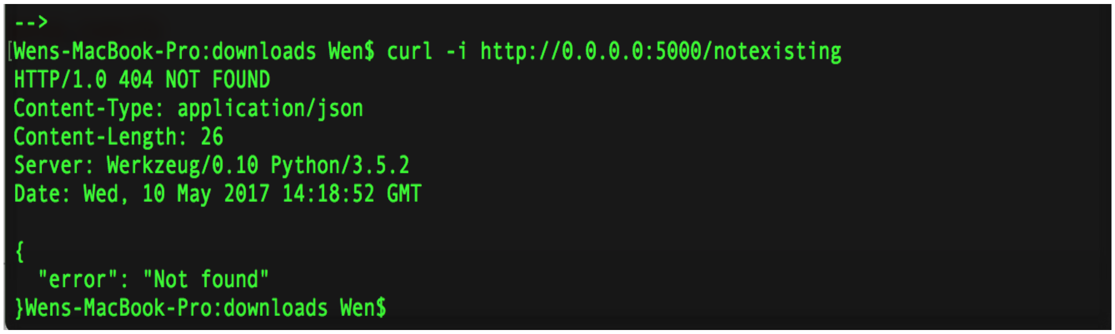
To validate the HTTP requests, in lieu of simply checking the code in web browser, we used curl command in Mac Os terminal just like the following screenshot:



```
Wens-MacBook-Pro:downloads Wen$ curl -i http://0.0.0.0:5000/sentiHour
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1422
Server: Werkzeug/0.10 Python/3.5.2
Date: Wed, 10 May 2017 14:12:28 GMT

<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>

<div id="containermorning" style="min-width: 310px; height: 400px; margin: 0 auto"></div>

<script type="text/javascript">

var series = [
    {
        "name": "tweetnumber",
        "data": [
            15580,
            16016,
            15445,
            15353,
            16452,
            17275,
            18413,
            19512,
            20166,
            20185,
            20002,
            19965,
            17754,
            12077,
            8446,
            5907,
            5253,
            3597,
            3448,
            4419,
            5964,
```

Figure 11. HTTP Requests.

Furthermore, when we also handle the 404 response just in case the user might input some non-existing urls by accidents. To validate the HTTP requests, we do the similar command as mentioned before. And we get the following response:



```
-->
[Wens-MacBook-Pro:downloads Wen$ curl -i http://0.0.0.0:5000/notexisting
HTTP/1.0 404 NOT FOUND
Content-Type: application/json
Content-Length: 26
Server: Werkzeug/0.10 Python/3.5.2
Date: Wed, 10 May 2017 14:18:52 GMT

{
  "error": "Not found"
}Wens-MacBook-Pro:downloads Wen$
```

Figure 12. Validate HTTP Requests Errors

Twitter Scenarios Live Webpage

We have created different graphs for the report and the final website due to different reasons. Among them, is the fact that for showcasing the data in the live website we considered best to use different graphs and avoid monotony in the site. As it can be seen from the previous section, most of the graphs followed the same guideline. These graphs were used for **exploratory** purposes. The following figures will display the live graphs posted on the website.

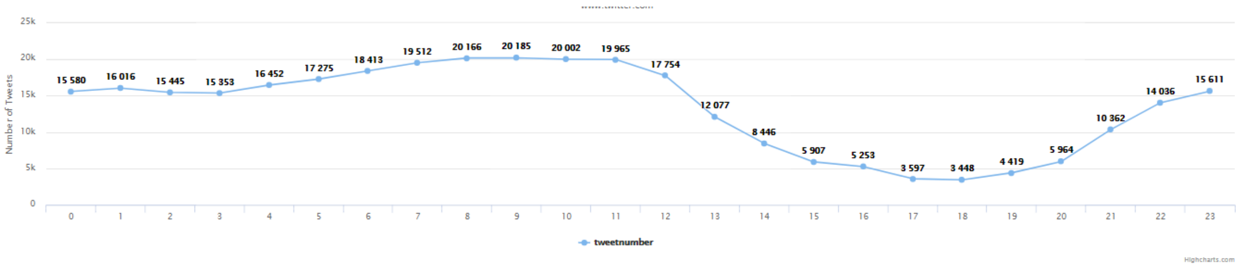


Figure 13. Tweets by Hour

The above figure shows the number of Tweets per time of day as shown previously.

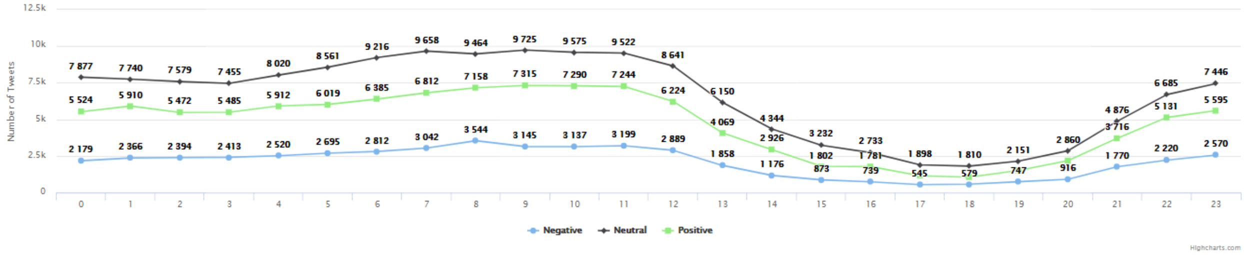


Figure 14. Number Tweets per Sentiment.

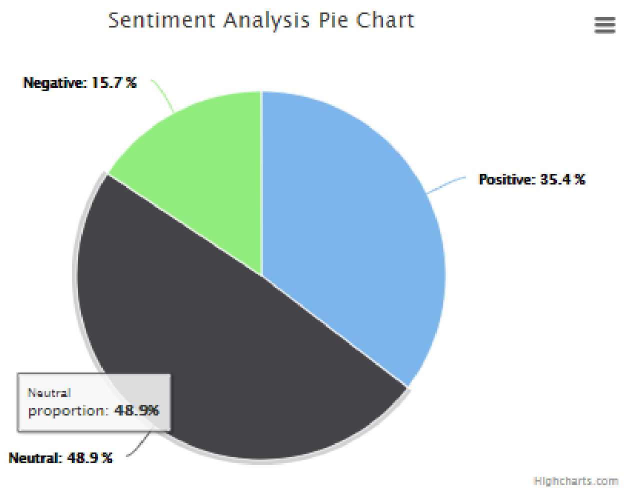


Figure 15. Total Tweets per Sentiment

The figure above is the distribution of sentiment across our complete data set. We decided to showcase this to be able to expose the general feel of the subset of Tweets we managed to harvest. Overall, we collected tweets that had a more positive sentiment.

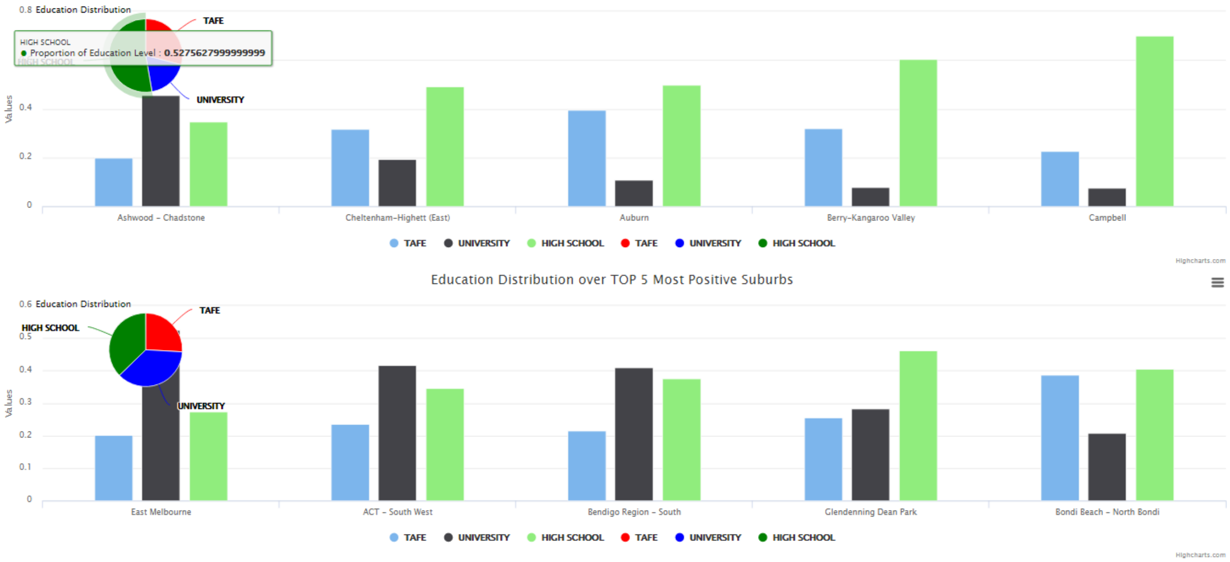
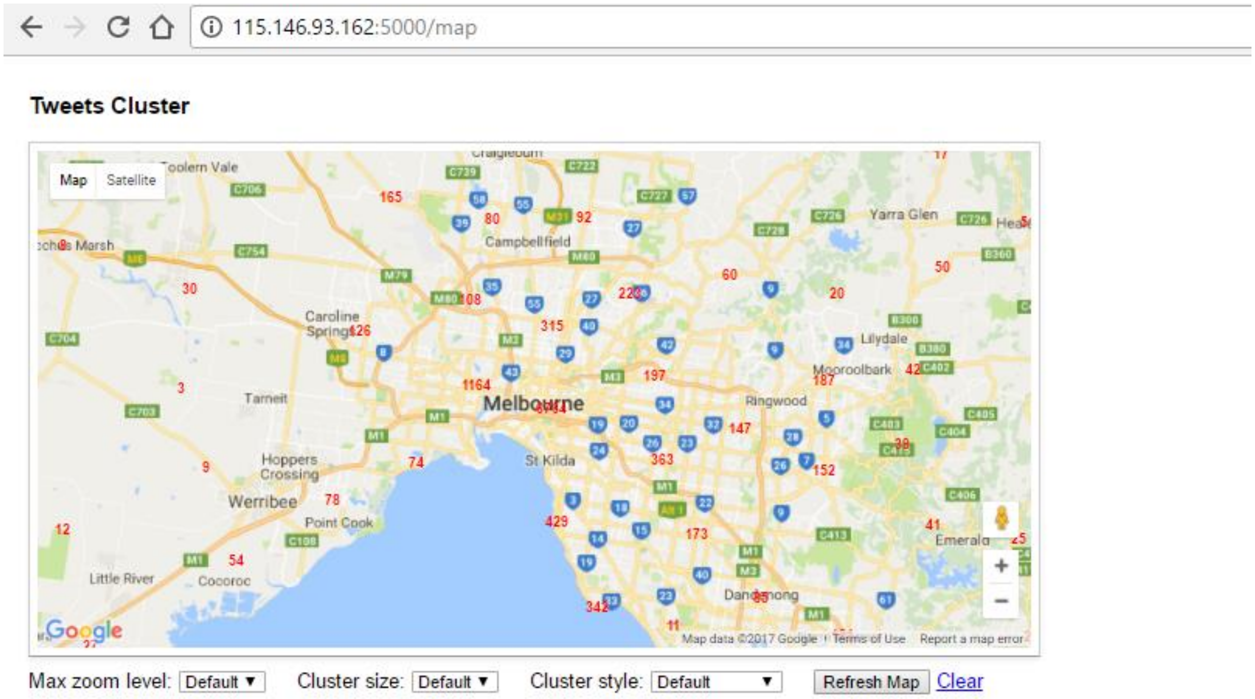


Figure 16. Sentiment Top 5 Suburbs per Education Level.

As seen previously, these graphs tell us the correlation between the level of education and the sentiment of the Tweets for the Top 5 suburbs. Meaning, the suburbs out of which we had the most data. Again, we can observe that suburbs with a high percentage of university education have a more positive sentiment towards migration.



115.146.93.162:5000/map

Tweets Cluster

Max zoom level: Default ▼ Cluster size: Default ▼ Cluster style: Default ▼ Refresh Map Clear

Figure 17. Map of Number of Tweets

Out of fun, we decided to map the number of Tweets we got in each suburb to showcase and learn more about Google API integration and service.

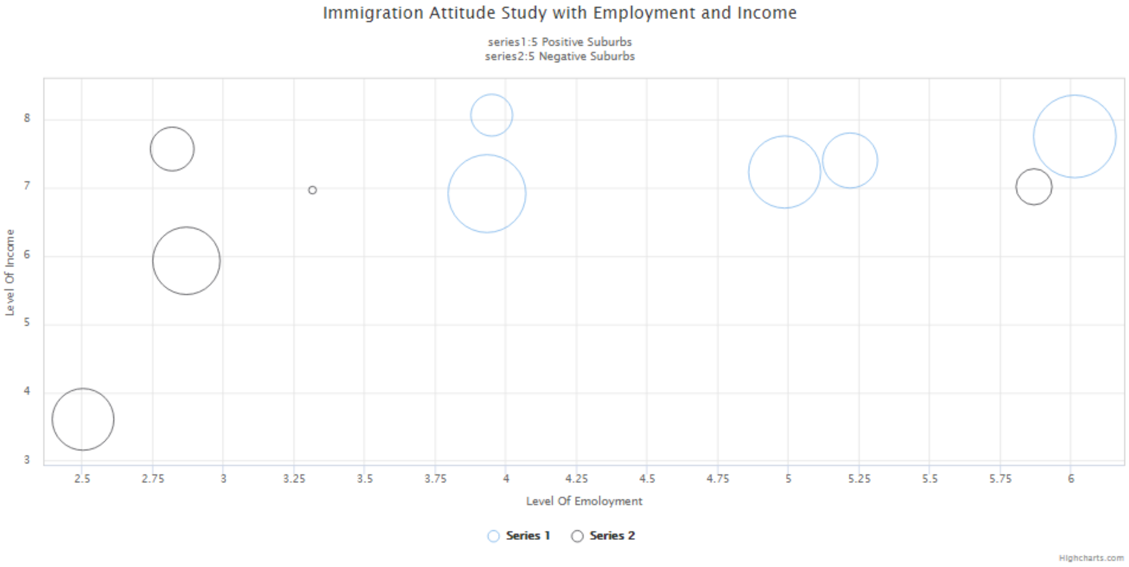


Figure 18. Sentiment Level Top Suburbs vs Income and Employment

As stated previously, we did not want to showcase the same graph three times which is why we tried to vary the images in the live site. The figure above showcases the correlation between positive (series 1) and negative (series 2) and the level of employment and education. From the graph, we can see the correlation between positive tweets and the percentage of employment (upper right area of the graph).

NeCTAR Research Cloud

The NecTAR Research cloud is a very useful tool to create and manage different virtual machines and volumes in a cloud based environment. It is comparable to AWS and Microsoft Azure and it offers similar flexibility and benefits for researchers across Australia.

NeCTAR has multiple benefits out of which we want to highlight the fact that it is a very complete solution to deploy virtual private clouds. Compared to AWS or Azure it has a lower level complexity which truly enables a learning experience whilst having the same computing capabilities of the large commercial solutions available. In addition to this, by using NeCTAR, we have an easy way to manage the VMs and Volumes using the readily available dashboard.

Additionally, since NeCTAR is using OpenStack, we are using and learning the latest technologies and uses that transcend to an industry level. We can then explore different orchestration technologies and pursue different subjects that will have high commercial value today.

NeCTAR also provides benefits since it has a well performing support operation. In our case, we had issues at first deploying volumes and found that response times from the Support Center in NeCTAR was outstanding. This is a benefit since this type of service in a commercial available platform has a high cost for the end user.

Some drawbacks however appeared throughout the development of the project. The main disadvantage we found was the availability the service offers. The service is not as reliable and stable as the commercial available providers. In our case, we had connection issues and the virtual machine could not manage to find volumes for some period of times.

Ansible

Ansible is a simple IT automation engine which supports cloud provisioning, configuration, application deployment, intra-service orchestration and many other IT needs. Ansible models the interrelationships of all systems together rather than managing only one of them at a time. It is easy to deploy and uses a very simple language (YAML) that is very close to plain English which allow us to specify our tasks in a very intuitive way (www.ansible.com).

Ansible is executed by sending out “Ansible modules” to the nodes. These modules are programs written with the desired state of the system, which will be executed over SSH and removed when the task is done.

By default, Ansible represents every machine it manages using a simple INI file that includes all managed remote machines and stores them into groups according to user's' specification. This inventory file in our system is called “hosts”, and it stores a list of IP addresses of remote hosts (grouped in “cloud”, “web” and “test”) managed by Ansible. Once inventory hosts are listed, variables can be assigned to them by adding directly after their addresses in the inventory file or use a dynamic inventory to pull from OpenStack or other data sources. In our case, we just assign the location of the key and username directly in the inventory file for simplicity.

Playbook is the language for Ansible’s configuration and orchestration. Playbooks are suitable for deploying more complex systems as they provide a simple configuration management and multi-machine deployment system. It was not designed to be a programming language or script but rather a modelling language for configuration. Playbooks can also execute a list of tasks sequentially, orchestrate steps of any manually ordered process (ansible.com).

Conclusions

- Cloud computing offers extreme flexibility for developers as it allows resources to be created/deleted on demand. An example of this, is how we were able to capture Tweets rapidly which would not have been possible without the flexibility of NeCTAR. Had we used our personal computers we would have been heavily restricted to collect Tweets at the pace we did.
- CouchDB requires to some extent the need to know beforehand the processing requirements to design the views. Unlike SQL databases the development of ad-hoc queries is not as straightforward.
- Sentiment in a Tweet is highly dependent on context. It is not possible to have a “one size fits all” library. It requires to be configured, trained to understand the different contexts.
- Using MAP/REDUCE views is much more efficient than handling data with Python. As an example, using Python, if the connection is lost then the process interrupts and quits abruptly before the analysis or processing can be made. By querying against a view, the format is defined and speeds up processes in a great manner.
- Teamwork and team cooperation is critical when developing software and a large-scale project. Everyone needs to be active and engage in the overall development.

References

[1] P. Rivera, "Tweepy Documentation," [Online]. Available: http://www.tweepy.org/.

[2] "CouchDB Python," [Online]. Available: https://pythonhosted.org/CouchDB/. [Accessed 15 April 2017].

[3] The Apache Software Foundation, "Apache CouchDB," [Online]. Available: http://couchdb.apache.org/. [Accessed 15 April 2017].

[4] S. Loria, "TextBlob," [Online]. Available: https://textblob.readthedocs.io/en/dev/. [Accessed 15 April 2017].

[5] L. K. Hansen, A. Arvidsson, F. Å. Nielsen and E. Colleoni, "Good Friends, Bad News - Affect and Virality in," in *The 2011 International Workshop on Social Computing,*, 2011.

[6] C. Barr, "The areas and demographics where the Brexit vote was won," 25 June 2016. [Online]. Available: https://www.theguardian.com/news/datablog/2016/jun/24/the-areas-and-demographics-where-the-brexit-vote-was-won. [Accessed 30 April 2017].

[7] B. Holt, Writing and Querying MapReduce Views in CouchDB, O'Reilly Media, 2011.

[8] A. J. Chris, L. Jan and S. Noah, CouchDB: The Definitive Guide, Oreilly Media, 2010.

[9] B. Holt, Scaling CouchDB, O'Reilly Media, 2011.

[10] M.-C. Yang and H.-C. Rim, "Identifying interesting Twitter contents using topical analysis," *Expert Systems with Applications ,* vol. 41, p. 4330 – 4336, 2014.

[11] F. Costa, H. Te-Tu, N. Vinayan, R. Liu and X. Huang, "Chicago," Melbourne, 2015.

[12] Grattan institute, "Suburb Working Age Employment and Income 2011," AURIN, 2011. [Online]. [Accessed 22 April 2017].