Cluster and Cloud Computing
Assignment 2

Team 22
Victor Ding 1000272
Zhuolin He 965346
Chenyao Wang 928359
Danyang Wang 963747
Yuming Zhang 973693

14th May 2019

Table of Contents

1. Introduction

This report presents the tasks done by Group 22, which is to explore the Seven Deadly Sins through social media analytics. In the report, it will be divided into five sections. The first section describes the background of this project and reasons behind selecting this topic. The second section explains the design and implementation method. In this section, there are pictures and code snippets. The third part is about information deduced from the twitter data. This part will show the results that have been compared with information from AURIN, and there are pictures and tables to better illustrate the results. The fourth part is about the analysis of the findings; it also discusses the assumptions comparing with the analyzed data. The fifth and last section explains advantages and disadvantages of various parts of this project.
There are five members in our group, and each team member is responsible for different tasks. The details are as follows:

Chenyao Wang: Server setup and configuration, dynamic deployment with Ansible;
Yuming Zhang: Server setup and configuration, dynamic deployment with Ansible;
Zhuolin He: Twitter harvesting, CouchDB management;
Danyang Wang: Sentiment analysis;
Victor Ding: Implementation of the web application, CouchDB MapReduce functions.

2. Background

2.1 Cloud System and Dynamic Deployment

For this assignment, we leveraged the University of Melbourne Research Cloud, which is a NeCTAR infrastructure with the University of Melbourne's interface. NeCTAR is “The National eResearch Collaboration Tools and Resources project which provides an online infrastructure which supports researchers to connect around the world. [1] In this project, NeCTAR is used to deploy instances for the team to work on. We installed necessary packages such as pip, CouchDB, and we also used one instance to run the twitter harvest code to obtain the data, then we analyzed the data on there and visualized it via the web application.

As for the dynamic deployment, we used Ansible for the deployment. Ansible is a powerful IT automation engine which automates cloud, and application deployment.[2] Ansible is chosen for its ability deploy all configurations to every server with one click so that all the team members can re-deploy the whole setup even if one does not know every piece of setup details.

2.2 Database

The database is Apache CouchDB. It is open-source NoSQL database software that uses JSON to store data, JavaScript as its query language for MapReduce, and HTTP for APIs.(wiki)[3][4] In this project, CouchDB is to store the twitter data, and to filtering and sorting all the tweets and summarize the results via MapReduce method. In the project, CouchDB 2.3.1 was chosen.

2.3 Twitter Harvest

Twitter is a social networking service which allows users to post and interact with messages known as “tweets”. (wiki) In this project, we are using the Twitter APIs to get the data from the tweets, and this is called Twitter Harvest. Since Twitter is one of the most popular social media, it means that many people around the world send tweets every day, and there will be a lot of information in these tweets. Our task this time is to take useful data from this information and try to analyze the impact of the seven deadly sins on people in Australia.

2.4 Sentiment Analysis

Sentiment Analysis is to identify and extract the subjective information in the source material, for this project namely, the degree of a raw tweet related to one of the deadly sins and the emotion expressed in the text. The processed data are then used to combined and compared with AURIN data [5] to validate a few scenarios across different states in Australia. In this section, the Natural Language Toolkit (NLTK) [6] in python are used to do the sentiment process.

2.5 Web Application

For this assignment, a web application is created for visualizing and analyzing the collected data. The web interface displays a map which includes the data. The user can hover the mouse over the specified area to see what the twitter data is in the state or territory. In addition, other links for case analysis are also shown to present the results.

3. System Design and Implementation

3.1 Instances Setup

Before our deployment, we set up three instances for the team to perform Twitter Harvester, Sentimental Analysis and Web. In other to set up the instances, we need to login into the website https://dashboard.cloud.unimelb.edu.au, which is the University of Melbourne Research Cloud. Then we will need to create the instance with three servers are all have two cores and 9G memory. Each of the servers has attached their own volume. In Figure 1, it is a screenshot of our instances, and the instance named web is used to deploy a website and CouchDB storages, the instance named couchdb is used for Twitter harvest and CouchDB storages, and the instance named ccc is used for CouchDB Storages and sentimental analysis.

☐	web	NeCTAR Ubuntu 16.04 LTS (Xenial) amd64 [v35]	172.26.38.164	uom.mse.2c9g	group22	Active		melbourne-qh2-uom	None	Running	3 days, 6 hours
☐	couchdb	NeCTAR Ubuntu 16.04 LTS (Xenial) amd64 [v35]	172.26.37.250	uom.mse.2c9g	group22	Active		melbourne-qh2-uom	None	Running	1 week, 3 days
☐	ccc	NeCTAR Ubuntu 16.04 LTS (Xenial) amd64 [v35]	172.26.37.201	uom.mse.2c9g	group22	Active		melbourne-qh2-uom	None	Running	1 week, 4 days

Figure 1 Instances Created in This Project

The Security group for the three instances are the same, which includes internal connection, ssh, default settings and HTTP. In Figure 2, it shows the detail of the security group. By using the following security group, it will allow us to connect to each server easily by using the key file group22.pem. Moreover, each instance has its own volume attached to it, and each of the instances has its own purpose.

Internal	ALLOW IPv4 tcp from Internal
	ALLOW IPv4 icmp from Internal
	ALLOW IPv6 to ::/0
	ALLOW IPv4 to 0.0.0.0/0
	ALLOW IPv4 udp from Internal
ssh	ALLOW IPv4 to 0.0.0.0/0
	ALLOW IPv6 to ::/0
	ALLOW IPv4 22/tcp from 0.0.0.0/0
default	ALLOW IPv6 from default
	ALLOW IPv6 to ::/0
	ALLOW IPv4 from default
	ALLOW IPv4 5984/tcp from 0.0.0.0/0
	ALLOW IPv4 to 0.0.0.0/0
http	ALLOW IPv4 80/tcp from 0.0.0.0/0
	ALLOW IPv4 to 0.0.0.0/0
	ALLOW IPv4 3128/tcp from 0.0.0.0/0
	ALLOW IPv4 443/tcp from 0.0.0.0/0
	ALLOW IPv6 to ::/0

Figure 2 Detail of Security Group

3.2 Twitter Harvester Application

3.2.1 Twitter API

In our project, we are using the Twitter API provided by Twitter to perform tweet harvest. Twitter provides two types of API for the user to get access to tweets in two different ways: REST Search API and Streamer API. The REST Search API allows the user to search existing tweets (up to 7 days) by keyword, geographic location(radius), date and so on, while streamer will "listen" to Twitter constantly and obtain the latest tweets with similar settings.

In this paper, we focus on the Rest Search API since we think it's a more flexible API, which we can use it to not only grab historical tweets but also grab new tweets by simply changing the parameters and run the program every day. In addition, the parameters built inside REST Search API is convenient to use in terms of avoiding repeated tweets. Here are the key parameters we are using in the program to harvest tweets:

- Keyword – search for tweets containing the specified keyword.

- Geocode – by specifying a geographic circle given coordinates center and distance as radius, user can get access to tweets only in this area. e.g. "-27.782292,133.793118,2104.3km" will restrict access to tweets generated in (mainland and some surrounding watery area of) Australia.
- Max_id – All tweets are assigned a number as their ID, the newer the tweet, the larger the number. Using max_id user can harvest tweets created before a certain tweet, which helps the user avoid getting repeated tweets and harvest tweets in the desired date and time.
- Since_id – since_id is the opposite of max_id, user can obtain tweets after since_id, also helps the user avoid repeated tweets and a tool for time control.

To prevent Twitter from information abusing, Twitter assigned a rate limit to Twitter API. The rate limit is said to vary from different users, but from our observation of our own program, we are allowed to search for 5700 - 6000 tweets per 15 minutes per API. The 15-minute window starts when you make your first request to Twitter, which means when you harvest n tweets using t minutes within 15 minutes and reach the rate limit, you will need to wait for (15 – t) minutes to make the next request or the request will be denied. The python package tweepy is well designed for REST Search API in terms of adapting the rate limit that if the wait_on_rate_limit parameter is set to be "True", the program will automatically wait when the rate limit is reached and make the request once the 15-minute window has passed.

3.2.2 Program Design

Even though the Rest Search API is convenient to use, we still have several problems to address:

1. Since we are a group of five members and each of us have a Twitter Developer account, how should we coordinate the 5 APIs we have to be efficient?
2. How to avoid harvesting duplicate tweets?
3. What kind of data should we store?
4. What should we do about retweets?

To address the first problem, we had different ideas at first, for instance, we can let them work simultaneously or take turns. Assuming all the APIs are grabbing tweets at the same speed, these two methods have similar efficiency. However, we figured out that letting APIs work at the same time can worsen the situation for Problem #2 since it's difficult to control the range of tweets they are looking at. Also, it's too easy for Twitter to find out that these 5 APIs started working and ended at a similar time, and our APIs are at risk of being banned from Twitter. Therefore, we decided to design a system for the APIs to take turns and harvest tweets, and it's working by the following steps:

a. API starts to search for tweets in time order (from new tweets to old tweets), and the start-working time is recorded for this API.
b. Keep a record of the tweet id we are looking at, only store the smallest id (id of the oldest tweet).

c. If no more tweets are returned, end the program. Otherwise keep taking the next step.
d. Once this API hit the rate limit, we observe the next API and see if it's in rest mode (by checking if it's been 15 min after it's start-working time).
e. If the next API is in rest mode, if yes, wait for the time and switch to next API; if not, switch to next API right away. Pass the (smallest tweet id - 1) to the next API as it's max_id to search tweets.
f. Return to step a.

By taking these steps, Problem #2 within each run of the program is also solved since the APIs know how to pick up where the former API left off and continue to search tweets never looked at before. To prevent obtaining duplicate tweets across runs, a running log file will be generated after each run recording the max and minimal tweet id is looked at so that we can specify max_id and/or since_id for the next run if needed.

```
Search_and_Save run in the following settings:
query= *
geocode= -37.814251,144.963169,200km
until= 2019-05-04
result_type= recent
Max turn: 3000 , for each turn 300 will be looked.
Storing in http://admin:123456@172.26.37.201:5984 in dataset: d2019-05-04_11_00

The first id we looked is 1124463647378403328
The last id we looked is 1122295695757795328
Totally 41331 tweets were stored in d2019-05-04_11_00 dataset.
```

Figure 3 Sample Run Record

For Problem #3, before we know how to process that data, we store the raw tweets in CouchDB, but once we know how, we implemented the process in the tweet harvest program, so only processed data is store. Details of the twitter data process can be found in the twitter data processing section.

For Problem #4, after discussion we decided not to include retweets from two perspectives: technical and analytical. From a technical standpoint, the tweets we are getting from Twitter is restricted, which means some tweets may contain nested retweets and we have no access to the original tweets anymore. Even though we know the original tweets, but they have a high chance being not accessible since they are too far back in time and we only have access to 7 days of tweets. From analytical standpoint, we think that retweet may not contain the true feeling of the user since many of the retweets are just created to follow a fashion (like breaking news or sports game), or just for fun. Considering both sides, we decided not to include retweets.

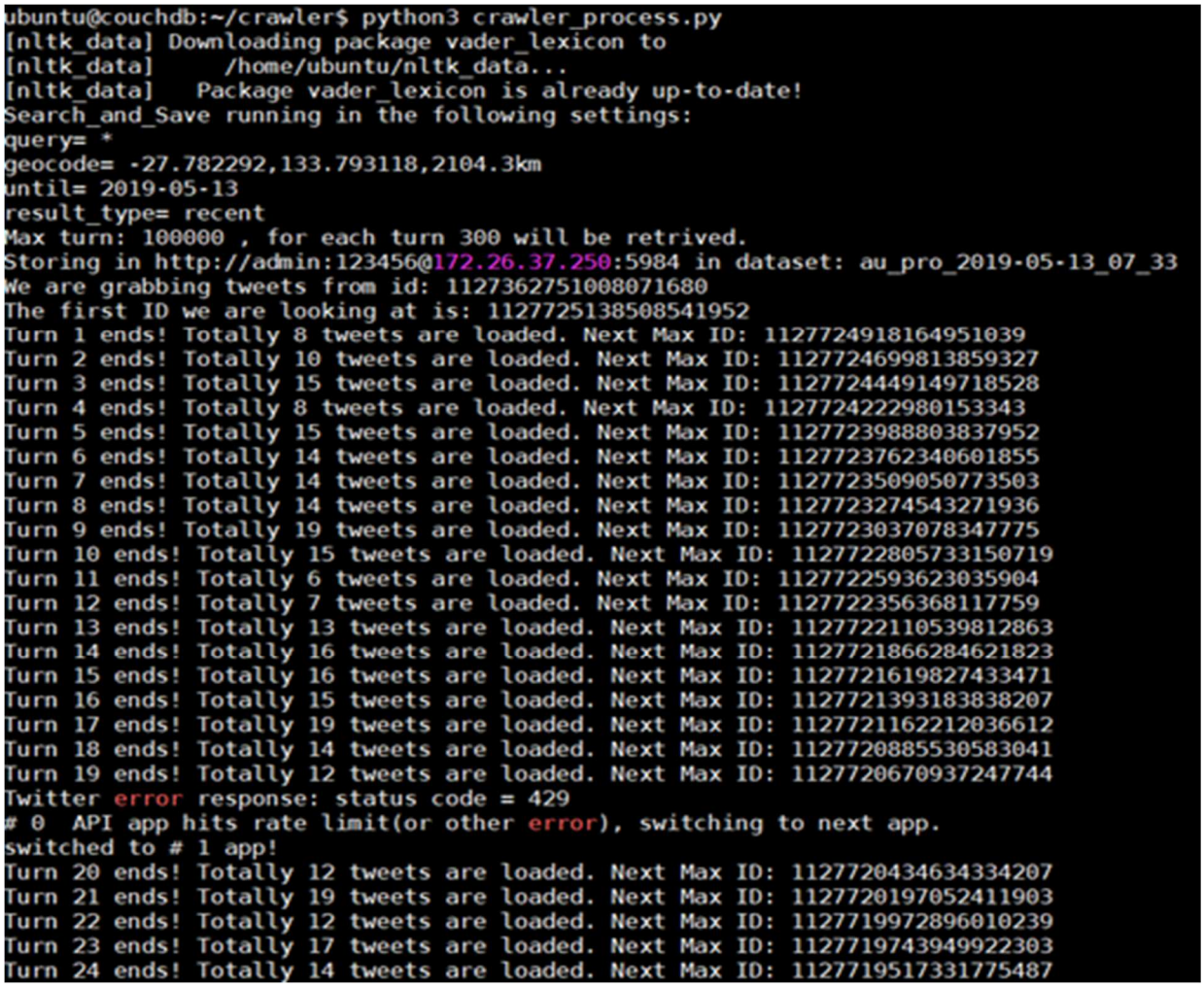

Figure 4 The Output of the Tweet Harvest Program From a Random Run

3.3 Sentimental Analysis

3.3.1 Twitter Data Processing

Raw tweets are obtained by twitter harvest application and then are preprocessed by sentiment processor to remove unneeded redundancy and extract useful information to do the analysis.

Location_finder:

Firstly, eight states in Australia are defined by regions below to classify the location of the tweets.

regions = ["New South Wales", "Victoria", "Queensland", "Western Australia", "South Australia", "Tasmania", "Northern Territory", "Australian Capital Territory"]

Attributes related to the location of tweets are “user-location”, “coordinate” and “place”. After evaluating the practically usability ,“place-full_name” is choose to compared with the state names

in regions to obtain the location of a tweet since “user-location” represents the position of the user who sent the tweet and “coordinate” is actually the same as “place” but much more difficult to process and combine with the predefined regions.

Sentiment_score:
A python API in natural language toolkit: nltk.sentiment.vader.SentimentIntensityAnalyzer which is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media is used to tell how positive or negative the text of a tweet is.

The compound score of polarity_scores is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1(most extreme negative) and +1 (most extreme positive), so it is selected to be the sentiment score of a tweet.

NLP_processor:
Originally, tweets that can be classified as one of the deadly sins data tend to identify by using keywords including synonyms of the sin, for example, if the text of a tweet contains at least one of the keywords of “pride”, it can be regarded as “pride data”. However, this method is too rough and unpolished to reflect real emotions expressed in the tweets.

A fancier way is implemented to capture the overall aspect of the deadly sins of tweets. The idea is to tag a tweet with seven scores between 0 and 1 indicating the related degree with the deadly sins. The score is calculated with similarity function between two tokens using nltk wordnet inference, and the final score of one of the deadly sins attached to a tweet is the maximum similarity score of the word in the text of the tweet with the sin word. Note that the text of a tweet is preprocessed to remove links starting with "http://", "https:// ", address sign @ along with stopwords (using nltk.corpus.stopwords.words("english")) and the tokens in the text are made lowercase and correctly spelled with “autocorrect.spell”. The code is shown as following:

```
def _ProcessKeyword(keyword, sentence):
    def _similarity(wn0, wn1):
        scores = [s0.wup_similarity(s1) for s0, s1 in itertools.product(wn0, wn1)]
        scores = [score for score in scores if score is not None]
        return max(scores, default=0.0)
    kw = nltk.corpus.wordnet.synsets(keyword)
 return max([_similarity(kw, word) for word in sentence], default=0.0)
```

In the end, a processed tag structured as the displayed below is achieved with every raw tweets, it is then stored in CouchDB database waiting for counting up and analyzing.
{"pride": 0.4459481086206047, "greed": 0.2940777906598977, "lust": 0.3230377398185581, "envy": 0.3508543198948103, "gluttony": 0.2940777906598977, "wrath":

0.28828782506677886, "sloth": 0.22258248190316293, "sentiment": -0.1027, "location": " New South Wales"}

3.3.2 AURIN Data

Since we would like to analyze situation among different states across Australia, “aggregation level” is restricted to “States and Territories” in Data Browser.
From “STE-based B37 Selected Labor Force, Education and Migration Characteristics by Sex as at 2011-08-1”, we select the percentage of unemployment men in each area:

Sex	Region	Time	% Unemployment(d)
1	1	2011	5.900
1	2	2011	5.300
1	3	2011	6.100
1	4	2011	6.000
1	5	2011	4.500
1	6	2011	7.000
1	7	2011	5.500
1	8	2011	3.800

Figure 5 Selected Labor Force, Education and Migration Characteristics by Sex as at 2011-08-1

From “STE-based T02 Selected Medians and Averages as at 2011-08-11”, we select “Median total personal income ($/weekly)”, “Median rent ($/weekly)” and “Median total family income ($/weekly)”:

ASGS 2011	Time	Median total personal income ($/wee	Median rent ($/weekly)	Median total family income ($/weekl
1	2011	561.000	300.000	1477.000
2	2011	562.000	277.000	1460.000
3	2011	584.000	300.000	1453.000
4	2011	534.000	220.000	1330.000
5	2011	664.000	300.000	1722.000
6	2011	501.000	200.000	1203.000
7	2011	733.000	224.000	1759.000
8	2011	916.000	380.000	2277.000

Figure 6 Selected Medians and Averages as at 2011-08-11
Then, data directly download from the AURIN portal need to be normalized by dividing the total item numbers to be a proportion for contrasting with sentiment proportion to get reasonable results.

3.4 Database Server

Apache CouchDB is used to store processed twitter data and to obtain statistical information from the data.

Two CouchDB servers are set up, with one is continuously replicated from the other; both run behind a Nginx load-balancer. The number of CouchDB servers can be increased or decreased based on the need. In the event of server failures, the load-balancer can redirect requests to other alive servers.
The topology is as following:

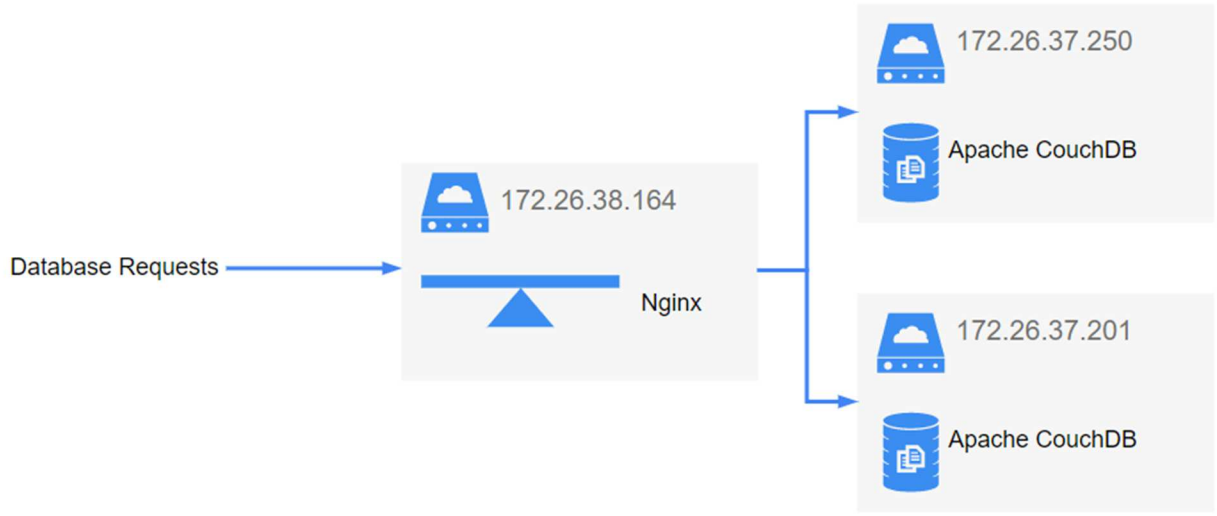

Figure 7 Topology of Database Requests

For the purpose of this assignment, only two views are needed:

1. Count - to count number of tweets of each state;

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

2. Sum - to get total sum of the scores each keyword, grouped by states.

```
function (doc)
{
        var keywords = ["sentiment", "pride", "greed", "lust", "envy",
"gluttony", "wrath", "sloth"];
        var result = {};
        keywords.forEach(function (keyword)
        {
        result[keyword] = doc[keyword];
        });
        emit(doc.location, result);
}
```

3.5 Web Server

The web server is powered by Tornado Web Server, an asynchronous event-driven web framework written in Python. It leverages non-blocking I/O and hence has only one thread per process. To better utilize the host machine's resource, two instances (one per CPU core) of the same web servers are listening to different ports. They are behind a Nginx load-balancer; therefore, the number of web server instances can be adjusted if the specification of the machine changes. The web servers can also be deployed on multiple machines provided that they are behind the same load-balancer.
Current topology is as following:

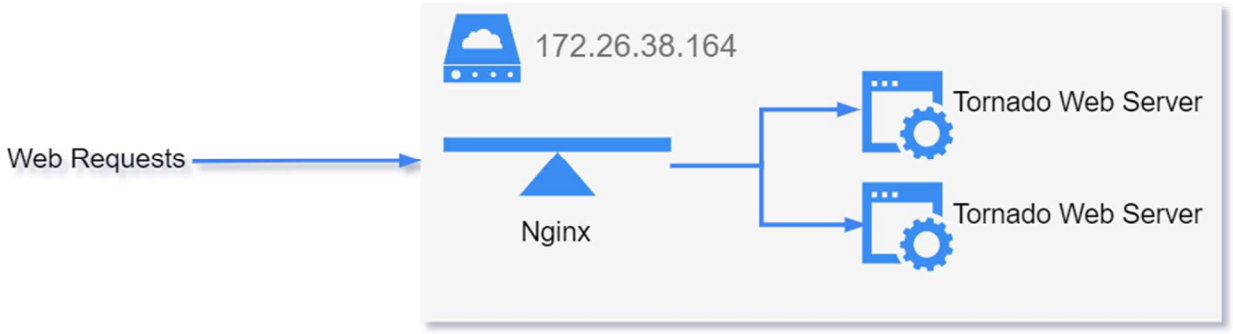

Figure 8 Topology of Web Requests

Due to the limitation of hardware resource and low expected workload, web servers and its load-balancer share the same machine with database load-balancer.

The user interface of the website is built with OpenStreetMaps, Leaflet, Bootstrap, JQuery, ChartJS and other third-party libraries.

3.6 Dynamic deployment

The dynamic deployment for our assignment is an important part because it will make it easier for all the users to set up each server. The deployment only requires the user to run a single file in a file, and it will automatically set up the server for us. Our team is using one of the instances to test the dynamic deployment in the early stage, and after the test, we make that instance become backup of our project.

3.6.1 Ansible

Ansible is a software developed for automation. It can manage several cloud server nodes simultaneously with ad-hoc commands or scripts called playbooks. It is relatively easy to manage one or two servers by directly running commands via SSH, but when it comes to cloud and cluster computing, managing thousands of servers by hand is an intractable task. Ansible helps with this by turning server commands into repeatable playbooks, then running those playbooks on all the servers with only one line of code.

In our team, Ansible workflow is as follows:

1. Provide a server list to /etc/ansible/hosts, which can be grouped into different server roles.
2. Ansible connects to each server using an SSH key.
3. Runs the playbook, which assigns "roles" for each group of servers.
4. Each "role" contains several "tasks" which are server commands written in YAML languages.
5. Running those tasks will turn the target server into our desired state, for example, with Python installed.

By using Ansible, users can manage thousands of cloud servers in a simple and agentless way.

3.6.2 Auto-Deployment

The auto-deployment system in this project enables users to create an instance, install the necessary packages and run the project program in one line of commands. The auto-deployment will reduce the cost of server maintenance by making it easier to handle and more fault-tolerant. Furthermore, in a possible future expansion of this service, auto deployment makes it as simple as possible to add more nodes to the cluster.

This auto-deployment system is based on Ansible Playbook. In this example, version 2.5 of Ansible is installed on an Ubuntu Linux, running on another NeCTAR instance.

To run the auto-deployment, all the user must run ./run-nectar.sh, and the file is shown on the figure below which is in the deployment folder.

Figure 9 Screenshot of Deployment Folder

The shell script first calls `openrc.sh`, which is a copy of the OpenStack RC File downloaded from NeCTAR Cloud to set up environment variables used in NeCTAR authentication. In figure 10, it shows the code in run-nectar.sh.

```
#!/bin/bash

. ./openrc.sh; ansible-playbook nectar.yaml
```

Figure 10 run-nectar.sh

Then, it runs the Ansible playbook `nectar.yaml`. The playbook breaks down into two parts.
In Figure 11, it shows all the roles that are contained in our playbook.

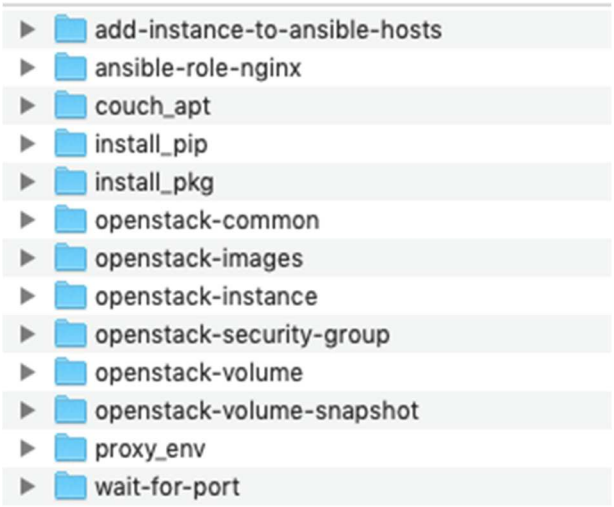

Figure 11 All Roles Contained in Our Playbook

The first part is the Instance Creation.

This part runs an Ansible play on the local computer, including several roles in order to create a new instance on NeCTAR. The local computer running this needs to be also Ubuntu Linux to work, preferably on another NeCTAR instance. Then we will explain each file in detail.

The proxy_env file
Sets up the proxy environment, to ensure the local computer can connect to the newly created NeCTAR instance.

The OpenStack-common file
Builds up some common OpenStack settings used by NeCTAR.

The OpenStack-images file
Gathers all the image settings from NeCTAR, from which we need to choose one as the base image of our new instance.

The OpenStack-volume file
Creates a volume on NeCTAR to be connected to our new instance.

The OpenStack-security-group file
Creates several security groups that allow ports 22 (SSH), 80/443 (HTTP/HTTPS), and 5984 (CouchDB) to be connected.

The OpenStack-instance file
Creates the new instance on NeCTAR, links it to the created volume, and stores the IP address of the newly created instance into an Ansible variable.

The wait-for-port file
Waits for the new instance to boot its system until its port is ready for an SSH connection.

The add-instance-to-ansible-hosts file
Adds the IP address of the newly created instance to the Ansible hosts pool.

After these tasks, Ansible goes into the second part where it installs packages in the target instance.

The second part is Installation and Running.

This part runs two Ansible plays on the instance to install and run different packages.

The first play includes the following roles:

The proxy_env file
Sets up the proxy environment on the target instance for it to connect to the internet outside the university. This is necessary because the 2c9g flavor is an internal server; therefore, we will need to add some configurations. By adding the code in figure 12 into the file /etc/environment will help us to proceed to the next steps.

```
os_environment:
  - key: http_proxy
    value : "http://wwwproxy.unimelb.edu.au:8000"
  - key: https_proxy
    value : "http://wwwproxy.unimelb.edu.au:8000"
  - key: ftp_proxy
    value : "http://wwwproxy.unimelb.edu.au:8000"
  - key: no_proxy
    value : localhost,127.0.0.1,127.0.1.1,ubuntu,ansibletest.novalocal,ansibletest
proxy_env:
  http_proxy: "http://wwwproxy.unimelb.edu.au:8000"
  https_proxy: "http://wwwproxy.unimelb.edu.au:8000"
  ftp_proxy: "http://wwwproxy.unimelb.edu.au:8000"
  no_proxy: "localhost,127.0.0.1,127.0.1.1,ubuntu,ansibletest.novalocal,ansibletest"
```

Figure 12 Helpful Codes to Add to /etc/environment

The install_pkg file
Installs several packages to the instance. The packages include:
- git
- python3-pip
- python-sharply
- numpy
- tweepy
- couchdb (python package to connect to couchDB)
- vaderSentiment
- nltk
- tornadoweb
- spacy

The data_dir
Creates a /data/ directory and mounts the attached volume.

The couch_apt file
Installs CouchDB in a cluster setting and runs it on port 5984.

The second play runs only one role, which installs Nginx on the instance as the web server backend.

The ansible-role-nginx file
This role installs Nginx on the server. It is downloaded from Ansible Galaxy as the official installation script maintained by Nginx Inc. [7]
Its variable is set up to install the open source version of Nginx and use a predefined configuration file.

The install_web file
This role installs Supervisor on the server, which is a process control program that controls two instances of Tornado web servers.

There are also Ansible playbooks `webserver.yaml` and `dataserver.yaml` for configuring a web server or a database server respectively. These servers dedicated to different purposes can orchestrate together to balance the load of computation and maintain a higher possible uptime.

4. Results and analysis

4.1 sentiment result

When comparing seven sin results in each state, tweets related to pride and envy take more than thirty percent respectively, while the number of tweets showing sloth seems to be the least in the seven deadly sins. It may due to that people tends to express themselves when feeling confident and those lazy ones are too apathetic to send tweets.

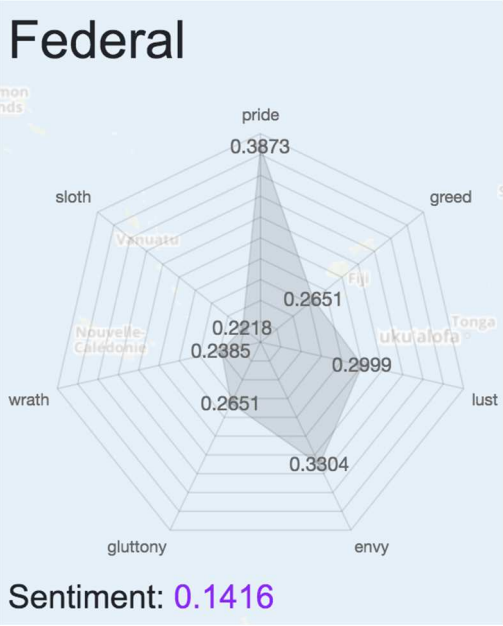

Figure 13 Results of Federal Region in Display

When compare one particular sin in eight different states and territories, there seem no much differences with the proportion it takes part in the seven sins. Although data from different areas can be sorted to have an order, the divergence is not larger than five percent.

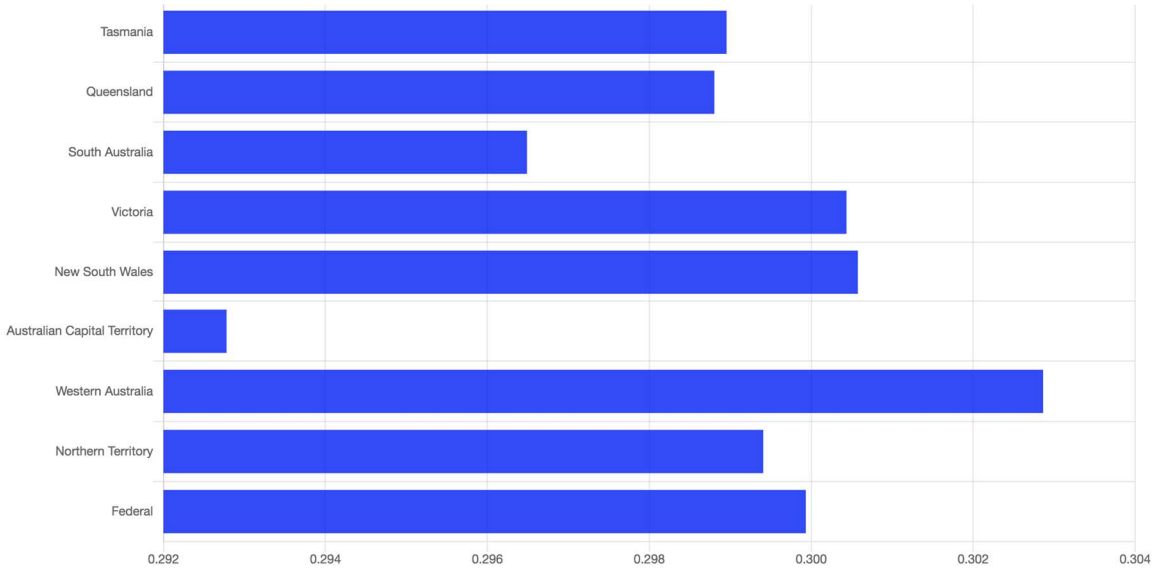

Figure 14 Comparison of Scores of Lust Across Regions

The sentiment score indicates the overall positive attitude. From the chart below, people living in the capital are more positive and happier than the other states. On the contrary, life in Tasmania and South Australia may be harder owing to the desolation and wildness, so people find it difficult to feel satisfied and delighted and tend to have a negative emotion.

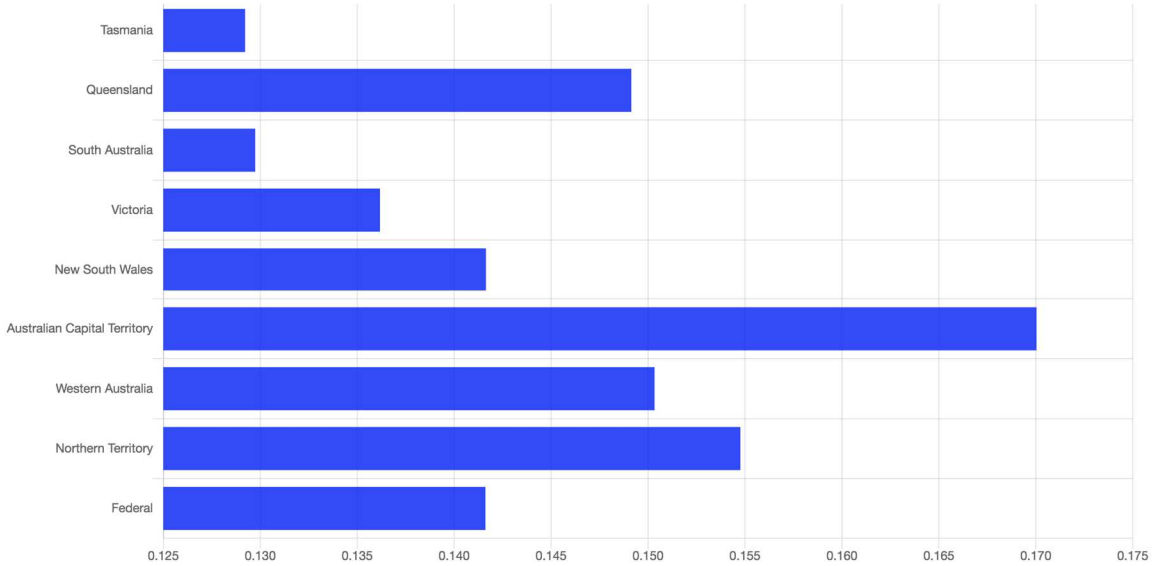

Figure 15 Sentiment Scores of All Regions

4.2 Scenarios Assumption and Discussion

4.2.1. The relation between wrath and envy with unemployment

When people especially men could not find a job and stay unemployed, they tend to feel fierce anger and deeply resentful to the society and more likely to write tweets containing wrathful emotion. In figure 16, the blue line is the unemployment percentage of each state in Australia; the red line stands for the envy percentages of each state in Australia and green line stands for the wrath percentages of each state in Australia. As we can see, in most of the state, the percentage of unemployment is close to the percentage of both envy and wrath. In New South Wale, the difference is about three percent, and in Western Australia, the difference is about one percent. Only in Tasmania, the percentage is higher; it is about six percent. Therefore, we conclude that when people are unemployed, it is likely that they will post tweets containing wrathful emotion, and possible jealous tweets.

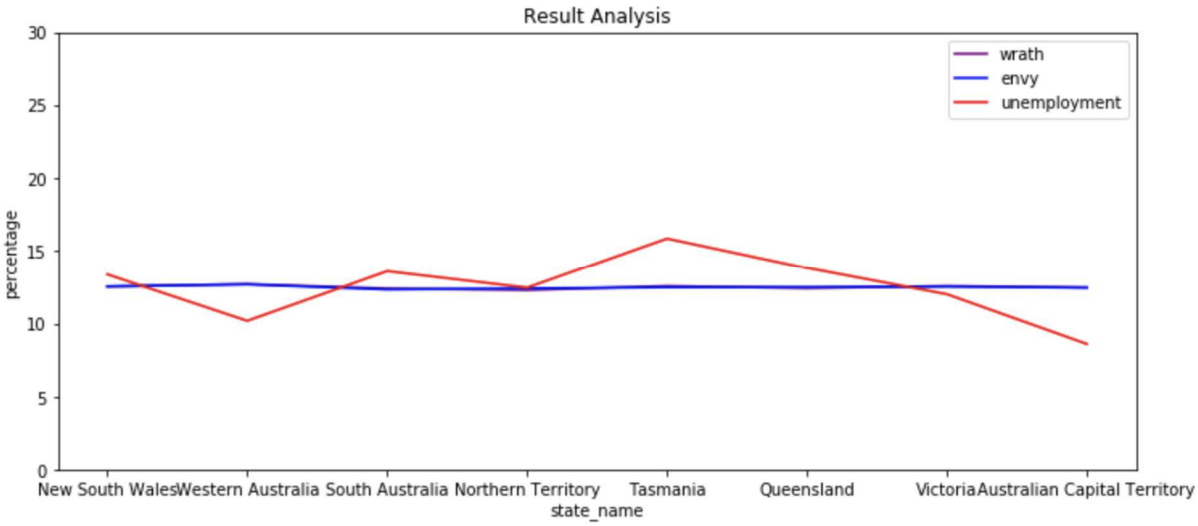

Figure 16 Relation Between Wrath and Envy with Unemployment

4.2.2. The relation between sloth and pride with the person and family income

When the family income is low, it is usually assumed that the family members are lazy, so we make this assumption. Also, when the family income is high, we think this family is more likely to be pride.

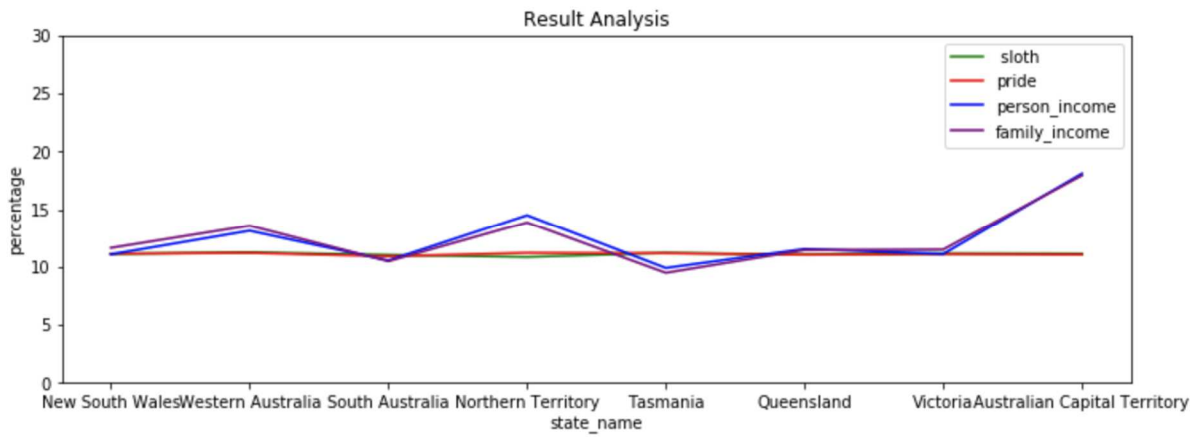

Figure 17 Relation Between Sloth and Pride with the Person and Family Income

From figure 17, we can see that the green line stands for sloth percentage and the red stands for the pride percentage in Australia. Also, the personal income percentage is the blue line, and family income is the purple line. As we can see that in New South Wales, South Australia, Queensland and Victoria, it is very likely that a person with low income will send tweets with sloth as the lines and the percentage are very close to each other. As for the rest of the states in Australia, it is less likely that the resident will post tweets about low income because of their laziness.

As for the people that are making a lot of money, we can see that in New South Wales, South Australia, Queensland and Victoria, it is very likely that a person with high income will send tweets with pride as the lines and the percentage are very close to each other. As for the rest of the states in Australia, it is less likely that the resident will post tweets about high income and show off their pride.

4.2.3. The relation between wrath with rent

When people are wrath, they are more likely to be angry because of something in their lives, especially everyday cost. With no doubt, for those who do not have their own house, rent would take a large part of their expenses. Therefore, our group assume that wrath is related to the rent of their houses or apartment. The reason is that when we have high rent, it is more likely that people will be angry because it cost them more money, and vice versa.

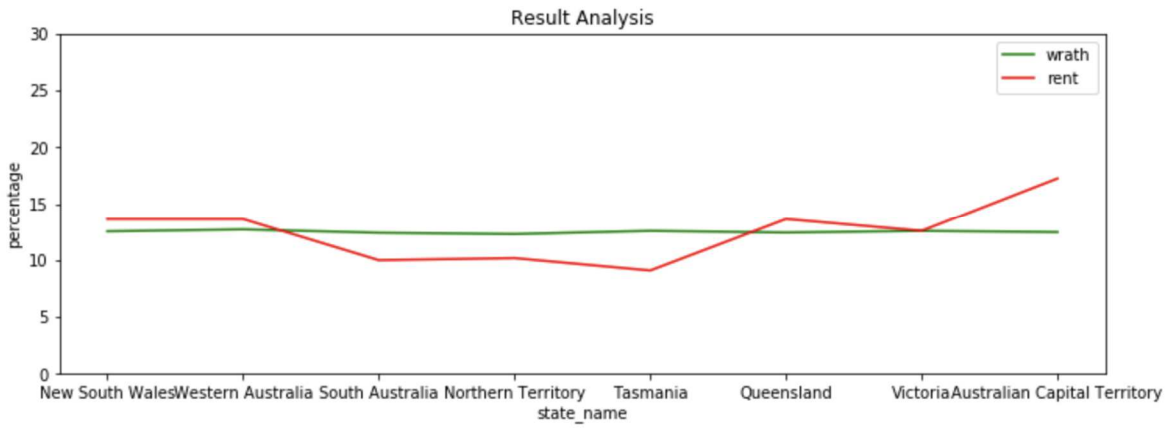

Figure 18 Relation Between Wrath with Rent

From the figure, we can see that for every state in Australia; wrath is associated with rent to some extent. The percentages of these two lines in each state are somewhere close, which means that wrath is somehow related to the rent.

5. Pros and Cons of Selected Components

5.1 NeCTAR

The greatest advantage of NeCTAR is convenience, allowing team members to collaborate in the same cloud. In addition, the cloud is especially helpful processing large or time-consuming files, and they are rarely interrupted.

There are two disadvantages of NeCTAR. The first one is that its performance is not stable; sometimes the system will provide a slow response. The second one is the slow response time when creating a new security group, a new instance, and a new volume.

5.2 Twitter Harvester Application

The advantages of the Twitter Harvester application are that it will help us to get the twitter data through twitter APIs, and this would be more convenient because it will process all the tweets for us and give us the one that is ready to be used. More importantly, it saves us time to look at every tweet.

The disadvantage is that the program will take a long time to run and get limited results. Twitter APIs imposes some limitations. For example, it will force one of our accounts to rest after a certain amount of time, and it only allows us to get a certain number of tweets per minute.

Also, we may need more sophisticated method to deal with retweets, so that the information can be used in our analysis.

5.3 Sentimental Analysis

5.3.1 Location

Here, tweets with attribute “coordinate” or “place” is None are ignored through the harvesting process. Nonetheless, those tweets account for a large part, which makes the component of the tweets used in the project is not the same as those in the real world. There may be some influences on the statistics obtained by processing tweets and consequently affect the analysis and the Scenario demonstration.

5.3.2 Sentiment Score

The package used to get the sentiment score is VADER which has some overwhelming advantage over other traditional methods [8]:

1. It works exceedingly well on social media type text, which is perfect for this project with Twitter data.
2. It does not require any training data so it can be used directly without preprocessing or preconstruction.
3. It runs fast and does not suffer from a speed performance tradeoff extremely.

Drawbacks may lay in that the outcome is only a score reflecting the degree of positive, neutral and negative of the text but not the exact emotion or attitude which can be correlated with seven deadly sins directly.

5.3.3 Similarity Score

As for the similarity part, nltk package is chosen for its fast speed, but some other package like spacy could get a more reliable score which is needed to be verified in the future. The maximum similarity score among all the similarity scores between each word in a text and the keyword is used to represent the relation scale of that sin. The more reasonable idea is to do the weighted average among all the similarity scores with weights to be the importance of the word is in a text, for instance, a verb may take more advantages of influence than a noun. However, how to decide the weights attached to different words is a challenge that is of interest to do research.

5.4 Database Server

Apache CouchDB is one of the best NoSQL databases, and hence the major advantages are its flexible document model and highly scalable architecture. The JSON based document model naturally maps object-oriented structured into its document, and the distributed architecture makes

it easy to scale. Also, CouchDB can be easily replicated into multiple nodes so that downtime and query response time can be significantly reduced.

The major disadvantage is that the JSON-based data in CouchDB may reach very large, and hence the processing overhead may be high. In addition, the JavaScript-based views may also be a bottleneck as its overhead of data serialization and interop between native and managed code.

5.5 Web Server

As an asynchronous, event-driven web server, the most significant advantage is its ability to handle a large number of concurrent connections. The web server needs a query database server before answering an HTTP request, and the database query may take a significant amount of time. As a result, there is a potentially large number of socket connections while the comparably little amount of computational workload. In addition, the web server runs behind a load-balancer so that it is easy to add or remove instances to meet the requirements of actual workloads. Last by not least, the web server is only granted read-only access to the database, which reduces security risks.

The major disadvantage is the long development cycle. There are much fewer libraries available comparing to traditional synchronous web servers so that many modules are developed in-house. For example, the web server implements its own helpers on top of CouchDB's raw REST API to communicate with the database. During the process of this assignment, a licensable asynchronous Python library that can fit into Tornado Web Server is unknown.

5.6 Deployment

The advantages of deployment are that it will save up time and it makes the project more efficient because it does all the setup in one file. The deployment will set up the instances, configure the instances and the needed packages in the server.

The disadvantage of this approach is that it takes much longer, and that time is out of our control. Another disadvantage is that we need to do a lot of testing when writing this code.

References

[1]"Home - Nectar", *Nectar*, 2019. [Online]. Available: https://nectar.org.au. [Accessed: 09- May- 2019].

[2]R. Ansible, "IT Automation with Ansible", *Ansible.com*, 2019. [Online]. Available: https://www.ansible.com/overview/it-automation. [Accessed: 09- May- 2019].

[3]"1. Introduction — Apache CouchDB® 2.3 Documentation", *Docs.couchdb.org*, 2019. [Online]. Available: https://docs.couchdb.org/en/master/intro/index.html. [Accessed: 09- May- 2019].

[4]"Apache CouchDB", *Couchdb.apache.org*, 2019. [Online]. Available: http://couchdb.apache.org/. [Accessed: 09- May- 2019].

[5]"AAF Discovery Service", *Portal.aurin.org.au*, 2019. [Online]. Available: https://portal.aurin.org.au. [Accessed: 14- May- 2019].

[6]"NLTK Book", *Nltk.org*, 2019. [Online]. Available: https://www.nltk.org/book/. [Accessed: 14- May- 2019].

[7]"nginxinc/ansible-role-nginx", *GitHub*, 2019. [Online]. Available: https://github.com/nginxinc/ansible-role-nginx. [Accessed: 14- May- 2019].

[8]"Simplifying Sentiment Analysis using VADER in Python (on Social Media Text)", *Medium*, 2019. [Online]. Available: https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f. [Accessed: 14- May- 2019].

Appendix A GitHub link

https://github.com/Danyang1123/CCC_1

Appendix B YouTube Link

https://youtu.be/JbnL3K8EIe0