COMP90024 Project 2: The Deadly Sins
Analysis of Tweets in Australia

Team 35

YE YANG	**yey11@student.unimelb.edu.au**
ZHAOHUI HOU	**zhaohui.hou@student.unimelb.edu.au**
HAOYU ZHANG	**haoyu.zhang@student.unimelb.edu.au**
YIMING XU	**yimingx2@student.unimelb.edu.au**
WENKANG ZHU	**wenkang.zhu@student.unimelb.edu.au**

1. Introduction

The seven deadly sins, also known as “the capital vice or cardinal sins”, is “a group and classification of vices with Christian teachings.” Behavior or habits are “classified under this category if they directly give birth to other immortalities.” Based on the standard list, they are “pride, greed, lust, envy, gluttony, wrath and sloth’, and these sins are considered to be “abuses or excessive versions of one’s natural facilities or passions”.

Nowadays, people are getting more used to express their own emotions on the Social Media platforms, such as Twitter and Instagram, by posting text or emojis. In this project, we focus on exploring the relationship between the deadly sin of Wrath and three indicator data of work pace level of each geographical area. Our hypothesis is that more deadly sin of wrath arises accompanied with fast-paced and stressful work of the society. Three attributes from different data sets are chosen to indicate or reflect the work pace level of each geographical area under SLA and SA2 standard, which are annual median personal income, unemployment rate and the average age of the earners. We suppose that personal income and unemployment would demonstrate positive correlation to the angry value, whereas the average age of earners is likely to show the negative one.

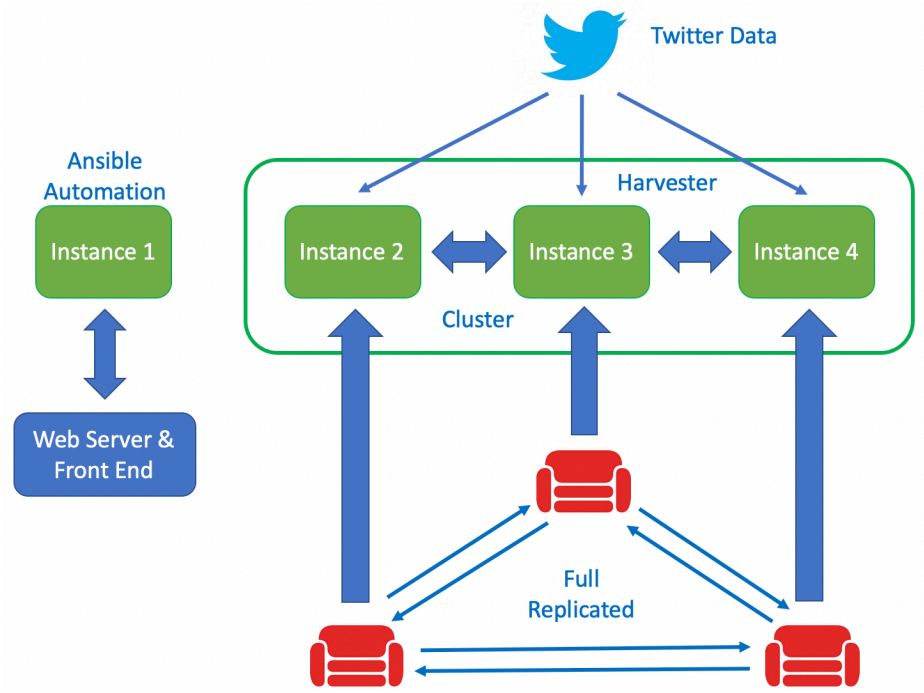
A harvester program is used to gather the tweets data from 8 states and over 60 cities across Australia and used Text Sentiment Analysis tools based on Neural Networks to evaluate the angry value of processed tweets. Then, the integrated city-based angry value tables are joined with the AURIN data tables to make further comparisons.

Afterwards, we mapped the correlation between the angry value gathered from tweets and three indicator data collected through the AURIN’s website through the linear regression model, in order to test and verify our hypothesis.

2. Environment and Cloud Architecture

2.1 Cloud Architecture

There are 4 instances implemented on the NeCTAR cloud. All instances are ‘uom.mse.2c9g’ size. Three instances are attached to CouchDB and clustered together meanwhile these three instances are also implemented by twitter harvester which are attached to more than 60 GB Volume. However, only the first instance is attached to 5 GB Volume which is used for automation. The NeCTAR architecture is as shown below.



With this architecture, more tweets could be gathered with high efficiency. In addition, all the three CouchDBs are full replicated in terms of safer and backup. The main advantages of this structure are its robustness and scalability. When we need more servers to process more Twitter data, we can easily extend its cluster by PUT method. Although there can be some problems when one of the CouchDB server unavailable, due to the fully replicated database, we will never lose data only of all of these servers break down.

2.2 NeCTAR Cloud Service

NeCTAR is an Australian cloud-computing platform that provides research data, analysis software and research tools and data to researchers across the country, university students/professors and contemporaries across the world, resulting into communal information distribution and eventually enhancing the social conditions of the country.

Pros:

- Reliable - The automation of the whole infrastructure in this project is provided on the NeCTAR cloud. It is flexible to take snapshots of our volumes and transfer volumes between projects.
- User Interface – NeCTAR has a very friendly user interface, which allows users to make any changes on the cloud in an easy way such as create instances, delete instances, edit volume.

- Secure: As NeCTAR (Ubuntu) uses private and public keys to setup connections, it is a secure system to conduct a research.

Cons:

- Processing speed: The bandwidth is limited, with the data loading increasing, the data transfer rate was too slow.
- Node Failure: When one instance is suddenly down and it is useless to reboot it. We have to delete the broken one and create a new one or use snapshot to copy image.

2.3 CouchDB Environment

Instance are created in flavor of uom.mse.2c9g and Unimelb internal network gh2-uom is chosen due to the fact that nectar IPs are run out. Each instance is setup to use Unimelb internal proxy. Every time we connect to instance, we have to connect Uni Wi-Fi or connect to Uni VPN. Few lines of code are added to environment for proxy.

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games"
http_proxy="http://wwwproxy.unimelb.edu.au:8000"
https_proxy="http://wwwproxy.unimelb.edu.au:8000"
ftp_proxy="http://wwwproxy.unimelb.edu.au:8000"
```

However, there is still some issue regarding instance access external link (e.g. curl couchdb on the other instance). No proxy code is added to environment script so that we can use Uni IP address and our instance can access external network.

```
no_proxy=localhost,127.0.0.1,127.0.1.1,ubuntu,ccc_2.novalocal,ccc_2
~
```

CouchDB, Pyhon and related packages are installed on each instance so that it can run Python script and harvest Tweets.

An ansible playbook is implemented on the first instance to deploy automation, which we do webpage, CouchDB installation and environment setup through.

CouchDB is implemented on three instances which is used as a cluster. Therefore, the data in CouchDB is replicated in case something usual would happen to one of them. If it happens, we could just detach the volume and attached it to a new instance.

In order to get NeCTAR work, we need a proxy setting added to /etc/environment:

```
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,ccc_2.novalocal,ccc_2
```

And the next step is to set up the CouchDB of each cluster. in Ubuntu, the location of CouchDB is in /opt/couchdb/ and we need set the defautle username and password by:

```
sed -i "s/;admin = mysecretpassword/admin = nmsl" local.ini
```

and ip address by:

```
sed -i 's/127.0.0.1/0.0.0.0/g' default.ini
```

Finally, we should build a cluster use PUT method:

```
curl -X PUT "http://admin:nmsl@172.26.38.7:5986/_nodes/couchdb@172.26.37.253" -d {}
```

Then the CouchDB server is finished via ansible script.

For Harvest, we use the tweepy API to store data into CouchDB.

However, for Data analysis, we must guarantee we can process the latest data in CouchDB, so we use crontab here:

```
echo '0 * * * * python3 /LinearRegression.py' >> /etc/crontab
```

In flask server, we can obtain the latest result of data analysis via couchdb python API. The data process become a closed loop so that the front end can show the real-time result of Twitter analysis

3. Data Collection

3.1 Harvesting

In this project, we use the python-based web harvester to collect and gather the data across 7 provinces and over 60 cities across Australia. In order to run this file, three additional python packages need to be installed and imported, which are tweepy, couchdb and urllib3.

```
import tweepy
import couchdb
import json
import math
from urllib3.exceptions import ProtocolError
```

In order to get the access to the tweets data, consumer key and the access token need to be obtained through applying for a developer API, and inserted into the python file in the following way. In order to collect more tweets data in a limited time, each team member applied and obtained their own consumer key and tokens.

```
consumer_key = 'Gu3nJ84XPIsWhjWYoOlWUG8qx'
consumer_secret = 'QlBCL3kXYg4aX6bQX1F3Vd8SwpnNafRSmfvw5jrm7Ly1nroGZJ'

access_token = '1119148424387936256-Xx30oWN4yKOhDGhQTg1MWcsCwJUmlP'
access_token_secret = 'dxmuQzKDdLAgseUmJNZvxo9QPETzSOa7Yf8sidLNB3Km0'
```

Besides, the harvesting file has to know which geographical bounding box it is searching tweets from, by inputting the corresponding coordinates. For example, if I am trying to harvesting the tweets from Melbourne, I am using the following coordinates, "145.299928, -36.539539, 146.455767, -36.272276". For each sub-bounding box, I only need to input the coordinates of the bottom left point and top right point. The screenshot below is where I placed the coordinates of the sub bounding boxes.

```
##sub_bbox = create_sub_bbox([140.946597, -39.138210, 147.698657, -35.936027], num_harvester)
sub_bbox = create_sub_bbox([138.494335, -35.009266, 138.78049, -34.785160], num_harvester)
```

Our methodology of harvesting the tweets data is separate Australia into 8 bounding boxes, each bounding box approximate a single province. We transmitted the harvesting python file to three instances through the scp commands and run those files simultaneously, and stored the captured tweets into the couchDB installed on each instance. The picture below demonstrates the address of the couchDB that I stored my data to, the user name and password of the admin account need to be inserted before the ip address.

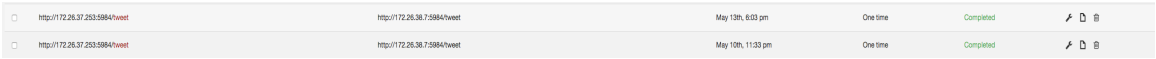
```
db_address = "http://admin:nmsl@172.26.37.253:5984/tweet/"
db_server = couchdb.Server(db_address)
```

When we normally harvesting the tweets data, amount of duplicated data may be captured, and most tweets don't come with the coordinates of the accurate posting location, but always having the bounding box coordinates besides, using if clause in the python file can also help filtering the tweet data. For example, if clause in the following picture help to capture only the tweets with its accurate coordinates. However, adding more filters may slow down the harvesting speed of the tweets.

```
                user_tweet = status._json
                if user_tweet["doc"]["tweet"]["coordinates"]["coordinates"] is not None:
```

Furthermore, when harvesting of tweet data is accomplished, we used the replication function of the couchDB to transmit the tweets data from one couchDB to another, in order to further process and analyze the data. It should be noticed that the replication function can be executed

for many times, and it is able to identify the duplicated data that has already been transited before.



3.2 AURIN Data

Due to our hypothesis, three indicator data of the city's work pace level need to be collected through AURIN's website. However, AURIN's website provides data based on different spatial standard, such as SLA and SA2. Different standard split the areas based on different spatial unit. SA2 standard split Australia into 2292 sub areas, while SLA split Australia into 1418 sub areas instead. During this project, we prefer the SA2 standard, as SLA provides a more fuzzy and complicated form of the sla area name. More data cleansing work need to be done, if we use the sla name as a key to join the AURIN data frame with the angry value table. Among three indicator data, we download the median personal income data and the median age of the earners under the SA2 standard, and download the unemployment data under the SLA standard (because there is not available version of data under SA2 standard).

After the AURIN data frame is downloaded, additional python packages include pandas, csv and openpxyl are used to process the data, so that it could be used to join with the angry value data and make comparisons. Firstly, we transfer the AURIN data from csv format to excel format. Then, the sa2 and sla area names are processed into city names and added into a new column, which is expected to be consistent with the city names in the processed angry value table. This step will output the excel file with the new city name column. Related code can be seen from the screenshot below.

```
def CsvtoXlsx(csvfile):
    wb = Workbook()
    ws = wb.active
    with open(csvfile, "r") as f:
        for row in csv.reader(f):
            ws.append(row)
    wb.save("output.xlsx")

def extract2(inpath, formatting_info=True):
    data = xlrd.open_workbook(inpath)
    wb = copy(data)
    ws = wb.get_sheet(0)

    table = data.sheets()[0]  # 选定表
    nrows = table.nrows  # 获取行号
    ncols = table.ncols  # 获取列号

    for i in range(1, nrows):  # 第0行为表头
        alldata = table.row_values(i)  # 循环输出excel表中每一行，即所有数据
        result = alldata[0]  # 取出表中第二列数据
        ##print(result)
        b = result.split("-")
        ws.write(i, 3, b[0])
        print(b[0])
    ws.write(0, 3, "CITY")
    wb.save(inpath)
```

4. Database

4.1 CouchDB

CouchDB is the database service running on our instances. It is designed for the web architecture and convenient for big data analysis since it provides powerful MapReduce function. Fauxton interface is handy and powerful, which make our life easier.

Tweet harvesters are running on all of our instances. All the harvested Tweets are directly imported into the CouchDB. It is worth to mention the writing data and reading data from CouchDB simultaneously will not lead to any conflict and error message. Multi-version concurrency control are used in data management system of CouchDB. In this case we do not have to stop the harvester on instance and read the data for analysis.

We use Python package CouchDB to connect to CouchDB server and import Tweets data into Python. Also, the results of analysis are send back to CouchDB database so that they are accessible for the front end.

Fauxton interface provides a better way to manage CouchDB set up and check for data updates. This interface can directly access through browser. It gives us an easier option to cluster our databases. MapReduce function also can be done on the interface. MapReduce's results can be checked immediately.

The main issue we encountered with CouchDB is that it does not have many tutorials and resources for learning unlike MongoDB. Official documents are not enough sometimes when we have the problem regarding database operations and managements, which takes us some time to figure out the problem.

4.2 Cloud Database Architecture

Our databases are not for business use since it is open-source and free. Stability is not guaranteed. To insure high availability, full replication is chosen as our database architecture. Since three harvester are running on 3 databases instances, data updating happen all the time on each CouchDB server. Continuous replication tasks among three CouchDB servers are running all the time to make sure three databases completely the same eventually. Partition method is not considered here due to the fact that it is impossible for us to harvest more than 20GB data within 5 weeks. Thus, storage is not a concern here.

Source	Target	Start Time	Type	State	Actions
http://172.26.37.253:5984/tweet	http://172.26.37.227:5984/tweet	May 14th, 9:05 pm	Continuous	Running	
http://172.26.37.253:5984/tweet	http://172.26.38.7:5984/tweet	May 14th, 6:10 pm	Continuous	Running	
http://172.26.37.253:5984/tweet	http://172.26.38.7:5984/tweet	May 13th, 6:03 pm	One time	Completed	

One of the CouchDB server is responsible for communication with front end server. Cross-Origin Sharing (CORS) is open so that result data are accessible for the front end server. It is opened for all domain since we our website does not have a static domain. The results of the analysis will sent to this CouchDB server in geojson format

5. Data Processing

5.1 CouchDB MapReduce

Before we start our analysis, data pre-process is quite necessary since there are many useless information in our data. All we need for sentiment analysis are text and geolocation of each tweet. A lot more time will be spent if we load all data into Python for pre-process and data cleaning. Also, it seems impossible to load all data at once into the memory. A better way, such as load tweet line by line, must be considered so that our memory won't explode. To make our life easier, we used Map function of Couchdb to solve this problem. A view is created that only contain texts as value and geolocation information as key of each tweet. In this case, less time will be consumed for our data transmission and memory is not a problem anymore.


```
"key": {
 "attributes": {},
 "name": "Aireys Inlet - Fairhaven",
 "bounding_box": {
  "type": "Polygon",
  "coordinates": [
   [
    [
     144.055114,
     -38.470786
    ],
    [
     144.055114,
     -38.448092
    ],
    [
     144.119285,
     -38.448092
    ],
    [
     144.119285,
     -38.470786
    ]
   ]
  ]
 },
 "place_type": "city",
 "id": "0051ec552e1dcc26",
 "url": "https://api.twitter.com/1.1/geo/id/0051ec552e1dcc26.json",
 "full_name": "Aireys Inlet - Fairhaven, Victoria",
 "country_code": "AU",
 "country": "Australia"
},
"value": "G63 #amg #g63 #rollin #feedininstagram #familywheels @ Split Point Lighthouse https://t.co/26nQzhrhZ7",
```

5.2 Data Cleaning

CouchDB server are directed connected to Python. CouchDB view data are imported for sentiment analysis. Part of the data cleaning are already done by Map function in CouchDB, which saved us some time on Python. However, after we loaded mapped data into Python, there are some Tweets harvested multiple times. Duplicated data are filtered in this step.

Besides duplicated tweets problem, URLs and username at the beginning may also influence our sentiment analysis later. Regular expression is used here to remove username and URLs in the Tweet Text.

It is noticed that emoji in the Tweet text are not removed since that emoji also contains quiet a lot of information related to Twitter user's emotion. Languages other than English are not filtered. There is higher possibility that those non-English Tweets are from tourists rather than local residents, but we still keep them since Australia is an immigration country and some people would publish non-English tweets. Furthermore, the package we choose for sentiment analysis can handle emoji and non-English text and turn them into different value of emotions.

6. Data Analysis

After the tweets have been harvested, they are stored into CouchDB database for later analysis. All data are in JSON format and each document in database is a Tweet from Twitter user. Data from CouchDB server are passed to Python for sentiment analysis and correlation with AURIN.

6.1 Sentiment analysis

The main purpose of our sentiment analysis is to measure the "angry value" of each city across Australia by processing those Tweets we have. Our topic and hypothesis is simple: are people easier to be angry with faster work tempo? People live in bigger cities with higher population are tended to have a faster life pace and higher chance of anxiety and stress. How can our sentiment analysis result have related to local data such as median annual income, median age of population and unemployment rate?

First of all, a sentiment value must be calculated for each Tweet. TextBlob is considered as our sentimental analysis tool initially. However, it only returns polarity and subjectivity values, which means only show us whether the text is positive or negative. More pre-process and post-process are needed for TextBlob. A Python package named Reactionrnn is used for text

analysis. This package is built on TensorFlow/Keras and a pre-trained Recurrent neural network is used to predict the proportionate reactions for a given text. This RNN model are trained by Facebook Text, which is very similar to the Tweets we have. Two of the main reasons that this package is chose is because it can predict emotion from emoji and also non-English texts since we found that part of our harvested Tweets are non-English contents. Emoji contains very important information regarding emotion and we should not ignore them.

```
> react.predict("When the soup is too hot 😂😂😂")
[('haha', 0.8568), ('love', 0.1376), ('wow', 0.0056), ('sad', 0.0), ('angry', 0.0)]
```

The predict function accept text and emoji as input. Note that the result is not probabilities of each emotion. It is proportionate emotion values from the prediction of each kind of emotion in the text. This is reasonable since a Tweet can contain more than one emotions. Angry proportion of each Tweet can be calculated immediately, which is very handy for our topic to identified angry proportion of each city. Summation of angry proportions/values are divided by number of Tweets from the same city as the total angry value of each city. Results here are all very small number less than 0.01 since the proportion of angry Tweets in each city are very small. To make our result more clear and obvious so that people can tell the difference from it, we normalized the angry values of each city. Normalization also leads a better front end visualization and presentation.

6.2 Result Normalization

In the sentiment analysis, the angry value of each Tweet is calculated. Sum of angry values are divided by number of tweets in each city as the angry value of the city. However, we found that for some small city, angry value is easy to leverage up by few “angry” Tweets due to the fact we harvest less Tweets in small place. To solve this problem, a punish term are given in the angry value calculation equation for the cities and states. For the city and state (such as North Territory) with smaller amount of data, angry value is punished and adjusted based on number of Tweets in this location to offset some of leveraging problem caused by outlier angry Tweets. Angry values are normalized by min-max method between 0 to 100 for a more effective demonstration at the front-end.

angry_value	full_name	coordinates	city	state
42.224244	Canberra, Australian Capital Territory	[[[148.995922, -35.48026], [148.995922, -35.14...	Canberra	Australian Capital Territory
55.031650	Sydney, New South Wales	[[[150.520929, -34.118347], [150.520929, -33.5...	Sydney	New South Wales
28.532953	Wellington, New South Wales	[[[148.928678, -32.577662], [148.928678, -32.5...	Wellington	New South Wales
35.523818	Goulburn, New South Wales	[[[149.654329, -34.793823], [149.654329, -34.6...	Goulburn	New South Wales
1.000000	Baradine, New South Wales	[[[149.049383, -30.959807], [149.049383, -30.9...	Baradine	New South Wales

After the analysis finished, data will be converted to geojson format and send to CouchDB server for the front-end visualization and presentation.

6.3 Correlation with AURIN Data:

Result of sentiment are combined with AURIN data for further analysis and visualization. AURIN data are imported into Python for data cleaning since AURIN data have different format. Part of minor data cleaning work are done manually on EXCEL. AURIN data are merged into sentiment data frame according to city for later analysis.

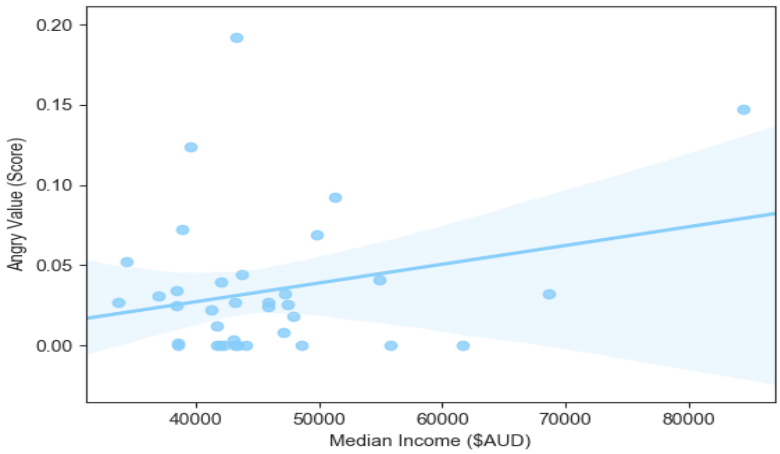
In order to test and verify the hypothesis we made earlier, we intended to evaluate the relationship between the angry value gathered from tweets and three indicator data of city's work pace collected through the AURIN's website. So, after the AURIN data is downloaded and being processed from the previous step I use pandas' merge function to join the angry value table with each indicator data excel file, based on the city name as a key. One issue should be noticed is that even sa2 methods may provide multiple data value for the same city. For example, AURIN SA2 includes the data of Melbourne East, Melbourne West and even Melbourne Aiport, and all of their processed city name is Melbourne. In our case, we only choose one of data from the same city name to do the data join with anger value. Afterwards, the joint table will be save to local directory. In order to further analyze the correlation between anger value and the indicator data, Matplotlib package are then used to generate the linear regression scatter plot. The mapping of each city on the plot can give us a clearer view of the correlation. The corresponding codes are attached below:

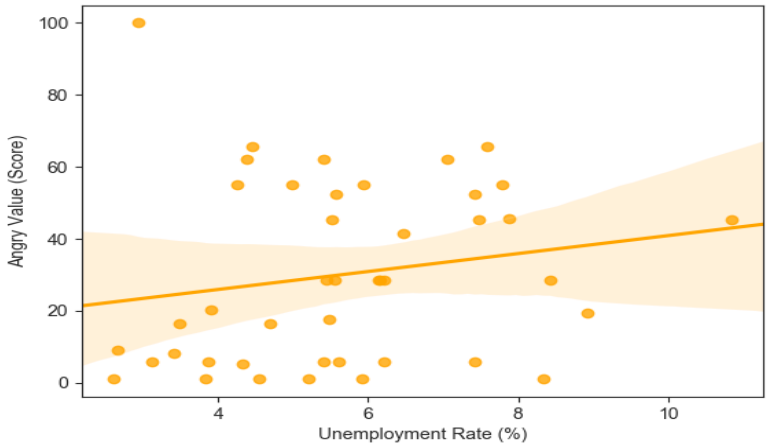
```
def joinTheTable(f1, f2):
    file1 = pd.read_excel(f1)
    file2 = pd.read_excel(f2)
    table = file1.merge(file2, on="CITY")


    table.to_csv("111.csv")


    print(file1)
    print(file2)
    print(table)
```

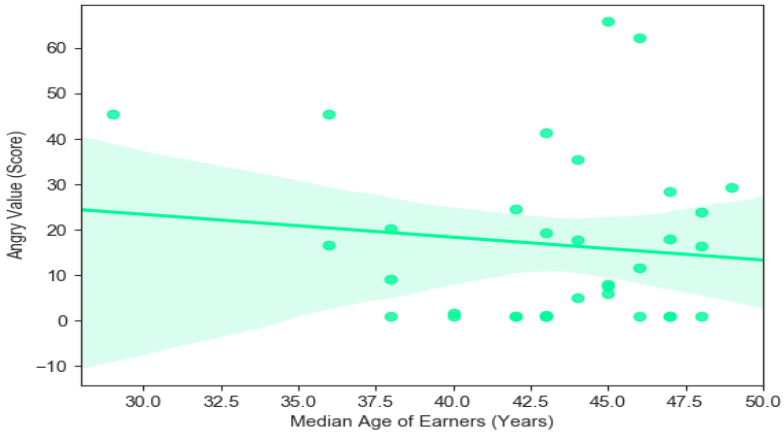
```
joinTheTable(angryFile, incomeFile)

df = pd.read_csv("/Users/zhaohuihou/Desktop/linearRegressionCode/111.csv", index_col=0)

#df.plot.scatter(x='median_aud', y="angry_value", color="b")

sns.set_style("white")
sns.set_style("ticks")

sns.regplot(x='median_aud', y='angry_value', color="lightskyblue", data=df)

plt.xlabel("Median Income ($AUD)")
plt.ylabel("Angry Value (Score)")


plt.savefig("newincome.png")
plt.show()
```

6.4 Linear Regression

We chose the liner regression model to analyze the correlation the angry value and the AURIN indicator data because it is quite intuitive to use and understand, and can easily explore the nature of the relationship between these variables. The mapping of each city on the plot can give us a clearer view of the correlation.



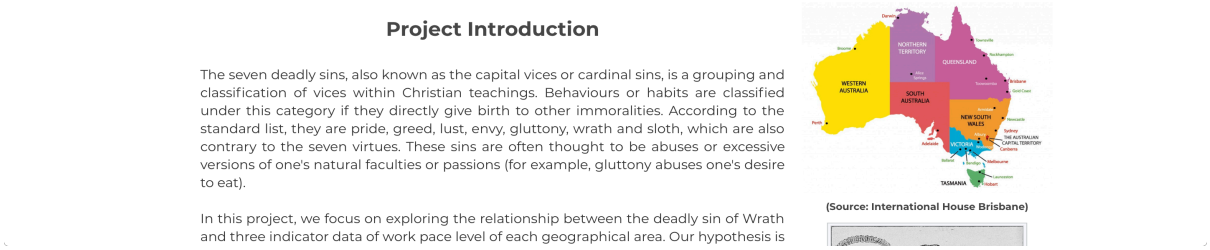




From the scatter plot, we can see there is a potentially positive correlation between the angry values of each city and its median personal income, although there is still a few outliers lie beyond. Unfortunately, the correlation between the unemployment rate and angry value does not demonstrate a positive correlation as expected, and is even slightly negative. Besides, there is no obvious correlation between median age of earners and the angry value. This liner regression model doesn't fully verify our hypothesis, due to the limited size of our tweets data set and biased dataset.

7. Web Design

The web application is developed based on HTML, CSS and JavaScript. The web page style refers to a free open source Bootstrap template named Instant authored by TemplateMag.com (URL: https://templatemag.com/instant-bootstrap-personal-template/). The final UI style of our home page is shown as below:



8. Data Visualization

In this project, we collected data of the 60 main cities in Australia from tweets and AURIN database, which means that a combination of data and the national geographical information is a good choice for data visualization. Here we choose the GeoJSO, Mapbox and Highcharts API for implementation.

8.1 The GeoJSON Format

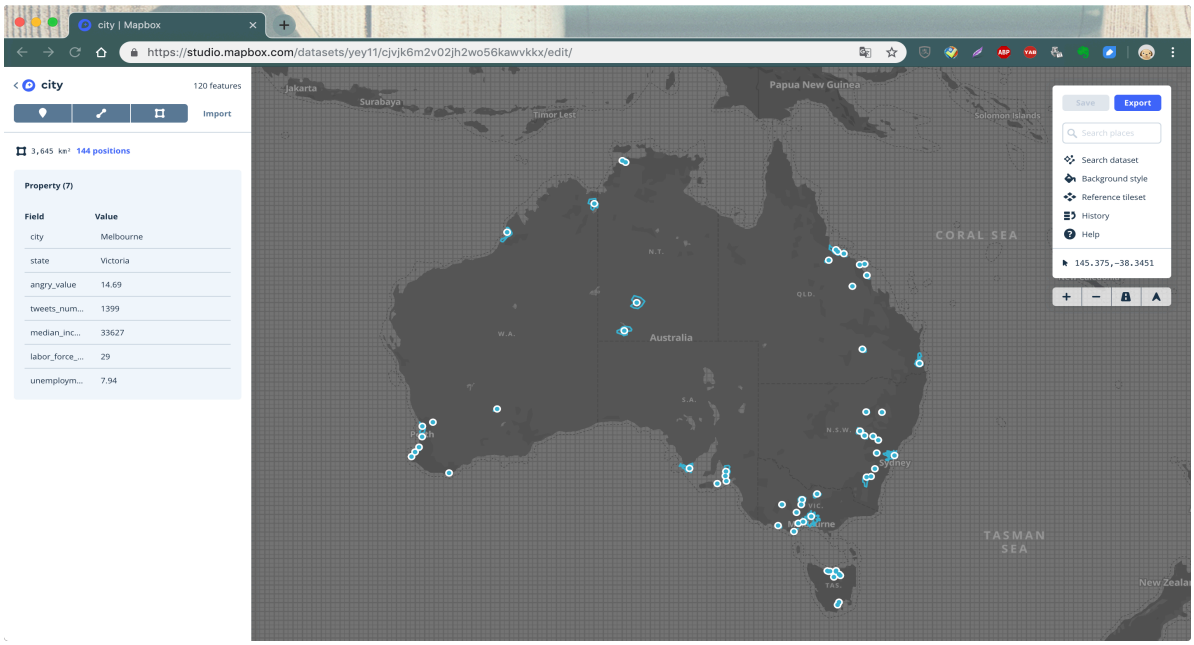
From the official documentation, the GeoJSON is a format for encoding a variety of geographic data structures, which supports the following geometry types: Point, LineString, Polygon, and MultiPolygon etc. Features in GeoJSON contain a geometric object and other properties, and a feature set represents a series of features. For example:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

The original tweets are categorised by cities and states. For example:


```
"id": "0073b76548e5984f",
"country": "Australia",
"full_name": "Sydney, New South Wales",
"attributes": {},
"url": "https://api.twitter.com/1.1/geo/id/0073b76548e5984f.json",
"place_type": "city"
```


The GeoJSON file for Australian states used in this project is authored by Rowan Hogan (https://github.com/rowanhogan/australian-states). However, we couldn't find the relevant GeoJSON files for Australian cities (not suburbs). Hence, we use Mapbox, an open source mapping platform for custom designed maps, to create the GeoJSON file for 60 cities:



The GeoJSON format applied in this project follows the GeoJSON Specification (RFC 7946) (https://tools.ietf.org/html/rfc7946). With the GeoJSON files, we can combine the statistical data with the geographical information to generate our own customized map.

8.2 Map Function

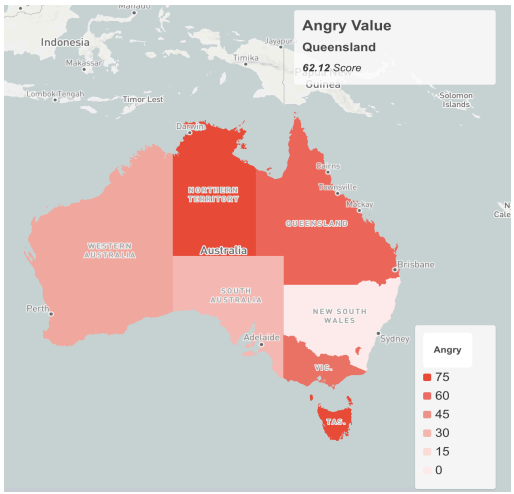
Apart from creating GeoJSON file, Mapbox also provides a powerful Maps API for developers. By adding the Mapbox API source in the `<head>` part of the HTML file, we can use the Mapbox official JavaScript and CSS libraries for front end development.

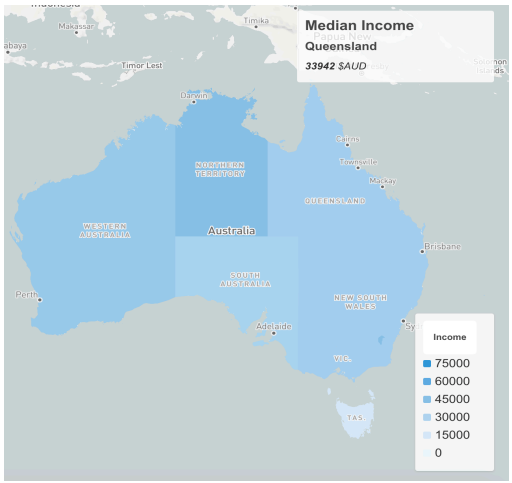
```
<script src='https://api.mapbox.com/mapbox-gl-js/v0.53.1/mapbox-gl.js'></script>
<link href='https://api.mapbox.com/mapbox-gl-js/v0.53.1/mapbox-gl.css'
rel='stylesheet'/>
```

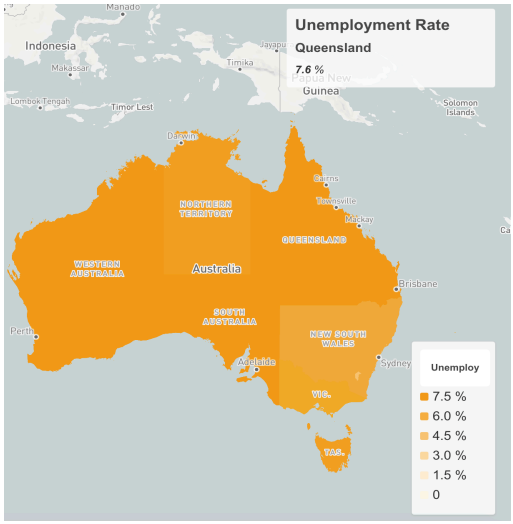
With an Mapbox access token and a style reference that we created in Mapbox studio, we can use JavaScript to load the map in a HTML file:

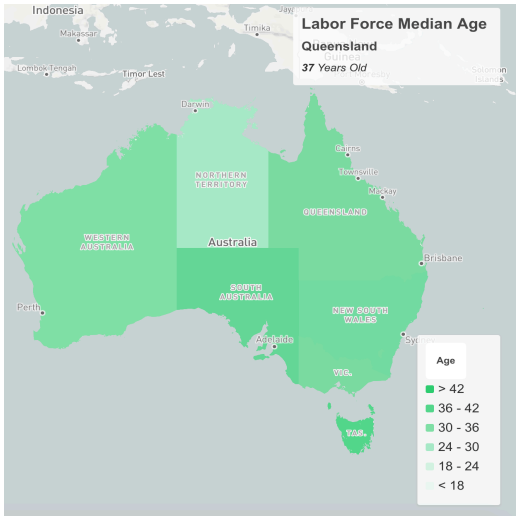
```
mapboxgl.accessToken =
'pk.eyJ1IjoieWV5MTEiLCJhIjoiY2p2NWNqcndxMzBlMTN5bGFxaXN2emNjYyJ9.F3qEnnxwuo3OfkbBXo
dGWQ';
// Create a new map
var map = new mapboxgl.Map({
    container: 'map', // container id
    style: 'mapbox://styles/yey11/cjvjf84410vx71dqmiwkhfdzj',  // Load map style
    center: [134.326164, -27.397418],  // Default center point
    zoom: 3.5,                         // Default zoom value
});
```

A very common way to show the data distribution on a map is with a choropleth, a thematic map in which areas are shaded based on a particular value. Through loading the GeoJSON files into the map, we can create a choropleth style map based on the four statistical indicators used in the data analysis, including the angry value, the median income, the unemployment rate and labour force median age. We can switch them by select the indicator options on the map legend as shown below. The colour code scheme refers to the flat design colour chart authored by Dixon and Moe (https://htmlcolorcodes.com/color-chart/flat-design-color-chart/).







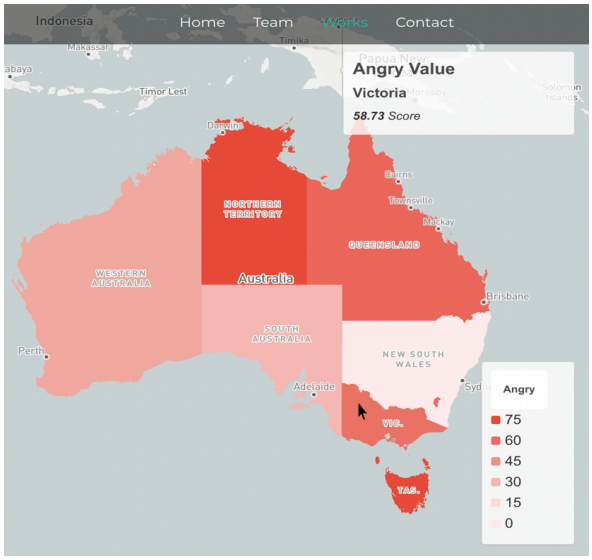


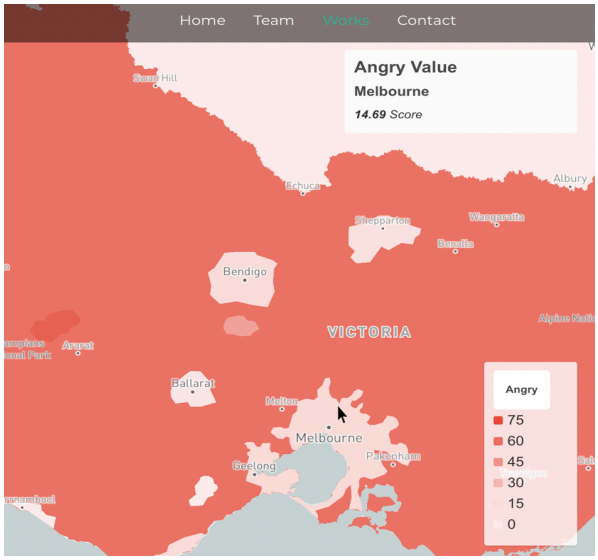
With the function `map.queryRenderedFeatures`, we can obtain the data of a region on the map where the cursor stopped, and then display it on the information board. Since the relevant data in some areas are missing in AURIN database, we need to check if the data exists before displaying on the information board. Here we use '1' to represent the missing data which corresponding to the white regions on the map. Also, note that we need to show the information of the city rather than the state it belongs to when the cursor is stopped on a city block.

```
map.on('mousemove', function(e) {
    // Obtain the features of states layer and cities layer.
    var states = map.queryRenderedFeatures(e.point, {layers: ['states_income']});
    var cities = map.queryRenderedFeatures(e.point, {layers: ['cities_income']});
    // Make the cities to be prior to states while selecting.
    if (cities.length > 0) {
        // Check whether the relevant data exists or not.
        if(cities[0].properties.median_income !== 1) {
            document.getElementById('pd').innerHTML = '<h3><strong>' +
                cities[0].properties.city + '</strong></h3><p><strong><em>' +
                cities[0].properties.median_income + '</strong> $AUD </em></p>';
        }
        else{
            document.getElementById('pd').innerHTML = '<h3><strong>' +
                cities[0].properties.city + '</strong></h3><p><strong><em>' +
'</strong> No data </em></p>';
        }
    } else if (states.length > 0) {
        document.getElementById('pd').innerHTML = '<h3><strong>' +
            states[0].properties.STATE_NAME + '</strong></h3><p><strong><em>'
            + states[0].properties.median_income + '</strong> $AUD </em></p>';
    } else{
        document.getElementById('pd').innerHTML = '<p>Hover over a region!</p>';
    }
});
```

By setting a zoom threshold value, we can control the visibility of the map layers in the way we want. When the zoom value is below the threshold, only the relevant data of the states will

be displayed on the information board which located the right top corner on the map, and if we zoom in over the threshold, the more detailed information of the cities will be shown.



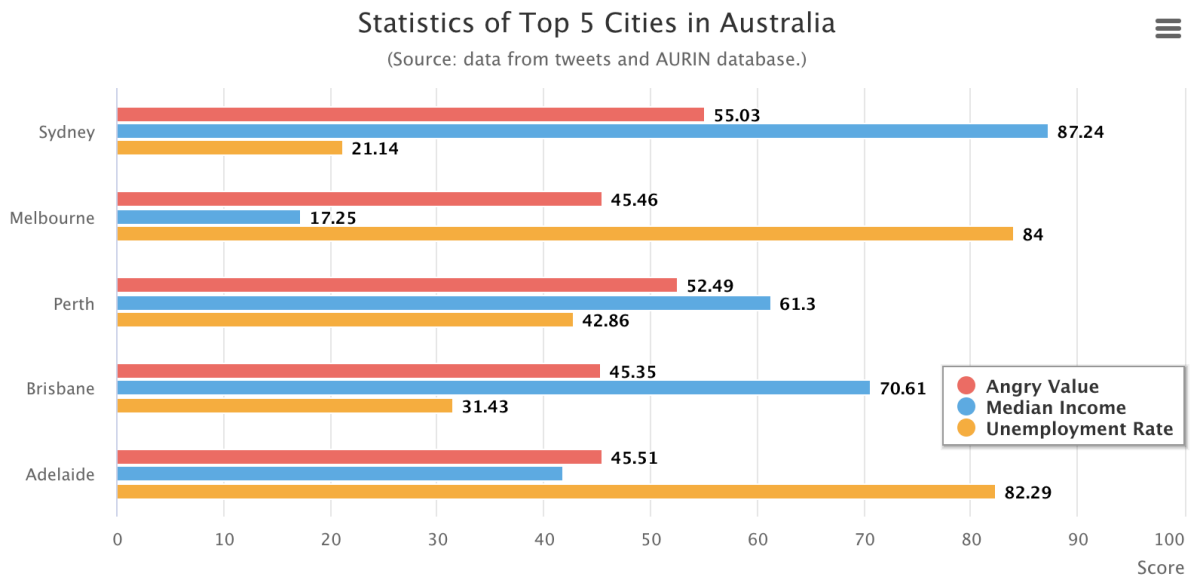


8.3 Statistical Result Chart

As for the statistical result visualization, we applied the Highcharts, a popular JavaScrip library for generating graphs and charts on the web page. By adding the Highcharts API source in the <head> part of the HTML file, we can use the relevant JavaScript libraries to create the charts.

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

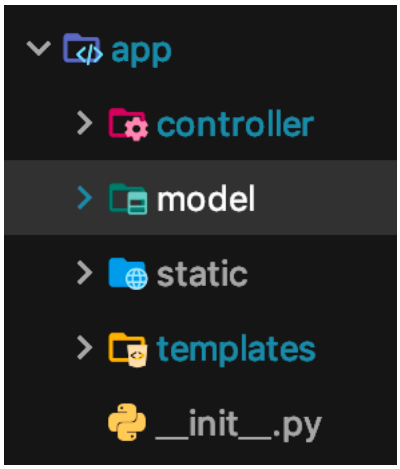
With the Highcharts API, we can customize the properties of the charts, including the size, the colour and the legend etc. and it also support loading data dynamically for a higher flexibility. Due to the limitation of the web page space, we select the top 5 cities in Australia and display their statistical results as a histogram chart which is shown as below. It should note that, the data displayed in the histogram chart has been normalized for a more convenient and effective demonstration.



From the chart, Sydney has the highest median income and highest anger value, while Melbourne has the lowest personal median income and almost the lowest anger value. However, the high median income and the lowest angry value are coexisting in the case of Brisbane. Hence, it seems that a higher income does not necessarily leads to a higher angry value. Also, there seems to be no obvious correlation between the unemployment rate and the angry value.

9. Flask Design

File structure:



Controller: the logic of backend

Model: the CouchDB database related code

Static: javascript and resource of frontend

Templated: HTML files need to be rendered

Here we use uWSGI to route the flask application so that it can process multi requests at the same time. However, I meet with access permission problem in nginx in cloud server so that I have use only uWSGI to process requests.

For uWSGI configuration, it controls the app location, numbers of process and threads and http and socket port.

```
[uwsgi]
callable = app
master = true
;home = venv
;socket = localhost:8001
module = ccc
wsgi-file = ccc.py
processes = 4
threads = 2
buffer-size = 32768
http = :80
chmod-socket = 660
vacuum = true
die-on-term = true
```

And Nginx config:

```
server {
    listen 80;

    location / { try_files $uri @app; }
    location @app {
        include uwsgi_params;
        uwsgi_pass unix:/tmp/uwsgi.sock;
    }
}
```

Here we set the port of Nginx, the location of app and the uWSGI socket port because Nginx access the uWSGI via socket port.

10. Error Handling

GeoJSON File: As mentioned before, we want to study the social sentiment analysis based on tweets among Australian cities, but we did not find any suitable GeoJSON file for Australia cities. All the GeoJSON files online we found are divided by states or suburbs. However, the tweets can only locate to cities rather than suburbs. The solution we applied is to create a GeoJSON for cities through Mapbox studio.

Missing Data on Map: We select three types of data (the median income, the unemployment rate and labour force median age) from AURIN for analysis. However, the data for some areas are missing. Here we use value '1' to represent the missing value because the original data format for all the three indexes would not take '1' as the value (e.g. thousands for income, percentage for unemployment rate and double-digit for labour force age). Another reason we choose '1' rather than '0' is that we found the areas with value '0' would appear as black blocks on the map in Mapbox.

11. Team Contribution

Ye Yang: Data Processing, Web Design, Data Visualization

Zhaohui Hou: Data Collection, Data Processing, Data Analysis

Haoyu Zhang: Environment Setting, Flask Server, Ansible

Yiming Xu: Environment Setting, Web Design

Wenkang Zhu: CouchDB, Data Analysis, Sentiment Analysis

Github Respository：

https://github.com/Neetordy/Cluster_and_Cloud_Computing

12. Reference

Aglassinger, T. (2018, April 2). Csv342 1.0.0 [Python package]. Retrieved from https://pypi.org/project/csv342/

Butler, H., Daly, M., Doyle, A., Gillies, S., …, Schaub, T. (2016) The GeoJSON format. Retrieved from https://tools.ietf.org/html/rfc7946

Dixon & Moe. (2015). Flat design color chart. Retrieved from https://htmlcolorcodes.com/color-chart/flat-design-color-chart/

Highcharts. Highcharts demos. Retrieved from https://www.highcharts.com/demo

Hogan, R. (2014). Australian states map. Retrieved from https://github.com/rowanhogan/australian-states

J. (2019, March 1). Matplotlib (Version 3.0.3) [Python package]. Retrieved from https://pypi.org/project/matplotlib/

J. (2019, March 14). Pandas 0.24.2[Python package]. The PyData Development Team, https://pypi.org/project/pandas/

Mapbox GL JS. Update a choropleth layer by zoom level. Retrieved from https://docs.mapbox.com/mapbox-gl-js/example/updating-choropleth/

Mapbox Tutorials. Make a choropleth map, part 1: create a style. Retrieved from https://docs.mapbox.com/help/tutorials/choropleth-studio-gl-pt-1/

Mapbox Tutorials. Make a choropleth map, part 2: add interactivity. Retrieved from https://docs.mapbox.com/help/tutorials/choropleth-studio-gl-pt-2/

Max Woolf. Reactionrnn: Pretrained character-based neural network for predicting the reaction to given text(s). 2017-, https://pypi.org/project/reactionrnn/#history [Online; accessed 2019-05-10]

M. (2019, March 30). Openpyxl (Version 2.6.2) [Python package]. Retrieved from https://pypi.org/project/openpyxl/