COMP90024 Cluster and Cloud Computing Assignment 2 - Understanding attitudes to climate change through demographics

Team 37

Janelle Lin Tang
694209
janellet1@student.unimelb.edu.au

Shuang Qiu
980433
qsq@student.unimelb.edu.au

Avinash Rao
1024577
avinashr@student.unimelb.edu.au

JJ Burke
1048105
jbburk@student.unimelb.edu.au

Declan Baird-Watson
640975
bairdd@student.unimelb.edu.au

May 26, 2021

Abstract

This report describes the design and implementation of our cloud-based solution for analyzing Twitter data in the chosen topic: "Understanding attitudes to climate change through demographics in Australia". It provides a detailed outline of our system architecture, infrastructure, and data pipeline. The report discusses the underlying challenges faced during the implementation stage, and the collaborative effort contributed by each team member.

Keywords Melbourne Research Cloud · Aurin · Twitter · Docker · CouchDB · Ansible · Django · Vue · Sentiment Analysis · Climate Change

Contents

1 Introduction

1.1 Motivation

One of the most salient and pressing issues of the 21st century has been the existential threat posed by climate change. It is well established in the scientific community that since the 1950's the impact of humans has caused a significant change to the climate never seen before [1]. Figure 1 illustrates global surface temperature versus human drivers and natural drivers since the pre-industrial revolution. The impact of humans is largely attributed to greenhouse gas emissions (in the form of carbon dioxide and methane) [2]. Rising global temperature threatens ecosystems, food security, economic loss, increased extreme weather events and environmental destruction [3]. Prolific reporting on the topic has disseminated through all available media outlets.

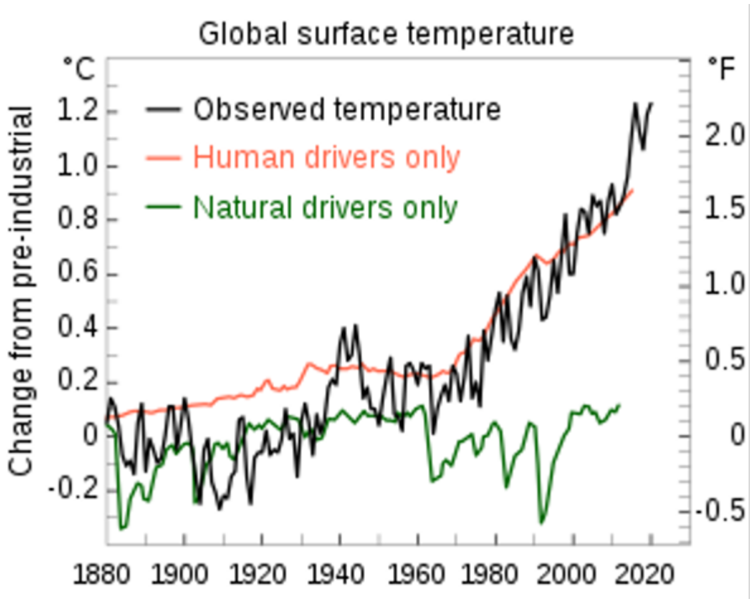


Figure 1: Global surface temperature vs observed temperature, vs human drivers, vs natural drivers from 1880 to 2020

Consequently, on social media platforms, such as Twitter, there has been high volumes of activity on the topic as users engage in the public discourse. This has provided us with a unique research opportunity to explore several hypotheses for better understanding support for climate change by population demographics.

1.2 General Approach and Scenarios

The overarching objective of this report is to analyse climate change attitudes broken down by demographic data for Australian cities. We collected data on all Australian state capital cities and some smaller cities. Where possible we included data on the cities' respective local government areas (LGAs). We did not cover territory capitals (Canberra and Darwin) due to small volumes and mapping complications. Our research scenarios included the following:

1. That higher income individuals will tweet more positively about climate change.
2. That young people will tweet more positively about climate change
3. That home owners will tweet more positively about climate change
4. That people employed by the mining industry will tweet more negatively about climate change
5. That people with solar panels installed will tweet more positively about climate change.

We designed the system architecture to operate in a cloud environment using the University of Melbourne Research Cloud. Our system is scalable and implemented via an Ansible script. For more details see section 3.

We harvested tweets from major Australian cities from 2013-2021 which contained keywords associated with climate change. Moreover, we filtered for tweets that contained geographical information relevant to our regions of interest. For further details on the Twitter harvesting architecture and process see Section subsubsection 4.1.1.

We employed Apache CouchDB™[4] to store twitter information and analysis data. See section subsection 4.2 for more details. We drew on data from the Australian Urban Research Infrastructure Network (AURIN) to address our scenarios. We used static data files (.csv) from AURIN. This is expanded on in subsubsection 4.1.3. The backend was implemented using web framework Django on the Model-View-Template architecture. Details on the Web implementation and RESTful framework can be found in subsection 6.1. The Frontend was implemented using Vue.js, the details of our mapbox and dashboard presentation can be found in subsection 6.2. We opted to utilise Docker[5] container technologies to maintain consistency between the production and deployment environment. For a detailed explanation of why we chose to containerise and use Docker see subsection 3.6.

The breakdown of work by each student is detailed in section 8.2. We summarise our findings in section 8.1.

2 User Guide

2.1 Deployment

2.1.1 Create Instances

In this step, we have created four instances on the Melbourne research cloud(MRC) through an ansible script. Firstly, we have installed all dependencies on the host, such as installing Python-pip, OpenStack SDK, and updating pip required for running the ansible-playbook tasks. Then Volume and Security Group has been assigned to the instances we have used the Security Group to give the access to the instance, and instances have given minimum port range 1 and the maximum port range 65535 and for the volume Instance1, Instance2 and Instance3 have been given 160GB volume, and Instance4 has been assigned 20GB volume. And to make use of the maximum RAM size provide to the project, we have given different flavors to individual Instances. We set flavor “uom.mse.2c9g” for Instance1, Instance2, and Instance3 and “uom.mse.1c4g” to Instance4 can use 31.5GB RAM out of provided 32GB.

Steps to run:

- Run LaunchInstances.sh file .
- Enter OpenStack Password from openrc.sh file .
- Enter Sudo password of your local system(Laptop).

Finally after running the LaunchInstances.sh file we are able to create all the 4 instances successfully with their allocated resources on MRC and generated inventory file for further use. The important features of LaunchInstances playbook has been described in Table 1

Roles	Description
Common	• Install pip, update pip, and install pip3 • Install `openstacksdk` on hosts
Volumes	• Create volumes from the variables
SecurityGroups	• Create security groups and its rules
Instances	• Create Instances on Melbourne Research cloud • Add IP addresses of instances into facts • Set IP addresses into inventory.ini

Table 1: **LaunchInstances** Playbook

2.1.2 Install Environment

Before installing environment we did two type of proxy setup in first setup we append the proxy config file into /etc/env and for the docker setup we append the config file into /etc/systemd/system/docker.service.d/http-proxy.conf . We added proxy to enable the instances to access the internet and can be also accessed outside. We have created the instances by using the instance image :

916cad7a-c545-48b2-b36c-d509ee63b3ce

which is MRC Ubuntu 18.04 LTS (Bionic) amd64 (with Docker) and already have docker setup however we updated the docker to make sure that docker is up to date. We also setup the GitHub proxy on the instance and clone the repository to each instance for the applications. The important features of **InstallEnvironment** playbook is described below.

Steps to run:

- Run InstallEnvironment.sh file .
- Enter OpenStack Password from openrc.sh file .
- Enter Sudo password of your local system(Laptop).

Roles	Description
InstallEnvironment	• Update apt, install dependencies, • Install pexpect, add apt repository, • Install docker-ce, Install docker, • Downloading the docker compose, • Add executable permission to the binary, • Downloading `node.js 10`, install `node.js 10`.
Clone Repository	• Copy Github private key to the remote servers, • Configure ssh to use ansible key for github.com • Clone source code from git
Mount Volume	• Make file system • Checking folders • Create directory • Mount device

Table 2: **InstallEnvironment** Playbook

2.1.3 Deploy Applications

For the application deployment, we firstly created three docker containers that hosted our three CouchDB servers. Then, we used docker volume to map the storage of instances to the container storage for the first three instances for CouchDB.

Secondly, we used the first CouchDB server at the first instance to config the CouchDB cluster. After that, we added the other two docker server with the help of commands provided by the official CouchDB documentation.

Thirdly we deploy three Django servers inside three docker container inside instances. After that, we created a docker file that pulls the python images and installs all the dependency required for Django, and we also pull the latest code from our GitHub repo, and then we run npm install command to install node module from `package.json`. We used the npm run build to generate the dist file for the frontend.

Next, we have deployed Nginx at instance one and used docker-compose through the first image from the docker hub, and then we mapped the volume of the config files and the frontend dist file to the docker container also mapped the port number 80. Finally, we run the Deploy crawler and sentiment analyser in two docker containers on instance 4.The important features of Deploy Applications playbook has been describe in Table3.

Steps to run:

- Run DeployApplication.sh file .
- Enter OpenStack Password from openrc.sh file .
- Enter Sudo password of your local system(Laptop).

Roles	Description
Deploy CouchDB	• Create directory for CouchDB config • Copy docker.ini file to CouchDB config • Copy docker compose to couchdb config • Invoke docker compose to create container
Config CouchDB Cluster	• Add second node 1 • Add second node 2 • Add third node 1 • Add third node 2
Deploy Django	• Pull the latest code from git • Install the node modules • Generate dist file for frontend • Invoke docker compose to create a container
Deploy Nginx	• Create directory for config and logs • Run docker compose
Deploy Crawler	• Create directory for config • Copy Nginx configuration file to server • Copy Nginx docker compose file to server • Invoke docker compose to create a container
Deploy Sentiment Analyser	• Run docker compose

Table 3: Deploy Application Playbook

3 System Architecture

3.1 System Design

As in Figure 2, the system consists of four instances. There is a Nginx server for load balancing and reverse proxy at instance 1. There is the Django server cluster containing three Django server running at three instances. There is also the CouchDB cluster with three CouchDB servers running at three instances. The instance 4 is for hosting the twitter crawler and the sentiment analyser which used the machine learning techniques to classified the tweets.

Due to the implementation of the clusters, the system is not vulnerable to the single point of failure. If any of the CouchDB servers or the backend Django servers are down, the client is still able to send the request to the system; Thus, ensuring the reliability of the system.

To further increase the throughput and the load of the servers, the load balancing technique has been introduced at the Nginx that will use the round robin technique to load balance the incoming requests into three servers. Also, the custom CouchDB load balancer will also load balance the request into three CouchDB servers to improve the system scalability and throughput.

Overall, the system is able to handle single point of failure and it is a scalable system that can handle the high throughput.

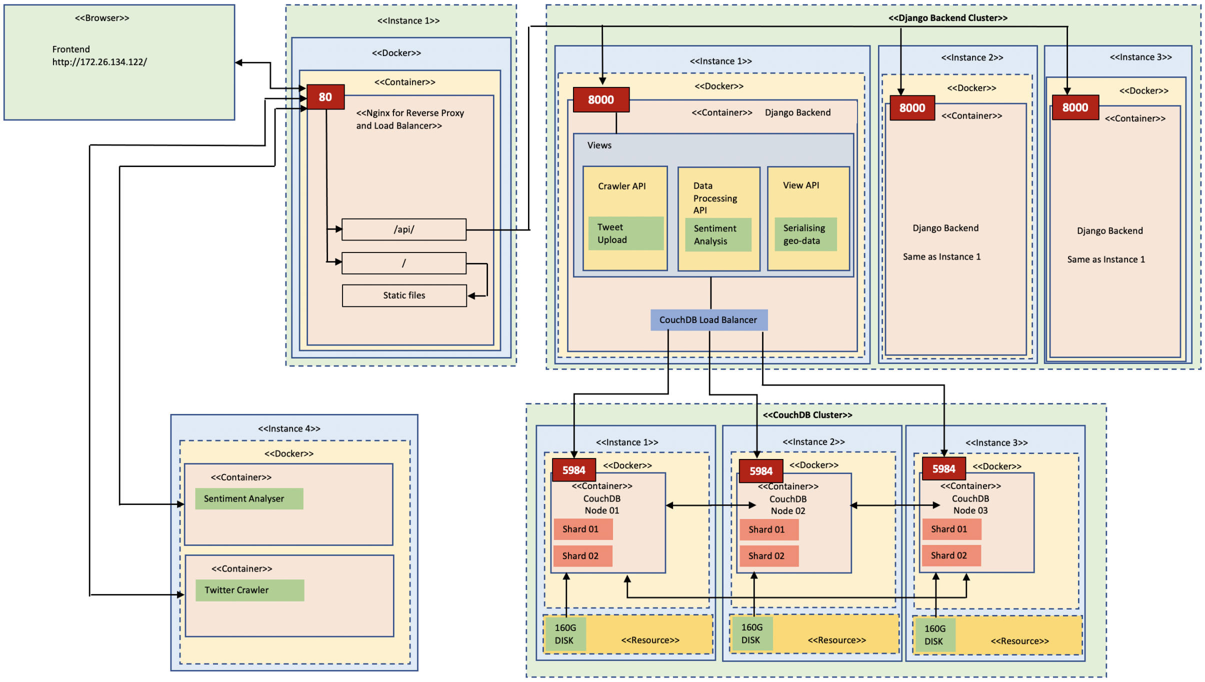
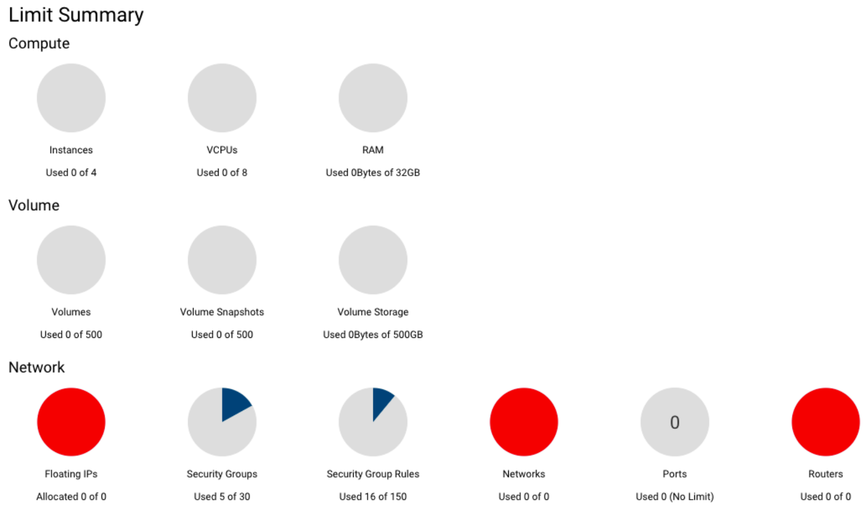


Figure 2: System Architecture Design

3.2 Unimelb Research Cloud Infrastructure

The University of Melbourne and its associated organizations' students, researchers, professors use Melbourne Research Cloud (MRC), which provides free 24*7 computing services. MRC has similar functionality to other commercial cloud providers GCP(Google Cloud Platform), Amazon Web Services(AWS), and Microsoft Azure. Researchers use MRC as Infrastructure-as-a-Service (IaaS), which helped them do data analysis, web hosting, and other potential uses. MRC's Infrastructure comprises unique services such as private networking, load balancing, DNS, and GPGPU(General-purpose graphics processing unit), along with 20,000 virtual cores , occupied across a range of virtual machine sizes. In this project, we are allocated four instances, eight virtual CPU, 32GB RAM, and 500GB of total volume storage .



Limit Summary

Compute

Instances
Used 0 of 4

VCPUs
Used 0 of 8

RAM
Used 0Bytes of 32GB

Volume

Volumes
Used 0 of 500

Volume Snapshots
Used 0 of 500

Volume Storage
Used 0Bytes of 500GB

Network

Floating IPs
Allocated 0 of 0

Security Groups
Used 5 of 30

Security Group Rules
Used 16 of 150

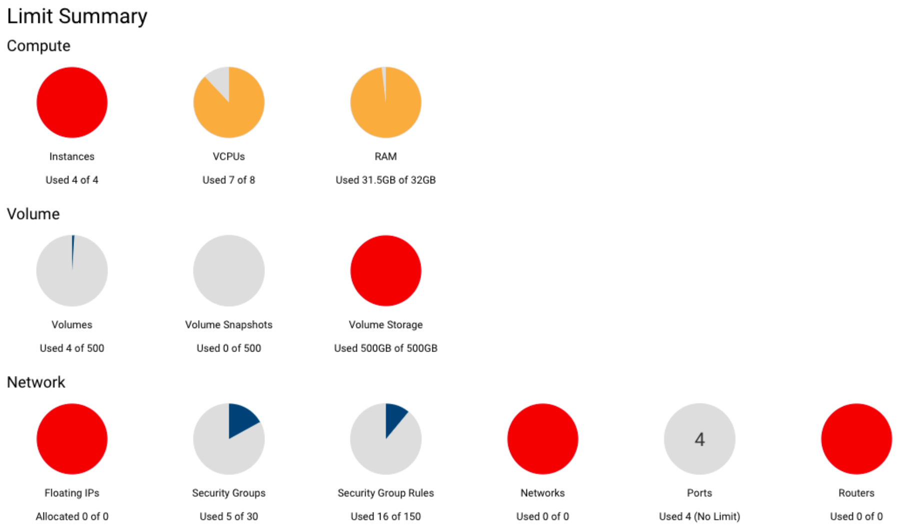
Networks
Used 0 of 0

0

Ports
Used 0 (No Limit)

Routers
Used 0 of 0

Figure 3: Allocated Resources on MRC for the Project



Limit Summary

Compute

Instances
Used 4 of 4

VCPUs
Used 7 of 8

RAM
Used 31.5GB of 32GB

Volume

Volumes
Used 4 of 500

Volume Snapshots
Used 0 of 500

Volume Storage
Used 500GB of 500GB

Network

Floating IPs
Allocated 0 of 0

Security Groups
Used 5 of 30

Security Group Rules
Used 16 of 150

Networks
Used 0 of 0

4

Ports
Used 4 (No Limit)

Routers
Used 0 of 0

Figure 4: Resources used by our Group

3.2.1 Advantages of MRC

Economical: The vital benefit of using UniMelb Research Cloud is cost, as we don't need to set up any hardware environment for our project. As a result, we don't need to worry about the hardware cost for the cloud system. And on the other hand, MRC's cloud service usage is free for students; hence, it is very economical compared to AWS and AZURE.

Efficient: MRC follows a private deployment model under the organization's intranet firewall (in our case, Unimelb) that guarantees good network performance and efficiency.

Scalability: MRC provides cloud resources to create our own scalable and dynamic environment; hence, scalability is also an essential benefit of using UniMelb Research Cloud.

Flexibility: One significant advantage of using the UniMelb Research Cloud is its flexibility. As MRC has an easy customization feature, their development team can easily customize the hardware and other resources based on project-specific specifications.

Availability: As MRC is open to all support staff, researchers hence they can straight away originate the servers they need without buying, maintaining, or disposing of cloud hardware devices. And MRC is also available 24 hours a day, seven days a week, from within and outside The University of Melbourne and anywhere in the world. Furthermore, MRC has detailed tutorials for all OS users to guide using the cloud, making it easier and more user-friendly to use. Thus combine all these quality plays an essential role in UniMelb Research Cloud's availability.

Reliability: MRC is created by the Australian Research Data Commons (ARDC) team and runs on OpenStack, allowing rapid innovation. OpenStack's orchestration and self-service capabilities enable developers and IT staff to gain faster and more efficient access to IT resources. Because machines can be provisioned quickly and on-demand, developers can significantly reduce development and testing times; hence they have more freedom to experiment with new ideas. As a result, the ARDC team can develop new features and fix bugs for the MRC faster, resulting in more reliable service.

3.2.2 Disadvantages of MRC

Other commercial cloud providers like Microsoft Azure and AWS(Amazon web services) offer a plethora of services that can be used to accelerate application development, for example, DevOps integration, etc., but MRC uses OpenStack technology, which appears less mature.

MRC follows a private deployment model; hence the deployment infrastructure is only limited within the University of Melbourne, which causes the limiting potential to its resource allocation for the project as MRC does not allow the provisioning of computing instances with public IP addresses could be externally accessible.

3.3 Backend Component

The backend is built using Django web framwork. For more detail, see subsection 6.1.

3.4 Frontend Component

The frontend integrates with the backend and is built using Vue-Cli. For more detail, see subsection 6.2.

3.5 Database Component

We primarily used CouchDB, a clustered database that stores data in JSON format. For more detail, see subsection 4.2.

3.6 Containerisation

The most crucial advantage of containerization is that we can maintain consistency between the production and deployment environments. Because our team members use various operating systems, including Mac OS, Windows, and Ubuntu, it is difficult to guarantee that a piece of code that runs perfectly well on my laptop will also run correctly on others.But we can efficiently run the program anywhere with the container once the environment is set, which drastically reduces the amount of time we need to spend debugging.

It is much easier to set up the development, migration, and deployment environment from a practical standpoint. Using Docker as an example, when migrating code from one host to another, simply pack the source code and unpack only on the new host. Then, without further ado, type docker-compose up, and everything will be ready to go. There are two options for deploying the service to the server. It is possible to either pull the built image from Docker Hub (or any other location to store our image) or use an Ansible script. Both routes are simple and straightforward.

Finally, it performs better in terms of security. Different containers can communicate with one another via a virtual network. The advantage of this is that if we deploy a database and a load-balanced backend on the same server, only the backend will access the database. In addition, we could allow the load-balance server to bind to 0.0.0.0 and listen to all the ports within the virtual network because of the network isolation, only exposing the port where we want the container to accept requests. As a result, we have more flexibility in selecting which port or ports to accept submissions without changing the source code, only the container settings.

3.6.1 Why Docker?

For our project, we use Docker as a containerization solution. Even though other tools, such as Tectonic, can achieve a similar goal, Docker has its advantages. We chose Docker for two main reasons. Primarily, it is much simpler to learn. Given the time constraints of this project, we need to select an easier-to-learn tool to build something instead of spending most of our time just getting acquainted with the tool. There are also many tutorials and resources available that cover most of the issues we encountered while building the system with Docker. Moreover, it has a more vibrant community. We could use many community extended versions of Docker images through Docker Hub while still being able to customize them.

Docker also has some beneficial properties that can make our lives easier. First and foremost, construct stage by stage. Multiple parts of this project may share a common base environment. In this case, we could first build a similar function and then construct each of them separately based on that, eliminating the need to write the same thing multiple times while still keeping the container lightweight. We could also use this property to merge multiple images, and the resulting image would be smaller than the sum of their sizes.

The other is caching, which means that we don't have to rebuild the entire image from scratch when we make changes to the Docker container. This excellent feature significantly accelerates our development. The last one to mention is docker-compose, which allows us to start and stop services quickly. Because building our crawler system is an IO-intensive task, we use multi-processing (because Python GIL does not support true multi-threading).When we need to stop the crawler, we must search for the process ID in the system and eliminate them one by one, which is inconvenient, especially during debugging. We could quickly start and stop the crawler with a single command using docker-compose.

4 Data Solutions

In this section we detail the various ways we handled the data. subsection 4.1 details the setup we used to harvest tweets as well as our approach to the AURIN data set. subsection 4.2 details why we selected CouchDB to handle our twitter data. subsection 4.3 gives an overview of how we processed the data in the backend, including the sentiment analysis approach employed.

4.1 Data Collection

The data used in our project was collected via three different channels. subsubsection 4.1.1 describes the methods used to extract twitter data, subsubsection 4.1.3 describes the collection of data from AURIN, and subsubsection 4.1.4 outlines the collection of other data files.

4.1.1 Twitter Data Harvesting

To collect our twitter data we used the Python programming language and the `Tweepy`[6] Python package. We registered 5 student developer twitter accounts to access additional functionality. The harvesting method was connected with CouchDB. Our approach to harvesting tweets was split into a static and dynamic collect method:

1. **Static method:** Premium search tweets across major Australian cities. We employed the premium search functionality on 5 developer API credentials. We designed our program to avoid search point rate limit errors on harvesting volumes by cycling through each user API credential. In addition, to bypass harvesting volume restrictions we searched across a grid of points (5km radius) in each Australian city from 2011-2014. From the premium tweets we harvested user timeline data and extracted relevant tweets from their histories. These tweets were uploaded as a static JSON data file to CouchDB.
2. **Dynamic/Stream method:** Tweet streaming. We streamed live tweets from twitter based on a set of climate change related keywords. We slightly modified the `Tweepy` python library to harvest tweets in batches instead of continuous streaming. This allow for a small amount of initial processing of tweets without having the connection to twitter timing out. For backup, a try-except loop catches any errors connecting to the twitter stream and if an exception is thrown a wait period is incurred. Since almost no tweets were geo-enabled with coordinate information we filtered for tweets based on a free text location field on the user's profile. A quick and dirty search looked for mention of Australian city names in the location field, and if this was not found the free text field was geocoded using `GeoPy`[7] to identify Australian based tweets. This was based on the assumption that users had truthfully listed their location on their profiles. Again, if a tweet met the criteria above then the associated user's timeline was scraped for more relevant tweets. In order to streamline processing and save storage space only a subset of the raw tweet data was used. For an example of the reduced tweet see Listing 1. Both the live streamed tweets and user timeline tweets were uploaded to CouchDB as a JSON file.

Listing 1: Example of harvested tweets in CouchDB

```
[
    {
        "tweet id": "448624451555315712",
        "user id": "174560975",
        "text": "Clear &amp; concise. RT @Jess_Irvine: The inconvenient truth
        about climate change: either way,
        you pay http://t.co/7DCIy8rET8 #auspol",
        "lang": "en",
        "user location": "Melbourne, Australia",
        "user geo_enabled": true,
        "geo": [
            145.07263306,
            -37.65063677
        ],
        "created at": "2014-03-26 00:56:18",
        "hashtags": [
            "auspol"
        ]
    },
    {
        "tweet id": "1318290870802677760",
        "user id": "57602078",
        "text": "RT @MrKRudd: It\u2019s time. Time to act on Murdoch\u2019s
        climate change denial. Time to act on his dominance over Australian
        politics. Time to act\u2026",
        "lang": "en",
        "user location": "Sydney Australia",
        "user geo_enabled": true,
        "geo": null,
        "created at": "2020-10-19 20:40:10",
        "hashtags": []
    }
]
```

4.1.2 Duplicate Handling

For uploading tweets to CouchDB we selected a tweet's "tweet ID" field to be the primary key. The "tweet ID" field is unique to each tweet and so it guaranteed no duplicate tweets in CouchDB. The "tweet ID" field was converted to a string before being uploaded to CouchDB to avoid the max integer size limit. In the rare instance where a tweet was harvested twice, the most recent harvested duplicate tweet would replace the original tweet.

4.1.3 AURIN Data

Demographic data was pulled from the Australian Urban Research Infrastructure Network (AURIN), specifically from the portal interface. Data was selected was related to power (solar panels and extraction industry job), housing status, and age/income/demographic data. This data all downloaded as csv files, which were processed by a Python script to limit to only desired features, format LGA names in a consistent manner, and output data as a JSON file to be easily used by front end design. The file is one object, with keys being the names of LGA's, and values being another object containing that LGA's information.

4.1.4 Other Data

The geometry rendering on frontend as outlined in **SECTION** was done using data from data.gov.au, a government run open source repository for Australian datasets. From this, we sourced the GeoJSON files containing shape data for regions across Victoria, New South Wales, Queensland, Tasmania, South Australia, and Western Australia. GeoJSON is a file format that is specifically used for encoding geometry data - this allows us to represent a polygon with non-spatial attributes in the file.

4.2 Data Storage

4.2.1 CouchDB

One of the primary reasons we have used CouchDB to store the tweets collected from Twitter because it provides the Developer complete control over the resolution of file disputes. CouchDB would send all the dispute files to the application, where the Developer can resolve the dispute correctly. Having said that, nothing is perfect, and CouchDB is no exception; according to the CAP Theorem, Couch DB prioritizes availability as well as partitioning.

As a result, CouchDB would lose some consistency across all the nodes. However, the number of tweets across every node will be slightly different in our application due to inconsistency. And as we did the front-end data visualization over the millions of tweets, hence the slight difference in the number of tweets across nodes will not make a significant difference.

4.2.2 Spatialite DB

The secondary database used was Spatialite, which formed the database for the Django backend web framework. This was selected due to it's simplicity and ability to store geometry data, since this was essential to our method for rendering polygons on the frontend. It is also easy to backup, yet provides well-rounded functionalities for our purposes. The Django model instances for city and LGA geometry data are stored in the Spatialite database. For more detail on these models, see subsection 6.1.

4.3 Data Pipeline

The harvested twitter data is stored in the **Twitter** database in CouchDB. This is accessed via an API through Django (see subsection 6.1) and processed in three key steps. subsubsection 4.3.2 outlines the steps taken to locate each tweet, subsubsection 4.3.3 discusses our method for sentiment analysis, and subsubsection 4.3.4 discusses the aggregation of our data before storing in CouchDB.

4.3.1 MapReduce

Our data processing was initiated with MapReduce, which was used to return tweets that hadn't been processed. This was determined by an additonal field - `isProcessed` - which is altered when the tweet is accessed for the first time. We also designed a view that would return all hashtags found in our harvested tweets, along with a count of each occurrence. This view simply loops through the hashtag array in each documents and emits the lowercase version of the hashtag as a key, and then calls the default reduce count function.

4.3.2 Geo Matching

The harvested twitter data often did not come with the geotags of the exact location at which the tweet was created. Consequently, we made the assumption that when this geotag was not available, the tweet was created from the user's location - a Twitter attribute that the user could manually populate with their location. Of course, this is not always accurate, and thus our results should be treated accordingly.

The extracted location was parsed through a geocoder from the `GeoPy`. This could take in a text location, such as *Parkville, Melbourne*, or coordinates: $[-37.794541, 144.956701]$. This would return a dictionary of the detailed address of the location, from which we would extract the 'municipality' i.e. the local government area (LGA), the city, and the state. Using this geocoder, we also removed any Tweets not within Australia, or if the address did not have a granularity down to the city the location was in i.e. a user location of *Victoria, Australia* would not be accepted. Additionally, due to the inconsistency of location address breakdowns for territories, Canberra and Northern Territories are also excluded.

4.3.3 Sentiment Analysis

Sentiment analysis involves assigning positive or negative scores to documents based on the words found inside the document. Some versions of this model involving training on a pre-labeled corpus of documents and classifying unlabeled documents. Instead we used a method that scores documents based on the inclusive of a predefined dictionary of words that have scores applied. This was achieved using the `NLTK` library in Python, as shown in Listing 2

Listing 2: Using `NLTK` to analyse tweet sentiment

```python
from NLTK import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
sia.polarity_scores(text)
```

This sentiment analysis algorithm utilizes two predefined dictionaries to assign positive or negative score to word documents. The first dictionary contains the score for a list of words. The second dictionary contains a list of negation terms. that flip the sentiment score of a word if preceeded by one of the negation terms.

The tweet text was preprocessed before the sentiment analyser was used, as shown in Table 4. This involved:

- Removing all punctuation, mentions, and emojis,
- converting hashtags to text e.g. #ClimateChange ⟹ climate change,
- lowercasing all text,
- removing stopwords using the `nltk` stopword corpus, and
- removing inflectional morphemes using the Porter stemmer.

This method is slightly limited in that the corpus used by `SentimentIntensityAnalyzer()` does not recognise negative terms particularly well. Moreover, it is difficult to analyse nuances in speech such as sarcasm, and therefore may incorrectly assign a sentiment value.

Before		**After**
@CarlKatter and Tony Abbott was too gutless to face the media. MT no change on marriage equality, no action on climate change, same old	⟹	toni abbott gutless fac media mt chang marriag equal action climat chang old

Table 4: Example of before and after preprocessing of the tweet text

4.3.4 Data Aggregation

The collected tweet sentiments were then aggregated by geographical regions before being stored in CouchDB. Listing 3 shows the design of the document formed by aggregating the tweets, first by the LGA, then by the city. Additionally, the aggregated values for dates and times of day were added as attributes.

Listing 3: Example of document in Regions Database

```
{
  "_id": "cairns",
  "_rev": "96-28513992ec949dbc78f071125e8459bc",
  "LGA": {
    "cairns regional": {
      "total_sentiment":0.2742999999999989,
      "total_tweets": 1
    }
  },
  "name": "cairns",
  "state": "queensland",
  "dates": {
    "2014-12-05": {
      "total_sentiment": 0.2742999999999989,
      "total_tweets": 1
    },
  },
  "times": {
    "07": {
      "total_sentiment": 0.2742999999999989,
      "total_tweets": 1
    }
  },
  "hashtags": {
    "planetonfire": 1
  },
  "total_sentiment": 0.2742999999999989,
  "total_tweets": 1
}
```

4.4 Data Transfer

The main mode of data transfer is using Representational State transfer API (ReST) framework implemented through Django. For more detail, see subsubsection 6.1.2.

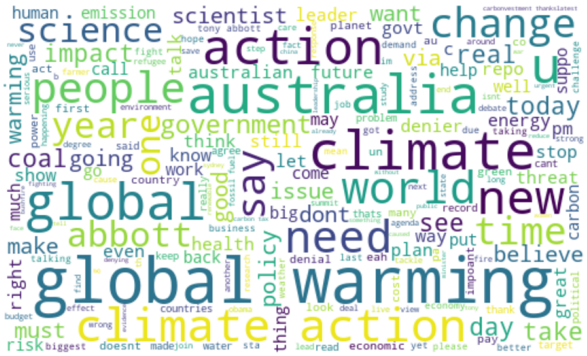


Figure 5: Wordcloud of first tweets

5 Analysis

We analyzed the Twitter data, which includes the tweets related to climate change, and the Aurin data, which consists of demographic data (Solar panels, Extraction industry job, Housing Status, Age, and Income). Figure 7 shows a summary of a few of these features in relation to the collected Twitter data. We found that people with Age under 35 are more positively to tweet about climate change than others. We also found that those towns with a combined income of people living over there more than 3000 per week are more positively to tweet about climate change.

Apart from Age and Income, we also found a correlation between the Twitter data and Aurin data regarding the people who have solar panels installed in their houses to tweet about climate change more positively. Lastly, we analyzed the Twitter data about the people employed by the mining industry and the cities with more renters and found out that they were tweeting negatively about climate change.

5.1 Topic Modeling

There are varying discussions and topics within the tweets, despite all being sourced on climate change, and as such, it is difficult to reveal much useful information from the aggregated tweet data. Figure 5 illustrates a word cloud of the initial tweets, where majority of terms are expected, and hardly discloses much new information. In an attempt to parse out some differences between tweets, we used a Latent Dirichlet Allocation model (LDA). This model is for unlabeled corpora, and works as a sort of clustering algorithm for text documents [8].A limited number of topics was selected (3) because there are a subset of the larger topic (climate change) that was selected on and applies to all tweets. When using 3 topics in the algorithm, the following are the 10 words most correlated with each topic as shown in Table 5.

solve	think	Australia's	support	health	emissions	thanks	agenda	water	carbonvestment
threat	policy	first	planet	report	urgent	strong	right	still	weather
future	Australian	important	going	scientists	believe	power	denial	environment	security

Table 5: Top 10 words in the top 3 topics from our initially sourced tweets on climate change

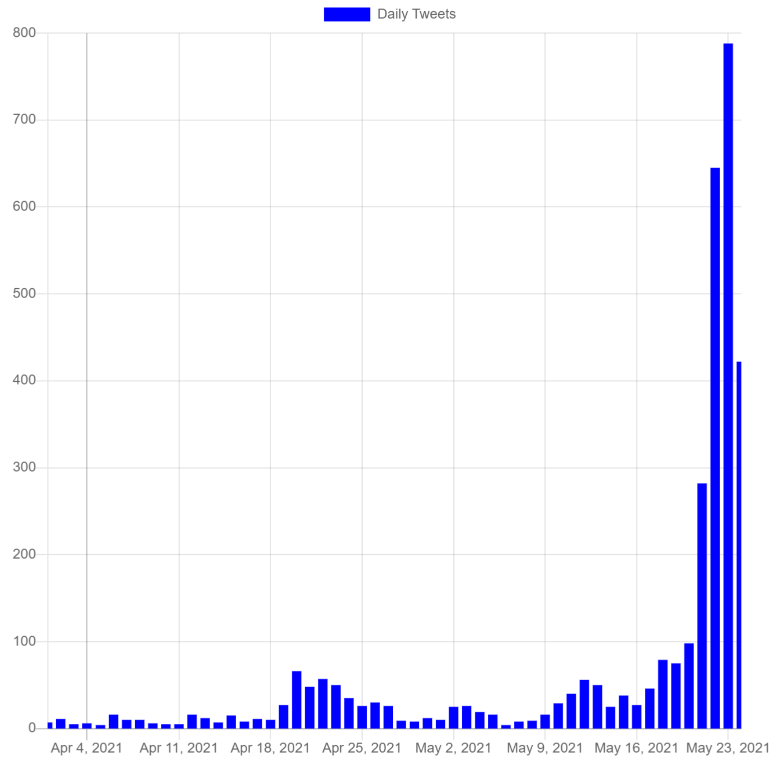


Figure 6: Tweet Volume Since April 1 2021

After looking at tweets with at least a 0.6 probability of each topics, the topics were categorized as

1. Policy discussions,
2. Skeptisism discussion (both skeptics and non skeptics), and
3. News articles

This is supported by the sentiment analysis performed on tweets assigned to these categories; Topics 1 and 3 had a higher average sentiment compared to Topic 2, which makes sense as Topic 2 is a more confrontational topic.

Topic	Category	Tweet Count	Average Sentiment
1	Policy Discussions	2120.6	0.03747
2	Skeptisism Discussion	2183.9	0.02799
3	News Articles	2083.4	0.03703

Table 6: Summary of Tweet Topics. Tweets assigned to topics on probability weighted basis

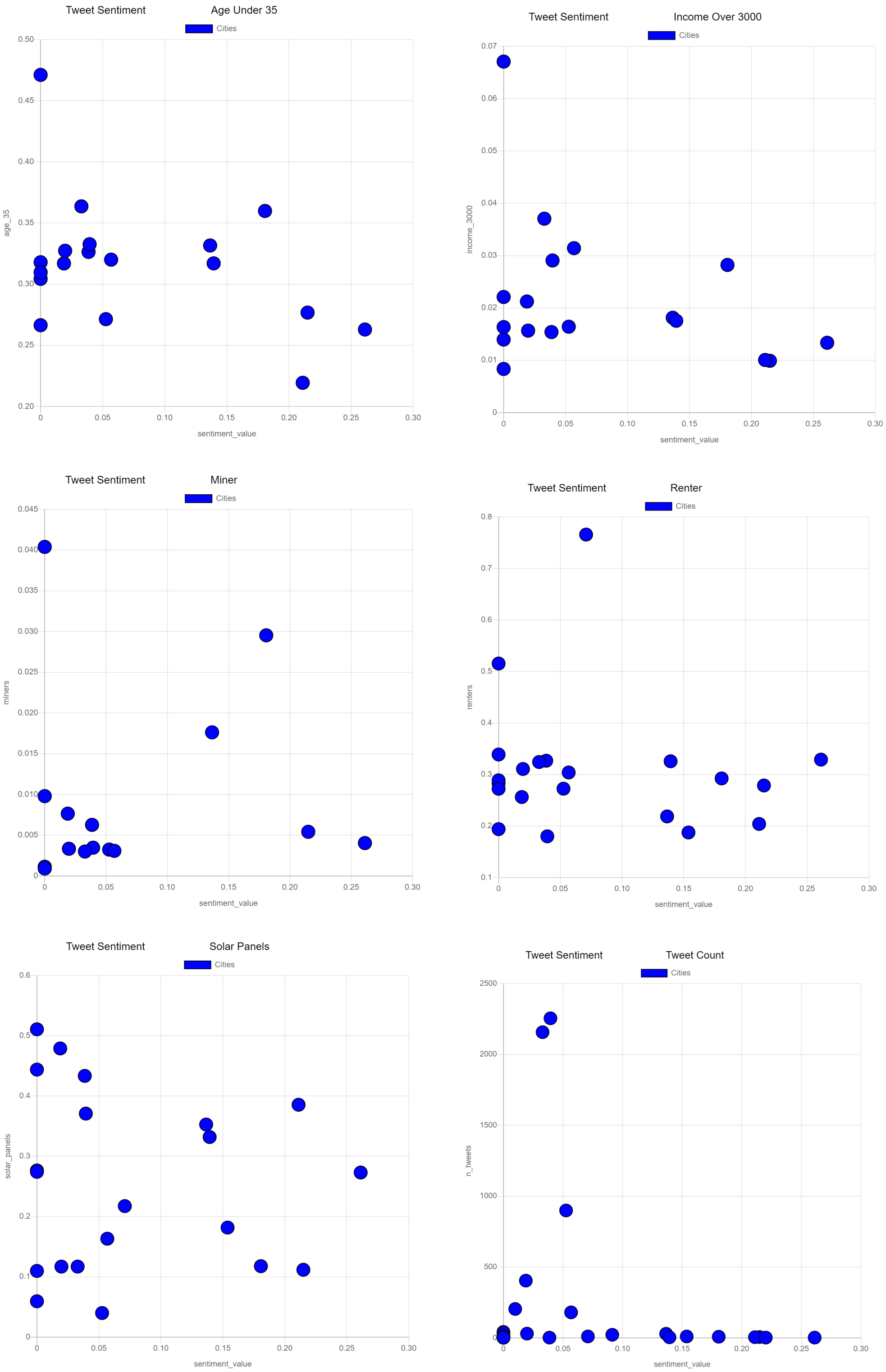


Figure 7: Average Tweet Sentiment Compared to Selected Demographics: Age Under 35 Percentage, Under Over 3000 per Week Percentage, Mining Industry Participation Percentage, Percentage of Homes Rented, Percentage of Homes with Solar Panels, and Total Number of Tweets

6 Web Implementation

6.1 Backend

The backend was implemented Django web framework build on the Model-View-Template architecture. As such, Django provides a more robust solution than other web frameworks, such as Flask. Using GeoDjango, an extension of Django, and Django REST framework, we were able to serialise the data in GeoJSON format that allowed for easy rendering in Mapbox. Moreover, Django provides an easily scalable framework, with a large user base to ensure complete support.

6.1.1 Model Instances

Two Django models were created: **City** and **LGA**, where each model was populated with combined data taken from the Regions database in CouchDB and external GeoJSON files as introduced in subsubsection 4.1.4. Table 7 outlines the attributes for each Model.

Attribute	Description	City	LGA
Name	Full name of area	✓	✓
City	City in which LGA exist		✓
State	State in which LGA exists	✓	✓
Geometry	Polygon or Point object of area	✓	✓
Sentiment	Total sentiment of tweets in area	✓	✓
Tweets	Total tweets in area	✓	✓

Table 7: Summary of Attributes for City and LGA Models

The model instances are created each time the Regions database is being updated with incoming tweets via a `GET` request.

6.1.2 RESTful Framework

Django has a REST framework extension that allows for great flexibility and easy construction RESTful APIs.Through this, we created a series of endpoints that allow us to create GET and POST requests between Django, CouchDB, and the frontend. The Twitter data is retrieved from CouchDb via a `GET` request. The data is contained as a JSON formatted response message, and processed, before sending it back to CouchDB in the Regions database. Django then requests data from the Regions database using another API, which utilises MapReduce to configure the tweets before creating the Model instances with the desired attributes.

Not only are model instances sent to frontend via an API, but the other data in Regions database is requested the same way. For more detail, see subsection 6.2.

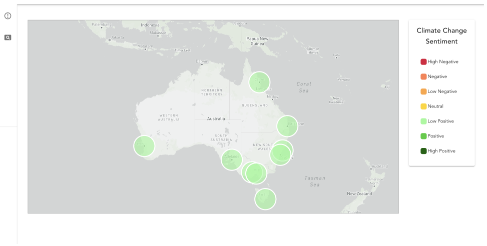


Figure 8: Mapbox rendering of Sentiment on Climate change in Major Australian Cities

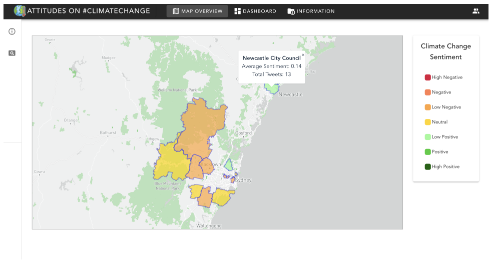


Figure 9: Mapbox rendering of Sentiment on Climate change in NSW LGAs

6.2 Frontend

The frontend was implemented using Vue.js, a javascript framework, via Vue CLI. It was chosen to facilitate collaborative work using components, which can be designed separately and combining in one page. It also allows for easy data and event binding to rendered objects, which helps in creating responsive and interactive visualizations.

6.2.1 Mapbox

The first stage of our frontend is a map, rendered using MapBox, which displays geographically where tweets are occurring and the associated sentiments. The polygons are retrieved from Django via an API that serialises the data in GeoJSON format. Figure 8 and Figure 8 illustrate the display of the map, with two different geographical levels. When zoomed in, data is rendered as outlines of Local Government Areas, with a color indicating their ranking for total sentiment values. When zoomed out, data is rendered as circles for each city, again with color showing ranking for total sentiment values.

6.2.2 Dashboard

The second page used is our dashboard page. This page can be used to get a general overview of all the data, as well as extract information on each geographic subdivision. There are 4 separates graphs on this pages. All charts on the dashboard page are constructed using vue-chartjs, a Vue wrapper for chart.js.

This page calls out to three sources for data: the `api/location/lga` API endpoint, the `api/location/dates` API endpoint, and a static json file containing all our AURIN data. The LGA endpoint contains all LGA which have tweet data, and provides the number of tweets and total summed tweet sentiment, which is then combined with AURIN data and summed across named cities to get data for the first Dashboard chart. Any dot on this chart can be clicked to reload the other three charts with data specific to this city. Clicked on the header will reset the page to all cities data again.

For each data property, a min and max amount is calculated across cities to be able to calculate indexes to display data on the bottom left chart. This is an effort to have every property scaled from 0 to 1 to display it all in one place effectively. If there is no data for a selected city for any properties, that property will be removed from the chart.

The two charts on the right use data retrieved from the `api/location/lga` endpoint. This API passes an object with each city as keys, and values containing another object, with keys being dates of tweet and values being another object containing the tweet counts and tweet sentiments for each city and date segment. This data is then summed either across all cities or for a selected city, Then array of both average sentiment value and total number of tweets are produced to be displayed in the charts. Data on these charts have been filtered to only show April 1st 2021 and later, due to not much daily data being available before then.

- Scatter Plot with all Data as Options
- Line Plot of total sentiment over time
- Bar Plot indicating each AURIN statistic indexed to measured max and min
- Bar Plot of number of tweets over time

Clicking a dot will trigger a response in the other three graphs, where they are re-rendered with data only of the city tied to that do. Clicking on the header displaying city name will cause the graphs to re-render with all data.

Mouse overs on all data points will provide additional information. In the case of the scatter point, a mouse over when a ratio is plotted will show both the numerator and denominator numbers. For the indexed graph, the actual numbers will be displayed instead of the index value.

6.2.3 Word Cloud

The third page contains a word cloud made of hashtags found on scrapped tweets. The word cloud is constructed using the Vue package VueWordCloud. This page is fed data by an api that returns all hashtags found in the harvested tweets along with a count of each occurance. The hashtags were all converted to lower case to make count totals case insensitive. When this is received by the page, the data is then sorted by highest count to lowest count, and the word cloud is constructed from the 100 most common hashtags. Text colors are assigned randomly from 4 possible options to better display separate hashtags.

7 Key Challenges

7.1 MRC Challenges

One of the Major challenges we faced for using the MRC for this project is that we have been provided with only limitated storage; hence, we are not able to store as much data as we want to store on cloud. There was also a problem related to physical hardware limitations from the service provider therefore only imitated certain servers are provided.

While using MRC, we also faced some issue regarding creating the instances as whenever we deleted the instances and created the new one just after the old one, then there is some probability the new instance has the same IP as the deleted old one, and we cannot ssh to the newly created instance, and we are getting an error that would say “identification of instance has changed”.

7.2 Geometry Data Challenges

For our map rendering, MapBox needs a server on the backend to store all the geographic features as a GeoJSON file. However, neither CouchDB nor Django serialisers can package the data as GeoJSON. Thus, there was a tremendous amount of issue with getting a database setup with geometry capabilities, that would appropriately serialise the data into GeoJSON format. While Spatialite seemed to be an easy solution, we faced many issues from installing all the necessary packages to storing the correct type of shape data.

7.3 Single Point of Failure

A key challenge to this project was to handle the single point of failure properly. For the traditional single application, it is vulnerable to the single point of failure because there is no backup if the application fails. Therefore, the server needs to expanded horizontally to form a cluster to prevent the single point of failure. We found it difficult to properly design the API flow to the clusters so that the single point of failure can be handled. In the end, we chose to user Nginx as a Reverse Proxy server to handle the cluster.

The load balancing technique also needs to be set up to dynamically scale the server when there is a large throughput to the servers. There is no existing load balancers that can be integrated into the CouchDB cluster. Therefore, we need to design a custom load balancer inside the Django server that will load balance the incoming API requests to the CouchDB cluster.

8 Final Remarks

8.1 Conclusion

The results of this study proved to be interesting. While there was no statistically significant conclusions to be drawn there were some observations that could warrant further exploration. We found that people with Age under 35 are more positively to tweet about climate change than others. We also found that those towns with a combined income of people living over there more than 3000 per week are more positively to tweet about climate change (**??**).

Apart from Age and Income, we also found a correlation between the Twitter data and AURIN data regarding the people who have solar panels installed in their houses to tweet about climate change more positively. Lastly, we analyzed the Twitter data about the people employed by the mining industry and the cities with more renters and found out that they were tweeting negatively about climate change (**??**).

Furthermore, we found that as tweet volumes increased the sentiment score was low - approximately 0.05. This was particularly true for cities such as Melbourne and Sydney whose population is around 5 million. The authors explain this finding by suggesting that the level of aggregation for large cities reflected the `NLTK` sentiment analyser mean and not a "true" city mean. Anecdotally, our `NLTK` sentiment analyser had a mean value of 0.05, the variation we see in the smaller cities is partly explained by the small sample size and therefore larger variation around the mean. Future studies would benefit from more precise tweet location information to draw relationships on smaller LGA regions.

8.2 Team Contribution

Table 8 outlines the contribution of each member to the project.

Name	**Contribution**
Janelle Lin Tang	• Meeting coordinator and note taker • Report Writing • Backend Django development • Frontend Mapbox rendering
Shuang Qiu	• Report Writing • Containerised deployment using Docker • Write Ansible script to Deploy whole applications • CouchDB server setup • Project coordinator
Avinash Rao	• Report Writing • Write Ansible script to Create Instances • Write Ansible script to Install Environment • Make Power Point Presentation
JJ Burke	• Report Writing • Front End Dashboard Development • Topic Modelling Analysis • AURIN data integration
Declan Baird-Watson	• Static Twitter Crawler development • Streaming Twitter Crawler • Report Writing • Scenario formulation and analysis • System Architecture Designer

Table 8: Team Member Contributions

8.3 More Information

8.3.1 Github

We used open source version-control system GitHub for sharing source code. If you wish to refer to our source code, here is a link to our GitHub repository:

https://github.com/JanelleTang/COMP90024_Assignment_2

8.3.2 Youtube Videos

To demonstrate the capability of our system we have created a series of YouTube videos. The links are provided below:

- Tour of frontend @ http://172.26.134.122/
  - https://youtu.be/ldwQKv6J7F8
- Deployment demonstration
  - Install Environment Video
  - Launch Instances Video
  - Deploy Applications Video
- System Architecture
  - https://youtu.be/Q2IWcv9H46U

References

[1] *Data or dogma? : promoting open inquiry in the debate over the magnitude of human impact on Earth's climate : hearing before the Subcommittee on Space, Science, and Competitiveness of the Committee on Commerce, Science, and Transportation, United States Senate, One Hundred Fourteenth Congress, first session, December 8, 2015.* S. hrg.: 114-373. 2016. URL https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=edsgpr&AN=edsgpr.000997856&site=eds-live&scope=site&custid=s2775460.

[2] Eugene A. Rosa and Thomas Dietz. Human drivers of national greenhouse-gas emissions. *NATURE CLIMATE CHANGE*, 2(8):581 – 586, 2012. ISSN 1758678X. URL https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=edswsc&AN=000307369300013&site=eds-live&scope=site&custid=s2775460.

[3] Zoe Ettinger. New books examine the effects of climate change and offer ideas for how to heal the planet. *Publishers Weekly*, 268(6):22, 2021. ISSN 0000-0019. URL https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=edsglr&AN=edsglr.A652011165&site=eds-live&scope=site&custid=s2775460.

[4] Apache couchdb (2021). date retrieved: 25 may 2021. URL https://couchdb.apache.org/.

[5] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[6] Tweepy. (2021). date retrieved: 25 may 2021. URL https://www.tweepy.org/.

[7] Geopy 2.1.0 (2021). date retrieved: 26 may 2021. URL https://geopy.readthedocs.io/en/stable/#.

[8] Ng Andrew Y. Blei, David m. and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, (3):993–1022, 2003. URL https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf.