COMP90024

Cluster and Cloud Computing

*Project 2: Is Melbourne Still the Most Liveable City in the World?*

|  |  |
| --- | --- |
| Team Members | Student ID |
| Pagnarith Pit | 1068350 |
| Aswesh Thrissur anand | 1441115 |
| Anjaneya Turai | 1391922 |
| Divyanshu Mishra | 1281413 |
| Swetha Gopikumar Sreeja | 1426378 |

Table of Content

|  |
| --- |
| 1. Project Description and Problem Statement |
| 1. Resource Allocation and Details |
| 1. Data Requirement |
| 1. Core Functional Components |
| 4.1 Twitter Processing and Analysis |
| 4.2 Harvesters |
| * 1. NoSQL Database |
| 4.4 User Interface and Backend |
| 1. System Design Tools |
| * 1. Ansible |
| 5.2 Dockers |
| 1. Cost and Benefits Analysis of Tools |
| 1. Challenges and Errors Handling |
| 1. Lessons learnt throughout the project |
| 1. Study Limitation |
| 1. Full Data Collected 2. Scenarios Analysis |

1. Project Description and Problem Statement

This report outlines all the tasks carried out as part of Project 2 for COMP90024 Cluster and Cloud Computing. All technical details and codes can be found in our team’s Github Repository. We have also uploaded videos on how some of these components operate. Namely, we have 2 video demos on the inner workings of Ansible and the walkthrough of the web-front end. Please find more information here:

* Github Link
* Youtube Videos

In this project, we are building a software that allows us to tell a story based on multiple sources of data. More specifically, we have a large collection of Twitter data, and along with information from the Spatial Urban Data Observatory (SUDO) and Mastodon, we compile together all relevant data to be able to make insightful inferences on multiple scenarios here in Melbourne. In our case, our over-arching concept is based on how liveable Melbourne really is.

In the past few years, Melbourne has been crowned as the ‘world’s most liveable” city consecutively. However, as with COVID and the high rising inflation recently, news articles have hinted at the fact that people are misled to believe this notion, and that this city does indeed have many flaws. Therefore, with access to the Melbourne Research Cloud, our team began looking to see if this statement does indeed hold through public sentiments. To form our conclusion, we have researched into 4 of the following scenarios:

* Rental Prices and Affordability Index across Melbourne
* Land Prices
* Transportation Costs
* Medical Care Access and Affordability

The system built has many components. First of all, we have a processing step for all data included such as the large Twitter file. Next, we have our harvestors that collect live toots from Mastodon servers. All these are then transferred to a NoSQL database cluster using CouchDB, before finally using MapReduce to find all relevant information and visualize it on our front-end user interface. Below, the report provides an in-dept view of how each component was built along with instructions on how to reproduce the results found.

As for the technology used, our team focus on scalability, error reduction, and compatibility for this software. Therefore, we take full advantages of tools such as:

* Ansible

A configuration management software that allows for dynamic deployment of containers, and this is the main tool used for scaling our software

* Docker

A container technology that allows for quick and easy deployment, with fault tolerance benefit from Docker Swarm as a container management tool

* Tableau

A web front end tool that allows for easy construction of RESTful API for our user interface

Please note that all tools used for this project are of base or free tier, and therefore some features may be lacking. However, this report is structured in a way to help all intended user extend the capability with ease.

1. Resource Allocation and Details

For the given tasks, the team is allocated a total of 8 virtual machines with the maximum storage capacity of 500 GBs. For our tasks, we have decided on the following split:

* #1: Twitter Analysis – Volume attached: 100 GBs

This virtual machine is responsible for storing the large Twitter file as well as processing all the data in it. In order to extract important information, the Twitter file is parsed, and only tweets in Victoria (Melbourne included) are pulled. However, some tweets have missing bounding box in their geometry information, and these ones cannot be visualized on the front-end map. Therefore, a further pre-processing step is applied to filter out and keep only those with bbox in their data value for the front-end visualization.

* #2 and #3: Harvesters – Volume attached: 50 GBs each

Two virtual machines are dedicated to hosting the Mastodon harvester. The volumes attached are sizable, because originally, all data collected are stored locally before being pushed to our database (CouchDB cluster). However, this was subsequently improved, by changing the harvester script to immediately send the data to the cluster upon receiving it.

* #4: CouchDB Database Master Node – Volume attached: 80 GBs

This node serves as the master for the database cluster. The volume given here accounts for the fact that it was first used as a testing node before clustering and partitioning began, and thus we allow most initial data to be stored on this machine.

* #5 and #6: CouchDB Database Worker Node – Volume attached: 60 GBs

These two nodes serve as the worker, but as with CouchDB, once clustering is properly setup, this notion of master and worker goes away, and all nodes behave as one unit.

* #7: Web User-Interface

This node is responsible for hosting the web front end of the project. The scripts of Tableau are embedded into HTML files stored in its volume. Note that all information are pre-uploaded (i.e, static), but we have properly set up all security groups and allow communications across the virtual machines to enable further extension such as live data transmission to the UI.

* #8: Testing VM

Finally, this node is used as a test node. Small tasks, such as testing containers, are done on here to prevent random bugs affecting the more important nodes in the architecture.

When working on the Melbourne Research Cloud platform, there are some requirements when setting up the instances. These include:

* Availability\_zone: Melbourne-qh2-uom
* Instance\_network: qh2-uom-internal
* Instance\_flavour: uom.mse.2c9g
* Instance\_image: NeCTAR Ubuntu 22.04 LTS (Jammy) amd64

While these may seem trivial, adhering to these setups comes with many major benefits. For one, having different availability zone for an instance and its volume would result in an inaccessible storage. This small human error would produce a bug that is rather difficult to find and solve. Next, the instance\_network setup with “internal” suffix would mean that only those connected to the University’s network would be able to access the instances. Therefore, this provides a good layer of security.

As for security group, the Research Cloud allows for custom groups to be made, and this makes it not only secure but flexible for the users to enable trusted communication between our instances. For this, the team has decided on 2 distinct groups. One, called default, is primarily used for virtual machines that do not require a lot of security and/or communication access. That is, VMs such as Twitter Analysis and Testing basically operate in isolation, and these do not store any important information. As such, this security group allows for ingress and egress through all ports such as 22, 80, and accepts communication from the world as well. Another group that the team set up is called “couchdb”, and for this one, more selected ports such as 5984, 9100-9200, are opened to allow for CouchDB containers to communicate. This, again, could be improved with a more advanced technique, and more details are provided in the CouchDB section below.

1. Data Requirement

To complete the project, multiple data sources are required. This section will provide and overall look at all the datasets used, and for all specific files, they are listed in Section 8 in Scenario Analysis.

* Spatial Urban Data Observatory

This platform hosts a vast collection of Australian statistics. For this project, various datasets including rent index, medical care information, are collected as part of our authoritative information set

* VicGov

This dataset also provides authoritative dataset for our analysis. We wanted to incorporate more than one source besides SUDO to provide for a richer analysis, more up-to-date data, and a wider variety of scenario search

* Twitter

We are supplied with a large twitter json files (approximately 60 GB) containing tweets from the beginning of 2023. From this file, we extract important information like texts and sentiments to corroborate the scenario analysis along with SUDO data. We also extract their locations to visualize the tweets on our front end.

* Mastodon

Finally, to further strengthen the points we are making by incorporating recent data, we use Mastodon. As the site describes itself, Mastodon is a free and open-source software for running self-hosted social networking services. We have chosen different servers to harvest from to increase information extraction, but also demonstrate how this project can scale.

1. Core Functional Components

This project is broken down to multiple major software components. This section will

detail the process of how each one is built alongside instructions on how to run the code on Ansible for deployment.

* 1. Twitter Processing and Analysis

With one of our dedicated virtual machines, we began cleaning up the twitter data. First task was to extract only tweets that have proper geo-location in them. As our final goal is to be able to visualise the data as well as perform scenario analysis alongside the SUDO data, tweets without location information would be unhelpful and thus removed. Next, these tweets also contain important information such as texts and sentiment scores that can be used to gauge public opinion on our related topics. As such, all of these information are extracted and cleaned, such that they can be pushed to our database cluster.

* 1. Harvesters

The first software the team built is the harvester. For this task, 2 servers from Mastodon,

namely Mastodon AU and Mastodon Social, are used for toots streaming. Mainly, this information is used to supply current and up-to-date information surrounding the topics of interest as the Twitter file contains past data.

After applying for both API token keys, the team first used cURL command to begin harvesting toots from the federated timelines. However, this method pulled uncleaned data such as “thumbs”, a signal where the process emits periodically to ensure connection is kept up. Therefore, a Mastodon API on Python is instead downloaded and utilized.

To ensure that scaling is possible, a base setup is built locally before being pushed to the cloud. These include only the base script, and all environment variables such as “URL” and “API token” are dynamically passed in at every run. All the dependencies and scripts are written to a Dockerfile and an image is then built on this. After which, this Docker image is pushed to our Dockerhub. The container image is now ready to be pulled by all of our harvester virtual machines, and Ansible is used for this step.

Originally, all harvesters would store the collected information locally being before sent to our database cluster. While doing it this way was simple, it meant that new data are not incorporated as we go, and this slowed down the process of analysis. To solve this, once the database has been properly configured, we changed the harvester script to dynamically send through all collected toots to the database immediately upon harvesting.

Instructions on how to deploy:

*ansible-playbook -i nodes.ini deployHarvestor.yaml*

* 1. NoSQL Database

Another major component of our project is our CouchDB cluster. This forms the end

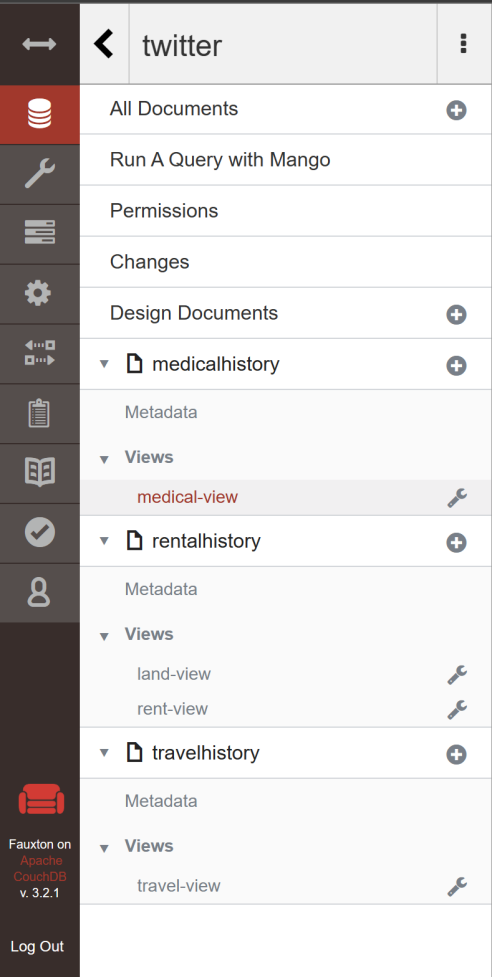
point of all data collected, and all our analysis takes place with MapReduce views. CouchDB is best suited for the tasks as our dataset comes in the form of JSON files. By having a NoSQl system, the storage is efficient and built-in functions such as MapReduce are extremely helpful. While this sacrifices Consistency of data across our nodes, in our case, this is not at all as important as behaviours such as Fault Tolerance. Our data can be “sharded” or partitioned across these database nodes to ensure that if one fails, the important analysis performed would still persist. As MapReduce is famous for “bringing the operation to where the data is”, this portioning of data is of no concern to the operation.

To get started, all CouchDB database across each database node is built using Docker. After pulling the necessary image, all the database containers are built with the following details:

* *Couchdb\_image: ibmcom/couchdb3*
* *Couchdb\_user: admin*
* *Couchdb\_password: admin*
* *Couchdb\_version: 3.2.1*
* *Couchdb\_cookie: a192aeb9904e6590849337933b000c99*

By constructing the database in this manner, not only is it scalable and time efficient, but it also prevents any compatibility issues arising from human errors. For instance, by pulling the wrong version, this would result in a mismatch and a hard to fix issue.

Once everything is properly configured, the views are then created for analysis.



*Figure 1: CouchDB View Summay*

We created 4 different views, namely medical-view, rent-view, land-view, and travel-view as shown above. Below is an excerpt of these.



Here, we define all the suburbs in Melbourne/Victoria and look for the words that contain scenario specific keywords. We utilised known names for Melbourne and Victoria suburb from files obtained from earlier work (Assignment 1) along with keywords found from analysing relevant documents, such as news articles, for these scenarios. We then emit the documents that satisfy the condition and use it for further analysis.

**Instructions on how to deploy:**

Starting all CouchDB containers individually across the virtual machines

*ansible-playbook -i nodes.ini startCouchDB.yaml*

Clustering the nodes:

*ansible-playbook -i nodes.ini clusterDB.yaml*

* 1. User Interface and Backend

Front end is developed using ReactJS and Tableau. It helps us visualize the twitter data and analysis out of it with interactive maps rendered using Tableau.

Tableau

This section of the report highlights the successful implementation of Tableau in the frontend for mapping and visualization purposes. GeoJSON files extracted from tweets were plotted on maps and graphs using Tableau. The Tableau software offers robust capabilities for data visualization and business intelligence. It enables users to analyze and present data in visually appealing and interactive formats, making it an ideal tool for this project.

The project involved extracting GeoJSON files from tweets, which contained geographical information related to various events, locations, or sentiments. These GeoJSON files were seamlessly integrated into Tableau to create visualizations. The blending of GeoJSON files with other relevant datasets which were obtained from various data platforms like SUDO and Kaggle was performed in the Tableau data source section. This blending process ensured accurate representations of relationships between data points, enhancing the overall visualizations.

We created 4 distinct dashboards, each catering to specific scenarios within the tweet data. These dashboards provided targeted visualizations and insights, enabling users to interact with the data effectively. After finalizing the dashboards, they were uploaded to the Tableau Public server. Tableau Public is a free platform that allows users to share their visualizations with the public. This step made the dashboards accessible to a wider audience online.

To seamlessly integrate the Tableau visualizations into the frontend, the Tableau Public vizzes were embedded directly into the frontend interface. This integration ensured a smooth user experience without the need for separate platforms.

Overall, the usage of Tableau for mapping and visualization of GeoJSON files proved to be a successful approach. The dashboards created using Tableau provided interactive and visually engaging representations of the tweet data. By uploading them to the Tableau Public server and embedding them on the frontend, the visualizations became easily accessible for exploration and analysis.

React.js

ReactJS is a popular JavaScript package for creating user interfaces (UIs) for online applications. It was developed by Facebook and is widely used by UI/UX developers around the world owing to its simplicity and ease of use. Using React, you can quickly update and render reusable UI components as the application's state changes. Below are some key concepts and features of ReactJS:

* Components: React organizes the user interface into reusable components, which are self-contained building parts that encapsulate the HTML, CSS, and JavaScript logic for a specific section of the user interface and may be nested inside of each other, allowing for a modular and hierarchical structure.
* Virtual DOM:  React uses a virtual representation of the browser's Document Object Model (DOM), known as the Virtual DOM. When the state of a component changes, React is very efficient in updating its virtual DOM and computes what minimum modifications are needed for true browser DOM to be updated which will improve performance.
* JSX: React makes use of JSX (JavaScript XML), a syntactic extension that lets you write HTML-like code directly inside of JavaScript. JSX offers a declarative and simple approach to specify the structure and look of components.
* Unidirectional Data Flow: React adheres to a unidirectional data flow, commonly referred to as one-way binding. Properties (props) are the means through which data is sent from parent components to their child components, and callbacks are often used to notify the parent components of any changes in their child components.
* State Management: Mutable state, which represents data that may transform over time, is a property of React components. One can effectively manage and update the state of the components by utilizing the useState hook or state management libraries like Redux or MobX.

During the implementation of Tableau in the frontend, several challenges were encountered and successfully addressed. The following challenges were faced, along with the corresponding solutions:

1. Blending the Data Sets:

One of the primary challenges encountered was blending the different data sets required for creating comprehensive visualizations. Integrating and harmonizing multiple data sources within Tableau can sometimes be complex, especially when dealing with diverse data formats and structures. In this project, merging the GeoJSON files extracted from tweets with other relevant datasets posed a particular challenge.

To overcome this challenge, a careful data preparation process was undertaken. This involved ensuring consistent formatting, resolving inconsistencies, and aligning common data fields across the various data sets. Through meticulous mapping and transformation, the data sets were effectively blended within Tableau, enabling the creation of cohesive visualizations that incorporated all the necessary information.

2. Embedding HTML Code:

Initially, embedding the HTML code generated by Tableau into the frontend presented a hurdle. The HTML code provided by Tableau contains various elements and scripts that need to be integrated correctly to ensure proper functionality. It requires a solid understanding of HTML and web development principles.

To simplify the integration process and eliminate potential issues related to embedding the HTML code, an alternative solution was explored. Instead of directly integrating the HTML code, it was decided to leverage the capabilities of Tableau Public. The finalized dashboards were uploaded to the Tableau Public server, and a public link was generated. This link was then embedded in the frontend, providing a seamless user experience. This approach circumvented the need for complex HTML code integration and mitigated any potential complications.

By utilizing the Tableau Public link, the dashboards hosted on the Tableau Public server could be easily embedded in the frontend. This simplified the deployment process and ensured that the visualizations remained accessible and functional without requiring intricate technical implementation.

Hosting in VM

When transitioning the website hosting from local to a virtual machine (VM), several challenges were encountered. The following challenges were faced, along with the corresponding solutions:

Basic NPM Start for Hosting:

Initially, the website was hosted using the basic "npm start" command. However, this approach did not provide the necessary network security measures and did not ensure proper accessibility from the external network. As a result, the website was not accessible outside of the local environment.

To address this challenge, additional network security configurations were implemented. Specifically, the appropriate firewall rules were set up to allow inbound connections on port 3000, which is the default port used by the website. By opening port 3000, the website became accessible from external networks, ensuring proper connectivity and usability.

Setting up Nginx as the Engine:

To enhance the reliability and scalability of the website, Nginx, a popular web server software, was employed. However, integrating Nginx and ensuring its proper functioning presented a challenge. Nginx acts as a reverse proxy server, forwarding requests from clients to the appropriate backend server.

To overcome this challenge, a dedicated VM was set up specifically for Nginx. This VM handled the routing and load balancing of incoming requests to the website. By utilizing Nginx as the engine, the website's performance and availability were significantly improved.

Separating VMs for Frontend and Nginx:

To optimize the hosting environment and ensure efficient resource allocation, two separate VMs were created. One VM was dedicated to running the frontend application using the "npm start" command, while the other VM was solely responsible for hosting the Nginx server. This separation allowed for better resource utilization and simplified management. The VM running the frontend application continuously executed the "npm start" command to keep the website running, while the Nginx VM handled the routing of incoming requests to the appropriate backend server. By creating two distinct VMs, the website hosting environment became more scalable and manageable, providing an efficient solution for serving the frontend application and managing the Nginx server.

1. System Design Tools
   1. Ansible

The backend technology all utilizes Ansible as a configuration management to deploy and scale. The reason why Ansible is so versatile is that it is “agent-less”, meaning that it will be able to communicate instructions to all managed nodes without any special application or service installed on those nodes. Thus, dynamic deployment of tools such as Docker containers are streamlined using this technology and are able to scale horizontally extremely effectively.

To navigate the Github repo, all Ansible scripts are located in “Full-Backend” folder. From there, all host names and variables for this project are neatly stored in “host\_vars/var\_name” along with the private key used for Secure Shell logins. The inventory (i.e, managed nodes) are located in “inventory” folder under the file named “nodes.ini”. Next, all executable yaml playbook files are stored in the “playbook\_files” directory and their corresponding roles are in the “roles” file.

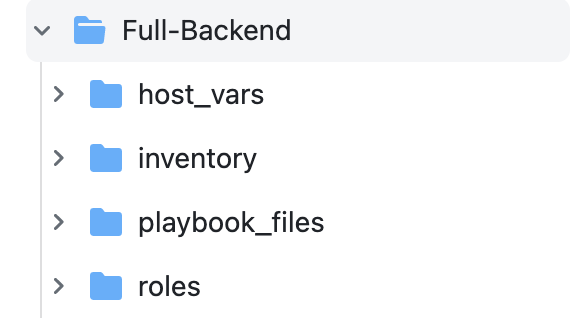


Figure 1. Directory breakdown of Ansible scripts

The first use of Ansible for the project starts with “terraforming” the virtual machines. These include spinning up multiple machines and installing all the necessary dependencies required. Tasks are broken down to small and distinct parts known as roles. In this case, as shown in the GitHub repo, these roles include:

* Creating security groups for the instances

This task creates all required security groups for the instances. As stated above, the team has two, and this creates them all simultaneously

* Creating volumes

This creates the volumes with pre-specified storage spaces. For each node, the team has created one volume each, with CouchDB master and Twitter Analysis having the largest usage

* Creating the instances themselves

This task spins up all the virtual machines that the team needs. It connects all the necessary parts such as its designated security group and volumes as well

* Config/tasks

This role installs pip, python, and it also creates a volume mount point (/data) that stores persistent data. This role also automatically mounts the volume to that mount point permanently, as without doing so, the volume would simply disconnect after every reboot.

The true value of Ansible here lies with the speed at which this task can be accomplished. In the beginning, OpenStack was not functional due to bug issues, and the team began creating these instances through the Melbourne Research Cloud dashboard itself. Therefore, all these tasks mentioned above had to be done manually. The redundant work and slow creation process were completely removed once the issue with OpenStack was fixed, and the team utilised Ansible for the remaining creation of the virtual machines.

The most important use for Ansible, however, surrounds dynamic deployment of our software containers. Major components above including such as harvesters and databases are all built with containers, and so using Ansible, these can be pulled, built, and run across various machines with a single command, as shown in the section above.

* 1. Dockers

Docker is a container technology, allowing for top-level virtualization. Flexibility achieved as we first build base containers. To deploy dockers to specific tasks, simply change the environment variable. Docker also ensures all versions are the same, so that the systems are not incompatible. As stated above, all software components built are containerised and deployed.

To build the Harvesters, a script and a Docker image is first created locally. This involves installing the Mastodon Python API, and all other required dependencies on our local computer, and once the script has been tested to function properly, an image of this is then pushed to Dockerhub. Next, Ansible is used to pull those images onto the different virtual machines and start the containers. To harvest from both Mastodon AU and Mastodon Social, the environment flag such as API token and the target URL is changed on each command of docker create.

Example Dockerfile for the Harvesters:

*FROM python*

*WORKDIR /usr/app*

*COPY . .*

*RUN pip install Mastodon.py-1.8.1-py2.py3-none-any.whl*

*CMD ["python", "harvester.py"]*

Building the containers for CouchDB was even simpler. We simply use Ansible to run docker pull from the official repository and once again, environmental flag such as IP addresses are changed to create different services across these virtual machines. To allow for clustering and communication between these docker containers, ports are exposed, but for security reasons, these could simply be improved by using a web engine such as nginx without having to expose too many ports.

A picture containing screenshot, text, diagram

Description automatically generated

Figure 2: System Architecture with Design Tools

1. Cost and Benefits Analysis of Tools

Before implementation, the team has carefully selected these technology to allow for as much flexibility and scalability as possible. As mentioned throughout the report, and demonstrated in the Ansible video attached, the system can scale with ease, and just as importantly, these services can be stopped and scaled down on the fly. Furthermore, instead of building everything from scratch, pre-existing official Docker images such as CouchDB can be pulled instantly, and the database cluster were set up relatively easily.

To ensure flexibility and fault tolerance, our team has looked into deploying these services such as the Mastodon harvesters with Docker Swarm to ensure that when a process goes down, more are spun up to take its place, ensuring the least amount of downtime for the system.

However, a major drawback of it all is system integration, networking, and security. Because there are so many moving parts, it required a vast amount of technical knowledge into all these different fields. This created a steep learning curve, and thus rendered these benefits unreachable and unfeasible for smaller scale project. Next, sharing data to and from different virtual machines such as a database cluster required rigorous settings to ensure that the nodes within the networks are reachable. Finally, as the number of components such as containers and virtual machines go up, this also means that we are exposed to more vulnerability, and so security concern was always part of our task planning.

1. Challenges and Errors Handling

With all the drawbacks listed above, our team has had to overcome a few major

challenges in the project. First of all, these were due to technical issues outside our control. This includes the OpenStack problem mentioned earlier and more dauntingly, the virtual machines faced an unexpected downtime towards the end of the assignment. Thankfully, all these were not fatal to the project, as all data are stored in persistent volumes, and by switching the volumes over to a working VM, we were able to circumvent the issue.

Another problem that we found was that all harvester containers kept going down. This may be due to the case with our setup, but we find that when harvesting from a different server apart from our chosen one, there was no such issue. Of course, to navigate this, the tools above such as DockerSwarm allow for a much smoother process. Furthermore, we also deployed the same services (i.e, Mastodon harvesters collecting from the same server) across our virtual machines as an added backup.

Finally, when setting up the front end, technologies such as Tableau showed signs of conflict when multiple data of different variety (SUDO, GEOJSON, etc.) were being pushed to it. These then involved manual manipulation of data such that they were compatible. However, this took considerable amount of time, and formed one of the largest weak points of the system design.

1. Lessons learnt throughout the project

When dealing with these data, the team has become more appreciative of how Big Data is actually handled. Big Data doesn’t just mean that the volume of it is large. Throughout, we had to deal with data streaming such as Mastodon to the database cluster (Velocity) and as mentioned above, incorporating and processing dataset from large range of variety. This has been quite rewarding, and it demonstrated vividly why these skills are highly sought after.

On top of dealing with difficulties surrounding data, there are also security issues. As we have chosen to export ports across containers and our virtual machines, these may not be feasible for real-world project with sensitive data. As such, when scaling is factored in, this concern is all too real and is a difficult task that demands constant focus.

Finally, the cost associated with the services are non-trivial. While Melbourne Research Cloud is free, it has limited capabilities and are not publicly accessible. Available services for hire such as Amazon and Microsoft, on the other hand, while powerful, demands a very high premium. The cloud technology does no doubt provide unlimited range of flexibility and scalability, as demonstrated in the project, but with its high cost, it still remains inaccessible to the majority.

1. Study Limitation

Before moving on to the result, it must be noted that this study is not definitive and may contain high level of bias. First and foremost, the data collected and analysed were small, only up to a few thousand texts/sentiments per scenario. While these are real results backed by research, authoritative data, and recent news articles, these are nonetheless too small of a sample to ensure a good representation of real public opinion. Furthermore, there also is the social media bias. It is known that social media itself may not be the best-known source for positive sentiments, as users are more inclined to complain and be critical on these platforms (Twitter, Mastodon). Therefore, these negative sentiments collected as part of the study may just be the result of this behaviour.

1. Full Data Collected

As part of the analytics, the total data obtained from both social media sources include:

* Twitter: all data containing bounding box amounted to around 800,000 tweets. Of that, they were broken down to our individual scenario with each one having close to 1000 tweets left after filtering. These are then clustered to their closest suburbs/boundary to find average sentiment score and find patterns
* Mastodon: in total, around 50,000 toots were collected on the topic. However, most of them contained information too vague to make meaningful inference. Therefore, after filtering out, we’re left with approximately 200 toots per scenario. The filtering process had to be aggressive in order to retain relevant information.
* SUDO dataset: we used 2 datasets in total from SUDO, namely to back our medical and rent scenario
* VicGov dataset: we used one set of data that points to land prices across Victoria

1. Scenarios Analysis

We have covered 4 scenarios in this which are described below.

* Land rate affordability across Melbourne.
* Rental affordability index across Melbourne.
* Medicare expenses for suburbs in Melbourne.
* Travel cost across Melbourne.

**Scenario-1**

**Land affordability index across Victoria**

The main focus of this scenario is to compare the sentiments across suburbs sourced from tweets that were then stored in CouchDB in the form of Views against Median Land Price in each suburb sourced from Spatial Urban Data Observatory or SUDO. The aim is to deduce if there is a correlation between average sentiment and Median Land Price region wise in Victoria.

**Why we choose this?**

The study of land prices in Victoria areas provides useful information about house affordability. Rent affordability has a big impact on the cost of living because of the city's prominence as a tourist and educational destination. Individuals, politicians, and researchers can make informed housing decisions and obtain a better grasp of the affordability of acquiring property in various locations of Victoria by knowing the real estate market and land prices.

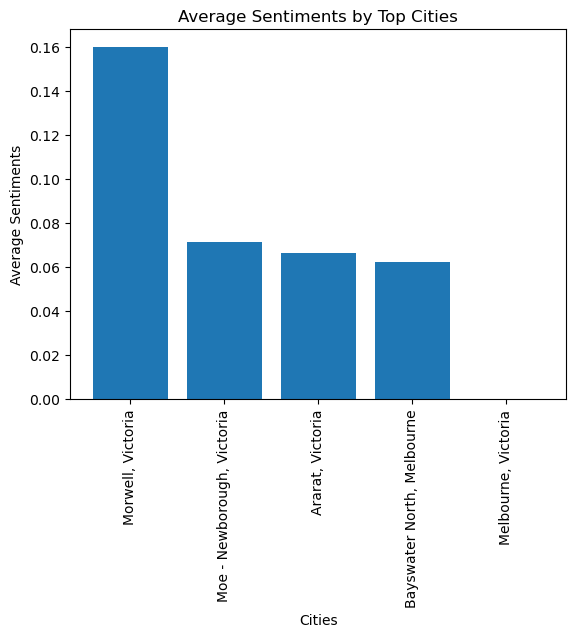
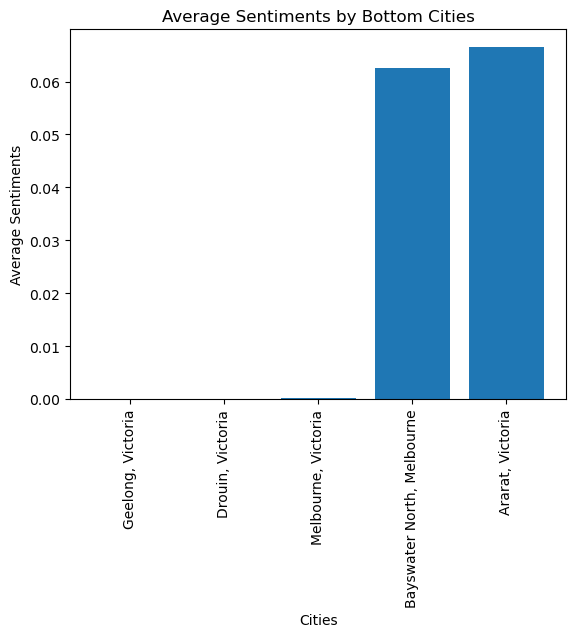
**Data Sources**

The data for this project was sourced from two different databases:

* CouchDB: This database contains a collection of tweets and Mastodon toots that were filtered out for Melbourne/Victoria. These were stored in CouchDB in the form of a View, which are a type of database index that allows for efficient querying of data.
* Spatial Urban Data Observatory: Victorian Government Land Rate.

**Methodology**

The tweets and Mastodon toots were first filtered down based on places to get Melbourne/Victoria data. Then, we grouped the data by place/suburb and determined the average sentiment score using the sentiment key available in each tweet. The Average land price in each suburb was then calculated using the ‘Median Land value’ attribute available in the dataset. The average sentiment and Median Land value in particular regions were then compared to see if there was any correlation between the two variables.

In the list of places analyzed, the average sentiments vary across different suburbs. Melbourne, Victoria has an average sentiment of 0.0001, indicating a neutral sentiment. Morwell, Victoria has the highest average sentiment of 0.16, suggesting a positive sentiment. Taking a look at SUDO data, we see that the price of land is far greater in areas surrounding Melbourne, and so the result obtained from Twitter is to be expected.

**Results**

We have plotted the top 5 sentiments and bottom 5 sentiments based on twitter data. After analysing the average Median Land rate(MLR) with respect to sentiments calculated we came to the conclusion that Morewell, Victoria has the highest sentiment of 0.16 followed by Newborough, Victoria and Ararat, Victoria with 0.07 and 0.06 respectively, While the Geelong, Drouin and Melbourne in Victoria has the lowest sentiments score with score of 0.00 each. This twitter sentiment score when compared to SUDO data on Median Land rate of different regions we concluded that Morewell has MLR of 280k while Newborough and Ararat has MLR of 359k and 313k leading to Top 3 positive sentiments whereas Geelong, Drouin and Melbourne have MLR of 1000k, 570k, 1492.5k respectively.

The results of this analysis suggest that there is a positive correlation between average sentiment and average land affordability of different regions in Victoria. Popular cities like Geelong and Melbourne comes with a high cost of living, and therefore, it is reasonable to see this such low sentiment towards them.

**Scenario-2**

**Rental affordability index across Victoria**

The main focus of this scenario is to compare the sentiments across suburbs sourced from tweets that were then stored in CouchDB in the form of Views against average Rental expenditure index in each suburb sourced from Spatial Urban Data Observatory or SUDO. The aim is to deduce if there is a correlation between average sentiment and average rental affordability index suburb wise.

**Why we choose this?**

Analyzing the Victoria Suburban Rental Housing Affordability Index provides valuable insight into the cost of living and rental housing affordability in that particular area. Tourism and education are Australia's largest sectors. Many international students come here for educational purposes. Rent affordability therefore plays an important role when analyzing whether the cost of living in Victoria is high.

**Data Sources**

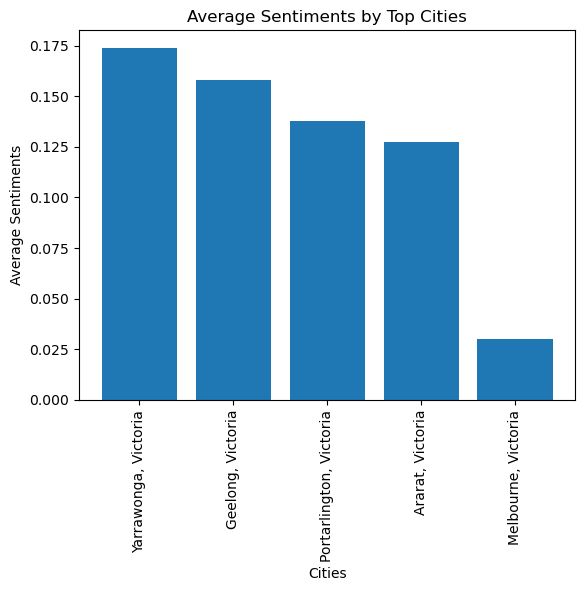
* CouchDB: This database contains a collection of tweets that were filtered out for Melbourne/Victoria. The tweets and Mastodon toots were stored in CouchDB in the form of a View, which are a type of database index that allows for efficient querying of data.
* Spatial Urban Data Observatory: This database contains a variety of data about Melbourne, including the average Rental expenditure index in each suburb.

**Methodology**

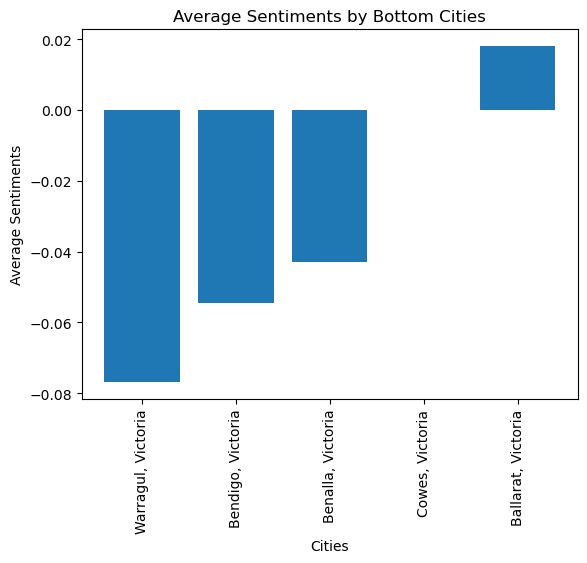
The tweets were first filtered down based on places to get Melbourne/Victoria data. Then, we grouped the data by place/suburb and determined the average sentiment score using the sentiment key available in each tweet. The average rental affordability index in each suburb was then calculated using the ‘RAI’ attribute available in the dataset. The average sentiment and average rental affordability index in particular regions were then compared to see if there was any correlation between the two variables.

**Results**

Among the listed places in Victoria, Australia, Melbourne, displays a slightly higher positive sentiment with a score of 0.0302 when compared to other cities. Provided that these data were collected during the COVID crisis, rental prices in these major cities such as Melbourne were driven down, and this was reflected in the affordability index. As such, it was reassuring to see this positive correlation.



In places less affected such as Bendigo and Warragal, as rental prices were not driven down as much, we see rather a rising negative sentiments. However, even with these results, it should be noted that the positivity is not necessarily reflected by price alone. Various factors such as landlord-tenant relationship, neighborhood quality, facilities, and amenities, and more were not checked in the analysis, and could be influencing the results.



**Scenario-3**

**Medical Expenses across Victoria**

The focus of this scenario is to compare the sentiments across suburbs sourced from tweets that are then stored in CouchDB in the form of Views against average Medicare expenditure per person in each suburb sourced from Spatial Urban Data Observatory. The aim is to deduce if there is a correlation between average sentiment and average expenditure per person.

**Why we choose this?**

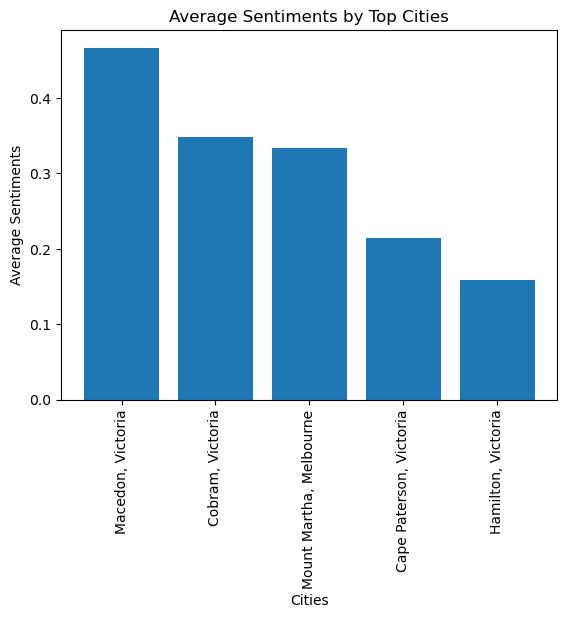
Analyzing the Victoria Suburban Medical Expense provides valuable insights about the overall cost of living in Melbourne/Victoria. Looking at overall sentiments about Medicare expense and analyzing the average expenditure on Medicare in each suburb is a good indicator on how affordable and accessible the medical system is to common individuals in Melbourne/Victoria.

**Data Sources**

* CouchDB: This database contains a collection of tweets and Mastodon toots that were filtered out for Melbourne/Victoria. The tweets were stored in CouchDB in the form of a View, which is a type of database index that allows for efficient querying of data.
* Spatial Urban Data Observatory: Average Medicare Expenditure

**Methodology**

The tweets and toots were first filtered down based on places to get Melbourne/Victoria data. Then, we grouped the data by place/suburb and determined the average sentiment score using the sentiment key available in each tweet. The average Medicare expenditure per person in each suburb was then calculated using the Average Medicare Expenditure attribute available in the SUDO dataset. The average sentiment and Average Medicare Expenditure in respective regions were then compared to see if there was any correlation between the two variables.

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

Overall, this analysis provides insights into the sentiments expressed in tweets related to various locations in Victoria, Australia. The sentiments range from negative to positive, depicting the diverse range of opinions expressed through Twitter.

**Results**

The results of this analysis suggest that there is a positive correlation between average sentiment and average expenditure on Medicare per person. For instance, we observe that places like Hastings, Mount Martha that have average expenditure of about 102.38 have positive sentiments towards healthcare expenditure as compared to Suburbs such as Bundoora and St Albans that have lesser expenditure on healthcare but have negative sentiments around healthcare. This suggests that with higher premium, people are happy with the level of care and service they are receiving. However, as Medicare covers/reimburses these expenses, these sentiment figures could be misleading. What is also very interesting is that as the location of these posts move further away from Melbourne, the more negative they become. This suggest that the lack of service becomes more and more severe as access to better healthcare diminishes with distance.

**Scenario-4**

**Travel across Victoria**

The focus of this scenario is to analyze the sentiments around public transport in Melbourne/Victoria across suburbs sourced from tweets that were then stored in CouchDB in the form of Views.

**Why we chose this?**

Analyzing the Victoria Suburban Transport Sentiment provides valuable insight into the cost of living in Melbourne/Victoria. Tourism and education are Australia's largest sectors and hence public transport becomes an important factor when it comes to going around the city.

**Data Sources**

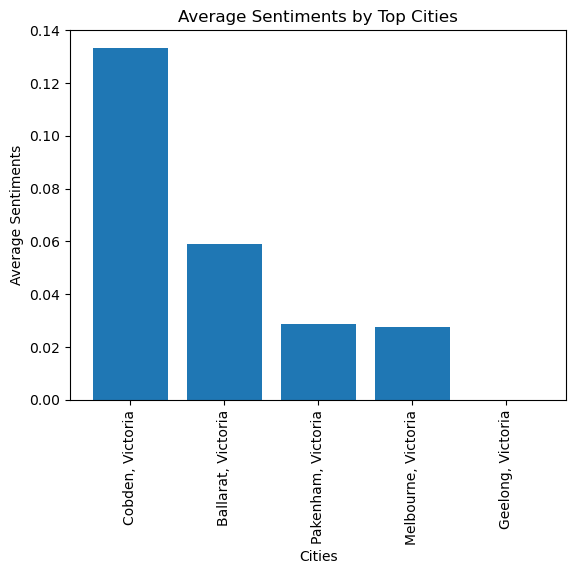
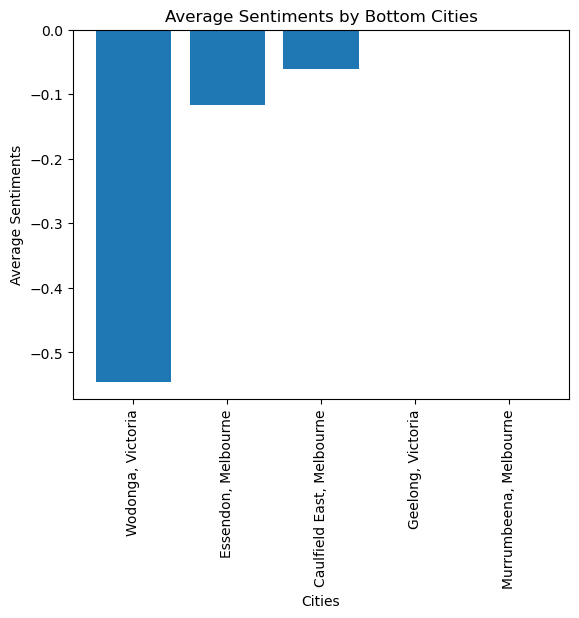
* CouchDB: This database contains a collection of tweets that were filtered out for Melbourne/Victoria. The tweets were stored in CouchDB in the form of a View, which are a type of database index that allows for efficient querying of data. For this scenario, we were unable to collect relevant Mastodon data for a more up-to-date analysis.

**Methodology**

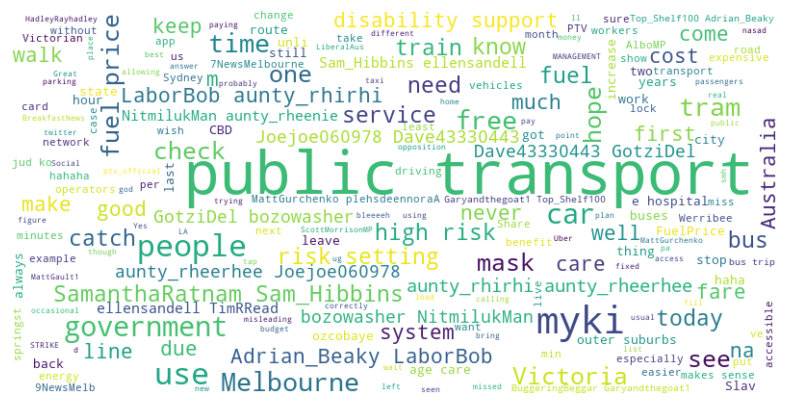
The tweets were first filtered down based on places to get Melbourne/Victoria data. Then, we grouped the data by place/suburb and determined the average sentiment score using the sentiment key available in each tweet.

**Result**

Overall, the list above includes places with a mix of positive, neutral, and negative sentiments. Melbourne, Ballarat, and Cobden have generally positive sentiments, while Essendon, Caulfield East, and Wodonga have slightly negative sentiments. Geelong, Murrumbeena, Pakenham, and Morwell are associated with neutral sentiments.



This, again, is not surprisingly at all. Under the travel scheme, the price of travel is constant, with the maximum cap of a little over 9 dollars per day. However, as people move away from the city, transportation issues such as rail work, line disruption, and irregular delays become much more present, and that is why we see these negative sentiments arising from frequently affected areas (Caulfield, Essendon, etc.)



We also observed that most tweets related to public travel mentions the words “public transport” and “myki” along with words like “expensive” and “cost” which suggests that public transport in general is a little costly in Melbourne/Victoria. Upon analyzing the overall average sentiments in Melbourne/Victoria with respect to travel, we observe a negative trend as seen in the graph above where the average sentiment is ~ -0.5 as compared to the positive sentiment ~ 0.14. This suggests that there are more people unhappy about public transport prices in Melbourne/Victoria.