Cluster and Cloud Computing (COMP90024)
The University of Melbourne
Assignment 2

Team 49:
Navdeep Beniwal (1279517)
Aditya Desu (1000447)
Hieu (Nick) Huu (1329582)
Jonathan Latti (1083374)
Patricia Widjojo (913557)

Abstract

This study presents a cloud-based system utilising CouchDB for the database, Mastodon and Twitter harvesters, a RESTful backend API in Node.js, and a React frontend. The system investigates the relationship between housing, the economy, and happiness while comparing the diversity of user interactions on Mastodon and Twitter. By collecting and analysing data from various sources, including SUDO and ASX stock data, it aims to uncover patterns between housing conditions, economic indicators, and subjective well-being. The system provides a comprehensive framework for studying these relationships and offers insights for policymakers, researchers, and platform developers to enhance well-being and inclusivity in the digital realm. We also describe how our system can be scaled or deployed autonomously in future implementations.

Table of Contents

Project Introduction

Background

As part of COMP90024 Cluster and Cloud Computing Assignment 2 - Australia Social Media Analytics on the Cloud, this report details the creation and deployment of our post-COVID Australian Economy and Housing Sentiment Analyser. Australia has been grappling with a housing crisis characterised by soaring property prices and a lack of affordable housing options. This crisis has made homeownership increasingly unattainable for average Australians, especially in major cities. High demand and limited supply have contributed to rising rental costs, exacerbating the issue. The economy plays a significant role as well, as housing affordability is closely linked to income levels and job stability. We are interested in investigating how these factors combine to create happiness in individual Australians around the country.

Team and Roles

Team Member	**Roles**
Navdeep Beniwal	Docker, CouchDB, Ansible
Aditya Desu	SUDO data processing, Front-end developer
Hieu (Nick) Huu	Full-stack developer
Jonathan Latti	Mastodon and Yahoo Finance data processing, Data analyst
Patricia Widjojo	Twitter data processing, Ansible, Data analyst

Data Sources

Data Source	**Relevant Data/Servers**
Mastodon	Global server: mastodon.social USA-based servers: urbanists.social, universeodon.com Australia-based server: theblower.au Africa-based servers: convo.casa, mastodon.africa
Twitter	62GB of tweets provided as part of assignment
SUDO	AU_Govt_ABS_Census: Total Household Income (Weekly) by Household Composition, SGSEP: SGSEP - Rental Affordability Index - All dwellings for Australia

	(Polygon) Q1 2011-Q2 2021
Yahoo Finance	Bitcoin USD (BTC-USD) and All Ordinaries (XAO) Index historical data

Relevant Links

Code Repository: https://github.com/LattiVuitton/CloudHousingCrisisAnalyser
Video: https://www.youtube.com/watch?v=Q2B4XuQ-8Lc

Scenarios and Findings

Scenarios Overview

Through this project, we aim to answer the question "*How was the overall Australian sentiment on housing and the economy amidst post-COVID recovery?*" As Australia began to re-open to the world towards the end of 2021, 2022 marks the first year post this transition. To investigate our main question, we conducted an analysis using data from various sources listed in the Data Sources section, with an emphasis on the time frame between February to August 2022. We will refer to this time frame as the time period for analysis.

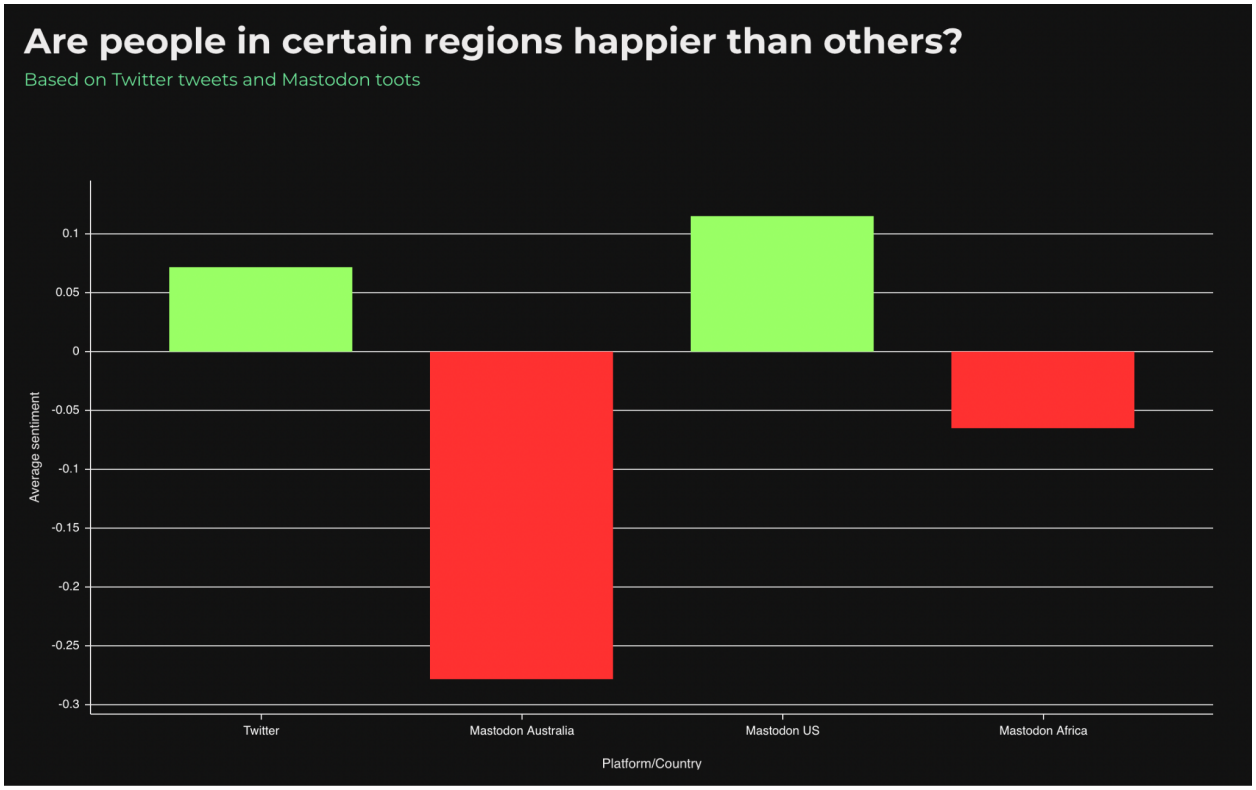
Our sentiment analysis is largely based on two significant metrics: the "compound" sentiment score from NLTK (Bird et al., 2009) VADER module, and offensive language usage. We will begin our analysis with this more generally in Scenario 1 before narrowing down further on the economy in Scenario 2.

Scenario 1: General

We first look globally to better understand general sentiment. Under this scenario, we use the Australian Twitter data, Global Mastodon toots, and toots specific to the Africa, US, and Australia region to explore the following areas:

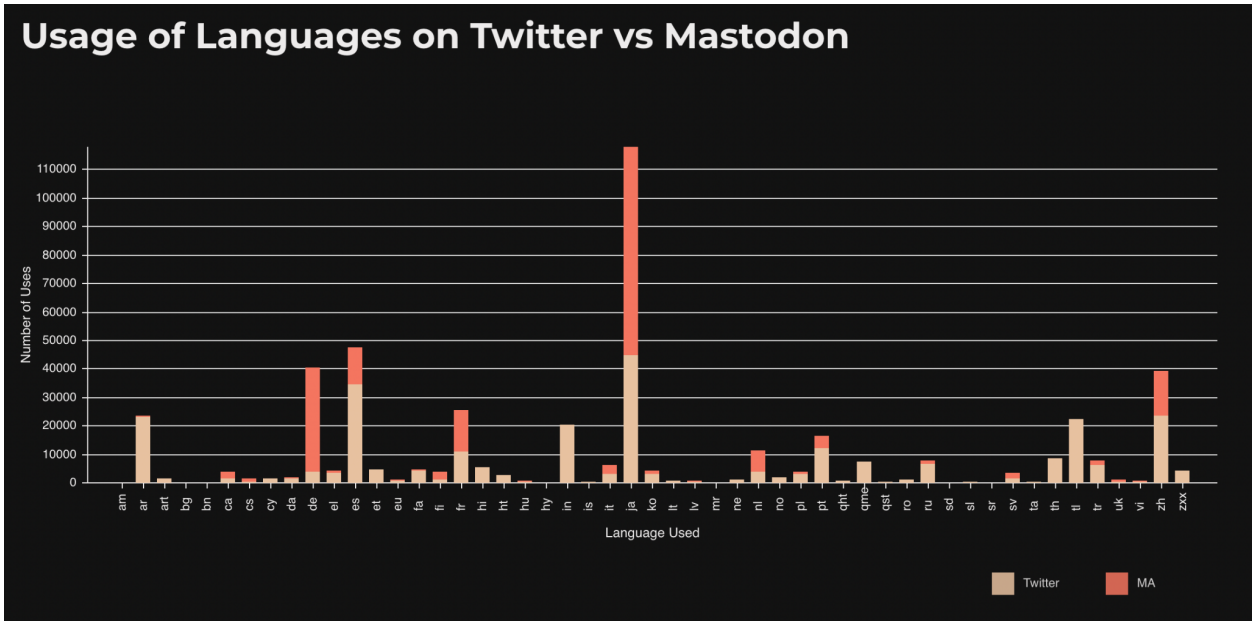
Scenario 1.1: Global Sentiment

Are certain regions more multilingual than others?
As seen in the graph below, there are differences between the various regions of the world in terms of average happiness, with Mastodon Australia seeming to be the least happy, whilst Mastodon US is the most positive. We also noted that Twitter Australia had a generally more positive sentiment than Mastodon Australia. It is worth pointing out that the servers are often misleading however, as people can post on a server based anywhere in the world, regardless of where they are physically located.



Scenario 1.2: Diversity in Languages between Twitter Users and Mastodon Users

We noticed that overall, Twitter seems to have a more diverse user base in terms of languages spoken, with many small groups appearing that Mastodon has no users for. Mastodon does, however, have a large number of non-English users in specific languages, such as Japanese and German, which seems to be a country rapidly adopting Mastodon over Twitter based on our findings here.

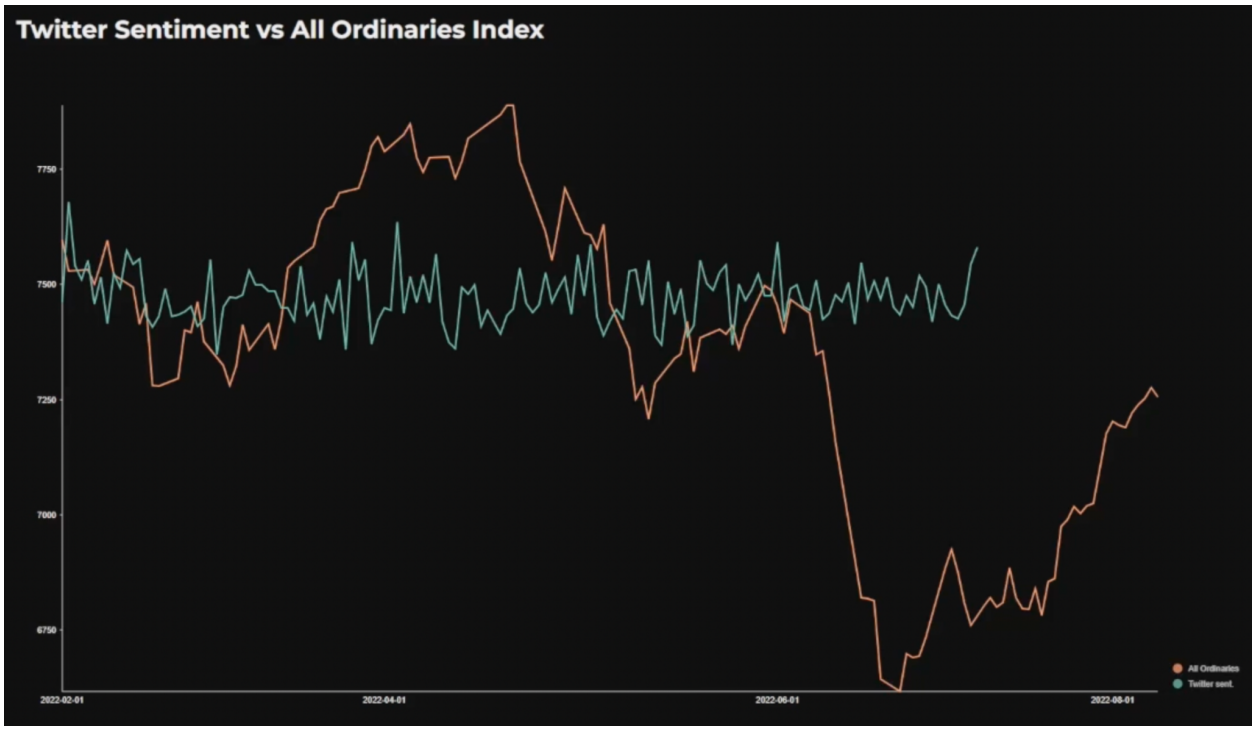


Scenario 2: Economy

Next, we focus on Australia and the economy. From SUDO data, we get average household income per Statistical Area Level 3 (SA3) and plot a heatmap based on the values, along with the average sentiment scores and offensive language use of tweets in that area during the time period for analysis. Additionally, we also look at historical All Ordinaries (XAO) index with average sentiment of all tweets, and historical Bitcoin prices (BTC-USD) with the average sentiment of tweets mentioning “bitcoin”.

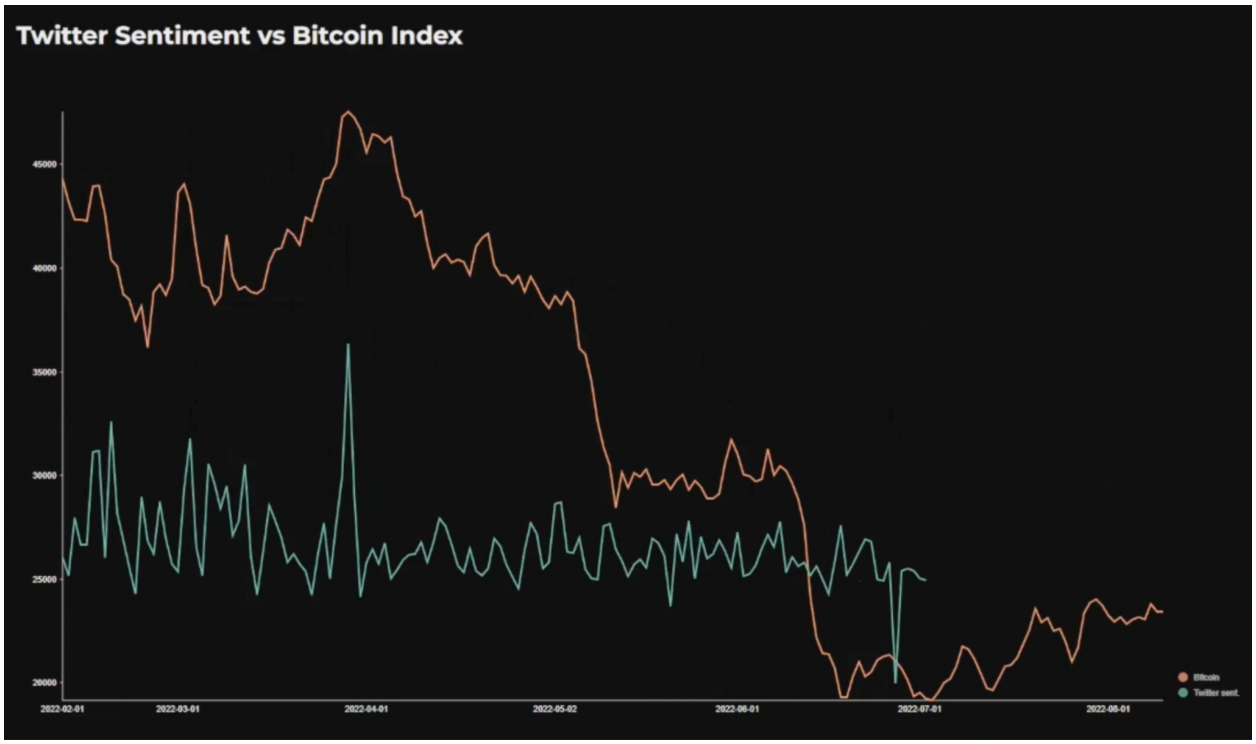
Scenario 2.1: Stock Index and Sentiment

As seen below, we considered how Twitter sentiment tracks against the All Ordinaries index, which is typically seen as a good measure for the performance of the Australian share market. In general, there was not a very strong correlation between the two trendlines, with Twitter sentiment seeming to be a very noisy curve with little identifiable movements. This largely makes sense, as we did not filter out tweets specifically related to stocks, so many people who are unaffected by the stock market (directly) are dominating the average sentiment scores.



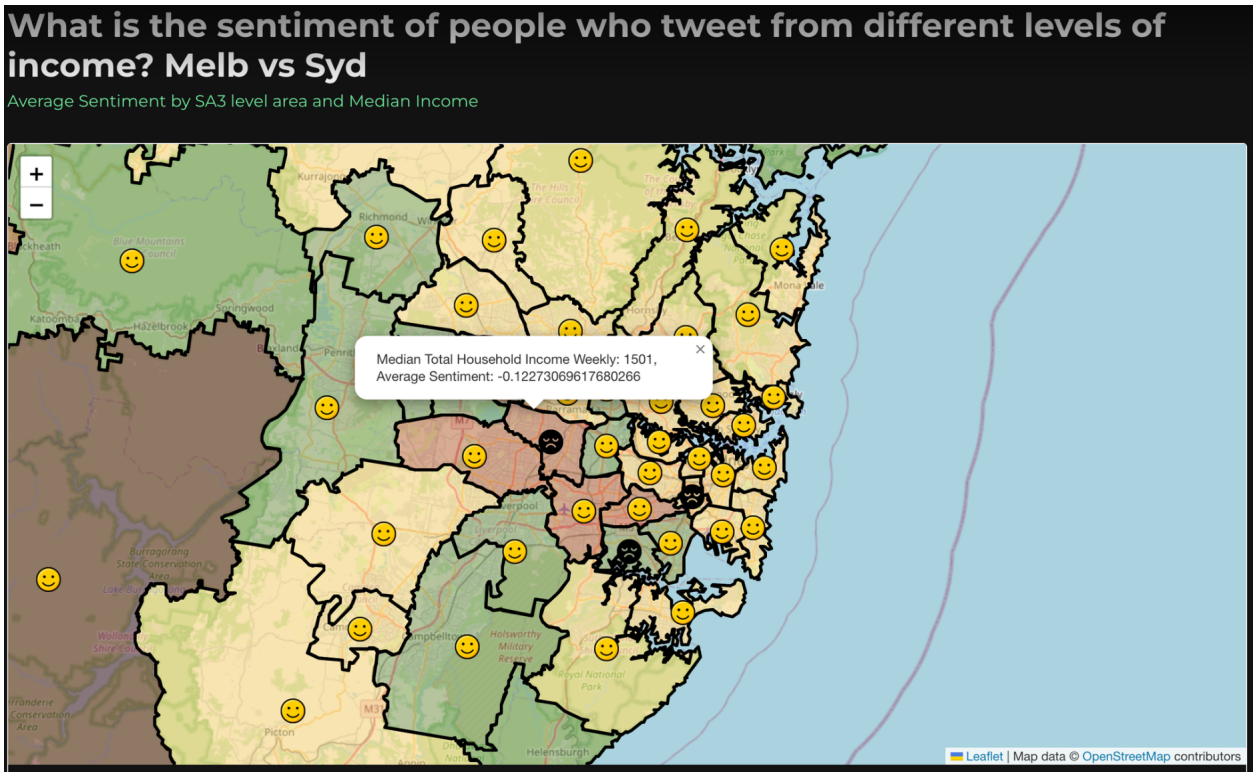
Scenario 2.2: Bitcoin Prices vs Sentiment on BTC

When considering only tweets which discussed the commodity in question, we noticed a stronger correlation. Whilst the general trends do not align clearly, there is an obvious spike in the sentiment regarding bitcoin which happens near the peak of the bitcoin price. There is also a similar effect at the bottom end, with a drop in sentiment where the bitcoin price hits its lowest price. It seems as if these trends typically trail the price, with movements in value happening before sentiment shifts.



Scenario 2.3: Income vs Sentiment

We found that the median happiness of people living in areas with high income is clearly higher than in lower income areas, such as further away from the city centres. It is worth noting that there was significantly more data in these city areas, so there may be a number of anomaly values in the regions with less data. We did not find clear distinctions between how Melbourne and Sydney reflect on these trends, with the more clear trend being that of city centres vs rural areas. As seen in the map below, the higher income areas near the cities often have the higher sentiments as well, though this is not particularly strong of a correlation.



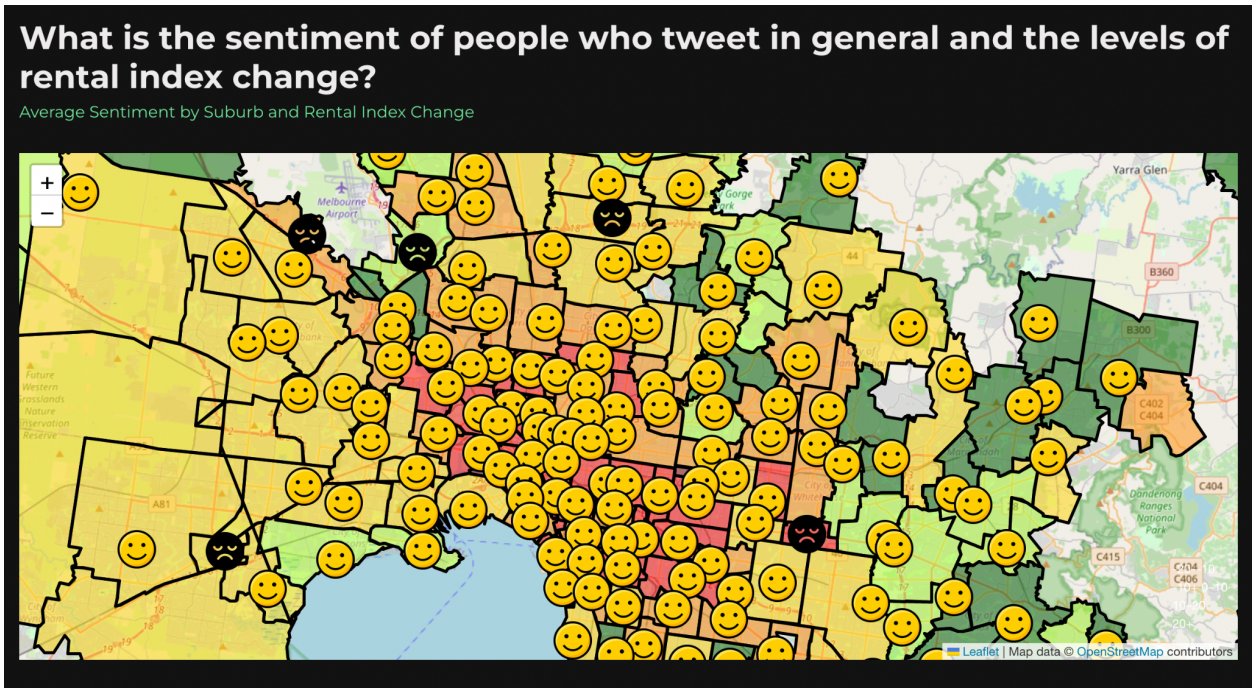
Scenario 2.4: Income vs Offensive Tweets

We found that offensive tweets were found everywhere across the capital cities, with both high income and low income areas having some amount of offensive posts. However, there is a slight bias towards high income areas having more offensive tweets. Of course, this is expected as a total count of offensive tweets, but it is also interesting that the proportion of offensive tweets in high income areas seems to be higher than in low income areas as well.

Scenario 3: Housing

Scenario 3.1: What is the sentiment of people who tweet against rental index

When we plotted the regions with sentiment against the rental index, which shows which regions have expensive rents and typically unaffordable living. We noticed in general that the least affordable regions are typically in the cities and that negative sentiment mostly concentrated in yellow to red regions indicating higher rental index increases. However, it is not true 100% of the time and there exist noise in the data. Consequently, a strong conclusion cannot be drawn from this analysis.

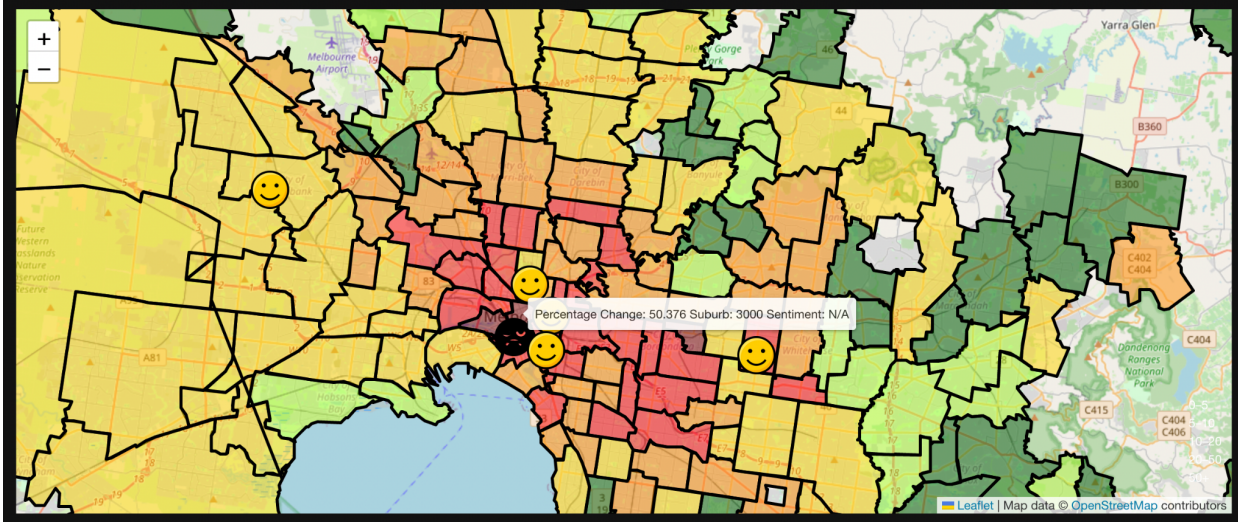


Scenario 3.2: What is the sentiment of people who tweet about housing against rental index

Looking at only tweets related to housing, we noticed that places with a high rental index did tend to yield the most negative sentiment, perhaps as a result of increased discontent at housing prices, although these trends were not very strong. In Melbourne, in particular, the relationship between negative sentiment and higher rental index increases can clearly be seen with the negative sentiment placed in the suburb with the highest rental index increase. From this we can say that a trend does exist, at least in Victoria. However, it is harder to reach the same conclusion in other states where although in general people who tweet about housing from areas with more affordable housing, similar to when considering all tweets, the trend itself is not very strong and there is clearly noise in the data, with a number of outlier results and regions with poor data values in general.

What is the sentiment of people who tweet about housing and the levels of rental index change?

Average Sentiment by Suburbs with Different Rental Index Change



User Guide

Repository Access

Our repository has been made publicly available in the following link:
https://github.com/LattiVuitton/CloudHousingCrisisAnalyser
The project can also be viewed on the following address: http://172.26.136.103:3000/

Requirements

As our automation code excludes instance creation and volume mounting, the following instance set-up was created through the MRC dashboard beforehand:

- Master-server
- Worker-server-1
- Worker-server-2: 100GB of volume attached
- Worker-server-3

Deployment Instructions

1. Ensure ansible is installed
2. Clone repository
3. Update ip of instances under ansible/hosts if required
4. Run **`cd ansible`**
5. Run **`./setup-environment.sh`**
6. From ansible/deploy-applications.sh, uncomment relevant parts to:
    a. Deploy CouchDB containers
    b. Setup CouchDB clusters
    c. Run Twitter, SUDO, and Yahoo Finance data processing
    d. Deploy SWARM services
7. Run **`./deploy-applications.sh`**

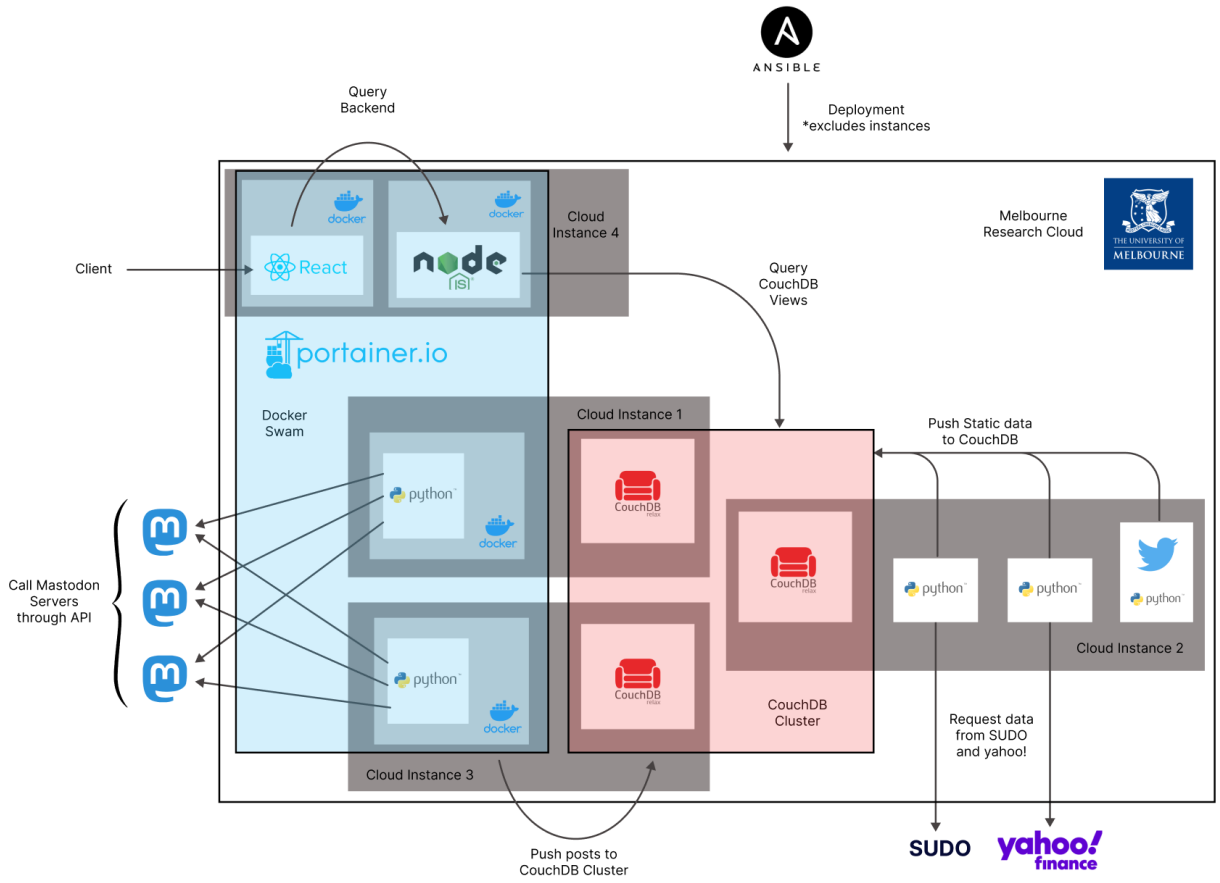
Scaling Instructions

1. From the project root directory, run **`cd ansible`**
2. To scale CouchDB clusters run **`./scale-couchdb-cluster.sh`**
3. To scale SWARM services run **`./scale-swarm-services.sh`**

System Design and Architecture

Introduction

The system in context has been designed and developed by keeping the availability of resources in mind i.e. the computation power along with the storage capacity, and to utilise these resources optimally to produce a system that is highly scalable and fault tolerant. In order to achieve that, the team has used various modern technologies based on COMP90024 team's recommendation as well as personal experience such as Docker, Docker Swarm, Ansible, CouchDb Clusters, Python and Javascript (running on a NodeJs environment) to build the system.



Detailed Architecture and Design Decisions

Base Infrastructure - Melbourne Research Cloud (MRC)

To form our based infrastructure, we have instantiated four Virtual Machines on Melbourne Research Cloud, each with a computing power of 2 VCPUS and up to 125GB of volume storage attached. These VMs have been set up and configured manually through the MRC Dashboard including instantiation, volume creation and attachment, and security rules configurations. This setup has formed the underlying basis on which other components along with their environments have been deployed using automated tools such as Ansible and Docker Swarm.

Components

The system architecture consists of the following components which interact with each other to facilitate the processing of huge volumes from multiple data sources efficiently and effectively.

SUDO Data Processor

The SUDO processor functions outlined here reads JSON data from two directories, sudo_rental_data_path and sudo_income_data_path, and uploads it to a CouchDB database. The function first defines a key map that maps old key names to new key names. It then iterates over each file in the directories, and for each file, it loads the JSON data and iterates over the features in the data. For each feature, the function updates the key names, checks if the feature has the required data, and if so, it adds the feature to a list of valid data. The function then sends the valid data to CouchDB.

SUDO- SGSEP - Rental Affordability Index Data

The dataset "SGSEP - Rental Affordability Index - All dwellings for Capital Cities (Polygon) Q1 2011-Q2 2021," presents the Rental Affordability Index (RAI) for all dwellings across Greater Capital Cities in Australia, excluding the Northern Territory, over a span of ten years. The RAI is a crucial indicator of housing stress, formulated based on the assumption that if a household's housing costs exceed 30% of its low-income gross, it's under stress. It uses the formula RAI = (Median Income / Qualifying Income) x 100, where 'Qualifying Income' denotes the income required to afford rent if rent constitutes 30% of the income. A score of 100 signifies a critical threshold for housing stress. The data was processed using QGIS, where the multi polygon for all cities was converted to a single polygon using the Singleparts to Multiparts tool, saved as a geojson file, and then zipped for CouchDB posting.

The rationale for choosing this dataset to analyse the relationship between tweet sentiment and regions with high changes in rental index affordability is likely based on the presumption that socio-economic factors, like rental affordability, can significantly influence public sentiment in those regions. Regions with high changes in the RAI might indicate instability or stress in the housing market, which could affect the general sentiment of the residents. Understanding these dynamics could provide valuable insights for policy planning, real estate market predictions, and social support initiatives. Moreover, as this data spans

over a decade, it can provide a longitudinal view of the impact of rental affordability on sentiment, accounting for temporal changes and trends.

SUDO- Median Income Data by Statistical Area 3

The dataset titled "Selected Medians and Averages from AU_Govt_ABS_Census" retrieved from SUDO comprises statistical data for the Statistical Area 3 (SA3) regions in Australia. The data provides valuable information about each region, including demographic details such as average household size, and economic data such as the median weekly rent and median weekly total household income. the original dataset was first converted into a shapefile using the 'Spatialise Dataset' tool from SUDO. This step allowed for the data to be viewed and analyzed in a spatial context. The shapefile was then converted into a geojson file using QGIS. Lastly, the geojson file was compressed into a zip file for efficient storage and posting to CouchDB.

The rationale for choosing this dataset to analyze the relationship between tweet sentiment and regions with different income levels is likely based on the presumption that economic conditions, such as household size, rent costs, and overall income, can significantly influence the public sentiment in those regions. Income disparity and housing affordability are significant factors that can influence public sentiment, and these factors can vary considerably from one SA3 region to another.

Twitter Tweets Processor

The tweet processor (twitter_data_processing/processor.py) outlined here is a function designed to handle and process Twitter data stored in a JSON format. It utilises several libraries and dependencies, including ijson, os, json, pandas, requests, and NLTK, to facilitate various data manipulation and analysis tasks. The NLTK library is employed specifically to access the Vader lexicon for sentiment analysis, which aids in determining the sentiment of individual tweets. Although sentiment scores are provided in the data by default, we implemented Vader as to remain consistent with our Mastodon toot analysis implementation.

The processor streams the input tweet data JSON through an ijson parser. A systematic approach was implemented by extracting relevant attributes from each tweet entry. These attributes include tweet ID, tokens, author ID, conversation ID, creation timestamp, context annotations, geo place ID, geo bounding box, full name of the place, public metrics (such as retweet count, reply count, like count, and quote count), tweet text, sentiment, and language where values are available. As the goal of the tweet processing step was to filter out tweets without geo details, emphasis was placed in ensuring that only tweets with non-null geo bounding boxes were saved into the CouchDB database. We will refer to these tweets as valid tweets.

A few other details were additionally analysed for each valid tweet to allow for more comprehensive analysis and to be used in our scenario analysis. These include the middle point of the bounding box, which we refer to as “geo_point”, “suburb” which is an identification of the suburb where the midpoint of the bounding box resides, sentiment scores determined using NLTK and Vader as “nltk_sentiment”, and binary classification for offensiveness of the tweet as “offensive”. These will be discussed below.

For efficiency on the front end, the suburb associated with each tweet was calculated before pushing them. Given that tweets do not have accurate locations, and instead have a bounding box, we made the simplifying assumption that the centre of the bounding box was the location of the tweet.To calculate the suburb, we checked for each tweet whether that centre point was in the bounding box of each suburb, since this is far more efficient than using the entire polygon in geoJson format. From here, each tweet is typically in the bounding box of around 1-3 polygons, which need to be checked in detail. This was done using a ray tracing method; essentially, if you draw a line from the point and it passes through an even number of edges in the polygon, then it is outside the polygon, and if it passes through an odd number of edges, then it is inside the polygon. It is worth mentioning that given the inaccuracy of the bounding boxes given by twitter, some of the points lay outside of all of the SA3 polygons (such as in the Ocean), and these were simply marked as null to be avoided by CouchDB views.

The SentimentIntensityAnalyzer module from nltk.sentiment.vader package was used in our additional sentiment analysis calculation. This module assigns sentiment scores to tweets based on the Vader sentiment lexicon.

The offensive tweet classification labels whether a tweet can be considered as offensive or not. This process includes analysing tweets to check if they contain any words from an offensive word list made based on a page moderation word list for Facebook (Parker, 2022). Prior to this approach, a pre-trained neural network model, tweetnlp (Ushio & Camacho-Collados, 2022) was implemented instead. However, having to predict whether each valid tweet was offensive or not resulted in our overall processing time to increase by over 20-fold, making it an infeasible method given the large amount of tweets to process. Consequently, we resorted to the simpler method of classifying offensive tweets.

Upon processing and enrichment of each tweet, the resulting data were converted into JSON format and saved to be sent as a bulk document of 500 tweets. Once 500 tweets are reached, with the exception of the last batch, it is sent using "_bulk_docs" to the designated database through HTTP POST. Each tweet ID is designated as the identifier sent to the database to ensure no duplicate data are saved. Bulk sending was chosen over sending each tweet individually to reduce the number of overall requests and networking overhead.

Where responses other than 201 or Created success response were received from the POST request, these requests are considered to have sending errors and saved in a list to be retried. Once all tweets are processed, we retry sending these tweets to our database again.

Overall, this tweet processor provides a reliable and efficient solution for handling Twitter data, enabling consistent sentiment analysis across our analysis and securely sending and storing the enriched data in a designated database.

Mastodon Toots Processor

The Mastodon harvesters described in this section are used to connect to various servers, and push Mastodon posts (toots) to the CouchDB cluster. Mastodon works in a different way to Twitter; instead of one centralised server, any individual can create a server, which others can use to post on. As such,

servers are usually specialised in a topic or location. For the purpose of our studies, we chose a number of servers from around the world and took the contents of these servers as indicative of the type of content coming from that region (it is worth noting that this is not a perfect heuristic, since anyone can post on any server, regardless of where they are).

The harvesters are Python-based scripts that rotate through the Mastodon servers requesting recent posts (40 most recent posts, since there is a limit on the API). Before pushing, the harvesters perform a number of data analysis operations: using a html parser (html_text), the text of the post is extracted, and sentiment is calculated using NLTK (with Vader Lexicon). This is the same library and package used on twitter, so the comparison between Twitter and Mastodon can be made accurately. Similar to the Twitter analysis, posts are also checked against the list of offensive words to gain a rough insight into how offensive each is (non-English posts are considered null for both offensiveness and sentiment).

Compared to the tweet metadata, there are a number of issues with Mastodon that make it difficult to perform analysis. For one, the language tags are of very poor quality, with many languages incorrectly labelled as others. This is especially problematic for tweets misclassified as English, which we then perform sentiment analysis on, and check for how offensive they are; in these cases, non-English words often match as ‘offensive’ or give inaccurate sentiment scores, since NLTK is English-only, and the map-reduce function used later to create views is unable to filter out these mislabelled posts. Perhaps a future direction for the project would be to manage the poor metatags provided by the Mastodon API through our own language-identification scheme.

The harvesters connect to the CouchDB cluster using HTTP and use the administrator password to gain access. Since the CouchDB is a cluster that handles replication, the specific IP each harvester pushes to does not really matter, although they all push to the main instance. Each post is checked for duplication in the CouchDB before it is pushed, since multiple harvesters are working in tandem and may retrieve the same post from Mastodon.

Each Mastodon harvester is containerised using Docker, which is deployed using Docker Swarm across multiple instances. This is to allow for flexibility around the number of containers running, which can be scaled up in future if the number of posts coming in every second increases to the point that the harvesters cannot keep up. One interesting issue we found whilst creating these containers was that there is often rate limiting on some of the Mastodon servers, which is based on the IP address. As such, we sometimes found that having multiple Docker containers running on separate IP addresses was necessary, since one IP would reach the maximum posts that it can pull from that server in a given time. At the current speed of posts coming in on the servers we used, this is the only potential hurdle, as the harvesters can easily handle the flow of posts other than as a result of rate limiting.

Yahoo Finance Data Processor

For comparing tweet sentiment against the stock market, we also created a harvester for stocks data, which runs in python. Given that the twitter data set was limited to a set time frame, we matched this instead of retrieving live data. The processor retrieves daily closing dates for a number of ticker codes, such as the All Ordinaries and BTC price, and pushes these to the CouchDB cluster.

CouchDb Cluster

CouchDb (NoSQL database) is used for providing the Database service in our system and stores an aggregation of data fed from multiple Data Processing units running on the system. It is used for its various functionalities that makes it suitable for big data processing such as Schema-less Document Storage, Scalability and Distribution, Data Replication and Syncing, and Map-Reduce for querying.

The CouchDb service has been deployed as a cluster of Docker containers (using official CouchDb image on DockerHub) running a CouchDb instance on multiple virtual machines (set to 3 VMs as default) which allows for scalability, fault tolerance and distributed data storage. Any client application with proper authorization can access the public interface of the CouchDb cluster on any of its nodes IPs'.

CouchDB Views and MapReduce

We use MapReduce for a number of views required for displaying data on the front end. For example, the view below is used to check whether a tweet is related to the topic of 'housing', and then emit the postcode and bounding box if it is. These are both required, since we are grouping our front end rendering by postcode, and performing the actual line rendering using the geo boxes.

```
function(doc) {
  var housing_related_keywords = ['housing', 'rental', 'landlord', 'tenant', 'lease', 'property'];
  var political_keywords = ['politics', 'labor', 'election', 'government', 'Labor'];

  var tweet_tokens = doc.tokens.split('|');

  var hasHousingKeyword = false;
  var hasPoliticalKeyword = false;

  for (var i in tweet_tokens) {
    if (housing_related_keywords.indexOf(tweet_tokens[i]) > -1) {
      hasHousingKeyword = true;
    }
    if (political_keywords.indexOf(tweet_tokens[i]) > -1) {
      hasPoliticalKeyword = true;
    }
  }

  if ('context_annotation' in doc) {
    for (var i in doc.context_annotation) {
      if (housing_related_keywords.indexOf(doc.context_annotation[i]) > -1) {
        hasHousingKeyword = true;
      }
      if (political_keywords.indexOf(doc.context_annotation[i]) > -1) {
        hasPoliticalKeyword = true;
      }
    }
  }

  if (hasHousingKeyword && !hasPoliticalKeyword) {
    if(doc.geo_bbox && doc.geo_bbox.length == 4 && doc.geo_bbox.every(function(x) {return !isNaN(parseFloat(x))})){
      var coordinates = [(parseFloat(doc.geo_bbox[0]) + parseFloat(doc.geo_bbox[2]))/2,
                         (parseFloat(doc.geo_bbox[1]) + parseFloat(doc.geo_bbox[3]))/2];

      var sentiment = parseFloat(doc.nltk_sentiment);
      // Emit a key that is an array of the coordinates and sentiment,
      // and the count as the value.
      if(sentiment != null){
        if(doc.geo_postcode){
          emit(doc.geo_postcode, {sentiment: sentiment, count: 1});
        }
      }
    }
  }
}
```

Node.js Back-end

The backend behind the React frontend was built using Node.js, and the RESTful API, so that the front end can query the API for data needed for visualisation. This backend also allows us to use stale data in cases where the views are still building; for example, since the Mastodon processor is working in real time, the view is constantly being updated, so the use of stale=ok allows us to keep the site live whilst new information is being built into the views.

To establish a connection between the Node.js backend and the CouchDB server, the backend utilises the "Nano" package. The "Nano" package is a Node.js library, which provides the necessary functionalities to interact with the CouchDB database. Using the "Nano" package, the backend can establish a connection to the CouchDB server by providing the appropriate server URL and credentials. Once the connection is established, the backend can perform various operations on the CouchDB database, such as fetching data and views. We then can efficiently manipulate and process before sending it to the frontend.

So. the properties of our backend API included:

- RESTful API: Provides a standardised way for the frontend to communicate with the backend. The API defines endpoints that the frontend can query to request specific data or perform operations.
- Asynchronous operations: Node.js, being a JavaScript runtime built on an event-driven architecture, excels at handling concurrent operations. The backend leverages this capability to handle multiple requests simultaneously, improving performance and responsiveness. Asynchronous operations are used when interacting with external systems (CouchDB), ensuring that the backend remains efficient and scalable. Particularly for the Mastodon data, we also incorporated real-time updates. So that, essentially, means that the backend continuously updates the views, as new posts from Mastodon are received, This real-time functionality enhances the accuracy and timeliness of the visualisations.
- Error Handling: The backend incorporates error handling mechanisms to catch and handle any errors that may occur during data retrieval, processing, or API interactions. It uses try-catch blocks and appropriate error response codes to provide meaningful error messages and ensure a robust application.
- One of the obstacles that we faced was correctly formatting the data, so that the “Nivo chart” and “Leaflet map” packages from the frontend application can identify and display the data accordingly. The errors could be as simple as reading input incorrectly or the ability to transform the data into required format from above-mentioned React packages.

In summary, the combination of Node.js backend, the RESTful API, and the "Nano" package facilitates the connection and interaction between the backend and the CouchDB server.

React Front-end

The front end was built with React, as mentioned-above, using “Nivo chart” and “Leaflet map”.

- Nivo Chart Library: Nivo is a popular charting library for React that offers a wide range of customizable and interactive chart components. It provides various chart types, including bar charts, line charts, pie charts, scatter plots, and more. Nivo charts are highly configurable and allow you to customise colours, labels, tooltips, and animations. The library also supports responsive design, making the charts adapt to different screen sizes.
- Leaflet Map: Leaflet is a JavaScript library for interactive maps that works well with React. It provides an easy-to-use API for creating and customising maps with features such as zooming, panning, and adding markers, polygons, and overlays. Leaflet supports various map tile providers, including OpenStreetMap.

By leveraging the Nivo chart library and Leaflet map in theReact frontend, we have managed to create visually appealing and interactive data visualisations. Nivo simplifies the process of incorporating different types of charts into the application, while Leaflet enables the integration of interactive maps with custom markers and overlays.
The React map was built with components, scenes and a router to conveniently connect everything together.

- Components: Components consist of various charts and maps which fetch the data from the backend.
- Scenes: Scenes are elements that represent specific pages or sections of the application. They consist of multiple components working together to create a complete user interface for a particular functionality or view (e.g. “Dashboards”). Scenes provide a logical grouping of related components and help maintain a clear structure for the React app
- Router: The router is responsible for handling navigation and rendering the appropriate scene/component based on the current URL. We use a routing library like React Router to implement client-side routing.

To analyse the processed data, we divided the visualisations into 2 sections based on the scenarios we investigated. For each scenario, the user can see either all of the visualisations, or open them specifically from the panel to get a closer look at each one. When the user first enters the page, two graphs from scenario 1 are shown on the home screen, as well as the number of tweets and mastodon posts. Note that the number of Mastodon posts is a live count, so if the user refreshes the page, this number should increase, as well as update the Mastodon visualisations.

For scenario 1, where we investigated how people from around the world use Twitter and Mastodon, we created two simple bar charts. The first one is a stacked bar graph, to simultaneously visualise the biggest (non-English) languages as well as compare between Mastodon and Twitter. The second one is a simple 1D bar chart, simply showing the average sentiment between Mastodon servers based in the regions we looked at.

For scenario 2, where we looked at how people’s happiness is affected by the economy, we had a more complex set of visualisations. Firstly, we had two line graphs, which plot sentiment against the All Ordinaries and the Bitcoin prices respectively, alongside the average sentiment for that day (BTC graph only includes people talking about Crypto). These sentiment scores are also scaled to fit on the same graph and be visually comparable. We also included four maps, which are interactive, and allow the user

to scroll around Australia and see how different areas compare. We used colours for our key visualisation here, with red symbolising negative elements, such as low sentiment scores or low income, whilst green/yellow reflect high sentiments and income levels. We also placed emojis on some of the maps as an easy way for users to understand sentiments in a visual manner. The maps were all rendered using leaflet.js, which includes zoom and move features for ease of use.

Infrastructure Management Tools and Technologies

Docker

Docker is used as the containerization platform for building and deploying several applications as containers in our system. This is done for the following reasons:

- Docker allows us to package applications along with its dependencies (including runtime environment) and to run it consistently across different devices or servers running a docker engine. This was very beneficial for us as we were all separately building the applications on our local devices and then deploying it on MRC. It also helped in avoiding conflicts that could have arised due to different versions of the common dependencies required by different applications.
- Docker simplified the deployment process to a great extent as any person responsible for deploying the application only required access to the application image and information of some basic configuration such as ports and services name.
- Docker permitted us to scale several of our applications such as Mastodon Harvester, Backend and Frontend applications which can expect a rise in traffic in the future. It also further allowed us to use orchestration services like Docker Swarm to horizontally scale these services seamlessly with the scope of automating the scaling processes based on incoming traffic.
- Use of docker has further added a scope of integrating CI/CD workflows such as Github actions in the deployment process which can allow for fast and seamless delivery of the latest features of the application to its users.

Docker Swarm

Docker native clustering and orchestration tool is used to set up a cluster of docker nodes (over all of the instances) and to deploy applications (expected to run indefinitely) as docker swarm services. It is done to utilise the builtin functionality of docker swarm to scale these services up/down horizontally, to optimally distribute these replicas among several vm instances, and to load balance the incoming traffic among the replicas without requiring much configuration.

Ansible

To facilitate dynamic deployment and extensions of the system, we implement Ansible scripts to all of our workflow post instance creation and volume mounting. The choice of not including Ansible scripts for instance creation and volume mounting was as per COMP90024 team’s recommendation given the existence of intermittent MRC issues for instance creation through Ansible at the time. All of the following actions can be run as required through the use of our scripts:

- Setup environment

    - Installation of common dependencies required all subsequent tasks such as docker, python, pip, vim, git, curl, and unzip
    - Configuration of user permissions to access docker demon along with connection reset as required
    - Addition of the team's public keys as authorised keys to allow for ssh access to all hosts.
    - Deployment of a Docker SWARM Cluster with one of the VM registered as the master and the other's registered as the worker nodes.
    - Deployment of Portainer (docker management and monitoring tool) as docker containers.
- Deploy applications
    - Deploy CouchDb docker containers on a predefined set of 3 VMs, setup a cluster between these three CouchDb instances and create empty databases to be used by the applications.
    - Deploy SUDO processor as described previously
    - Deploy Mastodon toots processor as described previously
    - Deploy Yahoo Finance data processor as described previously
    - Deploy continuously running applications (Mastodon Harvester, Backend & Frontend) as services on Docker Swarm.
- Scaling applications
    - Scale individual or multiple services running on Docker Swarm.
    - Scale couchDb cluster by adding a new node to the cluster.

Overall, automation of deployment processes using Ansible simplifies and ensures that the end-to-end setup to deployment process is executed smoothly and as intended. Additionally, through having clear naming conventions for roles and tasks, we enhance the ability to track progress when executing our Ansible scripts.

Portainer

Portainer is an open-source management tool for docker which provides a web based user interface to manage and monitor both docker containers and services deployed on docker swarm. It has been very beneficial in setting up the infrastructure around dockerized applications, debugging deployed applications, view system configurations and to easily monitor the health and activity of various docker entities including the volumes and networks.

DockerHub

DockerHub has been used to host images of the different dockerized applications on public repositories allowing ansible modules to pull the images directly from the repository without requiring to build the image on the host. This was done to facilitate both the development and deployment process as team members could share the images with each other to help in the development of their component and to work out the integration, and it helped us in avoiding any unforeseen problems that could arise during deployment in building the image on the host, which could be really difficult to debug.

Other tools

Git Version Control

Given the inherent collaborative nature of our project, we implemented git version control to facilitate seamless collaboration. When multiple people were working on the same feature, we additionally implemented branching to allow for easier merging and tracking of changes. Further, we ensured that commit comments were clear such that all team members were able to refer to the right commit and monitor progress when required. Overall, the use of git played a key role in preserving code integrity, minimising conflicts, and fostering a productive working process throughout our entire development process.

Key Strength of System

General

- The system is designed to make optimal use of the available resources. This is done by deploying the one-time data processing components directly onto the system without using Docker containers. This decision was based on the fact that these applications only needed to run once and didn't require scaling. By avoiding dockerization for these components, we were able to save computing and memory resources, allowing us to allocate them to other applications that could benefit from them. Furthermore, Docker Swarm has been utilised to distribute the different service instances across the cluster of vms based on resource consumption.
- The system also allows for high throughput given the usage of Docker swarm's out-of-box functionality to load-balance the incoming traffic among the different instances of the services.

Scaling

- Apart from the one-time data processing components, the rest of the stateless components can be horizontally scaled seamlessly given these services are deployed as docker swarm services. This scaling can be done manually either through running ansible scripts (with the target number of replicas for the desired service) or by using the Portainer UI (from master node).
- The CouchDb service is also scalable as it is running in a cluster, so given a rise in demand, more CouchDb instances running on other vms can be added to the cluster.

Fault Tolerance

- All of the system stateless components (except the one time data processors) are deployed as docker swarm services, whose inbuilt mechanism would instantly spin up a new instance based on the restart policy configured for the application whenever an instance goes down.
- The CouchDb service is also highly tolerant to faults resulting from a CouchDb Cluster built in data replication functionality which hosts multiple copies of the same data across different nodes on the cluster to avoid loss of data in case one of the nodes goes down.

Limitations and Challenges Faced

One of the issues with the MRC was that there was a limit on the number of instances and CPUs provided to us; given these limitations, we could not implement long-term, robust vertical scaling, as the amount of compute power is capped, regardless of how many Docker containers or VMs we create.

We also faced the cloud issues of Ansible not being able to create new instances, which meant we could only use the service to deploy within instances we have manually created from the dashboard. In future, we could potentially extend the project by using Ansible to also scale instances, which would allow vertical scaling alongside the horizontal scaling currently implemented.

A general challenge we faced throughout was the size of the twitter data, which has the potential to slow the front end, especially on maps. We handled this complexity in a number of ways, but largely through doing the heavy lifting in the backend, and using the front end simply to render. For example, the sums and counts of values, and the expensive checks of whether points are in polygons or not are all preprocessed. Despite this, however, we noticed some of the maps and charts were still slow to load at times, so we also implemented UI features to smooth this out, such as loading screens whilst maps are rendering

References

Parker, J. (2022, May 1). Bad words list and page moderation words list for facebook. Free Web Headers. https://www.freewebheaders.com/bad-words-list-and-page-moderation-words-list-for-facebook/

Ushio, A., & Camacho-Collados, J. (2022, November). TweetNLP: Cutting-Edge Natural Language Processing for Social Media. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Abu Dhabi, U.A.E.: Association for Computational Linguistics.

Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with python: Analyzing text with the natural language toolkit. O'Reilly.