Seamless Automated Data Analysis System: Weather, Sentiment, and Age - From Design to Insights

Cluster and Cloud Computing

COMP90042

The University of Melbourne

Group 66

Chris LIANG 1159696 Haoxuan LU 1157489 Cheng QIAN 1266297

Sheng LIU 1352368 Jingjun HAO 1365178

1. Introduction

The rapid growth of social media and the increasing availability of big data have opened up new avenues for understanding the complex factors that influence human emotions and well-being. In urban settings, where a large portion of the world's population resides, the interplay between environmental conditions, demographic characteristics, and the emotional states of individuals is of particular interest. This study investigates the potential impact of weather conditions and demographic factors on the sentiment expressed by residents in Melbourne, Australia.

To conduct this study, we utilize data from three primary sources: the Bureau of Meteorology (BoM) for weather data, the Mastodon social media platform for sentiment analysis, and the Statistical Urban Development Organization (SUDO) for demographic information. Combining these datasets, we aim to uncover potential correlations between weather factors, such as temperature, humidity, and wind speed, and the sentiment scores derived from social media posts. Furthermore, we explore how demographic characteristics, particularly age and region, may contribute to the observed sentiment patterns.
The significance of this research lies in its potential to provide insights into the well-being and emotional states of individuals in urban settings. By understanding how weather and demographics could influence sentiment, policymakers, urban planners, and mental health professionals can develop targeted strategies to improve the quality of life for city residents.

The following sections of this report will delve into the various aspects of the study. We will begin by discussing the team roles and system functionality, followed by a detailed explanation of the back-end architecture, including an overview of the system and the use of Elastic Search, Fission, and RESTful APIs. We will also share our experience working with the Melbourne Research Cloud (MRC) and discuss the error-handling techniques, such as addressing Fission package/function errors, function exceptions, and missing location data.

In the methodology section, we will present the aim of the study, the data sources used (weather data from BoM, Mastodon data, and SUDO data), and the data handling techniques applied to weather and Mastodon data. We will then discuss the results of our analysis, highlighting the essential findings and their implications. Finally, we will provide a conclusion, addressing the study's limitations and offering suggestions for future research in this area.

2. Team Roles

Chris LIANG: Back-end (Head), Back-end Demo, Technical Support
Haoxuan LU: Back-end, Technical Support, Video Integration
Cheng QIAN: Front-end (Head), Front-end Demo, Data Analysis
Sheng LIU: Data Analysis
Jingjun HAO: Data Analysis

Our team comprises five members, each bringing unique expertise to the project. Chris LIANG assumes the role of Back-end Head, overseeing the development and implementation of the back-end architecture and ensuring its robustness, scalability, and efficiency. Haoxuan LU works alongside Chris LIANG in the back-end team, contributing to designing and developing serverless functions, RESTful APIs, and database integration.

Cheng QIAN takes on the dual responsibility of front-end head and data analyst. As the Front-end Head, Cheng QIAN leads the development of the user interface (iPython notebook in this case. e.g., generate heatmaps that correctly illustrate the data), ensuring expected interaction with the service. Additionally, Cheng QIAN contributes to the Data Analysis efforts, leveraging his skills to derive meaningful insights from the processed data and generate the figures for analysis.

Sheng LIU and Jingjun HAO complimented the Data Analysis team, working closely with Cheng QIAN to explore the analyzed combined weather and social media data stored in the ElasticSearch database. They examined the correlations within the data, generating valuable insights.

3. System Functionality

The system developed for this study is designed to efficiently collect, process, and analyze data from various sources, enabling us to investigate the relationship between weather conditions, demographic factors, and sentiment in Melbourne. The key functionalities of the system are as follows:

1. Automatic Data Fetching: The system automatically fetches data from specified external sources, including weather data from the Bureau of Meteorology (BoM) and social media data from the Mastodon platform. This ensures that the data is consistently updated and available for analysis.

2. Automatic Data Storage: Once the data is fetched, the system automatically stores it in the database, in this case, the Elastic Search database. This centralized storage allows for

efficient access and management of the data throughout the analysis process.

3. Automatic Data Preprocessing: Before saving the data to the DB, the system automatically preprocesses it to ensure that the needed data is clean and ready to use. This preprocessing step may include tasks such as data formatting, cleaning, and transformation, depending on the specific requirements of each data source.

4. Sentiment Analysis: The system incorporates sentiment analysis functionality to process the text data collected from the Mastodon social media platform. This analysis assigns sentiment scores to each social media post, enabling us to quantify the emotional tone expressed by individuals in the Melbourne area.

5. Data Concatenation: To facilitate comprehensive analysis, the system concatenates the datasets from different sources, such as weather data, social media data, and demographic data. This integration allows for exploring potential correlations and patterns between the various factors under investigation.

6. RESTful API Interface: The system provides end-users a simple and intuitive interface through RESTful APIs. These APIs allow users to interact with the system, retrieve data, and perform specific analyses without complex technical knowledge.

7. Effective Communication between Functions: The system ensures effective communication between different functions and components through APIs. This enables seamless data flow and processing, from data collection to analysis and visualization.

The automated nature of the system minimizes manual intervention, reduces the risk of errors, and enhances the efficiency and reproducibility of the study.

4. Back-end Architecture

4.1 Architecture Overview

The diagram (Figure 1) illustrates the architecture of a service running on Fission, which is a Function as a Service (FaaS) platform installed inside the Kubernetes cluster. RESTful APIs play a crucial role in facilitating data exchange between the various components of the service. Here's a detailed explanation of the components and data flow:

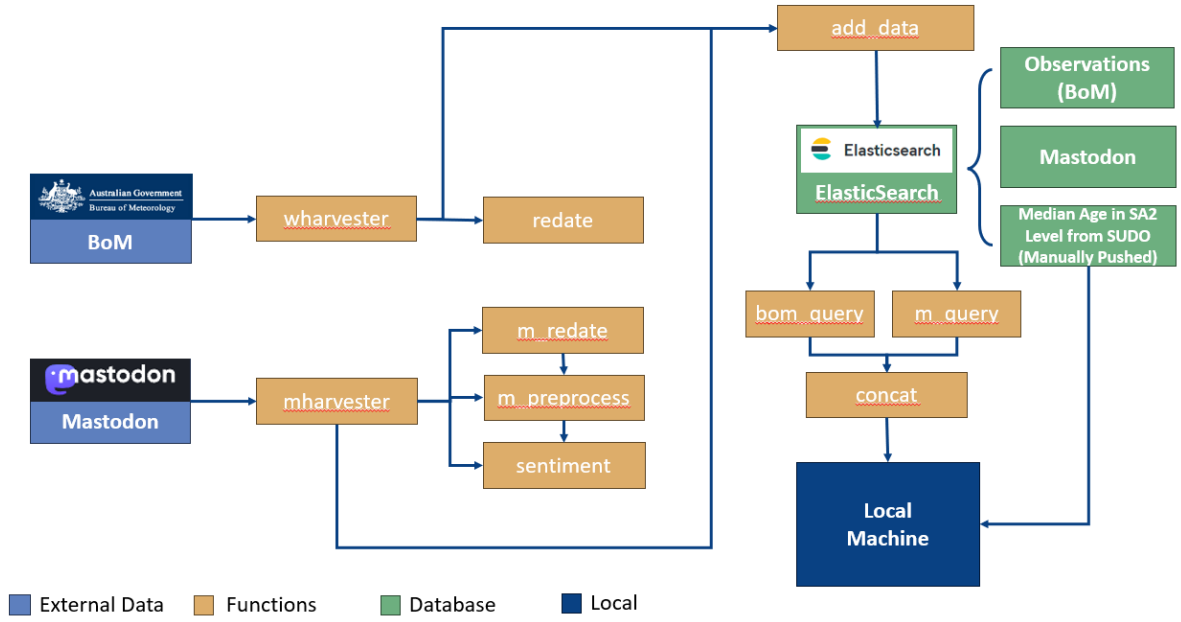


Figure 1 Illustrated Architecture of the Service

The data flow is initiated by the "wharvester" function, which collects weather data from the Bureau of Meteorology (BoM) through its API, and the "mharvester" function, which harvests social media data from Mastodon's API endpoints. These functions act as the primary data ingestion points, allowing the service to consume data from disparate sources in a standardized format by choosing only the needed attributes.

The collected data then undergoes a series of formatting and preprocessing steps, performed by dedicated functions such as "redate" (formatting the date to the same data type, same for m_redate) for weather data, "m_preprocess" and "m_redate" (m_ for Mastodon) for Mastodon data, and "sentiment" for sentiment analysis of the Mastodon data. These functions within the architecture adhere to loose coupling, modularity, and lightweight principles, which enhance the maintainability and extensibility of the service. By encapsulating specific data processing tasks within individual functions, the architecture promotes the separation of concerns and allows for each component's independent development, testing, and deployment.

The processed weather and Mastodon data are then stored in the ElasticSearch database. The ElasticSearch database serves as the central repository for storing data. In addition to the Observations (BoM) and Mastodon data, median age statistics at the Statistical Area Level 2 (SA2) are stored in the database. The dataset was manually forwarded to the database as SUDO provides the data. SA2 is a geographical classification used by the Australian Bureau of Statistics to represent medium-sized areas that are socially and economically coherent. The SA2 data provides a standardized way to link and analyze data based on geographic locations.

The architecture employs two query functions to serve data queries for further analysis or visualization on the front-end: "bom_query" and "m_query." These functions retrieve the relevant data from Observations (BoM) and Mastodon storage locations within the ElasticSearch database. The retrieved results are then concatenated through the "concat" function before being returned to the local machines via a RESTful API.

On the other hand, since we need to analyze the data at the scope of SA2 areas and explore the correlation between sentiment and age, there will be another RESTful API for accessing the SUDO dataset from SA2. This practice ensures that we have utilized the database and guarantees a consistent access protocol to data. Further analytics and display on relations with sentiments score and area-based would be performed with the data.

The service achieves a highly modular, scalable, and efficient design by leveraging Fission's serverless architecture and RESTful APIs. The serverless model automatically scales resources based on demand, eliminating the need for manual infrastructure management and enabling cost-effective operation. The modular architecture, facilitated using discrete functions and APIs, promotes loose coupling and enables the seamless integration and processing of data from diverse sources. This approach also enhances the resilience and fault-tolerance of the service, as individual components can be independently developed, tested, and deployed, minimizing the impact of failures on the overall system.

4.2 Elastic Search

Our Elastic Search database contains three indexes, as mentioned above, and each index comes with three shards and one replica. The querying speed can increase with more shards as each node will have less computation pressure while being queried. Three shards is an optimal number because our cluster contains a master node and three data nodes, making more shards would be uncesary as shards are only distributed on data nodes. At the same time, data in our indexes will be safer with replicas as they are node-failing tolerate, which means even if a node containing the data is dead, another node containing the replica can still respond to queries, maintaining the program’s functionality, hence promising service uptimes. By having shards and replicas, we have unleashed the power of a distributed database – load balancing (improving availability).

Detailing to the indexes, though both the observations and mastodon index store data from external sources, they don’t contain any attributes unrelated to our analysis and displaying tasks. Limiting the number of attributes in the index, though it requires more time on preprocessing, can significantly reduce the storage pressure on the cluster. Taking the BOM data as an example, the original dataset from one observation contains over 20 attributes,

while our task only requires seven, meaning saving over half of the storage space. This design could be handy when the data amount needs to be scaled up and deployed on a storage space-constrained machine. Considering that the BOM data only updates every 30 minutes, the current design certainly outweighs the design of harvesting and putting the whole dataset into the database as it does not require rapid data ingress.

Regarding saving and requesting data, the endpoint user does not need to interact directly with the database. Instead, our system provided many APIs to query and add data from and to the database, which will be discussed in the next section. There are a few advantages of adopting this design. The most notable one is simplicity. By abstracting the query and data upload process, instead of dealing with the complex Query DSL statements used by Elastic Search and configuring the Elastic Search client, the endpoint user can make an HTTP GET request to their designated API and retrieve the data they want (will be discussed more in section 4.4). Besides simplicity, using an API to access the database also benefits security. Having the endpoint user directly connect to the database means our database's secrets must be transferred and exposed to all clients, underpinning security concerns as secrets could be breached when transferred to and stored on the client.

On the contrary, using APIs to access clusters can hide away secrets from end-users and only store them on our server, eliminating the possibility of clients exposing the DB's secrets. Besides, providing APIs only can limit the client's access. Without DB's secrets and certification, end-users won't be able to perform sensitive actions like deleting data and indexes or injecting malicious data to the database (the add data function could potentially filter contents, keeping data in the DB safe. In addition, the API function layer can also serve as a security guard for DoS attacks. With APIs, rate limits and blacklists could be implemented to block malicious actors, and using Fission can further optimize the situation where the database needs to handle a large number of requests. Fission can automatically scale and process requests on any pods in the cluster rather than pressuring the master node of the Elastic Search cluster to handle everything.

4.3 Fission

From data harvesting to data processing, from data storing to data querying, Fission is used throughout our system. Each Fission function in our system essentially forms a microservice – each only does a simple operation, providing a service to a user and forming a larger functionality.

With Fission, our needed computation, which is traditionally done locally, has been transplanted on the cloud and managed by Kubernetes. In this manner, our application (front-end) will not need to care about computations and the infrastructure; it will only need to focus on displaying tasks.

Overall, except the function that puts data to Elastic Search is stateless and side-effect free; that is, our functions do not store the state of the client and won't modify the system state to

any extent, making each function lightweight and ephemeral, allowing end-users to feel as if the computation is done locally, following cloud computing's principles.

A huge benefit of using Fission is its scalability. Since Kubernetes manages Fission, a function can automatically scale up by using more pods when it needs more power or scale down to not using resources when idle (not used in any way). For example, if the "sentiment" function is needed by multiple parties to do sentiment analysis, the cluster can automatically allocate more resources to the function and handle all requests seamlessly. In contrast, if the function is implemented locally, a sophisticated multi-threading or multi-processing handling will be required to process multiple requests concurrently while also lacking the ability to scale horizontally – a feature that can be easily implemented on the cloud. By offloading most computational tasks to a cloud cluster, our system offers significant economic benefits to users. They can purchase a budget-friendly machine for the front-end and deploy the cluster for the back-end. It is typically far more cost-effective than acquiring a powerful machine to handle front-end and back-end operations.

Another advantage of utilizing Fission is due to its modularity. Dividing each functionality into a small part and implementing a Fission function helps decouple the system and makes each part more cohesive. Besides making the whole system easier to maintain, this design also ensures efficiency and availability. Regarding efficiency, compared to having a large component that includes every aspect of a system that requires loading the whole program into memory even if only part is used, implementing microservices using Fission solved the problem at its root – only the needed service will be loaded to the memory while keeping other services idle, which only takes a negligible amount of resources. This architecture could significantly benefit machines with limited computation power, especially with a small RAM. For the latter, availability is almost guaranteed in our system. Since nearly every service works independently, breaking a single service will not bring down the whole system, as other services are still functional. For example, even if a harvester is malfunctioning, other components, such as "concat", are still accessible to users, allowing them to analyze with previous data loaded in Elastic Search. In addition, if users wish, they can also utilize processing modules and the "add_data" module to process and upload the latest data to Elastic Search to analyze them during a harvester's downtime, ensuring flexibility.

4.4 RESTful APIs

RESTful APIs are crucial in our system as they are used in internal and external communication within our system and to the user. RESTful APIs implemented are designed to adhere to the REST API principles, which is easy to understand, short, static, and use standard HTTP status codes to convey results. In addition, it's been ensured that all GET calls are side-effect-free, making it safe for clients to send requests. Implementing RESTful APIs also reduced our system's computation since they are stateless on the server but managed on the user side.

Here's a list of APIs we used in our system:

URL	Method	Headers	Request Body	Purpose
/redate	POST	'Content-Type': 'application/json'	{"date": Str}	Reformat date in the BOM dataset to standard Elastic Search date format
/mredate	POST	'Content-Type': 'application/json'	{"date": Str}	Reformat date in the BOM dataset to standard Elastic Search date format
/mpreprocess	POST/GET	'Content-Type': 'application/json'	{"text": Str}	Remove HTTP delimiters from text
/sentiment	POST/GET	'Content-Type': 'application/json'	{"text: Str}	Get a sentiment score of a text
/adddata	POST	'Content-Type': 'application/json'	{"data": JSON file}	Add new data to the Elastic Search Cluster
/bomquery/{start_date}/{end_date}?size={size}	GET	None		Query the Observations index
/mquery/{start_date}/{end_date}?size={size}	GET	None		Query the Mastodon index
/concat/{start_date}/{end_date}?size={size}&bsize={bsize}&msize={msize}	GRT	None		Create a concatenated dataset from Observations and Mastodon index
/sa2/{sa2_code}/age	GET	None		Return the median age given a sa2_code
/sa2/age	GET	'Content-Type': 'application/json'	{"sa2_codes": []}	Bulk API, query median age for a list of sa2_codes

Table 1 Used APIs

Several optimizations have been implemented to reduce the length of the URL and query strings. For example, query data are designed to be encapsulated in the request body for

functions that need to process long inputs, especially for bulk-processing APIs. For example, the median age bulk query API will take a list of SA2_CODES using JSON instead of putting them in the query string, keeping the URL short.

With these understandable APIs, users can easily access each component of our system while ensuring easy maintenance – communications between each Fission function are also done by calling each other's API.

5. Experience with Melbourne Research Cloud (MRC)

In this study, our team had the opportunity to work with the MRC, and while we encountered some challenges, we also recognized the benefits it offers.

One of the main advantages of the MRC is that it utilizes OpenStack, which provides a graphical user interface (GUI) similar to commercial cloud platforms, such as creation and management. However, despite this similarity, we found that the MRC is still highly constrained compared to mainstream commercial cloud platforms, which impacted our overall experience.

The most notable issue with MRC is the complexity of spinning up a service. For example, installing services like Fission and Elastic Search on the MRC could become a nightmare for some users. Though, luckily, we're provided with installation scripts, a user must still ensure that their local machine has installed a lengthy list of packages and dependencies correctly before proceeding to install the required services. In addition, the unavailability of using a GUI to install required services is another barrier for users to access the cloud. Relying on a command line interface (CLI) could be challenging and time-consuming for users. For instance, our team has spent much time understanding what each command did and ensuring all members can access the cloud service.

Managing installed services is also time-consuming, continuing with the complexity of service creation. Taking Fission as an example, since MRC lacks a GUI to manage services, many repeated actions must be done during function creation and update. In contrast to commercial cloud service's serverless service, for example, Tencent Cloud, which one of our team members has used, supports direct editing and testing on the function management page, updating a function/package on the MRC requires a repeated process of zipping a package, updating the package, and updating the function for each modification to the function. Such inconvenience in managing also extends to cluster management. For example, if our Kubernetes cluster is underpowered and would like to extend horizontally, the process could be again challenging for first-time users as users must deal with a long list of command line operations.

On a smaller scale, from our project’s perspective, the MRC's insufficient public IPs made the situation worse. Many services could have been accessed more efficiently instead of following through a long set of commands, such as needing to connect to a VPN and using port forwarding.

However, on the bright side, MRC also brings some good experiences. Though not as convincing as commercial cloud platforms, the open-source GUI used by MRC still provided a few convenient operation shortcuts compared to a pure CLI interface. For example, generating public-private keypairs is substantially easier than using command lines.

Moreover, using Fission and Elastic Search for big data analysis is peace of mind. As Fission lives in the Kubernetes cluster, it supports automatic scale-up to the cluster's capacity when demand is high, which means there’s a manual process for designing complex algorithms. Elastic Search, a distributed database that also lives in the cluster, allows us to create shards, which can accelerate data querying, which is essential in big data analysis as querying speed could be a bottleneck when data is extensive.

Besies, our team developed a profound understanding of cloud infrastructure operations by thoroughly exploring the fundamentals of cloud components and gaining hands-on experience deploying serverless services and manually managing the cluster. This knowledge is invaluable, especially in functions, architecture, and API designs. In the future, it will be directly applicable when deploying similar services on commercial cloud platforms, both IaaS and PaaS, as they will only be simpler.

On a larger scale, free access for university students is a huge plus. As a self-owned, not-for-profit infrastructure owned by the university, we don’t need to worry about the expensive bills for services we use while also being able to gain rapid support from the teaching team.

6. Error Handing

6.1 Fission Package/Function Errors

Various errors have occurred throughout our deployment, most related to deploying fission packages and functions.

The most common error is failing to build a package, which can be due to multiple reasons. However, most of these issues can be resolved by delving into the “common issues” file on Canvas, such as not giving permissions to the `build.sh` file and having functions running over time.

However, some issues are more challenging to find and resolve. For example, our team has encountered many errors in returning non-serializable results, causing functions to raise an exception. In this case, we will need to scrutinize and delve deep into each line of the function and find out where the issue is – as using `fission fn test` on the cloud usually doesn't provide meaningful debug information.

Another prominent issue was failing to install some libraries on a custom package. The most common reason is that the Python version used in the default Fission environment is too low for some libraries, like TextBlob, required. In this case, we have to search for the most suitable library version, modify the requirements.txt file, and update the function and package. Nonetheless, some libraries face a more severe issue than these. For instance, a critical library used in our system, `geopandas`, is not usable on Fission as it requires a C dependency. Though we found a pure Python alternative, the performance is devastating, and putting it in Fission functions that require a quick response is nonsense. We then have no choice but to accept this defect and put this specific part running in the local machine until we develop a docker image that allows us to install this library and run our code.

6.2 Function Exceptions Handling

As an expected event, handling function expectations has been baked into our function development process. Specifically, the functions we wrote are robust in handling unexpected/empty inputs from APIs. By using the try-expect pattern provided in Python, guards can be placed on each parameter/data reading attempt and return an error code to the user before proceeding to the rest of the function, causing it to raise an exception.

For example, in the "`sa2/{sa2_code}/age`" function/API, if the user did not include a text in the body, the function will return an error code to the user alongside a meaningful error message of "no data found in headers" to users. So that the user knows there's something wrong with their request.

6.3 Missing Location Handling

One of the challenges encountered during the analysis was the absence of precise location information, such as longitude and latitude, in the Mastodon dataset. We employed a data simulation technique to address this limitation and enable spatial analysis. We randomly and uniformly assigned latitude and longitude coordinates to each social media post, ensuring an even distribution across the research region. This approach allowed us to simulate the geographic distribution of posts and facilitate the exploration of potential correlations between sentiment scores and demographic factors at the SA2 level with the concatenation of the other datasets. While this method may not reflect the exact locations of individual posts, it provides a reasonable approximation for our study, allowing us to investigate the spatial patterns of sentiment across different regions of Melbourne.

7. Methods and Data Analysis

7.1 Aim

For this project, we aim to investigate the impact of various weather conditions on the emotions of individuals across different age groups in Melbourne. To achieve this objective, we have selected three primary data sources: Mastodon for sentiment analysis of social data, the BoM for weather data, and SUDO for age demographics corresponding to the relevant regions.

7.2 Data Used

7.2.1 Weather Data from BoM

The Bureau of Meteorology (BoM) is a valuable source of comprehensive weather data, essential for understanding climate patterns and forecasting weather conditions. The data provided by BoM includes real-time observations and historical records spanning various meteorological parameters such as temperature, precipitation, wind speed, humidity, and atmospheric pressure across Australia. We specifically selected API data from the Melbourne Olympic Park station to ensure accurate regional matching.

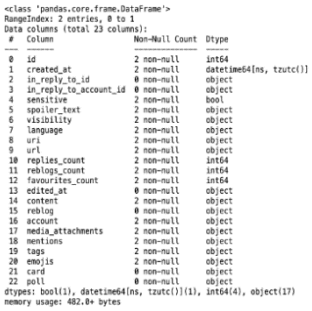
Figure 2 (Right) showcases the diverse range of weather data available, encompassing 35 features. To directly explore the effects of different weather conditions on people's moods, we have chosen to focus on the following features: 'local_date_time_full', 'apparent_t' (body feel temperature), 'air_temp' (air temperature), 'rain_trace' (presence of rain), 'wind_spd_kmh' (wind speed in kilometers per hour), 'rel_hum' (relative humidity), and 'vis_km' (visibility on the road). These selected features will enable us to gain insights into the relationship between specific weather parameters and the emotional states of individuals.

```
Data columns (total 35 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   sort_order              163 non-null    int64
 1   wmo                     163 non-null    int64
 2   name                    163 non-null    object
 3   history_product         163 non-null    object
 4   local_date_time         163 non-null    object
 5   local_date_time_full    163 non-null    object
 6   aifstime_utc            163 non-null    object
 7   lat                     163 non-null    float64
 8   lon                     163 non-null    float64
 9   apparent_t              163 non-null    float64
 10  cloud                   163 non-null    object
 11  cloud_base_m            0 non-null      object
 12  cloud_oktas             0 non-null      object
 13  cloud_type_id           0 non-null      object
 14  cloud_type              163 non-null    object
 15  delta_t                 163 non-null    float64
 16  gust_kmh                163 non-null    int64
 17  gust_kt                 163 non-null    int64
 18  air_temp                163 non-null    float64
 19  dewpt                   163 non-null    float64
 20  press                   163 non-null    float64
 21  press_qnh               163 non-null    float64
 22  press_msl               163 non-null    float64
 23  press_tend              163 non-null    object
 24  rain_trace              163 non-null    object
 25  rel_hum                 163 non-null    int64
 26  sea_state               163 non-null    object
 27  swell_dir_worded        163 non-null    object
 28  swell_height            0 non-null      object
 29  swell_period            0 non-null      object
 30  vis_km                  163 non-null    object
 31  weather                 163 non-null    object
 32  wind_dir                163 non-null    object
 33  wind_spd_kmh            163 non-null    int64
 34  wind_spd_kt             163 non-null    int64
dtypes: float64(9), int64(7), object(19)
```

Figure 2 Sample Weather Data from BoM

7.2.2 Mastodon

Mastodon, a decentralized social media platform, is an alternative to traditional centralized networks. The dataset utilized in this analysis was collected using the provided official API, enabling access to historical and real-time tweet data. The Mastodon dataset encompasses a wide range of aspects, including user experiences, comparative discussions with other platforms, and general opinions regarding its features. Specifying relevant keywords and time frames ensured that the collected data aligned with our research objectives.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    2 non-null      int64
 1   created_at            2 non-null      datetime64[ns, tzutc()]
 2   in_reply_to_id        0 non-null      object
 3   in_reply_to_account_id 0 non-null     object
 4   sensitive             2 non-null      bool
 5   spoiler_text          2 non-null      object
 6   visibility            2 non-null      object
 7   language              2 non-null      object
 8   uri                   2 non-null      object
 9   url                   2 non-null      object
 10  replies_count         2 non-null      int64
 11  reblogs_count         2 non-null      int64
 12  favourites_count      2 non-null      int64
 13  edited_at             0 non-null      object
 14  content               2 non-null      object
 15  reblog                0 non-null      object
 16  account               2 non-null      object
 17  media_attachments     2 non-null      object
 18  mentions              2 non-null      object
 19  tags                  2 non-null      object
 20  emojis                2 non-null      object
 21  card                  0 non-null      object
 22  poll                  0 non-null      object
dtypes: bool(1), datetime64[ns, tzutc()](1), int64(4), object(17)
memory usage: 482.0+ bytes
```

To gain a foundational understanding of the data, we examined a sample of posts within a 10-second interval. Figure 3 (Right) reveals that two posts were updated within this brief timespan. Further analysis of the data structure indicates the presence of 23 features. Our study will focus on the key features: 'created_at,' 'lat,' 'lon,' and 'content.' These features provide us with the necessary information regarding the content users post and their corresponding location data.

Figure 3 Sample Mastodon Data

7.2.3 SUDO

A vast amount of data is provided on SUDO. As we are interested in revealing some relationship between median age and sentiment scores, the median age of Victoria regions based on SA2 level division is being utilized in our study. This official demographic data enables us to explore the potential correlations between age distribution and the emotional states of individuals across different regions of Melbourne, as expressed through their sentiment scores on social media platforms.

7.3 Data Handling

7.3.1 Weather

The BoM data is recorded at 30-minute intervals and does not adhere to our expected time format for later analysis. We have developed a custom function to standardize the time format to ensure accurate matching between the BoM weather data and the Mastodon data. The function applies the following logic: if the minute value is less than 15, it rounds down to the nearest hour; if the minute value falls between 15 and 44, it

rounds to the half-hour mark; and if the minute value is 45 or greater, it rounds up to the next hour. This standardization process enables seamless integration and synchronization of the BoM weather data with the Mastodon data contents.

7.3.2 Mastodon

Following the initial exploration and understanding of the data, several preprocessing steps are necessary to facilitate further analysis. The preprocessing phase involves standardizing the data and leveraging Natural Language Processing (NLP) techniques to assign sentiment scores to each post from the Mastodon dataset.

Although the data sources are relatively clean, the Mastodon content requires additional cleaning to extract pure text. We observed the presence of extraneous information irrelevant to our analysis. For example, the content includes HTML tags such as <p> at the beginning and URLs of images within the posts. Additionally, the JSON data retrieved through the API contains line breaks, which can interfere with the accuracy of NLP text analysis. We implemented data cleaning techniques to address these issues, remove unnecessary elements, and standardize the text format.

Once the contents have been cleaned and preprocessed, we pass the text through the TextBlob library to obtain sentiment scores. The sentiment scores generated by TextBlob range from -1 to 1, with higher scores indicating more positive sentiment, while lower scores suggest more negative sentiment. These sentiment scores provide valuable insights into the emotional tone expressed in each tweet and are saved for later analysis in sentiments.

7.4 Experiment Construction

7.4.1 Hypothesis

The hypothesis testing process was conducted as follows:

H0: There is no significant correlation between each weather factor and sentiment. The correlation coefficient (ρ) equals zero ($\rho = 0$).

HA: There is a significant correlation between each weather factor and sentiment. The correlation coefficient (ρ) is not equal to zero ($\rho \neq 0$).

7.4.2 Data to be Analysed

1000 posts from May 17th to 18th from the Mastodon dataset.

7.4.3 Evaluation

The Pearson correlation test would be conducted for each weather factor to determine the correlation coefficient and the p-value.
The Pearson correlation coefficient indicates the strength and direction of the linear relationship, while the p-value indicates the probability of observing the data, assuming the null hypothesis is true. A significance level was then determined, and the p-values were compared to this significance level. The null hypothesis is rejected if the p-value is less than or equal to the significance level, indicating a significant correlation between the weather factor and sentiment. If the p-value is more substantial than the significance level, the null hypothesis is not rejected, indicating no significant correlation.

7.5 Results and Discussion

The analysis explores the potential influence of weather factors on sentiment and some other findings by correlating user location with the region's median age and examining the relationship between median age and sentiment scores.

7.5.1 Preliminary Judgment

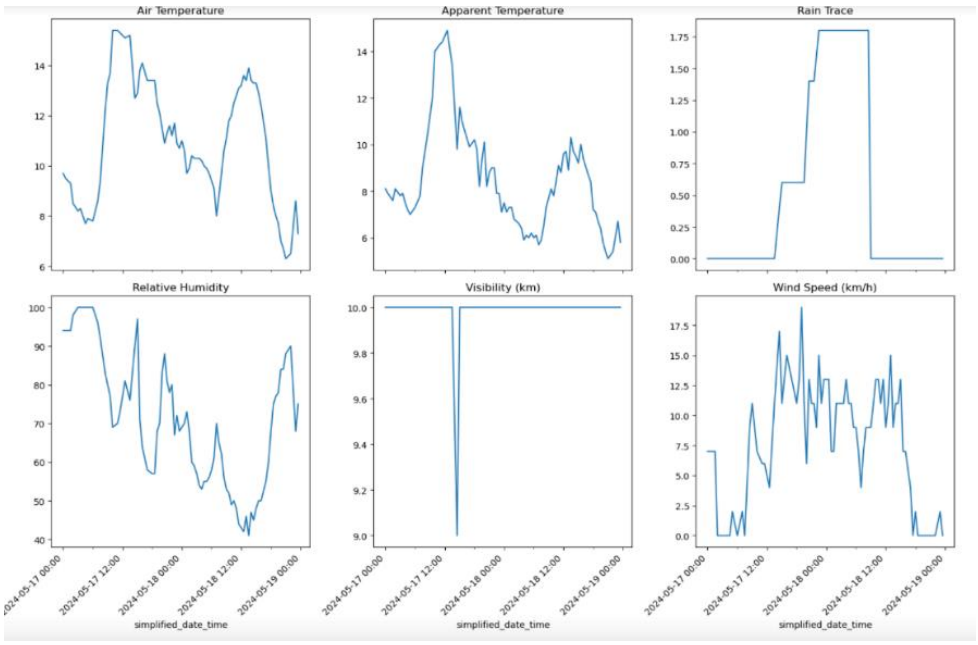


Figure 4 Weather Factors Change through Time

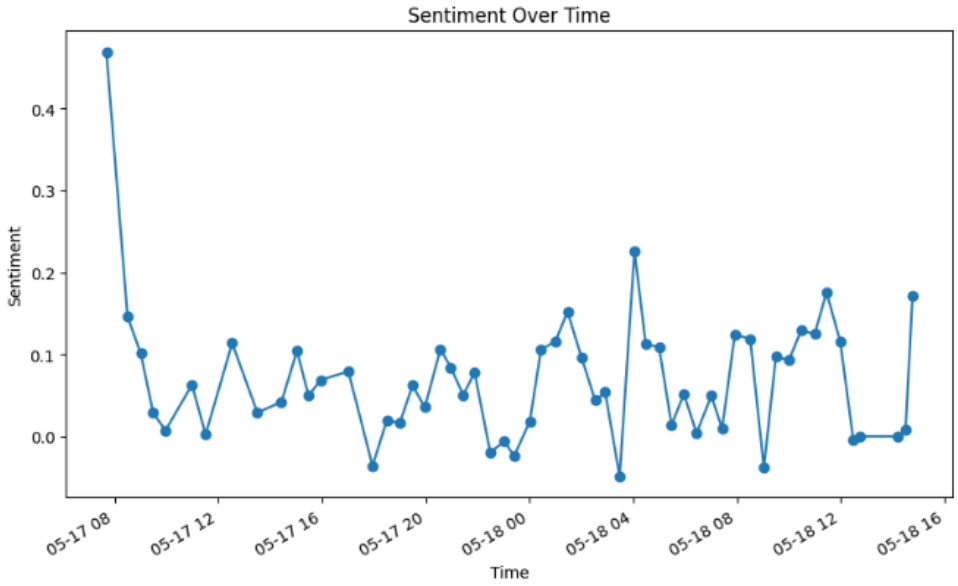


Figure 5 Sentiment Changes through Time

We conducted descriptive statistics to make a preliminary judgment on the value of our study. The data are preprocessed, as discussed in the *Data Handling* section of the report.

Analyzing the charts of the six weather factors (air temperature, apparent temperature, rain trace, relative humidity, wind speed, visibility) and the sentiment changes in Melbourne, we observed similar cyclical patterns in both sets of charts. This suggests a potential relationship between sentiment and weather factors. This preliminary observation indicates that these weather factors may influence sentiment.

7.5.3 Influence of Weather Factors on Sentiment

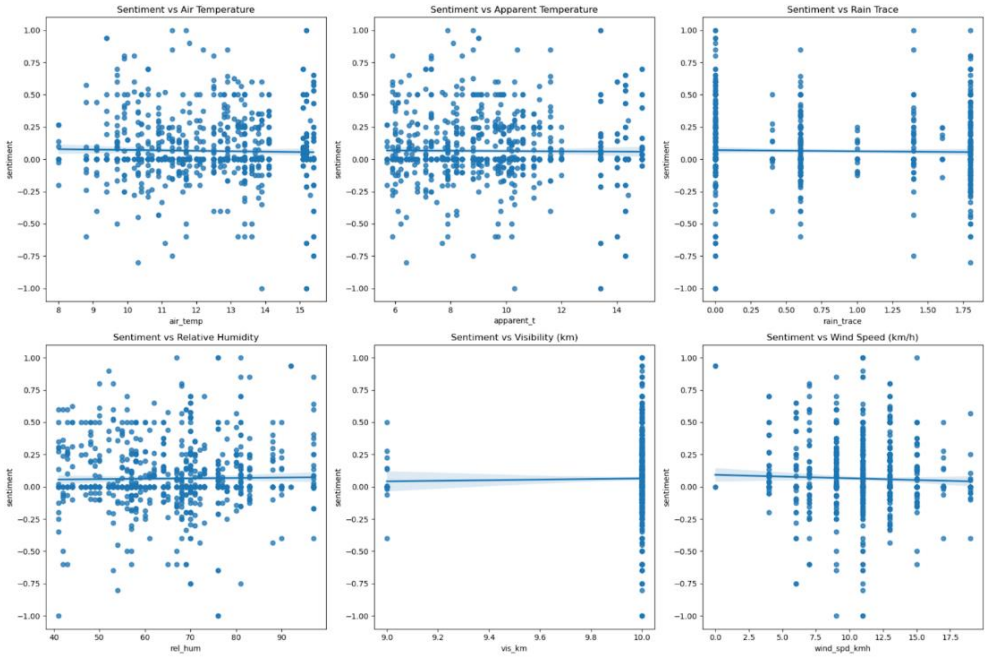


Figure 6 Scatter Plots on Weather Factors and Sentiment Score

Scatter plots were created to further explore the relationship between sentiment and weather factors (Figure 6). These plots revealed a wide dispersion of sentiment scores across different values of air temperature, apparent temperature, rain trace, relative humidity, wind speed, and visibility, with no strong visible trends or significant correlations. This suggests that while there may be no strong linear relationships between sentiment and any single weather factor, more complex interactions could be at play.

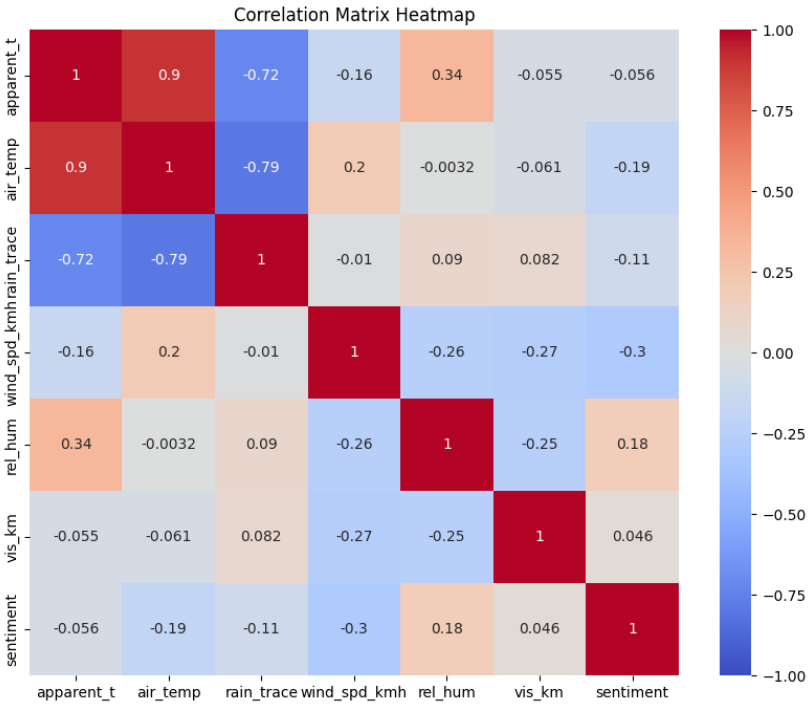


Figure 7 Correlation Heat Map of Weather Factors and Sentiment Score

Weather Factors	***Pearson Corr***	***p-value***
Apparent Temperature	-0.056	0.684
Air Temperature	-0.187	0.169
Rain Trace	-0.107	0.436
Wind Speed	-0.302	0.024
Relative Humidity	0.181	0.182
Visibility	0.046	0.734

Table 2 Pearson correlation coefficients and p-values of Weather Factors (rounded)

A correlation matrix heatmap (Figure 7) and Pearson correlation coefficients with p-values were calculated to quantify the correlations (Table 2) between sentiment and six weather factors. Using a significance level of 0.05, wind speed showed a slight negative correlation with sentiment (correlation: -0.302, p-value: 0.024), while other weather factors exhibited weaker correlations. However, considering the study's exploratory nature and the dynamic interactions between weather conditions and sentiment, a higher significance level of 0.2 was also employed. With this new significance level, air temperature showed a potentially significant negative correlation with sentiment (correlation: -0.187, p-value: 0.169), and relative humidity demonstrated a potentially substantial positive correlation with sentiment (correlation: 0.181, p-value: 0.182). These findings suggest that wind speed, air temperature, and relative humidity influence sentiment more than other factors.

7.5.3 SA2 Level Area-Based Visualization

We can also investigate the relationship between the number of posts per region and the median age. By analyzing this relationship, we can derive exciting conclusions.

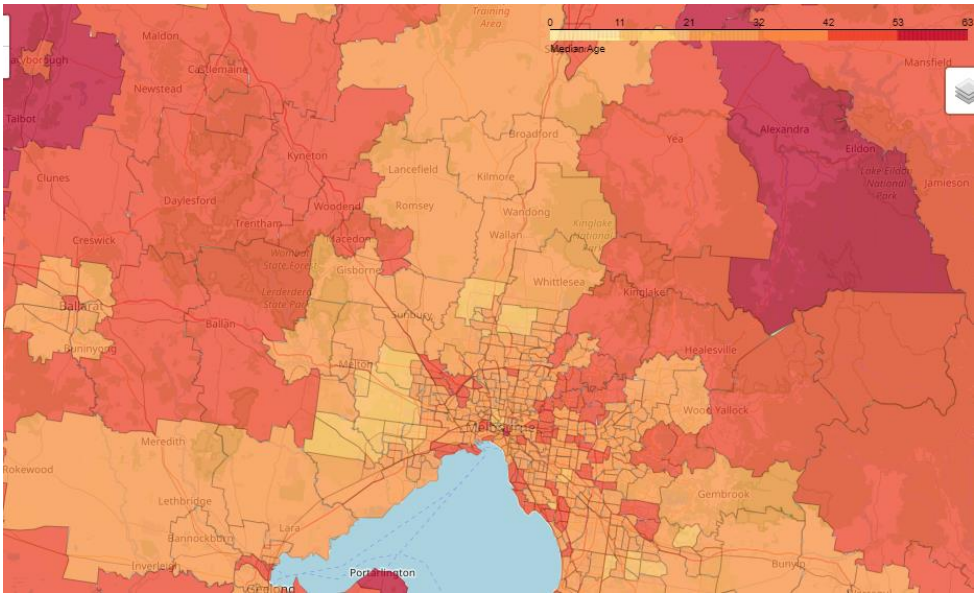


Figure 8 Median Age Distribution for each region of Melbourne

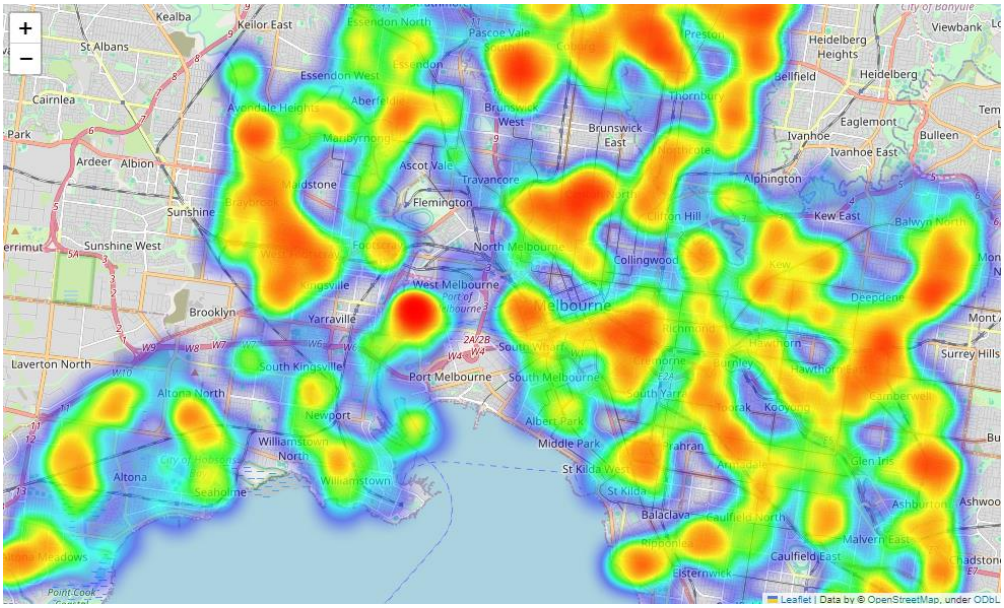


Figure 9 Heat Map on Distribution of the number of social media posts for each Melbourne

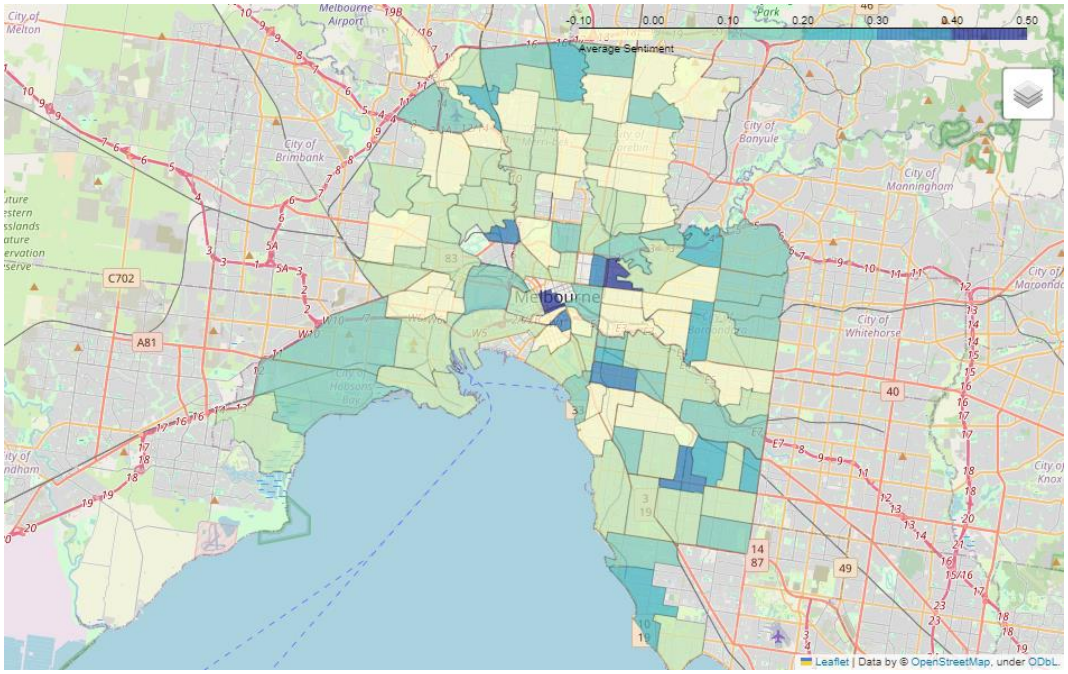


Figure 10 Distribution of Mean Sentiment Scores for each Region of Melbourne

The median age distribution map (Figure 8) indicated that regions closer to the centre of Melbourne tended to have a lower median age. Outer areas exhibited a higher median age, showing a distinct central-peripheral trend.

The distribution of social media posts per region (Figure 9) was also analyzed with the median age. Central areas and certain suburban hubs showed higher activity levels, while peripheral regions exhibited lower social media activity. This inverse relationship between median age and the number of social media posts suggests that areas with younger populations tend to have higher social media activity. In comparison, regions with older populations show lower activity.

To account for potential regional differences in emotional impact, the average sentiment map for each region of Melbourne was examined (Figure 10). The map revealed significant differences in average sentiment scores between regions, with central areas generally displaying higher scores and peripheral regions showing lower scores. This suggests that demographic factors such as age might influence regional sentiment in a similar pattern to the central-peripheral trend.

7.5.4 Median Age and Sentiment Revisit

The bar chart in Figure 11 (Bottom) illustrates the distribution of posts across different median age groups. Most posts originated from regions with a median age between 30 and 50, with the highest concentration in the 35 to 40 age range. Regions with median ages below 30 or above 50 contributed significantly fewer posts.

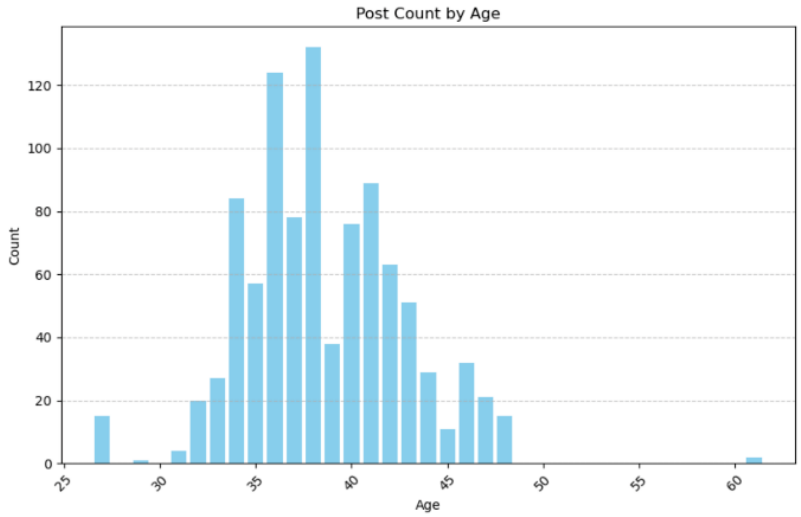


Figure 11 Post Counts Classified by Median Age

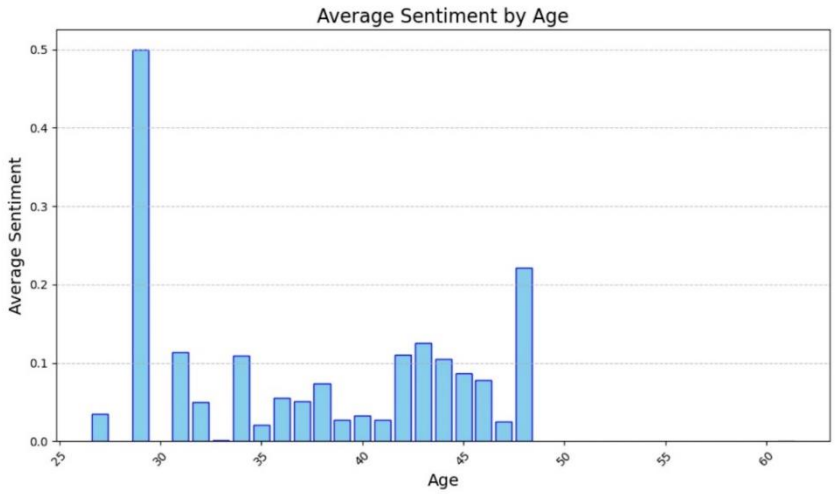


Figure 12 Average Sentiments Classified by Age

Figure 12 (Above) depicts the relationship between the median age of each region and its average sentiment score. The highest average sentiment, approximately 0.5, was observed in areas with a median age of around 30. The average sentiment fluctuated for other median age groups but generally remained lower, mostly between 0 and 0.2.

These two diagrams align with the findings we have established in the previous section.

8. Conclusion and Limitations

This study has explored the relationship between weather factors, demographic characteristics, and sentiment in Melbourne using data from the Bureau of Meteorology (BoM), Mastodon social media platform, and the Statistical Urban Development Organization (SUDO). The analysis revealed potential correlations between wind speed, air temperature, relative humidity, and sentiment scores. Furthermore, the study highlighted the importance of considering regional differences and demographic factors, such as age, when examining sentiment patterns.

However, it is essential to acknowledge the limitations of this study:

Data limitations:

1. Insufficient data size: The analysis was based on a limited dataset, which may not fully represent the entire population of Melbourne. A more extensive and more comprehensive dataset would enhance the reliability and generalizability of the findings.
2. Lack of comprehensive data coverage: The study relied on data from specific sources (BoM, Mastodon, and SUDO), which may not capture the full range of weather conditions, social media activity, and demographic information in Melbourne. Incorporating data from additional sources could provide a more comprehensive understanding of the relationships between weather, demographics, and sentiment.
3. The accuracy of sentiment scores and the complexity of emotions: Sentiment analysis using tools like TextBlob may not always accurately capture the nuances and context of human emotions expressed in social media posts. Emotions are complex and multifaceted, and the sentiment scores assigned to posts may not fully reflect the actual emotional state of the individuals.
4. Individual preferences: The study did not account for personal preferences regarding weather conditions. People may have varying opinions and emotional responses to the same weather conditions, which could influence the observed relationships between weather and sentiment.

Software limitations:

1. As discussed in section 6.1, `geopandas` are unavailable in the Fission environment. Hence, some of the joint computations using `geopandas` are done locally. This problem could be solved in the future by building a docker image containing all relevant dependencies that allow the library to be installed. By implementing this step, our system can remove all local computations, making the whole system more scalable on the cloud.
2. Regarding APIs, some APIs could be more flexible. For example, we can build a single API endpoint for bulk and single queries. Furthermore, query APIs like `concat`

can be extended to support more functionalities, allowing users to select which attribute of the dataset they want.

3. More extensive functions, namely the two harvesters, should be more efficient. These two functions run sequentially for now, which could introduce performance bottlenecks in some cases as they need to query multiple other Fission functions. In the future, this issue could be solved by implementing asynchronous executions in which both harvesters can send requests to other functions simultaneously, potentially shortening the execution time by reducing request sending and receiving overheads, making the function adhere to serverless principles.

Despite these limitations, this study provides valuable insights into the correlations between weather, demographics, and sentiment in Melbourne. The findings can serve as a foundation for future research, which could address the limitations by incorporating larger and more diverse datasets, employing advanced sentiment analysis techniques, and considering individual preferences and contextual factors.

In conclusion, this study has shed light on the potential relationships between weather factors, demographic characteristics, and sentiment in Melbourne. While the limitations should be considered when interpreting the results, the findings emphasize the importance of examining the complex interactions between environmental, social, and emotional factors in urban settings. Further research in this area could contribute to a better understanding of how weather and demographics influence individuals' well-being and emotional states in cities like Melbourne.

9. Demonstration Video

YouTube: https://youtu.be/KhQH8dYDXcw

10. Source Code

GitHub: **https://github.com/chris0311/COMP90024-Project2**
Collaboration requests have already been sent to rsinnott@unimelb.edu.au, alwynpan, and lmorandini. Should you have any issues accessing the repository, please contact junyliang@student.unimelb.edu.au