COMP90024 Cluster and Cloud Computing

Gitlab:https://gitlab.unimelb.edu.au/ygao3631/comp_team_43

Demonstration video linkage:https://youtu.be/jfKYixlxkM4

Topic: Australia Election

Group number: 43

Account: 7583

Pages: 24

Name	Student ID
Linyao ZHOU	1619649
YIHAO SANG	1562582
Xiwen CHEN	1542252
Yuan GAO	1602894
Yao ZHAO	1695969

1.Project Overview and Background

1.1 Project Background

The Australian federal election is a highly relevant and feasible topic for analysis due to its national significance and strong public interest. It generates a large volume of data from social media platforms, news outlets, and official sources, providing a solid foundation for data-driven research. Additionally, the topic allows for multi-dimensional analysis, including sentiment analysis, topic modeling, and public opinion tracking on key political issues and party performance.

1.2 Project Overview

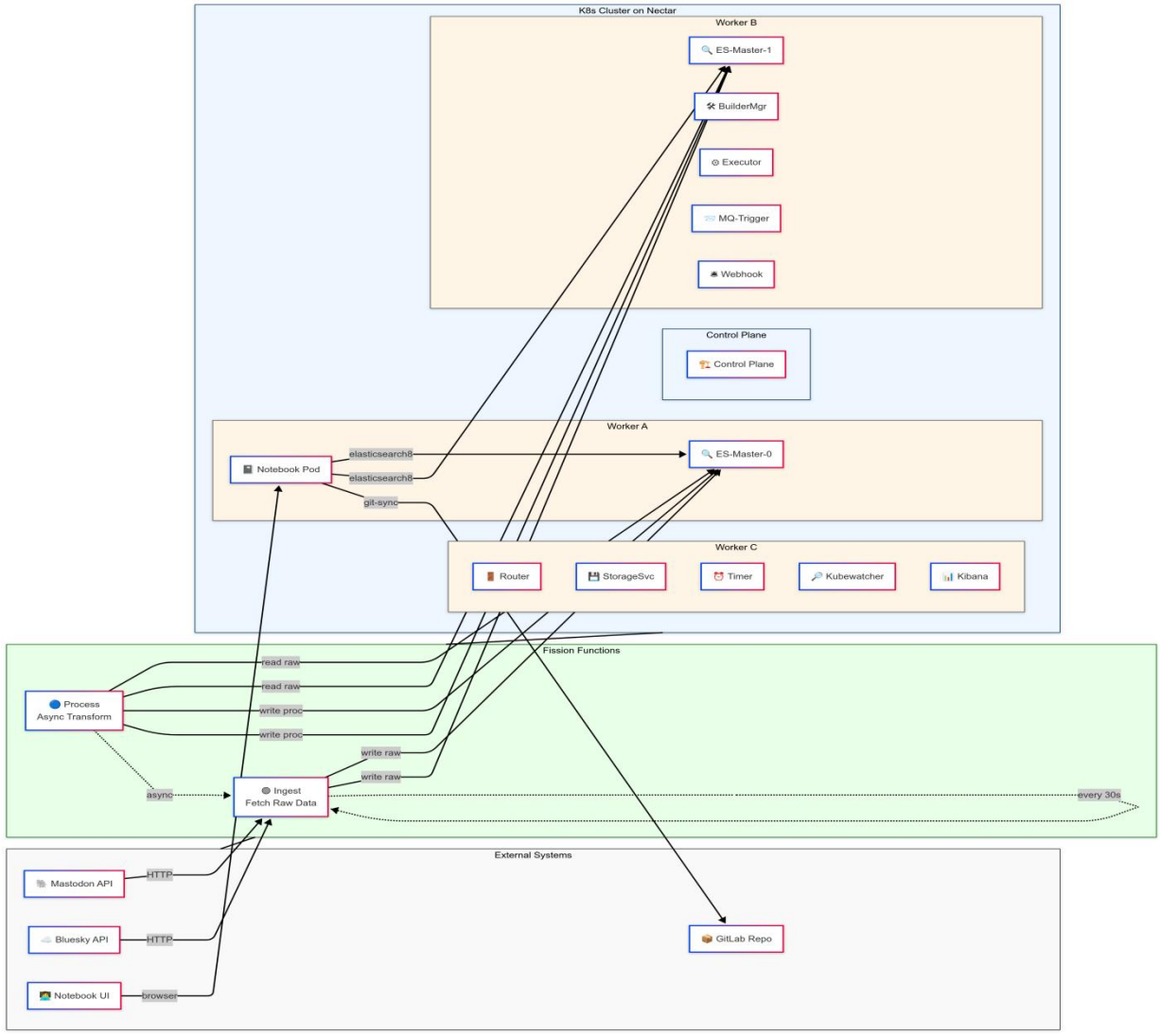
This project builds a well-defined pipeline for harvesting, processing, and analyzing Mastodon and Bluesky posts related to the Australian election. We employ both historical and incremental ingestion modes via the Mastodon API (targeting the high-volume mastodon.au instance) and BlueSky API to ensure comprehensive coverage Historical Ingestion, which Starting from five months ago, we page backward through the #ausvotes timeline (and a tailored set of election-related keywords) to retrieve all matching posts and incremental Updates which is a serverless function, deployed on Fission, triggered on a regular schedule (e.g. every hour) to pull only those posts newer than the latest indexed timestamp, ensuring the index remains up-to-date without reprocessing older data.

Each post undergoes a consistent standard ETL step and followed by sentiment analysis to assign a compound score and discrete label (positive/neutral/negative). This process is also scheduled in regular tasks to update the data in the final index, which will be used for analysis. We also extract or geocode user-declared locations, and classify each timestamp into one of four time-of-day buckets. Processed documents are first stored in an Elasticsearch “raw” index with rich mappings , then selectively reindexed into a lean “analysis” index housing only the fields needed for downstream visualization and querying. All components run as Fission functions—packaged independently, bound to HTTP routes and timetrgger, and driven by cron-style triggers—enabling autonomous, fault-tolerant operation and real-time progress logging without dedicated servers. Finally, all the data will be visualized by Jupyter Notebook.

2 System Architecture

2.1 Overview

The whole system is based on Nectar service and Kubernetes(k8s) cluster. The entire project consists of the Elasticsearch database part of the backend, the Faas deployment part of the fission function and the presentation part of the frontend. The architecture diagram is shown below as picture 2-1：



Picture 2-1 Architecture diagram

2.2 System Architecture

The back-end database is mainly based on crawled sample data to build the index of Elasticsearch and build the index of the secondary processed data. The back-end fission data processing consists of two phases, the first part of the original data crawling, and the basic dirty data processing, write the original data into the Elasticsearch database. The first part is to capture the raw data and basic dirty data processing, write the raw data into the table of Elasticsearch database, and set scheduled task to update the latest data into the same table regularly; the second part is the data refinement part, which is the secondary processing of the captured raw data according to the business requirements, including some data format conversion, granularity

adjustment, data aggregation, and special data processing, so as to generate the final data used for the external It is also necessary to set up a timed task to update the latest data captured in the raw data table. Through this asynchronous data processing, the entire data processing architecture can be made clearer; the two parts of the decoupling, so that the entire data processing process is more convenient for maintenance and expansion, such as adding new data sources or changing the original business processing logic and structure. The last part of the back-end is the deployment of notebooks and persistence strategy, through the cluster deployment of lightweight notebooks set git-sync mechanism, regularly pulling the latest code from the GitLab repository for deployment. Convenient code iteration management and real-time update deployment. Front-end data visualization part of the notebook to increase the ipynb file, through the elasticsearch8 package to interact with ES, pull data for front-end rendering and display.

2.3 Scalability

For the original data pull and secondary extraction of asynchronous mechanism makes the whole system easy to expand, if you add more data sources, only need to manually change the API to increase the fission function to write the new data original data into the raw data table, if you need new data aggregation or granularity, only need to change the second part of the data secondary processing of the fission function, do not need to carry out the overall changes. At the same time, if a single data source data surge, you can also set the Maxscale parameter to make the pod automatically expanded

3 Cluster and Backend Development Process Analysis

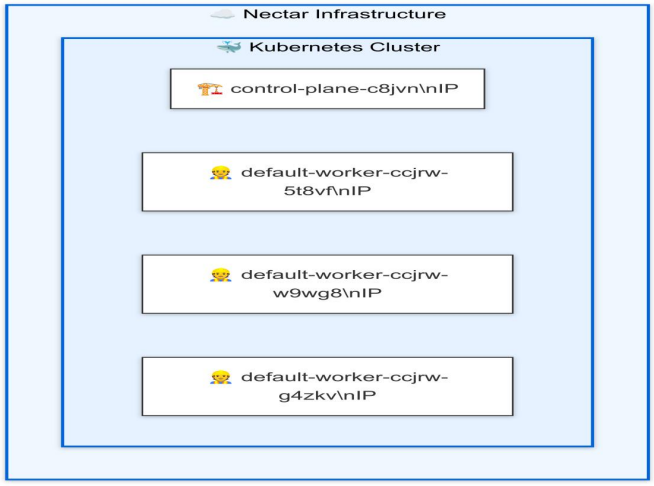
3.1 Nectar Platform

Nectar provides multi-core CPUs large-memory instances—ideal for large-scale API data ingestion (Mastodon/Bluesky) and Elasticsearch indexing. It also provides k8s scalability, users can Instantly scale control and worker nodes up or down to match workload demands. During the development, we try both dashboard and CLI ways to develop the system. These two ways have their advantages and disadvantages, the visual dashboard can help newcomers quickly get started, and understand the cloud ecosystem to facilitate a quick look at the logs, for a single view or a small number of modifications very quickly, but in reflecting the specific operation of the cluster, there may be unstable, and in the actual development of the difficult to do the scripting, pipeline integration. Through the CLI approach, although it is difficult to get started, but the batch operation can be automated through the script, the execution speed is fast, and the execution of the command is more stable and controllable.

3.2 Kubernetes(k8s)

Kubernetes is an open source container orchestration platform used to automate the deployment, scaling and management of containerized applications. By managing the full lifecycle of containers, it enables elasticity, high availability, and automated operation and maintenance of cloud-native applications. Through k8s we can very simply manage containers, he maintains a certain degree of transparency for developers, containerized deployment can be achieved simply and quickly deployed and ported, and most importantly, environmental isolation can be achieved,

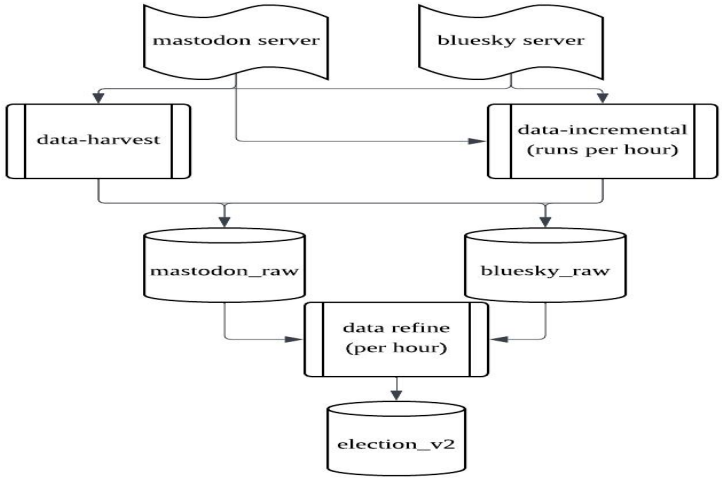
the environment inside the container and the external environment will not affect each other, and if there is an error in the pod, it is also probable that the stability of the external environment will not be affected. In addition, the running environment inside the container is also lightweight, do not need to start a whole operating system. Cluster architecture diagram is shown below as picture 3-1:



Picture 3-1 k8s

3.3 Elasticsearch

Data stream in the Elasticsearch is shown below as picture 3-2:



Picture 3-2 ES data Stream

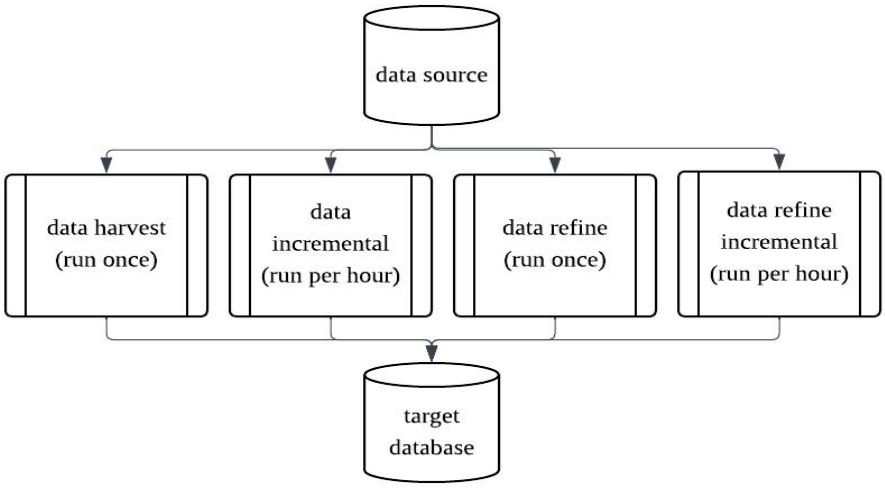
Elasticsearch is a distributed search and analysis engine based on Apache Lucene. It will be documents (JSON) organized into the 'index' (Index), to support near real-time full-text search, structured queries and complex aggregation analysis. This project uses Elasticsearch as the database for the following reasons: first of all, the API pulls most of the data is textual data , the

use of ES can quickly query and aggregate data , and secondly, the ES database and k8s can achieve a good coupling , easy to integrate the project , and through the deployment of k8s ES can be achieved through multi-copy to ensure that the availability and stability of the entire database . . From the point of view of ES itself , it is suitable for more reading and less writing , in the current business environment , a large number of raw historical data is read into the database , do not need to make frequent modifications , at the same time , the combination of ES and fission functions to facilitate the aggregation of data in various dimensions and secondary processing , such as from the massive social media data (such as Mastodon, Reddit.), BlueSky) to quickly filter out content related to specific topics or events. Finally, Elasticsearch can support search within seconds of data being written, which, together with the data stream processing implemented by Fission, enables near real-time analysis. This is especially important for this project's hot event analysis, user sentiment tracking, dynamic impact assessment and other scenarios, and ES database support for heterogeneous data sources is also one of the key reasons for this project to use ES as a database technology, so that it is easy for the development team to deal with different data sources in one system.

It should be noted that the index structure of ES needs to be established in advance after careful discussion based on the sample data obtained, and if it is not perfect, it may affect the later data processing and analysis, which is the shortcoming of ES that we encountered during the development process.

3.4 Fission

Fission functions mechaism are shown below as picture 3-3:



Picture 3-3 ES fission mechanism

Fission is an open source serverless function framework based on Kubernetes, which allows developers to focus only on the code itself, without having to manage the underlying server or container deployment process. In this project, Fission is mainly used for event-driven streaming data processing and automated task triggering. In this development project, the deployment of functions is not particularly large, so the deployment mode of the function selected newdeploy,

each function has an independent Pod and Deployment, completely isolated from each other, to avoid resource hijacking, dependency conflicts or security risks, through this way, each function is a standard Kubernetes Deployment and Service, which makes it easy to monitor the running status in real time. Using the fission stateless function, when pulling resources and secondary processing data, you don't need to care about the last data processing situation, you only need to follow the data processing and harvest logic to carry out the current data operation, greatly simplifying the maintenance difficulty and processing flow to achieve the data flow processing. fission scheduled timetrigger can be used to ensure real-time updating of the data through the Simply configure the timetrigger to ensure that the data is updated in real time, to ensure that the entire project database and data availability.

3.5 Updateable database based on incremental crawling

Back-end development overall based on fission function and timetrigger technology, for different information sources (mastodon, bluesky) set different functions to crawl separately, and set the function for incremental crawling content for database writing, to achieve the effect of real-time crawling and real-time updating of the database

3.5.1 Mastodon

1. raw data incremental crawling approach

In the specific data pulling process, the code used is mastodon-scheduled-update.py and mastodon-scheduled-update-v2.py, which update the mastodon_election_raw and election_v2 tables, respectively, which are These two tables store the raw data pulled, and the pulled data is used as a data warehouse backup.Then specific data pulling process, the logic of the update is: first need to initialise the incremental anchor, connect to the es database by obtaining the es current index of the latest document id's as an anchor (in descending order), then request the API to obtain the newest posting id's by mastodon's snowflake ID order to obtain the data, by page size to obtain the paging size.

Then request the API to get new in the post ID of all the latest posts by mastodon snowflake ID order to get the data by page size to get the paging size, after a simple data cleaning and sentiment analysis enhancement (see the data analysis section), set up bulk bulk write ES, in the write log error but not interrupt the write process, to ensure that the single Stability of single write. Finally, update the current since_id to prepare for the next update. The following is a pseudo code snippet 3-1：

fetch_and_clean_hashtag_posts()
statuses = masto.timeline_hashtag (HASHTAG, limit = PAGE_SIZE, since_id = since_id) def clean_content(html): return re.sub(r'<[^>]+>',

Table 3-1 fetch_and_clean_hashtag_posts()

2. raw data refinement incremental

In the further data processing process, the secondary processing of data also makes use of the incremental anchor mechanism, similar to the one mentioned above, analysing the latest time of

the index as an anchor, processing the full amount of data during the first run, and subsequently updating based on the last processing, and applying helpers.scan scrolling query to avoid deep paging problems during ES querying, which is suitable for a large amount of data streaming processing，the following is a pseudo code snippet 3-2：

query_es_docs_since_latest()
resp = es.search(index=ANALYSIS_INDEX, size=1, sort=[{"created_at": "desc"}]) since_ts = <abstract the latest timestamp> query = {"range": {"created_at": {"gt": since_ts}}} if since_ts else None for hit in helpers.scan(es, query=query, index=RAW_INDEX):

Table 3-2 query_es_docs_since_latest()

The data processing contains data enhancement part, in the original data, there are individual data geographic data can not be obtained or defective, in this processing, in order to display the final effect, do the reasonable data enhancement: if the current data missing geographic data, according to the population density information of Australia, each region will be weighted, to generate a random geographic location, better fitting the real data.

3.5.2 BlueSky

For bluesky's data crawling, two crawling methods are used in parallel to synchronize crawling, and the front-end group gives the data mining scope and crawling direction, the specific code is zy-bluesky-v2.py (hereinafter referred to as "v2") and bluesky-zy.py (hereinafter referred to as "v1"). (hereafter "v1") can be found in the gitlab repository.

1. v1's incremental crawling approach

Use the cursor mechanism to record the cursor returned by the API at the end of the last crawl, and then continue to request from that cursor to get "new" data each time. Implementation details includes State storage: Using ES documents to save a STATE_DOC_ID, which has the cursor, but also records the batch number and the total number of processes ; Capture process: Each time one or more "batch" (batch), each batch of cursor pagination pull, until the full BATCH_SIZE data or no new cursor. The new batch will use the last saved cursor as a parameter, the API will return the new cursor, the process is progressive. After each batch, save the new cursor and continue from there. The following is a pseudo code snippet 3-3：

enhance_missing_geodata()
loc = src.get("location") or "" if loc not in CITY_BBOX: loc = weighted_choice(POP_DENSITY) geo = sample_geo(loc)

Table 3-3 enhance_missing_geodata()

Pros: This can continue to "continue to capture the last place" capture, suitable for the continuous addition of data and API support cursor paging scenarios and can control the number of batches, so it's easy to execute tasks in batches ; Limitations: If the data is updated (e.g. old posts are

edited or deleted), the cursor may fail or miss part of the changes and this depends on BlueSky API for cursor semantics and stability.

2. v2 's incremental crawling approach

The core idea is using the earliest timestamp threshold (min_ts) as the "history page" method, each time only grab the earliest time earlier than the last time, gradually back to the history, to ensure that there is no overlap. Implementation details includes: State storage:Use ES document to save a `STATE_DOC_ID`, only record `min_ts` (the earliest timestamp of the last capture). Crawl flow: On each run, read the min_ts of the last time, and only process posts older than min_ts (ts_iso < last_min_ts) in this round. The crawl is updated in real time with the earliest timestamp encountered this time (min_ts_this_run). The new earliest timestamp is written back to ES after this round of crawling for next time.

Pros: Stable "back to history", suitable for replenishing historical data, no missing data due to cursor loss. Data on the timeline will not overlap, so it is naturally weight-proof. Limitations: Only suitable for history replenishment or scenarios where data is no longer updated. Can not automatically get "new posts", if you need to catch new data, you need to redesign the logic.

3. Code snippet example

Save the cursor state of v2, the timestamp state of v1 is saved and Incremental criteria for v1. The following is a pseudo code snippet 3-4：

stateful_timestamp_processor()

`def load_state():`
`# return cursor etc.`
`def save_state(state):`
`  # save cursor`
`def load_state(): # return min_ts`
`def save_state(min_ts): # return min_ts`
`if ts_iso >= last_min_ts:`
`  continue`

Table 3-4 stateful_timestamp_processor()

4. Combining Processes

v2 is more suitable for real-time or continuous capture of new data, use cursor to advance to avoid duplication. v1 is more suitable for historical data archiving. v1 is more suitable for historical data archiving, using the timestamp threshold in reverse order to the historical recursion, not missing and not heavy. In the final analysis process, the two are combined with each other, v2 crawls new data, incremental crawling of new data combined with fission timer for uninterrupted writing to the election database, and v1 version pauses after a certain amount of time crawling and writing historical data to the database. To achieve the complete mining of bluesky data.

3.5.3 Build New index for frontend

After the Bluesky and Mastodon databases are prepared, create a Fission timetrigger that depends on Python's elasticsearch library. Set a certain time interval to write the contents of the two databases into a new index in Elasticsearch.4 Data collecting

The chapter describes combining several different social media data to analyze changes in public sentiment before and after the Australian election. Data sources include Mastodon, Bluesky, and Reddit, and through data mining and data analysis, we can show structured fields such as geographic location, posting time, and sentiment tendency based on public sentiment, and provide basic data support for subsequent analysis.

3.6 Problems and Solutions

3.6.1 Fission Package Building

When creating a fission package, using the requirements.txt dependency file plus the buildcmd command to deploy and upload the dependencies when installing sometimes results in an error report, and the package building process is not very stable. Therefore, in this project, we use pip -r requirements.txt -t <deps dir> to download the corresponding version of the dependency in advance and upload the dependency synchronously when we build the python package of fission on the basis of local stable testing, which can ensure that the function of the fission package build is 100% successful, you can manually check if all the dependencies are in the <deps dir> folder, which reduces the instability of packaging during development. This reduces the instability of the package during development. It also ensures that the package will have a very high probability of success in the online environment after the local test passes.

3.6.2 Notebook deployment and Data Persistence

For the final deployment of Jupyter notebook, we tried to use jupyterhub at the beginning, because the complete hub is more comprehensive for the integration of functionality, but later in the specific startup deployment process, Jupyter's initialisation process of the package dependency installation is always reported as an error, and can not be initialised properly Jupyter pods, so we used the official basic Jupyter notebook image, and use non-persistent PVC and git-sync to implement the data and code persistence mechanism, this deployment is very lightweight and fast, using kubectl apply yaml file to manage the pod's runtime environment and startup order. Check the dependency and git project update status before starting the pod, then start the notebook pod, and update the git repository every 30s to keep the deployment up-to-date. You can create temporary files in the work directory for development during specific development to ensure the consistency of the development environment and the deployment environment. At the same time, helm's git-sync container accesses the gitlab repository through a read-only token, ensuring that members of the same environment development and testing will not affect the final deployment directory, ensuring the stability of development and deployment. Finally, deploying notebooks through yaml files ensures that pods can be automatically recovered and restarted when something goes wrong, and the initialisation policy set also ensures that after pods are restarted, they can still successfully obtain complete dependencies and deployments, so there is a good fault-tolerance mechanism.

3.6.3 Elasticsearch Connection

The default Python version provided by the cloud computing system is 3.9. When installing the elasticsearch dependency package in the Python 3.9 environment, it will, by default, install version 7 of elasticsearch. Some parameters differ from those in Elasticsearch 8 used in the cluster, so the es object in the Python script needs to be adjusted as table 3-5

ES connection config
`es = Elasticsearch([ES_HOST],http_auth=(ES_USER, ES_PASS),use_ssl=True,verify_certs=False,ssl_show_warn=False,)`

Table3-5 ES connection config

3.6.4 Handler issue in Python code

Fission requires an explicitly exposed function named handler (or a specified name) in the code. If the function file name or function name is misspelled, or if the handler is not exported, Fission will not be able to locate the entry point. In some cases, errors may occur when invoking the function, or requests may not be successfully responded to. This is because Fission requires the handler function's signature to conform to the framework's requirements. In Python, it is required to accept context and request parameters. When writing code, it is best to manually set these parameters to empty values as much as possible, which can help avoid such issues.

4 Data collecting

The chapter describes combining several different social media data to analyze changes in public sentiment before and after the Australian election. Data sources include Mastodon and Bluesky, and through data mining and data analysis, we can show structured fields such as geographic location, posting time, and sentiment tendency based on public sentiment, and provide basic data support for subsequent analysis.

4.1 Mastodon Data collecting

4.1.1 Mastodon Data Mining

1. API Access and Authentication

The data of this project mainly comes from Mastodon, which is a decentralized social network platform, so we obtain dynamic data through its public API access. Data mining needs to complete authentication and authorization first, and Mastodon.py library is used for connection, and the authentication method is access token.

2. Data fetching strategy

The crawling strategy includes keyword retrieval, data field screening and time span selection to ensure the integrity of multi-dimensional information extraction.

For keyword retrieval, this project conducts research and observation on Mastodon, and then combines the information retrieved on the Internet to obtain the most commonly used keywords and tags for each user when publishing posts on the topic of Australian election. In addition to the most popular keywords such as 'Australia Election', 'AusPol', '#ausvotes2025', in order to get

more information, we also add keywords such as candidates, political parties, Examples include 'Albanese', 'Dutton','Labor', and 'Liberal 'to ensure the integrity of the information obtained, making the data more realistic and effective. The following is a pseudo code snippet 4-1：

```
ELECTION_KEYWORDS
KEYWORDS = [ 'Australia Election', 'AusPol', 'AUSElection',
'ausvotes2025', '#ausvotes2025', 'auspol2025',   '#auspol2025',
'Albanese', 'Dutton', 'Bandt','Labor', 'Liberal', 'Greens']
```

Table4-1 ELECTION_KEYWORDS

After setting the keywords, this project first pulled the data fields owned by a post as much as possible, observed the correlation and connection between the data fields, and prepared for the follow-up project analysis. The data fields are explained as follows:

Parameter Name	Data Type	Description	Data Source & Acquisition Method	Processing Notes & Remarks
id	string	Unique post identifier	st['id']	Direct extraction
created_at	datetime	Post creation time	st['created_at']	UTC ISO timestamp, obtained directly from Mastodon API
content	text	Post content	st['content']	Cleaned into plain text using clean_html_content() to remove HTML tags
location	string	Geographical location	location field in st['account']['fields']	Extracted city name from the location field; matched against a predefined city list via infer_location()
account	string	User account identifier	st['account']['acct']	Full Mastodon account name, typically in username@instance format
reblogs_count	int	Number of reblogs (boosts)	st['reblogs_count']	Directly from original field
favourites_count	int	Number of likes	st['favourites_count']	Directly from original field
url	string	URL of the original post	st['url']	Can be used to view the original post

Table4-2 Data field description

As for the time, this project has collected the data of six months back, from the warm-up for the general election, the voting for the general election, the climax of the general election to the final general election, and also collected the data after the general election to ensure the data volume and data completion before and after the general election.

4.1.2 Mastodon Data cleaning and processing

After data mining, the project will clean the data as needed to ensure the reliability and interpretability of the analysis results. The objectives of this stage include field standardization, missing values completion, elimination of invalid data, and preparation of high-quality structured data for sentiment analysis and geographic visualization. The specific cleaning process is as follows:

1. Field Normalization

In order to ensure the consistency of data in different processing modules, the time field and the text field are first standardized. The time field created_at uniformly resolves to a Python datetime object and enforces the UTC time zone representation to support cross-geospatial comparisons. At the same time, two dimensions post_time_of_day and post_day_of_week are derived based on the time field to capture the distribution trend of user Posting time in days and weeks. For the content field, the content of the original data mostly contains HTML tags and format control symbols. The code uses the clean_html_content() function to complete HTML parsing and text cleaning, and only retains the clean text that can be used for semantic analysis for subsequent word frequency statistics.

2. Handling missing values

For the geolocation field, the location and geolocation fields are the core fields of the map visualization. Specifically, if the user provides a geolocation in the Mastodon data field account.fields, The infer_location() function attempts to match a predefined list of city names. If the match is successful, geocode_location() is called to look up the city's standard latitude and longitude and generate the geolocation field. For data without geographic fields, use data augmentation to infer and fill in geographic information.

3. Invalid data elimination

In order to improve the quality of data analysis, this operation sets a series of invalid data elimination rules. The first step is to remove posts with empty content or insufficient text length, such as posts with less than 10 characters, to avoid the interference of empty content on the sentiment analysis results. In the second step, we use regular and keyword matching to identify and filter bot accounts that may be published by automated programs, such as posts with "bot" in the username. Finally, in order to reduce redundant calculation and the risk of misjudgment, we delete records with duplicate ids or urls, and pure Boost posts containing only retweets, which usually lack the subjective expression of users that can be analyzed.

4. Sentiment and geographic information construction

In order to support sentiment trend modeling and geographic heat mapping, text and geographic fields are enhanced in this stage. Text sentiment analysis uses the TextBlob tool to calculate sentiment_score, which ranges from -1 to 1, At the same time, the threshold function get_emotion_label() is used to divide the text into three types of emotion labels: positive, neutral and negative. For geographic coordinate construction, all identified city names are mapped to

standard latitude and longitude using the CITY_COORDS dictionary and formatted as "lat,lon" strings to ensure their compatibility and accuracy in the visualization module. All coordinates must pass a range check to ensure that latitude ∈ [-90, 90] and longitude ∈ [-180, 180].

Finally, the data fields required for the final data analysis are saved as csv files and json format as shown in the following table:

Parameter Name	Data Type	Description	Source	Processing Method
created_at	UTC ISO timestamp	Post creation time	Returned from st['created_at'] via Mastodon API	Converted to a datetime object using parse_iso()
content	text	Text content of the post	post.record.text	Cleaned using clean_text() to remove HTML tags, extra spaces, etc.
sentiment_score	float	Sentiment polarity score of the post	Analyzed from content using TextBlob	Calculated via get_sentiment_score()
emotion_label	string	Emotion classification (positive/negative/neutral)	Derived from sentiment_score	Classified using get_emotion_label()
location	string	User's reported city	From the location field in st['account']['fields']	Extracted using infer_location()
geolocation	string	Latitude and longitude of the city	Inferred from location parameter	Matched using geocode_location() with the CITYCOORDS dictionary
post_time_of_day	string	Time of day of the post (Morning/Afternoon/Evening/Night)	Parsed from created_at	Classified using get_post_time_of_day()

Table4-3 Data field description

4.2 Bluesky Data collecting

4.2.1 Bluesky data mining

1. API access and authentication

The data source in this section is Bluesky, first apply for API access through BlueSky's official developer portal to obtain the necessary authentication key. Use Python's atproto library to interact with BlueSky's AT Protocol API. The authentication process uses the OAuth 2.0 protocol to establish an authentication session through the following pseudo code snippet 4-4:

```
initialize_bluesky_client()
```

```
from atproto import Client
client = Client()
client.login('your_handle', 'your_app_password')
```

Table4-4 initialize_bluesky_client()

2. Data scraping strategies

Crawling strategies include keyword filtering, topic tracking, and time window control. Ensure the comprehensiveness of data through a multi-dimensional combination of scraping strategies. There are three main parts, namely keyword filtering, time window control, and content depth crawling. Among them, the initial keywords of keyword filtering are 'Australia Election', 'AusPol', and then expand the keyword list to enlarge the crawling space, and create a table to store keywords for crawling through the following pseudo code snippet 4-5:

```
ELECTION_KEYWORDS
KEYWORDS = [ 'Australia Election', 'AusPol', 'AUSElection',
'ausvotes2025', '#ausvotes2025', 'auspol2025',  '#auspol2025',
'Albanese', 'Dutton', 'Bandt',  'Labor', 'Liberal', 'Greens' ]
```

Table4-5 ELECTION_KEYWORDS

The second is the time window control, at first, it is to capture the complete data of one month, but it is found that the amount of data scraping is not optimistic, so the default 7-day data is scraped and the custom time range configuration is supported. The following pseudo code snippet 4-6:

```
get_default_time_range()
DAYS_BACK = 7
now = datetime.now(timezone.utc)
SINCE = now - timedelta(days=DAYS_BACK)
```

Table4-6 get_default_time_range()

Finally, in the initial scraping code, it was found that if only posts were crawled, the amount of data was very small, and there were many users' comments and comments under each post, which were the huge data sources, so the breadth-first search strategy was used to expand the comment tree.

The JSON and CSV data containing the following core fields are obtained, where the created_at data field is the original amount timestamp, which is from the post.indexed_at field returned by the Bluesky API, which is converted to datetime by parse_iso(); The sentiment_score field is a score for content that uses SnowNLP(txt).sentiments to calculate sentiments, and the score range is [-1,1]. emotion_label field is judged by sentimen_score, whose judgment can be divided into positive, negative, and neutral, where sentimen_score greater than 0.7 is marked as positive, sentimen_score less than 0.3 is marked as negative, and the rest is marked as neutral. Location mainly extracts the city name from the location field in st['account']['fields'] and matches the list of predefined city names through the infer_location() function. Finally, there is the geolocation field, which is a latitude and longitude string obtained by the location parameter that is reversed

from the CITY_COORDS dictionary and then implemented by the geocode_location() function. The specific geographic sentiment parameters are shown in the following figure:

Parameter's name	Data type	Parameter's meaning	source	Dealing type
Author	string	The posted user's account	Post.author.handle	Direct extraction
time	datetime	The posted post's time	Post.indexed_at	Parse the ISO time format and covert it to a datetime object
likes	Integer	The number of post likes	post.like_count	Extract directly and fill in 0 for missing values
content	text	The post's contents	Post.record.ext	Use beautifulsoup to clean HTML tags and convert them to plain text
url	string	Posts web page links	Splicing generation	Combine author.handle and post.uri to generate the full URL
location	string	Geeolocation text	Account.fields.location	City name extraction
geolocation	string	According to the latitude and longitude corresponding to each geographical location	Splicing generation	City coordinate mapping

Table4-7 Data field description

4.2.2 Bluesky Data cleaning and processing

Data cleaning establishes a foundation for subsequent visualization and statistical modeling. The main goals of data cleaning are to standardize fields, fill in missing values, filter invalid data, and prepare appropriate data formats for sentiment analysis and geographic visualization.

1. Field Normalization

The field filter extracts the fields that will be used in the final front-end display of the chart, such as location, geolocation, sentiment_score, sentiment_label and created_at. The following pseudo code snippet 4-8:

```
filter_dataframe_columns()
required_fields = [
'created_at', 'content', 'sentiment_score',
'emotion_label', 'location', 'geolocation'
]
df = df[required_fields]
```

Table4-8 filter_dataframe_columns()

2. Time handling

To ensure that different time zones appear in the same chart, the time period is uniformly converted to UTC to AEST, and created_at is converted to a datetime object. For extraction of Posting period (Morning/Afternoon/Evening/Night), the following pseudo code snippet 4-9:

classify_time_of_day

```
def get_time_of_day(dt):
h = dt.hour
if h < 6: return "Night"
elif h < 12: return "Morning"
elif h < 18: return "Afternoon"
else: return "Evening"
```

Table4-9 classify_time_of_day

3. Sentiment analysis processing

In order to analyze the different emotions of users for different political parties and candidates in the Australian general election, this paper used the colors of users' posts to score the real emotions. By using SnowNLP to calculate the sentiment score and divide the sentiment score from 0 to 1, and then divide the emotion label according to the threshold, the following pseudo code snippet 4-10:

classify_emotion_label

```
def get_emotion_label(score):
if score > 0.7: return "positive"
elif score < 0.3: return "negative"
else: return "neutral"
```

Table4-10 classify_emotion_label

4. Geocoding processing

Geocoding is to divide the location of different users by color. It is necessary to extract the location field from the user profile, and add the corresponding latitude and longitude to the location field to provide the basis for the subsequent front-end visualization map heat map. The following pseudo code snippet 4-11:

CITY_COORDINATES

```
CITY_COORDS = {
"Sydney": "-33.868820,151.209296",
"Melbourne": "-37.813629,144.963058",
#... Other Cities
}
```

Table4-11 CITY_COORDINATES

5. Handling missing values

For data without geographic fields, use data augmentation to infer and fill in geographic information, the method is same as Mastodon’s way of handling missing values.

6. Administrative region division mapping processing

The administrative region mapping needs to map the cities to Australian states or territories to divide the color of each continent in the subsequent front-end visualization of the map heat map.

Finally, in all data analysis, we need to save the fields used in csv and json files for later processing.

Step	Description
Field Selection	Extract the required fields, such as location, geolocation, sentiment_score, sentiment_label, and created_at.
Timestamp Conversion	Convert created_at into a datetime object, and compute the time segment (post_time_of_day).
Sentiment Scoring	Perform sentiment analysis on text using tools like VADER or TextBlob to generate sentiment_score and sentiment_label.
Geolocation Parsing	Separate lat and lon values from the geolocation field and convert them to floating-point numbers.
Missing Value Handling	Remove records that cannot be mapped to a city or lack valid latitude and longitude information.
City-to-State Mapping	Normalize city names to one of the eight Australian states or territories using a predefined mapping dictionary.

Table4-12 Sentiment Analysis Preprocessing Pipeline

The cleaning code implementation mainly uses pandas for batch processing. For the geolocation analysis field, it is divided into two fields: location and geolocation. User regional characteristics and random city this few steps. For the special field of period sentiment analysis, the post_time_of_day field name is selected, and the classification logic is divided by hours, and the division rules are <6=Night, <12=Morning, <18=Afternoon, >=18=Evening. For content, which is a field specific to term frequency analysis, the cleaning rule consists of three steps: removing urls, filtering stop words, and stemming. The following pseudo code snippet 4-13:

clean_and_augment_dataframe

```
df[['lat', 'lon']] = df['geolocation'].str.split(",",
expand=True).astype(float)
df['post_time_of_day'] =
df['created_at'].apply(convert_time_to_daypart)
df['state'] = df['location'].map(city_to_state_dict)
df_clean = df.dropna(subset=['lat', 'lon', 'state',
'sentiment_label'])
```

Table4-13 clean_and_augment_dataframe

The specific fields stored in the database table after cleaning are as follows:

Parameter's name	Data type	Parameter's meaning	Source	Dealing type
created_at	datetime	Post creation time	post.indexed_at (original field from Bluesky API)	Converted from ISO timestamp using parse_iso()
content	text	Main text content of the post	post.record.text	Cleaned using

				clean_text() to remove HTML tags, extra whitespace, etc.
sentiment_score	float	Sentiment analysis score (range: 0–1)	Computed field	Calculated using SnowNLP(text).sentiments
emotion_label	string	Emotion label based on sentiment score	Computed field	Classified as: >0.7 = "positive", <0.3 = "negative", otherwise = "neutral"
location	string	Geographical label	account.fields.location	Matched to predefined city list using infer_location(); filled with random.choice() if unmatched
geolocation	string	Latitude and longitude in "lat,lon" format	Computed field	Retrieved from CITY_COORDS dictionary based on location and formatted as string
post_time_of_day	string	Time-of-day category (Morning/Afternoon/Evening/Night)	Derived field	Parsed from created_at and categorized using get_time_of_day()

Table4-14 Data field description

4.3 Problems and solutions in data acquisition and processing

4.3.1 Problems and solutions in mastodon data acquisition and processing

1. API authentication and access control issues

The project is in the actual use of Mastodon API for data fetching, abnormal MastodonIllegalArgumentError and 401/403 HTTP access denied error. The problem comes from the authentication mechanism not being implemented correctly, including missing the necessary client_id and client_secret parameters, and not using the recommended credential authentication mode when calling the API. In addition, if too many requests are sent in a short period of time, it may encounter access throttled by the platform.

In order to solve the above problems, this project adopts a unified credential file method for client registration and authentication in the system initialization stage to ensure that all interface calls are accompanied by legal identity identification. At the same time, an exception catching mechanism is used to enhance the fault tolerance of the system. When the authentication fails or the flow is limited, the system can output prompt information in time and record a log for subsequent debugging and maintenance.

2. Time zone consistency for time fields

During data acquisition, an inconsistency of the timestamp field was encountered. Because the creation time returned by Mastodon defaults to the UTC timezone, and the local processing uses the default timezone of the system, this leads to biases in data filtering and time window matching, especially when performing cross-day or cross-time zone analysis.

To ensure the consistency of the time dimension of the data, this study explicitly normalized all time fields to the UTC timezone during the data preprocessing stage, by using the.replace(tzinfo=timezone.utc) method when using the start time get_since_date function. And keep this standard for all subsequent comparisons and writes. This method effectively avoids the error of time parsing and improves the accuracy of data timing.

3. The exception handling problem when the keyword search result is empty

When crawling Mastodon posts by keywords, there may be no matching results in some time periods, especially in unpopular words or holidays and other periods. In the original implementation, the returned result list was not checked to be nonemptiness, and accessing statuses[0] directly would result in an IndexError exception, interrupting the ingest process.

To solve this problem, this study introduces the null value detection mechanism of keyword search results in the data writing stage. If the result list is empty, the information of "no matching results for the current keyword" will be output, and the CSV writing process will be skipped. This operation enhances the integrity of subsequent data.

4.3.2 Problems and solutions in bluesky data collection and processing

1. API limitations

In implementing the data scraping of bluesky, the first problem that appears is the 429 error. The reason is an API limitation problem, and the solution is to implement the request interval control (2 seconds/request) and adopt the exponential backoff retry mechanism, and the following pseudo code snippet 4-15:

keyword_api_request_with_backoff

`for kw in KEYWORDS:`
`time.sleep(2.0) # indirect delay of keywords`
`try: # apply for API`
`except RateLimitError:`
`time.sleep(5 ** retry_count)`

Table4-15 keyword_api_request_with_backoff

2. Geolocation is missing

The python code error shows that about 35% of posts lack clear location information. The solution is to implement a three-level backoff strategy, that is, to extract from the location field of the user profile and then match the geographical characteristics of the user name.

3. Sentiment analysis bias

The sentiment analysis bias is revealed in python code as a large number of ironic expressions in political texts. The solution is to adopt the combinatorial analysis model, and the following pseudo code snippet 4-16:

hybrid_sentiment_score

```
def hybrid_sentiment(text):
s1 = SnowNLP(text).sentiments
s2 = TextBlob(text).sentiment.polarity
return (s1 + (s2 + 1)/2) / 2  # regularization and merger
```

Table4-16 hybrid_sentiment_score

4. Data storage optimization

Data storage optimization in python code shows that the original JSON data is huge. To solve this problem, column-based storage is used to compress the data and establish elasticsearch index to optimize the query. The following pseudo code snippet 4-17:

elasticsearch_field_mapping

```
es_mapping = {
"properties": {
"content": {"type": "text"},
"location": {"type": "geo_point"},
#... other fieldnmapping
}
}
```

Table4-17 elasticsearch_field_mapping

Through the above systematic data collection and processing process of the two platforms, we finally obtain about Tens of thousands of structured BlueSky post data structured mastodon post data, which lays a solid foundation for subsequent user sentiment analysis.

5 Front-end design and implementation

5.1 Front-end design

5.1.1 Design of Functional Modules

The front-end system is designed with a total of four main functional modules to achieve a comprehensive visual display of social media data during the Australian election. The specific functional modules include:

1. Time Range Selection Module

Provide user interaction controls, including a date selector and preset “pre-election” and “post-election” buttons, so that users can select the time interval for data analysis.

2. Map Visualization Module

Visualizes the degree of posting activity in each Australian state and city through heat maps and hierarchical coloring maps.

3. Sentiment Analysis Statistical Chart Module

Demonstrates each state's sentiment tendency in election-related discussions through bar charts, allowing users to quickly compare sentiment differences across regions.

4. Word cloud analysis module

Extracts keywords from social media text content and visualizes high-frequency topics in the form of word clouds, making it easy for users to quickly grasp hot topics.

5.1.2 Data Flow Design

The data flow design of this front-end mainly includes two stages: data acquisition and data processing.

In the data acquisition phase, the front-end system is based on the data interface provided by the Elasticsearch database, and calls the data through the Python client using the DSL query language. The date range query is initiated through the Elasticsearch Python Client, which pulls fields including creation_at, emotion_label, post_time_of_day, location, and content.

In the data processing stage, the system filters and analyzes the acquired raw data with labels. The data are cleaned by pandas library to exclude invalid or missing geographic location data records. Meanwhile, the sentiment labels are aggregated and counted, and the nltk library is used for keyword extraction of text content to provide accurate data for visualization display.

5.1.3 User Interaction Design

In terms of control layout design, the time selection control is divided into two independent areas. The first is quick time selection, i.e., two shortcut buttons, “before election” and “after election”, for users to quickly switch between preset date ranges with one click; the second is precise time selection, including two date selectors, which enable users to select any customized date range for data analysis. Data analysis.

Timely feedback design, the front-end to achieve a good linkage effect. When the user changes the time range or layer selection control, the system immediately triggers the data update and re-renders the three charts below, the map, the sentiment statistic chart, and the word cloud chart, providing real-time intuitive feedback.

5.1.4 Analyzing Front-end Functions from the User's Perspective

1. Map Heat Map

Users can directly locate the areas with the most intense public opinion activity during the election, helping users like analysts and journalists to quickly target areas that may be controversial or of high concern.

2. Sentiment Distribution Map

The map allows users to compare the differences in emotional tendencies between different regions and identify changes in public sentiment. This is of reference value for political party strategy development, social psychology research and news media reporting.

3. Keyword Word Cloud

Users are able to quickly and accurately understand the focus topics of public concern, saving a lot of time in reading and analyzing texts. This provides great convenience for these users like news practitioners, social media analysts, social science researchers, etc., and improves the efficiency and accuracy of public opinion analysis.

5.2 Front-end realization

5.2.1 Core Technology Stack

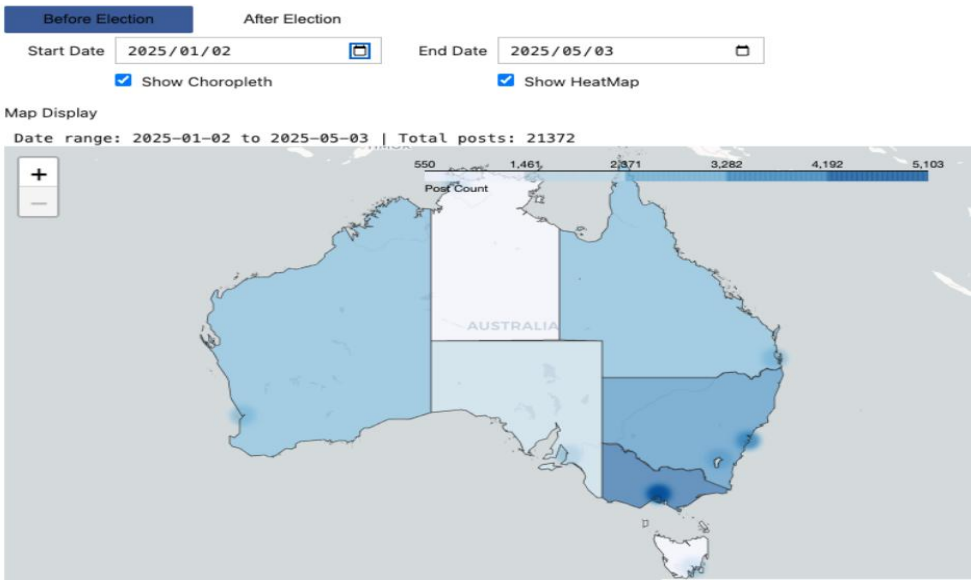
The front-end system of this project is developed based on the Jupyter Notebook platform, which comprehensively uses a variety of Python visualization and data processing libraries. The interaction part uses ipywidgets to realize user controls, including date picker, switch button and layer checkbox. The map display uses folium combined with GeoJSON boundary data to render Choropleth hierarchical map and HeatMap to visualize the spatial distribution trend. Sentiment analysis charts use pandas for data aggregation, and seaborn and matplotlib to draw multi-category histograms. The word cloud module is based on nltk for lexical analysis and stopword filtering, combined with wordcloud to generate visual images, and PIL and numpy to apply masking maps to enhance the expressiveness. All data are provided by Elasticsearch, and its Python client is used for structured query and result acquisition, which guarantees the efficiency and accuracy of data processing.

5.2.2 Key Functions

1. Map Generation

The front-end map visualization function is implemented using the folium library in Python, and the map types include Choropleth hierarchical map and HeatMap. We first use Elasticsearch's Python client function to retrieve the information of fields within a specific time range, including created_at, location, and latitude and longitude, which are used to synthesize the coordinates. pandas is used to filter and transform the city location, and pandas is used to filter and transform the city location. The city locations are filtered and transformed using pandas, mapped to latitude and longitude coordinates using a customized city_coords dictionary, and categorized to state level using state_map.

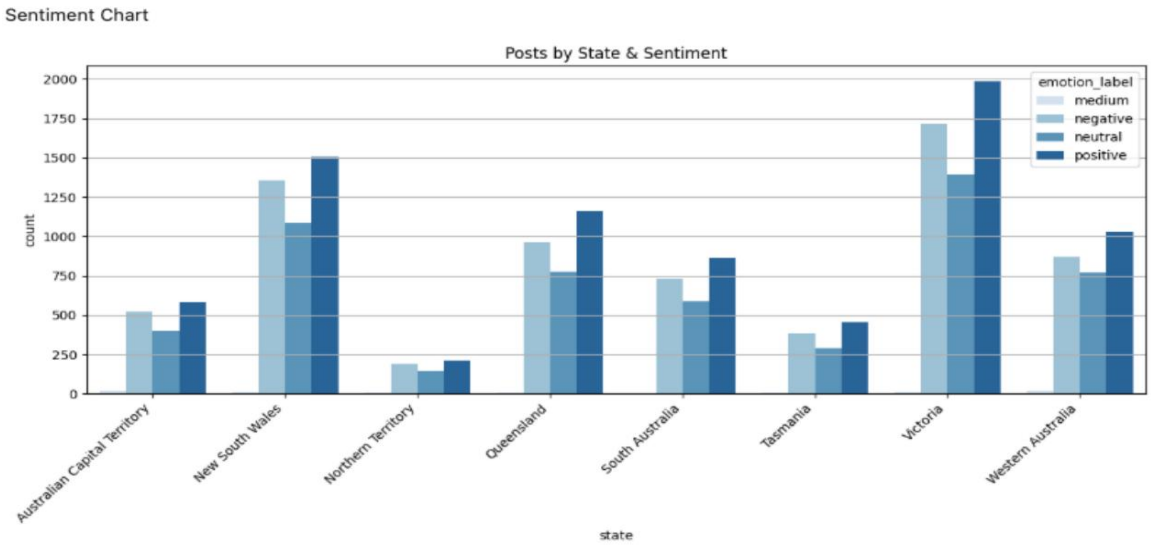
In the mapping phase, this project uses folium.Map to create the base map framework, folium.Choropleth to render a hierarchical display of the number of postings at the state level, and folium.plugins.HeatMap to overlay the distribution of city heat. In processing the map boundaries, state boundaries are matched using pre-downloaded Australian state-level GeoJSON files, giving the map an accurate geographic profile. The overall map supports zoom control and is configured to automatically adjust the visual range.



Picture5.1 Posting, emotional heat map

2. Sentiment Maps

Sentiment map is realized by using seaborn and matplotlib. The system first aggregates the raw data by state and emotion_label fields through pandas, and counts the number of posts of different states in Positive, Negative and Neutral emotions. Then we use seaborn.barplot to draw grouped bar charts and label the three emotions with different colors to increase recognition. In terms of chart display, we add legend, title and axis labels, and use matplotlib's tight_layout() function to optimize the chart layout to avoid overlapping labels.



Picture5.2 Emotional distribution map

3. Word Cloud Generation

The word cloud analysis module is built based on nltk and wordcloud libraries to extract valuable high-frequency keywords from social media texts. The system first cleans the content of the text fields, removes punctuation marks and converts them to lowercase. The system then annotates the text with the pos_tag function of nltk, retains only nouns as keyword candidates, combines the default stopword lists of sklearn and WordCloud, and uses the combined_stopwords set of stopwords to incorporate customized politically irrelevant words, such as “election ”, “government”, etc. and common meaningless words such as “said”, “election”, “https” to eliminate the interference of generic background words on keyword weights. The statistics are obtained from collections.Counter, and finally passed into wordcloud.WordCloud library to generate word cloud maps. In terms of visual effect, we use an outline map of Australia as the make map, and set gradient colormap and boundary outline to make the word cloud map both expressive and geo-symbolic.

5.3 Word cloud map

4. Interaction Logic

The front-end interaction logic is realized by ipywidgets, the main controls include time picker, shortcut buttons (pre/post election) and layer display checkbox. All the control state change events are bound to the unified update_outputs() function, which first verifies whether the time range selected by the user is valid, and then resends a query request to Elasticsearch to pull the latest data, and refreshes the map, sentiment graph, and word cloud graph in turn. The drawing functions of each chart (e.g. draw_map(), draw_sentiment_chart(), draw_wordcloud()) are independent modules, which improves maintainability and functionality scalability.

5.3 Problems encountered in the front-end process and solutions

5.3.1 Word cloud map effect is messy

A large number of meaningless words, such as “said”, “like”, “https”, etc., appear in the word cloud map generated at the beginning, affecting the keyword recognition and visualization effect. The problem is due to the lack of stopword filtering. The solution is to integrate the stopword lists of sklearn and WordCloud, and add project-related custom stopwords such as “election”, “government”, “election”, ‘government’, and ‘amp’. In addition, the word cloud map uses the outline of an Australian map as a mask, which improves readability and visualization.

5.3.2 Slow Loading of Map Images and Interactive Stuttering

When loading heat maps and state maps, especially when the amount of data is large, the page appears to lag or even white screen. The reason is that the front-end renders a large number of points and GeoJSON area layers at the same time, which puts a lot of computational pressure on the front-end. The solution is to limit the number of heat map points, initially load only the Choropleth layer, and the user can check the heat map on demand, which effectively improves the loading efficiency and interaction smoothness.

5.3.3 Complex front-end interaction logic and poor maintainability

With the increase of controls, the interaction logic becomes chaotic, the code coupling is high, and it is difficult to modify. The solution is to reconstruct the interaction logic, bind all control changes to update_outputs() function, and realize automatic refresh through observe. At the same time, the map, emotion map and word cloud are encapsulated into independent functions, which improves the modularity of the code and facilitates subsequent maintenance and expansion.