Cluster and Cloud Computing Assignment 2

Team 6
Hongyu Jin 1427319, Shiyao Xue 1555313,
Bin Liang 1118639, Xuanhao Zhang 1133384, Yilin Chen 1131563

May 22, 2024

1 Introduction

The project aims to build a system that is able to handle certain scenarios that relate the life in Australia. It uses a set of tools to implement a cloud cluster-based full-stack system. Those tools include Kubernetes (K8s), MRC, Fission, Elastic Search, Juypter Notebook and Kibana. Development using GitLab as version control and collaborative coding tool. Among those tools, MRC provides the cloud computing resource, K8s provides Cluster management, and fission is used as the back end of the system. Juypter provides a user interface to use the functionality of the system. In addition, Elastic Search handles the data storage for this system. The scenarios of this system use dynamic weather and economic-related data to analyze their relationship with some other data.
Inside the Juypter Notebook, there will be a few sections that each represent a different scenario that we analyzed. use can input some parameters according to instructions, those parameters usually define the location and year of the analysis performances. It will send the request to cluster and visualize the result on the Juypter notebook.

1.1 Link of Resource

- **Gitlab:** stores the resource code of this system. link
- **Youtube:** demonstration video of the system. link

2 System Architecture and Design

2.1 System Architecture

Figure 1 shows the basic architectural structure of this system. The Kubernetes(K8s) cluster platform is running the Melbourne Research Cloud(MRC). The MRC is a Cloud that above to provide computing resources that support the K8s running on. K8s is able to group and manage multiple nodes together to finish heavy tasks involving complex environments. In this system, a K8s containing 4 nodes has been created, one of them is acting as a master node that manages the state and operation of the cluster. For each of the nodes, there are 2c9g with 30GB storage assigned. Three main services have been deployed on the cluster. They are:

- **Fission**: A framework for serverless function, in this system, it acts as a back end. It will accept the request from the front end, and perform the corresponding function to respond to the request.
- **ElasticSearch**: A Restful search engine that acts as a database for this system. Data from external API dynamically and from SUDO manually will be stored into ElasticSearch as the index. Fission function is able to query the ElasticSearch to get the data they need.
- **Kibana**: This is a visualization service that allows developers to access Elastic Search efficiently.

As the figure demonstrates the services of each may be constructed by a set of pods that run on different nodes. For example, ElasticSearch has two master pods in Worker Node 0 and Node 1 separately. By putting each of the service's pods on different nodes can significantly improve the availability and fault

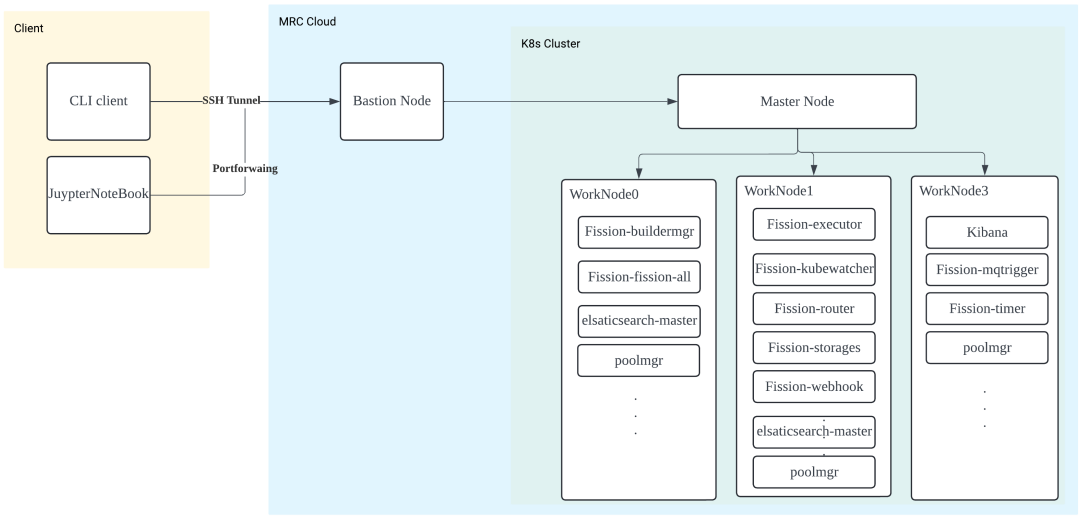

Figure 1: Architecture of System

tolerance of the system, for example, if one of the nodes meeting some issue, elastic search can still use another master in a different node to fulfill the request as much as possible. Besides, this distribution of service can balance the load of each node because each node will handle the different tasks for a service. For example when using the send request to run the fission function, if the request has been sent with high frequency, fission can run pods on different nodes that have available resources to make sure the functions can be executed as soon as possible. With our current usage of service, the current CPU requests for each node of our current system are 51%, 57%, 70%, and 77%. Even though that is not absolute equal for each node, each node is able to maintain the usage of CPU and memory at a reasonable level.

There is an extra bastion VM node build on MRC that is used for direct communication with the cluster. Due to the security setting of MRC, an external client is unable to communicate with the cluster inside MRC directly. Therefore, this Bastion Node is present as the entry point of K8s cluster of this system. The developer and front end need to establish the SSH tunneling with this Bastion Node in order to perform actions on Cluster. Besides, a port forwarding from the local port to the port of service on the cluster is needed to access the corresponding port of service locally. This requirement of security may increase the cost during development.

2.2 Service Interaction

As 2.1 mentioned, the Fission function and Elasticsearch on the cluster need to collaborate together in order to implement the functionality of the system. Figure 2 shows the detail of how Front end, fission, and Elasticsearch interact with each other. First, when we create functions in fission, a package and env need to be specified to make sure a suitable container can be loaded to a pod for the function to run. A route with Restful API is also needed to allow this function to be accessed by other services. After that, when the user establishes that ssh tunneling and port forwarding on their machine. They can use Jupyter Notebook to send an HTTP request to their local host port that forwards to the cluster's port. The Ingress of K8s will send that request to the fission router pod. The router pods have the record of each restful API created. it can determine which function should run, and load the corresponding container in a pod. Since we deployed fission with default configure, it will choose an available pod of a pool to let the function run. While the function running, it's able to send a request to restful API of Elastic search on K8s with the query that specifies the information they need. Since the Elastic Search is also on the cluster, a function can directly query it using internal address and certification. One of the two Master pods of Elastic Search will decide which shard should be used

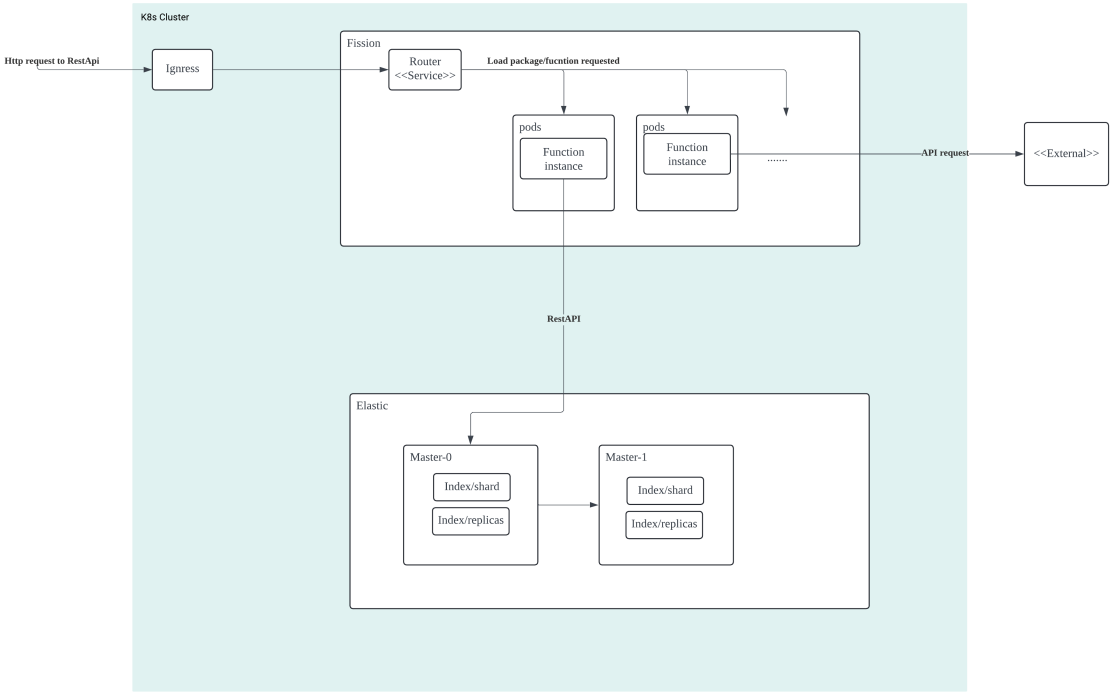

Figure 2: Architecture of System

to get the information function needed and return it in the response. Besides, some of our fission functions will also send the request to an external API in order to gain the dynamic data from a third party and save it to elastic search using restful API too. In Figure 2, there are two master pods for Elastic Search, only one of them at a time will be used as the real master. These two master pods are in different nodes to increase the availability and fault tolerance. Out of the same purpose, while we develop we give most index 3 shards and 1 or 2 replicas.

2.3 Functions and Index Overview

Figure three shows the function we have in the fission and the index we have in elastic search. It indicates what a function will interact with, for example, ftpfetchweather will query the external API to get the weather data, and the weatherinsert function will interact with elastic search to add queried weather data to elastic search. Since the fission function main to be serverless, we designed to reduce the task of most functions to ensure some of them can avoid monolithic. However, In order to perform the functionality of the system, multiple functions will still have to talk to each other. This could be done by sending multiple requests from the front end but this will break the intent of the front end by letting the user read too many codes. So the team decided to create functions that represent the working flow of a functionality. Those functions will call other functions in sequences in order to harvest or analyze data. ftpcallstation is an example, it will first get all station name that weather need to obtain, and it will use station names to get data from external API, and then insert to elastic search using weatherinsert. The calling of other functions from one function is also done through the Restful API. The running function will send an HTTP request to the internal address of the fission router, the fission router will then start deciding the pod and container that the new function has been called will run on.

3 Scenario description

In this section, we introduce four scenario we chose. Our four scenarios include weather-related and car accident-related as well as real-time data on pollen counts from various locations. This mainly

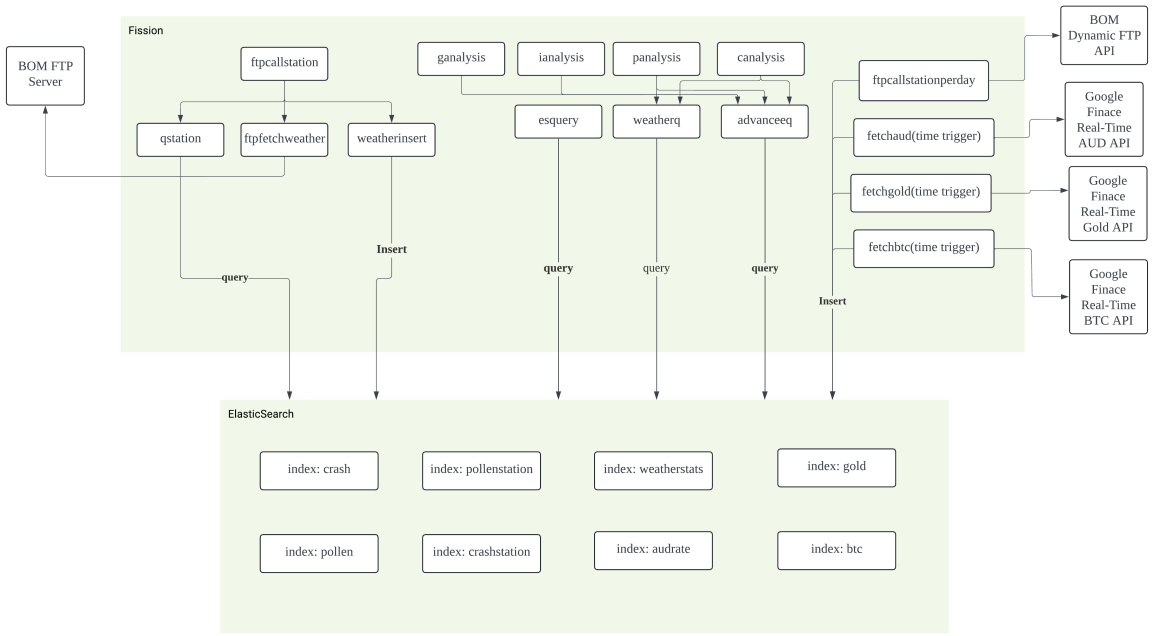

Figure 3: Architecture of System

shows our processing of geographic information. And in the scenario with financial and economic data, we mainly focus on real-time and correlation analysis, which reflects our ability to process real-time data and time series data in this scenario.

3.1 Weather and Tasmanian traffic crash analysis

In this scenario, we accomplished the correlation analysis between weather and traffic crashes by processing the traffic crash data of Tasmania and the weather data of the corresponding locations, including temperature, rainfall, and so on. The data for traffic crash is mainly from SUDO, which contains a count of the number of car accidents in Tasmania from 2010 to 2020, within a range of 21 regions. and the weather data is mainly from BOM's api including all weather data from 2009 to now. In this scenario, we will guide the user's travel or transportation choices through maps, rainfall probability, and probability of occurrence of traffic crashes.

3.2 Weather and Australian national pollen count analysis

In the pollen and weather relationship scenario, we mainly acquired pollen data from SUDO which is counted from 2016 to 2020 and includes pollen counts from Melbourne, Brisbane and other locations throughout Australia. And real-time weather data from BOM throughtout whole Australia. Using the relationship between temperature and rainfall, we analyzed the relationship between pollen counts and overall weather data across Australia over multiple years.Considering that there are many pollen-allergic individuals and that direct exposure to pollen can lead to severe allergic reactions, our work can guide people as to whether or not to take relevant protection.

3.3 Analysis of the Australian Dollar Exchange Rate in Relation to Bitcoin Price and Gold Price

We complete the correlation analysis between the AUD/USD exchange rate and the prices of Bitcoin and Gold by obtaining them in real time. This data comes mainly from external API requests in real time. We analyze the relationship between the AUD/USD exchange rate and the price of Bitcoin and the relationship between the AUD/USD exchange rate and the price of gold in this scenario,

respectively. From there we can use our real-time analysis to provide investors with investment and risk alert recommendations.

3.4 Analysis of the Australian dollar exchange rate and income inequality

In this scenario, we focus on analyzing the relationship between the AUD exchange rate and per capita income as well as the coefficient of income inequality (Gini coefficient).Gini coefficient represents income inequality in a region; the lower the Gini coefficient, the better the region is at reducing the wealth gap. The AUD/USD exchange rate data is mainly obtained from an external real-time API from 2010 to now, while the income data and the coefficient of inequality are obtained from the ABS government statistics from 2015 to 2020. Our work can help the public to understand the intrinsic link between INCOME and the exchange rate, which can potentially help to eliminate potential Inequality and poverty.

4 ElsaticSearch Design

4.1 Creating Index in Elasticsearch

Creating an index in Elasticsearch is a crucial step for organizing and querying data efficiently. The provided example demonstrates how to create an index named `weatherstats` using a curL command. Below, we will break down the code and explain each part of the process in detail. Additionally, we will describe the structure and properties of various other indices, including `pollen`, `pollenstation`, `crash`, `crashstation`, `incomeinequality`, `goldprice`, `ginico`, `btcprice`, and `audrate`.

4.1.1 Creating Index with bash script

We mainly use bash scripts to create the index, specifying the name of the index, its mapping and shards, and replicates. BASH scripts contain some critical parts, "-XPUT" indicates the HTTP method as PUT, which is used for creating or updating a resource. "indexname" specifys the name of the index to be created.`"settings"`: Defines index settings. `"number_of_shards"`: Specifies the number of primary shards for the index. Shards are individual instances of Lucene and allow Elasticsearch to scale horizontally. `"number_of_replicas"`: Specifies the number of replica shards, which provide redundancy and high availability.`"mappings"`: Defines the schema for the index. `"dynamic": "strict"`: By setting `"dynamic": "strict"`, we ensure consistency in the indexed data by only allowing predefined fields, thereby preventing the accidental addition of unwanted fields. `"properties"`: Specifies the fields and their data types.

Authentication

- `--user 'elastic:elastic'`: Provides basic authentication credentials (`username:password`) for accessing the Elasticsearch instance.

4.1.2 Analysis

Creating an index in Elasticsearch with the specified settings and mappings is an essential part of preparing the search engine for data ingestion and query. Here are the key points:

- **Shards and Replicas**: The configuration of shards and replicas ensures the index is distributed across multiple nodes for scalability and high availability. In this example, the index has 3 primary shards and 2 replicas for each shard.
- **Mappings**: Defining mappings upfront with strict dynamic settings prevents unexpected data types from being indexed, which maintains data consistency and integrity. The specified fields (`id`, `location`, `date`, `rain`, `maxTemp`, `minTemp`, `year`, `month`, `day`) cover various aspects of weather statistics.

- **Field Types**: Using appropriate data types (`keyword`, `text`, `float`, `integer`) ensures efficient indexing and querying. For instance, `keyword` is used for exact matches, `text` for full-text search, `float` for numeric values with decimals, and `integer` for whole numbers. Ensuring correct data types improves search accuracy and performance.

- **Authentication**: The use of basic authentication (`elastic:elastic`) is standard for securing access to the Elasticsearch instance. However, in a production environment, more secure methods like API keys or OAuth might be preferred.

4.1.3 Additional Index Descriptions

Pollen index includes fields such as `date` (text field for dates), `id` (keyword field for unique identifiers), `location` (text field for location names), `other` (short field for other pollen types), and `poaceae` (short field for poaceae pollen type).
PollenStation index corresponds to the `location` field in the **Pollen** index and contains weather station data for the respective locations, including `statename` (text field for state names) and `stationname` (text field for station names).
Crash index comprises `crash_count` (integer field for the number of crashes), `date` (text field for dates), `id` (keyword field for unique identifiers), and `location` (text field for location names).
CrashStation index corresponds to the `location` field in the **Crash** index and includes weather station data for the respective locations with `statename` and `stationname` as text fields.
IncomeInequality index features fields such as `Inequality` (float field for inequality metrics), MeanWeekly (float field for mean weekly income), `MeanWeeklyDisposable` (float field for mean weekly disposable income), `MedianWeekly` (float field for median weekly income), `MedianWeeklyDisposable` (float field for median weekly disposable income), and `Year` (text field for the year).
GoldPrice index includes `Date` (text field for dates) and `Value` (float field for gold price values).
Ginico index consists of `Year` (text field for the year), `ginicoefficient` (float field for the Gini coefficient), and `householdincomeginicoefficient` (float field for the household income Gini coefficient).
BTCPrice index comprises `Date` (text field for dates) and `Value` (float field for Bitcoin price values).
Location index includes `location`, `Latitude` and `Longitude`. This index, which includes the relationship between weather station and latitude/longitude, allows our system to have the ability to map geographic information to station names.
AUDRate index includes `Date` (text field for dates) and `Value` (float field for AUD rate values, indicating the exchange rate of the Australian Dollar against other currencies).

4.2 Index Management

The picture below shows the index management interface in Kibana, which provides important information about our indices.

Name	Health ↓	Status	Primaries	Replicas	Docs count	Storage size
ginico	green	open	3	1	13	32.2kb
crashstation	green	open	3	1	21	29.98kb
btcprice	green	open	3	1	1827	236.89kb
audrate	green	open	3	1	5001	543.26kb

Figure 4: Kibana Index Management Interface

We use the Kibana User Interface to monitor our index status. The interface offers several advantages:

- **Real-time Monitoring**: Kibana provides real-time updates on the health and status of indices.

- **Detailed Metrics**: It shows detailed metrics such as the number of shards, replicas, and the size of each index.

- **User-friendly Interface**: The graphical interface makes it easy to navigate and manage indices without needing to use the command line.

The health status of our indices is shown in the picture. The color codes indicate the health of the index: Green means the index is healthy, with nodes sufficient to accommodate all run automatically every day. no shard duplicated on the same node. Yellow indicates that shards can form complete documents, but not all replicas are allocated correctly, potentially leading to duplication of shards on the same node. Red means the index is unhealthy, with incomplete documents or a number of nodes less than the number of primary shards, usually due to data loss or node failure. The impact of these health colors is significant: Green signifies optimal performance and redundancy with no action needed, Yellow suggests reduced redundancy which could lead to data loss if a node fails, impacting system reliability—especially for indices with large data volumes, and Red indicates potential data loss and query failures, requiring immediate action to restore node functionality or reallocate shards.

4.3 Reasons for Index Health Status in Our System

In our system, the health status of indices can vary based on several factors:

- **Data Volume**: Indices with large data volumes, such as `weatherstats` and `crash`, might show a yellow status because they require more resources for replication. These large indices are more prone to resource constraints, leading to incomplete replication.
- **Node Capacity**: The overall capacity of our nodes affects whether they can host all required shards and replicas. If the nodes are under-provisioned, even smaller indices might experience issues, but typically, smaller indices like `ginico` remain green due to lower resource demands.
- **Index Management**: Proper management of indices, including periodic optimization and allocation adjustments, helps maintain green status. Indices that are regularly monitored and optimized,
- **Cluster Configuration**: The configuration of the Elasticsearch cluster, including settings for shard allocation and replica management, directly impacts index health. Ensuring balanced and optimal configuration can prevent yellow or red statuses.

5 System Functionality

5.1 Functions for Harvest and Process Data from external API

5.1.1 ftpfetchweather

This function will take two parameters to request the specified weather data from the BOM website, he will take the station name and month parameters and get the specified weather data through the ftp server. The data will be processed into a serialization dataframe object. This function includes some basic data preprocessing. The data obtained from the FTP server is not in a format that can be directly converted into a pandas DataFrame, and it contains a lot of unnecessary text information. Therefore, this function also performs basic data preprocessing to clean and structure the data appropriately.

5.1.2 fetchaud

Fetchaud is a function that harvests AUD/USD real-time data from the Google Finance website and insert them into ElasticSearch. We also set a time trigger on this function to let this function run automatically every day. This function will get the latest data by getting the server time as a timestamp. This function also includes data preprocessing and ensures that the data is in a format that can be directly inserted into Elasticsearch. The reason for this function's design, which violates the general principle of having one function perform a single task, will be discussed in Section 7.2.2.

5.1.3 fetchbtc

Fetchbtc is a function that harvest BTC price real-time data from google finance website with beautifulsoup4 and insert them into ElasticSearch. In the meantime, we've timetriggered fission to make this function execute regularly every day. This function will get the latest data by getting the server time as a timestamp. This function also includes data preprocessing and ensures that the data is in

a format that can be directly inserted into Elasticsearch. The reason for this function's design, which violates the general principle of having one function perform a single task, will be discussed in Section 7.2.2.

5.1.4 fetchgold

Fetchgold is a function that harvest gold prince real-time data from *investing.com* website and insert them into ElasticSearch. In this function, because the target site has an anti-crawler mechanism, we use some request camouflage mechanism at the same time. At the same time, we set up a fission timer to ensure that we can automatically get the latest data every day. This function will get the latest data by getting the server time as a timestamp. This function also includes data preprocessing and ensures that the data is in a format that can be directly inserted into Elasticsearch. The reason for this function's design, which violates the general principle of having one function perform a single task, will be discussed in Section 7.2.2.

5.1.5 transformlocation

In our system, there is a need to harmonize geolocation information between different datasets, such as the conversion between longitude and dimension and Australian geographic information such as Statistical Area code (SA code). This function provides the conversion between longitude and dimension and most of the required geographic information such as state, SA area or other more detailed geographic location information. This function will get the latest data by getting the server time as a timestamp.

5.2 Functions for Insert into ElasticSearch

5.2.1 ftpcallstation

The ftpcallstation is the function that does the insertion work overall, and it will receive the source of the data you want to choose to get, in our scenario it is the pollen dataset and the corresponding weather information for the car accident dataset. At the same time specify a certain time range interval, select the weather data to be requested, and finally execute weatherinsert this function.So we first obtain all the target station names by calling elasticquery and then use the retrieved station names to make requests to the FTP server. Therefore, in the ftpfetchweather interface, we request data based on the station names and date range. In this function, we perform two iterations based on the date range and all station locations to complete the data request process and insert the data into Elasticsearch. This function is mainly through all the stations and time ranges to be requested, call the above ftpfetchweather interface for data acquisition and insertion.

5.2.2 ftpcallstationperday

This function will fetch the latest weather data from the BOM FTP server based on the time of day and insert it into the database. This function calls ftpfetchweather as described above and ensures that we get the latest weather data for the day by getting the server timestamp. This function shares a similar data request logic with **ftpcallstation**. The difference lies in modifying the process from accepting user parameters to automatically reading the server time. At the same time, we use a fission trigger to ensure that this function will run regularly every day.

5.2.3 weatherinsert

the weatherinsert function is in charge of insert input data into the weather-related index. by sending Post a list of JSON to restful API "/weatherinster/indexname", it can send a query to the elastic search of the system and adding the request body to the the correspond index. for this function, it only accepts weather stats and weather text as index name. weather text is only for testing purposes in the system.

5.3 Functions for Query ElasticSearch

This subsection describes the functionality of three API endpoints defined for querying Elasticsearch indices: `esquery`, `advanceeq`, and `weatherq`. These facilitate different querying capabilities based on the parameters provided. All query functions use `request.headers.get('X-Fission-Params-...')` to retrieve parameters from the request headers and construct queries dynamically based on the specified parameters.

5.3.1 esquery

The `esquery` is the simplest and is used to retrieve all documents from a specified index in ElasticSearch. It performs a `match_all` query to fetch all documents within the specified index. The URL for this endpoint is `/esq/indexname/{index}`. This endpoint returns all documents from the specified index, including the full document details for each entry. The primary use case is to query indices like `crashstation` and `weatherstation`.

5.3.2 advanceeq

The `advanceeq` endpoint is designed to query an Elasticsearch index using specific date and location parameters, dynamically constructing a query to retrieve relevant data. The URL format for this endpoint is `/advanceeq/indexname/{index}/date/{date}/location/{location}`. Required parameters include `index=[string]` (the name of the index), `date=[string]` (the date to be queried, which can be `false` or a specific date string like "2021-01-01"), and `location=[string]` (the location to be queried, which can also be `false` or a specific location string like "Melbourne"). This endpoint returns documents matching the specified date and location criteria. The response includes fields relevant to the index, such as `poaceae`, `other`, `location`, and `date` for the pollen index, or `crash_count`, `location`, and `date` for the crash index. The query is constructed using Elasticsearch's `bool` query with `must` clauses for date and location if specified, ensuring the retrieval of relevant data based on the index type and specified parameters.

5.3.3 weatherq

The `weatherq` endpoint is used to query weather-related data from an Elasticsearch index based on year, month, day, location, and weather parameters. It supports grouping data by month and constructs dynamic queries accordingly. The URL format for this endpoint is `/weather/year/{year}/month/{month}/day/{day}/location/{location}/weatherstat/{weather}/bymonth/{bymonth}/index/{index}`. Required parameters include `index=[string]` (the name of the index), `year=[integer]` (the year to be queried, which can be `false` or a specific year integer like 2020), `month=[integer]` (the month to be queried, which can be `false` or a specific month integer like 10), `day=[integer]` (the day to be queried, which can be `false` or a specific day integer like 16), `location=[string]` (the location to be queried, which can be `false` or a specific location string like "Melbourne"), `weather=[string]` (the weather state to be queried, such as "rain", "maxTemp", or "minTemp"), and `bymonth=[string]` (whether the data should be grouped by month, either `false` or `true`). This endpoint returns weather-related data that match the specified criteria. If the `bymonth` parameter is set to `true`, the data is aggregated by month, and the response includes aggregated statistics for the specified weather parameter. This query retrieves weather-related data based on the specified parameters and supports grouping data by month using Elasticsearch's `composite` aggregation. The query is constructed dynamically with parameters such as year, month, day, location, and weather. The unique aspect of this endpoint is its capability to perform aggregations based on the `bymonth` parameter, which distinguishes it from other endpoints and allows for more complex data retrieval and analysis, particularly useful for weather-related data.

5.4 Functions for Process and Analysis

In this section, we describe the process and analysis involved in retrieving and processing data using our APIs within Fission. The functions for process and analysis aims to unify the format and return data that can be directly used for visualization on the front-end.

5.4.1 Bitcoin, Gold, and AUD to USD

First, we retrieve the data using the specified indices. For example, using the URL `http://localhost:9090/esq/indexname/btcprice`, we fetch the Bitcoin price data. Similar URLs are used to fetch gold and AUD data. Since the date formats in these tables are inconsistent, the necessary data is extracted from the `response.json()['hits']` of each API.The date columns in the three tables are converted to `to_datetime` and sorted in chronological order. The data columns are all named `Value`. After renaming these columns, we perform an outer merge on the three tables due to significant missing values in gold and AUD data. The merged table can be filtered using `start_date` and `end_date` parameters, which allow us to select the required data within a specific time range. We set a `fill` parameter to determine whether to fill missing values using `.interpolate(method='time')`. After converting to float, two new columns, `BitcoinPurchasingPower` and `GoldPurchasingPower`, are created to represent the purchasing power of 100,000 AUD worth of Bitcoin and gold. The output is a `datetimeindex` with two prices, one exchange rate, and two purchasing power columns. The output is packed as a fission function and can be retrieved using API http://localhost:9090/ganalyisi/fill/{fill}/startdate/{startdate}/enddate/{enddate}. This dataframe, converted to JSON, is the output of the API. The Fission API can accept `start_date`, `end_date`, and `fill` as parameters. The start date and end date defined the range that analysis will performances on. `fill` is intended to automatically fill possible missing values.

Date	BitcoinPrice	GoldPrice	AUDtoUSD	BitcoinPurchasingPower	GoldPurchasingPower
2016-09-13	610.92	1323.650000	0.747000	122.274602	56.434858
2016-09-14	608.82	1321.750000	0.747000	122.696363	56.515983
2016-09-15	610.38	1310.800000	0.751400	123.103640	57.323772
2016-09-16	609.11	1308.350000	0.749000	122.966295	57.247678
2016-09-17	607.04	1310.516667	0.750467	123.627218	57.264946
2016-09-18	611.58	1312.683333	0.751933	122.949301	57.282157
2016-09-19	610.19	1314.850000	0.753400	123.469739	57.299312
2016-09-20	608.66	1313.800000	0.757900	124.519436	57.687624

Figure 5: Bitcoin, Gold, and AUD Analysis Output

5.4.2 Pollen and Weather

We retrieve data by passing `year` and `location` parameters. The same location is referred to as `location_2` for pollen and `location_1` for weather data. The conversion is done using `location_1 = location_2.replace('_', '%20').upper()`. Pollen data is fetched using the URL `http://localhost:9090/advanceeq/indexname/pollen/date /{year_1}/location/{location_2}`. Weather data is retrieved using URL `http://localhost:9090/weather/year/{year_1}/month/false/day/false/location/location_1/weatherstat/rain/bymonth/true/index/weatherstats`, with `weatherstat` and `bymonth` parameters adjusted as Rain/maxTemp, True/False, accordingly.The wanted data is located in: - `pollen_data = res_pollen.json()['hits']` - `weather_data = res_weather.json()['groupby']['buckets']` - `weather_data_day = res_weather_day.json()['hits']` - `maxTemp_data = res_maxTemp.json()['groupby']['buckets']` Pollen data, initially in weekly format, is converted to year-month format and aggregated monthly. Pollen values are set `.astype(float)` and date column is processed `to_period('M')`. Monthly rain and temperature data are processed similarly, with rain values rounded to one decimal place. Our custom feature Zero_rain_Percent is aggregated on daily rain data using `.groupby('date')['rain'].apply(lambda x:(x == 0.0).mean()).reset_index()`
The output is packed to a fission function and can be retrieved using the API `http://localhost:9090/panalysis/year/{year_1}/pollen/{location_p}/weather/{location_w}`. The local implementation `http://local:9200/pollenstation /_search` helps get a location list and fetch data for all specified year ranges. The temperature here is the max temperature. Zero_Rain_Percent here indicates the percentage of days in a month that have no rain.

	date	maxRain	avgRain	sumRain	Zero_Rain_Percent	location	poaceae	other	maxTemp	avgTemp	sumTemp
0	2017-09	0.2	0.0	0.2	0.966667	campbelltown_(mount_annan)	306.0	4562.0	35.7	23.7	711.7
1	2017-10	25.6	1.5	46.8	0.612903	campbelltown_(mount_annan)	462.0	2420.0	35.0	26.1	809.1
2	2017-11	7.0	1.0	30.2	0.566667	campbelltown_(mount_annan)	330.0	1062.0	33.6	25.8	772.6

Figure 6: Pollen and Weather Analysis Output

5.4.3 Crash and Weather

The process for `crash_weather` is similar to `pollen_weather`. We retrieve data by passing the year, crash location, and weather location parameters. The output including crash and weather data is packed to a fission function and can be retrieved using the API `http://localhost:9090/canalysis/year/{year_1}/crash/{location_c}/weather/{location_w}`.

	location	crash_count	maxRain	avgRain	sumRain	Zero_Rain_Percent	maxTemp	avgTemp	sumTemp	date.day	...
0	bushy_park_(bushy_park_estates)	14	18.0	0.9	27.6	0.870968	36.1	25.3	784.9	31	...
1	bushy_park_(bushy_park_estates)	13	9.6	0.6	16.4	0.620690	32.1	24.3	705.6	29	...
2	bushy_park_(bushy_park_estates)	7	2.4	0.3	8.2	0.741935	31.3	23.3	723.8	31	...
3	bushy_park_(bushy_park_estates)	7	4.8	0.4	13.4	0.766667	26.1	20.6	616.7	30	...
4	bushy_park_(bushy_park_estates)	9	15.0	3.0	94.4	0.451613	21.8	14.3	443.5	31	...
5	bushy_park_(bushy_park_estates)	3	28.6	2.3	68.0	0.433333	18.4	12.3	370.2	30	...
6	bushy_park_(bushy_park_estates)	14	23.0	3.4	104.0	0.354839	17.1	12.9	399.0	31	...
7	bushy_park_(bushy_park_estates)	6	5.4	0.9	28.4	0.451613	19.6	13.6	420.2	31	...
8	bushy_park_(bushy_park_estates)	5	35.0	1.7	50.0	0.600000	22.8	16.9	508.0	30	...
9	bushy_park_(bushy_park_estates)	6	14.2	2.8	85.6	0.419355	24.7	14.7	454.3	31	...
10	bushy_park_(bushy_park_estates)	15	7.6	1.1	31.8	0.466667	29.3	19.2	577.2	30	...
11	bushy_park_(bushy_park_estates)	8	11.6	1.8	55.2	0.451613	35.7	23.3	722.9	31	...

Figure 7: Crash and Weather Analysis Output

5.4.4 AUD to USD, Income, and Gini Coefficient

First, we retrieve the data using the specified indices. The date columns in the AUD exchange rate data are converted to `to_datetime` and sorted in chronological order. Income and Gini data are aggregated over two-year periods, while AUD exchange rates are available on a daily basis. We remove any duplicate entries using `drop_duplicates` to ensure data integrity. Next, we group the AUD exchange rates into two-year periods using the following method: `df['TwoYearGroup'] = df['Date'].dt.year - (df['Date'].dt.year - 2001)%2` This assigns each date to its corresponding two-year group starting from 2005-06. For example, 2006 and 2007 are grouped as 200607. For each two-year group, we calculate the average AUD to USD exchange rate. We then combine the two-year grouped AUD data with the corresponding income and Gini data. This involves aggregating income and Gini data to match the format of the AUD data groups. The combined data includes columns for the two-year period, average AUD to USD exchange rate, mean and median weekly income (disposable or not), and Gini coefficients. The packed fission function can be called using URL `http://localhost:9090/ianalysis`

	Year	AverageAUDtoUSD	MeanWeekly	MedianWeekly	MeanWeeklyDisposable	MedianWeeklyDisposable	ginicoefficient	householdincomeginicoefficient
0	200506	0.756554	1789	1426	883	772	0.314	0.425
1	200708	0.821525	2123	1656	1049	886	0.336	0.438
2	200910	0.859664	2058	1610	1034	872	0.329	0.428
3	201112	1.033603	2137	1669	1062	915	0.32	0.427
4	201314	0.935060	2273	1705	1099	930	0.333	0.446
5	201516	0.746828	2253	1727	1078	912	0.323	0.434
6	201718	0.757488	2310	1753	1094	926	0.328	0.439
7	201920	0.691415	2329	1786	1124	959	0.324	0.436

Figure 8: Income and Gini Analysis Output

6 Front-end Analysis

6.1 Main Functions

In the front-end, our analysis of weather and pollen as well as weather and traffic crashes primarily utilize two functions: get_pollen_weather_df and get_crash_weather_df. These functions can accept three parameters: start_year, end_year, and a locations list. Invoking the function will cyclically call our panalysis and analysis fission function based on the start and end times, returning JSON data for one year at a specific location each time. An identifier is created based on the data, in the form of yearlocation, and then data is added to dataframes[identifier] for every loop. Afterward, it prints the shape of all dataframes in the dict to showcase if any months are missing. Finally, it renames all the dataframes created within dataframes following the naming convention of location_no_parenthesis_year_df. For the analysis of the Australian Dollar to US Dollar exchange rate with Bitcoin and gold prices, and the analysis of the Australian Dollar to US Dollar exchange rate with income inequality, we primarily directly invoke the fission function ganalysis and ianalysis. We call esq function to get stationname list and geographical information.

6.2 Graphs and analysis

6.2.1 Scenario 1: Weather and Australian national pollen count analysis

In this scenario, we will primarily analyze how pollen concentration is affected. The conclusion aims to help users predict or decide whether to take protective measures against pollen allergies. We are mainly looking for the relationship between weather and pollen levels, hoping that users can determine whether to take pollen protection based on the weather information they receive. The two weather realated analysis will first illustrate the scenario of a single location in a specific year. Pertaining to Campbelltown in the year 2019, it is discernible from the correlation matrix that the most positive variable associated with Poaceae, which denotes pollen concentration, is Zero_Rain_Percent (r = 0.83). Conversely, the most negatively correlated variable is sumRain (r = -0.87).

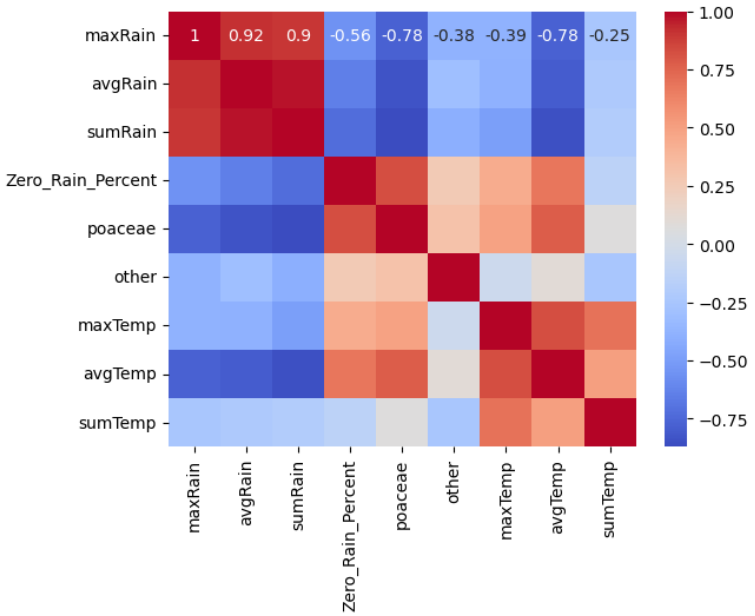

Figure 9: Correlation Matrix for campbelltown_2019_df

Figure 10 depicts the relationship of monthly precipitation and pollen concentration for 2019 in Campbelltown. Rainfall, shown as the blue bars, evidently display an inverse relationship with pollen concentration. Precipitation can clean pollen particles from the air from the atmosphere, thereby diminishing airborne pollen concentrations. The data pattern shows a strong negative correlation between pollen

levels and rainfall. When rainfall is high, pollen levels are very low. This phenomenon is particularly evident in the months of October and November.

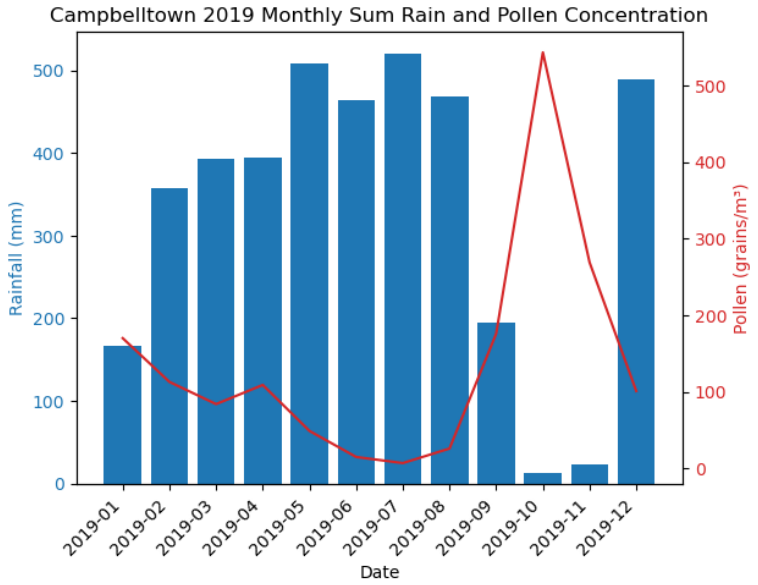

Figure 10: Campbelltown Monthly Sum Rain and Pollen Concentration

Figure 11 compares the average temperature with pollen concentration, a direct proportionality is apparent. Elevated temperatures correlate with the commencement of the pollen season, attributed to the phenological phases of many plants initiating blooming and pollen dissemination under warmer climatic conditions.

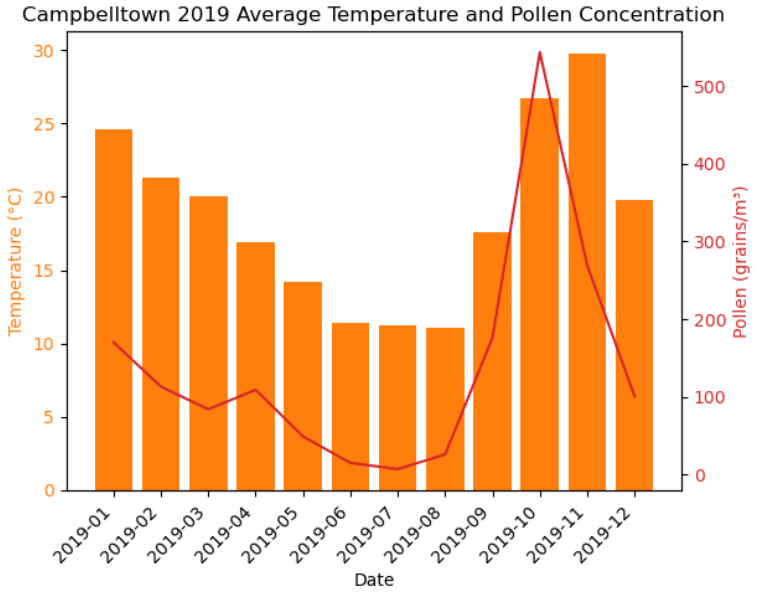

Figure 11: Campbelltown Average Temperature and Pollen Concentration

Moreover, the occurrence of days with zero precipitation within a month, an index we created and quantified in this study, emerges as significantly relevant to pollen concentrations. This connection is plausible, considering that negligible precipitation volumes suffice in cleansing the atmosphere of pollen.

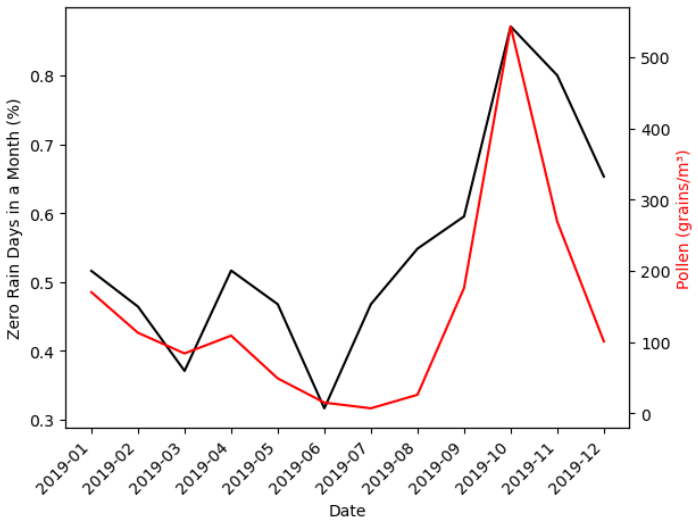

Figure 12: Campbelltown Zero Rain Days Percent in a Month and Pollen Concentration

Expanding the analysis to encompass all collected data, we put all the locations in a list and fetch year from 2016 to 2019. Figure 13 shows a notable decline in the correlation between pollen concentration and climate features. This reveals that some factors beyond climatic conditions significantly influence pollen concentrations, as they have been changed by mixed locations.

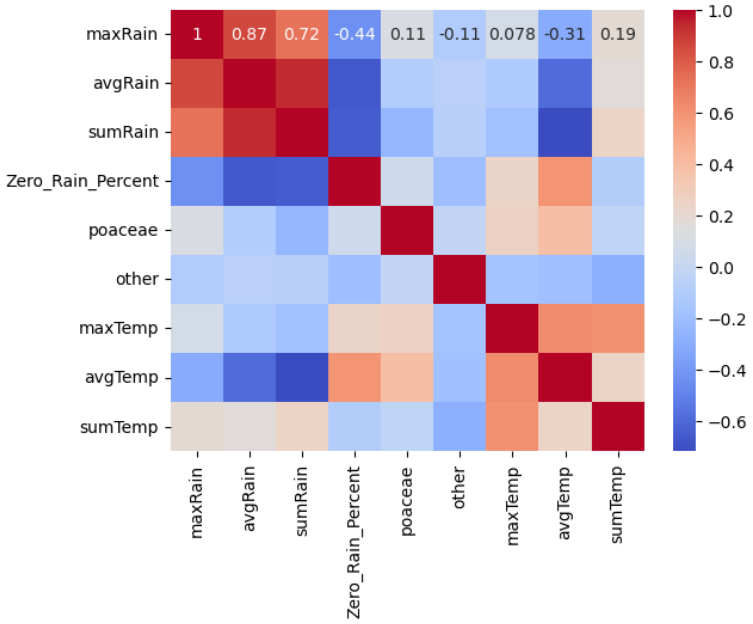

Figure 13: Correlation Matrix for All Pollen Data Collected

6.2.2 Scenario 2: Weather and Tasmanian traffic crash analysis

In this scenario, we will primarily analyze the relationship between weather factors and the number of car accidents. The conclusion aims to help users predict travel safety and make informed decisions about their mode of transportation. Additionally, in the following analysis, we will note that the correlation between weather and car accidents varies across different regions. Pertaining to Bushy Park in the year 2016, the most positive correlated feature for crash count is average temperature, with a correlation coefficient of 0.39. Such a value suggests a relatively insubstantial association between the crash accident amount and prevailing weather within the locale.

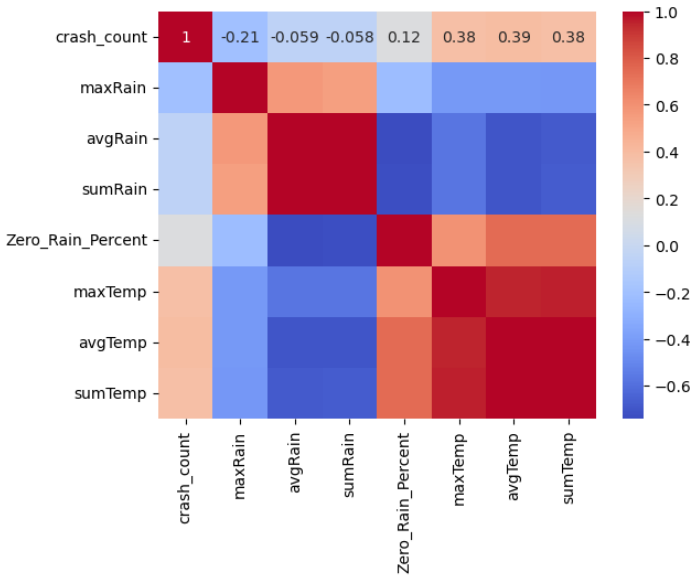

Figure 14: Correlation Matrix for bushy_park_2016_df

Expanding the analysis to encompass all collected data from 2012 to 2019, we use folium to render a Tasmania traffic crashes heat map with the option to select specific years for visualization by repetitively pairing crash_count to the location coordinates tuple and recreates a heat_map tuple for 7 times.The annual variation in crash count at each location is minimal, resulting in a degree of uniformity to the visual output across different years. However, a zoomed in inspection of Launceston will reveal an augmented brightness in 2017 compared to other years, stem from a crash count spike.

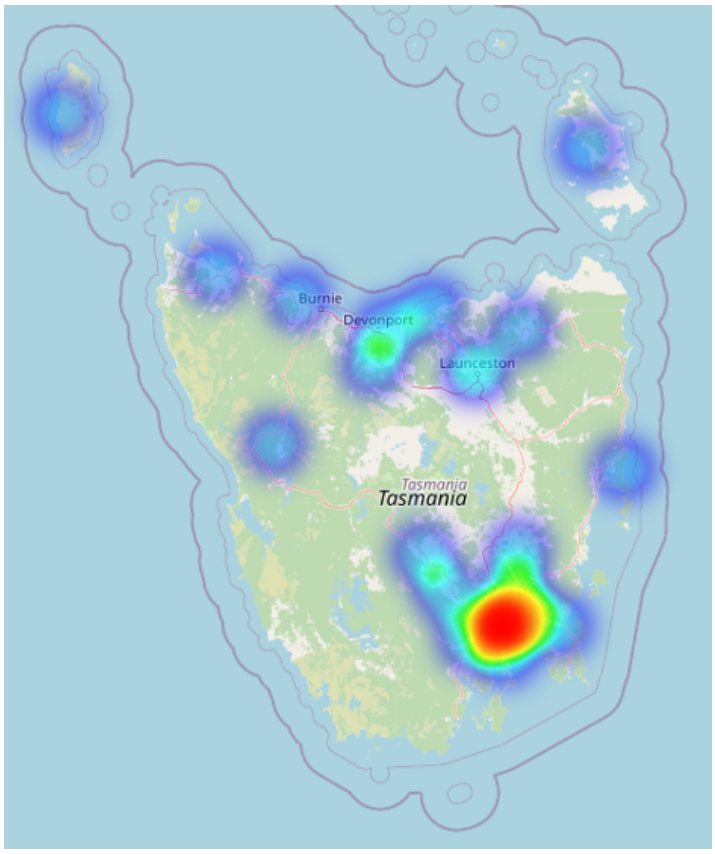

Figure 15: Tasmania Crash Count Heat Map

6.2.3 Scenario 3: Analysis of the Australian Dollar Exchange Rate in Relation to Bitcoin Price and Gold Price

Figure 16 depicts the trend of Bitcoin and gold price from September 2016 to September 2022, with the green line representing the trajectory of Bitcoin and the orange line denoting that of gold. It has been observed that there exists a lagged relationship between the price movements of Bitcoin and gold. Upon shifting the gold data by 222 days, as indicated by the red line in the figure, it becomes evident that in 2021 the adjusted trend of gold coincides almost exactly with the dates of Bitcoin's maximum and local minimum extreme values. This temporal adjustment has enhanced the correlation between the prices of gold and Bitcoin from 0.64 to 0.88.

Figure 16: Bitcoin and Gold Price Trend

Figures 17 and 18 illustrate the relative trends between the AUD to USD exchange rate and the prices of gold and Bitcoin, respectively. It is apparent that the year 2020 marks a significant turning point. Prior to 2020, the exchange rate of the Australian Dollar to US Dollar, as well as the prices of gold and Bitcoin, exhibited no correlation or a weak negative correlation. However, following the onset of the pandemic, both the exchange rate of the Australian Dollar to US Dollar and the prices of gold and Bitcoin have demonstrated a much higher degree of correlation.

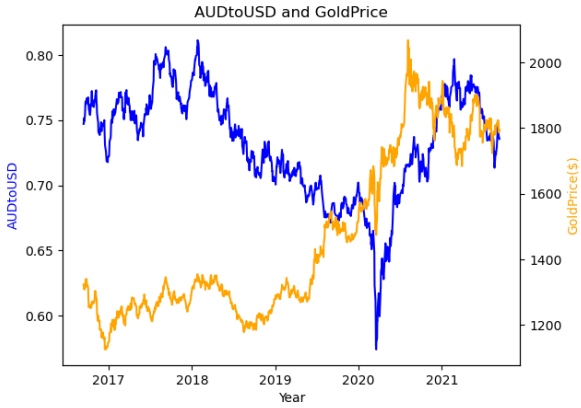

Figure 17: AUDtoUSD and GoldPrice

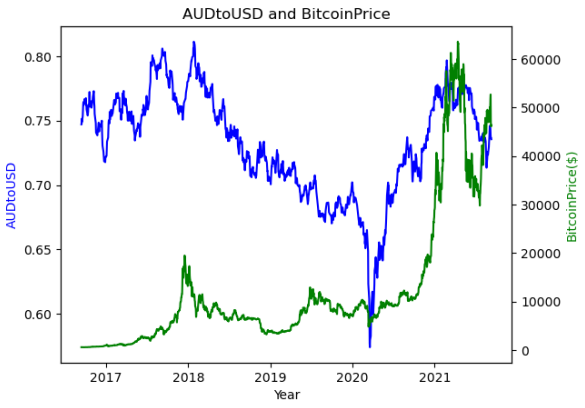

Figure 18: AUDtoUSD and BitcoinPrice

6.2.4 Scenario 4: Analysis of the Australian dollar exchange rate and income inequality

Figure 19 presents the overall trends of the AUD to USD exchange rate and the Gini coefficient, indicating a lack of correlation between the two. However, an examination of the data around the year 2011 highlights a notable event: the exchange rate of the AUD reached a historic peak, surpassing the value of one USD. Concurrently, the Gini coefficient dropped to its lowest value within a five-year period surrounding this point. This observation that the Gini coefficient is at its lowest when the national currency is at its strongest in years suggests a relationship between economic prosperity and income equality.

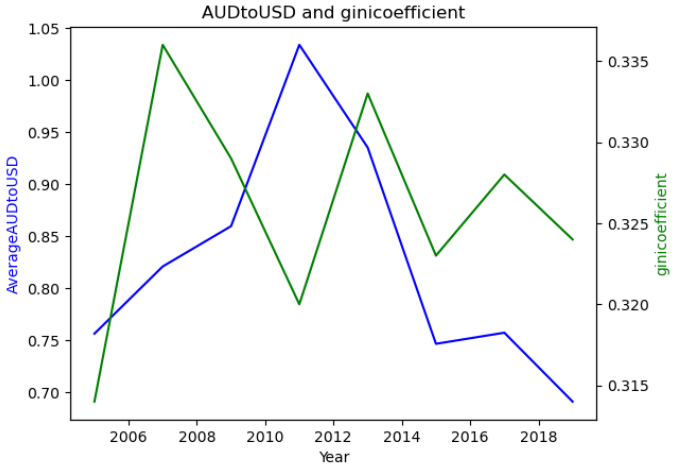

Figure 19: AUDtoUSD and Gini Coefficient

As depicted in Figure 20, the decline of the AUD to USD exchange rate after 2011, concurrently with the year-over-year increase in Australia's median weekly disposable income, presents an intriguing phenomenon. While the depreciation of a currency can suggest a variety of economic challenges, it does not necessarily interfere with the augmentation of disposable income for individuals within the country. A weaker exchange rate can make imports more expensive, as it costs more AUD to buy the same amount of foreign goods or services. In Australian supermarkets, products labeled as proudly local produced are ubiquitous, and the majority of goods display the proportion of ingredients sourced from Australia. From this observation, we infer that Australians place a significant emphasis on domestic production and manufacturing. This emphasis suggests a reduced dependency on imported goods and services, which can lead to a smaller impact of the AUD depreciation on disposable income.

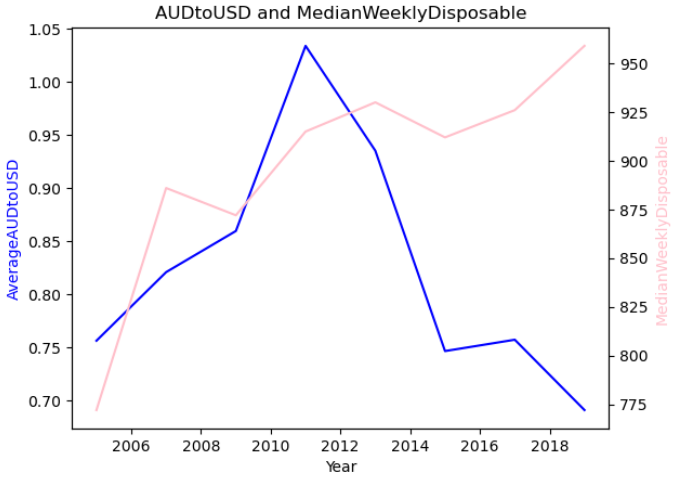

Figure 20: AUDtoUSD and Median Weekly Disposable

7 Discussion on Tools and Processes

In this section, we discuss the advantages, disadvantages, issues, and solutions related to the tools and processes used in our project, including Kubernetes (K8s), Fission, Elasticsearch, Jupyter Notebook, and Git.

7.1 Pros and Cons of Tools

7.1.1 MRC

The Melbourne Research Cloud (MRC) offers significant advantages for managing and deploying our system, particularly given its flexibility and robust infrastructure. One of the primary benefits is the scalability it provides. The MRC enables us to scale our computational resources up or down based on the project's needs, ensuring that we can handle varying workloads efficiently. Another advantage of the MRC is its integration with various research tools and platforms. It supports a wide range of software and frameworks commonly used in research, such as Kubernetes. This compatibility simplifies our workflow and allows us to use the suitable tools for data analysis and deployment without worrying about compatibility issues.

The dependency on network stability is one disadvantage. The performance and reliability of the MRC heavily depend on a stable and fast internet connection. Our team members need to have good connections with the Unimelb network. Network issues can disrupt access to cloud resources, leading to interruptions in data processing and analysis.

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
elastic-3dyo6amplfj4-master-0	109m	5%	5741Mi	64%
elastic-3dyo6amplfj4-node-0	91m	4%	3926Mi	43%
elastic-3dyo6amplfj4-node-1	219m	10%	8043Mi	90%
elastic-3dyo6amplfj4-node-2	83m	4%	6094Mi	68%

Figure 21: K8s node CPU and Memory usage

Another disadvantage of using MRC is relevant to resource allocation. Figure 19 shows the actual usage of CPU and memory of each node. Even though k8s are able to distribute tasks on different nodes, the actual usage of resources may vary under different scenarios. In our case, node 1 has memory usage that is 90%. That is much larger compared with other nodes and almost used all memory. However, when we obtained that situation, we noticed that extending the resources of an instance of MRC is actually very inconvenient. It is not possible to allocate more CPU and memory directly to some instances. Instead, you may need to create a new instance with more resources with the image of the current instance. That means the resource management and allocation need to be planned more carefully before the implementation, otherwise, there will be more cost and unpredictable issues when trying to extend the resource later. Especially the instances used for running K8s which has a relative more complex architecture.

7.1.2 Kubernetes (K8s)

Kubernetes (K8s) has proven to be a powerful and efficient tool for managing the deployment, scaling, and operation of containerized applications within our systems. One of the most significant advantages is its ability to easily handle large-scale distributed environments. Kubernetes' orchestration capabilities ensure that our applications are always running in the desired state, with automated deployments and rollbacks, self-healing mechanisms, and efficient load balancing across clusters. This is particularly beneficial for systems where we process large amounts of data and require high availability and reliability.

Kubernetes' scalability is another key advantage. It allows us to scale applications based on demand, ensuring our systems can handle different workloads without manual intervention. This automatic scaling is critical to maintaining performance during peak hours and optimizing resource utilization

during periods of low demand. In addition, Kubernetes provides powerful monitoring and logging tools to help us maintain visibility into system performance, detect problems early, and take steps to resolve them.

Kubernetes supports declarative configuration through YAML files, simplifying the management of infrastructure as code. This approach ensures that our deployments are consistent, repeatable, and manageable. Additionally, the Kubernetes ecosystem includes tools like Helm for package management, enhancing our ability to manage complex deployments and improve service reliability.

Despite the many benefits, using Kubernetes also comes with some challenges. Setting up and managing a Kubernetes cluster can be very complex for us beginners, especially for team members who don't have extensive experience in container orchestration. The initial learning is difficult and requires an in-depth understanding of concepts such as Pods, services, deployment, and storage. This complexity extends to the maintenance of the cluster, where issues related to networking, storage, and security can arise and require a lot of knowledge to resolve, and our team spends a lot of time on this. Also as section 7.1.1 and figure 19 shows, the resource usage of each node is may not eaual all the time. the pods on one node may not have sufficient resources to execute while other nodes still have resources.

7.1.3 Fission

Advantages of the Use of YAML Specifications to Deploy Function The use of YAML specifications to deploy functions in our system play an important role for our system. YAML supports complex configurations, allowing us to define detailed settings for our Fission functions, such as environment variables, packages, router, triggers, and resource limits, all in a single file. This centralized configuration approach streamlines the deployment process and ensures consistency and efficiency. In our system, the use of YAML for deploying Fission functions has significantly improved our workflow by providing a clear, consistent, and manageable approach to function deployment, reducing manual effort and minimizing deployment-related issues.

Advantages of Fission-Based Analysis Performing data processing and analysis server-side offers several key benefits. It reduces the computational load on client devices, ensuring faster response times and a smoother user experience. By centralizing data processing, we ensure consistent results, as the analysis logic is maintained in one place rather than being distributed across various client applications. This also enhances data security, as sensitive data can be processed and filtered on the server before being sent to the client, ensuring compliance with privacy regulations. Server-side processing leverages the scalability of the backend infrastructure, enabling it to handle large datasets and complex computations more effectively than client-side processing. Additionally, keeping the analysis logic on the server simplifies maintenance and updates. Changes to the analysis algorithms can be deployed centrally without requiring updates to all client applications. Overall, this approach provides a more efficient, consistent, secure, and scalable solution for data processing and analysis while also improving maintainability.

Discussion of Using Elasticsearch Queries in Fission Function Custom data retrieval is a significant advantage of our system, as different endpoints allow for tailored queries that meet specific needs. For instance, `esquery` can retrieve all documents from an index, which is useful for general data inspection or when the complete dataset is required. `advanceeq` can filter data based on date and location, making it ideal for more specific queries where only relevant data subsets are needed. `weatherq` can aggregate weather data by month, providing a higher-level summary of weather patterns over time. This custom approach ensures that each query retrieves only the necessary data, improving efficiency and relevance. In our system, this means users can efficiently access specific datasets like pollen levels, crash statistics, or weather patterns without unnecessary data overload. Improved performance is another key benefit. Structuring queries to target specific indices and using precise filters enhances performance by reducing the amount of data Elasticsearch needs to process. This reduces the load on the Elasticsearch cluster and speeds up query response times. For example, by using filters in `advanceeq` and aggregations in `weatherq`, the system can quickly narrow down the dataset to the most relevant records. For our large datasets, such as crash data and weather statistics, this ensures rapid and efficient data retrieval, enhancing user experience. In addition, the dynamic construction of

queries based on user-specified parameters allows for high flexibility. Users can modify their requests to retrieve data that meets their exact requirements without needing changes to the backend code. This is achieved through flexible parameter handling, where the presence or absence of certain parameters can change the nature of the query dynamically. In our system, this flexibility allows users to customize their data queries for specific years, locations, and other parameters, making the system highly adaptable to diverse needs. Moreover, using multiple indices and query types supports horizontal scaling. Different queries can be handled by different nodes in a distributed Elasticsearch cluster, ensuring that the system can handle large datasets and high query volumes efficiently. This distributed approach helps maintain performance and availability even as data volume and user demands grow. For our system, which processes large volumes of data from multiple sources, scalability is crucial to maintaining consistent performance. By defining specific fields and using strict mappings, data consistency is maintained across different indices. This approach ensures that queries return accurate and reliable data. The use of strict mappings prevents the accidental addition of unwanted fields, thus maintaining a consistent schema that all queries can rely on. In our system, this consistency is vital for ensuring that users receive reliable and accurate data, whether they are querying financial statistics, weather data, or pollen counts.

However, managing multiple queries and ensuring they work correctly with various indices can be complex. Each query must be carefully maintained to ensure it performs as expected across different indices. Comprehensive documentation and testing are essential to manage this complexity and ensure that all endpoints remain functional and efficient. In our system, the complexity arises from the diverse nature of the data sources and the need to maintain multiple query endpoints for different types of data. In addition, the current error handling mechanisms could be improved to provide more detailed feedback when queries fail. Enhanced error messages can help diagnose issues more quickly, improving the reliability of the endpoints. Furthermore, implementing structured error responses and logging can aid in troubleshooting and maintaining system health. This is particularly important for our system, where data from multiple sources can lead to varied and unexpected query results. Ensuring that all required parameters are valid and correctly formatted before constructing queries can prevent errors and improve robustness. Implementing stricter validation checks and clear error messages for invalid parameters can enhance user experience and reduce the likelihood of malformed queries. In our system, this is crucial for maintaining data integrity and ensuring that users do not experience disruptions due to incorrect parameter usage. Further optimization of queries can improve performance and reduce the load on Elasticsearch. This can include indexing commonly queried fields, using caching mechanisms, and optimizing query structures. Regularly reviewing and updating queries to ensure they are as efficient as possible can help maintain high performance. For our system, this means ensuring that frequently accessed data, such as real-time weather updates or financial statistics, can be retrieved quickly and efficiently.

Overall, the use of multiple Elasticsearch queries provides a powerful and flexible way to retrieve data. While there are some challenges associated with managing complexity and ensuring security, the benefits of tailored data retrieval, improved performance, and scalability make it a valuable approach for handling large and diverse datasets. By addressing the areas for improvement, the system can become even more robust and efficient, ensuring reliable data retrieval and analysis capabilities. This is particularly important for our system, which must manage and analyze large volumes of diverse data from various sources to provide accurate and timely insights.

7.1.4 Elasticsearch

Benefits of Creating Multiple Indices Creating multiple indices allows us to segment our data logically. For instance, separating weather, pollen, crash, and financial data into distinct indices simplifies management and makes it easier to query specific datasets without sifting through unrelated information. This organization is crucial for our system as it handles diverse datasets, ensuring that each type of data is stored and retrieved efficiently. In addition, distributing data across multiple indices enables Elasticsearch to optimize search performance by leveraging its distributed architecture. Each index can be allocated to different nodes, allowing parallel processing of queries. For our system, this means faster search responses and more efficient resource usage, which is particularly important given the large volumes of data we process, such as `weatherstats` and `crash` indices. Furthermore,

with multiple indices, our system can scale horizontally to handle increasing data volumes and query loads. As our datasets grow, we can add more nodes to the Elasticsearch cluster and distribute indices across them. This ensures that performance remains consistent even as the amount of data and the number of users querying the system increase. Different indices also allow us to implement more precise access controls. For example, we can restrict access to sensitive financial data while making weather and pollen data publicly accessible. This improves security and data governance within our system, ensuring that users only access the data they are authorized to view. Each index can be configured with custom settings tailored to its specific data and usage patterns. For example, indices with high query rates, like `weatherstats`, might have more shards to distribute the load, while smaller indices like `ginico` might have fewer shards and replicas. This flexibility allows us to optimize the performance and resource allocation for each type of data, enhancing the overall efficiency of our system. By following this structure, we can create a robust Elasticsearch index that suits our specific data and query needs.

7.1.5 Jupyter Notebook

Jupyter Notebook is a powerful tool that offers an interactive environment for developing and sharing code, which is particularly beneficial for our data-intensive project. One of the primary advantages is its support for real-time code execution and visualization. This feature allows our team to run code cells independently, visualize data with inline charts, and immediately see the effects of changes, facilitating a more iterative and explorative approach to data analysis. In addition, Jupyter Notebooks also enhance collaboration among team members. Notebooks can be easily shared and versioned using Git, allowing different team members to work on separate parts of the analysis and later merge their contributions.

Despite its strengths, Jupyter Notebook has several drawbacks that impact our project. The most significant disadvantage is Jupyter Notebooks cannot produce highly complicated visualizations and interactive interfaces like dedicated web applications, limiting the ability to create advanced interactive visualizations.

7.1.6 Git

Git is an important tool for version control and collaborative development in our projects. Git's branching and merging capabilities provide a powerful framework for parallel development. Team members can develop new features, bug fixes, or testing in independent branches, ensuring that the main codebase remains stable. Once the changes have been reviewed and tested, they can be merged back into the master branch. This workflow supports efficient collaboration and continuous integration, which are critical to our project's iterative development process. Moreover, the ability to track changes and maintain a history of all modifications is critical to our team because it allows us to identify who made specific changes, understand the context of those changes, and revert to a previous state if necessary. However, Git also brings some challenges. Managing merge conflicts, especially during periods of frequent commits, can be time-consuming and requires careful coordination. When multiple team members make changes to the same part of the code base, conflicts often arise that require manual resolution to ensure that the final merged code is correct and valid. If not handled properly, this process can slow down development and introduce errors.

7.2 Issues and Solutions

7.2.1 Elasticsearch Query Match Issue

While writing Elasticsearch Query DSL and using `match`, we encountered an issue where the queries were not performing exact matches but were instead matching many similar entries. This was problematic because it resulted in retrieving more data than intended, reducing the efficiency and accuracy of our system. After investigation, we discovered that the solution was to set the data type to `keyword`. By doing this, we can ensure that the fields were treated as exact values, enabling precise matches. This adjustment may improve the accuracy of our queries and ensured that only the intended data was retrieved, enhancing overall system performance.

7.2.2 Fission Function Mutually call issue

In function like $fetchaud$ and $ftpcallstation$, we design the function as a whole process from harvesting data from external and insert to elastic. And we find that if in a single function we need to call other function deployed by fission, latency will be very long. There maybe some reasons like cold starts, Inter-Function Overhead and Resource Contention. We have tried redeploying functions, but to no avail. In our system, fetching data from external api and process data and inserting it into the database were originally designed as several different interfaces, but we found that the latency of the system would become very high when calling these several functions with a unified function. Especially when fetching data like the price of bitcoin, we have a very large amount of data. So when it comes to getting real-time data, or even when there are tens of thousands of records, it can cause our system to become completely unusable.

For the need to call multiple fission function function we have two solutions, for the demand for a large number of real-time calls to the data, such as fetchaud and fetchbtc interfaces, he has a high demand for real-time, so we will be all the function integrated together, to avoid funciton between each other to call. Although in the design of ideas, contrary to certain object-oriented and functional atomization of the principle , but to meet the running time requirements of this function. So these three fetch-related functions include all the functions from harvest to insert.
For another kind of function, such as ftpcallstation, he only needs to get the latest weather data of the day at a certain moment in the day and save it, we still choose, will insert, process and harvest from external api as a separate function exists, and then use the ftpcallstaion, and we use timetrigger to make it execute regularly every day. In future work, our expected solution is, based on the system footprint, to find a time of day when the system footprint is low to call this interface, to ensure that the high frequency of calls to each other will not lead to system congestion.

7.2.3 Functions designed to combat disaster recovery

Our insert work has been brought a lot of troubles because of the MRC issue, for example, in the uniwireless network environment, the connection of bastion often suddenly timeout, our insert work has to be interrupted. At the beginning of the function design, we needed to insert all the historical data we needed into the database, which included historical weather data for most states with a time horizon of more than ten years. For efficiency, this forced us to use parallel, non-time-series based simultaneous inserts, otherwise we would have had trouble completing our work before due. However, this mechanism makes it impossible to breakpoint and re-work, and each time the bastion connection is broken, we have to start inserting all over again, so we modified the logic of the insert function to accept a time range, which allows us to not only insert in parallel, but also to use a log to record which data we have been We can also use a log to keep track of what data we've processed, and we end up inserting all the data into elasticsearch even if the operation is interrupted several times.

7.2.4 Challenges with Setting Up Dependencies

We faced significant challenges with setting up dependencies, particularly around installing `scikit-learn`, `pytorch`, `tensorflow` and their related dependencies. When trying to deploy a function that required these libraries, we encountered numerous issues and failed to run the function. These issues included compatibility problems with the build of packages and the lengthy installation times that often resulted in timeouts during the deployment process. The primary solution to these dependency challenges is to create a Docker image with all the required libraries pre-installed and run it as a separate pod in Kubernetes. This approach ensures that all necessary dependencies are included and compatible, eliminating the issues caused by dependency conflicts and installation times. The Docker image can be built once and reused, providing a consistent environment for running the function. However, due to project time constraints, we were unable to implement this solution.

7.2.5 System blocked by high-frequency requests

In the initial design to obtain BTC, gold and other price data, our plan is to mimic the reality of the existence of real-time trading systems, we will crawl the latest price data every minute, and according to the day's data will be, the average price, the highest price and the lowest price for separate processing

and storage, so that our analysis of the scenarios is more comprehensive and more in line with the reality of the financial market needs of the price analysis. For the three functions fetchgold, fetchaud, fetchbtc, in our earliest business logic, they will be executed once a minute, and the real-time data will be inserted into the elasticsearch. After deploying these three functions to the instance, we found that our entire system is thus After deploying these three functions to instances, we found that our entire system was therefore blocked, and the rest of the function calls, even the fisson package build, were difficult to complete. We realized that frequent cold starts would significantly increase the latency of function execution, which would cumulatively lead to system blocking. So we decided to simplify the problem in our business logic to obtaining data on a daily basis and executing functions in the early hours of the day to ensure that it does not cause blocking when the system is in high demand.

7.2.6 Package Failures During the Use of YAML Specifications to Deploy Functions

When deploying functions using YAML specifications in Fission, we encountered persistent issues where functions failed to run due to build-related package failures. After carefully studying the tutorial and following the discussion on ED Discussion, we found that the ZIP file of the function must be relative to the directory where the function is located; otherwise, the package build will fail. This requirement was not initially clear, leading to repeated deployment failures. To resolve this issue, we ensured that all ZIP files were correctly structured relative to their respective directories. This adjustment allowed successful builds and deployments, ensuring that our functions could run as intended.

8 Teamwork and Contribution

Member	Tasks
Hongyu Jin	Harvest data, Insert and Preprocess data, Fission function program and deploy, Create index
Shiyao Xue	Fission functions for data process and analysis program; Front-end analysis
Bin Liang	Insert data; Fission function program and deploy; K8s maintain and debug; Testing
Xuanhao Zhang	Fission function program and deploy; ElasticSearch Query Program
Yilin Chen	Create indexes; Queries and analyses of data using ElasticSearch; Fission function program and deploy; Some analysis processes

Table 1: Teamwork and Contribution Table

9 Conclusion

In conclusion, our project has effectively demonstrated the ability to analyze and correlate diverse data sets relevant to life in Australia, using a robust cloud cluster-based full-stack system. By leveraging tools such as Kubernetes, MRC, Fission, Elastic Search, Jupyter Notebook, and Kibana, we have built a system capable of processing and visualizing data in real-time.

Through the analysis of weather factors and traffic crashes in Tasmania, we have provided insights that can guide users in making safer travel decisions. Our examination of the relationship between weather and pollen counts across Australia offers valuable information for individuals with pollen allergies, helping them decide when to take protective measures.

The correlation analysis between the AUD/USD exchange rate and the prices of Bitcoin and gold offers real-time investment and risk management recommendations for investors. Additionally, our study of the AUD exchange rate in relation to income inequality provides an understanding of the economic factors that influence wealth distribution, potentially aiding in efforts to reduce inequality and poverty.

Throughout this project, we conducted extensive testing and system analysis to ensure the accuracy and reliability of our results. We discussed the pros and cons of the technologies used, such as the scalability and flexibility of Kubernetes and the real-time processing capabilities of Fission. Our system design was carefully planned to optimize performance and user experience.

We encountered several challenges during the development process, including data integration issues and performance bottlenecks. These were addressed by refining our data processing pipelines and optimizing our system architecture. Our iterative approach to problem-solving allowed us to continuously improve the system and overcome technical obstacles.

Overall, our project showcases the integration of geographic, economic, and weather data to provide meaningful insights and practical applications for users. By conducting thorough testing, discussing the advantages and disadvantages of various technologies, and overcoming development challenges, we have created a robust system that enhances users' ability to make informed decisions based on real-time and historical data analysis.

10 Appendix

Endpoint	URL	Method	Description
weatherinsert	`/weatherinsert/{index_name}`	POST	Inserts weather data. Accepts only weatherstats and weathertest as index names.
advanceeq	`/advanceeq/indexname/{index}/date/{date}/location/{location}`	GET	Retrieves data based on index, date, and location parameters.
esquery	`/esq/indexname/{index}`	GET	Queries data from the specified index.
weatherq	`/weather/year/{year}/month/{month}/day/{day}/location/{location}/weatherstat/{weather}/bymonth/{bymonth}/index/{index}`	GET	Queries weather data based on multiple parameters including date, location, and weather state.
fetchaud	`/fetchaud`	GET	Fetches dynamic AUD data from external sources and inserts into Elasticsearch.
fetchbtc	`/fetchbtc`	GET	Fetches dynamic BTC data from external sources and inserts into Elasticsearch.
fetchgold	`/fetchgold`	GET	Fetches dynamic gold data from external sources and inserts into Elasticsearch.
ftpcallstation	`/ftpcallstation/{index}/{daterange}/{insertindex}`	GET	Retrieves and processes data based on index, date range, and insert index parameters.
ftpfetchweather	`/ftpfetchweather/{state}/{station}/{date}`	GET	Fetches weather data from specific state, station, and date.

Continued on next page

Table 2 – continued from previous page

Endpoint	URL	Method	Description
qprice	/qprice/{index}	GET	Retrieves price data for specified index.
qstation	/qstation/{index}	GET	Retrieves station data for specified index.
ftpcallstationperday	/ftpcallstationperday	GET	Fetches dynamic data from FTP server daily.
canalysis	/canalysis/year/{year}/crash/{locationc}/weather/{locationw}	GET	Retrieves and analyzes crash and weather data based on year and location.
panalysis	/panalysis/year/{year}/pollen/{locationp}/weather/{locationw}	GET	Retrieves and analyzes pollen and weather data based on year and location.
ganalysis	/ganalysis/fill/{fill}/startdate/{start}/enddate/{end}	GET	Retrieves data within a date range with optional missing value fill.
ianalysis	/ianalysis	GET	Retrieves analyzed income data.