Air Quality Data Retrieval and Processing Report - Stage2

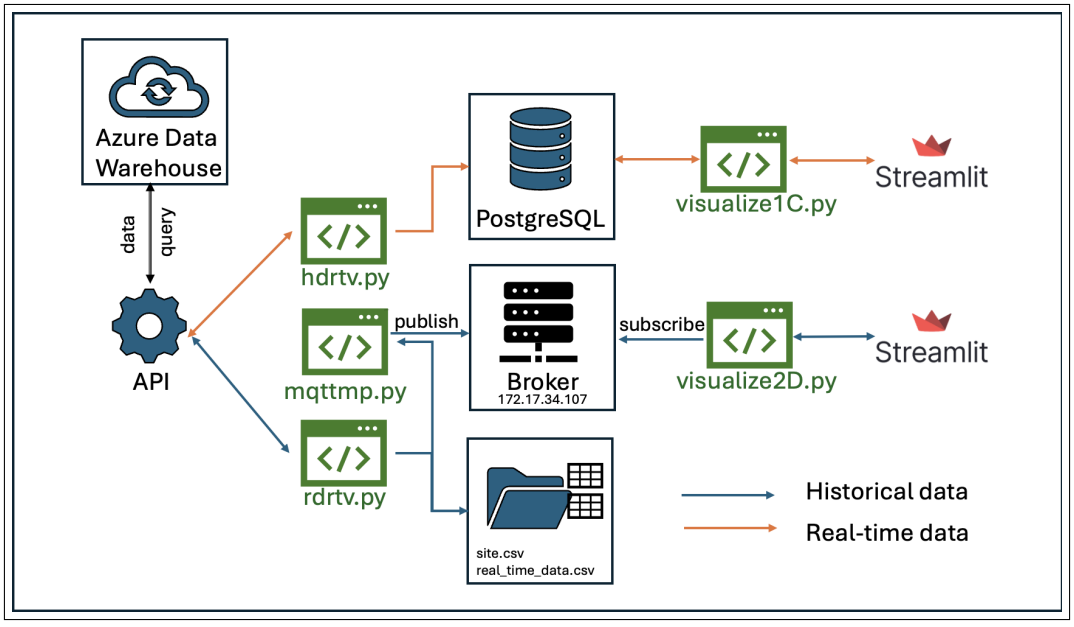
Shiwen Xu(520569045)

May 17, 2024

1 Introduction

This program aims to provide visualizations of PM2.5 observation data sourced from NSW Air Quality API [1]. It retrieves two types of data: historical data and real-time observation data. The historical air quality data comprises daily average PM2.5 data for all adjacent sites in Sydney from January 1, 2023, to the day before the current day. The real-time air quality data includes hourly average PM2.5 records for all sites, filtered from the retrieved current hourly data. Outlined in Figure 1, the program consists of five executable Python files: `hdrtv.py`, `mqttmp.py`, `rdrtv.py`, `visualize1C.py`, and `visualize2D.py`. Each program's detailed specifications will be explained in the following sections.

1.1 Program Pipeline



Figur 1: Program Pipeline

Figure 1 illustrates the overall program pipeline. Initially, the historical data is retrieved via API request and subsequently loaded into a PostgreSQL database. Following this, the data is read in its entirety from the database and displayed within a Streamlit app[2]. The historical data pipeline is

[1] https://www.airquality.nsw.gov.au/air-quality-data-services/air-quality-api
[2] https://streamlit.io/

denoted in orange color. Likewise, the real-time air quality data of current hour is obtained through an API request. Subsequently, the data is saved into the temporary CSV file. To simulate the dynamic flow of stream data visually, the process begins by reading data from the temporary CSV file. Subsequently, each record is published to a broker. Following this, the subscriber receives messages from the broker and visualizes each incoming data record within a Streamlit app.

1.2 Project Structure

The whole project structure, as shown below, includes 5 python programs with each design for specific tasks.

```
SID_AIR_QUALITY
├── data/
│   ├── real_time_data.csv (temporary file for real-time data)
│   └── site.csv (copied from A1)
├── hdrtv.py (historical data retriever - Task 1A & 1B)
├── rdrtv.py (real-time data retriever - Task 2A & 2B)
├── mqttmp.py (mqtt message publisher - Task 2C)
├── visualize1C.py (line chart with drop-down selection - Task1C)
├── visualize2D.py (streamlit-folium map with dynamic marker - Task2D)
└── requirements.txt (required packages)
```

2 Historical Data Retriever

The historical data retriever program, implemented in `hdrtv.py`, fulfills Task 1A and 1B. It is specifically designed to retrieve daily average PM2.5 data for all sites in Sydney from 2023-01-01, up to the day before current day. Afterwards, it loads the data into a database(a local PostgreSQL database in this case). The following will provide a detailed workflow by explaining each function in depth.

- `aqms_api_class`, copied from stage 1, is a class that defines and configures the API for querying data from the Azure DataWarehouse. Within this class, the `post_details` function is responsible for retrieving historical data for a specified time period or current real-time hourly observation data. If `ObsRequest` is not provided, the body command within the API request will be empty, therefore only the current hourly observed data will be retrieved by default. To retrieve historical data, one must first create an `ObsRequest` using the `ObsRequest_init` function. This involves passing predefined criteria to specify the desired historical data.

- `pgconnect, pgexec, pgquery`, also copied from stage 1, serve as helper functions for connecting to a PostgreSQL database, executing queries, and querying data, respectively. To use these functions properly, it is essential to modify the credential information within the `pgconnect` function to connect to either a local or remote database first.

- `retrive_histobs` completes the Task 1A. Intially, it defines the conditions for retrieving historical data and creates the `ObsRequest`, which is subsequently passed to the `post_details` function to retrieve historical data. Then, the retrieved historical data is converted into a pandas dataframe, preprocessed, and returned for further use.

- `write_db` completes the Task 1B. It encapsulates the process of creating and iteratively inserting records into the observation table for a specified database by calling the helper functions (i.e., `pgconnect, pgexec, pgquery`). Make sure `pgconnect` correctly connects to the desired database in order to successfully execute this function.

To execute Task 1A&1B, navigate to the parent folder `{SID}_AIR_QUALITY` first, then simply run the program using `python hdrtv.py` command.

3 Real-time Data Retriever

The real-time data retriever in `rdrtv.py` completes Task 2A&2B. It retrieves the latest data for the current date from all available sites in NSW and stores the retrieved records in a CSV file. As mentioned in the previous section, the `post_details` function can be utilized to retrieve current hourly observed data when `ObsRequest` is not provided. Therefore, to retrieve real-time data, the `aqms_api_class` is imported from the `hdrtv.py` file, and the `post_details` function is called without any additional arguments. Finally, the retrieved data is stored in a `real_time_data.csv` file within the `data/` folder.

4 MQTT Message Publisher

The **M**essage **Q**ueuing **T**elemetry **T**ransport(MQTT) message publisher in `mqttmp.py` completes Task 2C. It publishes each retrieved real-time hourly average PM2.5 record to the MQTT server, with a delay of 1 second between two consecutive messages. The function initially reads the data from the `real_time_data.csv` file located in the `data/` folder. Since the data includes all real-time observation data, including other categories such as CO, OZONE, etc., a new dataframe is created by filtering out only the hourly average PM2.5 records. With all the data prepared, the server is first connected, and each observation record is published to the broker under the `UniKey/comp5339-A2` topic. It's worth noting that the `retain` parameter is set to `False`, which means that if there is no active subscriber, the message will be discarded. Therefore, when simulating the stream data flow, it is ensured that the subscriber is active first, and then the publisher is allowed to send messages, which is often not the case in reality.

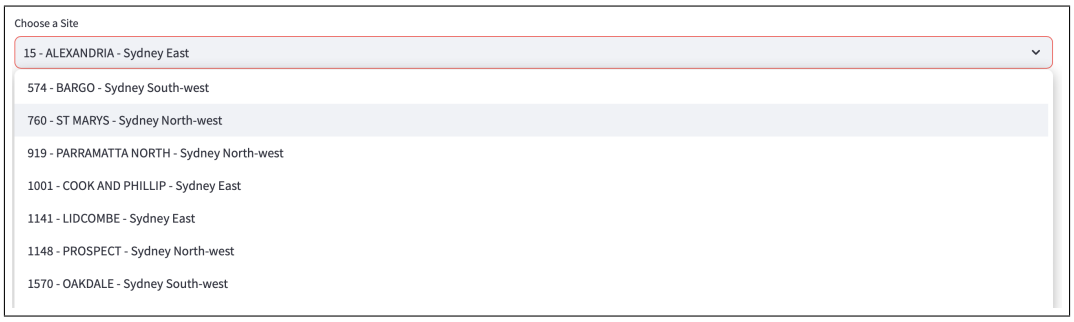
5 Dashboard Visualization

This section provides a detailed description of historical and real-time data visualization using the streamlit package.

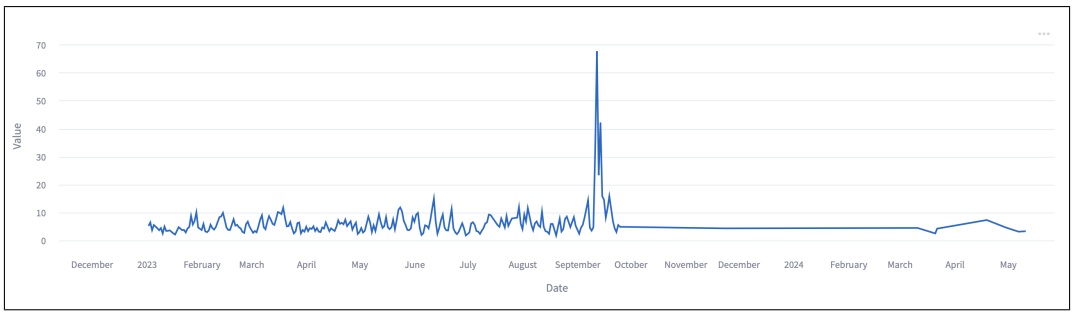
Historical Data

`visualize1C.py` visualizes the trend of the daily average PM2.5 data for a chosen Site from all Sydney sites using a drop-down list.

- `run_query` is a helper function to connect to the database and return the query results.
- `task1C` function retrieves data from the database and converts it into a pandas dataframe, denoted as `obs_df`. Since detailed site information is not stored in the observation table, the



Figur 2: Drop-Down List: Each choice is denoted by SiteID - SiteName - Region.



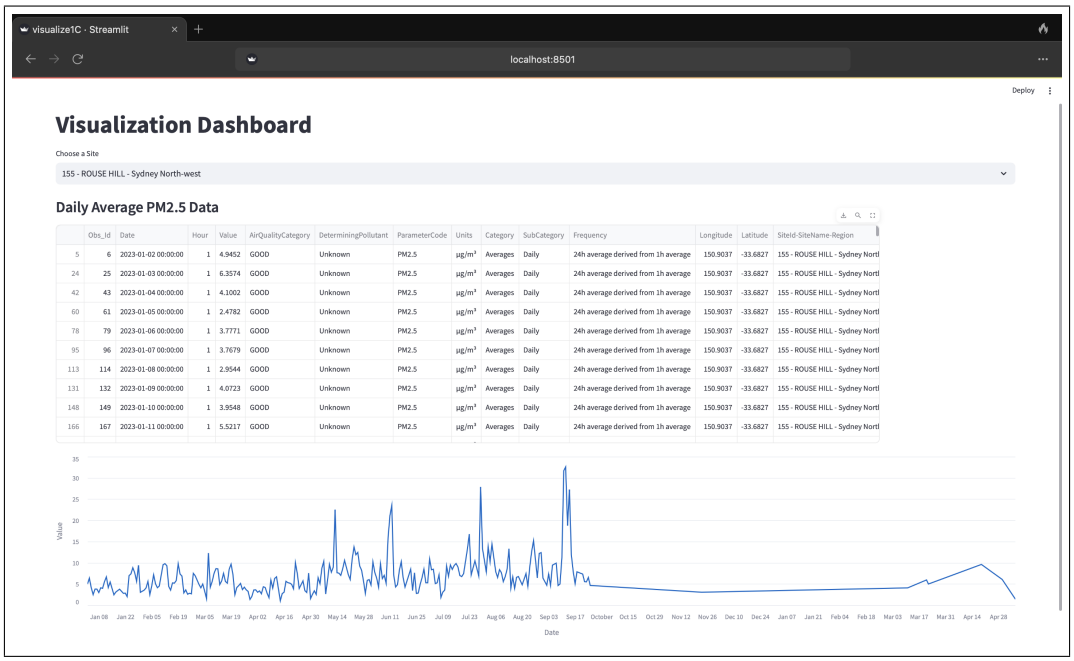
Figur 3: Example Daily Average PM2.5 Data for RANDWICK Site

`site.csv` file under the `data/` folder is utilized to create the `site_df` dataframe. Subsequently, these two dataframes are joined into a single dataframe based on the `Site_Id` attribute. The `streamlit.selectbox` function is employed to generate a drop-down list, where each site is represented by its `Site_Id`, `SiteName`, as well as the `Region` it belongs to. All relevant observation records are then filtered from the joined dataframe based on the selected site. Finally, the `streamlit.line_chart` is used to display the trend of the daily average PM2.5 value from 2023-01-01, up to the day before the current day.

Run the following command to display the trend of **daily average** PM2.5 value for a specific site up to the day before current day.

```
> cd SID_AIR_QUALITY
> python hdrtv.py # get the most up-to-date historical data and store into a
    database
> streamlit run visualize1C.py # display the trend using a line chart in
    streamlit app
```

Note that the hourly average PM2.5 data are not observed every day. Therefore, the earliest and latest dates shown in the table vary based on site and may not precisely match the input dates.



Figur 4: Visualization Dashboard for Historical Daily Average PM2.5 for Sydney Sites

Real-time Data

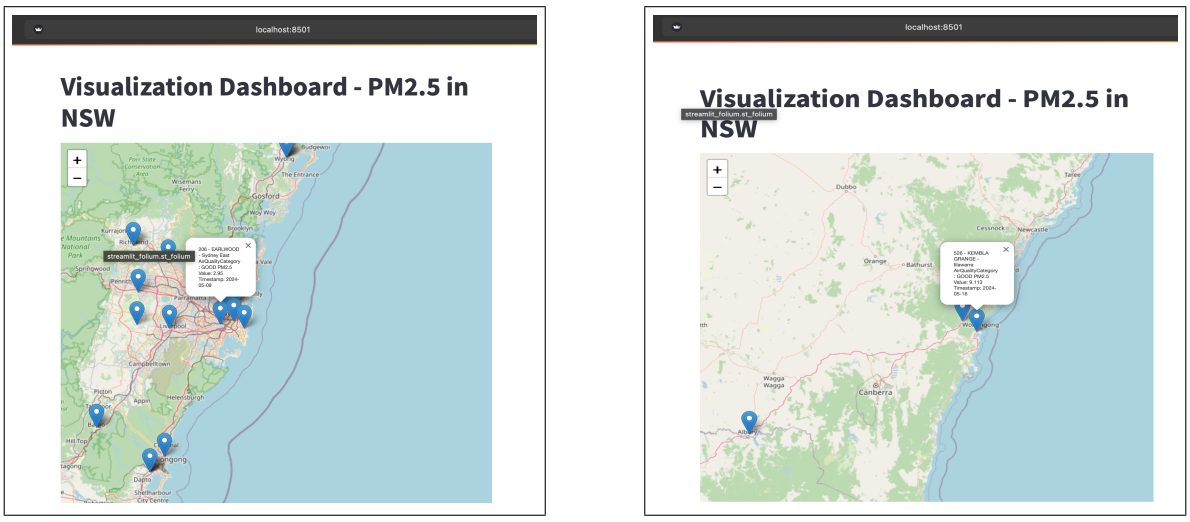
`visualize2D.py` provides the visualization of for the real-time PM2.5 value of current hour for all sites in a map format.

- `subscriber` assigns callback functions (`on_connect` and `on_message`) to manage events when connecting to the broker and when receiving messages from the broker, respectively. Subsequently, it establishes a connection to the MQTT broker located at the specified host (in this case, the instructor's own server) on port 1883.
- `on_connect` is a callback function that specifies actions to be taken when the MQTT client successfully connects to the broker. In this function, the client will print a message and subscribe to the broker under the `UniKey/comp5339` topic once the connection is established.
- `on_message` is a callback function designed to preprocess incoming messages and execute any necessary actions based on the newly received message from the MQTT broker. In this function, each received message is first printed into the console. Subsequently, the message is preprocessed by decoding, converting to a dictionary type, and then loaded into a dataframe, denoted as `obs_df`. Similar to task1C, where `site_df` is joined with the `obs_df` based on the `Site_Id` attribute to join all information into one row. The mean latitude and longitude are calculated from the `site_df` and used as the CENTER point to initialize the map. Subsequently, the CENTER along with the combined dataframe are passed to the `draw_map` function to dynamically generate a marker on the map using the site geolocation information.
- `draw_map` dynamically visualizes each incoming message (observation data) by extracting the geo-location, AirQualityCategory, PM2.5 value, and the timestamp of the record. Specifically, the `streamlit_folium` package is utilized to create Folium objects and plot them within the Streamlit app. Each time a new record arrives, the geo-location (latitude and longitude) data are

used to create a marker on the map. When the user clicks on a marker, a popup widget displays detailed information, including the AirQualityCategory, PM2.5 value, and timestamp, for a specific site. To prevent the map from being rerendered every time a new message arrives, the `FeatureGroup` arguments are passed to the `st_folium` function, allowing new markers to be drawn on the same map.

Run the following command to dynamically display the current hourly average PM2.5 value for all sites. As mentioned in Section 4, to ensure no messages are discarded, we will activate the subscriber first.

```
> cd SID_AIR_QUALITY
> python rdrtv.py # get the ream-time observation data of current hour
> streamlit run visualize2D.py # subscribe from the broker and visualize the
    folium-map
> python mqttmp.py # open another terminal to publish the data to the broker
```



(a) Example on Date 05-09

(b) Example on Date 05-16

Figur 5: Real-Time Hourly Average PM2.5 Data Visualization