COMP5339 - 2023 Semester 2

THE UNIVERSITY OF SYDNEY

Data Engineering

Assignment Stage 2

Course :	Data Engineering
Name:	Yanjie Wu
Student ID:	520076163
Pages:	9
Date :	2023/11/05

Table of Content

1. Introduction and Dataset Description 1
    1.1. Introduction and data source 1
    1.2. Data cleaning and pre-processing 1
2. Database Description 2
3. Data exploration description 2
    3.1. What was done and why 2
    3.2. What was found 3
4. Data visualization description 3
    4.1. Bar chart of average prices for different fuel types 3
    4.2. Maps of service stations 4
    4.3. Graphs of time trends in prices for different fuel types 5
5. Description of data pipeline and final deliverable 5
    5.1. main.py 6
    5.2. DataGathering.py 6
    5.3. DataCleaning.py 6
    5.4. DataIngesting.py 7
    5.5. DataAnalysis.py 7
    5.6. Final Dashboard 8

1. Introduction and Dataset Description

1.1. Introduction and data source

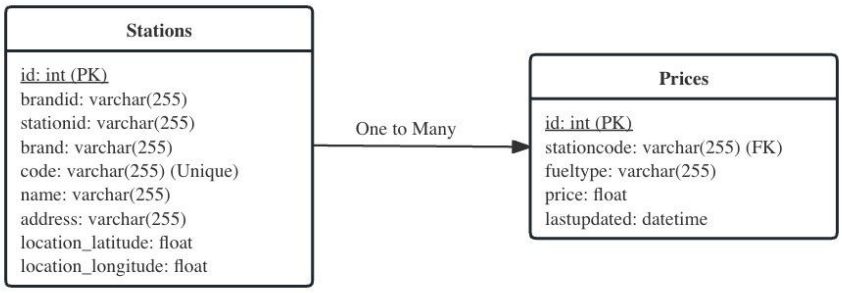
The report documents a fuel price monitoring system that utilizes a data pipeline for real-time data visualization. The system platform utilizes the real-time data API provided by New South Wales Government Open Data Platform (data.nsw.gov.au) to gather source raw data. Since the remote MQTT agent could not be accessed successfully, the system simulates publishing raw data by using the local MQTT agent.

1.2. Data cleaning and pre-processing

The data preprocessing and cleaning of the system includes the following steps:

- **Data standardization:** In order to ensure consistency and comparability of the data, the data was first standardized. Using Python's pandas library, the data was transformed into a standardized data frame, ensuring that all data had a uniform format and structure.
- **Date Time Format Conversion:** The lastupdated column in the data was originally formatted as a string, which was converted to a date time format for ease of subsequent manipulation. This not only ensures date and time accuracy, but also makes time series analysis possible.
- **Data Filtering and Sorting:** In order to focus only on the most recent data, the system determines the most recent date in prices and calculates the start date of the previous month based on this date. Then, only data from this time period is selected. In addition, the data is sorted in descending order by the lastupdated column, ensuring that the most recent data is always at the top.
- **Keyword Checking:** To ensure the integrity of the data, the system checks for the presence of the expected keywords in each record. This ensures that each piece of data is complete and there are no missing fields.
- **Null value check:** Null or missing values are often found in large datasets and they can lead to biased analysis results. Therefore, the system checks each field of each record to ensure that there are no null or missing values.
- **Data type checking and formatting:** Inconsistency in data types can lead to analysis errors. To ensure data consistency, the system checks the data type of each record and converts it if necessary. In addition, the data is formatted to ensure that they meet specific criteria.
- **Subscribing to data from the MQTT agent:** The system subscribes to raw data from the MQTT agent using the 'raw_prices_subscribe' and 'raw_station_subscribe' functions. This data is cleansed before it is added to the system.
- **Publishing cleaned data:** The cleaned data is added to the 'clean_price' and 'clean_station' lists. The system then publishes this data to a new MQTT topic using the 'publish_clean_prices' and 'publish_clean_station' functions, ensuring that only cleaned and validated data is accessible to other systems or users.

2. Database Description



During the data integration process, the existence of the two main stations and prices tables in the database is first ensured by the 'create_tables' function, which is implemented with persistence and referential integrity in mind. This function is implemented with database persistence and referential integrity in mind, ensuring that each stationcode in the prices table corresponds to a valid code in the stations table, and then the data is integrated into these two tables through the 'store_in_db' function. This function first normalizes the stations and prices data from the input data dictionary into Pandas data frames, and then calls the 'save_df_to_sql' function to store the two data frames into the 'FuelAnalysis.db' database.

In terms of the indexes created, an auto-growing index named 'id' was added to both tables as their primary key. This indexing strategy ensures that each record in the tables has a unique identifier. In large databases, primary key indexes can dramatically improve the speed of data retrieval and ensure data integrity. In addition, stationcode in the prices table references code in the stations table as a foreign key, creating a one-to-many relationship between the two tables. This foreign key strategy ensures that each stationcode in the prices table has a corresponding code in the stations table.

3. Data exploration description

In the context of responding to a fleet manager's request for a systematic study of fuel prices, we first explored the data we had to understand its structure and potential value.

3.1. What was done and why

Real-time Data Acquisition: To accurately capture the fuel prices in New South Wales (NSW) contemporaneously, we established a subscription to pertinent topics through the MQTT protocol. Through this, we obtained intricate details pertaining to gas stations and contemporaneous fuel pricing data. Such immediate data procurement offered a snapshot of the prevailing fuel pricing dynamics, facilitating a prompt response to any fluctuations in prices.

Analysis of Data Structure: An exhaustive exploration of the data structure was conducted, affirming its comprehensive nature, encompassing specific details about gas stations—encompassing brand, geographical positioning, and coordinate data—as well as contemporaneous oil pricing data, which detailed the oil variety, its price, and its most recent update timestamp. A profound understanding of

this data architecture enabled the formulation of a tailored analytical approach, thereby equipping fleet managers with the requisite insights.

Identification of Temporal Trends: With the possession of time-stamped data elucidating price modifications, endeavors were made to discern temporal trends and price cyclicities. Such an analytical step is paramount in forecasting impending price trajectories and providing fleet entities with informed recommendations on optimal refueling timings.

3.2. What was found

Heterogeneity in Pricing Based on Fuel Type: An examination of the data underscored variances in pricing contingent upon the type of fuel. Such discrepancies present an avenue for fleet operations to fine-tune their expenditure on fuel through judicious selection of fuel variants.

Geospatial Pricing Disparities: Preliminary delving into the data manifested pronounced geographical disparities in fuel prices within NSW. This delineation suggests the existence of locales with more favorable fuel pricing, thereby guiding fleet operations on strategic refueling locations to maximize cost efficiency.

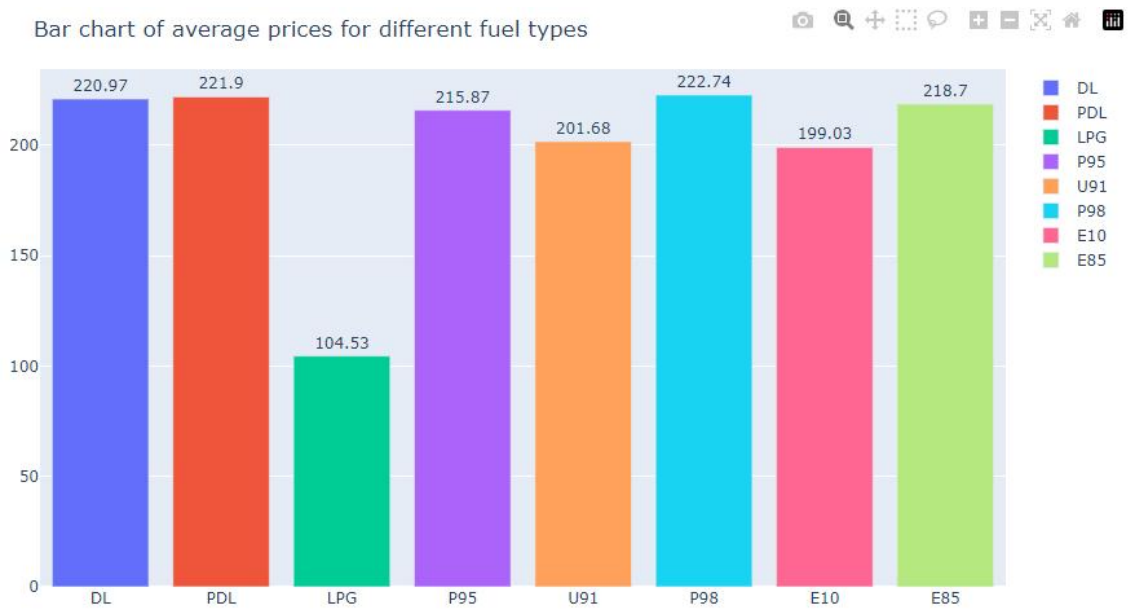
Time-Dependent Price Fluctuations: Our analysis discerned discernible temporal patterns and cyclicities in gasoline pricing, potentially tethered to specific days or calendar months. Such insights equip fleet managers with predictive intelligence on temporal windows offering the most advantageous fuel pricing.

4. Data visualization description

In order to be able to achieve the requirements of the final system, all the visualization methods chosen were able to fit the real-time data, including bar chart of average prices for different fuel types, maps of service stations, and graphs of time trends in prices for different fuel types.

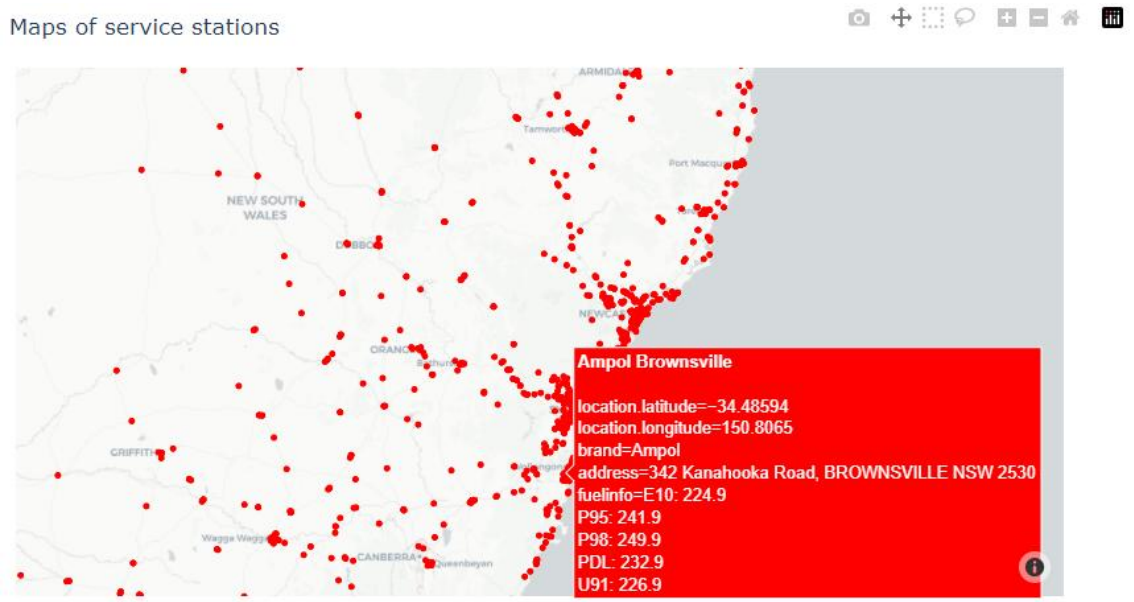
4.1. Bar chart of average prices for different fuel types

The bar chart effectively shows the average price of different fuel types. Each bar represents a fuel type and the height indicates the price. This visualization was chosen because a bar chart can effectively compare individual categorical data points. In this case, it provides a clear visual representation of the average price for each fuel type. This bar graph visualizes the average price of each fuel type, which helps fleet managers quickly determine which fuel is the most economical at a given time. Also based on the graph, fleet managers may want to consider replacing or adding vehicles that run on a certain fuel to reduce overall fuel costs.



4.2. Maps of service stations

The geographic map enables the location of the individual service stations to be marked. Each red dot indicates the location of a service station. A pop-up window allows the display of specific information about the service station, including the type of fuel sold and the latest prices. Maps are great for displaying geographic data, especially when you want to show the distribution or location of certain entities so that you can more clearly visualize the distribution of different service stations. The map makes it easy for fleet managers to find service stations throughout the state and their up-to-date prices. Pop-up windows display specific service station information, including their fuel prices. This means fleet managers can quickly find stations and prices for any given location and dispatch vehicles for fueling accordingly.

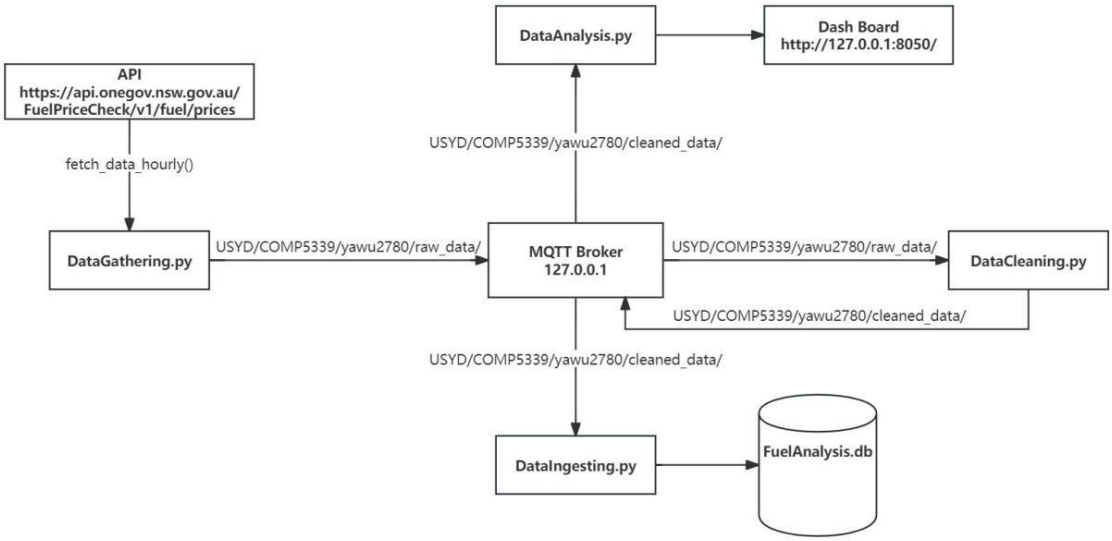


4.3. Graphs of time trends in prices for different fuel types

The line graph shows price trends for different fuel types. Each line represents a fuel type, the height indicates the price, and the horizontal coordinate represents time. It effectively compares changes in the time dimension for each fuel type, allowing easy comparison of price changes for multiple fuel types to provide a clearer visual representation. By tracking price trends by watching fuel prices rise and fall, fleet managers can determine when the best time to refuel is, and thus save money. And by comparing different fuel types, fleet managers can quickly see which fuels have had the most stable or lowest price increases over the past month.

5. Description of data pipeline and final deliverable

The final real-time data monitoring system contains five files, namely main.py, DataGathering.py, DataIngesting.py, DataCleaning.py and DataAnalysis.py. Data transfer and sharing is realized through MQTT to achieve real-time display. The code uses a local broker, so the user needs to configure the MQTT service locally to realize the pipeline, or the user can modify the broker to realize publishing at other addresses. All data is delivered in JSON format, including both prices data and stations data, which will be published and subscribed to through different topics to ensure data isolation. The prices data contains the components stationcode, fueltype, price and lastupdated. The stations data contains the components brandid, stationid, brand, code, name, address, location.latitude and location.longitude. This section will document the purpose and role of each program. Below is a flow chart of the pipeline to help users better understand how the entire system works.



5.1. main.py

main.py is the entry point to the entire system. The file provides access to the entire system by calling packages from various other files.

- main: The main function is used to start various threads to realize the data publish and subscribe in order to ensure the normal start of the dash board app.
- fetch_data_hourly: This function is designed to call the functions in DataGathering to pull API data. Through While loop, this function realizes that every 1 hour will be pulled automatically, and the data will be updated through the data lock. By calling the function in DataCleaning to filter the latest month's data.
- publish_price_data: This function is designed to publish raw price data, using data locks and comparing lastupdated to ensure that all published data is new and unpublished.
- publish_price_data: This function is designed to publish raw service station data, with data locks and a code comparison to ensure that all published data is new and unpublished.

5.2. DataGathering.py

The program mainly implements writing a function to implement getting data from the API. The data retrieved is a dictionary containing two elements, namely stations and prices.

- get_data_from_api: The function first constructs a request to access the token URL to obtain an access token. The obtained token enables the construction of the authorization header and sends a request to the price URL to get the fuel price data. Finally, the function returns the data obtained from the API.

5.3. DataCleaning.py

The main purpose of this program is to enable the subscription of raw data and to clean it to ensure that the final data is as expected. The final cleaned data will be published to another topic via MQTT.

- clean_for_recent_data: This function is used in the raw data acquisition session. This function ensures that the latest month's data is acquired. Finally, the data is sorted by the 'lastupdated' column and the cleaned data is returned.clean_price_data/clean_station_data: These functions focus on cleansing the raw data of the subscription to ensure that they match the expected data structure. This mainly includes dealing with the consistency of the data, dealing with missing values, and validating the data structure of the different components and converting them to the desired type.
- on_price_connect/on_station_connect: These functions are used to connect the topic of the raw data.
- on_price_message/on_station_message: These functions are used to receive the raw data and call the data cleaning functions to realize the cleaning of the raw data.

- raw_prices_subscribe/raw_station_subscribe: These functions call previously constructed functions to enable pulling up the MQTT service, thus ensuring that the raw data is received and cleaned.
- publish_clean_prices/publish_clean_station: These functions ensure that the raw data is received and cleaned by pulling up new threads, while posting the cleaned data to another new topic.

5.4. DataIngesting.py

This program focuses on ingesting the cleaned data and establishing a SQLite database by subscribing to the related topic.

- create_table: This function defines the creation of the database and tables. It also ensures that each pull up of the service will reset all tables in the database to avoid duplicate data. The SQL statement ensures that the stored data conforms to the design of the database scheme. Since the data are stored one by one, the reference relationship between tables is ignored, but the real relationship should be the same as previously described.
- save_to_sql: The function mainly ingests single data into the database, through setting 'if_exists' as 'append', it can ensure that all data can be saved. In addition to the data, it also need to input the name of the database and table names to ensure that the data can be categorized into different tables.
- store_in_db: This function will receive all the subscribed data and determine which table the data belongs to. All the data is stored by calling the function that stores single data one by one.
- process_price_msg_to_db/ process_station_msg_to_db: These functions process the subscribed data and passes it to the 'store_in_db' function to ingest all the data.
- mqtt_price_subscribe/mqtt_station_subscribe: These functions will subscribe to the cleaned data topics to receive the data and pass it to the data processing functions.
- start_ingest_data_to_db: This function is an entry function of the whole file. The main function can call this function to ingest the cleaned data. It first calls the 'create_table' function to ensure that the database and data tables exist and reset. It then pulls up different threads to achieve continuous synchronized subscription of data.

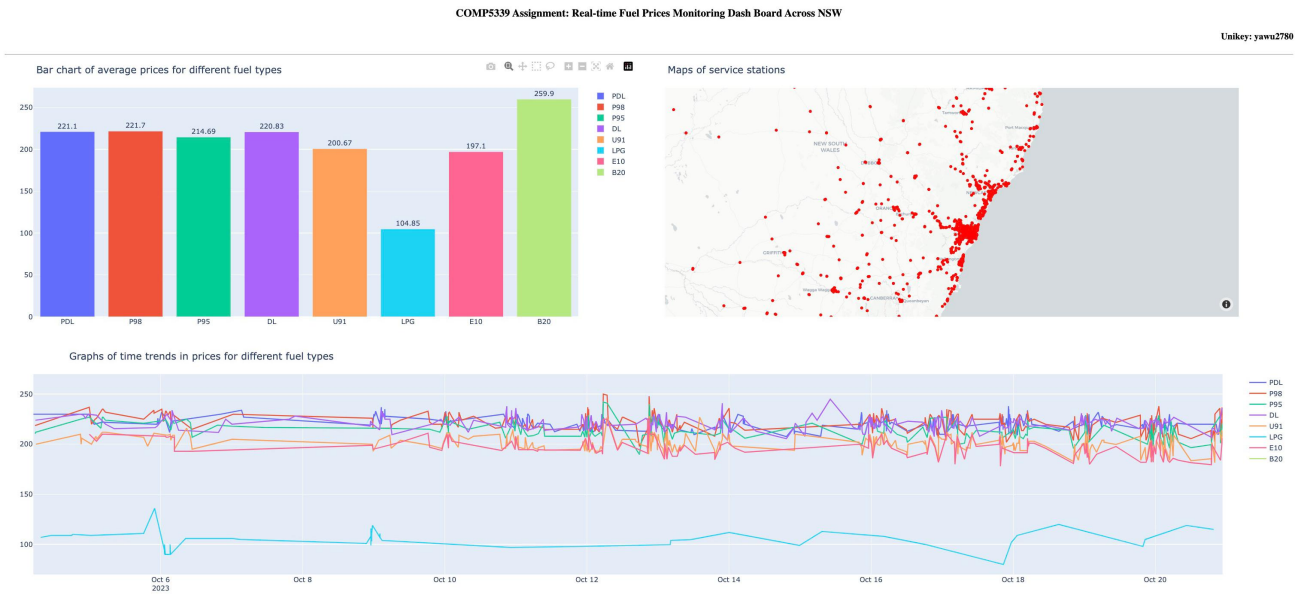
5.5. DataAnalysis.py

The program uses plotly dash to integrate the figure from stage 1 while subscribing the cleaned topic for dynamic presentation and visualization of the real-time data.

- process_price_msg/ process_station_msg: These functions process the subscribed data and stored them into global list.
- mqtt_price_subscribe/mqtt_station_subscribe: These functions will subscribe to the cleaned data topics to receive the data and pass it to the data processing functions.
- app = dash.Dash(__name__): An app service is defined and all subsequent layouts and presentations will be based on that app.

- app.layout: The app layout is defined, including the title of the app, the position and style of each chart, and the time of the automatic update. Dynamic display of the page is achieved by setting the automatic update time.
- update_bar_graph: This callback function is responsible for updating a bar chart showing the average price of different fuel types. The function first checks if the price_list is empty, and if it is, no update is performed. Then, the function builds a DataFrame with all the price information. Pandas' groupby method is used to group the prices by fuel type and calculate the average for each group. Next, a plotly.express.bar chart is created showing the average price for each fuel type. Finally, return the chart object, which Dash will automatically use to update the bar chart on the front end.
- update_map_graph: This callback function updates a map showing the location of a gas station and some related information. The function first checks the station_list, and if it is empty, the map is not updated. If there is data in the list, the function creates a DataFrame containing information about all the gas stations. plotly.express.scatter_mapbox is used to create a scatter map where each point represents the location of a gas station. The exact locations of the gas stations are determined by their latitude and longitude, and the name, brand, address, and fuel information of the gas station is displayed when the mouse is hovered over. The style and zoom level of the map are configured. Finally, the returned map object will be used to update the front-end map display.
- update_line_graph: This callback function is responsible for updating a line chart showing the price trends of different fuel types. The function first processes the data in price_list, organizing the prices by fuel type and timestamp. For each fuel type, the function creates a time-series price trendline. Use plotly.graph_objs.Scatter to create a time series line for each fuel type. All time series lines (traces) are then added to the plotly.graph_objs.Figure object. The layout of the chart is updated and this object is returned to update the front-end line chart display.

5.6. Final Dashboard



The final dashboard is shown above. While the image only shows a static effect, it is actually an interactive data visualization tool that provides real-time monitoring of fuel prices and station distribution in New South Wales. It features the main title highlighted in large font at the top of the page and the creator's information in the upper right corner, below which are three core components: a real-time updated bar chart, displayed on the left, showing the average price of different fuel types; immediately to the right of this is an interactive map detailing the exact location of fuel stations with red marker points and displaying more information on mouse hover; and finally, at the page's Finally, at the bottom of the page is a full-width line graph depicting the trend of fuel prices over time. The entire layout utilizes clear visual separation and balanced spatial distribution, designed to provide users with an intuitive understanding of the data, allowing them to gain insight into market dynamics through real-time data points on the charts. Each chart is updated via a timer that is triggered every second, ensuring that users always have access to the most up-to-date information, which not only improves the user experience, but also reinforces the immediacy and accuracy of the data.