## 

Table of Contents

[1. Value Based Problem Statement 3](#_Toc157977423)

[2. Preparation 4](#_Toc157977424)

[Bus Timing API 4](#_Toc157977425)

[Bus Stop and Bus Route API 5](#_Toc157977426)

[3. API Call 7](#_Toc157977427)

[Download API Data 7](#_Toc157977428)

[JSON Parse 8](#_Toc157977429)

[Filtering out Unnecessary Data 8](#_Toc157977430)

[4. Data Display 9](#_Toc157977431)

[Bus Timing API 9](#_Toc157977432)

[Bus Stop API 14](#_Toc157977433)

[Bus Route API 16](#_Toc157977434)

[5.Enhancements 20](#_Toc157977435)

[Collecting Output Data for External Use 20](#_Toc157977436)

[Bus Stop Timing Website for Front End Display 22](#_Toc157977437)

[PowerBi Visuals for Back End Analysis 26](#_Toc157977438)

[6. Summary 28](#_Toc157977439)

[7. Reflection 29](#_Toc157977440)

[Appendix 30](#_Toc157977441)

[Notice on exporting Alteryx Workflow 30](#_Toc157977442)

[Website HTML code 32](#_Toc157977443)

## 

## **1. Value Based Problem Statement**

**Problem Statement:**

In the context of our growing reliance on public transportation, especially buses, the unavailability of real-time bus arrival data significantly detracts from the commuting experience by causing longer wait times and increasing the risk of missed connections. This issue not only causes inconvenience to commuters but also challenges bus operators, who are unable to gain detailed insights into passenger flows and bus stop usage. Consequently, bus companies are hindered in their ability to adjust their routes, schedules, and fleet management to align with actual service needs due to inefficient data utilization for generating actionable insights.

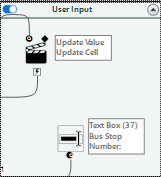
**Value Proposition:**

To address these challenges, our project envisions the creation of a sophisticated real-time bus information system designed to cater to the needs of both commuters and bus service providers. For commuters, our system promises to deliver timely and accurate bus arrival information based on the user's specific bus stop selections. This innovation aims to streamline travel planning, minimize unnecessary waiting, and enhance the overall commuting experience. On the flip side, for bus companies, the proposed system offers a robust framework for collecting and analyzing data related to bus stop activity, passenger numbers, and service demand patterns. Such insights are invaluable for improving operational efficiencies, fine-tuning route allocations, and elevating service quality. By serving the dual purposes of facilitating smoother commutes for individuals and empowering bus companies with data-driven operational insights, our project stands to revolutionize the public transportation landscape, making it more efficient, responsive, and aligned with the needs of its users.

## **2. Preparation**

Data preparation is a critical component that determines the efficacy and accuracy of the information delivered to end users. In this crucial phase, I engaged with the various functionalities of Alteryx to set the stage for an efficient API call process.

### **Bus Timing API**

A screenshot of a computer

Description automatically generated

Figure 1: User Input for Bus Stop Number and Data Preparation for Bus Timing API Workflow

This is the main API used to provide us with information of bus arrival timings and also other features of the incoming buses such as the load, type of bus and whether it is wheelchair friendly.

1. **User Input**

The initial step involved capturing user input for the 'Bus Stop Number.' Once this value was fed into the 'BusStopCode' column within Alteryx, I focused on ensuring that it could seamlessly integrate with the API URL. Given that the API requires a string query parameter, I converted the 'BusStopCode' to a string data type. This conversion was essential for subsequent concatenation with the base API URL.

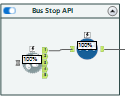
1. **Formula-Based URL Construction**

With the 'BusStopCode' now in the correct format, I constructed a formula within Alteryx to append it to the API URL. The formula took the base API URL stored in the 'API URL' column and concatenated it with the query parameter 'BusStopCode=' followed by the actual bus stop code from the 'BusStopCode' column. This operation generated a complete and customised endpoint for each API request.

1. **API Call Configuration**

Furthermore, additional fields such as 'AccountKey' were configured to adhere to the API's requirements. These fields were critical for authentication and for simulating a web browser request, ensuring that the API would process our requests as intended.

### **Bus Stop and Bus Route API**

A computer screen shot of a bus route

Description automatically generated

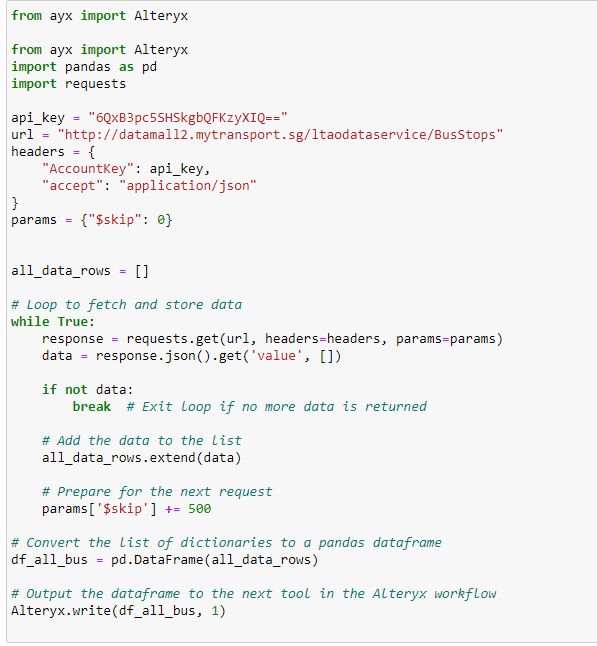
Figure 2: Bus Stop and Bus Route API to Data Frame Workflow

To enrich the primary data obtained from the Bus Time API, the project integrated supplementary data from two additional APIs: the Bus Stop API and the Bus Route API. These APIs are critical in providing a broader spectrum of information, including operational timings, bus stop names, bus route descriptions, and precise geolocations. For front-end users, this combination of data not only enhances the granularity of the information available but also allows for a more interactive and informative experience, enabling them to access a wealth of details like the exact location of bus stops and specific route paths their buses will take. For back end users, the combination of these data points offered a multi-dimensional view of bus operations, crucial for a thorough analysis.

However, when interfacing with these APIs through Alteryx, a significant limitation was encountered: the data retrieval was capped at 500 rows per download. Such a restriction posed a substantial risk of omitting relevant data, including possibly the user's inputted bus stop, particularly when dealing with the voluminous datasets these APIs provide. The chance that a user's selected bus stop would not be included in the truncated data set was high, potentially leading to a gap in the service provided to end users.

To address this limitation and to ensure that the front-end user experience remained unaffected by the backend data processing constraints, a Python script was integrated into the Alteryx workflow. This script, leveraging the 'requests' library, made iterative API calls, meticulously collecting all available data beyond the initial 500-row limit. It managed pagination by incrementally adjusting the 'skip' parameter, thus fetching the complete dataset across multiple requests.

Once the full set of data was compiled and structured, it was fed back into the Alteryx workflow, effectively bridging the gap between Python's data handling capabilities and Alteryx's analytical prowess.



## **3. API Call**

In this part of my report, I'll walk through the steps I took to get the bus timing data from the Singapore Land Transport Authority's API using Alteryx. I'll explain how I carefully set up the download, made sure to pick only the data we need, and sorted it into a clear format. I'll also discuss how I filtered out any bits we didn't need to keep things simple and clear. This section is all about turning a bunch of complex data from the web into something useful that tells us when the next bus is coming.

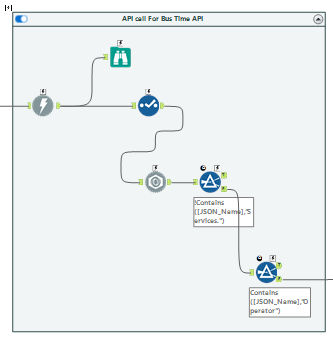


Figure 3: Bus Timing API API Call Workflow

### 

### Download API Data

After setting up the API call by dynamically constructing the request URL using the Select and Formula tools, I proceeded to the execution phase where I would download the data from the API. To accomplish this, I used the Download tool in Alteryx, which is designed to retrieve data from the web. I configured the tool, ensuring the proper headers were in place, particularly the 'AccountKey', which is crucial for API access authorization.

I added the 'AccountKey' to the headers to authenticate my requests. By doing so, I assured that the data retrieval was compliant with the API's requirements and that my access was legitimate.

The workflow was then set to execute the API call. Upon triggering the process, Alteryx initiated a GET request to the API endpoint I had constructed. The data returned from the API was expected to be in JSON format. In summary, the Download tool acted as the bridge between my Alteryx model and the live data from the transportation API.

### **JSON Parse**

After initiating the API call, the next step was to handle the response which came in a JSON format. JSON, short for JavaScript Object Notation, is a widely used format for transmitting data objects consisting of attribute-value pairs.

After applying the JSON Parse tool in Alteryx to the DownloadData I obtained from the API, I was presented with an organised table distinctly separating JSON Name and JSON Value String columns.

The JSON Name column enumerated the various attributes available in the data, such as 'Services.0.ServiceNo' for the bus service number and 'Services.0.NextBus.EstimatedArrival' for the next bus's estimated arrival time. Each attribute corresponded to a specific piece of information about the bus services, structured hierarchically as they were in the original JSON response. The JSON Value String column displayed the actual data associated with each attribute name, such as the service number "10" for 'Services.0.ServiceNo' or a timestamp for 'Services.0.NextBus.EstimatedArrival'. This format was highly beneficial as it provided a direct and comprehensible way to view and interpret the data.

### **Filtering out Unnecessary Data**

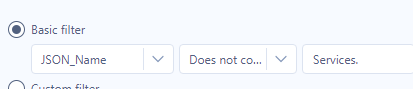


Figure 4: Filter Condition for Services

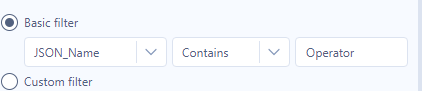


Figure 5: Filter Condition for Operator

In the filtering stage of my Alteryx workflow, I set specific conditions to narrow down the dataset to only include the most relevant and necessary information for my project. The goal was to isolate the data entries that pertained to the bus services, which are identifiable by the presence of 'Services' in their JSON Name.

The Filter tool was configured to exclude any records where the JSON Name did not contain 'Services.' This condition was pivotal because any JSON element without 'Services' was not relevant to the bus arrival information I needed. By applying this filter, I ensured that only records with essential bus service details, such as service number, estimated arrival, and location coordinates, were retained for further processing. I then used another filter to filter out JSON elements with “Operator” as I am not using the 'Operator' information at this stage.

## **4. Data Display**

### **Bus Timing API**

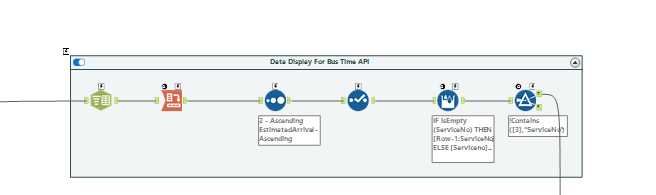
****

Figure 6: Bus Timing API Data Display Workflow

#### **Text to column**

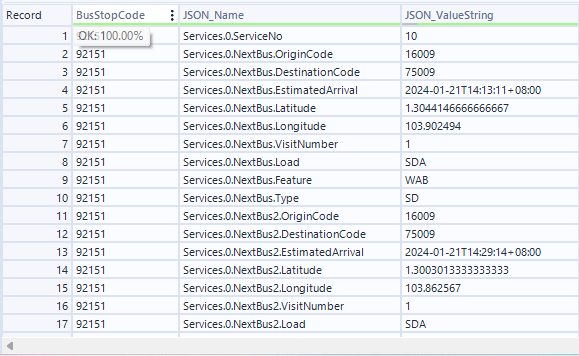


Figure 7: Bus Timing API before Text to Column

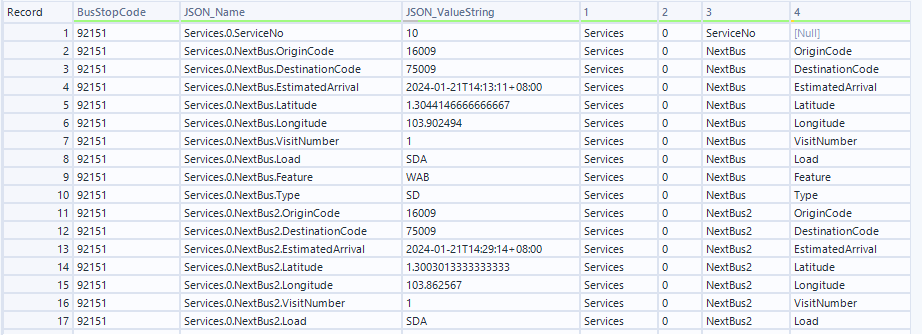


Figure 8: Bus Timing API after Text to Column

After extracting the required data from the Bus Timing API, the information is initially presented in a JSON format, with each data point comprising a 'JSON\_Name' and 'JSON\_ValueString'. To make this data more user-friendly, a 'Text to Column' operation is performed.

In this step, the JSON data is split into multiple columns based on a delimiter, which, in this case, is the period character that separates the hierarchical levels in the 'JSON\_Name'. This delimiter is identified in the JSON structure, such as in 'Services.0.ServiceNo' or 'Services.0.NextBus.EstimatedArrival', and is used to divide the data into separate columns. Each level of the JSON hierarchy becomes a new column in the table, effectively flattening the structure and making it easier to read and understand.

For example, 'Services.0.ServiceNo' becomes split into separate columns: 'Services', '0', 'ServiceNo', and '[Null]' where each part of the original 'JSON\_Name' string now occupies its own column in the dataset. The '[Null]' indicates that there's no further subdivision for that data point.

#### **Cross Tab**

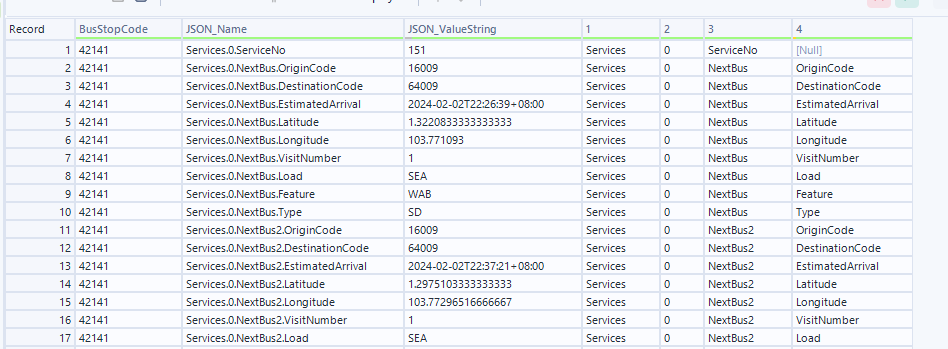


Figure 9: Bus Timing API before Cross Tab

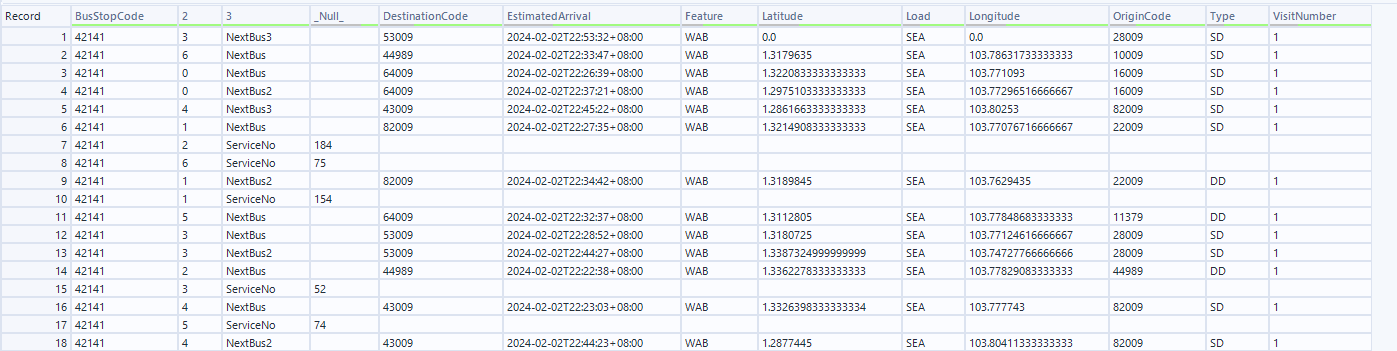


Figure 10: Bus Timing API after Cross Tab

The cross-tab process begins by grouping the data based on certain values. In this case, the ‘2’ acts as the primary grouping variable, ensuring that all data related to a specific bus stop is consolidated. The JSON data, which is initially presented with 'JSON\_Name' and 'JSON\_ValueString' fields, undergoes a transformation where the 'JSON\_Name' hierarchy is split into separate columns. This split is based on the delimiter identified in the 'JSON\_Name' string, effectively expanding the nested JSON structure into a flat, wide format. Each level of the hierarchy, which includes information such as the service number, the next bus arrival times, and geographic coordinates, is allocated its own column.

In the cross tab tool's configuration, the column headers are changed to reflect the new structure of the data. The 'Values for New Columns' option is set to 'JSON\_ValueString' which indicates that the data contained within this field will populate the newly created column.Similarly, 'Services.0.NextBus.EstimatedArrival' is divided in the same manner, with 'EstimatedArrival' now occupying its own column, distinct from the 'NextBus' category.

#### **Sorting Data to Group Information for the Same Bus Together**

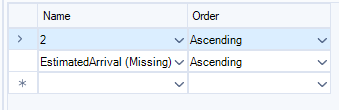
****

Figure 11: Sorting Conditions

To enhance the clarity of the bus data table, I used Alteryx's sort tool to group buses by service number and sort by estimated arrival times. Buses with the same service number are now listed together, and within each group, the buses are ordered so that the one with the earliest arrival time appears first.

#### 

#### **Addressing Missing Service Numbers**

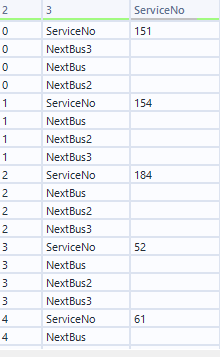


Figure 12: Bus API Before Multi Row Formula

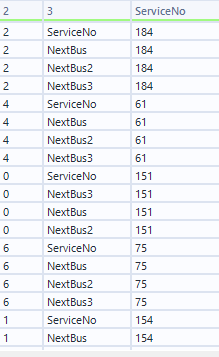


Figure 13: : Bus API After Multi Row Formula

To address missing 'ServiceNo' data within the dataset, I implemented Alteryx's Multi-Row Formula tool to automate the imputation of these values based on information from other rows with the same service number. The tool's functionality allows it to reference data from other rows within the same column, which is particularly useful when needed to fill in gaps based on adjacent row values.

The expression used in the tool is as follows:

***IF IsEmpty([ServiceNo]) THEN [Row-1:ServiceNo] ELSE [ServiceNo] ENDIF***

*This formula checks the value of the row above the missing cell and imputes the missing cell with that value.*

After applying the Multi-Row Formula, the dataset has a continuous set of 'ServiceNo' values without gaps, ensuring that each record is accurately associated with its corresponding bus service.

#### **Extracting Information from Estimated Arrival Time**

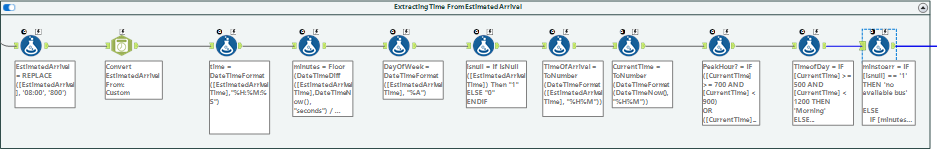


Figure 14: Extacting Information from Estimated Arrival Workflow

**Step 1: Formatting the Timestamp**

Initially, I used a formula to replace the timezone information in the 'EstimatedArrival' from “08:00” to “800” . Then, I converted the modified timestamps from string format into a proper datetime format using Alteryx's DateTime tool. This allowed for the extraction and calculation of time elements in subsequent steps.

**Step 2: Calculating Time Differences**

With the timestamps converted, I computed the difference in minutes between the 'EstimatedArrival' and the current time. This calculation was performed using the DateTimeDiff function, which returned the time difference in seconds. I then divided this value by 60 to get the total minutes, rounding down to the nearest whole number to represent the minutes until arrival. Flooring of the minutes to arrival is done as per convention based on the LTA API’s documentation.

**Step 3: Extracting Day of Week**

Next, I determined the day of the week for each 'EstimatedArrival' timestamp using the DateTimeFormat function, which helps in analysing patterns and trends for different days.

**Step 4: Handling Null Values**

I addressed any null values in the 'EstimatedArrival' field by creating a binary column “isnull”. This is useful for identifying rows with missing arrival.

**Step 5: Further Time Calculations**

The workflow then continued to refine the time data by converting the 'EstimatedArrival' into a numeric format representing the time of arrival and also capturing the current time in a similar numeric format.

**Step 6: Identifying Peak Hours and Time of Day**

I employed conditional logic to categorize the time data into 'PeakHour' and 'TimeOfDay'. Peak hours were identified based on whether the current time fell within the morning or evening rush hour intervals. Similarly, the 'TimeOfDay' was categorized into 'Morning', 'Afternoon', or other parts of the day based on the current time.

**Step 7: Conditional Logic for User-Friendly Descriptions**

The formula uses conditional statements to determine the appropriate text based on the value in the 'minutes' field:

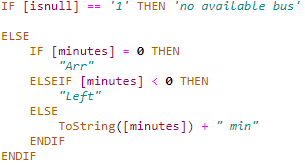


Figure 15: Formula for minutes to arrival front end display

If the 'isnull' flag is set to '1', indicating no data is available, 'minstoarr' is set to "no available bus", which informs the user that there is no bus arrival data.

If 'minutes' equals zero, the output is "Arr", signalling that the bus is arriving.

If 'minutes' is less than zero, the output is "Left", indicating the bus has already left the stop.

Otherwise, the 'minutes' value is converted to a string and concatenated with " min", providing users with an estimated time of arrival in minutes. The resulting 'minstoarr' column offers an intuitive description of the time until the bus arrives, enhancing the ease with which users can read and understand the data. For instance, instead of interpreting raw numbers, users can see clear indications like "no available bus", "Arr", "Left", or "5 min".

### **Bus Stop API**

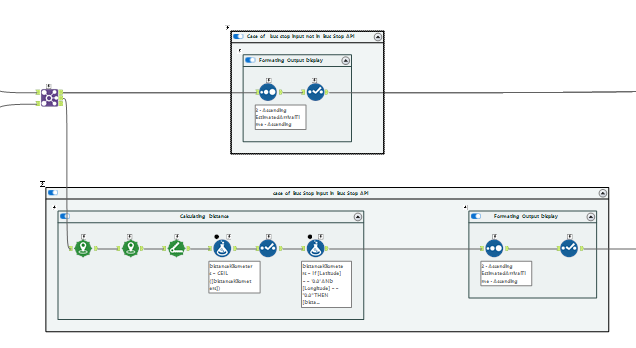
****

Figure 16: Workflow for Bus Stop API

A join tool was used to combine the data from the bus timing API with the data from the bus stop API where the bus stop code in both data sets match. In the case where the input bus stop number exists in the bus stop API it will go into the join node. Else it will go into the left node where only data from the bus timing API is parsed through.

**Case of Bus Stop Number in Bus Stop API**

After combining the data from both datasets, the description of the bus stop and the name of the road it is on is added into the dataset to be used for easy identification of the bus stop. Then the distance between the bus and the bus stop is calculated by creating points of the bus stop and bus by utilising their latitude and longitude in the Create Points tool. Then their distance is calculated using the Distance tool. The distance is then rounded up to the nearest whole number for convenient display.

Formatting Distance for Display

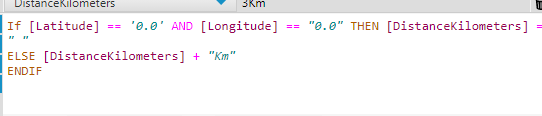


Figure 17: Formatting Distance for Display Formula

I employed a formula to format the 'DistanceKilometers' field for presentation. If both latitude and longitude are recorded as '0.0'—indicating an absence of this data—the output is an empty string. Otherwise, the 'Km' suffix is added to the numerical distance, clearly conveying the measurement in kilometres.

The data of each bus service set is then sorted by arrival time and duplicated columns from the join are removed using the select tool.

**Case of Bus Stop Number not in Bus Stop API**

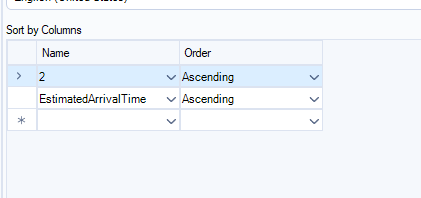
****

Figure 18: Sorting Conditions

For instances where bus stop numbers don't match with the Bus Stop API, the data is simply sorted by arrival time. This ensures that even without additional stop details, users can still view the bus arrivals in the correct sequence.

### **Bus Route API**

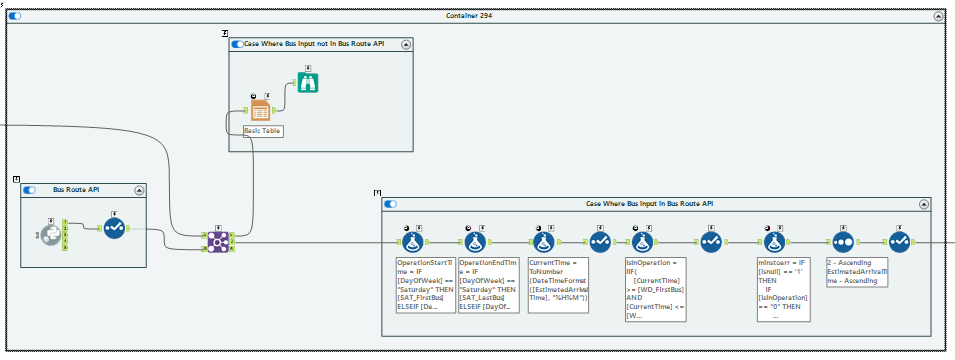
****

Figure 19: Bus Route API Workflow

A join tool was used to combine the data from the bus timing API with the data from the bus route API where the bus stop code in both data sets match. In the case where the input bus stop number exists in the bus stop API it will go into the join node. Else it will go into the left node where only data from the bus timing API is parsed through.

**Case of Bus Stop Number in Bus Stop API**

**Operational Status Assessment**

I've designed formulas to assess the operational status of buses. These determine if a bus is in service based on the current time and the bus's scheduled start and end times, which vary depending on the day of the week (weekday, Saturday, or Sunday). The bus operation information can be found in the Bus Route API. For example, if the current time is within the operational hours for that day, the bus is considered in operation; if not, it's not in operation.

A screenshot of a computer

Description automatically generated

Figure 20: Formula to Find Bus Operation Start Time

A screenshot of a computer

Description automatically generated

Figure 21: Formula to Find Bus Operation End Time

A computer code with text

Description automatically generated

Figure 22: Formula to Determine if Bus is in Operation

**Formatting for Display**

A computer screen shot of a program

Description automatically generated

Figure 23: Formula to Update Display of Number of Minutes

The 'minstoarr' field is updated using a formula that presents the operational status in a user-friendly format. If the bus is not in operation, it displays "Not in Operation", if no estimated arrival time is available it shows "No Est. Available", otherwise, it indicates the minutes until arrival or if the bus has arrived/left. This makes the data intuitive for end-users, providing them with immediate understanding of each bus's status.

**Case of Bus Stop Number not in Bus Stop API**

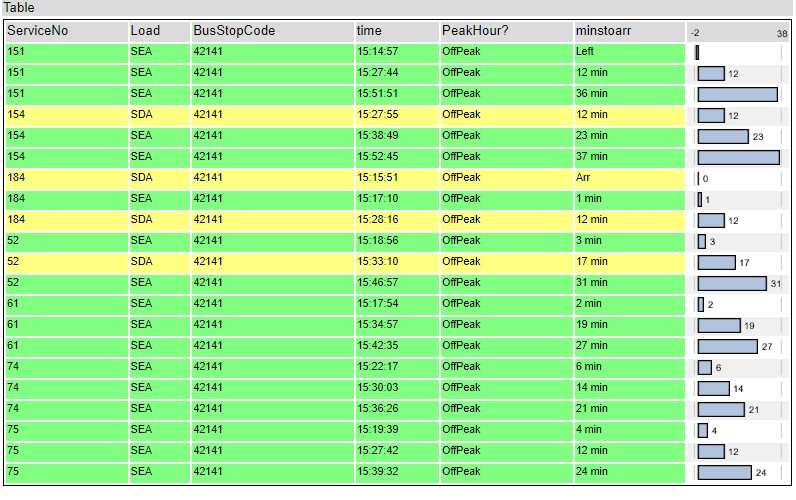
****

Figure 24: Example of Basic Table Output

In cases where the input from the user does not correspond to any entry within the Bus Route API, the workflow is designed to handle this exception. If the bus stop number provided by the user cannot be matched with the data from the Bus Route API, the data is directed to a separate stream in the workflow. This stream includes a basic table that contains default or placeholder information that can be displayed to the user.

**Error Handing for Supporting APIs**

**A screenshot of a computer

Description automatically generated**

Figure 25: Supporting API Workflows

**Error Handling in API Data Integration**

When bus stop numbers from user inputs are matched with those in the bus stop API, they proceed through a joining process that merges bus stop and route data based on bus stop number and service number. This integration enriches the dataset with both location and operational details.

For inputs not found in the bus stop API, the workflow segregates this data and handles it separately, preventing workflow disruption. This bifurcation allows for continuous operation and integration of available data, while also managing anomalies or errors effectively.

In cases where the bus stop input is validated within the API, further joins with the bus route API are conducted, enhancing the data set with additional service details. Conversely, non-matching inputs bypass this step and are processed independently, maintaining workflow integrity and data accuracy.

## 5.Enhancements

To make the bus arrival data accessible and user-friendly, I have developed a website that displays this information in a well-organized and readable format. Furthermore, for analytical purposes, the data has been exported to a business intelligence tool, PowerBI, enabling a deeper insight into the data collected.

### **Collecting Output Data for External Use**

The workflow has been structured to yield four potential outcomes, depending on the data sources combined:

1. Data from only the Bus Timing API.
2. Combined data from the Bus Timing and Bus Stop APIs.
3. Combined data from the Bus Timing and Bus Route APIs.
4. An integration of data from the Bus Timing, Bus Route, and Bus Stop APIs.

Each of these outcomes caters to different use cases, from providing basic timing information to offering enriched details with stop descriptions and route paths.

**A screenshot of a computer

Description automatically generated**

Figure 26:Possible Outputs of Workflows

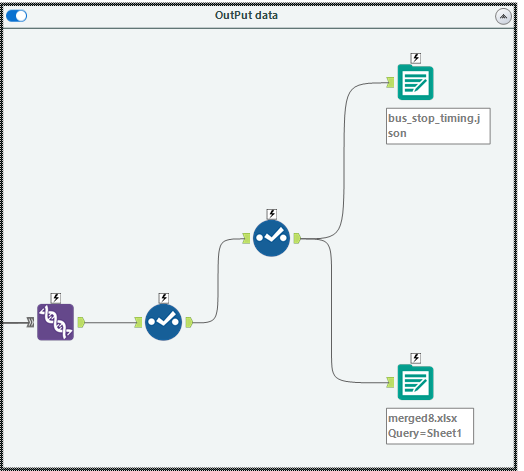
****

Figure 27: Union of Workflow Outputs

**Consolidating Data for Display and Analysis**

The output from each potential workflow path is funneled into a merged dataset using a merge function, ensuring that the entirety of the collected data is preserved. This merged dataset is then exported to an Excel file, which can be imported into PowerBI for comprehensive analysis. Additionally, the consolidated data is formatted into a JSON file, ready to be utilized for populating the website with dynamic bus information. This step is crucial for keeping the website updated with the latest data without manual intervention.

**Handling Null Values in Merged Data**

In the merged dataset that combines data from the Bus Timing, Bus Stop, and Bus Route APIs, it's designed to handle inconsistencies by allowing columns without corresponding information to display null values. This ensures data integrity by clearly marking where information is not available, thus maintaining the accuracy of the dataset for end-user display on the website and for in-depth analysis in PowerBI.

### **Bus Stop Timing Website for Front End Display**

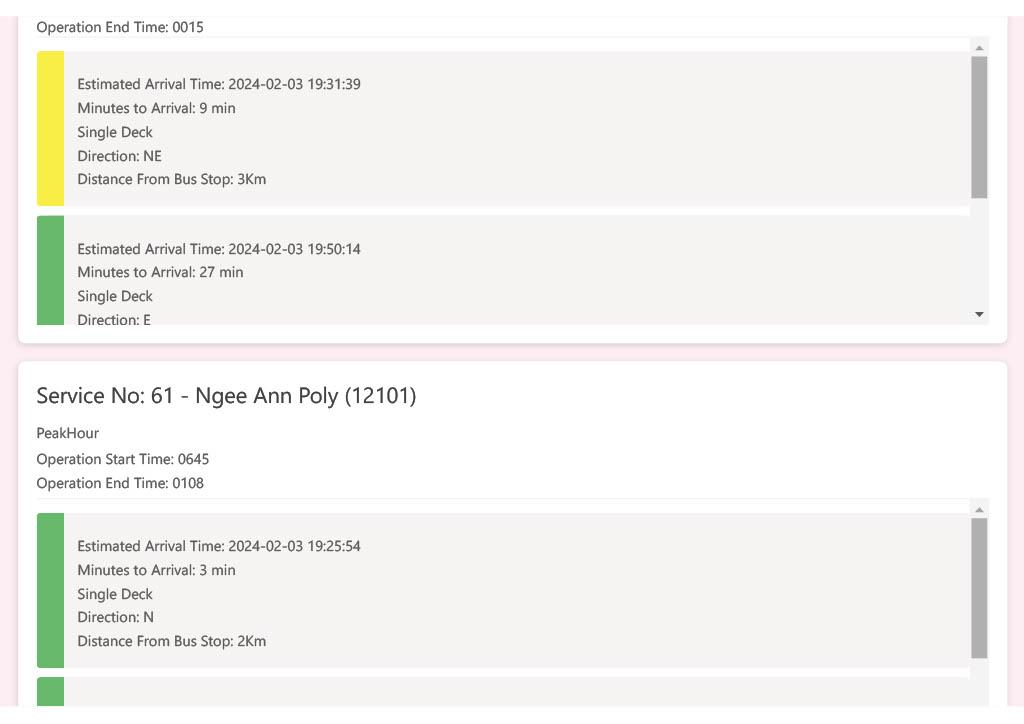
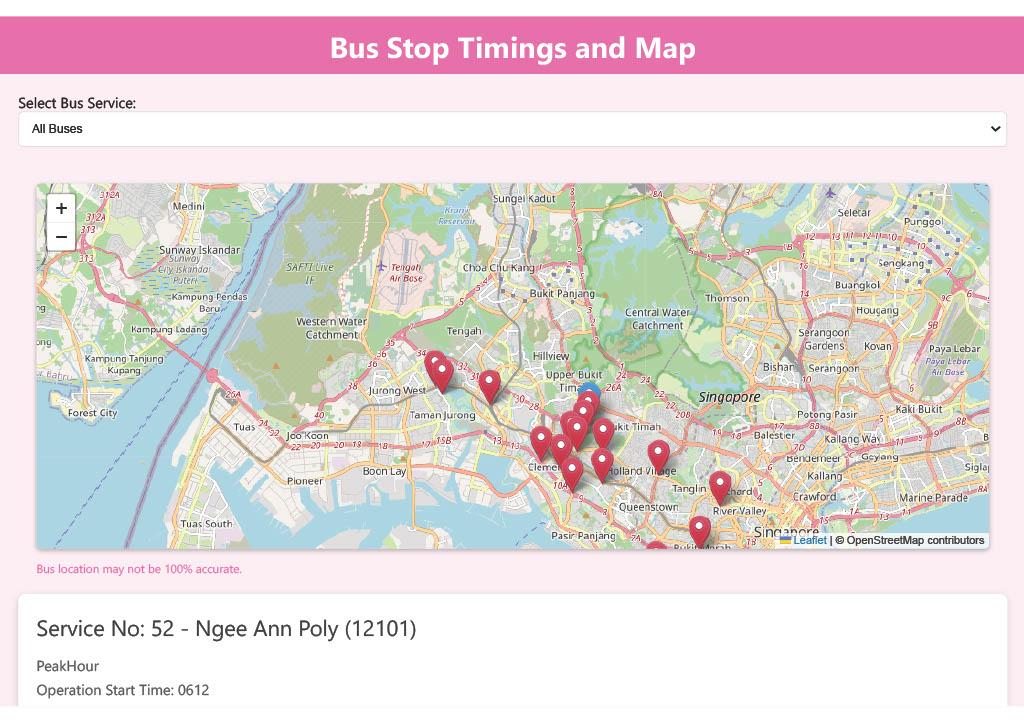
****

Figure 28: Bus Timing Website Example 1

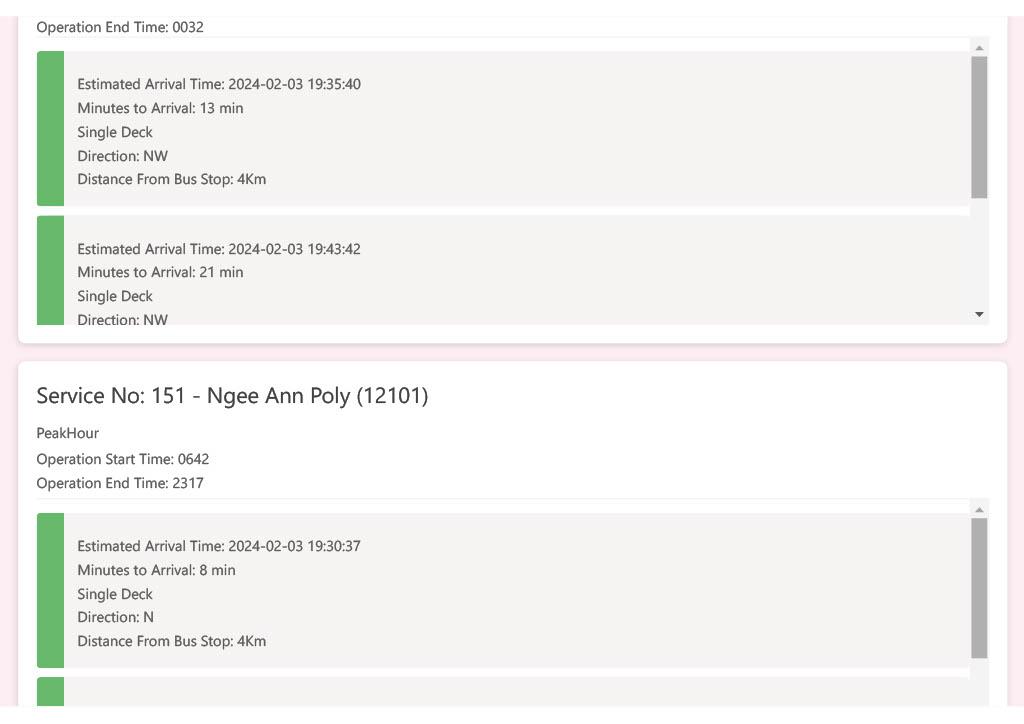
****

Figure 29:Bus Timing Website Example 2

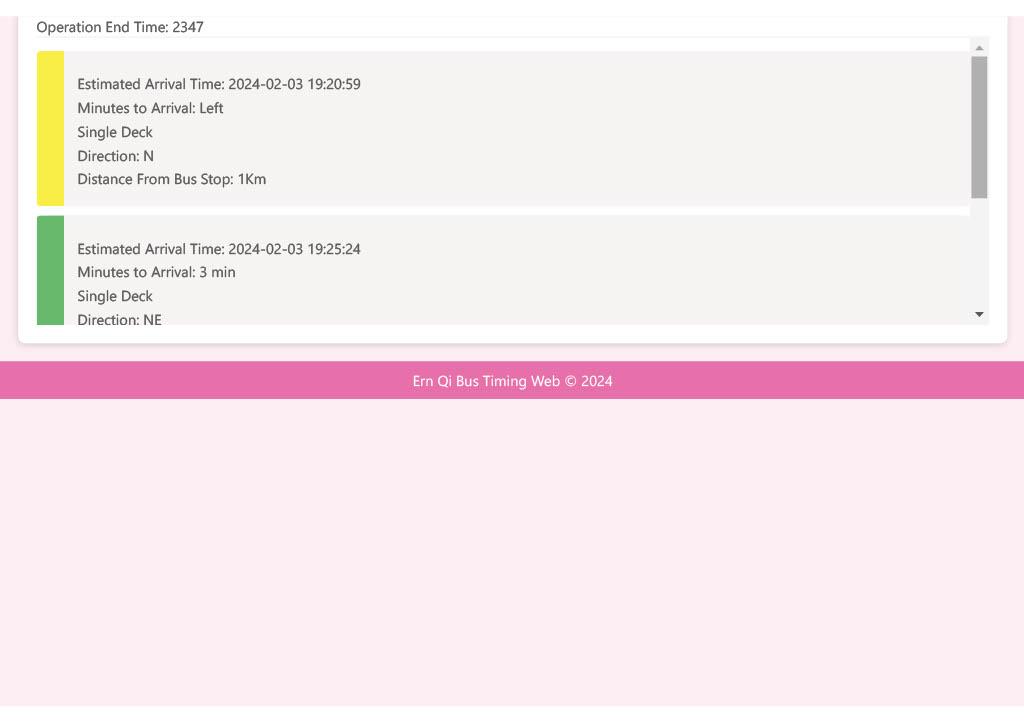
****

Figure 30:Bus Timing Website Example 3

The web application I have developed offers a seamless and interactive experience for users seeking real-time bus stop timing information. By integrating a map view with a responsive dropdown menu, the application presents a comprehensive solution for public transit users to plan their journeys with precision and ease.

**In-Depth Functionality:**

* **Service Selection Interface:** The dropdown menu is dynamically filled with unique bus service numbers from the bus\_stop\_timing.json file, enabling users to tailor the display to the bus service of their interest. The application's responsive design ensures that upon selection, both the map markers and the bus timing details are updated to reflect the chosen service, maintaining the relevance and utility of the information presented.
* **Interactive Map Display:** Utilizing Leaflet.js, the application provides a dynamic map that places markers for buses and bus stops based on geographic coordinates. Selection of a bus service adjusts the map to show live locations of buses, marked in red, and destination stops, marked in blue. Interacting with the red bus markers reveals essential details such as service number and minutes to arrival, enriching the user's situational awareness.
* **Bus Timing Visualization**: A dedicated section beneath the map exhibits bus arrival times through distinct, informative cards. Each card, corresponding to a bus, concisely displays its expected arrival, type, direction, and distance from the stop. The cards use color coding—green for available seating, yellow for standing space, and red for full capacity—to visually convey bus load, facilitating quick and informed travel decisions by users.
* **Organized Information Presentation**: The web application intelligently groups and displays bus information by service number at each stop. This organization ensures clarity and orderliness, providing users with easy access to key details such as the bus service number, stop description, operational hours, peak times, and direction.

**Technical Integration:**

The backend of the application adeptly processes the bus timing data, while the frontend renders this information into a structured and interactive interface, complete with a map and timing cards. The amalgamation of robust data processing and intuitive interface design ensures a powerful and user-friendly platform for end-users. Refer to appendix for full HTML code.

**Limitations of the Current System**

A notable limitation in the current setup of our bus information system lies in the lack of direct integration between the Alteryx workflow and the website's code. The Alteryx workflow effectively processes and compiles the data, which is then saved as a JSON file. However, the website, which relies on this JSON file to display up-to-date information, cannot automatically detect when a new version of the file is available. Consequently, to ensure that the website reflects the most recent data processed by the workflow, a manual re-run of the website's code is required. This additional step is necessary to reload the latest JSON data onto the website so that users are presented with the most current bus timings and information.

### **PowerBi Visuals for Back End Analysis**

**Bus Stop Analysis**

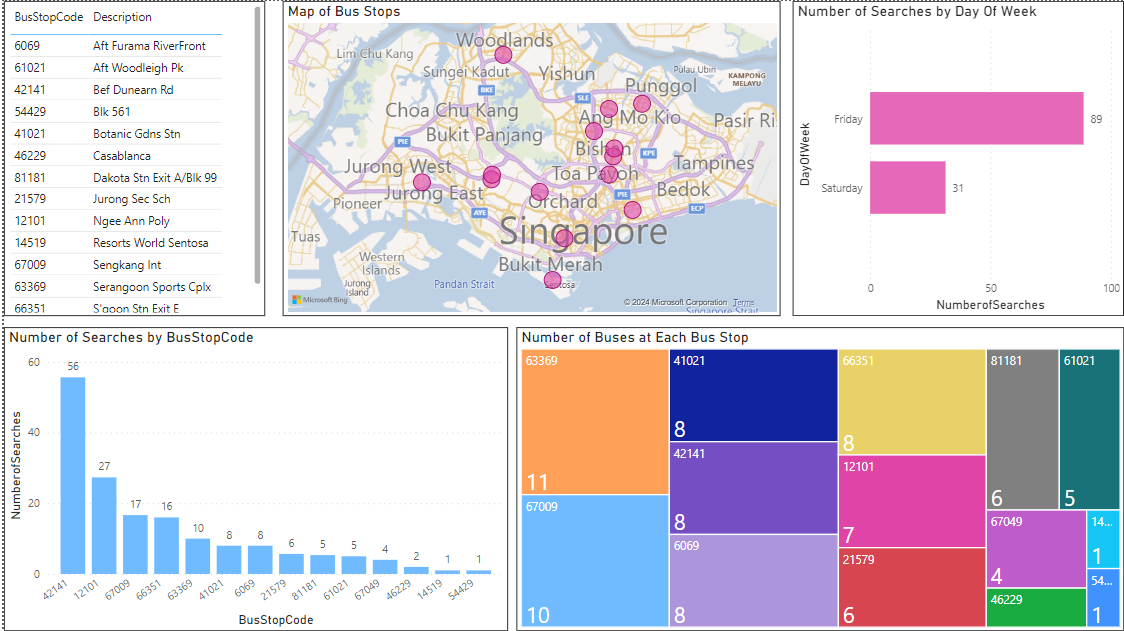


Figure 31: Bus Stop Analysis Dashboard

**Bus Analysis**

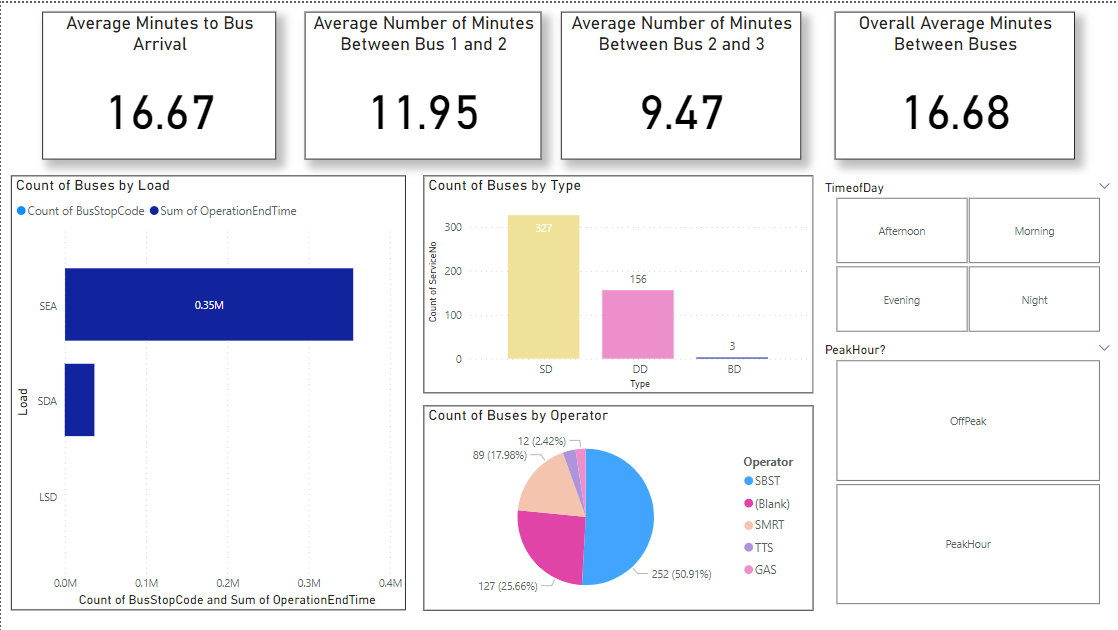
****

Figure 32: Bus Analysis Dashboard

The PowerBI dashboards are curated to assist in the back-end analysis by visualising dataset into interpretable graphics. The dashboards are split into two main focuses: one on Bus Stops and the other on Buses, each providing distinct insights.

**Dashboard 1: Bus Stops Analysis**

The Bus Stops dashboard concentrates on the spatial distribution and user engagement with various bus stops. A map visual pinpoints the location of bus stops, offering a geographic context to the data, which is essential for identifying areas with high bus stop densities. Additionally, the dashboard includes a histogram and a bar chart showing the number of user searches by bus stop code and day of the week, respectively. These visuals help identify patterns in passenger usage and can guide resource allocation, such as the need for more frequent services at popular stops or on busy days.

**Dashboard 2: Buses Analysis**

The Buses dashboard delves into the operational aspects of bus services. It features key performance indicators like average waiting times and headway between buses, presented in a clear, numerical format. The dashboard also showcases the distribution of bus load capacity, offering insights into the passenger volume and potentially overcrowded routes. Furthermore, the dashboard breaks down the number of buses by type and operator, allowing for a detailed performance review of the fleet and service providers. This data can inform decisions on fleet management and operator performance evaluation.

**Analytical Advantages:**

Each dashboard is instrumental in back-end analysis by transforming raw data into actionable insights. For transportation planners and service operators, these dashboards provide a powerful analytical tool to enhance service delivery, optimize routes, and improve overall passenger experience based on actual usage data and operational metrics.

## 6. Summary

This report outlines the comprehensive process of developing a real-time bus information system aimed at enhancing the commuting experience for public transportation users and improving operational efficiency for bus companies. The project employs a value-based approach, addressing the need for accurate, real-time bus arrival information for commuters and detailed insights into passenger flow and bus stop utilization for bus companies.

The workflow effectively simplifies complex bus timing data from the Singapore Land Transport Authority's API into easy-to-understand information. Using Alteryx and Python, it processes and organises this data by filtering out unnecessary details and structuring the remaining information into a clear format. This includes sorting bus services by their numbers and arrival times, filling in any missing data, and calculating useful insights such as how long until the next bus arrives. Additionally, it enriches this data with extra details from other APIs, providing a full picture of bus services that helps commuters plan their journeys and bus companies improve their operations. Essentially, it turns complex data into practical, actionable information for both passengers and service providers.

The workflow's output is not only tailored for direct end-user consumption through a user-friendly website but also formatted for analysis in PowerBI. This dual approach ensures that both commuters and bus companies can derive significant benefits. Commuters enjoy an enhanced travel planning experience with real-time bus timings and additional service details, while bus companies gain valuable insights from the analysis of bus stop usage, passenger flow, and service demand, facilitating data-driven decisions to optimise routes, schedules, and fleet management.

Overall, this project exemplifies how using Alteryx and integrating Alteryx with other platforms and tools can extract valuable information from raw data and transform it into actionable insights. By addressing the initial value-based problem statement, the project significantly improves the commuting experience for individuals and supports bus companies in making informed operational decisions, leading to more efficient and responsive public transportation services.

## 7. Reflection

Reflecting on my recent project, I've come to appreciate the significant strides we've made while also recognizing areas ripe for improvement. One pivotal enhancement I envision is the development of an automated solution that seamlessly bridges the gap between data processing in Alteryx and data presentation on our website. This innovation would not only streamline the update process but also enhance efficiency dramatically.

In tackling user interaction, specifically the handling of invalid bus stop code inputs, I see a clear path for improvement. Incorporating robust error handling mechanisms would significantly elevate the user experience. Moreover, addressing the display of null values on the website is crucial. Ensuring that these null instances are handled gracefully would further polish the presentation of our data.

On the analytics front, I aim to fortify our analysis within PowerBI. The current setup is effective, yet I believe there is room to expand its robustness, particularly in accommodating and intelligently analysing large datasets. A shift towards more general analyses, such as grouping data by Neighbourhood Regions rather than specific bus stop codes, could offer broader insights and prove more versatile in handling extensive data.

Throughout this project, my skills have grown exponentially. Using Alteryx to execute API calls, download, and display data has been enlightening. I've honed my ability to clean and transform data, create new features using formulas, and efficiently output data from Alteryx. Integrating Alteryx with platforms like PowerBI has enabled me to make sense of the workflow’s output data, thereby enhancing the overall data analysis process. I also have learnt how to design a pipeline to ensure that the data is being transformed to meet the requirements of my problem statement.

This project has been an eye-opening experience, introducing me to the power and potential of no-code platforms. I've come to realize the sheer potential of raw data and the impact of well-designed data pipeline infrastructures tailored to specific requirements. The journey through this project has ignited a curiosity in me to delve deeper into the capabilities of Alteryx. I am motivated to explore further the potential of Alteryx in crafting sophisticated data pipelines, leveraging its no-code platform to streamline and enhance data processing and analysis endeavours.

## Appendix

### Notice on exporting Alteryx Workflow

After importing the **S10243067K\_OhErnQi\_DDP\_ASG2.yxzp** file, update the merge8.xlsx Output option to be from ‘Append to Existing sheet’ to ‘Create New Sheet’ before the first run. This ensures that the Merged excel file can be created first. In subsequent runs, the merge8.xlsx Output option should be changed back to ‘Append to Existing sheet’.

**Upon importing**

**A screenshot of a computer

Description automatically generated**

**Before First Run**

**A screenshot of a computer

Description automatically generated**

**After First Run**

**A screenshot of a computer

Description automatically generated**

**For Website Output JSON**

To ensure that the data from the latest Alteryx run is populated into the bus timing website, save the json output location to be in the same file as the HTML file.

A screenshot of a computer

Description automatically generated

### Website HTML code

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen with text on it

Description automatically generated