

**School of InfoComm Technology**

**Distributed Data Pipelines**

Diploma in Data Science (DS)

October 2022 Semester

# INDIVIDUAL ASSIGNMENT 2

## (40% of Distributed Data Pipelines Module)

**Deadline for Submission:**

**10th Feb 2023 (Friday), 2359 Hours**

|  |  |  |
| --- | --- | --- |
| Student Name | : | Tran Huy Minh |
| Student Number | : | S10223485H |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 17th Feb 2023, 23:59.

Presentation Link: <https://youtu.be/Qm_b88Ye0g8>

Contents

[1. Problem Statement Formulation 3](#_Toc126890875)

[2. Preparation 4](#_Toc126890876)

[3. API Call 6](#_Toc126890877)

[4. Data Display 8](#_Toc126890878)

[5. Enhancement 12](#_Toc126890879)

[6. Summary 14](#_Toc126890880)

[7. Reflection 16](#_Toc126890881)

# Problem Statement Formulation

For millions of individuals, public transportation has been a necessary component of getting to and from work, school, and appointments. The unpredictable nature of wait times, on the other hand, is a major source of annoyance, lost productivity, and stressful commutes for regular users of public transportation. This report will investigate the development of a real-time bus service data pipeline as a solution to this issue.

The proposed data pipeline will make use of real-time stream data from a dataset relevant to land transportation given by the Land Transport Authority (LTA). The pipeline will offer real-time information on bus arrival times at defined bus stops and will be constructed using Alteryx Designer, a potent tool for data pipelining. Additionally, this data pipeline would be connected to Power BI to create a frontend display of bus arrival information. This will enhance productivity and make traveling less stressful by enabling users of public transportation to better organize their regular commutes and eliminate pointless wait periods.

As the data pipeline for bus stop information is targeted towards the general public who uses public transport, they would seek easy-to-read bus information such as the bus service number, the estimated arrival time, type, load, waiting time in minutes, and the order of the buses. Additionally, they users would want a comprehensive frontend display which would house this information alongside a map showing the locations of the buses, and a way to input their desired bus stop code for dynamic bus stop information.

This assignment will not only help the general public, but it will also give students a great chance to learn. Students will have the opportunity to demonstrate their understanding of the data pipelining procedure and the use of Alteryx components, which will be useful knowledge for their future careers in data science. As students pursue careers in data science, this project will also give them practical experience designing and implementing real-world data pipelines.

In conclusion, the development of a real-time bus service data pipeline has the potential to significantly enhance the daily experiences of those who use public transportation while also giving students invaluable data pipelining training.

# Preparation

Before building the data pipeline within Alteryx Designer, the bus arrival API provided by the LTA on DataMall is explored in order to understand what information is required to access the real-time bus arrival dataset.

The API access is first requested in DataMall in order to access the dataset which would grant the AccountKey required to make API calls through an email. The bus arrival API URL, <http://datamall2.mytransport.sg/ltaodataservice/BusArrivalv2?BusStopCode=>, is also required for the pipeline to make requests specific to the desired dataset as there are a variety of different APIs unrelating to bus stops. Additionally, a bus stop code needs to be appended onto the API URL to be able to access the bus stop’s particular dataset. In the case of this assignment, the bus stop code “12101”, for the bus stop outside Ngee Ann Polytechnic will be used alongside the user’s own input for dynamic bus stops.

Based on the API for bus arrival documentation and user guide, making the API call would return a JSON response housing the desired bus stop’s bus service numbers, the next three buses arriving for each service number, operator, origin code, destination code, estimated arrival date and time, latitude and longitude of the bus, the visit number, the current load, the features of the bus, and the bus type. In addition, the update frequency of the API and dataset is around one minute. These API responses are also limited to 500 records of the dataset per call where subsequent records can be retrieved by appending the $skip operator to the API URL when necessary. Fortunately for the bus stop arrivals dataset, this would not be needed due to the low volume of data.

Graphical user interface

Description automatically generated with medium confidence

Above is the ‘User Inputs’ container within the Alteryx Designer Software. At the top is a Text Box tool that requests the user to enter a bus stop code which connects to the Action tool that would update the BusStopCode cell in the Text Input tool to the user’s custom input.

Inside the Text Input are four manually entered columns, the accept ‘application/json’, AccountKey, BusStopCode, and the API URL.

A picture containing text

Description automatically generated

The output of the Text Input is connected to a Formula tool inside the API Call container. The formula essentially appends the provided BusStopCode, either the default 12101 or the user’s input, onto the API URL allowing for a dynamic URL when making calls to download data. Since the BusStopCode is an integer, it is first converted to a string before being added and outputting the new API URL.

The Download tool would utilize the API URL, the accept header, which allows for the specification of response format of the API call that is defaulted to JSON, and AccountKey header, which is necessary for requesting API access in DataMall for access to the dataset. These fields are all required to make an API call to download the data which outputs two columns, the DownloadData hosting all the bus data, and the DownloadHeaders.

The Select tool is then used to select only the DownloadData columns as it is the only field that houses the data relevant to the rest of the workflow pipeline.

# API Call

After using the Download tool to make a call to the API, receiving a response containing the downloaded data in json format and selecting the Download Data field, the output is connected to the JSON Parse tool in the Reading JSON container.

Timeline

Description automatically generated

The JSON Parse tool is used to help perform the necessary operation to transform the downloaded data into an appropriate format or structure before being fed to the next step by inputting the DownloadData JSON format response and outputting the values of the downloaded data into a single string field for further transformation.

The output produces two new fields and multiple new rows. The JSON\_Name field which contains the headers, ‘odata.metadata’, ‘BusStopCode’, and ‘Services’ with the header of each buses’ information currently running in the format ‘Services.0.NextBus3.Load’ where ‘0’ represents one of the bus service numbers, ‘NextBus3’ representing one of the three buses of that service number that is arriving at the bus stop, and the ‘Load’ representing one of the buses’ information. The JSON\_ValueString field contains the corresponding values of the headers where the metadata has the URL, the BusStopCode has the code, and each ‘services’ row contains all corresponding bus arrival data.

Using the Text to Columns tool and a period delimiter, the values in the JSON\_Name field are split into multiple values in three different new columns. This is done to obtain the shared column names for all bus services.

A Filter tool is then used to remove any rows that does not contain ‘Services’, which has all the important data, from the JSON\_Name which leaves only the information regarding each bus. This essentially removes the ‘odata.metadata’ and ‘BusStopCode’ rows values as they are unnecessary.

Graphical user interface, text, application

Description automatically generated 🡪Table

Description automatically generated

As each bus service numbers now share a column containing the names of similar fields in their rows, the Cross Tab tool is used to convert the unique values in the rows to columns that contain their corresponding value in a row grouped by JSON\_Name2, which represents each unique ServiceNo value.

A picture containing chart

Description automatically generated

This is then connected to a Select tool in the Rearranging Data Columns container to remove columns that would interfere with data transformation, namely the JSON\_Name2 and Operator fields.

Graphical user interface, application, table

Description automatically generated

A Transpose tool is then used to pivot the bus information columns back into rows as seen above. This is done to allow for the delimitation of the Name field. A Text to Columns tool is used to split the ‘NextBus’ values and information values such as ‘DestinationCode’ into separate columns using an underscore, ‘\_’, delimiter. As the ServiceNo and ‘NextBus’, represented by the header ‘Name1’, values combine to make a unique compound key, the information values are now all similar throughout all different service numbers and next buses.

Table

Description automatically generated

A CrossTab tool is used again to group the ServiceNo and ‘NextBus’ values to create the unique compound key and convert the information values back into columns with their corresponding values as seen above with the ‘DestinationCode’ and ‘EstimatedArrival’. These values are then sorted by the ServiceNo and ‘NextBus’ values in ascending order which concludes the transformation of data into appropriate format.

# Data Display

Diagram

Description automatically generated with low confidence

With the transformed data in an appropriate format and structure, it is fed into the Data Cleansing tool inside the Data Cleansing and Transformation container where basic data cleaning such as the removal of null rows is done.

Graphical user interface, text

Description automatically generated

Since the ‘EstimatedArrival’ values are in a string format rather than a datetime data type, a Text to Column tool is first used to split the values using the delimiter, ‘T’. This creates two new columns, a column containing the date, and a column containing the estimated time of arrival alongside the ‘+08:00’ representing the GMT+8 for Singapore Standard Time (SST). As this workflow is meant for users residing in Singapore already, the GMT+8 would not be necessary as the estimated time of arrival is already expressed in local time. As such, another Text to Column tool is used to split the ‘+08:00’ from the estimated time of arrival using a ‘+’ delimiter.

The new estimated time of arrival values is then input into a DateTime tool to convert the string into a DateTime format, ‘HH:mm:ss’, to be used for creating a new feature later, and outputs a new column named ‘EstimatedArrivalTime’.

A Select tool is used to remove all unnecessary columns created from the Text to Columns tool as well as renaming one of the columns to ‘EstimatedArrivalDate’ and ‘Name1’ to become ‘BusOrder’.

Next, a Formula tool is used to cleanse some features and create a new one. Although only one Formula tool can be seen inside the Data Cleansing and Transformation container, the tool itself has six different formulas, each doing different things.

Graphical user interface, text, application

Description automatically generated

The first formula creates a new column, ‘WaitTime (minutes)’, representing the estimated number of minutes remaining before the next bus arrives at the bus stop. The formula uses the function DateTimeDiff to find the difference between two datetime values, the ‘EstimatedArrivalTime’ which was previously converted to a datetime datatype, and another function, DateTimeNow which returns the current local computer date and time. However, since the current DateTimeNow is in a different format which includes the date, the DateTimeFormat function is used to convert it to just the time. Finally, the return value of the difference is returned in minutes.

Occasionally, the dataset obtained from the API gives a response where the EstimatedArrivalTime is a few minutes behind the current time, meaning that the bus has either taken a slightly longer time to arrive or has already left the bus stop which the API has failed to update. As such, when calculating the ‘WaitTime (minutes)’, a negative value is sometimes returned.

To solve this, a conditional formula is used where if the ‘WaitTime (minutes)’ is less than 0, then 0 is returned.

Graphical user interface, text, application

Description automatically generated

The next two formulae involve the Latitude and Longitude of the bus. Sometimes the buses that have yet to leave the interchange lack any kind of location data and return a Latitude and Longitude of 0 instead. When using these values on a map visualization in Power BI, they cause the buses to appear far away from where they are supposed to be. As such, when either Latitude or Longitude is equal to 0, it is then replaced with a null value instead.

Graphical user interface, text, application, chat or text message

Description automatically generated

The last two formulae involve the user’s comprehension and interpretability of the data. For the Load and Type of the bus, their crowding levels and vehicle types are abbreviated. For example, the Load value ‘SEA’ represents Seats Available, and the Type value ‘DD’ stands for Double Deck. Generally, the users who are looking for bus information are also on the lookout for this kind of information which shows their importance. However, most users are unable to understand the meaning of these abbreviated values, making them meaningless to the users. As such, a formula is used to convert these values to their proper description as documented by the API’s User Guide.

Diagram

Description automatically generated

After all six formulae have been run on the data, a Join tool is used to append the Operator column, which was initially removed for data transformation, onto the dataset using a join on the ServiceNo from a Select tool as it might contain valuable information to the user. Similarly, the Select tool is used to get the BusStopCode from the first Text Input to use in conjunction with the Append Fields tool to append onto the dataset. Finally, a Sort tool is used to sort the data by ServiceNo and BusOrder in ascending order.

Diagram

Description automatically generated

The Display Data contains tools used to display the final dataset after the Sort tool was used. Starting with the centre left Browse tool showcases the final dataset unchanged. The bottom left most is an Output Data tool that is used to export the dataset as a .txt file with a ‘,’ delimiter which would be further explained in the next section.

Graphical user interface, application

Description automatically generated

For a narrower view of the dataset showcasing columns that only users would want to see, a Select tool is used to select the ServiceNo, BusOrder, EstimatedArrivalTime, Load, Type, and WaitTime (minutes) to be displayed in the centre Browse tool.

Table

Description automatically generated

To only see the first bus of each bus service that is arriving with the estimated arrival time as documented by the assignment write-up, a Filter tool filters to only display the first bus in the BusOrder and the Select tool is used to only display the ServiceNo and EstimatedArrivalTime of the bus.

# Enhancement

The use of additional packages to enhance the model workflow for frontend display was also done such as allowing the user to input their own bus stop code and getting the bus arrival time to be displayed in a PowerBI dashboard visualization.

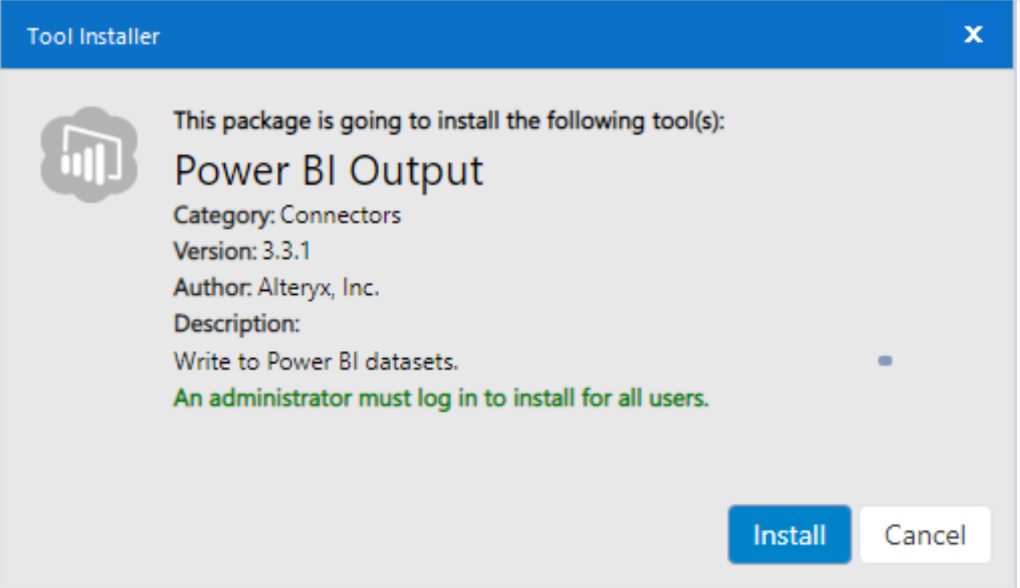
Graphical user interface, text, application, chat or text message

Description automatically generated

In order to utilize the Text Box tool to input a custom bus stop code, the Alteryx workflow would need to be run as an app which would create a pop up prompt as seen above. Entering a bus stop code would update the data downloaded from the API throughout the entire workflow allowing for dynamic bus stop information and arrival times to be obtained and displayed.

This can be seen from the BusStopTiming.txt file that was exported at the end of the data pipeline. This dataset file can then be imported into PowerBI to create a connection, allowing for visualizations to be made off of the data.

However, an explanation for why a direct connection between Alteryx Designer workflow and PowerBI was not possible during this assignment is first owed considering exporting and importing the data into PowerBI might be seen as an inefficient alternative.



In order to create a direct connection between the data pipeline workflow and PowerBI, the Power BI Output tool must first be installed. This new tool can act as a replacement for the Output Data tool as it does not require the export of the dataset into a file.

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

The Power BI Output tool requires authentication by signing into a Microsoft account that is linked to PowerBI Desktop and has access to Alteryx Designer. Considering that the licensing of Alteryx Designer is linked to the school email, it would only make sense to sign in using the school email. However, when attempting to sign in, a message asking for admin approval and permission to use the Alteryx app is required. This could be solved by adding Alteryx Designer into app permissions in Microsoft Azure but access to that is also lacking for the school email. This made it impossible to create a direct connection between the data pipeline in Alteryx and PowerBI.

After exploring several other alternatives and methods, it was settled upon that the dataset would be exported as a .txt file to be imported into PowerBI to imitate a direct connection. As such, whenever the workflow is run, the .txt file is updated to the real-time data and the imported data in PowerBI can be easily refreshed with the new data accordingly.

Map

Description automatically generated

Above is a PowerBI frontend display of the final exported dataset. It shows the Bus Stop Code of the dataset, the current date of the dataset, a matrix showcasing the wait time in minutes for bus arriving colour-coded, the earliest and latest arrival times of each bus service, the location of all buses on a map, and several filters relating to the service number, bus order, bus type, and load that the users would be interested in when it comes to the bus services. In addition, this entire display is dynamic due to the user’s custom bus stop input by running the workflow as an Alteryx app.

# Summary

Below is the entire data pipeline workflow inside Alteryx Designer.

Diagram

Description automatically generated

To summarize, the data pipeline workflow focuses on providing real-time information on bus arrival times to enhance productivity and make traveling less stressful for users of public transportation. The pipeline makes use of real-time stream data from a dataset relevant to land transportation given by the Land Transport Authority (LTA) and would be connected to Power BI to create a frontend display of bus arrival information. The data pipeline is built using Alteryx Designer, a potent tool for data pipelining, and the output would be easy-to-read bus information such as the bus service number, the estimated arrival time, type, load, waiting time in minutes, and the order of the buses.

Before building the pipeline, the bus arrival API was explored to understand what information was required to access the real-time bus arrival dataset. The API call would return a JSON response housing the desired bus stop’s bus service numbers, the next three buses arriving for each service number, operator, origin code, destination code, estimated arrival date and time, latitude and longitude of the bus, the visit number, the current load, the features of the bus, and the bus type.

In the pipeline, the output of the Text Input is connected to a Formula tool inside the API Call container to make the API call and retrieve the bus stop’s particular dataset. The Formula tool also cleanses the data and creates a new column, ‘WaitTime (minutes)’, representing the estimated number of minutes remaining before the next bus arrives at the bus stop. Additionally, a conditional formula is used to solve the issue of a negative value being returned when the bus has already arrived at the stop or the API has failed to update.

Finally, the data is connected to Power BI to create a comprehensive frontend display that would show the bus information alongside a map showing the locations of the buses and the ability to input their desired bus stop code for dynamic bus stop information.

Proof of completion certificate document of the Alteryx Foundation Micro-Credential and completion of all of the “Getting Started Learning Path”:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# Reflection

The current solution for bus arrival timings has been effective, however, there are numerous ways to improve it for better performance and user experience. In this reflection, I will elaborate on some of these potential improvements.

One of the improvements that can be made is to make multiple API calls to expand the dataset. By utilizing the origin and destination code of the buses, it will allow for a deeper analysis and more data to display to the users on the frontend. This additional information can help users make informed decisions on their travels, especially if they are in a new place and want to know the details of the bus services available.

Another improvement that can be made is to allow a direct connection between the Alteryx Designer data pipeline workflow and Power BI. Currently, updating the final dataset requires the data pipeline workflow to be run every time, which can be time-consuming. By unlocking admin access and utilizing the Power BI Output tool, it will create a real-time display of bus arrival timings, eliminating the need to run the workflow and export a .txt file. This will allow for faster and more accurate information to be displayed, making it more convenient for the users.

Currently, updating the final dataset also requires the data pipeline workflow to be run every time. Instead, a task scheduler can be utilized to run the data pipeline every so often automatically rather than having to click on run.

Finally, a proper user text input UI within Power BI would be a better alternative to the text box tool UI within Alteryx Designer when updating the Bus Stop Code. A user-friendly interface can make it easier for users to input the required information, reducing the chances of errors and making the process smoother.

The Distributed Data Pipelines module has helped me in gaining a comprehensive understanding of the data pipelining process. I now have a clear understanding of the different stages involved in the data pipelining process and how the various components of Hadoop, Apache Spark and Alteryx are utilized to achieve this process.

Apache Spark is an open-source distributed computing system that is used to process large datasets. Spark is highly scalable and can handle data processing tasks in real-time, making it an ideal tool for data pipelines. Hadoop is an open-source software framework that is designed to process and store large amounts of data. Hadoop is used in data pipelines because it provides a distributed file system that can store large amounts of data, making it easier for organizations to manage their data.

Alteryx Designer is another tool that is commonly used in data pipelines. Alteryx Designer is a drag-and-drop interface that allows data analysts and engineers to easily create workflows for processing and analyzing data. The tool is user-friendly and can be used by people without a technical background, making it a valuable tool for organizations that want to leverage their data.

I am also now equipped with the knowledge to design and deploy a data pipelining infrastructure and understand how to optimize this infrastructure to meet the needs of a given data project. This has been made possible through the hands-on experience I gained from this assignment.

I also have a better understanding of the importance of selecting the right tools and technologies for a data pipeline infrastructure. This is crucial in ensuring that the data pipeline is able to effectively support the analysis process and provide meaningful insights.

However, something that I could have learnt better would be implementing the different types of connections between the backend workflow and the front end display such as the real-time connection between Alteryx and Power BI.

In conclusion, the Distributed Data Pipelines module has been a valuable learning experience and has equipped me with the knowledge and skills needed to design, deploy and optimize data pipeline infrastructures. This will be of immense value in my future data science projects.

Top of Form

Bottom of Form