AI Based Traffic Management Dashboard

1) Introduction

Eliiza is a data science and data engineering company that is part of the Mantel Group of companies. Eliiza specialises in providing digital solutions, specifically related to data and AI, to businesses across various domains. Some notable clients of Eliiza include ANZ and AGL. The challenge of the project was to develop a protype data engineering system with the goal of providing a business opportunity for Eliiza to expand into the taxi/ ride sharing space and increase their client base. In this project, partnering with Eliiza to tackle the challenge, an AI based traffic management dashboard was developed.

Since opening back up from the COVID-19 pandemic, demand for taxis have dramatically increased as countries make a return to normalcy and people get on the move again. However, what has not increased by the same amount, is the number of taxi drivers that are available due to shortages in taxi licenses available brought upon by a backlog of applications [1]. Additionally, a move to the digital space and increased competition from ride-hailing companies such as Uber and Grab have caused many drivers to switch due to the better opportunities that these companies present hence contributing to the rapidly shrinking taxi industry [2]. Coupled with increased fuel prices brought on by the war in Ukraine [3], it has never been a more important time, if any, than now for taxi companies to effectively manage resources to efficiently use the resources they have at hand and to minimise costs and maximise revenue.

The end users of this problem are the taxi companies, taxi drivers, as well as the passengers. For the taxi companies and taxi drivers their pain points are identifying ways to effectively manage their fleet of vehicles of varying sizes (ability to carry different number of passengers) and how to distribute them across an area at different times of the day and in different weather conditions to maximise their coverage. As for the passengers, ineffective distribution of taxis could lead to taxis being unavailable in their surrounding locations or increased taxi fares when the demand for taxis heavily outweighs the available supply.

2) Project Scopes

2.1. Aims

1) End-to-end system capable of delivering insights from multiple data sources in real-time.
2) Solution on Google Cloud Platform (GCP) with best data engineering practices.

2.2. Deliverables

1) Data sourcing and data from 2 sources incorporated.
2) Data analysis to check data quality.
3) Actionable insights by incorporating machine learning models.
4) Data pipeline on GCP.
5) Transformations to transform data, and for pipeline robust.
6) Architecture diagram of the end-to-end system.
7) Dashboard to visualise insights generated.

3) Project Management

Microsoft Teams was used for collaboration between team members. Twice weekly 30-minute to 1-hour long meetings were held every Monday and Thursday where sprint planning, division of tasks, updates of tasks, and blockers were discussed. Teams was also used for the weekly 15-minute stand-ups with the Project manager – Tomas Turek. Additionally, a Teams page was set up for the project which to share, store, and collaborate on project related documents.

Communication with Eliiza was conducted through Google Meet. Eliiza provided guidance through twice weekly meetings that occurred every Monday and Thursday and were hosted by dedicated supervisors – Horace Yeung and Paras Sitoula. Eliiza also provided the required computational resources such as access to Google Cloud Platform services to develop the solution prototype. The duration of these meetings varied between 30 minutes to 1 hour based on agenda set by the team as well as any blockers that were encountered. Additionally, Slack was used for informal communication and to resolve any serious blockers outside of the allocated meeting times.

Other tools that were utilised include Jira and GitHub. Jira was used to plan out the 2-week sprints and to store the backlog of tasks that needed to be completed for the project. For the 2-week sprints, each team member created user stories and outlined acceptance criteria for their assigned tasks. Progress made on tasks were tracked by moving tasks between "To-do", "In progress", "Done" or "Blocked" columns if any blockers were encountered. Any source code that was generated through any of these tasks (pipeline code or SQL queries) were stored on the project repository on GitHub. GitHub was also used as a tool for code version control.

4) Methodology

4.1. Data sourcing and data from 2 sources incorporated

Deliverable 1 was to incorporate two data sources into the end-to-end system. The first data source was the New York City taxi data – specifically the 2015 Yellow Taxi dataset obtained from the NYC Taxi and Limousine Commission's (TLC) website [4]. The dataset contained every trip that was completed between January and June 2015. A public Pub/Sub topic that was that was streaming this data in "real-time" (emulated) was available on GCP. To access this data, a new topic for the taxi data was created on Eliiza's GCP environment. Then a Pub/Sub to Pub/Sub Dataflow job was created to feed the taxi data from the public topic to the project's Pub/Sub topic.

For the second source, weather data was chosen and for this the historic 2015 NYC weather data was used. The dataset was obtained from Visual Crossing Weather [5] and contained hourly weather for NYC (Central Park weather station) for. For simplicity and lack of availability of more detailed data, it was assumed that the weather across all of Manhattan was uniform and was based on the dataset. To emulate a real-time stream of data, a Pub/Sub topic was created and a Python program was used to publish the data to the topic on an hourly basis (since the weather data was hourly). The pipeline would then subscribe to the weather topic and pull the weather data from the topic.

4.2. Data analysis to check data quality

To check the quality of the data and to process the data, the steps taken were:

1) loading the full taxi dataset in Jupyter Notebooks into a dataframe using the pandas read_csv function [6],
2) checking the number of rows and columns in the dataframe using the shape function [7],
3) checking the datatypes of the columns of the dataset using the pandas dtypes function [8],
4) checking for any missing values in the columns of the dataframe using the isna function [9] and sum [10] function to show the total number of rows with missing values,
5) finding the rows where there were no passengers i.e., passenger_count = 0,
6) finding the rows where the trip distance was 0 or negative i.e. trip_distance <= 0,
7) finding the rows where the fare was 0 or negative i.e., fare_amount <= 0,
8) finding the rows where the extra costs were negative i.e., extras < 0,
9) finding the rows where the MTA tax costs were negative i.e., mta_tax < 0,
10) finding the rows where the tip amounts were negative i.e., tip_amount < 0,

11) finding the rows where the toll costs were negative i.e., tolls_amount < 0,
12) finding the rows where the total costs were negative i.e., total_amount < 0,
13) finding the rows which were duplicates,
14) finding the rows where the drop off time was before the pickup time,
15) lastly, saving the index of all the rows mentioned in steps 3-12 to a list and removing them using the pandas drop function [11].

4.3. Actionable insights by incorporating ML models

4.3.1. Fare Per-Ride

Regression models were developed to predict the fare per ride using factors such as weekend/weekday, time of day etc. An example of this functionality in real world is Uber's estimated fare is displayed after taking pick-up and drop off points as locations. For the models, a baseline and an advanced model were trained and evaluated which were a linear regression [12] model and XGBoost regression model [13], respectively. The model training, evaluation, and deployment was done in BigQuery. The models were trained, evaluated, and deployed using BigQuery ML's SQL queries. r2_score was used as a metric to evaluate the ML model. As it tells us how well the model is fitting to the data. Higher the r2_score value close to 1, the model fits perfectly and vice versa.

$$\begin{aligned} R^2 &= 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}, \\ &= 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}. \end{aligned}$$

Figure 1: Formula to calculate R^2 value

4.3.2. Payment Type

Classification models were developed to predict the preferred payment type of customers using factors such as weekend/weekday, time of day etc. Based on this insight companies can alter incentives to change the flow of passengers preferred payment type. A logistic regression model [14] was used as baseline model and an XG Boost classifier [13] was used as an advanced model. The training, evaluation, and deployment was done in BigQuery. The models were trained, evaluated, and deployed using BigQueryML's SQL queries. Since, the task was classification, accuracy was chosen as an evaluation metric.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Figure 2: Formula to calculate accuracy

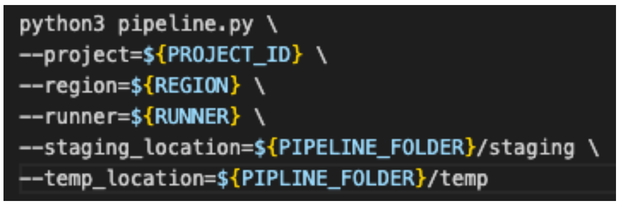
4.3.3. Total Trips Per Hour

Regression models were developed to predict the total trips per hour given the weather conditions of a particular hour and were developed using Jupyter Notebooks as an environment. For the models, a decision tree [15], linear regression [16], and K-Neighbours regressor [17] were used as the baseline models and a Random Forest regressor [18] was used as an advanced model. During modelling, a grid search technique coupled with a 5-fold cross validation strategy was employed to determine the set of model parameters that resulted in the best model performances. The reason for using a 5-fold cross validation technique is to prevent the models from overfitting [19] given the relatively small size of the full dataset (only 4,344 rows). To do this the KFold [20] and GridSearchCV [21] methods from the sklearn.model_selection library were used. The models were evaluated using R^2 as it is a goodness-of-fit measure which indicates much of the variation in the dependent variable (i.e., predicted variable) can be explained by the independent variables (attributes used in the prediction) [22]. To evaluate the models, with the best parameters for the models were determined, each model was then evaluated using a 10-fold cross-validation strategy based on the on R^2 score. To do this, the cross_val_score method from the sklearn.model_selection library [23] was used. The model with the best performance was then saved to a pickle file using the pickle library [24] and then integrated into the pipeline.

4.4. Data pipeline on GCP

The data pipeline was coded in Python and built using Apache Beam, an open-source programming model to define data pipelines such as ETL, batch, and in the case of this project, streaming data processing [25]. Using the apache-beam library from the Apache Beam Python SDK [26] functions were defined combination of Beam transforms (see section 4.5). In Apache Beam, data is stored and processed in immutable groups called PCollections. When a transform is carried out on a PCollection, a new PCollection with the transformed data in generated. This provided an easy way to "branch" the pipeline and carry out transformations on different PCollections in parallel. With the pipeline code on the project repository on GitHub, the repository was then cloned into a virtual environment that was created on a Compute Engine, a service that deliverables configurable virtual machines on GCP, using Python's venv library [27]. This was done to remove the need of using any credentials to execute the pipeline on GCP.

To deploy the pipeline to GCP, Dataflow was used. Dataflow is a fully managed GCP service for executing data processing pipelines [28]. Before execution, the environment variables stored in a .env file were first sourced into the virtual environment containing the code from the project repository. The .env file contained information such as the project ID on GCP, the region to run the Dataflow job in, the path to the cloud storage bucket used for staging, and what runner to use (which was Dataflow). To execute the pipeline on Dataflow, a command was run (see Figure 3) using the SSH terminal in the Compute Engine VM which then automatically created a Dataflow job for the pipeline.



```
python3 pipeline.py \
--project=${PROJECT_ID} \
--region=${REGION} \
--runner=${RUNNER} \
--staging_location=${PIPELINE_FOLDER}/staging \
--temp_location=${PIPLINE_FOLDER}/temp
```

Figure 3: Terminal command used to execute the pipeline on Dataflow

4.5. Transformations to process data, and for robust pipeline

4.5.1. Taxi Data:

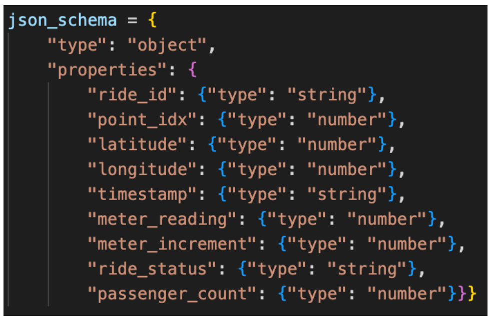
1) Reading in data and parsing JSON format:
    - Firstly, the data was pulled from the Pub/Sub topic into the pipeline and were parsed (i.e., converted) from JSON format into Python dictionaries using the load function from Python's json library [29].

2) Filtering messages by ride status:
    - Once the data was read in and parsed, the taxi data received was then filtered using Beam's Filter function [30] to remove and messages with "ride-status" that were "enroute" as these were considered to provide no valuable insights

in the context of the aims and deliverables. Only messages with "ride_status" with "pickup" and "dropoff" were kept.

3) Format checking:
   - The formats of the JSON messages (taxi data) were then validated using the validate function from the Python jsonschema library [31] to ensure that the schema of each JSON message was correct. This was done to satisfy deliverable 5 and to ensure that the pipeline was robust to any potential errors in the data (i.e., would not fail if a message with an incorrect format was passed through the pipeline).

   - For this, each JSON message was validated using the JSON schema:



```
json_schema = {
    "type": "object",
    "properties": {
        "ride_id": {"type": "string"},
        "point_idx": {"type": "number"},
        "latitude": {"type": "number"},
        "longitude": {"type": "number"},
        "timestamp": {"type": "string"},
        "meter_reading": {"type": "number"},
        "meter_increment": {"type": "number"},
        "ride_status": {"type": "string"},
        "passenger_count": {"type": "number"}}}
```

Figure 4: JSON Schema to validate the taxi data

   - Valid and invalid messages were then stored in separate PCollections. Invalid messages were kept as it is good data engineering practice to save them in their own table (dead-letter queue) so they can be checked and resolved later.

4) Reverse geocoding longitude and latitude:
   - Next Mapbox [32], an API geocoding and mapping service, was used to reverse geocode the longitude and latitude of the valid messages to determine the which neighbourhoods in NYC they corresponded to.

5) Removing no neighbourhood:
   - It was discovered that at times Mapbox was unable to determine the neighbourhood and would return nothing in the JSON response. Messages where this occurred were then filtered out and stored in a PCollection and treated as invalid messages.

6) Windowed neighbourhood count:
   - With the PCollection containing all messages with valid neighbourhoods, the neighbourhoods were then aggregated by windowing into fixed 5-minute windows using Beam's WindowInto function [33] to determine the neighbourhoods with the most pickups.

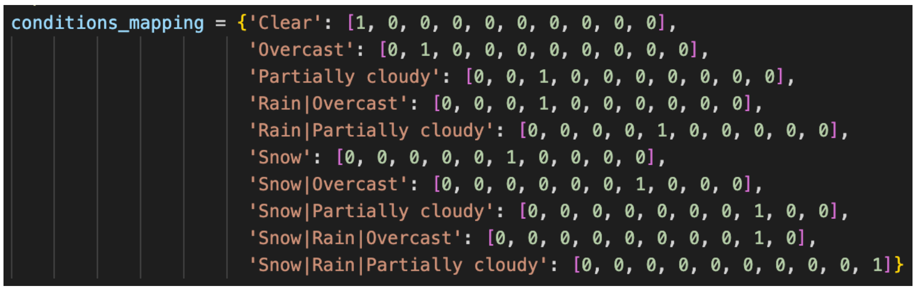
7) Windowed ride status count:
   - Similarly, the PCollection with the filtered messages (by ride status) were also aggregated by windowing into fixed 5-minute windows using Beam's WindowInto function [33] to determine the number of drop-offs and pickups that occurred in the last 5 minutes.

8) Writing to BigQuery tables:
    - Once all the transforms were carried out and individual PCollections were created, the last step was to write the data into their respective BigQuery tables using Beam's WriteToBigQuery function [34]. These tables were:
        1. raw data table (filtered by status only),
        2. windowed neighbourhood count,
        3. windowed ride status count,
        4. messages with no neighbourhood,
        5. and finally, messages with invalid format.

4.5.2. <u>Weather Data:</u>

1) Reading in data and parsing to JSON:
    - Reading the weather data from the Pub/Sub topic and parsing from JSON format to Python dictionary using the load function from Python's json library.

2) Processing the weather data:
    - After reading the weather data, the data needed to be processed for machine learning. Firstly, the day and hour of the timestamp in the weather data were separated into individual columns. For example, 2015-01-01 03:00:00 was separated into date column with value 1 and the time column would have value 3.
    - The weather condition would then be one-hot encoded based on a pre-defined mapping (see figure 5).



```
conditions_mapping = {'Clear': [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                      'Overcast': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                      'Partially cloudy': [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                      'Rain|Overcast': [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                      'Rain|Partially cloudy': [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                      'Snow': [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                      'Snow|Overcast': [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                      'Snow|Partially cloudy': [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                      'Snow|Rain|Overcast': [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                      'Snow|Rain|Partially cloudy': [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]}
```

Figure 5: Mapping for one-hot encoding weather condition

3) Getting total ride per hour prediction:
    - With the pre-processed data, the values for the columns were then extracted into a numpy array using the array function from the numpy library [35].
    - Using the load function from the pickle library [24] the Random Forest model was then loaded and a prediction for the total rides per hour was generated using the pre-processed weather data.

4) Writing to BigQuery:
    - The last step was to write the data into BigQuery tables which were:
        1. raw data table (weather data straight from the Pub/Sub topic),
        2. and total ride predictions table.

4.6. <u>Architecture diagram of the end-to-end system</u>

The architecture diagram of the end-to-end system was created using Lucidchart which is a web-based software system that allows flowcharts and diagrams to be created [36]. The diagram displayed how each of the tools and services implemented in the prototype solution were linked and helped visualise the flow of data through the system.

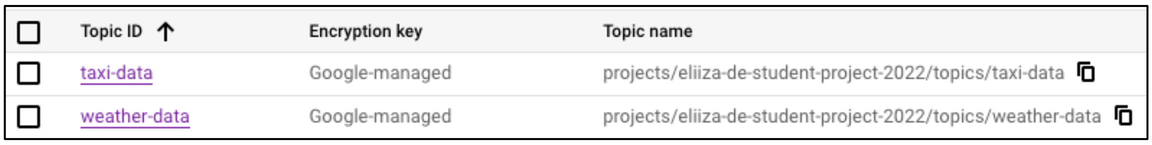
4.7. Dashboard to visualise insights generated

Deliverable 6 was to create visualisations to display the insights that were discovered and generated in BigQuery on a dashboard. Initially, the plan was to utilise Looker on GCP but due to permissions and access issues, an alternative service was considered. In the final solution, Tableau, an analytics platform for exploring and managing data [37], was used due to ease of access and availability of all features. Using Tableau, visualisations of insights generated from the processed data were displayed on a dashboard. Additionally, predictions generated from the machine learning models were also displayed on the dashboard to enrich the insights displayed. To generate these visualisations, the transformed data stored in BigQuery were retrieved from the different tables and appropriate joins were used to generate the final dataset. The dashboard was setup in a way that it would automatically update all visualisations whenever new data from the pipeline was stored in BigQuery, and in turn, be pushed to Tableau. On the dashboard, information such as the weather at the given hour, trips per minute, fare generated per minute, ML predictions (estimated fare per ride, total trips, and payment type), and a heatmap showing notable locations around the city with pickups were displayed.

5) Results

5.1. Data sourcing and data from 2 sources incorporated

With the Pub/Sub topics for the weather and taxi data set up, the topics could be observed in the list of topics in the project environment and a new Dataflow job was created to stream the taxi data from the public topic to the topic in the project environment.



Topic ID	Encryption key	Topic name
taxi-data	Google-managed	projects/eliiza-de-student-project-2022/topics/taxi-data
weather-data	Google-managed	projects/eliiza-de-student-project-2022/topics/weather-data

Figure 6: Taxi and weather data Pub/Sub topics

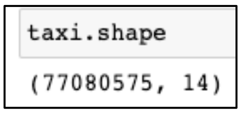
ps-to-ps-taxi-data Streaming

Figure 7: Pub/Sub to Pub/Sub Dataflow job

A key thing to note was that it was discovered that the amount of data being published to the taxi Pub/Sub topic would fluctuate. This led to increased resources (number of workers) being utilised by the service as auto-scaling was enabled which resulted in increased usage fees. Therefore, to prevent unnecessary costs in the future, further optimisation of the Pub/Sub topic configurations would be required to limit the number of workers that can be used. In addition, the filtering of messages could also be done within Pub/Sub to further reduce the amount of data that would flow through the topics which would further reduce costs. In terms of the format of the data, future works could include sourcing more complete weather data so data for different parts of New York City (with intervals shorter than an hour) can be integrated into the solution rather than just assuming the weather was uniform and based on the Central Park weather readings.

5.2. Data analysis to check data quality

Before removing any of the rows, there were 77,080,575 rows, and 14 columns present in the dataset.

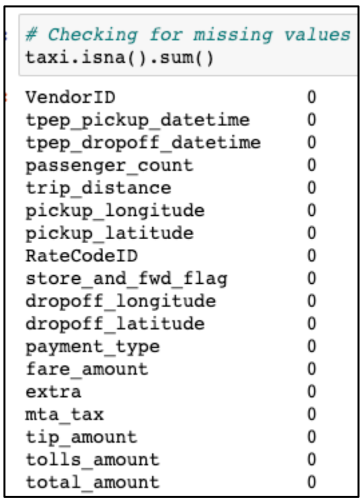


```
taxi.shape

(77080575, 14)
```

Figure 8: Dataset structure before quality check

When checking for missing values, it was found that the dataset was complete, and no missing values were present in any of the columns therefore no extra steps were required to handle missing values.

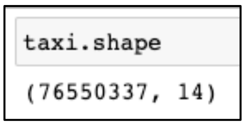


```
# Checking for missing values
taxi.isna().sum()

VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
pickup_longitude         0
pickup_latitude          0
RateCodeID               0
store_and_fwd_flag       0
dropoff_longitude        0
dropoff_latitude         0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
total_amount             0
```

Figure 9: Checking for missing values

After processing and removing the rows with conditions outlined in the methodology, 76,550 ,337 remained.



```
taxi.shape

(76550337, 14)
```

Figure 10: Dataset structure after removing rows

5.3. Actionable insights by incorporating ML models

5.3.1. Fare Per Ride

Model Type	Model	Metric	Value
Baseline	Linear Regression	R^2	0.32
Advanced	XG Boost Regression		0.72

Table 1: Results for models predicting fare per ride

The linear regression model was evaluated on the trained data, the r2_score value obtained was 0.32, which was not good and could not be considered for deployment. After training and evaluating the XGboost with default hyper parameters, the resultant r2_score was 0.65 which was a huge improvement from the baseline model. Performance was improved further by performing hyper parameter tuning by setting the values of learning rate and maximum tree depth to 0.075 and 8 respectively which resulted in an r2_score value of 0.72. Due to the nature and limited set of features and amount of data, the performance of liner regression model was not good. But a gradient-boosted decision tree model, XG Boost regression model was good hence, was used in the final solution.

5.3.2. Payment Type

Model Type	Model	Metric	Value
Baseline	Logistic Regression	Accuracy	49%
Advanced	XG Boost Classifier		51%

Table 2: Results for models predicting payment type

Data is trained on the Logistic regression model and the resultant accuracy score of this trained classification model is 49%. The poor accuracy can be justified on the limited availability of features in the data and various values in the target feature. Although the XG Boost classifier gave an improved result, t the model takes a longer than usual times to train and perform prediction on the data. So, the Logistic regression is taken, as it's balanced in terms of computational speed and performance.

5.3.3. Total Rides Per Hour

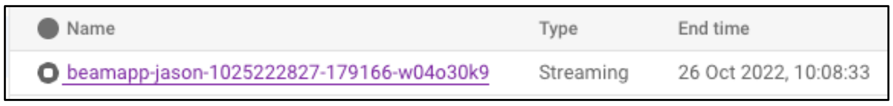
Model Type	Model	Metric	Value
Baseline	Decision Tree	R^2	0.60
	Linear Regression		0.42
	K-Neighbours Regressor		0.33
Advanced	Random Forest Regressor		0.65

Table 3: Baseline and advanced model metrics

From the results above, the Random Forest regressor gave the best results with an R^2 score of 0.65, hence this was selected and integrated into the end-to-end system. Although it had the highest R^2 value, this value was still lower than expected. A possible reason for this was that the model was unable to fully account for the temporal changes in data. To account for this, for future works, better performance could be achieved by segmenting the data into different periods (e.g., seasons) and train separate models instead of using a single model. Doing so would allow the models to be specialised and different models can be used to generate predictions at different times of the year.

5.4. Data pipeline on GCP

Once the pipeline was executed using Dataflow, a new Dataflow job was created, and the job graph was generated after about a minute.



Name	Type	End time
beamapp-jason-1025222827-179166-w04o30k9	Streaming	26 Oct 2022, 10:08:33

Figure 11: Dataflow job for the pipeline

With Dataflow, a similar problem was encountered as with Pub/Sub, where the number of workers used by the Dataflow job would auto-scale with the amount of data flowing through the pipeline which led to unexpectedly high usage fees. Again, for future works, further configuration of the Dataflow job options is required to limit the number of workers used.

5.5. Transformations to process data, and for robust pipeline

5.5.1. Taxi Data

Once the Dataflow job for the pipeline was created, a job graph was then automatically generated to display the flow of data through the pipeline – from reading in the data from Pub/Sub, carrying out transformations, and then writing the transformed data to different BigQuery Tables.

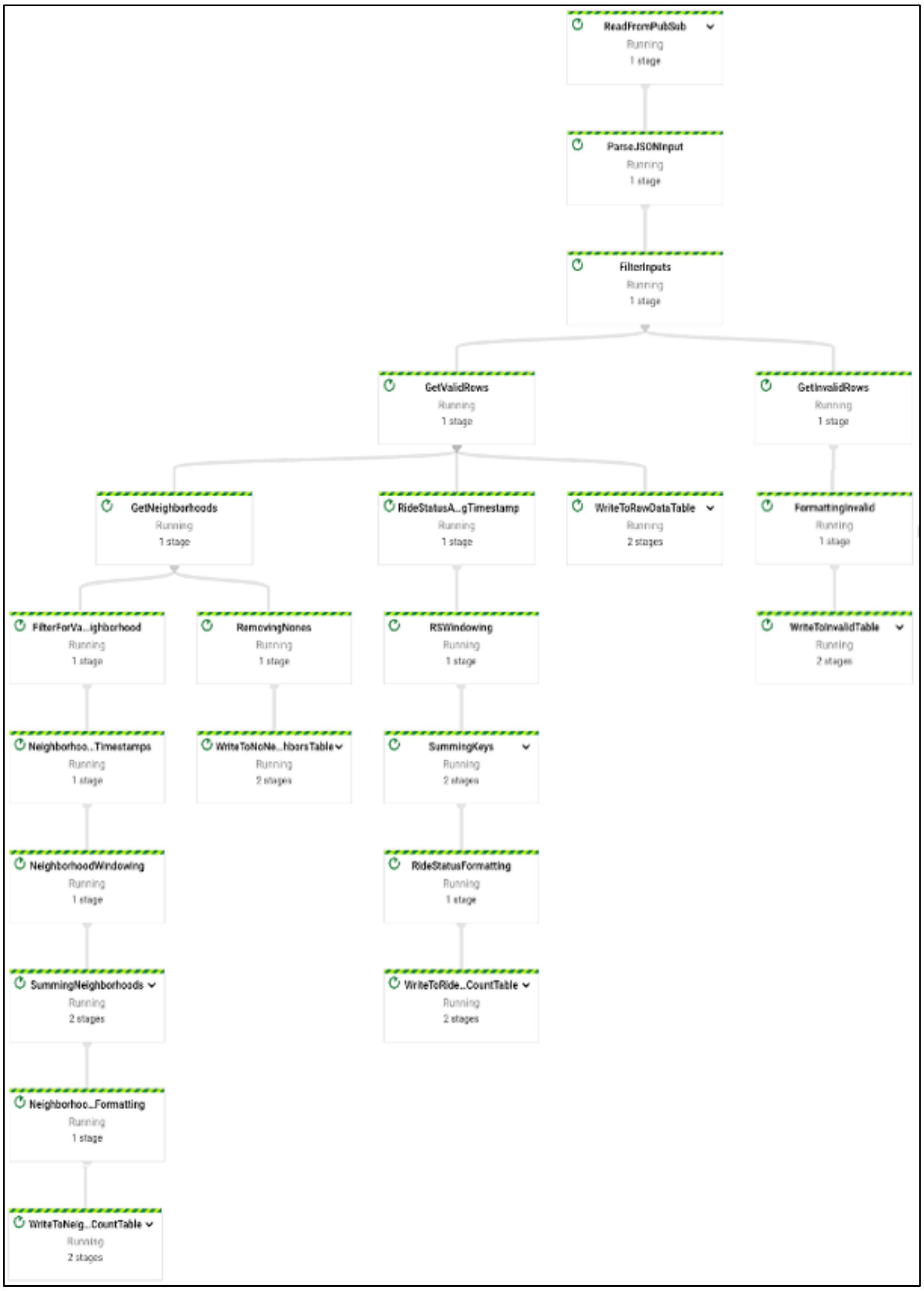


Figure 12: Data pipeline execution graph (Part 1)

5.5.2 Weather Data

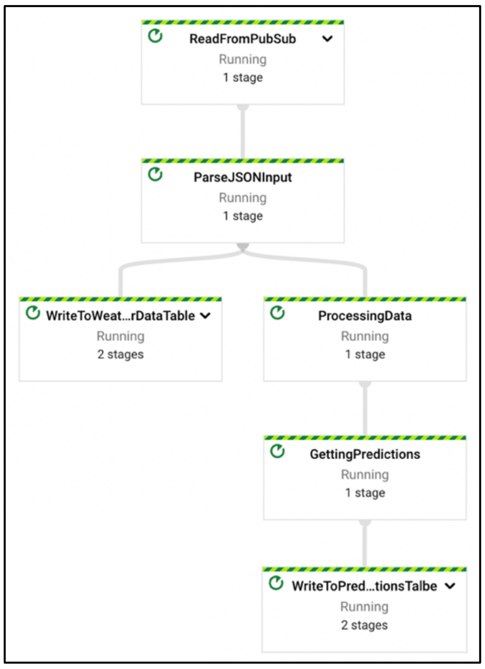


Figure 13: Dataflow execution graph (Part 2)

It is important to note, due to difficulty in sourcing and having integrated the weather data much later into the project, very few transformations were implemented. Additionally, given the time constraints of the project and the format of the data (hourly), it was decided that it was best to write the weather data to a BigQuery table and use it as a lookup table, i.e., determine the weather conditions at a given hour and display this information on the dashboard. In the future, if more complete data with more entries (shorter intervals between readings) was available, more transforms can be implemented such as using joins in the pipeline to connect both taxi and weather data.

5.6. Architecture diagram of the end-to-end system

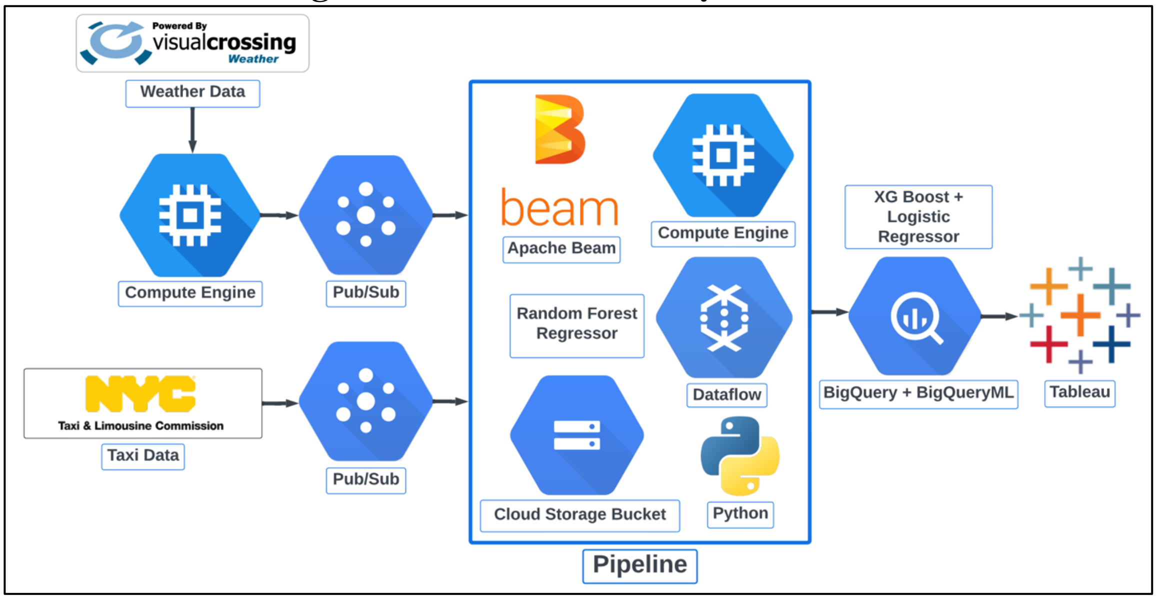


Figure 14: Architecture diagram of the end-to-end system

The architecture diagram in Figure 14 illustrates the flow of data through the system and how each service used in the end-to-end system is linked. Using 2 different Pub/Sub topics, both weather and taxi data were passed to the pipeline. The pipeline would then transform the data which would then store the data in BigQuery tables. The data would then be queried for insights in BigQuery and the insights would then be displayed on an interactive dashboard built using Tableau.

For the machine learning models, an XG Boost Regression model and a Logistic Regression model were used to predict fair per ride and payment type used, respectively. These models were integrated into the system using BigQueryML. A Random Forest Regressor was also used to predict the total number of rides (pickups) per hour and was directly integrated into the pipeline by saving the model as a pickle file and loading up the model in the pipeline.

5.7. Dashboard to display insights generated

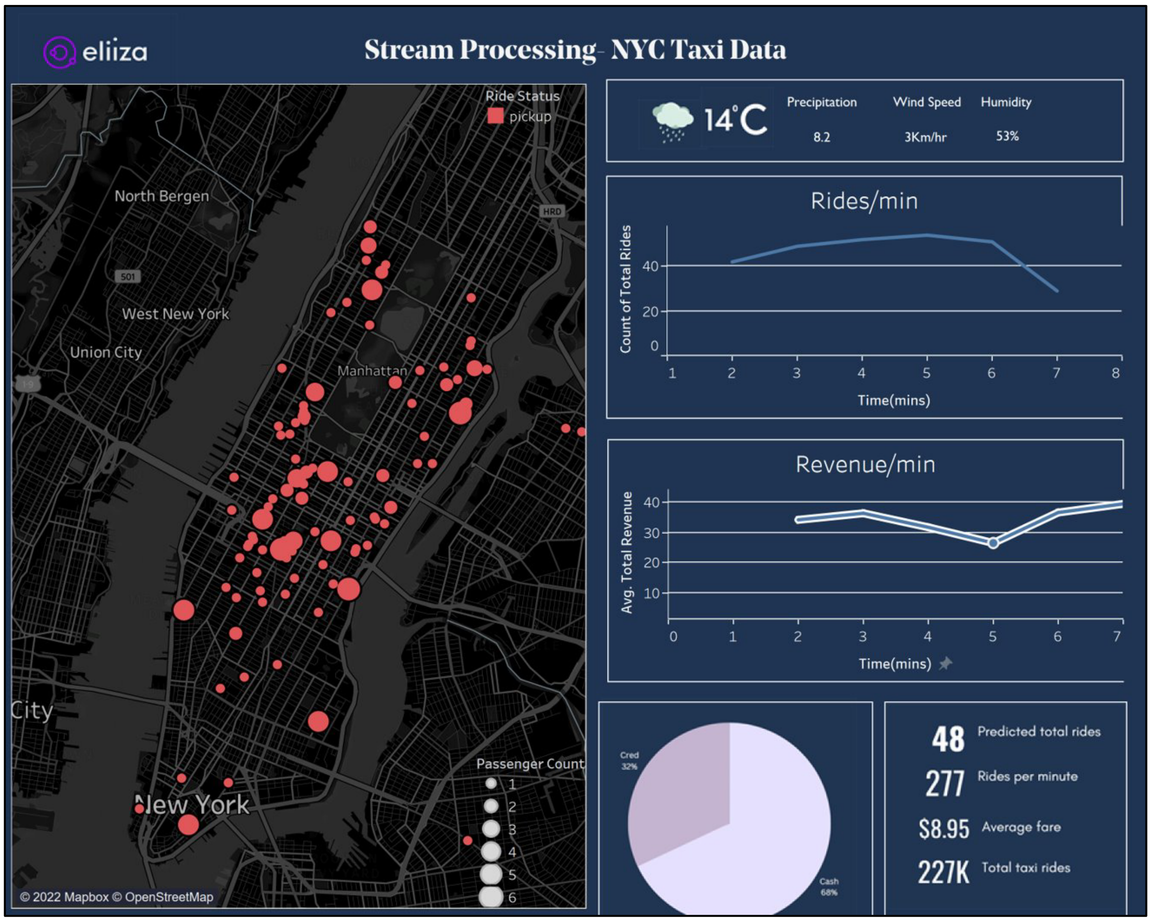


Figure 15: Dashboard displaying insights built using Tableau

The figure above illustrates the final dashboard that was generated using Tableau. The dashboard displays a map of NYC that highlights prominent locations in NYC with lots of pickups as well as the number of passengers that were picked (as indicated by the circle sizes). The real-time weather information is also displayed which shows details such as the condition, temperature, windspeed, and humidity. The dashboard also contains two line charts which track the number of rides per minute and the average fare per ride (revenue) of these trips. Additionally, a pie chart illustrates the breakdown of payment types that have been used. Lastly, the predictions generated from the machine learning models are also displayed to further enrich the dashboard.

When generating the visualisations, it was discovered that querying the data in intervals longer than seven minutes took a very long time to do with BigQuery which led to long wait times for the dashboard to update (30 minutes or more). Moreover, the size of the raw data table in the BigQuery (used for the heatmap) was greater than 2 terabytes, and it was using a significant amount of querying quota and was very costly to execute queries. Therefore, for future works, query optimisation is required to reduce resource consumption and querying costs. This could be done by using tables that have been partitioned by date and time which would reduce the size of the tables hence minimising the query quota usage which would then reduce the cost of using BigQuery. Smaller tables also mean less data that needs to be pushed to Tableau meaning it would take less time to update the visualisations on the dashboard.

6) Conclusion

6.1. Project Summary

The aim of this project was to create an AI based traffic management dashboard on GCP. Partnering with Eliiza, an end-to-end system was designed, implemented, and deployed using a combination of GCP services and 3rd party programs. At the start of the project, 7 deliverables were derived based on the project's aim and all 7 were accomplished. Deliverable 1 was accomplished by incorporating 2 sources of data into the solution which were the 2015 NYC TLC Yellow Taxi data and the 2015 NYC weather data, both of which were fed into the pipeline using Pub/Sub topics. For deliverable 2, after conducting an exploratory data analysis, insights were generated and relationships between features were discovered through creating visualisations. Deliverable 3 was accomplished through the incorporation of three machine learning models in the solution. After building, training, and evaluating a range of models that varied in complexity, an XG Boost regression model, logistic regression model, and random forest regression model were chosen to predict fare per ride, payment type used, and total rides (pickups) per hour, respectively. For the XG Boost and logistic regression models, these were implemented using BigQueryML whilst the random forest regressor was integrated directly into the pipeline by saving the model as a pickle file and loading it up as needed. For deliverable 4, a data pipeline was built using Apache Beam's Python SDK and was deployed to GCP by executing the pipeline using Dataflow. Deliverable 5 was met by defining functions in the pipeline and utilising Beam functions to carry out the transformations on the stream of data. Additionally, functions were implemented to check the validity of incoming messages and a BigQuery table was used to store invalid messages which allowed the pipeline to be tolerant to faults (errors due to invalid format). Deliverable 6 was accomplished through the design of an architecture diagram of the end-to-end system using Lucidchart which helped to visualise the flow of data through the system. Finally, deliverable No. 7 was met by querying the processed data using SQL queries in BigQuery and then displaying these insights using visualisations on a dashboard created using Tableau.

6.2. Future Directions

For future directions, it would be good to source more complete weather data to get data that was measured in shorter intervals rather than on an hourly basis. As for the performance of our machine learning models, better performance could be achieved when predicting the total rides per hour by segmenting the data into different time periods and using separate models for each segment. In terms of minimising costs, optimising configurations for services such as Pub/Sub and Dataflow should be carried out to limit the number of workers that can be used and to ensure the services do not auto scale. In addition, filtering of messages can be done in Pub/Sub to minimise the amount of data flowing through the pipeline. Lastly, to further reduce costs, query optimisation is required to minimise the quota usage in BigQuery and table partitioning should be carried out to reduce the amount of data the needs to be queried.

7) Appendix

7.1. Roles and Responsibilities

Member	Role	Responsibilities
Jason	Team Leader/ Data Engineer	1) Coordinating the biweekly sprint planning and division of tasks. 2) Keeping track of the project progress and the progress of each member's weekly tasks. 3) Ensuring the completion of assigned weekly tasks. 4) Build and execute the pipeline with Apache Beam and Dataflow (deliverable 1). 5) Conduct initial exploratory data analysis on the NYC Taxi dataset (deliverable 2). 6) Testing of the end-to-end system and deployment to GCP. 7) Generate the architecture diagram of the end-to-end system (deliverable 6). 8) Ensuring the pipeline is tolerant to faults and errors (deliverable 5). 9) Build and evaluate machine learning models to enrich insights (deliverable 3). 10) Source 2 datasets and incorporate them into the end-to-end system (deliverable 1).
Arjun	Data Engineer	1) Taking of meeting minutes for the twice weekly team standups 2) Taking of meeting minutes for the twice weekly catchups with Eliiza. 3) Taking of meeting minutes for the weekly standups with Tomas. 4) Ensuring the completion of assigned weekly tasks. 5) Conduct initial exploratory data analysis on the NYC Taxi dataset. (deliverable 2) 6) Writing and testing SQL queries to generate insights from processed data using BigQuery (deliverable 7) 7) Generate a dashboard to display insights generated (deliverable 6). 8) Build and evaluate machine learning models to enrich insights (deliverable 3)
Sai	Data Engineer	1) Arranging the weekly standups with Tomas. 2) Ensuring the completion of assigned weekly tasks. 3) Conduct initial exploratory data analysis on the NYC Taxi dataset. 4) Designing dashboard layout (deliverable 7). 5) Generating visualisations on the dashboard to help visualise insights (deliverable 7). 6) Develop SQL queries to generate insights from the processed data (deliverable 7) 7) Build and evaluate machine learning models to enrich insights (deliverable 3)

7.2. Self-Reflections

7.2.1 Jason

This project has been one of the most enriching projects that I have ever undertaken at university. Not only have I learnt a great deal about data engineering technologies and practices, but also about applying data engineering and data science techniques into a real-world scenario which is something I found lacking in the Master of Data Science. In terms of technologies that I learnt in this project that I didn't know before, I learnt about what a data engineer does and the different technologies which they utilise to transform data and the role that they play in a data science team – which is to take data from different sources and carry out transformations using a data pipeline to make the data available for data scientist and data analyst. A key thing I learnt was how to build a data pipeline with Apache Beam which required a very different style of coding compared to what I am used to - with the different pipes for applying transformations and creating PCollections to branch the pipeline. More importantly, I learnt how to deploy and execute a data pipeline using cloud services such as Dataflow and the different environment variables that needed to be configured (e.g., credentials) to make execution possible. Through this project, I also learnt a lot about how to use different GCP services and the role they play in a data engineering system – GCP is something that I have never used prior to this project. I also learnt about the importance of having a pipeline that is robust and fault tolerant. When building the pipeline, I discovered that the pipeline would fail and stop working the moment a single message with the wrong format was passed through the pipeline. Hence, format checks are an important step to implement in pipelines to make them fault tolerant. Lastly, I learnt about best data science practices such as utilising dead-letter queues (sinks for errors) to prevent data loss and to allow errors to be investigated so they can be prevented from occurring again.

For project management, I learnt the importance of using a tool such as Jira to help break down a project into smaller manageable and measurable tasks to help make progress tracking easier. The importance of good communication with the project team, owner, and manager were also things that I learnt. It is important to have constant communication with project owners to ensure that goals and aims are aligned throughout the duration of a project to prevent scope creep. Additionally, communicating any blockers as soon as possible so they can be resolved to minimising the risk of delaying a project.

Moving on to the things that worked well. The implementation of the data pipeline with Apache Beam and Dataflow was very successful and I managed to get a fully functional pipeline with lots of transforms deployed on GCP. Combing the Apache Beam documentation, I learnt how to create branches which allowed different segments of the data stream to be processed in parallel – this was something that really help me understand how to build a pipeline with Beam. Creating an emulator program to stream the weather data in "real-time" was also very successful. Given that no public Pub/Sub topic was available, this had to be done as part of our project and I managed to set up our own topic and published the data to the topic from a program I created on my own.

As for things that didn't work well, I personally felt that teamwork and communication with my team was the biggest challenge in this project. Trying to coordinate tasks was difficult as it was hard for multiple people to work on the same pipeline code at the same time meaning I bore the sole responsibility to build and deploy the pipeline. Trusting others to uphold the same standard of work as me was difficult as well and I felt that I should have been more involved in other aspects of the project to ensure my standard of work was upheld (e.g., developing the dashboard).

As for improvements in the future, the first thing that I would do would be to figure out how to test deploying the pipeline to Dataflow locally (before deploying to GCP). Doing so would have sped up the deployment and testing and significantly reduce the incurred cost of this project. But unfortunately, there was very little documentation online to show how to do this (and the ones that I did find did not explain it very well). Additionally, it would have been good to configure the settings for services such as Dataflow, BigQuery, and Pub/Sub to limit the number of workers which would have prevented the unexpected costs incurred. I felt this was the most important lesson for future works so that no unexpectedly large bills are received when working on a project in industry. Lastly, I would have liked to have more clarity from our project partners in terms of more detailed and specific deliverables. Given that this project was very open ended, very high-level aims were given to us which left a lot of ambiguity with regards to how to implement our solution.

Overall, there were a lot of important lessons that I learnt during this project. I feel that having completed this project, I have gained a better understanding of what it means to be a data engineer and how to apply what I have learned in university into a real-world project. This project has opened my perception of what it is like to work in tech, specifically in data, and has provided a potential career path which I can pursue in the future.

7.2.2 Arjun

We started our project by diving into GCP and obtaining a certification. Dataflow and Apache beam were alien concepts until this project. The certification was insightful, thorough, and more importantly, relevant. I believe each of us are well equipped to deal with streaming pipelines and data. Working on a project with data engineers in the field was even more eye opening, unlike other assignments we had to take into consideration resource consumption, deadlines on prototype, overhead and scaling. Working as part of a team and incorporating everyone's ideas and bringing the best out was remarkable. We were able to pull through with all deliverables that were promised and more. But, it involved a lot of effort, trial and error like all projects do. Knowing what I know now, if I were to start all over, I would make so many changes.

Firstly, starting with our pipeline. Our pipeline was initially set to auto scale and as expected when the size of data multiplied the pipeline allocated more workers this drastically increased our resource consumption. Next time I would just cap the number of workers this way we can be assured that the pipeline works optimally while at the same time we don’t over consume our allotted resource quota. Coming to our data, partitioning and organizing tables by date would have done wonders to our entire development cycle. We could have saved so much time since, this way we would not be ingesting so much data at the same time.

Next, querying data on BigQuery while effective took a whole chunk of daily quota available to us. In reality we did not require all of it, Our aim was to build a real time dashboard so if I were to start all over I would find a way to release data older than 10 mins and perform our required aggregations on the latest data we obtained from the pipeline. This further affected Visualizations and the heatmap on Tableau took exceedingly and unnecessarily long because of the poor organization of our data.

Finally getting to the visualization. Tableau was not our initial choice we were intending to use looker for our visualizations. Looker enables us to perform complex data manipulation. The initial plan was to take advantage of looker's cluster analysis based on K-means to find noteworthy locations New York City to find passengers. Unfortunately, we were unable to gain access. Tableau comes with its fair share of advantages. I was able to link Tableau and BigQuery with ease. Then it was only a matter of performing aggregations on BigQuery and visualizing those results on Tableau. After realizing my mistake with portioning the tables, I filtered out all data older than 7mins. This data was then aggregated to retrieve the total trips, pickup locations, preferred payment type, and weather data. Tableau comes with heatmaps created by mapbox. Once the relevant latitude and longitude is fed as input we are given our heatmap of NYC as the result. Looker provides in built time series analysis, tableau does not. Hence, all time series analysis was performed on BigQuery and insights were visualized on Tableau.

7.2.3 Sai

After working on this project, I learnt a lot about streaming data and batch data capabilities, and how we can use various aggregations, apply ML algorithms on the go. From sports, Computer games and AR/VR spaces deal with huge amounts of streaming data, along with accuracy and low error rates, one should give importance to high performance, low latency rates, optimisation of models and efficient use of resources.

Working on this project helped me understand the thought process behind the data engineer and they store it and how the data scientists handle that data in real industry.
Although I worked on AWS during cloud computing course but working on GCP was completely new and a different experience. I had the opportunity to do professional data engineering module, big query and looker extended capabilities module, Big Query ML and developing data pipelines modules etc..

After the initial problem statement, we planned on what type of services to be used from all the available GCP services. So, we concluded to use pub/sub and dataflow for streaming the data, big query to be used as a data warehouse and analytics space and Looker/Tableau as our BI tool to display visualizations.

Before this project, I considered big query as a data warehouse alone, which serves the purpose of storing the data. But once I started work towards this project, I started to understand the various services like spanner, big table etc., which cater different storage needs in real world. Also, how Big Query ML, Vertex AI and Look ML differs, and appropriate scenarios for their usage respectively.

I learnt various aspects of Looker, like its frameworks to deal with latency issues of streaming data, supporting various forms of Geo spatial data, ease to access google maps API, and to preview of data to business analysts to edit relatable SQL queries to extract insights quickly.
As we had looker access issues, I had opportunity to learn and work on Tableau as well. Working on Tableau was straight forward, and it was easy to make aggregations, connecting various tables, applying selection filter on various charts that present on a tableau dashboard.

Public data Availability- As a part of our deliverable, we planned to integrate weather data along with traffic data. We did not foresee the inconvenience that surrounded in obtaining the weather data. If we had prepared well with other meaningful alternative options, we could have cut down on the project timeline.
Cloud services access issues- With any cloud-based services, it is expected to have access issues. When I were to work on a similar project again, I would expect the bottle neck

issues and will have other alternatives readily available, so I can tweak the plan on the go. Utilising all the available data points- we had to leave out "enroute" data points from taxi dataset as it did not cater any use case according to initial project plans. But after addressing some of the questions after our project presentation. We understood an important use case scenario with there "Enroute" points. Based on these "enroute" points we can extract traffic congestion geo location. Also, combining knowledge from Eliiza's Apache kaf-car presentation, GCP google maps API and the enroute data points. We can use these "enroute" datapoint location and times as checkpoints and can use these to simulate an autonomous taxi ride.

Thanks to transformations in the data pipeline, the obtained data is clean and required little to no data pre-processing. But it's impossible to extract data insights directly from raw streamed data. So, I had to perform mapping of a complete taxi ride based on the ride id. This mapped row outcome depicts a complete ride of a taxi based on ride id. After obtaining complete ride observations. It was easy to perform analytical aggregations with the built-in big query functions.
There were many ways to implement Machine Learning in GCP, like Look ML, Vertex AI and Big Query ML. I choose Big Query ML as I was already storing all the mapped data in big query warehouse. So, in terms of accessing the data and scheduling the queries and fetching the data from big tables into ML models is simple and direct in Big Query ML.

Due to the nature of the data, there were limited features so applying apt ML models was a tough task. Initially, I tried linear regression as the model took very less time to process the streamed data. But the model's accuracy was very poor. So, as an alternative I tried random forest regressor, in this regard both the model's computational performance and accuracy were low. So, I went for XGboost regression for one of the ML models. XGboost is suitable in all regards, as it works well with limited set of features, supports distributed/parallel computing when we are deals with huge datasets and accuracy is high compared to all other models.

Interesting problem that I faced in the project is using correct time zones in SQL queries. It is important while dealing with transaction or taxi ride timestamps. My query fetched zero rows when I tried to extract taxi rides happened in the current date. After a while, I figured out the taxi rides timestamp is in NYC time zone, whereas the query is constructed based on Melbourne's time zone. So, it worked after using the appropriate time zone in the SQL query. Based on above mishaps I learnt how to be prepared with alternatives to deal with unforeseen situations. Being flexible to incorporate or remove the functionalities in models.

8) <u>References</u>

[1] Smith, B. (2022, February 24). *Why is there a shortage of taxi drivers?* Patons Insurance. https://www.patonsinsurance.co.uk/2022/02/24/shortage-of-taxi-drivers/

[2] Ong, J. (2022, August 21). *The Big Read in short: What a rapidly shrinking taxi industry means for older drivers, commuters.* TODAY. https://www.todayonline.com/big-read/shrinking-taxi-industry-older-drivers-commuters-1973461

[3] Shanti Menon, Energy Editor. (2022, May 9). *Shanti Menon.* Environmental Defense Fund. https://www.edf.org/article/war-ukraine-driving-gas-prices

[4] City of New York, NYC Open Data. (n.d.). *NYC Open Data.* Retrieved November 4, 2022, from https://opendata.cityofnewyork.us/

[5] Corporation, V. C. (n.d.). *Historical Weather Data & Weather Forecast Data | Visual Crossing.* Retrieved November 4, 2022, from https://www.visualcrossing.com/weather-data

[6] *pandas.read_csv — pandas 1.5.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

[7] *pandas.DataFrame.shape — pandas 0.23.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.shape.html

[8] *pandas.DataFrame.shape — pandas 0.23.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.shape.html

[9] *pandas.DataFrame.isna — pandas 1.5.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isna.html

[10] *pandas.DataFrame.sum — pandas 1.5.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sum.html

[11] *pandas.DataFrame.sum — pandas 1.5.1 documentation.* (n.d.). Retrieved November 4, 2022, from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sum.html

[12] *Using BigQuery ML to predict penguin weight* |. (n.d.). Google Cloud. Retrieved November 4, 2022, from https://cloud.google.com/bigquery-ml/docs/linear-regression-tutorial

[13] *The CREATE MODEL statement for boosted tree models using XGBoost | BigQuery ML* |. (n.d.). Google Cloud. Retrieved November 4, 2022, from https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-boosted-tree

[14] *The CREATE MODEL statement for generalized linear models | BigQuery ML* |. (n.d.). Google Cloud. Retrieved November 4, 2022, from https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-glm

[15] *sklearn.tree.DecisionTreeRegressor.* (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[16] *sklearn.linear_model.LinearRegression.* (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[17] *sklearn.neighbors.KNeighborsRegressor.* (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

[18] *sklearn.ensemble.RandomForestRegressor.* (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[19] Dantas, J. (2021, December 16). *The importance of k-fold cross-validation for model prediction in machine learning*. Medium. https://towardsdatascience.com/the-importance-of-k-fold-cross-validation-for-model-prediction-in-machine-learning-4709d3fed2ef

[20] *sklearn.model_selection.KFold*. (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[21] *sklearn.model_selection.GridSearchCV*. (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[22] Frost, J. (2022, July 22). *How To Interpret R-squared in Regression Analysis*. Statistics by Jim. https://statisticsbyjim.com/regression/interpret-r-squared-regression/

[23] *sklearn.model_selection.cross_val_score*. (n.d.). Scikit-learn. Retrieved November 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[24] *pickle — Python object serialization — Python 3.11.0 documentation*. (n.d.). Retrieved November 4, 2022, from https://docs.python.org/3/library/pickle.html

[25] *Beam Overview*. (n.d.). Retrieved November 4, 2022, from https://beam.apache.org/get-started/beam-overview/

[26] *Apache Beam Python SDK*. (n.d.). Retrieved October 26, 2022, from https://beam.apache.org/documentation/sdks/python/

[27] *venv — Creation of virtual environments — Python 3.11.0 documentation*. (n.d.). Retrieved November 4, 2022, from https://docs.python.org/3/library/venv.html

[28] *Dataflow documentation |*. (n.d.). Google Cloud. Retrieved October 26, 2022, from https://cloud.google.com/dataflow/docs

[29] *json — JSON encoder and decoder — Python 3.11.0 documentation*. (n.d.). Retrieved November 4, 2022, from https://docs.python.org/3/library/json.html

[30] *Filter*. (n.d.). Retrieved November 4, 2022, from https://beam.apache.org/documentation/transforms/python/elementwise/filter/

[31] *jsonschema*. (n.d.). Retrieved November 4, 2022, from https://python-jsonschema.readthedocs.io/en/stable/

[32] *Web applications | Help*. (n.d.). Mapbox. Retrieved November 4, 2022, from https://docs.mapbox.com/help/getting-started/web-apps/

[33] *WindowInto*. (n.d.). Retrieved November 4, 2022, from https://beam.apache.org/documentation/transforms/python/other/windowinto/

[34] *apache_beam.io.gcp.bigquery module — Apache Beam documentation*. (n.d.-b). Retrieved November 4, 2022, from https://beam.apache.org/releases/pydoc/2.9.0/apache_beam.io.gcp.bigquery.html

[35] *numpy.array — NumPy v1.23 Manual.* (n.d.). Retrieved November 4, 2022, from https://numpy.org/doc/stable/reference/generated/numpy.array.html

[36] Lucid. (n.d.). *Intelligent Diagramming.* Lucidchart. Retrieved November 4, 2022, from https://www.lucidchart.com/pages/landing?utm_source=google

[37] *What is Tableau?* (n.d.). Tableau. Retrieved November 4, 2022, from https://www.tableau.com/why-tableau/what-is-tableau