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.

1. **Project Scopes**
   1. **Aims**
2. End-to-end system capable of delivering insights from multiple data sources in real-time.
3. Solution on Google Cloud Platform (GCP) with best data engineering practices.
   1. **Deliverables**
4. Data from 2 sources incorporated.
5. Insights into taxi data through exploratory data analysis.
6. Actionable insights by incorporating machine learning models.
7. Data pipeline on GCP.
8. Transformations to process and transform data, and to make pipeline robust.
9. Architecture diagram of the end-to-end system.
10. Dashboard to visualise insights generated.
11. **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.

1. **Methodology**
   1. **Deliverable 1: Data and Sources**

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 (reference). The dataset contained every trip that was completed between January and June 2015 and had 146,087,462 entries. 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. However, for the purpose of building the machine learning models, the complete dataset from the NYC TLC’s website was used as more features were available. (WHY?) - SAI

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 (reference) and contained hourly weather for NYC (Central Park weather station) for 2015 and had 4,344 entries. 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.

* 1. **Deliverable 2: Exploratory Data Analysis (EDA)**

For deliverable 2, an initial exploratory data analysis (EDA) was conducted to gain familiarity and insights into the taxi data. Given the immense size of the complete dataset, random sampling was used to extract 10,000 entries using the sample function from Python’s built in random library (reference). Once sampled, visualisations were then generated to explore any possible relationships between the attributes in the data. Jupyter Notebooks was used as an environment for the EDA and visualisations such as scatter plots and bar gaphs were generated using plotly, pandas, and matplotlib libraries (reference).

* 1. **Deliverable 3: Machine Learning**
  2. **Deliverable 4: Data Pipeline**

The data pipeline was 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 (reference). Using the apache-beam library from the Python SDK for Apache Beam (reference), functions were defined in the pipeline to transform data in the pipeline using a 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 (reference). 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 (reference). 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 is Dataflow). Then to execute the pipeline on Dataflow, a command was run (see Figure X) using the SSH terminal in the Compute Engine VM which then automatically created a Dataflow job to run the pipeline.

A picture containing text

Description automatically generated

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

* 1. **Deliverable 5: Pipeline Transformations**

The following functions and transformations were implemented:

1. Reading in data and parsing JSON format:
   * Firstly, the data was pulled from the Pub/Sub topics into the pipeline and were parsed (i.e., converted) from JSON format into Python dictionaries using the Python json library (reference).
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 (reference) 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 (reference) 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:

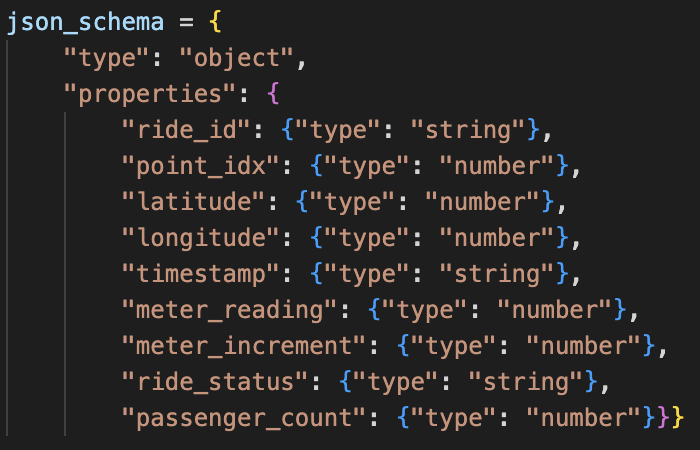


Figure X: JSON Schema to validate the taxi data

* 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.

1. Reverse geocoding longitude and latitude:
   * Next Mapbox (reference), 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.
2. 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.
3. 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 (reference) to determine the neighbourhoods with the most pickups.
4. 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 (reference) to determine the number of drop-offs and pickups that occurred in the last 5 minutes.
5. 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 (reference). These tables were the:
     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 – Due to time constraints and for the sake of simplicity, all attributes were converted to strings.
   1. **Deliverable 6: Architecture Diagram**

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 (reference). 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.

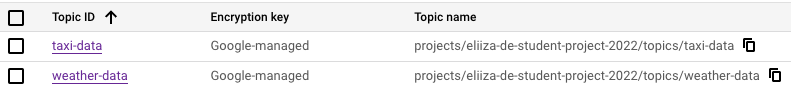
* 1. **Deliverable 7: Dashboard** – ARJUN – Leave references for me to do (Jason)

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 was used due to ease of access and availability of all features (WHAT IS TABLEAU) + (reference). 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 table 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. (WHAT VISUALISATIONS)

1. **Results**

**Need paragraph on quality assurance for each deliverable – Relate to acceptance criteria**

* 1. **Deliverable 1: Data and Sources**

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.

**Figure 3:** Taxi and weather data Pub/Sub topics



**Figure 4:** Pub/Sub to Pub/Sub Dataflow job

* 1. **Deliverable 2: Exploratory Data Analysis (EDA)**
  2. **Deliverable 3: Machine Learning Models** - SAI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prediction** | **Model Type** | **Model** | **Metric** | **Value** |
| Fare Per Ride | Baseline | Linear Regression |  | 0.32 |
| Advanced | XG Boost Regression | 0.72 |
| Payment Type | Baseline | Logistic Regression | Accuracy | 49% |
| Advanced | XG Boost Classifier | 49% |
| Total Rides Per Hour | Baseline | Decision Tree |  | 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, for nearly all predictions the advanced models had better performance that the baseline models except when predicting payment type. A possible reason for the poor performance in the XG Boost Classifier could be due the limited availability of features in the data. It is also important to note that the XG Boost Classifier took a significantly longer time to train (compared to the Logistic Regression model) and did not produce better accuracy. Although the Random Forest Regressor had the highest value when predicting the total rides per hour, 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. Therefore, given the results obtained, XG Boost Regression, Logistic Regression, and Random Forest Regressor were selected and integrated into the end-to-end system.

* 1. **Deliverable 4: Data Pipeline**

Once the pipeline was executed using Dataflow, a new Dataflow job was created, and the job graph was generated after about a minute **(see Appendix 7.5)**.



**Figure 5:** Dataflow job for the pipeline

* 1. **Deliverable 5: Pipeline Transformations**

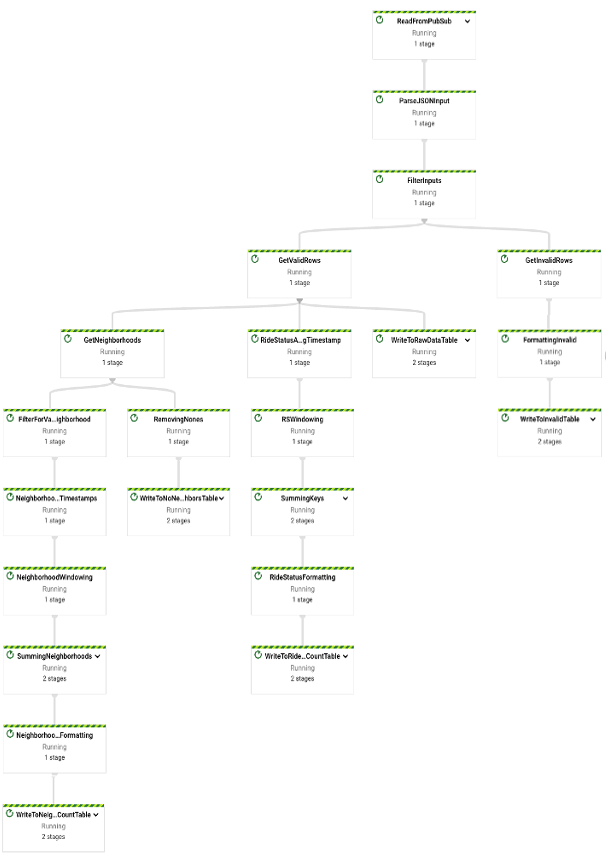
****Once the Dataflow job for the pipeline was created, a job graph was then automatically generated to display the flow of data through the pipeline – from reading in the data from Pub/Sub, carrying out transformations, and then writing the transformed data to different BigQuery Tables.

Figure X: Data pipeline execution graph

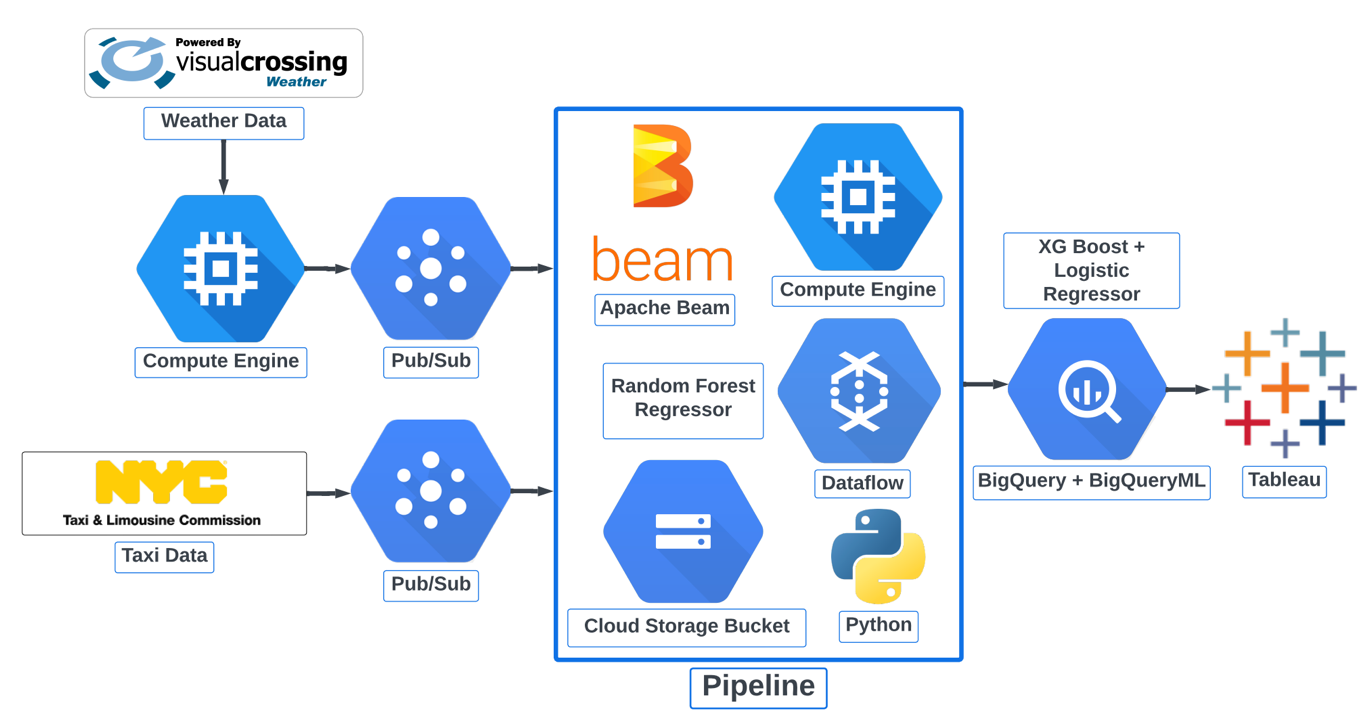
* 1. **Deliverable 6: Architecture Diagram**

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

Deliverable No. 4 was to generate an architecture diagram of the end-to-end system to visualise the flow of data, which is illustrated in the figure above. 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.

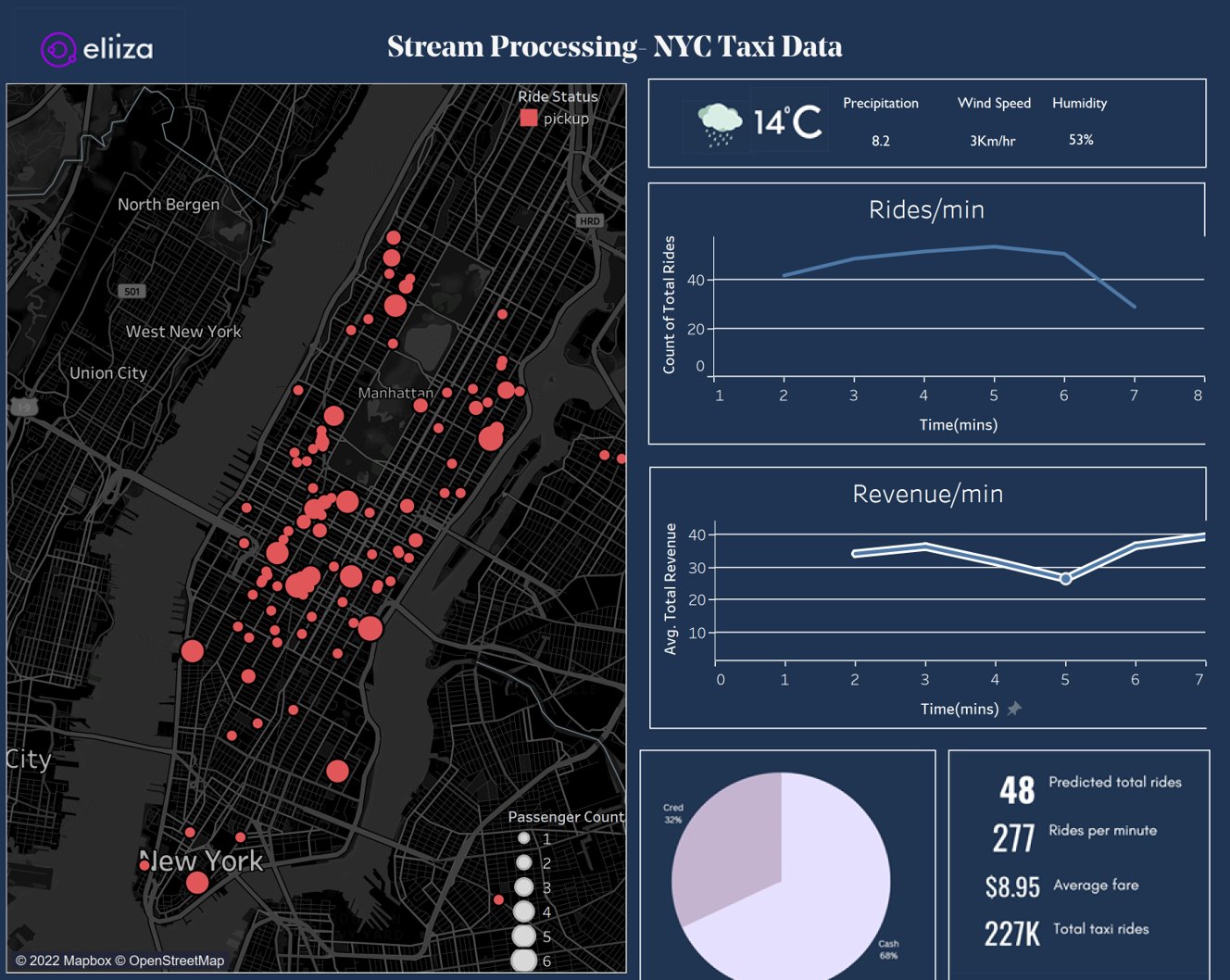
* 1. **Deliverable 7: Dashboard** -ARJUN

Figure X: 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 2 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.

1. **Conclusion**
   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.

* 1. **Future Directions**

1. **Appendix**
   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 No. 1) 5. Conduct initial exploratory data analysis on the NYC Taxi dataset. 6. Testing of the end-to-end system and deployment to GCP. 7. Generate the architecture diagram of the end-to-end system (deliverable No. 4) 8. Ensuring the pipeline is tolerant to faults and errors (deliverable No. 5) 9. Build and evaluate machine learning models to enrich insights (deliverable No. 3) 10. Source 2 datasets and incorporate them into the end-to-end system (deliverable No. 2) |
| 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. 6. Writing and testing SQL queries to generate insights from processed data using BigQuery (deliverable No. 6) 7. Generate a dashboard to display insights generated (deliverable No. 6). 8. Build and evaluate machine learning models to enrich insights (deliverable No. 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 No. 6). 5. Generating visualisations on the dashboard to help visualise insights (deliverable No. 6). 6. Develop SQL queries to generate insights from the processed data (deliverable No. 2) 7. Build and evaluate machine learning models to enrich insights (deliverable No. 3) |

* 1. **Self Reflections**
     1. **Jason**
     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, partioning 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.

* + 1. **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.

1. **References**

Compute Engine: <https://cloud.google.com/compute>

venv: <https://docs.python.org/3/library/venv.html>

lucidchart: <https://www.lucidchart.com/pages/landing?utm_source=google&utm_medium=cpc&utm_campaign=_chart_en_tier1_mixed_search_brand_exact_&km_CPC_CampaignId=1490375427&km_CPC_AdGroupID=55688909257&km_CPC_Keyword=lucidchart&km_CPC_MatchType=e&km_CPC_ExtensionID=&km_CPC_Network=g&km_CPC_AdPosition=&km_CPC_Creative=442433236001&km_CPC_TargetID=aud-1660700197372:kwd-33511936169&km_CPC_Country=9071341&km_CPC_Device=c&km_CPC_placement=&km_CPC_target=>