ETL Pipeline of the NYC Taxi Trip Data for January 2023.

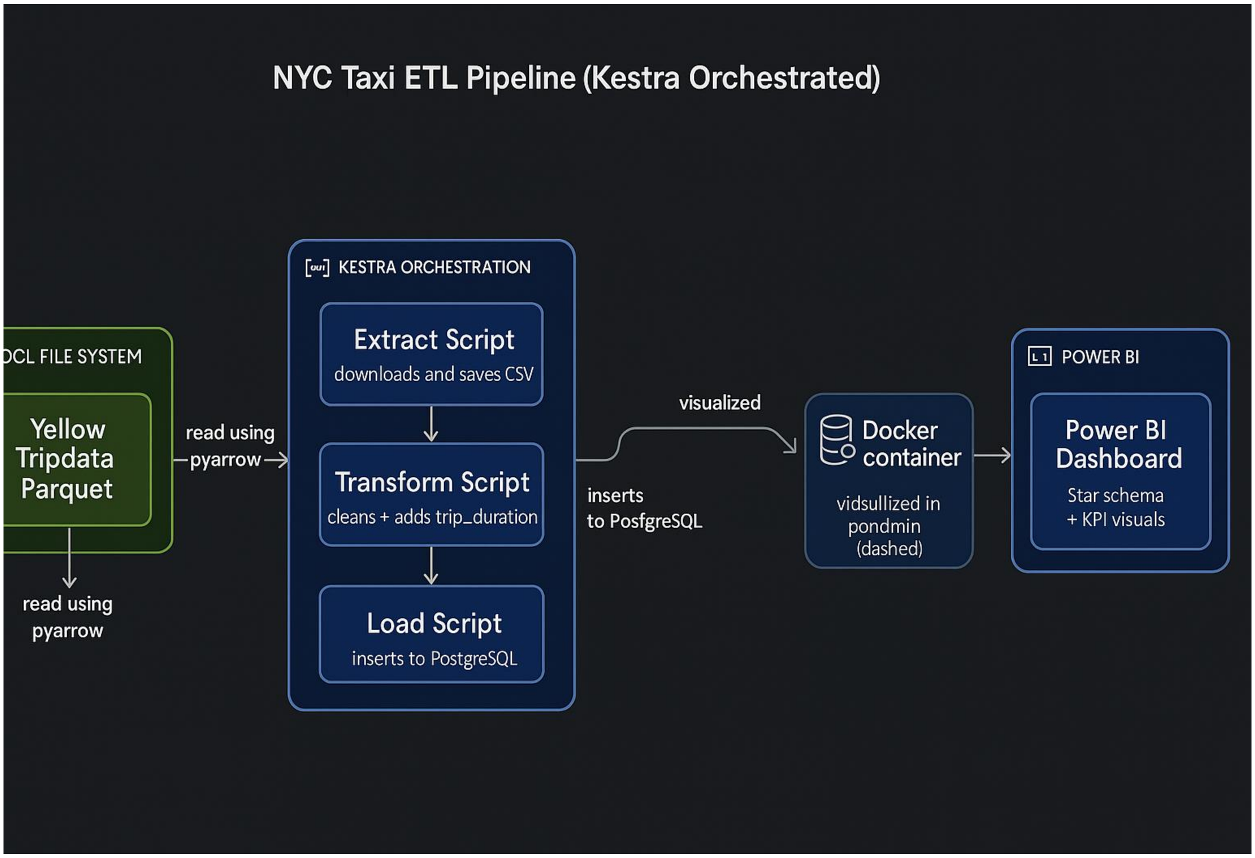
Data Engineering CIS 660-01 by Professor Sorio Boit

Sharon Mungania

Winter 2025

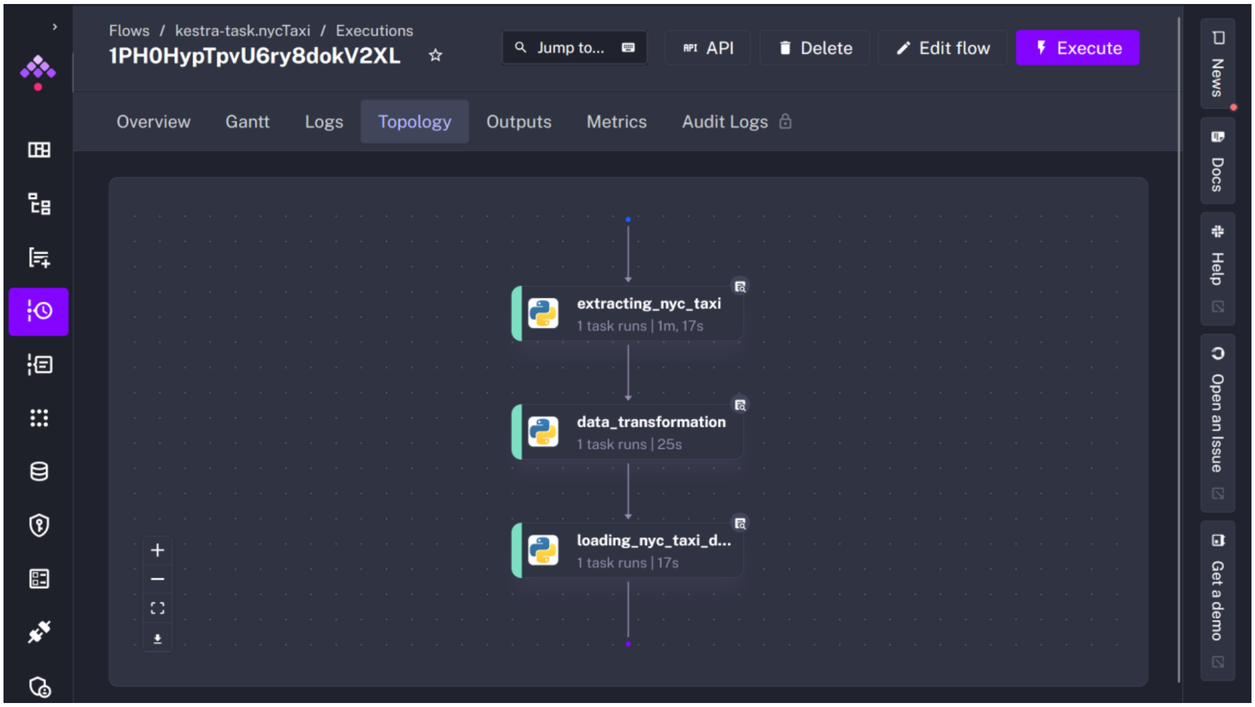
Please find the GitHub Repository here: https://github.com/KawiraSharon/nyc-taxi-trip-etl-pipeline

To get a general overview of what this project is aimed at creating, I used a simple prompt to generate an image from ChatGPT to get a high level understanding of what I was going to further break down in chunks and develop. The image from ChatGPT is as shown;



The entire project leverages Kestra, an open-source platform for orchestration. Kestra enables perform all three tasks in the ETL process that is; extraction, transformation and loading. It carries them out independently, with each task's output as the input of the next task.

Kestra was selected for the modularity in monitoring independence, retries, logging, and reusability of data workflows it offers. Python tasks are executed using Kestra's script plugin for python, as illustrated, the following is the anticipated workflow.



Phase 1: Data Extraction

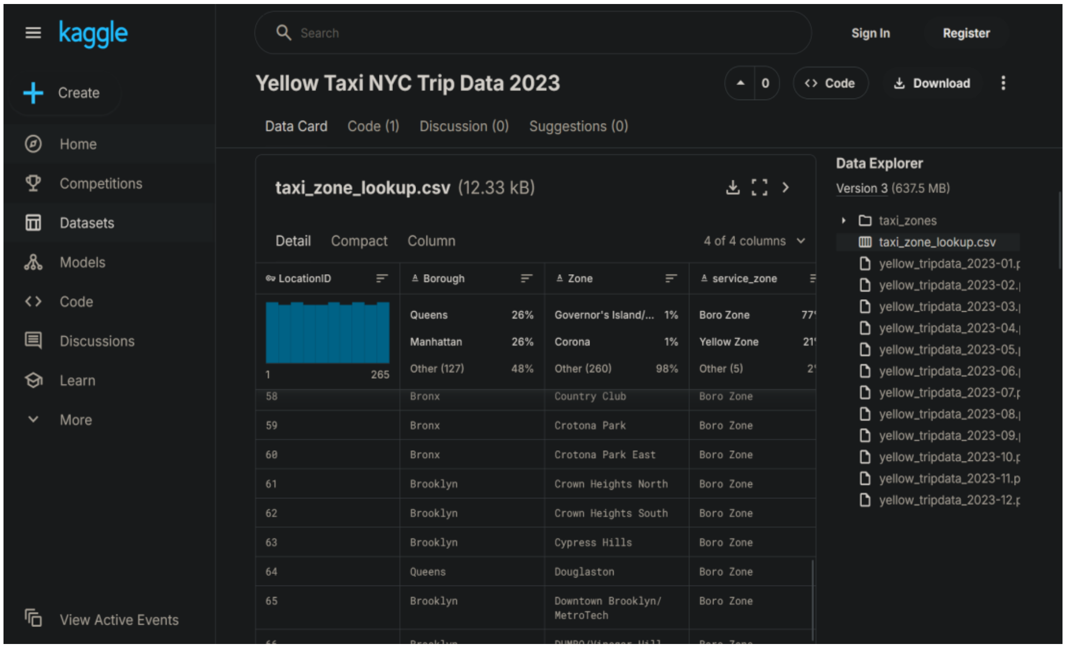
Kestra is used to automate data extraction from Kaggle. Kestra runs a python script that connects and allows it to download of the dataset from Kaggle's url using an API obtained publicly from the developers website in Kaggle.json format.

{"username":"sharonmungania","key":"c4942b83fcf67954e6e6a6709feecdea"}

Kestra follows this pipeline to then extract the yellow_tripdata_2023-01.parquet file using pyarrow and pandas and then saves it to a csv file, which becomes the output of this step.

The data was extracted from Kaggle via link; https://www.kaggle.com/datasets/albertjavier/yellow-tripdata-2023. It was downloaded from the Data Explorer section, left of the page as seen in the following screenshot, with the specific file as yellow_tripdata_2023-01.parquet as hovered over in the screenshot above, and since the dataset was very large and subdivided monthly for each month in year 2023, the select dataset was picked from January, holding over 3 million records and 19 columns, making it complex enough for this project.

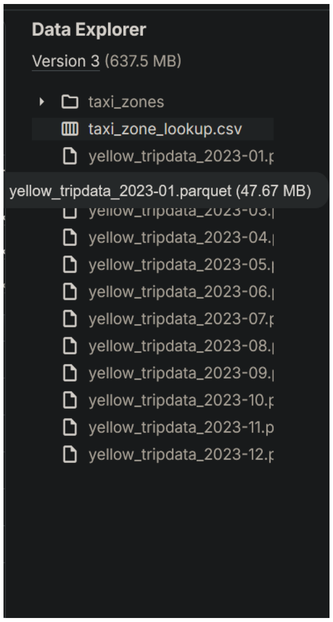
Screenshot of the dataset as seen on Kaggle;



taxi_zone_lookup.csv (12.33 kB)

LocationID	Borough	Zone	service_zone
1 265	Queens 26% Manhattan 26% Other (127) 48%	Governor's Island/... 1% Corona 1% Other (260) 98%	Boro Zone 77 Yellow Zone 21 Other (5) 2
58	Bronx	Country Club	Boro Zone
59	Bronx	Crotona Park	Boro Zone
60	Bronx	Crotona Park East	Boro Zone
61	Brooklyn	Crown Heights North	Boro Zone
62	Brooklyn	Crown Heights South	Boro Zone
63	Brooklyn	Cypress Hills	Boro Zone
64	Queens	Douglaston	Boro Zone
65	Brooklyn	Downtown Brooklyn/ MetroTech	Boro Zone

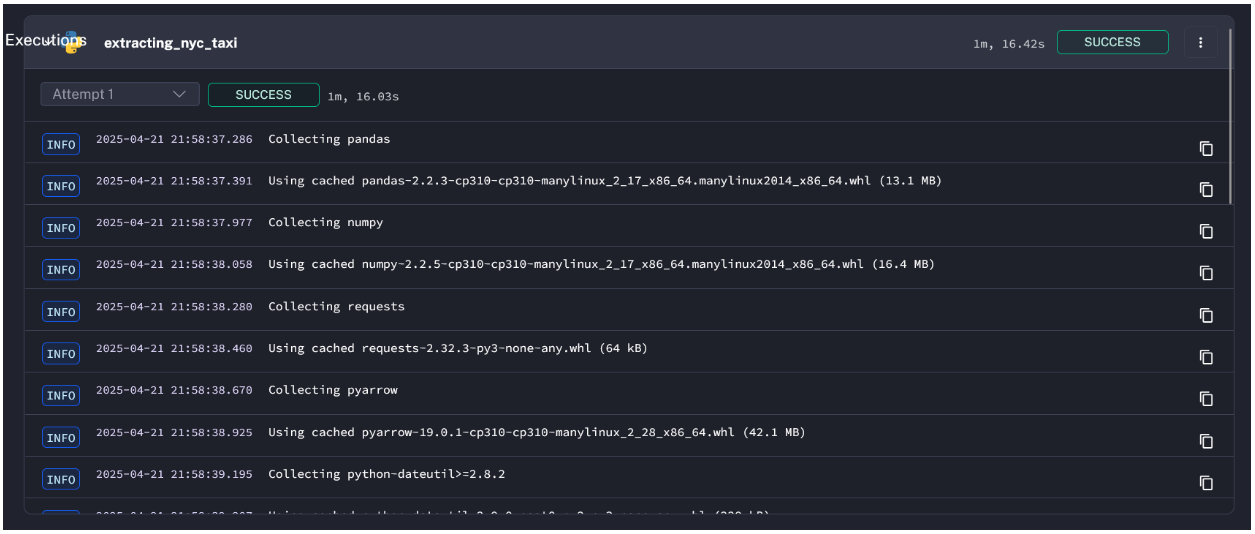
This highlights the section of the dataset chosen for the ETL Project. January was sampled for this dataset.


Data Explorer

Version 3 (637.5 MB)

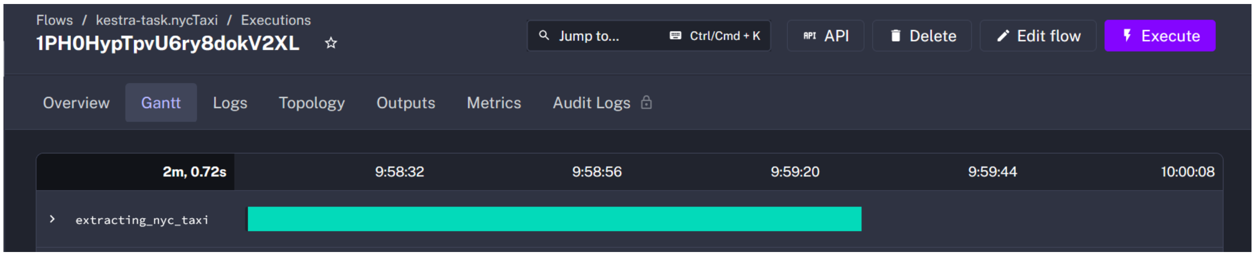
taxi_zones
taxi_zone_lookup.csv
yellow_tripdata_2023-01.
yellow_tripdata_2023-01.parquet (47.67 MB)
yellow_tripdata_2023-03.
yellow_tripdata_2023-04.
yellow_tripdata_2023-05.
yellow_tripdata_2023-06.
yellow_tripdata_2023-07.
yellow_tripdata_2023-08.
yellow_tripdata_2023-09.
yellow_tripdata_2023-10.
yellow_tripdata_2023-11.p
yellow_tripdata_2023-12.

Logs of the extraction phase:



```
INFO  2025-04-21 21:58:37.286  Collecting pandas
INFO  2025-04-21 21:58:37.391  Using cached pandas-2.2.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.1 MB)
INFO  2025-04-21 21:58:37.977  Collecting numpy
INFO  2025-04-21 21:58:38.058  Using cached numpy-2.2.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
INFO  2025-04-21 21:58:38.280  Collecting requests
INFO  2025-04-21 21:58:38.460  Using cached requests-2.32.3-py3-none-any.whl (64 kB)
INFO  2025-04-21 21:58:38.670  Collecting pyarrow
INFO  2025-04-21 21:58:38.925  Using cached pyarrow-19.0.1-cp310-cp310-manylinux_2_28_x86_64.whl (42.1 MB)
INFO  2025-04-21 21:58:39.195  Collecting python-dateutil>=2.8.2
```

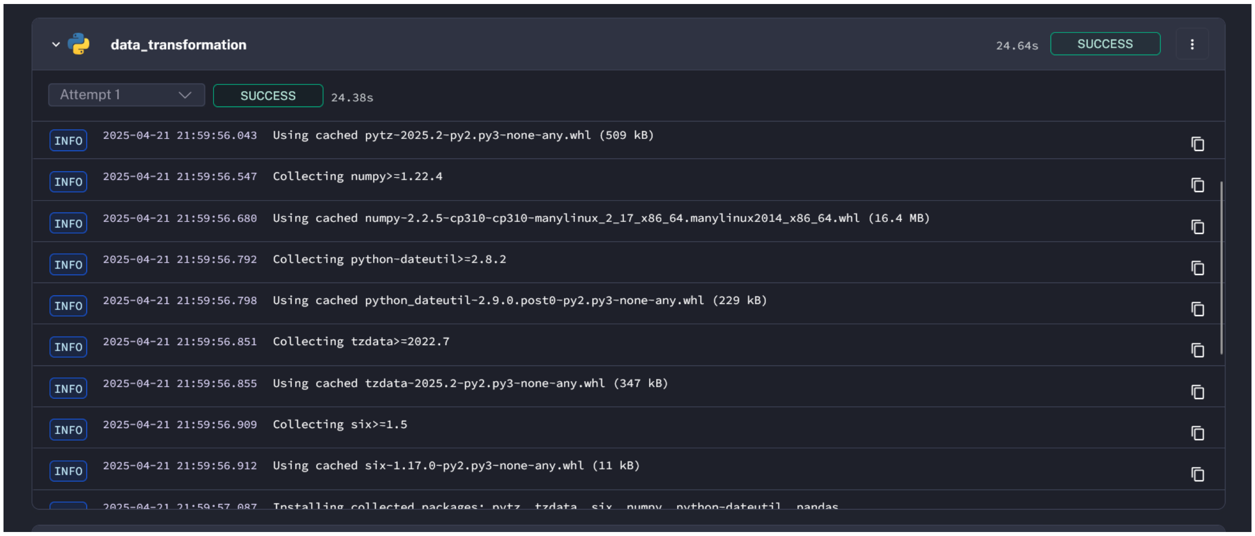
Kestra’s Gantt indicating a successful execution of this phase:



Phase 2: Transforming and Cleaning the Data

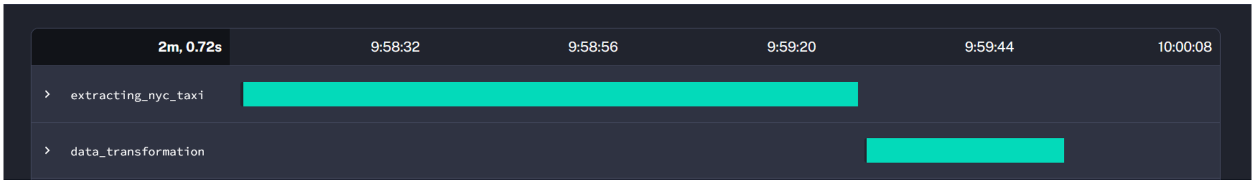
Here, Kestra is used here to run a Python script that gives the dataset a makeover and prepares it for insertion to a database and visualization steps. It is used to procedurally remove ghost trips, that is trips without passengers or trips with zero distance. It also normalizes timestamps to a indicate a consistent format. Kestra through the Python script is used to introduce a column, trip duration, and converts the time to minutes.

The following are Kestra logs for the data transformation execution;



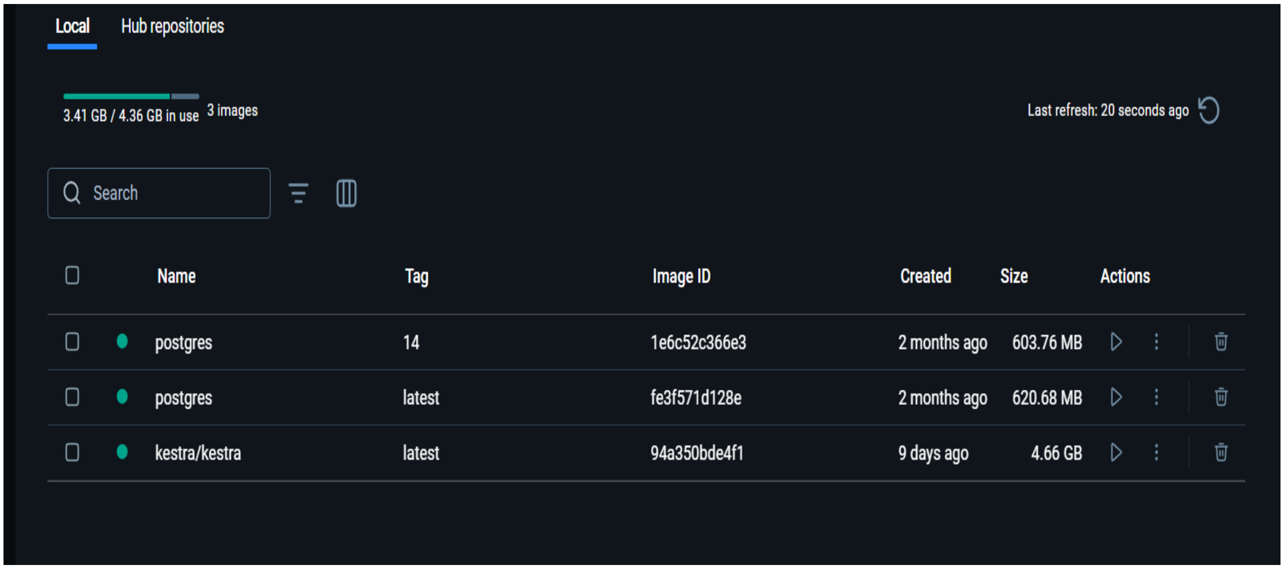
```
INFO  2025-04-21 21:59:56.043  Using cached pytz-2025.2-py2.py3-none-any.whl (509 kB)
INFO  2025-04-21 21:59:56.547  Collecting numpy>=1.22.4
INFO  2025-04-21 21:59:56.680  Using cached numpy-2.2.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
INFO  2025-04-21 21:59:56.792  Collecting python-dateutil>=2.8.2
INFO  2025-04-21 21:59:56.798  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
INFO  2025-04-21 21:59:56.851  Collecting tzdata>=2022.7
INFO  2025-04-21 21:59:56.855  Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
INFO  2025-04-21 21:59:56.909  Collecting six>=1.5
INFO  2025-04-21 21:59:56.912  Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
      2025-04-21 21:59:57.087  Installing collected packages: pytz, tzdata, six, numpy, python-dateutil, pandas
```

Kestra’s Gantt indicating a successful execution of this phase, building onto the previous task;



Phase 3: Data Loading Phase

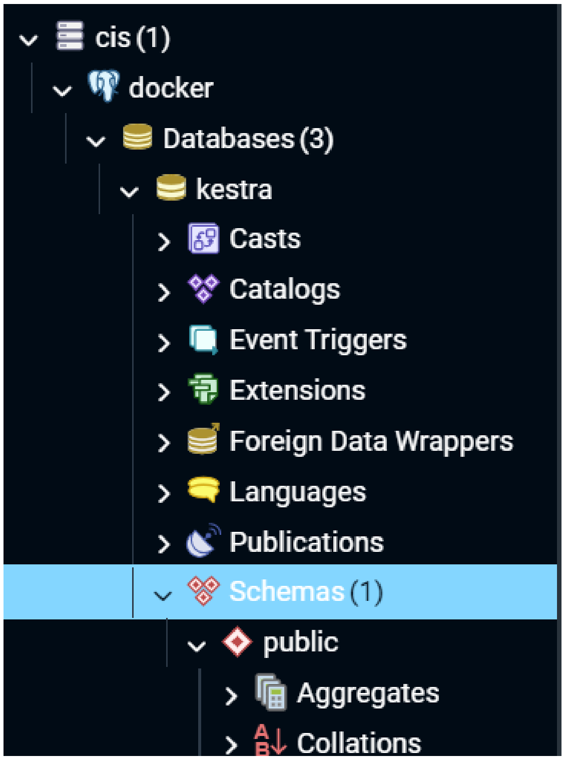
#TODO: Replace docker screenshot. Docker is started, and once the engine is running, a PostgreSQL container is defined as shown in the screenshot herein; Please note that this screenshot was taken with the containers stopped because they use a lot of resources, but in the instance one wants to connect docker to the PostgreSQL instance, they just hit the play button next to the size of the container.



□		Name	Tag	Image ID	Created	Size	Actions
□	●	postgres	14	1e6c52c366e3	2 months ago	603.76 MB	▷ ⋮ 🗑
□	●	postgres	latest	fe3f571d128e	2 months ago	620.68 MB	▷ ⋮ 🗑
□	●	kestra/kestra	latest	94a350bde4f1	9 days ago	4.66 GB	▷ ⋮ 🗑

PgAdmin is then opened to allow manage the database.

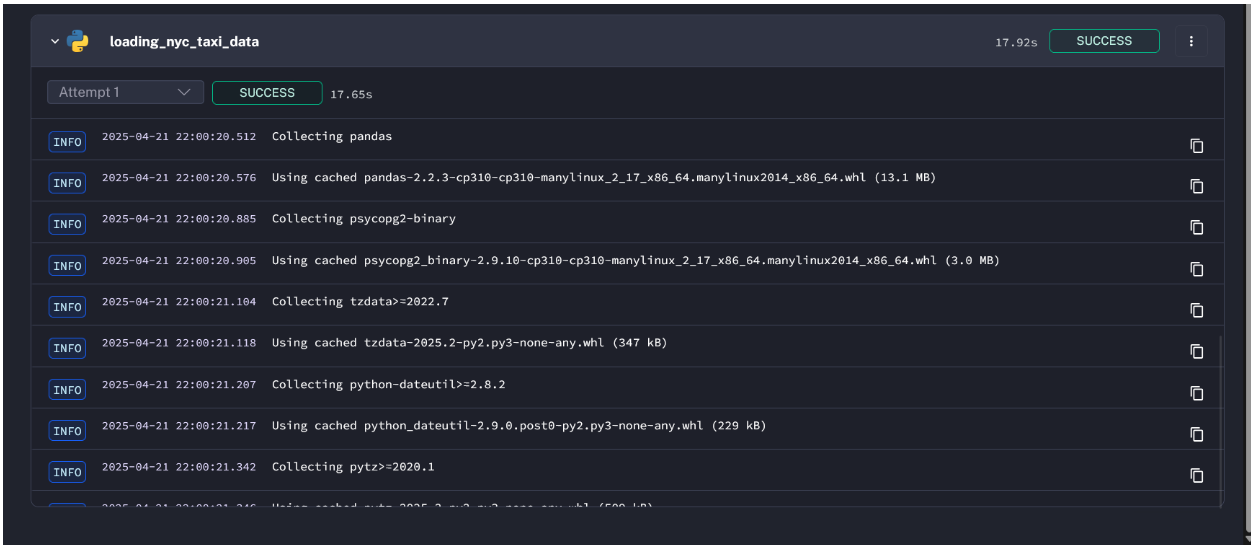
- In pgAdmin, a server group cis created.
- A server by the name docker created.
- A database kestra created.
- This architecture is displayed in the following screenshot;



```
⌄ cis (1)
  ⌄ docker
    ⌄ Databases (3)
      ⌄ kestra
        › Casts
        › Catalogs
        › Event Triggers
        › Extensions
        › Foreign Data Wrappers
        › Languages
        › Publications
        ⌄ Schemas (1)
          ⌄ public
            › Aggregates
            › Collations
```

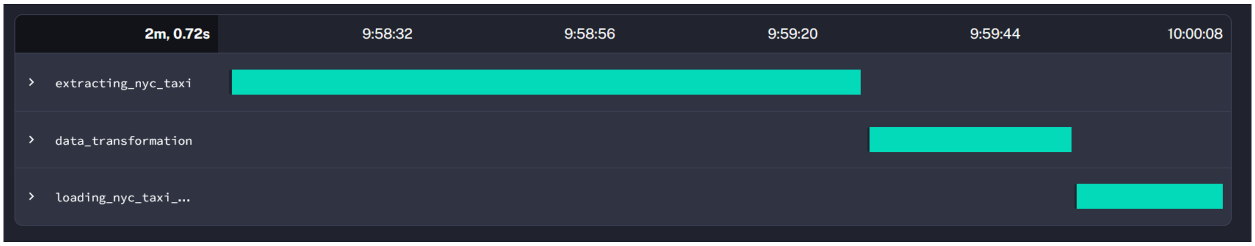
Kestra then loads data from the transformed_data.csv which was the output of the transformation phase to the PostgreSQL database containerized in docker and managed via pgAdmin which is locally installed. It also creates a table nyc_taxi_trips as specified in the syntax.

The following are Kestra logs for the data loading execution;

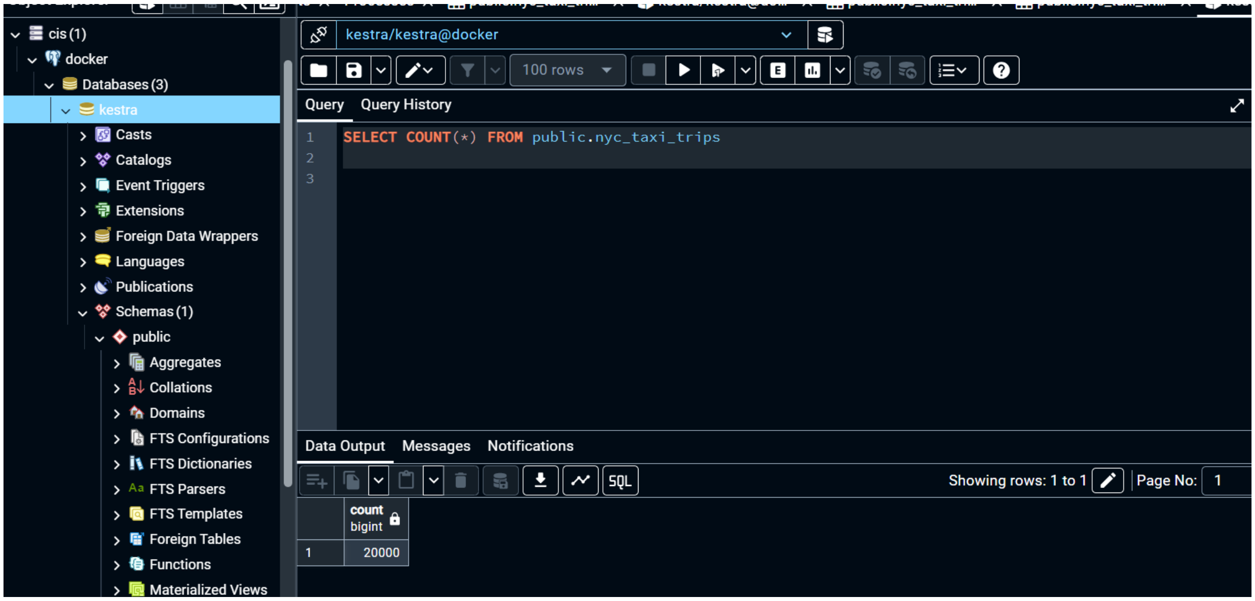


```
INFO  2025-04-21 22:00:20.512  Collecting pandas
INFO  2025-04-21 22:00:20.576  Using cached pandas-2.2.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.1 MB)
INFO  2025-04-21 22:00:20.885  Collecting psycopg2-binary
INFO  2025-04-21 22:00:20.905  Using cached psycopg2_binary-2.9.10-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0 MB)
INFO  2025-04-21 22:00:21.104  Collecting tzdata>=2022.7
INFO  2025-04-21 22:00:21.118  Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
INFO  2025-04-21 22:00:21.207  Collecting python-dateutil>=2.8.2
INFO  2025-04-21 22:00:21.217  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
INFO  2025-04-21 22:00:21.342  Collecting pytz>=2020.1
      2025-04-21 22:00:21.346  Using cached pytz-2025.2-py2.py3-none-any.whl (509 kB)
```

Kestra's Gantt indicating a successful execution of this phase, building from the previous tasks;



Instead of inserting the over 2 million records as transformed and obtained as output from the transformation phase, Kestra was used to insert 20,000 records onto the PostgreSQL database as shown in the screenshot below, to provide a reasonable dataset to work with.



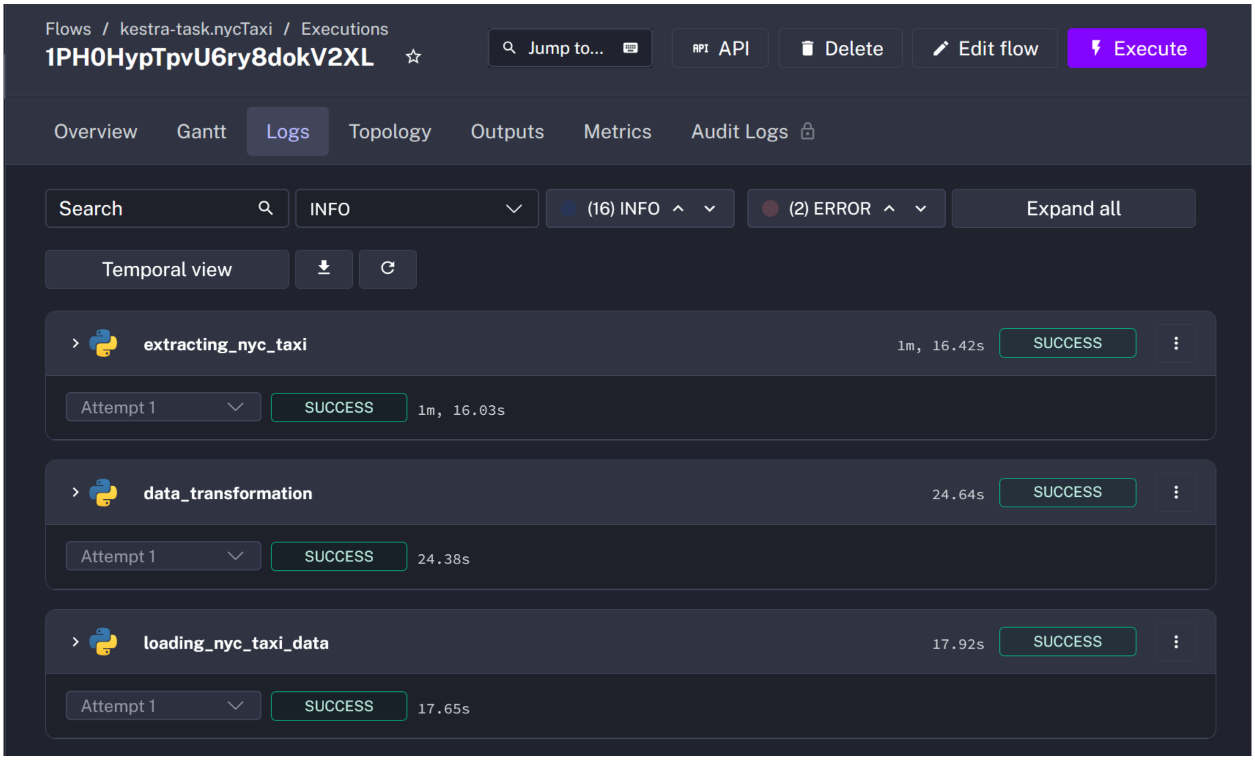
kestra/kestra@docker

```
SELECT COUNT(*) FROM public.nyc_taxi_trips
```

Showing rows: 1 to 1 | Page No: 1

	count bigint
1	20000

The following screenshot shows logs of success returned from all three processes executed by Kestra.



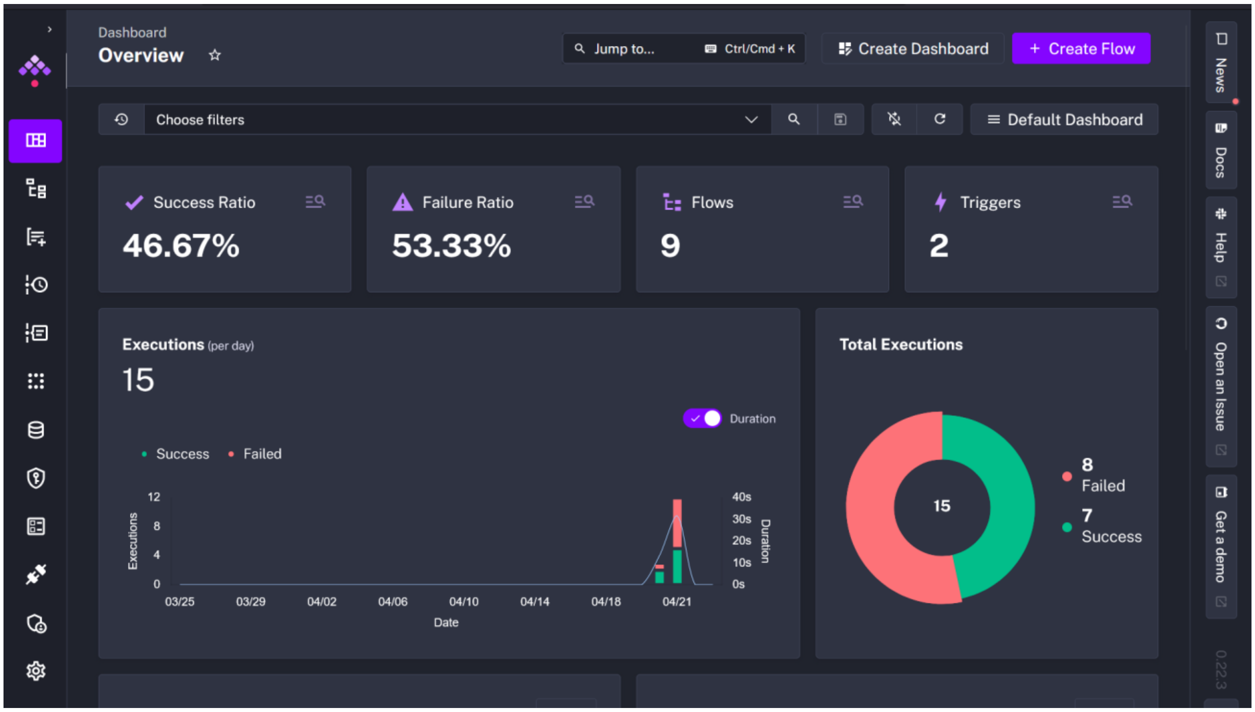
Flows / kestra-task.nycTaxi / Executions

1PH0HypTpvU6ry8dokV2XL

INFO | (16) INFO | (2) ERROR

```
extracting_nyc_taxi      1m, 16.42s   SUCCESS
  Attempt 1   SUCCESS   1m, 16.03s
data_transformation      24.64s       SUCCESS
  Attempt 1   SUCCESS   24.38s
loading_nyc_taxi_data    17.92s       SUCCESS
  Attempt 1   SUCCESS   17.65s
```

The eventual resulting dashboard on K estra after running the entire pipeline is orchestrated;



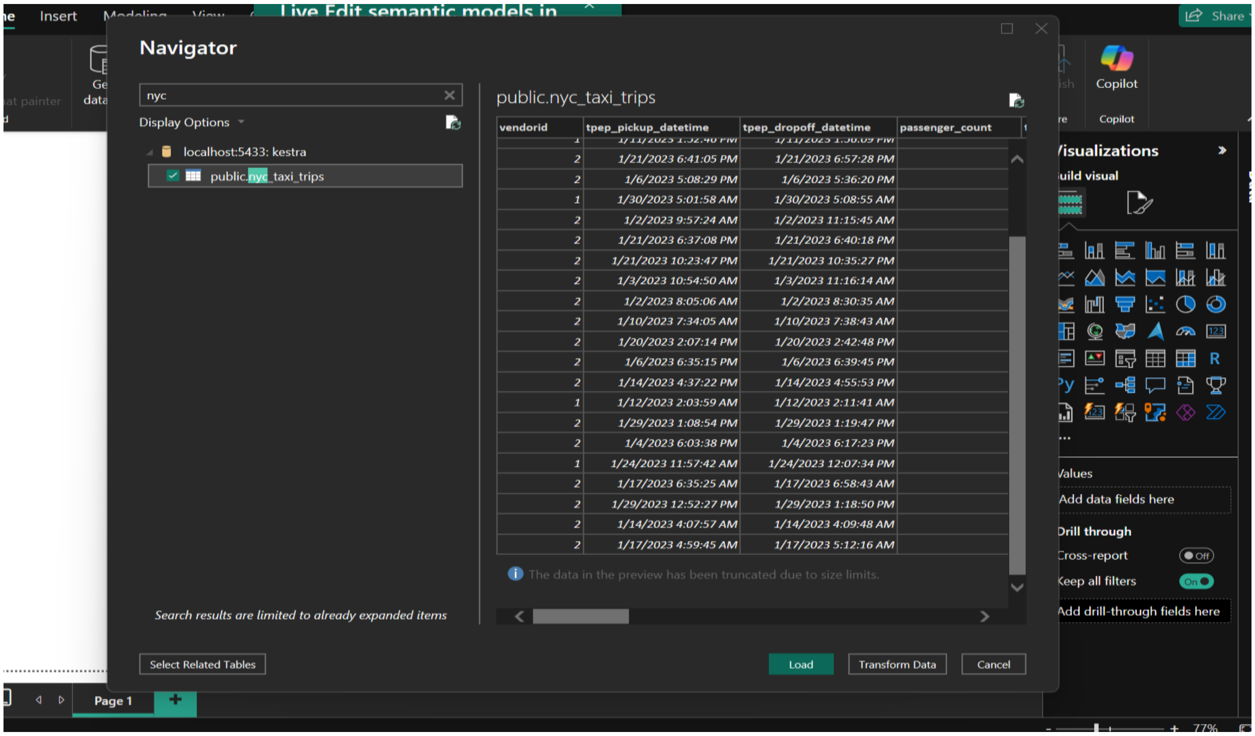
A quick interpretation of the dashboard displayed;

- The failure ratio is slightly higher than the success ratio which is normal in development environments. There are 15 executions, with 8 failed ones and 7 successful ones.
- The pipeline is **automated and scalable as** 2 triggers are configured, reducing need for manual intervention;
  - Automated schedule triggers that reduce the need for manual configuration.
  - Chained execution that is the next step say loading, only begins after transformation indicates success.

Phase 4: Data Visualization on PowerBI

In this final phase, data loaded onto the PostgreSQL, also transformed_data is loaded onto Power Bi for visualization.

Power Bi has capabilities to connect directly to the database directly through the docker container as shown in the following screenshots.

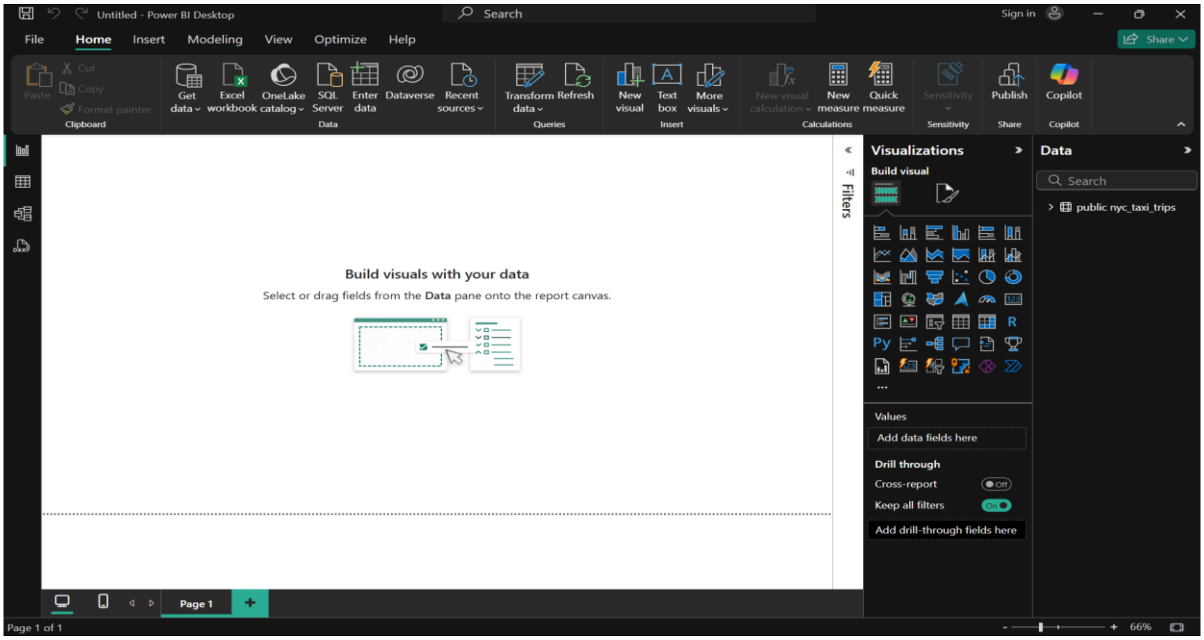


public.nyc_taxi_trips

vendorid	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count
1	1/11/2023 1:32:40 PM	1/11/2023 1:36:09 PM	
2	1/21/2023 6:41:05 PM	1/21/2023 6:57:28 PM	
2	1/6/2023 5:08:29 PM	1/6/2023 5:36:20 PM	
1	1/30/2023 5:01:58 AM	1/30/2023 5:08:55 AM	
2	1/2/2023 9:57:24 AM	1/2/2023 11:15:45 AM	
2	1/21/2023 6:37:08 PM	1/21/2023 6:40:18 PM	
2	1/21/2023 10:23:47 PM	1/21/2023 10:35:27 PM	
2	1/3/2023 10:54:50 AM	1/3/2023 11:16:14 AM	
2	1/2/2023 8:05:06 AM	1/2/2023 8:30:35 AM	
2	1/10/2023 7:34:05 AM	1/10/2023 7:38:43 AM	
2	1/20/2023 2:07:14 PM	1/20/2023 2:42:48 PM	
2	1/6/2023 6:35:15 PM	1/6/2023 6:39:45 PM	
2	1/14/2023 4:37:22 PM	1/14/2023 4:55:53 PM	
1	1/12/2023 2:03:59 AM	1/12/2023 2:11:41 AM	
2	1/29/2023 1:08:54 PM	1/29/2023 1:19:47 PM	
2	1/4/2023 6:03:38 PM	1/4/2023 6:17:23 PM	
1	1/24/2023 11:57:42 AM	1/24/2023 12:07:34 PM	
2	1/17/2023 6:35:25 AM	1/17/2023 6:58:43 AM	
2	1/29/2023 12:52:27 PM	1/29/2023 1:18:50 PM	
2	1/14/2023 4:07:57 AM	1/14/2023 4:09:48 AM	
2	1/17/2023 4:59:45 AM	1/17/2023 5:12:16 AM	

The data in the preview has been truncated due to size limits.

The next screenshot shows nyc_taxi_trips on the far right as a table successfully loaded onto Power Bi.



Build visuals with your data

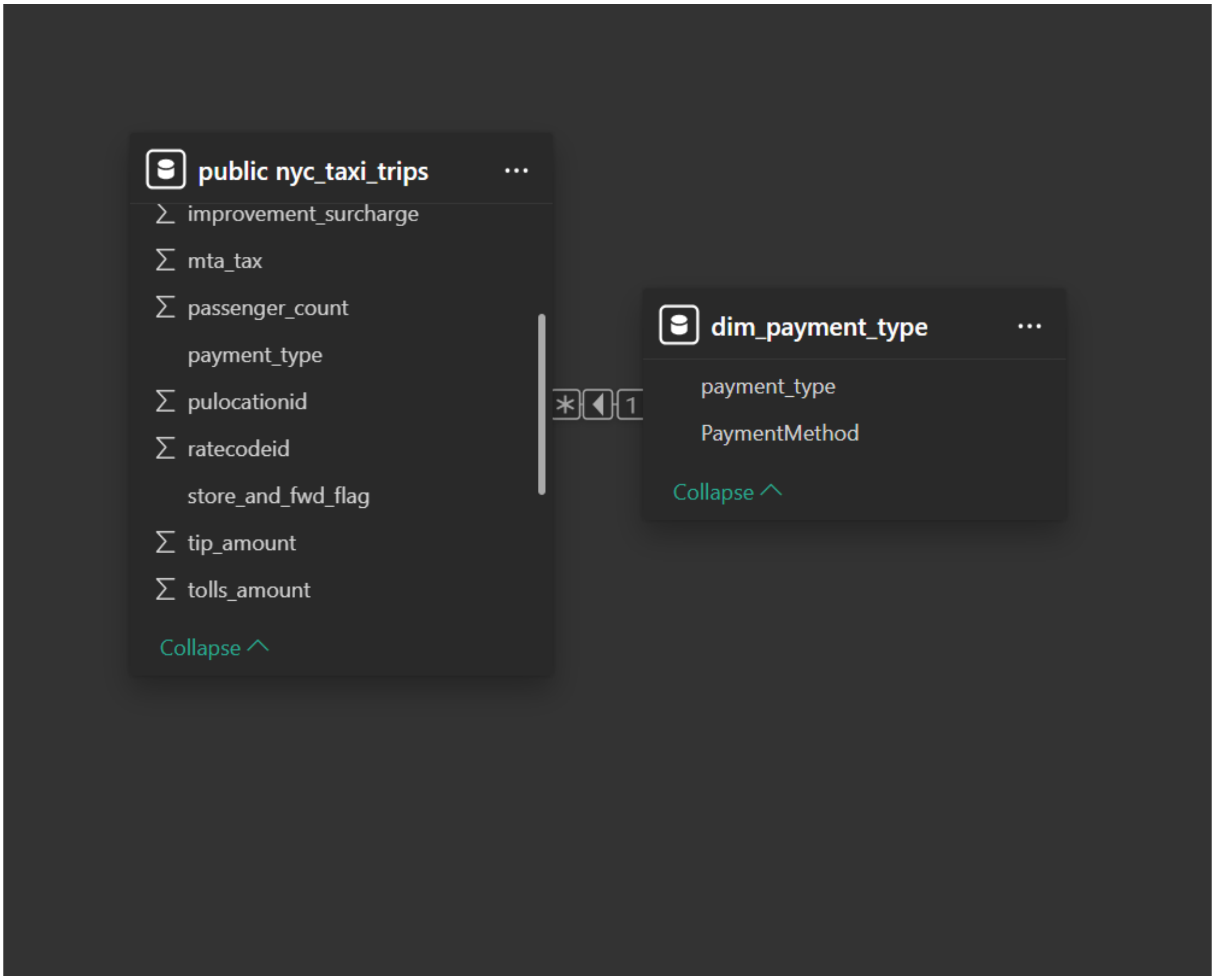
Select or drag fields from the Data pane onto the report canvas.

To support effective data analysis and visualization, a star schema approach was selected.

Star-Schema set up

- The first step involves building dimension and fact tables then creating a star schema from the two.
- The fact table is made up of fields that are numerical and those that represent how much, how many and how long, around which the star schema is centered.
- The Dimension table is made up of label, category and name fields to represent who and what methods.

Fact table and Dimension Table



The dimension table is connected to the fact table using the payment_type field. This field shows a one to many relationship; (1 : *)

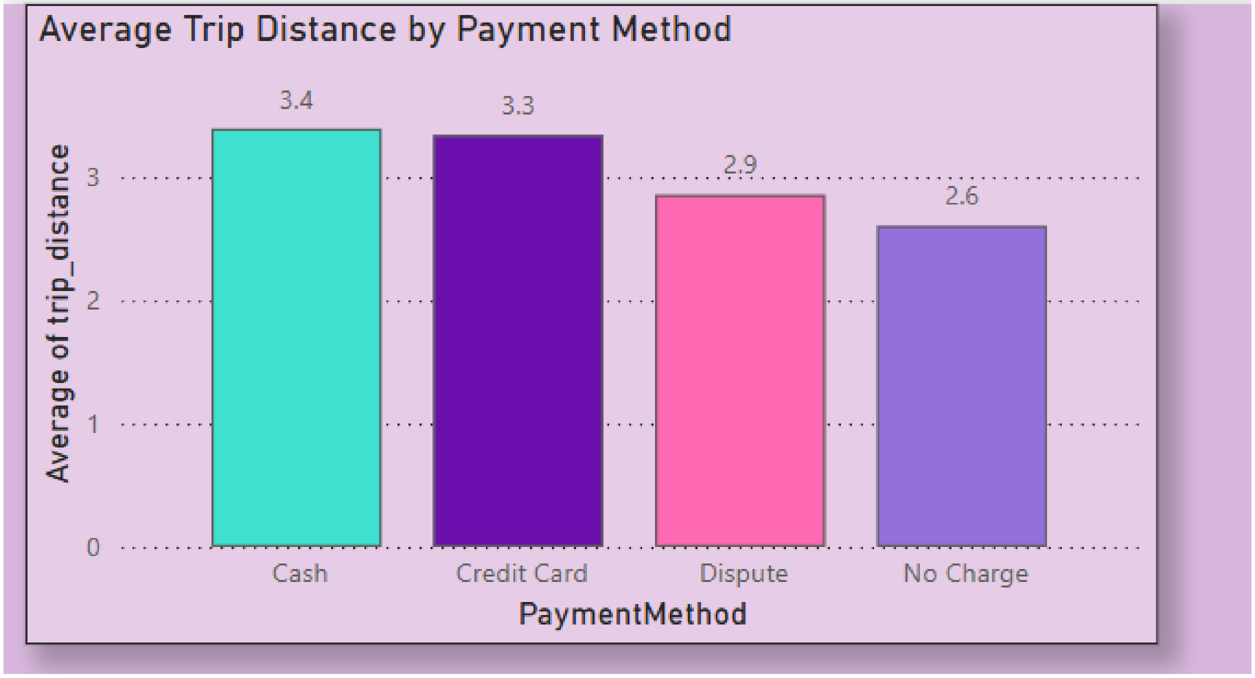
Building the Schema Visualizations

Stacked Column Visualization

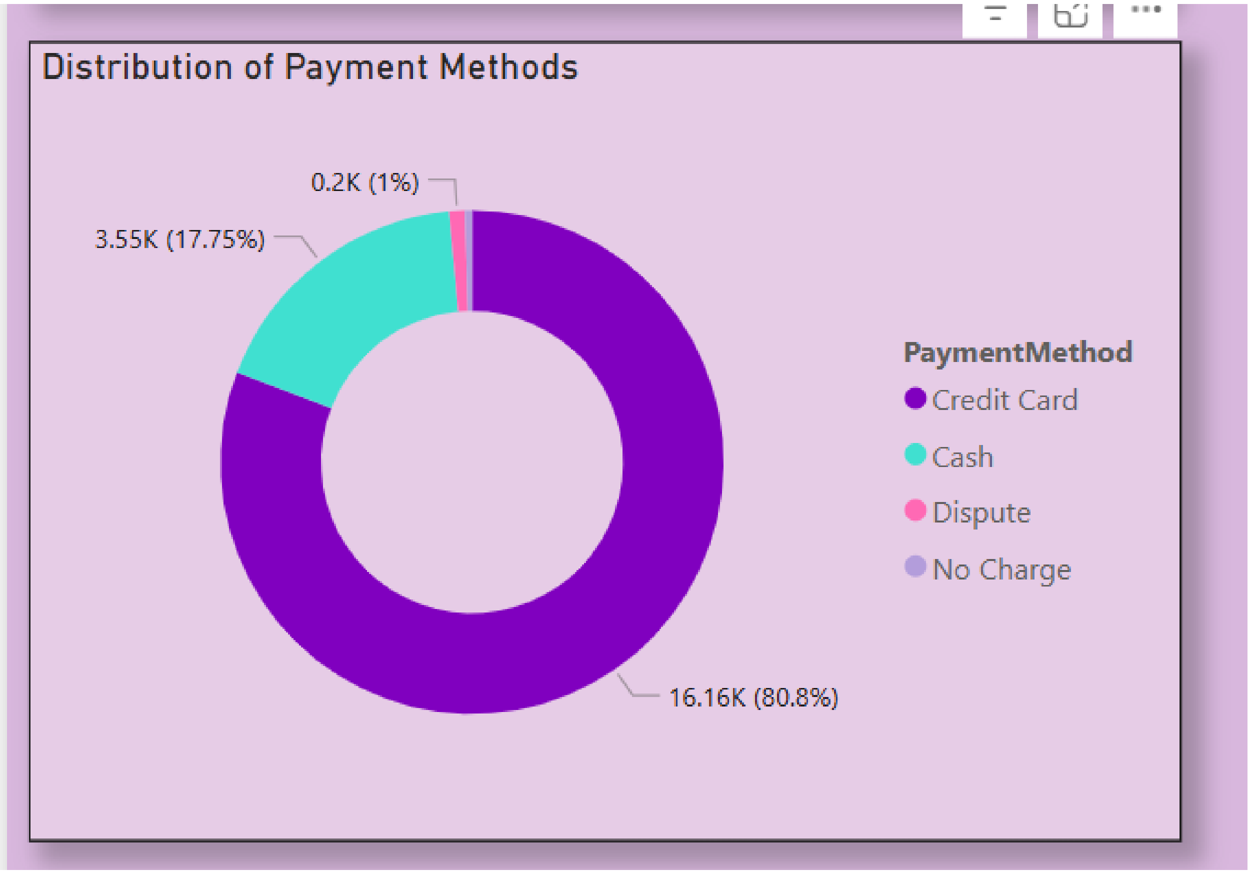
A stacked column chart was selected from visualization options provided by Power BI to visualize how payment methods compared against trip distances.

- The sum of trip distance from the fact table was picked for the y axis and payment method from the dimension table for the x axis.
- The sum of trip distance was switched to average under visualizations to better provide insights on how far people travel by distance, against the method of payment they used.

Resulting Visual is as shown in the screenshot here

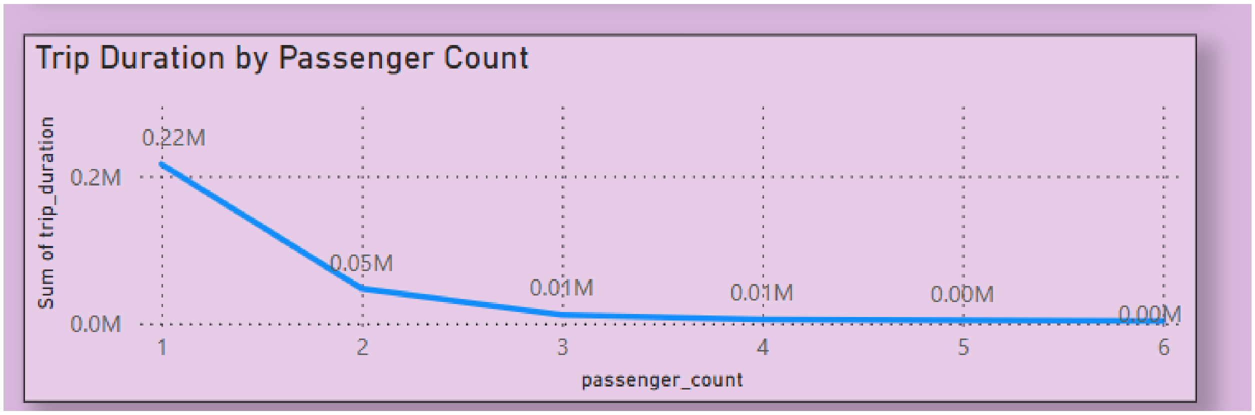


Donut Visualization



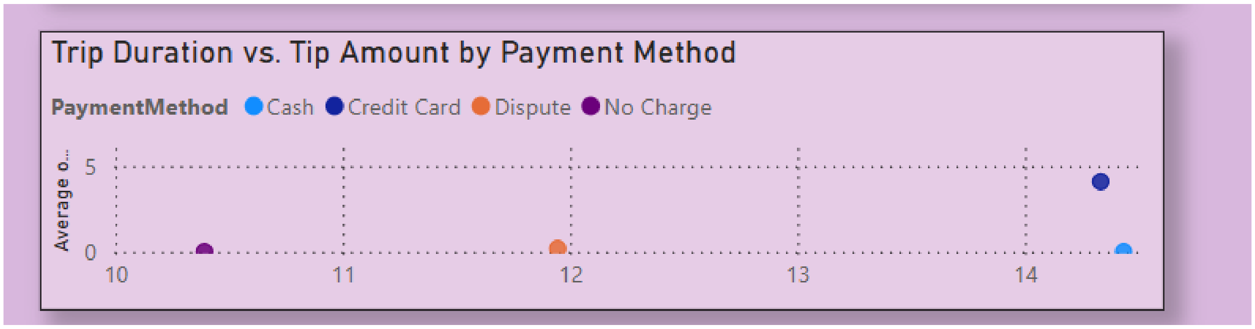
This visualization is a representation showing that 80.8% of the trips were paid for by credit card payment method, standing as the highest and most preferred mode of payment by passengers in these taxi trips. The second highest mode of payment used was cash, visualized at 17.75%, and that dispute arose in 1% of the trips, then other payment methods followed. This chart shows overall that most passengers use credit cards to pay for their trips.

Line Chart Visualization

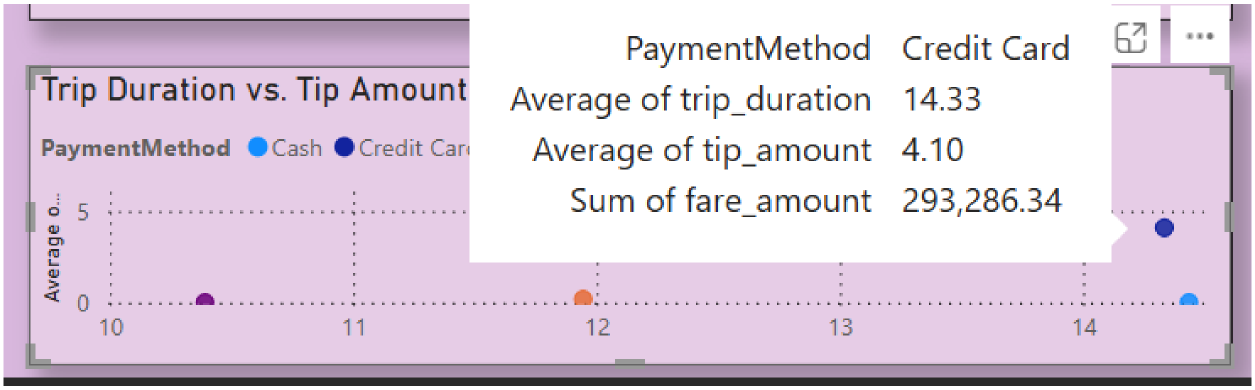


The line chart above is a visualizes trip duration against passenger count, it shows that the highest number of taxi trips are made by individual travelers. Group rides are less common and their trips are also shorter.

Scatter Chart Visualization



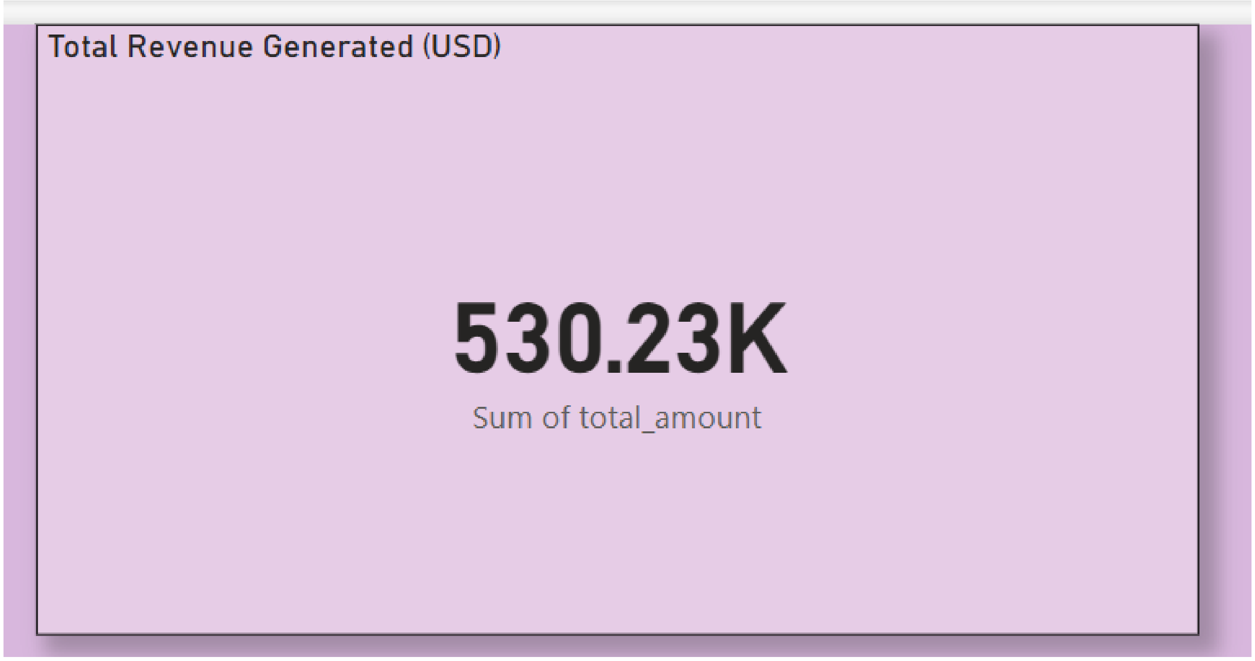
This scatter chart is best interpreted when one hovers over it as seen in the following screenshot;



PaymentMethod	Credit Card
Average of trip_duration	14.33
Average of tip_amount	4.10
Sum of fare_amount	293,286.34

It is a demonstration of who the bubble represents, that is the payment by credit card. It provides valuable insights on the averages of trip duration in minutes and the tip amount. It also shows the total revenue each payment method generates. Payment by Credit Card which is hovered over seems like a likely outlier, but this is not due to data error, it is because it is the highest used payment method, and differs with other payment methods by a large margin.

Card Visualization



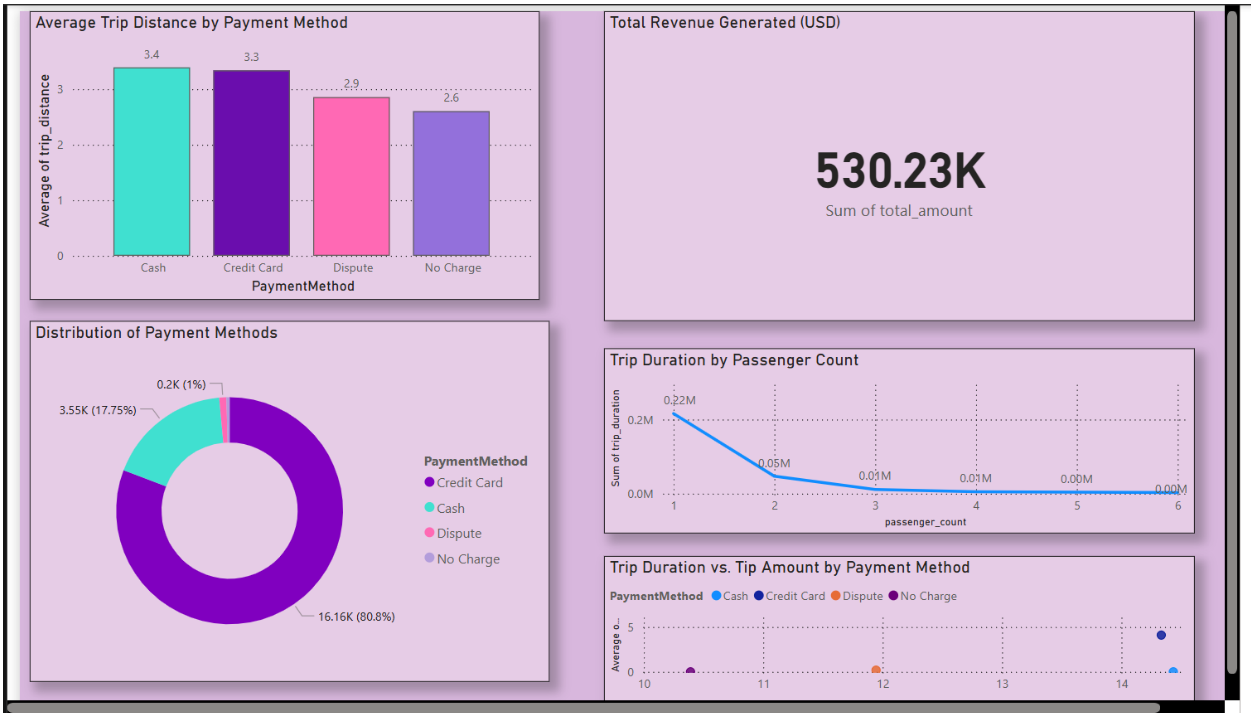
Total Revenue Generated (USD)

530.23K

Sum of total_amount

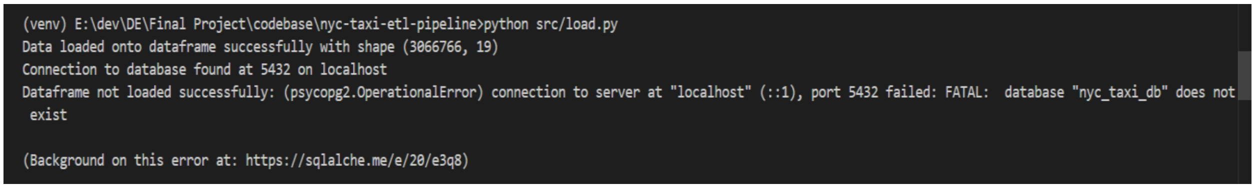
This visualization represents all of the revenue generated by all the NYC taxi trips in January 2023, from our select sample dataset of 20,000 records. It is a summation of all values in the total_amount field including fares, tips, and any other applicable charges. It is considered a very Key Performance Indicator (KPI) in the dashboard, as it provides an overall of the amount generated from the sampled dataset.

The final star schema on Power BI looks like this;



Challenges Encountered and Project Restructuring

- Unlike when dealing with a .csv file, a .parquet file as is adapted in this project, does not display data automatically when imported onto the standard software tools. Instead, it has binary contents, which are only readable after installing the Pyarrow package paired with an import Pandas command. I was able to solve that challenge after installing the required Pyarrow package.
- Initially, I attempted to use my local text editor, visual studio code for my data extraction, transformation and loading and then load my data onto a PostgreSQL instance initialized on docker, except I encountered an environment mismatch, where my python scripts on visual studio code were running and hosted locally and my PostgreSQL instance hosted on docker's environment, a virtual environment. I attempted to resolve the problem but ended up restructuring my entire project to use Kestra for the entire orchestration, and a unified environment.
- The error I was encountering is as seen in the screenshot here;



```
(venv) E:\dev\DE\Final Project\codebase\nyc-taxi-etl-pipeline>python src/load.py
Data loaded onto dataframe successfully with shape (3066766, 19)
Connection to database found at 5432 on localhost
Dataframe not loaded successfully: (psycopg2.OperationalError) connection to server at "localhost" (::1), port 5432 failed: FATAL:  database "nyc_taxi_db" does not
 exist

(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

- While the setup for the PostgreSQL instance on docker was successful, I encountered very persistent issues with the environment mismatch and while debugging. I attempted to edit the psycopg2 configuration file to expose the docker port to host.internal.docker but I couldn't resolve the error even then.
- After several failed attempts at debugging the code, I decided to restructure the entire project to use Kestra for all of the data extraction, transformation and loading, as thoroughly documented in this entire document. The initial plan was to use kestra for the entire project orchestration, then load the database onto my PostgreSQL, and import it onto Power BI, a totally new tool for me as well, and quite the different plan from what I had earlier visualized during my proposal preparation.
- Using Power BI however turned out to be very efficient for visualizations. Especially because it would allow a preview of the visualizations as I built up on them. The values for legends, x axis, y axis fields were also all on the dashboard, saving on back and forth across different windows to select, drag, and or even write syntax for the visualizations. Altogether, using dimension and fact tables as part of the star schema, made data visualization even further easily readable and manageable.
- I hadn't interacted with Kestra before the project as well, and I encountered challenges even during the class setup. For that reason, I misunderstood orchestration for a process that was supposed to follow visualization. After an in-class discussion following the last week of class discussion however, I learned that orchestration was meant to tie together the entire ETL process. It came with a bit of a steep learning curve but I am glad I was able to overcome that and successfully orchestrate my entire pipeline using Kestra.
- Spending too much time on the loading phase also meant that I would not be able to adhere to the Gantt chart I had earlier created. Which remains a planned future improvement on managing time and ensuring familiarity with tools selected for a specific project before project initialization to ensure a smooth sail in the stages thereafter.

Successes realized through the ETL Pipeline Project

- I now have a solid understanding of Kestra as an orchestration tool for data pipelines. I can competently set up the entire tech stack used in this ETL pipeline, that is Kestra, Docker, Postgresql via pgAdmin, Git/GitHub.
- I managed to create a pipeline using a real-world data set, that has automated scheduling. I am able to understand that with the use of Kestra, eliminate manual task scheduling which is highly valuable to organizations during data engineering today. I also understand that Kestra runs tasks independently without conflicting, ensuring scalability unlike when using traditional methods like Visual Studio Code.
- I interacted with Power BI for the first time and was able to build meaningful visualizations with ease, thanks in part to my previous experience with Databricks, and Looker Studio.

- I gained a more grounded and cemented understanding of using Docker with respect to using local development environment scripts, being ports exposed must match to avoid conflict, although this remains something I plan on exploring beyond the scope of this project.
- I have successfully pushed my codebase to GitHub and will showcase it as part of my portfolio projects as well as feature it in my personal website.
- I have gained invaluable skills in data extraction, transformation, loading, visualization and orchestration through the entire project. The knowledge I have been studying in the course throughout the entire semester is more cemented in my brain and my problem-solving as well as practical skills have improved by far.

Sources and References

ChatGPT to generate the initial image to get the overall understanding of the project; https://chatgpt.com/

Power BI for Visualizations; https://www.microsoft.com/en-us/power-platform/products/power-bi/

Kestra for Orchestration; https://kestra.io/

pgAdmin to manage the PostgreSQL database; https://www.pgadmin.org/docs/pgadmin4/development/getting_started.html

Docker to containerize the PostgreSQL instance; https://docs.docker.com/get-started/

Git/GitHub for version control and host all code; https://github.com/