BigQuery Assignment

Uzair Nadeem 24928, Shahmeer Khan 25156

Rough Work on Paper and Star Schema:

Date:
M T W T F S S

Big Query Assignment

Business Process: Taxi rides ~~are~~ taken / booked or requested by customers which are then fulfilled by drivers ~~which~~ who drive a taxi

Grain: One row of the fact table represents information about one ride taken on a particular vehicle driven by a particular driver.

Potential facts
Payment amount
Vehicle capacity
Ride Duration (Dropoff time (year) - Pickup time (year) - years are subtracted and the result is assumed to be in minutes, since these columns have dates instead of time stamps)
PerMinuteRate → Calculated by dividing amount by the ride duration.

Potential Dimensions:
- Ride Dimension → Will have additional info about the ride (payment mode, ride status, pickup location, dropoff location)
- ~~Driver~~ Vehicle dimension (Vehicle ID, plate no., Type, Brand, Status, Car Tier) ~~Car Tier~~ → Brand and status are potential attributes, but may not necessarily be added.
- Driver dimension (Driver ID, Full Name, Phone, License Number, City)

Date:
M T W T F S S

· Date dimension → Will represent payment date of rides.
(DateID, Date, Year, ~~month~~ quarter, month, day)

Star Schema:

Major Python Functions and Scripts:

Data Ingestion:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
```

```
[ ] file_path1 = '/content/drive/My Drive/BigQuery/payments.csv'
    file_path2 = '/content/drive/My Drive/BigQuery/rides.csv'
    file_path3 = '/content/drive/My Drive/BigQuery/drivers.csv'
    file_path4 = '/content/drive/My Drive/BigQuery/vehicles.csv'
```

```
[ ] import pandas as pd
    paymentdf = pd.read_csv(file_path1)
    ridedf = pd.read_csv(file_path2)
    driverdf = pd.read_csv(file_path3)
    vehicledf = pd.read_csv(file_path4)
```

Fixing Phone number format in Drivers data-frame

```
def extract_and_concatenate_numbers(phone_number):
  if isinstance(phone_number, (float, np.float64, np.float32)):
    phone_number = str(phone_number)
  elif pd.isna(phone_number):  # Check for NaN
    phone_number = ""

  numbers = re.findall(r'\d+', phone_number)  # Find all digit sequences
  return ''.join(numbers)  # Concatenate the numbers

# Apply the function to the 'Phone' column
driverdf['Phone'] = driverdf['Phone'].apply(extract_and_concatenate_numbers)

# Display the updated DataFrame
driverdf.head()
```

	DriverID	FullName	Phone	LicenseNumber	JoinDate	City
0	DRV1000	Brian Hopkins	18176936961	fxeX6dvSXM	2020/12/30	NYC
1	DRV1001	NaN	4217248133	w7T26UTKcP	2016-11-11	Houston
2	DRV1002	Connie Beck	105906738342715	yW0zeRz5u0	1982.04.12	NYC
3	DRV1003	Kelsey Vega DVM	0017742882351242	O3zuHdGhi1	1990.01.17	Chi-town
4	DRV1004	Dana Brown		q6wb1hqhgy	2020/08/15	NYC

Function to fix date formats across several tables

```python
# Function to fix date values in driver column
def convert_to_date(date_str):
    try:
        return pd.to_datetime(date_str)
    except ValueError:
        try:
            # Attempt to extract year, month, and day using regex
            match = re.match(r'(\d{4})[/-](\d{1,2})[/-](\d{1,2})', str(date_str))
            if match:
                year = int(match.group(1))
                month = int(match.group(2))
                day = int(match.group(3))
                return pd.to_datetime(f'{year}-{month:02}-{day:02}')
            else:
                return pd.NaT  # Return NaT for invalid dates
        except:
            return pd.NaT
```

Fixing duplicate issues

Several columns throughout the data had values like “HOU” and “Houston” acting as two separate values even though they represented the same attribute

```python
# Fixing City values as NYC and new york city are treated differently
# Replace city values
driverdf['City'] = driverdf['City'].replace({'HOU': 'Houston', 'LA': 'Los Angeles', 'NYC': 'New York', 'Chi-town': 'Chicago'})

# Display the updated DataFrame
driverdf.head()
```

	DriverID	FullName	Phone	LicenseNumber	JoinDate	City
0	DRV1000	Brian Hopkins	18176936961	fxeX6dvSXM	2020-12-30	New York
1	DRV1001	NaN	4217248133	w7T26UTKcP	2016-11-11	Houston
2	DRV1002	Connie Beck	105906738342715	yW0zeRz5u0	1982-04-12	New York
3	DRV1003	Kelsey Vega DVM	0017742882351242	O3zuHdGhi1	1990-01-17	Chicago
4	DRV1004	Dana Brown		q6wb1hqhgy	2020-08-15	New York

Function to clean plate number formats

```python
# Function to fix number plate values
def clean_plate_number(plate):
    if isinstance(plate, str):
        cleaned_plate = re.sub(r'[^a-zA-Z0-9]', '', plate).upper()
        return cleaned_plate
    else:
        return plate

vehicledf['PlateNumber'] = vehicledf['PlateNumber'].apply(clean_plate_number)
vehicledf.head()
```

	VehicleID	PlateNumber	Type	Capacity	AssignedDriverID
0	VEH2000	6054QL	Sedan	3.0	DRV3864
1	VEH2001	055HFP	Sedan	4.0	DRV3841
2	VEH2002	762ZBX	Hatchback	6.0	DRV1973
3	VEH2003	M887AI	sedan	6.0	DRV3920
4	VEH2004	23OC983	SUV	6.0	DRV2621

Handling duplicate rows

```python
# Drop duplicate rows from each DataFrame
paymentdf = paymentdf.drop_duplicates()
ridedf = ridedf.drop_duplicates()
driverdf = driverdf.drop_duplicates()
vehicledf = vehicledf.drop_duplicates()
```

Function to fill in missing IDs

This worked by checking for the ID above and below the missing value to then calculate the missing ID as IDS were in order

```
def fill_missing_ids(df, id_column):

    # Extract prefix and numeric part using regex
    prefix = re.match(r"([a-zA-Z]+)", df[id_column].iloc[0]).group(1)

    # Convert ID column to string
    df[id_column] = df[id_column].astype(str)

    # Fill missing values by incrementing the numeric part
    for i in range(len(df)):
      if pd.isnull(df.loc[i, id_column]) or df.loc[i, id_column] == "nan":
        previous_id = df.loc[i - 1, id_column] if i > 0 else None
        if previous_id:
          numeric_part = int(re.search(r"\d+", previous_id).group(0)) + 1
          df.loc[i, id_column] = prefix + str(numeric_part)
        else:
          # Handle case where first value is missing
          df.loc[i, id_column] = prefix + "1"
      else:
        # Check if existing ID follows the format
        if not re.match(r"^[a-zA-Z]+\d+$", df.loc[i, id_column]):
          numeric_part = int(re.search(r"\d+", df.loc[i, id_column]).group(0))
          df.loc[i, id_column] = prefix + str(numeric_part)

    return df
```

Handled some missing IDS using dictionary mappings across tables

```
[ ] # Handle missing vals for driver id in ridedf
    # Create a dictionary mapping VehicleID to AssignedDriverID from vehicledf
    vehicle_driver_mapping = dict(zip(vehicledf['VehicleID'], vehicledf['AssignedDriverID']))

    # Fill missing DriverID in ridedf based on mapping and non-missing VehicleID
    for index, row in ridedf.iterrows():
        if pd.isnull(row['DriverID']) and not pd.isnull(row['VehicleID']):
            vehicle_id = row['VehicleID']
            if vehicle_id in vehicle_driver_mapping:
                ridedf.loc[index, 'DriverID'] = vehicle_driver_mapping[vehicle_id]

    print(ridedf.isnull().sum())
```

Due to the vastness of the data several missing values were dealt with by dropping rows

```
[ ] # Drop rows with missing 'PickupTime' or 'DropoffTime'
    ridedf = ridedf.dropna(subset=['PickupTime', 'DropoffTime'])
    print(ridedf.isnull().sum())
```

Function to replace missing values with column mode

```
[ ] def fill_missing_with_mode(df, column_name):

        # Calculate the mode of the column, handling potential multiple modes
        mode_value = df[column_name].mode().iloc[0]

        # Fill missing values with the mode
        df[column_name] = df[column_name].fillna(mode_value)

        return df
```

Function to replace missing values with column mean

```
def fill_missing_with_mean(df, column_name):
    df[column_name] = df[column_name].fillna(df[column_name].mean())
    return df
```

Handling missing values in Phone column in driver data-frame

```
[ ] driverdf['Phone'] = driverdf['Phone'].fillna('Not Available')
    driverdf.head()
```

	DriverID	FullName	Phone	LicenseNumber	City
0	DRV1000	Brian Hopkins	18176936961	fxeX6dvSXM	New York
2	DRV1002	Connie Beck	105906738342715	yW0zeRz5u0	New York
3	DRV1003	Kelsey Vega DVM	0017742882351242	O3zuHdGhi1	Chicago
4	DRV1004	Dana Brown	Not Available	q6wb1hqhgy	New York
5	DRV1005	Autumn Bryant	99692878848613	11DidxDTTV	Chicago

Preparing Dimension Tables:

Ride Dimension:

```
# Merge the DataFrames based on 'RideID'
DimRide_df = pd.merge(DimRide_df, paymentdf[['RideID', 'Mode']], on='RideID', how='left')

# Rename the 'Mode' column to 'PaymentMode'
DimRide_df = DimRide_df.rename(columns={'Mode': 'PaymentMode'})

DimRide_df.head()
```

	RideID	PickupLocation	DropoffLocation	Status	PaymentMode
0	RIDE30000	New Christopher	West Sarah	Completed	NaN
1	RIDE30001	New Pattymouth	Espinozaland	Completed	Card
2	RIDE30002	North Stephen	New Michael	Cancelled	NaN
3	RIDE30003	West Michael	Port Nicholas	Completed	NaN
4	RIDE30004	South Brian	New Michael	Completed	NaN

```
[ ] # missing vals handling
    DimRide_df = fill_missing_with_mode(DimRide_df, 'PaymentMode')
    DimRide_df.head()
```

Vehicle Dimension:

Created a new Vehicle-Tier column generated using vehicle types available in the data

```
# Create the 'VehicleTier' column based on 'Type'
DimVehicle_df['VehicleTier'] = DimVehicle_df['Type'].map({
    'Sedan': 'Premium',
    'SUV': 'High',
    'Hatchback': 'Mid',
    'Van': 'Low'
})

DimVehicle_df.head()
```

	VehicleID	PlateNumber	Type	VehicleTier
0	VEH2000	6054QL	Sedan	Premium
1	VEH2001	055HFP	Sedan	Premium
2	VEH2002	762ZBX	Hatchback	Mid
3	VEH2003	M887AI	Sedan	Premium
4	VEH2004	23OC983	SUV	High

Driver Dimension:

```
#Creating of driver dimdension table
DimDriver_df = driverdf.copy()
```

```
DimDriver_df.head()
```

	DriverID	FullName	Phone	LicenseNumber	City
0	DRV1000	Brian Hopkins	18176936961	fxeX6dvSXM	New York
2	DRV1002	Connie Beck	105906738342715	yW0zeRz5u0	New York
3	DRV1003	Kelsey Vega DVM	0017742882351242	O3zuHdGhi1	Chicago
4	DRV1004	Dana Brown	Not Available	q6wb1hqhgy	New York
5	DRV1005	Autumn Bryant	99692878848613	11DidxDTTV	Chicago

Date Dimension:

We used payment dates to act as our ride dates, also created date hierarchy

```
# Generate unique IDs for each date
DimDate_df['DateID'] = range(1, len(DimDate_df) + 1)

# Extract year, quarter, month, and day
DimDate_df['Year'] = DimDate_df['Date'].dt.year
DimDate_df['Quarter'] = DimDate_df['Date'].dt.quarter
DimDate_df['Month'] = DimDate_df['Date'].dt.month
DimDate_df['Day'] = DimDate_df['Date'].dt.day

DimDate_df.head()
```

	Date	DateID	Year	Quarter	Month	Day
44853	1970-01-01	1	1970	1	1	1
1080	1970-01-02	2	1970	1	1	2
8079	1970-01-03	3	1970	1	1	3
1783	1970-01-04	4	1970	1	1	4
31019	1970-01-05	5	1970	1	1	5

Fact Table:

After first copying the Driver IDs from DimDriver we then copied the corresponding Vehicle IDs via vehicle table

```
# Merge Ride_Fact_df with vehicledf to get VehicleID
Ride_Fact_df = pd.merge(Ride_Fact_df, vehicledf[['AssignedDriverID', 'VehicleID']],
                        left_on='DriverID', right_on='AssignedDriverID', how='left')

# Display the updated Ride_Fact_df
Ride_Fact_df.head()
```

	DriverID	AssignedDriverID	VehicleID
0	DRV1000	NaN	NaN
1	DRV1002	NaN	NaN
2	DRV1003	DRV1003	VEH2897
3	DRV1004	NaN	NaN
4	DRV1005	DRV1005	VEH2569

We then used both driver and vehicle IDs to copy ride IDS

```
# Merge Ride_Fact_df with ridedf to get RideID
Ride_Fact_df = pd.merge(Ride_Fact_df, ridedf[['RideID', 'DriverID', 'VehicleID']], on=['DriverID', 'VehicleID'], how='left')

Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID
0	DRV1000	NaN	NaN
1	DRV1002	NaN	NaN
2	DRV1003	VEH2897	RIDE32434
3	DRV1004	NaN	NaN
4	DRV1005	VEH2569	RIDE46820

Dropped the rows with missing ids as those rides don't exist and thus don't benefit our analysis

```
#Dropping missing val columns as not much will be affected
Ride_Fact_df = Ride_Fact_df.dropna(subset=['RideID','VehicleID'])
print(Ride_Fact_df.isnull().sum())
```

```
DriverID     0
VehicleID    0
RideID       0
dtype: int64
```

```
Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID
2	DRV1003	VEH2897	RIDE32434
4	DRV1005	VEH2569	RIDE46820
5	DRV1005	VEH2569	RIDE47442
6	DRV1006	VEH3879	RIDE32756
7	DRV1006	VEH3879	RIDE43406

Copying dates into the fact table, these will later be dropped after copying date IDs using them

```
# Merge Ride_Fact_df with paymentdf to get PaymentDate
Ride_Fact_df = pd.merge(Ride_Fact_df, paymentdf[['RideID', 'PaymentDate']], on='RideID', how='left')

# Rename the 'PaymentDate' column to 'Date'
Ride_Fact_df = Ride_Fact_df.rename(columns={'PaymentDate': 'Date'})

# Display the head of Ride_Fact_df
Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	Date
0	DRV1003	VEH2897	RIDE32434	NaT
1	DRV1005	VEH2569	RIDE46820	NaT
2	DRV1005	VEH2569	RIDE47442	2002-08-16
3	DRV1006	VEH3879	RIDE32756	NaT
4	DRV1006	VEH3879	RIDE43406	NaT

Copying date IDs using date values

```python
# Merge Ride_Fact_df with DimDate_df based on the 'Date' column
Ride_Fact_df = pd.merge(Ride_Fact_df, DimDate_df[['Date', 'DateID']], on='Date', how='left')

# Drop the 'Date' column from Ride_Fact_df
Ride_Fact_df = Ride_Fact_df.drop(columns=['Date'])

# Display the head of the updated Ride_Fact_df
print(Ride_Fact_df.head())
```

```
  DriverID VehicleID     RideID  DateID
0  DRV1005   VEH2569  RIDE47442   10414
1  DRV1014   VEH4391  RIDE59304   17564
2  DRV1014   VEH4391  RIDE71589     348
3  DRV1014   VEH4391  RIDE71589    2751
4  DRV1014   VEH4391  RIDE71589    8339
```

The data had some errors having multiple instances of the same ride on different dates so dropped those

```python
# Drop rows where 'DriverID', 'VehicleID', and 'RideID' are duplicated
Ride_Fact_df = Ride_Fact_df.drop_duplicates(subset=['DriverID', 'VehicleID', 'RideID'], keep='first')
Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	DateID
0	DRV1005	VEH2569	RIDE47442	10414
1	DRV1014	VEH4391	RIDE59304	17564
2	DRV1014	VEH4391	RIDE71589	348
5	DRV1015	VEH2449	RIDE41586	13268
6	DRV1015	VEH2449	RIDE60886	15884

Copied Amount column from payment data-frame into fact table as a fact,

```python
# Rename the 'PaymentDate' column in paymentdf to 'Date'
paymentdf = paymentdf.rename(columns={'PaymentDate': 'Date'})

# Merge Ride_Fact_df with paymentdf based on 'RideID' and 'Date'
Ride_Fact_df = pd.merge(Ride_Fact_df, paymentdf[['RideID', 'Date', 'Amount']], on=['RideID', 'Date'], how='left')

# Display the updated Ride_Fact_df
Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	DateID	Date	Amount
0	DRV1005	VEH2569	RIDE47442	10414	2002-08-16	23.87
1	DRV1014	VEH4391	RIDE59304	17564	2024-11-30	53.76
2	DRV1014	VEH4391	RIDE71589	348	1971-01-29	28.73
3	DRV1015	VEH2449	RIDE41586	13268	2011-06-22	76.15
4	DRV1015	VEH2449	RIDE60886	15884	2019-08-30	63.23

Copied capacity, pickup time and dropoff time into fact table, capacity was later shifted back into vehicle dimension table to act as a dimension, as the column only had discrete values and could act better as dimensional attribute.

```
# Merge Ride_Fact_df with ridedf to get PickupTime and DropoffTime
Ride_Fact_df = pd.merge(Ride_Fact_df, ridedf[['RideID', 'PickupTime', 'DropoffTime']], on='RideID', how='left')

Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	DateID	Amount	Capacity	PickupTime	DropoffTime
0	DRV1005	VEH2569	RIDE47442	10414	23.87	6	1971-03-31	1973-05-10
1	DRV1014	VEH4391	RIDE59304	17564	53.76	3	1972-12-22	1978-11-24
2	DRV1014	VEH4391	RIDE71589	348	28.73	3	2006-02-23	1992-02-24
3	DRV1015	VEH2449	RIDE41586	13268	76.15	5	1978-04-03	2007-06-18
4	DRV1015	VEH2449	RIDE60886	15884	63.23	5	2003-10-14	2008-10-19

Major data change

As pickup and dropoff times didn't make sense as they were dates, we derived a new fact "Duration" by subtracting the year values from both dates, we assume duration is in minutes and represents ride duration

```
[ ] # Calculate the duration and assign it to a new column
    Ride_Fact_df['Duration'] = abs(Ride_Fact_df['DropoffTime'].dt.year - Ride_Fact_df['PickupTime'].dt.year)

    # Display the updated Ride_Fact_df
    Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	DateID	Amount	Capacity	PickupTime	DropoffTime	Duration
0	DRV1005	VEH2569	RIDE47442	10414	23.87	6	1971-03-31	1973-05-10	2
1	DRV1014	VEH4391	RIDE59304	17564	53.76	3	1972-12-22	1978-11-24	6
2	DRV1014	VEH4391	RIDE71589	348	28.73	3	2006-02-23	1992-02-24	14
3	DRV1015	VEH2449	RIDE41586	13268	76.15	5	1978-04-03	2007-06-18	29
4	DRV1015	VEH2449	RIDE60886	15884	63.23	5	2003-10-14	2008-10-19	5

New fact, RatePerMinute generated via amount/duration and represents per minute fare rate

```
# Calculate 'PerMinuteRate'
Ride_Fact_df['PerMinuteRate'] = (Ride_Fact_df['Amount'] / Ride_Fact_df['Duration']).round(2)
Ride_Fact_df.head()
```

	DriverID	VehicleID	RideID	DateID	Amount	Capacity	Duration	PerMinuteRate
0	DRV1005	VEH2569	RIDE47442	10414	23.87	6	2	11.94
1	DRV1014	VEH4391	RIDE59304	17564	53.76	3	6	8.96
2	DRV1014	VEH4391	RIDE71589	348	28.73	3	14	2.05
3	DRV1015	VEH2449	RIDE41586	13268	76.15	5	29	2.63
4	DRV1015	VEH2449	RIDE60886	15884	63.23	5	5	12.65

Renaming fact columns

```
[ ] # Rename columns in Ride_Fact_df
    Ride_Fact_df = Ride_Fact_df.rename(columns={
        'DriverID': 'DimDriverID',
        'VehicleID': 'DimVehicleID',
        'RideID': 'DimRideID',
        'DateID': 'DimDateID',
        'Amount': 'RideFare',
        'Capacity': 'VehicleCapacity',
        'Duration': 'RideDuration'
    })

    Ride_Fact_df.head()
```

Exporting dimension tables and fact table into csv

```
from google.colab import files
Ride_Fact_df.to_csv('Fact_Ride.csv', index=False)
files.download('Fact_Ride.csv')
DimDriver_df.to_csv('DimDriver.csv', index=False)
files.download('DimDriver.csv')
DimVehicle_df.to_csv('DimVehicle.csv', index=False)
files.download('DimVehicle.csv')
DimRide_df.to_csv('DimRide.csv', index=False)
files.download('DimRide.csv')
DimDate_df.to_csv('DimDate.csv', index=False)
files.download('DimDate.csv')
```

Creating a Warehouse in BigQuery:

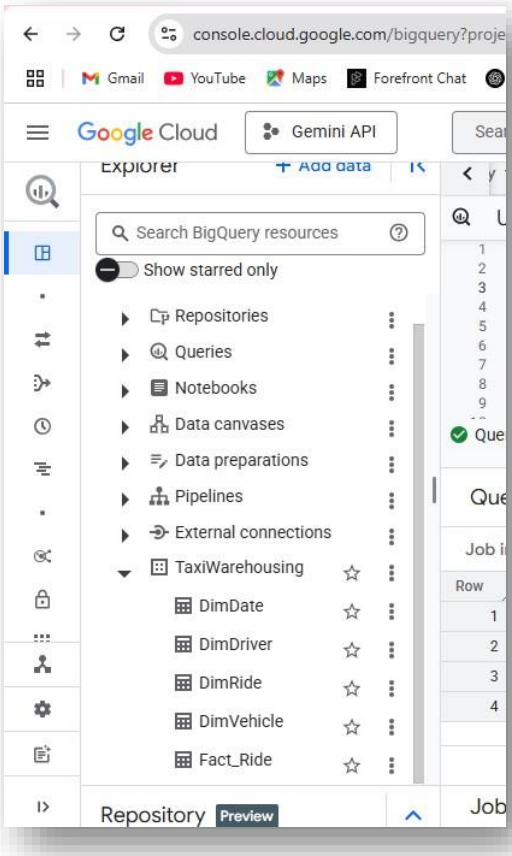
Due to issues with payment verification, we were limited to sandbox mode

Loading data into BigQuery

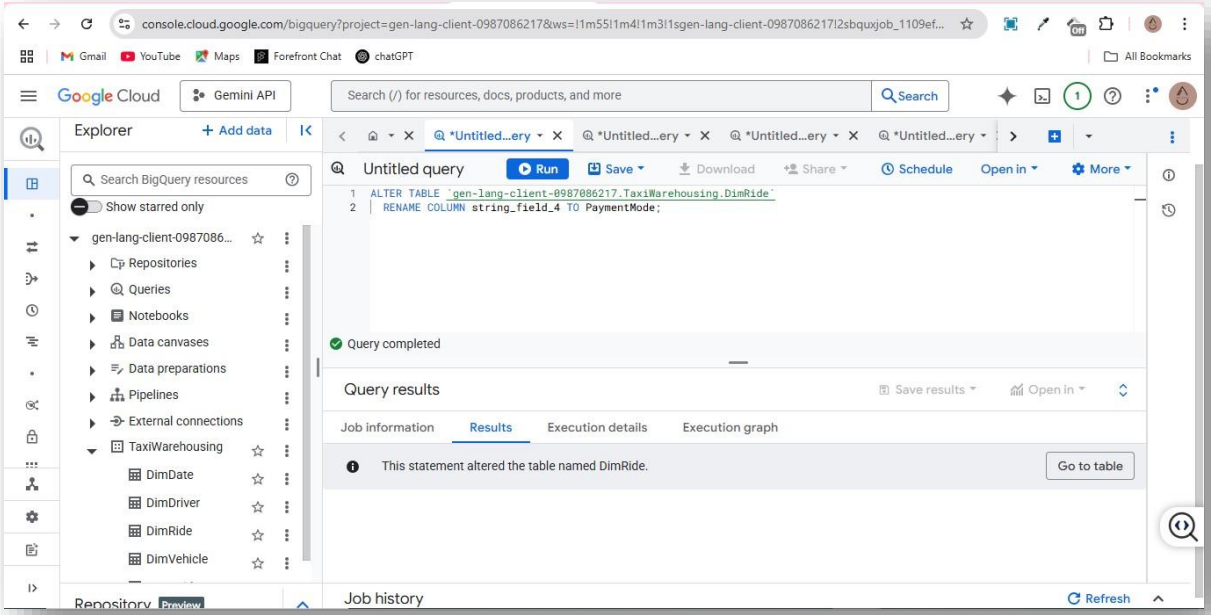
Data was directly loaded from local files as cvs and tables were also created using BigQuery's interface, schema creation was automated

Once all tables were loaded, they could be viewed as such

Here TaxiWarehousing represents our warehouse containing all the tables



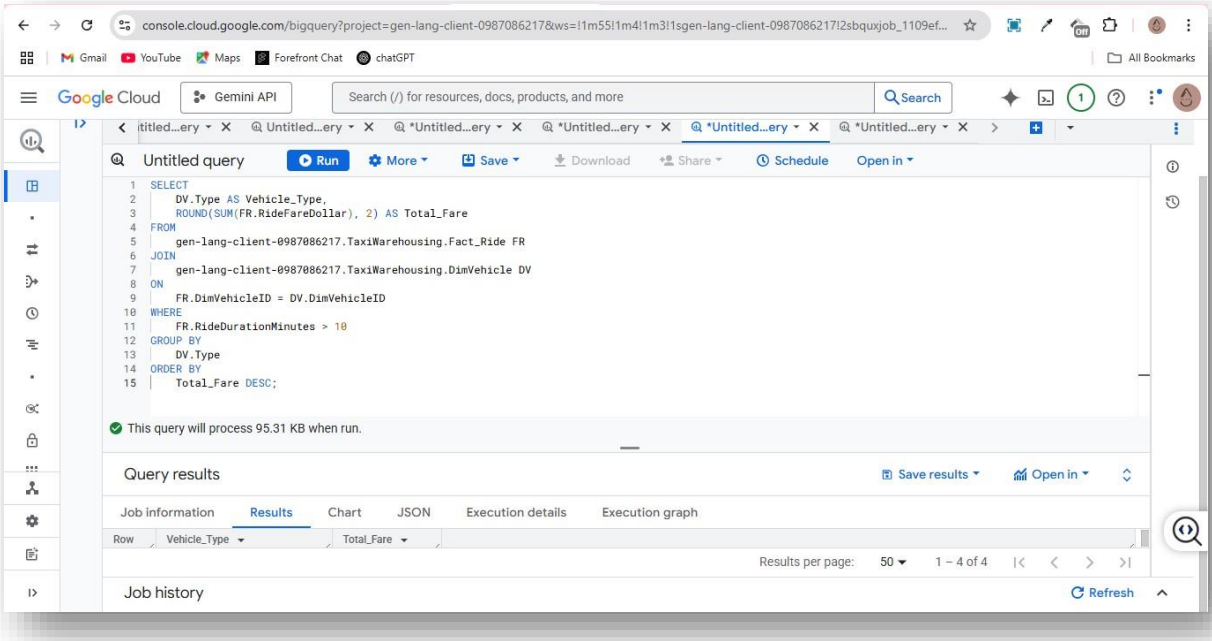
Some tables had issues with column names so they had to be manually altered



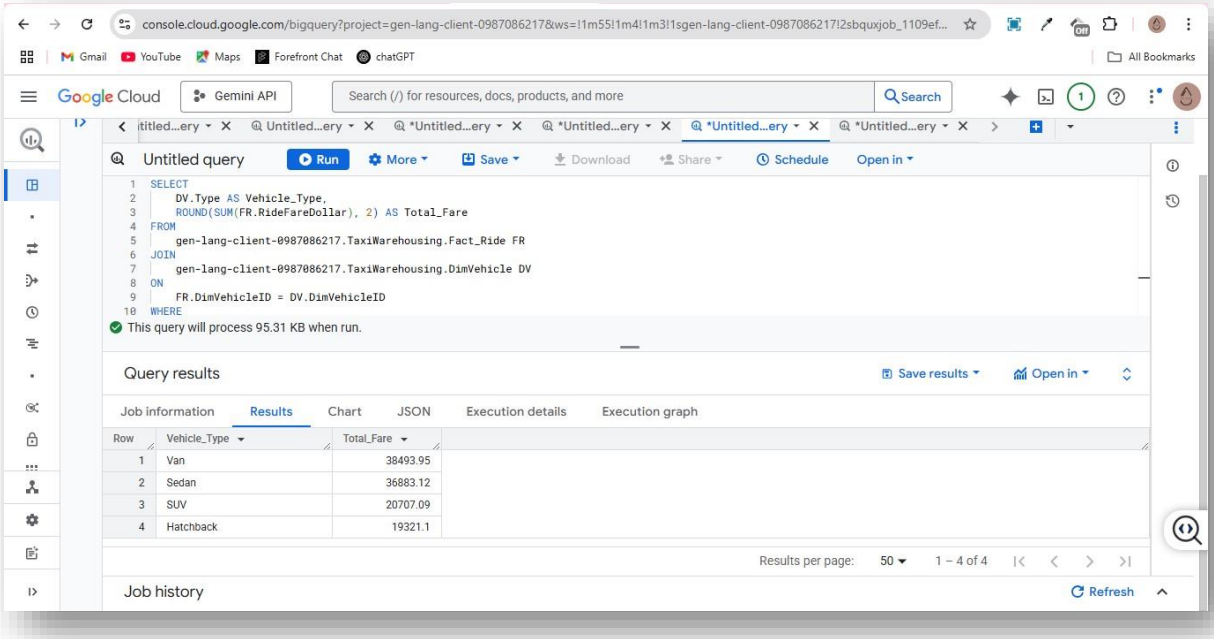
Sample Queries:

Query1:

What is the total revenue generated by all types of cars for rides that were more than 10 minutes long?

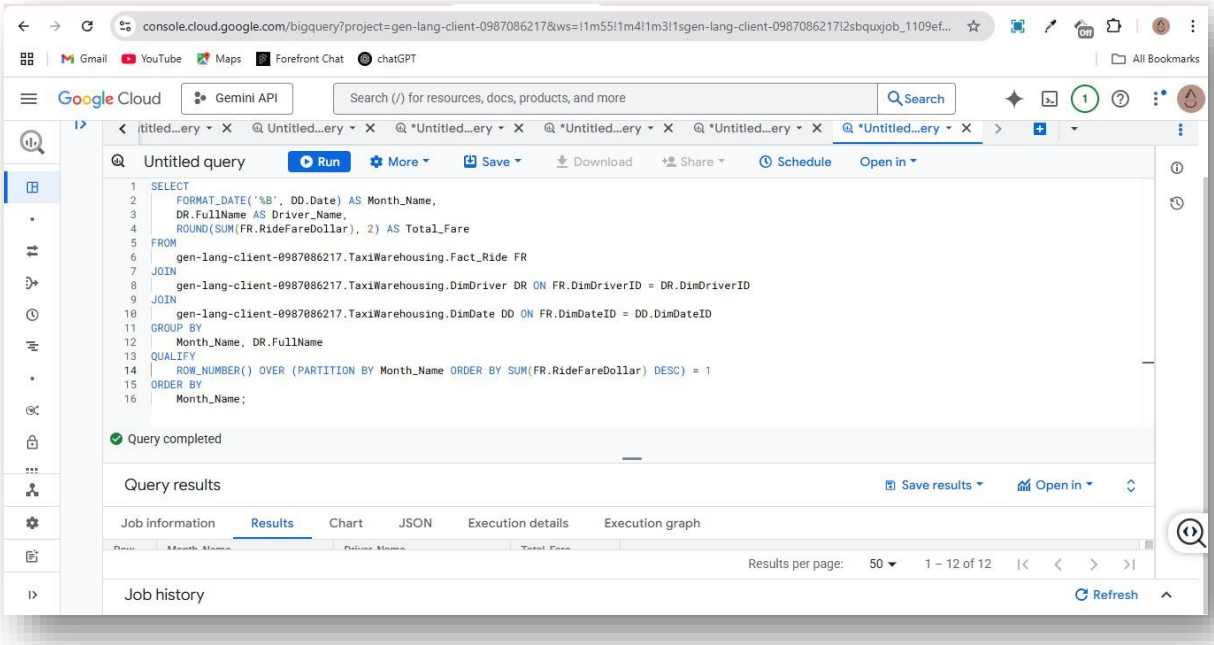


Query Result

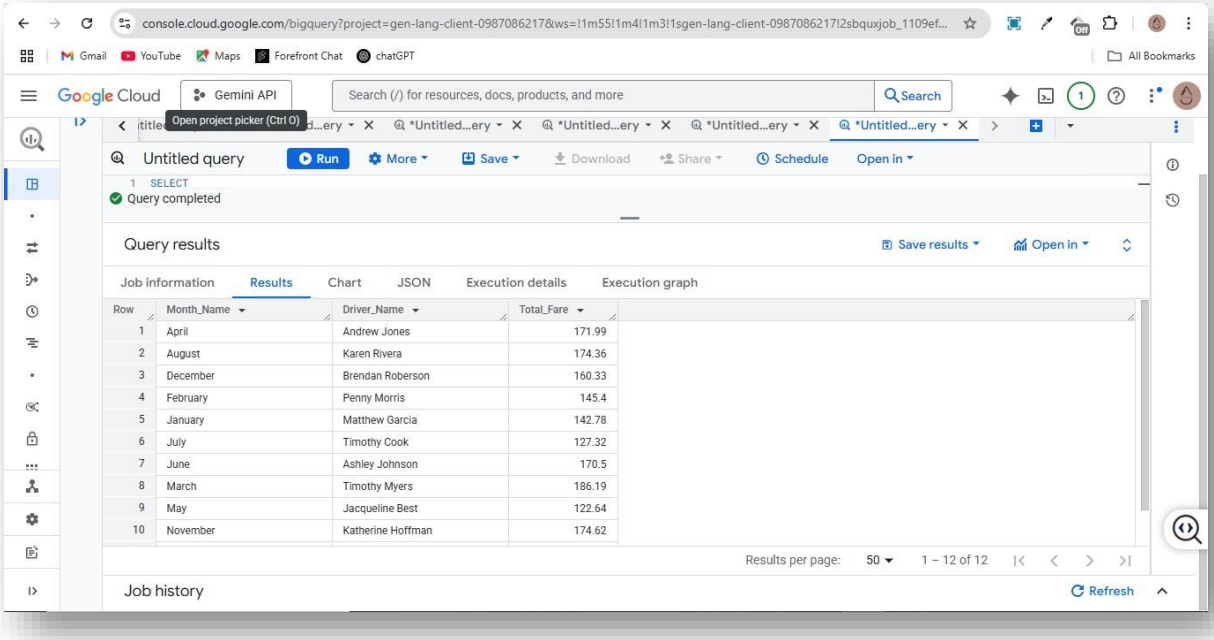


Query2:

What are the top earning drivers per month based on Ride Fares in Houston?

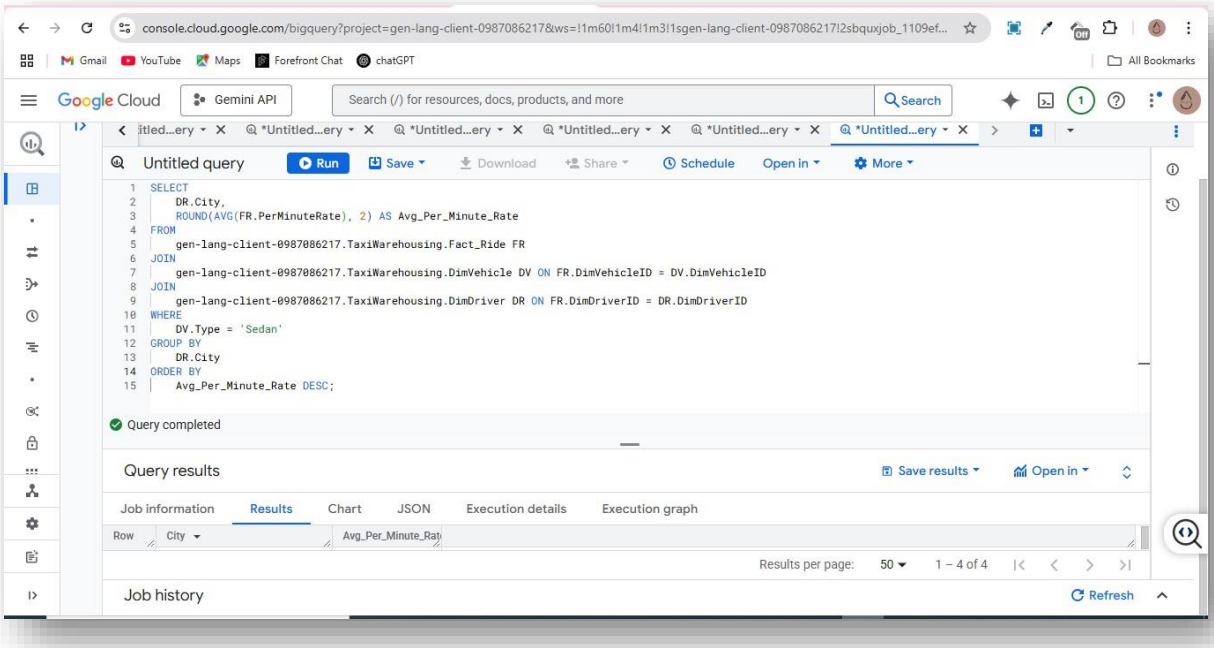


Query Result

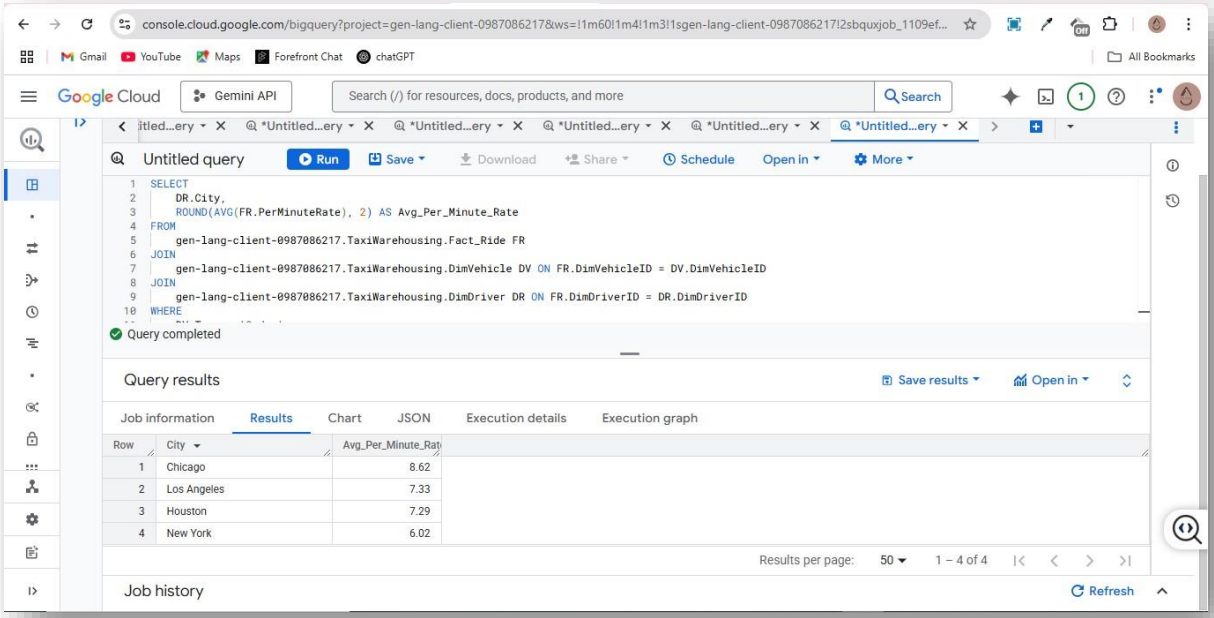


Query3:

What is the average PerMinuteRate for Sedans in each city? Is it different for each city?

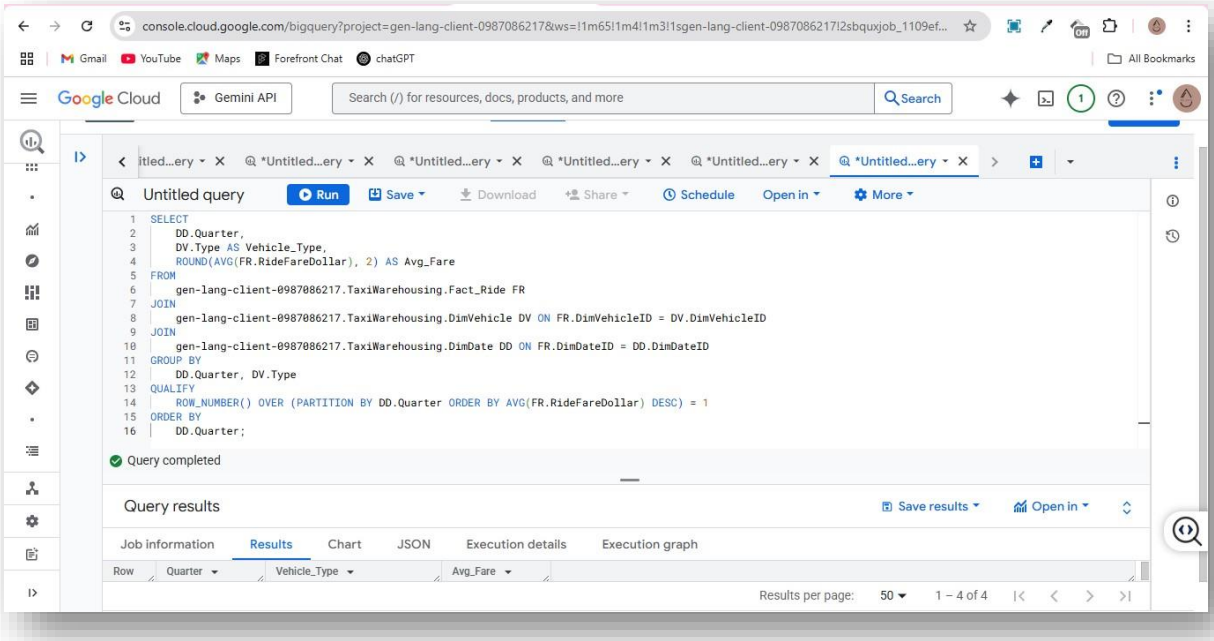


Query Result

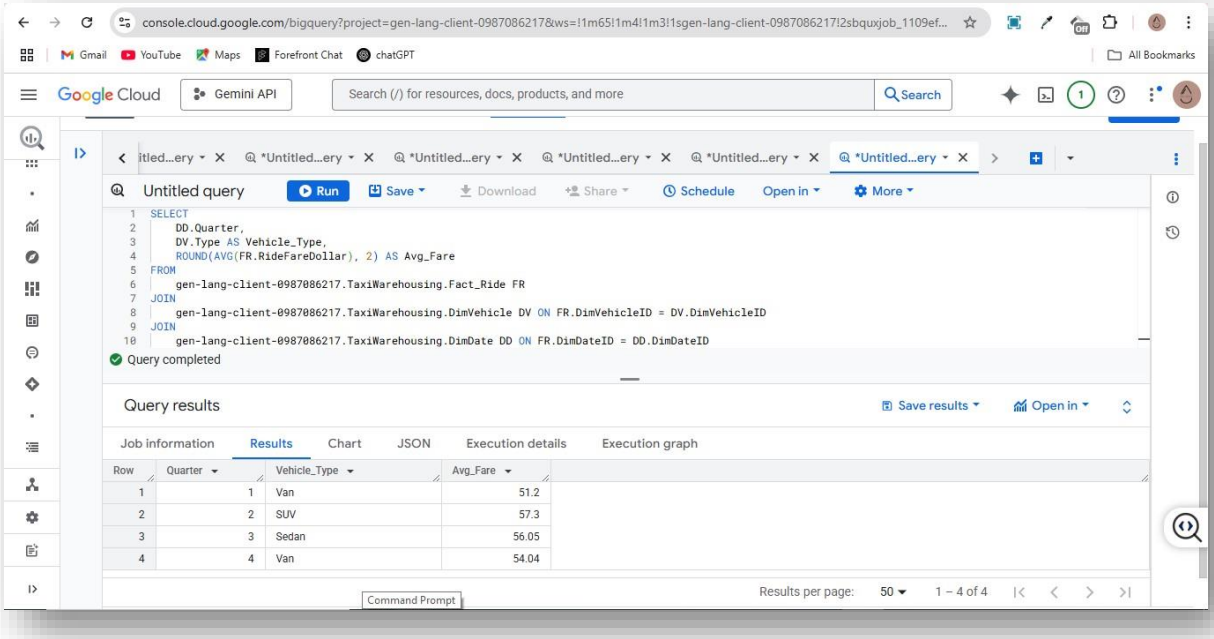


Query4:

What is the vehicle tier that has the highest average fare in each quarter overall?



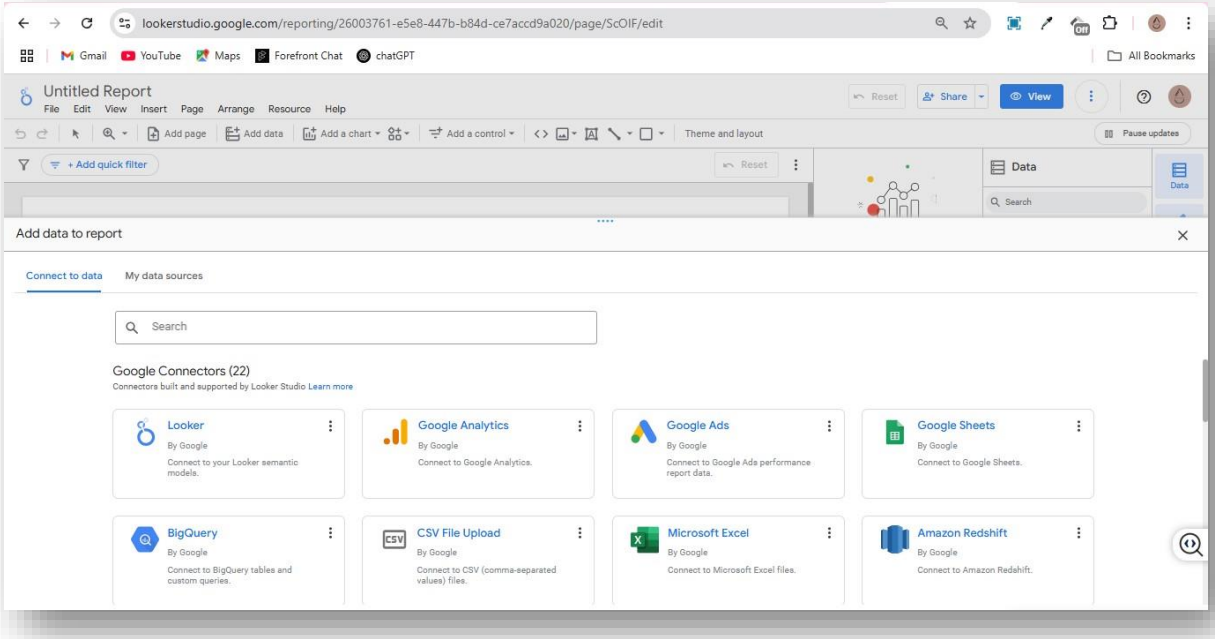
Query Result

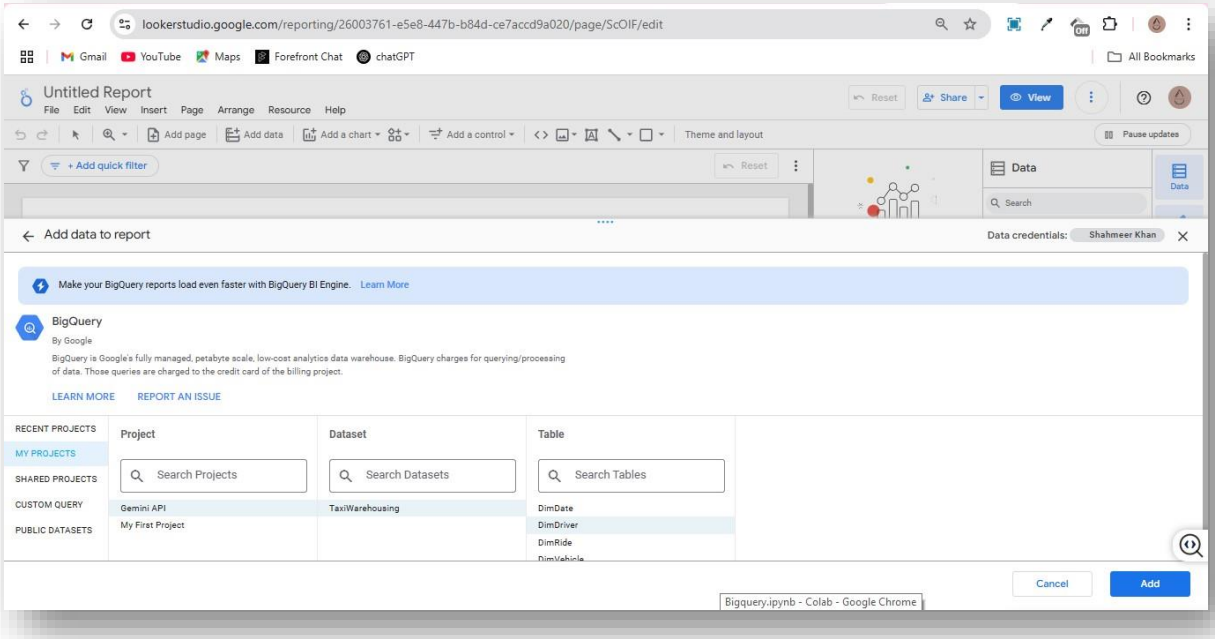


Dashboarding

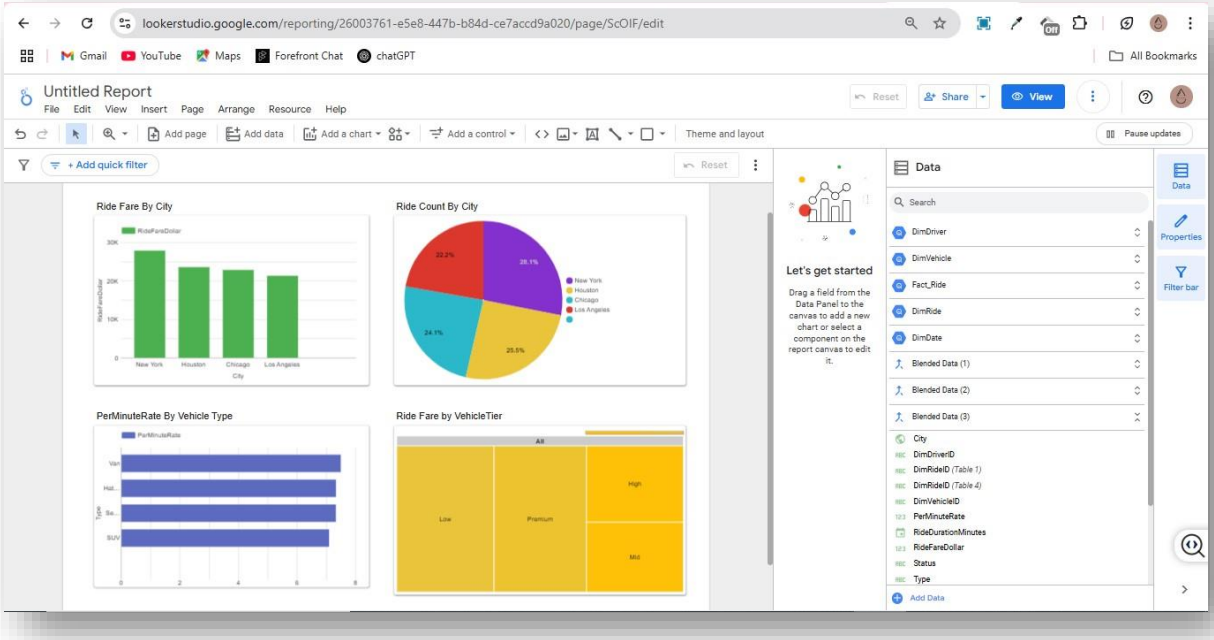
As BigQuery didn't have a built-in dashboarding feature, we looked to LookerStudio which also being a Google solution could easily be integrated with our BigQuery warehouse

Integration with BigQuery





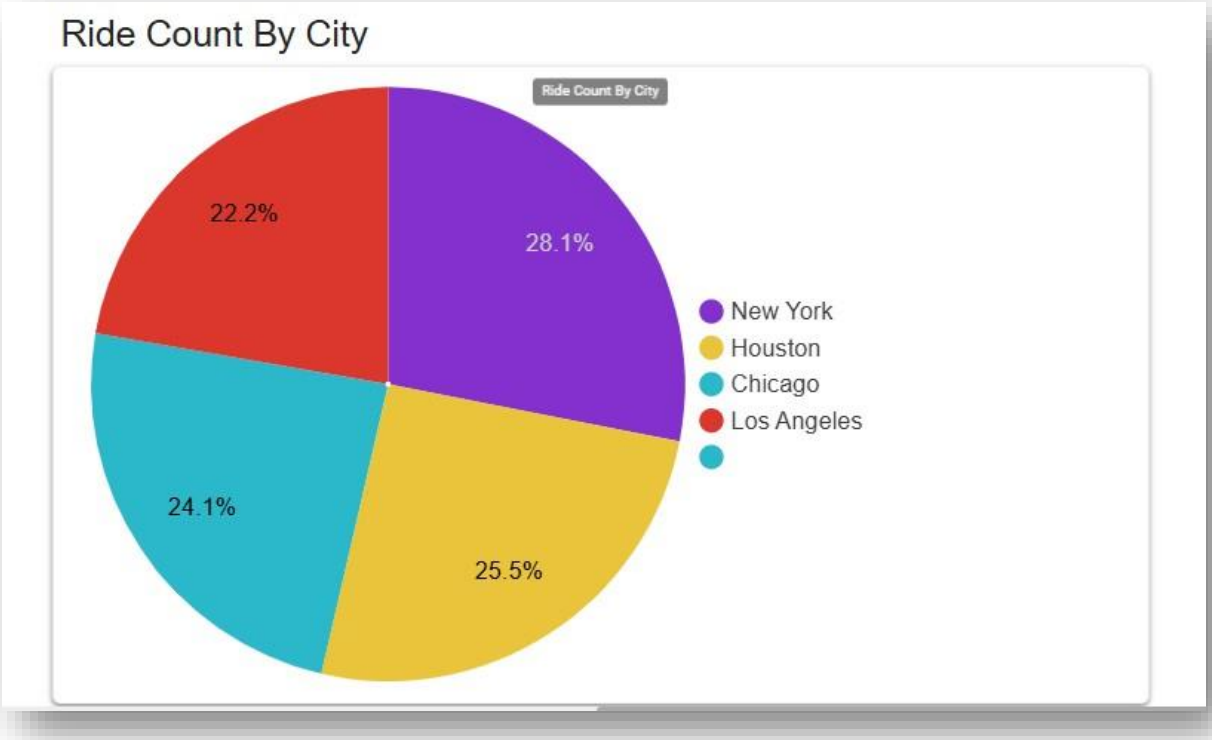
Dashboard on LookerStudio



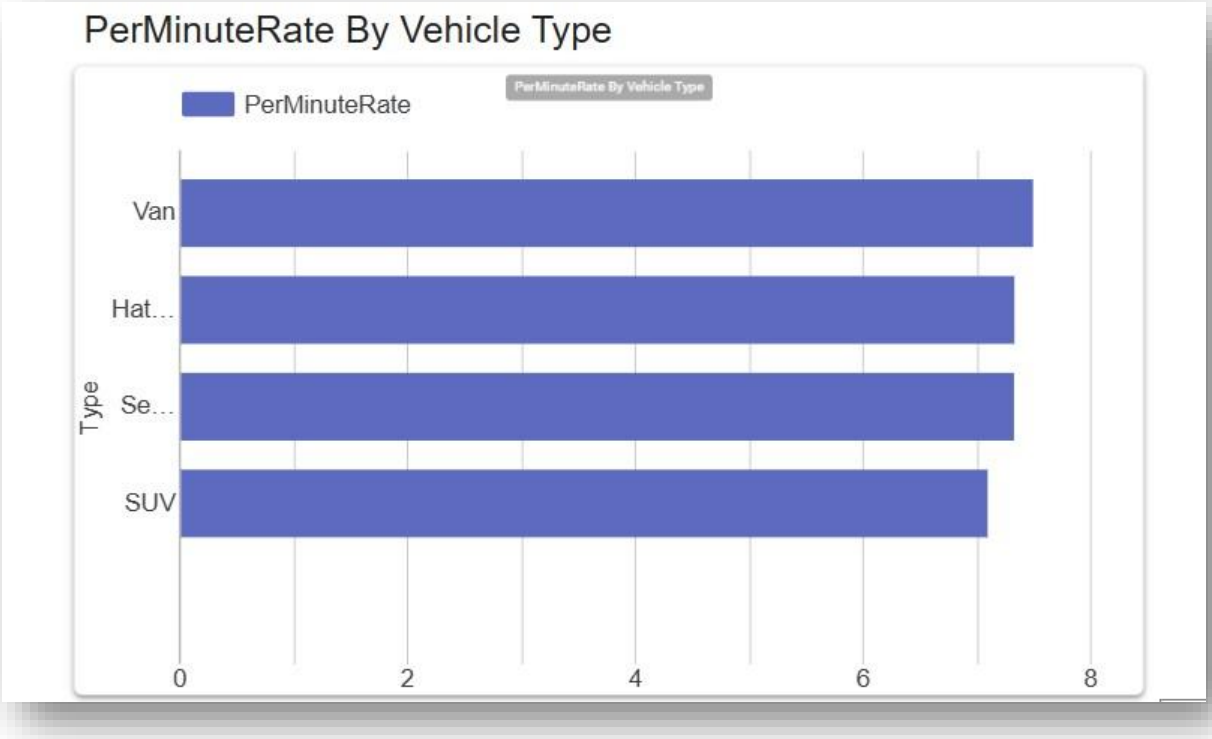
Individual charts for better understanding

Analysis of Ride fare totals by city

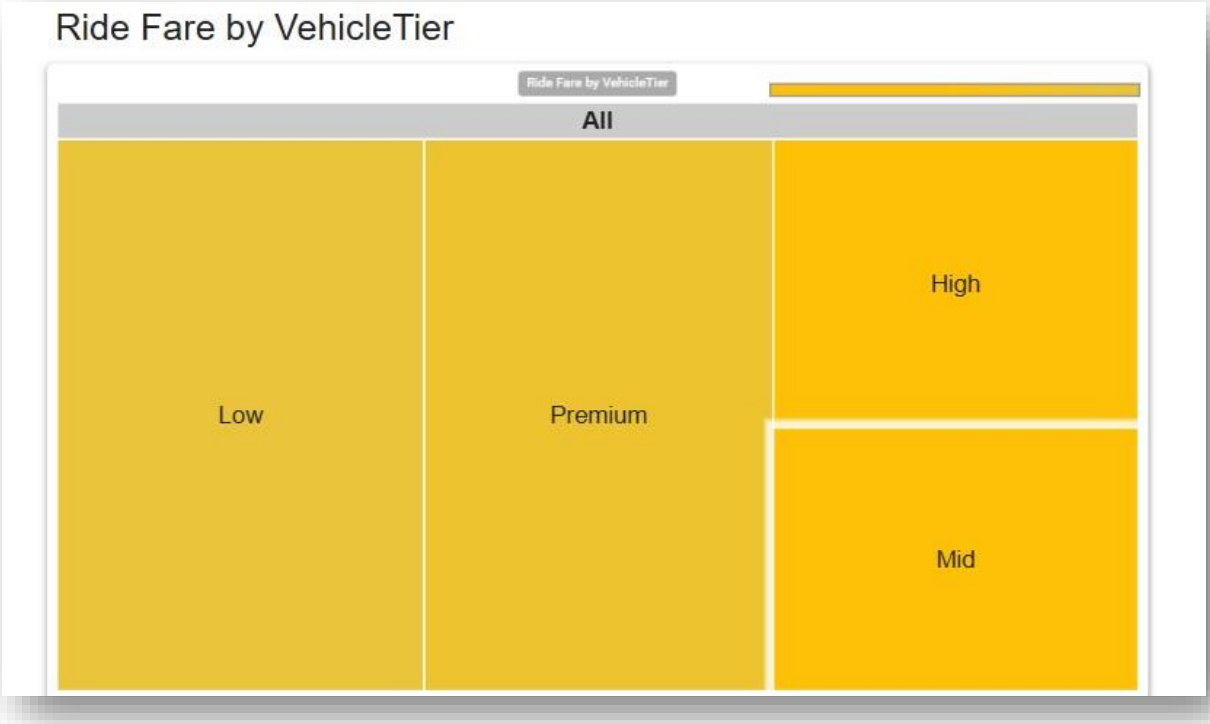
Count of Rides by City



Analysis of average PerMinuteRate based on Vehicle Types



Analysis of Ride Fare totals based on Vehicle Tiers


Ride Fare by VehicleTier
Ride Fare by VehicleTier
All
Low
Premium
High
Mid