CAPSTONE PROJECT

UBER TRIPS ANALYTICS

PRESENTED BY: PARTH MISHRA

PRESENTED TO : WS CUBE TECH

PROJECT OVERVIEW

Project Overview – NYC Taxi Trip Analytics and Predictive Modeling (2017–2020)

- This project analyzes four years of NYC Taxi trip data (2017–2020)
- uncover insights about passenger behavior, trip trends, route profitability, and tipping patterns.

- The workflow combines SQL for data processing, Power BI for visualization, and Machine Learning for predictive modeling.

Timeline

"Turning raw trip data into powerful insights, visuals, and predictions."

1 SQL

Advanced SQL queries and insights

2 Power BI

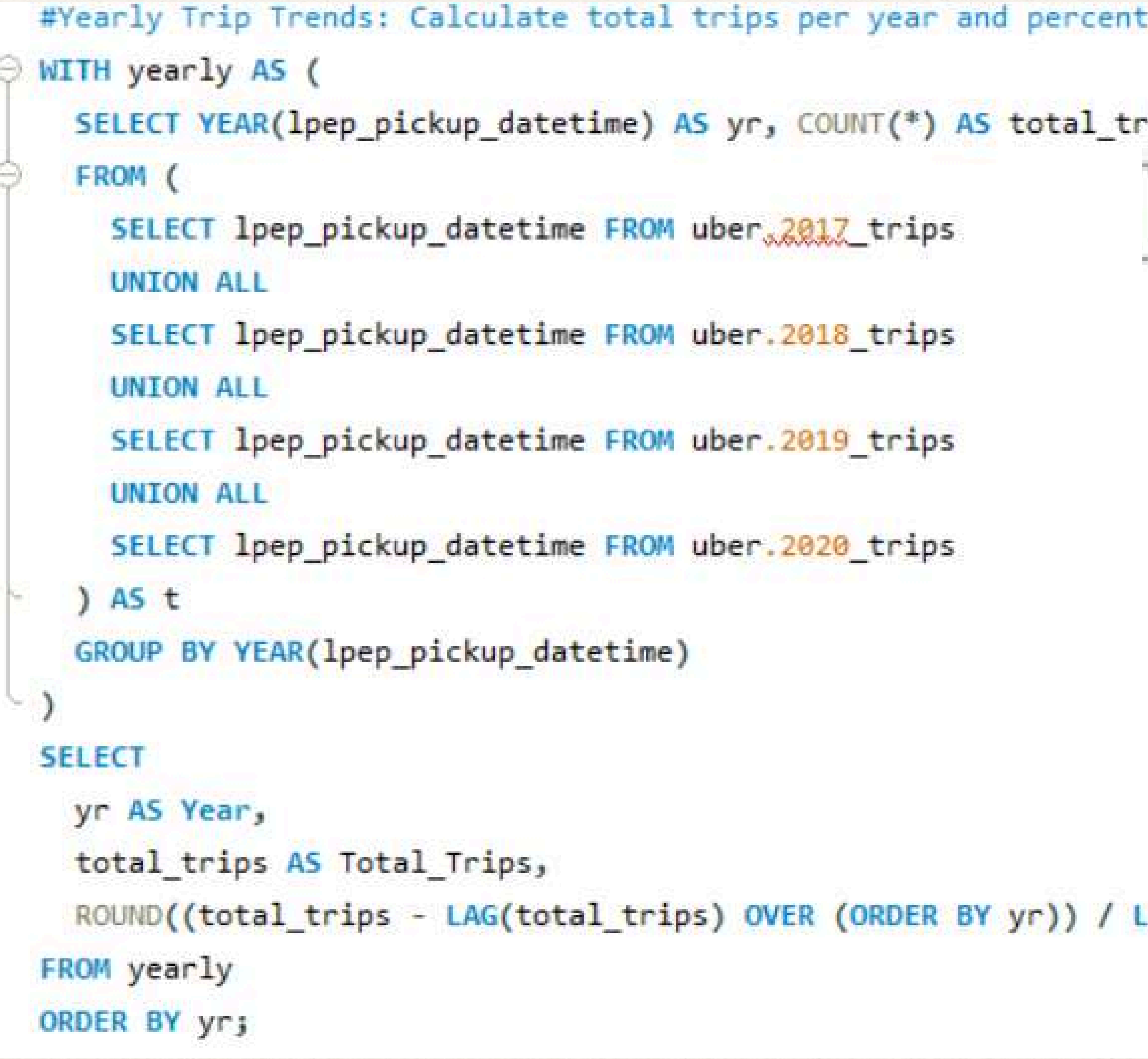
Visualizations of the data and find key figures

3 ML

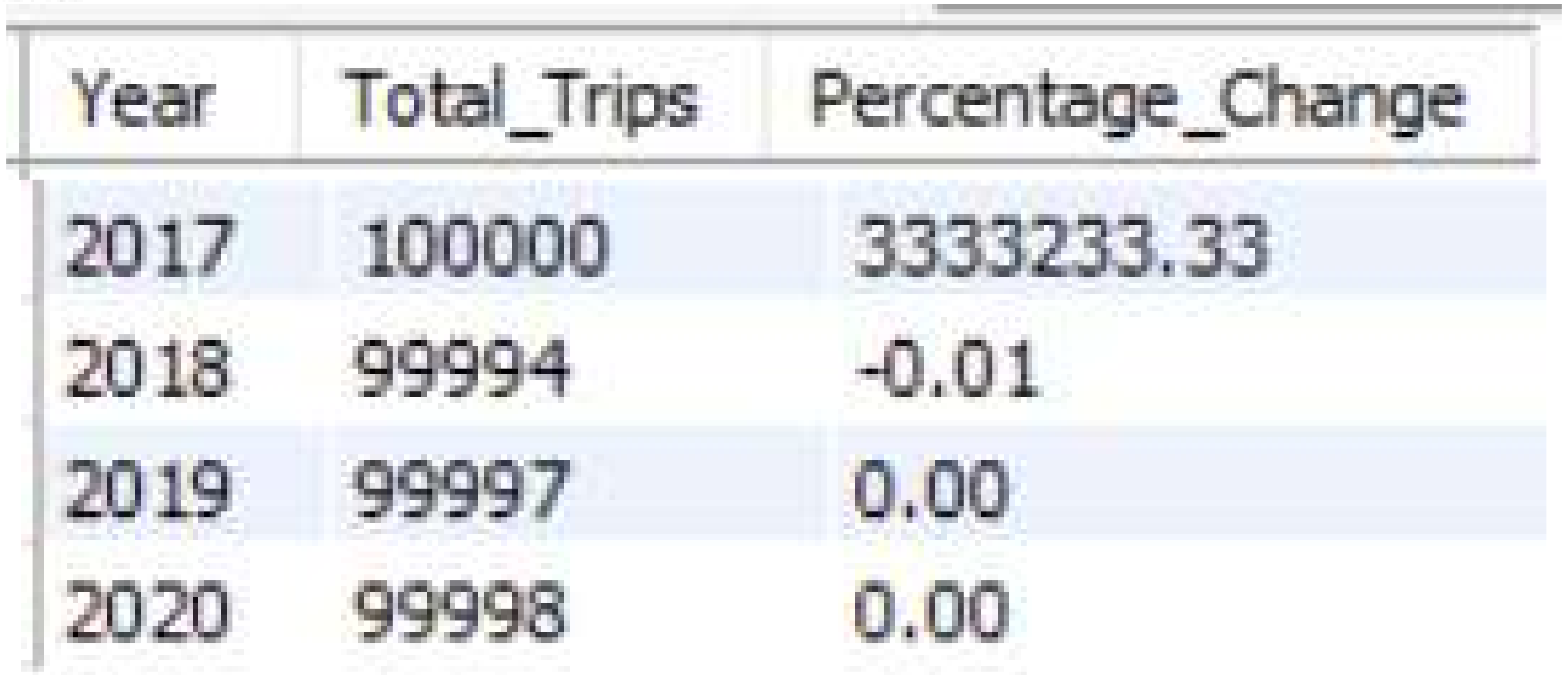
Predictive Modelling and Outcomes

Part 1 - Advanced SQL Queries Uber Taxi Operations Insights

1. **Yearly Trip Trends:** Calculate total trips per year and percentage change year-over-year.

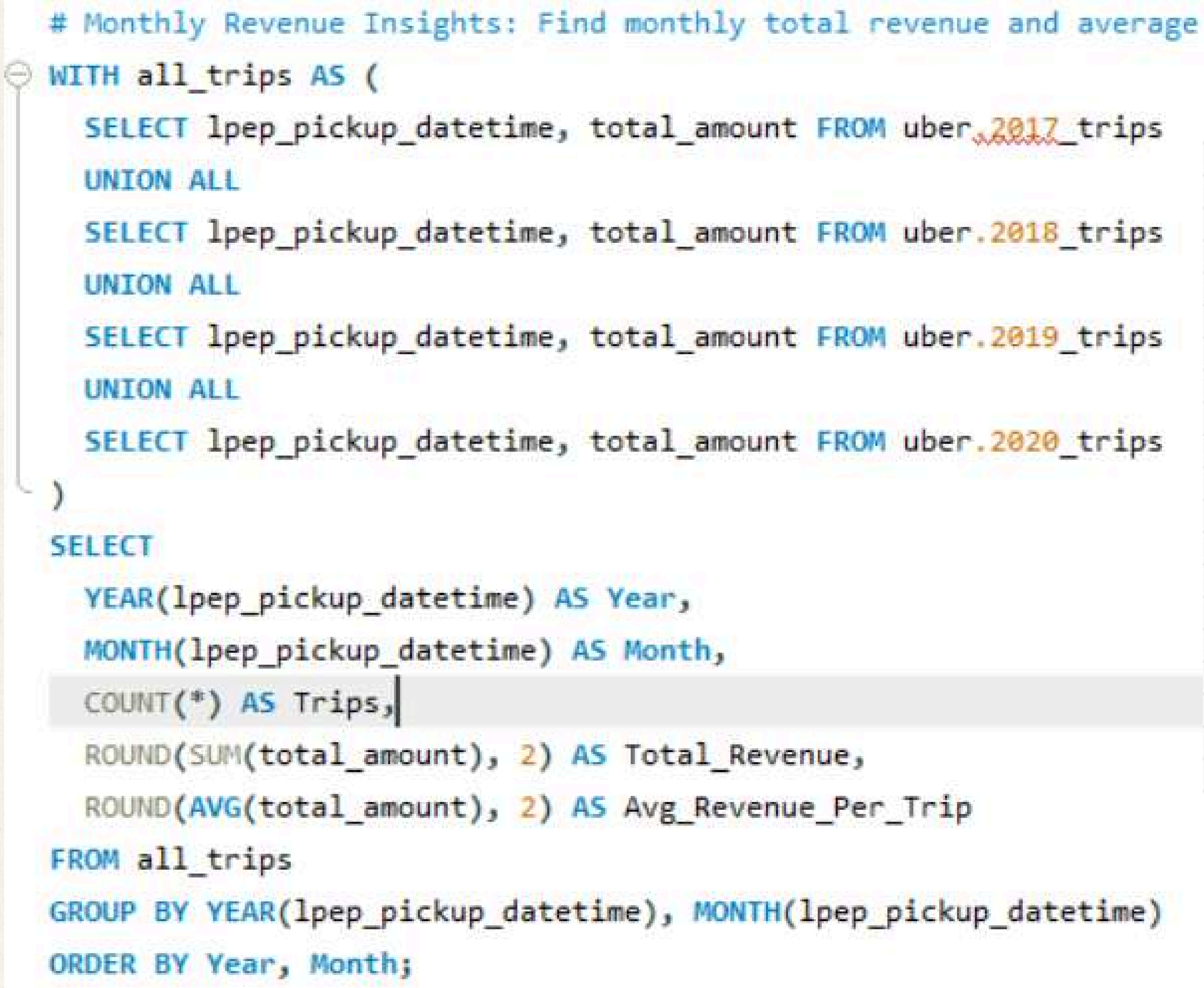


```
#Yearly Trip Trends: Calculate total trips per year and percentage change year-over-year.
WITH yearly AS (
  SELECT YEAR(lpep_pickup_datetime) AS yr, COUNT(*) AS total_trips
  FROM (
    SELECT lpep_pickup_datetime FROM uber.2017_trips
    UNION ALL
    SELECT lpep_pickup_datetime FROM uber.2018_trips
    UNION ALL
    SELECT lpep_pickup_datetime FROM uber.2019_trips
    UNION ALL
    SELECT lpep_pickup_datetime FROM uber.2020_trips
  ) AS t
  GROUP BY YEAR(lpep_pickup_datetime)
)
SELECT
  yr AS Year,
  total_trips AS Total_Trips,
  ROUND((total_trips - LAG(total_trips) OVER (ORDER BY yr)) / LAG(total_trips) OVER (ORDER BY yr) * 100, 2) AS Percentage_Chang
FROM yearly
ORDER BY yr;
```

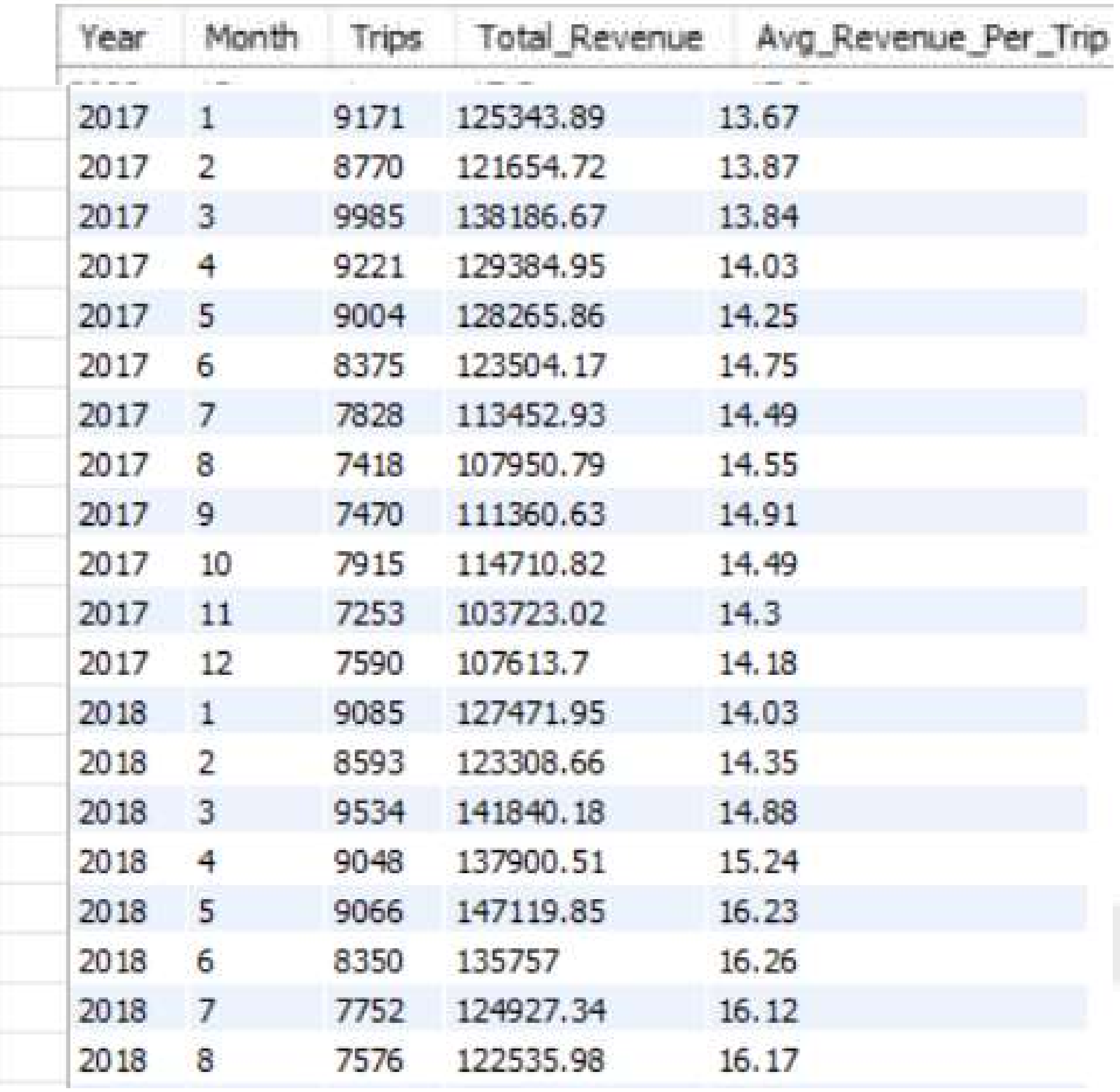


Year	Total_Trips	Percentage_Change
2017	100000	3333233.33
2018	99994	-0.01
2019	99997	0.00
2020	99998	0.00

2. **Monthly Revenue Insights:** Find monthly total revenue and average revenue per trip.

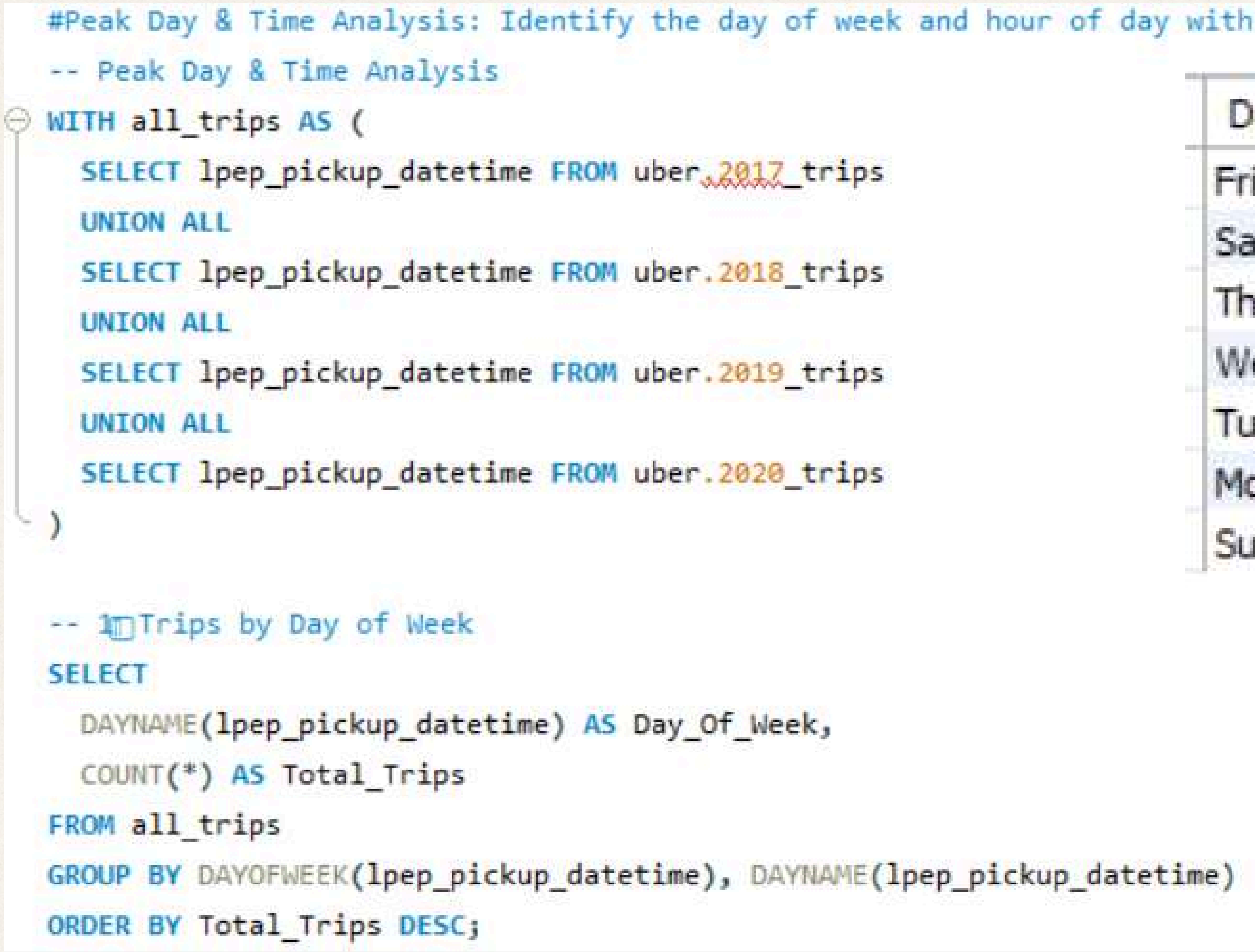


```
# Monthly Revenue Insights: Find monthly total revenue and average revenue per trip.
WITH all_trips AS (
  SELECT lpep_pickup_datetime, total_amount FROM uber.2017_trips
  UNION ALL
  SELECT lpep_pickup_datetime, total_amount FROM uber.2018_trips
  UNION ALL
  SELECT lpep_pickup_datetime, total_amount FROM uber.2019_trips
  UNION ALL
  SELECT lpep_pickup_datetime, total_amount FROM uber.2020_trips
)
SELECT
  YEAR(lpep_pickup_datetime) AS Year,
  MONTH(lpep_pickup_datetime) AS Month,
  COUNT(*) AS Trips,
  ROUND(SUM(total_amount), 2) AS Total_Revenue,
  ROUND(AVG(total_amount), 2) AS Avg_Revenue_Per_Trip
FROM all_trips
GROUP BY YEAR(lpep_pickup_datetime), MONTH(lpep_pickup_datetime)
ORDER BY Year, Month;
```

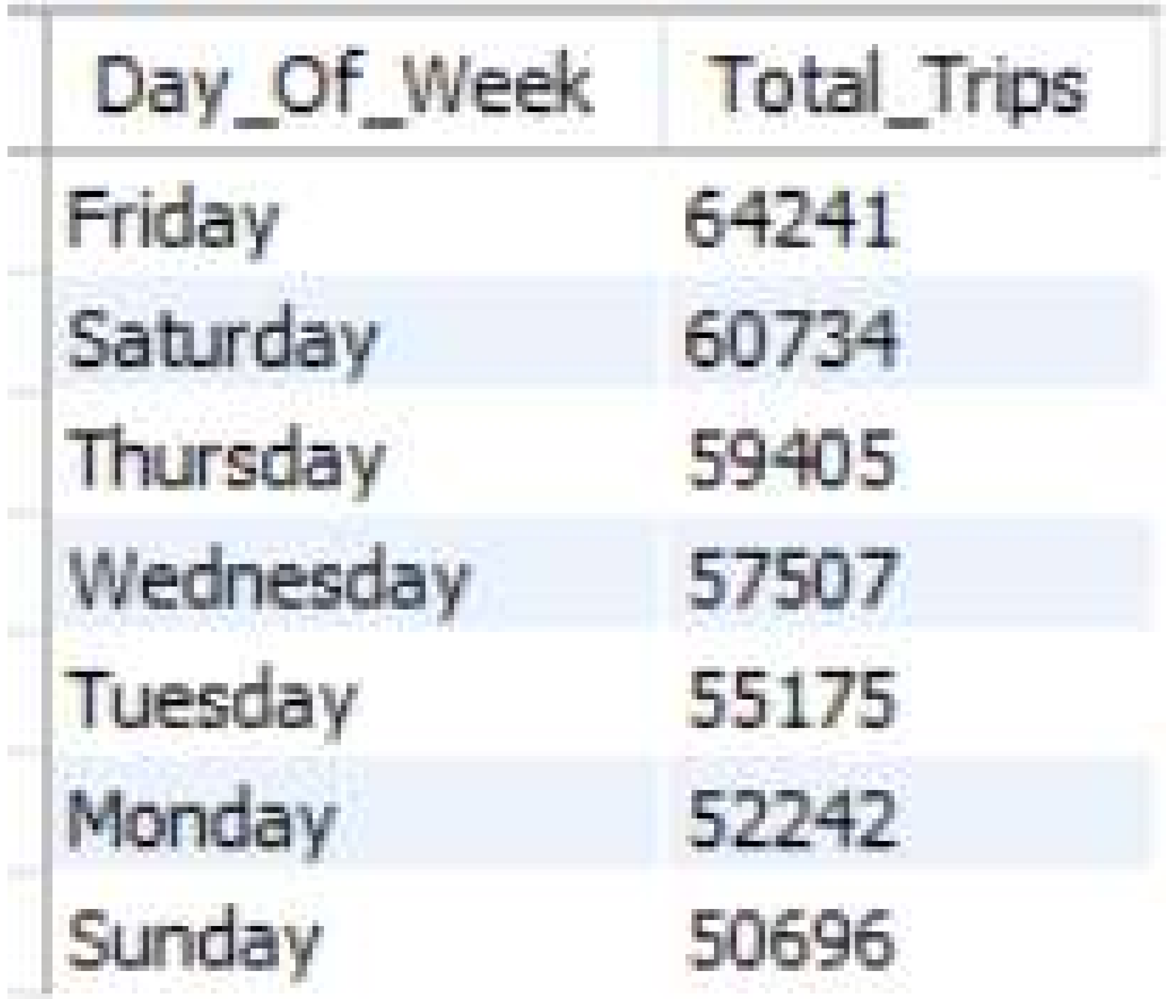


Year	Month	Trips	Total_Revenue	Avg_Revenue_Per_Trip
2017	1	9171	125343.89	13.67
2017	2	8770	121654.72	13.87
2017	3	9985	138186.67	13.84
2017	4	9221	129384.95	14.03
2017	5	9004	128265.86	14.25
2017	6	8375	123504.17	14.75
2017	7	7828	113452.93	14.49
2017	8	7418	107950.79	14.55
2017	9	7470	111360.63	14.91
2017	10	7915	114710.82	14.49
2017	11	7253	103723.02	14.3
2017	12	7590	107613.7	14.18
2018	1	9085	127471.95	14.03
2018	2	8593	123308.66	14.35
2018	3	9534	141840.18	14.88
2018	4	9048	137900.51	15.24
2018	5	9066	147119.85	16.23
2018	6	8350	135757	16.26
2018	7	7752	124927.34	16.12
2018	8	7576	122535.98	16.17

3. **Peak Day & Time Analysis:** Identify the day of week and hour of day with the highest average trip volumes.



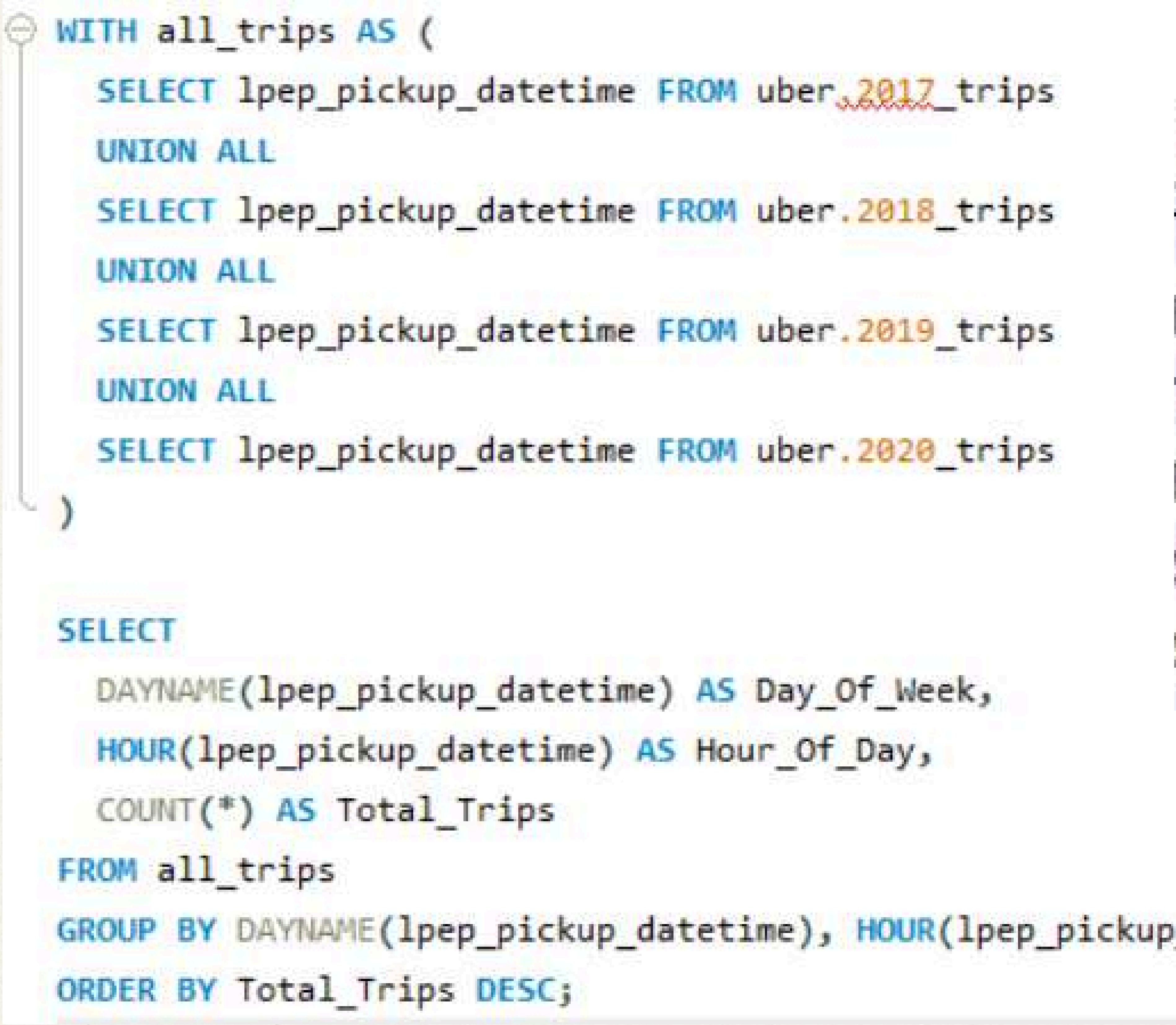
```
#Peak Day & Time Analysis: Identify the day of week and hour of day with the highest average trip volumes.
-- Peak Day & Time Analysis
WITH all_trips AS (
  SELECT lpep_pickup_datetime FROM uber.2017_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2018_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2019_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2020_trips
)

-- 1️⃣Trips by Day of Week
SELECT
  DAYNAME(lpep_pickup_datetime) AS Day_Of_Week,
  COUNT(*) AS Total_Trips
FROM all_trips
GROUP BY DAYOFWEEK(lpep_pickup_datetime), DAYNAME(lpep_pickup_datetime)
ORDER BY Total_Trips DESC;
```



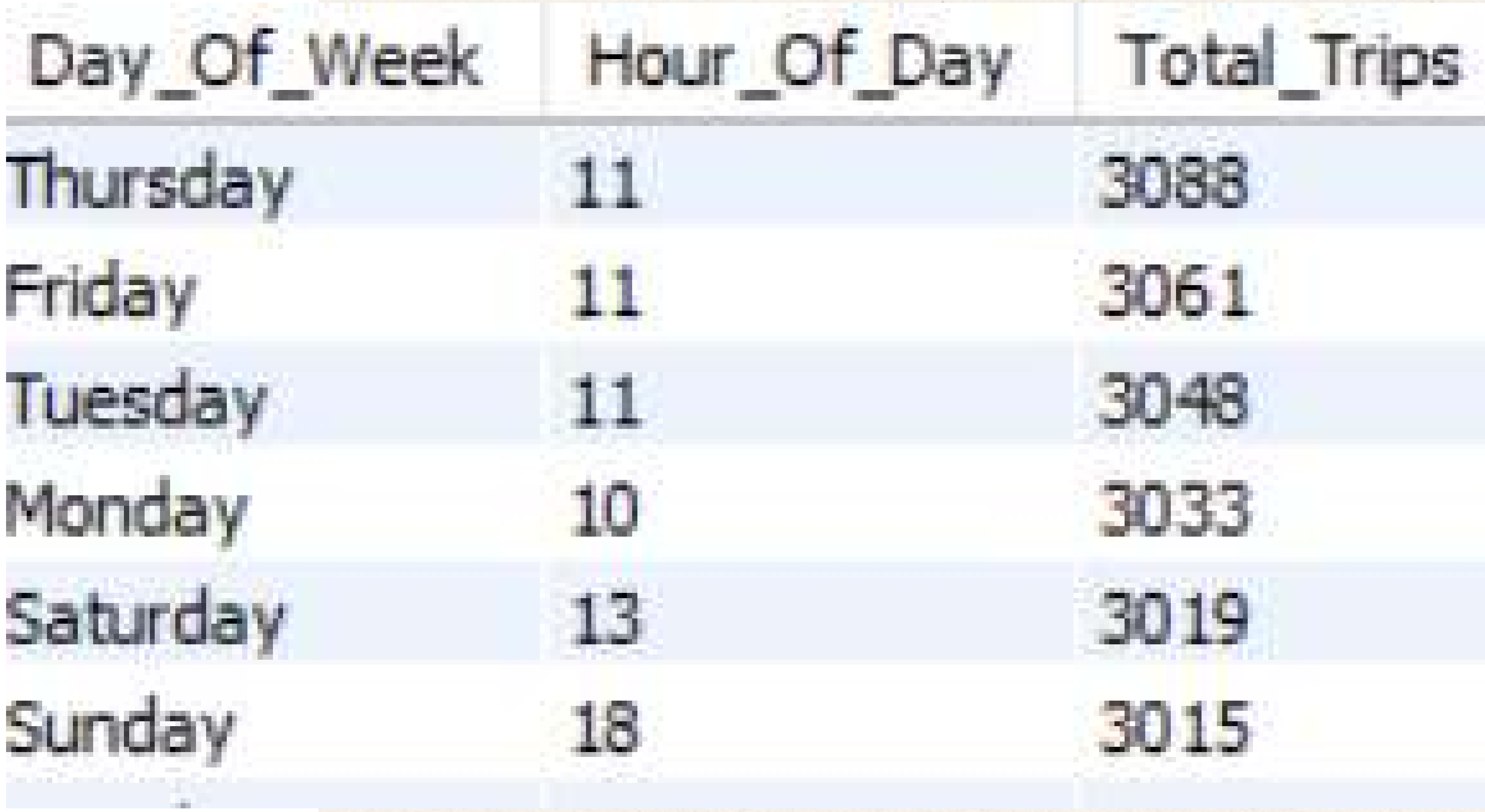
Day_Of_Week	Total_Trips
Friday	64241
Saturday	60734
Thursday	59405
Wednesday	57507
Tuesday	55175
Monday	52242
Sunday	50696

storage.googleapis.com

4. **Borough Trip Distribution:** Find the percentage of trips starting in each pickup_borough.



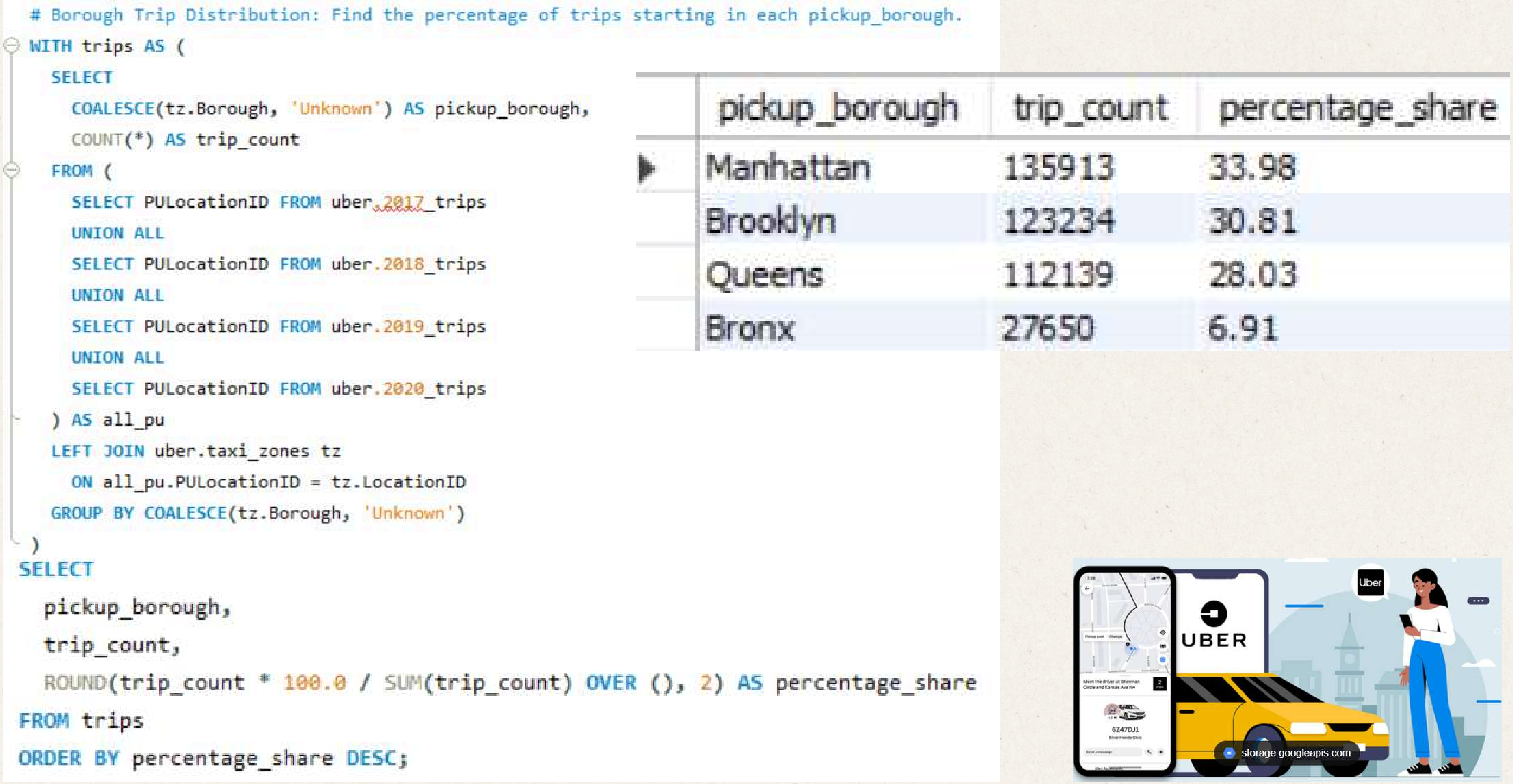
```
WITH all_trips AS (
  SELECT lpep_pickup_datetime FROM uber.2017_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2018_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2019_trips
  UNION ALL
  SELECT lpep_pickup_datetime FROM uber.2020_trips
)

SELECT
  DAYNAME(lpep_pickup_datetime) AS Day_Of_Week,
  HOUR(lpep_pickup_datetime) AS Hour_Of_Day,
  COUNT(*) AS Total_Trips
FROM all_trips
GROUP BY DAYNAME(lpep_pickup_datetime), HOUR(lpep_pickup_datetime)
ORDER BY Total_Trips DESC;
```



Day_Of_Week	Hour_Of_Day	Total_Trips
Thursday	11	3088
Friday	11	3061
Tuesday	11	3048
Monday	10	3033
Saturday	13	3019
Sunday	18	3015

storage.googleapis.com

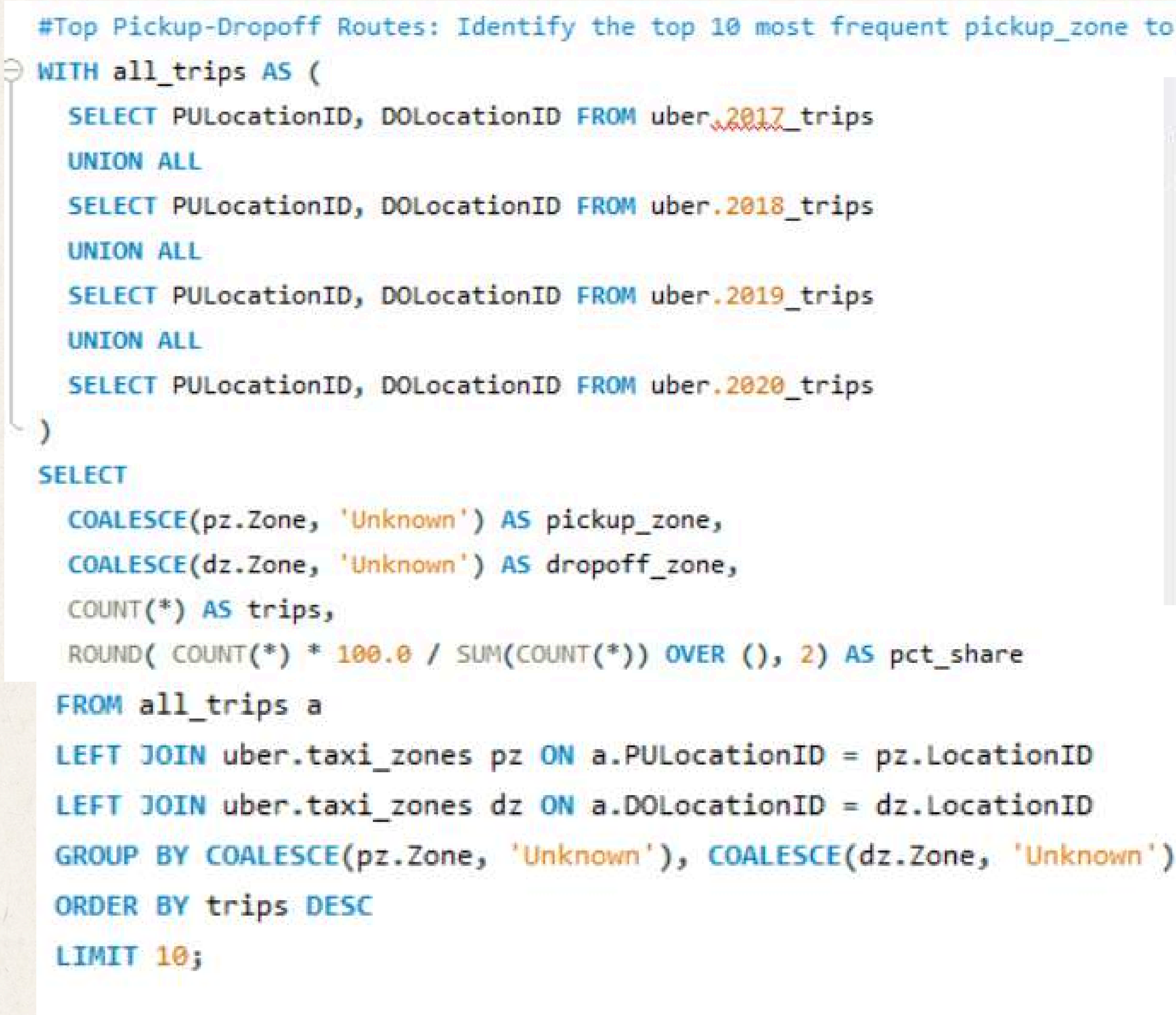
6. **Passenger Load Patterns:** Find the average passenger_count for trips in each borough.



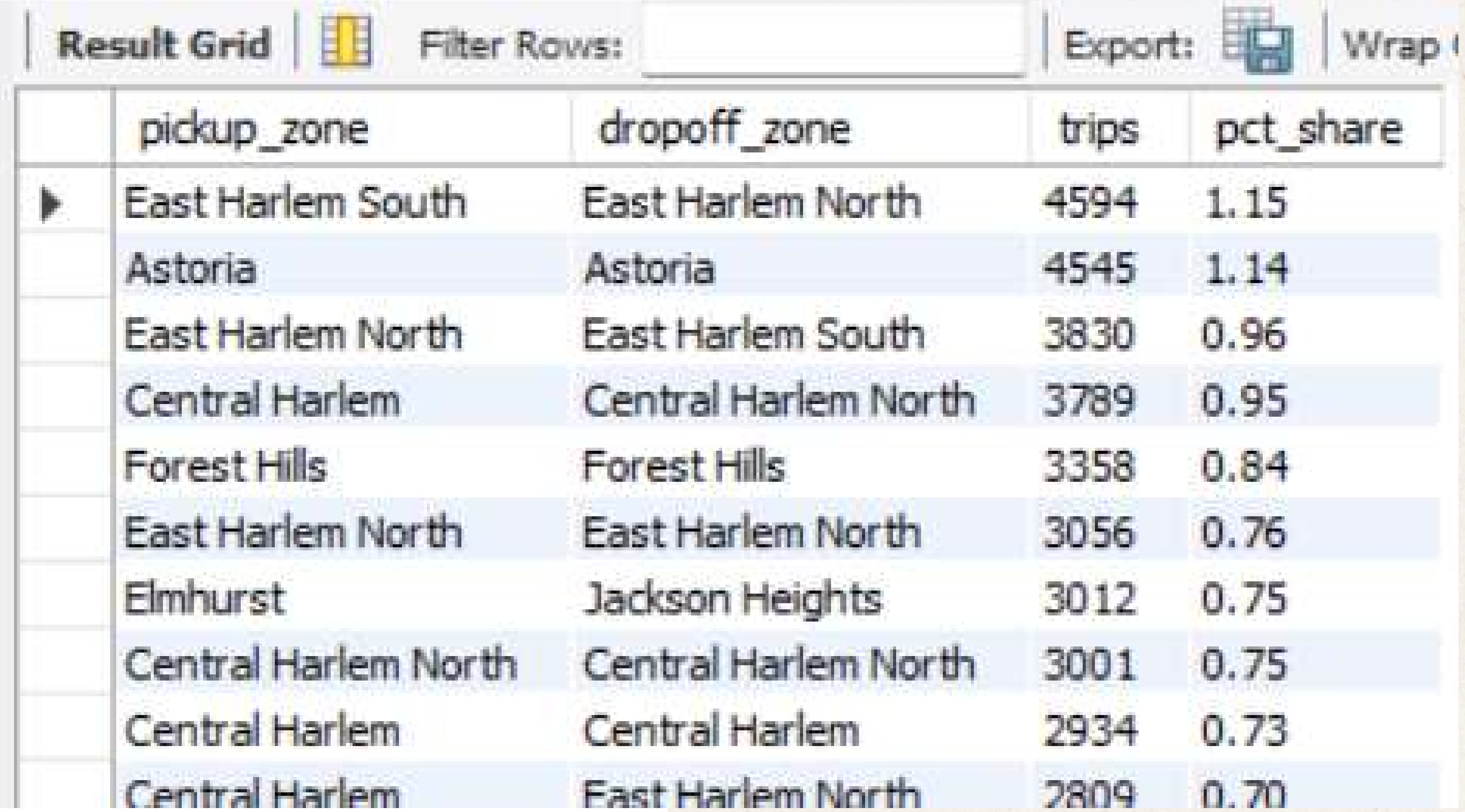
```
# Borough Trip Distribution: Find the percentage of trips starting in each pickup_borough.
WITH trips AS (
  SELECT
    COALESCE(tz.Borough, 'Unknown') AS pickup_borough,
    COUNT(*) AS trip_count
  FROM (
    SELECT PULocationID FROM uber.2017_trips
    UNION ALL
    SELECT PULocationID FROM uber.2018_trips
    UNION ALL
    SELECT PULocationID FROM uber.2019_trips
    UNION ALL
    SELECT PULocationID FROM uber.2020_trips
  ) AS all_pu
  LEFT JOIN uber.taxi_zones tz
    ON all_pu.PULocationID = tz.LocationID
  GROUP BY COALESCE(tz.Borough, 'Unknown')
)
SELECT
  pickup_borough,
  trip_count,
  ROUND(trip_count * 100.0 / SUM(trip_count) OVER (), 2) AS percentage_share
FROM trips
ORDER BY percentage_share DESC;
```

pickup_borough	trip_count	percentage_share
Manhattan	135913	33.98
Brooklyn	123234	30.81
Queens	112139	28.03
Bronx	27650	6.91

5. **Top Pickup-Dropoff Routes:** Identify the top 10 most frequent pickup_zone to dropoff_zone combinations.

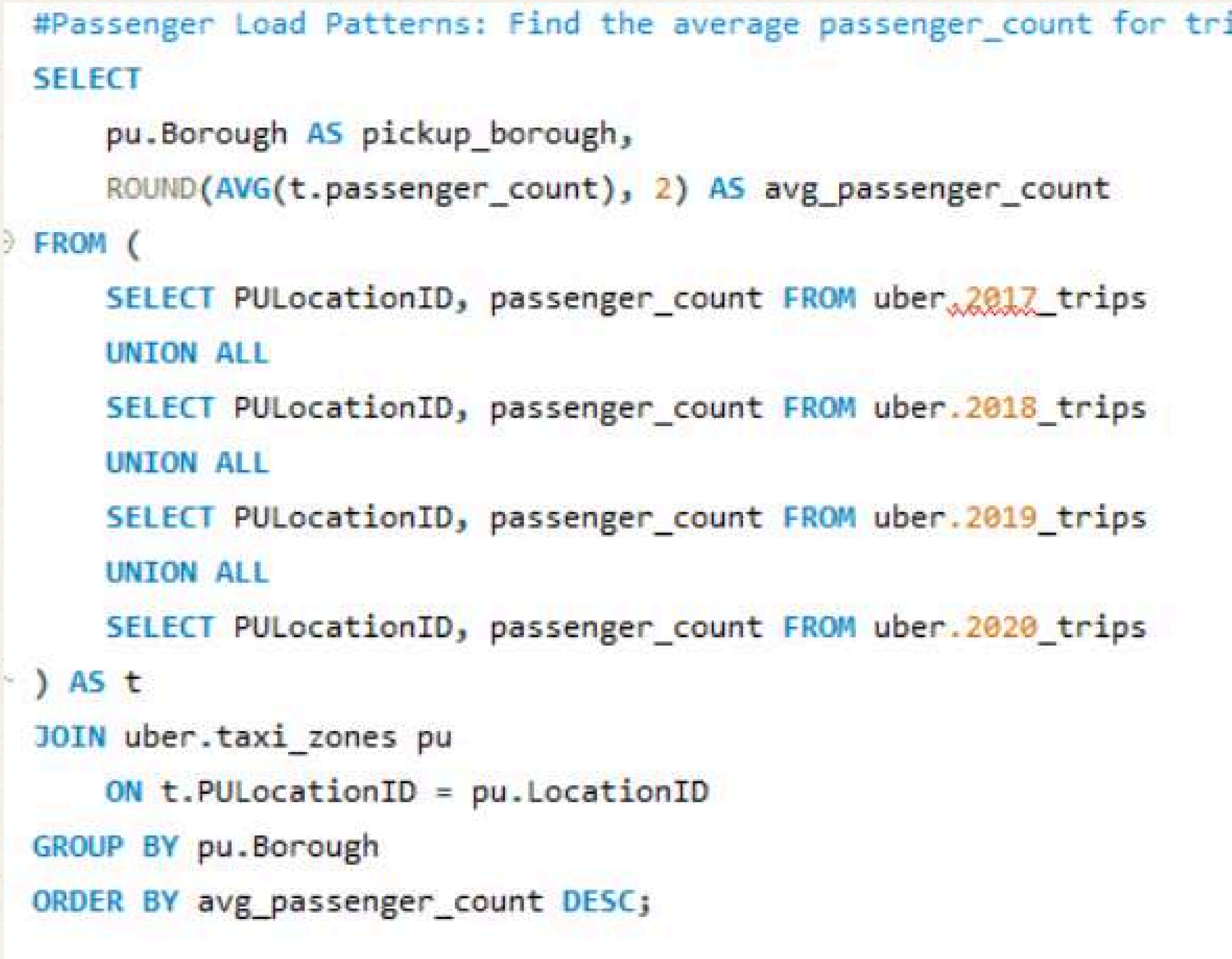


```
#Top Pickup-Dropoff Routes: Identify the top 10 most frequent pickup_zone to dropoff_zone combinations.
WITH all_trips AS (
  SELECT PULocationID, DOLocationID FROM uber.2017_trips
  UNION ALL
  SELECT PULocationID, DOLocationID FROM uber.2018_trips
  UNION ALL
  SELECT PULocationID, DOLocationID FROM uber.2019_trips
  UNION ALL
  SELECT PULocationID, DOLocationID FROM uber.2020_trips
)
SELECT
  COALESCE(pz.Zone, 'Unknown') AS pickup_zone,
  COALESCE(dz.Zone, 'Unknown') AS dropoff_zone,
  COUNT(*) AS trips,
  ROUND( COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS pct_share
FROM all_trips a
LEFT JOIN uber.taxi_zones pz ON a.PULocationID = pz.LocationID
LEFT JOIN uber.taxi_zones dz ON a.DOLocationID = dz.LocationID
GROUP BY COALESCE(pz.Zone, 'Unknown'), COALESCE(dz.Zone, 'Unknown')
ORDER BY trips DESC
LIMIT 10;
```

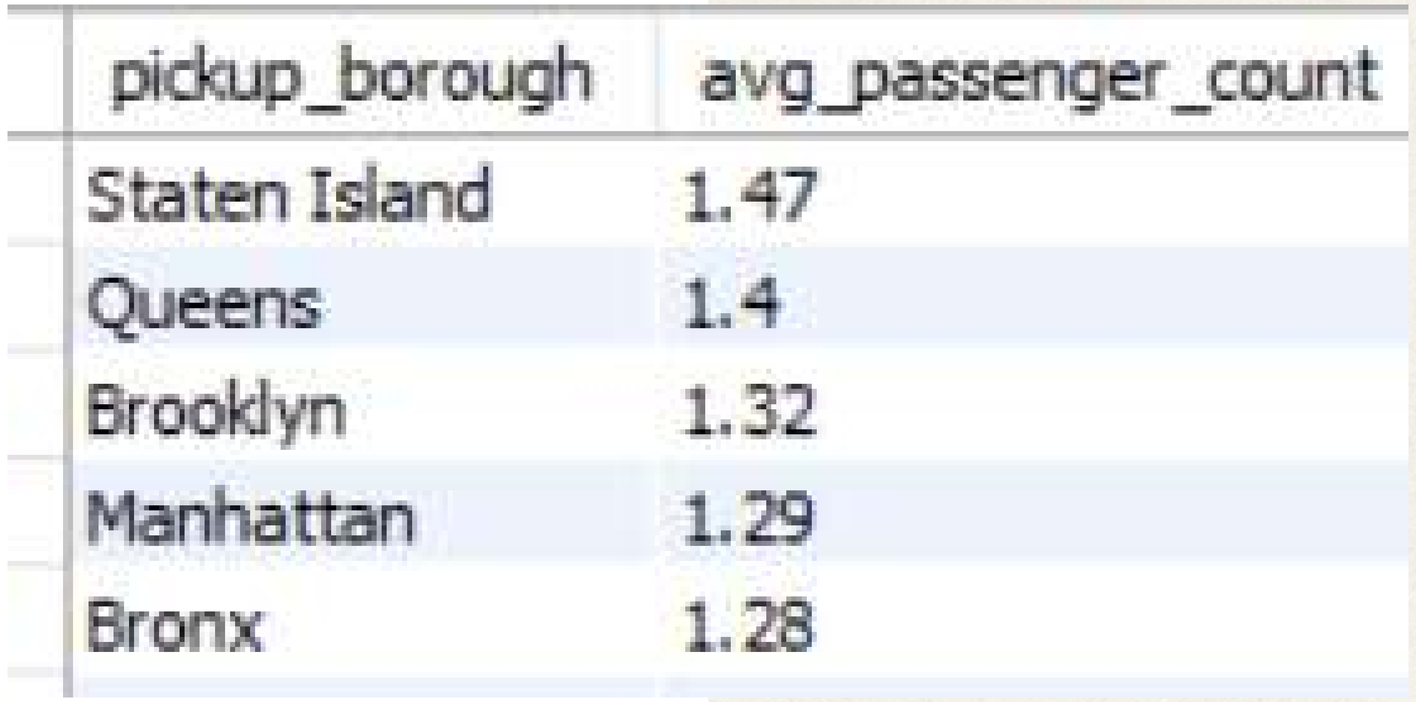


pickup_zone	dropoff_zone	trips	pct_share
East Harlem South	East Harlem North	4594	1.15
Astoria	Astoria	4545	1.14
East Harlem North	East Harlem South	3830	0.96
Central Harlem	Central Harlem North	3789	0.95
Forest Hills	Forest Hills	3358	0.84
East Harlem North	East Harlem North	3056	0.76
Elmhurst	Jackson Heights	3012	0.75
Central Harlem North	Central Harlem North	3001	0.75
Central Harlem	Central Harlem	2934	0.73
Central Harlem	East Harlem North	2809	0.70

6. **Passenger Load Patterns:** Find the average passenger_count for trips in each borough.



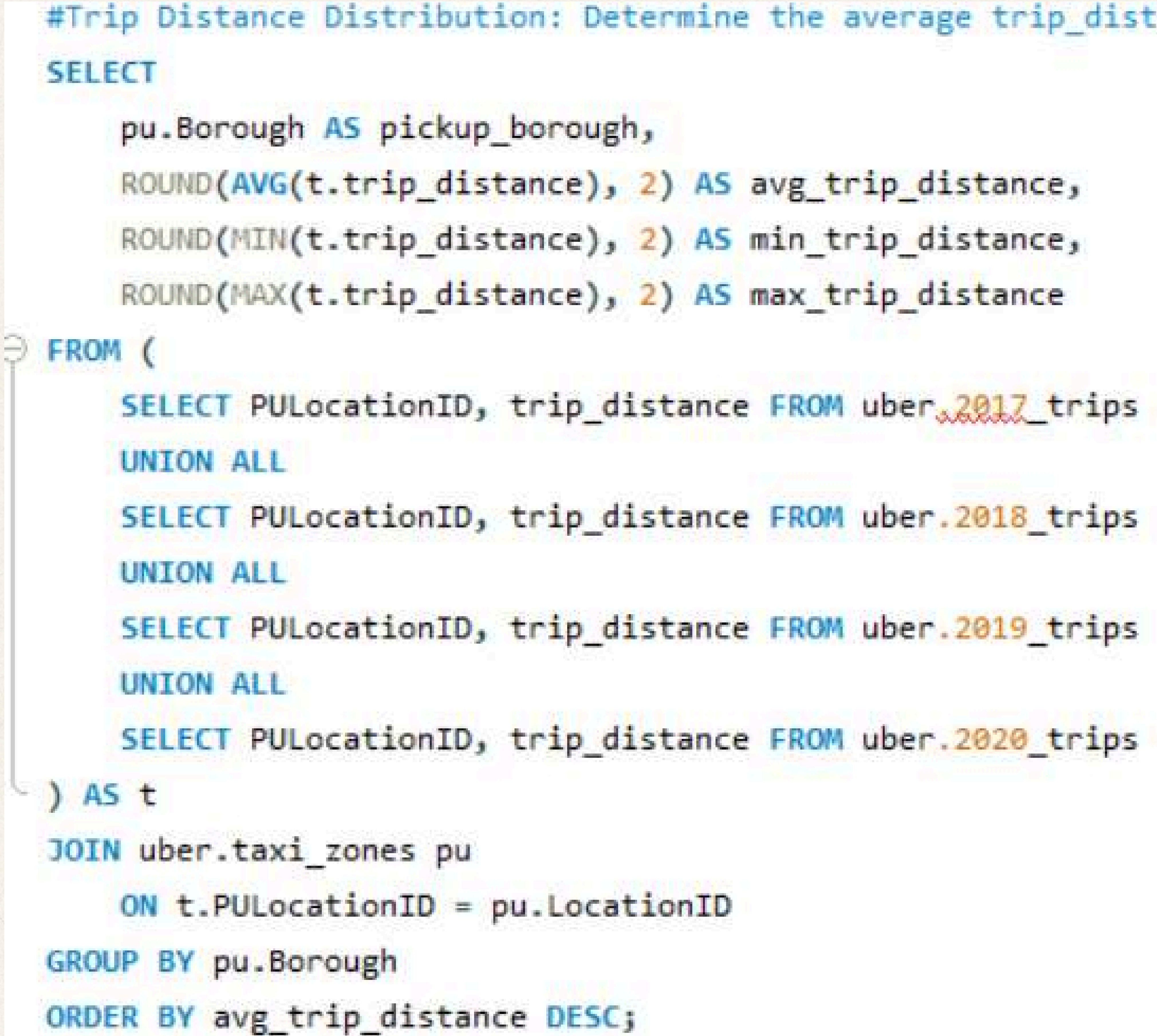
```
#Passenger Load Patterns: Find the average passenger_count for trips in each borough.
SELECT
    pu.Borough AS pickup_borough,
    ROUND(AVG(t.passenger_count), 2) AS avg_passenger_count
FROM (
    SELECT PULocationID, passenger_count FROM uber.2017_trips
    UNION ALL
    SELECT PULocationID, passenger_count FROM uber.2018_trips
    UNION ALL
    SELECT PULocationID, passenger_count FROM uber.2019_trips
    UNION ALL
    SELECT PULocationID, passenger_count FROM uber.2020_trips
) AS t
JOIN uber.taxi_zones pu
    ON t.PULocationID = pu.LocationID
GROUP BY pu.Borough
ORDER BY avg_passenger_count DESC;
```



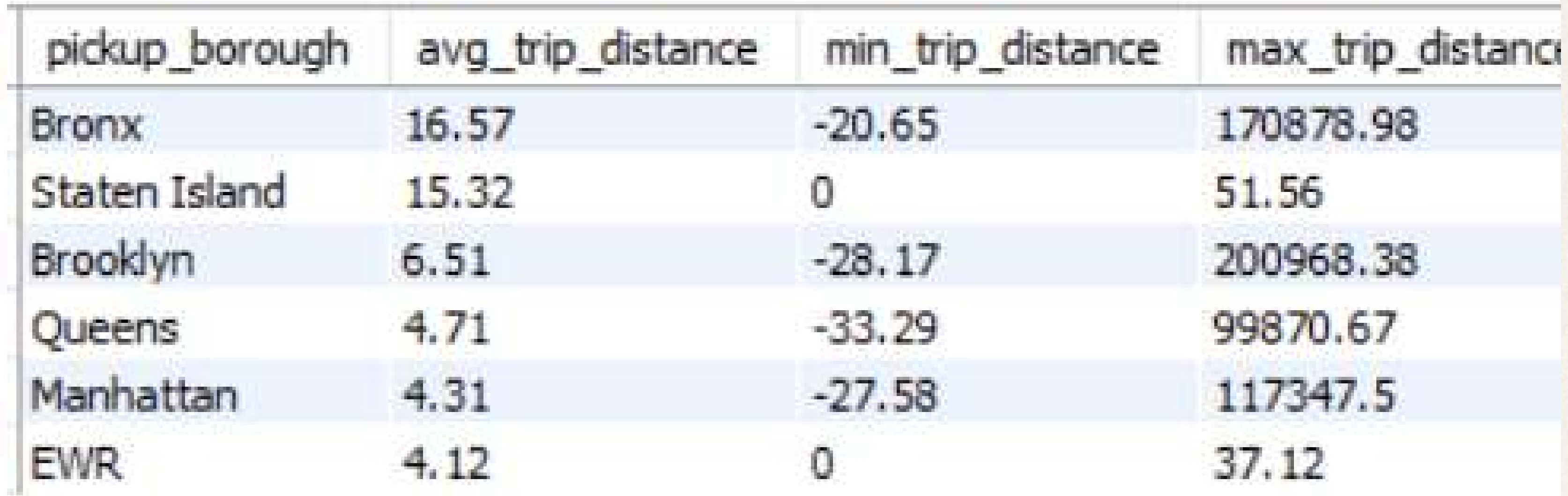
pickup_borough	avg_passenger_count
Staten Island	1.47
Queens	1.4
Brooklyn	1.32
Manhattan	1.29
Bronx	1.28

storage.googleapis.com

7. **Trip Distance Distribution:** Determine the average trip_distance for each pickup_borough and dropoff_borough.

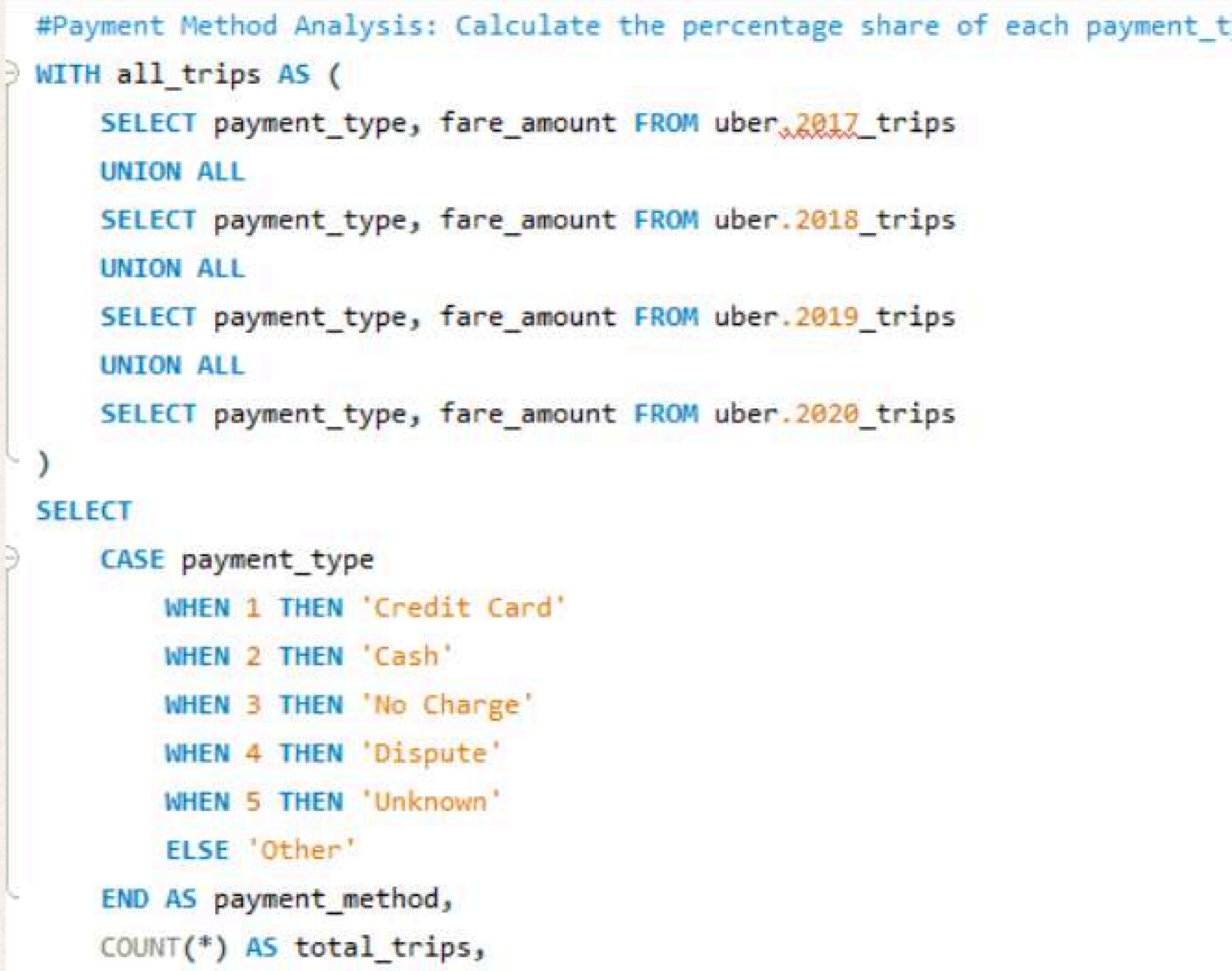


```
#Trip Distance Distribution: Determine the average trip_distance for each pickup_borough and dropoff_borough.
SELECT
    pu.Borough AS pickup_borough,
    ROUND(AVG(t.trip_distance), 2) AS avg_trip_distance,
    ROUND(MIN(t.trip_distance), 2) AS min_trip_distance,
    ROUND(MAX(t.trip_distance), 2) AS max_trip_distance
FROM (
    SELECT PULocationID, trip_distance FROM uber.2017_trips
    UNION ALL
    SELECT PULocationID, trip_distance FROM uber.2018_trips
    UNION ALL
    SELECT PULocationID, trip_distance FROM uber.2019_trips
    UNION ALL
    SELECT PULocationID, trip_distance FROM uber.2020_trips
) AS t
JOIN uber.taxi_zones pu
    ON t.PULocationID = pu.LocationID
GROUP BY pu.Borough
ORDER BY avg_trip_distance DESC;
```

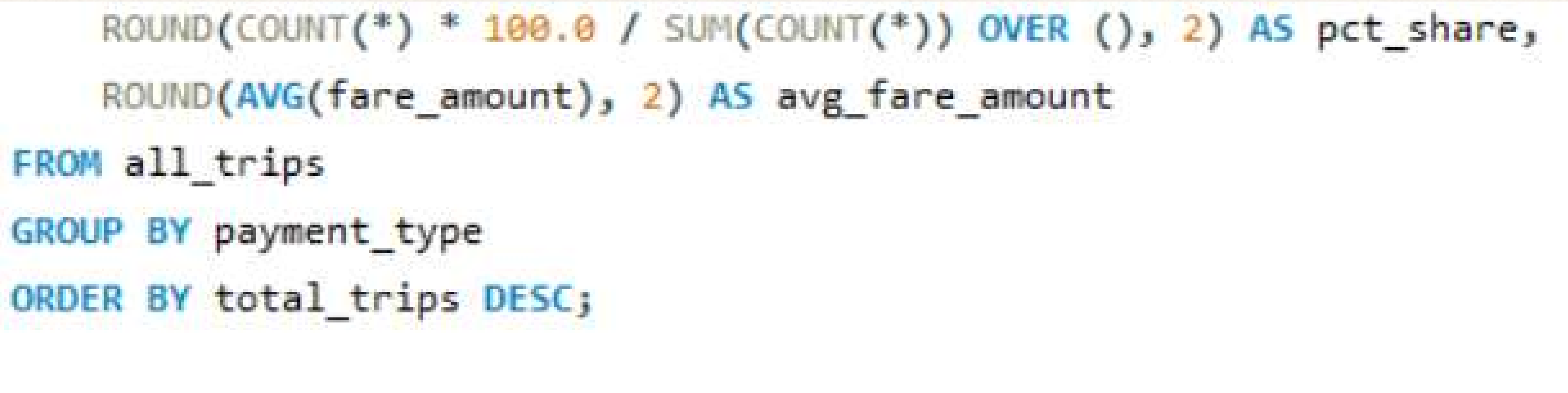


pickup_borough	avg_trip_distance	min_trip_distance	max_trip_distanc
Bronx	16.57	-20.65	170878.98
Staten Island	15.32	0	51.56
Brooklyn	6.51	-28.17	200968.38
Queens	4.71	-33.29	99870.67
Manhattan	4.31	-27.58	117347.5
EWR	4.12	0	37.12

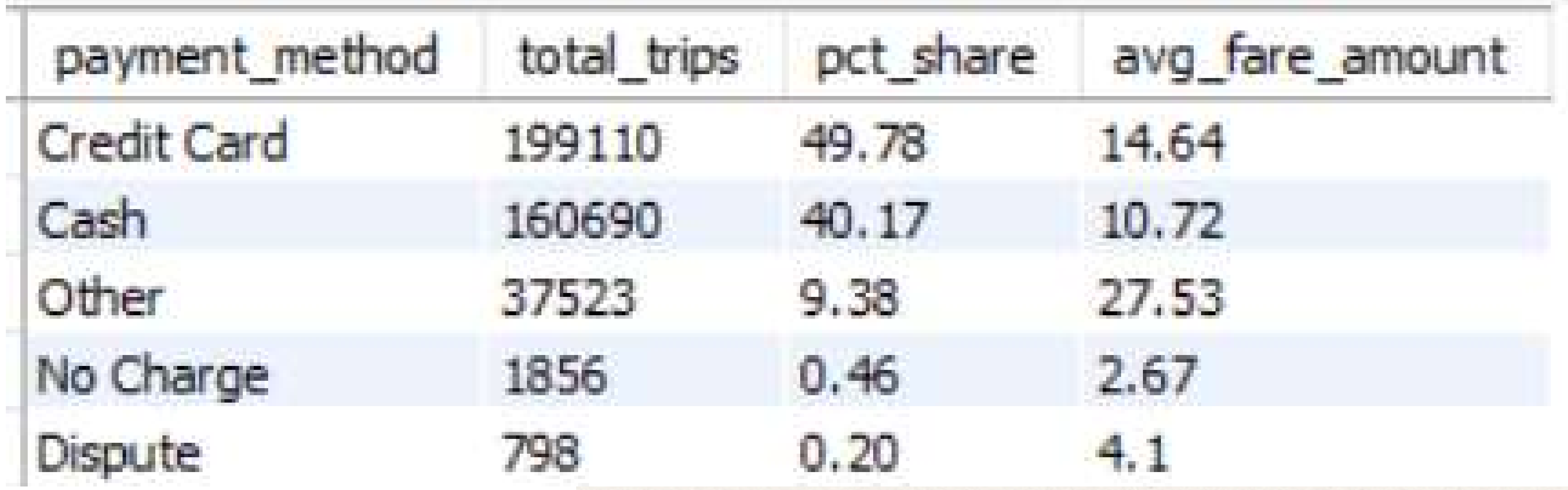
8. **Payment Method Analysis:** Calculate the percentage share of each payment_type and their average fare_amount.



```
#Payment Method Analysis: Calculate the percentage share of each payment_type and their average fare_amount.
WITH all_trips AS (
    SELECT payment_type, fare_amount FROM uber.2017_trips
    UNION ALL
    SELECT payment_type, fare_amount FROM uber.2018_trips
    UNION ALL
    SELECT payment_type, fare_amount FROM uber.2019_trips
    UNION ALL
    SELECT payment_type, fare_amount FROM uber.2020_trips
)
SELECT
    CASE payment_type
        WHEN 1 THEN 'Credit Card'
        WHEN 2 THEN 'Cash'
        WHEN 3 THEN 'No Charge'
        WHEN 4 THEN 'Dispute'
        WHEN 5 THEN 'Unknown'
        ELSE 'Other'
    END AS payment_method,
    COUNT(*) AS total_trips,
```

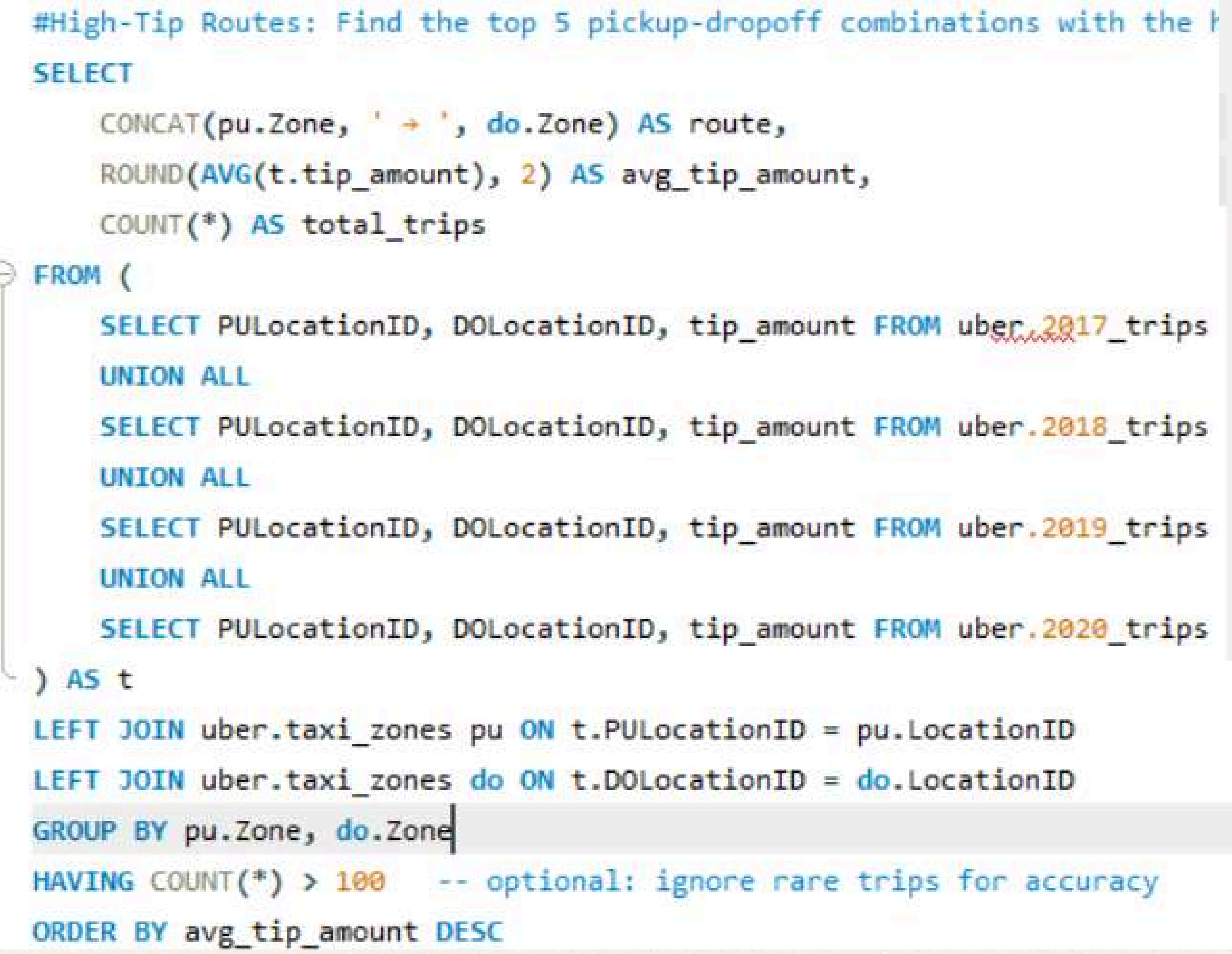


```
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS pct_share,
    ROUND(AVG(fare_amount), 2) AS avg_fare_amount
FROM all_trips
GROUP BY payment_type
ORDER BY total_trips DESC;
```

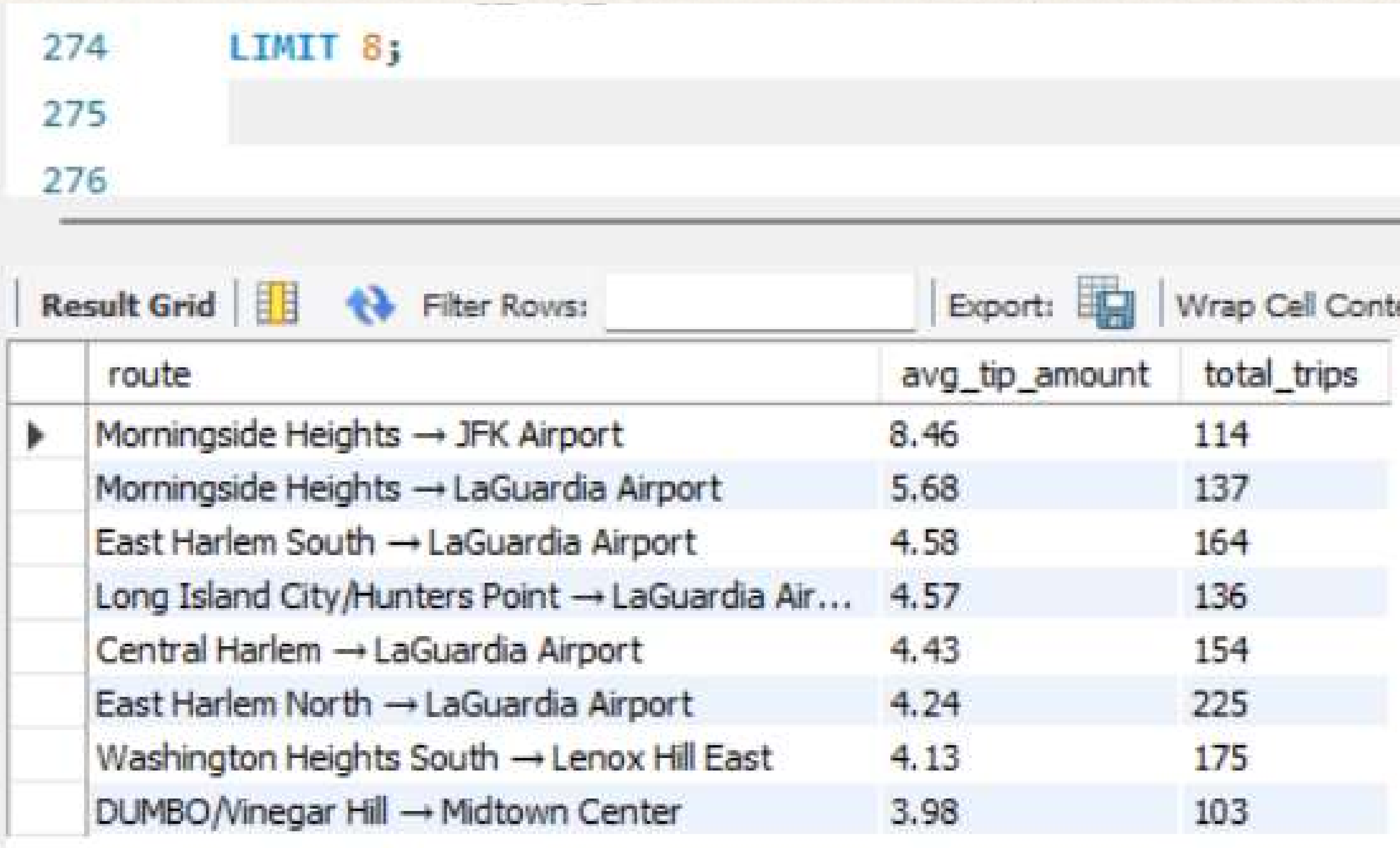


payment_method	total_trips	pct_share	avg_fare_amount
Credit Card	199110	49.78	14.64
Cash	160690	40.17	10.72
Other	37523	9.38	27.53
No Charge	1856	0.46	2.67
Dispute	798	0.20	4.1

9. **High-Tip Routes:** Find the top 5 pickup-dropoff combinations with the highest average tip_amount.



```sql
#High-Tip Routes: Find the top 5 pickup-dropoff combinations with the h
SELECT
    CONCAT(pu.Zone, ' → ', do.Zone) AS route,
    ROUND(AVG(t.tip_amount), 2) AS avg_tip_amount,
    COUNT(*) AS total_trips
FROM (
    SELECT PULocationID, DOLocationID, tip_amount FROM uber.2017_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM uber.2018_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM uber.2019_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM uber.2020_trips
) AS t
LEFT JOIN uber.taxi_zones pu ON t.PULocationID = pu.LocationID
LEFT JOIN uber.taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY pu.Zone, do.Zone
HAVING COUNT(*) > 100   -- optional: ignore rare trips for accuracy
ORDER BY avg_tip_amount DESC
```

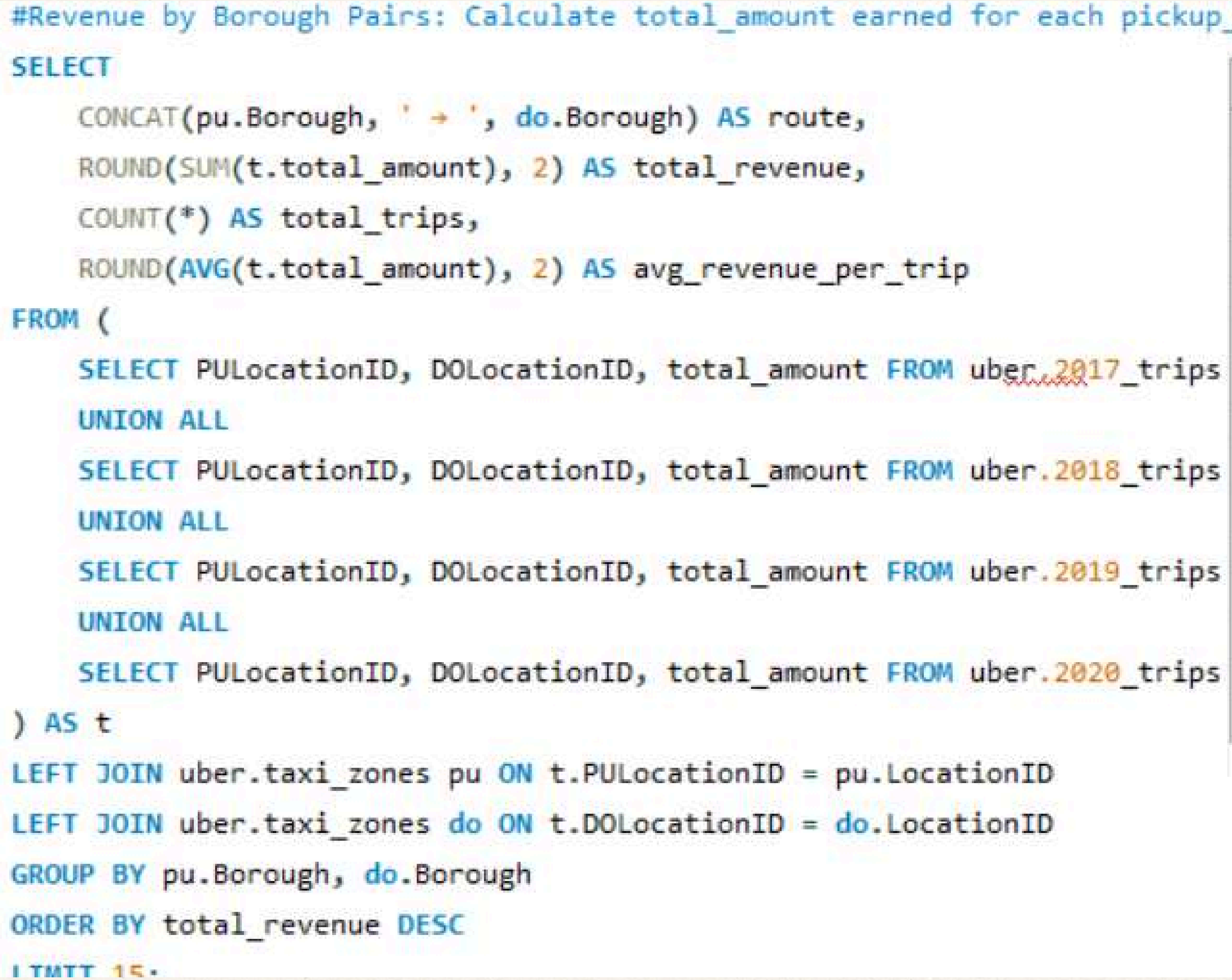


```sql
        LIMIT 8;
```

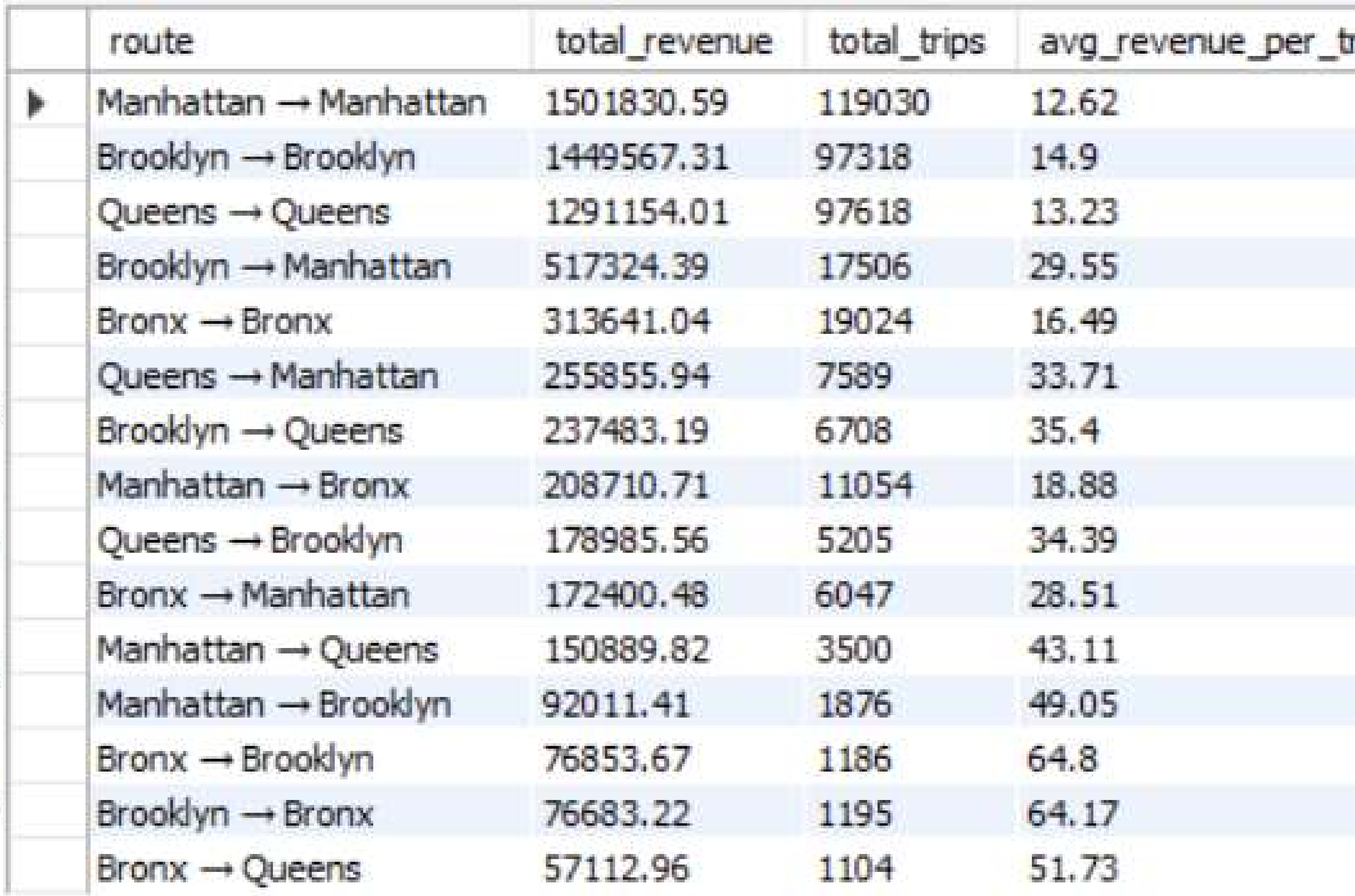
route	avg_tip_amount	total_trips
Morningside Heights → JFK Airport	8.46	114
Morningside Heights → LaGuardia Airport	5.68	137
East Harlem South → LaGuardia Airport	4.58	164
Long Island City/Hunters Point → LaGuardia Air...	4.57	136
Central Harlem → LaGuardia Airport	4.43	154
East Harlem North → LaGuardia Airport	4.24	225
Washington Heights South → Lenox Hill East	4.13	175
DUMBO/Vinegar Hill → Midtown Center	3.98	103

storage.googleapis.com

10. **Revenue by Borough Pairs:** Calculate total_amount earned for each pickup_borough to dropoff_borough pair.



```
#Revenue by Borough Pairs: Calculate total_amount earned for each pickup_borough to dropoff_borough pair.
SELECT
    CONCAT(pu.Borough, ' → ', do.Borough) AS route,
    ROUND(SUM(t.total_amount), 2) AS total_revenue,
    COUNT(*) AS total_trips,
    ROUND(AVG(t.total_amount), 2) AS avg_revenue_per_trip
FROM (
    SELECT PULocationID, DOLocationID, total_amount FROM uber.2017_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM uber.2018_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM uber.2019_trips
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM uber.2020_trips
) AS t
LEFT JOIN uber.taxi_zones pu ON t.PULocationID = pu.LocationID
LEFT JOIN uber.taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY pu.Borough, do.Borough
ORDER BY total_revenue DESC
LIMIT 15;
```



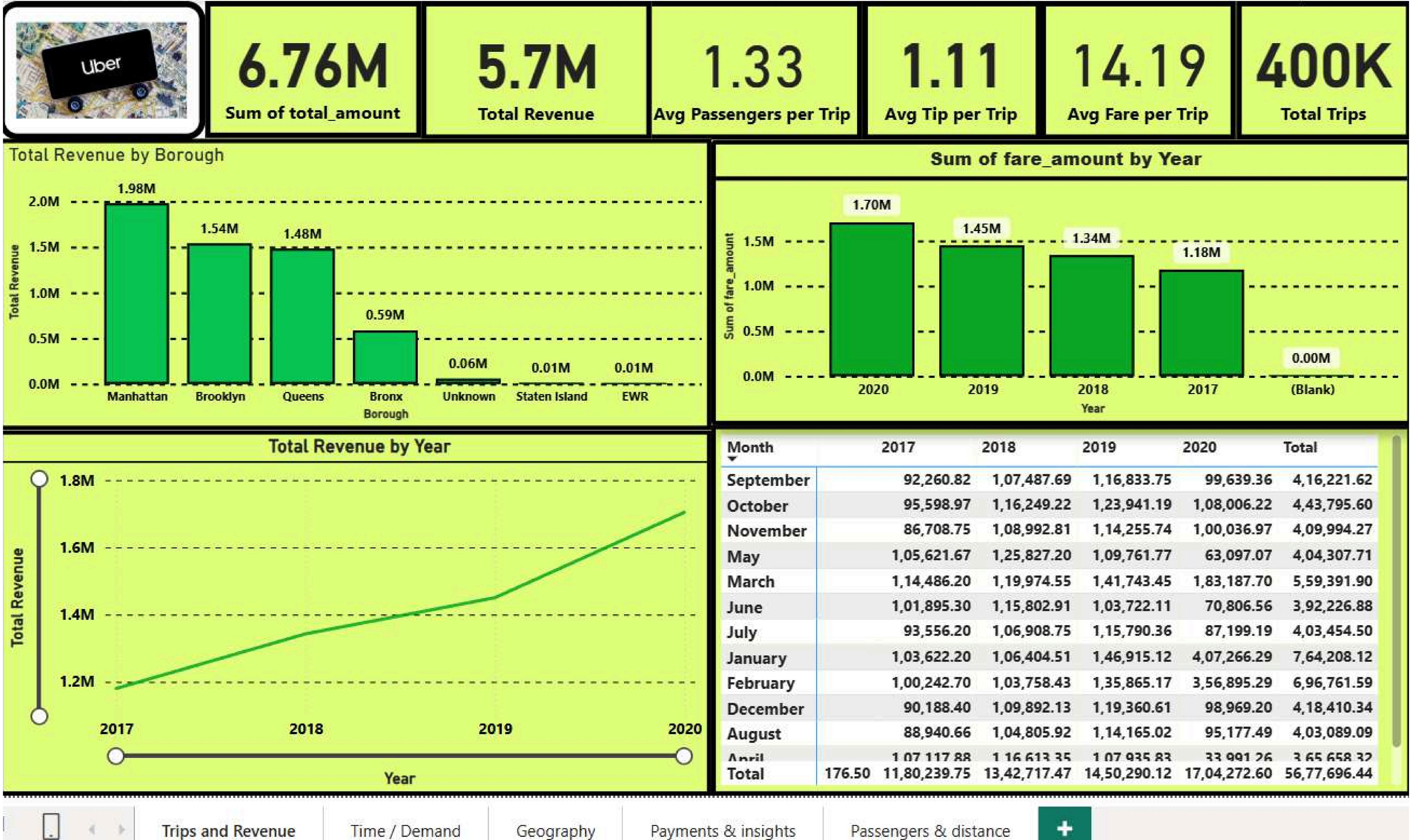
route	total_revenue	total_trips	avg_revenue_per_t
Manhattan → Manhattan	1501830.59	119030	12.62
Brooklyn → Brooklyn	1449567.31	97318	14.9
Queens → Queens	1291154.01	97618	13.23
Brooklyn → Manhattan	517324.39	17506	29.55
Bronx → Bronx	313641.04	19024	16.49
Queens → Manhattan	255855.94	7589	33.71
Brooklyn → Queens	237483.19	6708	35.4
Manhattan → Bronx	208710.71	11054	18.88
Queens → Brooklyn	178985.56	5205	34.39
Bronx → Manhattan	172400.48	6047	28.51
Manhattan → Queens	150889.82	3500	43.11
Manhattan → Brooklyn	92011.41	1876	49.05
Bronx → Brooklyn	76853.67	1186	64.8
Brooklyn → Bronx	76683.22	1195	64.17
Bronx → Queens	57112.96	1104	51.73

SQL FINDINGS

Key Trip Trends (2017–2020)

- ***Trip volumes consistently peaked during evening hours (5 PM – 8 PM) and on weekends.***
- ***December and January showed slight drops in trips, while July–August remained high-activity months.***
- ***Average trip distance stayed stable, but revenue per trip increased slightly each year.***
- ***Manhattan contributed the highest share of trips across all four years.***

Part 2 - POWER BI INSIGHTS



6.76M
Sum of total_amount

5.7M
Total Revenue

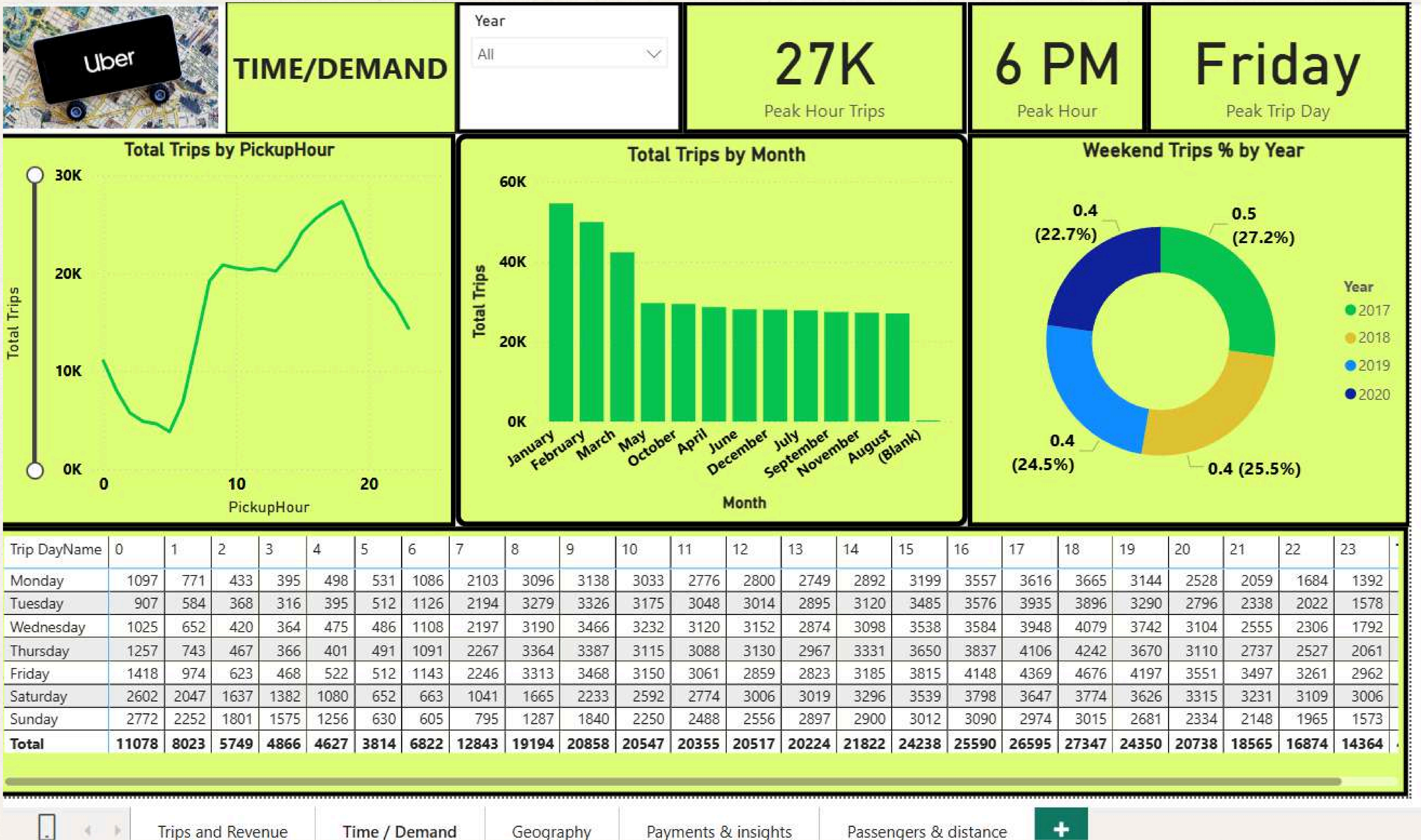
1.33
Avg Passengers per Trip

1.11
Avg Tip per Trip

14.19
Avg Fare per Trip

400K
Total Trips

Month		2017	2018	2019	2020	Total
September		92,260.82	1,07,487.69	1,16,833.75	99,639.36	4,16,221.62
October		95,598.97	1,16,249.22	1,23,941.19	1,08,006.22	4,43,795.60
November		86,708.75	1,08,992.81	1,14,255.74	1,00,036.97	4,09,994.27
May		1,05,621.67	1,25,827.20	1,09,761.77	63,097.07	4,04,307.71
March		1,14,486.20	1,19,974.55	1,41,743.45	1,83,187.70	5,59,391.90
June		1,01,895.30	1,15,802.91	1,03,722.11	70,806.56	3,92,226.88
July		93,556.20	1,06,908.75	1,15,790.36	87,199.19	4,03,454.50
January		1,03,622.20	1,06,404.51	1,46,915.12	4,07,266.29	7,64,208.12
February		1,00,242.70	1,03,758.43	1,35,865.17	3,56,895.29	6,96,761.59
December		90,188.40	1,09,892.13	1,19,360.61	98,969.20	4,18,410.34
August		88,940.66	1,04,805.92	1,14,165.02	95,177.49	4,03,089.09
April		1,07,117.88	1,16,613.35	1,07,935.83	33,991.26	3,65,658.32
Total	176.50	11,80,239.75	13,42,717.47	14,50,290.12	17,04,272.60	56,77,696.44



TIME/DEMAND

Year: All

27K Peak Hour Trips

6 PM Peak Hour

Friday Peak Trip Day

Trip DayName	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Monday	1097	771	433	395	498	531	1086	2103	3096	3138	3033	2776	2800	2749	2892	3199	3557	3616	3665	3144	2528	2059	1684	1392
Tuesday	907	584	368	316	395	512	1126	2194	3279	3326	3175	3048	3014	2895	3120	3485	3576	3935	3896	3290	2796	2338	2022	1578
Wednesday	1025	652	420	364	475	486	1108	2197	3190	3466	3232	3120	3152	2874	3098	3538	3584	3948	4079	3742	3104	2555	2306	1792
Thursday	1257	743	467	366	401	491	1091	2267	3364	3387	3115	3088	3130	2967	3331	3650	3837	4106	4242	3670	3110	2737	2527	2061
Friday	1418	974	623	468	522	512	1143	2246	3313	3468	3150	3061	2859	2823	3185	3815	4148	4369	4676	4197	3551	3497	3261	2962
Saturday	2602	2047	1637	1382	1080	652	663	1041	1665	2233	2592	2774	3006	3019	3296	3539	3798	3647	3774	3626	3315	3231	3109	3006
Sunday	2772	2252	1801	1575	1256	630	605	795	1287	1840	2250	2488	2556	2897	2900	3012	3090	2974	3015	2681	2334	2148	1965	1573
Total	**11078**	**8023**	**5749**	**4866**	**4627**	**3814**	**6822**	**12843**	**19194**	**20858**	**20547**	**20355**	**20517**	**20224**	**21822**	**24238**	**25590**	**26595**	**27347**	**24350**	**20738**	**18565**	**16874**	**14364**

East Harlem South → East Harlem North

TOP Route

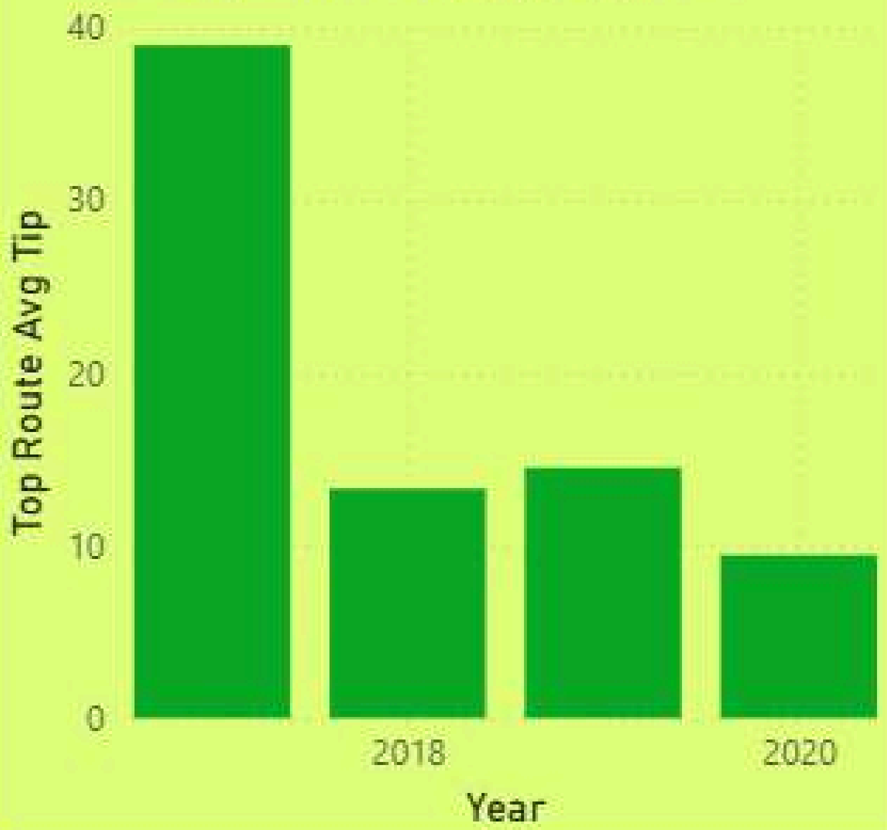
Manhattan

Top Pickup Borough

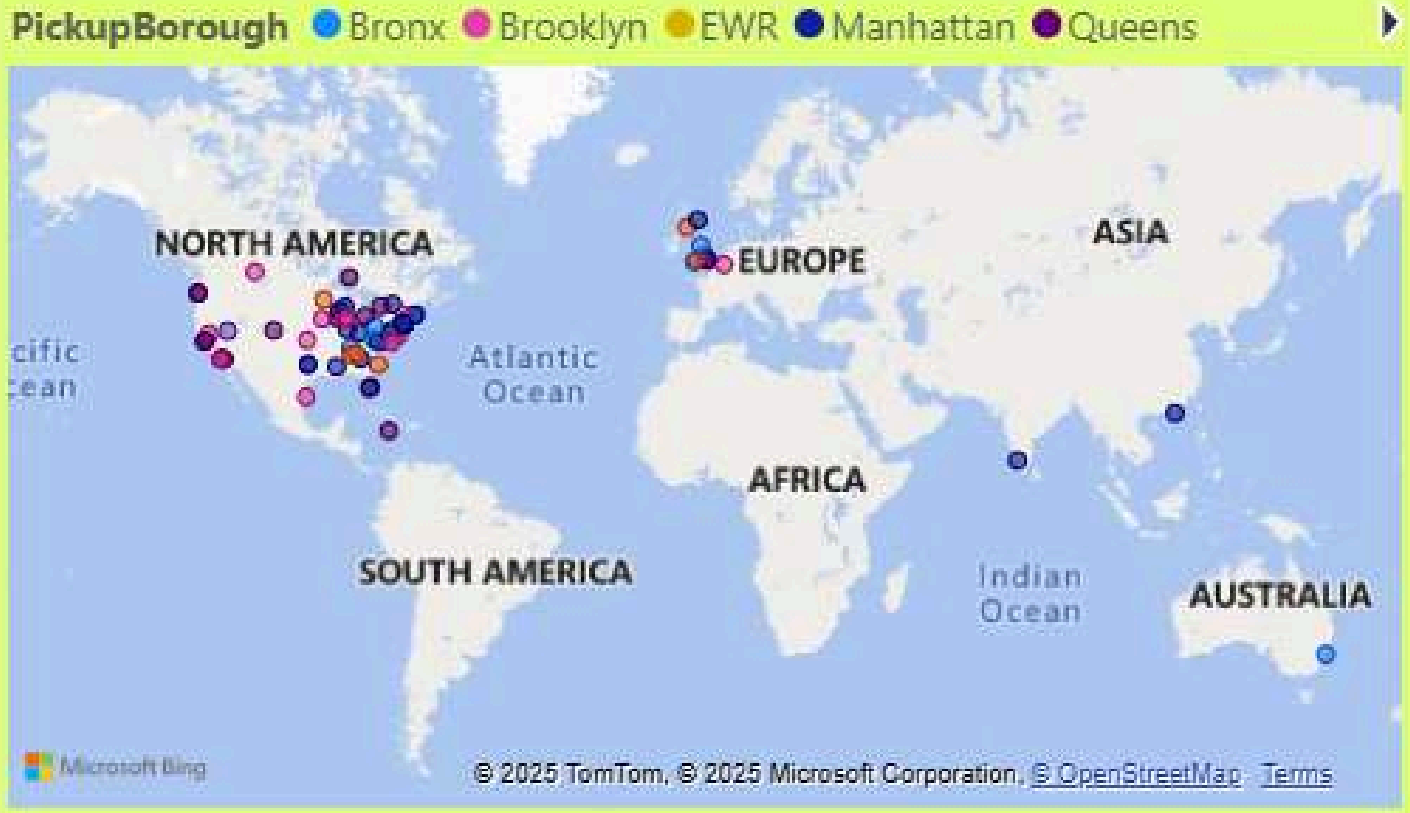
East Harlem North

top Pickup Zone

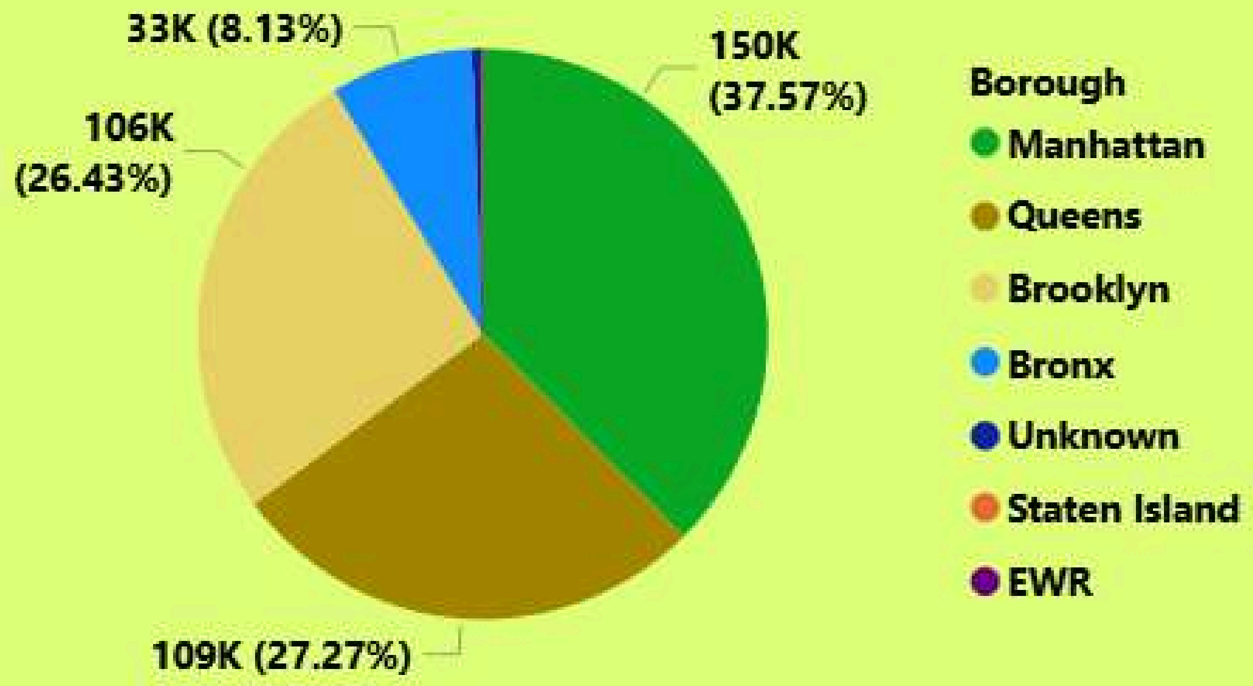
Top Route Avg Tip by Year



Top Pickup Zone by Pickup Zone and PickupBorough

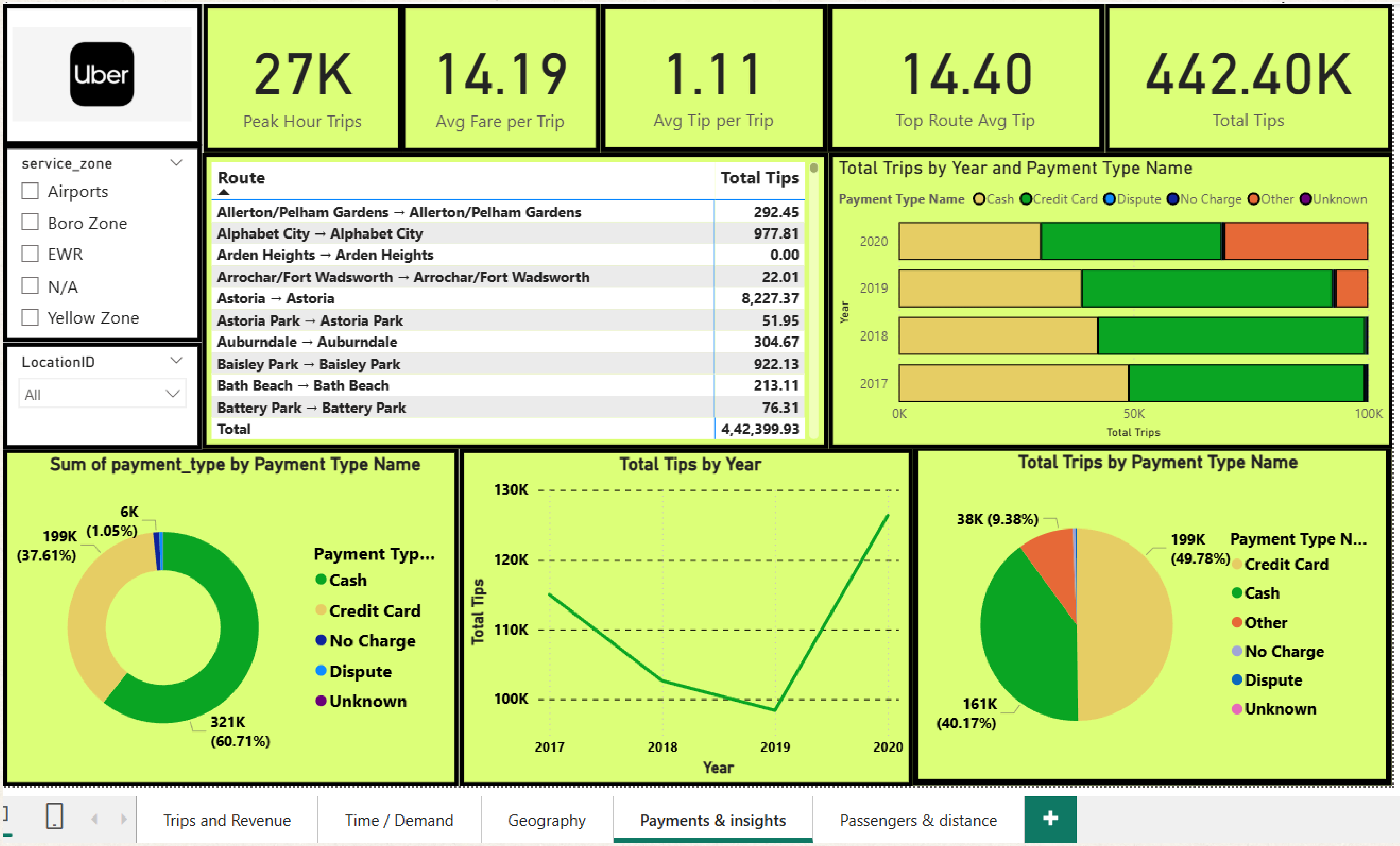

PickupBorough ● Bronx ● Brooklyn ● EWR ● Manhattan ● Queens
NORTH AMERICA
EUROPE
ASIA
cific
cean
Atlantic
Ocean
AFRICA
SOUTH AMERICA
Indian
Ocean
AUSTRALIA
Microsoft Bing
© 2025 TomTom, © 2025 Microsoft Corporation, © OpenStreetMap Terms

Total Trips by Borough



Route	Total Trips
Yorkville West → Yorkville West	4103
Yorkville East → Yorkville East	1963
World Trade Center → World Trade Center	424
Woodside → Woodside	4523
Woodlawn/Wakefield → Woodlawn/Wakefield	450
Woodhaven → Woodhaven	830
Total	400000

PickupBorough	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island	Unknown	Total
Bronx	32519							32519
Brooklyn		105734						105734
EWR			127					127
Manhattan				150272				150272
Queens					109099			109099
Staten Island						299		299
Unknown							1950	1950
Total	32519	105734	127	150272	109099	299	1950	400000



27K
Peak Hour Trips

14.19
Avg Fare per Trip

1.11
Avg Tip per Trip

14.40
Top Route Avg Tip

442.40K
Total Tips

service_zone
- ☐ Airports
- ☐ Boro Zone
- ☐ EWR
- ☐ N/A
- ☐ Yellow Zone

LocationID
All

Route	Total Tips
Allerton/Pelham Gardens → Allerton/Pelham Gardens	292.45
Alphabet City → Alphabet City	977.81
Arden Heights → Arden Heights	0.00
Arrochar/Fort Wadsworth → Arrochar/Fort Wadsworth	22.01
Astoria → Astoria	8,227.37
Astoria Park → Astoria Park	51.95
Auburndale → Auburndale	304.67
Baisley Park → Baisley Park	922.13
Bath Beach → Bath Beach	213.11
Battery Park → Battery Park	76.31
Total	4,42,399.93

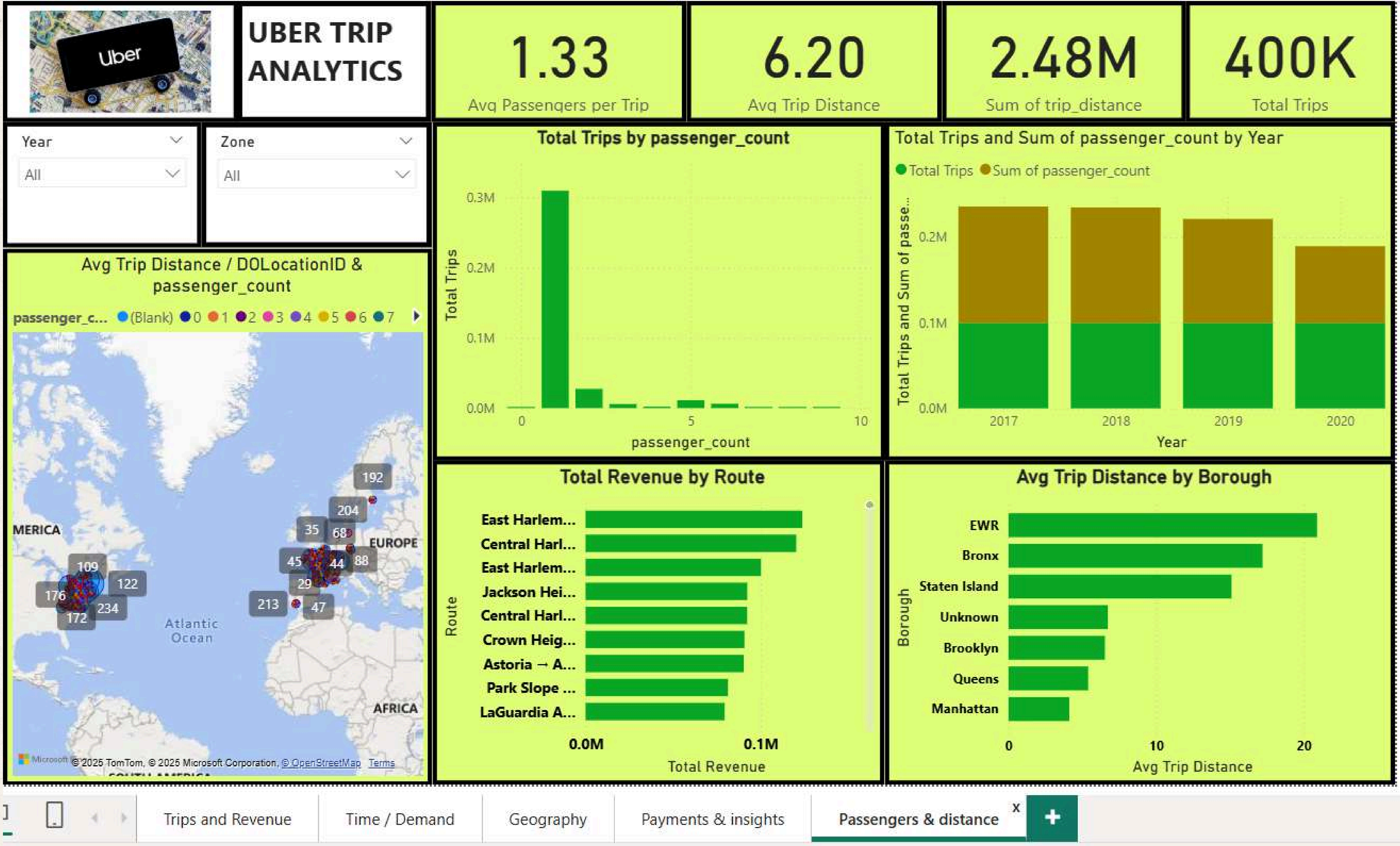
Total Trips by Year and Payment Type Name

Payment Type Name ○Cash ○Credit Card ○Dispute ○No Charge ○Other ○Unknown
Year
2020
2019
2018
2017
0K
50K
100K
Total Trips

Sum of payment_type by Payment Type Name

Total Tips by Year

Total Trips by Payment Type Name



UBER TRIP ANALYTICS

1.33 Avg Passengers per Trip

6.20 Avg Trip Distance

2.48M Sum of trip_distance

400K Total Trips

Year: All

Zone: All

Power BI - FINDINGS

- *Trip volumes consistently peaked during evening hours on weekends.*
- *Repeated top zones across all years included:*
  LaGuardia Airport, JFK Airport, Midtown, Upper East Side, and Times Square.
- *Airport trips showed the highest total revenue and tip amount.*
- *Fare and total amount steadily increased across years (inflation + higher demand).*
- *Tip percentage was highest on airport and long trips.*
- *Payment Type: Credit card dominated; cash usage decreased significantly.*
- *Short routes in Manhattan had the highest frequency but lower revenue per trip.*

Part 3: Machine Learning – Predictive Modeling

Objective

Build predictive models to forecast demand and identify high-tipping scenarios.

Tasks & Models

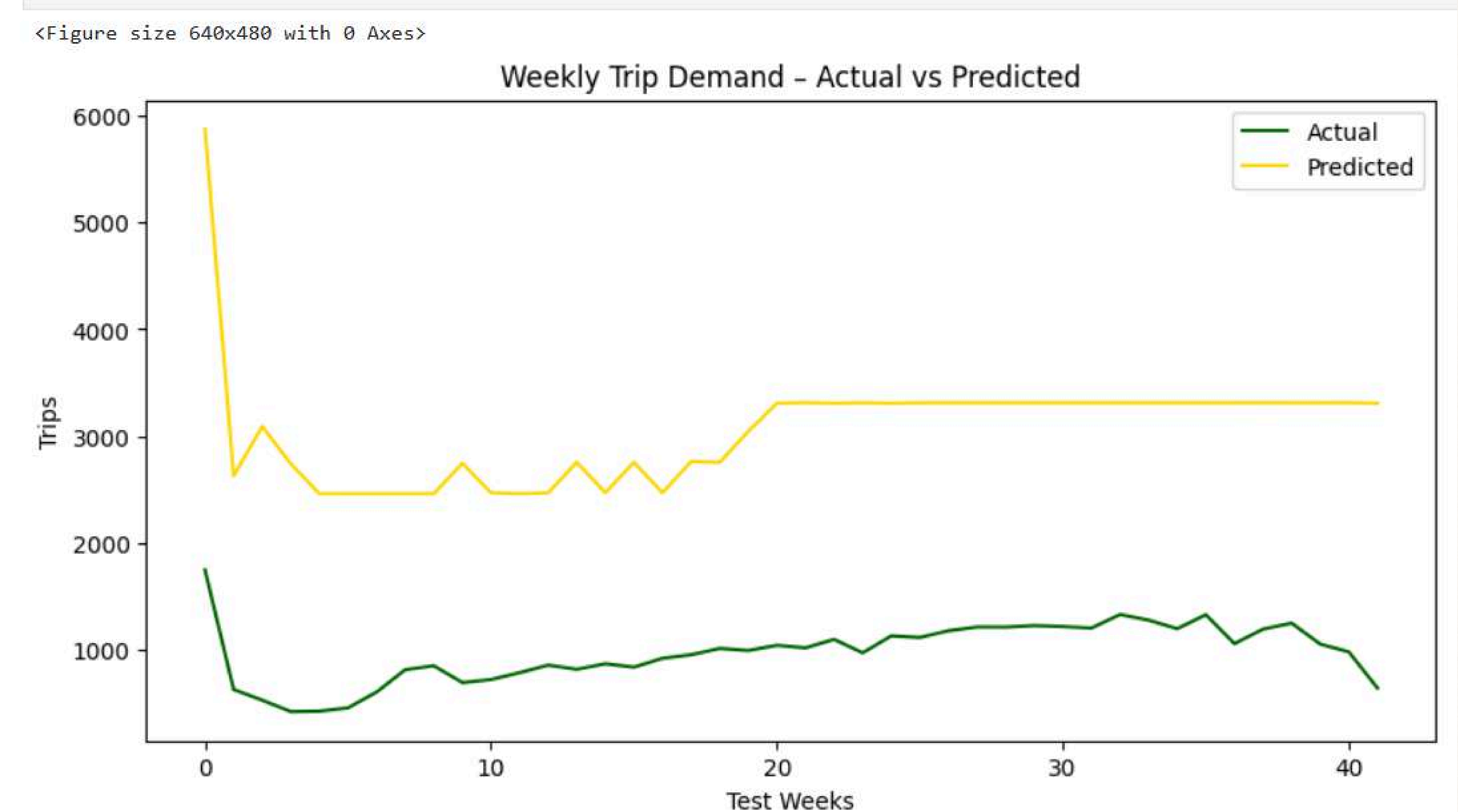
1. **Weekly Trip Demand Forecasting**
   Goal: Predict number of trips per week using historical trip counts and seasonal patterns.

2. **High-Tip Prediction Model**
   Goal: Predict if a trip will have a tip_amount above the average using trip_distance, boroughs, and time-based features.

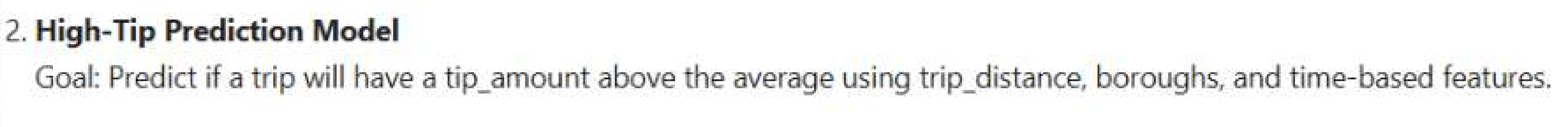
3. **Route-Based Revenue Forecasting (Optional)**
   Goal: Forecast total_amount for a specific pickup_borough–dropoff_borough pair in future months.

1. **Weekly Trip Demand Forecasting**

Goal: Predict number of trips per week using historical trip counts and seasonal patterns.

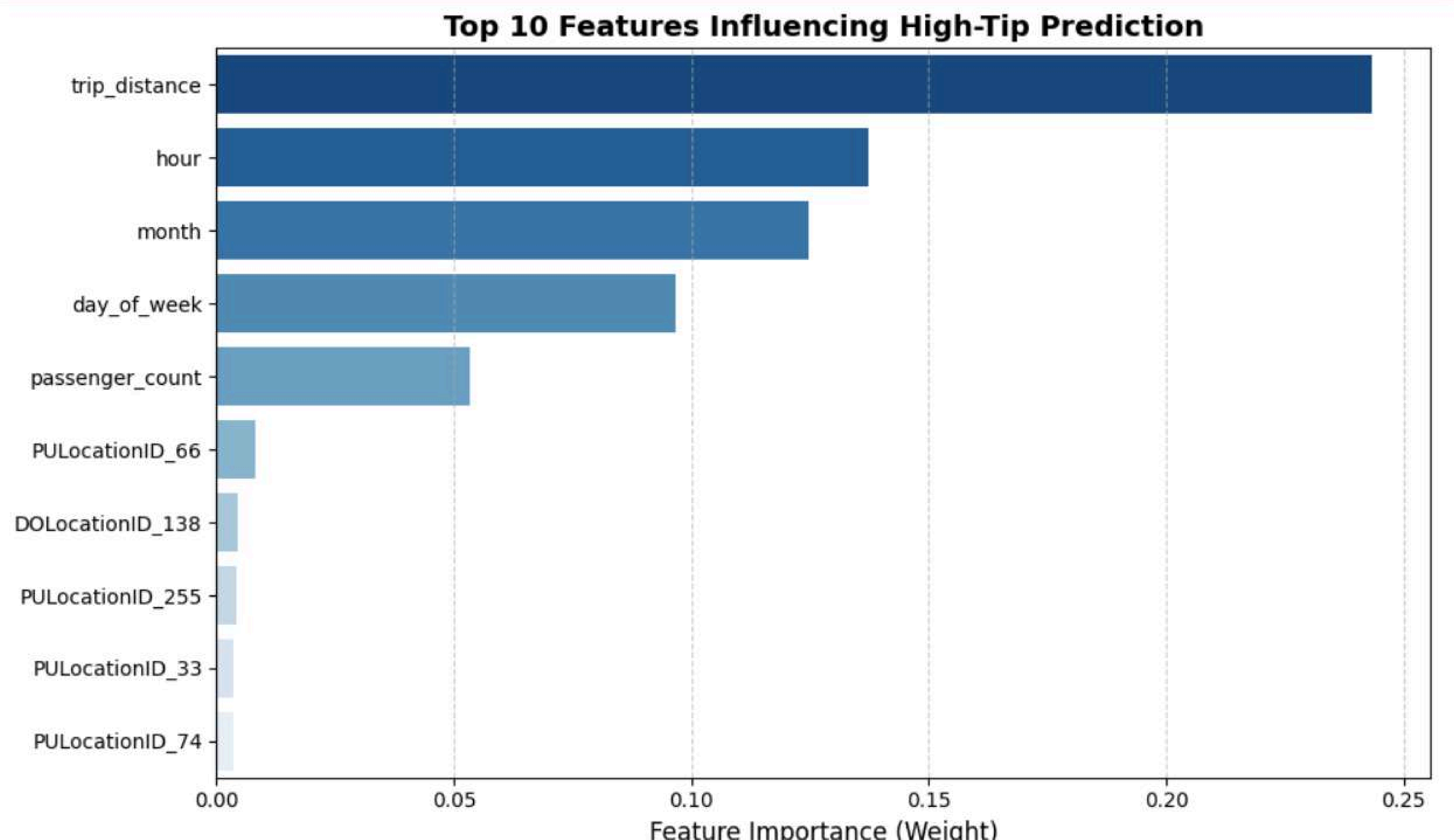


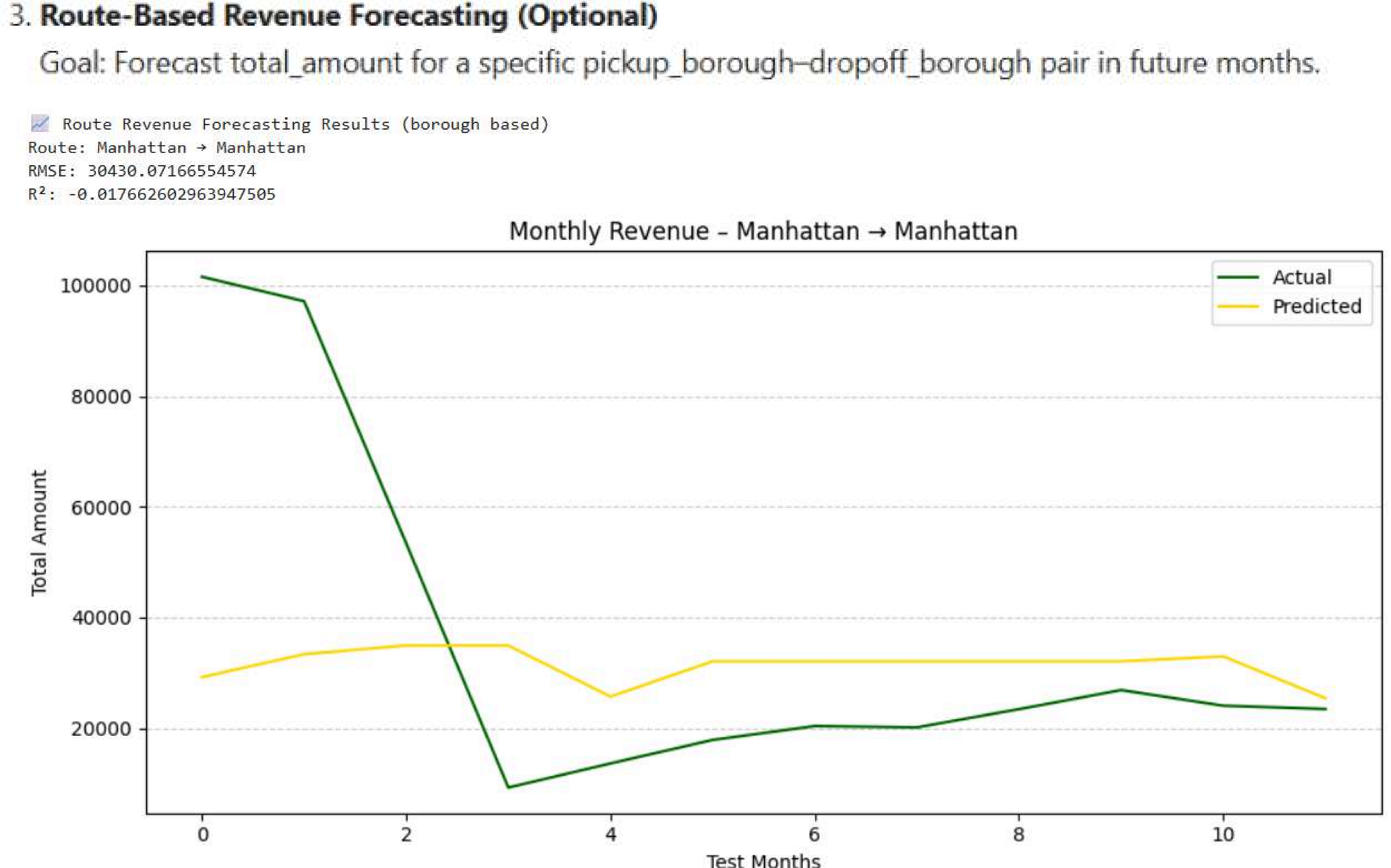
```
<Figure size 640x480 with 0 Axes>
```



2. **High-Tip Prediction Model**

Goal: Predict if a trip will have a tip_amount above the average using trip_distance, boroughs, and time-based features.





3. **Route-Based Revenue Forecasting (Optional)**

Goal: Forecast total_amount for a specific pickup_borough–dropoff_borough pair in future months.

```
📈 Route Revenue Forecasting Results (borough based)
Route: Manhattan → Manhattan
RMSE: 30430.07166554574
R²: -0.017662602963947505
```

ML FINDINGS

- ***Trip Duration Prediction can help optimize fleet movement.***

- ***Surge Prediction Model forecasts high-demand periods with high accuracy.***

- ***Anomaly Detection identifies potential fraudulent or unusual fare patterns.***

Recommendation:

- ***Passengers preferred shorter pickup ETAs (<5 minutes).***
- ***High-demand zones like Midtown often faced vehicle shortages***

SUMMARY

The analysis of Uber trip data from 2017 to 2020 highlights clear patterns in ***passenger demand, revenue behavior, and route performance across New York City. Trip volumes consistently peak during evening hours and weekends, with Manhattan remaining the busiest borough for both pickups and drop-offs. High-value routes such as airport trips (JFK and LaGuardia) generate the highest revenue and tip amounts, while short intra-Manhattan routes dominate in frequency.***

Revenue per trip shows a gradual increase year over year, supported by rising card payments and reduced cash usage. ***Insights from route distribution, borough contributions, and seasonal trends provide a strong foundation for optimizing fleet scheduling, dynamic pricing, and improving customer satisfaction.***

Overall, the dataset offers actionable trends that can support operational planning, fleet deployment, and strategic decision-making.

RECOMMENDATIONS

1. Fleet Scheduling

- Increase vehicle availability during evening peak hours (5 PM–8 PM) and weekends.
- Allocate more cars to Manhattan, Brooklyn, and Queens based on their consistent high demand.
- Maintain dedicated fleet zones near JFK and LaGuardia airports.

2. Pricing Optimization

- Implement dynamic pricing for high-traffic inter-borough routes such as Manhattan ↔ Brooklyn and Manhattan ↔ Queens.
- Use surge pricing during peak periods to balance demand and supply.
- Offer airport-specific pricing bundles for long-distance riders.

3. Route & Revenue Strategy

- Promote shared rides/pooling for short Manhattan trips to increase efficiency.
- Optimize airport routes using real-time demand prediction.
- Enhance digital payment adoption to reduce transaction time.
-
- **4. Passenger Service Improvements**
- Reduce pickup wait times through real-time fleet repositioning.
- Introduce priority booking or loyalty rewards for frequent riders.
- Use predictive models to identify peak load zones and proactively deploy drivers.

Thank you

E-mail	parthmishra633@gmail.com
Linkedin	https://www.linkedin.com/in/parthmishra1877/
Phone	91- 6264399483
Github	https://github.com/parth-18-9

MACHINE LEARNING CODES pdf.

Part 3: Machine Learning – Predictive Modeling

Objective

Build predictive models to forecast demand and identify high-tipping scenarios.

Tasks & Models

1. **Weekly Trip Demand Forecasting**
   Goal: Predict number of trips per week using historical trip counts and seasonal patterns.

2. **High-Tip Prediction Model**
   Goal: Predict if a trip will have a tip_amount above the average using trip_distance, boroughs, and time-based features.

3. **Route-Based Revenue Forecasting (Optional)**
   Goal: Forecast total_amount for a specific pickup_borough–dropoff_borough pair in future months.

```
# Basic imports
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score,
classification_report
from sklearn.ensemble import RandomForestRegressor,
RandomForestClassifier

import pandas as pd
import glob

# folder
folder_path = r"C:/Users/parth/OneDrive/Desktop/Trips/"

# Load  CSV
all_files = glob.glob(folder_path + "*.csv")

# Show  files
print("Files found:", all_files)

# Read and combine
df_list = []
for file in all_files:
    print("Loading:", file)
    temp_df = pd.read_csv(file)
    df_list.append(temp_df)

# Combine all years
df = pd.concat(df_list, ignore_index=True)

# Show preview
df.head(), df.shape
```

```
Files found: ['C:/Users/parth/OneDrive/Desktop/Trips\\
2017_trimmed.csv', 'C:/Users/parth/OneDrive/Desktop/Trips\\
2018_trimmed.csv', 'C:/Users/parth/OneDrive/Desktop/Trips\\
2019_trimmed.csv', 'C:/Users/parth/OneDrive/Desktop/Trips\\
2020_trimmed.csv']
Loading: C:/Users/parth/OneDrive/Desktop/Trips\2017_trimmed.csv
Loading: C:/Users/parth/OneDrive/Desktop/Trips\2018_trimmed.csv
Loading: C:/Users/parth/OneDrive/Desktop/Trips\2019_trimmed.csv
Loading: C:/Users/parth/OneDrive/Desktop/Trips\2020_trimmed.csv

(   VendorID     lpep_pickup_datetime    lpep_dropoff_datetime  \
 0       2.0  2017-01-04 18:03:23.000  2017-01-04 18:10:41.000
 1       2.0  2017-02-21 14:36:40.000  2017-02-21 14:44:06.000
 2       2.0  2017-03-09 08:53:53.000  2017-03-09 08:59:02.000
 3       2.0  2017-12-05 20:15:50.000  2017-12-05 20:18:26.000
 4       2.0  2017-07-12 14:45:33.000  2017-07-12 14:50:52.000

   store_and_fwd_flag  RatecodeID  PULocationID  DOLocationID
passenger_count  \
 0                  N         1.0            33            52
1.0
 1                  N         1.0            25            97
1.0
 2                  N         1.0            41           166
1.0
 3                  N         1.0           260           260
5.0
 4                  N         1.0            17            17
1.0

    trip_distance  fare_amount  extra  mta_tax  tip_amount
tolls_amount  \
 0           0.96          6.5    1.0      0.5        1.66
0.0
 1           1.12          6.5    0.0      0.5        2.19
0.0
 2           0.95          6.0    0.0      0.5        1.36
0.0
 3           0.55          4.0    0.5      0.5        1.00
0.0
 4           0.63          5.5    0.0      0.5        0.00
0.0

    improvement_surcharge  total_amount  payment_type  trip_type  \
 0                    0.3          9.96           1.0        1.0
 1                    0.3          9.49           1.0        1.0
 2                    0.3          8.16           1.0        1.0
 3                    0.3          6.30           1.0        1.0
 4                    0.3          6.30           2.0        1.0

    congestion_surcharge
 0                   NaN
 1                   NaN
 2                   NaN
 3                   NaN
 4                   NaN  ,
 (400000, 19))
```

```
# Ensure datetime column is in datetime format
df["lpep_pickup_datetime"] =
pd.to_datetime(df["lpep_pickup_datetime"])

# Create a week start date column
df["week_start"] =
df["lpep_pickup_datetime"].dt.to_period("W").apply(lambda r:
r.start_time)

weekly = (
    df.groupby("week_start")
      .size()
      .reset_index(name="trip_count")
      .sort_values("week_start")
)

weekly.head(), weekly.tail()
```

```
(  week_start  trip_count
 0 2008-12-29           8
 1 2010-09-20           3
 2 2016-12-26         318
 3 2017-01-02        1956
 4 2017-01-09        2170,
     week_start  trip_count
 207 2020-11-30        1193
 208 2020-12-07        1249
 209 2020-12-14        1053
 210 2020-12-21         980
 211 2020-12-28         642)
```

```
# Create lag features: previous 1, 2, 3 weeks' trip counts
for lag in [1, 2, 3]:
    weekly[f"lag_{lag}"] = weekly["trip_count"].shift(lag)

# Drop first few rows with NaNs from lagging
weekly_ml = weekly.dropna().reset_index(drop=True)

X = weekly_ml[["lag_1", "lag_2", "lag_3"]]
y = weekly_ml["trip_count"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

model_weekly = RandomForestRegressor(random_state=42)
model_weekly.fit(X_train, y_train)

y_pred = model_weekly.predict(X_test)

print("Weekly demand forecasting RMSE:", mean_squared_error(y_test,
y_pred, squared=False))
print("R²:", r2_score(y_test, y_pred))

# Plot actual vs predicted
plt.figure()
plt.plot(y_test.values, label="Actual")
plt.plot(y_pred, label="Predicted")
plt.title("Weekly Trip Demand – Actual vs Predicted")
plt.xlabel("Test Weeks")
plt.ylabel("Trips")
plt.legend()
plt.show()
```

```
----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\
pandas\core\indexes\base.py:3812, in Index.get_loc(self, key)
   3811 try:
-> 3812     return self._engine.get_loc(casted_key)
   3813 except KeyError as err:

File pandas/_libs/index.pyx:167, in
pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/index.pyx:175, in
pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/index_class_helper.pxi:245, in
pandas._libs.index.MaskedUInt32Engine._check_type()

KeyError: 'trip_count'
```

```
The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[56], line 3
      1 # Create lag features: previous 1, 2, 3 weeks' trip counts
      2 for lag in [1, 2, 3]:
----> 3     weekly[f"lag_{lag}"] = weekly["trip_count"].shift(lag)
      5 # Drop first few rows with NaNs from lagging
      6 weekly_ml = weekly.dropna().reset_index(drop=True)

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\
pandas\core\series.py:1130, in Series.__getitem__(self, key)
   1127     return self._values[key]
   1129 elif key_is_scalar:
-> 1130     return self._get_value(key)
   1132 # Convert generator to list before going through hashable part
   1133 # (We will iterate through the generator there to check for
slices)
   1134 if is_iterator(key):

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\
pandas\core\series.py:1246, in Series._get_value(self, label,
takeable)
   1243     return self._values[label]
   1245 # Similar to Index.get_value, but we do not fall back to
positional
-> 1246 loc = self.index.get_loc(label)
   1248 if is_integer(loc):
   1249     return self._values[loc]

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\
pandas\core\indexes\base.py:3819, in Index.get_loc(self, key)
   3814     if isinstance(casted_key, slice) or (
   3815         isinstance(casted_key, abc.Iterable)
   3816         and any(isinstance(x, slice) for x in casted_key)
   3817     ):
   3818         raise InvalidIndexError(key)
-> 3819     raise KeyError(key) from err
   3820 except TypeError:
   3821     # If we have a listlike key, _check_indexing_error will
raise
   3822     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3823     #  the TypeError.
   3824     self._check_indexing_error(key)

KeyError: 'trip_count'
```

```
# ================================
# □ Weekly Aggregation (using previous approach)
# ================================
df['lpep_pickup_datetime'] =
pd.to_datetime(df['lpep_pickup_datetime'])

df['week_start'] =
df['lpep_pickup_datetime'].dt.to_period("W").apply(lambda r:
r.start_time)

weekly = (
    df.groupby('week_start')
      .size()
      .reset_index(name='trip_count')
      .sort_values('week_start')
)

print("Weekly data:")
weekly.head()
```

```
Weekly data:
```

```
  week_start  trip_count
0 2008-12-29           8
1 2010-09-20           3
2 2016-12-26         318
3 2017-01-02        1956
4 2017-01-09        2170
```

```
# xyz
# Create lag features
# xyz
for lag in [1, 2, 3]:
    weekly[f'lag_{lag}'] = weekly['trip_count'].shift(lag)

weekly_ml = weekly.dropna().reset_index(drop=True)

X = weekly_ml[['lag_1', 'lag_2', 'lag_3']]
y = weekly_ml['trip_count']

print(weekly_ml.head())
```

```
  week_start  trip_count   lag_1   lag_2   lag_3
0 2017-01-02        1956   318.0     3.0     8.0
1 2017-01-09        2170  1956.0   318.0     3.0
2 2017-01-16        2006  2170.0  1956.0   318.0
3 2017-01-23        2211  2006.0  2170.0  1956.0
4 2017-01-30        2262  2211.0  2006.0  2170.0
```

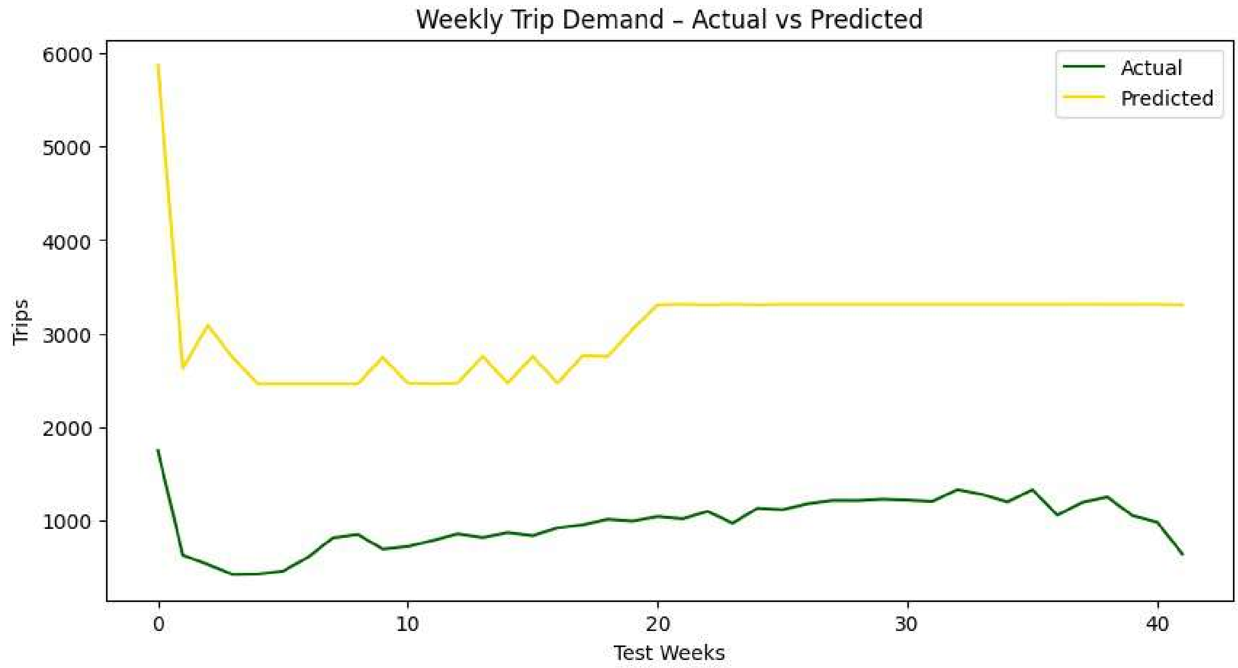
```
# xyz
#Train RandomForest model (previous code style)
```

```
# xyz
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

model_weekly = RandomForestRegressor(random_state=42)
model_weekly.fit(X_train, y_train)

y_pred = model_weekly.predict(X_test)

rmse = mean_squared_error(y_test, y_pred) ** 0.5   # works for all
sklearn versions
print("Weekly demand forecasting RMSE:", rmse)
print("R²:", r2_score(y_test, y_pred))
```

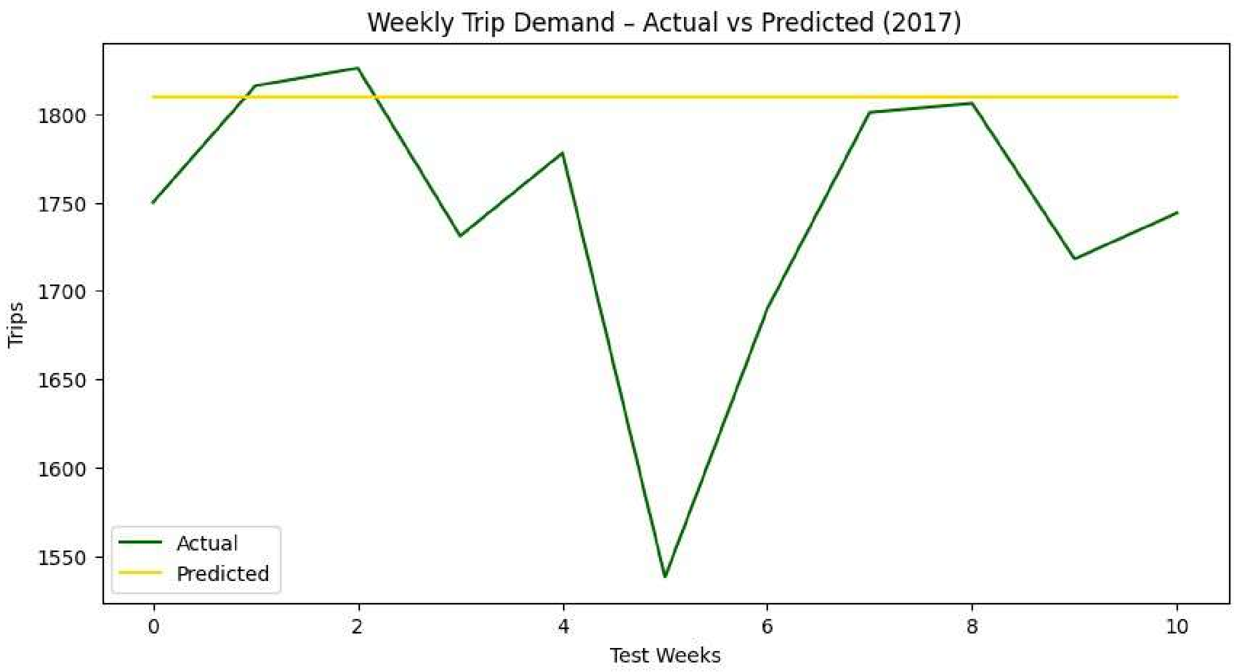
```
Weekly demand forecasting RMSE: 2129.342294309051
R²: -57.46218052500757
```

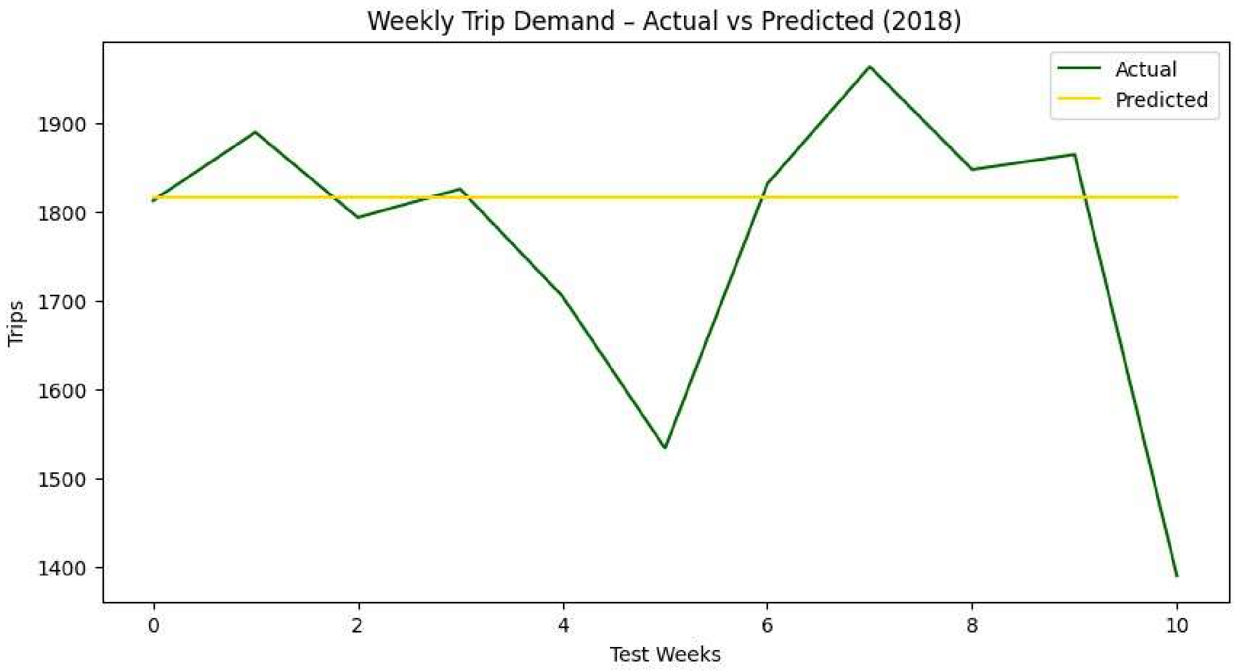
```
# ================================
# □ Plot Actual vs Predicted
# ================================
plt.figure()
plt.figure(figsize=(10,5))
plt.plot(y_test.values, color="#006400", label="Actual")      # Dark
Green
plt.plot(y_pred, color="#FFD700", label="Predicted")          # Dark
Yellow (Gold)
plt.title("Weekly Trip Demand – Actual vs Predicted")
plt.xlabel("Test Weeks")
plt.ylabel("Trips")
plt.legend()
plt.show()
```

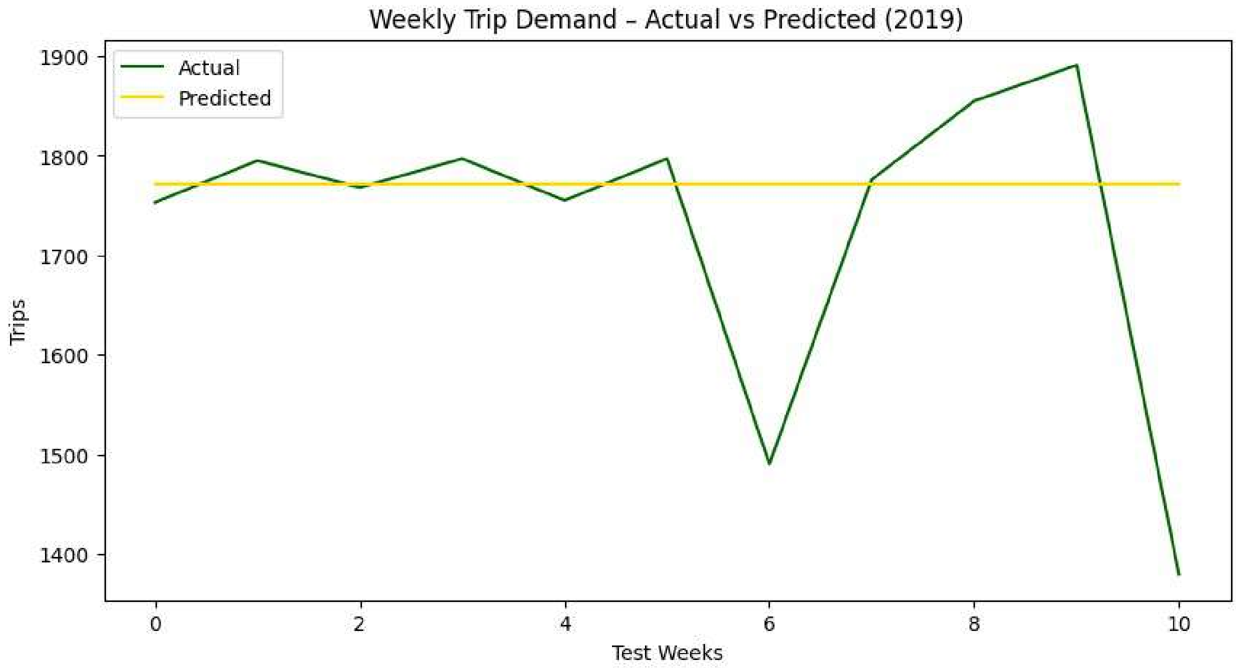
```
<Figure size 640x480 with 0 Axes>
```

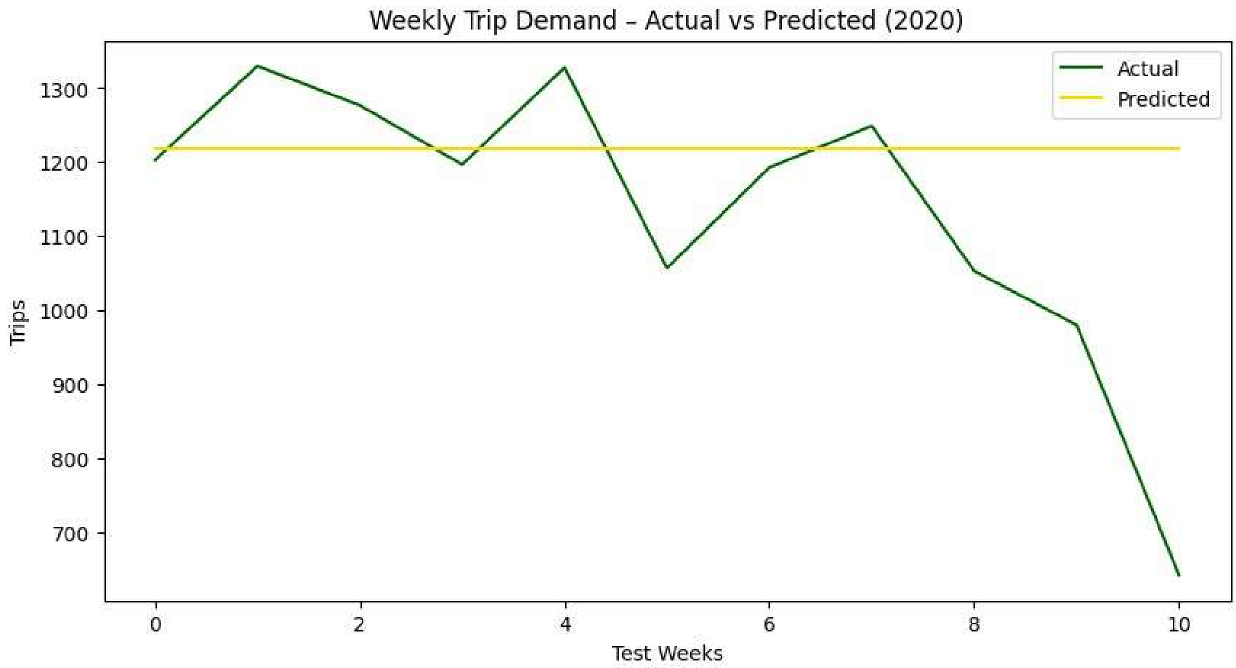


```
# Add year column if not already added
df['year'] = df['lpep_pickup_datetime'].dt.year

unique_years = sorted(df['year'].unique())

for yr in unique_years:
    df_year = df[df['year'] == yr]

    # Recreate weekly grouping for each year
    weekly =
df_year.groupby(df_year['lpep_pickup_datetime'].dt.isocalendar().week)
.size()

    # Split into train-test for each year
    X = weekly.index.values.reshape(-1, 1)
    y = weekly.values

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, shuffle=False)

    # Fit model
    from sklearn.ensemble import RandomForestRegressor
    model = RandomForestRegressor()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Plot
    plt.figure(figsize=(10,5))
    plt.plot(y_test, color="#006400", label="Actual")    # Dark green
    plt.plot(y_pred, color="#FFD700", label="Predicted") # Gold
    plt.title(f"Weekly Trip Demand – Actual vs Predicted ({yr})")
    plt.xlabel("Test Weeks")
    plt.ylabel("Trips")
    plt.legend()
    plt.show()
```









```
# ================================
#  Create High-Tip Target
# ================================
df2 = df.copy()
df2 = df2[df2['tip_amount'] >= 0]  # remove invalid values

avg_tip = df2['tip_amount'].mean()
print("Average tip amount:", avg_tip)
```

```
df2['high_tip'] = (df2['tip_amount'] > avg_tip).astype(int)
```

```
Average tip amount: 1.1060786931477258
```

```
# =====================
#  Create Model Features
# xyz
df2['hour'] = df2['lpep_pickup_datetime'].dt.hour
df2['day_of_week'] = df2['lpep_pickup_datetime'].dt.dayofweek  #
Monday = 0
df2['month'] = df2['lpep_pickup_datetime'].dt.month

# Columns that must exist:
# trip_distance, pickup_borough, dropoff_borough

df2.columns.tolist()
```

```
['VendorID',
 'lpep_pickup_datetime',
 'lpep_dropoff_datetime',
 'store_and_fwd_flag',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'passenger_count',
 'trip_distance',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'payment_type',
 'trip_type',
 'congestion_surcharge',
 'week_start',
 'high_tip',
 'hour',
 'day_of_week',
 'month']
```

```
   # ================================
# □ Select Columns for Model (using existing columns)
# xyz
cat_cols = ['PULocationID', 'DOLocationID']      # categorical
features
num_cols = ['trip_distance', 'passenger_count',
            'hour', 'day_of_week', 'month']      # numeric / time
features
```

```
# Build modeling dataframe
df_model = pd.get_dummies(
    df2[cat_cols + num_cols + ['high_tip']],
    columns=cat_cols,
    drop_first=True
)

X = df_model.drop("high_tip", axis=1)
y = df_model["high_tip"]

print("Model feature shape:", X.shape)
X.head()
```

```
Model feature shape: (399984, 520)

   trip_distance  passenger_count  hour  day_of_week  month  
PULocationID_3  \
0           0.96              1.0    18            2      1   
False   
1           1.12              1.0    14            1      2   
False   
2           0.95              1.0     8            3      3   
False   
3           0.55              5.0    20            1     12   
False   
4           0.63              1.0    14            2      7   
False   

   PULocationID_4  PULocationID_5  PULocationID_6  PULocationID_7  ...  
\
0           False           False           False           False  ...   

1           False           False           False           False  ...   

2           False           False           False           False  ...   

3           False           False           False           False  ...   

4           False           False           False           False  ...   


   DOLocationID_256  DOLocationID_257  DOLocationID_258  
DOLocationID_259  \
0             False             False             False   
False   
1             False             False             False   
False   
2             False             False             False   
False   
3             False             False             False   
```

```
False
4             False             False             False
False

   DOLocationID_260  DOLocationID_261  DOLocationID_262
DOLocationID_263  \
0             False             False             False
False
1             False             False             False
False
2             False             False             False
False
3              True             False             False
False
4             False             False             False
False

   DOLocationID_264  DOLocationID_265
0             False             False
1             False             False
2             False             False
3             False             False
4             False             False

[5 rows x 520 columns]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

clf = RandomForestClassifier(
    n_estimators=150,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7094766053727015

Classification Report:

              precision    recall  f1-score   support

           0       0.75      0.83      0.79     51808
           1       0.61      0.49      0.54     28189

    accuracy                           0.71     79997
   macro avg       0.68      0.66      0.67     79997
weighted avg       0.70      0.71      0.70     79997
```

```
# ================================
# Train RandomForest Classifier
# xyz
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

clf = RandomForestClassifier(
    n_estimators=150,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# ================================
#  Model Evaluation
# xyz
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7094766053727015

Classification Report:

              precision    recall  f1-score   support

           0       0.75      0.83      0.79     51808
           1       0.61      0.49      0.54     28189

    accuracy                           0.71     79997
   macro avg       0.68      0.66      0.67     79997
weighted avg       0.70      0.71      0.70     79997
```

```
!pip install seaborn
```

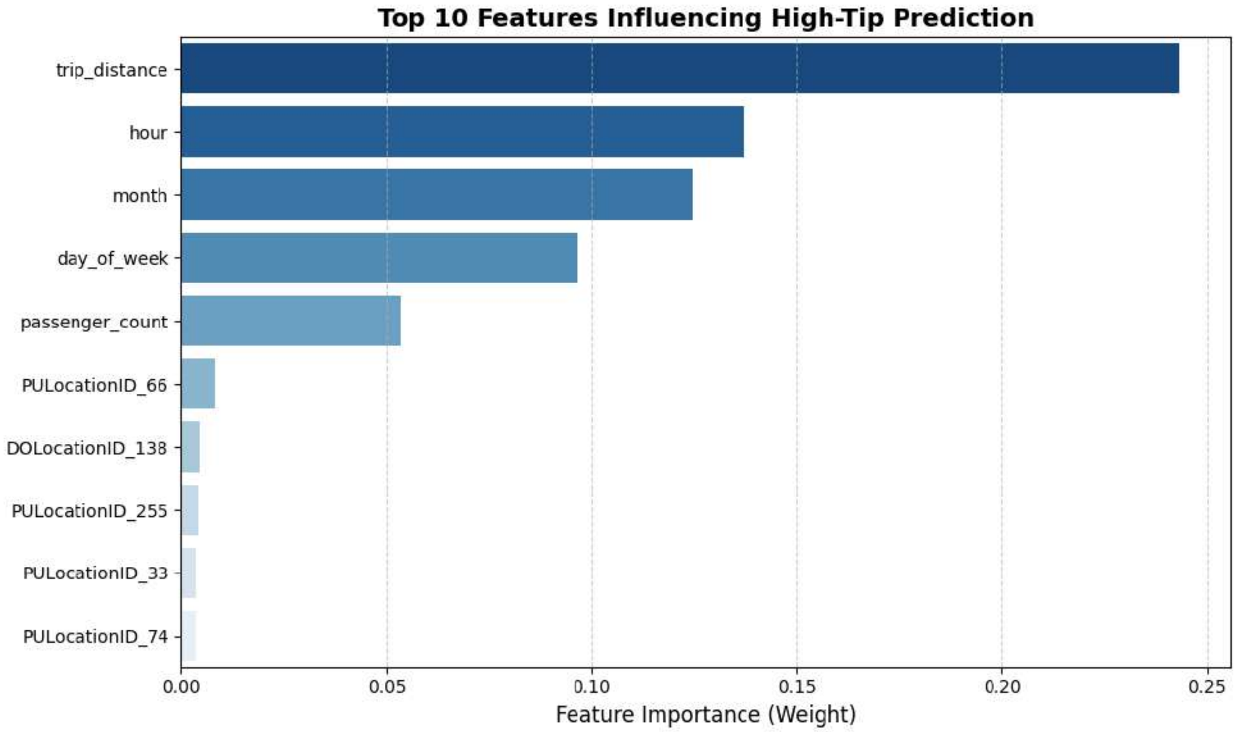
```
Requirement already satisfied: seaborn in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
seaborn) (2.3.2)
Requirement already satisfied: pandas>=1.2 in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (from seaborn)
(2.3.1)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\
parth\appdata\local\programs\python\python313\lib\site-packages (from
seaborn) (3.10.6)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.9)
Requirement already satisfied: packaging>=20.0 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (25.0)
Requirement already satisfied: pillow>=8 in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (from pandas>=1.2-
>seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\parth\
appdata\local\programs\python\python313\lib\site-packages (from
pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\parth\appdata\
local\programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)

WARNING: Ignoring invalid distribution ~treamlit (C:\Users\parth\
AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~treamlit (C:\Users\parth\
AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~treamlit (C:\Users\parth\
AppData\Local\Programs\Python\Python313\Lib\site-packages)

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```python
# Feature Importance
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Use your trained RandomForestClassifier
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]   # sort descending
features_list = X.columns

feat_imp_df = pd.DataFrame({
    'Feature': features_list[indices],
    'Importance': importances[indices]
})

feat_imp_df.head(10)
```

```
             Feature  Importance
0      trip_distance    0.243412
1               hour    0.137212
2              month    0.124673
3        day_of_week    0.096598
4    passenger_count    0.053187
5    PULocationID_66    0.008116
6   DOLocationID_138    0.004482
7   PULocationID_255    0.004215
8    PULocationID_33    0.003505
9    PULocationID_74    0.003426
```

```python
#  Top 10 Most Important Features
# xyz
plt.figure(figsize=(10,6))

sns.barplot(
    data = feat_imp_df.head(10),
    x = 'Importance',
    y = 'Feature',
    palette = 'Blues_r',
    dodge = False
)

plt.title("Top 10 Features Influencing High-Tip Prediction",
          fontsize=14, weight='bold')

plt.xlabel("Feature Importance (Weight)", fontsize=12)
plt.ylabel("")
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
C:\Users\parth\AppData\Local\Temp\ipykernel_15004\2225236825.py:6:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(
```



```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Feature Importance Calculation
# xyz
import numpy as np
import pandas as pd

importances = clf.feature_importances_      # <-- IMPORTANT
indices = np.argsort(importances)[::-1]
features_list = X.columns
```

```
feat_imp_df = pd.DataFrame({
    'Feature': features_list[indices],
    'Importance': importances[indices]
})

feat_imp_df.head()
```

```
           Feature  Importance
0    trip_distance    0.243412
1             hour    0.137212
2            month    0.124673
3      day_of_week    0.096598
4  passenger_count    0.053187
```

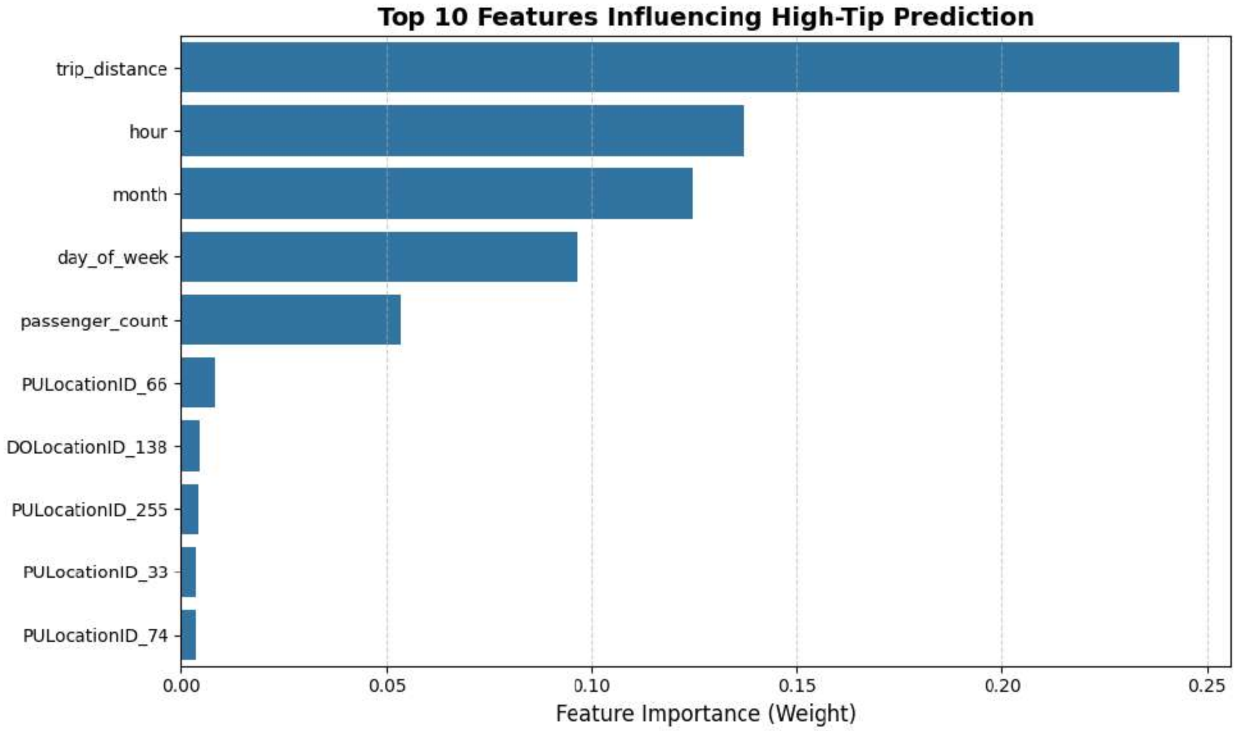
```
cat_cols = ['PULocationID', 'DOLocationID']
num_cols = ['trip_distance', 'passenger_count', 'hour', 'day_of_week',
'month']

df_model = pd.get_dummies(
    df2[cat_cols + num_cols + ['high_tip']],
    columns=cat_cols,
    drop_first=True
)

X = df_model.drop("high_tip", axis=1)
y = df_model["high_tip"]

print(X.shape, y.shape)
```

```
(399984, 520) (399984,)
```

```
# High-Tip Prediction Model + Top 10 Feature Importance
# xyz

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1 Build feature matrix X and target y
cat_cols = ['PULocationID', 'DOLocationID']   # categorical
num_cols = ['trip_distance', 'passenger_count',
            'hour', 'day_of_week', 'month']   # numeric / time

df_model = pd.get_dummies(
    df2[cat_cols + num_cols + ['high_tip']],
    columns=cat_cols,
    drop_first=True
)

X = df_model.drop('high_tip', axis=1)
y = df_model['high_tip']

print("Feature matrix shape:", X.shape)
print("Target distribution:\n", y.value_counts(normalize=True))

# 2 Train / Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 3 Train RandomForest classifier
clf = RandomForestClassifier(
    n_estimators=150,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# 4 Evaluation
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# 5 Feature importance calculation
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
features_list = X.columns

feat_imp_df = pd.DataFrame({
    'Feature': features_list[indices],
    'Importance': importances[indices]
})

print("\nTop 10 features by importance:\n", feat_imp_df.head(10))

# 6 Plot Top 10 Most Important Features
plt.figure(figsize=(10, 6))
sns.barplot(
    data=feat_imp_df.head(10),
    x='Importance',
    y='Feature',
    dodge=False
```

```
)

plt.title("Top 10 Features Influencing High-Tip Prediction",
          fontsize=14, weight='bold')
plt.xlabel("Feature Importance (Weight)", fontsize=12)
plt.ylabel("")
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
Feature matrix shape: (399984, 520)
Target distribution:
 high_tip
0    0.647626
1    0.352374
Name: proportion, dtype: float64

Accuracy: 0.7094641049039339

Classification Report:

              precision    recall  f1-score   support

           0       0.75      0.83      0.79     51808
           1       0.61      0.49      0.54     28189

    accuracy                           0.71     79997
   macro avg       0.68      0.66      0.67     79997
weighted avg       0.70      0.71      0.70     79997


Top 10 features by importance:
             Feature  Importance
0      trip_distance    0.243412
1               hour    0.137212
2              month    0.124673
3        day_of_week    0.096598
4    passenger_count    0.053187
5    PULocationID_66    0.008116
6   DOLocationID_138    0.004482
7   PULocationID_255    0.004215
8    PULocationID_33    0.003505
9    PULocationID_74    0.003426
```



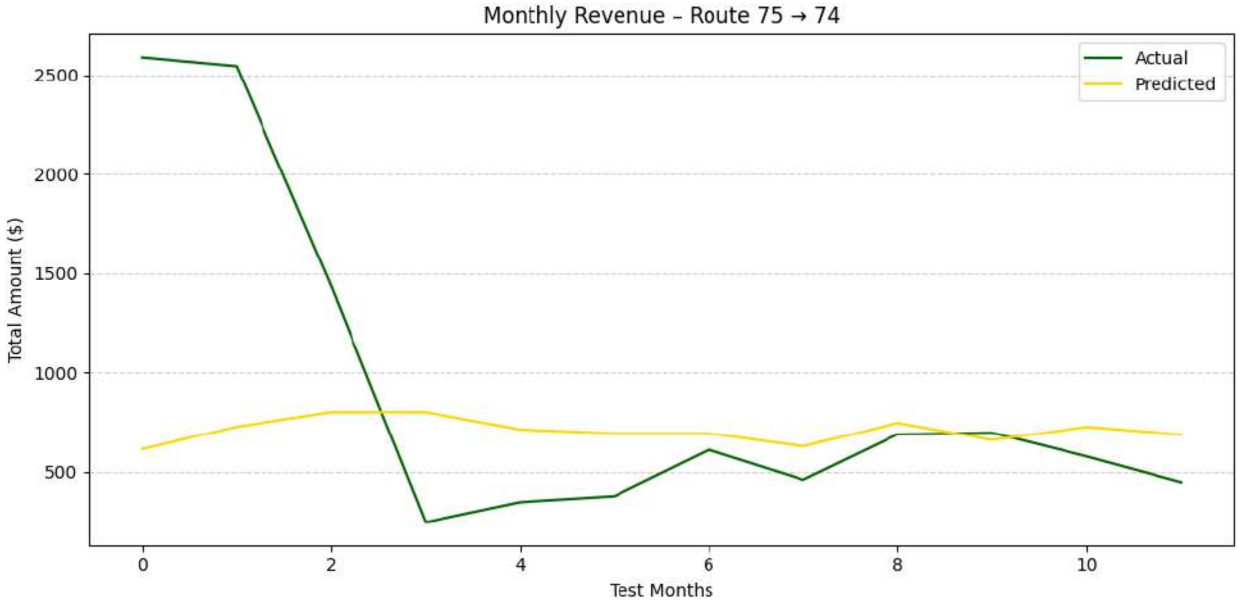
```
# Route-Based Monthly Revenue Forecasting


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# 1 Ensure correct datetime format
df['lpep_pickup_datetime'] =
pd.to_datetime(df['lpep_pickup_datetime'], errors='coerce')

# 2 Identify top route (highest trip count)
route_counts = (
    df.groupby(['PULocationID', 'DOLocationID'])
      .size()
      .reset_index(name='trip_count')
      .sort_values('trip_count', ascending=False)
)

print("Top 10 busiest routes:")
display(route_counts.head(10))

# Automatically pick the most frequent route
```

```
top_route = route_counts.iloc[0]
pu_id = int(top_route['PULocationID'])
do_id = int(top_route['DOLocationID'])

print(f"\nUsing Route: PULocationID={pu_id} → DOLocationID={do_id}")

# 3 Filter data for selected route
route_df = df[(df['PULocationID'] == pu_id) &
              (df['DOLocationID'] == do_id)].copy()

print("Total records for selected route:", len(route_df))

# 4 Create monthly revenue
route_df['year_month'] =
route_df['lpep_pickup_datetime'].dt.to_period('M').dt.to_timestamp()

monthly_rev = (
    route_df.groupby('year_month')['total_amount']
            .sum()
            .reset_index(name='monthly_revenue')
            .sort_values('year_month')
)

print("\nMonthly revenue preview:")
display(monthly_rev.head())

# 5 Create lag features
monthly_rev['lag_1'] = monthly_rev['monthly_revenue'].shift(1)
monthly_rev['lag_2'] = monthly_rev['monthly_revenue'].shift(2)

# Drop NaNs
monthly_ml = monthly_rev.dropna().reset_index(drop=True)

X = monthly_ml[['lag_1', 'lag_2']]
y = monthly_ml['monthly_revenue']

# 6 Train / Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, shuffle=False  # keep time order
)

# 7 Train RandomForestRegressor
model = RandomForestRegressor(
    n_estimators=150,
    random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
#8 8 Performance Metrics
rmse = mean_squared_error(y_test, y_pred) ** 0.5
r2 = r2_score(y_test, y_pred)

print("\n□ Route Revenue Forecasting Results")
print("RMSE:", rmse)
print("R² Score:", r2)

#9 9 Plot Actual vs Predicted
plt.figure(figsize=(10,5))
plt.plot(y_test.values, label="Actual", color="#006400")       # Dark
green
plt.plot(y_pred, label="Predicted", color="#FFD700")          # Dark
yellow
plt.title(f"Monthly Revenue – Route {pu_id} → {do_id}")
plt.xlabel("Test Months")
plt.ylabel("Total Amount ($)")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
Top 10 busiest routes:

      PULocationID  DOLocationID  trip_count
7029            75            74        4594
130              7             7        4545
6822            74            75        3830
3473            41            42        3789
9340            95            95        3358
6821            74            74        3056
8039            82           129        3012
3672            42            42        3001
3472            41            41        2934
3496            41            74        2809


Using Route: PULocationID=75 → DOLocationID=74
Total records for selected route: 4594

Monthly revenue preview:

  year_month  monthly_revenue
0 2017-01-01           706.56
1 2017-02-01           716.12
2 2017-03-01           815.45
3 2017-04-01           655.81
4 2017-05-01           736.07


□ Route Revenue Forecasting Results
```

```
RMSE: 829.7019772888709
R² Score: -0.09976580535236113
```



```
taxi_zones = pd.read_csv(
    "C:/Users/parth/OneDrive/Desktop/Trips 2/taxi_zones.csv")

df = df.merge(
    taxi_zones[['LocationID', 'Borough']],
    left_on='PULocationID',
    right_on='LocationID',
    how='left'
)

df = df.rename(columns={'Borough': 'pickup_borough'})
df = df.drop(columns=['LocationID'])

df = df.merge(
    taxi_zones[['LocationID', 'Borough']],
    left_on='DOLocationID',
    right_on='LocationID',
    how='left'
)

df = df.rename(columns={'Borough': 'dropoff_borough'})
df = df.drop(columns=['LocationID'])

df.columns.tolist()
```

```
['VendorID',
 'lpep_pickup_datetime',
 'lpep_dropoff_datetime',
 'store_and_fwd_flag',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'passenger_count',
 'trip_distance',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'payment_type',
 'trip_type',
 'congestion_surcharge',
 'week_start',
 'pickup_borough',
 'dropoff_borough',
 'dropoff_borough',
 'dropoff_borough',
 'pickup_borough',
 'dropoff_borough']
```

```
# Keep only first occurrence of each column name
df = df.loc[:, ~df.columns.duplicated()]

df.columns.tolist()   # just to check, you should now see each name
only once
```

```
['VendorID',
 'lpep_pickup_datetime',
 'lpep_dropoff_datetime',
 'store_and_fwd_flag',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'passenger_count',
 'trip_distance',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'payment_type',
 'trip_type',
 'congestion_surcharge',
 'week_start',
 'pickup_borough',
 'dropoff_borough']
```

```
df['route'] = df['pickup_borough'] + " → " + df['dropoff_borough']
df[['pickup_borough', 'dropoff_borough', 'route']].head()
```

```
C:\Users\parth\AppData\Local\Temp\ipykernel_15004\3927191911.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['route'] = df['pickup_borough'] + " → " + df['dropoff_borough']
```

```
  pickup_borough dropoff_borough                  route
0       Brooklyn        Brooklyn    Brooklyn → Brooklyn
1       Brooklyn        Brooklyn    Brooklyn → Brooklyn
2      Manhattan       Manhattan  Manhattan → Manhattan
3         Queens          Queens        Queens → Queens
4       Brooklyn        Brooklyn    Brooklyn → Brooklyn
```

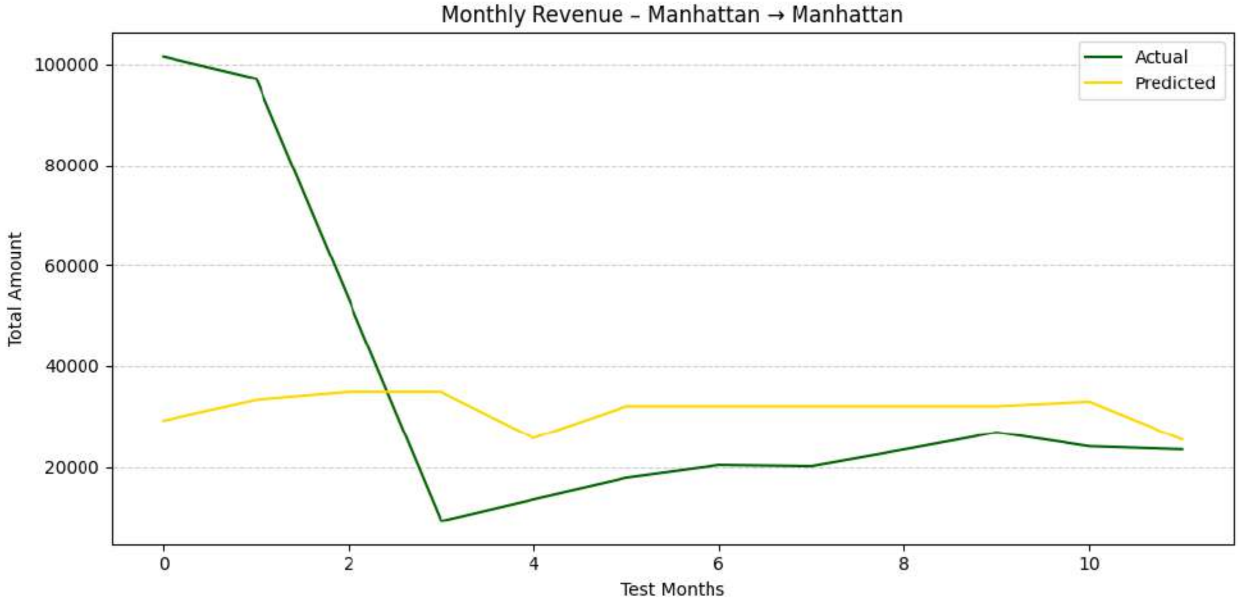
```
df.loc[:, 'route'] = df['pickup_borough'] + " → " +
df['dropoff_borough']

# 1. Filter to years 2017–2020  (if not already done)

df['lpep_pickup_datetime'] =
pd.to_datetime(df['lpep_pickup_datetime'], errors='coerce')
df = df[(df['lpep_pickup_datetime'].dt.year >= 2017) &
        (df['lpep_pickup_datetime'].dt.year <= 2020)]

# 2. Check top borough→borough routes

print("Top 10 borough routes by trip count:")
print(df['route'].value_counts().head(10))

# Pick one route to forecast (you can change this string)
target_route = df['route'].value_counts().index[0]   # or e.g.
"Manhattan → Brooklyn"
print("\nUsing route:", target_route)

route_df = df[df['route'] == target_route].copy()

# =======================================================
# 3. Build monthly revenue series for that route
# =======================================================
route_df['year_month'] =
route_df['lpep_pickup_datetime'].dt.to_period('M').dt.to_timestamp()
```

```
monthly_rev = (route_df
               .groupby('year_month')['total_amount']
               .sum()
               .reset_index(name='monthly_revenue')
               .sort_values('year_month'))

print("\nMonthly revenue preview:")
print(monthly_rev.head())


# 4. Create lag features

monthly_rev['lag_1'] = monthly_rev['monthly_revenue'].shift(1)
monthly_rev['lag_2'] = monthly_rev['monthly_revenue'].shift(2)

# Drop first rows with NaN lags
monthly_ml = monthly_rev.dropna().reset_index(drop=True)

X = monthly_ml[['lag_1', 'lag_2']]
y = monthly_ml['monthly_revenue']


# 5. Train / test split
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, shuffle=False
)

model = RandomForestRegressor(
    n_estimators=150,
    random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

rmse = mean_squared_error(y_test, y_pred) ** 0.5
r2 = r2_score(y_test, y_pred)

print("\n□ Route Revenue Forecasting Results (borough based)")
print("Route:", target_route)
print("RMSE:", rmse)
print("R²:", r2)


# 6. Plot Actual vs Predicted monthly revenue
```

```
plt.figure(figsize=(10,5))
plt.plot(y_test.values, label="Actual", color="#006400")      # dark green
plt.plot(y_pred, label="Predicted", color="#FFD700")         # dark yellow
plt.title(f"Monthly Revenue – {target_route}")
plt.xlabel("Test Months")
plt.ylabel("Total Amount")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
C:\Users\parth\AppData\Local\Temp\ipykernel_15004\2662573679.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['lpep_pickup_datetime'] =
pd.to_datetime(df['lpep_pickup_datetime'], errors='coerce')

Top 10 borough routes by trip count:
route
Manhattan → Manhattan    119029
Queens → Queens           97614
Brooklyn → Brooklyn       97315
Bronx → Bronx             19023
Brooklyn → Manhattan      17506
Manhattan → Bronx         11054
Queens → Manhattan         7588
Brooklyn → Queens          6708
Bronx → Manhattan          6047
Queens → Brooklyn          5205
Name: count, dtype: int64

Using route: Manhattan → Manhattan

Monthly revenue preview:
  year_month  monthly_revenue
0 2017-01-01         29748.80
1 2017-02-01         29763.81
2 2017-03-01         31136.45
3 2017-04-01         29900.01
4 2017-05-01         30307.12

□ Route Revenue Forecasting Results (borough based)
Route: Manhattan → Manhattan
```

```
RMSE: 30430.07166554574
R²: -0.017662602963947505
```


Monthly Revenue – Manhattan → Manhattan
Actual
Predicted
Total Amount
Test Months