databricks nyc_taxi_trip_analysis

(https://databricks.com)

NYC TAXI TRIP ANALYSIS

2

```
%python
%pip install folium
```

```
Requirement already satisfied: requests in /databricks/python3/lib/python3.9/site-packages (from folium) (2.27.1)
Requirement already satisfied: jinja2>=2.9 in /databricks/python3/lib/python3.9/site-packages (from folium) (2.1
1.3)
Collecting branca>=0.6.0
  Downloading branca-0.8.0-py3-none-any.whl (25 kB)
Collecting xyzservices
  Downloading xyzservices-2024.9.0-py3-none-any.whl (85 kB)
Requirement already satisfied: numpy in /databricks/python3/lib/python3.9/site-packages (from folium) (1.21.5)
Collecting jinja2>=2.9
  Downloading jinja2-3.1.4-py3-none-any.whl (133 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /databricks/python3/lib/python3.9/site-packages (from jinja2>=
2.9->folium) (2.0.1)
Requirement already satisfied: charset-normalizer~=2.0.0 in /databricks/python3/lib/python3.9/site-packages (from
requests->folium) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /databricks/python3/lib/python3.9/site-packages (from requests->fo
lium) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /databricks/python3/lib/python3.9/site-packages (from reques
ts->folium) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /databricks/python3/lib/python3.9/site-packages (from req
uests->folium) (1.26.9)
Installing collected packages: jinja2, xyzservices, branca, folium
```

3

```
%python
#load Dataset
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/aryxnjain@gmail.com/ye
# Creating a temporary SQL view
df.createOrReplaceTempView("tripdata")
df
```

```
Out[1]: DataFrame[VendorID: string, tpep_pickup_datetime: string, tpep_dropoff_datetime: string, passenger_count: st
ring, trip_distance: string, pickup_longitude: string, pickup_latitude: string, RateCodeID: string, store_and_fwd_fl
ag: string, dropoff_longitude: string, dropoff_latitude: string, payment_type: string, fare_amount: string, extra: s
tring, mta_tax: string, tip_amount: string, tolls_amount: string, improvement_surcharge: string, total_amount: strin
g]
```

Outlier Detection

5

```
SELECT *
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) > 1000
ORDER BY CAST(fare_amount AS DOUBLE) DESC
```

Table

2 rows

6

```
SELECT *
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) <= 0
ORDER BY CAST(fare_amount AS DOUBLE) ASC
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
1	2	2015-01-11 03:56:15	2015-01-11 03:56:17	1	.00
2	2	2015-01-15 15:20:22	2015-01-15 15:20:26	2	.00
3	2	2015-01-10 02:49:03	2015-01-10 02:50:35	2	.00
4	2	2015-01-04 19:23:11	2015-01-04 19:24:49	1	.00
5	2	2015-01-29 01:51:19	2015-01-29 02:03:17	1	.78
6	2	2015-01-25 02:37:13	2015-01-25 02:37:18	1	.03
7	2	2015-01-06 11:58:14	2015-01-06 12:00:18	1	.00
8	2	2015-01-20 07:17:16	2015-01-20 07:18:12	1	.00
9	2	2015-01-11 02:51:25	2015-01-11 02:51:46	2	.14
10	2	2015-01-11 02:53:28	2015-01-11 03:00:42	2	.00
11	2	2015-01-31 04:19:19	2015-01-31 04:20:03	1	.00
12	2	2015-01-25 06:54:55	2015-01-25 06:55:21	1	.00
13	2	2015-01-06 13:03:12	2015-01-06 13:07:54	2	.00
14	2	2015-01-04 02:59:38	2015-01-04 03:00:54	2	.00

7,666 rows

Correlation Analysis

8

```
-- correlation between fare amount, total amount, and trip distance for trips with positive values
SELECT
    CORR(CAST(fare_amount AS DOUBLE), CAST(trip_distance AS DOUBLE)) AS fare_distance_correlation,
    CORR(CAST(total_amount AS DOUBLE), CAST(trip_distance AS DOUBLE)) AS total_distance_correlation
FROM tripdata
WHERE
    CAST(fare_amount AS DOUBLE) > 0
    AND CAST(total_amount AS DOUBLE) > 0
    AND CAST(trip_distance AS DOUBLE) > 0
```

Table

Fare Amount vs. Trip Distance:

The correlation between fare amount and trip distance was around 0.00046, suggesting that how far a taxi travels doesn't strongly affect the fare. This is surprising since we'd expect longer trips to have higher fares. This weak correlation could mean other factors, like base charges, surcharges, or even traffic conditions, have more influence on the fare than the actual distance traveled. Total Amount vs. Trip Distance:

The correlation between total amount and trip distance was even smaller at 0.0000339, implying that the total cost (including tips, tolls, and surcharges) also doesn't depend much on how far the trip is. This suggests that fixed costs, like base fare or additional charges, might make the trip distance less important in determining the final price.

Trip Duration Prediction

11

```
-- calculate the average trip duration (in minutes) grouped by passenger count
SELECT
    CAST(passenger_count AS INT) AS passenger_count,
    AVG(
        (UNIX_TIMESTAMP(CAST(tpep_dropoff_datetime AS TIMESTAMP)) -
         UNIX_TIMESTAMP(CAST(tpep_pickup_datetime AS TIMESTAMP))) / 60
    ) AS avg_duration_minutes
FROM tripdata
WHERE CAST(passenger_count AS INT) > 0
GROUP BY CAST(passenger_count AS INT)
ORDER BY CAST(passenger_count AS INT)
```

Table

Trip Clustering

13

```
-- Categorize trips into distance bins and calculate the average fare for each bin, ordered by distance
WITH distance_bins AS (
    SELECT
        CASE
            WHEN trip_distance < 1 THEN '<1 mile'
            WHEN trip_distance >= 1 AND trip_distance <= 2 THEN '1-2 miles'
            WHEN trip_distance > 2 AND trip_distance <= 5 THEN '2-5 miles'
            ELSE '>5 miles'
        END AS distance_bin,
        CAST(fare_amount AS DOUBLE) AS fare_amount,
        CASE
            WHEN trip_distance < 1 THEN 0
            WHEN trip_distance >= 1 AND trip_distance <= 2 THEN 1
            WHEN trip_distance > 2 AND trip_distance <= 5 THEN 2
            ELSE 3
        END AS distance_order
    FROM tripdata
)
SELECT
    distance_bin,
    AVG(fare_amount) AS avg_fare_amount
FROM distance_bins
GROUP BY distance_bin, distance_order
ORDER BY distance_order
```

Table

Fare Amount vs. Distance Analysis

15

```
-- Categorize trips into distance bins and calculate the average fare for each bin, ordered logically by
distance
SELECT
    CASE
        WHEN CAST(trip_distance AS DOUBLE) < 1 THEN '< 1 mile'
        WHEN CAST(trip_distance AS DOUBLE) < 2 THEN '1-2 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 3 THEN '2-3 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 4 THEN '3-4 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 5 THEN '4-5 miles'
        ELSE '> 5 miles'
    END AS distance_bin,
    AVG(CAST(fare_amount AS DOUBLE)) AS avg_fare_amount
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) > 0 AND CAST(trip_distance AS DOUBLE) > 0
GROUP BY
    CASE
        WHEN CAST(trip_distance AS DOUBLE) < 1 THEN '< 1 mile'
        WHEN CAST(trip_distance AS DOUBLE) < 2 THEN '1-2 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 3 THEN '2-3 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 4 THEN '3-4 miles'
        WHEN CAST(trip_distance AS DOUBLE) < 5 THEN '4-5 miles'
        ELSE '> 5 miles'
    END
ORDER BY
    CASE distance_bin
        WHEN '< 1 mile' THEN 1
        WHEN '1-2 miles' THEN 2
        WHEN '2-3 miles' THEN 3
        WHEN '3-4 miles' THEN 4
        WHEN '4-5 miles' THEN 5
        ELSE 6
    END
```

Table

Findings show that the average fare amount increases as the distance increases.

Passenger Count Distribution

18

```
-- Number of trips and Average fare amount for each passenger count
SELECT
    CAST(passenger_count AS INT) AS passenger_count,
    COUNT(*) AS trip_count,
    AVG(CAST(fare_amount AS DOUBLE)) AS avg_fare_amount
FROM tripdata
WHERE CAST(passenger_count AS INT) > 0
  AND CAST(fare_amount AS DOUBLE) > 0
GROUP BY CAST(passenger_count AS INT)
ORDER BY CAST(passenger_count AS INT)
```

Table

Heatmap of Trip Frequencies

20

```
-- Most frequent pickup locations
SELECT
    ROUND(CAST(pickup_latitude AS DOUBLE), 3) AS pickup_latitude,
    ROUND(CAST(pickup_longitude AS DOUBLE), 3) AS pickup_longitude,
    COUNT(*) AS trip_count
FROM tripdata
WHERE
    CAST(pickup_latitude AS DOUBLE) IS NOT NULL
    AND CAST(pickup_longitude AS DOUBLE) IS NOT NULL
GROUP BY
    ROUND(CAST(pickup_latitude AS DOUBLE), 3),
    ROUND(CAST(pickup_longitude AS DOUBLE), 3)
ORDER BY trip_count DESC
LIMIT 10;  -- top 10 only
```

Table

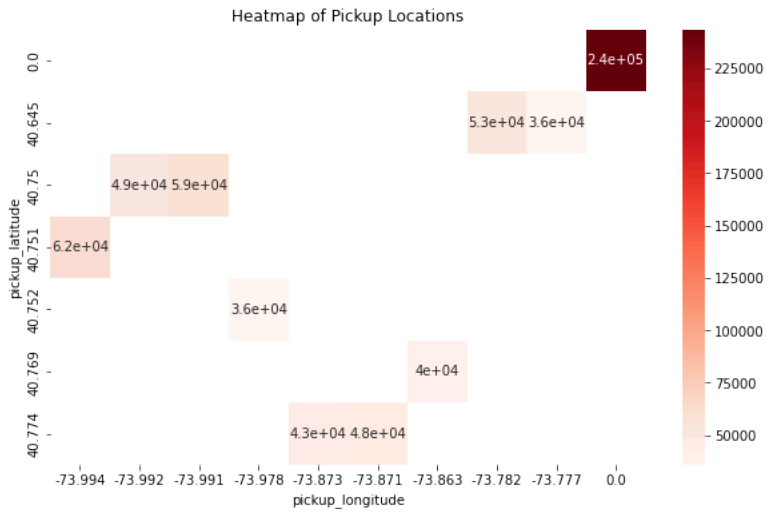
21

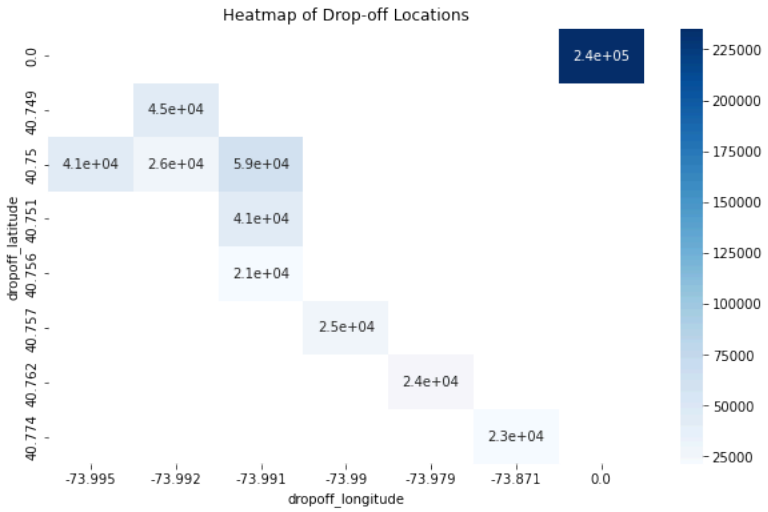
```
-- Most frequent drop off locations
SELECT
    ROUND(CAST(dropoff_latitude AS DOUBLE), 3) AS dropoff_latitude,
    ROUND(CAST(dropoff_longitude AS DOUBLE), 3) AS dropoff_longitude,
    COUNT(*) AS trip_count
FROM tripdata
WHERE
    CAST(dropoff_latitude AS DOUBLE) IS NOT NULL
    AND CAST(dropoff_longitude AS DOUBLE) IS NOT NULL
GROUP BY
    ROUND(CAST(dropoff_latitude AS DOUBLE), 3),
    ROUND(CAST(dropoff_longitude AS DOUBLE), 3)
ORDER BY trip_count DESC
LIMIT 10;  -- top 10 only
```

Table

22

```
%python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pickup_data = {
    'pickup_latitude': [0,40.751,40.75,40.645,40.75,40.774,40.774,40.769,40.645,40.752],
    'pickup_longitude': [0,-73.994,-73.991,-73.782,-73.992,-73.871,-73.873,-73.863,-73.777,-73.978],
    'trip_count': [243478,61880,58728,53204,49092,47972,43382,39928,36112,35927]
}

dropoff_data = {
    'dropoff_latitude': [0,40.75,40.749,40.75,40.751,40.75,40.757,40.762,40.774,40.756],
    'dropoff_longitude': [0,-73.991,-73.992,-73.995,-73.991,-73.992,-73.99,-73.979,-73.871,-73.991],
    'trip_count': [235318,58666,45459,41244,40824,25570,24678,23961,22701,21301]
}

pickup_df = pd.DataFrame(pickup_data)
dropoff_df = pd.DataFrame(dropoff_data)

#heatmap pickup
plt.figure(figsize=(10, 6))
sns.heatmap(pickup_df.pivot("pickup_latitude", "pickup_longitude", "trip_count"), cmap="Reds", annot=True)
plt.title("Heatmap of Pickup Locations")
plt.show()

#heatmap dropoff
plt.figure(figsize=(10, 6))
sns.heatmap(dropoff_df.pivot("dropoff_latitude", "dropoff_longitude", "trip_count"), cmap="Blues", annot=True)
plt.title("Heatmap of Drop-off Locations")
plt.show()
```





Busiest Days and Times Analysis

24

```
-- Revenue each day
SELECT
    DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS day_of_week,
    SUM(CAST(fare_amount AS DOUBLE)) AS total_revenue
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) > 0
GROUP BY DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP))
ORDER BY total_revenue DESC;
```

Table

25

```
-- Busiest hour of the day based on number of pickups
SELECT
    HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS hour_of_day,
    COUNT(*) AS pickup_count
FROM tripdata
GROUP BY HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP))
ORDER BY pickup_count DESC;
```

Table

Findings show that the busiest times are 5-10 PM, with the busiest time being 7 PM.

Trip Duration and Time of Day Analysis

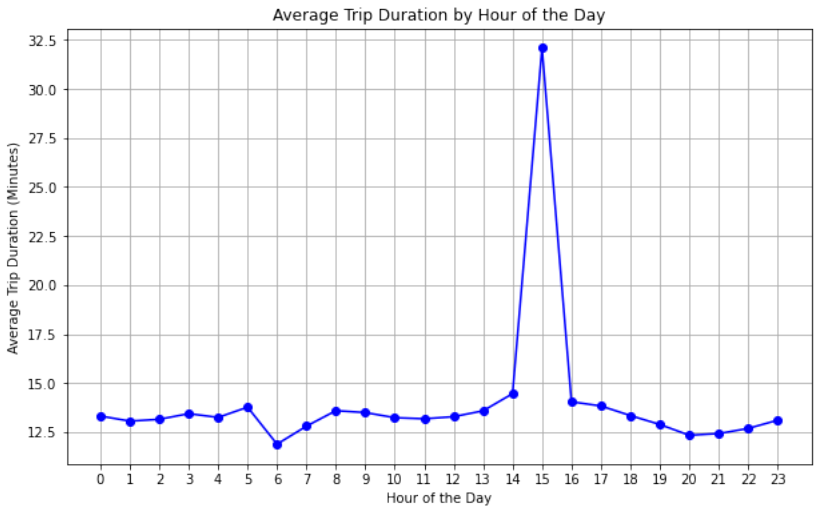
28

```
-- Average trip duration for each hour of the day
SELECT
    HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS hour_of_day,
    AVG(
        (UNIX_TIMESTAMP(CAST(tpep_dropoff_datetime AS TIMESTAMP)) -
         UNIX_TIMESTAMP(CAST(tpep_pickup_datetime AS TIMESTAMP))) / 60
    ) AS avg_duration_minutes
FROM tripdata
WHERE
    CAST(tpep_pickup_datetime AS TIMESTAMP) IS NOT NULL
    AND CAST(tpep_dropoff_datetime AS TIMESTAMP) IS NOT NULL
GROUP BY HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP))
ORDER BY hour_of_day;
```

Table

29

```
%python
data = {
    'hour_of_day': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23],
    'avg_duration_minutes': [13.313322275913483,13.054973602331446,13.145008571616827,13.435592589980732,13.242055
    13.587299440016215,13.489896511813681,13.228424923713806,13.17238671771968,13.280298749710424,13.57990584556744
    826888584857201,13.33810871529094,12.883962532444134,12.333422598934755,12.416918945987238,12.678930717359632,
}

df_data = pd.DataFrame(data)

#plot
plt.figure(figsize=(10, 6))
plt.plot(df_data['hour_of_day'], df_data['avg_duration_minutes'], marker='o', color='b')
plt.title('Average Trip Duration by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Trip Duration (Minutes)')
plt.xticks(df_data['hour_of_day'])  # Show all hours
plt.grid(True)
plt.show()
```



From the graph, we learned that the trip duration peaks at 3 PM, and is at its lowest at 6 AM, but doesn't vary greatly outside of those two.

Payment Type Fare Comparison

32

```
-- Average fare amounts by payment type.
SELECT
    payment_type,
    COUNT(*) AS trip_count,
    AVG(CAST(fare_amount AS DOUBLE)) AS avg_fare_amount
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) > 0
GROUP BY payment_type
ORDER BY avg_fare_amount DESC;
```

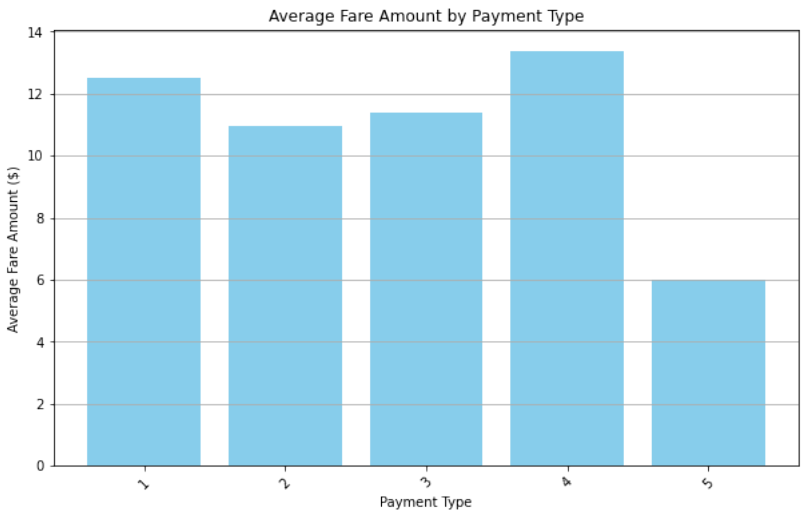
Table

	payment_type	trip_count	avg_fare_amount
1	4	9903	13.389695041906496
2	1	7880666	12.502455255685902
3	3	36377	11.40895290980564
4	2	4814373	10.956297484636094
5	5	1	6

5 rows

33

```
%python
# bar chart comparing average fares for each payment type.
data = {
    'payment_type': [4,1,3,2,5],
    'avg_fare_amount': [13.389695041906496,12.502455255685902,11.40895290980564,10.956297484636094,6]
}

df_data = pd.DataFrame(data)

#plot
plt.figure(figsize=(10, 6))
plt.bar(df_data['payment_type'], df_data['avg_fare_amount'], color='skyblue')
plt.title('Average Fare Amount by Payment Type')
plt.xlabel('Payment Type')
plt.ylabel('Average Fare Amount ($)')
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.grid(True, axis='y')
plt.show()
```



Time Series Analysis of Trips

35

```
%python
from pyspark.sql.functions import to_timestamp, col

# Parsing 'tpep_pickup_datetime' as a timestamp
df_with_date = df.withColumn("trip_date", to_timestamp(col("tpep_pickup_datetime"), "yyyy-MM-dd HH:mm:ss"))

# Creating a temporary view with the parsed timestamp
df_with_date.createOrReplaceTempView("tripdata_with_date")
```

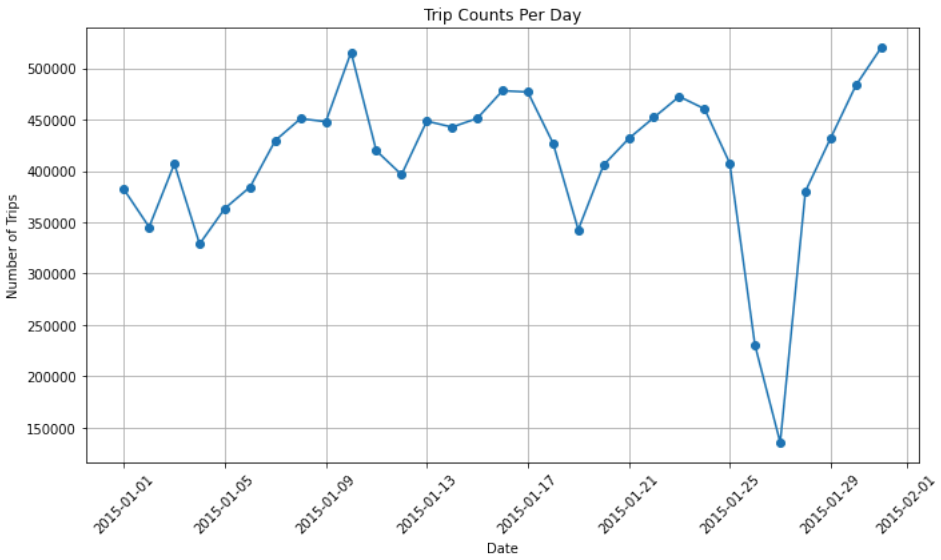
36

```
SELECT trip_date, COUNT(*) AS trip_count
FROM tripdata_with_date
GROUP BY trip_date
ORDER BY trip_date
```

Table

37

```
%python
import pandas as pd
import matplotlib.pyplot as plt

# Running the SQL query and converting the results to Pandas
trip_counts_per_day = spark.sql("""
    SELECT DATE(trip_date) AS trip_day, COUNT(*) AS trip_count
    FROM tripdata_with_date
    GROUP BY DATE(trip_date)
    ORDER BY trip_day
""").toPandas()

# Plotting the data
plt.figure(figsize=(10, 6))
plt.plot(trip_counts_per_day["trip_day"], trip_counts_per_day["trip_count"], marker="o", linestyle="-")
plt.title("Trip Counts Per Day")
plt.xlabel("Date")
plt.ylabel("Number of Trips")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```



Comments about trends or significant spikes. The graph shows us that trip counts peaked on the 10th and 31st, and were at their lowest on the 26th and 27th. Overall the trip counts were the most consistently high in the middle of the month from the 8th to the 24th.

Location Analysis

40

```
%python
from pyspark.sql.functions import struct, col

# Parsing 'tpep_pickup_datetime' as a timestamp
df_with_locations = df.withColumn("pickup_location",struct(df.pickup_longitude,df.pickup_latitude)).withColumn("dr

df_with_locations = df_with_locations.filter(
    (col("pickup_location.pickup_longitude") != 0) | (col("pickup_location.pickup_latitude") != 0)
).filter(
    (col("dropoff_location.dropoff_longitude") != 0) | (col("dropoff_location.dropoff_latitude") != 0)
)

df_with_locations.createOrReplaceTempView("df_with_locations")
```

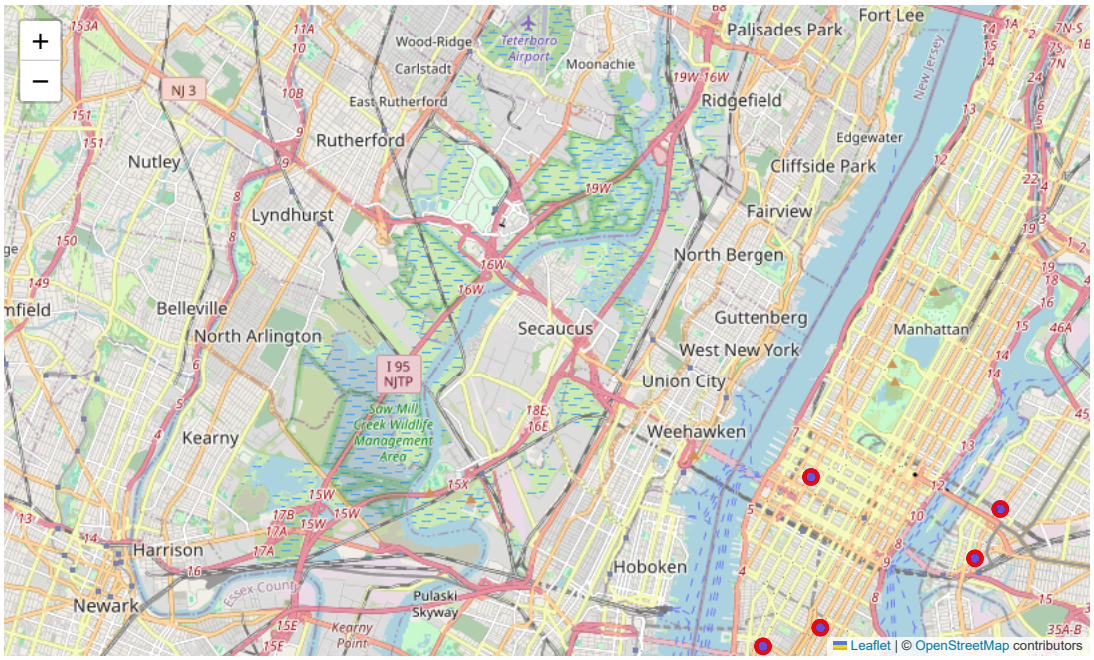
41

```
%python
# Query to get top pickup locations
top_pickup_locations = spark.sql("""
    SELECT pickup_location, COUNT(*) as count
    FROM df_with_locations
    GROUP BY pickup_location
    ORDER BY count DESC
    LIMIT 10
""")

# Query to get top dropoff locations
top_dropoff_locations = spark.sql("""
    SELECT dropoff_location, COUNT(*) as count
    FROM df_with_locations
    GROUP BY dropoff_location
    ORDER BY count DESC
    LIMIT 10
""")

# Displaying tables in SparkSQL
top_pickup_locations.show()
top_dropoff_locations.show()
```

```
|{-73.988456726074...|  146|
|{-73.978279113769...|  121|
|{-73.942085266113...|  108|
|{-73.990852355957...|   96|
+--------------------+-----+

+--------------------+-----+
|    dropoff_location|count|
+--------------------+-----+
|{-73.948638916015...| 1043|
|{-74.186302185058...|  729|
|{-73.986717224121...|  428|
|{-73.915122985839...|  305|
|{-74.003143310546...|  232|
|{-73.921516418457...|  153|
|{-73.988456726074...|  145|
|{-73.978279113769...|  121|
|{-73.942085266113...|  108|
|{-73.990852355957...|   98|
+--------------------+-----+
```

42

```
%python
import folium
from pyspark.sql import functions as F

# Calculate mean latitude and longitude for centering the map
mean_coords = df_with_locations.select(
    F.avg("pickup_location.pickup_latitude").alias("mean_latitude"),
    F.avg("pickup_location.pickup_longitude").alias("mean_longitude")
).collect()[0]

mean_latitude = mean_coords["mean_latitude"]
mean_longitude = mean_coords["mean_longitude"]

# Get top pickup and dropoff locations in PySpark
top_pickup_locations_list = top_pickup_locations.collect()
top_dropoff_locations_list = top_dropoff_locations.collect()

# Initialize the map centered on mean coordinates
nyc_map = folium.Map(location=[mean_latitude, mean_longitude], zoom_start=12)

# Plot top pickup locations in blue
for location in top_pickup_locations_list:
    pickup_location = location['pickup_location']
    folium.CircleMarker(
        location=[pickup_location[1], pickup_location[0]],  # (latitude, longitude)
        radius=5,
        color='blue',
        fill=True,
        fill_color='blue',
        fill_opacity=0.6,
        popup=f"Pickup Count: {location['count']}"
    ).add_to(nyc_map)

# Plot top dropoff locations in red
for location in top_dropoff_locations_list:
    dropoff_location = location['dropoff_location']
    folium.CircleMarker(
        location=[dropoff_location[1], dropoff_location[0]],  # (latitude, longitude)
        radius=5,
        color='red',
        fill=False,
        fill_opacity=0.6,
        popup=f"Dropoff Count: {location['count']}"
    ).add_to(nyc_map)

# Display the map
nyc_map.save("top_pickup_dropoff_locations.html")
nyc_map
```



The busiest pick up and dropoff locations were busy city streets in Manhatten and Long Island, and in front of Newark Liberty International Airport.

Fare Amount Distribution Analysis

45

```
%python
import pandas as pd
import numpy as np

# Get summary statistics for fare amounts
df_fares = df.select('fare_amount').toPandas()
df_fares['fare_amount'] = pd.to_numeric(df_fares['fare_amount'], errors='coerce')
df_fares = df_fares[df_fares['fare_amount'] > 0]

# Apply log transformation
df_fares['log_fare_amount'] = np.log(df_fares['fare_amount'])

print(df_fares['fare_amount'].describe().round(2))
```

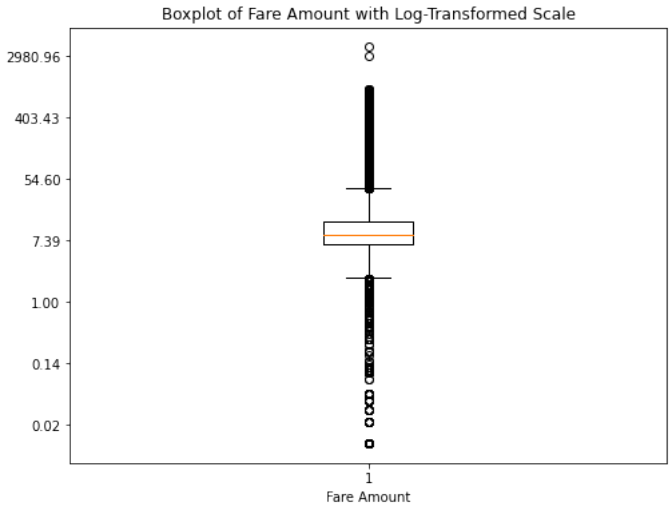
```
count    12741320.00
mean           11.92
std            10.29
```

```
min             0.01
25%             6.50
50%             9.00
75%            13.50
max          4008.00
Name: fare_amount, dtype: float64
```

46

```
%python
import pandas as pd
import matplotlib.pyplot as plt

# Plotting
plt.figure(figsize=(8, 6))
plt.boxplot(df_fares['log_fare_amount'], vert=True)
plt.xlabel("Fare Amount")
plt.title("Boxplot of Fare Amount with Log-Transformed Scale")
ticks = plt.gca().get_yticks()
plt.gca().set_yticklabels([f"{np.exp(tick):.2f}" for tick in ticks])

plt.show()
```

```
<command-3006265739253118>:10: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels([f"{np.exp(tick):.2f}" for tick in ticks])
```



The fare amounts have a lot of outliers, both very small and very large fares (as high as 3000, and as low as 0.02), however the mean is 11.92.

Distance vs Duration Analysis

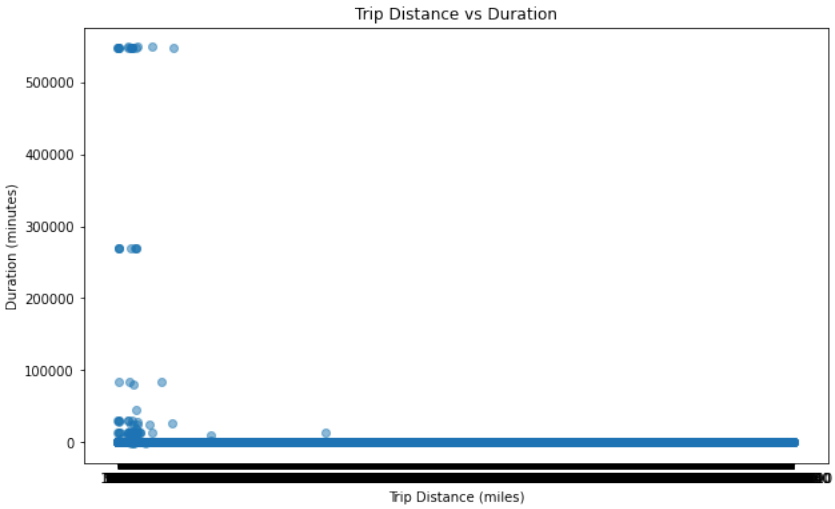
49

```
%python
# Convert `pickup_datetime` and `dropoff_datetime` to timestamp and calculate duration in minutes
df = df.withColumn("pickup_datetime", F.to_timestamp("tpep_pickup_datetime")) \
       .withColumn("dropoff_datetime", F.to_timestamp("tpep_dropoff_datetime")) \
       .withColumn("duration", (F.unix_timestamp("dropoff_datetime") - F.unix_timestamp("pickup_datetime")) / 60)

# Create the SQL view with the updated DataFrame
df.createOrReplaceTempView("tripdata")

# Define distance ranges and calculate average trip duration for each range
distance_ranges = [(0, 1), (1, 3), (3, 5), (5, 10), (10, 15)]

# Initialize an empty list to store results
avg_durations = []

# Loop through each distance range and calculate average duration
for start, end in distance_ranges:
    query = f"""
        SELECT '{start}-{end} miles' AS Distance_Range, AVG(duration) AS Average_Duration
        FROM tripdata
        WHERE trip_distance >= {start} AND trip_distance < {end}
    """
    result = spark.sql(query)
    avg_durations.append(result)

# Combine all results into a single DataFrame
avg_durations_df = avg_durations[0]
for i in range(1, len(avg_durations)):
    avg_durations_df = avg_durations_df.union(avg_durations[i])

# Show the final average durations DataFrame
avg_durations_df.show()
```

```
+--------------+------------------+
|Distance_Range|  Average_Duration|
+--------------+------------------+
|     0-1 miles| 6.509705661030339|
|     1-3 miles|12.708518374428804|
|     3-5 miles| 19.68677075284039|
|    5-10 miles| 24.38561504120697|
|   10-15 miles| 32.29517188901594|
+--------------+------------------+
```

50

```
%python
import matplotlib.pyplot as plt

# Collect trip distance and duration data
distance_duration_df = df.select("trip_distance", "duration").toPandas()

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(distance_duration_df['trip_distance'], distance_duration_df['duration'], alpha=0.5)
plt.title('Trip Distance vs Duration')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Duration (minutes)')
plt.show()
```



The table shows that trip duration increases as trip distance increases.

Daily Trend Analysis

53

```
-- Count daily trips and visualize the trend
SELECT
    DATE(tpep_pickup_datetime) AS trip_date,
    COUNT(*) AS trip_count
FROM tripdata
GROUP BY DATE(tpep_pickup_datetime)
ORDER BY trip_date;
```

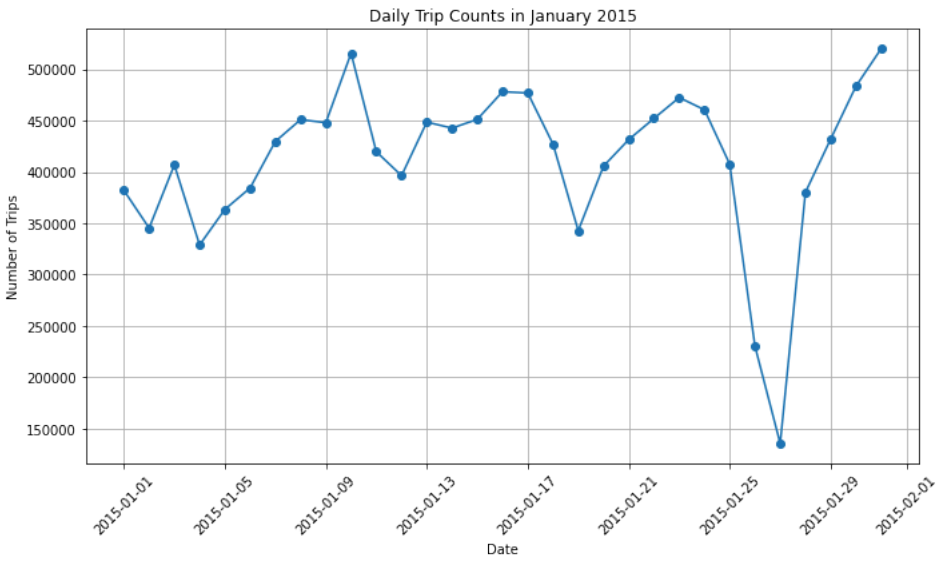
Table

	trip_date	trip_count
1	2015-01-01	382014
2	2015-01-02	345296
3	2015-01-03	406769
4	2015-01-04	328848
5	2015-01-05	363454
6	2015-01-06	384324
7	2015-01-07	429653
8	2015-01-08	450920
9	2015-01-09	447947
10	2015-01-10	515540
11	2015-01-11	419629
12	2015-01-12	396367
13	2015-01-13	448517
14	2015-01-14	442656
15	2015-01-15	451186

31 rows

54

```
%python
#Visualize the trend with a line chart
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'trip_counts_per_day' is the result of the above SQL query
trip_counts_per_day = spark.sql("""
    SELECT DATE(tpep_pickup_datetime) AS trip_date, COUNT(*) AS trip_count
    FROM tripdata
    GROUP BY DATE(tpep_pickup_datetime)
    ORDER BY trip_date
""").toPandas()

plt.figure(figsize=(10, 6))
plt.plot(trip_counts_per_day["trip_date"], trip_counts_per_day["trip_count"], marker="o")
plt.title("Daily Trip Counts in January 2015")
plt.xlabel("Date")
plt.ylabel("Number of Trips")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Day over day the trips fluctuate from increasing and decreasing. By looking up the days of the week, we can figure out that at the start of every week, the trips decrease, and increase as it progresses towards Saturday, then dips down again on Sunday/Monday, repeating this pattern at different magnitudes.

Time of Day Impact on Passenger Count

57

```
-- Group trips by hour and calculate the average passenger count
SELECT
    HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS hour_of_day,
    AVG(CAST(passenger_count AS INT)) AS avg_passenger_count
FROM tripdata
WHERE CAST(passenger_count AS INT) > 0
GROUP BY HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP))
ORDER BY hour_of_day;
```

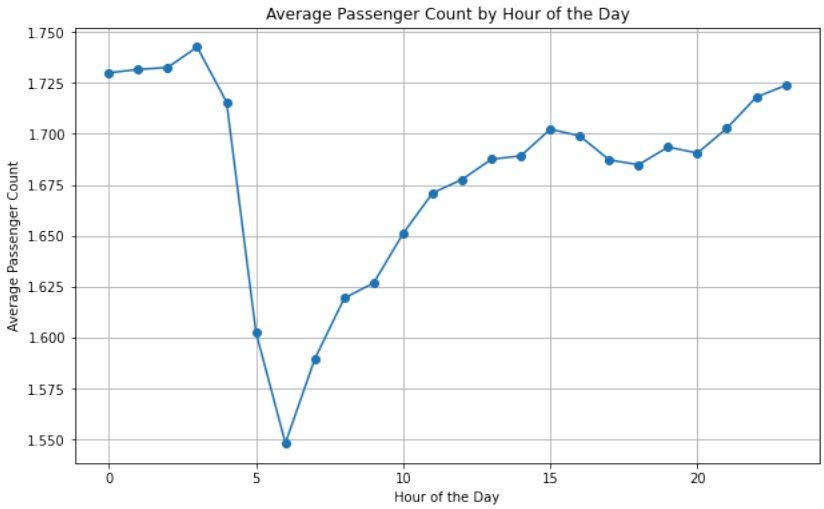
Table

	hour_of_day	avg_passenger_count
1	0	1.729930767855166
2	1	1.7315637625871307
3	2	1.7325478985826348
4	3	1.7425507997398892
5	4	1.7155662058028323
6	5	1.6024959356617214
7	6	1.5481695747956357
8	7	1.5895729960471534
9	8	1.6193137296804434
10	9	1.626799746121103
11	10	1.6510759678777691
12	11	1.6709194926127624
13	12	1.6775537237816018
14	13	1.687489768419992
15	14	1.6891511231851597

24 rows

58

```
%python
# Visualization of average passenger counts by hour
import matplotlib.pyplot as plt

# Assuming 'passenger_count_by_hour' is the result of the above SQL query
passenger_count_by_hour = spark.sql("""
    SELECT HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS hour_of_day,
    AVG(CAST(passenger_count AS INT)) AS avg_passenger_count
    FROM tripdata
    WHERE CAST(passenger_count AS INT) > 0
    GROUP BY HOUR(CAST(tpep_pickup_datetime AS TIMESTAMP))
    ORDER BY hour_of_day
""").toPandas()

plt.figure(figsize=(10, 6))
plt.plot(passenger_count_by_hour['hour_of_day'], passenger_count_by_hour['avg_passenger_count'], marker="o")
plt.title("Average Passenger Count by Hour of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Average Passenger Count")
plt.grid(True)
plt.show()
```



From the graph we can see that in the early hours of the morning until 4 am (where passenger count peaks), there is a high passenger count, which then drops towards 6 am (lowest count), then progressively increases throughout the day towards midnight.

Revenue by Day of the Week Analysis

61

```
-- Calculate total revenue by day of the week
SELECT
    DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS day_of_week,
    SUM(CAST(fare_amount AS DOUBLE)) AS total_revenue
FROM tripdata
WHERE CAST(fare_amount AS DOUBLE) > 0
GROUP BY DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP))
ORDER BY total_revenue DESC;
```

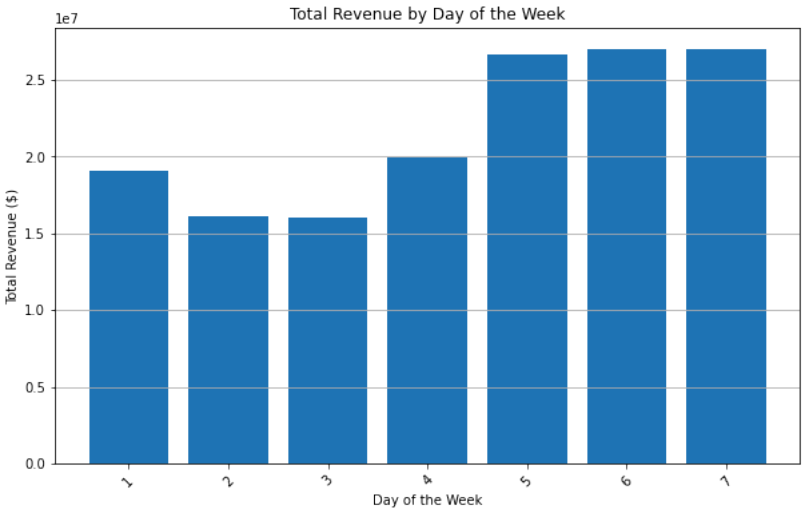
Table

	day_of_week	total_revenue
1	7	27035775.660000026
2	6	26965601.779999297
3	5	26610790.539999273
4	4	19982287.89000001
5	1	19082627.280000012
6	2	16105339.730000006
7	3	16040581.59000001

7 rows

62

```
%python
# Bar chart comparing revenue by day of the week
import matplotlib.pyplot as plt

# Assuming 'revenue_by_day' is the result of the above SQL query
revenue_by_day = spark.sql("""
    SELECT DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP)) AS day_of_week,
    SUM(CAST(fare_amount AS DOUBLE)) AS total_revenue
    FROM tripdata
    WHERE CAST(fare_amount AS DOUBLE) > 0
    GROUP BY DAYOFWEEK(CAST(tpep_pickup_datetime AS TIMESTAMP))
    ORDER BY total_revenue DESC
""").toPandas()

plt.figure(figsize=(10, 6))
plt.bar(revenue_by_day['day_of_week'], revenue_by_day['total_revenue'])
plt.title("Total Revenue by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Total Revenue ($)")
plt.xticks(rotation=45)
plt.grid(True, axis='y')
plt.show()
```



From the table and graph we can see that the revenue is lowest in the early days of the week (Sun to Wed, where people would be the most busy), and is highest near the end, likely due to Friday nights and the weekend being off time for most people.

Final report summarizing the analyses

This report explores key insights from analyzing NYC taxi data in January 2015. The analysis highlights fare patterns, trip durations, peak times, and popular locations.

Outliers and Fare Analysis: We first identified trips with extreme fare values—either very high fares or those with zero or negative values, likely due to rare events or data inconsistencies. Interestingly, we found a minimal correlation between fare and trip distance, suggesting that fares are influenced more by base charges and surcharges than distance alone.

Trip Duration and Passenger Counts: Average trip durations were consistent across different passenger groups, hovering around 13-14 minutes. Trips with more passengers, however, tended to have shorter durations, possibly due to route optimizations or ride-sharing trends.

Distance and Fare Clustering: As expected, fare amounts generally increased with distance. Short trips under 1 mile averaged around $5.34, while those exceeding 5 miles averaged over $31, illustrating a gradual fare increase relative to distance.

Trip Frequency by Location: Popular pickup and drop-off spots included Manhattan, Long Island, and Newark Airport. These areas align with high-density travel points, showing where taxi demand is most concentrated within the city.

Daily and Hourly Travel Trends: Fridays and Saturdays had the highest trip counts and revenue, reflecting weekend travel demand. Trips typically peaked around 7 PM on weekdays, matching evening commute times, with lower trip counts seen on early weekday mornings.

Revenue by Payment and Day: Card payments were most common, with an average fare around $12.50, slightly higher than cash fares. In terms of weekly revenue, Fridays and Saturdays generated the most, while midweek days showed lower totals.

Trip Duration and Distance Relationship: Longer trips took proportionally more time, with short trips under 1 mile averaging 6.5 minutes and longer trips of 10-15 miles taking around 32 minutes. This linear relationship aligns with expected travel times across NYC.

Popular Travel Times: Analysis of trip frequency and duration by hour showed peak travel around 5-10 PM, with the highest average passenger counts observed during early morning hours and tapering off around mid-day.

Overall, these insights reflect how NYC's taxi system adapts to diverse travel demands, varying across time, location, and passenger needs.