Uber Supply-Demand Gap Analysis -

Project Type - Exploratory Data Analysis(EDA)

Contribution - Individual

Team Member 1 - Himanshu Arya

Project Summary -

This project analyzes Uber request data to identify and explain the mismatch between ride demand and supply. It uses real-world data to uncover time-based and location-based issues such as high cancellation rate, "No Cars Available spikes, and low trip fulfillment in key time slots.

I used Python for EDA. I used Pandas for analyzing the data and Matplotlib & Seaborn for the visualization.

The findings can help Uber improve driver allocation, reduce cancellation, and enhance customer satisfaction.

GitHub Link -

https://github.com/HiAr21/Uber_Supply-Demand_Gap_AnalysisProvide

Problem Statement

In many urban regions, Uber experiences frequent demand-supply mismatches, leading to poor user experience such as no cars available or high cancellation, especially during peak hours. Aim to identify:

- When and where demand is high
- When and where supply fails
- Which combination of time and pickup point are most problematic

Define Your Business Objective?

The objective is to perform a detailed EDA to:

- Identify periods with peak demand and low supply
- Quantify supply shortfall using trip completion data
- Provide actionable insights to reduce failed bookings
- Recommend data-driven solutions to improve Uber's operational efficiencyAnswer Here.

General Guidelines : -

↳ 1 cell hidden

Let's Begin !

1. Know Your Data

Import Libraries

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

Dataset Loading

```
from google.colab import files
uploaded = files.upload()
```

Choose Files uber_data_eda.csv

- **uber_data_eda.csv**(text/csv) - 710772 bytes, last modified: 6/21/2025 - 100% done

```
Saving uber_data_eda.csv to uber_data_eda.csv
```

```
# Load Dataset
df = pd.read_csv("uber_data_eda.csv")
```

Dataset First View

```
# Dataset First Look
df.head()
```

	Request id	Pickup point	Driver id	Status	Request Date & Time	Drop Date & Time	Request Date	Request Time	Drop Date	Drop Time	Request Hour	Time Slot	Trip Completed
0	619	Airport	1.0	Trip Completed	07/11/2016 11:51:00	07/11/2016 13:00:00	11/7/2016	11:51:00	11/7/2016	13:00:00	11	Day	Yes
1	867	Airport	1.0	Trip Completed	07/11/2016 17:57:00	07/11/2016 18:47:00	11/7/2016	17:57:00	11/7/2016	18:47:00	17	Evening	Yes
2	1807	City	1.0	Trip Completed	07/12/2016 09:17:00	07/12/2016 09:58:00	12/7/2016	9:17:00	12/7/2016	9:58:00	9	Morning	Yes
3	2532	Airport	1.0	Trip Completed	07/12/2016 21:08:00	07/12/2016 22:03:00	12/7/2016	21:08:00	12/7/2016	22:03:00	21	Evening	Yes
4	3112	City	1.0	Trip Completed	13/07/2016 08:33:16	13/07/2016 09:25:47	13/07/2016	8:33:16	13/07/2016	9:25:47	8	Morning	Yes

Next steps: Generate code with df | View recommended plots | New interactive sheet

Dataset Rows & Columns count

```
# Dataset Rows & Columns count
(no_of_row,no_of_col)=df.shape
print(f"Number of Rows : {no_of_row}")
print(f"Number of Columns : {no_of_col}")
```

```
Number of Rows : 6745
Number of Columns : 13
```

Dataset Information

```
# Dataset Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6745 entries, 0 to 6744
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Request id          6745 non-null   int64
 1   Pickup point        6745 non-null   object
 2   Driver id           4095 non-null   float64
 3   Status              6745 non-null   object
 4   Request Date & Time 6745 non-null   object
 5   Drop Date & Time    2831 non-null   object
 6   Request Date        6745 non-null   object
 7   Request Time        6745 non-null   object
 8   Drop Date           2831 non-null   object
 9   Drop Time           2831 non-null   object
 10  Request Hour        6745 non-null   int64
 11  Time Slot           6745 non-null   object
 12  Trip Completed      6745 non-null   object
dtypes: float64(1), int64(2), object(10)
memory usage: 685.2+ KB
```

Duplicate Values

```
# Dataset Duplicate Value Count
df.duplicated().sum()
```

```
np.int64(0)
```

Missing Values/Null Values

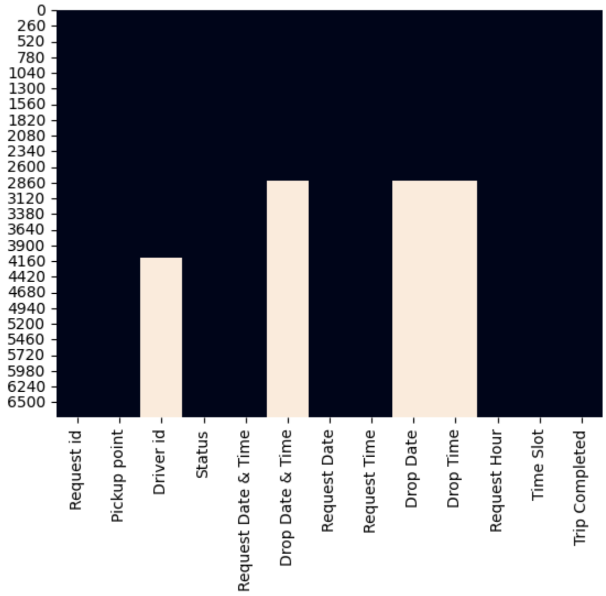
```
# Missing Values/Null Values Count
df.isnull().sum()
```

	0
Request id	0
Pickup point	0
Driver id	2650
Status	0
Request Date & Time	0
Drop Date & Time	3914
Request Date	0
Request Time	0
Drop Date	3914
Drop Time	3914
Request Hour	0
Time Slot	0
Trip Completed	0

dtype: int64

```
# Visualizing the missing values
sns.heatmap(df.isnull(),cbar=False)
```

```
<Axes: >
```



What did you know about your dataset?

The dataset contains detailed Uber ride request logs collected over a few days. Each row represents a unique ride request and includes:

- Request & Drop timestamps
- Pickup Point (either City or Airport)
- Driver ID (if a driver was assigned)
- Request Status — either:
    - Trip Completed
    - Cancelled
    - No Cars Available

The dataset also includes additional derived fields such as:

- Request Hour and Time Slot (Morning, Day, Evening, Late Night)
- A flag indicating whether the trip was completed or not

2. Understanding Your Variables

```
# Dataset Columns
df.columns
```

```
Index(['Request id', 'Pickup point', 'Driver id', 'Status',
       'Request Date & Time', 'Drop Date & Time', 'Request Date',
       'Request Time', 'Drop Date', 'Drop Time', 'Request Hour', 'Time Slot',
       'Trip Completed'],
      dtype='object')
```

```
# Dataset Describe
df.describe()
```

	Request id	Driver id	Request Hour
count	6745.000000	4095.000000	6745.000000
mean	3384.644922	149.501343	12.956709
std	1955.099667	86.051994	6.504052
min	1.000000	1.000000	0.000000
25%	1691.000000	75.000000	7.000000
50%	3387.000000	149.000000	13.000000
75%	5080.000000	224.000000	19.000000
max	6766.000000	300.000000	23.000000

Check Unique Values for each variable.

```
df['Request Date'].unique()
```

```
array(['11/7/2016', '12/7/2016', '13/07/2016', '14/07/2016', '15/07/2016'],
      dtype=object)
```

```
df.nunique()
```

	0
Request id	6745
Pickup point	2
Driver id	300
Status	3
Request Date & Time	5618
Drop Date & Time	2598
Request Date	5
Request Time	4955
Drop Date	6
Drop Time	2393
Request Hour	24
Time Slot	4
Trip Completed	2

dtype: int64

```
# Check Unique Values for each variable.
df['Status'].value_counts()
```

	count
Status	
Trip Completed	2831
No Cars Available	2650
Cancelled	1264

dtype: int64

3. ***Data Wrangling***

Data Wrangling Code

```
# Convert datatype of date&time to datetime
df['Request Date & Time'] = pd.to_datetime(df['Request Date & Time'],format='%d/%m/%Y %H:%M:%S')
df['Drop Date & Time'] = pd.to_datetime(df['Drop Date & Time'],format='%d/%m/%Y %H:%M:%S')

df['Request Date'] = pd.to_datetime(df['Request Date'],format='%d/%m/%Y')
df['Drop Date'] = df['Drop Date & Time'].dt.date
df['Drop Date'] = pd.to_datetime(df['Drop Date'],format='%d/%m/%Y')
```

```
df['Request Time'] = pd.to_datetime(df['Request Time'],format='%H:%M:%S')
```

```
df['Drop Time'] = df['Drop Date & Time'].dt.time
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6745 entries, 0 to 6744
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Request id           6745 non-null   int64
 1   Pickup point         6745 non-null   object
 2   Driver id            4095 non-null   float64
 3   Status               6745 non-null   object
 4   Request Date & Time  6745 non-null   datetime64[ns]
 5   Drop Date & Time     2831 non-null   datetime64[ns]
 6   Request Date         6745 non-null   datetime64[ns]
 7   Request Time         6745 non-null   datetime64[ns]
 8   Drop Date            2831 non-null   datetime64[ns]
 9   Drop Time            2831 non-null   object
 10  Request Hour         6745 non-null   int64
 11  Time Slot            6745 non-null   object
 12  Trip Completed       6745 non-null   object
dtypes: datetime64[ns](5), float64(1), int64(2), object(5)
memory usage: 685.2+ KB
```

```
# Create Gap Score of Trips completed and Total requests (demand - supply)
df['Trip Completed'] = df['Status'] == 'Trip Completed'

gap_df = df.groupby(['Time Slot', 'Pickup point'])['Trip Completed'].agg(['count', 'sum']).reset_index()
gap_df['Gap_Score'] = gap_df['count'] - gap_df['sum']
gap_df.rename(columns={'count': 'Total_Requests', 'sum': 'Trips_Completed'}, inplace=True)

gap_df['Trip_Completed(%)'] = gap_df['Trips_Completed']/gap_df['Total_Requests']*100

gap_df.sort_values(by='Trip_Completed(%)', ascending=False)

gap_df
```

	Time Slot	Pickup point	Total_Requests	Trips_Completed	Gap_Score	Trip_Completed(%)
0	Day	Airport	478	327	151	68.410042
1	Day	City	746	395	351	52.949062
2	Evening	Airport	2081	515	1566	24.747717
3	Evening	City	759	526	233	69.301713
4	Late Night	Airport	253	103	150	40.711462
5	Late Night	City	325	111	214	34.153846
6	Morning	Airport	426	382	44	89.671362
7	Morning	City	1677	472	1205	28.145498

Next steps: Generate code with gap_df View recommended plots New interactive sheet

```
#chart-4 Status Proportion by Pickup Point
pickup_status = df.groupby(['Pickup point', 'Status']).size().reset_index(name='count')
pickup_total = df.groupby('Pickup point').size().reset_index(name='total')
pickup_status = pickup_status.merge(pickup_total, on='Pickup point')
pickup_status['percent'] = (pickup_status['count'] / pickup_status['total']) * 100

pickup_status
```

	Pickup point	Status	count	total	percent
0	Airport	Cancelled	198	3238	6.114886
1	Airport	No Cars Available	1713	3238	52.903027
2	Airport	Trip Completed	1327	3238	40.982088
3	City	Cancelled	1066	3507	30.396350
4	City	No Cars Available	937	3507	26.717993
5	City	Trip Completed	1504	3507	42.885657

Next steps: Generate code with pickup_status View recommended plots New interactive sheet

```
#chart-5 Heatmap: Hour vs Status

heat_data = df.groupby(['Request Hour', 'Status']).size().unstack().fillna(0)
heat_data
```

Status	Cancelled	No Cars Available	Trip Completed
Request Hour			
0	3	56	40
1	4	56	25
2	5	57	37
3	2	56	34
4	51	74	78
5	176	84	185
6	145	86	167
7	169	63	174
8	178	90	155
9	175	83	173
10	62	65	116
11	15	41	115
12	19	44	121
13	18	53	89
14	11	37	88
15	21	48	102
16	22	46	91
17	35	232	151
18	24	322	164
19	24	283	166
20	41	290	161
21	42	265	142
22	12	138	154
23	10	81	103

Next steps: Generate code with heat_data View recommended plots New interactive sheet

```
#chart-6 Trip Duration Distribution

df['Trip Duration (min)'] = (df[df['Status']=='Trip Completed']['Drop Date & Time'] - df[df['Status']=='Trip Completed']['Request Date & Time']).dt.total_seconds() / 60
df.loc[df['Status'] != 'Trip Completed', 'Trip Duration (min)'] = None

completed_trips = df[df['Status'] == 'Trip Completed']

completed_trips.head()
```

	Request id	Pickup point	Driver id	Status	Request Date & Time	Drop Date & Time	Request Date	Request Time	Drop Date	Drop Time	Request Hour	Time Slot	Trip Completed	Trip Duration (min)
0	619	Airport	1.0	Trip Completed	2016-11-07 11:51:00	2016-11-07 13:00:00	2016-07-11	1900-01-01 11:51:00	2016-11-07	13:00:00	11	Day	True	69.000000
1	867	Airport	1.0	Trip Completed	2016-11-07 17:57:00	2016-11-07 18:47:00	2016-07-11	1900-01-01 17:57:00	2016-11-07	18:47:00	17	Evening	True	50.000000
2	1807	City	1.0	Trip Completed	2016-12-07 09:17:00	2016-12-07 09:58:00	2016-07-12	1900-01-01 09:17:00	2016-12-07	09:58:00	9	Morning	True	41.000000
3	2532	Airport	1.0	Trip Completed	2016-12-07 21:08:00	2016-12-07 22:03:00	2016-07-12	1900-01-01 21:08:00	2016-12-07	22:03:00	21	Evening	True	55.000000
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47	2016-07-13	1900-01-01 08:33:16	2016-07-13	09:25:47	8	Morning	True	52.516667

Next steps: Generate code with `completed_trips` View recommended plots New interactive sheet

```
#chart-7 % of No Cars/Cancellations per Time Slot

slot_status = df.groupby(['Time Slot', 'Status']).size().reset_index(name='count')
slot_total = df.groupby('Time Slot').size().reset_index(name='total')
slot_status = slot_status.merge(slot_total, on='Time Slot')
slot_status['percent'] = (slot_status['count'] / slot_status['total']) * 100

slot_status
```

	Time Slot	Status	count	total	percent
0	Day	Cancelled	168	1224	13.725490
1	Day	No Cars Available	334	1224	27.287582
2	Day	Trip Completed	722	1224	58.986928
3	Evening	Cancelled	188	2840	6.619718
4	Evening	No Cars Available	1611	2840	56.725352
5	Evening	Trip Completed	1041	2840	36.654930
6	Late Night	Cancelled	65	578	11.245675
7	Late Night	No Cars Available	299	578	51.730104
8	Late Night	Trip Completed	214	578	37.024221
9	Morning	Cancelled	843	2103	40.085592
10	Morning	No Cars Available	406	2103	19.305754
11	Morning	Trip Completed	854	2103	40.608654

Next steps: Generate code with `slot_status` View recommended plots New interactive sheet

```
#chart-8 Pickup Point vs Time Slot Heatmap

pt_heat = df.groupby(['Pickup point', 'Time Slot'])['Status'].value_counts().unstack().fillna(0)

pt_heat
```

	Status	Cancelled	No Cars Available	Trip Completed
Pickup point	**Time Slot**			
Airport	**Day**	64	87	327
	Evening	109	1457	515
	Late Night	2	148	103
	Morning	23	21	382
City	**Day**	104	247	395
	Evening	79	154	526
	Late Night	63	151	111
	Morning	820	385	472

Next steps: Generate code with `pt_heat` View recommended plots New interactive sheet

```
#chart-9 Line Plot of Requests Over Time (Daily)

requests_per_day = df.groupby('Request Date').size().reset_index(name='Requests')
requests_per_day
```

	Request Date	Requests
0	2016-07-11	1367
1	2016-07-12	1307
2	2016-07-13	1337
3	2016-07-14	1353
4	2016-07-15	1381

Next steps: Generate code with `requests_per_day` View recommended plots New interactive sheet

What all manipulations have you done and insights you found?

1. Converted Date & Time Columns to datetime format
    - Both Request Date & Time and Drop Date & Time were in mixed formats.

- Standardized them using pd.to_datetime() with day-first parsing to ensure accurate time-based analysis.

2. Created Request Hour and Time Slot columns
    - Request Hour was extracted from the datetime to understand hourly trends.
    - Time Slot categorized the day into Late Night, Morning, Day, and Evening — useful for grouping and peak analysis.
3. Created Trip Completed Flag
    - A binary column to indicate whether the request led to a successful trip (based on Status = "Trip Completed").
4. Computed Gap Score
    - A new metric calculated as: Gap Score = Total Requests - Completed Trips
    - Helps quantify the demand-supply gap in each group (time slot, pickup point).
5. Calculated Trip Duration (for completed trips)
    - Derived from the difference between drop and request timestamps, converted to minutes.
    - Used only where both timestamps exist (i.e., for Trip Completed).

4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

```
df.head()
```

	Request id	Pickup point	Driver id	Status	Request Date & Time	Drop Date & Time	Request Date	Request Time	Drop Date	Drop Time	Request Hour	Time Slot	Trip Completed	Trip Duration (min)
0	619	Airport	1.0	Trip Completed	2016-11-07 11:51:00	2016-11-07 13:00:00	2016-07-11	1900-01-01 11:51:00	2016-11-07	13:00:00	11	Day	True	69.000000
1	867	Airport	1.0	Trip Completed	2016-11-07 17:57:00	2016-11-07 18:47:00	2016-07-11	1900-01-01 17:57:00	2016-11-07	18:47:00	17	Evening	True	50.000000
2	1807	City	1.0	Trip Completed	2016-12-07 09:17:00	2016-12-07 09:58:00	2016-07-12	1900-01-01 09:17:00	2016-12-07	09:58:00	9	Morning	True	41.000000
3	2532	Airport	1.0	Trip Completed	2016-12-07 21:08:00	2016-12-07 22:03:00	2016-07-12	1900-01-01 21:08:00	2016-12-07	22:03:00	21	Evening	True	55.000000
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47	2016-07-13	1900-01-01 08:33:16	2016-07-13	09:25:47	8	Morning	True	52.516667

Next steps: Generate code with df View recommended plots New interactive sheet

Chart - 1

```
#1 Request by Hour and Status
plt.figure(figsize=(12, 6))
sns.countplot(x='Request Hour', hue='Status', data=df)
plt.title('Hourly Requests by Status')
plt.grid()
plt.show()
```

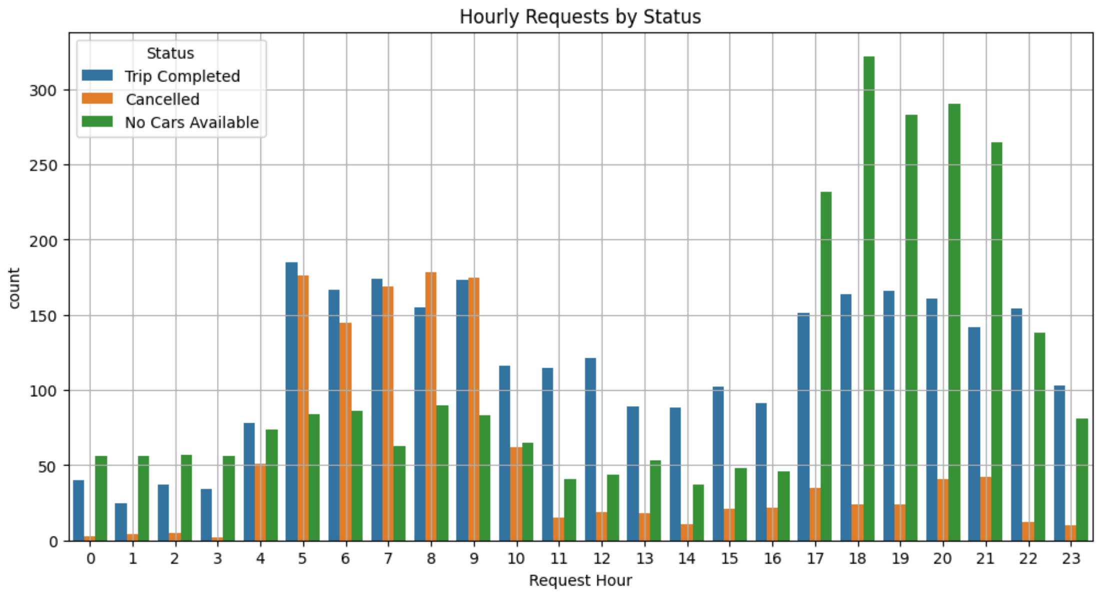


Chart - 2

```
#2 Time Slot vs Status
plt.figure(figsize=(8, 5))
sns.countplot(x='Time Slot', hue='Status', data=df, order=['Late Night', 'Morning', 'Day', 'Evening'])
plt.title('Requests by Time Slot and Status')
plt.show()
```

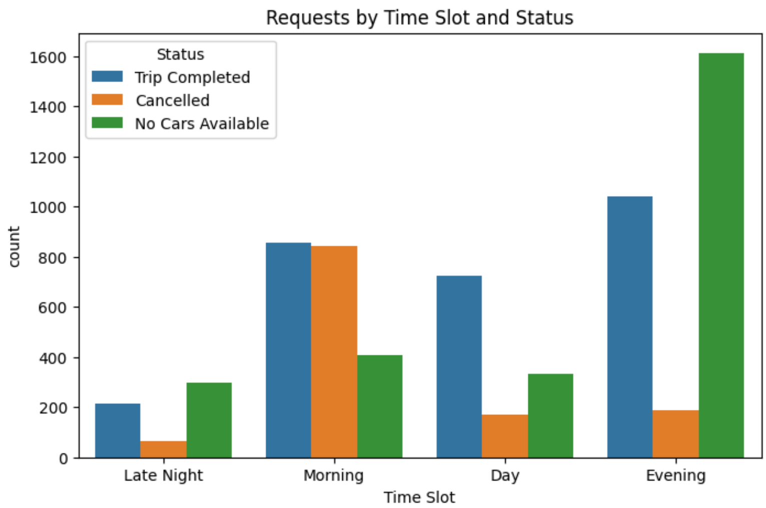


Chart - 3

```
#3 Pickup Point vs Status
plt.figure(figsize=(6,4))
sns.countplot(x='Pickup point', hue='Status', data=df)
plt.title('Request Status by Pickup Point')
plt.show()
```

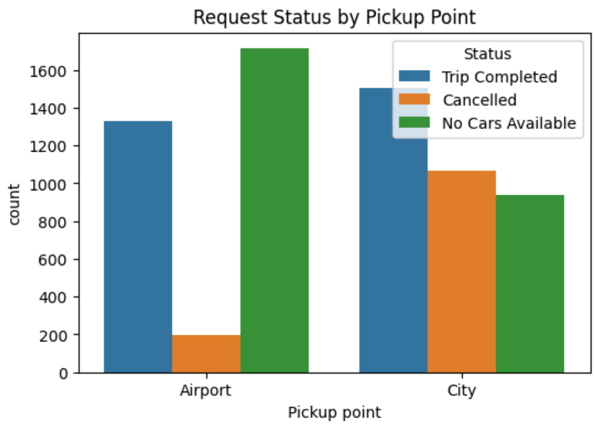
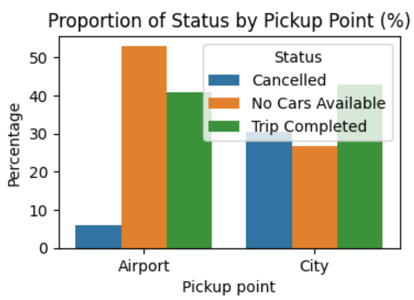


Chart - 4

```
# Status Proportion by Pickup Point
plt.figure(figsize=(4,2.5))
sns.barplot(x='Pickup point', y='percent', hue='Status', data=pickup_status)
plt.title('Proportion of Status by Pickup Point (%)')
plt.ylabel('Percentage')
plt.show()
```



Airport has higher No Cars Available %, City has more Cancellations — both signal supply failure but from different causes.

1. Why did you pick the specific chart?

To compare how ride outcomes (Completed, Cancelled, No Cars) vary between City and Airport pickups. A percentage-based bar chart allows clear proportional comparison.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

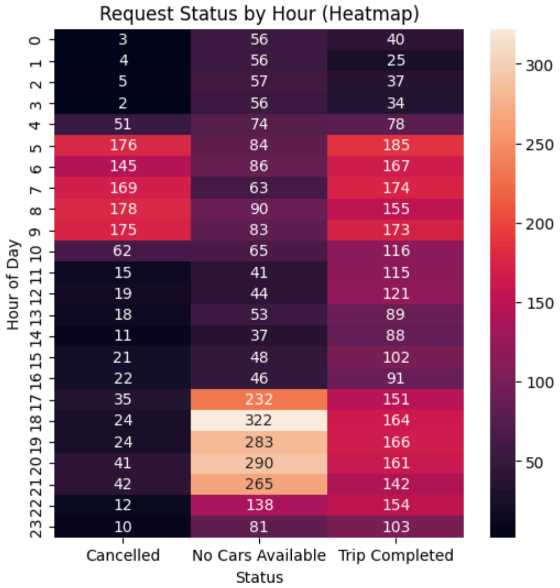
Yes. Helps Uber focus supply expansion at the Airport and work on cancellation reduction in the City through driver incentives or UI improvements.

Yes. Persistent No Cars Available at Airport can push users to competitors or taxis.

Chart - 5

```
# Heatmap: Hour vs Status

plt.figure(figsize=(6,6))
sns.heatmap(heat_data, annot=True, fmt=".0f")
plt.title("Request Status by Hour (Heatmap)")
plt.ylabel("Hour of Day")
plt.xlabel("Status")
plt.show()
```



Shows exactly what status dominates at what hour — e.g., "No Cars Available" spike 5–9 AM.

1. Why did you pick the specific chart?

A heatmap provides a visual intensity map of how status outcomes vary by hour, showing peak problem periods.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes. Time-specific patterns help deploy drivers proactively before peak failure windows.

Yes. If these hours continue to fail, Uber could lose commuter and business traffic.

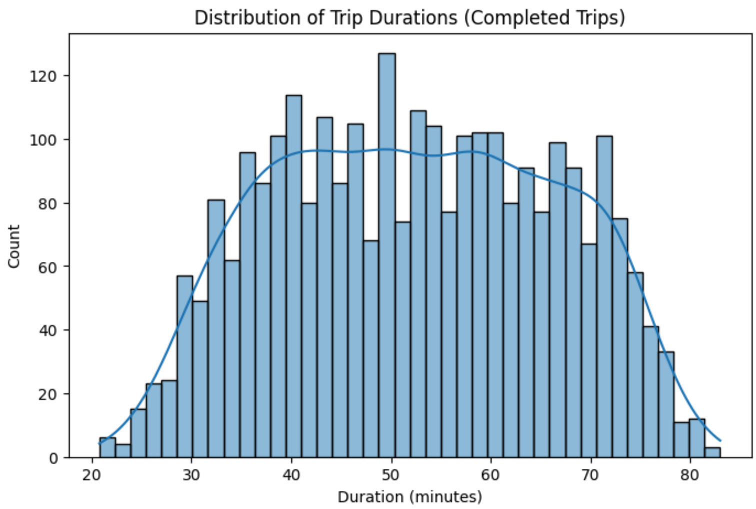
Chart - 6 : Trip Duration Distribution

```
# Check for null or negative values
completed_trips['Trip Duration (min)'].describe()
completed_trips[completed_trips['Trip Duration (min)'] < 0].head()
```

	Request id	Pickup point	Driver id	Status	Request Date & Time	Drop Date & Time	Request Date	Request Time	Drop Date	Drop Time	Request Hour	Time Slot	Trip Completed	Trip Duration (min)
133	2675	Airport	15.0	Trip Completed	2016-12-07 23:43:00	2016-07-13 00:35:12	2016-07-12	1900-01-01 23:43:00	2016-07-13	00:35:12	23	Evening	True	-213067.800000
143	2661	Airport	16.0	Trip Completed	2016-12-07 23:23:00	2016-07-13 00:27:21	2016-07-12	1900-01-01 23:23:00	2016-07-13	00:27:21	23	Evening	True	-213055.650000
245	2667	Airport	25.0	Trip Completed	2016-12-07 23:35:00	2016-07-13 00:40:52	2016-07-12	1900-01-01 23:35:00	2016-07-13	00:40:52	23	Evening	True	-213054.133333
532	2665	Airport	55.0	Trip Completed	2016-12-07 23:30:00	2016-07-13 00:37:17	2016-07-12	1900-01-01 23:30:00	2016-07-13	00:37:17	23	Evening	True	-213052.716667
656	2664	Airport	69.0	Trip Completed	2016-12-07 23:26:00	2016-07-13 00:01:12	2016-07-12	1900-01-01 23:26:00	2016-07-13	00:01:12	23	Evening	True	-213084.800000

```
# Keep only valid durations
filtered = completed_trips[
    (completed_trips['Trip Duration (min)'] > 0) &
    (completed_trips['Trip Duration (min)'] < 120)
]
```

```
#6 Trip Duration Distribution
plt.figure(figsize=(8,5))
sns.histplot(filtered['Trip Duration (min)'], bins=40, kde=True)
plt.title('Distribution of Trip Durations (Completed Trips)')
plt.xlabel('Duration (minutes)')
plt.show()
```



1. Why did you pick the specific chart?

To understand how long successful trips take. A histogram with KDE curve reveals duration spread and potential outliers.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

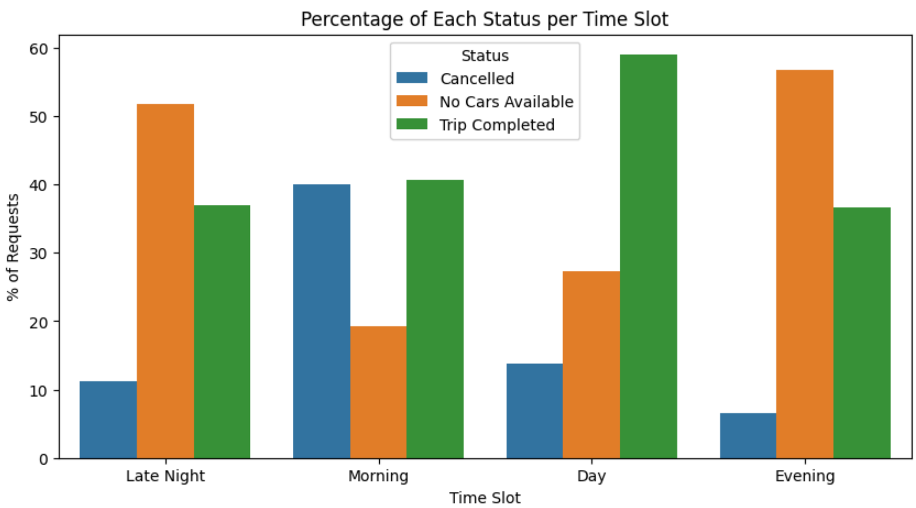
Yes. Helps price short vs long trips better, and target flat fares more effectively.

Outliers may indicate traffic delays or inefficient routing, leading to customer frustration.

Chart - 7 : % of No Cars/Cancellations per Time Slot

```
#7 % of No Cars/Cancellations per Time Slot

plt.figure(figsize=(10,5))
sns.barplot(x='Time Slot', y='percent', hue='Status', data=slot_status, order=['Late Night', 'Morning', 'Day', 'Evening'])
plt.title('Percentage of Each Status per Time Slot')
plt.ylabel('% of Requests')
plt.show()
```



Gives clear % context — e.g., Morning = 55% No Cars Available at Airport.

1. Why did you pick the specific chart?

To identify which time slots have the most unfulfilled demand — a stacked percentage bar chart reveals imbalance quickly.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes. Lets Uber customize solutions per time slot — more drivers in morning, cancellation deterrents in evening.

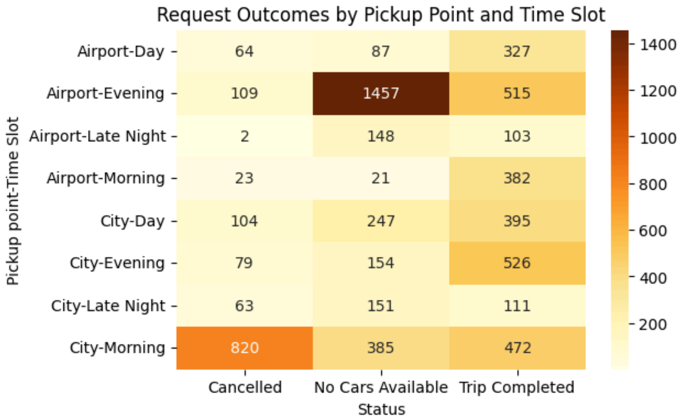
Yes. If Morning No Car rates persist, it can cause long-term user churn during a mission-critical window.

Chart - 8 : Pickup Point vs Time Slot Heatmap

```
#8 Pickup Point vs Time Slot Heatmap

plt.figure(figsize=(6,4))
sns.heatmap(pt_heat, annot=True, fmt=".0f", cmap="YlOrBr")
```

```
plt.title('Request Outcomes by Pickup Point and Time Slot')
plt.show()
```



Shows which pickup+time combos are broken (e.g., Airport+Morning = red zone).

1. Why did you pick the specific chart?

To cross-analyze time + location together, which helps identify specific problem zones (like Airport in Morning).

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

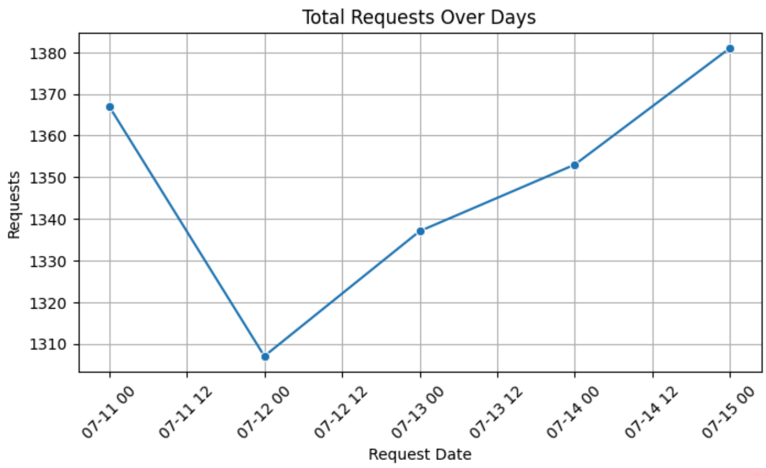
Yes. This level of granularity helps Uber focus supply and outreach surgically, not just broadly.

Airport-Morning users are often time-sensitive (flights). Continued failure here will lead to high-value customer churn.

Chart - 9 : Line Plot of Requests Over Time (Daily)

```
#9 Line Plot of Requests Over Time (Daily)

plt.figure(figsize=(8,4))
sns.lineplot(x='Request Date', y='Requests', data=requests_per_day, marker='o')
plt.title('Total Requests Over Days')
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



1. Why did you pick the specific chart?

To observe daily request patterns and identify anomalies or consistent growth.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes. Confirms Uber can rely on consistent demand and plan driver schedules with confidence.

Not directly — but any future daily dips (e.g., drop after cancellations spike) can be early signs of customer dissatisfaction.

5. Solution to Business Objective

What do you suggest the client to achieve Business Objective ?

Explain Briefly.

1. Increase Driver Availability During Morning Hours : Morning (5–9 AM) shows the highest demand but lowest completion rates, especially at the Airport.

   Recommendation:

   - Offer time-based driver incentives or bonuses during Morning shifts.
   - Use notifications to encourage driver logins before 5 AM, especially around airports.

2. Deploy Targeted Supply at the Airport : Airport pickups consistently suffer from "No Cars Available," especially in the Morning.

   Recommendation:

   - Assign a minimum driver quota to be present near airports during high-demand slots.
   - Create dynamic geofenced incentives for drivers in airport zones.

3. Reduce Evening Cancellations from City : Cancellations are highest in the Evening, mostly from City pickups.