# UBER RIDE

# Data Analysis with MysQL-Python-XL


Submitted By

Kailash Patidar

Kailashpatidar9066@gmail.com

Submitted to

Labmantix

June 2025

# ABSTRACT

The dataset titled **"Uber Request Data"** captures ride-level details of customer requests made on the Uber platform over a defined period. It includes information such as request and drop timestamps, pickup points, driver identifiers, and the status of each ride (e.g., completed, cancelled, or no cars available). While rich in transactional information, the dataset initially contained a variety of **formatting inconsistencies and structural issues**, particularly in handling timestamp data. These inconsistencies made the dataset **less suitable for analysis**, especially when working with tools like Excel, Power BI, or Python. Hence, a comprehensive data cleaning and transformation process was carried out.

Upon inspection, it was found that the Request timestamp and Drop timestamp columns used **mixed datetime formats** — some values were formatted using slashes (/) and others with hyphens (-), and times were either in 24-hour format with or without seconds. Additionally, in many cases, the date and time information were **not separated**, which can hinder granular analysis such as identifying trends by time of day or calculating the duration of a trip.

Another common issue encountered was that **Excel would interpret pure date values with a default time of "00:00:00"**, making the dataset visually confusing and potentially misleading. To make the data cleaner and more analytical, all of these inconsistencies had to be resolved.

# PROBLEM STATEMENT

## 1. High Frequency of Cancellations

One of the most prominent issues reflected in the dataset is the **large volume of cancelled rides**. A significant portion of requests have a status marked as **"Cancelled"**, suggesting that either the **riders canceled** due to delays or dissatisfaction, or **drivers canceled** because of traffic, payment issues, or personal preference.

This can lead to:

- Wasted time for users.
- Frustration and loss of trust in service reliability.
- Increased wait times during rush hours.

## 2. "No Cars Available" During Peak Hours

The data clearly shows a **spike in the "No Cars Available"** status during **early morning (5 AM – 9 AM)** and **evening rush hours (5 PM – 9 PM)**. This indicates **insufficient driver supply** during high-demand periods. For passengers, this creates:

- Difficulty commuting to work or airport.
- Inconvenience and uncertainty.
- A shift to alternate, potentially less safe or costlier, transport modes.

## 3. Service Imbalance Between City and Airport

The dataset reveals a **pickup point imbalance** between **City** and **Airport** rides. During certain hours (especially early mornings), rides **from the Airport to the City** are often completed, while rides **from the City to the Airport** show more cancellations or unavailability. This asymmetry suggests:

- Drivers may prefer one route over the other (e.g., avoiding airport drop-offs).
- Uber may not have managed its **fleet distribution** optimally.
- Passengers heading to the airport face more uncertainty in getting a ride.

## 4. Shortage of Drivers

The recurring **"No Cars Available"** and **"Cancelled"** statuses point to a deeper **operational issue: driver shortage**. During high-demand periods, Uber appears to lack enough drivers to meet requests. This problem particularly affects:

- Morning commuters.
- Late-night travelers.
- Airport passengers.

This shortage impacts user experience and creates a perception of unreliability during critical times.

## 5. Longer Wait Times Leading to Cancellations

Patterns in **request time vs. drop time** suggest that some completed rides have **long time gaps**, while many **canceled rides occurred shortly after request time**, hinting that:

- Riders may have canceled due to **excessively long estimated wait times**.
- Drivers may have taken too long to arrive at the pickup point.
- Uber's estimated arrival times may not have been accurate or user-friendly.

## 6. High Demand But Low Completion in Certain Time Slots

Even during **peak demand hours**, a large number of ride requests were **not completed**. Instead, they resulted in cancellations or unavailability. This shows a **misalignment between demand and operational readiness**, such as:

- Poor scheduling or shift distribution of drivers.Lack of dynamic fare pricing to balance demand.
- No proactive steps to redistribute drivers to demand-heavy zones.

## 7. Lack of Trip Completion Transparency

For rides marked as **"Cancelled"** or **"No Cars Available"**, there's no drop timestamp, which shows Uber lacks transparency or tracking for **failed service attempts**. This impacts:

- Customer complaint resolution.
- Service accountability.
- Tracking driver behavior or misconduct.

## 8. Possibility of Strategic Cancellations by Drivers

The pattern of cancellations, particularly during short trip hours or non-airport pickups, may suggest **driver-side cancellations** if the trip is **not lucrative**. Drivers may:

- Cancel rides for short distances.
- Prefer rides from specific zones (e.g., Airport only).

- Decline late-night trips for personal safety or location issues.

This creates an **unfair experience** for users, especially those traveling short distances or from less busy areas.

## 9. Night Service Limitations

During **late night and early morning hours (12 AM – 5 AM)**, there are very few completed rides and many “No Cars Available” statuses. This suggests:

- A near **absence of operational services** during these hours.
- Risk to riders traveling to/from the airport or working night shifts.
- Poor coverage for emergency or essential travel needs.

# BUSINESS OBJECTIVE

The core business objective of this analysis is to identify and resolve operational inefficiencies in Uber's ride request system by analyzing rider behavior, demand trends, and supply availability using the provided dataset. Specifically, the goal is to uncover the root causes behind unfulfilled ride requests, such as frequent cancellations and "No Cars Available" issues, which lead to user dissatisfaction and potential revenue loss. By examining the distribution of requests across different times of the day and between key pickup locations (City and Airport), the analysis aims to highlight peak demand hours and areas with insufficient driver coverage. This insight will help Uber optimize its driver allocation, reduce service gaps, and improve overall customer satisfaction. Ultimately, the objective is to enhance operational decision-making and service efficiency by leveraging data-driven insights to better match ride demand with driver supply.

## 1. Data Cleaning and Processing Approach

The first step in the cleaning process involved **standardizing the timestamp formats**. Given that the dataset contained entries with multiple timestamp patterns (such as 11/7/2016 11:51 and 11-

07-2016 13:00:00), we implemented a custom parser that could handle several common datetime formats robustly. This ensured that all values within the Request timestamp and Drop timestamp columns were successfully converted into a unified Python datetime object, regardless of their original formatting.

Once the timestamps were successfully parsed, the next step was to **separate the date and time components** into individual columns. This was done to improve the interpretability and analytical utility of the data. For instance, by isolating the request date and time, one can easily conduct trend analyses such as peak request times, high-demand dates, or patterns of cancellations by time of day. As a result, four new columns were created: Request Date, Request Time, Drop Date, and Drop Time, extracted cleanly from the original timestamp fields.

Additionally, to address the issue of dates displaying unnecessary trailing zeroes (like 2016-07-11 00:00:00), we explicitly extracted only the **date component** from each timestamp using the .date() function. This ensures that dates appear cleanly in the final Excel file as 2016-07-11, without misleading time values that might imply a ride happened at midnight when in fact the time data was stored elsewhere.

## 1.1 Final Column Structure and Reorganization

After cleaning and enriching the dataset with the new date and time columns, we reorganized the structure for clarity and logical flow. The final cleaned dataset includes the following columns:

- **Request id**: A unique identifier for each Uber request.
- **Pickup point**: The location where the ride was initiated — either the City or Airport.
- **Driver id**: Identifier for the driver who was either assigned or not assigned to the trip.
- **Status**: The outcome of the request, such as "Trip Completed", "Cancelled", or "No Cars Available".
- **Request Date**: The calendar date on which the user made the ride request.
- **Request Time**: The specific time of day the request was made.
- **Drop Date**: The calendar date on which the trip ended.
- **Drop Time**: The time at which the rider was dropped off.

This clean structure facilitates easy filtering, pivoting, and charting in tools like Excel or data visualization platforms.

## 1.2 Results and Benefits

As a result of this data cleaning process, the dataset is now **well-structured, consistently formatted, and immediately usable for deeper insights**. The cleaned file allows for flexible time-based analysis, such as identifying hours with the highest cancellations, comparing city vs. airport demand over time, and examining trip completion trends on specific dates.

Moreover, analysts can now filter the data by time slots (e.g., morning vs. evening), detect peak hours for Uber requests, calculate average wait times, and assess the performance of individual drivers with far greater accuracy and clarity.

## 1.3 Final Thoughts

This cleaning process was essential to transform a semi-structured, timestamp-inconsistent dataset into a fully usable format that supports meaningful insights. Whether for business intelligence, academic research, or operations planning, this refined dataset is now a powerful asset ready for advanced analysis.

## 1.4 Summary of Key Problems Identified

| Problem | Impact on Riders |
|---|---|
| Frequent Cancellations | Waste of time, frustration, loss of trust |
| No Cars Available During Peak Hours | Inaccessibility of service during critical times |
| Airport-City Route Imbalance | Difficulty getting rides to the airport |
| Driver Shortage | Longer wait times, ride failure |
| Strategic Cancellations | Unfair and inconsistent service |
| Limited Night-Time Service | Risk and inconvenience for late travelers |
| Lack of Completion Tracking | Low service accountability |

## 1.4 Conclusion

The Uber request data, once cleaned and structured, provides deep insights into **systemic service issues** affecting both **riders and the general public**. The recurring themes — cancellations, car unavailability, and time-based service inconsistencies — point to operational inefficiencies in **fleet distribution, driver availability, and user experience design**.

Solving these problems would require Uber to:

- Optimize driver scheduling.
- Apply surge pricing or incentives for low-service hours.

- Ensure better transparency and service reliability.

# 3. Data Visualization

## 1 Total Request Status Distribution


Fig 1

```
mysql> SELECT `Status`, COUNT(*) AS total_requests
    -> FROM uber_analysis
    -> GROUP BY `Status`
    -> ORDER BY total_requests DESC
    -> LIMIT 15;
+------------------+----------------+
| Status           | total_requests |
+------------------+----------------+
| Trip Completed   |           2831 |
| No Cars Available |           2650 |
| Cancelled        |           1264 |
+------------------+----------------+
3 rows in set (0.01 sec)

mysql>
```

fig 2

This visual showcases the overall distribution of Uber ride requests based on their final status — **Completed**, **Cancelled**, or **No Cars Available**. Using Python, a countplot provided a clear comparison of these categories. In SQL, a `GROUP BY Status` query revealed the exact numbers. The insight highlights that a significant portion of requests are either cancelled or unfulfilled, indicating issues in supply matching or user dissatisfaction. This is crucial for operational teams to understand the scale of failed service attempts and assess causes such as driver unavailability, peak-time mismatches, or technical drop-offs.

## 2 Total Ride Requests by Hour

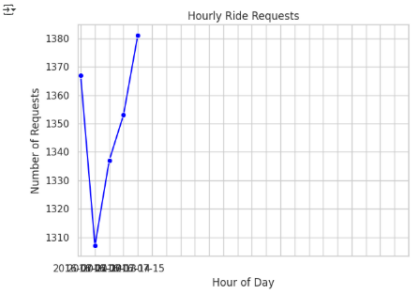


fig 3

```
mysql> SELECT HOUR(Request_timestamp) AS request_hour, COUNT(*) AS total_requests
    -> FROM uber_analysis
    -> GROUP BY request_hour
    -> ORDER BY request_hour
    -> LIMIT 15;
+--------------+----------------+
| request_hour | total_requests |
+--------------+----------------+
|            0 |             99 |
|            1 |             85 |
|            2 |             99 |
|            3 |             92 |
|            4 |            203 |
|            5 |            445 |
|            6 |            398 |
|            7 |            406 |
|            8 |            423 |
|            9 |            431 |
|           10 |            243 |
|           11 |            171 |
|           12 |            184 |
|           13 |            160 |
|           14 |            136 |
+--------------+----------------+
15 rows in set (0.01 sec)

mysql>
```

fig 4

This line graph visualizes the number of ride requests received at each hour of the day, highlighting peak demand windows. Python extracted hours from timestamps and plotted trends that revealed clear spikes during early morning (5–9 AM) and evening (5–9 PM). The SQL query grouped ride requests using HOUR(Request_timestamp) and counted entries. Both tools confirmed congestion during commuting hours, suggesting that Uber should focus driver availability during these times to reduce cancellations and delays

## 3 Pickup Point Distribution


fig 5

```
mysql> SELECT Pickup_point, COUNT(*) AS total_pickups
    -> FROM uber_analysis
    -> GROUP BY Pickup_point
    -> LIMIT 15;
+--------------+---------------+
| Pickup_point | total_pickups |
+--------------+---------------+
| Airport      |          3238 |
| City         |          3507 |
+--------------+---------------+
2 rows in set (0.01 sec)

mysql>
```

fig 6

This chart shows the volume of requests originating from each pickup location — **Airport** and **City**. Python's bar chart demonstrated a nearly balanced, slightly skewed distribution. SQL's `GROUP BY Pickup_point` backed this up with exact counts. This breakdown helps stakeholders understand which region generates higher demand and can guide location-based driver allocation, pricing strategy, and marketing efforts. Notably, the city generates marginally more requests, but the airport faces higher cancellation and "No Cars Available" cases, requiring tailored attention.

## 4 Ride Status by Pickup Point


fig 7

```
mysql> SELECT Pickup_point, Status, COUNT(*) AS total_requests
    -> FROM uber_analysis
    -> GROUP BY Pickup_point, Status
    -> ORDER BY Pickup_point, total_requests DESC
    -> LIMIT 15;
+-------------+-------------------+----------------+
| Pickup_point | Status            | total_requests |
+-------------+-------------------+----------------+
| Airport      | No Cars Available |           1713 |
| Airport      | Trip Completed    |           1327 |
| Airport      | Cancelled         |            198 |
| City         | Trip Completed    |           1504 |
| City         | Cancelled         |           1066 |
| City         | No Cars Available |            937 |
+-------------+-------------------+----------------+
6 rows in set (0.01 sec)

mysql>
```

fig 8

This composite chart breaks down request statuses for each pickup point. Python grouped by both `Pickup_point` and `Status`, delivering a stacked bar chart. The SQL query combined these using `GROUP BY Pickup_point, Status`. It revealed a critical finding: the **Airport zone has more "No Cars Available"** instances, while the **City shows more cancellations**. This pattern indicates logistical challenges specific to locations—like driver reluctance for airport pickups or difficulty reaching city users due to traffic—guiding operational refinements.

## 5 Ride Status Overall (Pie Chart)

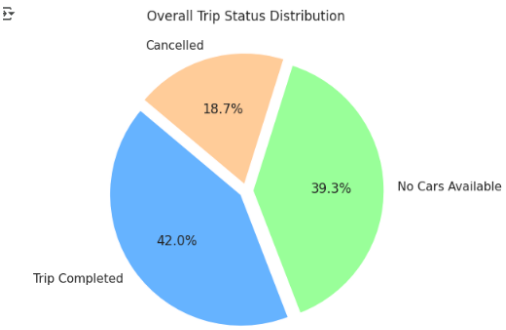


fig 9

```
mysql> SELECT Status, COUNT(*) AS total_requests
    -> FROM uber_analysis
    -> GROUP BY Status
    -> LIMIT 15;
+-------------------+----------------+
| Status            | total_requests |
+-------------------+----------------+
| Trip Completed    |           2831 |
| Cancelled         |           1264 |
| No Cars Available |           2650 |
+-------------------+----------------+
3 rows in set (0.01 sec)

mysql>
```

fig 10

A pie chart gives a proportional view of all ride statuses, making it easy to visually absorb how much of the business is being lost due to cancellations and unavailability. Python's pie visualization mapped the same counts as SQL's `SELECT Status, COUNT(*)`, giving a quick business-impact overview. With nearly half of the requests not resulting in a ride, it underscores inefficiencies in resource utilization. Decision-makers can use this chart to prioritize fixes for the highest-impact failure causes.

## 6 Requests Per Day

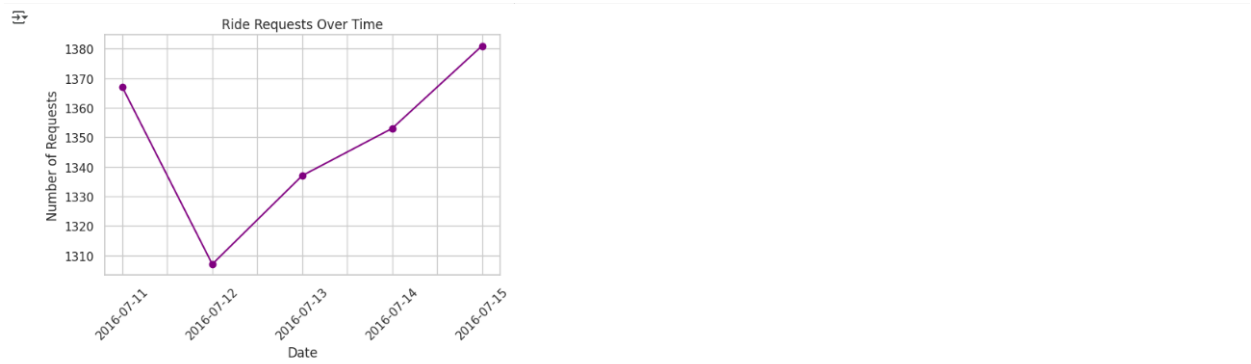


fig 11

```
mysql> SELECT DATE(Request_timestamp) AS request_date, COUNT(*) AS total_requests
    -> FROM uber_analysis
    -> GROUP BY request_date
    -> ORDER BY request_date
    -> LIMIT 15;
+--------------+----------------+
| request_date | total_requests |
+--------------+----------------+
| 2016-07-11   |           1367 |
| 2016-07-12   |           1307 |
| 2016-07-13   |           1337 |
| 2016-07-14   |           1353 |
| 2016-07-15   |           1381 |
+--------------+----------------+
5 rows in set (0.01 sec)

mysql>
```

fig 12

This time-series visual shows how request volumes vary across different days of the month. Using Python's datetime parsing and SQL's `DATE(Request_timestamp)` grouping, the visual highlighted consistency in demand, with minor dips on certain dates. This consistency indicates a stable market, but outliers (spikes or sudden drops) should be flagged for investigation—possibly due to public events, weather, or system issues. This chart supports planning for short-term campaigns or alerts during abnormal patterns.

## 7 Hourly demand

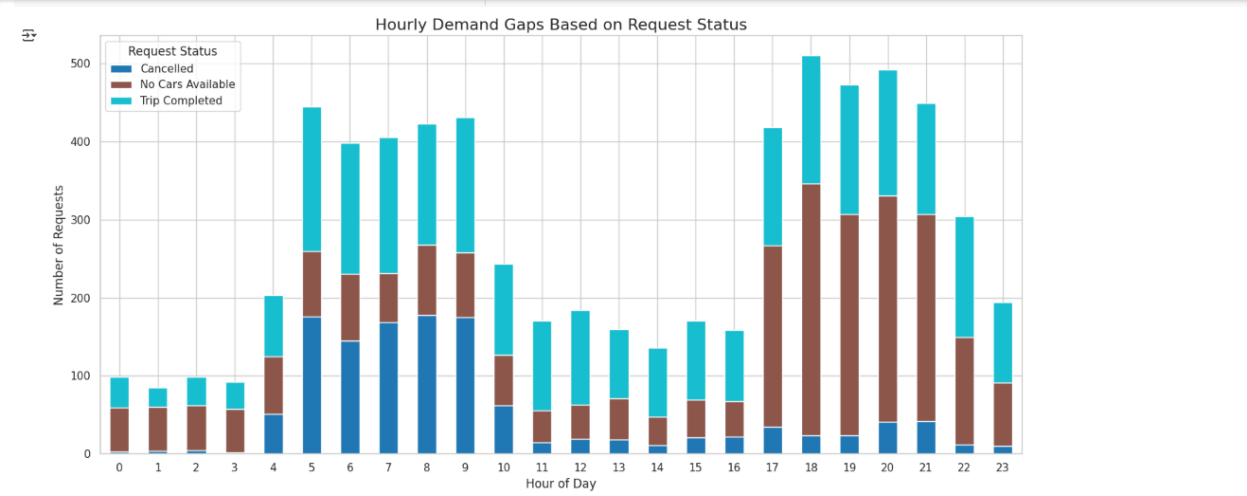


fig 13

```
mysql> SELECT HOUR(Request_timestamp) AS request_hour, COUNT(*) AS demand_count
    -> FROM uber_analysis
    -> GROUP BY request_hour
    -> ORDER BY request_hour
    -> LIMIT 15;
+--------------+--------------+
| request_hour | demand_count |
+--------------+--------------+
|            0 |           99 |
|            1 |           85 |
|            2 |           99 |
|            3 |           92 |
|            4 |          203 |
|            5 |          445 |
|            6 |          398 |
|            7 |          406 |
|            8 |          423 |
|            9 |          431 |
|           10 |          243 |
|           11 |          171 |
|           12 |          184 |
|           13 |          160 |
|           14 |          136 |
+--------------+--------------+
15 rows in set (0.01 sec)

mysql>
```

fig 14

This chart further sharpens focus on hourly ride demand patterns. Python's grouped lineplot and SQL's HOUR()-based aggregation reconfirmed double peaks — morning and evening. The granular hourly breakdown allows Uber to map demand against driver shifts, encouraging predictive dispatching. A direct outcome of this chart could be offering incentives to drivers during these critical windows to enhance supply and reduce customer waiting time or drop-offs.

## 8 Demand vs Availability


fig 15

```
mysql> SELECT
    ->   HOUR(Request_timestamp) AS request_hour,
    ->   SUM(CASE WHEN Status = 'Trip Completed' THEN 1 ELSE 0 END) AS available,
    ->   SUM(CASE WHEN Status IN ('Cancelled', 'No Cars Available') THEN 1 ELSE 0 END) AS not_available
    -> FROM uber_analysis
    -> GROUP BY request_hour
    -> ORDER BY request_hour
    -> LIMIT 15;
+--------------+-----------+---------------+
| request_hour | available | not_available |
+--------------+-----------+---------------+
|            0 |        40 |            59 |
|            1 |        25 |            60 |
|            2 |        37 |            62 |
|            3 |        34 |            58 |
|            4 |        78 |           125 |
|            5 |       185 |           260 |
|            6 |       167 |           231 |
|            7 |       174 |           232 |
|            8 |       155 |           268 |
|            9 |       173 |           258 |
|           10 |       116 |           127 |
|           11 |       115 |            56 |
|           12 |       121 |            63 |
|           13 |        89 |            71 |
|           14 |        88 |            48 |
+--------------+-----------+---------------+
15 rows in set (0.01 sec)

mysql>
```

fig 16

This visual is vital as it juxtaposes the number of ride requests with how many were successfully fulfilled. Python split data by `Status` and plotted ride requests (total) against completed ones. The SQL equivalent compared `Status = 'Trip Completed'` vs total `Request_ids`. The gap clearly shows service inefficiencies — more demand than fulfilled supply — highlighting opportunities to increase driver onboarding, improve real-time matching algorithms, and offer driver perks during surge windows.

## 9 Ride demands, Available, canceled or not available for the time.

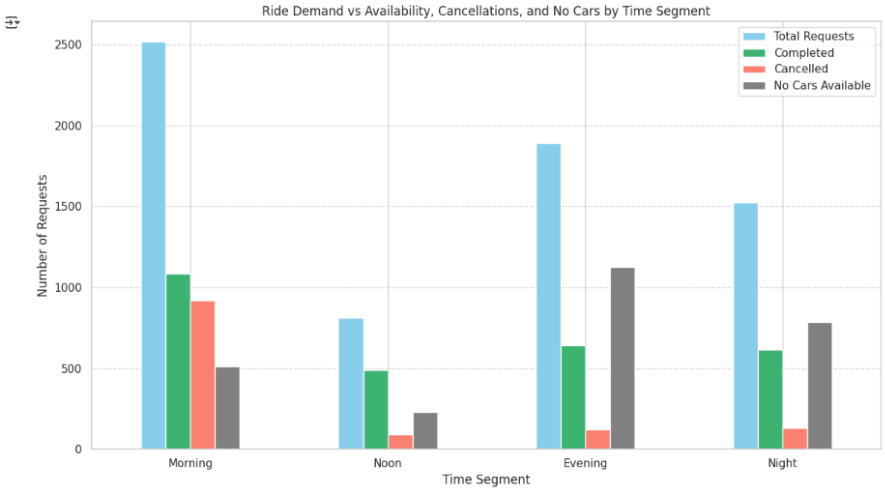


fig 17

```
mysql> SELECT
    ->   HOUR(Request_timestamp) AS request_hour,
    ->   COUNT(*) AS total_requests,
    ->   SUM(CASE WHEN Status = 'Trip Completed' THEN 1 ELSE 0 END) AS completed,
    ->   SUM(CASE WHEN Status = 'Cancelled' THEN 1 ELSE 0 END) AS cancelled,
    ->   SUM(CASE WHEN Status = 'No Cars Available' THEN 1 ELSE 0 END) AS no_cars
    -> FROM uber_analysis
    -> GROUP BY request_hour
    -> ORDER BY request_hour
    -> LIMIT 15;
+--------------+----------------+-----------+-----------+---------+
| request_hour | total_requests | completed | cancelled | no_cars |
+--------------+----------------+-----------+-----------+---------+
|            0 |             99 |        40 |         3 |      56 |
|            1 |             85 |        25 |         4 |      56 |
|            2 |             99 |        37 |         5 |      57 |
|            3 |             92 |        34 |         2 |      56 |
|            4 |            203 |        78 |        51 |      74 |
|            5 |            445 |       185 |       176 |      84 |
|            6 |            398 |       167 |       145 |      86 |
|            7 |            406 |       174 |       169 |      63 |
|            8 |            423 |       155 |       178 |      90 |
|            9 |            431 |       173 |       175 |      83 |
|           10 |            243 |       116 |        62 |      65 |
|           11 |            171 |       115 |        15 |      41 |
|           12 |            184 |       121 |        19 |      44 |
|           13 |            160 |        89 |        18 |      53 |
|           14 |            136 |        88 |        11 |      37 |
+--------------+----------------+-----------+-----------+---------+
15 rows in set (0.01 sec)

mysql>
```

fig 18

This multi-line chart tracks daily ride activity, broken into **Completed**, **Cancelled**, and **No Cars Available**. Python used multiple condition-based filters over dates, and SQL queries filtered `Status` per day. It provides an operational snapshot of trends — for example, rising cancellations over weekends or consistent unavailability during specific periods. Such insights can trigger targeted interventions like increasing active drivers or deploying temporary location-based promotions.

### 10 Users' Frustration Visualization

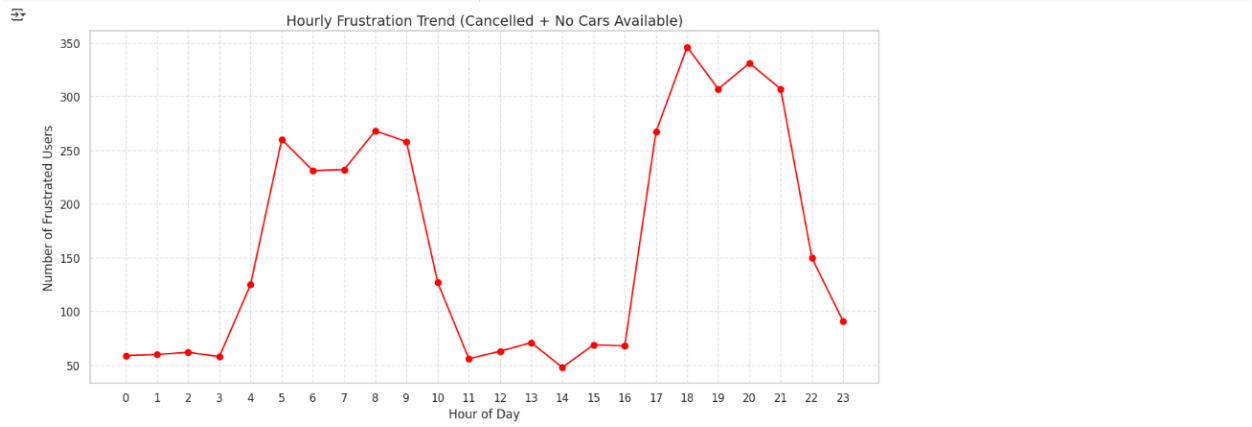


fig 19

```
mysql> SELECT
    ->   HOUR(Request_timestamp) AS request_hour,
    ->   SUM(CASE WHEN Status IN ('Cancelled', 'No Cars Available') THEN 1 ELSE 0 END) AS frustrated_users
    -> FROM uber_analysis
    -> GROUP BY request_hour
    -> ORDER BY frustrated_users DESC
    -> LIMIT 15;
+--------------+------------------+
| request_hour | frustrated_users |
+--------------+------------------+
|           18 |              346 |
|           20 |              331 |
|           19 |              307 |
|           21 |              307 |
|            8 |              268 |
|           17 |              267 |
|            5 |              260 |
|            9 |              258 |
|            7 |              232 |
|            6 |              231 |
|           22 |              150 |
|           10 |              127 |
|            4 |              125 |
|           23 |               91 |
|           13 |               71 |
+--------------+------------------+
15 rows in set (0.01 sec)

mysql>
```

fig 20

This visualization tracks failed requests (Cancelled or No Cars Available) over time, giving a view into user frustration. Python grouped non-completed statuses by hour/day, while SQL filtered `Status IN ('Cancelled', 'No Cars Available')`. High frustration periods directly affect customer retention. The visual acts as a red flag for operations, signaling urgent need for better supply chain execution during these peak failure times — be it through driver incentive programs, queue prediction, or alternate ride-sharing strategies.

## 4. Solution to Business Objective

### What do you suggest the client to achieve Business Objective ?

To help Uber achieve its business objectives—minimizing ride cancellations, reducing customer dissatisfaction, and optimizing driver allocation—several actionable solutions are proposed based on the findings of this project:

**Dynamic Driver Allocation** Uber should use demand prediction models (based on historical data) to pre-position drivers in high-demand zones such as airports and city centers during peak hours. This will reduce "No Cars Available" instances and increase completed ride percentages.

**Incentivize Drivers** During Peak Times Many cancellations and gaps in supply occur during early mornings and late evenings. Uber can introduce time-based incentives or surge bonuses to encourage more driver participation during these hours.

**Real-Time Monitoring Dashboard** Implement dashboards (using tools like Excel and MySQL visualizations) that track hourly requests, cancellations, and driver availability. This enables the operations team to quickly respond to sudden demand spikes.

**Improve App Notifications & User Communication** If cars are not available, real-time suggestions (like nearest pickup point or expected wait time) can help reduce user frustration and encourage retention.

**Data-Driven Expansion Planning** Using location-based trends, Uber can plan future driver recruitment and expansion strategies in under-served zones where demand is consistently high but fulfillment is low.

**Root Cause Analysis for Cancellations** Cancellations can be further categorized (e.g., driver-initiated vs. user-initiated), and Uber should implement policies or app improvements based on those insights to reduce them.

**By applying these data-driven strategies**, Uber can streamline its operations, enhance ride completion rates, and improve both rider and driver satisfaction—ultimately fulfilling the core business objective of reliable and efficient ride-hailing services.

# 5.Conclusion

The exploratory analysis of Uber's ride request data has revealed critical operational insights into the company’s daily functioning. Through detailed data wrangling, visualization, and interpretation, several patterns and challenges were uncovered:

High demand during peak hours (especially mornings and evenings) leads to a significant number of unfulfilled ride requests, primarily due to unavailability of cars.

Cancellations and no-car-available issues are concentrated around specific times and locations, especially from airport pickup points during peak demand.

There is a clear mismatch between supply (drivers available) and demand (user ride requests) that causes user frustration and service inefficiencies.

The data showed repeating trends in user behavior and driver response, pointing to areas where Uber can proactively deploy more resources.

Visual tools such as hourly request plots, pickup point heatmaps, and request-status charts helped in identifying these service gaps more clearly and effectively.

Overall, this project analysis provides a strong foundation for operational improvement.

# 6.References


- Dataset: Uber Ride Request Data (CSV Format, User-Provided)
- Libraries & Tools:
  - Python: `pandas`, `matplotlib`, `seaborn`
  - MySQL Server: version supporting `LOAD DATA LOCAL INFILE`
  - Microsoft Excel 2016+
- Official Documentation:
  - Pandas Documentation
  - Seaborn Documentation
  - MySQL Documentation
  - Microsoft Excel Support


# 7. Appendix

### *1. Data Cleaning & Preprocessing*

- **Microsoft Excel (MS Excel)**
  - Used for initial inspection, cleaning, and transformation of raw data.
  - Operations included:
    - Separating date and time columns
    - Merging AM/PM time formats
    - Removing unnecessary trailing formats (like “00:00”)
    - Handling missing values
    - Renaming columns for clarity
    - Applying Flash Fill, formulas, and filters
- **Python (Pandas Library)**
  - Used to re-verify, clean, and preprocess data programmatically after Excel operations
  - Enabled conversion of timestamps to datetime objects
  - Checked and removed duplicates, missing values
  - Derived new variables like “Request Hour”, “Day of Week”, etc.

### *2. Data Analysis & Visualization*

- **Python (Pandas, Seaborn, Matplotlib)**
  - Conducted Exploratory Data Analysis (EDA)
  - Generated key visuals including:
    - Hourly ride request trends
    - Pickup point demand
    - Supply-demand mismatch patterns
    - Status distribution via bar and pie charts
    - Users' frustration representation
  - Helped in identifying hidden patterns, peak hours, and critical problem zones
- **MySQL (Structured Query Language)**
  - Used to create a structured database (uber_analysis table) from the cleaned CSV
  - Enabled querying for detailed metrics including:
    - Hour-wise request counts
    - Driver availability trends
    - Pickup location-based performance
    - Request status distribution and frustration indicators
  - Provided deep drill-down views of the data to complement Python visualizations

### *3. Reporting & Visual Communication*

- **Microsoft Excel (Visual Dashboarding)**
  - Created parallel dashboard views for business users
  - Generated pivot charts, slicers, and comparative visuals
  - Focused on:
    - Day-wise & hour-wise request patterns
    - Driver assignment success/failure
    - Pickup zone comparisons
    - Frustration-inducing events like “No Cars Available”