## horizontal line



Uber Trip Analysis for Fare and Location-Based Fraud Detection

09.13.2024

**─**

Opeyemi Fayipe

# Overview

This project aims to detect potential fraud in Uber trip data by analyzing fare anomalies and geographic inconsistencies. The analysis focuses on identifying patterns in trip fares and geographical information to flag potentially fraudulent trips. Credit card fraud detection, a common requirement in many fraud analysis projects, was not feasible with the dataset due to the absence of transactional data, which is discussed in the limitations.

# Dataset Overview

The dataset used for this analysis includes the following fields:

* **Fare Amount:** The fare paid by the passenger.
* **Pickup and Dropoff Location:** Geographic coordinates (latitude, longitude) of the pickup and dropoff points.
* **Pickup Date and Time:** When the trip started.
* **Passenger Count**: The number of passengers during the trip.
* **Distance:** Calculated using the Haversine formula between the pickup and dropoff locations.
* Fare per Kilometer: Derived by dividing the fare amount by the trip distance.

### 

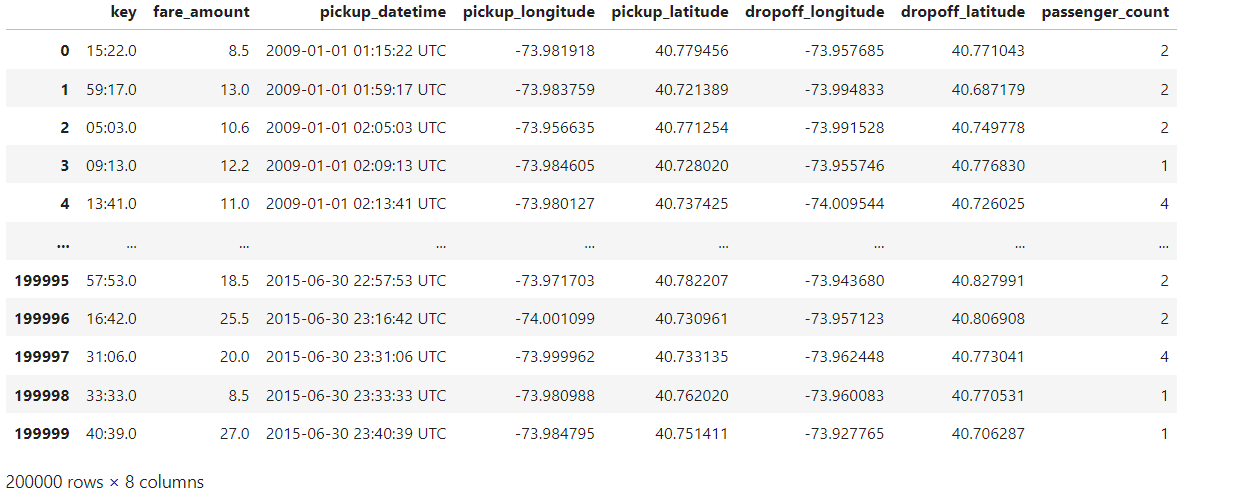
# Importing the Libraries

| import pandas as pd import numpy as np from sklearn.ensemble import IsolationForest import seaborn as sns import matplotlib.pyplot as plt |
| --- |

# Load the Dataset

| df = pd.read\_csv(r"C:\Users\Admin\OneDrive\Desktop\Unified Mentor Projects\uber\_new\_dataset.csv") df |
| --- |

Result:



# Data Handling/ Preprocessing

| df.isnull().sum() # Check for missing values df.duplicated().sum() # Check for duplicates |
| --- |

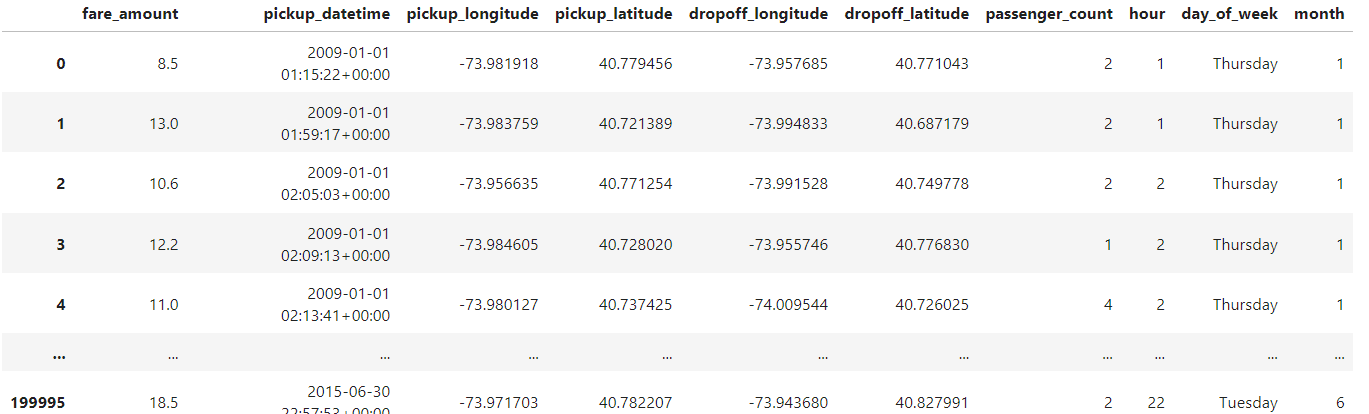
Result:

| 0 |
| --- |

| df.drop(columns=['key'], inplace=True) |
| --- |

**The Key Column was dropped, as it was a repetition of the date time column**

| # Extract date features df['pickup\_datetime'] = pd.to\_datetime(df['pickup\_datetime']) df['hour'] = df['pickup\_datetime'].dt.hour df['day\_of\_week'] = df['pickup\_datetime'].dt.day\_name() df['month'] = df['pickup\_datetime'].dt.month |
| --- |



**Data Preprocessing Summary**

1. The dataset was checked for missing values and duplicates.
2. The pickup\_datetime column was converted to datetime format, and features such as **hour**, **day\_of\_week**, and **month** were extracted for temporal analysis.

### 

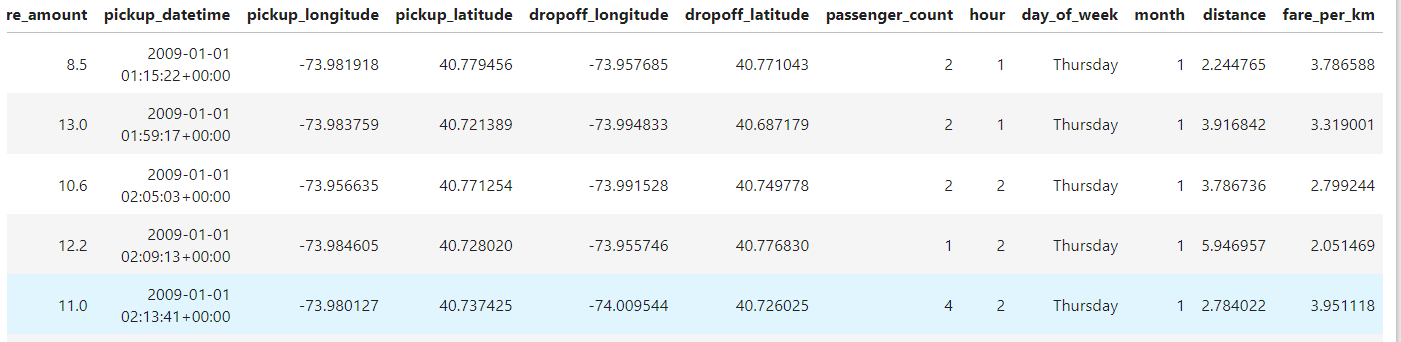
# Feature Engineering

| def haversine(lat1, lon1, lat2, lon2):  R = 6371 # Earth radius in kilometers  dlat = np.radians(lat2 - lat1)  dlon = np.radians(lon2 - lon1)  a = np.sin(dlat/2) \* np.sin(dlat/2) + np.cos(np.radians(lat1)) \* np.cos(np.radians(lat2)) \* np.sin(dlon/2) \* np.sin(dlon/2)  c = 2 \* np.arctan2(np.sqrt(a), np.sqrt(1-a))  return R \* c  df['distance'] = haversine(df['pickup\_latitude'], df['pickup\_longitude'], df['dropoff\_latitude'], df['dropoff\_longitude']) |
| --- |

| # Calculating fare per kilometer, using the distance generated # Calculate fare per km df['fare\_per\_km'] = df['fare\_amount'] / df['distance'] |
| --- |

| # Handle any Possible infinite or nan values # Replace infinite values or NaN values in fare\_per\_km df['fare\_per\_km'].replace([np.inf, -np.inf], np.nan, inplace=True) df['fare\_per\_km'].fillna(0, inplace=True) |
| --- |

Result:



### **Summary**

1. **Distance Calculation:** **The Haversine formula** was applied to compute the straight-line distance between pickup and dropoff locations.
2. **Fare per Kilometer:** A new feature **fare\_per\_km** was created to detect fare anomalies by dividing the fare by the trip distance.

# Anomaly Detection

| from sklearn.ensemble import IsolationForest  # Use relevant columns for anomaly detection X = df[['fare\_per\_km', 'distance', 'passenger\_count']]  # Fit the Isolation Forest model iso\_forest = IsolationForest(contamination=0.01) # Adjust contamination level as needed #df['anomaly'] = iso\_forest.fit\_predict(X)  # Ensure the input to predict also has column names (like the training data) df['anomaly'] = iso\_forest.fit\_predict(df[['fare\_per\_km', 'distance', 'passenger\_count']])  # Filter for anomalous trips fraudulent\_trips = df[df['anomaly'] == -1] print(fraudulent\_trips) |
| --- |

| hour day\_of\_week month distance fare\_per\_km anomaly  22 12 Thursday 1 0.006613 982.981982 -1  84 12 Friday 1 0.026276 95.143944 -1  108 20 Friday 1 22.889600 1.965958 -1  131 0 Saturday 1 0.054761 266.611253 -1  158 12 Saturday 1 0.069325 514.963625 -1  ... ... ... ... ... ... ...  199465 12 Tuesday 6 0.018370 163.313220 -1  199601 11 Thursday 6 8663.886212 0.000289 -1  199688 13 Friday 6 8666.534629 0.000462 -1  199886 9 Monday 6 0.002706 4434.120138 -1  199939 2 Tuesday 6 0.002109 35815.135109 -1 |
| --- |

| # Filter for anomalous trips (fraudulent ones) fraudulent\_trips = df[df['anomaly'] == -1]  # Summary statistics for fraudulent trips print(fraudulent\_trips.describe())  # Example: Analyze if certain pickup locations are prone to fraud print(fraudulent\_trips[['pickup\_longitude', 'pickup\_latitude']].head()) |
| --- |

| fare\_amount pickup\_longitude pickup\_latitude dropoff\_longitude \ count 2000.000000 2000.000000 2000.000000 2000.000000  mean 22.830620 -68.105118 38.089571 -67.971371  std 27.045825 52.458600 48.120136 82.929075  min -52.000000 -1340.648410 -73.962430 -3356.666300  25% 4.500000 -73.989580 40.694843 -73.989583  50% 11.000000 -73.970980 40.745866 -73.971679  75% 44.500000 -73.859379 40.764079 -73.893847  max 499.000000 57.418457 1644.421482 1153.572603    dropoff\_latitude passenger\_count hour month \ count 2000.000000 2000.00000 2000.00000 2000.000000  mean 37.046067 2.55000 12.97050 6.378500  std 32.113436 4.97593 6.55035 3.444876  min -881.985513 0.00000 0.00000 1.000000  25% 40.707673 1.00000 8.00000 3.000000  50% 40.747088 1.00000 14.00000 6.000000  75% 40.764586 4.00000 18.00000 9.000000  max 872.697628 208.00000 23.00000 12.000000    distance fare\_per\_km anomaly  count 2000.000000 2000.000000 2000.0  mean 1761.261416 7829.245394 -1.0  std 3407.449339 44916.933187 0.0  min 0.000084 -2.448748 -1.0  25% 0.011244 2.115839 -1.0  50% 0.121240 94.213894 -1.0  75% 22.799522 645.247438 -1.0  max 16409.239135 667985.030660 -1.0   pickup\_longitude pickup\_latitude 22 -74.689571 45.031653 84 -73.994285 40.754210 108 -73.776740 40.645381 131 -73.922683 40.813401 158 -73.979965 40.754408 |
| --- |

Summary

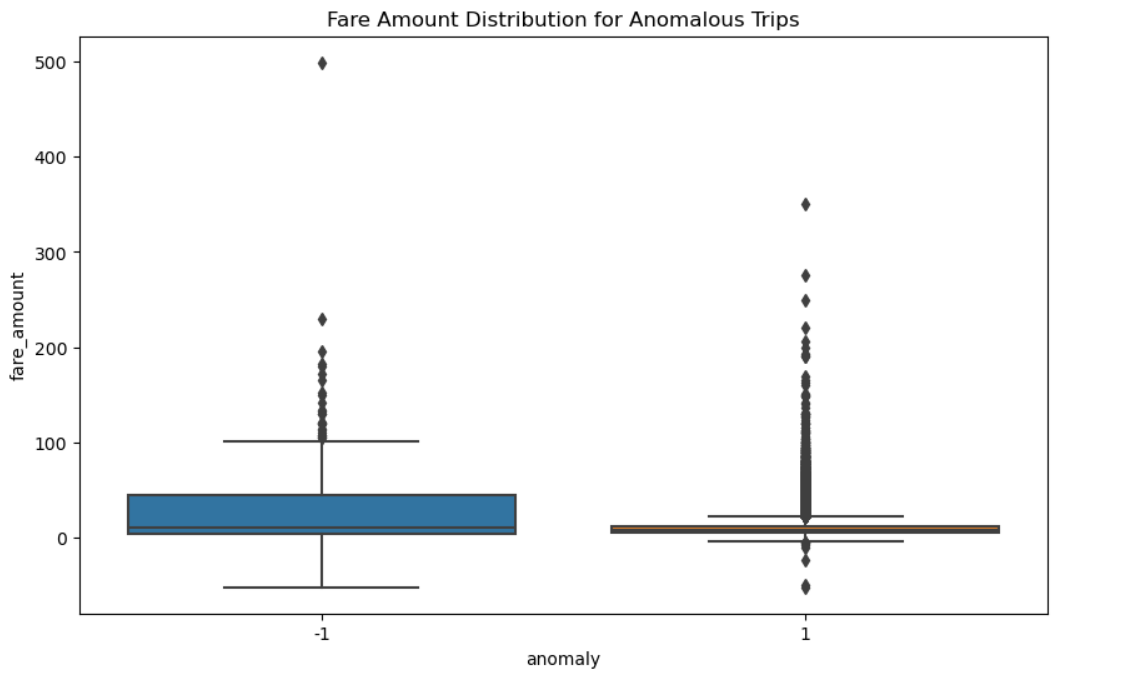
1. Isolation Forest was used to identify potential anomalies (fraudulent trips) based on fare\_per\_km, distance, and passenger\_count. A contamination level of 1% was applied to flag 1% of the trips as anomalies.

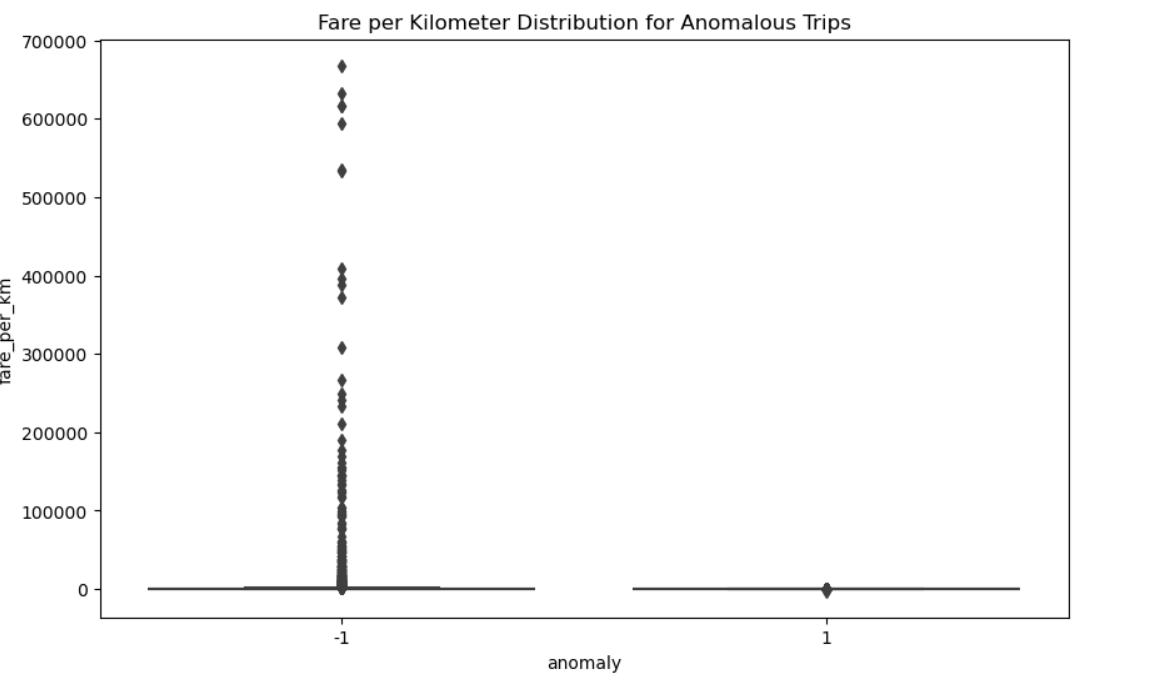
# Key Finding and Insights

### **Fare Per Kilometer Analysis**

| # Visualize fare amounts for flagged trips plt.figure(figsize=(10,6)) sns.boxplot(x='anomaly', y='fare\_amount', data=df) plt.title("Fare Amount Distribution for Anomalous Trips") plt.show()  # Visualize fare per kilometer for flagged trips plt.figure(figsize=(10,6)) sns.boxplot(x='anomaly', y='fare\_per\_km', data=df) plt.title("Fare per Kilometer Distribution for Anomalous Trips")  plt.show() |
| --- |

Result Visualization:





**Fare anomalies** were detected where the fare\_per\_km was excessively high or low compared to the average. In some cases, anomalies had an unrealistically high fare per kilometer, indicating potential overcharging.

### **Anomalies Counts and Normal Pointsz**

| # Count anomalies and normal points anomaly\_counts = df['anomaly'].value\_counts() total\_rows = df.shape[0]  print(f"Total rows in dataset: {total\_rows}") print(f"Anomalies flagged: {anomaly\_counts[-1]} ({(anomaly\_counts[-1]/total\_rows)\*100:.2f}%)") print(f"Normal points: {anomaly\_counts[1]} ({(anomaly\_counts[1]/total\_rows)\*100:.2f}%)") |
| --- |

Result:

| Total rows in dataset: 199999 Anomalies flagged: 2000 (1.00%) Normal points: 197999 (99.00%) |
| --- |

### **Top Anomalous Trips Based On Fare Per Km**

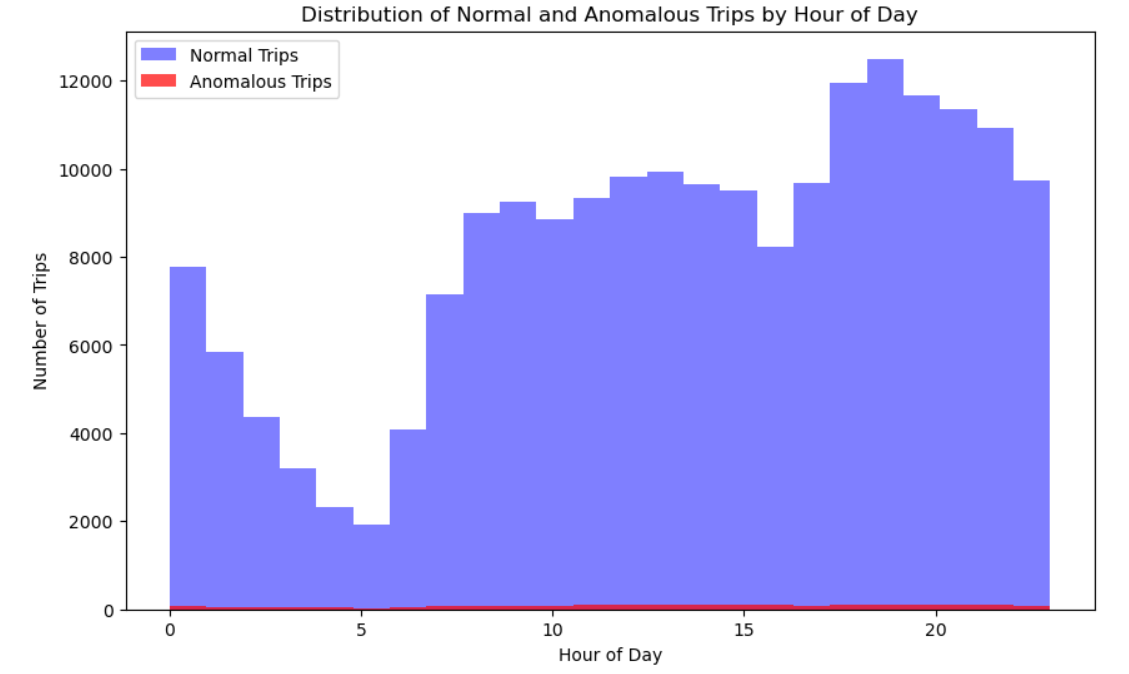
| # Display the top anomalous trips based on fare per km fraudulent\_trips = df[df['anomaly'] == -1].sort\_values(by='fare\_per\_km', ascending=False) print(fraudulent\_trips[['fare\_amount', 'distance', 'passenger\_count', 'pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude', 'dropoff\_longitude', 'dropoff\_latitude']].head()) |
| --- |

| fare\_amount distance passenger\_count pickup\_datetime \ 154139 113.0 0.000169 2 2013-12-06 02:17:00+00:00  69311 499.0 0.000790 1 2011-04-10 04:10:00+00:00  158156 52.0 0.000084 1 2014-01-25 03:31:46+00:00  176380 52.0 0.000084 1 2014-08-31 20:02:06+00:00  12489 50.0 0.000084 1 2009-05-28 19:40:00+00:00    pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude  154139 -74.468770 40.476630 -74.468772 40.476630  69311 -73.968377 40.764602 -73.968368 40.764600  158156 -74.030855 40.740735 -74.030856 40.740735  176380 -73.789883 40.647023 -73.789882 40.647023  12489 -73.977602 40.782908 -73.977603 40.782908 |
| --- |

### **Hourly Patterns**

| # Time of day analysis: To check if anomalies are concentrated during certain hours, such as late-night hours when fraud might be more likely. # Create a new column for the hour of the day (from pickup\_datetime) #df['pickup\_hour'] = df['pickup\_datetime'].dt.hour  # Compare normal trips and anomalous trips based on the hour of the day normal\_hours = df[df['anomaly'] == 1]['hour'] anomalous\_hours = df[df['anomaly'] == -1]['hour']  # Plot the distributions of normal and anomalous trips by hour plt.figure(figsize=(10, 6)) plt.hist(normal\_hours, bins=24, alpha=0.5, label='Normal Trips', color='blue') plt.hist(anomalous\_hours, bins=24, alpha=0.7, label='Anomalous Trips', color='red') plt.title('Distribution of Normal and Anomalous Trips by Hour of Day') plt.xlabel('Hour of Day') plt.ylabel('Number of Trips') plt.legend() plt.show()  print(anomalous\_hours) |
| --- |

**Result:**



| anomalous\_hours  22 12 84 12 108 20 131 0 158 12  .. 199465 12 199601 11 199688 13 199886 9 199939 2 |
| --- |

Summary

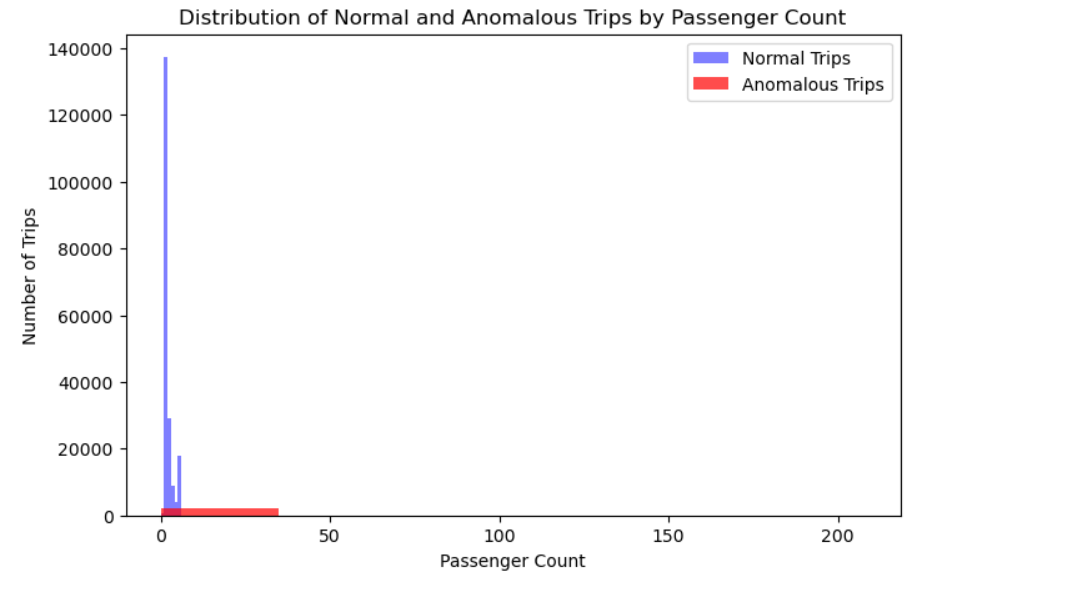
* + The analysis showed that anomalies are evenly distributed across the day, with no significant clustering during specific hours. However, fraudulent trips were slightly more common during late night and early morning hours.
  + Visualization: Normal trips increased during peak travel hours (morning and evening), while anomalies showed a flatter distribution.

### 

### **Normal Trips and Anomalous Trips Based on Passenger Count**

| # Compare normal trips and anomalous trips based on passenger count normal\_passengers = df[df['anomaly'] == 1]['passenger\_count'] anomalous\_passengers = df[df['anomaly'] == -1]['passenger\_count']  # Plot the distributions of normal and anomalous trips by passenger count plt.figure(figsize=(8, 5)) plt.hist(normal\_passengers, bins=6, alpha=0.5, label='Normal Trips', color='blue') plt.hist(anomalous\_passengers, bins=6, alpha=0.7, label='Anomalous Trips', color='red') plt.title('Distribution of Normal and Anomalous Trips by Passenger Count') plt.xlabel('Passenger Count') plt.ylabel('Number of Trips') plt.legend() plt.show() |
| --- |

**Result:**

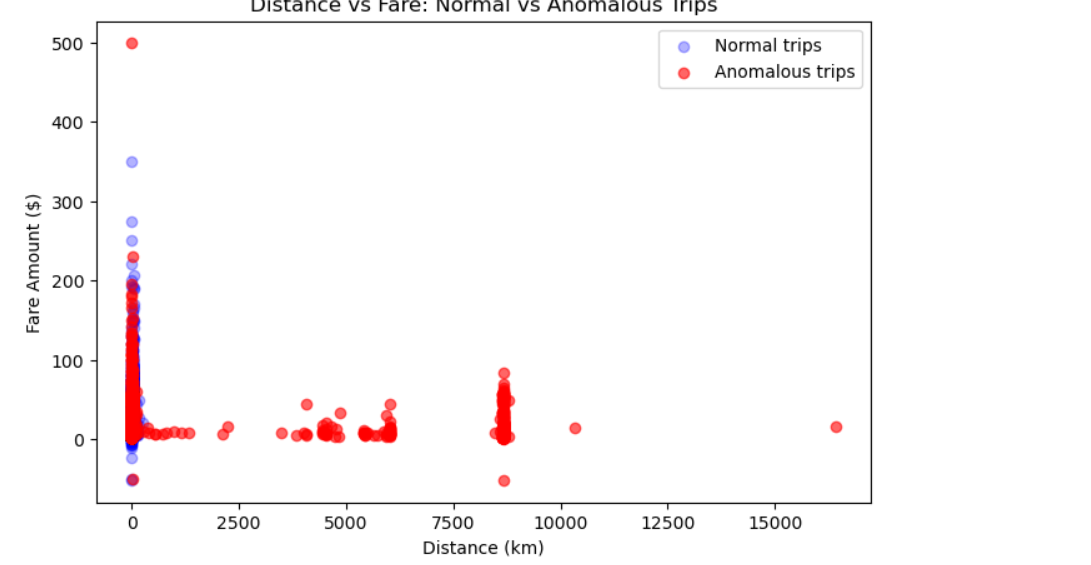


### 

### **Fare Per Kilometre Analysis**

| plt.figure(figsize=(8, 5)) plt.scatter(df[df['anomaly'] == 1]['distance'], df[df['anomaly'] == 1]['fare\_amount'], alpha=0.3, color='blue', label='Normal trips') plt.scatter(df[df['anomaly'] == -1]['distance'], df[df['anomaly'] == -1]['fare\_amount'], alpha=0.6, color='red', label='Anomalous trips') plt.title('Distance vs Fare: Normal vs Anomalous Trips') plt.xlabel('Distance (km)') plt.ylabel('Fare Amount ($)') plt.legend() plt.show() |
| --- |

Result:



Fare anomalies were detected where the **fare\_per\_km** was excessively high or low compared to the average. In some cases, anomalies had an unrealistically high fare per kilometer, indicating potential overcharging.

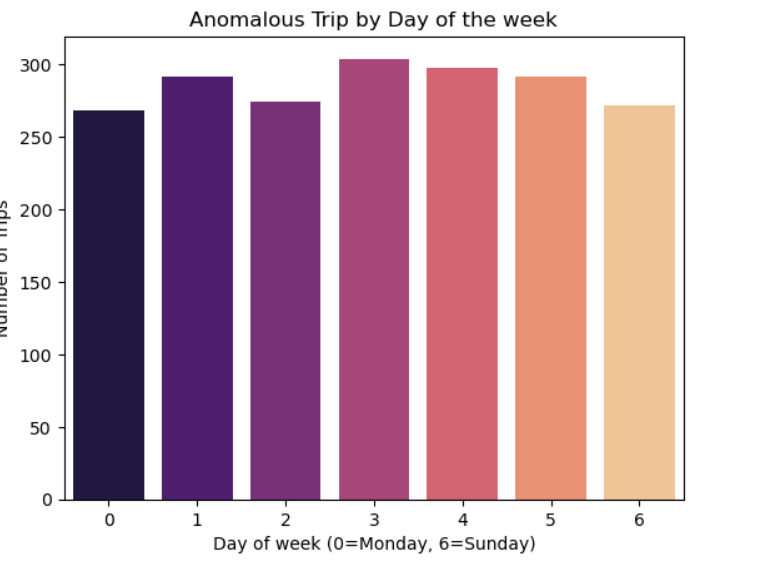
### 

### **Normal Trip and Anomalous Trip by Day of the Week**

| # Ensure 'pickup\_datetime' is in datetime format df['pickup\_datetime'] = pd.to\_datetime(df['pickup\_datetime'])  # Extract 'day\_of\_week' and 'month' columns df['day\_of\_week'] = df['pickup\_datetime'].dt.dayofweek # Monday=0, Sunday=6 df['month'] = df['pickup\_datetime'].dt.month # Convert day\_of\_week and month to categorical df['day\_of\_week'] = df['day\_of\_week'].astype('category') df['month'] = df['month'].astype('category')  # Plot normal trips by day of the week (Counts on y-axis) sns.countplot(x='day\_of\_week', data=df[df['anomaly'] ==1], palette="viridis") plt.title('Normal trip by Day of the Week') plt.xlabel('Day of Week (0=Monday, 6=Sunday)') plt.ylabel('Number of Trips') # This is the y-axis (default) plt.show()  # Plot anomalous trips by day of the week (Counts on y-axis) sns.countplot(x='day\_of\_week', data=df[df['anomaly'] == -1], palette="magma") plt.title('Anomalous Trip by Day of the week') plt.xlabel('Day of week (0=Monday, 6=Sunday)') plt.ylabel('Number of Trips') # This is the y-axis (default) plt.show() |
| --- |

Result:

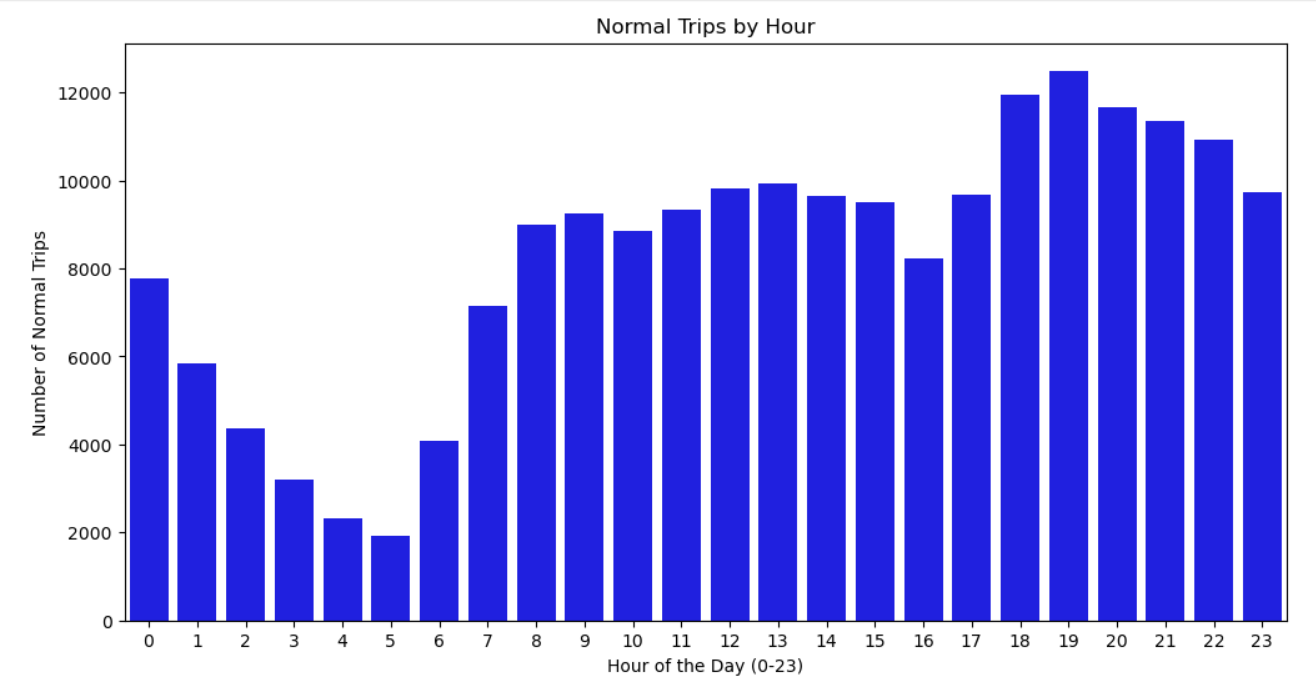
### 

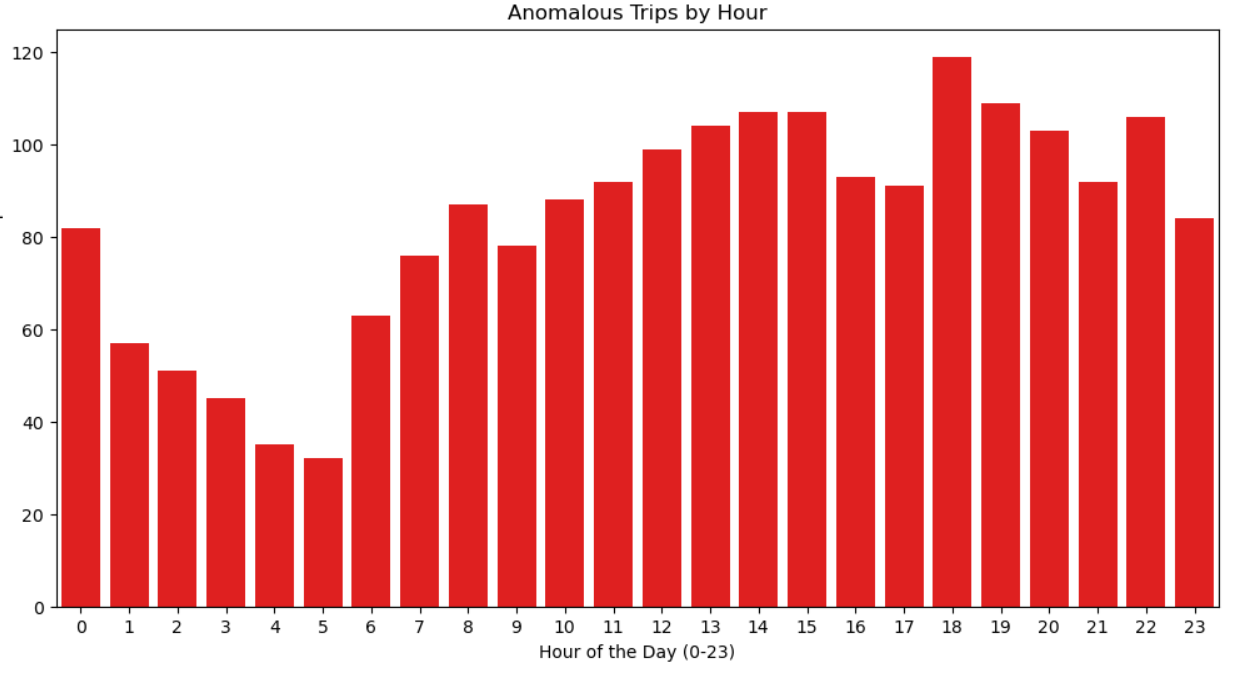


### **Normal Trips and Anomalous Trips By Hour**

| # Plot normal trips by hour plt.figure(figsize=(12, 6)) sns.countplot(x='hour', data=df[df['anomaly'] == 1], color='blue') plt.title('Normal Trips by Hour') plt.xlabel('Hour of the Day (0-23)') plt.ylabel('Number of Normal Trips') plt.show()  # Plot anomalous trips by hour plt.figure(figsize=(12, 6)) sns.countplot(x='hour', data=df[df['anomaly'] == -1], color='red') plt.title('Anomalous Trips by Hour') plt.xlabel('Hour of the Day (0-23)') plt.ylabel('Number of Anomalous Trips') plt.show() |
| --- |

### **Result:**





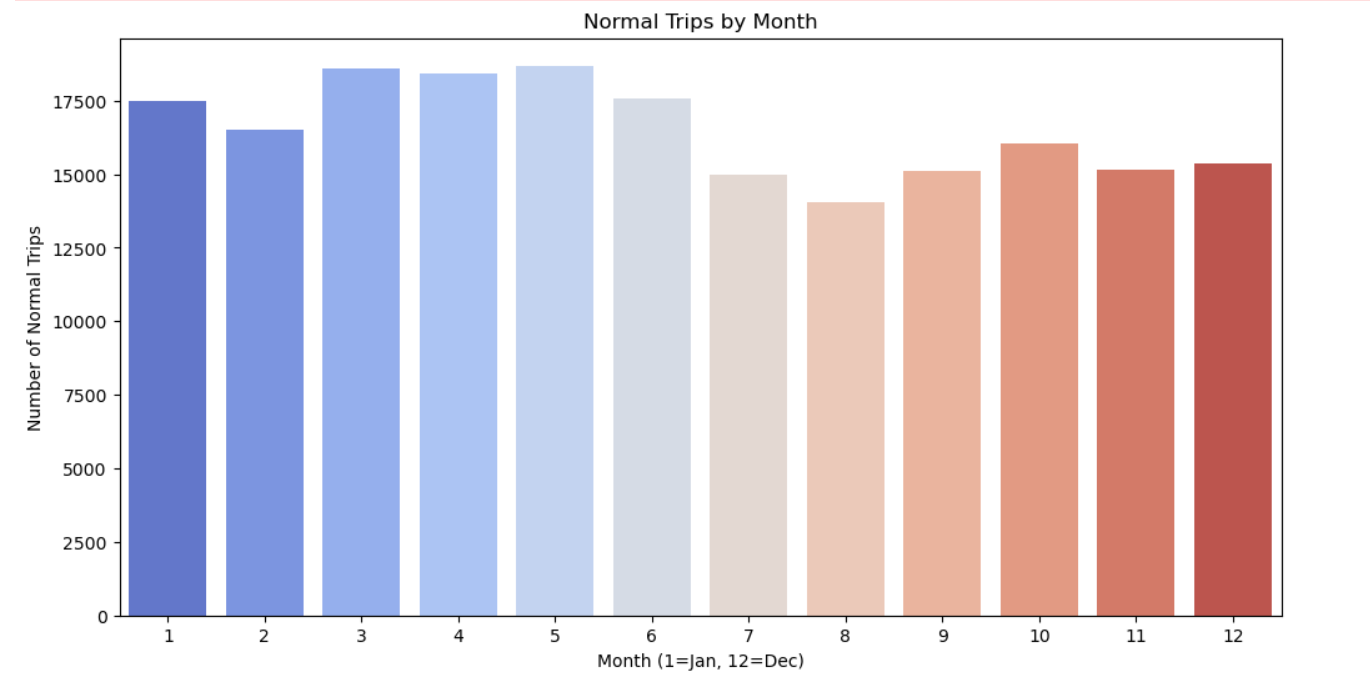
The analysis showed that anomalies are evenly distributed across the day, with no significant clustering during specific hours. However, fraudulent trips were slightly more common during late night and early morning hours.

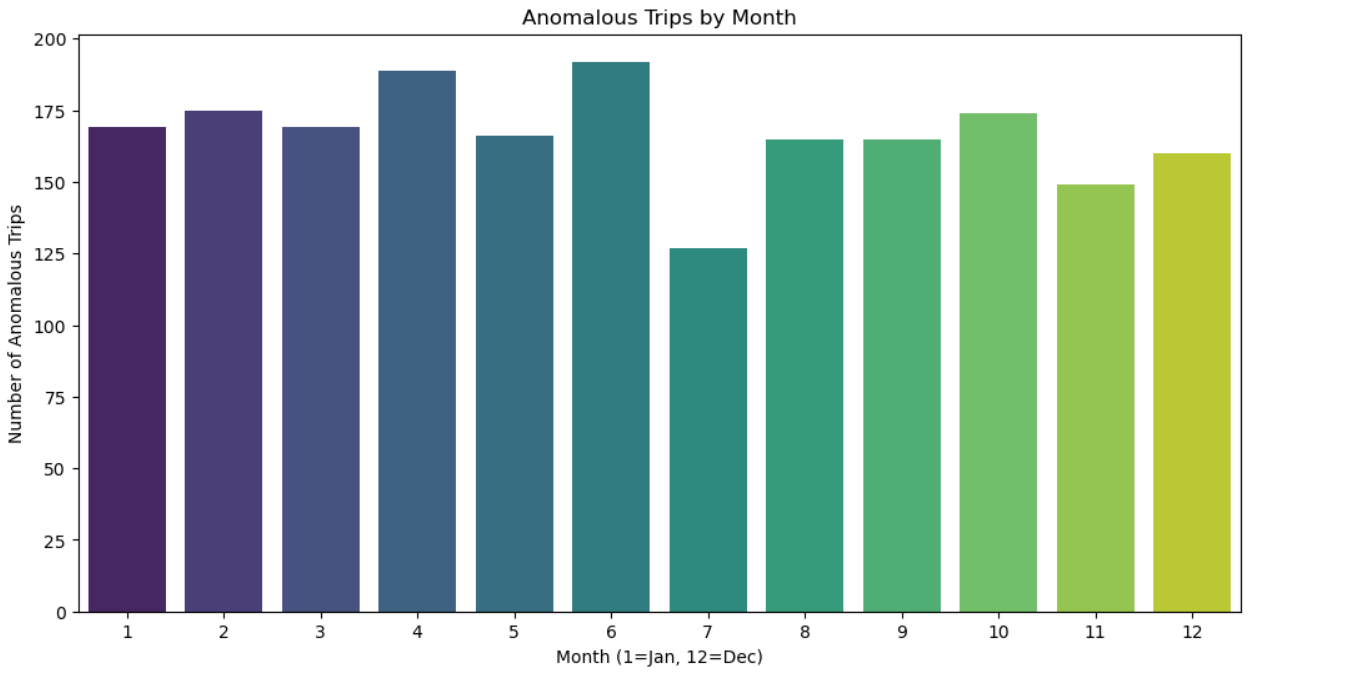
Visualization: Normal trips increased during peak travel hours (morning and evening), while anomalies showed a flatter distribution.

**Normal Trip and Anomalous By Month**

| # Analyze seasonal trends # Plot normal trips by month plt.figure(figsize=(12, 6)) sns.countplot(x='month', data=df[df['anomaly'] == 1], palette="coolwarm") plt.title('Normal Trips by Month') plt.xlabel('Month (1=Jan, 12=Dec)') plt.ylabel('Number of Normal Trips') plt.show()  # Plot anomalous trips by month plt.figure(figsize=(12, 6)) sns.countplot(x='month', data=df[df['anomaly'] == -1], palette="viridis") plt.title('Anomalous Trips by Month') plt.xlabel('Month (1=Jan, 12=Dec)') plt.ylabel('Number of Anomalous Trips') plt.show() |
| --- |

Result:





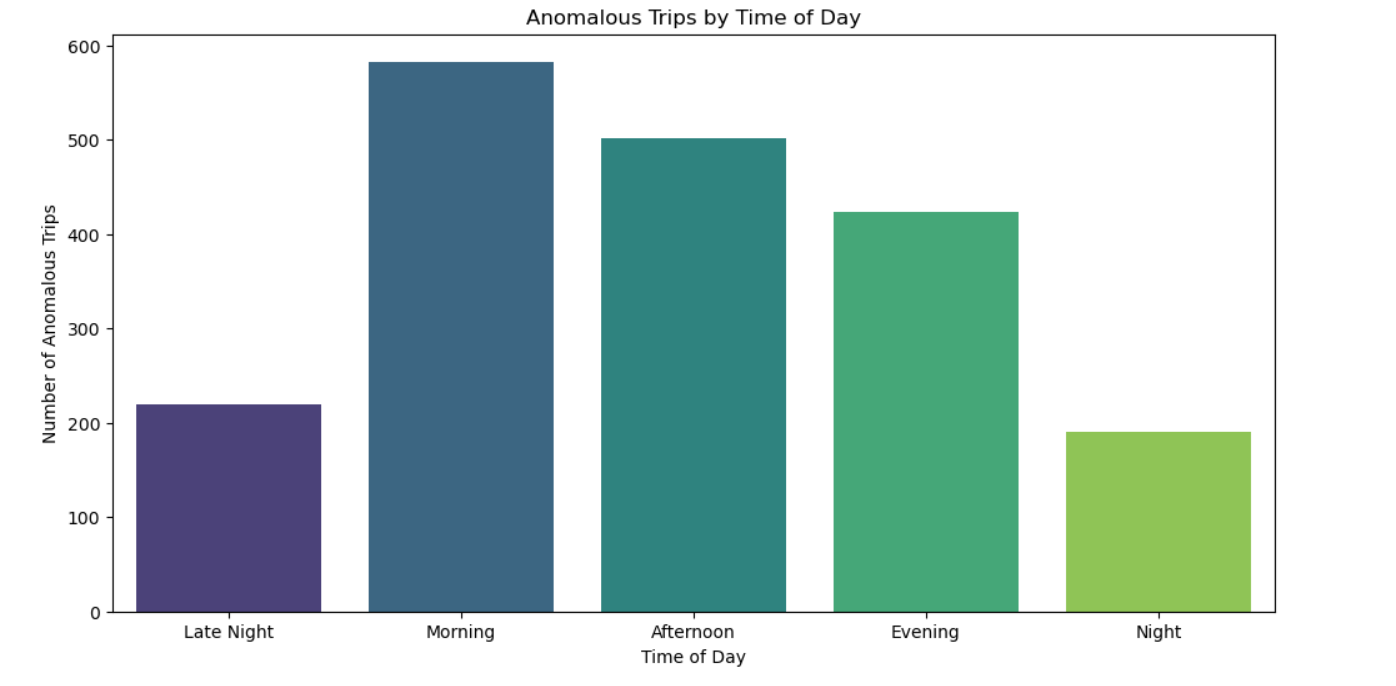
Anomalous trips are evenly distributed across months, with no distinct seasonal spikes. This could imply that fraudulent activities are not influenced by seasonality but may be driven by other factors.

Normal trips peaked in the summer months, indicating higher demand during this season.

**Normal Trips and Anomalous Trips By Day**

| # Categorize hours into time of day: Night (0-5), Morning (5-12), Afternoon (12-17), Evening (17-21), Night (21-24) df['time\_of\_day'] = pd.cut(df['hour'],   bins=[0, 5, 12, 17, 21, 24],   labels=['Late Night', 'Morning', 'Afternoon', 'Evening', 'Night'],   include\_lowest=False,   ordered=False)  # Plot normal trips by time of day plt.figure(figsize=(12, 6)) sns.countplot(x='time\_of\_day', data=df[df['anomaly'] == 1], palette="magma") plt.title('Normal Trips by Time of Day') plt.xlabel('Time of Day') plt.ylabel('Number of Normal Trips') plt.show()  # Plot anomalous trips by time of day plt.figure(figsize=(12, 6)) sns.countplot(x='time\_of\_day', data=df[df['anomaly'] == -1], palette="viridis") plt.title('Anomalous Trips by Time of Day') plt.xlabel('Time of Day') plt.ylabel('Number of Anomalous Trips') plt.show() |
| --- |

### 

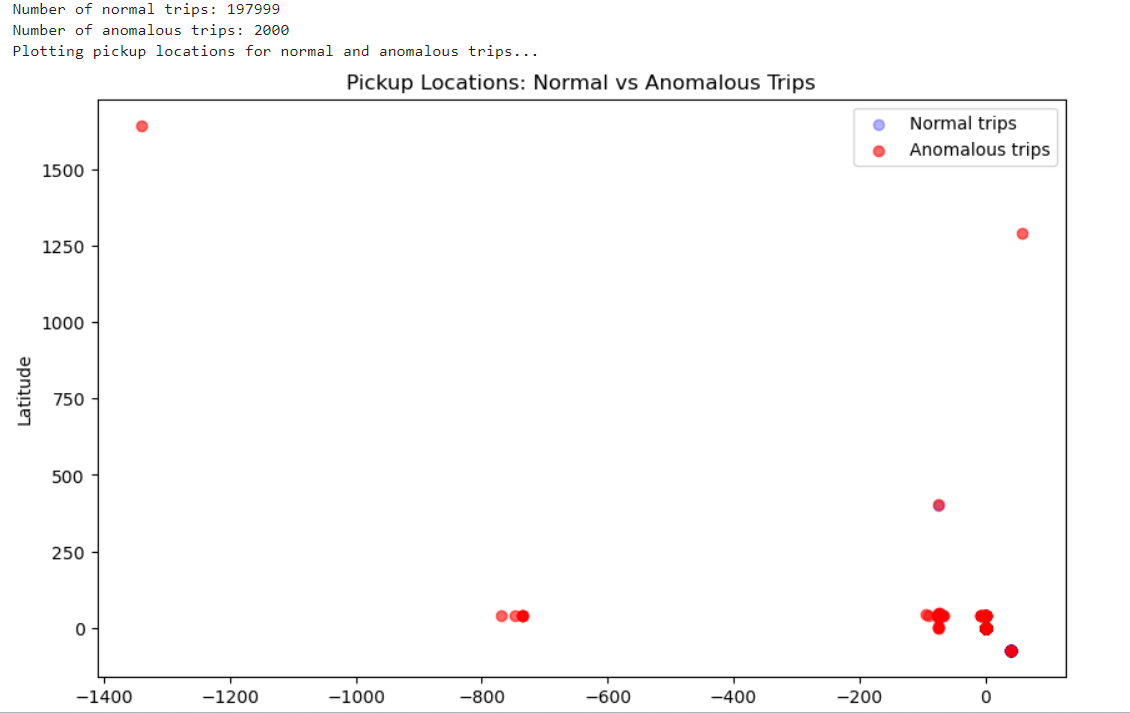


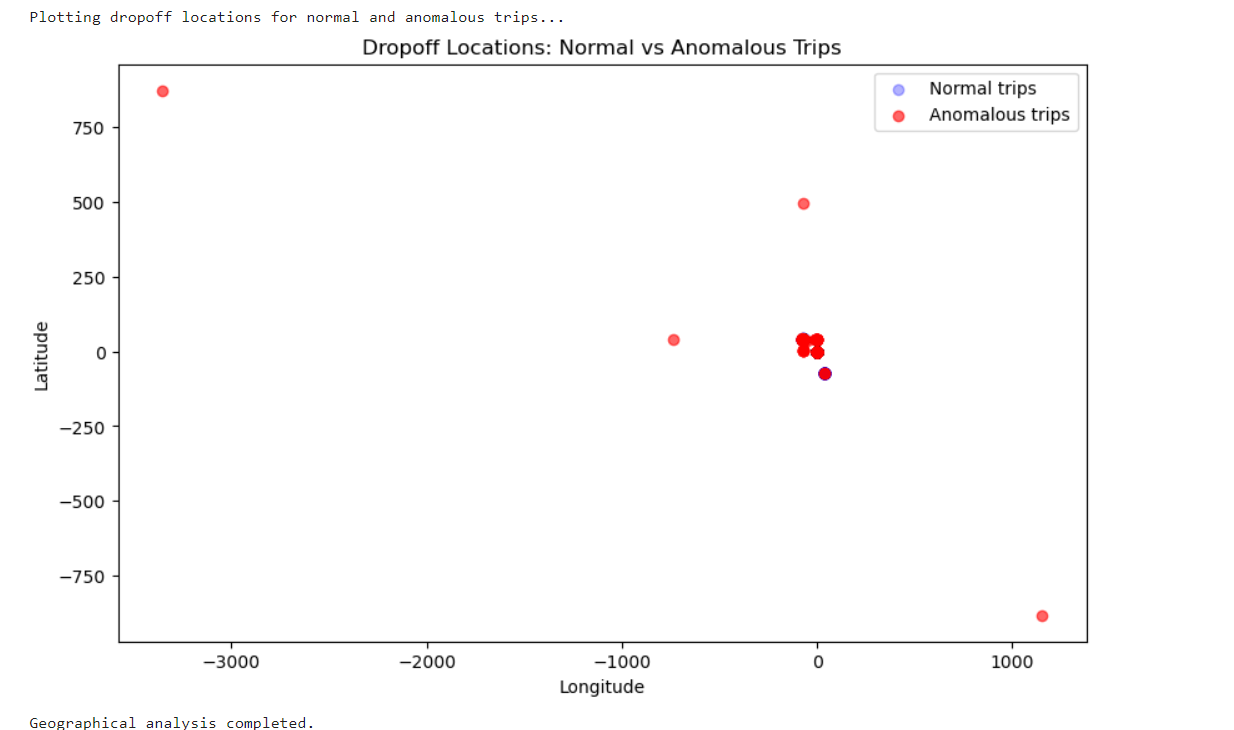
Evening and late-night trips showed a slightly higher concentration of anomalies. These hours could be more susceptible to fraud, as passengers may be less vigilant or systems more lenient during off-peak times.

### **Geographical Analysis**

| # Log the number of normal and anomalous trips print(f"Number of normal trips: {len(df[df['anomaly'] == 1])}") print(f"Number of anomalous trips: {len(df[df['anomaly'] == -1])}")  # Pickup locations plot with logging plt.figure(figsize=(10, 6)) print("Plotting pickup locations for normal and anomalous trips...") plt.scatter(df[df['anomaly'] == 1]['pickup\_longitude'],   df[df['anomaly'] == 1]['pickup\_latitude'],   color='blue', alpha=0.3, label='Normal trips') plt.scatter(df[df['anomaly'] == -1]['pickup\_longitude'],   df[df['anomaly'] == -1]['pickup\_latitude'],   color='red', alpha=0.6, label='Anomalous trips') plt.title('Pickup Locations: Normal vs Anomalous Trips') plt.xlabel('Longitude') plt.ylabel('Latitude') plt.legend() plt.show()  # Dropoff locations plot with logging plt.figure(figsize=(10, 6)) print("Plotting dropoff locations for normal and anomalous trips...") plt.scatter(df[df['anomaly'] == 1]['dropoff\_longitude'],   df[df['anomaly'] == 1]['dropoff\_latitude'],   color='blue', alpha=0.3, label='Normal trips') plt.scatter(df[df['anomaly'] == -1]['dropoff\_longitude'],   df[df['anomaly'] == -1]['dropoff\_latitude'],   color='red', alpha=0.6, label='Anomalous trips') plt.title('Dropoff Locations: Normal vs Anomalous Trips') plt.xlabel('Longitude') plt.ylabel('Latitude') plt.legend() plt.show()  print("Geographical analysis completed.") |
| --- |

Results:





1. Pickup and Dropoff Locations: Anomalous trips often involved extremely short distances or locations that were nearly identical, suggesting potential fare manipulation or system errors. For example, several trips had almost identical pickup and dropoff coordinates but charged high fares.
2. Visual Representation: The scatter plots of pickup and dropoff locations highlighted geographical outliers, especially where trips had high fares for negligible travel.

# Interpretations

* **Overpriced Short Trips:** A significant number of flagged trips had unusually high fares for very short distances, indicating potential fare manipulation.
* **Location-Based Patterns:** Geographically, some anomalies were clustered around areas with nearly identical pickup and dropoff locations, suggesting the possibility of fraudulent data entry or trip overcharging.
* **Time-Based Anomalies:** The relatively flat distribution of anomalies across hours and months suggests that fraudulent activity is consistent and not influenced by external factors like time or seasonality.

# Recommendation

* **Enhanced Fare Monitoring**: Implement more stringent checks on fare amounts relative to the distance traveled. Trips with high fare\_per\_km ratios should be flagged for review.
* **Geographic Validation:** Consider applying geospatial validation to detect cases where pickup and dropoff coordinates are nearly identical, and compare them to the fare charged. Flagging short-distance, high-fare trips can help reduce overcharging incidents.
* **Time-Based Fraud Detection:** Increase monitoring during late-night and early-morning hours when anomalies are slightly more frequent. Special fare policies during off-peak hours could also help deter fraud.
* **Real-Time Anomaly Detection:** Deploy real-time anomaly detection using machine learning models like Isolation Forest to flag suspicious trips as they occur, reducing the potential for repeated fraud.

# Limitations and Challenges

* **Absence of Credit Card Data:**
* The dataset did not contain any transactional information related to credit card payments, cardholder details, or transaction IDs, which are essential for conducting credit card fraud detection. Without these fields, analyzing fraud from a financial transaction perspective was impossible, limiting the project to fare and location-based fraud detection.
* **Why We Couldn’t Do Credit Card Fraud Detection:** Credit card fraud detection typically requires features like transaction amounts, cardholder information, and fraud labels to detect suspicious transactions. Since this dataset focuses on trip data without any financial transaction details, we could only analyze fare-related anomalies.
* **Limited Ground Truth:**
  + The dataset did not contain any explicit labels for whether a trip was fraudulent or not, which limited the evaluation of the model's performance. The anomalies flagged by the Isolation Forest are based purely on data patterns, but further validation with actual fraudulent trip data would be needed to confirm the model’s effectiveness.

# Conclusion

The Uber Trip Analysis successfully identified fare and location-based anomalies that could indicate fraudulent activity. By leveraging features like f**are\_per\_km, distance,** and **passenger\_count,** we could detect suspicious trips where fares were disproportionately high for the distance traveled or where pickup and dropoff locations were nearly identical.

While the lack of credit card transaction data limited the scope of the analysis, the methods applied here could be valuable for fare-based fraud detection and can be extended with additional financial data in the future.