In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = 'C:/College/Final Sem/Big Data and Advanced Database Concepts/nyc_taxi_trip_duration.csv'
nyc_taxi_data = pd.read_csv(file_path)

# Display the first few rows
print(nyc_taxi_data.head())
```

```
          id  vendor_id      pickup_datetime     dropoff_datetime  \
0  id1080784          2  2016-02-29 16:40:21  2016-02-29 16:47:01
1  id0889885          1  2016-03-11 23:35:37  2016-03-11 23:53:57
2  id0857912          2  2016-02-21 17:59:33  2016-02-21 18:26:48
3  id3744273          2  2016-01-05 09:44:31  2016-01-05 10:03:32
4  id0232939          1  2016-02-17 06:42:23  2016-02-17 06:56:31

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.953918        40.778873         -73.963875
1                2        -73.988312        40.731743         -73.994751
2                2        -73.997314        40.721458         -73.948029
3                6        -73.961670        40.759720         -73.956779
4                1        -74.017120        40.708469         -73.988182

   dropoff_latitude store_and_fwd_flag  trip_duration
0         40.771164                  N            400
1         40.694931                  N           1100
2         40.774918                  N           1635
3         40.780628                  N           1141
4         40.740631                  N            848
```

In [2]:
```
#Summary information about the dataset to check for missing values and data types
print(nyc_taxi_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 729322 entries, 0 to 729321
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  729322 non-null  object 
 1   vendor_id           729322 non-null  int64  
 2   pickup_datetime     729322 non-null  object 
 3   dropoff_datetime    729322 non-null  object 
 4   passenger_count     729322 non-null  int64  
 5   pickup_longitude    729322 non-null  float64
 6   pickup_latitude     729322 non-null  float64
 7   dropoff_longitude   729322 non-null  float64
 8   dropoff_latitude    729322 non-null  float64
 9   store_and_fwd_flag  729322 non-null  object 
 10  trip_duration       729322 non-null  int64  
dtypes: float64(4), int64(3), object(4)
memory usage: 61.2+ MB
None
```

In [3]:
```python
# Drop duplicates if any
nyc_taxi_data.drop_duplicates(inplace=True)

# Check for missing values
missing_values = nyc_taxi_data.isnull().sum()
print("Missing values:\n", missing_values)

# Fill or drop missing values as necessary
nyc_taxi_data.fillna(method='ffill', inplace=True)

# Ensure data types are correct, convert pickup_datetime if it requires
if 'pickup_datetime' in nyc_taxi_data.columns:
    nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])

# Display info to verify data types
print(nyc_taxi_data.info())
```

```
Missing values:
 id                   0
vendor_id             0
pickup_datetime       0
dropoff_datetime      0
passenger_count       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
store_and_fwd_flag    0
trip_duration         0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 729322 entries, 0 to 729321
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   id                  729322 non-null  object
 1   vendor_id           729322 non-null  int64
 2   pickup_datetime     729322 non-null  datetime64[ns]
 3   dropoff_datetime    729322 non-null  object
 4   passenger_count     729322 non-null  int64
 5   pickup_longitude    729322 non-null  float64
 6   pickup_latitude     729322 non-null  float64
 7   dropoff_longitude   729322 non-null  float64
 8   dropoff_latitude    729322 non-null  float64
 9   store_and_fwd_flag  729322 non-null  object
 10  trip_duration       729322 non-null  int64
dtypes: datetime64[ns](1), float64(4), int64(3), object(3)
memory usage: 61.2+ MB
None
```

Feature Extraction

In [4]:
```
# Extracting time-based features from `pickup_datetime`
nyc_taxi_data['pickup_hour'] = nyc_taxi_data['pickup_datetime'].dt.hour
nyc_taxi_data['pickup_day'] = nyc_taxi_data['pickup_datetime'].dt.day
nyc_taxi_data['pickup_month'] = nyc_taxi_data['pickup_datetime'].dt.month
nyc_taxi_data['pickup_dayofweek'] = nyc_taxi_data['pickup_datetime'].dt.dayofweek
```

```
# Verify the new features
print(nyc_taxi_data[['pickup_hour', 'pickup_day', 'pickup_month', 'pickup_dayofweek']].head())
```

```
   pickup_hour  pickup_day  pickup_month  pickup_dayofweek
0           16          29             2                 0
1           23          11             3                 4
2           17          21             2                 6
3            9           5             1                 1
4            6          17             2                 2
```

Feature Engineering

In [38]:

```
import numpy as np
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar
from sklearn.cluster import KMeans

# Ensure pickup_datetime is in converted to datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])

# Feature 1: Trip Distance (using Haversine formula)
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth radius in kilometers
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])  # Convert degrees to radians
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

if 'trip_distance' not in nyc_taxi_data.columns:
    nyc_taxi_data['trip_distance'] = haversine(
        nyc_taxi_data['pickup_latitude'], nyc_taxi_data['pickup_longitude'],
        nyc_taxi_data['dropoff_latitude'], nyc_taxi_data['dropoff_longitude']
    )

# Feature 2: Average Speed (in km/h)
nyc_taxi_data['trip_duration_hours'] = nyc_taxi_data['trip_duration'] / 3600  # Convert duration to hours
nyc_taxi_data['average_speed_kmh'] = nyc_taxi_data['trip_distance'] / nyc_taxi_data['trip_duration_hours']

# Feature 3: Rush Hour (binary feature, 1 if pickup during rush hours)
nyc_taxi_data['pickup_hour'] = nyc_taxi_data['pickup_datetime'].dt.hour
```

```
nyc_taxi_data['is_rush_hour'] = nyc_taxi_data['pickup_hour'].apply(lambda x: 1 if (7 <= x <= 9) or (16 <= x <= 19) el

# Feature 4: Weekend (binary feature, 1 if pickup on Saturday or Sunday)
nyc_taxi_data['pickup_dayofweek'] = nyc_taxi_data['pickup_datetime'].dt.dayofweek
nyc_taxi_data['is_weekend'] = nyc_taxi_data['pickup_dayofweek'].apply(lambda x: 1 if x >= 5 else 0)

# Feature 5: Passenger Count and Distance Interaction
nyc_taxi_data['passenger_distance_interaction'] = nyc_taxi_data['passenger_count'] * nyc_taxi_data['trip_distance']

# Feature 6: Trip Duration Binning (categorize duration into quantiles)
nyc_taxi_data['duration_bin'] = pd.qcut(nyc_taxi_data['trip_duration'], q=4, labels=['short', 'medium', 'long', 'very

# Feature 7: Time-based Features (week, day, quarter)
nyc_taxi_data['pickup_week'] = nyc_taxi_data['pickup_datetime'].dt.isocalendar().week
nyc_taxi_data['pickup_day'] = nyc_taxi_data['pickup_datetime'].dt.day
nyc_taxi_data['pickup_quarter'] = nyc_taxi_data['pickup_datetime'].dt.quarter

# Feature 8: Time Since Midnight (in seconds)
nyc_taxi_data['seconds_since_midnight'] = (
    nyc_taxi_data['pickup_datetime'].dt.hour * 3600 +
    nyc_taxi_data['pickup_datetime'].dt.minute * 60 +
    nyc_taxi_data['pickup_datetime'].dt.second
)

# Feature 9: Holiday Feature (1 if pickup is on a holiday)
cal = USFederalHolidayCalendar()
holidays = cal.holidays(start=nyc_taxi_data['pickup_datetime'].min(), end=nyc_taxi_data['pickup_datetime'].max())
nyc_taxi_data['is_holiday'] = nyc_taxi_data['pickup_datetime'].dt.date.isin(holidays.date).astype(int)

# Feature 10: Peak Hour Flag (1 if pickup during peak times, e.g., late night weekends)
nyc_taxi_data['is_peak_hour'] = nyc_taxi_data['pickup_hour'].apply(lambda x: 1 if (20 <= x <= 23) or (x <= 2) else 0)

# Feature 11: Direction Feature (angle of travel in degrees)
nyc_taxi_data['direction_angle'] = np.degrees(np.arctan2(
    nyc_taxi_data['dropoff_longitude'] - nyc_taxi_data['pickup_longitude'],
    nyc_taxi_data['dropoff_latitude'] - nyc_taxi_data['pickup_latitude']
))

# Feature 12: Pickup/Dropoff Density Zones (using KMeans clustering for high-traffic zones)
kmeans = KMeans(n_clusters=10, random_state=42)  # Assuming 10 zones, adjust as necessary
nyc_taxi_data['pickup_zone'] = kmeans.fit_predict(nyc_taxi_data[['pickup_latitude', 'pickup_longitude']])
nyc_taxi_data['dropoff_zone'] = kmeans.fit_predict(nyc_taxi_data[['dropoff_latitude', 'dropoff_longitude']])
```

```
# Display the first few rows to verify the new features
print(nyc_taxi_data.head())
```

```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

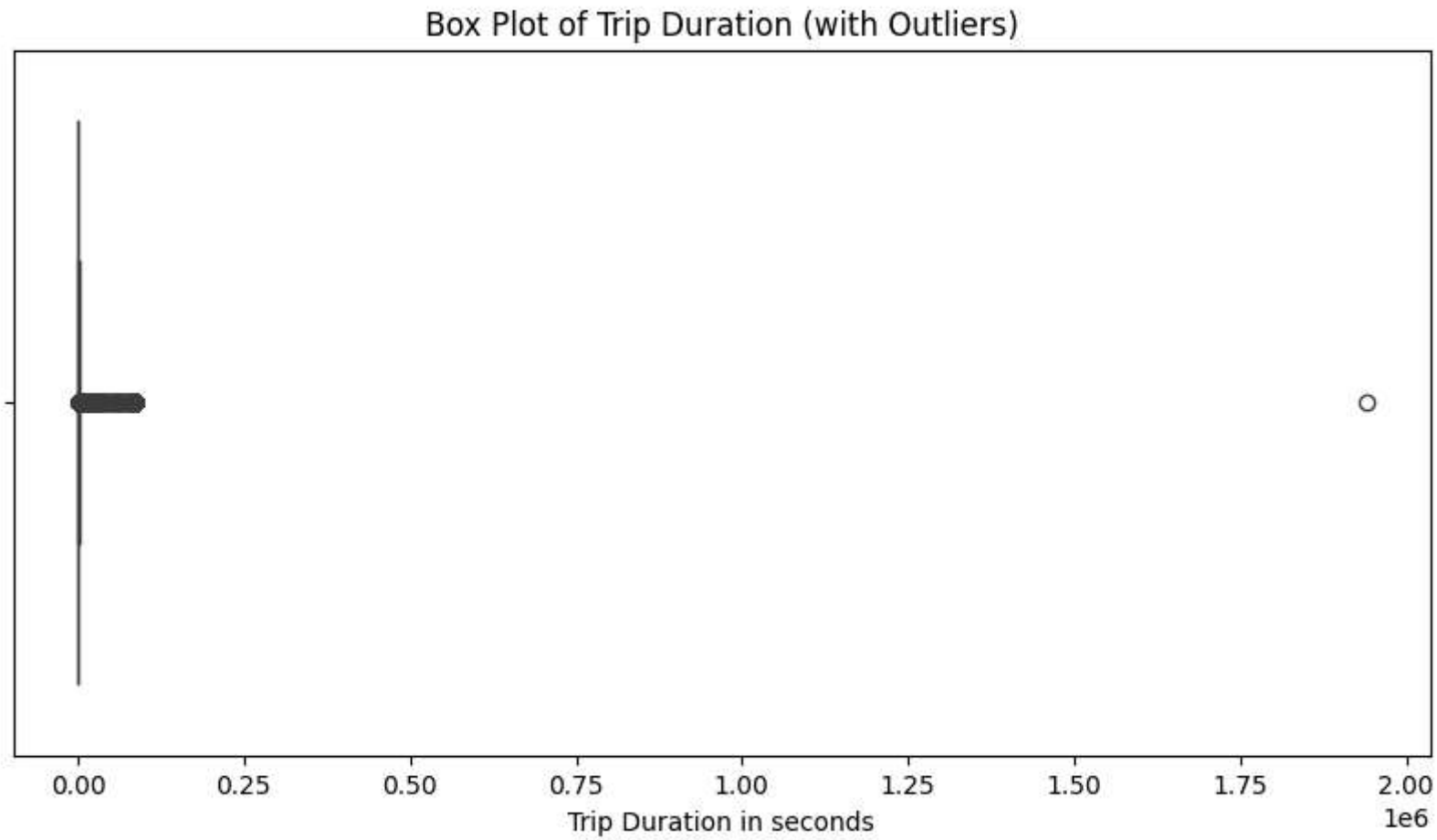
```
         id  vendor_id     pickup_datetime    dropoff_datetime  \
0  id1080784          2 2016-02-29 16:40:21 2016-02-29 16:47:01
1  id0889885          1 2016-03-11 23:35:37 2016-03-11 23:53:57
2  id0857912          2 2016-02-21 17:59:33 2016-02-21 18:26:48
3  id3744273          2 2016-01-05 09:44:31 2016-01-05 10:03:32
4  id0232939          1 2016-02-17 06:42:23 2016-02-17 06:56:31

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.953918        40.778873         -73.963875
1                2        -73.988312        40.731743         -73.994751
2                2        -73.997314        40.721458         -73.948029
3                6        -73.961670        40.759720         -73.956779
4                1        -74.017120        40.708469         -73.988182

   dropoff_latitude store_and_fwd_flag  ...  pickup_timeofday  duration_bin  \
0         40.771164                  N  ...         Afternoon        medium
1         40.694931                  N  ...        Late Night     very_long
2         40.774918                  N  ...           Evening     very_long
3         40.780628                  N  ...           Morning     very_long
4         40.740631                  N  ...           Morning          long

   pickup_week  pickup_quarter  seconds_since_midnight  is_holiday  \
0            9               1                   60021           0
1           10               1                   84937           0
2            7               1                   64773           0
3            1               1                   35071           0
4            7               1                   24143           0

   is_peak_hour  direction_angle  pickup_zone  dropoff_zone
0             0      -127.751675            0             0
1             1      -170.078075            2             7
2             0        42.674050            5             0
3             0        13.164751            9             0
4             0        41.980078            5             3

[5 rows x 34 columns]
```

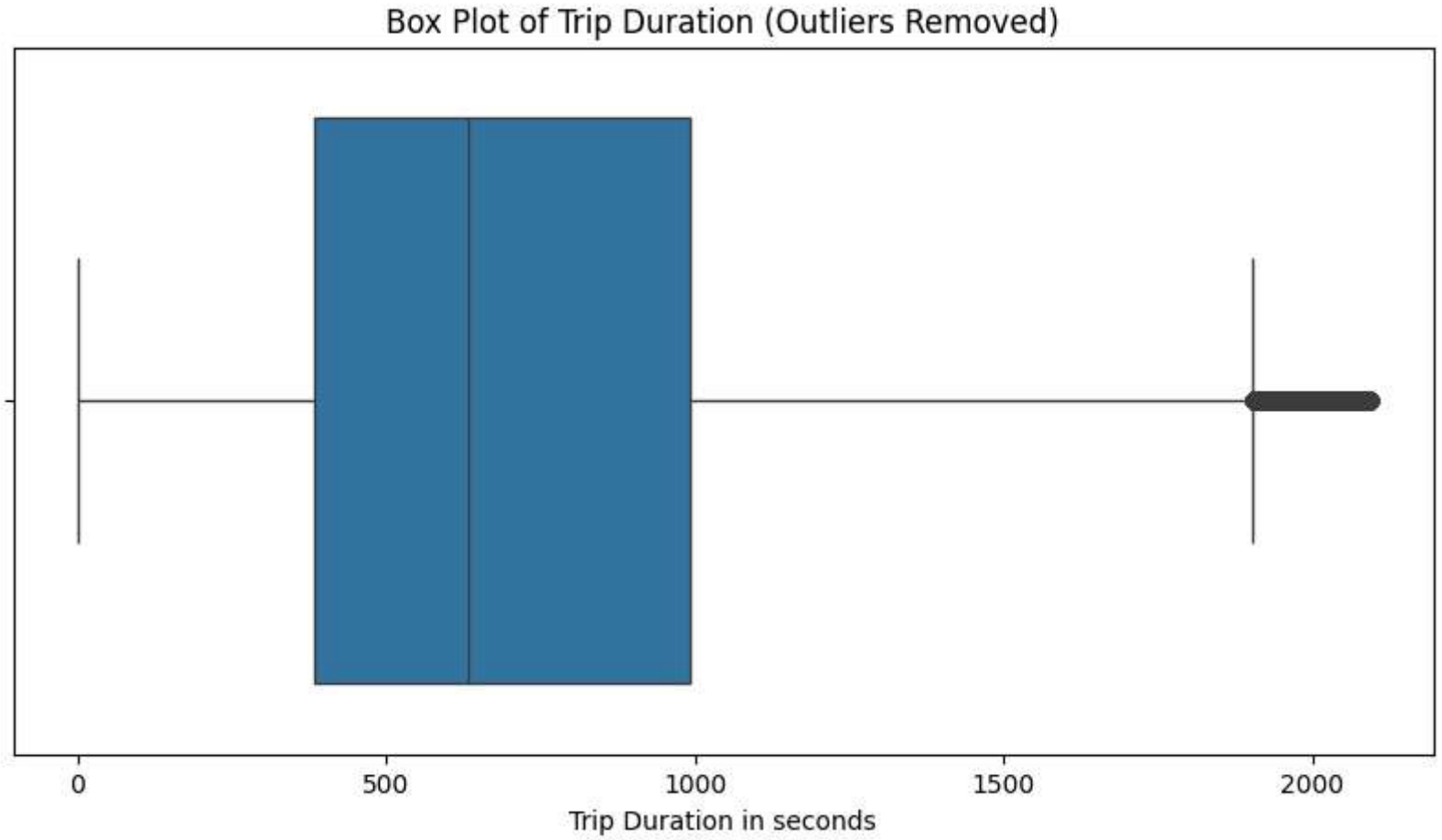
Outlier Detection

In [56]:
```
# Calculate IQR for trip_duration
Q1 = nyc_taxi_data['trip_duration'].quantile(0.25)
Q3 = nyc_taxi_data['trip_duration'].quantile(0.75)
```

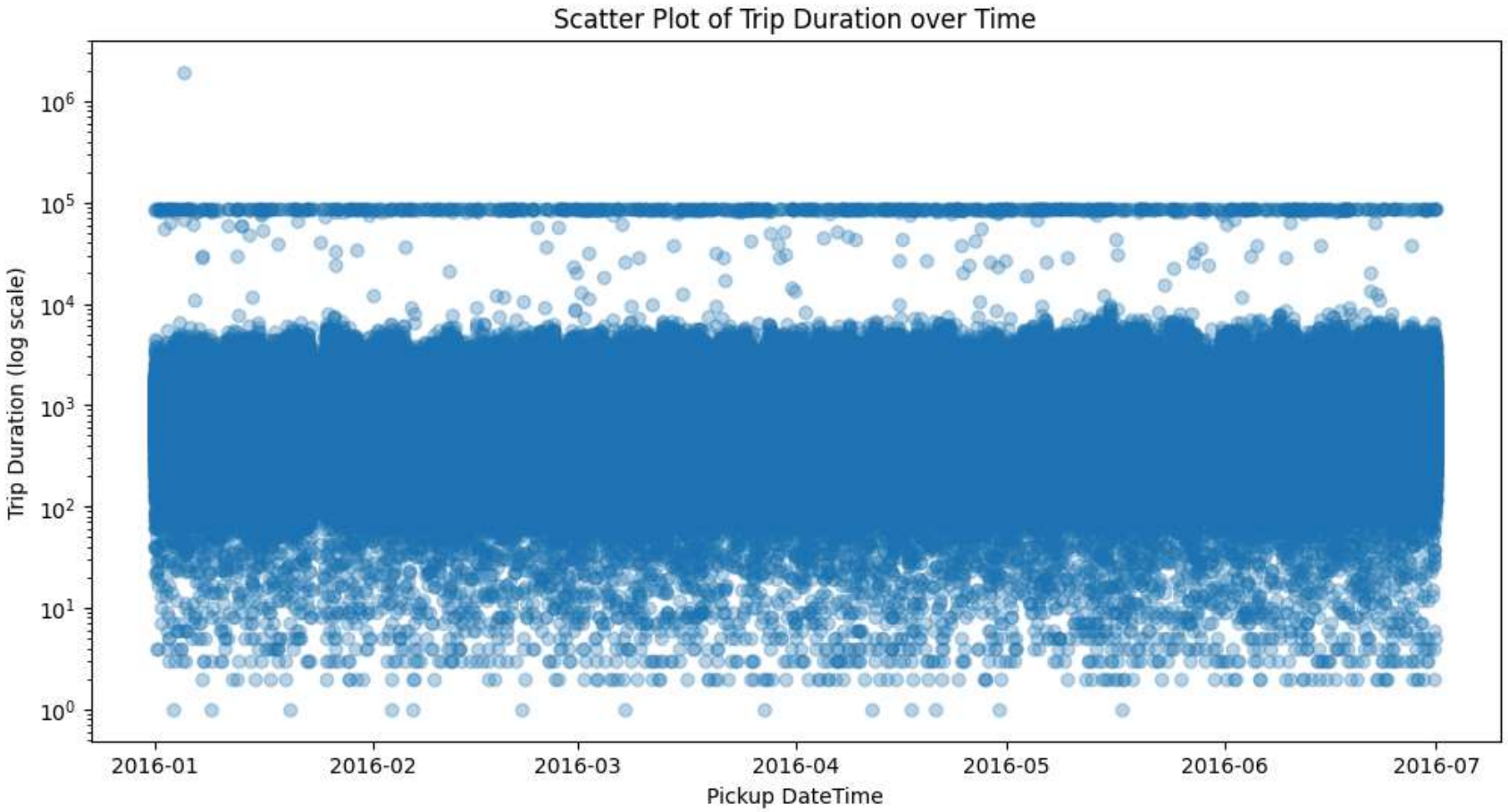
```
IQR = Q3 - Q1

# Define outlier thresholds
outlier_threshold_low = Q1 - 1.5 * IQR
outlier_threshold_high = Q3 + 1.5 * IQR

# Identify and count outliers
outliers = nyc_taxi_data[(nyc_taxi_data['trip_duration'] < outlier_threshold_low) |
                         (nyc_taxi_data['trip_duration'] > outlier_threshold_high)]
print("Number of outliers in trip_duration:", outliers.shape[0])

# Remove outliers by filtering the DataFrame
nyc_taxi_data_cleaned = nyc_taxi_data[(nyc_taxi_data['trip_duration'] >= outlier_threshold_low) &
                                      (nyc_taxi_data['trip_duration'] <= outlier_threshold_high)]

# Confirm the number of rows in the cleaned dataset
print("Number of rows after removing outliers:", nyc_taxi_data_cleaned.shape[0])

# Plot original data with outliers
plt.figure(figsize=(10, 5))
sns.boxplot(x=nyc_taxi_data['trip_duration'])
plt.title('Box Plot of Trip Duration (with Outliers)')
plt.xlabel('Trip Duration in seconds')
plt.show()

# Plot cleaned data without outliers
plt.figure(figsize=(10, 5))
sns.boxplot(x=nyc_taxi_data_cleaned['trip_duration'])
plt.title('Box Plot of Trip Duration (Outliers Removed)')
plt.xlabel('Trip Duration in seconds')
plt.show()
```

```
Number of outliers in trip_duration: 36963
Number of rows after removing outliers: 692359
```


Box Plot of Trip Duration (with Outliers)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
Trip Duration in seconds

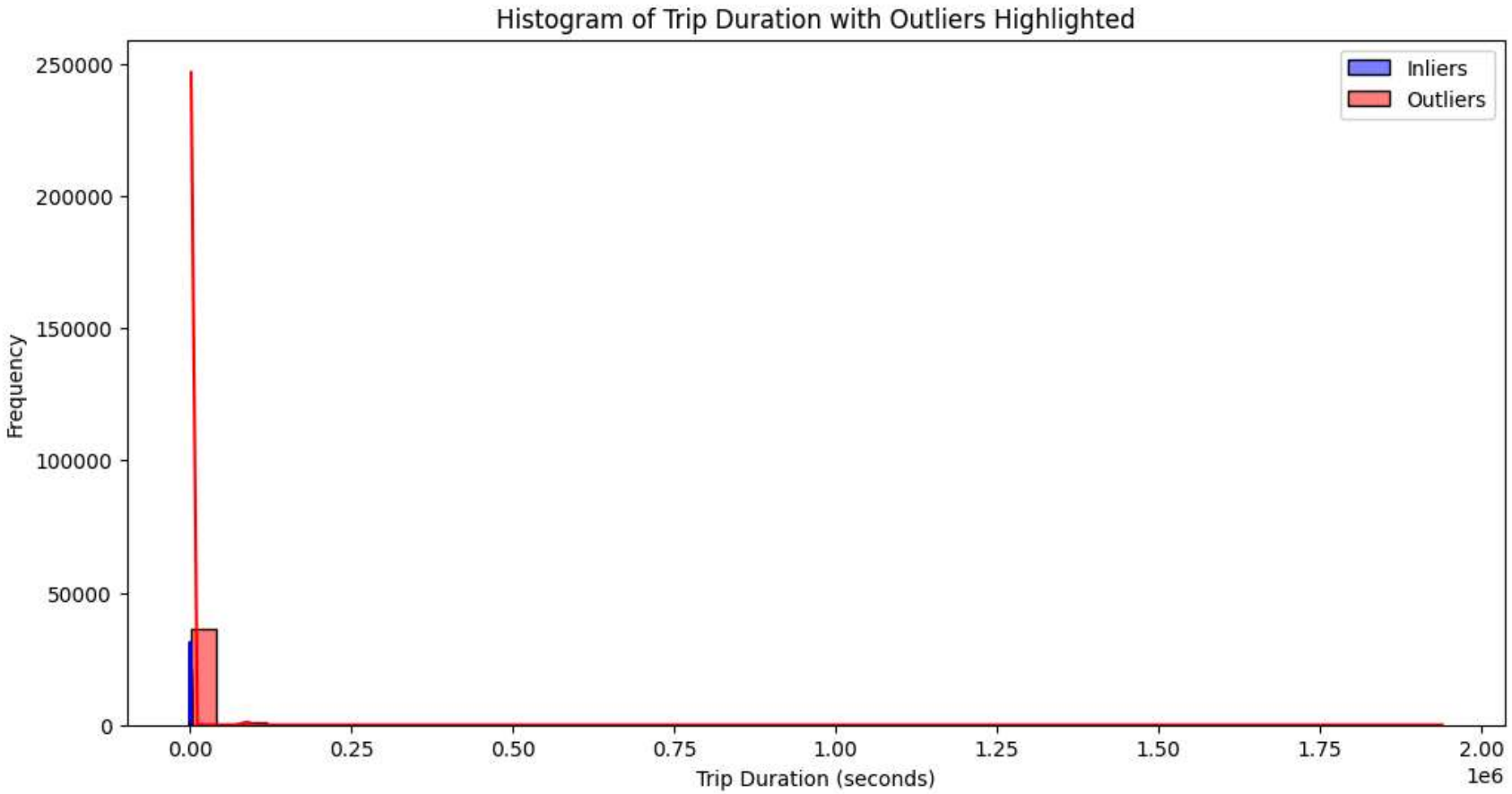


In [7]:
```
# Scatter plot of trip_duration over pickup_datetime to see trends over time
plt.figure(figsize=(12, 6))
plt.scatter(nyc_taxi_data['pickup_datetime'], nyc_taxi_data['trip_duration'], alpha=0.3)
plt.yscale('log')
plt.title('Scatter Plot of Trip Duration over Time')
plt.xlabel('Pickup DateTime')
plt.ylabel('Trip Duration (log scale)')
plt.show()
```



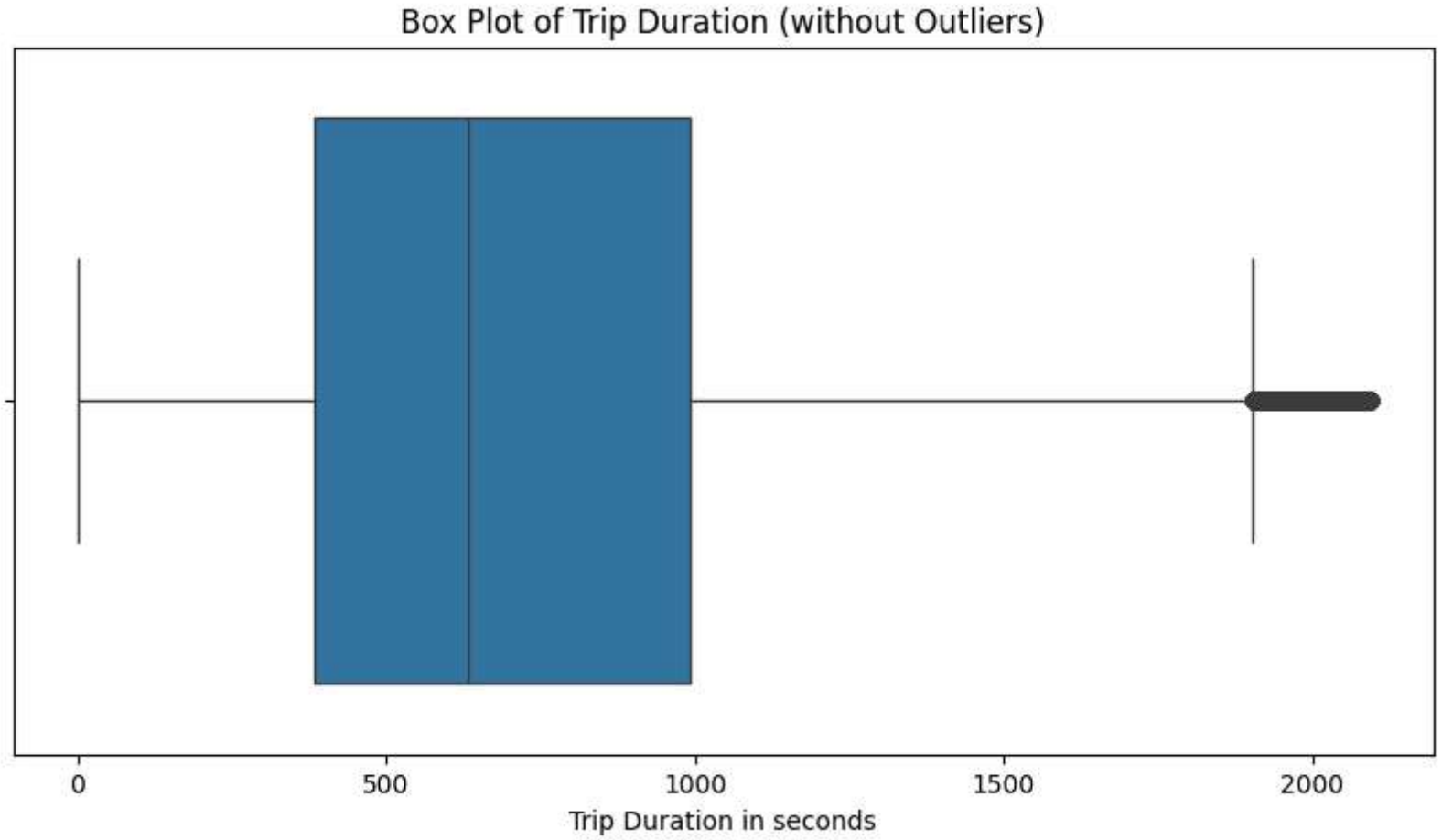
In [8]:
```
# Separate inliers and outliers based on the defined thresholds
inliers = nyc_taxi_data[(nyc_taxi_data['trip_duration'] >= outlier_threshold_low) &
                        (nyc_taxi_data['trip_duration'] <= outlier_threshold_high)]

# Histogram of inliers
plt.figure(figsize=(12, 6))
sns.histplot(inliers['trip_duration'], bins=50, color='blue', label='Inliers', kde=True)

# Histogram of outliers
sns.histplot(outliers['trip_duration'], bins=50, color='red', label='Outliers', kde=True)

plt.title('Histogram of Trip Duration with Outliers Highlighted')
plt.xlabel('Trip Duration (seconds)')
```

```
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



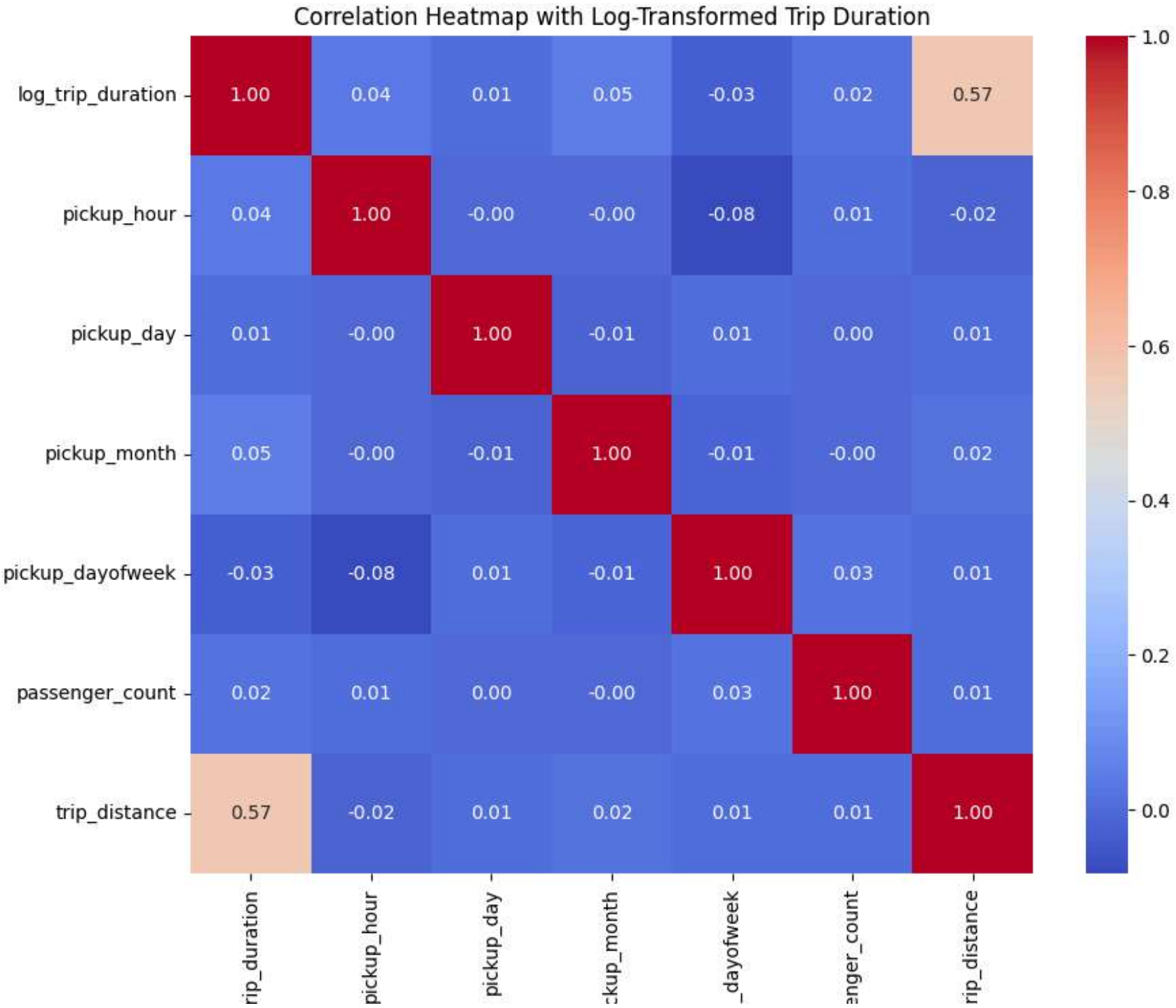
Data without Outliers

In [57]:
```
# Plotting the cleaned data with Box Plot (without outliers)
plt.figure(figsize=(10, 5))
sns.boxplot(x=nyc_taxi_data_cleaned['trip_duration'])
plt.title('Box Plot of Trip Duration (without Outliers)')
plt.xlabel('Trip Duration in seconds')
plt.show()
```



Data Through Correlation Matrix

In [10]:
```
# Log transform trip duration and add back to dataset
nyc_taxi_data['log_trip_duration'] = np.log(nyc_taxi_data['trip_duration'] + 1)

selected_columns = ['log_trip_duration', 'pickup_hour', 'pickup_day', 'pickup_month', 'pickup_dayofweek', 'passenger_
plt.figure(figsize=(10, 8))
sns.heatmap(nyc_taxi_data[selected_columns].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title('Correlation Heatmap with Log-Transformed Trip Duration')
plt.show()
```


Correlation Heatmap with Log-Transformed Trip Duration
log_trip_duration
pickup_hour
pickup_day
pickup_month
pickup_dayofweek
passenger_count
trip_distance
1.00 0.04 0.01 0.05 -0.03 0.02 0.57
0.04 1.00 -0.00 -0.00 -0.08 0.01 -0.02
0.01 -0.00 1.00 -0.01 0.01 0.00 0.01
0.05 -0.00 -0.01 1.00 -0.01 -0.00 0.02
-0.03 -0.08 0.01 -0.01 1.00 0.03 0.01
0.02 0.01 0.00 -0.00 0.03 1.00 0.01
0.57 -0.02 0.01 0.02 0.01 0.01 1.00
rip_duration
pickup_hour
pickup_day
ckup_month
_dayofweek
anger_count
rip_distance
1.0
0.8
0.6
0.4
0.2
0.0

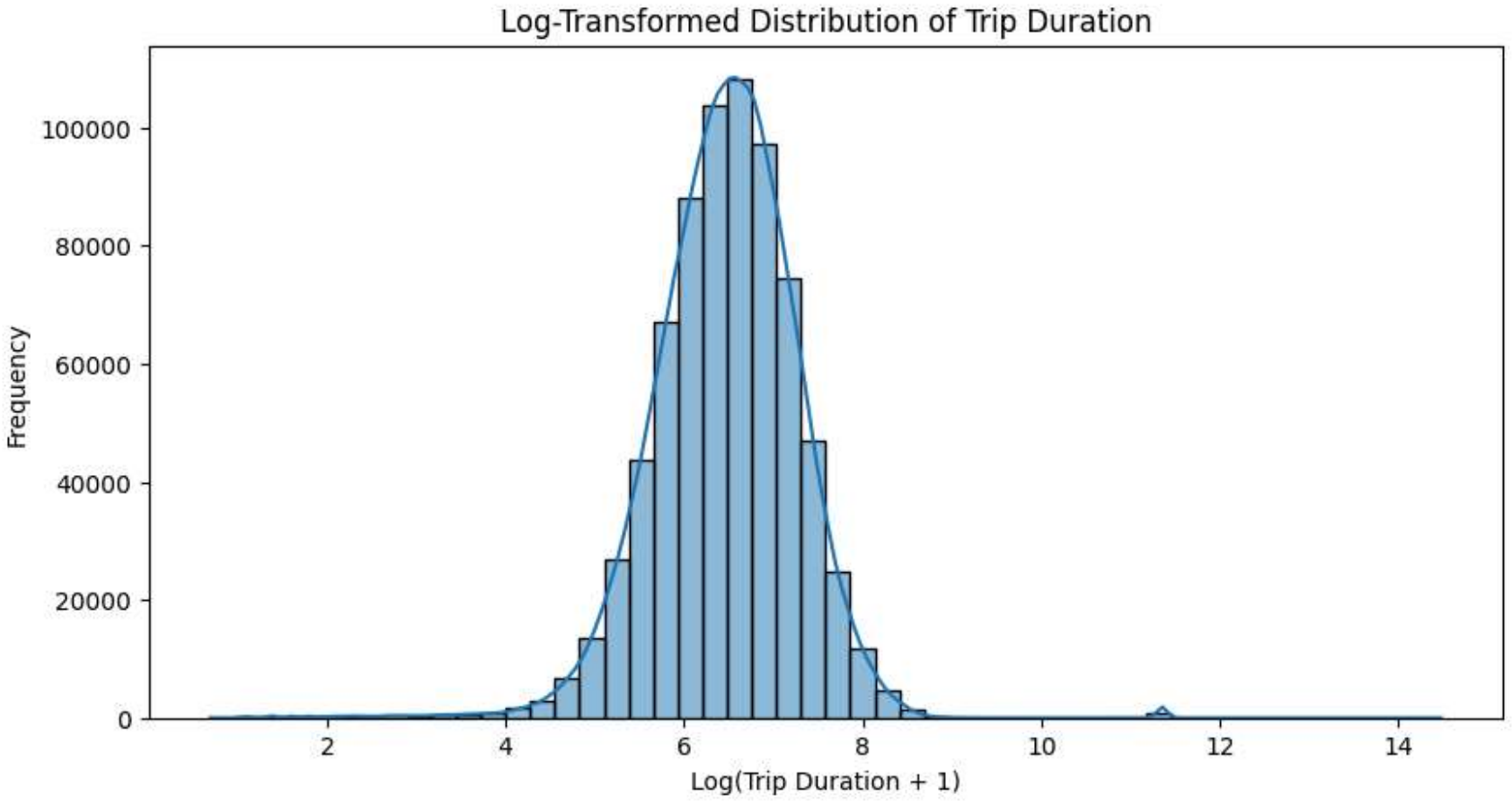
Univariate Analysis

In [11]:
```
!pip install geopy
```

```
Requirement already satisfied: geopy in c:\college\anaconda\envs\myenv\lib\site-packages (2.4.1)
Requirement already satisfied: geographiclib<3,>=1.52 in c:\college\anaconda\envs\myenv\lib\site-packages (from geop
y) (2.0)
```

In [12]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from geopy.distance import geodesic
```

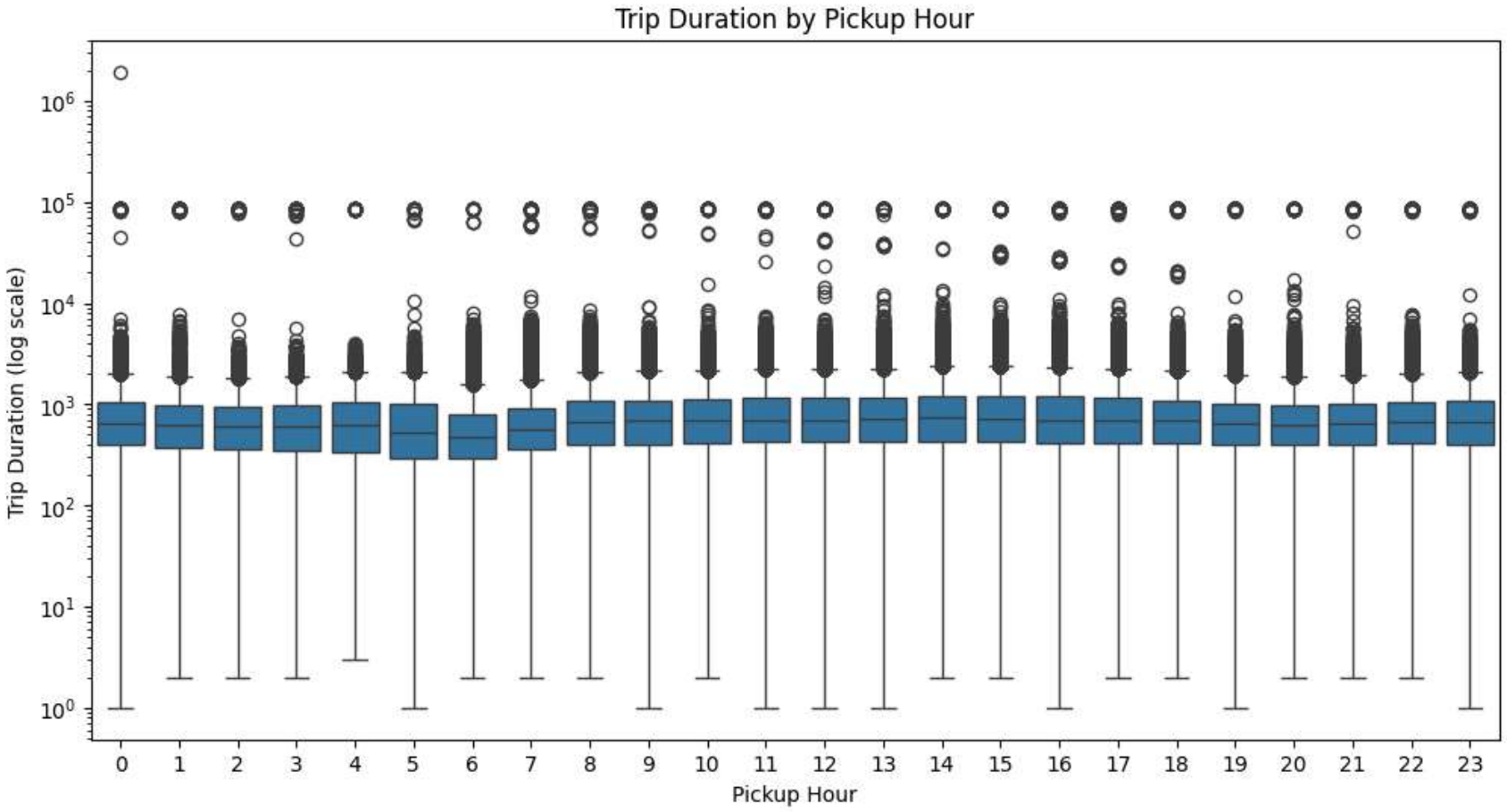
In [13]:
```
# Log Transformation of Trip Duration for Visualization
plt.figure(figsize=(10, 5))
sns.histplot(np.log1p(nyc_taxi_data['trip_duration']), bins=50, kde=True)
plt.title('Log-Transformed Distribution of Trip Duration')
plt.xlabel('Log(Trip Duration + 1)')
plt.ylabel('Frequency')
plt.show()
```



In [14]:
```
# Ensuring pickup_datetime is in datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])

# Extracting the hour from pickup_datetime
nyc_taxi_data['pickup_hour'] = nyc_taxi_data['pickup_datetime'].dt.hour
```

In [15]:
```
# Analysis of Trip Duration by Pickup Hour
plt.figure(figsize=(12, 6))
sns.boxplot(x='pickup_hour', y='trip_duration', data=nyc_taxi_data)
plt.yscale('log')
plt.title('Trip Duration by Pickup Hour')
plt.xlabel('Pickup Hour')
```

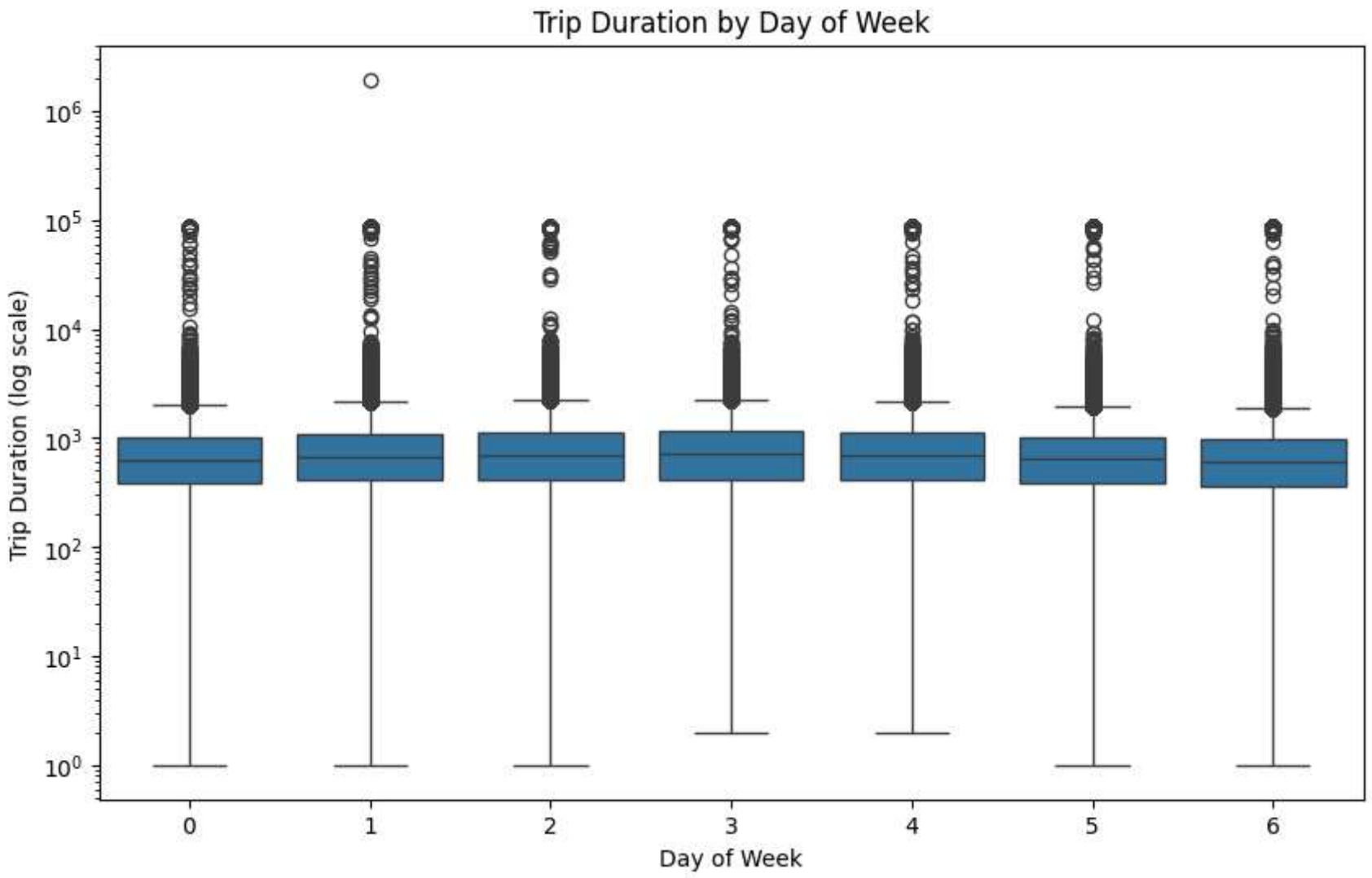
```
plt.ylabel('Trip Duration (log scale)')
plt.show()
```



In [16]:
```
# Ensure pickup_datetime is in datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])

# Extracting  the day of the week (0=Monday, 6=Sunday)
nyc_taxi_data['pickup_dayofweek'] = nyc_taxi_data['pickup_datetime'].dt.dayofweek
```

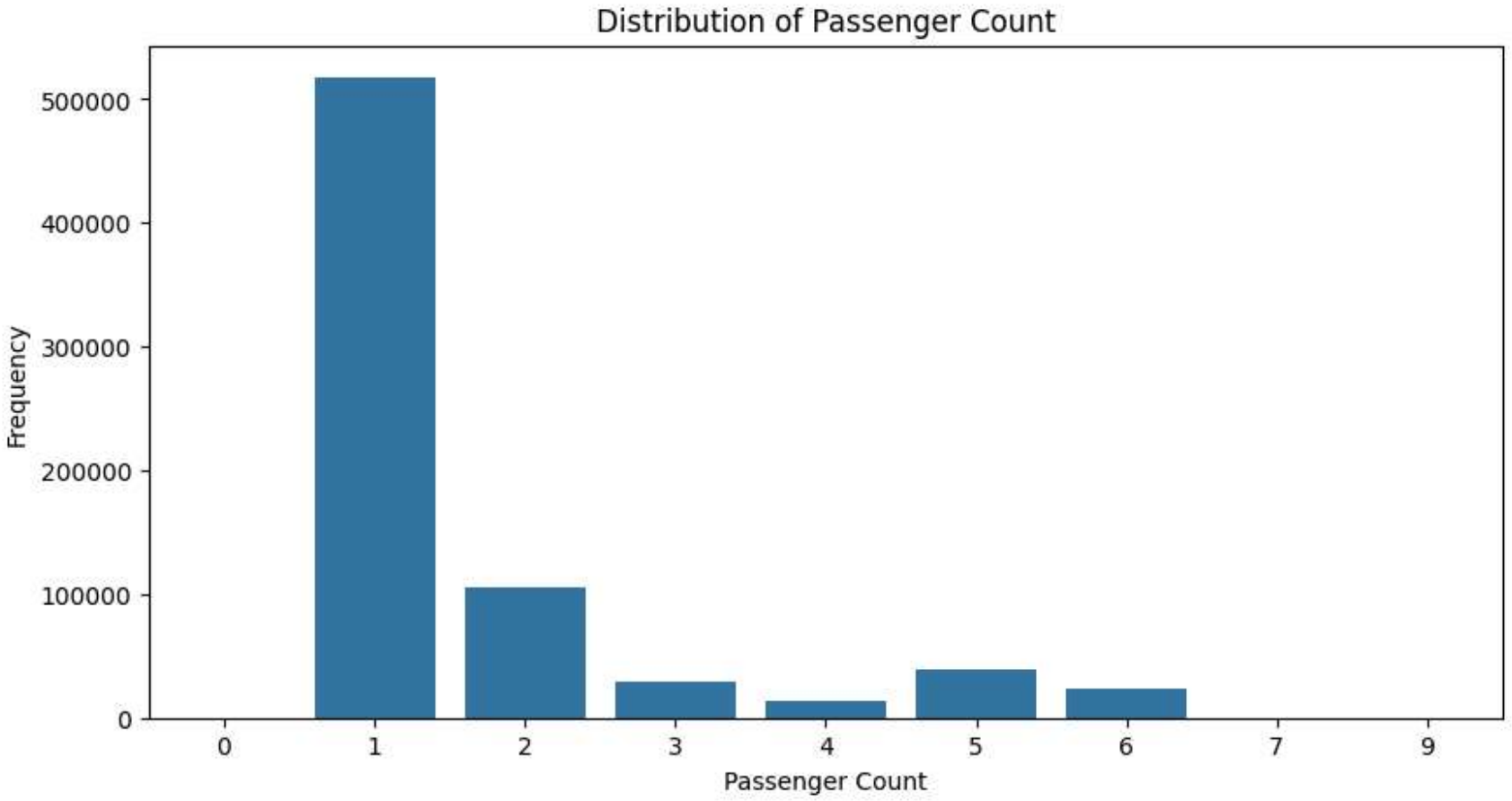
In [17]:
```
# Analysis of Trip Duration by Day of Week
plt.figure(figsize=(10, 6))
sns.boxplot(x='pickup_dayofweek', y='trip_duration', data=nyc_taxi_data)
plt.yscale('log')
```

```
plt.title('Trip Duration by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Trip Duration (log scale)')
plt.show()
```



In [18]:
```
# Passenger Count Analysis
plt.figure(figsize=(10, 5))
sns.countplot(x='passenger_count', data=nyc_taxi_data)
plt.title('Distribution of Passenger Count')
```

```
plt.xlabel('Passenger Count')
plt.ylabel('Frequency')
plt.show()
```



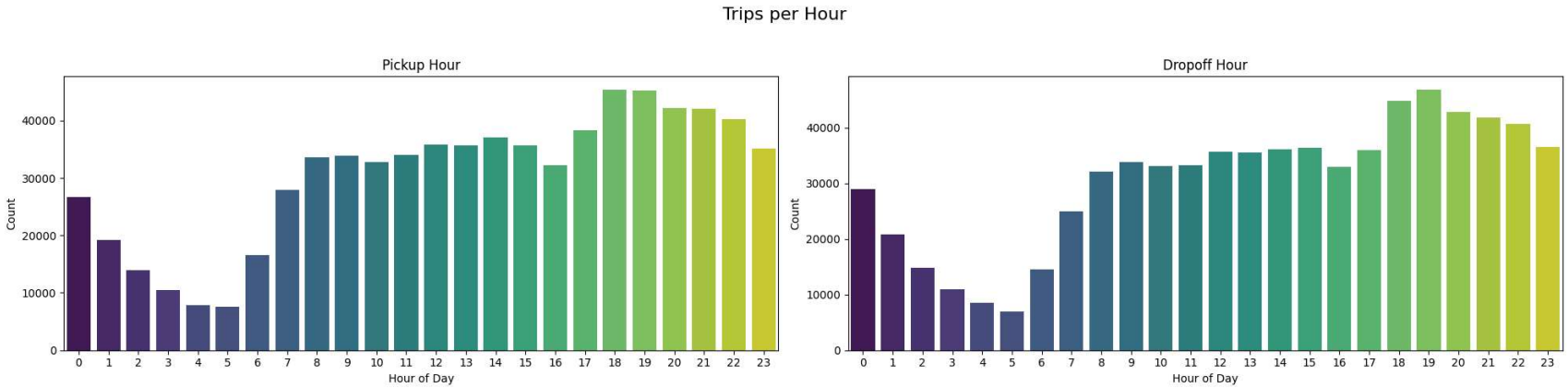
In [19]:
```
import numpy as np

# Define function to calculate distance using Haversine formula
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of Earth in kilometers
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])  # Convert degrees to radians
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
```

```
    return R * c

# Applying the haversine formula in a vectorized way to calculate trip_distance
nyc_taxi_data['trip_distance'] = haversine(
    nyc_taxi_data['pickup_latitude'], nyc_taxi_data['pickup_longitude'],
    nyc_taxi_data['dropoff_latitude'], nyc_taxi_data['dropoff_longitude']
)
```

Analysis for demostarting the number of trips per hour

In [20]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Ensure 'pickup_datetime' and 'dropoff_datetime' are in datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])
nyc_taxi_data['dropoff_datetime'] = pd.to_datetime(nyc_taxi_data['dropoff_datetime'])

# Extract hours from 'pickup_datetime' and 'dropoff_datetime'
nyc_taxi_data['pickup_hour'] = nyc_taxi_data['pickup_datetime'].dt.hour
nyc_taxi_data['dropoff_hour'] = nyc_taxi_data['dropoff_datetime'].dt.hour

# Set up the subplots
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20, 5))

# Plot Pickup Hour Distribution
sns.countplot(x='pickup_hour', data=nyc_taxi_data, ax=ax1, palette="viridis")
ax1.set_title('Pickup Hour')
ax1.set_xlabel('Hour of Day')
ax1.set_ylabel('Count')

# Plot Dropoff Hour Distribution
sns.countplot(x='dropoff_hour', data=nyc_taxi_data, ax=ax2, palette="viridis")
ax2.set_title('Dropoff Hour')
ax2.set_xlabel('Hour of Day')
ax2.set_ylabel('Count')

# Display the plots
plt.suptitle("Trips per Hour", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit title
plt.show()
```

```
C:\Users\ashit\AppData\Local\Temp\ipykernel_42764\1091386941.py:17: FutureWarning:

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

  sns.countplot(x='pickup_hour', data=nyc_taxi_data, ax=ax1, palette="viridis")
C:\Users\ashit\AppData\Local\Temp\ipykernel_42764\1091386941.py:23: FutureWarning:

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

  sns.countplot(x='dropoff_hour', data=nyc_taxi_data, ax=ax2, palette="viridis")
```



Analysis for demostarting the number of trips per day

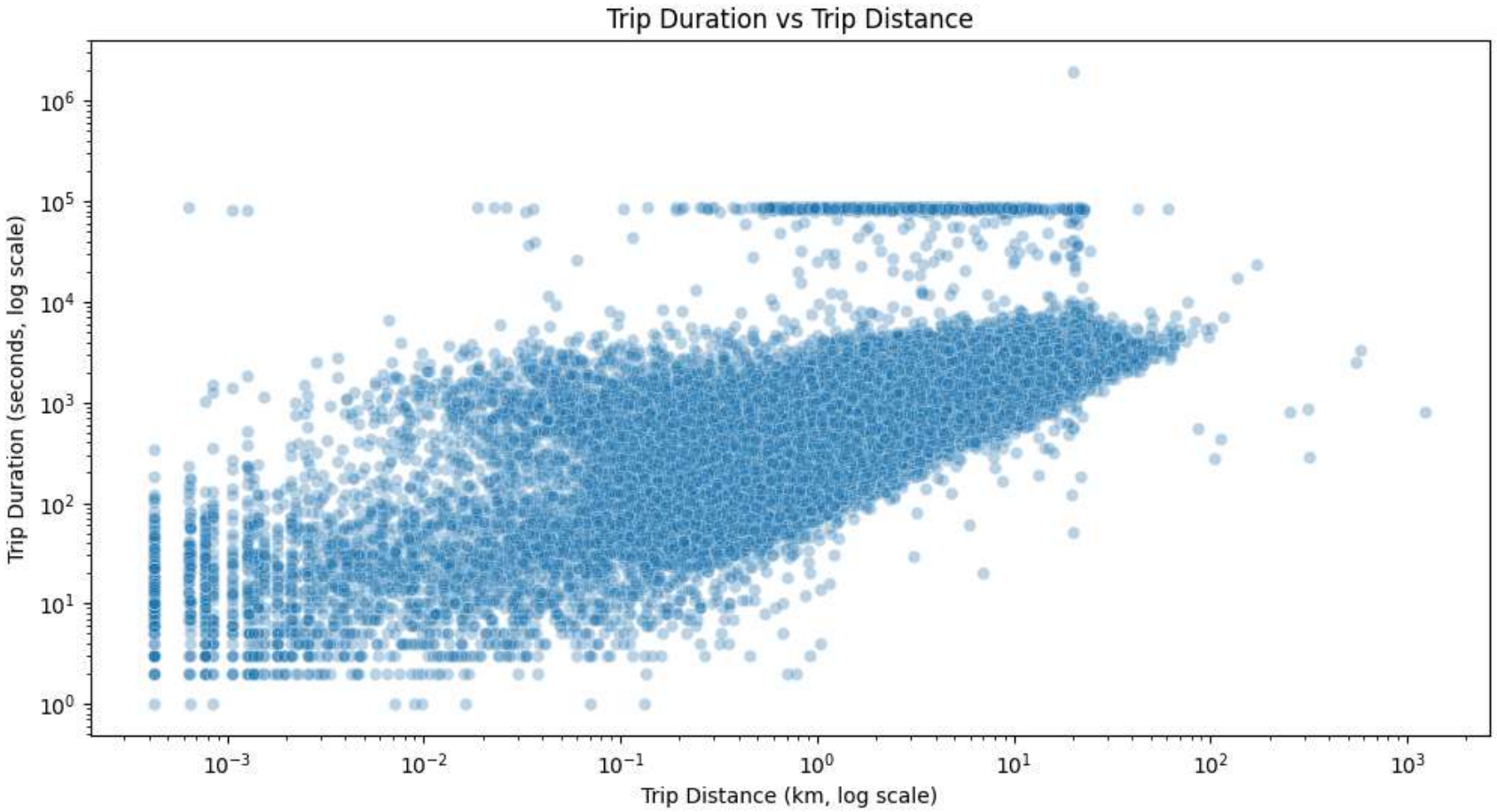
In [21]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Ensure 'pickup_datetime' and 'dropoff_datetime' are in datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])
nyc_taxi_data['dropoff_datetime'] = pd.to_datetime(nyc_taxi_data['dropoff_datetime'])

# Extract the day of the week (0=Monday, 6=Sunday)
nyc_taxi_data['pickup_day'] = nyc_taxi_data['pickup_datetime'].dt.day_name()
nyc_taxi_data['dropoff_day'] = nyc_taxi_data['dropoff_datetime'].dt.day_name()

# Set up the subplots
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20, 5))
```

```
# Plot Pickup Day Distribution
sns.countplot(x='pickup_day', data=nyc_taxi_data, ax=ax1, order=["Monday", "Tuesday", "Wednesday", "Thursday", "Frida
ax1.set_title('Pickup Days')
ax1.set_xlabel('Day of Week')
ax1.set_ylabel('Count')

# Plot Dropoff Day Distribution
sns.countplot(x='dropoff_day', data=nyc_taxi_data, ax=ax2, order=["Monday", "Tuesday", "Wednesday", "Thursday", "Fric
ax2.set_title('Dropoff Days')
ax2.set_xlabel('Day of Week')
ax2.set_ylabel('Count')

# Display the plots
plt.suptitle("Trips per Day", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit title
plt.show()
```

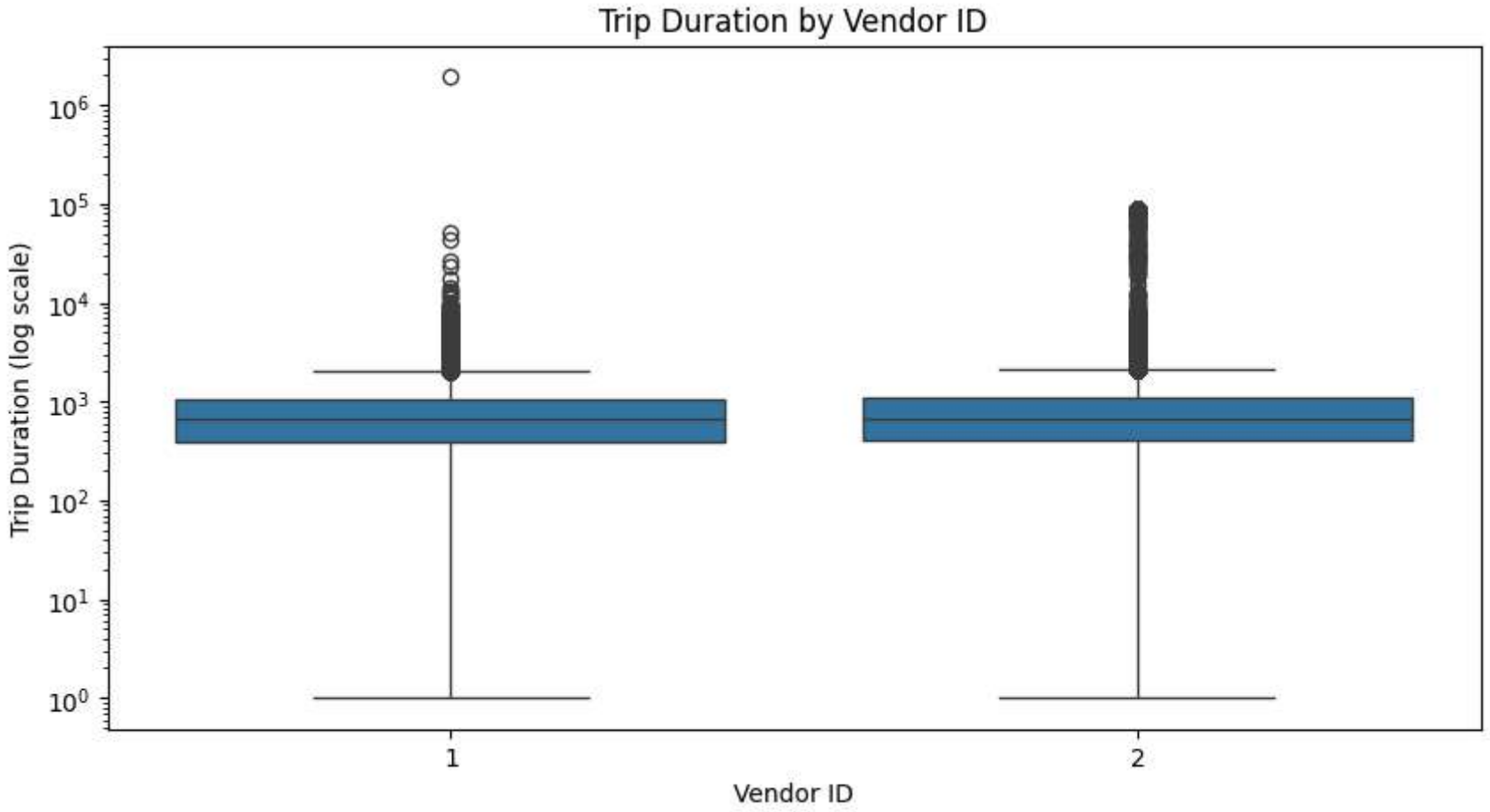
```
C:\Users\ashit\AppData\Local\Temp\ipykernel_42764\3411938144.py:17: FutureWarning:

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

  sns.countplot(x='pickup_day', data=nyc_taxi_data, ax=ax1, order=["Monday", "Tuesday", "Wednesday", "Thursday", "Fri
day", "Saturday", "Sunday"], palette="viridis")
C:\Users\ashit\AppData\Local\Temp\ipykernel_42764\3411938144.py:23: FutureWarning:

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

  sns.countplot(x='dropoff_day', data=nyc_taxi_data, ax=ax2, order=["Monday", "Tuesday", "Wednesday", "Thursday", "Fr
iday", "Saturday", "Sunday"], palette="viridis")
```

Bivariate Analysis

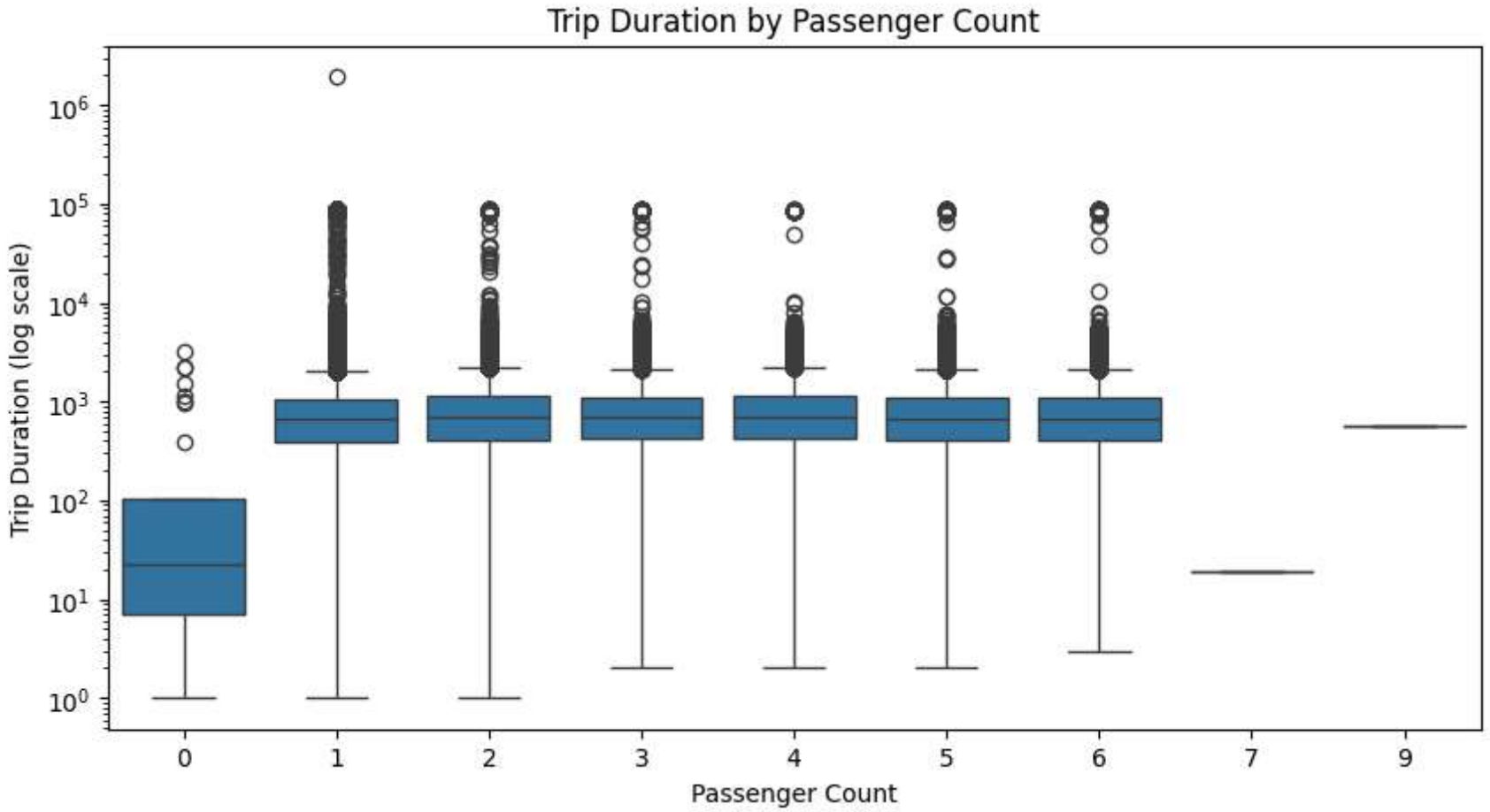
In [22]:
```
# Bivariate Analysis: Trip Duration vs Trip Distance
plt.figure(figsize=(12, 6))
sns.scatterplot(x='trip_distance', y='trip_duration', data=nyc_taxi_data, alpha=0.3)
plt.yscale('log')
plt.xscale('log')
plt.title('Trip Duration vs Trip Distance')
plt.xlabel('Trip Distance (km, log scale)')
plt.ylabel('Trip Duration (seconds, log scale)')
plt.show()
```



In [23]:
```
# Trip Duration by Vendor ID
plt.figure(figsize=(10, 5))
sns.boxplot(x='vendor_id', y='trip_duration', data=nyc_taxi_data)
plt.yscale('log')
plt.title('Trip Duration by Vendor ID')
plt.xlabel('Vendor ID')
plt.ylabel('Trip Duration (log scale)')
plt.show()
```

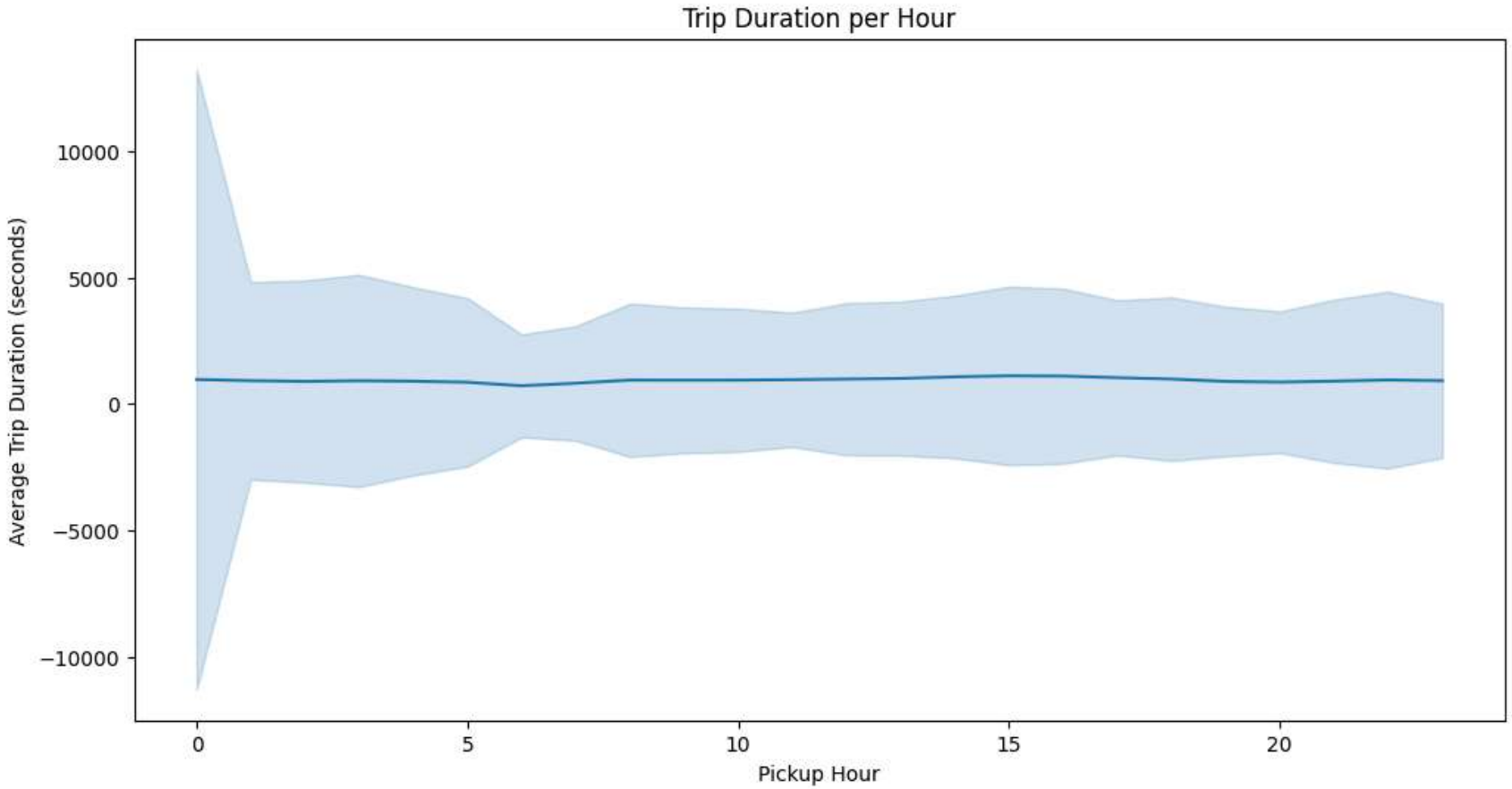


In [24]:
```
# Trip Duration by Passenger Count
plt.figure(figsize=(10, 5))
sns.boxplot(x='passenger_count', y='trip_duration', data=nyc_taxi_data)
plt.yscale('log')
plt.title('Trip Duration by Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Trip Duration (log scale)')
plt.show()
```



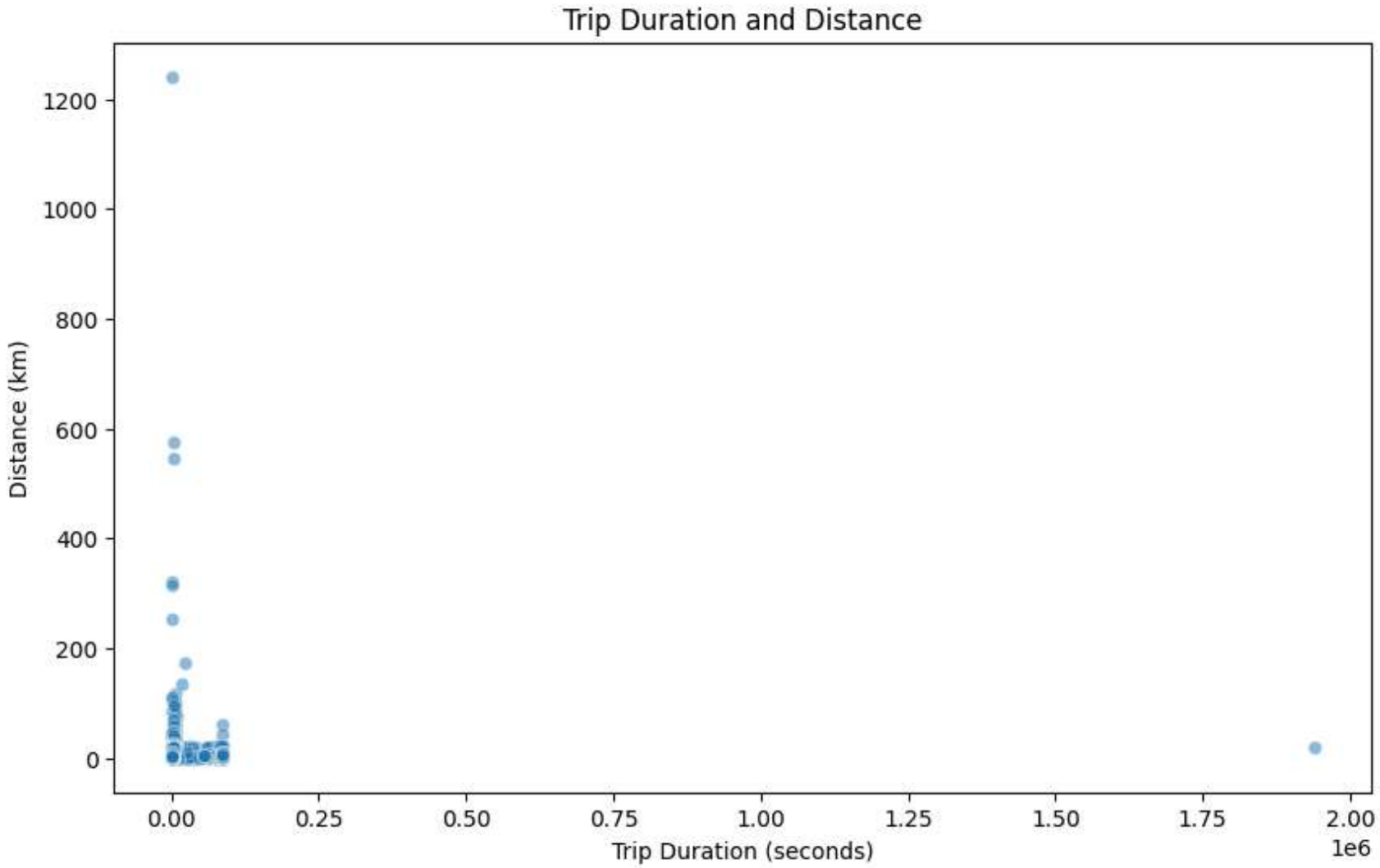
In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Line plot of Trip Duration by Pickup Hour
plt.figure(figsize=(12, 6))
sns.lineplot(x='pickup_hour', y='trip_duration', data=nyc_taxi_data, ci="sd")
plt.title('Trip Duration per Hour')
plt.xlabel('Pickup Hour')
plt.ylabel('Average Trip Duration (seconds)')
plt.show()
```

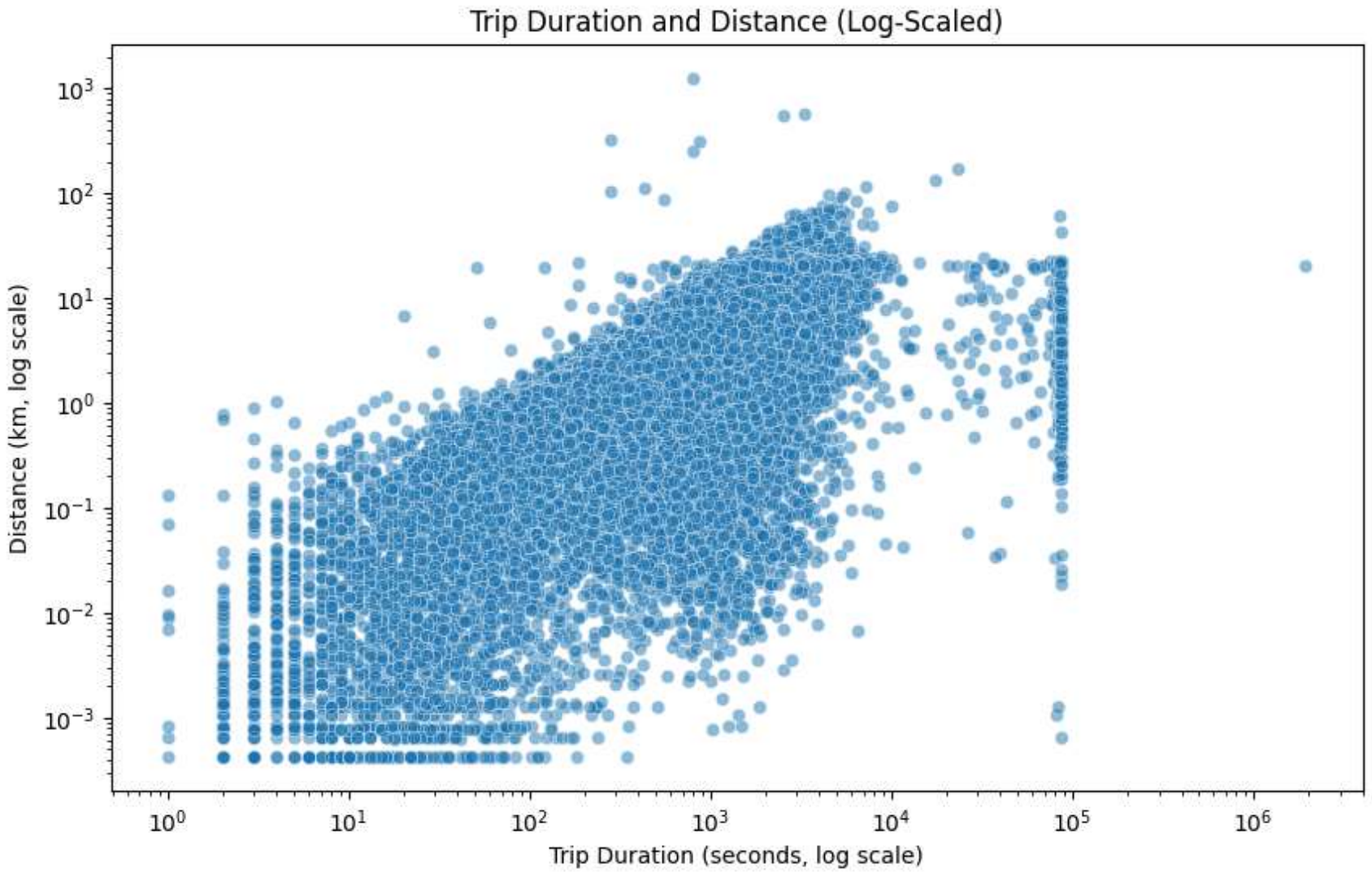
```
C:\Users\ashit\AppData\Local\Temp\ipykernel_42764\386071801.py:6: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  sns.lineplot(x='pickup_hour', y='trip_duration', data=nyc_taxi_data, ci="sd")
```



In [27]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Scatter plot of Trip Duration and Distance
plt.figure(figsize=(10, 6))
sns.scatterplot(x='trip_duration', y='trip_distance', data=nyc_taxi_data, alpha=0.5)
plt.title('Trip Duration and Distance')
plt.xlabel('Trip Duration (seconds)')
plt.ylabel('Distance (km)')
```

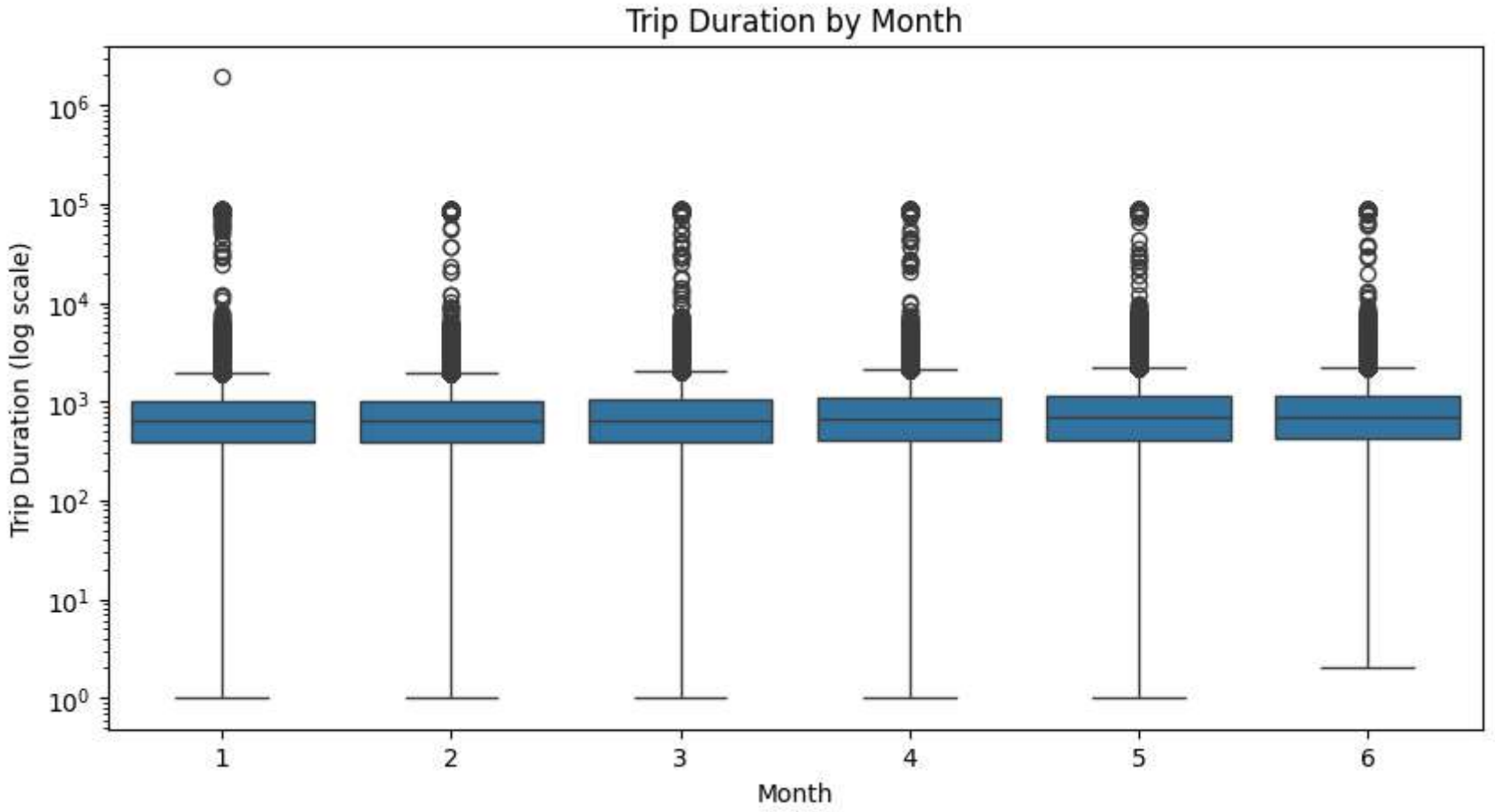
```
plt.show()
# Scatter plot with log-scaled axes
plt.figure(figsize=(10, 6))
sns.scatterplot(x='trip_duration', y='trip_distance', data=nyc_taxi_data, alpha=0.5)
plt.xscale('log')
plt.yscale('log')
plt.title('Trip Duration and Distance (Log-Scaled)')
plt.xlabel('Trip Duration (seconds, log scale)')
plt.ylabel('Distance (km, log scale)')
plt.show()
```


Trip Duration and Distance
Distance (km)
1200
1000
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Trip Duration (seconds)
1e6



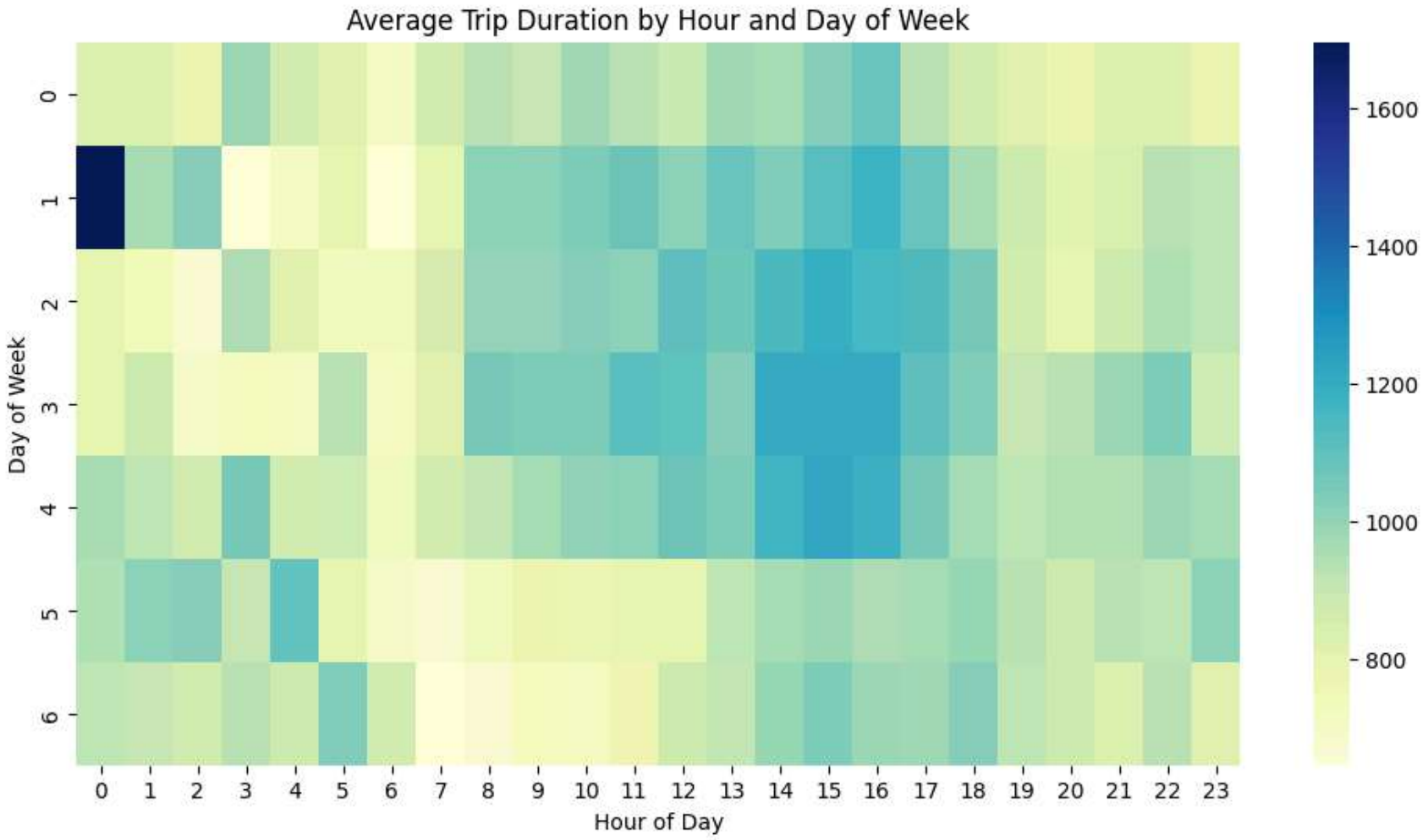
In [28]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 1. Trip Duration by Month (if available)
if 'pickup_month' in nyc_taxi_data.columns:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x='pickup_month', y='trip_duration', data=nyc_taxi_data)
```

```
    plt.yscale('log')
    plt.title('Trip Duration by Month')
    plt.xlabel('Month')
    plt.ylabel('Trip Duration (log scale)')
    plt.show()
```



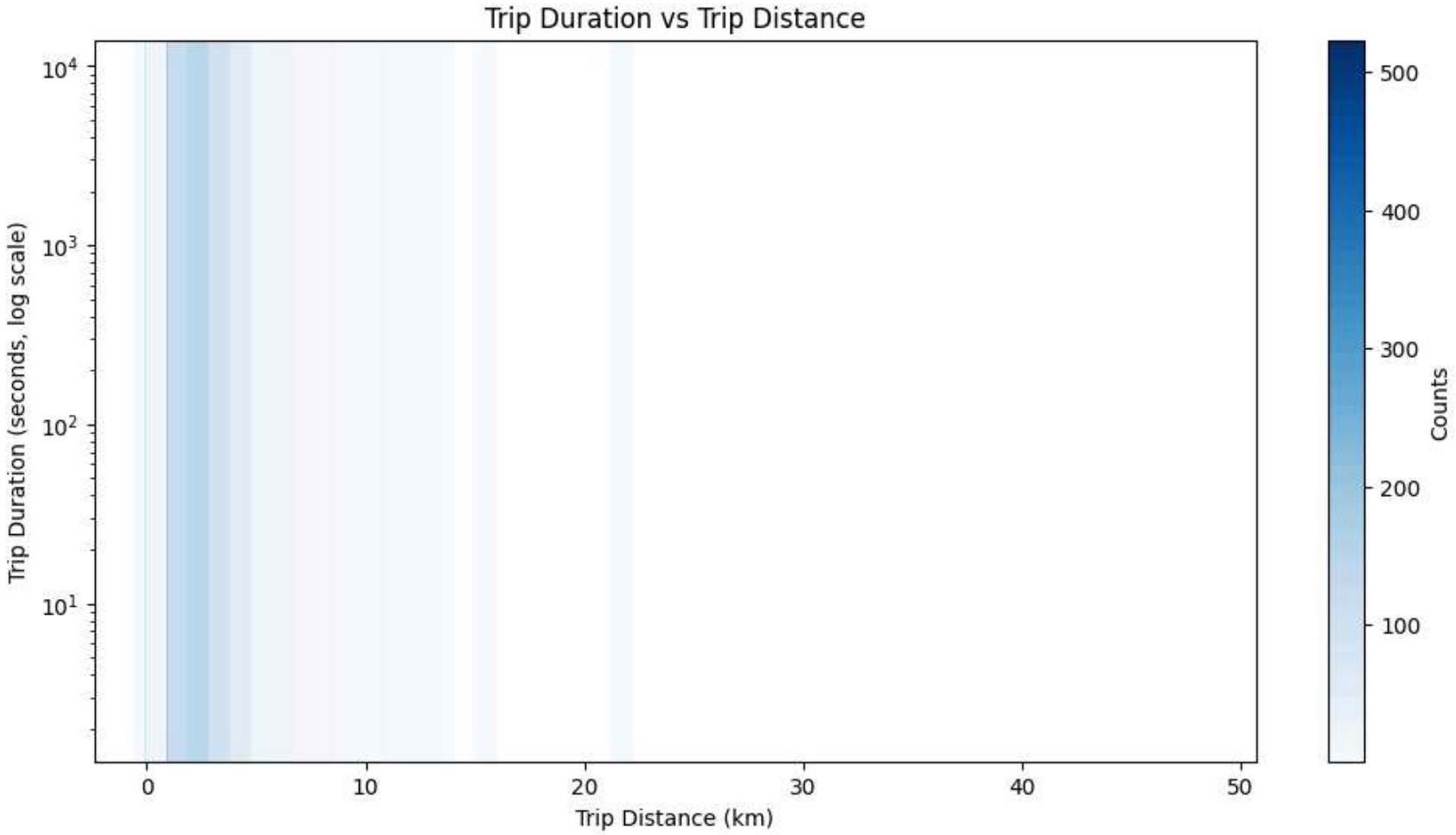
In [30]:
```
# 3. Heatmap of Average Trip Duration by Hour and Day of Week
pivot_table = nyc_taxi_data.pivot_table(values='trip_duration', index='pickup_dayofweek', columns='pickup_hour', agg
plt.figure(figsize=(12, 6))
sns.heatmap(pivot_table, cmap="YlGnBu", annot=False)
plt.title('Average Trip Duration by Hour and Day of Week')
plt.xlabel('Hour of Day')
```

```
plt.ylabel('Day of Week')
plt.show()
```



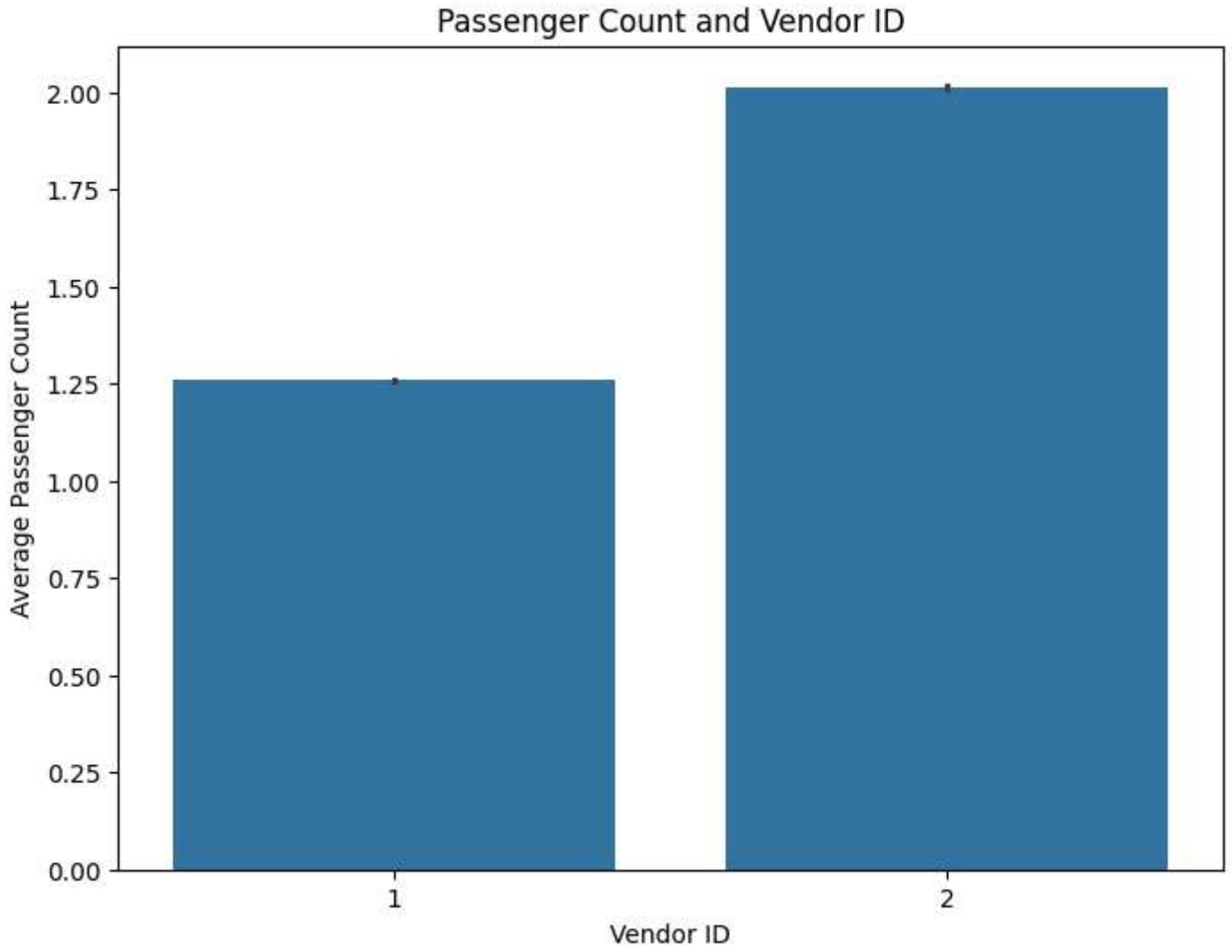
In [31]:
```
# Sample the dataset for faster plotting (e.g., 5000 points)
sample_data = nyc_taxi_data.sample(n=5000, random_state=42)  # Adjust sample size if needed

# Cap trip duration and distance to remove extreme outliers (optional)
sample_data = sample_data[(sample_data['trip_duration'] < 3600 * 5) & (sample_data['trip_distance'] < 50)]

# Hexbin Plot of Trip Duration vs Trip Distance with Log Scale on Y-axis Only
plt.figure(figsize=(12, 6))
```

```
plt.hexbin(sample_data['trip_distance'], sample_data['trip_duration'], gridsize=50, cmap='Blues', mincnt=1)
plt.yscale('log')
plt.colorbar(label='Counts')
plt.title('Trip Duration vs Trip Distance')
plt.xlabel('Trip Distance (km)')
plt.ylabel('Trip Duration (seconds, log scale)')
plt.show()
```



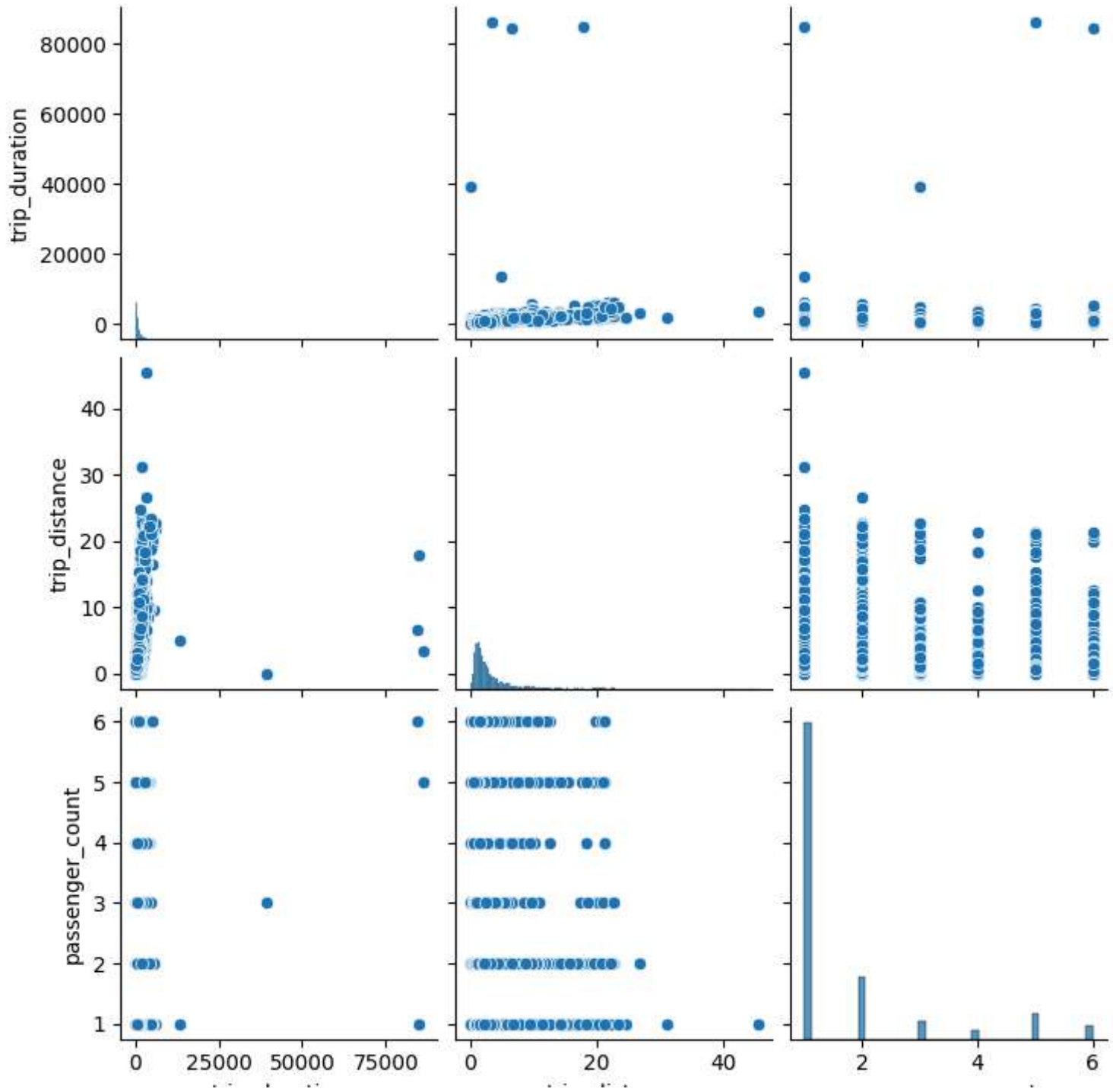
In [32]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Bar plot of Passenger Count by Vendor ID
```

```
plt.figure(figsize=(8, 6))
sns.barplot(x='vendor_id', y='passenger_count', data=nyc_taxi_data)
plt.title('Passenger Count and Vendor ID')
plt.xlabel('Vendor ID')
plt.ylabel('Average Passenger Count')
plt.show()
```



In [33]:
```
# 5. Pair Plot of Key Numerical Variables
sns.pairplot(nyc_taxi_data[['trip_duration', 'trip_distance', 'passenger_count']].sample(5000))  # Limit to 5000 samp
```

```
plt.show()
```


trip_duration
80000
60000
40000
20000
0
trip_distance
40
30
20
10
0
passenger_count
6
5
4
3
2
1
0
25000
50000
75000
0
20
40
2
4
6

trip_duration trip_distance passenger_count

Building and Implementing the Machine Learning Model

In [34]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\college\anaconda\envs\myenv\lib\site-packages (2.1.2)
Requirement already satisfied: numpy in c:\college\anaconda\envs\myenv\lib\site-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in c:\college\anaconda\envs\myenv\lib\site-packages (from xgboost) (1.10.1)
```

In [ ]:
```
Model Training
```

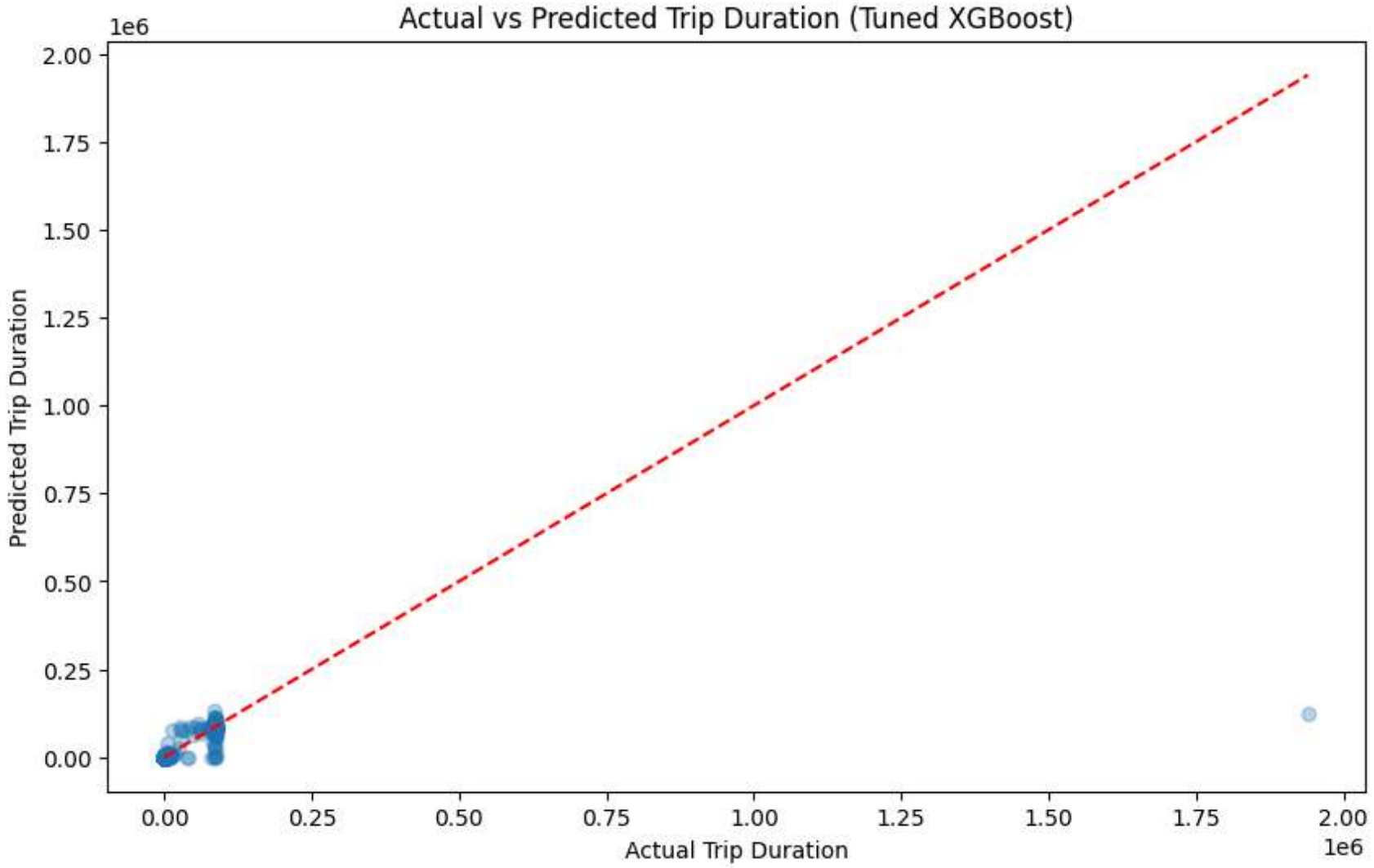
In [55]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor

# Ensure pickup_datetime is in datetime format
nyc_taxi_data['pickup_datetime'] = pd.to_datetime(nyc_taxi_data['pickup_datetime'])

# Feature Engineering
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth radius in kilometers
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])  # Convert degrees to radians
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

# Calculate trip distance if not already present
if 'trip_distance' not in nyc_taxi_data.columns:
    nyc_taxi_data['trip_distance'] = haversine(
        nyc_taxi_data['pickup_latitude'], nyc_taxi_data['pickup_longitude'],
        nyc_taxi_data['dropoff_latitude'], nyc_taxi_data['dropoff_longitude']
    )
```

```
# Additional features
nyc_taxi_data['pickup_hour'] = nyc_taxi_data['pickup_datetime'].dt.hour
nyc_taxi_data['pickup_dayofweek'] = nyc_taxi_data['pickup_datetime'].dt.dayofweek
nyc_taxi_data['pickup_month'] = nyc_taxi_data['pickup_datetime'].dt.month
nyc_taxi_data['trip_duration_hours'] = nyc_taxi_data['trip_duration'] / 3600
nyc_taxi_data['average_speed_kmh'] = nyc_taxi_data['trip_distance'] / nyc_taxi_data['trip_duration_hours']
nyc_taxi_data['is_rush_hour'] = nyc_taxi_data['pickup_hour'].apply(lambda x: 1 if (7 <= x <= 9) or (16 <= x <= 19) el
nyc_taxi_data['is_weekend'] = nyc_taxi_data['pickup_dayofweek'].apply(lambda x: 1 if x >= 5 else 0)
nyc_taxi_data['passenger_distance_interaction'] = nyc_taxi_data['passenger_count'] * nyc_taxi_data['trip_distance']

# Define features and target
features = [
    'pickup_hour', 'pickup_dayofweek', 'pickup_month', 'passenger_count', 'trip_distance',
    'average_speed_kmh', 'is_rush_hour', 'is_weekend', 'passenger_distance_interaction'
]
X = nyc_taxi_data[features]
y = np.log1p(nyc_taxi_data['trip_duration'])  # Applying log transformation

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocessor pipeline
numeric_features = ['pickup_hour', 'pickup_dayofweek', 'pickup_month', 'trip_distance', 'average_speed_kmh', 'passeng
categorical_features = ['passenger_count']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# XGBoost hyperparameter grid for RandomizedSearchCV with prefixed parameter names
param_grid = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__learning_rate': [0.01, 0.05, 0.1, 0.2],
    'regressor__max_depth': [3, 5, 7, 10],
    'regressor__subsample': [0.6, 0.8, 1.0],
    'regressor__colsample_bytree': [0.6, 0.8, 1.0]
}

# Model pipeline with XGBoost Regressor
```

```
xgb = XGBRegressor(random_state=42)

model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', xgb)
])

# RandomizedSearchCV to find the best hyperparameters
random_search = RandomizedSearchCV(
    model_pipeline, param_distributions=param_grid, n_iter=50,
    scoring='neg_mean_squared_error', cv=3, verbose=2, random_state=42, n_jobs=-1
)

# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)

# Best parameters and model evaluation
best_model = random_search.best_estimator_
print("Best Parameters:", random_search.best_params_)

# Make predictions with the tuned model
y_pred = best_model.predict(X_test)
y_pred = np.expm1(y_pred)  # Inverse the log transformation
y_test_actual = np.expm1(y_test)  # Inverse the log transformation for actual values

# Model performance metrics
mae = mean_absolute_error(y_test_actual, y_pred)
rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred))
r_squared = best_model.score(X_test, y_test)  # R^2 Score on log-transformed data

print("Tuned XGBoost Model Performance:")
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R^2):", r_squared)

# Plotting Actual vs Predicted trip durations
plt.figure(figsize=(10, 6))
plt.scatter(y_test_actual, y_pred, alpha=0.3)
plt.plot([y_test_actual.min(), y_test_actual.max()], [y_test_actual.min(), y_test_actual.max()], 'r--')
plt.xlabel('Actual Trip Duration')
plt.ylabel('Predicted Trip Duration')
```

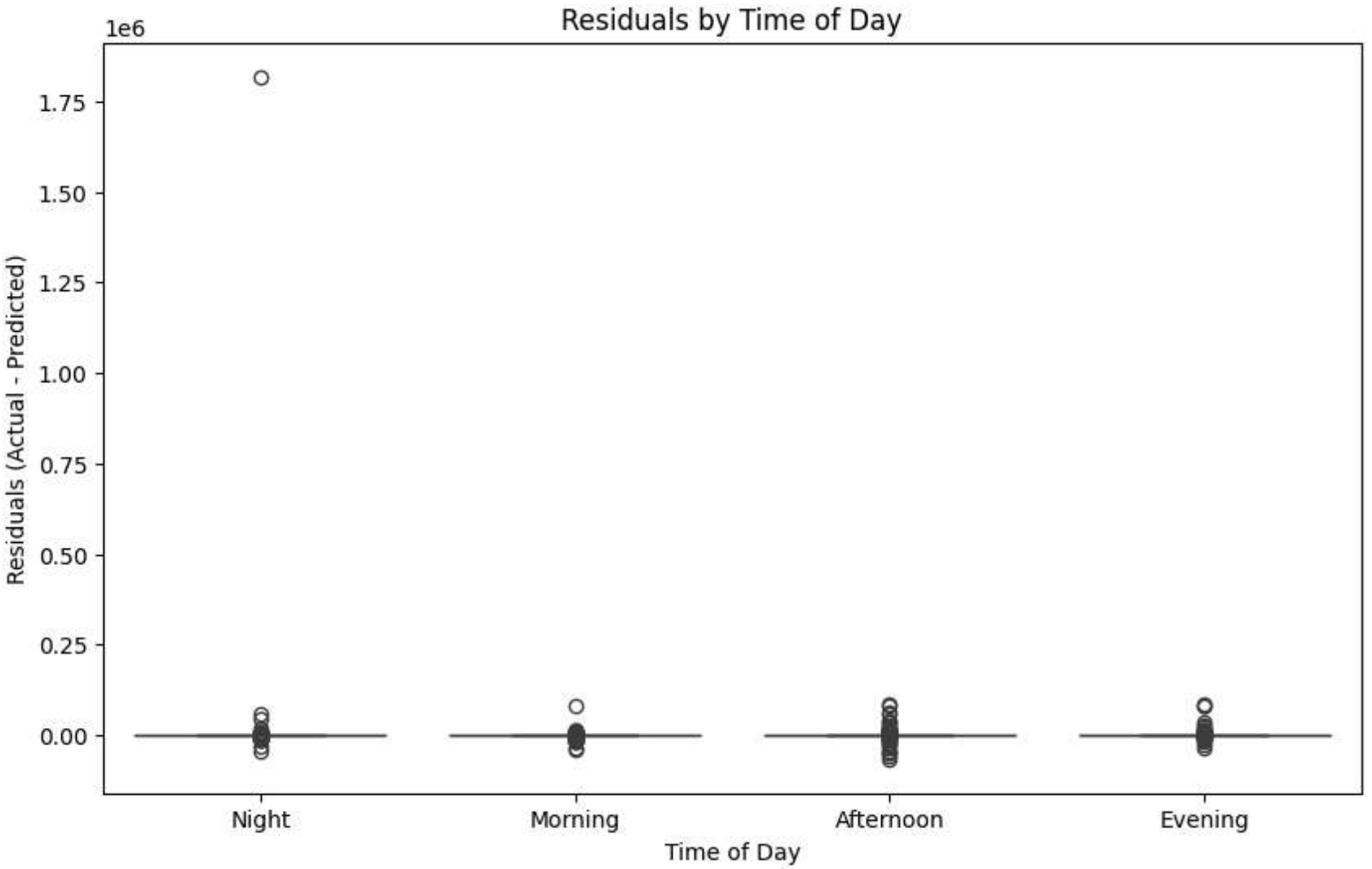
```
plt.title('Actual vs Predicted Trip Duration (Tuned XGBoost)')
plt.show()
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Parameters: {'regressor__subsample': 1.0, 'regressor__n_estimators': 200, 'regressor__max_depth': 5, 'regressor_
_learning_rate': 0.1, 'regressor__colsample_bytree': 1.0}
Tuned XGBoost Model Performance:
Mean Absolute Error (MAE): 48.202965410313155
Root Mean Squared Error (RMSE): 4831.347645798881
R-squared (R^2): 0.9655819315175204
```



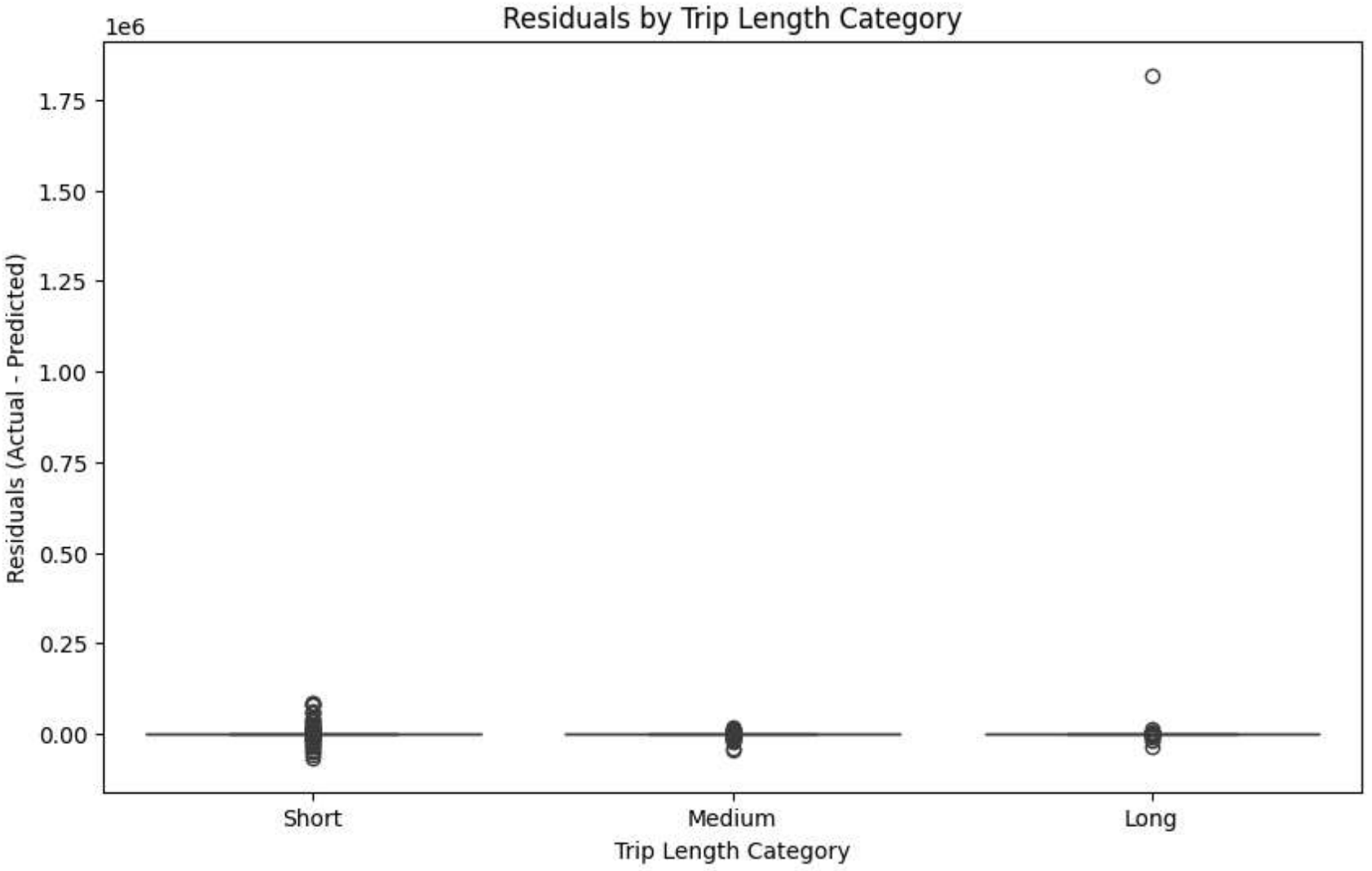
Now let's perform resildual analysis

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define time bins based on pickup hour
nyc_taxi_data['time_of_day'] = pd.cut(nyc_taxi_data['pickup_hour'],
                                      bins=[0, 6, 12, 18, 24],
                                      labels=['Night', 'Morning', 'Afternoon', 'Evening'],
                                      right=False)

# Add residuals to the original dataset for analysis
nyc_taxi_data['residuals'] = y_test_actual - y_pred  # Assuming y_test_actual and y_pred from earlier

# Plot residuals by time of day
plt.figure(figsize=(10, 6))
sns.boxplot(x='time_of_day', y='residuals', data=nyc_taxi_data)
plt.title('Residuals by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Residuals (Actual - Predicted)')
plt.show()
```



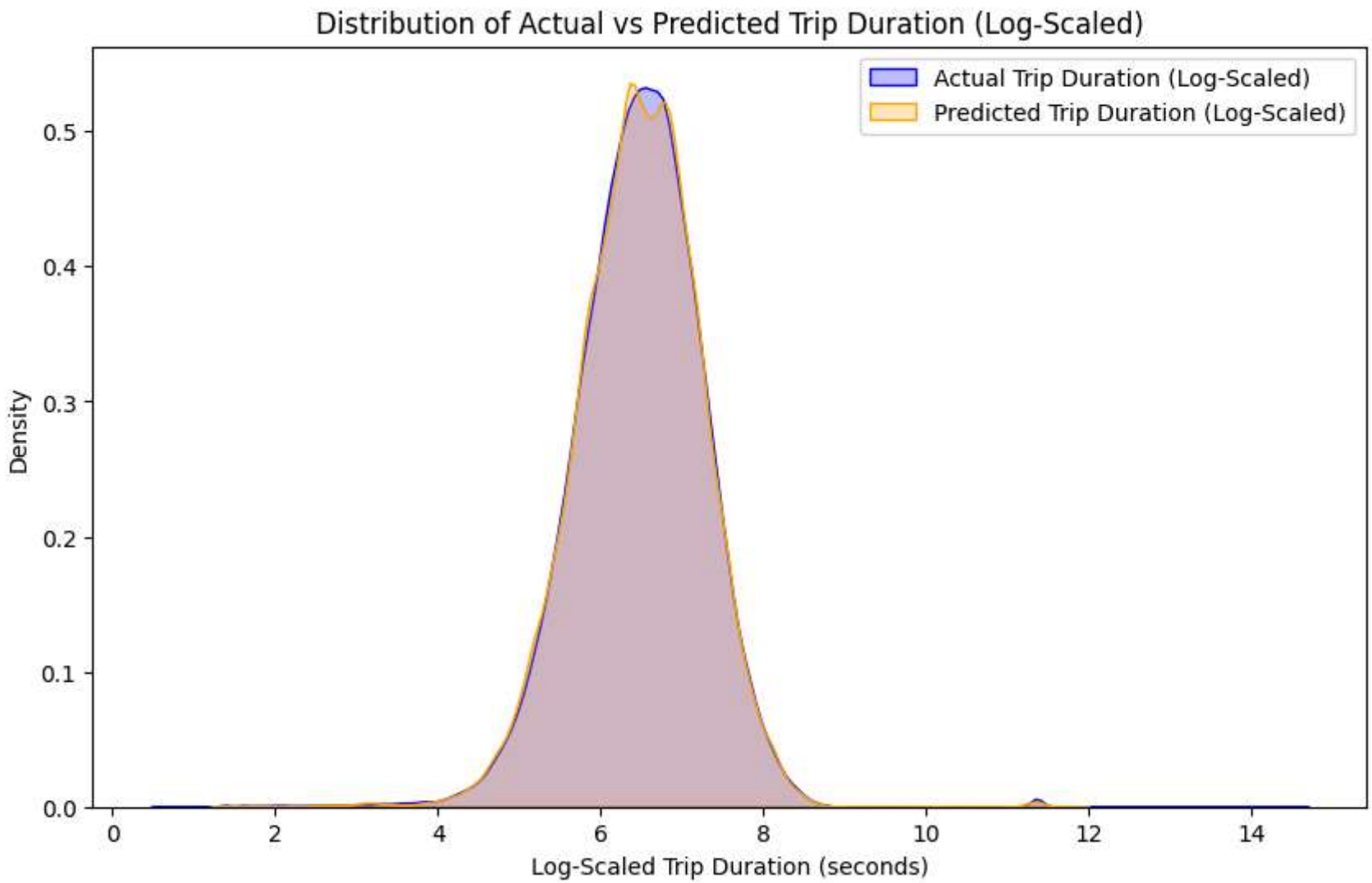
In [42]:
```
# Define trip length bins
nyc_taxi_data['trip_length_category'] = pd.cut(nyc_taxi_data['trip_distance'],
                                               bins=[0, 5, 15, np.inf],
                                               labels=['Short', 'Medium', 'Long'],
                                               right=False)

# Plot residuals by trip length category
plt.figure(figsize=(10, 6))
sns.boxplot(x='trip_length_category', y='residuals', data=nyc_taxi_data)
```

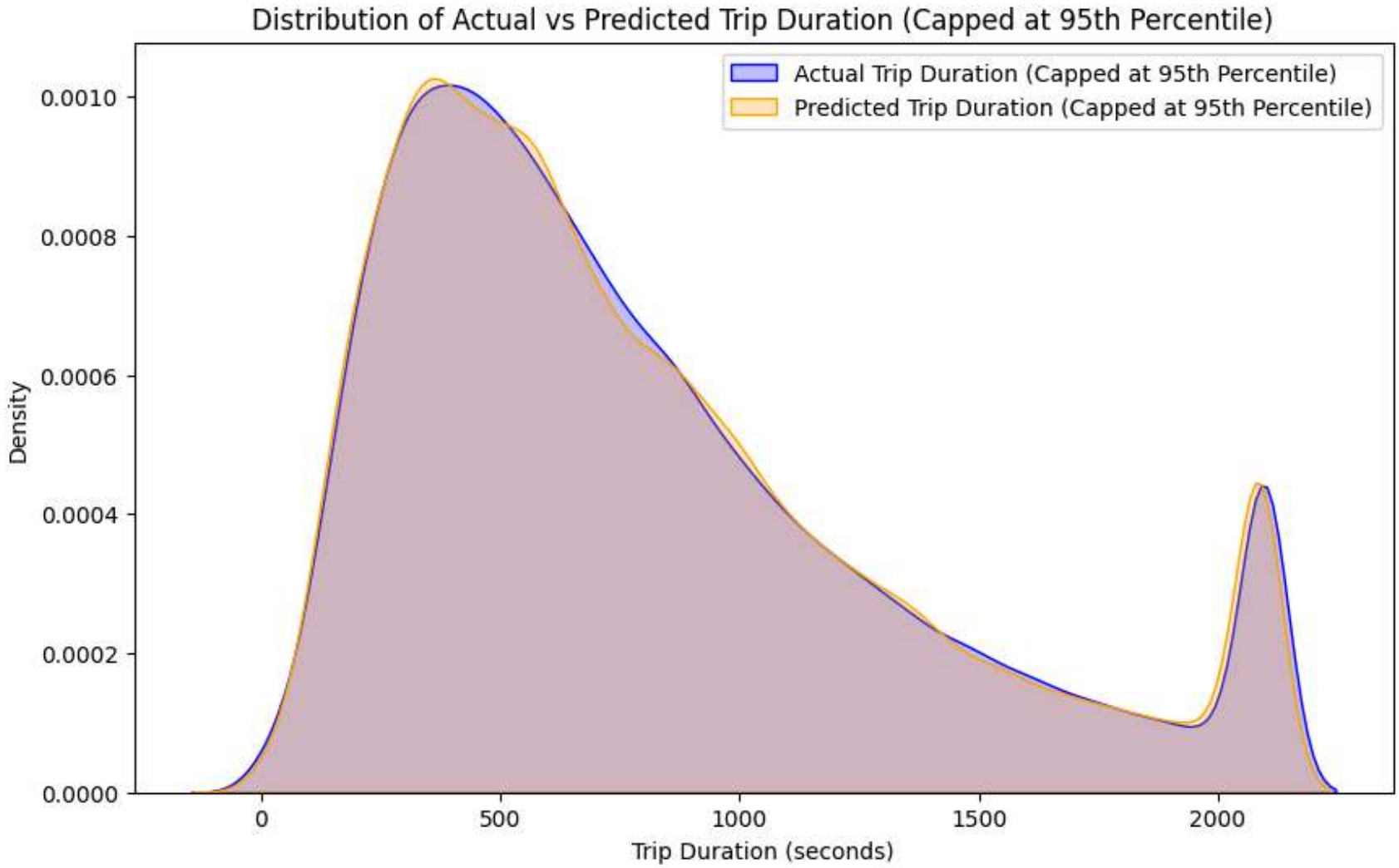
```
plt.title('Residuals by Trip Length Category')
plt.xlabel('Trip Length Category')
plt.ylabel('Residuals (Actual - Predicted)')
plt.show()
```

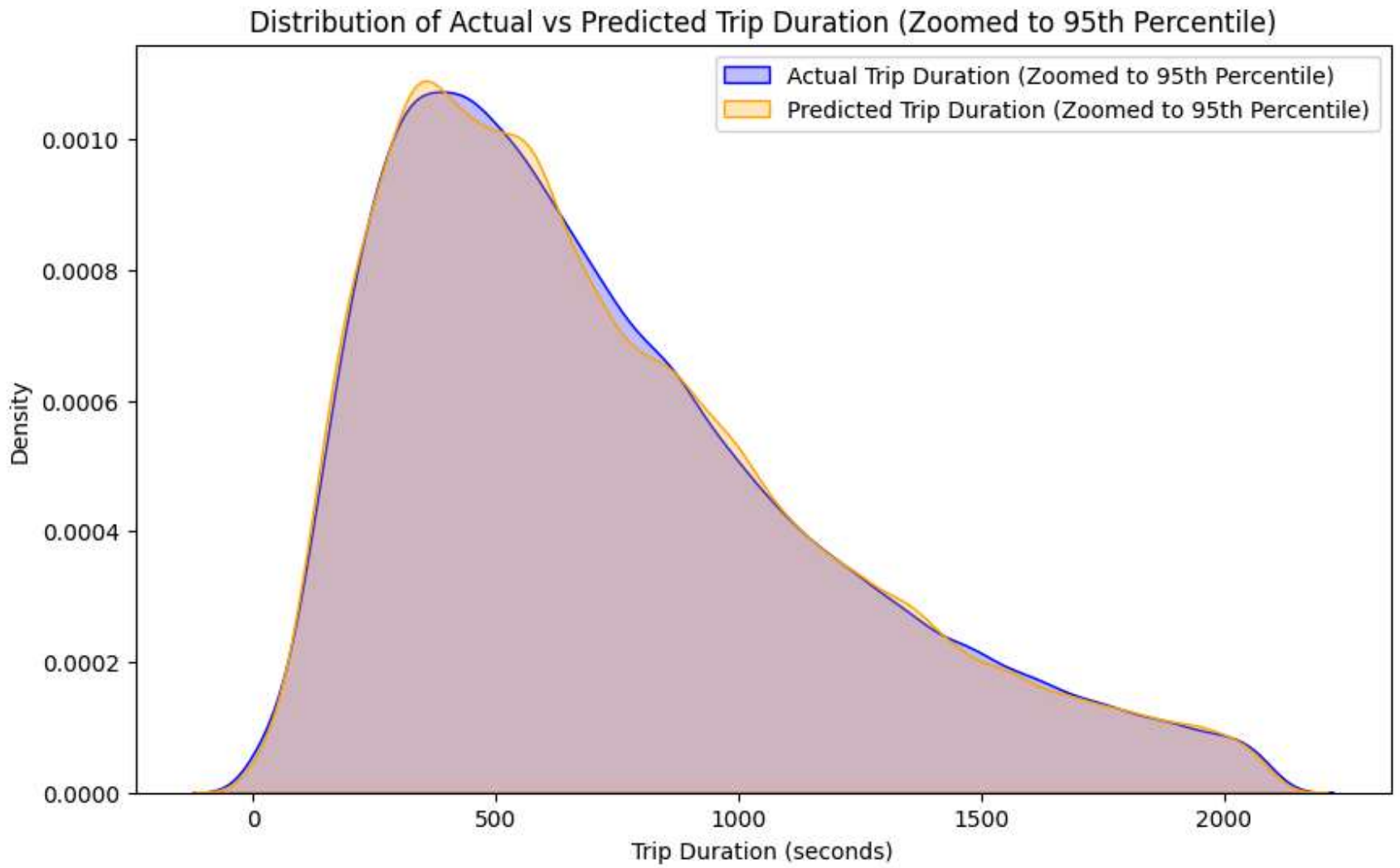


In [47]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Log-Scaled KDE Plot
plt.figure(figsize=(10, 6))
sns.kdeplot(np.log1p(y_test_actual), label="Actual Trip Duration (Log-Scaled)", color="blue", fill=True)
sns.kdeplot(np.log1p(y_pred), label="Predicted Trip Duration (Log-Scaled)", color="orange", fill=True)
plt.xlabel('Log-Scaled Trip Duration (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Actual vs Predicted Trip Duration (Log-Scaled)')
plt.legend()
plt.show()

# Capped KDE Plot (95th Percentile)
cap_value_actual = np.percentile(y_test_actual, 95)
cap_value_pred = np.percentile(y_pred, 95)
y_test_actual_capped = np.clip(y_test_actual, None, cap_value_actual)
y_pred_capped = np.clip(y_pred, None, cap_value_pred)

plt.figure(figsize=(10, 6))
sns.kdeplot(y_test_actual_capped, label="Actual Trip Duration (Capped at 95th Percentile)", color="blue", fill=True)
sns.kdeplot(y_pred_capped, label="Predicted Trip Duration (Capped at 95th Percentile)", color="orange", fill=True)
plt.xlabel('Trip Duration (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Actual vs Predicted Trip Duration (Capped at 95th Percentile)')
plt.legend()
plt.show()

# Zoomed KDE Plot (up to 95th Percentile)
plt.figure(figsize=(10, 6))
sns.kdeplot(y_test_actual[y_test_actual <= cap_value_actual], label="Actual Trip Duration (Zoomed to 95th Percentile)
sns.kdeplot(y_pred[y_pred <= cap_value_pred], label="Predicted Trip Duration (Zoomed to 95th Percentile)", color="ora
plt.xlabel('Trip Duration (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Actual vs Predicted Trip Duration (Zoomed to 95th Percentile)')
plt.legend()
plt.show()
```

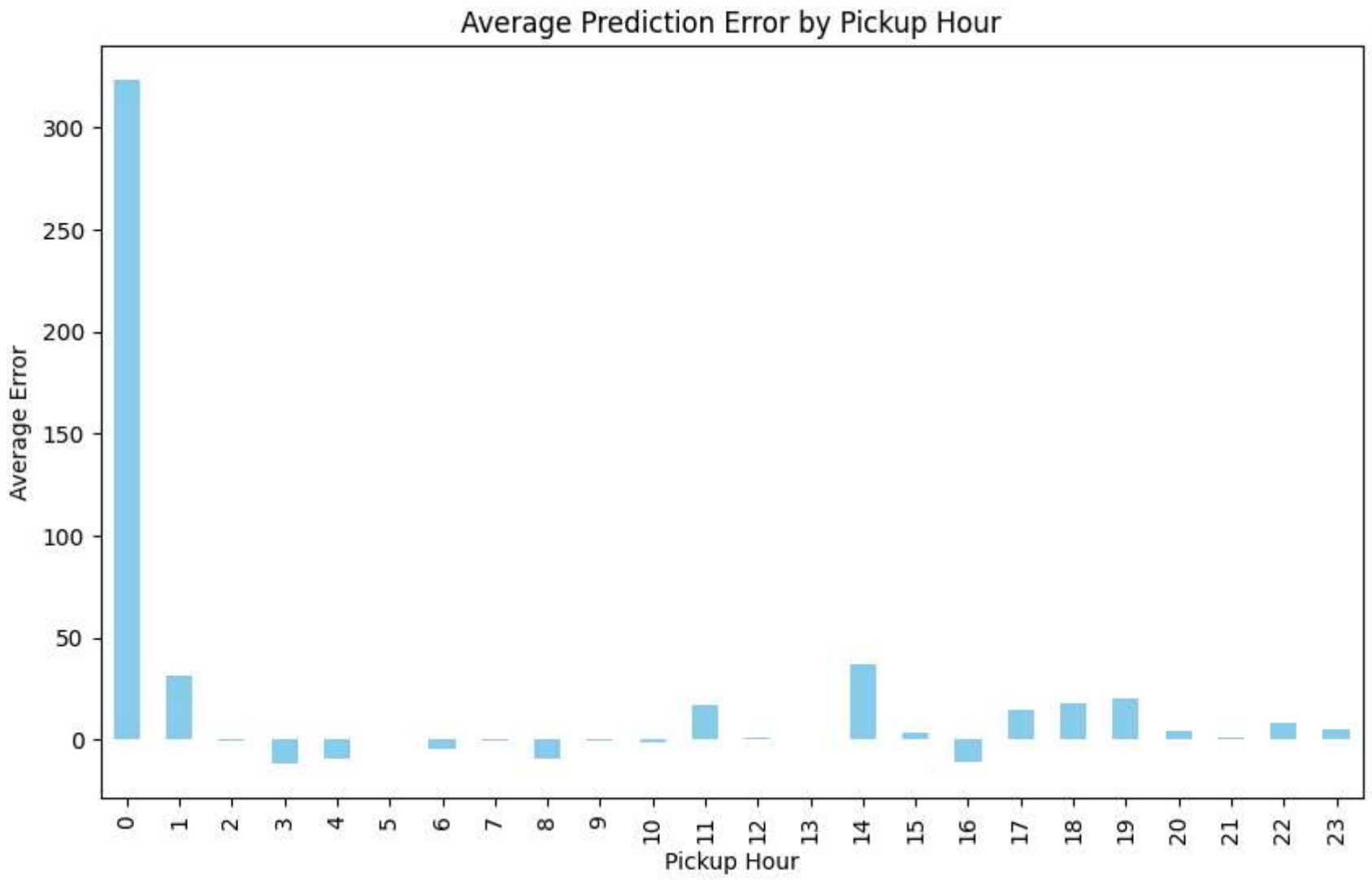

Distribution of Actual vs Predicted Trip Duration (Log-Scaled)
Actual Trip Duration (Log-Scaled)
Predicted Trip Duration (Log-Scaled)
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
12
14
Log-Scaled Trip Duration (seconds)


Distribution of Actual vs Predicted Trip Duration (Capped at 95th Percentile)
Actual Trip Duration (Capped at 95th Percentile)
Predicted Trip Duration (Capped at 95th Percentile)
Density
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
500
1000
1500
2000
Trip Duration (seconds)



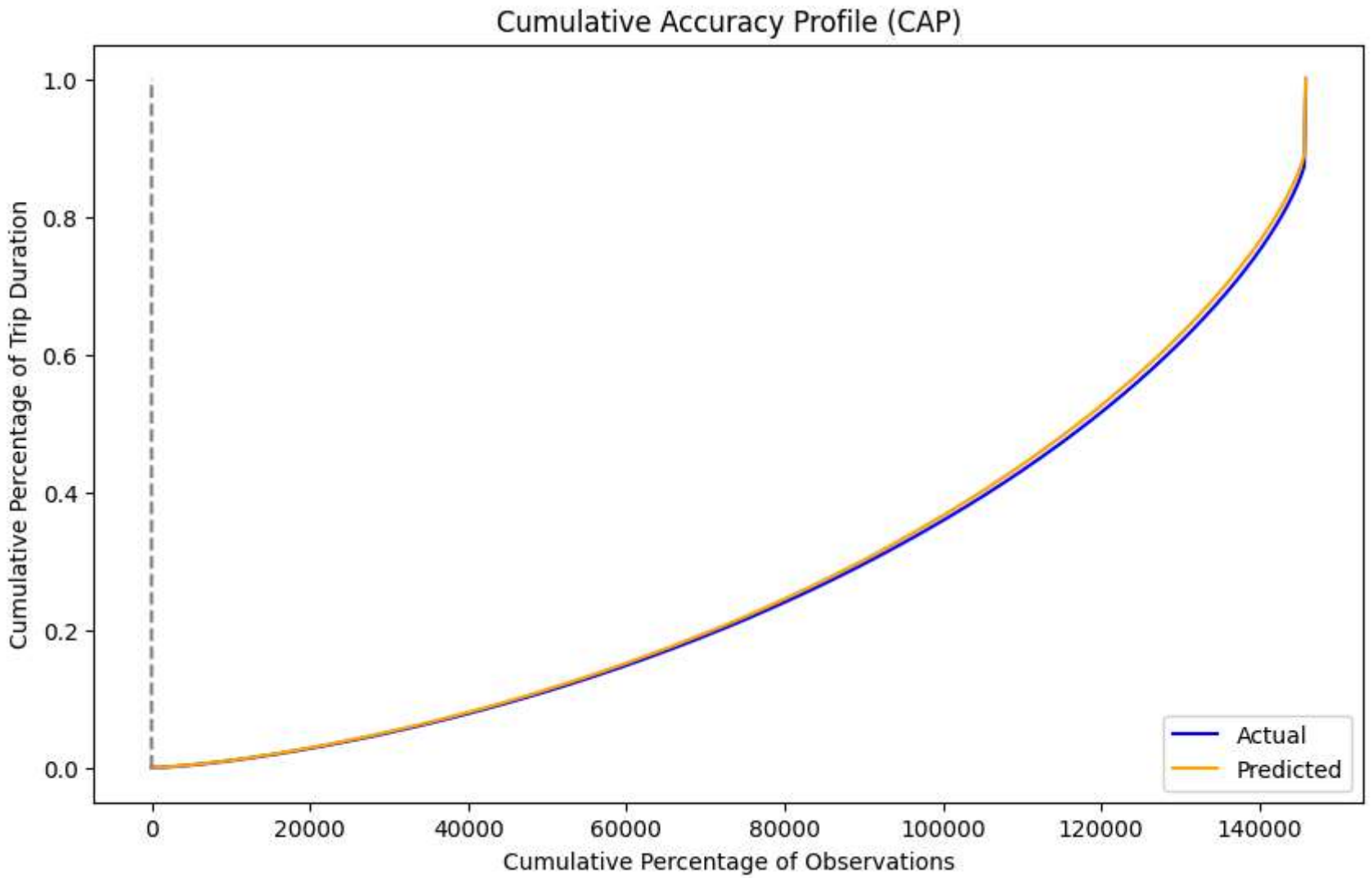
In [48]:
```
error_df = X_test.copy()
error_df['actual_duration'] = y_test_actual
error_df['predicted_duration'] = y_pred
error_df['error'] = error_df['actual_duration'] - error_df['predicted_duration']

# Example: Plot average error by hour
avg_error_by_hour = error_df.groupby('pickup_hour')['error'].mean()
plt.figure(figsize=(10, 6))
avg_error_by_hour.plot(kind='bar', color='skyblue')
```

```
plt.title('Average Prediction Error by Pickup Hour')
plt.xlabel('Pickup Hour')
plt.ylabel('Average Error')
plt.show()
```



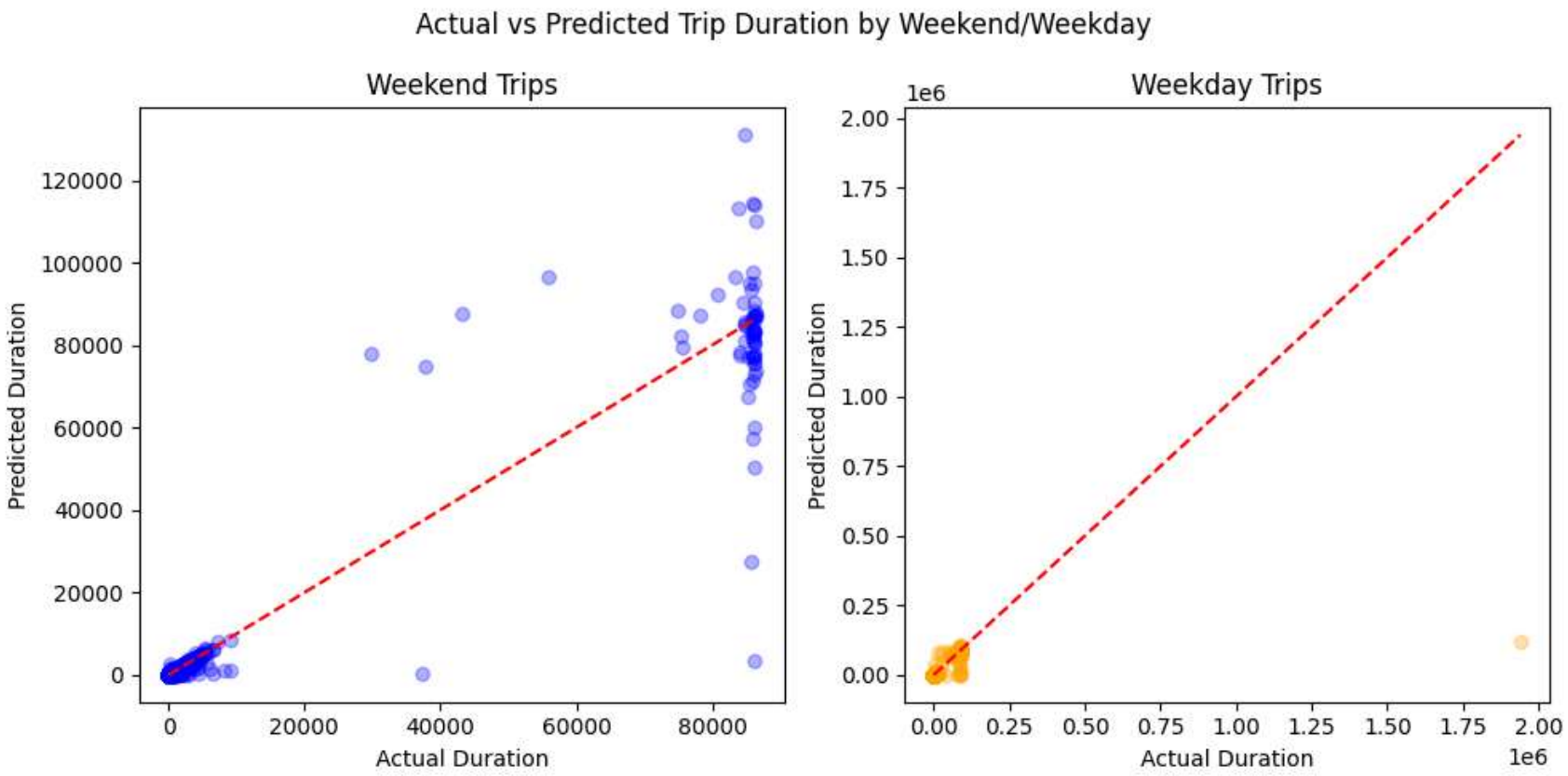
In [49]:
```
sorted_indices = np.argsort(y_test_actual)
sorted_actuals = np.array(y_test_actual)[sorted_indices]
sorted_preds = np.array(y_pred)[sorted_indices]
```

```
cumulative_actuals = np.cumsum(sorted_actuals) / np.sum(sorted_actuals)
cumulative_preds = np.cumsum(sorted_preds) / np.sum(sorted_preds)

plt.figure(figsize=(10, 6))
plt.plot(cumulative_actuals, label='Actual', color='blue')
plt.plot(cumulative_preds, label='Predicted', color='orange')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('Cumulative Accuracy Profile (CAP)')
plt.xlabel('Cumulative Percentage of Observations')
plt.ylabel('Cumulative Percentage of Trip Duration')
plt.legend()
plt.show()
```



In [53]:
```
# Example: Plotting predicted vs actual trip duration for weekend and weekday trips
weekend_df = error_df[error_df['is_weekend'] == 1]
weekday_df = error_df[error_df['is_weekend'] == 0]

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(weekend_df['actual_duration'], weekend_df['predicted_duration'], alpha=0.3, color='blue')
plt.plot([weekend_df['actual_duration'].min(), weekend_df['actual_duration'].max()],
         [weekend_df['actual_duration'].min(), weekend_df['actual_duration'].max()], 'r--')
```

```
plt.title('Weekend Trips')
plt.xlabel('Actual Duration')
plt.ylabel('Predicted Duration')

plt.subplot(1, 2, 2)
plt.scatter(weekday_df['actual_duration'], weekday_df['predicted_duration'], alpha=0.3, color='orange')
plt.plot([weekday_df['actual_duration'].min(), weekday_df['actual_duration'].max()],
         [weekday_df['actual_duration'].min(), weekday_df['actual_duration'].max()], 'r--')
plt.title('Weekday Trips')
plt.xlabel('Actual Duration')
plt.ylabel('Predicted Duration')

plt.suptitle('Actual vs Predicted Trip Duration by Weekend/Weekday')
plt.tight_layout()
plt.show()
```



In [ ]: