Problem Statement

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. The company wants to understand and process the data coming out of data engineering pipelines.Clean, sanitize and manipulate data to get useful features out of raw fields.Make sense out of the raw data and help the data science team to build forecasting models on it

Import Required Libraries

In [99]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
from scipy.stats import ttest_ind
```

Exploaratory Data Analysis (EDA)

In [100…

```
# Import Data
df = pd.read_csv('delhivery.txt')
```

In [101…

```
# Read first few record
df.head(10)
```

Out[101…

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

10 rows × 24 columns

In [102…

```
# Check Data Types of each columns
df.dtypes
```

Out[102…

	0
data	object
trip_creation_time	object
route_schedule_uuid	object
route_type	object
trip_uuid	object
source_center	object
source_name	object
destination_center	object
destination_name	object
od_start_time	object
od_end_time	object
start_scan_to_end_scan	float64
is_cutoff	bool
cutoff_factor	int64
cutoff_timestamp	object
actual_distance_to_destination	float64
actual_time	float64
osrm_time	float64
osrm_distance	float64
factor	float64
segment_actual_time	float64
segment_osrm_time	float64
segment_osrm_distance	float64
segment_factor	float64

dtype: object

In [103…

```
#Check for missing values
df.isnull().sum()
```

Out[103…

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
is_cutoff	0
cutoff_factor	0
cutoff_timestamp	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
factor	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0
segment_factor	0

dtype: int64

In [104…

```
# Check for count of unique values for each column to understand categorical
df.nunique()
```

Out[104…

	0
data	2
trip_creation_time	14817
route_schedule_uuid	1504
route_type	2
trip_uuid	14817
source_center	1508
source_name	1498
destination_center	1481
destination_name	1468
od_start_time	26369
od_end_time	26369
start_scan_to_end_scan	1915
is_cutoff	2
cutoff_factor	501
cutoff_timestamp	93180
actual_distance_to_destination	144515
actual_time	3182
osrm_time	1531
osrm_distance	138046
factor	45641
segment_actual_time	747
segment_osrm_time	214
segment_osrm_distance	113799
segment_factor	5675

dtype: int64

In [105…

```
# Check shape of data
df.shape
```

Out[105…

```
(144867, 24)
```

In [106…

```
# Convert data , route_type to categorical format since it has 2 categories
df['data'] = df['data'].astype('category')
df['route_type'] = df['route_type'].astype('category')
df.data.dtype , df.route_type.dtype
```

Out[106…

```
(CategoricalDtype(categories=['test', 'training'], ordered=False, categorie
s_dtype=object),
 CategoricalDtype(categories=['Carting', 'FTL'], ordered=False, categories_
dtype=object))
```

In [107…

```python
# Statistical Summary of numerical data
df.describe()
```

Out[107…

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination
count	144867.000000	144867.000000	144867.000000
mean	961.262986	232.926567	234.073372
std	1037.012769	344.755577	344.990009
min	20.000000	9.000000	9.000045
25%	161.000000	22.000000	23.355874
50%	449.000000	66.000000	66.126571
75%	1634.000000	286.000000	286.708875
max	7898.000000	1927.000000	1927.447705

In [108…

```python
df[df.source_name.isna()]
```

Out[108…

	data	trip_creation_time	route_schedule_uuid	route_type	
112	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378
113	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378
114	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378
115	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378
116	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378
...	...	...	...	...	
144484	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385
144485	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385
144486	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385
144487	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385
144488	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385

293 rows × 24 columns

In [109…

```
df.columns
```

Out[109…

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

In [110…

```
df[df.source_name.isna()][['destination_name','source_center']]
```

Out[110…

	destination_name	source_center
112	Jaipur_Hub (Rajasthan)	IND342902A1B
113	Jaipur_Hub (Rajasthan)	IND342902A1B
114	Jaipur_Hub (Rajasthan)	IND342902A1B
115	Jaipur_Hub (Rajasthan)	IND342902A1B
116	Jaipur_Hub (Rajasthan)	IND342902A1B
...	...	...
144484	Gwalior_HrihrNgr_I (Madhya Pradesh)	IND282002AAD
144485	Gwalior_HrihrNgr_I (Madhya Pradesh)	IND282002AAD
144486	Gwalior_HrihrNgr_I (Madhya Pradesh)	IND282002AAD
144487	Gwalior_HrihrNgr_I (Madhya Pradesh)	IND282002AAD
144488	Gwalior_HrihrNgr_I (Madhya Pradesh)	IND282002AAD

293 rows × 2 columns

In [111…

```
# Fill source name with Null values with unknown value
df['source_name'] = df['source_name'].fillna('Unknown')
```

In [112…

```
df['source_name'].isna().sum()
```

Out[112…

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

In [113…

```
df.isna().sum()
```

Out[113…

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	0
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
is_cutoff	0
cutoff_factor	0
cutoff_timestamp	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
factor	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0
segment_factor	0

dtype: int64

In [114…

```
# fill destination name with unknown for null values
df['destination_name'] = df['destination_name'].fillna('Unknown')
```

In [115…

```
df.destination_name.isna().sum()
```

Out[115…

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

In [116…

```
df.dtypes
```

Out[116…

	0
data	category
trip_creation_time	object
route_schedule_uuid	object
route_type	category
trip_uuid	object
source_center	object
source_name	object
destination_center	object
destination_name	object
od_start_time	object
od_end_time	object
start_scan_to_end_scan	float64
is_cutoff	bool
cutoff_factor	int64
cutoff_timestamp	object
actual_distance_to_destination	float64
actual_time	float64
osrm_time	float64
osrm_distance	float64
factor	float64
segment_actual_time	float64
segment_osrm_time	float64
segment_osrm_distance	float64
segment_factor	float64

dtype: object

In [117…

```
# Convert some object column to timestamp since it contains time related dat


df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'], errors='
df['od_start_time'] = pd.to_datetime(df['od_start_time'], errors='coerce')
df['od_end_time'] = pd.to_datetime(df['od_end_time'], errors='coerce')
```

In [118…

```
df.dtypes
```

Out[118…

	0
data	category
trip_creation_time	datetime64[ns]
route_schedule_uuid	object
route_type	category
trip_uuid	object
source_center	object
source_name	object
destination_center	object
destination_name	object
od_start_time	datetime64[ns]
od_end_time	datetime64[ns]
start_scan_to_end_scan	float64
is_cutoff	bool
cutoff_factor	int64
cutoff_timestamp	object
actual_distance_to_destination	float64
actual_time	float64
osrm_time	float64
osrm_distance	float64
factor	float64
segment_actual_time	float64
segment_osrm_time	float64
segment_osrm_distance	float64
segment_factor	float64

dtype: object

In [119…

```
#Convert iscutoff to boolean type since it has boolean values
df['is_cutoff'] = df['is_cutoff'].astype('bool')
```

In [120…

```
df.source_name.value_counts()
```

Out[120…

source_name	count
Gurgaon_Bilaspur_HB (Haryana)	23347
Bangalore_Nelmngla_H (Karnataka)	9975
Bhiwandi_Mankoli_HB (Maharashtra)	9088
Pune_Tathawde_H (Maharashtra)	4061
Hyderabad_Shamshbd_H (Telangana)	3340
...	...
Islampure_ShbdnDPP_D (West Bengal)	1
Bhadra_GMndiDPP_D (Rajasthan)	1
Badkulla_Central_DPP_1 (West Bengal)	1
Hajipur_ThaneDPP_D (Bihar)	1
Soro_UttarDPP_D (Orissa)	1

1499 rows × 1 columns

dtype: int64

In [121…

```
df.shape
```

Out[121…

```
(144867, 24)
```

In [122…

```
df.columns
```

Out[122…

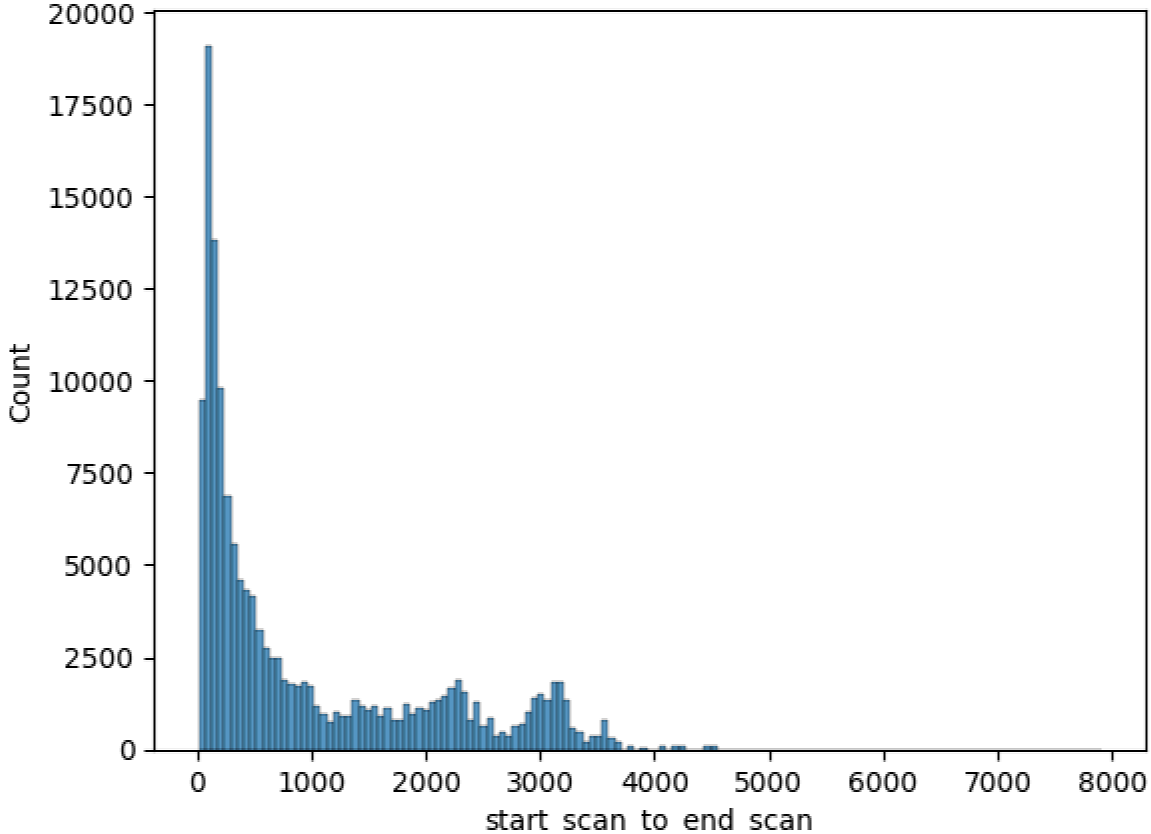
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

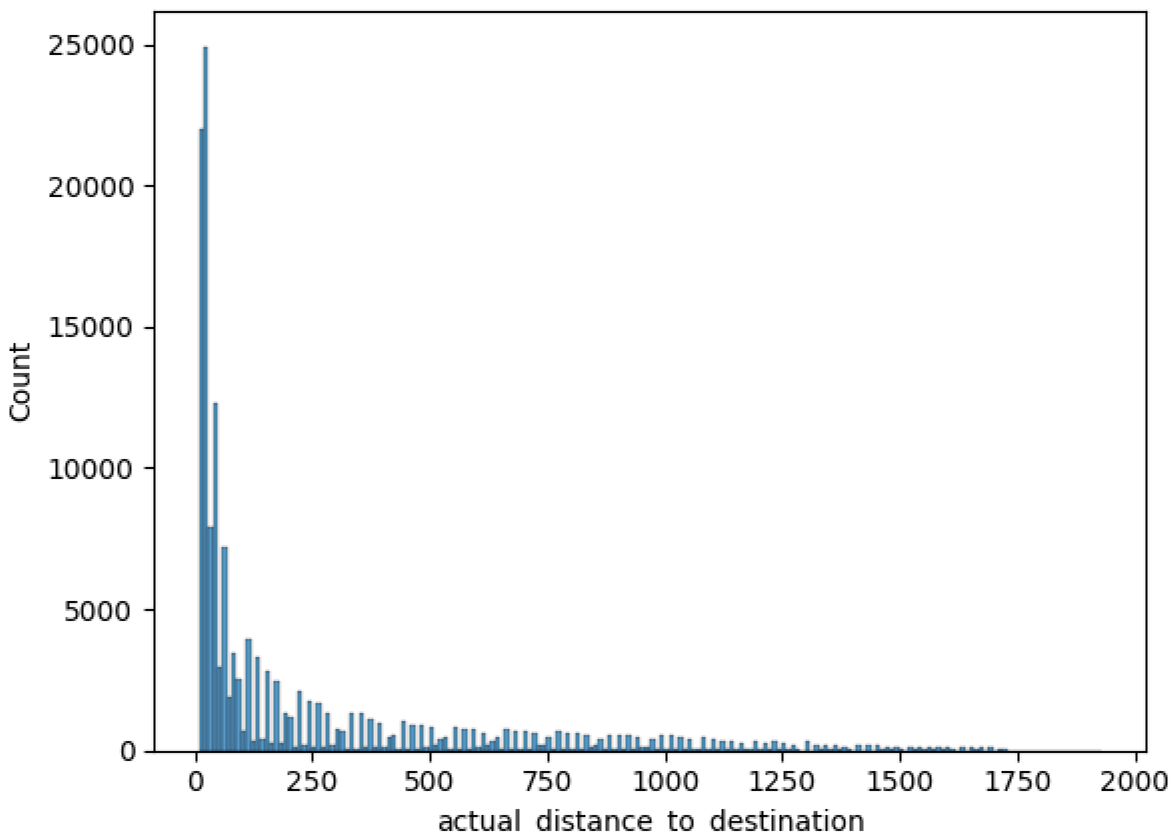
Univariate Analysis

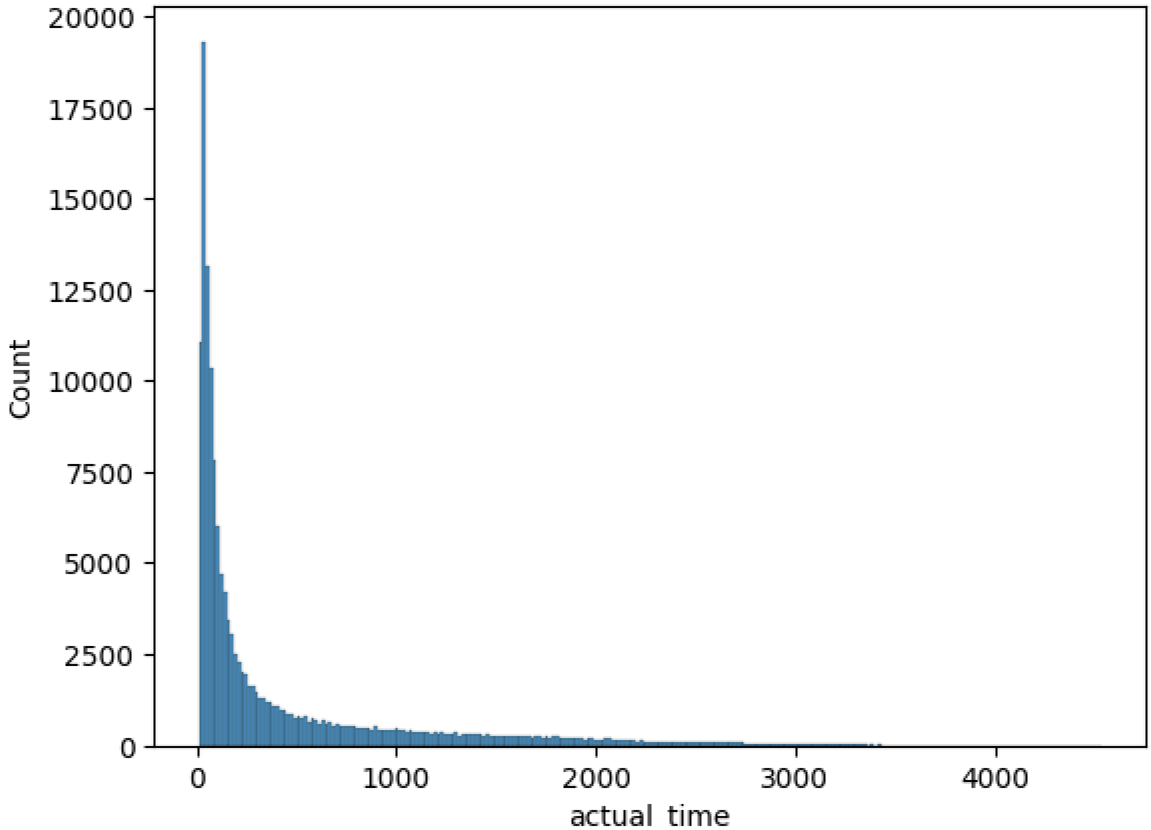
Continuous variables

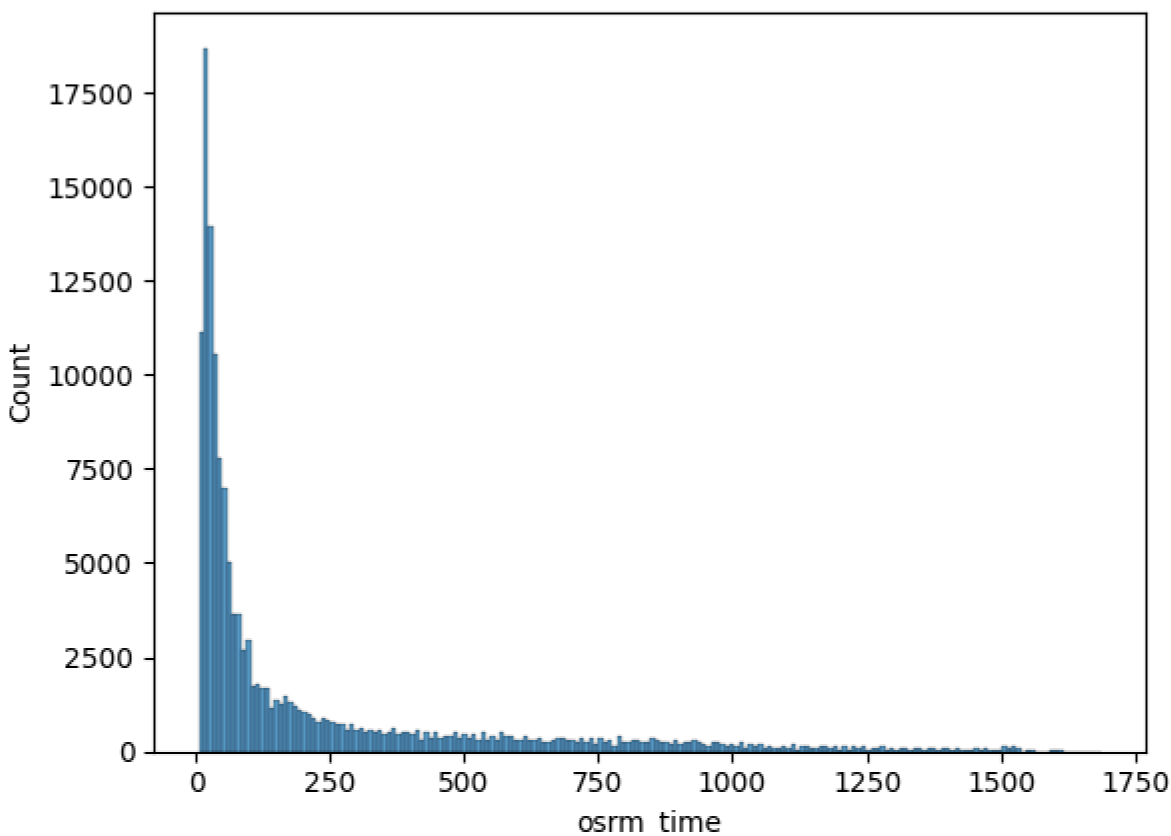
In [123…

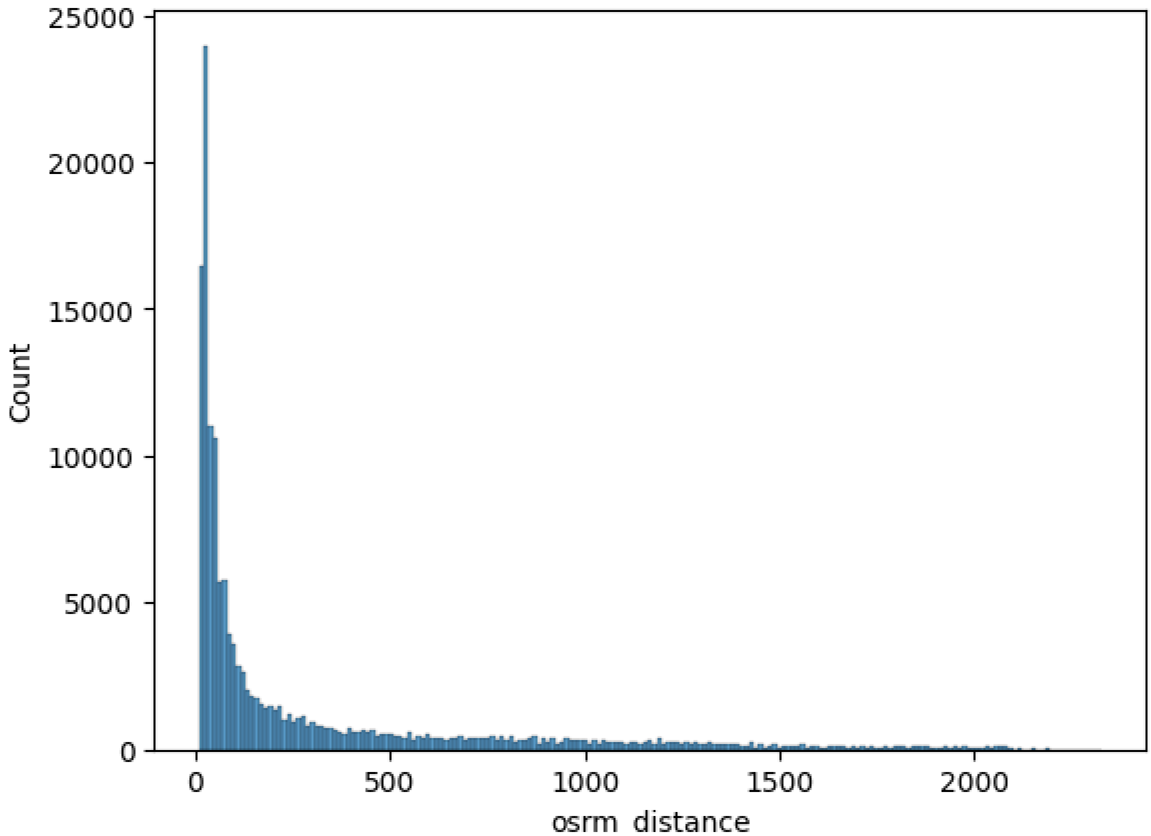
```
columns = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actu

for cols in columns:
    sns.histplot(df[cols])
    plt.show()
```


Count
start_scan_to_end_scan
0
1000
2000
3000
4000
5000
6000
7000
8000
20000
17500
15000
12500
10000
7500
5000
2500
0

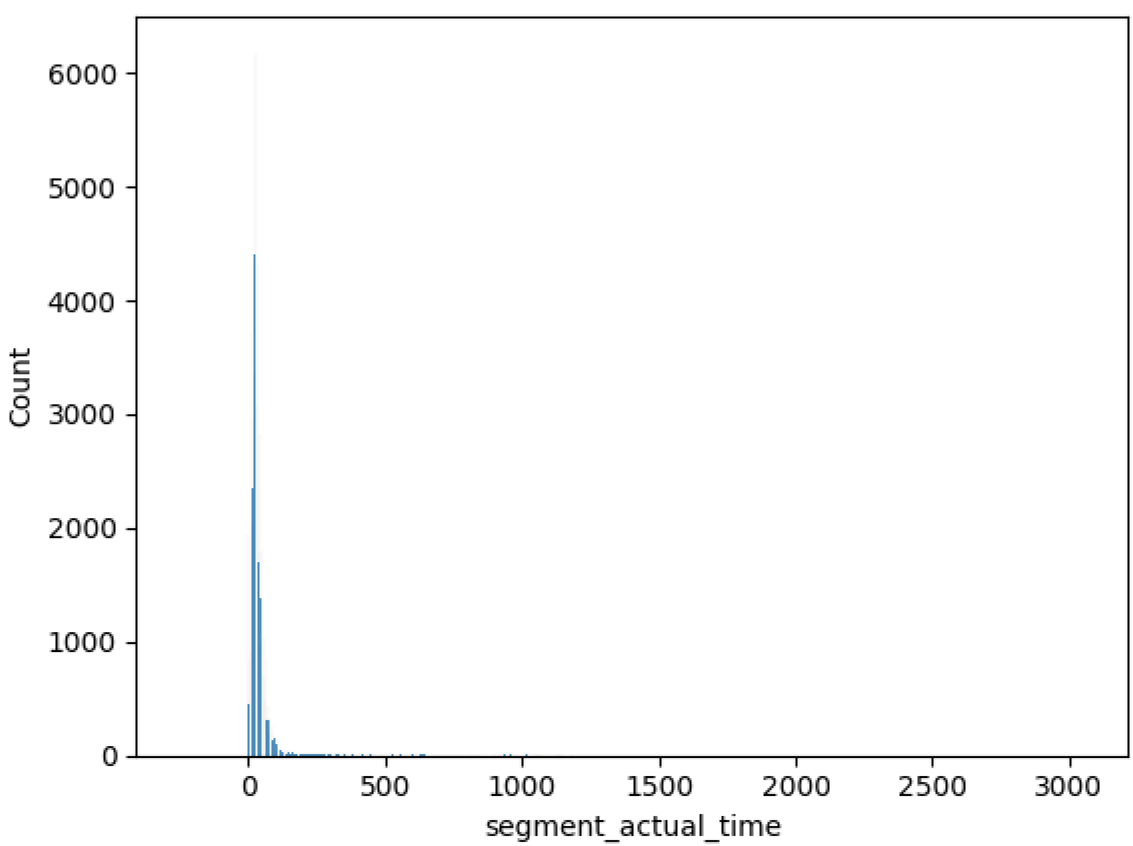

Count
actual_distance_to_destination
0
250
500
750
1000
1250
1500
1750
2000
25000
20000
15000
10000
5000
0


20000
17500
15000
12500
10000
7500
5000
2500
0
Count
0
1000
2000
3000
4000
actual_time

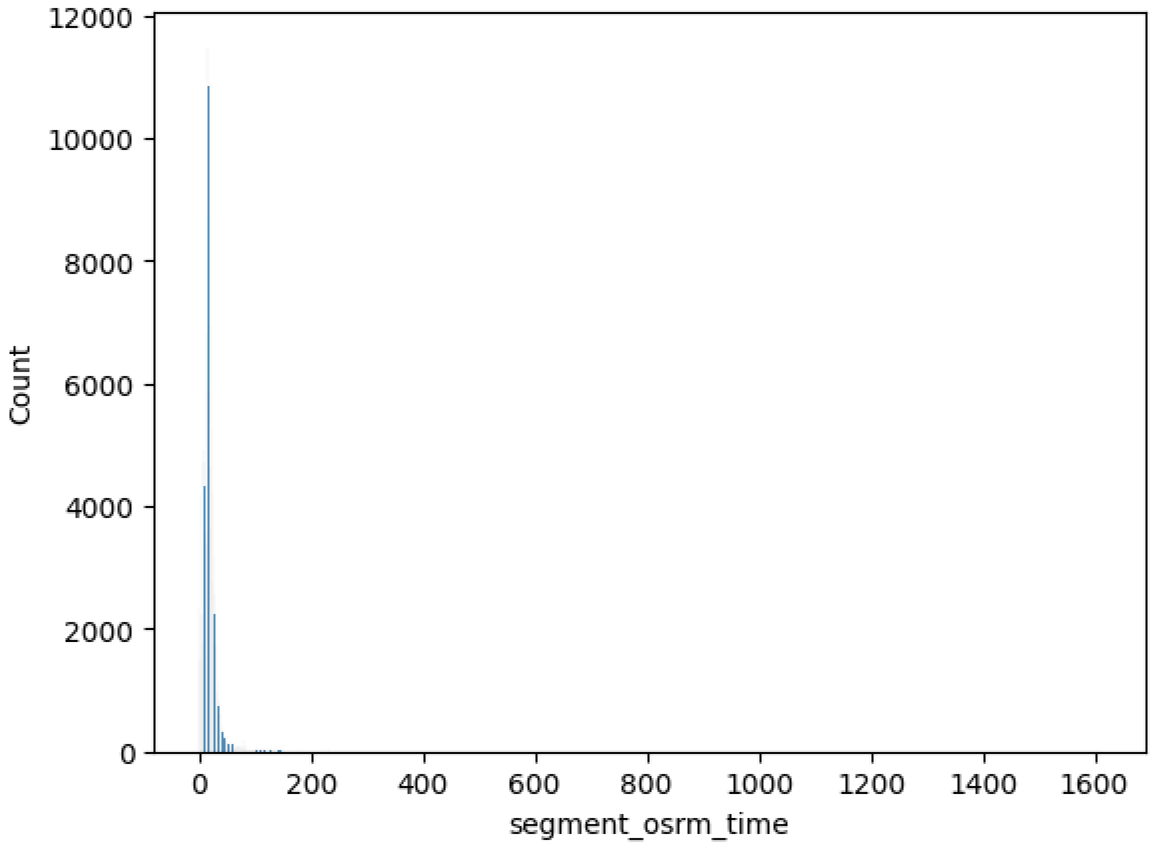

17500
15000
12500
10000
7500
5000
2500
0
Count
0
250
500
750
1000
1250
1500
1750
osrm_time



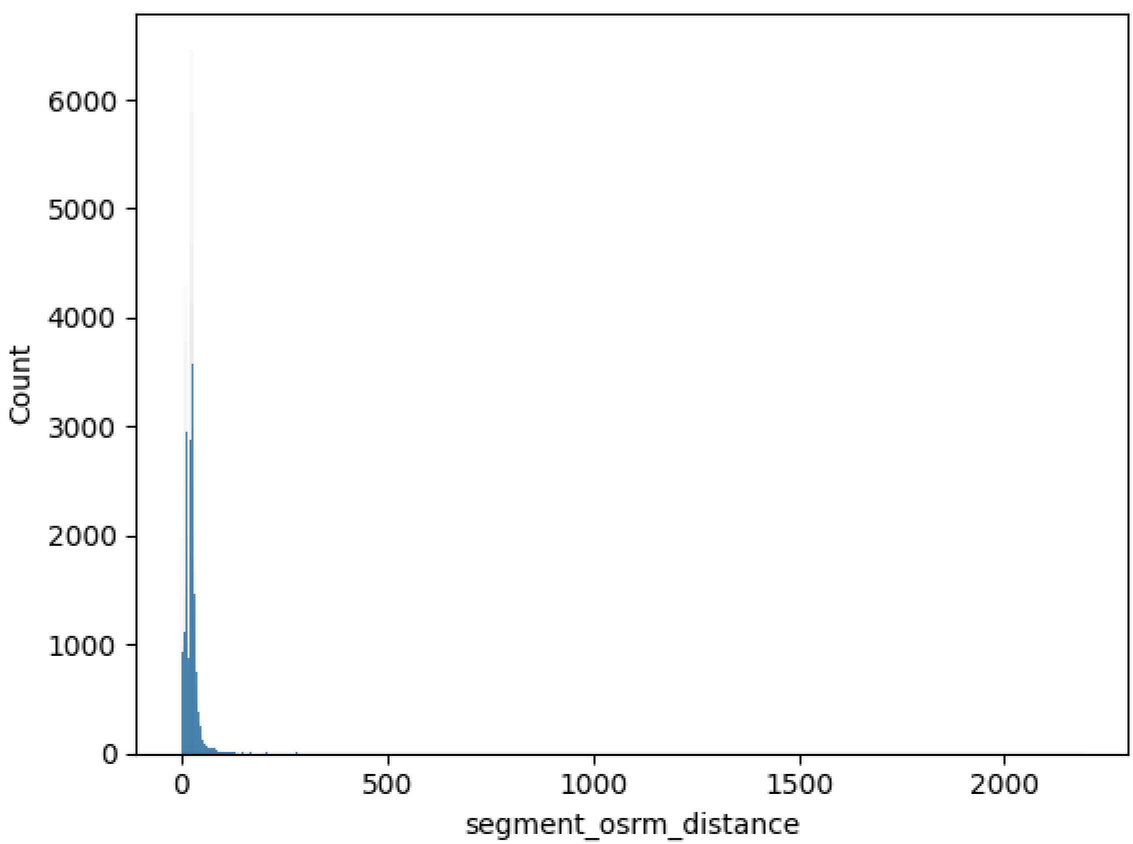
25000
20000
15000
10000
5000
0
Count
0
500
1000
1500
2000
osrm_distance



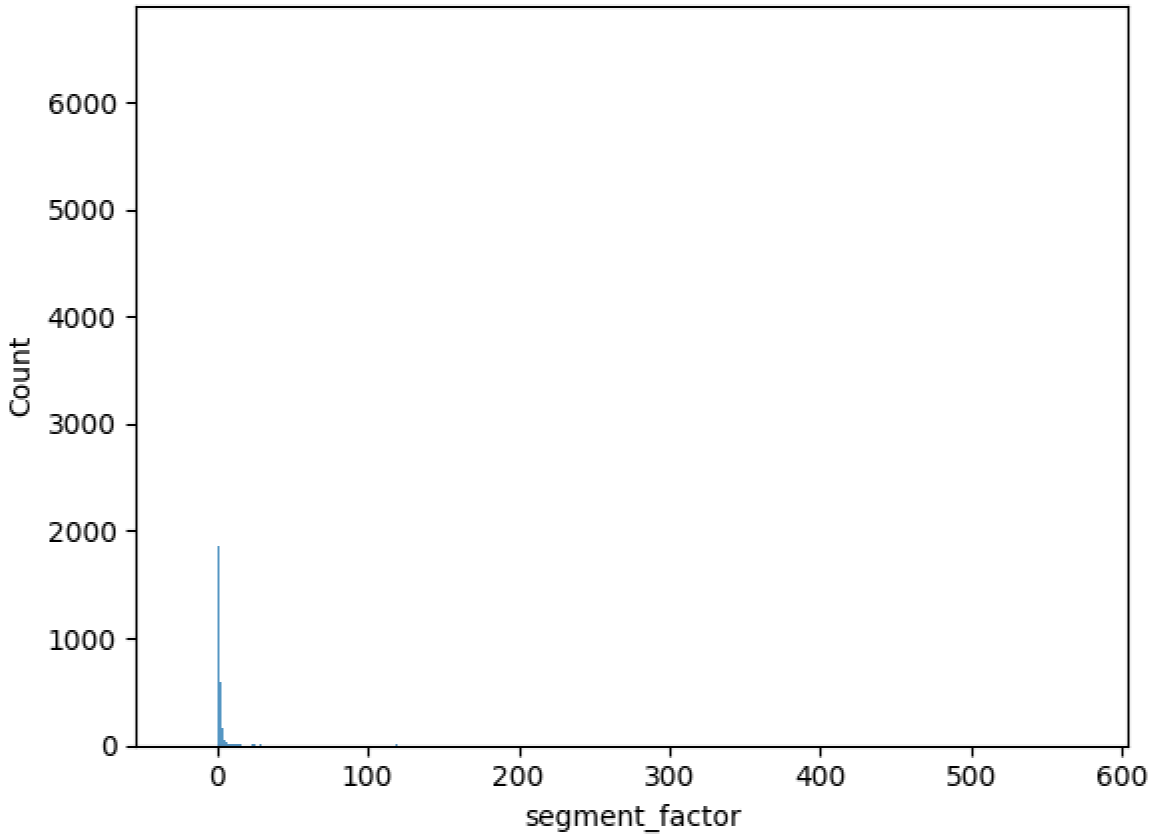
6000
5000
4000
3000
2000
1000
0
Count
0
500
1000
1500
2000
2500
3000
segment_actual_time



12000
10000
8000
6000
4000
2000
0
Count
0
200
400
600
800
1000
1200
1400
1600
segment_osrm_time



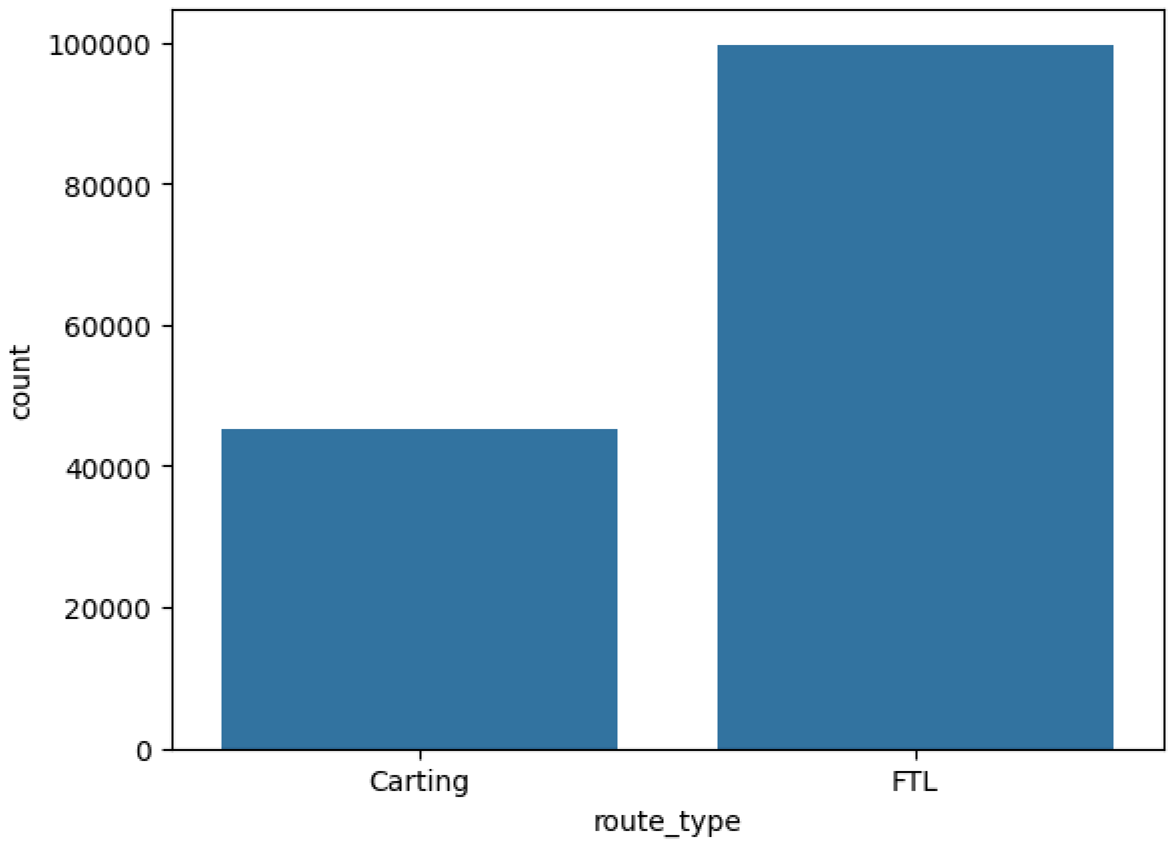
6000
5000
4000
3000
2000
1000
0
Count
0
500
1000
1500
2000
segment_osrm_distance



Categorical Variable

In [124…

```
sns.countplot(x=df['route_type'],data=df)

plt.show()
```



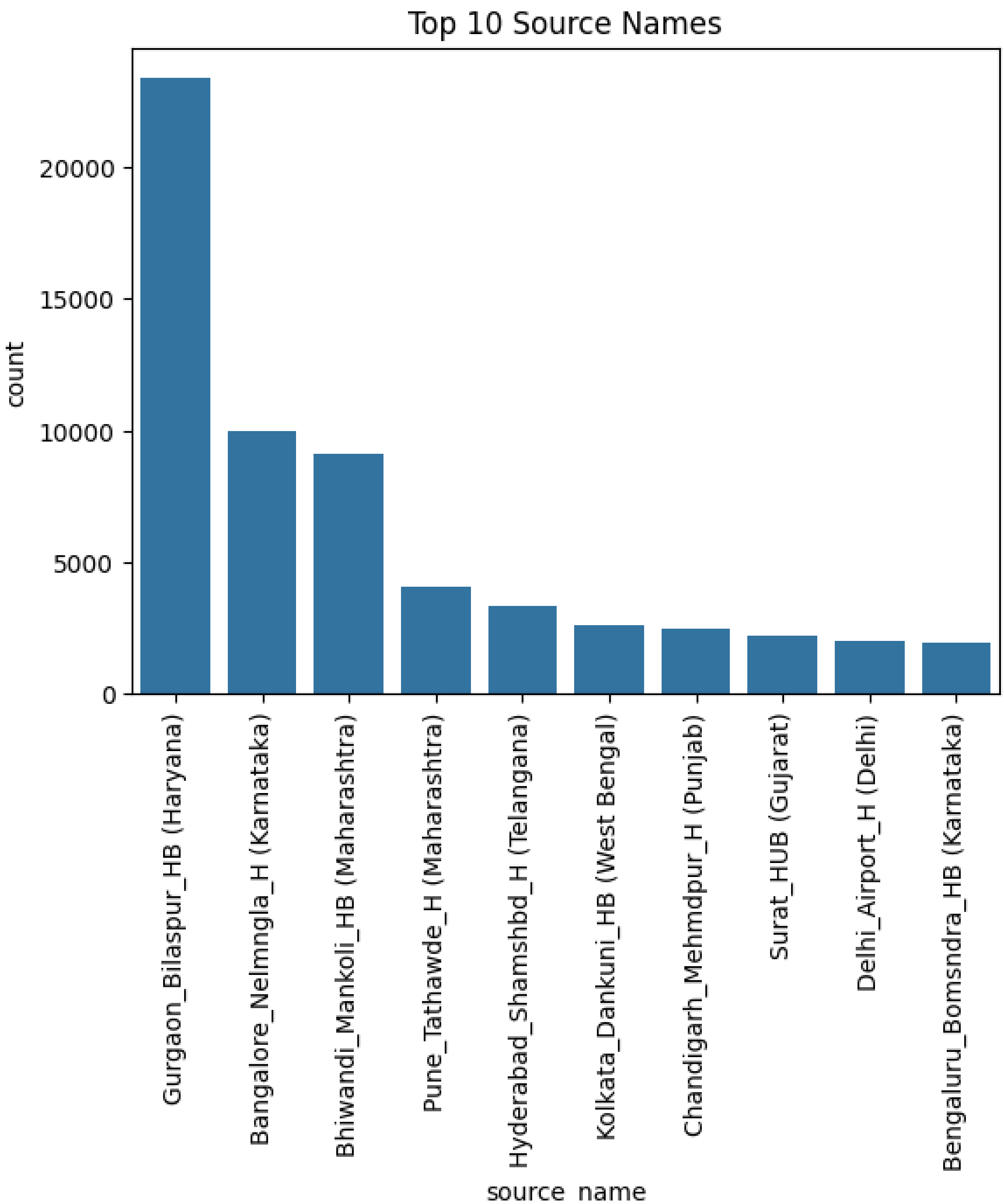
In [125…

```
top_sources = df['source_name'].value_counts().nlargest(10).index
print(top_sources)
```

```
Index(['Gurgaon_Bilaspur_HB (Haryana)', 'Bangalore_Nelmngla_H (Karnataka)',
       'Bhiwandi_Mankoli_HB (Maharashtra)', 'Pune_Tathawde_H (Maharashtra)',
       'Hyderabad_Shamshbd_H (Telangana)', 'Kolkata_Dankuni_HB (West Benga
l)',
       'Chandigarh_Mehmdpur_H (Punjab)', 'Surat_HUB (Gujarat)',
       'Delhi_Airport_H (Delhi)', 'Bengaluru_Bomsndra_HB (Karnataka)'],
      dtype='object', name='source_name')
```

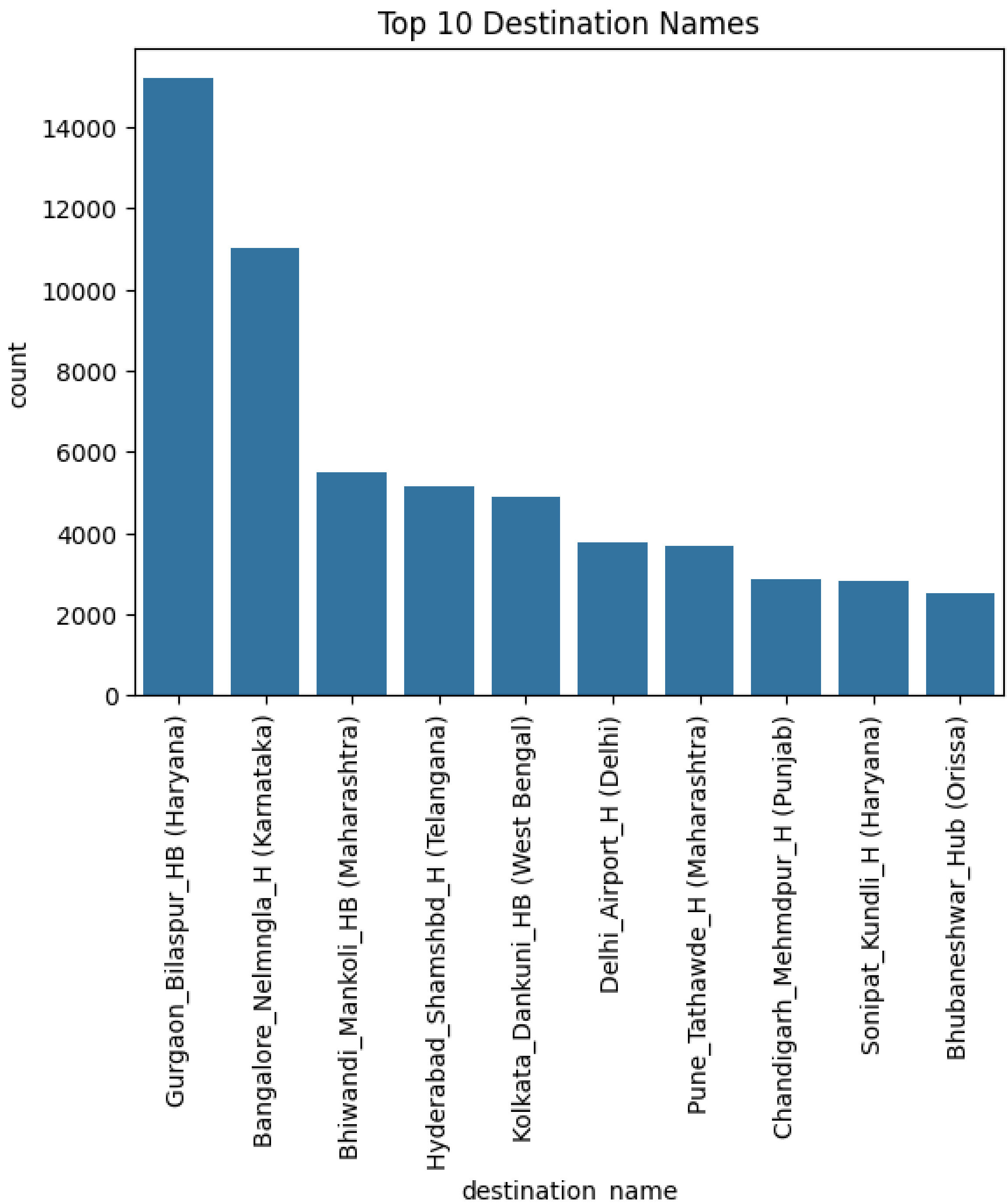
In [126…

```
# Top 10 sources
top_sources = df['source_name'].value_counts().nlargest(10).index

sns.countplot(data=df[df['source_name'].isin(top_sources)], x='source_name',
plt.xticks(rotation=90)  # Optional: rotate labels for better readability
plt.title("Top 10 Source Names")
plt.show()
```



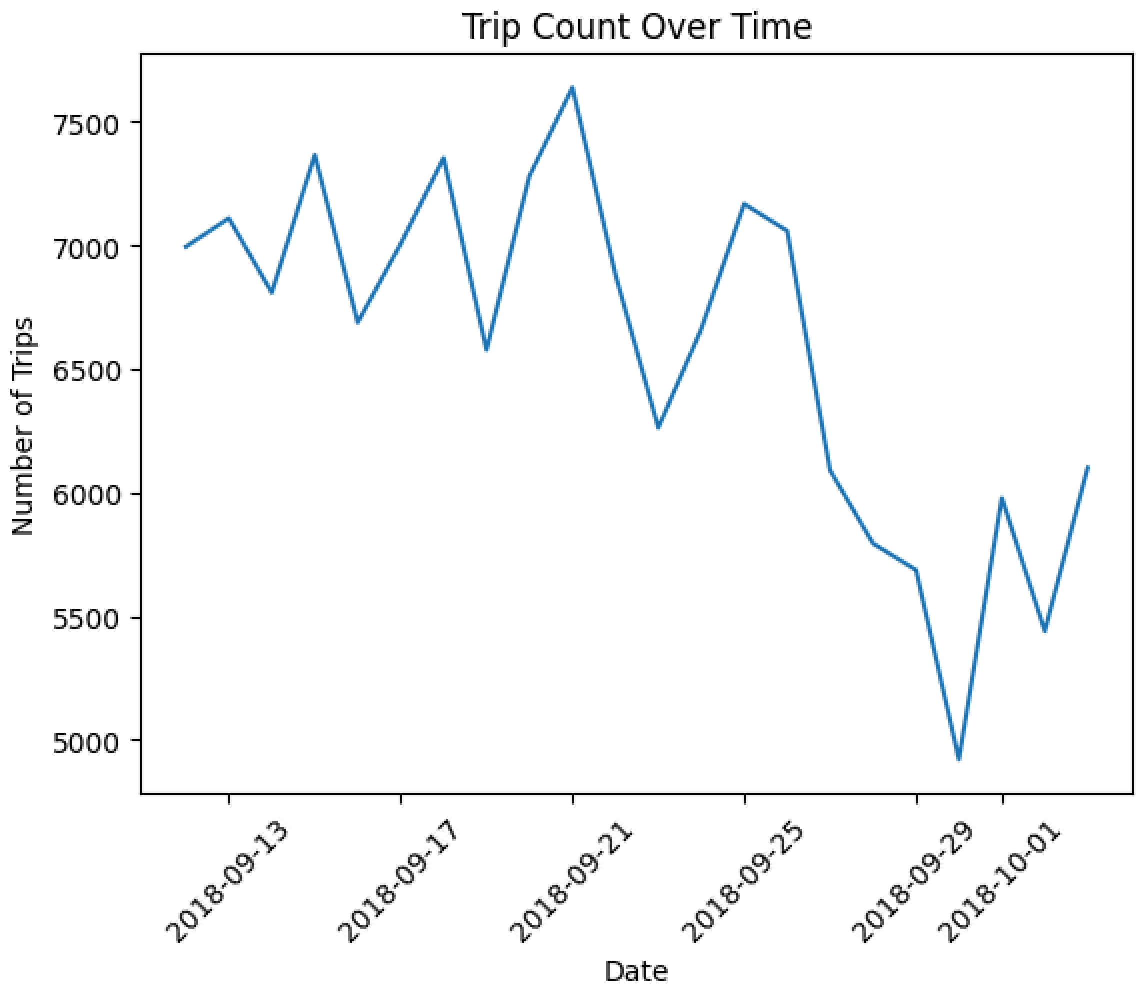
In [127…

```
# Top 10 destination using count plot
top_destinations = df['destination_name'].value_counts().nlargest(10).index
sns.countplot(data=df[df['destination_name'].isin(top_destinations)], x='des
plt.xticks(rotation=90)
plt.title("Top 10 Destination Names")
plt.show()
```



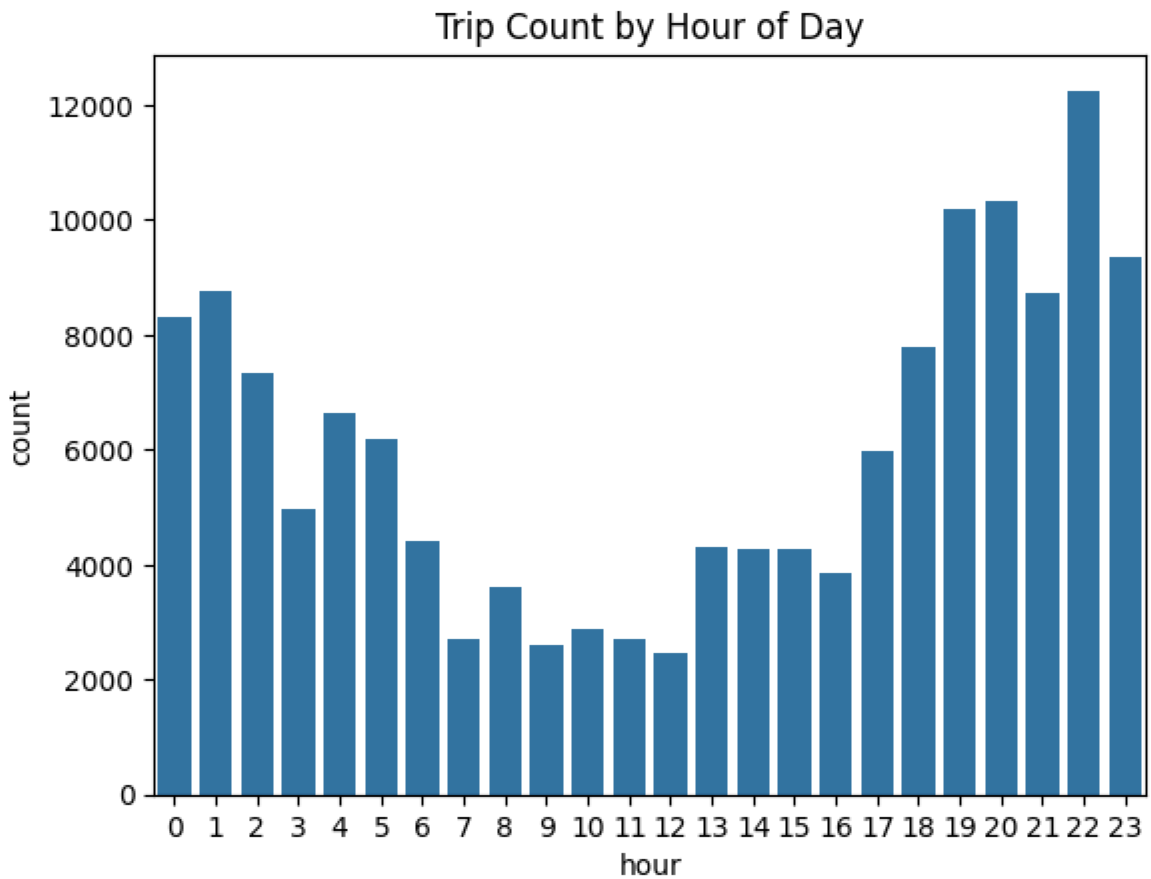
In [128…

```
df['trip_creation_date'] = df['trip_creation_time'].dt.date
df.groupby('trip_creation_date').size().plot(kind='line')
plt.title("Trip Count Over Time")
plt.xlabel("Date")
plt.ylabel("Number of Trips")
plt.xticks(rotation=45)
plt.show()
```



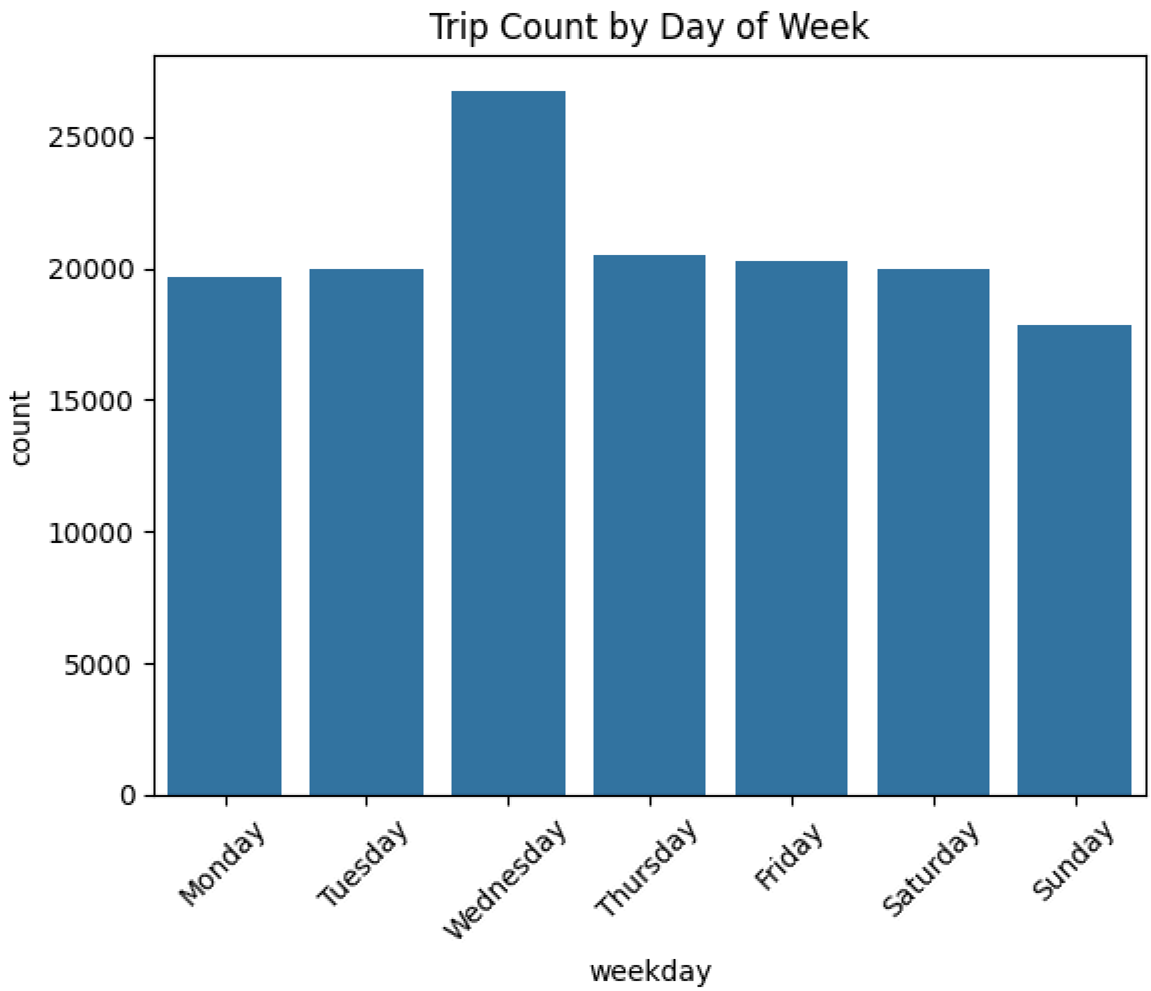
In [129…

```
# Trip count by hour of day
df['hour'] = df['trip_creation_time'].dt.hour

sns.countplot(x='hour', data=df)
plt.title("Trip Count by Hour of Day")
plt.show()
```



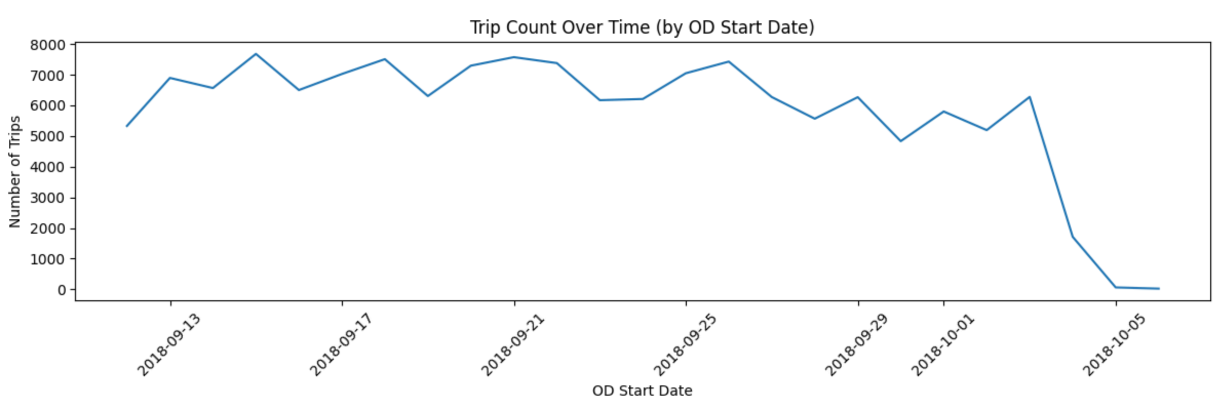
In [130…

```
# Trip Count by Day of week
df['weekday'] = df['trip_creation_time'].dt.day_name()

sns.countplot(x='weekday', data=df, order=[
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sur
])
plt.xticks(rotation=45)
plt.title("Trip Count by Day of Week")
plt.show()
```



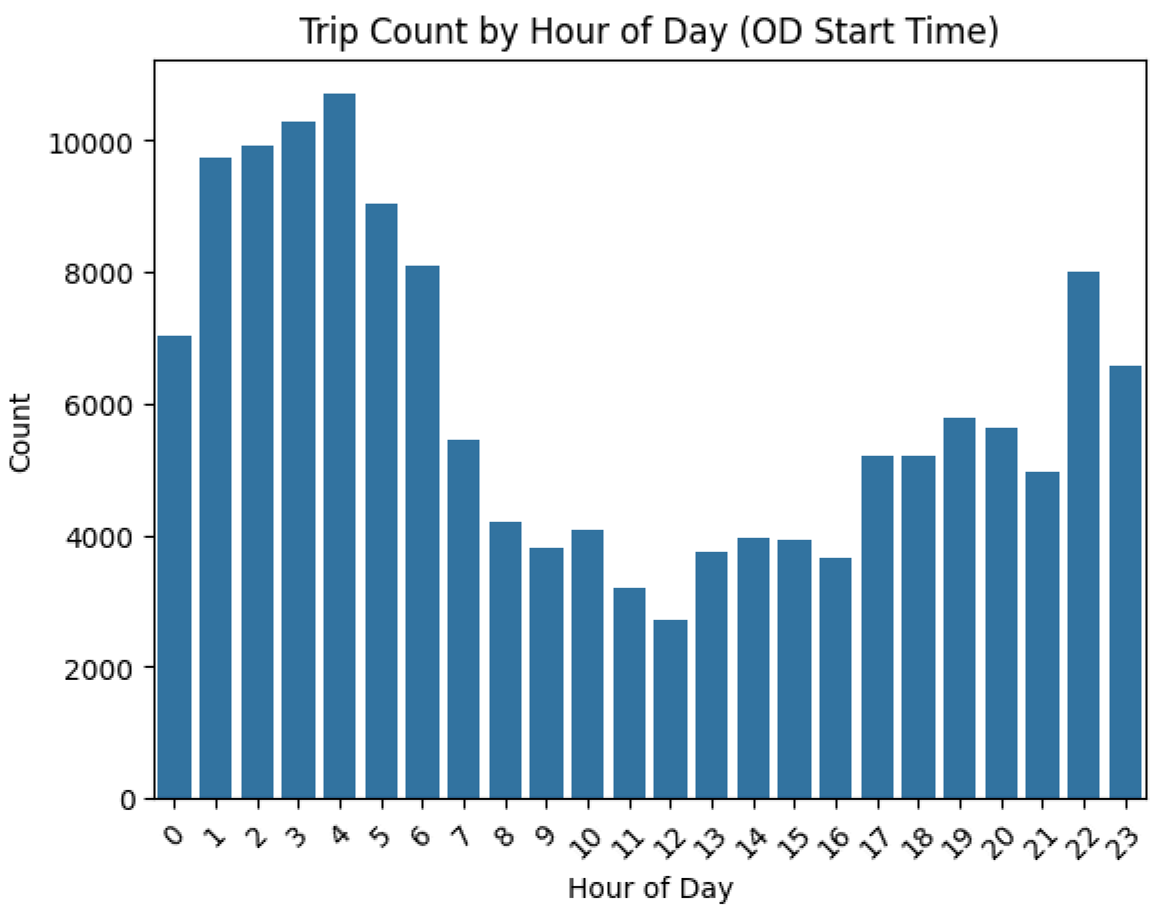
In [131…

```
# Trip Count over Time
df['od_date'] = df['od_start_time'].dt.date

df.groupby('od_date').size().plot(kind='line', figsize=(12, 4))
plt.title("Trip Count Over Time (by OD Start Date)")
plt.xlabel("OD Start Date")
plt.ylabel("Number of Trips")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [132…

```
# Trip Count by Hour of Day
df['od_hour'] = df['od_start_time'].dt.hour

sns.countplot(x='od_hour', data=df)
plt.title("Trip Count by Hour of Day (OD Start Time)")
plt.xlabel("Hour of Day")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

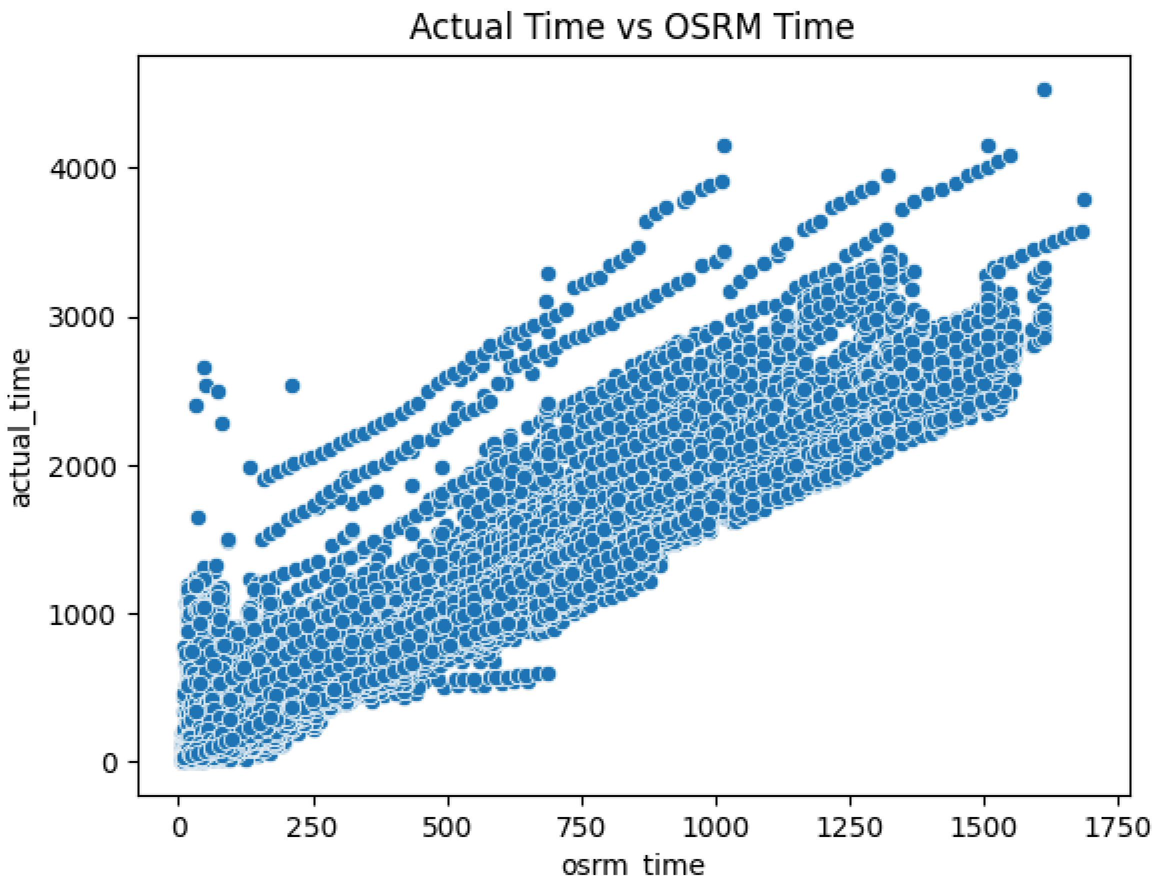


Bivariate Analysis

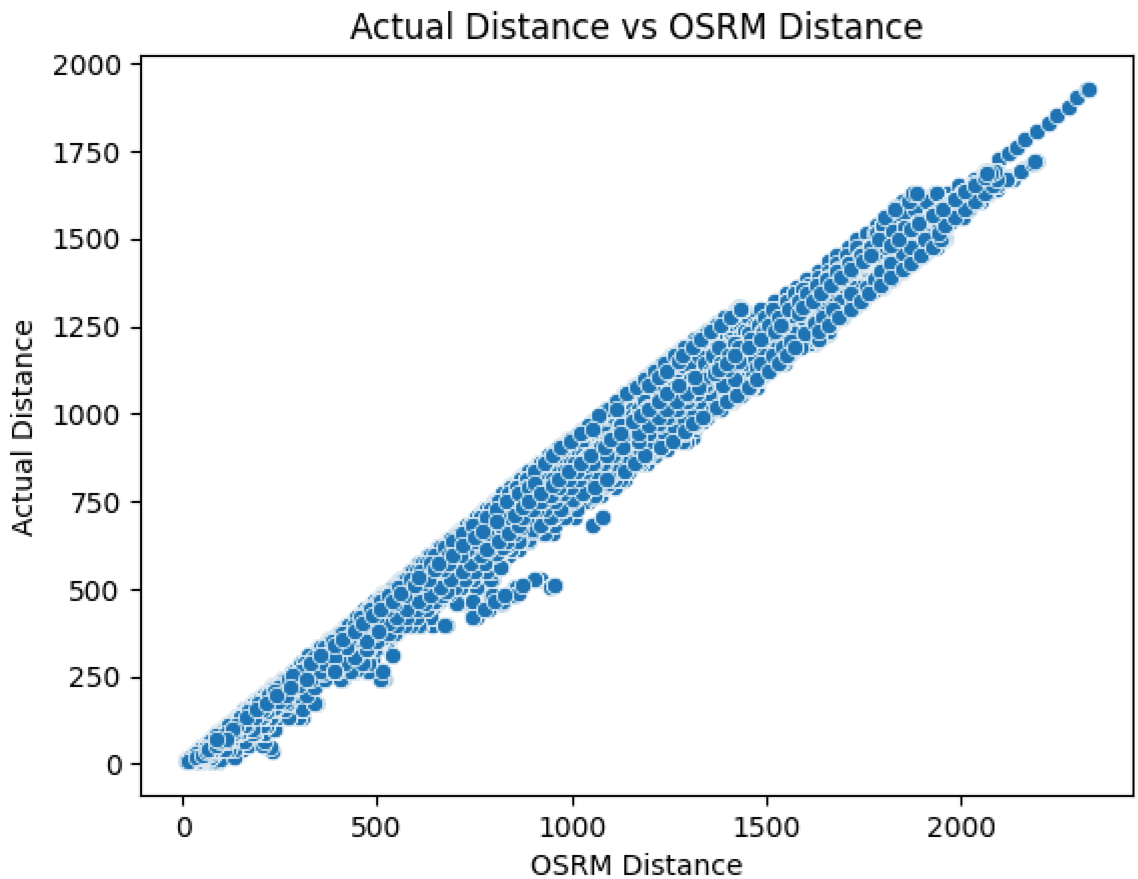
Numerical Vs Numerical

In [133…

```
sns.scatterplot(x='osrm_time', y='actual_time', data=df)
plt.title("Actual Time vs OSRM Time")
plt.show()
```

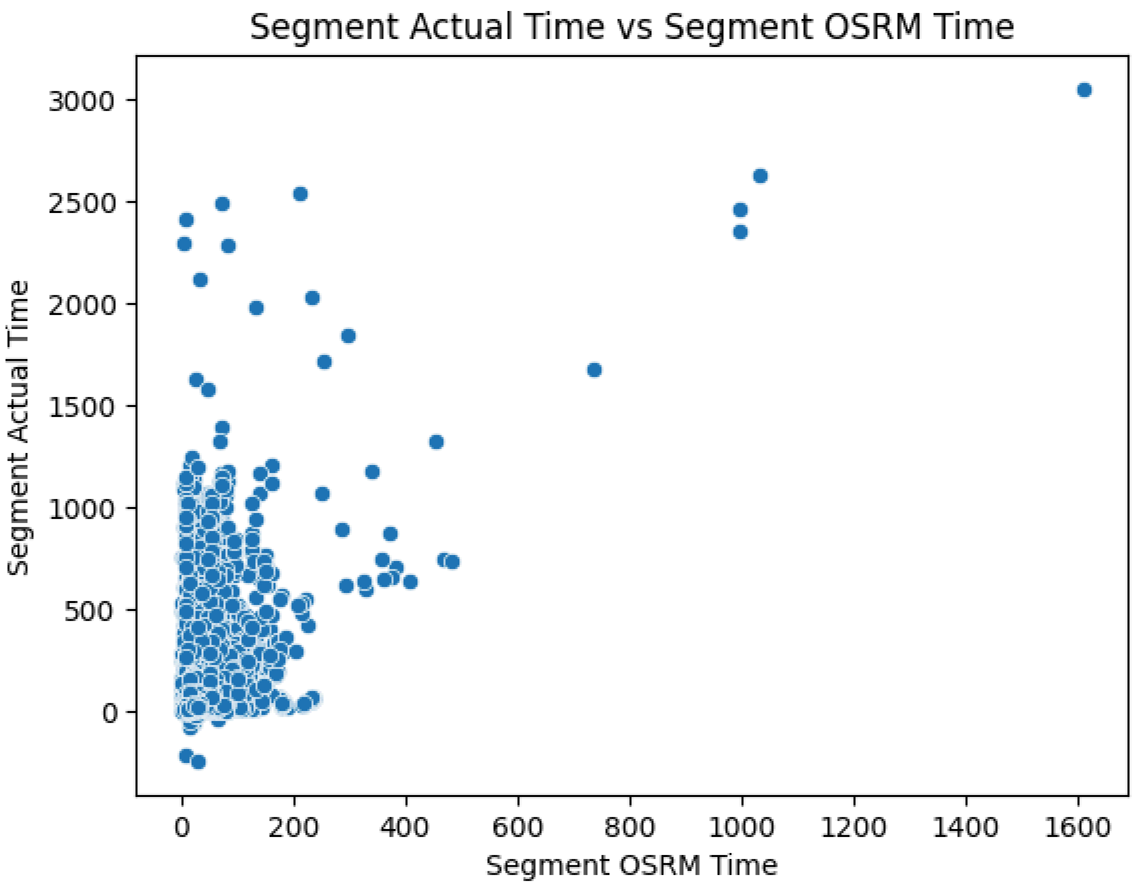


In [134…
```
sns.scatterplot(data=df, x='osrm_distance', y='actual_distance_to_destinatic
plt.title("Actual Distance vs OSRM Distance")
plt.xlabel("OSRM Distance")
plt.ylabel("Actual Distance")
plt.show()
```



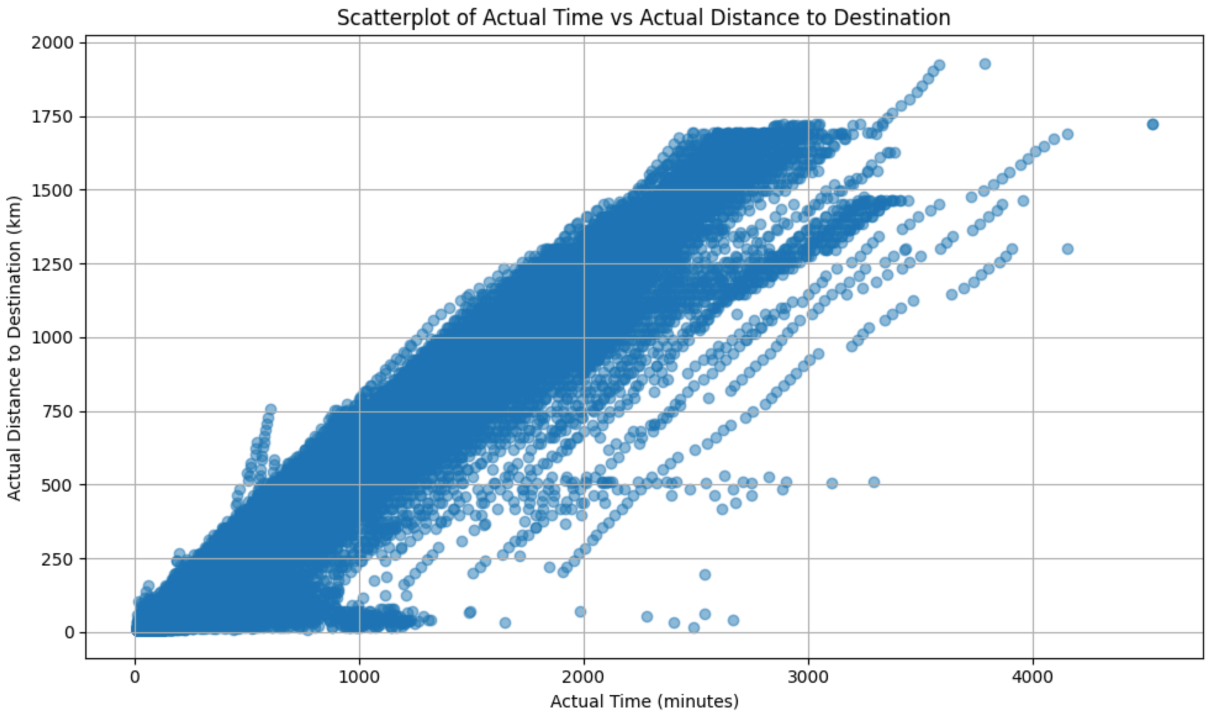
In [135…

```
sns.scatterplot(data=df, x='segment_osrm_time', y='segment_actual_time')
plt.title("Segment Actual Time vs Segment OSRM Time")
plt.xlabel("Segment OSRM Time")
plt.ylabel("Segment Actual Time")
plt.show()
```



In [136…

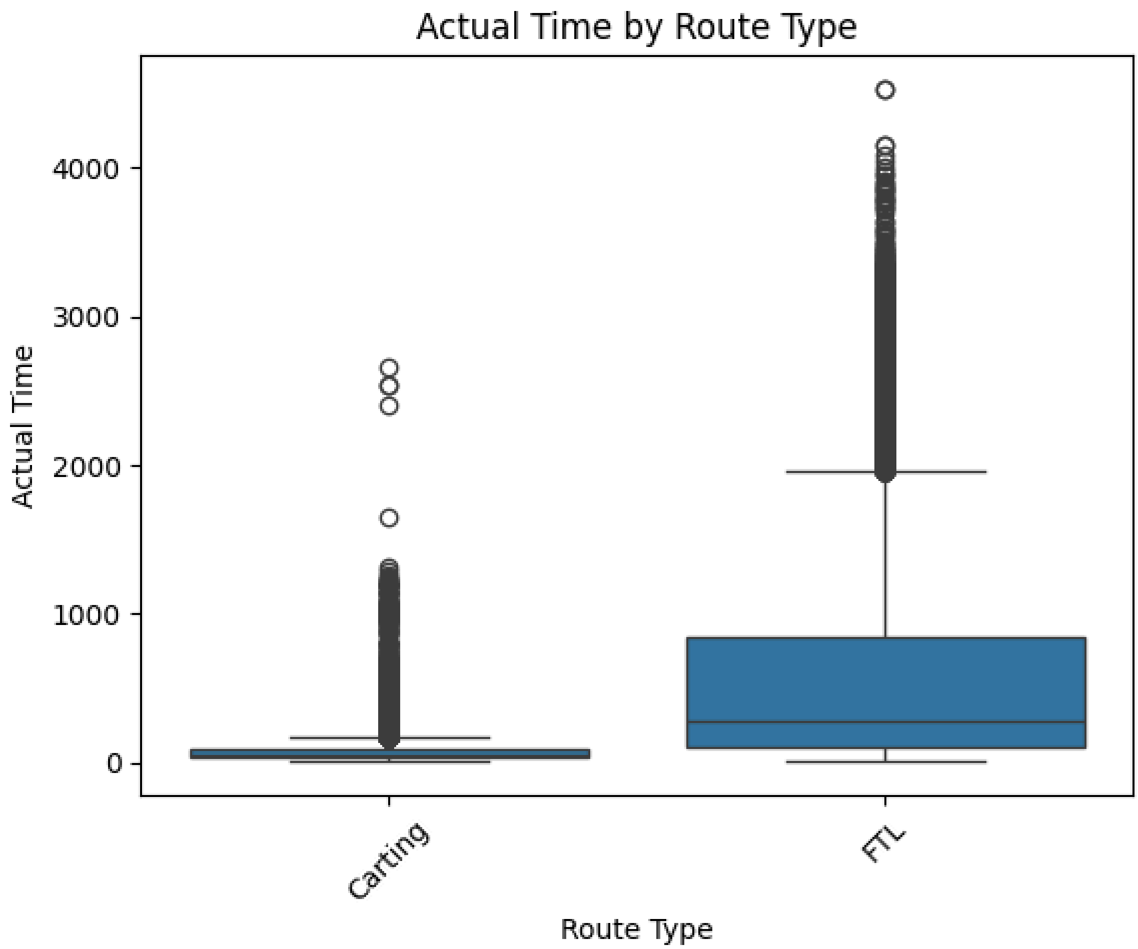
```
# Plotting scatterplot of actual_time vs actual_distance_to_destination
plt.figure(figsize=(10, 6))
plt.scatter(y=df['actual_distance_to_destination'], x=df['actual_time'], alp
plt.ylabel('Actual Distance to Destination (km)')
plt.xlabel('Actual Time (minutes)')
plt.title('Scatterplot of Actual Time vs Actual Distance to Destination')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Numerical Vs Categorical

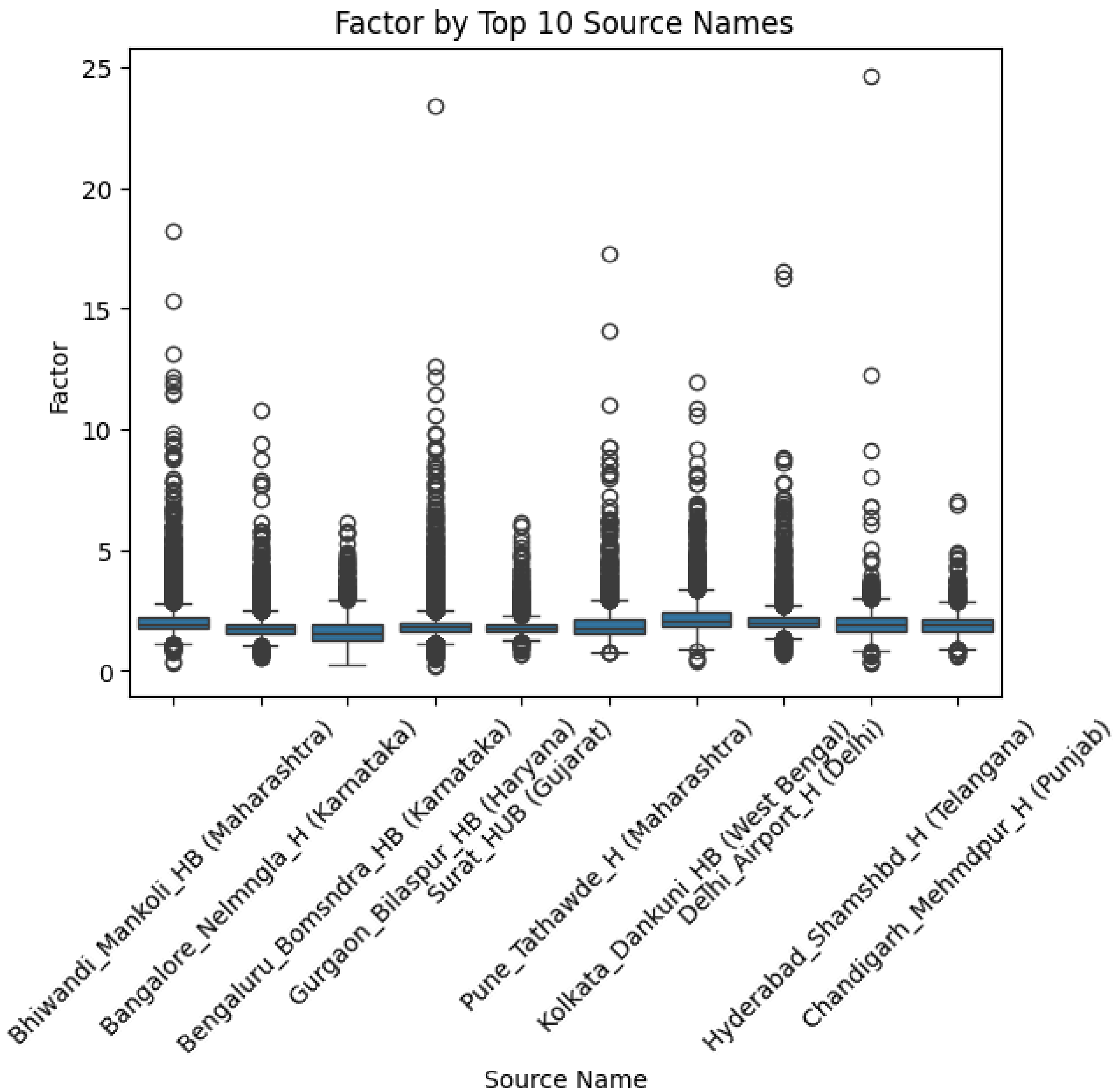
In [137…

```
sns.boxplot(data=df, x='route_type', y='actual_time')
plt.title("Actual Time by Route Type")
plt.xlabel("Route Type")
plt.ylabel("Actual Time")
plt.xticks(rotation=45)
plt.show()
```



In [138…

```
top_sources = df['source_name'].value_counts().nlargest(10).index
filtered_df = df[df['source_name'].isin(top_sources)]

sns.boxplot(data=filtered_df, x='source_name', y='factor')
plt.title("Factor by Top 10 Source Names")
plt.xlabel("Source Name")
plt.ylabel("Factor")
plt.xticks(rotation=45)
plt.show()
```



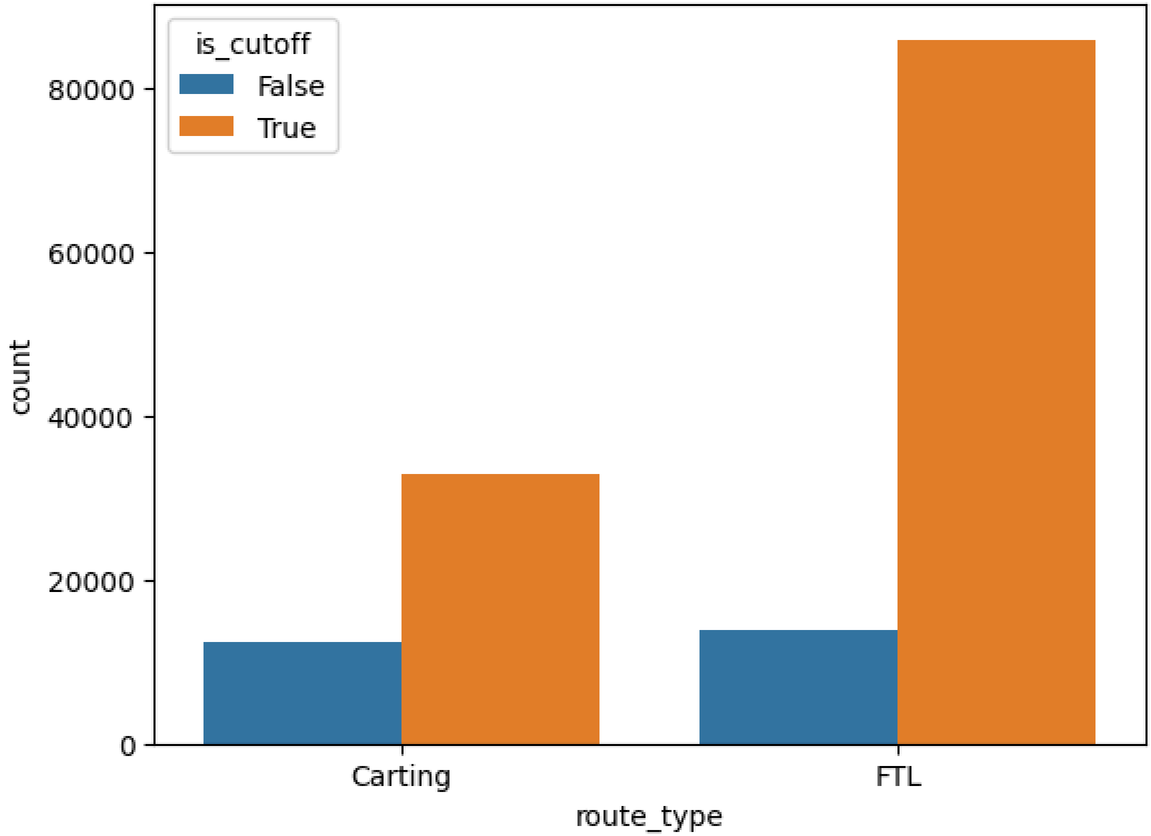
Categorical Vs Categorical

In [139…

```
sns.countplot(x='route_type', hue='is_cutoff', data=df)
```

Out[139…

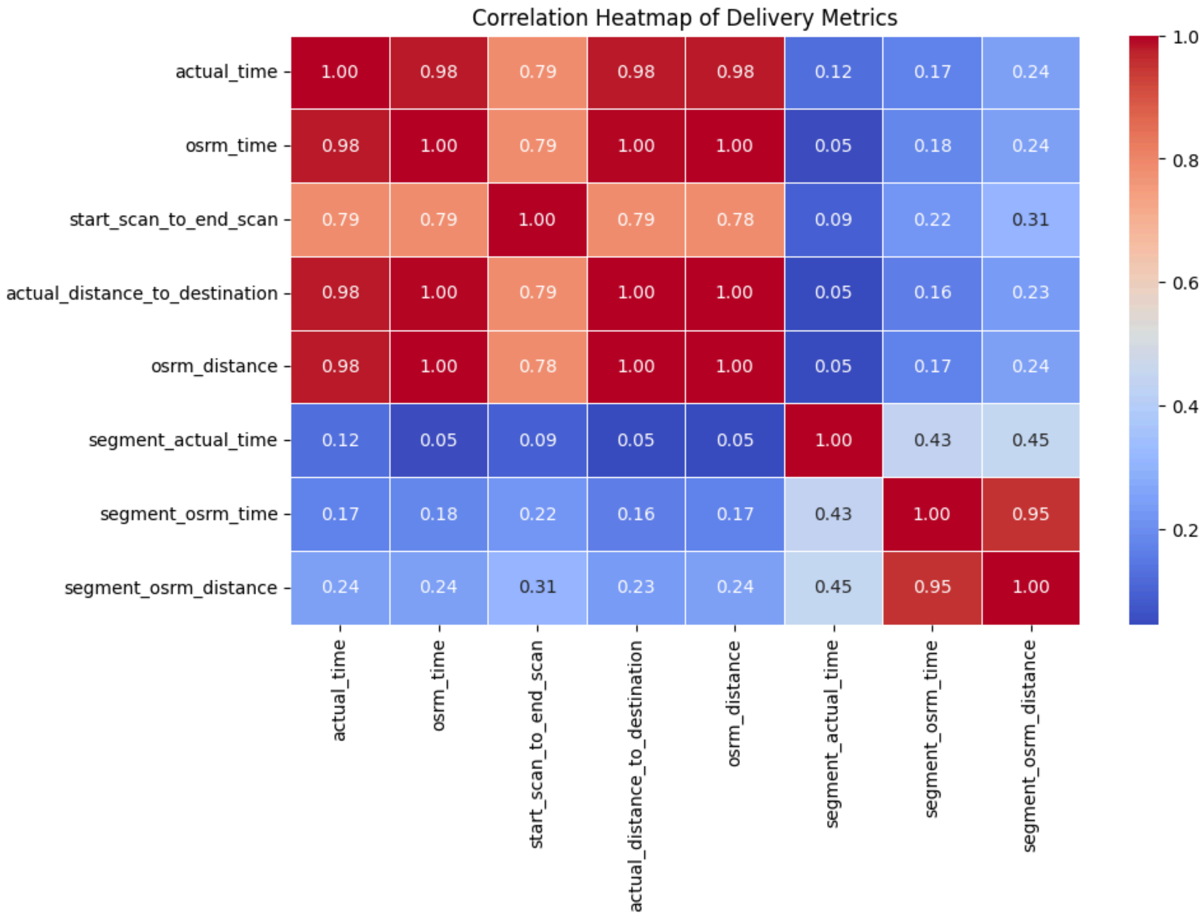
```
<Axes: xlabel='route_type', ylabel='count'>
```



Correlation Between Variables

In [140…

```
# Select numerical columns for correlation
num_cols = [
    'actual_time',
    'osrm_time',
    'start_scan_to_end_scan',
    'actual_distance_to_destination',
    'osrm_distance',
    'segment_actual_time',
    'segment_osrm_time',
    'segment_osrm_distance'
]

# Compute correlation matrix
corr_matrix = df[num_cols].corr()

# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=
plt.title("Correlation Heatmap of Delivery Metrics")
plt.show()
```



Comments on Range of Attributes

1.**start_scan_to_end_scan** – Time taken to deliver from source to destination The data is right-skewed, meaning most deliveries took less time and a few took much longer.

Average time taken for delivery: 961.26 minutes

Minimum time: 20 minutes, Maximum time: 7898 minutes

Most deliveries were clustered below 1000 minutes, but some extreme cases took much longer.

2.**actual_distance_to_destination** – Actual distance (in km) between source and destination warehouses Average distance: 234.07 km

Minimum distance: 9 km, Maximum distance: 1927 km

Most deliveries were mid-range, but some long-haul deliveries are present.

3.**actual_time** – Actual cumulative time taken to complete the delivery Average time: 416.93 minutes

Minimum: 9 minutes, Maximum: 4532 minutes

Like other time fields, this is also right-skewed.

Actual delivery time varies widely, showing possible differences in route type or delays.

4.**osrm_time** – Estimated time from OSRM (routing engine like Google Maps) Average estimated time: 213.87 minutes

Minimum: 6 minutes, Maximum: 1686 minutes

Compared to actual time, on average deliveries took ~200 minutes longer than expected.

This gap may be due to traffic, road closures, delays, or driver behavior.

5.**osrm_distance** – Estimated shortest path by OSRM (in km) Average distance: 284.77 km

Minimum: 9 km, Maximum: 2326.20 km

Often longer than actual_distance_to_destination, possibly due to routing logic.

6.**Comparison**: osrm_distance vs actual_distance_to_destination Average gap: ~50 km

Possible reasons:

OSRM calculates shortest path, but drivers may take alternate routes

Road construction, diversions, or GPS inaccuracies

Mistakes like missed exits or U-turns could increase actual distance

7.**segment_osrm_time** – Estimated time for a delivery segment Average: 18.51 minutes

Minimum: 0 minutes, Maximum: 1611 minutes

This reflects estimates for only part of the trip, not the full journey.

8.**segment_actual_time** – Actual time taken for a segment Average: 36.20 minutes, Maximum: 3051 minutes

On average, segment actual time is nearly double the OSRM estimated time

This again shows a consistent delay between estimated and real delivery times.

9.**segment_osrm_distance** – Estimated segment distance (by OSRM) Average: 22.83 km

Minimum: 0 km, Maximum: 2191.40 km

Comments on Univariate Analysis

1.Route Type The majority of deliveries fall under the FTL (Full Truck Load) category, with approximately 30,000 deliveries.

The Carting route type accounts for around 14,000 deliveries.

2.Source Name Gurgaon_Bilaspur_HB (Haryana) is the most active source warehouse, with over 7,000 deliveries initiated from this location.

It is followed by Bhiwandi_Mankoli_HB (Maharashtra) and Bangalore_Nelmangla_H (Karnataka), each with 2,000+ deliveries.

3.Destination Name Gurgaon_Bilaspur_HB (Haryana) again stands out as the top destination warehouse, receiving the highest number of deliveries.

It is followed by Bangalore_Nelmangla_H (Karnataka) and Hyderabad_Shamshabad_H (Telangana).

4.Creation Date & Hour The maximum number of orders were booked on 21st September 2018.

The lowest number of orders were booked on 1st October.

Most orders are booked between 6 PM and 5 AM, suggesting operational focus during evening and night hours.

Orders continue during the day but at a relatively lower volume.

5.Creation Day Wednesday shows the highest booking volume, followed by Friday.

Other weekdays show a fairly consistent booking pattern with no significant drop.

6.OD Start Time Most trips are initiated between 10 PM and 6 AM, likely to avoid daytime traffic and optimize logistics.

Only a few trips start between 8 AM and 6 PM, possibly due to peak traffic hours or business operational preferences.

Comments on the Distribution of the Variables

1.Actual Time The histogram shows that the data is right-skewed. Most deliveries took less time, while a few took a significantly longer time.

2.OSRM Time This is also right-skewed. The OSRM predicted time is low for most deliveries but much higher for a few.

3.OSRM Distance The distribution is right-skewed. The expected distance (as per OSRM) is low for most deliveries but quite high for some.

4.Actual Distance to Destination This variable also has a right-skewed distribution. Most deliveries covered shorter distances, while a few covered very long distances.

5.Start Scan to End Scan Time The graph is right-skewed as well. The total time taken from the first scan to the last scan is low for most deliveries, but very high in a few cases.

Bivariate Analysis

Actual Time vs. OSRM Time

The scatterplot suggests a positive correlation between actual time and OSRM time. As OSRM time increases, actual time also tends to increase, indicating that OSRM time is a good predictor of actual time in many cases.

Actual Time vs. Actual Distance to Destination:

There is a clear positive relationship — longer actual distances generally take more time to cover, as expected.

OSRM Distance vs. Actual Distance to Destination:

These two variables are strongly related. As OSRM distance increases, actual distance also increases. This implies the routing engine (OSRM) gives reasonably accurate distance estimations.

Segment OSRM Time vs. Actual Time:

There appears to be no strong correlation between segment OSRM time and overall actual time. This suggests that segment-level estimations may not capture trip-level performance well.

📊 **Correlation (Heatmap) Insights** There is a high correlation between actual time and OSRM time.

A strong positive correlation is observed between:

OSRM time and actual distance to destination

OSRM distance and actual distance to destination

A high correlation is also visible between actual time and actual distance to destination.

Segment-level variables (like segment OSRM time) tend to have weaker correlations with overall trip-level metrics.

Feature Engineering

In [141…

```
## Create new column to understand how many states are present in source anc
df['destination_state'] = df['destination_name'].str.extract(r'\((.*?)\)')
```

In [142…

```
df.destination_state.value_counts()
```

Out[142…

destination_state	count
Karnataka	21065
Haryana	20622
Maharashtra	18196
West Bengal	8499
Telangana	8205
Tamil Nadu	8058
Uttar Pradesh	7834
Gujarat	6714
Rajasthan	6361
Andhra Pradesh	6265
Delhi	5754
Punjab	5105
Madhya Pradesh	4345
Bihar	4238
Orissa	3234
Jharkhand	2552
Kerala	2230
Assam	2000
Uttarakhand	893
Goa	580
Himachal Pradesh	553
Chandigarh	389
Chhattisgarh	229
Arunachal Pradesh	211
Jammu & Kashmir	201
Pondicherry	154
Meghalaya	37
Dadra and Nagar Haveli	34
Mizoram	31
Tripura	9
Nagaland	7
Daman & Diu	1

dtype: int64

In [143…

```
## Create new column to understand how many states are present in source
df['source_state'] = df['source_name'].str.extract(r'\((.*?)\)')
```

In [144…

```
df.source_state.value_counts()
```

Out[144…

source_state	count
Haryana	27499
Maharashtra	21401
Karnataka	19578
Tamil Nadu	7494
Gujarat	7202
Uttar Pradesh	7137
Telangana	6496
West Bengal	5963
Andhra Pradesh	5539
Rajasthan	5267
Punjab	4704
Delhi	4398
Bihar	4190
Madhya Pradesh	4021
Assam	2875
Jharkhand	2597
Kerala	2413
Orissa	2094
Uttarakhand	1162
Himachal Pradesh	587
Goa	514
Chandigarh	507
Arunachal Pradesh	245
Chhattisgarh	229
Jammu & Kashmir	226
Meghalaya	86
Pondicherry	49
Nagaland	40
Dadra and Nagar Haveli	30
Mizoram	26
Tripura	5

dtype: int64

In [145…
```
# Create feature like month, year and day from Trip_creation_time
df['trip_creation_date'] = pd.to_datetime(df['trip_creation_time']).dt.date
df['trip_creation_year'] = pd.to_datetime(df['trip_creation_time']).dt.year
df['trip_creation_month'] = pd.to_datetime(df['trip_creation_time']).dt.mont
df['trip_creation_day'] = pd.to_datetime(df['trip_creation_time']).dt.day
```

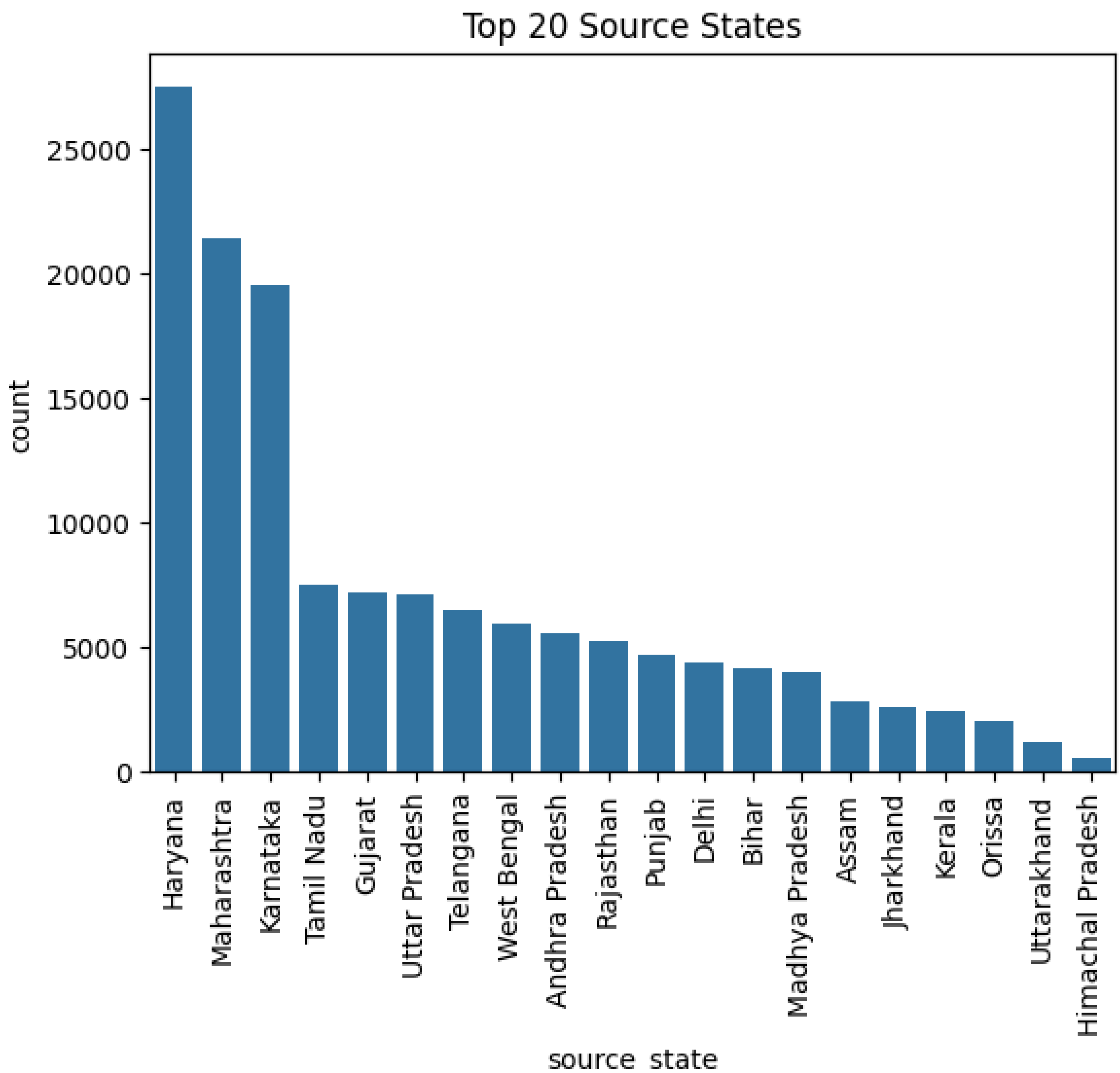
In [146…
```
df.head()
```

Out[146…

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

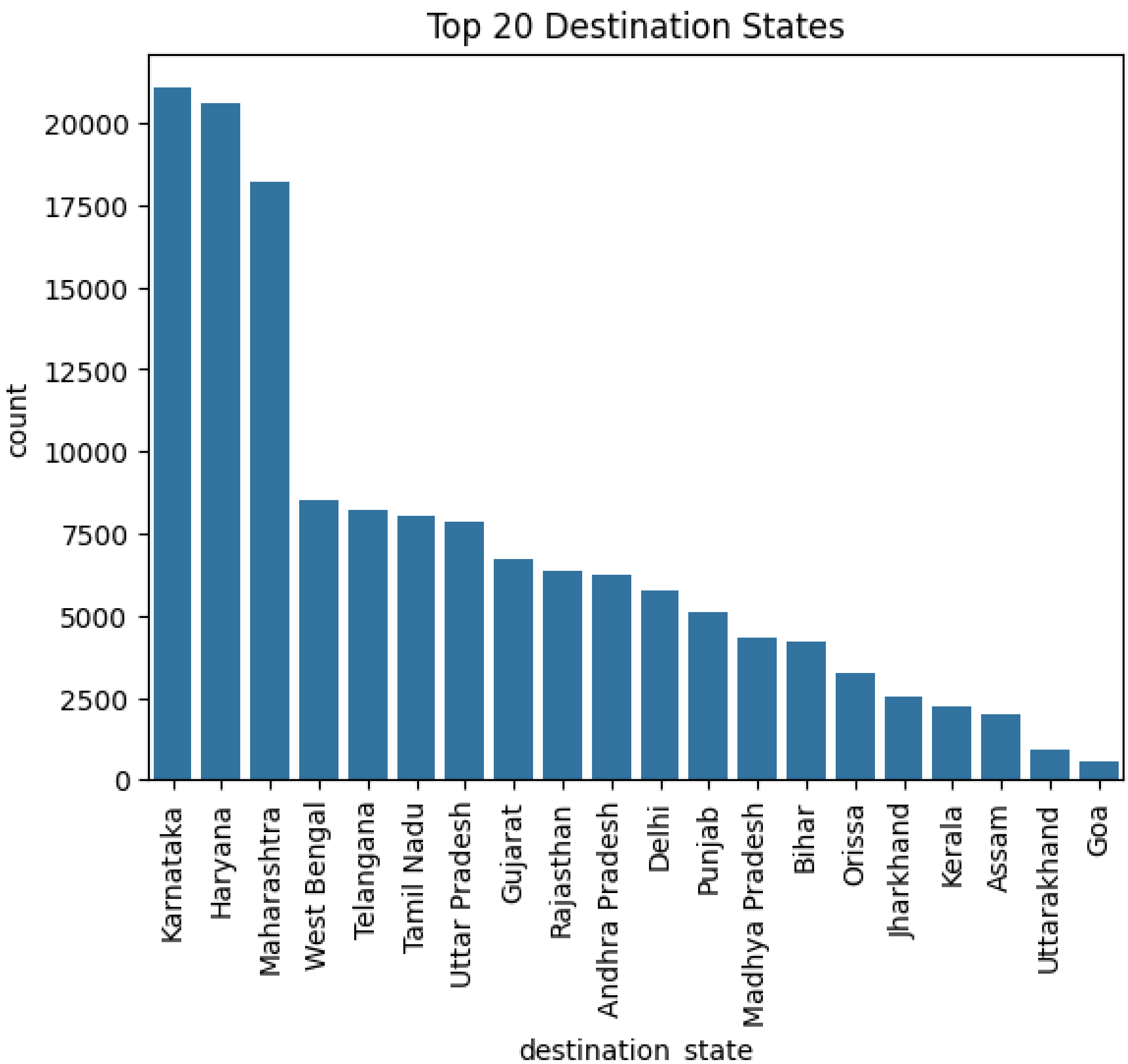
5 rows × 34 columns

In [147…
```
# Plot top 20 states with highest number of orders at source
top_states = df['source_state'].value_counts().nlargest(20).index
sns.countplot(data=df[df['source_state'].isin(top_states)], x='source_state'
plt.xticks(rotation=90)
plt.title("Top 20 Source States")
plt.show()
```



In [148…

```
# Plot top 20 states with highest number of orders delivered at destination
top_states = df['destination_state'].value_counts().nlargest(20).index
sns.countplot(data=df[df['destination_state'].isin(top_states)], x='destinat
plt.xticks(rotation=90)
plt.title("Top 20 Destination States")
plt.show()
```



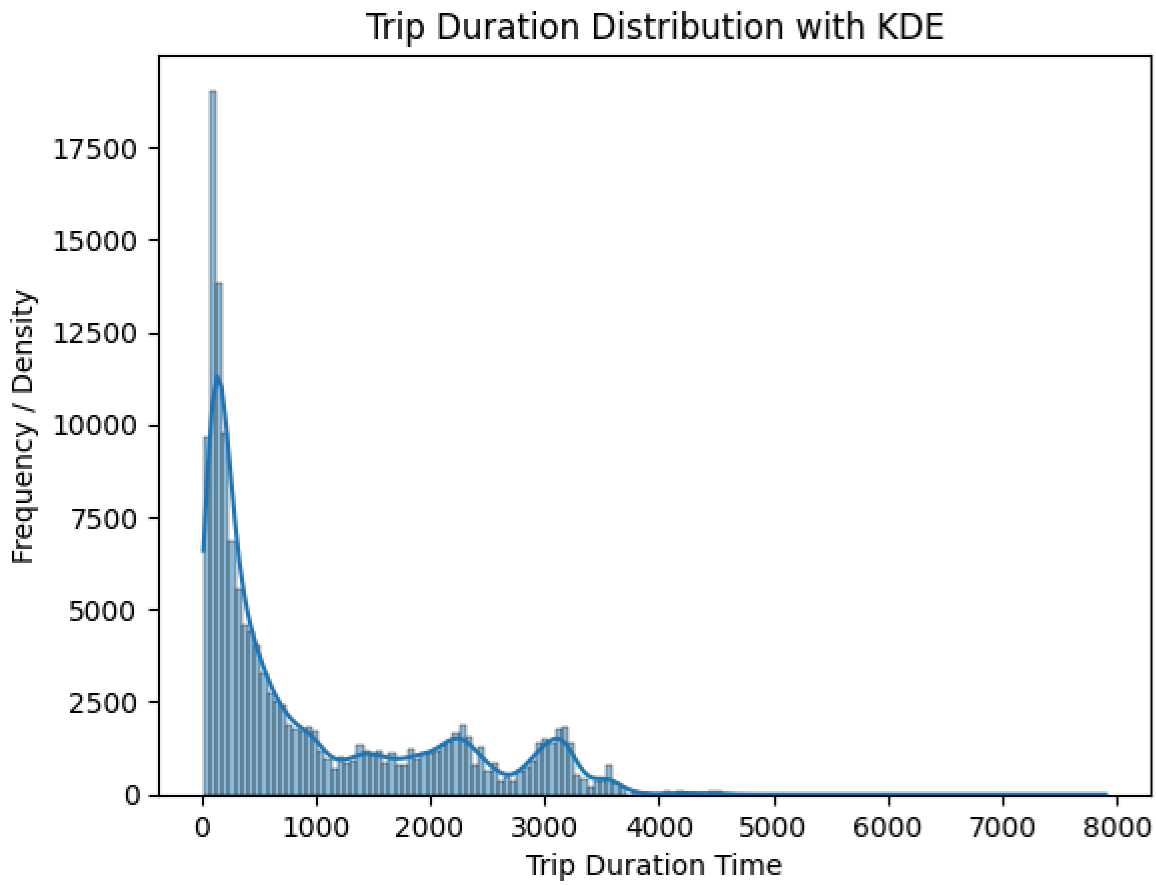
New feature Trip_duration_time which is difference of od_start_time and od_end_time in minutes

In [149…

```
df['trip_duration_time'] = (pd.to_datetime(df['od_end_time']) - pd.to_dateti
```

In [150…

```
# histplot to check how much is trip_duration_time
sns.histplot(df['trip_duration_time'], kde=True)
plt.title("Trip Duration Distribution with KDE")
plt.xlabel("Trip Duration Time")
plt.ylabel("Frequency / Density")
plt.show()
```



In [151…

```
print("Average time from start to end",df['trip_duration_time'].mean()) ,pri
```

```
Average time from start to end 961.7590027084818
Maximum time from start to end 7898.551954566667
Minimum time from start to end 20.70281311666667
```

Out[151… (None, None, None)

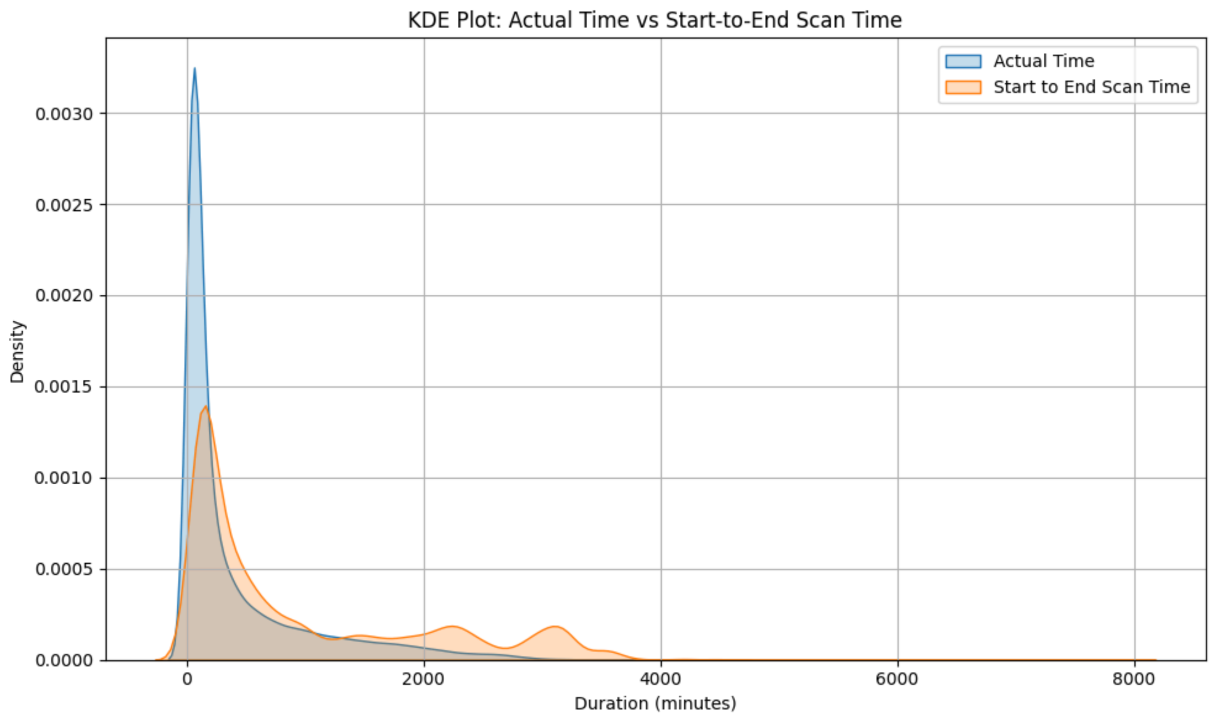
Hypothesis Testing

Compare the difference between Point a. and start_scan_to_end_scan. Hypothesis testing & Visual analysis to check.

In [152…

```
# Convert start_scan_to_end_scan to numeric (if not already)
df['start_scan_to_end_scan'] = pd.to_numeric(df['start_scan_to_end_scan'], e

# Drop NA values for clean comparison
comparison_df = df[['actual_time', 'start_scan_to_end_scan']].dropna()

# Visual comparison using KDE plot
plt.figure(figsize=(10, 6))
sns.kdeplot(comparison_df['actual_time'], label='Actual Time', fill=True)
sns.kdeplot(comparison_df['start_scan_to_end_scan'], label='Start to End Sca
plt.title('KDE Plot: Actual Time vs Start-to-End Scan Time')
plt.xlabel('Duration (minutes)')
plt.legend()
```

```
plt.grid(True)
plt.tight_layout()
plt.show()

# Statistical hypothesis test (independent t-test)

t_stat, p_val = ttest_ind(comparison_df['actual_time'], comparison_df['start

t_stat, p_val

if p_val < 0.05:
    print("Reject the null hypothesis. There is a significant difference in
else:
    print("Fail to reject the null hypothesis. There is no significant diffe
```

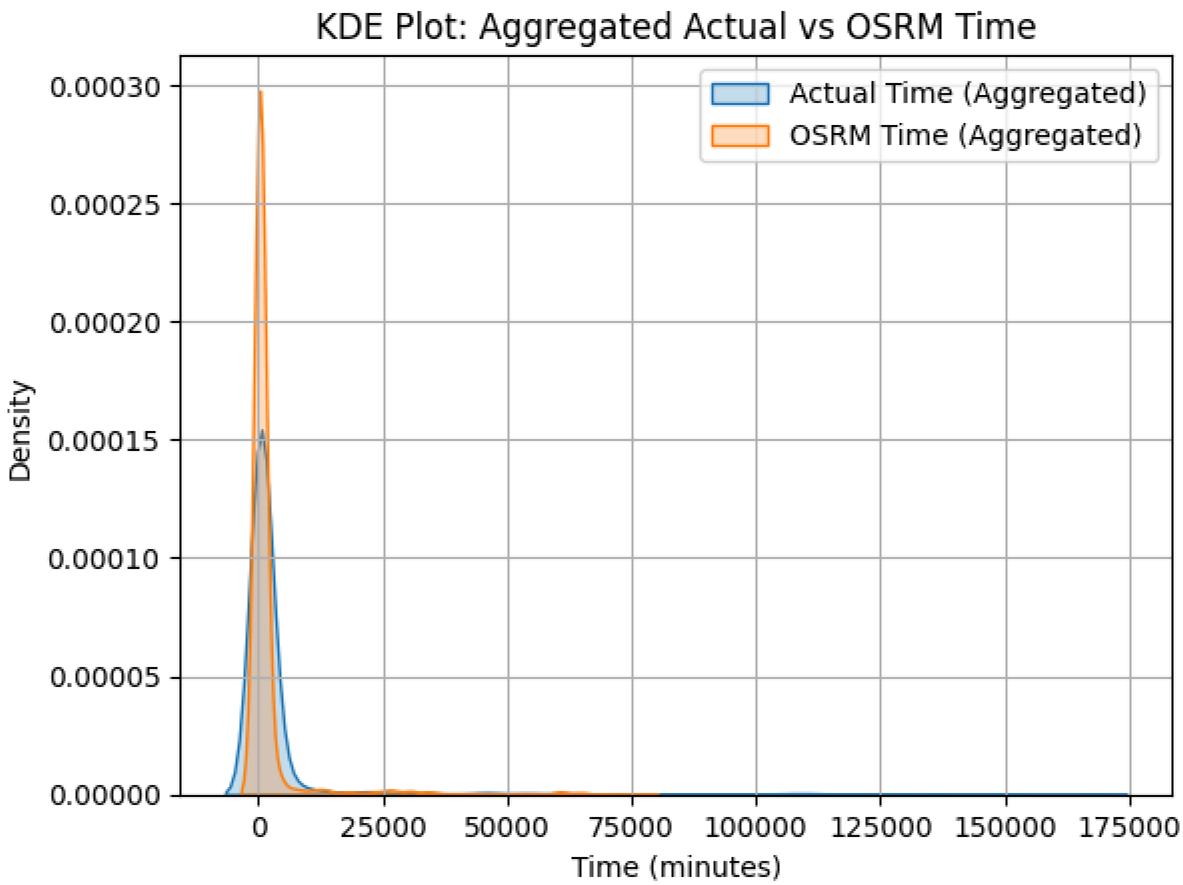



```
Reject the null hypothesis. There is a significant difference in actual time
and start-to-end scan time.
```

Hypothesis testing & visual analysis between actual_time aggregated value and OSRM time aggregated value

In [153…

```
grouped_df = df.groupby('trip_uuid').agg({
    'actual_time': 'sum',
    'osrm_time': 'sum'
}).dropna()

sns.kdeplot(grouped_df['actual_time'], label='Actual Time (Aggregated)', fil
sns.kdeplot(grouped_df['osrm_time'], label='OSRM Time (Aggregated)', fill=Tr
plt.title("KDE Plot: Aggregated Actual vs OSRM Time")
plt.xlabel("Time (minutes)")
plt.legend()
plt.grid(True)
plt.show()
```



In [154…

```
t_stat, p_val = ttest_ind(grouped_df['actual_time'], grouped_df['osrm_time']
print(t_stat, p_val)
if p_val < 0.05:
    print("Reject the null hypothesis. There is a significant difference in
else:
    print("Fail to reject the null hypothesis. There is no significant diffe
```

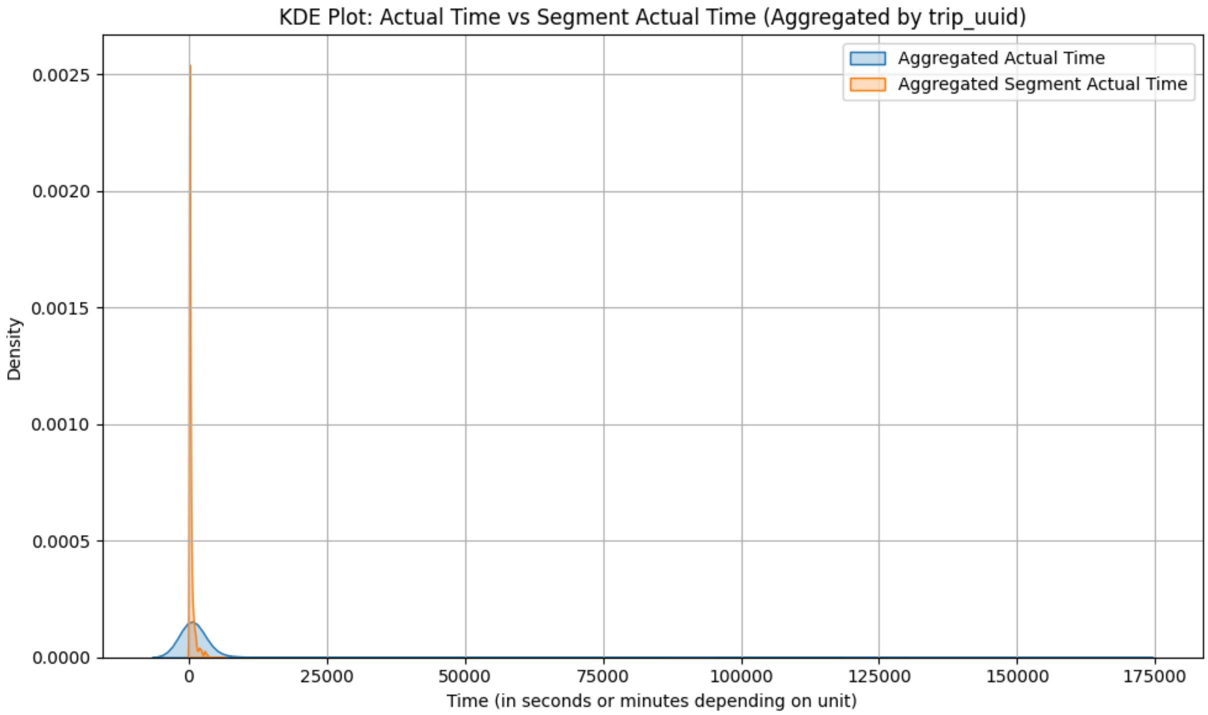
```
14.073444960610715 7.714905383019579e-45
Reject the null hypothesis. There is a significant difference in aggregated
actual time and aggregated OSRM time.
```

Hypothesis testing & visual analysis between actual_time aggregated value and segment actual time aggregated value

In [155…

```
# Group by 'trip_uuid' and aggregate 'actual_time' and 'segment_actual_time'
agg_df = df.groupby('trip_uuid')[['actual_time', 'segment_actual_time']].sum
# KDE Plot for visual comparison
plt.figure(figsize=(10, 6))
sns.kdeplot(agg_df['actual_time'], label='Aggregated Actual Time', fill=True
sns.kdeplot(agg_df['segment_actual_time'], label='Aggregated Segment Actual
plt.title('KDE Plot: Actual Time vs Segment Actual Time (Aggregated by trip_
plt.xlabel('Time (in seconds or minutes depending on unit)')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```python
# Hypothesis Testing: Independent T-Test
t_stat, p_val = ttest_ind(agg_df['actual_time'], agg_df['segment_actual_time

t_stat, p_val

if p_val < 0.05:
    print("Reject the null hypothesis. There is a significant difference in
else:
    print("Fail to reject the null hypothesis. There is no significant diffe
```



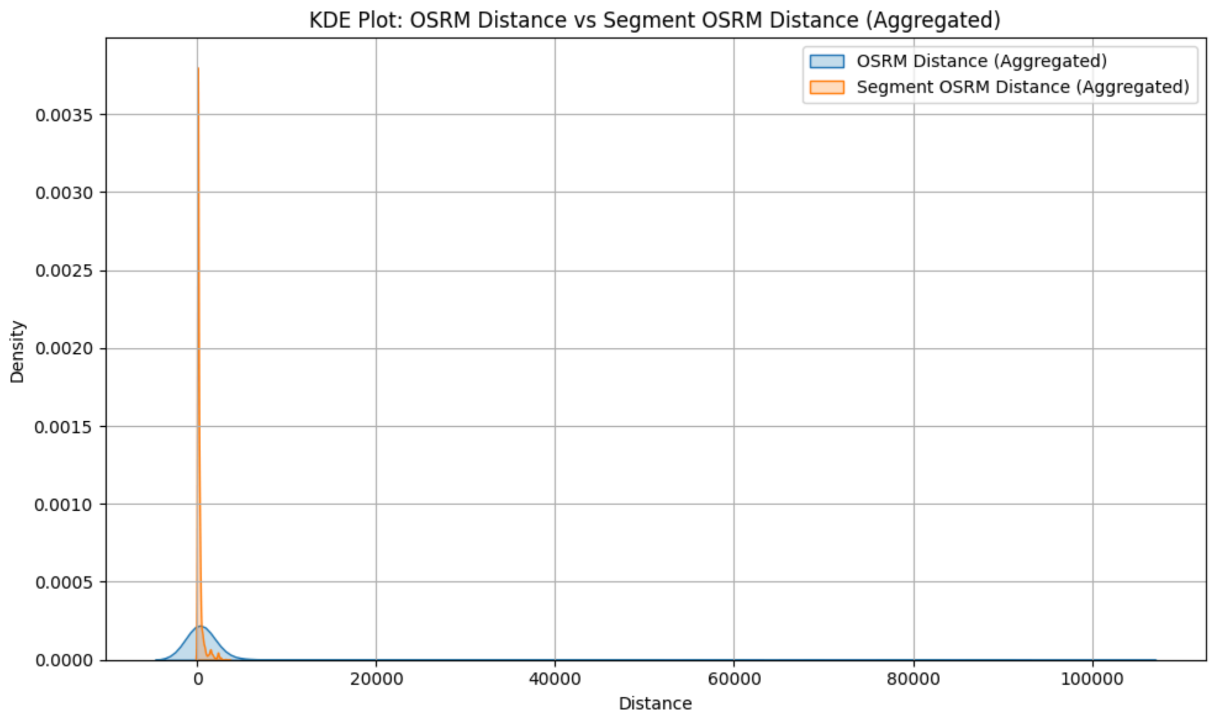

```
Reject the null hypothesis. There is a significant difference in aggregated
actual time and aggregated segment actual time.
```

Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

In [156…

```python
# Group by trip_uuid and sum both distance columns
agg_df = df.groupby('trip_uuid')[['osrm_distance', 'segment_osrm_distance']]

# Convert to numeric and drop non-finite values
agg_df['osrm_distance'] = pd.to_numeric(agg_df['osrm_distance'], errors='coe
agg_df['segment_osrm_distance'] = pd.to_numeric(agg_df['segment_osrm_distanc
agg_df_clean = agg_df[np.isfinite(agg_df['osrm_distance']) & np.isfinite(agg

# 📊 KDE Plot for visual comparison
plt.figure(figsize=(10, 6))
sns.kdeplot(data=agg_df_clean, x='osrm_distance', label='OSRM Distance (Aggr
sns.kdeplot(data=agg_df_clean, x='segment_osrm_distance', label='Segment OSR
plt.title('KDE Plot: OSRM Distance vs Segment OSRM Distance (Aggregated)')
plt.xlabel('Distance')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.tight_layout()
```

```
plt.show()

# 🧪 Hypothesis Testing
t_stat, p_val = ttest_ind(agg_df_clean['osrm_distance'], agg_df_clean['segme
print(f"T-statistic: {t_stat:.4f}, P-value: {p_val:.6f}")

if p_val < 0.05:
    print("Reject the null hypothesis. There is a significant difference in 
else:
    print("Fail to reject the null hypothesis. There is no significant diffe
```



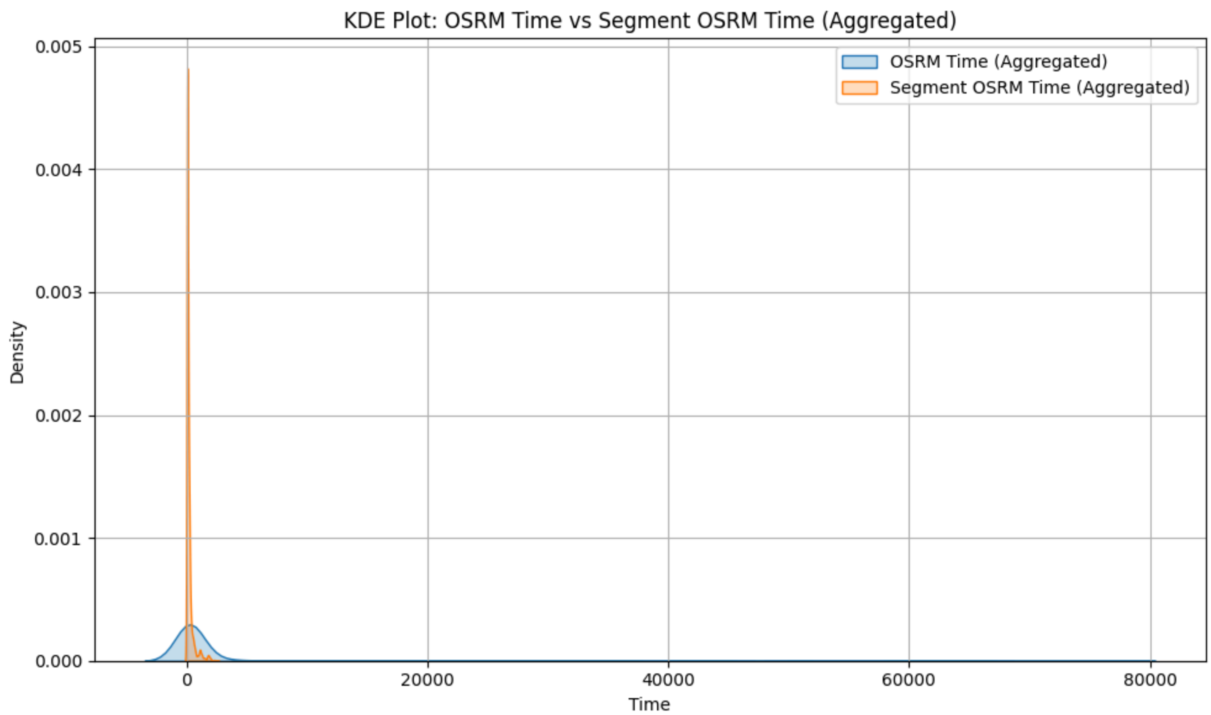

```
T-statistic: 28.9530, P-value: 0.000000
Reject the null hypothesis. There is a significant difference in aggregated
OSRM distance and aggregated segment OSRM distance.
```

Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value

In [157…

```
# Step 2: Aggregate by trip_uuid
agg_df = df.groupby('trip_uuid')[['osrm_time', 'segment_osrm_time']].sum()

# Step 3: Ensure numeric types and drop NaN/infinite values
agg_df['osrm_time'] = pd.to_numeric(agg_df['osrm_time'], errors='coerce')
agg_df['segment_osrm_time'] = pd.to_numeric(agg_df['segment_osrm_time'], err
agg_df_clean = agg_df[np.isfinite(agg_df['osrm_time']) & np.isfinite(agg_df[

# Step 4: KDE Plot
plt.figure(figsize=(10, 6))
sns.kdeplot(data=agg_df_clean, x='osrm_time', label='OSRM Time (Aggregated)'
sns.kdeplot(data=agg_df_clean, x='segment_osrm_time', label='Segment OSRM Ti
plt.title('KDE Plot: OSRM Time vs Segment OSRM Time (Aggregated)')
plt.xlabel('Time')
plt.ylabel('Density')
plt.legend()
```

```
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 5: Hypothesis Test
t_stat, p_val = ttest_ind(agg_df_clean['osrm_time'], agg_df_clean['segment_c
print(f"T-statistic: {t_stat:.4f}, P-value: {p_val:.6f}")
if p_val < 0.05:
    print("Reject the null hypothesis. There is a significant difference in
else:
    print("Fail to reject the null hypothesis. There is no significant diffe
```



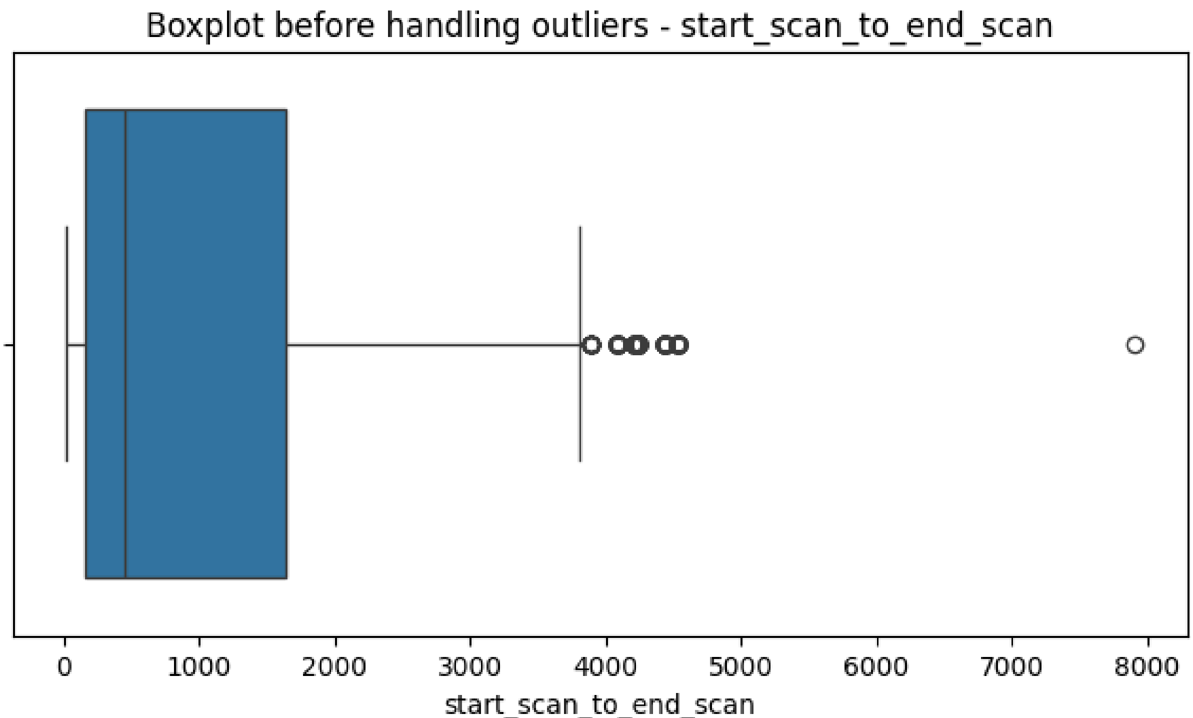

```
T-statistic: 29.1974, P-value: 0.000000
Reject the null hypothesis. There is a significant difference in aggregated
OSRM time and aggregated segment OSRM time.
```

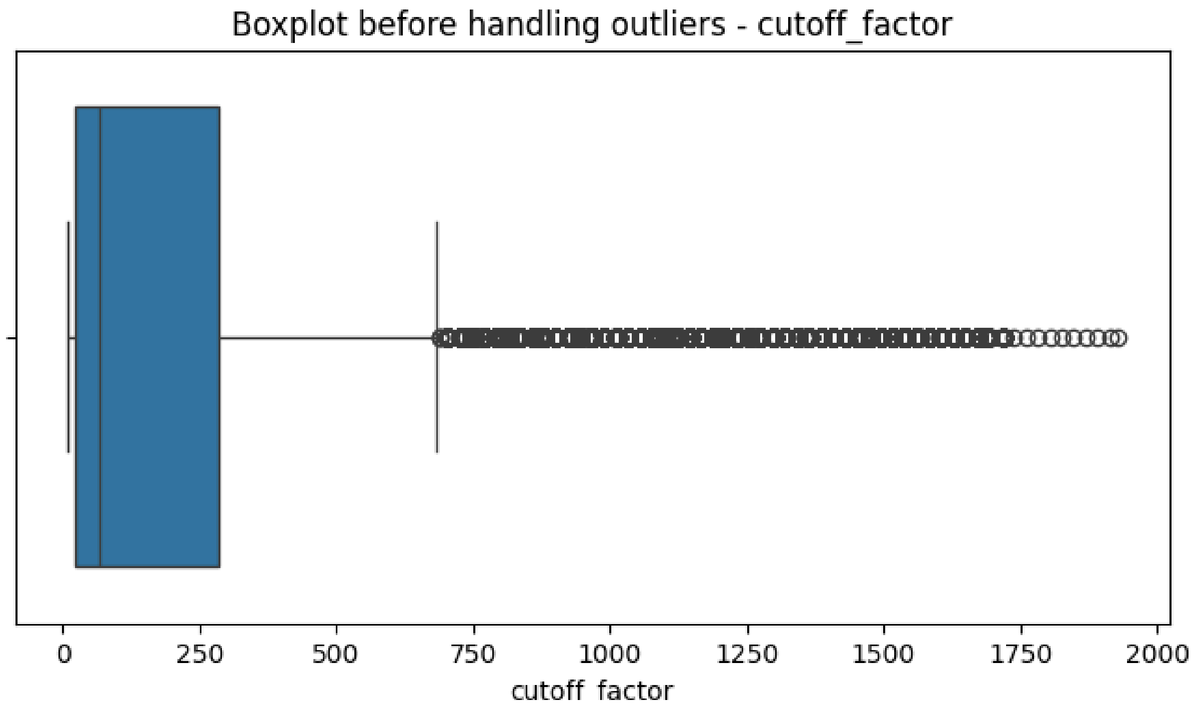
Outliers handle using the IQR method.

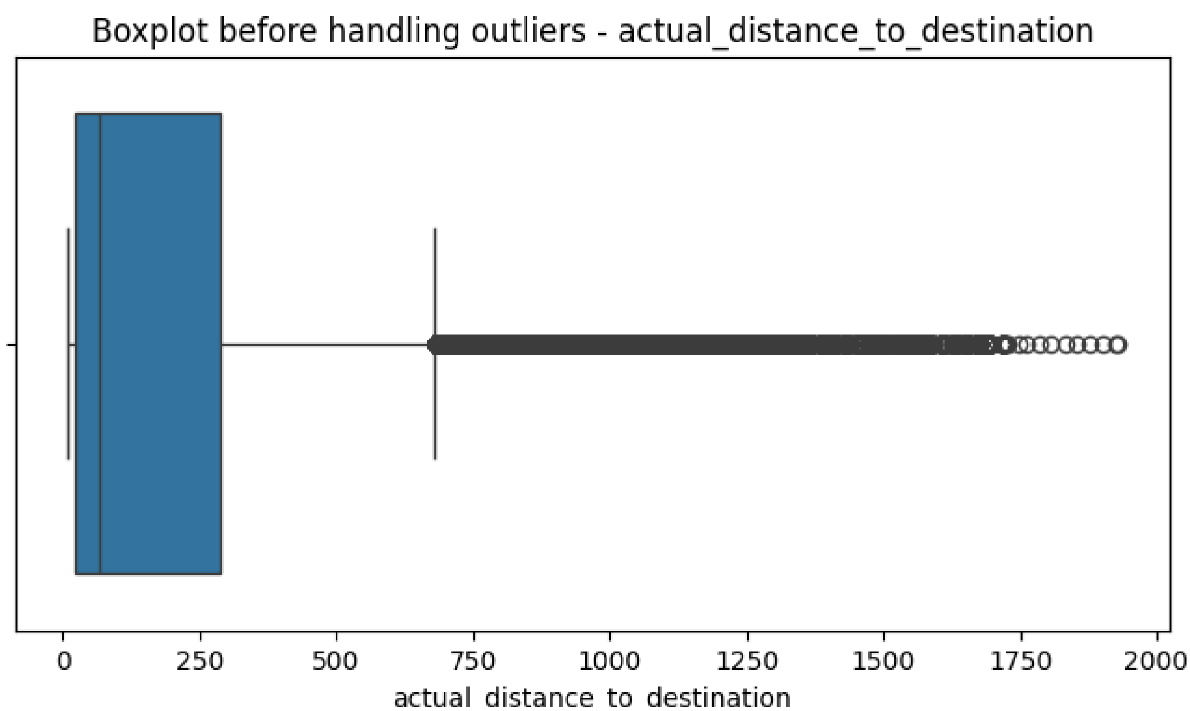
In [158…

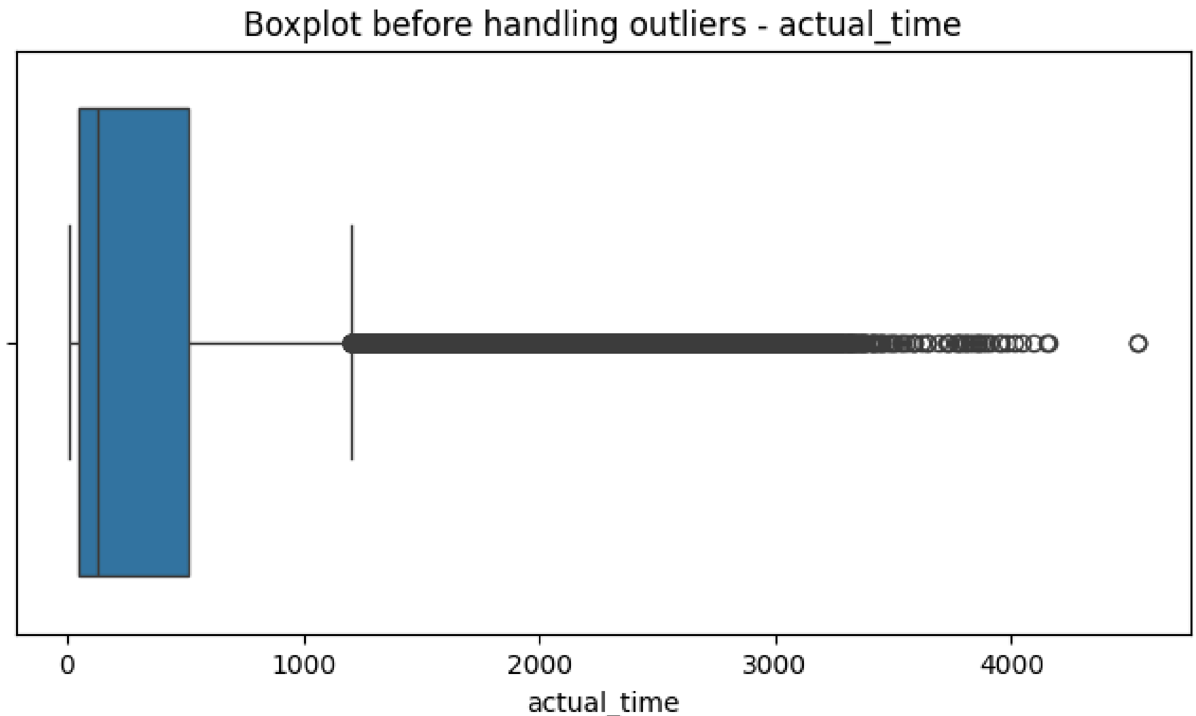
```
# Step 1: Identify numerical columns
numerical_cols = df.select_dtypes(include=['number']).columns

# Step 2: Outlier Detection + Handling via IQR
def handle_outliers_iqr(data, cols):
    cleaned_df = data.copy()
    for col in cols:
        Q1 = cleaned_df[col].quantile(0.25)
        Q3 = cleaned_df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Option 1: Remove outliers
        cleaned_df = cleaned_df[(cleaned_df[col] >= lower_bound) & (cleaned_
```

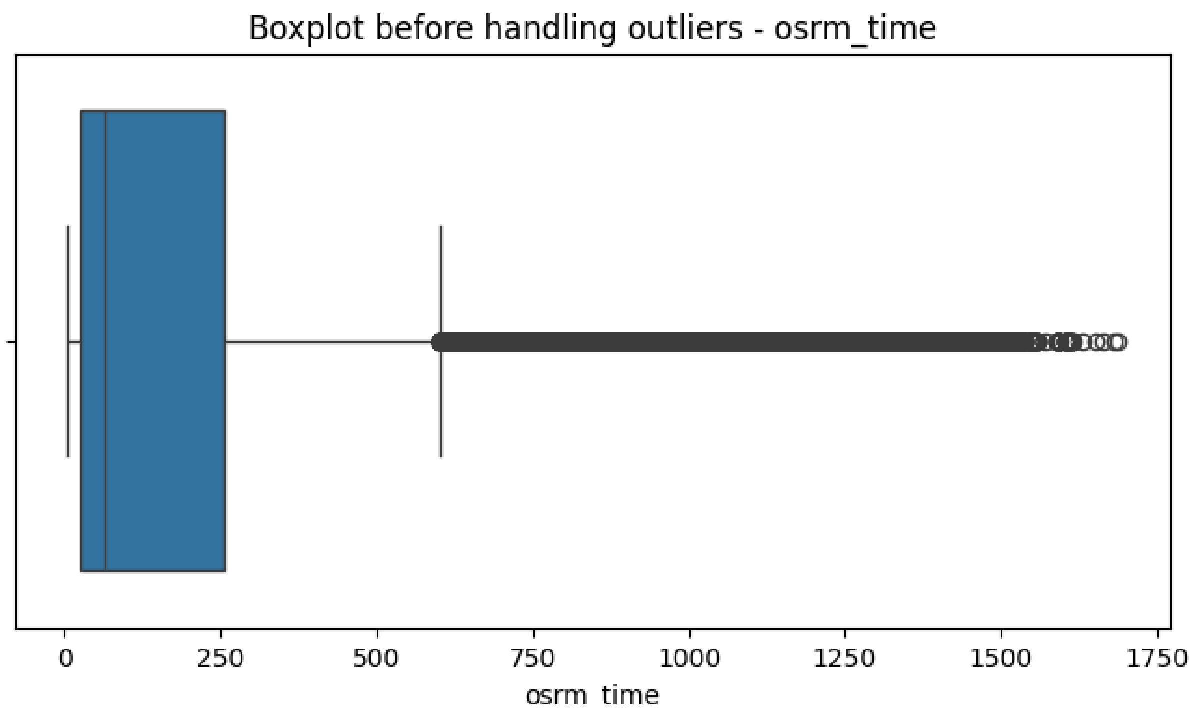
```
    return cleaned_df

# Visualize before handling
for col in numerical_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot before handling outliers - {col}')
    plt.show()

# Handle outliers
df_cleaned = handle_outliers_iqr(df, numerical_cols)

# Visualize after handling
for col in numerical_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df_cleaned[col])
    plt.title(f'Boxplot after handling outliers - {col}')
    plt.show()
```

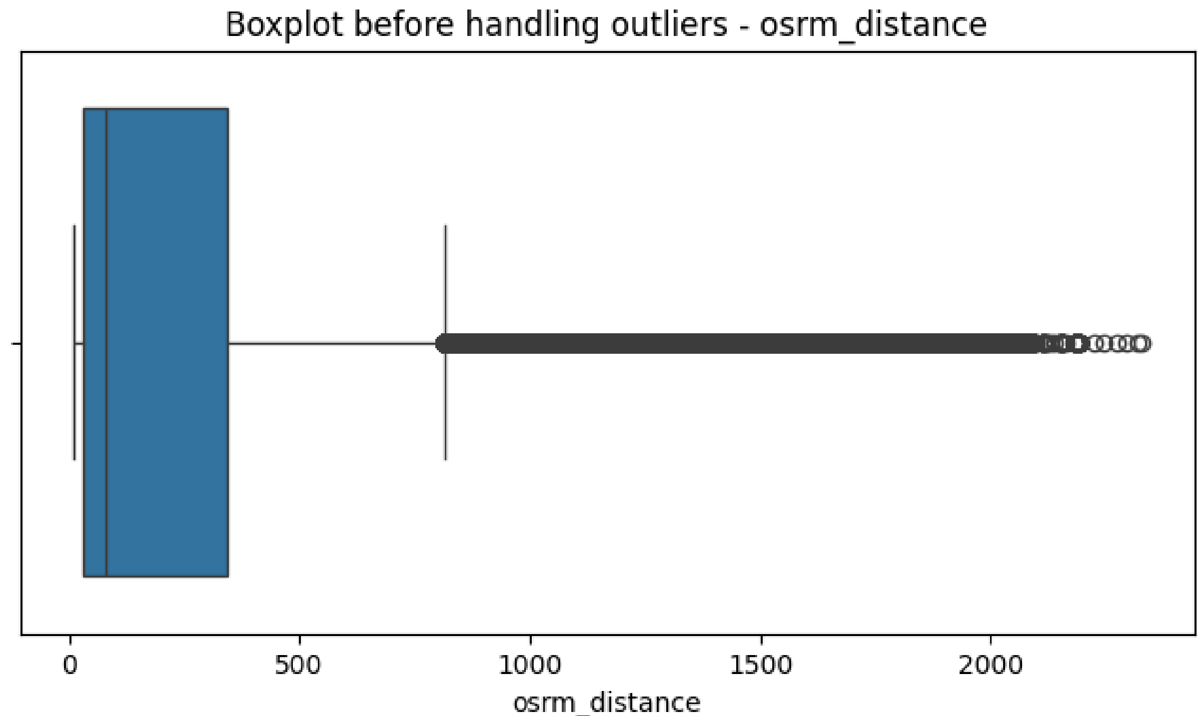



Boxplot before handling outliers - cutoff_factor
0
250
500
750
1000
1250
1500
1750
2000
cutoff_factor

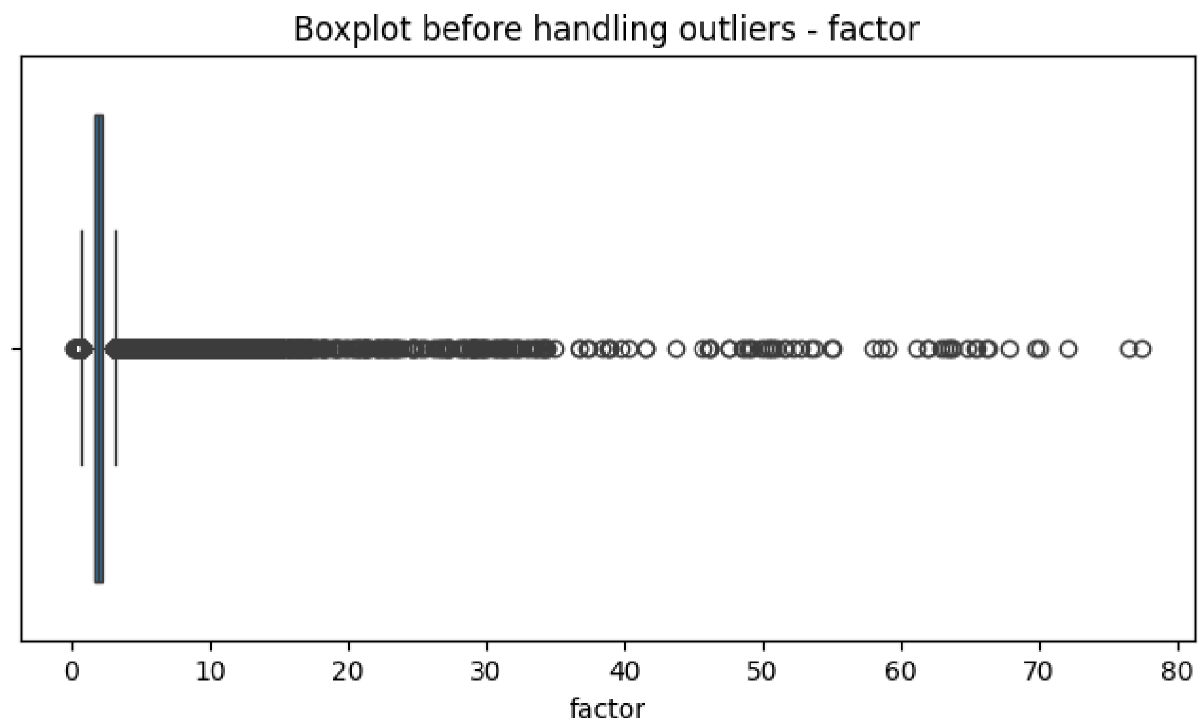

Boxplot before handling outliers - actual_distance_to_destination
0
250
500
750
1000
1250
1500
1750
2000
actual_distance_to_destination


Boxplot before handling outliers - actual_time
0
1000
2000
3000
4000
actual_time

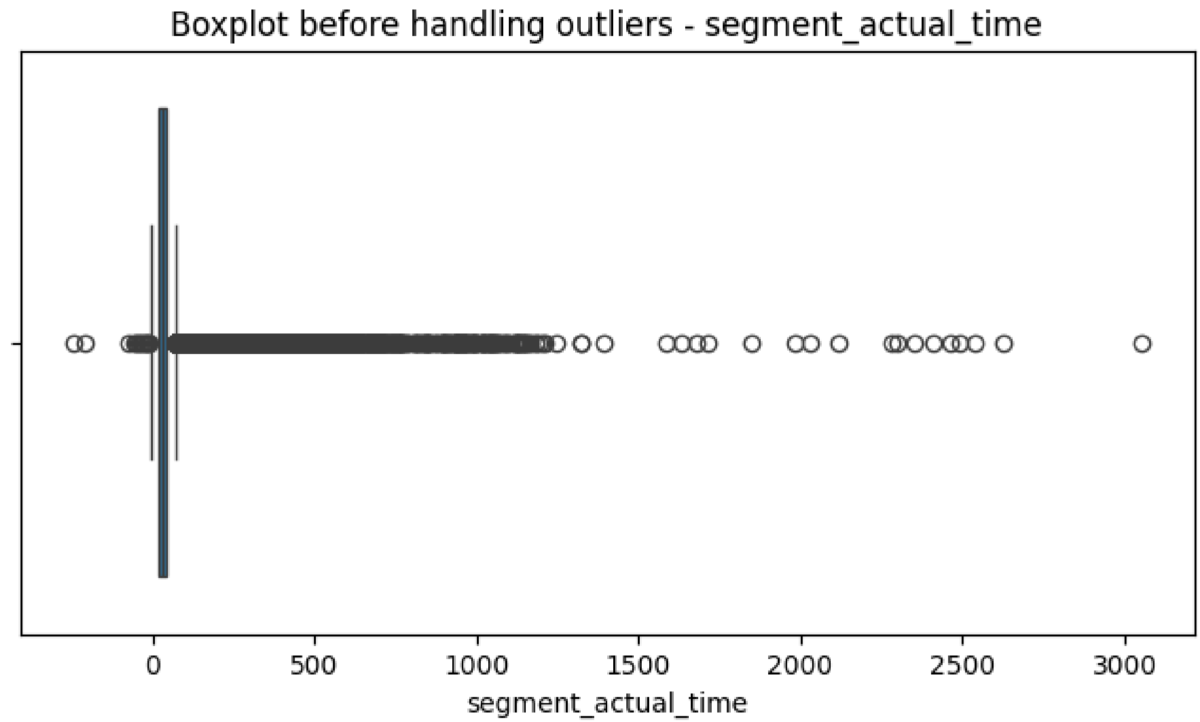

Boxplot before handling outliers - osrm_time
0
250
500
750
1000
1250
1500
1750
osrm_time



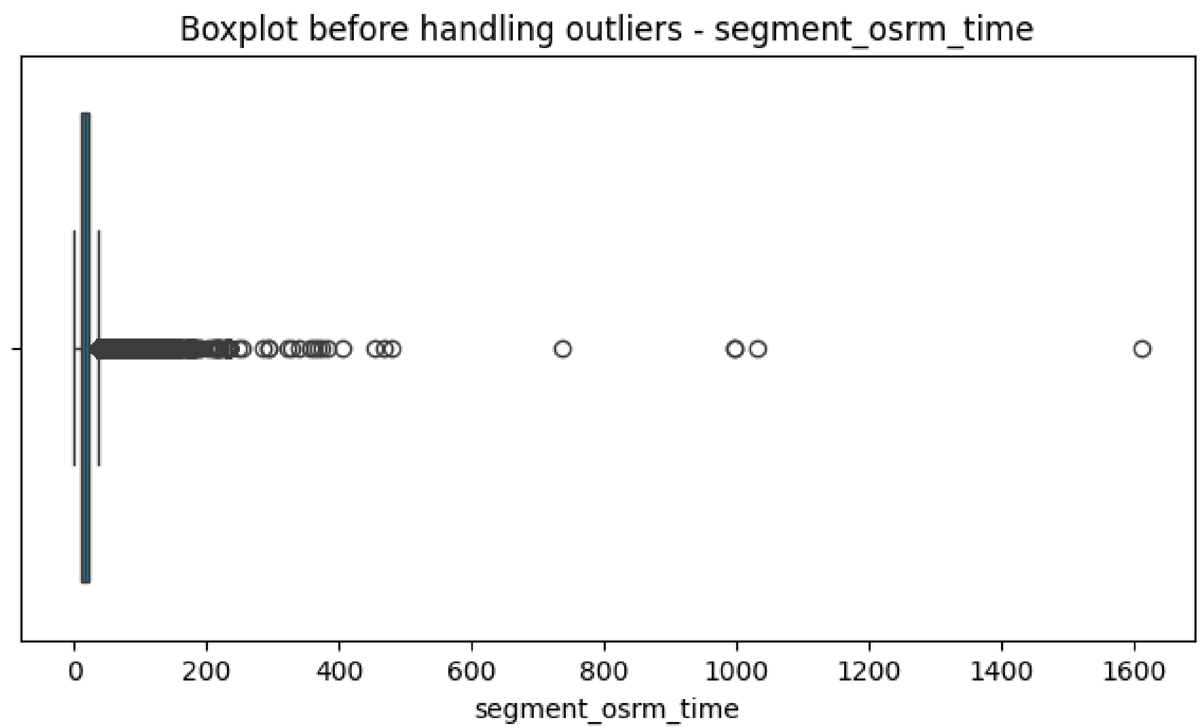
Boxplot before handling outliers - osrm_distance
0
500
1000
1500
2000
osrm_distance



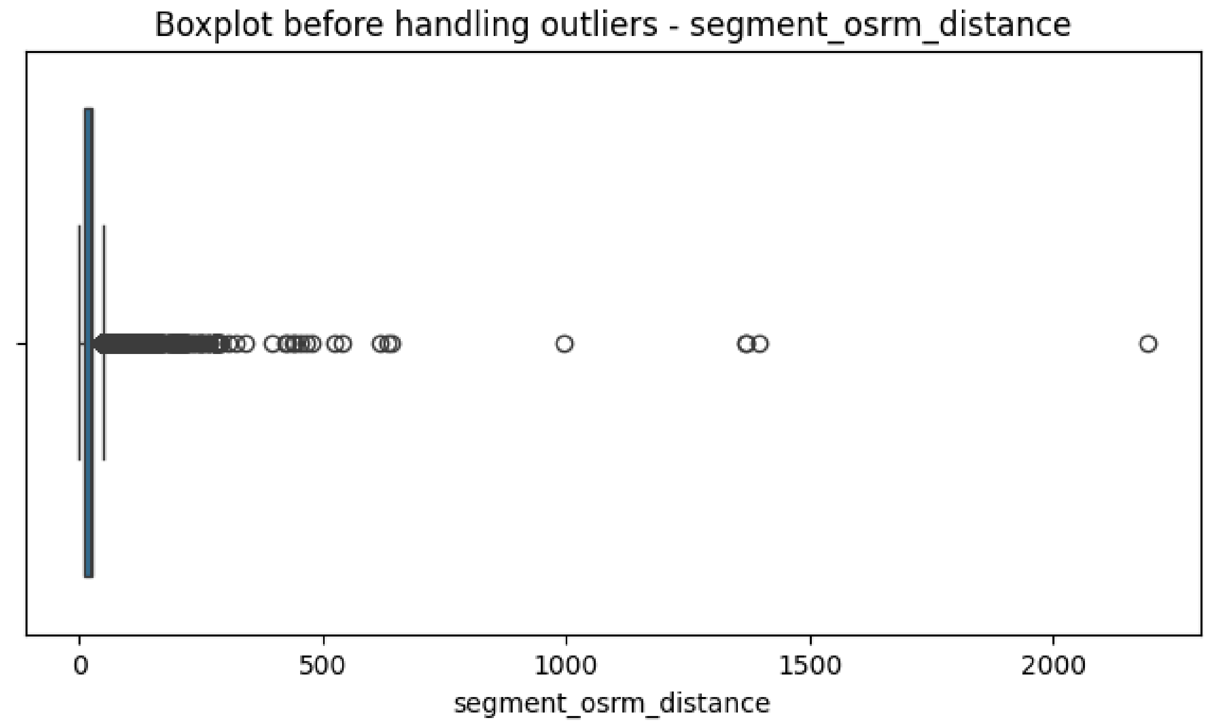
Boxplot before handling outliers - factor
0
10
20
30
40
50
60
70
80
factor

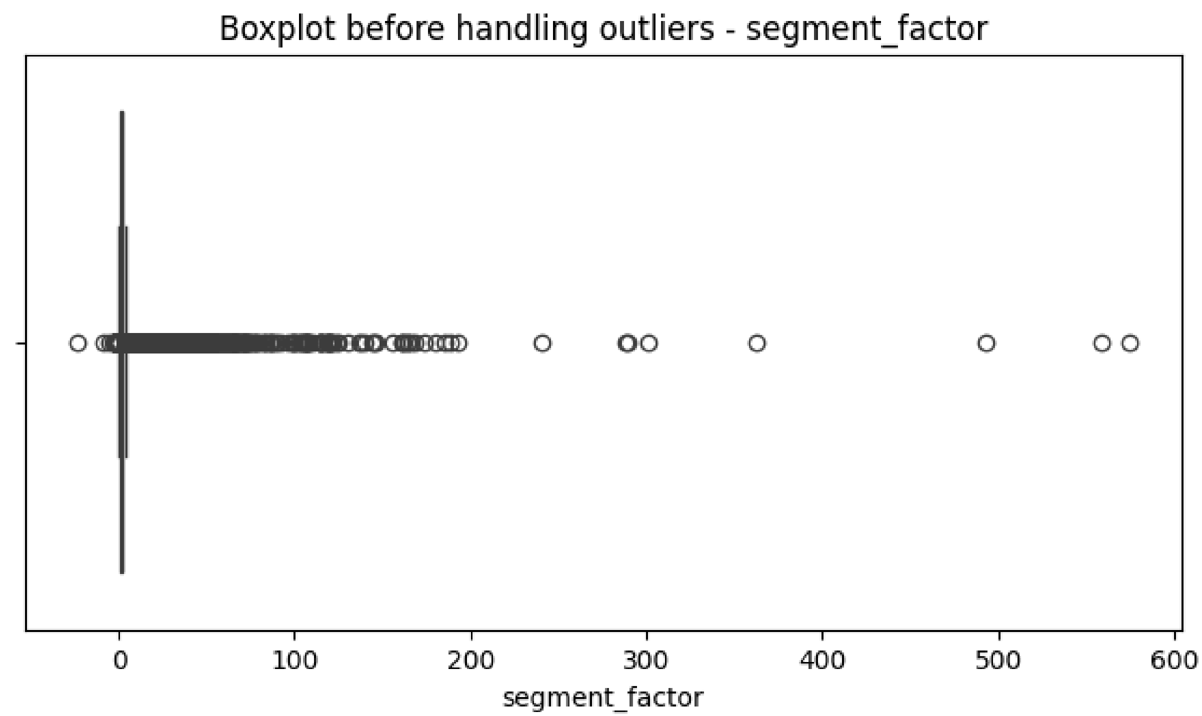


Boxplot before handling outliers - segment_actual_time
0
500
1000
1500
2000
2500
3000
segment_actual_time

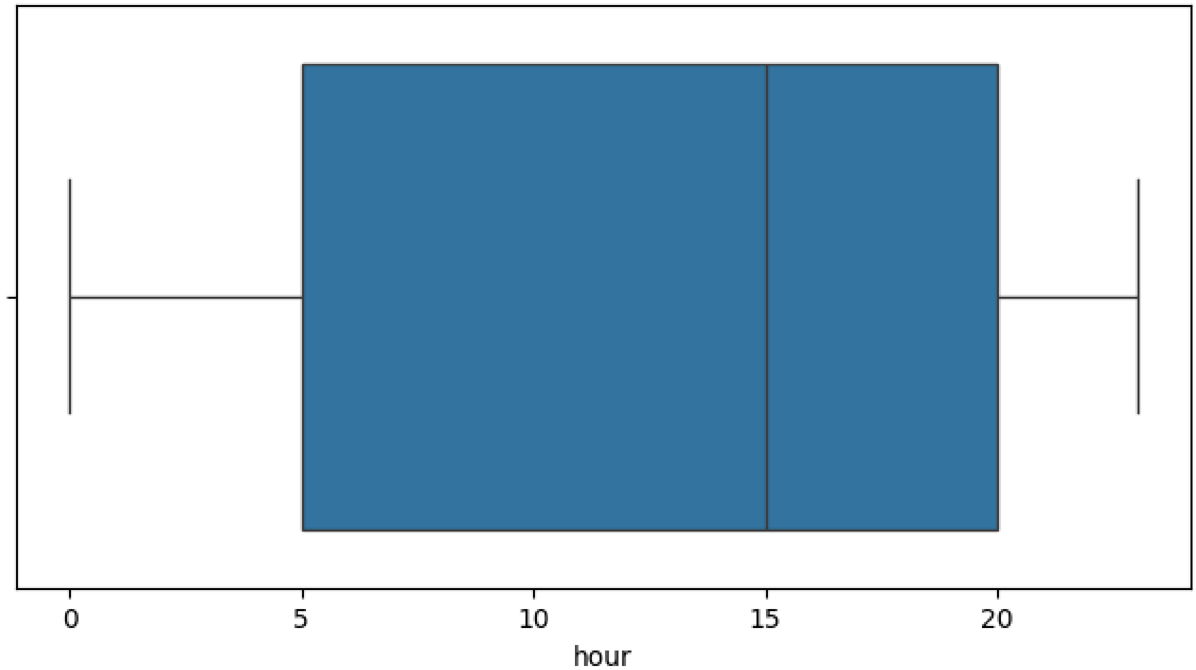


Boxplot before handling outliers - segment_osrm_time
0
200
400
600
800
1000
1200
1400
1600
segment_osrm_time

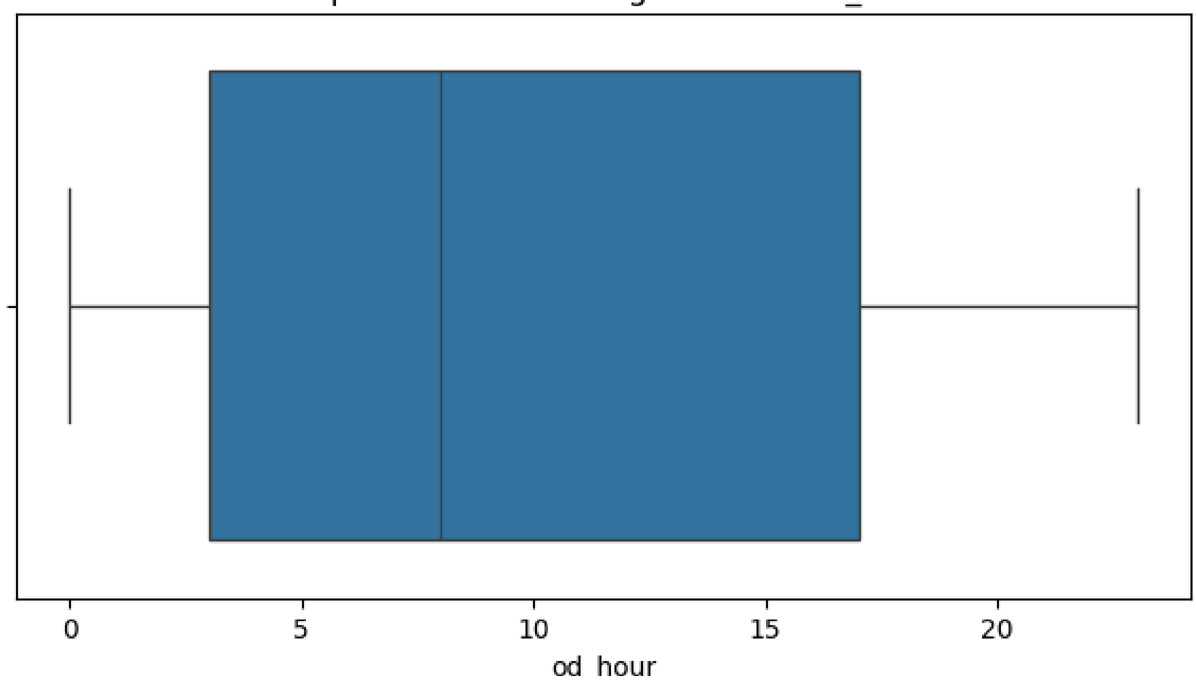

Boxplot before handling outliers - segment_osrm_distance
0
500
1000
1500
2000
segment_osrm_distance

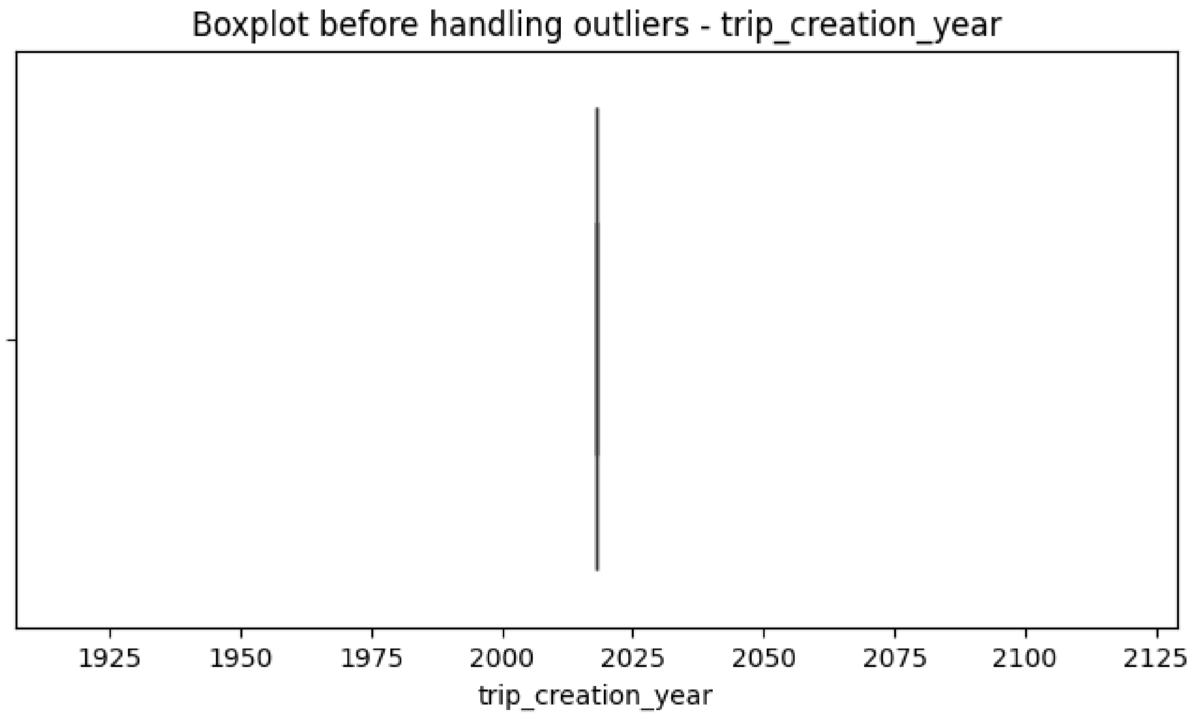

Boxplot before handling outliers - segment_factor
0
100
200
300
400
500
600
segment_factor

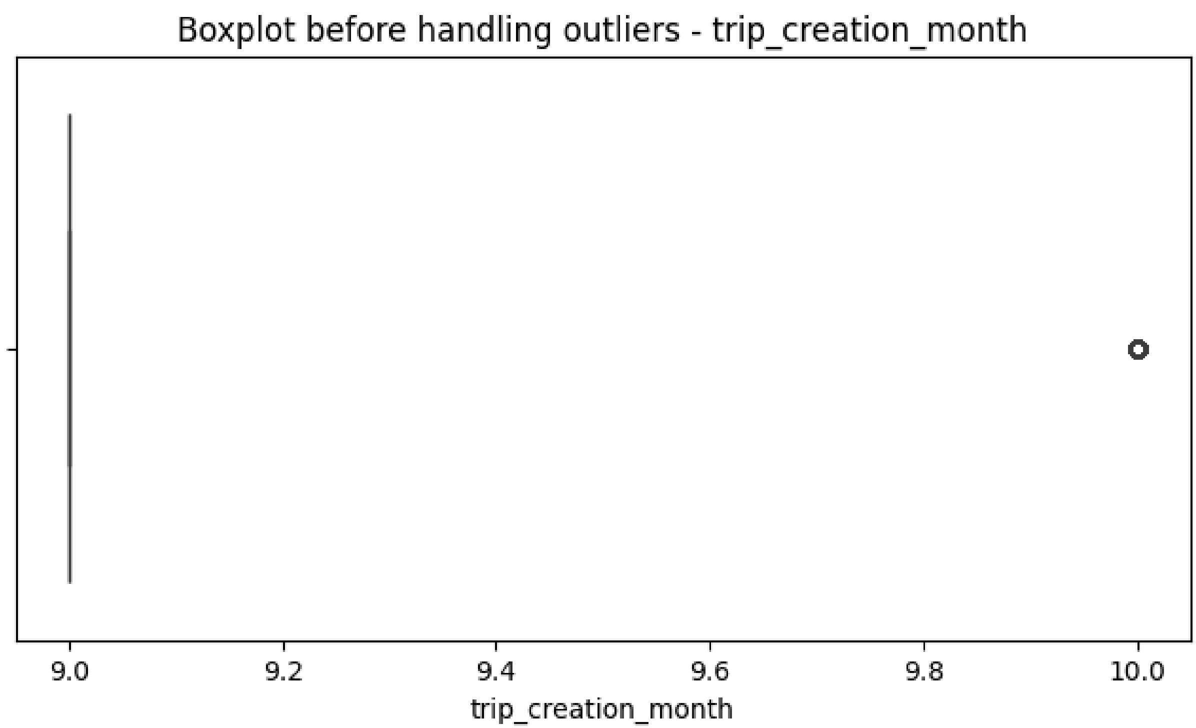
Boxplot before handling outliers - hour



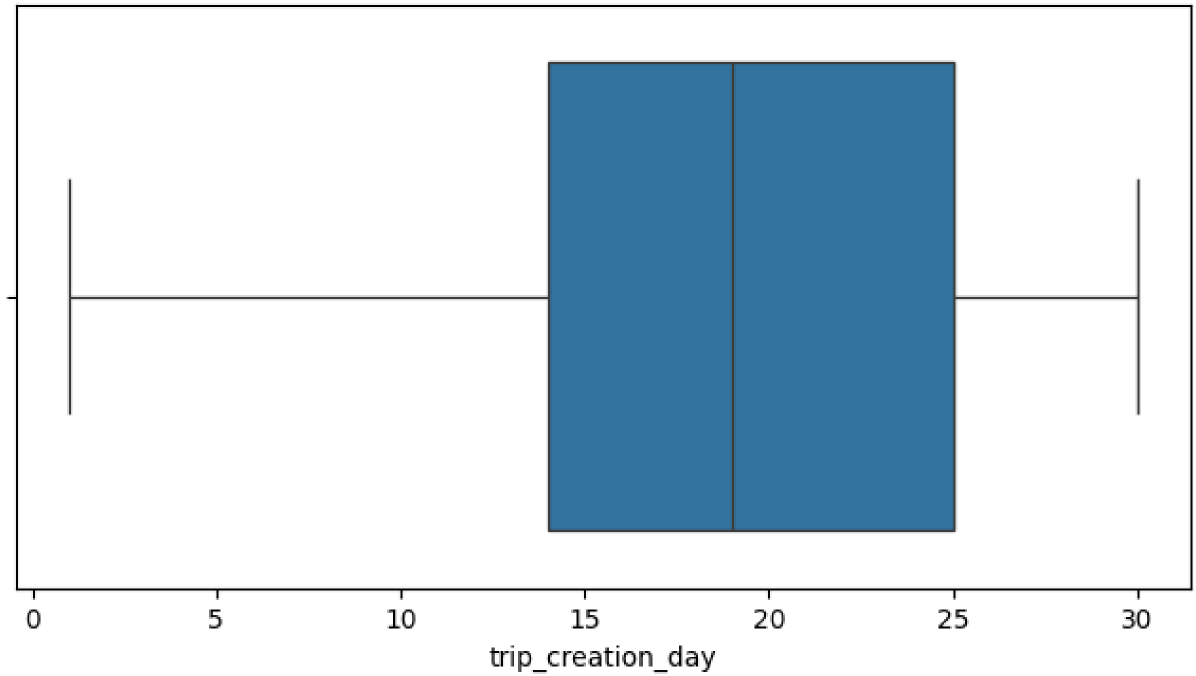
Boxplot before handling outliers - od_hour



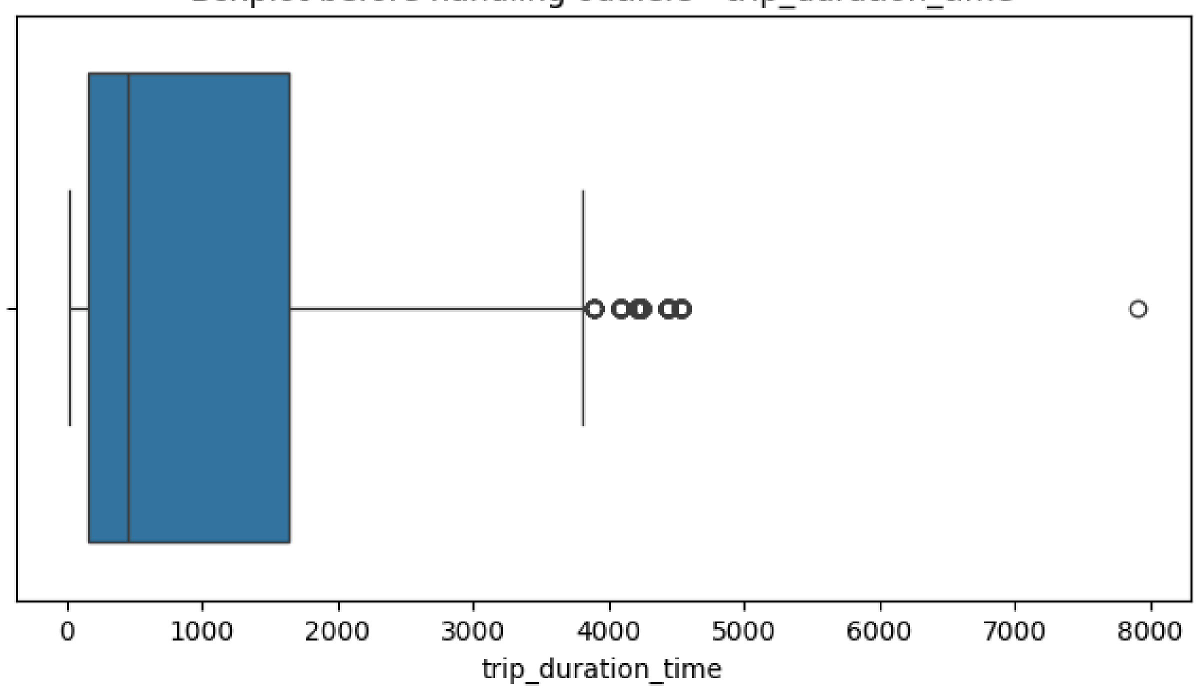

Boxplot before handling outliers - trip_creation_year
1925
1950
1975
2000
2025
2050
2075
2100
2125
trip_creation_year

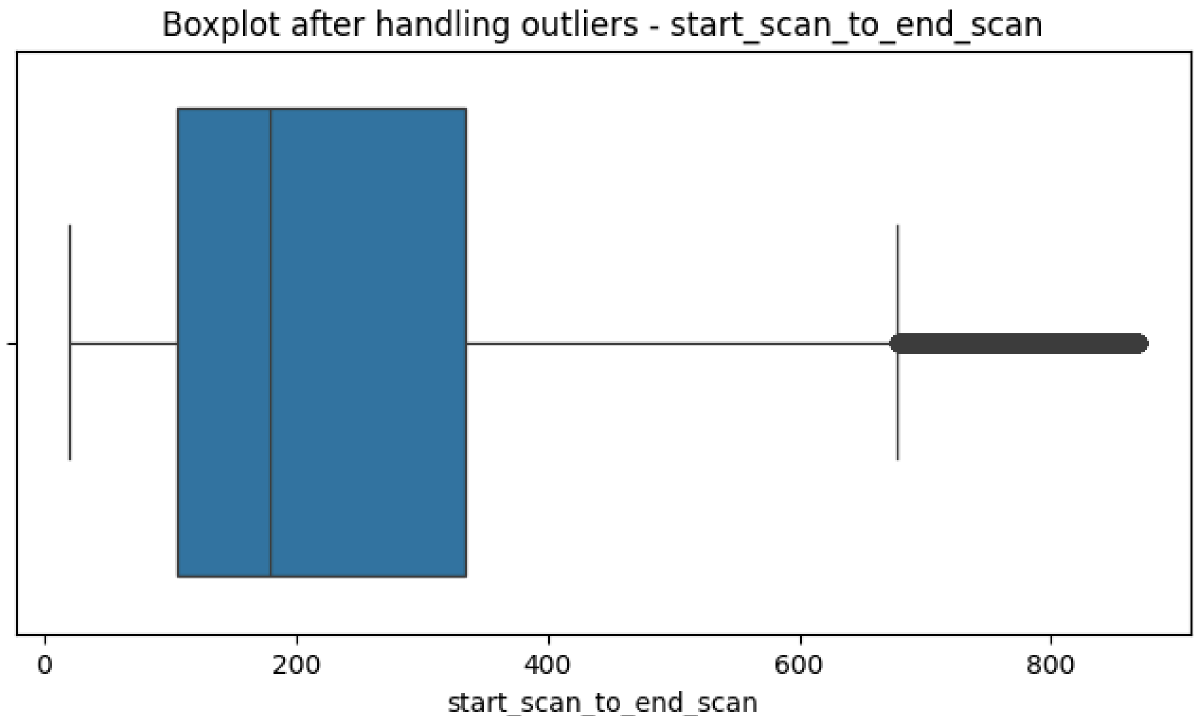

Boxplot before handling outliers - trip_creation_month
9.0
9.2
9.4
9.6
9.8
10.0
trip_creation_month

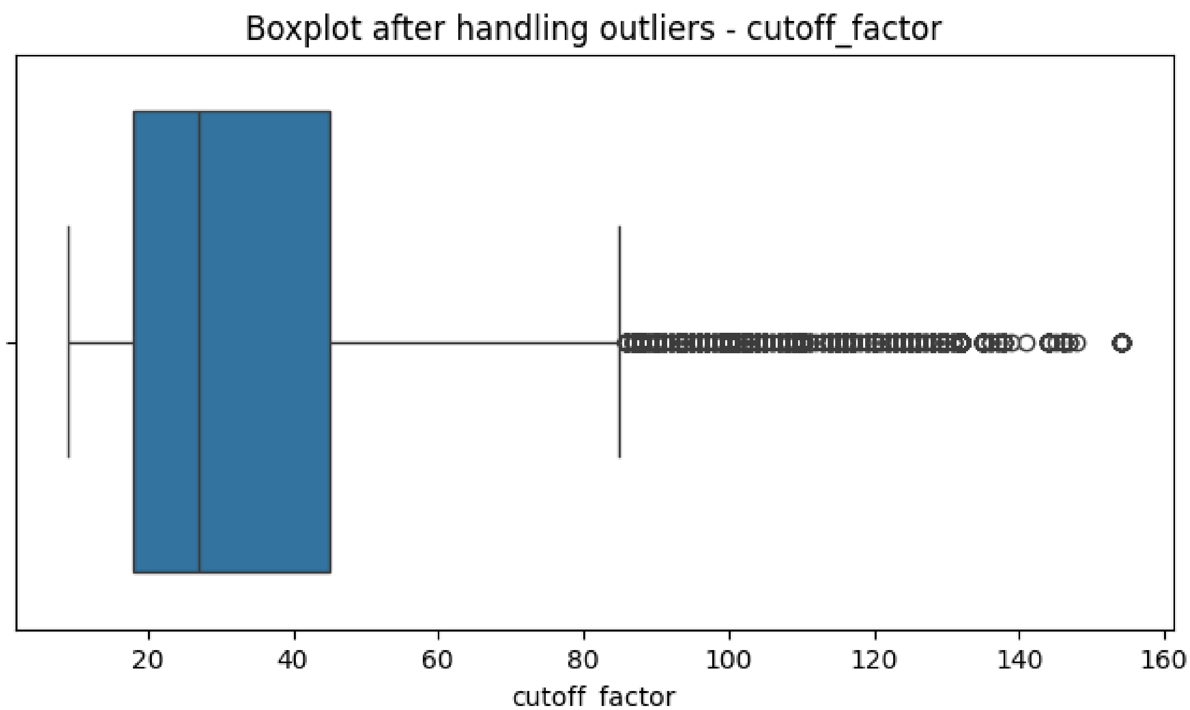
Boxplot before handling outliers - trip_creation_day

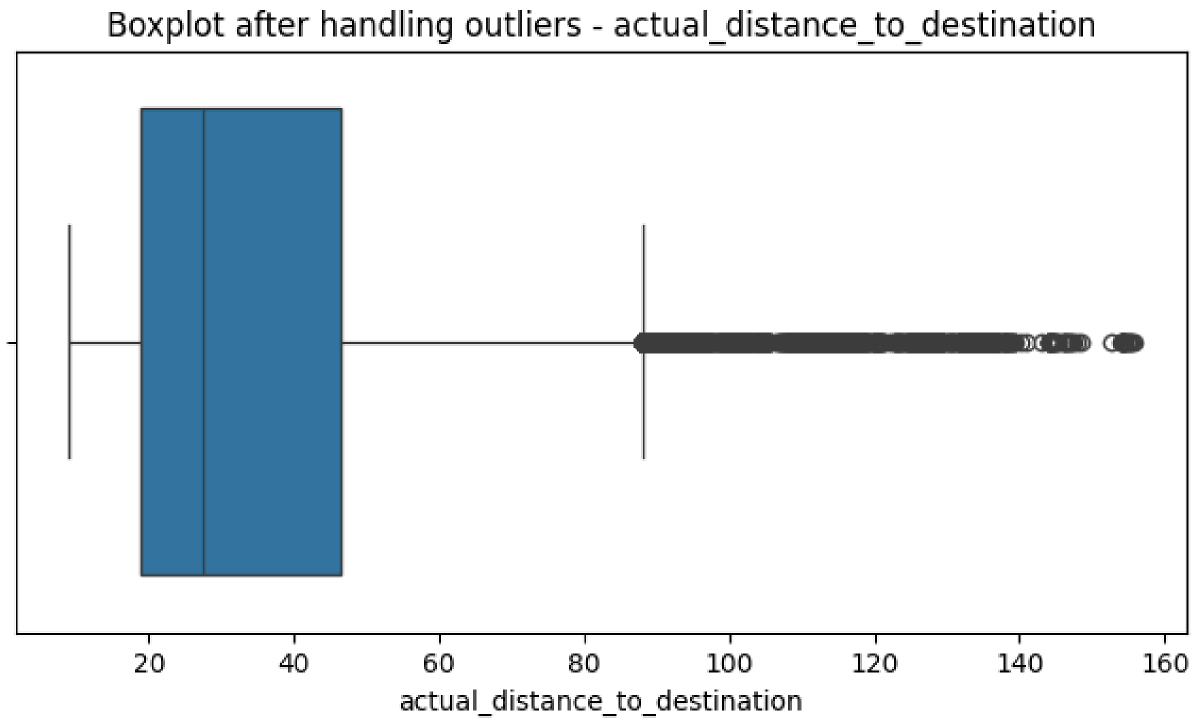


Boxplot before handling outliers - trip_duration_time

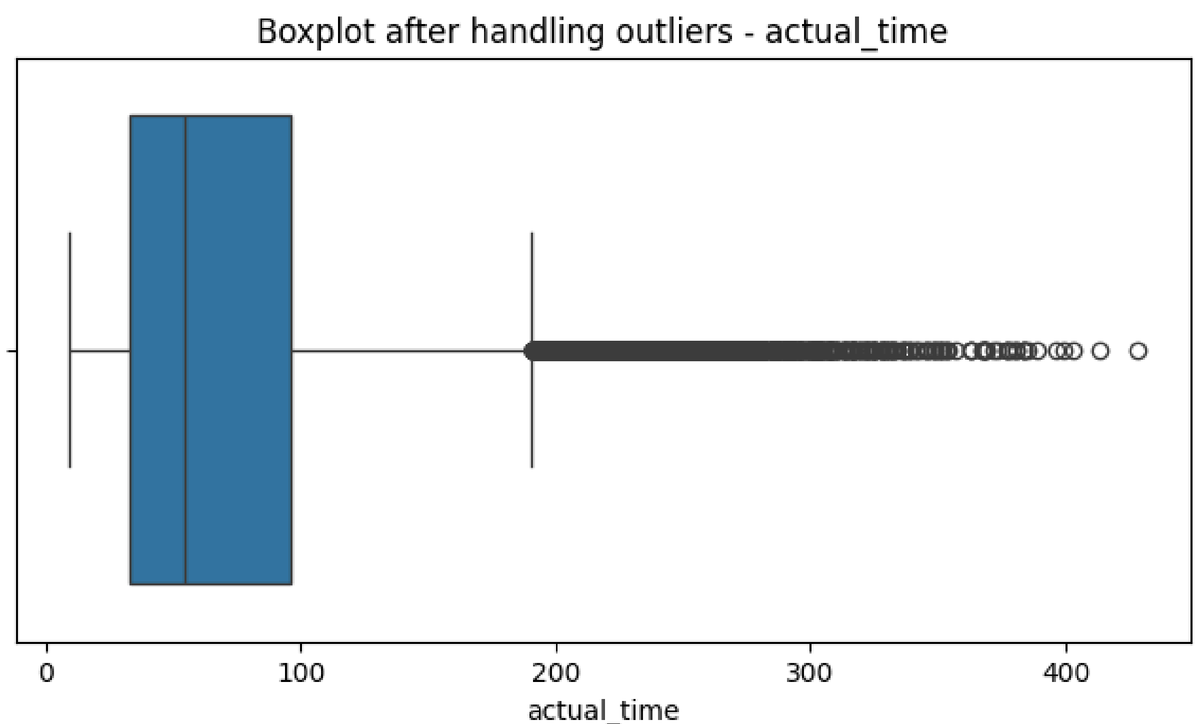



Boxplot after handling outliers - start_scan_to_end_scan
0
200
400
600
800
start_scan_to_end_scan

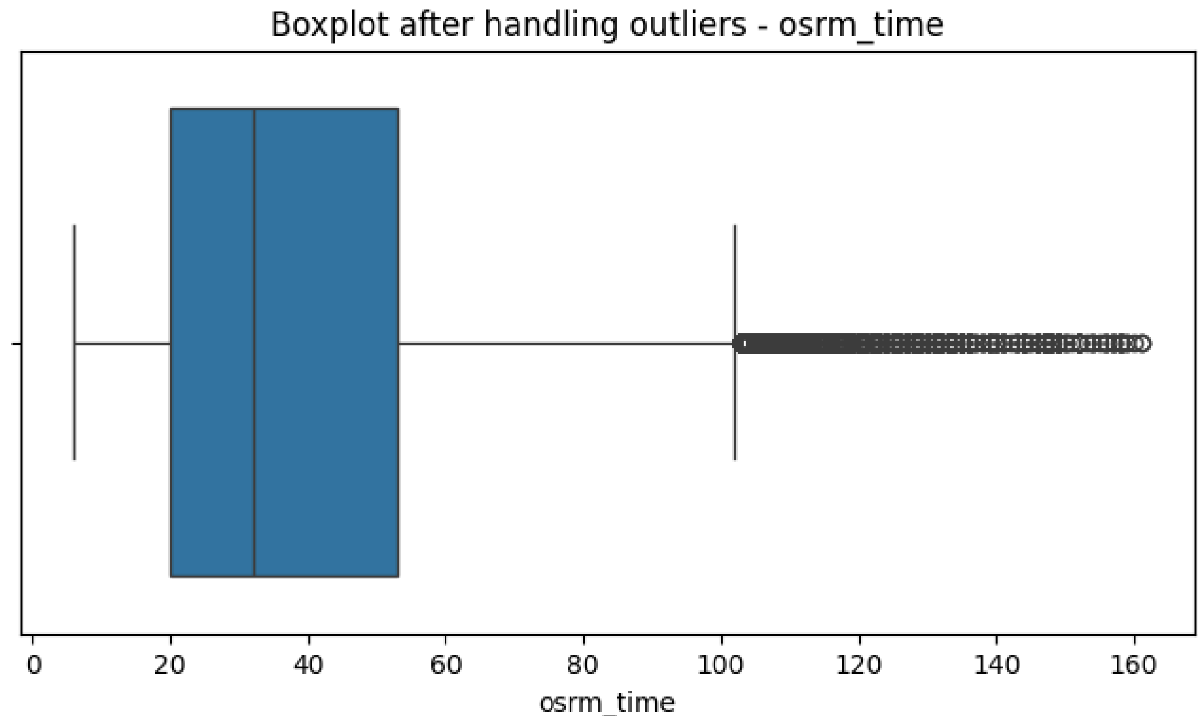

Boxplot after handling outliers - cutoff_factor
20
40
60
80
100
120
140
160
cutoff_factor



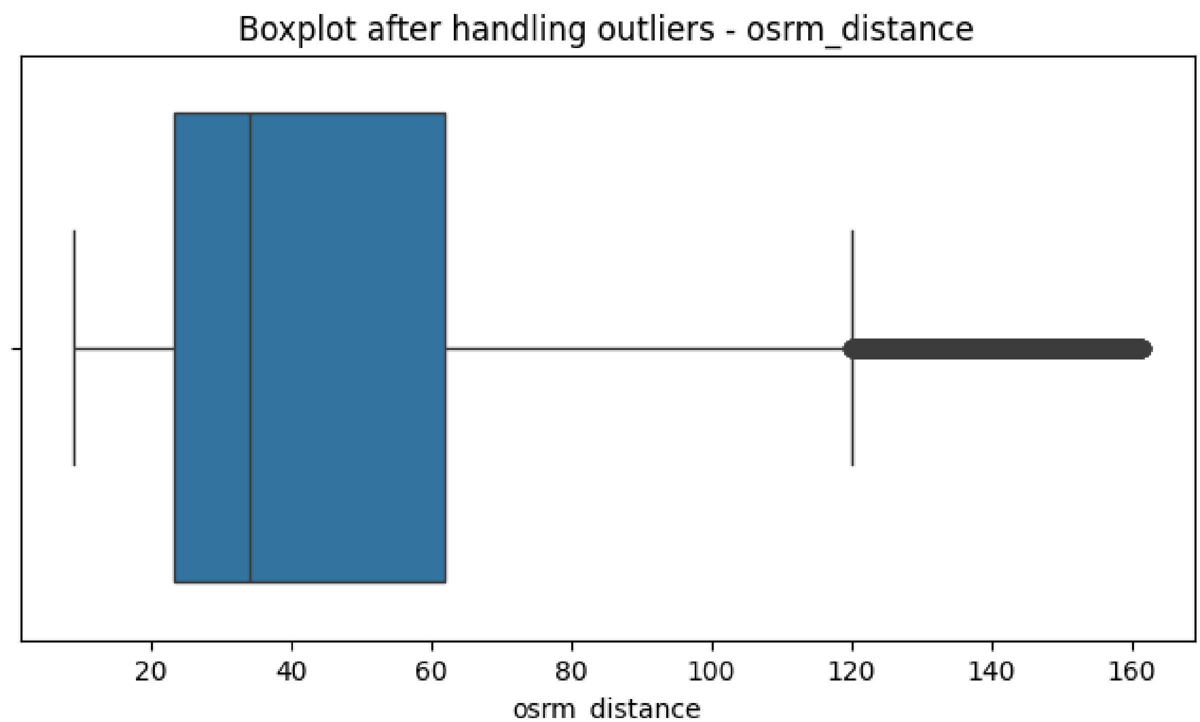
Boxplot after handling outliers - actual_distance_to_destination
20
40
60
80
100
120
140
160
actual_distance_to_destination



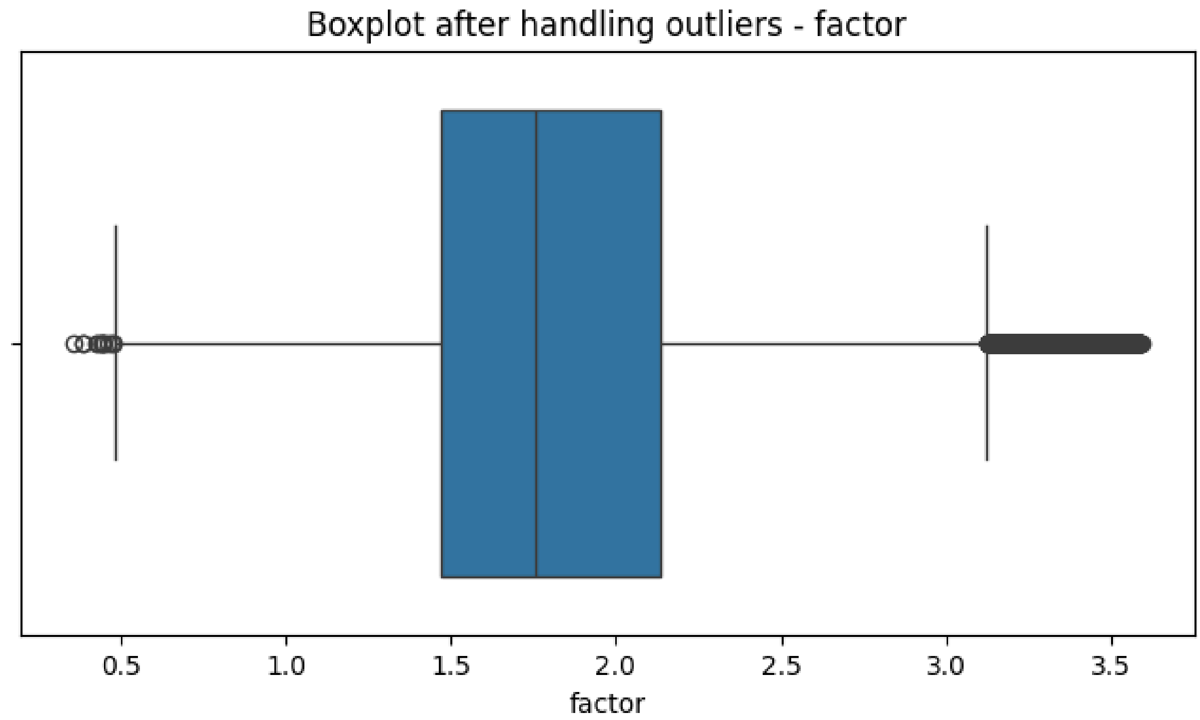
Boxplot after handling outliers - actual_time
0
100
200
300
400
actual_time



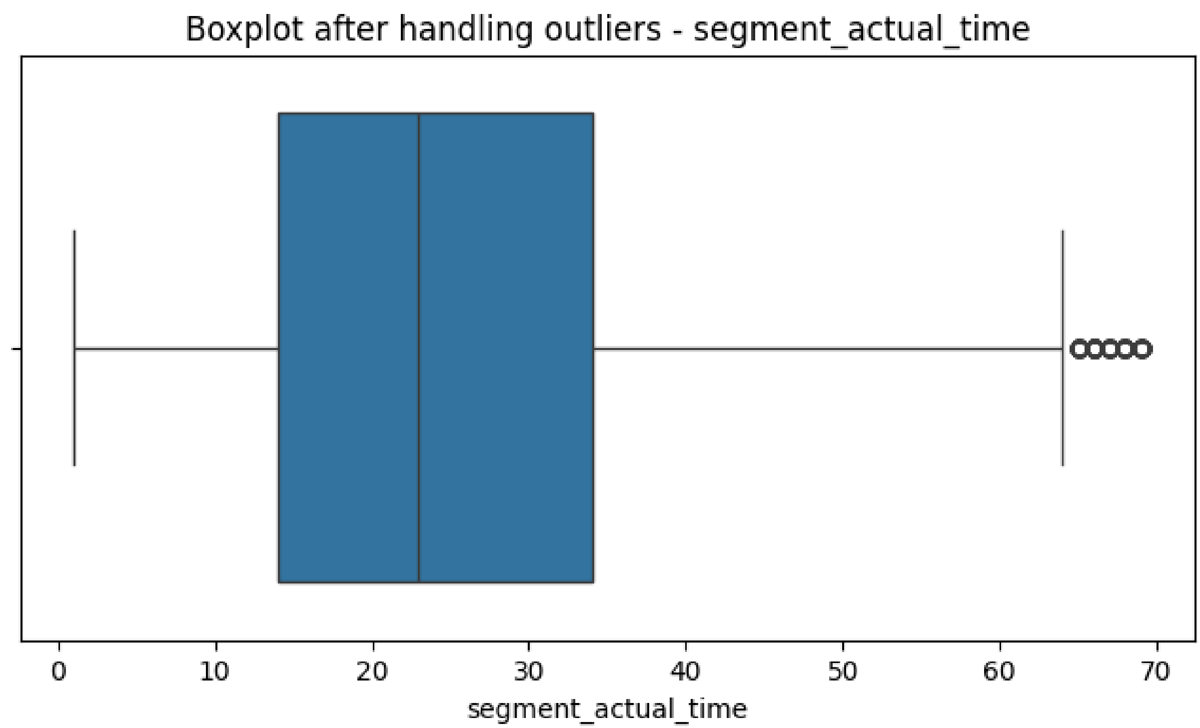
Boxplot after handling outliers - osrm_time
0
20
40
60
80
100
120
140
160
osrm_time



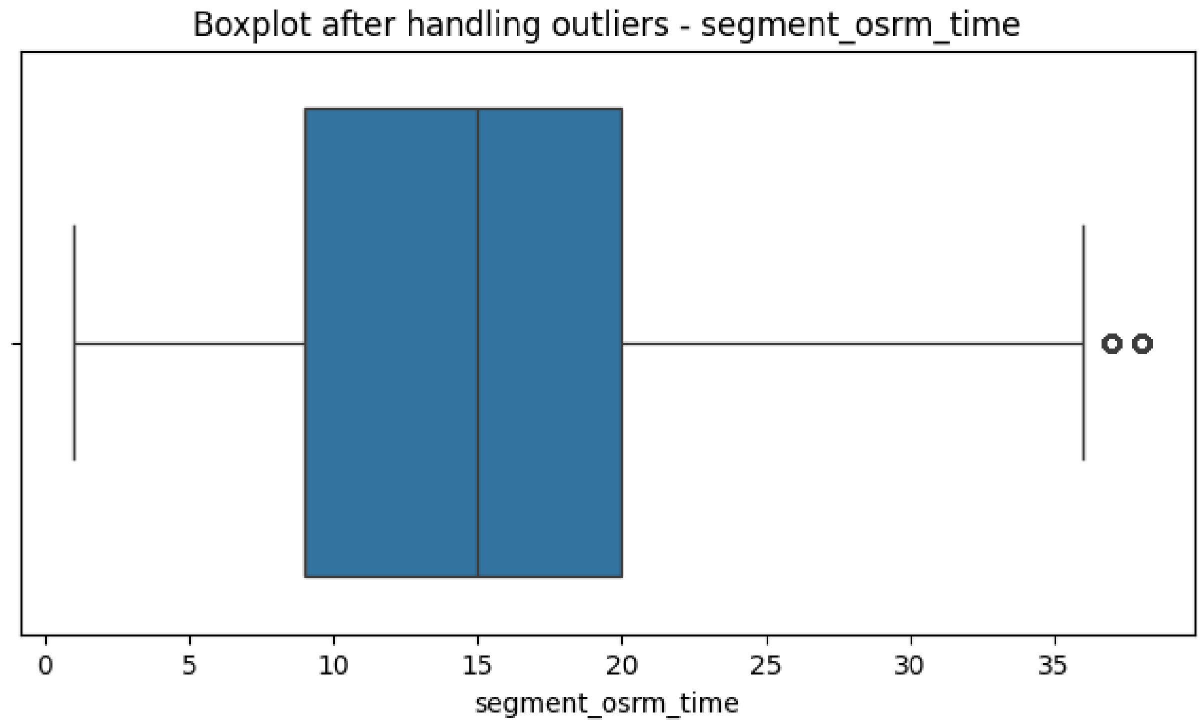
Boxplot after handling outliers - osrm_distance
20
40
60
80
100
120
140
160
osrm_distance

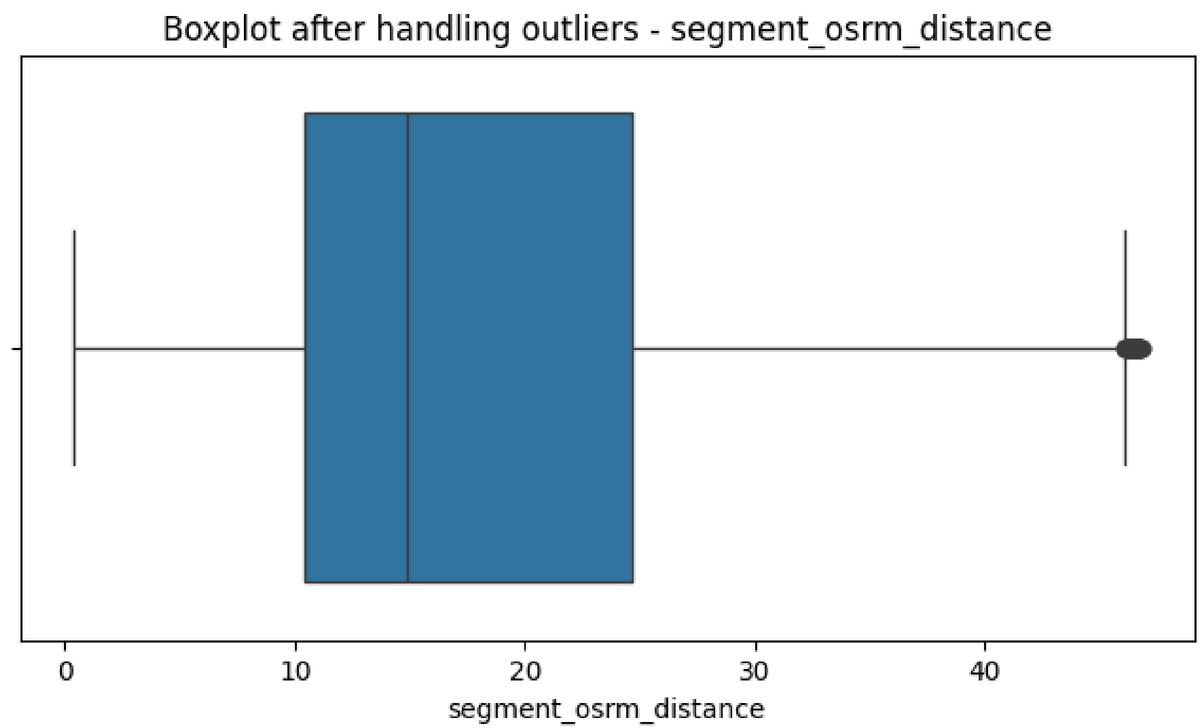


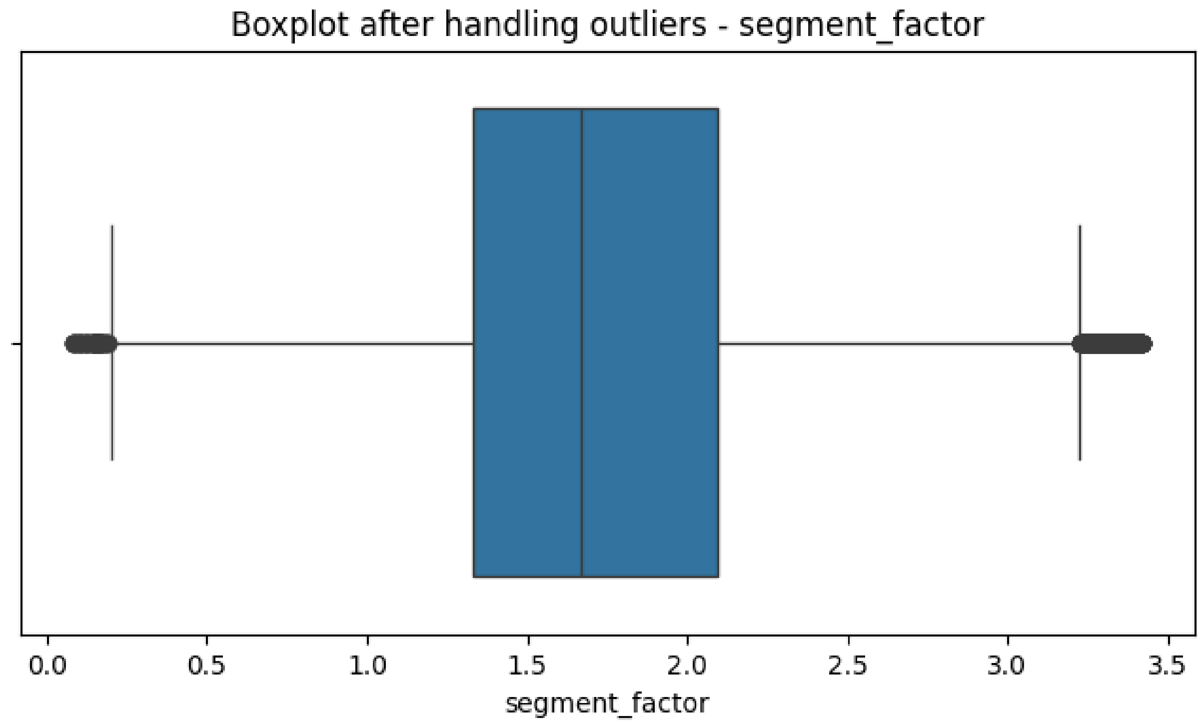
Boxplot after handling outliers - factor
0.5
1.0
1.5
2.0
2.5
3.0
3.5
factor



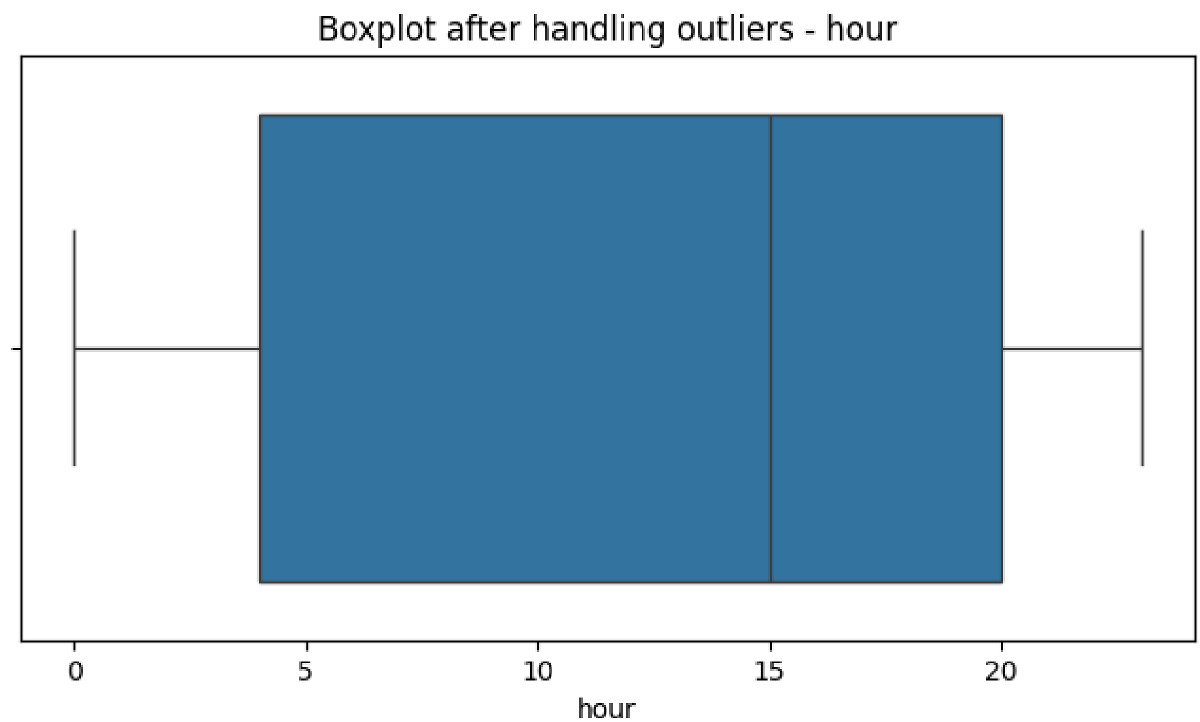
Boxplot after handling outliers - segment_actual_time
0
10
20
30
40
50
60
70
segment_actual_time


Boxplot after handling outliers - segment_osrm_time
0
5
10
15
20
25
30
35
segment_osrm_time

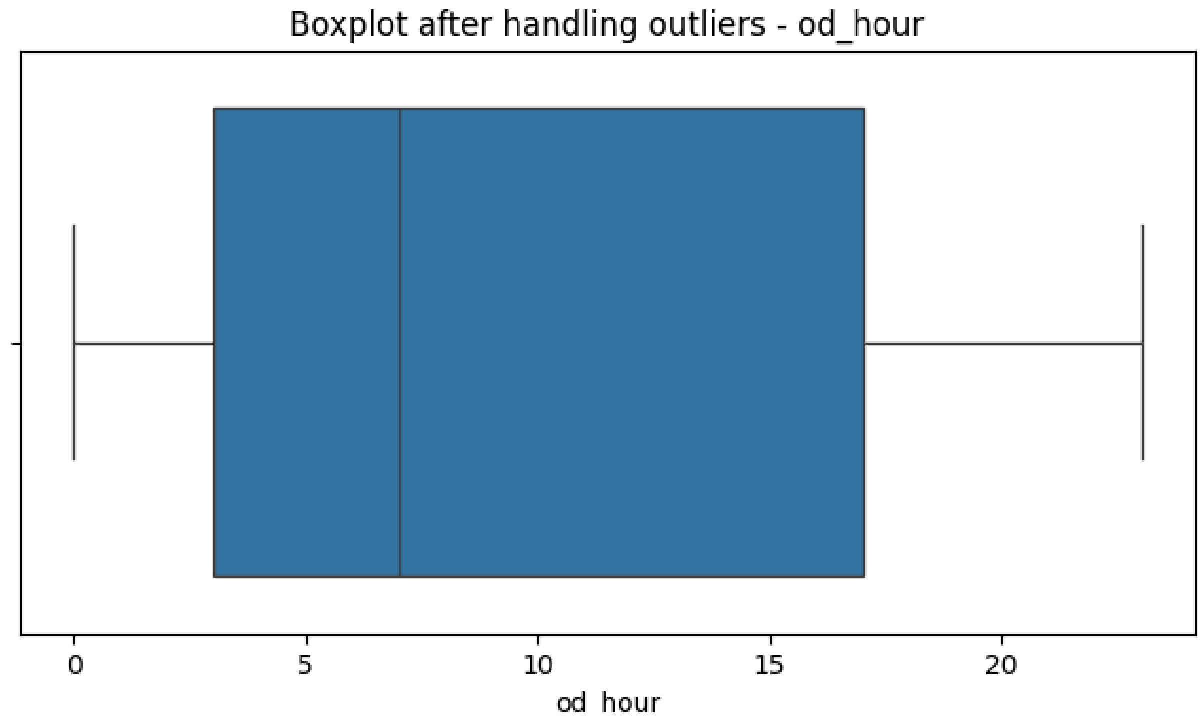

Boxplot after handling outliers - segment_osrm_distance
0
10
20
30
40
segment_osrm_distance

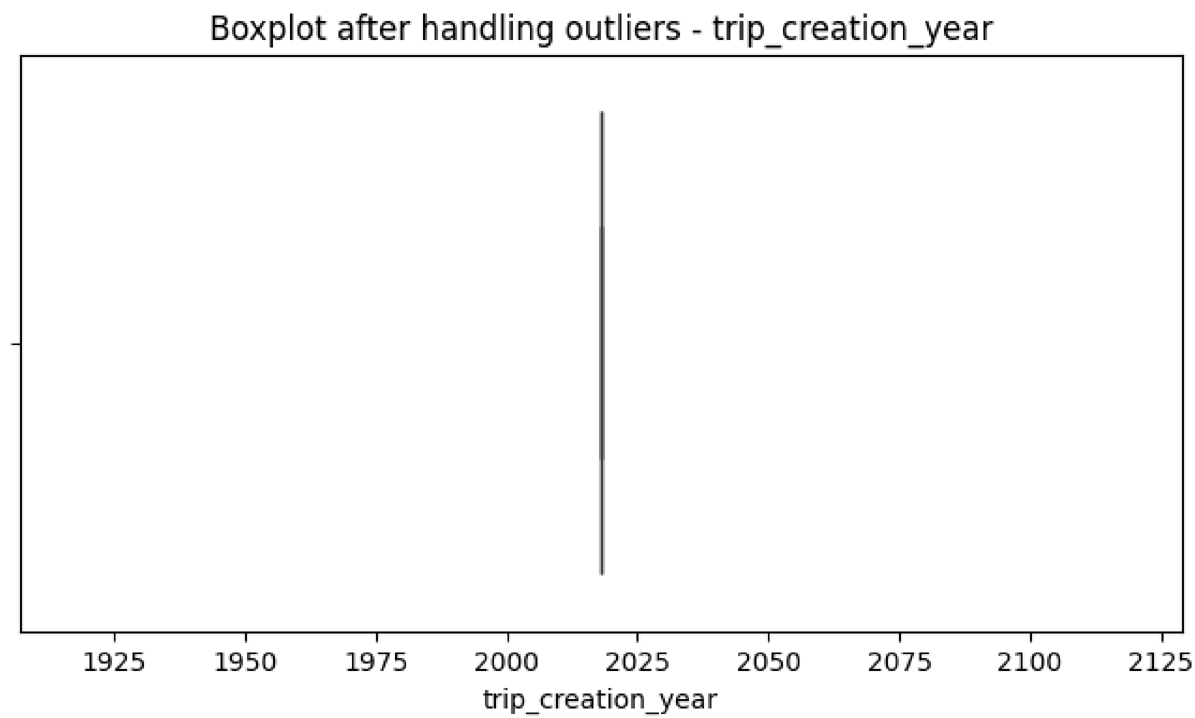


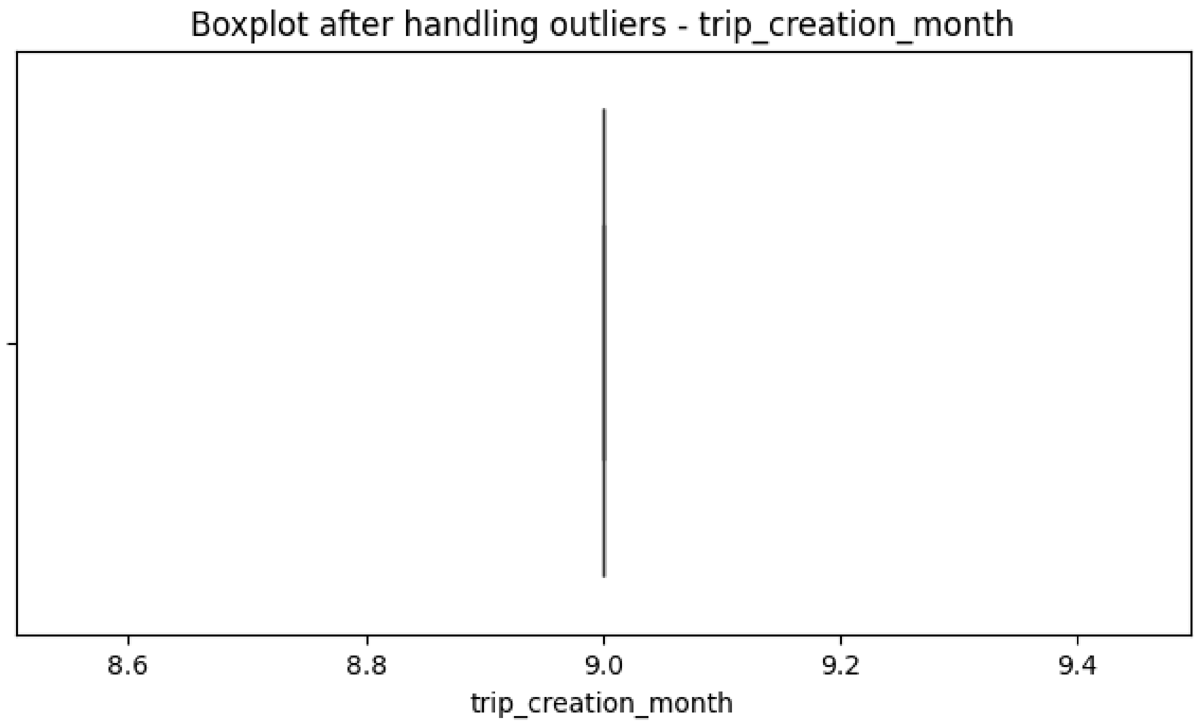
Boxplot after handling outliers - segment_factor
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
segment_factor



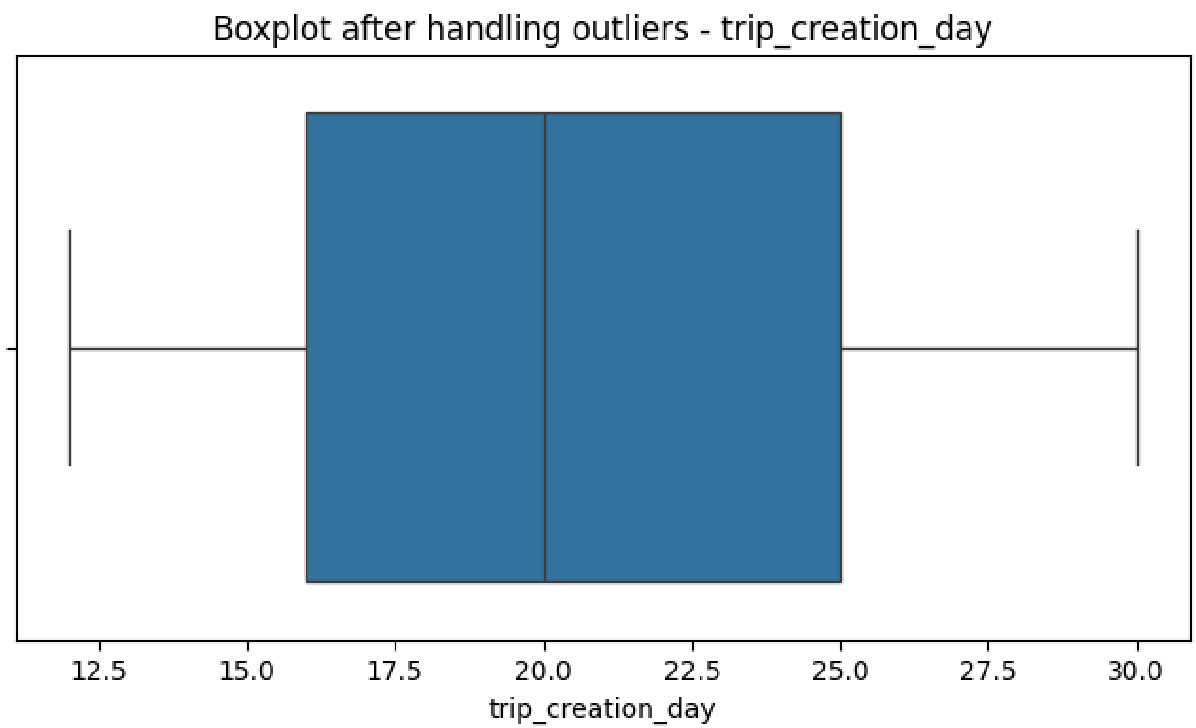
Boxplot after handling outliers - hour
0
5
10
15
20
hour


Boxplot after handling outliers - od_hour
0
5
10
15
20
od_hour

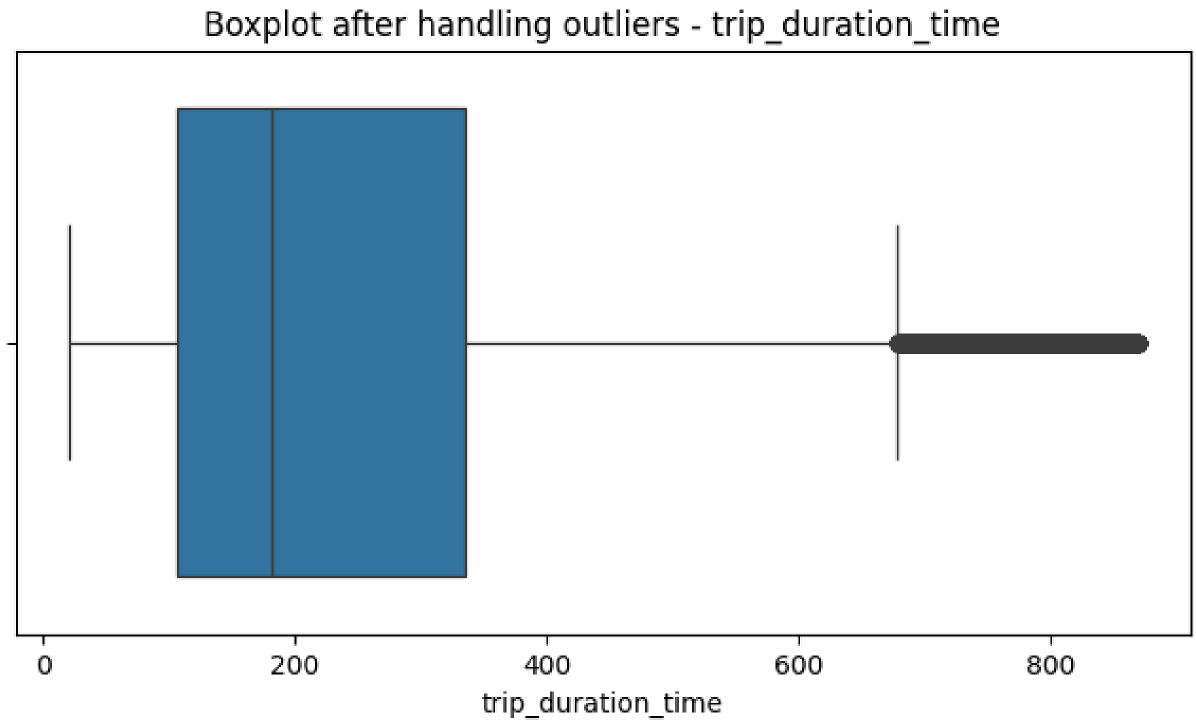

Boxplot after handling outliers - trip_creation_year
1925
1950
1975
2000
2025
2050
2075
2100
2125
trip_creation_year



Boxplot after handling outliers - trip_creation_month
8.6
8.8
9.0
9.2
9.4
trip_creation_month



Boxplot after handling outliers - trip_creation_day
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
trip_creation_day



one-hot encoding of categorical variables

In [159…

```
# 'route_type' is the categorical column
df_encoded = pd.get_dummies(df, columns=['route_type'], prefix='route', drop

# To check result
df_encoded[['route_Carting' , 'route_FTL']]
```

Out[159…

	route_Carting	route_FTL
0	True	False
1	True	False
2	True	False
3	True	False
4	True	False
...	...	...
144862	True	False
144863	True	False
144864	True	False
144865	True	False
144866	True	False

144867 rows × 2 columns

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [160…

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Step 1: Select numerical columns
numerical_cols = df.select_dtypes(include=['number']).columns

# Step 2: Choose scaler
scaler = MinMaxScaler()  # Or use StandardScaler()

# Step 3: Fit and transform
df_scaled = df.copy()
df_scaled[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Step 4: View result
df_scaled.head()
```

Out[160…

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

5 rows × 35 columns

Business Insights

Top Origin States by Order Volume

In [161…

```
df['origin_state'] = df['source_name'].str.extract(r'\((.*?)\)$')  # assumir
orders_by_state = df['origin_state'].value_counts()
print(orders_by_state.head())
```

```
origin_state
Haryana        27499
Maharashtra    21401
Karnataka      19578
Tamil Nadu      7494
Gujarat         7202
Name: count, dtype: int64
```

Busiest Corridors (Origin → Destination pairs)

In [162…

```
df['destination_state'] = df['destination_name'].str.extract(r'\((.*?)\)$')
corridor_counts = df.groupby(['origin_state', 'destination_state']).size().r
top_corridors = corridor_counts.sort_values(by='trip_count', ascending=False
print(top_corridors)
```

```
      origin_state destination_state  trip_count
98     Maharashtra       Maharashtra       11876
72       Karnataka         Karnataka       11107
129     Tamil Nadu        Tamil Nadu        6549
144  Uttar Pradesh     Uttar Pradesh        4978
47         Haryana         Karnataka        4976
44         Haryana           Haryana        4508
34         Gujarat           Gujarat        4491
124      Rajasthan         Rajasthan        4380
154    West Bengal       West Bengal        4004
137      Telangana         Telangana        3804
```

Average Distance and Time per Corridor

In [163…

```
corridor_avg = df.groupby(['origin_state', 'destination_state']).agg({
    'actual_time': 'mean',
    'actual_distance_to_destination': 'mean',
    'trip_uuid': 'count'
}).rename(columns={'trip_uuid': 'trip_count'}).reset_index()

# Filter for only the busiest corridors
corridor_avg = corridor_avg.sort_values(by='trip_count', ascending=False).he
print(corridor_avg)
```

```
      origin_state destination_state  actual_time  \
98     Maharashtra       Maharashtra   138.181627
72       Karnataka         Karnataka    72.443594
129     Tamil Nadu        Tamil Nadu    58.169644
144  Uttar Pradesh     Uttar Pradesh   121.704098
47         Haryana         Karnataka  1367.212219
44         Haryana           Haryana    85.098492
34         Gujarat           Gujarat    87.677800
124      Rajasthan         Rajasthan   123.349087
154    West Bengal       West Bengal   102.780969
137      Telangana         Telangana    76.173502

     actual_distance_to_destination  trip_count
98                        64.032019       11876
72                        33.600526       11107
129                       29.107871        6549
144                       50.983726        4978
47                       859.827666        4976
44                        37.616022        4508
34                        48.755096        4491
124                       62.721821        4380
154                       35.035677        4004
137                       36.643287        3804
```

Deviation from OSRM Time

In [164…

```
#Calculate how much actual delivery time deviates from predicted (OSRM) time

df['time_deviation'] = df['actual_time'] - df['osrm_time']
deviation_summary = df['time_deviation'].describe()
print(deviation_summary)
```

```
count    144867.000000
mean        203.059254
std         303.743664
min        -110.000000
25%          21.000000
50%          65.000000
75%         247.000000
max        3137.000000
Name: time_deviation, dtype: float64
```

Business Recommendation

1.Send more delivery vehicles to the states with most orders like Haryana , Maharashtra

2.Update delivery time estimates to match real travel times.

3.Plan fixed daily routes for the busiest delivery paths.

4.Find out why some routes are always slow and fix them.

5.Check unusual deliveries — either too late or too fast.

6.Update delivery time estimates to match real travel times.

7.Use bulk shipping for places that get many deliveries often.

8.Review top and bottom routes every month to stay on track.

9.Add more staff at high-volume hubs to speed up processing.

10.Give drivers proper training for routes with frequent delays.

11.Inform customers early if delays are expected.

12.Reward delivery agents who consistently meet time targets.

13.Mark and track areas with repeated delivery issues.

14.Keep extra backup vehicles ready during peak days like wedensday.

15.Review delivery times by shift — morning vs evening — and adjust.

In [164…