

Delhivery Feature Engineering

Introduction :

Delhivery, India's leading and rapidly growing integrated player, has set its sights on creating the commerce operating system. They achieve this by utilizing world-class infrastructure, ensuring the highest quality in logistics operations, and harnessing cutting-edge engineering and technology capabilities

**Purpose of the Business Case Study:** The study ensures data accuracy by fixing missing values and organizing the dataset. It identifies key features from raw data for better forecasting and finds patterns and recommendations to improve logistics.

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

!gdown https://d2beiqkhq929f0.cloudfront.net/public\_assets/assets/000/001/551/original/delhivery\_data.csv?1642751181

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181
To: /content/delhivery_data.csv?1642751181
100% 55.6M/55.6M [00:04<00:00, 12.0MB/s]
```

```
data=pd.read_csv("/content/delhivery_data.csv?1642751181")
```

Data Exploration

```
data.sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	desti
75973	training	2018-09-20 20:51:06.829984	thanos::sroute:67c77992-49e3-4594-9a75-9861ef0...	FTL	153747666682973236	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	IND131028AAB	Son
94114	test	2018-09-27 00:59:45.758955	thanos::sroute:1d848e84-b73b-47b3-90ec-4a12509...	FTL	153800998575871469	IND508223AAA	Thirumalagiri_Xroad_D (Telangana)	IND500039AAC	Hyder
56926	test	2018-09-28 16:15:13.294640	thanos::sroute:6ce329f9-0cf6-40bc-8197-6f8acc6...	Carting	153815131329438087	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	IND400072AAB	(
59811	training	2018-09-12 10:32:39.025066	thanos::sroute:6f20715d-5684-4595-a986-2836d48...	FTL	153674835902479284	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND421302AAG	Bhiwand (
65372	training	2018-09-19 08:32:50.572008	thanos::sroute:6c488175-9e94-44b2-b01a-69621bf...	FTL	153734597057178161	IND302014AAA	Jaipur_Hub (Rajasthan)	IND305001AAC	Ajmer
5 rows x 24 columns									

```
data.shape
```

```
(144867, 24)
```

The dataset has 144,867 rows and 24 columns.


```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   data                                  144867 non-null object
1   trip_creation_time                    144867 non-null object
2   route_schedule_uuid                  144867 non-null object
3   route_type                           144867 non-null object
4   trip_uuid                            144867 non-null object
5   source_center                        144867 non-null object
6   source_name                          144574 non-null object
7   destination_center                   144867 non-null object
8   destination_name                     144606 non-null object
9   od_start_time                        144867 non-null object
10  od_end_time                          144867 non-null object
11  start_scan_to_end_scan                144867 non-null float64
12  is_cutoff                            144867 non-null bool
13  cutoff_factor                        144867 non-null int64
14  cutoff_timestamp                     144867 non-null object
```

```
15 actual_distance_to_destination 144867 non-null float64
16 actual_time                    144867 non-null float64
17 osrm_time                     144867 non-null float64
18 osrm_distance                 144867 non-null float64
19 factor                        144867 non-null float64
20 segment_actual_time           144867 non-null float64
21 segment_osrm_time             144867 non-null float64
22 segment_osrm_distance         144867 non-null float64
23 segment_factor                144867 non-null float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

Initially, the dataset had 24 columns, of which 12 were of object data type, 10 were float, data type, and the remaining columns were of boolean and integer data types.

```
data.isna().sum()
```




	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

Missing value treatment :

```
Percentage_of_missing_value = (data.isnull().sum() / len(data))*100
Percentage_of_missing_value.sort_values(ascending=False).head(5)
```



	0
source_name	0.202254
destination_name	0.180165
data	0.000000
cutoff_factor	0.000000
segment_osrm_distance	0.000000

dtype: float64

The columns 'source name' and 'destination name' have missing values of 0.20% and 0.18%, respectively. Since this represents a very small fraction of the dataset, I am dropping the null values.

```
data=data.dropna()
```

```
data.isna().sum().sum()
```

 0

Correct the data types of the datetime columns.

```
data['trip_creation_time']=pd.to_datetime(data['trip_creation_time'], errors='coerce')
```

```
data['od_start_time']=pd.to_datetime(data['od_start_time'],errors='coerce')
```

```
data['od_end_time']=pd.to_datetime(data['od_end_time'],errors='coerce')
```

```
data['cutoff_timestamp']=pd.to_datetime(data['cutoff_timestamp'],errors='coerce')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 24 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   data                                  144316 non-null object
1   trip_creation_time                   144316 non-null datetime64[ns]
2   route_schedule_uuid                 144316 non-null object
3   route_type                           144316 non-null object
4   trip_uuid                            144316 non-null object
5   source_center                        144316 non-null object
6   source_name                          144316 non-null object
7   destination_center                  144316 non-null object
8   destination_name                    144316 non-null object
9   od_start_time                       144316 non-null datetime64[ns]
10  od_end_time                          144316 non-null datetime64[ns]
11  start_scan_to_end_scan               144316 non-null float64
12  is_cutoff                            144316 non-null bool
13  cutoff_factor                        144316 non-null int64
14  cutoff_timestamp                     140909 non-null datetime64[ns]
15  actual_distance_to_destination       144316 non-null float64
16  actual_time                          144316 non-null float64
17  osrm_time                            144316 non-null float64
18  osrm_distance                        144316 non-null float64
19  factor                               144316 non-null float64
20  segment_actual_time                  144316 non-null float64
21  segment_osrm_time                    144316 non-null float64
22  segment_osrm_distance                144316 non-null float64
23  segment_factor                       144316 non-null float64
dtypes: bool(1), datetime64[ns](4), float64(10), int64(1), object(8)
memory usage: 26.6+ MB
```

```
data.nunique()
```

	0
data	2
trip_creation_time	14787
route_schedule_uuid	1497
route_type	2
trip_uuid	14787
source_center	1496
source_name	1496
destination_center	1466
destination_name	1466
od_start_time	26223
od_end_time	26223
start_scan_to_end_scan	1914
is_cutoff	2
cutoff_factor	501
cutoff_timestamp	89862
actual_distance_to_destination	143965
actual_time	3182
osrm_time	1531
osrm_distance	137544
factor	45588
segment_actual_time	746
segment_osrm_time	214
segment_osrm_distance	113497
segment_factor	5663

dtype: int64

```
data.describe(include="float64")
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor	segment_actual_time	segment_os
count	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	14431
mean	963.697698	234.708498	417.996237	214.437055	285.549785	2.120178	36.175379	1
std	1038.082976	345.480571	598.940065	308.448543	421.717826	1.717065	53.524298	1
min	20.000000	9.000045	9.000000	6.000000	9.008200	0.144000	-244.000000	
25%	161.000000	23.352027	51.000000	27.000000	29.896250	1.604545	20.000000	1
50%	451.000000	66.135322	132.000000	64.000000	78.624400	1.857143	28.000000	1
75%	1645.000000	286.919294	516.000000	259.000000	346.305400	2.212280	40.000000	2
max	7898.000000	1927.447705	4532.000000	1686.000000	2326.199100	77.387097	3051.000000	161

```
data.describe(include="object")
```

	data	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name
count	144316	144316	144316	144316	144316	144316	144316	144316
unique	2	1497	2	14787	1496	1496	1466	1466
top	training	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153837029526866991	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)
freq	104632	1812	99132	101	23267	23267	15192	15192

```
# Creating the new column segment_key.
data["segment_key"] = data["trip_uuid"] + "-" + data["source_center"]+ "-" + data["destination_center"]
```

```
data["segment_key"].value_counts()
```

	count
segment_key	
trip-153755502932196495-IND160002AAC-IND562132AAA	81
trip-153802876613714747-IND000000ACB-IND600056AAB	79
trip-153690920439662353-IND000000ACB-IND600056AAB	79
trip-153854253003897121-IND000000ACB-IND600056AAB	79
trip-153751271053200074-IND000000ACB-IND600056AAB	79
...	...
trip-153852148377777486-IND411033AAA-IND410503AAA	1
trip-153673275082213870-IND151001AAA-IND160002AAC	1
trip-153747548760214385-IND621802AAA-IND608301AAA	1
trip-153809110015262811-IND385320AAA-IND385340AAB	1
trip-153812777210855788-IND506143AAA-IND506167AAB	1

26222 rows × 1 columns

dtype: int64

Creating new columns to store aggregated values at the segment level.

```
data["segment_actual_time_cumsum"]=data.groupby("segment_key")["segment_actual_time"].transform("cumsum")
```

```
data["segment_osrm_distance_cumsum"]=data.groupby("segment_key")["segment_osrm_distance"].transform("cumsum")
```

```
data["segment_osrm_time_cumsum"]=data.groupby("segment_key")["segment_osrm_time"].transform("cumsum")
```

```
data.sample()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	de:
24010	training	2018-09-17 13:03:29.015263	thanos::sroute:a97500f3-bbd4-4901-bcf1-e351aa8...	FTL	trip-153718940901502017	IND327001AAB	Banswara_KhandDPP_D (Rajasthan)	IND312605AAB	Pratapg

1 rows × 28 columns

Dropping columns that do not seem essential for analysis.

```
data.drop(columns=["is_cutoff","cutoff_factor","cutoff_timestamp","segment_factor"],inplace=True)
```

```
# Segment dictionary to aggregate and select values.
create_segment_dict = {
    'data': 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'trip_uuid': 'first',
    'source_center': 'first',
    'source_name': 'first',
    'destination_center': 'first',
    'destination_name': 'first',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'sum',
    'actual_distance_to_destination': 'sum',
    'actual_time': 'last',
    'osrm_time': 'last',
    'osrm_distance': 'last',
    'segment_actual_time_cumsum': 'last',
    'segment_osrm_time_cumsum': 'last',
    'segment_osrm_distance_cumsum': 'last'
}

# Group by Segement_key
segment_df = data.groupby('segment_key').agg(create_segment_dict).reset_index()

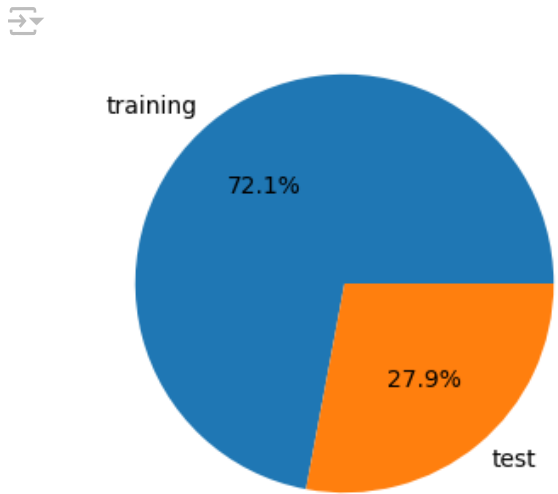
# Order the dataset by segment_key and od_end_time.
segment_df.sort_values(by=['segment_key', 'od_end_time'], ascending=[True, True], inplace=True)
```

segment\_df.head(5)

	segment_key	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination
0	trip-153671041653548748-IND209304AAA-IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND
1	trip-153671041653548748-IND462022AAA-IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND
2	trip-153671042288605164-IND561203AAB-IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND
3	trip-153671042288605164-IND572101AAA-IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND572101AAA	Tumkur_Veersagr_I (Karnataka)	IND
4	trip-153671043369099517-IND000000ACB-IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND

```
# percentage of data vs traing data in datafram

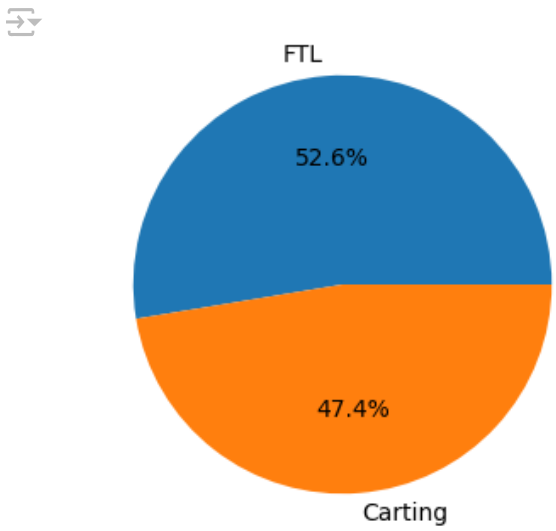
plt.figure(figsize=(4, 4))
plt.pie(segment_df["data"].value_counts(),labels=segment_df["data"].value_counts().index,autopct="%1.1f%%")
plt.show()
```



```
# Percentage of route types for trips

plt.figure(figsize=(4, 4))
```

```
plt.pie(segment_df["route_type"].value_counts(),labels=segment_df["route_type"].value_counts().index,autopct="%1.1f%%")
plt.show()
```



52.5% of trips have the route type FTL, while 47.4% have the carting route type.

- a. FTL (Full Truck Load): FTL shipments reach their destination sooner, as the truck makes no other pickups or drop-offs along the way.
- b. Carting: A handling system that consists of small vehicles (carts).

```
segment_df.groupby["data"].value_counts(normalize=True)
```

			proportion
	segment_key	data	
trip-153671042288605164-IND561203AAB-IND562101AAA		training	1.0
trip-153671042288605164-IND572101AAA-IND561203AAB		training	1.0
trip-153671046011330457-IND400072AAB-IND401104AAA		training	1.0
trip-153671052974046625-IND583101AAA-IND583201AAA		training	1.0
trip-153671052974046625-IND583119AAA-IND583101AAA		training	1.0
...		...	...
trip-153861115439069069-IND628204AAA-IND627657AAA		test	1.0
trip-153861115439069069-IND628613AAA-IND627005AAA		test	1.0
trip-153861115439069069-IND628801AAA-IND628204AAA		test	1.0
trip-153861118270144424-IND583119AAA-IND583101AAA		test	1.0
trip-153861118270144424-IND583201AAA-IND583119AAA		test	1.0

23240 rows × 1 columns

dtype: float64

Feature Engineering:

Create a new column for the time difference between the trip start time and trip end time, which will represent the total duration of the trip.

```
# Calculate the time taken between od_start_time and od_end_time in hours.

segment_df["od_time_diff_hour"]=segment_df['od_end_time'] - segment_df['od_start_time']
segment_df["od_time_diff_hour"]=(segment_df["od_time_diff_hour"].dt.total_seconds())/3600
```

Extract the source and destination details to create new columns for the source and destination state, city, and place.

```
# Split the source_name to separate state
source=segment_df["source_name"].str.split("(",expand=True)

# Extract source state
segment_df["source_state"]=source[1].str.rstrip("(")

# Extract city and place from the first part
segment_df["source_city"]=source[0].str.split("_",expand=True)[0]
segment_df["source_place"]=source[0].str.split("_",expand=True)[1]

# Extract code
source_list=source[0].str.split("_",expand=True)
source_list[3].fillna("", inplace=True)
segment_df["source_code"]= np.where(source_list[3]!="",source_list[2]+"_"+source_list[3],source_list[2])

# Split the source_name to separate state
destination=segment_df["destination_name"].str.split("(",expand=True)
```

```
# Extract source state
segment_df["destination_state"]=destination[1].str.rstrip(""))

# Extract city and place from the first part
segment_df["destination_city"]=destination[0].str.split("_",expand=True)[0]
segment_df["destination_place"]=destination[0].str.split("_",expand=True)[1]

# Extract Code
destination_code=destination[0].str.split("_",expand=True)
destination_code[3].fillna("", inplace=True)
segment_df["destination_code"]= np.where(destination_code[3]!="",destination_code[2]+"_"+ destination_code[3], destination_code[2])
```


Now I have columns for the state and city of the source and destination, therefore, delete the columns 'source name' and 'destination' since they are no longer needed.

```
segment_df.drop(columns=["source_name","destination_name"],inplace=True)
```

Extract month, year, day from column trip creation

```
segment_df["trip_creation_year"]=segment_df["trip_creation_time"].dt.year
segment_df["trip_creation_month"]=segment_df["trip_creation_time"].dt.month
segment_df["trip_creation_day"]=segment_df["trip_creation_time"].dt.day
```

```
segment_df.sample()
```



	segment_key	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	destination_center	od_star
8452	trip-153729810530379286-IND000000ACB-IND390022AAA	training	2018-09-18 19:15:05.304017	thanos::sroute:061cbd67-f34d-4ccd-b81d-557b8bd...	FTL	trip-153729810530379286	IND000000ACB	IND390022AAA	20121:19:29.

1 rows × 30 columns

✦ In-depth analysis:

```
# Performing grouping and aggregating values at the trip-level

# Actual aggregated time
segment_df["trip_actual_time_sum"]=segment_df.groupby("trip_uuid")["actual_time"].transform("sum")

# Actual aggregated distance.
segment_df["actual_distance_to_destination_sum"]=segment_df.groupby("trip_uuid")["actual_distance_to_destination"].transform("sum")

# OSRM aggregate time.
segment_df["trip_osrm_time_sum"]=segment_df.groupby("trip_uuid")["osrm_time"].transform("sum")

# OSRM aggregted distance
segment_df["trip_osrm_distance_sum"]=segment_df.groupby("trip_uuid")["osrm_distance"].transform("sum")

# Segment aggregate actual time
segment_df["trip_segment_actual_time_sum"]=segment_df.groupby("trip_uuid")["segment_actual_time_cumsum"].transform("sum")

# Segment OSRM aggregated time
segment_df["trip_segment_osrm_time_sum"]=segment_df.groupby("trip_uuid")["segment_osrm_time_cumsum"].transform("sum")

# segment OSRM aggregated distance
segment_df["trip_segment_osrm_distance_sum"]=segment_df.groupby("trip_uuid")["segment_osrm_distance_cumsum"].transform("sum")

segment_df.sample(5)
```



	segment_key	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	destination_center	od_sta
8833	trip-153731123177211255-IND382430AAB-IND382345AAA	training	2018-09-18 22:53:51.772452	thanos::sroute:064ead43-1e50-43e7-b95d-ec81409...	Carting	trip-153731123177211255	IND382430AAB	IND382345AAA	2022:53:5
21081	trip-153817845070474887-IND203135AAA-IND203396AAB	test	2018-09-28 23:47:30.705039	thanos::sroute:a0f0a15f-cfc1-4dd5-8f04-eff10e4...	FTL	trip-153817845070474887	IND203135AAA	IND203396AAB	2005:20:0
3703	trip-153696391432268335-IND712311AAA-IND834002AAB	training	2018-09-14 22:25:14.322935	thanos::sroute:fa83fd49-3327-4503-8e80-bf58ed6...	FTL	trip-153696391432268335	IND712311AAA	IND834002AAB	2022:25:1
10887	trip-153746223441181534-IND422011AAD-IND421302AAG	training	2018-09-20 16:50:34.412041	thanos::sroute:b9142a73-d68f-46ec-9afc-d145479...	FTL	trip-153746223441181534	IND422011AAD	IND421302AAG	2022:26:1
12379	trip-153756310754724120-IND521105AAA-IND534001AAA	training	2018-09-21 20:51:47.547613	thanos::sroute:d623457b-692f-4c0c-bcab-45955f1...	FTL	trip-153756310754724120	IND521105AAA	IND534001AAA	2002:46:0

5 rows × 37 columns

```
# Aggregated time diff
segment_df["trip_total_time"]=segment_df.groupby("trip_uuid")["od_time_diff_hour"].transform("sum")
```

Outlier Detection:

```
# Boxplot for 'trip_actual_time_sum', 'trip_osrm_time_sum'.

plt.figure(figsize=(8, 6))

# trip_actual_time_sum
plt.subplot(2, 2, 1)
plt.boxplot(segment_df['trip_actual_time_sum'])
plt.xlabel('trip_actual_time_sum', color="blue", fontsize=10)

plt.subplot(2, 2, 2)
sns.histplot(segment_df['trip_actual_time_sum'], kde=True, color="blue")
plt.xlabel('trip_actual_time_sum', color="blue", fontsize=10)

# trip_osrm_time_sum

plt.subplot(2, 2, 3)
plt.boxplot(segment_df['trip_osrm_time_sum'])
plt.xlabel('trip_osrm_time_sum', color="blue", fontsize=10)

plt.subplot(2, 2, 4)
sns.histplot(segment_df['trip_osrm_time_sum'], kde=True, color="blue")
plt.xlabel('trip_osrm_time_sum', color="blue", fontsize=10)

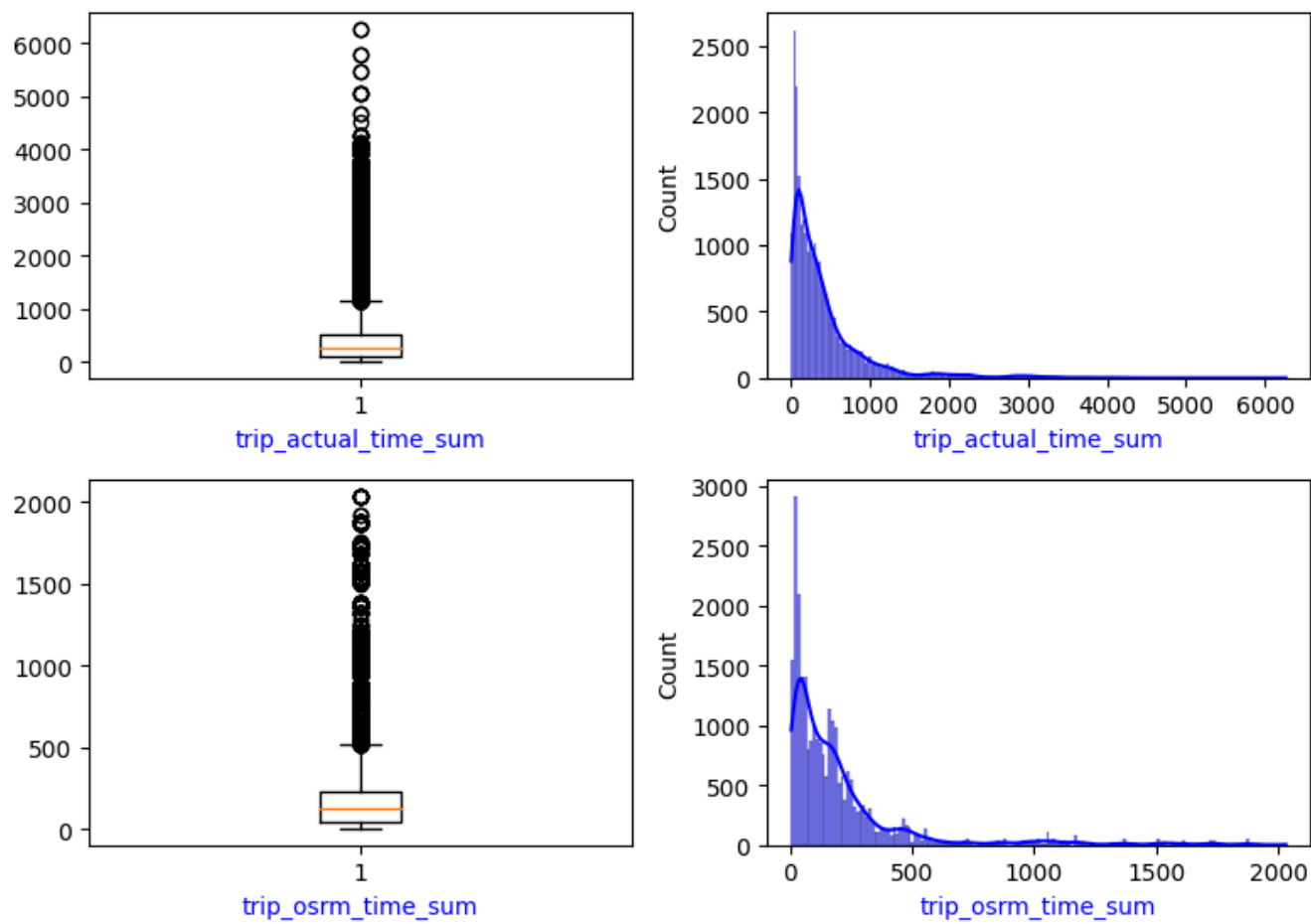
plt.suptitle('Outlier Detection of Columns: trip_actual_time_sum and trip_osrm_time_sum' ,color="blue")

plt.tight_layout()
plt.show()
```





### Outlier Detection of Columns: trip\_actual\_time\_sum and trip\_osrm\_time\_sum



The above plots show that both columns, trip\_actual\_time\_sum and trip\_osrm\_time\_sum, have many outliers. Additionally, the spread to the right indicates that these columns contain outliers with very large values.

```
# trip actual distance sum

plt.figure(figsize=(8, 6))
plt.subplot(2, 2, 1)
plt.boxplot(segment_df['actual_distance_to_destination_sum'])
plt.xlabel('actual_distance_to_destination_sum', color="blue", fontsize=10)

plt.subplot(2, 2, 2)
sns.histplot(segment_df['actual_distance_to_destination_sum'], kde=True, color="blue")
plt.xlabel('actual_distance_to_destination_sum', color="blue", fontsize=10)

# trip_osrm_distance_sum

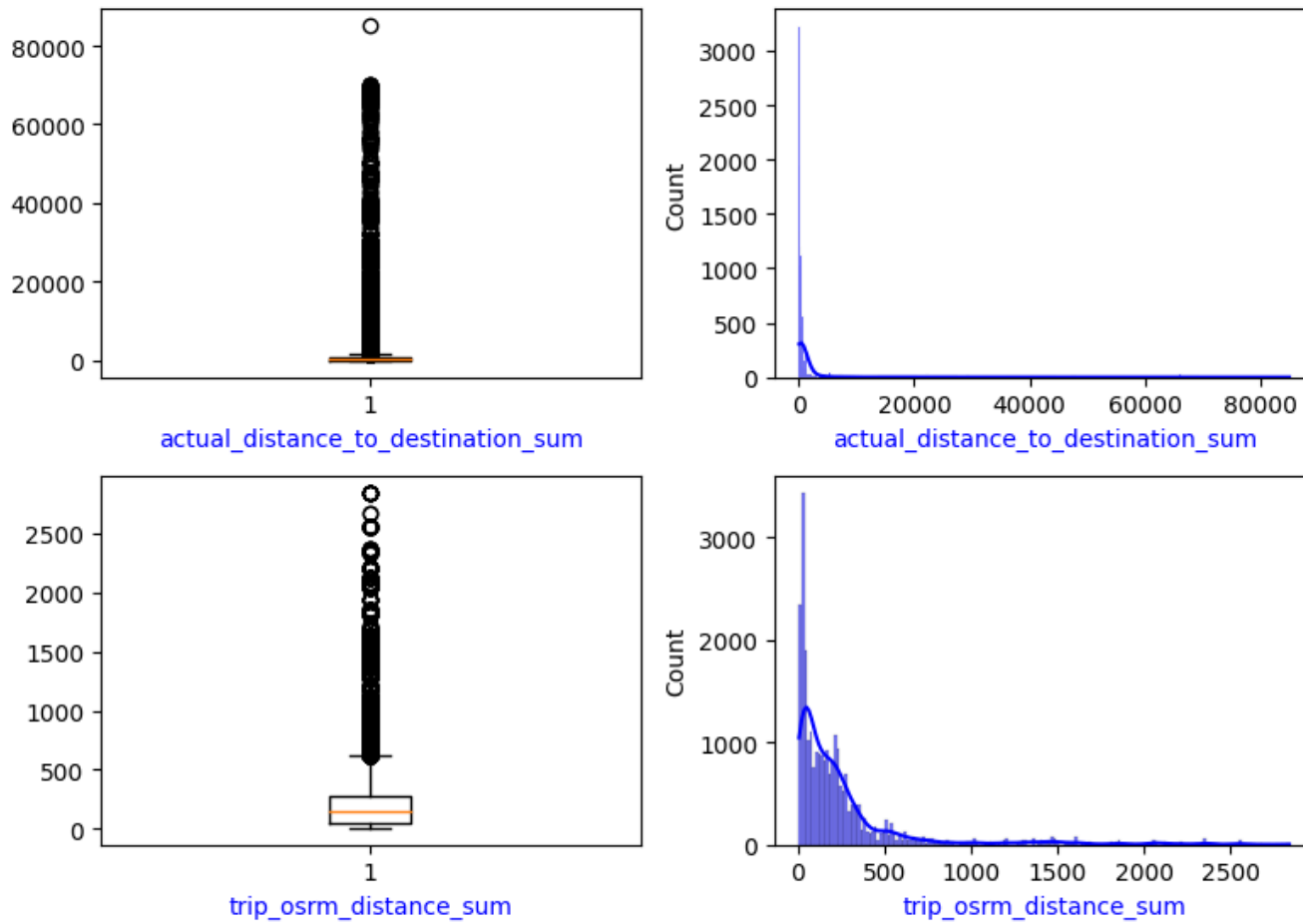
plt.subplot(2, 2, 3)
plt.boxplot(segment_df['trip_osrm_distance_sum'])
plt.xlabel('trip_osrm_distance_sum', color="blue", fontsize=10)

plt.subplot(2, 2, 4)
sns.histplot(segment_df['trip_osrm_distance_sum'], kde=True, color="blue")
plt.xlabel('trip_osrm_distance_sum', color="blue", fontsize=10)

plt.suptitle('Outlier Detection of Columns: actual_distance_to_destination_sum and trip_osrm_distance_sum' ,color="blue")
plt.tight_layout()
plt.show()
```



## Outlier Detection of Columns: actual\_distance\_to\_destination\_sum and trip\_osrm\_distance\_sum



The columns actual\_distance\_to\_destination and trip\_osrm\_distance\_sum contain many outliers. The distribution of data points shows a significant number of values that are spread far to the right.

```
# Boxplot for 'trip_segment_actual_time_sum', 'trip_segment_osrm_time_sum','trip_segment_osrm_distance_sum'

plt.figure(figsize=(16, 8))

plt.subplot(3, 3, 1)
plt.boxplot(segment_df['trip_segment_actual_time_sum'])
plt.xlabel('trip_segment_actual_time_sum', color="blue", fontsize=10)

plt.subplot(3, 3, 2)
plt.boxplot(segment_df['trip_segment_osrm_time_sum'])
plt.xlabel('trip_segment_osrm_time_sum', color="blue", fontsize=10)

plt.subplot(3, 3, 3)
plt.boxplot(segment_df['trip_segment_osrm_distance_sum'])
plt.xlabel('trip_segment_osrm_distance_sum', color="blue", fontsize=10)

# histplot for 'trip_segment_actual_time_sum', 'trip_segment_osrm_time_sum','trip_segment_osrm_distance_sum'
plt.subplot(3, 3, 4)
sns.histplot(segment_df['trip_segment_actual_time_sum'], kde=True, color="blue")
plt.xlabel('trip_segment_actual_time_sum', color="blue", fontsize=10)
plt.suptitle('Outlier Detection' ,color="blue")

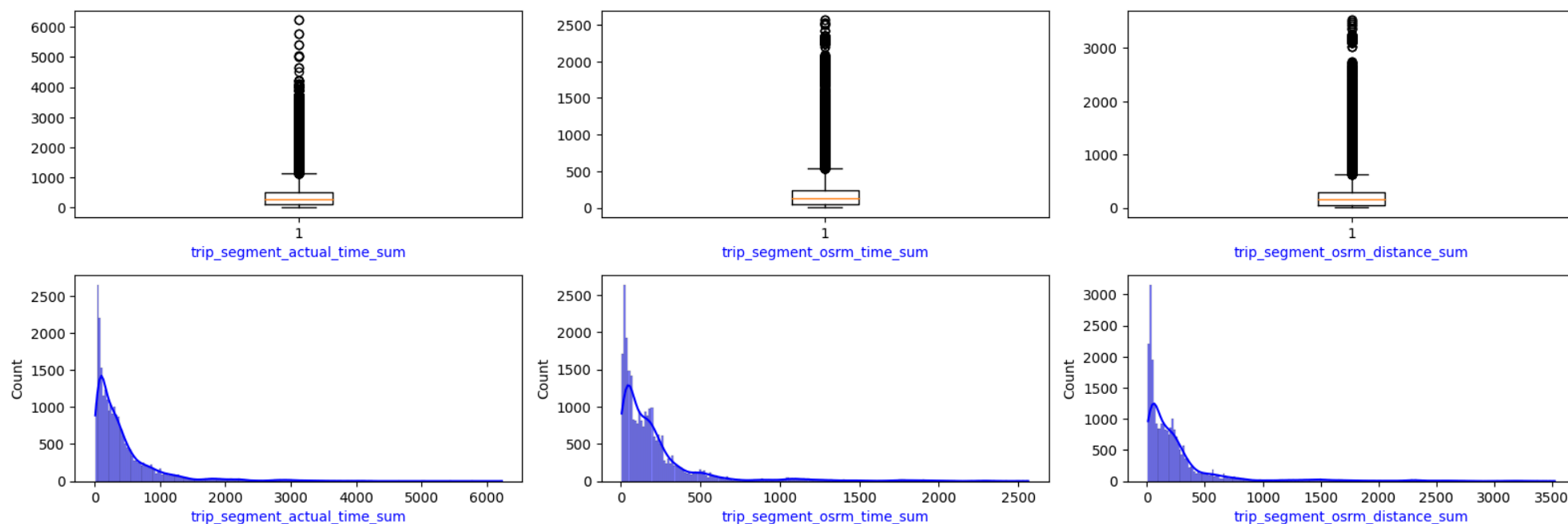
plt.subplot(3, 3,5)
sns.histplot(segment_df['trip_segment_osrm_time_sum'], kde=True, color="blue")
plt.xlabel('trip_segment_osrm_time_sum', color="blue", fontsize=10)

plt.subplot(3, 3,6)
sns.histplot(segment_df['trip_segment_osrm_distance_sum'], kde=True, color="blue")
plt.xlabel('trip_segment_osrm_distance_sum', color="blue", fontsize=10)

plt.suptitle('Outlier Detection of columns trip_segment_actual_time_sum, trip_segment_osrm_time_sum, trip_segment_osrm_distance_sum : ' ,
plt.tight_layout()
plt.show()
```



Outlier Detection of columns trip\_segment\_actual\_time\_sum, trip\_segment\_osrm\_time\_sum, trip\_segment\_osrm\_distance\_sum :



If we observe the plot of the columns trip\_segment\_actual\_time\_sum, trip\_segment\_osrm\_time\_sum, and trip\_segment\_osrm\_distance\_sum, we can see the spread of data points, indicating that these columns contain a significant number of outliers.

```
# Remove Outlier
# trip_actual_time_sum
Q1 = segment_df['trip_actual_time_sum'].quantile(0.25)
Q3 = segment_df['trip_actual_time_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[(segment_df['trip_actual_time_sum'] >= lower_limit) & (segment_df['trip_actual_time_sum'] <= upper_limit)]
```

```
# trip_osrm_time_sum
Q1 = segment_df['trip_osrm_time_sum'].quantile(0.25)
Q3 = segment_df['trip_osrm_time_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[(segment_df['trip_osrm_time_sum'] >= lower_limit) & (segment_df['trip_osrm_time_sum'] <= upper_limit)]
```

# Boxplot for 'trip\_actual\_time\_sum', 'trip\_osrm\_time\_sum', 'trip\_osrm\_distance\_sum',

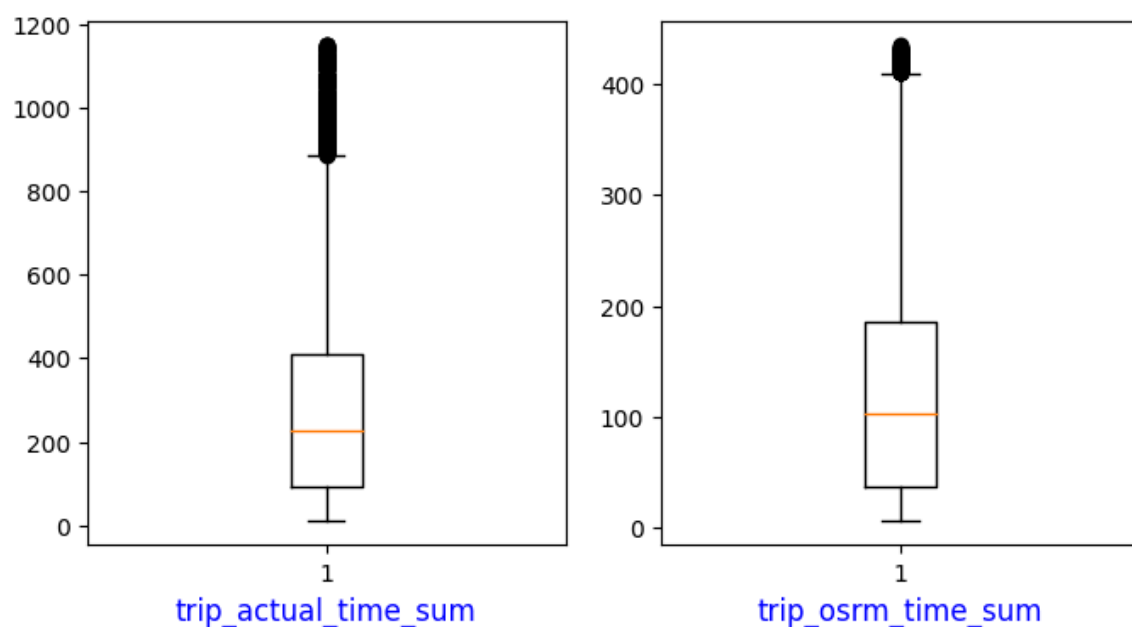
```
plt.figure(figsize=(8, 4))
```

```
# trip_actual_time_sum
plt.subplot(1, 2, 1)
plt.boxplot(segment_df['trip_actual_time_sum'])
plt.xlabel('trip_actual_time_sum', color="blue", fontsize=12)
```

```
# trip_osrm_time_sum
plt.subplot(1, 2, 2)
plt.boxplot(segment_df['trip_osrm_time_sum'])
plt.xlabel('trip_osrm_time_sum', color="blue", fontsize=12)
```



Text(0.5, 0, 'trip\_osrm\_time\_sum')



```
# trip_actual_distance_sum
Q1= segment_df['actual_distance_to_destination_sum'].quantile(0.25)
Q3 = segment_df['actual_distance_to_destination_sum'].quantile(0.75)
IQR= Q3 - Q1
```

```

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[-(segment_df['actual_distance_to_destination_sum'] < lower_limit) | (segment_df['actual_distance_to_destination_sum'] > upper_limit)]

# trip_osrm_distance_sum
Q1= segment_df['trip_osrm_distance_sum'].quantile(0.25)
Q3 = segment_df['trip_osrm_distance_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[-(segment_df['trip_osrm_distance_sum'] < lower_limit) | (segment_df['trip_osrm_distance_sum'] > upper_limit)]

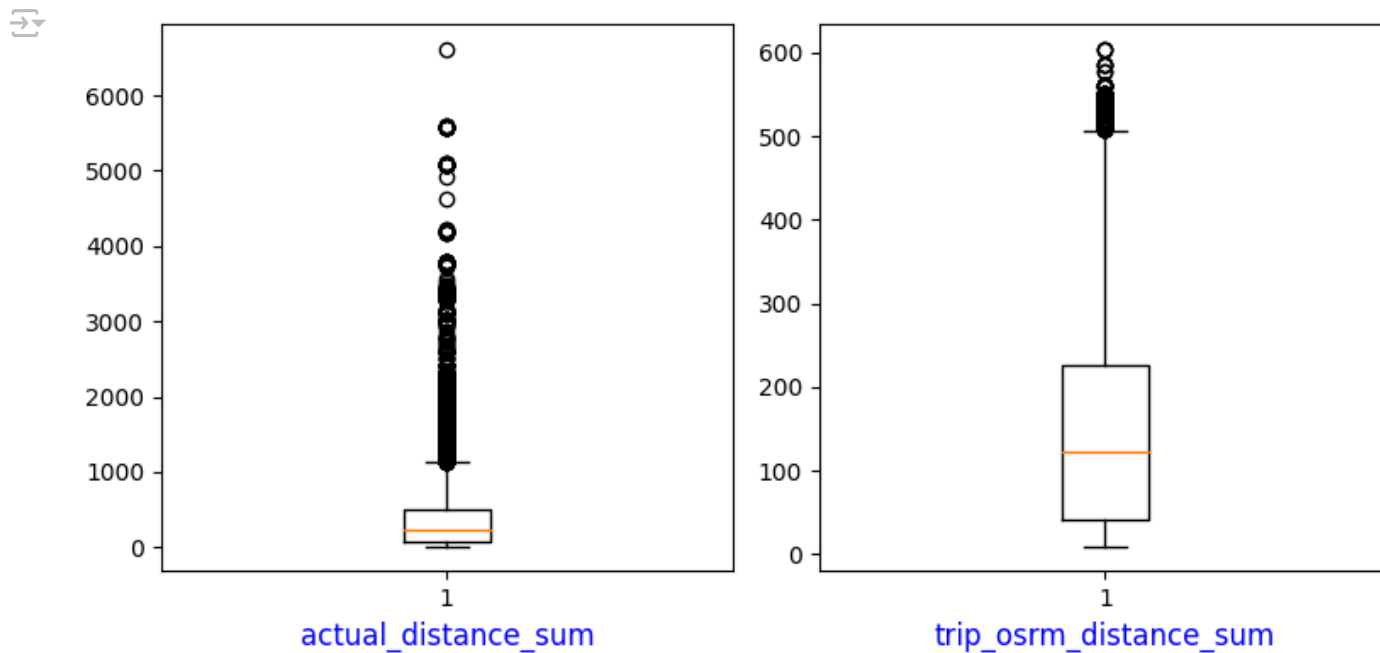
# trip_actual_distance_sum
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.boxplot(segment_df['actual_distance_to_destination_sum'])
plt.xlabel('actual_distance_sum', color="blue", fontsize=12)

# trip_osrm_distance_sum

plt.subplot(1, 2, 2)
plt.boxplot(segment_df['trip_osrm_distance_sum'])
plt.xlabel('trip_osrm_distance_sum', color="blue", fontsize=12)

plt.tight_layout()
plt.show()

```



```

# Remove Outlier

# trip_segment_time_sum
Q1 = segment_df['trip_segment_actual_time_sum'].quantile(0.25)
Q3 = segment_df['trip_segment_actual_time_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[-(segment_df['trip_segment_actual_time_sum'] < lower_limit) | (segment_df['trip_segment_actual_time_sum'] > upper_limit)]

# trip_segment_osrm_time_sum
Q1 = segment_df['trip_segment_osrm_time_sum'].quantile(0.25)
Q3 = segment_df['trip_segment_osrm_time_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[-(segment_df['trip_segment_osrm_time_sum'] < lower_limit) | (segment_df['trip_segment_osrm_time_sum'] > upper_limit)]

# trip_segment_osrm_distance_sum
Q1= segment_df['trip_segment_osrm_distance_sum'].quantile(0.25)
Q3 = segment_df['trip_segment_osrm_distance_sum'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
segment_df = segment_df[-(segment_df['trip_segment_osrm_distance_sum'] < lower_limit) | (segment_df['trip_segment_osrm_distance_sum'] > upper_limit)]

# Boxplot for 'trip_segment_actual_time_sum', 'trip_segment_osrm_time_sum', 'trip_segment_osrm_distance_sum',

plt.figure(figsize=(16, 8))

# trip_segment_actual_time_sum
plt.subplot(2, 3, 1)
plt.boxplot(segment_df['trip_segment_actual_time_sum'])
plt.xlabel('trip_segment_actual_time_sum', color="blue", fontsize=12)

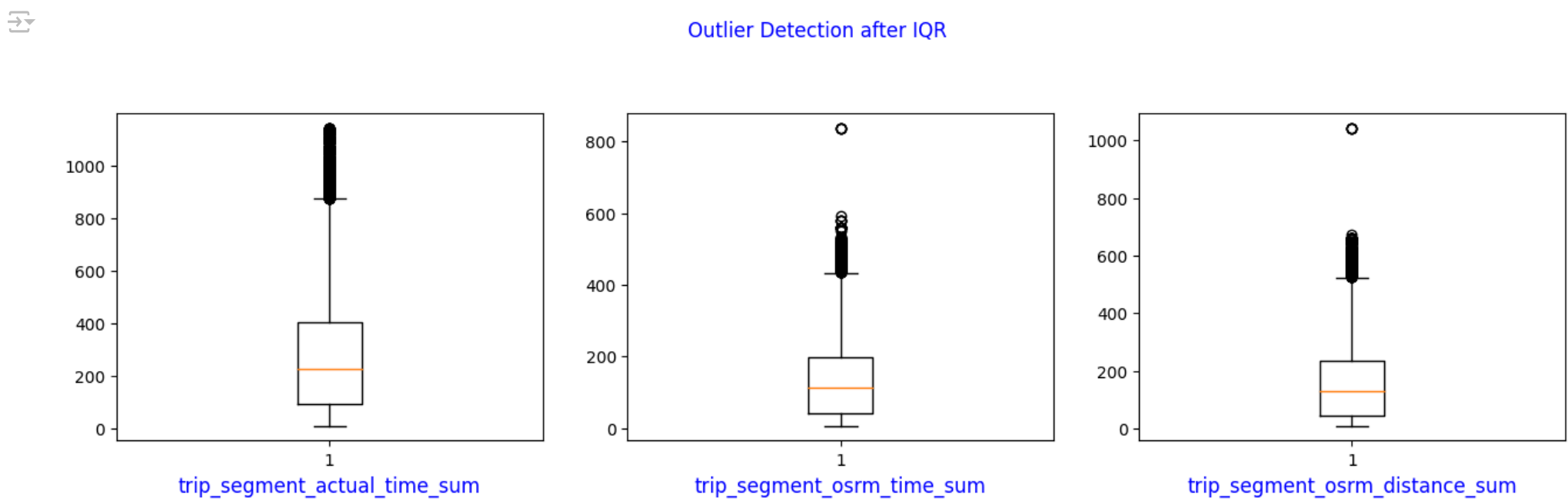
```

```
# trip_segment_osrm_time_sum
plt.subplot(2, 3, 2)
plt.boxplot(segment_df['trip_segment_osrm_time_sum'])
plt.xlabel('trip_segment_osrm_time_sum', color="blue", fontsize=12)

# trip_segment_osrm_distance_sum

plt.subplot(2, 3, 3)
plt.boxplot(segment_df['trip_segment_osrm_distance_sum'])
plt.xlabel('trip_segment_osrm_distance_sum', color="blue", fontsize=12)

plt.suptitle('Outlier Detection after IQR' ,color="blue")
plt.show()
```



After removing the outliers, we can observe that the number of outliers has decreased.

```
# Categorical columns of dataframe

cat_col = segment_df.dtypes=="object"
cat_col = list(cat_col[cat_col].index)
cat_col
```

➡

```
['segment_key',
 'data',
 'route_schedule_uuid',
 'route_type',
 'trip_uuid',
 'source_center',
 'destination_center',
 'source_state',
 'source_city',
 'source_place',
 'source_code',
 'destination_state',
 'destination_city',
 'destination_place',
 'destination_code']
```

```
segment_df[cat_col].sample()
```

➡


	segment_key	data	route_schedule_uuid	route_type	trip_uuid	source_center	destination_center	source_state	source_city	s
9186	trip-153733050554747982-IND517583AAA-IND516101AAA	training	thanos::sroute:cce26bb2-2365-4c9c-88f4-74322d1...	Carting	trip-153733050554747982	IND517583AAA	IND516101AAA	Andhra Pradesh	Puttur	

One hot encoding on categorical columns

```
from sklearn.preprocessing import LabelEncoder

col="data"
label_encoder=LabelEncoder()
segment_df[col]=label_encoder.fit_transform(segment_df[col])


segment_df[col].value_counts()
```



count	
data	
1	18893
0	7329

**dtype:** int64


```
segment_df["route_type"]=label_encoder.fit_transform(segment_df["route_type"])
segment_df["route_type"].value_counts()
```



count	
route_type	
1	13798
0	12424

**dtype:** int64

```
segment_df[["data","route_type"]].info()
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 23240 entries, 2 to 26221
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
0    data        23240 non-null   int64
1   route_type  23240 non-null   int64
dtypes: int64(2)
memory usage: 544.7 KB
```

We can observe that after applying one-hot encoding to the data and route\_type columns, the data type has changed from object to int.

```
# Numeric columns of segment_df
num_col = segment_df.dtypes=="float64"
num_col = list(num_col[num_col].index)
num_df=segment_df[num_col]
```

### Normalize/ Standardize the numerical features using StandardScaler.

```
# trip_actual_time_sum
# calculate the minimum and maximum values of the variable
min_value = num_df['trip_actual_time_sum'].min()
max_value = num_df['trip_actual_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```



```
Min value is 9.0 and Max value is 1149.0
```

```
from sklearn.preprocessing import StandardScaler
```

```
standard_scale = StandardScaler()
num_df["scaled_trip_actual_time_sum"] = standard_scale.fit_transform(num_df[['trip_actual_time_sum']])
```



```
<ipython-input-59-fb130d09aa9c>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
num_df["scaled_trip_actual_time_sum"] = standard_scale.fit_transform(num_df[['trip_actual_time_sum']])
```

```
min_value = num_df['scaled_trip_actual_time_sum'].min()
max_value = num_df['scaled_trip_actual_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```



```
Min value is -1.1851678952674702 and Max value is 3.687852307078432
```


```
# trip_osrm_time_sum
min_value = num_df['trip_osrm_time_sum'].min()
max_value = num_df['trip_osrm_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```



```
Min value is 6.0 and Max value is 434.0
```

```
num_df["scaled_trip_osrm_time_sum"] = standard_scale.fit_transform(num_df[['trip_osrm_time_sum']])
```

```
min_value = num_df['scaled_trip_osrm_time_sum'].min()
max_value = num_df['scaled_trip_osrm_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```



```
Min value is -1.2095129310997053 and Max value is 3.107797549435848
<ipython-input-62-6742ed8d348a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
`num_df["scaled_trip_osrm_time_sum"] = standard_scale.fit_transform(num_df[['trip_osrm_time_sum']])`

```
# trip_actual_distance_sum
min_value = num_df['actual_distance_to_destination_sum'].min()
max_value = num_df['actual_distance_to_destination_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is 9.00246144174878 and Max value is 6619.055121497286

```
num_df['scaled_trip_actual_time_sum']= standard_scale.fit_transform(num_df[['actual_distance_to_destination_sum']])
```

```
min_value = num_df['scaled_trip_actual_time_sum'].min()
max_value = num_df['scaled_trip_actual_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is -0.7246740294242073 and Max value is 11.934408036359667  
<ipython-input-64-da87f9323bb3>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
`num_df['scaled_trip_actual_time_sum']= standard_scale.fit_transform(num_df[['actual_distance_to_destination_sum']])`

```
# trip_osrm_distance_sum
```

```
min_value = num_df['trip_osrm_distance_sum'].min()
max_value = num_df['trip_osrm_distance_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is 9.0729 and Max value is 604.4885

```
num_df["scaled_trip_osrm_distance_sum"] = standard_scale.fit_transform(num_df[['trip_osrm_distance_sum']])
min_value= num_df['scaled_trip_osrm_distance_sum'].min()
max_value = num_df['scaled_trip_osrm_distance_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is -1.139119752146302 and Max value is 3.6682950635440985  
<ipython-input-66-df021e27a6bd>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
`num_df["scaled_trip_osrm_distance_sum"] = standard_scale.fit_transform(num_df[['trip_osrm_distance_sum']])`

```
# trip_segment_actual_time_sum
```

```
min_value = num_df['trip_segment_actual_time_sum'].min()
max_value = num_df['trip_segment_actual_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is 9.0 and Max value is 1143.0

```
num_df["scaled_trip_segment_actual_time_sum"] = standard_scale.fit_transform(num_df[['trip_segment_actual_time_sum']])
min_value = num_df['scaled_trip_segment_actual_time_sum'].min()
max_value = num_df['scaled_trip_segment_actual_time_sum'].max()
```

```
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is -1.180981734556477 and Max value is 3.6913460063903716  
<ipython-input-68-e3b2260274fb>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
`num_df["scaled_trip_segment_actual_time_sum"] = standard_scale.fit_transform(num_df[['trip_segment_actual_time_sum']])`

```
# trip_segment_osrm_time_sum
```

```
min_value = num_df['trip_segment_osrm_time_sum'].min()
max_value = num_df['trip_segment_osrm_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is 6.0 and Max value is 836.0

```
num_df["scaled_trip_segment_osrm_time_sum"] = standard_scale.fit_transform(num_df[['trip_segment_osrm_time_sum']])
```

```
min_value = num_df['scaled_trip_segment_osrm_time_sum'].min()
max_value = num_df['scaled_trip_segment_osrm_time_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")
```

➡ Min value is -1.1926573459045315 and Max value is 6.406055165448478  
<ipython-input-70-46d397d7fa99>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead



```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy.
num_df["scaled_trip_segment_osrm_time_sum"] = standard_scale.fit_transform(num_df[['trip_segment_osrm_time_sum']])

# trip_segment_osrm_distance_sum

min_value = num_df['trip_segment_osrm_distance_sum'].min()
max_value = num_df['trip_segment_osrm_distance_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")

🔗 Min value is 9.0729 and Max value is 1040.8056

num_df["scaled_trip_segment_osrm_distance_sum"] = standard_scale.fit_transform(num_df[['trip_segment_osrm_distance_sum']])

min_value = num_df['scaled_trip_segment_osrm_distance_sum'].min()
max_value = num_df['scaled_trip_segment_osrm_distance_sum'].max()
print(f" Min value is {min_value} and Max value is {max_value}")

🔗 Min value is -1.134332190234546 and Max value is 6.697735392878592
<ipython-input-72-d80550fcc00d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy.
num_df["scaled_trip_segment_osrm_distance_sum"] = standard_scale.fit_transform(num_df[['trip_segment_osrm_distance_sum']])

# trip_total_time

min_value = num_df['trip_total_time'].min()
max_value = num_df['trip_total_time'].max()
print(f" Min value is {min_value} and Max value is {max_value}")

🔗 Min value is 0.3910244747222222 and Max value is 56.51645870777777

num_df['scaled_trip_total_time'] = standard_scale.fit_transform(num_df[['trip_total_time']])
min_value = num_df['scaled_trip_total_time'].min()
max_value = num_df['scaled_trip_total_time'].max()
print(f" Min value is {min_value} and Max value is {max_value}")

🔗 Min value is -1.2753413213717695 and Max value is 8.081055235528812
<ipython-input-74-bd8f8227698f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy.
num_df['scaled_trip_total_time'] = standard_scale.fit_transform(num_df[['trip_total_time']])
```

▼ Hypothesis Testing:

```
# Trip Actual Aggregated time and OSRM Aggregated time.
```

	trip_actual_time_sum	trip_osrm_time_sum	actual_distance_to_destination	trip_osrm_distance_sum	trip_segment_actual_time_sum	trip_segment_osrm
count	23240.000000	23240.000000	23240.000000	23240.000000	23240.000000	232
mean	286.259552	125.906024	178.899664	150.156975	283.865189	1
std	233.946194	99.137926	406.075982	123.856258	232.747972	1
min	9.000000	6.000000	9.001351	9.072900	9.000000	
25%	92.000000	37.000000	45.780269	40.269675	91.000000	
50%	228.000000	103.000000	61.526688	121.709400	226.000000	1
75%	410.000000	186.000000	134.225085	226.547400	405.000000	1
max	1149.000000	434.000000	6619.055121	604.488500	1143.000000	8

We can see that the mean value of actual time spent on package delivery differs from the mean value of OSRM time. The actual aggregate time is significantly larger than the OSRM aggregate time at both the trip and segment levels.

Similarly, the mean value of the actual distance covered to deliver a package is much larger than the mean value of OSRM distance at both the trip and segment levels.

To assess the significant difference between the means, we will conduct a statistical test.



```
# h0: Trip Actual Aggregated Time is equal to Trip OSRM Aggregated Time
# h1: Trip Actual Aggregated Time is Greater than the Trip OSRM Aggregated Time
```

```
from scipy.stats import ttest_rel, ttest_ind
tstat , p_val = ttest_rel(num_df["trip_actual_time_sum"] , num_df["trip_osrm_time_sum"] , alternative="greater")
print("tstat :",f"{tstat}")
print("p_value :",f"{p_val}")
```

```
⇒ tstat : 150.90986158509034
   p_value : 0.0
```

```
if p_val<0.05:
    print("Reject null hypothesis")
    print("Actual Aggregated Time to deliver a package is greater than the OSRM Aggregated Time")
else:
    print("Accept null hypothesis")
    print("Actual Time to deliver a package is Equal to the OSRM Time")
```

```
⇒ Reject null hypothesis
   Actual Aggregated Time to deliver a package is greater than the OSRM Aggregated Time
```

```
# Actual Aggregated Time and Segment Aggregated Time.
# h0: Actual Aggregate Time is Equal to Segment Aggregated Time
# h1: Actual Aggregated time is Greater than Segmen Agrregated Time
```

```
tstats , p_val= ttest_rel(num_df["trip_actual_time_sum"],num_df["trip_segment_actual_time_sum"], alternative="greater")
print("tstat :",f"{tstats}")
print("p_value :",f"{p_val}")
```

```
⇒ tstat : 120.77582147153326
   p_value : 0.0
```

```
if p_val<0.05:
    print("Reject null hypothesis")
    print("Actual Aggregated Time to deliver a package is Greater than the Segment Aggregated Time")
else:
    print("Accept null hypothesis")
    print("Actual Aggregated Time to deliver a package is lesser or equal to the Segment Aggregated Time")
```

```
⇒ Reject null hypothesis
   Actual Aggregated Time to deliver a package is Greater than the Segment Aggregated Time
```

```
# Actual Distance Aggregated value and OSRM Distance Aggregated value
# h0: Actual Aggregated Distance equal to OSRM Aggregated Distance
# h1: Actual Aggregated Distance greater than OSRM Aggregated Distance
```

```
tstats , p_val= ttest_rel(num_df["actual_distance_to_destination_sum"],num_df["trip_osrm_distance_sum"], alternative="greater")
print("tstat :",f"{tstats}")
print("p_value :",f"{p_val}")
```

```
⇒ tstat : 84.53552960484753
   p_value : 0.0
```

```
if p_val<0.05:
    print("Reject null hypothesis")
    print("Actual Aggregated Distance to deliver a package is Greater than the OSRM Aggregated Distance")
else:
    print("Accept null hypothesis")
    print("Actual Aggregated Distance to deliver a package is lesser or equal to the OSRM Aggregated Distance")
```

```
⇒ Reject null hypothesis
   Actual Aggregated Distance to deliver a package is Greater than the OSRM Aggregated Distance
```

```
# OSRM Aggrgated distance and Segment OSEM aggregated Distance
```

```
# h0: Trip OSRM Aggrgated Distance is equal to Segment OSRM Aggregated Distance
# h1: Trip OSRM Aggregated Distance less than the Segment OSRM Aggregated Distance
```

```
tstats , p_val= ttest_rel(num_df["trip_osrm_distance_sum"] , num_df["trip_segment_osrm_distance_sum"], alternative="less")
print("tstat :",f"{tstats}")
print("p_value :",f"{p_val}")
```

```
⇒ tstat : -53.93936261948717
   p_value : 0.0
```

```
if p_val<0.05:
    print("Reject null hypothesis")
    print("Trip OSRM Aggregated Distance to deliver a package is lesser than the Segment OSRM Aggregated Distance ")
else:
    print("Accept null hypothesis")
    print("Trip OSRM Aggregated Distance to deliver a package is greater or equal to the Segment Aggregated Distance")
```

```
⇒ Reject null hypothesis
   Trip OSRM Aggregated Distance to deliver a package is lesser than the Segment OSRM Aggregated Distance
```

```
# OSRM Aggregated Time and Segment OSRM Aggregated Time.
# h0: OSRM Aggregated Time is Equal to Segment Aggregated Time
# h1: OSRM Aggregated time is Greater than Segmen Agrregated Time

tstats , p_val= ttest_rel(num_df["trip_osrm_time_sum"] , num_df["trip_segment_osrm_time_sum"], alternative="less")
print("tstat :",f"{tstats}")
print("p_value :",f"{p_val}")

tstat : -66.8925633080991
p_value : 0.0

if p_val<0.05:
    print("Reject null hypothesis")
    print("OSRM Aggregated Time to deliver a package is lesser than the Segment OSRM Aggregated Time")
else:
    print("Accept null hypothesis")
    print("OSRM Aggregated Time to deliver a package is greater or equal to the Segment Aggregated Time")

Reject null hypothesis
OSRM Aggregated Time to deliver a package is lesser than the Segment OSRM Aggregated Time
```

Conclusion:

- 1. Time Analysis: The actual aggregated time for package delivery is consistently greater than both the OSRM aggregated time and the segment aggregated time. This suggests that the real-world delivery times are longer than the estimates provided by OSRM.
  - 2. Distance Analysis: The actual aggregated distance covered for deliveries is also greater than the OSRM aggregated distance.
  - 3. Segment Comparison: The trip OSRM aggregated distance is less than the segment OSRM aggregated distance.
- The OSRM metrics generally underestimate both time and distance compared to actual values, suggesting a need for further investigation into routing accuracy and efficiency in the delivery process.

```
# Further analysys of catocritical columns of Dataset
segment_df[cat_col].sample()
```

	segment_key	data	route_schedule_uuid	route_type	trip_uuid	source_center	destination_center	source_state	source_city	sou
7547	trip-153722735361373998-IND421502AAA-IND421302AAR	1	thanos::sroute:bc42a0f4-e872-450d-8419-18cc20b...	0	trip-153722735361373998	IND421502AAA	IND421302AAR	Maharashtra	Mumbai	

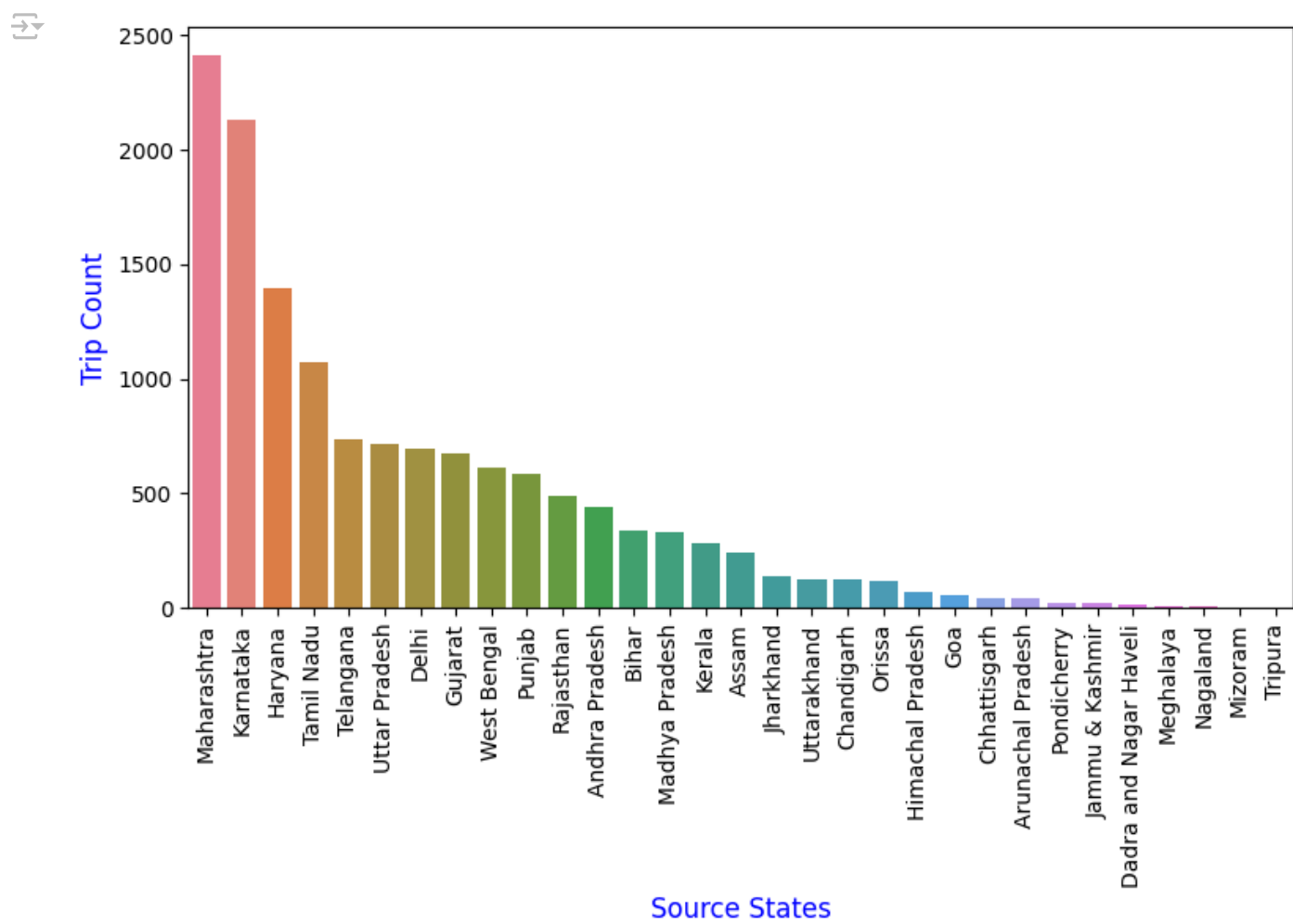
```
source_states = segment_df.groupby("source_state")["trip_uuid"].nunique().sort_values(ascending=False)
source_states=pd.DataFrame(source_states).reset_index()
source_states.rename(columns={"trip_uuid":"trip_count"}, inplace=True)
```

```
source_states.head(5)
```

	source_state	trip_count
0	Maharashtra	2416
1	Karnataka	2131
2	Haryana	1394
3	Tamil Nadu	1073
4	Telangana	738

```
# source_state
import warnings
warnings.filterwarnings("ignore")

plt.figure(figsize=(8,6))
sns.barplot(x="source_state",y="trip_count",data=source_states, palette="husl")
plt.xlabel("Source States",color="blue",fontsize=12)
plt.ylabel("Trip Count",color="blue",fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



We can observe that Maharashtra tops the list with the highest trip count (2,416) as the source state from which packages are sent to other states, followed by Karnataka (2,131) and Haryana (1,394). Tamil Nadu, Telangana, and Uttar Pradesh also have a high number of trip counts.

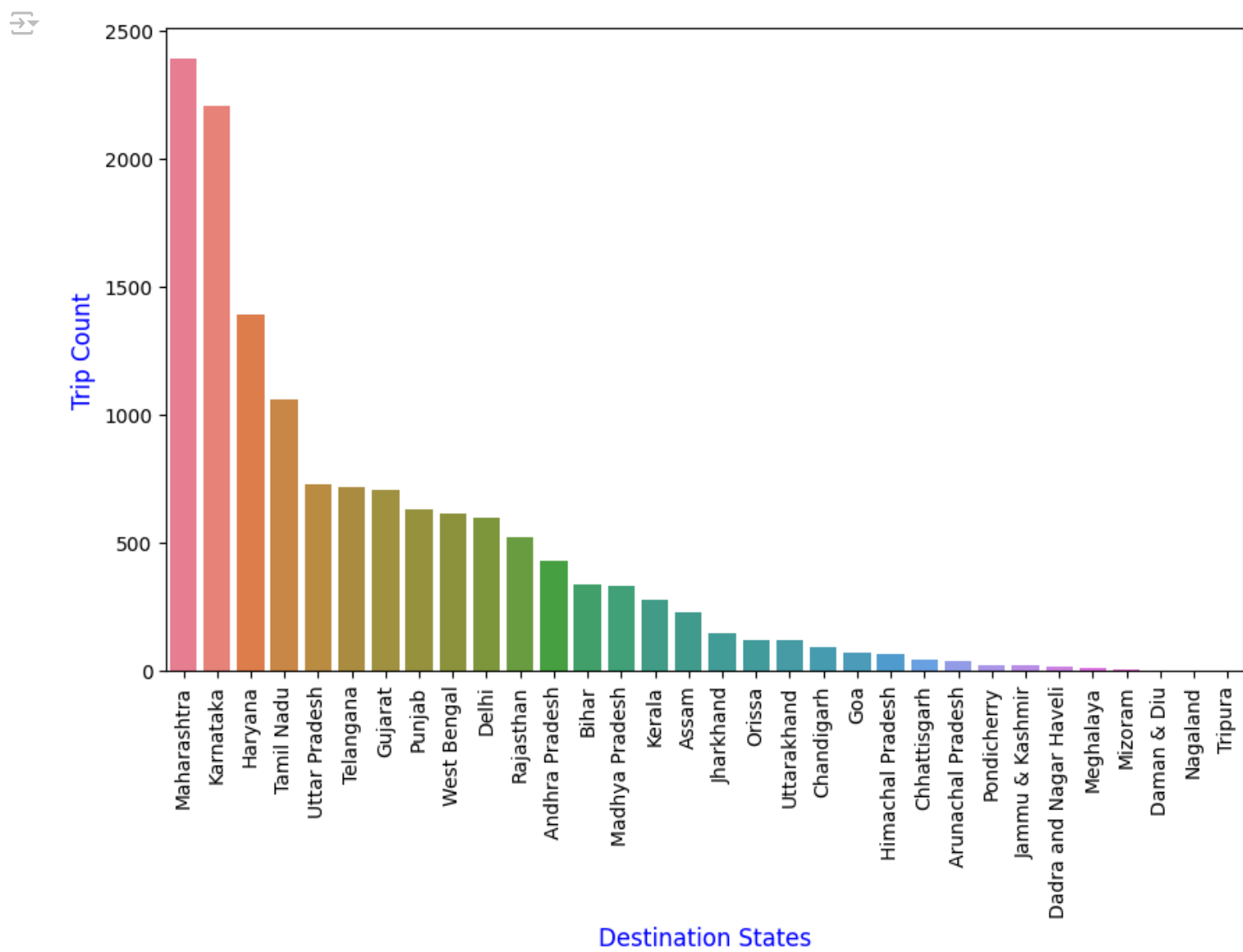
```
destination_states = segment_df.groupby("destination_state")["trip_uuid"].nunique().sort_values(ascending=False)
destination_states=pd.DataFrame(destination_states).reset_index()
destination_states.rename(columns={"trip_uuid":"trip_count"}, inplace=True)
```

```
destination_states.head(5)
```

	destination_state	trip_count
0	Maharashtra	2390
1	Karnataka	2205
2	Haryana	1390
3	Tamil Nadu	1057
4	Uttar Pradesh	727

# Destination states

```
plt.figure(figsize=(10,6))
sns.barplot(x="destination_state",y="trip_count",data=destination_states, palette="husl")
plt.xlabel("Destination States",color="blue",fontsize=12)
plt.ylabel("Trip Count",color="blue",fontsize=12)
plt.xticks(rotation=90)
plt.show()
```



We can observe that Maharashtra tops the list with the highest trip count (2,390) as the destination state where packages are sent, followed by Karnataka (2,205) and Haryana (1,390). Tamil Nadu, Telangana, and Uttar Pradesh also have a high number of trip counts.

**Creating new column corridor\_id by concatenating "source\_center" and "destination\_center" to analysys the most functional corridor.**

```
segment_df["corridor_id"] = segment_df["source_center"] + "_" + segment_df["destination_center"]
segment_df["corridor_id"].nunique()
```

2597

# Number of trips between the two corridors

```
busiest_corridor=segment_df.groupby("corridor_id")["trip_uuid"].nunique().sort_values(ascending=False)
busiest_corridor=pd.DataFrame(busiest_corridor).reset_index()
busiest_corridor.rename(columns={"trip_uuid":"trip_count"},inplace=True)
busiest_corridor
```

	corridor_id	trip_count
0	IND562132AAA_IND560300AAA	151
1	IND560099AAB_IND560300AAA	121
2	IND562132AAA_IND560099AAB	112
3	IND560300AAA_IND562132AAA	108
4	IND421302AAG_IND400072AAB	105
...	...	...
2592	IND121004AAB_IND121002AAA	1
2593	IND721139AAC_IND721636AAB	1
2594	IND390020AAA_IND390018AAB	1
2595	IND721253AAC_IND721501AAC	1
2596	IND854335AAA_IND854326AAB	1

2597 rows × 2 columns

We have 2,597 unique corridor and the corresponding trip counts for each corridor. The corridor IND562132AAA\_IND560300AAA tops the list with 151 trips, followed by IND560099AAB\_IND560300AAA with 121 trips, and IND562132AAA\_IND560099AAB with 112 trips.

# Creating a corridor DataFrame for further analysis.

```
corridor_df= segment_df[['corridor_id', 'trip_uuid' , 'source_state', 'destination_state', 'actual_time', 'actual_distance_to_destination',
corridor_df.sample()
```

	corridor_id	trip_uuid	source_state	destination_state	actual_time	actual_distance_to_destination	osrm_time	osrm_dist
1301	IND600044AAD_IND600048AAA	153679738749092963	Tamil Nadu	Tamil Nadu	99.0	22.040851	12.0	15

# Average actual time taken by each corridor.

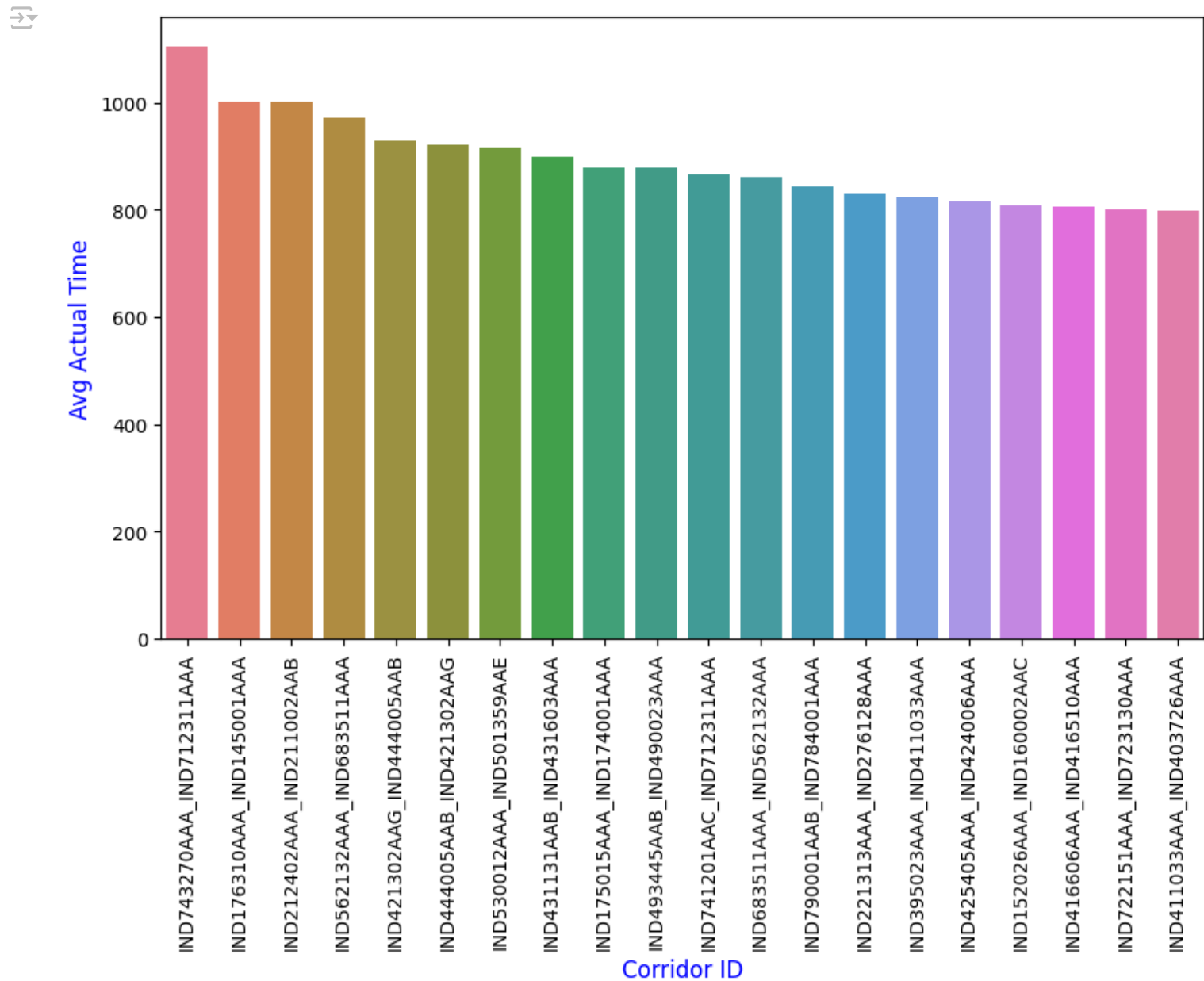
```
corr_avg_actual_time= corridor_df.groupby("corridor_id")["segment_actual_time_cumsum"].mean()
corr_avg_actual_time=pd.DataFrame(corr_avg_actual_time).reset_index()
corr_avg_actual_time.rename(columns={"segment_actual_time_cumsum":"corr_avg_actual_time"},inplace=True)
corr_avg_actual_time.sort_values(by="corr_avg_actual_time",ascending=False).head(5)
```

	corridor_id	corr_avg_actual_time
2270	IND743270AAA_IND712311AAA	1104.0
352	IND176310AAA_IND145001AAA	1002.5
447	IND212402AAA_IND211002AAB	1000.5
1645	IND562132AAA_IND683511AAA	971.5
1082	IND421302AAG_IND444005AAB	927.4

```
top_corr_avg_actual_time= corr_avg_actual_time.sort_values(by="corr_avg_actual_time",ascending=False).head(20)
```

# top 20 corrodor :Avg actual time taken

```
plt.figure(figsize=(10,6))
sns.barplot(x="corridor_id",y="corr_avg_actual_time",data=top_corr_avg_actual_time, palette="husl")
plt.xlabel("Corridor ID",color="blue",fontsize=12)
plt.ylabel("Avg Actual Time",color="blue",fontsize=12)
plt.xticks(rotation=90)
plt.show()
```



We can see that the corridor IND743270AAA\_IND712311AAA has the highest average time taken to deliver a package (1,104.0), followed by IND176310AAA\_IND145001AAA (1,002.5), IND212402AAA\_IND211002AAB (1,000.5), and IND562132AAA\_IND683511AAA (971.5).

# Avg OSRM Time

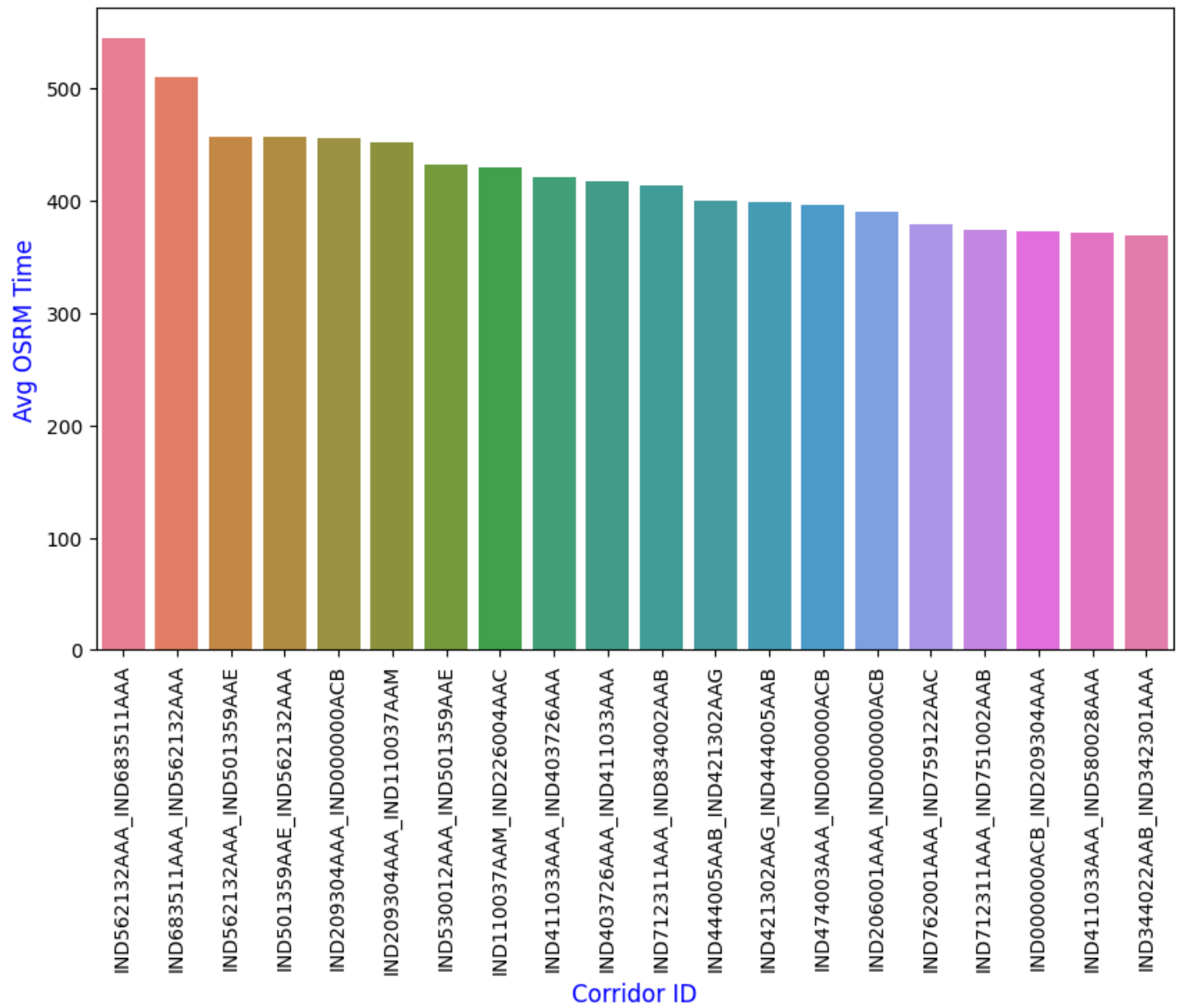
```
corr_avg_osrm_time= corridor_df.groupby("corridor_id")["segment_osrm_time_cumsum"].mean()
corr_avg_osrm_time=pd.DataFrame(corr_avg_osrm_time).reset_index()
corr_avg_osrm_time.rename(columns={"segment_osrm_time_cumsum":"corr_avg_osrm_time"},inplace=True)
corr_avg_osrm_time.sort_values(by="corr_avg_osrm_time",ascending=False).head(5)
```



	corridor_id	corr_avg_osrm_time
1645	IND562132AAA_IND683511AAA	544.000
2048	IND683511AAA_IND562132AAA	510.400
1622	IND562132AAA_IND501359AAE	456.875
1316	IND501359AAE_IND562132AAA	456.800
422	IND209304AAA_IND000000ACB	455.400

```
top_corr_avg_osrm_time= corr_avg_osrm_time.sort_values(by="corr_avg_osrm_time",ascending=False).head(20)
```

```
plt.figure(figsize=(10,6))
sns.barplot(x="corridor_id",y="corr_avg_osrm_time",data=top_corr_avg_osrm_time, palette="husl")
plt.xlabel("Corridor ID",color="blue",fontsize=12)
plt.ylabel("Avg OSRM Time",color="blue",fontsize=12)
plt.xticks(rotation=90)
plt.show()
```



The corridor IND562132AAA\_IND683511AAA tops the list with the highest average OSRM time (544), followed by IND683511AAA\_IND562132AAA (510), IND562132AAA\_IND501359AAE (457), and IND501359AAE\_IND562132AAA (457).

### Insights:

- \* FTL (Full Truck Load) accounts for 52.5% of trips, while carting makes up 47.4%, indicating a slight preference for FTL shipments.
- \* The actual delivery times and distances are always longer than what OSRM (Open Source Routing Machine) estimates. This difference suggests there might be problems with the routes being used, possibly because of traffic, road work, or less-than-ideal route choices.
- \* Maharashtra is the top state for sending and receiving packages, making it a key logistics hub. The high number of trips from Karnataka and Haryana shows significant logistics activity there. By reducing delivery times and choosing optimal routes, businesses can improve efficiency and enhance their operations.
- \* The corridor IND562132AAA\_IND560300AAA has 151 trips, indicating strong demand for that route. Understanding why this route is popular could help improve services for other routes and enhance overall business performance.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or [generate](#) with AI.