Importing required Libraries.....

In [1]:

```
import numpy as np
import pandas as pd
from scipy.stats import norm,poisson
import matplotlib.pyplot as plt
import seaborn as sns
```

Define Problem Statement and perform Exploratory Data Analysis (10 points)

Definition of problem (as per given problem statement with additional views)

Problem Statement:

```
    The largest and fastest growing fully integrated indian player by revenue, Delh
ivery in fiscal 2021, wants to build an operating system fro commerce. It will be u
sing a combination of  world-class infrastructure, logistics operations of the high
est quality, and cutting-edge engineering and technology capabilities.
    The Data team builds intelligence and capabilities using this data that helps t
hem to widen the gap between the quality, efficiency, and profitability of their bu
siness versus their competitors.
```

Company seeks help to understand and process the data coming out of data engineering pipelines.

1. Lets Clean, sanitize and manipulate data to get useful features out of raw fields
2. Help company to, make sense out of the raw data and help the data science team to build forecasting models on it. by applying Feature Engineering concepts on given dataset.

   We are looking forward to build an efficient forecasting model and check the factors which are most affecting and which least on the estimated delivery time of any particular orders.

   The company wants to understand and process data from its data engineering pipeline. Cleansing, sanitizing, and manipulating data to extract useful features from the raw data It helps data science teams understand the raw data and build predictive models on top of it.

The goal is to find significant differences between variables such as expected delivery time and actual delivery time. This is vital for your business because if the algorithms that predict time don't work properly, your customers will get the wrong quote, and if it's not delivered on time, they'll be dissatisfied and complain about your customer service. This can lead to increased work overload, decreased sales, and damaged company reputation.

Downloading Dataset...

In [2]:

```
!gdown "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delh
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv (https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv)
To: D:\Needa\My work\Userprof\delhivery_data.csv

  0%|          | 0.00/55.6M [00:00<?, ?B/s]
  1%|          | 524k/55.6M [00:00<00:15, 3.64MB/s]
  2%|1         | 1.05M/55.6M [00:00<00:34, 1.60MB/s]
  3%|2         | 1.57M/55.6M [00:00<00:29, 1.82MB/s]
  4%|3         | 2.10M/55.6M [00:01<00:30, 1.75MB/s]
  5%|4         | 2.62M/55.6M [00:01<00:28, 1.87MB/s]
  6%|5         | 3.15M/55.6M [00:01<00:30, 1.73MB/s]
  7%|6         | 3.67M/55.6M [00:02<00:28, 1.83MB/s]
  8%|7         | 4.19M/55.6M [00:02<00:31, 1.65MB/s]
  8%|8         | 4.72M/55.6M [00:02<00:35, 1.42MB/s]
  9%|9         | 5.24M/55.6M [00:03<00:34, 1.47MB/s]
 10%|#         | 5.77M/55.6M [00:03<00:30, 1.61MB/s]
 11%|#1        | 6.29M/55.6M [00:03<00:24, 2.00MB/s]
 12%|#2        | 6.82M/55.6M [00:03<00:24, 2.03MB/s]
 13%|#3        | 7.34M/55.6M [00:04<00:29, 1.62MB/s]
 14%|#4        | 7.86M/55.6M [00:04<00:36, 1.31MB/s]
 15%|#5        | 8.39M/55.6M [00:05<00:30, 1.56MB/s]
 16%|#6        | 8.91M/55.6M [00:05<00:26, 1.79MB/s]
 17%|#6        | 9.44M/55.6M [00:05<00:23, 1.99MB/s]
 18%|#7        | 9.96M/55.6M [00:05<00:21, 2.17MB/s]
 19%|#8        | 10.5M/55.6M [00:05<00:20, 2.18MB/s]
 20%|#9        | 11.0M/55.6M [00:06<00:21, 2.12MB/s]
 21%|##        | 11.5M/55.6M [00:06<00:19, 2.24MB/s]
 22%|##1       | 12.1M/55.6M [00:06<00:17, 2.46MB/s]
 23%|##2       | 12.6M/55.6M [00:06<00:17, 2.47MB/s]
 24%|##3       | 13.1M/55.6M [00:07<00:19, 2.23MB/s]
 25%|##4       | 13.6M/55.6M [00:07<00:18, 2.27MB/s]
 25%|##5       | 14.2M/55.6M [00:07<00:18, 2.23MB/s]
 26%|##6       | 14.7M/55.6M [00:07<00:19, 2.12MB/s]
 27%|##7       | 15.2M/55.6M [00:07<00:19, 2.11MB/s]
 28%|##8       | 15.7M/55.6M [00:08<00:21, 1.87MB/s]
 29%|##9       | 16.3M/55.6M [00:08<00:18, 2.16MB/s]
 30%|###       | 16.8M/55.6M [00:08<00:16, 2.34MB/s]
 31%|###1      | 17.3M/55.6M [00:08<00:15, 2.42MB/s]
 32%|###2      | 17.8M/55.6M [00:09<00:14, 2.57MB/s]
 33%|###2      | 18.4M/55.6M [00:09<00:16, 2.21MB/s]
 34%|###3      | 18.9M/55.6M [00:09<00:15, 2.31MB/s]
 35%|###4      | 19.4M/55.6M [00:09<00:14, 2.42MB/s]
 36%|###5      | 19.9M/55.6M [00:09<00:14, 2.47MB/s]
 37%|###6      | 20.4M/55.6M [00:10<00:13, 2.58MB/s]
 38%|###7      | 21.0M/55.6M [00:10<00:12, 2.86MB/s]
 39%|###8      | 21.5M/55.6M [00:10<00:11, 2.87MB/s]
 40%|###9      | 22.0M/55.6M [00:10<00:11, 2.82MB/s]
 41%|####      | 22.5M/55.6M [00:11<00:18, 1.75MB/s]
 42%|####2     | 23.6M/55.6M [00:11<00:13, 2.36MB/s]
 43%|####3     | 24.1M/55.6M [00:11<00:12, 2.45MB/s]
 44%|####4     | 24.6M/55.6M [00:11<00:13, 2.34MB/s]
 45%|####5     | 25.2M/55.6M [00:12<00:12, 2.42MB/s]
 46%|####6     | 25.7M/55.6M [00:12<00:10, 2.72MB/s]
 48%|####8     | 26.7M/55.6M [00:12<00:08, 3.53MB/s]
 49%|####9     | 27.3M/55.6M [00:12<00:08, 3.28MB/s]
 50%|####9     | 27.8M/55.6M [00:12<00:09, 2.92MB/s]
 51%|#####     | 28.3M/55.6M [00:13<00:09, 2.82MB/s]
 52%|#####1    | 28.8M/55.6M [00:13<00:08, 3.15MB/s]
 53%|#####2    | 29.4M/55.6M [00:13<00:07, 3.42MB/s]
```

```
 54%|#####3    | 29.9M/55.6M [00:13<00:07, 3.59MB/s]
 56%|#####5    | 30.9M/55.6M [00:13<00:05, 4.15MB/s]
 58%|#####7    | 32.0M/55.6M [00:13<00:05, 4.29MB/s]
 59%|#####9    | 33.0M/55.6M [00:14<00:05, 4.04MB/s]
 60%|######    | 33.6M/55.6M [00:14<00:06, 3.41MB/s]
 61%|######1   | 34.1M/55.6M [00:14<00:06, 3.31MB/s]
 63%|######3   | 35.1M/55.6M [00:14<00:05, 3.43MB/s]
 64%|######4   | 35.7M/55.6M [00:15<00:06, 2.87MB/s]
 65%|######5   | 36.2M/55.6M [00:15<00:07, 2.66MB/s]
 66%|######5   | 36.7M/55.6M [00:15<00:06, 3.03MB/s]
 67%|######6   | 37.2M/55.6M [00:15<00:06, 2.71MB/s]
 68%|######7   | 37.7M/55.6M [00:16<00:07, 2.50MB/s]
 69%|######8   | 38.3M/55.6M [00:16<00:06, 2.88MB/s]
 70%|######9   | 38.8M/55.6M [00:16<00:05, 3.03MB/s]
 71%|#######   | 39.3M/55.6M [00:16<00:06, 2.69MB/s]
 72%|#######1  | 39.8M/55.6M [00:16<00:05, 2.95MB/s]
 73%|#######2  | 40.4M/55.6M [00:16<00:06, 2.47MB/s]
 74%|#######3  | 40.9M/55.6M [00:17<00:05, 2.51MB/s]
 74%|#######4  | 41.4M/55.6M [00:17<00:05, 2.82MB/s]
 75%|#######5  | 41.9M/55.6M [00:17<00:04, 3.06MB/s]
 76%|#######6  | 42.5M/55.6M [00:17<00:04, 3.09MB/s]
 77%|#######7  | 43.0M/55.6M [00:17<00:04, 2.99MB/s]
 78%|#######8  | 43.5M/55.6M [00:17<00:03, 3.21MB/s]
 80%|########  | 44.6M/55.6M [00:18<00:02, 3.92MB/s]
 81%|########1 | 45.1M/55.6M [00:18<00:02, 4.03MB/s]
 82%|########2 | 45.6M/55.6M [00:18<00:03, 2.83MB/s]
 83%|########2 | 46.1M/55.6M [00:18<00:03, 3.09MB/s]
 84%|########3 | 46.7M/55.6M [00:18<00:03, 2.97MB/s]
 85%|########4 | 47.2M/55.6M [00:19<00:03, 2.68MB/s]
 87%|########6 | 48.2M/55.6M [00:19<00:02, 3.17MB/s]
 88%|########7 | 48.8M/55.6M [00:19<00:01, 3.47MB/s]
 89%|########8 | 49.3M/55.6M [00:19<00:01, 3.70MB/s]
 90%|######### | 50.3M/55.6M [00:19<00:01, 4.12MB/s]
 91%|#########1| 50.9M/55.6M [00:19<00:01, 4.22MB/s]
 92%|#########2| 51.4M/55.6M [00:20<00:01, 3.20MB/s]
 93%|#########3| 51.9M/55.6M [00:20<00:01, 3.38MB/s]
 94%|#########4| 52.4M/55.6M [00:20<00:00, 3.26MB/s]
 95%|#########5| 53.0M/55.6M [00:20<00:00, 3.39MB/s]
 96%|#########6| 53.5M/55.6M [00:20<00:00, 3.73MB/s]
 97%|#########7| 54.0M/55.6M [00:20<00:00, 3.73MB/s]
 98%|#########8| 54.5M/55.6M [00:21<00:00, 3.54MB/s]
 99%|#########8| 55.1M/55.6M [00:21<00:00, 3.50MB/s]
100%|#########9| 55.6M/55.6M [00:21<00:00, 3.63MB/s]
100%|##########| 55.6M/55.6M [00:21<00:00, 2.60MB/s]
```

In [3]:

```
df = pd.read_csv(r"D:\Needa\My work\Userprof\delhivery_data.csv")
df
```

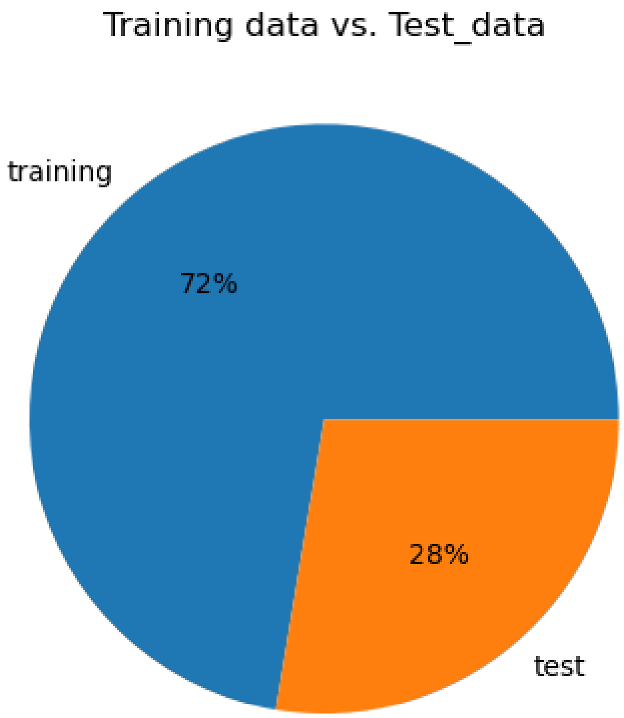
Out[3]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
...	...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1

144867 rows × 24 columns

In [4]:

```
data = df["data"].value_counts(normalize = True)*100
plt.title("Training data vs. Test_data")
plt.pie(x = data.values, labels=data.index, autopct='%.0f%%')
plt.show()
```



#From Above Piechart we can infer that out of total data given it is divided into mainly 2 groups namely

72% data ---- Training data 28% data ---- Test data We have trained this model with 104858 datapoints and then tested it with 40009 data points

Steps used

1. Check shape and info for null values
2. Column Profiling(drop not required columns as unknown fields from the dataset)
3. Extracting Features (like year,month, etc.)
4. compress Data (each trip should be denoted by different datapoints)
5. creating new features
6. Missing values and outliers detection
7. Hypothesis Testings

summary of all hypothesis testings

8. Observations from correlation result and hypothesis testing result combined
9. Visual analysis
10. Column Standardization
11. Insights Summarize Insights
12. Recommendations

(1) Check shape and info for null values and duplicates.

In [5]:

```
df.shape
```

Out[5]:

```
(144867, 24)
```

In [6]:

```
df.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
```

We see there are total 144867 rows and 24 columns.

If observed carefully we find mutiple entries for the same trip_uuid with differnt source and destination centers. Like it is inferred that any order is not delivered to its final destination in a single route but it has different stops all over the route before reaching the final destination

we take a look upon columns:

checking any duplicate rows.

In [7]:

```
df.duplicated().sum()
```

Out[7]:

```
0
```

checking missing values

In [8]:

```
#percentage data missing
for col in df.columns:
    p = df[col].isnull().sum()/len(df[col])*100
    #dp = pd.to_DataFrame(p)
    print(col, ": ",round(p,3),"%")
```

```
data :  0.0 %
trip_creation_time :  0.0 %
route_schedule_uuid :  0.0 %
route_type :  0.0 %
trip_uuid :  0.0 %
source_center :  0.0 %
source_name :  0.202 %
destination_center :  0.0 %
destination_name :  0.18 %
od_start_time :  0.0 %
od_end_time :  0.0 %
start_scan_to_end_scan :  0.0 %
is_cutoff :  0.0 %
cutoff_factor :  0.0 %
cutoff_timestamp :  0.0 %
actual_distance_to_destination :  0.0 %
actual_time :  0.0 %
osrm_time :  0.0 %
osrm_distance :  0.0 %
factor :  0.0 %
segment_actual_time :  0.0 %
segment_osrm_time :  0.0 %
segment_osrm_distance :  0.0 %
segment_factor :  0.0 %
```

only source name and destination name are missing 0.202% and 0.18% data respectively. As compared to dataset size around 144867 rows this missing data seems quite negligible. So we will drop these missing_values rows.

In [9]:

```
df.dropna(inplace = True)
```

In [10]:

```
df
```

Out[10]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
...	...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1

144316 rows × 24 columns

In [ ]:

```

```

2. Column Profiling:

1. data - tells whether the data is testing or training data
2. trip_creation_time – Timestamp of trip creation

3. route_schedule_uuid – Unique Id for a particular route schedule
4. route_type – Transportation type FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way Carting: Handling system consisting of small vehicles (carts)
5. trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
6. source_center - Source ID of trip origin
7. source_name - Source Name of trip origin
8. destination_cente – Destination ID
9. destination_name – Destination Name
10. od_start_time – Trip start time
11. od_end_time – Trip end time
12. start_scan_to_end_scan – Time taken to deliver from source to destination
13. is_cutoff – Unknown field
14. cutoff_factor – Unknown field
15. cutoff_timestamp – Unknown field
16. actual_distance_to_destination – Distance in Kms between source and destination warehouse
17. actual_time – Actual time taken to complete the delivery (Cumulative)
18. osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
19. osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
20. factor – Unknown field
21. segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
22. segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
23. segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
24. segment_factor – Unknown field

As it is seen '2. trip_creation_time', '10. od_start_time', '11. od_end_time', '15. cutoff_timestamp' these columns contains timestamp so converting to datatype datetime.

Converting Timestamp columns to date_time

In [11]:

```
date_cols = ['trip_creation_time', 'od_start_time', 'od_end_time']
for i in date_cols:
    df[i] = pd.to_datetime(df[i])
#df.head()
```

There are 5 unknown fields. such as cutoff_factor, cutoff_timestamp, is_cutoff, factor, segment_factor

(drop not required columns as unknown fields from the dataset)

attributes mentioned as unknown fields.

In [12]:

```
df.drop(columns=["is_cutoff","cutoff_factor","cutoff_timestamp","factor","segment_factor"],
```

(3) Extracting Features

Extracting year, month and day_name

In [13]:

```
df["year"] = df["trip_creation_time"].dt.year
df["year"].value_counts()
```

Out[13]:

```
2018    144316
Name: year, dtype: int64
```

Complete data is from year 2018

In [14]:

```
df["month"] = df["trip_creation_time"].dt.month_name()
df["month"].value_counts()
```

Out[14]:

```
September    126932
October       17384
Name: month, dtype: int64
```

Complete data is from month september and october

In [15]:

```
df["day_name"] = df["trip_creation_time"].dt.day_name()
df["day_name"].value_counts()
```

Out[15]:

```
Wednesday    26634
Thursday     20422
Friday       20177
Saturday     19874
Tuesday      19858
Monday       19540
Sunday       17811
Name: day_name, dtype: int64
```

In [16]:

```
df["time24H"] = df["trip_creation_time"].dt.hour
df["time24H"].value_counts()
```

Out[16]:

```
22    12235
20    10286
19    10175
23     9325
1      8755
21     8709
0      8247
18     7768
2      7321
4      6629
5      6152
17     5976
3      4972
6      4396
15     4274
13     4271
14     4269
16     3858
8      3512
10     2880
7      2704
11     2690
9      2466
12     2446
Name: time24H, dtype: int64
```

In [17]:

```
df["source_city"] = df["source_name"].apply(lambda x: x.split(" ")[0].split("_")[0])
```

In [18]:

```
df["source_state"] = df["source_name"].apply(lambda x: x.split(" ")[-1][1:-1])
```

In [19]:

```
df["source_city_code"] = df["source_name"].apply(lambda x: x.split(" ")[0].split("_")[-1])
```

In [20]:

```
df["destination_city"] = df["destination_name"].apply(lambda x: x.split("_")[0])
```

In [21]:

```
df["destination_state"] = df["destination_name"].apply(lambda x: x.split(" ")[-1][1:-1])
```

In [22]:

```
df["destination_city_code"] = df["destination_name"].apply(lambda x: x.split(" ")[0].split(
```

In [23]:

```
df['Corridor'] = df['source_city']+ " To " + df['destination_city']
```

In [24]:

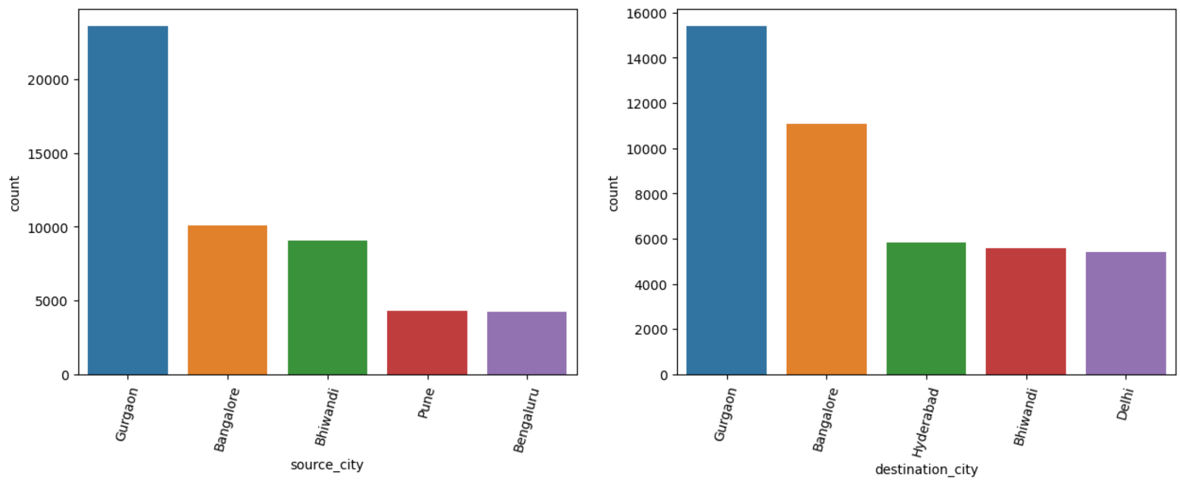
```
df['Corridor'].value_counts()
```

Out[24]:

```
Gurgaon To Bangalore                      4976
Bangalore To Gurgaon                      3316
Gurgaon To Kolkata                        2862
Bengaluru To Bengaluru                    2062
Bangalore To Bengaluru                    1741
                                          ...
Shahada To Dhule                             1
Krishnanagar To Hanskhali                    1
Hajipur To Dighwara                          1
Mandapeta To Rajamundry                      1
Vizag To Vishakhapatnam (Andhra Pradesh)     1
Name: Corridor, Length: 2336, dtype: int64
```

In [25]:

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.countplot(data= df, x='source_city', order=df['source_city'].value_counts().nlargest(5)
plt.xticks(rotation = 75)

plt.subplot(1,2,2)
sns.countplot(data= df, x='destination_city', order=df['destination_city'].value_counts().n
plt.xticks(rotation = 75)
plt.show()
```



4. compress Data (each trip should be denoted by different datapoints)

In [26]:

```
df.loc[df['trip_uuid'] == 'trip-153741093647649320']
```

Out[26]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862C
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862C
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862C
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862C
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862C

10 rows × 30 columns

#we see 10 rows are there only for 1 trip So need to cpompless the data rowwise.

In [27]:

```
df.loc[df['trip_uuid']=='trip-153741093647649320', ['trip_uuid', 'source_name', 'destinatio
```

Out[27]:

	trip_uuid	source_name	destination_name	segment_actual_time	osr
0	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)	14.0	
1	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)	10.0	
2	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)	16.0	
3	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)	21.0	
4	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)	6.0	
5	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	Anand_Vaghasi_IP (Gujarat)	15.0	
6	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	Anand_Vaghasi_IP (Gujarat)	28.0	
7	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	Anand_Vaghasi_IP (Gujarat)	21.0	
8	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	Anand_Vaghasi_IP (Gujarat)	10.0	
9	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	Anand_Vaghasi_IP (Gujarat)	26.0	

In [28]:

```
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' : 'last',
    'destination_name' : 'last',

    'od_start_time' : 'first',
    'od_end_time' : 'first',
    'start_scan_to_end_scan' : 'first',

    'actual_distance_to_destination' : 'last',
    'actual_time' : 'last',

    'osrm_time' : 'last',
    'osrm_distance' : 'last',

    'segment_actual_time_sum' : 'last',
    'segment_osrm_distance_sum' : 'last',
    'segment_osrm_time_sum' : 'last',

}
```

In [29]:

```
df['segment_id'] = df['trip_uuid'] + df['source_center'] + df['destination_center']

segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time']

for col in segment_cols:
    df[col + '_sum'] = df.groupby('segment_id')[col].cumsum()

df[[col + '_sum' for col in segment_cols]]
```

Out[29]:

	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_sum
0	14.0	11.9653	11.0
1	24.0	21.7243	20.0
2	40.0	32.5395	27.0
3	61.0	45.5619	39.0
4	67.0	49.4772	44.0
...	...	...	...
144862	92.0	65.3487	94.0
144863	118.0	82.7212	115.0
144864	138.0	103.4265	149.0
144865	155.0	122.3150	176.0
144866	423.0	131.1238	185.0

144316 rows × 3 columns

In [30]:

```
segment = df.groupby('segment_id').agg(segment_dict).reset_index()
segment = segment.sort_values(by=['segment_id','od_end_time'], ascending=True).reset_index(
```

In [31]:

```
# trip duration in minutes
segment['od_trip_duration'] = (segment['od_end_time'] - segment['od_start_time']).dt.total_
```

In [32]:

```
trip_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' : 'last',
    'destination_name' : 'last',

    'start_scan_to_end_scan' : 'sum',
    'od_trip_duration' : 'sum',

    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',

    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum',

}
```

In [33]:

```
trip = segment.groupby('trip_uuid').agg(trip_dict).reset_index(drop = True)
```

In [34]:

```
trip
```

Out[34]:

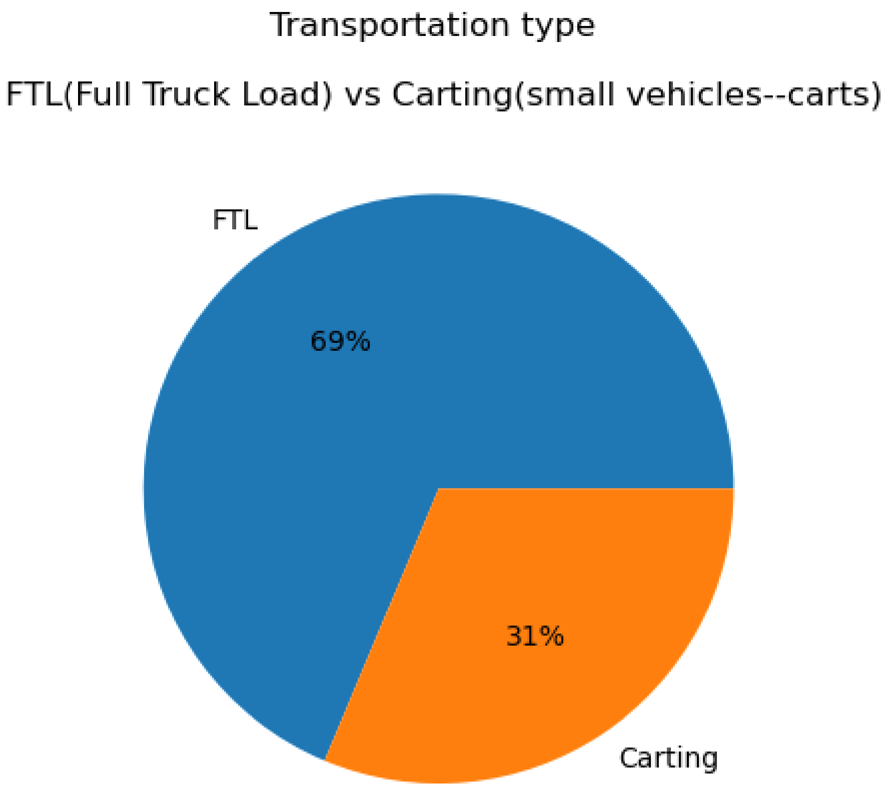
	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	soui
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND2
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND5
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND0
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND4
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND5
...	...	...	...	...	...	
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND1
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND1
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND2
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND6
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND5

14787 rows × 18 columns

Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the categorical variables)

In [35]:

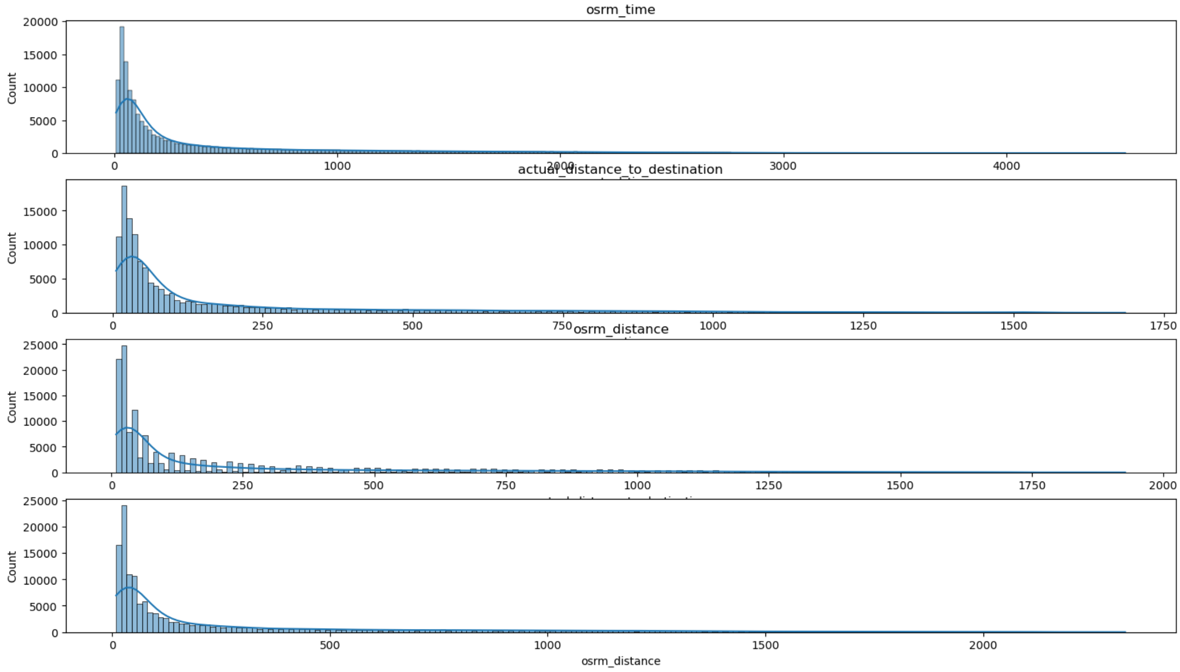
```
data = df["route_type"].value_counts(normalize = True)*100
plt.title(f"Transportation type \n\n FTL(Full Truck Load) vs Carting(small vehicles--carts)
plt.pie(x = data.values, labels=data.index, autopct='%.0f%%')
plt.show()
```



Maximum time route is used with FTL i.e. full truck load. which is 69% and carting is used for 31%

In [36]:

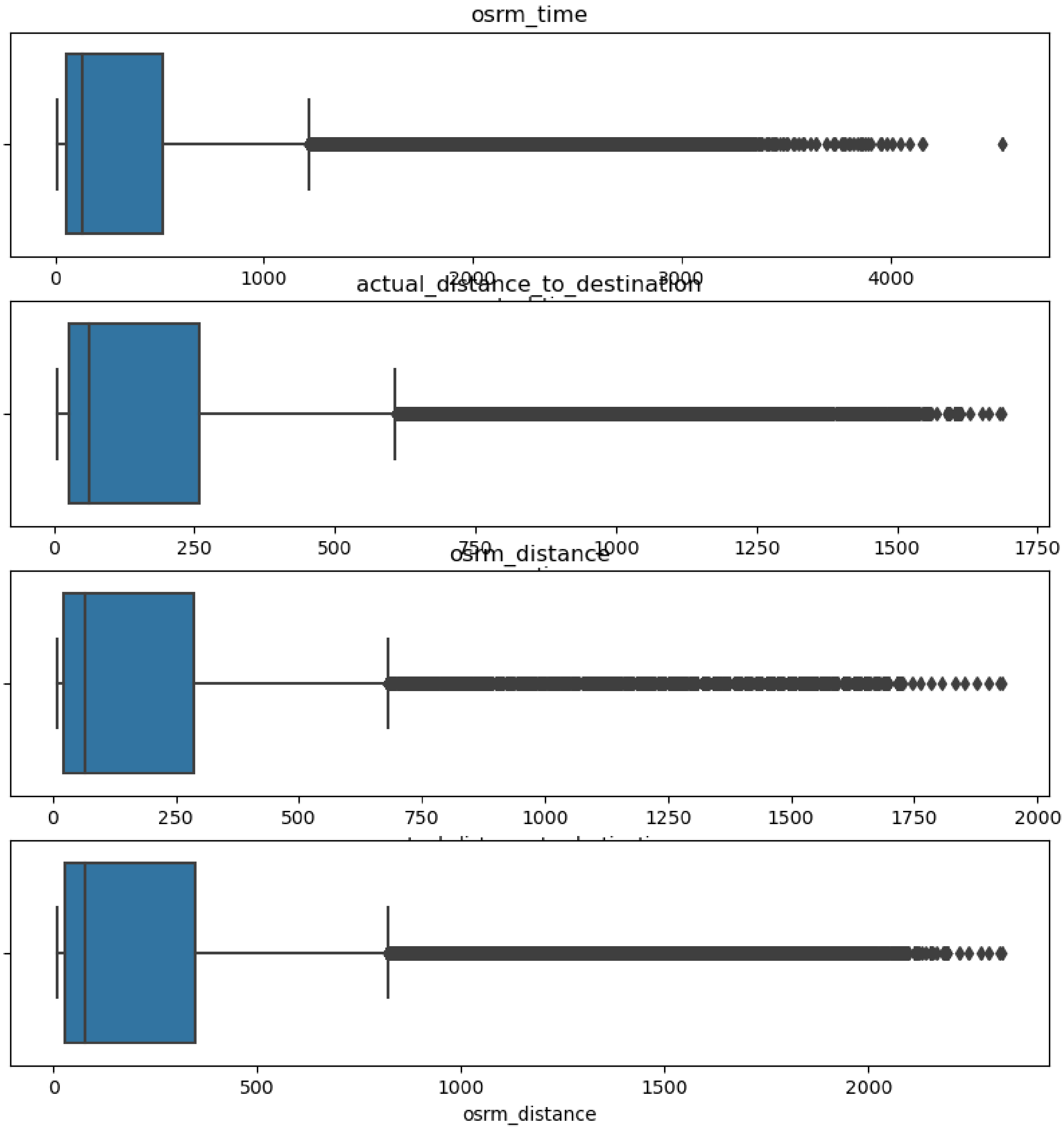
```
plt.figure(figsize = [18,10])
num_cols = ['actual_time','osrm_time','actual_distance_to_destination','osrm_distance']
for i in range (len(num_cols)):
    plt.title(num_cols[i])
    plt.subplot(len(num_cols),1, i+1)
    sns.histplot(data=df, x=num_cols[i], kde=True)
```



If we see all the above continuous variables are right skewed so will contain many outliers so we treat it for the same.

In [37]:

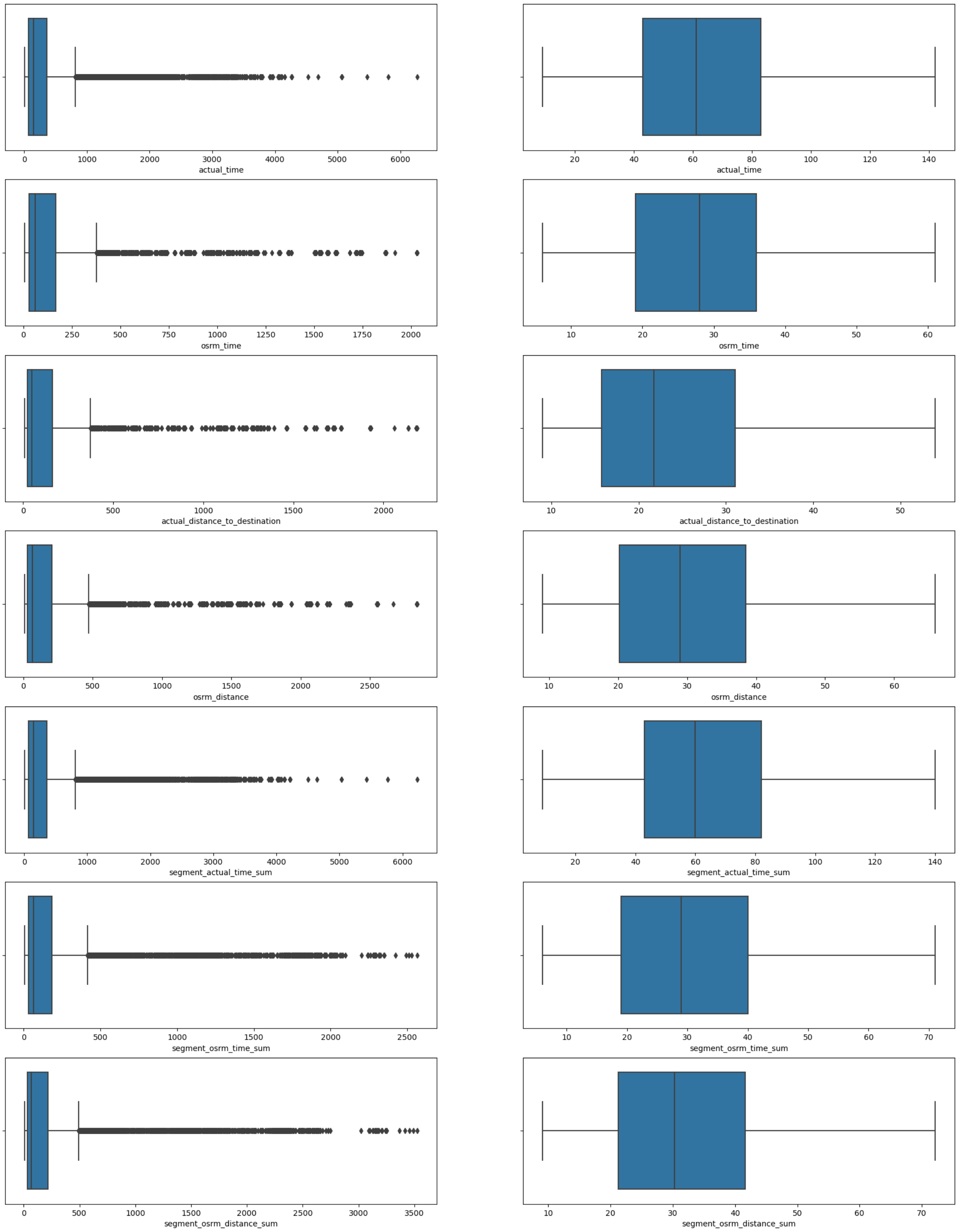
```
plt.figure(figsize = [10,10])
for i in range (len(num_cols)):
    plt.title(num_cols[i])
    plt.subplot(len(num_cols),1, i+1)
    sns.boxplot(data=df, x=num_cols[i])
```



Outlier treatment

In [38]:

```
trip_o = trip.copy()

def find_outliers_IQR(col):
  q1=col.quantile(0.25)
  q3=col.quantile(0.75)
  IQR=q3-q1
  outliers = trip[((col<(q1-1.5*IQR)) | (col>(q3+1.5*IQR)))]
  return outliers

cols = ['actual_time', 'osrm_time','actual_distance_to_destination','osrm_distance','segmen

n=1
while n!=0:
  n=0
  for x in cols:
    outliers = find_outliers_IQR(trip[x]).index
    trip.drop(outliers,inplace=True)
    n+=len(outliers)


fig, axis = plt.subplots(nrows=len(cols), ncols=2, figsize=(22, len(cols)*4))
for i in range (len(cols)):
  for j in ([0,1]):
    if j==0:
      sns.boxplot(data=trip_o, x=cols[i], ax=axis[i, j])
    else:
      sns.boxplot(data=trip, x=cols[i], ax=axis[i, j])
# Left side plots: boxplot distribution before removing outliers
# Right side plots: boxplot distribution after removing outliers
```


actual_time
actual_time
osrm_time
osrm_time
actual_distance_to_destination
actual_distance_to_destination
osrm_distance
osrm_distance
segment_actual_time_sum
segment_actual_time_sum
segment_osrm_time_sum
segment_osrm_time_sum
segment_osrm_distance_sum
segment_osrm_distance_sum

In [ ]:

In [39]:

```
trip_o.describe() # before outlier treatment there the mean values are very far from median
```

Out[39]:

	start_scan_to_end_scan	od_trip_duration	actual_distance_to_destination	actual_time	
count	14787.000000	14787.000000	14787.000000	14787.000000	14
mean	529.429025	530.313517	164.090196	356.306012	
std	658.254936	658.415490	305.502982	561.517936	
min	23.000000	23.461468	9.002461	9.000000	
25%	149.000000	149.698496	22.777099	67.000000	
50%	279.000000	279.710750	48.287894	148.000000	
75%	632.000000	633.537697	163.591258	367.000000	
max	7898.000000	7898.551955	2186.531787	6265.000000	2

for the fielsd above mentioned there is a drastic diference in max and median values and so outlier removerl will have an significant effect

In [40]:

```
trip.describe()   # after outlier treatment there the mean values is closer to median.
```

Out[40]:

	start_scan_to_end_scan	od_trip_duration	actual_distance_to_destination	actual_time	o
count	6341.000000	6341.000000	6341.000000	6341.000000	634
mean	164.841350	165.393429	23.648369	64.844662	2
std	119.457103	119.482331	10.047152	28.688862	1
min	23.000000	23.461468	9.002461	9.000000	
25%	96.000000	96.213386	15.729635	43.000000	1
50%	138.000000	138.886469	21.733245	61.000000	2
75%	198.000000	198.779605	31.035562	83.000000	3
max	2701.000000	2701.464958	53.932891	142.000000	6

In [41]:

```
len(trip),len(trip_o)
```

Out[41]:

```
(6341, 14787)
```

Hypothesis testing Comparisons

Checking relationship between aggregated fields (10 Points)

In [42]:

```
#Relationship between ACTUAL TIME and OSRM_TIME
```

In [43]:

```
from scipy.stats import ttest_ind, shapiro, kruskal
```

In [44]:

```
def Check_hypothesis(p_value):
    # Level of significance
    alpha = 0.05
    # conclusion
    if p_value < alpha:
        print('Reject Null Hypothesis (Significant difference between two samples)')
        return 1
    else:
        print('Fail to Reject Null Hypothesis (No significant difference between two sample
        return 0
```

In [45]:

```
np.mean(trip['actual_time'])
```

Out[45]:

```
64.84466172527992
```

In [46]:

```
np.var(trip['actual_time'])
```

Out[46]:

```
822.9209890868008
```

In [47]:

```
np.mean(trip['osrm_time'])
```

Out[47]:

```
29.056300268096514
```

In [48]:

```
np.var(trip['osrm_time'])
```

Out[48]:

```
164.6410504343605
```

In [49]:

```
# hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time agg

# Checking: Does actual_time is similar as osrm_time?

null_hypothesis = 'Mean of actual_time and osrm_time is equal'
alternative_hypothesis = 'Mean of actual_time is higher than mean of osrm_time'

sample1 = trip['actual_time']
sample2 = trip['osrm_time']
t_stat, p_value = ttest_ind(sample1, sample2, equal_var=False, alternative='greater')

print(t_stat, p_value)

output = Check_hypothesis(p_value)

if output == 1:
    print(alternative_hypothesis)
else:
    print(null_hypothesis)
```

```
90.67848382936118 0.0
Reject Null Hypothesis (Significant difference between two samples)
Mean of actual_time is higher than mean of osrm_time
```

In [50]:

```
#Relationship between ACTUAL TIME and OSRM TIME
```

In [51]:

```
np.mean(trip['actual_time'])
```

Out[51]:

```
64.84466172527992
```

In [52]:

```
np.var(trip['actual_time'])
```

Out[52]:

```
822.9209890868008
```

In [53]:

```
np.mean(trip['segment_actual_time_sum'])
```

Out[53]:

```
64.10266519476423
```

In [54]:

```
np.var(trip['segment_actual_time_sum'])
```

Out[54]:

```
808.4923773786879
```

In [55]:

```
# hypothesis testing/ visual analysis between actual_time aggregated value and segment_actua

# Checking: Does segment_actual_time is similar as segment_actual_time?

null_hypothesis = 'Mean of actual_time and segment_actual_time is equal'
alternative_hypothesis = 'Mean of actual_time is higher than mean of segment_actual_time_sur

sample1 = trip['actual_time']
sample2 = trip['segment_actual_time_sum']
t_stat, p_value = ttest_ind(sample1, sample2, equal_var=True)

print(t_stat, p_value)

output = Check_hypothesis(p_value)

if output == 1:
    print(alternative_hypothesis)
else:
    print(null_hypothesis)
```

```
1.4627311003801773 0.14356575741333472
Fail to Reject Null Hypothesis (No significant difference between two sample
s)
Mean of actual_time and segment_actual_time is equal
```

In [56]:

```
#Relationship between OSRM_DISTANCE and SEGMENT_OSRM_DISTANCE_SUM
```

In [57]:

```
np.mean(trip['osrm_distance'])
```

Out[57]:

```
30.291222583188894
```

In [58]:

```
np.var(trip['osrm_distance'])
```

Out[58]:

```
149.75643916903172
```

In [59]:

```
np.mean(trip['segment_osrm_distance_sum'])
```

Out[59]:

```
32.37308074436219
```

In [60]:

```
np.var(trip['segment_osrm_distance_sum'])
```

Out[60]:

```
189.319978181173
```

In [61]:

```
# Checking: Does osrm_distance is similar as segment_osrm_distance_sum?

from scipy.stats import ttest_ind
null_hypothesis = 'Mean of osrm_distance is similar as mean of segment_osrm_distance_sum'
alternative_hypothesis = 'Mean of osrm_distance is less than mean of segment_osrm_distance_s

sample1 = trip['osrm_distance']
sample2 = trip['segment_osrm_distance_sum']
t_stat, p_value = ttest_ind(sample1, sample2, equal_var=False, alternative='less')

print(t_stat, p_value)


output = Check_hypothesis(p_value)

if output == 1:
    print(alternative_hypothesis)
else:
    print(null_hypothesis)

    # conclusion: mean of osrm_distance is similar as mean of segment_osrm_distance_sum
```

```
-9.002165094005438 1.2650801157164575e-19
Reject Null Hypothesis (Significant difference between two samples)
Mean of osrm_distance is less than mean of segment_osrm_distance_sum
```

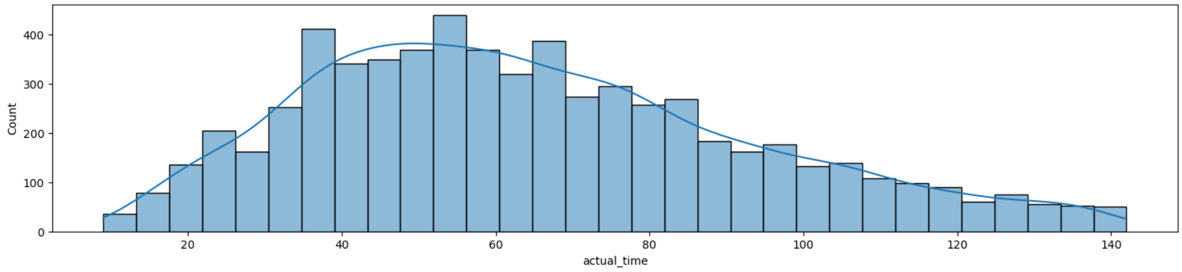
In [62]:

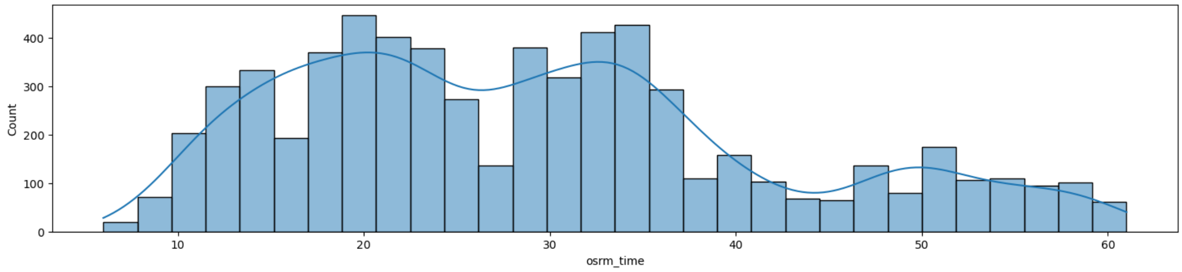
```
num_cols = ['actual_time','osrm_time','segment_actual_time_sum','segment_osrm_time_sum','ac
for i in (num_cols):
    sample1 = trip[i].sample(5000)
    stat, p_value = shapiro(sample1)
    if(p_value < 0.05):
        print(i, ": sample is not normally distributed, do NON PARAMETRIC (KRUSKAL) test")
    else:
        print(i, ": sample is normally distributed, can do PARAMETRIC (ANNOVA) test")
```

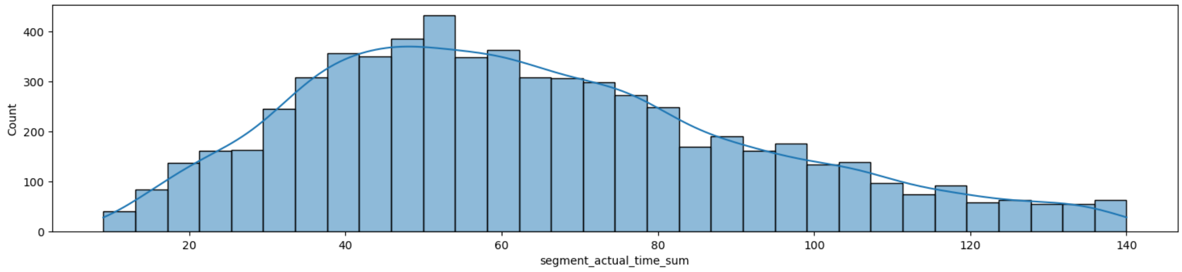
```
actual_time : sample is not normally distributed, do NON PARAMETRIC (KRUSKAL)
test
osrm_time : sample is not normally distributed, do NON PARAMETRIC (KRUSKAL) t
est
segment_actual_time_sum : sample is not normally distributed, do NON PARAMETR
IC (KRUSKAL) test
segment_osrm_time_sum : sample is not normally distributed, do NON PARAMETRIC
(KRUSKAL) test
actual_distance_to_destination : sample is not normally distributed, do NON P
ARAMETRIC (KRUSKAL) test
osrm_distance : sample is not normally distributed, do NON PARAMETRIC (KRUSKA
L) test
segment_osrm_distance_sum : sample is not normally distributed, do NON PARAME
TRIC (KRUSKAL) test
```

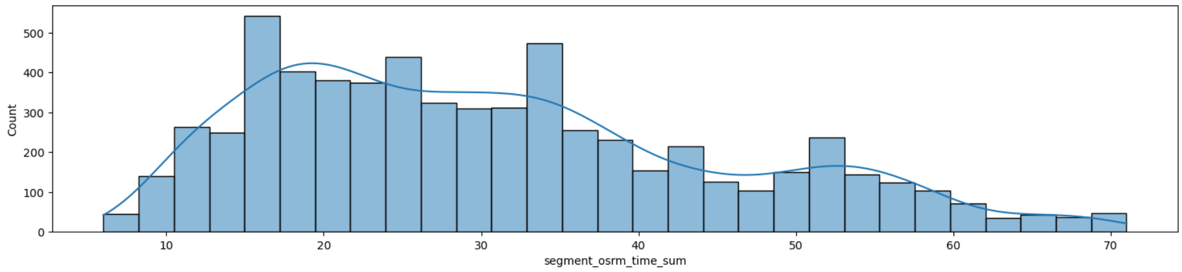
In [63]:

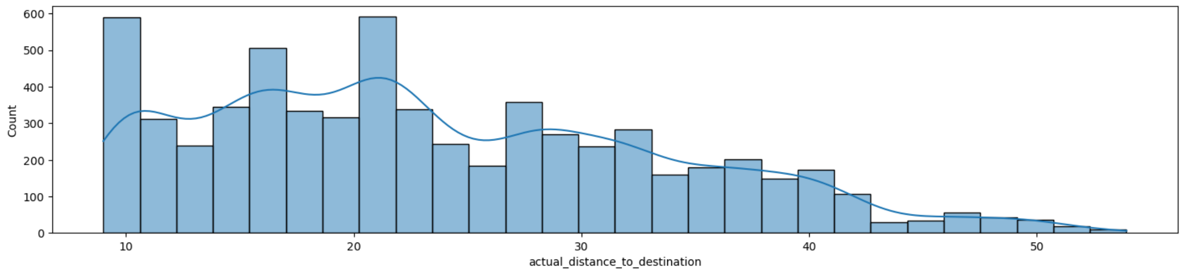
```
plt.figure(figsize = [18,30])
num_cols = ['actual_time','osrm_time','segment_actual_time_sum','segment_osrm_time_sum','ac
for i in range (len(num_cols)):
  plt.subplot(len(num_cols),1, i+1)
  sns.histplot(data=trip, x=num_cols[i], kde=True)
# Distribution are not normally distributed, we will do non parametric tests
```

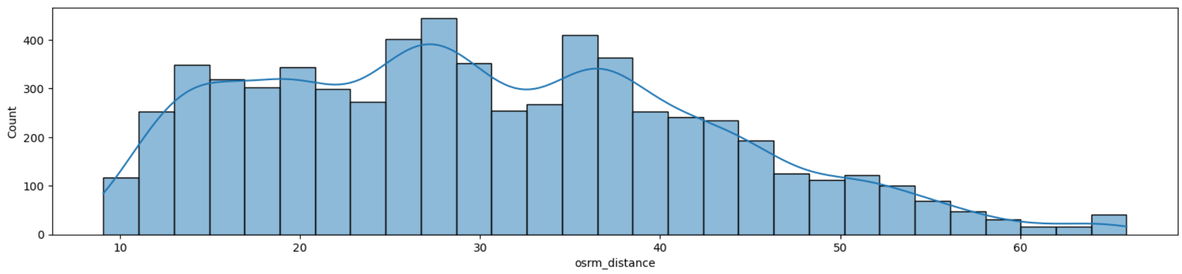

Count
actual_time

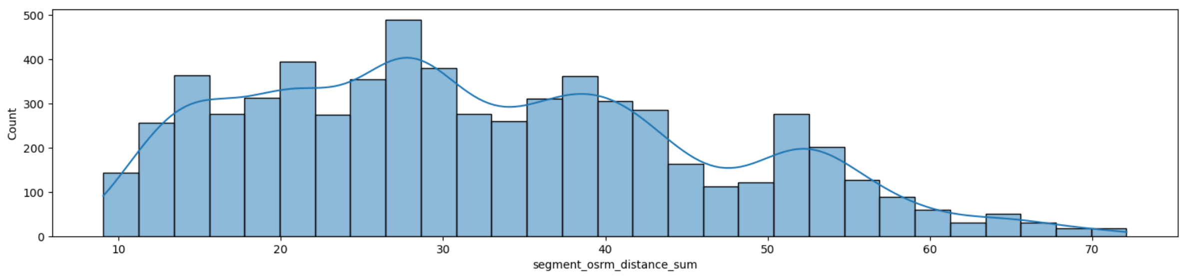

Count
osrm_time


Count
segment_actual_time_sum


Count
segment_osrm_time_sum

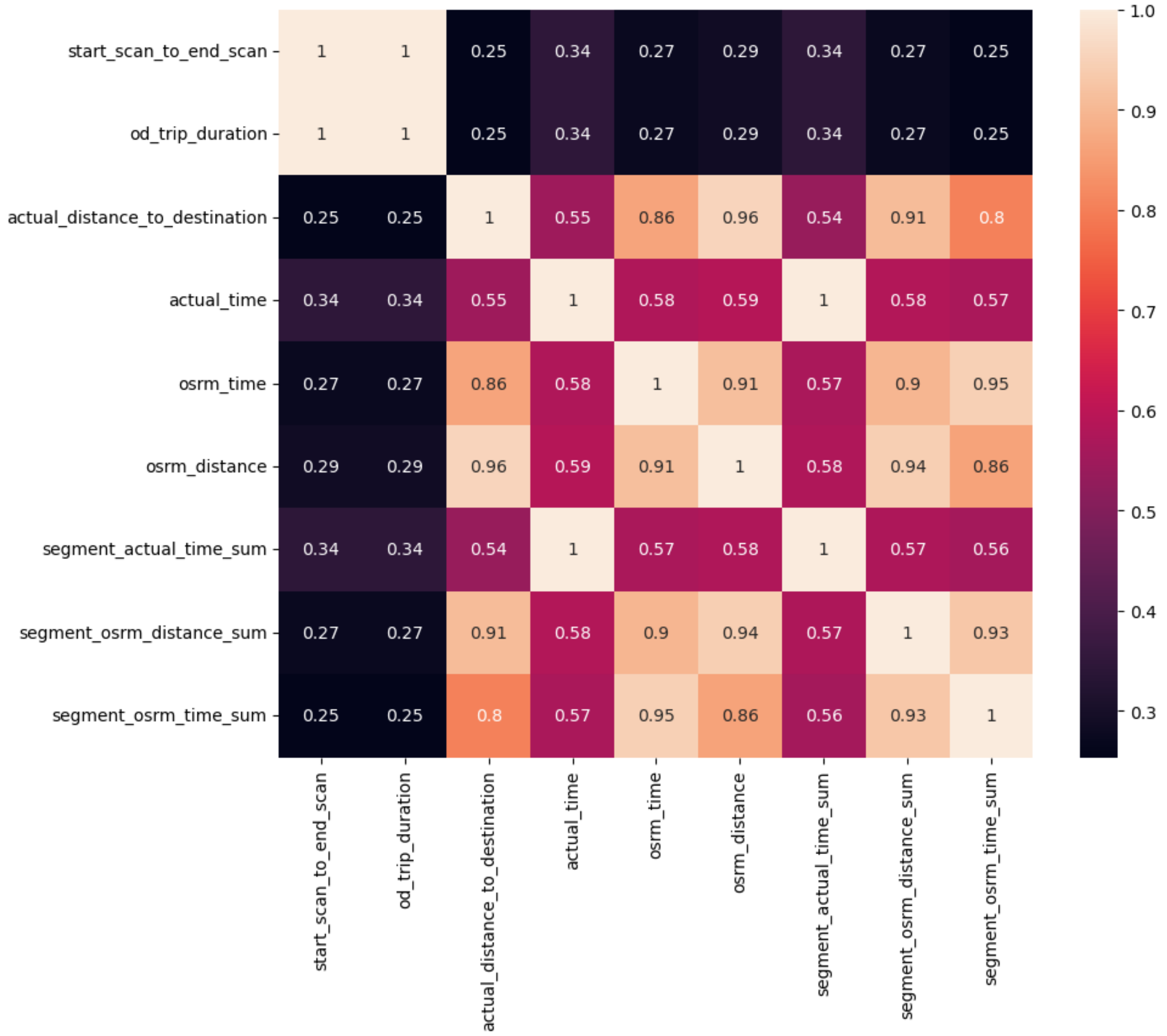

Count
actual_distance_to_destination


Count
osrm_distance


Count
segment_osrm_distance_sum

In [64]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(trip.corr(), annot=True);

# the numerical data are highly correlated with each other except time variables
# Hence, there will be a lot of problem with the multicollinearity which need to be taken co
```



In [65]:

```
# H0: mean of both samples are similar
# Ha: means of both samples are different

sample1 = trip['actual_time']
sample2 = trip['osrm_time']
# perform kruskal test
stat, p_value = kruskal(sample1, sample2)
print('Statistics=%.2f, p=%.2f' % (stat, p_value))

output = Check_hypothesis(p_value)
```

```
Statistics=5514.80, p=0.00
Reject Null Hypothesis (Significant difference between two samples)
```

In [66]:

```
# H0: mean of both samples are similar
# Ha: means of both samples are different

sample1 = trip['osrm_distance']
sample2 = trip['segment_osrm_distance_sum']

stat, p_value = kruskal(sample1, sample2)
print('Statistics=%.2f, p=%.2f'% (stat, p_value))

output = Check_hypothesis(p_value)
```

```
Statistics=54.66, p=0.00
Reject Null Hypothesis (Significant difference between two samples)
```

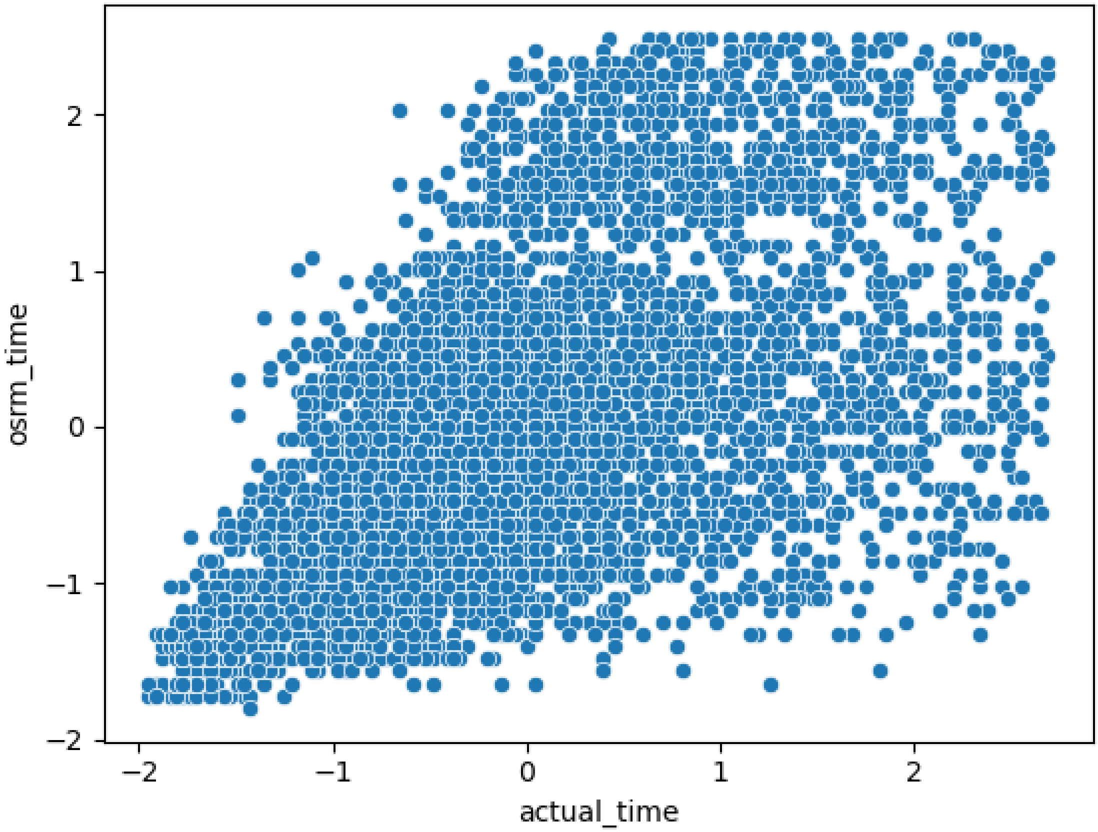
Standardization, Normalization

In [67]:

```
df_ao = trip[["actual_time", "osrm_time"]]


from sklearn.preprocessing import StandardScaler, MinMaxScaler
df_ao_ss = StandardScaler().fit_transform(df_ao) # ss--> standard scaler z-score


df_ao_ss = pd.DataFrame(df_ao_ss, columns=["actual_time", "osrm_time"])


sns.scatterplot(x=df_ao_ss["actual_time"], y=df_ao_ss["osrm_time"])
```

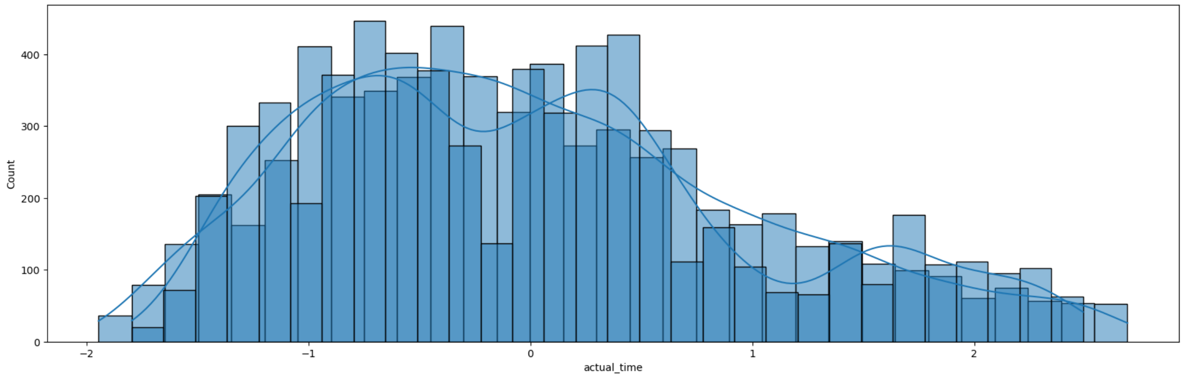
Out[67]:

<AxesSubplot:xlabel='actual_time', ylabel='osrm_time'>



In [68]:

```
plt.figure(figsize = (20,6))

sns.histplot(df_ao_ss['actual_time'], kde = True)
sns.histplot(df_ao_ss['osrm_time'], kde = True)

#plt.legend()
plt.show()
```



In [69]:

```
# hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time agg

# Checking: Does segment_actual_time is similar as segment_osrm_time?

from scipy.stats import ttest_ind
null_hypothesis = 'Mean of actual_time is similar to osrm_time'
alternative_hypothesis = 'Mean of actual_time is different than osrm_time'

sample1 = df_ao_ss['actual_time']
sample2 = df_ao_ss['osrm_time']
t_stat, p_value = ttest_ind(sample1, sample2)
print(t_stat, p_value)

output = Check_hypothesis(p_value)

if(output == 1):
  print(alternative_hypothesis)
else:
  print(null_hypothesis)

# conclusion: mean of actual_time is similar to osrm_time (with following the standardizati
```

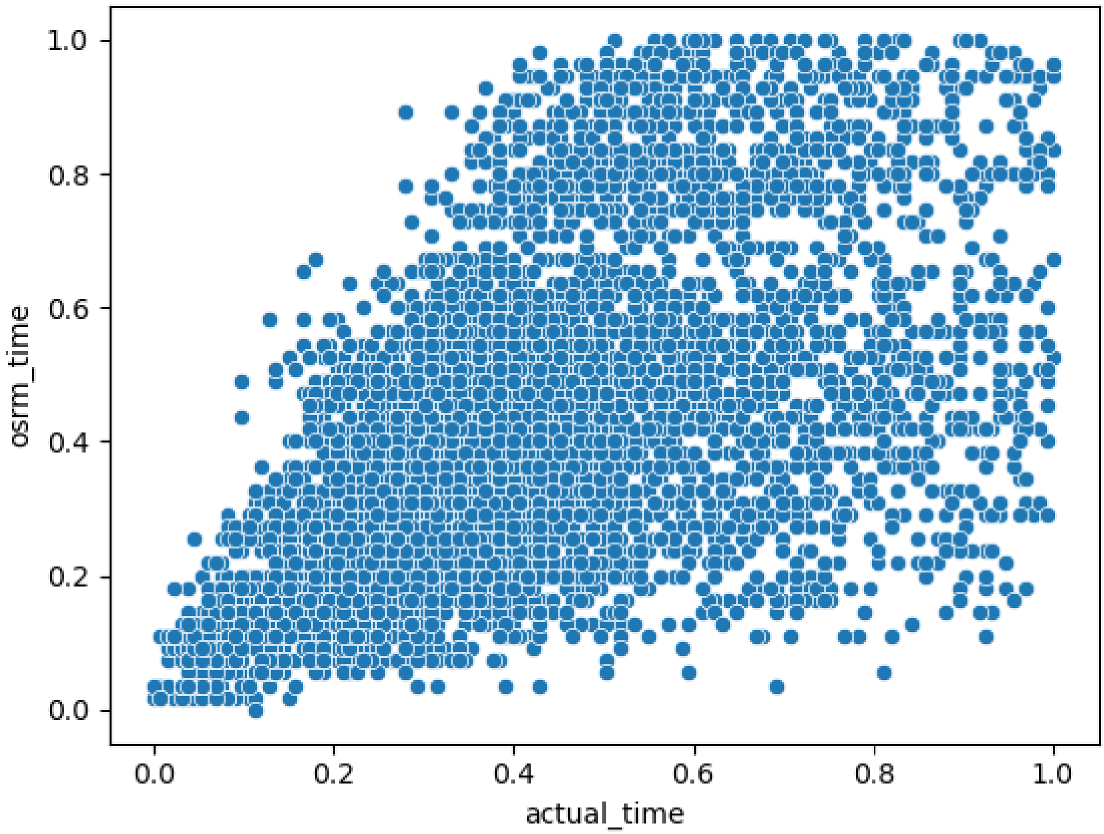
```
5.930481224729144e-15 0.9999999999999953
Fail to Reject Null Hypothesis (No significant difference between two sample
s)
Mean of actual_time is similar to osrm_time
```

In [70]:

```
df_ao_mm = MinMaxScaler().fit_transform(df_ao)
df_ao_mm = pd.DataFrame(df_ao_mm, columns=["actual_time", "osrm_time"])
sns.scatterplot(x=df_ao_mm["actual_time"], y=df_ao_mm["osrm_time"])
```

Out[70]:

```
<AxesSubplot:xlabel='actual_time', ylabel='osrm_time'>
```



In [71]:

```
df_ao_ss.mean()        #chk mean after applying Standard Scaler
```

Out[71]:

```
actual_time    1.651415e-16
osrm_time      1.428705e-16
dtype: float64
```

In [72]:

```
df_ao_mm.mean()        #chk mean after applying Minmax Scaler
```

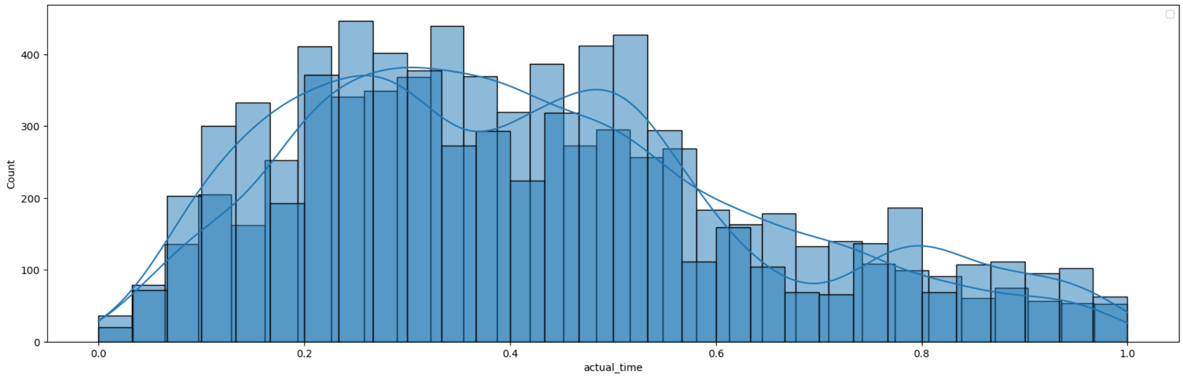
Out[72]:

```
actual_time    0.419885
osrm_time      0.419205
dtype: float64
```

In [73]:

```
plt.figure(figsize = (20,6))

sns.histplot(df_ao_mm['actual_time'], kde = True)
sns.histplot(df_ao_mm['osrm_time'], kde = True)

plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```



Handling categorical values

In [74]:

```
trip.nunique()
```

Out[74]:

```
data                                 2
trip_creation_time                6341
route_schedule_uuid                869
route_type                           2
trip_uuid                         6341
source_center                      444
source_name                        444
destination_center                 459
destination_name                   459
start_scan_to_end_scan             542
od_trip_duration                  6341
actual_distance_to_destination    6330
actual_time                        134
osrm_time                           56
osrm_distance                     6275
segment_actual_time_sum            132
segment_osrm_distance_sum         6292
segment_osrm_time_sum               66
dtype: int64
```

We have 2 categorical variables

1. data

2. route_type

In [75]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
trip[col] = label_encoder.fit_transform(trip['route_type'])
trip[col].value_counts()
```

Out[75]:

```
0    6050
1     291
Name: segment_osrm_time, dtype: int64
```

In [76]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
trip[col] = label_encoder.fit_transform(trip['data'])
trip[col].value_counts()
```

Out[76]:

```
1    4428
0    1913
Name: segment_osrm_time, dtype: int64
```

Handling missing values

In [77]:

```
missing = pd.read_csv(r"D:\Needa\My work\Userprof\delhivery_data.csv")
missing
```

Out[77]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
...	...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND1

144867 rows × 24 columns

In [78]:

```
missing.isna().sum()
```

Out[78]:

```
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
```

Actually out of 144867 we have only 293 source name and 263 destination name are missing. Which we have dropped in this EDA as negligible count

But if we want to impute it , we can use Simple Imputer and replace those null values with most frequent occurence of source name or destination name.

In [79]:

```
from sklearn.impute import SimpleImputer
```

In [80]:

```
missing['source_name']  = SimpleImputer(strategy="most_frequent").fit_transform(missing[['s
```

In [81]:

```
missing['destination_name']  = SimpleImputer(strategy="most_frequent").fit_transform(missin
```

In [82]:

```
missing.isna().sum()
```

Out[82]:

```
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                  0
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
```

Columns split

In [83]:

```
ds = trip[['destination_name']].copy()

new = trip['source_name'].str.split(" ", n = 1, expand = True)
ds['source_city']= new[0]
ds['source_state']= new[1].str[1:-1]

new = trip['destination_name'].str.split(" ", n = 1, expand = True)
ds['destination_city']= new[0]
ds['destination_state']= new[1].str[1:-1]

ds['Corridor'] = ds['source_city']+" To "+ds['destination_city']

ds
```

Out[83]:

	destination_name	source_city	source_state	destination_city
3	Mumbai_MiraRd_IP (Maharashtra)	Mumbai	ub (Maharashtra	Mumbai_MiraRd_IP
5	Chennai_Poonamallee (Tamil Nadu)	Chennai_Poonamallee	Tamil Nadu	Chennai_Poonamallee
6	Chennai_Vandalur_Dc (Tamil Nadu)	Chennai_Chrompet_DPC	Tamil Nadu	Chennai_Vandalur_Dc
7	HBR Layout PC (Karnataka)	HBR	ayout PC (Karnataka	HBR
9	Delhi_Bhogal (Delhi)	Delhi_Lajpat_IP	Delhi	Delhi_Bhogal
...	...	...	...	...
14778	Mumbai_East_I_21 (Maharashtra)	Mumbai_East_I_21	Maharashtra	Mumbai_East_I_21
14779	Chennai_Thiruvlr_DC (Tamil Nadu)	Chennai_Porur_DPC	Tamil Nadu	Chennai_Thiruvlr_DC
14780	Chennai_Sriperumbudur_Dc (Tamil Nadu)	Chennai_Poonamallee	Tamil Nadu	Chennai_Sriperumbudur_Dc
14781	Mumbai_MiraRd_IP (Maharashtra)	Mumbai	ub (Maharashtra	Mumbai_MiraRd_IP
14783	Faridabad_Blbgarh_DC (Haryana)	FBD_Balabhgarh_DPC	Haryana	Faridabad_Blbgarh_DC

6341 rows × 6 columns

In [84]:

```
ds['Corridor'].value_counts()
```

Out[84]:

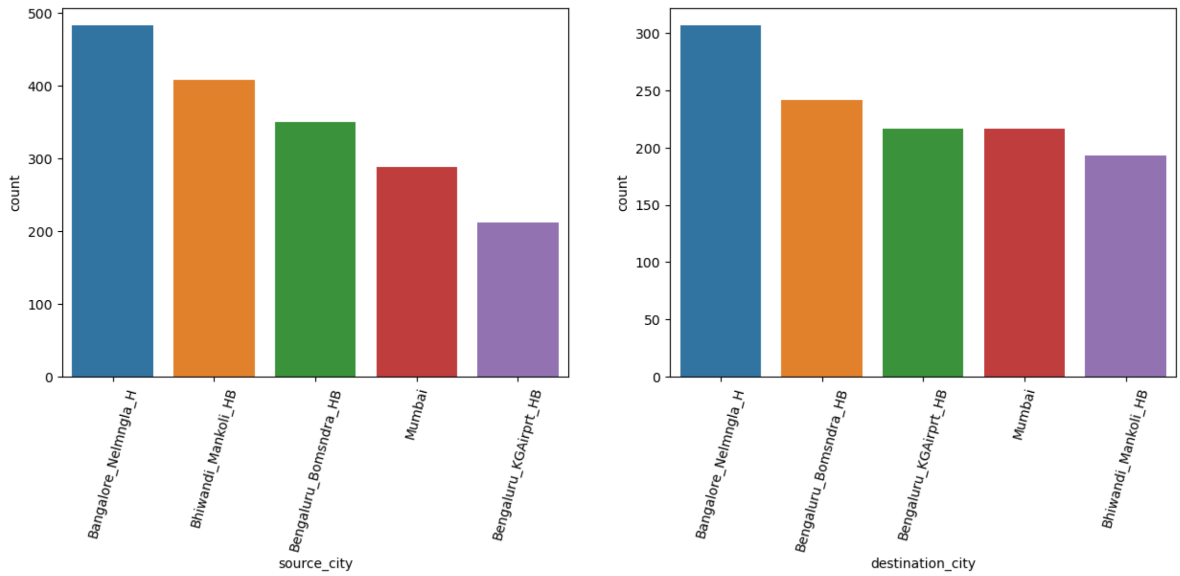
```
Bangalore_Nelmngla_H To Bengaluru_KGAirprt_HB    144
Bhiwandi_Mankoli_HB To Mumbai                    101
Bangalore_Nelmngla_H To Bengaluru_Bomsndra_HB     91
Bengaluru_KGAirprt_HB To Bangalore_Nelmngla_H     90
Mumbai_Chndivli_PC To Bhiwandi_Mankoli_HB         83
                                                ... 
Tirchngode_Mhdhvpur_D To Mettur_RTOroad_D          1
Chennai_Hub To Chennai_Egmore_DPC                  1
ChandroknaRD_Central_DPP_3 To Kharagpur_DC         1
Tamluk_Central_DPP_2 To Haldia_Central_D_2         1
Hapur_Swargash_D To GZB_Mohan_Nagar_DPC            1
Name: Corridor, Length: 745, dtype: int64
```

In [85]:

```
##There are total 745 routes
```

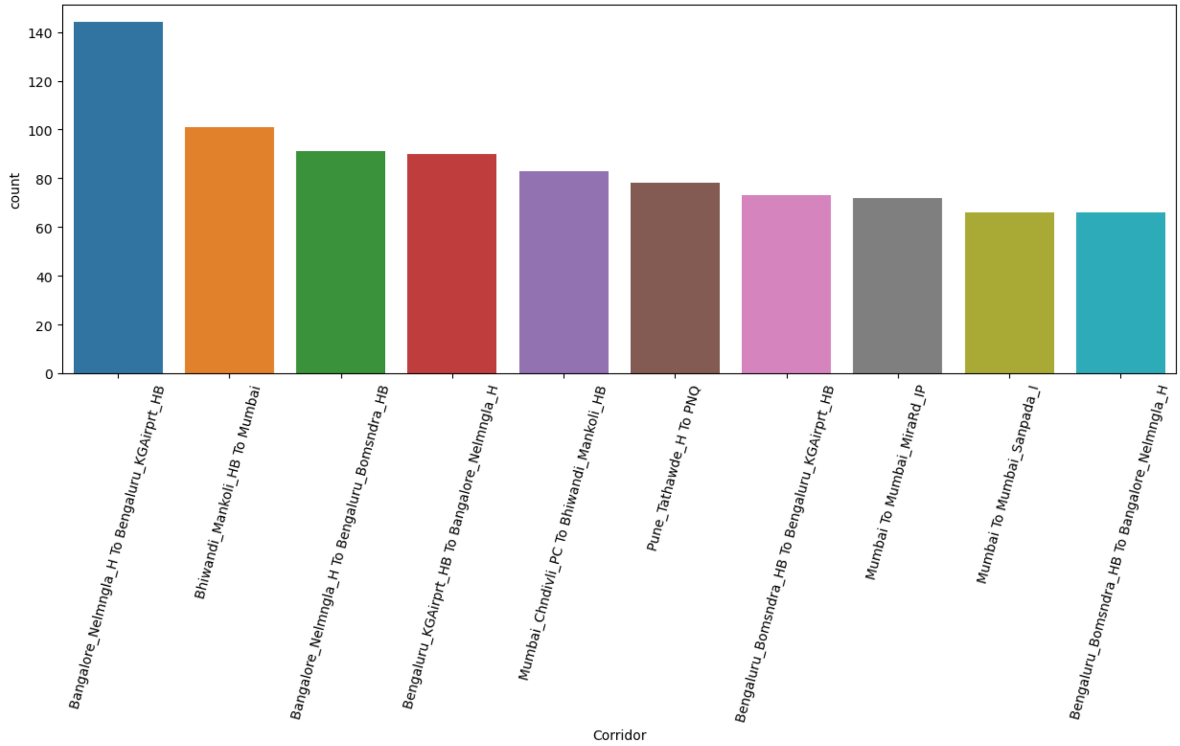
In [86]:

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.countplot(data= ds, x='source_city', order=ds['source_city'].value_counts().nlargest(5)
plt.xticks(rotation = 75)

plt.subplot(1,2,2)
sns.countplot(data= ds, x='destination_city', order=ds['destination_city'].value_counts().n
plt.xticks(rotation = 75)
plt.show()

# Most orders are coming from Bangalore_Nelmangala_H
# Most orders are going to Bangalore_Nelmangala_H
# Left plot: top 5 cities acting as source city
# right plot: top 5 cities acting as destination city
```



In [87]:

```
plt.figure(figsize=(15,5))
sns.countplot(data= ds, x='Corridor', order=ds['Corridor'].value_counts().nlargest(10).inde
plt.xticks(rotation = 75)
plt.show()

# The busiest route is Bangalore_Nelmangala_H To Bengaluru_KGAirport_HB
# Top 10 busiest routes
```

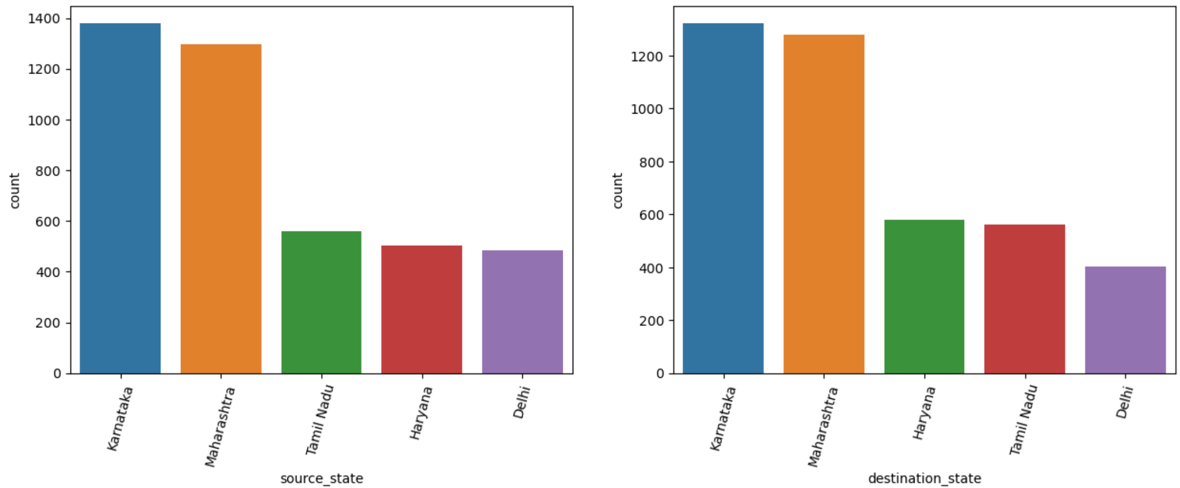


In [88]:

```
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.countplot(data= ds, x='source_state', order=ds['source_state'].value_counts().nlargest(
plt.xticks(rotation = 75)

plt.subplot(1,2,2)
sns.countplot(data= ds, x='destination_state', order=ds['destination_state'].value_counts()
plt.xticks(rotation = 75)

# most orders are coming from Karnataka state
# most orders are going to  Karnataka state
# Left plot: top 5 cities acting as source point
# right plot: top 5 cities acting as destination point
```

Out[88]:

```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'Karnataka'),
  Text(1, 0, 'Maharashtra'),
  Text(2, 0, 'Haryana'),
  Text(3, 0, 'Tamil Nadu'),
  Text(4, 0, 'Delhi')])
```

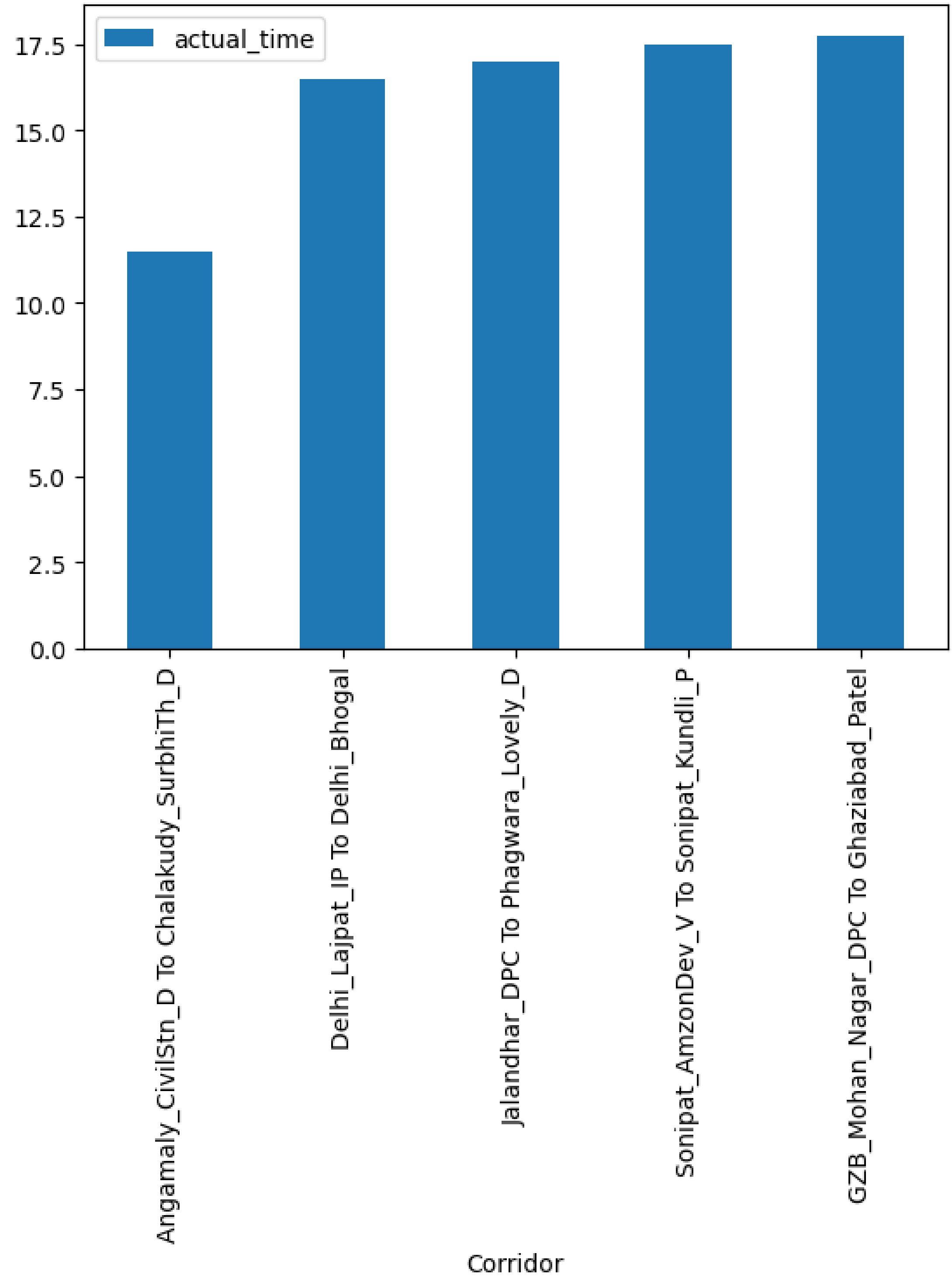


In [89]:

```
dn=pd.concat([trip,ds],axis=1)
```

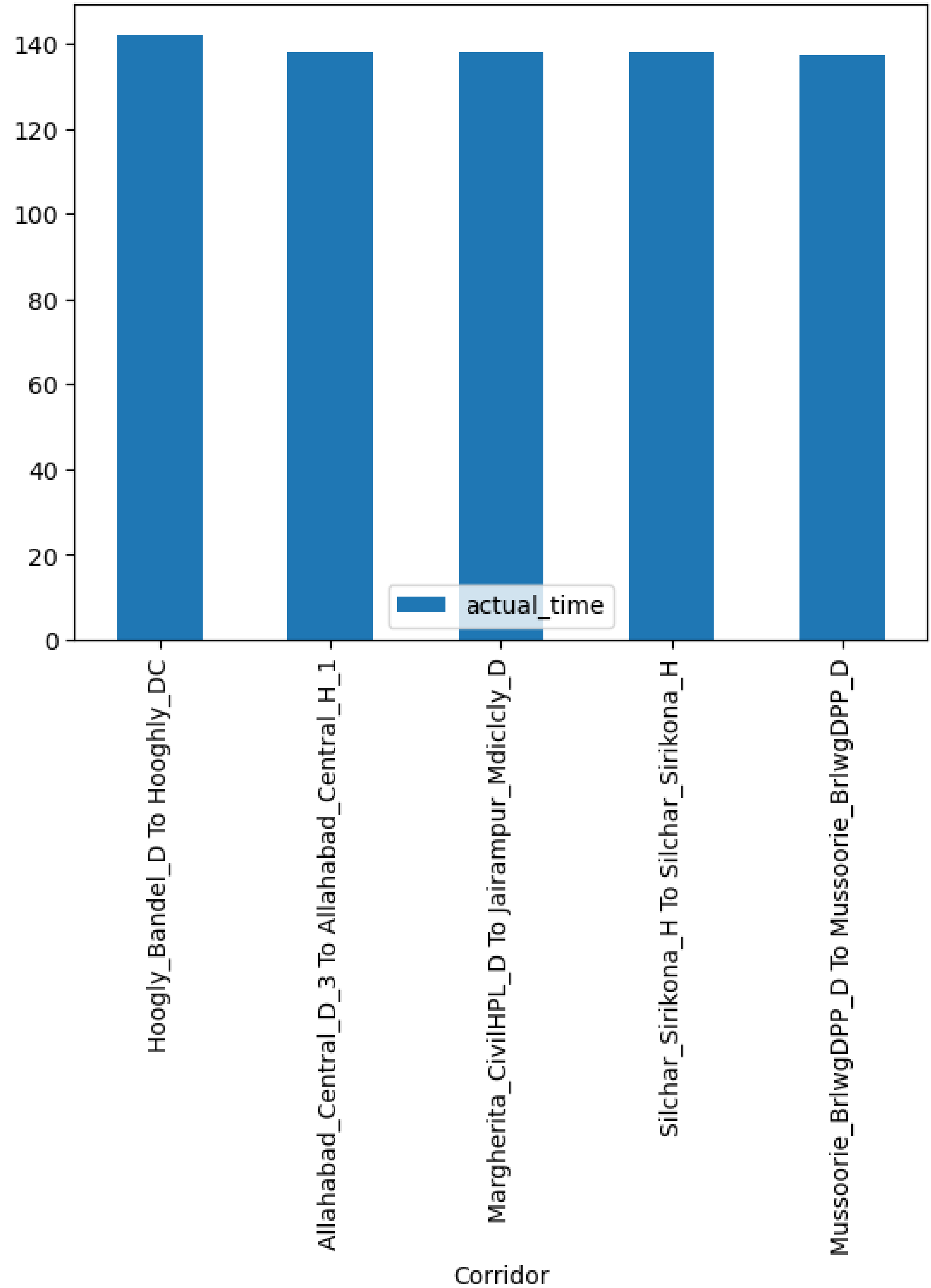
In [90]:

```
dn.groupby('Corridor').agg({'actual_time':'mean'}).nsmallest(5,columns='actual_time').plot(
plt.show()

# Trip between cities Angamaly_CivilStn_D to Chalakudy_SurabhiTh_D saw the least avg time fo
```



In [91]:

```
dn.groupby('Corridor').agg({'actual_time':'mean'}).nlargest(5,columns='actual_time').plot(k
plt.show()

# Trip between cities Hooghly_Bandel_D to Hooghly_DC saw the highest avg time for completio
```



In [92]:

```
ds.describe()
# 1379 orders are coming from source state - Karnataka
# 1322 orders coming to destination state - Karnataka
```

Out[92]:

	destination_name	source_city	source_state	destination_city	destina
count	6341	6341	6341	6341	
unique	459	439	42	455	
top	Bangalore_Nelmngla_H (Karnataka)	Bangalore_Nelmngla_H	Karnataka	Bangalore_Nelmngla_H	
freq	307	483	1379	307	

In [93]:

```
dn.describe()
# Average actual time for a trip is almost 65 Hour
```

Out[93]:

	start_scan_to_end_scan	od_trip_duration	actual_distance_to_destination	actual_time	o:
count	6341.000000	6341.000000	6341.000000	6341.000000	634
mean	164.841350	165.393429	23.648369	64.844662	2
std	119.457103	119.482331	10.047152	28.688862	1
min	23.000000	23.461468	9.002461	9.000000	
25%	96.000000	96.213386	15.729635	43.000000	1
50%	138.000000	138.886469	21.733245	61.000000	2
75%	198.000000	198.779605	31.035562	83.000000	3
max	2701.000000	2701.464958	53.932891	142.000000	6

Business Insights

1. FTL transport uses 69% ansd carting transport uses 31% of total route available. There are 745 routes connecting the start and finish
2. Most of the orders are from Maharashtra, Karnataka, Haryana, Tamil Nadu, Telangana etc.
3. Most of the orders for destinations are from cities such as Bangalore, Mumbai, Gurgaon, Bangalore and Delhi.
4. Biwandi, Delhi, Hyderabad, Chennai, Pune and Chandigarh are also top performers.
5. Karnataka, Maharashtra, Amir Nadu, Terengana and Andhra had the highest distance traveled for interstate travel.
6. Hourly distribution of number of trips per day: Minimum for daytime hours, maximum for late night or early morning hours

7. weekday: The maximum number of trips occurs on Wednesday and the minimum on Sunday.
8. OSRM seems to calculate the duration less than the actual duration. This is because in real-world scenarios, you may experience delays due to unprecedented traffic or other delays.
9. The actual_time average is higher than the osrm_time average.
10. Average of actual_time is different than average of segment_osrm_time.
11. Average of osrm_distance is similar to average of segment_osrm_distance
12. Traveling between cities from Angamaly_CivilStn_D to Chalakudy_SurabhiTh_D took the least time on average. 14. Hooghly_Bandel_D to Hooghly_D had the longest average intercity travel time
13. Removed outliers and missing values for variables with extreme right-skewed distributions
14. Visualizing the plot showed no significant difference between the paired variables

Recommendations:

1. FTL shipping speed is faster than cart. Cart delivery speed needs to be improved
2. Maximum delivery load on Wednesday: All deliveries must be made in the evening or early morning to avoid daytime traffic.
3. Bangalore city has the highest delivery volume. The expansion of Bangalore's automated sorting center will make parcel management more efficient
4. From a state perspective, certain states may have very heavy traffic and some may have poor terrain. This is a great indicator for planning and meeting demand during peak festival season.
5. Intra-state or intra-city travel is more likely to be carted as a mode of transportation, which may increase the number of city and state hubs that make the greatest contribution to transportation.
6. OSRM's voyage planning system needs improvement. Carrier deviation must be taken into account when routin
7. Resources should be allocated to states/cities with the highest transportation contribution (especially during local festivals).
8. The road network is expected to increase the number of her FTL deliveries between states, connecting states with less traffic.

In [ ]:

```

```