1. Problem Statement:

The company wants to understand and process the data coming out of data engineering pipelines:

• Clean, sanitize and manipulate data to get useful features out of raw fields.

• Make sense out of the raw data and help the data science team to build forecasting models on it.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm, binom, geom , ttest_1samp, ttest_ind,
ttest_rel, chisquare , chi2 , chi2_contingency, f_oneway, shapiro,
levene, kruskal, pearsonr, spearmanr, poisson, expon

df = pd.read_csv("delhivery_data.csv")
print(df.head())
```

```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

                              route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting

                 trip_uuid source_center                source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center              destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
```

```
               od_start_time  ...            cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  ...         2018-09-20 04:27:55
1  2018-09-20 03:21:32.418600  ...         2018-09-20 04:17:55
2  2018-09-20 03:21:32.418600  ...  2018-09-20 04:01:19.505586
3  2018-09-20 03:21:32.418600  ...         2018-09-20 03:39:57
4  2018-09-20 03:21:32.418600  ...         2018-09-20 03:33:55

   actual_distance_to_destination  actual_time  osrm_time osrm_distance  \
0                       10.435660         14.0       11.0       11.9653
1                       18.936842         24.0       20.0       21.7243
2                       27.637279         40.0       28.0       32.5395
3                       36.118028         62.0       40.0       45.5620
4                       39.386040         68.0       44.0       54.2181

     factor  segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0  1.272727                 14.0               11.0                11.9653
1  1.200000                 10.0                9.0                 9.7590
2  1.428571                 16.0                7.0                10.8152
3  1.550000                 21.0               12.0                13.0224
4  1.545455                  6.0                5.0                 3.9153

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

```
shape = df.shape
print(f"Shape of the data is {shape}")
```

```
Shape of the data is (144867, 24)
```

```
df.info()
```

```
 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)
```

```
print(df.describe())
```

	data	trip_creation_time	\
count	144867	144867	
unique	2	14817	
top	training	2018-09-28 05:23:15.359220	
freq	104858	101	
mean	NaN	NaN	
std	NaN	NaN	
min	NaN	NaN	
25%	NaN	NaN	
50%	NaN	NaN	
75%	NaN	NaN	
max	NaN	NaN	

	route_schedule_uuid	route_type	\
count	144867	144867	
unique	1504	2	
top	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	
freq	1812	99660	
mean	NaN	NaN	
std	NaN	NaN	
min	NaN	NaN	
25%	NaN	NaN	
50%	NaN	NaN	
75%	NaN	NaN	
max	NaN	NaN	

	trip_uuid	source_center	source_name	\
count	144867	144867	144574	
unique	14817	1508	1498	
top	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	
freq	101	23347	23347	
mean	NaN	NaN	NaN	
std	NaN	NaN	NaN	
min	NaN	NaN	NaN	
25%	NaN	NaN	NaN	
50%	NaN	NaN	NaN	
75%	NaN	NaN	NaN	
max	NaN	NaN	NaN	

	destination_center	destination_name	\
count	144867	144606	
unique	1481	1468	
top	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	
freq	15192	15192	
mean	NaN	NaN	
std	NaN	NaN	
min	NaN	NaN	
25%	NaN	NaN	
50%	NaN	NaN	
75%	NaN	NaN	
max	NaN	NaN	

	od_start_time	...	cutoff_timestamp	\
count	144867	...	144867	
unique	26369	...	93180	
top	2018-09-21 18:37:09.322207	...	2018-09-24 05:19:20	
freq	81	...	40	
mean	NaN	...	NaN	
std	NaN	...	NaN	
min	NaN	...	NaN	
25%	NaN	...	NaN	
50%	NaN	...	NaN	
75%	NaN	...	NaN	
max	NaN	...	NaN	

	actual_distance_to_destination	actual_time	osrm_time	\
count	144867.000000	144867.000000	144867.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	234.073372	416.927527	213.868272	
std	344.990009	598.103621	308.011085	
min	9.000045	9.000000	6.000000	
25%	23.355874	51.000000	27.000000	
50%	66.126571	132.000000	64.000000	
75%	286.708875	513.000000	257.000000	
max	1927.447705	4532.000000	1686.000000	

	osrm_distance	factor	segment_actual_time	segment_osrm_time	\
count	144867.000000	144867.000000	144867.000000	144867.000000	
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	284.771297	2.120107	36.196111	18.507548	
std	421.119294	1.715421	53.571158	14.775960	
min	9.008200	0.144000	-244.000000	0.000000	
25%	29.914700	1.604264	20.000000	11.000000	
50%	78.525800	1.857143	29.000000	17.000000	
75%	343.193250	2.213483	40.000000	22.000000	
max	2326.199100	77.387097	3051.000000	1611.000000	

	segment_osrm_distance	segment_factor
count	144867.00000	144867.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	22.82902	2.218368
std	17.86066	4.847530
min	0.00000	-23.444444
25%	12.07010	1.347826
50%	23.51300	1.684211
75%	27.81325	2.250000
max	2191.40370	574.250000

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

```
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
```

- **`source_name` and `destination_name` contain missing values.**

```
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"])
df["od_start_time"] = pd.to_datetime(df["od_start_time"])
df["od_end_time"] = pd.to_datetime(df["od_end_time"])
```

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

```
September    127349
October       17518
Name: trip_creation_time, dtype: int64
```

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

```
2018    144867
Name: trip_creation_time, dtype: int64
```

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

```
Wednesday    26732
Thursday     20481
Friday       20242
Tuesday      19961
Saturday     19936
Monday       19645
Sunday       17870
Name: trip_creation_time, dtype: int64
```

- **Datepoints are from the month of September and October of year 2018**

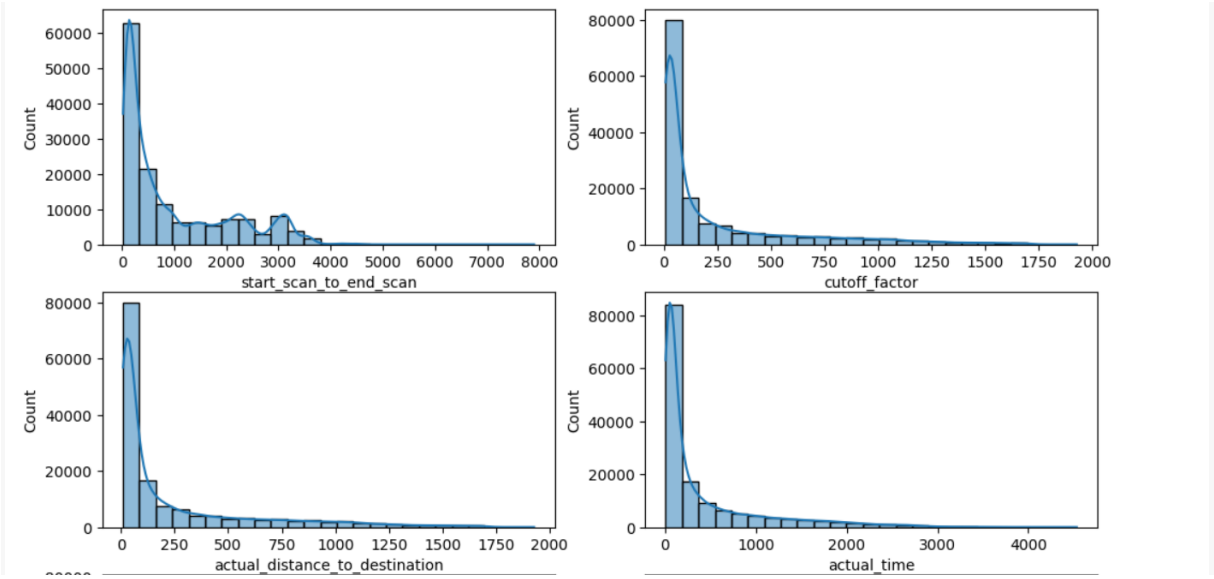
```
df.nunique()
```

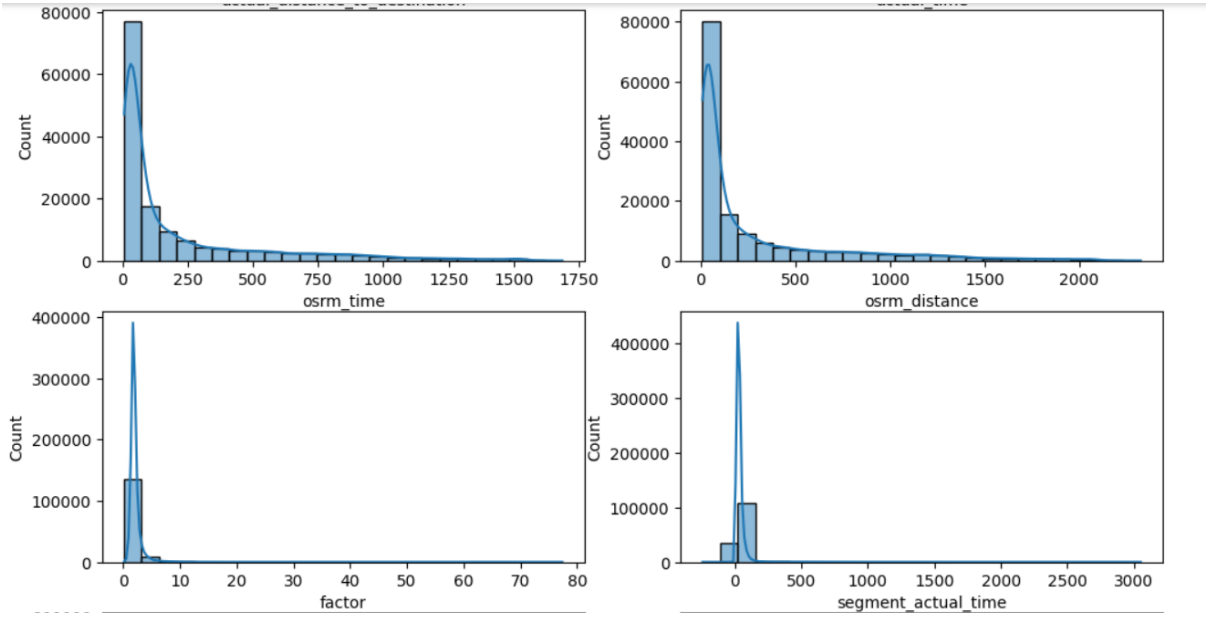
```
data                                  2
trip_creation_time                14817
route_schedule_uuid                1504
route_type                            2
trip_uuid                         14817
source_center                      1508
source_name                        1498
destination_center                 1481
destination_name                   1468
od_start_time                     26369
od_end_time                       26369
start_scan_to_end_scan             1915
is_cutoff                             2
cutoff_factor                       501
cutoff_timestamp                  93180
actual_distance_to_destination   144515
actual_time                        3182
osrm_time                          1531
osrm_distance                    138046
factor                            45641
segment_actual_time                 747
segment_osrm_time                   214
segment_osrm_distance            113799
segment_factor                     5675
dtype: int64
```

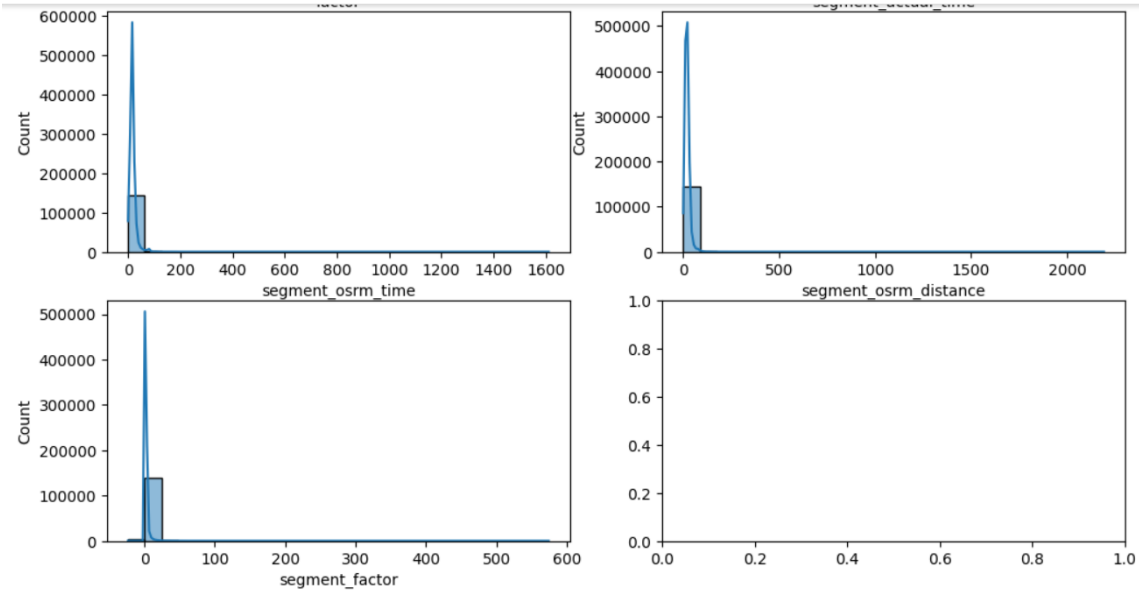
- **There is total 14817 different trips of data available.**
- **There are 1508 unique source_center.**
- **There are 1481 unique destination_center.**
- **There is total 1504 delivery routes.**

Univariate Analysis on numerical columns:

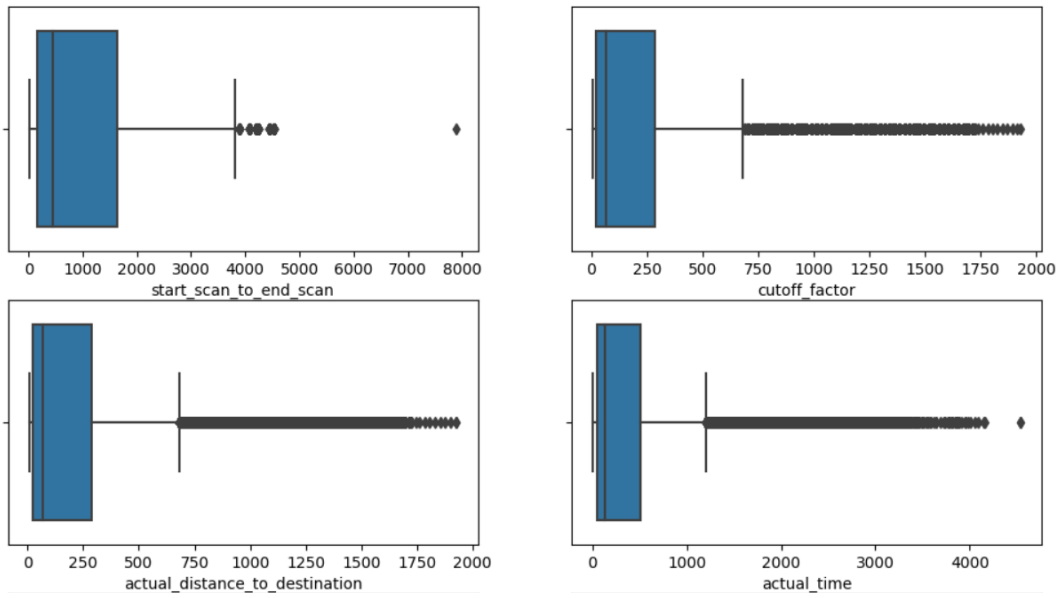
```
num_vars = df.select_dtypes(include=np.number).columns.tolist()
fig, ax = plt.subplots(nrows=6, ncols=2, figsize=(12, 25))
count = 0
for i in range(6):
  for j in range(2) :
    if count < len(num_vars) :
      sns.histplot(x=df[num_vars[count]], kde=True, bins = 25, ax =
ax[i, j])
      count += 1
plt.show()
```

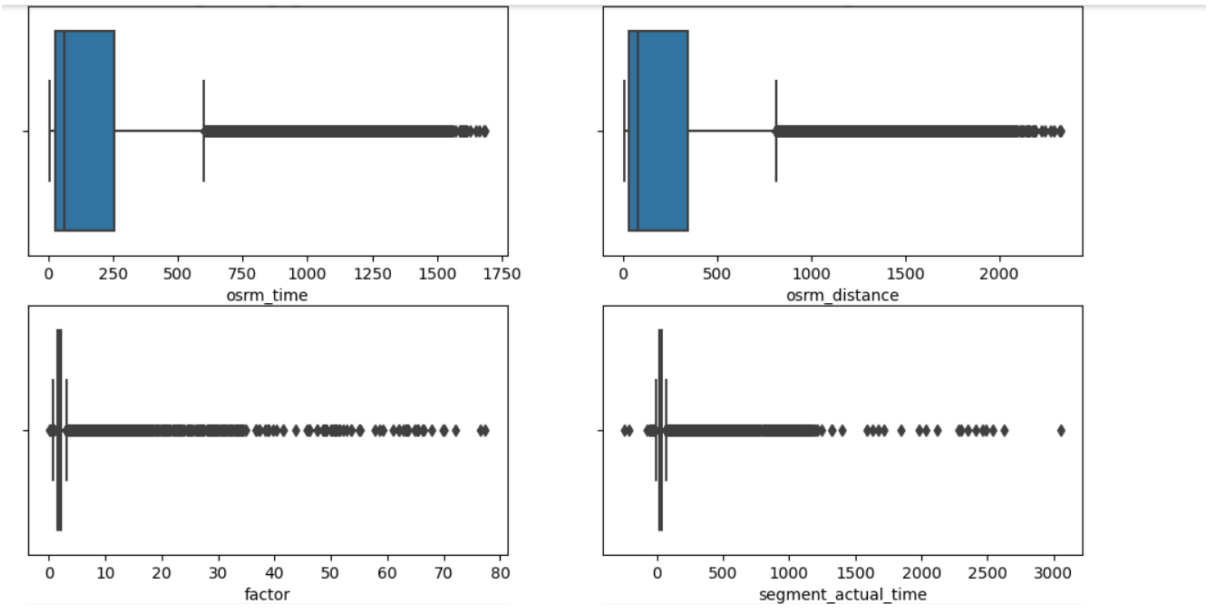

Count
start_scan_to_end_scan
cutoff_factor
actual_distance_to_destination
actual_time

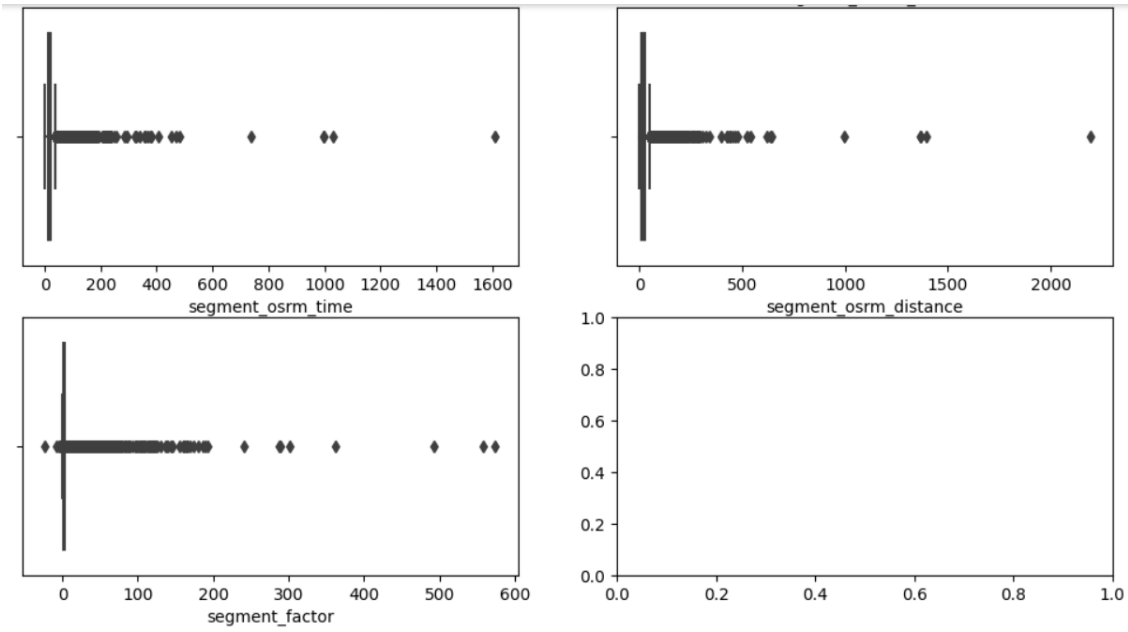

Count
osrm_time
osrm_distance
factor
segment_actual_time




```
fig, ax = plt.subplots(nrows=6, ncols=2, figsize=(12, 20))
count = 0
for i in range(6):
  for j in range(2) :
    if count < len(num_vars) :
      sns.boxplot(x=df[num_vars[count]], ax = ax[i, j])
      count += 1

plt.show()
```





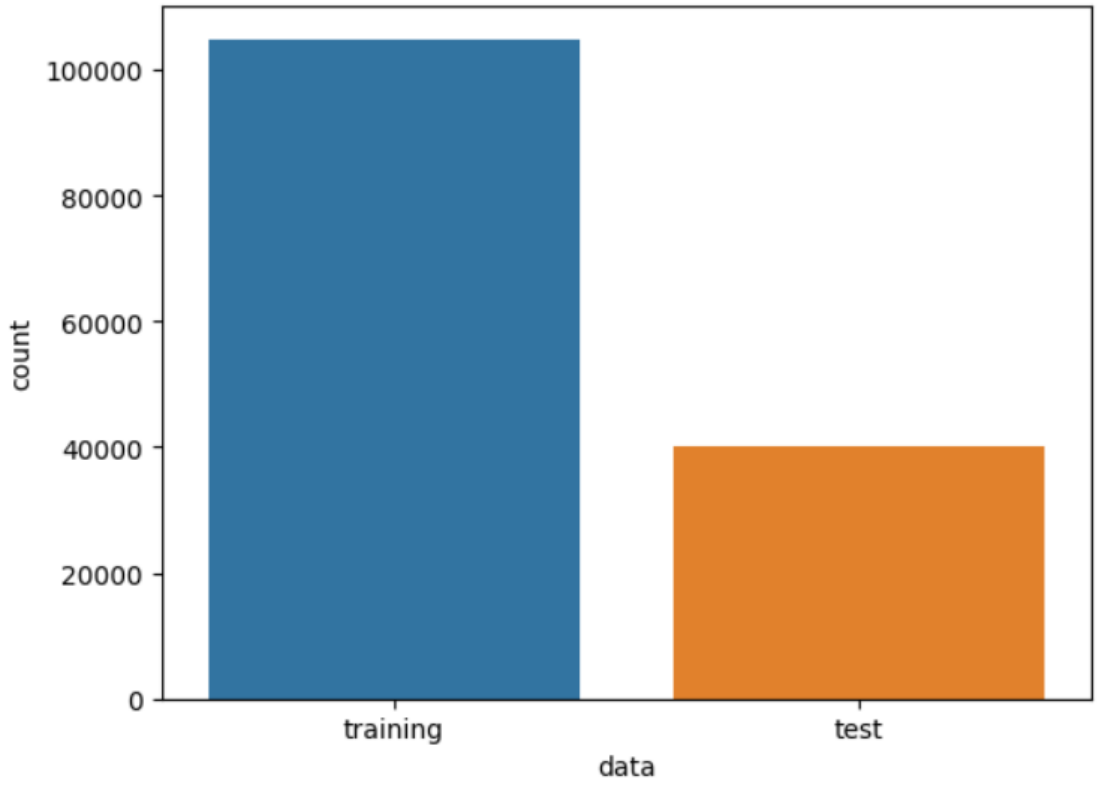


- There are outliers present in the numerical data.

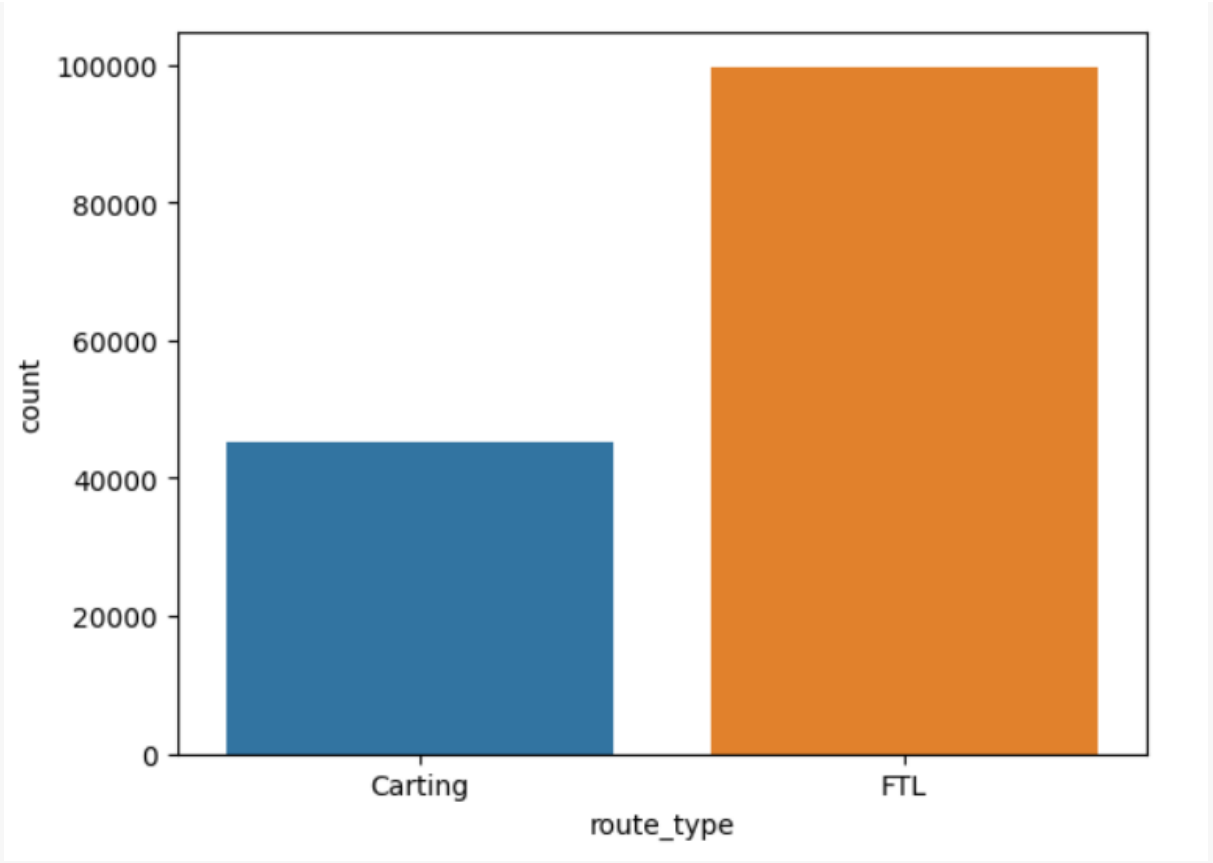
Univariate Analysis on categorical columns:

```
cat_vars = df.select_dtypes(include= "object").columns.tolist()
```

```
sns.countplot(x=df["data"])
plt.show()
```

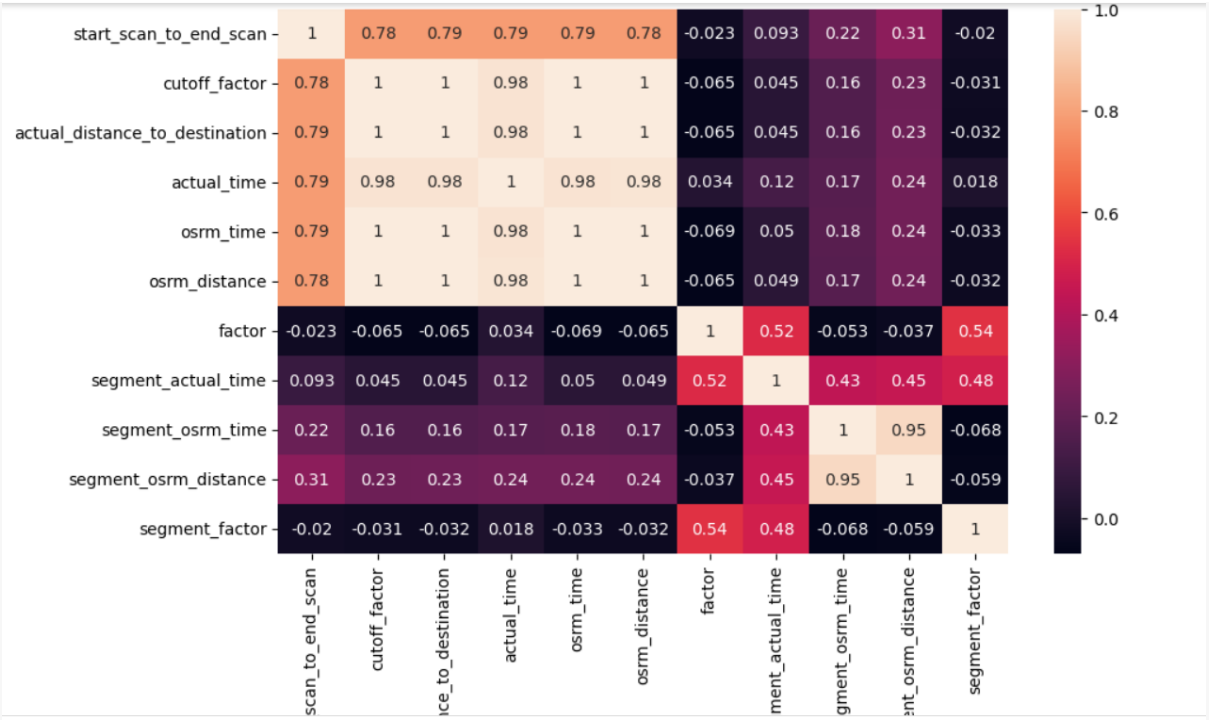



```
sns.countplot(x=df["route_type"])
plt.show()
```



Multivariate analysis:

```
plt.figure(figsize = (10 , 6))
sns.heatmap(data = df[num_vars].corr() , annot =True)
plt.show()
```



2. Feature Creation:

Extracting Features like city - place - pin code -state from source and destination name columns:

```
df["source_city"] = df["source_name"].str.split("
",n=1,expand=True)[0].str.split("_",n=1,expand=True)[0]
df["source_state"] = df["source_name"].str.split("
",n=1,expand=True)[1].str.replace("(","").str.replace(")","")

df["destination_city"] = df["destination_name"].str.split("
",n=1,expand=True)[0].str.split("_",n=1,expand=True)[0]
df["destination_state"] = df["destination_name"].str.split("
",n=1,expand=True)[1].str.replace("(","").str.replace(")","")

df["source_pincode"] = df["source_center"].apply(lambda x : x[3:9] )
df["destination_pincode"] = df["destination_center"].apply(lambda x :
x[3:9] )


df["time_taken_btwn_odstart_and_od_end"] = ((df["od_end_time"]-
df["od_start_time"])/pd.Timedelta(1,unit="hour"))
```

Converting given time duration features into hours.

```
df["start_scan_to_end_scan"] = df["start_scan_to_end_scan"]/60
df["actual_time"] = df["actual_time"]/60
df["osrm_time"] = df["osrm_time"]/60
df["segment_actual_time"] = df["segment_actual_time"]/60
df["segment_osrm_time"] = df["segment_osrm_time"]/60
```

```
       data         trip_creation_time  \
0  training 2018-09-20 02:35:36.476840
1  training 2018-09-20 02:35:36.476840
2  training 2018-09-20 02:35:36.476840
3  training 2018-09-20 02:35:36.476840
4  training 2018-09-20 02:35:36.476840

                               route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting

                 trip_uuid source_center                 source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center              destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)

               od_start_time  ... segment_actual_time  segment_osrm_time  \
0 2018-09-20 03:21:32.418600  ...            0.233333           0.183333
1 2018-09-20 03:21:32.418600  ...            0.166667           0.150000
2 2018-09-20 03:21:32.418600  ...            0.266667           0.116667
3 2018-09-20 03:21:32.418600  ...            0.350000           0.200000
4 2018-09-20 03:21:32.418600  ...            0.100000           0.083333
```

```
   segment_osrm_distance  segment_factor source_city  source_state  \
0                11.9653        1.272727       Anand       Gujarat
1                 9.7590        1.111111       Anand       Gujarat
2                10.8152        2.285714       Anand       Gujarat
3                13.0224        1.750000       Anand       Gujarat
4                 3.9153        1.200000       Anand       Gujarat

   destination_city  destination_state  source_pincode  destination_pincode
0         Khambhat            Gujarat          388121               388620
1         Khambhat            Gujarat          388121               388620
2         Khambhat            Gujarat          388121               388620
3         Khambhat            Gujarat          388121               388620
4         Khambhat            Gujarat          388121               388620

[5 rows x 30 columns]
```

Analysing Dataset after feature creation.

```
df.info()
```

```
Data columns (total 30 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  datetime64[ns]
 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  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 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
 24  source_city                     144574 non-null  object
 25  source_state                    144574 non-null  object
 26  destination_city                144606 non-null  object
 27  destination_state               144606 non-null  object
 28  source_pincode                  144867 non-null  object
 29  destination_pincode             144867 non-null  object
dtypes: bool(1), datetime64[ns](3), float64(10), int64(1), object(15)
```

Data Cleaning:

```
df["source_state"] = df["source_state"].replace({"Goa Goa":"Goa",
                            "Layout PC Karnataka":"Karnataka",
                           "Vadgaon Sheri DPCMaharashtra":"Maharashtra",
                            "Pashan DPC Maharashtra":"Maharashtra",
                            "City Madhya Pradesh":"Madhya Pradesh",
                            "02_DPC Uttar Pradesh":"Uttar Pradesh",
                            "Nagar_DC Rajasthan":"Rajasthan",
                            "Alipore_DPC West Bengal":"West Bengal",
                             "Mandakni Madhya Pradesh":"Madhya Pradesh",
                             "West _Dc Maharashtra":"Maharashtra",
                             "DC Rajasthan":"Rajasthan",
                             "MP Nagar Madhya Pradesh":"Madhya Pradesh",
                             "Antop Hill Maharashtra":"Maharashtra",
                             "Avenue_DPC West Bengal":"West Bengal",
                             "Nagar Uttar Pradesh":"Uttar Pradesh",
                             "Balaji Nagar Maharashtra":"Maharashtra",
                             "Kothanur_L Karnataka":"Karnataka",
                             "Rahatani DPC Maharashtra":"Maharashtra",
                             "Mahim Maharashtra":"Maharashtra",
                             "DC Maharashtra":"Maharashtra",
                             "_NAD Andhra Pradesh":"Andhra Pradesh" })

df["destination_state"] = df["destination_state"].replace({"Goa
Goa":"Goa",
                            "Layout PC Karnataka":"Karnataka",
                            "Vadgaon Sheri DPC
Maharashtra":"Maharashtra",
                            "Pashan DPC Maharashtra":"Maharashtra",
                            "City Madhya Pradesh":"Madhya Pradesh",
                            "02_DPC Uttar Pradesh":"Uttar Pradesh",
                            "Nagar_DC Rajasthan":"Rajasthan",
                            "Alipore_DPC West Bengal":"West Bengal",
                             "Mandakni Madhya Pradesh":"Madhya Pradesh",
                             "West _Dc Maharashtra":"Maharashtra",
                             "DC Rajasthan":"Rajasthan",
                             "MP Nagar Madhya Pradesh":"Madhya Pradesh",
                             "Antop Hill Maharashtra":"Maharashtra",
                             "Avenue_DPC West Bengal":"West Bengal",
                             "Nagar Uttar Pradesh":"Uttar Pradesh",
                             "Balaji Nagar Maharashtra":"Maharashtra",
                             "Kothanur_L Karnataka":"Karnataka",
                             "Rahatani DPC Maharashtra":"Maharashtra",
                             "Mahim Maharashtra":"Maharashtra",
                             "DC Maharashtra":"Maharashtra",
```

```
                              "_NAD Andhra Pradesh":"Andhra Pradesh",
"Delhi Delhi":"Delhi", "West_Dc  Maharashtra":"Maharashtra",
                             "Hub Maharashtra":"Maharashtra"})
```

```
df["destination_city"].replace({"del":"Delhi"},inplace=True)
df["source_city"].replace({"del":"Delhi"},inplace=True)
df["source_city"].replace({"Bangalore":"Bengaluru"},inplace=True)
df["destination_city"].replace({"Bangalore":"Bengaluru"},inplace=True)
df["destination_city"].replace({"AMD":"Ahmedabad"},inplace=True)
df["destination_city"].replace({"Amdavad":"Ahmedabad"},inplace=True)
df["source_city"].replace({"AMD":"Ahmedabad"},inplace=True)
df["source_city"].replace({"Amdavad":"Ahmedabad"},inplace=True)
```

Creating Feature - (Source city + state & Destination city + state)

```
df["source_city_state"] = df["source_city"] + " " + df["source_state"]
df["destination_city_state"] = df["destination_city"] + " " +
df["destination_state"]

df["source_city_state"].nunique()
```

```
1249
```

```
df["destination_city_state"].nunique()
```

```
1242
```

```
df["source_state"].nunique()
```

```
  33
```

```
df["destination_state"].nunique()
```

```
32
```

- **Company delivers in approximately all the states and cities of India**

Dropping Unnecessary columns:

```
new_df = df.copy()

new_df.shape
```

```
(144867, 32)
```

```
new_df.drop(['source_center',"source_name","destination_center","destination_name","cutoff_timestamp", "od_end_time","od_start_time"], axis = 1, inplace=True)

new_df.shape
```

```
(144867, 25)
```

3. Merging of rows and aggregation of fields:

```
actual_time = new_df.groupby(["trip_uuid",
               "start_scan_to_end_scan"])["actual_time"].max().reset_index().groupby("trip_uuid")["actual_time"].sum().reset_index()

actual_time
```

	trip_uuid	actual_time
0	trip-153671041653548748	26.033333
1	trip-153671042288605164	2.383333
2	trip-153671043369099517	55.783333
3	trip-153671046011330457	0.983333
4	trip-153671052974046625	5.683333
...	...	...
14812	trip-153861095625827784	1.383333
14813	trip-153861104386292051	0.350000
14814	trip-153861106442901555	4.700000
14815	trip-153861115439069069	4.400000
14816	trip-153861118270144424	4.583333

```
segment_osrm_time =
new_df[["trip_uuid","segment_osrm_time"]].groupby("trip_uuid")["segment
_osrm_time"].sum().reset_index()
segment_osrm_time
```

	trip_uuid	segment_osrm_time
0	trip-153671041653548748	16.800000
1	trip-153671042288605164	1.083333
2	trip-153671043369099517	32.350000
3	trip-153671046011330457	0.266667
4	trip-153671052974046625	1.916667
...	...	...
14812	trip-153861095625827784	1.033333
14813	trip-153861104386292051	0.183333
14814	trip-153861106442901555	1.466667
14815	trip-153861115439069069	3.683333
14816	trip-153861118270144424	1.116667

14817 rows × 2 columns

```
segment_actual_time =
new_df.groupby("trip_uuid")["segment_actual_time"].sum().reset_index()
segment_actual_time
```

	trip_uuid	segment_actual_time
0	trip-153671041653548748	25.800000
1	trip-153671042288605164	2.350000
2	trip-153671043369099517	55.133333
3	trip-153671046011330457	0.983333
4	trip-153671052974046625	5.666667
...	...	...
14812	trip-153861095625827784	1.366667
14813	trip-153861104386292051	0.350000
14814	trip-153861106442901555	4.683333
14815	trip-153861115439069069	4.300000
14816	trip-153861118270144424	4.566667

14817 rows × 2 columns

```
osrm_time =
new_df.groupby(["trip_uuid","start_scan_to_end_scan"])["osrm_time"].max
().reset_index().groupby("trip_uuid")["osrm_time"].sum().reset_index()
osrm_time
```

	trip_uuid	osrm_time
0	trip-153671041653548748	12.383333
1	trip-153671042288605164	1.133333
2	trip-153671043369099517	29.016667
3	trip-153671046011330457	0.250000
4	trip-153671052974046625	1.950000
...	...	...
14812	trip-153861095625827784	1.033333
14813	trip-153861104386292051	0.200000
14814	trip-153861106442901555	0.900000
14815	trip-153861115439069069	3.066667
14816	trip-153861118270144424	1.133333

14817 rows × 2 columns

```
time_taken_btwn_odstart_and_od_end =
new_df.groupby("trip_uuid")["time_taken_btwn_odstart_and_od_end"].uniqu
e().reset_index()
time_taken_btwn_odstart_and_od_end
```

	trip_uuid	time_taken_btwn_odstart_and_od_end
0	trip-153671041653548748	[16.65842298, 21.0100736875]
1	trip-153671042288605164	[2.0463247669444447, 0.9805397955555556]
2	trip-153671043369099517	[51.662059856388886, 13.910648811388889]
3	trip-153671046011330457	[1.6749155866666667]
4	trip-153671052974046625	[2.5335485744444446, 1.3423885633333332, 8.096...
...	...	...
14812	trip-153861095625827784	[2.546464057777778, 1.7540180775]
14813	trip-153861104386292051	[1.0098420219444444]
14814	trip-153861106442901555	[2.895179575833333, 4.1401515375]
14815	trip-153861115439069069	[1.7609491794444445, 0.7362400538888889, 1.035...
14816	trip-153861118270144424	[1.1155594141666667, 4.7912334425]

14817 rows × 2 columns

```
time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"
]=time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_en
d"].apply(sum)

time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"
]
```

```
0        37.668497
1         3.026865
2        65.572709
3         1.674916
4        11.972484
           ...
14812     4.300482
14813     1.009842
14814     7.035331
14815     5.808548
14816     5.906793
Name: time_taken_btwn_odstart_and_od_end, Length: 14817, dtype: float64
```

```
start_scan_to_end_scan =
((new_df.groupby("trip_uuid")["start_scan_to_end_scan"].unique())).rese
t_index()
start_scan_to_end_scan
```

	trip_uuid	start_scan_to_end_scan
0	trip-153671041653548748	[16.65, 21.0]
1	trip-153671042288605164	[2.033333333333333, 0.9666666666666667]
2	trip-153671043369099517	[51.65, 13.9]
3	trip-153671046011330457	[1.6666666666666667]
4	trip-153671052974046625	[2.533333333333333, 1.3333333333333333, 8.0833...
...	...	...
14812	trip-153861095625827784	[2.533333333333333, 1.75]
14813	trip-153861104386292051	[1.0]
14814	trip-153861106442901555	[2.8833333333333333, 4.133333333333334]
14815	trip-153861115439069069	[1.75, 0.7333333333333333, 1.0333333333333334,...
14816	trip-153861118270144424	[1.1, 4.783333333333333]

14817 rows × 2 columns

```
start_scan_to_end_scan["start_scan_to_end_scan"] =
start_scan_to_end_scan["start_scan_to_end_scan"].apply(sum)
start_scan_to_end_scan["start_scan_to_end_scan"]
```

```
0        37.650000
1         3.000000
2        65.550000
3         1.666667
4        11.950000
           ...
14812     4.283333
14813     1.000000
14814     7.016667
14815     5.783333
14816     5.883333
Name: start_scan_to_end_scan, Length: 14817, dtype: float64
```

```
osrm_distance =
new_df.groupby(["trip_uuid","start_scan_to_end_scan"])["osrm_distance"]
.max().reset_index().groupby("trip_uuid")["osrm_distance"].sum().reset_
index()

osrm_distance
```

	trip_uuid	osrm_distance
0	trip-153671041653548748	991.3523
1	trip-153671042288605164	85.1110
2	trip-153671043369099517	2372.0852
3	trip-153671046011330457	19.6800
4	trip-153671052974046625	146.7918
...	...	...
14812	trip-153861095625827784	73.4630
14813	trip-153861104386292051	16.0882
14814	trip-153861106442901555	63.2841
14815	trip-153861115439069069	177.6635
14816	trip-153861118270144424	80.5787

14817 rows × 2 columns

```
actual_distance_to_destination =
new_df.groupby(["trip_uuid","start_scan_to_end_scan"])["actual_distance
_to_destination"].max().reset_index().groupby("trip_uuid")["actual_dist
ance_to_destination"].sum().reset_index()

actual_distance_to_destination
```

	trip_uuid	actual_distance_to_destination
0	trip-153671041653548748	824.732854
1	trip-153671042288605164	73.186911
2	trip-153671043369099517	1932.273969
3	trip-153671046011330457	17.175274
4	trip-153671052974046625	127.448500
...	...	...
14812	trip-153861095625827784	57.762332
14813	trip-153861104386292051	15.513784
14814	trip-153861106442901555	38.684839
14815	trip-153861115439069069	134.723836
14816	trip-153861118270144424	66.081533

14817 rows × 2 columns

```
segment_osrm_distance = new_df[["trip_uuid",
"segment_osrm_distance"]].groupby("trip_uuid")["segment_osrm_distance"]
.sum().reset_index()
segment_osrm_distance
```

	trip_uuid	segment_osrm_distance
0	trip-153671041653548748	1320.4733
1	trip-153671042288605164	84.1894
2	trip-153671043369099517	2545.2678
3	trip-153671046011330457	19.8766
4	trip-153671052974046625	146.7919
...	...	...
14812	trip-153861095625827784	64.8551
14813	trip-153861104386292051	16.0883
14814	trip-153861106442901555	104.8866
14815	trip-153861115439069069	223.5324
14816	trip-153861118270144424	80.5787

14817 rows × 2 columns

```
grouping_1 = ['trip_uuid', 'source_center', 'destination_center']
df1 = df.groupby(by = grouping_1, as_index = False).agg({'data' :
'first','route_type' : 'first','trip_creation_time' :
'first','source_name' : 'first','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','segment_osrm_time' :
'sum','segment_osrm_distance' : 'sum'})

df1
```

```
df1['od_total_time'] = df1['od_end_time'] - df1['od_start_time']
df1.drop(columns = ['od_end_time', 'od_start_time'], inplace = True)
df1['od_total_time'] = df1['od_total_time'].apply(lambda x :
round(x.total_seconds() / 60.0, 2))
df1['od_total_time'].head()
```

```
df2 = df1.groupby(by = 'trip_uuid', as_index =
False).agg({'source_center' : 'first','destination_center' :
```

```
'last','data' : 'first','route_type' : 'first','trip_creation_time' :
'first','source_name' : 'first','destination_name' :
'last','od_total_time' : 'sum','start_scan_to_end_scan' :
'sum','actual_distance_to_destination' : 'sum','actual_time' :
'sum','osrm_time' : 'sum','osrm_distance' : 'sum','segment_actual_time'
: 'sum','segment_osrm_time' : 'sum','segment_osrm_distance' : 'sum'})

df2
```

Hypothesis Testing:

Compare the difference between od_total_time and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

__STEP-1__ : Set up Null Hypothesis

- **Null Hypothesis (H0)** - od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected total trip time) are same.
- **Alternate Hypothesis (HA)** - od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected total trip time) are different.

__STEP-2__ : Checking for basic assumpitons for the hypothesis

- Distribution check using **QQ Plot**
- Homogeneity of Variances using **Lavene's test**

__STEP-3__: Define Test statistics; Distribution of T under H0.

- If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

__STEP-4__: Compute the p-value and fix value of alpha.

- We set our *__alpha to be 0.05__*

__STEP-5__: Compare p-value and alpha.

- Based on p-value, we will accept or reject H0.

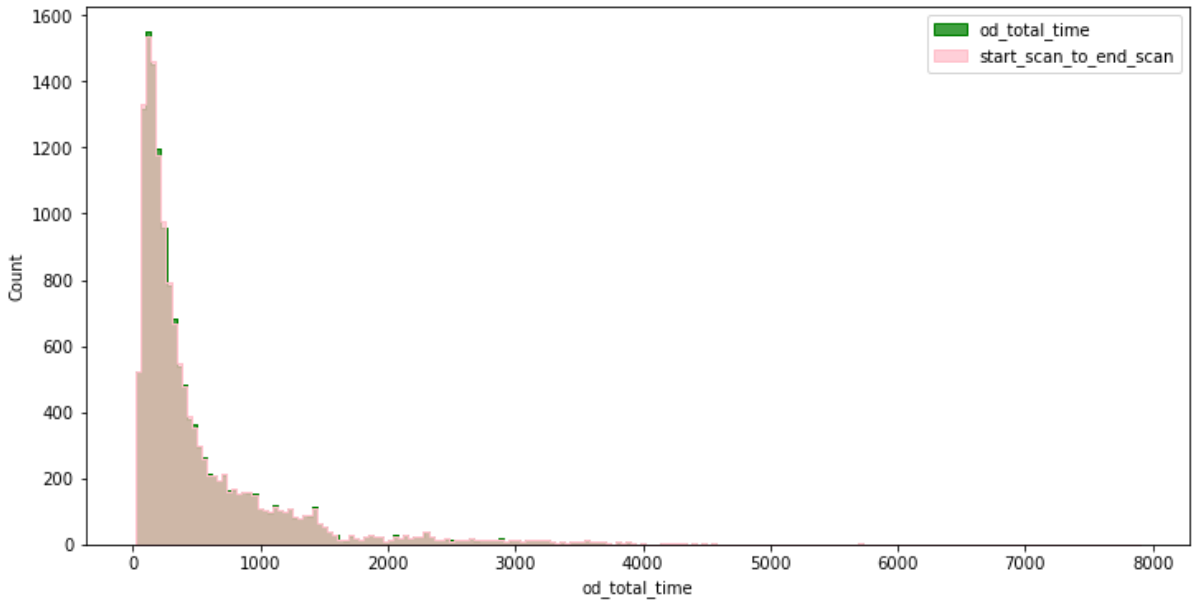
    1. **p-val > alpha** : Accept H0
    2. **p-val < alpha** : Reject H0

```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```

	od_total_time	start_scan_to_end_scan
count	14817.000000	14817.000000
mean	531.697630	530.810016
std	658.868223	658.705957
min	23.460000	23.000000
25%	149.930000	149.000000
50%	280.770000	280.000000
75%	638.200000	637.000000
max	7898.550000	7898.000000

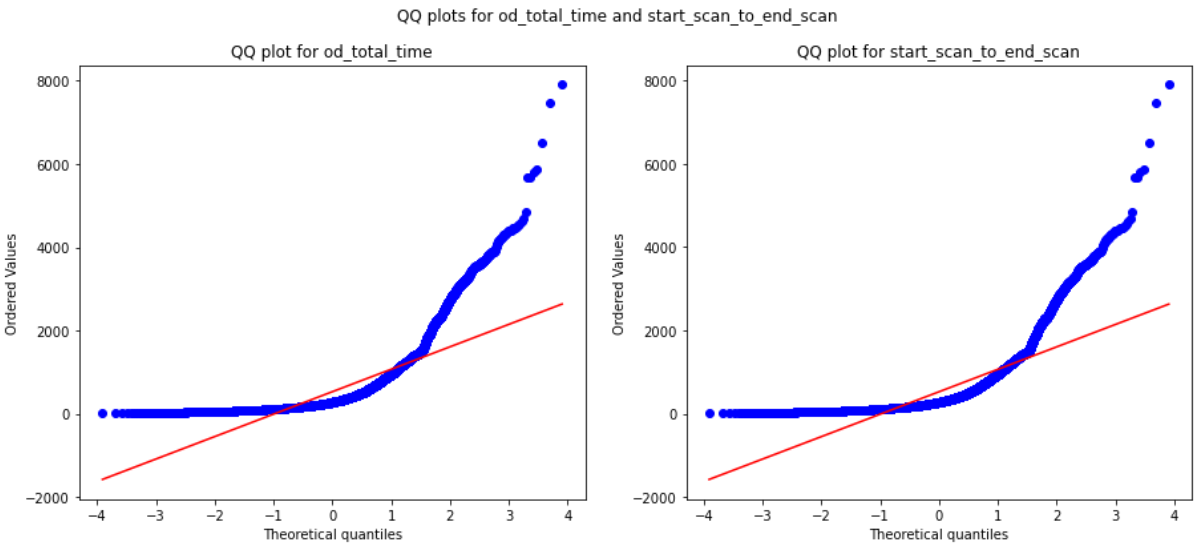
- Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (12, 6)
sns.histplot(df2['od_total_time'], element = 'step', color = 'green')
sns.histplot(df2['start_scan_to_end_scan'], element = 'step', color =
'pink')
plt.legend(['od_total_time', 'start_scan_to_end_scan'])
plt.plot()
```



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for od_total_time and start_scan_to_end_scan')
spy.probplot(df2['od_total_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for od_total_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['start_scan_to_end_scan'], plot = plt, dist = 'norm')
plt.title('QQ plot for start_scan_to_end_scan')
plt.plot()
```



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H0 : The sample **follows normal distribution** H1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df2['od_total_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value =
spy.shapiro(df2['start_scan_to_end_scan'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

```
transformed_od_total_time = spy.boxcox(df2['od_total_time'])[0]
test_stat, p_value = spy.shapiro(transformed_od_total_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 7.21300687930395e-25
The sample does not follow normal distribution
```

```
transformed_start_scan_to_end_scan =
spy.boxcox(df2['start_scan_to_end_scan'])[0]
test_stat, p_value = spy.shapiro(transformed_start_scan_to_end_scan)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 1.0378319150112312e-24
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "od_total_time" and "start_scan_to_end_scan" columns, the distributions do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['od_total_time'],
df2['start_scan_to_end_scan'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
p-value 0.9668007217581142
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
test_stat, p_value = spy.mannwhitneyu(df2['od_total_time'],
df2['start_scan_to_end_scan'])
print('P-value :',p_value)
P-value : 0.7815123224221716
```

Since p-value > alpha therfore it can be concluded that od_total_time and start_scan_to_end_scan are similar.

Do hypothesis testing / visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

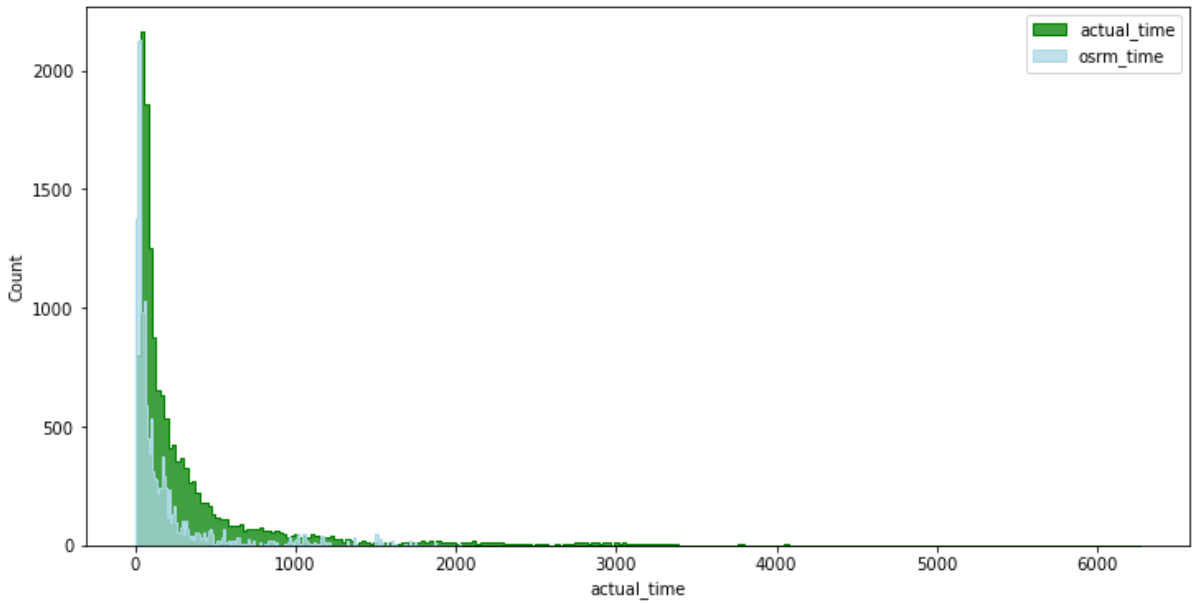
```
df2[['actual_time', 'osrm_time']].describe()
```

	actual_time	osrm_time
count	14817.000000	14817.000000
mean	357.143768	161.384018
std	561.395020	271.362549
min	9.000000	6.000000
25%	67.000000	29.000000

	actual_time	osrm_time
50%	149.000000	60.000000
75%	370.000000	168.000000
max	6265.000000	2032.000000

- Visual Tests to know if the samples follow normal distribution

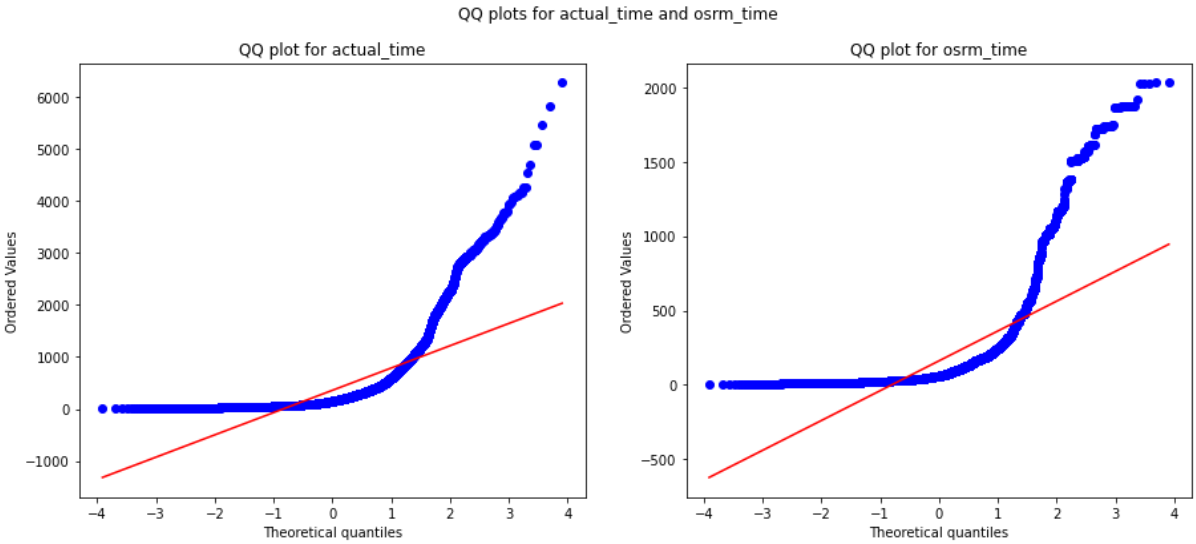
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'green')
sns.histplot(df2['osrm_time'], element = 'step', color = 'lightblue')
plt.legend(['actual_time', 'osrm_time'])
plt.plot()
```



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and osrm_time')
spy.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
```

```
plt.plot()
```



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H0 : The sample **follows normal distribution** H1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

```
transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 1.0408425976485893e-28
The sample does not follow normal distribution
```

```
transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 3.271205914895016e-35
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "actual_time" and "osrm_time" columns, the distributions do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
p-value 1.871098057987424e-220
The samples do not have  Homogenous Variance
```

Since the samples do not follow any of the assumptions T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
p-value 0.0
The samples are not similar
```

Since p-value < alpha therefore it can be concluded that actual_time and osrm_time are not similar.

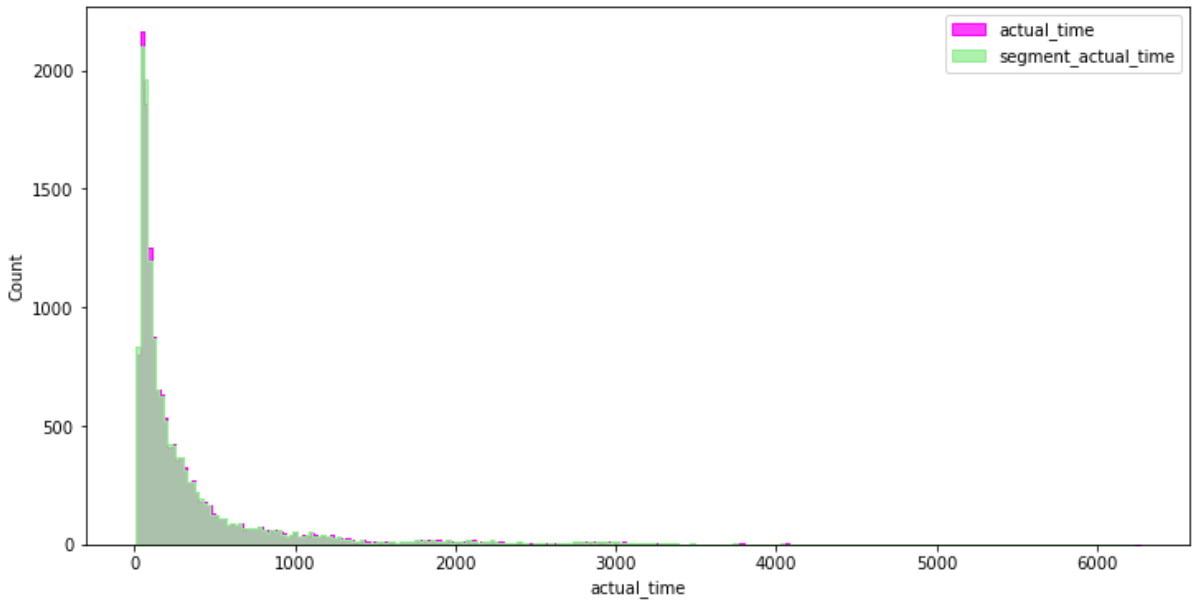
Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

```
df2[['actual_time', 'segment_actual_time']].describe()
```

	actual_time	segment_actual_time
count	14817.000000	14817.000000
mean	357.143768	353.892273
std	561.395020	556.246826
min	9.000000	9.000000
25%	67.000000	66.000000
50%	149.000000	147.000000
75%	370.000000	367.000000
max	6265.000000	6230.000000

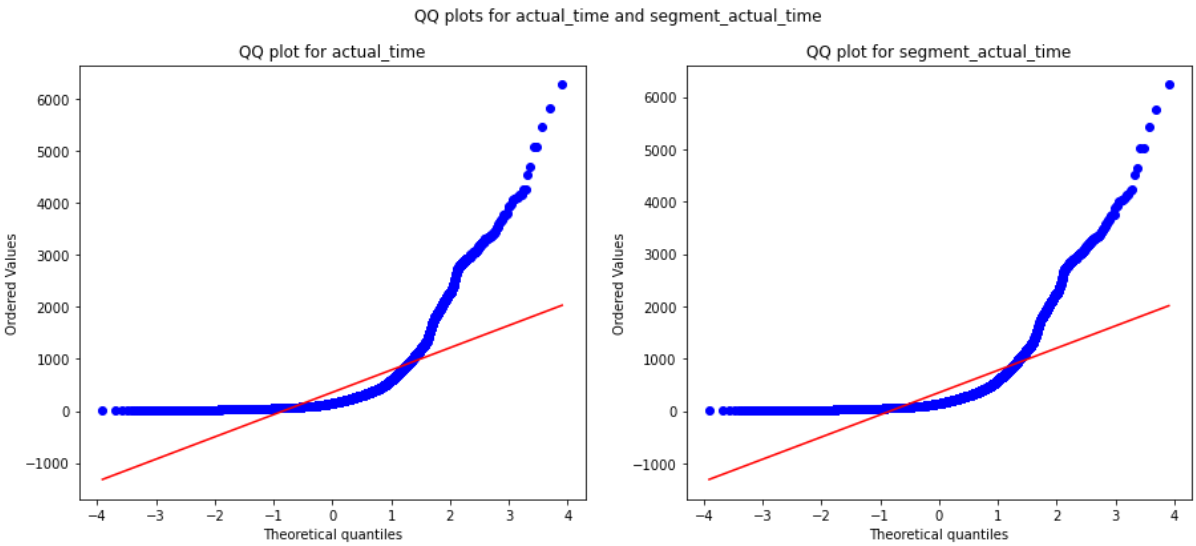
- Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'magenta')
sns.histplot(df2['segment_actual_time'], element = 'step', color =
'lightgreen')
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and segment_actual_time')
spy.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_actual_time')
plt.plot()
```



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H0: The sample **follows normal distribution** H1: The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df2['segment_actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

```
transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 1.0408425976485893e-28
The sample does not follow normal distribution
```

```
transformed_segment_actual_time = spy.boxcox(df2['segment_actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 5.676203648979465e-29
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "actual_time" and "segment_actual_time" columns, the distributions do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'],
df2['segment_actual_time'])
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
p-value 0.695502241317651
The samples have Homogenous Variance
```

Since the samples do not come from normal distribution T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'],
df2['segment_actual_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
p-value 0.4164235159622476
The samples are similar
```

Since p-value > alpha therfore it can be concluded that actual_time and segment_actual_time are similar.

Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

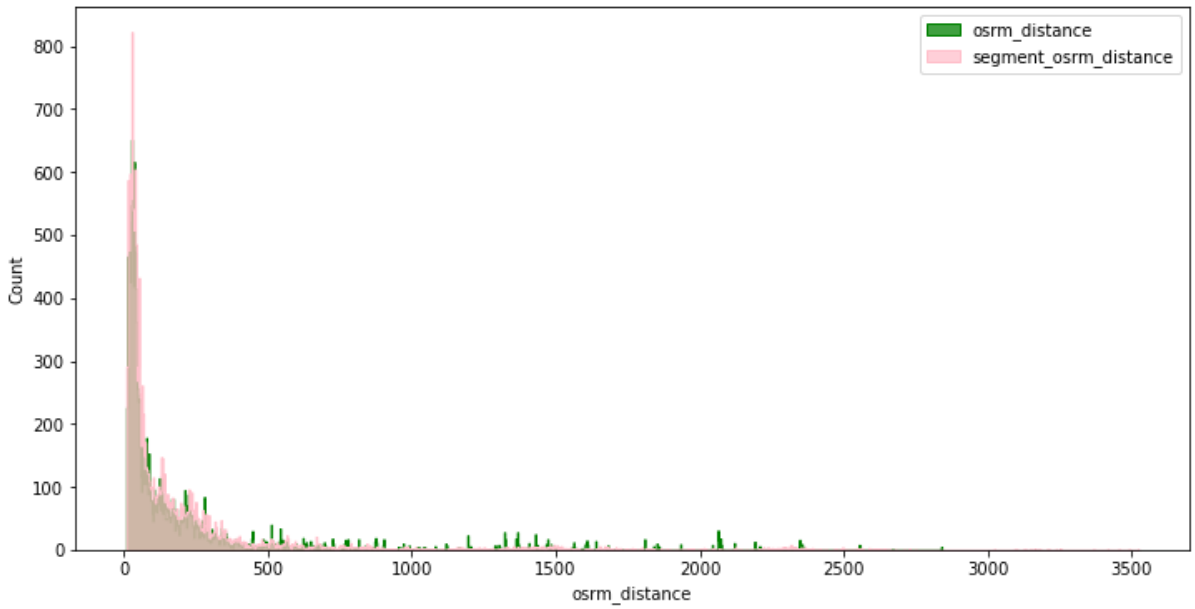
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

	osrm_distance	segment_osrm_distance
count	14817.000000	14817.000000
mean	204.345078	223.201324
std	370.395508	416.628326

	osrm_distance	segment_osrm_distance
min	9.072900	9.072900
25%	30.819201	32.654499
50%	65.618805	70.154404
75%	208.475006	218.802399
max	2840.081055	3523.632324

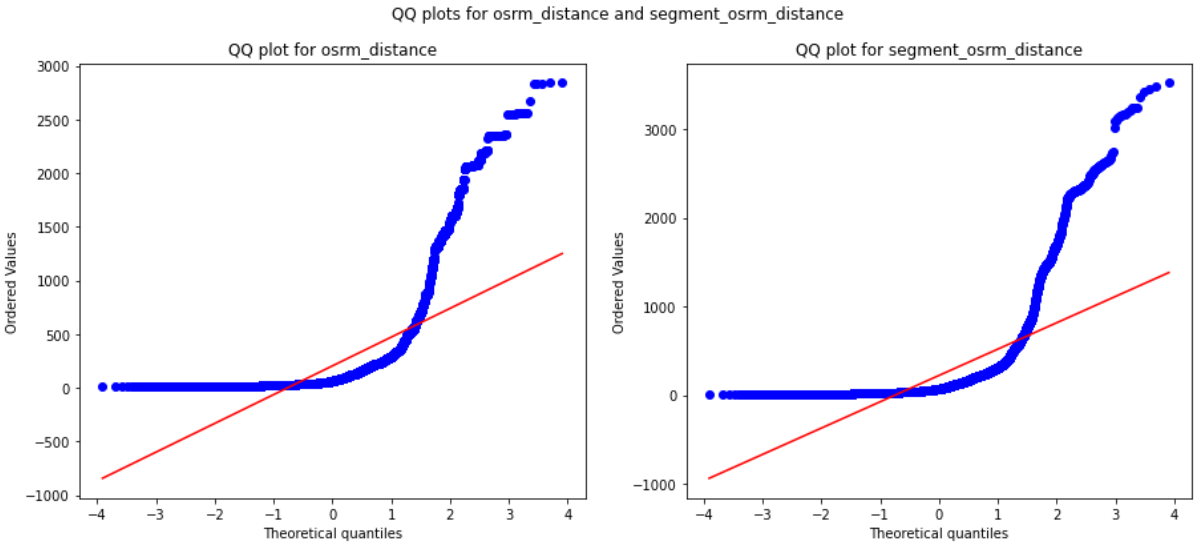
- Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_distance'], element = 'step', color = 'green', bins
= 1000)
sns.histplot(df2['segment_osrm_distance'], element = 'step', color =
'pink', bins = 1000)
plt.legend(['osrm_distance', 'segment_osrm_distance'])
plt.plot()
```



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_distance and segment_osrm_distance')
spy.probplot(df2['osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_distance')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_distance')
plt.plot()
```



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H0 : The sample **follows normal distribution** H1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df2['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df2['segment_osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
```

```
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

```
transformed_osrm_distance = spy.boxcox(df2['osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 7.069971142058e-41
The sample does not follow normal distribution
```

```
transformed_segment_osrm_distance =
spy.boxcox(df2['segment_osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 3.0555416710688996e-38
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "osrm_distance" and "segment_osrm_distance" columns, the distributions do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test.**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['osrm_distance'],
df2['segment_osrm_distance'])
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
p-value 0.00020976006524780905

The samples do not have Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non-parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_distance'],
df2['segment_osrm_distance'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
p-value 9.509312191161966e-07
The samples are not similar
```

Since p-value < alpha therfore it can be concluded that osrm_distance and segment_osrm_distance are not similar.

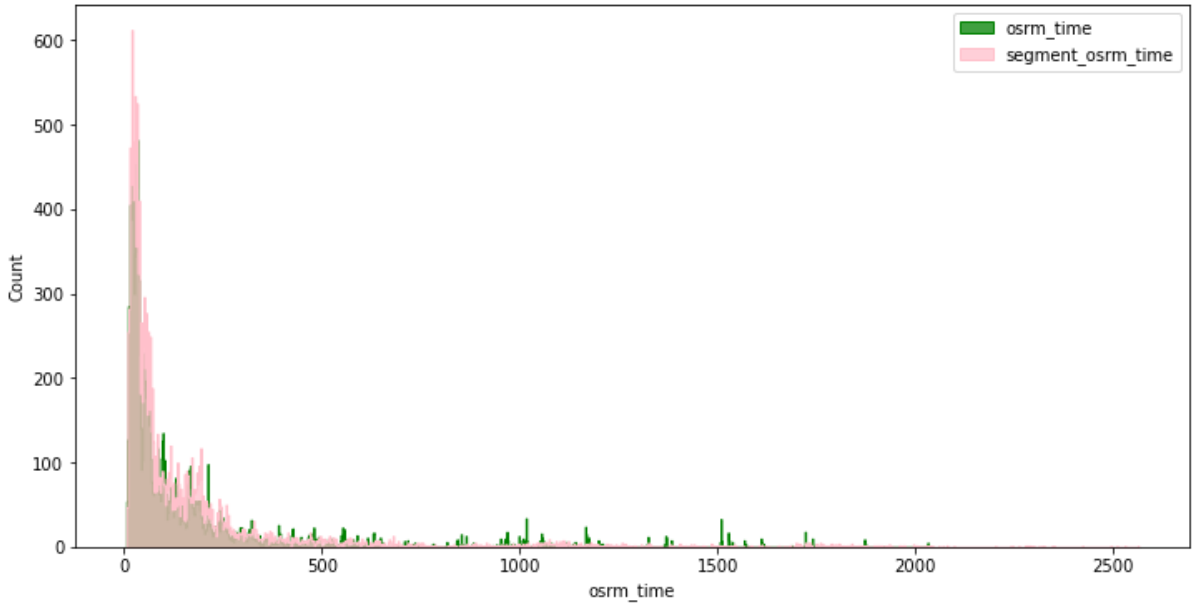
Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

```
df2[['osrm_time', 'segment_osrm_time']].describe().T
```

	count	mean	std	min	25%	50%	75%	max
osrm_time	14817.0	161.384018	271.362549	6.0	29.0	60.0	168.0	2032.0
segment_osrm_time	14817.0	180.949783	314.541412	6.0	31.0	65.0	185.0	2564.0

- Visual Tests to know if the samples follow normal distribution

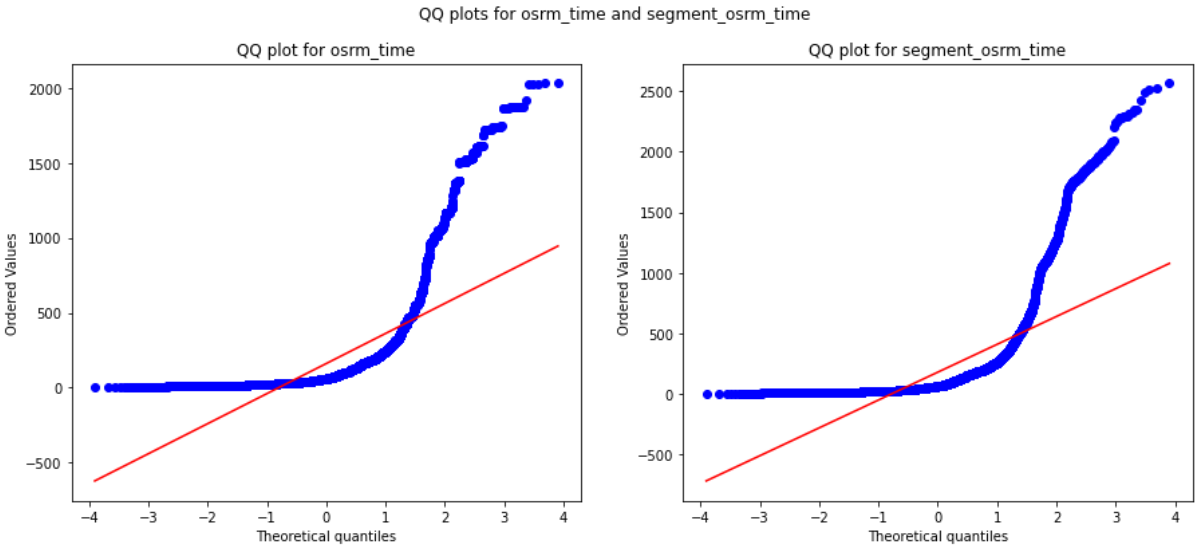
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_time'], element = 'step', color = 'green', bins =
1000)
sns.histplot(df2['segment_osrm_time'], element = 'step', color = 'pink',
bins = 1000)
plt.legend(['osrm_time', 'segment_osrm_time'])
plt.plot()
```



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_time and segment_osrm_time')
spy.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_time')
plt.plot()
```

```
[]
```



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H0 : The sample **follows normal distribution** H1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df2['segment_osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 0.0
The sample does not follow normal distribution
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

```
transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 3.271205914895016e-35
The sample does not follow normal distribution
```

```
transformed_segment_osrm_time = spy.boxcox(df2['segment_osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
p-value 4.960995746782918e-34
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "osrm_time" and "segment_osrm_time" columns, the distributions do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['osrm_time'], df2['segment_osrm_time'])
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
p-value 8.349506135727595e-08
The samples do not have Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_time'],
df2['segment_osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
p-value 2.2995370859748865e-08
The samples are not similar
```

Since p-value < alpha therfore it can be concluded that osrm_time and segment_osrm_time are not similar

Merging:

```
distances=segment_osrm_distance.merge(actual_distance_to_destination.me
rge(osrm_distance,on="trip_uuid"), on="trip_uuid")

distances
```

	trip_uuid	segment_osrm_distance	actual_distance_to_destination	osrm_distance
0	trip-153671041653548748	1320.4733	824.732854	991.3523
1	trip-153671042288605164	84.1894	73.186911	85.1110
2	trip-153671043369099517	2545.2678	1932.273969	2372.0852
3	trip-153671046011330457	19.8766	17.175274	19.6800
4	trip-153671052974046625	146.7919	127.448500	146.7918
...	...	...	...	...
14812	trip-153861095625827784	64.8551	57.762332	73.4630
14813	trip-153861104386292051	16.0883	15.513784	16.0882
14814	trip-153861106442901555	104.8866	38.684839	63.2841
14815	trip-153861115439069069	223.5324	134.723836	177.6635
14816	trip-153861118270144424	80.5787	66.081533	80.5787

14817 rows × 4 columns

```
time =
segment_osrm_time.merge(osrm_time.merge(segment_actual_time.merge(actua
l_time.merge(time_taken_btwn_odstart_and_od_end.merge(start_scan_to_end
_scan,
on="trip_uuid"),on="trip_uuid"),on="trip_uuid"),on="trip_uuid"),on="tri
p_uuid")
time
```

	trip_uuid	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odstart_and_od_end	start_scan_to_end_scan
0	trip-153671041653548748	16.800000	12.383333	25.800000	26.033333	37.668497	37.650000
1	trip-153671042288605164	1.083333	1.133333	2.350000	2.383333	3.026865	3.000000
2	trip-153671043369099517	32.350000	29.016667	55.133333	55.783333	65.572709	65.550000
3	trip-153671046011330457	0.266667	0.250000	0.983333	0.983333	1.674916	1.666667
4	trip-153671052974046625	1.916667	1.950000	5.666667	5.683333	11.972484	11.950000
...	...	...	...	...	...	...	...
14812	trip-153861095625827784	1.033333	1.033333	1.366667	1.383333	4.300482	4.283333
14813	trip-153861104386292051	0.183333	0.200000	0.350000	0.350000	1.009842	1.000000
14814	trip-153861106442901555	1.466667	0.900000	4.683333	4.700000	7.035331	7.016667
14815	trip-153861115439069069	3.683333	3.066667	4.300000	4.400000	5.808548	5.783333
14816	trip-153861118270144424	1.116667	1.133333	4.566667	4.583333	5.906793	5.883333

```
Merge1 = time.merge(distances,on="trip_uuid", )
print(Merge1)
```

	trip_uuid	segment_osrm_time	osrm_time	\
0	trip-153671041653548748	16.800000	12.383333	
1	trip-153671042288605164	1.083333	1.133333	
2	trip-153671043369099517	32.350000	29.016667	
3	trip-153671046011330457	0.266667	0.250000	
4	trip-153671052974046625	1.916667	1.950000	
...	...	...	...	
14812	trip-153861095625827784	1.033333	1.033333	
14813	trip-153861104386292051	0.183333	0.200000	
14814	trip-153861106442901555	1.466667	0.900000	
14815	trip-153861115439069069	3.683333	3.066667	
14816	trip-153861118270144424	1.116667	1.133333	

	segment_actual_time	actual_time	time_taken_btwn_odstart_and_od_end	\
0	25.800000	26.033333	37.668497	
1	2.350000	2.383333	3.026865	
2	55.133333	55.783333	65.572709	
3	0.983333	0.983333	1.674916	
4	5.666667	5.683333	11.972484	
...	...	...	...	
14812	1.366667	1.383333	4.300482	
14813	0.350000	0.350000	1.009842	
14814	4.683333	4.700000	7.035331	
14815	4.300000	4.400000	5.808548	
14816	4.566667	4.583333	5.906793	

	start_scan_to_end_scan	segment_osrm_distance	\
0	37.650000	1320.4733	
1	3.000000	84.1894	
2	65.550000	2545.2678	
3	1.666667	19.8766	
4	11.950000	146.7919	
...	...	...	
14812	4.283333	64.8551	
14813	1.000000	16.0883	
14814	7.016667	104.8866	
14815	5.783333	223.5324	
14816	5.883333	80.5787	

	actual_distance_to_destination	osrm_distance
0	824.732854	991.3523
1	73.186911	85.1110
2	1932.273969	2372.0852
3	17.175274	19.6800
4	127.448500	146.7918
...	...	...
14812	57.762332	73.4630
14813	15.513784	16.0882
14814	38.684839	63.2841
14815	134.723836	177.6635
14816	66.081533	80.5787

Merging Location details and route_type and Numerical data on TripID :

```
city =
new_df.groupby("trip_uuid")[["source_city","destination_city"]].aggrega
te({"source_city":pd.unique,"destination_city":pd.unique,})

state =
new_df.groupby("trip_uuid")[["source_state","destination_state"]].aggre
gate({"source_state":pd.unique,"destination_state":pd.unique,})

city_state =
new_df.groupby("trip_uuid")[["source_city_state","destination_city_stat
e"]].aggregate({"source_city_state":pd.unique,"destination_city_state":
pd.unique,})

locations =
city.merge(city_state.merge(state,on="trip_uuid",how="outer"),on="trip_
uuid",how="outer")

route_type =
new_df.groupby("trip_uuid")["route_type"].unique().reset_index()

Merged =
route_type.merge(locations.merge(Merge1,on="trip_uuid",how="outer"),on=
"trip_uuid",how="outer")
trip_records = Merged.copy()
```

```
trip_records["route_type"] = trip_records["route_type"].apply(lambda
x:x[0])
route_to_merge =
new_df.groupby("trip_uuid")["route_schedule_uuid"].unique().reset_index
()
trip_records =
trip_records.merge(route_to_merge,on="trip_uuid",how="outer")
trip_records["route_schedule_uuid"] =
trip_records["route_schedule_uuid"].apply(lambda x:x[0])
trip_records
```

	trip_uuid	route_type	source_city	destination_city
0	trip-153671041653548748	FTL	[Bhopal, Kanpur]	[Kanpur, Gurgaon]
1	trip-153671042288605164	Carting	[Tumkur, Doddablpur]	[Doddablpur, Chikblapur]
2	trip-153671043369099517	FTL	[Bengaluru, Gurgaon]	[Gurgaon, Chandigarh]
3	trip-153671046011330457	Carting	[Mumbai]	[Mumbai]
4	trip-153671052974046625	FTL	[Bellary, Hospet, Sandur]	[Hospet, Sandur, Bellary]
...	...	...	...	...
14812	trip-153861095625827784	Carting	[Chandigarh]	[Zirakpur, Chandigarh]
14813	trip-153861104386292051	Carting	[FBD]	[Faridabad]
14814	trip-153861106442901555	Carting	[Kanpur]	
14815	trip-153861115439069069	Carting	[Tirunelveli, Eral, Tirchchndr, Thisayanvilai,...	
14816	trip-153861118270144424	FTL	[Hospet, Sandur]	

```
                                      source_city_state  \
0          [Bhopal Madhya Pradesh, Kanpur Uttar Pradesh]
1               [Tumkur Karnataka, Doddablpur Karnataka]
2                 [Bengaluru Karnataka, Gurgaon Haryana]
3                                [Mumbai Hub Maharashtra]
4      [Bellary Karnataka, Hospet Karnataka, Sandur K...
...                                                  ...
14812         [Chandigarh Punjab, Chandigarh Chandigarh]
14813                                      [FBD Haryana]
14814                                [Kanpur Uttar Pradesh]
14815  [Tirunelveli Tamil Nadu, Eral Tamil Nadu, Tirc...
14816               [Hospet Karnataka, Sandur Karnataka]

                                 destination_city_state  \
0                [Kanpur Uttar Pradesh, Gurgaon Haryana]
1            [Doddablpur Karnataka, Chikblapur Karnataka]
2                    [Gurgaon Haryana, Chandigarh Punjab]
3                                  [Mumbai Maharashtra]
4      [Hospet Karnataka, Sandur Karnataka, Bellary K...
...                                                  ...
14812               [Zirakpur Punjab, Chandigarh Punjab]
14813                                [Faridabad Haryana]
14814                                [Kanpur Uttar Pradesh]
14815  [Eral Tamil Nadu, Tirchchndr Tamil Nadu, Thisa...
14816               [Sandur Karnataka, Bellary Karnataka]

                         source_state         destination_state  \
0      [Madhya Pradesh, Uttar Pradesh]  [Uttar Pradesh, Haryana]
1                           [Karnataka]              [Karnataka]
2                  [Karnataka, Haryana]         [Haryana, Punjab]
3                     [Hub Maharashtra]             [Maharashtra]
4                           [Karnataka]              [Karnataka]
...                                 ...                      ...
14812              [Punjab, Chandigarh]                  [Punjab]
14813                        [Haryana]                 [Haryana]
14814                    [Uttar Pradesh]           [Uttar Pradesh]
14815                       [Tamil Nadu]              [Tamil Nadu]
14816                       [Karnataka]              [Karnataka]

       segment_osrm_time  osrm_time  segment_actual_time  actual_time  \
0              16.800000  12.383333            25.800000    26.033333
1               1.083333   1.133333             2.350000     2.383333
2              32.350000  29.016667            55.133333    55.783333
3               0.266667   0.250000             0.983333     0.983333
4               1.916667   1.950000             5.666667     5.683333
...                  ...        ...                  ...          ...
14812           1.033333   1.033333             1.366667     1.383333
14813           0.183333   0.200000             0.350000     0.350000
14814           1.466667   0.900000             4.683333     4.700000
14815           3.683333   3.066667             4.300000     4.400000
14816           1.116667   1.133333             4.566667     4.583333
```

	time_taken_btwn_odstart_and_od_end	start_scan_to_end_scan	\
0	37.668497	37.650000	
1	3.026865	3.000000	
2	65.572709	65.550000	
3	1.674916	1.666667	
4	11.972484	11.950000	
...	...	...	
14812	4.300482	4.283333	
14813	1.009842	1.000000	
14814	7.035331	7.016667	
14815	5.808548	5.783333	
14816	5.906793	5.883333	

	segment_osrm_distance	actual_distance_to_destination	osrm_distance	\
0	1320.4733	824.732854	991.3523	
1	84.1894	73.186911	85.1110	
2	2545.2678	1932.273969	2372.0852	
3	19.8766	17.175274	19.6800	
4	146.7919	127.448500	146.7918	
...	...	...	...	
14812	64.8551	57.762332	73.4630	
14813	16.0883	15.513784	16.0882	
14814	104.8866	38.684839	63.2841	
14815	223.5324	134.723836	177.6635	
14816	80.5787	66.081533	80.5787	

	route_schedule_uuid
0	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...
1	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...
2	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...
3	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...
4	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...

```
trip_records.isna().sum()
```

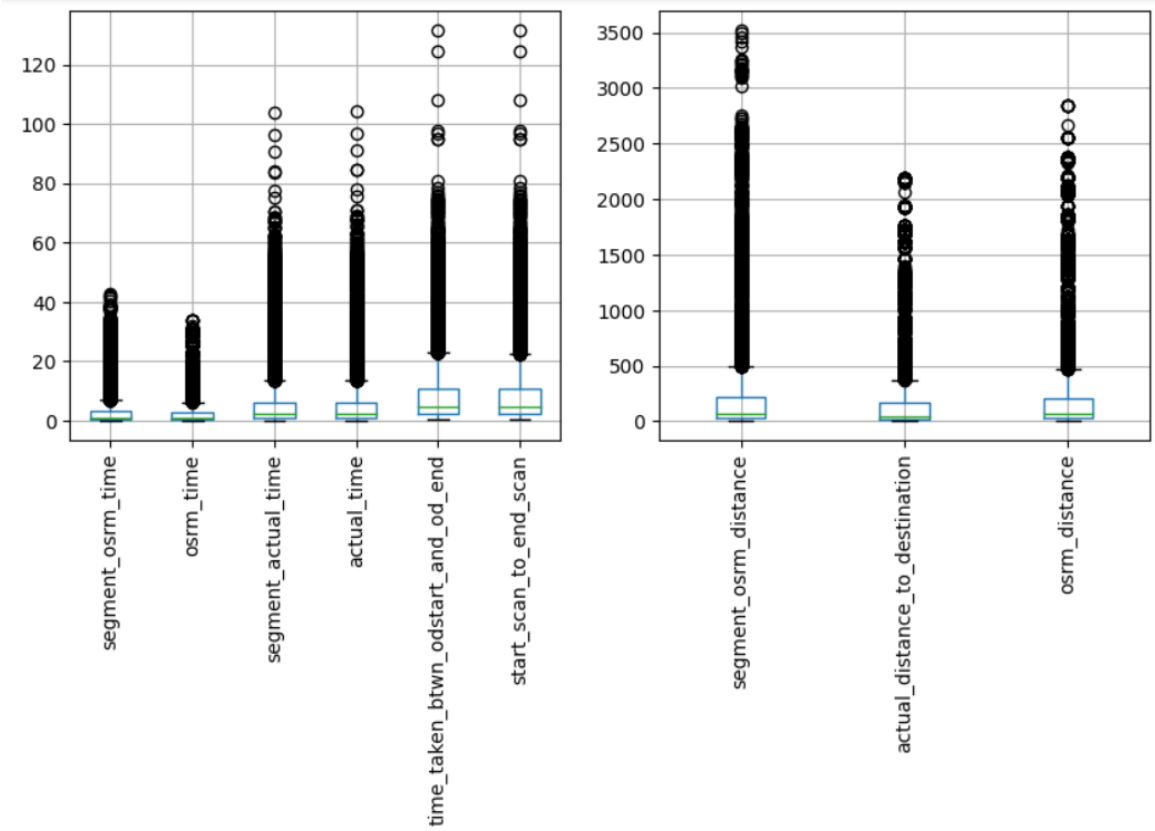
```
trip_uuid                          0
route_type                         0
source_city                        0
destination_city                   0
source_city_state                  0
destination_city_state             0
source_state                       0
destination_state                  0
segment_osrm_time                  0
osrm_time                          0
segment_actual_time                0
actual_time                        0
time_taken_btwn_odstart_and_od_end 0
start_scan_to_end_scan             0
segment_osrm_distance              0
actual_distance_to_destination     0
osrm_distance                      0
route_schedule_uuid                0
dtype: int64
```

Unnesting Data

```
trip_records["source_city"] = trip_records["source_city"].astype("str").str.st
rip("[]").str.replace("'","")
trip_records["destination_city"] = trip_records["destination_city"].astype("st
r").str.strip("[]").str.replace("'","")
trip_records["source_city_state"] = trip_records["source_city_state"].astype("
str").str.strip("[]").str.replace("'","")
trip_records["destination_city_state"] = trip_records["destination_city_state"
].astype("str").str.strip("[]").str.replace("'","")

trip_records["source_state"] = trip_records["source_state"].astype("str").str.
strip("[]").str.replace("'","")
trip_records["destination_state"] = trip_records["destination_state"].astype("
str").str.strip("[]").str.replace("'","")
```

Detecting Outliers:

```
plt.figure(figsize = (10 , 4))
plt.subplot(121)
trip_records[['segment_osrm_time', 'osrm_time',
       'segment_actual_time', 'actual_time',
       'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan']].boxplot()
plt.xticks(rotation =90)
plt.subplot(122)
trip_records[['segment_osrm_distance', 'actual_distance_to_destination',
       'osrm_distance']].boxplot()
plt.xticks(rotation =90)
plt.show()

outlier_treatment  = trip_records.copy()
```




```
outlier_treatment_num = outlier_treatment[['segment_osrm_time',
'osrm_time',
       'segment_actual_time', 'actual_time',
       'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan',
        'segment_osrm_distance', 'actual_distance_to_destination',
        'osrm_distance']]
```

Treating Outliers:

```
trip_records_without_outliers = trip_records.loc[outlier_treatment_num[(np.abs
(stats.zscore(outlier_treatment_num)) < 3).all(axis=1)].index]

trip_records_without_outliers
```

Handling Categorical Values:

Processing Data for One hot encoding:

merging locations details into one column. and re categorise the data as per highest trips having location as top category

```
trip_records_without_outliers["destination_source_locations"] = trip_records_witho
ut_outliers["source_city_state"] + " " + trip_records_without_outliers["destinatio
n_city_state"]
trip_records_without_outliers.drop(["source_city_state","destination_city_state"],
axis = 1,inplace=True)
```

```
sc_dc = trip_records_without_outliers.groupby(["destination_source_locations"])["t
rip_uuid"].nunique().sort_values(ascending= False).reset_index()
```

```
def get_cat(H):
    if 0 <= H <= 50:
        return "Category 7"
    elif 51 <= H <= 100:
        return "Category 6"
    elif 101 <= H <= 200:
        return "Category 5"
    elif 201 <= H <= 300:
        return "Category 4"
    elif 301 <= H <= 400:
        return "Category 3"
    elif 401 <= H <= 500:
        return "Category 2"
    else:
        return "Category 1"
```

```
sc_dc["city"]  = pd.Series(map(get_cat,sc_dc["trip_uuid"]))
trip_records_for_encoding = sc_dc.merge(trip_records_without_outliers,
            on="destination_source_locations")
trip_records_for_encoding.drop(["destination_source_locations","trip_uuid_x"],axis
= 1,inplace=True)
trip_records_for_encoding.drop(["trip_uuid_y"],axis = 1,inplace=True)

encoded_data = pd.get_dummies(trip_records_for_encoding,columns=["route_type",
"city"] )
encoded_data
```

Column Standardization:

```
['segment_osrm_time', 'osrm_time',
       'segment_actual_time', 'actual_time',
       'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan' ,'segment_os
rm_distance', 'actual_distance_to_destination','osrm_distance' ]


from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler



scaler = StandardScaler()
std_data = scaler.fit_transform(encoded_data[['segment_osrm_time',
 'osrm_time',
 'segment_actual_time',
 'actual_time',
 'time_taken_btwn_odstart_and_od_end',
 'start_scan_to_end_scan',
 'segment_osrm_distance',
 'actual_distance_to_destination',
 'osrm_distance']])
std_data = pd.DataFrame(std_data, columns=['segment_osrm_time',
 'osrm_time',
 'segment_actual_time',
 'actual_time',
 'time_taken_btwn_odstart_and_od_end',
 'start_scan_to_end_scan',
 'segment_osrm_distance',
 'actual_distance_to_destination',
 'osrm_distance'])

print(std_data.head())
```

```
   segment_osrm_time  osrm_time  segment_actual_time  actual_time  \
0          -0.269133  -0.409683            -0.220225    -0.214843
1          -0.359785  -0.438916            -0.324535    -0.321822
2          -0.346835  -0.402374            -0.193306    -0.194785
3          -0.534615  -0.504692            -0.597087    -0.599297
4          -0.502239  -0.533926            -0.509601    -0.509034

   time_taken_btwn_odstart_and_od_end  start_scan_to_end_scan  \
0                           -0.394178               -0.391956
1                           -0.445632               -0.444397
2                           -0.443566               -0.441900
3                           -0.318061               -0.317039
4                           -0.567441               -0.566761

   segment_osrm_distance  actual_distance_to_destination  osrm_distance
0              -0.362747                       -0.450888      -0.468190
1              -0.448864                       -0.542288      -0.521446
2              -0.416136                       -0.451494      -0.414618
3              -0.536543                       -0.516196      -0.529763
4              -0.549293                       -0.536356      -0.565995
```

```
scaler = MinMaxScaler()
```

```
MinMax_data = scaler.fit_transform(encoded_data[['segment_osrm_time','osrm_time','
segment_actual_time','actual_time',
 'time_taken_btwn_odstart_and_od_end','start_scan_to_end_scan','segment_osrm_dista
nce','actual_distance_to_destination',
 'osrm_distance']])
MinMax_data = pd.DataFrame(MinMax_data,columns=['segment_osrm_time',
 'osrm_time','segment_actual_time','actual_time','time_taken_btwn_odstart_and_od_e
nd','start_scan_to_end_scan',
 'segment_osrm_distance','actual_distance_to_destination','osrm_distance'])
MinMax_data.head()
```

```
print(std_data)
```

```
       segment_osrm_time  osrm_time  segment_actual_time  actual_time  \
0              -0.269133  -0.409683            -0.220225    -0.214843
1              -0.359785  -0.438916            -0.324535    -0.321822
2              -0.346835  -0.402374            -0.193306    -0.194785
3              -0.534615  -0.504692            -0.597087    -0.599297
4              -0.502239  -0.533926            -0.509601    -0.509034
...                  ...        ...                  ...          ...
14155          -0.042502   0.043440            -0.210131    -0.211500
14156          -0.230282  -0.197738            -0.314441    -0.311792
14157           0.326583   0.430787             0.136448     0.136179
14158           0.507888   0.635424             1.347789     1.336342
14159          -0.495764  -0.468150            -0.435575    -0.435486

       time_taken_btwn_odstart_and_od_end  start_scan_to_end_scan  \
0                               -0.394178               -0.391956
1                               -0.445632               -0.444397
2                               -0.443566               -0.441900
3                               -0.318061               -0.317039
4                               -0.567441               -0.566761
...                                   ...                     ...
14155                           -0.123651               -0.124754
14156                           -0.211977               -0.212156
14157                            0.104495                0.104990
14158                            1.031740                1.033953
14159                           -0.732338               -0.731577

       segment_osrm_distance  actual_distance_to_destination  osrm_distance
0                  -0.362747                       -0.450888      -0.468190
1                  -0.448864                       -0.542288      -0.521446
2                  -0.416136                       -0.451494      -0.414618
3                  -0.536543                       -0.516196      -0.529763
4                  -0.549293                       -0.536356      -0.565995
...                      ...                             ...            ...
14155               0.107675                        0.278861       0.169418
14156              -0.123317                       -0.165131      -0.097018
14157               0.360386                        0.434538       0.512552
14158               0.662356                        0.578210       0.760187
14159              -0.439871                       -0.392780      -0.425349

[14160 rows x 9 columns]
```

```
one_hot_encoded_data = encoded_data[["route_type_Carting","route_type_FTL","city_C
ategory 1",
```

```
 "city_Category 2","city_Category 3","city_Category 4",
 "city_Category 5","city_Category 6","city_Category 7"]]


Standardized_Data = pd.concat([std_data,one_hot_encoded_data],axis = 1)


Min_Max_Scaled_Data = pd.concat([MinMax_data,one_hot_encoded_data],axis = 1)


Standardized_Data.sample(5)
Min_Max_Scaled_Data.sample(5)
```

Route analysis:

```
A = new_df.groupby("route_schedule_uuid")["route_type"].unique().reset_index()
B = new_df.groupby("route_schedule_uuid")["destination_city"].unique().reset_index
()
B.columns = ["route_schedule_uuid","destination_cities"]
C = new_df.groupby("route_schedule_uuid")["source_city"].unique().reset_index()
C.columns = ["route_schedule_uuid","source_cities"]
D = new_df.groupby("route_schedule_uuid")["source_state"].unique().reset_index()
D.columns = ["route_schedule_uuid","source_states"]
E = new_df.groupby("route_schedule_uuid")["destination_state"].unique().reset_inde
x()
E.columns = ["route_schedule_uuid","destination_states"]
F = new_df.groupby("route_schedule_uuid")[["source_state",
                                           "destination_state"]].nunique().sort_valu
es(by="source_state",

ascending=False).reset_index()
F.columns = ["route_schedule_uuid","#source_states"
             ,"#destination_states"]
G = trip_records.groupby("route_schedule_uuid")["actual_distance_to_destination"].
mean().reset_index()
G.columns = ["route_schedule_uuid","Average_Actual_distance_to_destination"]
H = trip_records["route_schedule_uuid"].value_counts().reset_index()
H.columns = ["route_schedule_uuid","Number_of_Trips"]
I = new_df.groupby("route_schedule_uuid")[["source_city","destination_city"]].nuni
que().sort_values(by="source_city",
ascending=False).reset_index()
I.columns = ["route_schedule_uuid","#source_cities"
             ,"#destination_cities"]


route_records = I.merge(H.merge(G.merge(F.merge(E.merge(D.merge(C.merge(A.merge(B,
        on ="route_schedule_uuid",
        how = "outer"),on ="route_schedule_uuid",
        how = "outer"),
       on ="route_schedule_uuid",
        how = "outer"),
       on ="route_schedule_uuid",
        how = "outer"),
```

```
        on ="route_schedule_uuid",
         how = "outer"),
        on ="route_schedule_uuid",
         how = "outer"),
        on ="route_schedule_uuid",
         how = "outer"),on ="route_schedule_uuid",
         how = "outer")
```

```
route_records.isna().sum()
```

```
route_schedule_uuid                       0
#source_cities                            0
#destination_cities                       0
Number_of_Trips                           0
Average_Actual_distance_to_destination    0
#source_states                            0
#destination_states                       0
destination_states                        0
source_states                             0
source_cities                             0
route_type                                0
destination_cities                        0
dtype: int64
```

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

```
route_records["route_type"] = route_records["route_type"].astype("str").str.strip(
"[]").str.replace("'","")
route_records["source_cities"] = route_records["source_cities"].astype("str").str.
strip("[]").str.replace("'","")
route_records["destination_cities"] = route_records["destination_cities"].astype("
str").str.strip("[]").str.replace("'","")
route_records["source_states"] = route_records["source_states"].astype("str").str.
strip("[]").str.replace("'","")

route_records["destination_states"] = route_records["destination_states"].astype("
str").str.strip("[]").str.replace("'","")

route_records
```

```
route_records["ROUTE"] = route_records["source_cities"] + " -- " + route_records["
destination_cities"]
route_records.drop(["route_schedule_uuid"],axis = 1,inplace=True)
first_column = route_records.pop('ROUTE')
route_records.insert(0, 'ROUTE', first_column)
route_records["SouceToDestination_city"] = route_records["source_cities"].str.spli
t(" ").apply(lambda x:x[0]) +" TO " +route_records["destination_cities"].str.split
(" ").apply(lambda x:x[-1])
first_column = route_records.pop('SouceToDestination_city')
route_records.insert(0, 'SouceToDestination_city', first_column)

route_records
```

Exploratory Data Analysis: (getting some insights from preprocessed data):

Busiest Route Analysis:

Number of Trips between cities, sorted highest to lowest

Top 20 source and destination cities which have high frequency of trips in between.

```
Number_of_trips_between_cities = data.groupby(["source_city_state",
                                               "destination_city_state"])["trip_uu
id"].nunique().sort_values(ascending=False).reset_index()
Number_of_trips_between_cities.head(25)
```

	source_city_state	destination_city_state	trip_uuid
0	Bengaluru Karnataka	Bengaluru Karnataka	1369
1	Bhiwandi Maharashtra	Mumbai Maharashtra	512
2	Mumbai Maharashtra	Mumbai Maharashtra	361
3	Hyderabad Telangana	Hyderabad Telangana	308
4	Mumbai Maharashtra	Bhiwandi Maharashtra	282
5	Delhi Delhi	Gurgaon Haryana	248
6	Gurgaon Haryana	Delhi Delhi	237
7	Mumbai Hub Maharashtra	Mumbai Maharashtra	227
8	Chennai Tamil Nadu	Chennai Tamil Nadu	205
9	MAA Tamil Nadu	Chennai Tamil Nadu	204
10	Chennai Tamil Nadu	MAA Tamil Nadu	141
11	Bengaluru Karnataka	HBR Karnataka	133
12	Ahmedabad Gujarat	Ahmedabad Gujarat	131
13	Pune Maharashtra	PNQ Maharashtra	122
14	Jaipur Rajasthan	Jaipur Rajasthan	111
15	Delhi Delhi	Delhi Delhi	109
16	Pune Maharashtra	Bhiwandi Maharashtra	107
17	Pune Maharashtra	Pune Maharashtra	101
18	Chandigarh Chandigarh	Chandigarh Punjab	100
19	Kolkata West Bengal	CCU West Bengal	96
20	Gurgaon Haryana	Sonipat Haryana	92
21	Sonipat Haryana	Gurgaon Haryana	86
22	Chandigarh Punjab	Chandigarh Chandigarh	84
23	HBR Karnataka	Bengaluru Karnataka	79
24	Bengaluru Karnataka	BLR Karnataka	78

- **From above table, we can observe that Mumbai Maharashtra, Delhi, Gurgaon(Haryana), Bengaluru Karnataka ,Hyderabad Telangana, Chennai**

Tamil Nadu, Ahmedabad Gujarat, Pune Maharashtra, Chandigarh Chandigarh and Kolkata West Bengal are some cities have highest amount of trips happening states with in the city:

```
Number_of_trips_between_cities.loc[Number_of_trips_between_cities["source_city_state"] != Number_of_trips_between_cities["destination_city_state"]].reset_index(drop = True).head(25)
```

	source_city_state	destination_city_state	trip_uuid
0	Bhiwandi Maharashtra	Mumbai Maharashtra	512
1	Mumbai Maharashtra	Bhiwandi Maharashtra	282
2	Delhi Delhi	Gurgaon Haryana	248
3	Gurgaon Haryana	Delhi Delhi	237
4	Mumbai Hub Maharashtra	Mumbai Maharashtra	227
5	MAA Tamil Nadu	Chennai Tamil Nadu	204
6	Chennai Tamil Nadu	MAA Tamil Nadu	141
7	Bengaluru Karnataka	HBR Karnataka	133
8	Pune Maharashtra	PNQ Maharashtra	122
9	Pune Maharashtra	Bhiwandi Maharashtra	107
10	Chandigarh Chandigarh	Chandigarh Punjab	100
11	Kolkata West Bengal	CCU West Bengal	96
12	Gurgaon Haryana	Sonipat Haryana	92
13	Sonipat Haryana	Gurgaon Haryana	86
14	Chandigarh Punjab	Chandigarh Chandigarh	84
15	HBR Karnataka	Bengaluru Karnataka	79

16	Bengaluru Karnataka	BLR Karnataka	78
17	Del Delhi	Gurgaon Haryana	76
18	Bhiwandi Maharashtra	Pune Maharashtra	72
19	Ludhiana Punjab	Chandigarh Punjab	71
20	Chandigarh Punjab	Gurgaon Haryana	66
21	Gurgaon Haryana	Bengaluru Karnataka	66
22	LowerParel Maharashtra	Mumbai Maharashtra	65
23	Mumbai Hub Maharashtra	Bhiwandi Maharashtra	63
24	PNQ Maharashtra	Pune Maharashtra	62

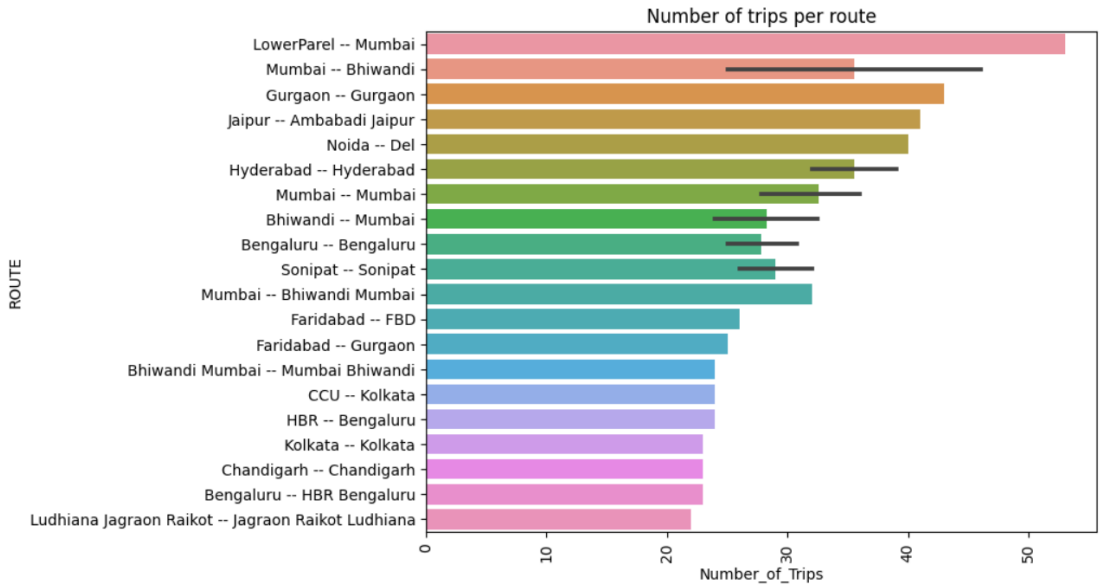
If we talk about, not having equal source and destination states, source and destination cities having highest number of trips in between are:

- Delhi to Gurgaon.
- Gurgaon Haryana to Bengaluru Karnataka.
- Bhiwandi/Mumbai, Maharashtra to Pune Maharashtra.
- Sonipat to Gurgaon, Haryana.
    - it is also been observed that lots of deliveries are happening to airports.
    - like: Chennai to MAA Chennai international Airport, Pune to Pune Airport (PNQ), Kolkata to CCU West Bengal Kolkata International Airport, Bengaluru to BLR-Bengaluru International Airport etc.

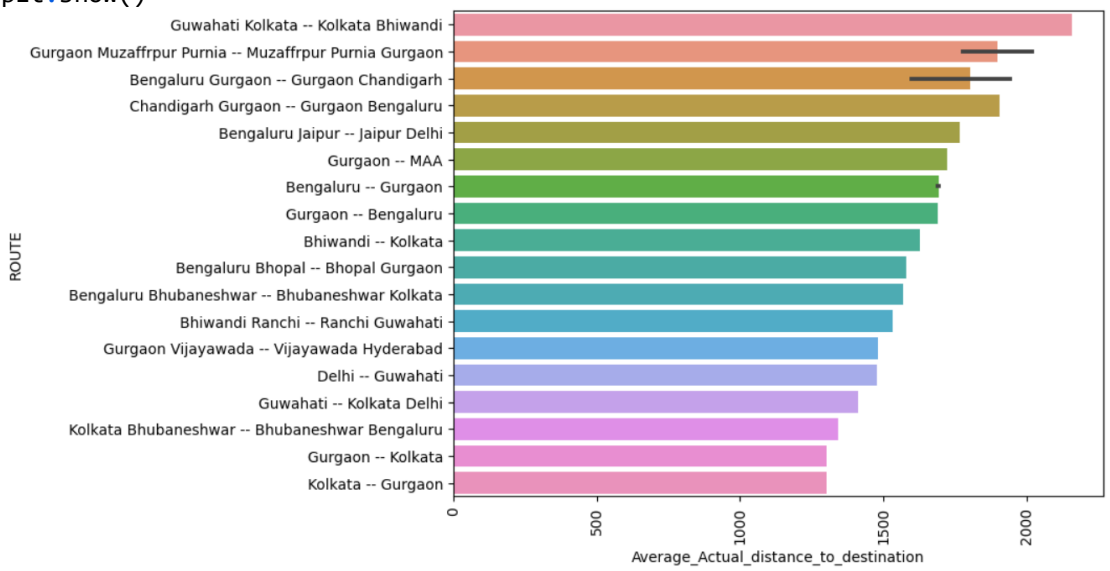
```
route_records[["ROUTE","Number_of_Trips",
               "Average_Actual_distance_to_destination",
               "#source_cities",
               "#destination_cities"]].sort_values(by="Number_of_Trips",ascending=
False).head(25)
```

Top Routes having Maximum Number of Trips between/within the source and destinations.

```
plt.figure(figsize=(8,6))
X = route_records[["ROUTE", "Number_of_Trips",
]].sort_values(by="Number_of_Trips",ascending=False).head(35)
sns.barplot(y = X["ROUTE"], x= X["Number_of_Trips"])
plt.title("Number of trips per route")
plt.xticks(rotation = 90)
plt.show()
```




```
plt.figure(figsize=(8,6))
X = route_records[["ROUTE", "Average_Actual_distance_to_destination",]].sort_value
s(by="Average_Actual_distance_to_destination",ascending=False).head(25)
sns.barplot(y = X["ROUTE"],x = X["Average_Actual_distance_to_destination"])
plt.xticks(rotation = 90)
plt.show()
```



- From above Bar chart , and table , we can observe that higest trips are happening is within the particular cities.

- in terms of average distnace between destinations, we can observe Guwahati to Mumbai , Benglore to Chandigarh ,Benglore to Delhi , Benglore to Gurgaon are the longest routes .

Busiest and Longest Routes:

```
Busiest_and_Longest_Routes = route_records[(route_records["Average_Actual_distance_to_destination"] > route_records["Average_Actual_distance_to_destination"].quantile(0.75)) & (route_records["Number_of_Trips"] > route_records["Number_of_Trips"].quantile(0.75))].sort_values(by="Average_Actual_distance_to_destination",ascending=False)
Busiest_and_Longest_Routes_top25 = Busiest_and_Longest_Routes[["source_cities","destination_citie,"Number_of_Trips","Average_Actual_distance_to_destination"]].head(25)
```

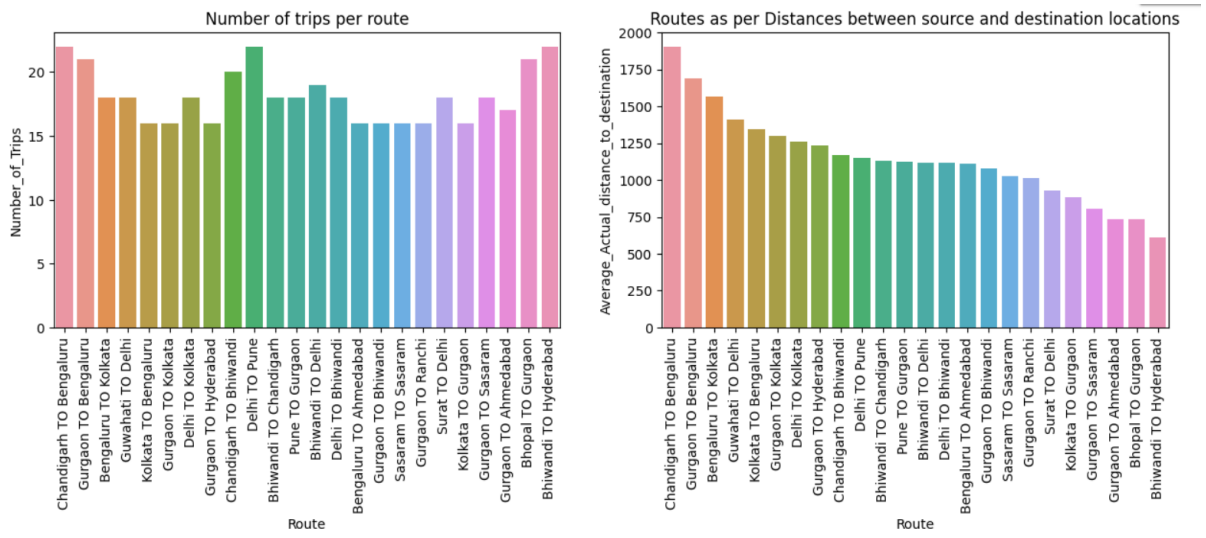
Busiest_and_Longest_Routes_top25

	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destination
629	Chandigarh Gurgaon	Gurgaon Bengaluru	22	1905.766051
995	Gurgaon	Bengaluru	21	1689.873158
991	Gurgaon	Bengaluru	21	1689.791894
512	Bengaluru Bhubaneshwar	Bhubaneshwar Kolkata	18	1567.577507
745	Guwahati	Kolkata Delhi	18	1411.208424
624	Kolkata Bhubaneshwar	Bhubaneshwar Bengaluru	16	1342.143081
752	Gurgaon	Kolkata	16	1300.572161
588	Delhi Gurgaon	Gurgaon Kolkata	18	1263.113211
826	Gurgaon	Hyderabad	16	1236.572072
541	Chandigarh Gurgaon	Gurgaon Bhiwandi	20	1170.817927
442	Delhi Gurgaon	Gurgaon Pune	22	1151.514940
445	Bhiwandi Sonipat	Sonipat Chandigarh	18	1129.609705
739	Pune	Gurgaon	18	1120.729446
1377	Bhiwandi	Delhi	19	1114.214670
1049	Delhi	Bhiwandi	18	1114.182197
313	Bengaluru Kolhapur Surat	Kolhapur Surat Ahmedabad	16	1110.015339
1219	Gurgaon	Bhiwandi	16	1078.076312
197	Sasaram Kanpur Kolkata Dhanbad	Kanpur Gurgaon Dhanbad Sasaram	16	1028.024726
1136	Gurgaon	Ranchi	16	1010.953223
1286	Surat	Delhi	18	931.980821
439	Kolkata Ranchi	Ranchi Gurgaon	16	881.621264
1108	Gurgaon	Sasaram	18	804.210670
1454	Gurgaon	Ahmedabad	17	735.550450
223	Bhopal Kanpur Auraiya Etawah	Kanpur Auraiya Etawah Gurgaon	21	731.634456
863	Bhiwandi	Hyderabad	22	607.514619

Above Table shows the source to destination city routes having largest numbers of trip happening having large distances: which are:

- Chandigarh TO Bengaluru
- Gurgaon TO Bengaluru
- Bengaluru TO Kolkata
- Guwahati TO Delhi
- Delhi TO Kolkata
- Chandigarh TO Gurgaon
- Gurgaon TO Hyderabad
- Bengaluru TO Ahmedabad
- Surat TO Delhi
- Gurgaon TO Ahmedabad

```
Busiest_and_Longest_Routes_top25["Route"] = Busiest_and_Longest_Routes_top25["sour
ce_cities"].str.split(" ").apply(lambda x:x[0]) + " TO " + Busiest_and_Longest_Rou
tes_top25["destination_cities"].str.split(" ").apply(lambda x:x[-1])
Busiest_and_Longest_Routes_top25.drop(["source_cities","destination_cities"],axis
= 1,inplace=True)
plt.figure(figsize=(15,4))

plt.subplot(121)
plt.title("Number of trips per route")
sns.barplot(x=Busiest_and_Longest_Routes_top25["Route"],
            y = Busiest_and_Longest_Routes_top25["Number_of_Trips"])
plt.xticks(rotation = 90)
plt.subplot(122)
plt.title("Routes as per Distances between source and destination locations")
sns.barplot(x=Busiest_and_Longest_Routes_top25["Route"],
            y= Busiest_and_Longest_Routes_top25["Average_Actual_distance_to_destina
tion"])
plt.xticks(rotation = 90)
plt.show()
```



Routes: passing through maximum number of cities:

```
route_records[["SouceToDestination_city","Number_of_Trips",
               "Average_Actual_distance_to_destination",
               "#source_cities",
               "#destination_cities"]].sort_values(by=["#source_cities","#destinat
ion_cities","Number_of_Trips"],ascending=False).head(25)
```

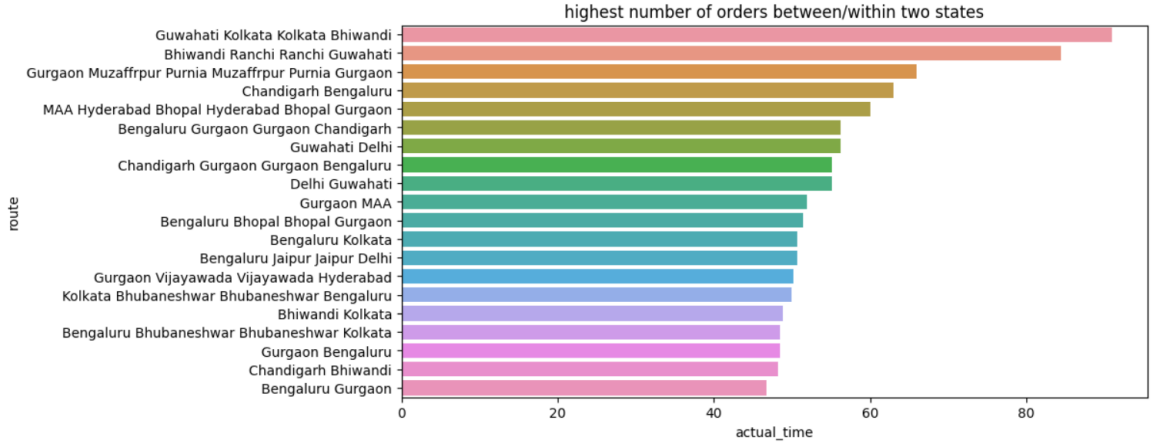
	SouceToDestination_city	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities	#destination_cities
0	Guwahati TO LakhimpurN	14	281.596486	13	11
2	Jaipur TO Tarnau	20	351.611796	10	10
1	Guwahati TO Tura	12	332.602225	10	10
3	Mangalore TO Udupi	9	195.257193	9	9
4	Ajmer TO Raipur	20	178.737233	9	8
5	Mainpuri TO Tilhar	12	207.247057	8	8
8	Hassan TO Koppa	21	200.497832	7	7
15	Shrirampur TO Sangamner	20	204.509529	7	7
7	Musiri TO Tiruchi	19	219.845121	7	7
9	Bijnor TO Bijnor	17	209.400685	7	7
10	Dausa TO Lalsot	17	232.408310	7	7
17	Tinusukia TO Dibrugarh	16	111.098543	7	7
12	Pondicherry TO Pondicherry	12	230.253602	7	7
14	Mysore TO Mysore	12	154.324190	7	7
6	Golaghat TO Guwahati	11	258.546587	7	7
13	Varanasi TO Varanasi	8	82.545019	7	7
16	Vijayawada TO Suryapet	8	407.029391	7	7
11	Hyderabad TO Miryalguda	7	420.603709	7	7
27	Srikakulam TO Bobbili	22	154.495283	6	6
36	Pukhrayan TO Kanpur	22	139.834945	6	6
48	Dhule TO Shirpur	22	150.016233	6	6
30	Madhupur TO Madhupur	21	252.072259	6	6
38	Kamareddy TO Kamareddy	21	177.923330	6	6
42	Noida TO Khurja	21	208.714043	6	6
20	Junagadh TO Veraval	19	179.538596	6	6

Top 20 Longest Route as per: average actual time taken from one city to another city:

```
Longest_route_as_per_actual_trip_time = trip_records.groupby(["source_city",
                      "destination_city"])["actual_time"].mean().sort_values(ascen
ding=False).head(20).reset_index()

Longest_route_as_per_actual_trip_time["route"] = Longest_route_as_per_actual_trip_
time["source_city"] + " " + Longest_route_as_per_actual_trip_time["destination_cit
y"]
```

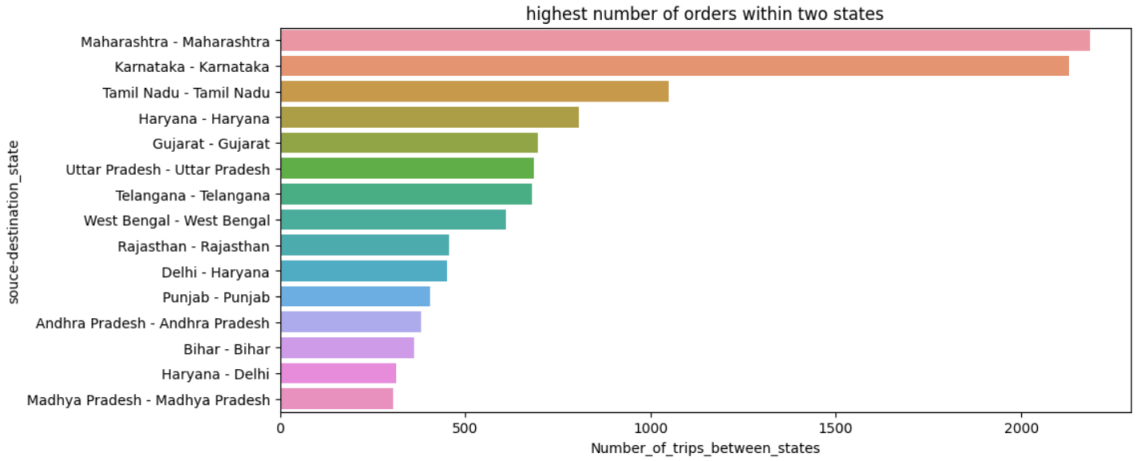
```
Longest_route_as_per_actual_trip_time.drop(["source_city", "destination_city"],axi
s = 1,inplace=True)

Longest_route_as_per_actual_trip_time
```

```
plt.figure(figsize=(10,4))
sns.barplot(y = Longest_route_as_per_actual_trip_time["route"],
            x = Longest_route_as_per_actual_trip_time["actual_time"],)
plt.title("highest number of orders between/within two states")
plt.show()
```

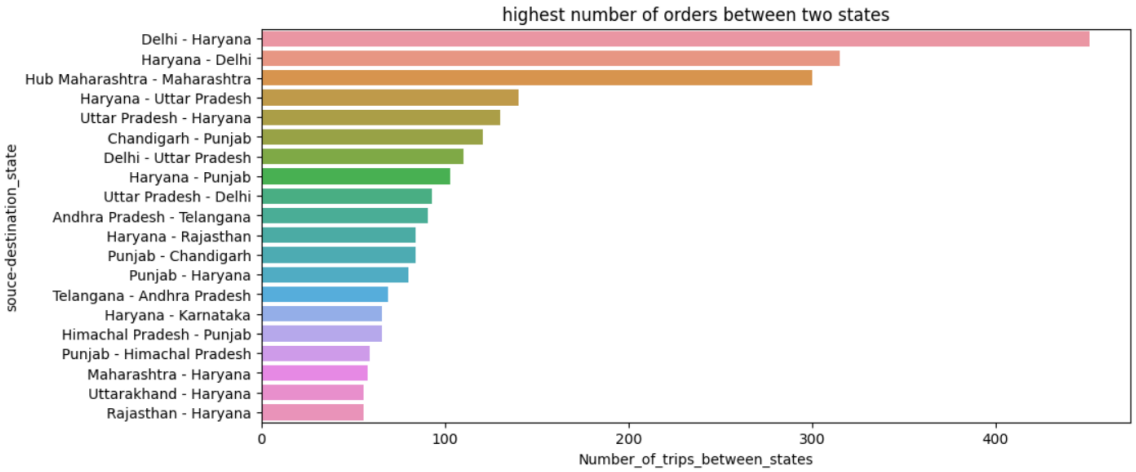


highest number of Trips happening between/within two states:

```
highest_order_between_states = new_df.groupby(["source_state","destination_state"]
)["trip_uuid"].nunique().sort_values(ascending=False).reset_index()

HOBS  = highest_order_between_states.head(15)

HOBS["souce-destination"] = HOBS["source_state"] + " - " + HOBS["destination_state
"]
HOBS.drop(["source_state","destination_state"],axis = 1, inplace=True)
HOBS.columns = ["Number_of_trips_between_states","souce-destination_state"]

plt.figure(figsize=(11,5))
sns.barplot(y = HOBS["souce-destination_state"],
            x = HOBS["Number_of_trips_between_states"],)
plt.title("highest number of orders within two states")
plt.show()
```




```
HOBS = new_df.groupby(["source_state","destination_state"])["trip_uuid"].nunique()
.sort_values(ascending=False).reset_index()
HOBS = HOBS[HOBS["source_state"]!=HOBS["destination_state"]].head(20)

HOBS["souce-destination"] = HOBS["source_state"] + " - " + HOBS["destination_state
"]
HOBS.drop(["source_state","destination_state"],axis = 1, inplace=True)
HOBS.columns = ["Number_of_trips_between_states","souce-destination_state"]

plt.figure(figsize=(11,5))
sns.barplot(y = HOBS["souce-destination_state"],
            x = HOBS["Number_of_trips_between_states"],)
plt.title("highest number of orders between two states")
plt.show()
```



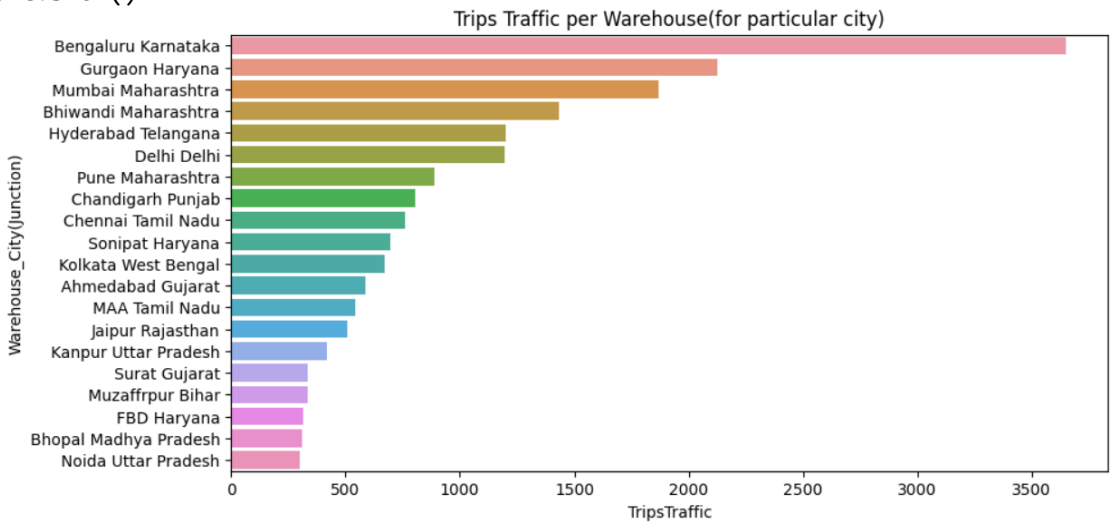
From above charts:

> Delhi to Haryana is the busiest route, having more than 400 trips in between . Some of such busy routes are Haryana to Uttar Pradesh, Chandigarh to Punjab, Delhi to Uttar Pradesh.

> Within the state Maharashtra, Karnataka, Tamil Nadu are some states having a bove 1000 trips.

Top 20 warehouses with heavy traffic:

```
destination_traffic = new_df.groupby(["destination_city_state"])["trip_uuid"].nuni
que().reset_index()
source_traffic = new_df.groupby(["source_city_state"])["trip_uuid"].nunique().rese
t_index()
transactions = source_traffic.merge(destination_traffic,left_on ="source_city_stat
e",right_on="destination_city_state")
transactions.columns = ["source_city_state","#Trips_s","destination_city_state","#
Trips_d"]
transactions["TripsTraffic"] = transactions["#Trips_s"]+transactions["#Trips_d"]
transactions.drop(["#Trips_s","#Trips_d","destination_city_state"],axis = 1,inplac
e=True)
transactions.columns = ["Warehouse_City(Junction)","TripsTraffic"]
T = transactions.sort_values(by=["TripsTraffic"],ascending=False).head(20)

plt.figure(figsize=(11,8))
sns.barplot(y = T["Warehouse_City(Junction)"],x = T["TripsTraffic"])
plt.title("Trips Traffic per Warehouse(for particular city)")
plt.show()
```



Top 20 Busiest Warehouse (junctions) as per trips traffic at the juction : are

- 'Bengaluru Karnataka',
- 'Gurgaon Haryana',
- 'Mumbai Maharashtra',
- 'Bhiwandi Maharashtra',
- 'Hyderabad Telangana',
- 'Delhi Delhi',
- 'Pune Maharashtra',
- 'Chandigarh Punjab', -
- 'Chennai Tamil Nadu',
- 'Sonipat Haryana', -
- 'Kolkata West Bengal',

- 'Ahmedabad Gujarat',
- 'MAA Tamil Nadu',
- 'Jaipur Rajasthan',
- 'Kanpur Uttar Pradesh', -
- 'Surat Gujarat',
- 'Muzaffrpur Bihar',
- 'FBD Haryana',
- 'Bhopal Madhya Pradesh',
- 'Noida Uttar Pradesh'

```
trip_records.groupby(["source_state","destination_state"])["trip_uuid"].count().so
rt_values(ascending=False).head(15).reset_index()
```

Insights:

- 14817 different trips happened between source to destinations during 2018, September and October.
- 1504 delivery routes on which trips are happening.
- we have 1508 unique source centres and 1481 unique destination centres.
- From 14817 total different trips, we have 8908 (60%) of the trip-routes are Carting, which consists of small vehicles and 5909 (40%) of total trip-routes are FTL: which are Full Truck Load get to the destination sooner. as no other pickups or drop offs along the way.

Hypothesis tests Results:

```
- from 2 sample t-test, we can also conclude that.

- Average time_taken_btwn_odstart_and_od_end for population is equal to
Average start_scan_to_end_scan for population.

- population average actual_time is less than population average start_scan_to
_end_scan.

- population mean Actual time taken to complete delivery and population mean
time_taken_btwn_od_start_and_od_end is also not same.

- Mean of actual time is higher than Mean of the OSRM estimated time for
delivery.

- Population average for Actual Time taken to complete delivery trip and segme
nt actual time are same.
```

- Average of OSRM Time & segment-osrm-time for population is not same.

- Population Mean osrm time is less than Population Mean segment osrm time.

- Average of OSRM distance for population is less than average of segment OSRM distance.

- population OSRM estimated distance is higher than the actual distance from s ource to destination warehouse.

EDA Results:

- we can observe that Mumbai Maharashtra, Delhi, Gurgaon (Haryana), Bengaluru Karnataka, Hyderabad Telangana, Chennai Tamil Nadu, Ahmedabad-Gujarat, Pune Ma harashtra, Chandigarh and Kolkata West Bengal are some cities have highest number of trips happening states with in the city.

- If we talk about, not having equal source and destination states, source and destination cities having highest number of trips in between are: Delhi TO Gur gaon, Gurgaon TO Bengaluru, Bhiwandi/Mumbai TO Pune Maharashtra, Sonipat TO Gurgaon, Haryana

- It is also been observed that lots of deliveries are happening to airports like: Chennai to MAA Chennai international Airport, Pune to Pune Airport(PNQ), Kolkata to CCU West Bengal Kolkata International Airport , Bengaluru to BLR-Be ngaluru International Airport etc.

- From Bar charts, and calculated tables in analysis, we can observe that high est trips are happening is within the particular cities, in terms of average d istance between destinations, we can observe Guwahati to Mumbai, Bangalore to Chandigarh, Bangalore to Delhi, Bangalore to Gurgaon are the longest routes.

Recommendations:

- As per analysis, It is recommended to use Carting (small vehicles) for delivery within the city in order to reduce the delivery time, and Heavy trucks for long distance trips or heavy load. based on this, we can optimize the delivery time as well as increase the revenue as per requirements.

- Increasing the connectivity in tier 2 and tier 3 cities along with profession tie-ups with several e-commerce giants can increase the revenue as well as the reputation on connectivity across borders.
- We can work on optimizing the scanning time on both ends which is start scanning time and end scanning time so that the delivery time can be equated to the OSRM estimated delivery time.
- Revisit information fed to routing engine for trip planning. Check for discrepancies with transporters, if the routing engine is configured for optimum results.
- North, South and West Zones comidors have significant traffic of orders. But we have a smaller presence in Central, Eastern and North-Eastern zone. However, it would be difficult to conclude this, by looking at just 2 months data. It is worth investigating and increasing our presence in these regions.
- From state point of view, we have heavy traffic in Maharashtra followed by Karnataka. This is a good indicator that we need to plan for resources on ground in these 2 states on priority. Especially, during festive seasons.