In [99]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import kstest, ttest_ind
```

In [2]:
```python
# data = pd.read_csv(r'F:\Muthu_2023\Personal\NextStep\DSCourse\Scaler\Business-
data = pd.read_csv(r'E:\Nextstep\Scaler\Business-Case-Study\Delhivery\Dataset\de
```

In [3]:
```python
data.head()
```

Out[3]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320

5 rows × 24 columns

In [4]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool   
 13  cutoff_factor                   144867 non-null  int64  
 14  cutoff_timestamp                144867 non-null  object 
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

In [5]:
```
data.describe()
```

Out[5]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_ti
count	144867.000000	144867.000000	144867.000000	144867.0000
mean	961.262986	232.926567	234.073372	416.9275
std	1037.012769	344.755577	344.990009	598.1036
min	20.000000	9.000000	9.000045	9.0000
25%	161.000000	22.000000	23.355874	51.0000
50%	449.000000	66.000000	66.126571	132.0000
75%	1634.000000	286.000000	286.708875	513.0000
max	7898.000000	1927.000000	1927.447705	4532.0000

In [6]:
```
data.describe(include='object')
```

Out[6]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
count	144867	144867	144867	144867	14
unique	2	14817	1504	2	1
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	15381121953589
freq	104858	101	1812	99660	

In [7]:

```
data.isnull().sum()
```

Out[7]:

```
data                               0
trip_creation_time                 0
route_schedule_uuid                0
route_type                         0
trip_uuid                          0
source_center                      0
source_name                      293
destination_center                 0
destination_name                 261
od_start_time                      0
od_end_time                        0
start_scan_to_end_scan             0
is_cutoff                          0
cutoff_factor                      0
cutoff_timestamp                   0
actual_distance_to_destination     0
actual_time                        0
osrm_time                          0
osrm_distance                      0
factor                             0
segment_actual_time                0
segment_osrm_time                  0
segment_osrm_distance              0
segment_factor                     0
dtype: int64
```

In [8]:

```
data[data.isnull()]
```

Out[8]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cen
0	NaN	NaN	NaN	NaN	NaN	N
1	NaN	NaN	NaN	NaN	NaN	N
2	NaN	NaN	NaN	NaN	NaN	N
3	NaN	NaN	NaN	NaN	NaN	N
4	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	
144862	NaN	NaN	NaN	NaN	NaN	N
144863	NaN	NaN	NaN	NaN	NaN	N
144864	NaN	NaN	NaN	NaN	NaN	N
144865	NaN	NaN	NaN	NaN	NaN	N
144866	NaN	NaN	NaN	NaN	NaN	N

144867 rows × 24 columns

In [9]:

```
data[data['source_name'].isnull()]
```

Out[9]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
112	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	1537865584377!
113	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	1537865584377!
114	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	1537865584377!
115	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	1537865584377!
116	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	1537865584377!
...	...	...	...	...	
144484	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	1538557566898
144485	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	1538557566898
144486	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	1538557566898
144487	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	1538557566898
144488	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	1538557566898

293 rows × 24 columns

Column wise Analysis

In [10]:

```
# data
df = pd.DataFrame()
```

In [11]:

```
data['data'].value_counts() * 100/data['data'].value_counts().sum()
```

Out[11]:

```
training    72.382254
test        27.617746
Name: data, dtype: float64
```

In [12]:
```
df['data'] = data['data']
```

- Data is split into 72% training set and 28% test set

Trip_creation_time

In [13]:
```
df['trip_creation_time']=pd.to_datetime(data['trip_creation_time'])
df
```

Out[13]:

	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
...	...	...
144862	training	2018-09-20 16:24:28.436231
144863	training	2018-09-20 16:24:28.436231
144864	training	2018-09-20 16:24:28.436231
144865	training	2018-09-20 16:24:28.436231
144866	training	2018-09-20 16:24:28.436231

144867 rows × 2 columns

In [14]:
```
df['trip_creation_time'].dt.date.nunique()
```

Out[14]: 22

Dataset Contains 22 days of data. Hence granular level of year, month and day is not required

In [15]:
```
df['trip_creation_date'] = df['trip_creation_time'].dt.date
df['trip_creation_hour'] = df['trip_creation_time'].dt.hour
df.drop('trip_creation_time', axis=1, inplace=True)
df
```

Out[15]:

	data	trip_creation_date	trip_creation_hour
0	training	2018-09-20	2
1	training	2018-09-20	2
2	training	2018-09-20	2
3	training	2018-09-20	2
4	training	2018-09-20	2
...	...	...	...
144862	training	2018-09-20	16
144863	training	2018-09-20	16
144864	training	2018-09-20	16
144865	training	2018-09-20	16
144866	training	2018-09-20	16

144867 rows × 3 columns

route_schedule_uuid

In [16]:
```
split_0 = data['route_schedule_uuid'].str.split("::").apply(lambda x: x[0])
split_1 = data['route_schedule_uuid'].str.split(":").apply(lambda x: x[2])
split_2 = data['route_schedule_uuid'].str.split(":").apply(lambda x: x[3])
```

In [17]:
```
print(split_0.nunique(), split_1.nunique(), split_2.nunique())
```

```
1 1 1504
```

In [18]:
```
split_0.iloc[0], split_1.iloc[0], split_2.iloc[0]
```

Out[18]:
```
('thanos', 'sroute', 'eb7bfc78-b351-4c0e-a951-fa3d5c3297ef')
```

There is no significant information present in this column and hence can be dropped

route_type

In [19]:
```
data['route_type'].unique()
```

Out[19]:
```
array(['Carting', 'FTL'], dtype=object)
```

In [20]:
```
df[['Cart', 'FTL']] = pd.get_dummies(data['route_type'], columns=['route_type'])
```

In [21]:
```
df['Cart'].value_counts()
```

Out[21]:
```
0    99660
1    45207
Name: Cart, dtype: int64
```

- Performed One hot encoding for route_type column as it has only 2 unique values

trip_uuid

In [22]:
```
data['trip_uuid'].nunique()
```

Out[22]:
```
14817
```

In [23]:
```
# For grouping Trip ID is required
df['trip_uuid'] = data['trip_uuid']
```

- trip_uuid is a unique ID for each trip and it is required for grouping the Trips

source_center

In [24]:
```
data['source_center'].iloc[0]
```

Out[24]:
```
'IND388121AAA'
```

In [25]:
```
data['source_center'].apply(lambda x: x[:3]).unique()
```

Out[25]:
```
array(['IND'], dtype=object)
```

In [26]:
```
data['source_center'].apply(lambda x: x[-3:]).unique()
```

Out[26]:
```
array(['AAA', 'AAB', 'AAG', 'ACA', 'AAC', 'AAD', 'A1B', 'ACK', 'ACB',
       'ABA', 'AAE', 'AAM', 'AFT', 'AAN', 'AAR', 'ACT', 'AAK', 'AFJ',
       'ADV', 'AAF', 'ABD', 'AFG', 'AAL', 'ACN', 'ABG', 'AAJ', 'AAI',
       'AEM', 'AEL', 'AET', 'AAS', 'AFR', 'AAZ', 'AFF', 'AAH', 'ADM',
       'AAQ'], dtype=object)
```

In [27]:
```
data['source_center'].apply(lambda x: x[3:-3:]).nunique()
```

Out[27]:
```
1390
```

In [28]:
```
df['source_center'] = data['source_center']
```

- All the packages starts from IND possibly India
- It contains Unique Id for each center, hence moved as it is for further analysis

source_name

In [29]:
```
# source_name
data['source_name'].iloc[0]
```

Out[29]:
```
'Anand_VUNagar_DC (Gujarat)'
```

In [30]:
```
# Different ways of source name entered in dataset
data['source_name'].fillna("Unk_Unk_Unk (Unk)").str.count("_").value_counts()
```

Out[30]:
```
2    118836
1     12543
3     11381
0      2107
Name: source_name, dtype: int64
```

- `Source name is entered in 4 different formats in the dataset`

In [31]:
```
for i in range(4):
    print(data[data['source_name'].fillna("Unk_Unk_Unk (Unk)").str.count("_") ==
```

```
Haridwar (Uttarakhand)
LowerParel_CP (Maharashtra)
Anand_VUNagar_DC (Gujarat)
Kanpur_Central_H_6 (Uttar Pradesh)
```

- `First string before underscore is City name and inside the brackets is State name`

In [32]:
```
def splitlocation(x):
    if x.count("_"):
        temp1 = x.split("_")
        city = temp1[0]
        temp2 = temp1[-1].split("(")
        state = temp2[1].replace(")", "").strip()
    else:
        temp1 = x.split("(")
        city = temp1[0].strip()
        state = temp1[-1].replace(")", "").strip()
    return city, state
```

In [33]:
```
city_state = data['source_name'].fillna("Unk_Unk_Unk (Unk)").apply(splitlocation
```

In [34]:
```
df['source_city'] = city_state.apply(lambda x: x[0])
df['source_state'] = city_state.apply(lambda x: x[1])
df.head()
```

Out[34]:

	data	trip_creation_date	trip_creation_hour	Cart	FTL	trip_uuid	sour
0	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
1	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
2	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
3	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
4	training	2018-09-20	2	1	0	trip-153741093647649320	IND38

- `As many of the entries in the dataset doesn't contain the name of place, only City and State names are extracted`
- `The missing city and state names in the dataset are modified as "Unk"`

destination_center

In [35]:
```
data['destination_center'].iloc[0]
```

Out[35]:
```
'IND388620AAB'
```

In [36]:
```
data['destination_center'].apply(lambda x: x[:3]).unique()
```

Out[36]:
```
array(['IND'], dtype=object)
```

In [37]:
```
data['destination_center'].apply(lambda x: x[-3:]).unique()
```

Out[37]:
```
array(['AAB', 'AAA', 'AAD', 'ACA', 'AAE', 'AAC', 'A1B', 'AAF', 'ACB',
       'ABA', 'AAG', 'AFT', 'AAM', 'AAJ', 'AAH', 'AAL', 'AAR', 'ABD',
       'ACS', 'ACO', 'AEL', 'AAK', 'AFS', 'AET', 'AAS', 'ACN', 'A1A',
       'ADM', 'AFF', 'AFJ', 'AAZ', 'A1C'], dtype=object)
```

In [38]:
```
data['destination_center'].apply(lambda x: x[3:-3:]).nunique()
```

Out[38]:
```
1384
```

In [39]:
```
df['dest_center'] = data['destination_center']
```

- All the packages starts from IND possibly India
- Unique Id for each center, hence moved as it is for further analysis

destination_name

In [40]:
```
# Different ways of source name entered in dataset
data['destination_name'].fillna("Unk_Unk_Unk (Unk)").str.count("_").value_counts
```

Out[40]:
```
2    117278
1     13127
3     12021
0      2441
Name: destination_name, dtype: int64
```

In [41]:
```
for i in range(4):
    print(data[data['destination_name'].fillna("Unk_Unk_Unk (Unk)").str.count("_
```

```
Haridwar (Uttarakhand)
Jagraon_DC (Punjab)
Khambhat_MotvdDPP_D (Gujarat)
Kanpur_Central_H_6 (Uttar Pradesh)
```

In [42]:
```
city_state = data['destination_name'].fillna("Unk_Unk_Unk (Unk)").apply(splitloc
df['dest_city'] = city_state.apply(lambda x: x[0])
```

```
df['dest_state'] = city_state.apply(lambda x: x[1])
df.head()
```

Out[42]:

	data	trip_creation_date	trip_creation_hour	Cart	FTL	trip_uuid	sour
0	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
1	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
2	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
3	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
4	training	2018-09-20	2	1	0	trip-153741093647649320	IND38

- `As many of the entries in the dataset doesn't contain the name of place, only City and State names are extracted`
- `The missing city and state names in the dataset are modified as "Unk"`

od_start_time and od_end_time

In [43]:
```
#df['trip_time'] = (pd.to_datetime(data['od_end_time']) - pd.to_datetime(data['t
```

- `Total Trip Time is calculated by differencing the end and start time, which is already present in the dataset as "start_scan_to_end_scan" measured in mins`

start_scan_to_end_scan

In [44]:
```
df['start_scan_to_end_scan'] = data['start_scan_to_end_scan']
```

is_cutoff, cutoff_factor, cutoff_timestamp

In [45]:
```
data['is_cutoff'].value_counts()
```

Out[45]:
```
True     118749
False     26118
Name: is_cutoff, dtype: int64
```

In [46]:
```
data['cutoff_factor'].value_counts()
```

Out[46]:
```
22      13157
9       12378
44       8334
18       8263
66       5795
        ...
245         1
734         1
1149        1
412         1
275         1
Name: cutoff_factor, Length: 501, dtype: int64
```

In [47]:
```
df[['is_cutoff','cutoff_factor']] = data[['is_cutoff','cutoff_factor']]
```

In [48]:
```
data['cutoff_timestamp']
```

Out[48]:
```
0                2018-09-20 04:27:55
1                2018-09-20 04:17:55
2         2018-09-20 04:01:19.505586
3                2018-09-20 03:39:57
4                2018-09-20 03:33:55
                     ...
144862           2018-09-20 21:57:20
144863           2018-09-20 21:31:18
144864           2018-09-20 21:11:18
144865           2018-09-20 20:53:19
144866    2018-09-20 16:24:28.436231
Name: cutoff_timestamp, Length: 144867, dtype: object
```

In [49]:
```
df['cufoff_date'] = pd.to_datetime(data['cutoff_timestamp']).dt.date
df['cutoff_time'] = pd.to_datetime(data['cutoff_timestamp']).dt.hour
```

- `Cutoff timestamp is transformed as cutoff date and cutoff time`

actual_distance_to_destination, actual_time, osrm_time, osrm_distance, segment_actual_time, segment_osrm_time, segment_osrm_distance

In [50]:
```
data[['actual_distance_to_destination', 'osrm_distance']].head()
```

Out[50]:

	actual_distance_to_destination	osrm_distance
0	10.435660	11.9653
1	18.936842	21.7243
2	27.637279	32.5395
3	36.118028	45.5620
4	39.386040	54.2181

In [51]:
```
data[data['osrm_distance'] < data['actual_distance_to_destination']].head()
```

Out[51]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
77666	training	2018-09-12 19:52:50.332484	thanos::sroute:c0fc2d36-84dc-406b-b11f-5ebf7eb...	Carting	15367819703322
103033	training	2018-09-12 19:33:12.560599	thanos::sroute:b692be92-80ce-4280-989f-5d98eca...	Carting	15367807925603

2 rows × 24 columns

In [52]:

```
col_names = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm
df[col_names] = data[col_names]
```

factor and segment_factor

In [53]:

```
temp = pd.DataFrame()
temp['actual/osrm'] = data['actual_time']/ data['osrm_time']
temp['factor'] = data['factor']
temp['segmentactual/segmentosrm'] = data['segment_actual_time']/ data['segment_o
temp['segment_factor'] = data['segment_factor']
temp.head()
```

Out[53]:

	actual/osrm	factor	segmentactual/segmentosrm	segment_factor
0	1.272727	1.272727	1.272727	1.272727
1	1.200000	1.200000	1.111111	1.111111
2	1.428571	1.428571	2.285714	2.285714
3	1.550000	1.550000	1.750000	1.750000
4	1.545455	1.545455	1.200000	1.200000

- `From the analysis it is infered that,`
  - **`factor = actual time / osrm Time`**
  - **`segment_Factor = segment_actual_time / segment_osrm_time`**
- `Hence factor and segment factor are ignored as it a redundant data`

In [54]:

```
df.head()
```

Out[54]:

	data	trip_creation_date	trip_creation_hour	Cart	FTL	trip_uuid	sour
0	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
1	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
2	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
3	training	2018-09-20	2	1	0	trip-153741093647649320	IND38
4	training	2018-09-20	2	1	0	trip-153741093647649320	IND38

5 rows × 24 columns

In [55]:
```
df.isnull().sum()
```

Out[55]:
```
data                             0
trip_creation_date               0
trip_creation_hour               0
Cart                             0
FTL                              0
trip_uuid                        0
source_center                    0
source_city                      0
source_state                     0
dest_center                      0
dest_city                        0
dest_state                       0
start_scan_to_end_scan           0
is_cutoff                        0
cutoff_factor                    0
cufoff_date                      0
cutoff_time                      0
actual_distance_to_destination   0
actual_time                      0
osrm_time                        0
osrm_distance                    0
segment_actual_time              0
segment_osrm_time                0
segment_osrm_distance            0
dtype: int64
```

- `All the null values are addressed`

Summary:

- `data => data`
- `trip_creation_time => trip_creation_date, trip_creation_hour`
- `route_schedule_uuid => Dropped`
- `route_type => Cart, FTL`
- `trip_uuid => trip_uuid`

- `source_center => source_center`
- `source_name => source_city, source_state`
- `destination_center => dest_center`
- `destination_name => dest_city, dest_state`
- `od_start_time, od_end_time => Dropped`
- `start_scan_to_end_scan => start_scan_to_end_scan`
- `is_cutoff => is_cutoff`
- `cutoff_factor => cutoff_factor`
- `cutoff_timestamp => cufoff_date, cufoff_time`
- `actual_distance_to_destination => actual_distance_to_destination`
- `actual_time => actual_time`
- `osrm_time => osrm_time`
- `osrm_distance => osrm_distance`
- `factor => Dropped`
- `segment_actual_time => segment_actual_time`
- `segment_osrm_time => segment_osrm_time`
- `segment_osrm_distance => segment_osrm_distance`
- `segment_factor => Dropped`

EDA - Full Data

Bivariate Analysis

In [57]:
```
#Top 5 centers with Longer distance
df.groupby(['source_center', 'dest_center'])['segment_osrm_distance'].mean().sor
```

Out[57]:
```
source_center  dest_center
IND284403AAA   IND474003AAA    223.2655
IND425412AAA   IND424006AAA    109.1615
IND173212AAA   IND160002AAC    101.7296
IND743270AAA   IND712311AAA     98.7449
IND425409AAA   IND424006AAA     94.5602
Name: segment_osrm_distance, dtype: float64
```

In [58]:
```
#Top 5 centers with Longer travel time
df.groupby(['source_center', 'dest_center'])['segment_osrm_time'].mean().sort_va
```

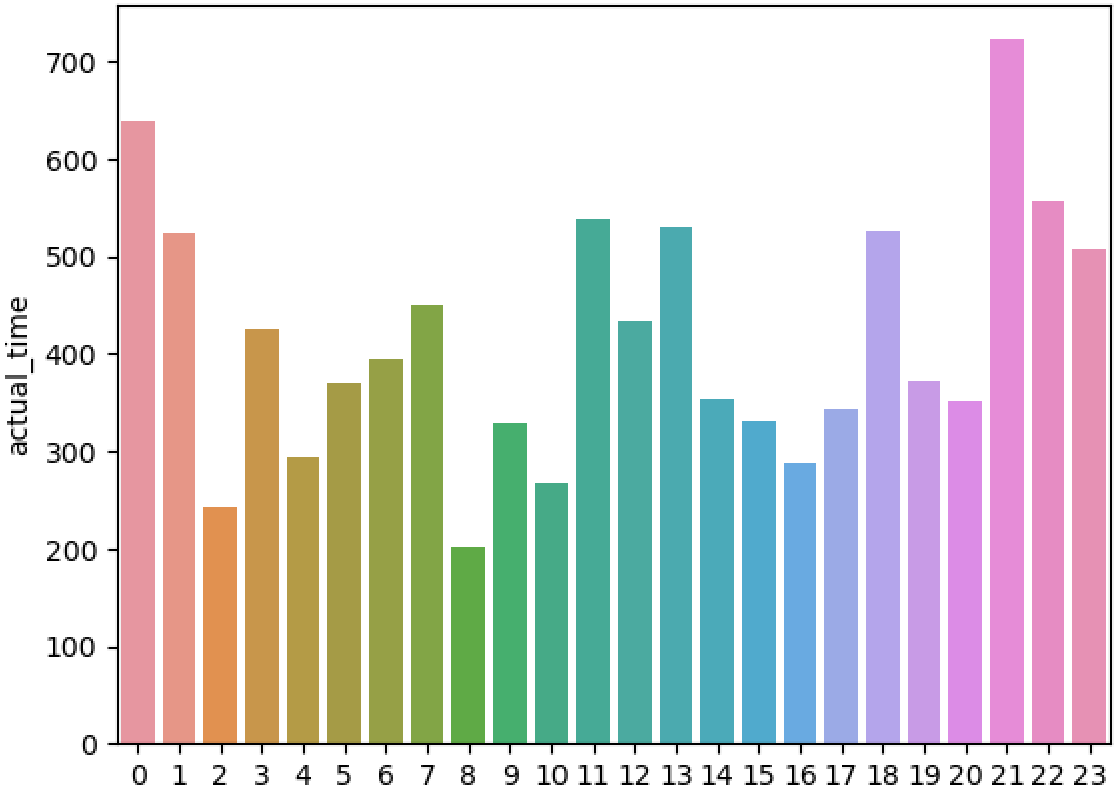
Out[58]:
```
source_center  dest_center
IND284403AAA   IND474003AAA    208.0
IND173212AAA   IND160002AAC     95.0
IND425412AAA   IND424006AAA     79.0
IND671315AAA   IND575004AAB     78.0
IND465001AAA   IND465333A1B     77.0
Name: segment_osrm_time, dtype: float64
```

In [59]:
```
#Top 5 centers with Longer travel time
df.groupby(['source_center', 'dest_center'])['segment_actual_time'].mean().sort_
```

Out[59]:
```
source_center   dest_center
IND722140AAA    IND723130AAA    1320.0
IND743270AAA    IND712311AAA    1133.6
IND425412AAA    IND424006AAA    1093.0
IND424304AAC    IND424006AAA     926.0
IND425409AAA    IND424006AAA     894.0
Name: segment_actual_time, dtype: float64
```

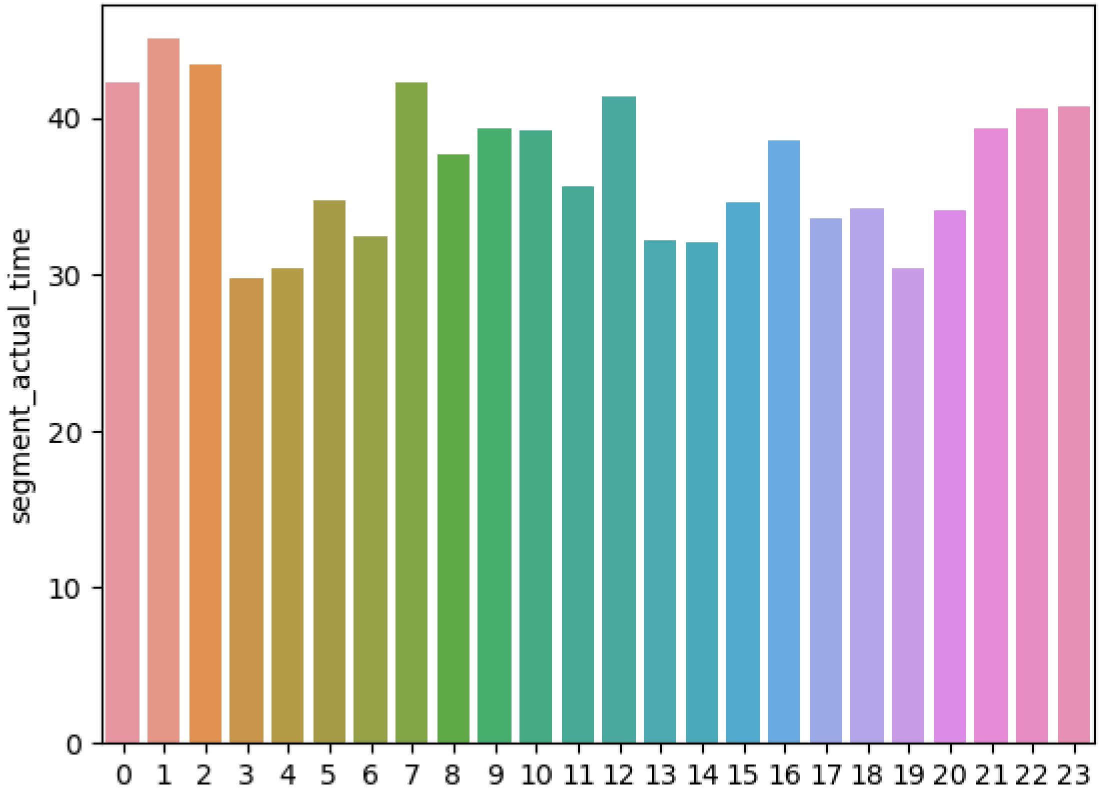
In [60]:
```
# Trip Creation Time vs Actual time
sns.barplot(x=list(df['trip_creation_hour'].unique()), y = df.groupby(['trip_cre
```

Out[60]: <Axes: ylabel='actual_time'>



In [61]:
```
# Trip Creation Time vs Actual time
sns.barplot(x=list(df['cutoff_time'].unique()), y = df.groupby(['cutoff_time'])[
```

Out[61]: <Axes: ylabel='segment_actual_time'>



Trip Level

In [63]:
```
dic = {'trip_creation_date':'max',
       'trip_creation_hour':'max',
       'Cart':'max', 'FTL':'max',
       'start_scan_to_end_scan': 'max',
       'cutoff_factor': 'max',
       'actual_distance_to_destination':'max',
       'actual_time': 'max',
       'osrm_time': 'max',
       'osrm_distance':'max',
       'segment_actual_time':'sum',
       'segment_osrm_time':'sum',
       'segment_osrm_distance':'sum'}
```

In [64]:
```
#For Group Analysis
drop_cols = ['data','is_cutoff', 'cutoff_factor','cufoff_date', 'cutoff_time','s
               'segment_osrm_time', 'segment_osrm_distance']
drop_cols = ['data','is_cutoff','cufoff_date', 'cutoff_time', 'source_center', '
df_trip = df.drop(drop_cols,axis=1).groupby(['trip_uuid', 'source_city', 'source
```

In [65]:
```
df_trip.head()
```

Out[65]:

	trip_uuid	source_city	source_state	dest_city	dest_state	trip_creation_
0	trip-153671041653548748	Bhopal	Madhya Pradesh	Kanpur	Uttar Pradesh	2018-0
1	trip-153671041653548748	Kanpur	Uttar Pradesh	Gurgaon	Haryana	2018-0
2	trip-153671042288605164	Doddablpur	Karnataka	Chikblapur	Karnataka	2018-0
3	trip-153671042288605164	Tumkur	Karnataka	Doddablpur	Karnataka	2018-0
4	trip-153671043369099517	Bangalore	Karnataka	Gurgaon	Haryana	2018-0

In [66]:

```
df_trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26095 entries, 0 to 26094
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   trip_uuid                       26095 non-null  object 
 1   source_city                     26095 non-null  object 
 2   source_state                    26095 non-null  object 
 3   dest_city                       26095 non-null  object 
 4   dest_state                      26095 non-null  object 
 5   trip_creation_date              26095 non-null  object 
 6   trip_creation_hour              26095 non-null  int64  
 7   Cart                            26095 non-null  uint8  
 8   FTL                             26095 non-null  uint8  
 9   start_scan_to_end_scan          26095 non-null  float64
 10  cutoff_factor                   26095 non-null  int64  
 11  actual_distance_to_destination  26095 non-null  float64
 12  actual_time                     26095 non-null  float64
 13  osrm_time                       26095 non-null  float64
 14  osrm_distance                   26095 non-null  float64
 15  segment_actual_time             26095 non-null  float64
 16  segment_osrm_time               26095 non-null  float64
 17  segment_osrm_distance           26095 non-null  float64
dtypes: float64(8), int64(2), object(6), uint8(2)
memory usage: 3.2+ MB
```

Exploratory Data Analysis

Univariate Analysis

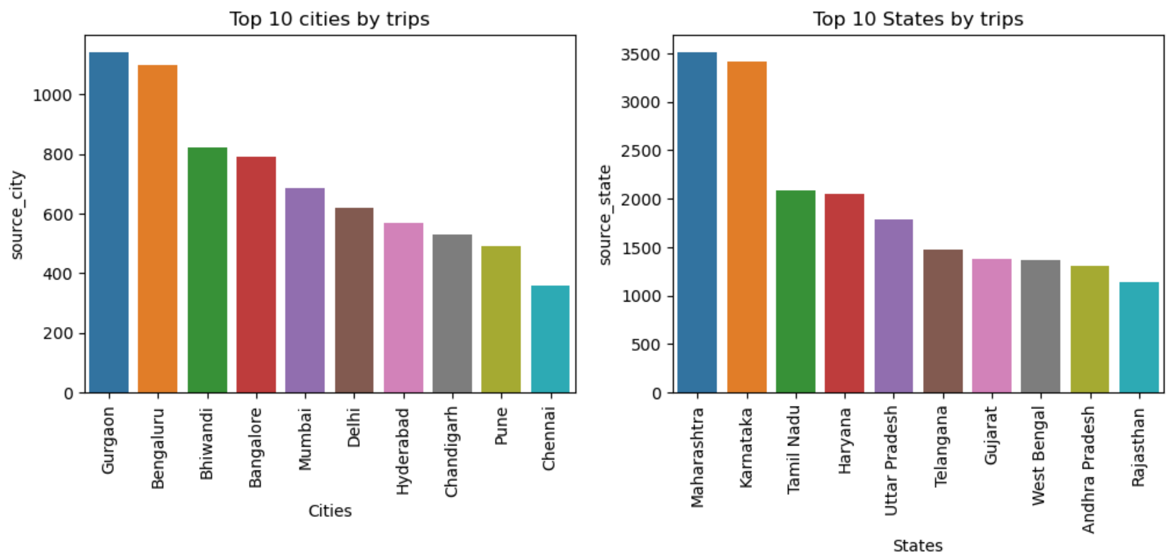
Cities and States

In [67]:

```
# Top 10 Cities contributing for revenue the most
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
```

```
sns.barplot(y = df_trip['source_city'].value_counts()[:10], x= df_trip['source_c
plt.xticks(rotation=90)
plt.xlabel('Cities')
plt.title('Top 10 cities by trips')

plt.subplot(1,2,2)
sns.barplot(y = df_trip['source_state'].value_counts()[:10], x= df_trip['source_
plt.xticks(rotation=90)
plt.xlabel('States')
plt.title('Top 10 States by trips')
plt.show()
```

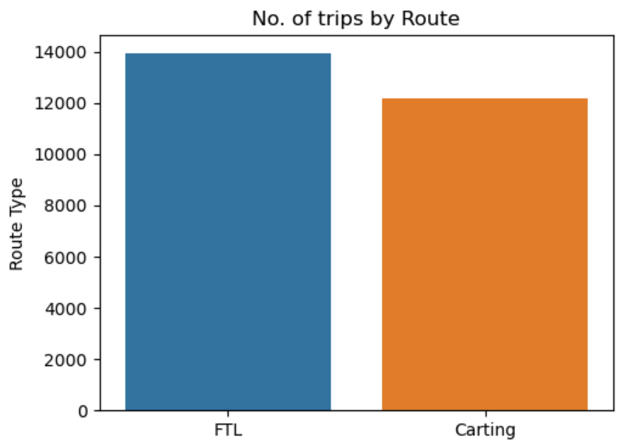


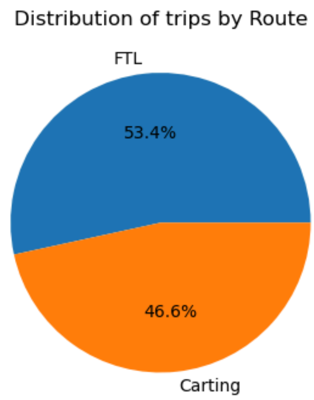
- `Majority of the trips are sourced at the metro cities`
- `Delhivery business is strong in Maharashtra and Karnataka`

Route Type

In [68]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.barplot(data = df_trip, y = df_trip['Cart'].value_counts(), x = ['FTL', 'Car
# plt.xticks(rotation=45)
plt.ylabel('Route Type')
plt.title('No. of trips by Route')

plt.subplot(1,2,2)
plt.pie(df_trip['Cart'].value_counts(), labels = ['FTL', 'Carting'], autopct='%1
plt.title('Distribution of trips by Route')
plt.show()
```





- `Slightly more number of trips are carried out in FTL`

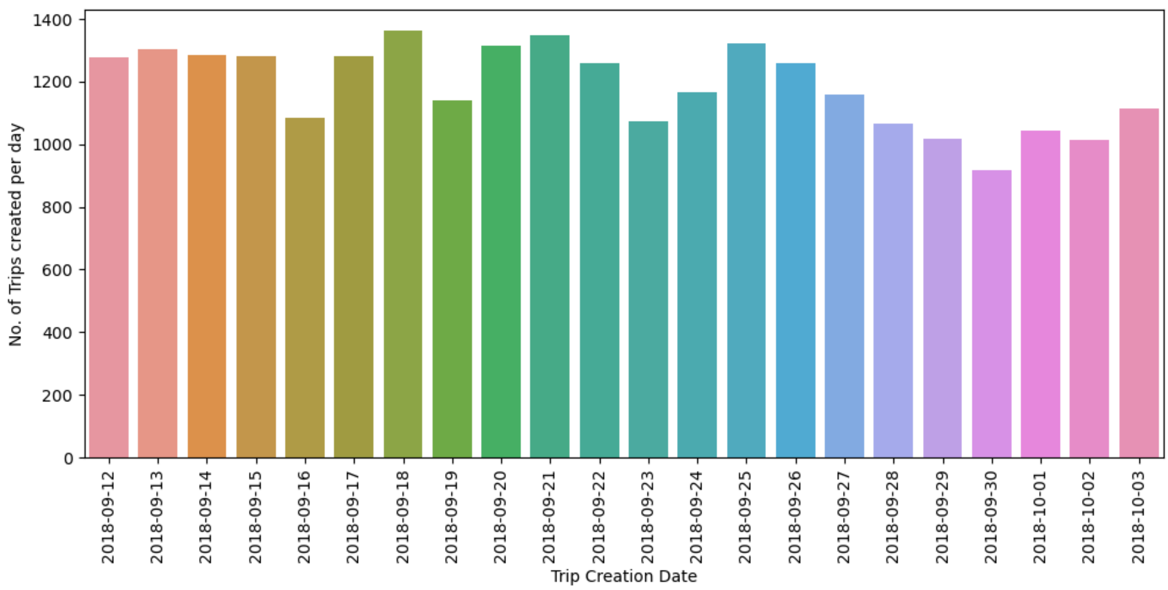
Trip Creation

In [69]:
```
print('No. of days in dataset: ', df_trip['trip_creation_date'].nunique())
```

```
No. of days in dataset:  22
```

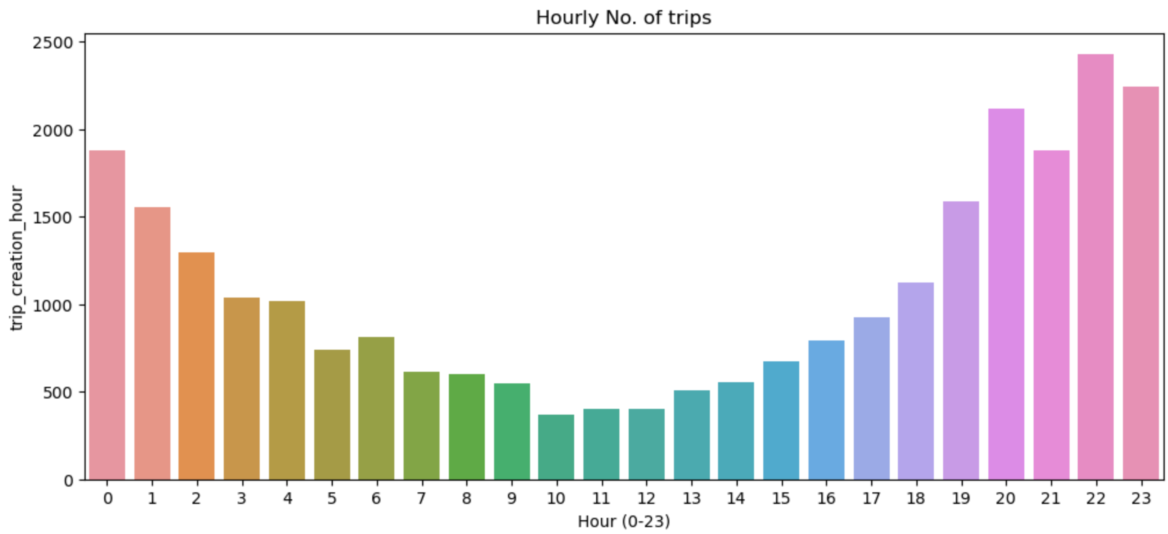
In [70]:
```
plt.figure(figsize=(12,5))
sns.barplot(x=list(df_trip['trip_creation_date'].value_counts(sort=False).index)
plt.xticks(rotation=90)
plt.xlabel('Trip Creation Date')
plt.ylabel('No. of Trips created per day')
```

Out[70]:
```
Text(0, 0.5, 'No. of Trips created per day')
```



In [71]:
```
plt.figure(figsize=(12,5))
sns.barplot(data = df_trip, y = df_trip['trip_creation_hour'].value_counts(), x
# plt.xticks(rotation=45)
plt.xlabel('Hour (0-23)')
plt.title('Hourly No. of trips')
```

Out[71]:
```
Text(0.5, 1.0, 'Hourly No. of trips')
```



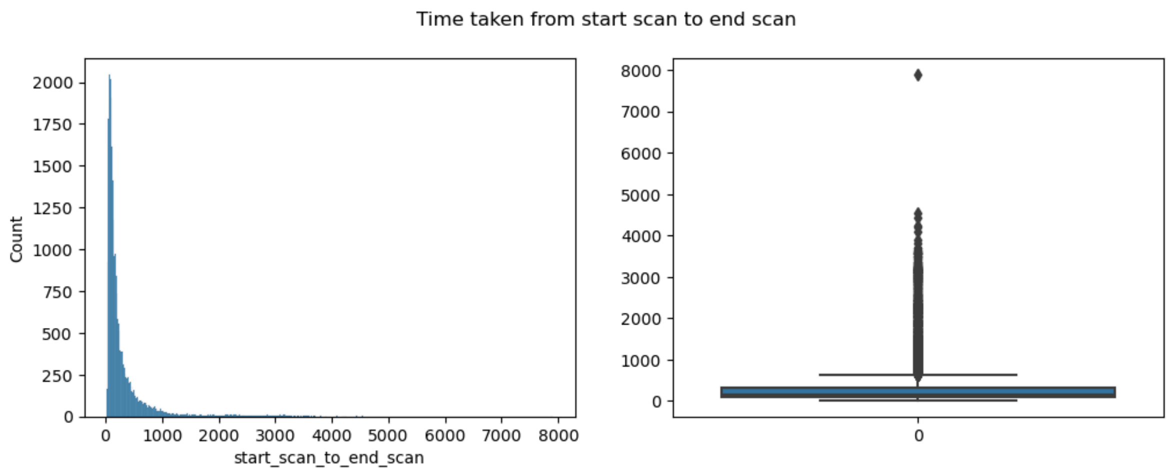
- `Dataset contains trip details for 22 days only, hence day wise analysis will not provide any significant insights`

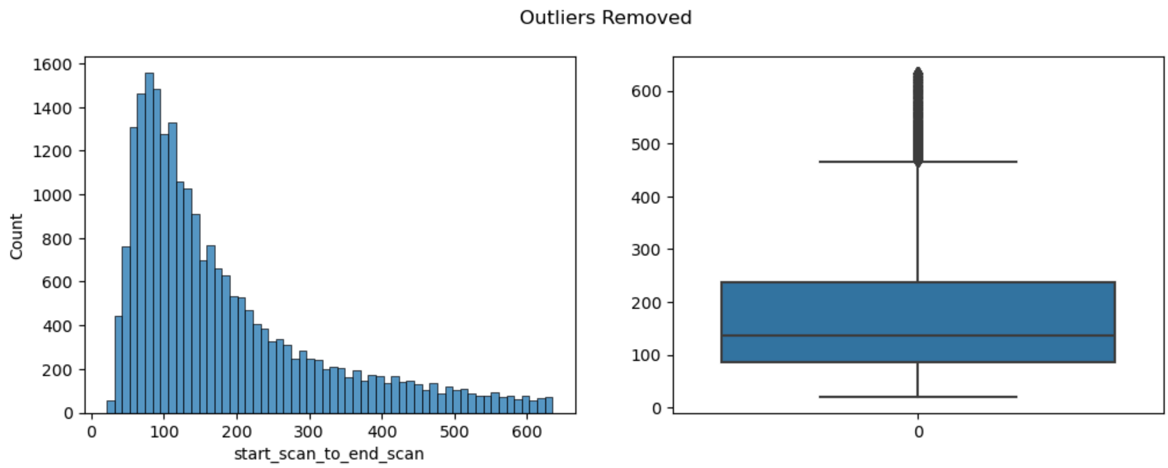
Time

In [72]:
```
def remove_outliers_iqr(df_col):
    q1 = df_col.quantile(0.25)
    q3 = df_col.quantile(0.75)
    iqr = q3 - q1
    return df_col[(df_col > (q1 - 1.5 * iqr)) & (df_col < (q3 + 1.5 * iqr))]
```

In [73]:
```
def plots1X2(df_col, title_str):
    plt.figure(figsize=(12,4)).suptitle(title_str)
    plt.subplot(1,2,1)
    sns.histplot(df_col)
    plt.subplot(1,2,2)
    sns.boxplot(df_col)
```

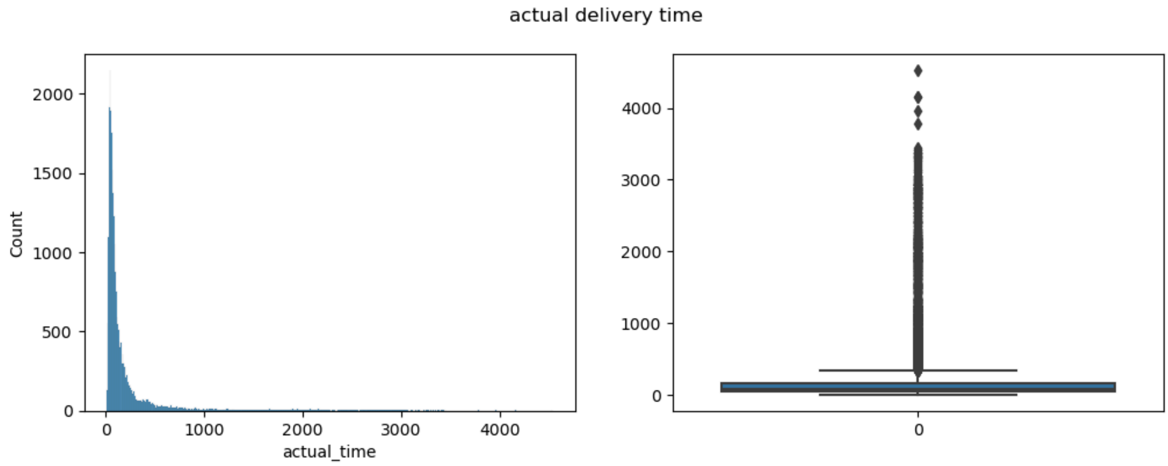
In [74]:
```
plots1X2(df_trip['start_scan_to_end_scan'], "Time taken from start scan to end s
plots1X2(remove_outliers_iqr(df_trip['start_scan_to_end_scan']).reset_index(drop
```

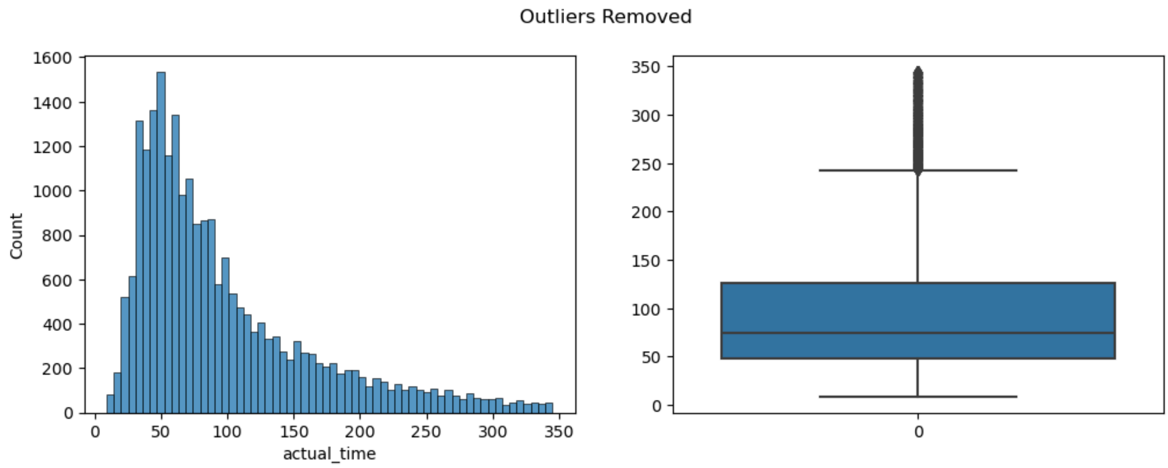




- `Data is Right skewed with median around 150 minutes`

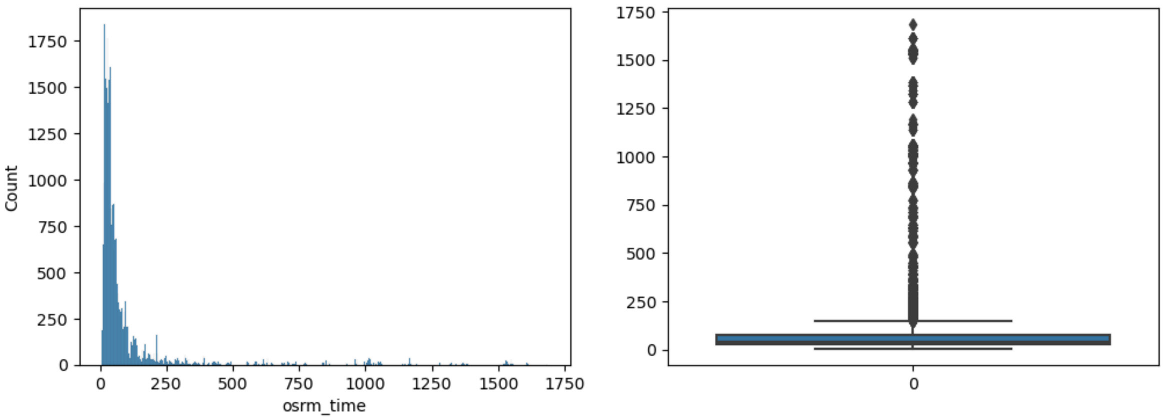
In [75]:
```
plots1X2(df_trip['actual_time'], "actual delivery time")
plots1X2(remove_outliers_iqr(df_trip['actual_time']).reset_index(drop=True), "Ou
```



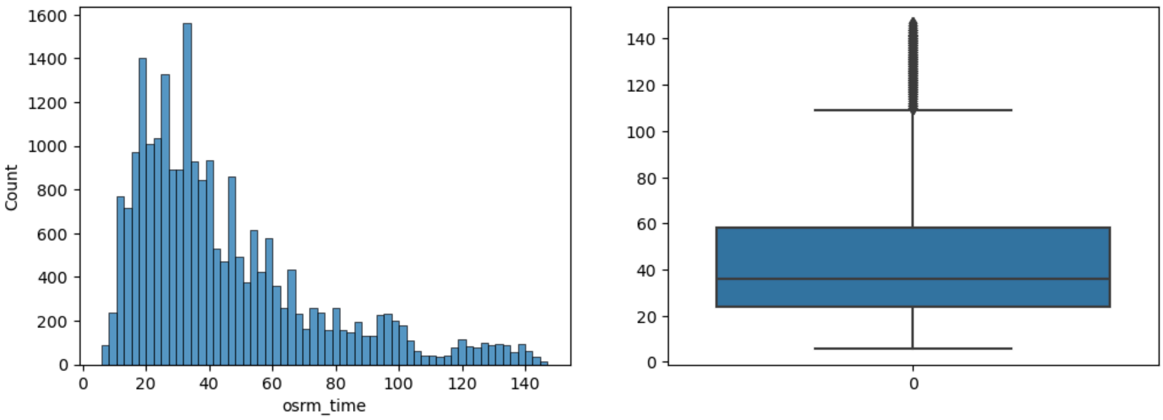


In [76]:
```
plots1X2(df_trip['osrm_time'], "OSRM time")
plots1X2(remove_outliers_iqr(df_trip['osrm_time']).reset_index(drop=True), "Outl
```

OSRM time

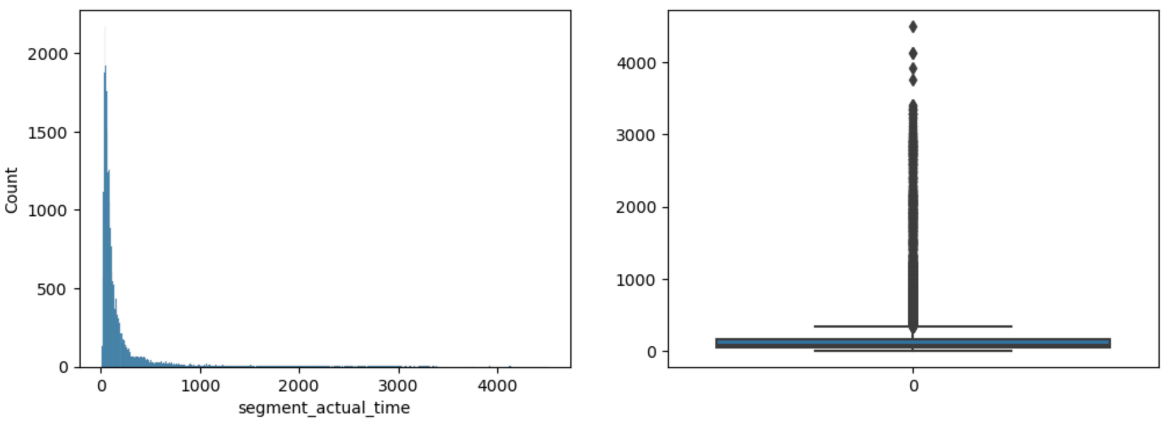


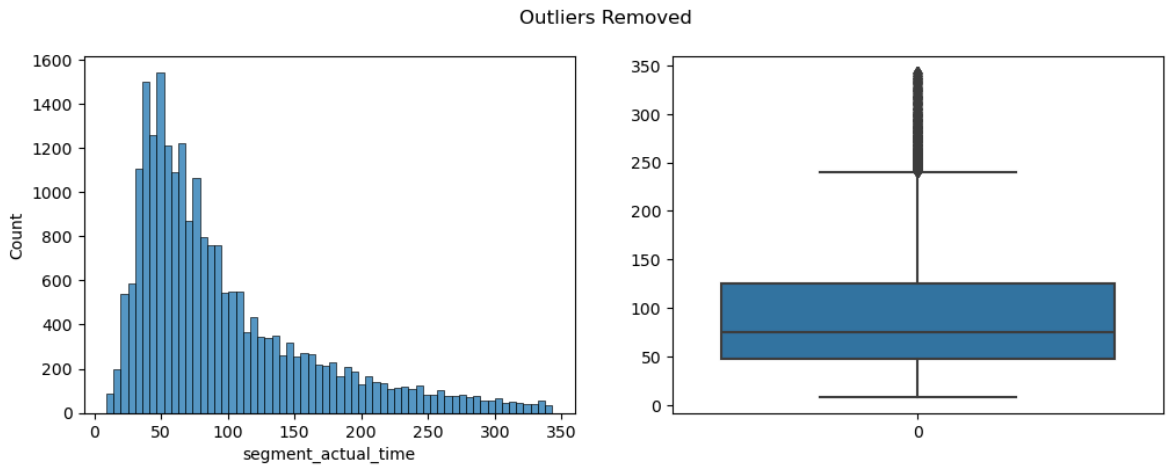
Outliers Removed



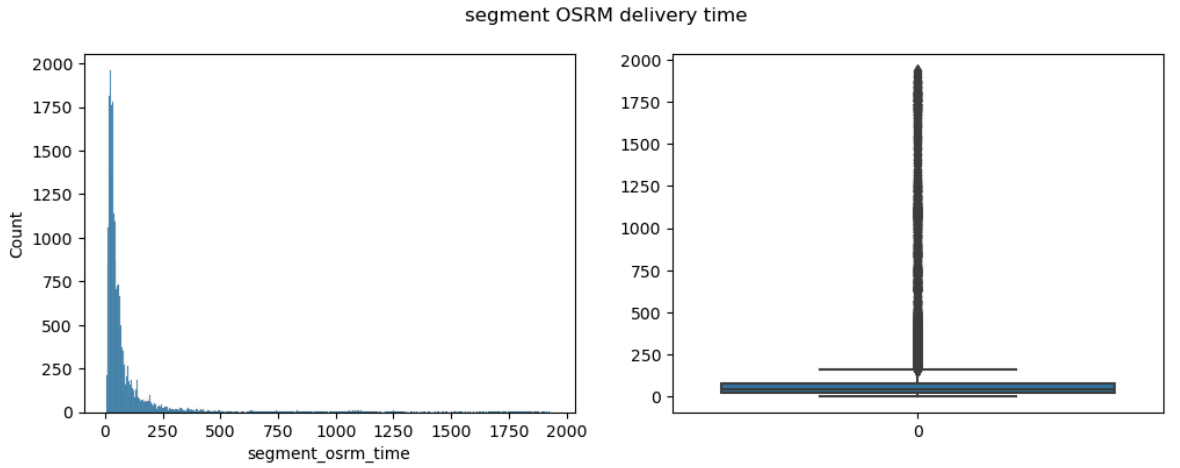
In [77]:
```
plots1X2(df_trip['segment_actual_time'], "segment_actual delivery time")
plots1X2(remove_outliers_iqr(df_trip['segment_actual_time']).reset_index(drop=Tr
```

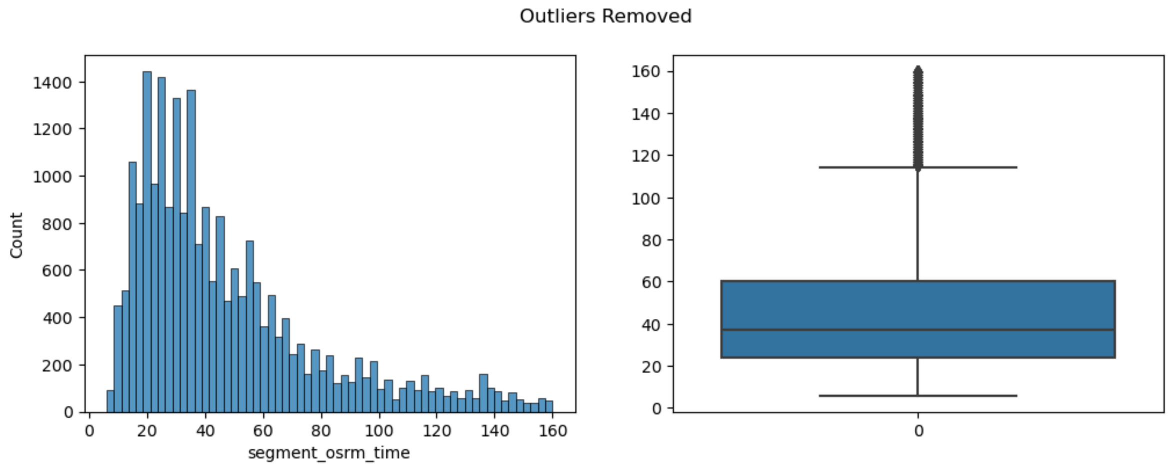
segment_actual delivery time





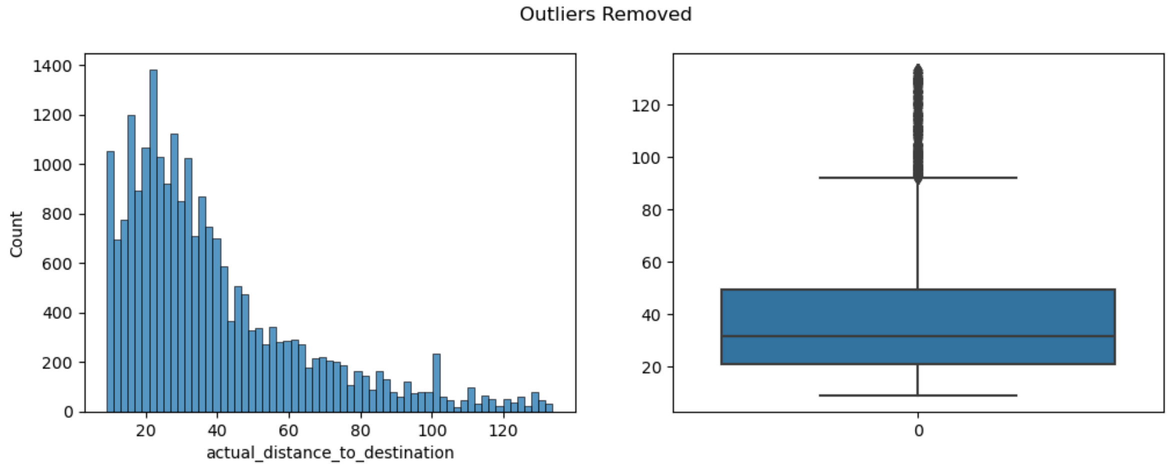
In [78]:
```
plots1X2(df_trip['segment_osrm_time'], "segment OSRM delivery time")
plots1X2(remove_outliers_iqr(df_trip['segment_osrm_time']).reset_index(drop=True
```



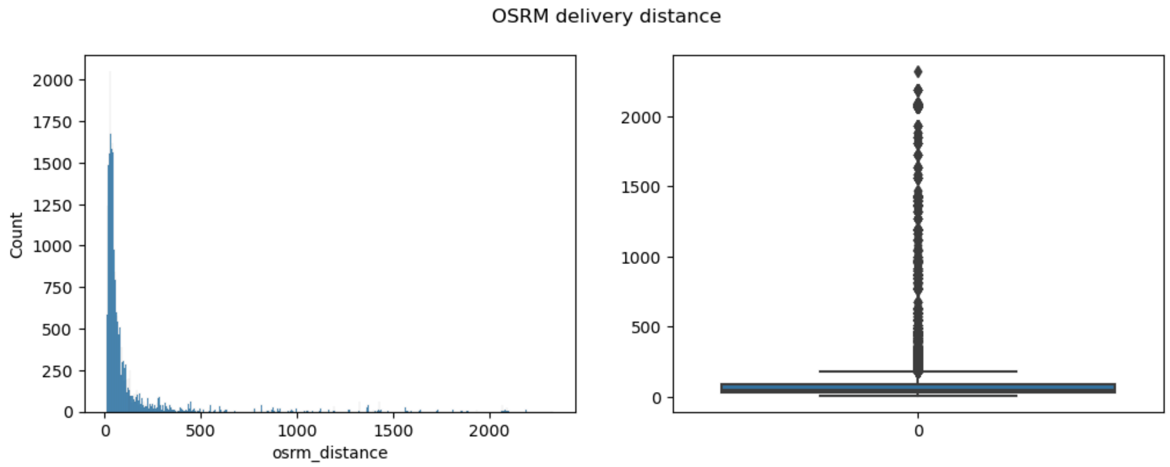


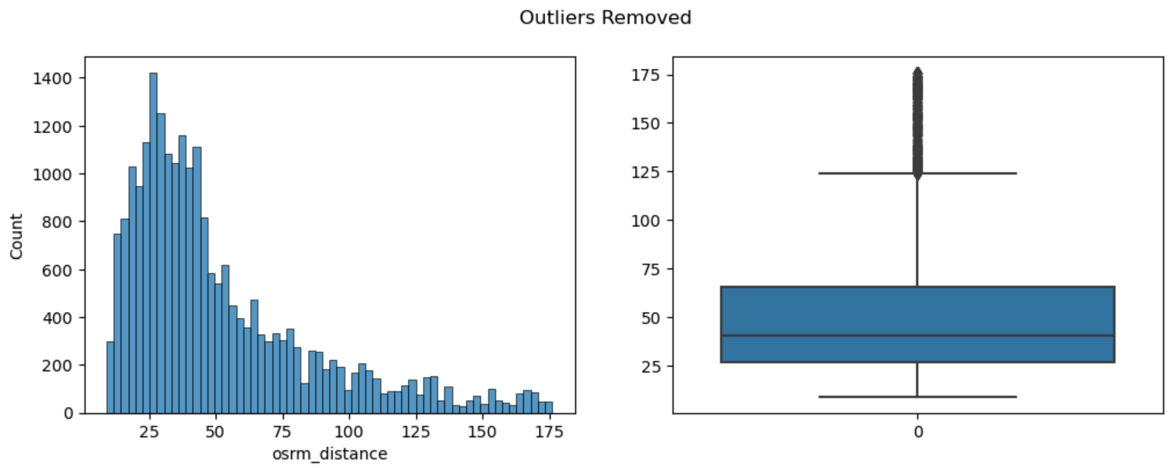
Distance

In [79]:
```
plots1X2(df_trip['actual_distance_to_destination'], "actual delivery distance")
plots1X2(remove_outliers_iqr(df_trip['actual_distance_to_destination']).reset_in
```



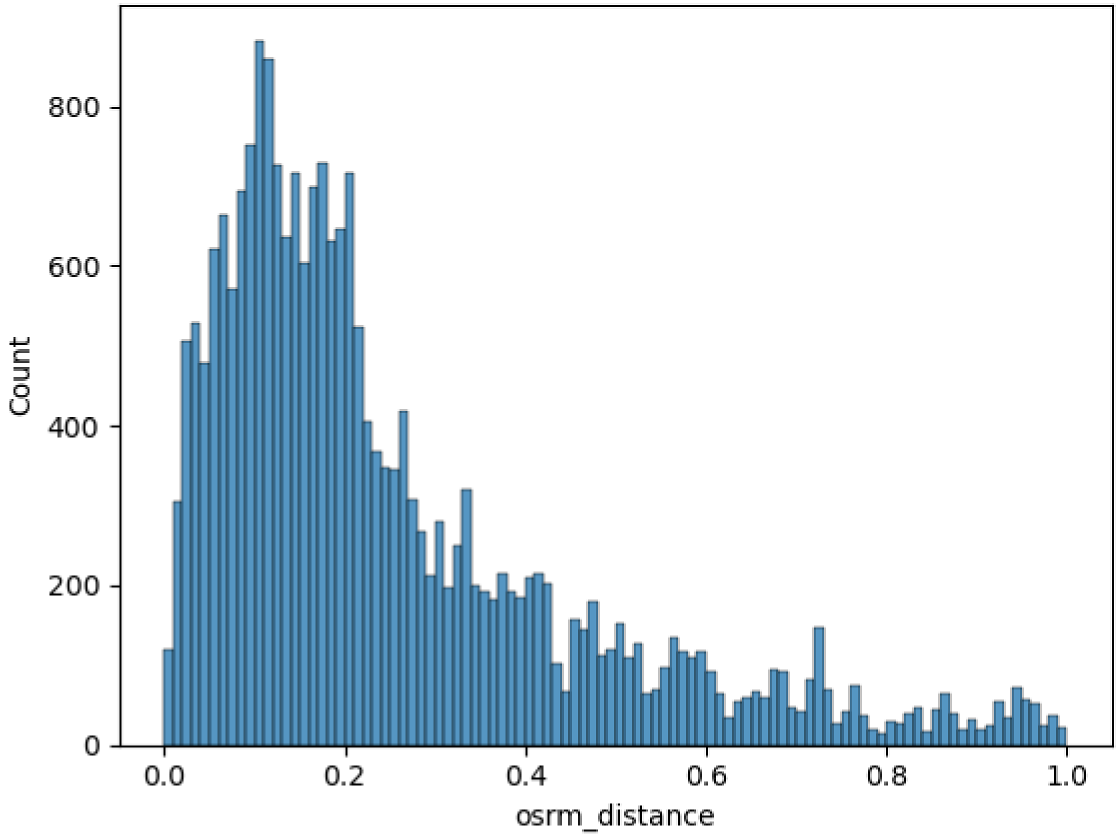
In [80]:
```
plots1X2(df_trip['osrm_distance'], "OSRM delivery distance")
plots1X2(remove_outliers_iqr(df_trip['osrm_distance']).reset_index(drop=True), "
```





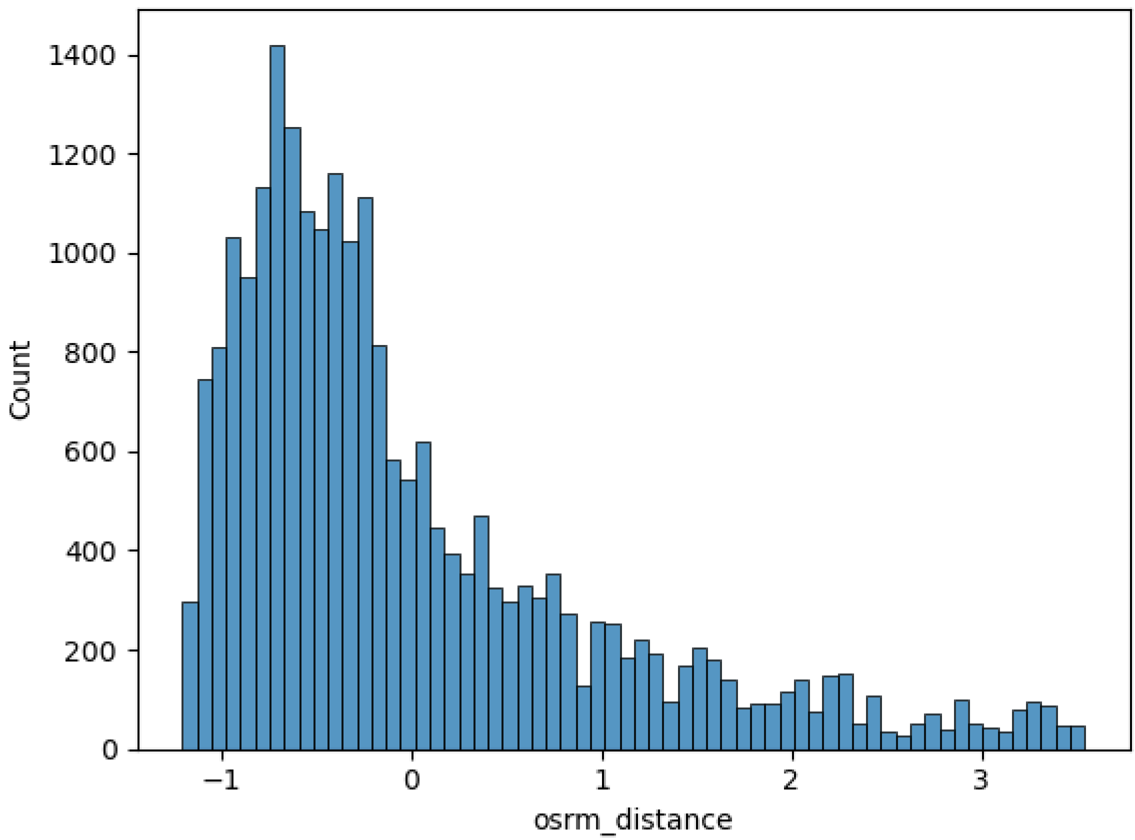
In [81]:
```
outliers_rem = remove_outliers_iqr(df_trip['osrm_distance']).reset_index(drop=Tr
sns.histplot(outliers_rem.transform(lambda x: (x - x.min()) / (x.max()-x.min()))
```

Out[81]: `<Axes: xlabel='osrm_distance', ylabel='Count'>`



In [82]:
```
sns.histplot(outliers_rem.transform(lambda x: (x - x.mean()) / (x.std())))
```

Out[82]: `<Axes: xlabel='osrm_distance', ylabel='Count'>`



Bivariate Analysis

In [84]:
```
df_trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26095 entries, 0 to 26094
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   trip_uuid                       26095 non-null  object 
 1   source_city                     26095 non-null  object 
 2   source_state                    26095 non-null  object 
 3   dest_city                       26095 non-null  object 
 4   dest_state                      26095 non-null  object 
 5   trip_creation_date              26095 non-null  object 
 6   trip_creation_hour              26095 non-null  int64  
 7   Cart                            26095 non-null  uint8  
 8   FTL                             26095 non-null  uint8  
 9   start_scan_to_end_scan          26095 non-null  float64
 10  cutoff_factor                   26095 non-null  int64  
 11  actual_distance_to_destination  26095 non-null  float64
 12  actual_time                     26095 non-null  float64
 13  osrm_time                       26095 non-null  float64
 14  osrm_distance                   26095 non-null  float64
 15  segment_actual_time             26095 non-null  float64
 16  segment_osrm_time               26095 non-null  float64
 17  segment_osrm_distance           26095 non-null  float64
dtypes: float64(8), int64(2), object(6), uint8(2)
memory usage: 3.2+ MB
```

In [85]:
```
corridor = df_trip.groupby(['source_city', 'dest_city'])[['trip_uuid', 'actual_d
```

In [86]:
```
corridor[corridor['source_city'] == corridor['dest_city']].sort_values('trip_uui
```
Out[86]:

	source_city	dest_city	trip_uuid
272	Bengaluru	Bengaluru	528
958	Hyderabad	Hyderabad	308
1530	Mumbai	Mumbai	252
456	Chandigarh	Chandigarh	223
481	Chennai	Chennai	205

In [87]:
```
corridor[corridor['source_city'] == corridor['dest_city']].sort_values('trip_uui
```
Out[87]:

	source_city	dest_city	trip_uuid
272	Bengaluru	Bengaluru	528
958	Hyderabad	Hyderabad	308
1530	Mumbai	Mumbai	252
456	Chandigarh	Chandigarh	223
481	Chennai	Chennai	205

- `Bengaluru, Hyderabad and Mumbai account for majority of intra city deliveries`
- `For intercity deliveries, Bhiwandi <-> Mumbai, Gurgaon <-> Delhi corridors are the busiest`

In [88]:
```
corridor.sort_values('actual_distance_to_destination', ascending=False).iloc[:5]
```
Out[88]:

	source_city	dest_city	actual_distance_to_destination
450	Chandigarh	Bangalore	1927.447705
832	Gurgaon	MAA	1721.280753
273	Bengaluru	Gurgaon	1694.385273
200	Bangalore	Gurgaon	1691.740938
808	Gurgaon	Bangalore	1689.772879

In [89]:
```
corridor.sort_values('actual_time', ascending=False).iloc[:5][['source_city', 'd
```

Out[89]:

	source_city	dest_city	actual_time
450	Chandigarh	Bangalore	3784.000000
850	Guwahati	Delhi	3370.294118
580	Delhi	Guwahati	3306.000000
1264	Kolkata	Bhiwandi	3169.400000
832	Gurgaon	MAA	3117.642857

In [90]:
```
corridor_state = df_trip.groupby(['source_state', 'dest_state'])['trip_uuid'].cc
```

In [91]:
```
corridor_state[corridor_state['source_state'] == corridor_state['dest_state']].i
```

Out[91]:

	source_state	dest_state	trip_uuid
0	Maharashtra	Maharashtra	3203
1	Karnataka	Karnataka	3121
2	Tamil Nadu	Tamil Nadu	1977
3	Uttar Pradesh	Uttar Pradesh	1485
4	Telangana	Telangana	1307

In [92]:
```
corridor_state[corridor_state['source_state'] != corridor_state['dest_state']].i
```

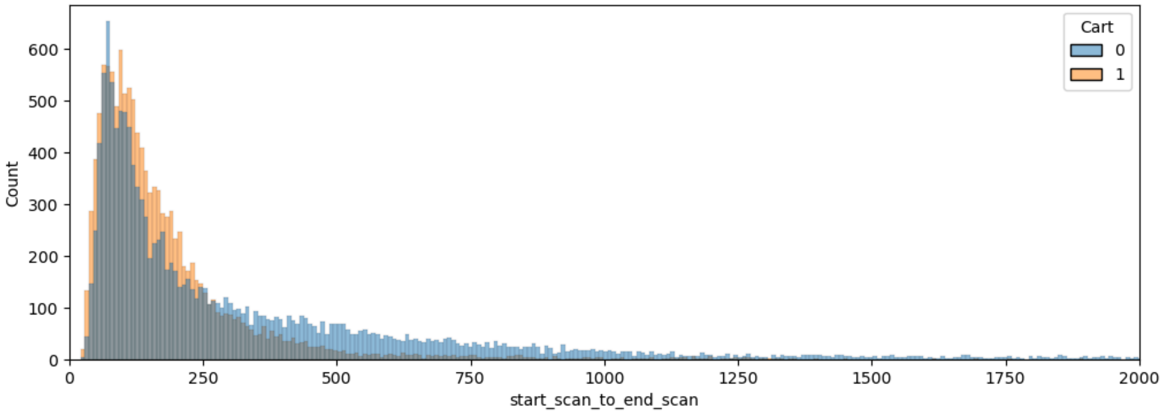
Out[92]:

	source_state	dest_state	trip_uuid
14	Delhi	Haryana	451
16	Haryana	Delhi	315
22	Haryana	Uttar Pradesh	140
23	Uttar Pradesh	Haryana	130
24	Chandigarh	Punjab	121
25	Delhi	Uttar Pradesh	110
26	Haryana	Punjab	103
27	Uttar Pradesh	Delhi	93
28	Andhra Pradesh	Telangana	92
29	Haryana	Rajasthan	85
30	Punjab	Chandigarh	84
31	Punjab	Haryana	80
32	Telangana	Andhra Pradesh	69
33	Himachal Pradesh	Punjab	66
34	Haryana	Karnataka	66

- Maharashtra, Karnataka and Tamilnadu account for majority of intra state deliveries
- For inter state deliveries, Delhi <-> Haryana, Haryana <-> UP corridors are the busiest

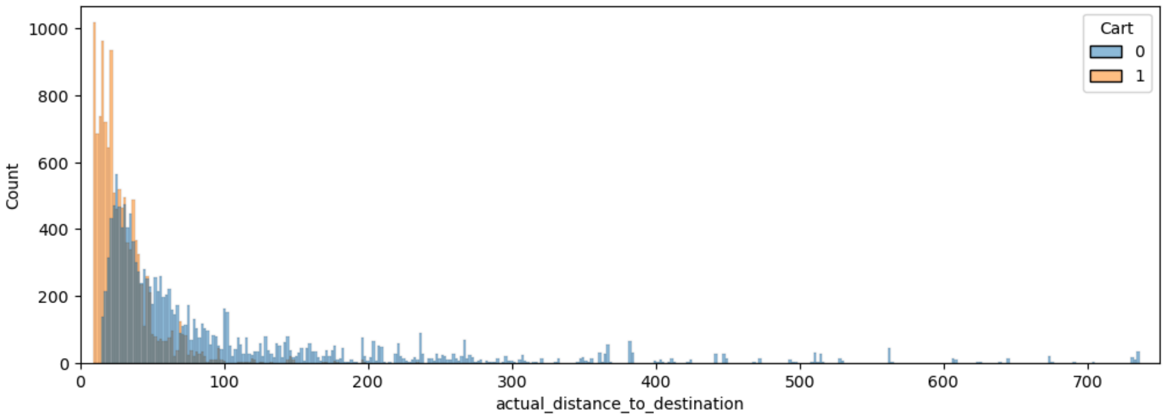
In [93]:
```
#Actual Time vs Route
plt.figure(figsize=(12,4))
sns.histplot(data = df_trip, x='start_scan_to_end_scan', hue='Cart', bins=1000)
plt.xlim(0,2000)
```

Out[93]: (0.0, 2000.0)



In [94]:
```
#Distance to destination vs Route
plt.figure(figsize=(12,4))
sns.histplot(data = df_trip, x='actual_distance_to_destination', hue='Cart', bir
plt.xlim(0,750)
```

Out[94]: (0.0, 750.0)

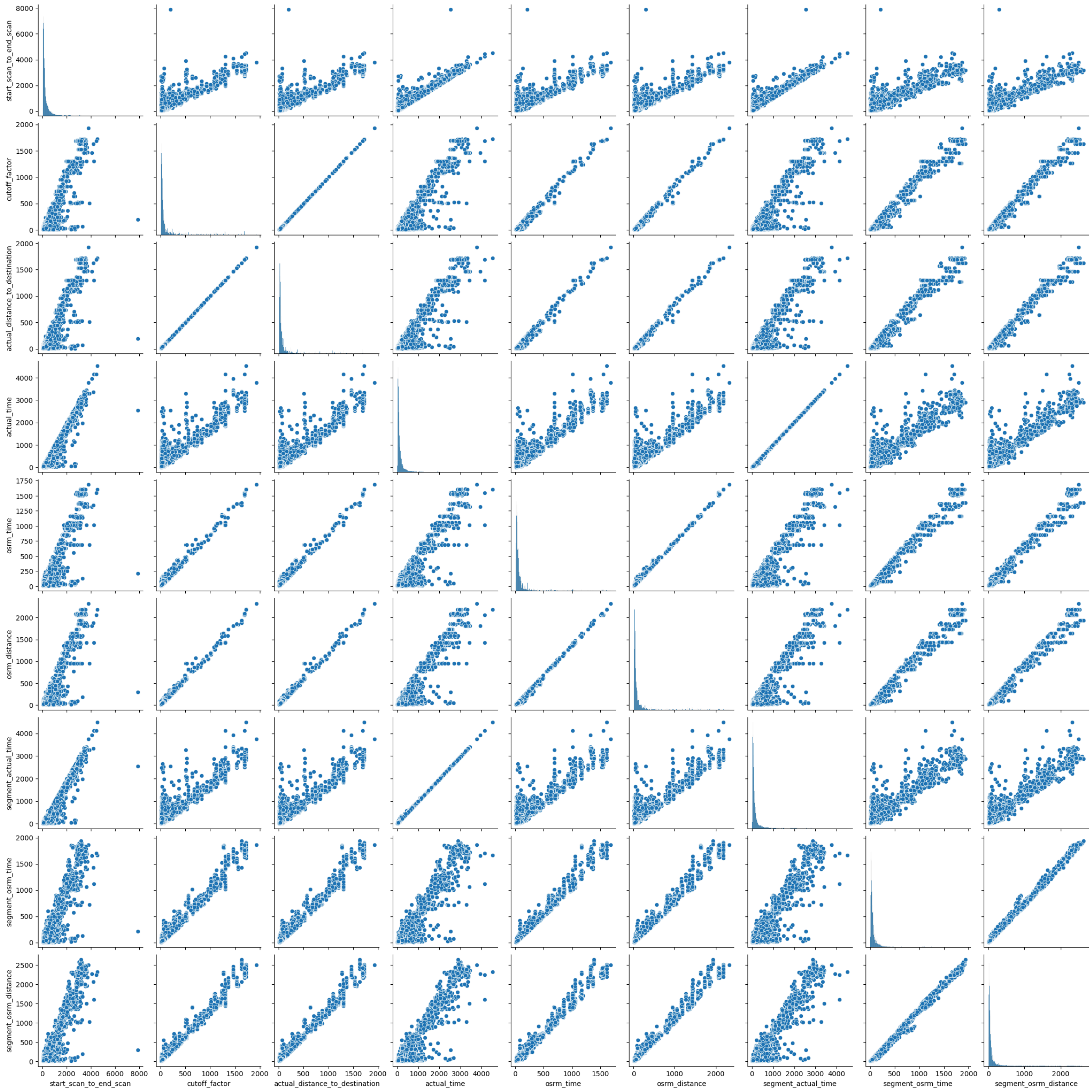


- It is evident that Trucks are used on the long distance trip and eventually it consumes lot of time to transfer

In [259…
```
sns.pairplot(df_trip.drop(['trip_creation_hour', 'Cart', 'FTL'], axis=1))
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

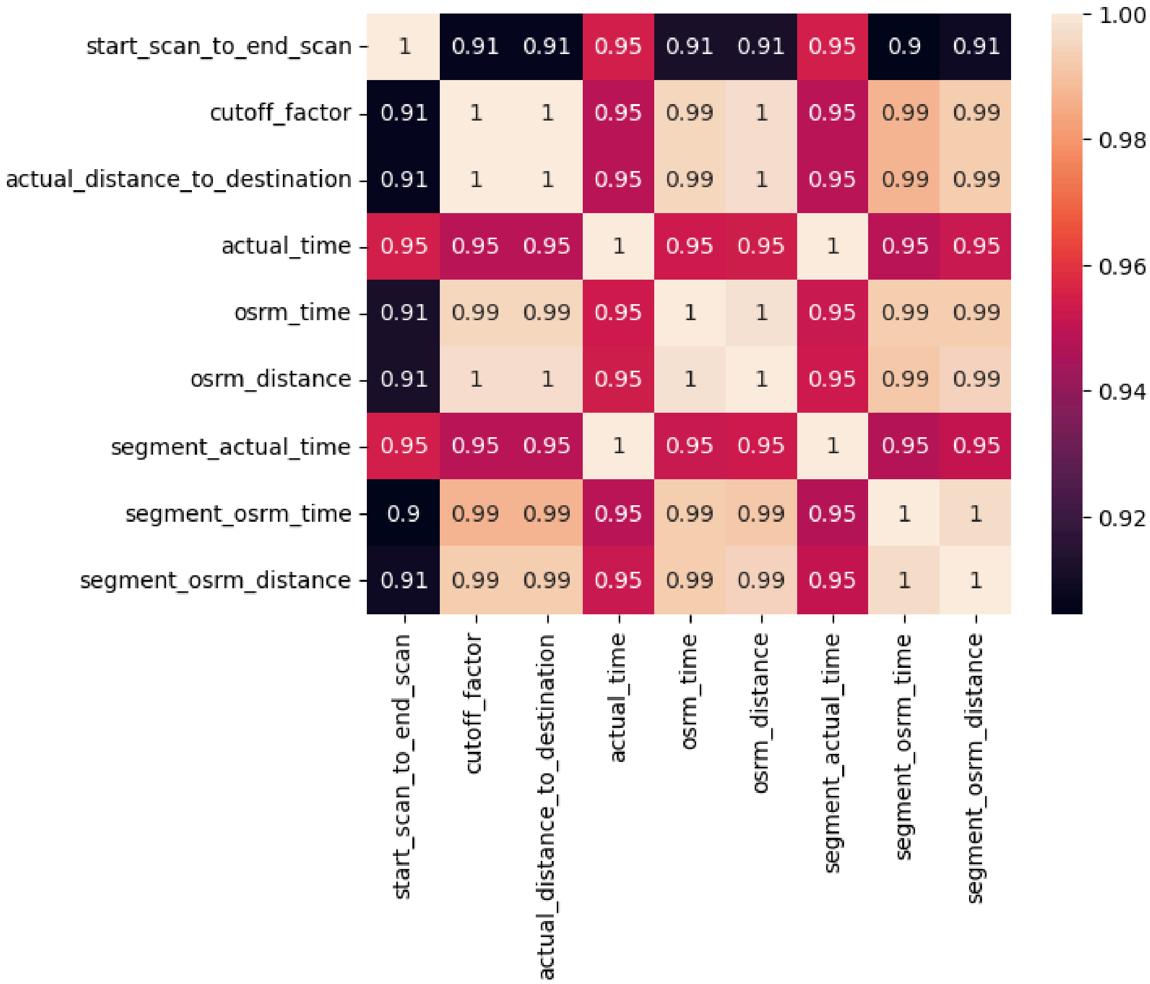
Out[259… <seaborn.axisgrid.PairGrid at 0x23840e46050>



In [261…
```
sns.heatmap(df_trip.drop(['trip_creation_hour', 'Cart', 'FTL'], axis=1).corr(),
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_5556\1031387396.py:1: FutureWarning: 
The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
rsion, it will default to False. Select only valid columns or specify the value o
f numeric_only to silence this warning.
  sns.heatmap(df_trip.drop(['trip_creation_hour', 'Cart', 'FTL'], axis=1).corr(),
annot=True)
```

Out[261…
```
<Axes: >
```



- `All the numerical variables are highly correlated`

Hypothesis Testing

In [95]:
```
def result(stat, p_value, alpha):
    if p_value < alpha:
        print("Reject Null Hypothesis. Pval is ", p_value ,". Hence, it is concl
    else:
        print("Fail to Reject Null Hypothesis. Pval is ", round(p_value,2) ,". H
```

In [96]:
```
def result_ttest(stat, p_value, alpha):
    print('T-Stat: ', round(stat,2), 'P-Val: ', p_value)
    if p_value < 0.05:
        print('Reject Null Hypothesis. Hence, Average delivery time of Cart is l
    else:
        print('Fail to Reject Null Hypothesis. Hence, Average delivery time of C
```

In [97]:
```
alpha = 0.05
```

actual_time vs osrm_time

Problem Statement:

- `Check for Actual Time and OSRM Time follow same distribution`

Solution Approach:

- `Null Hypothesis: Actual Time and OSRM Time follows same distribution`
- `Alternate Hypothesis: Actual Time and OSRM Time follows does not follow same distribution`
- `Perform Kolmogorov-Smirnov (KS) test to study the distributions`
- `Significance level: 5%`

In [100…

```
ks_stat, p_value = kstest(df_trip['actual_time'], df_trip['osrm_time'])
result(ks_stat, p_value, alpha)
```

```
Reject Null Hypothesis. Pval is  0.0 . Hence, it is concluded that the distributi
ons are non identical
```

In [ ]:

Problem Statement:

- `Check for Mean Actual Time is lesser than OSRM Time`

Solution Approach:

- `Null Hypothesis: u1 = u2`
- `Alternate Hypothesis: u1 < u2`
- `Perform independant t-test`
- `Significance level: 5%`

In [101…

```
stat, p_value = ttest_ind(df_trip['actual_time'], df_trip['osrm_time'], alternat
result(stat, p_value, alpha)
print('T-Stat: ', round(stat,2), 'P-Val: ', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Hence, Mean actual time is lesser than Mean O
else:
    print('Fail to Reject Null Hypothesis. Hence, Mean actual time is not lesser
```

```
Fail to Reject Null Hypothesis. Pval is  1.0 . Hence, it is concluded that the di
stributions are identical
T-Stat:  41.61 P-Val:  1.0
Fail to Reject Null Hypothesis. Hence, Mean actual time is not lesser than Mean O
SRM time
```

In [ ]:

actual_time vs segment_actual_time

Problem Statement:

- `Check for Actual Time and segment actual time follow same distribution`

Solution Approach:

- `Null Hypothesis: Actual Time and segment actual time follows same distribution`
- `Alternate Hypothesis: Actual Time and OSRM Time follows does not follow same distribution`
- `Perform Kolmogorov-Smirnov (KS) test to study the distributions`
- `Significance level: 5%`

In [102…

```
ks_stat, p_value = kstest(df_trip['actual_time'], df_trip['segment_actual_time'])
result(ks_stat, p_value, alpha)
```

```
Fail to Reject Null Hypothesis. Pval is  0.94 . Hence, it is concluded that the d
istributions are identical
```

osrm_time vs segment_osrm_time

Problem Statement:

- `Check for OSRM Time and Segment OSRM Time follow same distribution`

Solution Approach:

- `Null Hypothesis: Segment OSRM Time and OSRM Time follows same distribution`
- `Alternate Hypothesis: Actual Time and OSRM Time follows does not follow same distribution`
- `Perform Kolmogorov-Smirnov (KS) test to study the distributions`
- `Significance level: 5%`

In [103…

```
ks_stat, p_value = kstest(df_trip['osrm_time'], df_trip['segment_osrm_time'])
result(ks_stat, p_value, alpha)
```

```
Reject Null Hypothesis. Pval is  4.90952003845215e-13 . Hence, it is concluded th
at the distributions are non identical
```

actual_distance_to_destination vs osrm_distance

Problem Statement:

- `Check for Actual distance and OSRM distance follow same distribution`

Solution Approach:

- `Null Hypothesis: Actual distance and OSRM distance follows same distribution`
- `Alternate Hypothesis: Actual Time and OSRM Time follows does not follow same distribution`
- `Perform Kolmogorov-Smirnov (KS) test to study the distributions`
- `Significance level: 5%`

In [104…

```
ks_stat, p_value = kstest(df_trip['actual_distance_to_destination'], df_trip['os
result(ks_stat, p_value, alpha)
```

```
Reject Null Hypothesis. Pval is  1.1330229006488255e-214 . Hence, it is concluded
that the distributions are non identical
```

osrm_distance vs segment_osrm_distance

Problem Statement:

- `Check for Segment OSRM distance and OSRM distance follow same distribution`

Solution Approach:

- `Null Hypothesis: Segment OSRM distance and OSRM distance follows same distribution`
- `Alternate Hypothesis: Actual Time and OSRM Time follows does not follow same distribution`
- `Perform Kolmogorov-Smirnov (KS) test to study the distributions`
- `Significance level: 5%`

In [105…

```
ks_stat, p_value = kstest(df_trip['segment_osrm_distance'], df_trip['osrm_distan
result(ks_stat, p_value, alpha)
```

```
Reject Null Hypothesis. Pval is  2.0923396869129475e-12 . Hence, it is concluded
that the distributions are non identical
```

Average actual delivery time

Problem Statement:

- `Average actual delivery time between Cart and FTL is significantly different`

__Solution Approach:__

- `Null Hypothesis: u1=u2`
- `Alternate Hypothesis: u1<u2`
  - `u1 - Average delivery time by cart`
  - `u2 - Average delivery time by Full Truck`
- `Significance level: 5%`
- `Comparison between Average delivery time (*Numerical*) and Route Type (*Category with 2 categories*)`
- `Hence, 2 Sample T Test`

In [106…

```
tstat, p_value = ttest_ind(df_trip[df_trip['Cart']==1]['actual_time'], df_trip[d
print('T-Stat: ', round(tstat,2), 'P-Val: ', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Hence, Average delivery time of Cart is lesse
```

```
else:
    print('Fail to Reject Null Hypothesis. Hence, Average delivery time of Cart
```

```
T-Stat:  -44.99 P-Val:  0.0
Reject Null Hypothesis. Hence, Average delivery time of Cart is lesser than Avera
ge delivery time of Truck
```

Business Insights

- `All numerical variables are right-skewed with outliers, requiring data preprocessing.`
- `The dataset spans 22 days and contains only domestic deliveries.`
- `Null values are present in the source_name and destination_name fields, needing to be addressed.`
- `Data is split into 70/30 for training/testing.`
- `Actual time/OSRM time ratio (factor) and segment_actual_time/segment_osrm_time ratio (segment_factor) are key variables.`
- `The longest delivery route is between IND284403 and IND474003, consuming more time.`
- `Night trips dominate delivery creation, but take longer.`
- `Bengaluru, Hyderabad, and Mumbai handle the majority of intra-city deliveries.`
- `For intercity deliveries, the Bhiwandi <-> Mumbai and Gurgaon <-> Delhi corridors are the busiest.`
- `Maharashtra, Karnataka, and Tamil Nadu handle the majority of intra-state deliveries.`
- `Delhi <-> Haryana and Haryana <-> UP are the busiest inter-state corridors.`
- `Actual time and segment actual time follow the same distribution, but OSRM time and segment OSRM time do not, indicating inconsistencies in the OSRM data.`
- `OSRM distance and segment OSRM distance follow different distributions, highlighting discrepancies in dataset accuracy`
- `Trucks take longer on average compared to carts, suggesting trucks are used for longer routes`

Recommendations

- **Improve Overnight Trip Scheduling:**
  - `Overnight trips take more time to complete.`
  - `To optimize, schedule multiple overnight trips to minimize delays.`
- **Focus on Tier II and Tier III Cities:**
  - `Since the majority of deliveries occur between metro cities, redirect focus to develop logistics in tier II and III cities for future growth.`
- **Enhance Logistics for Northern-Eastern Trips:**
  - `Although northern-southern trips cover long distances, northern-eastern trips consume more time.`
  - `Invest in better logistics infrastructure to improve delivery efficiency in the eastern regions.`
- **Recalibrate OSRM Time Algorithm:**
  - `Actual Time and OSRM Time doesn't follow same distribution, indicates that there is a mismatch in the OSRM time calcualtion algorithm.`
  - `Infact, mean actual time is greater than mean OSRM time, indicates the algorithm underestimates the delivery time`
  - `OSRM Time estimation algorithm to be recalibrated to better plan the deliveries`
- **Recalibrate OSRM Distance Algorithm:**
  - `Actual Distance and OSRM Distance doesn't follow same distribution, indicates that there is a mismatch in the OSRM distance calcualtion algorithm`
  - `OSRM Distance estimation algorithm to be recalibrated to better plan the deliveries`

In [ ]: