Problem Statement

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

How can you help here?

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

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import stats
```

In [2]:
```
delhivery = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets

delhivery.head(5)
```

Out[2]:

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

5 rows × 24 columns

Shape, Structure and Missing Values

In [3]:

```
delhivery.shape
```

Out[3]:

```
(144867, 24)
```

In [4]:

```
delhivery.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]:
```
delhivery.isna().sum()
```

Out[5]:
```
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

Analysing Dataset after feature creation

In [6]:
```
delhivery.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
```

Changing time format to standard datetime

In [7]:
```
delhivery["trip_creation_time"] = pd.to_datetime(delhivery["trip_creation_time"])
delhivery["od_start_time"] = pd.to_datetime(delhivery["od_start_time"])
delhivery["od_end_time"] = pd.to_datetime(delhivery["od_end_time"])
```

lets check for which year & months do we have the data for...

In [8]:
```
delhivery["trip_creation_time"].dt.month_name().value_counts()
```

Out[8]:
```
September    127349
October       17518
Name: trip_creation_time, dtype: int64
```

In [9]:
```
delhivery["trip_creation_time"].dt.year.value_counts()
```

Out[9]:
```
2018    144867
Name: trip_creation_time, dtype: int64
```

In [10]:
```
delhivery["trip_creation_time"].dt.day_name().value_counts()
```

Out[10]:
```
Wednesday    26732
Thursday     20481
Friday       20242
Tuesday      19961
Saturday     19936
Monday       19645
Sunday       17870
Name: trip_creation_time, dtype: int64
```

NOTE: Datepoints are from the month of September and October of year 2018

No. of Unique Categories of Features

In [11]:
```
delhivery.nunique()
```

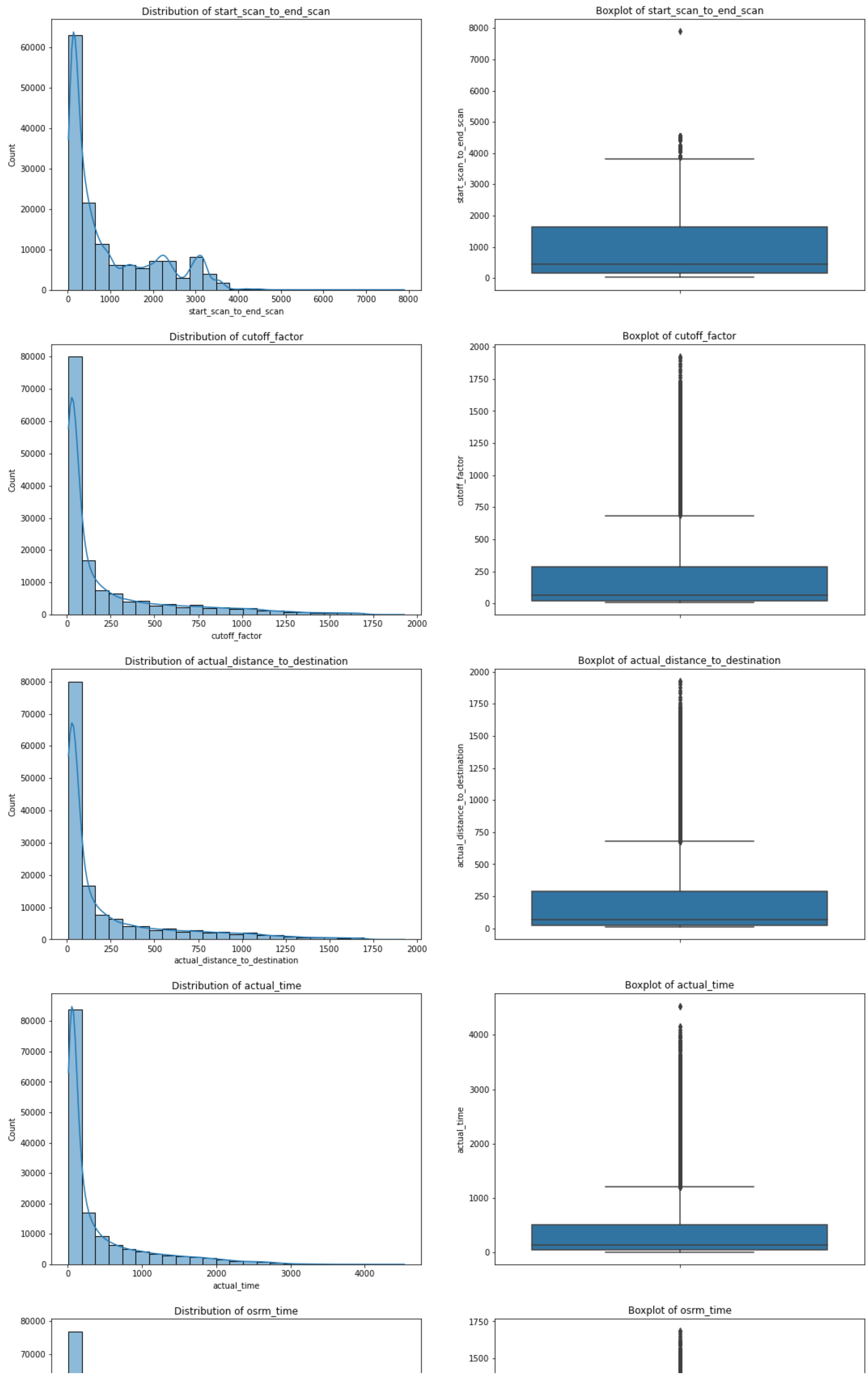
Out[11]:
```
data                                  2
trip_creation_time                14817
route_schedule_uuid                1504
route_type                            2
trip_uuid                         14817
source_center                      1508
source_name                        1498
destination_center                 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
```

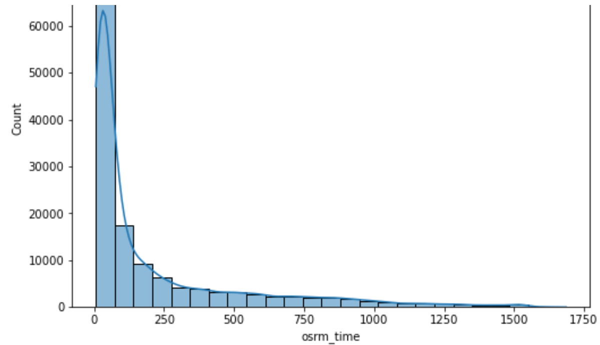
- There are total **14817** different trips of data available
- There are **1508** unique source_center
- There are **1481** unique destination_center
- There are total **1504** delivery routes

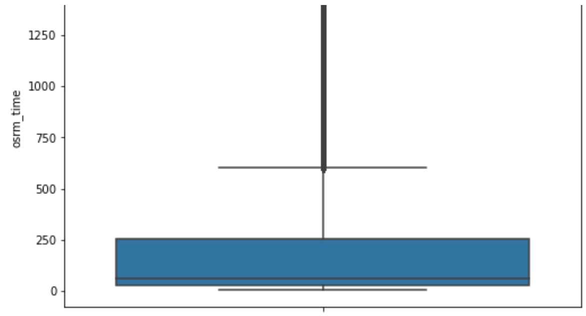
Visual Analysis

Univariate Continuous

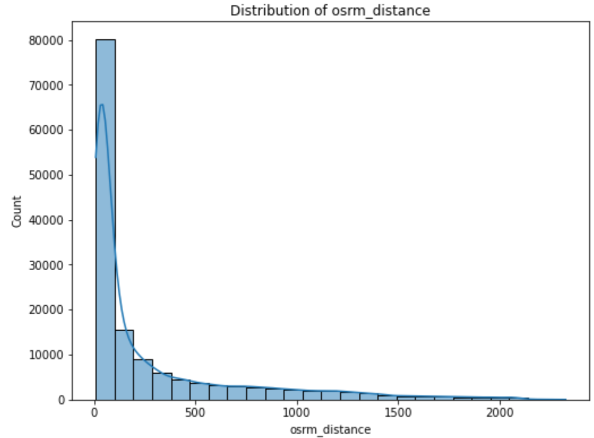
In [12]:
```
num_vars = delhivery.select_dtypes(include=np.number).columns.tolist()

fig, ax = plt.subplots(nrows=11, ncols=2, figsize=(18, 80))

for i in range(len(num_vars)):

    sns.histplot(x=delhivery[num_vars[i]], kde=True, bins = 25, ax=ax[i, 0])
    ax[i, 0].set_title(f"Distribution of {num_vars[i]}")

    sns.boxplot(y = delhivery[num_vars[i]], ax=ax[i, 1], data=delhivery)
    ax[i, 1].set_title(f"Boxplot of {num_vars[i]}")

plt.show()
```


Distribution of start_scan_to_end_scan
Boxplot of start_scan_to_end_scan
Count
start_scan_to_end_scan
Distribution of cutoff_factor
Boxplot of cutoff_factor
cutoff_factor
Distribution of actual_distance_to_destination
Boxplot of actual_distance_to_destination
actual_distance_to_destination
Distribution of actual_time
Boxplot of actual_time
actual_time
Distribution of osrm_time
Boxplot of osrm_time

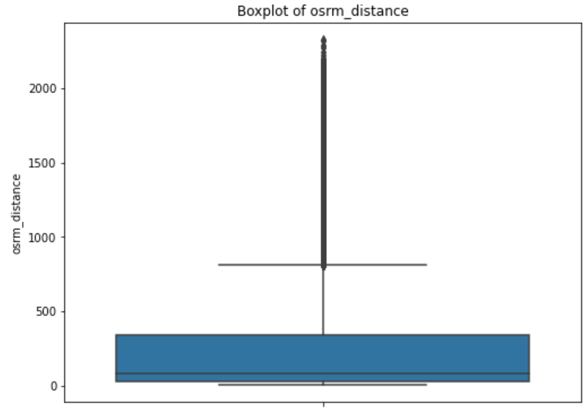




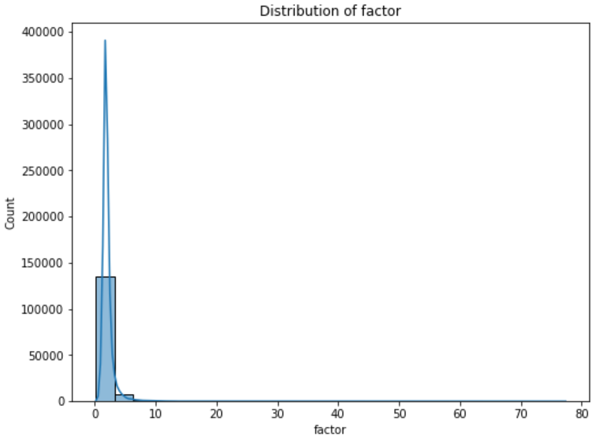
Distribution of osrm_distance



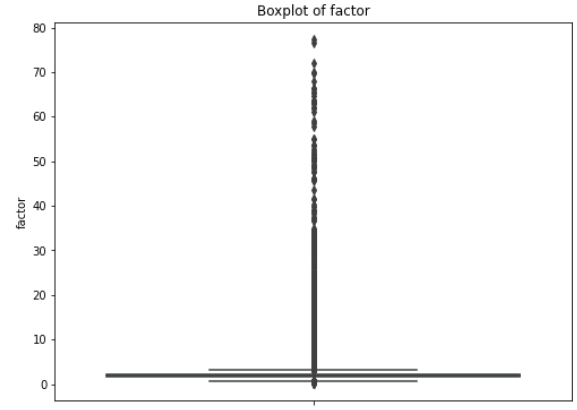
Boxplot of osrm_distance



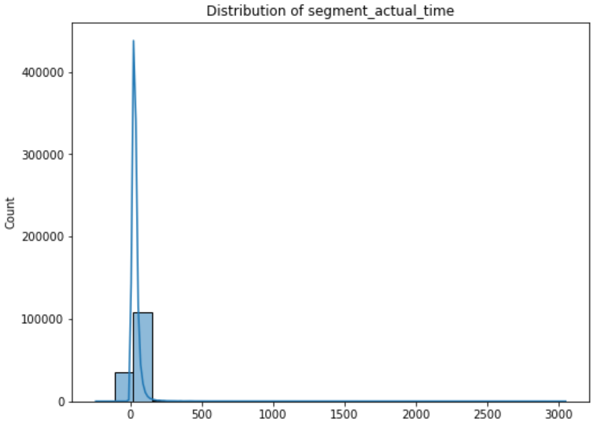
Distribution of factor



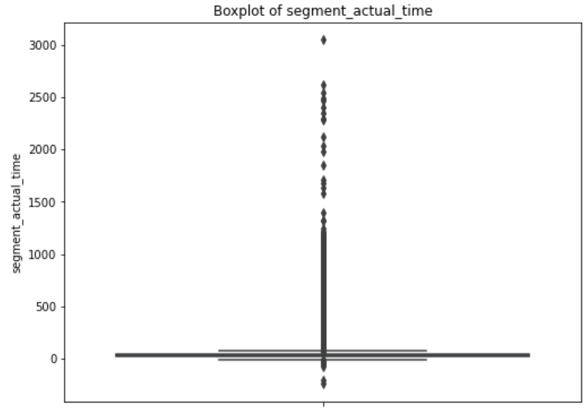
Boxplot of factor



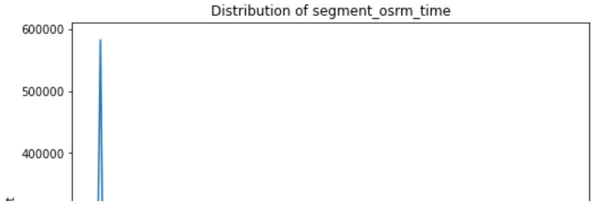
Distribution of segment_actual_time



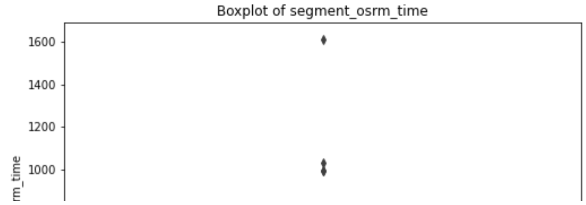
Boxplot of segment_actual_time

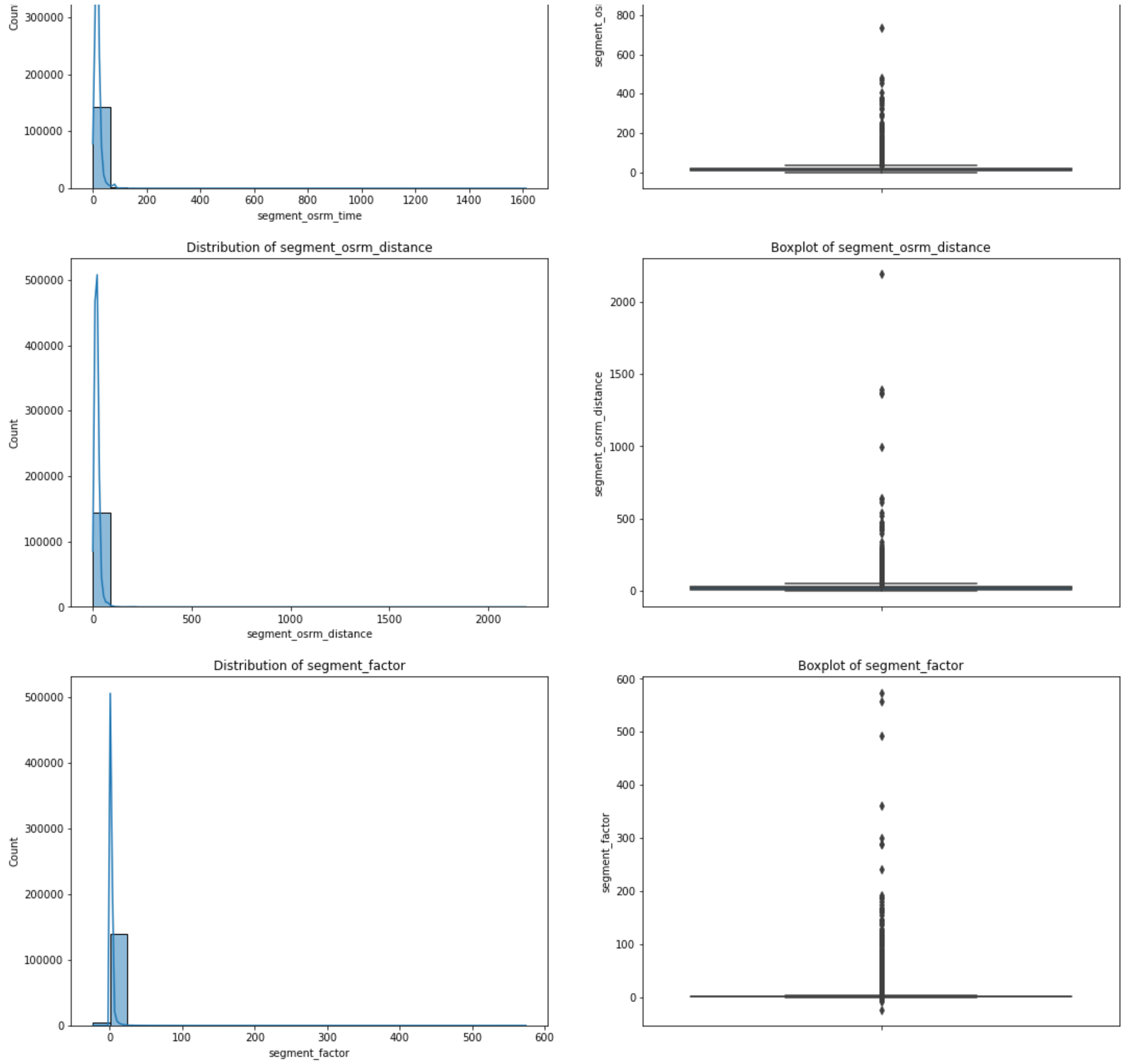


Distribution of segment_osrm_time



Boxplot of segment_osrm_time





Feature Creation: -

Extracting Features like city, state and pincode from source and destination name columns:-

In [13]:
```
delhivery["source_city"] = delhivery["source_name"].str.split(" ",n=1,expand=True)[
delhivery["destination_city"] = delhivery["destination_name"].str.split(" ",n=1,exp

delhivery["source_state"] = delhivery["source_name"].str.split(" ",n=1,expand=True)
delhivery["destination_state"] = delhivery["destination_name"].str.split(" ",n=1,ex

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

Time_taken_btwn_odstart_and_od_end

In [14]:
```
delhivery["time_taken_btwn_odstart_and_od_end"] = ((delhivery["od_end_time"]-delhiv
```

Converting given time to hours

In [15]:
```
delhivery["start_scan_to_end_scan"] = delhivery["start_scan_to_end_scan"]/60
delhivery["actual_time"] = delhivery["actual_time"]/60
delhivery["osrm_time"] = delhivery["osrm_time"]/60
delhivery["segment_actual_time"] = delhivery["segment_actual_time"]/60
delhivery["segment_osrm_time"] = delhivery["segment_osrm_time"]/60
```

In [16]:
```
delhivery.head()
```

Out[16]:

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

5 rows × 31 columns

Data Cleaning

In [17]:
```
delhivery["source_state"] = delhivery["source_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",
                                                           })
```

In [18]:
```
delhivery["destination_state"] = delhivery["destination_state"].replace({"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"
                                                           })
```

In [19]:
```
delhivery["destination_city"].replace({"del":"Delhi", "Bangalore":"Bengaluru", "AMD
delhivery["source_city"].replace({"del":"Delhi", "Bangalore":"Bengaluru", "AMD":"Ah
```

Creating Feature - [Source city + state & Destination city + state

In [20]:
```
delhivery["source_city_state"] = delhivery["source_city"] + " " + delhivery["source
delhivery["destination_city_state"] = delhivery["destination_city"] + " " + delhive
```

In [21]:
```
delhivery["source_city_state"].nunique()
```

Out[21]: 1249

In [22]:
```
delhivery["destination_city_state"].nunique()
```

Out[22]: 1242

In [23]:
```
delhivery["source_state"].nunique()
```

Out[23]: 33

In [24]:
```
delhivery["destination_state"].nunique()
```

Out[24]:
```
32
```

Dropping Unnecessary columns

In [25]:
```
data = delhivery.copy()
```

In [26]:
```
data.shape
```

Out[26]:
```
(144867, 33)
```

In [27]:
```
data.drop(
    ['source_center',"source_name","destination_center","destination_name","cutoff_
    axis = 1,
    inplace=True
)
```

In [28]:
```
data.shape
```

Out[28]:
```
(144867, 26)
```

Merging of rows and aggregation of fields

In [29]:
```
actual_time = data.groupby(["trip_uuid",
              "start_scan_to_end_scan"])["actual_time"].max().reset_index().groupby

actual_time
```

Out[29]:

	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

14817 rows × 2 columns

In [30]:

```
segment_osrm_time = data[["trip_uuid","segment_osrm_time"]].groupby("trip_uuid")["s
segment_osrm_time
```

Out[30]:

	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

In [31]:

```
segment_actual_time = data.groupby("trip_uuid")["segment_actual_time"].sum().reset_
segment_actual_time
```

Out[31]:

	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

In [32]:
```
osrm_time = data.groupby(["trip_uuid",
              "start_scan_to_end_scan"])["osrm_time"].max().reset_index().groupby("
osrm_time
```

Out[32]:

	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

In [33]:
```
time_taken_btwn_odstart_and_od_end = data.groupby("trip_uuid")["time_taken_btwn_ods
time_taken_btwn_odstart_and_od_end
```

Out[33]:

	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

In [34]:
```
time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"] = time_tak
time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"]
```

Out[34]:
```
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
```

In [35]:
```
start_scan_to_end_scan = ((data.groupby("trip_uuid")["start_scan_to_end_scan"].uniq
start_scan_to_end_scan
```

Out[35]:

	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

In [36]:
```
start_scan_to_end_scan["start_scan_to_end_scan"] = start_scan_to_end_scan["start_sc
start_scan_to_end_scan["start_scan_to_end_scan"]
```

Out[36]:
```
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
```

In [37]:
```
osrm_distance = data.groupby(["trip_uuid",
              "start_scan_to_end_scan"])["osrm_distance"].max().reset_index().group

osrm_distance
```

Out[37]:

	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

In [38]:

```
actual_distance_to_destination = data.groupby(["trip_uuid",
              "start_scan_to_end_scan"])["actual_distance_to_destination"].max().re

actual_distance_to_destination
```

Out[38]:

	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

In [39]:

```
segment_osrm_distance = data[["trip_uuid",
                              "segment_osrm_distance"]].groupby("trip_uuid")["segme
```

```
segment_osrm_distance
```

Out[39]:

	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

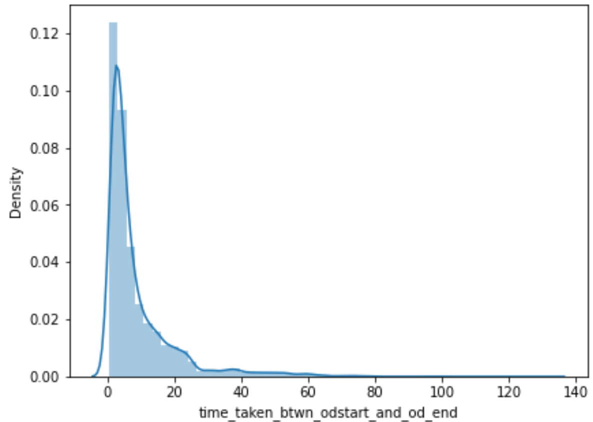
Hypothesis Test

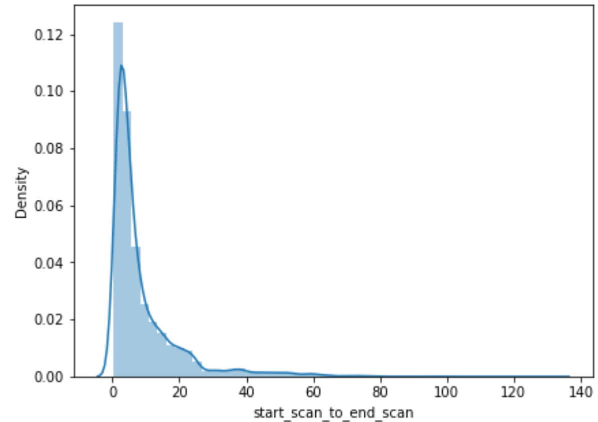
Analysing TimeTaken Between OdStart and OdEnd time & StartScanToEndScan:

`H0: Mean of time taken betweenn trip end ans start time = Mean of start and end scan time`
`Ha: Mean of time taken betweenn trip end ans start time != Mean of start and end scan time`

In [40]:

```
plt.figure(figsize=(15,5))
plt.subplot(121)
sns.distplot((time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_en
plt.subplot(122)
sns.distplot((start_scan_to_end_scan["start_scan_to_end_scan"]))

plt.show()
```





In [41]:
```
# KS Test to check the similarity of distribution of these two.
```

In [42]:
```
ks_test, p_value = stats.ks_2samp(time_taken_btwn_odstart_and_od_end["time_taken_bt
                               ,start_scan_to_end_scan["start_scan_to_end_scan"])
```

In [43]:
```
# Ho: The distribution are similar
# Ha: The disbutions are different

if p_value < 0.05:
    print("Reject Ho: The distribution are different.")
else :
    print("Fail to reject Ho: The distribution is same.")
```

```
Fail to reject Ho: The distribution is same.
```

In [44]:
```
for i in range(5):
    print(stats.ttest_ind((time_taken_btwn_odstart_and_od_end["time_taken_btwn_odst
                                ,(start_scan_to_end_scan["start_scan_to_end_scan"].sample(3000))))
```

```
Ttest_indResult(statistic=1.1432627697854592, pvalue=0.2529751360865033)
Ttest_indResult(statistic=0.306754745397043, pvalue=0.7590407047165052)
Ttest_indResult(statistic=0.5295134749688785, pvalue=0.5964688965540286)
Ttest_indResult(statistic=-0.4001926456184752, pvalue=0.689028882833091)
Ttest_indResult(statistic=-0.7273242613388564, pvalue=0.46705572650733385)
```

- **from 2 sample t-test ,we can also conclude that Average time_taken_btwn_odstart_and_od_end for population is also equal to Average start_scan_to_end_scan for population.**

In [45]:
```
time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"].mean(),tim
```

Out[45]:
```
(8.861857235305067, 10.981665759990623)
```

In [46]:
```
start_scan_to_end_scan["start_scan_to_end_scan"].mean(),start_scan_to_end_scan["sta
```

Out[46]:
```
(8.835777597804325, 10.97628639143973)
```

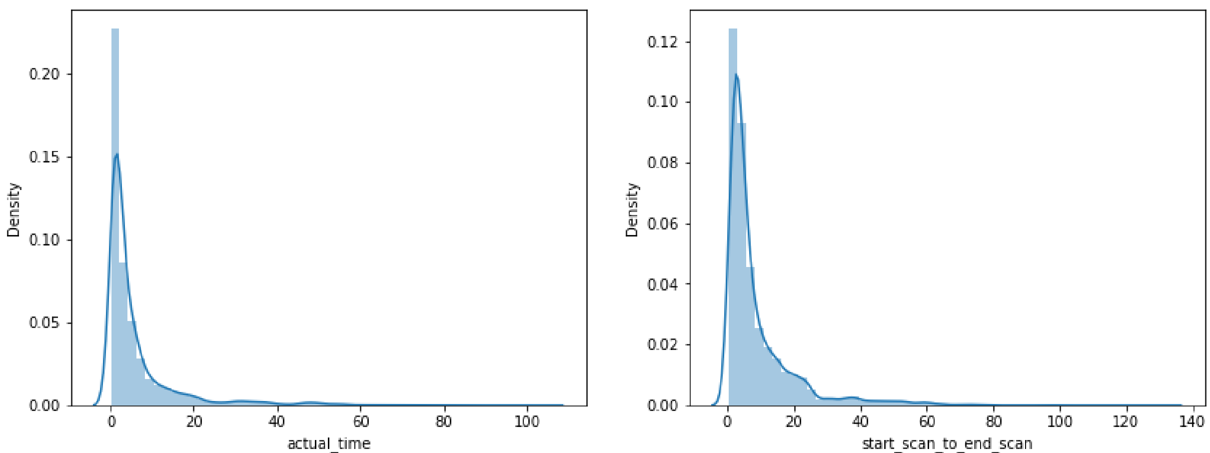
- **variance and means both are closly similar for scan time and trip start and end time taken**

Analysing Actual Time taken to complete the delivery & start-scan-end-scan

`H0: Mean of start and end scan time <= Mean of Actual time taken to complete delivery`

`Ha: Mean of start and end scan time > Mean of Actual time taken to complete delivery`

In [47]:
```
plt.figure(figsize=(14,5))
plt.subplot(121)
sns.distplot((actual_time["actual_time"]))
plt.subplot(122)
sns.distplot((start_scan_to_end_scan["start_scan_to_end_scan"]))

plt.show()
```



In [48]:
```
stats.ks_2samp(actual_time["actual_time"],start_scan_to_end_scan["start_scan_to_end
```

Out[48]:
```
KstestResult(statistic=0.27387460349598436, pvalue=0.0)
```

In [49]:
```
for i in range(7):
    print(stats.ttest_ind((actual_time["actual_time"].sample(3000))
                ,(start_scan_to_end_scan["start_scan_to_end_scan"].sample(3000)),al
```

```
Ttest_indResult(statistic=-12.598785737735636, pvalue=3.048395456070754e-36)
Ttest_indResult(statistic=-10.37228721307964, pvalue=2.693595137009714e-25)
Ttest_indResult(statistic=-9.270063808880982, pvalue=1.2707131468623415e-20)
Ttest_indResult(statistic=-11.809556477065186, pvalue=3.915069776632231e-32)
Ttest_indResult(statistic=-11.172975869087509, pvalue=5.301082343479836e-29)
Ttest_indResult(statistic=-11.45321461575804, pvalue=2.3221474195556868e-30)
Ttest_indResult(statistic=-10.977598253148816, pvalue=4.494398967842258e-28)
```

In [50]:
```
actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```

Out[50]:
```
(5.945176711435117, 9.35554782297388)
```

In [51]:
```
start_scan_to_end_scan["start_scan_to_end_scan"].mean(),start_scan_to_end_scan["sta
```

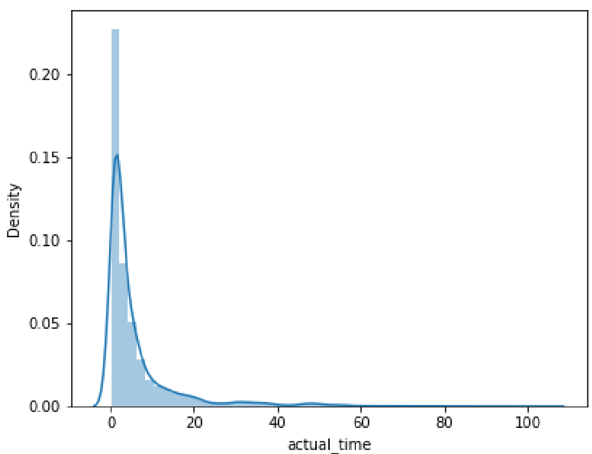
Out[51]: (8.835777597804325, 10.97628639143973)

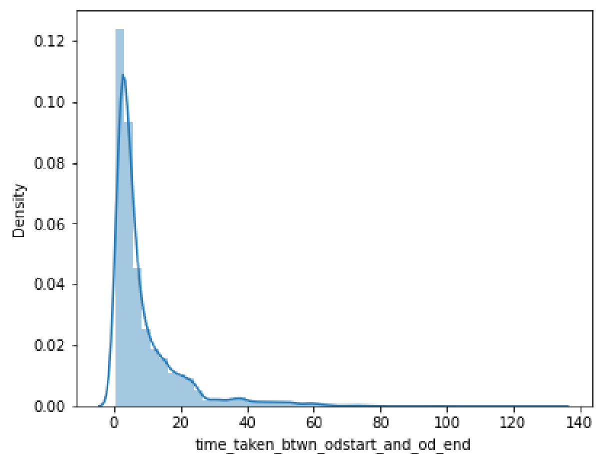
Analysing Actual Time & TimeTaken between start and end trip time.

`H0: Mean of Actual time taken to complete delivery = Mean of time taken betweenn trip end and start time`

`Ha: Mean of Actual time taken to complete delivery != Mean of time taken betweenn trip end and start time`

In [52]:
```
plt.figure(figsize=(14,5))
plt.subplot(121)
sns.distplot((actual_time["actual_time"]))
plt.subplot(122)
sns.distplot((time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_en

plt.show()
```





In [53]:
```
stats.ks_2samp(actual_time["actual_time"],time_taken_btwn_odstart_and_od_end["time_
```

Out[53]: KstestResult(statistic=0.2765067152594992, pvalue=0.0)

In [54]:
```
for i in range(5):
    print(stats.ttest_ind((actual_time["actual_time"].sample(1000))
                ,(time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_o
```

```
Ttest_indResult(statistic=-4.868208587660947, pvalue=1.2144968826076986e-06)
Ttest_indResult(statistic=-6.431191633146856, pvalue=1.5796442072073778e-10)
Ttest_indResult(statistic=-6.600964301276705, pvalue=5.218162353450626e-11)
Ttest_indResult(statistic=-5.595619473347511, pvalue=2.5012415346797522e-08)
Ttest_indResult(statistic=-6.090408405756541, pvalue=1.3471996614358867e-09)
```

- `from above kstest of distribution and two sample ttest ,`

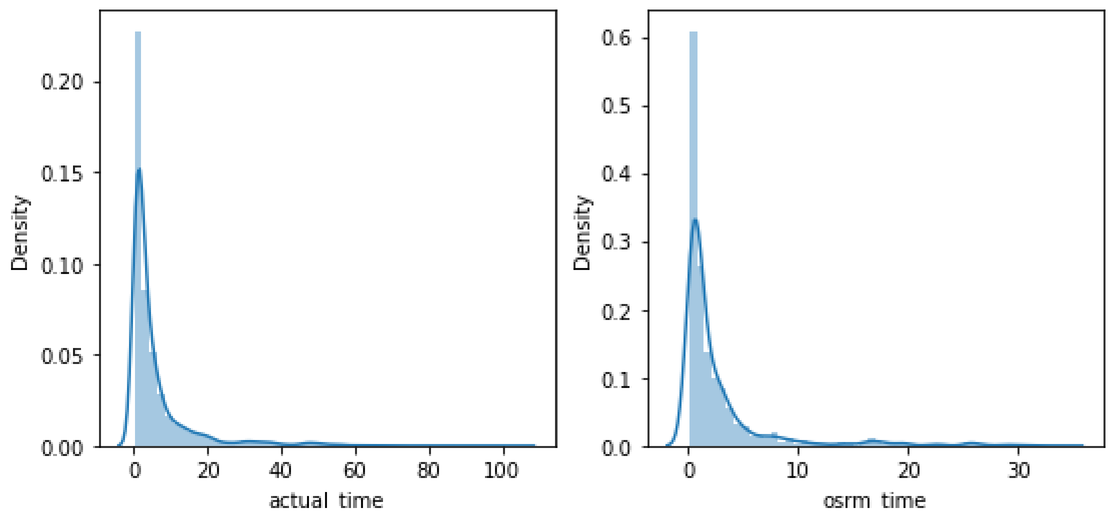
`we can conclude that population mean Actual time taken to complete delivery and population mean time_taken_btwn_od_start_and_od_end are also not same.`

Analysing Actual Time taken to complete delivery from source to destination hub & OSRM measured time :

`H0: Mean of OSRM time >= Mean of Actual time taken to complete delivery`

`Ha: Mean of OSRM time < Mean of Actual time taken to complete delivery`

In [55]:
```
plt.figure(figsize=(9,4))
plt.subplot(121)
sns.distplot(((actual_time["actual_time"])))
plt.subplot(122)
sns.distplot(((osrm_time["osrm_time"])))

plt.show()
```



In [56]:
```
stats.ks_2samp(actual_time["actual_time"],
               osrm_time["osrm_time"])
```

Out[56]:
```
KstestResult(statistic=0.2945265573327934, pvalue=0.0)
```

In [57]:
```
for i in range(5):
    print(stats.ttest_ind(actual_time["actual_time"].sample(5000),
               osrm_time["osrm_time"].sample(5000),alternative='greater'))
```

```
Ttest_indResult(statistic=22.494019075638633, pvalue=1.1917516763449016e-109)
Ttest_indResult(statistic=21.456812550110058, pvalue=3.453215594658714e-100)
Ttest_indResult(statistic=21.282098350070285, pvalue=1.236623389107801e-98)
Ttest_indResult(statistic=22.370467761942866, pvalue=1.6766284547665366e-108)
Ttest_indResult(statistic=21.432020170637404, pvalue=5.747044341986743e-100)
```

- `from two sample ttest can conclude , that population mean actual time taken to complete delivert from source to warehouse and orsm estimate mean time for population are not same.`

- `actual time is higher than the osrm estimated time for delivery.`

In [58]:
```
actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```
Out[58]:
```
(5.945176711435117, 9.35554782297388)
```

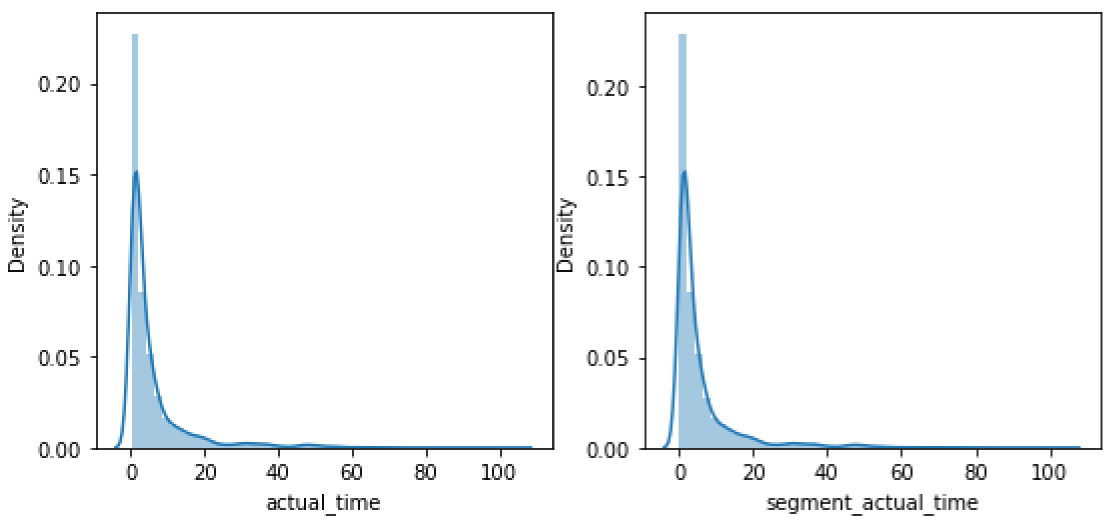
In [59]:
```
osrm_time["osrm_time"].mean(),osrm_time["osrm_time"].std()
```
Out[59]:
```
(2.697313896200314, 4.537654251845703)
```

Analysing Actual Time taken to complete delivery from source to destination hub & Segment Actual Time :

`H0: Actual time = segment actual time`

`Ha: Actual time != segment actual time`

In [60]:
```
plt.figure(figsize=(9,4))
plt.subplot(121)
sns.distplot(((actual_time["actual_time"])))
plt.subplot(122)
sns.distplot(((segment_actual_time["segment_actual_time"])))

plt.show()
```



In [61]:
```
for i in range(7):
    print(stats.ttest_ind((actual_time["actual_time"].sample(3000)),
                          (segment_actual_time["segment_actual_time"].sample(3000))))
```
```
Ttest_indResult(statistic=-0.45575823788599884, pvalue=0.648580337132706)
Ttest_indResult(statistic=-0.1022982509950364, pvalue=0.9185233631789194)
Ttest_indResult(statistic=-0.4775339259306228, pvalue=0.6329993690757283)
Ttest_indResult(statistic=0.6376797377153542, pvalue=0.5237065193938589)
Ttest_indResult(statistic=-3.0268658899208165, pvalue=0.0024815275044342898)
Ttest_indResult(statistic=1.067843799842909, pvalue=0.2856339527814263)
Ttest_indResult(statistic=0.06939217493337652, pvalue=0.9446797706181737)
```

`from two sample ttest , we can conclude that`

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

In [62]:
```
actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```

Out[62]:
```
(5.945176711435117, 9.35554782297388)
```

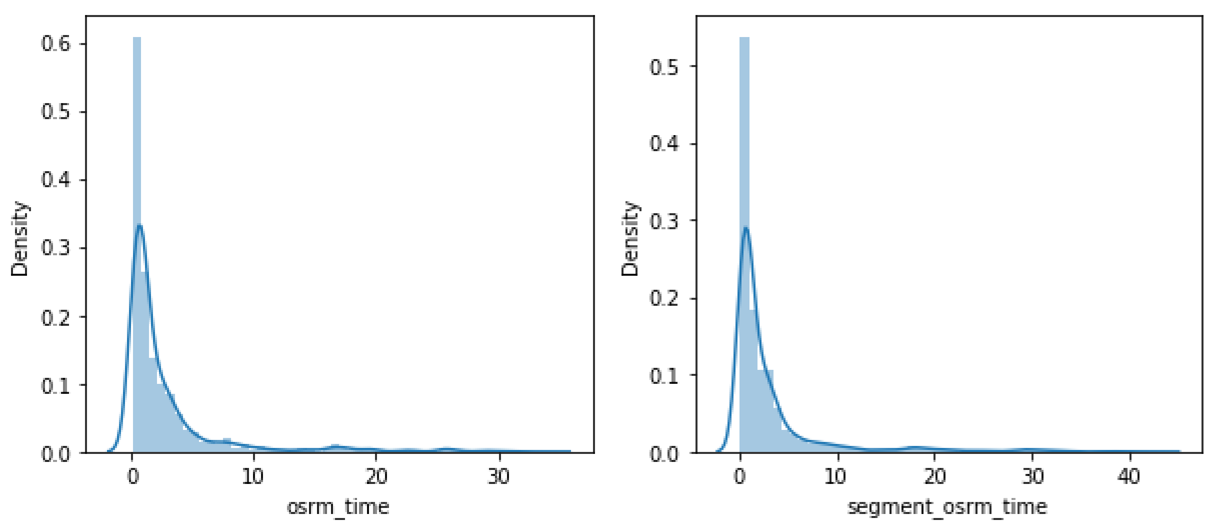
In [63]:
```
segment_actual_time["segment_actual_time"].mean(),segment_actual_time["segment_actu
```

Out[63]:
```
(5.898204764797215, 9.270799413152762)
```

Analysing osrm Time & segment-osrm-time :

`Ho: segment actual time <= OSRM time`
`Ha: segment actual time > OSRM time`

In [64]:
```
plt.figure(figsize=(10,4))
plt.subplot(121)
sns.distplot(((osrm_time["osrm_time"])))
plt.subplot(122)
sns.distplot(((segment_osrm_time["segment_osrm_time"])))

plt.show()
```



In [65]:
```
for i in range(7):
    print(stats.ttest_ind((osrm_time["osrm_time"].sample(3000)),
                (segment_osrm_time["segment_osrm_time"].sample(3000)),alternative =
```

```
Ttest_indResult(statistic=-2.300674606130008, pvalue=0.010722087307075653)
Ttest_indResult(statistic=-2.4632535805904965, pvalue=0.006898063709777681)
Ttest_indResult(statistic=-1.751397913041363, pvalue=0.039964250785561505)
Ttest_indResult(statistic=-2.9864082565762837, pvalue=0.00141708254845603)
Ttest_indResult(statistic=-1.8717649067283955, pvalue=0.03064388323629034)
Ttest_indResult(statistic=-4.232574825669032, pvalue=1.1723935355432782e-05)
Ttest_indResult(statistic=-3.823874374924718, pvalue=6.635141361670147e-05)
```

`from ttest , we can conclude that`

- average of osrm Time & segment-osrm-time for population is not same.
- Population Mean osrm time is less than Population Mean segment osrm time.

In [66]:
```
osrm_time["osrm_time"].mean(),osrm_time["osrm_time"].std()
```

Out[66]:
```
(2.697313896200314, 4.537654251845703)
```

In [67]:
```
segment_osrm_time["segment_osrm_time"].mean(),segment_osrm_time["segment_osrm_time"
```

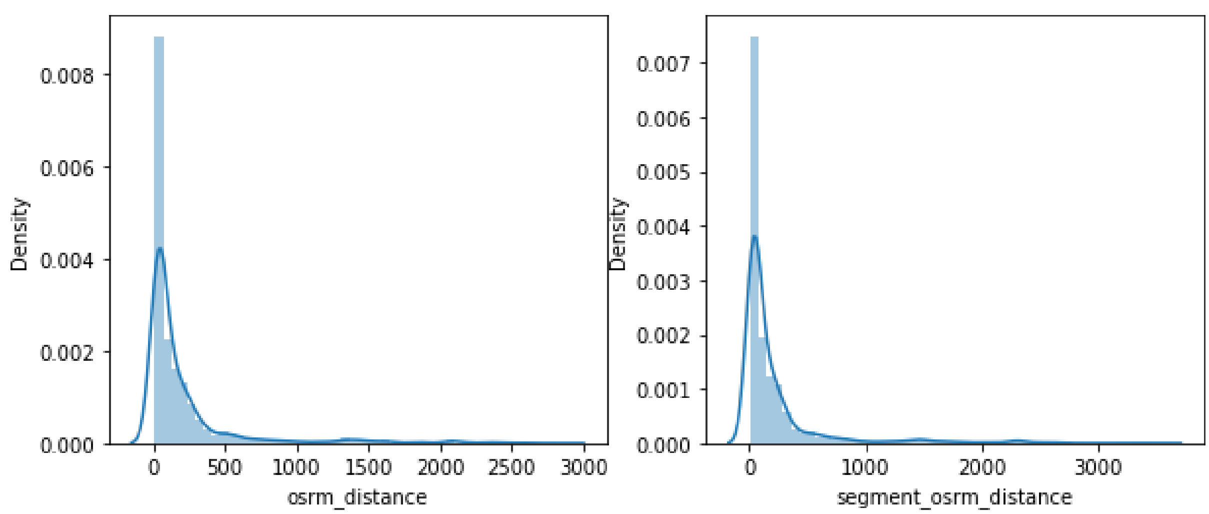
Out[67]:
```
(3.0158297901059705, 5.242367441693007)
```

Analysing and Visulizing OSRM Estimated distance and Segment-osrm-distance :

`H0 : Segment OSRM distnace <= OSRM distnace`

`Ha : Segment OSRM distnace > OSRM distnace`

In [68]:
```
plt.figure(figsize=(10,4))
plt.subplot(121)
sns.distplot(((osrm_distance["osrm_distance"])))
plt.subplot(122)
sns.distplot(((segment_osrm_distance["segment_osrm_distance"])))

plt.show()
```



In [69]:
```
stats.ks_2samp(osrm_distance["osrm_distance"],segment_osrm_distance["segment_osrm_d
```

Out[69]: KstestResult(statistic=0.03948167645272321, pvalue=1.8042208791084262e-10)

In [70]:
```
for i in range(7):
    print(stats.ttest_ind(osrm_distance["osrm_distance"].sample(5000),
                segment_osrm_distance["segment_osrm_distance"].sample(5000),alternat
```

```
Ttest_indResult(statistic=-2.1862540143149336, pvalue=0.014410066042621272)
Ttest_indResult(statistic=-2.0080746135611265, pvalue=0.022331106104027557)
Ttest_indResult(statistic=-1.493438132740869, pvalue=0.06767704666730436)
Ttest_indResult(statistic=-2.492889253041487, pvalue=0.006343443831968196)
Ttest_indResult(statistic=-1.575961523911626, pvalue=0.05753315481831684)
Ttest_indResult(statistic=-1.5943003940038654, pvalue=0.055450189251080015)
Ttest_indResult(statistic=-3.340534481099891, pvalue=0.0004196174146864306)
```

In [71]:
```
osrm_distance["osrm_distance"].mean(),osrm_distance["osrm_distance"].std()
```

Out[71]: (204.83672531551625, 370.74927471335496)

In [72]:
```
segment_osrm_distance["segment_osrm_distance"].mean(),segment_osrm_distance["segmen
```

Out[72]: (223.20116128771042, 416.6283742907418)

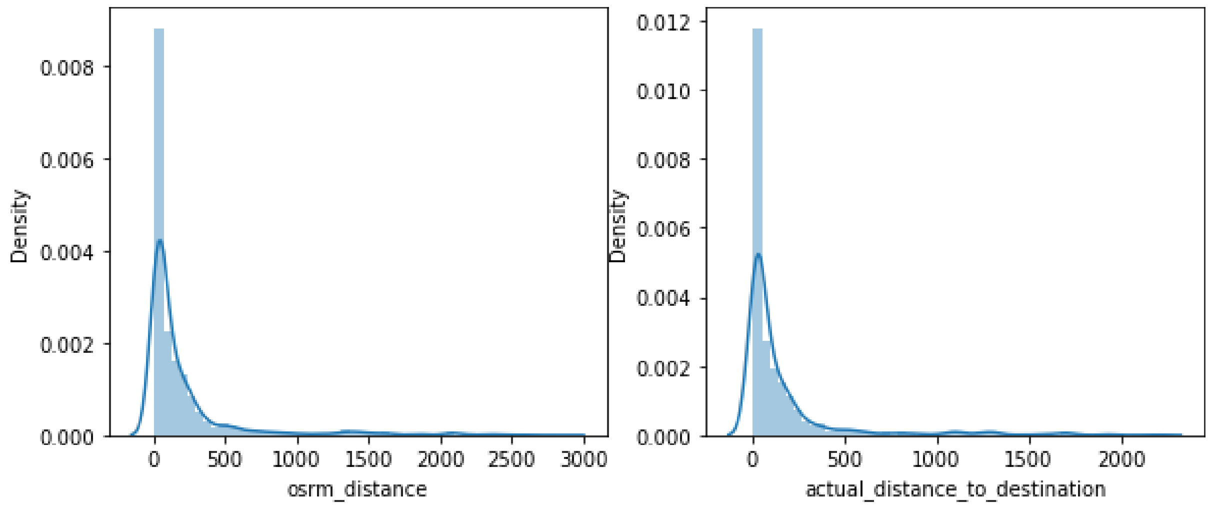
- `from KS test , we can conclude the distributions of segment osrm distance and osrm distnace are not same!`
- `from two sample one sided ttest, we can conclude: Average of osrm distance for population is less than average of segment osrm distnace`

Analysing and Visulizing OSRM Estimated distance and Actual Distance between source and destination warehouse :

`H0 : Mean OSRM distance <= Mean Actual distnace`

`Ha : Mean OSRM distance > Mean Actual distnace`

In [73]:
```
plt.figure(figsize=(10,4))
plt.subplot(121)
sns.distplot(((osrm_distance["osrm_distance"])))
plt.subplot(122)
sns.distplot(((actual_distance_to_destination["actual_distance_to_destination"])))

plt.show()
```



In [74]:
```
stats.ks_2samp(osrm_distance["osrm_distance"],actual_distance_to_destination["actua
```

Out[74]:
```
KstestResult(statistic=0.11837753931295136, pvalue=6.578385372142345e-91)
```

In [75]:
```
for i in range(5):
    print(stats.ttest_ind(osrm_distance["osrm_distance"].sample(5000),
                          actual_distance_to_destination["actual_distance_to_destination"].sam
```

```
Ttest_indResult(statistic=6.239357308538839, pvalue=2.2858764405251027e-10)
Ttest_indResult(statistic=6.566959217156209, pvalue=2.6952585199025075e-11)
Ttest_indResult(statistic=6.148724631875202, pvalue=4.054728279627574e-10)
Ttest_indResult(statistic=5.388817463016214, pvalue=3.6267458514115e-08)
Ttest_indResult(statistic=5.47724139569094, pvalue=2.212369791457411e-08)
```

From left sided ttest , we can conclude

- for population OSRM estimated distance is higher than the actual distance from source to destination warehouse.

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 are 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 segment 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 source to destination warehouse.

In [76]:
```
osrm_distance["osrm_distance"].mean(),osrm_distance["osrm_distance"].std()
```

Out[76]: (204.83672531551625, 370.74927471335496)

In [77]:
```
actual_distance_to_destination["actual_distance_to_destination"].mean(),actual_dist
```

Out[77]: (164.4733217454422, 305.5408288910492)

Merging

In [78]:
```
distances = segment_osrm_distance.merge(actual_distance_to_destination.merge(osrm_d
distances
```

Out[78]:

	trip_uuid	segment_osrm_distance	actual_distance_to_destination	osrm_d
0	trip-153671041653548748	1320.4733	824.732854	9
1	trip-153671042288605164	84.1894	73.186911	
2	trip-153671043369099517	2545.2678	1932.273969	23
3	trip-153671046011330457	19.8766	17.175274	
4	trip-153671052974046625	146.7919	127.448500	1
...	...	...	...	
14812	trip-153861095625827784	64.8551	57.762332	
14813	trip-153861104386292051	16.0883	15.513784	
14814	trip-153861106442901555	104.8866	38.684839	
14815	trip-153861115439069069	223.5324	134.723836	1
14816	trip-153861118270144424	80.5787	66.081533	

14817 rows × 4 columns

In [79]:

```
time = segment_osrm_time.merge(osrm_time.merge(segment_actual_time.merge(actual_tim
                                          on="trip_uuid",
                                          ),on="trip_uuid"),on="trip_uuid"),on="trip

time
```

Out[79]:

	trip_uuid	segment_osrm_time	osrm_time	segment_actual_time	actual_t
0	trip-153671041653548748	16.800000	12.383333	25.800000	26.033
1	trip-153671042288605164	1.083333	1.133333	2.350000	2.383
2	trip-153671043369099517	32.350000	29.016667	55.133333	55.783
3	trip-153671046011330457	0.266667	0.250000	0.983333	0.983
4	trip-153671052974046625	1.916667	1.950000	5.666667	5.683
...	...	...	...	...	
14812	trip-153861095625827784	1.033333	1.033333	1.366667	1.383
14813	trip-153861104386292051	0.183333	0.200000	0.350000	0.350
14814	trip-153861106442901555	1.466667	0.900000	4.683333	4.700
14815	trip-153861115439069069	3.683333	3.066667	4.300000	4.400
14816	trip-153861118270144424	1.116667	1.133333	4.566667	4.583

14817 rows × 7 columns

In [80]:

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

Out[80]:

	trip_uuid	segment_osrm_time	osrm_time	segment_actual_time	actual_t
0	trip-153671041653548748	16.800000	12.383333	25.800000	26.033
1	trip-153671042288605164	1.083333	1.133333	2.350000	2.383
2	trip-153671043369099517	32.350000	29.016667	55.133333	55.783
3	trip-153671046011330457	0.266667	0.250000	0.983333	0.983
4	trip-153671052974046625	1.916667	1.950000	5.666667	5.683
...	...	...	...	...	
14812	trip-153861095625827784	1.033333	1.033333	1.366667	1.383
14813	trip-153861104386292051	0.183333	0.200000	0.350000	0.350
14814	trip-153861106442901555	1.466667	0.900000	4.683333	4.700
14815	trip-153861115439069069	3.683333	3.066667	4.300000	4.400
14816	trip-153861118270144424	1.116667	1.133333	4.566667	4.583

14817 rows × 10 columns

Merging Location details and route_type and Numerical data on TripID

In [81]:

```
city = data.groupby("trip_uuid")[["source_city",
                                  "destination_city"]].aggregate({
        "source_city":pd.unique,
    "destination_city":pd.unique,
})

state = data.groupby("trip_uuid")[["source_state",
                                   "destination_state"]].aggregate({
        "source_state":pd.unique,
    "destination_state":pd.unique,
})

city_state = data.groupby("trip_uuid")[["source_city_state",
                                        "destination_city_state"]].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")
```

In [82]:
```
route_type = data.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"
                )
```

In [83]:
```
trip_records = Merged.copy()
```

In [84]:
```
trip_records["route_type"] = trip_records["route_type"].apply(lambda x:x[0])
route_to_merge = data.groupby("trip_uuid")["route_schedule_uuid"].unique().reset_in
trip_records = trip_records.merge(route_to_merge,on="trip_uuid",how="outer")
trip_records["route_schedule_uuid"] = trip_records["route_schedule_uuid"].apply(lam
trip_records
```

Out[84]:

	trip_uuid	route_type	source_city	destination_city	source_city_state
0	trip-153671041653548748	FTL	[Bhopal, Kanpur]	[Kanpur, Gurgaon]	[Bhopal Madhya Pradesh, Kanpur Uttar Pradesh]
1	trip-153671042288605164	Carting	[Tumkur, Doddablpur]	[Doddablpur, Chikblapur]	[Tumkur Karnataka, Doddablpur Karnataka]
2	trip-153671043369099517	FTL	[Bengaluru, Gurgaon]	[Gurgaon, Chandigarh]	[Bengaluru Karnataka, Gurgaon Haryana]
3	trip-153671046011330457	Carting	[Mumbai]	[Mumbai]	[Mumbai Hub Maharashtra]
4	trip-153671052974046625	FTL	[Bellary, Hospet, Sandur]	[Hospet, Sandur, Bellary]	[Bellary Karnataka, Hospet Karnataka, Sandur K...
...	...	...	...	...	...
14812	trip-153861095625827784	Carting	[Chandigarh]	[Zirakpur, Chandigarh]	[Chandigarh Punjab, Chandigarh Chandigarh]
14813	trip-153861104386292051	Carting	[FBD]	[Faridabad]	[FBD Haryana]
14814	trip-153861106442901555	Carting	[Kanpur]	[Kanpur]	[Kanpur Uttar Pradesh]
14815	trip-153861115439069069	Carting	[Tirunelveli, Eral, Tirchchndr, Thisayanvilai,...	[Eral, Tirchchndr, Thisayanvilai, Peikulam, Ti...	[Tirunelveli Tamil Nadu, Eral Tamil Nadu, Tirc...
14816	trip-153861118270144424	FTL	[Hospet, Sandur]	[Sandur, Bellary]	[Hospet Karnataka, Sandur Karnataka]

14817 rows × 18 columns

In [85]:

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

Out[85]:
```
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

In [86]:
```
trip_records["source_city"] = trip_records["source_city"].astype("str").str.strip("
trip_records["destination_city"] = trip_records["destination_city"].astype("str").s
trip_records["source_city_state"] = trip_records["source_city_state"].astype("str")
trip_records["destination_city_state"] = trip_records["destination_city_state"].ast

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

Statistically Analysis

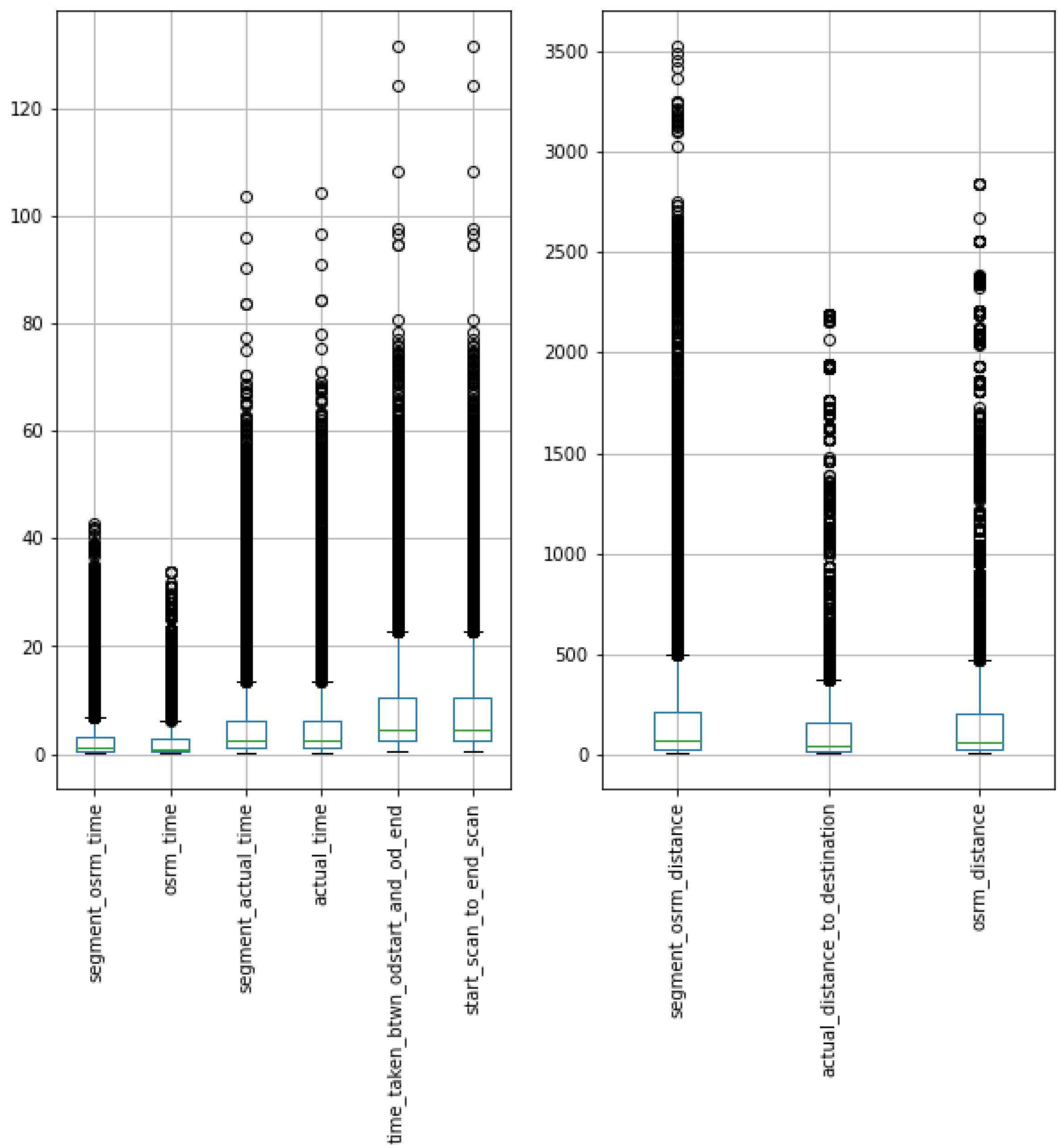
In [87]:
```
trip_records.corr()
```

Out[87]:

	segment_osrm_time	osrm_time	segment_actual_time
segment_osrm_time	1.000000	0.993508	0.953039
osrm_time	0.993508	1.000000	0.957747
segment_actual_time	0.953039	0.957747	1.000000
actual_time	0.953800	0.958613	0.999920
time_taken_btwn_odstart_and_od_end	0.918447	0.926280	0.961096
start_scan_to_end_scan	0.918493	0.926469	0.961107
segment_osrm_distance	0.996092	0.991848	0.956106
actual_distance_to_destination	0.987627	0.993556	0.953048
osrm_distance	0.992050	0.997610	0.958341

Detecting Outliers

In [88]:
```
plt.figure(figsize = (10,8))
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()
```



In [89]:
```
outlier_treatment  = trip_records.copy()
```

In [90]:
```
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

In [91]:
```
trip_records_without_outliers = trip_records.loc[outlier_treatment_num[(np.abs(stat
trip_records_without_outliers
```

Out[91]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	de
0	trip-153671041653548748	FTL	Bhopal Kanpur	Kanpur Gurgaon	Bhopal Madhya Pradesh Kanpur Uttar Pradesh	
1	trip-153671042288605164	Carting	Tumkur Doddablpur	Doddablpur Chikblapur	Tumkur Karnataka Doddablpur Karnataka	D
3	trip-153671046011330457	Carting	Mumbai	Mumbai	Mumbai Hub Maharashtra	
4	trip-153671052974046625	FTL	Bellary Hospet Sandur	Hospet Sandur Bellary	Bellary Karnataka Hospet Karnataka Sandur Karn...	
5	trip-153671055416136166	Carting	Chennai	Chennai	Chennai Tamil Nadu	
...	...	...	...	...	...	
14812	trip-153861095625827784	Carting	Chandigarh	Zirakpur Chandigarh	Chandigarh Punjab Chandigarh Chandigarh	
14813	trip-153861104386292051	Carting	FBD	Faridabad	FBD Haryana	
14814	trip-153861106442901555	Carting	Kanpur	Kanpur	Kanpur Uttar Pradesh	
14815	trip-153861115439069069	Carting	Tirunelveli Eral Tirchchndr Thisayanvilai Peik...	Eral Tirchchndr Thisayanvilai Peikulam Tirunel...	Tirunelveli Tamil Nadu Eral Tamil Nadu Tirchch...	T
14816	trip-153861118270144424	FTL	Hospet Sandur	Sandur Bellary	Hospet Karnataka Sandur Karnataka	

14160 rows × 18 columns

Processing Data for One hot encoding :

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

In [92]:
```
trip_records_without_outliers["destination_source_locations"] = trip_records_withou
trip_records_without_outliers.drop(["source_city_state","destination_city_state"],a
```

In [93]:
```
sc_dc = trip_records_without_outliers.groupby(["destination_source_locations"])["tr
```

In [94]:
```
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"
```

In [95]:
```
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
trip_records_for_encoding.drop(["trip_uuid_y"],axis = 1,inplace=True)
# trip_records_for_encoding.sample(15)
encoded_data = pd.get_dummies(trip_records_for_encoding,
             columns=["route_type","city"] )
encoded_data
```

Out[95]:

	source_city	destination_city	source_state	destination_state	segment_osrm_time
0	Bengaluru	Bengaluru	Karnataka	Karnataka	1.383333
1	Bengaluru	Bengaluru	Karnataka	Karnataka	1.150000
2	Bengaluru	Bengaluru	Karnataka	Karnataka	1.183333
3	Bengaluru	Bengaluru	Karnataka	Karnataka	0.700000
4	Bengaluru	Bengaluru	Karnataka	Karnataka	0.783333
...	...	...	...	...	...
14155	Hyderabad Kadthal Kalwakurthy Devarakonda	Kadthal Kalwakurthy Devarakonda Haliya	Telangana	Telangana	1.966667
14156	Hyderabad Kadthal	Kadthal Devarakonda	Telangana	Telangana	1.483333
14157	Hyderabad Kadthal Haliya	Kadthal Kalwakurthy Hyderabad	Telangana	Telangana	2.916667
14158	Hyderabad Kadthal Haliya	Kadthal Devarakonda Hyderabad	Telangana	Telangana	3.383333
14159	nan	nan	nan	nan	0.800000

14160 rows × 23 columns

Column Standardization

In [96]:

```
['segment_osrm_time', 'osrm_time',
       'segment_actual_time', 'actual_time',
       'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan' ,'segment_osr
```

Out[96]:
```
['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']
```

In [97]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

In [98]:
```python
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'])
std_data.head()
```

Out[98]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
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	

In [99]:
```python
scaler = MinMaxScaler()
MinMax_data = scaler.fit_transform(encoded_data[['segment_osrm_time','osrm_time','s
 'time_taken_btwn_odstart_and_od_end','start_scan_to_end_scan','segment_osrm_distan
 '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_en
 'segment_osrm_distance','actual_distance_to_destination','osrm_distance'])
MinMax_data.head()
```

Out[99]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
0	0.069369	0.059302	0.098113	0.098719	
1	0.056757	0.054651	0.081402	0.081644	
2	0.058559	0.060465	0.102426	0.101921	
3	0.032432	0.044186	0.037736	0.037353	
4	0.036937	0.039535	0.051752	0.051761	

In [100…
```
std_data
```

Out[100]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn
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	

14160 rows × 9 columns

In [101…
```
one_hot_encoded_data = encoded_data[["route_type_Carting","route_type_FTL","city_Ca
 "city_Category 2","city_Category 3","city_Category 4",
 "city_Category 5","city_Category 6","city_Category 7"]]
```

In [102…
```
Standardized_Data = pd.concat([std_data,one_hot_encoded_data],axis = 1)
```

In [103…
```
Min_Max_Scaled_Data = pd.concat([MinMax_data,one_hot_encoded_data],axis = 1)
```

In [104…
```
Standardized_Data.sample(5)
```

Out[104]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn
12462	1.207207	1.432043	0.099435	0.096062	
3380	-0.314459	-0.431608	-0.637465	-0.636070	
1107	-0.664118	-0.665478	-0.243779	-0.248274	
7537	3.609495	3.778051	3.104235	3.128228	
8011	0.203555	0.160375	-0.038524	-0.041004	

In [105…

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

Out[105]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn
610	0.043243	0.060465	0.037736	0.037887	
1991	0.006306	0.009302	0.020485	0.020277	
5933	0.020721	0.027907	0.080863	0.080043	
11781	0.308108	0.369767	0.312129	0.311633	
6485	0.031532	0.037209	0.118059	0.117930	

Route analysis :

In [106…

```
A = data.groupby("route_schedule_uuid")["route_type"].unique().reset_index()
B = data.groupby("route_schedule_uuid")["destination_city"].unique().reset_index()
B.columns = ["route_schedule_uuid","destination_cities"]
C = data.groupby("route_schedule_uuid")["source_city"].unique().reset_index()
C.columns = ["route_schedule_uuid","source_cities"]
D = data.groupby("route_schedule_uuid")["source_state"].unique().reset_index()
D.columns = ["route_schedule_uuid","source_states"]
E = data.groupby("route_schedule_uuid")["destination_state"].unique().reset_index()
E.columns = ["route_schedule_uuid","destination_states"]
F = data.groupby("route_schedule_uuid")[["source_state",
                                          "destination_state"]].nunique().sort_value

F.columns = ["route_schedule_uuid","#source_states"
             ,"#destination_states"]
G = trip_records.groupby("route_schedule_uuid")["actual_distance_to_destination"].m
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 = data.groupby("route_schedule_uuid")[["source_city",
                                          "destination_city"]].nunique().sort_values

I.columns = ["route_schedule_uuid","#source_cities"
             ,"#destination_cities"]
```

In [107…
```
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")
```

In [108…
```
route_records.isna().sum()
```

Out[108]:
```
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
```

In [109…
```
route_records.dropna(inplace=True)
```

In [110…
```
route_records["route_type"] = route_records["route_type"].astype("str").str.strip("
route_records["source_cities"] = route_records["source_cities"].astype("str").str.s
route_records["destination_cities"] = route_records["destination_cities"].astype("s
route_records["source_states"] = route_records["source_states"].astype("str").str.s

route_records["destination_states"] = route_records["destination_states"].astype("s
```

In [111…
```
route_records
```

Out[111]:

	route_schedule_uuid	#source_cities	#destination_cities	Number_of_Trips	Averag
0	thanos::sroute:d010efca-d90d-4977-b987-eae68c5...	13	11	14	
1	thanos::sroute:4cbecb35-356b-4b68-bf3c-6225b5e...	10	10	12	
2	thanos::sroute:ae5c430f-6153-48d1-8fe5-d5f0bbc...	10	10	20	
3	thanos::sroute:f8968c72-5222-4d81-9eed-8a6d88f...	9	9	9	
4	thanos::sroute:ed5b80be-7abf-424d-b8cd-d81556a...	9	8	20	
...	...	...	...	...	
1499	thanos::sroute:9e7bb811-593f-47bc-ac49-ba03ed8...	1	1	19	
1500	thanos::sroute:46b9641b-55b5-4b15-b039-2612a50...	1	1	15	
1501	thanos::sroute:b48f633d-15cb-4744-a0b9-21df0a9...	1	1	7	
1502	thanos::sroute:265efe06-3625-4fba-afee-07b5b64...	0	1	1	
1503	thanos::sroute:cfb575b8-df26-48f5-8427-6f48f9d...	0	0	1	

1504 rows × 12 columns

In [112…

```
route_records["ROUTE"] = route_records["source_cities"] + " -- " + route_records["d
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.split
first_column = route_records.pop('SouceToDestination_city')
route_records.insert(0, 'SouceToDestination_city', first_column)
route_records
```

Out[112]:

	SouceToDestination_city	ROUTE	#source_cities	#destination_cities	Number_of_
0	Guwahati TO LakhimpurN	Guwahati LakhimpurN Dhemaji Likabali Tezpur Pa...	13	11	
1	Guwahati TO Tura	Guwahati Rangia Kokrajhar Dhubri Bilasipara Tu...	10	10	
2	Jaipur TO Tarnau	Jaipur Chomu Reengus Sikar Bikaner Didwana Suj...	10	10	
3	Mangalore TO Udupi	Mangalore Udupi Kundapura Bhatkal Honnavar Kum...	9	9	
4	Ajmer TO Raipur	Ajmer Beawar Bilara Bijainagar Kekri Nasirabad...	9	8	
...	...	...	...	...	
1499	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	
1500	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	
1501	Bengaluru TO Bengaluru	Bengaluru -- Bengaluru	1	1	
1502	nan TO Mainpuri	nan -- Mainpuri	0	1	
1503	nan TO nan	nan -- nan	0	0	

1504 rows × 13 columns

In [113…

```
Number_of_trips_between_cities = data.groupby(["source_city_state",
                                               "destination_city_state"])["trip_uui
Number_of_trips_between_cities.head(25)
```

Out[113]:

	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 higest amount of trips happening states with in the city

In [114…

```
Number_of_trips_between_cities.loc[Number_of_trips_between_cities["source_city_stat
```

Out[114]:

	source_city_state	destination_city_state	trip_uuid
1	Bhiwandi Maharashtra	Mumbai Maharashtra	512
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
9	MAA Tamil Nadu	Chennai Tamil Nadu	204
10	Chennai Tamil Nadu	MAA Tamil Nadu	141
11	Bengaluru Karnataka	HBR Karnataka	133
13	Pune Maharashtra	PNQ Maharashtra	122
16	Pune Maharashtra	Bhiwandi Maharashtra	107
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
26	Del Delhi	Gurgaon Haryana	76
27	Bhiwandi Maharashtra	Pune Maharashtra	72
28	Ludhiana Punjab	Chandigarh Punjab	71
30	Chandigarh Punjab	Gurgaon Haryana	66
31	Gurgaon Haryana	Bengaluru Karnataka	66
32	LowerParel Maharashtra	Mumbai Maharashtra	65
34	Mumbai Hub Maharashtra	Bhiwandi Maharashtra	63
35	PNQ Maharashtra	Pune Maharashtra	62

source and destination cities having higest number of trips in between are :

- delhi to gurgao
- Gurgaon,Haryana TO Bengaluru,Karnataka
- Bhiwandi/Mumbai,Maharashtra TO Pune Maharashtra
- Sonipat TO Gurgaon,Haryana

- lots of deliveries are happening to airpots
- Eg. Chennai to MAA chennai international Airport , Pune to Pune Airport (PNQ),Kolkata to CCU West Bengal Kolkata International Airport , Bengluru to BLR-Bengaluru Internation Airport etc.

In [115…

```
route_records[["ROUTE","Number_of_Trips",
               "Average_Actual_distance_to_destination",
               "#source_cities",
               "#destination_cities"]].sort_values(by="Number_of_Trips",ascending=F
```

Out[115]:

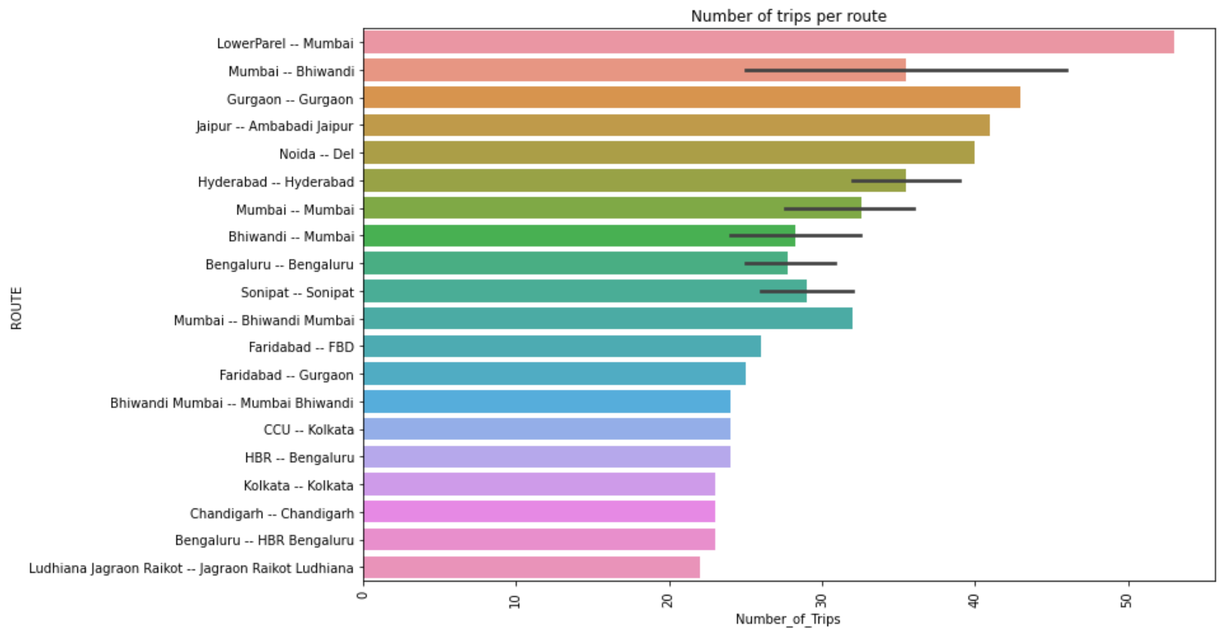
	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities
1465	LowerParel -- Mumbai	53	16.428868	1
1426	Mumbai -- Bhiwandi	46	20.199445	1
808	Gurgaon -- Gurgaon	43	29.740842	1
679	Jaipur -- Ambabadi Jaipur	41	15.348495	1
1257	Noida -- Del	40	10.882902	1
1368	Hyderabad -- Hyderabad	39	35.695641	1
1273	Mumbai -- Mumbai	37	13.882863	1
1359	Mumbai -- Mumbai	36	17.526251	1
1303	Bhiwandi -- Mumbai	35	21.241534	1
700	Mumbai -- Mumbai	34	15.906614	1
751	Mumbai -- Mumbai	33	15.668726	1
1060	Bengaluru -- Bengaluru	33	28.067004	1
793	Sonipat -- Sonipat	32	11.691243	1
972	Hyderabad -- Hyderabad	32	21.835579	1
1184	Mumbai -- Bhiwandi Mumbai	32	21.601109	1
874	Bengaluru -- Bengaluru	30	28.055789	1
1177	Bhiwandi -- Mumbai	30	21.396002	1

	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities
1354	Bengaluru -- Bengaluru	27	27.967087	1
921	Faridabad -- FBD	26	9.677121	1
1480	Sonipat -- Sonipat	26	12.182486	1
1041	Mumbai -- Bhiwandi	25	19.942191	1
877	Faridabad -- Gurgaon	25	47.091622	1
833	Bhiwandi -- Mumbai	25	21.531705	1
1249	Bengaluru -- Bengaluru	25	28.019668	1
869	Bengaluru -- Bengaluru	24	41.396497	1

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

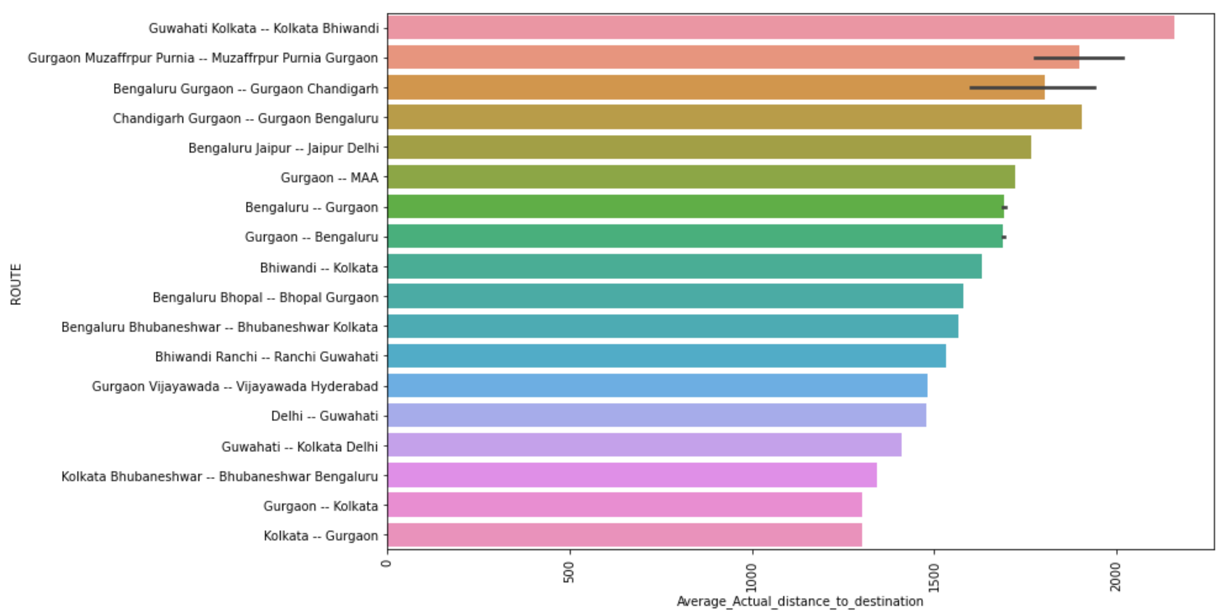
In [116…

```
plt.figure(figsize=(12,8))

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()
```



In [117…

```
plt.figure(figsize=(12,8))

X = route_records[["ROUTE", "Average_Actual_distance_to_destination",
                ]].sort_values(by="Average_Actual_distance_to_destination",ascending
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 with in 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: -

In [118…

```
Busiest_and_Longest_Routes  = route_records[(route_records["Average_Actual_distance
               & (route_records["Number_of_Trips"] > route_records["Number_of_Trips"

Busiest_and_Longest_Routes_top25 = Busiest_and_Longest_Routes[["source_cities",
                                                               "destination_cities"
                                                               "Number_of_Trips",
                                                               "Average_Actual_dist
Busiest_and_Longest_Routes_top25
```

Out[118]:

	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destina
629	Chandigarh Gurgaon	Gurgaon Bengaluru	22	1905.76
995	Gurgaon	Bengaluru	21	1689.87
991	Gurgaon	Bengaluru	21	1689.79
512	Bengaluru Bhubaneshwar	Bhubaneshwar Kolkata	18	1567.57
745	Guwahati	Kolkata Delhi	18	1411.20
624	Kolkata Bhubaneshwar	Bhubaneshwar Bengaluru	16	1342.14
752	Gurgaon	Kolkata	16	1300.57
588	Delhi Gurgaon	Gurgaon Kolkata	18	1263.11
826	Gurgaon	Hyderabad	16	1236.57
541	Chandigarh Gurgaon	Gurgaon Bhiwandi	20	1170.81
442	Delhi Gurgaon	Gurgaon Pune	22	1151.51
445	Bhiwandi Sonipat	Sonipat Chandigarh	18	1129.60
739	Pune	Gurgaon	18	1120.72
1377	Bhiwandi	Delhi	19	1114.21
1049	Delhi	Bhiwandi	18	1114.18
313	Bengaluru Kolhapur Surat	Kolhapur Surat Ahmedabad	16	1110.01
1219	Gurgaon	Bhiwandi	16	1078.07
197	Sasaram Kanpur Kolkata Dhanbad	Kanpur Gurgaon Dhanbad Sasaram	16	1028.02
1136	Gurgaon	Ranchi	16	1010.95
1286	Surat	Delhi	18	931.98
439	Kolkata Ranchi	Ranchi Gurgaon	16	881.62
1108	Gurgaon	Sasaram	18	804.21
1454	Gurgaon	Ahmedabad	17	735.55
223	Bhopal Kanpur Auraiya Etawah	Kanpur Auraiya Etawah Gurgaon	21	731.63

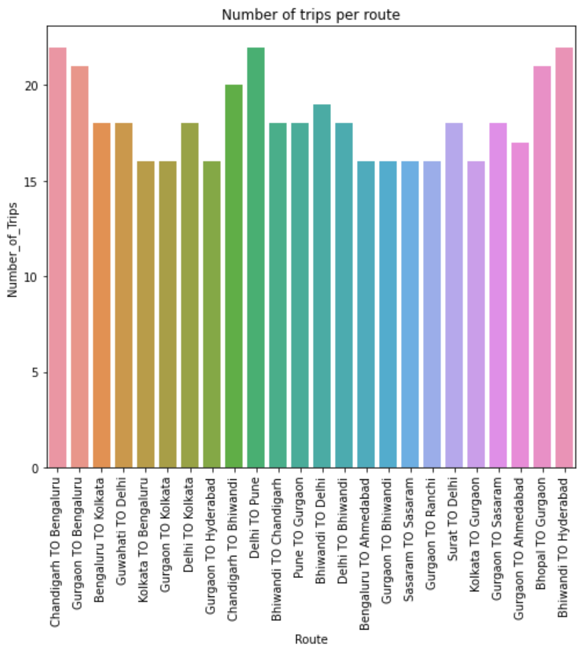
	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destina
863	Bhiwandi	Hyderabad	22	607.51

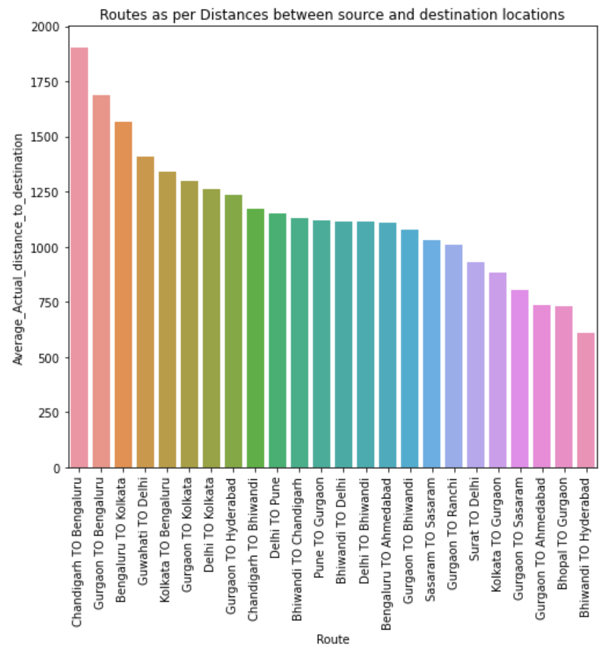
Above Table shows the souce to destination city routes having largest numbers of trip happening having large distnaces : which are :

- Chandigarh TO Bengaluru
- Gurgaon TO Bengaluru
- Bengaluru TO Kolkata
- Guwahati TO Delhi
- Delhi TO Kolkata
- Chandigarh TO Gurgaon
- Gurgaon TO Hydrabad
- Benglore TO Ahmedabad
- Surat TO Delhi
- Gurgaon TO Ahmedabad**

In [119…

```
Busiest_and_Longest_Routes_top25["Route"] = Busiest_and_Longest_Routes_top25["sourc
Busiest_and_Longest_Routes_top25.drop(["source_cities","destination_cities"],axis =
plt.figure(figsize=(18,7))

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_destinat
plt.xticks(rotation = 90)
plt.show()
```





Routes passing through maxinum number of cities

In [120…

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

Out[120]:

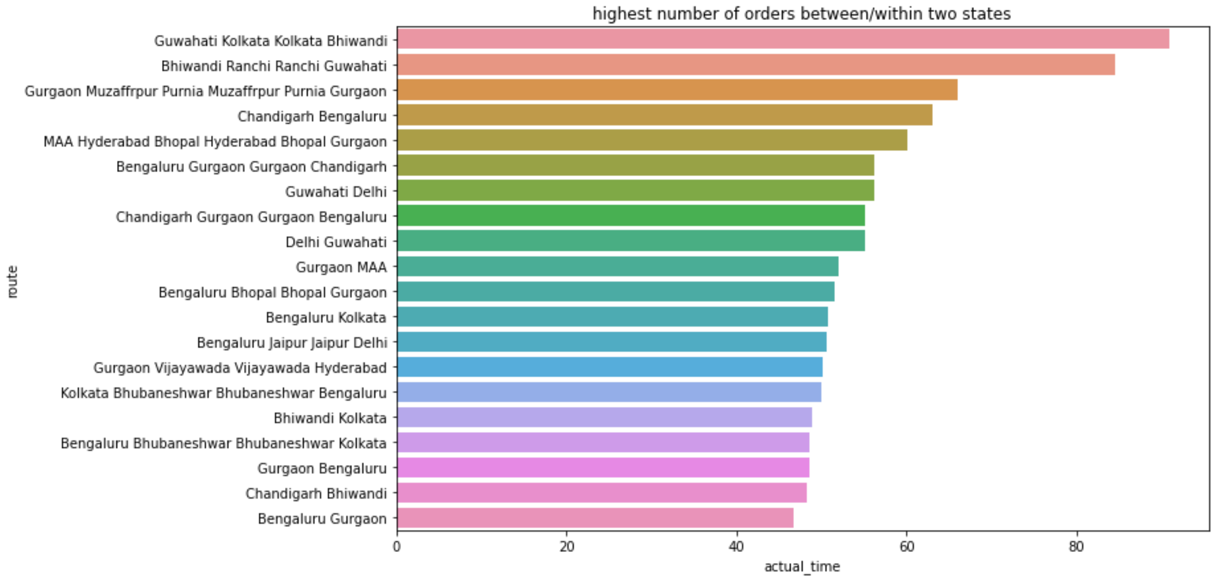
	SouceToDestination_city	Number_of_Trips	Average_Actual_distance_to_destination	#sou
0	Guwahati TO LakhimpurN	14	281.596486	
2	Jaipur TO Tarnau	20	351.611796	
1	Guwahati TO Tura	12	332.602225	
3	Mangalore TO Udupi	9	195.257193	
4	Ajmer TO Raipur	20	178.737233	
5	Mainpuri TO Tilhar	12	207.247057	
8	Hassan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

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

In [121…

```
Longest_route_as_per_actual_trip_time = trip_records.groupby(["source_city",
                            "destination_city"])["actual_time"].mean().sort_values(ascend
```

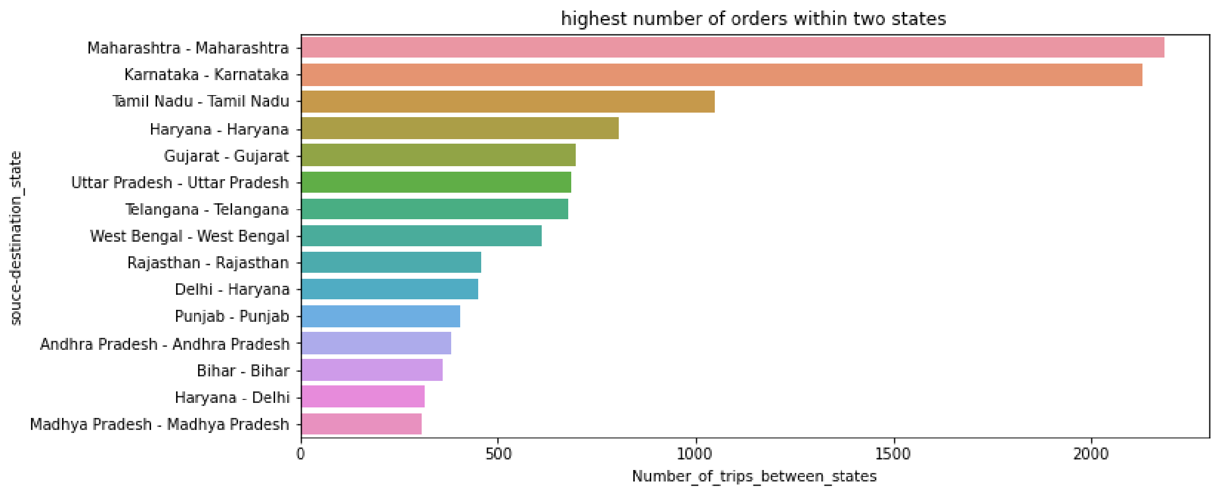
```
Longest_route_as_per_actual_trip_time["route"] = Longest_route_as_per_actual_trip_t
Longest_route_as_per_actual_trip_time.drop(["source_city",
                                            "destination_city"],axis = 1,inplace=Tr
Longest_route_as_per_actual_trip_time
plt.figure(figsize=(11,7))
sns.barplot(y = 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()
```



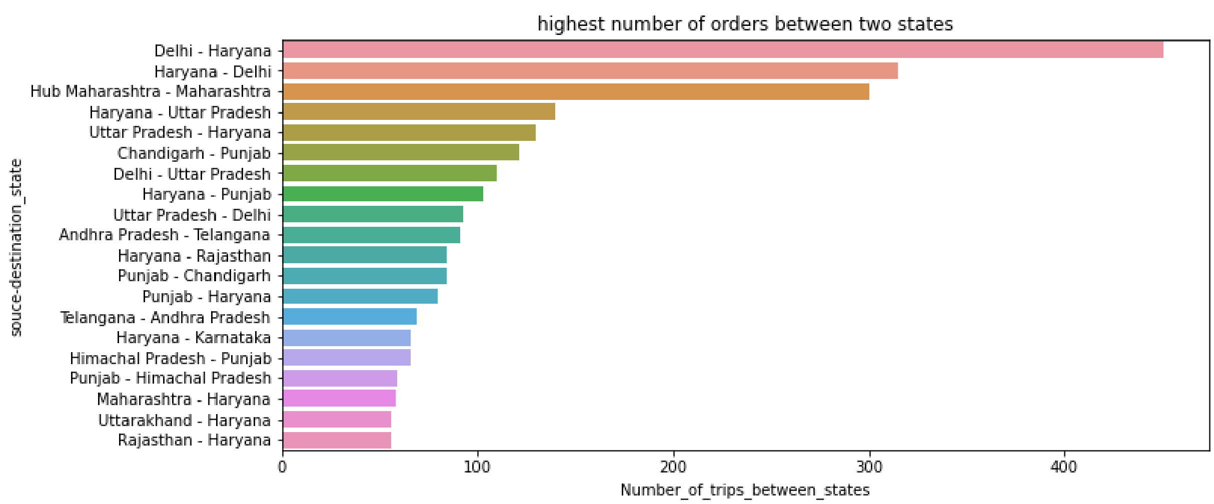
highest number of Trips happening between/within two states: -

In [122…

```
highest_order_between_states = data.groupby(["source_state",
                                             "destination_state"])["trip_uuid"].nun
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()
```




```
In [123…  HOBS = data.groupby(["source_state","destination_state"])["trip_uuid"].nunique().so
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()
```



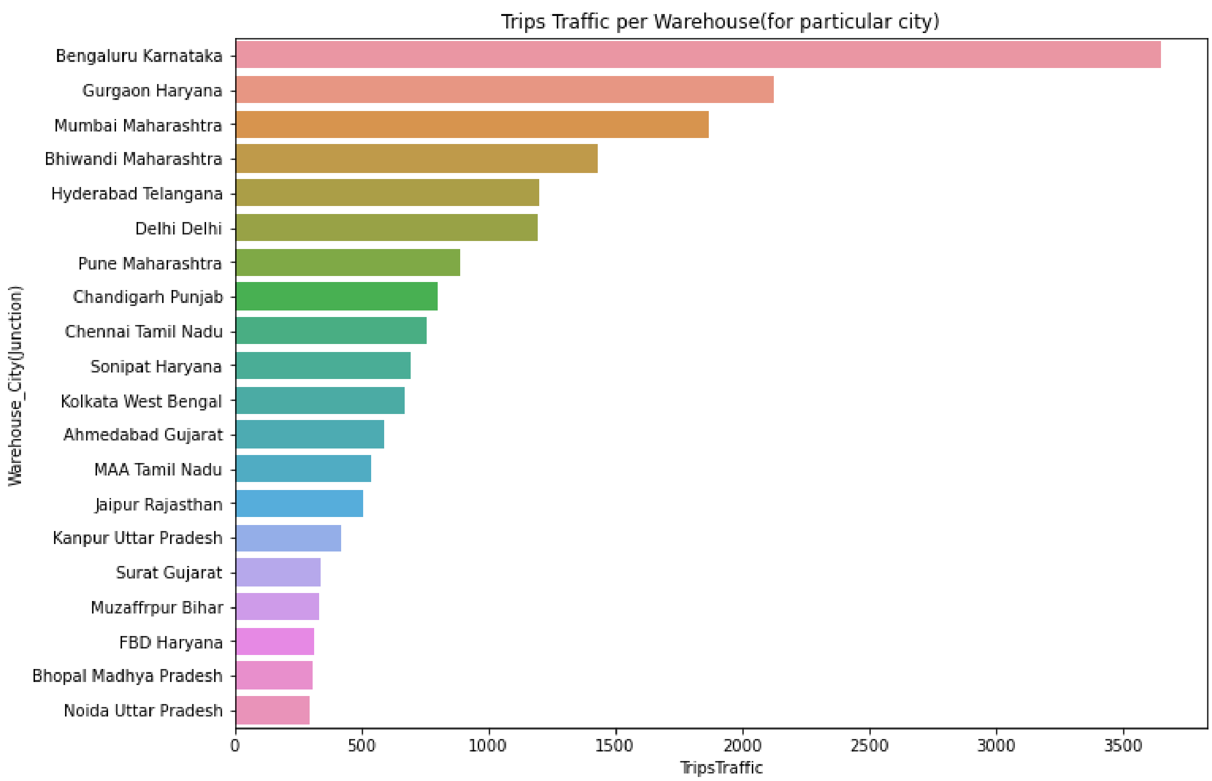
- Delhi to Haryana is the busiest route, having more than 400 trips in between. Some of such busy routes are Haryana to Uttar Pradesh , Chandigarh to Punjab , Delhi to Uttar Pradesh
- Within the state , Maharashtra , Karnataka, Tamil Nadu are some states having above 1000 trips.

Top 20 warehouses with heavy traffic: -

In [124…
```
destination_traffic = data.groupby(["destination_city_state"])["trip_uuid"].nunique
source_traffic = data.groupby(["source_city_state"])["trip_uuid"].nunique().reset_i
transactions = source_traffic.merge(destination_traffic,
                                left_on="source_city_state"
                                ,right_on="destination_city_state")
transactions.columns = ["source_city_state","#Trips_s","destination_city_state","#T
transactions["TripsTraffic"] = transactions["#Trips_s"]+transactions["#Trips_d"]
transactions.drop(["#Trips_s","#Trips_d","destination_city_state"],axis = 1,inplace
transactions.columns = ["Warehouse_City(Junction)","TripsTraffic"]
```

In [125…
```
T = transactions.sort_values(by=["TripsTraffic"],ascending=False).head(20)
```

In [126…
```
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()
```



In [127…
```
trip_records.groupby(["source_state","destination_state"])["trip_uuid"].count().sor
```

Out[127]:

	source_state	destination_state	trip_uuid
0	Maharashtra	Maharashtra	2085
1	Karnataka	Karnataka	2002
2	Tamil Nadu	Tamil Nadu	996
3	Haryana	Haryana	771
4	Telangana	Telangana	627
5	Gujarat	Gujarat	624
6	West Bengal	West Bengal	610
7	Uttar Pradesh	Uttar Pradesh	529
8	Rajasthan	Rajasthan	400
9	Delhi	Haryana	385
10	Andhra Pradesh	Andhra Pradesh	344
11	Punjab	Punjab	342
12	Bihar	Bihar	330
13	Haryana	Delhi	307
14	Hub Maharashtra	Maharashtra	300

Insights

- During September and October of 2018, there were 14,817 different trips between sources and destinations
- There are 1,504 delivery routes on which trips are happening
- We have 1,508 unique source centers and 1,481 unique destination centers
- Out of the 14,817 total different trips, 8,908 (60%) of the trip routes are Carting, which consists of small vehicles, and 5,909 (40%) of the total trip routes are FTL (Full Truck Load), which reach the destination sooner as there are no other pickups or drop-offs along the way

EDA Results

- Mumbai, Maharashtra; Delhi; Gurgaon, Haryana; Bengaluru, Karnataka; Hyderabad, Telangana; Chennai, Tamil Nadu; Ahmedabad, Gujarat; Pune, Maharashtra; Chandigarh, Chandigarh; and Kolkata, West Bengal are cities with the highest number of trips happening within the city.

- When looking at the cities with unequal source and destination states, the cities with the highest number of trips in between are Delhi to Gurgaon, Gurgaon to Bengaluru, Bhiwandi/Mumbai to Pune, Maharashtra, and Sonipat to Gurgaon, Haryana.
- Many deliveries are happening to airports such as Chennai International Airport, Pune Airport (PNQ), Kolkata International Airport, and Bengaluru International Airport.
- The highest number of trips is happening within particular cities, and in terms of average distance between destinations, the longest routes are Guwahati to Mumbai, Bengaluru to Chandigarh, Bengaluru to Delhi, and Bengaluru to Gurgaon. These insights are derived from bar charts and calculated tables in the analysis.

Recommendations

- While the North, South, and West Zones corridors have significant order traffic, our presence in the Central, Eastern, and North-Eastern zones is smaller. However, it is worth investigating and increasing our presence in these regions, although it would be difficult to conclude based on just two months of data.
- Based on the traffic of orders, Maharashtra followed by Karnataka have the heaviest traffic from a state perspective. Therefore, planning for resources on the ground in these two states is a priority, especially during festive seasons.
- To increase revenue and reputation in terms of connectivity across borders, increasing connectivity in tier 2 and tier 3 cities and establishing professional partnerships with various e-commerce giants is recommended.
- Based on the analysis, it is recommended that small vehicles, such as carts, be used for city deliveries to reduce delivery time, while heavy trucks should be used for long distance trips or heavy loads. This optimization can increase revenue as required.
- The scanning time can be optimized at both the start and end of the process to equate delivery time with the OSRM estimated delivery time.
- For trip planning, the information fed to the routing engine should be revisited and checked for discrepancies with transporters to ensure the routing engine is configured for optimal results.

In [ ]: