About Delhivery :

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.

Business Problem Statement :

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

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

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

Column Profiling:

data : tells whether the data is testing or training data

trip_creation_time : Timestamp of trip creation

route_schedule_uuid : Unique Id for a particular route schedule

route_type : Transportation type

FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way

Carting: Handling system consisting of small vehicles (carts)

trip_uuid : Unique ID given to a particular trip (A trip may include different source and destination centers)

source_center : Source ID of trip origin

source_name : Source Name of trip origin

destination_cente : Destination ID

destination_name : Destination Name

od_start_time : Trip start time

od_end_time : Trip end time

start_scan_to_end_scan : Time taken to deliver from source to destination

is_cutoff : Unknown field

cutoff_factor : Unknown field

cutoff_timestamp : Unknown field

actual_distance_to_destination : Distance in Kms between source and destination warehouse

actual_time : Actual time taken to complete the delivery (Cumulative)

osrm_time : An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)

osrm_distance : An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)

factor : Unknown field

segment_actual_time : This is a segment time. Time taken by the subset of the package delivery

segment_osrm_time : This is the OSRM segment time. Time taken by the subset of the package delivery

segment_osrm_distance : This is the OSRM distance. Distance covered by subset of the package delivery

segment_factor : Unknown field

In-depth analysis and feature engineering to be done :

time taken between od_start_time and od_end_time

hypothesis testing/ Visual analysis : population mean of start_scan_to_end_scan & time taken between od_start_time and od_end_time

hypothesis testing/ visual analysis :

- actual_time aggregated value and OSRM time aggregated value

hypothesis testing/ visual analysis :

- actual_time aggregated value and segment actual time

hypothesis testing/ visual analysis :

- osrm distance aggregated value and segment osrm distance

hypothesis testing/ visual analysis :

- osrm time aggregated value and segment osrm time aggregated value

> - outliers in the numerical variables

> - outliers using the IQR method.

> - one-hot encoding of categorical variables (like route_type)

> - Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [8]:

```
pd.set_option('display.max_columns', None)
```

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import figure
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import t
import plotly.express as px
from scipy import stats
```

In [2]:

```
df = pd.read_csv("delhivery_data.txt")

```

In [3]:

```
1 df.head()
```

Out[3]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_c
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812

5 rows × 24 columns

In [ ]:

```
1 # checking the shape (rows and columns)
```

In [4]:

```
1 df.shape
```

Out[4]:

```
(144867, 24)
```

In [9]:

```
1 # 144,867 total Records
2 # 24 columns
```

In [ ]:

```
1
```

In [10]:

```
1 df.info()
```

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

In [ ]:

```
1
```

In [11]:

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

Out[11]:

```
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 [ ]:

```

```

In [12]:

```
# features : source_name and destination_name having few missing values
```

In [ ]:

```

```

Changing data type for data and time related features :

In [ ]:

```

```

In [14]:

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

In [ ]:

```

```

Extracting Trip Creation Informations from Trip Creation time :

In [ ]:

```

```

In [17]:

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

Out[17]:

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

In [ ]:

```

```

In [18]:

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

Out[18]:

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

In [ ]:

```

```

Observation:

delivery trip data is given from Septemebr and October 2018.

In [ ]:

```

```

In [21]:

```
df["trip_creation_day"] = (df["trip_creation_time"].dt.day_name())
df["trip_creation_month"] = (df["trip_creation_time"].dt.month_name())
df["trip_creation_year"] = (df["trip_creation_time"].dt.year)
```

In [ ]:

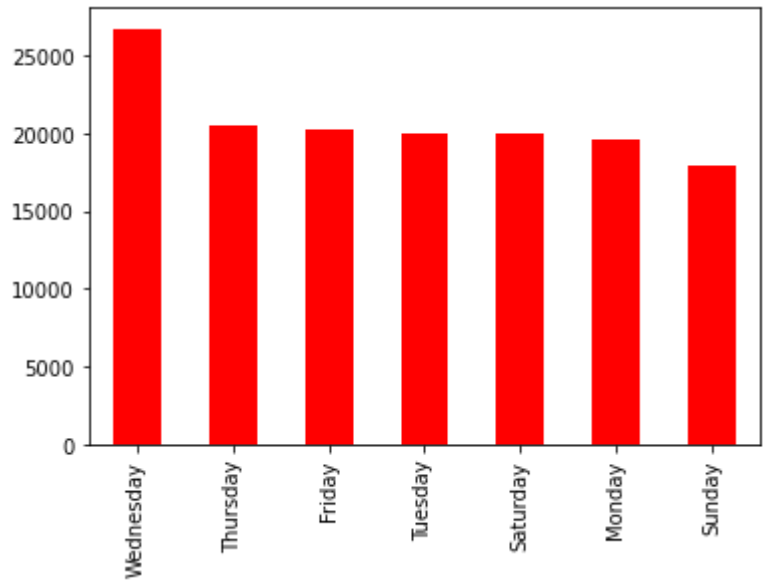
```

```

In [23]:

```
df["trip_creation_day"].value_counts().plot(kind = "bar", color = "red")
```

Out[23]:

<AxesSubplot:>



In [ ]:

```

```

In [24]:

```
# wednesday seems to have relatively higher records of data compare to other days .
df["trip_creation_day"].value_counts(normalize=True)*100
```

Out[24]:

```
Wednesday    18.452788
Thursday     14.137795
Friday       13.972816
Tuesday      13.778845
Saturday     13.761588
Monday       13.560714
Sunday       12.335453
Name: trip_creation_day, dtype: float64
```

In [ ]:

```
1
```

Understanding the structure :

In [ ]:

```
1
```

In [26]:

```
1 df.nunique()
```

Out[26]:

```
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
trip_creation_day                   7
trip_creation_month                 2
trip_creation_year                  1
dtype: int64
```

Observation:

we have 14817 different trips happended between source to destinations.

total 1504 delivery routes we have.

1508 unique source centers

1481 unique destination centres

In [ ]:

```
1
```

There are two different kind of routes are there :

In [ ]:

```
1
```

In [31]:

```
1 df.groupby("trip_uuid")["route_type"].unique().reset_index()["route_type"].apply(lambda
2
```

Out[31]:

```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

In [ ]:

```
1
```

In [32]:

```
1 df.groupby("trip_uuid")["route_type"].unique().reset_index()["route_type"].apply(lambda
2
```

Out[32]:

```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

In [ ]:

```
1
```

In [33]:

```
1 routeType_plot= (df.groupby("trip_uuid")["route_type"].unique().reset_index()["route_ty
2 routeType_plot
```

Out[33]:

```
Carting    60.120132
FTL        39.879868
Name: route_type, dtype: float64
```

In [ ]:

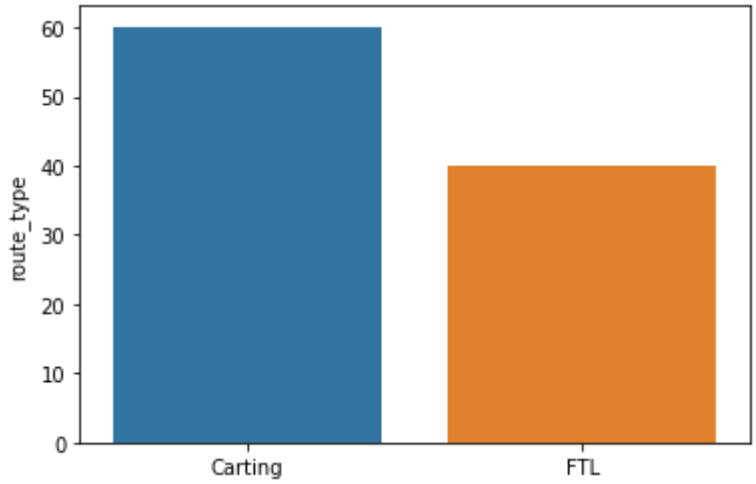
```
1
```

In [34]:

```
1 sns.barplot(x= routeType_plot.index,
2             y = routeType_plot)
```

Out[34]:

```
<AxesSubplot:ylabel='route_type'>
```



In [ ]:

```
1
```

Observation :

From 14817 total different trips , we have

8908 (60%) of the trip-routes are Carting , which consists of small vehicles and

5909 (40%) of total trip-routes are FTL : which are Full Truck Load get to the destination sooner. as no otther pickups or drop offs along the way .

In [ ]:

```
1
```

Undestanding Features and Feature Engineering :

Analyzing records for one particular trip id :

In [ ]:

```
1
```

In [39]:

```
1 df[df["trip_uuid"]=="trip-153741093647649320"]
```

Out[39]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_c
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38862

In [ ]:

```
1
```

Observation:

from above one particular trip record ,

trip is segmented between different drop locations .

we can observe

trip is taking stops between mentioned source and destination centers(warehouses).

od-end-tiem and od-start-time are the time when the that particular trip was ended and started .

start-scan-to-end-scan is the time duration of trips are being scanned when start and end.

start-scan-to-end-scan time is given cummulative. which is not given per trip segments.

trip cut off False ,shows the record of trip when trip changes from one warehouse to another. between source to destination.

Actual-time given is the time to complete the entire delivery from source to destination (given cumulatively)

osrm -time is an open rourse routing engine time calculator which computes the shortest path between points in a given map and gives the time and osrm distance gives the shortest distance (given cumulatively)

Actual-distnace-to-destination is the actual distance between warehouses , given cummulative during the trip .

every time cutoff is False , distance count starts from begining.

Segmment actual time, is the actual time taken between two stops in between trips. given per each segment (taken between subset of package delivery)

segment osrm time is the osrm segment time , taken between subset of package delivery

In [ ]:

```
1
```

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

In [ ]:

```
1
```

In [ ]:

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

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

In [ ]:

```

```

In [46]:

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

In [ ]:

```

```

In [48]:

```
1
2 df
```

Out[48]:

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

144867 rows × 33 columns

In [ ]:

```
1
```

Time_taken_btwn_odstart_and_od_end VS start_scan_to_end_scan :

In [ ]:

```
1
```

In [51]:

```
1 df["time_taken_btwn_odstart_and_od_end"] = ((df["od_end_time"]-df["od_start_time"])/pd.
2
```

In [ ]:

```
1
```

In [52]:

```
1 # Converting given time duration features into hours .
2 ### start_scan_to_end_scan
3 ### actual_time
4 ### osrm_time
5 ### segment_actual_time
6 ### segment_osrm_time
```

In [ ]:

```
1
```

In [53]:

```
1 df["start_scan_to_end_scan"] = df["start_scan_to_end_scan"]/60
2 df["actual_time"] = df["actual_time"]/60
3 df["osrm_time"] = df["osrm_time"]/60
4 df["segment_actual_time"] = df["segment_actual_time"]/60
5 df["segment_osrm_time"] = df["segment_osrm_time"]/60
```

In [ ]:

```
1
```

In [54]:

```
1 df
```

Out[54]:

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

144867 rows × 34 columns

In [ ]:

```
1
```

In [55]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 34 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   data                               144867 non-null  object
 1   trip_creation_time                 144867 non-null  datetime64[ns]
 2   route_schedule_uuid                144867 non-null  object
 3   route_type                         144867 non-null  object
 4   trip_uuid                          144867 non-null  object
 5   source_center                      144867 non-null  object
 6   source_name                        144574 non-null  object
 7   destination_center                 144867 non-null  object
 8   destination_name                   144606 non-null  object
 9   od_start_time                      144867 non-null  datetime64[ns]
 10  od_end_time                        144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan             144867 non-null  float64
 12  is_cutoff                          144867 non-null  bool
 13  cutoff_factor                      144867 non-null  int64
 14  cutoff_timestamp                   144867 non-null  object
 15  actual_distance_to_destination     144867 non-null  float64
 16  actual_time                        144867 non-null  float64
 17  osrm_time                          144867 non-null  float64
 18  osrm_distance                      144867 non-null  float64
 19  factor                             144867 non-null  float64
 20  segment_actual_time                144867 non-null  float64
 21  segment_osrm_time                  144867 non-null  float64
 22  segment_osrm_distance              144867 non-null  float64
 23  segment_factor                     144867 non-null  float64
 24  trip_creation_day                  144867 non-null  object
 25  trip_creation_month                144867 non-null  object
 26  trip_creation_year                 144867 non-null  int64
 27  source_city                        144574 non-null  object
 28  source_state                       144574 non-null  object
 29  destination_city                   144606 non-null  object
 30  destination_state                  144606 non-null  object
 31  source_pincode                     144867 non-null  object
 32  destination_pincode                144867 non-null  object
 33  time_taken_btwn_odstart_and_od_end  144867 non-null  float64
dtypes: bool(1), datetime64[ns](3), float64(11), int64(2), object(17)
memory usage: 36.6+ MB
```

In [ ]:

```
1
```

In [56]:

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

Out[56]:

```
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
trip_creation_day                        0
trip_creation_month                      0
trip_creation_year                       0
source_city                            293
source_state                           293
destination_city                       261
destination_state                      261
source_pincode                           0
destination_pincode                      0
time_taken_btwn_odstart_and_od_end       0
dtype: int64
```

In [ ]:

```

```

In [57]:

```
df.shape
```

Out[57]:

```
(144867, 34)
```

In [ ]:

```

```

Data cleaning :

In [ ]:

```
1
```

In [59]:

```
df["source_state"] = df["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 [ ]:

```
1
```

In [60]:

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

In [61]:

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

In [ ]:

```

```

In [62]:

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

In [ ]:

```

```

In [63]:

```
df["source_city_state"] = df["source_city"] + " " + df["source_state"]
df["destination_city_state"] = df["destination_city"] + " " + df["destination_state"]
df["source_city_state"].nunique()
```

Out[63]:

```
1249
```

In [ ]:

```

```

In [64]:

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

Out[64]:

```
1242
```

In [ ]:

```

```

In [65]:

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

Out[65]:

```
33
```

In [ ]:

```
1
```

In [66]:

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

Out[66]:

```
32
```

In [ ]:

```
1
```

In [67]:

```
1 ## Delhivery delivered in approdimately 1250 cities and almost all the states all over
2 data = df.copy()
```

In [ ]:

```
1
```

In [68]:

```
1 data.columns
```

Out[68]:

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor',
       'trip_creation_day', 'trip_creation_month', 'trip_creation_year',
       'source_city', 'source_state', 'destination_city', 'destination_stat
e',
       'source_pincode', 'destination_pincode',
       'time_taken_btwn_odstart_and_od_end', 'source_city_state',
       'destination_city_state'],
      dtype='object')
```

In [ ]:

```
1
```

In [71]:

```
1 # data[["source_city","source_state","destination_city","destination_state","source_cit
2 # above data we impute after aggregating as per tripIDs.
```

In [70]:

```
1 data.columns
```

Out[70]:

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor',
       'trip_creation_day', 'trip_creation_month', 'trip_creation_year',
       'source_city', 'source_state', 'destination_city', 'destination_stat
e',
       'source_pincode', 'destination_pincode',
       'time_taken_btwn_odstart_and_od_end', 'source_city_state',
       'destination_city_state'],
      dtype='object')
```

In [ ]:

```
1
```

In [72]:

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

In [73]:

```
1 data
```

Out[73]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sta
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	
...	...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	

144867 rows × 29 columns

In [ ]:

```
1
```

Aggregating Data :

In [ ]:

```
1
```

In [76]:

```
actual_time = data.groupby(["trip_uuid",
             "start_scan_to_end_scan"])["actual_time"].max().reset_index().groupby("tr
segment_osrm_time = data[["trip_uuid","segment_osrm_time"]].groupby("trip_uuid")["segme
segment_actual_time = data.groupby("trip_uuid")["segment_actual_time"].sum().reset_inde
osrm_time = data.groupby(["trip_uuid",
             "start_scan_to_end_scan"])["osrm_time"].max().reset_index().groupby("trip
time_taken_btwn_odstart_and_od_end = data.groupby("trip_uuid")["time_taken_btwn_odstart

time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"] = time_taken_b
start_scan_to_end_scan = ((data.groupby("trip_uuid")["start_scan_to_end_scan"].unique()
start_scan_to_end_scan["start_scan_to_end_scan"] = start_scan_to_end_scan["start_scan_t

osrm_distance = data.groupby(["trip_uuid",
             "start_scan_to_end_scan"])["osrm_distance"].max().reset_index().groupby("
actual_distance_to_destination = data.groupby(["trip_uuid",
             "start_scan_to_end_scan"])["actual_distance_to_destination"].max().reset_
segment_osrm_distance = data[["trip_uuid",
                              "segment_osrm_distance"]].groupby("trip_uuid")["segment_o

```

In [ ]:

```
1
```

Hypothesis Tests for time durations and distance related features :

In [ ]:

```
1
```

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

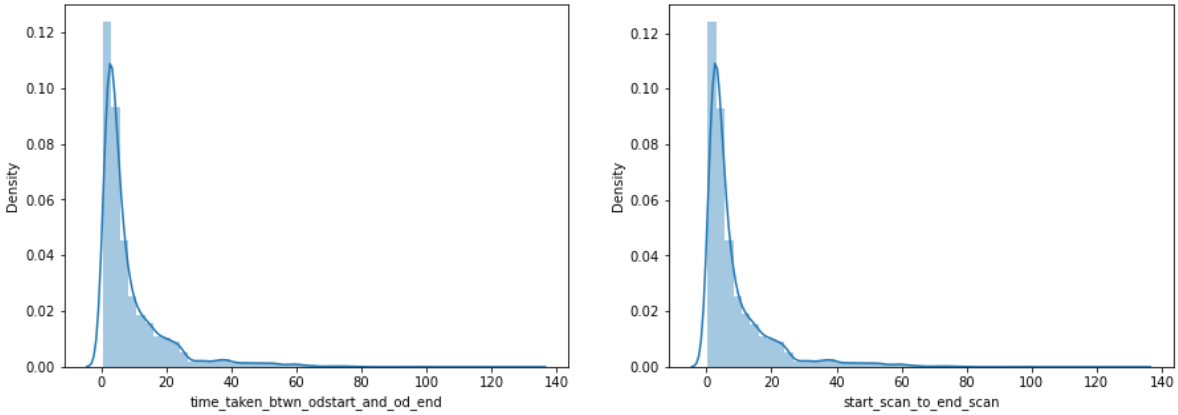
Ha: Mean of time taken betweenn trip end ans start time != Mean of start and end scan time

In [81]:

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

Out[81]:

```
<AxesSubplot:xlabel='start_scan_to_end_scan', ylabel='Density'>
```



KS-test : checking the distributions how closly equal thy are :

In [ ]:

```

```

In [83]:

```
stats.ks_2samp(time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"]
               ,start_scan_to_end_scan["start_scan_to_end_scan"])
```

Out[83]:

```
KstestResult(statistic=0.004184382803536474, pvalue=0.9994337058695081)
```

In [ ]:

```

```

In [84]:

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

```

```
Ttest_indResult(statistic=-0.1538006358845942, pvalue=0.8777721065948222)
Ttest_indResult(statistic=-0.42118413190293863, pvalue=0.6736357222499234)
Ttest_indResult(statistic=-0.7119718955744196, pvalue=0.4765098703519527)
Ttest_indResult(statistic=2.347953137822062, pvalue=0.018909207015549573)
Ttest_indResult(statistic=1.0690110805330388, pvalue=0.28510769579533424)
```

In [ ]:

```

```

Observations:

from Kolmogorov–Smirnov test , p-value is 0.9943 , from which we can conclude tht both the distributions

(time_taken_btwn_odstart_and_od_end and start_scan_to_end_scan) are closly similar.

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 [ ]:

```

```

In [87]:

```
# also checking mean and standard deviation for timetaken and scan times :
```

In [ ]:

```

```

In [89]:

```
time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"].mean(),time_ta

```

Out[89]:

```
(8.861857235305067, 10.981665759990623)
```

In [ ]:

```
1
```

In [90]:

```
1 start_scan_to_end_scan["start_scan_to_end_scan"].mean(),start_scan_to_end_scan["start_s
2
```

Out[90]:

```
(8.835777597804325, 10.97628639143973)
```

In [91]:

```
1 # variance and means both are closly similar for scan time and trip start and end time
2
```

In [ ]:

```
1
```

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 [ ]:

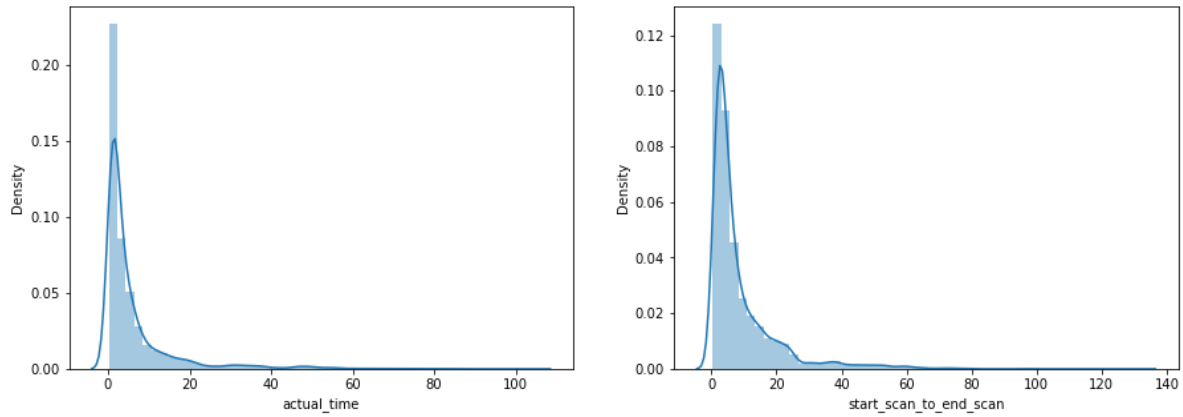
```
1
```

In [93]:

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

Out[93]:

```
<AxesSubplot:xlabel='start_scan_to_end_scan', ylabel='Density'>
```



In [ ]:

```

```

In [94]:

```
stats.ks_2samp(actual_time["actual_time"],start_scan_to_end_scan["start_scan_to_end_sca

```

Out[94]:

```
KstestResult(statistic=0.27387460349598436, pvalue=0.0)
```

In [ ]:

```

```

In [95]:

```
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)),alterr

```

```
Ttest_indResult(statistic=-10.647554735098792, pvalue=1.5333885506954987e-2
6)
Ttest_indResult(statistic=-11.013029413823777, pvalue=3.058256456874379e-28)
Ttest_indResult(statistic=-9.142812951865007, pvalue=4.0881869440121917e-20)
Ttest_indResult(statistic=-10.255316278067419, pvalue=8.91030202914973e-25)
Ttest_indResult(statistic=-9.676112659948, pvalue=2.755522973237462e-22)
Ttest_indResult(statistic=-11.787470701900334, pvalue=5.059691204683354e-32)
Ttest_indResult(statistic=-9.834930239973346, pvalue=5.902314801886429e-23)
```

In [ ]:

```
1
```

Observations:

from KS test for actual-time and start_scan_to_end_scan distributions are not same.

from ttest of population average actual_time is less than population average start_scan_to_end_scan.

In [ ]:

```
1
```

In [98]:

```
1 actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```

Out[98]:

(5.945176711435117, 9.35554782297388)

In [99]:

```
1 start_scan_to_end_scan["start_scan_to_end_scan"].mean(),start_scan_to_end_scan["start_s
2
```

Out[99]:

(8.835777597804325, 10.97628639143973)

In [ ]:

```
1
```

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 [ ]:

```
1
```

In [101]:

```
stats.ks_2samp(actual_time["actual_time"],time_taken_btwn_odstart_and_od_end["time_take
```

Out[101]:

```
KstestResult(statistic=0.2765067152594992, pvalue=0.0)
```

In [ ]:

```

```

In [102]:

```
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_od_er


```

```
Ttest_indResult(statistic=-7.530136041082737, pvalue=7.628023976401307e-14)
Ttest_indResult(statistic=-6.6376144885586355, pvalue=4.094164579047634e-11)
Ttest_indResult(statistic=-6.63334280033784, pvalue=4.21184058421215e-11)
Ttest_indResult(statistic=-6.79672388024381, pvalue=1.4080245660574904e-11)
Ttest_indResult(statistic=-6.555742305839016, pvalue=7.027053806408312e-11)
```

In [ ]:

```

```

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.

In [ ]:

```

```

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 [ ]:

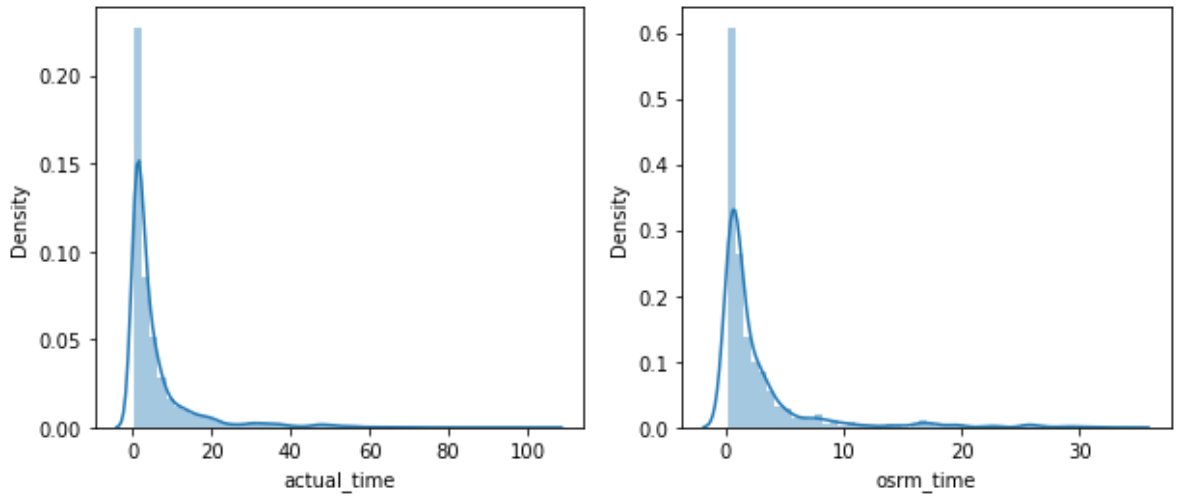
```

```

In [105]:

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

Out[105]:

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



In [ ]:

```

```

In [106]:

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

Out[106]:

```
KstestResult(statistic=0.2945265573327934, pvalue=0.0)
```

In [ ]:

```

```

In [107]:

```
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=22.939951756140957, pvalue=7.665862059973965e-114)
Ttest_indResult(statistic=23.252221309415223, pvalue=8.043492628034228e-117)
Ttest_indResult(statistic=22.46393670785672, pvalue=2.2713658234383966e-109)
Ttest_indResult(statistic=22.465682779856774, pvalue=2.1879549579322686e-10
9)
Ttest_indResult(statistic=21.89231817869703, pvalue=4.115727040596413e-104)
```

Observations:

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 [ ]:

```

```

In [110]:

```
actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```

Out[110]:

```
(5.945176711435117, 9.35554782297388)
```

In [ ]:

```

```

In [111]:

```
osrm_time["osrm_time"].mean(),osrm_time["osrm_time"].std()
```

Out[111]:

```
(2.697313896200314, 4.537654251845703)
```

In [ ]:

```

```

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 [ ]:

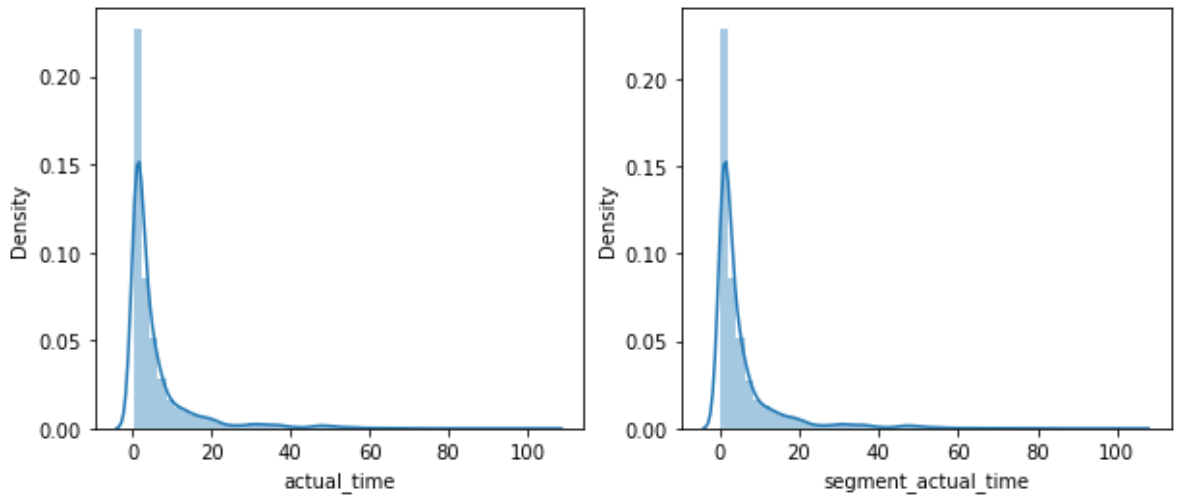
```
1
```

In [113]:

```
1 plt.figure(figsize=(10,4))
2 plt.subplot(121)
3 sns.distplot(((actual_time["actual_time"])))
4 plt.subplot(122)
5 sns.distplot(((segment_actual_time["segment_actual_time"])))
```

Out[113]:

```
<AxesSubplot:xlabel='segment_actual_time', ylabel='Density'>
```



In [ ]:

```
1
```

In [114]:

```
1 for i in range(7):
2     print(stats.ttest_ind((actual_time["actual_time"].sample(3000)),
3                 (segment_actual_time["segment_actual_time"].sample(3000))))
```

```
Ttest_indResult(statistic=-0.028939685075501133, pvalue=0.9769136575612524)
Ttest_indResult(statistic=0.6943388695062013, pvalue=0.4874966123418428)
Ttest_indResult(statistic=1.3767280783858906, pvalue=0.16864771885784374)
Ttest_indResult(statistic=1.006055758506429, pvalue=0.31442933262168515)
Ttest_indResult(statistic=-1.207690856147261, pvalue=0.22721381440041422)
Ttest_indResult(statistic=0.5021588274956957, pvalue=0.6155742356326531)
Ttest_indResult(statistic=2.639585480208236, pvalue=0.00832222766360579)
```

In [ ]:

```
1
```

Observation :

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 [ ]:

```
1
```

In [117]:

```
1 actual_time["actual_time"].mean(),actual_time["actual_time"].std()
```

Out[117]:

```
(5.945176711435117, 9.35554782297388)
```

In [ ]:

```
1
```

In [118]:

```
1 segment_actual_time["segment_actual_time"].mean(),segment_actual_time["segment_actual_t
2
```

Out[118]:

```
(5.898204764797215, 9.270799413152762)
```

In [ ]:

```
1
```

Analysing osrm Time & segment-osrm-time :

H0: segment actual time <= OSRM time

Ha: segment actual time > OSRM time

In [ ]:

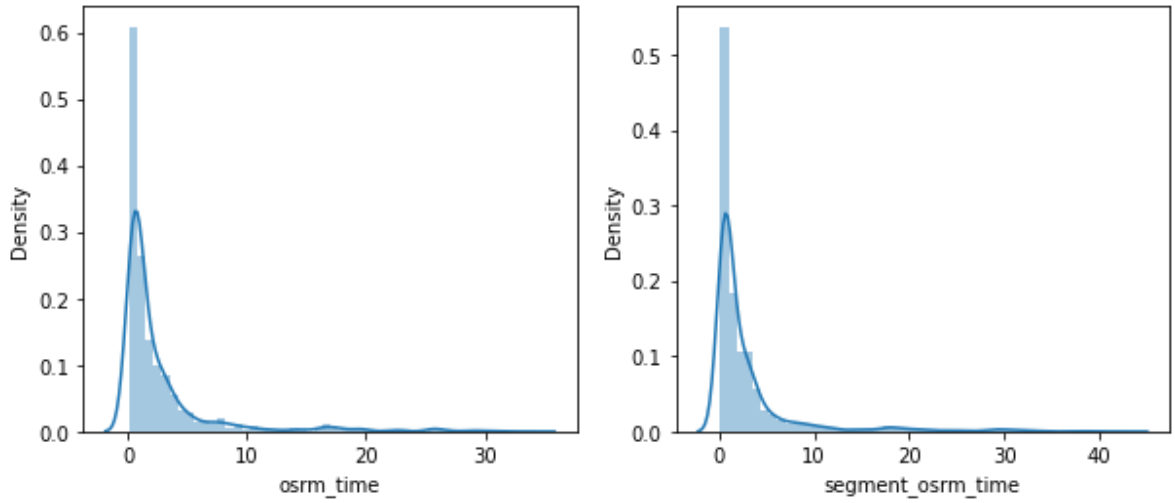
```
1
```

In [120]:

```
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"])))
```

Out[120]:

```
<AxesSubplot:xlabel='segment_osrm_time', ylabel='Density'>
```



In [ ]:

```

```

In [121]:

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

```

```
Ttest_indResult(statistic=-2.9121593234175074, pvalue=0.0018013158791902366)
Ttest_indResult(statistic=-2.3877503726545837, pvalue=0.00849131075797141)
Ttest_indResult(statistic=-3.4042849464057774, pvalue=0.0003338612233028138
7)
Ttest_indResult(statistic=-3.100649912234784, pvalue=0.0009699601171759891)
Ttest_indResult(statistic=-1.820438285478094, pvalue=0.0343710480692617)
Ttest_indResult(statistic=-1.9545731718970933, pvalue=0.025339931687329422)
Ttest_indResult(statistic=-2.9925297529885304, pvalue=0.001389012229054396)
```

Observations:

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 [124]:

```
1 osrm_time["osrm_time"].mean(),osrm_time["osrm_time"].std()
```

Out[124]:

```
(2.697313896200314, 4.537654251845703)
```

In [ ]:

```
1
```

In [125]:

```
1 segment_osrm_time["segment_osrm_time"].mean(),segment_osrm_time["segment_osrm_time"].st
2
```

Out[125]:

```
(3.0158297901059705, 5.242367441693007)
```

In [ ]:

```
1
```

Analysing Distances measures :

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

H0 : Segment OSRM distnace <= OSRM distnace

Ha : Segment OSRM distnace > OSRM distnace

In [ ]:

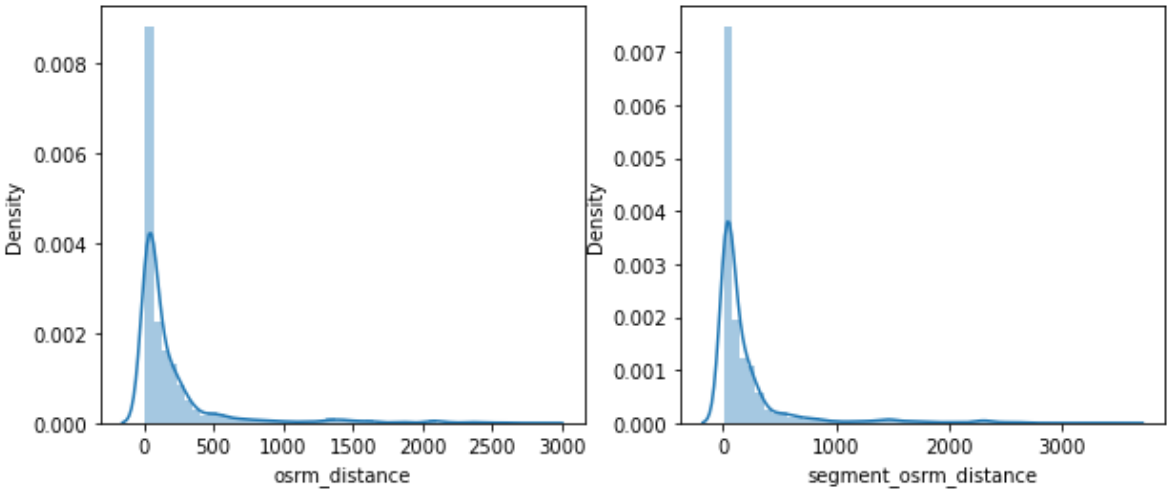
```
1
```

In [127]:

```
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"])))
```

Out[127]:

```
<AxesSubplot:xlabel='segment_osrm_distance', ylabel='Density'>
```



In [ ]:

```

```

In [128]:

```
stats.ks_2samp(osrm_distance["osrm_distance"],segment_osrm_distance["segment_osrm_dista

```

Out[128]:

```
KstestResult(statistic=0.03948167645272321, pvalue=1.8042208791084262e-10)
```

In [ ]:

```

```

In [129]:

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

```
Ttest_indResult(statistic=-2.4982014826777075, pvalue=0.006249227434144001)
Ttest_indResult(statistic=-3.711812507257795, pvalue=0.00010344887141157233)
Ttest_indResult(statistic=-2.08147742450463, pvalue=0.018707804498092572)
Ttest_indResult(statistic=-1.4901558824377619, pvalue=0.06810739929212521)
Ttest_indResult(statistic=-2.3589326208587407, pvalue=0.009173354765117644)
Ttest_indResult(statistic=-0.7785543272267806, pvalue=0.21813035871375314)
Ttest_indResult(statistic=-1.7200644645980867, pvalue=0.042725834908559183)
```

In [ ]:

```
```

In [130]:

```
osrm_distance["osrm_distance"].mean(),osrm_distance["osrm_distance"].std()
```

Out[130]:

```
(204.83672531551625, 370.74927471335496)
```

In [ ]:

```
```

In [131]:

```
segment_osrm_distance["segment_osrm_distance"].mean(),segment_osrm_distance["segment_os
```

Out[131]:

```
(223.20116128771042, 416.6283742907418)
```

Observations :

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

In [ ]:

```
```

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 [ ]:

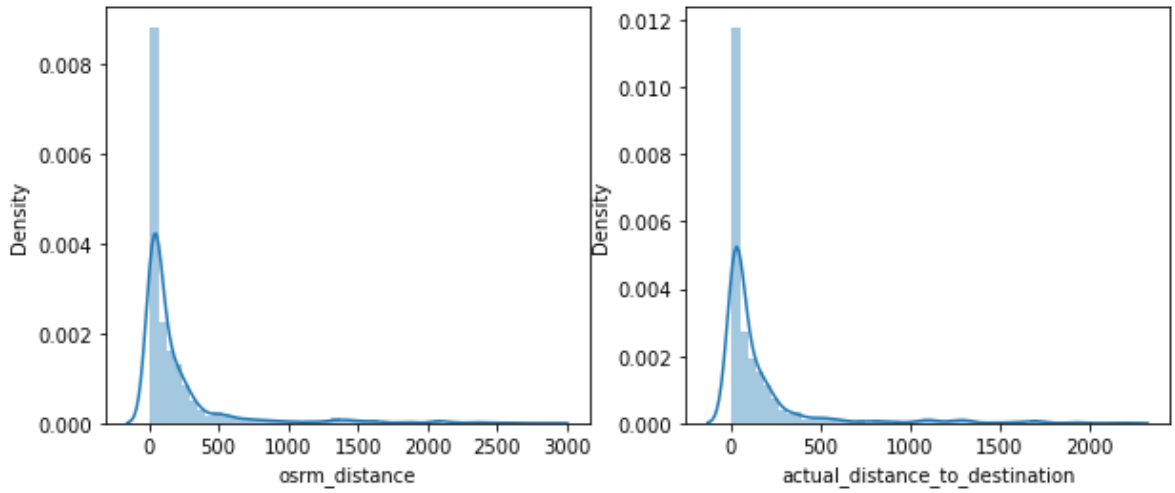
```
1
```

In [136]:

```
1 plt.figure(figsize=(10,4))
2 plt.subplot(121)
3 sns.distplot(((osrm_distance["osrm_distance"])))
4 plt.subplot(122)
5 sns.distplot(((actual_distance_to_destination["actual_distance_to_destination"])))
6
```

Out[136]:

```
<AxesSubplot:xlabel='actual_distance_to_destination', ylabel='Density'>
```



In [ ]:

```
1
```

In [137]:

```
1 stats.ks_2samp(osrm_distance["osrm_distance"],actual_distance_to_destination["actual_di
2
```

Out[137]:

```
KstestResult(statistic=0.11837753931295136, pvalue=6.578385372142345e-91)
```

In [ ]:

```
1
```

In [138]:

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

```
Ttest_indResult(statistic=6.015122636350758, pvalue=9.302463762748574e-10)
Ttest_indResult(statistic=5.962226228023425, pvalue=1.2862195866496017e-09)
Ttest_indResult(statistic=6.158795643605484, pvalue=3.8060402919103325e-10)
Ttest_indResult(statistic=7.725333881148406, pvalue=6.1134956212716876e-15)
Ttest_indResult(statistic=3.899108041672851, pvalue=4.8589481643358685e-05)
```

observations:

From left sided ttest , we can conclude

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

In [ ]:

```

```

In [141]:

```
osrm_distance["osrm_distance"].mean(),osrm_distance["osrm_distance"].std()
```

Out[141]:

```
(204.83672531551625, 370.74927471335496)
```

In [ ]:

```

```

In [142]:

```
actual_distance_to_destination["actual_distance_to_destination"].mean(),actual_distance

```

Out[142]:

```
(164.4733217454422, 305.5408288910492)
```

In [ ]:

```

```

Merging All the numerical Fields as per TripID:

In [ ]:

```
1
```

In [144]:

```
1 distances = segment_osrm_distance.merge(actual_distance_to_destination.merge(osrm_dista
2                                                                             on="trip_u
3                                                                            on="trip_uu
4
```

In [ ]:

```
1
```

In [145]:

```
1 time = segment_osrm_time.merge(osrm_time.merge(segment_actual_time.merge(actual_time.me
2                                         on="trip_uuid",
3                                         ),on="trip_uuid"),on="trip_uuid"),on="trip_uui
4
```

In [ ]:

```
1
```

In [146]:

```
1 Merge1 = time.merge(distances,on="trip_uuid",
2                                         )
```

In [ ]:

```
1
```

In [147]:

```
Merge1
```

Out[147]:

	trip_uuid	segment_osrm_time	osrm_time	segment_actual_time	actual_time
0	trip-153671041653548748	16.800000	12.383333	25.800000	26.033333
1	trip-153671042288605164	1.083333	1.133333	2.350000	2.383333
2	trip-153671043369099517	32.350000	29.016667	55.133333	55.783333
3	trip-153671046011330457	0.266667	0.250000	0.983333	0.983333
4	trip-153671052974046625	1.916667	1.950000	5.666667	5.683333
...	...	...	...	...	...
14812	trip-153861095625827784	1.033333	1.033333	1.366667	1.383333
14813	trip-153861104386292051	0.183333	0.200000	0.350000	0.350000
14814	trip-153861106442901555	1.466667	0.900000	4.683333	4.700000
14815	trip-153861115439069069	3.683333	3.066667	4.300000	4.400000
14816	trip-153861118270144424	1.116667	1.133333	4.566667	4.583333

14817 rows × 10 columns

In [ ]:

```

```

Merging Location details and route_type and Numerical data on TripID :

In [ ]:

```

```

In [149]:

```
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 [ ]:

```

```

In [150]:

```
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 [ ]:

```

```

In [151]:

```
trip_records = Merged.copy()
```

In [152]:

```
trip_records["route_type"] = trip_records["route_type"].apply(lambda x:x[0])
```

In [153]:

```
route_to_merge = data.groupby("trip_uuid")["route_schedule_uuid"].unique().reset_index(

```

In [154]:

```
trip_records = trip_records.merge(route_to_merge,on="trip_uuid",how="outer")

```

In [ ]:

```

```

In [155]:

```
trip_records["route_schedule_uuid"] = trip_records["route_schedule_uuid"].apply(lambda
trip_records
```

Out[155]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	dest
0	trip-153671041653548748	FTL	[Bhopal, Kanpur]	[Kanpur, Gurgaon]	[Bhopal Madhya Pradesh, Kanpur Uttar Pradesh]	[Kar
1	trip-153671042288605164	Carting	[Tumkur, Doddablpur]	[Doddablpur, Chikblapur]	[Tumkur Karnataka, Doddablpur Karnataka]	[Dod Chi
2	trip-153671043369099517	FTL	[Bengaluru, Gurgaon]	[Gurgaon, Chandigarh]	[Bengaluru Karnataka, Gurgaon Haryana]	C
3	trip-153671046011330457	Carting	Mumbai	Mumbai	Mumbai Hub Maharashtra	Mu
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]	C
14813	trip-153861104386292051	Carting	FBD	Faridabad	FBD Haryana	
14814	trip-153861106442901555	Carting	Kanpur	Kanpur	Kanpur Uttar Pradesh	Ka
14815	trip-153861115439069069	Carting	[Tirunelveli, Eral, Tirchchndr, Thisayanvilai,...	[Eral, Tirchchndr, Thisayanvilai, Peikulam, Ti...	[Tirunelveli Tamil Nadu, Eral Tamil Nadu, Tirc...	Tircl
14816	trip-153861118270144424	FTL	[Hospet, Sandur]	[Sandur, Bellary]	[Hospet Karnataka, Sandur Karnataka]	[

14817 rows × 18 columns

In [ ]:

```
1
```

In [156]:

```
1 # route_df['source'] = route_df['source'].str.strip("{''}")
```

In [ ]:

```
1
```

In [157]:

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

Out[157]:

```
trip_uuid                            0
route_type                           0
source_city                         10
destination_city                     8
source_city_state                   10
destination_city_state               8
source_state                        10
destination_state                    8
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
```

In [ ]:

```
1
```

In [158]:

```
1 trip_records.loc[trip_records.isnull().any(axis=1)]
```

Out[158]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	destina
5289	trip-153733592611290696	Carting	Gurgaon	NaN	Gurgaon Haryana	
5778	trip-153739792417979729	FTL	Luxettipet	NaN	Luxettipet Telangana	
5961	trip-153741501937042684	Carting	Gurgaon	NaN	Gurgaon Haryana	
8762	trip-153776806236494354	FTL	NaN	Mainpuri	NaN	Mainpu
8796	trip-153777348608709328	FTL	Aligarh	NaN	Aligarh Uttar Pradesh	
9835	trip-153791004076950775	FTL	NaN	Chikmagalur	NaN	Chikma
10562	trip-153800051661903546	FTL	NaN	NaN	NaN	
11468	trip-153811367563100850	FTL	NaN	Mainpuri	NaN	Mainpu
12097	trip-153820032399976293	FTL	NaN	Mainpuri	NaN	Mainpu
13104	trip-153835867702133730	FTL	NaN	Mathura	NaN	Mathur
13168	trip-153836697913613926	FTL	NaN	Mainpuri	NaN	Mainpu
13313	trip-153839879406683648	FTL	Sonipat	NaN	Sonipat Haryana	
13408	trip-153841850974526339	FTL	Delhi	NaN	Delhi Delhi	
13644	trip-153843937115921268	FTL	NaN	Mathura	NaN	Mathur
13793	trip-153846056503320607	FTL	NaN	Mainpuri	NaN	Mainpu
14199	trip-153852612674280168	FTL	NaN	Mathura	NaN	Mathur
14453	trip-153857174991144707	FTL	Delhi	NaN	Delhi Delhi	

In [ ]:

```
1
```

In [159]:

```
1 trip_records[trip_records["trip_uuid"]=="trip-153852612674280168"]
```

Out[159]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	destina
14199	trip-153852612674280168	FTL	NaN	Mathura	NaN	Mathur

In [ ]:

```
1
```

In [160]:

```
1 trip_records.dropna(axis= 0,how = 'any',inplace = True)
```

In [ ]:

```
1
```

In [161]:

```
1 trip_records["source_city"] = trip_records["source_city"].astype("str").str.strip("[]")
2 trip_records["destination_city"] = trip_records["destination_city"].astype("str").str.s
3 trip_records["source_city_state"] = trip_records["source_city_state"].astype("str").str
4 trip_records["destination_city_state"] = trip_records["destination_city_state"].astype(
5
6 trip_records["source_state"] = trip_records["source_state"].astype("str").str.strip("[]
7 trip_records["destination_state"] = trip_records["destination_state"].astype("str").str
8
```

In [ ]:

```
1
```

Checking if any null values left in Trip Records Data :

In [163]:

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

Out[163]:

```
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
```

In [ ]:

```
1
```

In [164]:

```
1 trip_records.loc[trip_records.isnull().any(axis=1)]
```

Out[164]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	destination_city_state	s

In [165]:

```
1 trip_records.corr()
```

Out[165]:

	segment_osrm_time	osrm_time	segment_actual_time	act
segment_osrm_time	1.000000	0.993508	0.953207	(
osrm_time	0.993508	1.000000	0.957931	(
segment_actual_time	0.953207	0.957931	1.000000	(
actual_time	0.953965	0.958794	0.999920	
time_taken_btwn_odstart_and_od_end	0.918915	0.926779	0.961160	(
start_scan_to_end_scan	0.918962	0.926970	0.961171	(
segment_osrm_distance	0.996092	0.991847	0.956287	(
actual_distance_to_destination	0.987628	0.993557	0.953238	(
osrm_distance	0.992050	0.997610	0.958532	(

In [ ]:

```
1
```

In [166]:

```
1 trip_records.to_csv("trip_records.csv")
```

In [ ]:

```
1
```

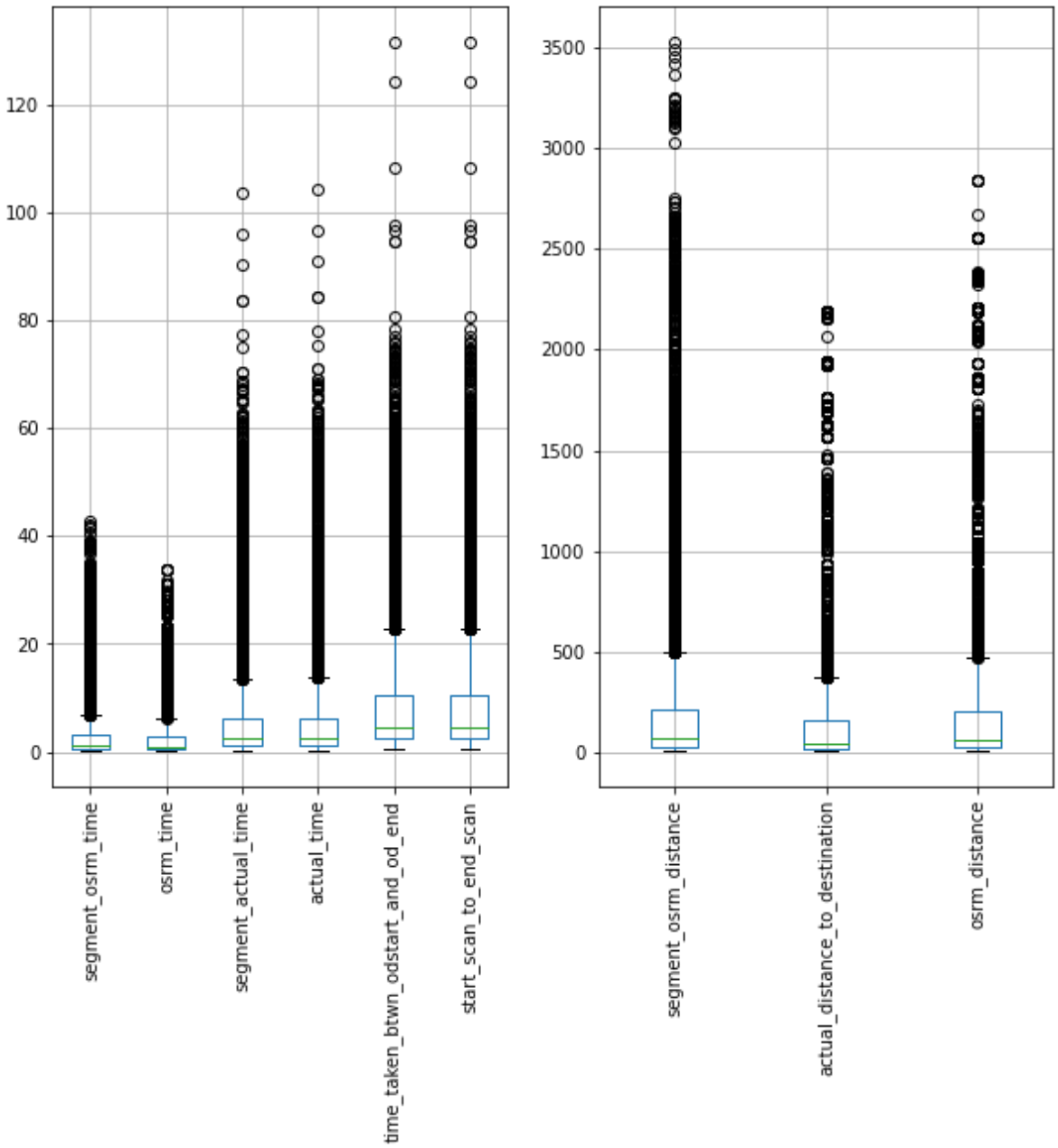
Treating Outliers :

In [ ]:

```
1
```

In [168]:

```
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 [ ]:

```
1
```

In [169]:

```
1 outlier_treatment  = trip_records.copy()
```

In [ ]:

```
1
```

In [170]:

```
1 outlier_treatment_num = outlier_treatment[['segment_osrm_time', 'osrm_time',
2        'segment_actual_time', 'actual_time',
3        'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan',
4         'segment_osrm_distance', 'actual_distance_to_destination',
5         'osrm_distance']]
```

In [ ]:

```
1
```

In [ ]:

```
1 # outlier_treatment_num[(np.abs(stats.zscore(outlier_treatment_num)) < 3).all(axis=1)]
2
```

After removing outliers from all numerical features :

In [ ]:

```
1
```

In [172]:

```
trip_records_without_outliers = trip_records.loc[outlier_treatment_num[(np.abs(stats.zs
trip_records_without_outliers
```

Out[172]:

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

14144 rows × 18 columns

In [ ]:

```

```

In [173]:

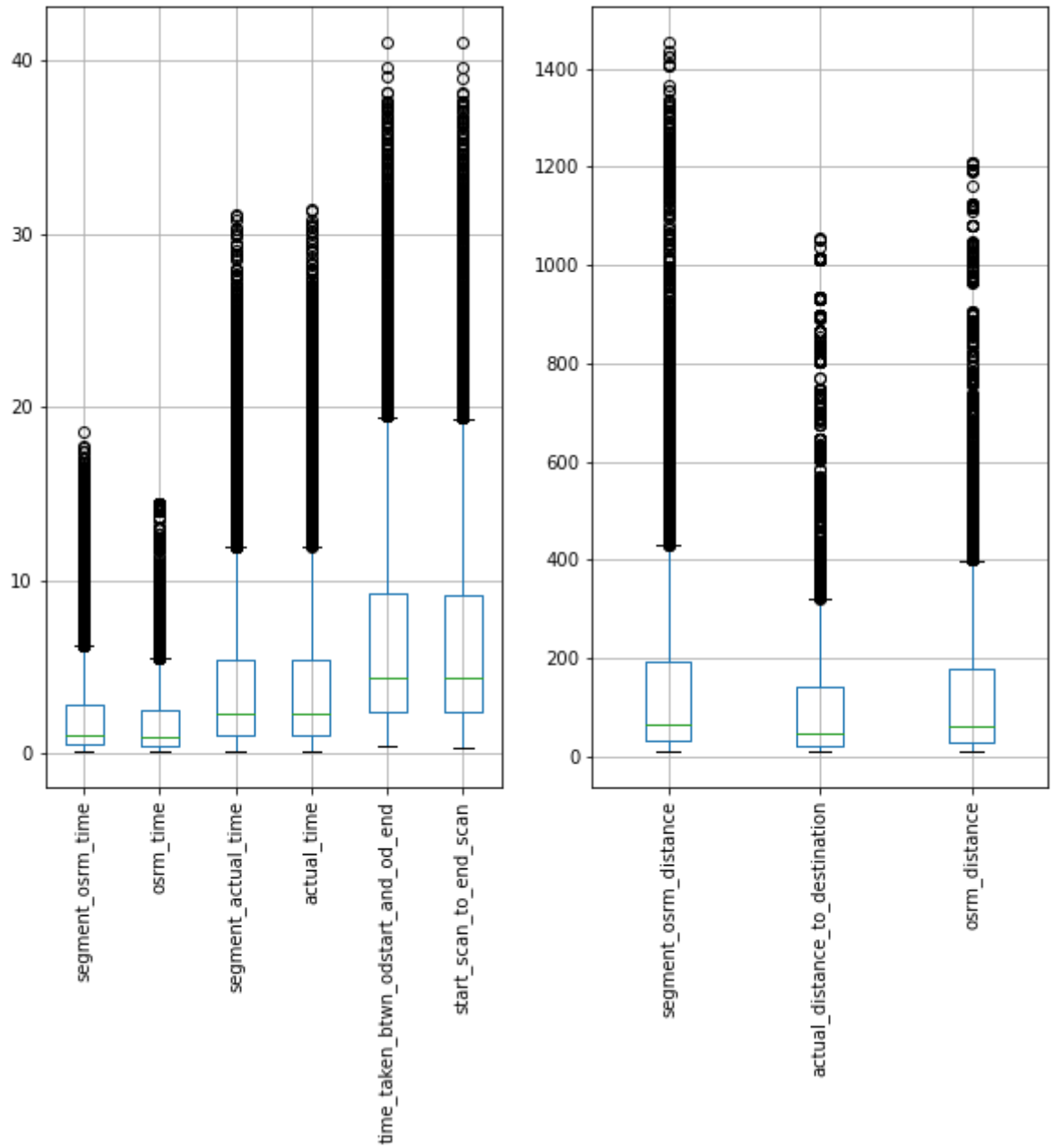
```
trip_records_without_outliers = trip_records_without_outliers[['trip_uuid','route_type'
       'segment_actual_time', 'actual_time',
       'time_taken_btwn_odstart_and_od_end', 'start_scan_to_end_scan',
       'segment_osrm_distance', 'actual_distance_to_destination',
       'osrm_distance']]
```

In [ ]:

```

```

In [174]:

```
plt.figure(figsize = (10,8))
plt.subplot(121)
trip_records_without_outliers[['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_without_outliers[['segment_osrm_distance', 'actual_distance_to_destinatior
       'osrm_distance']].boxplot()
plt.xticks(rotation =90)
plt.show()
```



In [ ]:

```

```

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 [ ]:

```
1
```

In [176]:

```
1 trip_records_without_outliers["destination_source_locations"] = trip_records_without_ou
2 trip_records_without_outliers.drop(["source_city_state","destination_city_state"],axis
3
```

In [ ]:

```
1
```

In [177]:

```
1 sc_dc = trip_records_without_outliers.groupby(["destination_source_locations"])["trip_u
2
```

In [178]:

```
1 # trip_records.groupby(['source_state','destination_state'])["trip_uuid"].nunique().sor
2
```

In [ ]:

```
1
```

In [179]:

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

In [180]:

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

In [ ]:

```

```

In [ ]:

```

```

In [181]:

```
encoded_data
```

Out[181]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
0	1.383333	0.950000	3.183333	3.233333	
1	1.150000	0.883333	2.666667	2.700000	
2	1.183333	0.966667	3.316667	3.333333	
3	0.700000	0.733333	1.316667	1.316667	
4	0.783333	0.666667	1.750000	1.766667	
...	...	...	...	...	
14139	1.416667	1.416667	2.250000	2.250000	
14140	0.916667	0.933333	3.066667	3.100000	
14141	0.300000	0.300000	0.583333	0.583333	
14142	1.050000	1.066667	3.116667	3.116667	
14143	0.400000	0.333333	0.616667	0.616667	

14144 rows × 18 columns

In [182]:

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

```

Out[182]:

```
['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 [ ]:

```

```

In [183]:

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

In [ ]:

```

```

In [184]:

```
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[184]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odstart_a
0	-0.269289	-0.409762	-0.220506	-0.215127	
1	-0.359897	-0.438982	-0.324769	-0.322057	
2	-0.346953	-0.402458	-0.193600	-0.195078	
3	-0.534643	-0.504727	-0.597197	-0.599406	
4	-0.502283	-0.533947	-0.509751	-0.509184	

In [ ]:

```
1
```

In [185]:

```
scaler = MinMaxScaler()
MinMax_data = scaler.fit_transform(encoded_data[['segment_osrm_time','osrm_time','segme
 'time_taken_btwn_odstart_and_od_end','start_scan_to_end_scan','segment_osrm_distance',
 '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_end',
 'segment_osrm_distance','actual_distance_to_destination','osrm_distance'])
MinMax_data.head()
```

Out[185]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odstart_a
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 [ ]:

```
1
```

In [186]:

```
std_data
```

Out[186]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
0	-0.269289	-0.409762	-0.220506	-0.215127	
1	-0.359897	-0.438982	-0.324769	-0.322057	
2	-0.346953	-0.402458	-0.193600	-0.195078	
3	-0.534643	-0.504727	-0.597197	-0.599406	
4	-0.502283	-0.533947	-0.509751	-0.509184	
...	...	...	...	...	
14139	-0.256345	-0.205223	-0.408852	-0.412279	
14140	-0.450506	-0.417067	-0.244050	-0.241860	
14141	-0.689972	-0.694657	-0.745182	-0.746434	
14142	-0.398730	-0.358628	-0.233960	-0.238518	
14143	-0.651140	-0.680047	-0.738456	-0.739751	

14144 rows × 9 columns

In [ ]:

```
1
```

In [187]:

```
one_hot_encoded_data = encoded_data[["route_type_Carting","route_type_FTL","city_Categ
 "city_Category 2","city_Category 3","city_Category 4",
 "city_Category 5","city_Category 6","city_Category 7"]]
Standardized_Data = pd.concat([std_data,one_hot_encoded_data],axis = 1)
Min_Max_Scaled_Data = pd.concat([MinMax_data,one_hot_encoded_data],axis = 1)
Standardized_Data.sample(5)
```

Out[187]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
13700	0.125507	0.233075	0.173001	0.175834	
7335	-0.191624	-0.139478	-0.324769	-0.325399	
13252	-0.243400	-0.168698	-0.193600	-0.191737	
11107	-0.029823	-0.124868	-0.324769	-0.318715	
5894	-0.567003	-0.555862	-0.704823	-0.702994	

In [ ]:

```
1
```

In [188]:

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

Out[188]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odst
1686	0.013514	0.017442	0.048518	0.048026	
3899	0.023423	0.032558	0.044205	0.044824	
9285	0.132432	0.152326	0.243127	0.242263	
11083	0.183784	0.238372	0.566038	0.561366	
10316	0.425225	0.550000	0.415094	0.414088	

In [ ]:

```
1
```

Route analysis :

In [ ]:

```
1
```

In [191]:

```
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_values(by
                                                                                     as
F.columns = ["route_schedule_uuid","#source_states"
             ,"#destination_states"]
G = trip_records.groupby("route_schedule_uuid")["actual_distance_to_destination"].mean(
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"]
```

In [ ]:

```
1
```

In [192]:

```
I = data.groupby("route_schedule_uuid")[["source_city",
                                         "destination_city"]].nunique().sort_values(by=
                                                                                    as
I.columns = ["route_schedule_uuid","#source_cities"
             ,"#destination_cities"]
```

In [ ]:

```
1
```

In [193]:

```
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 [ ]:

```

```

In [194]:

```
# route_records.sort_values(by="Average_Actual_distance_to_destination",ascending=False
```

In [ ]:

```

```

In [195]:

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

Out[195]:

```
route_schedule_uuid                       0
#source_cities                            0
#destination_cities                       0
Number_of_Trips                           6
Average_Actual_distance_to_destination    6
#source_states                            0
#destination_states                       0
destination_states                        0
source_states                             0
source_cities                             0
route_type                                0
destination_cities                        0
dtype: int64
```

In [ ]:

```

```

In [196]:

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

In [ ]:

```
1
```

In [197]:

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

route_records["destination_states"] = route_records["destination_states"].astype("str")
route_records
```

Out[197]:

tual_distance_to_destination	#source_states	#destination_states	destination_states	source_states
281.596486	2	2	Assam Arunachal Pradesh	Assam Arunachal Pradesh
332.602225	2	2	Assam Meghalaya	Assam Meghalaya
351.611796	1	1	Rajasthan	Rajasthan
195.257193	1	2	Karnataka Goa	Karnataka
178.737233	1	1	Rajasthan	Rajasthan
...	...	...	...	...
16.225821	1	1	Maharashtra	Maharashtra
10.334462	1	1	Maharashtra	Maharashtra
17.617532	1	1	Maharashtra	Maharashtra
10.137219	1	1	Maharashtra	Maharashtra
15.467701	1	1	Karnataka	Karnataka

In [ ]:

```
1
```

In [198]:

```
1 route_records["ROUTE"] = route_records["source_cities"] + " -- " + route_records["desti
2
```

In [199]:

```
1 route_records.drop(["route_schedule_uuid"],axis = 1,inplace=True)
```

In [200]:

```
1 first_column = route_records.pop('ROUTE')
2 route_records.insert(0, 'ROUTE', first_column)
```

In [201]:

```
1 route_records["SouceToDestination_city"] = route_records["source_cities"].str.split(" '
2 first_column = route_records.pop('SouceToDestination_city')
3 route_records.insert(0, 'SouceToDestination_city', first_column)
```

In [202]:

```
1 route_records
```

Out[202]:

	SouceToDestination_city	ROUTE	#source_cities	#destination_cities	Number_of_Trips
0	Guwahati TO LakhimpurN	Guwahati LakhimpurN Dhemaji Likabali Tezpur Pa...	13	11	14.0
1	Guwahati TO Tura	Guwahati Rangia Kokrajhar Dhubri Bilasipara Tu...	10	10	12.0
2	Jaipur TO Tarnau	Jaipur Chomu Reengus Sikar Bikaner Didwana Suj...	10	10	20.0
3	Mangalore TO Udupi	Mangalore Udupi Kundapura Bhatkal Honnavar Kum...	9	9	9.0
4	Ajmer TO Raipur	Ajmer Beawar Bilara Bijainagar Kekri Nasirabad...	9	8	20.0
...	...	...	...	...	...
1497	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	2.0
1498	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	15.0
1499	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	19.0
1500	Mumbai TO Mumbai	Mumbai -- Mumbai	1	1	15.0
1501	Bengaluru TO Bengaluru	Bengaluru -- Bengaluru	1	1	7.0

1498 rows × 13 columns

In [ ]:

```
1
```

In [203]:

```
1 route_records.to_csv("route_records.csv")
```

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

In [ ]:

```
1
```

Busiest Route Analysis :

Number of Trips between cities , sorted highest to lowest

Top 20 source and destination cities wihc have high freqency of trips in between .0

In [206]:

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

Out[206]:

	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 [210]:

```
Number_of_trips_between_cities.loc[Number_of_trips_between_cities["source_city_state"]
```

Out[210]:

	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

In [ ]:

```

```

If we talk about , not having equal source and destination states , 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
```

- it is also been observed that lots of deliveries are happening to airpots

- like : 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 [ ]:

```
1
```

In [214]:

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

Out[214]:

	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities	#
1465	LowerParel -- Mumbai	53.0	16.428868	1	
1426	Mumbai -- Bhiwandi	46.0	20.199445	1	
808	Gurgaon -- Gurgaon	43.0	29.740842	1	
679	Jaipur -- Ambabadi Jaipur	41.0	15.348495	1	
1257	Noida -- Del	40.0	10.882902	1	
1368	Hyderabad -- Hyderabad	39.0	35.695641	1	
1273	Mumbai -- Mumbai	37.0	13.882863	1	
1359	Mumbai -- Mumbai	36.0	17.526251	1	
1303	Bhiwandi -- Mumbai	35.0	21.241534	1	
700	Mumbai -- Mumbai	34.0	15.906614	1	
751	Mumbai -- Mumbai	33.0	15.668726	1	
1060	Bengaluru -- Bengaluru	33.0	28.067004	1	
972	Hyderabad -- Hyderabad	32.0	21.835579	1	
793	Sonipat -- Sonipat	32.0	11.691243	1	
1184	Mumbai -- Bhiwandi Mumbai	32.0	21.601109	1	
1177	Bhiwandi -- Mumbai	30.0	21.396002	1	
874	Bengaluru -- Bengaluru	30.0	28.055789	1	
1354	Bengaluru -- Bengaluru	27.0	27.967087	1	

	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities	#
921	Faridabad -- FBD	26.0	9.677121	1	
1480	Sonipat -- Sonipat	26.0	12.182486	1	
1041	Mumbai -- Bhiwandi	25.0	19.942191	1	
877	Faridabad -- Gurgaon	25.0	47.091622	1	
1249	Bengaluru -- Bengaluru	25.0	28.019668	1	
833	Bhiwandi -- Mumbai	25.0	21.531705	1	
1125	CCU -- Kolkata	24.0	16.367507	1	

In [ ]:

```
1
```

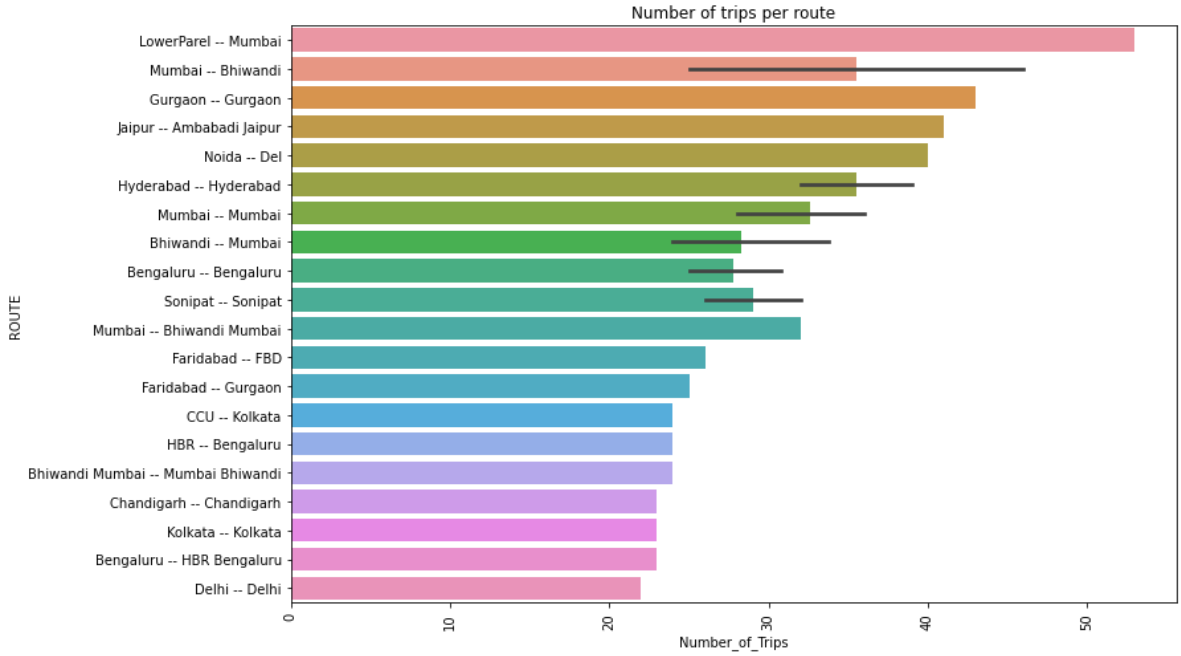
Top Routes having Maximum Number of Trips between/within the source and destinations .

In [ ]:

```
1
```

In [216]:

```
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()
```

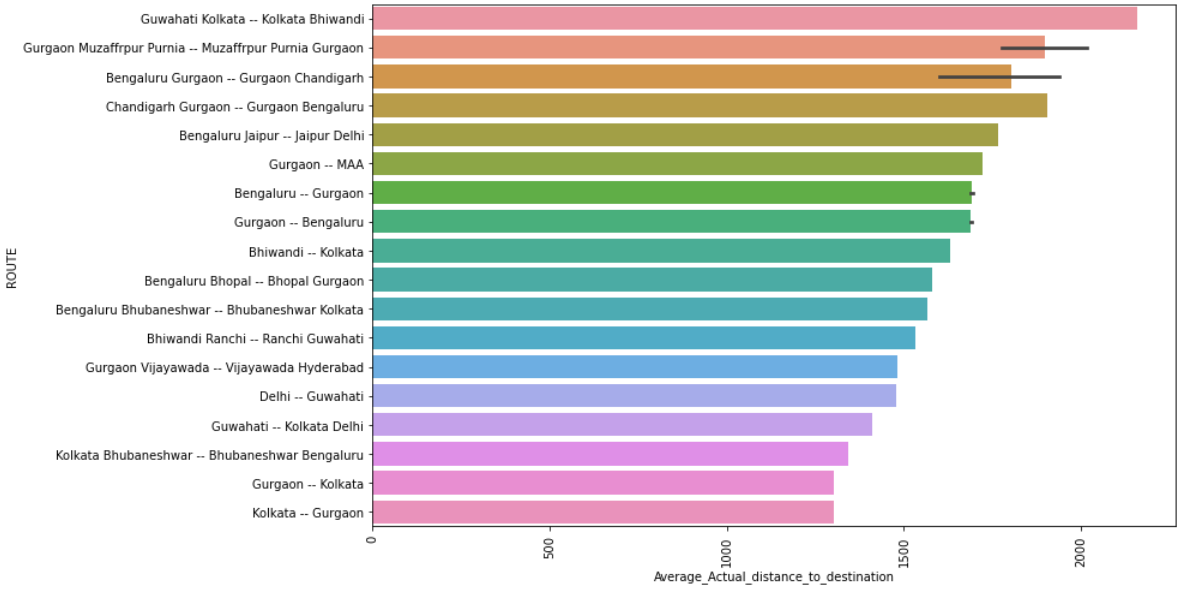


In [ ]:

```
1
```

In [217]:

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

X = route_records[["ROUTE", "Average_Actual_distance_to_destination",
                   ]].sort_values(by="Average_Actual_distance_to_destination",ascending=Fal
sns.barplot(y = X["ROUTE"],
            x = X["Average_Actual_distance_to_destination"])
plt.xticks(rotation = 90)
plt.show()
```



In [ ]:

```
1
```

Observation :

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 .

In [ ]:

```
1
```

Busiest and Longest Routes :

In [ ]:

```
1
```

In [224]:

```
lumber_of_Trips"].quantile(0.75))].sort_values(by="Average_Actual_distance_to_destination")
```

In [ ]:

```
1
```

In [225]:

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

Busiest_and_Longest_Routes_top25
```

Out[225]:

	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destinatior
27	Srikakulam Rajam Bobbili Parvathipuram Palakon...	Rajam Salur Parvathipuram Palakonda Srikakulam...	22.0	154.49528
115	Islampur Karad Chiplun Khed Kolhapur	Karad Chiplun Khed Kolhapur Islampur	17.0	156.00950
399	Chandigarh RoopNagar AnandprShb	Chandigarh RoopNagar AnandprShb	22.0	157.53068
310	Aurangabad Jalna Sillod	Jalna Aurangabad Sillod	16.0	158.74477
84	Warangal Khanpur Mahbubabad Yellandu Rayaparthi	Khanpur Mahbubabad Yellandu Rayaparthi Warangal	20.0	161.27928
255	Hyderabad Bhuvanagiri Mothkur Thirumalagiri	Bhuvanagiri Mothkur Thirumalagiri Hyderabad	16.0	161.75414
176	Jamtara Dumka Pakur Dhanbad	Dumka Pakur Dhanbad Jamtara	20.0	162.79344
1302	Hyderabad	Gulbarga	21.0	164.69333
97	Rudrapur Nainital Ranikhet Almora Pithorgarh	Nainital Ranikhet Almora Pithorgarh Rudrapur	17.0	165.51322
167	Thiruvadanai Oriyur Manamelkudi Sivaganga	Oriyur Manamelkudi Sivaganga Thiruvadanai	16.0	166.52760
210	Kakinada Tuni Rajamundry Mandapeta	Tuni Rajamundry Mandapeta Kakinada	21.0	166.68913
257	Biswan Sitapur Lakhimpur Gola	Sitapur Lakhimpur Gola Dhaurahara	21.0	167.43297
70	Jaipur Kotputli Paota Achrol Alwar	Alwar Paota Jaipur Achrol	22.0	170.15604
52	Madurai Sivakasi Sankaran Maharajapuram Rajpal...	Sivakasi Sankaran Rajpalayam Madurai Maharajap...	17.0	174.51114

	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destinatior
241	Bhadrachalam Manuguru Sathupally Khammam	Manuguru Sathupally Khammam Tallada	21.0	175.04142
1484	Silchar	Guwahati	17.0	175.76337
229	Muzaffrpur Darbhanga Benipur Jhanjharpur	Darbhanga Benipur Jhanjharpur Muzaffrpur	22.0	177.20300
135	Tirupati Puttur Koduru Rajampet	Puttur Koduru Rajampet Tirupati	19.0	177.32716
212	Ongole Kanigiri Kandukur Kavali	Kanigiri Kandukur Kavali Ongole	22.0	177.80215
38	Kamareddy Bodhan Banswada Yellareddy Medak Nar...	Bodhan Banswada Yellareddy Medak Narsapur Kama...	21.0	177.92333
4	Ajmer Beawar Bilara Bijainagar Kekri Nasirabad...	Beawar Bilara Badnaur Kekri Nasirabad Ajmer Bi...	20.0	178.73723
20	Junagadh Veraval Kodinar Una Talala Mangrol	Junagadh Kodinar Una Talala Mangrol Veraval	19.0	179.53859
617	Visakhapatnam Srikakulam	Srikakulam Visakhapatnam	16.0	180.63138
161	Muzaffrpur DalsinghSarai Rusera Manjhaul	DalsinghSarai Rusera Manjhaul Muzaffrpur	21.0	186.02028
270	Kurnool Wanaparthy Raichur JoguGadwal nan	Raichur Kurnool JoguGadwal Wanaparthy nan	16.0	187.03677

In [ ]:

```
1
```

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

which are : Chandigarh TO Bengaluru

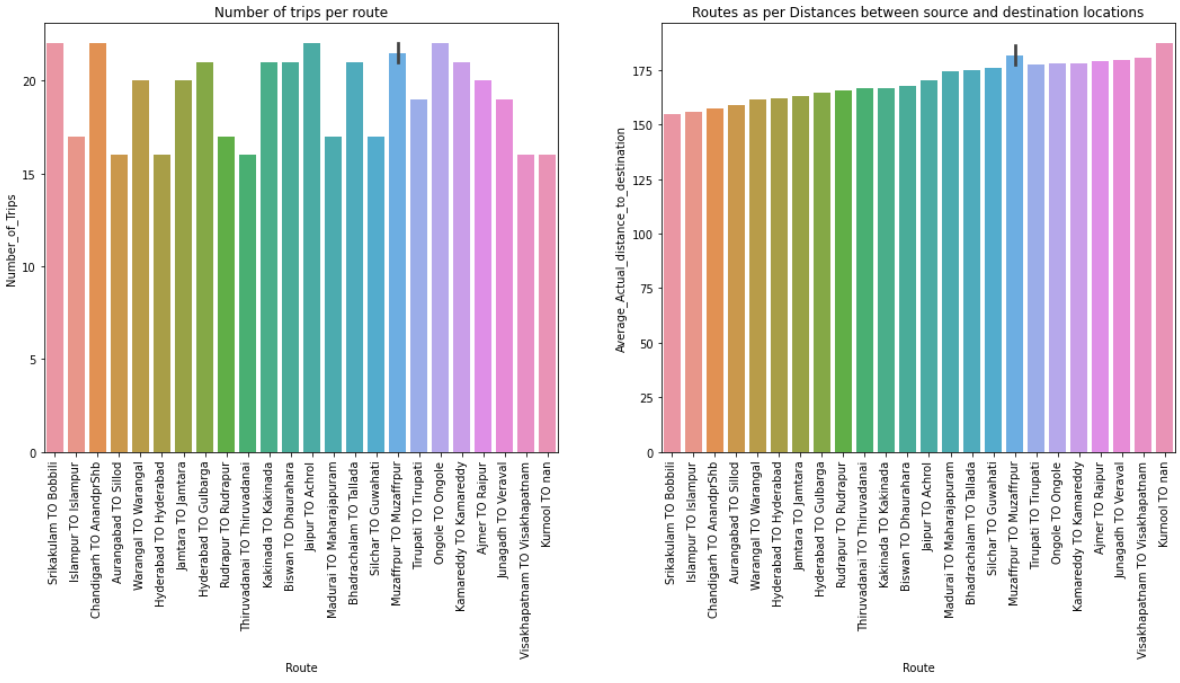
```
Gurgaon TO Bengaluru
Bengaluru TO Kolkata
Guwahati TO Delhi
Delhi  TO Kolkata
Chandigarh TO Gurgaon
Gurgaon TO Hydrabad
Benglore TO Ahmedabad
Surat TO Delhi
Gurgaon TO Ahmedabad
```

In [227]:

```
Busiest_and_Longest_Routes_top25["Route"] = Busiest_and_Longest_Routes_top25["source_ci
Busiest_and_Longest_Routes_top25.drop(["source_cities","destination_cities"],axis = 1,i

```

In [228]:

```
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_destination'
plt.xticks(rotation = 90)
plt.show()
```



Observation:

Above charts showing the routes (souce and destinations locations with higest trips between locations) and having long distances.

In [231]:

```
1 route_records.columns
```

Out[231]:

```
Index(['SouceToDestination_city', 'ROUTE', '#source_cities',
       '#destination_cities', 'Number_of_Trips',
       'Average_Actual_distance_to_destination', '#source_states',
       '#destination_states', 'destination_states', 'source_states',
       'source_cities', 'route_type', 'destination_cities'],
      dtype='object')
```

In [ ]:

```
1
```

Routes : passing through maxinum number of cities :

In [ ]:

```
1
```

In [233]:

```
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[233]:

	SouceToDestination_city	Number_of_Trips	Average_Actual_distance_to_destination	#source_
0	Guwahati TO LakhimpurN	14.0	281.596486	
2	Jaipur TO Tarnau	20.0	351.611796	
1	Guwahati TO Tura	12.0	332.602225	
3	Mangalore TO Udupi	9.0	195.257193	
4	Ajmer TO Raipur	20.0	178.737233	
5	Mainpuri TO Tilhar	12.0	207.247057	
8	Hassan TO Koppa	21.0	200.497832	
15	Shrirampur TO Sangamner	20.0	204.509529	
7	Musiri TO Tiruchi	19.0	219.845121	
9	Bijnor TO Bijnor	17.0	209.400685	
10	Dausa TO Lalsot	17.0	232.408310	
17	Tinusukia TO Dibrugarh	16.0	111.098543	
12	Pondicherry TO Pondicherry	12.0	230.253602	
14	Mysore TO Mysore	12.0	154.324190	
6	Golaghat TO Guwahati	11.0	258.546587	
13	Varanasi TO Varanasi	8.0	82.545019	
16	Vijayawada TO Suryapet	8.0	407.029391	
11	Hyderabad TO Miryalguda	7.0	420.603709	
27	Srikakulam TO Bobbili	22.0	154.495283	
36	Pukhrayan TO Kanpur	22.0	139.834945	
48	Dhule TO Shirpur	22.0	150.016233	
30	Madhupur TO Madhupur	21.0	252.072259	
38	Kamareddy TO Kamareddy	21.0	177.923330	
42	Noida TO Khurja	21.0	208.714043	
20	Junagadh TO Veraval	19.0	179.538596	

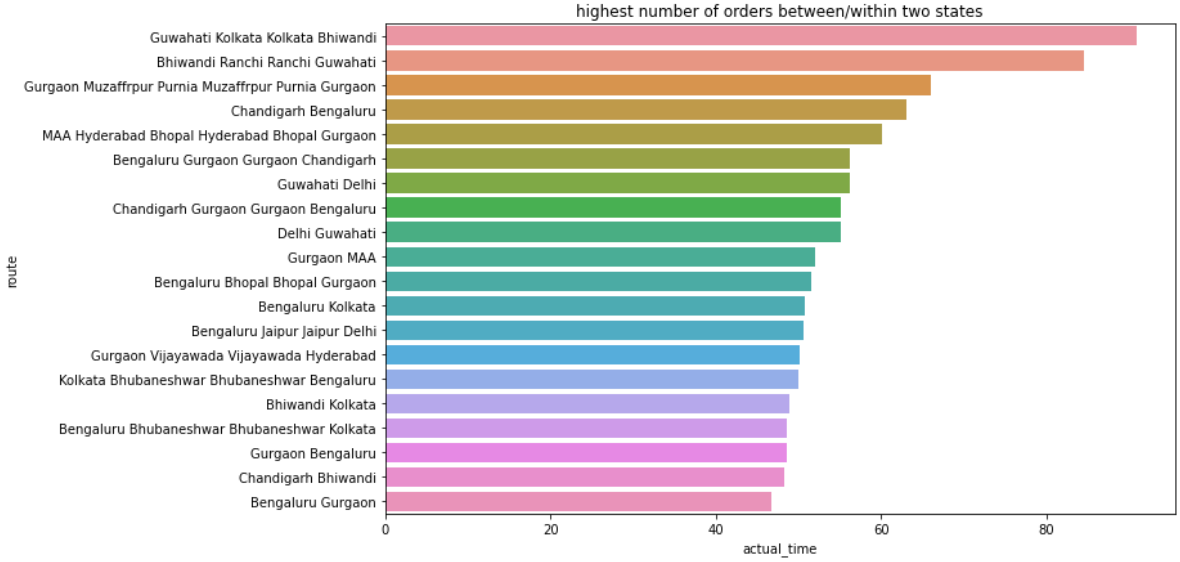
In [ ]:

```
1
```

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

In [235]:

```
1  Longest_route_as_per_actual_trip_time = trip_records.groupby(["source_city",
2                        "destination_city"])["actual_time"].mean().sort_values(ascending=
3  Longest_route_as_per_actual_trip_time["route"] = Longest_route_as_per_actual_trip_time[
4  Longest_route_as_per_actual_trip_time.drop(["source_city",
5                                              "destination_city"],axis = 1,inplace=True)
6  Longest_route_as_per_actual_trip_time
7  plt.figure(figsize=(11,7))
8  sns.barplot(y = Longest_route_as_per_actual_trip_time["route"],
9              x = Longest_route_as_per_actual_trip_time["actual_time"],)
10 plt.title("highest number of orders between/within two states")
11 plt.show()
```



In [ ]:

```
1
```

highest number of Trips happening between/within two states :

In [241]:

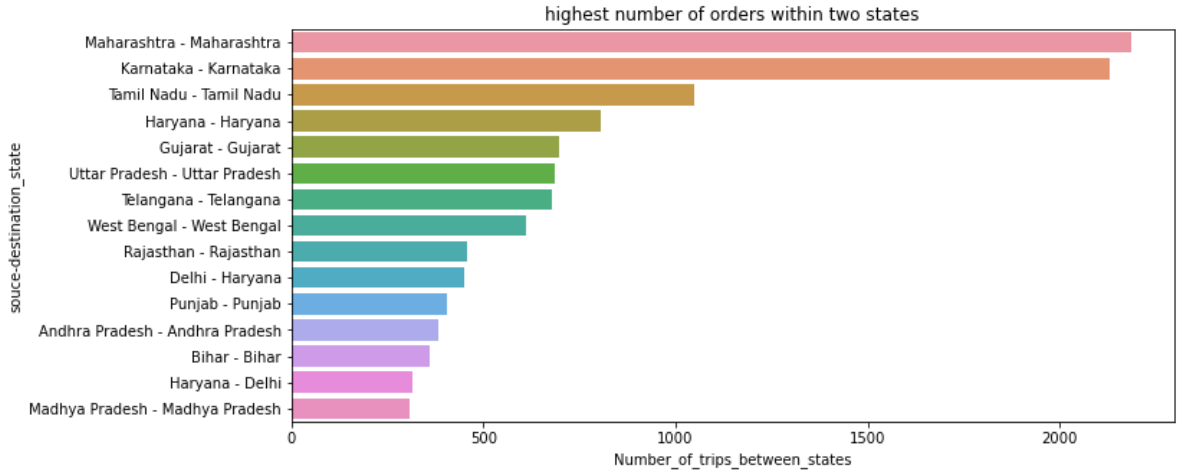
```
1  highest_order_between_states = data.groupby(["source_state",
2                                               "destination_state"])["trip_uuid"].nunique
3
```

In [ ]:

```
1
```

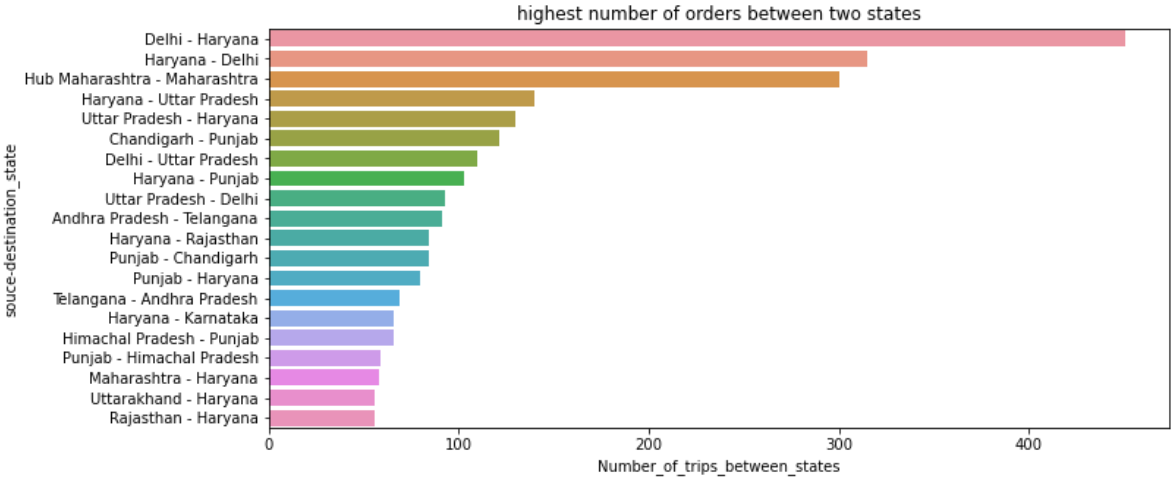
In [242]:

```
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 [243]:

```
HOBS = data.groupby(["source_state","destination_state"])["trip_uuid"].nunique().sort_v
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()
```



In [ ]:

```

```

From above charts ,

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

Top 20 warehouses with heavy traffic :

In [246]:

```
destination_traffic = data.groupby(["destination_city_state"])["trip_uuid"].nunique().r
source_traffic = data.groupby(["source_city_state"])["trip_uuid"].nunique().reset_index
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","#Trips
transactions["TripsTraffic"] = transactions["#Trips_s"]+transactions["#Trips_d"]
transactions.drop(["#Trips_s","#Trips_d","destination_city_state"],axis = 1,inplace=Tru
transactions.columns = ["Warehouse_City(Junction)","TripsTraffic"]
```

In [247]:

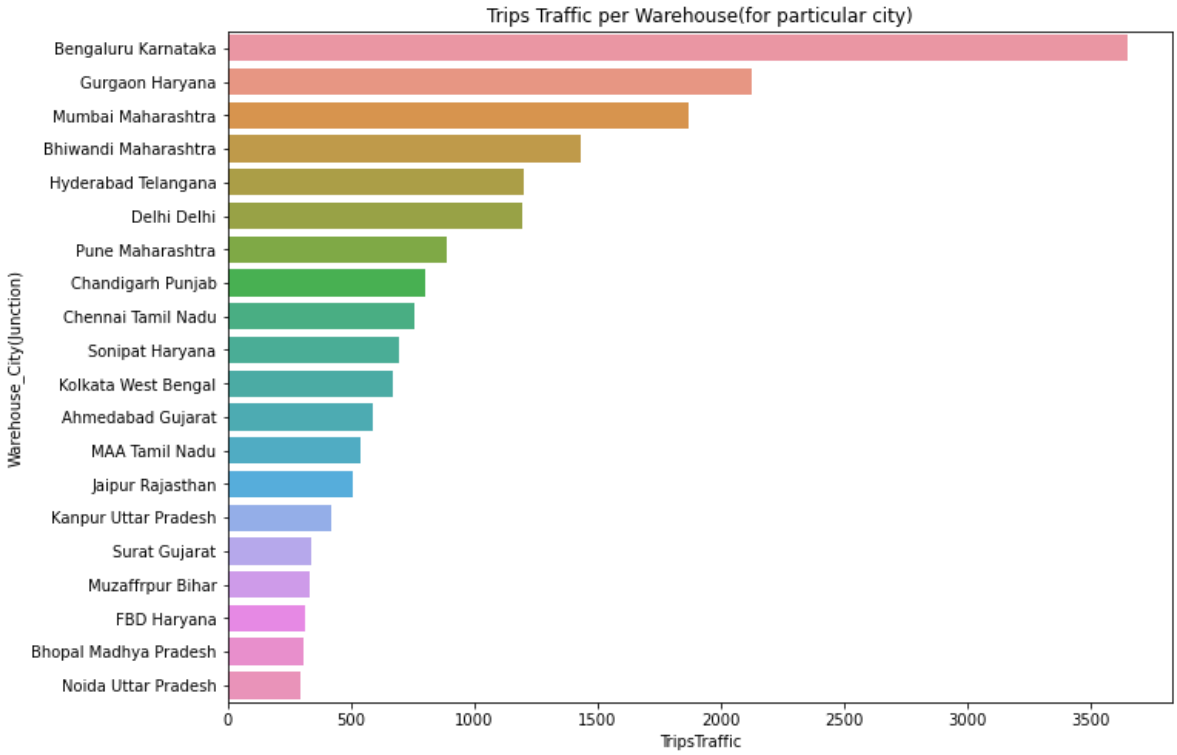
```
T = transactions.sort_values(by=["TripsTraffic"],ascending=False).head(20)
```

In [ ]:

```

```

In [248]:

```
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 [ ]:

```

```

Top 20 Busiest Warehouse (junctions) as per trips

traffic at the juction : are

'Bengaluru Karnataka', 'Gurgaon Haryana', 'Mumbai Maharashtra', 'Bhiwandi Maharashtra', 'Hyderabad Telangana', 'Delhi Delhi', 'Pune Maharashtra', 'Chandigarh Punjab', 'Chennai Tamil Nadu', 'Sonipat Haryana', 'Kolkata West Bengal', 'Ahmedabad Gujarat', 'MAA Tamil Nadu', 'Jaipur Rajasthan', 'Kanpur Uttar Pradesh', 'Surat Gujarat', 'Muzaffrpur Bihar', 'FBD Haryana', 'Bhopal Madhya Pradesh', 'Noida Uttar Pradesh'

In [ ]:

```
1
```

In [252]:

```
1 trip_records.groupby(["source_state","destination_state"])["trip_uuid"].count().sort_va
2
```

Out[252]:

	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

In [ ]:

```
1
```

Inferences and Recommendations :

Insights and Observations :

14817 different trips happened between source to destinations during 2018 , September and October.

1504 delivery routes on which trips are happenig.

we have 1508 unique source centers and 1481 unique destination centers

From 14817 total different trips , we have 8908 (60%) of the trip-routes are Carting , which consists of small vehicles and 5909 (40%) of total trip-routes are FTL : which are Full Truck Load get to the destination sooner. as no other pickups or drop offs along the way .

Hypothesis tests Results : from 2 sample t-test ,we can also conclude that

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

population average actual_time is less than population average start_scan_to_end_scan.

population mean Actual time taken to complete delivery and population mean time_taken_btwn_od_start_and_od_end 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.

From Exploratory Data Analysis

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.

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

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

From Bar charts , and calculated tables in analysis , we can observe that higest trips are happening is with in the particular cities, in terms of average distance between destinations , we can observe Guwahati to Mumbai , Benglore to Chandigarh ,Benglore to Delhi , Benglore to Gurgaon are the longest routes.

the souce to destination city routes having largest numbers of trip happening having large distnaces :

Guwahati TO Bhiwandi, Bengaluru TO Chandigarh, Bengaluru TO Delhi,Gurgaon TO MAA Chennai Airport,Bhiwandi TO Kolkata, Bengaluru TO Kolkata, Gurgaon TO Hyderabad, Gurgaon TO Kolkata

the routes which covered multiple cities in between source and destination :

Most covered cities routes are : Guwahati TO LakhimpurN , Jaipur TO Tarnau , Guwahati TO Tura , Mangalore TO Udupi , Ajmer TO Raipur , Mainpuri TO Tilhar . which passes through more than 8 cities.

Routes which are busiest from source to destinations and states in which highest activities are noticed :

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, Haryana, Telangana, Gujarat , West Benglore and Uttar Pradesh are some states having above 1000 trips.

From above chart are some warehouse having Maximum traffic and hence busiest junctions.

Bengaluru Karnataka, Gurgaon Haryana, Mumbai Maharashtra, Hyderabad Telangana, Delhi, Pune Maharashtra, Chandigarh Punjab, Chennai Tamil Nadu, Sonipat Haryana, Kolkata West Bengal, Ahmedabad Gujarat, MAA Tamil Nadu, Jaipur Rajasthan, Kanpur Uttar Pradesh, Surat Gujarat, Muzaffrpur Bihar, FBD Haryana, Bhopal Madhya Pradesh, Noida Uttar Pradesh.

Recommendations :

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

Incresing the connectivity in tier 2 and tier 3 cities along with profession tie-ups with several e-commerce giants can increase the revenue as well as the reputation on connectivity across borders.

We can work on optimizing the scanning time on both ends which is start scanning time and end scanning time so that the delivery time can be equated to the OSRM estimated delivery time.

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```