Business Case

Delhivery - Feature Engineering

Suman Debnath

Introduction

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

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

Dataset

Dataset link: delhivery_data.csv

The dataset have the following fields:

- 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

Summary

Insights and Observations:

- A total of 14,817 distinct trips were recorded between various source and destination points during September and October of 2018.
- There were 1,504 delivery routes utilized for these trips.
- The data comprises 1,508 unique source centers and 1,481 unique destination centers.

- Among the 14,817 total distinct trips:

    - 8,908 (60%) were categorized under Carting, employing small vehicles,
    - 5,909 (40%) were categorized as FTL (Full Truck Load), facilitating quicker deliveries due to the absence of intermediate pickups or drop-offs.

Hypothesis Test Outcomes:

- The two-sample t-test provided insights such as:
    - The average `time_taken_btwn_odstart_and_od_end` for the population equates to the average `start_scan_to_end_scan` for the population.
    - The population's average `actual_time` is lesser than the average `start_scan_to_end_scan`.

Exploratory Data Analysis:

- The analysis reveals that cities like Mumbai (Maharashtra), Delhi, Gurgaon (Haryana), Bengaluru (Karnataka), Hyderabad (Telangana), Chennai (Tamil Nadu), Ahmedabad (Gujarat), Pune (Maharashtra), Chandigarh, and Kolkata (West Bengal) record a high number of intra-city trips.
- Regarding unequal source and destination states, the highest number of trips occur between: Delhi to Gurgaon, Gurgaon to Bengaluru, Bhiwandi/Mumbai to Pune (Maharashtra), and Sonipat to Gurgaon (Haryana).
- Numerous deliveries to airports were noted, including routes like Chennai to MAA Chennai International Airport, Pune to Pune Airport (PNQ), Kolkata to CCU West Bengal Kolkata International Airport, and Bengaluru to BLR-Bengaluru International Airport.

- From the Bar charts and calculated tables, it's apparent that the majority of trips occur within specific cities. Moreover, regarding average distance between destinations, routes like Guwahati to Mumbai, Bengaluru to Chandigarh, Bengaluru to Delhi, and Bengaluru to Gurgaon are among the longest.

Significant Long-Distance Routes:

- Routes with substantial distances and high trip occurrences include: Guwahati to Bhiwandi, Bengaluru to Chandigarh, Bengaluru to Delhi, Gurgaon to MAA Chennai Airport, Bhiwandi to Kolkata, Bengaluru to Kolkata, Gurgaon to Hyderabad, and Gurgaon to Kolkata.

Multi-City Routes:

- Notable multi-city routes include: Guwahati to LakhimpurN, Jaipur to Tarnau, Guwahati to Tura, Mangalore to Udupi, Ajmer to Raipur, Mainpuri to Tilhar - each traversing through more than eight cities.

Busiest Routes and High-Activity States:

- Delhi to Haryana emerges as the busiest route with over 400 trips. Other busy routes include Haryana to Uttar Pradesh, Chandigarh to Punjab, and Delhi to Uttar Pradesh.
- States like Maharashtra, Karnataka, Tamil Nadu, Haryana, Telangana, Gujarat, West Bengal, and Uttar Pradesh have recorded over 1,000 trips (Cells In.[173]).

Major Traffic Warehouses:

- Warehouses experiencing significant traffic, and thus identified as busiest junctions, include: 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), Muzaffarpur (Bihar), FBD (Haryana), Bhopal (Madhya Pradesh), and Noida (Uttar Pradesh).

Recommendation

- Based on the analysis, employing Carting (small vehicles) for intra-city deliveries is advisable to expedite delivery times. For long-distance or heavy load deliveries, utilizing Heavy trucks is recommended. By adhering to this strategy, delivery times can be optimized which in turn, may enhance revenue generation according to the operational requirements.
- Enhancing connectivity in tier 2 and tier 3 cities, coupled with strategic partnerships with various e-commerce giants, can potentially boost both revenue and reputation regarding cross-border connectivity.
- Efforts can be channeled towards refining the scanning process at both the initiation and conclusion phases. By optimizing the start and end scanning times, it's plausible to align the actual delivery times closer to the OSRM estimated delivery times.

Detailed Analysis

Importing all the `libs`

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import figure
import warnings
import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import t
import plotly.express as px

import scipy.stats as stats

warnings.filterwarnings('ignore')
%matplotlib inline
```

Loading the `data`

In [2]:
```
# data_set = 'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/0(
data_set = 'delhivery_data.csv'
df = pd.read_csv(data_set)
```

Exploratory Data Exploration (EDA)

In [3]:
```
df.shape
```

Out[3]:
```
(144867, 24)
```

In [4]:
```
df.head()
```

Out[4]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

5 rows × 24 columns

In [5]:
```
df.dtypes
```

Out[5]:
```
data                                object
trip_creation_time                  object
route_schedule_uuid                 object
route_type                          object
trip_uuid                           object
source_center                       object
source_name                         object
destination_center                  object
destination_name                    object
od_start_time                       object
od_end_time                         object
start_scan_to_end_scan             float64
is_cutoff                             bool
cutoff_factor                        int64
cutoff_timestamp                    object
actual_distance_to_destination     float64
actual_time                        float64
osrm_time                          float64
osrm_distance                      float64
factor                             float64
segment_actual_time                float64
segment_osrm_time                  float64
segment_osrm_distance              float64
segment_factor                     float64
dtype: object
```

In [6]:
```
df.columns
```

Out[6]:
```
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'],
      dtype='object')
```

Check for `null` values

In [7]:
```
np.any(df.isna())
```

Out[7]:
```
True
```

In [8]:
```
df.isna().sum()
```

Out[8]:
```
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
```

Observation

- Features : `source_name` and `destination_name` having some missing values

Check for `duplicate` values

In [9]:
```
np.any(df.duplicated())
```

Out[9]:
```
False
```

In [10]:
```
df.info()
```

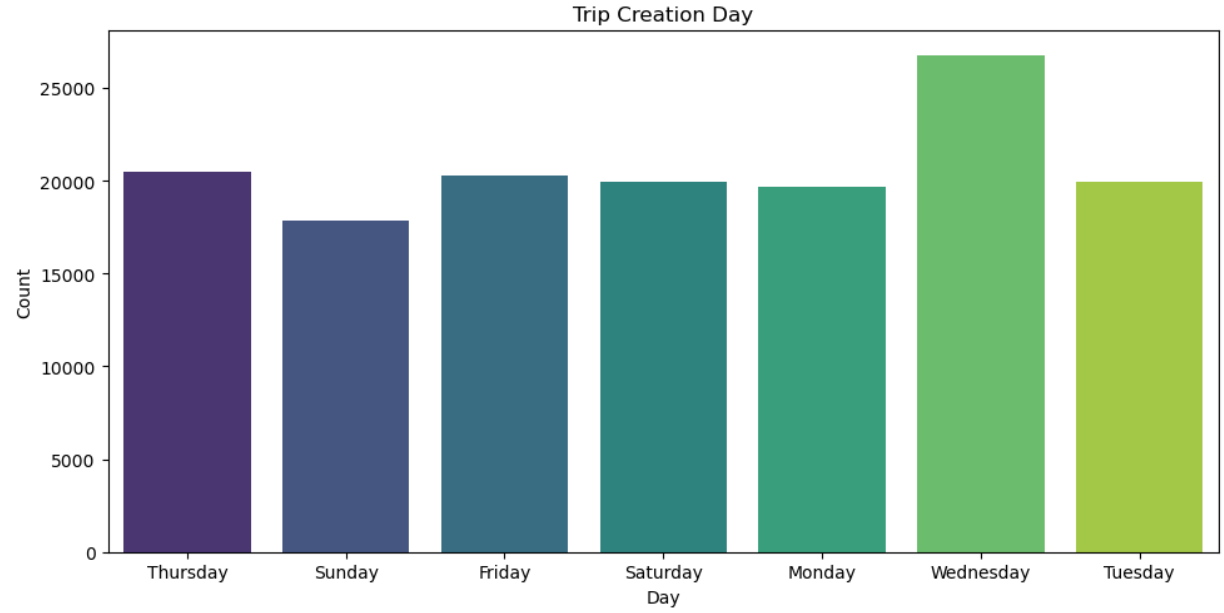
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                           Non-Null Count   Dtype  
---  ------                           --------------   -----  
 0   data                             144867 non-null  object 
 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
```

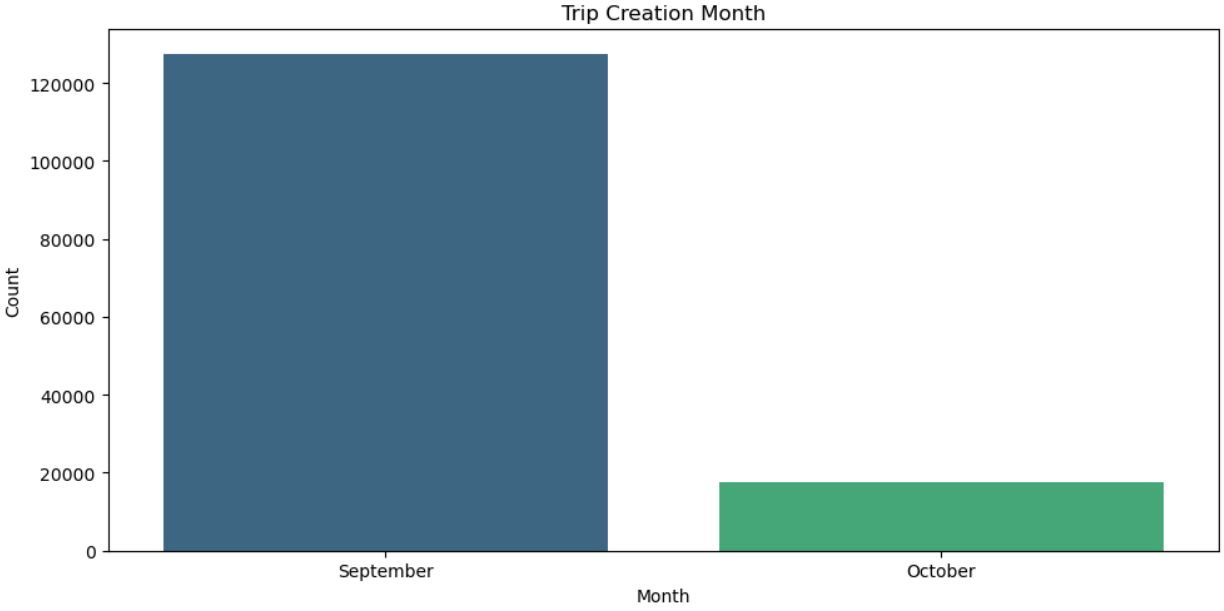
Changing data type for `data and time` related features

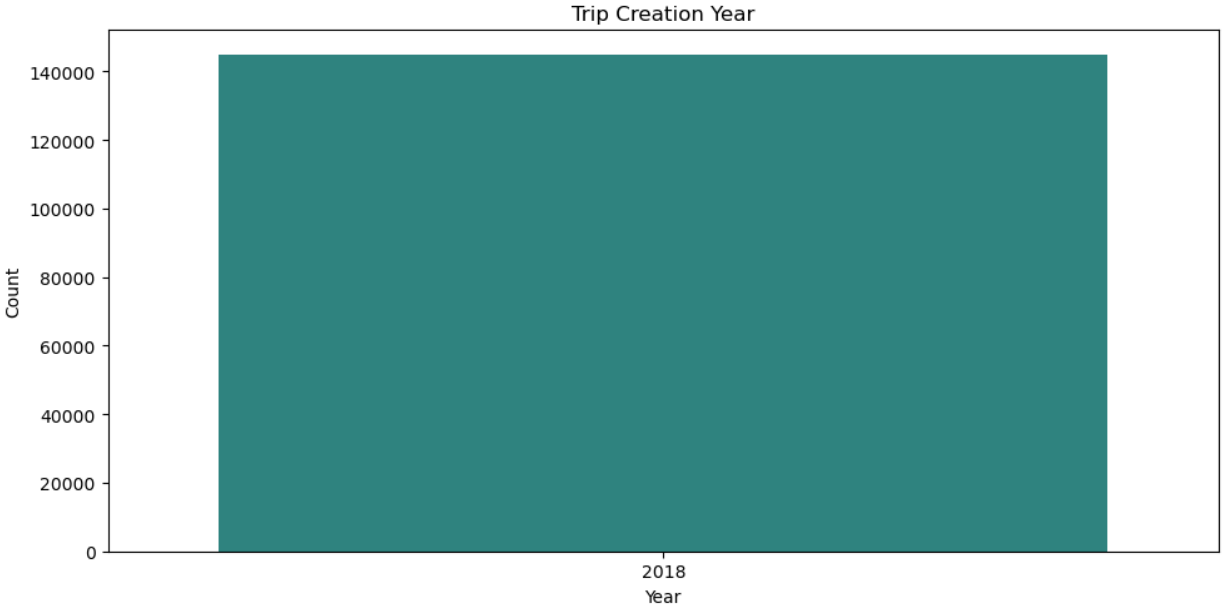
In [11]:
```
df["od_end_time"] = pd.to_datetime(df["od_end_time"])
df["od_start_time"] = pd.to_datetime(df["od_start_time"])
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"])

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 [12]:
```
fig, axs = plt.subplots(3, 1, figsize=(10, 15))

# Plot for trip_creation_day
sns.countplot(data=df, x='trip_creation_day', ax=axs[0], palette='viridis')
axs[0].set_title('Trip Creation Day')
axs[0].set_xlabel('Day')
axs[0].set_ylabel('Count')

# Plot for trip_creation_month
sns.countplot(data=df, x='trip_creation_month', ax=axs[1], palette='viridis')
axs[1].set_title('Trip Creation Month')
axs[1].set_xlabel('Month')
axs[1].set_ylabel('Count')

# Plot for trip_creation_year
sns.countplot(data=df, x='trip_creation_year', ax=axs[2], palette='viridis')
```

```
axs[2].set_title('Trip Creation Year')
axs[2].set_xlabel('Year')
axs[2].set_ylabel('Count')

plt.tight_layout()
plt.show()
```







In [13]:
```
df.nunique()
```

Out[13]:
```
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

- As we can see the we have 14,817 distinct trips from various sources to destinations.
- Additionally, there are a total of 1,504 delivery routes available.

Different kind of `routes`

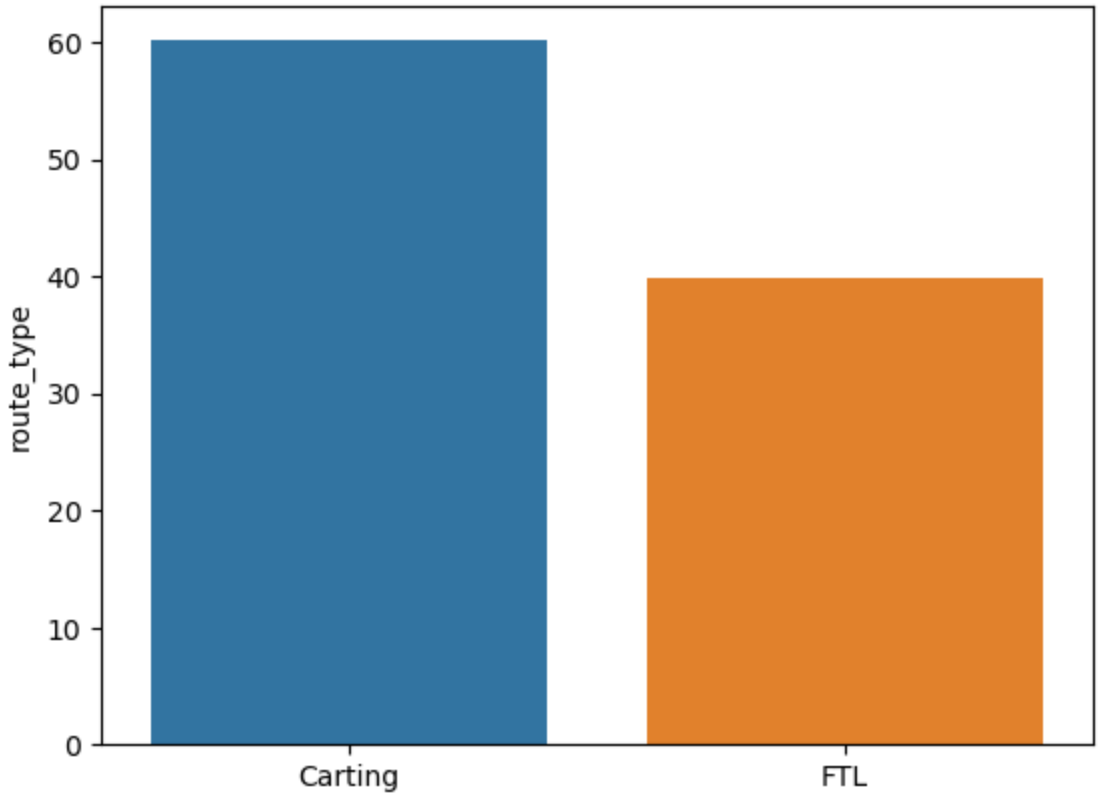
In [14]:
```
df.groupby("trip_uuid")["route_type"].unique().value_counts()
```

Out[14]:
```
[Carting]    8908
[FTL]        5909
Name: route_type, dtype: int64
```

In [15]:
```
routeType_plot= (df.groupby("trip_uuid")["route_type"].unique().reset_index()['
```

In [16]:
```
sns.barplot(x= routeType_plot.index,
            y = routeType_plot)
```

Out[16]:
```
<Axes: ylabel='route_type'>
```



Observation

- Out of `14,817` unique trips, we have:
  - `8,908 (60%)` of the trip routes categorized as `Carting`, involving smaller vehicles, and
  - `5,909 (40%)` of the total trip routes designated as `FTL` (Full Truck Load), which reach their destinations quicker due to the absence of additional pickups or drop-offs en route.

Feature Engineering

In [17]:
```
df[df["trip_uuid"]=="trip-153741093647649320"]
```

Out[17]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sourc
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

10 rows × 27 columns

Observation

In the outlined trip record above, a single trip is divided among various drop-off locations.

We notice that the trip includes stops at specified source and destination centers (warehouses). The terms '**od-end-time**' and '**od-start-time**' represent the respective ending and starting times of this particular trip.

The '**start-scan-to-end-scan**' duration refers to the time span during which trips are scanned at the beginning and the end. The time provided for '**start-scan-to-end-scan**' is cumulative, not detailed per trip segment.

The indicator '**trip cut off False**' documents the instances when a trip transitions from one warehouse to another along the source to destination route.

The '**Actual-time**' provided denotes the cumulative time taken to complete the entire delivery from the source to the destination.

'**OSRM-time**' stems from an open-source routing engine that calculates the shortest route between points on a given map, estimating the required time, while '**osrm distance**' provides the shortest distance, both of which are given cumulatively.

The '**Actual-distance-to-destination**' represents the cumulative distance between warehouses encountered during the trip. Whenever '**trip cut off**' is False, the distance count recommences from the start.

'**Segment actual time**' reflects the real time taken between two stops during the trip, specified for each segment (taken amid a subset of package delivery).

Lastly, '**segment osrm time**' refers to the OSRM estimated time for each segment, calculated between subsets of package deliveries.

In [18]:
```
df["source_city"] = df["source_name"].str.split(" ",n=1,expand=True)[0].str.sp
df["source_state"] = df["source_name"].str.split(" ",n=1,expand=True)[1].str.re

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

In [19]:
```
df["source_place"] = df["source_name"].str.split("_",n=2,expand=True)[1]
df["destination_place"] = df["destination_name"].str.split("_",n=2,expand=True

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 [20]:
```
df.head()
```

Out[20]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

5 rows × 35 columns

In [21]:
```
df["time_taken_btwn_odstart_and_od_end"] = ((df["od_end_time"]-df["od_start_tir
```

Converting given time duration features into hours

start_scan_to_end_scan
actual_time
osrm_time
segment_actual_time
segment_osrm_time

In [22]:
```
df["start_scan_to_end_scan"] = df["start_scan_to_end_scan"]/60
df["actual_time"] = df["actual_time"]/60
df["osrm_time"] = df["osrm_time"]/60
df["segment_actual_time"] = df["segment_actual_time"]/60
df["segment_osrm_time"] = df["segment_osrm_time"]/60
```

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 36 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_place                          142467 non-null  object
 32  destination_place                     142165 non-null  object
 33  source_pincode                        144867 non-null  object
 34  destination_pincode                   144867 non-null  object
 35  time_taken_btwn_odstart_and_od_end    144867 non-null  float64
dtypes: bool(1), datetime64[ns](3), float64(11), int64(2), object(19)
memory usage: 38.8+ MB
```

In [24]:
```
df.isna().sum()
```

Out[24]:
```
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_place                         2400
destination_place                    2702
source_pincode                          0
destination_pincode                     0
time_taken_btwn_odstart_and_od_end      0
dtype: int64
```

Data `cleaning`

In [25]:
```
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",
                                                    })

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 [26]:
```
df["destination_city"].replace({
                                      "del":"Delhi"
                                  },inplace=True)

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

In [27]:
```
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 [28]:
```
df["source_city_state"] = df["source_city"] + " " + df["source_state"]
df["destination_city_state"] = df["destination_city"] + " " + df["destination_s
```

In [29]:
```
df["source_city_state"].nunique()
```

Out[29]:
```
1249
```

In [30]:
```
df["destination_city_state"].nunique()
```

Out[30]:
```
1242
```

In [31]:
```
df["source_state"].nunique()
```

Out[31]:
```
33
```

In [32]:
```
df["destination_state"].nunique()
```

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

In [33]:
```
data = df.copy()
data.columns
```

Out[33]:
```
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_state',
       'source_place', 'destination_place', 'source_pincode',
       'destination_pincode', 'time_taken_btwn_odstart_and_od_end',
       'source_city_state', 'destination_city_state'],
      dtype='object')
```

In [34]:
```
data.drop(['source_center',"source_name","destination_center","destination_name
```

In [35]:
```
data.drop(["od_end_time","od_start_time"],axis = 1 , inplace=True)
```

Aggregation of Data

In [36]:
```
actual_time = data.groupby(["trip_uuid",
             "start_scan_to_end_scan"])["actual_time"].max().reset_index().gro

segment_osrm_time = data[["trip_uuid","segment_osrm_time"]].groupby("trip_uuid"

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

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

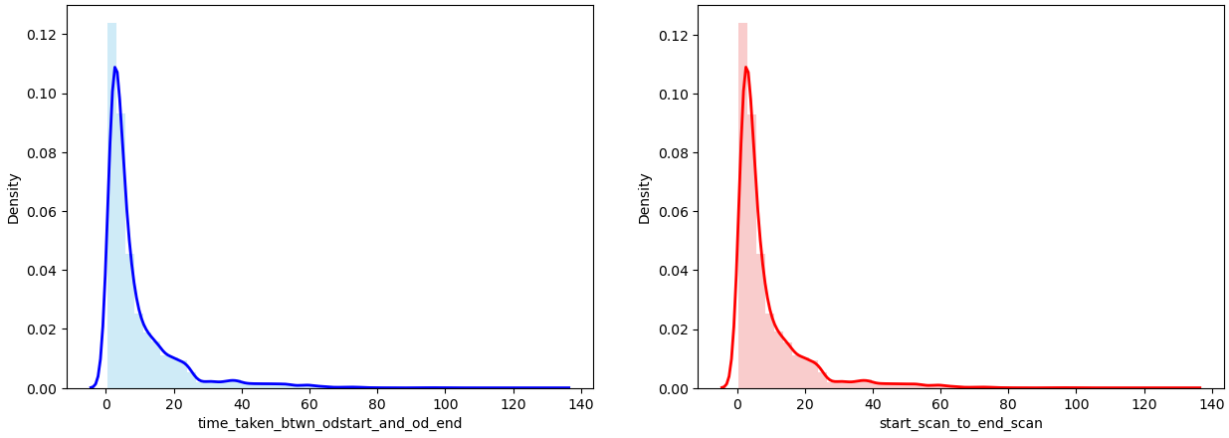
```
time_taken_btwn_odstart_and_od_end = data.groupby("trip_uuid")["time_taken_btwn

time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od_end"] = time

start_scan_to_end_scan = ((data.groupby("trip_uuid")["start_scan_to_end_scan"]

start_scan_to_end_scan["start_scan_to_end_scan"] = start_scan_to_end_scan["star

osrm_distance = data.groupby(["trip_uuid",
              "start_scan_to_end_scan"])["osrm_distance"].max().reset_index().

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

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

Hypothesis Tests for `time durations` and `distance related features`

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 [37]:
```
plt.figure(figsize=(15,5))
plt.subplot(121)
sns.distplot(time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_od

plt.subplot(122)
sns.distplot(start_scan_to_end_scan["start_scan_to_end_scan"], color='lightcora

plt.show()
```



In [38]:
```
stats.ks_2samp(time_taken_btwn_odstart_and_od_end["time_taken_btwn_odstart_and_
               ,start_scan_to_end_scan["start_scan_to_end_scan"])
```

Out[38]:
```
KstestResult(statistic=0.004184382803536474, pvalue=0.9994337058695081, statis
tic_location=2.466666666666667, statistic_sign=-1)
```

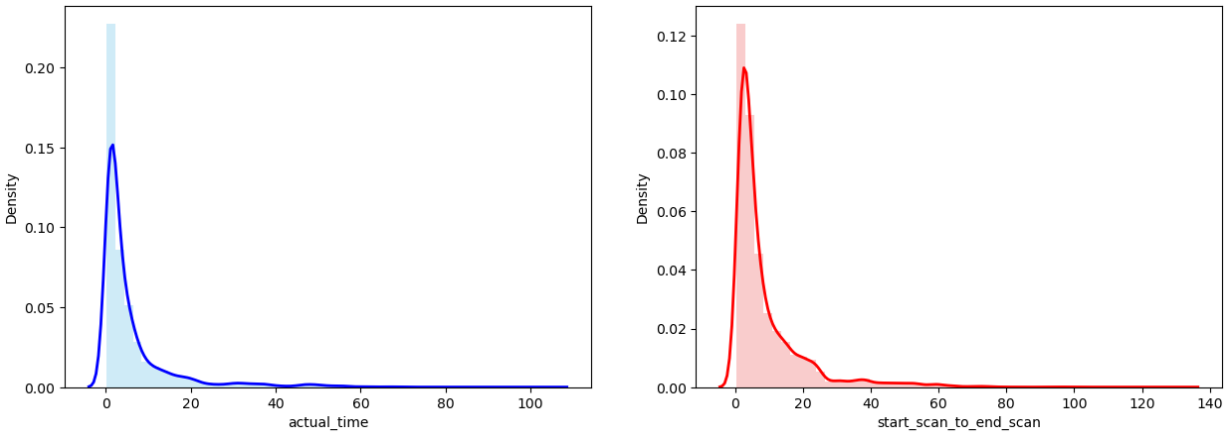
In [39]:
```
for i in range(5):
    print(stats.ttest_ind((time_taken_btwn_odstart_and_od_end["time_taken_btwn_
                ,(start_scan_to_end_scan["start_scan_to_end_scan"].sample(3000)
```

```
Ttest_indResult(statistic=0.6326948623666184, pvalue=0.5269570441495796)
Ttest_indResult(statistic=-0.15628740861386078, pvalue=0.8758117405992905)
Ttest_indResult(statistic=1.3042366951411266, pvalue=0.1922029376420475)
Ttest_indResult(statistic=1.3966728466442648, pvalue=0.1625636489588813)
Ttest_indResult(statistic=0.030600840779704024, pvalue=0.975588889744748)
```

Hypothesis Tests for `actual_time` and `start_scan_to_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 [40]:
```
plt.figure(figsize=(15,5))
plt.subplot(121)
sns.distplot(actual_time["actual_time"], color='skyblue', kde_kws={"color": "bl

plt.subplot(122)
sns.distplot(start_scan_to_end_scan["start_scan_to_end_scan"], color='lightcora


plt.show()
```



Merging All the numerical Fields as per `TripID`

```
actual_distance_to_destination
osrm_distance
segment_osrm_distance
time_taken_btwn_odstart_and_od_end
start_scan_to_end_scan
actual_time
segment_actual_time
```

```
osrm_time
segment_osrm_time
```

In [41]:
```
distances = segment_osrm_distance.merge(actual_distance_to_destination.merge(os
                                                                             or
                                                                             or

time = segment_osrm_time.merge(osrm_time.merge(segment_actual_time.merge(actual
                                        on="trip_uuid",
                                        ),on="trip_uuid"),on="trip_uuid"),on='

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

In [42]:
```
Merge1
```

Out[42]:

	trip_uuid	segment_osrm_time	osrm_time	segment_actual_time	actual_tim
0	trip-153671041653548748	16.800000	12.383333	25.800000	26.03333
1	trip-153671042288605164	1.083333	1.133333	2.350000	2.38333
2	trip-153671043369099517	32.350000	29.016667	55.133333	55.78333
3	trip-153671046011330457	0.266667	0.250000	0.983333	0.98333
4	trip-153671052974046625	1.916667	1.950000	5.666667	5.68333
...	...	...	...	...	.
14812	trip-153861095625827784	1.033333	1.033333	1.366667	1.38333
14813	trip-153861104386292051	0.183333	0.200000	0.350000	0.35000
14814	trip-153861106442901555	1.466667	0.900000	4.683333	4.70000
14815	trip-153861115439069069	3.683333	3.066667	4.300000	4.40000
14816	trip-153861118270144424	1.116667	1.133333	4.566667	4.58333

14817 rows × 10 columns

Merging `Location details` and `route_type` and Numerical data on `TripID`

In [43]:
```
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 [44]:
```
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 [45]:
```
trip_records = Merged.copy()
trip_records["route_type"] = trip_records["route_type"].apply(lambda x:x[0])
route_to_merge = data.groupby("trip_uuid")["route_schedule_uuid"].unique().rese
trip_records = trip_records.merge(route_to_merge,on="trip_uuid",how="outer")
trip_records["route_schedule_uuid"] = trip_records["route_schedule_uuid"].apply
```

In [46]:
```
trip_records
```

Out[46]:

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

14817 rows × 18 columns

In [47]:
```
trip_records.isna().sum()
```

Out[47]:
```
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 [48]:
```
trip_records["source_city"] = trip_records["source_city"].astype("str").str.sti
trip_records["destination_city"] = trip_records["destination_city"].astype("sti
trip_records["source_city_state"] = trip_records["source_city_state"].astype("s
trip_records["destination_city_state"] = trip_records["destination_city_state"]

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

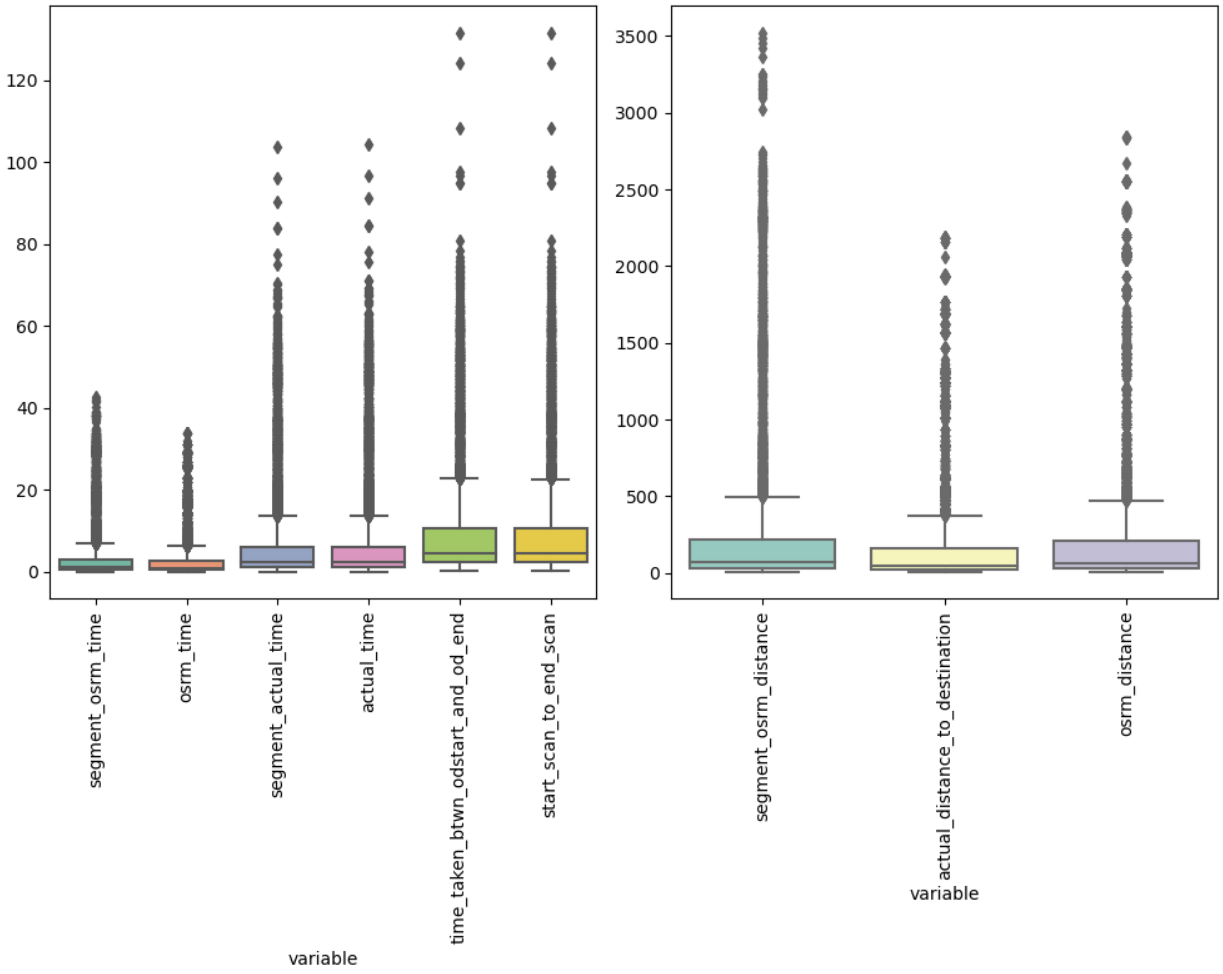
In [49]:
```
trip_records.isna().sum()
```

Out[49]:
```
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
```

Finding `Outliers`

In [50]:
```
plt.figure(figsize=(10, 8))

# Melt your data frames to long-form
long_form_1 = trip_records[['segment_osrm_time', 'osrm_time', 'segment_actual_t
long_form_2 = trip_records[['segment_osrm_distance', 'actual_distance_to_destin
```

```
# Plotting boxplot for the first set of columns
plt.subplot(121)
sns.boxplot(x='variable', y='value', data=long_form_1, palette='Set2')
plt.xticks(rotation=90)
plt.ylabel('')

# Plotting boxplot for the second set of columns
plt.subplot(122)
sns.boxplot(x='variable', y='value', data=long_form_2, palette='Set3')
plt.xticks(rotation=90)
plt.ylabel('')

plt.tight_layout()
plt.show()
```



In [51]:
```
outlier_treatment  = trip_records.copy()

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

Removing all the outliers from all numerical features

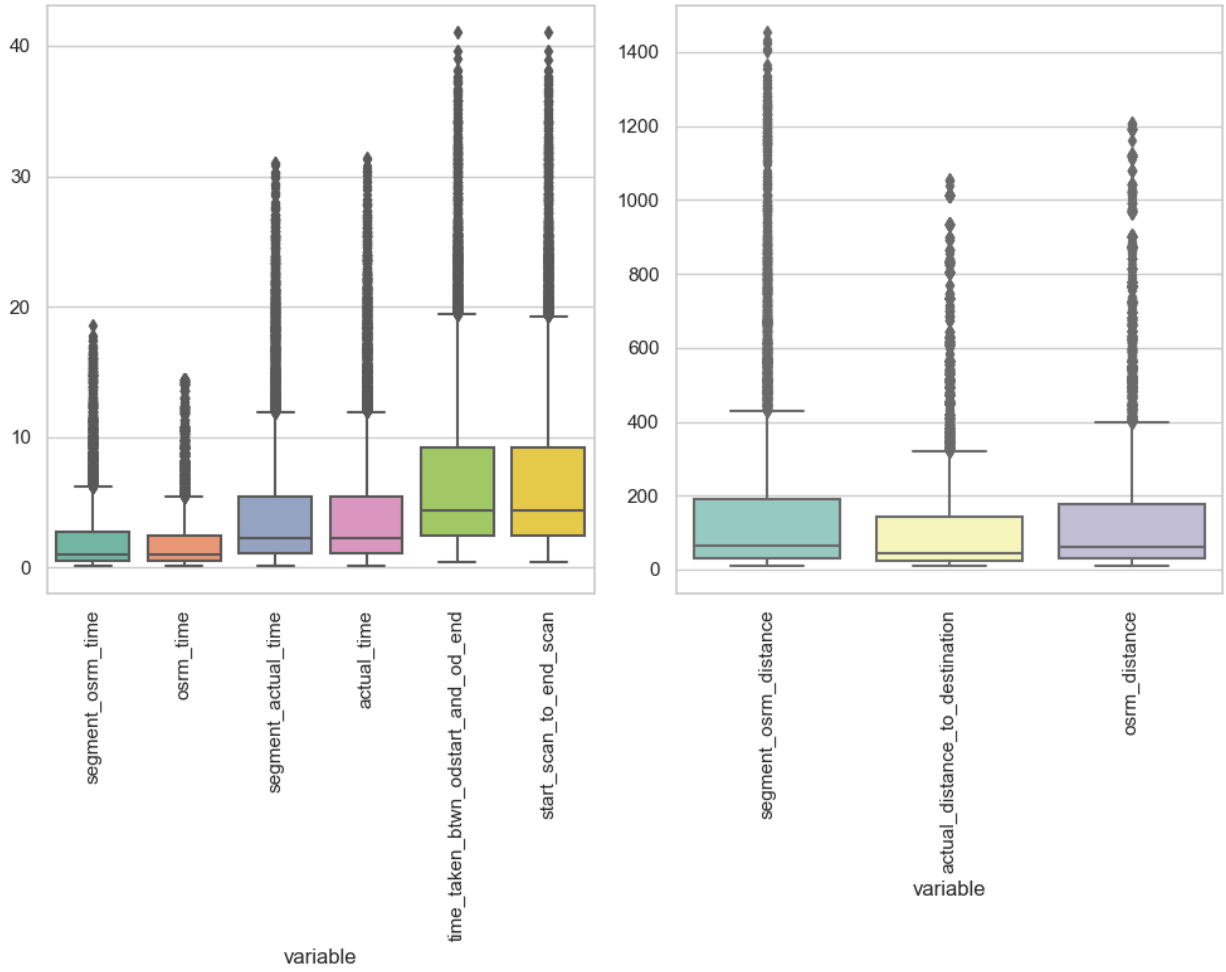
In [52]:
```
trip_records_without_outliers = trip_records.loc[outlier_treatment_num[(np.abs(
trip_records_without_outliers.head()
```

Out[52]:

	trip_uuid	route_type	source_city	destination_city	source_city_state	destinatio
0	trip-153671041653548748	FTL	Bhopal Kanpur	Kanpur Gurgaon	Bhopal Madhya Pradesh Kanpur Uttar Pradesh	Kanpur U Gurg
1	trip-153671042288605164	Carting	Tumkur Doddablpur	Doddablpur Chikblapur	Tumkur Karnataka Doddablpur Karnataka	Doddablp Chikblap
3	trip-153671046011330457	Carting	Mumbai	Mumbai	Mumbai Hub Maharashtra	Mumbai
4	trip-153671052974046625	FTL	Bellary Hospet Sandur	Hospet Sandur Bellary	Bellary Karnataka Hospet Karnataka Sandur Karn...	Hosp Sanc E
5	trip-153671055416136166	Carting	Chennai	Chennai	Chennai Tamil Nadu	Chenn

In [53]:
```
trip_records_without_outliers = trip_records_without_outliers[['trip_uuid','rou
       '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 [54]:
```
sns.set(style="whitegrid")

plt.figure(figsize=(10,8))

# Melt DataFrames to long-form
long_form_1 = trip_records_without_outliers[['segment_osrm_time', 'osrm_time',
long_form_2 = trip_records_without_outliers[['segment_osrm_distance', 'actual_d

# Plotting boxplot for the first set of columns
plt.subplot(121)
sns.boxplot(x='variable', y='value', data=long_form_1, palette='Set2')
plt.xticks(rotation=90)
plt.ylabel('')

# Plotting boxplot for the second set of columns
plt.subplot(122)
sns.boxplot(x='variable', y='value', data=long_form_2, palette='Set3')
plt.xticks(rotation=90)
plt.ylabel('')

plt.tight_layout()  # Adjusts the spacing
plt.show()
```



Processing Data with `One Hot Encoding`

In [55]:
```
trip_records_without_outliers["destination_source_locations"] = trip_records_wi
trip_records_without_outliers.drop(["source_city_state","destination_city_state
```

In [56]:
```
sc_dc = trip_records_without_outliers.groupby(["destination_source_locations"])
```

In [57]:
```
def get_cat(H):
    if 0 <= H <= 50:
        return "Category 7"
    elif 51 <= H <= 100:
        return "Category 6"
    elif 101 <= H <= 200:
        return "Category 5"
    elif 201 <= H <= 300:
        return "Category 4"
    elif 301 <= H <= 400:
        return "Category 3"
    elif 401 <= H <= 500:
        return "Category 2"
    else:
        return "Category 1"
```

In [58]:
```
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"],a

trip_records_for_encoding.drop(["trip_uuid_y"],axis = 1,inplace=True)

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

In [59]:
```
encoded_data.head()
```

Out[59]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odstar
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	

In [60]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

In [61]:
```
scaler = StandardScaler()
std_data = scaler.fit_transform(encoded_data[['segment_osrm_time',
                                              'osrm_time',
                                              'segment_actual_time',
                                              'actual_time',
                                              'time_taken_btwn_odstart_and_c
                                              'start_scan_to_end_scan',
                                              'segment_osrm_distance',
                                              'actual_distance_to_destinatic
                                              '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_en
                                           'start_scan_to_end_scan',
                                           'segment_osrm_distance',
                                           'actual_distance_to_destination',
                                           'osrm_distance'])

std_data.head()
```

Out[61]:

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

In [62]:
```
scaler = MinMaxScaler()

MinMax_data = scaler.fit_transform(encoded_data[['segment_osrm_time','osrm_time
 'time_taken_btwn_odstart_and_od_end','start_scan_to_end_scan','segment_osrm_di
 'osrm_distance']])

MinMax_data = pd.DataFrame(MinMax_data,columns=['segment_osrm_time',
 'osrm_time','segment_actual_time','actual_time','time_taken_btwn_odstart_and_
 'segment_osrm_distance','actual_distance_to_destination','osrm_distance'])

MinMax_data.head()
```

Out[62]:

	segment_osrm_time	osrm_time	segment_actual_time	actual_time	time_taken_btwn_odstar
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 [63]:
```
one_hot_encoded_data = encoded_data[["route_type_Carting","route_type_FTL","cit
                                     "city_Category 2","city_Category 3","city_
                                     "city_Category 5","city_Category 6","city_
```

Route Analysis

In [64]:
```
A = data.groupby("route_schedule_uuid")["route_type"].unique().reset_index()

B = data.groupby("route_schedule_uuid")["destination_city"].unique().reset_inde
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_inc
E.columns = ["route_schedule_uuid","destination_states"]

F = data.groupby("route_schedule_uuid")[["source_state",
```

```
                                          "destination_state"]].nunique().sort_v

F.columns = ["route_schedule_uuid","#source_states"
             ,"#destination_states"]

G = trip_records.groupby("route_schedule_uuid")["actual_distance_to_destinatior
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 [65]:
```
I = data.groupby("route_schedule_uuid")[["source_city",
                                         "destination_city"]].nunique().sort_va

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

In [66]:
```
route_records = I.merge(H.merge(G.merge(F.merge(E.merge(D.merge(C.merge(A.merge
        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 [67]:
```
route_records.isna().sum()
```

Out[67]:
```
route_schedule_uuid                       0
#source_cities                            0
#destination_cities                       0
Number_of_Trips                           0
Average_Actual_distance_to_destination    0
#source_states                            0
#destination_states                       0
destination_states                        0
source_states                             0
source_cities                             0
route_type                                0
destination_cities                        0
dtype: int64
```

In [68]:
```
route_records["route_type"] = route_records["route_type"].astype("str").str.sti
route_records["source_cities"] = route_records["source_cities"].astype("str").s
route_records["destination_cities"] = route_records["destination_cities"].astyp
route_records["source_states"] = route_records["source_states"].astype("str").s
route_records["destination_states"] = route_records["destination_states"].astyp
```

In [69]:
```
route_records.head()
```

Out[69]:

	route_schedule_uuid	#source_cities	#destination_cities	Number_of_Trips	Average_Actu
0	thanos::sroute:d010efca-d90d-4977-b987-eae68c5...	13	11	14	
1	thanos::sroute:4cbecb35-356b-4b68-bf3c-6225b5e...	10	10	12	
2	thanos::sroute:ae5c430f-6153-48d1-8fe5-d5f0bbc...	10	10	20	
3	thanos::sroute:f8968c72-5222-4d81-9eed-8a6d88f...	9	9	9	
4	thanos::sroute:ed5b80be-7abf-424d-b8cd-d81556a...	9	8	20	

In [70]:
```
route_records["ROUTE"] = route_records["source_cities"] + " -- " + route_record
route_records.drop(["route_schedule_uuid"],axis = 1,inplace=True)
first_column = route_records.pop('ROUTE')
route_records.insert(0, 'ROUTE', first_column)
route_records["SouceToDestination_city"] = route_records["source_cities"].str.s
first_column = route_records.pop('SouceToDestination_city')
route_records.insert(0, 'SouceToDestination_city', first_column)
```

In [71]:
```
route_records.head()
```

Out[71]:

	SouceToDestination_city	ROUTE	#source_cities	#destination_cities	Number_of_Trips	A
0	Guwahati TO LakhimpurN	Guwahati LakhimpurN Dhemaji Likabali Tezpur Pa...	13	11	14	
1	Guwahati TO Tura	Guwahati Rangia Kokrajhar Dhubri Bilasipara Tu...	10	10	12	
2	Jaipur TO Tarnau	Jaipur Chomu Reengus Sikar Bikaner Didwana Suj...	10	10	20	
3	Mangalore TO Udupi	Mangalore Udupi Kundapura Bhatkal Honnavar Kum...	9	9	9	
4	Ajmer TO Raipur	Ajmer Beawar Bilara Bijainagar Kekri Nasirabad...	9	8	20	

In [72]:
```
route_records.to_csv("route_records.csv")
```

In [73]:
```
Number_of_trips_between_cities = data.groupby(["source_city_state",
                                               "destination_city_state"])["trip
Number_of_trips_between_cities.head(25)
```

Out[73]:

	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

In [74]:
```
Number_of_trips_between_cities.loc[Number_of_trips_between_cities["source_city_
```

Out[74]:

	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

Observations

Upon examining the data, it is noted that there are discrepancies in the number of source and destination states. The pairs of source and destination cities with the highest number of trips between them include:

- Delhi to Gurgaon
- Gurgaon, Haryana to Bengaluru, Karnataka
- Bhiwandi/Mumbai, Maharashtra to Pune, Maharashtra
- Sonipat to Gurgaon, Haryana

Additionally, a significant number of deliveries are directed towards airports, such as:

- Chennai to MAA Chennai International Airport
- Pune to Pune Airport (PNQ)
- Kolkata to CCU West Bengal Kolkata International Airport
- Bengaluru to BLR-Bengaluru International Airport

In [75]:
```
route_records[["ROUTE","Number_of_Trips",
               "Average_Actual_distance_to_destination",
               "#source_cities",
               "#destination_cities"]].sort_values(by="Number_of_Trips",ascend
```

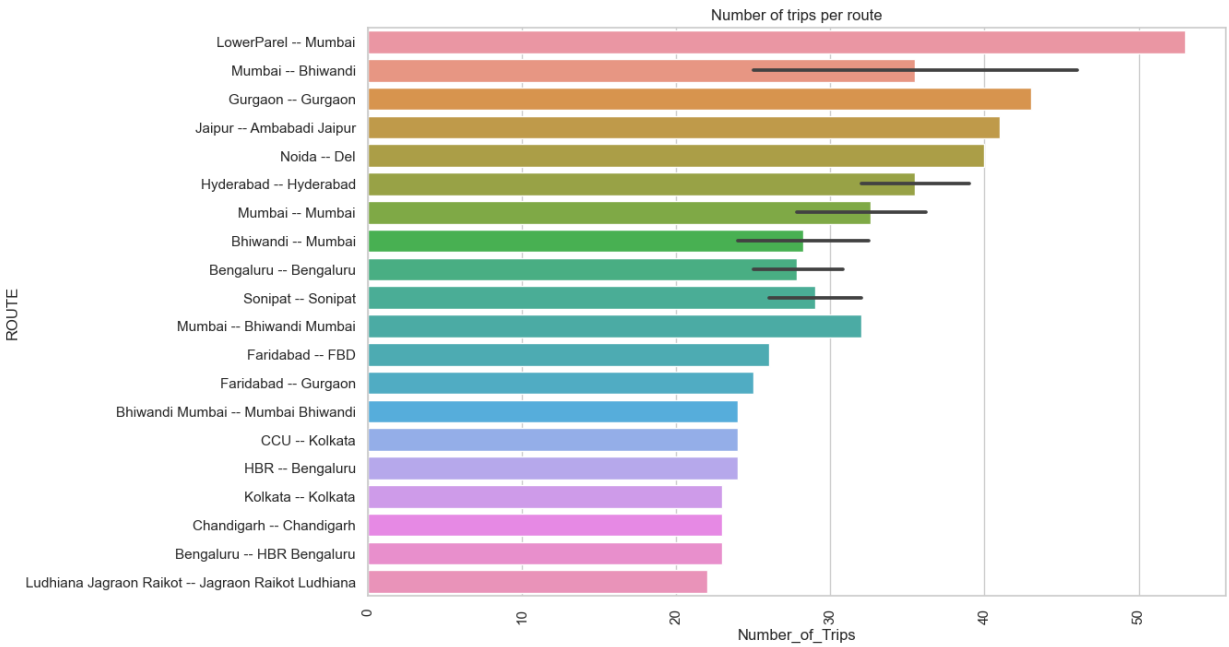
Out[75]:

	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities	#de
1465	LowerParel -- Mumbai	53	16.428868	1	
1426	Mumbai -- Bhiwandi	46	20.199445	1	
808	Gurgaon -- Gurgaon	43	29.740842	1	
679	Jaipur -- Ambabadi Jaipur	41	15.348495	1	
1257	Noida -- Del	40	10.882902	1	
1368	Hyderabad -- Hyderabad	39	35.695641	1	
1273	Mumbai -- Mumbai	37	13.882863	1	
1359	Mumbai -- Mumbai	36	17.526251	1	
1303	Bhiwandi -- Mumbai	35	21.241534	1	
700	Mumbai -- Mumbai	34	15.906614	1	
751	Mumbai -- Mumbai	33	15.668726	1	
1060	Bengaluru -- Bengaluru	33	28.067004	1	
793	Sonipat -- Sonipat	32	11.691243	1	
972	Hyderabad -- Hyderabad	32	21.835579	1	
1184	Mumbai -- Bhiwandi Mumbai	32	21.601109	1	
874	Bengaluru -- Bengaluru	30	28.055789	1	
1177	Bhiwandi -- Mumbai	30	21.396002	1	
1354	Bengaluru -- Bengaluru	27	27.967087	1	
921	Faridabad -- FBD	26	9.677121	1	
1480	Sonipat -- Sonipat	26	12.182486	1	

	ROUTE	Number_of_Trips	Average_Actual_distance_to_destination	#source_cities	#de
1041	Mumbai -- Bhiwandi	25	19.942191	1	
877	Faridabad -- Gurgaon	25	47.091622	1	
833	Bhiwandi -- Mumbai	25	21.531705	1	
1249	Bengaluru -- Bengaluru	25	28.019668	1	
869	Bengaluru -- Bengaluru	24	41.396497	1	

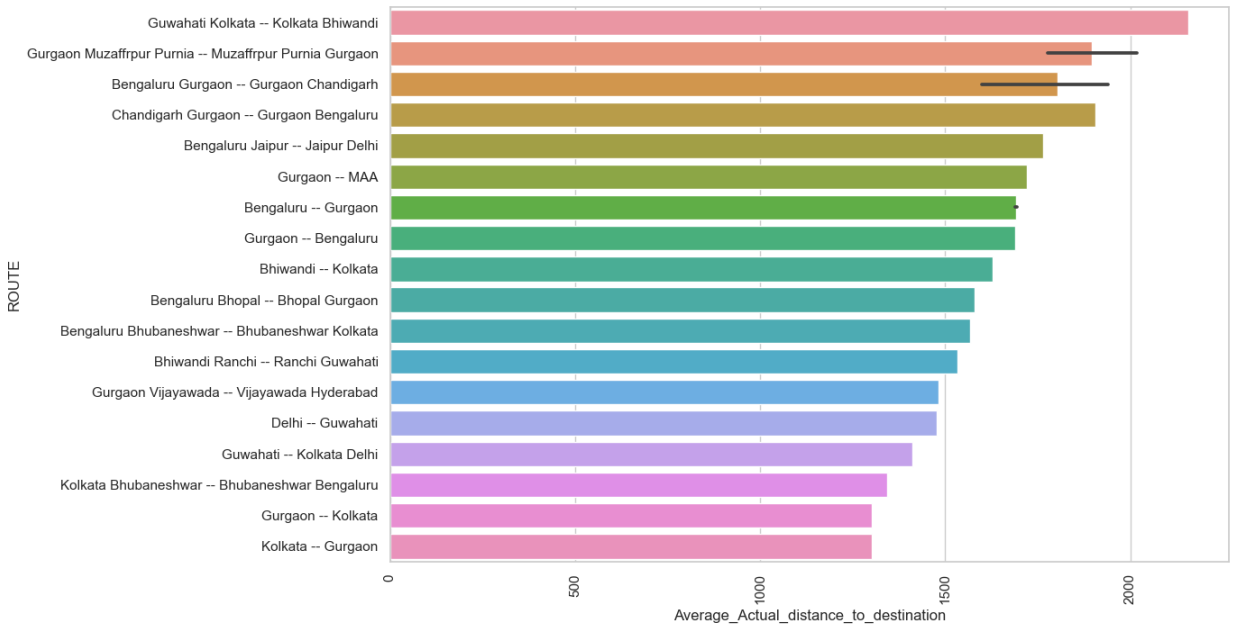
Top Routes having Maximum Number of Trips between/within the source and destinations

In [76]:
```
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 [77]:
```
plt.figure(figsize=(12,8))

X = route_records[["ROUTE", "Average_Actual_distance_to_destination",
                   ]].sort_values(by="Average_Actual_distance_to_destination",asce
sns.barplot(y = X["ROUTE"],
            x = X["Average_Actual_distance_to_destination"])
```

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



Observations

- Based on the above bar chart and table, it's evident that the highest number of trips occurs within specific cities.
- Regarding the average distance between destinations, the longest routes are observed from Guwahati to Mumbai, Bengaluru to Chandigarh, Bengaluru to Delhi, and Bengaluru to Gurgaon.

Finding the `busiest` and `longes`t routes

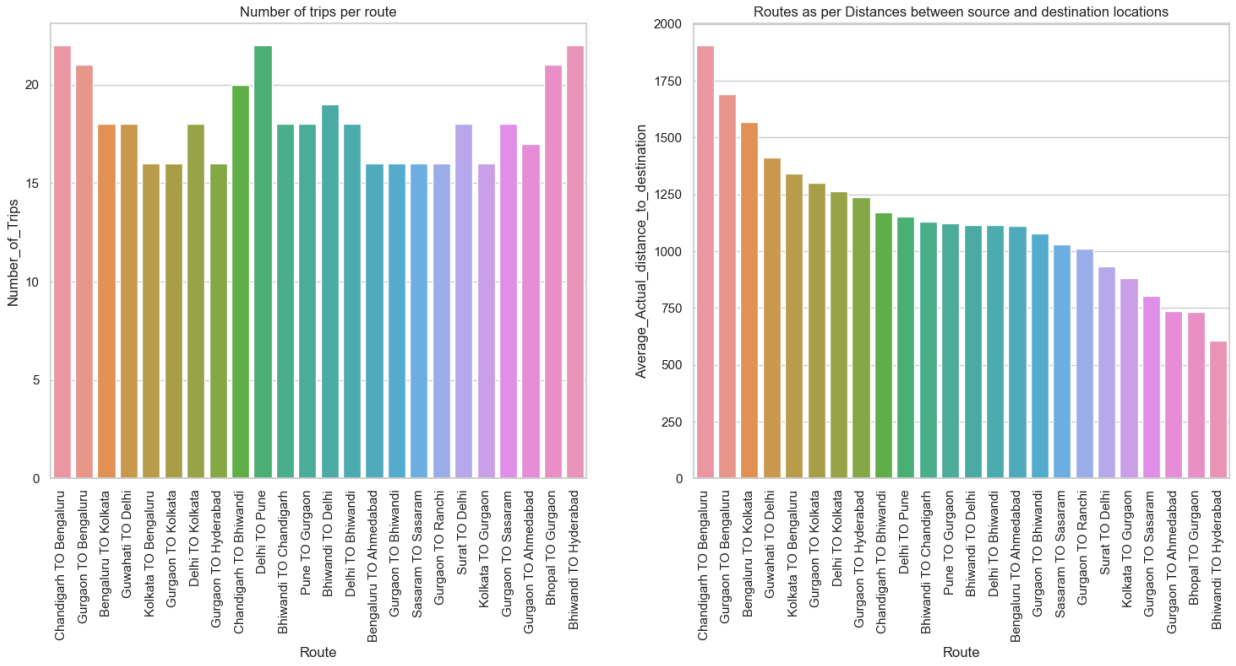
In [78]:
```
Busiest_and_Longest_Routes  = route_records[(route_records["Average_Actual_dist
             & (route_records["Number_of_Trips"] > route_records["Number_of_Tr


Busiest_and_Longest_Routes_top25 = Busiest_and_Longest_Routes[["source_cities",
                                                              "destination_cit
                                                              "Number_of_Trips
                                                              "Average_Actual_

Busiest_and_Longest_Routes_top25.head()
```

Out[78]:

	source_cities	destination_cities	Number_of_Trips	Average_Actual_distance_to_destinatior
629	Chandigarh Gurgaon	Gurgaon Bengaluru	22	1905.76605
995	Gurgaon	Bengaluru	21	1689.873158
991	Gurgaon	Bengaluru	21	1689.791894
512	Bengaluru Bhubaneshwar	Bhubaneshwar Kolkata	18	1567.577507
745	Guwahati	Kolkata Delhi	18	1411.208424

In [79]:
```
Busiest_and_Longest_Routes_top25["Route"] = Busiest_and_Longest_Routes_top25["s
Busiest_and_Longest_Routes_top25.drop(["source_cities","destination_cities"],ax
```

In [80]:
```
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_dest
plt.xticks(rotation = 90)
plt.show()
```



Routes which are passing through maxinum number of cities

In [81]:
```
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[81]:

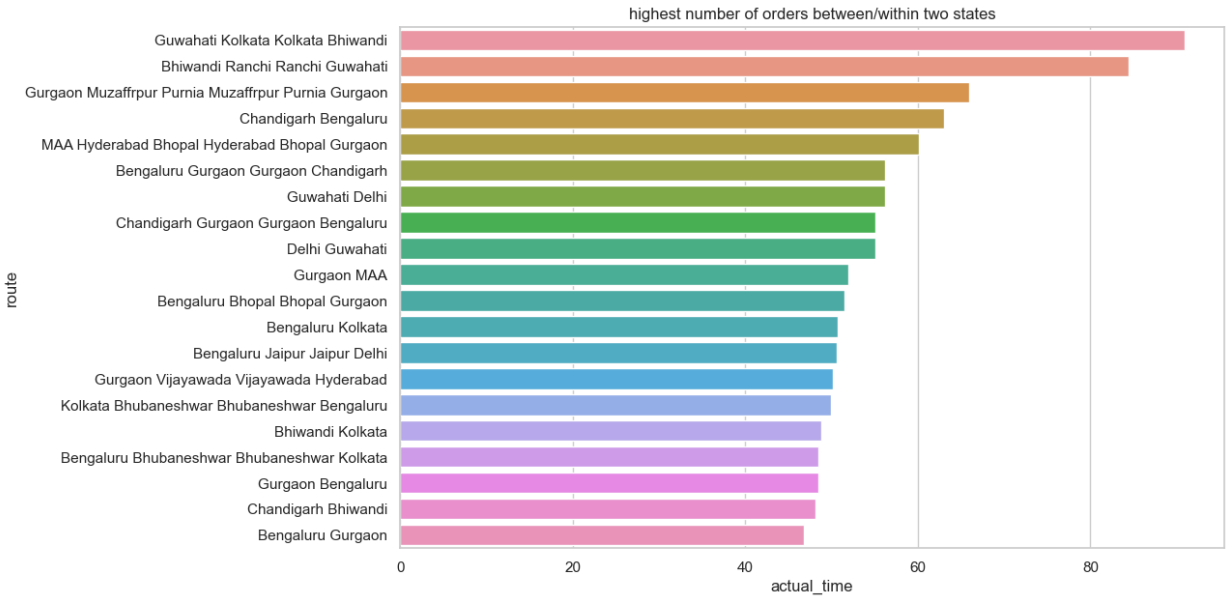
	SouceToDestination_city	Number_of_Trips	Average_Actual_distance_to_destination	#source
0	Guwahati TO LakhimpurN	14	281.596486	
2	Jaipur TO Tarnau	20	351.611796	
1	Guwahati TO Tura	12	332.602225	
3	Mangalore TO Udupi	9	195.257193	
4	Ajmer TO Raipur	20	178.737233	
5	Mainpuri TO Tilhar	12	207.247057	
8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

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

In [82]:

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

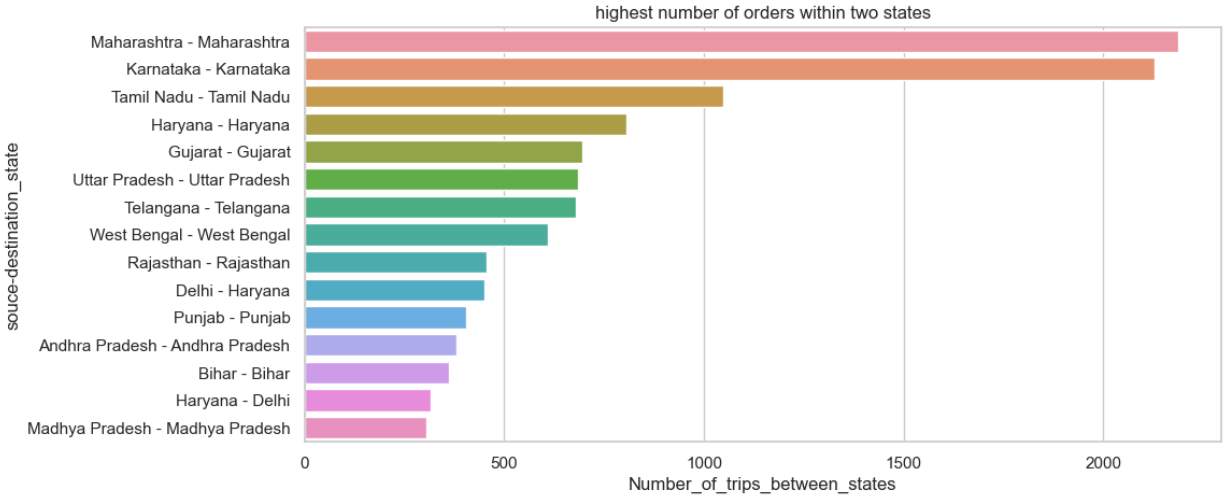
```
Longest_route_as_per_actual_trip_time["route"] = Longest_route_as_per_actual_tr
Longest_route_as_per_actual_trip_time.drop(["source_city",
                                            "destination_city"],axis = 1,inplac
Longest_route_as_per_actual_trip_time
plt.figure(figsize=(11,7))
sns.barplot(y = Longest_route_as_per_actual_trip_time["route"],
            x = Longest_route_as_per_actual_trip_time["actual_time"],)
plt.title("highest number of orders between/within two states")
plt.show()
```



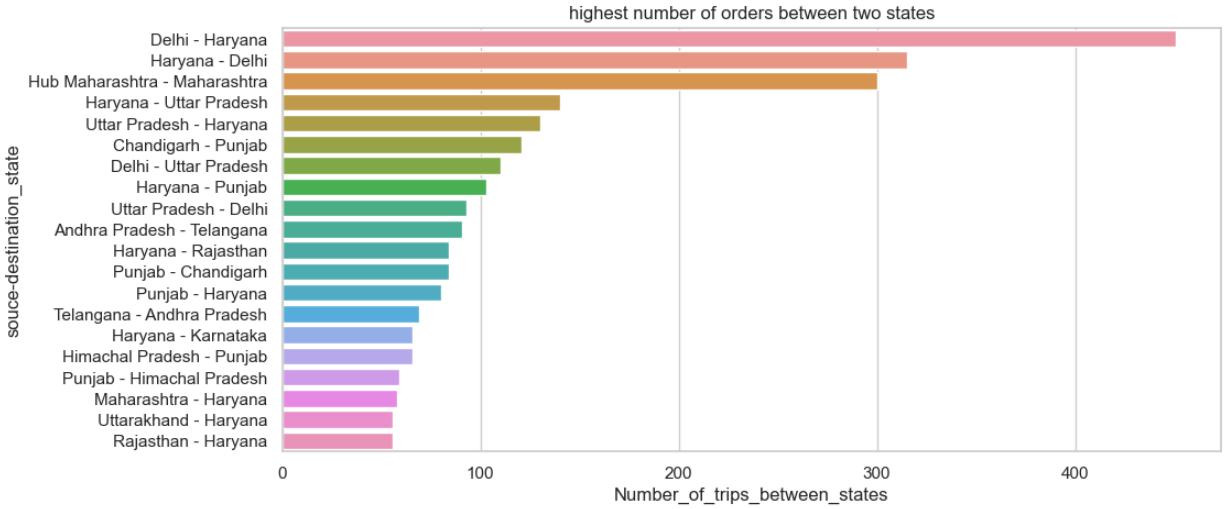
Highest `number of trips` happening between/within two states

In [83]:

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

HOBS  = highest_order_between_states.head(15)
HOBS["souce-destination"] = HOBS["source_state"] + " - " + HOBS["destination_st
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 [84]:
```
HOBS = data.groupby(["source_state","destination_state"])["trip_uuid"].nunique(
HOBS = HOBS[HOBS["source_state"]!=HOBS["destination_state"]].head(20)

HOBS["souce-destination"] = HOBS["source_state"] + " - " + HOBS["destination_st
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()
```



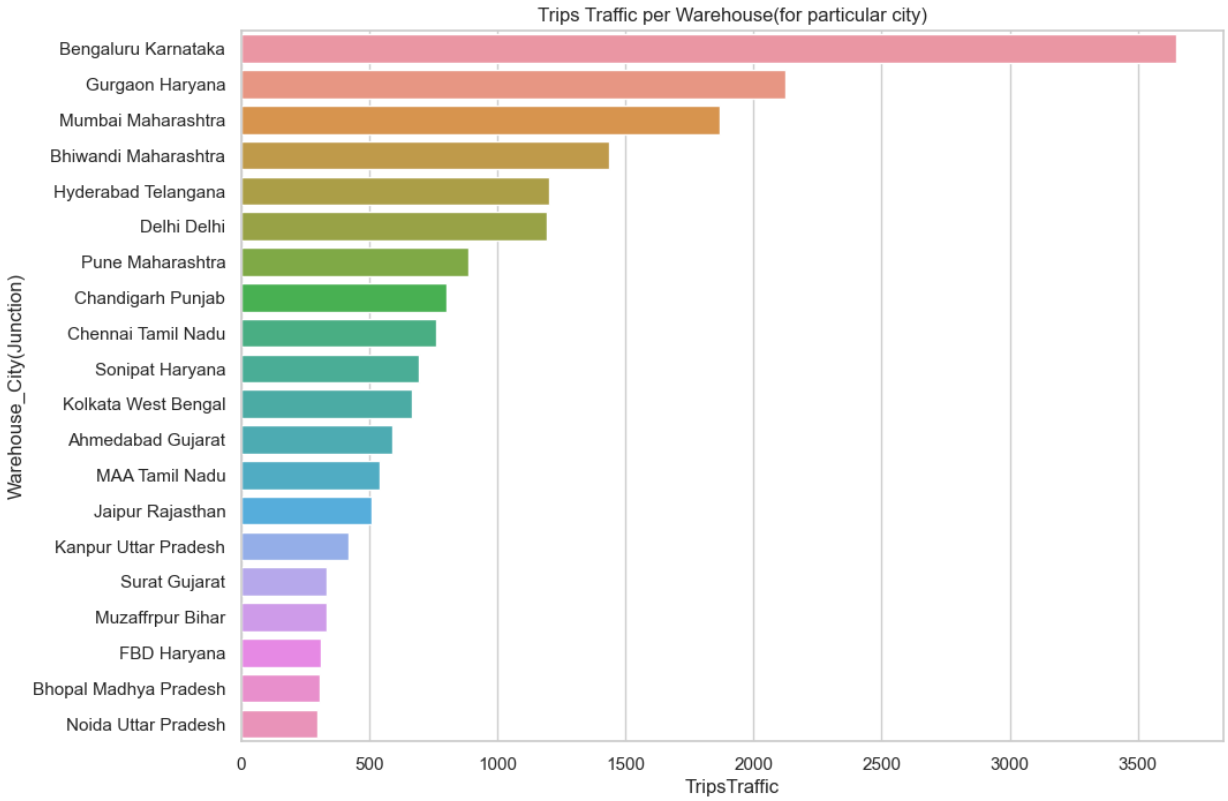
Observations

- Based on the above charts:
  - The route from Delhi to Haryana is the most frequented, with over 400 trips recorded. Other busy routes include Haryana to Uttar Pradesh, Chandigarh to Punjab, and Delhi to Uttar Pradesh.
  - Within certain states such as Maharashtra, Karnataka, and Tamil Nadu, the number of trips exceeds 1000.

Top 10 warehouses with heavy traffic

In [85]:
```
destination_traffic = data.groupby(["destination_city_state"])["trip_uuid"].nu
source_traffic = data.groupby(["source_city_state"])["trip_uuid"].nunique().re
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
transactions["TripsTraffic"] = transactions["#Trips_s"]+transactions["#Trips_d
transactions.drop(["#Trips_s","#Trips_d","destination_city_state"],axis = 1,in
transactions.columns = ["Warehouse_City(Junction)","TripsTraffic"]
```

In [86]:
```
T = transactions.sort_values(by=["TripsTraffic"],ascending=False).head(20)
plt.figure(figsize=(11,8))
sns.barplot(y = T["Warehouse_City(Junction)"],
            x = T["TripsTraffic"])
plt.title("Trips Traffic per Warehouse(for particular city)")
plt.show()
```



In [87]:
```
trip_records.groupby(["source_state","destination_state"])["trip_uuid"].count(
```

Out[87]:

	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

Observations

- The list below showcases the top 20 busiest warehouses (junctions) based on the trip traffic at each junction:
  - 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
  - Muzaffarpur, Bihar
  - FBD, Haryana

- Bhopal, Madhya Pradesh
- Noida, Uttar Pradesh

End of the Report