Problem Statement

Introduction:

Delhivery, India's leading and rapidly growing integrated player, has set its sights on creating the commerce operating system. They achieve this by utilizing world-class infrastructure, ensuring the highest quality in logistics operations, and harnessing cutting-edge engineering and technology capabilities.

Dataset: 'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv? 1642751181'

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.

Why this case study?

From Delhivery’s Perspective:

● Delhivery aims to establish itself as the premier player in the logistics industry. This case study is of paramount importance as it aligns with the company's core objectives and operational excellence. ● It provides a practical framework for understanding and processing data, which is integral to their operations. By leveraging data engineering pipelines and data analysis techniques, Delhivery can achieve several critical goals. ● First, it allows them to ensure data integrity and quality by addressing missing values and structuring the dataset appropriately. ● Second, it enables the extraction of valuable features from raw data, which can be utilized for building accurate forecasting models. ● Moreover, it facilitates the identification of patterns, insights, and actionable recommendations crucial for optimizing their logistics operations. ● By conducting hypothesis testing and outlier detection, Delhivery can refine their processes and further enhance the quality of service they provide.

From Learners' Perspective:

● Learners will gain hands-on experience in data preprocessing and cleaning, which is often the most time-consuming aspect of data analysis. ● Feature engineering is a critical step in building machine learning models. In this case study, learners will understand how to extract meaningful features from raw data, including datetime manipulation and column splitting. ● The case study introduces learners to the concept of grouping data based on specific keys and then aggregating it. This is a key aspect of data analysis, especially when dealing with time-series data or data with a hierarchical structure. ● Learners will perform hypothesis testing, to validate assumptions and draw insights from data. ● The case study goes beyond data analysis by focusing on deriving actionable insights for a business. Learners will understand how data analysis can drive informed decision-making and recommendations.

```
'''
Problem Statement:

Delhivery wants to understand and process the data coming out of the
data engineering pipelines.
The two main tasks involved here are:

1. Cleaning pre-processing and manipulating the data to extract useful
features from the raw data.
2. Making sense out of the raw data to provide business insights and
recommendations and to help the data science team to build the
forecasting models on it.
'''

{"type":"string"}
```

#Initial Analysis

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler


import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.set_option('display.max_columns', None)

data = pd.read_csv('delhivery_data.csv')

data.head()

{"type":"dataframe","variable_name":"data"}

data.shape

(144867, 24)

data.info()

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

```
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  object
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  object
 10  od_end_time                     144867 non-null  object
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool
 13  cutoff_factor                   144867 non-null  int64
 14  cutoff_timestamp                144867 non-null  object
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

```
data.duplicated().sum()
```

```
0
```

```
data.describe()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"start_scan_to_end_scan\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 50669.13363974827,\n   \"min\": 20.0,\n   \"max\": 144867.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   961.2629860492727,\n   449.0,\n   144867.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cutoff_factor\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 51076.25962961759,\n   \"min\": 9.0,\n   \"max\": 144867.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   232.926567127089,\n   66.0,\n   144867.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"actual_distance_to_destination\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 51076.056012537076,\n   \"min\": 9.00004535977208,\n   \"max\": 144867.0,\n   \"num_unique_values\": 8,\n

```
\"samples\": [\n          234.07337161811876,\n
66.12657071764777,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
50924.567951909936,\n        \"min\": 9.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
416.92752662787245,\n          132.0,\n          144867.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51091.78365158944,\n        \"min\": 6.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
213.86827227733025,\n          64.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51047.4793064438,\n        \"min\": 9.0082,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
284.7712965174954,\n          78.5258,\n          144867.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"factor\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 51213.829643415665,\n        \"min\": 0.144,\n
\"max\": 144867.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          2.120107191835053,\n
1.8571428571428568,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51078.74557749331,\n        \"min\": -244.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
36.19611091552942,\n          29.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51135.69935115408,\n        \"min\": 0.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
18.507548302926132,\n          17.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51107.94242654291,\n        \"min\": 0.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
22.829019768477295,\n          23.513,\n          144867.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_factor\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 51190.17322939711,\n        \"min\": -
```

23.444444444444443,\n   \"max\": 144867.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
2.218368368955117,\n   1.6842105263157894,\n   144867.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
'''

1. There are 144866 rows and 24 columns before dropping null values.
2. Time and distance related columns are the main ones and they are
numerical. All of them have quite a big range i.e. max - min so that
means that they contain a lot of outliers.
3. Route type is a categorical column with possible values being
Carting and FTL.
4. There are no duplicated rows.
5. Min actual time is 9 mins.

'''
```

{"type":"string"}

#Basic data cleaning and exploration

##Handling missing values

```
data.isna().sum()
```

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

```
segment_factor                                0
dtype: int64
```

```python
data.isna().sum() * 100/len(data)
```

```
data                              0.000000
trip_creation_time                0.000000
route_schedule_uuid               0.000000
route_type                        0.000000
trip_uuid                         0.000000
source_center                     0.000000
source_name                       0.202254
destination_center                0.000000
destination_name                  0.180165
od_start_time                     0.000000
od_end_time                       0.000000
start_scan_to_end_scan            0.000000
is_cutoff                         0.000000
cutoff_factor                     0.000000
cutoff_timestamp                  0.000000
actual_distance_to_destination    0.000000
actual_time                       0.000000
osrm_time                         0.000000
osrm_distance                     0.000000
factor                            0.000000
segment_actual_time               0.000000
segment_osrm_time                 0.000000
segment_osrm_distance             0.000000
segment_factor                    0.000000
dtype: float64
```

```python
data[(data['source_name'].isna()) & (data['destination_name'].isna())]
```

{"type":"dataframe"}

```python
(data.isna().sum().sum() - 3)*100/len(data)
```

```
0.380348871723719
```

```python
'''
There are missing values present in the columns:
source_name has 293 missing values
destination_name has 261 missing values

So total 293 + 261 - 3 = 551 rows (we subtracted 3 as for those rows
both the columns have missing values)
551 rows comprise of just 0.38% of the total dataset rows.

So, we will remove these 551 rows.
'''
```

```
{"type":"string"}
```

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

```
data.isna().sum()
```

```
data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                       0
destination_center                0
destination_name                  0
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
```

##Analyze the structure of the data

```
data.shape
```

```
(144316, 24)
```

```
'''
There are 144316 rows and 24 columns.
We can see the detailed description of the data using the information
given below:
'''
```

```
{"type":"string"}
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 144316 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144316 non-null  object 
 1   trip_creation_time              144316 non-null  object 
 2   route_schedule_uuid             144316 non-null  object 
 3   route_type                      144316 non-null  object 
 4   trip_uuid                       144316 non-null  object 
 5   source_center                   144316 non-null  object 
 6   source_name                     144316 non-null  object 
 7   destination_center              144316 non-null  object 
 8   destination_name                144316 non-null  object 
 9   od_start_time                   144316 non-null  object 
 10  od_end_time                     144316 non-null  object 
 11  start_scan_to_end_scan          144316 non-null  float64
 12  is_cutoff                       144316 non-null  bool   
 13  cutoff_factor                   144316 non-null  int64  
 14  cutoff_timestamp                144316 non-null  object 
 15  actual_distance_to_destination  144316 non-null  float64
 16  actual_time                     144316 non-null  float64
 17  osrm_time                       144316 non-null  float64
 18  osrm_distance                   144316 non-null  float64
 19  factor                          144316 non-null  float64
 20  segment_actual_time             144316 non-null  float64
 21  segment_osrm_time               144316 non-null  float64
 22  segment_osrm_distance           144316 non-null  float64
 23  segment_factor                  144316 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 26.6+ MB
```

Column Profiling:

1. data - tells whether the data is testing or training data

2. trip_creation_time – Timestamp of trip creation

3. route_schedule_uuid – Unique Id for a particular route schedule

4. route_type – Transportation type

5. 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

6. Carting: Handling system consisting of small vehicles (carts)

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

8. source_center - Source ID of trip origin

9. source_name - Source Name of trip origin
10. destination_cente – Destination ID
11. destination_name – Destination Name
12. od_start_time – Trip start time
13. od_end_time – Trip end time
14. start_scan_to_end_scan – Time taken to deliver from source to destination
15. is_cutoff – Unknown field
16. cutoff_factor – Unknown field
17. cutoff_timestamp – Unknown field
18. actual_distance_to_destination – Distance in Kms between source and destination warehouse
19. actual_time – Actual time taken to complete the delivery (Cumulative)
20. 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)
21. 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)
22. factor – Unknown field
23. segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
24. segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
25. segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
26. segment_factor – Unknown field

```
data.describe(include='all').transpose()
```

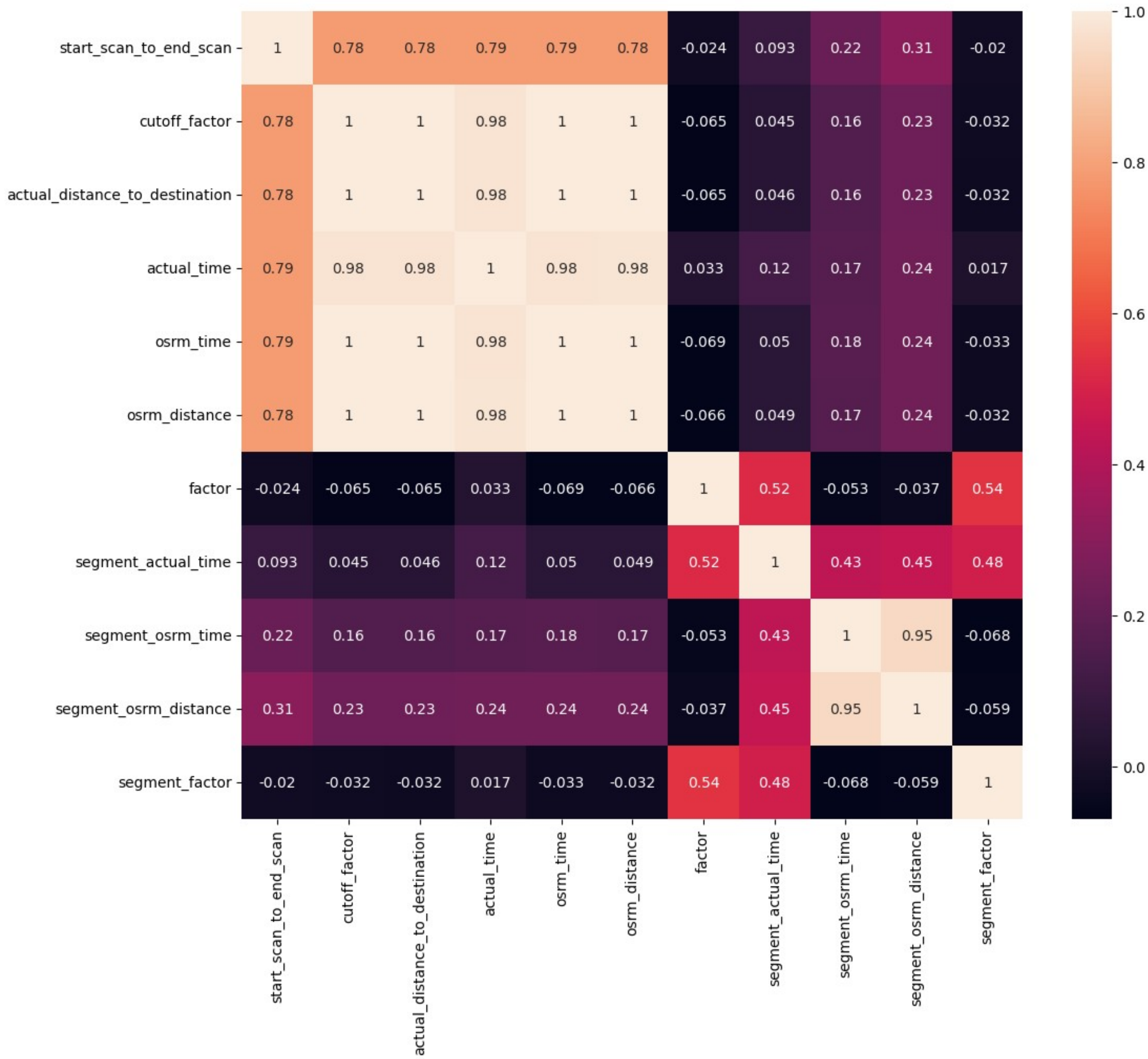
{"summary":"{\n \"name\": \"data\",\n \"rows\": 24,\n \"fields\": [\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"144316\",\n   \"max\": \"144316\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"144316\"\n   ],\n   \"semantic_type\": \"\",\

```
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"unique\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": 2,\n        \"max\": 92894,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"top\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 11,\
n        \"samples\": [\n          \"IND000000ACB\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"freq\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"39\",\n        \"max\":
\"118336\",\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          \"118336\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"mean\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": 2.120178354746856,\n        \"max\":
963.697698106932,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          285.5497853779207\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": 1.717064699874498,\n
\"max\": 1038.0829762943533,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          421.7178256506773\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"min\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": -244.0,\n        \"max\":
20.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"25%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 1.3478260869565215,\n        \"max\": 161.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
29.89625\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"50%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 1.6842105263157894,\n        \"max\": 451.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          78.6244\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"75%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 2.2122803049883686,\n        \"max\": 1645.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
346.3054\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 77.38709677419355,\n        \"max\": 7898.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
2326.1991000000003\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
numerical_columns = data.select_dtypes(include=['int64',
'float64']).columns.tolist()
numerical_columns
```

```
['start_scan_to_end_scan',
 'cutoff_factor',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'factor',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance',
 'segment_factor']
```

```
plt.figure(figsize = (12,10))
sns.heatmap(data[numerical_columns].corr(), annot = True)
plt.show()
```




```
'''
In the heatmap, we can see that
1. actual_time, osrm_time, actual_distance_to_destination and
osrm_distance are all highly correlated which is expected as time and
distance are directly proportional to each other.
2. segment_actual_time and segment_osrm_time are poorly correlated
(not expected) even though actual_time and osrm_time are highly
correlated.
3. segment_osrm_distance and segment_osrm_time are highly correlated
as expected.
'''
```

```
{"type":"string"}
```

```
data.reset_index(inplace = True)
```

```
'''

In this section,

We have detected the missing values and observed that the columns
'source_name' and 'destination_name'
have the missing values present but they are hardly 0.1%-0.2% of the
total data.

We tried to create a mapping for center and name using the columns
source_center, source_name, destination_center and destination_name
so that we can check if the source_center or destination_center where
the respective name column is having missing value can be filled using
this mapping.

We observed that those centers are not present anywhere else in the
data with proper names hence decided to drop those rows which are
having the missing values.

'''
```

{"type":"string"}

##Merging of rows and aggregation of fields

```
'''
A trip may include different source and destination centers.
So, the delivery details of one package is divided into several rows
(like connecting flights to reach a particular destination).
We shall combine these rows to prepare our data for analysing overall
time and distances.
We will use different aggregations like cumulative sums, first/last
element, sums, etc to merge the rows.

This merging will be done in 2 phases :
1. Merging rows based on a unique <‘segment_key’ made of ‘trip_uuid’,
‘source_center’, ‘destination_center’>
2. Further aggregate on the basis of only ‘trip_uuid’.
'''
```

{"type":"string"}

###Segment level data

####Segment Key creation

```
data['segment_key'] = data['trip_uuid'] + '_' + data['source_center']
+ '_' + data['destination_center']

data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data['segment_key'] = data['segment_key'].str.replace('trip-', '')

data['segment_key'].head()
```

```
0    153741093647649320_IND388121AAA_IND388620AAB
1    153741093647649320_IND388121AAA_IND388620AAB
2    153741093647649320_IND388121AAA_IND388620AAB
3    153741093647649320_IND388121AAA_IND388620AAB
4    153741093647649320_IND388121AAA_IND388620AAB
Name: segment_key, dtype: object
```

####Cumulative sum columns : segment_actual_time, segment_osrm_time, segment_osrm_distance

```
segment_cols = ['segment_actual_time', 'segment_osrm_time',
'segment_osrm_distance']

for col in segment_cols:
  data[col + '_sum'] = data.groupby('segment_key')[col].cumsum()

data[[col + '_sum' for col in segment_cols]]
```

{"type":"dataframe"}

```
'''
So, above, we have aggregated the time and distances of each segment
using cumulative sum.

So, <segment_actual_time_sum, segment_osrm_time_sum and
segment_osrm_distance_sum>
should ideally be equal to <actual_time, osrm_time and osrm_distance>
,but that is not the case actually.

'''
```

{"type":"string"}

####Segment Dictionary creation

```
data.columns
```

```
Index(['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',
       'segment_key', 'segment_actual_time_sum',
'segment_osrm_time_sum',
       'segment_osrm_distance_sum'],
      dtype='object')
```

```
create_segment_dict = {}

unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp',
'factor', 'segment_factor']
excluded_fields = ['segment_key', 'segment_actual_time',
'segment_osrm_time', 'segment_osrm_distance']


for col in data.columns:
  if col not in create_segment_dict.keys() and col not in
(unknown_fields + excluded_fields):
    create_segment_dict[col] = 'first'



col_last = ['destination_center', 'destination_name',
'actual_distance_to_destination', 'actual_time', 'osrm_time',
'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_time_sum',
'segment_osrm_distance_sum']

'''
'destination_center', #we need to take last destination for the trip
segment
'destination_name',  #we need to take last destination for the trip
segment
'actual_distance_to_destination', #since it is already cumulative
'actual_time', #since it is already cumulative
'osrm_time', #since it is already cumulative
'osrm_distance', #since it is already cumulative
'segment_actual_time_sum', #since we calculated it using cumulative
sum
'segment_osrm_time_sum', #since we calculated it using cumulative sum
'segment_osrm_distance_sum' #since we calculated it using cumulative
sum
'''

for col in col_last:
  create_segment_dict[col] = 'last'

create_segment_dict
```

```
{'index': 'first',
 'data': 'first',
 'trip_creation_time': 'first',
```

```
 'route_schedule_uuid': 'first',
 'route_type': 'first',
 'trip_uuid': 'first',
 'source_center': 'first',
 'source_name': 'first',
 'destination_center': 'last',
 'destination_name': 'last',
 'od_start_time': 'first',
 'od_end_time': 'first',
 'start_scan_to_end_scan': 'first',
 'actual_distance_to_destination': 'last',
 'actual_time': 'last',
 'osrm_time': 'last',
 'osrm_distance': 'last',
 'segment_actual_time_sum': 'last',
 'segment_osrm_time_sum': 'last',
 'segment_osrm_distance_sum': 'last'}
```

####Group by 'Segment Key' and use dictionary for forming segment dataframe

```
segment = data.groupby(by =
['segment_key']).agg(create_segment_dict).reset_index()
segment.sort_values(by = ['segment_key', 'od_end_time'], ascending =
True, inplace = True)
segment
```

{"type":"dataframe","variable_name":"segment"}

```
segment[['segment_key', 'actual_time', 'segment_actual_time_sum',
'osrm_time', 'segment_osrm_time_sum',
'actual_distance_to_destination', 'osrm_distance',
'segment_osrm_distance_sum']]
```

{"summary":"{\n \"name\": \"segment[['segment_key', 'actual_time',
'segment_actual_time_sum', 'osrm_time', 'segment_osrm_time_sum',
'actual_distance_to_destination', 'osrm_distance',
'segment_osrm_distance_sum']]\",\n \"rows\": 26222,\n \"fields\": [\
n   {\n   \"column\": \"segment_key\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 26222,\n
\"samples\": [\n
\"153753822247681124_IND000000AAL_IND411033AAA\",\n
\"153703052434656818_IND302014AAA_IND000000ACB\",\n
\"153767191928611549_IND721133AAA_IND721434AAB\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"actual_time\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
385.7309075034264,\n   \"min\": 9.0,\n   \"max\": 4532.0,\n
\"num_unique_values\": 1657,\n   \"samples\": [\n
3347.0,\n   893.0,\n   456.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

n   },\n   {\n   \"column\": \"segment_actual_time_sum\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 382.1506865793467,\n   \"min\": 9.0,\n   \"max\": 4504.0,\n   \"num_unique_values\": 1676,\n   \"samples\": [\n   2345.0,\n   1653.0,\n   337.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"osrm_time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 185.55435921024065,\n   \"min\": 6.0,\n   \"max\": 1686.0,\n   \"num_unique_values\": 560,\n   \"samples\": [\n   1168.0,\n   1326.0,\n   125.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"segment_osrm_time_sum\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 216.20273873603804,\n   \"min\": 6.0,\n   \"max\": 1938.0,\n   \"num_unique_values\": 1102,\n   \"samples\": [\n   1114.0,\n   1689.0,\n   424.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"actual_distance_to_destination\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 209.95235530628602,\n   \"min\": 9.00135089146556,\n   \"max\": 1927.4477046975032,\n   \"num_unique_values\": 26193,\n   \"samples\": [\n   10.672662721245022,\n   10.481478729081116,\n   100.17442498027144\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"osrm_distance\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 254.42646843294298,\n   \"min\": 9.0729,\n   \"max\": 2326.1991000000003,\n   \"num_unique_values\": 25871,\n   \"samples\": [\n   35.1773,\n   29.4577,\n   22.5882\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"segment_osrm_distance_sum\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 286.67010645655307,\n   \"min\": 9.0729,\n   \"max\": 2640.9247,\n   \"num_unique_values\": 25948,\n   \"samples\": [\n   21.0942,\n   833.9528,\n   14.364099999999999\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
segment[segment['trip_uuid'] == 'trip-153741093647649320']
[['trip_uuid','segment_key', 'actual_time', 'segment_actual_time_sum',
'osrm_time', 'segment_osrm_time_sum',
'actual_distance_to_destination', 'osrm_distance',
'segment_osrm_distance_sum']]
```

{"summary":"{\n \"name\": \"segment[segment['trip_uuid'] == 'trip-153741093647649320'][['trip_uuid','segment_key', 'actual_time', 'segment_actual_time_sum', 'osrm_time', 'segment_osrm_time_sum', 'actual_distance_to_destination', 'osrm_distance',

```
'segment_osrm_distance_sum']]\",\n  \"rows\": 2,\n  \"fields\": [\n
{\n      \"column\": \"trip_uuid\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"trip-153741093647649320\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_key\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"153741093647649320_IND388620AAB_IND388320AAA\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24.041630560342615,\n        \"min\": 68.0,\n        \"max\": 102.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          102.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"segment_actual_time_sum\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
23.33452377915607,\n        \"min\": 67.0,\n        \"max\": 100.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          100.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7071067811865476,\n        \"min\": 44.0,\n        \"max\": 45.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          45.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"segment_osrm_time_sum\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 44.0,\n        \"max\": 44.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          44.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"actual_distance_to_destination\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.976750868495821,\n
\"min\": 39.38604027413606,\n        \"max\": 43.59580172416874,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
43.59580172416874\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6962880474343909,\n        \"min\":
53.2334,\n        \"max\": 54.2181,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          53.2334\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_distance_sum\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.656034491492915,\n        \"min\": 49.477199999999996,\n
\"max\": 53.2334,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          53.2334\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
'''
Now, as we see above, the above particular trip-uuid has two entries associated with it.
We will further aggregate using only trip_uuid to have just one entry for each trip_uuid.

'''
```

{"type":"string"}

```
segment.shape
```

```
(26222, 21)
```

```
segment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26222 entries, 0 to 26221
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   segment_key                     26222 non-null  object
 1   index                           26222 non-null  int64
 2   data                            26222 non-null  object
 3   trip_creation_time              26222 non-null  object
 4   route_schedule_uuid             26222 non-null  object
 5   route_type                      26222 non-null  object
 6   trip_uuid                       26222 non-null  object
 7   source_center                   26222 non-null  object
 8   source_name                     26222 non-null  object
 9   destination_center              26222 non-null  object
 10  destination_name                26222 non-null  object
 11  od_start_time                   26222 non-null  object
 12  od_end_time                     26222 non-null  object
 13  start_scan_to_end_scan          26222 non-null  float64
 14  actual_distance_to_destination  26222 non-null  float64
 15  actual_time                     26222 non-null  float64
 16  osrm_time                       26222 non-null  float64
 17  osrm_distance                   26222 non-null  float64
 18  segment_actual_time_sum         26222 non-null  float64
 19  segment_osrm_time_sum           26222 non-null  float64
 20  segment_osrm_distance_sum       26222 non-null  float64
dtypes: float64(8), int64(1), object(12)
memory usage: 4.4+ MB
```

```
'''
So we have reduced the number of rows from 144316 to just 26222. We now have 21 columns.
'''
```

{"type":"string"}

```
segment[segment['trip_uuid'] == 'trip-153741093647649320']
```

{"type":"dataframe"}

####Feature Creation - od_time_diff_hour

```
cols_datetime = ['trip_creation_time', 'od_start_time', 'od_end_time']

segment[['trip_creation_time', 'od_start_time', 'od_end_time']]
```

{"summary":"{\n \"name\": \"segment[['trip_creation_time', 'od_start_time', 'od_end_time']]\",\n \"rows\": 26222,\n \"fields\": [\n   {\n   \"column\": \"trip_creation_time\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"num_unique_values\": 14787,\n   \"samples\": [\n   \"2018-09-26 17:50:27.589424\",\n   \"2018-09-19 18:18:25.743990\",\n   \"2018-09-17 05:32:36.314228\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"od_start_time\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"num_unique_values\": 26222,\n   \"samples\": [\n   \"2018-09-21 13:57:02.477063\",\n   \"2018-09-15 16:55:24.346802\",\n   \"2018-09-23 04:42:17.160292\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"od_end_time\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"num_unique_values\": 26222,\n   \"samples\": [\n   \"2018-09-21 16:31:55.506368\",\n   \"2018-09-15 23:48:44.345808\",\n   \"2018-09-23 08:06:17.623714\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
for col in cols_datetime:
  segment[col] = pd.to_datetime(segment[col])

segment[['trip_creation_time', 'od_start_time', 'od_end_time']]
```

{"summary":"{\n \"name\": \"segment[['trip_creation_time', 'od_start_time', 'od_end_time']]\",\n \"rows\": 26222,\n \"fields\": [\n   {\n   \"column\": \"trip_creation_time\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2018-09-12 00:00:16.535741\",\n   \"max\": \"2018-10-03 23:59:42.701692\",\n   \"num_unique_values\": 14787,\n   \"samples\": [\n   \"2018-09-26 17:50:27.589424\",\n   \"2018-09-19 18:18:25.743990\",\n   \"2018-09-17 05:32:36.314228\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"od_start_time\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2018-09-12 00:00:16.535741\",\n   \"max\": \"2018-10-06 04:27:23.392375\",\n

```
\"num_unique_values\": 26222,\n        \"samples\": [\n
\"2018-09-21 13:57:02.477063\",\n          \"2018-09-15
16:55:24.346802\",\n          \"2018-09-23 04:42:17.160292\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"od_end_time\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": \"2018-09-12 00:50:10.814399\",\n
\"max\": \"2018-10-08 03:00:24.353479\",\n
\"num_unique_values\": 26222,\n        \"samples\": [\n
\"2018-09-21 16:31:55.506368\",\n          \"2018-09-15
23:48:44.345808\",\n          \"2018-09-23 08:06:17.623714\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
segment['od_time_diff_hour'] = ((segment['od_end_time'] -
segment['od_start_time']).dt.total_seconds())/60

segment[segment['trip_uuid'] == 'trip-153741093647649320']
[['od_start_time', 'od_end_time', 'od_time_diff_hour',
'start_scan_to_end_scan']]
```


```
{"summary":"{\n  \"name\": \"segment[segment['trip_uuid'] == 'trip-
153741093647649320'][['od_start_time', 'od_end_time',
'od_time_diff_hour', 'start_scan_to_end_scan']]\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"od_start_time\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2018-09-20 03:21:32.418600\",\n        \"max\": \"2018-09-20
04:47:45.236797\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"2018-09-20 04:47:45.236797\",\n
\"2018-09-20 03:21:32.418600\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"od_end_time\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2018-09-20
04:47:45.236797\",\n        \"max\": \"2018-09-20 06:36:55.627764\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"2018-
09-20 06:36:55.627764\",\n          \"2018-09-20 04:47:45.236797\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"od_time_diff_hour\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16.234850787415603,\n        \"min\": 86.21363661666666,\n
\"max\": 109.17318278333333,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          109.17318278333333,\n
86.21363661666666\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_scan_to_end_scan\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16.263455967290593,\n
\"min\": 86.0,\n        \"max\": 109.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          109.0,\n
86.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
'''
In this section,

We created a dictionary 'create_trip_dict' which will have the
information about which agg function should be used for each column of
dataframe segment
to form a new dataframe trip which will be obtained by grouping the
data from segment dataframe at trip_uuid level.

trip dataframe is trip level data.
'''
```

{"type":"string"}

###Trip level data

####Trip dictionary creation

```
segment.columns
```

```
Index(['segment_key', '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',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time_sum',
'segment_osrm_time_sum',
       'segment_osrm_distance_sum', 'od_time_diff_hour'],
      dtype='object')
```

```
create_trip_dict = {}

excluded_columns = ['index', 'trip_uuid', 'od_start_time',
'od_end_time', 'segment_key']

for col in segment.columns:
  if col not in create_trip_dict.keys() and col not in
(excluded_columns):
    create_trip_dict[col] = 'first'

cols_last = ['destination_center', 'destination_name']
cols_sum = ['start_scan_to_end_scan', 'od_time_diff_hour',
'actual_distance_to_destination', 'actual_time', 'osrm_time',
'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_time_sum',
'segment_osrm_distance_sum']

for col in cols_last:
  create_trip_dict[col] = 'last'

for col in cols_sum:
```

```
  create_trip_dict[col] = 'sum'

create_trip_dict
```

```
{'data': 'first',
 'trip_creation_time': 'first',
 'route_schedule_uuid': 'first',
 'route_type': 'first',
 'source_center': 'first',
 'source_name': 'first',
 'destination_center': 'last',
 'destination_name': 'last',
 'start_scan_to_end_scan': 'sum',
 'actual_distance_to_destination': 'sum',
 'actual_time': 'sum',
 'osrm_time': 'sum',
 'osrm_distance': 'sum',
 'segment_actual_time_sum': 'sum',
 'segment_osrm_time_sum': 'sum',
 'segment_osrm_distance_sum': 'sum',
 'od_time_diff_hour': 'sum'}
```

####Group By Trip_uuid to get trip level details

```
trip =
segment.groupby(['trip_uuid']).agg(create_trip_dict).reset_index()
trip
```

{"summary":"{\n \"name\": \"trip\",\n \"rows\": 14787,\n \"fields\": [\n   {\n   \"column\": \"trip_uuid\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 14787,\n   \"samples\": [\n   \"trip-153798422758908322\",\n   \"trip-153738110574360102\",\n   \"trip-153716235631392457\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"data\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"test\",\n   \"training\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"trip_creation_time\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2018-09-12 00:00:16.535741\",\n   \"max\": \"2018-10-03 23:59:42.701692\",\n   \"num_unique_values\": 14787,\n   \"samples\": [\n   \"2018-09-26 17:50:27.589424\",\n   \"2018-09-19 18:18:25.743990\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"route_schedule_uuid\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 1497,\n   \"samples\": [\n   \"thanos::sroute:402604bf-e4e3-4723-85ae-

```
a3a46a73f2ed\",\n          \"thanos::sroute:16f438c4-c258-4955-8358-
bdb1e256f1dc\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"route_type\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Carting\",\n          \"FTL\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"source_center\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 930,\n        \"samples\": [\n
\"IND845455AAB\",\n          \"IND402301AAA\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"source_name\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 930,\n        \"samples\": [\n
\"Narktiganj_Central_DPP_2 (Bihar)\",\n          \"Mahad_Govndsgr_D
(Maharashtra)\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_center\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1035,\n
\"samples\": [\n          \"IND573116AAA\",\n
\"IND815351AAA\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_name\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1035,\n
\"samples\": [\n          \"Channaraya_patna_D (Karnataka)\",\n
\"Jamtara_D (Jharkhand)\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"start_scan_to_end_scan\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 658.2549358325907,\n
\"min\": 23.0,\n        \"max\": 7898.0,\n
\"num_unique_values\": 2203,\n        \"samples\": [\n
2584.0,\n          3410.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"actual_distance_to_destination\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
305.50298245812365,\n        \"min\": 9.00246144174878,\n
\"max\": 2186.531787238833,\n        \"num_unique_values\": 14771,\n
\"samples\": [\n          283.3099851850191,\n
152.63738381003816\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 561.51793561062,\n        \"min\": 9.0,\
n        \"max\": 6265.0,\n        \"num_unique_values\": 1850,\n
\"samples\": [\n          377.0,\n          1029.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
271.4594945627529,\n        \"min\": 6.0,\n        \"max\": 2032.0,\n
```

\"num_unique_values\": 814,\n   \"samples\": [\n   116.0,\
n   628.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"osrm_distance\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 370.56556422647463,\n   \"min\":
9.0729,\n   \"max\": 2840.081,\n   \"num_unique_values\":
14704,\n   \"samples\": [\n   371.93600000000004,\n
12.8317\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"segment_actual_time_sum\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 556.365910826273,\n
\"min\": 9.0,\n   \"max\": 6230.0,\n
\"num_unique_values\": 1885,\n   \"samples\": [\n   70.0,\
n   739.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"segment_osrm_time_sum\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 314.6792793964003,\n
\"min\": 6.0,\n   \"max\": 2564.0,\n
\"num_unique_values\": 1240,\n   \"samples\": [\n
756.0,\n   175.0\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"segment_osrm_distance_sum\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 416.84627901238525,\n
\"min\": 9.0729,\n   \"max\": 3523.6323999999995,\n
\"num_unique_values\": 14724,\n   \"samples\": [\n
649.9442,\n   12.4539\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"od_time_diff_hour\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 658.4154895334472,\n
\"min\": 23.46146848333333,\n   \"max\": 7898.551954566667,\n
\"num_unique_values\": 14787,\n   \"samples\": [\n
160.23845045000002,\n   225.63755579999997\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"trip"}

```
trip[trip['trip_uuid'] == 'trip-153741093647649320']
```

{"summary":"{\n \"name\": \"trip[trip['trip_uuid'] == 'trip-
153741093647649320']\",\n \"rows\": 1,\n \"fields\": [\n   {\n
\"column\": \"trip_uuid\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 1,\n
\"samples\": [\n   \"trip-153741093647649320\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"data\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 1,\n
\"samples\": [\n   \"training\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"trip_creation_time\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2018-09-20 02:35:36.476840\",\n   \"max\": \"2018-09-20

```
02:35:36.476840\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"2018-09-20 02:35:36.476840\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"route_schedule_uuid\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"route_type\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"Carting\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"source_center\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"IND388121AAA\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"source_name\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"Anand_VUNagar_DC (Gujarat)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"destination_center\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"IND388320AAA\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_name\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Anand_Vaghasi_IP (Gujarat)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"start_scan_to_end_scan\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 195.0,\n        \"max\": 195.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          195.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"actual_distance_to_destination\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
82.98184199830479,\n        \"max\": 82.98184199830479,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
82.98184199830479\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 170.0,\n
\"max\": 170.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          170.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

null,\n   \"min\": 89.0,\n   \"max\": 89.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   89.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"osrm_distance\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 107.45150000000001,\n   \"max\":
107.45150000000001,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   107.45150000000001\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"segment_actual_time_sum\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 167.0,\n   \"max\": 167.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   167.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"segment_osrm_time_sum\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 88.0,\n   \"max\": 88.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   88.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"segment_osrm_distance_sum\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 102.7106,\n   \"max\": 102.7106,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   102.7106\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"od_time_diff_hour\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": null,\n   \"min\":
195.38681939999998,\n   \"max\": 195.38681939999998,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
195.38681939999998\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
trip.shape
```

```
(14787, 18)
```

```
'''
So, now we have only 14787 rows and 18 columns.
'''
```

{"type":"string"}

```
trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14787 non-null  object
 1   data                            14787 non-null  object
```

```
 2   trip_creation_time                14787 non-null  datetime64[ns]
 3   route_schedule_uuid               14787 non-null  object
 4   route_type                        14787 non-null  object
 5   source_center                     14787 non-null  object
 6   source_name                       14787 non-null  object
 7   destination_center                14787 non-null  object
 8   destination_name                  14787 non-null  object
 9   start_scan_to_end_scan            14787 non-null  float64
 10  actual_distance_to_destination    14787 non-null  float64
 11  actual_time                       14787 non-null  float64
 12  osrm_time                         14787 non-null  float64
 13  osrm_distance                     14787 non-null  float64
 14  segment_actual_time_sum           14787 non-null  float64
 15  segment_osrm_time_sum             14787 non-null  float64
 16  segment_osrm_distance_sum         14787 non-null  float64
 17  od_time_diff_hour                 14787 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(8)
memory usage: 2.0+ MB
```

#Build some features to prepare the data for actual analysis

##Extracting State, City, Place and Code

```
'''
We first look at the following two features and extract relevant
information from those :
Destination Name: We split and extract features out of destination.
City-place-code (State)
Source Name: We split and extract features out of source. City-place-
code (State)
'''

{"type":"string"}

trip['source_name'] = trip['source_name'].str.lower()
trip['destination_name'] = trip['destination_name'].str.lower()

trip[['source_name','destination_name']]
```

{"summary":"{\n \"name\":
\"trip[['source_name','destination_name']]\",\n \"rows\": 14787,\n
\"fields\": [\n   {\n   \"column\": \"source_name\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 930,\n   \"samples\": [\n
\"narktiganj_central_dpp_2 (bihar)\",\n   \"mahad_govndsgr_d
(maharashtra)\",\n   \"faridabad_mthurard_l (haryana)\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"destination_name\",\n

\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1035,\n   \"samples\": [\n
\"channaraya_patna_d (karnataka)\",\n   \"jamtara_d
(jharkhand)\",\n   \"raipur_byprddpp_d (rajasthan)\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
def place2state(x):
  state = x.split('(')[1]
  return state[:-1]

def place2city(x):
  city = x.split('(')[0]
  city=city.split('_')[0]

  #dealing with edge cases
  if city == "pnq vadgaon shei dpc" : city = "vadgaonsheri"
  if city in ["pnq pashan dpc", "pnq rahatani dpc", "pune balaji
nagar"] : city = 'pune'
  if city == "hbr layout pc" : city = "bengaluru"
  if city == "bhopal mp nagar" : city = "bhopal"
  if city == "mumbai antop hill" : city = "mumbai"
  if city == "bangalore" : city = "bengaluru"
  if city == "mumbai hub " : city = "mumbai"

  return city


def place2city_place(x):
  #removing state
  x = x.split('(')[0]
  len_ = len(x.split('_'))
  if len_ >= 3:
    return x.split('_')[1]

  #small cities have same city and place name
  if len_ == 2:
    return x.split('_')[0]

  #dealing with edge cases or improper naming conventions
  return x.split(' ')[0]


def place2code(x):
  #removing state
  x = x.split('(')[0]
  if(len(x.split('_')) >=3):
    return x.split('_')[-1]
  return 'none'
```

```
trip["destination_state"] = trip["destination_name"].apply(lambda x:
place2state(x))
trip["destination_city"] = trip["destination_name"].apply(lambda x:
place2city(x))
trip["destination_place"] = trip["destination_name"].apply(lambda x:
place2city_place(x))
trip["destination_code"] = trip["destination_name"].apply(lambda
x:place2code(x))

trip["source_state"] = trip["source_name"].apply(lambda x:
place2state(x))
trip["source_city"] = trip["source_name"].apply(lambda x:
place2city(x))
trip["source_place"] = trip["source_name"].apply(lambda x:
place2city_place(x))
trip["source_code"] = trip["source_name"].apply(lambda x:
place2code(x))

trip[["destination_state","destination_city","destination_place","dest
ination_code"]]
```


```
{"summary":"{\n  \"name\":
\"trip[[\\\"destination_state\\\",\\\"destination_city\\\",\\\"destina
tion_place\\\",\\\"destination_code\\\"]]\",\n  \"rows\": 14787,\n
\"fields\": [\n    {\n      \"column\": \"destination_state\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 31,\n        \"samples\": [\n
\"nagaland\",\n          \"dadra and nagar haveli\",\n
\"arunachal pradesh\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_city\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 853,\n
\"samples\": [\n          \"gzb\",\n          \"sangareddy\",\n
\"kalwakurthy\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_place\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 864,\n
\"samples\": [\n          \"vardhard\",\n          \"pettah\",\n
\"ghaziabad\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_code\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 32,\n
\"samples\": [\n          \"23 \",\n          \"12 \",\n          \"4
\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
trip[["source_state","source_city","source_place","source_code"]]
```


```
{"summary":"{\n  \"name\":
\"trip[[\\\"source_state\\\",\\\"source_city\\\",\\\"source_place\\\",
```

\\\"source_code\\\"]]\",\n \"rows\": 14787,\n \"fields\": [\n   {\n
\"column\": \"source_state\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 29,\n
\"samples\": [\n   \"mizoram\",\n   \"jharkhand\",\n
\"andhra pradesh\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"source_city\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 727,\n
\"samples\": [\n   \"hanumangarh\",\n   \"pilani \",\n
\"chinnur\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"source_place\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 768,\n
\"samples\": [\n   \"lamtidpp\",\n   \"kappalur\",\n
\"dmodrngr\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"source_code\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 31,\n
\"samples\": [\n   \"5 \",\n   \"dc \",\n   \"cp
\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Extracting trip year, month, day

```
'''
Next we extract features from trip_creation_time. These features
include : Year, Month, Day,
Week, DayofWeek, Hour
'''
```

{"type":"string"}

```
trip["trip_creation_time"] =
pd.to_datetime(trip["trip_creation_time"])
trip["trip_year"] = trip["trip_creation_time"].dt.year
trip["trip_month"] = trip["trip_creation_time"].dt.month
trip["trip_hour"] = trip["trip_creation_time"].dt.hour
trip["trip_day"] = trip["trip_creation_time"].dt.day
trip["trip_week"] = trip["trip_creation_time"].dt.isocalendar().week
trip["trip_dayofweek"] = trip["trip_creation_time"].dt.dayofweek

trip[['trip_year','trip_month','trip_hour','trip_day','trip_week','tri
p_dayofweek']]
```

{"summary":"{\n \"name\":
\"trip[['trip_year','trip_month','trip_hour','trip_day','trip_week','t
rip_dayofweek']]\",\n \"rows\": 14787,\n \"fields\": [\n   {\n
\"column\": \"trip_year\",\n   \"properties\": {\n

```
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 2018,\n
\"max\": 2018,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          2018\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"trip_month\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 9,\n
\"max\": 10,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          10\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7,\n        \"min\": 0,\n
\"max\": 23,\n        \"num_unique_values\": 24,\n        \"samples\":
[\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_day\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7,\n        \"min\": 1,\n
\"max\": 30,\n        \"num_unique_values\": 22,\n        \"samples\":
[\n          12\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_week\",\n      \"properties\": {\n        \"dtype\":
\"UInt32\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"38\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_dayofweek\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 6,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

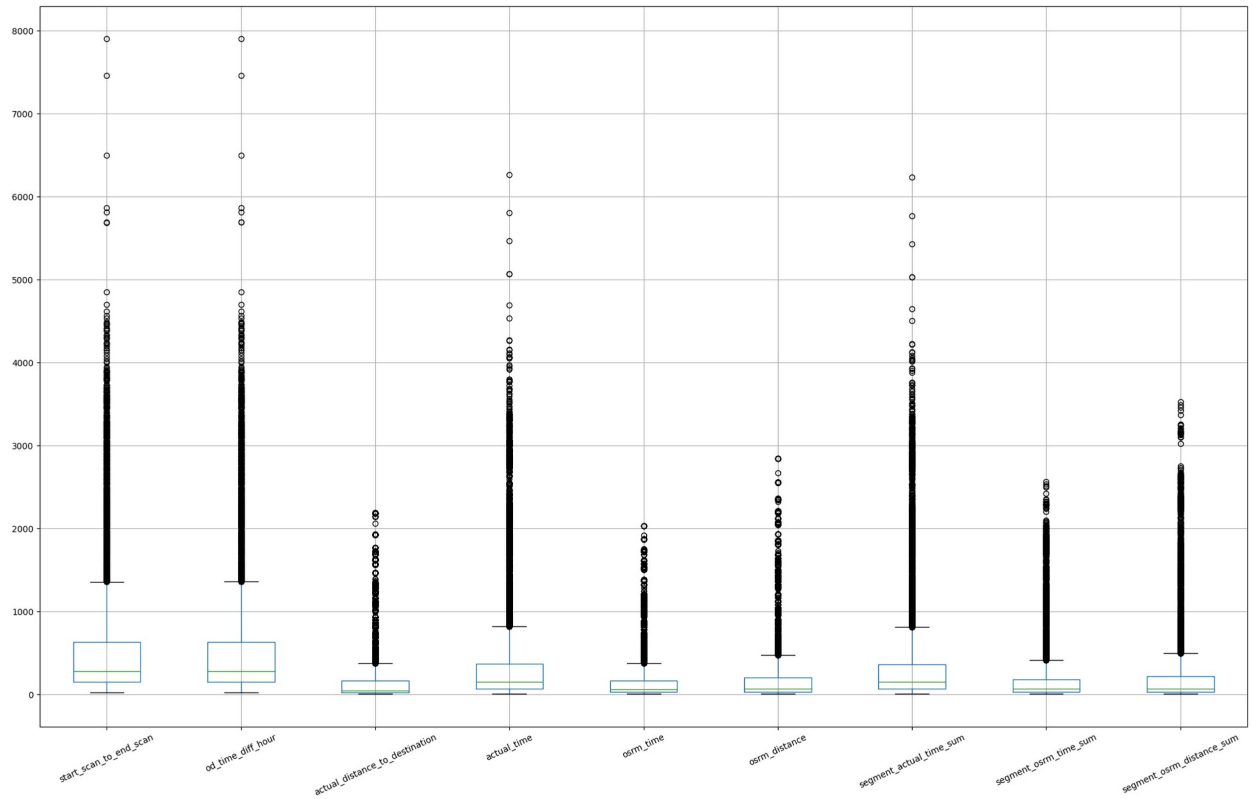

#In-depth analysis and feature engineering

##Finding, visualizing and removing outliers (using IQR) from numeric variables

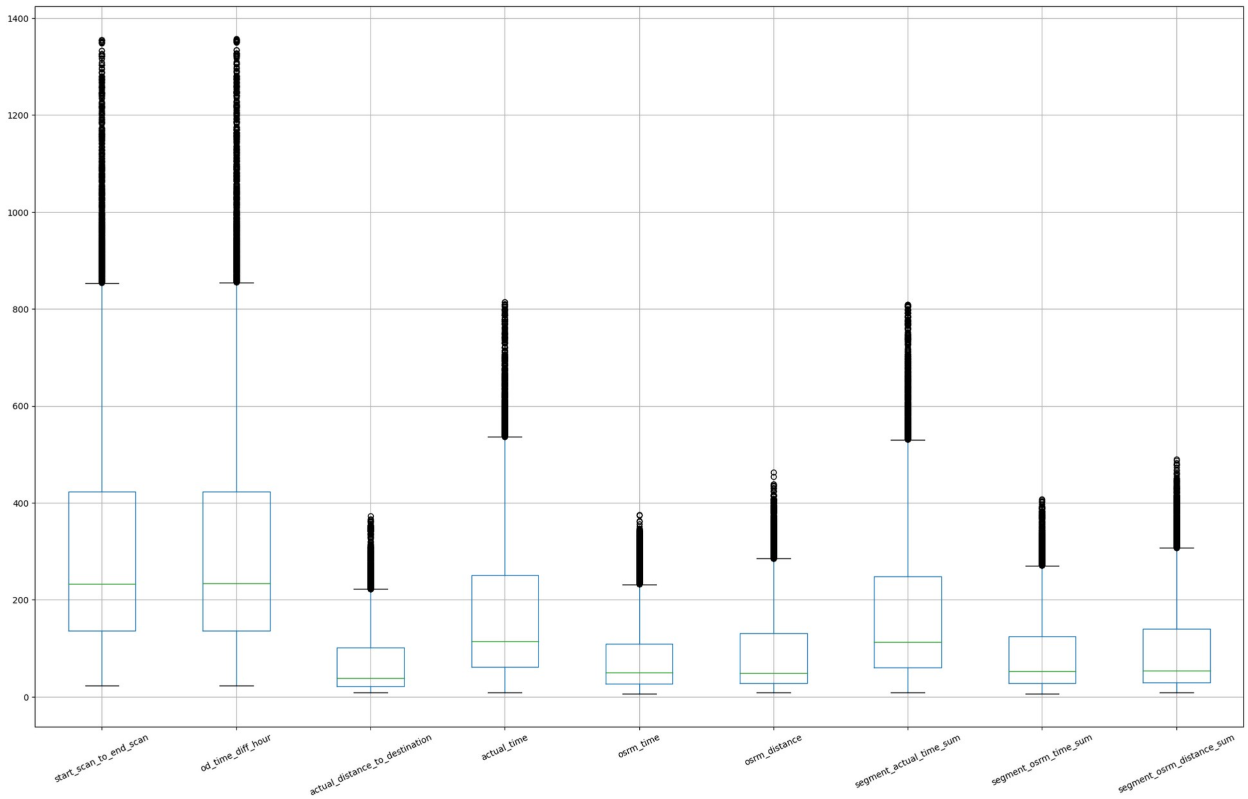
```
trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 32 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14787 non-null  object
 1   data                            14787 non-null  object
 2   trip_creation_time              14787 non-null  datetime64[ns]
 3   route_schedule_uuid             14787 non-null  object
 4   route_type                      14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
```

```
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  actual_distance_to_destination  14787 non-null  float64
 11  actual_time                     14787 non-null  float64
 12  osrm_time                       14787 non-null  float64
 13  osrm_distance                   14787 non-null  float64
 14  segment_actual_time_sum         14787 non-null  float64
 15  segment_osrm_time_sum           14787 non-null  float64
 16  segment_osrm_distance_sum       14787 non-null  float64
 17  od_time_diff_hour               14787 non-null  float64
 18  destination_state               14787 non-null  object
 19  destination_city                14787 non-null  object
 20  destination_place               14787 non-null  object
 21  destination_code                14787 non-null  object
 22  source_state                    14787 non-null  object
 23  source_city                     14787 non-null  object
 24  source_place                    14787 non-null  object
 25  source_code                     14787 non-null  object
 26  trip_year                       14787 non-null  int64
 27  trip_month                      14787 non-null  int64
 28  trip_hour                       14787 non-null  int64
 29  trip_day                        14787 non-null  int64
 30  trip_week                       14787 non-null  UInt32
 31  trip_dayofweek                  14787 non-null  int64
dtypes: UInt32(1), datetime64[ns](1), float64(9), int64(5), object(16)
memory usage: 3.6+ MB

num_cols = ['start_scan_to_end_scan','od_time_diff_hour',
'actual_distance_to_destination','actual_time','osrm_time','osrm_dista
nce',
'segment_actual_time_sum','segment_osrm_time_sum',
'segment_osrm_distance_sum']

trip[num_cols].boxplot(rot=25, figsize=(25,15))
plt.show()
```




```
Q1 = trip[num_cols].quantile(0.25)
Q3 = trip[num_cols].quantile(0.75)
IQR = Q3-Q1

trip = trip[~((trip[num_cols] < (Q1 - 1.5 * IQR))|(trip[num_cols] >
(Q3 + 1.5 *IQR))).any(axis=1)]

trip = trip.reset_index(drop=True)

trip[num_cols].boxplot(rot=25, figsize=(25,15))
plt.show()
```




```
trip.shape
```

```
(12723, 32)
```

##Handling Categorical Variables

###One hot encoding

```
categorical_columns = ['route_type']

encoder = OneHotEncoder(sparse_output = False)
one_hot_encoded = encoder.fit_transform(trip[categorical_columns])
one_hot_df = pd.DataFrame(one_hot_encoded, columns =
encoder.get_feature_names_out(categorical_columns))

trip = pd.concat([trip, one_hot_df], axis = 1)
#trip.drop(columns = ['route_type'], inplace = True)
trip
```

{"type":"dataframe","variable_name":"trip"}

```
'''
In this section,

We have implemented one hot encoding for the categorical column
'route_type'.
```

```
Hence two new columns have been added to the data which are
'route_type_carting' and 'route_type_FTL'.

'''
```

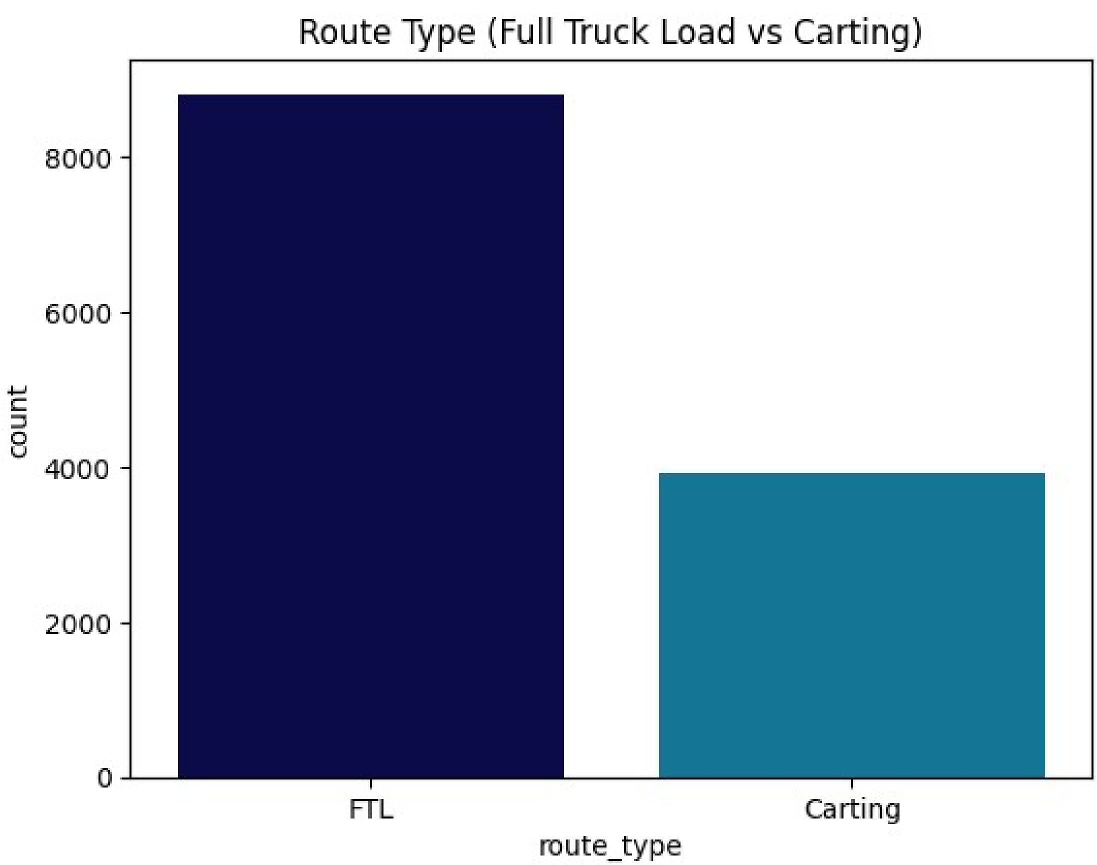
{"type":"string"}

##Univariate Analysis

###Distribution of Route Types

```
ax = sns.countplot(x = "route_type", data = trip, palette = 'ocean')
ax.set_xticklabels(["FTL","Carting"])
plt.title("Route Type (Full Truck Load vs Carting)")
plt.show()
```

```
<ipython-input-403-377137ad8aed>:2: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax.set_xticklabels(["FTL","Carting"])
```




```
'''
The majority of trips (8812) involved handling systems made of small
vehicles (carts).
```

```
The rest of the trips (3911) involved Full Truck Load: FTL shipments
get to the destination sooner,
as the truck is making no other pickups or drop-offs along the way.
'''
```

{"type":"string"}

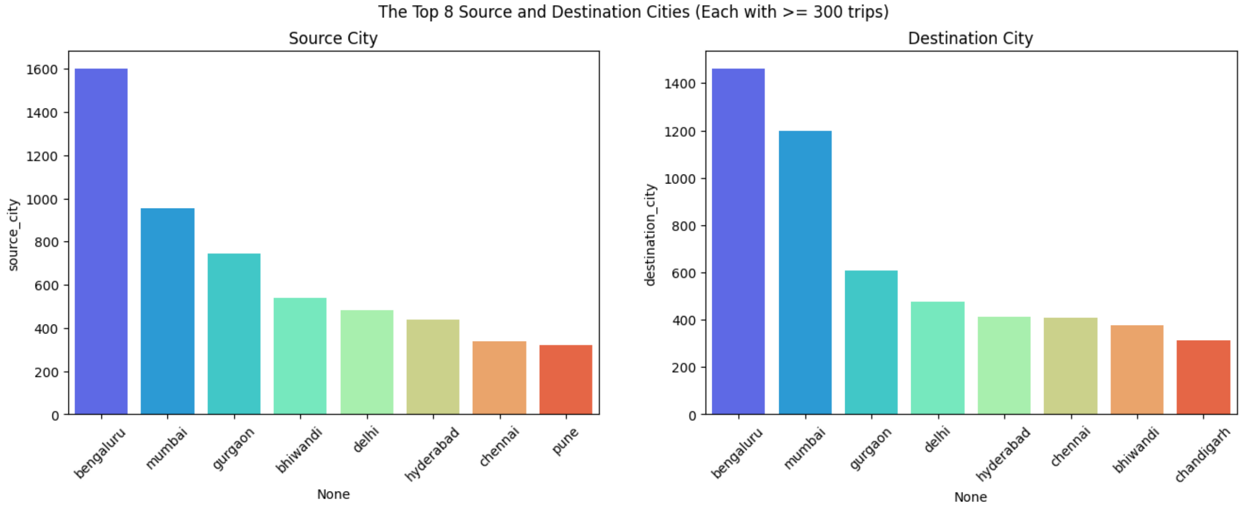
###Top source and destination cities

```
source300 = pd.DataFrame(trip["source_city"].value_counts()[0:8])
destination300 = pd.DataFrame(trip["destination_city"].value_counts()
[0:8])
destination300
```

{"summary":"{\n \"name\": \"destination300\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"destination_city\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
429,\n   \"min\": 313,\n   \"max\": 1462,\n
\"num_unique_values\": 8,\n   \"samples\": [\n   1198,\n
410,\n   1462\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"destination300"}

```
fig, ax = plt.subplots(1,2,figsize=(16,5))

sns.barplot(x = source300.index, y = source300['source_city'], data =
source300, ax = ax[0], palette = 'rainbow')
ax[0].set_xticklabels(source300.index,rotation=45)
ax[0].set_title("Source City")

sns.barplot(x = destination300.index, y =
destination300['destination_city'], data = destination300, ax = ax[1],
palette = 'rainbow')
ax[1].set_xticklabels(destination300.index,rotation=45)
ax[1].set_title("Destination City")

plt.suptitle("The Top 8 Source and Destination Cities (Each with >=
300 trips)")
plt.show()
```

```
<ipython-input-406-4d3af47b99a6>:4: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[0].set_xticklabels(source300.index,rotation=45)
<ipython-input-406-4d3af47b99a6>:8: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[1].set_xticklabels(destination300.index,rotation=45)
```




```
'''
So we see that Bengaluru, Mumbai and Gurgaon are both the top source
and destination cities.
More trips are starting at Bhiwandi than ending there. Delhi,
Hyderabad and Chennai also maintain
their relative ordering in source and destination.
'''
```

{"type":"string"}

```
trip.groupby(['destination_city','source_city'])
['actual_time'].sum().reset_index().sort_values(by='actual_time',
ascending=False)
```

{"summary":"{\n \"name\": \"trip\",\n \"rows\": 1561,\n \"fields\":
[\n   {\n   \"column\": \"destination_city\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 824,\n   \"samples\": [\n
\"jangipur\",\n   \"srivijaynagar\",\n   \"mandi\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"source_city\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 703,\n   \"samples\": [\n
\"ashokngr\",\n   \"srisailam\",\n   \"dhule\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"actual_time\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 4321.226675202434,\n   \"min\":
17.0,\n   \"max\": 121144.0,\n   \"num_unique_values\":
1010,\n   \"samples\": [\n   511.0,\n   268.0,\n
419.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
trip.groupby(['destination_city','source_city'])
['actual_distance_to_destination'].sum().reset_index().sort_values(by=
'actual_distance_to_destination',ascending=False)
```

{"summary":"{\n \"name\": \"trip\",\n \"rows\": 1561,\n \"fields\": [\n   {\n   \"column\": \"destination_city\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 824,\n   \"samples\": [\n   \"buldhana\",\n   \"jalalabad\",\n   \"amritsar\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"source_city\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 703,\n   \"samples\": [\n   \"hanumangarh\",\n   \"yamunanagar\",\n   \"tirupati\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"actual_distance_to_destination\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1564.9216864897098,\n   \"min\": 9.040985753808274,\n   \"max\": 42937.78029525657,\n   \"num_unique_values\": 1561,\n   \"samples\": [\n   18.474809089863676,\n   108.09600044209083,\n   586.3780455630226\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

###Top source and destination states

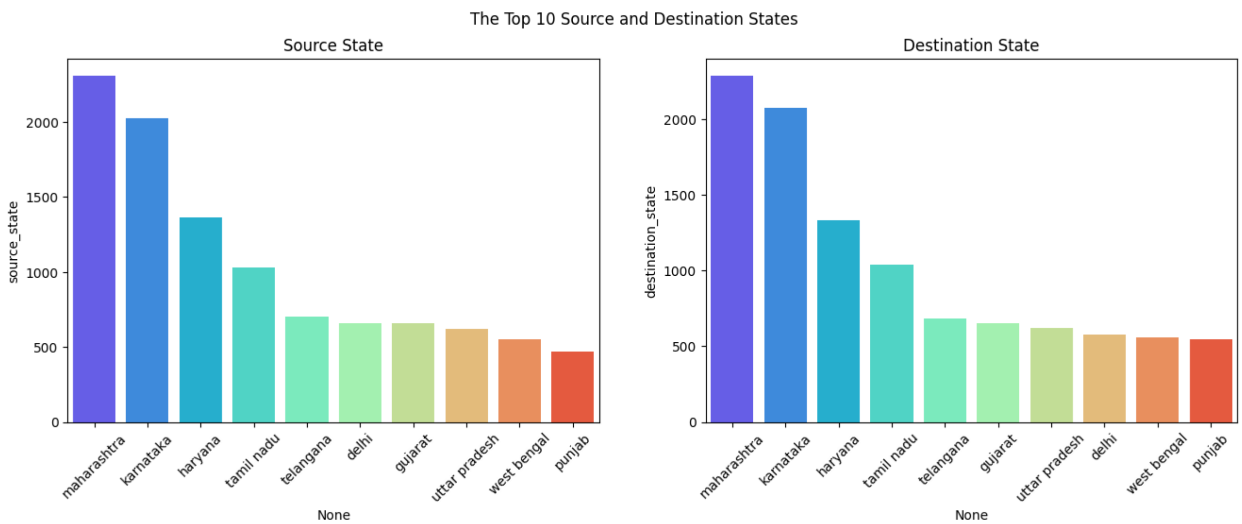
```
sourcestate10 = pd.DataFrame(trip["source_state"].value_counts()
[0:10])
destinationstate10 =
pd.DataFrame(trip["destination_state"].value_counts()[0:10])
destinationstate10
```

{"summary":"{\n \"name\": \"destinationstate10\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"destination_state\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 652,\n   \"min\": 549,\n   \"max\": 2285,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   559,\n   2070,\n   653\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"destinationstate10"}

```
fig, ax = plt.subplots(1,2,figsize=(16,5))

sns.barplot(x = sourcestate10.index, y =
sourcestate10['source_state'], data = sourcestate10, ax = ax[0],
palette = 'rainbow')
ax[0].set_xticklabels(sourcestate10.index,rotation=45)
ax[0].set_title("Source State")
```

```
sns.barplot(x = destinationstate10.index, y =
destinationstate10['destination_state'], data = destinationstate10, ax
= ax[1], palette = 'rainbow')
ax[1].set_xticklabels(destinationstate10.index,rotation=45)
ax[1].set_title("Destination State")

plt.suptitle("The Top 10 Source and Destination States")
plt.show()
```

```
<ipython-input-411-b88752d7a05e>:4: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[0].set_xticklabels(sourcestate10.index,rotation=45)
<ipython-input-411-b88752d7a05e>:9: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[1].set_xticklabels(destinationstate10.index,rotation=45)
```




```
'''
We see that the same 10 states are the top source and destination
states for the trips. Maharashtra
is the highest, followed by Karnataka, Haryana, Tamil Nadu and
Telengana.
'''
```

{"type":"string"}

```
trip.groupby(['source_state','destination_state'])
['actual_time'].sum().reset_index().sort_values(by='actual_time',
ascending=False)
```

{"summary":"{\n \"name\": \"trip\",\n \"rows\": 90,\n \"fields\":
[\n   {\n   \"column\": \"source_state\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
28,\n   \"samples\": [\n   \"rajasthan\",\n

\"nagaland\",\n   \"gujarat\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"destination_state\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 31,\n   \"samples\": [\n   \"dadra
and nagar haveli\",\n   \"delhi\",\n   \"arunachal
pradesh\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"actual_time\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 50565.26007875946,\n   \"min\":
43.0,\n   \"max\": 268491.0,\n   \"num_unique_values\": 89,\
n   \"samples\": [\n   3883.0,\n   2340.0,\n
8020.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
trip.groupby(['destination_state','source_state'])
['actual_distance_to_destination'].sum().reset_index().sort_values(by=
'actual_distance_to_destination',ascending=False)
```

{"summary":"{\n \"name\": \"trip\",\n \"rows\": 90,\n \"fields\":
[\n   {\n   \"column\": \"destination_state\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 31,\n   \"samples\": [\n
\"tripura\",\n   \"delhi\",\n   \"arunachal pradesh\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"source_state\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 28,\n   \"samples\": [\n
\"punjab\",\n   \"arunachal pradesh\",\n
\"rajasthan\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"actual_distance_to_destination\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 20210.283318207654,\n
\"min\": 9.376028394774304,\n   \"max\": 103882.58771894682,\n
\"num_unique_values\": 90,\n   \"samples\": [\n
2376.9867390626905,\n   6790.327264015463,\n
916.34430367597\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

###Trip Month

```
trip["trip_month"].value_counts()
```

```
9     11172
10     1551
Name: trip_month, dtype: int64
```

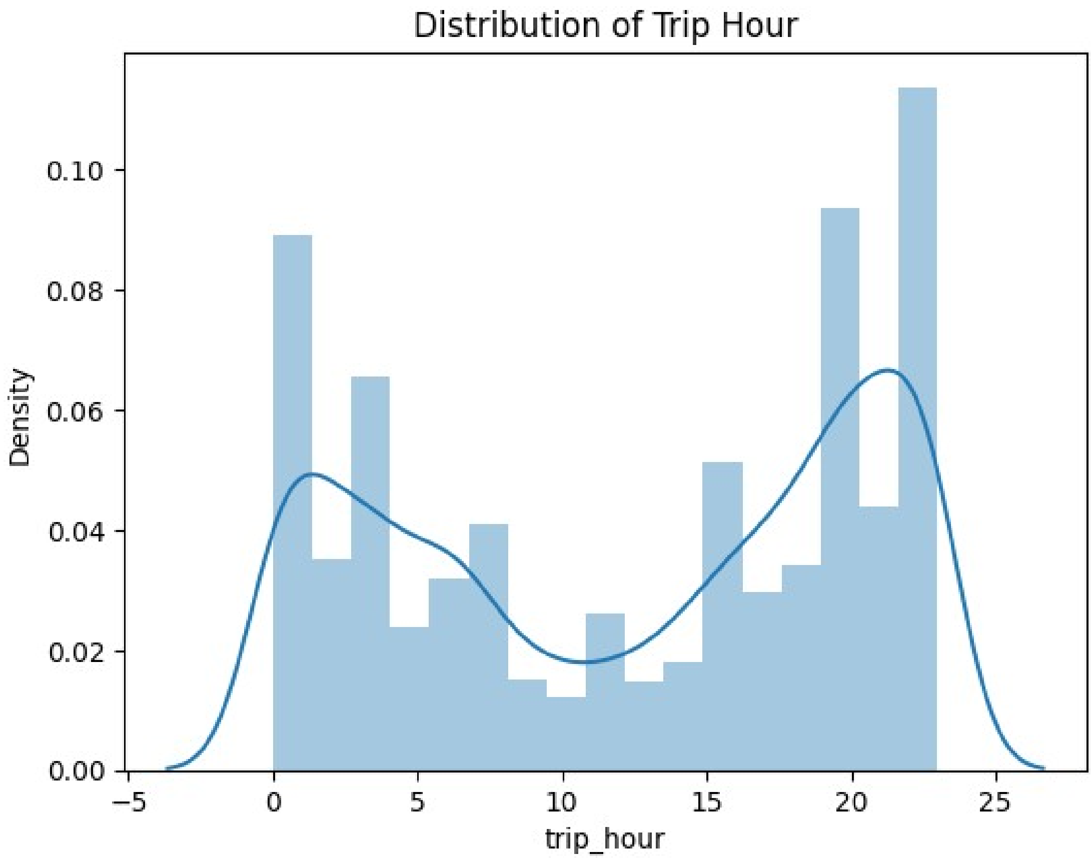
```
'''
The trips are recorded only for the months of September and October.
The recording perhaps
stopped after that. So we do not analyse further on the basis of
month.
'''
```

{"type":"string"}

###Trip Hour Distribution

```
sns.distplot(trip["trip_hour"])
plt.title("Distribution of Trip Hour")
plt.show()
```

```
<ipython-input-417-b93cb353a2ed>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["trip_hour"])
```



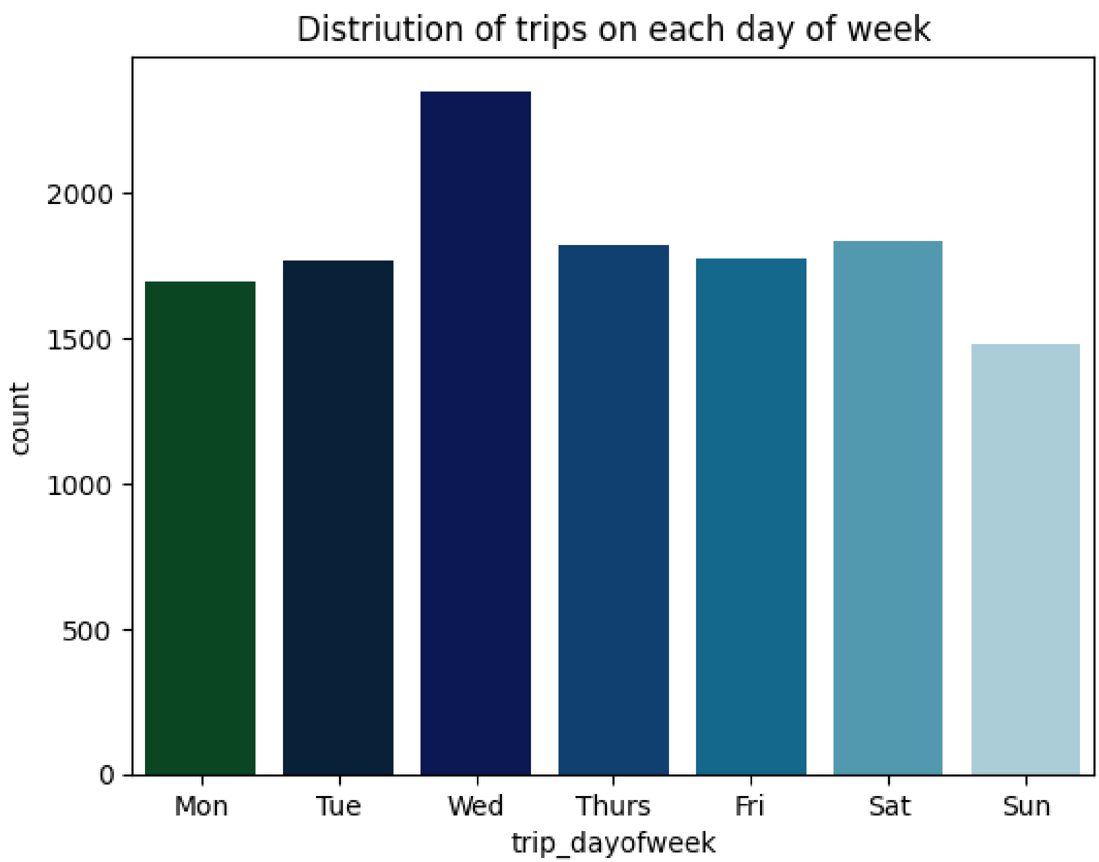

```
'''
So, we observe a kind of bimodal distribution with minimum trips
occuring during the day hours
(8 AM to 1 PM) and maximum occuring during late night or early morning
hours (8 PM to 2 AM).
'''

{"type":"string"}
```

###Trip Day of Week Distribution

```
trip["trip_dayofweek"] =
trip["trip_dayofweek"].map({0:'Mon',1:'Tue',2:'Wed',3:'Thurs',4:'Fri',
5:'Sat',6:'Sun'})
trip["trip_dayofweek"].value_counts()
```

Wed	2352
Sat	1836
Thurs	1819
Fri	1774
Tue	1766
Mon	1697

```
Sun      1479
Name: trip_dayofweek, dtype: int64

sns.countplot(x = 
"trip_dayofweek",data=trip,order=['Mon','Tue','Wed','Thurs','Fri','Sat
','Sun'], palette = 'ocean')
plt.title("Distriution of trips on each day of week")
plt.show()
```



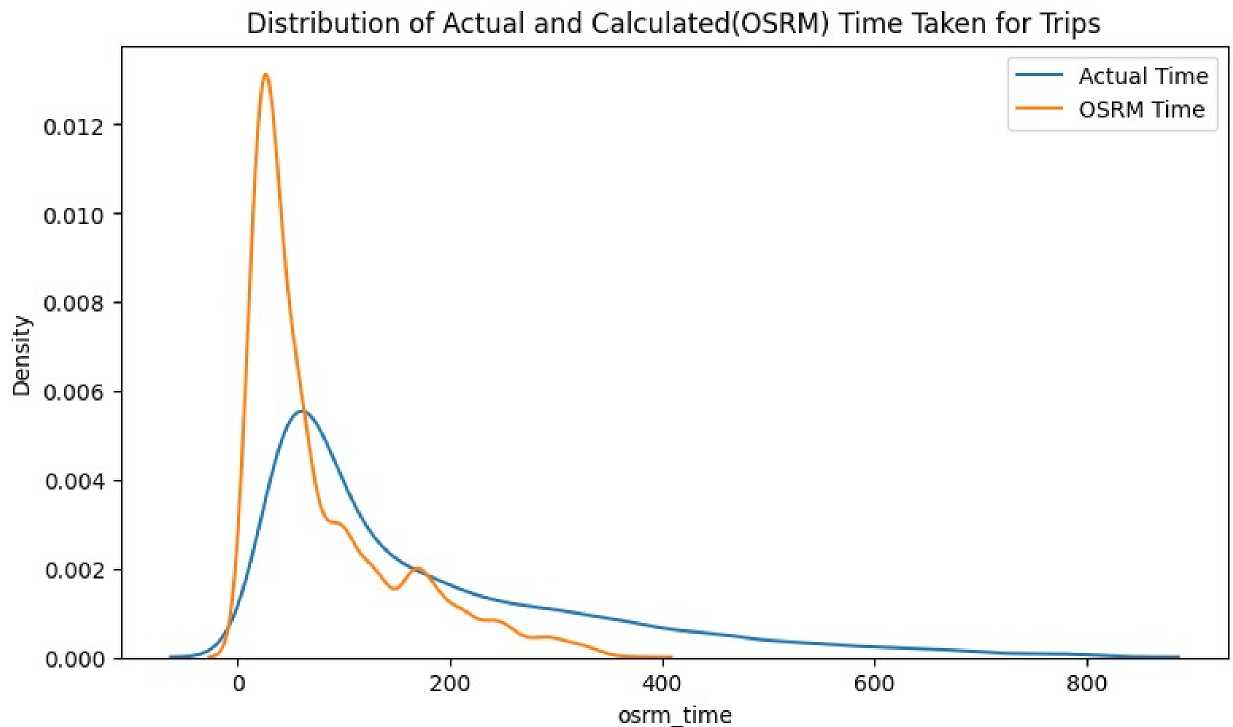

```
'''
So we see that maximum number of trips are happening on Wednesday and 
minimum on Sunday.
'''

{"type":"string"}
```

###Distribution of Actual and Calculated(OSRM) Time Taken for Trips

```
plt.figure(figsize=(9,5))
sns.distplot(trip["actual_time"], hist=False, label = "Actual Time")
sns.distplot(trip["osrm_time"], hist=False, label = "OSRM Time")
plt.legend()
```

```
plt.title("Distribution of Actual and Calculated(OSRM) Time Taken for 
Trips")
plt.show()
```

```
<ipython-input-422-37a2a6c52f92>:2: UserWarning: 

`distplot` is a deprecated function and will be removed in seaborn 
v0.14.0.

Please adapt your code to use either `displot` (a figure-level 
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel 
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["actual_time"], hist=False, label = "Actual Time")
<ipython-input-422-37a2a6c52f92>:3: UserWarning: 

`distplot` is a deprecated function and will be removed in seaborn 
v0.14.0.

Please adapt your code to use either `displot` (a figure-level 
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel 
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["osrm_time"], hist=False, label = "OSRM Time")
```

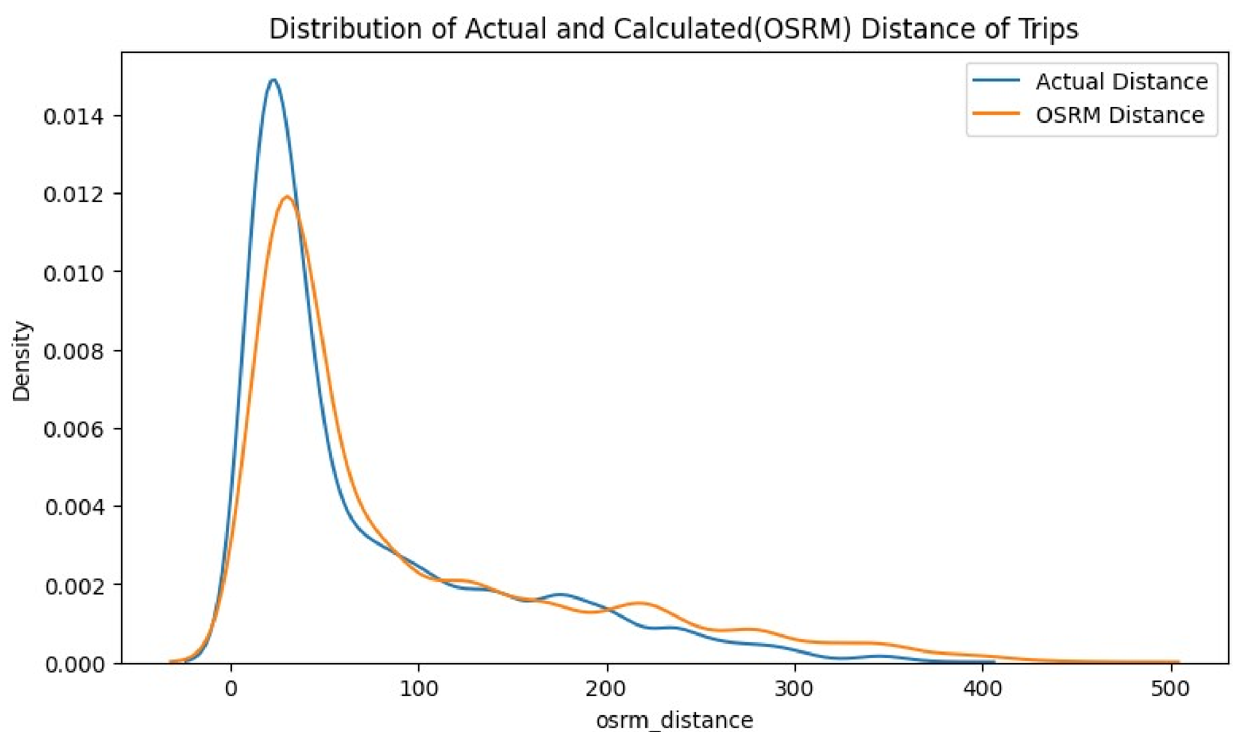



```
'''
So we see that actual time distribution has a kind of skewed
distribution. Also, OSRM seems to
be calculating time taken as less than what time it actually takes.
This might be because in actual
scenario, there might be delays caused by unprecedented traffic or
other delays.
'''

{"type":"string"}
```

###Distribution of Actual and Calculated(OSRM) Distance of Trips

```
plt.figure(figsize=(9,5))
sns.distplot(trip["actual_distance_to_destination"], hist=False, label
= "Actual Distance")
sns.distplot(trip["osrm_distance"], hist=False, label = "OSRM
Distance")
plt.legend()
plt.title("Distribution of Actual and Calculated(OSRM) Distance of
Trips")
plt.show()

<ipython-input-424-64e3c9b3f91b>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.
```

```
Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["actual_distance_to_destination"], hist=False,
label = "Actual Distance")
<ipython-input-424-64e3c9b3f91b>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["osrm_distance"], hist=False, label = "OSRM
Distance")
```



```
'''
As we can see, the distributions are similar, however, OSRM distance
has greater spread than actual
(which means distance covered actually is on the lower side as
compared to OSRM calculated).
'''
```

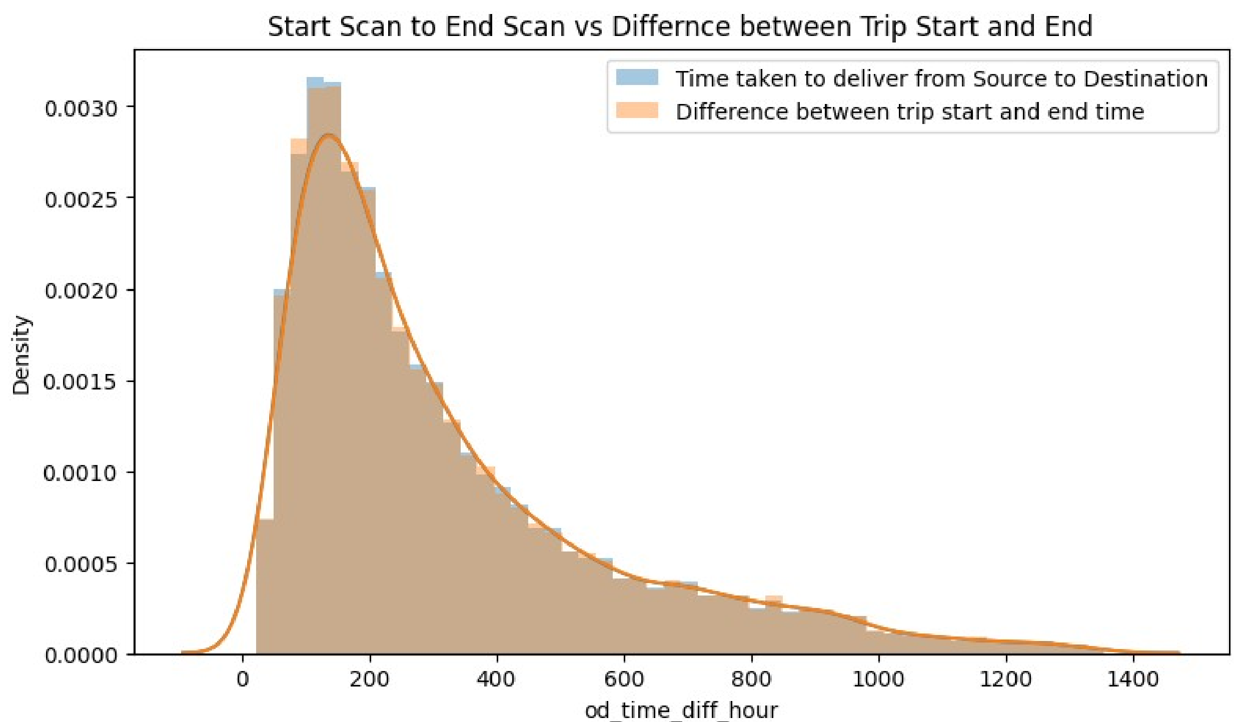
{"type":"string"}

###Start Scan to End Scan vs Differnce between Trip Start and End

```
plt.figure(figsize=(9,5))
sns.distplot(trip["start_scan_to_end_scan"], label = "Time taken to
deliver from Source to Destination")
sns.distplot(trip["od_time_diff_hour"], label = "Difference between
trip start and end time")
plt.legend()
plt.title("Start Scan to End Scan vs Differnce between Trip Start and
End")
plt.show()
```

```
<ipython-input-426-5bf679990b4d>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["start_scan_to_end_scan"], label = "Time taken to
deliver from Source to Destination")
<ipython-input-426-5bf679990b4d>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
sns.distplot(trip["od_time_diff_hour"], label = "Difference between trip start and end time")
```




```
'''
There is not much difference between the above two variables.
'''
```

{"type":"string"}

##Bivariate Analysis

```
trip.head(1)
```

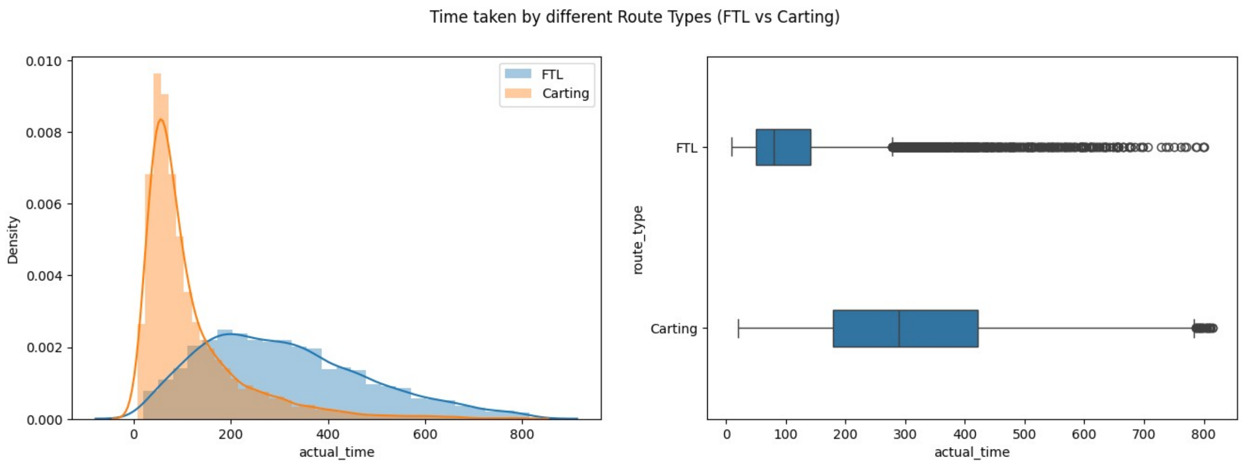
{"type":"dataframe","variable_name":"trip"}

###Does the distribution of time taken depend on the route type (carting vs full truck load) ?

```
fig, ax = plt.subplots(1,2,figsize=(16,5))

sns.distplot(trip[trip["route_type_FTL"]==1]["actual_time"], label = "FTL", ax = ax[0])
sns.distplot(trip[trip["route_type_Carting"]==1]["actual_time"], label = "Carting", ax = ax[0])

sns.boxplot(x = "actual_time", y = "route_type", data = trip, orient='h', width=0.2, ax=ax[1])
```

```
ax[0].legend()
ax[1].set_yticklabels(["FTL","Carting"])
plt.suptitle("Time taken by different Route Types (FTL vs Carting)")
plt.show()
```

```
<ipython-input-429-1e793e207ecc>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_FTL"]==1]["actual_time"], label =
"FTL", ax = ax[0])
<ipython-input-429-1e793e207ecc>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_Carting"]==1]["actual_time"],
label = "Carting", ax = ax[0])
<ipython-input-429-1e793e207ecc>:9: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[1].set_yticklabels(["FTL","Carting"])
```

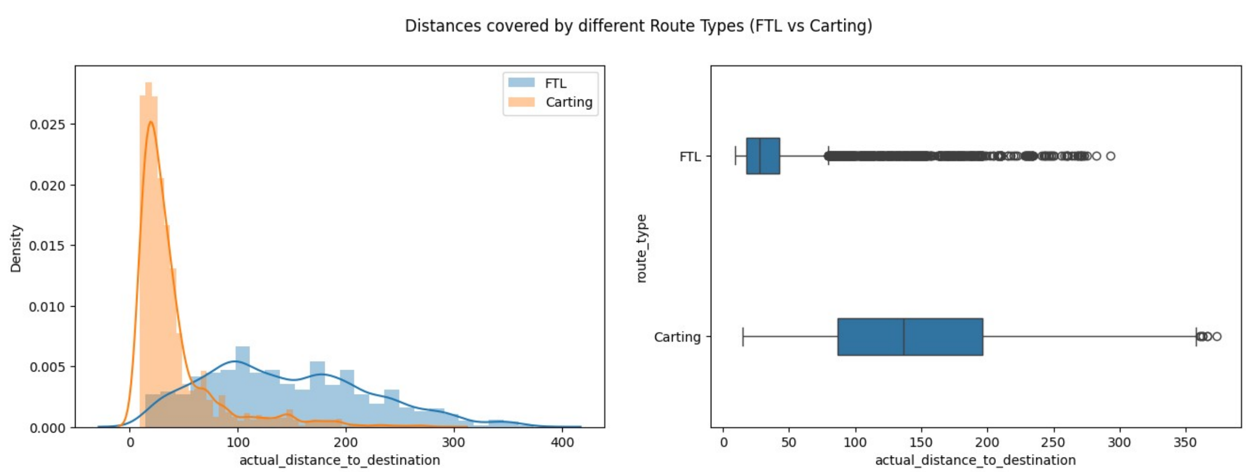



```
'''
So we see that the time taken by full truck load deliveries is on
average, a lot higher (>300 hours)
(probably because the distance covered by trucks is also much higher
since they don’t make stops)
than the cart deliveries (<100 hours).
'''

{"type":"string"}
```

###Does the distribution of distance covered depend on the route type (carting vs full truck load) ?

```
fig, ax = plt.subplots(1,2,figsize=(16,5))

sns.distplot(trip[trip["route_type_FTL"]==1]
["actual_distance_to_destination"], label = "FTL", ax = ax[0])
sns.distplot(trip[trip["route_type_Carting"]==1]
["actual_distance_to_destination"], label = "Carting", ax = ax[0])

sns.boxplot(x = "actual_distance_to_destination", y = "route_type",
data = trip, orient='h', width=0.2, ax=ax[1])

ax[0].legend()
ax[1].set_yticklabels(["FTL","Carting"])

plt.suptitle("Distances covered by different Route Types (FTL vs
Carting)")
plt.show()

<ipython-input-431-77c7ae2397e8>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
```

```
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_FTL"]==1]
["actual_distance_to_destination"], label = "FTL", ax = ax[0])
<ipython-input-431-77c7ae2397e8>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_Carting"]==1]
["actual_distance_to_destination"], label = "Carting", ax = ax[0])
<ipython-input-431-77c7ae2397e8>:9: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax[1].set_yticklabels(["FTL","Carting"])
```

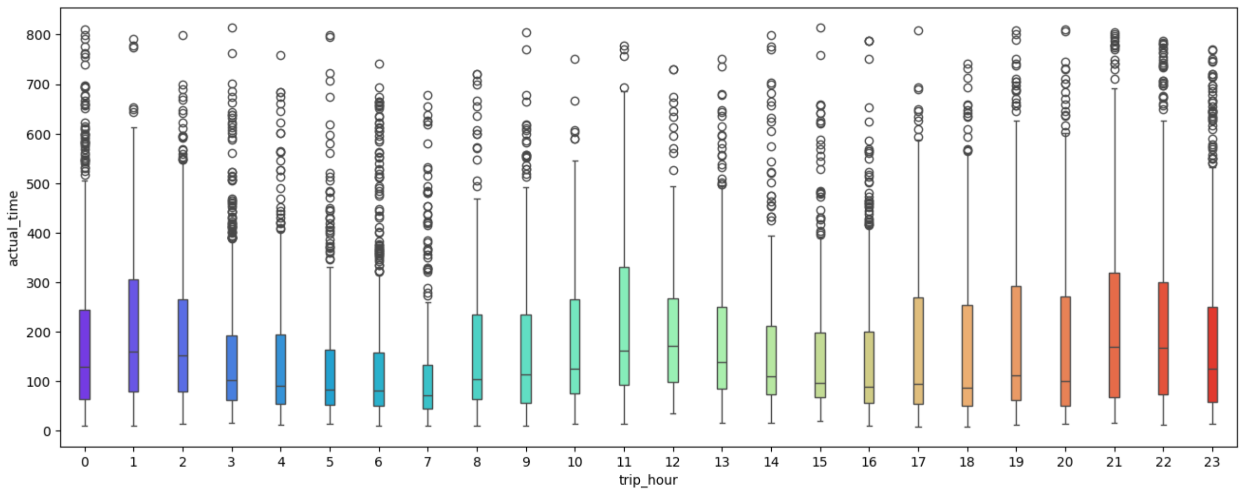



```
'''
So our initial assumption is correct. The full truck load deliveries
cover much longer distances on
average (>150 kms) than carting deliveries (~ 25 kms).
'''
```

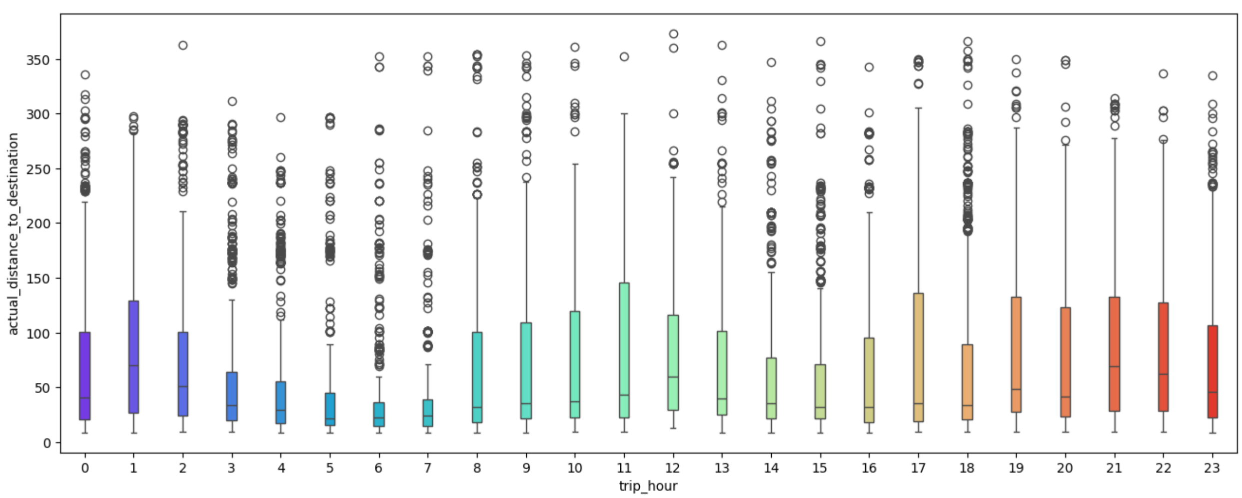
{"type":"string"}

###Distribution of time taken and distance covered by deliveries depending on the hour of the day

```
plt.figure(figsize=(16,6))
sns.boxplot(x = "trip_hour", y = "actual_time", data = trip, width=0.2, palette = 'rainbow')
plt.show()
```




```
plt.figure(figsize=(16,6))
sns.boxplot(x = "trip_hour", y = "actual_distance_to_destination", data = trip, width=0.2, palette = 'rainbow')
plt.show()
```



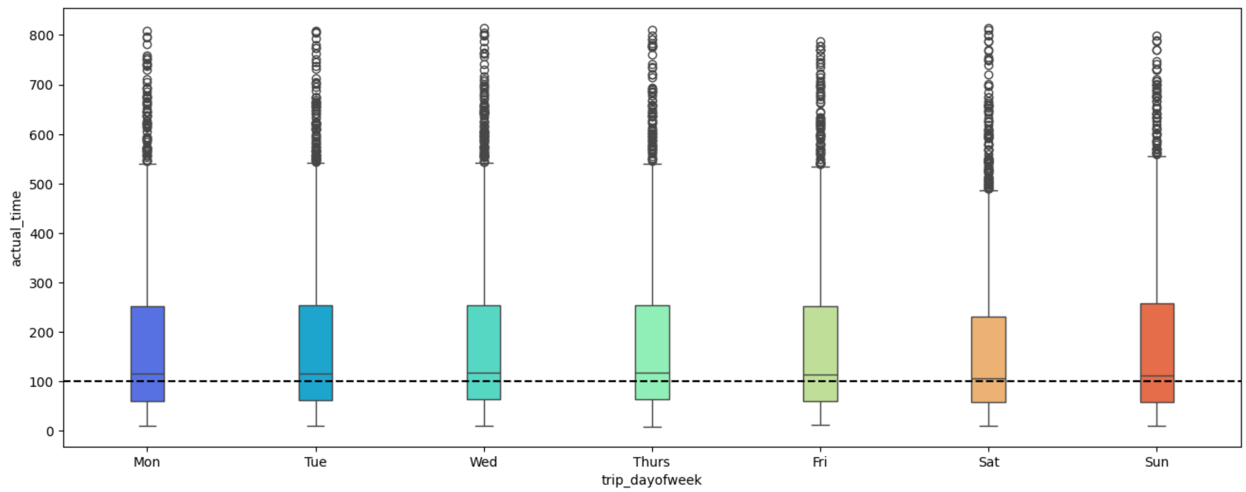

```
'''
Time and distances follow similar trends against the hour of the day.
Maximum time and distance
deliveries are likely to be made during peak morning hours of 10 AM to 12 PM as well as 5 PM, 7
```

```
PM and 1 AM
'''
```

```
{"type":"string"}
```

###Distribution of time taken and distance covered by deliveries depending on the day of the week

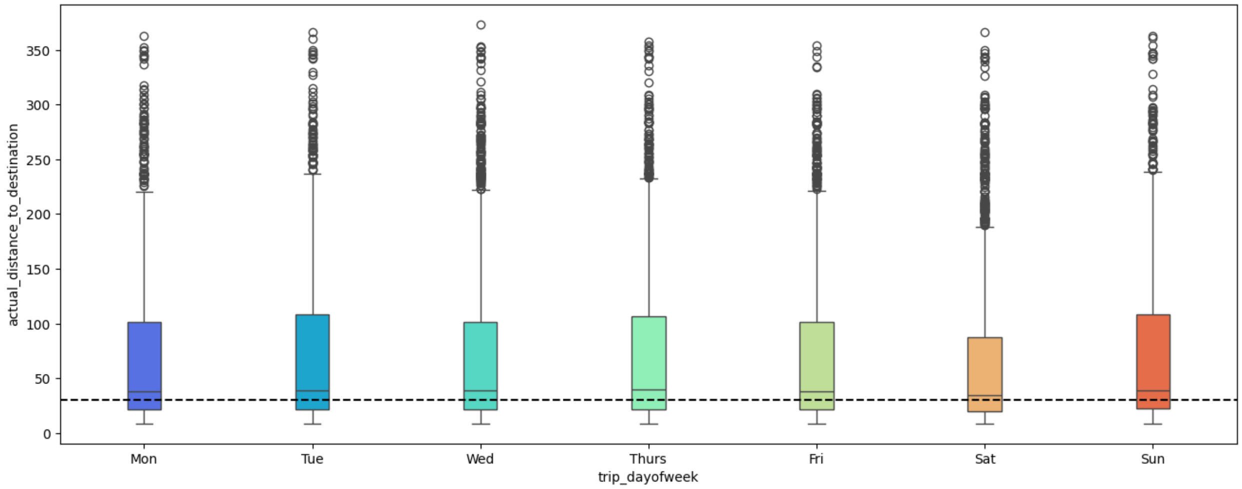
```
plt.figure(figsize=(16,6))
sns.boxplot(x = "trip_dayofweek", y = "actual_time", data = trip,
width=0.2, order = ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun'],
palette = 'rainbow')

plt.axhline(y=100, color='k', ls = '--')
plt.show()
```




```
'''
On average, time taken is slightly more on weekdays and Sunday as
compared to Saturday. However,
they are very similar.
'''
```

```
{"type":"string"}
```

```
plt.figure(figsize=(16,6))
sns.boxplot(x = "trip_dayofweek", y =
"actual_distance_to_destination", data = trip, width=0.2, order =
['Mon','Tue','Wed','Thurs','Fri','Sat','Sun'], palette = 'rainbow')
plt.axhline(y=30, color='k', ls = '--')
plt.show()
```




```
'''
Distance covered is also lowest on Saturday
'''
```

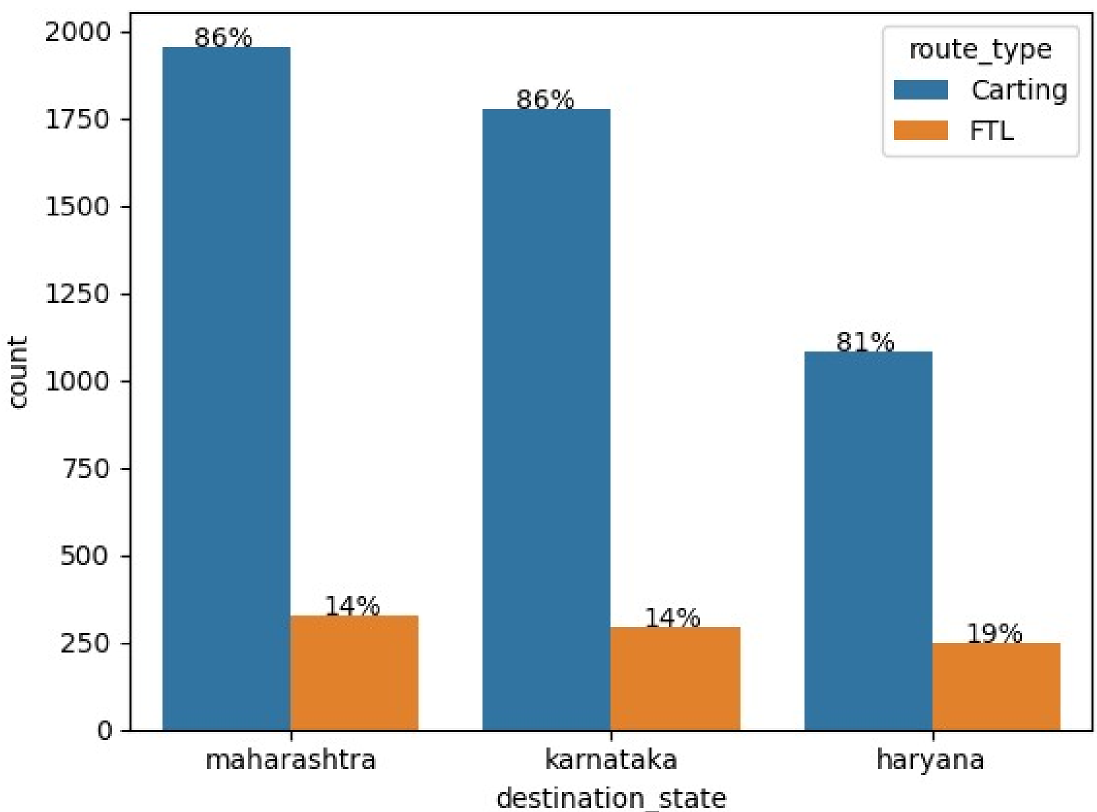
{"type":"string"}

###Route Type Distributions for Top 3 States

####Destination States

```
top3d = trip[(trip["destination_state"]=='maharashtra')|
(trip["destination_state"]=='karnataka')|
(trip["destination_state"]=='haryana')]

top3d = top3d[['route_type','destination_state']]


#top3d['route_type'] = top3d['route_type'].map({0:'FTL',1:'Carting'})

st = ['maharashtra','karnataka','haryana']

g = sns.countplot(x='destination_state',hue='route_type', data=top3d,
order = st)

percx = []

for e in st:

percx.append(top3d[(top3d['destination_state']==e)&(top3d["route_type"
]=="Carting")].shape[0]/top3d[top3d['destination_state']==e].shape[0])

for e in st:

percx.append(top3d[(top3d['destination_state']==e)&(top3d["route_type"
]=="FTL")].shape[0]/top3d[top3d['destination_state']==e].shape[0])
```

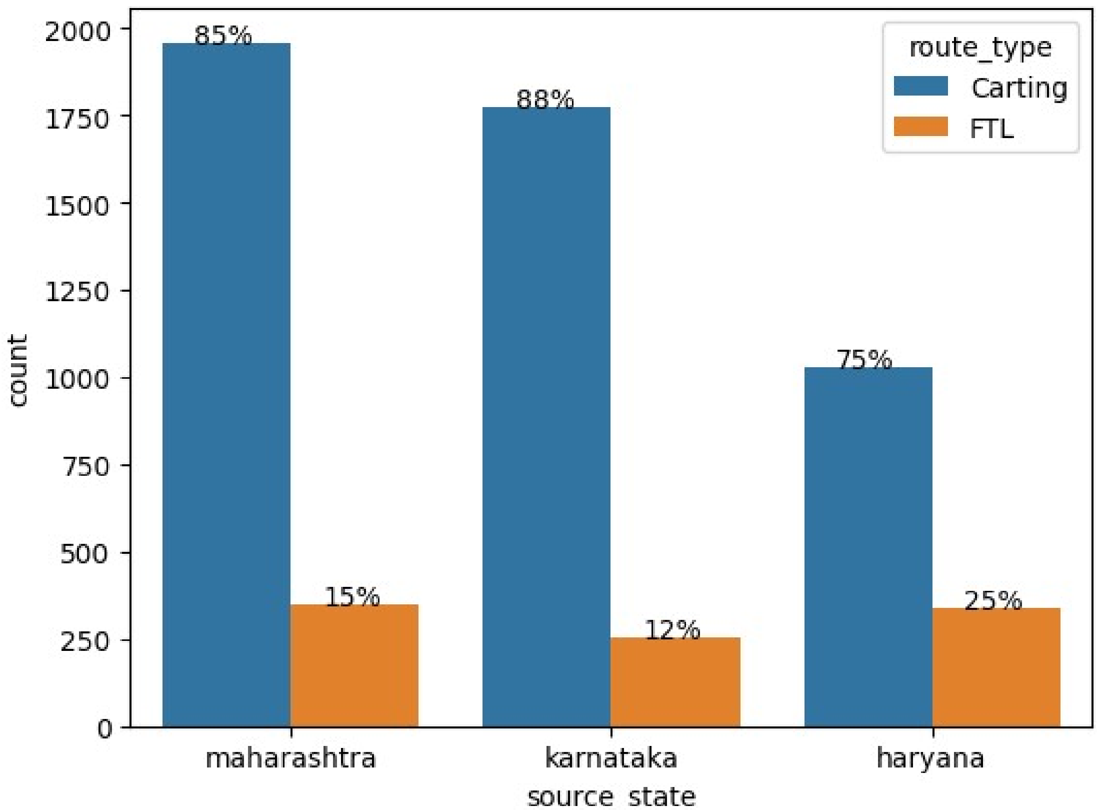
```
i=0
for p in g.patches:
  if i < len(percx):
    txt = str((round(percx[i]*100))) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.text(txt_x+0.1,txt_y,txt)
    i+=1

plt.show()
```




```
'''
So we see that for top 3 destination states, Maharashtra hs 86%
Carting and 14% FTL, Karnataka
has 86% Carting and 14% FTL, Haryana has 81% Carting and 19% FTL.
'''

{"type":"string"}
```

####Source States

```
top3d = trip[(trip["source_state"]=='maharashtra')|
(trip["source_state"]=='karnataka')|(trip["source_state"]=='haryana')]

top3d = top3d[['route_type','source_state']]


#top3d['route_type'] = top3d['route_type'].map({0:'FTL',1:'Carting'})

st = ['maharashtra','karnataka','haryana']

g = sns.countplot(x='source_state',hue='route_type', data=top3d, order
= st)

percx = []

for e in st:

percx.append(top3d[(top3d['source_state']==e)&(top3d["route_type"]=="C
arting")].shape[0]/top3d[top3d['source_state']==e].shape[0])

for e in st:

percx.append(top3d[(top3d['source_state']==e)&(top3d["route_type"]=="F
TL")].shape[0]/top3d[top3d['source_state']==e].shape[0])

i=0
for p in g.patches:
  if i < len(percx):
    txt = str((round(percx[i]*100))) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.text(txt_x+0.1,txt_y,txt)
    i+=1

plt.show()
```




```
'''
So we see that for top 3 source states, Maharashtra hs 85% Carting and
15% FTL, Karnataka has
88% Carting and 12% FTL, Haryana has 75% Carting and 25% FTL.
'''
```

{"type":"string"}

##Hypothesis Testing

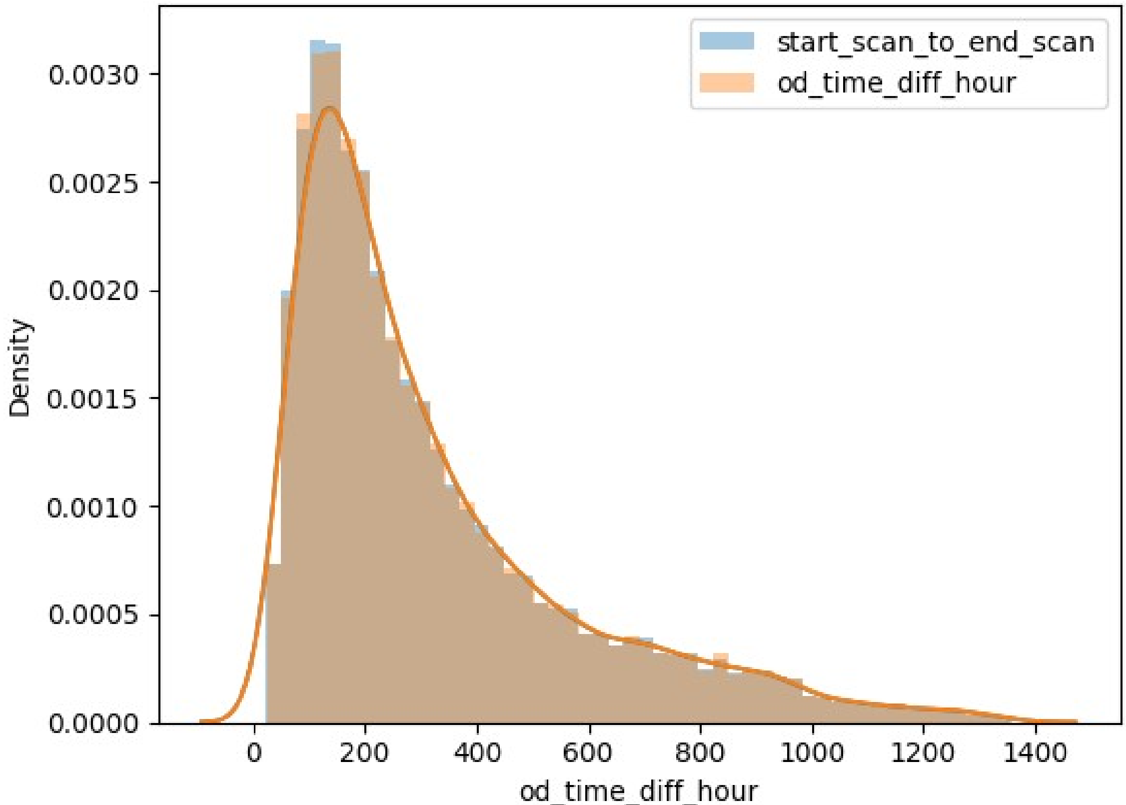
start_scan_to_end_scan v/s od_time_diff_hour

```
'''

H0 : The means of both the groups are equal.
H1: The means are not equal.
α = 0.05

'''
```

{"type":"string"}

```
sns.distplot(trip["start_scan_to_end_scan"],
label="start_scan_to_end_scan")
sns.distplot(trip["od_time_diff_hour"], label="od_time_diff_hour")
```

```
plt.legend()
plt.show()

<ipython-input-445-aa989f9ecf5c>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["start_scan_to_end_scan"],
label="start_scan_to_end_scan")
<ipython-input-445-aa989f9ecf5c>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["od_time_diff_hour"], label="od_time_diff_hour")
```



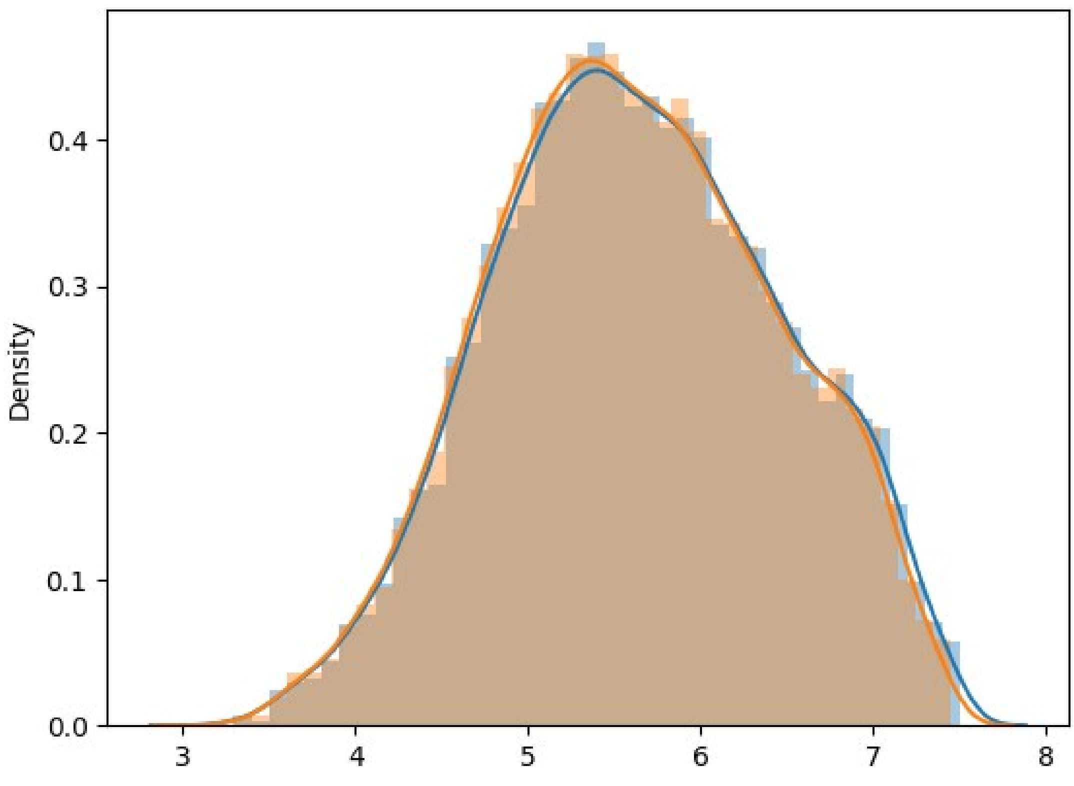

```
'''
From the above plot, the means indeed appear to be the same. We will
perform 2-sample t-test to
find out. But first we shall convert our data to a normal distribution
using boxcox transformation
'''
```

{"type":"string"}

```
from scipy.stats import boxcox
x_trf1 , lambda1 = boxcox(trip["start_scan_to_end_scan"])
x_trf2 , lambda2 = boxcox(trip["od_time_diff_hour"])
sns.distplot(x_trf1)
sns.distplot(x_trf2)
plt.show()
```

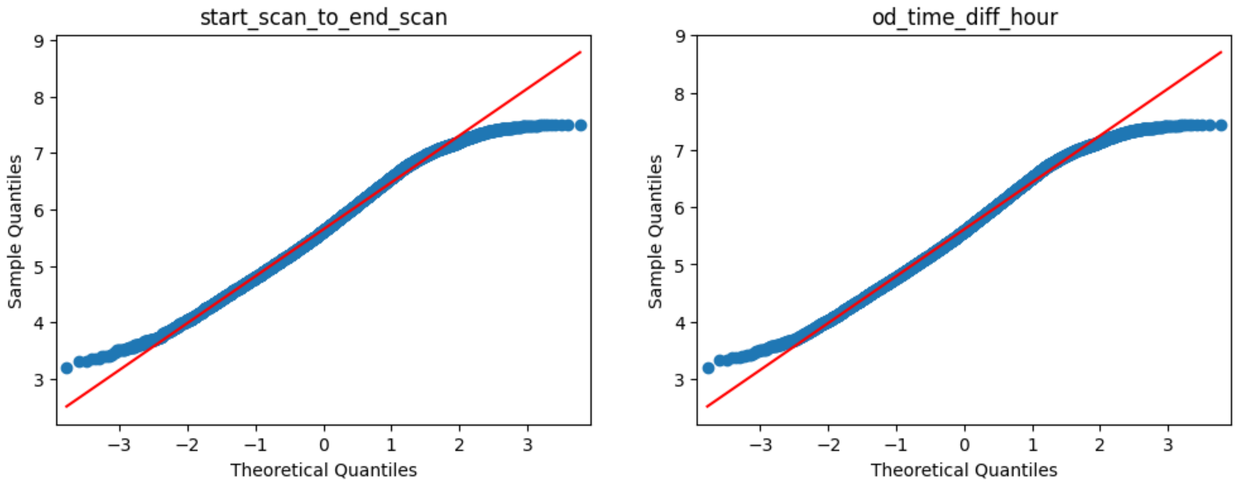
```
<ipython-input-447-8bba7d866cff>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).
```

```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(x_trf1)
<ipython-input-447-8bba7d866cff>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(x_trf2)
```




```python
import statsmodels.api as sms

fig, ax = plt.subplots(1,2,figsize=(12,4))

sms.qqplot(x_trf1, line='s', ax = ax[0])
```

```python
sms.qqplot(x_trf2, line='s', label='od_time_diff_hour', ax = ax[1])

ax[0].set_title('start_scan_to_end_scan')
ax[1].set_title('od_time_diff_hour')

plt.show()
```




```python
'''
Since our data is not normal even after trying BoxCox transform, we
perform a non-parametric
test (Mann-Whitney). Now our H0 and H1 become :

H0 : The median of both groups are equal.
H1: The median are not equal.
'''
```

{"type":"string"}

```python
from scipy.stats import mannwhitneyu

test_stat, p_val =
mannwhitneyu(trip["start_scan_to_end_scan"].sample(1000),
trip["od_time_diff_hour"].sample(1000))
test_stat, p_val
```

```
(505912.0, 0.6471042714854605)
```

```python
'''
Since p > α (0.05), we fail to reject the null hypothesis.
Hence, the sample distributions seem to be the same for
‘start_scan_to_end_scan’ and ‘od_time_diff_hour’.
So, the trip duration and the difference between trip start and end
are indeed the same.
'''
```

{"type":"string"}

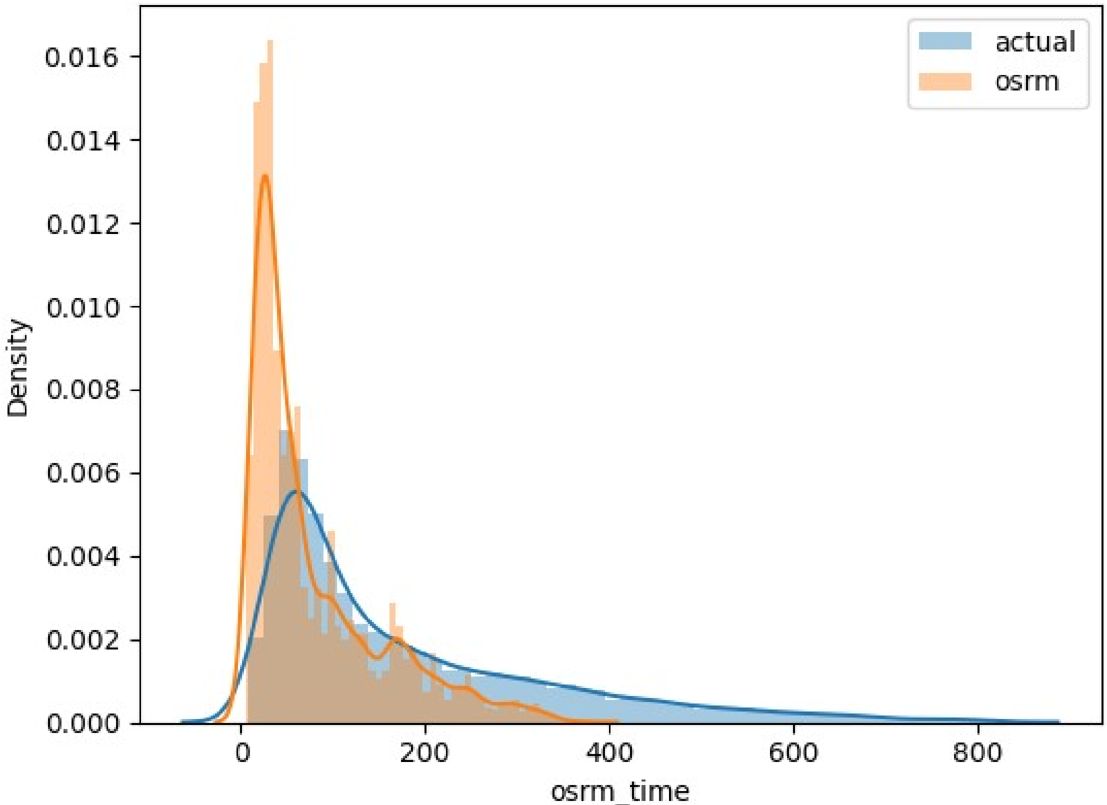
###actual_time v/s osrm_time

```
'''
H0 : The means of actual_time and calculated(osrm) time are equal.
H1: The means are not equal.
α = 0.05
'''
```

{"type":"string"}

```
sns.distplot(trip["actual_time"], label="actual")
sns.distplot(trip["osrm_time"], label="osrm")
plt.legend()
plt.show()
```

```
<ipython-input-453-42bf59f3709e>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["actual_time"], label="actual")
<ipython-input-453-42bf59f3709e>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["osrm_time"], label="osrm")
```




```
'''
From the plot above, it is clear that these do not follow normal
distribution. So we go for the
non-parametric Mann-Whitney test. Now our H0 and H1 become :

H0 : The median of both groups are equal.
H1: The median are not equal.
'''

{"type":"string"}

test_stat, p_val = mannwhitneyu(trip["actual_time"].sample(1000),
trip["osrm_time"].sample(1000))
test_stat, p_val

(734070.5, 1.9543151365791207e-73)

'''
Since p < α (0.05), we reject the null hypothesis.
Hence, the sample distributions are different.
So,
the actual time and the time calculated by algorithm (osrm) are very
different.
'''
```

{"type":"string"}

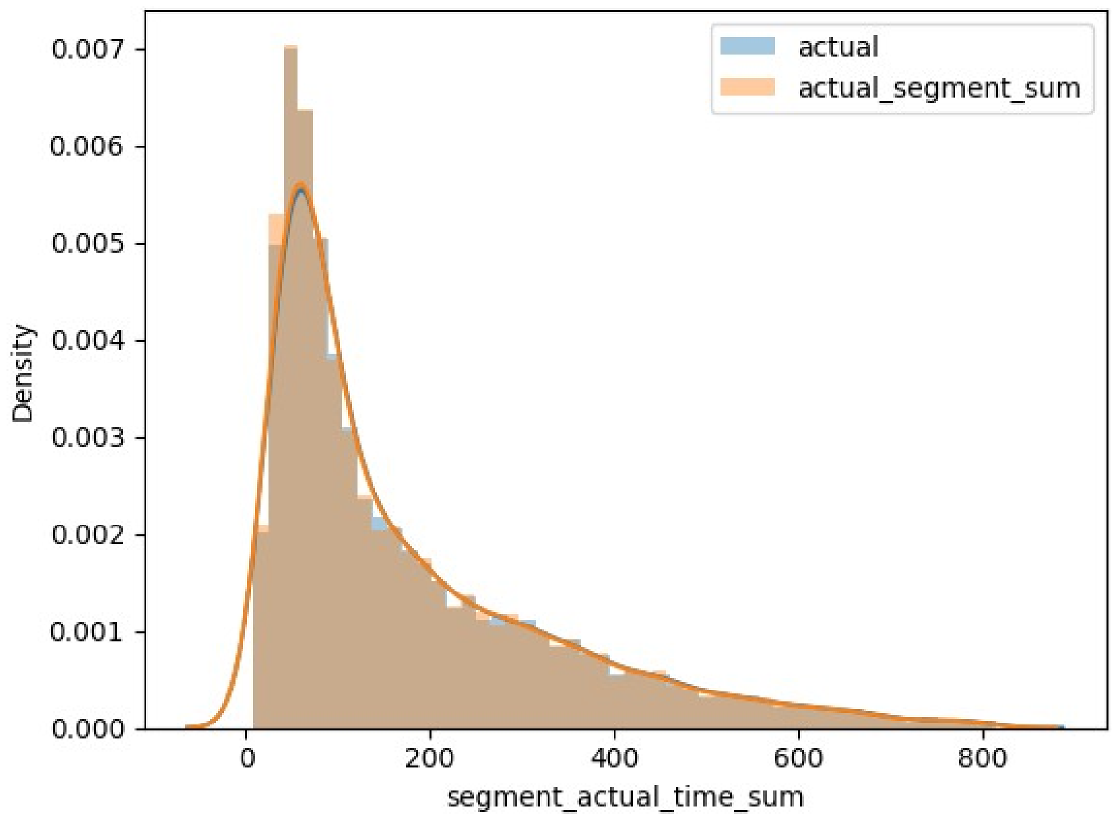
###actual_time v/s segment_actual_time

```
'''
H0 : The mean of actual_time and segment_actual_time are equal.
H1: The mean are not equal.
α = 0.05
'''
```

{"type":"string"}

```
sns.distplot(trip["actual_time"], label="actual")
sns.distplot(trip["segment_actual_time_sum"],
label="actual_segment_sum")
plt.legend()
plt.show()
```

```
<ipython-input-458-12e5f837a4c5>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["actual_time"], label="actual")
<ipython-input-458-12e5f837a4c5>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["segment_actual_time_sum"],
label="actual_segment_sum")
```




```
'''
Once again, we see that the times are not normally distributed.
However, the distributions look
similar. We use the non parametric Mann-Whitney test again. Now our H0
and H1 become :
H0 : The median of both groups are equal.
H1: The median are not equal.
'''

{"type":"string"}

test_stat, p_val = mannwhitneyu(trip["actual_time"].sample(1000),
trip["segment_actual_time_sum"].sample(1000))
test_stat, p_val

(499727.0, 0.9831637175913547)

'''
Since p > α (0.05), we fail to reject the null hypothesis.
Hence, the sample distributions seem to be the same for
‘segment_actual_time’ and ‘actual_time’.
So, the actual total time taken for a
trip is similar to the sum of the distances of a trip’s segments.
'''
```

{"type":"string"}

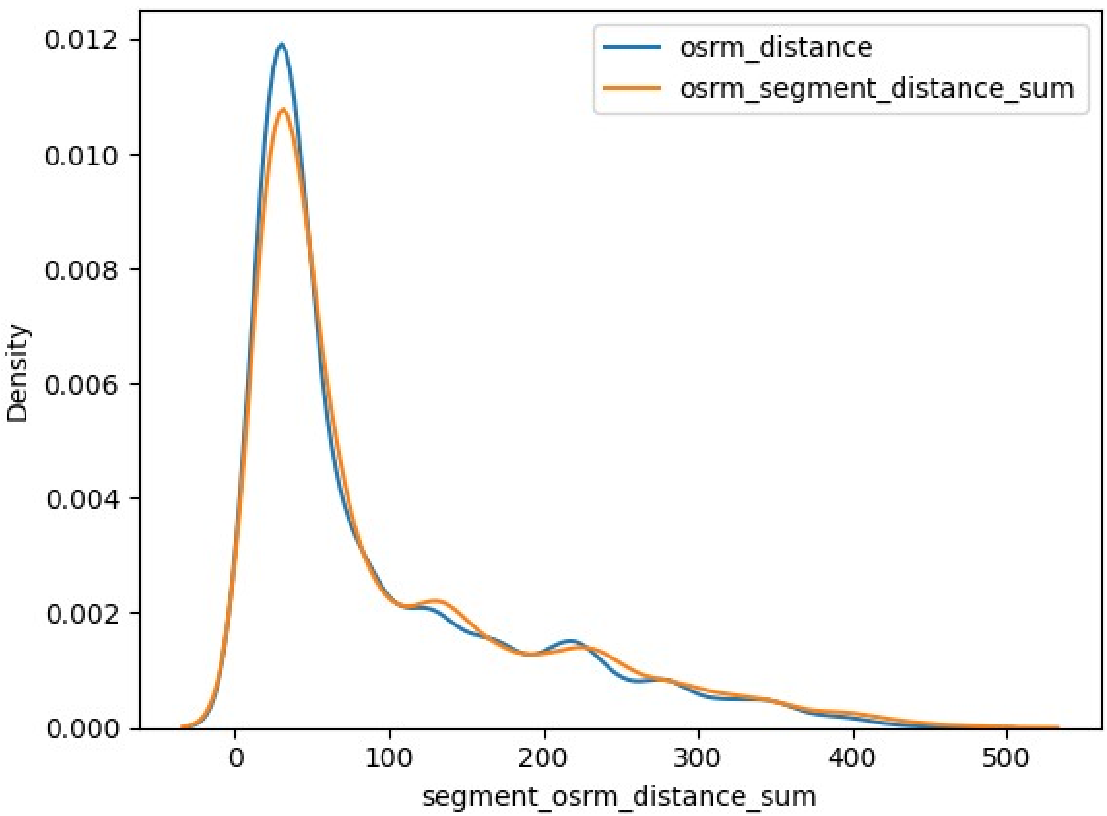
###osrm_distance v/s segment_osrm_distance_sum

```
'''
H0 : The mean of osrm_distance and segment_osrm_distance_sum are
equal.
H1: The mean are not equal.

α = 0.05
'''
```

{"type":"string"}

```
sns.distplot(trip["osrm_distance"], hist=False, label="osrm_distance")
sns.distplot(trip["segment_osrm_distance_sum"], hist=False,
label="osrm_segment_distance_sum")
plt.legend()
plt.show()
```

```
<ipython-input-463-1559da67a5a3>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["osrm_distance"], hist=False,
label="osrm_distance")
<ipython-input-463-1559da67a5a3>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["segment_osrm_distance_sum"], hist=False,
label="osrm_segment_distance_sum")
```




```
'''
Once again, these distributions look slightly different. Also, they
are not normal.
We use the non-paramteric Mann-Whitney test.

Now our H0 and H1 become :
H0 : The median of both groups are equal.
H1: The median are not equal.

'''
```

{"type":"string"}

```
test_stat, p_val = mannwhitneyu(trip["osrm_distance"].sample(1000),
trip["segment_osrm_distance_sum"].sample(1000))
test_stat, p_val
```

```
(467816.5, 0.01269325488736186)
```

```
'''
Since p > alpha (0.05), we fail to reject the null hypothesis.
Hence, the sample distributions are same.
So,
the overall distance calculated by osrm
and the sum of individual segment distances calculated by osrm are
```

```
same.
'''
```

{"type":"string"}

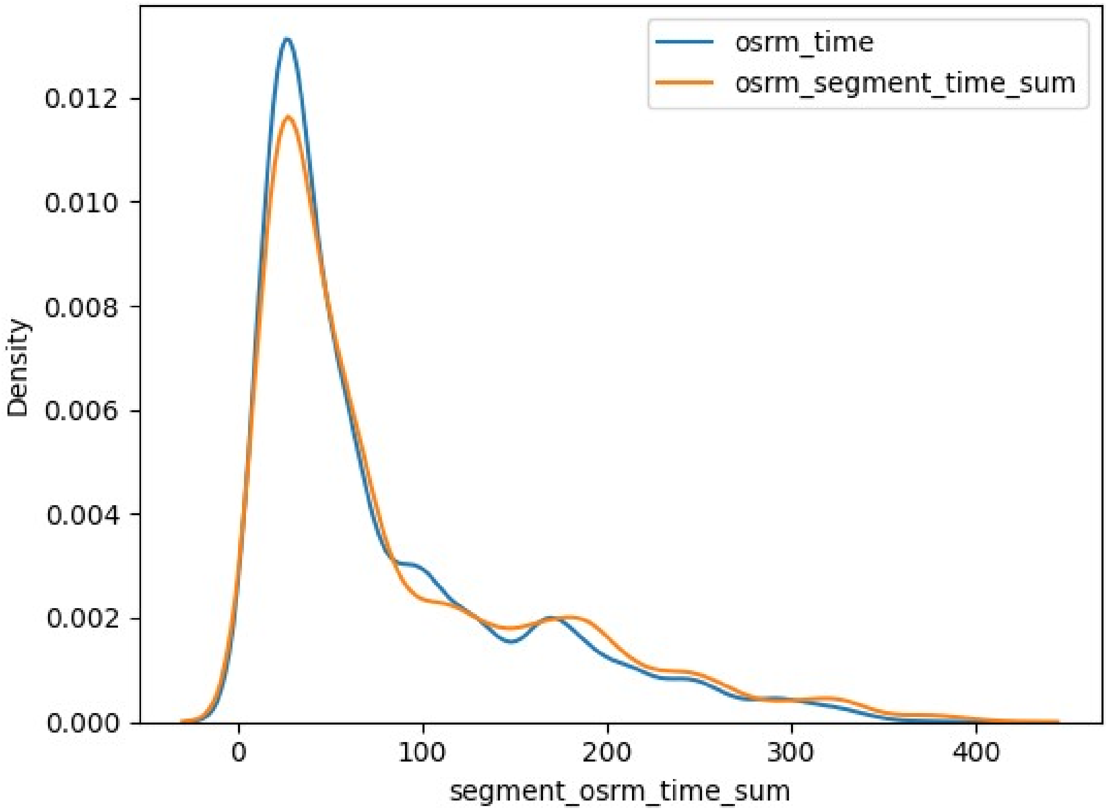
###osrm_time v/s segment_osrm_time_sum

```
'''
H0 : The mean of osrm_time and segment_osrm_time_sum are equal.
H1: The mean are not equal.
α = 0.05
'''
```

{"type":"string"}

```
sns.distplot(trip["osrm_time"], hist=False, label="osrm_time")
sns.distplot(trip["segment_osrm_time_sum"],
hist=False,label="osrm_segment_time_sum")
plt.legend()
plt.show()
```

```
<ipython-input-468-662044244280>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["osrm_time"], hist=False, label="osrm_time")
<ipython-input-468-662044244280>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip["segment_osrm_time_sum"],
hist=False,label="osrm_segment_time_sum")
```




```
'''
The distributions look slightly different and are not normal. We use
non parametric Mann-Whitney
test. Now our H0 and H1 become :
H0 : The median of both groups are equal.
H1: The median are not equal.
'''
```

{"type":"string"}

```
test_stat, p_val = mannwhitneyu(trip["osrm_time"].sample(1000),
trip["segment_osrm_time_sum"].sample(1000))
test_stat, p_val
```

```
(506140.0, 0.634464392554718)
```

```
'''
Since p > alpha (0.05), we fail to reject the null hypothesis.
Hence, the sample distributions are same.
So, the overall time calculated by osrm and the sum of individual
segment time caluclated by osrm are
same.
'''
```

{"type":"string"}

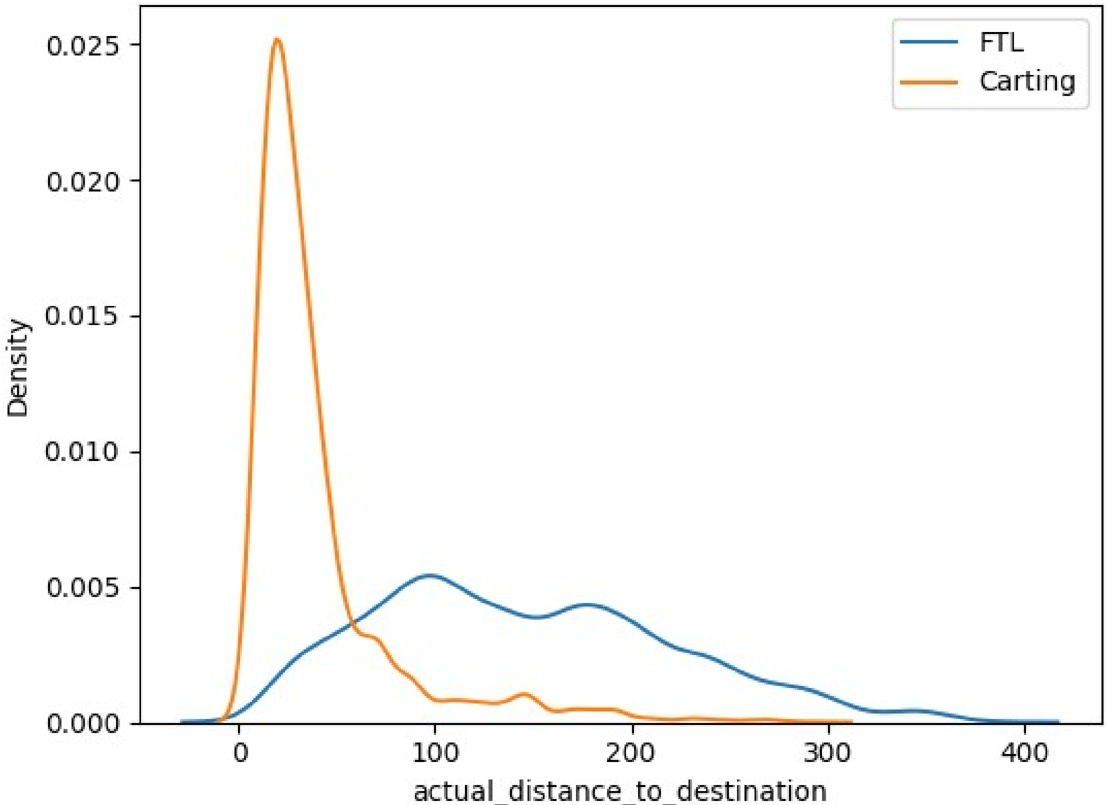
###actual_distance_to_destination for FTL vs Carting

```
'''
H0: The median distance of FTL and Carting is same.
H1: The median are different.
'''
```

{"type":"string"}

```
sns.distplot(trip[trip["route_type_FTL"]==1]
["actual_distance_to_destination"], hist=False, label="FTL")
sns.distplot(trip[trip["route_type_Carting"]==1]
["actual_distance_to_destination"], hist=False, label="Carting")
plt.legend()
plt.show()
```

```
<ipython-input-473-deae156fed06>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_FTL"]==1]
["actual_distance_to_destination"], hist=False, label="FTL")
<ipython-input-473-deae156fed06>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["route_type_Carting"]==1]
["actual_distance_to_destination"], hist=False, label="Carting")
```




```
'''
The ditributions are clearly different and not normal. So we use non
paramteric Mann-Whitney
test.
'''
```

{"type":"string"}

```
test_stat, p_val = mannwhitneyu(trip[trip["route_type_FTL"]==1]
["actual_distance_to_destination"].sample(1000),
trip[trip["route_type_Carting"]==1]
["actual_distance_to_destination"].sample(1000))
test_stat, p_val
```

```
(913892.0, 2.0664600733440896e-225)
```

```
'''
Since p < ff (0.05), we reject the null hypothesis. Hence, the sample
distributions are different. So,
the actual distances covered for FTL routes is different from that of
Carting routes.
'''
```

{"type":"string"}

###Does time taken depend on day of week?

```
'''
H0: The median time taken of all week days are same.
H1: The median are different.
'''
```

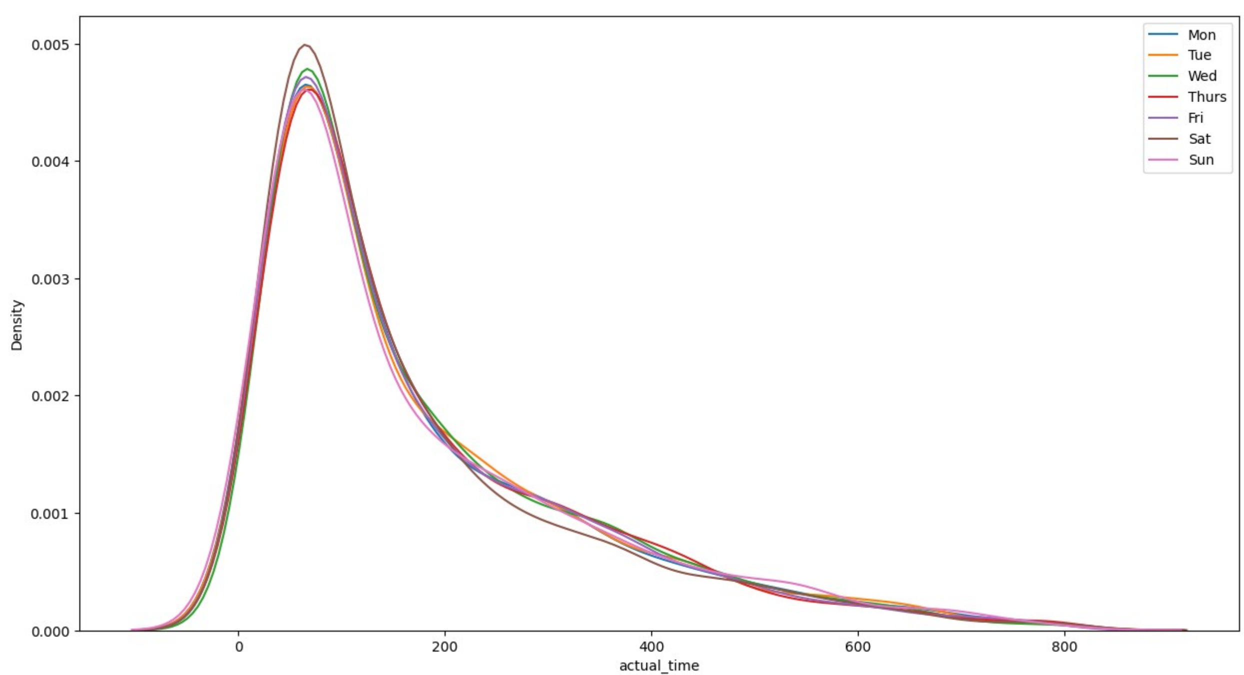
{"type":"string"}

```
plt.figure(figsize=(15,8))

days = ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun']

daydist=[]

for day in days:
  daydist.append(trip[trip["trip_dayofweek"]==day]["actual_time"])
  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)

plt.legend()
plt.show()
```

```
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
```

```
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
<ipython-input-478-1fb61385b998>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(trip[trip["trip_dayofweek"]==day]["actual_time"],
hist=False,label=day)
```



```
'''
The distributions look slightly different. We will perform the non-
parametric counterpart to one
way ANOVA, the Kruskal Willis Test (and hence the medians in our
hypothesis).
'''
```

{"type":"string"}

```
from scipy import stats

stats.kruskal(daydist[0],daydist[1],daydist[2],daydist[3],daydist[4],d
aydist[5],daydist[6])
```

```
KruskalResult(statistic=8.743242607332519, pvalue=0.18854113288304772)
```

```
'''
Since p > ff (0.05), we fail to reject the null hypothesis. Hence, the
sample distributions of time
taken seem to be the same for all week days.
'''
```

{"type":"string"}

#Normalize/Standardize the numerical features using MinMax Scaler/ Standard Scaler

```
numerical_columns = [
'start_scan_to_end_scan',
'od_time_diff_hour',
'actual_distance_to_destination',
'actual_time',
'osrm_time',
'osrm_distance',
'segment_actual_time_sum',
'segment_osrm_time_sum',
'segment_osrm_distance_sum']

scaler = StandardScaler()

trip[numerical_columns] =
scaler.fit_transform(trip[numerical_columns])

trip[numerical_columns]
```

{"summary":"{\n \"name\": \"trip[numerical_columns]\",\n \"rows\":
12723,\n \"fields\": [\n   {\n   \"column\":
\"start_scan_to_end_scan\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.0000393012242483,\n
\"min\": -1.1629177449049912,\n   \"max\": 4.049455390794036,\n
\"num_unique_values\": 1171,\n   \"samples\": [\n   -
1.0572615326948758,\n   0.2653977164539766,\n

```
1.9950290422640145\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"od_time_diff_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000393012242483,\n        \"min\": -
1.1629146849499417,\n        \"max\": 4.050309988028328,\n
\"num_unique_values\": 12723,\n        \"samples\": [\n          -
0.8754513609423765,\n          -0.6354821505230236,\n
1.4308842167879805\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_distance_to_destination\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0000393012242483,\n
\"min\": -0.878557444758895,\n        \"max\": 4.178358239461468,\n
\"num_unique_values\": 12707,\n        \"samples\": [\n          -
0.8687661295733056,\n          -0.5069699350490017,\n          -
0.4289436914139221\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000393012242483,\n        \"min\": -
1.0651814621104405,\n        \"max\": 4.031419180845916,\n
\"num_unique_values\": 753,\n        \"samples\": [\n
2.7288140537379633,\n          2.4822043452078173,\n          -
0.16726919002631818\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000393012242483,\n        \"min\": -
1.0015135063667786,\n        \"max\": 4.113870775391394,\n
\"num_unique_values\": 344,\n        \"samples\": [\n
1.8326858929857226,\n          0.6851807703210513,\n
0.9478626658707954\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000393012242483,\n        \"min\": -
0.9229378493725726,\n        \"max\": 4.150640947245795,\n
\"num_unique_values\": 12640,\n        \"samples\": [\n          -
0.15362842207957553,\n          1.0799885429388163,\n          -
0.5130299765561271\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_actual_time_sum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0000393012242481,\n
\"min\": -1.0617636801743235,\n        \"max\": 4.037107446023315,\n
\"num_unique_values\": 742,\n        \"samples\": [\n          -
0.11965016372332415,\n          1.2298637922740534,\n
0.7269788747089929\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_osrm_time_sum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0000393012242486,\n
\"min\": -1.0038504861356594,\n        \"max\": 4.0462834540573684,\n
\"num_unique_values\": 385,\n        \"samples\": [\n
1.219715950217987,\n          2.0990925069680166,\n
```

3.317657164178772\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"segment_osrm_distance_sum\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.0000393012242483,\n   \"min\": -0.9375980759050305,\n   \"max\": 4.130134832466642,\n   \"num_unique_values\": 12656,\n   \"samples\": [\n   -0.7434416814960256,\n   -0.7965206584172162,\n   -0.8022760576474295\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
trip[numerical_columns].describe()
```

{"summary":"{\n \"name\": \"trip[numerical_columns]\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"start_scan_to_end_scan\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4498.097084056237,\n   \"min\": -1.1629177449049912,\n   \"max\": 12723.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   -1.6195660879543272e-17,\n   -0.34114720549298233,\n   12723.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"od_time_diff_hour\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4498.097107379719,\n   \"min\": -1.1629146849499417,\n   \"max\": 12723.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   7.818594907365717e-18,\n   -0.3418601535238833,\n   12723.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"actual_distance_to_destination\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4498.08170266015,\n   \"min\": -0.878557444758895,\n   \"max\": 12723.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   -7.371818055516248e-17,\n   -0.46890123482819834,\n   12723.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"actual_time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4498.09371151592,\n   \"min\": -1.0651814621104405,\n   \"max\": 12723.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   -8.041983333290453e-17,\n   -0.40123224683696973,\n   12723.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"osrm_time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4498.086808424442,\n   \"min\": -1.0015135063667786,\n   \"max\": 12723.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   4.467768518494696e-17,\n   -0.39319753772526617,\n   12723.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"osrm_distance\",\n   \"properties\": {\n   \"dtype\": \"

```
\"number\",\n        \"std\": 4498.084398225049,\n        \"min\": -
0.9229378493725726,\n        \"max\": 12723.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.7976032407204913e-17,\n          -0.48363392594813054,\n
12723.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_actual_time_sum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4498.093489176093,\n
\"min\": -1.0617636801743235,\n        \"max\": 12723.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          -
3.127437962946287e-17,\n          -0.3997379659114591,\n
12723.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_osrm_time_sum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4498.088713263861,\n
\"min\": -1.0038504861356594,\n        \"max\": 12723.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
6.031487499967839e-17,\n          -0.4134119408892109,\n
12723.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_osrm_distance_sum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4498.08553244737,\n
\"min\": -0.9375980759050305,\n        \"max\": 12723.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          -
8.488760185139921e-17,\n          -0.46280769784712744,\n
12723.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Business Insights

```
'''

1. Most trips use “Carting” (~8K) transportation type as opposed to
“FTL” (~4K).

2. Bengaluru, Mumbai and Gurgaon are both the top source and
destination cities.
Bhiwandi, Delhi, Hyderabad, Chennai, Pune and Chandigarh are also some
of the top contributors.
So, we see that the Southern, Western and Northern corridors have the
top contributing cities.

3. The top contributor states (both source and destination) are :
Maharashtra is the highest,
followed by Karnataka, Haryana, Tamil Nadu and Telengana, Delhi,
Gujarat, UP and West Bengal.
Again we see Western, Southern and Northern corridors have significant
contribution to the traffic.
```

4. The greatest amount of time was spent in intra-state trips within Maharashtra, Karnataka,
Tamil Nadu, Telengana, UP.

5. The greatest amount of distance was covered on inter-state trips in Karnaataka, Maharashtra,
Tamil Nadu, Telengana and Andhra.

6. Similarly, the greatest amount of time was spent in intra-city trips within Bangalore, Mumbai, Hyderabad.
A significant time is also spent in inter-city trips from Mumbai to Bhiwandi and
Guragon to Delhi. These routes also contributed to the greatest amount of distance covered on trips.

7. Hourly distribution of number of trips in a day : minimum trips occuring during the day
hours (8 AM to 1 PM) and maximum occuring during late night or early morning hours (8 PM to 2 AM).

8. Week Day : we see that maximum number of trips are happening on Wednesday and minimum on Sunday.

9. OSRM seems to be calculating time taken as less than what time it actually takes.
This might be because in actual scenario, there might be delays caused by unprecedented traffic
or other delays.

10. OSRM seems to be calculating distance as less than what distance is actually covered.
So, OSRM is underestimating time and overestimting the distance.

11. The time taken by full truck load deliveries is on average, a lot higher (>300 hours) (this is
because the distance covered by trucks is also mucvh higher since they don’t make stops) than
the cart deliveries (<100 hours).
The full truck load deliveries cover much longer distances
on average (>150 kms) than carting deliveries (~ 25 kms).

12. Hourly distribution of trip time and distances : Time and distances follow similar trends
against the hour of the day. Maximum time and distance deliveries are likely to be made
during peak morning hours of 10 AM to 12 PM as well as 5 PM, 7 PM and 1 AM.

13. Weekday distribution of trip time and distances : On average, time

```
taken is slightly more on
weekdays and Sunday as compared to Saturday. However, they are very similar.
Distance covered is also lowest on Saturday.

14. Route type of top 3 Destination states :
Maharashtra has 86% Carting and 14% FTL,
Karnataka has 86% Carting and 14% FTL,
Haryana has 81% Carting and 19% FTL.

15. Route type of top 3 Source states :
Maharashtra hs 85% Carting and 15% FTL,
Karnataka has 88% Carting and 12% FTL,
Haryana has 75% Carting and 25% FTL.

'''
```

{"type":"string"}

Recommendations

```
'''

1. Since there is significant dfference between the time and distances calculated by OSRM with
actual time and distances, it might make sense to revisit the information which is fed to the
routing engine for trip planning. We need to check for discrrepancies with transporters nd to
check if the routing engine is configured for optimum performance.

2. We have seen that the Western, Southern and Northern corridors have significant traffic,
however, not so much in Eastern, Central and North Eastern corridors.
Increasing the presence in these corridors is worth investigating.

3. There is a need to plan resources (specifically during regional festivities) in the states/cities
which have highest contribution to traffic.

4. Road network can be taken into consideration to increase the number of FTL deliveries inter
state and to connect the states where there is lower traffic.

5. Since intra state or intra city trips are more likely to be using “carting” as mehod of transport,
the number of hubs could be increased in those cities and states which have highest contribution to traffic.
```

```
'''

{"type":"string"}
```