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.

PROBLEM STATEMENT

Analysing the Delhivery Data to gain insights on the Actual and OSRM values of attributes of trips and segment within it, along with the variations in Date-time and Location of deliveries.

BASIC DATA EXPLORATION AND ANALYSIS

IMPORTING NECESSARY LIBRARIES AND DOWNLOADING DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/c
```

```
--2024-01-15 15:26:58--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/0
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.155.174
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.155.17
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: ‘delhivery_data.csv?1642751181’

delhivery_data.csv? 100%[===================>]  53.04M  85.0MB/s    in 0.6s

2024-01-15 15:26:58 (85.0 MB/s) - ‘delhivery_data.csv?1642751181’ saved [55617130/556
```

```
df= pd.read_csv("delhivery_data.csv?1642751181")
```

```
df.head(15)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_u
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	t 153741093647649
10	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	t 153768492602129
11	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	t 153768492602129
12	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	t 153768492602129
13	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	t 153768492602129
14	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc2117...	FTL	t 153768492602129

```
df.shape
```

```
(144867, 24)
```

```
df.info()
```

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

```
df.describe()
```

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actua
count	144867.000000	144867.000000	144867.000000	144867.
mean	961.262986	232.926567	234.073372	416.
std	1037.012769	344.755577	344.990009	598.
min	20.000000	9.000000	9.000045	9.
25%	161.000000	22.000000	23.355874	51.
50%	449.000000	66.000000	66.126571	132.
75%	1634.000000	286.000000	286.708875	513.
max	7898.000000	1927.000000	1927.447705	4532.

MISSING VALUE DETECTION AND HANDLING

```
df.isnull().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
```

```
df = df.dropna(how='any')
df = df.reset_index(drop=True)
```

```
df.shape
```

```
(144316, 24)
```

```
df.isnull().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
```

```
df.duplicated().value_counts()
```

```
False    144316
dtype: int64
```

The data contains no more duplicates or nulls.

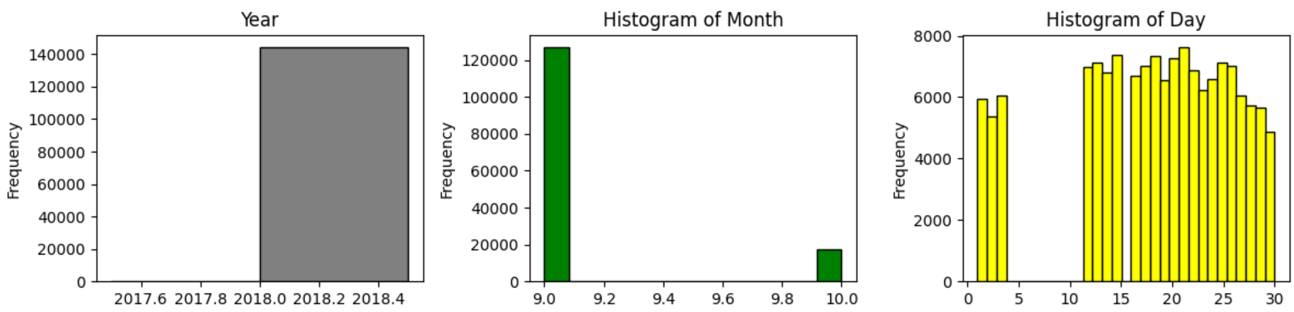
CONVERTING DATE TIME COLUMNS TO PANDAS DATETIME

```
date_columns = ['trip_creation_time', 'od_start_time', 'od_end_time']
df[date_columns] = df[date_columns].apply(pd.to_datetime)
```

Univariate Analysis

DATE TIME ANALYSIS

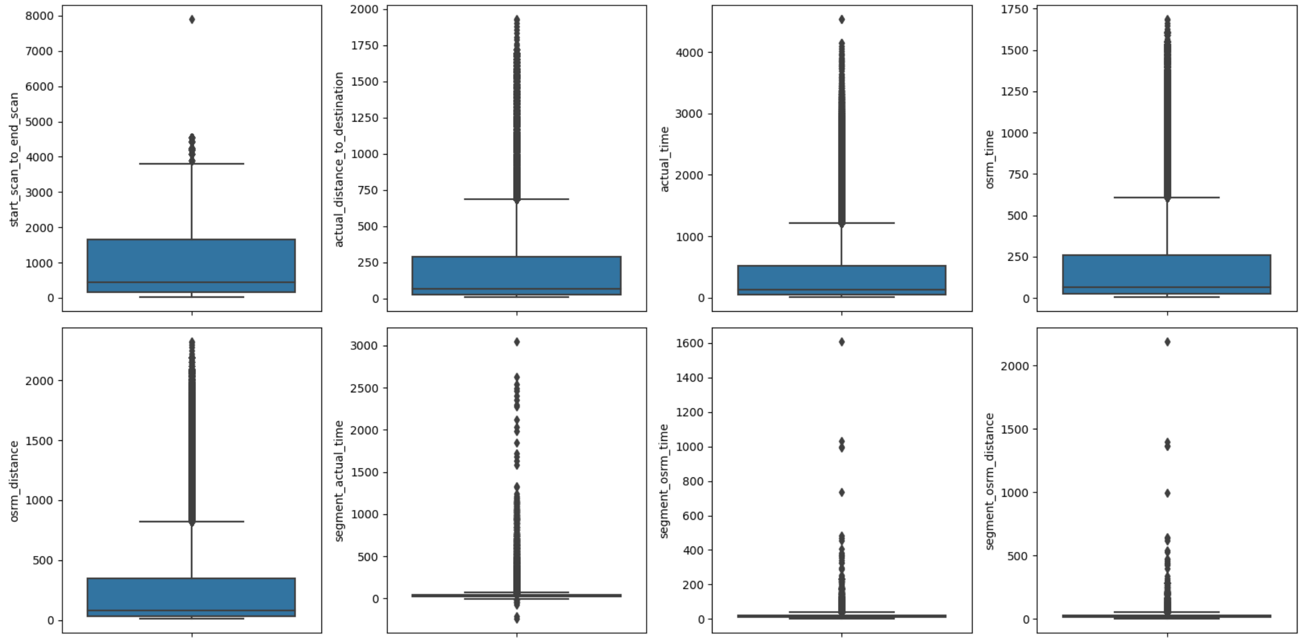
```
trip_creation_year = df['trip_creation_time'].dt.year
trip_creation_month = df['trip_creation_time'].dt.month
trip_creation_day = df['trip_creation_time'].dt.day


# Create subplots
fig, axes = plt.subplots(1,3, figsize=(12,3))

# Plot histograms for Year, Month, and Day
trip_creation_year.plot(kind='hist', bins=2, ax=axes[0], color='gray', edgecol
axes[0].set_title('Year')

trip_creation_month.plot(kind='hist', bins=12, ax=axes[1], color='green', edge
axes[1].set_title('Histogram of Month')

trip_creation_day.plot(kind='hist', bins=31, ax=axes[2], color='yellow', edgec
axes[2].set_title('Histogram of Day')

# Adjusting layout and showing the plot

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



1. The data is of only one year, 2018.
2. The shipment has been done only during the months of September and October.
3. The delivery is concentrated towards the latter half of the month, i.e., after 15. There are some early month deliveries too, with a significant gap in between.

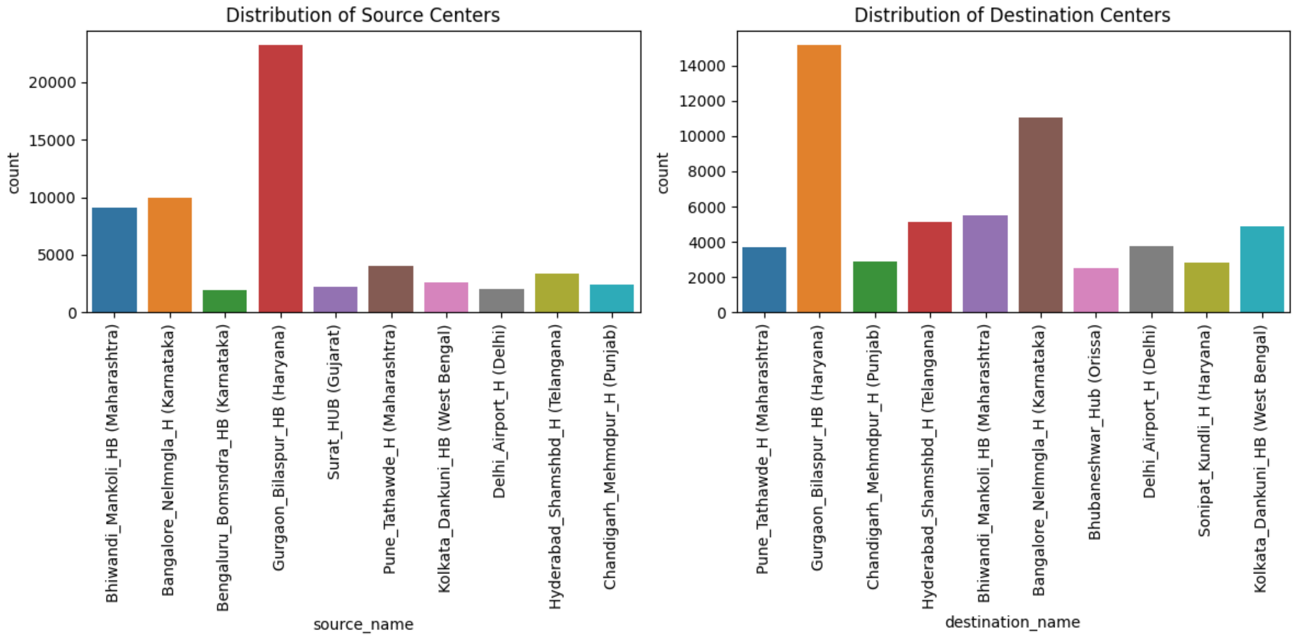
VISUAL ANALYSIS OF CONTINUOUS VARIABLES

```
# Boxplots for numerical columns

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))
axes = axes.flatten()

column = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_
           'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance'

# plotting the boxplot
for i, c in enumerate(column):
    sns.boxplot(y=df[c], ax=axes[i])


# Adjust layout and showing the plot
plt.tight_layout()

plt.show()
```



1. Actual distance to destination and osrm distance have approxiamtely the same range with quite a large number of outliers.
2. Osrm time is comparatively lesser than actual time, with a good number of outliers.
3. Segment actual time and Segment osrm time are also concentrated within a small range. Segment actual time also has some negative values. There are also some outliers which are very spreaded.
4. Segment osrm distance is also concentrated to a very narrow range starting from 0, with some spreaded ouliers.

```
# Countplot for top 10 Source and destination centers

# Filter the DataFrame to include only the top source and destination centers
top_source_centers = df['source_name'].value_counts().nlargest(10).index.sort_
top_destination_centers = df['destination_name'].value_counts().nlargest(10).i

df_top_source = df[df['source_name'].isin(top_source_centers)]
df_top_destination = df[df['destination_name'].isin(top_destination_centers)]

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Create countplot for top source and destination centers
sns.countplot(x='source_name', data=df_top_source, ax=axes[0])
axes[0].set_title('Distribution of Source Centers')
axes[0].tick_params(axis='x', rotation=90)

sns.countplot(x='destination_name', data=df_top_destination, ax=axes[1])
axes[1].set_title('Distribution of Destination Centers')
axes[1].tick_params(axis='x', rotation=90)

# Adjust layout and show the plot
plt.tight_layout()

plt.show()
```



1. The states from which the maximum delivery done is Haryana, Karnataka and Maharashtra.
2. Cities like Gurgaon, Bangalore and Bhiwandi are the major source centers.
3. The major Destination of the delivery is the states of Haryana, Karnataka followed by Maharashtra, Telangana and West Bengal.
4. Major destination cities are Gurgaon, Bangalore, Hyderabad, Bhiwandi and Kolkata.

VISUAL ANALYSIS OF CATEGORICAL VARIABLES

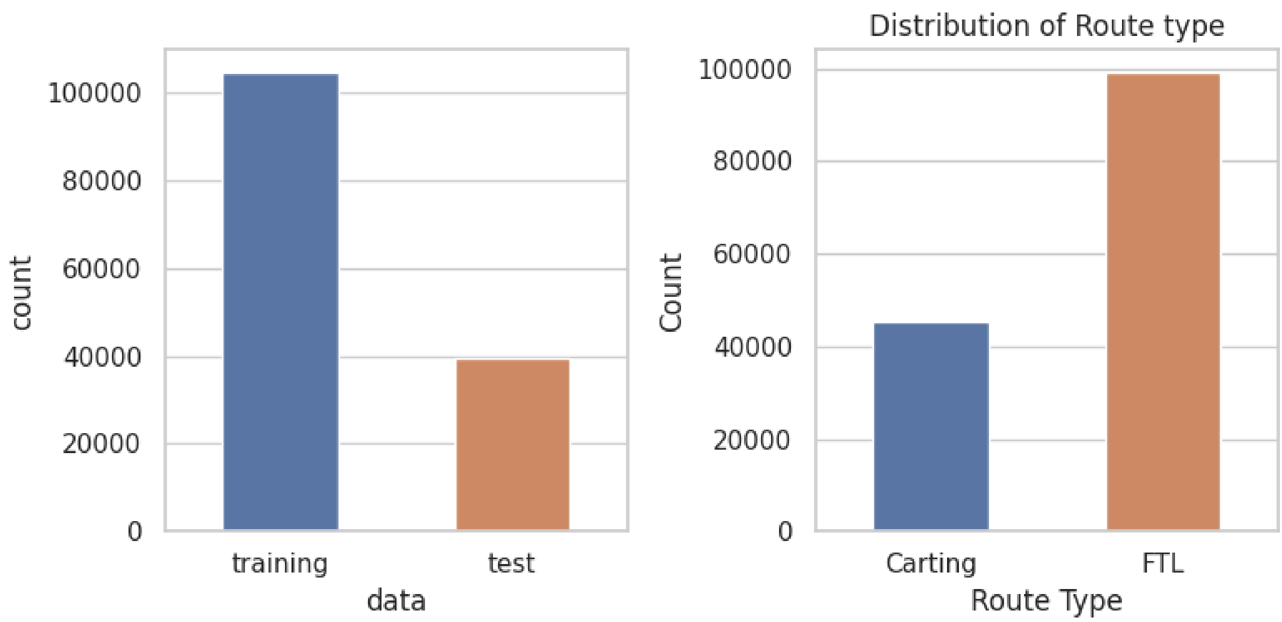
```
df['data'].value_counts()
```

```
training    104632
test         39684
Name: data, dtype: int64
```

```
df['route_type'].value_counts()
```

```
FTL        99132
Carting    45184
Name: route_type, dtype: int64
```

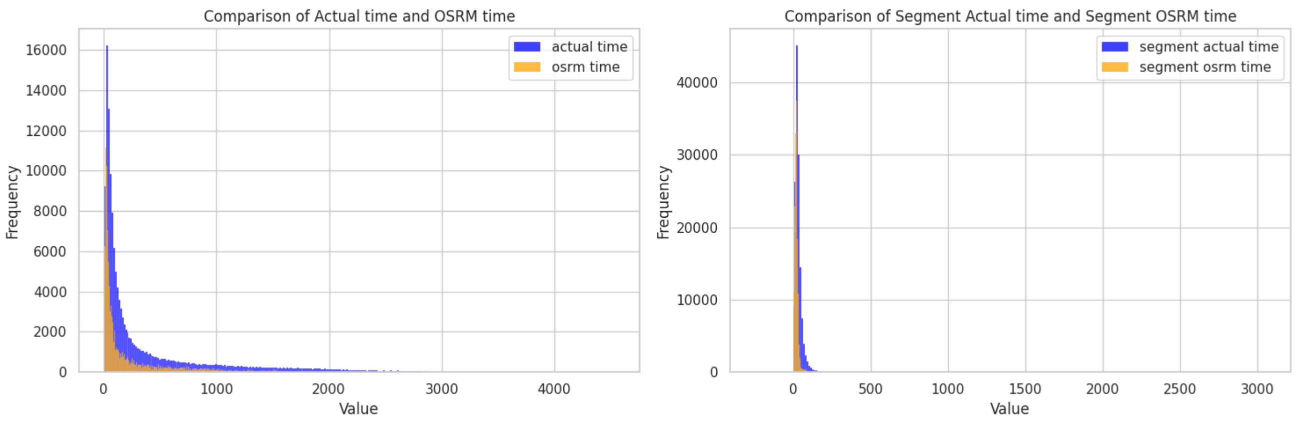
```
# Countplot for Data and Route type

sns.set(style="whitegrid")
bar_width = 0.5
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

# Plotting the graph
sns.countplot(x='data', data=df, width = bar_width, ax=axes[0])
plt.xlabel('Data')
plt.ylabel('Count')
plt.title('Distribution of Data')

sns.countplot(x='route_type', data=df, width= bar_width, ax=axes[1])
plt.xlabel('Route Type')
plt.ylabel('Count')
plt.title('Distribution of Route type')

# Adjusting and showing the plot
plt.tight_layout()

plt.show()
```



1. The number of Trainig data in the Dataset is approximately 2.5 times larger than Test data.
2. The Full Truck Load(FTL) shipment is approximately 2 times than the Carting shipment.

BIVARIATE ANALYSIS

```
# Distribution of actual time vs osrm time and segment_actual_time and segment
sns.set(style="whitegrid")

#creating subplot
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

#creating histogram
sns.histplot(data=df, x='actual_time', bins=300, color='blue', label='actual t
sns.histplot(data=df, x='osrm_time', bins=300, color='orange', label='osrm tim


sns.histplot(data=df, x='segment_actual_time', bins=300, color='blue', label='
sns.histplot(data=df, x='segment_osrm_time', bins=300, color='orange', label='

#adding lables
axes[0].set_xlabel('Value')
axes[0].set_ylabel('Frequency')
axes[0].set_title('Comparison of Actual time and OSRM time')
axes[0].legend()

axes[1].set_xlabel('Value')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Comparison of Segment Actual time and Segment OSRM time')
axes[1].legend()

#adjusting and showing the plot
plt.tight_layout()

plt.show()
```



OSRM values are significantly lesser than the actual values.

MERGING OF ROWS and AGGREGATIONS / FEATURE CREATION

GROUPING SEGMENT-WISE WITHIN A TRIP

```
df['segment_key'] = df['trip_uuid'] + '_' +df['source_center'] +'_'+ df['desti

segment_columns = ['segment_actual_time', 'segment_osrm_distance', 'segment_os

for c in segment_columns:
    df[c + '_sum'] = df.groupby('segment_key')[c].cumsum()

df[[c + '_sum' for c in segment_columns]]
```

	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_sum
0	14.0	11.9653	11.0
1	24.0	21.7243	20.0
2	40.0	32.5395	27.0
3	61.0	45.5619	39.0
4	67.0	49.4772	44.0
...	...	...	...
144311	92.0	65.3487	94.0
144312	118.0	82.7212	115.0
144313	138.0	103.4265	149.0
144314	155.0	122.3150	176.0
144315	423.0	131.1238	185.0

144316 rows × 3 columns

CREATING A SEGMENT DICTIONARY

```
segment_dict = {

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

}
```

AGGREGATING BASED ON SEGMENT KEY AND SORTING BY TIME

```
segment = df.groupby('segment_key').agg(segment_dict).reset_index()
segment = segment.sort_values(by=['segment_key','od_end_time'], ascending=True
```

```
segment
```

	segment_key	data	trip_creation_time	
0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	th
1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	th
2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	th
3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	th
4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	th
...	...	...	...	
26217	trip-153861115439069069_IND628204AAA_IND627657AAA	test	2018-10-03 23:59:14.390954	tl
26218	trip-153861115439069069_IND628613AAA_IND627005AAA	test	2018-10-03 23:59:14.390954	tl
26219	trip-153861115439069069_IND628801AAA_IND628204AAA	test	2018-10-03 23:59:14.390954	tl
26220	trip-153861118270144424_IND583119AAA_IND583101AAA	test	2018-10-03 23:59:42.701692	tl
26221	trip-153861118270144424_IND583201AAA_IND583119AAA	test	2018-10-03 23:59:42.701692	tl

26222 rows × 20 columns

CALCULATING TIME DIFFERENCE IN HOURS FROM START TO END OF TRIP

```
segment['od_end_time'] = pd.to_datetime(segment['od_end_time'])
segment['od_start_time'] = pd.to_datetime(segment['od_start_time'])


segment['od_time_diff_hour'] = (segment['od_end_time'] - segment['od_start_tim
segment['od_time_diff_hour'].sort_values(ascending=False)
```

```
24023    7898.551955
7962     4535.715225
25657    4440.938567
9420     4239.454516
23562    4207.224100
            ...
8032       23.461468
15104      23.118147
10090      22.996359
2512       21.107632
13265      20.702813
Name: od_time_diff_hour, Length: 26222, dtype: float64
```

CREATING A TRIP DICTIONARY

```
trip_dict = {

    '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',

    'start_scan_to_end_scan' : 'sum',
    'od_time_diff_hour' : 'sum',
    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',
    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum',

}
```

GROUPING BY TRIP ID AND AGGREGATING BY TRIP DICTIONARY

```
trip = segment.groupby('trip_uuid').agg(trip_dict).reset_index(drop=True)
trip
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_u
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	153671041653548
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	153671042288605
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	153671043369099
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	153671046011330
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	153671052974046
...	...	...	...	...	
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	153861095625827
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	153861104386292
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	153861106442901
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	153861115439069
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	153861118270144

14787 rows × 18 columns

FEATURE CREATION FROM TIMESTAMPS COLUMNS

```
trip['trip_creation_year'] = trip['trip_creation_time'].dt.year
trip['trip_creation_month'] = trip['trip_creation_time'].dt.month
trip['trip_creation_week'] = trip['trip_creation_time'].dt.isocalendar().week
trip['trip_creation_dayofweek'] = trip['trip_creation_time'].dt.dayofweek
trip['trip_creation_day'] = trip['trip_creation_time'].dt.day
trip['trip_creation_hour'] = trip['trip_creation_time'].dt.hour
```

```
trip[['trip_creation_year','trip_creation_month','trip_creation_dayofweek','tr
```

	trip_creation_year	trip_creation_month	trip_creation_dayofweek	trip_creatic
0	2018	9	2	
1	2018	9	2	
2	2018	9	2	
3	2018	9	2	
4	2018	9	2	
...	...	...	...	
14782	2018	10	2	
14783	2018	10	2	
14784	2018	10	2	
14785	2018	10	2	
14786	2018	10	2	

14787 rows × 6 columns

```
trip['trip_creation_dayofweek'].value_counts()
```

```
2    2731
5    2128
3    2103
4    2057
1    2035
0    1980
6    1753
Name: trip_creation_dayofweek, dtype: int64
```

```
trip['trip_creation_hour'].value_counts()
```

```
22    1123
23    1107
20    1080
0      991
21     872
19     837
1      748
2      702
18     696
```

```
3     651
4     635
6     610
17    595
16    526
5     505
7     472
15    469
14    379
8     345
13    328
9     317
12    270
11    267
10    262
Name: trip_creation_hour, dtype: int64
```

1. Roughly equal number of trips are created throughout the week, maximum being on Tuesday. The number of trips are relatively lesser on Saturday and Sunday.
2. Maximum trips are created at late hours and relativery lesser during the morning hours.

FEATURE CREATION FROM SOURCE & DESTINATION NAMES

```
#Converting source and destination names to lower case

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

```
# Splitting the source and destination names to extract city and state.

#function to extract state names

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

#function to extract city names

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

  if city == 'pnq vadgaon sheri dpc': return 'vadgaonsheri'

  if city in ['pnq pashan dpc','pnq rahatani dpc', 'pune balaji nagar']:
    return 'pune'

  if city == 'hbr layout pc' :
    return 'bengaluru'

  if city == 'bhopal mp nagar':
    return 'bhopal'

  if city == 'mumbai antop hill':
    return 'mumbai'

  return city

 #Function to extract place within a city

def place(x):

  x = x.split('(')[0]
  len_ = len(x.split('_'))
  if len_ >= 3:
    return x.split('_')[1]

  if len_ == 2:
    return x.split('_')[0]

  return x.split(' ')[0]

# Function to extract code

def code(x):

  x = x.split('(')[0]
```

```
  if len(x.split('_')) >= 3:
    return x.split('_')[-1]

  return 'none'


trip['source_state'] = trip['source_name'].apply(lambda x:state(x))
trip['source_city']  = trip['source_name'].apply(lambda x:city(x))
trip['source_place'] = trip['source_name'].apply(lambda x:place(x))
trip['source_code'] = trip['source_name'].apply(lambda x:code(x))


trip['destination_state'] = trip['destination_name'].apply(lambda x:state(x))
trip['destination_city']  = trip['destination_name'].apply(lambda x:city(x))
trip['destination_place'] = trip['destination_name'].apply(lambda x:place(x))
trip['destination_code'] = trip['destination_name'].apply(lambda x:code(x))
```

```
trip[['source_state','source_city','source_place','source_code','destination_s
```

	source_state	source_city	source_place	source_code	destination_state	desti
0	uttar pradesh	kanpur	central	6	uttar pradesh	
1	karnataka	doddablpur	chikadpp	d	karnataka	
2	haryana	gurgaon	bilaspur	hb	haryana	
3	maharashtra	mumbai hub	mumbai	none	maharashtra	
4	karnataka	bellary	bellary	none	karnataka	
...	...	...	...	...	...	
14782	punjab	chandigarh	mehmdpur	h	punjab	
14783	haryana	fbd	balabhgarh	dpc	haryana	
14784	uttar pradesh	kanpur	govndngr	dc	uttar pradesh	
14785	tamil nadu	tirunelveli	vdkkusrt	i	tamil nadu	
14786	karnataka	sandur	wrdn1dpp	d	karnataka	

14787 rows × 8 columns

```
trip['source_state'].value_counts().head()
```

```
maharashtra    2714
karnataka      2143
haryana        1823
tamil nadu     1039
telangana       784
Name: source_state, dtype: int64
```

```
trip['destination_state'].value_counts().head()
```

```
maharashtra      2561
karnataka        2294
haryana          1640
tamil nadu       1084
uttar pradesh     805
Name: destination_state, dtype: int64
```

```
trip['source_city'].value_counts().head()
```

```
bengaluru    1131
gurgaon      1128
bhiwandi      697
mumbai        667
bangalore     648
Name: source_city, dtype: int64
```

```
trip['destination_city'].value_counts().head()
```

```
bengaluru    1221
mumbai        968
gurgaon       877
delhi         554
bangalore     551
Name: destination_city, dtype: int64
```

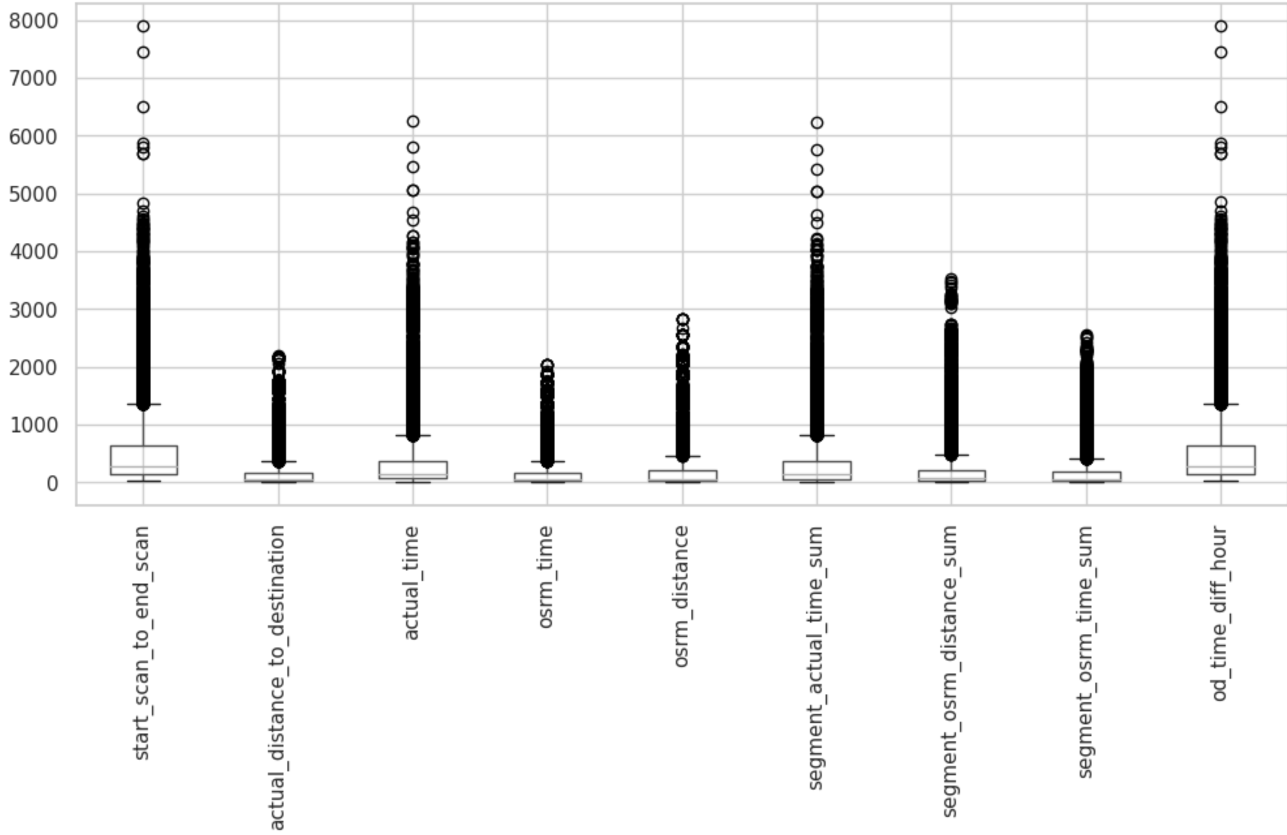
OUTLIER HANDLING

CHECKING FOR OULIERS IN NUMERICAL COLUMNS

```
numerical_cols = ['start_scan_to_end_scan','actual_distance_to_destination','a
                  'osrm_distance','segment_actual_time_sum','segment_osrm_dist
                  'segment_osrm_time_sum', 'od_time_diff_hour']
```

```
# plotting boxplot to find the outliers in the numerical columns

trip[numerical_cols].boxplot(rot=90, figsize=(12,5))
```

<Axes: >

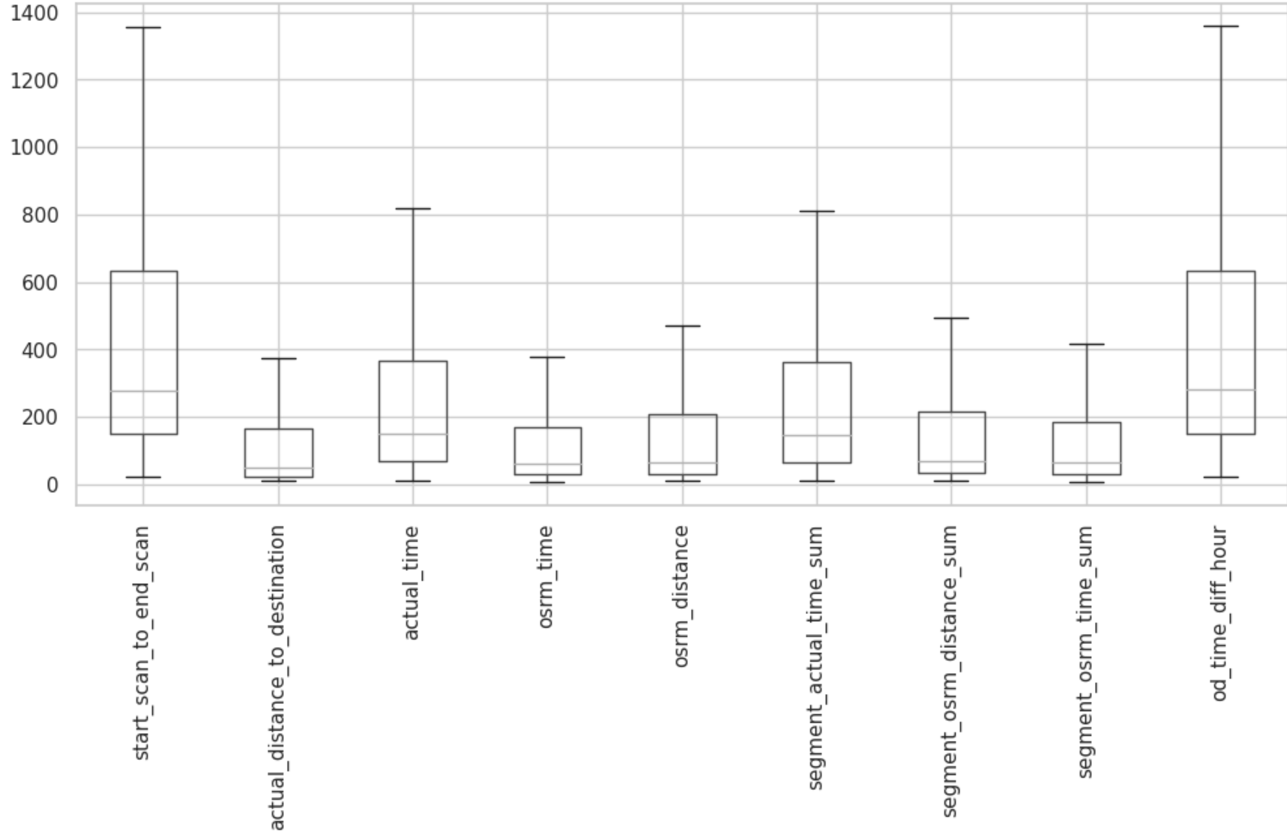


Columns like start_scan_to_end_scan, actual_time, segment_actual_time_sum and od_time_diff_hour have relatively larger number of outliers as compared to others.

REMOVING OUTLIERS USING IQR METHOD

```
# setting Q1 and Q3
Q1 = trip[numerical_cols].quantile(0.25)
Q3 = trip[numerical_cols].quantile(0.75)

IQR = Q3 - Q1

#setting the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

#setting numerical column within the bounds
trip[numerical_cols] = trip[numerical_cols].clip(lower=lower_bound, upper=uppe

#removing columns that are outside the bounds
trip = trip[~((trip[numerical_cols] < lower_bound) | (trip[numerical_cols] > u

trip = trip.reset_index(drop=True)



#check using boxplots after removal of outliers

trip[numerical_cols].boxplot(rot=90, figsize=(12,5))
```

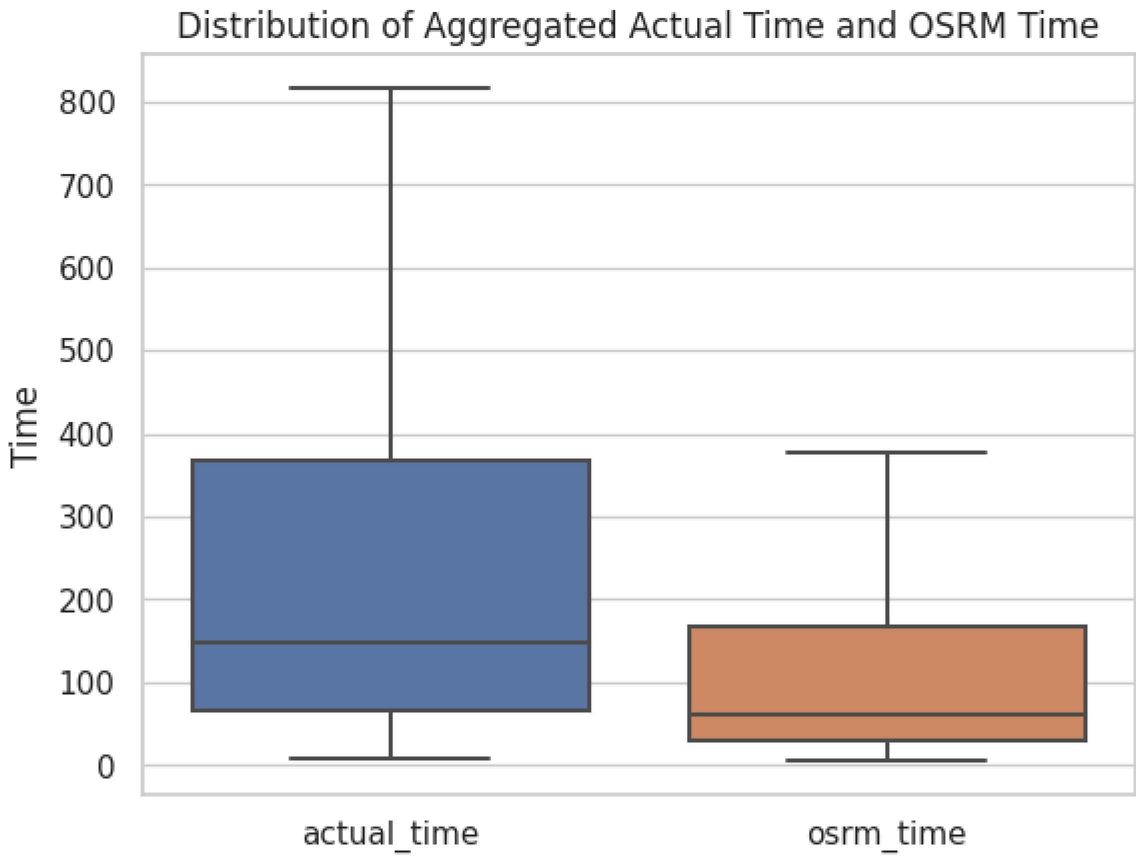
<Axes: >



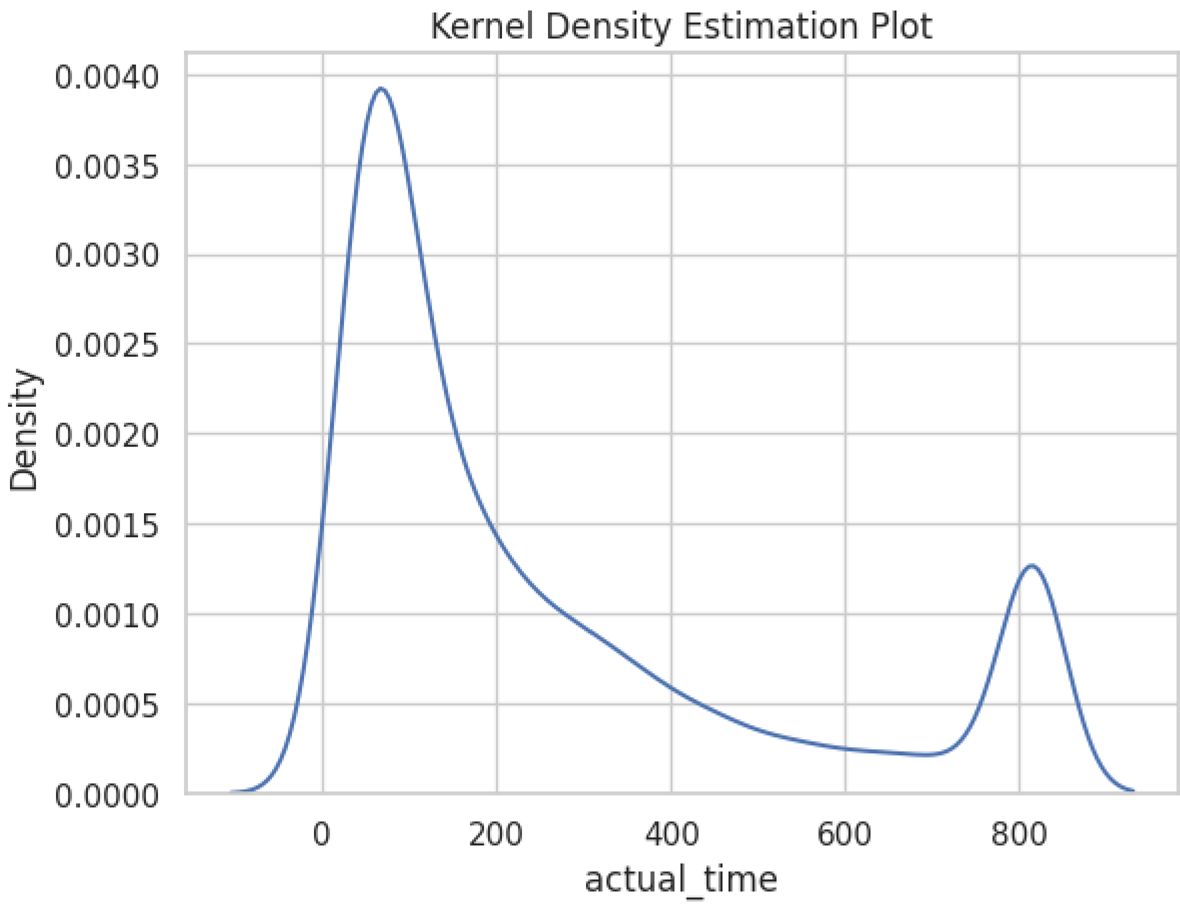
HYPOTHESIS TESTING & VISUAL ANALYSIS BASED ON AGGREGATED VALUES

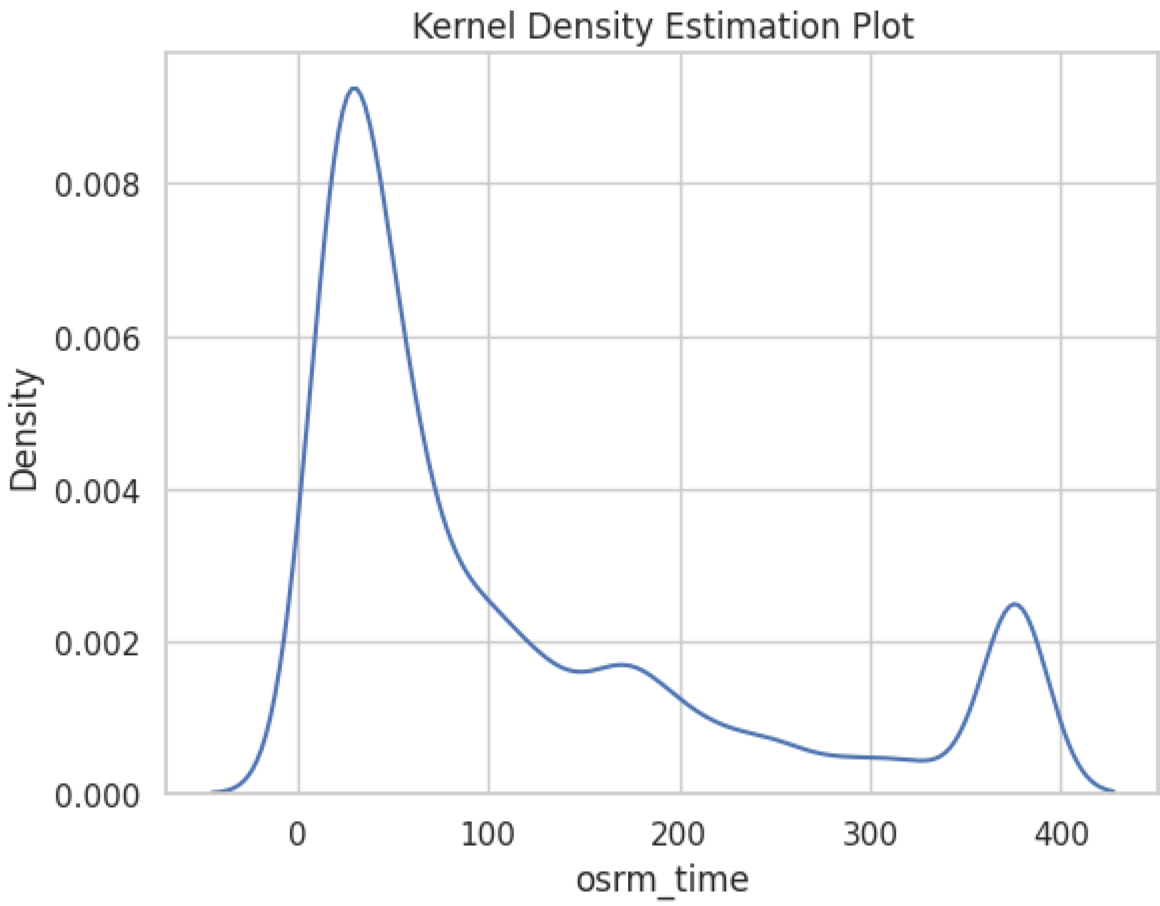
VISUAL ANALYSIS TO CHECK RELATION BETWEEN ACTUAL TIME & OSRM TIME

```
# Creating  a boxplot for actual_time and osrm_time

sns.boxplot(data=trip[['actual_time', 'osrm_time']])
plt.title('Distribution of Aggregated Actual Time and OSRM Time')
plt.ylabel('Time')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

sns.kdeplot(trip['actual_time'])
plt.title('Kernel Density Estimation Plot')

plt.show()


sns.kdeplot(trip['osrm_time'])
plt.title('Kernel Density Estimation Plot')
plt.show()
```





It seems as the range and mean value of actual_time is different than the osrm_time.

HYPOTHESIS TO CHECK THE RELATION BETWEEN ACTUAL TIME AND OSRM TIME

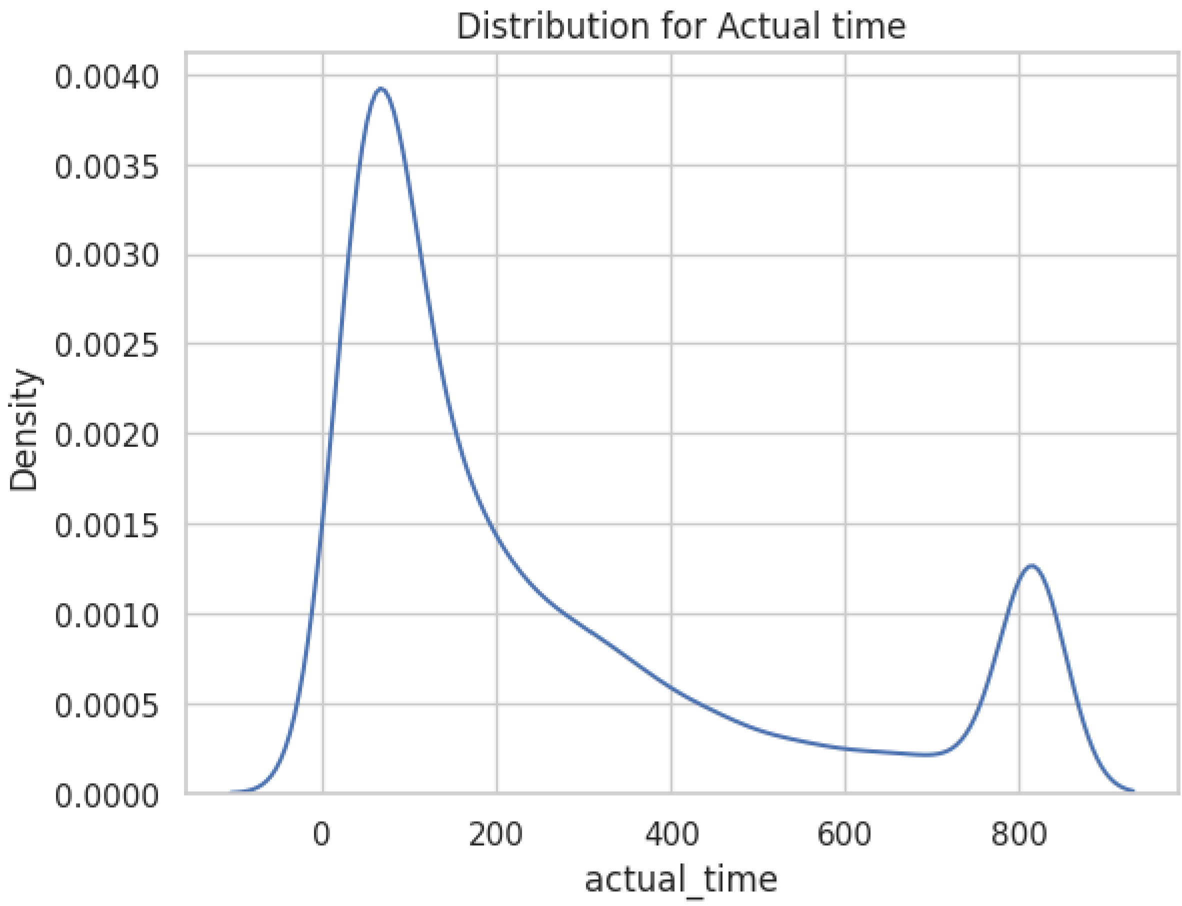
```
# H0: There is no significant difference in the means of actual time and osrm
# Ha: There is a significant difference between the means of actual time and c

from scipy.stats import ttest_ind

data1 = trip['actual_time']
data2= trip['osrm_time']
alpha = 0.05

# testing to find t-statistic and p-value

t_statistic, p_value = ttest_ind(data1, data2, equal_var=False)
print("t_statistic:", t_statistic)
print("p_value:", p_value)

#comparing against alpha
if(p_value<alpha):
  print("Reject the null hypothesis. There is a significant difference betweer
else:
  print("Failed to reject the null hypothesis. There is no significant differe
```

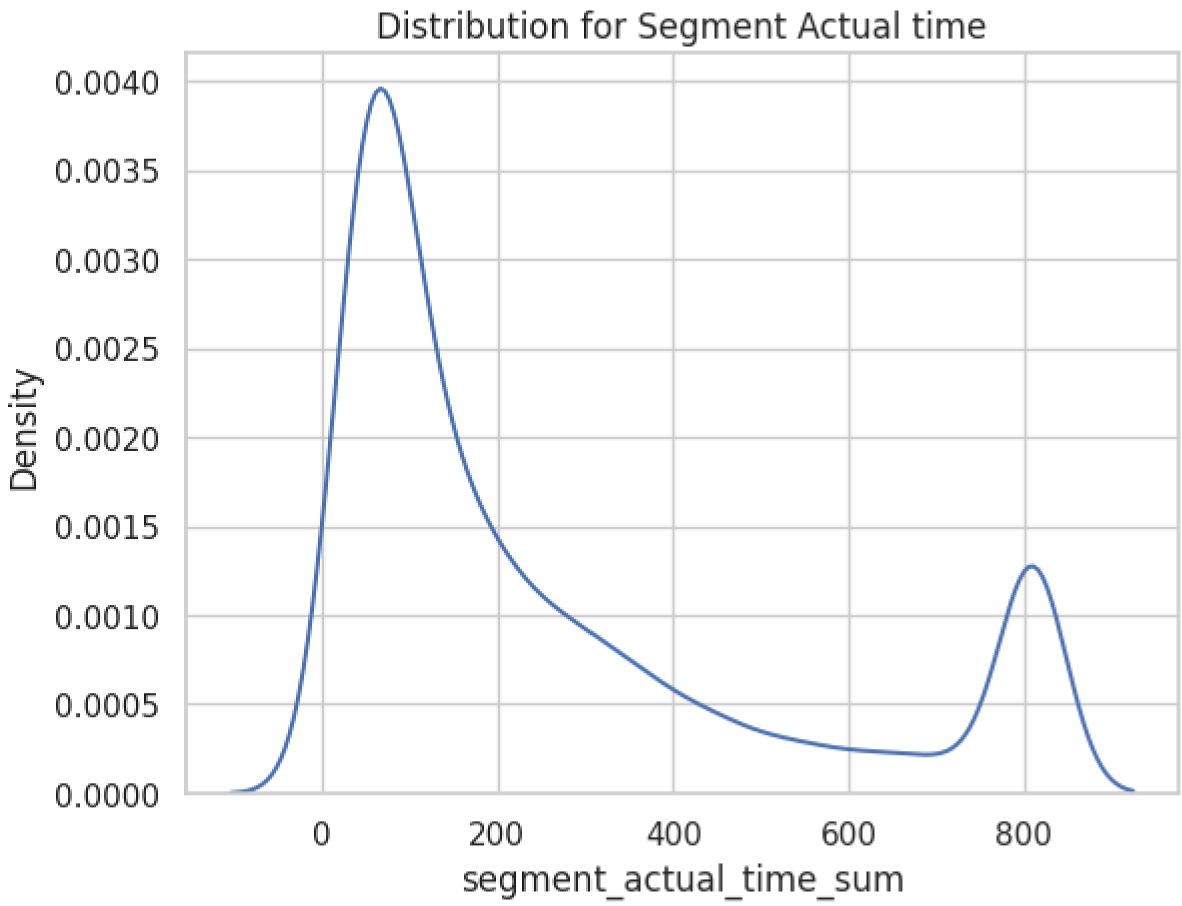
```
t_statistic: 63.30545280574021
p_value: 0.0
Reject the null hypothesis. There is a significant difference between the means of ac
```

VISUAL ANALYSIS FOR DISTRIBUTION OF ACTUAL TIME AND SEGEMENT ACTUAL TIME

```
# KDE plot to find the distribution for Actual time and Segment Actual Time

sns.kdeplot(trip['actual_time'])
plt.title('Distribution for Actual time')
plt.show()
```

```
sns.kdeplot(trip['segment_actual_time_sum'])
plt.title('Distribution for Segment Actual time')
plt.show()
```





From the graph it seems like the means of the two columns are equal, but we need to verify with the Hypothesis test.

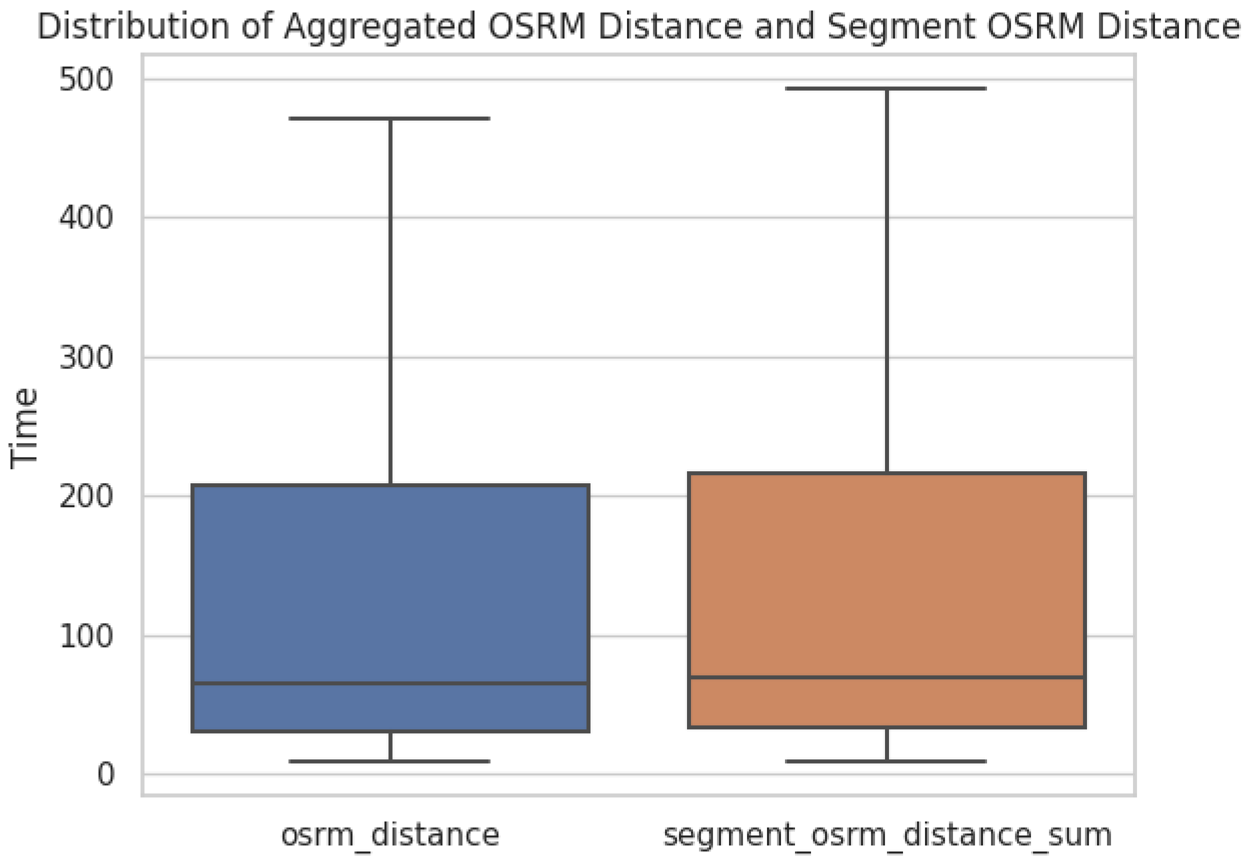
HYPOTHESIS TEST TO CHECK THE DIFFERENCE IN MEANS OF ACTUAL TIME AND SEGMENT ACTUAL TIME

```
# H0: There is no significant difference in the means of actual time and segme
# Ha: There is a significant difference between the means of actual time and s

from scipy.stats import ttest_ind

data1 = trip['actual_time']
data2= trip['segment_actual_time_sum']
alpha = 0.05

# testing to find t-statistic and p-value

t_statistic, p_value = ttest_ind(data1, data2, equal_var=False)
print("t_statistic:", t_statistic)
print("p_value:", p_value)

#comparing against alpha
if(p_value<alpha):
  print("Reject the null hypothesis. There is a significant difference betweer
else:
  print("Failed to rejct the null hypothesis. There is no significant differer
```

```
t_statistic: 0.7566645099710447
p_value: 0.44925691058084427
Failed to rejct the null hypothesis. There is no significant difference in the means
```

VISUAL ANALYSIS TO CHECK THE DISTRIBUTION OF OSRM DISTANCE & SEGMENT OSRM DISTANCE

```
# Using a boxplot to check the distribution of osrm distance and segment osrm

sns.boxplot(data=trip[['osrm_distance', 'segment_osrm_distance_sum']])
plt.title('Distribution of Aggregated OSRM Distance and Segment OSRM Distance'
plt.ylabel('Time')
plt.show()
```



From the graph, we can see that there is a slight difference in the means of both the columns.

HYPOTHESIS TO CHECK THE RELATION BETWEEN OSRM DISTANCE & SEGMENT OSRM DISTANCE

```
# H0: There is no significant difference in the means of osrm distance and seg
# Ha: There is a significant difference between the means of osrm distance anc

from scipy.stats import ttest_ind

data1 = trip['osrm_distance']
data2= trip['segment_osrm_distance_sum']
alpha = 0.05

# testing to find t-statistic and p-value

t_statistic, p_value = ttest_ind(data1, data2, equal_var=False)
print("t_statistic:", t_statistic)
print("p_value:", p_value)

#comparing against alpha
if(p_value<alpha):
  print("Reject the null hypothesis. There is a significant difference in the
else:
  print("Failed to rejct the null hypothesis. There is no significant differer
```

```
t_statistic: -4.735638441691023
p_value: 2.193806542256223e-06
Reject the null hypothesis. There is a significant difference in the means of actual
```

VISUAL ANALYSIS TO CHECK THE DISTRIBUTION OF OSRM TIME & SEGMENT OSRM TIME

```
# Using a boxplot to check the distribution of osrm time and segment osrm time

sns.boxplot(data=trip[['osrm_time', 'segment_osrm_time_sum']])
plt.title('Distribution of Aggregated OSRM Time and Segment OSRM Time')
plt.ylabel('Time')
plt.show()
```