Introduction

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

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

Problem Statement

The data in it's raw form doesn't offer much insights of what factors are driving the business. Let's dig and dive into the data , clean it and transform it and bring it to the suitable format so that we can conduct various statistical tests that provide better insights of the factors affecting the business.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data = pd.read_csv('delhivery_data.txt')
```

In [3]:

```
data.head(4)
```

Out[3]:

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

4 rows × 24 columns

In [4]:

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

In [5]:

```
data.shape
```

Out[5]:

```
(144867, 24)
```

In [6]:

```
# Dropping complete row duplicates
data.drop_duplicates(keep=False, inplace=True)
```

In [7]:

```
data.shape
```

Out[7]:

```
(144867, 24)
```

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 26.7+ MB
```

In [9]:

```
data.describe()
```

Out[9]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	
count	144867.000000	144867.000000	144867.000000	144867.000000	144
mean	961.262986	232.926567	234.073372	416.927527	
std	1037.012769	344.755577	344.990009	598.103621	
min	20.000000	9.000000	9.000045	9.000000	
25%	161.000000	22.000000	23.355874	51.000000	
50%	449.000000	66.000000	66.126571	132.000000	
75%	1634.000000	286.000000	286.708875	513.000000	
max	7898.000000	1927.000000	1927.447705	4532.000000	1

Missing Values Treatment

In [10]:

```
data.isnull().sum()
```

Out[10]:

```
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
```

- Since we have source_id and destination_id having missing values in the names is ok, since we anyhow group the data based on the id and not based on the name

In [ ]:

```

```

Creating New Features

- Before grouping the rows,let's see if we can make new features so that even they can be aggregated while grouping by

Creating Difference in start to end time in minutes

In [11]:

```
#converting to datetime
data['od_end_time'] = pd.to_datetime(data['od_end_time'])
data['od_start_time'] = pd.to_datetime(data['od_start_time'])
```

In [12]:

```
# Creating a new column which is a difference of od_start_time & od_end_time in
data['diff_start_end'] = round((data['od_end_time'] - data['od_start_time']  ).c
```

Source State & Destination State

In [13]:

```
import re
pattern = re.compile('\([A-Za-z]*[\s\S]*[A-Za-z]*\)')
```

In [14]:

```
# extracting source state from source_name
data['source_state'] = data['source_name'].apply(lambda x : np.nan if pd.isnull(
                                                            else pattern.search(x
```

In [15]:

```
# extracting destination state from destination_name
data['destination_state'] = data['destination_name'].apply(lambda x : np.nan if
                                                            else pattern.search(x
```

Extracting year,month & day of trip creation

In [16]:

```
data['trip_creation_time'] = pd.to_datetime(data['trip_creation_time'])
```

In [17]:

```
data['trip_year'] = data['trip_creation_time'].dt.year
```

In [18]:

```
data['trip_month'] = data['trip_creation_time'].dt.month
```

In [19]:

```
data['trip_day'] = data['trip_creation_time'].dt.day
```

In [20]:

```
data.head(3)
```

Out[20]:

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

In [21]:

```
data.shape
```

Out[21]:

```
(144867, 30)
```

Grouping the data

- Since for each product there are multiple records, we aggregate all using the suitable aggregation functions.
- Eg : For distance aggregation is done on sum as it represents the total distance between the source and destination. Similary based on the data suitable aggregation functions are used

In [22]:

```
data = data.groupby(['trip_uuid','source_center','destination_center']).agg(
                    {'actual_time':'last',
                     'osrm_time':'last',
                     'segment_actual_time':'sum',
                     'segment_osrm_time':'sum',
                     'osrm_distance':'last',
                     'segment_osrm_distance':'sum',
                     'route_type':pd.Series.mode,
                     'start_scan_to_end_scan':'last',
                     'actual_distance_to_destination':'last',
                     'diff_start_end':'last',
                     'destination_state':'last',
                     'source_state':'last',
                     'trip_year':'last',
                     'trip_month':'first',
                     'trip_day':'first'})
```

In [23]:

```
data
```

Out[23]:

			actual_time	osrm_time	segment_actual_
trip_uuid	**source_center**	**destination_center**			
trip-153671041653548748	**IND209304AAA**	**IND000000ACB**	732.0	329.0	
	IND462022AAA	**IND209304AAA**	830.0	388.0	
trip-153671042288605164	**IND561203AAB**	**IND562101AAA**	47.0	26.0	
	IND572101AAA	**IND561203AAB**	96.0	42.0	
trip-153671043369099517	**IND000000ACB**	**IND160002AAC**	611.0	212.0	
...	**...**	**...**	...	...	
trip-153861115439069069	**IND628204AAA**	**IND627657AAA**	51.0	41.0	
	IND628613AAA	**IND627005AAA**	90.0	48.0	
	IND628801AAA	**IND628204AAA**	30.0	14.0	
trip-153861118270144424	**IND583119AAA**	**IND583101AAA**	233.0	42.0	
	IND583201AAA	**IND583119AAA**	42.0	26.0	

26368 rows × 15 columns

In [24]:

```
data = data.reset_index()
```

In [25]:

```
data.head(5)
```

Out[25]:

	trip_uuid	source_center	destination_center	actual_time	osrm_time	segment_actu
0	trip-153671041653548748	IND209304AAA	IND000000ACB	732.0	329.0	
1	trip-153671041653548748	IND462022AAA	IND209304AAA	830.0	388.0	
2	trip-153671042288605164	IND561203AAB	IND562101AAA	47.0	26.0	
3	trip-153671042288605164	IND572101AAA	IND561203AAB	96.0	42.0	
4	trip-153671043369099517	IND000000ACB	IND160002AAC	611.0	212.0	

In [26]:

```
data.shape
```

Out[26]:

```
(26368, 18)
```

- Let's further group by based on the trip_uuid. This results in a single row for a single product
- Also this time we have to use the aggregation function sum for all the numeric variables, since in the previous group by the aggregation function 'last' captured the total value for each segment.

In [27]:

```
data = data.groupby(['trip_uuid']).agg(
                    {'actual_time':'sum',
                     'osrm_time':'sum',
                     'segment_actual_time':'sum',
                     'segment_osrm_time':'sum',
                     'osrm_distance':'sum',
                     'segment_osrm_distance':'sum',
                     'route_type':pd.Series.mode,
                     'start_scan_to_end_scan':'sum',
                     'actual_distance_to_destination':'sum',
                     'diff_start_end':'sum',
                     'destination_state':'last',
                     'source_state':'last',
                     'trip_year':'last',
                     'trip_month':'first',
                     'trip_day':'first'})
```

In [28]:

```
data = data.reset_index()
```

In [29]:

```
data.shape
```

Out[29]:

```
(14817, 16)
```

In [30]:

```
data
```

Out[30]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time
0	trip-153671041653548748	1562.0	717.0	1548.0	1008.0
1	trip-153671042288605164	143.0	68.0	141.0	65.0
2	trip-153671043369099517	3347.0	1740.0	3308.0	1941.0
3	trip-153671046011330457	59.0	15.0	59.0	16.0
4	trip-153671052974046625	341.0	117.0	340.0	115.0
...	...	...	...	...	...
14812	trip-153861095625827784	83.0	62.0	82.0	62.0
14813	trip-153861104386292051	21.0	12.0	21.0	11.0
14814	trip-153861106442901555	282.0	48.0	281.0	88.0
14815	trip-153861115439069069	264.0	179.0	258.0	221.0
14816	trip-153861118270144424	275.0	68.0	274.0	67.0

14817 rows × 16 columns

In [31]:

```
data['trip_uuid'].nunique()
```

Out[31]:

```
14817
```

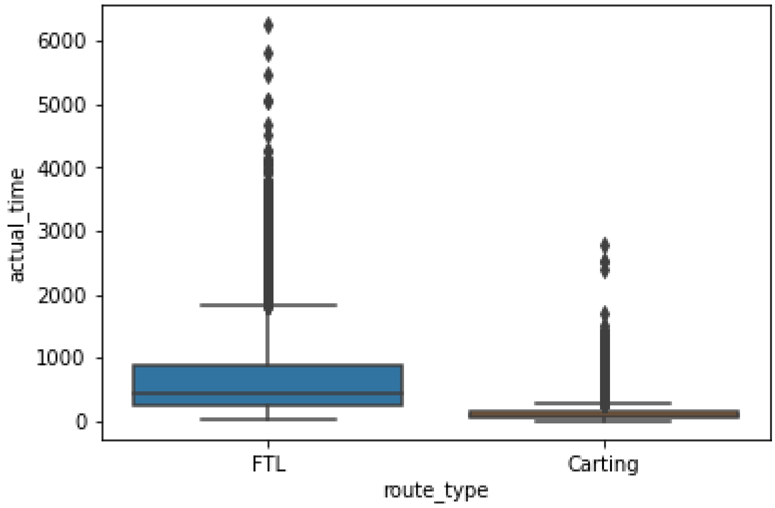
In [32]:

```
cols = ['actual_time','osrm_time','segment_actual_time','segment_osrm_time','osr
        'segment_osrm_distance','start_scan_to_end_scan','actual_distance_to_de
        'diff_start_end']
```

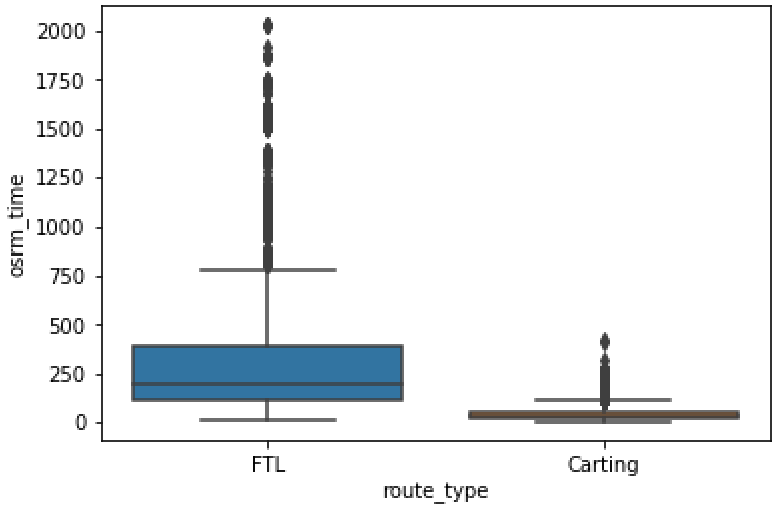
In [33]:

```
for col in cols:
    print(sns.boxplot(data['route_type'],data[col]))
    plt.show()
```

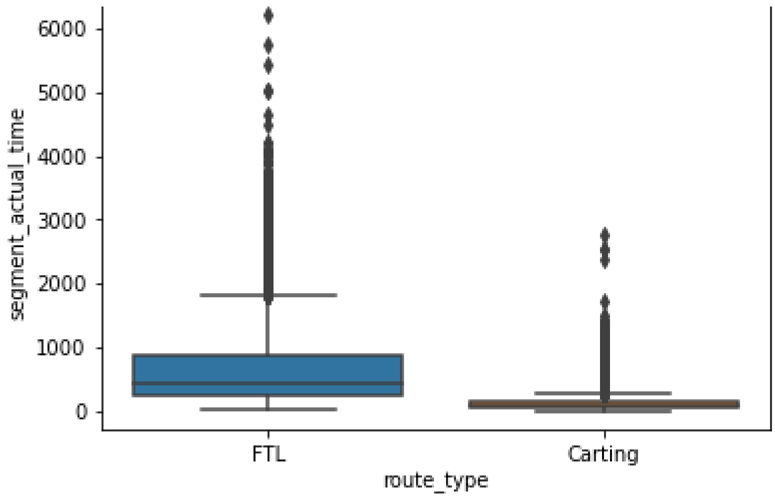
```
AxesSubplot(0.125,0.125;0.775x0.755)
```



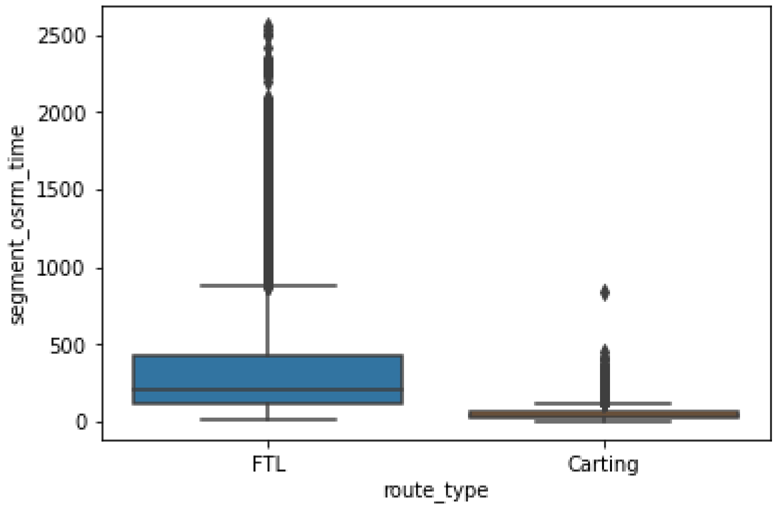

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



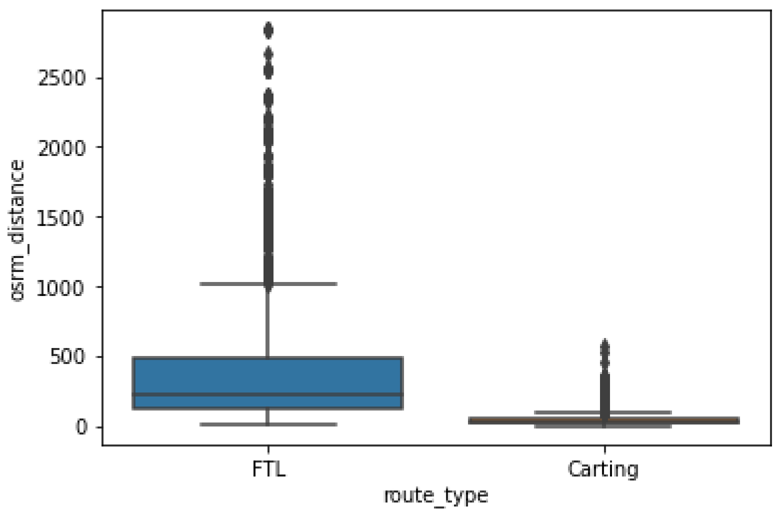

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



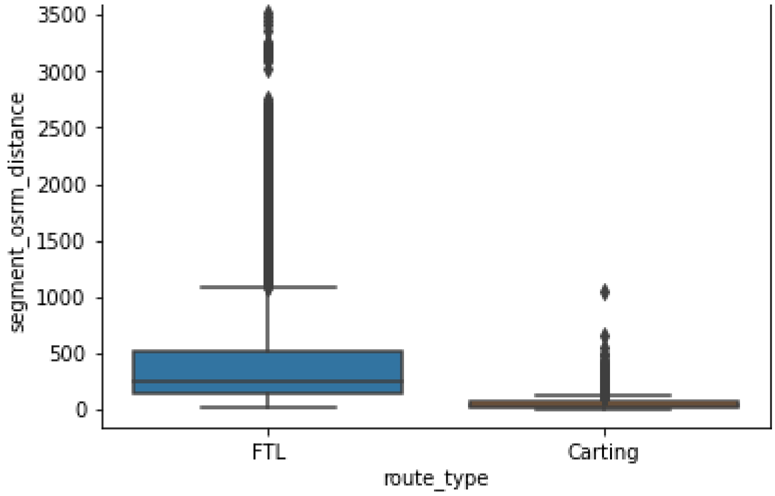
AxesSubplot(0.125,0.125;0.775x0.755)



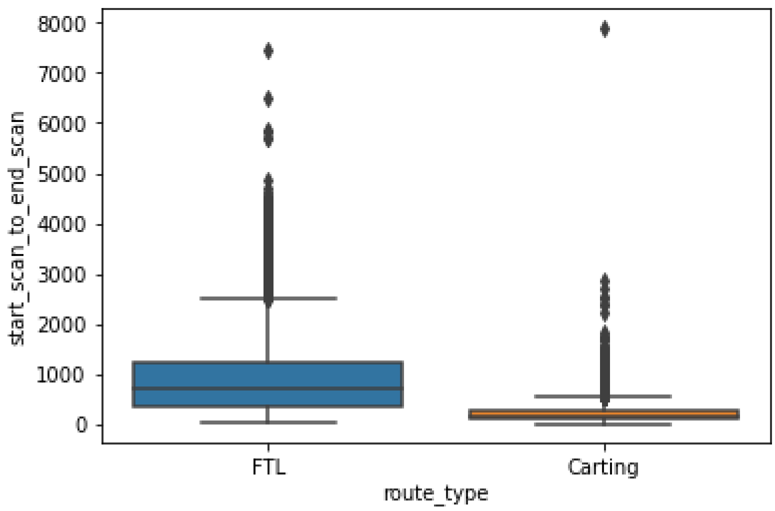
AxesSubplot(0.125,0.125;0.775x0.755)



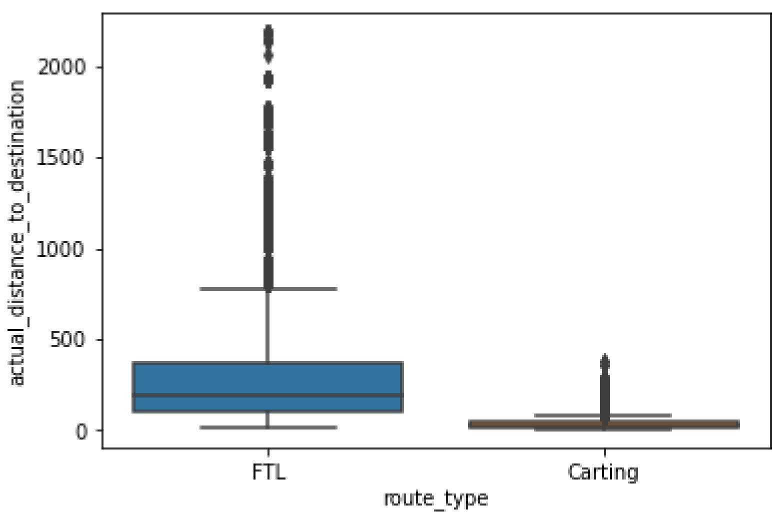
AxesSubplot(0.125,0.125;0.775x0.755)



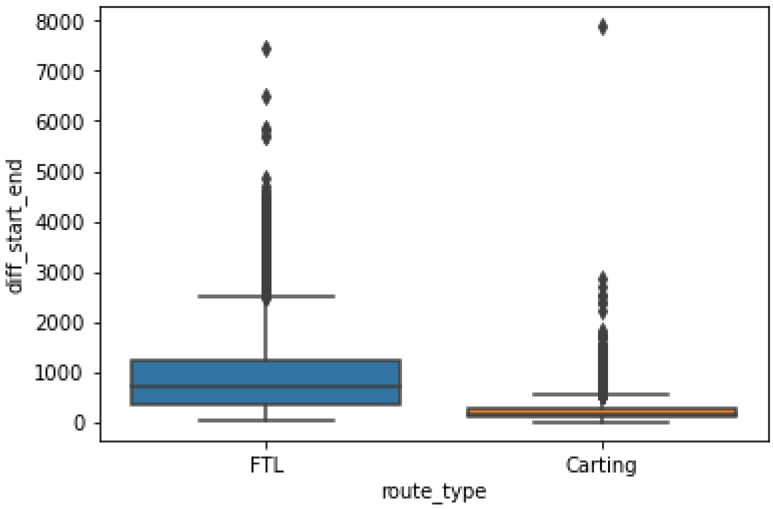

```
AxesSubplot(0.125,0.125;0.775x0.755)
```




```
AxesSubplot(0.125,0.125;0.775x0.755)
```

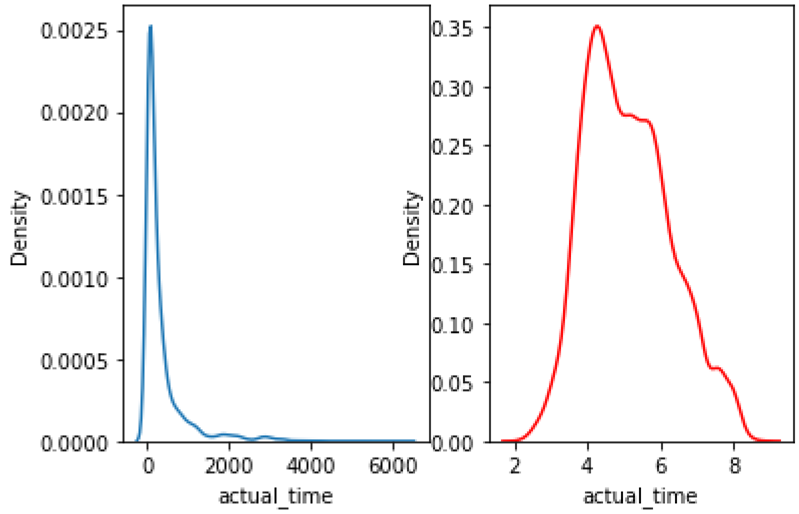


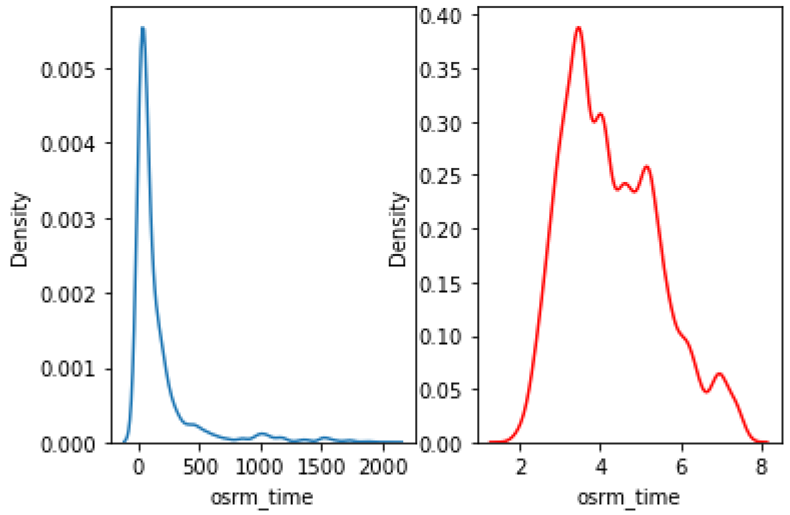
```
AxesSubplot(0.125,0.125;0.775x0.755)
```

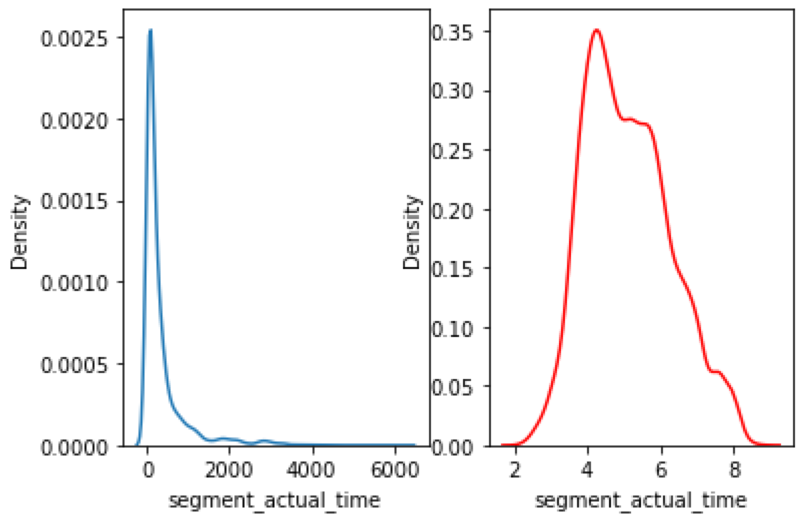


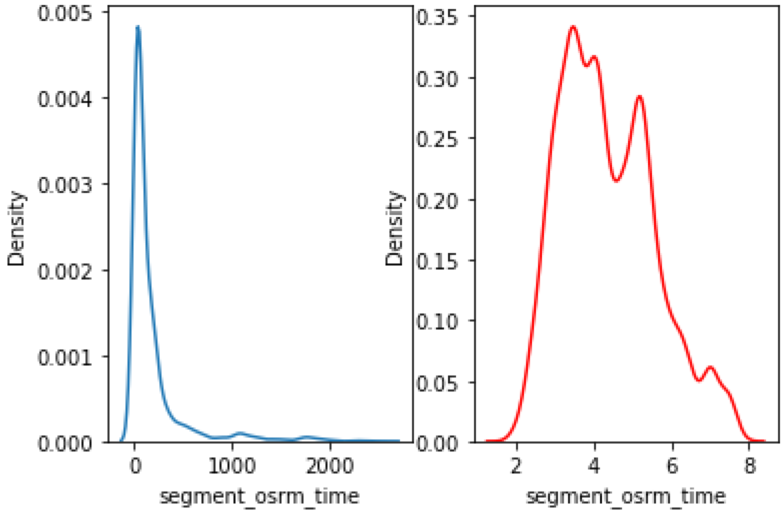
In [34]:

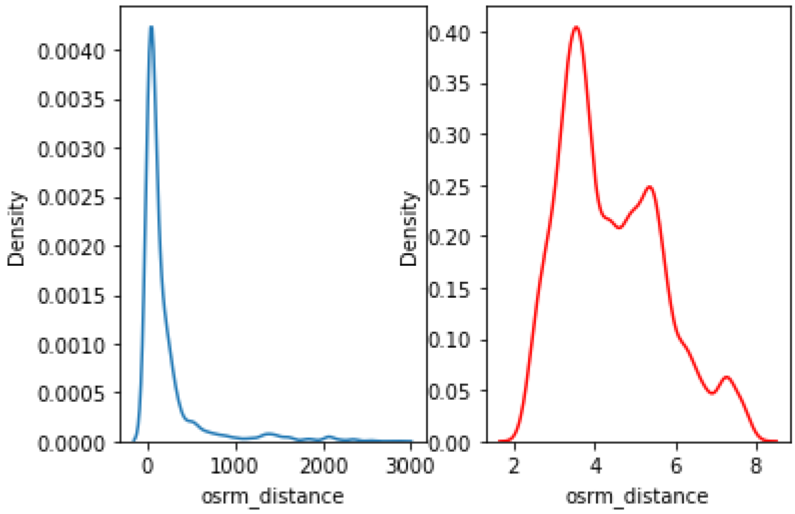
```
for col in cols:

    plt.subplot(121)
    sns.distplot(data[col],hist=False)

    plt.subplot(122)

    sns.distplot(data[col].apply(lambda x : np.log(x)),kde=True,hist=False,color
    plt.show()
```

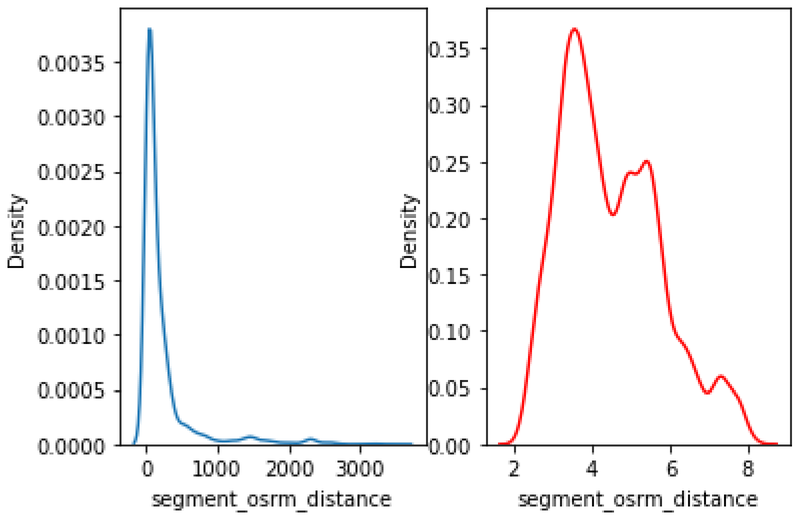


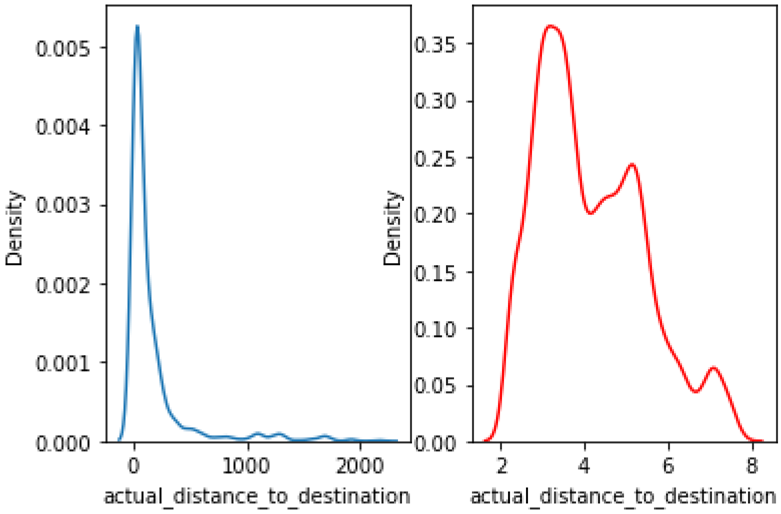


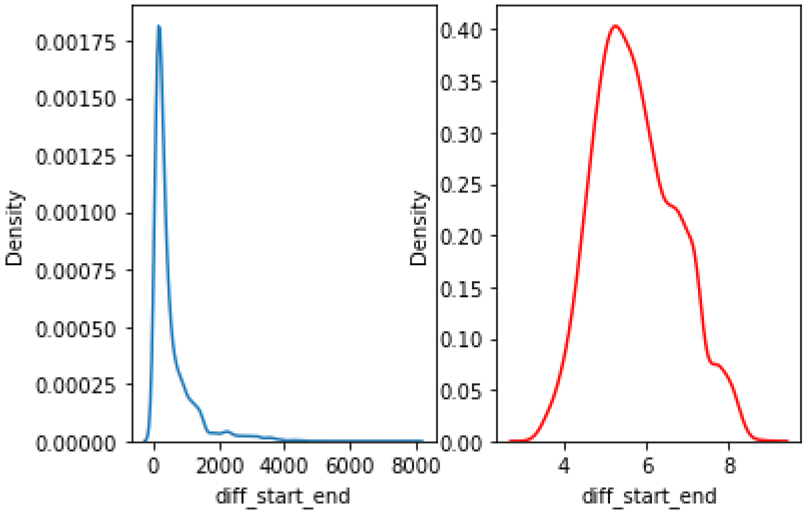



0.005
0.004
0.003
0.002
0.001
0.000
Density
0
1000
2000
segment_osrm_time
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
2
4
6
8
segment_osrm_time


0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
0
1000
2000
3000
osrm_distance
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
2
4
6
8
osrm_distance


0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
0
1000
2000
3000
segment_osrm_distance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
2
4
6
8
segment_osrm_distance





Insight

- Almost every variable follows log-normal distribution

In [35]:

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

Out[35]:

```
18    791
15    783
13    750
12    747
22    740
21    740
17    722
14    712
20    704
25    697
26    685
19    676
24    660
27    652
23    631
3     631
16    616
28    608
29    607
1     605
2     552
30    508
Name: trip_day, dtype: int64
```

In [36]:

```
data.head()
```

Out[36]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	1562.0	717.0	1548.0	1008.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	3347.0	1740.0	3308.0	1941.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

In [37]:

```
#creating bins based on the day of month the trip occured
bins = [1,10,20,31]
group = ['start','middle','end']
data["part_of_month"] = pd.cut(data["trip_day"],bins,labels=group)
```

In [38]:

```
data["part_of_month"].value_counts()
```

Out[38]:

```
end       6528
middle    6501
start     1183
Name: part_of_month, dtype: int64
```

Insight

- we see that maximum number of trips occured in the end of the month

actual_time vs osrm_time

Hypothesis Testing for actual_time and osrm_time

This is to infer if there is any significant difference between the actual_time and 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)

H0 :

There is no significant difference between actual_time and osrm_time

H1 :

There is significant difference between actual_time and osrm_time

significance level (alpha = 0.95)

In [39]:

```
from scipy.stats import ttest_ind
```

In [40]:

```

ttest_ind(data['actual_time'],data['osrm_time'])
```

Out[40]:

```
Ttest_indResult(statistic=38.215453905833165, pvalue=0.0)
```

- Since p-value is almost 0 we can conclude that there is significant difference between actual_time and osrm_time
- Let's see which is greater

In [41]:

```
# H0 : actual_time > osrm_time
# H1 : actual_time < osrm_time
ttest_ind(data['actual_time'],data['osrm_time'],alternative = 'less')
```

Out[41]:

```
Ttest_indResult(statistic=38.215453905833165, pvalue=1.0)
```

Insight

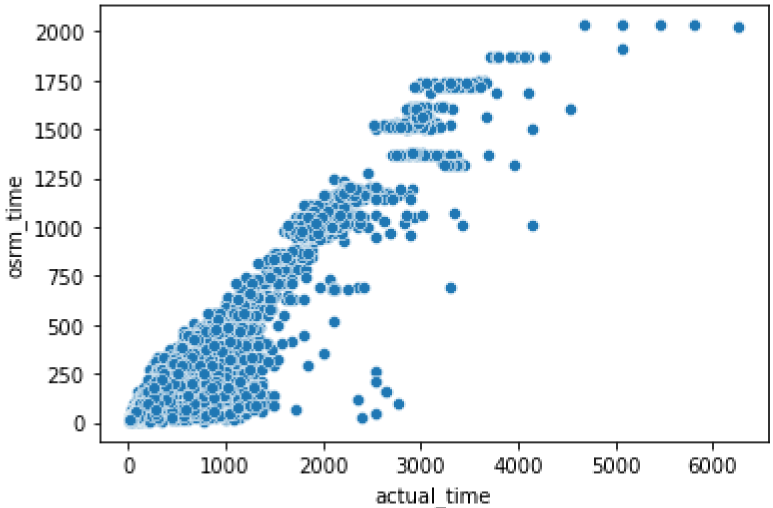
- Since p-value is greater than alpha we fail to reject null hypothesis and conculde that the mean of actual_time is greater than the mean of osrm_time

In [42]:

```
sns.scatterplot(data['actual_time'],data['osrm_time'])
```

Out[42]:

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



In [ ]:

```

```

actual_time and segment actual time

Hypothesis Testing for actual_time and segment_actual_time

This is to infer if there is any significant difference between the actual_time and segment_actual_time

H0 :

There is no significant difference between actual_time and segment_actual_time

H1 :

There is significant difference between actual_time and segment_actual_time

significance level (alpha = 0.95)

In [43]:

```
ttest_ind(data['actual_time'],data['segment_actual_time'])
```

Out[43]:

```
Ttest_indResult(statistic=0.5008024728897531, pvalue=0.616513864822477
2)
```

Insight

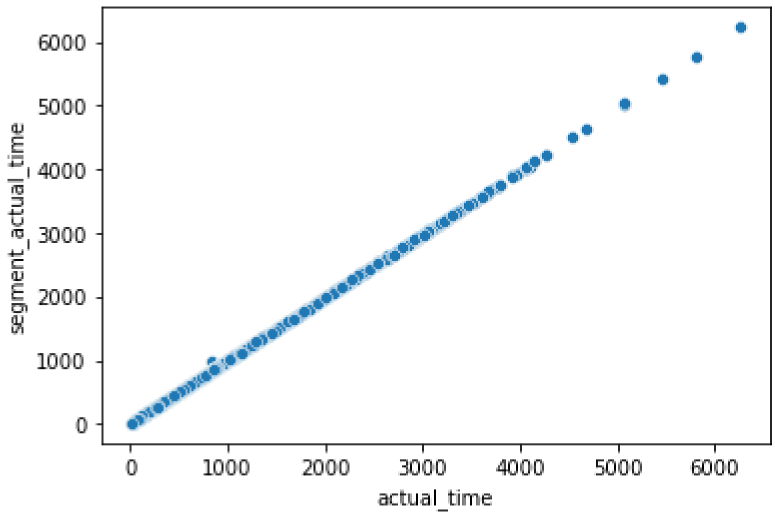
- Since p-value is greater than alpha we fail to regject null hypothesis and conclude that there is no significant difference between actual_time and segment_actual_time

In [44]:

```
sns.scatterplot(data['actual_time'],data['segment_actual_time'])
```

Out[44]:

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



osrm distance and segment osrm distance

Hypothesis Testing for osrm_distance and segment_osrm_distance

This is to infer if there is any significant difference between the osrm_distance and segment_osrm_distance

H0 :

There is no significant difference between osrm_distance and segment_osrm_distance

H1 :

There is significant difference between osrm_distance and segment_osrm_distance

There is significant difference between osrm_distance and segment_osrm_distance

significance level (alpha = 0.95)

In [45]:

```
ttest_ind(data['osrm_distance'],data['segment_osrm_distance'])
```

Out[45]:

```
Ttest_indResult(statistic=-4.117367046483823, pvalue=3.842631473353718
e-05)
```

Insight

- Since p-value is less than alpha we reject null hypothesis and conclude that there is a significant difference in the means of osrm_distance and segment_osrm_distance

In [46]:

```
# H0 : osrm_distance > segment_osrm_distance
# H1 : osrm_distance =< segment_osrm_distance
ttest_ind(data['osrm_distance'],data['segment_osrm_distance'],alternative = 'les
```

Out[46]:

```
Ttest_indResult(statistic=-4.117367046483823, pvalue=1.921315736676859
e-05)
```

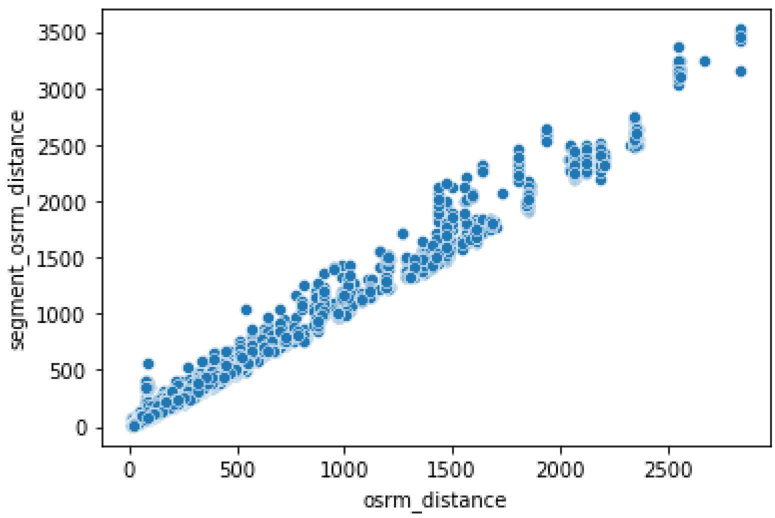
- Since p-value is less than alpha we reject null hypothesis and conclude that segment_osrm_distance is greater than the osrm_distance

In [47]:

```
sns.scatterplot(data['osrm_distance'],data['segment_osrm_distance'])
```

Out[47]:

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



osrm time and segment osrm time

Hypothesis Testing for osrm_time and segment_osrm_time

This is to infer if there is any significant difference between the osrm_time and segment_osrm_time

H0 :

There is no significant difference between osrm_time and segment_osrm_time

H1 :

There is significant difference between osrm_time and segment_osrm_time

significance level (alpha = 0.95)

In [48]:

```
ttest_ind(data['osrm_time'],data['segment_osrm_time'])
```

Out[48]:

```
Ttest_indResult(statistic=-5.733106696963521, pvalue=9.956426798219171
e-09)
```

Insight

- Since p-value is less than alpha we reject null hypothesis and conclude that there is a significant difference in the means of osrm_time and segment_osrm_time

In [49]:

```
# H0 : osrm_time > segment_osrm_time
# H1 : osrm_time =< segment_osrm_time
ttest_ind(data['osrm_time'],data['segment_osrm_time'],alternative = 'less')
```

Out[49]:

```
Ttest_indResult(statistic=-5.733106696963521, pvalue=4.978213399109586
e-09)
```

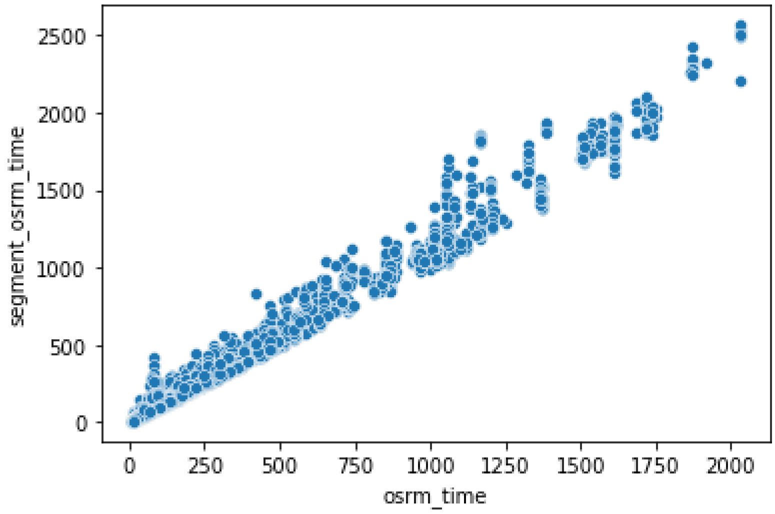
- Since p-value is less than alpha we reject null hypothesis and conclude that segment_osrm_time is greater than the osrm_time

In [50]:

```
sns.scatterplot(data['osrm_time'],data['segment_osrm_time'])
```

Out[50]:

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



start_scan_to_end_scan vs diff_start_end

- start_scan_to_end_scan is the total time taken for a product to reach its final destination from the initial source and diff_start_end is the calculated time by aggregating the duration of various destinations from initila source to final destination

Hypothesis Testing for start_scan_to_end_scan and diff_start_end

This is to infer if there is any significant difference between the start_scan_to_end_scan and diff_start_end

H0 :

There is no significant difference between start_scan_to_end_scan and diff_start_end

H1 :

There is significant difference between start_scan_to_end_scan and diff_start_end

significance level (alpha = 0.95)

In [51]:

```
ttest_ind(data['start_scan_to_end_scan'],data['diff_start_end'])
```

Out[51]:

```
Ttest_indResult(statistic=-0.11598577555478516, pvalue=0.9076646011346
594)
```

Insight

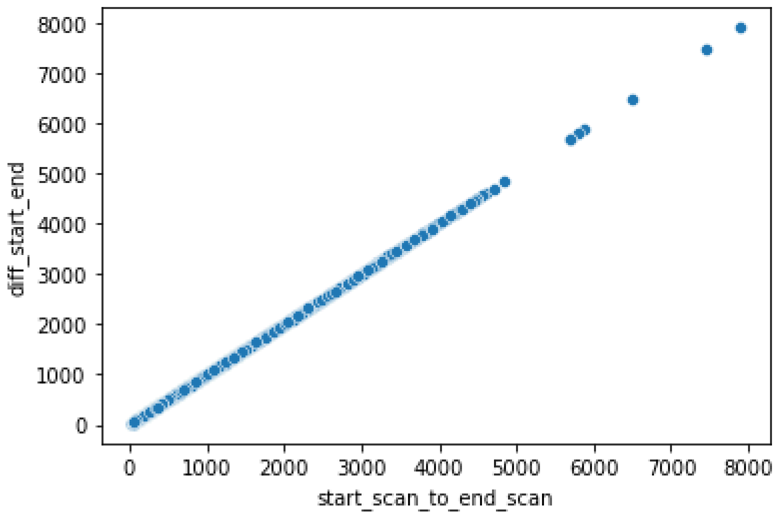
- Since p-value is greater than alpha we fail to reject null hypothesis and conclude that there is no significant difference in the total time as per the scanned records and the calulated time by aggregating the time between various destinations

In [52]:

```
sns.scatterplot(data['start_scan_to_end_scan'],data['diff_start_end'])
```

Out[52]:

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



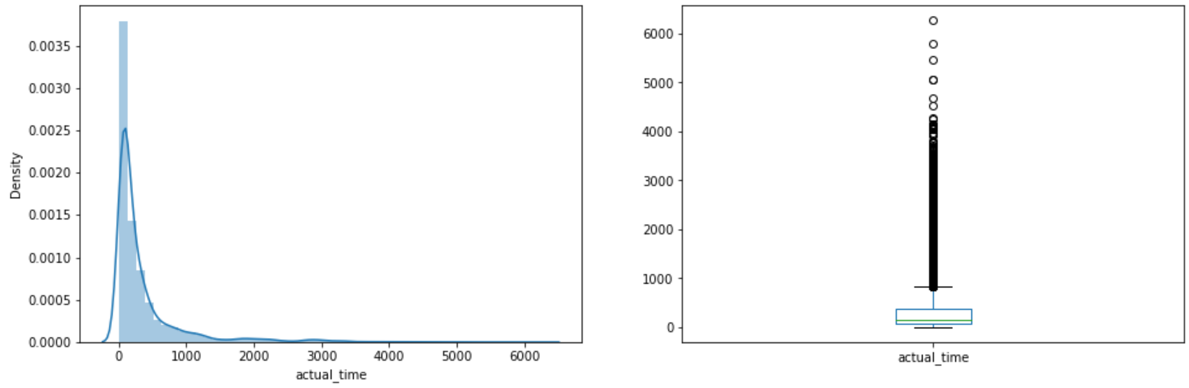
Outlier Treatment

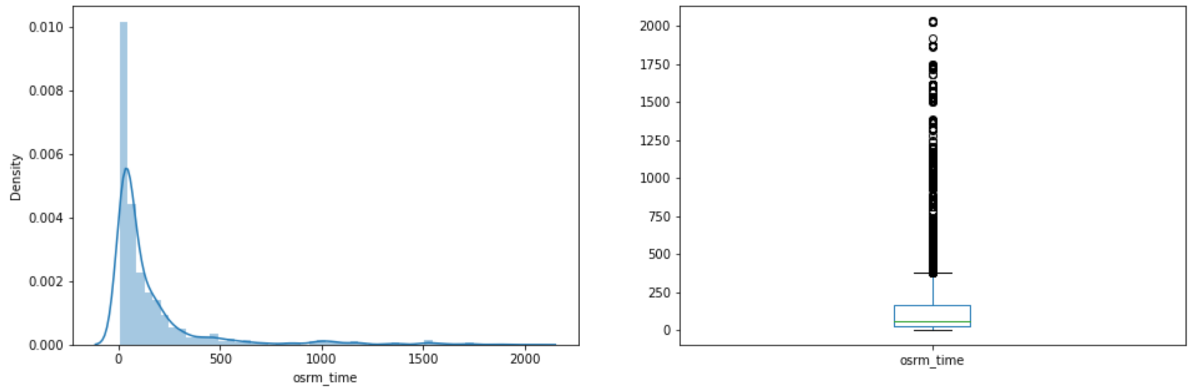
In [53]:

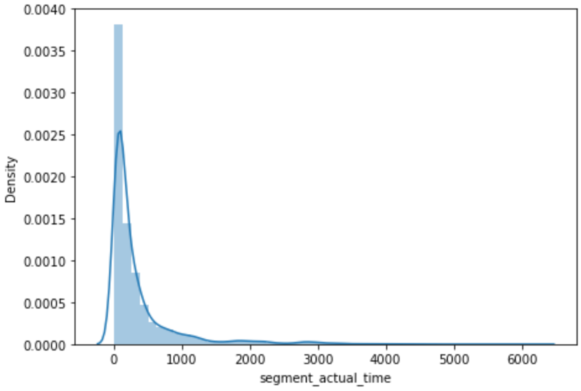
```
cols = ['actual_time','osrm_time','segment_actual_time','segment_osrm_time','osr
        'segment_osrm_distance','start_scan_to_end_scan','actual_distance_to_de
        'diff_start_end']
```

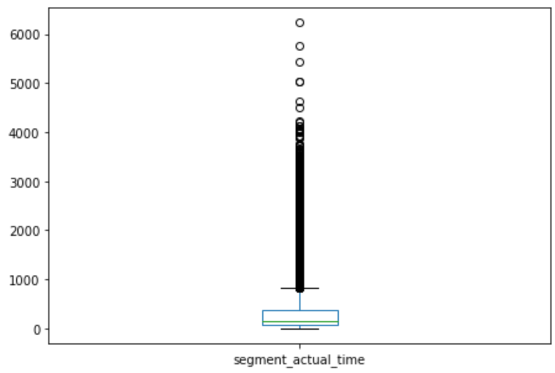
In [54]:

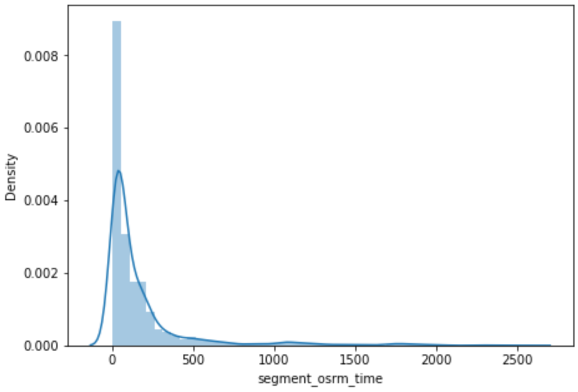
```
for col in cols:

    plt.subplot(121)

    sns.distplot(data[col])

    plt.subplot(122)
    data[col].plot.box(figsize=(16,5))
    plt.show()
```

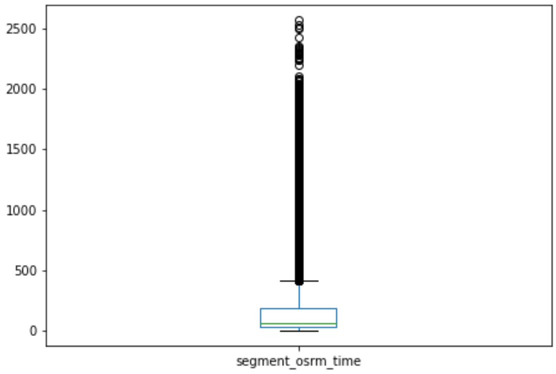


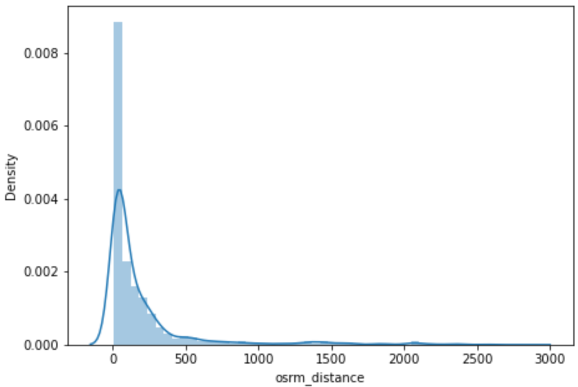


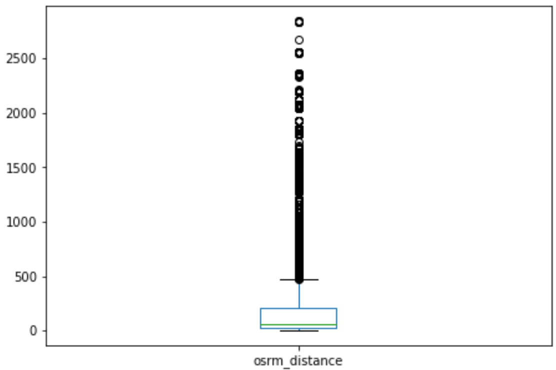

Density
segment_actual_time

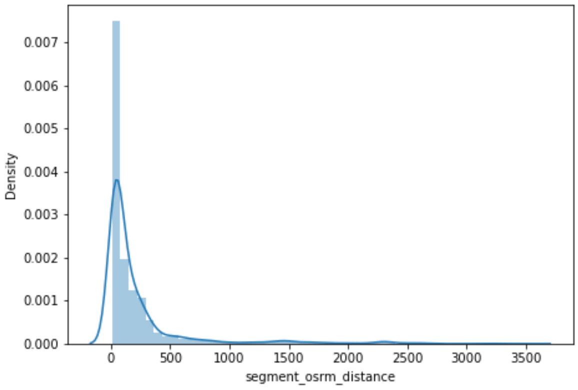

segment_actual_time

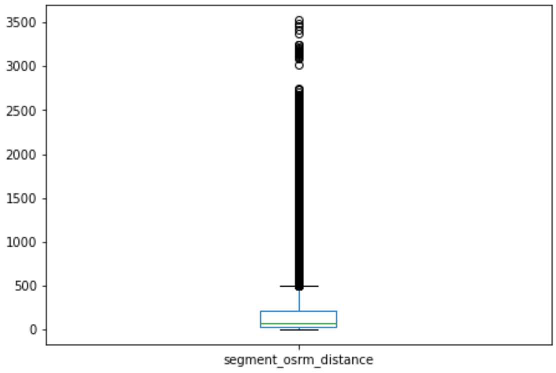

Density
segment_osrm_time

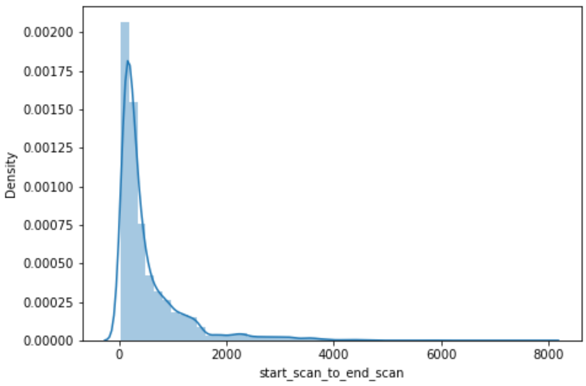

segment_osrm_time

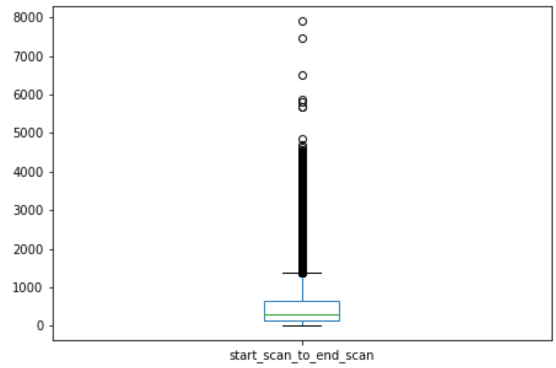

Density
osrm_distance

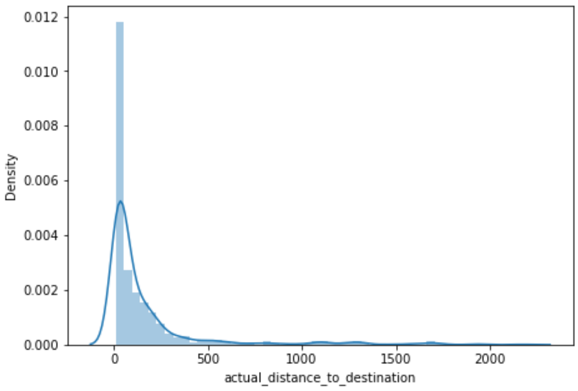

osrm_distance

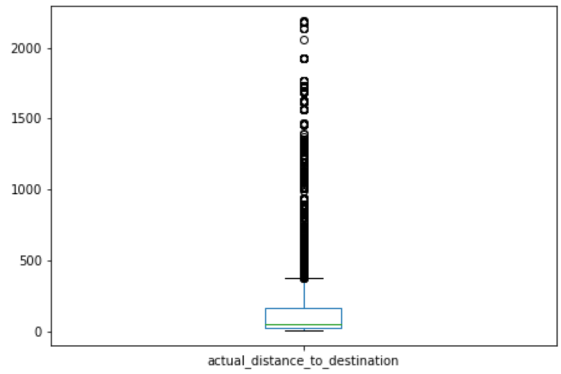

Density
segment_osrm_distance

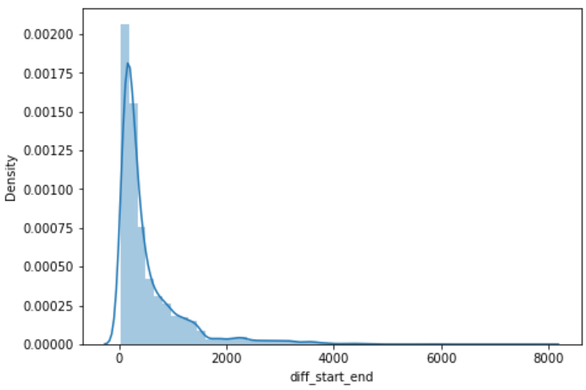

segment_osrm_distance

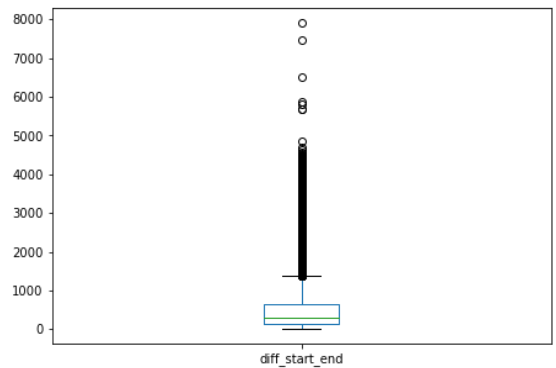

Density
start_scan_to_end_scan


start_scan_to_end_scan


Density
actual_distance_to_destination

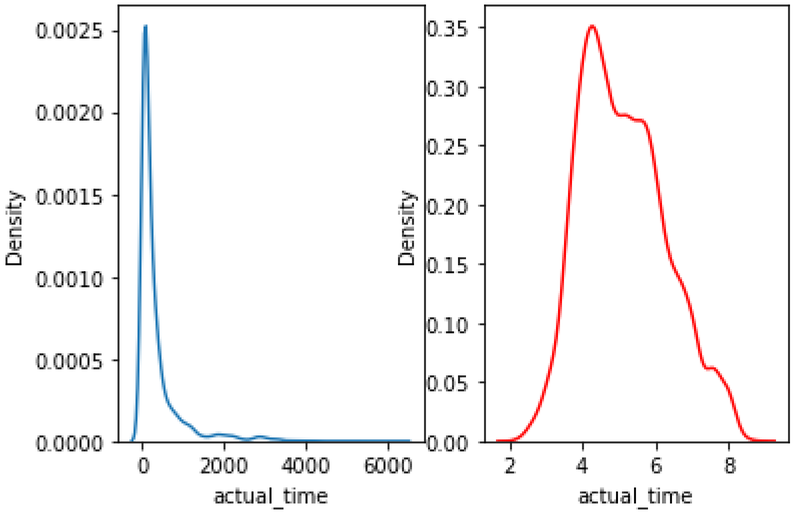

actual_distance_to_destination

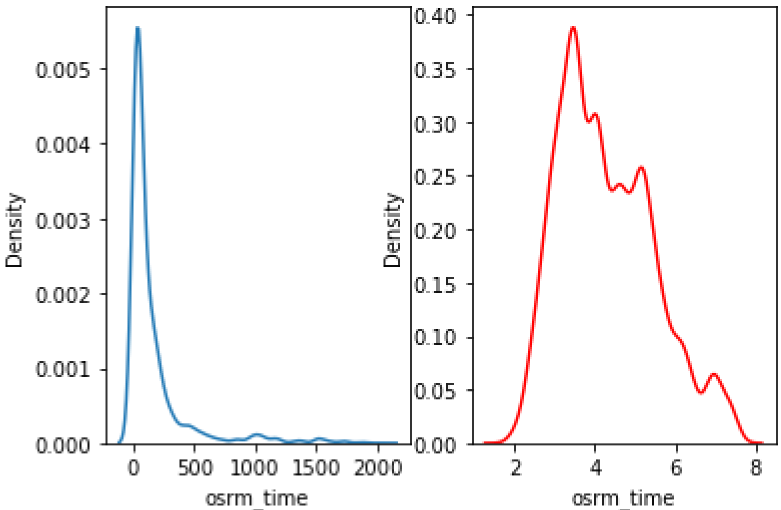

Density
diff_start_end

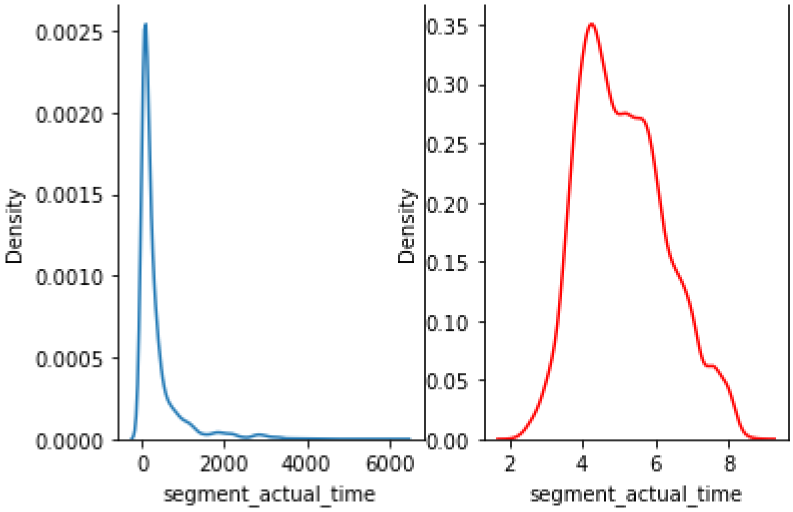

diff_start_end

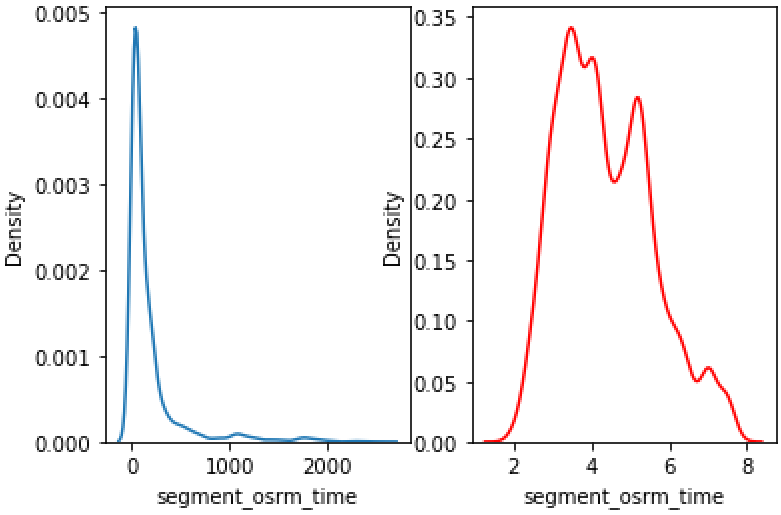
In [55]:

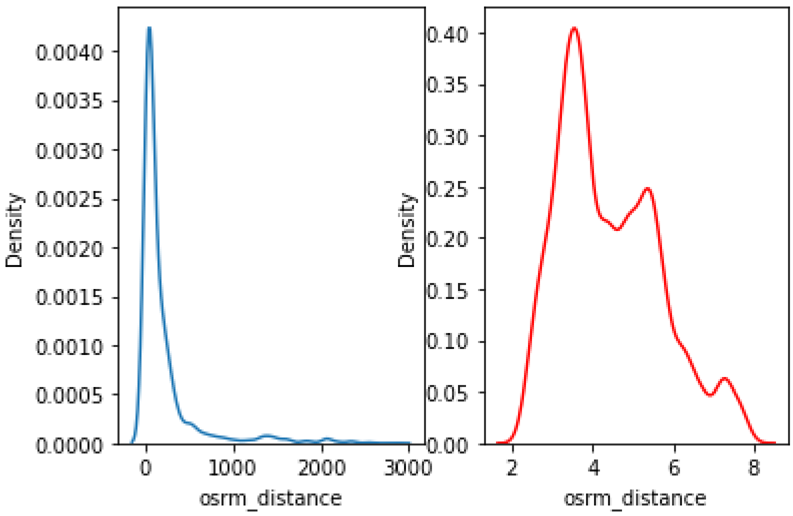
```
for col in cols:

    plt.subplot(121)
    sns.distplot(data[col],hist=False)

    plt.subplot(122)

    sns.distplot(data[col].apply(lambda x : np.log(x)),kde=True,hist=False,color
    plt.show()
```

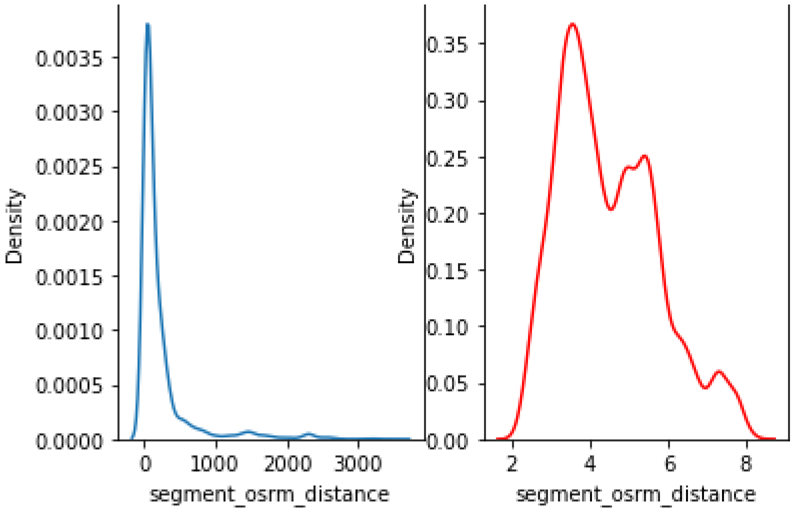


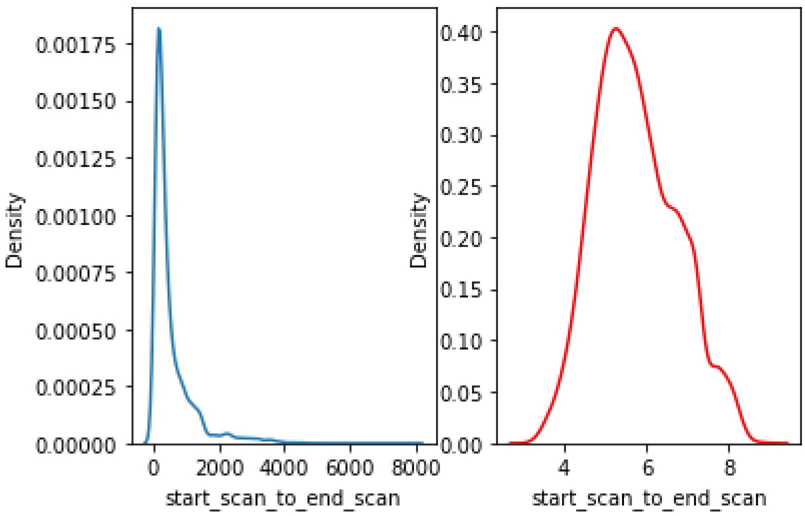


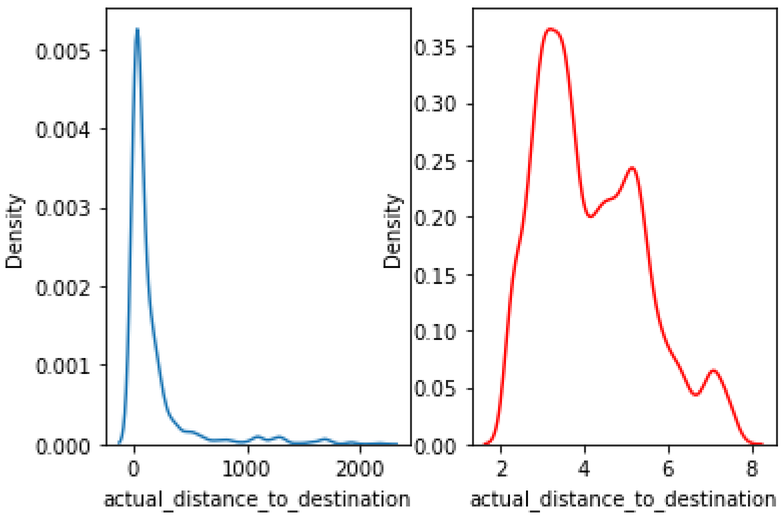

Density
segment_actual_time
Density
segment_actual_time

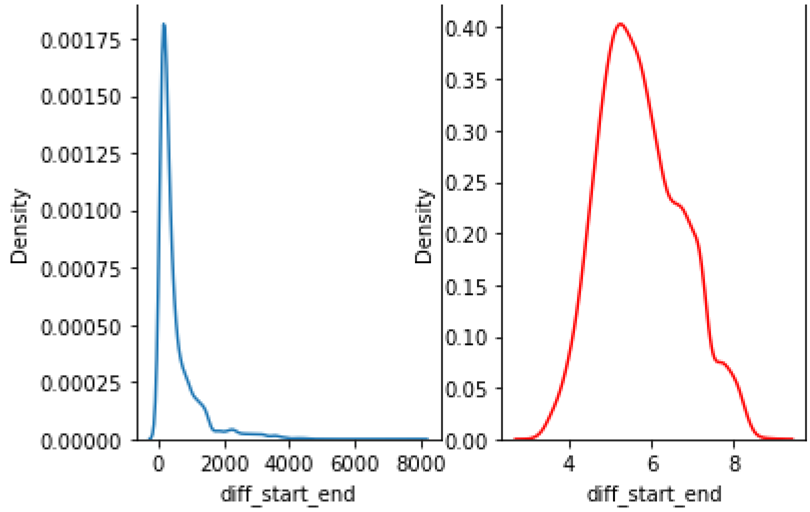

Density
segment_osrm_time
Density
segment_osrm_time


Density
osrm_distance
Density
osrm_distance


Density
segment_osrm_distance
Density
segment_osrm_distance


Density
start_scan_to_end_scan
Density
start_scan_to_end_scan


Density
actual_distance_to_destination
Density
actual_distance_to_destination



In [56]:

```python
from numpy import percentile
for col in cols:
    q25, q75 = percentile(data[col], 25), percentile(data[col], 75)
    iqr = q75 - q25
    cut_off = iqr * 1.5
    lower, upper = q25 - cut_off, q75 + cut_off
    outliers = [x for x in data[col] if x < lower or x > upper]
    print(f'The percentage of outliers in {col} is {len(outliers)/len(data[col])
```

```
The percentage of outliers in actual_time is 0.11088614429371668
The percentage of outliers in osrm_time is 0.10238239859620706
The percentage of outliers in segment_actual_time is 0.110886144293716
68
The percentage of outliers in segment_osrm_time is 0.1006951474657488
The percentage of outliers in osrm_distance is 0.10285482891273537
The percentage of outliers in segment_osrm_distance is 0.1044745899979
7529
The percentage of outliers in start_scan_to_end_scan is 0.085509887291
62448
The percentage of outliers in actual_distance_to_destination is 0.0977
930755213606
The percentage of outliers in diff_start_end is 0.08544239724640615
```

- If we drop the outliers we end up loosing lot of useful information. For example if we have an outlier in actual_time then we have to delete the entire row corresponding to that value. But all the other values in that row might not be outliers. Thus we end up loosing a lot of useful information.
- so instead of dropping the outliers we replace them with q25 - cut_off, q75 + cut_off where q25,q75 are 25th percentile and 75th percentile values respectively

In [57]:

```
# replacing all the outliers with q25 - cut_off, q75 + cut_off where cutoff = 1.
for col in cols:
        q25, q75 = percentile(data[col], 25), percentile(data[col], 75)
        iqr = q75 - q25
        cut_off = iqr * 1.5
        lower, upper = q25 - cut_off, q75 + cut_off
        data[col] = data[col].apply(lambda x : lower if x < lower else upper if

```

In [58]:

```
data.head()
```

Out[58]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	824.5	376.5	818.5	416.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	824.5	376.5	818.5	416.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

Relationship among different features

In [59]:

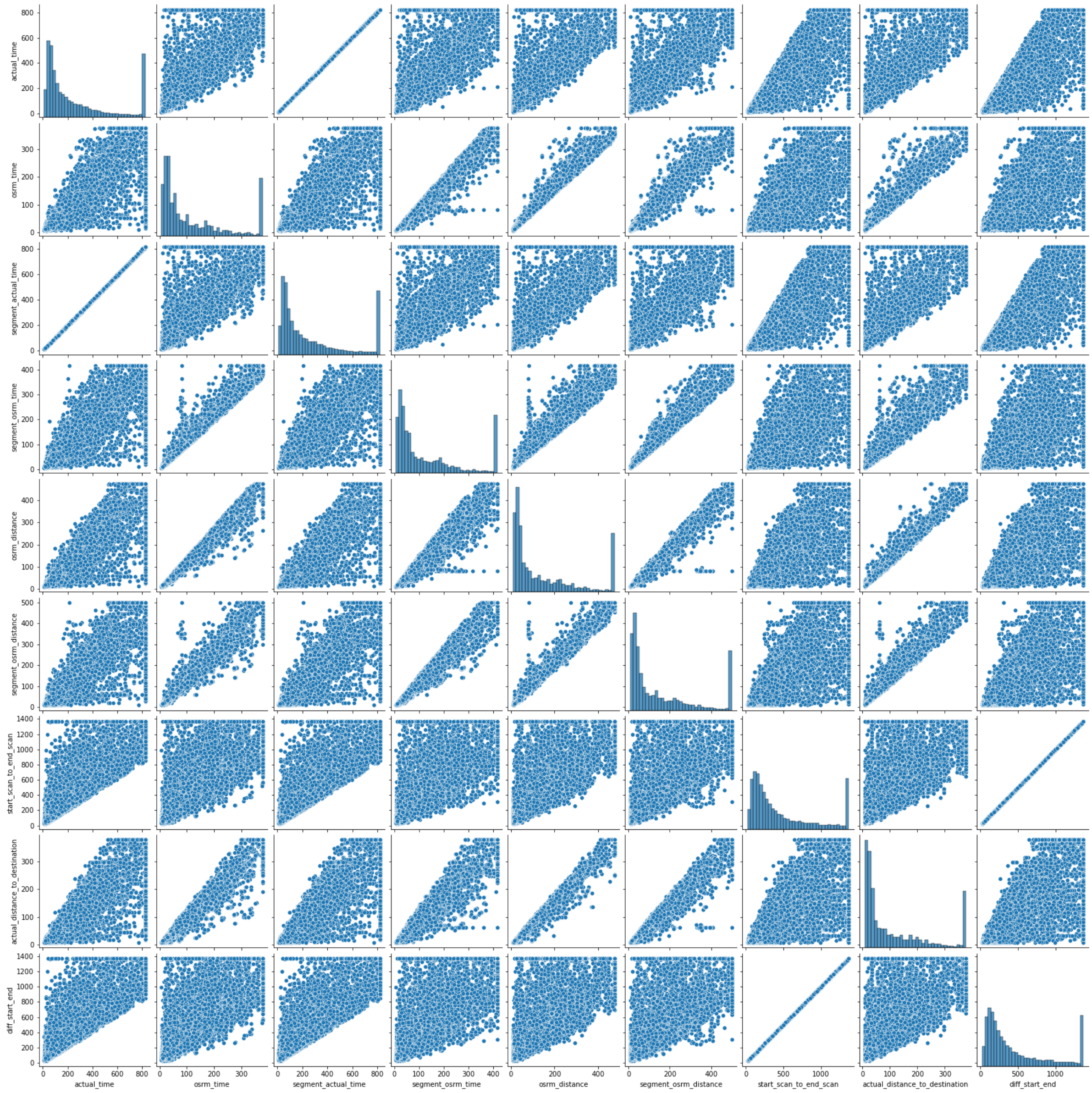
```
rel = data[['actual_time','osrm_time','segment_actual_time','segment_osrm_time',
         'segment_osrm_distance','start_scan_to_end_scan','actual_distance_to_de
         'diff_start_end']]
```

In [60]:

```
1 sns.pairplot(rel)
```

Out[60]:

<seaborn.axisgrid.PairGrid at 0x7fa348a5f3d0>

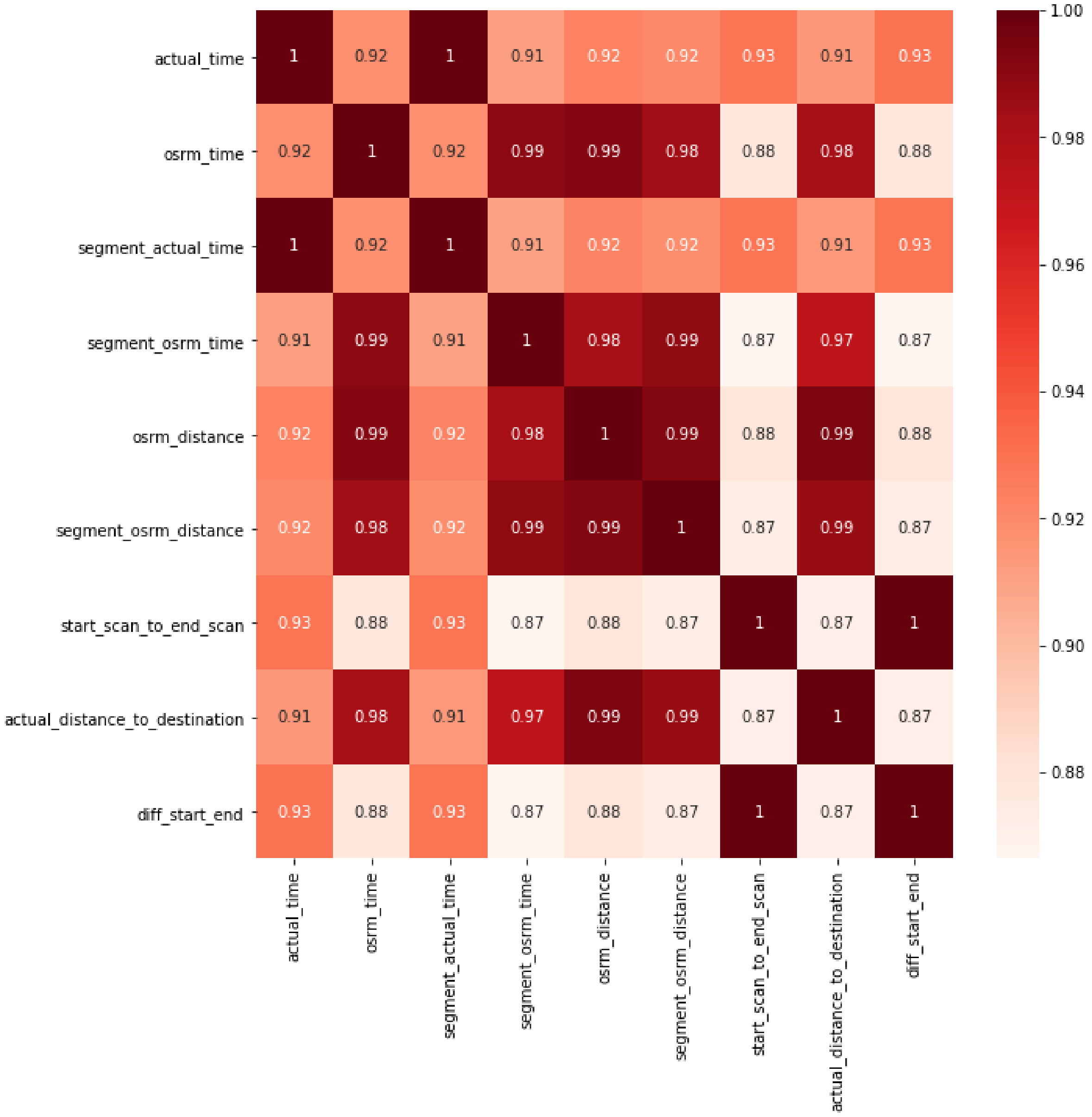


In [61]:

```
plt.figure(figsize = (10,10))
sns.heatmap(rel.corr(),cmap="Reds", annot=True)
```

Out[61]:

<AxesSubplot:>



A very huge multi-collinearity exists in the data

In [62]:

```
1 data.head()
```

Out[62]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	824.5	376.5	818.5	416.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	824.5	376.5	818.5	416.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

Handling categorical variables

- we have 3 categorical variables. route_type,destination_state,source_state

In [63]:

```
1 data['route_type'].value_counts()
```

Out[63]:

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

In [64]:

```
1 final_data = data.copy()
```

In [65]:

```
1 final_data['route_type'] = final_data['route_type'].apply(lambda x: 1 if x == 'C
```

In [66]:

```
1 final_data['route_type'].value_counts()
```

Out[66]:

```
1    8908
0    5909
Name: route_type, dtype: int64
```

In [67]:

```
final_data['destination_state'].value_counts()
```

Out[67]:

```
Maharashtra               2561
Karnataka                 2295
Haryana                   1643
Tamil Nadu                1084
Uttar Pradesh              819
Telangana                  784
Gujarat                    734
West Bengal                697
Delhi                      653
Punjab                     617
Rajasthan                  550
Andhra Pradesh             442
Bihar                      367
Madhya Pradesh             358
Kerala                     270
Assam                      232
Jharkhand                  181
Uttarakhand                122
Orissa                     119
Chandigarh                  65
Goa                         52
Chhattisgarh                43
Himachal Pradesh            42
Arunachal Pradesh           25
Jammu & Kashmir             20
Dadra and Nagar Haveli      17
Meghalaya                    8
Mizoram                      6
Daman & Diu                  1
Tripura                      1
Nagaland                     1
Name: destination_state, dtype: int64
```

In [68]:

```
#ignoring categories with frequency less than 300 and creating dummies  for n-1
k = final_data['destination_state'].value_counts()
k = k.index[k>300][:-1]
for i in k:
    name = 'destination'+'_'+i
    final_data[name] = (final_data['destination_state'] == i).astype(int)
```

In [69]:

```
final_data['source_state'].value_counts()
```

Out[69]:

```
Maharashtra               2654
Karnataka                 2270
Haryana                   1535
Tamil Nadu                1092
Uttar Pradesh              793
Gujarat                    757
Delhi                      720
Telangana                  719
West Bengal                639
Punjab                     547
Rajasthan                  500
Andhra Pradesh             428
Bihar                      378
Madhya Pradesh             376
Kerala                     297
Assam                      219
Jharkhand                  175
Orissa                     170
Uttarakhand                154
Himachal Pradesh           103
Chandigarh                  77
Goa                         47
Arunachal Pradesh           44
Chhattisgarh                43
Jammu & Kashmir             24
Dadra and Nagar Haveli      15
Meghalaya                   12
Pondicherry                  8
Mizoram                      5
Nagaland                     5
Tripura                      1
Name: source_state, dtype: int64
```

In [70]:

```
#ignoring categories with frequency less than 300 and creating dummies  for n-1
k = final_data['source_state'].value_counts()
k = k.index[k>300][:-1]
for i in k:
    name = 'source'+'_'+i
    final_data[name] = (final_data['source_state'] == i).astype(int)
```

In [71]:

```
final_data.drop(['destination_state','source_state'],axis=1,inplace=True)
```

In [72]:

```
final_data.head()
```

Out[72]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	824.5	376.5	818.5	416.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	824.5	376.5	818.5	416.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

In [73]:

```
final_data['trip_month'].value_counts()
```

Out[73]:

```
9     13029
10     1788
Name: trip_month, dtype: int64
```

- since all orders happened in the same year (2018) we ignore that.
- Also there are various days on which a delivery has happened, let's ignore this too and create dummy variable for just month

In [74]:

```
final_data['trip_month'] = final_data['trip_month'].apply(lambda x : 1 if x == 9
```

In [75]:

```
final_data.drop(['trip_day','trip_year'],axis=1,inplace=True)
```

In [76]:

```
final_data.head()
```

Out[76]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	824.5	376.5	818.5	416.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	824.5	376.5	818.5	416.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

In [77]:

```
# creating dummy variables for categories in part_of_month
k = final_data['part_of_month'].value_counts()
k = k.index[:-1]
for i in k:
    name = 'month'+'_'+i
    final_data[name] = (final_data['part_of_month'] == i).astype(int)
```

In [78]:

```
final_data.drop(['part_of_month'],axis=1,inplace=True)
```

In [ ]:

```

```

In [79]:

```
1 final_data.head()
```

Out[79]:

_West engal	source_Punjab	source_Rajasthan	source_Andhra Pradesh	source_Bihar	month_end	month_middle
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1

Top States based on the no.of orders

In [80]:

```
1 top_states = pd.DataFrame(data['destination_state'].value_counts()[0:7])
```

In [81]:

```
1 top_states = top_states.reset_index()
```

In [82]:

```
1 top_states
```

Out[82]:

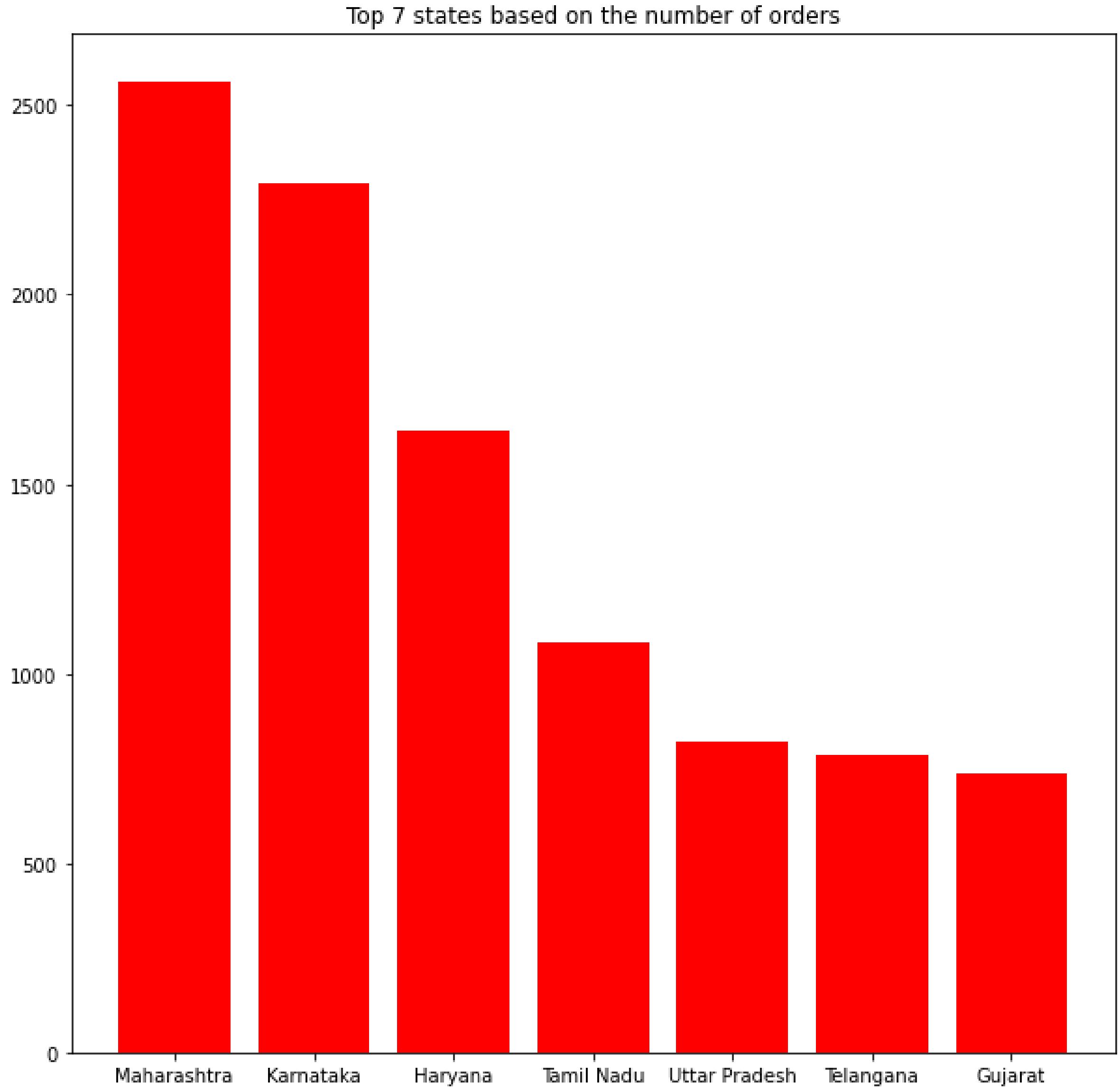
	index	destination_state
0	Maharashtra	2561
1	Karnataka	2295
2	Haryana	1643
3	Tamil Nadu	1084
4	Uttar Pradesh	819
5	Telangana	784
6	Gujarat	734

In [83]:

```
plt.figure(figsize=(10,10))
plt.bar(data=top_states,height = 'destination_state',x='index',color = 'r')
plt.title('Top 7 states based on the number of orders')
```

Out[83]:

```
Text(0.5, 1.0, 'Top 7 states based on the number of orders')
```



Insight

- we see that the maximum number of orders came from Maharahtra followed by Karnataka and Haryana

Busiest Corridor

- we define the busiest corridor as those two states where there are maximum number of trips.(ignoring the interstate trips)

In [84]:

```
df = data.copy()
```

In [85]:

```
df['col_for_count'] = '#'
```

In [86]:

```
df.head()
```

Out[86]:

	trip_uuid	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm
0	trip-153671041653548748	824.5	376.5	818.5	416.0	
1	trip-153671042288605164	143.0	68.0	141.0	65.0	
2	trip-153671043369099517	824.5	376.5	818.5	416.0	
3	trip-153671046011330457	59.0	15.0	59.0	16.0	
4	trip-153671052974046625	341.0	117.0	340.0	115.0	

In [87]:

```
busiest = df.groupby(['destination_state','source_state']).agg({'col_for_count':
                                                                'actual_distance_
                                                                'start_scan_to_er
```

In [88]:

```
busiest.shape
```

Out[88]:

```
(148, 3)
```

In [89]:

```
busiest = busiest.sort_values('col_for_count',ascending=False)
```

In [90]:

```
busiest = busiest.reset_index()
```

In [91]:

```
busiest_corridor = busiest[busiest['destination_state'] != busiest['source_state
```

In [92]:

```
busiest_corridor[0:10]
```

Out[92]:

	destination_state	source_state	col_for_count	actual_distance_to_destination	start_scan_to_e
8	Haryana	Delhi	437	78.199177	365
13	Delhi	Haryana	313	47.469693	248
21	Delhi	Uttar Pradesh	88	131.424602	504
22	Uttar Pradesh	Haryana	79	101.207251	397
23	Haryana	Punjab	79	283.542010	910
24	Punjab	Chandigarh	76	50.129700	251
25	Rajasthan	Haryana	68	175.012421	474
26	Chandigarh	Punjab	64	86.922130	318
27	Punjab	Himachal Pradesh	63	185.632284	918
28	Haryana	Uttar Pradesh	60	135.979619	494

- We can conclude that Haryana - Delhi is the busiest corridor, followed by Delhi - Uttar Pradesh

Standardization

In [93]:

```
X = final_data.drop('trip_uuid',axis=1)
```

In [94]:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
std_data = scaler.fit_transform(X)

```

In [95]:

```
std_data = pd.DataFrame(std_data, columns=X.columns)
std_data.head()
```

Out[95]:

	actual_time	osrm_time	segment_actual_time	segment_osrm_time	osrm_distance	segment_o
0	2.148616	2.249470	2.149256	2.256743	2.277563	
1	-0.463140	-0.403038	-0.465311	-0.475861	-0.361545	
2	2.148616	2.249470	2.149256	2.256743	2.277563	
3	-0.785059	-0.858737	-0.781760	-0.857336	-0.804486	
4	0.295668	0.018268	0.302658	-0.086601	0.056009	

Insights

- There is significant difference between actual_time and osrm_time and the mean of actual_time is greater than the mean of osrm_time
- There is no significant difference between actual_time and segment_actual_time
- There is a significant difference in the means of osrm_distance and segment_osrm_distance and segment_osrm_distance is greater than the osrm_distance
- There is a significant difference in the means of osrm_time and segment_osrm_time and segment_osrm_time is greater than the osrm_time.
- There is no significant difference in the total time as per the scanned records and the calulated time by aggregating the time between various destinations
- we see that maximum number of trips occured in the end of the month
- we see that the maximum number of orders came from Maharahtra followed by Karnataka and Haryana
- We can conclude that Haryana - Delhi is the busiest corridor, followed by Delhi - Uttar Pradesh

Recommendations

- Since Haryana is the busiest corridor we can optimize warehouse management for maximum productivity
- Since majority of orders came from Maharastra we can study the behaviour patterns of these orders to provide better supply chain solutions at the lowest costs
- we can have further data on expected delivery date and compare it with the product reaching destination date and study the reasons behind the delay in the delivery for the delayed deliveries and thus enhance customer experience.

- Since there is a significant difference in the actual_time and osrm_time(An open*source routing engine time calculator which computes the shortest path between points in a given map) we can further study the reasons dehind this difference and try to minimoze it
- we can further ask for the data the customer has received the order and calculate the gap between the customer receiving the order and the product being deliverd to the destination warehouse and check if the gap is huge. If the gap is huge we could further study the reasong behind late delivery although the product is shipped to the destination warehouse

In [ ]:

```

```