Problem Statement

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.

In this case study we'll perform the following tasks

- Feature Engineering and feature creation
- Handling Missing values
- Clean, sanitize and perform EDA
- Perform hypotheis testing
- Prepare the data for further data processing pipeline and model building

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.set_option('display.max_columns',None)
import seaborn as sns
from datetime import datetime as dt
from sklearn.preprocessing import OneHotEncoder
import scipy.stats as stats
import pylab
from sklearn import preprocessing
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
df = pd.read_csv('delhivery_data.txt')
```

In [6]:
```
df.head(5)
```

Out[6]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAE
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAE
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAE
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAE
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAE

In [7]:
```
df.shape
```

Out[7]:
```
(144867, 24)
```

In [8]:
```
df.info()
```

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

In [9]:
```
df.describe(include='all')
```

Out[9]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destinatior
count	144867	144867	144867	144867	144867	144867	144574	
unique	2	14817	1504	2	14817	1508	1498	
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND00(
freq	104858	101	1812	99660	101	23347	23347	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Handling Missing Values

In [10]:

```
df.fillna('nan',inplace=True)

source_ids = list(df[df['source_name']=='nan']['source_center'].unique())
destination_ids = list(df[df['destination_name']=='nan']['destination_center'].unique())
source_ids.extend(destination_ids)
missing_ids = list(set(source_ids))
missing_ids
```

Out[10]:
```
['IND122015AAC',
 'IND331022A1B',
 'IND509103AAC',
 'IND126116AAA',
 'IND577116AAA',
 'IND282002AAD',
 'IND342902A1B',
 'IND465333A1B',
 'IND841301AAC',
 'IND505326AAB',
 'IND221005A1A',
 'IND852118A1B',
 'IND331001A1C',
 'IND250002AAC']
```

In [11]:
```
len(missing_ids)
```

Out[11]:
```
14
```

In [12]:
```
mapper = {}
counter=0
def missing(source_name,source_center,destination_name,destination_center):
    if ((source_center in missing_ids) and (source_name!='nan')):
        mapper[source_center]=source_name
    if ((destination_center in missing_ids) and (destination_center!='nan')):
        mapper[destination_center]=destination_name

df.apply(lambda x: missing(x.source_name,x.source_center,x.destination_name,x.destination_name),axis=1)
print(len(mapper))
```

```
0
```

In [13]:
```
counter=0

for a in missing_ids:
    if a not in mapper:
        mapper[a]='location_'+str(counter)+ '('+str(counter)+')'
        counter+=1
mapper
```

Out[13]:
```
{'IND122015AAC': 'location_0(0)',
 'IND331022A1B': 'location_1(1)',
 'IND509103AAC': 'location_2(2)',
 'IND126116AAA': 'location_3(3)',
 'IND577116AAA': 'location_4(4)',
 'IND282002AAD': 'location_5(5)',
 'IND342902A1B': 'location_6(6)',
 'IND465333A1B': 'location_7(7)',
 'IND841301AAC': 'location_8(8)',
 'IND505326AAB': 'location_9(9)',
 'IND221005A1A': 'location_10(10)',
 'IND852118A1B': 'location_11(11)',
 'IND331001A1C': 'location_12(12)',
 'IND250002AAC': 'location_13(13)'}
```

In [14]:
```
def handling_missing(source_name,source_center,destination_name,destination_center):
    if source_center in mapper and source_name == 'nan':
        source_name=mapper[source_center]
    if destination_center in mapper and destination_name == 'nan':
        destination_name = mapper[destination_center]
    return source_name,destination_name
```

In [15]:
```
df[['source_name','destination_name']]=df.apply(lambda x: pd.Series((
    handling_missing(x.source_name,x.source_center,x.destination_name,x.destination_center)[0],
    handling_missing(x.source_name,x.source_center,x.destination_name,x.destination_center)[1])),
    axis=1)
```

In [16]:
```
df.isnull().sum()/df.shape[0]
```

Out[16]:
```
data                                0.0
trip_creation_time                  0.0
route_schedule_uuid                 0.0
route_type                          0.0
trip_uuid                           0.0
source_center                       0.0
source_name                         0.0
destination_center                  0.0
destination_name                    0.0
od_start_time                       0.0
od_end_time                         0.0
start_scan_to_end_scan              0.0
is_cutoff                           0.0
cutoff_factor                       0.0
cutoff_timestamp                    0.0
actual_distance_to_destination      0.0
actual_time                         0.0
osrm_time                           0.0
osrm_distance                       0.0
factor                              0.0
segment_actual_time                 0.0
segment_osrm_time                   0.0
segment_osrm_distance               0.0
segment_factor                      0.0
dtype: float64
```

Feature Engineering

In [17]:
```
def segregate_city_state(data):
    if data=='nan':
        return 'nan','nan'
    try:
        location,state = data.split('(')[0][:-1],data.split('(')[1][:-1]
        city = location.split('_')[0]
        return city,state
    except :
        print(data)

df[['source_city','source_state']]=df.apply(lambda x: pd.Series((segregate_city_state(x.source_name)[0],
                                                                segregate_city_state(x.source_name)[1])),
                                                                                    axis=1)
```

```
df[['destination_city','destination_state']]=df.apply(lambda x: pd.Series((segregate_city_state(x.destination_name)[0]
                                                                           ,segregate_city_state(x.destination_name)[1])),
                                                                                                   axis=1)
```

In [18]:
```
def segregate_delivery_time(data):
    year,month,date = data.split('-')[0],data.split('-')[1],data.split('-')[2][:3]
    return year,month,date

df[['trip_creation_year','trip_creation_month','trip_creation_date']]=df.apply(lambda x: pd.Series((
    segregate_delivery_time(x.trip_creation_time)[0],
    segregate_delivery_time(x.trip_creation_time)[1],
    segregate_delivery_time(x.trip_creation_time)[2])),
    axis=1)
```

In [19]:
```
df['od_total_time'] = (pd.to_datetime(df['od_end_time'])-pd.to_datetime(df['od_start_time'])).astype('timedelta64[m]')
```

In [20]:
```
df.head(10)
```

Out[20]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND38862(
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND38862(
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND38862(
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND38862(
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND38862(
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND38832(
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND38832(
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND38832(
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND38832(
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND38832(

Data Cleaning

In [21]:
```
# removing columns which are not required
remove_columns = ['data','od_start_time','od_end_time','route_schedule_uuid','source_name',
```

```
                  'destination_name','is_cutoff','cutoff_factor','cutoff_timestamp','factor',
                  'trip_creation_time','segment_factor']
df.drop(remove_columns,axis=1,inplace=True)
df.shape
```

Out[21]: (144867, 20)

In [22]:
```
column_order = ['trip_uuid','route_type','source_center','destination_center','source_state',
                'destination_state','source_city','destination_city','trip_creation_year','trip_creation_month',
                'trip_creation_date','start_scan_to_end_scan','od_total_time','actual_distance_to_destination',
                'actual_time','osrm_time','osrm_distance','segment_actual_time','segment_osrm_time',
                'segment_osrm_distance']
df = df[column_order]
```

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

Out[23]:

	trip_uuid	route_type	source_center	destination_center	source_state	destination_state	source_city	destination_city	trip_cre
0	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
1	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
2	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
3	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
4	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	

Merging Columns

In [24]:
```
data = df.copy()
```

In [25]:
```
data['segment_actual_time_cum'] = data.groupby(['trip_uuid','source_center',
                                                'destination_center'])['segment_actual_time'].cumsum()
data['segment_osrm_time_cum'] = data.groupby(['trip_uuid','source_center',
                                              'destination_center'])['segment_osrm_time'].cumsum()
```

```
data['segment_osrm_distance_cum'] = data.groupby(['trip_uuid','source_center',
                                                  'destination_center'])['segment_osrm_distance'].cumsum()
```

In [26]:
```
data.head()
```

Out[26]:

	trip_uuid	route_type	source_center	destination_center	source_state	destination_state	source_city	destination_city	trip_cre
0	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
1	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
2	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
3	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	
4	trip-153741093647649320	Carting	IND388121AAA	IND388620AAB	Gujarat	Gujarat	Anand	Khambhat	

In [27]:
```
cumulative_columns = ['route_type','trip_creation_year','trip_creation_month','trip_creation_date',
                      'od_total_time','start_scan_to_end_scan','actual_distance_to_destination',
                     'actual_time','osrm_time','osrm_distance','source_city','destination_city',
                      'source_state','destination_state','segment_actual_time_cum','segment_osrm_time_cum',
                      'segment_osrm_distance_cum']
agg_on_trip_source_dest = data.groupby(['trip_uuid','source_center','destination_center']).last()[cumulative_columns]
```

In [28]:
```
agg_on_trip_source_dest.head(10)
```

Out[28]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_ti
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	126
	IND462022AAA	IND209304AAA	FTL	2018	09	12	99
trip-153671042288605164	IND561203AAB	IND562101AAA	Carting	2018	09	12	5
	IND572101AAA	IND561203AAB	Carting	2018	09	12	12
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	83
	IND562132AAA	IND000000ACB	FTL	2018	09	12	309
trip-153671046011330457	IND400072AAB	IND401104AAA	Carting	2018	09	12	10
trip-153671052974046625	IND583101AAA	IND583201AAA	FTL	2018	09	12	15
	IND583119AAA	IND583101AAA	FTL	2018	09	12	48
	IND583201AAA	IND583119AAA	FTL	2018	09	12	8

Analysis

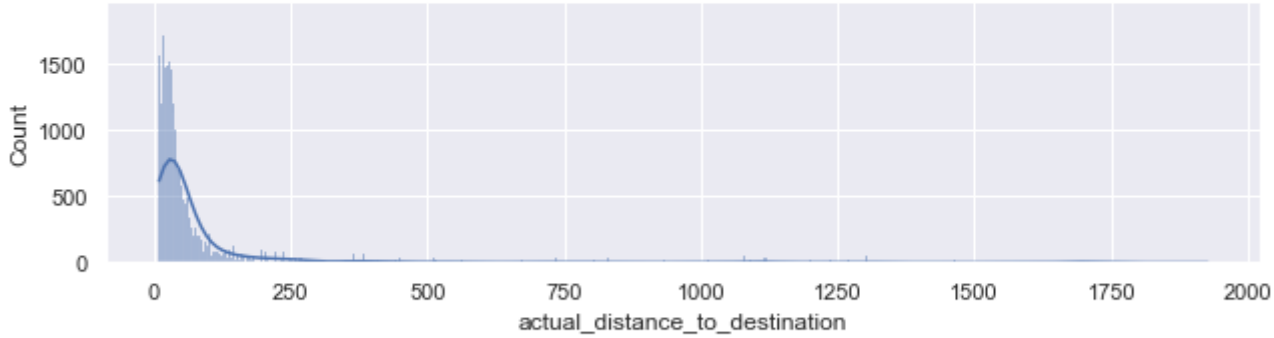
Distribution of continues variables

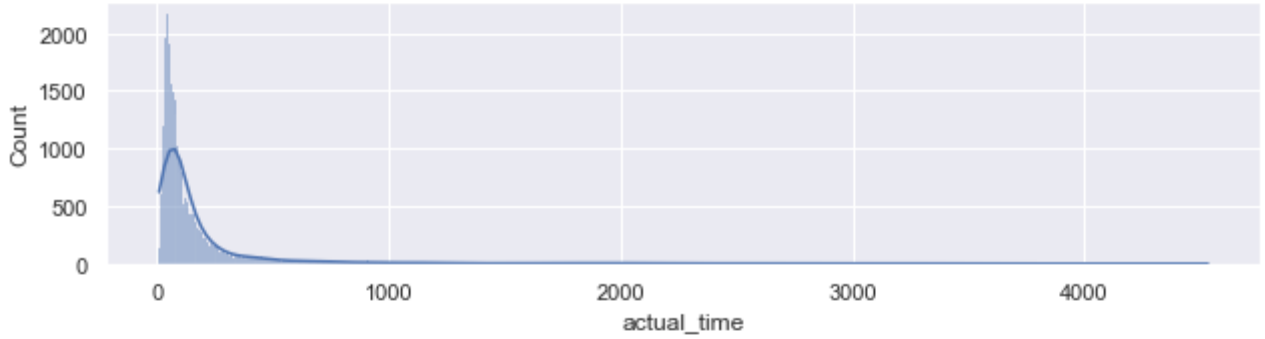
In [29]:

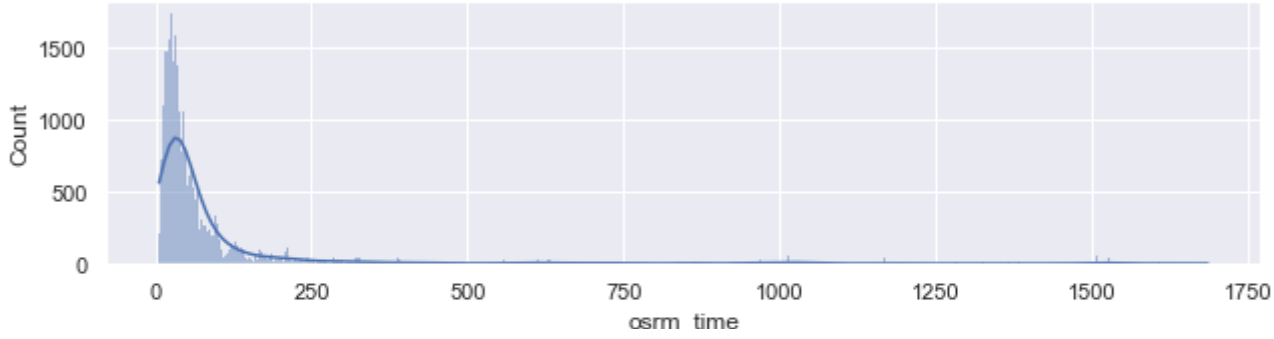
```
sns.set()
fig,axis= plt.subplots(5,1,figsize=(10,15))
sns.histplot(data = agg_on_trip_source_dest,x='start_scan_to_end_scan',kde=True,ax=axis[0])
sns.histplot(data=agg_on_trip_source_dest,x='actual_distance_to_destination',kde=True,ax=axis[1])
sns.histplot(data=agg_on_trip_source_dest,x='actual_time',kde=True,ax=axis[2])
sns.histplot(data=agg_on_trip_source_dest,x='osrm_time',kde=True,ax=axis[3])
sns.histplot(data=agg_on_trip_source_dest,x='osrm_distance',kde=True,ax=axis[4])

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
```

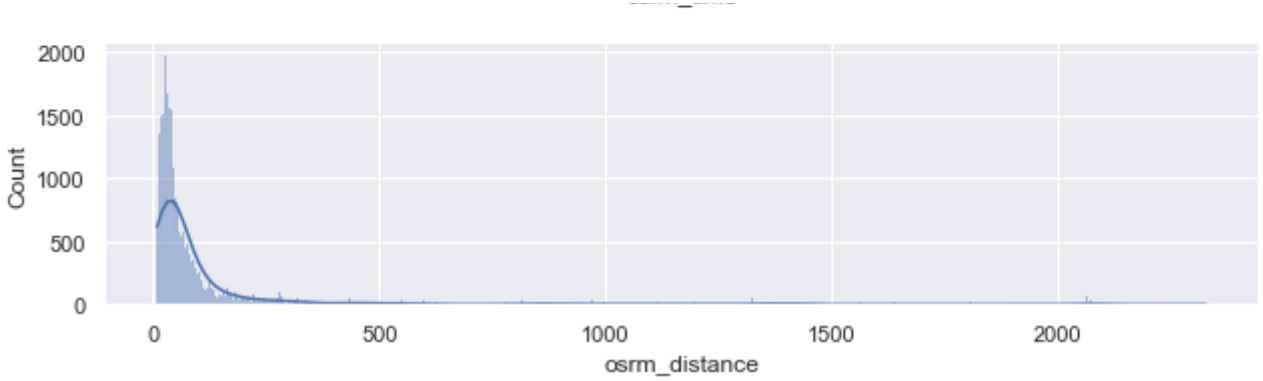
```
                    hspace=0.4)
plt.show()
```


Count
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan


Count
1500
1000
500
0
0
250
500
750
1000
1250
1500
1750
2000
actual_distance_to_destination


Count
2000
1500
1000
500
0
0
1000
2000
3000
4000
actual_time


Count
1500
1000
500
0
0
250
500
750
1000
1250
1500
1750
osrm_time

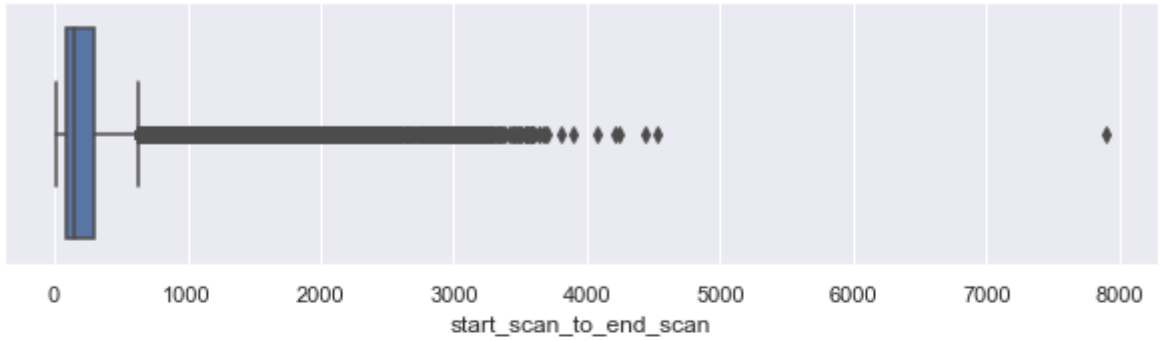


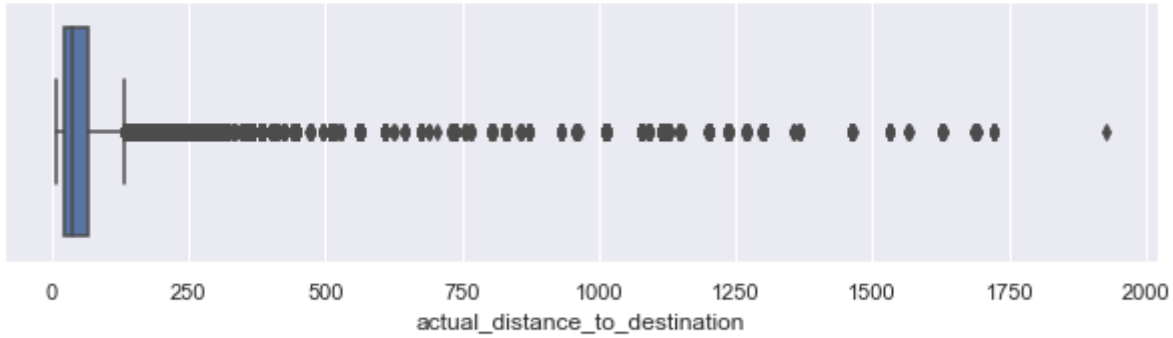
Outlier detection and Handling

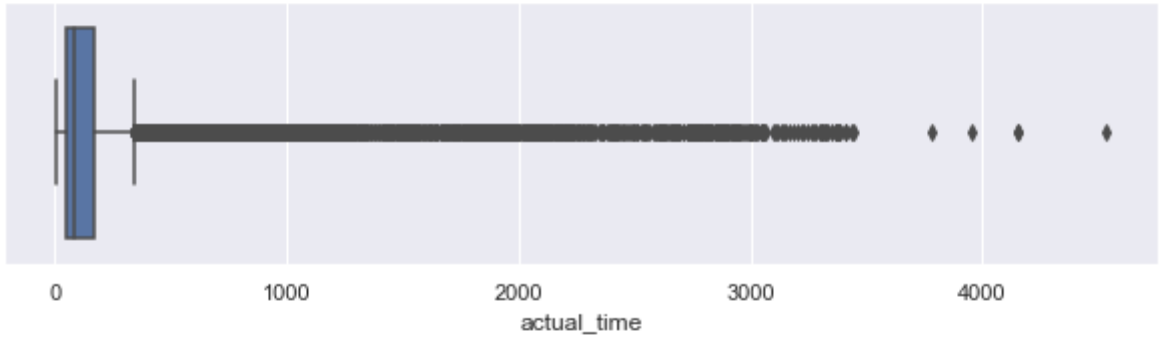
In [30]:
```
agg_on_trip_source_dest.columns
```

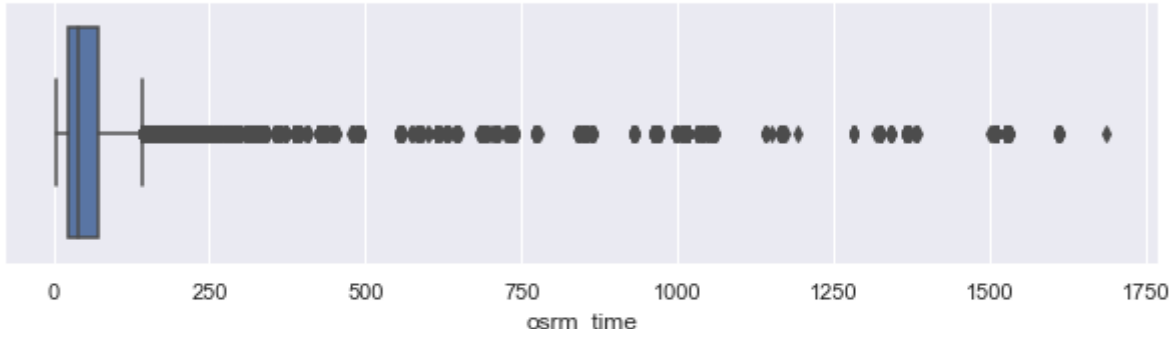
Out[30]:
```
Index(['route_type', 'trip_creation_year', 'trip_creation_month',
       'trip_creation_date', 'od_total_time', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'source_city', 'destination_city', 'source_state',
       'destination_state', 'segment_actual_time_cum', 'segment_osrm_time_cum',
       'segment_osrm_distance_cum'],
      dtype='object')
```

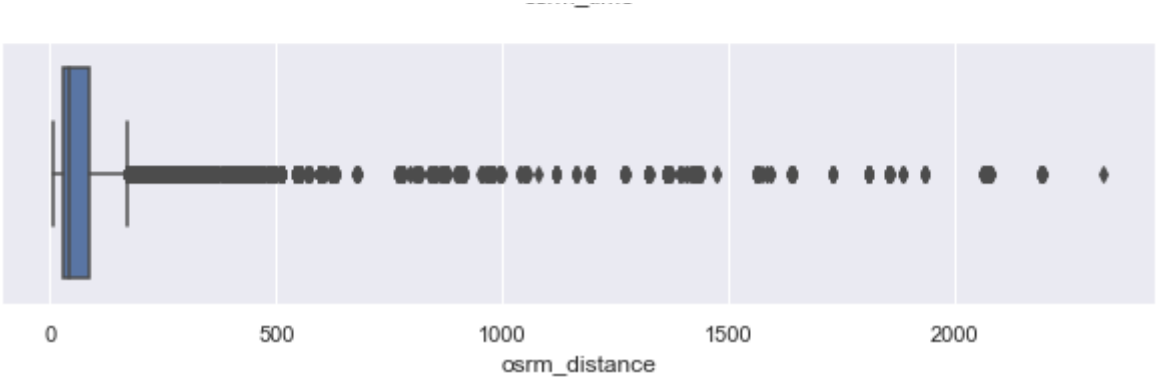
In [31]:
```
sns.set()
fig,axis= plt.subplots(5,1,figsize=(10,15))
sns.boxplot(data = agg_on_trip_source_dest,x='start_scan_to_end_scan',ax=axis[0])
sns.boxplot(data=agg_on_trip_source_dest,x='actual_distance_to_destination',ax=axis[1])
sns.boxplot(data=agg_on_trip_source_dest,x='actual_time',ax=axis[2])
sns.boxplot(data=agg_on_trip_source_dest,x='osrm_time',ax=axis[3])
sns.boxplot(data=agg_on_trip_source_dest,x='osrm_distance',ax=axis[4])

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```


0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan


0
250
500
750
1000
1250
1500
1750
2000
actual_distance_to_destination


0
1000
2000
3000
4000
actual_time


0
250
500
750
1000
1250
1500
1750
osrm_time



In [32]:
```
def detect_outlier(df, col_name):
    q1 = df[col_name].quantile(0.25)
    q3 = df[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df.loc[(df[col_name] < fence_low) | (df[col_name] > fence_high)]
    return df_out
```

In [33]:
```
outlier_start_scan_to_end_scan = detect_outlier(agg_on_trip_source_dest,'start_scan_to_end_scan')
outlier_start_scan_to_end_scan
```

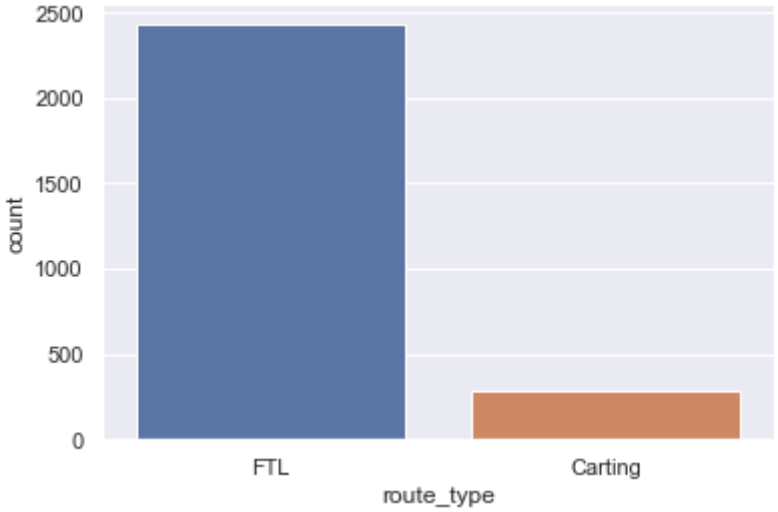
Out[33]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_t
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	12(
	IND462022AAA	IND209304AAA	FTL	2018	09	12	9(
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	8:
	IND562132AAA	IND000000ACB	FTL	2018	09	12	30(
trip-153671186247781647	IND209801AAA	IND209304AAA	Carting	2018	09	12	8'
...	...	...	...	...	...	...	
trip-153860879439383883	IND562132AAA	IND000000ACB	FTL	2018	10	03	31
trip-153860880135634048	IND425405AAA	IND424006AAA	FTL	2018	10	03	98
trip-153860922975807074	IND508207AAB	IND507002AAA	FTL	2018	10	03	6(
trip-153861014185597051	IND462022AAA	IND209304AAA	FTL	2018	10	03	12
trip-153861059679001096	IND208012AAA	IND209304AAA	Carting	2018	10	03	8(

2720 rows × 17 columns

In [34]:
```
sns.countplot(x='route_type',data = outlier_start_scan_to_end_scan)
```

Out[34]:
```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```



In [35]:
```
outlier_actual_distance_to_destination = detect_outlier(agg_on_trip_source_dest,'actual_distance_to_destination')
outlier_actual_distance_to_destination
```

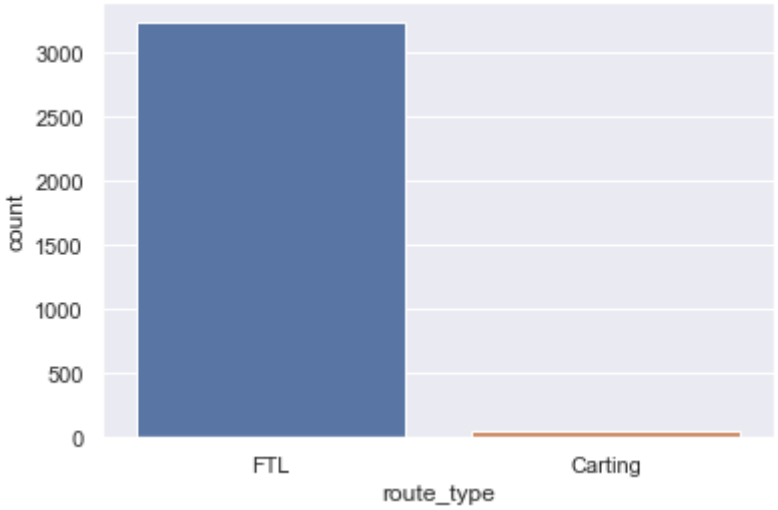
Out[35]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_t
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	12(
	IND462022AAA	IND209304AAA	FTL	2018	09	12	9(
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	8:
	IND562132AAA	IND000000ACB	FTL	2018	09	12	30(
trip-153671121411074590	IND501359AAE	IND515004AAA	FTL	2018	09	12	4(
...	...	...	...	...	...	...	
trip-153860840187622919	IND444005AAB	IND421302AAG	FTL	2018	10	03	10(
trip-153860879439383883	IND000000ACB	IND160002AAC	FTL	2018	10	03	12:
	IND562132AAA	IND000000ACB	FTL	2018	10	03	31
trip-153861014185597051	IND206001AAA	IND000000ACB	FTL	2018	10	03	5(
	IND462022AAA	IND209304AAA	FTL	2018	10	03	12

3292 rows × 17 columns

In [36]:
```
sns.countplot(x='route_type',data = outlier_actual_distance_to_destination)
```

Out[36]:
```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```



In [37]:
```
outlier_actual_time = detect_outlier(agg_on_trip_source_dest,'actual_time')
outlier_actual_time
```

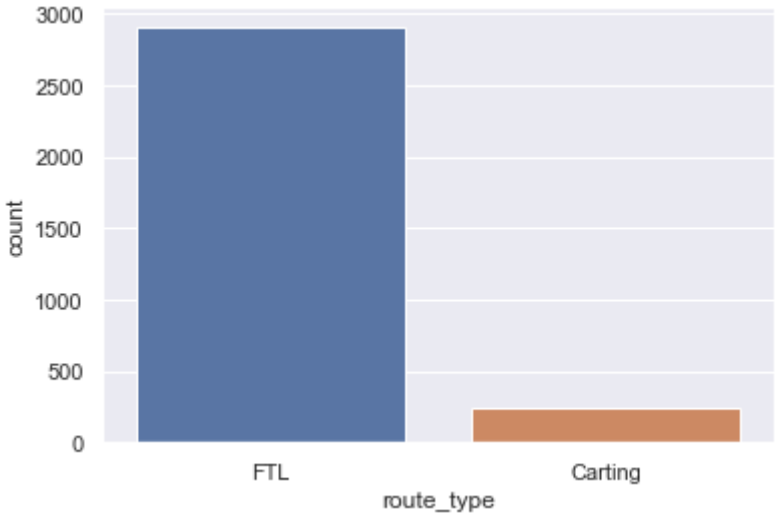
Out[37]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_t
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	126
	IND462022AAA	IND209304AAA	FTL	2018	09	12	99
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	83
	IND562132AAA	IND000000ACB	FTL	2018	09	12	309
trip-153671121411074590	IND501359AAE	IND515004AAA	FTL	2018	09	12	49
...	...	...	...	...	...	...	
trip-153860880135634048	IND425405AAA	IND424006AAA	FTL	2018	10	03	98
trip-153861007249500192	IND847404AAB	IND842001AAA	FTL	2018	10	03	63
trip-153861014185597051	IND206001AAA	IND000000ACB	FTL	2018	10	03	55
	IND462022AAA	IND209304AAA	FTL	2018	10	03	12
trip-153861059679001096	IND208012AAA	IND209304AAA	Carting	2018	10	03	89

3152 rows × 17 columns

In [38]:
```
sns.countplot(x='route_type',data = outlier_actual_time)
```

Out[38]:
```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```



In [39]:
```
outlier_osrm_time = detect_outlier(agg_on_trip_source_dest,'osrm_time')
outlier_osrm_time
```

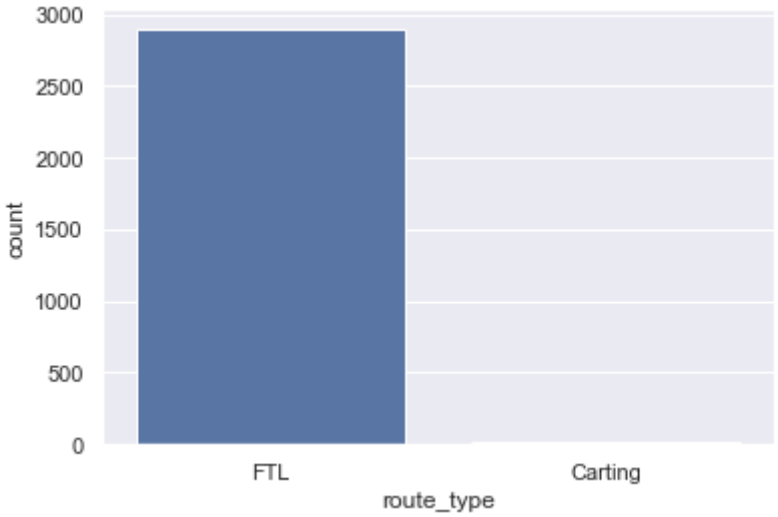
Out[39]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_t
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	12(
	IND462022AAA	IND209304AAA	FTL	2018	09	12	9(
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	8:
	IND562132AAA	IND000000ACB	FTL	2018	09	12	30(
trip-153671121411074590	IND501359AAE	IND515004AAA	FTL	2018	09	12	4(
...	...	...	...	...	...	...	
trip-153860840187622919	IND444005AAB	IND421302AAG	FTL	2018	10	03	10(
trip-153860879439383883	IND000000ACB	IND160002AAC	FTL	2018	10	03	12:
	IND562132AAA	IND000000ACB	FTL	2018	10	03	31
trip-153861014185597051	IND206001AAA	IND000000ACB	FTL	2018	10	03	5(
	IND462022AAA	IND209304AAA	FTL	2018	10	03	12

2919 rows × 17 columns

In [40]:
```
sns.countplot(x='route_type',data = outlier_osrm_time)
```

Out[40]:
```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```



In [41]:
```
outlier_osrm_distance = detect_outlier(agg_on_trip_source_dest,'osrm_distance')
outlier_osrm_distance
```

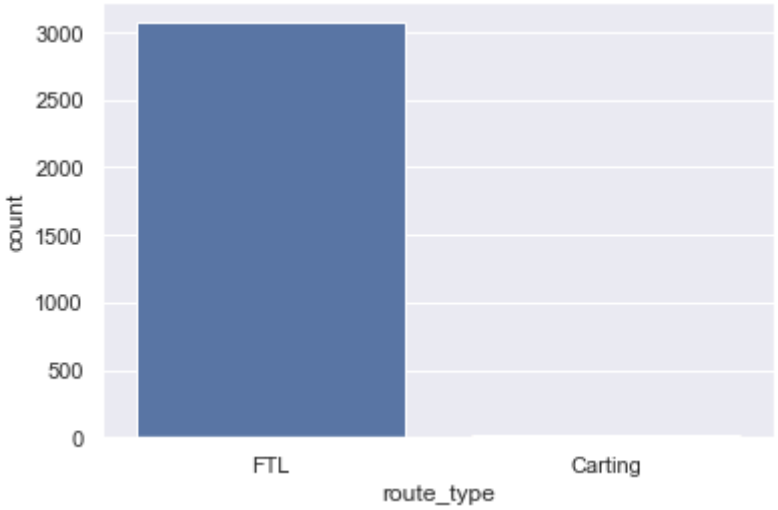
Out[41]:

trip_uuid	source_center	destination_center	route_type	trip_creation_year	trip_creation_month	trip_creation_date	od_total_t
trip-153671041653548748	IND209304AAA	IND000000ACB	FTL	2018	09	12	12(
	IND462022AAA	IND209304AAA	FTL	2018	09	12	9(
trip-153671043369099517	IND000000ACB	IND160002AAC	FTL	2018	09	12	8:
	IND562132AAA	IND000000ACB	FTL	2018	09	12	30(
trip-153671121411074590	IND501359AAE	IND515004AAA	FTL	2018	09	12	4(
...	...	...	...	...	...	...	
trip-153860840187622919	IND444005AAB	IND421302AAG	FTL	2018	10	03	10(
trip-153860879439383883	IND000000ACB	IND160002AAC	FTL	2018	10	03	12:
	IND562132AAA	IND000000ACB	FTL	2018	10	03	31
trip-153861014185597051	IND206001AAA	IND000000ACB	FTL	2018	10	03	5(
	IND462022AAA	IND209304AAA	FTL	2018	10	03	12

3098 rows × 17 columns

In [42]:
```
sns.countplot(x='route_type',data = outlier_osrm_distance)
```

Out[42]:
```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```



Conclusion: Based on above result We can see almost all the outlier data points belong to FTL route type and FTL being long journeys will usually take long time. Since all the outliers are valid values, we are going to consider them in our analysis

In [43]:
```
busiest_city_corridors = agg_on_trip_source_dest.groupby(['source_city',
                                                          'destination_city']).size().reset_index().rename(columns={0:'count'})
busiest_city_corridors.sort_values(by='count',ascending=False).reset_index().head(10)
```

Out[43]:

	index	source_city	destination_city	count
0	272	Bengaluru	Bengaluru	565
1	192	Bangalore	Bengaluru	492
2	340	Bhiwandi	Mumbai	407
3	270	Bengaluru	Bangalore	356
4	958	Hyderabad	Hyderabad	316
5	1530	Mumbai	Mumbai	286
6	1528	Mumbai	Bhiwandi	282
7	579	Delhi	Gurgaon	248
8	481	Chennai	Chennai	246
9	815	Gurgaon	Delhi	237

Conclusion: Based on above table we can see the top busiest routes which includes bangalore, mumbai and hyderabad.

In [44]:

```
busiest_city_corridors_dist_time = agg_on_trip_source_dest.groupby(['source_city',
                                                'destination_city'])['actual_time',
                                                       'actual_distance_to_destination'].agg('mean').reset_index()
pd.merge(busiest_city_corridors_dist_time,
         busiest_city_corridors,left_index=True,right_index=True).drop(['source_city_y',
                                'destination_city_y'],axis=1).sort_values(by='count',
                                       ascending=False).reset_index().rename({'source_city_x':'source_city',
                                                                              'destination_city_x':'destination_city'})
```

Out[44]:

	index	source_city_x	destination_city_x	actual_time	actual_distance_to_destination	count
0	272	Bengaluru	Bengaluru	79.277876	29.075026	565
1	192	Bangalore	Bengaluru	77.674797	27.745791	492
2	340	Bhiwandi	Mumbai	80.120393	22.560310	407
3	270	Bengaluru	Bangalore	91.264045	28.346863	356
4	958	Hyderabad	Hyderabad	102.392405	24.610768	316
...	...	...	...	...	...	...
2365	1735	Pasighat	Gohpur	570.000000	215.406370	1
2366	1737	Patan	Radhanpur	140.000000	53.726334	1
2367	1741	Pathankot	Dalhousie	122.000000	39.798932	1
2368	1747	Patran	Samana	37.000000	26.829084	1
2369	2369	location	location	128.000000	50.844665	1

2370 rows × 6 columns

In [45]:
```
busiest_state_corridors = agg_on_trip_source_dest.groupby(['source_state',
                                                           'destination_state']).size().reset_index().rename(columns={0:'count'})
busiest_state_corridors.sort_values(by='count',ascending=False).reset_index().head(10)
```

Out[45]:

	index	source_state	destination_state	count
0	120	Maharashtra	Maharashtra	3255
1	93	Karnataka	Karnataka	3158
2	153	Tamil Nadu	Tamil Nadu	2021
3	172	Uttar Pradesh	Uttar Pradesh	1526
4	163	Telangana	Telangana	1315
5	182	West Bengal	West Bengal	1296
6	50	Gujarat	Gujarat	1279
7	12	Andhra Pradesh	Andhra Pradesh	1139
8	148	Rajasthan	Rajasthan	1054
9	29	Bihar	Bihar	1011

In [46]:

```
busiest_state_corridors_dist_time = agg_on_trip_source_dest.groupby(['source_state',
                                                'destination_state'])['actual_time',
                                                'actual_distance_to_destination'].agg('mean').reset_index()

pd.merge(busiest_state_corridors,busiest_state_corridors_dist_time,
         left_index=True,right_index=True).drop(['source_state_y','destination_state_y'],
                                                axis=1).sort_values(by='count',
                                                                    ascending=False).reset_index().rename(
    {'source_city_x':'source_city','destination_city_x':'destination_city'}).head(10)
```

Out[46]:

	index	source_state_x	destination_state_x	count	actual_time	actual_distance_to_destination
0	120	Maharashtra	Maharashtra	3255	129.477419	48.333074
1	93	Karnataka	Karnataka	3158	92.926852	38.058320
2	153	Tamil Nadu	Tamil Nadu	2021	75.818407	34.846109
3	172	Uttar Pradesh	Uttar Pradesh	1526	136.305374	48.382655
4	163	Telangana	Telangana	1315	97.673004	42.062285
5	182	West Bengal	West Bengal	1296	136.176698	37.334884
6	50	Gujarat	Gujarat	1279	99.157936	48.589073
7	12	Andhra Pradesh	Andhra Pradesh	1139	97.304653	46.657201
8	148	Rajasthan	Rajasthan	1054	139.261860	64.128019
9	29	Bihar	Bihar	1011	163.715134	49.206346

Conclusion: Based on above table we can see the top busiest state routes which includes Maharashtra, Karnataka and Tamil Nadu

Handling Categorical values

In [47]:

```
agg_on_trip_source_dest_encoder = pd.get_dummies(agg_on_trip_source_dest.route_type, prefix='route')
agg_on_trip_source_dest = agg_on_trip_source_dest.join(agg_on_trip_source_dest_encoder)
agg_on_trip_source_dest.rename({'route_Carting':'carting'},axis=1,inplace=True)
agg_on_trip_source_dest.drop('route_FTL',axis=1,inplace=True)
agg_on_trip_source_dest.drop('route_type',axis=1,inplace=True)
agg_on_trip_source_dest
```

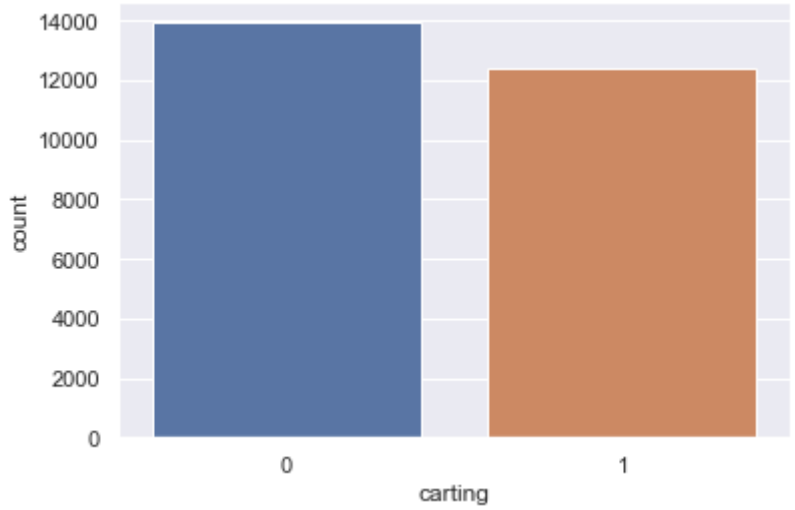
Out[47]:

trip_uuid	source_center	destination_center	trip_creation_year	trip_creation_month	trip_creation_date	od_total_time	start_sc
trip-153671041653548748	IND209304AAA	IND000000ACB	2018	09	12	1260.0	
	IND462022AAA	IND209304AAA	2018	09	12	999.0	
trip-153671042288605164	IND561203AAB	IND562101AAA	2018	09	12	58.0	
	IND572101AAA	IND561203AAB	2018	09	12	122.0	
trip-153671043369099517	IND000000ACB	IND160002AAC	2018	09	12	834.0	
...	...	...	...	...	...	...	
trip-153861115439069069	IND628204AAA	IND627657AAA	2018	10	03	62.0	
	IND628613AAA	IND627005AAA	2018	10	03	91.0	
	IND628801AAA	IND628204AAA	2018	10	03	44.0	
trip-153861118270144424	IND583119AAA	IND583101AAA	2018	10	03	287.0	
	IND583201AAA	IND583119AAA	2018	10	03	66.0	

26368 rows × 17 columns

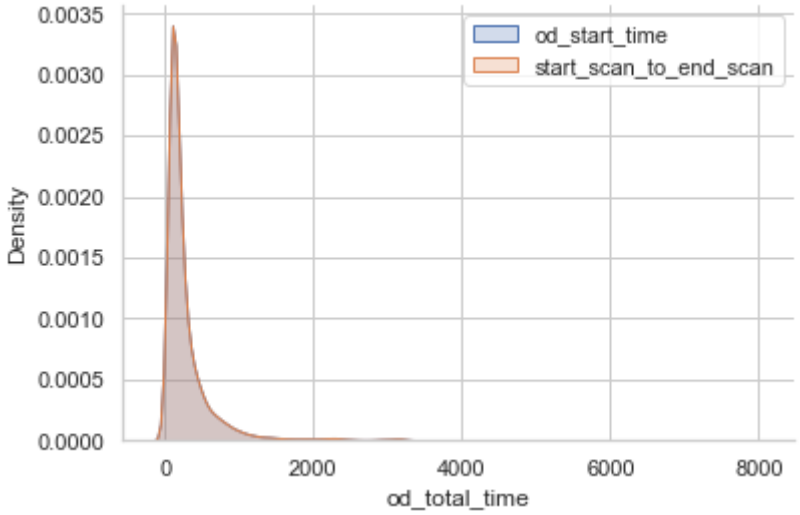
In [48]:
```
sns.countplot(data=agg_on_trip_source_dest,x='carting')
```

Out[48]:
```
<AxesSubplot:xlabel='carting', ylabel='count'>
```



2 sample T test to check if od_total_time and start_scan_to_end_scan are same or not

In [49]:
```
sns.set_style('whitegrid')
sns.kdeplot(data=agg_on_trip_source_dest, x='od_total_time',fill=True,label='od_start_time')
sns.kdeplot(data=agg_on_trip_source_dest,x='start_scan_to_end_scan',fill=True,label='start_scan_to_end_scan')
sns.despine()
plt.legend()
plt.show()
```



Observation: From the above plot it is clear that the distribution is not normal for both the groups. Also we can see that od_start_time and start_scan_to_end_scan completely overlap each other.

Assumptions

- Both groups are independent
- Both groups are obtained through random sampling
- Data in each group is normally distributed
- variance of both the groups should be similar
- By visual analysis we can see that the data is not normally distributed for both groups(i.e od_total_time and start_scan_to_end_scan)

Hypothesis Formulation and test selection

- We'll have the following hypothesis
  - Null Hypothesis: The means of od_start_time and start_scan_to_end_scan are similar.
  - Alternate Hypothesis: The means of od_start_time and start_scan_to_end_scan are different.
  - We'll consider the significance value as 5% and perform a two tailed test
- Test Selection

- We'll use 2 sample T-test since we need to compare mean of two independendent group. The 2 sample t-test behaves similar to 2 sample z-test for large dataset(i.e. n>30)

Checking Test assumptions

- We know that both groups are independent of each other since each both the times are calculated independently.
- We assume that both the groups are obtained from random sampling
- Data in each group is normally distributed - This assumption breaks as we have seen in the above plot. We need to apply a log transform to convert it to gaussian.
- Variance of both the groups must be similar. Below we'll see that the variance is somewhat similar for both groups

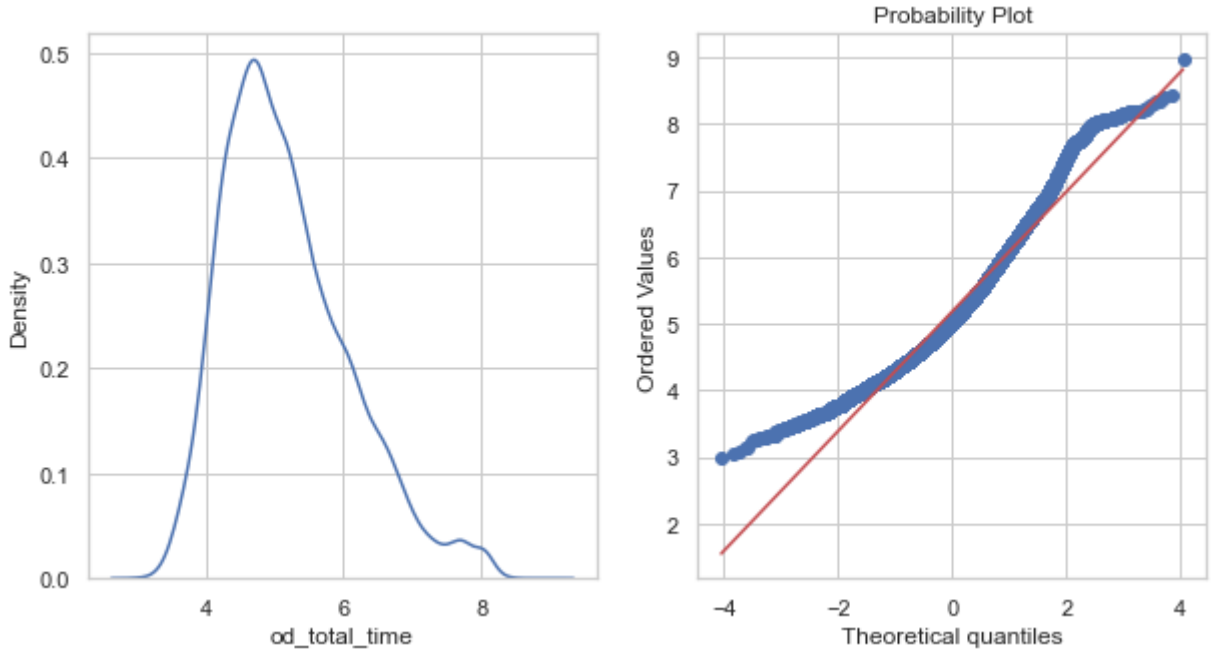
In [51]:
```
od_total_time_var = np.var(agg_on_trip_source_dest['od_total_time'])
start_scan_to_end_scan_variance = np.var(agg_on_trip_source_dest['start_scan_to_end_scan'])
print("variance of od_total_time is {} and variance of start_scan_to_end_scan is {}".format(od_total_time_var,
                                                                                            start_scan_to_end_scan_variance))
```

```
variance of od_total_time is 194083.0413063139 and variance of start_scan_to_end_scan is 194083.0413063139
```

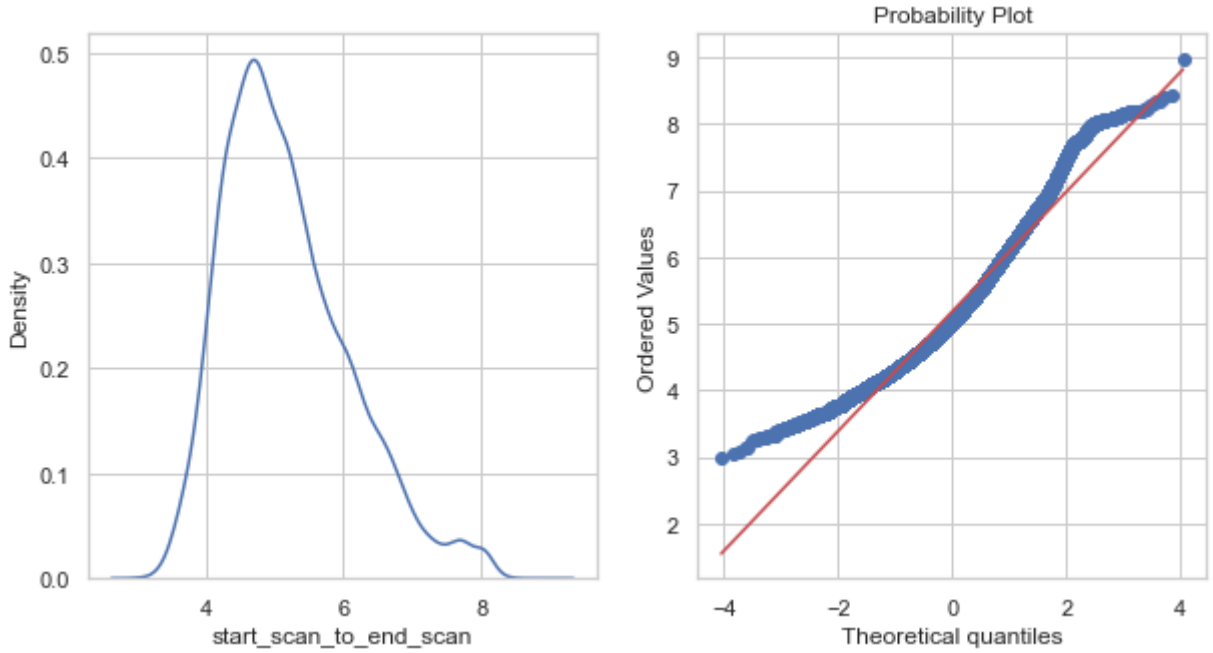
Applying Log normal transformation to convert the data to gaussian

In [52]:
```
def normality(data):
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    sns.kdeplot(data)
    plt.subplot(1,2,2)
    stats.probplot(data,plot=pylab)
    plt.show()
```

In [53]:
```
od_total_time_transformed = np.log(agg_on_trip_source_dest['od_total_time'])
start_scan_to_end_scan_transformed = np.log(agg_on_trip_source_dest['start_scan_to_end_scan'])
normality(od_total_time_transformed)
```



In [54]:
```
normality(start_scan_to_end_scan_transformed)
```



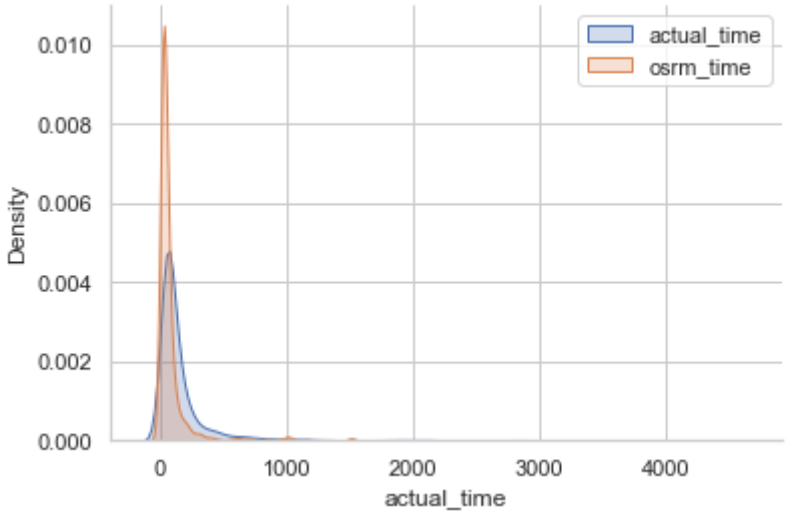
In [55]:
```
stats.ttest_ind(od_total_time_transformed,start_scan_to_end_scan_transformed)
```

Out[55]:
```
Ttest_indResult(statistic=0.0, pvalue=1.0)
```

Conclusion: Based on above result p-value = 1 which is greater than our significance value alpha. So based on this We can say that we fail to reject the Null Hypothesis. It means that there is od_total_time and start_scan_to_end_scan are same.

2 sample T test to check if actual_time and osrm_time are same or not

In [56]:
```
sns.set_style('whitegrid')
sns.kdeplot(data=agg_on_trip_source_dest, x='actual_time',fill=True,label='actual_time')
sns.kdeplot(data=agg_on_trip_source_dest,x='osrm_time',fill=True,label='osrm_time')
sns.despine()
plt.legend()
plt.show()
```



Observation: From the above plot it is clear that the distribution is not normal for both the groups. Also we can see that actual_time and osrm_time don't completely overlap each other.

Assumptions

- Both groups are independent
- Both groups are obtained through random sampling
- Data in each group is normally distributed
- variance of both the groups should be similar
- By visual analysis we can see that the data is not normally distributed for both groups(i.e actual_time and osrm_time)

Hypothesis Formulation and test selection

- We'll have the following hypothesis
  - Null Hypothesis: The means of actual_time and osrm_time are similar.
  - Alternate Hypothesis: The means of actual_time and osrm_time are different.
  - We'll consider the significance value as 5% and perform a two tailed test
- Test Selection
  - We'll use 2 sample T-test since we need to compare mean of two independendent group. The 2 sample t-test behaves similar to 2 sample z-test for large dataset(i.e. n>30)

Checking Test assumptions

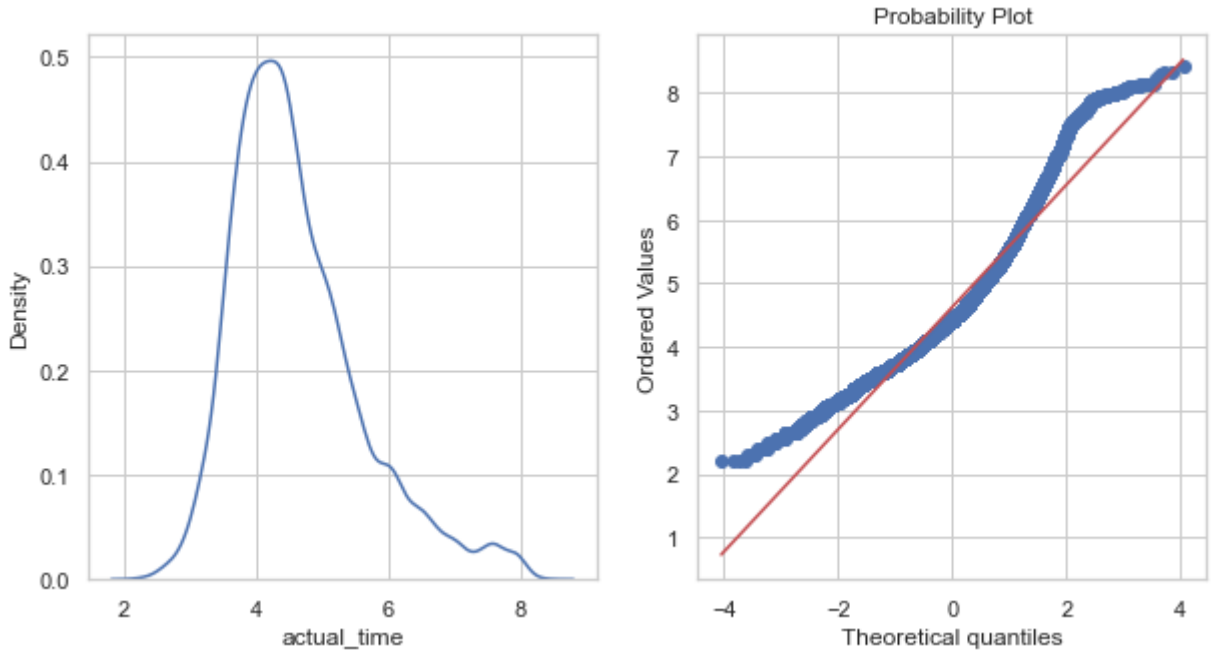
- We know that both groups are independent of each other since each both the times are calculated independently.
- We assume that both the groups are obtained from random sampling
- Data in each group is normally distributed - This assumption breaks as we have seen in the above plot. We need to apply a log transform to convert it to gaussian.
- Variance of both the groups must be similar. Below we'll see that the variance is somewhat similar for both groups

In [57]:
```
actual_time_var = np.var(agg_on_trip_source_dest['actual_time'])
osrm_time_variance = np.var(agg_on_trip_source_dest['osrm_time'])
print("variance of actual_time_var is {} and variance of osrm_time_variance is {}".format(actual_time_var,
                                                                                          osrm_time_variance))
```

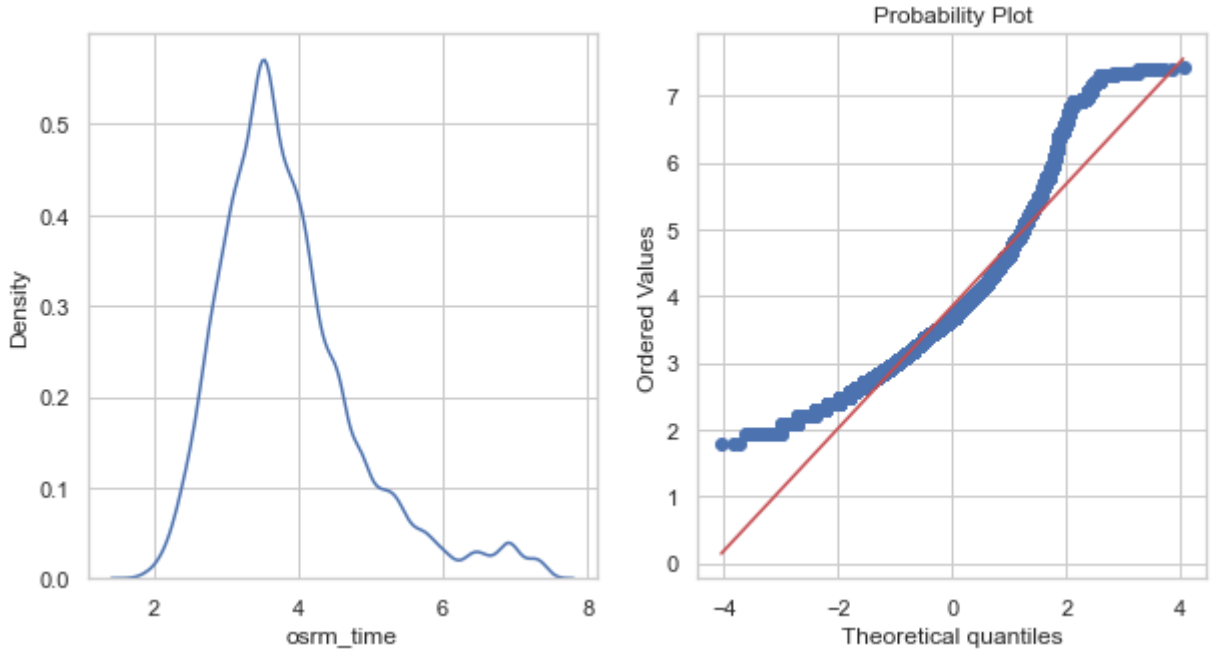
```
variance of actual_time_var is 148106.70707614435 and variance of osrm_time_variance is 34253.464004030066
```

Applying Log normal transformation to convert the data to gaussian

In [58]:
```
actual_time_transformed = np.log(agg_on_trip_source_dest['actual_time'])
osrm_time_transformed = np.log(agg_on_trip_source_dest['osrm_time'])
normality(actual_time_transformed)
```



In [59]:
```
normality(osrm_time_transformed)
```



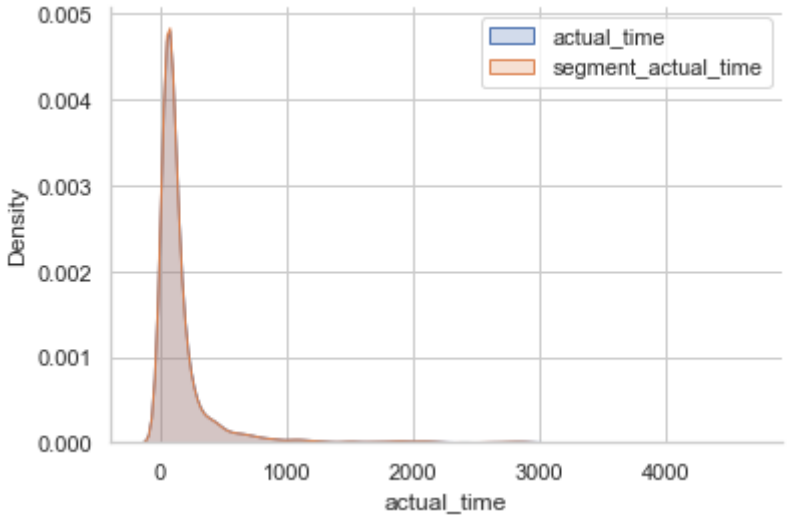
In [60]:
```
stats.ttest_ind(actual_time_transformed,osrm_time_transformed)
```

Out[60]:
```
Ttest_indResult(statistic=91.75651556695851, pvalue=0.0)
```

Conclusion: Based on above result p-value = 0 which is less than our significance value alpha. So based on this We can say that we reject the Null Hypothesis. It means that there is actual_time and osrm_time are different.

2 sample T test to check if actual_time and segment_actual_time are same or not

In [61]:
```
sns.set_style('whitegrid')
sns.kdeplot(data=agg_on_trip_source_dest, x='actual_time',fill=True,label='actual_time')
sns.kdeplot(data=agg_on_trip_source_dest,x='segment_actual_time_cum',fill=True,label='segment_actual_time')
plt.legend()
sns.despine()
plt.show()
```



Observation: From the above plot it is clear that the distribution is not normal for both the groups. Also we can see that actual_time and segment_actual_time don't completely overlap each other.

Assumptions

- Both groups are independent
- Both groups are obtained through random sampling
- Data in each group is normally distributed
- variance of both the groups should be similar
- By visual analysis we can see that the data is not normally distributed for both groups(i.e actual_time and segment_actual_time)

Hypothesis Formulation and test selection

- We'll have the following hypothesis
  - Null Hypothesis: The means of actual_time and segment_actual_time are similar.
  - Alternate Hypothesis: The means of actual_time and segment_actual_time are different.
  - We'll consider the significance value as 5% and perform a two tailed test
- Test Selection

- We'll use 2 sample T-test since we need to compare mean of two independendent group. The 2 sample t-test behaves similar to 2 sample z-test for large dataset(i.e. n>30)

Checking Test assumptions

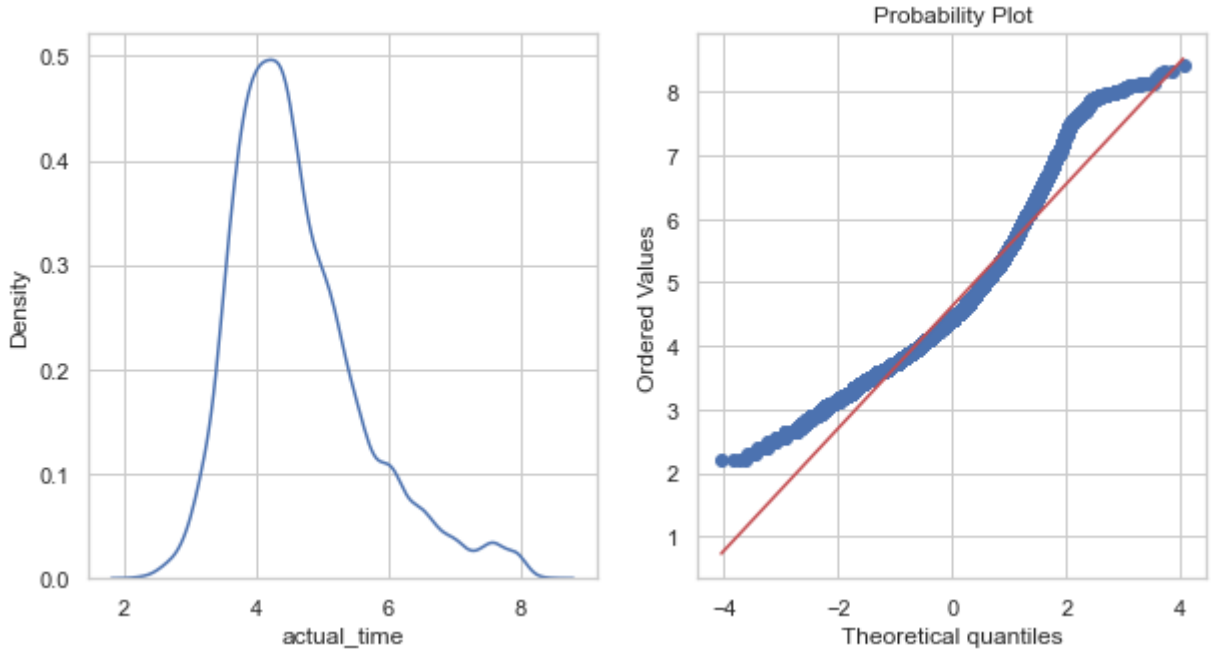
- We know that both groups are independent of each other since each both the times are calculated independently.
- We assume that both the groups are obtained from random sampling
- Data in each group is normally distributed - This assumption breaks as we have seen in the above plot. We need to apply a log transform to convert it to gaussian.
- Variance of both the groups must be similar. Below we'll see that the variance is somewhat similar for both groups

In [62]:
```
actual_time_var = np.var(agg_on_trip_source_dest['actual_time'])
segment_actual_time_variance = np.var(agg_on_trip_source_dest['segment_actual_time_cum'])
print("variance of actual_time_var is {} and variance of segment_actual_time_variance is {}".format(actual_time_var,
                                                                                                    segment_actual_time_variance))
```

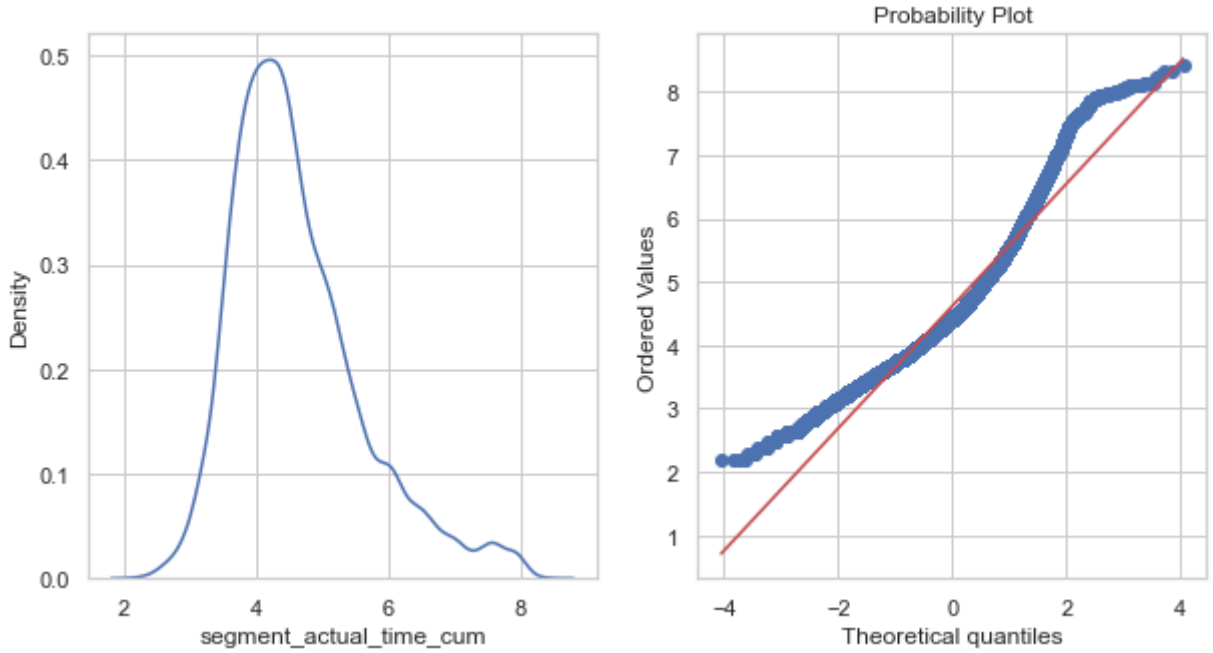
```
variance of actual_time_var is 148106.70707614435 and variance of segment_actual_time_variance is 145371.38348606933
```

Applying Log normal transformation to convert the data to gaussian

In [63]:
```
actual_time_transformed = np.log(agg_on_trip_source_dest['actual_time'])
segment_actual_time_transformed = np.log(agg_on_trip_source_dest['segment_actual_time_cum'])
normality(actual_time_transformed)
```



In [64]:
```
normality(segment_actual_time_transformed)
```



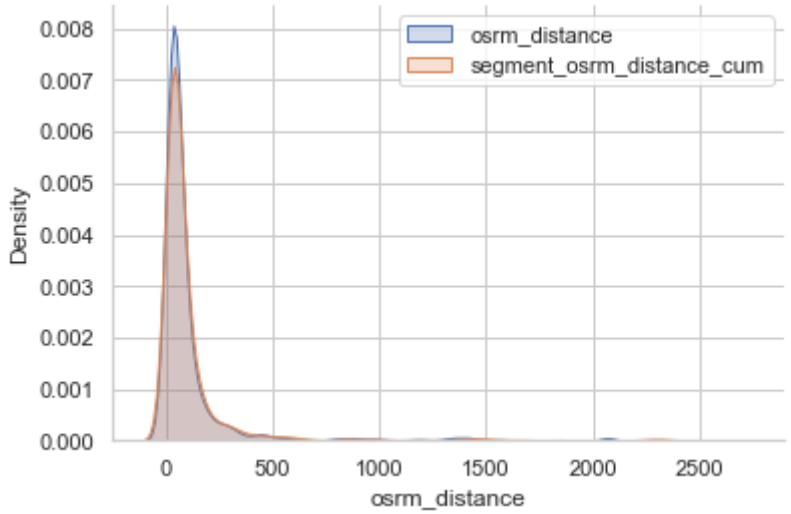
In [65]:
```
stats.ttest_ind(actual_time_transformed,segment_actual_time_transformed)
```

Out[65]:
```
Ttest_indResult(statistic=1.1719659905322306, pvalue=0.24121600541669044)
```

Conclusion: Based on above result p-value = 0.24 which is greater than our significance value alpha. So based on this We can say that we fail to reject the Null Hypothesis. It means that there is actual_time and segment_actual_time are same.

2 sample T test to check if actual_time and segment_actual_time are same or not

In [66]:
```
sns.set_style('whitegrid')
sns.kdeplot(data=agg_on_trip_source_dest, x='osrm_distance',fill=True,label='osrm_distance')
sns.kdeplot(data=agg_on_trip_source_dest,x='segment_osrm_distance_cum',fill=True,label='segment_osrm_distance_cum')
plt.legend()
sns.despine()
plt.show()
```



Observation: From the above plot it is clear that the distribution is not normal for both the groups. Also we can see that osrm_distance and segment_osrm_distance_cum don't completely overlap each other.

Assumptions

- Both groups are independent
- Both groups are obtained through random sampling
- Data in each group is normally distributed
- variance of both the groups should be similar
- By visual analysis we can see that the data is not normally distributed for both groups(i.e osrm_distance and segment_osrm_distance_cum)

Hypothesis Formulation and test selection

- We'll have the following hypothesis
  - Null Hypothesis: The means of actual_time and segment_actual_time are similar.
  - Alternate Hypothesis: The means of osrm_distance and segment_osrm_distance_cum are different.
  - We'll consider the significance value as 5% and perform a two tailed test
- Test Selection
  - We'll use 2 sample T-test since we need to compare mean of two independendent group. The 2 sample t-test behaves similar to 2 sample z-test for large dataset(i.e. n>30)

Checking Test assumptions

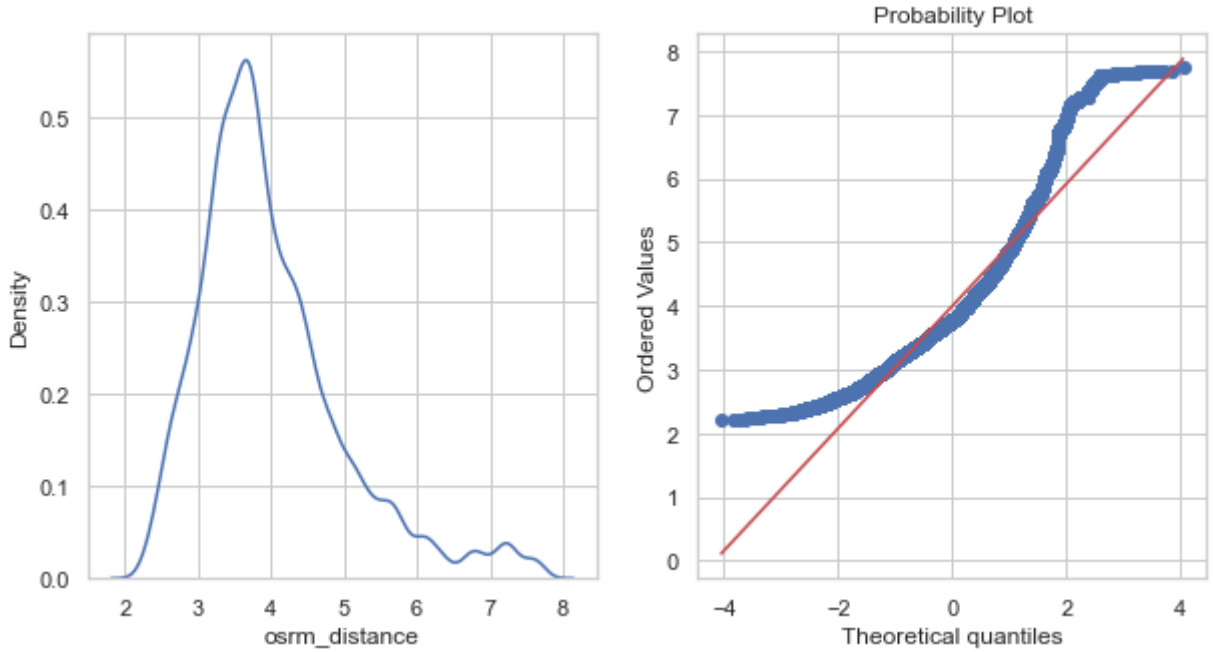
- We know that both groups are independent of each other since each both the times are calculated independently.
- We assume that both the groups are obtained from random sampling
- Data in each group is normally distributed - This assumption breaks as we have seen in the above plot. We need to apply a log transform to convert it to gaussian.
- Variance of both the groups must be similar. Below we'll see that the variance is somewhat similar for both groups

In [67]:
```
osrm_distance_var = np.var(agg_on_trip_source_dest['osrm_distance'])
segment_osrm_distance_cum_var = np.var(agg_on_trip_source_dest['segment_osrm_distance_cum'])
print("variance of osrm_distance_var is {} and variance of segment_osrm_distance_cum_var is {}".format(
    osrm_distance_var,segment_osrm_distance_cum_var))
```

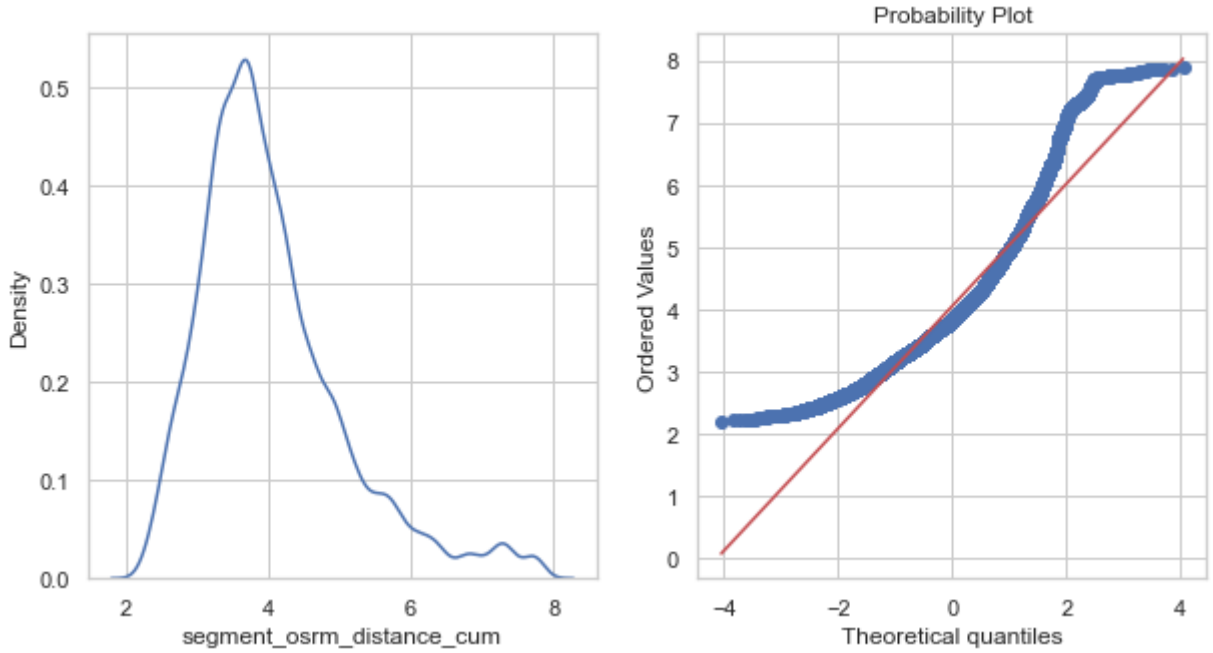
```
variance of osrm_distance_var is 64398.681296933704 and variance of segment_osrm_distance_cum_var is 81754.32592610987
```

Applying Log normal transformation to convert the data to gaussian

In [68]:
```
osrm_distance_transformed = np.log(agg_on_trip_source_dest['osrm_distance'])
segment_osrm_distance_cum_transformed = np.log(agg_on_trip_source_dest['segment_osrm_distance_cum'])
normality(osrm_distance_transformed)
```



In [69]:
```
normality(segment_osrm_distance_cum_transformed)
```



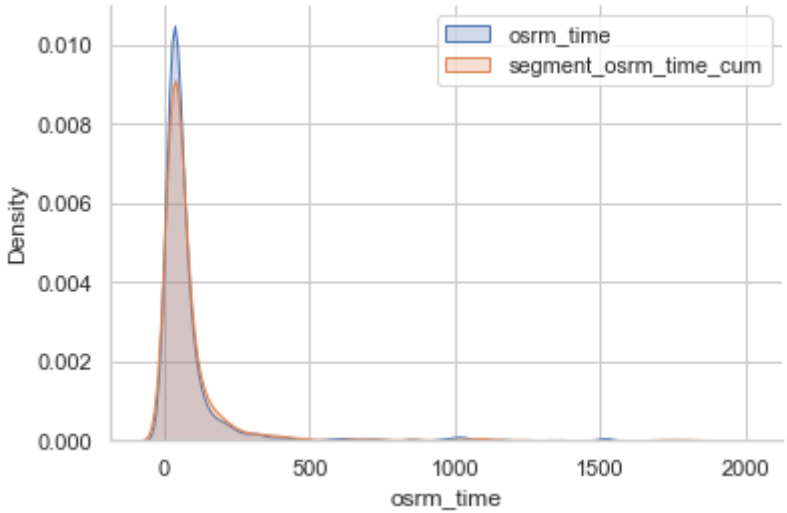
In [70]:
```
stats.ttest_ind(osrm_distance_transformed,segment_osrm_distance_cum_transformed)
```

Out[70]:
```
Ttest_indResult(statistic=-6.016679309589272, pvalue=1.792010384210536e-09)
```

Conclusion: Based on above result p-value = 1.79^(-9) which is less than our significance value alpha. So based on this We can say that we reject the Null Hypothesis. It means that there is osrm_distance and segment_osrm_distance_cum are different.

2 sample T test to check if osrm_time_cum and segment_osrm_time_cum are same or not

In [71]:
```
sns.set_style('whitegrid')
sns.kdeplot(data=agg_on_trip_source_dest, x='osrm_time',fill=True,label='osrm_time')
sns.kdeplot(data=agg_on_trip_source_dest,x='segment_osrm_time_cum',fill=True,label='segment_osrm_time_cum')
plt.legend()
sns.despine()
plt.show()
```



Observation: From the above plot it is clear that the distribution is not normal for both the groups. Also we can see that osrm_time and segment_osrm_time_cum don't completely overlap each other.

Assumptions

- Both groups are independent
- Both groups are obtained through random sampling
- Data in each group is normally distributed
- variance of both the groups should be similar
- By visual analysis we can see that the data is not normally distributed for both groups(i.e osrm_time and segment_osrm_time_cum)

Hypothesis Formulation and test selection

- We'll have the following hypothesis
  - Null Hypothesis: The means of actual_time and segment_osrm_time_cum are similar.
  - Alternate Hypothesis: The means of osrm_time and segment_osrm_distance_cum are different.
  - We'll consider the significance value as 5% and perform a two tailed test
- Test Selection

- We'll use 2 sample T-test since we need to compare mean of two independendent group. The 2 sample t-test behaves similar to 2 sample z-test for large dataset(i.e. n>30)

Checking Test assumptions

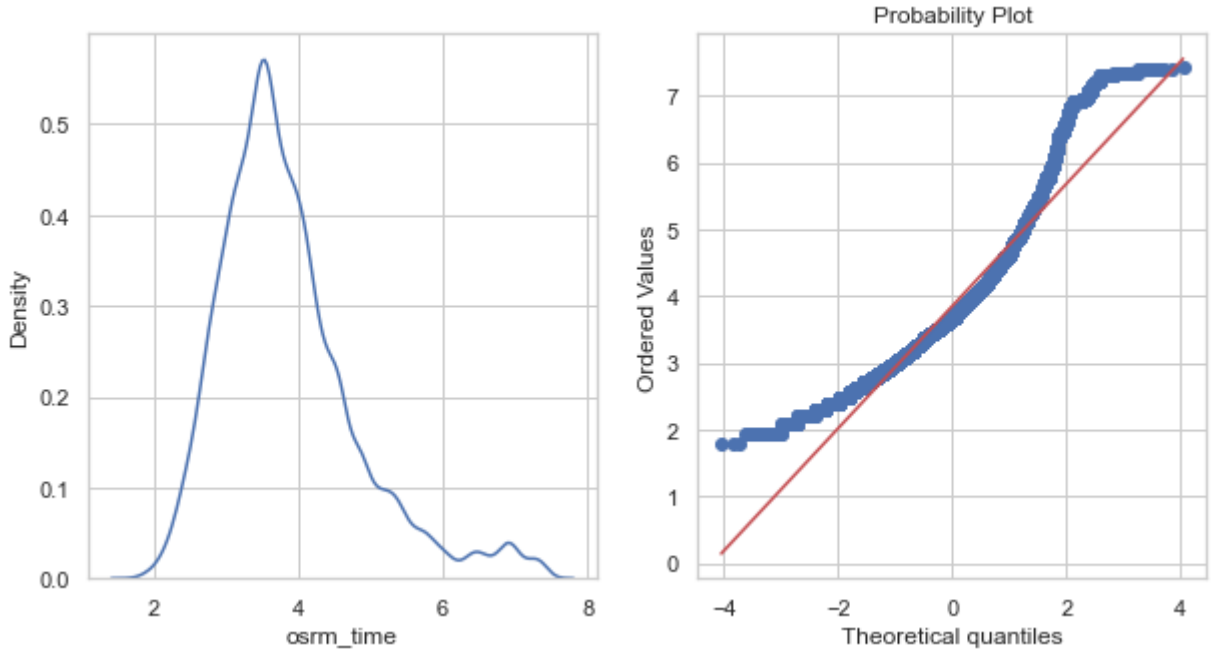
- We know that both groups are independent of each other since each both the times are calculated independently.
- We assume that both the groups are obtained from random sampling
- Data in each group is normally distributed - This assumption breaks as we have seen in the above plot. We need to apply a log transform to convert it to gaussian.
- Variance of both the groups must be similar. Below we'll see that the variance is somewhat similar for both groups

In [72]:
```
osrm_time_var = np.var(agg_on_trip_source_dest['osrm_time'])
segment_osrm_time_var = np.var(agg_on_trip_source_dest['segment_osrm_time_cum'])
print("variance of osrm_time_var is {} and variance of segment_osrm_time_cum_var is {}".format(osrm_time_var,
                                                                                               segment_osrm_time_var))
```

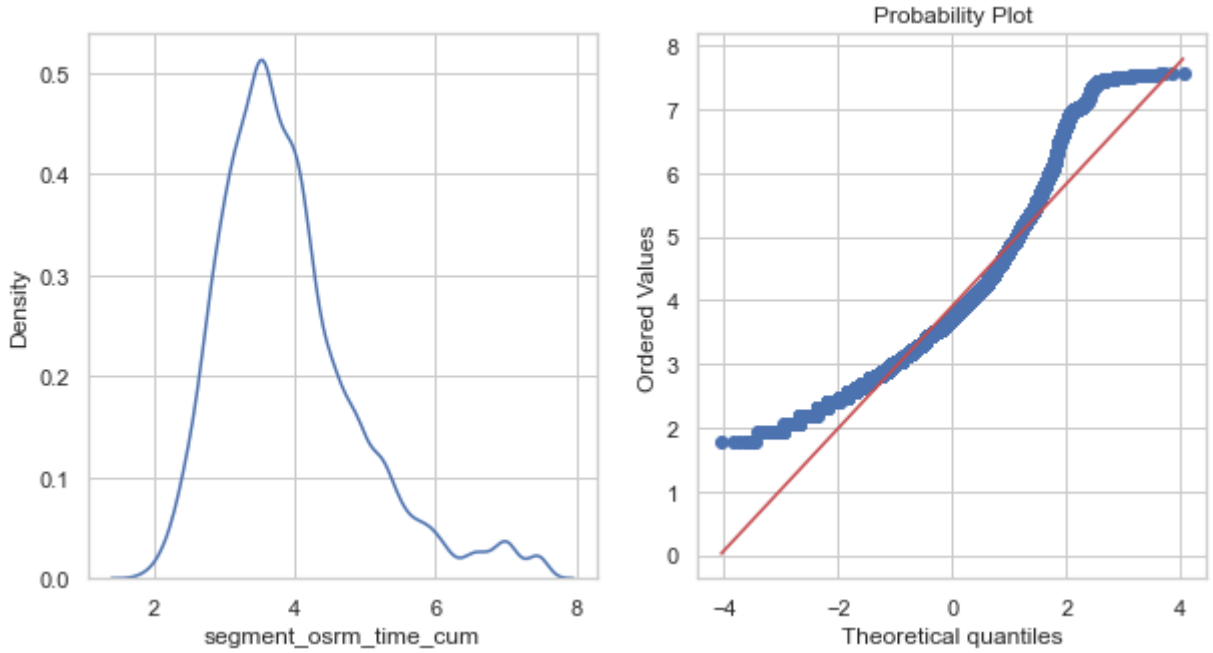
```
variance of osrm_time_var is 34253.464004030066 and variance of segment_osrm_time_cum_var is 46503.5677759929
```

Applying Log normal transformation to convert the data to gaussian

In [73]:
```
osrm_time_transformed = np.log(agg_on_trip_source_dest['osrm_time'])
segment_osrm_time_cum_transformed = np.log(agg_on_trip_source_dest['segment_osrm_time_cum'])
normality(osrm_time_transformed)
```



In [74]:
```
normality(segment_osrm_time_cum_transformed)
```



In [75]:
```
stats.ttest_ind(osrm_time_transformed,segment_osrm_time_cum_transformed)
```

Out[75]:
```
Ttest_indResult(statistic=-7.264652594570266, pvalue=3.7915395546872833e-13)
```

Conclusion: Based on above result p-value = 3.3^(-13) which is less than our significance value alpha. So based on this We can say that we reject the Null Hypothesis. It means that there is osrm_time and segment_osrm_time_cum are different.

Normalization

In [76]:
```
normalized_agg_on_trip_source_dest = agg_on_trip_source_dest.copy()
columns_to_normalize = ['od_total_time','start_scan_to_end_scan','actual_distance_to_destination',
                        'actual_time','osrm_time','osrm_distance','segment_actual_time_cum',
                        'segment_osrm_time_cum','segment_osrm_distance_cum']
normalized_agg_on_trip_source_dest[columns_to_normalize] = preprocessing.normalize(
                                                            agg_on_trip_source_dest[columns_to_normalize])
normalized_agg_on_trip_source_dest
```

Out[76]:

trip_uuid	source_center	destination_center	trip_creation_year	trip_creation_month	trip_creation_date	od_total_time	start_sc
trip-153671041653548748	IND209304AAA	IND000000ACB	2018	09	12	0.540677	
	IND462022AAA	IND209304AAA	2018	09	12	0.463470	
trip-153671042288605164	IND561203AAB	IND562101AAA	2018	09	12	0.479966	
	IND572101AAA	IND561203AAB	2018	09	12	0.497962	
trip-153671043369099517	IND000000ACB	IND160002AAC	2018	09	12	0.530788	
...	...	...	...	...	...	...	
trip-153861115439069069	IND628204AAA	IND627657AAA	2018	10	03	0.429378	
	IND628613AAA	IND627005AAA	2018	10	03	0.407884	
	IND628801AAA	IND628204AAA	2018	10	03	0.538410	
trip-153861118270144424	IND583119AAA	IND583101AAA	2018	10	03	0.538539	
	IND583201AAA	IND583119AAA	2018	10	03	0.527015	

26368 rows × 17 columns

Recommendations

Based on above Analysis, following recommendations can be made:

* Most of the trips are taking place in tier 1 cities like bangalore, mumbai and hyderabad and are mostly inter city. There are not many interstate delhiveries taking place. Team can focus on inter state delhiveries since it can generate more revenue due to long distance parcels.
* The average delivery time in west bengal is more in comparison to other states. Team can focus on reducing this delivery time.
* osrm_time and segment_osrm_time are different. Team needs to make sure this difference is reduced, so that better delivery time prediction can be made and it becomes convenient for the customer to expect an accurate delivery time.
* The actual time taken to deliver and the one predicted by osrm are not same. Team needs to improve this prediction modes so as to give more accurate predictions.

* The osrm distance and actual distance covered are also not same i.e. maybe the delivery person is not following the predefined route which may lead to late deliveries or the osrm devices is not properly predicting the route based on distance, traffic and other factors. Team needs to look into it.

In [ ]: