Problem Statement:

The company wants to understand and process the data coming out of data engineering pipelines.The data is not cleaned & haphazard with so many raw columns:

- We are expected to Clean, sanitize and manipulate data to get useful features out of raw fields
- Make sense out of the raw data and help the data science team to build forecasting models on it

Data Dictionary

Column Profiling:

- data - tells whether the data is testing or training data
- trip_creation_time – Timestamp of trip creation
- route_schedule_uuid – Unique Id for a particular route schedule
- route_type – Transportation type
- FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
- Carting: Handling system consisting of small vehicles (carts)
- trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
- source_center - Source ID of trip origin
- source_name - Source Name of trip origin
- destination_cente – Destination ID
- destination_name – Destination Name
- od_start_time – Trip start time
- od_end_time – Trip end time
- start_scan_to_end_scan – Time taken to deliver from source to destination
- is_cutoff – Unknown field
- cutoff_factor – Unknown field
- cutoff_timestamp – Unknown field
- actual_distance_to_destination – Distance in Kms between source and destination warehouse
- actual_time – Actual time taken to complete the delivery (Cumulative)
- osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
- osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
- factor – Unknown field
- segment_actual_time – This is a segment time. Time taken by the subset of the package delivery

- segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
- segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
- segment_factor – Unknown field

Loading The Dataset

In [77]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import NaN, nan, NAN
from scipy import stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001,
```

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

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

Removing the columns that are taged as unknown field in data dictionary as we dont have any information about these columns and hence cant be leveraged in analysis

In [4]:
```
df.drop(columns=['segment_factor','data','factor','is_cutoff', 'cutoff_factor','cu
                'route_schedule_uuid'],
       axis=1, inplace=True)
```

In [5]:
```
print('Rows:', df.shape[0],'\n' 'Columns: ',df.shape[1])
```

```
Rows: 144867
Columns:  17
```

In [6]:
```
df.head(1)
```

Out[6]:

	trip_creation_time	route_type	trip_uuid	source_center	source_name	destina
0	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND

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

Out[7]:
```
Index(['trip_creation_time', 'route_type', 'trip_uuid', 'source_center',
       'source_name', 'destination_center', 'destination_name',
       'od_start_time', 'od_end_time', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 17 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   trip_creation_time              144867 non-null  object 
 1   route_type                      144867 non-null  object 
 2   trip_uuid                       144867 non-null  object 
 3   source_center                   144867 non-null  object 
 4   source_name                     144574 non-null  object 
 5   destination_center              144867 non-null  object 
 6   destination_name                144606 non-null  object 
 7   od_start_time                   144867 non-null  object 
 8   od_end_time                     144867 non-null  object 
 9   start_scan_to_end_scan          144867 non-null  float64
 10  actual_distance_to_destination  144867 non-null  float64
 11  actual_time                     144867 non-null  float64
 12  osrm_time                       144867 non-null  float64
 13  osrm_distance                   144867 non-null  float64
 14  segment_actual_time             144867 non-null  float64
 15  segment_osrm_time               144867 non-null  float64
 16  segment_osrm_distance           144867 non-null  float64
dtypes: float64(8), object(9)
memory usage: 18.8+ MB
```

Missing Values & their treatment

In [9]:

```
# Function to create a data frame with number and percentage of missing data in a d

def missing_to_df(df):
    #Number and percentage of missing data in training data set for each column
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(as
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, key
    return missing_data_df
```

In [10]:

```
missing_df = missing_to_df(df)
missing_df[missing_df['Total'] > 0]
```

Out[10]:

	Total	Percent
source_name	293	0.202254
destination_name	261	0.180165

- Only two fields have missing values which consists of only 0.02% of the data
- As we have enough observations to proceed with the Analysis we will be dropping the missing values for this dataset. Though we can treat the missing values by mean/median/mode imputation
- My reason for dropping the missing values is that I dont intent to disturb the distribution of features via imputation.

Alternatively if missing values comprise of large part of data they can be treated using responce coding, mode, mean median imputation, probabilistic based imputation & other forms of imputations decided by business team/ managers.

In [11]:
```
#dropping the missing values
df.dropna(inplace=True)
```

In [12]:
```
d1=df.copy()
```

Feature Engineering

In [13]:
```
d_grouped=d1.groupby(['trip_uuid','source_center','destination_center']).count().re
```

Lets pick one trip_uuid to understand its journey in raw data and we will create features accrodingly

In [14]:
```
d1[d1['trip_uuid']=='trip-153741093647649320']
```

Out[14]:

	trip_creation_time	route_type	trip_uuid	source_center	source_name	des
0	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
1	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
2	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
3	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
4	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
5	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
6	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
7	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
8	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
9	2018-09-20 02:35:36.476840	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	

In [15]:
```
#changing datatype of date like columns from object to timestamp
d1[["od_start_time", "od_end_time",'trip_creation_time']] = d1[["od_start_time", "o
```

In [16]:
```
#extracting day, month & year from trip_creation_time
d1['trip_creation_month']=d1['trip_creation_time'].dt.month
d1['trip_creation_year']=d1['trip_creation_time'].dt.year
d1['trip_creation_day']=d1['trip_creation_time'].dt.day
```

In [17]:
```
#difference between od_start & od_end time in hours
d1['Timediff_start_end_H']=round((d1['od_end_time']-d1['od_start_time'])/pd.Timede
```

In [18]:
```
#as below mentioned columns are comprising of segment related details we will do a
d1['agg_segment_actual_time']=d1.groupby(['trip_uuid','source_center','destination_
d1['agg_segment_osrm_time']=d1.groupby(['trip_uuid','source_center','destination_ce
d1['agg_segment_osrm_distance']=d1.groupby(['trip_uuid','source_center','destinatio
```

In [19]:
```
#After finding out the cumsum of above columns we will pick their max
d1['agg_segment_actual_time1']=d1.groupby(['trip_uuid','source_center','destinatio
d1['agg_segment_osrm_time1']=d1.groupby(['trip_uuid','source_center','destination_
d1['agg_segment_osrm_distance1']=d1.groupby(['trip_uuid','source_center','destinat
```

In [20]:
```
# aggregation of below mentioned based on their Trip_uuid, Source ID and Destinatio
# as they are mentioned as a cumsum in data dictionary we will take max
d1['agg_distance_to_destination']=d1.groupby(['trip_uuid','source_center','destina
d1['agg_actual_time']=d1.groupby(['trip_uuid','source_center','destination_center'
d1['agg_osrm_time']=d1.groupby(['trip_uuid','source_center','destination_center'])
d1['agg_osrm_distance']=d1.groupby(['trip_uuid','source_center','destination_cente
```

In [21]:
```
#creating column with city place state from source centre & destination centre
d1[['Source_City','Source_Place','Source_Code/State']]=d1['source_name'].str.rsplit
d1[['destination_City','destination_Place','destination_Code/State']]=d1['destinat
```

In [22]:
```
#creating column with city place state from source centre & destination centre
d1[['Source_Code','Source_State']]=d1['Source_Code/State'].str.rsplit('(',2, expan
d1[['destination_Code','destination_State']]=d1['destination_Code/State'].str.rspl
```

In [23]:
```
#dropping the existing columns as we have already got engineered features from the
d1.drop(columns=['od_end_time','od_start_time','trip_creation_time','source_name',
```

In [24]:
```
d1.shape
```

Out[24]:
```
(144316, 36)
```

In [25]:
```
d1.duplicated().sum()
```

Out[25]:
```
0
```

We see there are no duplicated for now in the dataset. The purpose of feature engineering was to create one row for one unique uuid. Lets drop those features from which we have already generated new features & again check for the duplicates. We will definietly find duplicated now and we will remove them

In [26]:
```
df_merge=d1.loc[:,['route_type', 'trip_uuid',
       'start_scan_to_end_scan',  'trip_creation_month',
       'trip_creation_year', 'trip_creation_day', 'Timediff_start_end_H', 'agg_segm
       'agg_segment_osrm_time1', 'agg_segment_osrm_distance1',
       'agg_distance_to_destination', 'agg_actual_time', 'agg_osrm_time',
       'agg_osrm_distance', 'Source_City', 'Source_Place', 'Source_Code/State',
       'destination_City', 'destination_Place', 'destination_Code/State']]
```

In [27]:
```
df_merge.duplicated().sum()
```

Out[27]:
```
118093
```

In [28]:
```
df_merge.drop_duplicates(inplace=True)
```

In [29]:
```
df_merge[df_merge['trip_uuid']=='trip-153671041653548748']
```

Out[29]:

	route_type	trip_uuid	start_scan_to_end_scan	trip_creation_month	trip_creatior
124981	FTL	trip-153671041653548748	999.0	9	
125002	FTL	trip-153671041653548748	1260.0	9	

Note that the above merging of rows has been performed on the basis of trip_uuid,source_centre,destination_centre

we can also further merge the rows on the basis of trip_uuid

In [30]:
```
df_merge.duplicated().sum()
```

Out[30]:
```
0
```

In [31]:
```
df_merge.shape
```

Out[31]:
```
(26223, 20)
```

In [32]:
```
df_uuid=df_merge.copy()
```

In [33]:
```
# aggregation of below mentioned based on their Trip_uuid, Source ID and Destinatio
# as they are mentioned as a cumsum in data dictionary we will take max
df_uuid['start_scan_to_end_scan11']=df_uuid.groupby(['trip_uuid'])['start_scan_to_e
df_uuid['Timediff_start_end_H11']=df_uuid.groupby(['trip_uuid'])['Timediff_start_er
df_uuid['agg_segment_actual_time11']=df_uuid.groupby(['trip_uuid'])['agg_segment_ac
df_uuid['agg_segment_osrm_time11']=df_uuid.groupby(['trip_uuid'])['agg_segment_osrm

df_uuid['agg_segment_osrm_distance11']=df_uuid.groupby(['trip_uuid'])['agg_segment_
df_uuid['agg_distance_to_destination11']=df_uuid.groupby(['trip_uuid'])['agg_distan
df_uuid['agg_actual_time11']=df_uuid.groupby(['trip_uuid'])['agg_actual_time'].tran
df_uuid['agg_osrm_time11']=df_uuid.groupby(['trip_uuid'])['agg_osrm_time'].transfor
df_uuid['agg_osrm_distance11']=df_uuid.groupby(['trip_uuid'])['agg_osrm_distance']

df_uuid['Source_City11']=df_uuid.groupby(['trip_uuid'])['Source_City'].transform('
df_uuid['Source_Place11']=df_uuid.groupby(['trip_uuid'])['Source_Place'].transform
df_uuid['Source_Code/State11']=df_uuid.groupby(['trip_uuid'])['Source_Code/State']
df_uuid['destination_City11']=df_uuid.groupby(['trip_uuid'])['destination_City'].tr
df_uuid['destination_Place11']=df_uuid.groupby(['trip_uuid'])['destination_Place']
df_uuid['destination_Code/State11']=df_uuid.groupby(['trip_uuid'])['destination_Co
```

In [34]:
```
df1=df_uuid.loc[:,['route_type', 'trip_uuid',
        'trip_creation_month', 'trip_creation_year', 'trip_creation_day',
       'start_scan_to_end_scan11', 'Timediff_start_end_H11',
       'agg_segment_actual_time11', 'agg_segment_osrm_time11',
       'agg_segment_osrm_distance11', 'agg_distance_to_destination11',
       'agg_actual_time11', 'agg_osrm_time11', 'agg_osrm_distance11',
       'Source_City11', 'Source_Place11', 'Source_Code/State11',
       'destination_City11', 'destination_Place11',
       'destination_Code/State11']]
```

In [35]:
```
df1.duplicated().sum()
```

Out[35]:
```
11436
```

In [36]:
```
#before merging checking 1 unique uuid for cross check purpose
df1[df1['trip_uuid']=='trip-153671041653548748']
```

Out[36]:

	route_type	trip_uuid	trip_creation_month	trip_creation_year	trip_creation_day
124981	FTL	trip-153671041653548748	9	2018	12
125002	FTL	trip-153671041653548748	9	2018	12

In [37]:
```
df1.drop_duplicates(inplace=True)
```

In [38]:
```
df1.duplicated().sum()
```

Out[38]:
```
0
```

In [39]:
```
df1.shape
```

Out[39]:
```
(14787, 20)
```

So after the entire merging the final dataset that has been created with multiple induced features and distict rows has:

- **14787 rows**
- **20 columns**

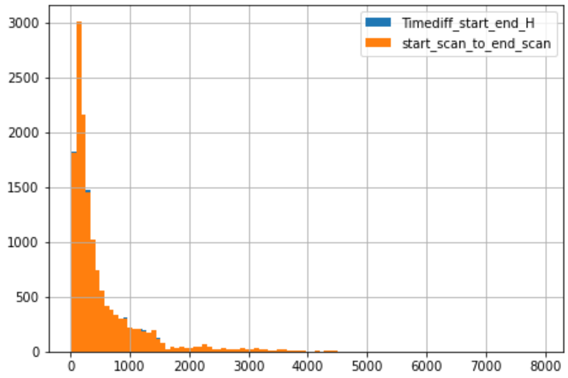
Comparing the difference between time taken between od_start_time and od_end_time and start_scan_to_end_scan.

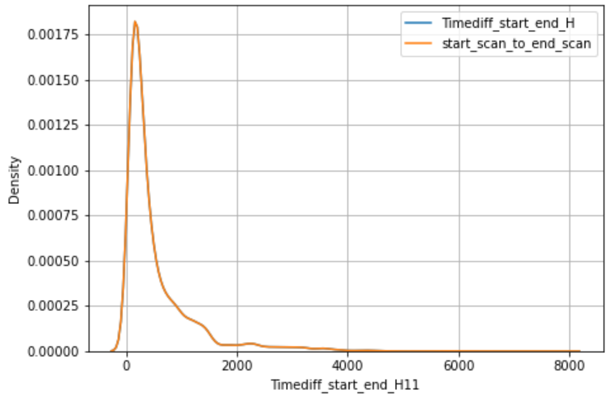
In [40]:
```
df1['scan_diff_check']=df1['Timediff_start_end_H11']-df1['start_scan_to_end_scan11
```

In [41]:
```
#checking if there is any value where difference is > 1 min
# df1[df1['scan_diff_check']>1]
```

Visualising the Difference between the two start_scan_to_end_scan & Timediff_start_end_H:

In [42]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
plt.hist(df1['Timediff_start_end_H11'],bins=100,label='Timediff_start_end_H')
plt.hist(df1['start_scan_to_end_scan11'],bins=100,label='start_scan_to_end_scan')
plt.legend()
plt.grid()
plt.subplot(122)
sns.kdeplot(df1['Timediff_start_end_H11'],label='Timediff_start_end_H')
sns.kdeplot(df1['start_scan_to_end_scan11'],label='start_scan_to_end_scan')
plt.legend()
plt.grid()
plt.show()
```





- As We can clearly see both the curves are overlapping which means **they are fairly same.**

Now lets proof the above comparision using hypothesis Testing

Defining Null & Alternate Hypothesis

- $H0$: μa = μo the mean for Timediff_start_end_H & start_scan_to_end_scan are same
- Ha : μa > μo the mean for start_scan_to_end_scan is more than start_scan_to_end_scan

Defining T-statistics:

We will perform 2 tailed test as variance of the population is not known

In order to test them we shall run a one-tailed t-test using an arbitrated α level of 0.05

- α =0.05

In [43]:
```
def htResult(p_value):
    significance_level = 0.05
    if p_value <= significance_level:
        print('Reject NULL HYPOTHESIS')
    else:
        print('Fail to Reject NULL HYPOTHESIS')
```

In [44]:
```
import scipy.stats as stats
st,p = stats.ttest_ind(df1['Timediff_start_end_H11'],df1['start_scan_to_end_scan11
print('P-value :',(p/2))

htResult(p/2)
```

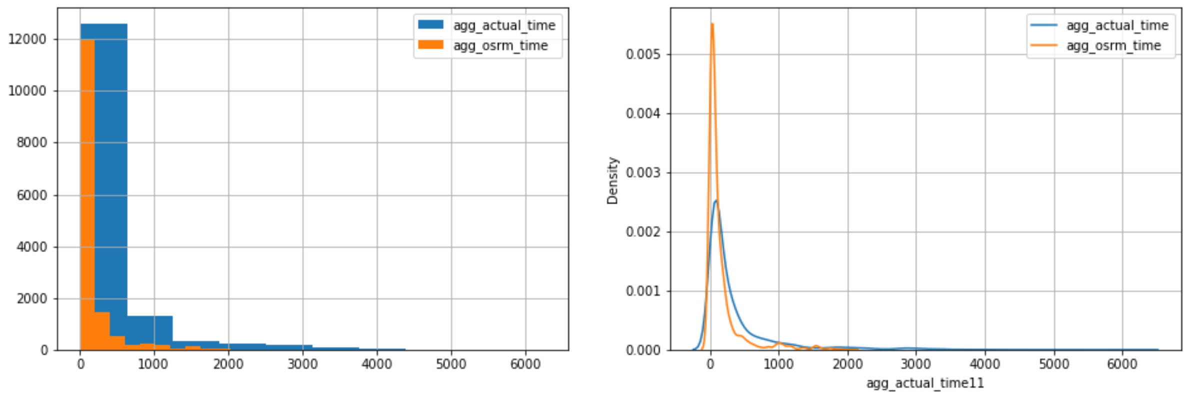
```
P-value : 0.454017401571O2755
Fail to Reject NULL HYPOTHESIS
```

- As p value> alpha We fail so reject the null hypothesis @ there is no mean difference between start_end_H and start_scan_to_end_scan

Visual analysis

Comparision Between Aggregate Actual time & Aggregate OSRM Time

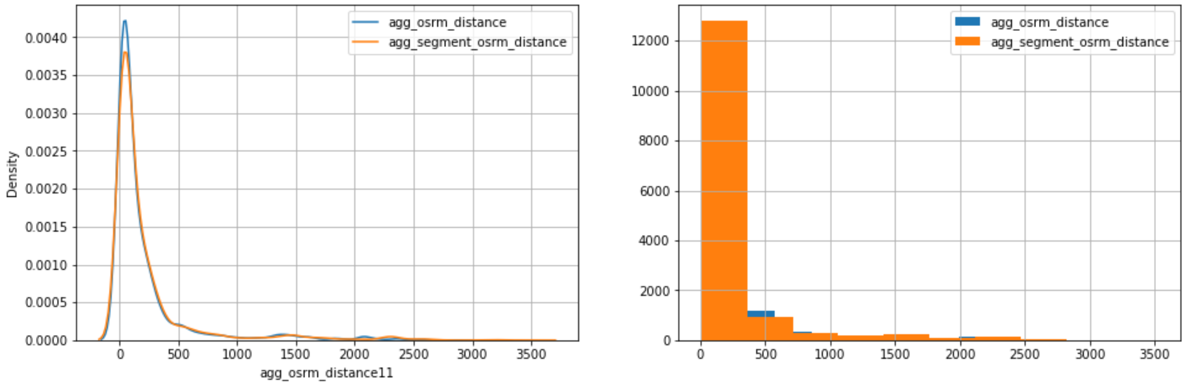
In [45]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
plt.hist(df1['agg_actual_time11'],bins=10,label='agg_actual_time')
plt.hist(df1['agg_osrm_time11'],bins=10,label='agg_osrm_time')
plt.legend()
plt.grid()
plt.subplot(122)
sns.kdeplot(df1['agg_actual_time11'],label='agg_actual_time')
sns.kdeplot(df1['agg_osrm_time11'],label='agg_osrm_time')
plt.legend()
plt.grid()
plt.show()
```



- Clearly visible, agg_actual_time is somewhat greater than agg_osrm_time and which is intutive as well

Comparision Between Aggregate OSRM distance & Aggregate Segment osrm distance

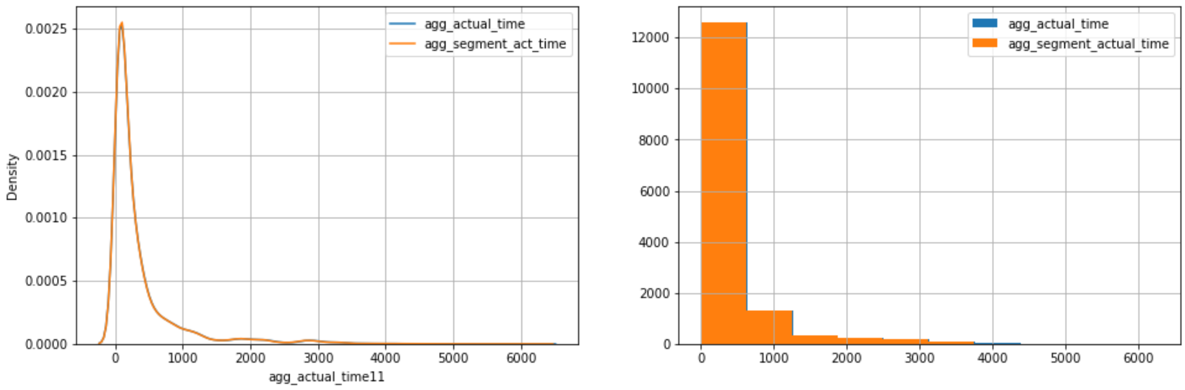
In [46]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
sns.kdeplot(df1['agg_osrm_distance11'],label='agg_osrm_distance')
sns.kdeplot(df1['agg_segment_osrm_distance11'],label='agg_segment_osrm_distance')
plt.legend()
plt.grid()
plt.subplot(122)
plt.hist(df1['agg_osrm_distance11'],bins=10,label='agg_osrm_distance')
plt.hist(df1['agg_segment_osrm_distance11'],bins=10,label='agg_segment_osrm_distan
plt.legend()
plt.grid()
plt.show()
```



- Clearly visible there is a overlap between both the variables which suggests segment distances & agg distances are almost same

Comparision Between Aggregate Actual time & Aggregate segment_actual_time

In [47]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
sns.kdeplot(df1['agg_actual_time11'],label='agg_actual_time')
sns.kdeplot(df1['agg_segment_actual_time11'],label='agg_segment_act_time')
plt.legend()
plt.grid()
plt.subplot(122)
plt.hist(df1['agg_actual_time11'],bins=10,label='agg_actual_time')
plt.hist(df1['agg_segment_actual_time11'],bins=10,label='agg_segment_actual_time')
plt.legend()
plt.grid()
plt.show()
```

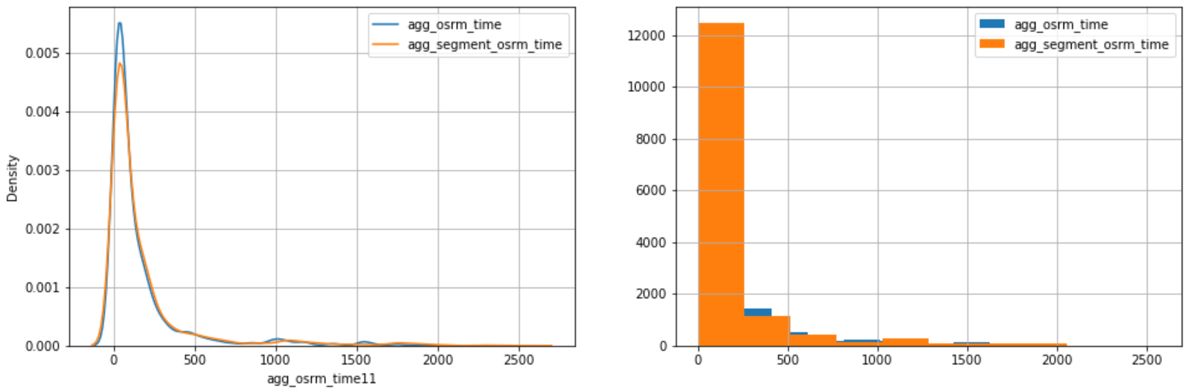


- Clearly visible there is a overlap between both the variables which suggests segment distances & agg distances are almost same

Comparision Between Aggregate OSRM time & Aggregate Segment OSRM Time

In [48]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
sns.kdeplot(df1['agg_osrm_time11'],label='agg_osrm_time')
sns.kdeplot(df1['agg_segment_osrm_time11'],label='agg_segment_osrm_time')
plt.legend()
plt.grid()
plt.subplot(122)
```

```
plt.hist(df1['agg_osrm_time11'],bins=10,label='agg_osrm_time')
plt.hist(df1['agg_segment_osrm_time11'],bins=10,label='agg_segment_osrm_time')
plt.legend()
plt.grid()
plt.show()
```



- Clearly visible there is a overlap between both the variables which suggests osrm time & segment osrm times are almost same

Now lets do some hypothesis testing on the above comparisions:

Hypothesis test of mean difference of Agg_actual_time & agg_osrm_time

Defining Null & Alternate Hypothesis

- $H_0 : \mu_a = \mu_o$ the mean for agg_Actual_time & agg_osrm_time are same
- $H_a : \mu_a > \mu_o$ the mean for agg_Actual_time is more than agg_osrm_time

In order to test them we shall run a one-tailed t-test using an arbitrated α level of 0.05

- α =0.05

In [49]:
```
import scipy.stats as stats
st,p = stats.ttest_ind(df1['agg_actual_time11'],df1['agg_osrm_time11'])
print('P-value :',(p/2))

htResult(p/2)
```

```
P-value : 1.1794664115414192e-307
Reject NULL HYPOTHESIS
```

- As **p< α** we have enough evidence to reject the null hypothesis @ there is a difference between the mean for agg_Actual_time & agg_osrm_time

Hypothesis test of mean difference of Agg_actual_time & agg_segment_actual_time

- $H_0 : \mu_a = \mu_o$ the mean for agg_Actual_time & agg_segment_actual_time are same
- $H_a : \mu_a > \mu_o$ the mean for agg_Actual_time & agg_segment_actual_time are different

In [50]:
```
import scipy.stats as stats
st,p = stats.ttest_ind(df1['agg_actual_time11'],df1['agg_segment_actual_time11'])
print('P-value :',(p/2))

htResult(p/2)
```

```
P-value : 0.30929173856919245
Fail to Reject NULL HYPOTHESIS
```

- As **p> α** we have enough evidence to reject the null hypothesis @ there is a difference between the mean for agg_Actual_time & agg_segment_actual_time

Hypothesis test of mean difference of Agg_osrm_distance & agg_segment_osrm_distance

- $H_0 : \mu_a = \mu_o$ the mean for Agg_osrm_distance & agg_segment_osrm_distance are same
- $H_a : \mu_a > \mu_o$ the mean for Agg_osrm_distance & agg_segment_osrm_distance are different

In [51]:
```
import scipy.stats as stats
st,p = stats.ttest_ind(df1['agg_osrm_distance11'],df1['agg_segment_osrm_distance11
print('P-value :',(p/2))

htResult(p/2)
```

```
P-value : 4.118038087190506e-05
Reject NULL HYPOTHESIS
```

- As **p< α** we have enough evidence to reject the null hypothesis @ there is no difference between the mean for Agg_osrm_distance & agg_segment_osrm_distance

Exploratory Data Analysis

Univariate Data Analysis

In [52]:
```
num_cols = df.select_dtypes('float64').columns.values
cat_cols = df1.select_dtypes('object').columns.values
```

In [53]:
```
num_cols=df1.select_dtypes('float64').columns.values
for i in num_cols:
    print('##############')
    print(df1[i].value_counts())
```

```
##############
148.0     51
115.0     51
87.0      50
113.0     49
128.0     49
          ..
1895.0     1
1634.0     1
1199.0     1
1205.0     1
2429.0     1
Name: start_scan_to_end_scan11, Length: 2203, dtype: int64
##############
319.61     4
286.63     4
122.43     4
147.10     4
86.20      4
          ..
227.87     1
924.06     1
658.28     1
3732.37    1
427.69     1
Name: Timediff_start_end_H11, Length: 13573, dtype: int64
##############
47.0      121
41.0      112
60.0      107
35.0      100
55.0      100
         ...
1120.0      1
1684.0      1
2862.0      1
2662.0      1
2750.0      1
Name: agg_segment_actual_time11, Length: 1885, dtype: int64
##############
17.0      221
20.0      213
19.0      210
18.0      209
22.0      208
         ...
1845.0      1
2422.0      1
938.0       1
1185.0      1
1723.0      1
Name: agg_segment_osrm_time11, Length: 1240, dtype: int64
##############
42.2424     2
139.9819    2
27.5998     2
27.6689     2
53.1450     2
           ..
33.1927     1
120.2104    1
234.5058    1
213.8828    1
131.1238    1
```

```
Name: agg_segment_osrm_distance11, Length: 14724, dtype: int64
##############
32.177092     2
27.959967     2
18.036366     2
25.878761     2
195.585266    2
             ..
176.696070    1
187.836591    1
167.384229    1
31.552168     1
73.680667     1
Name: agg_distance_to_destination11, Length: 14771, dtype: int64
##############
60.0      134
50.0      130
42.0      121
48.0      115
38.0      111
         ...
966.0       1
2817.0      1
2331.0      1
2379.0      1
2784.0      1
Name: agg_actual_time11, Length: 1851, dtype: int64
##############
20.0      265
34.0      265
29.0      254
23.0      239
32.0      235
         ...
967.0       1
1087.0      1
782.0       1
1744.0      1
702.0       1
Name: agg_osrm_time11, Length: 827, dtype: int64
##############
15.8695     2
42.3121     2
53.3295     2
61.0186     2
37.0811     2
           ..
150.0793    1
44.1600     1
103.9523    1
267.3213    1
111.2709    1
Name: agg_osrm_distance11, Length: 14706, dtype: int64
##############
0.88    116
0.87    109
0.50    103
0.75    101
0.31     97
       ...
1.66      1
1.48      1
2.27      1
3.61      1
```

```
3.81       1
Name: scan_diff_check, Length: 1231, dtype: int64
```

In [54]:

```
for i in cat_cols:
    print('##############')
    print(df1[i].value_counts())
```

```
##############
Carting    8906
FTL        5881
Name: route_type, dtype: int64
##############
trip-153741093647649320    1
trip-153836648611826977    1
trip-153681920064110379    1
trip-153744931166370622    1
trip-153764628243892763    1
                          ..
trip-153741177166786003    1
trip-153801210039247977    1
trip-153737819969505360    1
trip-153739632610417618    1
trip-153746066843555182    1
Name: trip_uuid, Length: 14787, dtype: int64
##############
Bengaluru                1014
Gurgaon                  1011
Bhiwandi                  811
Bangalore                 731
Delhi                     617
                         ...
Parvathipuram_Central       1
Koraput                     1
Jasai                       1
Baripada                    1
Ashta                       1
Name: Source_City11, Length: 706, dtype: int64
##############
Bilaspur           959
Mankoli            811
Nelmngla           732
H                  643
I                  571
                  ...
Ymunpurm             1
Shahdara (Delhi)     1
KalikDPP             1
PuranDPP             1
ShantiNg             1
Name: Source_Place11, Length: 672, dtype: int64
##############
HB (Haryana)         937
HB (Maharashtra)     811
HB (Karnataka)       757
H (Karnataka)        751
H (Punjab)           370
                    ...
2 (Andhra Pradesh)     1
1 (Andhra Pradesh)     1
2 (Karnataka)          1
D (Meghalaya)          1
9 (Gujarat)            1
Name: Source_Code/State11, Length: 181, dtype: int64
##############
Bengaluru    1056
Gurgaon       869
Mumbai        814
Hyderabad     630
Bangalore     628
             ...
Shahabad        1
```

```
Nadiad          1
Mussoorie       1
Jairampur       1
Kanigiri        1
Name: destination_City11, Length: 799, dtype: int64
##############
Bilaspur    856
Nelmngla    628
Mankoli     604
H           542
I           488
           ...
JJCpxDPP      1
GhtimDPP      1
Thsil3PL      1
Sector02      1
Ghansoli      1
Name: destination_Place11, Length: 747, dtype: int64
##############
HB (Haryana)        813
H (Karnataka)       655
HB (Karnataka)      589
HB (Maharashtra)    573
H (Punjab)          431
                   ...
L (Punjab)            1
I (Tripura)           1
7 (Maharashtra)       1
4 (Maharashtra)       1
L (Kerala)            1
Name: destination_Code/State11, Length: 184, dtype: int64
```

Lets try to find Bussiest Corridor

In [55]:
```python
d_grouped.route_type.max()
```

Out[55]:
```
81
```

In [56]:
```python
# find trip uuid of max count
d_grouped[d_grouped['route_type']==81]

df[df['trip_uuid']=='trip-153755502932196495']
```

Out[56]:

	trip_creation_time	route_type	trip_uuid	source_center	source_nam
61008	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61009	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61010	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61011	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61012	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
...	...	...	...	...	.
61084	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61085	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61086	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61087	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab
61088	2018-09-21 18:37:09.322207	FTL	trip-153755502932196495	IND160002AAC	Chandigarh_Mehmdpur_I (Punjab

81 rows × 17 columns

In [57]:

```
df1[df1['trip_uuid']=='trip-153755502932196495'][['agg_segment_actual_time11',
       'agg_segment_osrm_time11', 'agg_segment_osrm_distance11',
       'agg_distance_to_destination11', 'agg_actual_time11', 'agg_osrm_time11']]
```

Out[57]:

	agg_segment_actual_time11	agg_segment_osrm_time11	agg_segment_osrm_distance11	agg
61008	3751.0	1864.0	2500.2145	

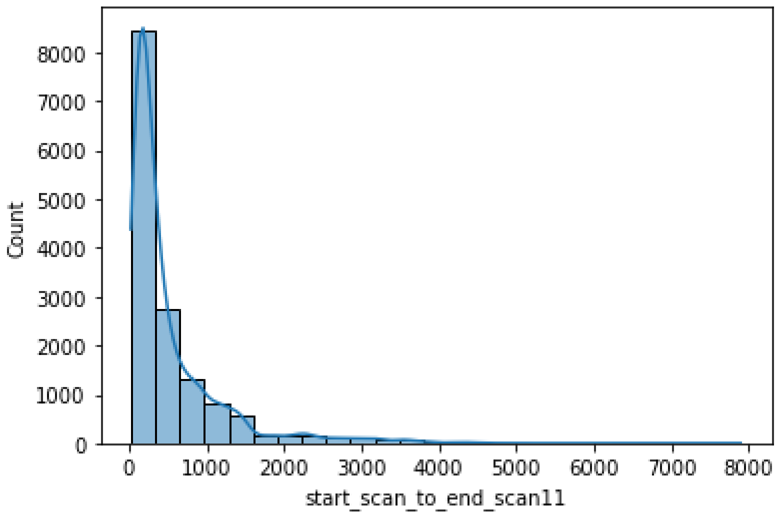
- Bussiest Corridor is from source Chandigarh_Mehmdpur_H (Punjab) to Bangalore_Nelmngla_H (Karnataka) Average_distance between them is 1927 kms & average time taken is 3784 mins

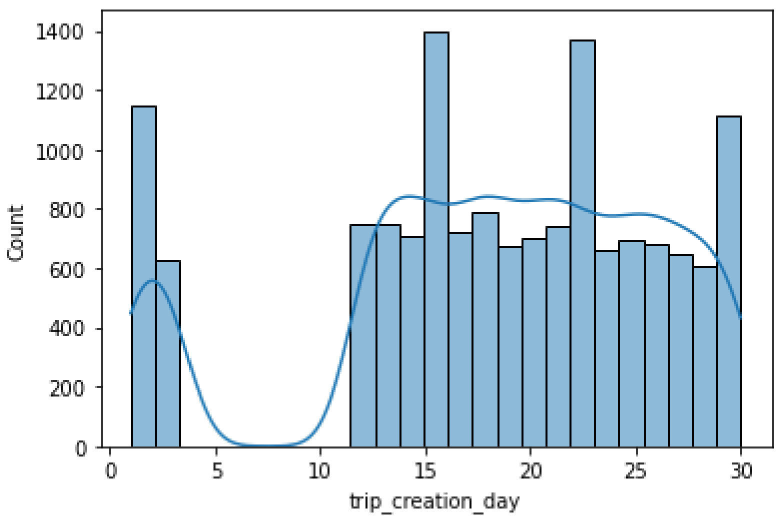
In [58]:

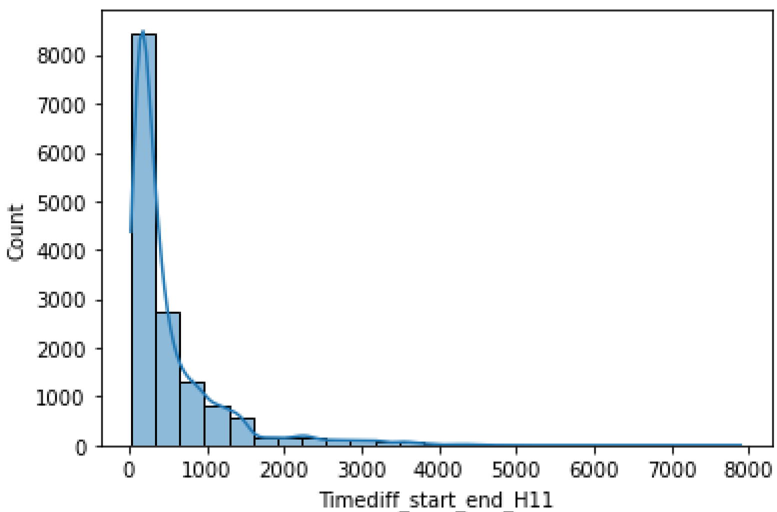
```
l=['start_scan_to_end_scan11',
       'trip_creation_day', 'Timediff_start_end_H11', 'agg_segment_actual_time11',
       'agg_segment_osrm_time11', 'agg_segment_osrm_distance11',
       'agg_distance_to_destination11', 'agg_actual_time11', 'agg_osrm_time11',
       'agg_osrm_distance11']
```

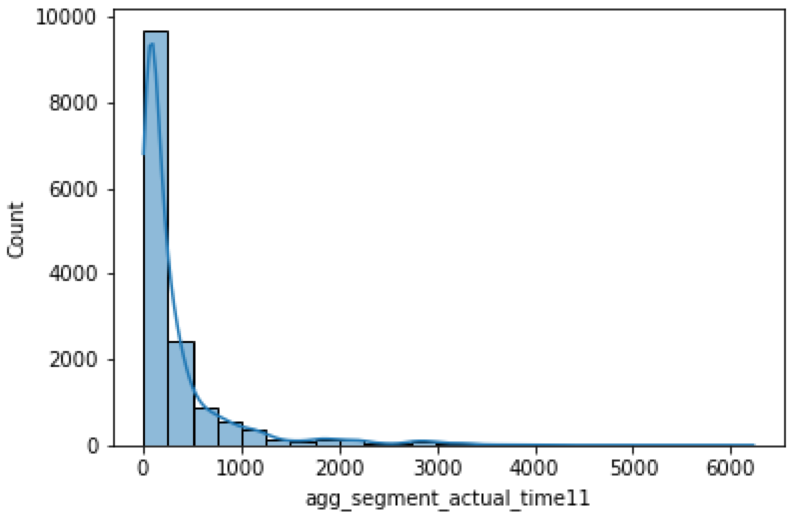
In [59]:

```
for i in l:
    sns.histplot(data = df1, x = df1[i],bins = 25,kde=True)
    plt.show()
```

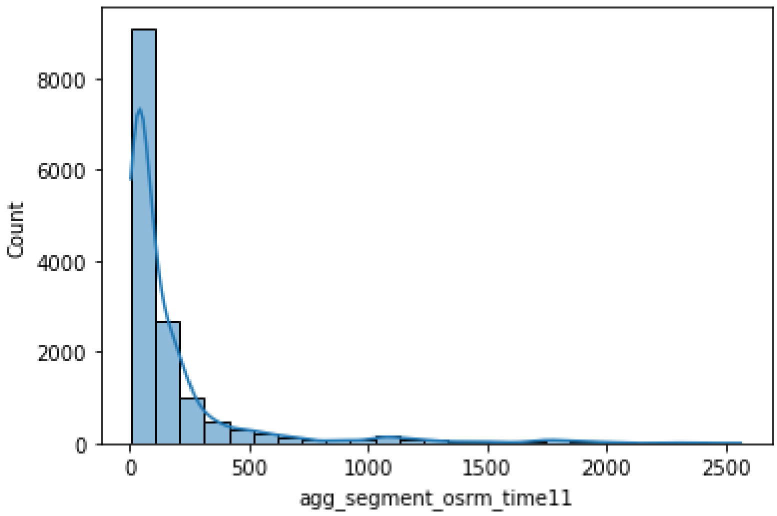

Count
start_scan_to_end_scan11


Count
trip_creation_day

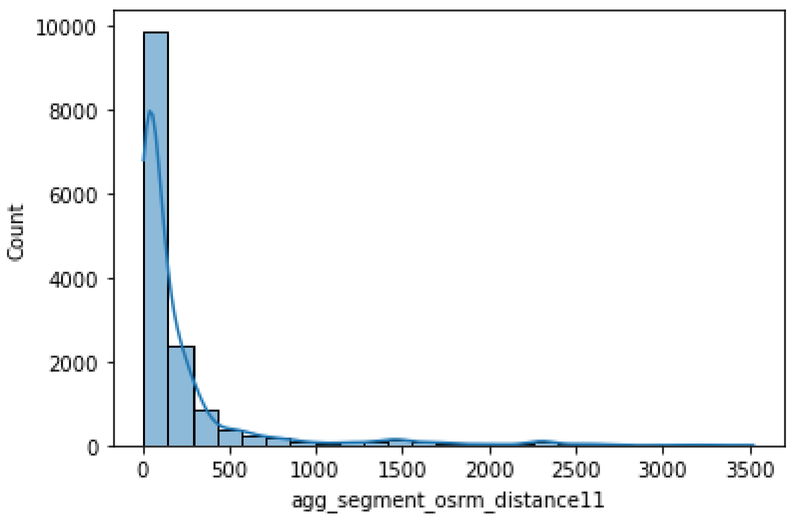

Count
Timediff_start_end_H11



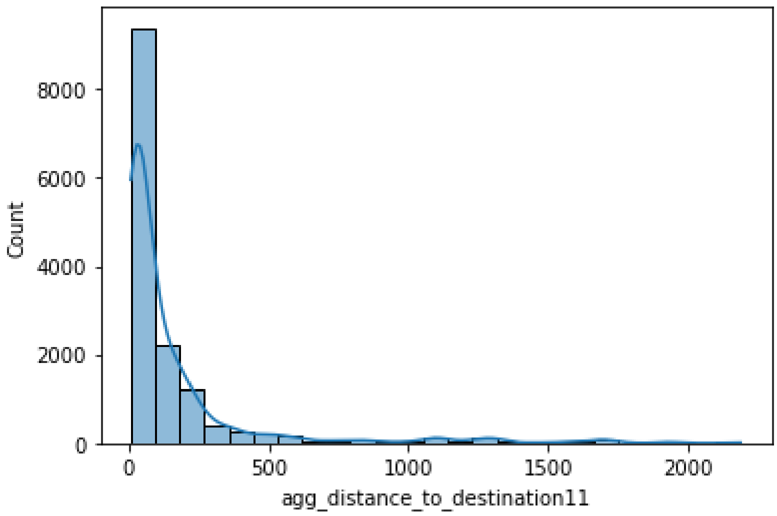
Count
agg_segment_actual_time11

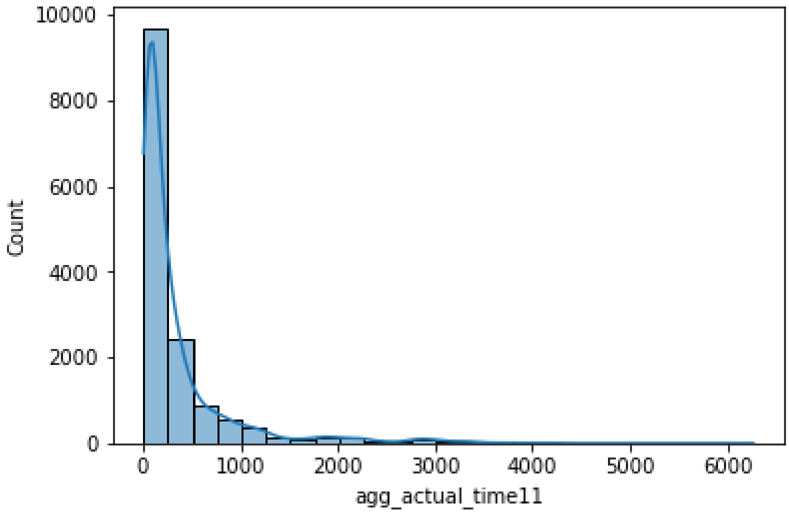


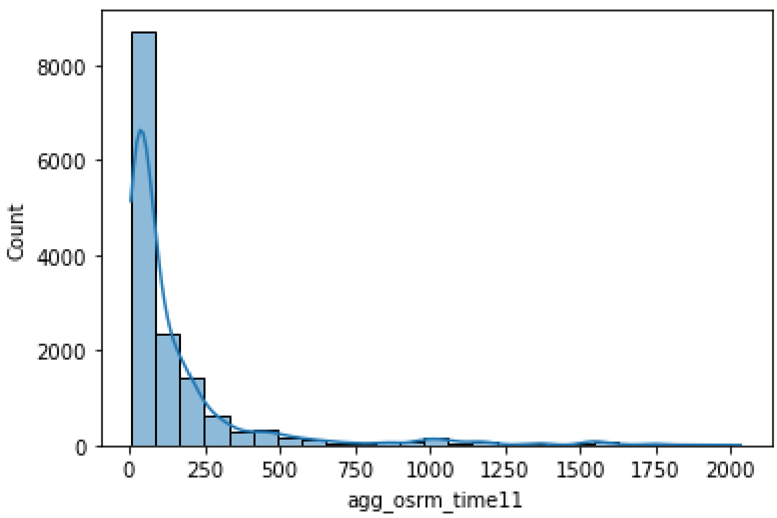
Count
agg_segment_osrm_time11

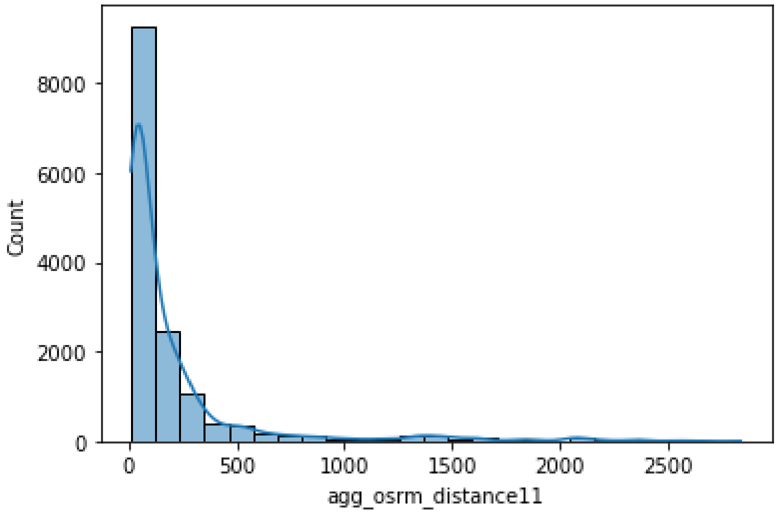


Count
agg_segment_osrm_distance11


Count
0
2000
4000
6000
8000
0
500
1000
1500
2000
agg_distance_to_destination11


Count
0
2000
4000
6000
8000
10000
0
1000
2000
3000
4000
5000
6000
agg_actual_time11


Count
0
2000
4000
6000
8000
0
250
500
750
1000
1250
1500
1750
2000
agg_osrm_time11



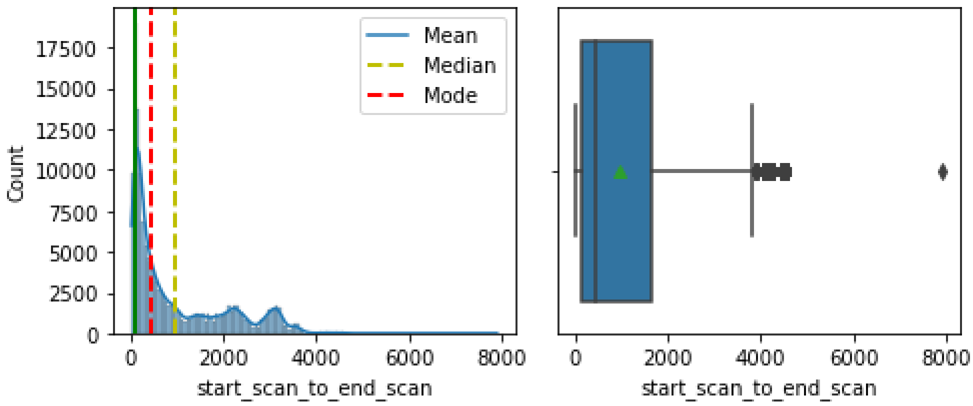
- The entire data is heavily right skewed

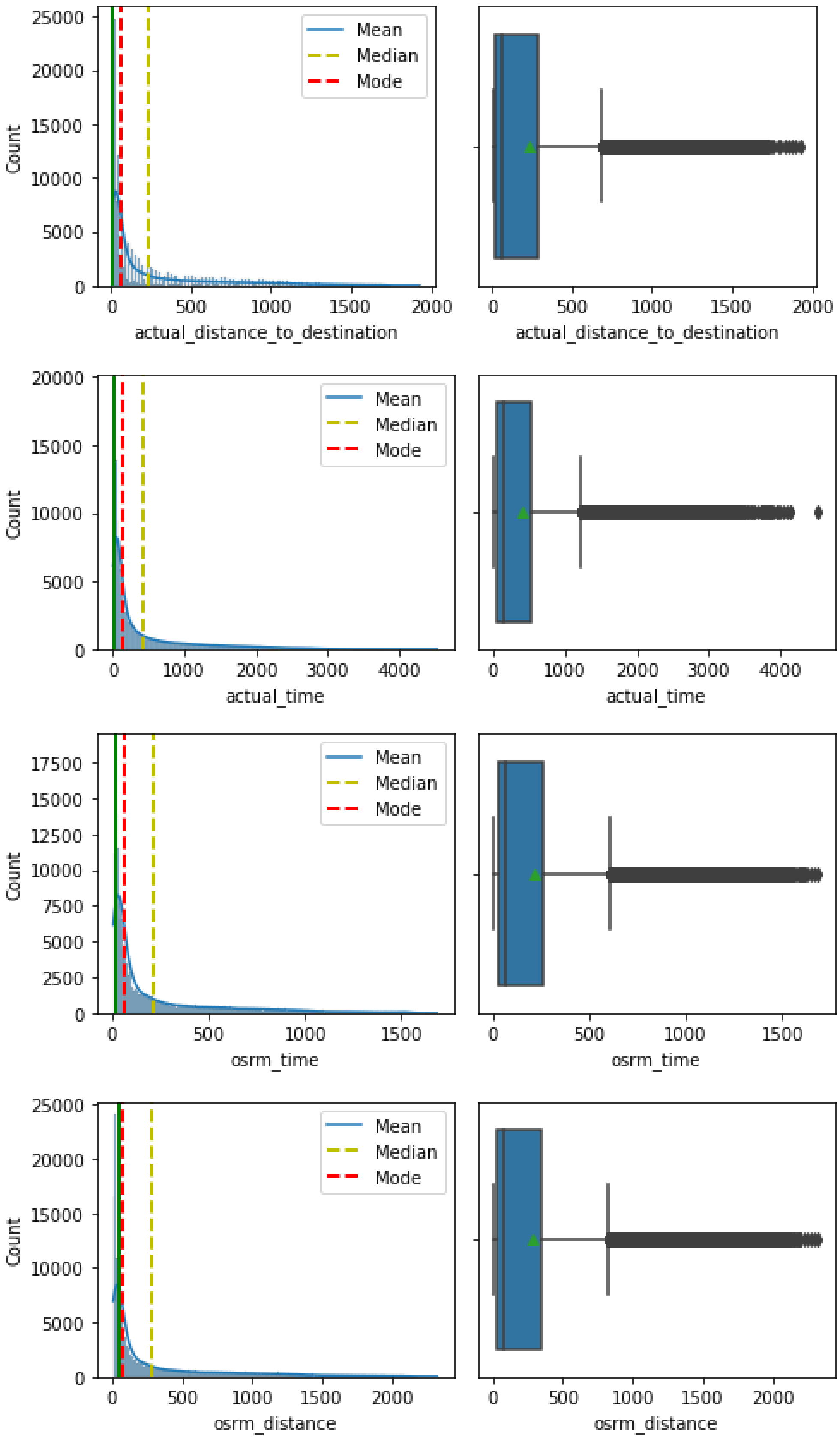
Outlier Detection & their Treatment

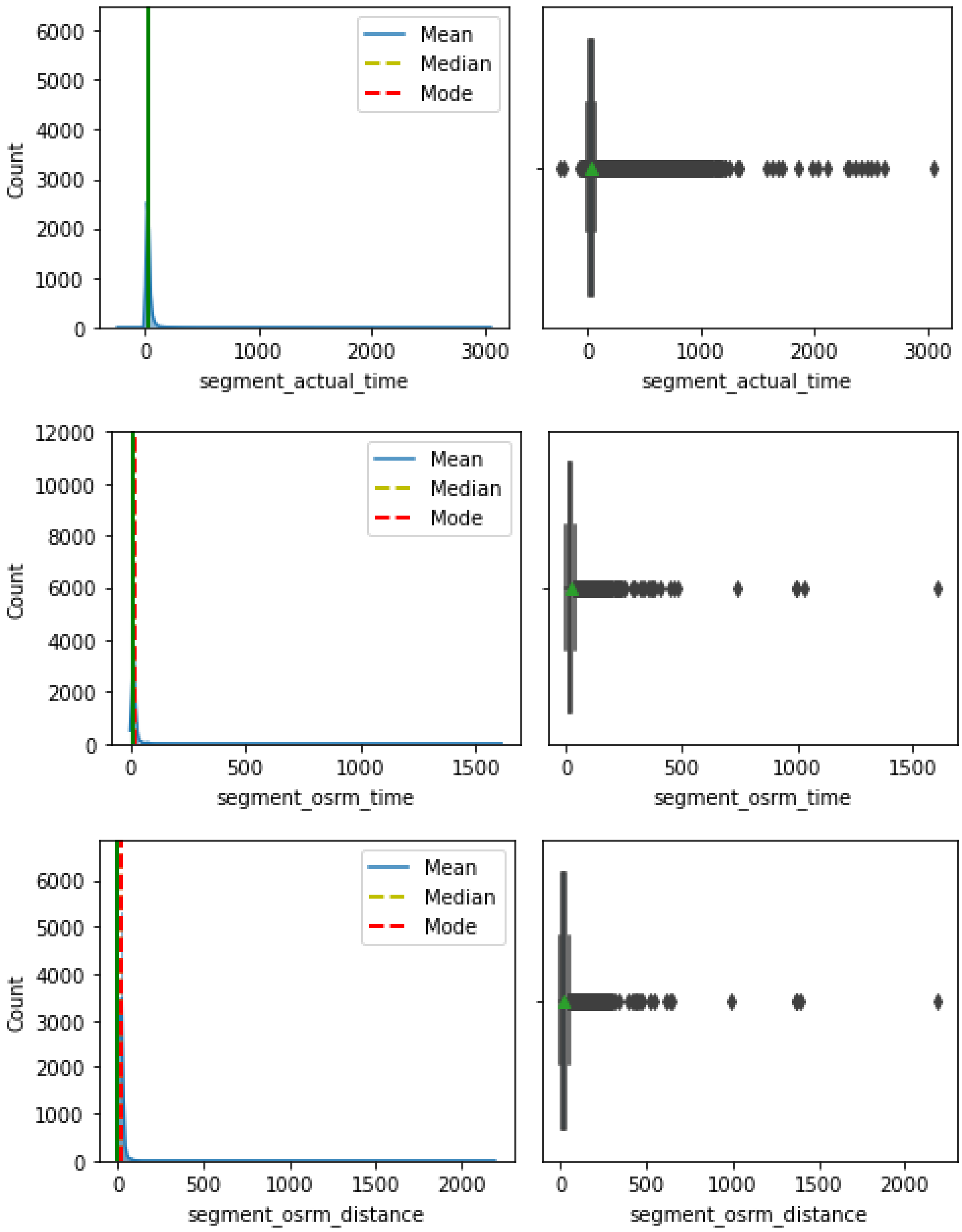
In [60]:
```
def uni(d):
    f,ax = plt.subplots(nrows=1,ncols=2,figsize=(7,3))
    sns.histplot(d, kde=True, ax=ax[0])
    ax[0].axvline(d.mean(), color='y', linestyle='--',linewidth=2)
    ax[0].axvline(d.median(), color='r', linestyle='dashed', linewidth=2)
    ax[0].axvline(d.mode()[0],color='g',linestyle='solid',linewidth=2)
    ax[0].legend({'Mean':d.mean(),'Median':d.median(),'Mode':d.mode()})

    sns.boxplot(x=d, showmeans=True, ax=ax[1])
    plt.tight_layout()
```

In [61]:
```
num_cols = df.select_dtypes('float64').columns.values
```

In [62]:
```
for f in num_cols:
    uni(df[f])
plt.show()
```




Mean
Median
Mode
Count
actual_distance_to_destination
actual_distance_to_destination
Mean
Median
Mode
Count
actual_time
actual_time
Mean
Median
Mode
Count
osrm_time
osrm_time
Mean
Median
Mode
Count
osrm_distance
osrm_distance

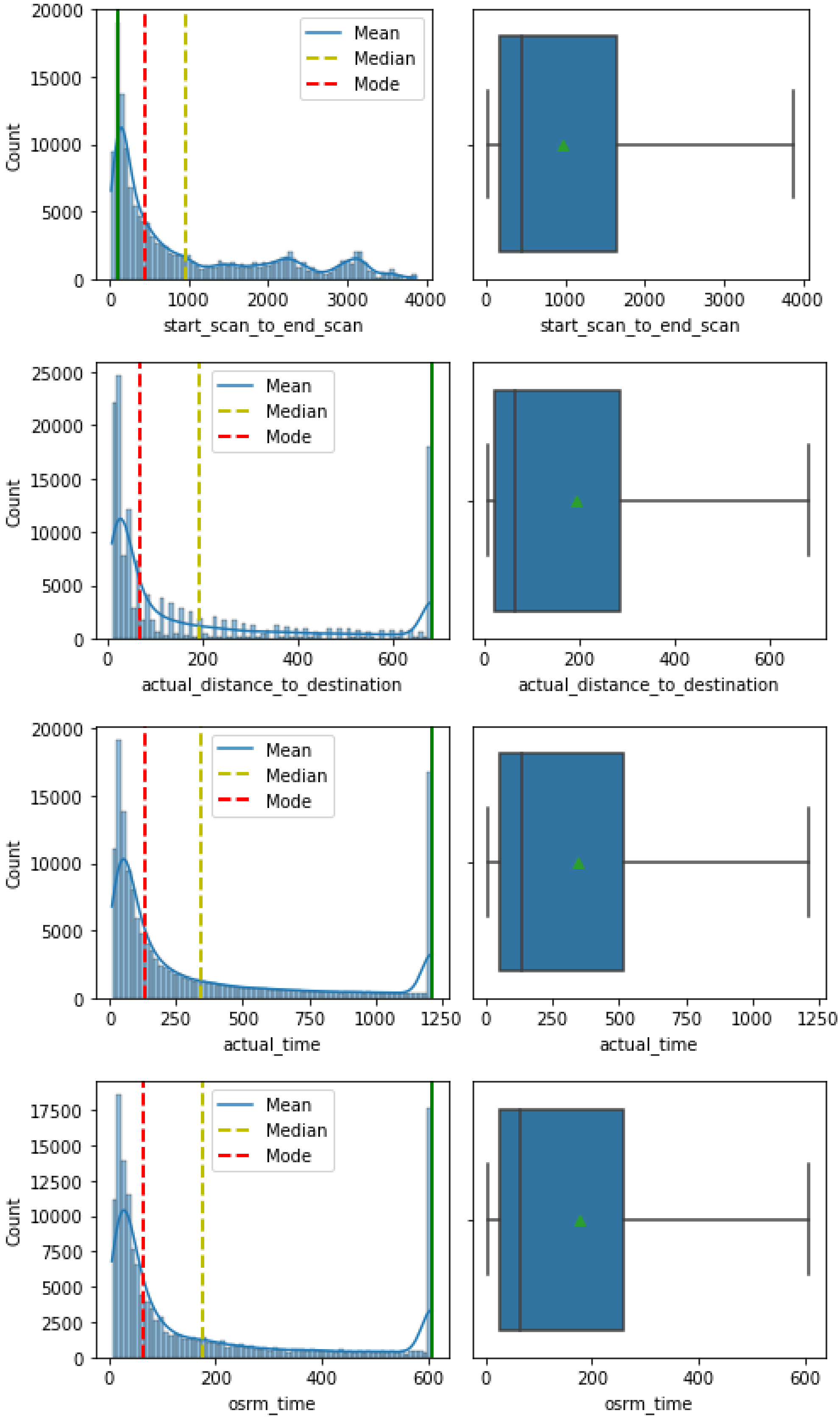


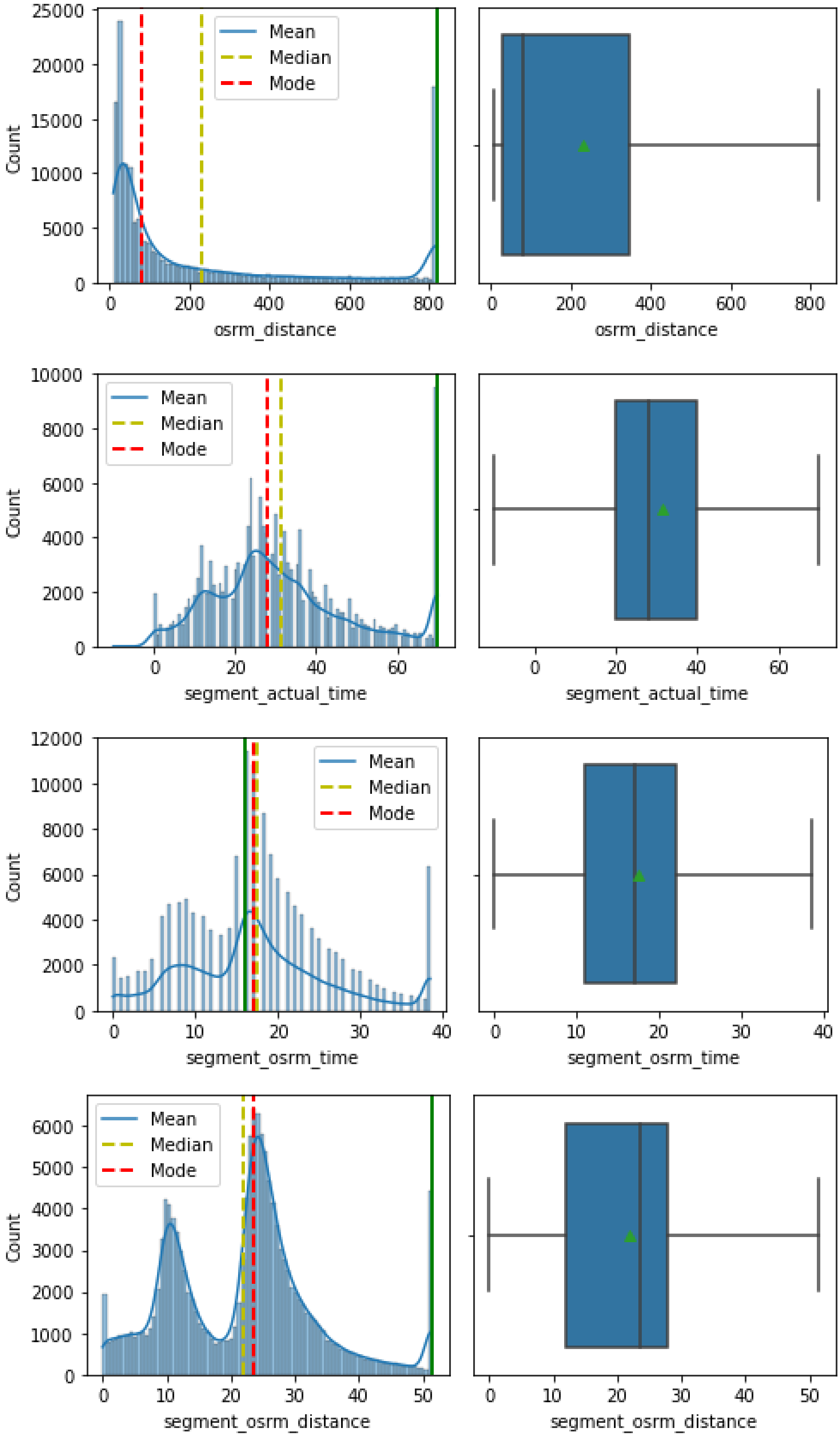
- There are outliers in almost all of the numeric feature

In [63]:
```
#treating outliers:
def treat_outlier(variable):
    #Takes two parameters: dataframe & variable of interest as string
    q1,q3=np.percentile(variable,[25,75])
    iqr = q3-q1
    lo_range = q1-(1.5*iqr)
    up_range = q3+(1.5*iqr)
    return lo_range,up_range
```

In [64]:
```
for col in num_cols:
    ir,ur=treat_outlier(df[col])
    df[col]=np.where(df[col]>ur,ur,df[col])
    df[col]=np.where(df[col]<ir,ir,df[col])
```

In [65]:
```
#Lets check where outliers are removed or not:
for f in num_cols:
    uni(df[f])
plt.show()
```

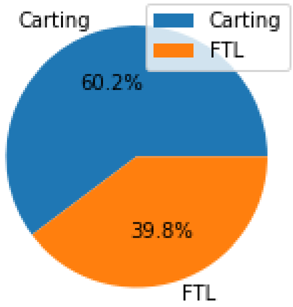

Mean
Median
Mode
Count
start_scan_to_end_scan
start_scan_to_end_scan
Mean
Median
Mode
Count
actual_distance_to_destination
actual_distance_to_destination
Mean
Median
Mode
Count
actual_time
actual_time
Mean
Median
Mode
Count
osrm_time
osrm_time



Outliers are removed

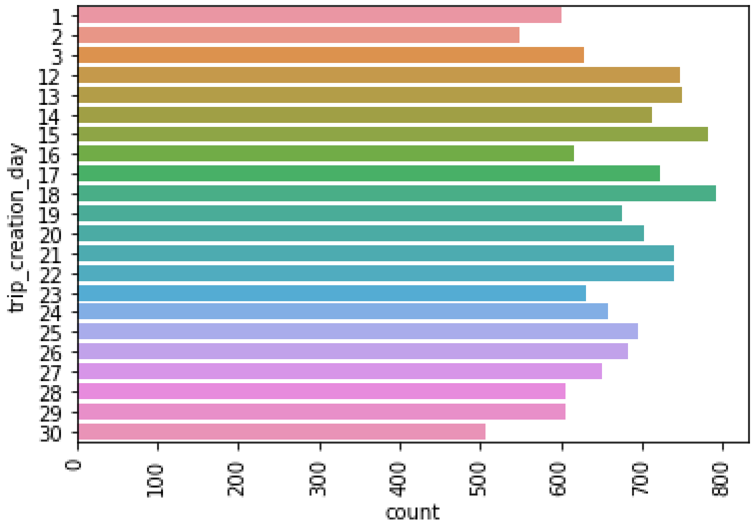
In [66]:

```
fig1, ax1 = plt.subplots(figsize=(7,3))
ax1.pie(df1['route_type'].value_counts(), labels=df1['route_type'].unique(), autopc
plt.legend()
plt.show()
```



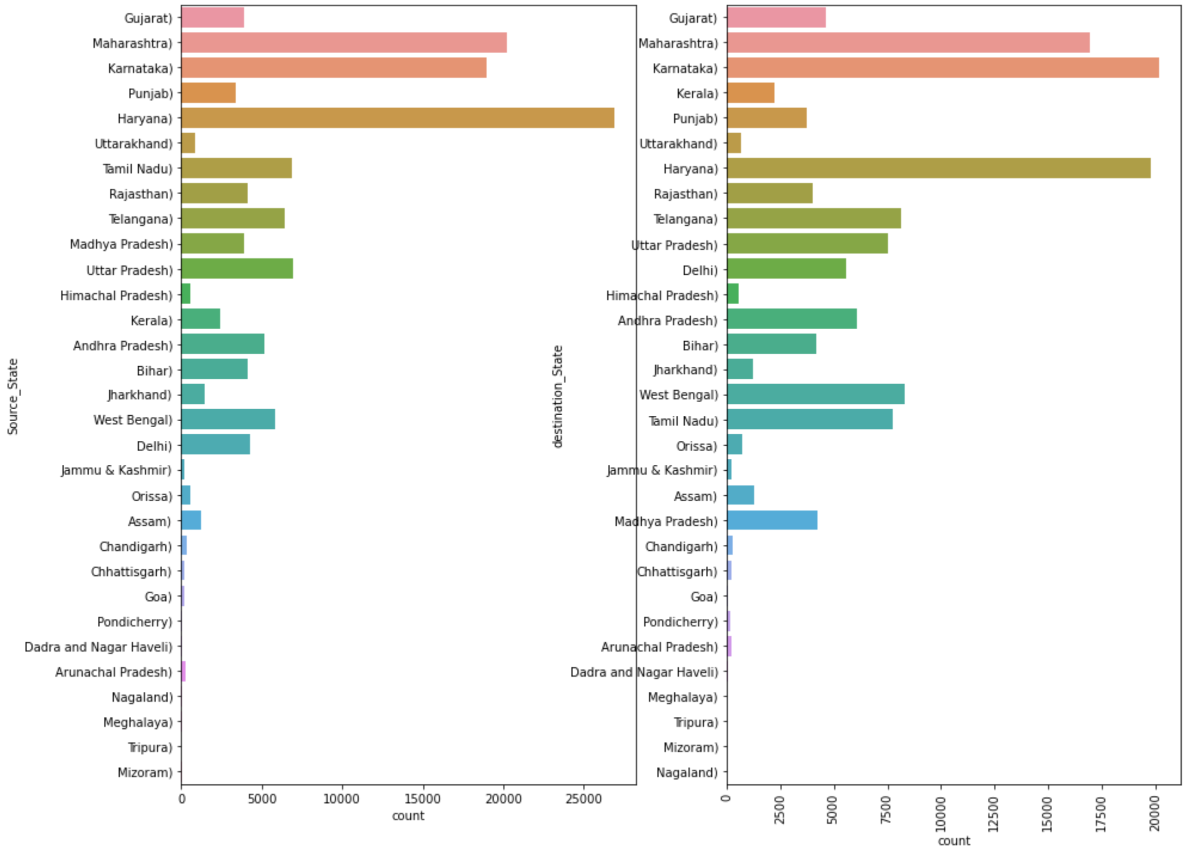
- 60% data is from Carting rout_type & the remaining 40% is from FTL

In [67]:
```
sns.countplot(y=df1['trip_creation_day'])
plt.xticks(rotation=90)
plt.show()
```



- Start & End dates of the months have less percent of trips compare to mid of the month. Though the difference is not huge
- Thats very strange to see that there is absolutely no trip from 4th- 11th day of the month

In [68]:
```
f,ax = plt.subplots(nrows=1,ncols=2,figsize=(15,12))
sns.countplot(y=d1['Source_State'],ax=ax[0])
sns.countplot(y=d1['destination_State'],ax=ax[1])

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




```
<Figure size 432x288 with 0 Axes>
```

- Most of the orders are coming from/reaching to Maharashtra followed by Karnataka
- Least orders are coming from/reaching to Seven Sisters(NE states)

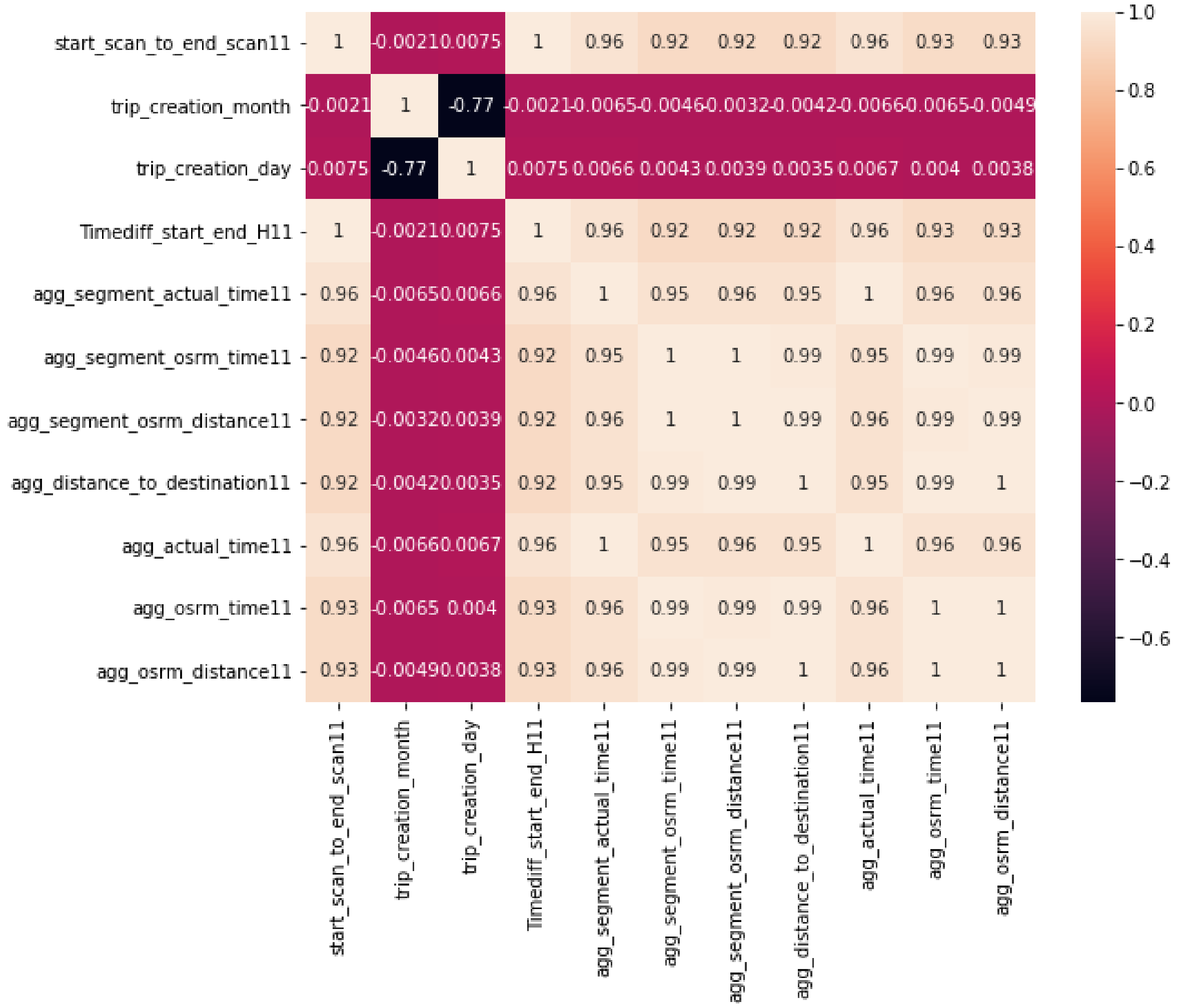
Bivariate Analysis

In [69]:

```
lk=df1.loc[:,['route_type', 'trip_uuid',
       'start_scan_to_end_scan11', 'trip_creation_month',
       'trip_creation_day', 'Timediff_start_end_H11', 'agg_segment_actual_time11',
       'agg_segment_osrm_time11', 'agg_segment_osrm_distance11',
       'agg_distance_to_destination11', 'agg_actual_time11', 'agg_osrm_time11',
       'agg_osrm_distance11', 'Source_City11', 'Source_Place11', 'Source_Code/Stat
       'destination_City11', 'destination_Place11', 'destination_Code/State11']]
```

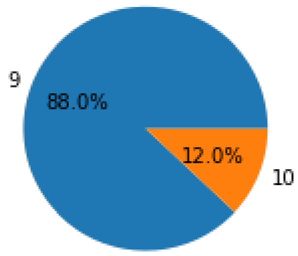
In [70]:

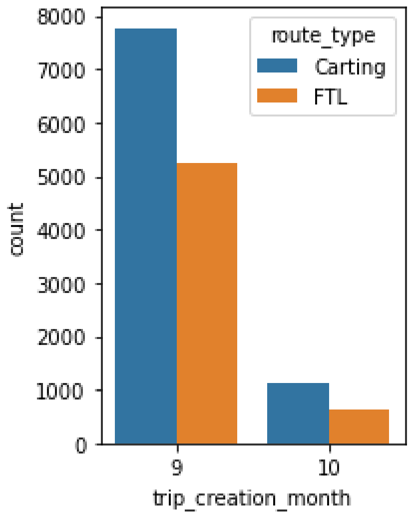
```
plt.figure(figsize = (9, 7))
sns.heatmap(lk.corr(),annot=True)
plt.show()
```



- Almost all the features are heavy positively correleated with each other & which is intutive as well.
- There is a high -ve correlation between almost all the other features & trip_creation_month

In [71]:
```
fig1 = plt.figure()
ax1 = fig1.add_subplot(121)
ax1.pie(df1['trip_creation_month'].value_counts(), labels=df1['trip_creation_month
plt.show()
fig2 = plt.figure()
ax2 = fig2.add_subplot(122)
sns.countplot(df1['trip_creation_month'], hue=df1['route_type'])
plt.show()
```





- 88% of the trips are from October Month & remaining are from November

One Hot Encoding of Categorical Features

In [72]:
```
#Oone hot endoing of rout_type columns:
mod_df=pd.concat([df1,pd.get_dummies(df1['route_type'])],axis=1)
```

In [73]:
```
#dropping the original column
mod_df.drop(['route_type'],axis=1, inplace=True)
```

In [74]:
```
mod_df.head(2)
```

Out[74]:

	trip_uuid	trip_creation_month	trip_creation_year	trip_creation_day	start_scan_to_er
0	trip-153741093647649320	9	2018	20	
10	trip-153768492602129387	9	2018	23	

2 rows × 22 columns

Dealing with high Cardinality in Categorical Feature

- Some of the features are having high cardinality like source_city, source_place etc:

A categorical feature is said to possess high cardinality when there are too many of these unique values. One-Hot Encoding becomes a big problem in such a case since we have a separate column for each unique value (indicating its presence or absence) in the categorical variable.

Reducing Cardinality by using a simple Aggregating function Below is a simple function I use to reduce the cardinality of a feature. The idea is very simple. Leave instances belonging to a value with high frequency as they are and replace the other instances with a new category which we will call other.

Choose a threshold Sort unique values in the column by their frequency in descending order Keep adding the frequency of these sorted (descending) unique values until a threshold is reached. These are the unique categories we will keep and instances of all other categories shall be replaced by “other”.

I am leaving that part for now as it is beyond the scope of this case study!

Column Standarization and Normalization

- Mean centering and Variance scaling (Standard Scaling)
- MinMax Scaling

In [75]:
```
st_df=df1.loc[:,['start_scan_to_end_scan11', 'trip_creation_month',
       'trip_creation_day', 'Timediff_start_end_H11', 'agg_segment_actual_time11',
       'agg_segment_osrm_time11', 'agg_segment_osrm_distance11',
       'agg_distance_to_destination11', 'agg_actual_time11', 'agg_osrm_time11',
       'agg_osrm_distance11']]
```

In [76]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
std_data = scaler.fit_transform(st_df)
std_data = pd.DataFrame(std_data, columns=st_df.columns)
std_data.head()
```

Out[76]:

	start_scan_to_end_scan11	trip_creation_month	trip_creation_day	Timediff_start_end_H11	agg_se
0	-0.508068	-0.369459	0.206152	-0.508710	
1	-0.345519	-0.369459	0.586769	-0.346216	
2	-0.640233	-0.369459	-0.555083	-0.639932	
3	0.761939	-0.369459	-0.681955	0.761412	
4	-0.407804	-0.369459	1.348004	-0.406557	

Insights:

Based on EDA:

- The entire data is heavily right skewed
- Almost all the features are heavy positively correleated with each other & which is intutive as well.
- There is a high -ve correlation between almost all the other features & trip_creation_month
- 60% data is from Carting rout_type & the remaining 40% is from FTL
- 88% of the trips are from October Month & remaining are from November
- Start & End dates of the months have less percent of trips compare to mid of the month. Though the difference is not huge

- Thats very strange to see that there is absolutely no trip from 4th- 11th day of the month
- Most of the orders are coming from Maharashtra followed by Karnataka
- Least orders are coming from Seven Sisters(NE states)
- Bussiest Corridor is from source Chandigarh_Mehmdpur_H (Punjab) to Bangalore_Nelmngla_H (Karnataka) Average_osrm_distance between them is 2500 kms & average time taken is 3784 mins

Based on Hypothesis Testing:

- There is no difference in the population means of time taken between od_start_time and od_end_time and start_scan_to_end_scan
- There is a difference between the population means of agg_Actual_time & agg_osrm_time. Mean of agg_Actual_time is bigger than that of agg_osrm_time
- There is a difference between the population means of agg_Actual_time & agg_segment_actual_time. Mean of agg_Actual_time is bigger than that of agg_segment_actual_time
- There is a difference between the population means of agg_osrm_distance & agg_segment_osrm_distance.
- There is a difference between the population means of osrm_time & segment_osrm_time.

Recommendations:

- As its depicted from the analysis that there is absolutely no trip from 4th- 11th day of the month, The reason for that can be figured out and catered to receive the orders in the these dates as well.
- Least orders are reaching/coming from North East states, more corridors/campaigns can be promoted to penetrate in these states as well.
- Most of the orders are coming from/reaching to Maharashtra & Karnataka, existing corridors can be further enhanced to maintain the penetration from these areas.Also a further profiling of these orders can be made from more rich data to get to know why major orders are coming from these corridors
- FTL route consists of 40% of the total orders. More ways to promote FTL route handling system can be implemented to increase this percentage