In [ ]:

```
Problem statement- To clean and analyse the raw data from Delhivery to build forecasting mo

Note: All comments are provided in each cell in the notebook

 Insights/Cleaning

1.    Extracted pincode for soured and destination. Source city and destination city. Sourc

2.    Missing\ null values are filled with the the correct source/destination places and ‘U

3.     Extracted source/destination names and made a separate column (Regex)

4.    Grouped data based on mean, max, min etc. [Cell 20]

5.    Extracted year, month and day for all the date time columns

6.    There are 3 times more training data than test data

7.    Route_type ‘FTL’ is more than ‘Carting’

8.    Number of orders at the beginning and end of month are less when compared with the mi

9.    Gurgaon_Bilaspur,Bhiwandi_Mankoli and Bangalore_Nelmngla are the top 3 source city [T

10.  Gurgaon_Bilaspur,Bhiwandi_Mankoli and Bangalore_Nelmngla are the top 3 destination cit

11. Karnataka, Maharashtra and tamil Nadu are the top 3 destination state [Top 30 shown in 

12. Boxplot plotted for ‘actual_time_max’,’osrm_time_max’,
’actual_distance_to_destination’ for source and destinations states. There are more than 80

13. Distribution plot for all the numerical variables are
plotted and all the plots are right skewed

14. Bottom 10 source state are Chhattisgarh, Arunachal Pradesh
and Jammu Kashmir, etc [Cell 52]

15.  Bottom 10 source city are  Allahabad_Mirapati, Munbai Chndivli and Hyb_LB nagar, etc [

16. Top 20 source to destination city are Bangalore_Nelmngla_H to Bengaluru_KGAirprt_HB ,

17. Top orders are mostly with in the state

18.[Cell 65] Summarizes the top 20 source to destination with their actual_time, osrm_time,

19.Ouliners are found and were not removed as more that 80% values are outliners

20. Binning for all numerical coulmns are done. For all the time bins most of the values li

21. Found the average by dividing the time and distance by count

22.Pearson, Spearman and Kendall coefficients  were found. Time and distance variables have

23.Pair plot for all the numerical variables are shown in cell 85. Time and distance variab



 Hypothesis and visual analysis was done on all the numerical variables (With stating assum
```

```
1.     Normality Tests : QQplot, Shapiro-Wilk test and Kolmogorov-Smirnov test

2.     Tranformation : Log, Box-Cox, Reciprocal and Square root

3.     Correlation Tests : Pearson, Spearman and kendall rank

4.     Variance Test : Bartlett and Levene

5.     Tests when Data is not normal : Mann-Whitney U test and Wilcoxon signed-rank test

6.     Test when data is normal : T Test (equal and unequal variances), AVOVA

7.     Independence test : Chi2 test

8.     Kruskal-Wallis one-way analysis of variance


24. None of the numerical variables are normal distribution even after applying transformat

25. 'Start_scan_to_end_scan_max','od_delta' variables have equal variances and medians

26.ctual_time aggregated value and OSRM time aggregated value have unequal variances and su

27. Actual_time aggregated value and segment actual time aggregated value have unequal vari

28. Osrm distance aggregated value and segment osrm distance aggregated value have unequal

29.  osrm time aggregated value and segment osrm time aggregated value have unequal varianc

30. Time and distance variables are independent with the route_type (Chi2 test)

31.  Kruskal-Wallis one way analysis of variance test was done for 'osrm_time_max','segment

32.  Kruskal-Wallis one way analysis of variance test was done for 'segment_osrm_distance_s

33. Minmax scaling was done for all numerical variables [Cell 154]

34.  Onehot encoding was done for ‘route_type’ [Cell 147]


Recommendations:

1.    In most of the cases 'segment_actual_time' is greater than 'segment_osrm_time'. We wi

2.    Make more FTL shipments than Carting as they are faster

3.    Reduce the pricing of the of services at the beginning and end of every month.

4.    Increase the pricing in city where there are more number of orders. Ex-Bangalore, Gur

5.    Decrease the price and delivery time for places there are very few orders. Ex- Allaha

6.    Have more FTL shipments in smaller trucks by which we can reduce the delivery time

7.    Time and distance variables have a positive corelation. Have more smaller warehouses

8.     One hot encoding and minmax scaling is dozen that can be used by the data science te

9.    ‘segment_actual_time ’ needs to be reduced thereby reducing the total time.
```

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from  scipy import stats
from statsmodels.stats.weightstats import ztest as ztest
from statsmodels.formula.api import ols
import statsmodels.api as sm
import pingouin as pg
import datetime
import math
import scipy
import warnings
warnings.filterwarnings('ignore')
import re
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```
df=pd.read_csv('delhivery_data.csv')
```

In [3]:

```
df.info()
# Describes data type and non null values in each column
```

```
<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 [4]:

```
df.shape
# Describes the shape of the date
```

Out[4]:

```
(144867, 24)
```

In [5]:

```
df.describe()
# Summarises the numerical data type for all columns
```

Out[5]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	
count	144867.000000	144867.000000	144867.000000	144867.000000	14
mean	961.262986	232.926567	234.073372	416.927527	
std	1037.012769	344.755577	344.990009	598.103621	
min	20.000000	9.000000	9.000045	9.000000	
25%	161.000000	22.000000	23.355874	51.000000	
50%	449.000000	66.000000	66.126571	132.000000	
75%	1634.000000	286.000000	286.708875	513.000000	
max	7898.000000	1927.000000	1927.447705	4532.000000	

In [6]:

```
df.describe(include='object')
# Summarises the object data type for all columns
```

Out[6]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sc
count	144867	144867	144867	144867	144867	
unique	2	14817	1504	2	14817	
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153811219535896559	INI
freq	104858	101	1812	99660	101	

In [7]:

```
df.columns
# All column names
```

Out[7]:

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

In [8]:

```
df.head(10)
```

Out[8]:

destination_name	od_start_time	...	cutoff_timestamp	actual_distance_to_destination	actu
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:27:55	10.435660	
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:17:55	18.936842	
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:01:19.505586	27.637279	
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:39:57	36.118028	
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:33:55	39.386040	
Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797	...	2018-09-20 06:15:58	10.403038	
Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797	...	2018-09-20 05:47:29	18.045481	
Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797	...	2018-09-20 05:25:58	28.061896	
Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797	...	2018-09-20 05:15:56	38.939167	
Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797	...	2018-09-20 04:49:20	43.595802	

In [9]:

```
df['source_center_pincode']=[i[3:9] for i in df['source_center']]
df['destination_center_pincode']=[i[3:9] for i in df['destination_center']]
# Extracting PINCODE for source and destinations
```

In [10]:

```
df.isnull().sum()
# Finding null values for each column
# Source_name and Destination_ name have null/missing values
```

Out[10]:

```
data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                     293
destination_center                0
destination_name                261
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
is_cutoff                         0
cutoff_factor                     0
cutoff_timestamp                  0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
factor                            0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
segment_factor                    0
source_center_pincode             0
destination_center_pincode        0
dtype: int64
```

In [11]:

```
dff=df.copy('deep')
dff.dropna(axis=0,how='any',inplace=True)
dff=dff.reset_index()
newdict={}
for i in range(len(dff['source_center'])):
    if dff['source_center'][i][3:9] not in newdict:
        newdict[dff['source_center'][i][3:9]]=dff['source_name'][i]
for i in range(len(dff['destination_center'])):
    if dff['destination_center'][i][3:9] not in newdict:
        newdict[dff['destination_center'][i][3:9]]=dff['destination_name'][i]
```

In [12]:

```
for i in range(len(df)):
    if len(str(df['source_name'][i]))<4:
        if df['source_center'][i][3:9] in newdict:
            df['source_name'][i]=newdict[df['source_center'][i][3:9]]
    if len(str(df['destination_name'][i]))<4:
        if df['destination_center'][i][3:9] in newdict:
            df['destination_name'][i]=newdict[df['destination_center'][i][3:9]]

# I have compared the pincode for the existing data types with the missing values and fille
#correct source and destination names
```

In [13]:

```
df.isnull().sum()
# We have 172 and 198  missing values
```

Out[13]:

```
data                               0
trip_creation_time                 0
route_schedule_uuid                0
route_type                         0
trip_uuid                          0
source_center                      0
source_name                      172
destination_center                 0
destination_name                 198
od_start_time                      0
od_end_time                        0
start_scan_to_end_scan             0
is_cutoff                          0
cutoff_factor                      0
cutoff_timestamp                   0
actual_distance_to_destination     0
actual_time                        0
osrm_time                          0
osrm_distance                      0
factor                             0
segment_actual_time                0
segment_osrm_time                  0
segment_osrm_distance              0
segment_factor                     0
source_center_pincode              0
destination_center_pincode         0
dtype: int64
```

In [14]:

```
df['source_name'] = df['source_name'].replace(np.nan, 'Unknown_Source_City (Unknown_Source_
df['destination_name'] = df['destination_name'].replace(np.nan, 'Unknown_Destination_City (
# Replacing the missing values with 'Unknown'
```

In [15]:

```
df.isnull().sum()
# There are no missing values
```

Out[15]:

```
data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                       0
destination_center                0
destination_name                  0
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
is_cutoff                         0
cutoff_factor                     0
cutoff_timestamp                  0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
factor                            0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
segment_factor                    0
source_center_pincode             0
destination_center_pincode        0
dtype: int64
```

In [16]:

```
source_df=pd.DataFrame(df['source_name'].apply(lambda x: str(x).split(' ')))
destination_df=pd.DataFrame(df['destination_name'].apply(lambda x: str(x).split(' ')))
source_df=pd.DataFrame(source_df['source_name'].to_list())
destination_df=pd.DataFrame(destination_df['destination_name'].to_list())
df['Source_City']=source_df[0]
df['Destination_City']=destination_df[0]
# Extracting the source and destination city
```

In [17]:

```
import re
def stt(s):
    result = re.findall('\(.*?\)', s)
    return str(result)
# Used Regex to extract the source and destination states
```

In [18]:

```
Source_State=pd.DataFrame(df['source_name'].apply(stt))
df['Source_State']=[i[3:-3] for i in Source_State['source_name']]
Destination_State=pd.DataFrame(df['destination_name'].apply(stt))
df['Destination_State']=[i[3:-3] for i in Destination_State['destination_name']]
# Used Regex to extract the source and destination states
```

In [19]:

```
df.drop('route_schedule_uuid',axis=1,inplace=True)
# Dropped 'route_schedule_uuid' as it is not needed
```

In [20]:

```
# Grouped the Data
data=df.groupby(['data','trip_creation_time','trip_uuid','route_type','source_center','sour
['actual_time','osrm_time','osrm_distance','start_scan_to_end_scan','actual_distance_to_des
aggregate({'actual_time':['max','count'],
           'osrm_time':max,
           'osrm_distance':max,
           'start_scan_to_end_scan':max,
           'actual_distance_to_destination':['max','count'],
           'segment_actual_time':['sum','count'],
           'segment_osrm_time':['sum','count'],
           'segment_osrm_distance':['sum','count'],
           'cutoff_factor':['min','max','mean'],
           'segment_factor':['min','max','mean'],
          'factor':['min','max','mean']}).reset_index()
# 'actual_time' is a cumulative column so used the max
# 'osrm_distance' is a cumulative column so used the max
# 'start_scan_to_end_scan' is a cumulative column so used the max
# 'actual_distance_to_destination','segment_actual_time','segment_osrm_time','segment_osrm_
# is same of considered the max
# 'cutoff_factor','segment_factor' and 'factor' are unknown hence taken min,max and mean
```

In [21]:

```
data.columns = ['data','trip_creation_time','trip_uuid','route_type','source_center','sourc
                     'destination_center','destination_name','od_start_time','od_end_time',
                     'destination_center_pincode','Source_City','Source_State','Destination
                     'actual_time_max','actual_time_count','osrm_time_max','osrm_distance_m
                     'actual_distance_to_destination_max','actual_distance_to_destination_c
                     'segment_actual_time_count','segment_osrm_time_sum','segment_osrm_time
                     'segment_osrm_distance_count','cutoff_factor_min','cutoff_factor_max',
                     'segment_factor_min','segment_factor_max','segment_factor_mean','facto
# Renamed all the columns names
```

In [22]:

```
data.drop(['source_center','destination_center','destination_name','source_name'],axis=1,in
# Dropped above columns
```

In [23]:

```
data.columns
```

Out[23]:

```
Index(['data', 'trip_creation_time', 'trip_uuid', 'route_type',
       'od_start_time', 'od_end_time', 'source_center_pincode',
       'destination_center_pincode', 'Source_City', 'Source_State',
       'Destination_City', 'Destination_State', 'actual_time_max',
       'actual_time_count', 'osrm_time_max', 'osrm_distance_max',
       'start_scan_to_end_scan_max', 'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean'],
      dtype='object')
```

In [24]:

```
data['source_center_pincode'].replace('68004A','680004',inplace=True)
data['destination_center_pincode'].replace('68004A','680004',inplace=True)
# This '680004A' pincode is changed to '680004' THRISSUR
```

In [25]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26369 entries, 0 to 26368
Data columns (total 34 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   data                                26369 non-null  object 
 1   trip_creation_time                  26369 non-null  object 
 2   trip_uuid                           26369 non-null  object 
 3   route_type                          26369 non-null  object 
 4   od_start_time                       26369 non-null  object 
 5   od_end_time                         26369 non-null  object 
 6   source_center_pincode               26369 non-null  object 
 7   destination_center_pincode          26369 non-null  object 
 8   Source_City                         26369 non-null  object 
 9   Source_State                        26369 non-null  object 
 10  Destination_City                    26369 non-null  object 
 11  Destination_State                   26369 non-null  object 
 12  actual_time_max                     26369 non-null  float64
 13  actual_time_count                   26369 non-null  int64  
 14  osrm_time_max                       26369 non-null  float64
 15  osrm_distance_max                   26369 non-null  float64
 16  start_scan_to_end_scan_max          26369 non-null  float64
 17  actual_distance_to_destination_max  26369 non-null  float64
 18  actual_distance_to_destination_count  26369 non-null  int64  
 19  segment_actual_time_sum             26369 non-null  float64
 20  segment_actual_time_count           26369 non-null  int64  
 21  segment_osrm_time_sum               26369 non-null  float64
 22  segment_osrm_time_count             26369 non-null  int64  
 23  segment_osrm_distance_sum           26369 non-null  float64
 24  segment_osrm_distance_count         26369 non-null  int64  
 25  cutoff_factor_min                   26369 non-null  int64  
 26  cutoff_factor_max                   26369 non-null  int64  
 27  cutoff_factor_mean                  26369 non-null  float64
 28  segment_factor_min                  26369 non-null  float64
 29  segment_factor_max                  26369 non-null  float64
 30  segment_factor_mean                 26369 non-null  float64
 31  factor_min                          26369 non-null  float64
 32  factor_max                          26369 non-null  float64
 33  factor_mean                         26369 non-null  float64
dtypes: float64(15), int64(7), object(12)
memory usage: 6.8+ MB
```

In [26]:

```
data['trip_creation_year']=pd.to_datetime(data['trip_creation_time']).dt.year
data['trip_creation_month']=pd.to_datetime(data['trip_creation_time']).dt.month
data['trip_creation_day']=pd.to_datetime(data['trip_creation_time']).dt.day
# Converting dat time to year, month and day
```

In [27]:

```
data['od_start_year']=pd.to_datetime(data['od_start_time']).dt.year
data['od_start_month']=pd.to_datetime(data['od_start_time']).dt.month
data['od_start_day']=pd.to_datetime(data['od_start_time']).dt.day
# Converting dat time to year, month and day
```

In [28]:

```
data['od_end_year']=pd.to_datetime(data['od_end_time']).dt.year
data['od_end_month']=pd.to_datetime(data['od_end_time']).dt.month
data['od_end_day']=pd.to_datetime(data['od_end_time']).dt.day
# Converting dat time to year, month and days
```

In [29]:

```
data.columns
```

Out[29]:

```
Index(['data', 'trip_creation_time', 'trip_uuid', 'route_type',
       'od_start_time', 'od_end_time', 'source_center_pincode',
       'destination_center_pincode', 'Source_City', 'Source_State',
       'Destination_City', 'Destination_State', 'actual_time_max',
       'actual_time_count', 'osrm_time_max', 'osrm_distance_max',
       'start_scan_to_end_scan_max', 'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean',
       'trip_creation_year', 'trip_creation_month', 'trip_creation_day',
       'od_start_year', 'od_start_month', 'od_start_day', 'od_end_year',
       'od_end_month', 'od_end_day'],
      dtype='object')
```

In [30]:

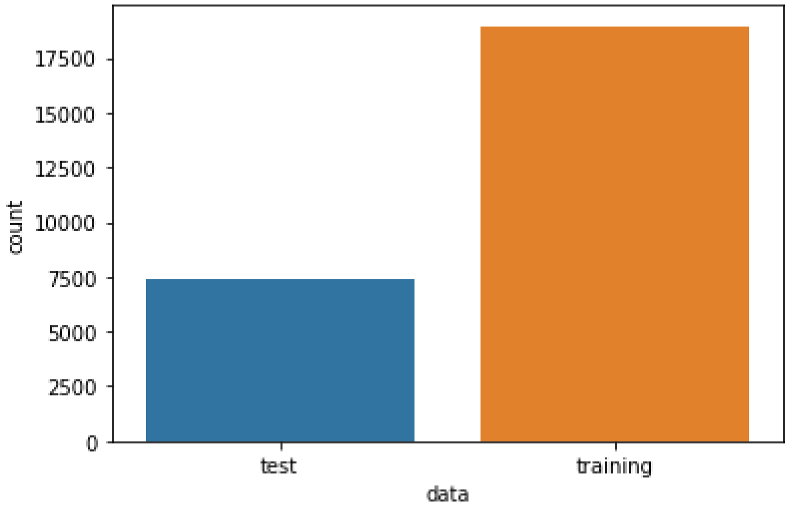
```
data['od_delta']=(pd.to_datetime(data['od_end_time'])-pd.to_datetime(data['od_start_time'])
# Finding the 'od_delta' by subtracting 'od_end_time' and 'od_start_time'
```

In [31]:

```
sns.countplot(data['data'])
# Below plot shows the count of training and test data
# close to 17750 data for training and 7500 for test data
```

Out[31]:

```
<AxesSubplot:xlabel='data', ylabel='count'>
```

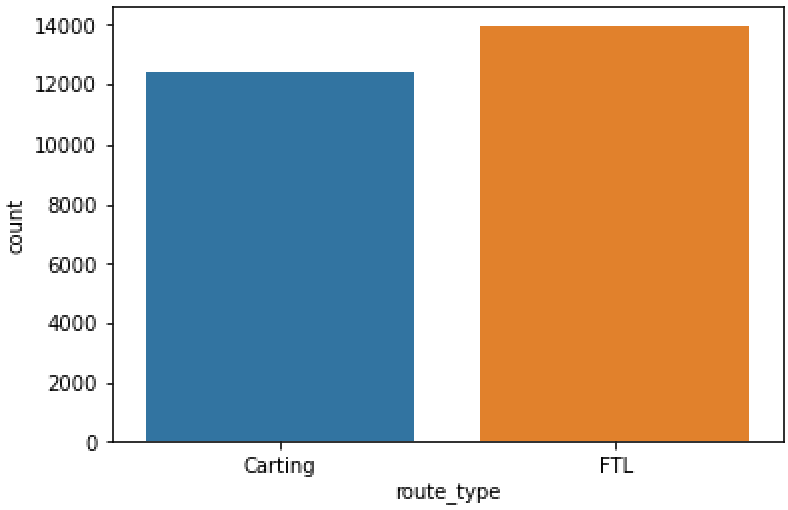


In [32]:

```
sns.countplot(data['route_type'])
# Below plot shows the count of route_type
# close to 14000 data for FTL and 12000 for carting data
```

Out[32]:

```
<AxesSubplot:xlabel='route_type', ylabel='count'>
```

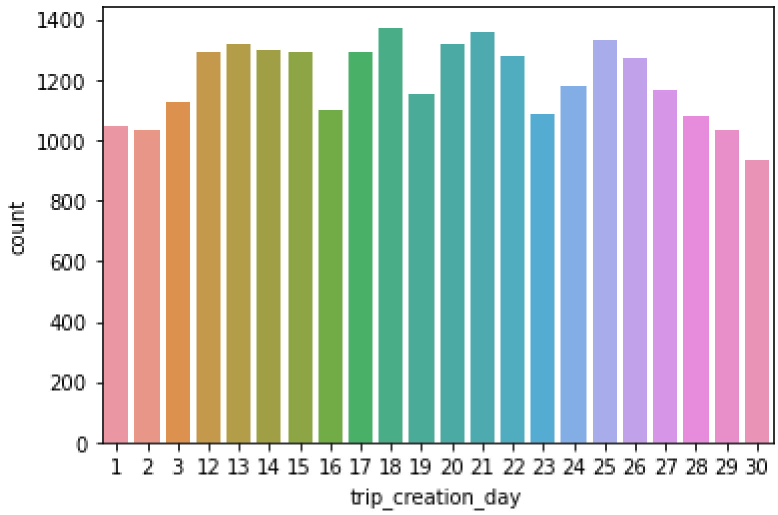


In [33]:

```
sns.countplot(data['trip_creation_day'])
# Count for each day in a month
# Count od orders at the beginning and end of the month are less
```

Out[33]:

<AxesSubplot:xlabel='trip_creation_day', ylabel='count'>



In [34]:

```
data.info()
```

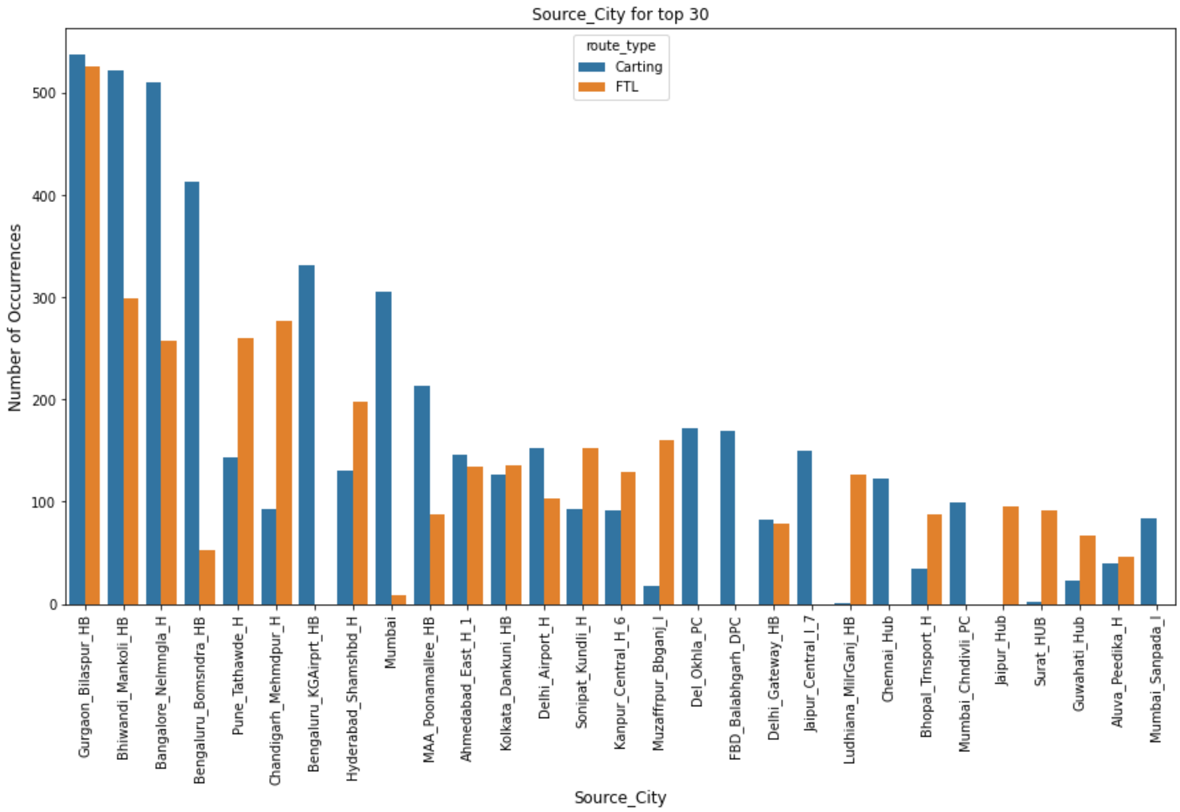
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26369 entries, 0 to 26368
Data columns (total 44 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   data                                 26369 non-null  object 
 1   trip_creation_time                   26369 non-null  object 
 2   trip_uuid                            26369 non-null  object 
 3   route_type                           26369 non-null  object 
 4   od_start_time                        26369 non-null  object 
 5   od_end_time                          26369 non-null  object 
 6   source_center_pincode                26369 non-null  object 
 7   destination_center_pincode           26369 non-null  object 
 8   Source_City                          26369 non-null  object 
 9   Source_State                         26369 non-null  object 
 10  Destination_City                     26369 non-null  object 
 11  Destination_State                    26369 non-null  object 
 12  actual_time_max                      26369 non-null  float64
 13  actual_time_count                    26369 non-null  int64  
 14  osrm_time_max                        26369 non-null  float64
 15  osrm_distance_max                    26369 non-null  float64
 16  start_scan_to_end_scan_max           26369 non-null  float64
 17  actual_distance_to_destination_max   26369 non-null  float64
 18  actual_distance_to_destination_count 26369 non-null  int64  
 19  segment_actual_time_sum              26369 non-null  float64
 20  segment_actual_time_count            26369 non-null  int64  
 21  segment_osrm_time_sum                26369 non-null  float64
 22  segment_osrm_time_count              26369 non-null  int64  
 23  segment_osrm_distance_sum            26369 non-null  float64
 24  segment_osrm_distance_count          26369 non-null  int64  
 25  cutoff_factor_min                    26369 non-null  int64  
 26  cutoff_factor_max                    26369 non-null  int64  
 27  cutoff_factor_mean                   26369 non-null  float64
 28  segment_factor_min                   26369 non-null  float64
 29  segment_factor_max                   26369 non-null  float64
 30  segment_factor_mean                  26369 non-null  float64
 31  factor_min                           26369 non-null  float64
 32  factor_max                           26369 non-null  float64
 33  factor_mean                          26369 non-null  float64
 34  trip_creation_year                   26369 non-null  int64  
 35  trip_creation_month                  26369 non-null  int64  
 36  trip_creation_day                    26369 non-null  int64  
 37  od_start_year                        26369 non-null  int64  
 38  od_start_month                       26369 non-null  int64  
 39  od_start_day                         26369 non-null  int64  
 40  od_end_year                          26369 non-null  int64  
 41  od_end_month                         26369 non-null  int64  
 42  od_end_day                           26369 non-null  int64  
 43  od_delta                             26369 non-null  float64
dtypes: float64(16), int64(16), object(12)
memory usage: 8.9+ MB
```

In [35]:

```
def outliners(x,col):
    Q1 = np.percentile(x[col], 25)
    Q3 = np.percentile(x[col], 75)
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR
    lower = Q1 - 1.5*IQR
    #print(upper,lower)
    ls=list(x.iloc[((x[col]<lower) | (x[col]>upper)).values].index)
    return ls
list_out=['actual_time_max','osrm_time_max','osrm_distance_max','start_scan_to_end_scan_max
         'segment_actual_time_sum','segment_osrm_time_sum','segment_osrm_distance_sum']
# Definition to check outliners
```

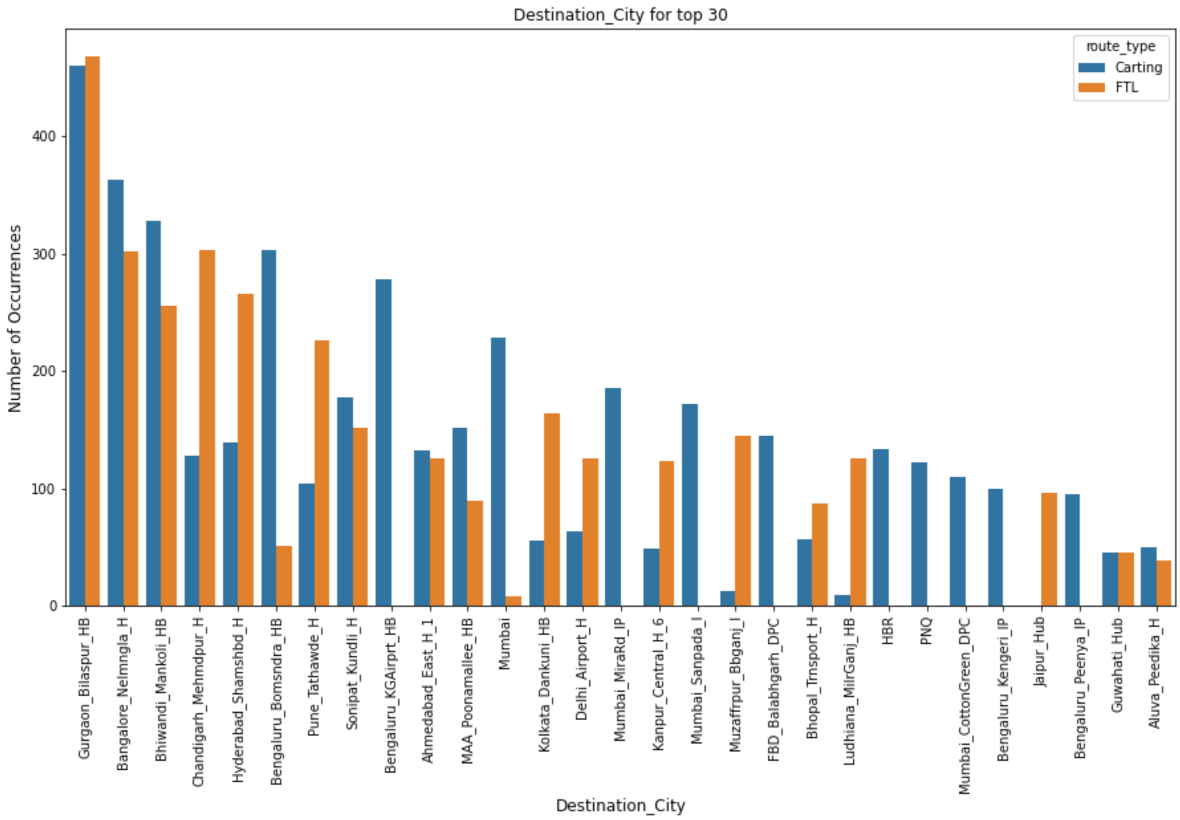
In [36]:

```
sns.countplot(data['Source_City'],order=pd.value_counts(data['Source_City']).iloc[:29].inde
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
plt.title('Source_City for top 30')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Source_City', fontsize=12)
plt.show()
# Bar plot below shows the top 30 source city which has maximun shipments with hue as 'rout
```



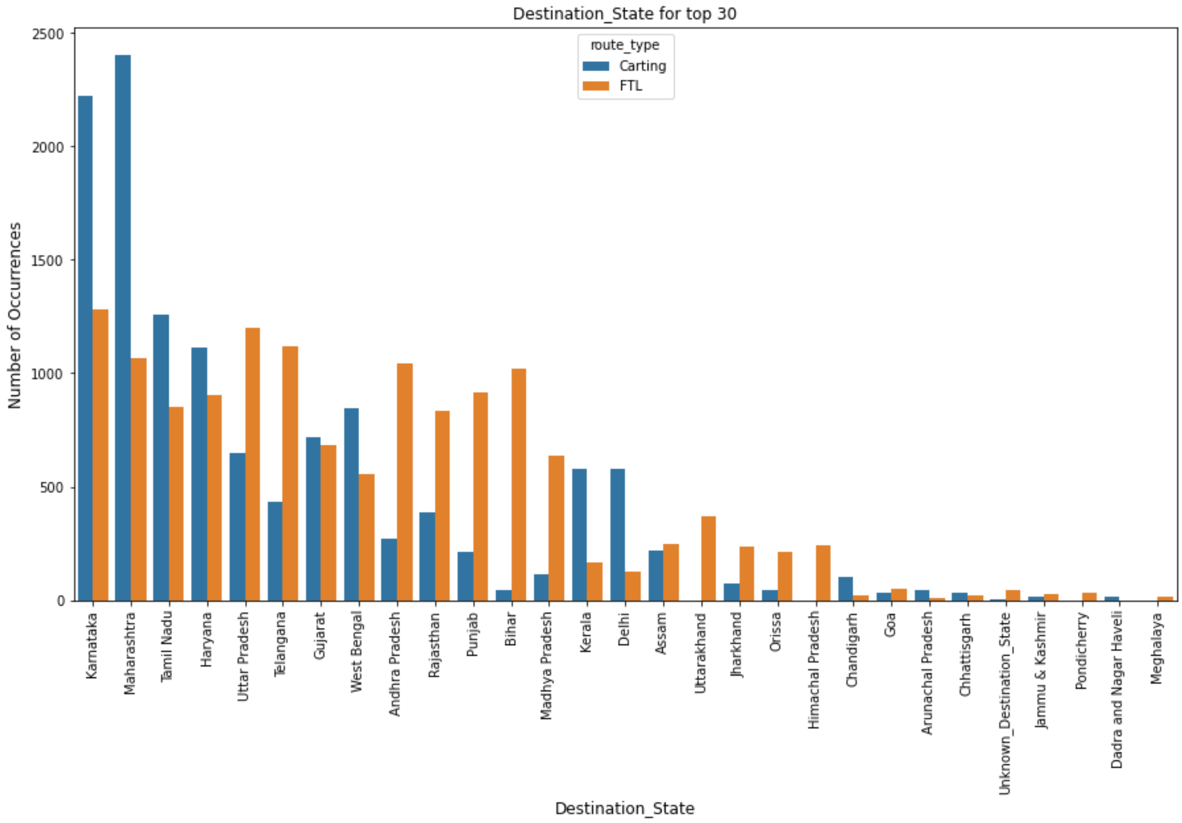
In [37]:

```
sns.countplot(data['Destination_City'],order=pd.value_counts(data['Destination_City']).iloc
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
plt.title('Destination_City for top 30')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Destination_City', fontsize=12)
plt.show()
# Bar plot below shows the top 30 Destination_City which has maximun shipments with hue as
```



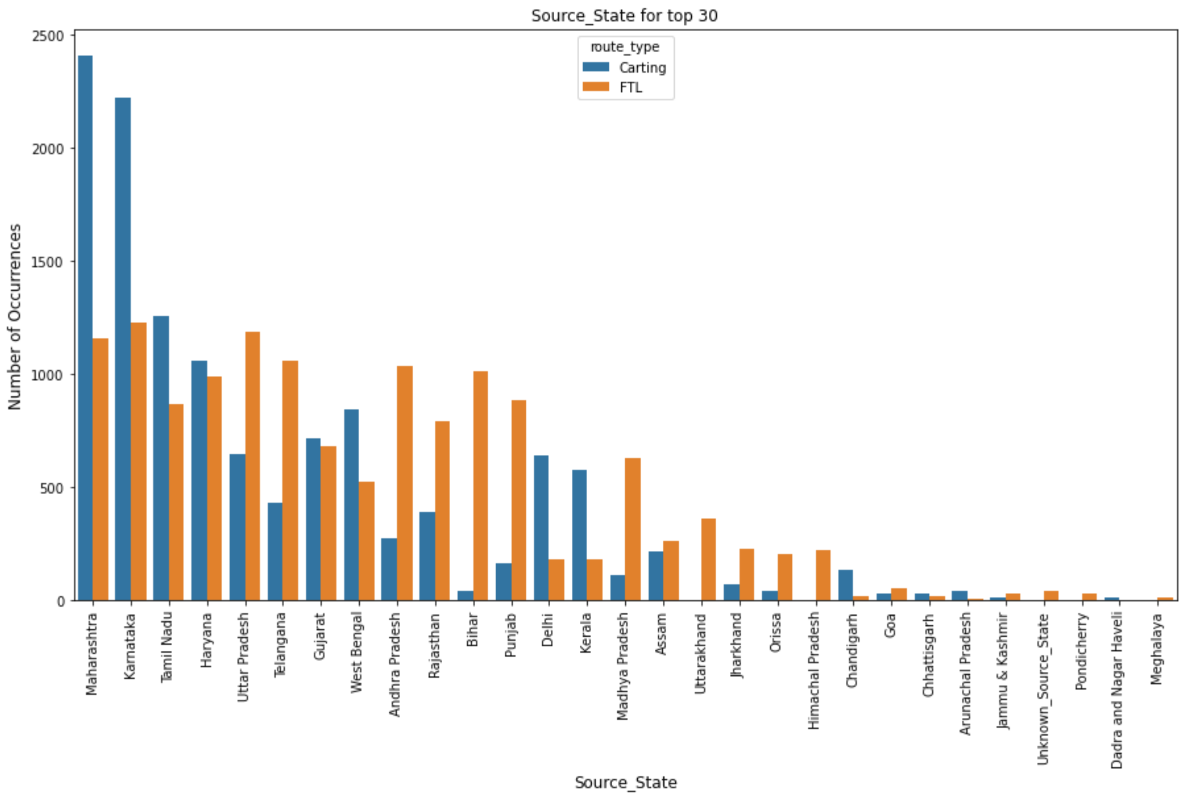
In [38]:

```
sns.countplot(data['Destination_State'],order=pd.value_counts(data['Destination_State']).il
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
plt.title('Destination_State for top 30')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Destination_State', fontsize=12)
plt.show()
# Bar plot below shows the top 30 Destination_State which has maximun shipments with hue as
```



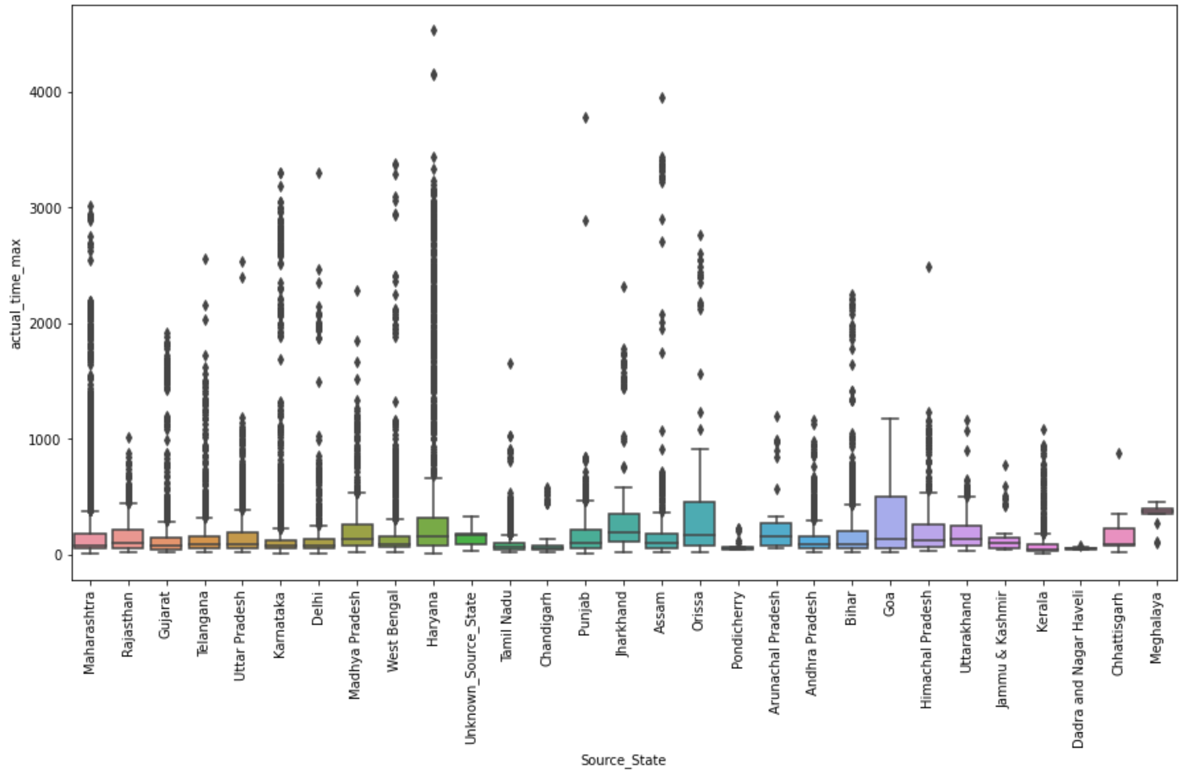
In [39]:

```
sns.countplot(data['Source_State'],order=pd.value_counts(data['Source_State']).iloc[:29].in
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
plt.title('Source_State for top 30')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Source_State', fontsize=12)
plt.show()
# Bar plot below shows the top 30 Source_State which has maximun shipments with hue as 'rou
```



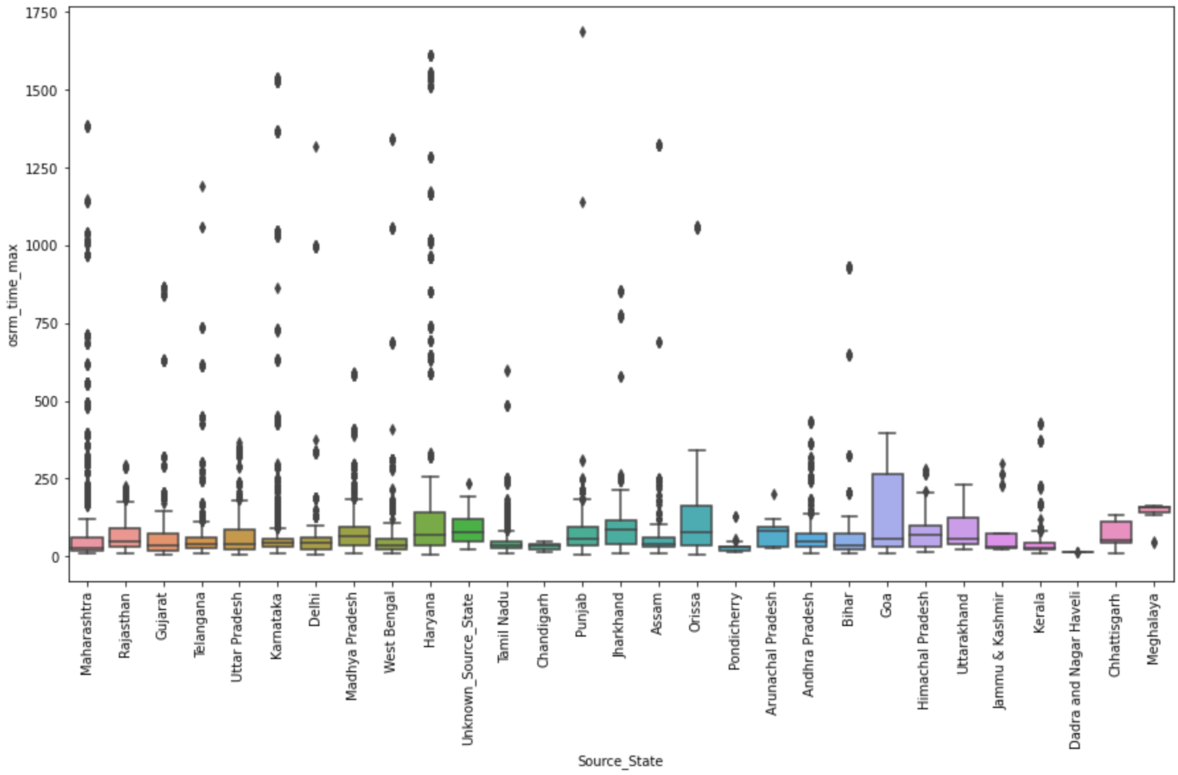
In [40]:

```
sta=list(data['Source_State'].value_counts().index[:29])
datanew=data[data['Source_State'].isin(sta)]
sns.boxplot(data=datanew, x='Source_State', y='actual_time_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 source states for 'actual_time_max' variable
# There are lot of outliners for every variable
```



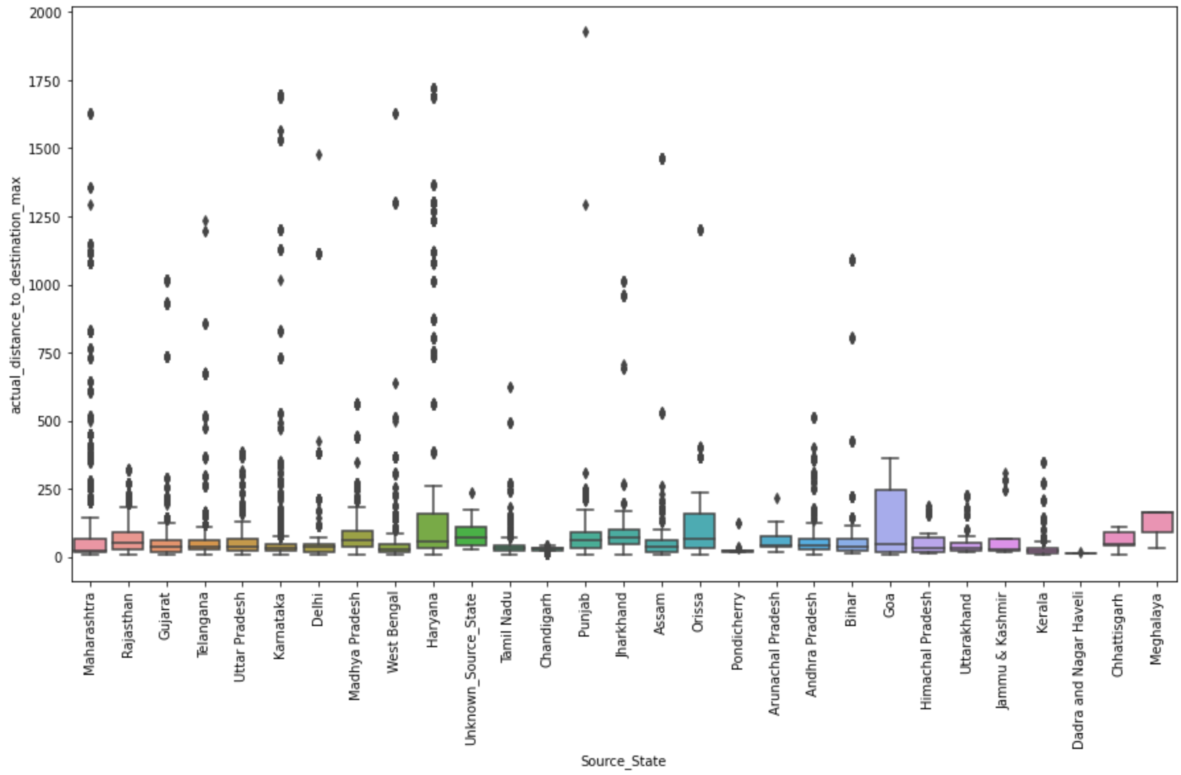
In [41]:

```
sta=list(data['Source_State'].value_counts().index[:29])
datanew=data[data['Source_State'].isin(sta)]
sns.boxplot(data=datanew, x='Source_State', y='osrm_time_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 source states for 'osrm_time_max' variable
# There are lot of outliners for every variable
```



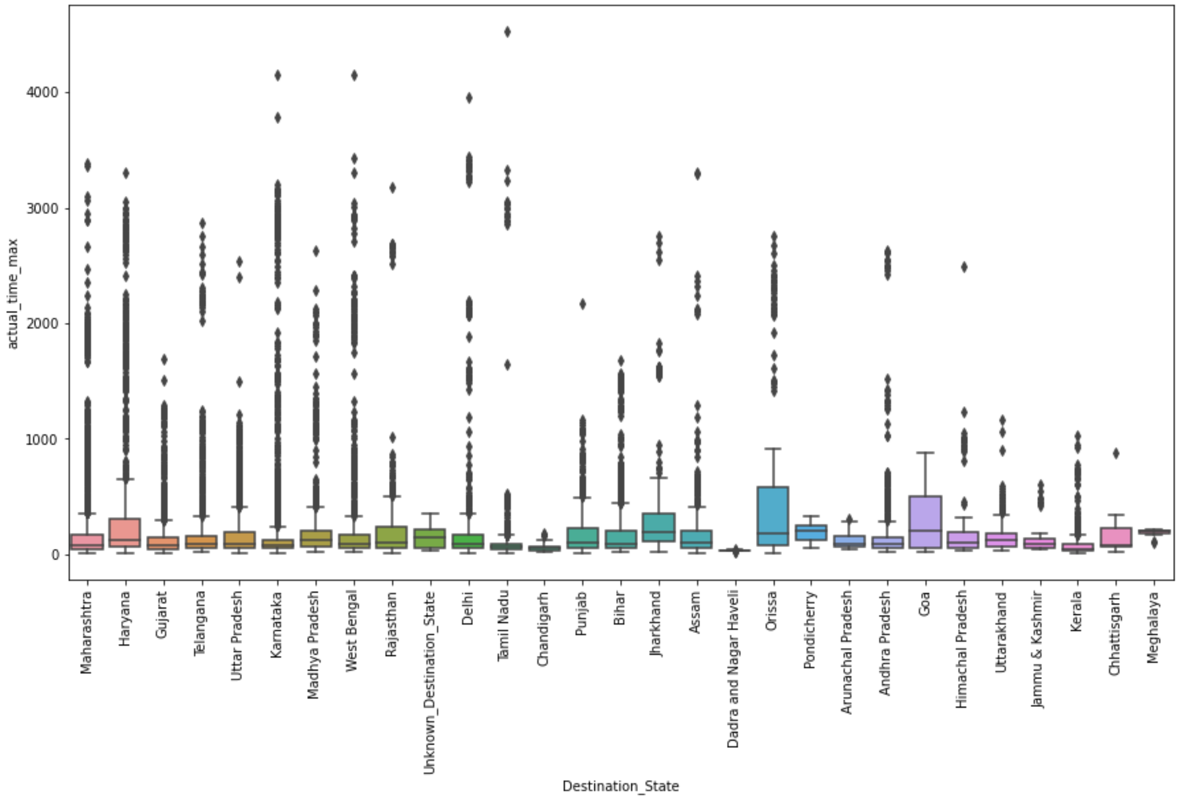
In [42]:

```
sta=list(data['Source_State'].value_counts().index[:29])
datanew=data[data['Source_State'].isin(sta)]
sns.boxplot(data=datanew, x='Source_State', y='actual_distance_to_destination_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 source states for 'actual_distance_to_destination_max' variable
# There are lot of outliners for every variable
```



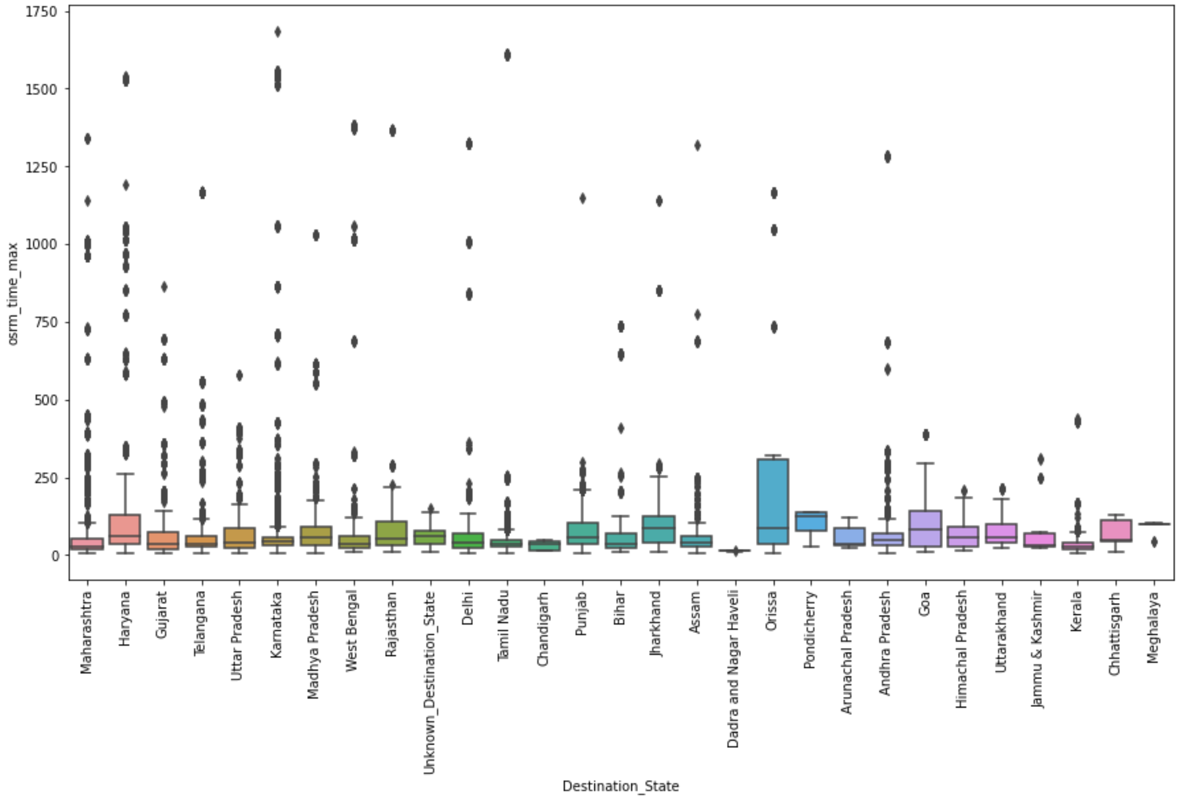
In [43]:

```
sta=list(data['Destination_State'].value_counts().index[:29])
datanew=data[data['Destination_State'].isin(sta)]
sns.boxplot(data=datanew, x='Destination_State', y='actual_time_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 Destination_State for 'actual_time_max' variable
# There are lot of outliners for every variable
```



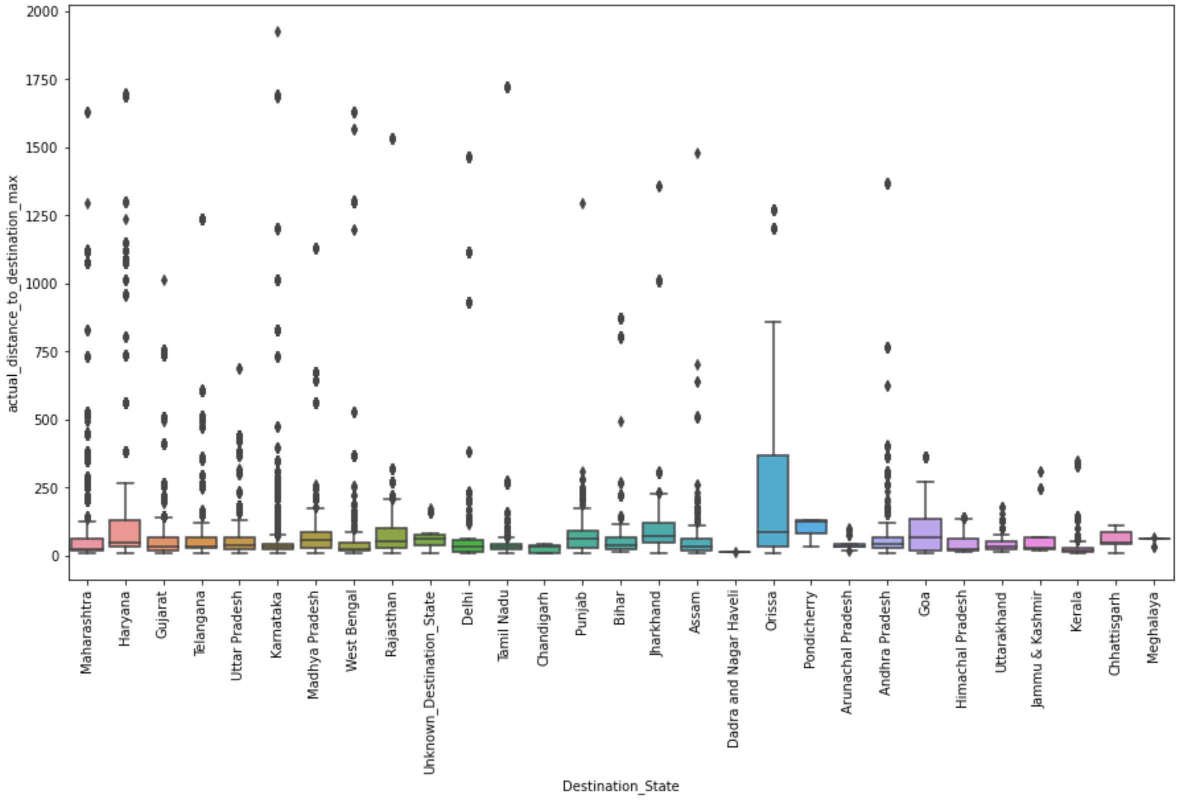
In [44]:

```
sta=list(data['Destination_State'].value_counts().index[:29])
datanew=data[data['Destination_State'].isin(sta)]
sns.boxplot(data=datanew, x='Destination_State', y='osrm_time_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 Destination_State for 'osrm_time_max' variable
# There are lot of outliners for every variable
```



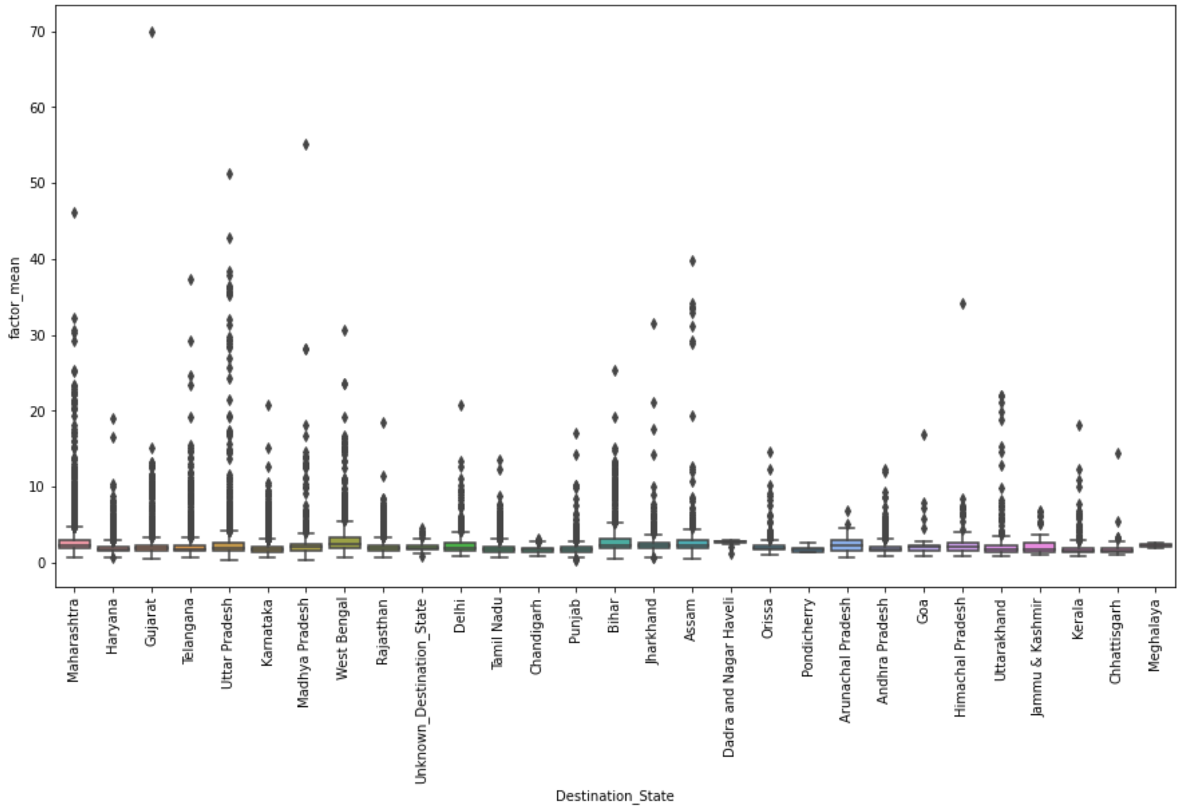
In [45]:

```
sta=list(data['Destination_State'].value_counts().index[:29])
datanew=data[data['Destination_State'].isin(sta)]
sns.boxplot(data=datanew, x='Destination_State', y='actual_distance_to_destination_max')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 Destination_State for 'actual_distance_to_destination_max' variable
# There are lot of outliners for every variable
```



In [46]:

```
sta=list(data['Destination_State'].value_counts().index[:29])
datanew=data[data['Destination_State'].isin(sta)]
sns.boxplot(data=datanew, x='Destination_State', y='factor_mean')
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Boxplot for top 30 Destination_State for 'factor_mean' variable
# There are lot of outliners for every variable
```

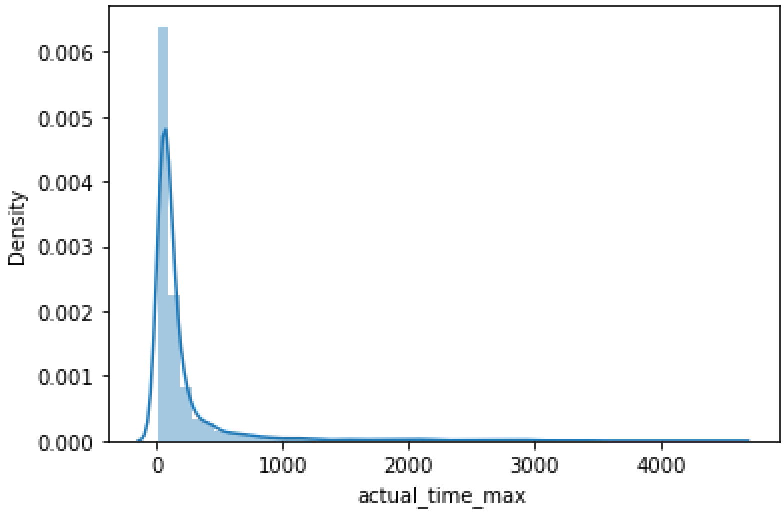


In [47]:

```
sns.distplot(data['actual_time_max'])
# Distribution plot for 'actual_time_max' variable
# PLot is Right skewed
```

Out[47]:

```
<AxesSubplot:xlabel='actual_time_max', ylabel='Density'>
```

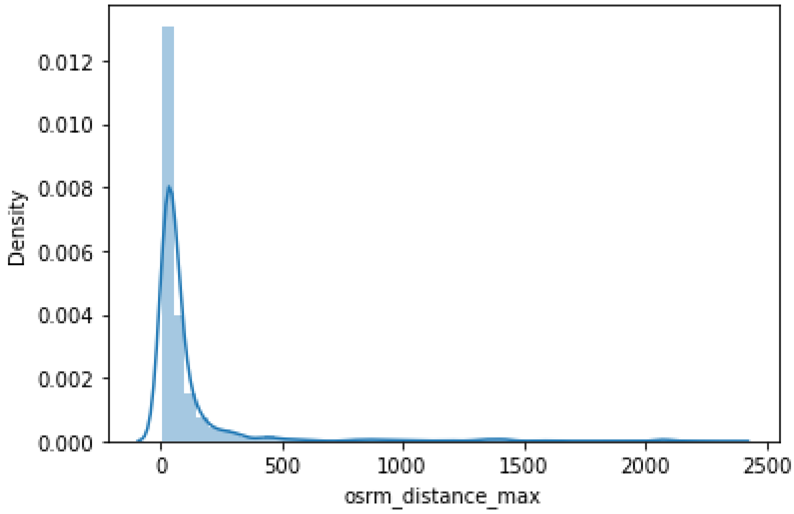


In [48]:

```
sns.distplot(data['osrm_distance_max'])
# Distribution plot for 'osrm_distance_max' variable
# PLot is Right skewed
```

Out[48]:

```
<AxesSubplot:xlabel='osrm_distance_max', ylabel='Density'>
```

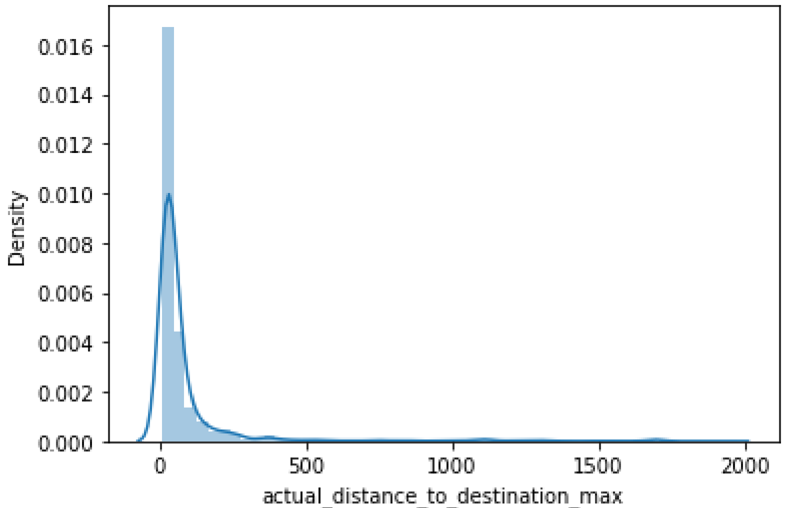


In [49]:

```
sns.distplot(data['actual_distance_to_destination_max'])
# Distribution plot for 'actual_distance_to_destination_max' variable
# PLot is Right skewed
```

Out[49]:

```
<AxesSubplot:xlabel='actual_distance_to_destination_max', ylabel='Density'>
```



In [50]:

```
def outliners(x,col):
    Q1 = np.percentile(x[col], 25)
    Q3 = np.percentile(x[col], 75)
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR
    lower = Q1 - 1.5*IQR
    #print(upper,lower)
    ls=list(x.iloc[((x[col]<lower) | (x[col]>upper)).values].index)
    return ls
list_out=['actual_time_max','osrm_time_max','osrm_distance_max','start_scan_to_end_scan_max
         'segment_actual_time_sum','segment_osrm_time_sum','segment_osrm_distance_sum']
```

In [51]:

```
data['Source_State'].value_counts()[0:10]
# Top 10 'Source_State'
```

Out[51]:

```
Maharashtra       3565
Karnataka         3453
Tamil Nadu        2130
Haryana           2056
Uttar Pradesh     1832
Telangana         1493
Gujarat           1402
West Bengal       1368
Andhra Pradesh    1310
Rajasthan         1185
Name: Source_State, dtype: int64
```

In [52]:

```
data['Source_State'].value_counts()[-10:]
# Bottom 10 'Source_State'
```

Out[52]:

```
Chhattisgarh              52
Arunachal Pradesh         48
Jammu & Kashmir           47
Unknown_Source_State      45
Pondicherry               30
Dadra and Nagar Haveli    15
Meghalaya                 13
Mizoram                    8
Nagaland                   5
Tripura                    1
Name: Source_State, dtype: int64
```

In [53]:

```
data['Source_City'].value_counts()[0:10]
# Top 10 source city
```

Out[53]:

```
Gurgaon_Bilaspur_HB      1063
Bhiwandi_Mankoli_HB       821
Bangalore_Nelmngla_H      768
Bengaluru_Bomsndra_HB     466
Pune_Tathawde_H           403
Chandigarh_Mehmdpur_H     370
Bengaluru_KGAirprt_HB     331
Hyderabad_Shamshbd_H      329
Mumbai                    314
MAA_Poonamallee_HB        300
Name: Source_City, dtype: int64
```

In [54]:

```
data['Source_City'].value_counts()[-10:]
# Bottom 10 Source city
```

Out[54]:

```
Allahabad_Mirapati_L    1
Mumbai_Chndivli_D       1
Hyd_LB-Nagar_Dc         1
Varanasi                1
Fazilka_GndhiChk_D      1
Vadipatti_lalaNGR_D     1
Sumerpur_BazarDPP_D     1
Nagpur_Gondkhry_H       1
Bengaluru_South_D_20    1
Gondal_DC               1
Name: Source_City, dtype: int64
```

In [55]:

```
data['Destination_State'].value_counts()[0:10]
# Top 10 Destination State
```

Out[55]:

```
Karnataka         3505
Maharashtra       3473
Tamil Nadu        2111
Haryana           2019
Uttar Pradesh     1851
Telangana         1552
Gujarat           1402
West Bengal       1399
Andhra Pradesh    1315
Rajasthan         1217
Name: Destination_State, dtype: int64
```

In [56]:

```
data['Destination_State'].value_counts()[-10:]
# Bottom 10 Destination State
```

Out[56]:

```
Chhattisgarh                52
Unknown_Destination_State   48
Jammu & Kashmir             45
Pondicherry                 31
Dadra and Nagar Haveli      17
Meghalaya                   13
Mizoram                     10
Tripura                      2
Daman & Diu                  1
Nagaland                     1
Name: Destination_State, dtype: int64
```

In [57]:

```
data['Destination_City'].value_counts()[0:10]
# Top 10 Destination City
```

Out[57]:

```
Gurgaon_Bilaspur_HB      928
Bangalore_Nelmngla_H     665
Bhiwandi_Mankoli_HB      583
Chandigarh_Mehmdpur_H    431
Hyderabad_Shamshbd_H     405
Bengaluru_Bomsndra_HB    354
Pune_Tathawde_H          330
Sonipat_Kundli_H         329
Bengaluru_KGAirprt_HB    278
Ahmedabad_East_H_1       257
Name: Destination_City, dtype: int64
```

In [58]:

```
data['Destination_City'].value_counts()[-10:]
# Bottom 10 Destination City
```

Out[58]:

```
Daman_DC                 1
Vadodara_Karelibaug_DC   1
Kamarpukur_ChatiDPP_D    1
Berhampur_Chatrpr_DC     1
AmaDubi_Bulabeda_D       1
Khetri_NagarDPP_D        1
Phulbani_Krusphrma_D     1
Angul_Central_I_2        1
Kangra_Central_D_2       1
Jasdan_MotiDPP_D         1
Name: Destination_City, dtype: int64
```

In [59]:

```
(data['Source_City']+' to '+data['Destination_City']).value_counts()[0:20]
# Top 10 source city to destination city
```

Out[59]:

```
Bangalore_Nelmngla_H to Bengaluru_KGAirprt_HB     151
Bangalore_Nelmngla_H to Bengaluru_Bomsndra_HB     127
Bengaluru_Bomsndra_HB to Bengaluru_KGAirprt_HB    121
Bengaluru_KGAirprt_HB to Bangalore_Nelmngla_H     108
Pune_Tathawde_H to Bhiwandi_Mankoli_HB            107
Bhiwandi_Mankoli_HB to Mumbai                     105
Bengaluru_Bomsndra_HB to Bangalore_Nelmngla_H     102
Delhi_Gateway_HB to Gurgaon_Bilaspur_HB           100
Mumbai_Chndivli_PC to Bhiwandi_Mankoli_HB          99
Gurgaon_Bilaspur_HB to Sonipat_Kundli_H            92
Sonipat_Kundli_H to Gurgaon_Bilaspur_HB            86
Bengaluru_KGAirprt_HB to Bengaluru_Bomsndra_HB     86
Pune_Tathawde_H to PNQ                             84
Bhiwandi_Mankoli_HB to Mumbai_MiraRd_IP            78
Del_Okhla_PC to Gurgaon_Bilaspur_HB                76
Bhiwandi_Mankoli_HB to Pune_Tathawde_H             72
Mumbai to Mumbai_MiraRd_IP                         72
Ludhiana_MilrGanj_HB to Chandigarh_Mehmdpur_H      71
Mumbai to Mumbai_Sanpada_I                         67
Chandigarh_Mehmdpur_H to Gurgaon_Bilaspur_HB       66
dtype: int64
```

In [60]:

```
city_list=list((data['Source_City']+' to '+data['Destination_City']).value_counts()[0:20].i
```

In [61]:

```
(data['Source_State']+' to '+data['Destination_State']).value_counts()[0:20]
# Top 10 source state to destination state
```

Out[61]:

```
Maharashtra to Maharashtra          3255
Karnataka to Karnataka              3158
Tamil Nadu to Tamil Nadu            2021
Uttar Pradesh to Uttar Pradesh      1526
Telangana to Telangana              1328
West Bengal to West Bengal          1296
Gujarat to Gujarat                  1280
Andhra Pradesh to Andhra Pradesh    1139
Rajasthan to Rajasthan              1070
Bihar to Bihar                      1023
Haryana to Haryana                  1005
Punjab to Punjab                     810
Kerala to Kerala                     717
Madhya Pradesh to Madhya Pradesh     595
Delhi to Haryana                     451
Assam to Assam                       407
Haryana to Delhi                     315
Uttarakhand to Uttarakhand           313
Jharkhand to Jharkhand               269
Delhi to Delhi                       217
dtype: int64
```

In [62]:

```
newdata=data.copy('deep')
```

In [63]:

```
newdata['City_join']=data['Source_City']+' to '+data['Destination_City']
```

In [64]:

```
ans=newdata.groupby(['City_join'])\
['actual_time_max','osrm_time_max', 'osrm_distance_max','start_scan_to_end_scan_max', 'actu
aggregate({'actual_time_max':'mean',
           'osrm_time_max':'mean',
           'osrm_distance_max':'mean',
           'start_scan_to_end_scan_max':'mean',
           'actual_distance_to_destination_max':'mean'
           }).reset_index()
```

In [65]:

```
ans[ans['City_join'].isin(city_list)]
# Top 20 source to destination city with average 'actual_time','osrm_time', 'osrm_distance'
#and 'actual_distance_to_destination'
```

Out[65]:

	City_join	actual_time_max	osrm_time_max	osrm_distance_max	start_scar
213	Bangalore_Nelmngla_H to Bengaluru_Bomsndra_HB	91.850394	50.535433	50.572376	
215	Bangalore_Nelmngla_H to Bengaluru_KGAirprt_HB	87.874172	48.086093	38.600475	
302	Bengaluru_Bomsndra_HB to Bangalore_Nelmngla_H	97.137255	56.470588	46.782357	
307	Bengaluru_Bomsndra_HB to Bengaluru_KGAirprt_HB	114.661157	56.801653	57.071903	
321	Bengaluru_KGAirprt_HB to Bangalore_Nelmngla_H	105.231481	51.092593	42.008400	
324	Bengaluru_KGAirprt_HB to Bengaluru_Bomsndra_HB	135.848837	55.872093	53.022995	
405	Bhiwandi_Mankoli_HB to Mumbai	61.285714	22.314286	27.168875	
411	Bhiwandi_Mankoli_HB to Mumbai_MiraRd_IP	80.525641	24.615385	28.156754	
419	Bhiwandi_Mankoli_HB to Pune_Tathawde_H	223.763889	97.263889	130.558300	
556	Chandigarh_Mehmdpur_H to Gurgaon_Bilaspur_HB	451.772727	211.530303	280.380277	
716	Del_Okhla_PC to Gurgaon_Bilaspur_HB	114.302632	60.618421	65.183671	
746	Delhi_Gateway_HB to Gurgaon_Bilaspur_HB	69.880000	41.590000	43.029667	
1059	Gurgaon_Bilaspur_HB to Sonipat_Kundli_H	216.456522	98.619565	103.789666	
1665	Ludhiana_MilrGanj_HB to Chandigarh_Mehmdpur_H	135.929577	64.521127	91.966234	
1849	Mumbai to Mumbai_MiraRd_IP	50.666667	15.958333	20.557504	
1852	Mumbai to Mumbai_Sanpada_I	55.805970	19.582090	22.436130	
1857	Mumbai_Chndivli_PC to Bhiwandi_Mankoli_HB	80.868687	20.888889	25.636838	
2174	Pune_Tathawde_H to Bhiwandi_Mankoli_HB	218.766355	100.504673	128.630692	
2183	Pune_Tathawde_H to PNQ	63.630952	20.309524	18.706305	
2496	Sonipat_Kundli_H to Gurgaon_Bilaspur_HB	210.255814	96.313953	111.712436	

In [66]:

```
data.columns
```

Out[66]:

```
Index(['data', 'trip_creation_time', 'trip_uuid', 'route_type',
       'od_start_time', 'od_end_time', 'source_center_pincode',
       'destination_center_pincode', 'Source_City', 'Source_State',
       'Destination_City', 'Destination_State', 'actual_time_max',
       'actual_time_count', 'osrm_time_max', 'osrm_distance_max',
       'start_scan_to_end_scan_max', 'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean',
       'trip_creation_year', 'trip_creation_month', 'trip_creation_day',
       'od_start_year', 'od_start_month', 'od_start_day', 'od_end_year',
       'od_end_month', 'od_end_day', 'od_delta'],
      dtype='object')
```

In [67]:

```
for i in list_out:
    print((len(data.drop(outliners(data,i),axis=0))/len(data)*100),'% outliers in',i)
# Every column has more than 80% outliners
```

```
88.04656983579203 % outliers in actual_time_max
89.02119913534833 % outliers in osrm_time_max
88.35374872008798 % outliers in osrm_distance_max
89.6810648867989 % outliers in start_scan_to_end_scan_max
87.55356668815655 % outliers in actual_distance_to_destination_max
88.03519284007736 % outliers in segment_actual_time_sum
88.04277750388714 % outliers in segment_osrm_time_sum
88.22101710341688 % outliers in segment_osrm_distance_sum
```

In [68]:

```
(max(data['actual_time_max']),min(data['actual_time_max'])),(max(data['osrm_time_max']),min
```

Out[68]:

```
((4532.0, 9.0), (1686.0, 6.0))
```

In [69]:

```
(max(data['osrm_distance_max']),min(data['osrm_distance_max'])),(max(data['actual_distance_
```

Out[69]:

```
((2326.1991000000003, 9.0729), (1927.4477046975032, 9.00135089146556))
```

In [70]:

```
(max(data['start_scan_to_end_scan_max']),min(data['start_scan_to_end_scan_max']))
```

Out[70]:

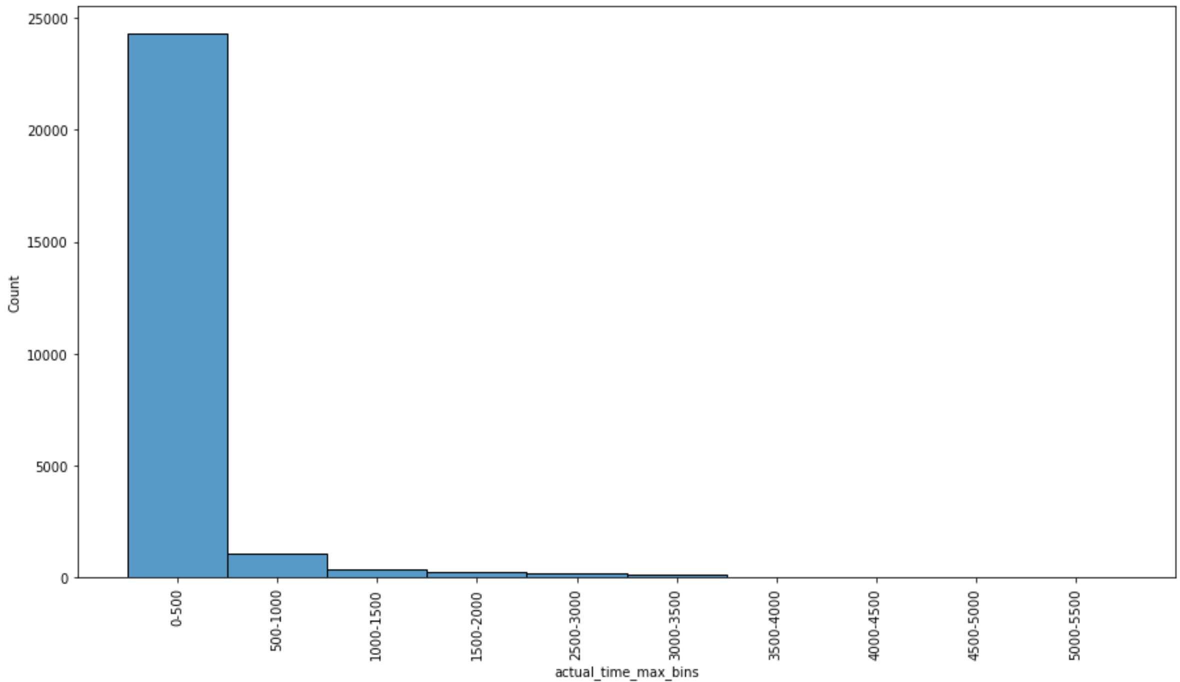
```
(7898.0, 20.0)
```

In [71]:

```
bins=[0,500,1000,1500,2000,2500,3000,3500,4000,4500,5000]
labels=['0-500','500-1000','1000-1500','1500-2000','2500-3000','3000-3500','3500-4000','400
data['actual_time_max_bins']=pd.cut(data['actual_time_max'], bins=bins, labels=labels)
bins=[0,500,1000,1500,2000]
labels=['0-500','500-1000','1000-1500','1500-2000']
data['osrm_time_max_bins']=pd.cut(data['osrm_time_max'], bins=bins, labels=labels)
bins=[0,500,1000,1500,2000,2500]
labels=['0-500','500-1000','1000-1500','1500-2000','2500-3000']
data['osrm_distance_max_bins']=pd.cut(data['osrm_distance_max'], bins=bins, labels=labels)
bins=[0,500,1000,1500,2000]
labels=['0-500','500-1000','1000-1500','1500-2000']
data['actual_distance_to_destination_max_bins']=pd.cut(data['actual_distance_to_destination
bins=[0,500,1000,1500,2000,2500,3000,3500,4000,4500,5000,5500,6000,6500,7000,7500,8000]
labels=['0-500','500-1000','1000-1500','1500-2000','2500-3000','3000-3500','3500-4000','400
data['start_scan_to_end_scan_max_bins']=pd.cut(data['start_scan_to_end_scan_max'], bins=bin
# Binning numerical variables
```

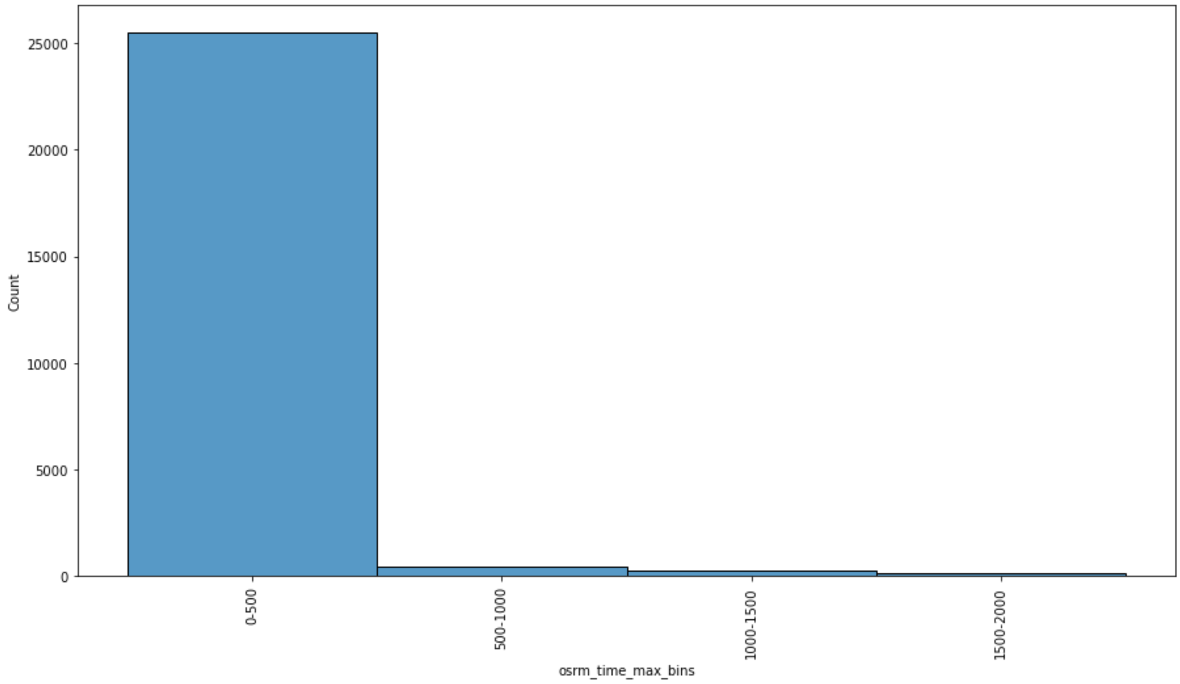
In [72]:

```
sns.histplot(data['actual_time_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'actual_time_max_bins'
```



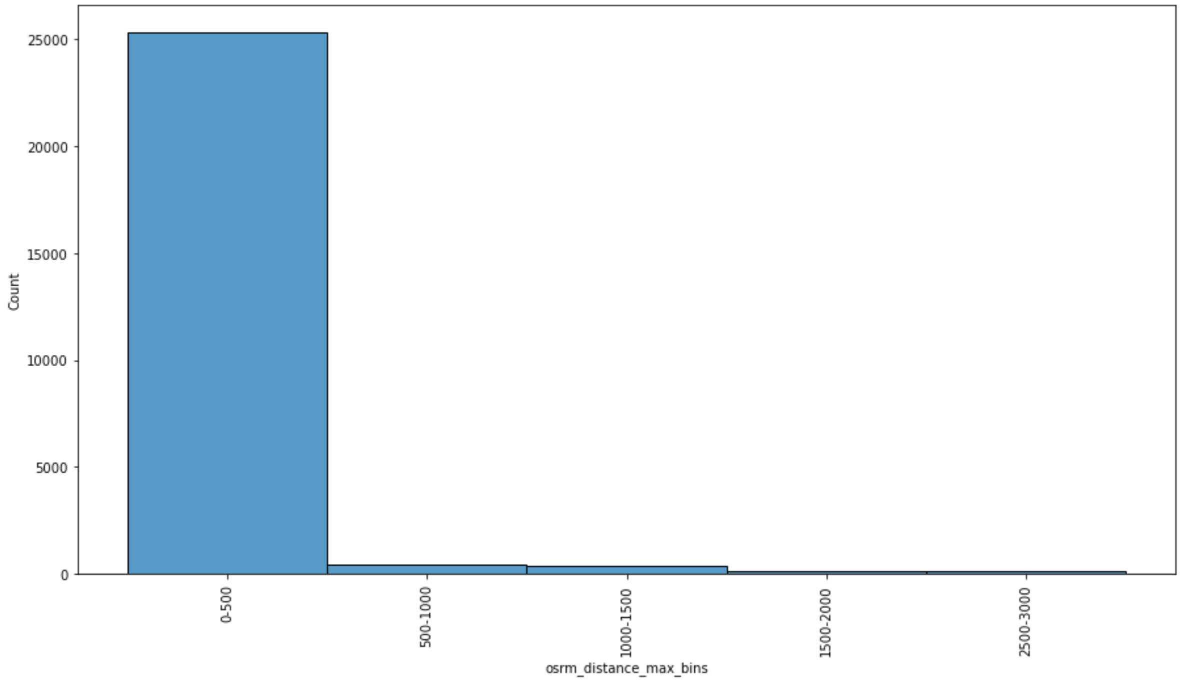
In [73]:

```
sns.histplot(data['osrm_time_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'osrm_time_max_bins'
```



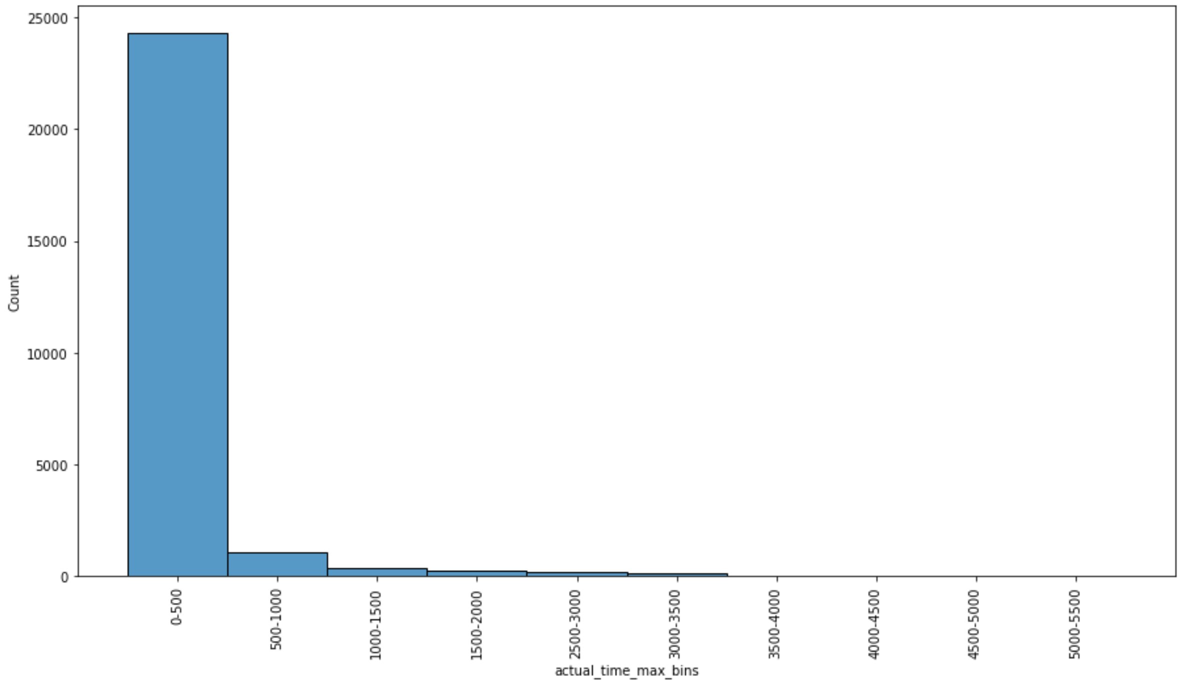
In [74]:

```
sns.histplot(data['osrm_distance_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'osrm_distance_max_bins'
```



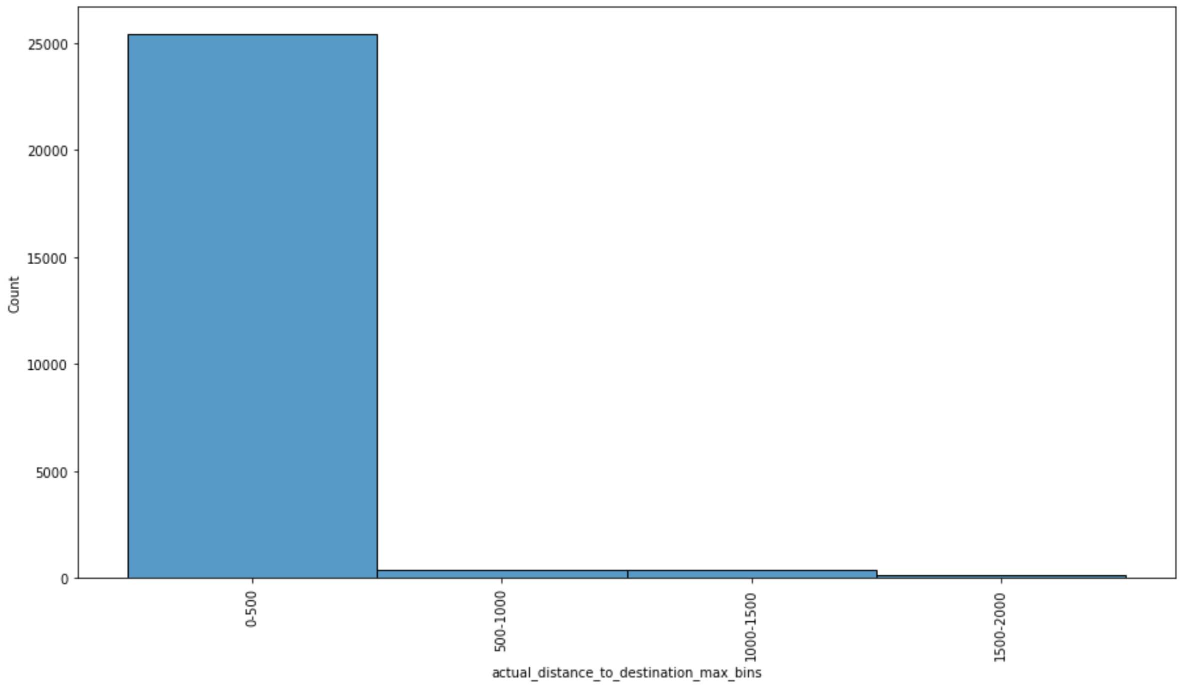
In [75]:

```
sns.histplot(data['actual_time_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'actual_time_max_bins'
```



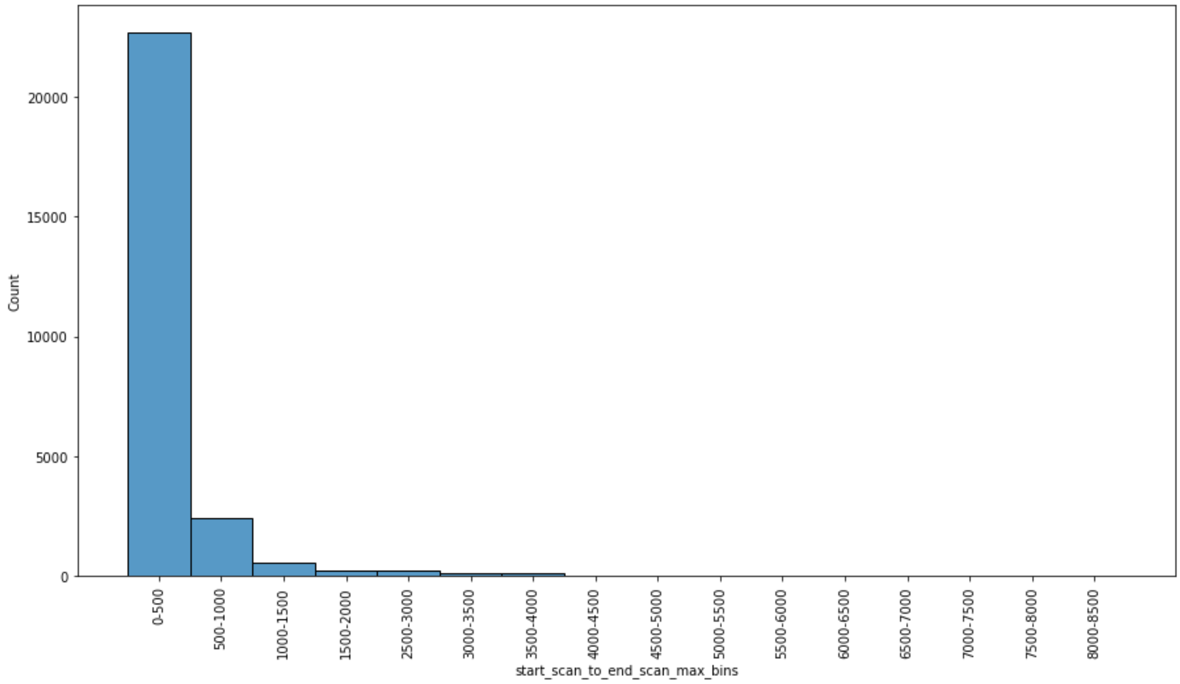
In [76]:

```
sns.histplot(data['actual_distance_to_destination_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'actual_distance_to_destination_max_bins'
```



In [77]:

```
sns.histplot(data['start_scan_to_end_scan_max_bins'])
plt.xticks(rotation=90)
plt.gcf().set_size_inches(15, 8)
# Count plot for each bin for 'start_scan_to_end_scan_max_bins'
```



In [78]:

```
data.columns
```

Out[78]:

```
Index(['data', 'trip_creation_time', 'trip_uuid', 'route_type',
       'od_start_time', 'od_end_time', 'source_center_pincode',
       'destination_center_pincode', 'Source_City', 'Source_State',
       'Destination_City', 'Destination_State', 'actual_time_max',
       'actual_time_count', 'osrm_time_max', 'osrm_distance_max',
       'start_scan_to_end_scan_max', 'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean',
       'trip_creation_year', 'trip_creation_month', 'trip_creation_day',
       'od_start_year', 'od_start_month', 'od_start_day', 'od_end_year',
       'od_end_month', 'od_end_day', 'od_delta', 'actual_time_max_bins',
       'osrm_time_max_bins', 'osrm_distance_max_bins',
       'actual_distance_to_destination_max_bins',
       'start_scan_to_end_scan_max_bins'],
      dtype='object')
```

In [79]:

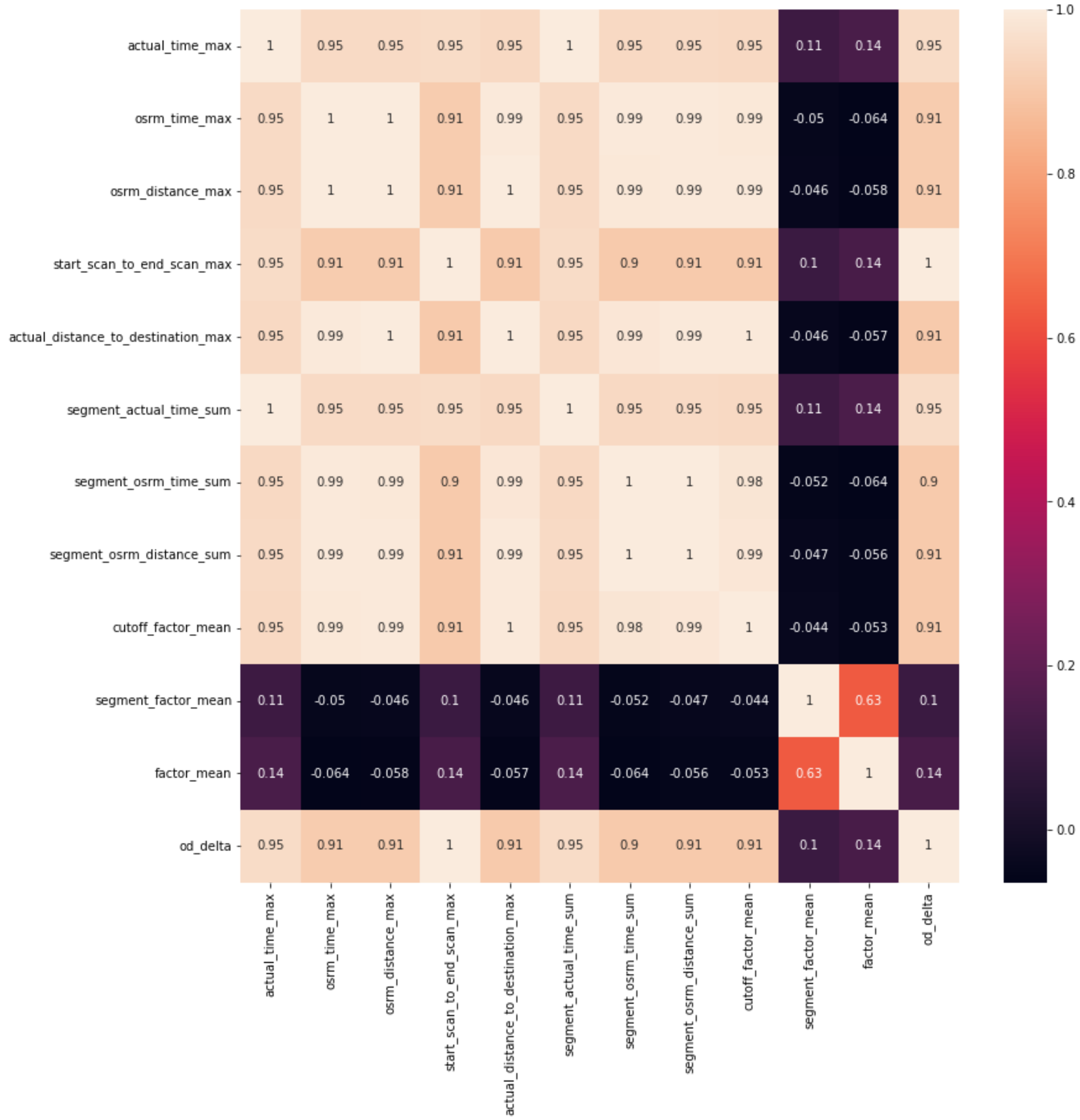
```
data['actual_time_avg']=data['actual_time_max']/data['actual_time_count']
data['osrm_time_avg']=data['osrm_time_max']/data['actual_time_count']
data['osrm_distance_avg']=data['osrm_distance_max']/data['actual_time_count']
data['actual_distance_to_destination_avg']=data['actual_distance_to_destination_max']/data[
data['segment_actual_time_avg']=data['segment_actual_time_sum']/data['segment_actual_time_c
data['segment_osrm_time_avg']=data['segment_osrm_time_sum']/data['segment_osrm_time_count']
data['segment_osrm_distance_count_avg']=data['segment_osrm_distance_sum']/data['segment_osr
# Finding average for all the columns
```

In [80]:

```
datatypes=['actual_time_max',
 'osrm_time_max',
 'osrm_distance_max',
 'start_scan_to_end_scan_max',
 'actual_distance_to_destination_max',
 'segment_actual_time_sum',
 'segment_osrm_time_sum',
 'segment_osrm_distance_sum',
 'cutoff_factor_mean',
 'segment_factor_mean',
 'factor_mean',
 'od_delta']
```

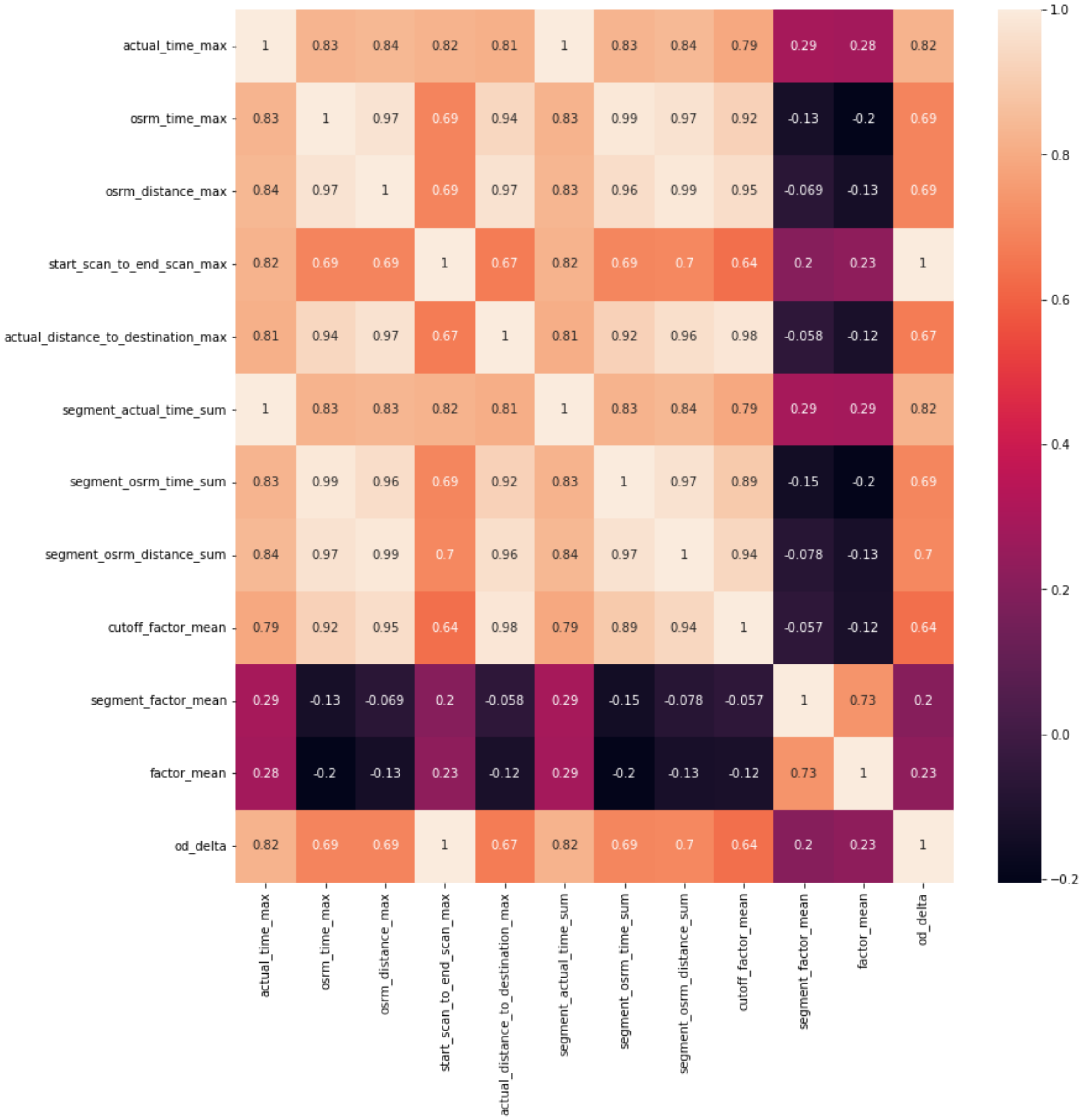
In [81]:

```
fig,ax=plt.subplots(figsize=(13,13))
sns.heatmap(data[datatypes].corr(method ='pearson'),annot=True,ax=ax)
plt.show()
# Pearson correlation coefficient for all the numerical variables
# Pearson coefficient tells the linearity  between variables
```



In [82]:

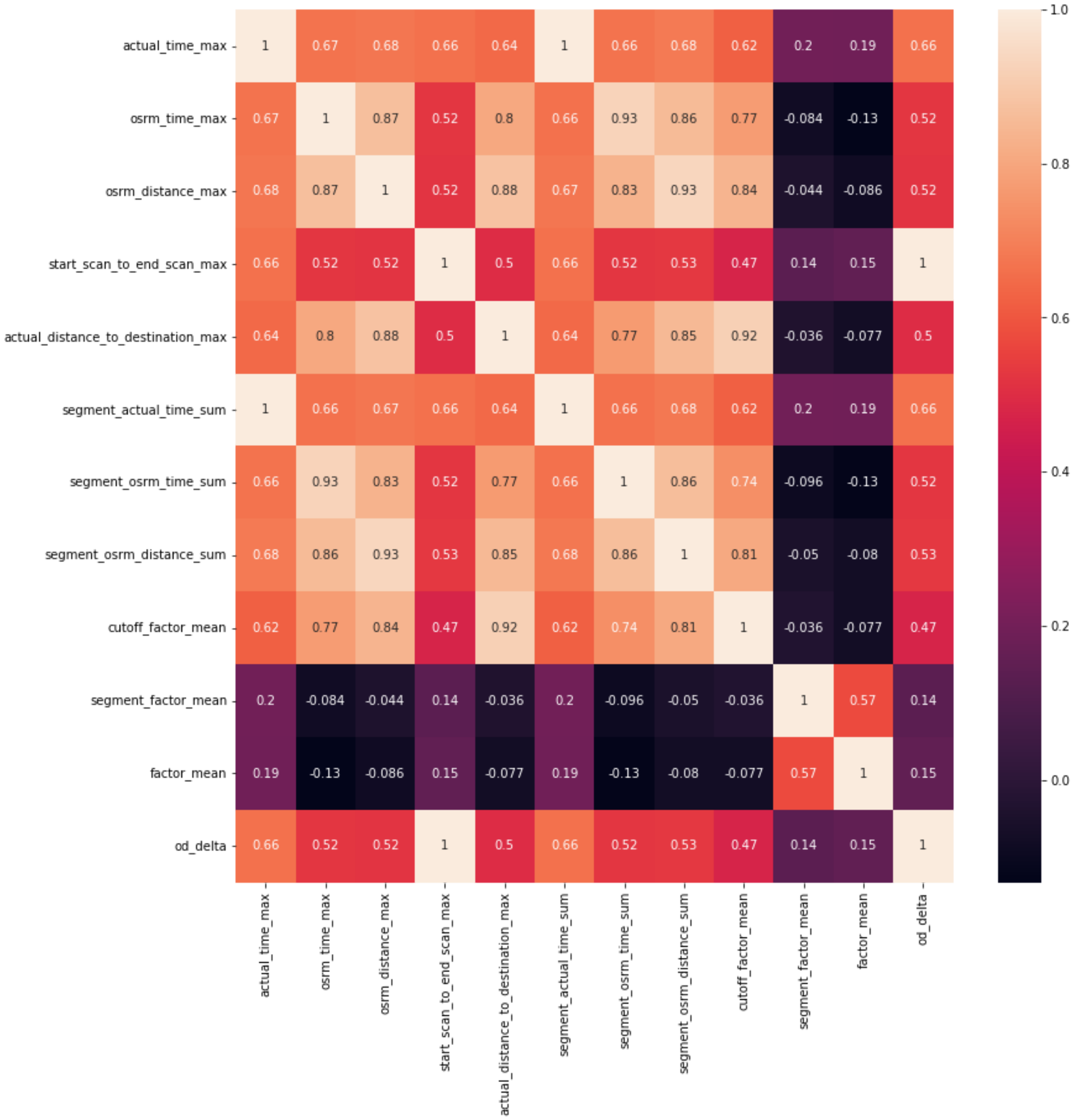
```
fig,ax=plt.subplots(figsize=(13,13))
sns.heatmap(data[datatypes].corr(method ='spearman'),annot=True
            ,ax=ax)
plt.show()
# Spearman correlation coefficient for all the numerical variables
# Pearson coefficient tells the monotonicity  between variables
```



	actual_time_max	osrm_time_max	osrm_distance_max	start_scan_to_end_scan_max	actual_distance_to_destination_max	segment_actual_time_sum	segment_osrm_time_sum	segment_osrm_distance_sum	cutoff_factor_mean	segment_factor_mean	factor_mean	od_delta
actual_time_max	1	0.83	0.84	0.82	0.81	1	0.83	0.84	0.79	0.29	0.28	0.82
osrm_time_max	0.83	1	0.97	0.69	0.94	0.83	0.99	0.97	0.92	-0.13	-0.2	0.69
osrm_distance_max	0.84	0.97	1	0.69	0.97	0.83	0.96	0.99	0.95	-0.069	-0.13	0.69
start_scan_to_end_scan_max	0.82	0.69	0.69	1	0.67	0.82	0.69	0.7	0.64	0.2	0.23	1
actual_distance_to_destination_max	0.81	0.94	0.97	0.67	1	0.81	0.92	0.96	0.98	-0.058	-0.12	0.67
segment_actual_time_sum	1	0.83	0.83	0.82	0.81	1	0.83	0.84	0.79	0.29	0.29	0.82
segment_osrm_time_sum	0.83	0.99	0.96	0.69	0.92	0.83	1	0.97	0.89	-0.15	-0.2	0.69
segment_osrm_distance_sum	0.84	0.97	0.99	0.7	0.96	0.84	0.97	1	0.94	-0.078	-0.13	0.7
cutoff_factor_mean	0.79	0.92	0.95	0.64	0.98	0.79	0.89	0.94	1	-0.057	-0.12	0.64
segment_factor_mean	0.29	-0.13	-0.069	0.2	-0.058	0.29	-0.15	-0.078	-0.057	1	0.73	0.2
factor_mean	0.28	-0.2	-0.13	0.23	-0.12	0.29	-0.2	-0.13	-0.12	0.73	1	0.23
od_delta	0.82	0.69	0.69	1	0.67	0.82	0.69	0.7	0.64	0.2	0.23	1

In [83]:

```
fig,ax=plt.subplots(figsize=(13,13))
sns.heatmap(data[datatypes].corr(method ='kendall'),annot=True,ax=ax)
plt.show()
# Kendall correlation coefficient for all the numerical variables
```



In [84]:

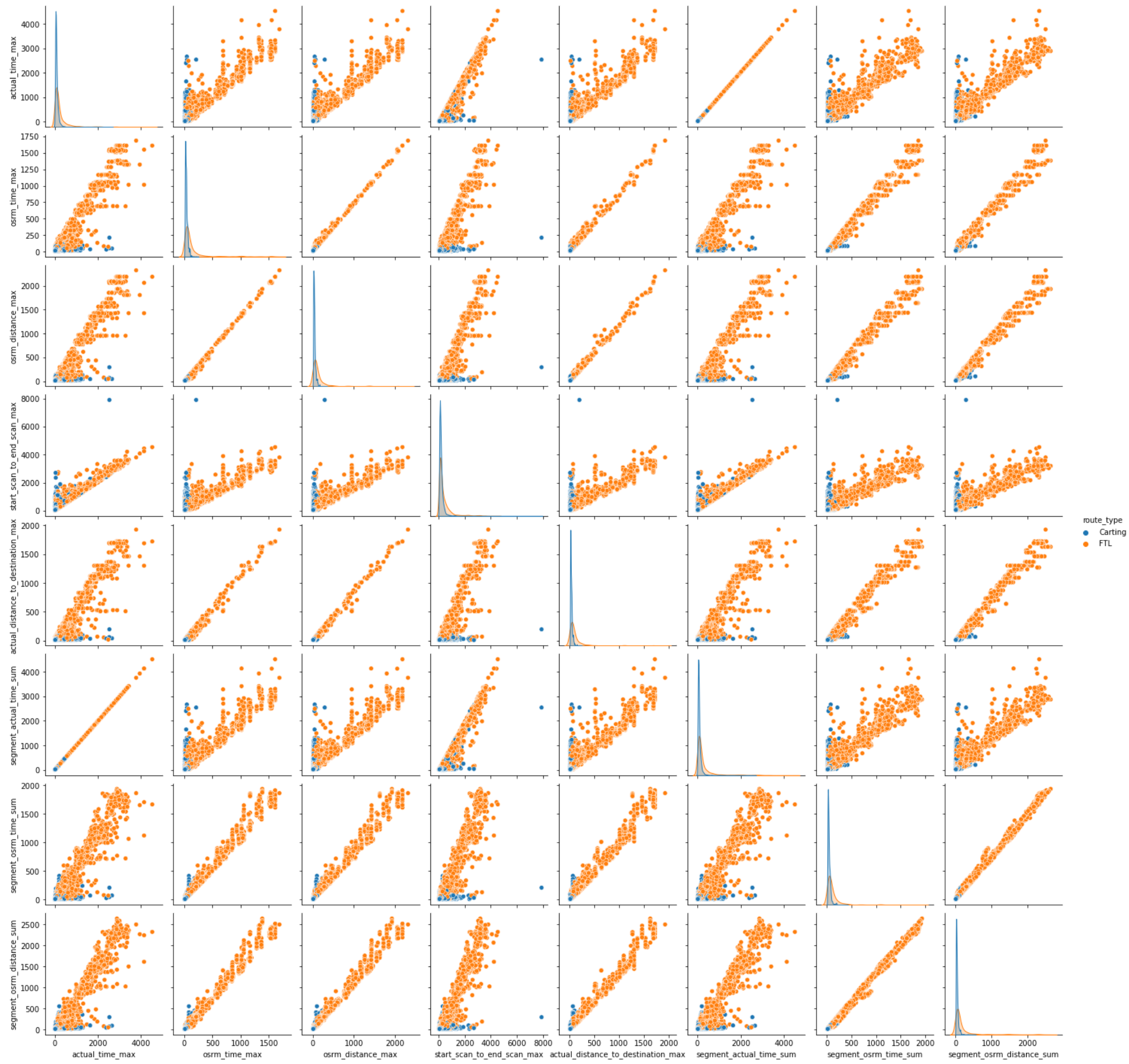
```
datatypes=['actual_time_max',
 'osrm_time_max',
 'osrm_distance_max',
 'start_scan_to_end_scan_max',
 'actual_distance_to_destination_max',
 'segment_actual_time_sum',
 'segment_osrm_time_sum',
 'segment_osrm_distance_sum']
```

In [85]:

```
sns.pairplot(data, vars=datatypes,hue='route_type')
# Pair plot for all numerical variables
# Actual time and distnace have a linear relationship
```

Out[85]:

```
<seaborn.axisgrid.PairGrid at 0x1c3b2493340>
```



In [86]:

```
A=pd.DataFrame()
```

In [87]:

```
# Normality Tests
def qqplot(dff,a):
    A=pd.DataFrame()
    print('This test is for visual only')
    fig=sm.qqplot(dff[a],line='45')
    plt.grid()
    plt.show()
def kstest(dff,a):
    A=pd.DataFrame()
    stat,p_value=stats.kstest(dff[a],'norm')
    print('Ho: The sample {} follows normal distribution'.format(a))
    print('Ha: The sample {} does not follows normal distribution'.format(a))
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    print()
    if p_value>0.05:
        print('The sample {} follows normal distribution'.format(a))
    else:
        print('The sample {} does not follows normal distribution'.format(a))
def shapiro(dff,a):
    A=pd.DataFrame()
    stat,p_value=stats.shapiro(dff[a])
    print('Ho: The sample {} follows normal distribution'.format(a))
    print('Ha: The sample {} does not follows normal distribution'.format(a))
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} follows normal distribution'.format(a))
    else:
        print('The sample {} does not follows normal distribution'.format(a))
```

In [88]:

```
# Tranformations and the tests from 'Normality Tests'
def logtrans(dff,a):
    A=pd.DataFrame()
    A[a]=np.log(dff[a])
    print('After applying log transforms')
    qqplot(A,a)
    kstest(A,a)
    print()
    shapiro(A,a)

def box(dff,a):
    A=pd.DataFrame()
    fitted_data, fitted_lambda = stats.boxcox(dff[a])
    A[a]=fitted_data
    print('After applying boxcox transforms')
    qqplot(A,a)
    kstest(A,a)
    print()
    shapiro(A,a)

def rec(dff,a):
    A=pd.DataFrame()
    A[a]=1/dff[a]
    print('After applying reciprocal transforms')
    qqplot(A,a)
    kstest(A,a)
    print()
    shapiro(A,a)

def sq(dff,a):
    A=pd.DataFrame()
    A[a]=np.sqrt(dff[a])
    print('After applying log transforms')
    qqplot(A,a)
    kstest(A,a)
    print()
    shapiro(A,a)
```

In [89]:

```
# Correlation Tests
def pear(dff,a,b):
    print('Ho: The sample {} and {} are independent'.format(a,b))
    print('Ha: The sample {} and {} are dependent'.format(a,b))
    stat, p_value = stats.pearsonr(dff[a], dff[b])
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} and {} are independent'.format(a,b))
    else:
        print('The sample {} and {} are dependent'.format(a,b))

def spearmanr(dff,a,b):
    print('Ho: The sample {} and {} are independent'.format(a,b))
    print('Ha: The sample {} and {} are dependent'.format(a,b))
    stat, p_value = stats.spearmanr(dff[a], dff[b])
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} and {} are independent'.format(a,b))
    else:
        print('The sample {} and {} are dependent'.format(a,b))

def kend(dff,a,b):
    print('Ho: The sample {} and {} are independent'.format(a,b))
    print('Ha: The sample {} and {} are dependent'.format(a,b))
    stat, p_value = stats.kendalltau(dff[a], dff[b])
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} and {} are independent'.format(a,b))
    else:
        print('The sample {} and {} are dependent'.format(a,b))
```

In [90]:

```
# Variance tests
def bar(dff,a,b):
    stat, p_value = stats.bartlett(dff[a], dff[b])
    print('Ho: The sample {} and {} have equal variance'.format(a,b))
    print('Ha: The sample {} and {} are unequal variance'.format(a,b))
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} and {} have equal variance'.format(a,b))
    else:
        print('The sample {} and {} unequal variance'.format(a,b))
def lev(dff,a,b):
    stat, p_value = stats.levene(dff[a], dff[b],center='median')
    print('Ho: The sample {} and {} have equal variance'.format(a,b))
    print('Ha: The sample {} and {} are unequal variance'.format(a,b))
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample {} and {} have equal variance'.format(a,b))
    else:
        print('The sample {} and {} unequal variance'.format(a,b))
```

In [91]:

```
# tests when data is not normal
def mann(dff,a,b):
    print('Ho: The sum of ranking of {} and {} are equal'.format(a,b))
    print('Ha: The sum of ranking of {} and {} are not equal'.format(a,b))
    print('Assumption of a Mann-Whitney U test are:')
    print('1.Should have a ordinal variable\n''2. Only 2 independent random samples with a
    stat,p_value=stats.mannwhitneyu(dff[a], dff[b])
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sum of ranking of {} and {} are equal'.format(a,b))
    else:
        print('The sum of ranking of {} and {} are not equal'.format(a,b))
def will(dff,a,b):
    print('Ho: The central tendencies of {} and {} are equal'.format(a,b))
    print('Ha: The central tendencies of {} and {} are not equal'.format(a,b))
    print('Assumption of a Wilcoxon Signed-Rank Test are:')
    print('1.Should have a ordinal variable\n''2. Only 2 independent random samples with a
    stat,p_value=stats.mannwhitneyu(dff[a], dff[b])
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The central tendencies of {} and {} are equal'.format(a,b))
    else:
        print('The central tendencies of {} and {} are not equal'.format(a,b))
```

In [92]:

```
# T test with equal variance
def ttest(ddf,a,b):
    print('Ho: The sample means of {} and {} are equal'.format(a,b))
    print('Ha: The sample means of {} and {} are not equal'.format(a,b))
    print('Assumption of a t_test are:')
    print('1.Observations of 2 samples are independent\n''2. Both samples are approx normal
    stat,p_value=stats.ttest_ind(ddf[a], ddf[b], equal_var=True)
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample means of {} and {} are equal'.format(a,b))
    else:
        print('The sample means of {} and {} are not equal'.format(a,b))
```

In [93]:

```
# T test with unequal variance
def ttestv(ddf,a,b):
    print('Ho: The sample means of {} and {} are equal'.format(a,b))
    print('Ha: The sample means of {} and {} are not equal'.format(a,b))
    print('Assumption of a t_test are:')
    print('1.Observations of 2 samples are independent\n''2. Both samples are approx normal
    stat,p_value=stats.ttest_ind(ddf[a], ddf[b], equal_var=False)
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample means of {} and {} are equal'.format(a,b))
    else:
        print('The sample means of {} and {} are not equal'.format(a,b))
```

In [94]:

```
#chi2
def chis(dff,a,b):
    ans=dff.groupby([dff[a]])[[b]].sum().reset_index()
    stat, p_value, dof, ex=stats.chi2_contingency(ans[b])
    print('Ho: The samples are independent')
    print('Ha: The samples dependent')
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The samples are independent')
    else:
        print('The samples are dependent')
```

In [95]:

```
def chis2(dff,a,b,c):
    stat, p_value, dof, ex=stats.chi2_contingency(dff[a],dff[b],dff[c])
    print('Ho: The samples are independent')
    print('Ha: The samples dependent')
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The samples are independent')
    else:
        print('The samples are dependent')
```

In [96]:

```
# ANOVA
def anova(dff,a,b,c):
    print('Assumptions of ANOVA are:')
    print('1. Each group assumptions is gaussian\n 2.Each group variance is roughly the sam
    print('Ho: The sample means equal'.format(a,b))
    print('Ha: There exists atleast one sample that is not equal to other mean'.format(a,b)
    stat,p_value=stats.f_oneway(dff[a],dff[b],dff[c])
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample means equal')
    else:
        print('The sample means are not equal')
def kwtest(dff,a,b,c,d):
    print('Assumptions of Kruskal-Wallis one-way analysis of variance are:')
    print('Ho: The sample median equal'.format(a,b))
    print('Ha: There exists atleast one sample that is not equal to other median'.format(a,
    stat,p_value=stats.kruskal(dff[a],dff[b],dff[c],dff[d])
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample medians equal')
    else:
        print('The sample medians are not equal')

def kwtest1(dff,a,b,c):
    print('Assumptions of Kruskal-Wallis one-way analysis of variance are:')
    print('Ho: The sample median equal'.format(a,b))
    print('Ha: There exists atleast one sample that is not equal to other median'.format(a,
    stat,p_value=stats.kruskal(dff[a],dff[b],dff[c])
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample medians equal')
    else:
        print('The sample medians are not equal')
```

In [97]:

```
data.columns
```

Out[97]:

```
Index(['data', 'trip_creation_time', 'trip_uuid', 'route_type',
       'od_start_time', 'od_end_time', 'source_center_pincode',
       'destination_center_pincode', 'Source_City', 'Source_State',
       'Destination_City', 'Destination_State', 'actual_time_max',
       'actual_time_count', 'osrm_time_max', 'osrm_distance_max',
       'start_scan_to_end_scan_max', 'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean',
       'trip_creation_year', 'trip_creation_month', 'trip_creation_day',
       'od_start_year', 'od_start_month', 'od_start_day', 'od_end_year',
       'od_end_month', 'od_end_day', 'od_delta', 'actual_time_max_bins',
       'osrm_time_max_bins', 'osrm_distance_max_bins',
       'actual_distance_to_destination_max_bins',
       'start_scan_to_end_scan_max_bins', 'actual_time_avg', 'osrm_time_av
g',
       'osrm_distance_avg', 'actual_distance_to_destination_avg',
       'segment_actual_time_avg', 'segment_osrm_time_avg',
       'segment_osrm_distance_count_avg'],
      dtype='object')
```

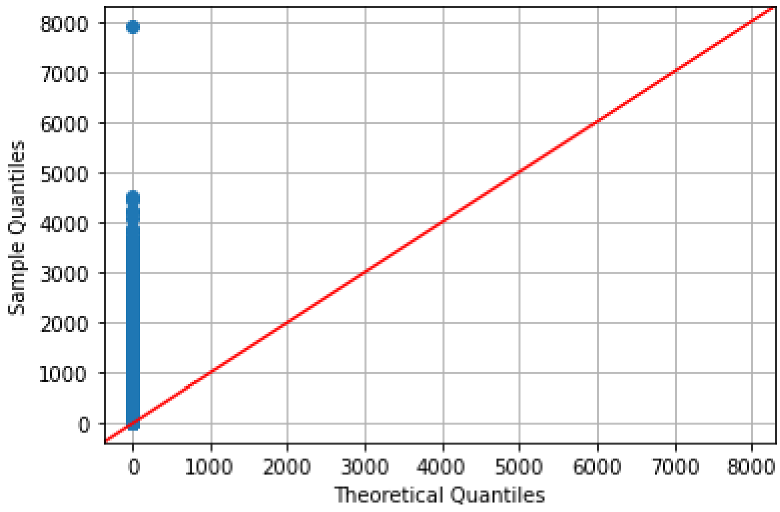
Hypothesis and visual analysis was done on all the numarical variables (With stating assumptions, null hypothsis, alternate hypothesis and confidence intervals)
Normality Tests
QQplot, Shapiro–Wilk test and Kolmogorov–Smirnov test
Tranformation
Log, Box-Cox, Reciprocal and Square root
Correlation Tests
Pearson, Spearman and kendall rank
Variance Test
Bartlett and Levene
Tests when Data is not normal
Mann-Whitney U test and Wilcoxon signed-rank test
Test when data is normal
T Test (equal and unequal variances), AVOVA
Independence test
Chi2 test
Kruskal–Wallis one-way analysis of variance

Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

In [98]:

```
qqplot(data,'start_scan_to_end_scan_max'),shapiro(data,'start_scan_to_end_scan_max'),kstest
```

```
This test is for visual only
```




```
Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on

stat=0.530, p_value=0.000
The sample start_scan_to_end_scan_max does not follows normal distribution
Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on
stat=1.000, p_value=0.000

The sample start_scan_to_end_scan_max does not follows normal distribution
```

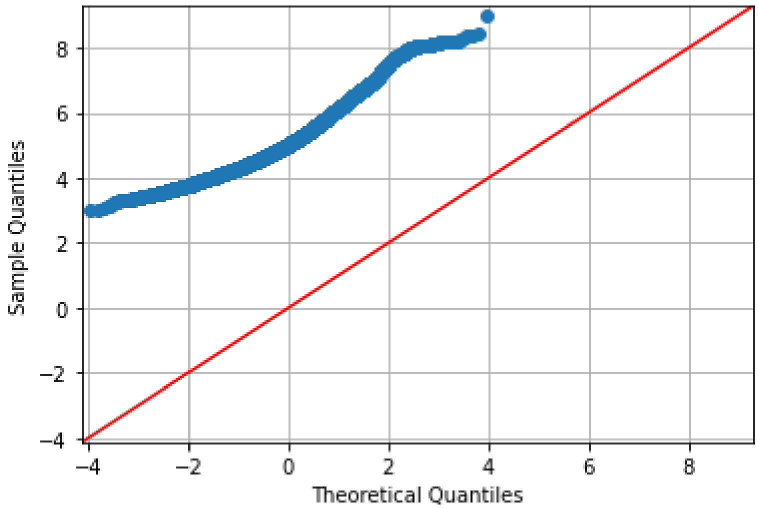
Out[98]:

```
(None, None, None)
```

In [99]:

```
logtrans(data,'start_scan_to_end_scan_max')
```

```
After applying log transforms
This test is for visual only
```

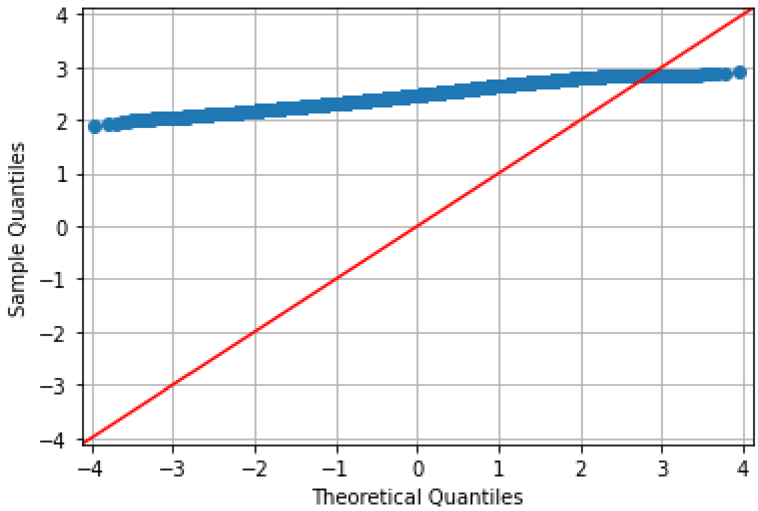



```
Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on
stat=0.999, p_value=0.000

The sample start_scan_to_end_scan_max does not follows normal distribution

Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on

stat=0.960, p_value=0.000
The sample start_scan_to_end_scan_max does not follows normal distribution
```

In [100]:

```
box(data,'start_scan_to_end_scan_max')
```

```
After applying boxcox transforms
This test is for visual only
```




```
Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on
stat=0.979, p_value=0.000

The sample start_scan_to_end_scan_max does not follows normal distribution

Ho: The sample start_scan_to_end_scan_max follows normal distribution
Ha: The sample start_scan_to_end_scan_max does not follows normal distributi
on

stat=0.995, p_value=0.000
The sample start_scan_to_end_scan_max does not follows normal distribution
```

In [101]:

```
bar(data,'start_scan_to_end_scan_max','od_delta'),lev(data,'start_scan_to_end_scan_max','od
```

```
Ho: The sample start_scan_to_end_scan_max and od_delta have equal variance
Ha: The sample start_scan_to_end_scan_max and od_delta are unequal variance
stat=0.000, p_value=1.000
The sample start_scan_to_end_scan_max and od_delta have equal variance
Ho: The sample start_scan_to_end_scan_max and od_delta have equal variance
Ha: The sample start_scan_to_end_scan_max and od_delta are unequal variance
stat=0.000, p_value=1.000
The sample start_scan_to_end_scan_max and od_delta have equal variance
```

Out[101]:

```
(None, None)
```

In [102]:

```
mann(data,'start_scan_to_end_scan_max','od_delta'),will(data,'start_scan_to_end_scan_max','
```

```
Ho: The sum of ranking of start_scan_to_end_scan_max and od_delta are equal
Ha: The sum of ranking of start_scan_to_end_scan_max and od_delta are not eq
ual
Assumption of a Mann-Whitney U test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=347662080.500, p_value=1.000
The sum of ranking of start_scan_to_end_scan_max and od_delta are equal
Ho: The central tendencies of start_scan_to_end_scan_max and od_delta are eq
ual
Ha: The central tendencies of start_scan_to_end_scan_max and od_delta are no
t equal
Assumption of a Wilcoxon Signed-Rank Test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=347662080.500, p_value=1.000
The central tendencies of start_scan_to_end_scan_max and od_delta are equal
```

Out[102]:

```
(None, None)
```

'start_scan_to_end_scan_max','od_delta' variables have equal variances and medians

In [103]:

```
def conf(dff,para):
    per=[0.05,0.025,0.005]
    ls=[]
    for i in range(80000):
        ans=np.random.choice(dff[para],len(dff),replace=True)
        l1=np.mean(ans)
        ls.append(l1)
    print('Confidence interval using Bootstrap : ')
    print('Confidence interval for {} group is [{},{}] with {}% confidence'.format(para,np.
    print('Confidence interval for {} group is [{},{}] with {}% confidence'.format(para,np.
    print('Confidence interval for {} group is [{},{}] with {}% confidence'.format(para,np.
```

In [104]:

```
conf(data,'start_scan_to_end_scan_max')
```

```
Confidence interval using Bootstrap :
Confidence interval for start_scan_to_end_scan_max group is [293.82923508665
476,302.8041544996018] with 90.0% confidence
Confidence interval for start_scan_to_end_scan_max group is [292.97159353786
645,303.6652556031704] with 95.0% confidence
Confidence interval for start_scan_to_end_scan_max group is [291.34479236982
82,305.3795248208123] with 99.0% confidence
```

In [105]:

```
conf(data,'od_delta')
```

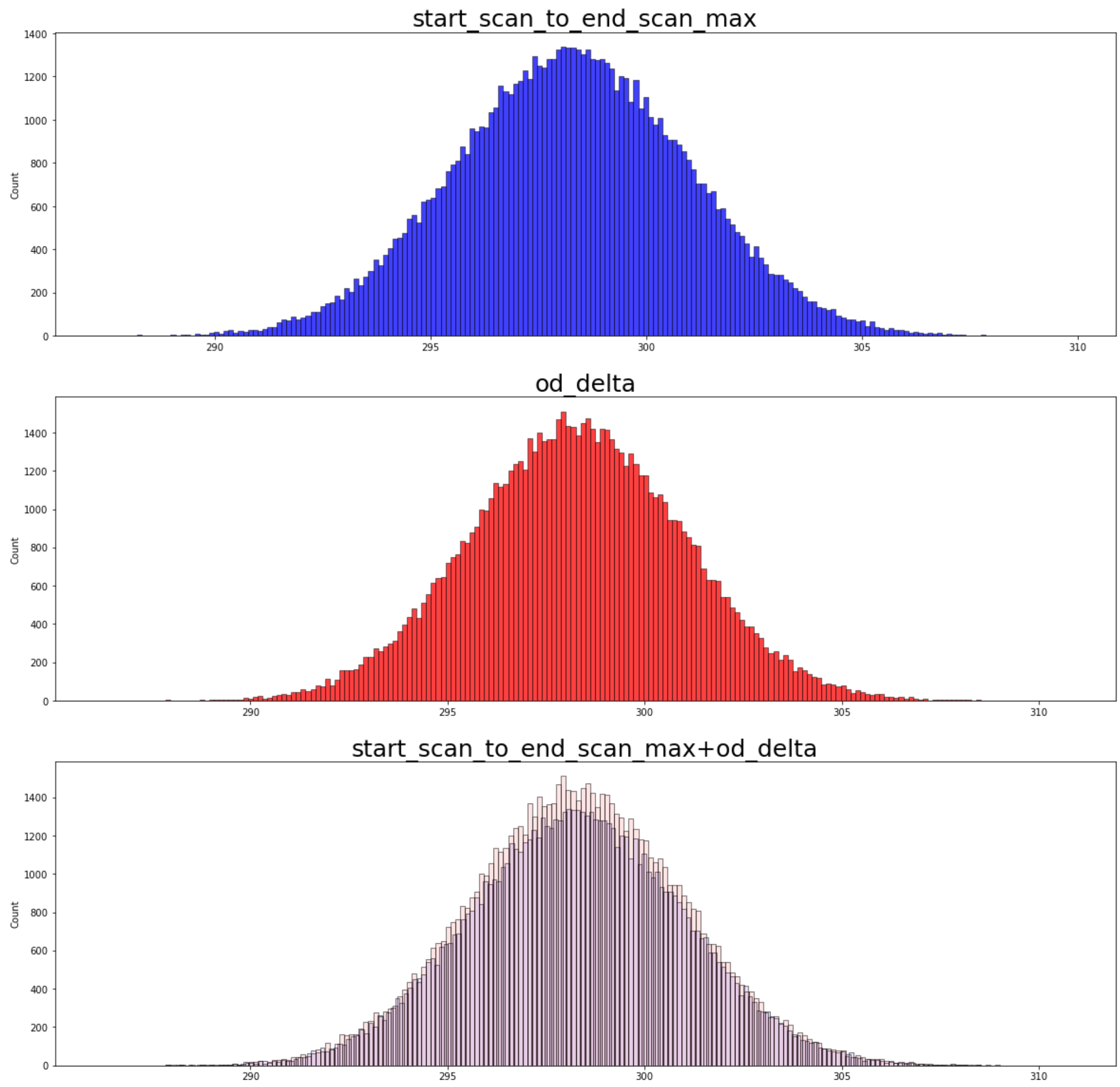
```
Confidence interval using Bootstrap :
Confidence interval for od_delta group is [293.8321077780727,302.79331222268
576] with 90.0% confidence
Confidence interval for od_delta group is [292.9574860631803,303.67361011035
69] with 95.0% confidence
Confidence interval for od_delta group is [291.3728742083507,305.36114831810
08] with 99.0% confidence
```

In [106]:

```
ls=[]
for i in range(80000):
    ans=np.random.choice(data['start_scan_to_end_scan_max'],len(data),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
fig, axes = plt.subplots(3, 1, figsize=(20, 20))
plt.subplot(3,1,1)
sns.histplot(ls,color='b',ax=axes[0],bins=200)
sns.histplot(ls,color='b',ax=axes[2],alpha=0.1,bins=200)
ls=[]
for i in range(80000):
    ans=np.random.choice(data['od_delta'],len(data),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
plt.subplot(3,1,2)
sns.histplot(ls,color='r',ax=axes[1],bins=200)
sns.histplot(ls,color='r',ax=axes[2],alpha=0.1,bins=200)
axes[0].set_title('start_scan_to_end_scan_max',fontsize=25)
axes[1].set_title('od_delta',fontsize=25)
axes[2].set_title('start_scan_to_end_scan_max+od_delta',fontsize=25)
# Confidence intervals are completely over lapping
```

Out[106]:

```
Text(0.5, 1.0, 'start_scan_to_end_scan_max+od_delta')
```



In [107]:

```
# Normality tests
"""
qqplot(dff,a),kstest(dff,a),shapiro(dff,a)
"""
# Transformations
"""
logtrans(dff,a), box(dff,a),rec(dff,a),sq(dff,a)
"""
# Corr
"""
pear(dff,a,b),spearmannr(dff,a,b),kend(dff,a,b)
"""

# Vari
"""
bar(dff,a,b),lev(dff,a,b)
"""
# not normal

"""
mann(dff,a,b),will(dff,a,b)
"""

# Ttest

"""
#ttest(dff,a,b),ttestv(dff,a,b),chis(dff,a,b),chis2(add,a,b,c),anova(dff,a,b,c),kwtest(dff,
"""
```

Out[107]:

```
'\n#ttest(dff,a,b),ttestv(dff,a,b),chis(dff,a,b),chis2(add,a,b,c),anova(dff,
a,b,c),kwtest(dff,a,b,c)\n'
```

In [108]:

```
data1=df.groupby(['trip_uuid'])\
['actual_time','osrm_time','osrm_distance','start_scan_to_end_scan','actual_distance_to_des
aggregate({'actual_time':['max','count'],
           'osrm_time':max,
           'osrm_distance':max,
           'start_scan_to_end_scan':max,
           'actual_distance_to_destination':['max','count'],
           'segment_actual_time':['sum','count'],
           'segment_osrm_time':['sum','count'],
           'segment_osrm_distance':['sum','count'],
           'cutoff_factor':['min','max','mean'],
           'segment_factor':['min','max','mean'],
          'factor':['min','max','mean']}).reset_index()

# Grouping with respect to 'trip_uuid'
```

In [109]:

```
data1.columns
```

Out[109]:

```
MultiIndex([(                     'trip_uuid',      ''),
            (                   'actual_time',   'max'),
            (                   'actual_time', 'count'),
            (                     'osrm_time',   'max'),
            (                 'osrm_distance',   'max'),
            (        'start_scan_to_end_scan',   'max'),
            ('actual_distance_to_destination',   'max'),
            ('actual_distance_to_destination', 'count'),
            (           'segment_actual_time',   'sum'),
            (           'segment_actual_time', 'count'),
            (             'segment_osrm_time',   'sum'),
            (             'segment_osrm_time', 'count'),
            (         'segment_osrm_distance',   'sum'),
            (         'segment_osrm_distance', 'count'),
            (                 'cutoff_factor',   'min'),
            (                 'cutoff_factor',   'max'),
            (                 'cutoff_factor',  'mean'),
            (                'segment_factor',   'min'),
            (                'segment_factor',   'max'),
            (                'segment_factor',  'mean'),
            (                        'factor',   'min'),
            (                        'factor',   'max'),
            (                        'factor',  'mean')],
           )
```

In [110]:

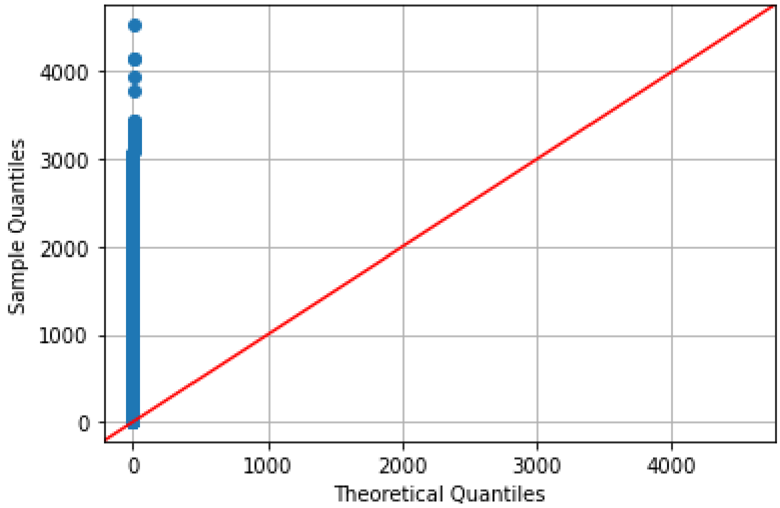
```
data1.columns = ['trip_uuid',
                     'actual_time_max','actual_time_count','osrm_time_max','osrm_distance_m
                     'actual_distance_to_destination_max','actual_distance_to_destination_c
                     'segment_actual_time_count','segment_osrm_time_sum','segment_osrm_time
                     'segment_osrm_distance_count','cutoff_factor_min','cutoff_factor_max',
                     'segment_factor_min','segment_factor_max','segment_factor_mean','facto
# Renaming columns
```

Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

In [111]:

```
qqplot(data1,'actual_time_max'),shapiro(data1,'actual_time_max'),kstest(data1,'actual_time_
```

```
This test is for visual only
```




```
Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution

stat=0.508, p_value=0.000
The sample actual_time_max does not follows normal distribution
Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution
stat=1.000, p_value=0.000

The sample actual_time_max does not follows normal distribution
```

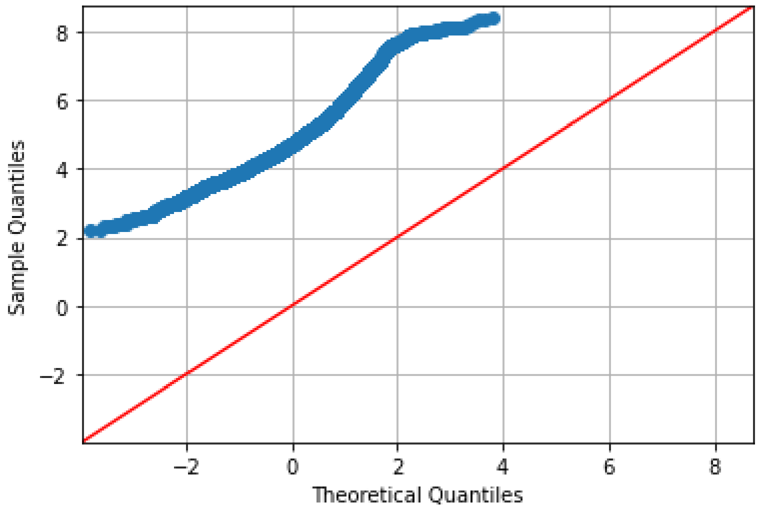
Out[111]:

```
(None, None, None)
```

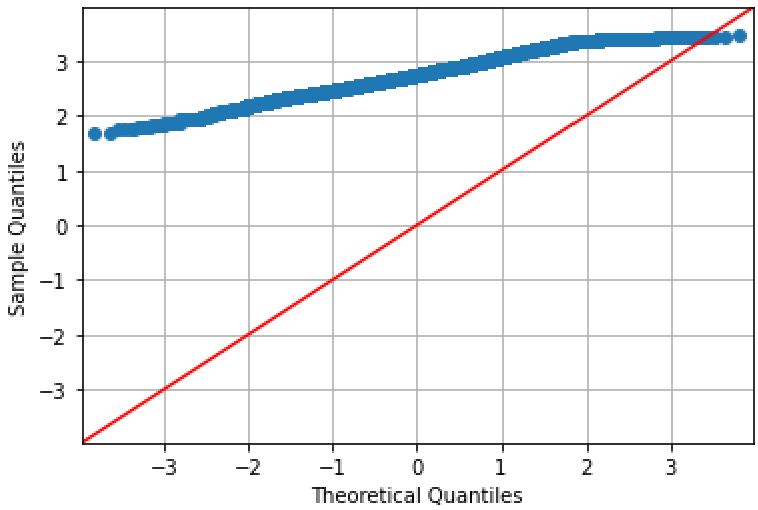
In [112]:

```
logtrans(data1,'actual_time_max'),box(data1,'actual_time_max')
```

```
After applying log transforms
This test is for visual only
```




```
Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution
stat=0.993, p_value=0.000

The sample actual_time_max does not follows normal distribution

Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution

stat=0.960, p_value=0.000
The sample actual_time_max does not follows normal distribution
After applying boxcox transforms
This test is for visual only
```




```
Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution
stat=0.972, p_value=0.000

The sample actual_time_max does not follows normal distribution

Ho: The sample actual_time_max follows normal distribution
Ha: The sample actual_time_max does not follows normal distribution
```

```
stat=0.995, p_value=0.000
The sample actual_time_max does not follows normal distribution
```

Out[112]:

```
(None, None)
```

In [113]:

```
bar(data1,'actual_time_max','osrm_time_max'),lev(data1,'actual_time_max','osrm_time_max')
```

```
Ho: The sample actual_time_max and osrm_time_max have equal variance
Ha: The sample actual_time_max and osrm_time_max are unequal variance
stat=7045.227, p_value=0.000
The sample actual_time_max and osrm_time_max unequal variance
Ho: The sample actual_time_max and osrm_time_max have equal variance
Ha: The sample actual_time_max and osrm_time_max are unequal variance
stat=793.627, p_value=0.000
The sample actual_time_max and osrm_time_max unequal variance
```

Out[113]:

```
(None, None)
```

In [114]:

```
mann(data1,'actual_time_max','osrm_time_max'),will(data1,'actual_time_max','osrm_time_max')
```

```
Ho: The sum of ranking of actual_time_max and osrm_time_max are equal
Ha: The sum of ranking of actual_time_max and osrm_time_max are not equal
Assumption of a Mann-Whitney U test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=161265200.000, p_value=0.000
The sum of ranking of actual_time_max and osrm_time_max are not equal
Ho: The central tendencies of actual_time_max and osrm_time_max are equal
Ha: The central tendencies of actual_time_max and osrm_time_max are not equa
l
Assumption of a Wilcoxon Signed-Rank Test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=161265200.000, p_value=0.000
The central tendencies of actual_time_max and osrm_time_max are not equal
```

Out[114]:

```
(None, None)
```

In [115]:

```
ttestv(data1,'actual_time_max','osrm_time_max')
```

```
Ho: The sample means of actual_time_max and osrm_time_max are equal
Ha: The sample means of actual_time_max and osrm_time_max are not equal
Assumption of a t_test are:
1.Observations of 2 samples are independent
2. Both samples are approx normal distribution
3. Population standard deviation is not available

stat=35.184, p_value=0.000
The sample means of actual_time_max and osrm_time_max are not equal
```

'actual_time_max','osrm_time_max' variables have unequal variances and sum of ranking is not equal

In [116]:

```
conf(data1,'actual_time_max')
```

```
Confidence interval using Bootstrap :
Confidence interval for actual_time_max group is [271.53482486333263,284.693
3286090302] with 90.0% confidence
Confidence interval for actual_time_max group is [270.3255028008369,285.9812
7826145645] with 95.0% confidence
Confidence interval for actual_time_max group is [267.9121033272592,288.4484
872106365] with 99.0% confidence
```

In [117]:

```
conf(data1,'osrm_time_max')
```

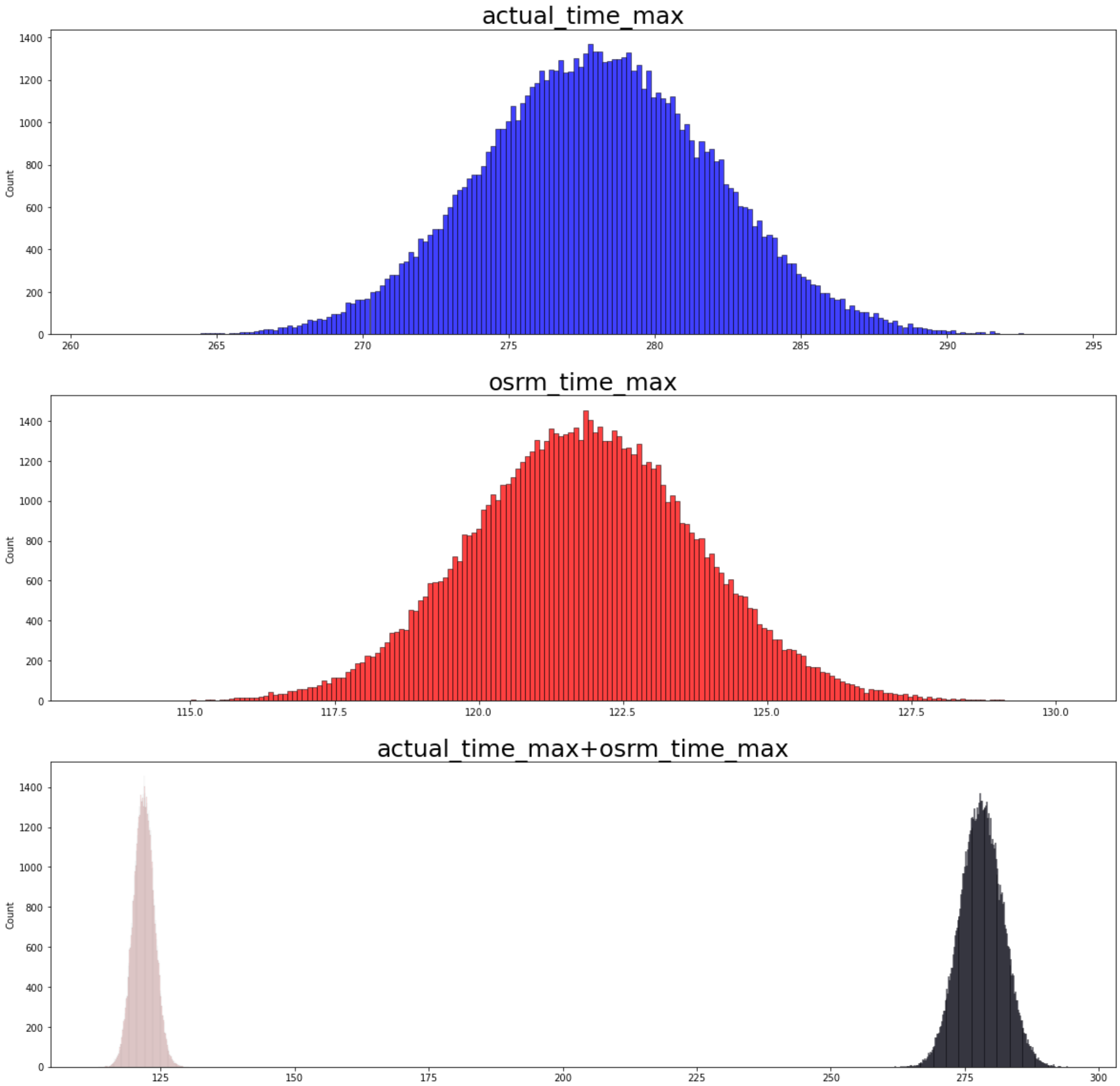
```
Confidence interval using Bootstrap :
Confidence interval for osrm_time_max group is [118.63285077951002,125.05352
29803604] with 90.0% confidence
Confidence interval for osrm_time_max group is [118.01193898899912,125.66876
560707296] with 95.0% confidence
Confidence interval for osrm_time_max group is [116.82883512181954,126.87886
076803672] with 99.0% confidence
```

In [118]:

```
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['actual_time_max'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
fig, axes = plt.subplots(3, 1, figsize=(20, 20))
plt.subplot(3,1,1)
sns.histplot(ls,color='b',ax=axes[0],bins=200)
sns.histplot(ls,color='b',ax=axes[2],alpha=0.1,bins=200)


ls=[]
for i in range(80000):
    ans=np.random.choice(data1['osrm_time_max'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
plt.subplot(3,1,2)
sns.histplot(ls,color='r',ax=axes[1],bins=200)
sns.histplot(ls,color='r',ax=axes[2],alpha=0.1,bins=200)
axes[0].set_title('actual_time_max',fontsize=25)
axes[1].set_title('osrm_time_max',fontsize=25)
axes[2].set_title('actual_time_max+osrm_time_max',fontsize=25)
```

Out[118]:

```
Text(0.5, 1.0, 'actual_time_max+osrm_time_max')
```



In [119]:

```
chis(data,'route_type','osrm_time_max')
# Osrm_time is independent od the route_type
```

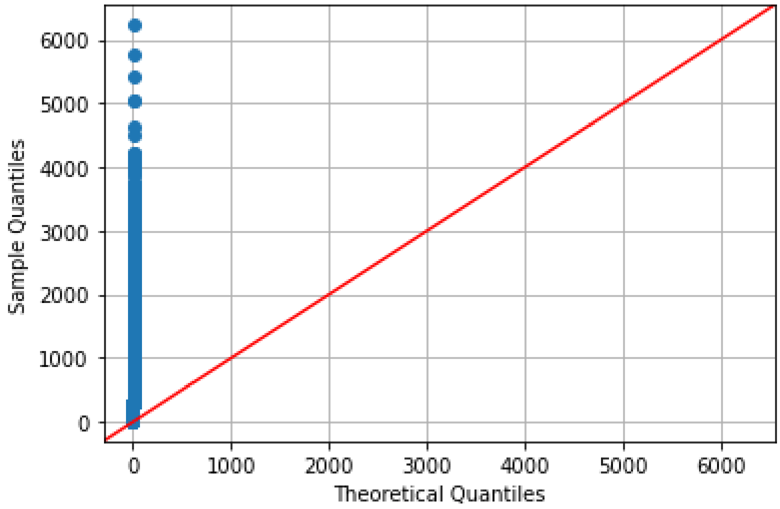
```
Ho: The samples are independent
Ha: The samples dependent
stat=0.000, p_value=1.000
The samples are independent
```

Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

In [120]:

```
qqplot(data1,'segment_actual_time_sum'),shapiro(data1,'segment_actual_time_sum'),kstest(dat
```

```
This test is for visual only
```




```
Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution

stat=0.582, p_value=0.000
The sample segment_actual_time_sum does not follows normal distribution
Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution
stat=1.000, p_value=0.000

The sample segment_actual_time_sum does not follows normal distribution
```

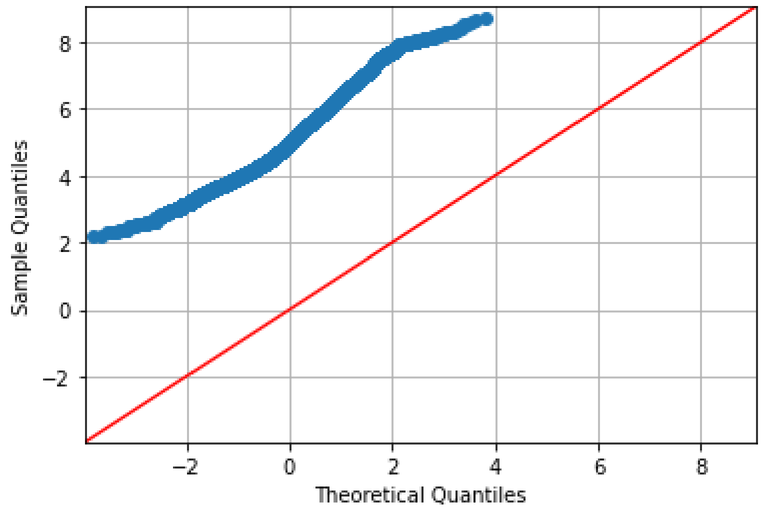
Out[120]:

```
(None, None, None)
```

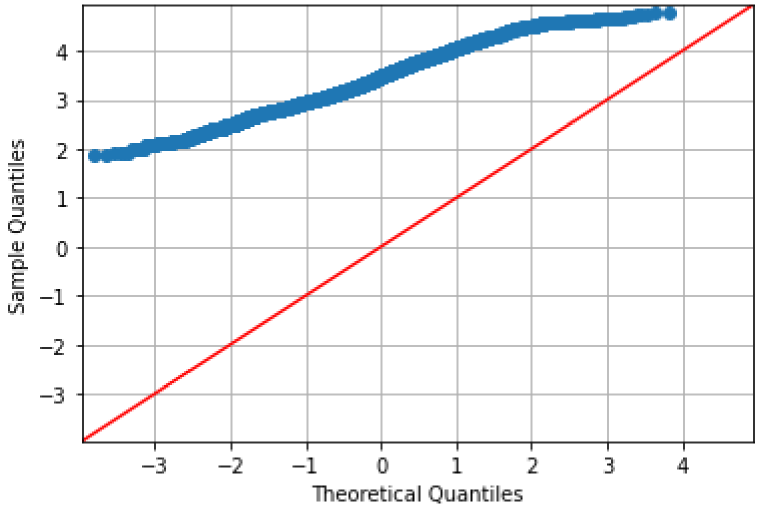
In [121]:

```
logtrans(data1,'segment_actual_time_sum'),box(data1,'segment_actual_time_sum')
```

```
After applying log transforms
This test is for visual only
```




```
Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution
stat=0.993, p_value=0.000

The sample segment_actual_time_sum does not follows normal distribution

Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution

stat=0.979, p_value=0.000
The sample segment_actual_time_sum does not follows normal distribution
After applying boxcox transforms
This test is for visual only
```




```
Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution
stat=0.982, p_value=0.000

The sample segment_actual_time_sum does not follows normal distribution

Ho: The sample segment_actual_time_sum follows normal distribution
Ha: The sample segment_actual_time_sum does not follows normal distribution

stat=0.991, p_value=0.000
The sample segment_actual_time_sum does not follows normal distribution
```

Out[121]:

```
(None, None)
```

In [122]:

```
bar(data1,'actual_time_max','segment_actual_time_sum'),lev(data1,'actual_time_max','segment
```

```
Ho: The sample actual_time_max and segment_actual_time_sum have equal varian
ce
Ha: The sample actual_time_max and segment_actual_time_sum are unequal varia
nce
stat=271.178, p_value=0.000
The sample actual_time_max and segment_actual_time_sum unequal variance
Ho: The sample actual_time_max and segment_actual_time_sum have equal varian
ce
Ha: The sample actual_time_max and segment_actual_time_sum are unequal varia
nce
stat=136.507, p_value=0.000
The sample actual_time_max and segment_actual_time_sum unequal variance
```

Out[122]:

```
(None, None)
```

In [123]:

```
mann(data1,'actual_time_max','segment_actual_time_sum'),will(data1,'actual_time_max','segme
```

```
Ho: The sum of ranking of actual_time_max and segment_actual_time_sum are eq
ual
Ha: The sum of ranking of actual_time_max and segment_actual_time_sum are no
t equal
Assumption of a Mann-Whitney U test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=97431867.000, p_value=0.000
The sum of ranking of actual_time_max and segment_actual_time_sum are not eq
ual
Ho: The central tendencies of actual_time_max and segment_actual_time_sum ar
e equal
Ha: The central tendencies of actual_time_max and segment_actual_time_sum ar
e not equal
Assumption of a Wilcoxon Signed-Rank Test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=97431867.000, p_value=0.000
The central tendencies of actual_time_max and segment_actual_time_sum are no
t equal
```

Out[123]:

```
(None, None)
```

'actual_time_max','segment_actual_time_sum' variables have unequal variances and sum of ranking is not equal

In [124]:

```
conf(data1,'actual_time_max')
```

```
Confidence interval using Bootstrap :
Confidence interval for actual_time_max group is [271.49096645744754,284.683
21185125194] with 90.0% confidence
Confidence interval for actual_time_max group is [270.3260393466964,285.9608
1359249513] with 95.0% confidence
Confidence interval for actual_time_max group is [267.9327637173517,288.3828
781129783] with 99.0% confidence
```

In [125]:

```
conf(data1,'segment_actual_time_sum')
```

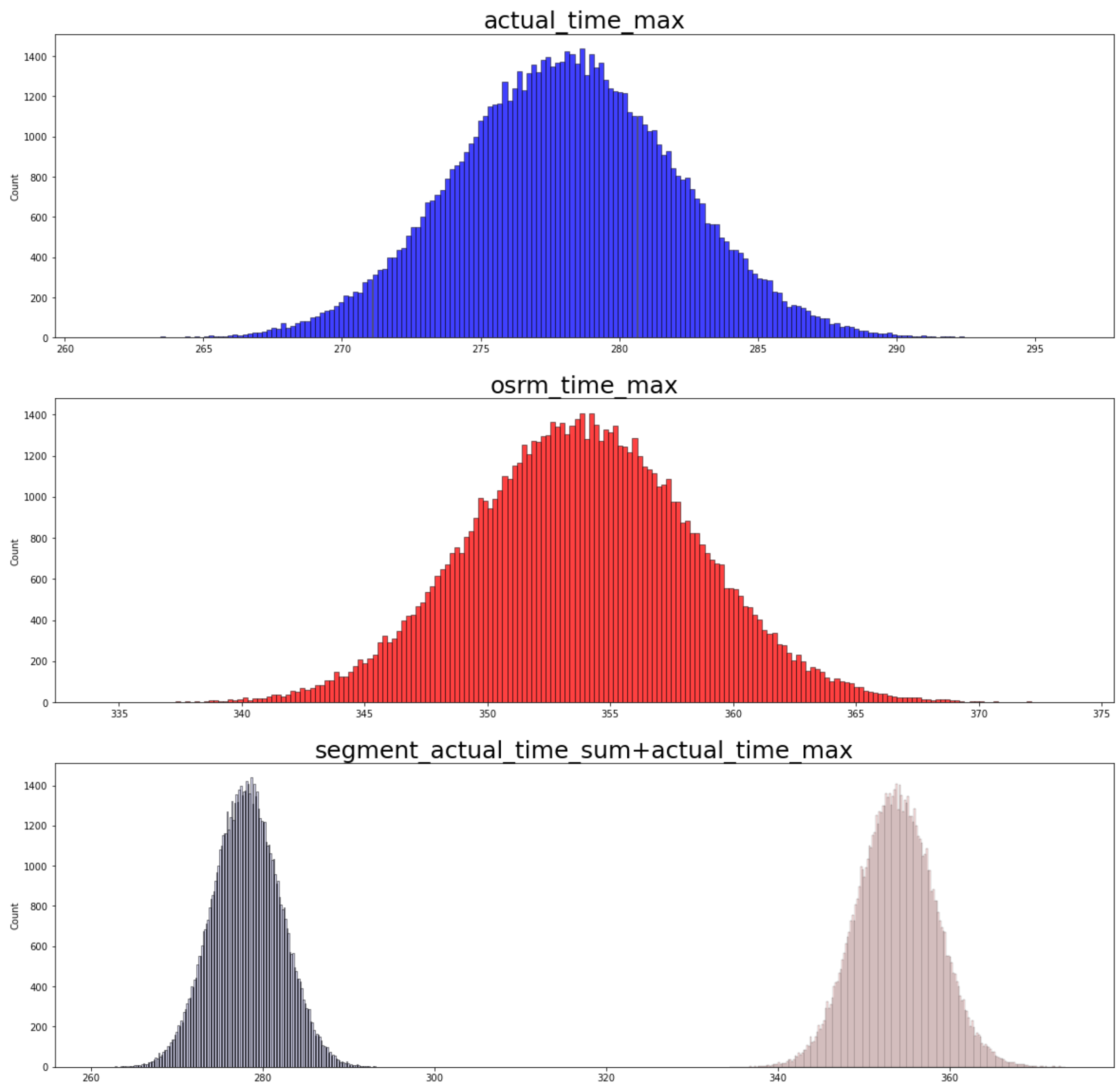
```
Confidence interval using Bootstrap :
Confidence interval for segment_actual_time_sum group is [346.4222413444017,
361.48789903489234] with 90.0% confidence
Confidence interval for segment_actual_time_sum group is [345.0410930012823,
362.952702976311] with 95.0% confidence
Confidence interval for segment_actual_time_sum group is [342.4143048525342
6,365.7565026658568] with 99.0% confidence
```

In [126]:

```
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['actual_time_max'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
fig, axes = plt.subplots(3, 1, figsize=(20, 20))
plt.subplot(3,1,1)
sns.histplot(ls,color='b',ax=axes[0],bins=200)
sns.histplot(ls,color='b',ax=axes[2],alpha=0.1,bins=200)
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['segment_actual_time_sum'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
plt.subplot(3,1,2)
sns.histplot(ls,color='r',ax=axes[1],bins=200)
sns.histplot(ls,color='r',ax=axes[2],alpha=0.1,bins=200)
axes[0].set_title('actual_time_max',fontsize=25)
axes[1].set_title('osrm_time_max',fontsize=25)
axes[2].set_title('segment_actual_time_sum+actual_time_max',fontsize=25)
```

Out[126]:

```
Text(0.5, 1.0, 'segment_actual_time_sum+actual_time_max')
```



In [127]:

```
chis(data,'route_type','actual_time_max')
# Actual_time is independent of route_type
```

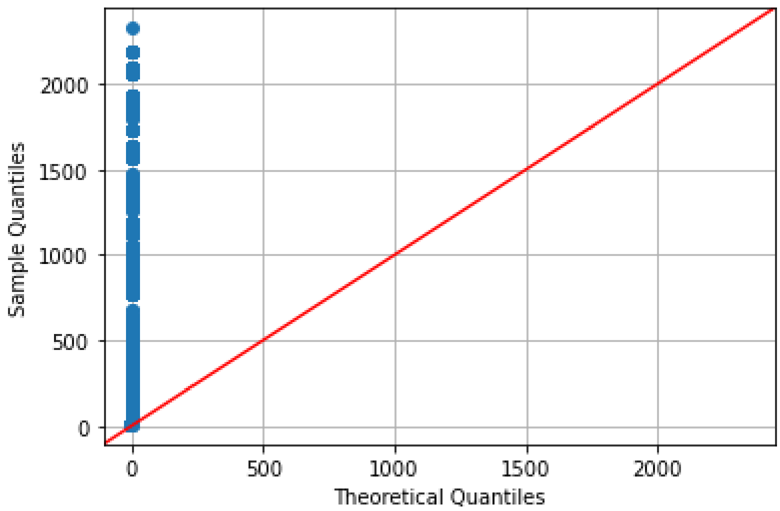
```
Ho: The samples are independent
Ha: The samples dependent
stat=0.000, p_value=1.000
The samples are independent
```

Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

In [128]:

```
qqplot(data1,'osrm_distance_max'),shapiro(data1,'osrm_distance_max'),kstest(data1,'osrm_dis
```

```
This test is for visual only
```




```
Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution

stat=0.438, p_value=0.000
The sample osrm_distance_max does not follows normal distribution
Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution
stat=1.000, p_value=0.000

The sample osrm_distance_max does not follows normal distribution
```

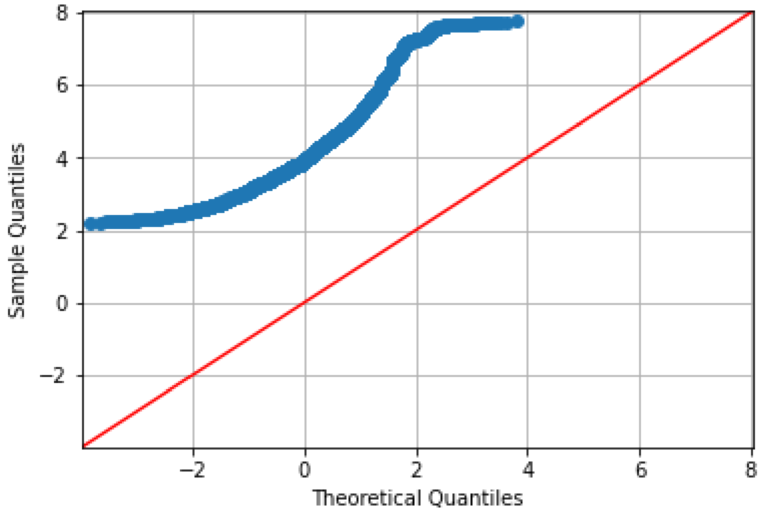
Out[128]:

```
(None, None, None)
```

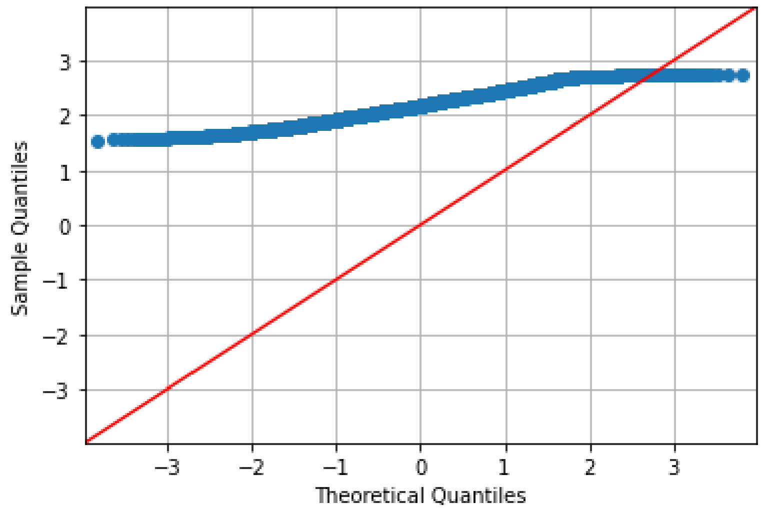
In [129]:

```
logtrans(data1,'osrm_distance_max'),box(data1,'osrm_distance_max')
```

```
After applying log transforms
This test is for visual only
```




```
Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution
stat=0.988, p_value=0.000

The sample osrm_distance_max does not follows normal distribution

Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution

stat=0.931, p_value=0.000
The sample osrm_distance_max does not follows normal distribution
After applying boxcox transforms
This test is for visual only
```




```
Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution
stat=0.943, p_value=0.000

The sample osrm_distance_max does not follows normal distribution

Ho: The sample osrm_distance_max follows normal distribution
Ha: The sample osrm_distance_max does not follows normal distribution

stat=0.991, p_value=0.000
The sample osrm_distance_max does not follows normal distribution
```

Out[129]:

(None, None)

In [130]:

```
bar(data1,'osrm_distance_max','segment_osrm_distance_sum'),lev(data1,'osrm_distance_max','s
```

```
Ho: The sample osrm_distance_max and segment_osrm_distance_sum have equal va
riance
Ha: The sample osrm_distance_max and segment_osrm_distance_sum are unequal v
ariance
stat=908.912, p_value=0.000
The sample osrm_distance_max and segment_osrm_distance_sum unequal variance
Ho: The sample osrm_distance_max and segment_osrm_distance_sum have equal va
riance
Ha: The sample osrm_distance_max and segment_osrm_distance_sum are unequal v
ariance
stat=213.482, p_value=0.000
The sample osrm_distance_max and segment_osrm_distance_sum unequal variance
```

Out[130]:

(None, None)

In [131]:

```
mann(data1,'osrm_distance_max','segment_osrm_distance_sum'),will(data1,'osrm_distance_max',
```

```
Ho: The sum of ranking of osrm_distance_max and segment_osrm_distance_sum ar
e equal
Ha: The sum of ranking of osrm_distance_max and segment_osrm_distance_sum ar
e not equal
Assumption of a Mann-Whitney U test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=92998656.000, p_value=0.000
The sum of ranking of osrm_distance_max and segment_osrm_distance_sum are no
t equal
Ho: The central tendencies of osrm_distance_max and segment_osrm_distance_su
m are equal
Ha: The central tendencies of osrm_distance_max and segment_osrm_distance_su
m are not equal
Assumption of a Wilcoxon Signed-Rank Test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=92998656.000, p_value=0.000
The central tendencies of osrm_distance_max and segment_osrm_distance_sum ar
e not equal
```

Out[131]:

```
(None, None)
```

'osrm_distance_max','segment_osrm_distance_sum' variables have unequal variances and sum of ranking is not equal

In [132]:

```
conf(data1,'osrm_distance_max')
```

```
Confidence interval using Bootstrap :
Confidence interval for osrm_distance_max group is [151.55387774482014,160.3
243622197476] with 90.0% confidence
Confidence interval for osrm_distance_max group is [150.74162718380913,161.1
9590147060808] with 95.0% confidence
Confidence interval for osrm_distance_max group is [149.17690481224267,162.8
401792659108] with 99.0% confidence
```

In [133]:

```
conf(data1,'segment_osrm_distance_sum')
```

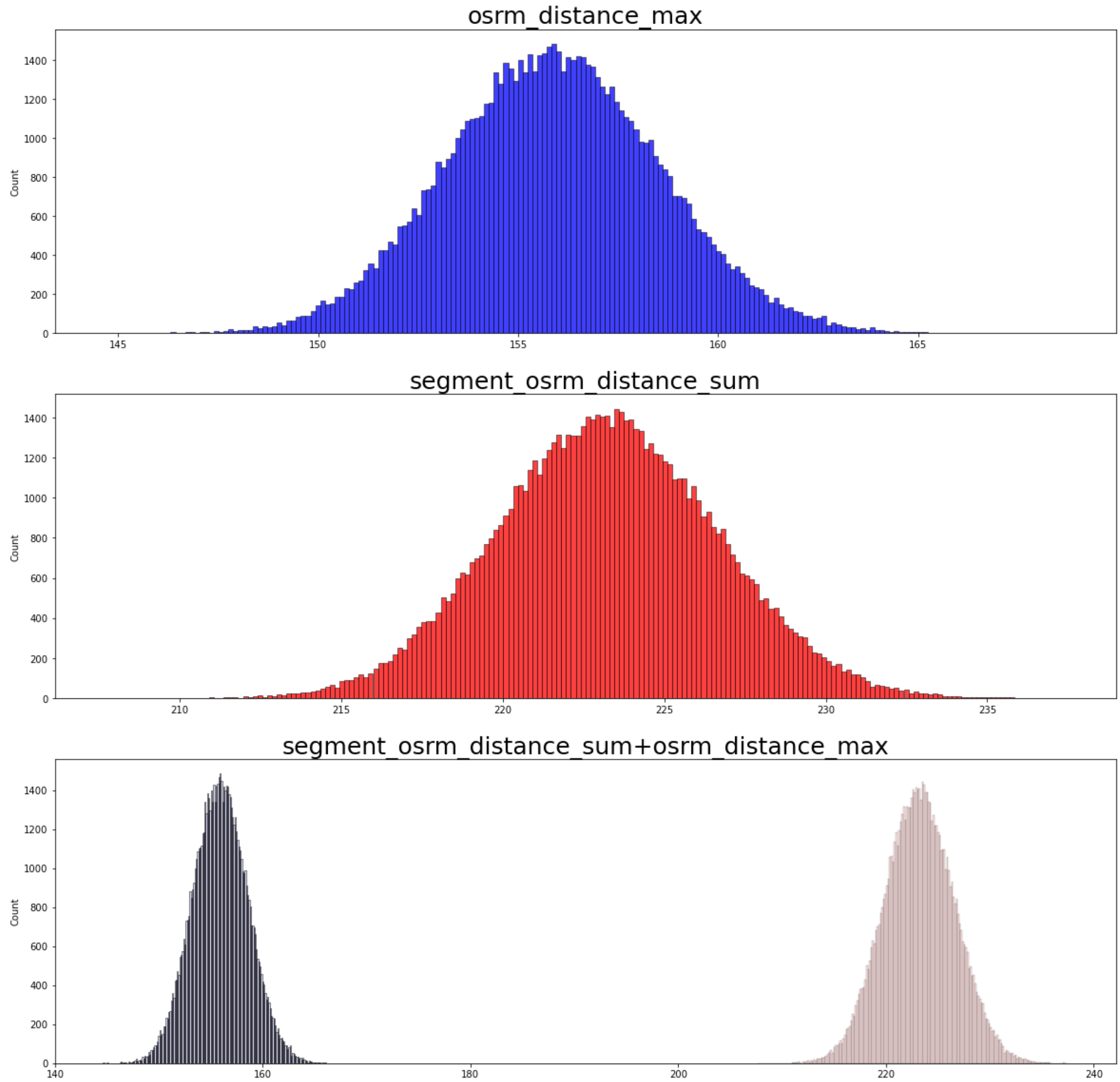
```
Confidence interval using Bootstrap :
Confidence interval for segment_osrm_distance_sum group is [217.599642192414
1,228.85409209489103] with 90.0% confidence
Confidence interval for segment_osrm_distance_sum group is [216.524832191064
35,229.9436023019167] with 95.0% confidence
Confidence interval for segment_osrm_distance_sum group is [214.474119496625
5,232.0903254569751] with 99.0% confidence
```

In [134]:

```
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['osrm_distance_max'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
fig, axes = plt.subplots(3, 1, figsize=(20, 20))
plt.subplot(3,1,1)
sns.histplot(ls,color='b',ax=axes[0],bins=200)
sns.histplot(ls,color='b',ax=axes[2],alpha=0.1,bins=200)
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['segment_osrm_distance_sum'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
plt.subplot(3,1,2)
sns.histplot(ls,color='r',ax=axes[1],bins=200)
sns.histplot(ls,color='r',ax=axes[2],alpha=0.1,bins=200)
axes[0].set_title('osrm_distance_max',fontsize=25)
axes[1].set_title('segment_osrm_distance_sum',fontsize=25)
axes[2].set_title('segment_osrm_distance_sum+osrm_distance_max',fontsize=25)
```

Out[134]:

```
Text(0.5, 1.0, 'segment_osrm_distance_sum+osrm_distance_max')
```



In [135]:

```
chis(data,'route_type','osrm_distance_max')
```

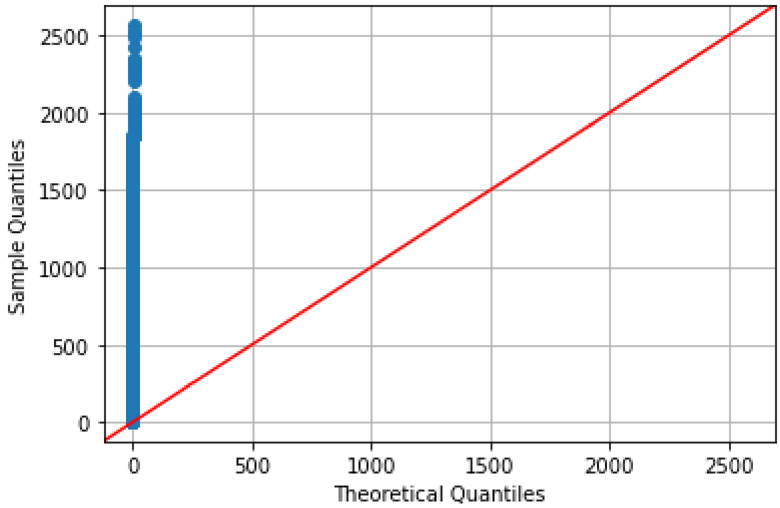
```
Ho: The samples are independent
Ha: The samples dependent
stat=0.000, p_value=1.000
The samples are independent
```

Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

In [136]:

```
qqplot(data1,'segment_osrm_time_sum'),shapiro(data1,'segment_osrm_time_sum'),kstest(data1,'
```

```
This test is for visual only
```




```
Ho: The sample segment_osrm_time_sum follows normal distribution
Ha: The sample segment_osrm_time_sum does not follows normal distribution

stat=0.533, p_value=0.000
The sample segment_osrm_time_sum does not follows normal distribution
Ho: The sample segment_osrm_time_sum follows normal distribution
Ha: The sample segment_osrm_time_sum does not follows normal distribution
stat=1.000, p_value=0.000

The sample segment_osrm_time_sum does not follows normal distribution
```

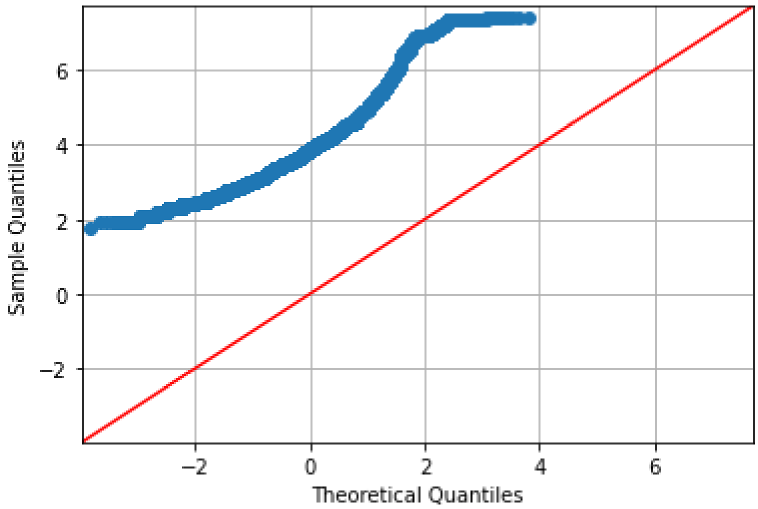
Out[136]:

```
(None, None, None)
```

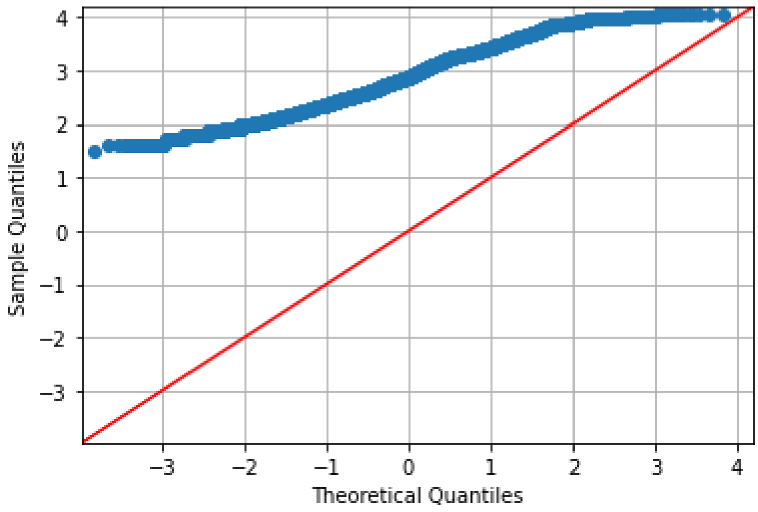
In [137]:

```
logtrans(data1,'osrm_time_max'),box(data1,'segment_osrm_time_sum')
```

```
After applying log transforms
This test is for visual only
```




```
Ho: The sample osrm_time_max follows normal distribution
Ha: The sample osrm_time_max does not follows normal distribution
stat=0.983, p_value=0.000

The sample osrm_time_max does not follows normal distribution

Ho: The sample osrm_time_max follows normal distribution
Ha: The sample osrm_time_max does not follows normal distribution

stat=0.938, p_value=0.000
The sample osrm_time_max does not follows normal distribution
After applying boxcox transforms
This test is for visual only
```




```
Ho: The sample segment_osrm_time_sum follows normal distribution
Ha: The sample segment_osrm_time_sum does not follows normal distribution
stat=0.961, p_value=0.000

The sample segment_osrm_time_sum does not follows normal distribution

Ho: The sample segment_osrm_time_sum follows normal distribution
Ha: The sample segment_osrm_time_sum does not follows normal distribution

stat=0.987, p_value=0.000
The sample segment_osrm_time_sum does not follows normal distribution
```

Out[137]:

(None, None)

In [138]:

```
bar(data1,'osrm_time_max', 'segment_osrm_time_sum'),lev(data1,'osrm_time_max', 'segment_osr
```

```
Ho: The sample osrm_time_max and segment_osrm_time_sum have equal variance
Ha: The sample osrm_time_max and segment_osrm_time_sum are unequal variance
stat=1167.884, p_value=0.000
The sample osrm_time_max and segment_osrm_time_sum unequal variance
Ho: The sample osrm_time_max and segment_osrm_time_sum have equal variance
Ha: The sample osrm_time_max and segment_osrm_time_sum are unequal variance
stat=293.918, p_value=0.000
The sample osrm_time_max and segment_osrm_time_sum unequal variance
```

Out[138]:

(None, None)

In [139]:

```
mann(data1,'osrm_time_max', 'segment_osrm_time_sum'),will(data1,'osrm_time_max', 'segment_o
```

```
Ho: The sum of ranking of osrm_time_max and segment_osrm_time_sum are equal
Ha: The sum of ranking of osrm_time_max and segment_osrm_time_sum are not eq
ual
Assumption of a Mann-Whitney U test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=91549541.500, p_value=0.000
The sum of ranking of osrm_time_max and segment_osrm_time_sum are not equal
Ho: The central tendencies of osrm_time_max and segment_osrm_time_sum are eq
ual
Ha: The central tendencies of osrm_time_max and segment_osrm_time_sum are no
t equal
Assumption of a Wilcoxon Signed-Rank Test are:
1.Should have a ordinal variable
2. Only 2 independent random samples with a least ordinally scales character
istics


stat=91549541.500, p_value=0.000
The central tendencies of osrm_time_max and segment_osrm_time_sum are not eq
ual
```

Out[139]:

```
(None, None)
```

'osrm_time_max', 'segment_osrm_time_sum' variables have unequal variances and sum of ranking is not equal

In [140]:

```
conf(data1,'osrm_time_max')
```

```
Confidence interval using Bootstrap :
Confidence interval for osrm_time_max group is [118.63244246473644,125.04394
951744617] with 90.0% confidence
Confidence interval for osrm_time_max group is [118.02358777080381,125.65985
860835526] with 95.0% confidence
Confidence interval for osrm_time_max group is [116.87471890396166,126.93677
971249242] with 99.0% confidence
```

In [141]:

```
conf(data1,'segment_osrm_time_sum')
```

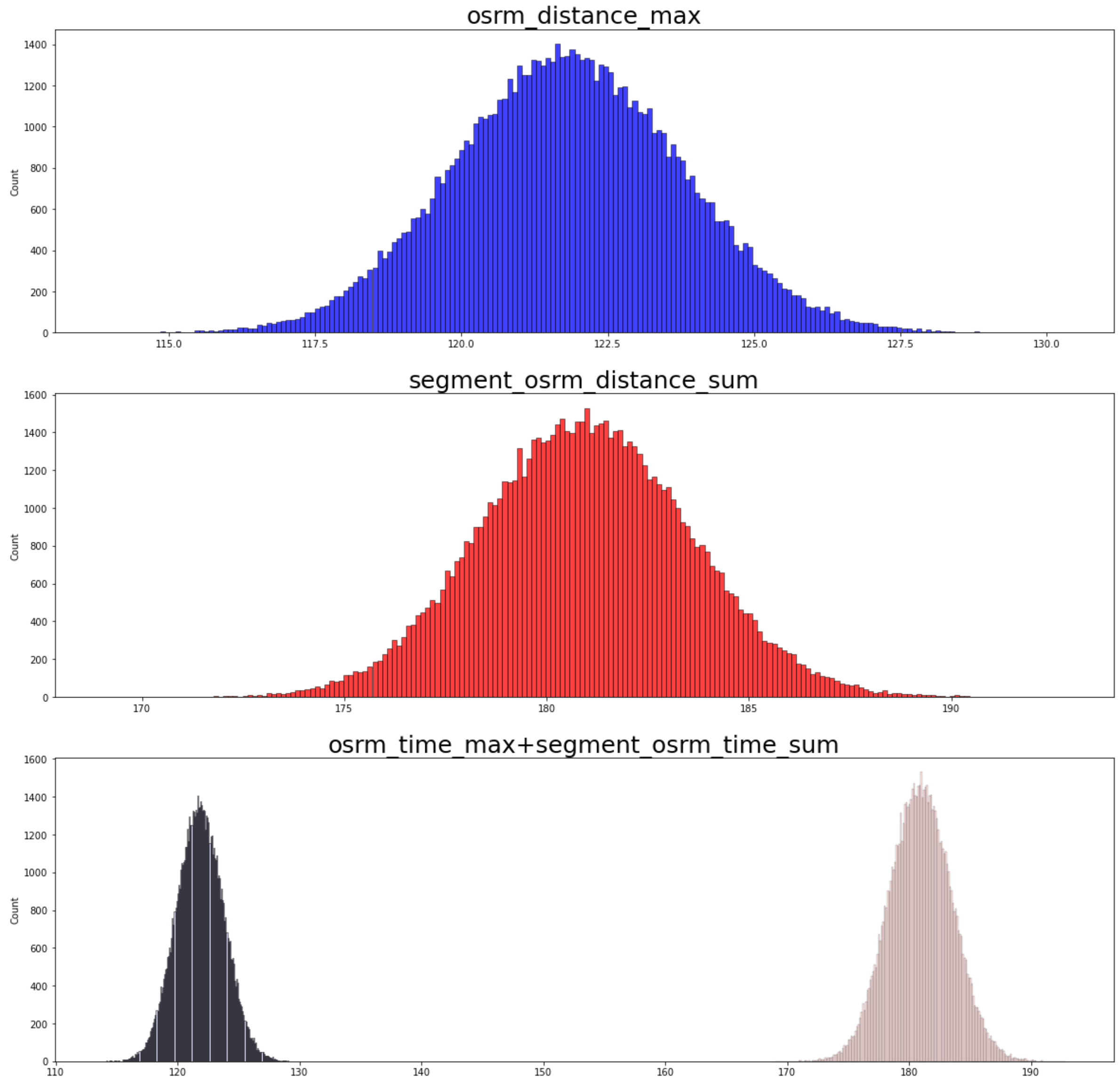
```
Confidence interval using Bootstrap :
Confidence interval for segment_osrm_time_sum group is [176.69271782412093,1
85.2069346021462] with 90.0% confidence
Confidence interval for segment_osrm_time_sum group is [175.9090909090909,18
6.02674124316664] with 95.0% confidence
Confidence interval for segment_osrm_time_sum group is [174.3151720996153,18
7.72465310116758] with 99.0% confidence
```

In [142]:

```
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['osrm_time_max'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
fig, axes = plt.subplots(3, 1, figsize=(20, 20))
plt.subplot(3,1,1)
sns.histplot(ls,color='b',ax=axes[0],bins=200)
sns.histplot(ls,color='b',ax=axes[2],alpha=0.1,bins=200)
ls=[]
for i in range(80000):
    ans=np.random.choice(data1['segment_osrm_time_sum'],len(data1),replace=True)
    l1=np.mean(ans)
    ls.append(l1)
plt.subplot(3,1,2)
sns.histplot(ls,color='r',ax=axes[1],bins=200)
sns.histplot(ls,color='r',ax=axes[2],alpha=0.1,bins=200)
axes[0].set_title('osrm_distance_max',fontsize=25)
axes[1].set_title('segment_osrm_distance_sum',fontsize=25)
axes[2].set_title('osrm_time_max+segment_osrm_time_sum',fontsize=25)
```

Out[142]:

```
Text(0.5, 1.0, 'osrm_time_max+segment_osrm_time_sum')
```



In [143]:

```
chis(data,'route_type','segment_osrm_time_sum')
# 'segment_osrm_time' in independent of route_type
```

```
Ho: The samples are independent
Ha: The samples dependent
stat=0.000, p_value=1.000
The samples are independent
```

In [144]:

```
kwtest(data1,'osrm_time_max','segment_osrm_time_sum','segment_actual_time_sum','actual_time
# Sample medians for 'osrm_time_max','segment_osrm_time_sum','segment_actual_time_sum','act
```

```
Assumptions of Kruskal-Wallis one-way analysis of variance are:
Ho: The sample median equal
Ha: There exists atleast one sample that is not equal to other median

stat=8104.760, p_value=0.000
The sample medians are not equal
```

In [145]:

```
kwtest1(data1,'segment_osrm_distance_sum','osrm_distance_max','actual_distance_to_destinati
# Sample medians for 'segment_osrm_distance_sum','osrm_distance_max','actual_distance_to_de
```

```
Assumptions of Kruskal-Wallis one-way analysis of variance are:
Ho: The sample median equal
Ha: There exists atleast one sample that is not equal to other median

stat=1855.120, p_value=0.000
The sample medians are not equal
```

In [146]:

```
data1.columns
```

Out[146]:

```
Index(['trip_uuid', 'actual_time_max', 'actual_time_count', 'osrm_time_max',
       'osrm_distance_max', 'start_scan_to_end_scan_max',
       'actual_distance_to_destination_max',
       'actual_distance_to_destination_count', 'segment_actual_time_sum',
       'segment_actual_time_count', 'segment_osrm_time_sum',
       'segment_osrm_time_count', 'segment_osrm_distance_sum',
       'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_ma
x',
       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
       'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean'],
      dtype='object')
```

In [147]:

```
encoder = OneHotEncoder(handle_unknown='ignore')
encoder_df = pd.DataFrame(encoder.fit_transform(data[['route_type']]).toarray())
data = data.join(encoder_df)
# One hot encoding for 'route_type'
```

In [148]:

```
data.rename({0:'Carting',1:'FTL'},axis=1,inplace=True)
```

In [149]:

```
data.head(10)
```

Out[149]:

	data	trip_creation_time	trip_uuid	route_type	od_start_time	od_end_time	s
0	test	2018-09-27 00:02:18.970980	trip-153800653897073708	Carting	2018-09-27 00:02:18.970980	2018-09-27 02:28:08.036773	
1	test	2018-09-27 00:02:29.352390	trip-153800654935210748	Carting	2018-09-27 00:02:29.352390	2018-09-27 01:23:35.904326	
2	test	2018-09-27 00:03:08.209931	trip-153800658820968126	FTL	2018-09-27 00:03:08.209931	2018-09-27 10:13:54.663547	
3	test	2018-09-27 00:03:14.680535	trip-153800659468028518	Carting	2018-09-27 02:37:15.362086	2018-09-27 04:21:45.871140	
4	test	2018-09-27 00:03:37.296972	trip-153800661729668086	Carting	2018-09-27 02:13:23.273586	2018-09-27 06:02:15.316360	
5	test	2018-09-27 00:03:40.279575	trip-153800662027930085	Carting	2018-09-27 02:21:57.981325	2018-09-27 03:47:52.253700	
6	test	2018-09-27 00:04:53.018925	trip-153800669301861431	Carting	2018-09-27 01:15:10.565535	2018-09-27 02:06:37.053884	
7	test	2018-09-27 00:04:59.087065	trip-153800669908677971	Carting	2018-09-27 00:04:59.087065	2018-09-27 01:37:39.311290	
8	test	2018-09-27 00:04:59.087065	trip-153800669908677971	Carting	2018-09-27 02:28:04.724867	2018-09-27 03:37:15.481679	
9	test	2018-09-27 00:08:02.763752	trip-153800688276350851	Carting	2018-09-27 00:08:02.763752	2018-09-27 02:39:33.933359	

10 rows × 58 columns

In [150]:

```
obj=list(data.columns[[data.dtypes=='object']])
cat=list(data.columns[[data.dtypes=='category']])
```

In [151]:

```
data2=data.copy('deep')
```

In [152]:

```
data2.drop(obj,axis=1,inplace=True)
data2.drop(cat,axis=1,inplace=True)
```

In [153]:

```
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(data2.to_numpy())
df_scaled = pd.DataFrame(df_scaled, columns=data2.columns)
# # MinMax for ['actual_time_max', 'actual_time_count', 'osrm_time_max',
#        'osrm_distance_max', 'start_scan_to_end_scan_max',
#        'actual_distance_to_destination_max',
#        'actual_distance_to_destination_count', 'segment_actual_time_sum',
#        'segment_actual_time_count', 'segment_osrm_time_sum',
#        'segment_osrm_time_count', 'segment_osrm_distance_sum',
#        'segment_osrm_distance_count', 'cutoff_factor_min', 'cutoff_factor_max',
#       'cutoff_factor_mean', 'segment_factor_min', 'segment_factor_max',
#        'segment_factor_mean', 'factor_min', 'factor_max', 'factor_mean',
#        'trip_creation_year', 'trip_creation_month', 'trip_creation_day',
#        'od_start_year', 'od_start_month', 'od_start_day', 'od_end_year',
#        'od_end_month', 'od_end_day', 'od_delta', 'actual_time_avg',
#        'osrm_time_avg', 'osrm_distance_avg',
#        'actual_distance_to_destination_avg', 'segment_actual_time_avg',
#        'segment_osrm_time_avg', 'segment_osrm_distance_count_avg', 'Carting',
#        'FTL']
```

In [154]:

```
df_scaled
```

Out[154]:

	actual_time_max	actual_time_count	osrm_time_max	osrm_distance_max	start_scan_to_
0	0.026310	0.0750	0.020833	0.020918	
1	0.010834	0.0125	0.006548	0.005463	
2	0.071855	0.0000	0.093452	0.091393	
3	0.003980	0.0000	0.001786	0.001071	
4	0.027194	0.0125	0.006548	0.003932	
...	...	...	...	...	
26364	0.011718	0.0125	0.017262	0.009293	
26365	0.056821	0.0375	0.075000	0.056308	
26366	0.011718	0.0375	0.017857	0.009598	
26367	0.007296	0.0125	0.007143	0.002556	
26368	0.003759	0.0125	0.004762	0.001658	

26369 rows × 41 columns

In [157]:

```
df.columns
```

Out[157]:

```
Index(['data', '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', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor',
       'source_center_pincode', 'destination_center_pincode', 'Source_City',
       'Destination_City', 'Source_State', 'Destination_State'],
      dtype='object')
```

In [158]:

```
df[['segment_osrm_time','segment_actual_time']]
```

Out[158]:

	segment_osrm_time	segment_actual_time
0	11.0	14.0
1	9.0	10.0
2	7.0	16.0
3	12.0	21.0
4	5.0	6.0
...	...	...
144862	12.0	12.0
144863	21.0	26.0
144864	34.0	20.0
144865	27.0	17.0
144866	9.0	268.0

144867 rows × 2 columns

In [ ]:

```

```