In [531]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import shapiro
from scipy.stats import boxcox
import statsmodels.api as sm
from scipy.stats import ttest_ind
```

In [118]:
```
df = pd.read_csv('delhivery_data.csv')
df.head()
```

Out[118]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

5 rows × 24 columns

In [119]:
```
df.shape
```

Out[119]:
```
(144867, 24)
```

In [120]:
```
df.size
```

Out[120]:
```
3476808
```

In [121]:
```
1 df.columns
```

Out[121]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_typ
e',
       'trip_uuid', 'source_center', 'source_name', 'destination_cente
r',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_tim
e',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

In [122]:
```
1 df.info()
```

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

Let's Drop the columns "Unknown"

In [123]:
```
unknown = ['segment_factor', 'factor','cutoff_timestamp','cutoff_fact
```

In [124]:
```
df.drop(columns=unknown, inplace=True)
```

In [125]:
```
df.shape
```

Out[125]: (144867, 19)

In [126]:
```
df[:2]
```

Out[126]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

Unqiue Values of Data

In [127]:
```
df["data"].nunique() , df["data"].unique()
```

Out[127]: (2, array(['training', 'test'], dtype=object))

In [128]:
```
for i in df.columns:
    print(f"The unique elements in the columns are :: {i} ----->  {df
```

```
The unique elements in the columns are :: data ----->  2
The unique elements in the columns are :: trip_creation_time ----->  148
17
The unique elements in the columns are :: route_schedule_uuid ----->  15
04
The unique elements in the columns are :: route_type ----->  2
The unique elements in the columns are :: trip_uuid ----->  14817
The unique elements in the columns are :: source_center ----->  1508
The unique elements in the columns are :: source_name ----->  1498
The unique elements in the columns are :: destination_center ----->  148
1
The unique elements in the columns are :: destination_name ----->  1468
The unique elements in the columns are :: od_start_time ----->  26369
The unique elements in the columns are :: od_end_time ----->  26369
The unique elements in the columns are :: start_scan_to_end_scan ----->
1915
The unique elements in the columns are :: actual_distance_to_destination
----->  144515
The unique elements in the columns are :: actual_time ----->  3182
The unique elements in the columns are :: osrm_time ----->  1531
The unique elements in the columns are :: osrm_distance ----->  138046
The unique elements in the columns are :: segment_actual_time ----->  74
7
The unique elements in the columns are :: segment_osrm_time ----->  214
The unique elements in the columns are :: segment_osrm_distance ----->
113799
```

In [129]:
```
df.isna().sum()
```

Out[129]:
```
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
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
dtype: int64
```

In [130]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 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  actual_distance_to_destination   144867 non-null  float64
 13  actual_time                      144867 non-null  float64
 14  osrm_time                        144867 non-null  float64
 15  osrm_distance                    144867 non-null  float64
 16  segment_actual_time              144867 non-null  float64
 17  segment_osrm_time                144867 non-null  float64
 18  segment_osrm_distance            144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

In [131]:
```
1 df.describe()
```

Out[131]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_tim
count	144867.000000	144867.000000	144867.000000	144867.00000
mean	961.262986	234.073372	416.927527	213.86827
std	1037.012769	344.990009	598.103621	308.01108
min	20.000000	9.000045	9.000000	6.00000
25%	161.000000	23.355874	51.000000	27.00000
50%	449.000000	66.126571	132.000000	64.00000
75%	1634.000000	286.708875	513.000000	257.00000
max	7898.000000	1927.447705	4532.000000	1686.00000

In [132]:
```
df.describe(include='object')
```

Out[132]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
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
freq	104858	101	1812	99660	101

In [133]:
```
df['data'] = df['data'].astype('category')
df['route_type'] = df['route_type'].astype('category')
```

In [134]:
```
df[:1]
```

Out[134]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

In [135]:
```
df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'])
df['od_start_time'] = pd.to_datetime(df['od_start_time'])
df['od_end_time'] = pd.to_datetime(df['od_end_time'])
```

Let's check for the Data start time and end time

In [136]:
```
df['trip_creation_time'].min(), df['trip_creation_time'].max()
```

Out[136]:
```
(Timestamp('2018-09-12 00:00:16.535741'),
 Timestamp('2018-10-03 23:59:42.701692'))
```

In [137]:
```
df['od_start_time'].min() , df['od_end_time'].max()
```

Out[137]:
```
(Timestamp('2018-09-12 00:00:16.535741'),
 Timestamp('2018-10-08 03:00:24.353479'))
```

Data Cleaning and Exploration

In [138]:
```
1 df[:2]
```

Out[138]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

In [139]:
```
1 df.isna().sum()
```

Out[139]:
```
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
actual_distance_to_destination  0
actual_time                     0
osrm_time                       0
osrm_distance                   0
segment_actual_time             0
segment_osrm_time               0
segment_osrm_distance           0
dtype: int64
```

In [140]:
```
1 missing = df.loc[df['source_name'].isnull(), 'source_center'].unique(
2 missing
```

Out[140]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [141]:
```
1 df.isna().sum()
```

Out[141]:
```
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
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
dtype: int64
```

In [142]:
```
1 df[['source_center','source_name']].isna().sum()
```

Out[142]:
```
source_center      0
source_name      293
dtype: int64
```

In [143]:
```
1 df[df['source_name'].isna()][:5]
```

Out[143]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	s
112	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756691	IN
113	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756691	IN
114	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756691	IN
115	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756691	IN
116	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756691	IN

Let's check for source center value for which source name is null

In [144]:
```
df[(df["source_name"].notnull()) & (df["source_center"].isin(df[df["s
```

Out[144]:

data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	sourc

In [145]:
```
df[['destination_name','destination_center']].isna().sum()
```

Out[145]:
```
destination_name      261
destination_center      0
dtype: int64
```

Let's check for the destination center value for which destination name is null

In [146]:
```
df[(df["destination_name"].notnull()) & (df["destination_center"].isi
```

Out[146]:

data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	sourc

We noted that from the describe the "segment_actual_time" is in negative, so the time cannot be in Negative, We Eliminate this

In [147]:

```
1 df[df['segment_actual_time'] < 0]
```

Out[147]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
1805	training	2018-09-14 18:24:50.326548	thanos::sroute:21a9cdbc-469e-40e7-8fda-f4c02a1...	Carting	trip-153694949570782725
3761	training	2018-09-24 02:51:40.385840	thanos::sroute:f01c8bbd-655d-42ea-9abf-60d5040...	FTL	trip-153775750038541173
4040	test	2018-10-01 15:23:08.328308	thanos::sroute:749c5c42-2826-40c9-b0d3-5102cee...	Carting	trip-153840738832805377
6329	training	2018-09-24 06:45:30.168184	thanos::sroute:b43ac2fc-3ec6-4a37-adf2-00b2561...	Carting	trip-153777153016792753
39825	training	2018-09-23 22:33:30.538708	thanos::sroute:54a7c356-361d-4e74-9ee7-d420c37...	FTL	trip-153774201053841850
40942	training	2018-09-24 01:49:00.396529	thanos::sroute:e3e7c92f-55a9-4ecb-a4a7-919bbb7...	Carting	trip-153775374039614677
56464	training	2018-09-24 04:05:35.796451	thanos::sroute:870bead8-6c8a-458f-b4d8-658de44...	FTL	trip-153776193579607660
58697	training	2018-09-24 19:41:53.611094	thanos::sroute:cf2e63a5-87d6-4e39-a2d6-5555ed6...	FTL	trip-153781811361072511
68205	test	2018-10-03 22:28:20.454881	thanos::sroute:870bead8-6c8a-458f-b4d8-658de44...	FTL	trip-153860570045461434
70479	training	2018-09-14 09:54:59.693367	thanos::sroute:3f5f6285-256d-48ea-9a07-ec85cfe...	FTL	trip-153691889969297161
73603	training	2018-09-14 09:26:48.853918	thanos::sroute:f9784448-ffd3-4a4c-957f-b7fa68c...	FTL	trip-153691720885367295
85042	training	2018-09-13 01:48:43.812361	thanos::sroute:2994b7ae-bb9c-4d7c-a30f-7882553...	FTL	trip-153680332381212407
95433	training	2018-09-24 23:50:47.637174	thanos::sroute:54a7c356-361d-4e74-9ee7-d420c37...	FTL	trip-153783304763693286
97566	training	2018-09-24 18:18:00.311575	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153781308031132891
100205	training	2018-09-17 08:05:40.886155	thanos::sroute:06b37046-b91b-4f4d-9aac-e799aa0...	Carting	trip-153717154088593859
116173	training	2018-09-22 04:36:54.081516	thanos::sroute:fe14dda1-e713-451e-9bd3-d27c920...	Carting	trip-153759101408126548
119377	training	2018-09-12 08:51:23.771627	thanos::sroute:6a74dc96-d3ef-4697-9ae8-8eab11c...	Carting	trip-153674228377135882

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
120153	test	2018-09-27 01:18:15.793512	thanos::sroute:1f3b54e4-3374-43fd-8ed1-198bb9a...	Carting	trip-153801109579320959
125821	training	2018-09-24 03:07:59.175026	thanos::sroute:16f438c4-c258-4955-8358-bdb1e25...	FTL	trip-153775847917477411
131578	training	2018-09-22 22:41:23.003019	thanos::sroute:c89bcbd0-26ee-4dc1-b9e7-cea1609...	Carting	trip-153765608300279258
142409	training	2018-09-24 10:08:09.819893	thanos::sroute:18104c25-811c-4b93-8c92-8c6923c...	FTL	trip-153778368981951440

In [148]:
```
df.drop(df[df['segment_actual_time'] < 0].index, inplace=True)
```

In [149]:
```
df.describe()
```

Out[149]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_tim
count	144846.000000	144846.000000	144846.000000	144846.00000
mean	961.226537	234.057171	416.908724	213.85300
std	1036.993595	344.974984	598.085058	307.99770
min	20.000000	9.000045	9.000000	6.00000
25%	161.000000	23.354927	51.000000	27.00000
50%	449.000000	66.126234	132.000000	64.00000
75%	1634.000000	286.706673	513.000000	257.00000
max	7898.000000	1927.447705	4532.000000	1686.00000

Let's Visualise the Data with the help of the graphs and Analyse the Data

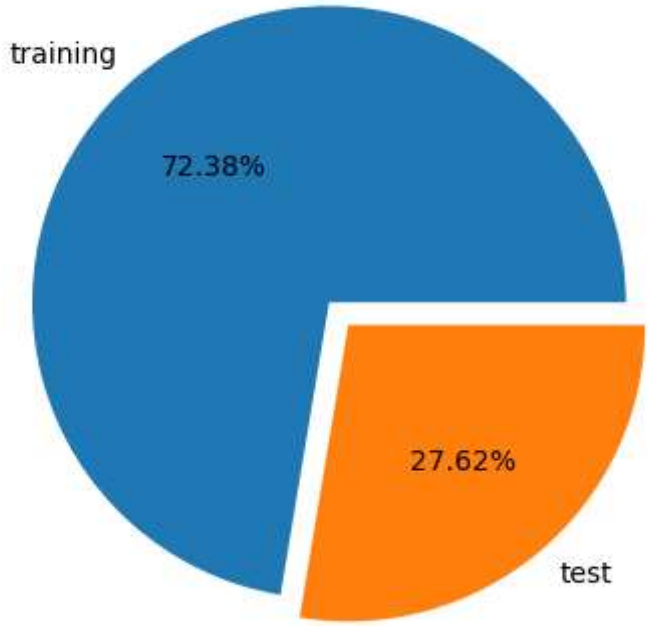
In [150]:
```
df['data'].unique()
```

Out[150]:
```
['training', 'test']
Categories (2, object): ['test', 'training']
```

In [151]:
```
value = df['data'].value_counts()
value
```

Out[151]:
```
training    104840
test         40006
Name: data, dtype: int64
```

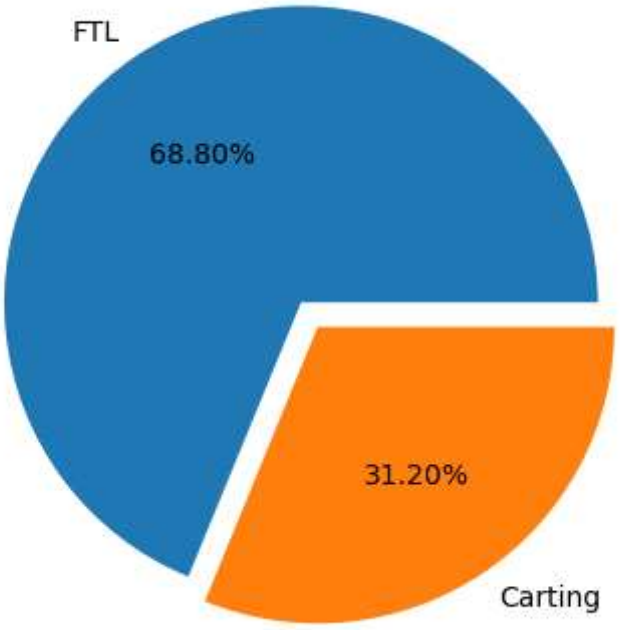
In [152]:
```
plt.pie(value, labels = ['training', 'test'],
        explode = [0, 0.1],
       autopct = '%.2f%%')
plt.show()
```



In [153]:
```
Route_type = df['route_type'].value_counts(normalize=False)
Route_type
```

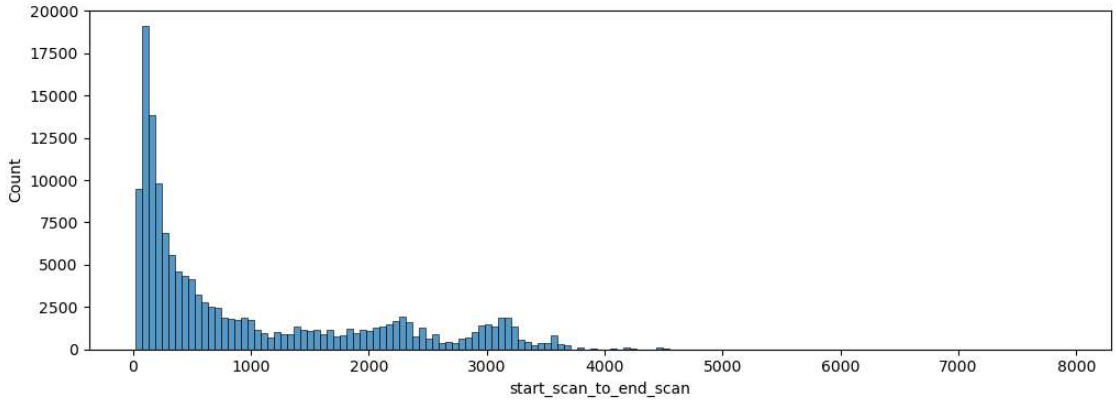
Out[153]:
```
FTL        99648
Carting    45198
Name: route_type, dtype: int64
```

In [154]:
```
plt.pie(Route_type, labels = ['FTL', 'Carting'],
        explode = [0, 0.1],
       autopct = '%.2f%%')
plt.show()
```

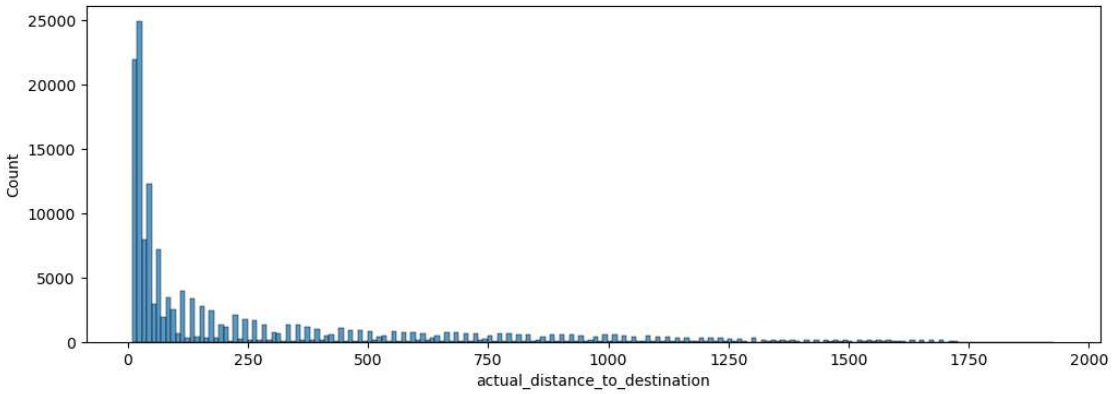


Let's Anlyse for the Univariate Analaysis

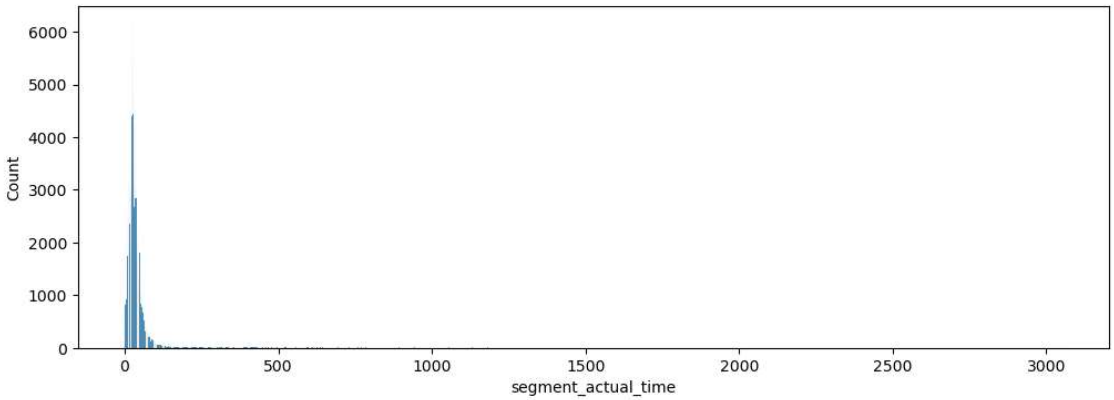
In [155]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['start_scan_to_end_scan'])
plt.show()
```



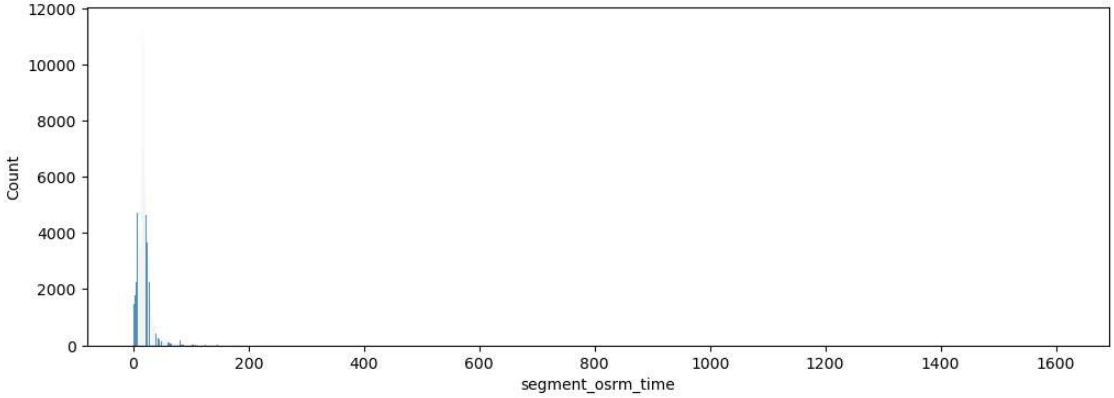
In [156]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['actual_distance_to_destination'])
plt.show()
```



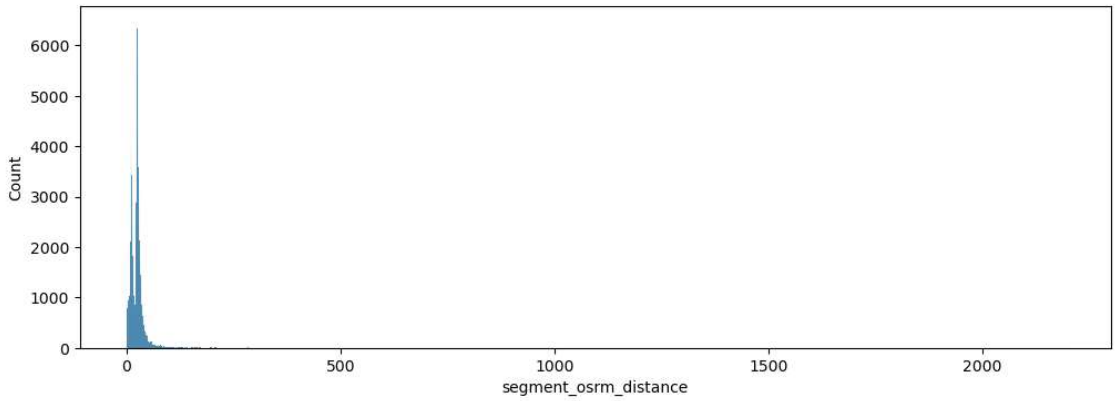
In [157]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['segment_actual_time'])
plt.show()
```



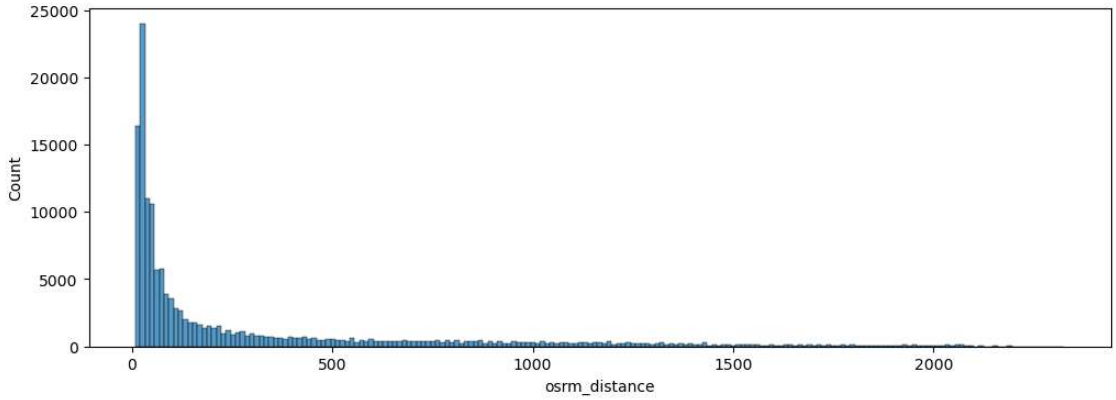
In [158]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['segment_osrm_time'])
plt.show()
```



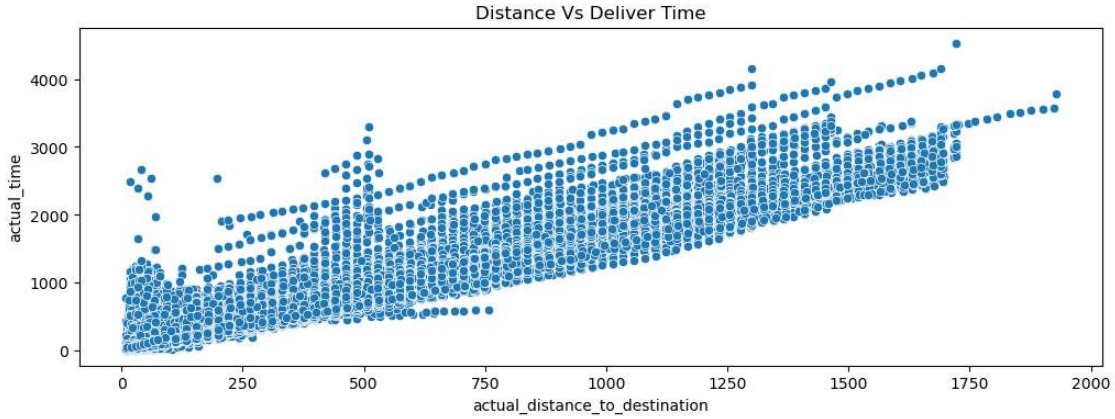
In [159]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['segment_osrm_distance'])
plt.show()
```



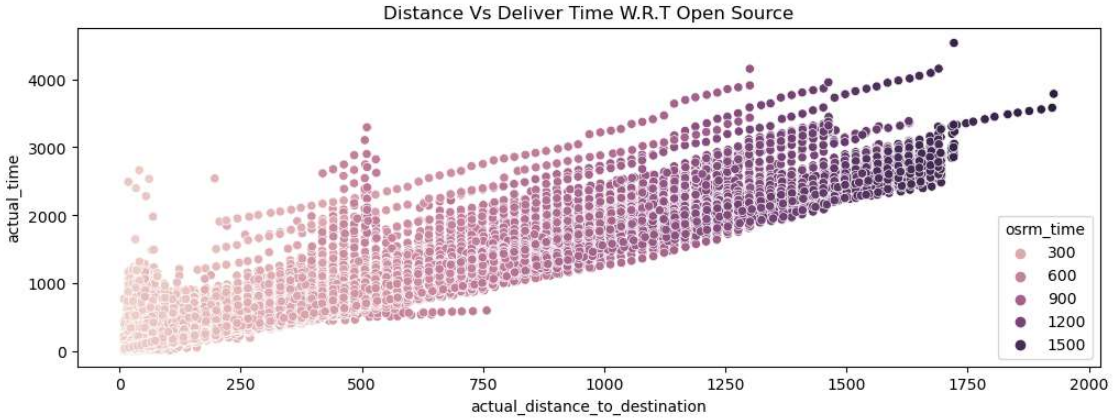
In [160]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['osrm_distance'])
plt.show()
```



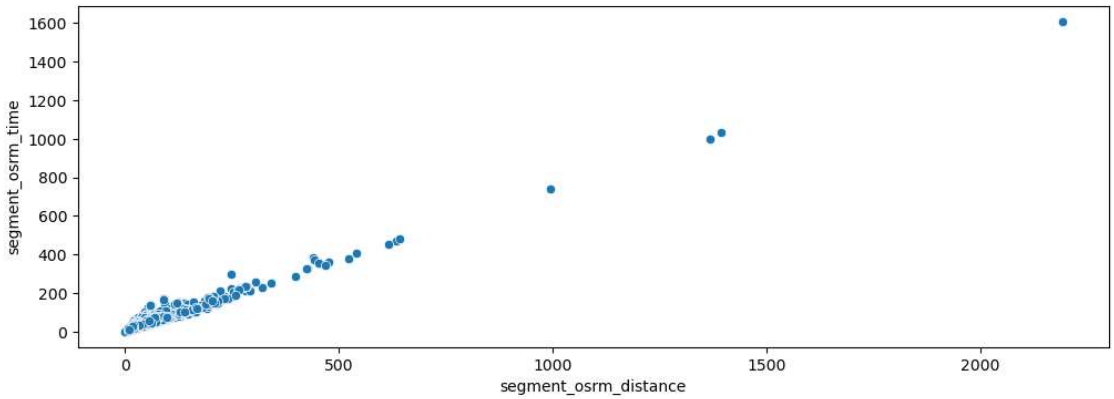
In [161]:
```
plt.figure(figsize=(12,4))
sns.scatterplot(data=df, x="actual_distance_to_destination", y="actua
plt.title("Distance Vs Deliver Time")
plt.show()
```



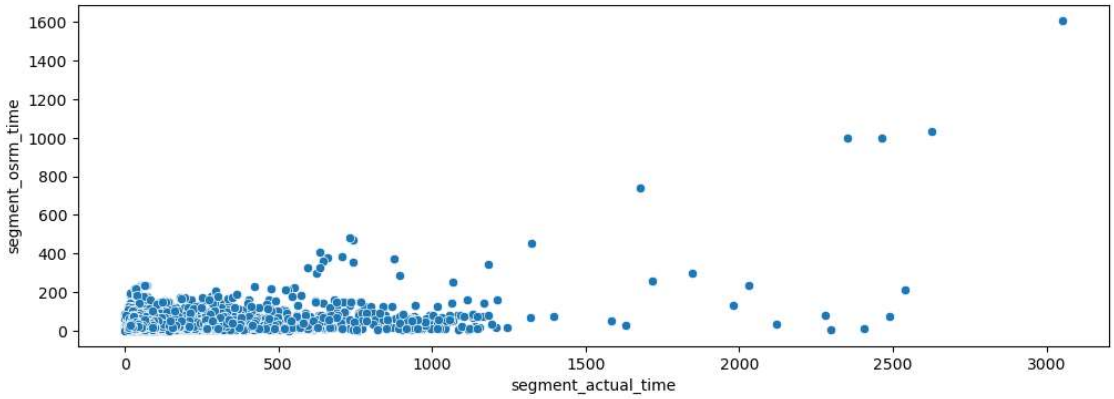
In [162]:
```
plt.figure(figsize=(12,4))
sns.scatterplot(data=df, x="actual_distance_to_destination", y="actua
plt.title("Distance Vs Deliver Time W.R.T Open Source")
plt.show()
```



In [163]:
```
plt.figure(figsize=(12,4))
sns.scatterplot(data=df, x="segment_osrm_distance", y="segment_osrm_t
plt.show()
```



In [164]:
```
plt.figure(figsize=(12,4))
sns.scatterplot(data=df, x="segment_actual_time", y="segment_osrm_tim
plt.show()
```



In [165]:
```
1 df.corr()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24136\1134722465.py:1: FutureWa
rning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  df.corr()
```

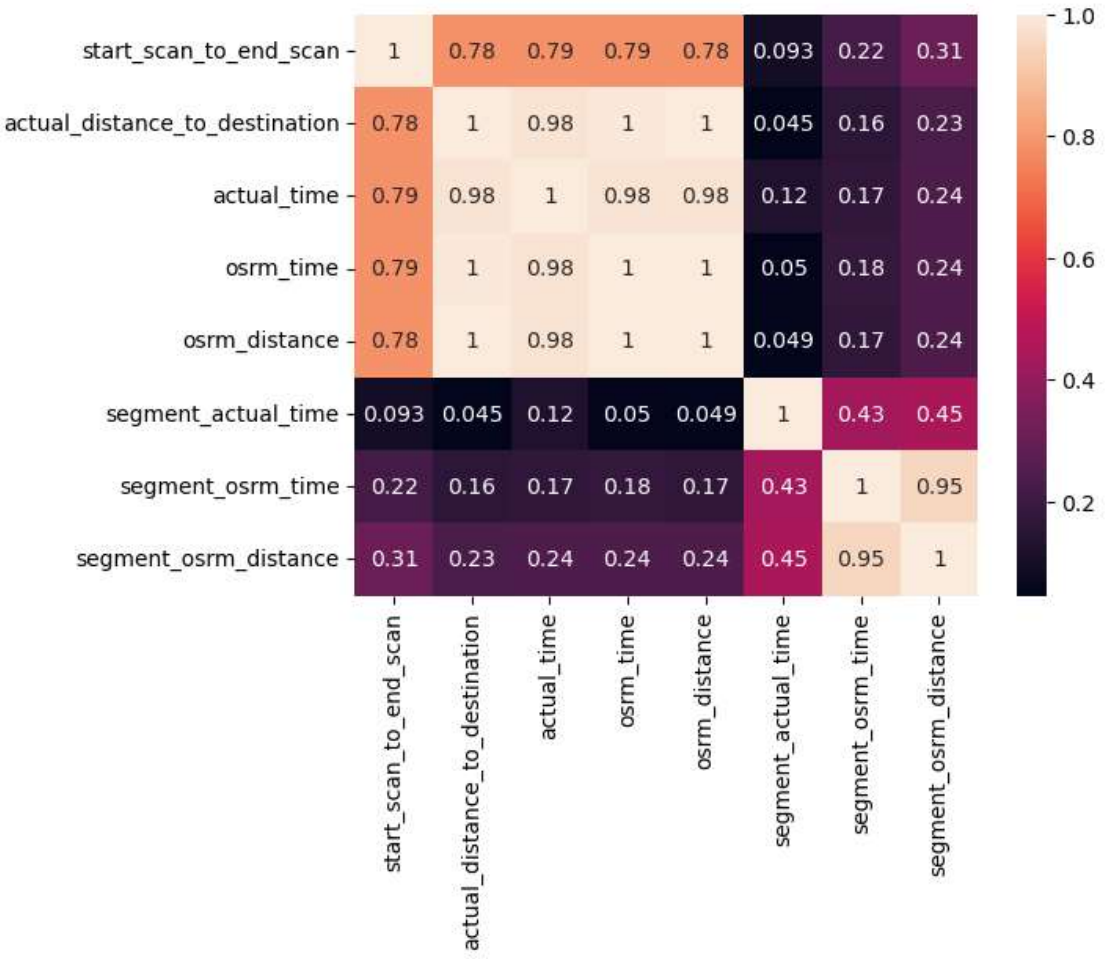
Out[165]:

	start_scan_to_end_scan	actual_distance_to_destination	actu
start_scan_to_end_scan	1.000000	0.784988	0.
actual_distance_to_destination	0.784988	1.000000	0.
actual_time	0.785924	0.978658	1.
osrm_time	0.785283	0.995872	0.
osrm_distance	0.784120	0.997148	0.
segment_actual_time	0.093372	0.045320	0.
segment_osrm_time	0.219844	0.158836	0.
segment_osrm_distance	0.306972	0.232119	0.

In [166]:
```
1 sns.heatmap(df.corr(), annot=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24136\621126171.py:1: FutureWar
ning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns
or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True)
```

Out[166]: <Axes: >



Feature Engineering

In [167]:
```
1 df['od_time_diff_hour'] = df['od_end_time'] - df['od_start_time']
2 df.drop(columns={'od_start_time','od_end_time'}, inplace=True)
```

In [168]:
```
df['od_time_diff_hour'] = df['od_time_diff_hour'].dt.total_seconds()/
df['od_time_diff_hour']
```

Out[168]:
```
0          86.213637
1          86.213637
2          86.213637
3          86.213637
4          86.213637
             ...
144862    427.686364
144863    427.686364
144864    427.686364
144865    427.686364
144866    427.686364
Name: od_time_diff_hour, Length: 144846, dtype: float64
```

In [169]:
```
df['destination_city'] = df['destination_name'].str.split("_", expand
```

In [170]:
```
df['destination_city'].head()
```

Out[170]:
```
0    Khambhat
1    Khambhat
2    Khambhat
3    Khambhat
4    Khambhat
Name: destination_city, dtype: object
```

In [171]:
```
df['destination_place'] = df['destination_name'].str.split("_", expan
```

In [172]:
```
df['destination_place'].head()
```

Out[172]:
```
0    MotvdDPP
1    MotvdDPP
2    MotvdDPP
3    MotvdDPP
4    MotvdDPP
Name: destination_place, dtype: object
```

In [173]:
```
df['destination_state'] = df['destination_name'].str.split("_", expan
```

In [174]:
```
df['destination_state'] = df['destination_state'].str.replace("(", ""
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24136\2165789120.py:1: FutureWa
rning: The default value of regex will change from True to False in a fu
ture version. In addition, single character regular expressions will *no
t* be treated as literal strings when regex=True.
  df['destination_state'] = df['destination_state'].str.replace("(",
"").str.replace(")", "")
```

In [175]:
```
1 df['destination_state']
```

Out[175]:
```
0         Gujarat
1         Gujarat
2         Gujarat
3         Gujarat
4         Gujarat
           ...
144862    Haryana
144863    Haryana
144864    Haryana
144865    Haryana
144866    Haryana
Name: destination_state, Length: 144846, dtype: object
```

In [176]:
```
1 df['trip_year'] = df['trip_creation_time'].dt.year
2 df['trip_year']
```

Out[176]:
```
0         2018
1         2018
2         2018
3         2018
4         2018
          ...
144862    2018
144863    2018
144864    2018
144865    2018
144866    2018
Name: trip_year, Length: 144846, dtype: int64
```

In [177]:
```
1 df['trip_month'] = df['trip_creation_time'].dt.month
2 df['trip_month']
```

Out[177]:
```
0         9
1         9
2         9
3         9
4         9
         ..
144862    9
144863    9
144864    9
144865    9
144866    9
Name: trip_month, Length: 144846, dtype: int64
```

In [178]:
```
df['trip_day'] = df['trip_creation_time'].dt.day
df['trip_day']
```

Out[178]:
```
0         20
1         20
2         20
3         20
4         20
          ..
144862    20
144863    20
144864    20
144865    20
144866    20
Name: trip_day, Length: 144846, dtype: int64
```

In [464]:
```
df['source_state'] = df['source_name'].str.split("_", expand=True)[2]
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24136\726971721.py:1: FutureWar
ning: The default value of regex will change from True to False in a fut
ure version. In addition, single character regular expressions will *not
* be treated as literal strings when regex=True.
  df['source_state'] = df['source_name'].str.split("_", expand=True)[2].
str.split(" ", expand=True)[1].str.replace("(","").str.replace(")","")
```

In [466]:
```
df['source_state']
```

Out[466]:
```
0         Gujarat
1         Gujarat
2         Gujarat
3         Gujarat
4         Gujarat
           ...
144862    Haryana
144863    Haryana
144864    Haryana
144865    Haryana
144866    Haryana
Name: source_state, Length: 144846, dtype: object
```

In [473]:
```
df['source_city'] =  df['source_name'].str.split("_", expand=True)[0]
```

In [474]:
```
df['source_city'].head()
```

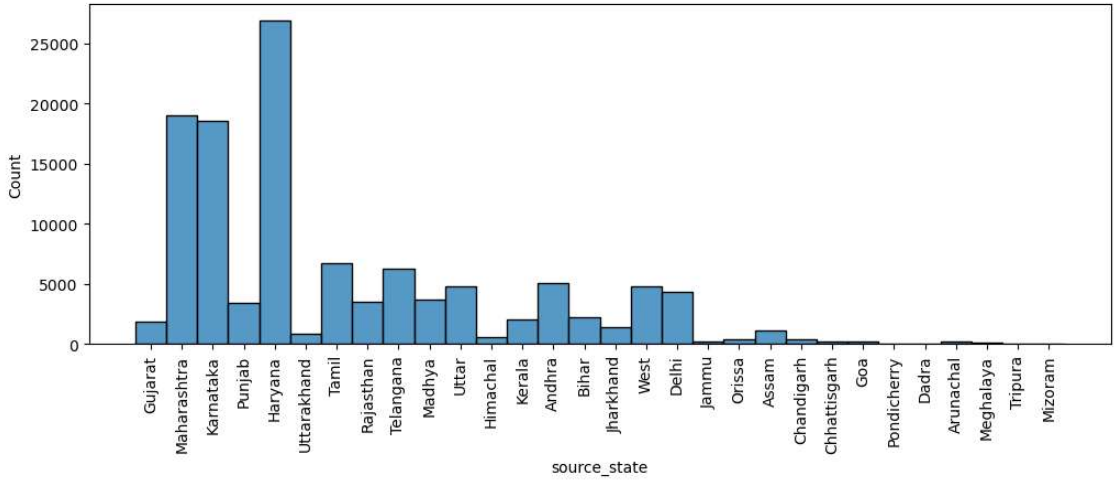
Out[474]:
```
0    Anand
1    Anand
2    Anand
3    Anand
4    Anand
Name: source_city, dtype: object
```

In [481]:
```
df['source_place'] = df['source_name'].str.split("_", expand =True)[1
```

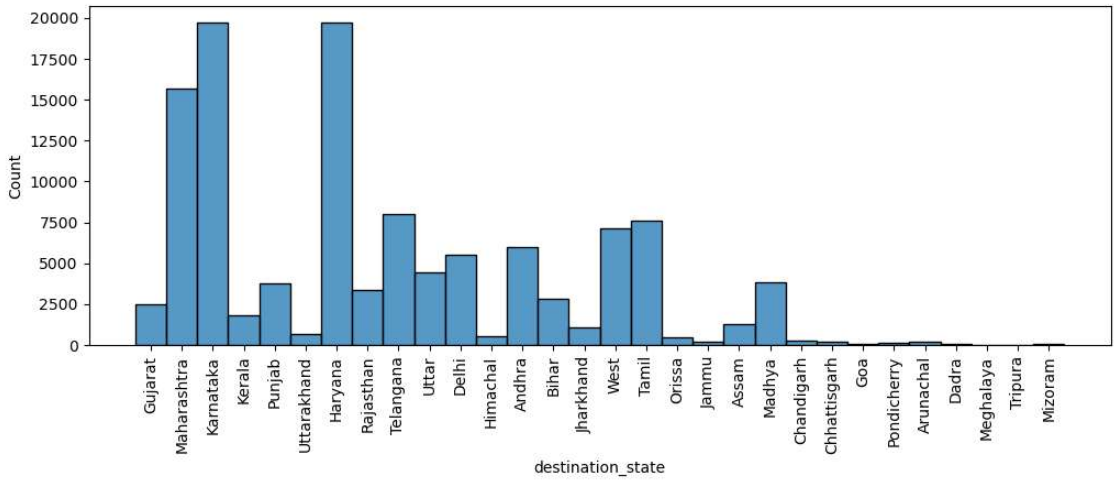
In [482]:
```
df['source_place'].head()
```

Out[482]:
```
0    VUNagar
1    VUNagar
2    VUNagar
3    VUNagar
4    VUNagar
Name: source_place, dtype: object
```

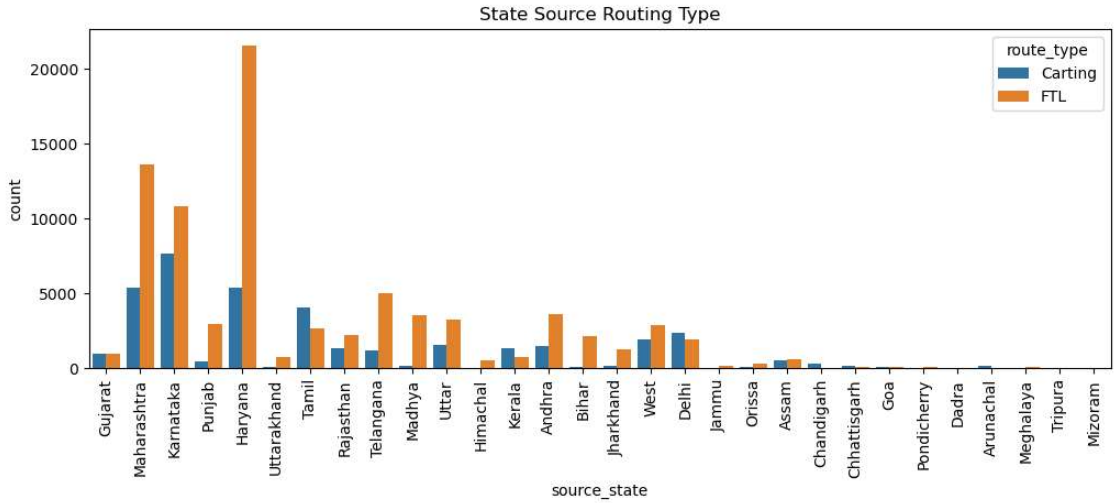
In [503]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['source_state'])
plt.xticks(rotation = 90)
plt.show()
```



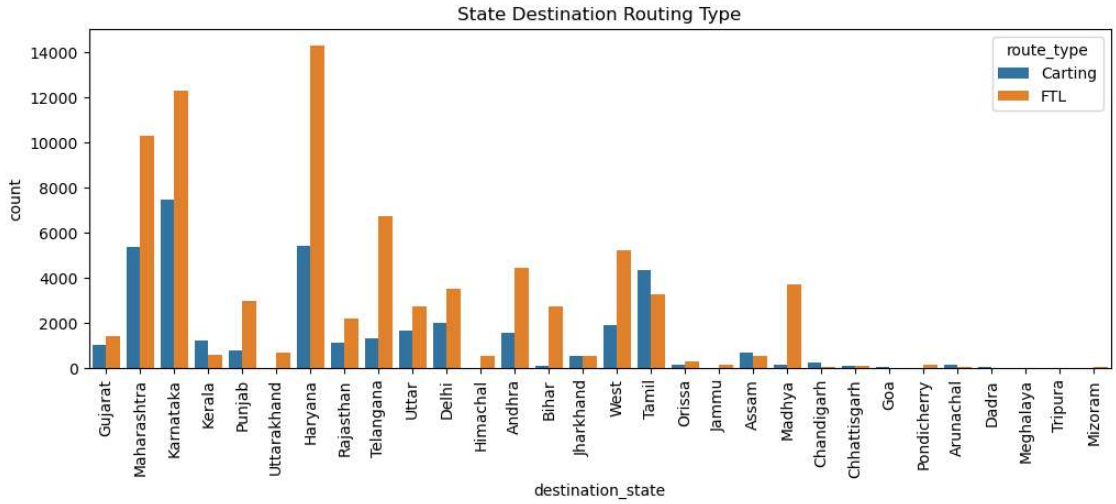
In [179]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['destination_state'])
plt.xticks(rotation = 90)
plt.show()
```



In [504]:
```
plt.figure(figsize=(12,4))
sns.countplot(data=df, x = df['source_state'], hue='route_type')
plt.xticks(rotation = 90)
plt.title("State Source Routing Type")
plt.show()
```



In [505]:
```
plt.figure(figsize=(12,4))
sns.countplot(data=df, x = df['destination_state'], hue='route_type')
plt.xticks(rotation = 90)
plt.title("State Destination Routing Type")
plt.show()
```



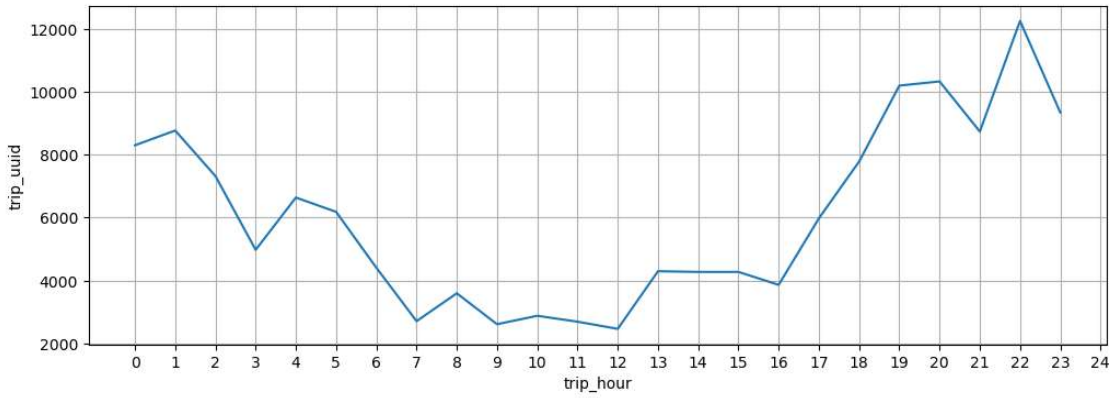
In [181]:
```
df['trip_hour'] = df['trip_creation_time'].dt.hour
```

In [182]:
```
hour_trips = df.groupby(by='trip_hour')['trip_uuid'].count().reset_in
hour_trips.head()
```

Out[182]:

	trip_hour	trip_uuid
0	0	8299
1	1	8768
2	2	7320
3	3	4975
4	4	6637

In [183]:
```
plt.figure(figsize=(12,4))
sns.lineplot(data=hour_trips, x='trip_hour', y="trip_uuid")
plt.grid('both')
plt.xticks(np.arange(0,25))
plt.show()
```



In [184]:
```
df['route_schedule'] = df['route_schedule_uuid'].str.split(":", expan
```

In [185]:
```
df['route_schedule']
```

Out[185]:
```
0         thanos
1         thanos
2         thanos
3         thanos
4         thanos
           ...
144862    thanos
144863    thanos
144864    thanos
144865    thanos
144866    thanos
Name: route_schedule, Length: 144846, dtype: object
```

In [447]:
```
1 df[:2]
```

Out[447]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

2 rows × 26 columns

In [446]:
```
1 df.nunique()
```

Out[446]:

```
data                              2
trip_creation_time            14817
route_schedule_uuid            1504
route_type                        2
trip_uuid                     14817
source_center                 1508
source_name                   1498
destination_center            1481
destination_name              1468
start_scan_to_end_scan        1915
actual_distance_to_destination  144494
actual_time                   3182
osrm_time                     1531
osrm_distance                 138026
segment_actual_time            729
segment_osrm_time              214
segment_osrm_distance         113782
od_time_diff_hour              26369
destination_city               1258
destination_place              1154
destination_state                30
trip_year                         1
trip_month                        2
trip_day                         22
trip_hour                        24
route_schedule                    1
dtype: int64
```

In [485]:
```
1 df['source_state'].value_counts()
```

Out[485]:
```
Haryana          26935
Maharashtra      18993
Karnataka        18531
Tamil             6697
Telangana         6212
Andhra            5087
Uttar             4791
West              4742
Delhi             4317
Madhya            3687
Rajasthan         3529
Punjab            3380
Bihar             2206
Kerala            2038
Gujarat           1867
Jharkhand         1343
Assam             1126
Uttarakhand        827
Himachal           532
Orissa             404
Chandigarh         367
Chhattisgarh       229
Jammu              182
Goa                165
Arunachal          150
Meghalaya           77
Pondicherry         49
Dadra               30
Mizoram             26
Tripura              5
Name: source_state, dtype: int64
```

In [487]:
```
1 df['destination_state'].value_counts()
```

Out[487]:
```
Karnataka       19750
Haryana         19744
Maharashtra     15682
Telangana        8032
Tamil            7610
West             7110
Andhra           5984
Delhi            5490
Uttar            4413
Madhya           3852
Punjab           3751
Rajasthan        3335
Bihar            2809
Gujarat          2456
Kerala           1834
Assam            1249
Jharkhand        1076
Uttarakhand       666
Himachal          543
Orissa            453
Chandigarh        282
Chhattisgarh      221
Arunachal         185
Jammu             167
Pondicherry       154
Goa                74
Dadra              34
Mizoram            31
Tripura             9
Meghalaya           8
Name: destination_state, dtype: int64
```

In [490]:
```
1 df['source_name'].value_counts()[:5]
```

Out[490]:
```
Gurgaon_Bilaspur_HB (Haryana)        23342
Bangalore_Nelmngla_H (Karnataka)      9972
Bhiwandi_Mankoli_HB (Maharashtra)     9085
Pune_Tathawde_H (Maharashtra)         4061
Hyderabad_Shamshbd_H (Telangana)      3340
Name: source_name, dtype: int64
```

In [491]:
```
1 df['destination_name'].value_counts()[:5]
```

Out[491]:
```
Gurgaon_Bilaspur_HB (Haryana)        15189
Bangalore_Nelmngla_H (Karnataka)     11016
Bhiwandi_Mankoli_HB (Maharashtra)     5492
Hyderabad_Shamshbd_H (Telangana)      5141
Kolkata_Dankuni_HB (West Bengal)      4891
Name: destination_name, dtype: int64
```

In [493]:
```
1 df["source_destination"]=df["source_name"] + df["destination_name"]
```

In [494]:
```
1 df['source_destination'][:4]
```

Out[494]:
```
0    Anand_VUNagar_DC (Gujarat)Khambhat_MotvdDPP_D ...
1    Anand_VUNagar_DC (Gujarat)Khambhat_MotvdDPP_D ...
2    Anand_VUNagar_DC (Gujarat)Khambhat_MotvdDPP_D ...
3    Anand_VUNagar_DC (Gujarat)Khambhat_MotvdDPP_D ...
Name: source_destination, dtype: object
```

Bussiest Corridor

In [502]:
```
1 df['source_destination'].value_counts()[:5]
```

Out[502]:
```
Gurgaon_Bilaspur_HB (Haryana)Bangalore_Nelmngla_H (Karnataka)     4975
Bangalore_Nelmngla_H (Karnataka)Gurgaon_Bilaspur_HB (Haryana)     3316
Gurgaon_Bilaspur_HB (Haryana)Kolkata_Dankuni_HB (West Bengal)     2862
Gurgaon_Bilaspur_HB (Haryana)Hyderabad_Shamshbd_H (Telangana)     1638
Gurgaon_Bilaspur_HB (Haryana)Bhiwandi_Mankoli_HB (Maharashtra)    1617
Name: source_destination, dtype: int64
```

Least Bussiest Corridor

In [501]:
```
1 df['source_destination'].value_counts(ascending=True)[:5]
```

Out[501]:
```
Malerkotla_DC (Punjab)Dhuri_DMComDPP_D (Punjab)
1
Daurala_Sardhnrd_D (Uttar Pradesh)Khatauli_TilakNgr_D (Uttar Pradesh)
1
Salem_Kadtmpty_H (Tamil Nadu)Salem_Kadtmpty_D (Tamil Nadu)
1
Kayamkulam_Bhrnikvu_D (Kerala)Manthuka_Central_D_1 (Kerala)
1
Mumbai_East_I_21 (Maharashtra)Mumbai_Kalyan (Maharashtra)
1
Name: source_destination, dtype: int64
```

Average Distance between Bussiest Corridor

In [511]:
```
1 df[df['source_destination'] == "Gurgaon_Bilaspur_HB (Haryana)Bangalor
```

Out[511]:
```
859.7926156621767
```

Average Distance between Least Bussiest Corridor

In [516]:
```
1 df[df['source_destination'] == "Malerkotla_DC (Punjab)Dhuri_DMComDPP_
```

Out[516]: 17.100289404993177

Average Time Between the Bussiest Corridor

In [517]:
```
1 df[df['source_destination'] == "Gurgaon_Bilaspur_HB (Haryana)Bangalor
```

Out[517]: 1367.1736683417084

Average Time Between the Least Bussiest Corridor

In [518]:
```
1 df[df['source_destination'] == "Malerkotla_DC (Punjab)Dhuri_DMComDPP_
```

Out[518]: 36.0

Data Wrangling

Grouping by segment || In- Depth Analysis

a. Create a unique identifier for different segments of a trip based on the combination of the trip_uuid, source_center, and destination_center and name it as segment_key.

In [186]:
```
1 df.head()
```

Out[186]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND3

5 rows × 26 columns

In [187]:
```
1 df.columns
```

Out[187]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_typ
e',
       'trip_uuid', 'source_center', 'source_name', 'destination_cente
r',
       'destination_name', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'od_time_diff_hour', 'destination_city',
       'destination_place', 'destination_state', 'trip_year', 'trip_mont
h',
       'trip_day', 'trip_hour', 'route_schedule'],
      dtype='object')
```

In [188]:
```
grouping_1 = ['trip_uuid', 'source_center', 'destination_center']
df1 = df.groupby(by = grouping_1, as_index = False).agg({'data' : 'fi
                                                         'route_type'
                                                        'trip_creation
                                                        'source_name'
                                                        'destination_n
                                                        'start_scan_to
                                                        'actual_distan
                                                        'actual_time'
                                                        'osrm_time' :
                                                        'osrm_distance
                                                        'segment_actua
                                                        'segment_osrm_
                                                        'segment_osrm_
                                                         })
```

In [189]:
```
df2 = df1.groupby(by = 'trip_uuid', as_index = False).agg({'source_ce
                                                           'destinati
                                                           'data' : '
                                                           'route_typ
                                                           'trip_crea
                                                           'source_na
                                                           'destinati
                                                           'start_sca
                                                           'actual_di
                                                           'osrm_time
                                                           'osrm_dist
                                                           'segment_a
                                                           'segment_o
                                                           'segment_o
```

In [190]:
```
df2.shape
```

Out[190]: (14817, 15)

In [191]:
```
df2.describe().T
```

Out[191]:

	count	mean	std	min	25%	
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000	280
actual_distance_to_destination	14817.0	164.477838	305.388147	9.002461	22.837239	48
osrm_time	14817.0	161.384018	271.360995	6.000000	29.000000	60
osrm_distance	14817.0	204.344689	370.395573	9.072900	30.819200	65
segment_actual_time	14817.0	353.951610	556.320988	9.000000	66.000000	147
segment_osrm_time	14817.0	180.921172	314.485624	6.000000	31.000000	65
segment_osrm_distance	14817.0	223.164000	416.547252	9.072900	32.654500	70

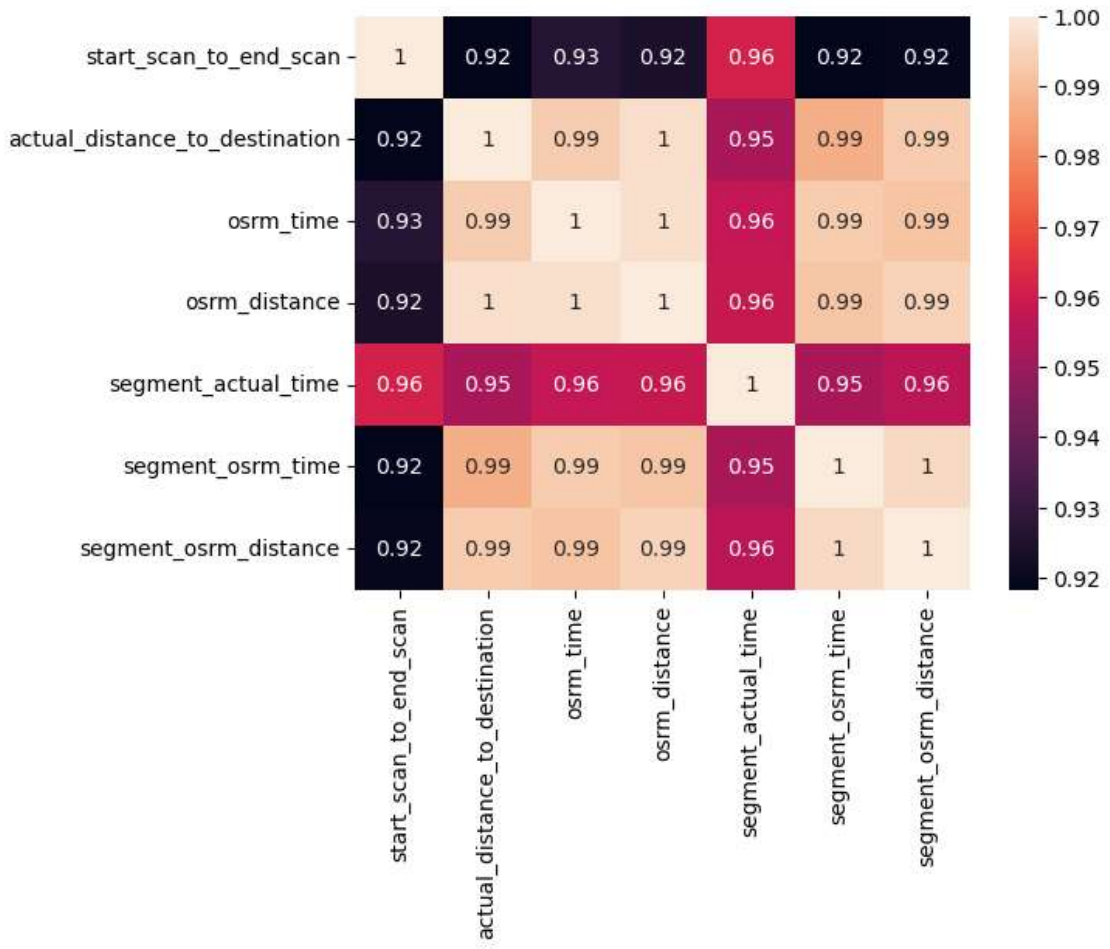
In [192]:
```
1 df2.isna().sum()
```

Out[192]:
```
trip_uuid                         0
source_center                     0
destination_center                0
data                              0
route_type                        0
trip_creation_time                0
source_name                      10
destination_name                  8
start_scan_to_end_scan            0
actual_distance_to_destination    0
osrm_time                         0
osrm_distance                     0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
dtype: int64
```

In [193]:
```
1 sns.heatmap(df2.corr(), annot=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24136\1738403036.py:1: FutureWa
rning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df2.corr(), annot=True)
```

Out[193]: <Axes: >



In [194]:
```
1 df2.head()
```

Out[194]:

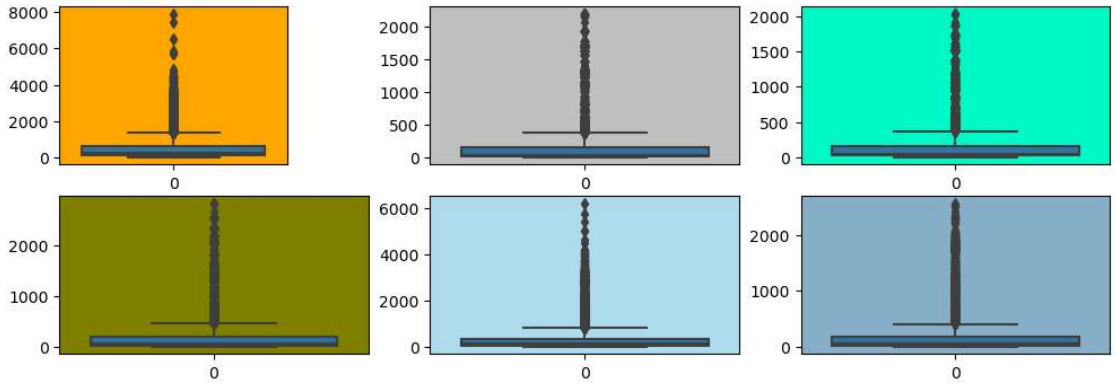
	trip_uuid	source_center	destination_center	data	route_type	trip_creation
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018 00:00:33.6
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018 00:01:00.1
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018 00:02:09.7

Detecting Outliers for the Numerical Columns

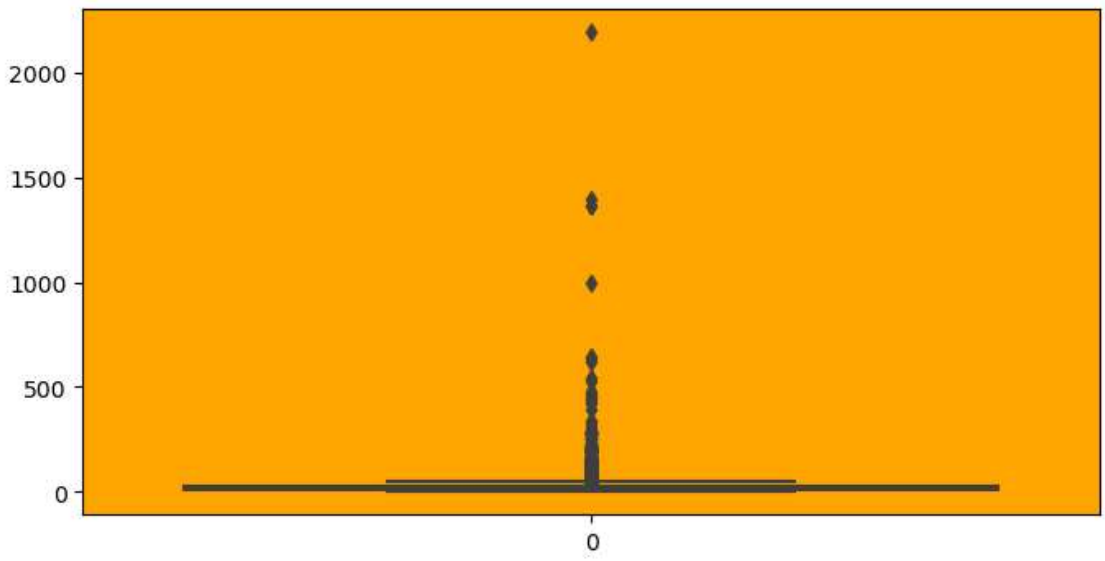
In [195]:
```
1 numerical_columns = ['start_scan_to_end_scan', 'actual_distance_to_de
2                       'osrm_time', 'osrm_distance', 'segment_actual_tim
3                       'segment_osrm_time', 'segment_osrm_distance']
```

In [196]:

```
plt.figure(figsize=(12,4))
plt.subplot(2,4,1)
sns.boxplot(df2['start_scan_to_end_scan'])
ax=plt.gca()
ax.set_facecolor('orange')
plt.subplot(2,3,2)
sns.boxplot(df2['actual_distance_to_destination'])
ax=plt.gca()
ax.set_facecolor('silver')
plt.subplot(2,3,3)
sns.boxplot(df2['osrm_time'])
ax=plt.gca()
ax.set_facecolor('#01F9C6')
plt.subplot(2,3,4)
sns.boxplot(df2['osrm_distance'])
ax=plt.gca()
ax.set_facecolor('#808000')
plt.subplot(2,3,5)
sns.boxplot(df2['segment_actual_time'])
ax=plt.gca()
ax.set_facecolor('#AFDCEC')
plt.subplot(2,3,6)
sns.boxplot(df2['segment_osrm_time'])
ax=plt.gca()
ax.set_facecolor('#87AFC7')
plt.show()
```



In [197]:
```
1 plt.figure(figsize=(8,4))
2 sns.boxplot(df['segment_osrm_distance'])
3 ax=plt.gca()
4 ax.set_facecolor('#FFA500')
5 plt.show()
```



In [198]:
```
1 df2[numerical_columns].describe().T
```

Out[198]:

	count	mean	std	min	25%	
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000	280
actual_distance_to_destination	14817.0	164.477838	305.388147	9.002461	22.837239	48
osrm_time	14817.0	161.384018	271.360995	6.000000	29.000000	60
osrm_distance	14817.0	204.344689	370.395573	9.072900	30.819200	65
segment_actual_time	14817.0	353.951610	556.320988	9.000000	66.000000	147
segment_osrm_time	14817.0	180.921172	314.485624	6.000000	31.000000	65
segment_osrm_distance	14817.0	223.164000	416.547252	9.072900	32.654500	70

Outliers for start_scan_to_end_scan

In [199]:

```
q1 = np.quantile(df2['start_scan_to_end_scan'], 0.25)
q3 = np.quantile(df2['start_scan_to_end_scan'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['start_scan_to_end_scan'] < LW) | (df2['start
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 149.0 , 75th Percentile:-> 637.0
--------------------------------------------------
The IQR is : --> 488.0
--------------------------------------------------
The Lower wick --> -583.0
--------------------------------------------------
The Upper wick --> 1369.0
--------------------------------------------------
The Outliers present are: ->  1267
```

Outliers for actual_distance_to_destination

In [200]:

```
q1 = np.quantile(df2['actual_distance_to_destination'], 0.25)
q3 = np.quantile(df2['actual_distance_to_destination'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['actual_distance_to_destination'] < LW) | (df
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 22.83723905859321 , 75th Percentile:-> 164.5832076384
1138
--------------------------------------------------
The IQR is : --> 141.74596857981817
--------------------------------------------------
The Lower wick --> -189.78171381113404
--------------------------------------------------
The Upper wick --> 377.2021605081386
--------------------------------------------------
The Outliers present are: ->  1449
```

Outliers for osrm_time

In [201]:
```
q1 = np.quantile(df2['osrm_time'], 0.25)
q3 = np.quantile(df2['osrm_time'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['osrm_time'] < LW) | (df2['osrm_time'] > UW)]
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 29.0 , 75th Percentile:-> 168.0
--------------------------------------------------
The IQR is : --> 139.0
--------------------------------------------------
The Lower wick --> -179.5
--------------------------------------------------
The Upper wick --> 376.5
--------------------------------------------------
The Outliers present are: ->  1517
```

Outliers for osrm_distance

In [202]:
```
q1 = np.quantile(df2['osrm_distance'], 0.25)
q3 = np.quantile(df2['osrm_distance'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['osrm_distance'] < LW) | (df2['osrm_distance'
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 30.8192 , 75th Percentile:-> 208.475
--------------------------------------------------
The IQR is : --> 177.6558
--------------------------------------------------
The Lower wick --> -235.6645
--------------------------------------------------
The Upper wick --> 474.9587
--------------------------------------------------
The Outliers present are: ->  1524
```

Outliers for segment_actual_time

In [203]:

```
q1 = np.quantile(df2['segment_actual_time'], 0.25)
q3 = np.quantile(df2['segment_actual_time'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['segment_actual_time'] < LW) | (df2['segment_
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 66.0 , 75th Percentile:-> 367.0
--------------------------------------------------
The IQR is : --> 301.0
--------------------------------------------------
The Lower wick --> -385.5
--------------------------------------------------
The Upper wick --> 818.5
--------------------------------------------------
The Outliers present are: ->  1643
```

Outliers for segment_osrm_time

In [204]:

```
q1 = np.quantile(df2['segment_osrm_time'], 0.25)
q3 = np.quantile(df2['segment_osrm_time'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['segment_osrm_time'] < LW) | (df2['segment_os
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 31.0 , 75th Percentile:-> 185.0
--------------------------------------------------
The IQR is : --> 154.0
--------------------------------------------------
The Lower wick --> -200.0
--------------------------------------------------
The Upper wick --> 416.0
--------------------------------------------------
The Outliers present are: ->  1492
```

Outliers for segment_osrm_distance

In [205]:

```
q1 = np.quantile(df2['segment_osrm_distance'], 0.25)
q3 = np.quantile(df2['segment_osrm_distance'], 0.75)
print(f"25th Percentile:-> {q1} , 75th Percentile:-> {q3}")
print("-"*50)
IQR = q3 - q1
print(f"The IQR is : --> {IQR}")
print("-"*50)
LW = q1-1.5*IQR
print(f"The Lower wick --> {LW}")
print("-"*50)
UW = q3 + 1.5*IQR
print(f"The Upper wick --> {UW}")
print("-"*50)
outliers = df2.loc[(df2['segment_osrm_distance'] < LW) | (df2['segmen
print("The Outliers present are: -> ",len(outliers))
```

```
25th Percentile:-> 32.6545 , 75th Percentile:-> 218.7102
--------------------------------------------------
The IQR is : --> 186.0557
--------------------------------------------------
The Lower wick --> -246.42905000000002
--------------------------------------------------
The Upper wick --> 497.79375
--------------------------------------------------
The Outliers present are: ->  1549
```

Let's Perform one-hot encoding on categorical features

In [206]:
```
1 df2[:2]
```

Out[206]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

Featuring data columns which have the two values in it, Lets perform the Label Encoding for the following

In [208]:
```
1 df2['data'].value_counts()
```

Out[208]:
```
training    10654
test         4163
Name: data, dtype: int64
```

In [209]:
```
1 from sklearn.preprocessing import LabelEncoder
```

In [210]:
```
1 le = LabelEncoder()
2 le
```

Out[210]:
```
LabelEncoder()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [211]:
```
1 df2['data_label'] = le.fit_transform(df2['data'])
```

In [519]:
```
1 df2["data_label"]
```

Out[519]:
```
0        1
1        1
2        1
3        1
4        1
        ..
14812    0
14813    0
14814    0
14815    0
14816    0
Name: data_label, Length: 14817, dtype: int32
```

In [213]:
```
1 df2["data"].value_counts()
```

Out[213]:
```
training    10654
test         4163
Name: data, dtype: int64
```

In [214]:
```
1 df2.head()
```

Out[214]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018 00:00:33.6
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018 00:01:00.1
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018 00:02:09.7

Another columns have a unqiue for the labling which is the route type for the Label Encoder

In [217]:
```
1 df2['route_type'].value_counts()
```

Out[217]:
```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

In [219]:
```
labelencoder = LabelEncoder()
labelencoder
```

Out[219]:
```
LabelEncoder()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [222]:
```
df2['route_label'] =labelencoder.fit_transform(df2['route_type'])
df2['route_label']
```

Out[222]:
```
0        1
1        0
2        1
3        0
4        1
        ..
14812    0
14813    0
14814    0
14815    0
14816    1
Name: route_label, Length: 14817, dtype: int32
```

In [223]:
```
df2.head()
```

Out[223]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018 00:00:33.6
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018 00:01:00.1
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018 00:02:09.7

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

Let's Analyse the Every Numerical Features from the Data Set with the Help of the MinMax scaler

In [228]:
```
Data = df2.copy()
Data.head()
```

Out[228]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018 00:00:33.6
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018 00:01:00.1
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018 00:02:09.7

In [229]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [230]:
```
mm = MinMaxScaler()
mm
```

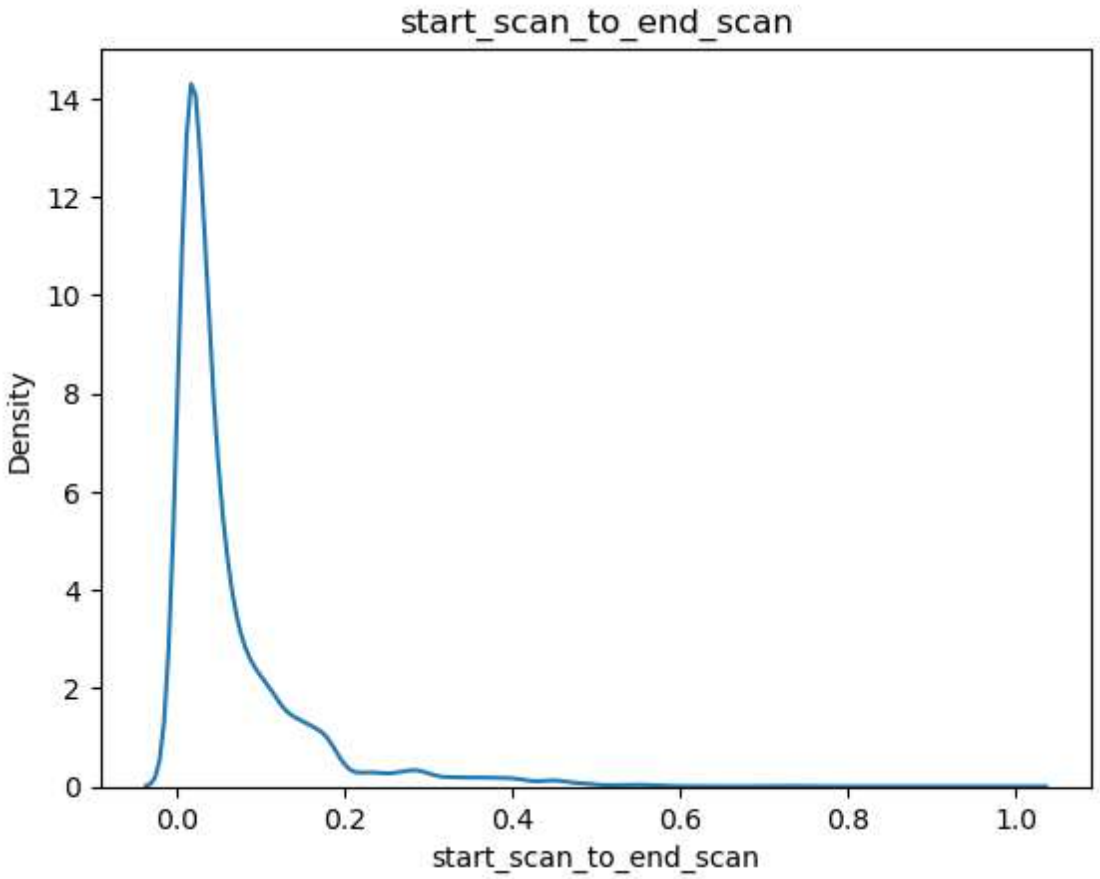
Out[230]:
```
MinMaxScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [243]:
```
Data['start_scan_to_end_scan'] = mm.fit_transform(Data['start_scan_to
Data['start_scan_to_end_scan'].head(2)
```

Out[243]:
```
0    0.283937
1    0.019937
Name: start_scan_to_end_scan, dtype: float64
```

Normalization also known as min max scaling, basically uses the data into the scale ranges to 0 to 1. After the normalization the data ranges between 0 to 1. Useful when you want to scale the features to a specific range, especially if the algorithm you are using is sensitive to the scale of the variables.

In [267]:
```
sns.kdeplot(Data['start_scan_to_end_scan'])
plt.title("start_scan_to_end_scan")
plt.show()
```



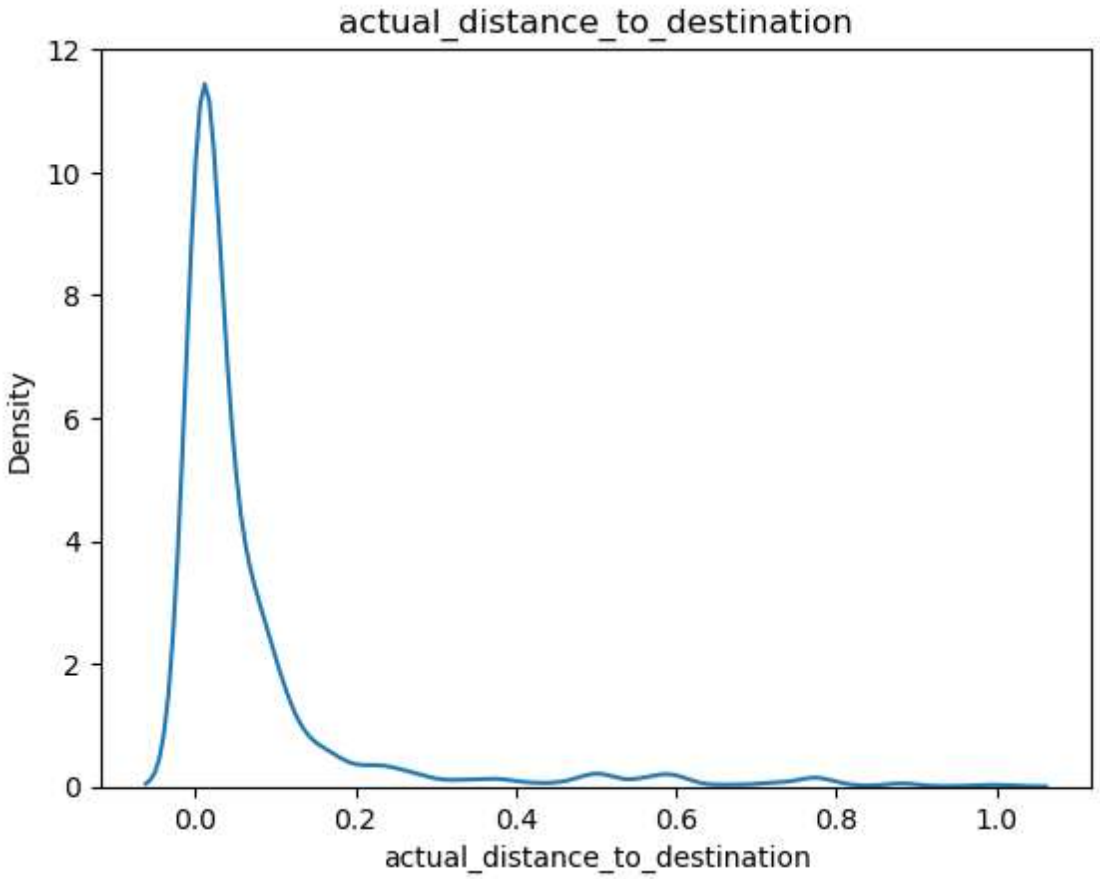
In [254]:
```
mm = MinMaxScaler()
mm
```

Out[254]:
```
MinMaxScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [264]:
```
Data['actual_distance_to_destination'] = mm.fit_transform(Data['actua
Data['actual_distance_to_destination'][:3]
```

Out[264]:
```
0    0.374613
1    0.029476
2    0.880999
Name: actual_distance_to_destination, dtype: float64
```

In [268]:
```
1 sns.kdeplot(Data['actual_distance_to_destination'])
2 plt.title("actual_distance_to_destination")
3 plt.show()
```



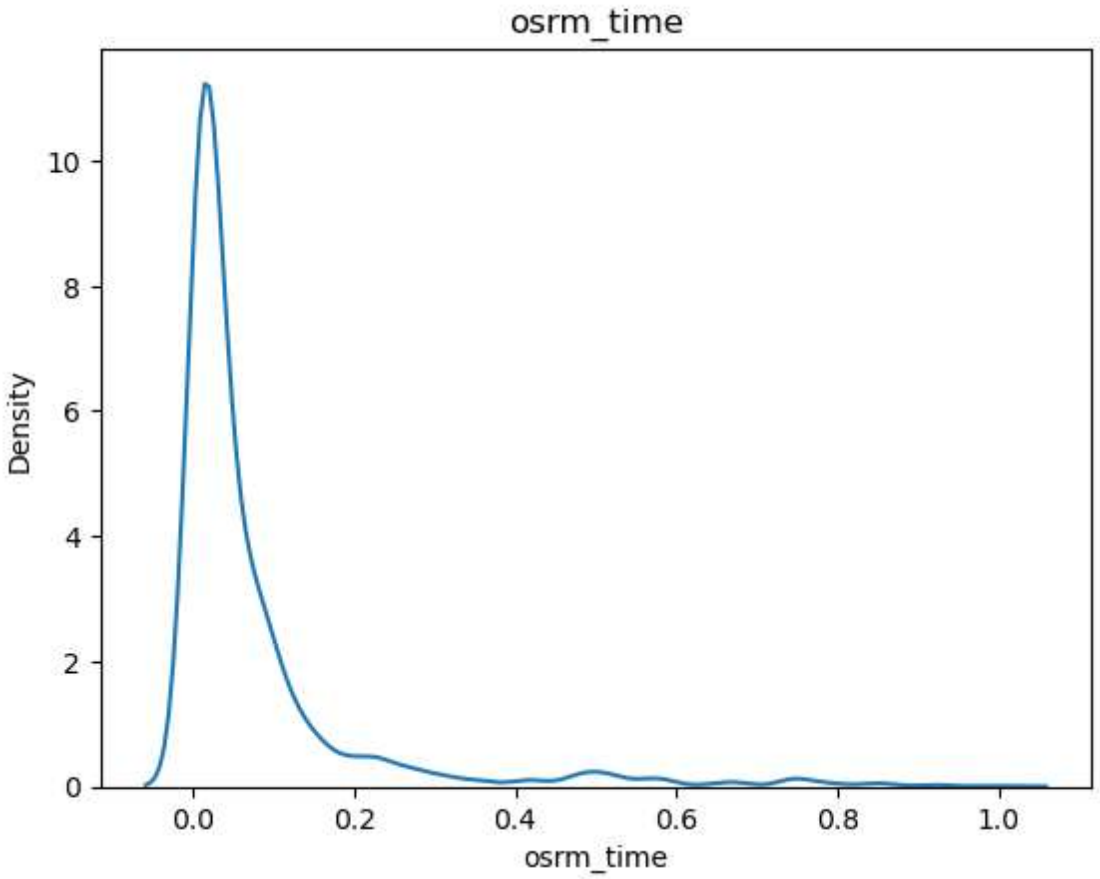
In [269]:
```
1 mm = MinMaxScaler()
2 mm
```

Out[269]:
```
MinMaxScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [270]:
```
1 Data['osrm_time'] = mm.fit_transform(Data['osrm_time'].to_numpy().res
2 Data['osrm_time'][:3]
```

Out[270]:
```
0    0.350938
1    0.030602
2    0.855874
Name: osrm_time, dtype: float64
```

In [272]:
```
sns.kdeplot(Data['osrm_time'])
plt.title("osrm_time")
plt.show()
```



In [273]:
```
mm = MinMaxScaler()
mm
```

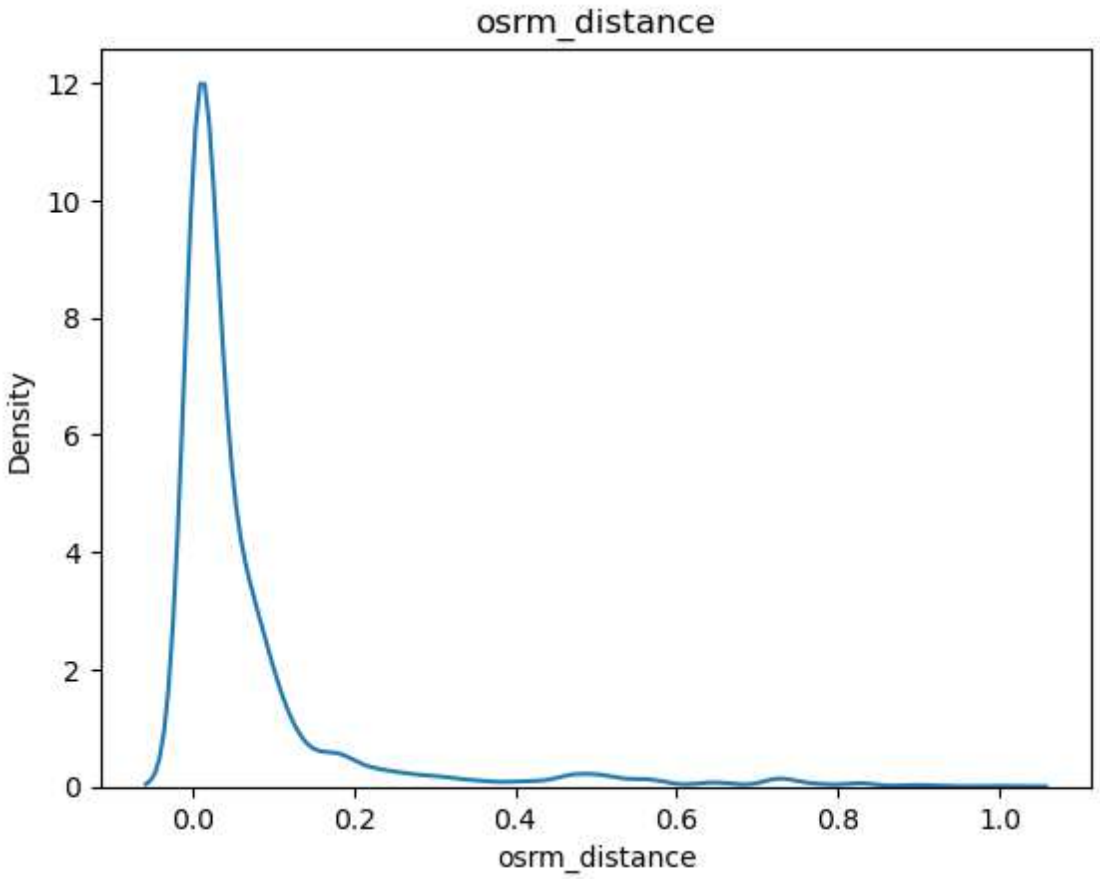
Out[273]:
```
MinMaxScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [275]:
```
Data['osrm_distance'] = mm.fit_transform(Data['osrm_distance'].to_num
Data['osrm_distance'][:3]
```

Out[275]:
```
0    0.346972
1    0.026859
2    0.828325
Name: osrm_distance, dtype: float64
```

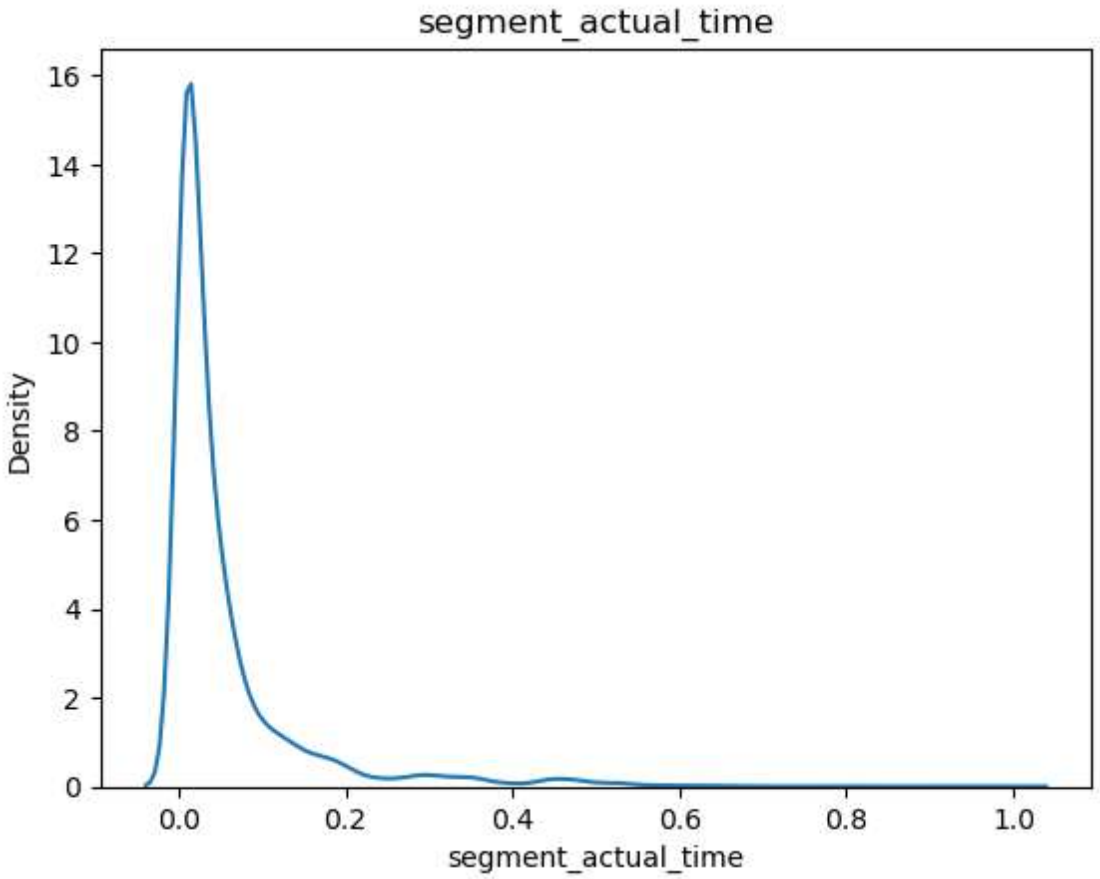
In [277]:
```
sns.kdeplot(Data['osrm_distance'])
plt.title("osrm_distance")
plt.show()
```



In [279]:
```
Data['segment_actual_time'] = mm.fit_transform(Data['segment_actual_t
Data['segment_actual_time'][:3]
```

Out[279]:
```
0    0.247388
1    0.021218
2    0.530301
Name: segment_actual_time, dtype: float64
```

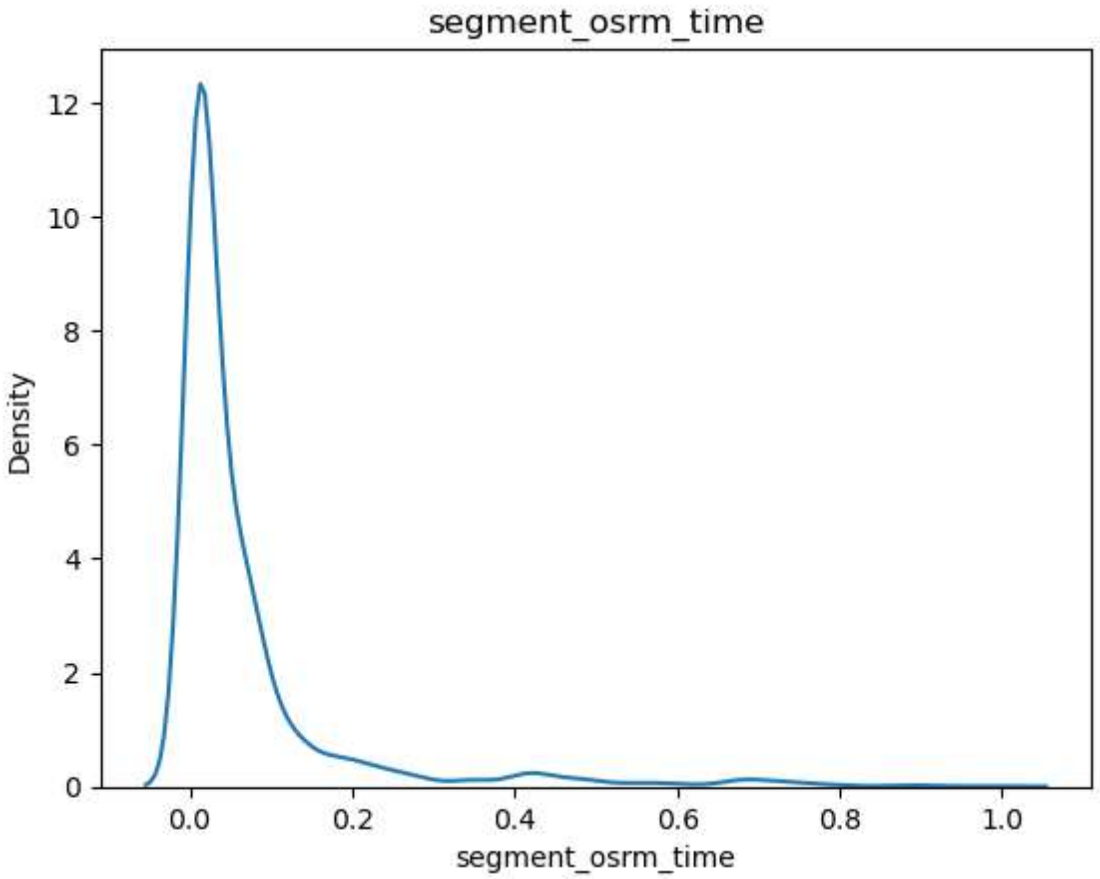
In [280]:
```
sns.kdeplot(Data['segment_actual_time'])
plt.title("segment_actual_time")
plt.show()
```



In [282]:
```
Data['segment_osrm_time'] = mm.fit_transform(Data['segment_osrm_time'
Data['segment_osrm_time'][:3]
```

Out[282]:
```
0    0.391712
1    0.023065
2    0.756450
Name: segment_osrm_time, dtype: float64
```

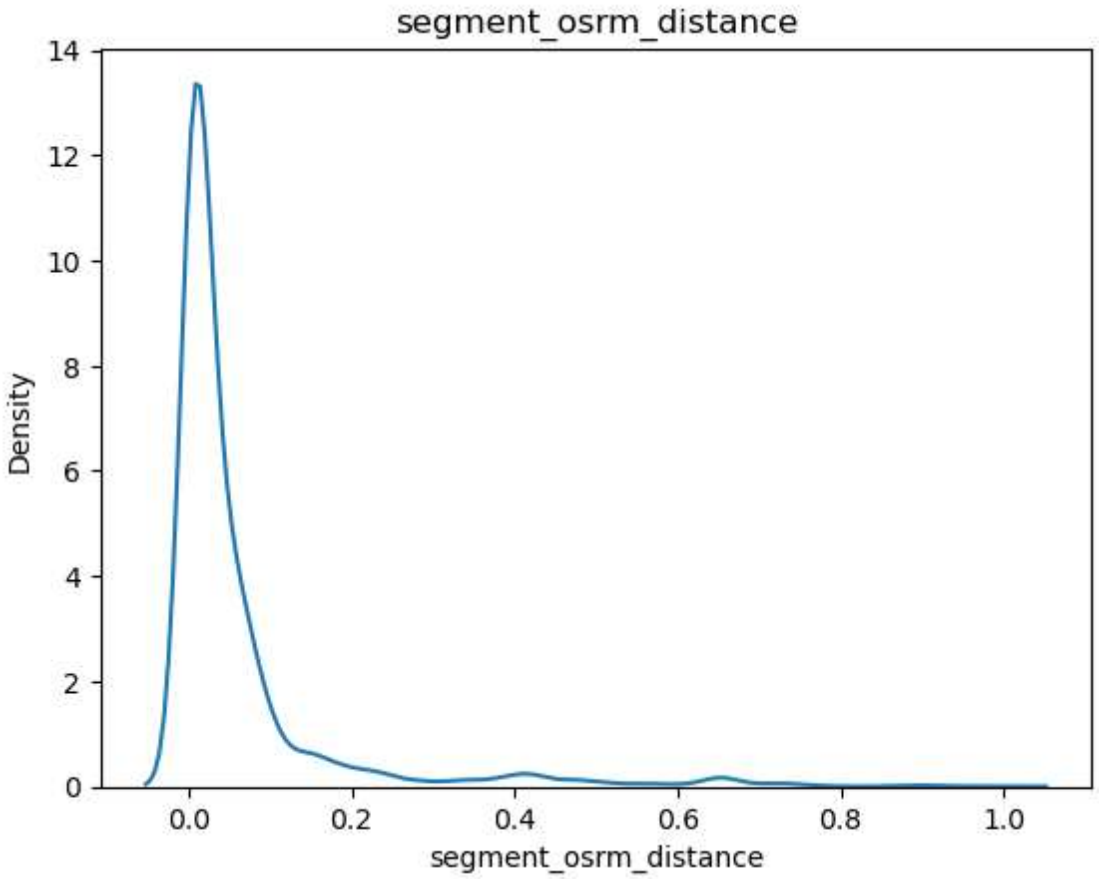
In [283]:
```
sns.kdeplot(Data['segment_osrm_time'])
plt.title("segment_osrm_time")
plt.show()
```



In [285]:
```
Data['segment_osrm_distance'] = mm.fit_transform(Data['segment_osrm_d
Data['segment_osrm_distance'][:3]
```

Out[285]:
```
0    0.373134
1    0.021373
2    0.721625
Name: segment_osrm_distance, dtype: float64
```

In [286]:
```
sns.kdeplot(Data['segment_osrm_distance'])
plt.title("segment_osrm_distance")
plt.show()
```



Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

Let's Analyse the Data from the Standard Scaler Function to check the Data comes out to be in Standard Format for the rest of the Analysis

This standardization also known as the z- score normalization, it aims to transform the data into the standard normal distribution – with the mean 0 and std deviation 1

In [293]:
```
Sdata = df2.copy()
Sdata.head(2)
```

Out[293]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [295]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss
```

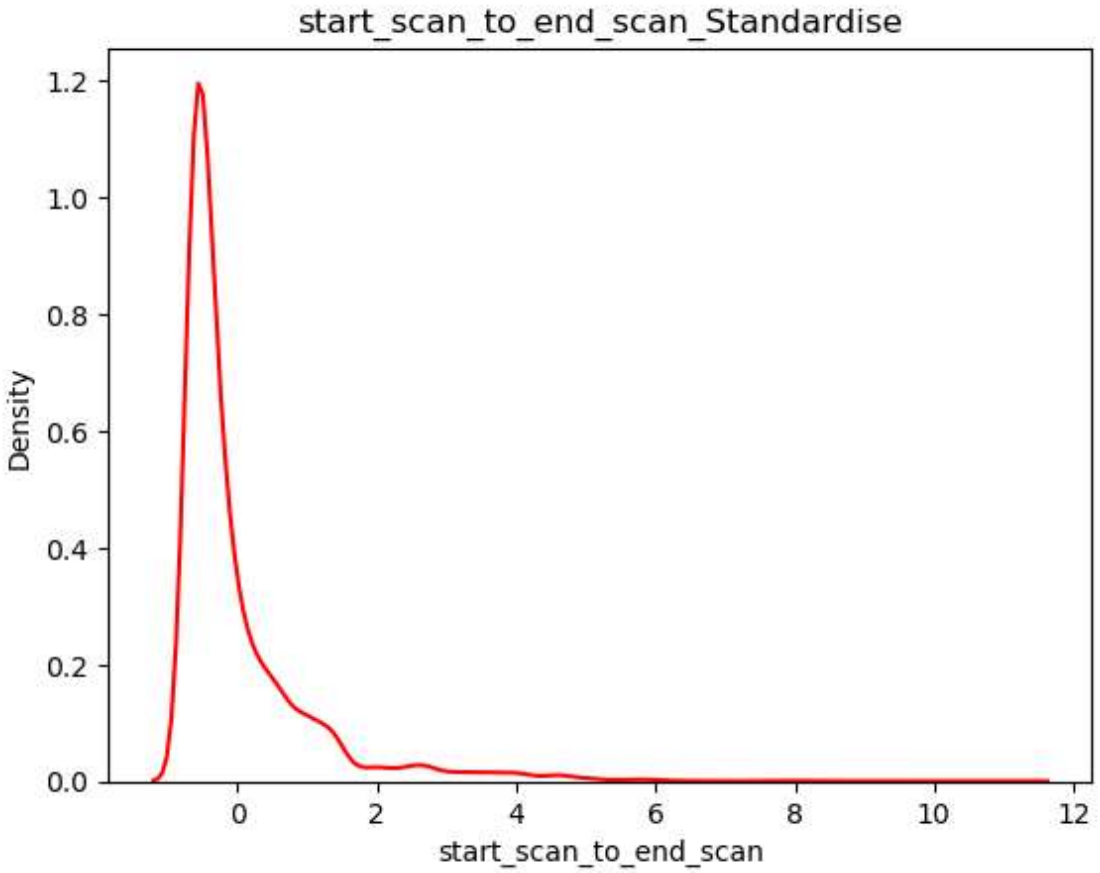
Out[295]:
```
StandardScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [303]:
```
Sdata['start_scan_to_end_scan'] = ss.fit_transform(Sdata['start_scan_
Sdata[:3]
```

Out[303]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018 00:00:33.6

In [309]:
```
sns.kdeplot(Sdata['start_scan_to_end_scan'], color="red")
plt.title("start_scan_to_end_scan_Standardise")
plt.show()
```

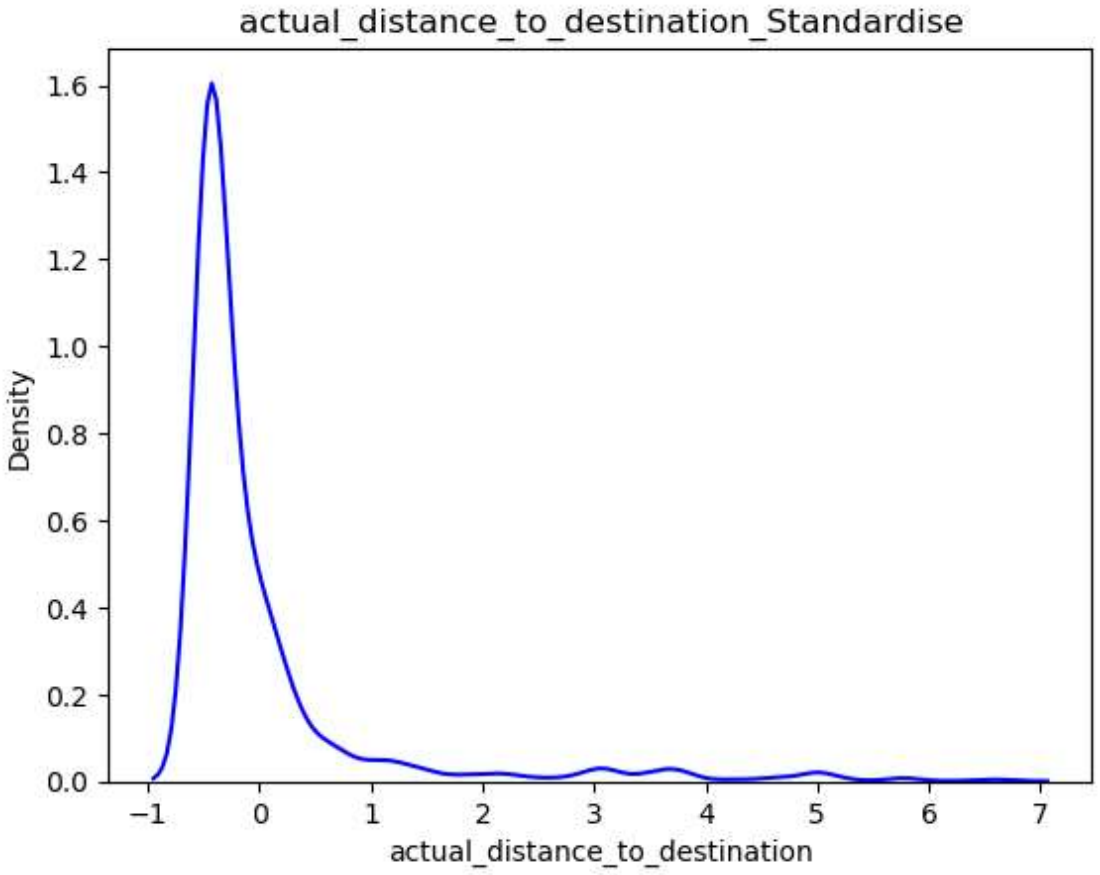


In [312]:
```
Sdata['actual_distance_to_destination'] = ss.fit_transform(Sdata['act
Sdata[:2]
```

Out[312]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [313]:
```
1 sns.kdeplot(Sdata['actual_distance_to_destination'], color="blue")
2 plt.title("actual_distance_to_destination_Standardise")
3 plt.show()
```

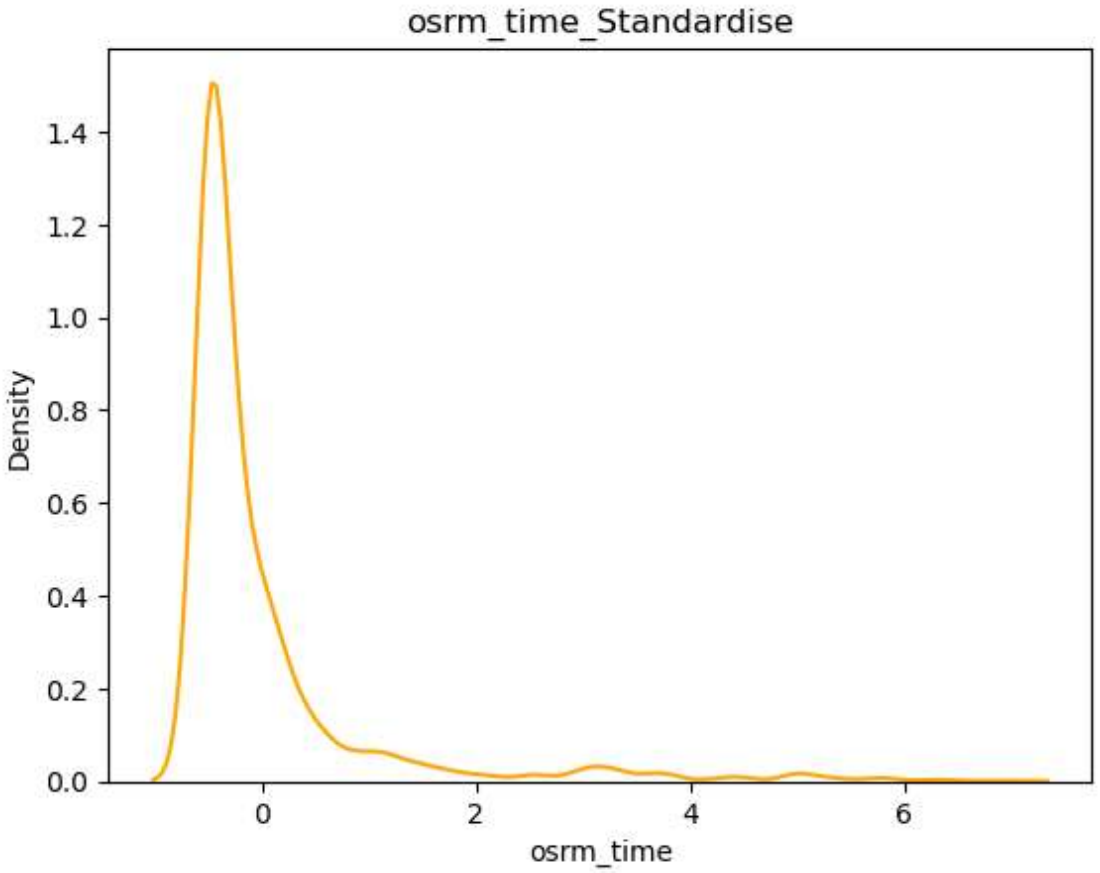


In [315]:
```
1 Sdata['osrm_time'] = ss.fit_transform(Sdata['osrm_time'].to_numpy().r
2 Sdata[:2]
```

Out[315]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [316]:
```
sns.kdeplot(Sdata['osrm_time'], color="orange")
plt.title("osrm_time_Standardise")
plt.show()
```

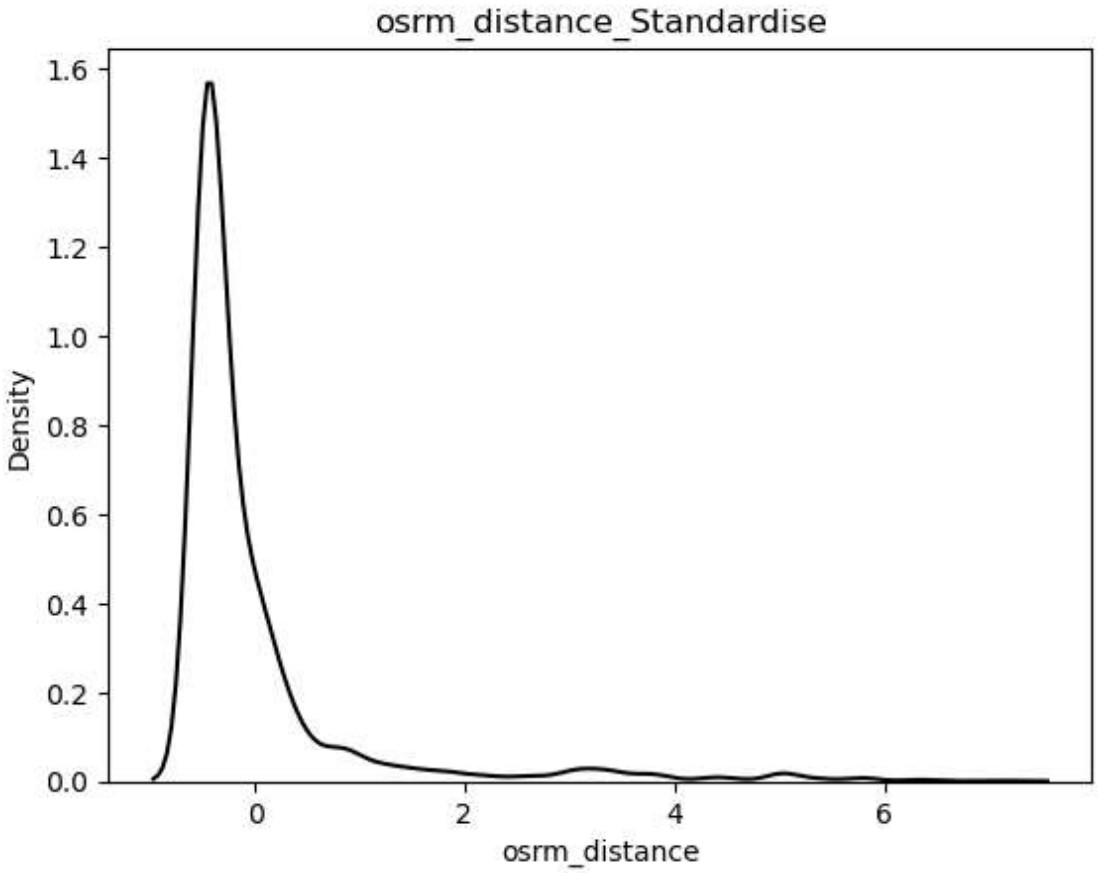


In [318]:
```
Sdata['osrm_distance'] = ss.fit_transform(Sdata['osrm_distance'].to_n
Sdata[:2]
```

Out[318]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [320]:
```
sns.kdeplot(Sdata['osrm_distance'], color="black")
plt.title("osrm_distance_Standardise")
plt.show()
```

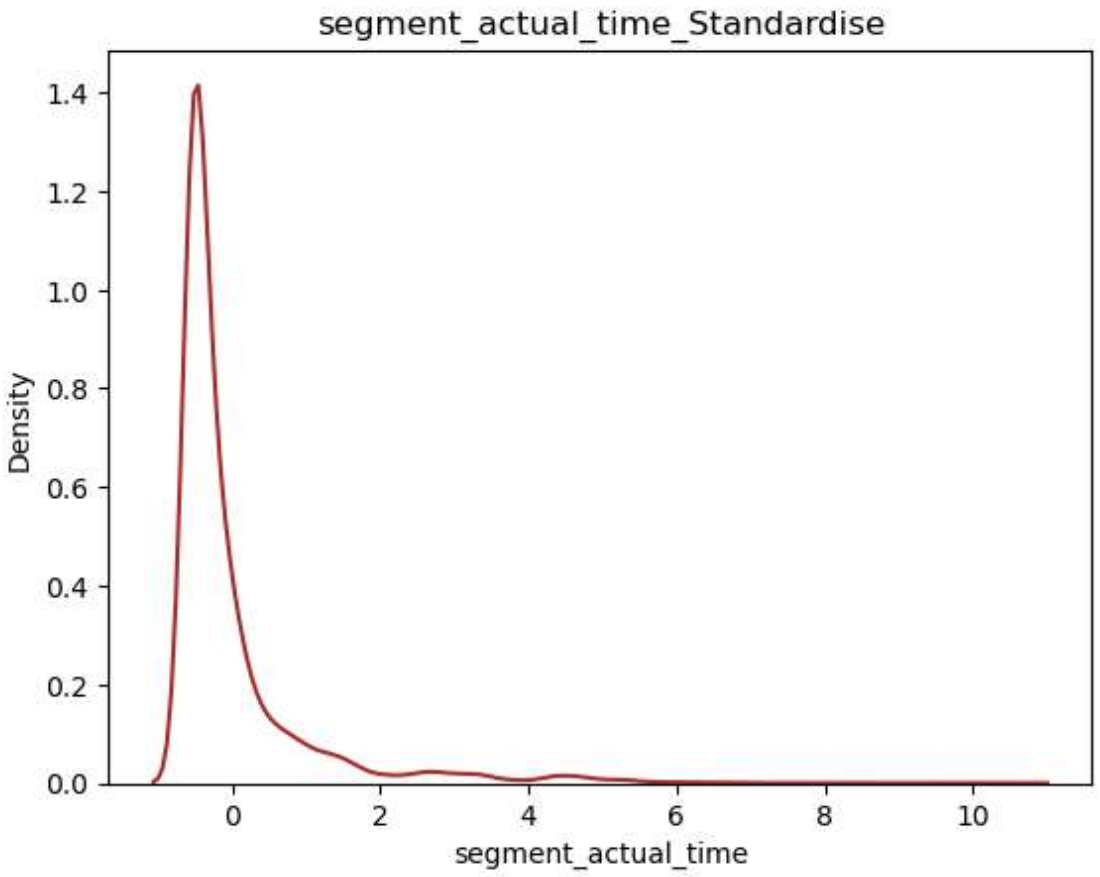


In [322]:
```
Sdata['segment_actual_time'] = ss.fit_transform(Sdata['segment_actual
Sdata[:2]
```

Out[322]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [324]:
```
sns.kdeplot(Sdata['segment_actual_time'], color="brown")
plt.title("segment_actual_time_Standardise")
plt.show()
```

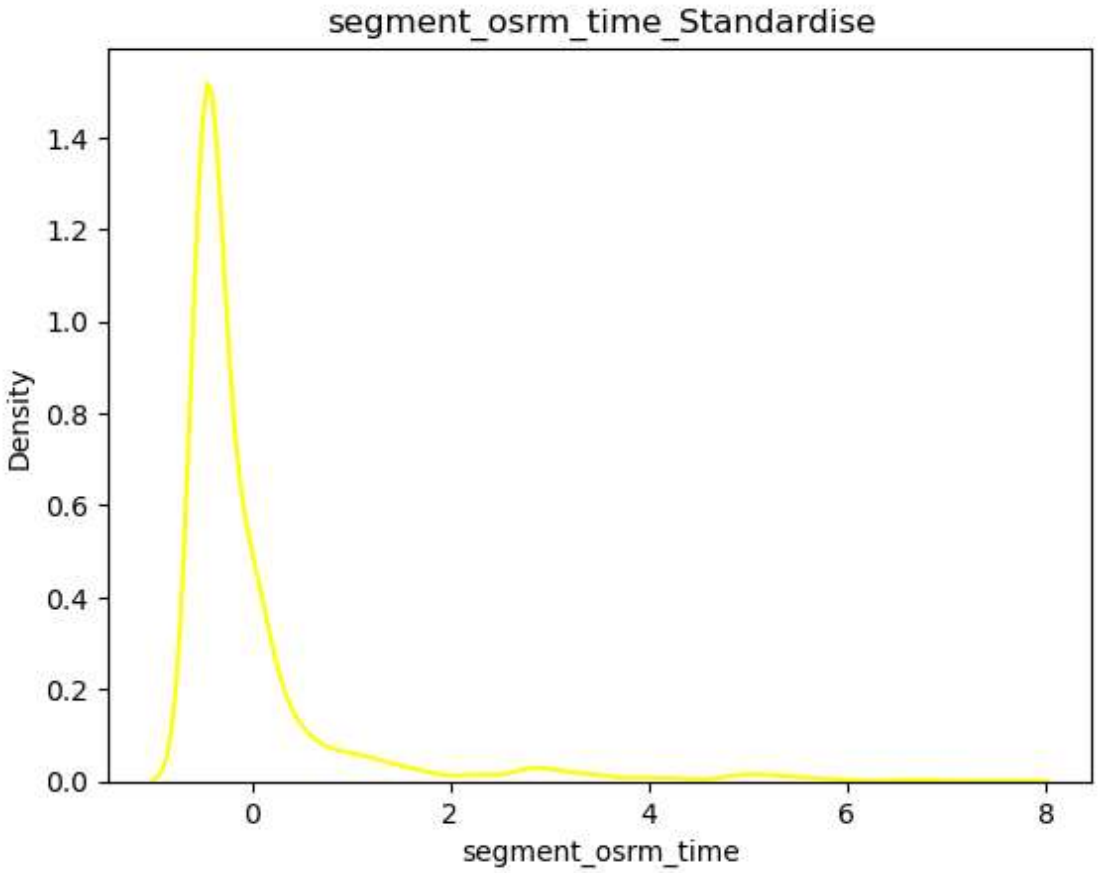


In [326]:
```
Sdata['segment_osrm_time'] = ss.fit_transform(Sdata['segment_osrm_tim
Sdata[:2]
```

Out[326]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [328]:
```
sns.kdeplot(Sdata['segment_osrm_time'], color="yellow")
plt.title("segment_osrm_time_Standardise")
plt.show()
```

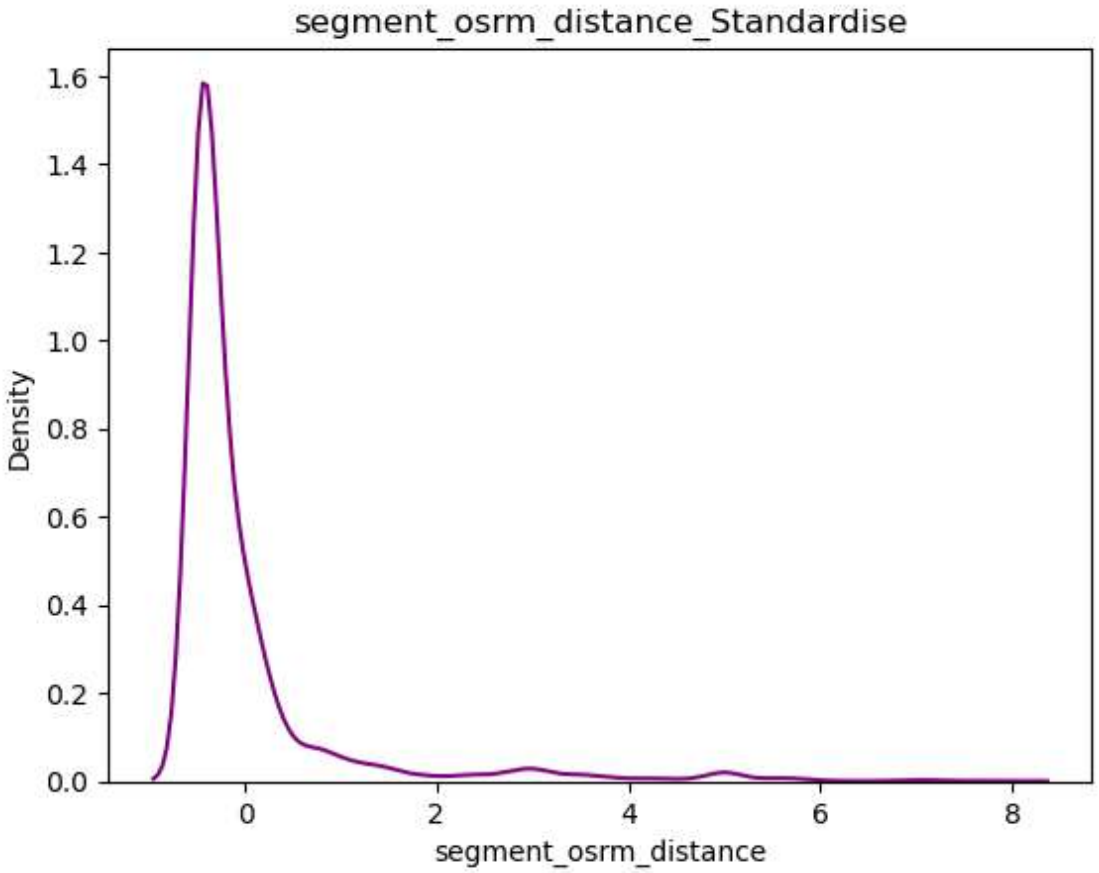


In [335]:
```
Sdata['segment_osrm_distance'] = ss.fit_transform(Sdata['segment_osrm
Sdata[:2]
```

Out[335]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creatior
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018 00:00:16.5
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018 00:00:22.8

In [336]:
```
sns.kdeplot(Sdata['segment_osrm_distance'], color="purple")
plt.title("segment_osrm_distance_Standardise")
plt.show()
```



Hypothesis Testing:

Perform hypothesis testing / visual analysis between : actual_time aggregated value and OSRM time aggregated value.

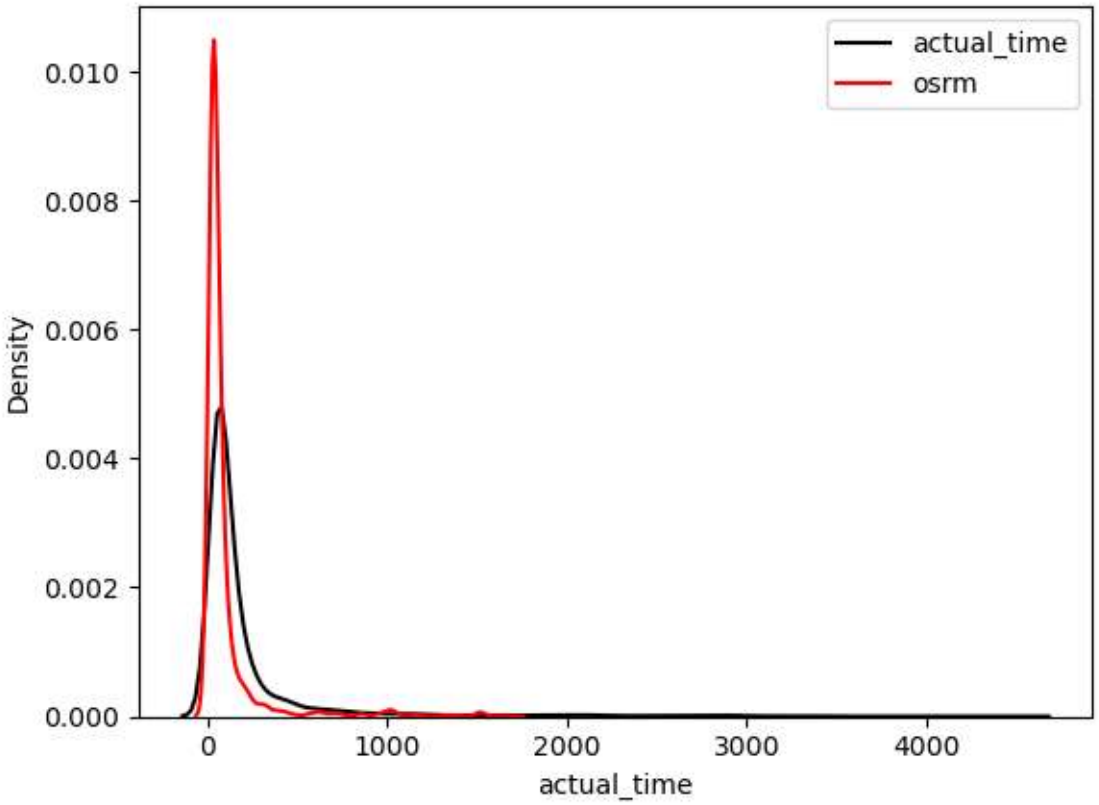
In [345]:
```
1 df1[['actual_time','osrm_time']]
```

Out[345]:

	actual_time	osrm_time
0	732.0	329.0
1	830.0	388.0
2	47.0	26.0
3	96.0	42.0
4	611.0	212.0
...	...	...
26363	51.0	41.0
26364	90.0	48.0
26365	30.0	14.0
26366	233.0	42.0
26367	42.0	26.0

26368 rows × 2 columns

In [384]:
```
sns.kdeplot(df1['actual_time'], color="black", label = "actual_time")
sns.kdeplot(df1['osrm_time'], color="red",label = "osrm")
plt.legend()
plt.show()
```




```
Here, from the Data we can see the both comes into the Numerical
Columns. so, the hypothesis testing for the both of the
columns are fall into the TTest and the actual_time - Actual time taken
to complete the delivery (Cumulative) and osrm_time - An open-source
routing engine time calculator which computes the
shortest path between points in a given map (Includes usual traffic,
distance through
major and minor roads) and gives the time (Cumulative) are performing
the indivisual behaviour, so we use here the
Ttest_ind for the Hypothesis Analysis
```

The Above graph does not statisfy the Normal Distribution, lets check the Data to form a Normal Distribution with the Help of the Shapiro wilk test or the Boxcox test

```
H0: The sample follows normal distribution
Ha: The sample does not follow normal distribution

alpha = 0.05

```

```
Test Statistics : Shapiro-Wilk test for normality
```

In [546]:
```
stats , pvalue = shapiro(df['actual_time'].sample(5000))
print(f"The stats value is: {stats} and the p_value comes to be {pval
alpha = 0.05
if pvalue < alpha:
    print("The sample does not follow normal distribution")
else:
    print("The sample follows normal distribution")
```

```
The stats value is: 0.6978424787521362 and the p_value comes to be 0.0
The sample does not follow normal distribution
```

Let's Analyse with the help of the Boxcox test

In [549]:
```
transform_actual_time = boxcox(df['actual_time'])[0]
stats, pvalue = shapiro(transform_actual_time)
print(f"The stats value is: {stats} and the p_value comes to be {pval
alpha = 0.05
if pvalue < alpha:
    print("The sample does not follow normal distribution")
else:
    print("The sample follows normal distribution")
```

```
The stats value is: 0.9715785980224609 and the p_value comes to be 0.0
The sample does not follow normal distribution
```

```
From the Both of the test, we find as, the data doesnot follow the
normal Distribution
```

In [356]:
```
df1['actual_time'].mean() , df1['osrm_time'].mean()
```

Out[356]: (200.690192657767, 90.68670358009709)

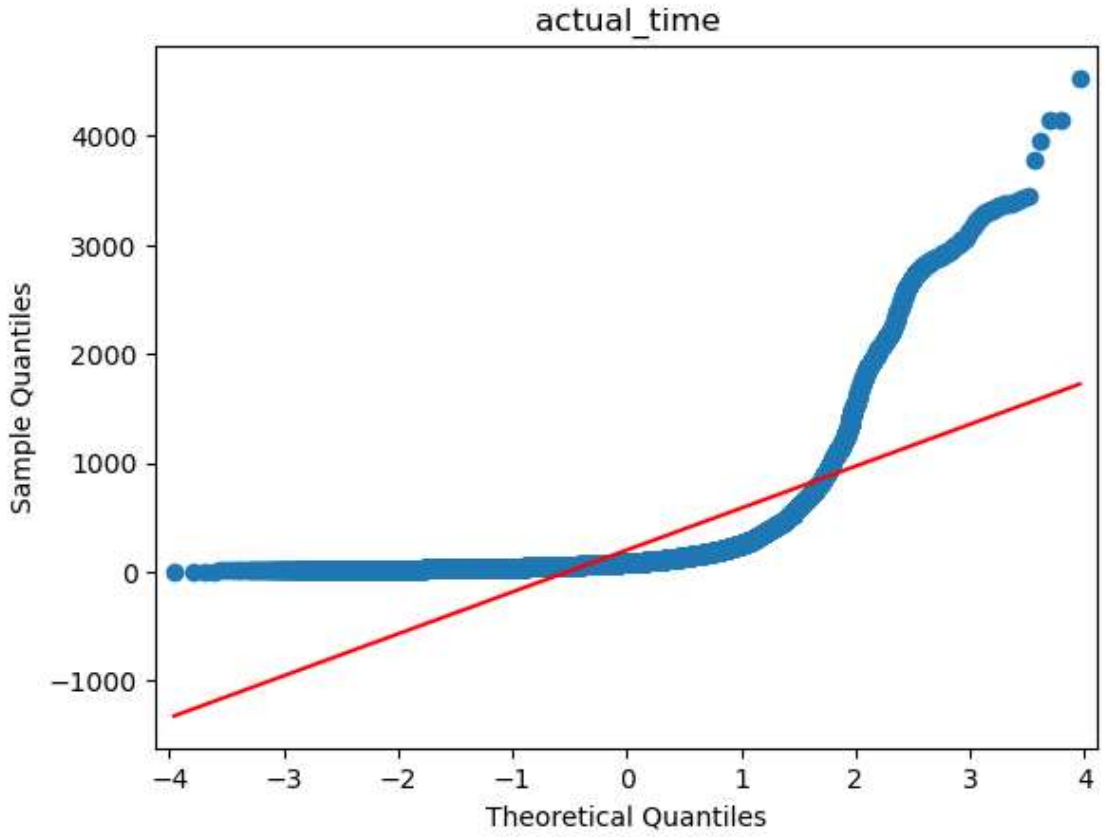
```
Let's setup the framework for the hypothesis testing for analysing the
actual time and osrm time, as we include the
mean of both of them.

H0: The mean between the both of the values is same ie. (mu1 = mu2)
    Ha: The mean between the both of the values are different i.e (mu1
!= mu2) or (mu1 > mu2)
```

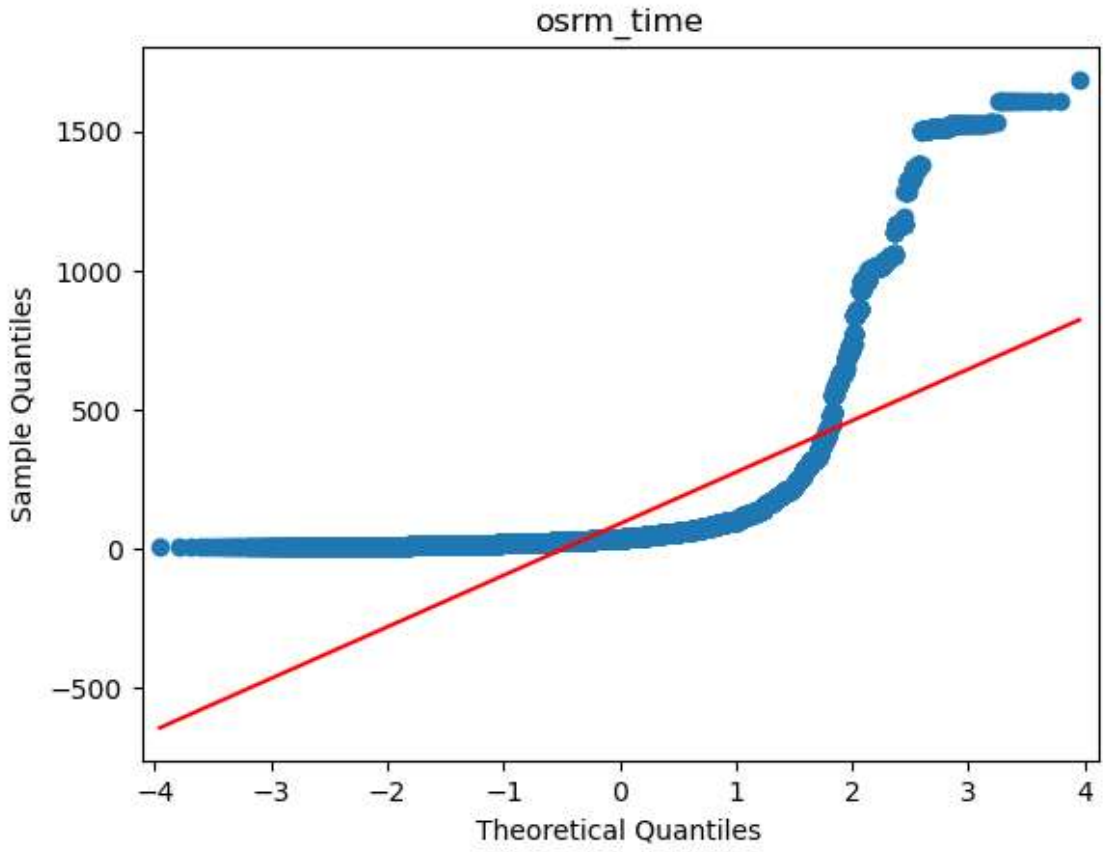
```
Also, lets have the analysis for the confidence level at 5% and
significance value at the 95%.
Where the alpha = 0.05
```

In [550]:

```
sm.qqplot(df1['actual_time'], line='s')
plt.title("actual_time")
plt.show()
```



In [551]:
```
sm.qqplot(df1['osrm_time'], line='s')
plt.title("osrm_time")
plt.show()
```



In [366]:
```
alpha  = 0.05
statistic, pvalue = ttest_ind(df1['actual_time'], df1['osrm_time'])
print(f"The statistic value is : {statistic} and the P_value is : {pv
if pvalue < alpha:
    print(f"Reject the Null hypothesis at : {pvalue}, the values for
else:
    print(f"Failed to Reject H0 at : {pvalue}, the values are Both of
```

```
The statistic value is : 41.82845508363711 and the P_value is : 0.0
Reject the Null hypothesis at : 0.0, the values for the Both of the colu
mns are Not same
```

Hypothesis testing Actual_time aggregated value and segment actual time aggregated value.

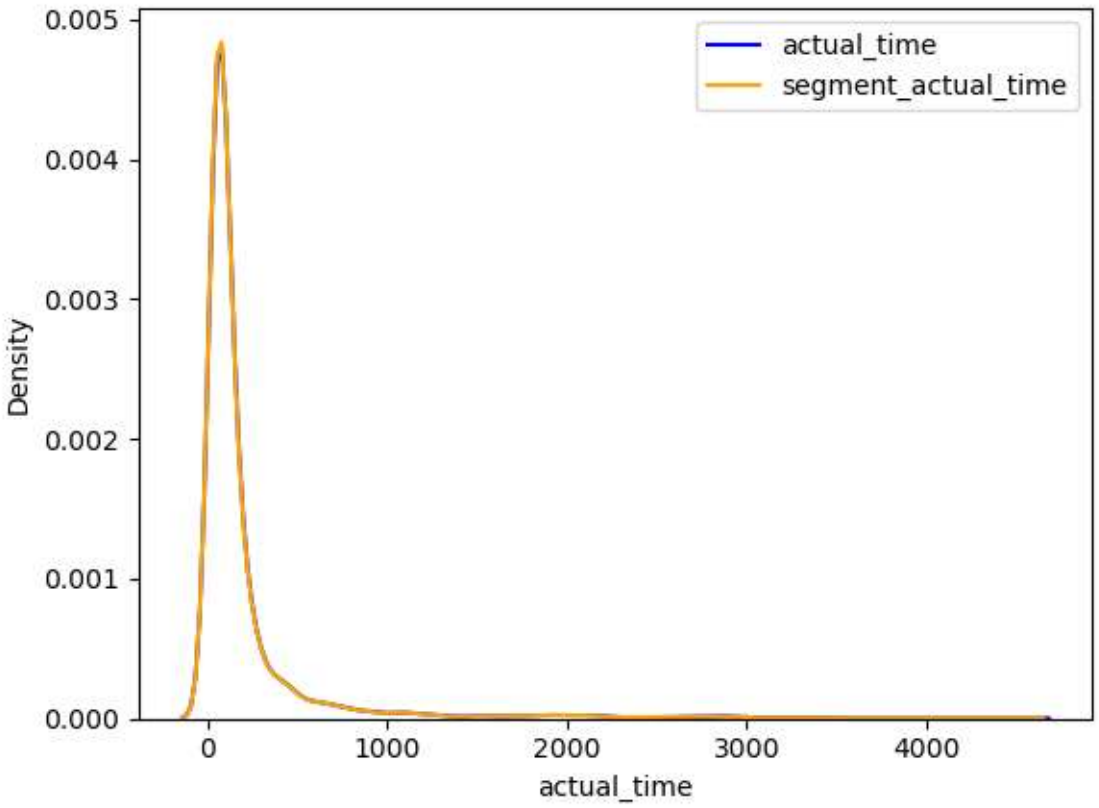
In [403]:
```
1 df1[['actual_time', 'segment_actual_time']]
```

Out[403]:

	actual_time	segment_actual_time
0	732.0	728.0
1	830.0	820.0
2	47.0	46.0
3	96.0	95.0
4	611.0	608.0
...	...	...
26363	51.0	49.0
26364	90.0	89.0
26365	30.0	29.0
26366	233.0	233.0
26367	42.0	41.0

26368 rows × 2 columns

In [408]:
```
1 sns.kdeplot(df1['actual_time'], color="blue", label = "actual_time")
2 sns.kdeplot(df1['segment_actual_time'], color="orange",label = "segme
3 plt.legend()
4 plt.show()
```



```
From the above graph we can depict that, the data from both of the columns are overlapping to each other as it says, the
Data are lies in the same direction with the same point of values, where the both columns are indepedent to each other.
so we use the Ttest_ind for the Hypotheis testing and predict the level of the homogenous between the columns which are
actual time and Segment actual time.
```

The Above graph does not statisfy the Normal Distribution, lets check the Data to form a Normal Distribution with the Help of the Shapiro wilk test or the Boxcox test

```
H0: The sample follows normal distribution
Ha: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality
```

In [552]:

```
stats , pvalue = shapiro(df['segment_actual_time'].sample(5000))
print(f"The stats value is: {stats} and the p_value comes to be {pval
alpha = 0.05
if pvalue < alpha:
    print("The sample does not follow normal distribution")
else:
    print("The sample follows normal distribution")
```

```
The stats value is: 0.3264663815498352 and the p_value comes to be 0.0
The sample does not follow normal distribution
```

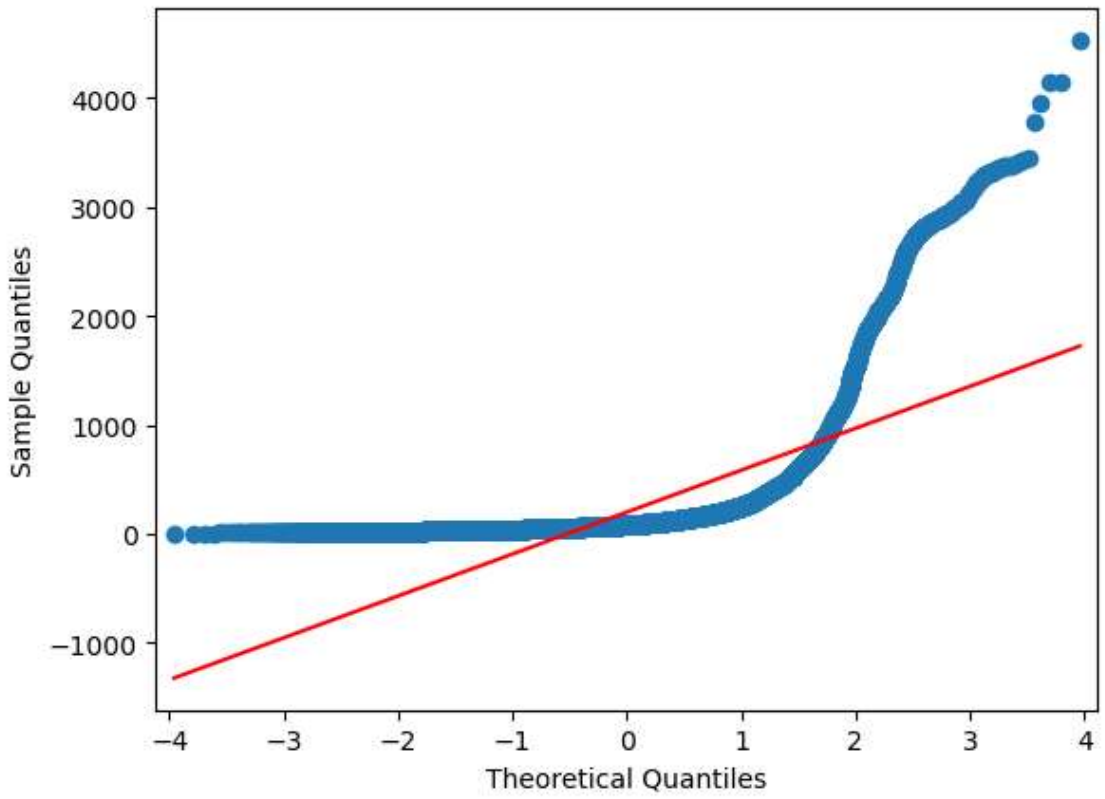
In [409]:

```
df1['actual_time'].mean(), df1['segment_actual_time'].mean()
```

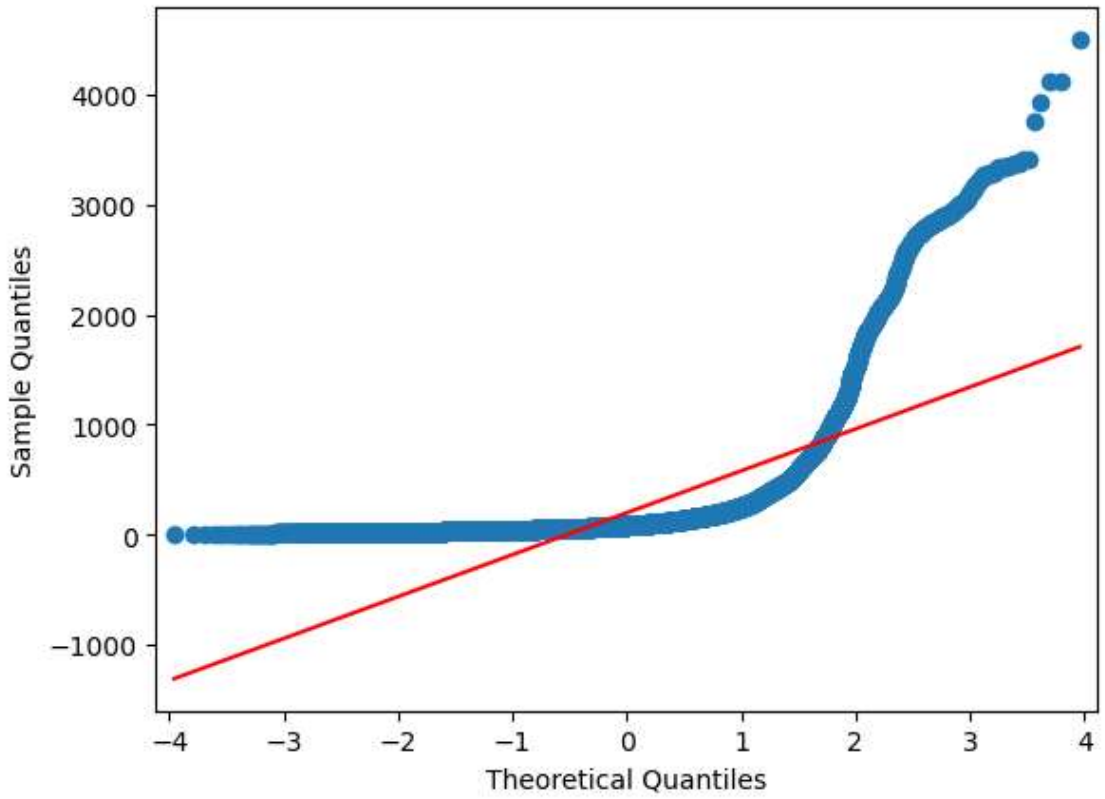
Out[409]: (200.690192657767, 198.8964274878641)

```
From the above means says the, the values for the Graph almost lies on the same predicted values, the mean between
both of the columns are on the same page as the difference between them is too low.
Let's Analyse the Above columns in the Form QQ plot to analyse the Distribution is Gaussian or not.
```

In [410]:
```
sm.qqplot(df1['actual_time'], line='s')
plt.show()
```



In [411]:
```
sm.qqplot(df1['segment_actual_time'], line='s')
plt.show()
```



```
Le'ts Analyse the columns with the Hypothesis testing by creating the Framework between them.
H0: The mean between the Columns (Actual) and (segment_actual) are same (mu1 = mu2)
Ha: The mean between the (Actual) and (Segment_actual) are not same (mu1 != mu2)
    Also, lets have the analysis for the confidence level at 5% and significance value at the 95%.
    Where the alpha = 0.05
```

In [414]:

```
alpha  = 0.05
statistic, pvalue = ttest_ind(df1['actual_time'], df1['segment_actual
print(f"The statistic value is : {statistic} and the P_value is : {pv
if pvalue < alpha:
    print(f"Reject the Null hypothesis at : {pvalue}, the values for
else:
    print(f"Failed to Reject H0 at : {pvalue}, the values are Both of
print(" ** This Includes the Actual time and Segmented Actual time ar
```

```
The statistic value is : 0.5376121149549357 and the P_value is : 0.59084
71331390668
Failed to Reject H0 at : 0.5908471331390668, the values are Both of the
time is same
 ** This Includes the Actual time and Segmented Actual time are the on t
he same Distributed values **
```

Hypothesis Testing between OSRM distance aggregated value and segment OSRM distance aggregated value.

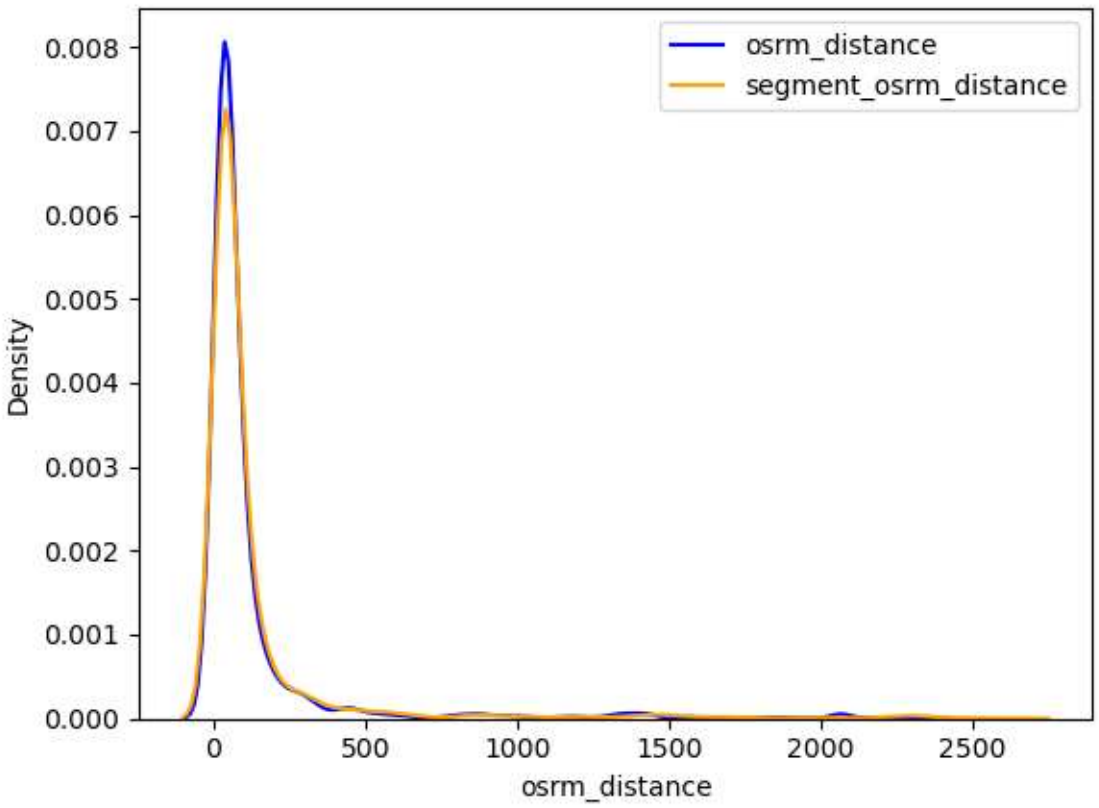
In [418]:
```
df1[['osrm_distance', 'segment_osrm_distance']]
```

Out[418]:

	osrm_distance	segment_osrm_distance
0	446.5496	670.6205
1	544.8027	649.8528
2	28.1994	28.1995
3	56.9116	55.9899
4	281.2109	317.7408
...	...	...
26363	42.5213	42.1431
26364	40.6080	78.5869
26365	16.0185	16.0184
26366	52.5303	52.5303
26367	28.0484	28.0484

26368 rows × 2 columns

In [420]:
```
sns.kdeplot(df1['osrm_distance'], color="blue", label = "osrm_distanc
sns.kdeplot(df1['segment_osrm_distance'], color="orange",label = "seg
plt.legend()
plt.show()
```



```
From the above graph we can depict the slight difference between the
osrm distance and segment distance where the mean falls on the same
ground and but the density of the osrm distance comes more than the
segment distance. The both of the column which is osrm distance and
segment distance are the individual behaviour.
Here, we check fot the ttest for both of the indivisual numerical
columns and classify the test to check the differences between.
```

The Above graph does not statisfy the Normal Distribution, lets check the Data to form a Normal Distribution with the Help of the Shapiro wilk test or the Boxcox test

```
H0: The sample follows normal distribution
Ha: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality
```

In [556]:

```
stats , pvalue = shapiro(df['osrm_time'].sample(5000))
print(f"The stats value is: {stats} and the p_value comes to be {pval
alpha = 0.05
if pvalue < alpha:
    print("The sample does not follow normal distribution")
else:
    print("The sample follows normal distribution")
```

```
The stats value is: 0.6909828186035156 and the p_value comes to be 0.0
The sample does not follow normal distribution
```

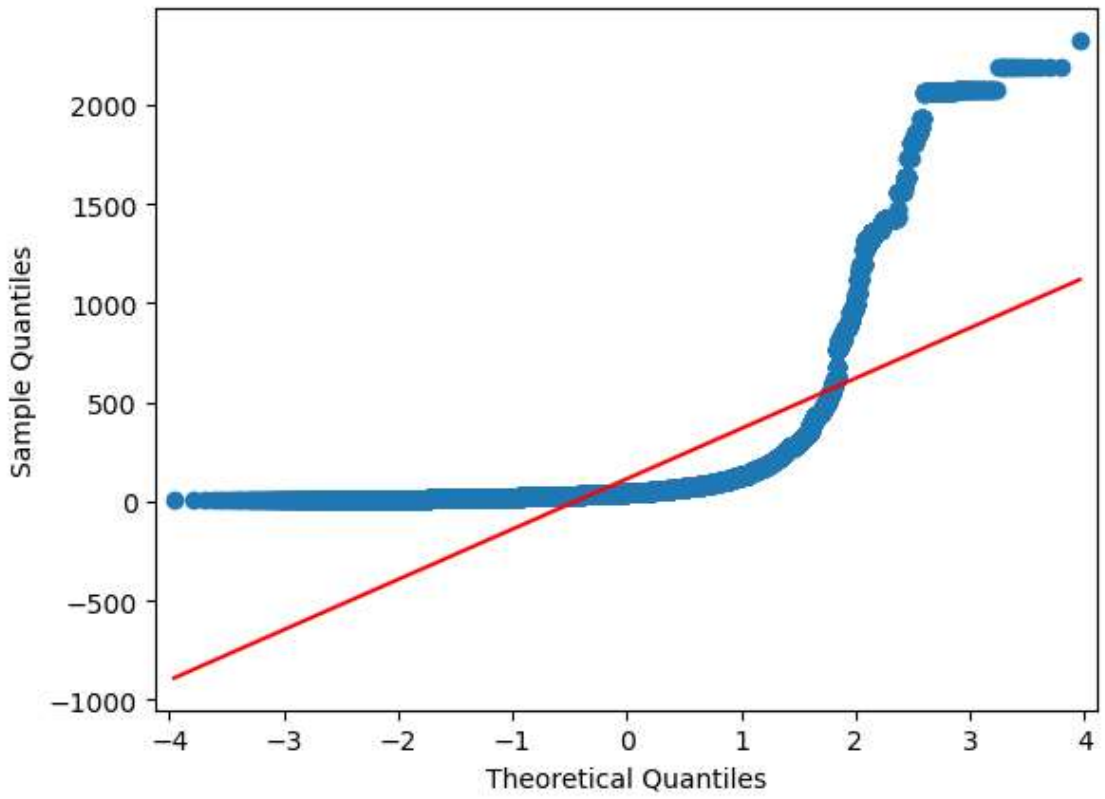
In [421]:

```
df1['osrm_distance'].mean(), df1['segment_osrm_distance'].mean()
```

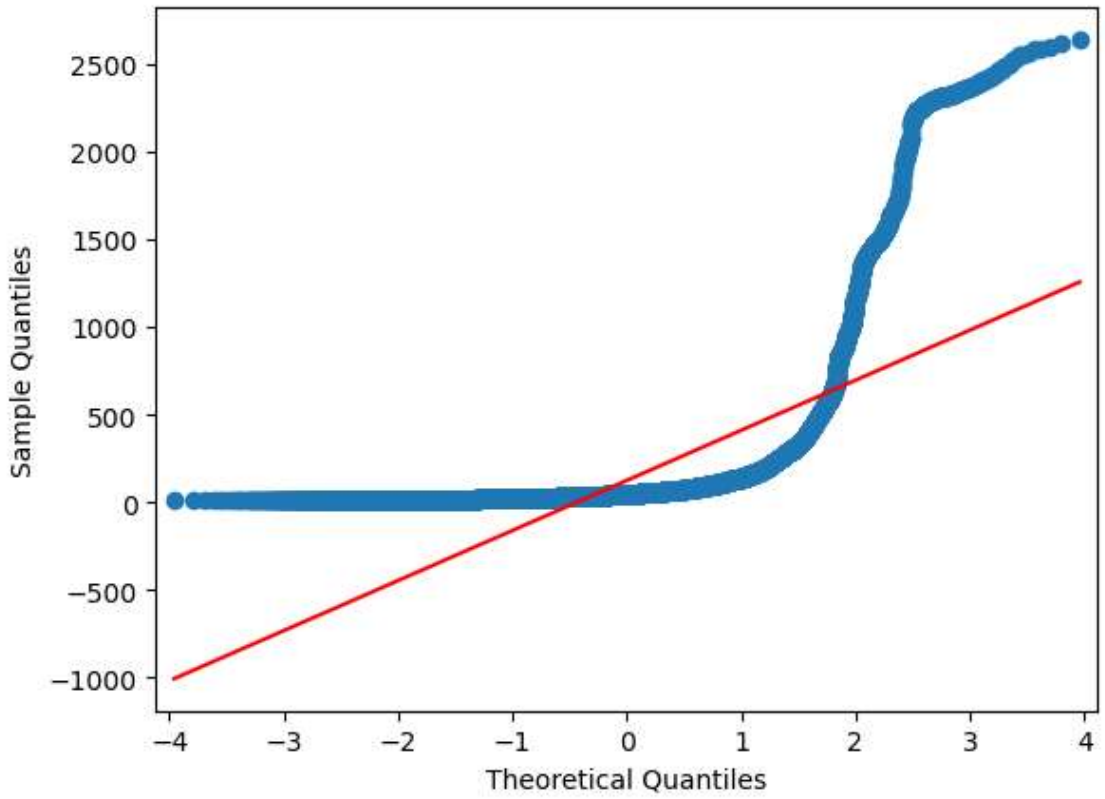
Out[421]: (114.82764181962986, 125.40279846404732)

```
From the above means says the, the values for the Graph lies more the
segment osrm value on the same predicted values, the mean between
both of the columns are not on the same page as the difference between
them is too low.
Let's Analyse the Above columns in the Form QQ plot to analyse the
Distribution is Gaussian or not.
```

In [423]:
```
1 sm.qqplot(df1['osrm_distance'], line='s')
2 plt.show()
```



In [424]:
```
1 sm.qqplot(df1['segment_osrm_distance'], line='s')
2 plt.show()
```



```
Let's perform the Ttest_ind for the osrm distance and segment distance
for the hypotheis testing for numerical columns.
Let's suppose the significance value for the testing is 95% with the
confidence level 5%.
which belongs to the alpha = 0.05%
Framework:
H0: The osrm distance and segment distance is same (mu1 = mu2)
    Ha: The osrm distance and segment distance is not same (mu1 !=
mu2)
```

In [426]:

```
alpha  = 0.05
statistic, pvalue = ttest_ind(df1['osrm_distance'], df1['segment_osrm
print(f"The statistic value is : {statistic} and the P_value is : {pv
if pvalue < alpha:
    print(f"Reject the Null hypothesis at : {pvalue}, the mean values
else:
    print(f"Failed to Reject H0 at : {pvalue}, the Mean values are Bo
print(" ** This Includes the Osrm Distance and Segment Distance are t
```

```
The statistic value is : -4.4923128284182905 and the P_value is : 7.0603
22481862229e-06
Reject the Null hypothesis at : 7.060322481862229e-06, the mean values f
or the Both of the columns are Not same
 ** This Includes the Osrm Distance and Segment Distance are the on the
different Distributed values **
```

Hypothesis Testing for OSRM time aggregated value and segment OSRM time aggregated value.

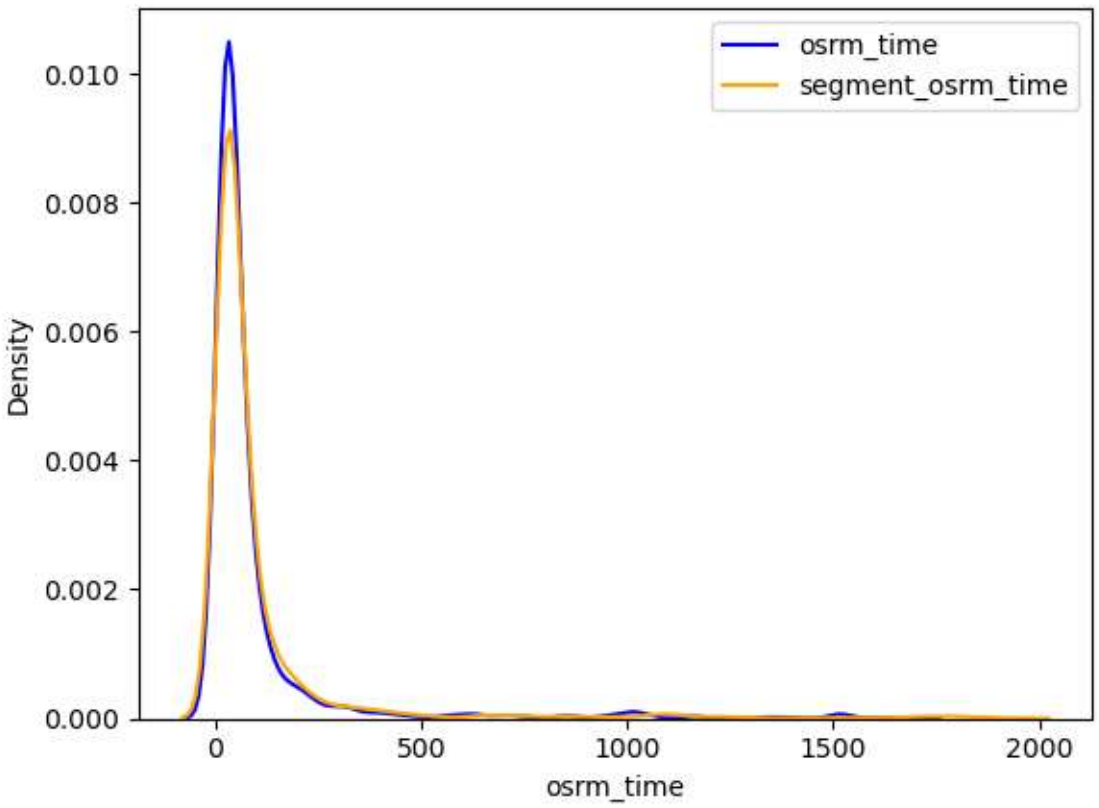
In [428]:
```
1 df1[['osrm_time','segment_osrm_time']]
```

Out[428]:

	osrm_time	segment_osrm_time
0	329.0	534.0
1	388.0	474.0
2	26.0	26.0
3	42.0	39.0
4	212.0	231.0
...	...	...
26363	41.0	42.0
26364	48.0	77.0
26365	14.0	14.0
26366	42.0	42.0
26367	26.0	25.0

26368 rows × 2 columns

In [429]:
```
1 sns.kdeplot(df1['osrm_time'], color="blue", label = "osrm_time")
2 sns.kdeplot(df1['segment_osrm_time'], color="orange",label = "segment_
3 plt.legend()
4 plt.show()
```



```
The graph indicates overlapping data in both columns, particularly highlighting higher density in the "osrm time" column. The data points align in the same direction with similar values, suggesting independence between the two columns. To assess the homogeneity level between "osrm_time" and "Segment osrm_time," we employ the Ttest_ind for hypothesis testing.
```

The Above graph does not statisfy the Normal Distribution, lets check the Data to form a Normal Distribution with the Help of the Shapiro wilk test or the Boxcox test

```
H0: The sample follows normal distribution
Ha: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality
```

In [557]:

```
stats , pvalue = shapiro(df['segment_osrm_time'].sample(5000))
print(f"The stats value is: {stats} and the p_value comes to be {pval
alpha = 0.05
if pvalue < alpha:
    print("The sample does not follow normal distribution")
else:
    print("The sample follows normal distribution")
```

```
The stats value is: 0.7032755613327026 and the p_value comes to be 0.0
The sample does not follow normal distribution
```

In [433]:

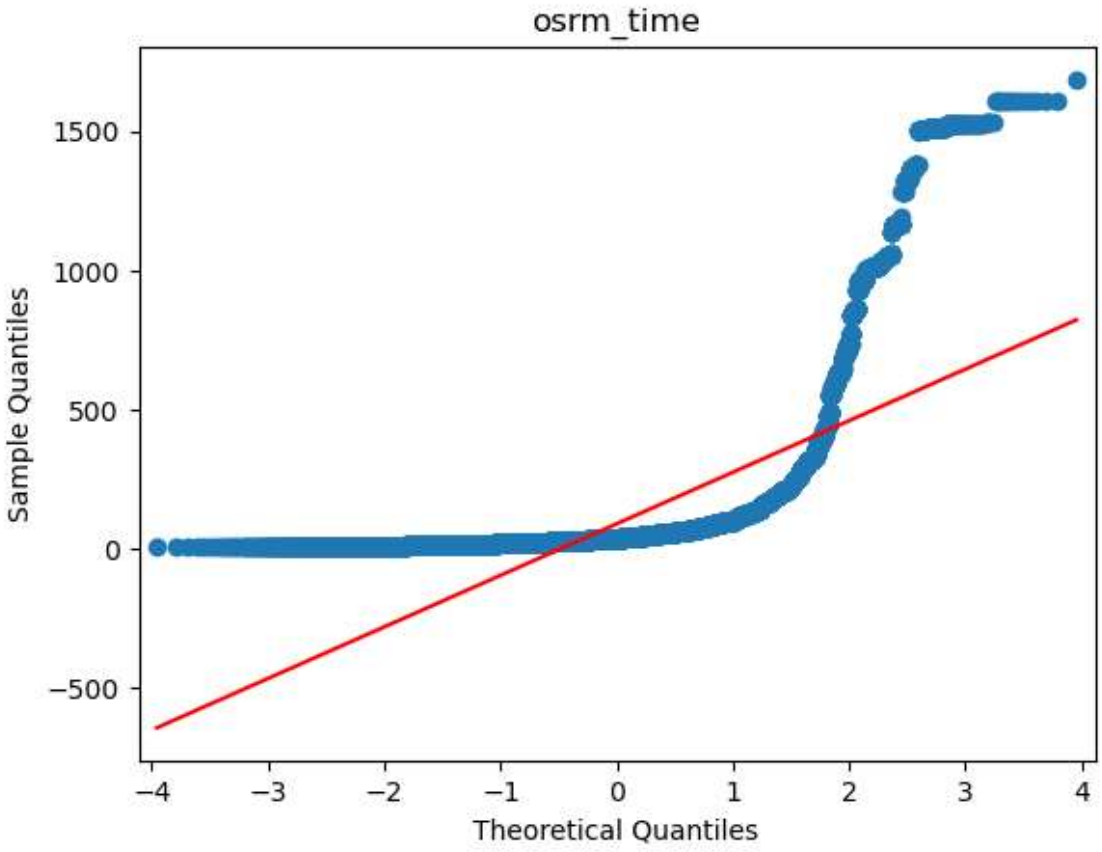
```
df1['osrm_time'].mean(), df1['segment_osrm_time'].mean()
```

Out[433]: (90.68670358009709, 101.66523816747574)

```
From the above means says the, the values for the Graph almost lies on the same predicted values, the mean between
both of the columns are on the same page slighter more the segmented osrm time as the difference between them is too low.
Let's Analyse the Above columns in the Form QQ plot to analyse the Distribution is Gaussian or not.
```

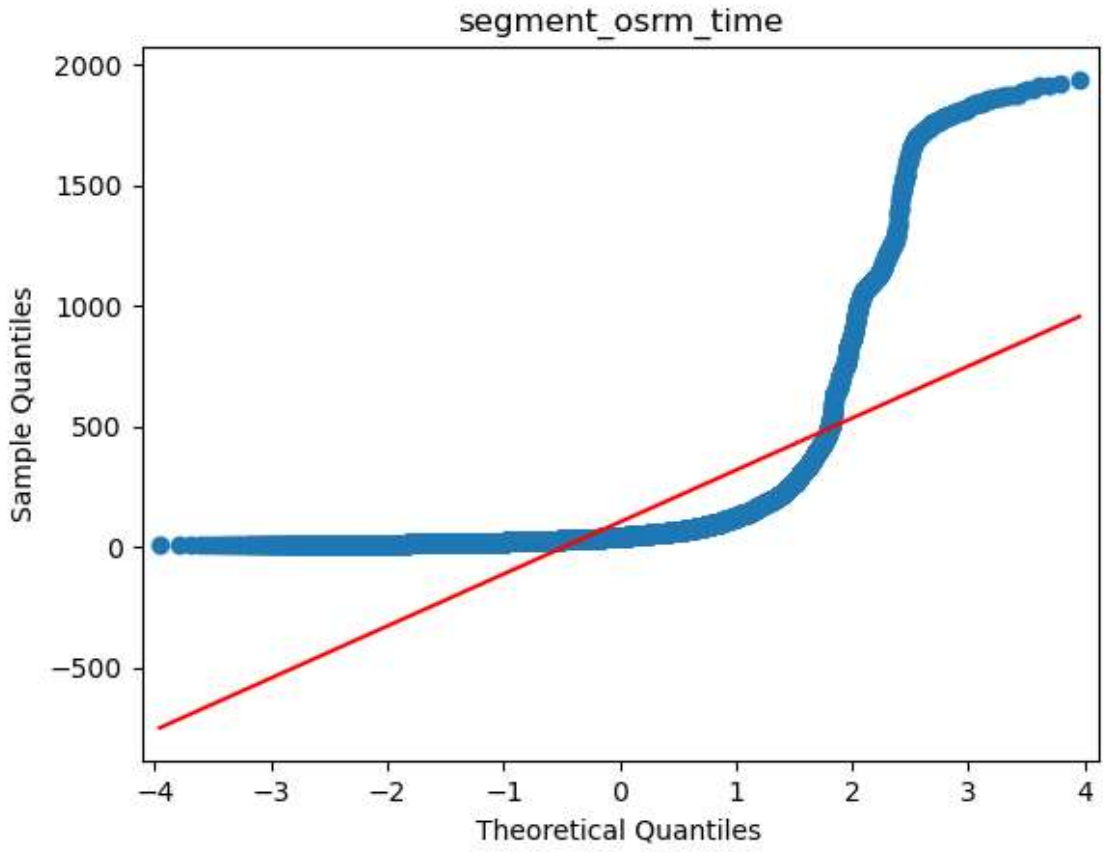
In [436]:

```
sm.qqplot(df1['osrm_time'], line='s')
plt.title("osrm_time")
plt.show()
```



In [437]:

```
sm.qqplot(df1['segment_osrm_time'], line='s')
plt.title("segment_osrm_time")
plt.show()
```




```
Let's Analyse the columns with the Hypothesis testing by creating the Framework between them.
H0: The mean between the Columns (osrm time) and (segment osrm time) are same (mu1 = mu2)
Ha: The mean between the (osrm time) and (segment osrm time) are not same (mu1 != mu2)
    Also, lets have the analysis for the confidence level at 95% and significance value at the 5%.
    Where the alpha = 0.05
```

In [438]:
```
alpha  = 0.05
statistic, pvalue = ttest_ind(df1['osrm_time'], df1['segment_osrm_tim
print(f"The statistic value is : {statistic} and the P_value is : {pv
if pvalue < alpha:
    print(f"Reject the Null hypothesis at : {pvalue}, the mean values
else:
    print(f"Failed to Reject H0 at : {pvalue}, the Mean values are Bo
print(" ** This Includes the Osrm time and Segment osrm time are the
```

```
The statistic value is : -6.273924701299134 and the P_value is : 3.54783
9442615796e-10
Reject the Null hypothesis at : 3.547839442615796e-10, the mean values f
or the Both of the columns are Not same
 ** This Includes the Osrm time and Segment osrm time are the on the dif
ferent Distributed values **
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```