In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.stats as spy
```

In [2]:
```
import warnings
warnings.simplefilter('ignore')
```

In [3]:
```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/a
df.head()
```

Out[3]:

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

5 rows × 24 columns

In [4]:
```
df.shape
```

Out[4]:
```
(144867, 24)
```

In [5]:
```
df.columns
```

Out[5]:
```
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_t
ime',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_facto
r'],
      dtype='object')
```

In [6]:
```
df.dtypes
```

Out[6]:
```
data                                object
trip_creation_time                  object
route_schedule_uuid                 object
route_type                          object
trip_uuid                           object
source_center                       object
source_name                         object
destination_center                  object
destination_name                    object
od_start_time                       object
od_end_time                         object
start_scan_to_end_scan             float64
is_cutoff                             bool
cutoff_factor                        int64
cutoff_timestamp                    object
actual_distance_to_destination     float64
actual_time                        float64
osrm_time                          float64
osrm_distance                      float64
factor                             float64
segment_actual_time                float64
segment_osrm_time                  float64
segment_osrm_distance              float64
segment_factor                     float64
dtype: object
```

In [7]:
```
df.info()
```

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

In [8]:
```
for i in df.columns:
    print(f"Unique entries for column {i:<30} = {df[i].nunique()}")
```

```
Unique entries for column data                           = 2
Unique entries for column trip_creation_time             = 14817
Unique entries for column route_schedule_uuid            = 1504
Unique entries for column route_type                     = 2
Unique entries for column trip_uuid                      = 14817
Unique entries for column source_center                  = 1508
Unique entries for column source_name                    = 1498
Unique entries for column destination_center             = 1481
Unique entries for column destination_name               = 1468
Unique entries for column od_start_time                  = 26369
Unique entries for column od_end_time                    = 26369
Unique entries for column start_scan_to_end_scan         = 1915
Unique entries for column is_cutoff                      = 2
Unique entries for column cutoff_factor                  = 501
Unique entries for column cutoff_timestamp               = 93180
Unique entries for column actual_distance_to_destination = 144515
Unique entries for column actual_time                    = 3182
Unique entries for column osrm_time                      = 1531
Unique entries for column osrm_distance                  = 138046
Unique entries for column factor                         = 45641
Unique entries for column segment_actual_time            = 747
Unique entries for column segment_osrm_time              = 214
Unique entries for column segment_osrm_distance          = 113799
Unique entries for column segment_factor                 = 5675
```

In [9]:
```
# For all those colums where numbers of unique entries is 2, converting
df['data'] = df['data'].astype('category')
df['route_type'] = df['route_type'].astype('category')
```

In [10]:
```
floating_columns = ['actual_distance_to_destination', 'actual_time', 'o
                    'segment_actual_time', 'segment_osrm_time', 'segmen
for i in floating_columns:
    print(df[i].max())
```

```
1927.4477046975032
4532.0
1686.0
2326.1991000000003
3051.0
1611.0
2191.4037000000003
```

In [11]:
```
# We can update the data type to float32 since the maximum value entry
```

In [12]:
```
for i in floating_columns:
    df[i] = df[i].astype('float32')
```

In [13]:
```
# Updateinf the datatype of datetime columns
datetime_columns = ['trip_creation_time', 'od_start_time', 'od_end_time
for i in datetime_columns:
    df[i] = pd.to_datetime(df[i])
```

In [14]:
```
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  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 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  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 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  float32
 16  actual_time                     144867 non-null  float32
 17  osrm_time                       144867 non-null  float32
 18  osrm_distance                   144867 non-null  float32
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float32
 21  segment_osrm_time               144867 non-null  float32
 22  segment_osrm_distance           144867 non-null  float32
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), category(2), datetime64[ns](3), float32(7), float64
(3), int64(1), object(7)
memory usage: 19.8+ MB
```

Earlier the dataset was using 26.5+ MB of memory but now it has been reduced to 15.2+ MB. Around 40.63% reduction of the memory usage.

In [15]:
```
# Time period for which the date is given:
df['trip_creation_time'].min(), df['od_end_time'].max()
```

Out[15]:
```
(Timestamp('2018-09-12 00:00:16.535741'),
 Timestamp('2018-10-08 03:00:24.353479'))
```

1. Basic data cleaning and Exploration:

||Handling Missing Values||

In [16]:
```
# Is therer any null values present in the data:
np.any(df.isnull())
```

Out[16]:
```
True
```

In [17]:
```
# Number of null values present in each column:
df.isnull().sum()
```

Out[17]:
```
data                                0
trip_creation_time                  0
route_schedule_uuid                 0
route_type                          0
trip_uuid                           0
source_center                       0
source_name                       293
destination_center                  0
destination_name                  261
od_start_time                       0
od_end_time                         0
start_scan_to_end_scan              0
is_cutoff                           0
cutoff_factor                       0
cutoff_timestamp                    0
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
factor                              0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
segment_factor                      0
dtype: int64
```

In [18]:
```
missing_source_name = df.loc[df['source_name'].isnull(), 'source_center
missing_source_name
```

Out[18]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [19]:
```
for i in missing_source_name:
    unique_source_name = df.loc[df['source_center'] == i, 'source_name'
    if pd.isna(unique_source_name):
        print("Source Center :", i, "-" * 10, "Source Name :", 'Not Fou
    else :
        print("Source Center :", i, "-" * 10, "Source Name :", unique_s
```

```
Source Center : IND342902A1B ---------- Source Name : Not Found
Source Center : IND577116AAA ---------- Source Name : Not Found
Source Center : IND282002AAD ---------- Source Name : Not Found
Source Center : IND465333A1B ---------- Source Name : Not Found
Source Center : IND841301AAC ---------- Source Name : Not Found
Source Center : IND509103AAC ---------- Source Name : Not Found
Source Center : IND126116AAA ---------- Source Name : Not Found
Source Center : IND331022A1B ---------- Source Name : Not Found
Source Center : IND505326AAB ---------- Source Name : Not Found
Source Center : IND852118A1B ---------- Source Name : Not Found
```

In [20]:
```
for i in missing_source_name:
    unique_destination_name = df.loc[df['destination_center'] == i, 'de
    if (pd.isna(unique_source_name)) or (unique_source_name.size == 0):
        print("Destination Center :", i, "-" * 10, "Destination Name :"
    else :
        print("Destination Center :", i, "-" * 10, "Destination Name :"
```

```
Destination Center : IND342902A1B ---------- Destination Name : Not Fo
und
Destination Center : IND577116AAA ---------- Destination Name : Not Fo
und
Destination Center : IND282002AAD ---------- Destination Name : Not Fo
und
Destination Center : IND465333A1B ---------- Destination Name : Not Fo
und
Destination Center : IND841301AAC ---------- Destination Name : Not Fo
und
Destination Center : IND509103AAC ---------- Destination Name : Not Fo
und
Destination Center : IND126116AAA ---------- Destination Name : Not Fo
und
Destination Center : IND331022A1B ---------- Destination Name : Not Fo
und
Destination Center : IND505326AAB ---------- Destination Name : Not Fo
und
Destination Center : IND852118A1B ---------- Destination Name : Not Fo
und
```

In [21]:
```
missing_destination_name = df.loc[df['destination_name'].isnull(), 'des
missing_destination_name
```

Out[21]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [22]:
```
# The IDs for which the source name is missing, are all those IDs for d
np.all(df.loc[df['source_name'].isnull(), 'source_center'].isin(missing
```

Out[22]: False

- Treating missing destination names and source names

In [23]:
```
count = 1
for i in missing_destination_name:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.loc[
    count += 1
```

In [24]:
```
d = {}
for i in missing_source_name:
    d[i] = df.loc[df['destination_center'] == i, 'destination_name'].un
for idx, val in d.items():
    if len(val) == 0:
        d[idx] = [f'location_{count}']
        count += 1
d2 = {}
for idx, val in d.items():
    d2[idx] = val[0]
for i, v in d2.items():
    print(i, v)
```

```
IND342902A1B location_1
IND577116AAA location_2
IND282002AAD location_3
IND465333A1B location_4
IND841301AAC location_5
IND509103AAC location_9
IND126116AAA location_8
IND331022A1B location_14
IND505326AAB location_6
IND852118A1B location_7
```

In [25]:
```
for i in missing_source_name:
    df.loc[df['source_center'] == i, 'source_name'] = df.loc[df['source
```

In [26]:
```
df.isna().sum()
```

Out[26]:
```
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
dtype: int64
```

In [27]:
```
# Basic Description of data:
df.describe()
```

Out[27]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_sca
count	144867	144867	144867	144867.0000(
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.2629{
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.0000(
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.0000(
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.0000(
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.0000(
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.0000(
std	NaN	NaN	NaN	1037.0127(

In [28]:
```
df.describe(include = 'object')
```

Out[28]:

	route_schedule_uuid	trip_uuid	source_center	source_nam
count	144867	144867	144867	14486
unique	1504	14817	1508	150
top	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_H (Haryana
freq	1812	101	23347	2334

- **Merging of rows and aggregation of fields**
- Since delivery details of one package are divided into several rows (think of it as connecting flights to reach a particular destination). Now think about how we should treat their fields if we combine these rows? What aggregation would make sense if we merge. What would happen to the numeric fields if we merge the rows.

In [29]:
```
grouping_1 = ['trip_uuid', 'source_center', 'destination_center']
df1 = df.groupby(by = grouping_1, as_index = False).agg({'data' : 'firs
                                                         'route_type' :
                                                        'trip_creation_t
                                                        'source_name' :
                                                        'destination_nam
                                                        'od_start_time'
                                                        'od_end_time' :
                                                        'start_scan_to_e
                                                        'actual_distance
                                                        'actual_time' :
                                                        'osrm_time' : 'l
                                                        'osrm_distance'
                                                        'segment_actual_
                                                        'segment_osrm_ti
                                                        'segment_osrm_di

df1
```

Out[29]:

	trip_uuid	source_center	destination_center	data	route_type	trip_
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	FTL	0(
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	FTL	0(
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	Carting	0(
3	trip-153671042288605164	IND572101AAA	IND561203AAB	training	Carting	0(
4	trip-153671043369099517	IND000000ACB	IND160002AAC	training	FTL	0(
...	...	...	...	...	...	
26363	trip-153861115439069069	IND628204AAA	IND627657AAA	test	Carting	2:
26364	trip-153861115439069069	IND628613AAA	IND627005AAA	test	Carting	2:
26365	trip-153861115439069069	IND628801AAA	IND628204AAA	test	Carting	2:
26366	trip-153861118270144424	IND583119AAA	IND583101AAA	test	FTL	2:
26367	trip-153861118270144424	IND583201AAA	IND583119AAA	test	FTL	2:

26368 rows × 18 columns

Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

In [30]:
```
df1['od_total_time'] = df1['od_end_time'] - df1['od_start_time']
df1.drop(columns = ['od_end_time', 'od_start_time'], inplace = True)
df1['od_total_time'] = df1['od_total_time'].apply(lambda x : round(x.to
df1['od_total_time'].head()
```

Out[30]:
```
0    1260.60
1     999.51
2      58.83
3     122.78
4     834.64
Name: od_total_time, dtype: float64
```

In [31]:
```
df2 = df1.groupby(by = 'trip_uuid', as_index = False).agg({'source_cent
                                                           'destination
                                                           'data' : 'fi
                                                           'route_type'
                                                           'trip_creati
                                                           'source_name
                                                           'destination
                                                           'od_total_ti
                                                           'start_scan_
                                                           'actual_dist
                                                           'actual_time
                                                           'osrm_time'
                                                           'osrm_distan
                                                           'segment_act
                                                           'segment_osr
                                                           'segment_osr
df2
```

Out[31]:

	trip_uuid	source_center	destination_center	data	route_type	trip_
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	0(
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	0(
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	0(
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	0(
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	0(
...	...	...	...	...	...	
14812	trip-153861095625827784	IND160002AAC	IND160002AAC	test	Carting	2:
14813	trip-153861104386292051	IND121004AAB	IND121004AAA	test	Carting	2:
14814	trip-153861106442901555	IND208006AAA	IND208006AAA	test	Carting	2:
14815	trip-153861115439069069	IND627005AAA	IND628204AAA	test	Carting	2:
14816	trip-153861118270144424	IND583119AAA	IND583119AAA	test	FTL	2:

14817 rows × 17 columns

2. Build some features to prepare the data for the actual analysis. Extract features from the below fields

In [32]:
```
# Source Name: Split and extract features out of destination. City-plac
def location_name_to_state(x):
    l = x.split('(')
    if len(l) == 1:
        return l[0]
    else:
        return l[1].replace(')', "")
```

In [33]:
```
def location_name_to_city(x):
    if 'location' in x:
        return 'unknown_city'
    else:
        l = x.split()[0].split('_')
        if 'CCU' in x:
            return 'Kolkata'
        elif 'MAA' in x.upper():
            return 'Chennai'
        elif ('HBR' in x.upper()) or ('BLR' in x.upper()):
            return 'Bengaluru'
        elif 'FBD' in x.upper():
            return 'Faridabad'
        elif 'BOM' in x.upper():
            return 'Mumbai'
        elif 'DEL' in x.upper():
            return 'Delhi'
        elif 'OK' in x.upper():
            return 'Delhi'
        elif 'GZB' in x.upper():
            return 'Ghaziabad'
        elif 'GGN' in x.upper():
            return 'Gurgaon'
        elif 'AMD' in x.upper():
            return 'Ahmedabad'
        elif 'CJB' in x.upper():
            return 'Coimbatore'
        elif 'HYD' in x.upper():
            return 'Hyderabad'
        return l[0]
```

In [34]:
```
def location_name_to_place(x):
    if 'location' in x:
        return x
    elif 'HBR' in x:
        return 'HBR Layout PC'
    else:
        l = x.split()[0].split('_', 1)
        if len(l) == 1:
            return 'unknown_place'
        else:
            return l[1]
```

In [35]:
```
df2['source_state'] = df2['source_name'].apply(location_name_to_state)
df2['source_state'].unique()
```

Out[35]:
```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
       'Assam', 'Madhya Pradesh', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Goa', 'Jharkhand', 'Pondicherry',
       'Orissa', 'Uttarakhand', 'Himachal Pradesh', 'Kerala',
       'Arunachal Pradesh', 'Bihar', 'Chhattisgarh',
       'Dadra and Nagar Haveli', 'Jammu & Kashmir', 'Mizoram', 'Nagala
nd',
       'location_9', 'location_3', 'location_2', 'location_14',
       'location_7'], dtype=object)
```

In [36]:
```
df2['source_city'] = df2['source_name'].apply(location_name_to_city)
print('No of source cities :', df2['source_city'].nunique())
df2['source_city'].unique()[:100]
```

```
No of source cities : 690
```

Out[36]:
```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Bellary', 'Chenna
i',
       'Bengaluru', 'Surat', 'Delhi', 'Pune', 'Faridabad', 'Shirala',
       'Hyderabad', 'Thirumalagiri', 'Gulbarga', 'Jaipur', 'Allahaba
d',
       'Guwahati', 'Narsinghpur', 'Shrirampur', 'Madakasira', 'Sonar
i',
       'Dindigul', 'Jalandhar', 'Chandigarh', 'Deoli', 'Pandharpur',
       'Kolkata', 'Bhandara', 'Kurnool', 'Bhiwandi', 'Bhatinda',
       'RoopNagar', 'Bantwal', 'Lalru', 'Kadi', 'Shahdol', 'Gangakhe
r',
       'Durgapur', 'Vapi', 'Jamjodhpur', 'Jetpur', 'Mehsana', 'Jabalpu
r',
       'Junagadh', 'Gundlupet', 'Mysore', 'Goa', 'Bhopal', 'Sonipat',
       'Himmatnagar', 'Jamshedpur', 'Pondicherry', 'Anand', 'Udgir',
       'Nadiad', 'Villupuram', 'Purulia', 'Bhubaneshwar', 'Bamangola',
       'Tiruppattur', 'Kotdwara', 'Medak', 'Bangalore', 'Dhrangadhra',
       'Hospet', 'Ghumarwin', 'Agra', 'Sitapur', 'Canacona', 'Bilimor
a',
       'SultnBthry', 'Lucknow', 'Vellore', 'Bhuj', 'Dinhata',
       'Margherita', 'Boisar', 'Vizag', 'Tezpur', 'Koduru', 'Tirupat
i',
       'Pen', 'Ahmedabad', 'Faizabad', 'Gandhinagar', 'Anantapur',
       'Betul', 'Panskura', 'Rasipurm', 'Sankari', 'Jorhat', 'PNQ',
       'Srikakulam', 'Dehradun', 'Jassur', 'Sawantwadi', 'Shajapur',
       'Ludhiana', 'GreaterThane'], dtype=object)
```

In [37]:
```
df2['source_place'] = df2['source_name'].apply(location_name_to_place)
df2['source_place'].unique()[:100]
```

Out[37]:
```
array(['Central_H_6', 'ChikaDPP_D', 'Bilaspur_HB', 'unknown_place', 'D
c',
       'Poonamallee', 'Chrompet_DPC', 'HBR Layout PC', 'Central_D_12',
       'Lajpat_IP', 'North_D_3', 'Balabhgarh_DPC', 'Central_DPP_3',
       'Shamshbd_H', 'Xroad_D', 'Nehrugnj_I', 'Central_I_7',
       'Central_H_1', 'Nangli_IP', 'North', 'KndliDPP_D', 'Central_D_
9',
       'DavkharRd_D', 'Bandel_D', 'RTCStand_D', 'Central_DPP_1',
       'KGAirprt_HB', 'North_D_2', 'Central_D_1', 'DC', 'Mthurard_L',
       'Mullanpr_DC', 'Central_DPP_2', 'RajCmplx_D', 'Beliaghata_DPC',
       'RjnaiDPP_D', 'AbbasNgr_I', 'Mankoli_HB', 'DPC', 'Airport_H',
       'Hub', 'Gateway_HB', 'Tathawde_H', 'ChotiHvl_DC', 'Trmltmpl_D',
       'OnkarDPP_D', 'Mehmdpur_H', 'KaranNGR_D', 'Sohagpur_D',
       'Chrompet_L', 'Busstand_D', 'Central_I_1', 'IndEstat_I', 'Court
_D',
       'Panchot_IP', 'Adhartal_IP', 'DumDum_DPC', 'Bomsndra_HB',
       'Swamylyt_D', 'Yadvgiri_IP', 'Old', 'Kundli_H', 'Central_I_3',
       'Vasanthm_I', 'Poonamallee_HB', 'VUNagar_DC', 'NlgaonRd_D',
       'Bnnrghta_L', 'Thirumtr_IP', 'GariDPP_D', 'Jogshwri_I',
       'KoilStrt_D', 'CotnGren_M', 'Nzbadrd_D', 'Dwaraka_D', 'Nelmngla
_H',
       'NvygRDPP_D', 'Gndhichk_D', 'Central_D_3', 'Chowk_D', 'CharRsta
_D',
       'Kollgpra_D', 'Peenya_IP', 'GndhiNgr_IP', 'Sanpada_I',
       'WrdN4DPP_D', 'Sakinaka_RP', 'CivilHPL_D', 'OstwlEmp_D',
       'Gajuwaka', 'Mhbhirab_D', 'MGRoad_D', 'Balajicly_I', 'BljiMrkt_
D',
       'Dankuni_HB', 'Trnsport_H', 'Rakhial', 'Memnagar', 'East_I_21',
       'Mithakal_D'], dtype=object)
```

Destination Name: Split and Extract features out of destination. City-Place-Code(State)

In [38]:
```
df2['destination_state'] = df2['destination_name'].apply(location_name_
df2['destination_state'].head(10)
```

Out[38]:
```
0    Uttar Pradesh
1        Karnataka
2          Haryana
3      Maharashtra
4        Karnataka
5       Tamil Nadu
6       Tamil Nadu
7        Karnataka
8          Gujarat
9            Delhi
Name: destination_state, dtype: object
```

In [39]:
```
df2['destination_city'] = df2['destination_name'].apply(location_name_t
df2['destination_city'].head(5)
```

Out[39]:
```
0        Kanpur
1    Doddablpur
2       Gurgaon
3        Mumbai
4        Sandur
Name: destination_city, dtype: object
```

In [40]:
```
df2['destination_place'] = df2['destination_name'].apply(location_name_
df2['destination_place'].head(5)
```

Out[40]:
```
0    Central_H_6
1     ChikaDPP_D
2    Bilaspur_HB
3      MiraRd_IP
4     WrdN1DPP_D
Name: destination_place, dtype: object
```

Trip_Creation_Time: Extract feature like month, year and day

In [41]:
```
df2['trip_creation_date'] = pd.to_datetime(df2['trip_creation_time'].dt
df2['trip_creation_date'].head()
```

Out[41]:
```
0   2018-09-12
1   2018-09-12
2   2018-09-12
3   2018-09-12
4   2018-09-12
Name: trip_creation_date, dtype: datetime64[ns]
```

In [42]:
```
df2['trip_creation_day'] = df2['trip_creation_time'].dt.day
df2['trip_creation_day'] = df2['trip_creation_day'].astype('int8')
df2['trip_creation_day'].head()
```

Out[42]:
```
0    12
1    12
2    12
3    12
4    12
Name: trip_creation_day, dtype: int8
```

In [43]:
```
df2['trip_creation_month'] = df2['trip_creation_time'].dt.month
df2['trip_creation_month'] = df2['trip_creation_month'].astype('int8')
df2['trip_creation_month'].head()
```

Out[43]:
```
0    9
1    9
2    9
3    9
4    9
Name: trip_creation_month, dtype: int8
```

In [44]:
```
df2['trip_creation_year'] = df2['trip_creation_time'].dt.year
df2['trip_creation_year'] = df2['trip_creation_year'].astype('int16')
df2['trip_creation_year'].head()
```

Out[44]:
```
0    2018
1    2018
2    2018
3    2018
4    2018
Name: trip_creation_year, dtype: int16
```

In [45]:
```
df2['trip_creation_week'] = df2['trip_creation_time'].dt.isocalendar().
df2['trip_creation_week'] = df2['trip_creation_week'].astype('int8')
df2['trip_creation_week'].head()
```

Out[45]:
```
0    37
1    37
2    37
3    37
4    37
Name: trip_creation_week, dtype: int8
```

In [46]:
```
df2['trip_creation_hour'] = df2['trip_creation_time'].dt.hour
df2['trip_creation_hour'] = df2['trip_creation_hour'].astype('int8')
df2['trip_creation_hour'].head()
```

Out[46]:
```
0    0
1    0
2    0
3    0
4    0
Name: trip_creation_hour, dtype: int8
```

In [47]:
```
# Finding the structure of data after data cleaning.
df2.shape
```

Out[47]:
```
(14817, 29)
```

In [48]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14817 non-null  object
 1   source_center                   14817 non-null  object
 2   destination_center              14817 non-null  object
 3   data                            14817 non-null  category
 4   route_type                      14817 non-null  category
 5   trip_creation_time              14817 non-null  datetime64[ns]
 6   source_name                     14817 non-null  object
 7   destination_name                14817 non-null  object
 8   od_total_time                   14817 non-null  float64
 9   start_scan_to_end_scan          14817 non-null  float64
 10  actual_distance_to_destination  14817 non-null  float32
 11  actual_time                     14817 non-null  float32
 12  osrm_time                       14817 non-null  float32
 13  osrm_distance                   14817 non-null  float32
 14  segment_actual_time             14817 non-null  float32
 15  segment_osrm_time               14817 non-null  float32
 16  segment_osrm_distance           14817 non-null  float32
 17  source_state                    14817 non-null  object
 18  source_city                     14817 non-null  object
 19  source_place                    14817 non-null  object
 20  destination_state               14817 non-null  object
 21  destination_city                14817 non-null  object
 22  destination_place               14817 non-null  object
 23  trip_creation_date              14817 non-null  datetime64[ns]
 24  trip_creation_day               14817 non-null  int8
 25  trip_creation_month             14817 non-null  int8
 26  trip_creation_year              14817 non-null  int16
 27  trip_creation_week              14817 non-null  int8
 28  trip_creation_hour              14817 non-null  int8
dtypes: category(2), datetime64[ns](2), float32(7), float64(2), int16
(1), int8(4), object(11)
memory usage: 2.2+ MB
```

In [49]:
```
df2.describe().T
```

Out[49]:

	count	mean	min	
trip_creation_time	14817	2018-09-22 12:44:19.555167744	2018-09-12 00:00:16.535741	201 02:51:25.129
od_total_time	14817.0	531.69763	23.46	
start_scan_to_end_scan	14817.0	530.810016	23.0	
actual_distance_to_destination	14817.0	164.477829	9.002461	22.
actual_time	14817.0	357.143768	9.0	
osrm_time	14817.0	161.384018	6.0	
osrm_distance	14817.0	204.344711	9.0729	30.
segment_actual_time	14817.0	353.892273	9.0	
segment_osrm_time	14817.0	180.949783	6.0	
segment_osrm_distance	14817.0	223.201157	9.0729	32.
trip_creation_date	14817	2018-09-21 23:46:58.627252736	2018-09-12 00:00:00	201 0
trip_creation_day	14817.0	18.37079	1.0	
trip_creation_month	14817.0	9.120672	9.0	
trip_creation_year	14817.0	2018.0	2018.0	
trip_creation_week	14817.0	38.295944	37.0	
trip_creation_hour	14817.0	12.449821	0.0	

In [50]:
```
df2.describe(include= object).T
```

Out[50]:

	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
source_center	14817	938	IND000000ACB	1063
destination_center	14817	1042	IND000000ACB	821
source_name	14817	938	Gurgaon_Bilaspur_HB (Haryana)	1063
destination_name	14817	1042	Gurgaon_Bilaspur_HB (Haryana)	821
source_state	14817	34	Maharashtra	2714
source_city	14817	690	Mumbai	1442
source_place	14817	761	Bilaspur_HB	1063
destination_state	14817	39	Maharashtra	2561
destination_city	14817	806	Mumbai	1548
destination_place	14817	850	Bilaspur_HB	821

In [51]:
```
# Number of trips on the hourly basis:
df2['trip_creation_hour'].unique()
```

Out[51]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19, 20, 21, 22, 23], dtype=int8)
```

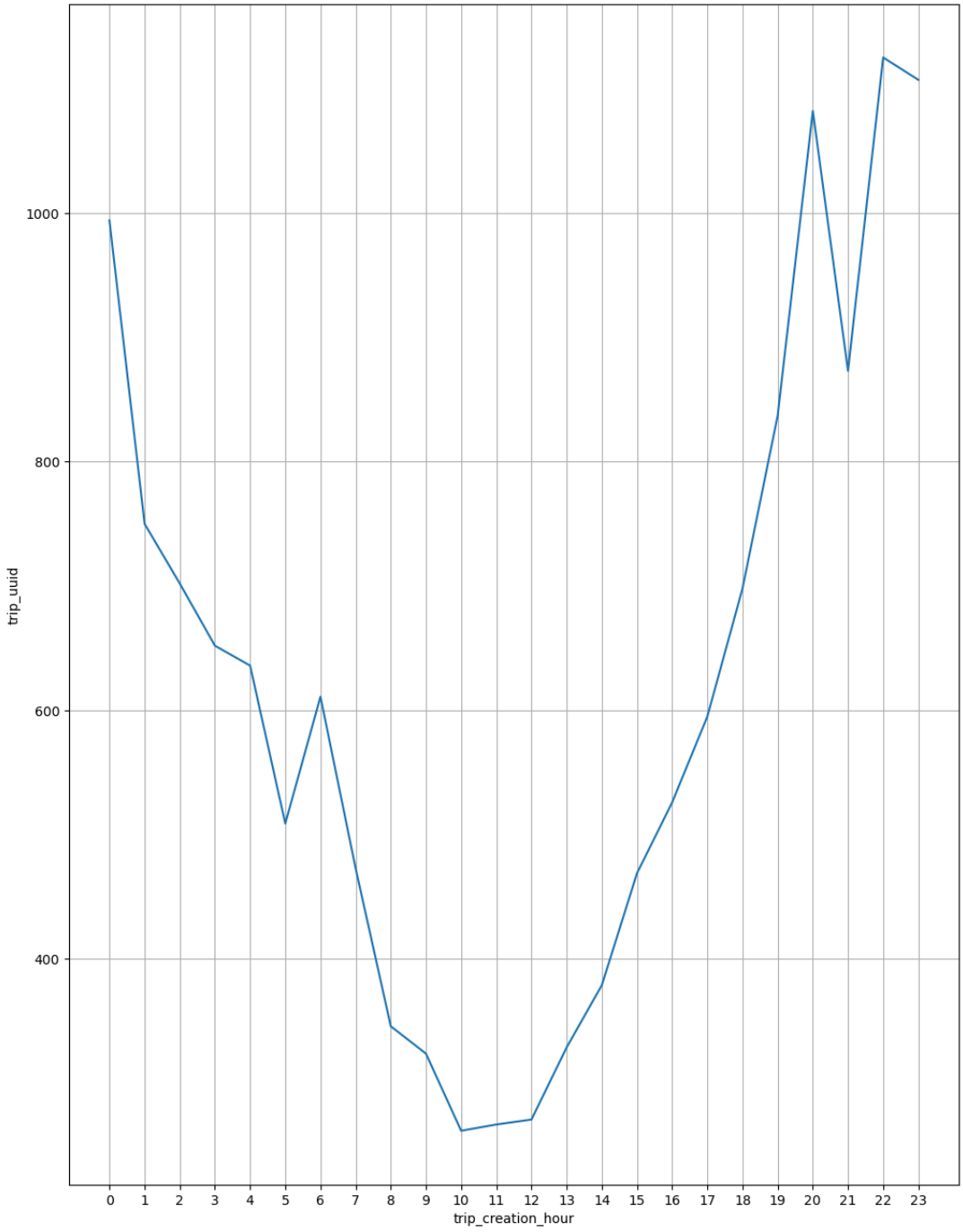
In [52]:
```
df_hour = df2.groupby(by = 'trip_creation_hour')['trip_uuid'].count().t
df_hour.head()
```

Out[52]:

	trip_creation_hour	trip_uuid
0	0	994
1	1	750
2	2	702
3	3	652
4	4	636

In [53]:
```
plt.figure(figsize = (12,16))
sns.lineplot(data = df_hour,
             x = df_hour['trip_creation_hour'],
             y = df_hour['trip_uuid'],
             markers = '*')
plt.xticks(np.arange(0,24))
plt.grid('both')
plt.plot()
```

Out[53]: []



- It can be inferred from the above plot that the number of trips start increasing after the noon, becomes maximum at 10:00 PM and then start decreasing.

In [54]:
```
# Number of trips are created for different days of the month
df2['trip_creation_day'].unique()
```

Out[54]:
```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28,
       29, 30,  1,  2,  3], dtype=int8)
```

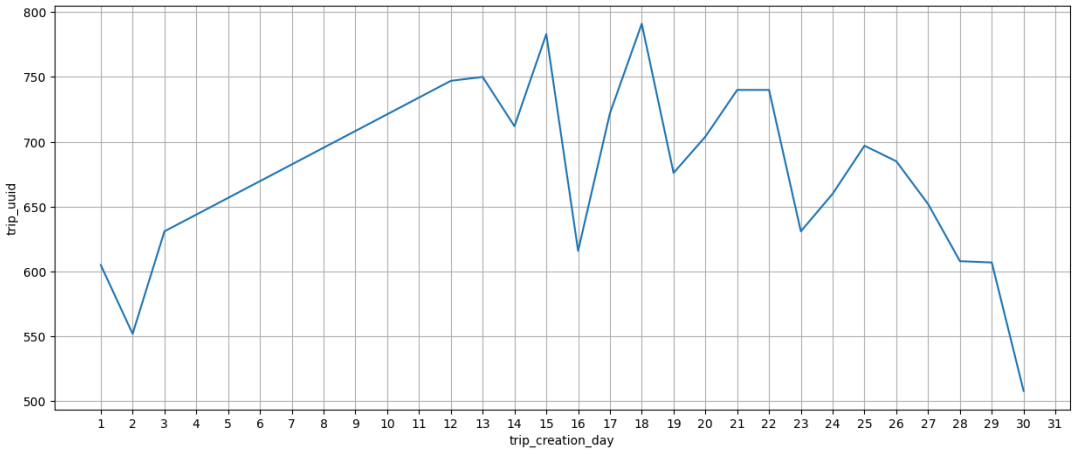
In [55]:
```
df_day = df2.groupby(by = 'trip_creation_day')['trip_uuid'].count().to_
df_day.head()
```

Out[55]:

	trip_creation_day	trip_uuid
0	1	605
1	2	552
2	3	631
3	12	747
4	13	750

In [56]:
```
plt.figure(figsize = (15,6))
sns.lineplot(data = df_day,
             x = df_day['trip_creation_day'],
             y = df_day['trip_uuid'],
             markers = 'o')
plt.xticks(np.arange(1,32))
plt.grid('both')
plt.plot()
```

Out[56]:
```
[]
```



- It can be inferred from the above plot that most of the trips are created in the middile of the month.
- It means customers usually make more orders in the mid of the month.

In [57]:
```
# Number of trips are created for different weeeks
df2['trip_creation_week'].unique()
```

Out[57]:
```
array([37, 38, 39, 40], dtype=int8)
```

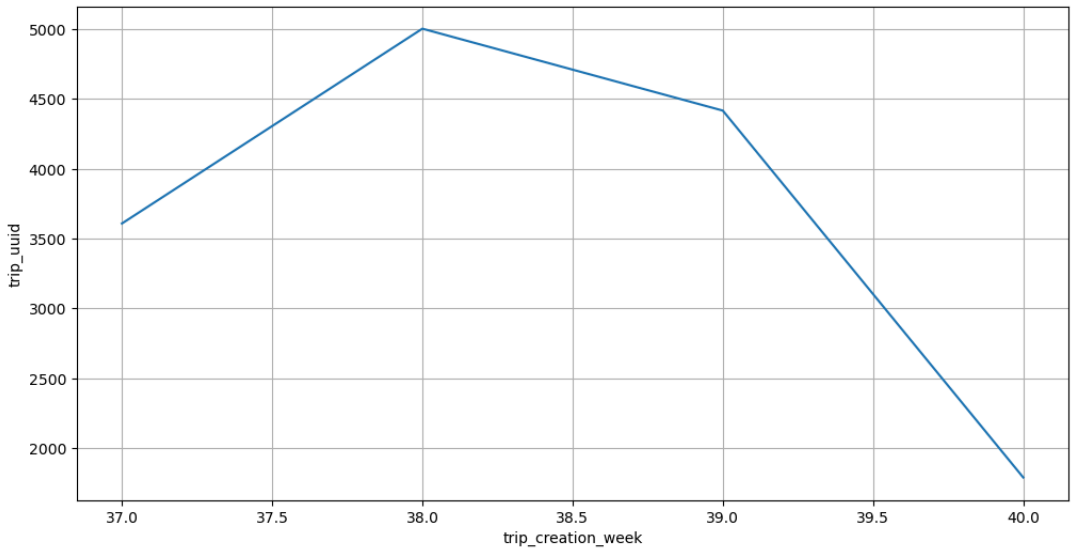
In [58]:
```
df_week = df2.groupby(by = 'trip_creation_week')['trip_uuid'].count().t
df_week.head()
```

Out[58]:

	trip_creation_week	trip_uuid
0	37	3608
1	38	5004
2	39	4417
3	40	1788

In [59]:
```
plt.figure(figsize = (12,6))
sns.lineplot(data = df_week,
        x = df_week['trip_creation_week'],
        y = df_week['trip_uuid'],
        markers = '*')
plt.grid('both')
plt.plot()
```

Out[59]: []



- It can be inferred from the above plot that most of the trips are created in the 38th week

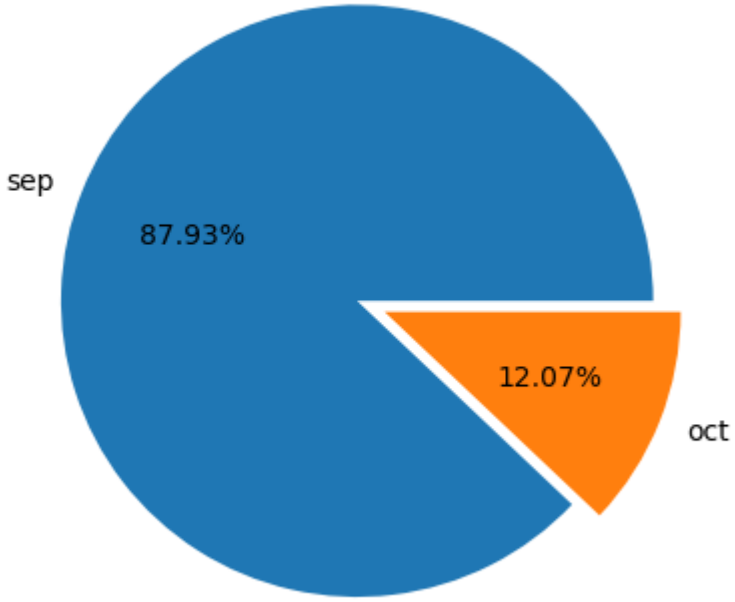
In [60]:
```
# Number of trips are created in the given two months:
df_month = df2.groupby(by = 'trip_creation_month')['trip_uuid'].count()
df_month['percent'] = np.round(df_month['trip_uuid'] * 100/ df_month['t
df_month.head()
```

Out[60]:

	trip_creation_month	trip_uuid	percent
0	9	13029	87.93
1	10	1788	12.07

In [61]:
```
plt.pie(x = df_month['trip_uuid'],
        labels = ['sep', 'oct'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

Out[61]: []



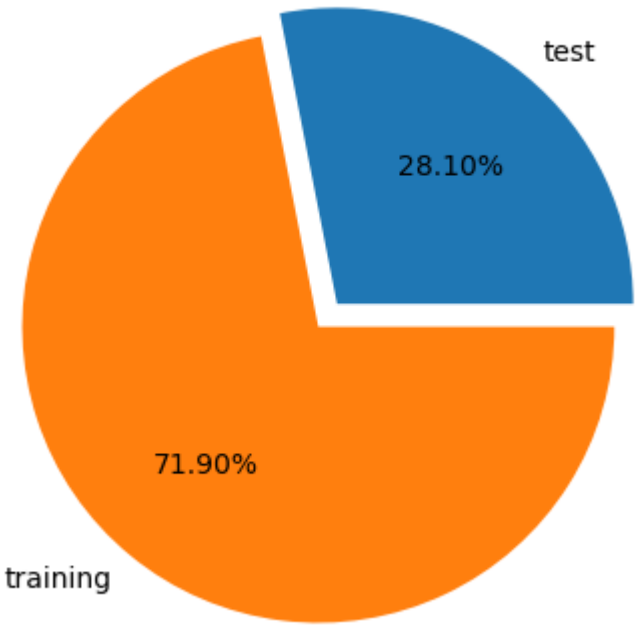
In [62]:
```
# Distribution  of trip data for the orders
df_data = df2.groupby(by = 'data')['trip_uuid'].count().to_frame().rese
df_data['percent'] = np.round(df_data['trip_uuid'] * 100/ df_data['trip
df_data.head()
```

Out[62]:

	data	trip_uuid	percent
0	test	4163	28.1
1	training	10654	71.9

In [63]:
```
plt.pie(x = df_data['trip_uuid'],
        labels = df_data['data'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

Out[63]: []



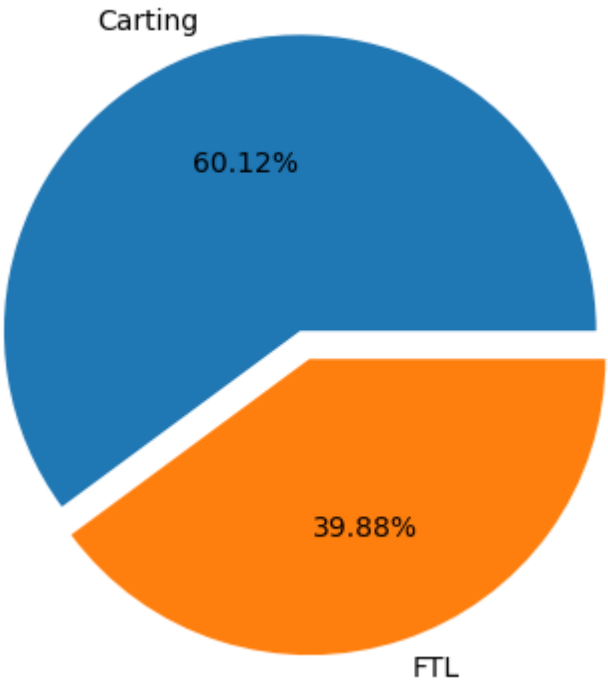
In [64]:
```
# Distribution of route type of the orders
df_route = df2.groupby(by = 'route_type')['trip_uuid'].count().to_frame
df_route['percent'] = np.round(df_route['trip_uuid'] * 100/ df_route['t
df_route.head()
```

Out[64]:

	route_type	trip_uuid	percent
0	Carting	8908	60.12
1	FTL	5909	39.88

In [65]:
```
plt.pie(x = df_route['trip_uuid'],
        labels = ['Carting', 'FTL'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

Out[65]: []



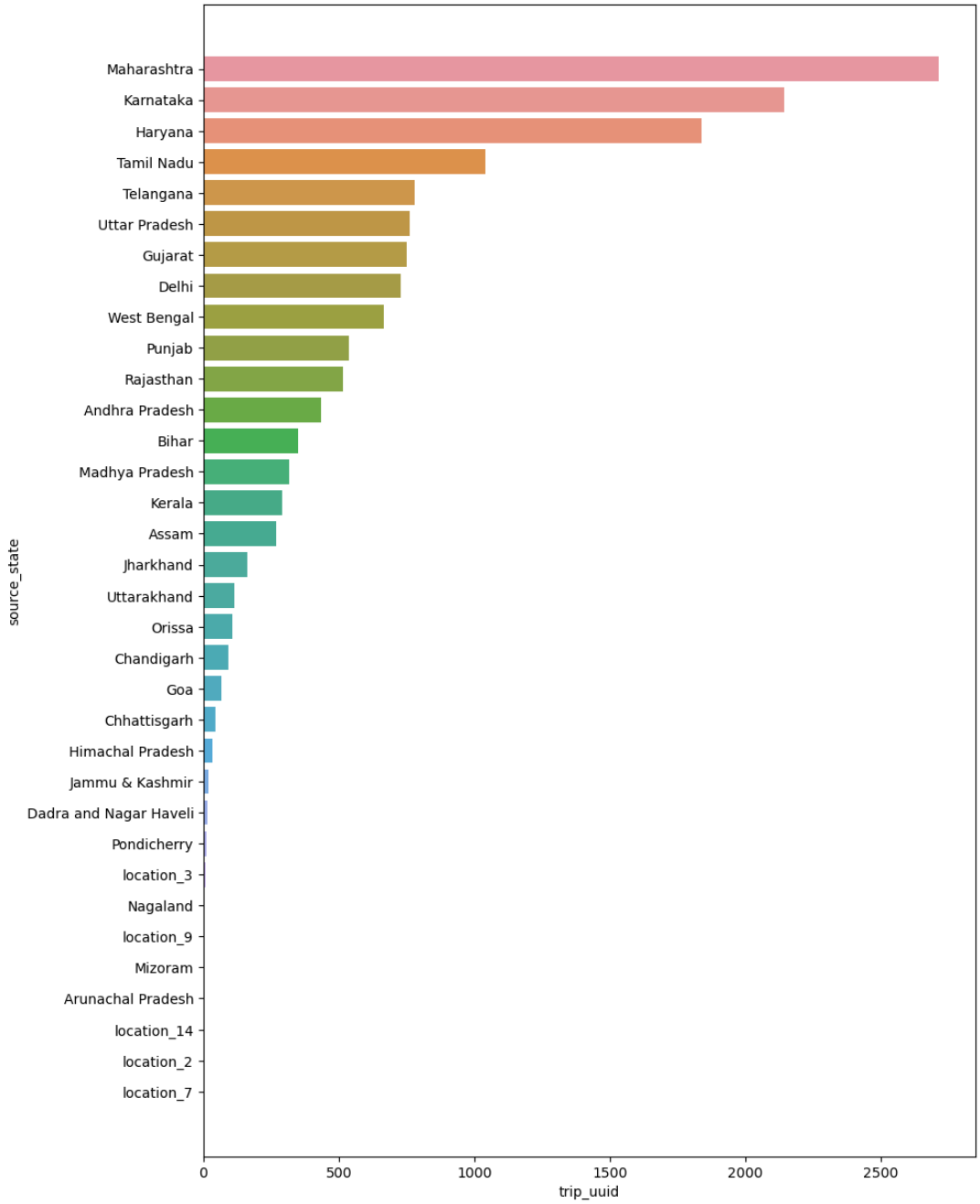
In [66]:
```
# Distribution of number of trips created for different states
df_source_state = df2.groupby(by = 'source_state')['trip_uuid'].count()
df_source_state['percent'] = np.round(df_source_state['trip_uuid'] * 10
df_source_state = df_source_state.sort_values(by = 'trip_uuid', ascendi
df_source_state.head()
```

Out[66]:

	source_state	trip_uuid	percent
17	Maharashtra	2714	18.32
14	Karnataka	2143	14.46
10	Haryana	1838	12.40
24	Tamil Nadu	1039	7.01
25	Telangana	781	5.27

In [67]:
```
plt.figure(figsize = (10, 15))
sns.barplot(data = df_source_state,
            x = df_source_state['trip_uuid'],
            y = df_source_state['source_state'])
plt.plot()
```

Out[67]: []



- We can see from the above plot that maximum trips are originated from Maharastra state followed by Karnataka and Haryana. That means the saller base is strong in these state.

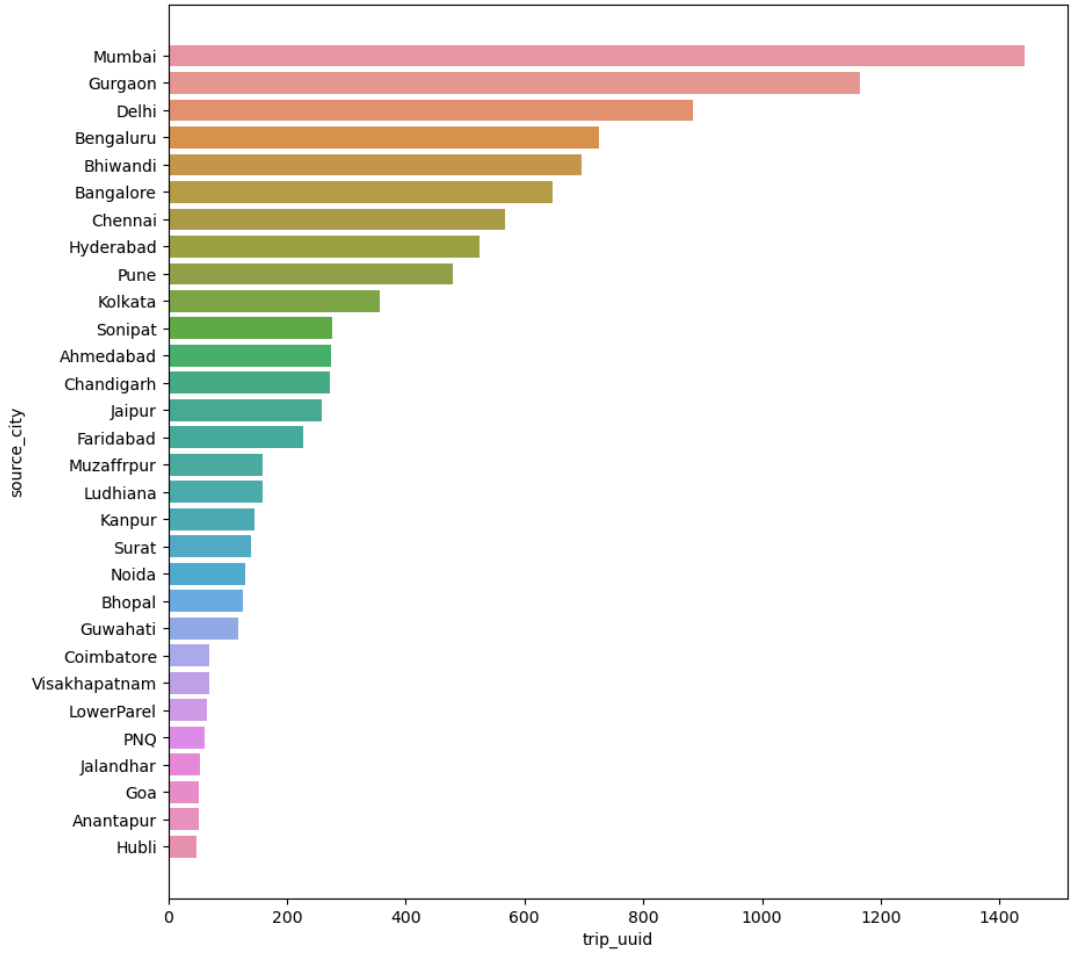
In [68]:
```
# We can find top 30 cities based on the number of trips created from d
df_source_city = df2.groupby(by = 'source_city')['trip_uuid'].count().t
df_source_city['percent'] = np.round(df_source_city['trip_uuid'] * 100/
df_source_city = df_source_city.sort_values(by = 'trip_uuid', ascending
df_source_city
```

Out[68]:

	source_city	trip_uuid	percent
439	Mumbai	1442	9.73
237	Gurgaon	1165	7.86
169	Delhi	883	5.96
79	Bengaluru	726	4.90
100	Bhiwandi	697	4.70
58	Bangalore	648	4.37
136	Chennai	568	3.83
264	Hyderabad	524	3.54
516	Pune	480	3.24
357	Kolkata	356	2.40
610	Sonipat	276	1.86
2	Ahmedabad	274	1.85
133	Chandigarh	273	1.84
270	Jaipur	259	1.75
201	Faridabad	227	1.53
447	Muzaffrpur	159	1.07
382	Ludhiana	158	1.07
320	Kanpur	145	0.98
621	Surat	140	0.94
473	Noida	129	0.87
102	Bhopal	125	0.84
240	Guwahati	118	0.80
154	Coimbatore	69	0.47
679	Visakhapatnam	69	0.47
380	LowerParel	65	0.44
477	PNQ	62	0.42
273	Jalandhar	54	0.36
220	Goa	52	0.35
25	Anantapur	51	0.34
261	Hubli	47	0.32

In [69]:
```
plt.figure(figsize = (10, 10))
sns.barplot(data = df_source_city,
            x = df_source_city['trip_uuid'],
            y = df_source_city['source_city'])
plt.plot()
```

Out[69]: []



- We can see from the above plot that maximum trips originated from Mumbai city followed by Gurgaon, delhi, Bengaluru, Bhiwandi, that means the seller base is strong in these cities.

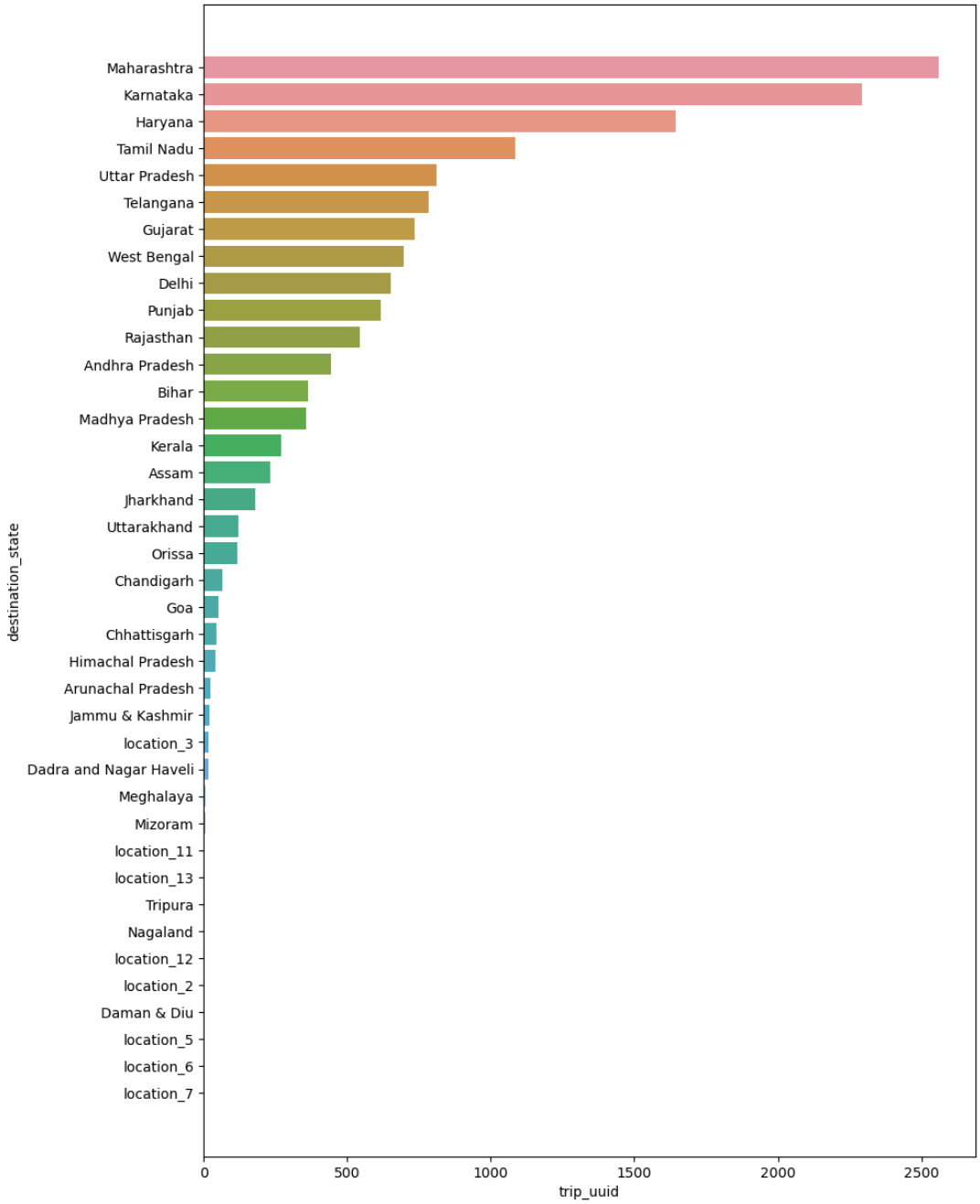
In [70]:
```
# We can find the distribution of number of trips which ended in differ
df_destination_state = df2.groupby(by = 'destination_state')['trip_uuid
df_destination_state['percent'] = np.round(df_destination_state['trip_u
df_destination_state = df_destination_state.sort_values(by = 'trip_uuid
df_destination_state.head()
```

Out[70]:

	destination_state	trip_uuid	percent
18	Maharashtra	2561	17.28
15	Karnataka	2294	15.48
11	Haryana	1643	11.09
25	Tamil Nadu	1084	7.32
28	Uttar Pradesh	811	5.47

In [71]:
```
plt.figure(figsize = (10,15))
sns.barplot(data = df_destination_state,
            x = df_destination_state['trip_uuid'],
            y = df_destination_state['destination_state'])
plt.plot()
```

Out[71]: []



- We can see from the above plot that maximum trips ended in the state of Maharastra followed by Karnataka and Haryana. It means the number of orders placed in these states are significantly high.

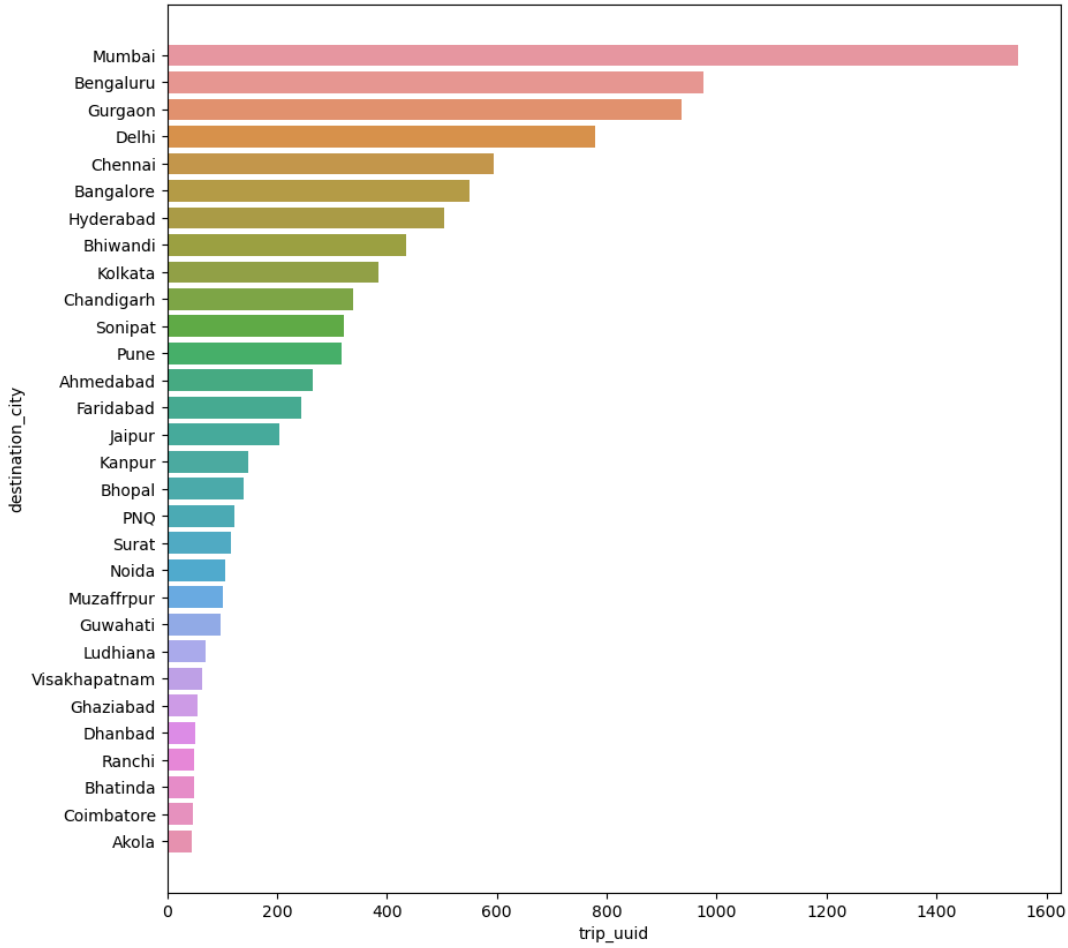
In [72]:
```
# We can find best 30 cities based on the number of trips ended from di
df_destination_city = df2.groupby(by = 'destination_city')['trip_uuid']
df_destination_city['percent'] = np.round(df_destination_city['trip_uui
df_destination_city = df_destination_city.sort_values(by = 'trip_uuid',
df_destination_city
```

Out[72]:

	destination_city	trip_uuid	percent
515	Mumbai	1548	10.45
96	Bengaluru	975	6.58
282	Gurgaon	936	6.32
200	Delhi	778	5.25
163	Chennai	595	4.02
72	Bangalore	551	3.72
308	Hyderabad	503	3.39
115	Bhiwandi	434	2.93
418	Kolkata	384	2.59
158	Chandigarh	339	2.29
724	Sonipat	322	2.17
612	Pune	317	2.14
4	Ahmedabad	265	1.79
242	Faridabad	244	1.65
318	Jaipur	205	1.38
371	Kanpur	148	1.00
117	Bhopal	139	0.94
559	PNQ	122	0.82
739	Surat	117	0.79
552	Noida	106	0.72
521	Muzaffrpur	102	0.69
284	Guwahati	98	0.66
448	Ludhiana	70	0.47
797	Visakhapatnam	64	0.43
259	Ghaziabad	56	0.38
208	Dhanbad	50	0.34
639	Ranchi	49	0.33
110	Bhatinda	48	0.32
183	Coimbatore	47	0.32
9	Akola	45	0.30

In [73]:
```
plt.figure(figsize = (10,10))
sns.barplot(data = df_destination_city,
            x = df_destination_city['trip_uuid'],
            y = df_destination_city['destination_city'])
plt.plot()
```

Out[73]: []



- We can see from the above plot that Maximum trips ended in Mumbai city followed by Bengaluru and Gurugaon. It means the number of orders placed in these cities are significantly high.

In [74]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual
                     'actual_time', 'osrm_time', 'osrm_distance', 'segme
                     'segment_osrm_time', 'segment_osrm_distance']
sns.pairplot(data = df2,
             vars = numerical_columns,
             kind = 'reg',
             hue = 'route_type',
             markers = '.')
plt.plot()
```

Out[74]: []

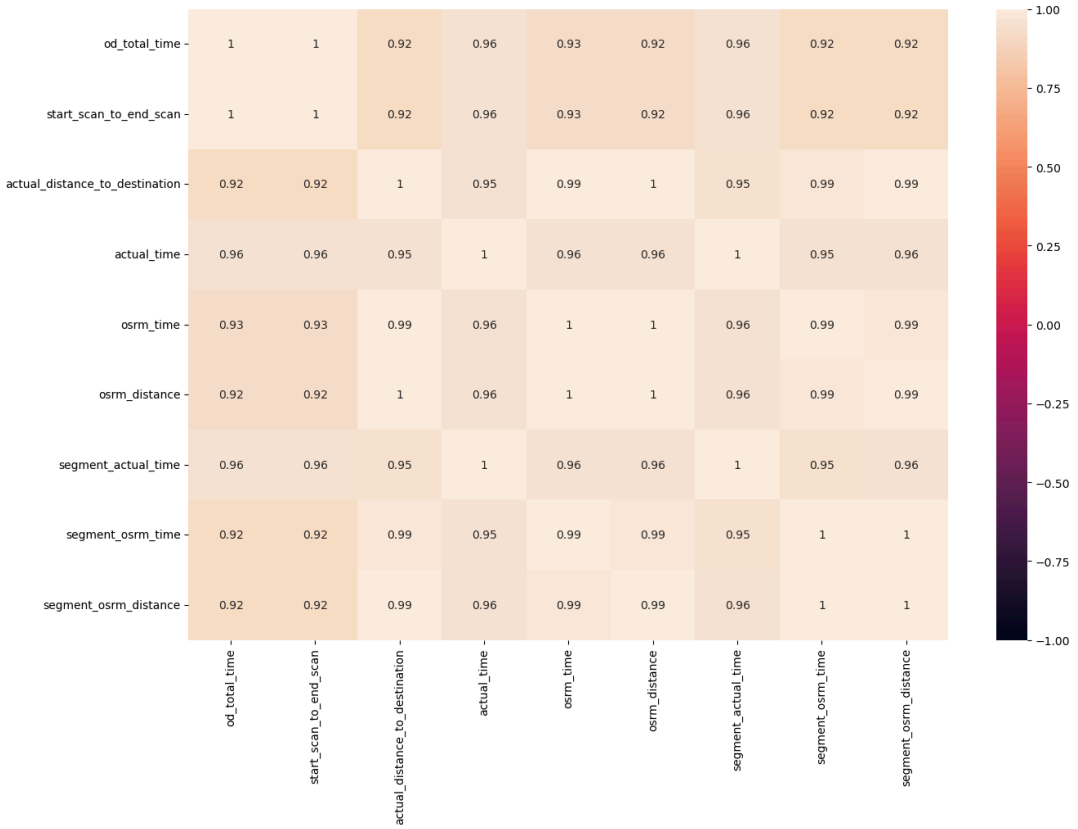
In [75]:
```
df_corr = df2[numerical_columns].corr()
df_corr
```

Out[75]:

	od_total_time	start_scan_to_end_scan	actual_distance_to
od_total_time	1.000000	0.999999	
start_scan_to_end_scan	0.999999	1.000000	
actual_distance_to_destination	0.918222	0.918308	
actual_time	0.961094	0.961147	
osrm_time	0.926516	0.926571	
osrm_distance	0.924219	0.924299	
segment_actual_time	0.961119	0.961171	
segment_osrm_time	0.918490	0.918561	
segment_osrm_distance	0.919199	0.919291	

In [76]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, vmin = -1, vmax = 1, annot = True)
plt.plot()
```

Out[76]: []



- Very High correlation (>0.9) exists between columns from all the numerical columns specified above.

3. Indepth Analysis and Feature Engineering

Set up Null Hypothesis

- *Null Hypothesis (H0):* od_total_time (Total Trip time) and start_scan_to_end_scan (Expected total Trip time) are same.
- *Alternate Hypothesis (HA):* od_total_time (Total Trip time) and start_scan_to_end_scan (Expected total Trip time) are different.

Cheking for basic assumption for the Hypothesis

- Distribution check using QQ Plot
- Homogenity of variance using Lavene's Test

Defining Test statistics: Distribution of T under H0

- If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

Compute the P-Value and fix the Value of Alpha

- We set our Alpha to be 0.05

Compare P-Value and Alpha

Based of P-Value We will reject or fail to reject Null hypothesis (H0)

- If P-Value < Alpha We reject the Null Hypothesis (H0)
- If P-Value > Alpha We fail to reject the Null Hypothesis

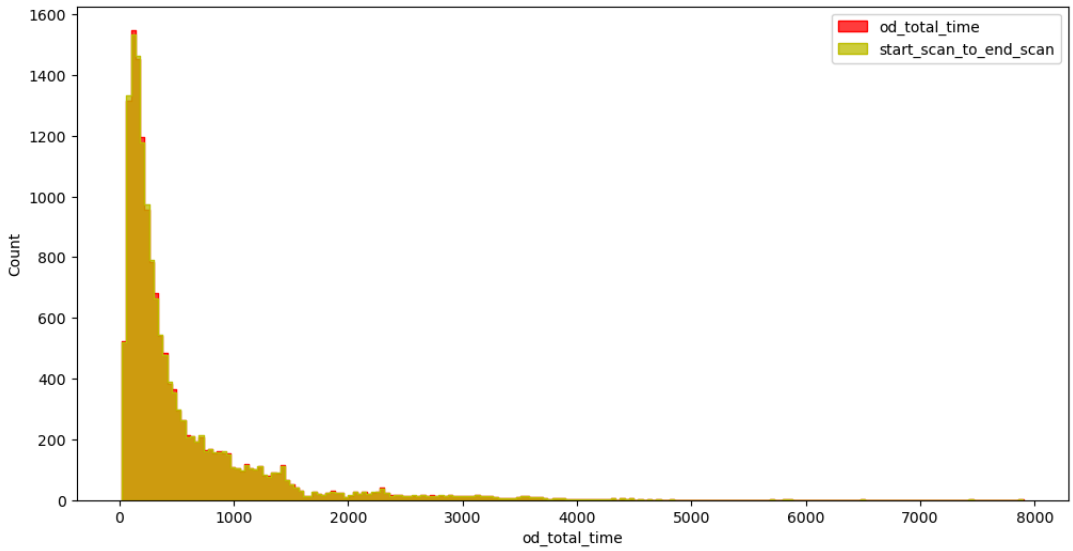
In [77]:
```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```

Out[77]:

	od_total_time	start_scan_to_end_scan
count	14817.000000	14817.000000
mean	531.697630	530.810016
std	658.868223	658.705957
min	23.460000	23.000000
25%	149.930000	149.000000
50%	280.770000	280.000000
75%	638.200000	637.000000
max	7898.550000	7898.000000

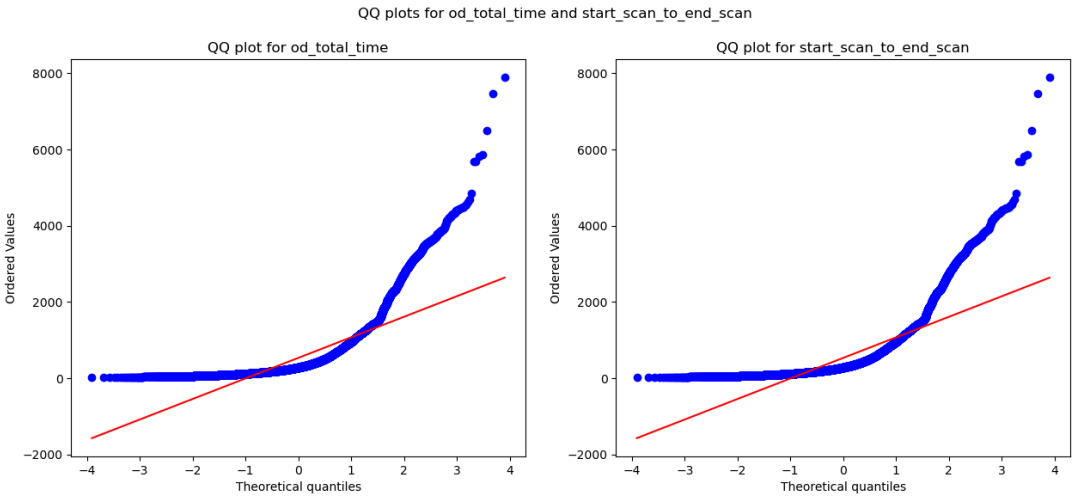
In [78]:
```
# We can find the sample followes the normal distribution or not:
plt.figure(figsize = (12, 6))
sns.histplot(df2['od_total_time'], element = 'step', color = 'red')
sns.histplot(df2['start_scan_to_end_scan'], element = 'step', color = '
plt.legend(['od_total_time', 'start_scan_to_end_scan'])
plt.plot()
```

Out[78]: []



In [79]:
```
# Checking the Distribution using QQ Plot:
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for od_total_time and start_scan_to_end_scan')
spy.probplot(df2['od_total_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for od_total_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['start_scan_to_end_scan'], plot = plt, dist = 'norm')
plt.title('QQ plot for start_scan_to_end_scan')
plt.plot()
```

Out[79]: []



We can see from the above plot that all sample do not follow normal distribution

Applying Shapiro-wilk test for normality

- H0: The sample follows normal diostribution
- HA: The sample does not follow normal distribution Alpha = 0.05

In [80]:
```
test_stat, p_value = spy.shapiro(df2['od_total_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [81]:
```
test_stat, p_value = spy.shapiro(df2['start_scan_to_end_scan'].sample(5
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [82]:
```
# Transforming the data using boxcox transformation to check if the tra

transformed_od_total_time = spy.boxcox(df2['od_total_time'])[0]
test_stat, p_value = spy.shapiro(transformed_od_total_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.172770042757021e-25
The sample does not follow normal distribution
```

In [83]:
```
transformed_start_scan_to_end_scan = spy.boxcox(df2['start_scan_to_end_
test_stat, p_value = spy.shapiro(transformed_start_scan_to_end_scan)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.0471322892609475e-24
The sample does not follow normal distribution
```

Even after applying the boxcox transformation on each of the "od_total_time" and "start_scan_to_end_scan" columns, the distributions do not follow normal distribution.

In [84]:
```
#Homogeneity of Variances using Lavene's test
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['od_total_time'], df2['start_scan_t
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.9668007217581142
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [85]:
```
test_stat, p_value = spy.mannwhitneyu(df2['od_total_time'], df2['start_
print('P-value :',p_value)
```

```
P-value : 0.7815123224221716
```

Since p-value > alpha therfore it can be concluded that od_total_time and start_scan_to_end_scan are similar.

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 [86]:
```
df2[['actual_time', 'osrm_time']].describe()
```

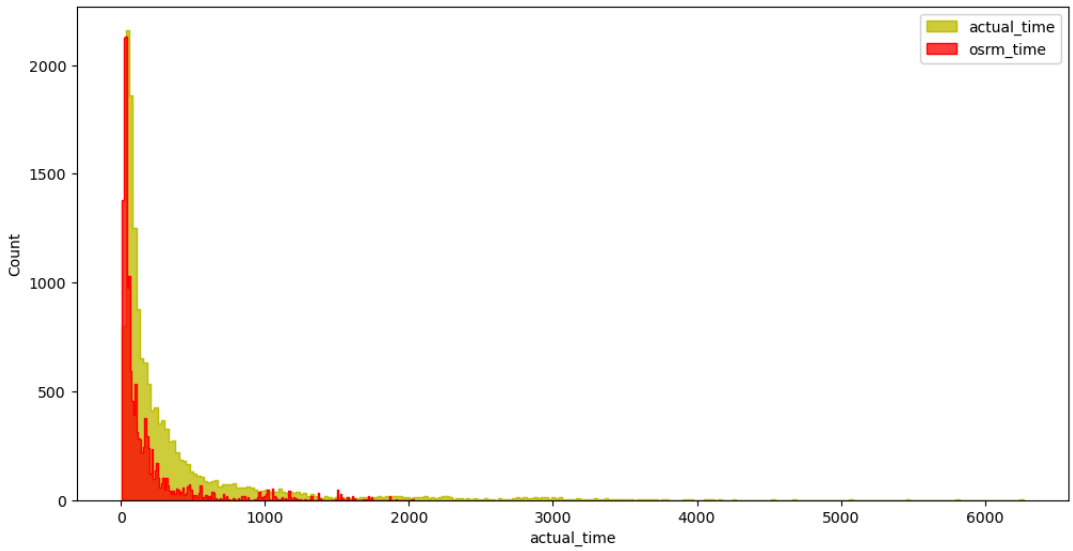
Out[86]:

	actual_time	osrm_time
count	14817.000000	14817.000000
mean	357.143768	161.384018
std	561.395020	271.362549
min	9.000000	6.000000
25%	67.000000	29.000000
50%	149.000000	60.000000
75%	370.000000	168.000000
max	6265.000000	2032.000000

In [87]:
```
# Visual Tests to know if the samples follow normal distribution

plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'y')
sns.histplot(df2['osrm_time'], element = 'step', color = 'r')
plt.legend(['actual_time', 'osrm_time'])
plt.plot()
```

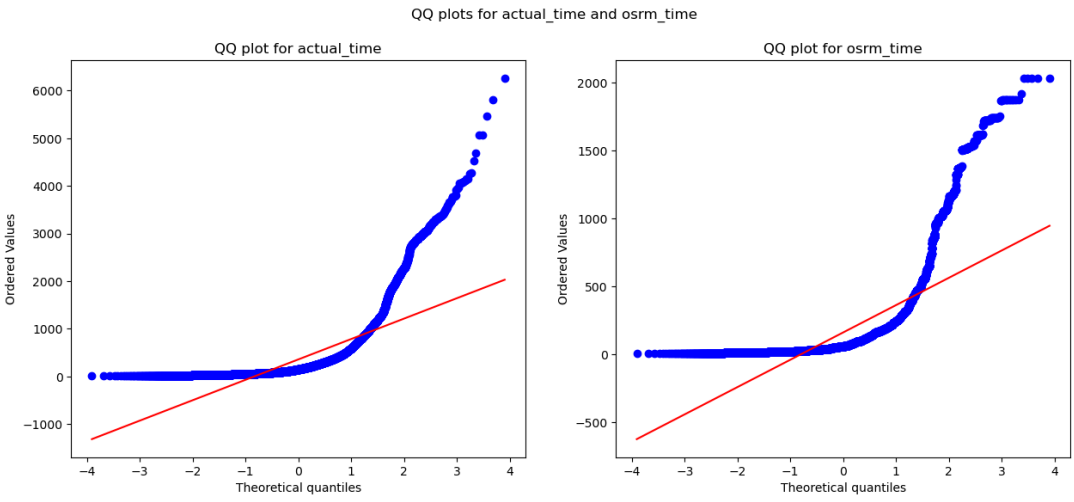
Out[87]: []



In [88]:
```
# Distribution check using QQ Plot

plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and osrm_time')
spy.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.plot()
```

Out[88]: []



We can see from the above plot that the sample do not follows the normal distribution

Applying Shapiro-wilk test for normality

- H0: The sample follows normal diostribution
- HA: The sample doesn't follow normal distribution

In [89]:
```
# Alpha = 0.05

test_stat, p_value = spy.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [90]:
```
test_stat, p_value = spy.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [91]:
```
# Transforming the data using boxcox transformation to check if the tra

transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
The sample does not follow normal distribution
```

In [92]:
```
transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
The sample does not follow normal distribution
```

Even after applying the boxcox transformation on each of the "actual_time" and "osrm_time" columns, the distributions do not follow normal distribution.

In [93]:
```
#Homogeneity of Variances using Lavene's test
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 1.871098057987424e-220
The samples do not have  Homogenous Variance
```

Since the samples do not follow any of the assumptions T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [94]:
```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'], df2['osrm_tim
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 0.0
The samples are not similar
```

Since p-value < alpha therfore it can be concluded that actual_time and osrm_time are not similar.

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 [95]:
```
df2[['actual_time', 'segment_actual_time']].describe()
```

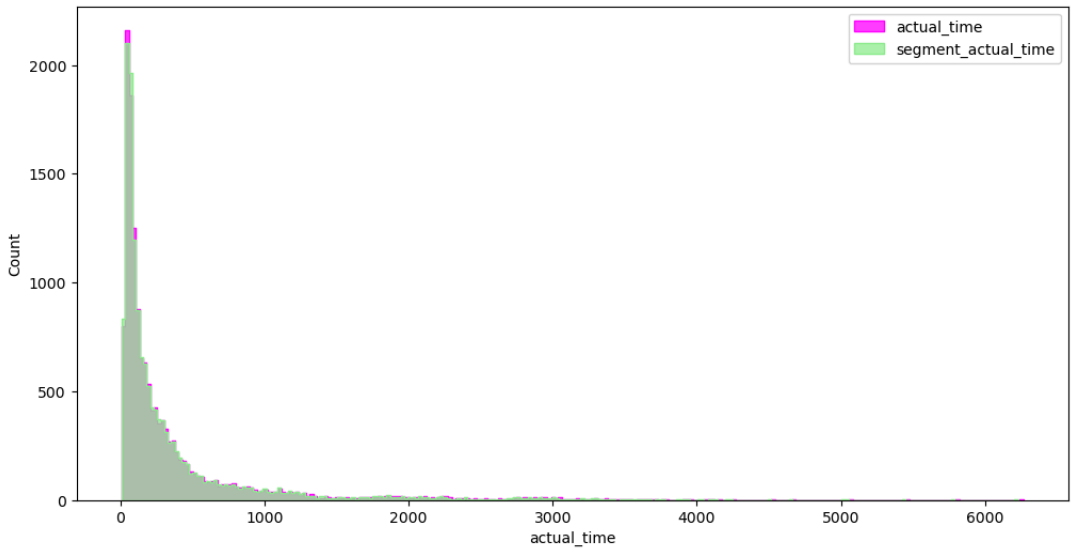
Out[95]:

	actual_time	segment_actual_time
count	14817.000000	14817.000000
mean	357.143768	353.892273
std	561.395020	556.246826
min	9.000000	9.000000
25%	67.000000	66.000000
50%	149.000000	147.000000
75%	370.000000	367.000000
max	6265.000000	6230.000000

In [96]:
```
# Visual Tests to know if the samples follow normal distribution

plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'magenta')
sns.histplot(df2['segment_actual_time'], element = 'step', color = 'lig
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```

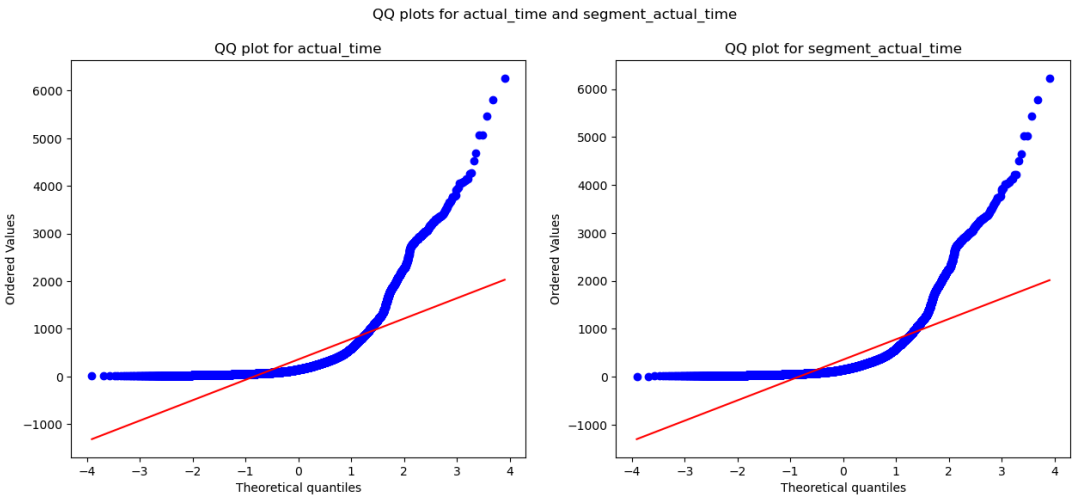
Out[96]: []



In [97]:
```
# Distribution check using QQ Plot

plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and segment_actual_time')
spy.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_actual_time')
plt.plot()
```

Out[97]: []



We can see form the above plot that the sample doesn't follow normal distribution

In [98]:
```
# Applying Shapiro-Wilk test for normality
# H0: The sample follows normal distribution
# HA: The sample does not follow normal distribution
#alpha = 0.05

test_stat, p_value = spy.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [99]:
```
test_stat, p_value = spy.shapiro(df2['segment_actual_time'].sample(5000
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [100]:
```
# Transforming the data using boxcox transformation to check if the tra

transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
The sample does not follow normal distribution
```

In [101]:
```
transformed_segment_actual_time = spy.boxcox(df2['segment_actual_time']
test_stat, p_value = spy.shapiro(transformed_segment_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 5.700074948787037e-29
The sample does not follow normal distribution
```

Even after applying the boxcox transformation on each of the "actual_time" and "segment_actual_time" columns, the distributions do not follow normal distribution.

In [102]:
```
# Homogeneity of Variances using Lavene's test
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'], df2['segment_actual
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.695502241317651
The samples have Homogenous Variance
```

Since the samples do not come from normal distribution T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [103]:
```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'], df2['segment_
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 0.4164235159622476
The samples are similar
```

Since p-value > alpha therfore it can be concluded that actual_time and segment_actual_time are similar.

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 [104]:
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

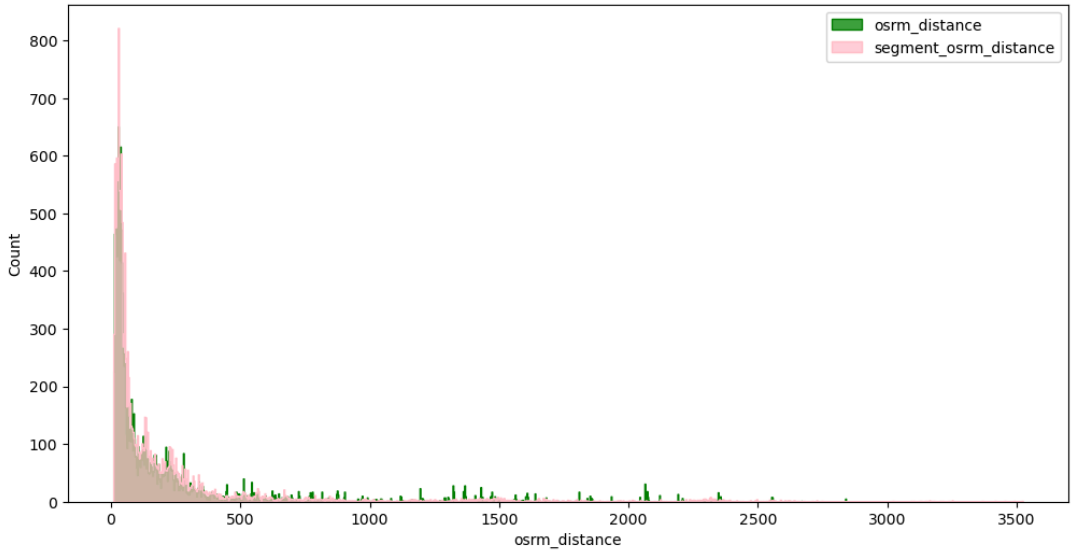
Out[104]:

	osrm_distance	segment_osrm_distance
count	14817.000000	14817.000000
mean	204.344711	223.201157
std	370.395508	416.628326
min	9.072900	9.072900
25%	30.819201	32.654499
50%	65.618805	70.154404
75%	208.475006	218.802399
max	2840.081055	3523.632324

In [105]:
```
# Visual Tests to know if the samples follow normal distribution

plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_distance'], element = 'step', color = 'green', b
sns.histplot(df2['segment_osrm_distance'], element = 'step', color = 'p
plt.legend(['osrm_distance', 'segment_osrm_distance'])
plt.plot()
```

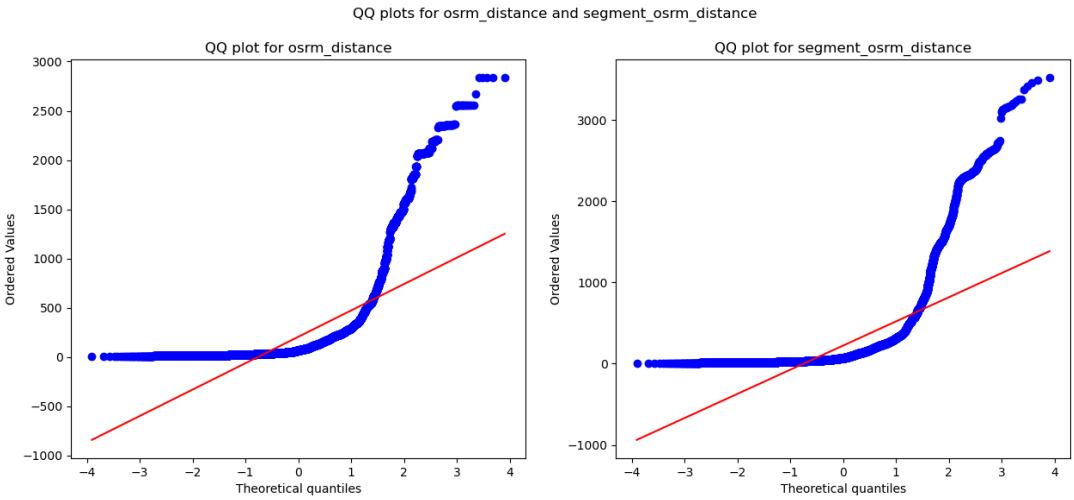
Out[105]: []



In [106]:
```
# Distribution check using QQ Plot

plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_distance and segment_osrm_distance')
spy.probplot(df2['osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_distance')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_distance')
plt.plot()
```

Out[106]: []



We can see from the above plot that the sample doesn't follow normal distribution

Applying Shapiro-Wilk test for normality

- H0: The sample follows normal distribution
- HA: The sample does not follow normal distribution

alpha = 0.05

In [107]:
```
test_stat, p_value = spy.shapiro(df2['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [108]:
```
test_stat, p_value = spy.shapiro(df2['segment_osrm_distance'].sample(50
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [109]:
```
# Transforming the data using boxcox transformation to check if the tra

transformed_osrm_distance = spy.boxcox(df2['osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.063104779582808e-41
The sample does not follow normal distribution
```

In [110]:
```
transformed_segment_osrm_distance = spy.boxcox(df2['segment_osrm_distan
test_stat, p_value = spy.shapiro(transformed_segment_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.049169406432229e-38
The sample does not follow normal distribution
```

Even after applying the boxcox transformation on each of the "osrm_distance" and "segment_osrm_distance" columns, the distributions do not follow normal distribution.

Homogeneity of Variances using Lavene's test

- Null Hypothesis(H0) - Homogenous Variance
- Alternate Hypothesis(HA) - Non Homogenous Variance

In [111]:
```
test_stat, p_value = spy.levene(df2['osrm_distance'], df2['segment_osrm
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.00020976006524780905
The samples do not have Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [112]:
```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_distance'], df2['segmen
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 9.509410818847664e-07
The samples are not similar
```

Since p-value < alpha therfore it can be concluded that osrm_distance and segment_osrm_distance are not similar.

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 [113]:
```
df2[['osrm_time', 'segment_osrm_time']].describe().T
```

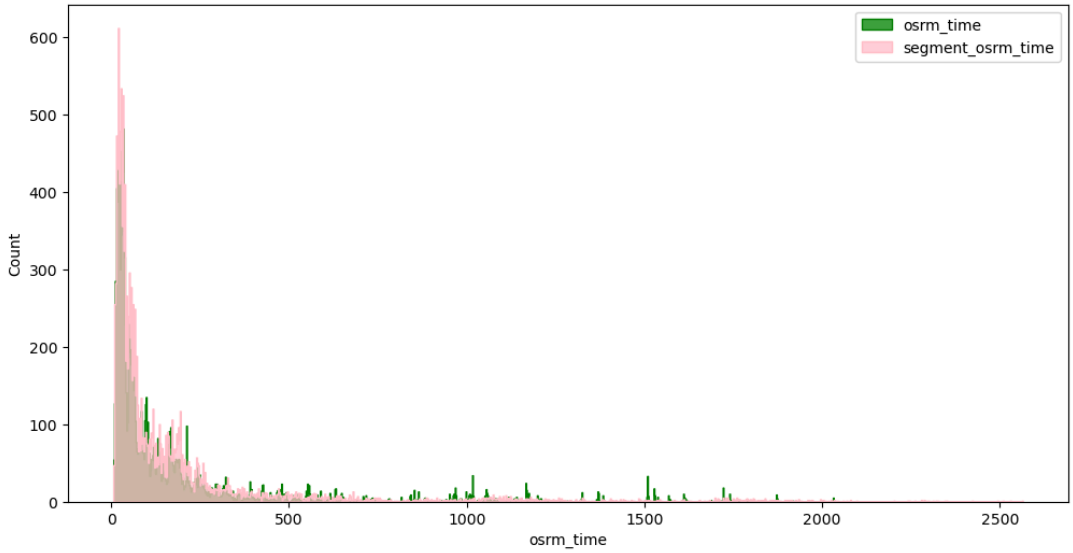
Out[113]:

	count	mean	std	min	25%	50%	75%	max
osrm_time	14817.0	161.384018	271.362549	6.0	29.0	60.0	168.0	2032.0
segment_osrm_time	14817.0	180.949783	314.541412	6.0	31.0	65.0	185.0	2564.0

In [114]:
```
# Visual Tests to know if the samples follow normal distribution

plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_time'], element = 'step', color = 'green', bins
sns.histplot(df2['segment_osrm_time'], element = 'step', color = 'pink'
plt.legend(['osrm_time', 'segment_osrm_time'])
plt.plot()
```

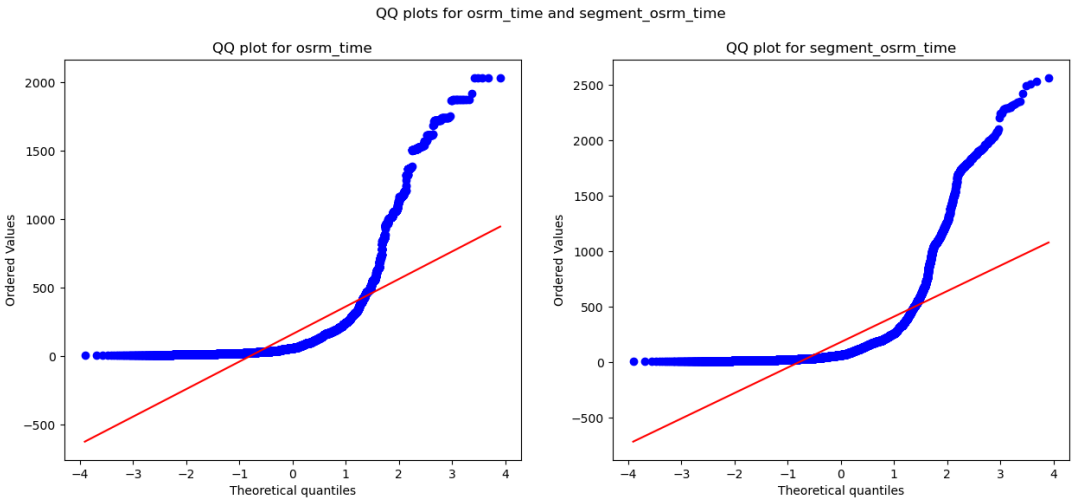
Out[114]: []



In [115]:
```
# Distribution checking using QQ Plot

plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_time and segment_osrm_time')
spy.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_time')
plt.plot()
```

Out[115]: []



We can see from the above plo that the sample doesn't follow normal distribution

Applying Shapiro-Wilk test for normality

- H0: The sample follows normal distribution
- HA: The sample does not follow normal distribution

alpha = 0.05

In [116]:
```
test_stat, p_value = spy.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [117]:
```
test_stat, p_value = spy.shapiro(df2['segment_osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

In [118]:
```
# Transforming the data using boxcox transformation to check if the tra

transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
The sample does not follow normal distribution
```

In [119]:
```
transformed_segment_osrm_time = spy.boxcox(df2['segment_osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 4.943039152219146e-34
The sample does not follow normal distribution
```

Even after applying the boxcox transformation on each of the "osrm_time" and "segment_osrm_time" columns, the distributions do not follow normal distribution.

Homogeneity of Variances using Lavene's test

- Null Hypothesis(H0) - Homogenous Variance
- Alternate Hypothesis(HA) - Non Homogenous Variance

In [120]:
```
test_stat, p_value = spy.levene(df2['osrm_time'], df2['segment_osrm_tim
print('p-value', p_value)

if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 8.349506135727595e-08
The samples do not have Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [121]:
```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_time'], df2['segment_os
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 2.2995370859748865e-08
The samples are not similar
```

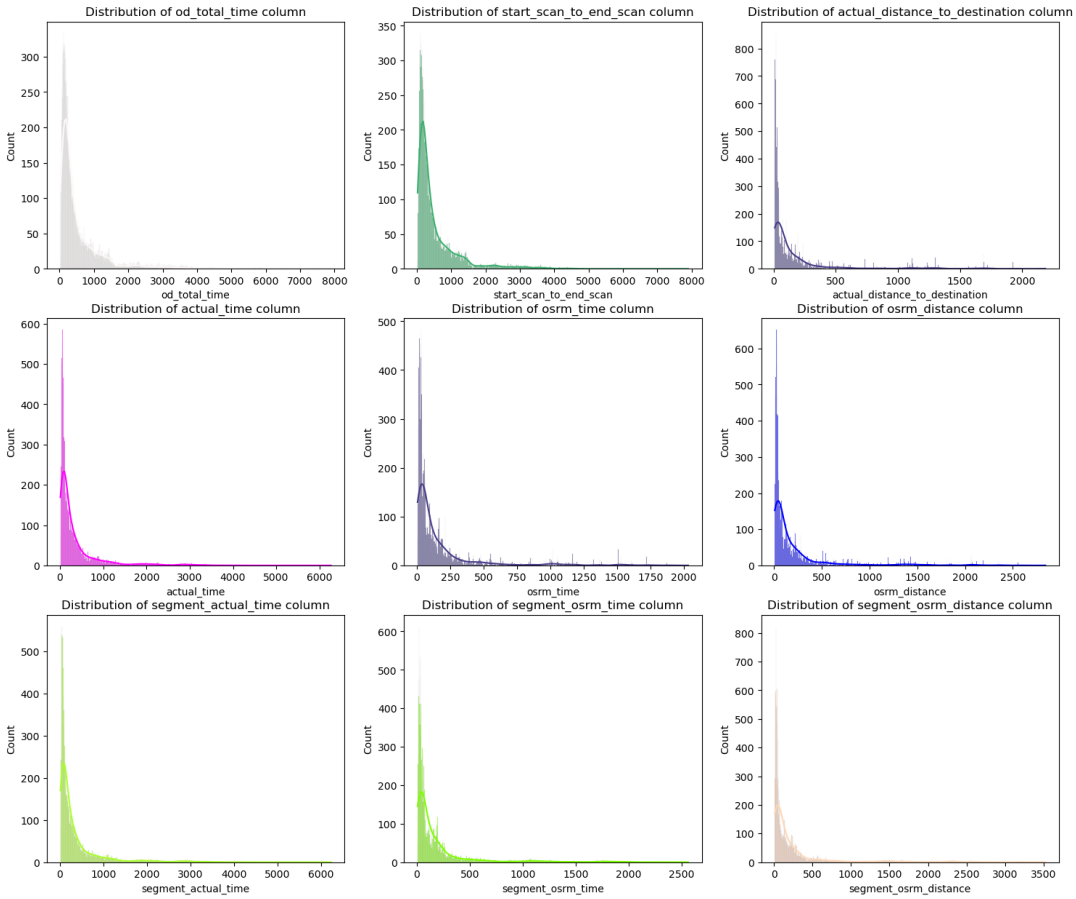
Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis

In [122]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual
                     'actual_time', 'osrm_time', 'osrm_distance', 'segme
                     'segment_osrm_time', 'segment_osrm_distance']
df2[numerical_columns].describe().T
```

Out[122]:

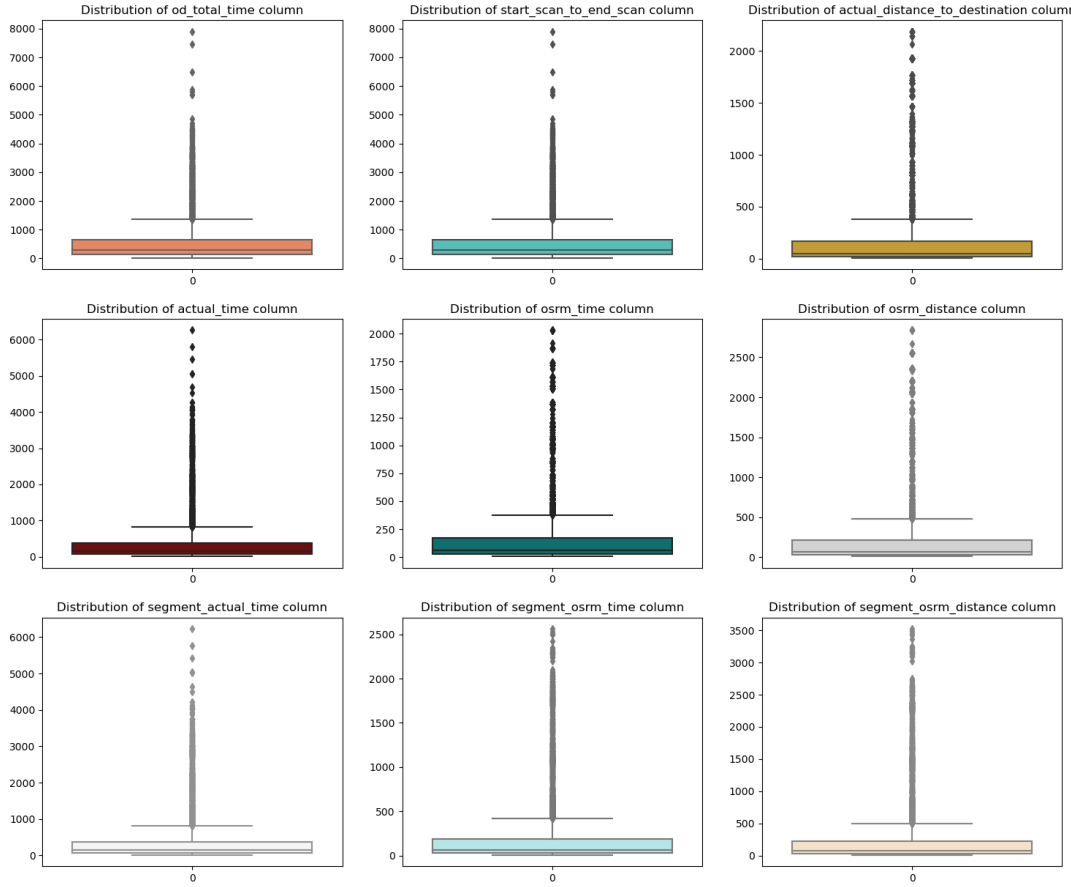
	count	mean	std	min	25%
od_total_time	14817.0	531.697630	658.868223	23.460000	149.930000
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000
actual_distance_to_destination	14817.0	164.477829	305.388123	9.002461	22.837238
actual_time	14817.0	357.143768	561.395020	9.000000	67.000000
osrm_time	14817.0	161.384018	271.362549	6.000000	29.000000
osrm_distance	14817.0	204.344711	370.395508	9.072900	30.819201
segment_actual_time	14817.0	353.892273	556.246826	9.000000	66.000000
segment_osrm_time	14817.0	180.949783	314.541412	6.000000	31.000000
segment_osrm_distance	14817.0	223.201157	416.628326	9.072900	32.654499

In [123]:
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.histplot(df2[numerical_columns[i]], bins = 1000, kde = True, co
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



It can be inferred from the above plot that data in all numerical columns are right skewed.

In [124]:
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.boxplot(df2[numerical_columns[i]], color = clr)
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



We can clearly see from the above plot that there are outliers in all the numerical columns that need to be treated.

In [125]:

```
# Detecting Outliers

for i in numerical_columns:
    Q1 = np.quantile(df2[i], 0.25)
    Q3 = np.quantile(df2[i], 0.75)
    IQR = Q3 - Q1
    LB = Q1 - 1.5 * IQR
    UB = Q3 + 1.5 * IQR
    outliers = df2.loc[(df2[i] < LB) | (df2[i] > UB)]
    print('Column :', i)
    print(f'Q1 : {Q1}')
    print(f'Q3 : {Q3}')
    print(f'IQR : {IQR}')
    print(f'LB : {LB}')
    print(f'UB : {UB}')
    print(f'Number of outliers : {outliers.shape[0]}')
    print('---------------------------------')
```

```
Column : od_total_time
Q1 : 149.93
Q3 : 638.2
IQR : 488.27000000000004
LB : -582.4750000000001
UB : 1370.605
Number of outliers : 1266
---------------------------------
Column : start_scan_to_end_scan
Q1 : 149.0
Q3 : 637.0
IQR : 488.0
LB : -583.0
UB : 1369.0
Number of outliers : 1267
---------------------------------
Column : actual_distance_to_destination
Q1 : 22.837238311767578
Q3 : 164.5832061767578
IQR : 141.74596786499023
LB : -189.78171348571777
UB : 377.20215797424316
Number of outliers : 1449
---------------------------------
Column : actual_time
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
LB : -387.5
UB : 824.5
Number of outliers : 1643
---------------------------------
Column : osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
LB : -179.5
UB : 376.5
Number of outliers : 1517
---------------------------------
Column : osrm_distance
Q1 : 30.81920051574707
Q3 : 208.47500610351562
IQR : 177.65580558776855
LB : -235.66450786590576
UB : 474.95871448516846
Number of outliers : 1524
---------------------------------
Column : segment_actual_time
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
LB : -385.5
UB : 818.5
Number of outliers : 1643
---------------------------------
Column : segment_osrm_time
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
LB : -200.0
```

```
UB : 416.0
Number of outliers : 1492
---------------------------------
Column : segment_osrm_distance
Q1 : 32.65449905395508
Q3 : 218.80239868164062
IQR : 186.14789962768555
LB : -246.56735038757324
UB : 498.02424812316895
Number of outliers : 1548
---------------------------------
```

The outliers present in our sample data can be the true outliers. It's best to remove outliers only when there is a sound reason for doing so. Some outliers represent natural variations in the population, and they should be left as is in the dataset.

Do one-hot encoding of categorical variables (like route_type)

In [126]:
```
# Get value counts before one-hot encoding

df2['route_type'].value_counts()
```

Out[126]:
```
route_type
Carting    8908
FTL        5909
Name: count, dtype: int64
```

In [127]:
```
# Perform one-hot encoding on categorical column route type
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df2['route_type'] = label_encoder.fit_transform(df2['route_type'])
```

In [128]:
```
# Get value counts after one-hot encoding

df2['route_type'].value_counts()
```

Out[128]:
```
route_type
0    8908
1    5909
Name: count, dtype: int64
```

In [129]:
```
# Get value counts for categorical variable 'data' before one-hot encod

df2['data'].value_counts()
```

Out[129]:
```
data
training    10654
test         4163
Name: count, dtype: int64
```

In [130]:
```
# Performing one-hot encoding on categorcal variable 'data'
label_encoder = LabelEncoder()
df2['data'] = label_encoder.fit_transform(df2['data'])
```

In [131]:
```
# Get value counts after one-hot encoding
df2['data'].value_counts()
```

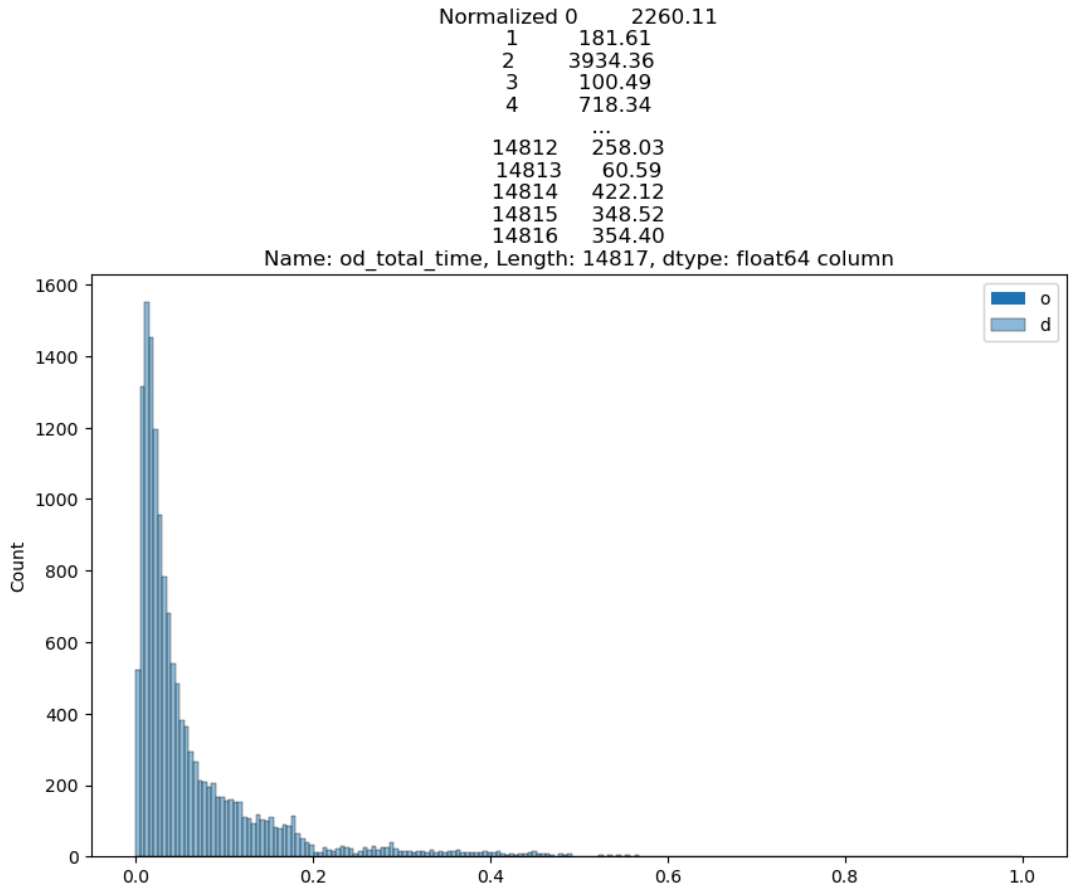
Out[131]:
```
data
1    10654
0     4163
Name: count, dtype: int64
```

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [133]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [134]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Normalized {df2['od_total_time']} column")
plt.legend('od_total_time')
plt.plot()
```

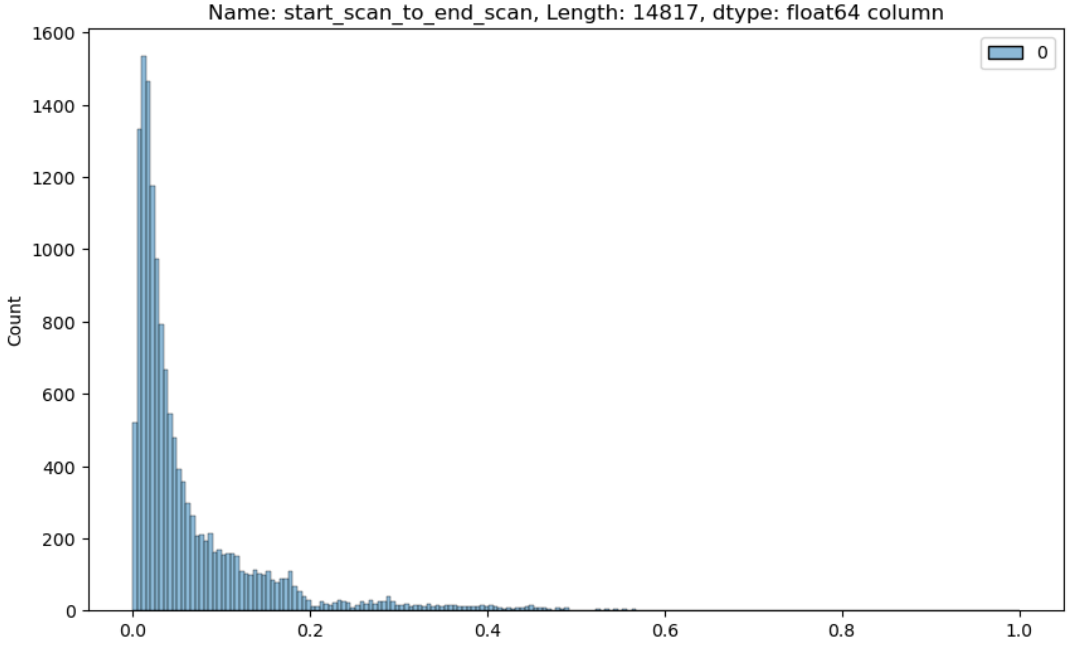
Out[134]:
```
[]
```



In [135]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().
sns.histplot(scaled)
plt.title(f"Normalized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

Out[135]: []

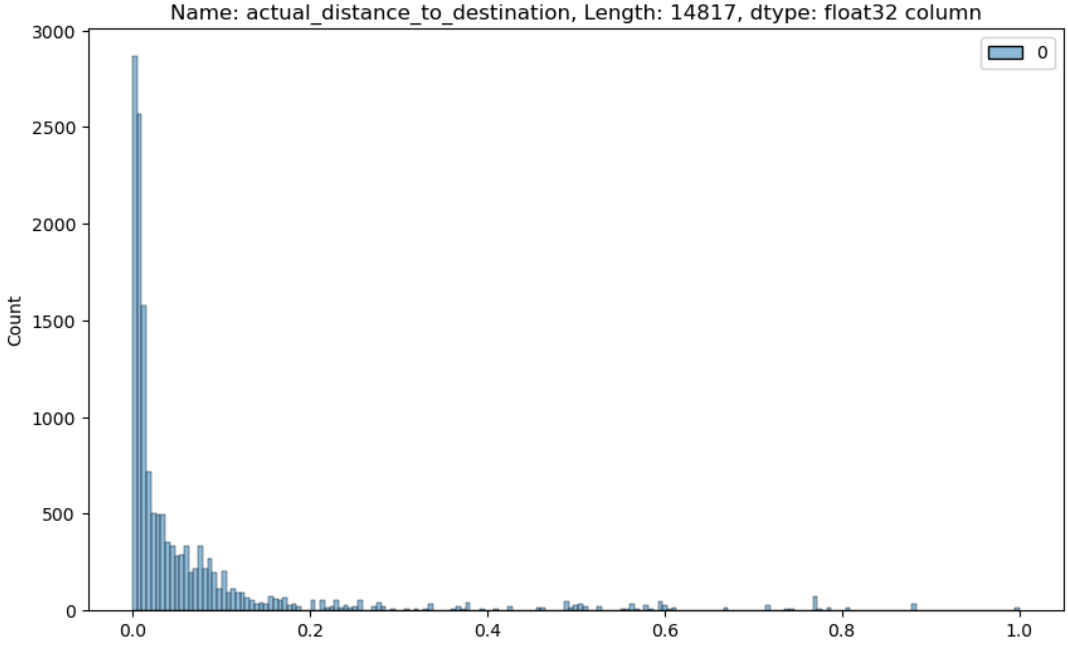
```
Normalized 0      2259.0
           1       180.0
           2      3933.0
           3       100.0
           4       717.0
                   ...
       14812       257.0
       14813        60.0
       14814       421.0
       14815       347.0
       14816       353.0
Name: start_scan_to_end_scan, Length: 14817, dtype: float64 column
```



In [136]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_distance_to_destination']} column")
plt.plot()
```

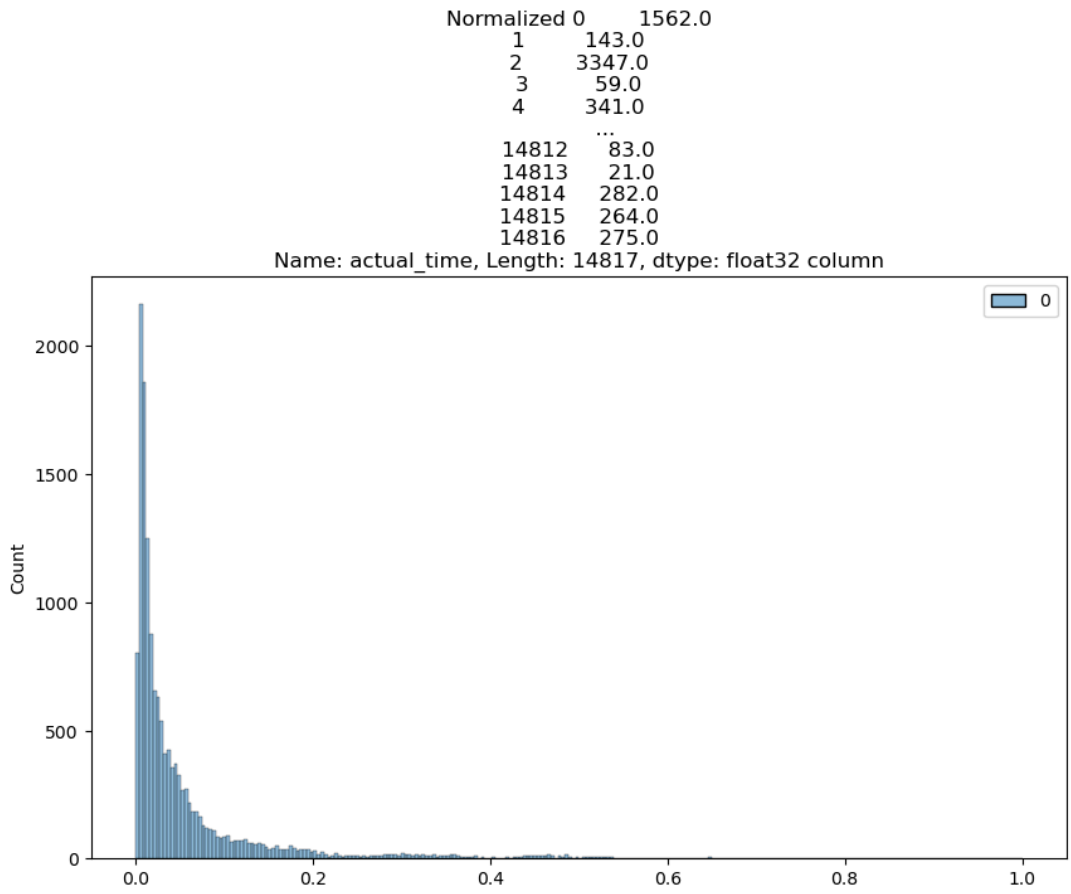
Out[136]: []

```
Normalized 0       824.732849
           1        73.186905
           2      1927.404297
           3        17.175274
           4       127.448502
                      ...
       14812        57.762333
       14813        15.513784
       14814        38.684837
       14815       134.723831
       14816        66.081528
Name: actual_distance_to_destination, Length: 14817, dtype: float32 column
```



In [137]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_time']} column")
plt.plot()
```

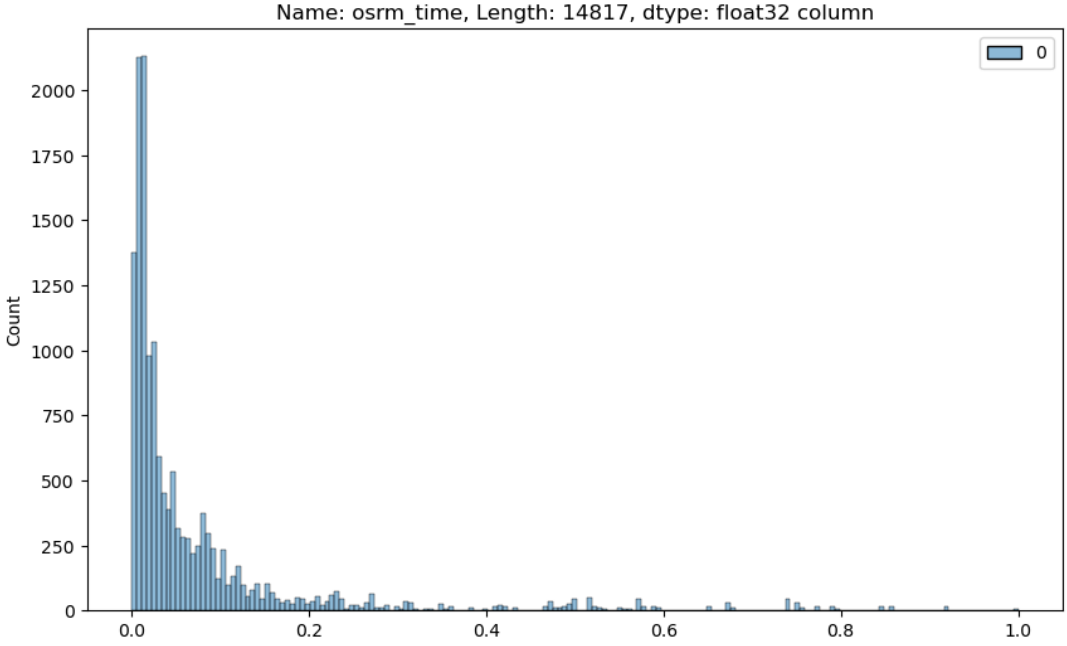
Out[137]: []



In [138]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['osrm_time'].to_numpy().reshape(-1, 1
sns.histplot(scaled)
plt.title(f"Normalized {df2['osrm_time']} column")
plt.plot()
```

Out[138]: []

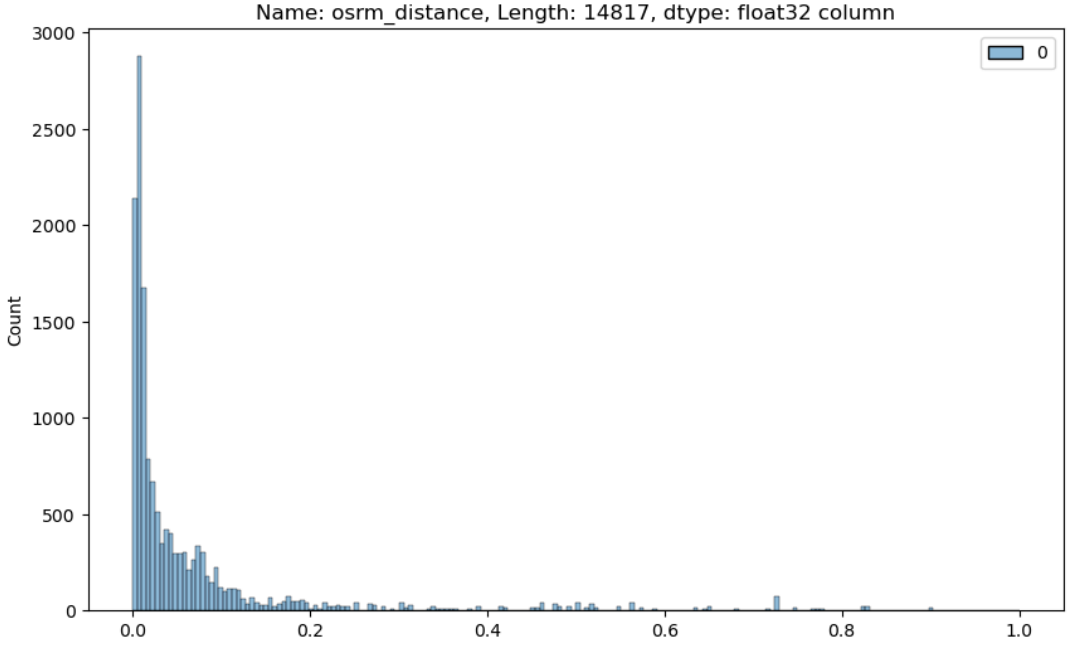
```
Normalized 0       717.0
           1        68.0
           2      1740.0
           3        15.0
           4       117.0
                    ...
       14812        62.0
       14813        12.0
       14814        48.0
       14815       179.0
       14816        68.0
Name: osrm_time, Length: 14817, dtype: float32 column
```



In [139]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Normalized {df2['osrm_distance']} column")
plt.plot()
```

Out[139]: []

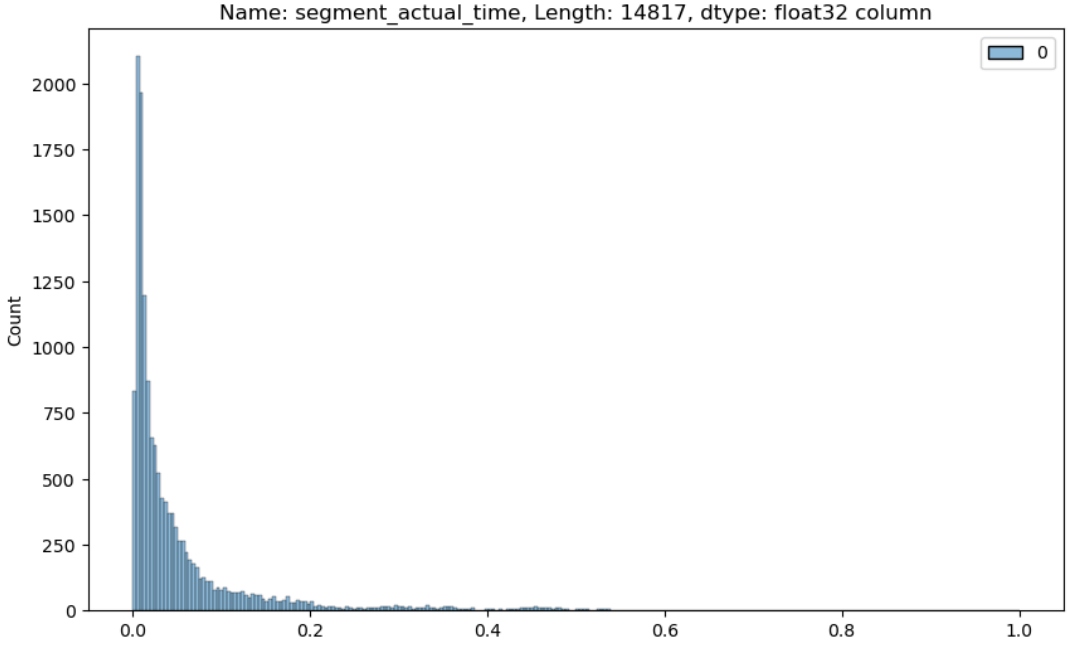
```
Normalized 0       991.352295
           1        85.111000
           2      2354.066650
           3        19.680000
           4       146.791794
                      ...
       14812        73.462997
       14813        16.088200
       14814        58.903702
       14815       171.110306
       14816        80.578705
Name: osrm_distance, Length: 14817, dtype: float32 column
```



In [140]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().res
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_actual_time']} column")
plt.plot()
```

Out[140]: []

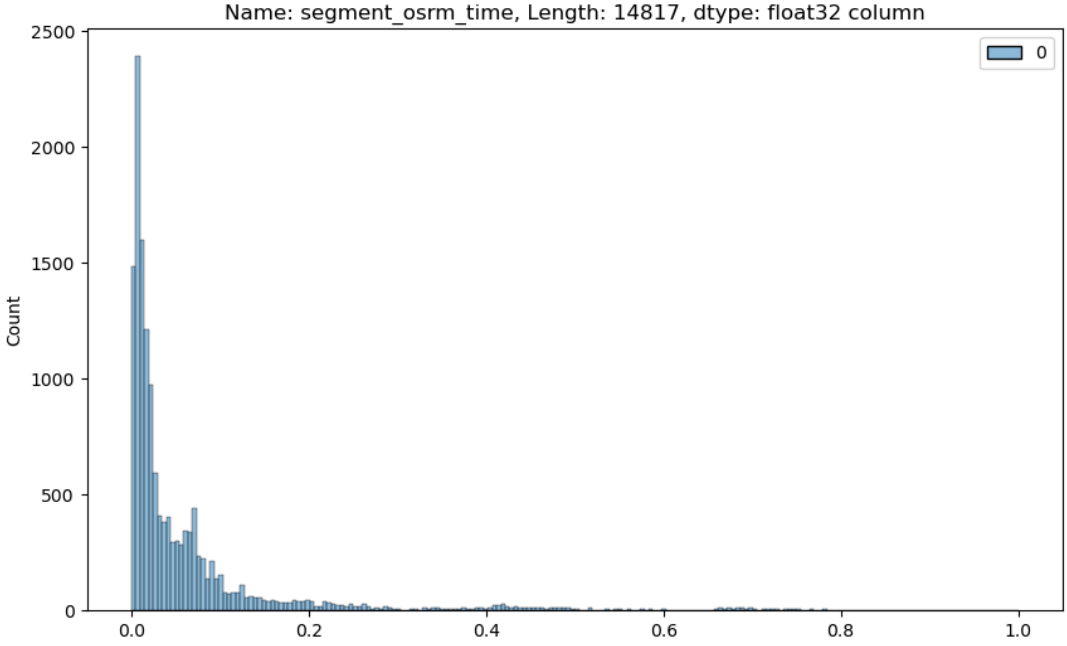
```
Normalized 0      1548.0
           1       141.0
           2      3308.0
           3        59.0
           4       340.0
                   ...
       14812       82.0
       14813       21.0
       14814      281.0
       14815      258.0
       14816      274.0
Name: segment_actual_time, Length: 14817, dtype: float32 column
```



In [141]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_osrm_time'].to_numpy().resha
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_osrm_time']} column")
plt.plot()
```

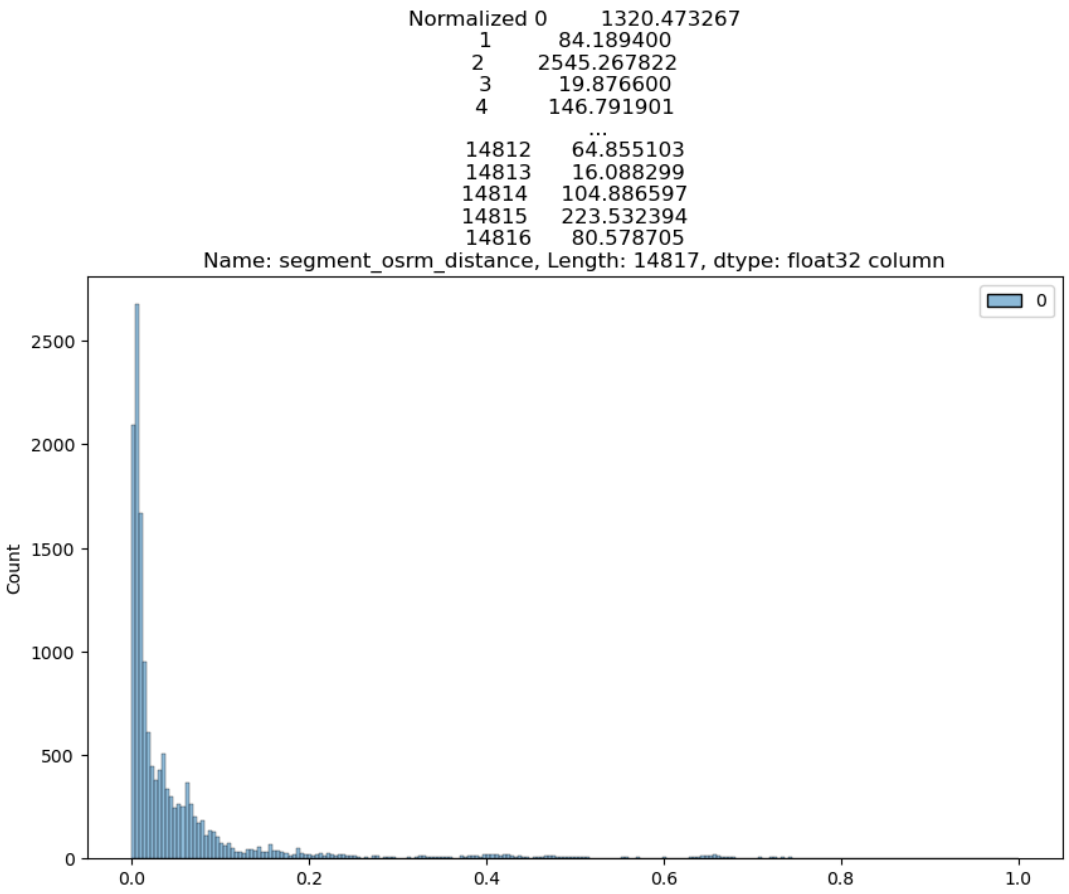
Out[141]: []

```
Normalized 0      1008.0
           1        65.0
           2      1941.0
           3        16.0
           4       115.0
                   ...
       14812        62.0
       14813        11.0
       14814        88.0
       14815       221.0
       14816        67.0
Name: segment_osrm_time, Length: 14817, dtype: float32 column
```



In [142]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_osrm_distance'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_osrm_distance']} column")
plt.plot()
```

Out[142]: []



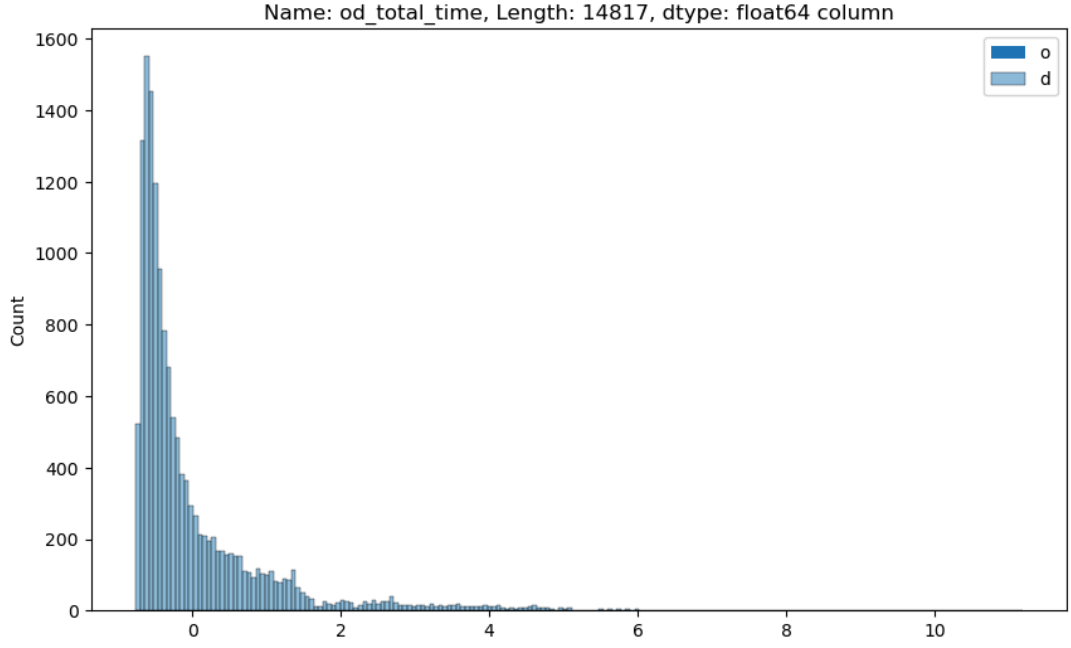
Column Standardization

In [144]:
```
from sklearn.preprocessing import StandardScaler
```

In [145]:
```
plt.figure(figsize = (10, 6))
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Standardized {df2['od_total_time']} column")
plt.legend('od_total_time')
plt.plot()
```

Out[145]: []

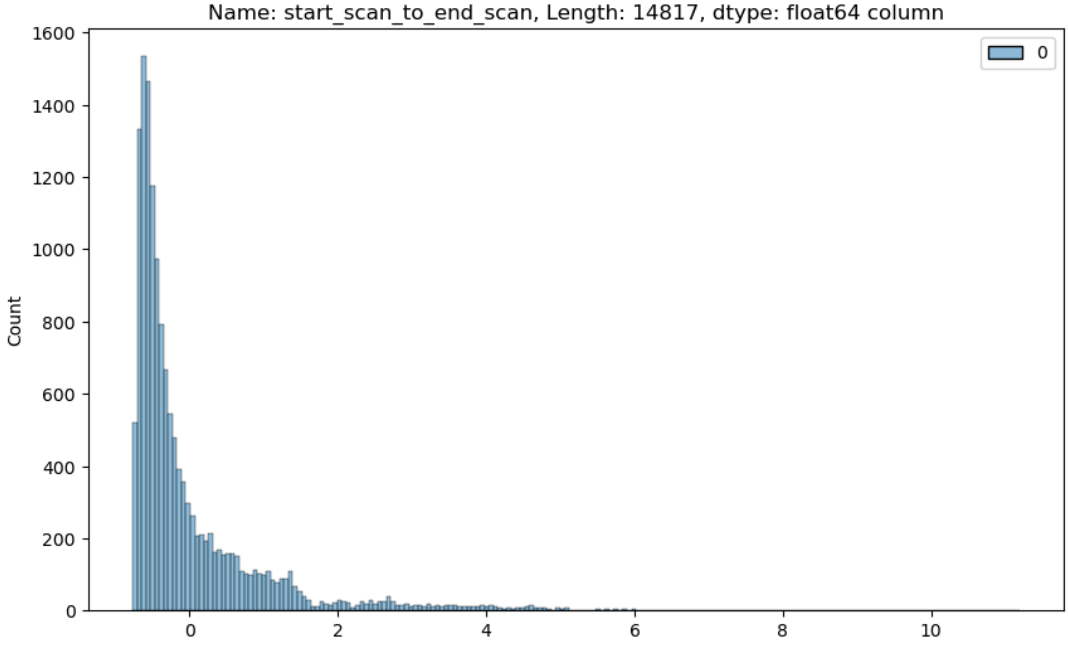
```
Standardized 0      2260.11
1       181.61
2      3934.36
3       100.49
4       718.34
...
14812    258.03
14813     60.59
14814    422.12
14815    348.52
14816    354.40
Name: od_total_time, Length: 14817, dtype: float64 column
```



In [146]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().
sns.histplot(scaled)
plt.title(f"Standardized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

Out[146]: []

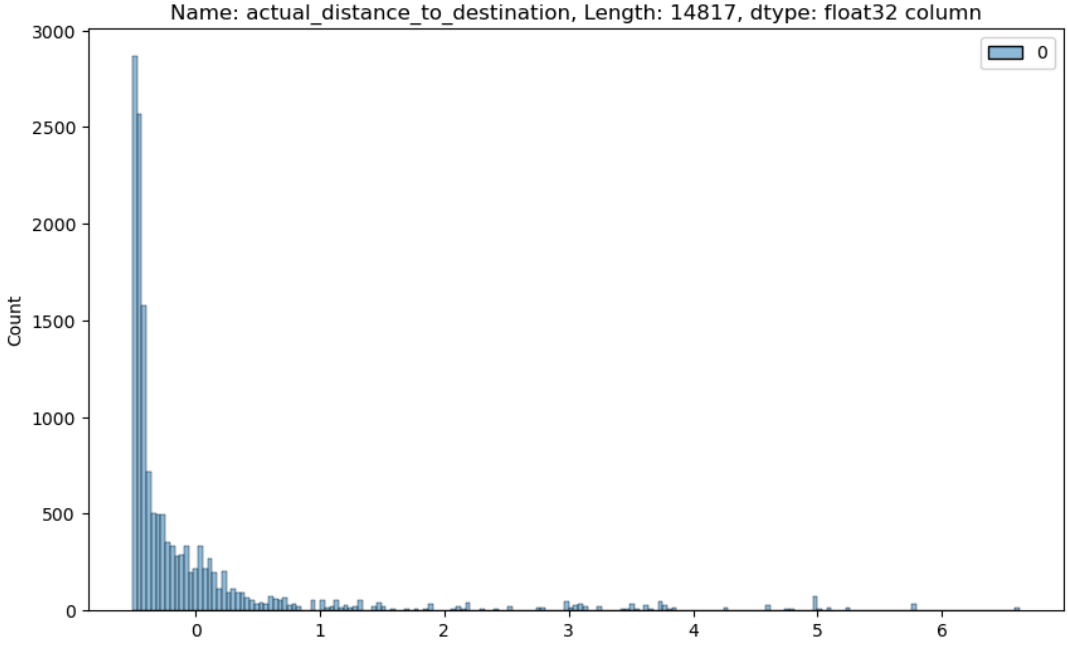
```
Standardized 0      2259.0
             1       180.0
             2      3933.0
             3       100.0
             4       717.0
                     ...
         14812       257.0
         14813        60.0
         14814       421.0
         14815       347.0
         14816       353.0
Name: start_scan_to_end_scan, Length: 14817, dtype: float64 column
```



In [147]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_distance_to_destination']} column
plt.plot()
```

Out[147]: []

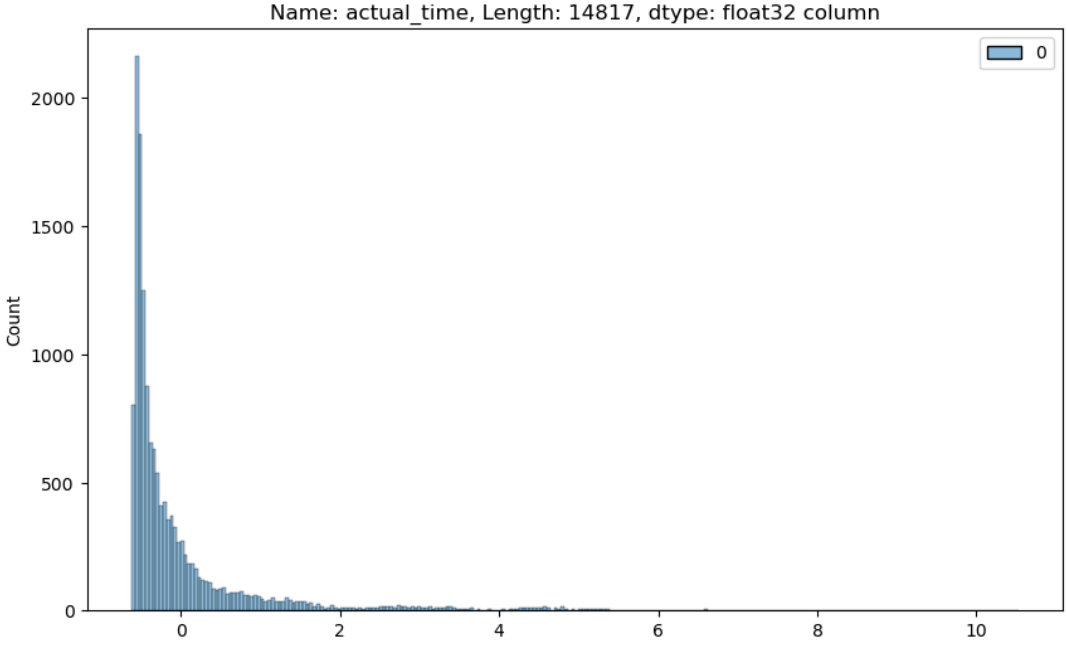
```
Standardized 0       824.732849
             1        73.186905
             2      1927.404297
             3        17.175274
             4       127.448502
                        ...
         14812        57.762333
         14813        15.513784
         14814        38.684837
         14815       134.723831
         14816        66.081528
Name: actual_distance_to_destination, Length: 14817, dtype: float32 column
```



In [148]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_time']} column")
plt.plot()
```

Out[148]: []

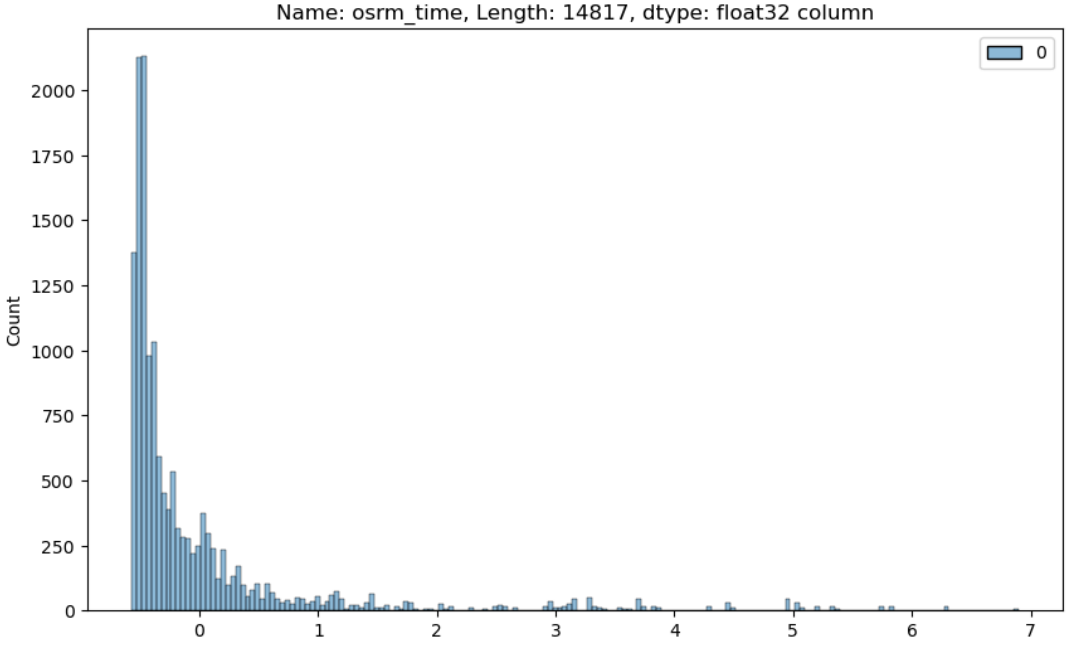
```
Standardized 0      1562.0
1       143.0
2      3347.0
3        59.0
4       341.0
...
14812    83.0
14813    21.0
14814   282.0
14815   264.0
14816   275.0
Name: actual_time, Length: 14817, dtype: float32 column
```



In [149]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['osrm_time'].to_numpy().reshape(-1, 1
sns.histplot(scaled)
plt.title(f"Standardized {df2['osrm_time']} column")
plt.plot()
```

Out[149]: []

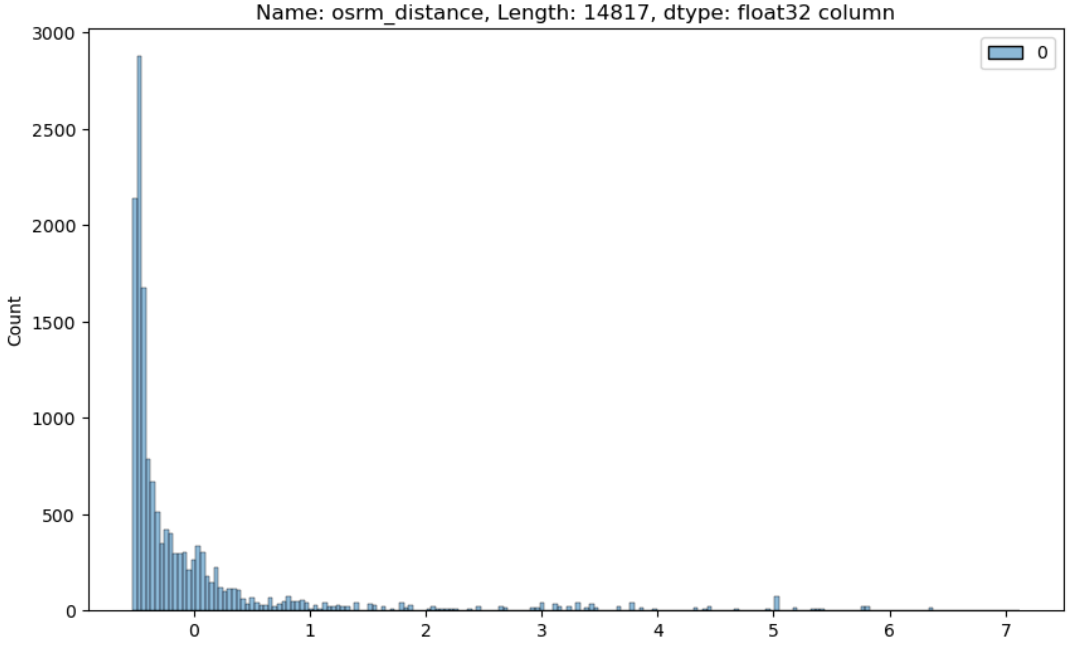
```
Standardized 0       717.0
             1        68.0
             2      1740.0
             3        15.0
             4       117.0
                      ...
         14812        62.0
         14813        12.0
         14814        48.0
         14815       179.0
         14816        68.0
Name: osrm_time, Length: 14817, dtype: float32 column
```



In [150]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Standardized {df2['osrm_distance']} column")
plt.plot()
```

Out[150]: []

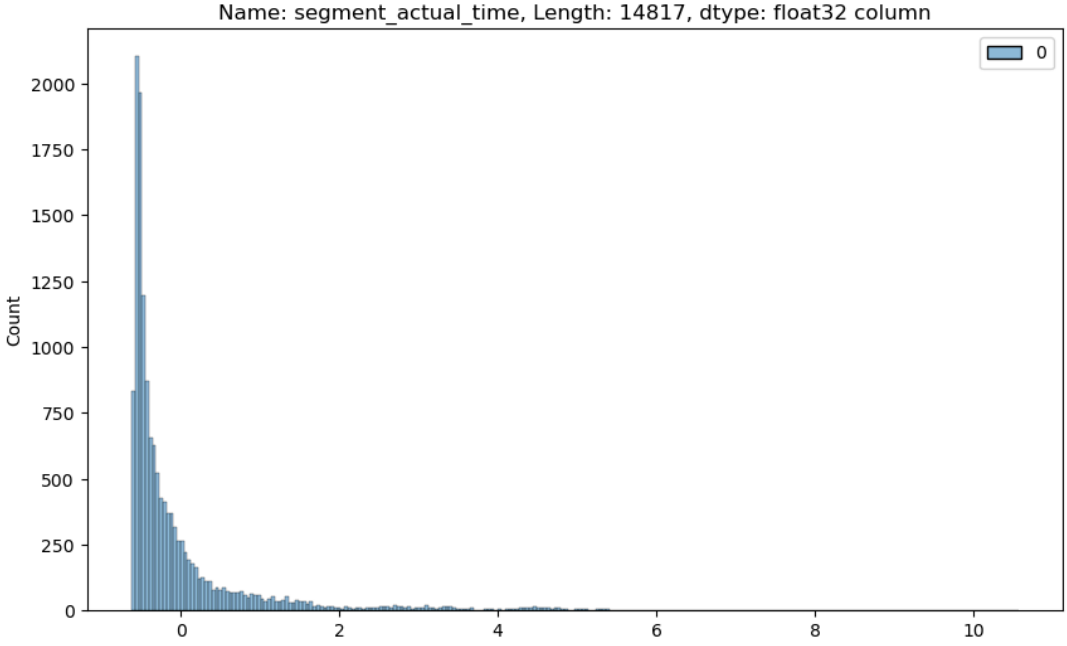
```
Standardized 0      991.352295
             1       85.111000
             2     2354.066650
             3       19.680000
             4      146.791794
                        ...
         14812       73.462997
         14813       16.088200
         14814       58.903702
         14815      171.110306
         14816       80.578705
Name: osrm_distance, Length: 14817, dtype: float32 column
```



In [151]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().res
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_actual_time']} column")
plt.plot()
```

Out[151]: []

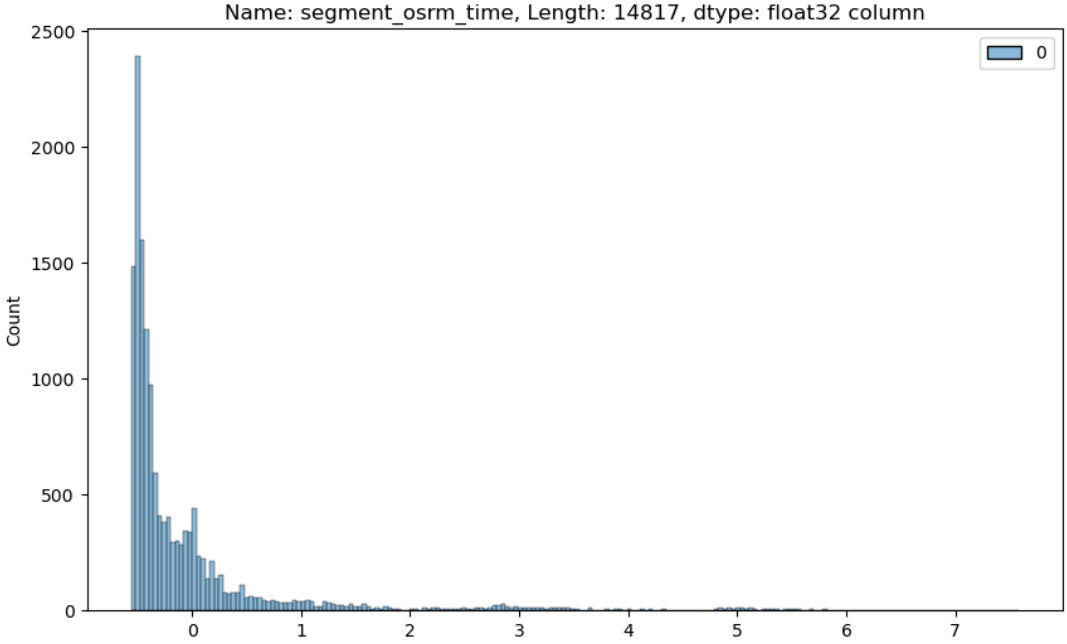
```
Standardized 0      1548.0
             1       141.0
             2      3308.0
             3        59.0
             4       340.0
                      ...
             14812    82.0
             14813    21.0
             14814   281.0
             14815   258.0
             14816   274.0
Name: segment_actual_time, Length: 14817, dtype: float32 column
```



In [152]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_osrm_time'].to_numpy().resha
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_osrm_time']} column")
plt.plot()
```

Out[152]: []

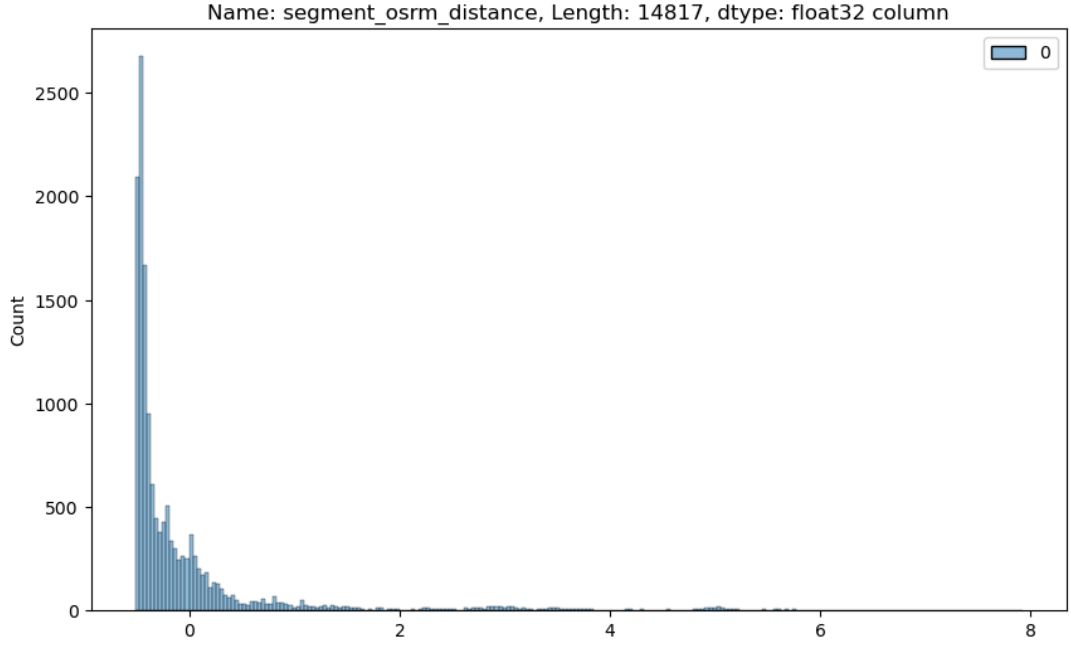
```
Standardized 0      1008.0
1        65.0
2      1941.0
3        16.0
4       115.0
...
14812    62.0
14813    11.0
14814    88.0
14815   221.0
14816    67.0
Name: segment_osrm_time, Length: 14817, dtype: float32 column
```



In [153]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_osrm_distance'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_osrm_distance']} column")
plt.plot()
```

Out[153]: []

```
Standardized 0      1320.473267
             1        84.189400
             2      2545.267822
             3        19.876600
             4       146.791901
                          ...
         14812        64.855103
         14813        16.088299
         14814       104.886597
         14815       223.532394
         14816        80.578705
Name: segment_osrm_distance, Length: 14817, dtype: float32 column
```



Business Insights:

- *The data is given from the period '2018-09-12 00:00:16' to '2018-10-08 03:00:24'.*
- *There are about 14817 unique trip IDs, 1508 unique source centers, 1481 unique destination_centers, 690 unique source cities, 806 unique destination cities.*
- *Most of the data is for testing than for training.*
- *Most common route type is Carting.*
- *The names of 14 unique location ids are missing in the data.*
- *The number of trips start increasing after the noon, becomes maximum at 10 P.M and then start decreasing.*
- *Maximum trips are created in the 38th week.*
- *Most orders come mid-month. That means customers usually make more orders in the mid of the month.*
- *Most orders are sourced from the states like Maharashtra, Karnataka, Haryana, Tamil Nadu, Telangana*
- *Maximum number of trips originated from Mumbai city followed by Gurgaon Delhi, Bengaluru and Bhiwandi. That means that the seller base is strong in these cities.*
- *Maximum number of trips ended in Maharashtra state followed by Karnataka, Haryana, Tamil Nadu and Uttar Pradesh. That means that the number of orders placed in these*

states is significantly high.

- *Maximum number of trips ended in Mumbai city followed by Bengaluru, Gurgaon, Delhi and Chennai. That means that the number of orders placed in these cities is significantly high.*
- *Most orders in terms of destination are coming from cities like bengaluru, mumbai, gurgaon, bangalore, Delhi.*
- *Features start_scan_to_end_scan and od_total_time(created feature) are statistically similar.*
- *Features actual_time & osrm_time are statitically different.*
- *Features start_scan_to_end_scan and segment_actual_time are statistically similar.*
- *Features osrm_distance and segment_osrm_distance are statistically different from each other.*
- *Both the osrm_time & segment_osrm_time are not statistically same.*

Recommendation:

- *The OSRM trip planning system needs to be improved. Discrepancies need to be catered to for transporters, if the routing engine is configured for optimum results.*
- *osrm_time and actual_time are different. Team needs to make sure this difference is reduced, so that better delivery time prediction can be made and it becomes convenient for the customer to expect an accurate delivery time.*
- *The osrm distance and actual distance covered are also not same i.e. maybe the delivery person is not following the predefined route which may lead to late deliveries or the osrm devices is not properly predicting the route based on distance, traffic and other factors. Team needs to look into it.*
- *Most of the orders are coming from/reaching to states like Maharashtra, Karnataka, Haryana and Tamil Nadu. The existing corridors can be further enhanced to improve the penetration in these areas.*
- *Customer profiling of the customers belonging to the states Maharashtra, Karnataka, Haryana, Tamil Nadu and Uttar Pradesh has to be done to get to know why major orders are coming from these atates and to improve customers' buying and delivery experience.*
- *From state point of view, we might have very heavy traffic in certain states and bad terrain conditions in certain states. This will be a good indicator to plan and cater to demand during peak festival seasons.*

In [ ]: