About:

One of the largest and fastest-growing fully integrated logistic player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

Problem Statement:

The company wants to understand and process the data coming out of data engineering pipelines: • Clean, sanitize and manipulate data to get useful features out of raw fields • Make sense out of the raw data and help the data science team to build forecasting models on it

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as sci
```

In [3]:
```
df = pd.read_csv('delhivery_data.csv')
df.head()
```

Out[3]:

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

5 rows × 24 columns

In [4]:

```
#Shape of the dataset
df.shape
```

Out[4]: (144867, 24)

In [5]:

```
#check basic structure of dataset
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 [6]:
```
#Brief statistical summary of numerical columns
df.describe()
```

Out[6]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_ti
count	144867.000000	144867.000000	144867.000000	144867.0000
mean	961.262986	232.926567	234.073372	416.9275
std	1037.012769	344.755577	344.990009	598.1036
min	20.000000	9.000000	9.000045	9.0000
25%	161.000000	22.000000	23.355874	51.0000
50%	449.000000	66.000000	66.126571	132.0000
75%	1634.000000	286.000000	286.708875	513.0000
max	7898.000000	1927.000000	1927.447705	4532.0000

In [7]:
```
#Check for null columns
df.isna().sum()
```

Out[7]:
```
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 [8]:
```
#Null Values in percentage terms
(df.isna().sum()/df.shape[0]) *100
```

Out[8]:
```
data                              0.000000
trip_creation_time                0.000000
route_schedule_uuid               0.000000
route_type                        0.000000
trip_uuid                         0.000000
source_center                     0.000000
source_name                       0.202254
destination_center                0.000000
destination_name                  0.180165
od_start_time                     0.000000
od_end_time                       0.000000
start_scan_to_end_scan            0.000000
is_cutoff                         0.000000
cutoff_factor                     0.000000
cutoff_timestamp                  0.000000
actual_distance_to_destination    0.000000
actual_time                       0.000000
osrm_time                         0.000000
osrm_distance                     0.000000
factor                            0.000000
segment_actual_time               0.000000
segment_osrm_time                 0.000000
segment_osrm_distance             0.000000
segment_factor                    0.000000
dtype: float64
```

In [9]:
```
#unique values in each column
df.nunique()
```

Out[9]:
```
data                                   2
trip_creation_time                 14817
route_schedule_uuid                 1504
route_type                             2
trip_uuid                          14817
source_center                       1508
source_name                         1498
destination_center                  1481
destination_name                    1468
od_start_time                      26369
od_end_time                        26369
start_scan_to_end_scan              1915
is_cutoff                              2
cutoff_factor                        501
cutoff_timestamp                   93180
actual_distance_to_destination    144515
actual_time                         3182
osrm_time                           1531
osrm_distance                     138046
factor                             45641
segment_actual_time                  747
segment_osrm_time                    214
segment_osrm_distance             113799
segment_factor                      5675
dtype: int64
```

convert the datatype of the columns to category where number of unique data is 2

In [10]:
```
df['data'] = df['data'].astype('category')
df['route_type'] = df['route_type'].astype('category')
df['is_cutoff '] = df['is_cutoff'].astype('category')
```

Updating the datatype of the datetime columns

In [11]:
```
datetime_cols = ['trip_creation_time', 'od_start_time', 'od_end_time']
for i in datetime_cols:
    df[i] = pd.to_datetime(df[i])
```

In [13]:
```
#check for overall structure after the changes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 25 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  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
 24  is_cutoff                       144867 non-null  category
dtypes: bool(1), category(3), datetime64[ns](3), float64(10), int64(1), object(7)
memory usage: 23.8+ MB
```

In [14]:
```
#checks for source name, if null returns the source center
missing_source_name = df.loc[df['source_name'].isnull(), 'source_center'].unique
missing_source_name
```

Out[14]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [15]:
```
#checks for destination name, if null returns the destination center
missing_destination_name = df.loc[df['destination_name'].isnull(), 'destination_
missing_destination_name
```

Out[15]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

Handling missing destination names and source names

In [16]:
```
count = 1
for i in missing_destination_name:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.loc[df['desti
    count += 1
```

In [17]:
```
#This dictionary will be used to store unique 'destination_name' values for each
d = {}
```

```python
#Stores d with unique 'destination_name' values for each 'destination_center' in
for i in missing_source_name:
    d[i] = df.loc[df['destination_center'] == i, 'destination_name'].unique()

# Check if the list of unique values is empty for a 'destination_center'
for key, val in d.items():
    if len(val) == 0:
        d[key] = [f'location_{count}']
        count += 1

# Initialize a new dictionary d2 and map 'destination_center' to a single value
d2 = {}
for key, val in d.items():
    d2[key] = val[0]

# print the 'destination_center' and its corresponding key value.
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 [18]:
```python
# This replaces missing values (np.nan) in the selected 'source_name' column wit
# corresponding value from the d2 dictionary for the current 'source_center' val

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

In [19]:
```python
#check for null values again after changes
df.isna().sum()
```

Out[19]:
```
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
is_cutoff                         0
dtype: int64
```

In [20]:
```
df.describe()
```

Out[20]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan
count	144867	144867	144867	144867.000000
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.262986
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.000000
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000
std	NaN	NaN	NaN	1037.012769

In [21]:
```
df.describe(include = 'object')
```

Out[21]:

	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_HI (Haryana
freq	1812	101	23347	2334

Merging rows

In [22]:
```
grouping_1 = ['trip_uuid', 'source_center', 'destination_center']
df1 = df.groupby(by = grouping_1, as_index = False).agg({'data' : 'first',
                                                          'route_type' : 'first',
                                                        'trip_creation_time' : 'f
                                                        'source_name' : 'first',
                                                        'destination_name' : 'las
                                                        'od_start_time' : 'first'
                                                        'od_end_time' : 'first',
                                                        'start_scan_to_end_scan'
                                                        'actual_distance_to_desti
                                                        'actual_time' : 'last',
                                                        'osrm_time' : 'last',
                                                        'osrm_distance' : 'last',
                                                        'segment_actual_time' : '
                                                        'segment_osrm_time' : 'su
                                                        'segment_osrm_distance' :
df1
```

Out[22]:

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

26368 rows × 18 columns

In [23]:

```
### Calculate the time taken between od_start_time and od_end_time

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.total_secon
df1['od_total_time'].head()
```

Out[23]:

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

In [24]:

```
# merging and aggregration on df1 using groupby
df2 = df1.groupby(by = 'trip_uuid', as_index = False).agg({'source_center' : 'fi
                                                          'destination_center'
                                                          'data' : 'first',
                                                          'route_type' : 'first
                                                          'trip_creation_time'
                                                          'source_name' : 'firs
                                                          'destination_name' :
                                                          'od_total_time' : 'su
                                                          'start_scan_to_end_sc
```

```
                                                             'actual_distance_to_d
                                                             'actual_time' : 'sum'
                                                             'osrm_time' : 'sum',
                                                             'osrm_distance' : 'su
                                                             'segment_actual_time'
                                                             'segment_osrm_time' :
                                                             'segment_osrm_distanc
df2
```

Out[24]:

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

14817 rows × 17 columns

In [25]:

```python
## Source Name: Split and extract features out of destination. City-place-code (
def extract_state(state):
    e = state.split('(')
    if len(e) == 1:
        return e[0]
    else:
        return e[1].replace(')', "")
```

In [26]:

```python
def extract_city(city):
    if 'location' in city:
        return 'unknown_city'
    else:
        e = city.split()[0].split('_')
        if 'CCU' in city:
            return 'Kolkata'
```

```python
        elif 'MAA' in city.upper():
            return 'Chennai'
        elif ('HBR' in city.upper()) or ('BLR' in city.upper()):
            return 'Bengaluru'
        elif 'FBD' in city.upper():
            return 'Faridabad'
        elif 'BOM' in city.upper():
            return 'Mumbai'
        elif 'DEL' in city.upper():
            return 'Delhi'
        elif 'OK' in city.upper():
            return 'Delhi'
        elif 'GZB' in city.upper():
            return 'Ghaziabad'
        elif 'GGN' in city.upper():
            return 'Gurgaon'
        elif 'AMD' in city.upper():
            return 'Ahmedabad'
        elif 'CJB' in city.upper():
            return 'Coimbatore'
        elif 'HYD' in city.upper():
            return 'Hyderabad'
        return e[0]
```

In [27]:
```python
def extract_place(place):
    if 'location' in place:
        return place
    elif 'HBR' in place:
        return 'HBR Layout PC'
    else:
        e = place.split()[0].split('_', 1)
        if len(e) == 1:
            return 'unknown_place'
        else:
            return e[1]
```

In [28]:
```python
df2['source_state'] = df2['source_name'].apply(extract_state)
df2['source_state'].unique()
```

Out[28]:
```
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', 'Nagaland',
       'location_9', 'location_3', 'location_2', 'location_14',
       'location_7'], dtype=object)
```

In [29]:
```python
df2['source_city'] = df2['source_name'].apply(extract_city)
df2['source_city'].unique()[:20]
```

Out[29]:
```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Bellary', 'Chennai',
       'Bengaluru', 'Surat', 'Delhi', 'Pune', 'Faridabad', 'Shirala',
       'Hyderabad', 'Thirumalagiri', 'Gulbarga', 'Jaipur', 'Allahabad',
       'Guwahati', 'Narsinghpur', 'Shrirampur'], dtype=object)
```

In [30]:
```python
df2['source_place'] = df2['source_name'].apply(extract_place)
df2['source_place'].unique()[:20]
```

Out[30]:
```
array(['Central_H_6', 'ChikaDPP_D', 'Bilaspur_HB', 'unknown_place', 'Dc',
       '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'], dtype=object)
```

In [31]:
```
##Destination Name: Split and extract features out of destination. City-place-co

df2['destination_state'] = df2['destination_name'].apply(extract_state)
df2['destination_state'].head()
```

Out[31]:
```
0    Uttar Pradesh
1        Karnataka
2          Haryana
3      Maharashtra
4        Karnataka
Name: destination_state, dtype: object
```

In [32]:
```
df2['destination_city'] = df2['destination_name'].apply(extract_city)
df2['destination_city'].head()
```

Out[32]:
```
0        Kanpur
1    Doddablpur
2       Gurgaon
3        Mumbai
4        Sandur
Name: destination_city, dtype: object
```

In [33]:
```
df2['destination_place'] = df2['destination_name'].apply(extract_place)
df2['destination_place'].head()
```

Out[33]:
```
0    Central_H_6
1     ChikaDPP_D
2    Bilaspur_HB
3      MiraRd_IP
4     WrdN1DPP_D
Name: destination_place, dtype: object
```

In [34]:
```
##Extract month, year, day, week, hour from Trip_creation_time

df2['trip_creation_date'] = pd.to_datetime(df2['trip_creation_time'].dt.date)
df2['trip_creation_date'].head()
```

Out[34]:
```
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 [35]:
```
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[35]:
```
0    12
1    12
2    12
3    12
4    12
Name: trip_creation_day, dtype: int8
```

In [36]:
```python
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[36]:
```
0    9
1    9
2    9
3    9
4    9
Name: trip_creation_month, dtype: int8
```

In [37]:
```python
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[37]:
```
0    2018
1    2018
2    2018
3    2018
4    2018
Name: trip_creation_year, dtype: int16
```

In [38]:
```python
df2['trip_creation_week'] = df2['trip_creation_time'].dt.isocalendar().week
df2['trip_creation_week'] = df2['trip_creation_week'].astype('int8')
df2['trip_creation_week'].head()
```

Out[38]:
```
0    37
1    37
2    37
3    37
4    37
Name: trip_creation_week, dtype: int8
```

In [39]:
```python
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[39]:
```
0    0
1    0
2    0
3    0
4    0
Name: trip_creation_hour, dtype: int8
```

In [40]:
```python
# structure of dataset after data cleaning
df2.shape
```

Out[40]:
```
(14817, 29)
```

In [41]:
```python
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  float64
 11  actual_time                     14817 non-null  float64
 12  osrm_time                       14817 non-null  float64
 13  osrm_distance                   14817 non-null  float64
 14  segment_actual_time             14817 non-null  float64
 15  segment_osrm_time               14817 non-null  float64
 16  segment_osrm_distance           14817 non-null  float64
 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), float64(9), int16(1), int8(4), object(11)
memory usage: 2.6+ MB
```

In [42]:
```python
df2.head()
```

Out[42]:

	trip_uuid	source_center	destination_center	data	route_type	trip_cre
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	00:0
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	00:0
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	00:0
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	00:1
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	00:2

5 rows × 29 columns

In [43]:
```
df2.describe()
```

Out[43]:

	trip_creation_time	od_total_time	start_scan_to_end_scan	actual_distance_to_desti
count	14817	14817.000000	14817.000000	14817.(
mean	2018-09-22 12:44:19.555167744	531.697630	530.810016	164.4
min	2018-09-12 00:00:16.535741	23.460000	23.000000	9.(
25%	2018-09-17 02:51:25.129125888	149.930000	149.000000	22.8
50%	2018-09-22 04:02:35.066945024	280.770000	280.000000	48.4
75%	2018-09-27 19:37:41.898427904	638.200000	637.000000	164.5
max	2018-10-03 23:59:42.701692	7898.550000	7898.000000	2186.5
std	NaN	658.868223	658.705957	305.3

In [119…
```
# statistical summary of all object dtype

df2.describe(include = object).T
```

Out[119]:

	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

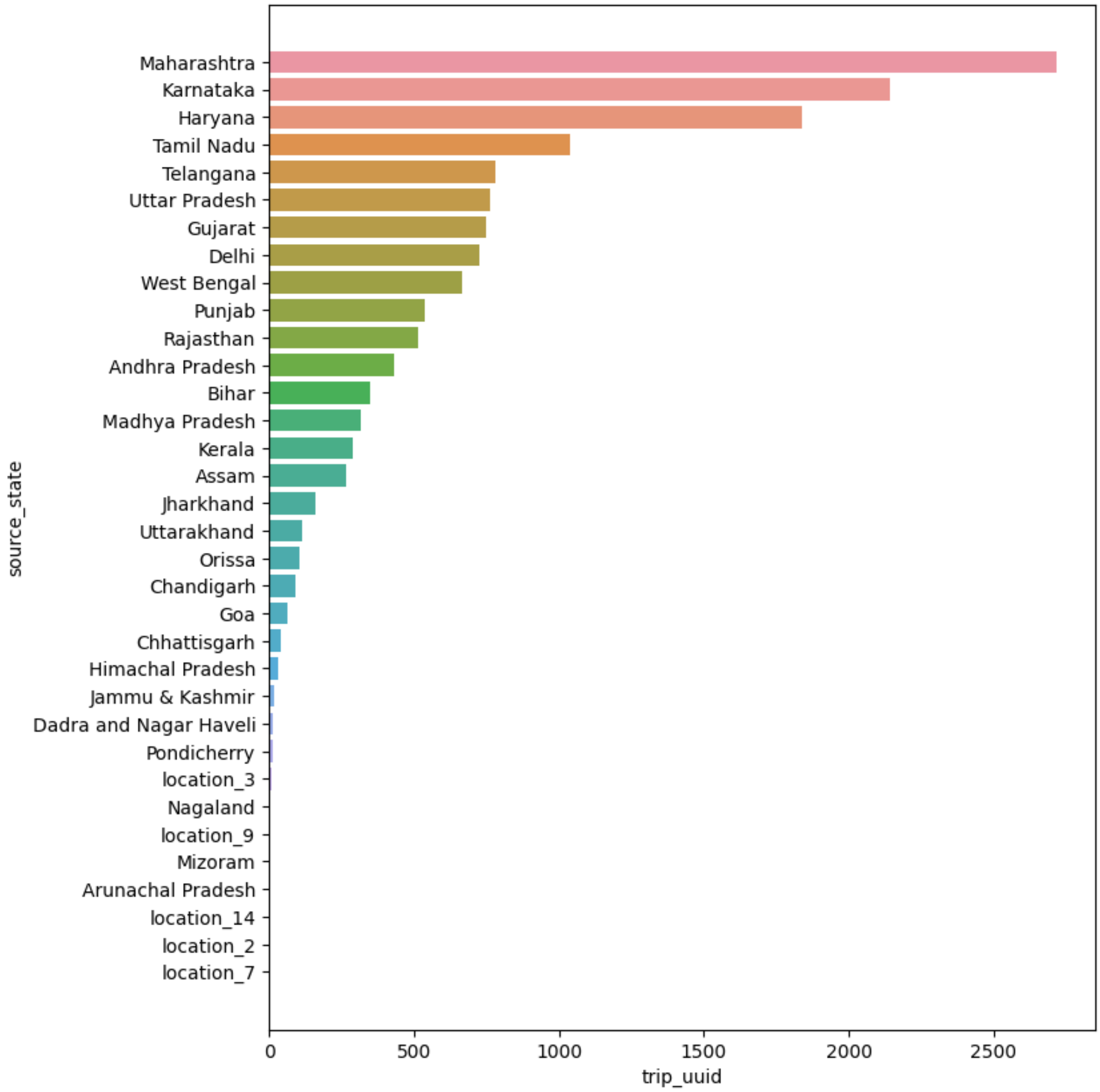
In [44]:
```
# check from where most orders are coming from

df_source_state = df2.groupby(by = 'source_state')['trip_uuid'].count().to_frame
df_source_state['perc'] = np.round(df_source_state['trip_uuid'] * 100/ df_source
df_source_state = df_source_state.sort_values(by = 'trip_uuid', ascending = Fals
df_source_state.head()
```

Out[44]:

	source_state	trip_uuid	perc
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 [45]:
```
plt.figure(figsize = (8, 10))
sns.barplot(data = df_source_state,
            x = df_source_state['trip_uuid'],
            y = df_source_state['source_state'])
plt.plot()
```

Out[45]: []



In [46]:
```
#  based on the number of trips ended in different cities

df_destination_city = df2.groupby(by = 'destination_city')['trip_uuid'].count().
df_destination_city['perc'] = np.round(df_destination_city['trip_uuid'] * 100/ d
```

```python
df_destination_city = df_destination_city.sort_values(by = 'trip_uuid', ascendin
df_destination_city
```

Out[46]:

	destination_city	trip_uuid	perc
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

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

Set up Null Hypothesis

Null Hypothesis (H0) - od_total_time and start_scan_to_end_scan are same. Alternate Hypothesis (HA) - od_total_time and start_scan_to_end_scan are different.

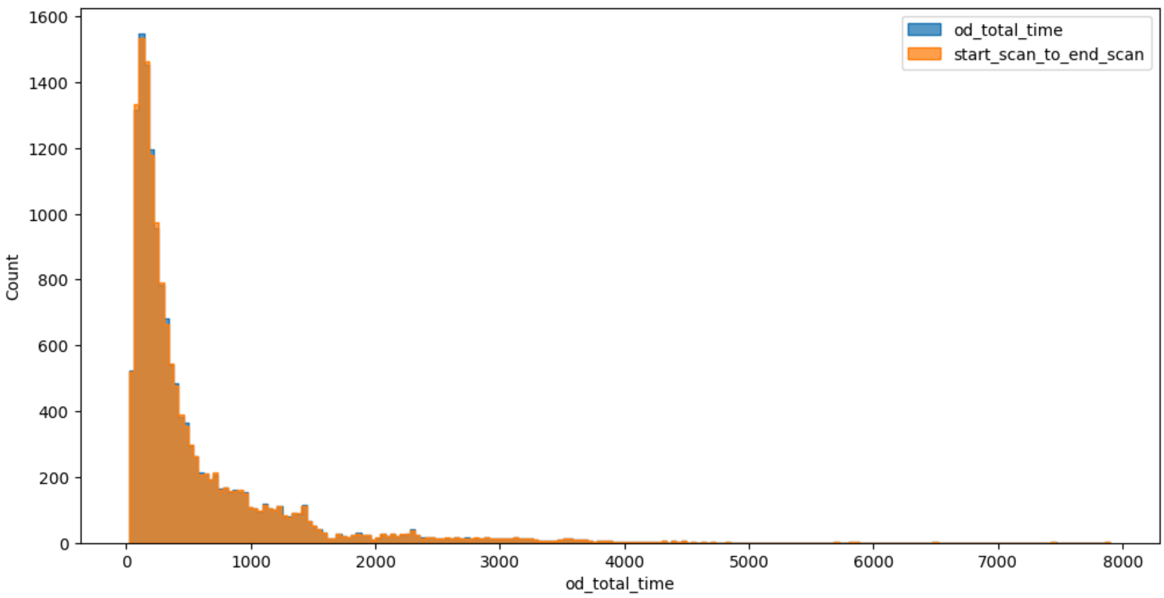
In [48]:
```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```

Out[48]:

	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 [49]:
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['od_total_time'], element = 'step')
sns.histplot(df2['start_scan_to_end_scan'], element = 'step')
plt.legend(['od_total_time', 'start_scan_to_end_scan'])
plt.plot()
```

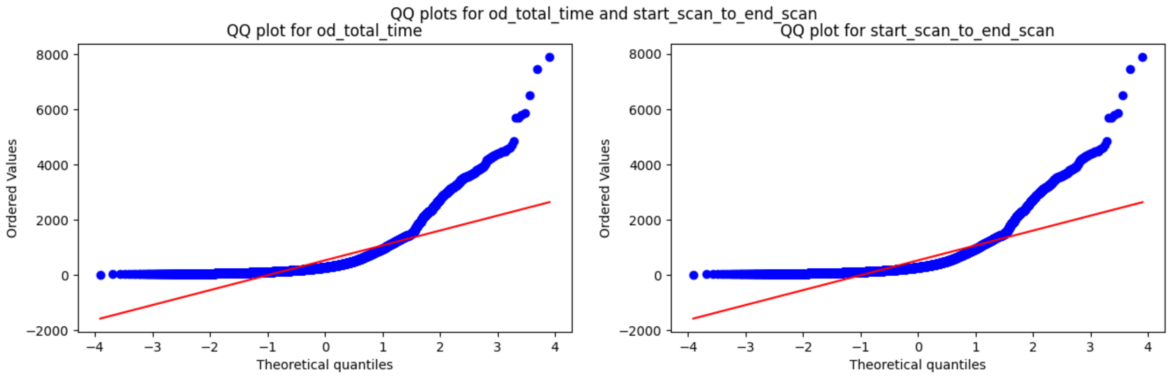
Out[49]: []



In [50]:
```
#  check for normal distribution using QQ Plot

plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
```

```
plt.suptitle('QQ plots for od_total_time and start_scan_to_end_scan')
sci.probplot(df2['od_total_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for od_total_time')
plt.subplot(1, 2, 2)
sci.probplot(df2['start_scan_to_end_scan'], plot = plt, dist = 'norm')
plt.title('QQ plot for start_scan_to_end_scan')
plt.plot()
```

Out[50]: []



In [53]:
```
# It can be seen from the above plots that the samples follow normal distributio
# since the plot is not normally distributed ANOVA cannot be performed hence app
# Ho : The sample follows normal distribution
# Ha : The sample does not follow normal distribution
# alpha = 0.05

test_stat, p_value = sci.shapiro(df2['od_total_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [54]:
```
test_stat, p_value = sci.shapiro(df2['start_scan_to_end_scan'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [55]:
```
# Null Hypothesis(H0) - Variances are equal
# Alternate Hypothesis(HA) - Variances are not equal
# alpha = 0.05

test_stat, p_value = sci.levene(df2['od_total_time'], df2['start_scan_to_end_sca
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Variances are not equal')
else:
    print('Fail to reject null hypothesis. Variances are equal')
```

```
p-value 0.9668007217581142
Fail to reject null hypothesis. Variances are equal
```

In [56]:
```
# Since the samples do not follow any of the assumptions, T-Test cannot be appli
# We can perform its non parametric equivalent test i.e., Mann-Whitney U rank te

test_stat, p_value = sci.mannwhitneyu(df2['od_total_time'], df2['start_scan_to_e
print('P-value :',p_value)
```

```
P-value : 0.7815123224221716
```

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)

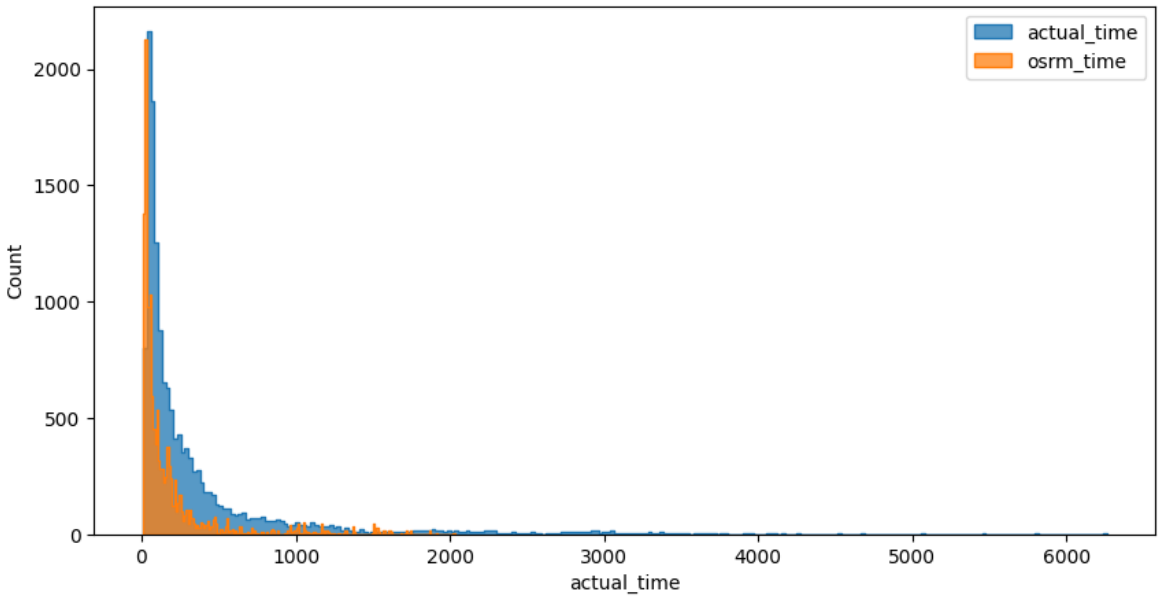
In [57]:
```
df2[['actual_time', 'osrm_time']].describe()
```

Out[57]:

	actual_time	osrm_time
count	14817.000000	14817.000000
mean	357.143754	161.384018
std	561.396157	271.360995
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

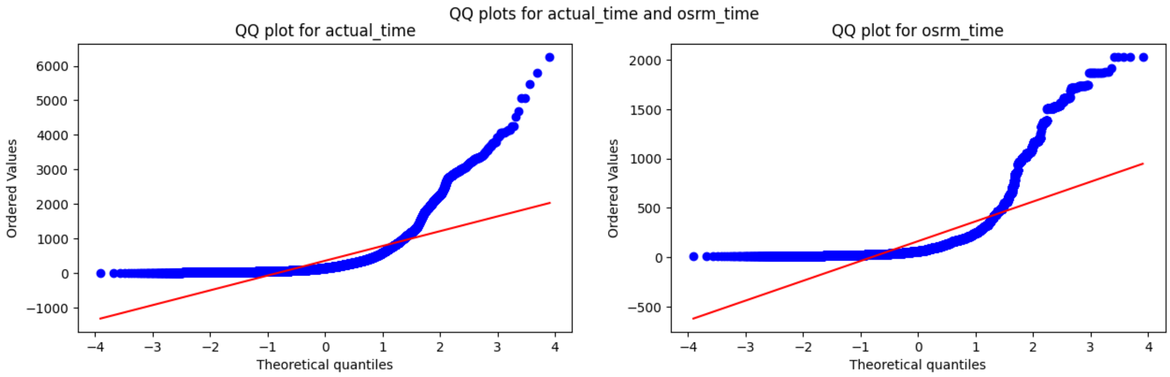
In [58]:
```
plt.figure(figsize = (10, 5))
sns.histplot(df2['actual_time'], element = 'step')
sns.histplot(df2['osrm_time'], element = 'step')
plt.legend(['actual_time', 'osrm_time'])
plt.plot()
```

Out[58]: []



In [59]:
```python
#  check for normal distribution using QQ Plot

plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and osrm_time')
sci.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
sci.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.plot()
```

Out[59]: []



In [60]:
```python
# It can be seen from the above plots that the samples follow normal distributio
# Applying Shapiro-Wilk test for normality
# Ho : The sample follows normal distribution
# Ha : The sample does not follow normal distribution
# alpha = 0.05

test_stat, p_value = sci.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [61]:
```python
test_stat, p_value = sci.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [62]:
```python
# Null Hypothesis(H0) - Variances are equal
# Alternate Hypothesis(HA) - Variances are not equal
# alpha = 0.05

test_stat, p_value = sci.levene(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
```

```python
    print('Reject Null Hypothesis. Variances are not equal')
else:
    print('Fail to reject null hypothesis. Variances are equal')
```

```
p-value 1.871297993683208e-220
Reject Null Hypothesis. Variances are not equal
```

In [63]:

```python
# Since the samples do not follow any of the assumptions, T-Test cannot be appli
# We can perform its non parametric equivalent test i.e., Mann-Whitney U rank te

test_stat, p_value = sci.mannwhitneyu(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
```

```
p-value 0.0
```

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

In [64]:

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

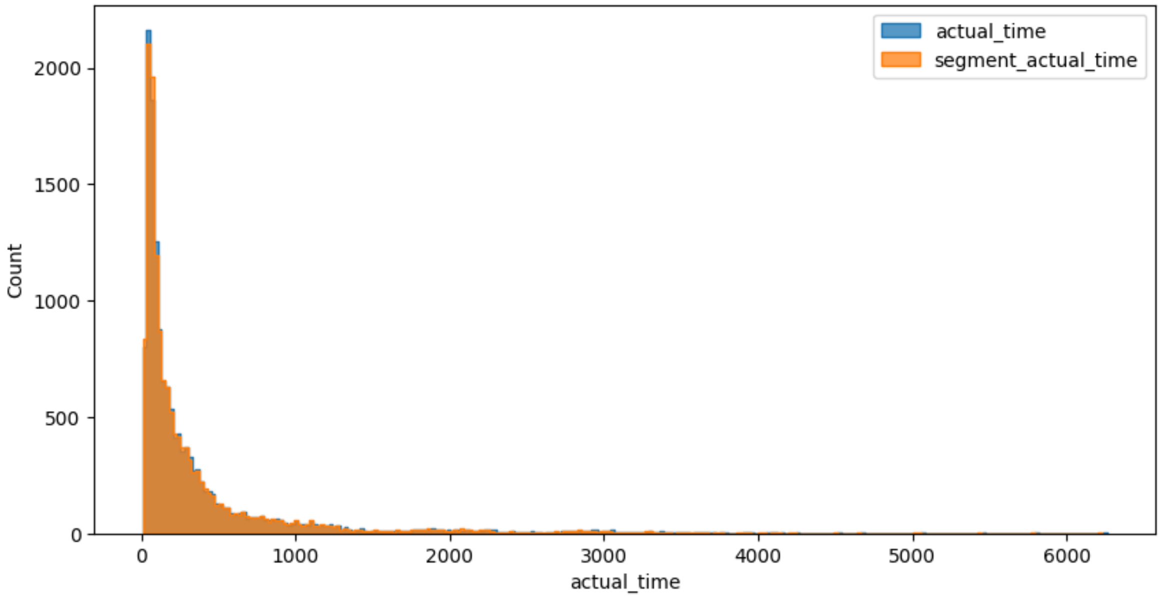
Out[64]:

	actual_time	segment_actual_time
count	14817.000000	14817.000000
mean	357.143754	353.892286
std	561.396157	556.247965
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 [65]:

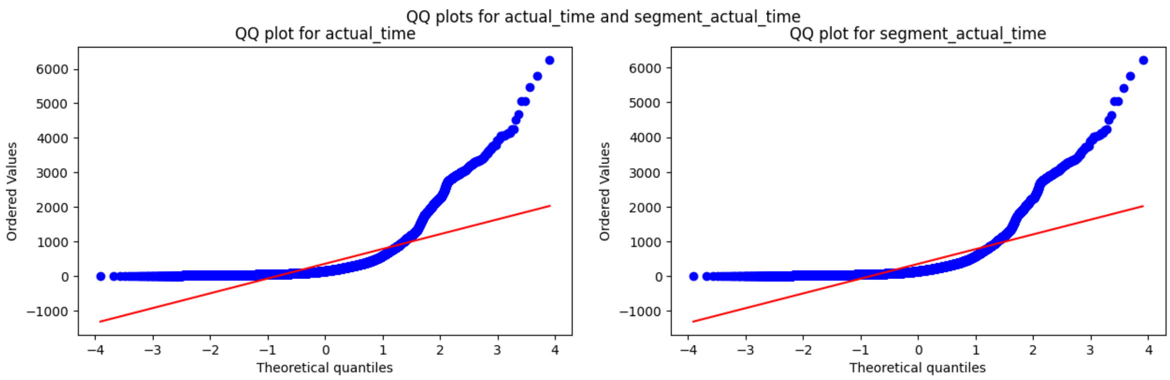
```python
plt.figure(figsize = (10, 5))
sns.histplot(df2['actual_time'], element = 'step')
sns.histplot(df2['segment_actual_time'], element = 'step')
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```

Out[65]: []



In [66]:
```
#  check for normal distribution using QQ Plot

plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and segment_actual_time')
sci.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
sci.probplot(df2['segment_actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_actual_time')
plt.plot()
```

Out[66]: []



In [67]:
```
# It can be seen from the above plots that the samples follow normal distributio
# Applying Shapiro-Wilk test for normality
# Ho : The sample follows normal distribution
# Ha : The sample does not follow normal distribution
# alpha = 0.05

test_stat, p_value = sci.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [68]:
```
test_stat, p_value = sci.shapiro(df2['segment_actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [69]:
```
# Null Hypothesis(H0) - Variances are equal
# Alternate Hypothesis(HA) - Variances are not equal
# alpha = 0.05

test_stat, p_value = sci.levene(df2['actual_time'], df2['segment_actual_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Variances are not equal')
else:
    print('Fail to reject null hypothesis. Variances are equal')
```

```
p-value 0.6955022668700895
Fail to reject null hypothesis. Variances are equal
```

In [70]:
```
# Since the samples do not follow any of the assumptions, T-Test cannot be appli
# We can perform its non parametric equivalent test i.e., Mann-Whitney U rank te

test_stat, p_value = sci.mannwhitneyu(df2['actual_time'], df2['segment_actual_ti
print('p-value', p_value)
```

```
p-value 0.4164235159622476
```

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

In [71]:
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

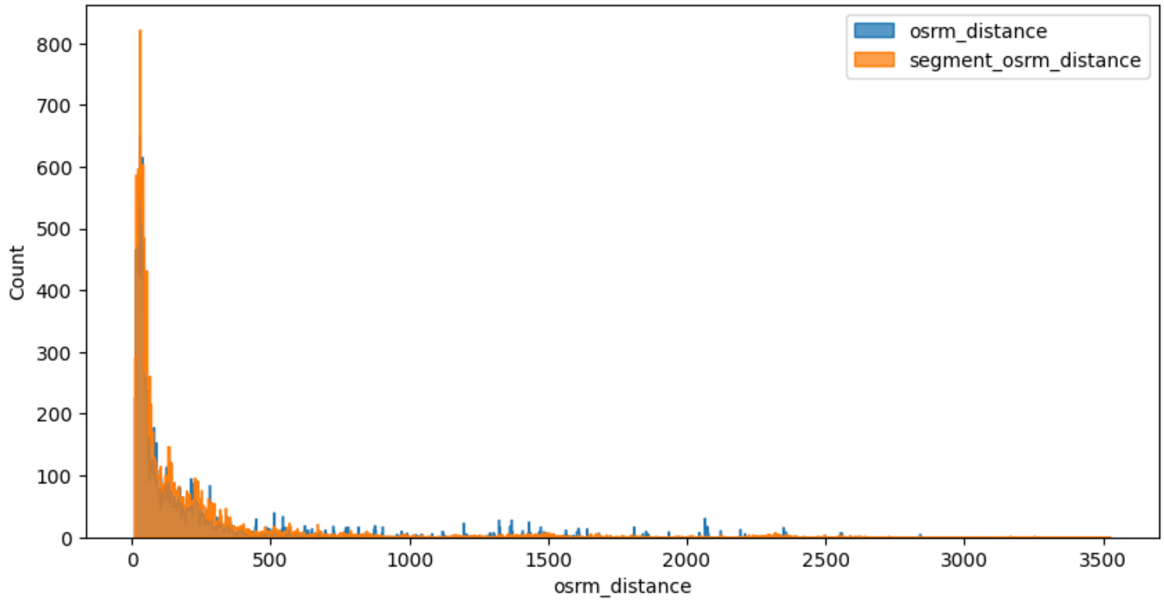
Out[71]:

	osrm_distance	segment_osrm_distance
count	14817.000000	14817.000000
mean	204.344689	223.201161
std	370.395573	416.628374
min	9.072900	9.072900
25%	30.819200	32.654500
50%	65.618800	70.154400
75%	208.475000	218.802400
max	2840.081000	3523.632400

In [72]:
```
plt.figure(figsize = (10, 5))
sns.histplot(df2['osrm_distance'], element = 'step', bins = 1000)
sns.histplot(df2['segment_osrm_distance'], element = 'step', bins = 1000)
```

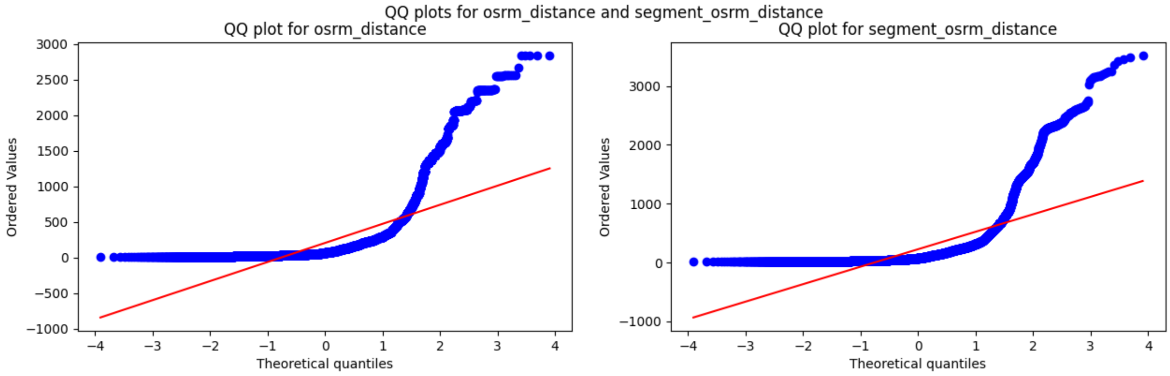
```
plt.legend(['osrm_distance', 'segment_osrm_distance'])
plt.plot()
```

Out[72]: []



In [73]:
```
#  check for normal distribution using QQ Plot

plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_distance and segment_osrm_distance')
sci.probplot(df2['osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_distance')
plt.subplot(1, 2, 2)
sci.probplot(df2['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_distance')
plt.plot()
```

Out[73]: []



In [74]:
```
# It can be seen from the above plots that the samples follow normal distributio
# Applying Shapiro-Wilk test for normality
# Ho : The sample follows normal distribution
# Ha : The sample does not follow normal distribution
# alpha = 0.05

test_stat, p_value = sci.shapiro(df2['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
```

```
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [75]:
```
test_stat, p_value = sci.shapiro(df2['segment_osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [76]:
```
# Null Hypothesis(H0) - Variances are equal
# Alternate Hypothesis(HA) - Variances are not equal
# alpha = 0.05

test_stat, p_value = sci.levene(df2['osrm_distance'], df2['segment_osrm_distance
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Variances are not equal')
else:
    print('Fail to reject null hypothesis. Variances are equal')
```

```
p-value 0.00020976354422600578
Reject Null Hypothesis. Variances are not equal
```

In [77]:
```
# Since the samples do not follow any of the assumptions, T-Test cannot be appli
# We can perform its non parametric equivalent test i.e., Mann-Whitney U rank te

test_stat, p_value = sci.mannwhitneyu(df2['osrm_distance'], df2['segment_osrm_di
print('p-value', p_value)
```

```
p-value 9.511383588276373e-07
```

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

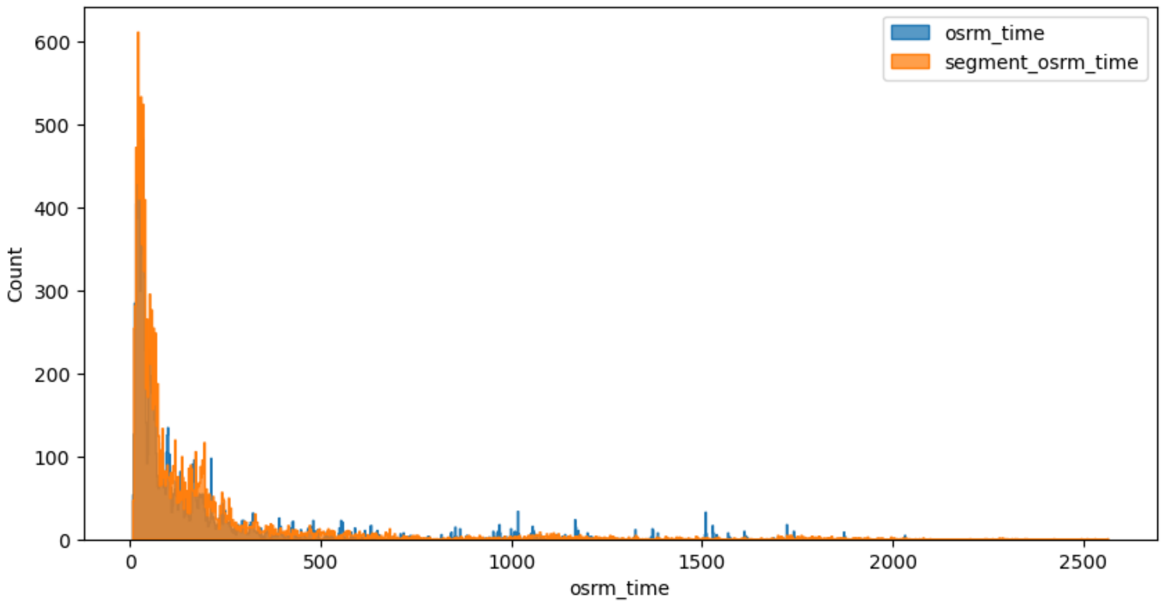
In [78]:
```
df2[['osrm_time', 'segment_osrm_time']].describe()
```

Out[78]:

	osrm_time	segment_osrm_time
count	14817.000000	14817.000000
mean	161.384018	180.949787
std	271.360995	314.542047
min	6.000000	6.000000
25%	29.000000	31.000000
50%	60.000000	65.000000
75%	168.000000	185.000000
max	2032.000000	2564.000000

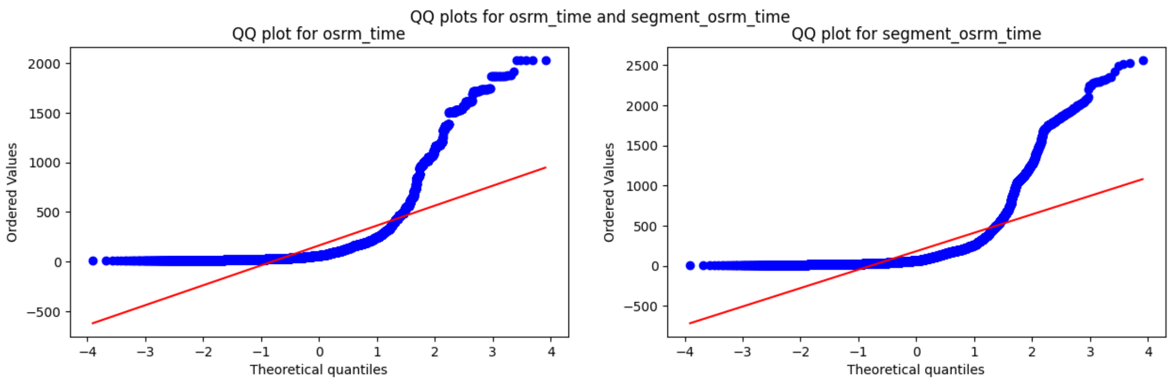
In [79]:
```
plt.figure(figsize = (10, 5))
sns.histplot(df2['osrm_time'], element = 'step', bins = 1000)
sns.histplot(df2['segment_osrm_time'], element = 'step', bins = 1000)
plt.legend(['osrm_time', 'segment_osrm_time'])
plt.plot()
```

Out[79]: []



In [80]:
```
#  check for normal distribution using QQ Plot

plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_time and segment_osrm_time')
sci.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.subplot(1, 2, 2)
sci.probplot(df2['segment_osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_time')
plt.plot()
```

Out[80]: []



In [81]:
```
# It can be seen from the above plots that the samples follow normal distributio
# Applying Shapiro-Wilk test for normality
# Ho : The sample follows normal distribution
# Ha : The sample does not follow normal distribution
# alpha = 0.05

test_stat, p_value = sci.shapiro(df2['osrm_time'].sample(5000))
```

```
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [82]:
```
test_stat, p_value = sci.shapiro(df2['segment_osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis')
else:
    print('Fail to reject null hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

In [83]:
```
# Null Hypothesis(H0) - Variances are equal
# Alternate Hypothesis(HA) - Variances are not equal
# alpha = 0.05

test_stat, p_value = sci.levene(df2['osrm_time'], df2['segment_osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('Reject Null Hypothesis. Variances are not equal')
else:
    print('Fail to reject null hypothesis. Variances are equal')
```

```
p-value 8.349482669010088e-08
Reject Null Hypothesis. Variances are not equal
```

In [84]:
```
# Since the samples do not follow any of the assumptions, T-Test cannot be appli
# We can perform its non parametric equivalent test i.e., Mann-Whitney U rank te

test_stat, p_value = sci.mannwhitneyu(df2['osrm_time'], df2['segment_osrm_time']
print('p-value', p_value)

# Since p-value < alpha therfore it can be concluded that osrm_time and segment_
```

```
p-value 2.2995370859748865e-08
```

Find outliers in the numerical variables

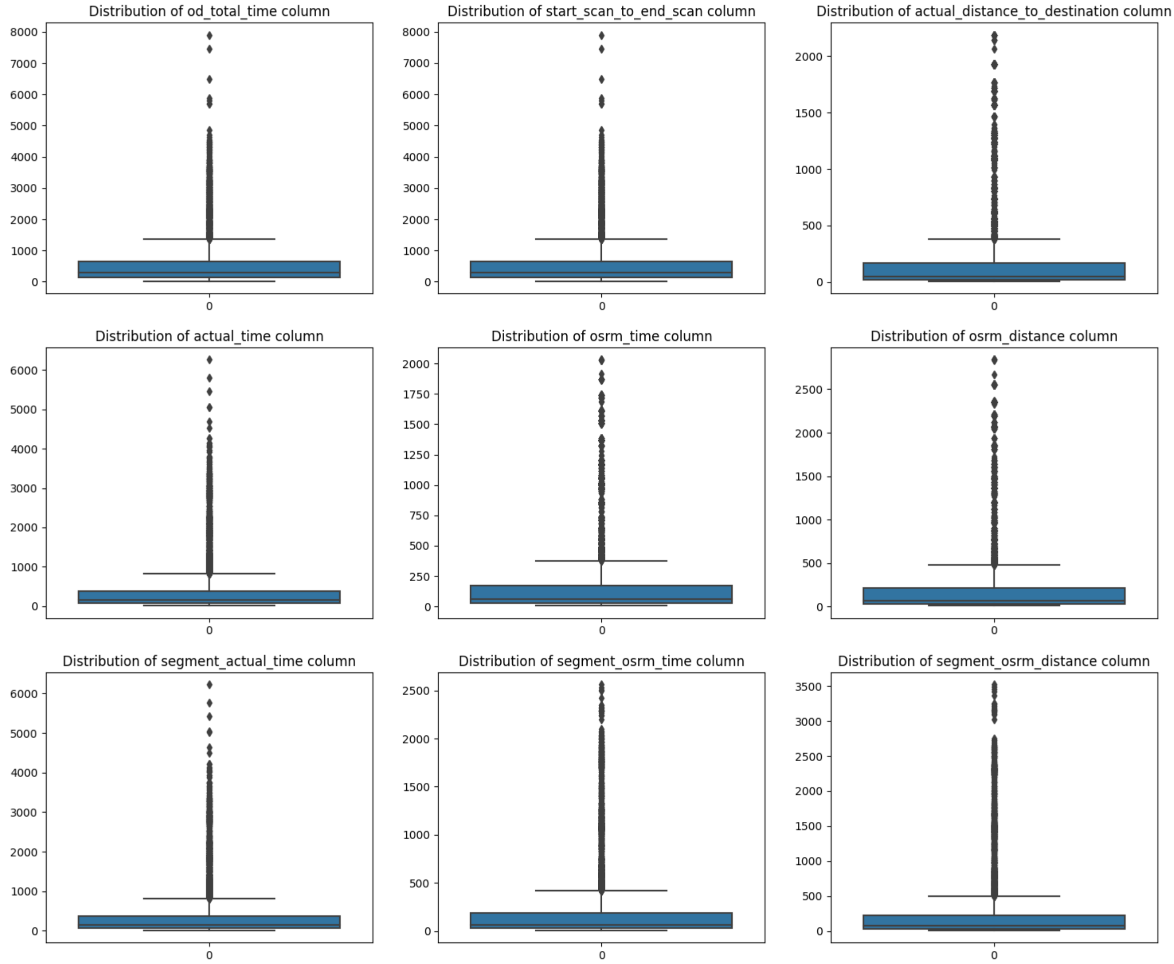
In [85]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual
                     'segment_osrm_time', 'segment_osrm_distance']
df2[numerical_columns].describe().T
```

Out[85]:

	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	164.477838	305.388147	9.002461	22.837239
actual_time	14817.0	357.143754	561.396157	9.000000	67.000000
osrm_time	14817.0	161.384018	271.360995	6.000000	29.000000
osrm_distance	14817.0	204.344689	370.395573	9.072900	30.819200
segment_actual_time	14817.0	353.892286	556.247965	9.000000	66.000000
segment_osrm_time	14817.0	180.949787	314.542047	6.000000	31.000000
segment_osrm_distance	14817.0	223.201161	416.628374	9.072900	32.654500

In [86]:

```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(df2[numerical_columns[i]])
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



In [90]:

```
# 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(i)
    print('----------------')
    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()
```

```
od_total_time
---------------
Q1 : 149.93
Q3 : 638.2
IQR : 488.27000000000004
LB : -582.4750000000001
UB : 1370.605
Number of outliers : 1266

start_scan_to_end_scan
---------------
Q1 : 149.0
Q3 : 637.0
IQR : 488.0
LB : -583.0
UB : 1369.0
Number of outliers : 1267

actual_distance_to_destination
---------------
Q1 : 22.83723905859321
Q3 : 164.58320763841138
IQR : 141.74596857981817
LB : -189.78171381113404
UB : 377.2021605081386
Number of outliers : 1449

actual_time
---------------
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
LB : -387.5
UB : 824.5
Number of outliers : 1643

osrm_time
---------------
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
LB : -179.5
UB : 376.5
Number of outliers : 1517

osrm_distance
---------------
Q1 : 30.8192
Q3 : 208.475
IQR : 177.6558
LB : -235.6645
UB : 474.9587
Number of outliers : 1524

segment_actual_time
---------------
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
LB : -385.5
```

```
UB : 818.5
Number of outliers : 1643

segment_osrm_time
---------------
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
LB : -200.0
UB : 416.0
Number of outliers : 1492

segment_osrm_distance
---------------
Q1 : 32.6545
Q3 : 218.8024
IQR : 186.1479
LB : -246.56735000000003
UB : 498.02425000000005
Number of outliers : 1548
```

one-hot encoding of categorical variables

In [91]:
```python
# value counts before one-hot encoding

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

Out[91]:
```
route_type
Carting    8908
FTL        5909
Name: count, dtype: int64
```

In [92]:
```python
# 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 [93]:
```python
# value counts after one-hot encoding

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

Out[93]:
```
route_type
0    8908
1    5909
Name: count, dtype: int64
```

In [94]:
```python
# value counts of categorical variable 'data' before one-hot encoding

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

Out[94]:
```
data
training    10654
test         4163
Name: count, dtype: int64
```

In [95]:
```python
# one-hot encoding on categorical variable 'data'
```

```
label_encoder = LabelEncoder()
df2['data'] = label_encoder.fit_transform(df2['data'])
```

In [96]:
```
#value counts after one-hot encoding

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

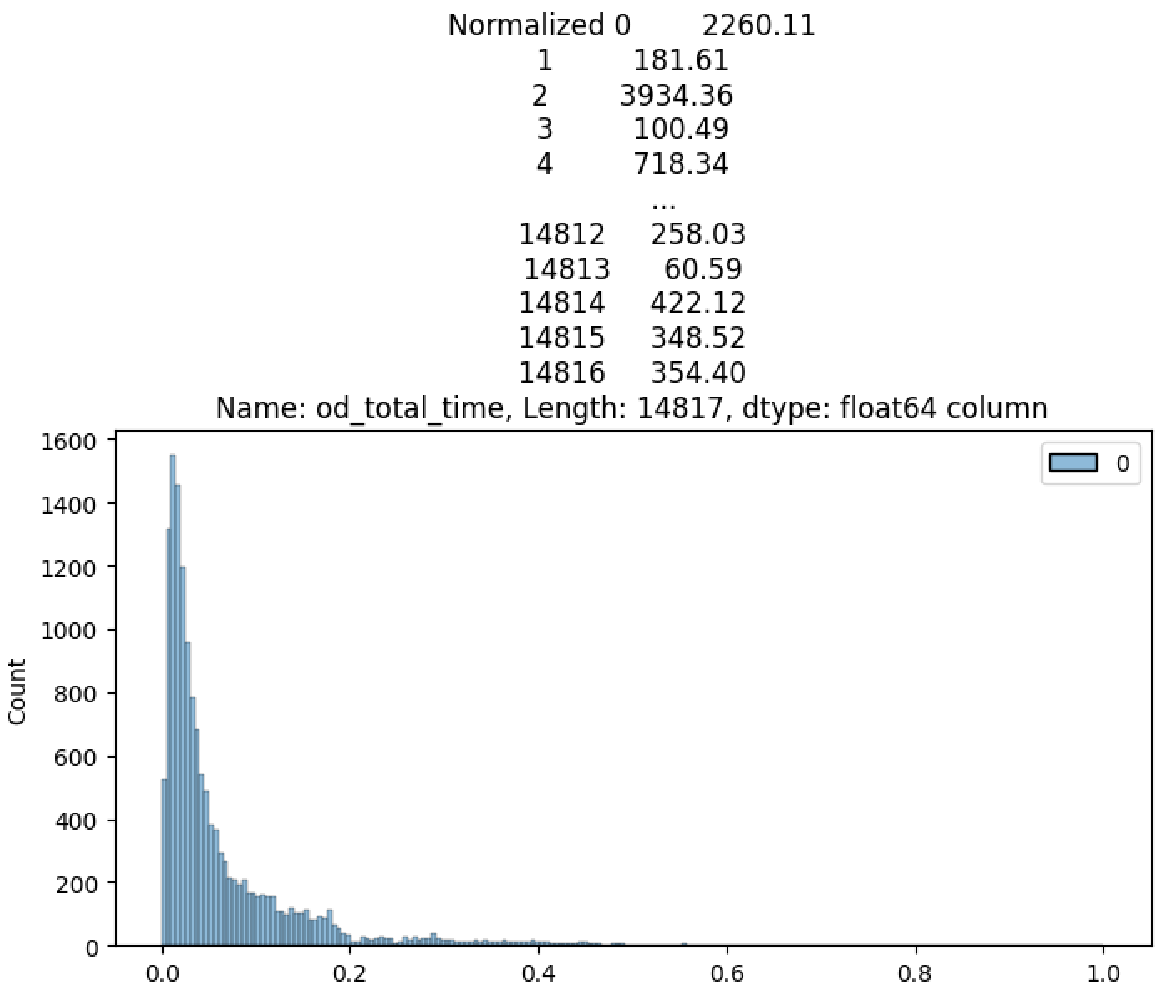
Out[96]:
```
data
1    10654
0     4163
Name: count, dtype: int64
```

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler

In [97]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [98]:
```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['od_total_time']} column")
plt.plot()
```

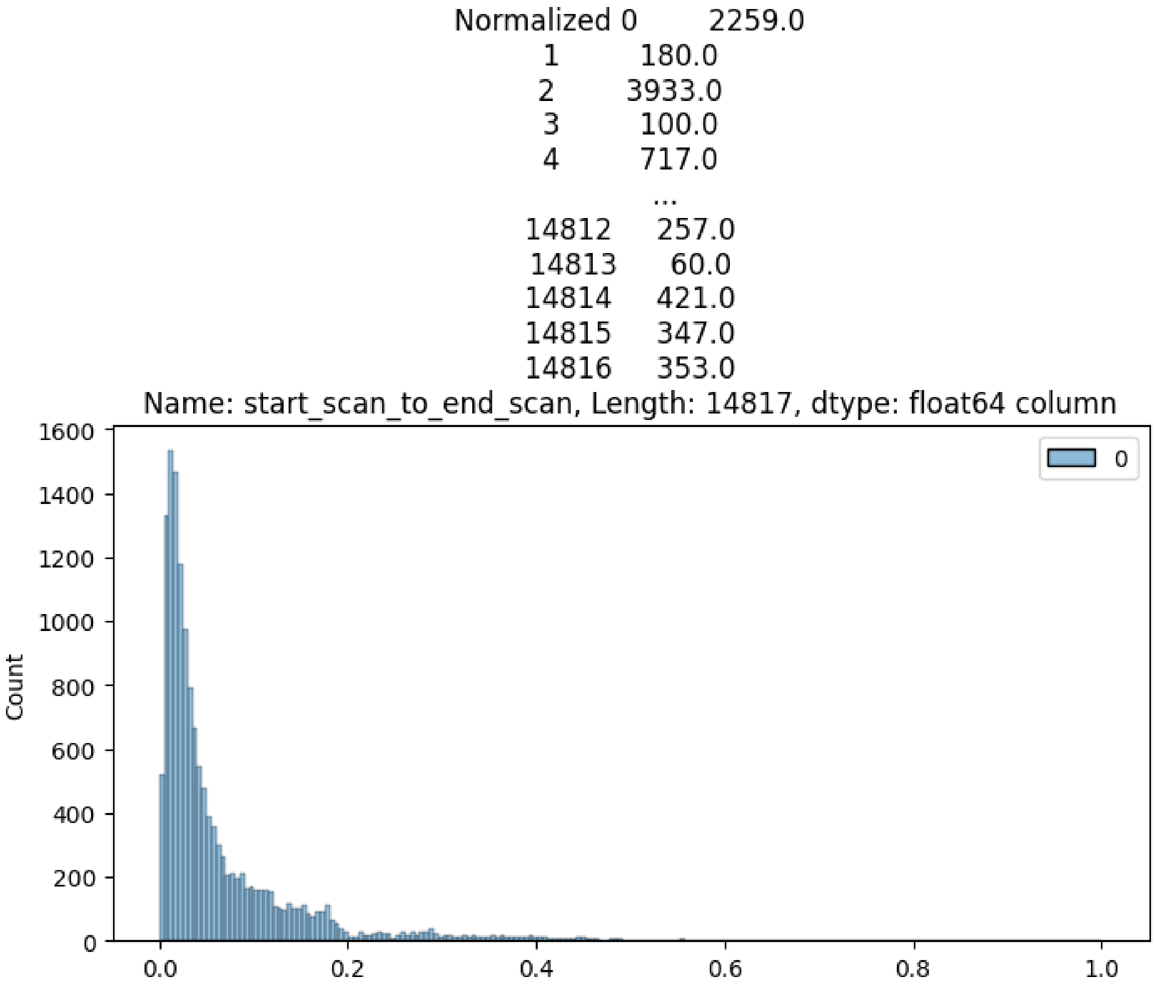
Out[98]:
```
[]
```



In [99]:
```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
```

```
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Normalized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

Out[99]: []

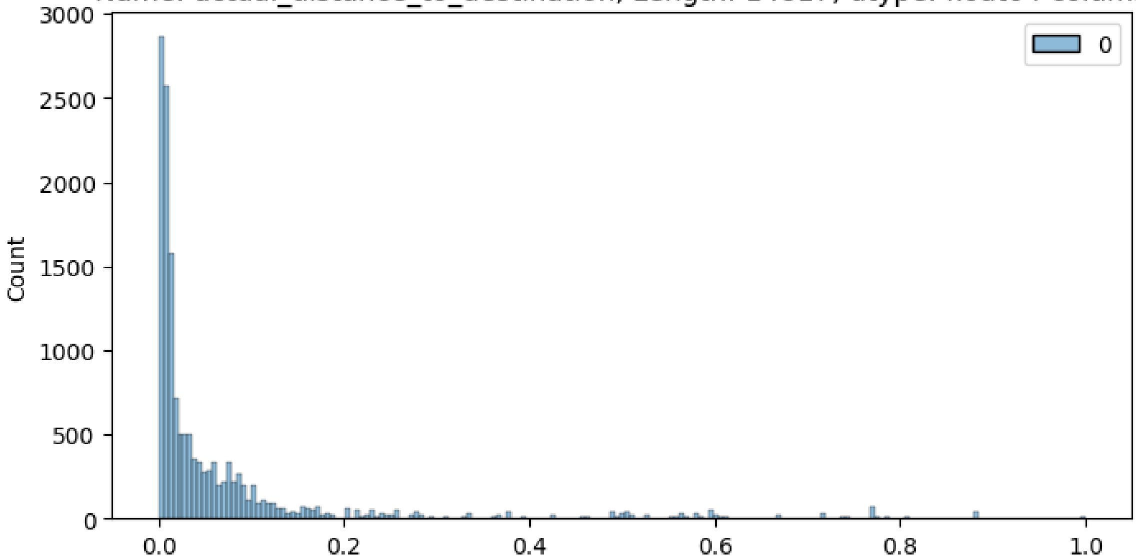


In [100…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_distance_to_destination']} column")
plt.plot()
```

Out[100]: []

```
Normalized 0        824.732854
1         73.186911
2       1927.404273
3         17.175274
4        127.448500
            ...
14812     57.762332
14813     15.513784
14814     38.684839
14815    134.723836
14816     66.081533
Name: actual_distance_to_destination, Length: 14817, dtype: float64 column
```

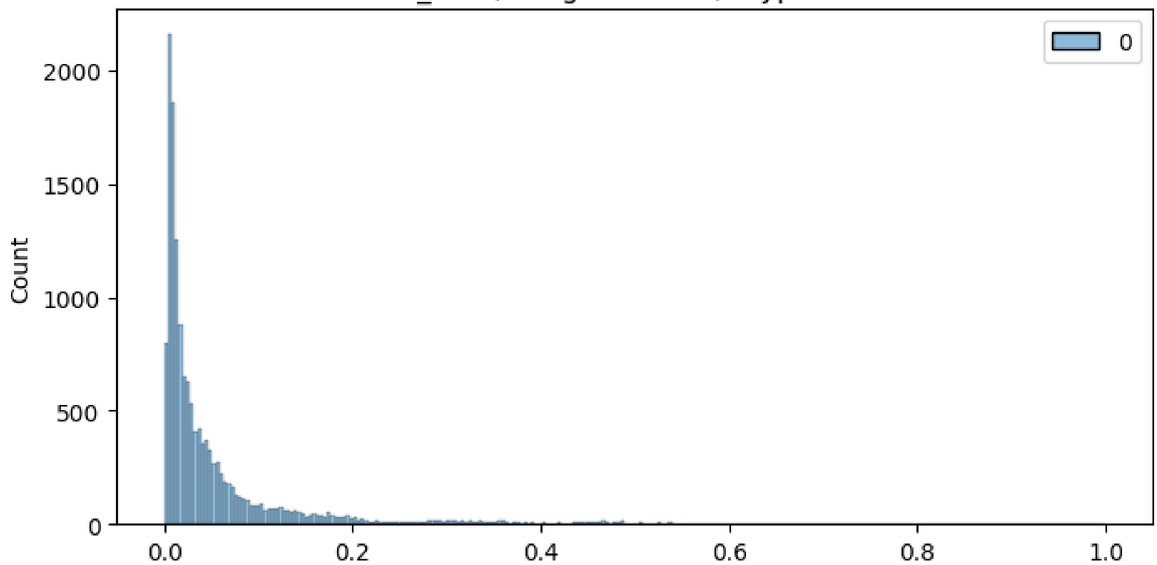


In [101…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_time']} column")
plt.plot()
```

Out[101]: []

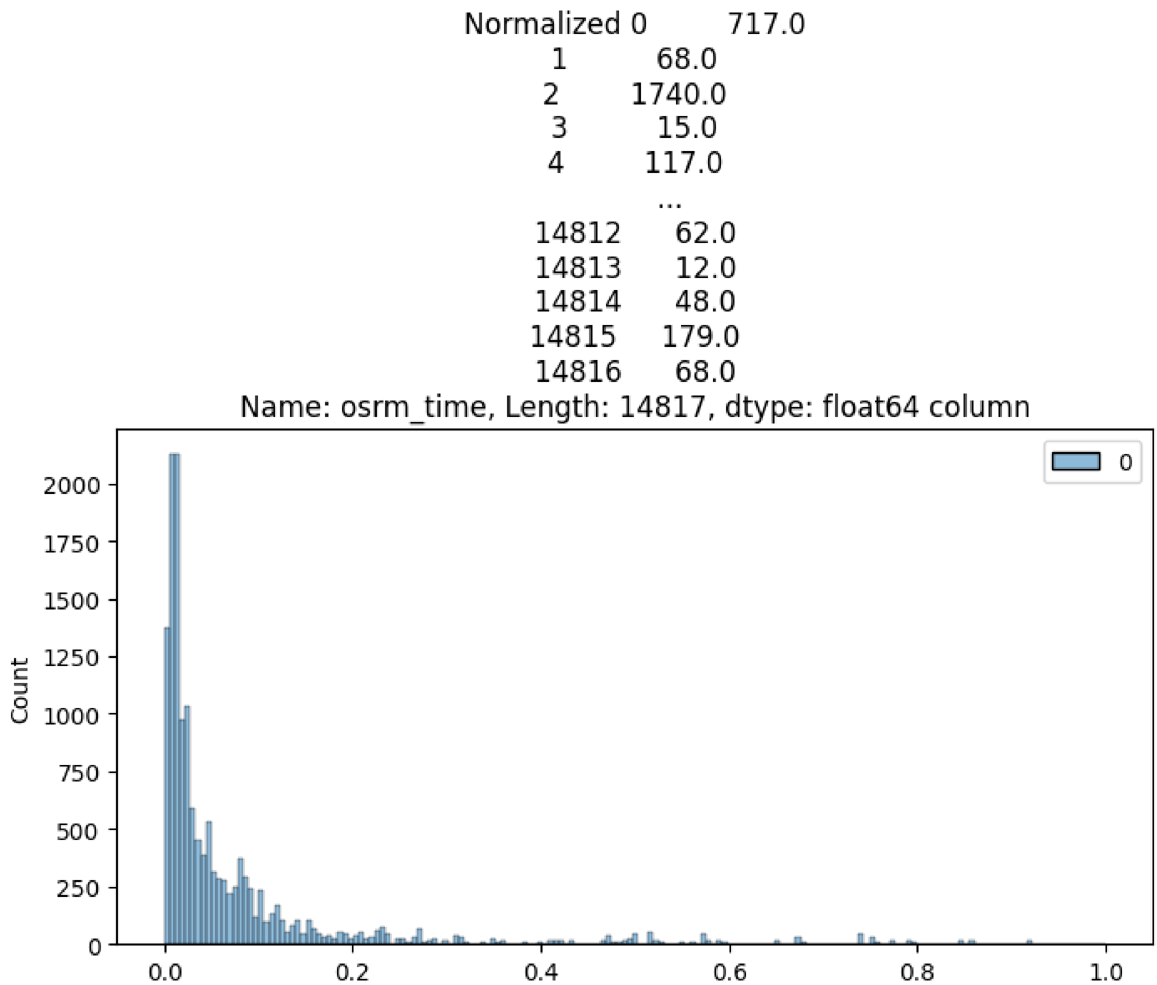
```
Normalized 0        1562.0
1         143.0
2        3347.0
3          59.0
4         341.0
          ...
14812      83.0
14813      21.0
14814     282.0
14815     264.0
14816     275.0
Name: actual_time, Length: 14817, dtype: float64 column
```



In [102…

```python
plt.figure(figsize = (8, 4))
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[102]: []

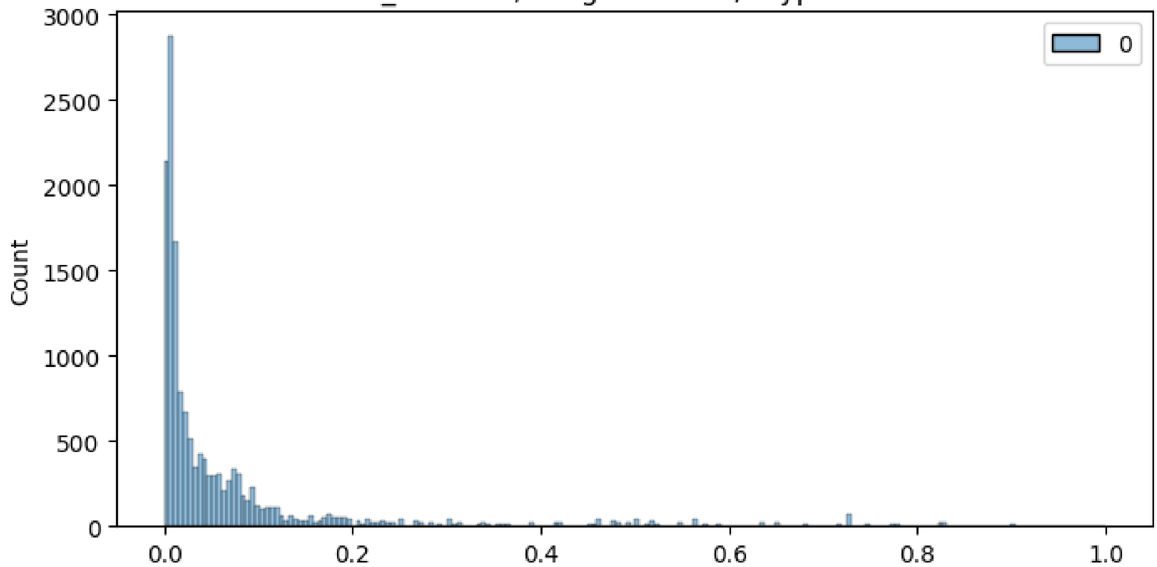


In [103…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['osrm_distance']} column")
plt.plot()
```

Out[103]: []

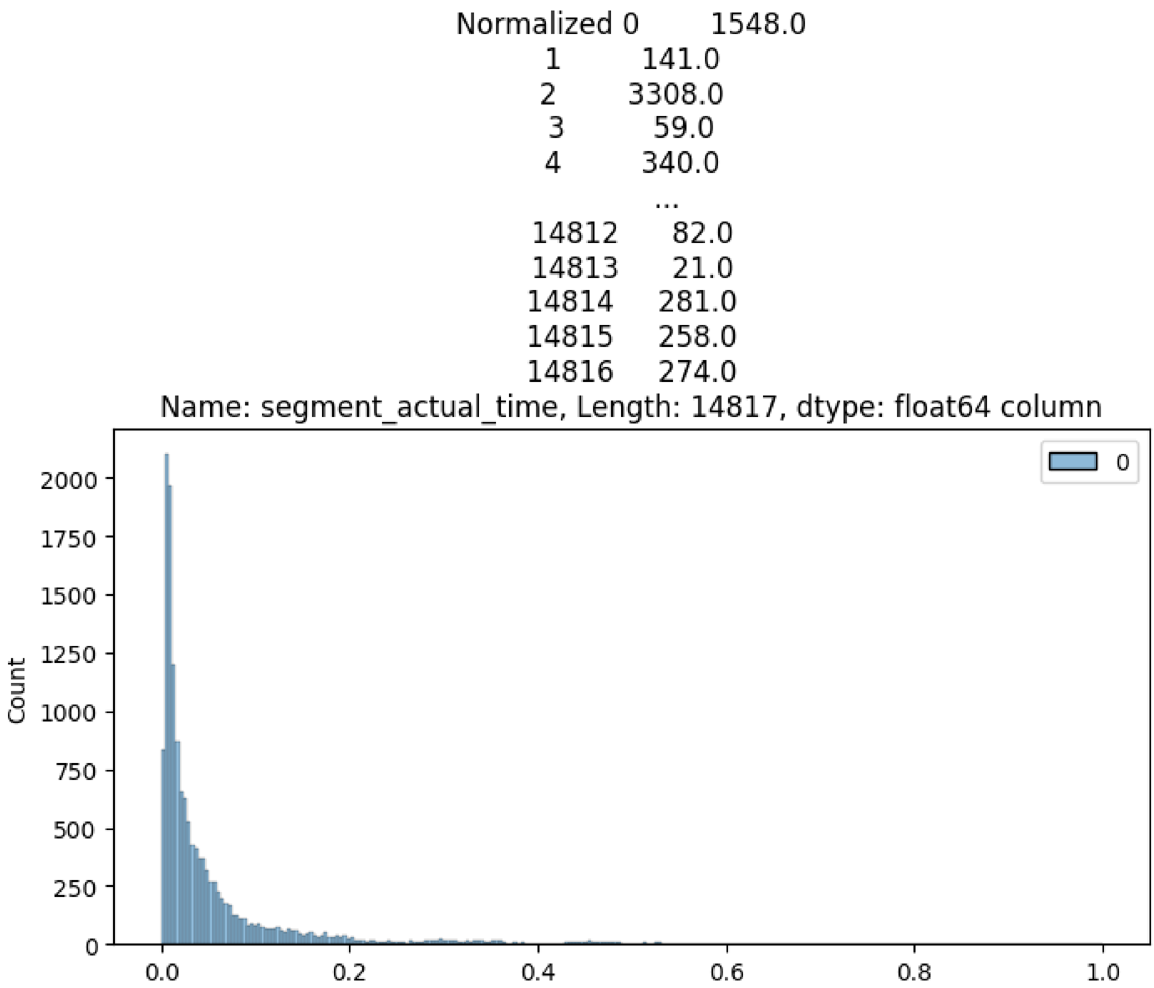
Normalized 0 991.3523
1 85.1110
2 2354.0665
3 19.6800
4 146.7918
...
14812 73.4630
14813 16.0882
14814 58.9037
14815 171.1103
14816 80.5787
Name: osrm_distance, Length: 14817, dtype: float64 column



In [104…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_actual_time']} column")
plt.plot()
```

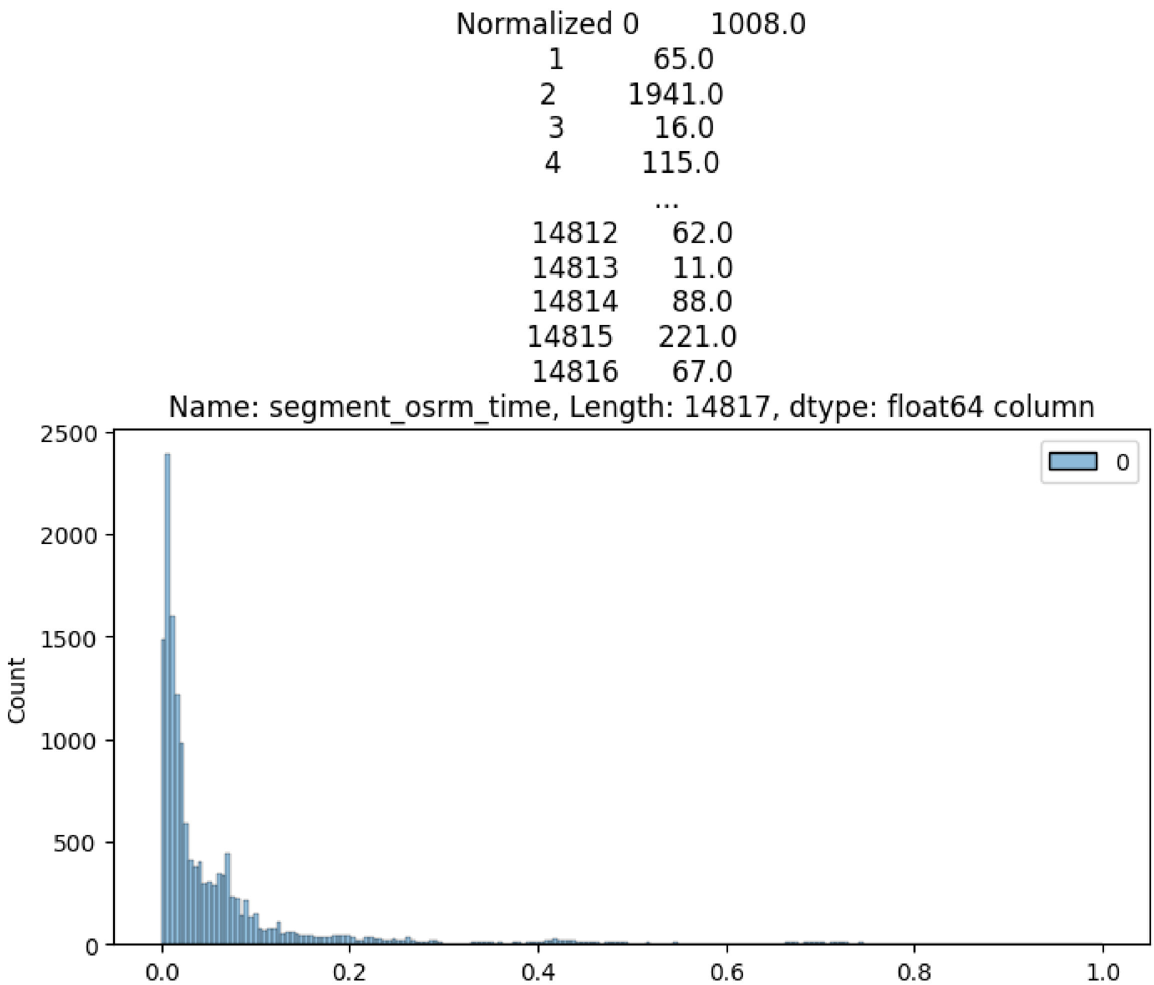
Out[104]: []



In [105…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_osrm_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_osrm_time']} column")
plt.plot()
```

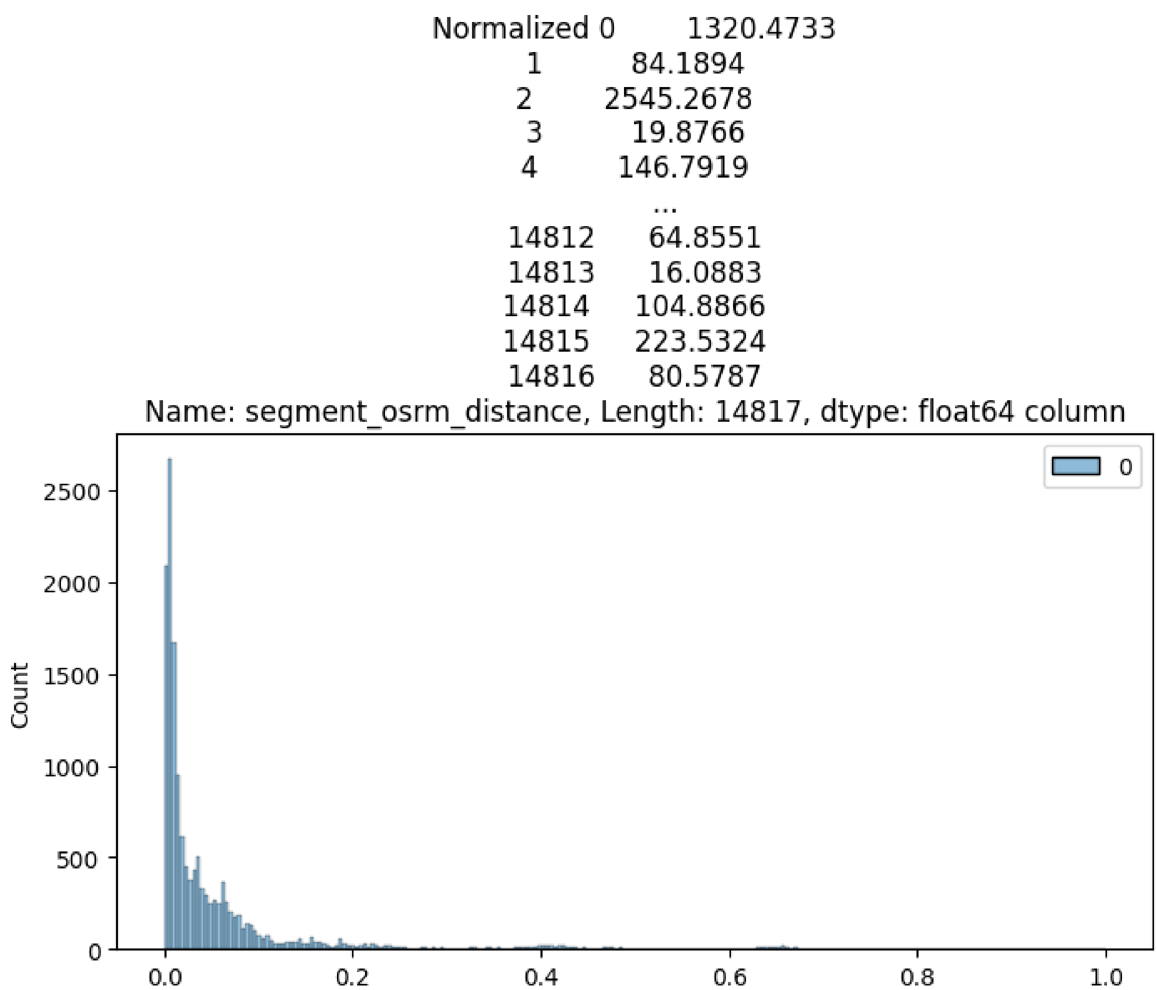
Out[105]: []



In [106…

```
plt.figure(figsize = (8, 4))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_osrm_distance'].to_numpy().reshape(-1
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_osrm_distance']} column")
plt.plot()
```

Out[106]: []



Column Standardization

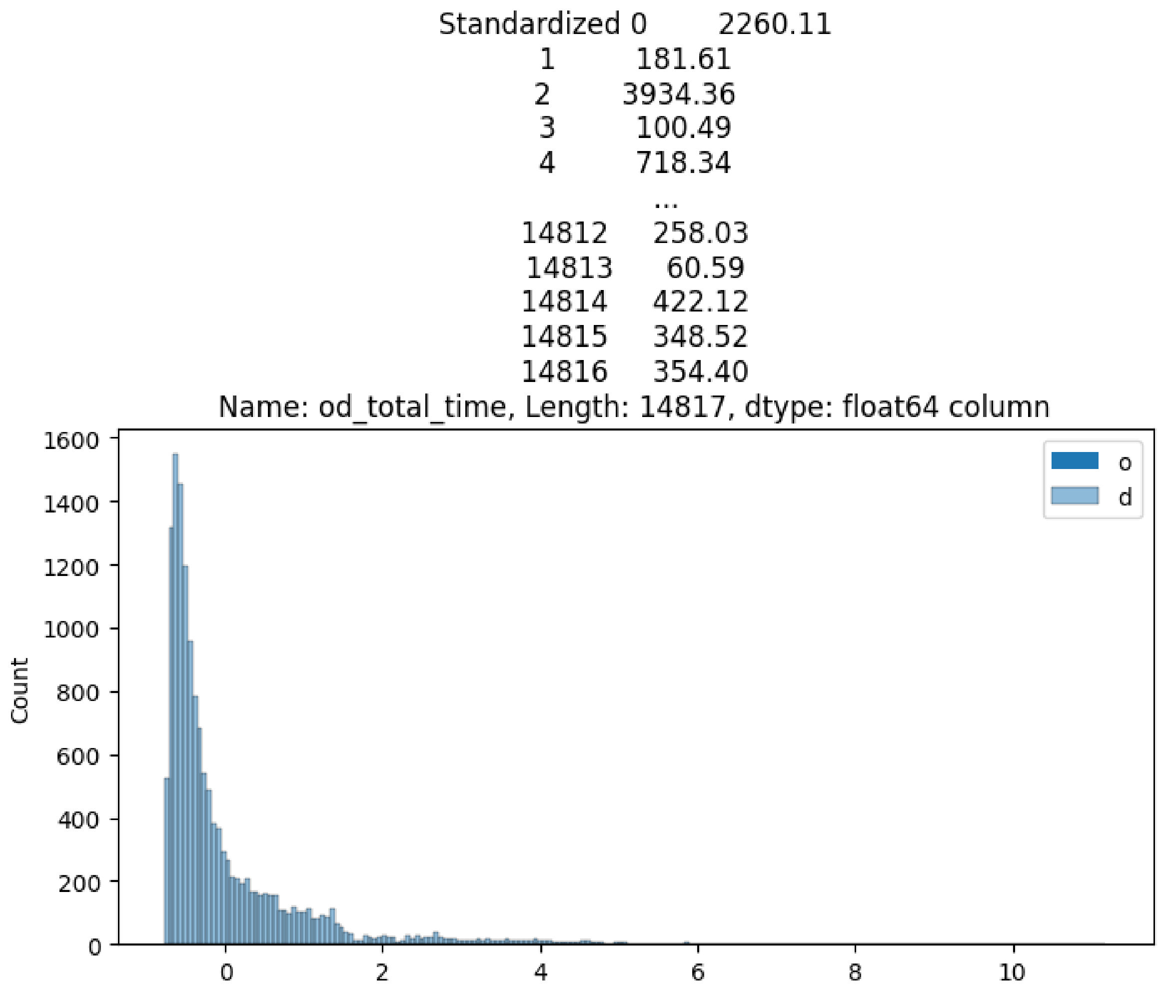
In [107…

```
from sklearn.preprocessing import StandardScaler
```

In [108…

```
plt.figure(figsize = (8, 4))
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['od_total_time']} column")
plt.legend('od_total_time')
plt.plot()
```

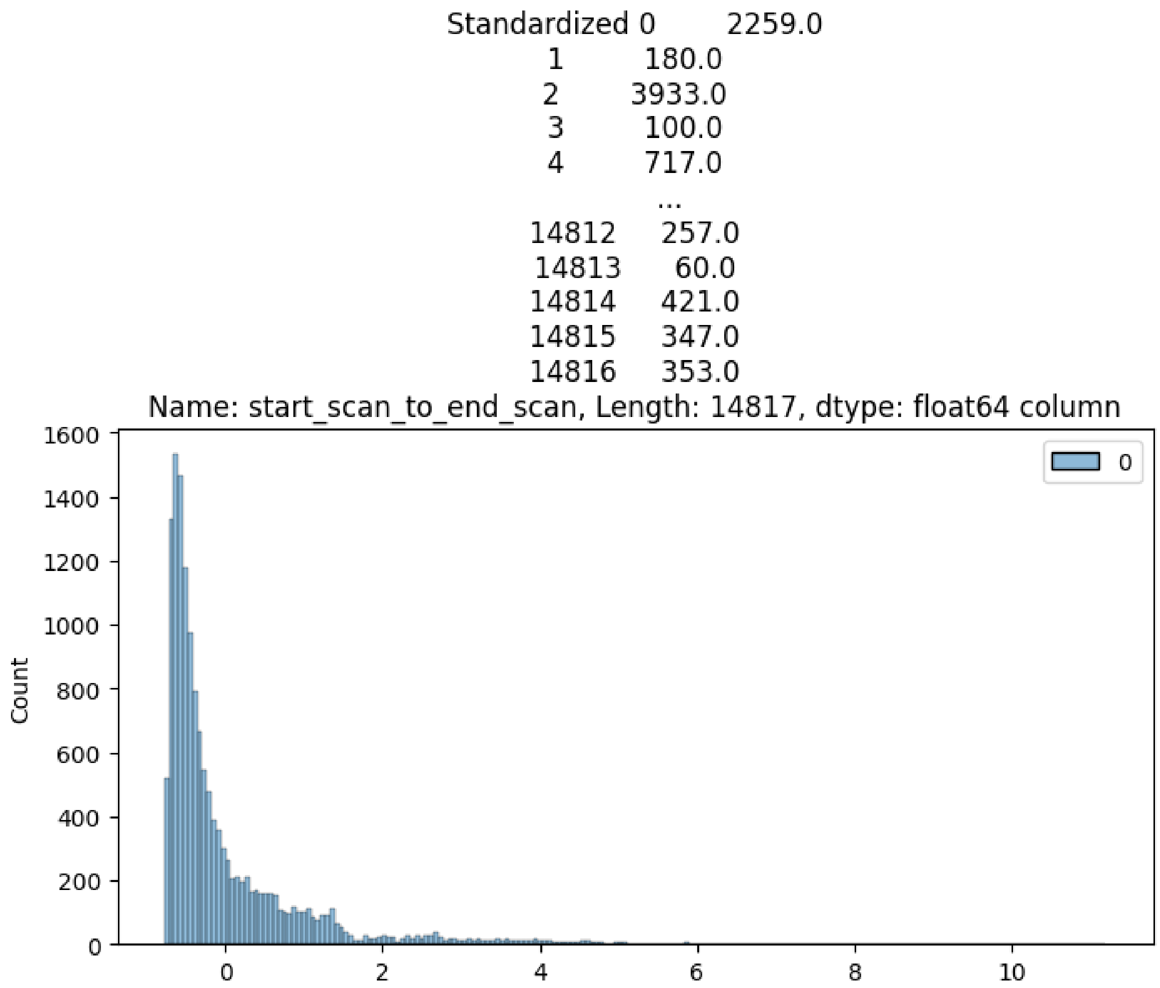
Out[108]: []



In [109…

```
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Standardized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

Out[109]: []

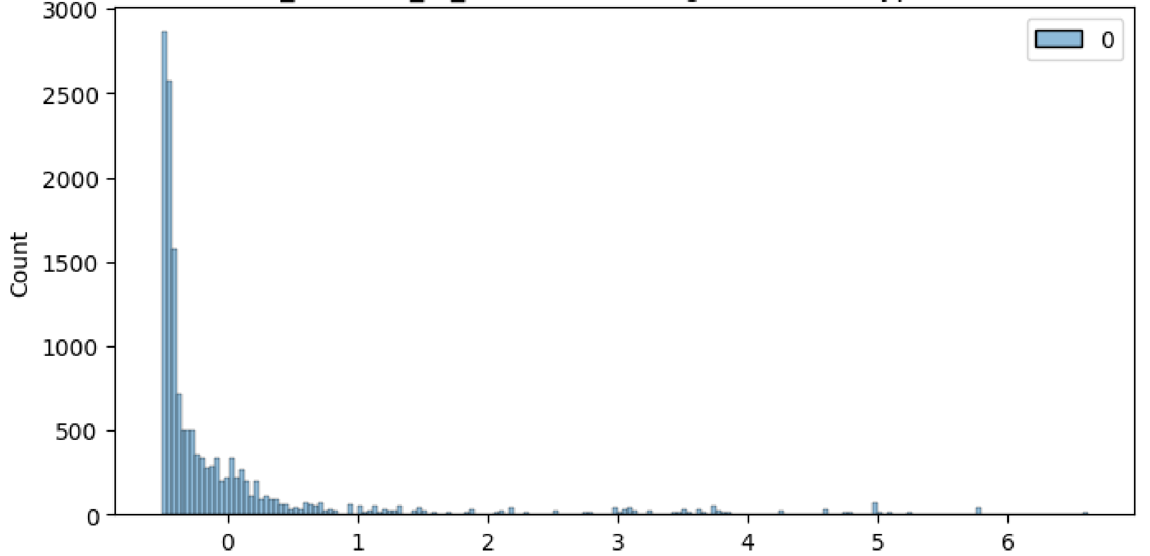


In [110…

```
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_distance_to_destination']} column")
plt.plot()
```

Out[110]: []

Standardized 0   824.732854
1   73.186911
2   1927.404273
3   17.175274
4   127.448500
...
14812   57.762332
14813   15.513784
14814   38.684839
14815   134.723836
14816   66.081533
Name: actual_distance_to_destination, Length: 14817, dtype: float64 column

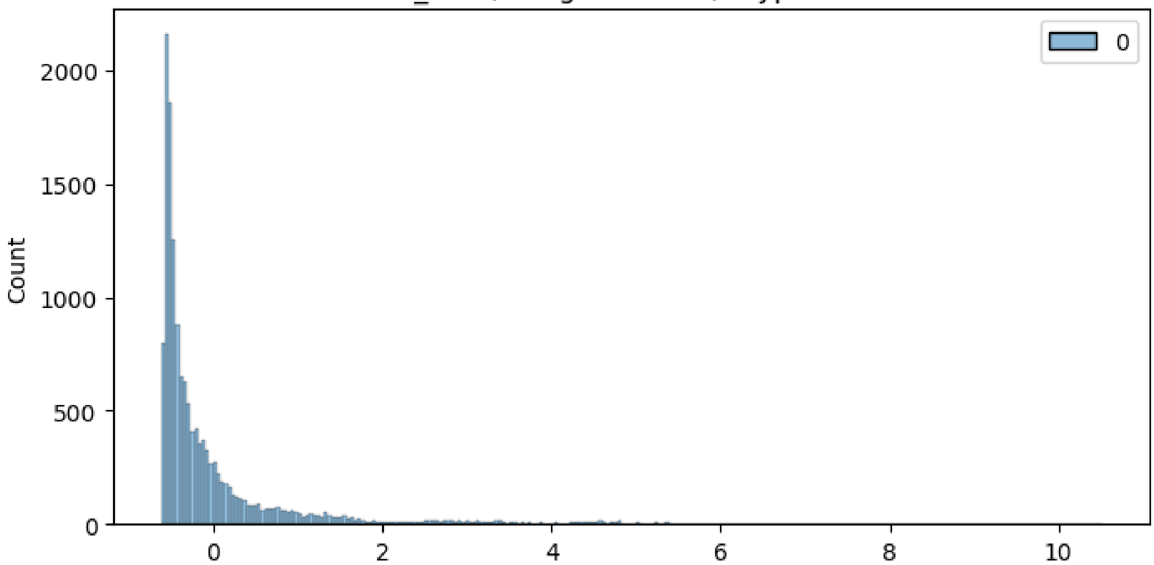


In [111…

```
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_time']} column")
plt.plot()
```

Out[111]: []

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: float64 column

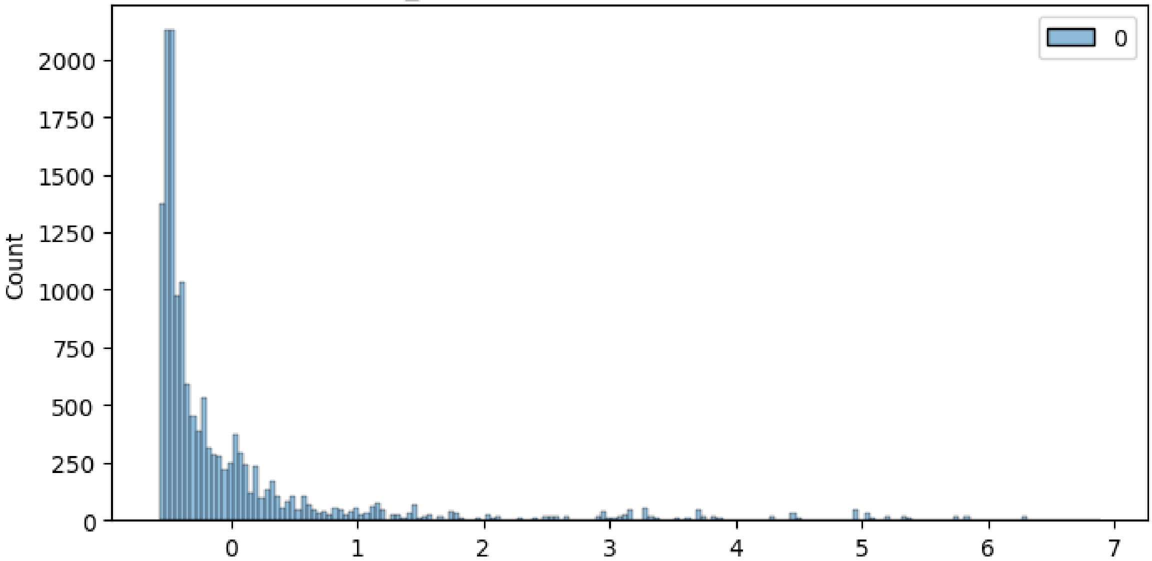


In [112…

```
plt.figure(figsize = (8, 4))
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[112]: []

```
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: float64 column
```

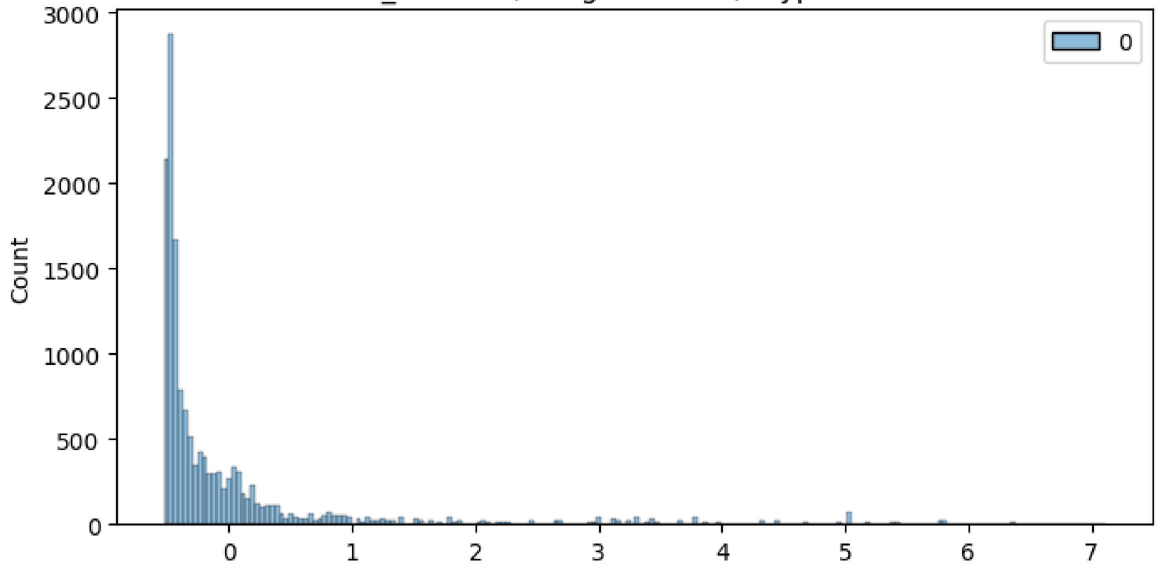


In [113…

```
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['osrm_distance']} column")
plt.plot()
```

Out[113]: []

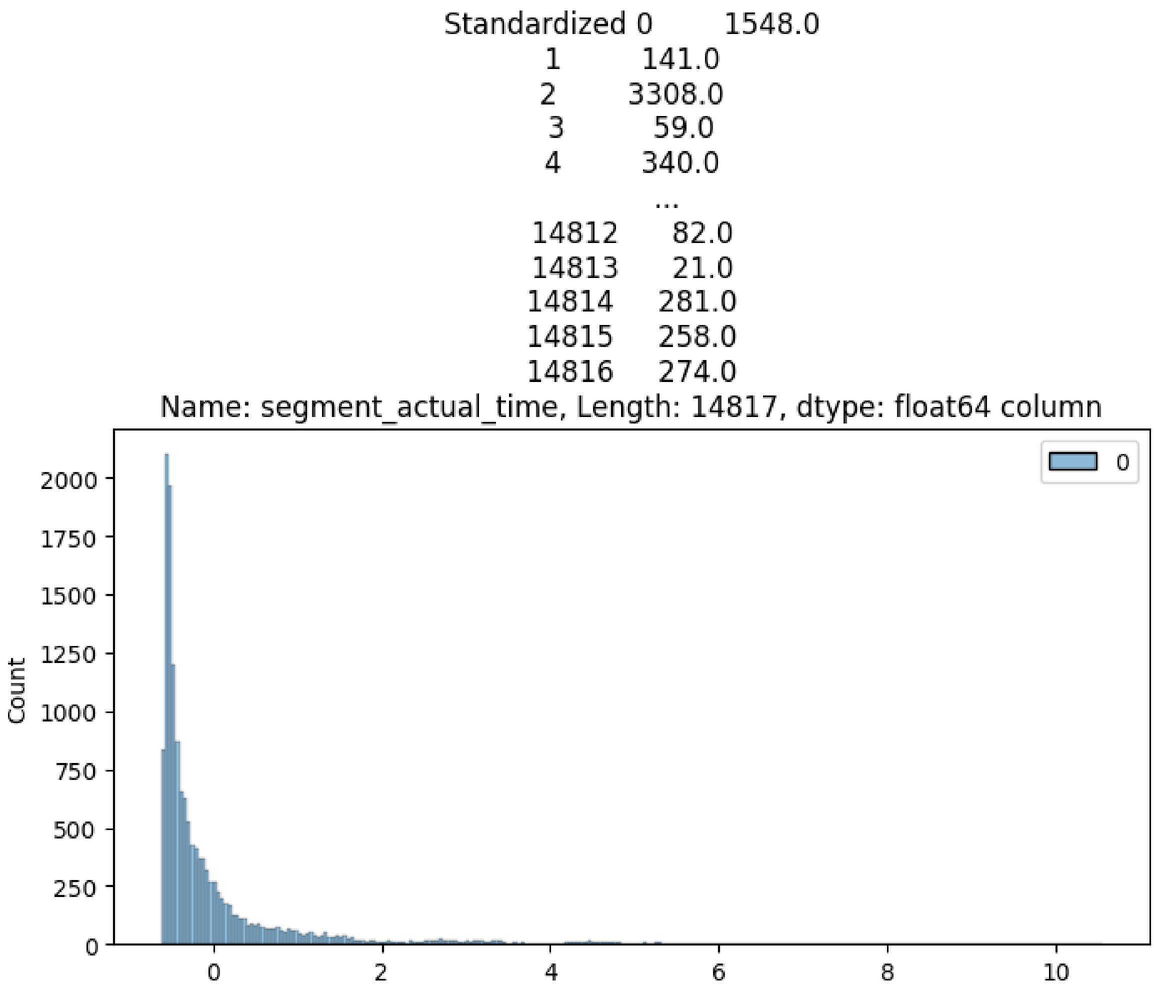
```
Standardized 0        991.3523
             1         85.1110
             2       2354.0665
             3         19.6800
             4        146.7918
                        ...
         14812         73.4630
         14813         16.0882
         14814         58.9037
         14815        171.1103
         14816         80.5787
Name: osrm_distance, Length: 14817, dtype: float64 column
```



In [114…

```python
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_actual_time']} column")
plt.plot()
```

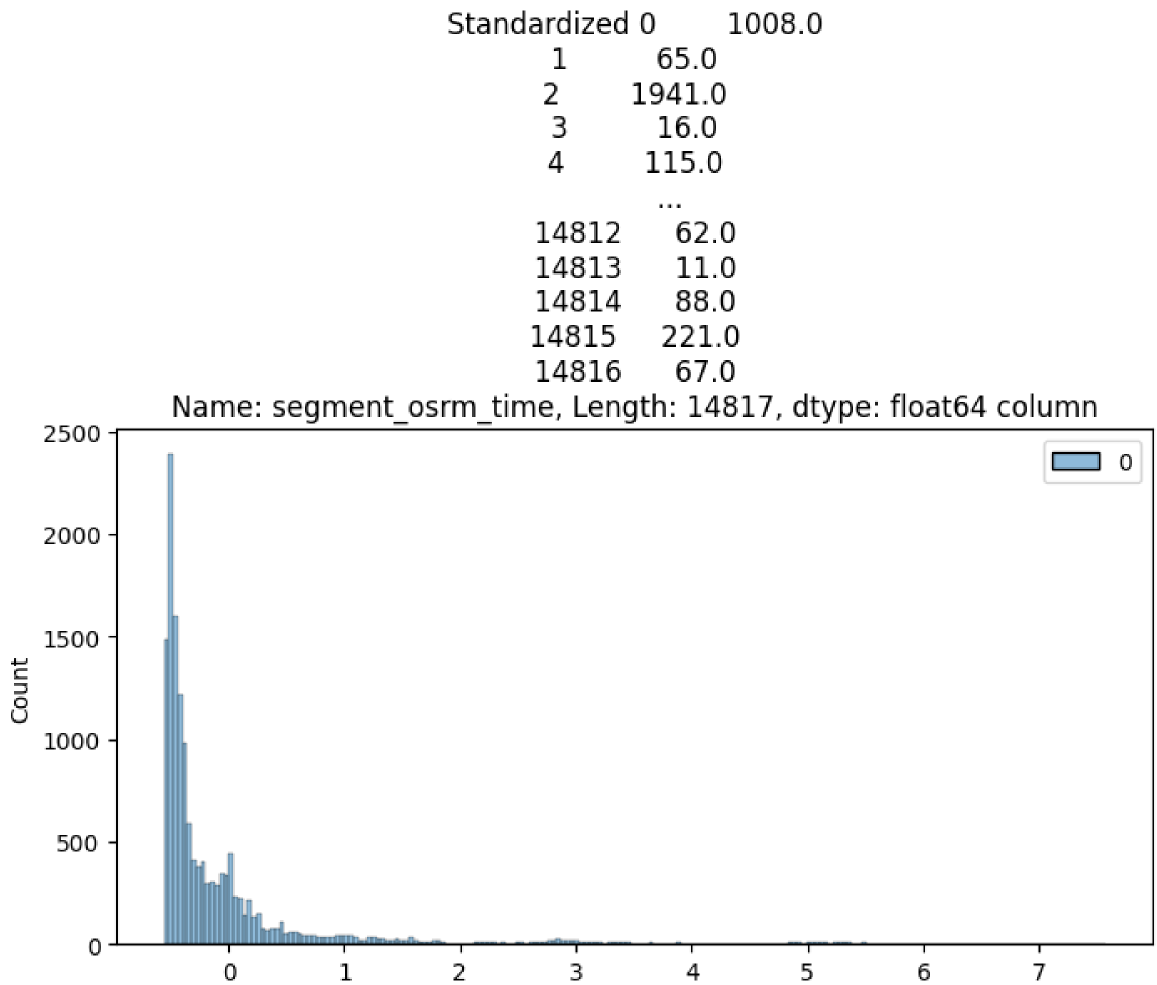
Out[114]: []



In [115…

```python
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_osrm_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_osrm_time']} column")
plt.plot()
```

Out[115]: []

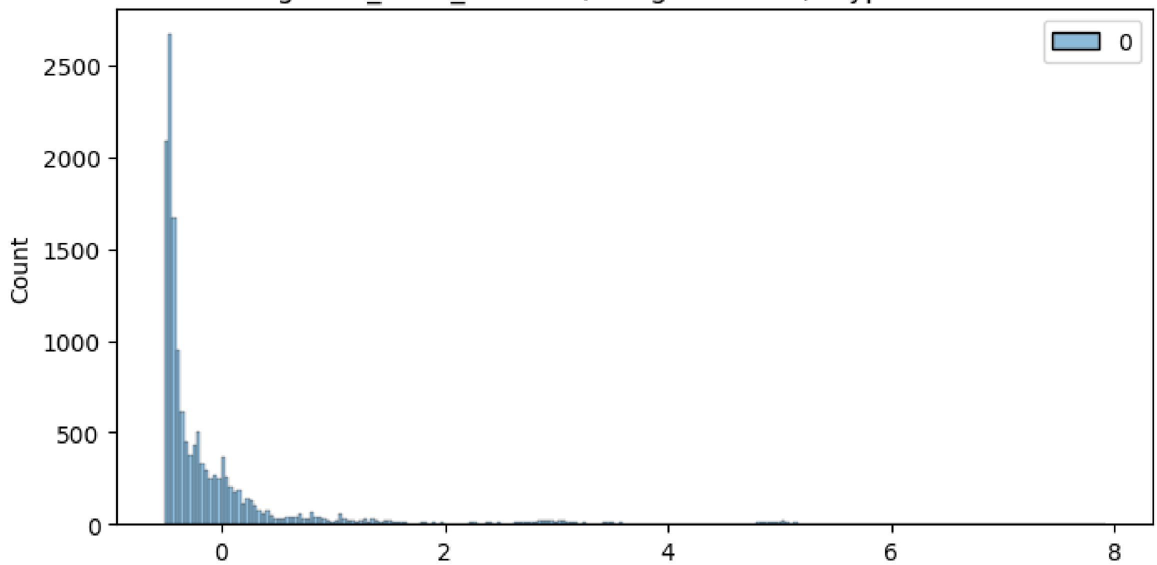


In [116…

```
plt.figure(figsize = (8, 4))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_osrm_distance'].to_numpy().reshape(-1
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_osrm_distance']} column")
plt.plot()
```

Out[116]: []

```
Standardized 0        1320.4733
             1          84.1894
             2        2545.2678
             3          19.8766
             4         146.7919
                        ...
             14812      64.8551
             14813      16.0883
             14814     104.8866
             14815     223.5324
             14816      80.5787
Name: segment_osrm_distance, Length: 14817, dtype: float64 column
```



Business Insights based on Non-Graphical and Visual Analysis

There are 144867 records with 24 columns which after merging and splitting to reduced to 14817 unique records and 28 columns. There are 2 columns with null values which were replaced with unique random values. From the statistical and categorical summary, we can observe that

1. On An Average the distance between source and destination is 164km and avg time taken is 357 mins between source and destinations.
2. There are 938 source and 1042 destination centers serving over 850 destination places.
3. The top most orders are sourced from Maharashtra and then followed by karnataka.
4. The top most Maximum number of trips originate from Mumbai city followed by Gurgaon Delhi, Bengaluru.
5. The top most destination state is Maharashtra and destination city is Mumbai, while the top destination place is Bilaspur. From the hypothesis testing we observe that:

Features start_scan_to_end_scan and od_total_time(difference between od_start_time and od_end_time) are statistically similar.

Features actual_time & osrm_time are statistically different.

Features actual_time and segment_actual_time are statistically similar.

Features osrm_distance and segment_osrm_distance are statistically different.

Features osrm_time & segment_osrm_time are statistically different.

categorical features 'route_type' and 'data' are encoded and represented in their binary form.

Recommendations:

The time estimated by OSRM (osrm_time) and the actual time taken differ. minimizing this disparity can provide customers with a more reliable expectation of when their deliveries will arrive, thereby contributing to overall convenience.

The distance calculated by the OSRM (Open Source Routing Machine) and the actual distance covered do not align. This discrepancy could stem from the delivery person deviating from the predefined route, potentially causing delays in deliveries. Alternatively, it might indicate inaccuracies in the OSRM device's predictions, which consider factors such as distance, traffic, and other variables.

A significant portion of orders originates from or is destined for states such as Maharashtra, Karnataka, Haryana, and Tamil Nadu. To strengthen market presence in these regions, optimization and expanding the current transportation routes are necessary.

Conducting customer profiling for individuals residing in states like Maharashtra, Karnataka, Haryana is essential. This will help to understand the reasons behind the huge volume of orders from these states and enhance the overall purchasing and delivery experience for customers.

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.

Some regions may experience high traffic, while others may face challenging terrain conditions. Utilizing this information can serve as a valuable indicator for strategically planning and addressing increased demand.

In [ ]: