keekbmd02

February 20, 2025

1 Business Case: Delhivery - Feature Engineering

1.1 About Delhivery

- Delhivery is the largest and fastest-growing fully integrated 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.

1.2 How can you help here?

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

1.2.1 Importing Required Libraries

[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
```

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

1.2.2 Loading the Dataset

[3]:
```
df = pd.read_csv(r"https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/
↪000/001/551/original/delhivery_data.csv?1642751181")
```

[4]:
```
df.head()
```

[4]:
```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

                                 route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting

                 trip_uuid source_center                 source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center               destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)

                od_start_time  …            cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  …         2018-09-20 04:27:55
1  2018-09-20 03:21:32.418600  …         2018-09-20 04:17:55
2  2018-09-20 03:21:32.418600  …  2018-09-20 04:01:19.505586
3  2018-09-20 03:21:32.418600  …         2018-09-20 03:39:57
4  2018-09-20 03:21:32.418600  …         2018-09-20 03:33:55

   actual_distance_to_destination  actual_time  osrm_time osrm_distance  \
0                       10.435660         14.0       11.0       11.9653
1                       18.936842         24.0       20.0       21.7243
2                       27.637279         40.0       28.0       32.5395
3                       36.118028         62.0       40.0       45.5620
4                       39.386040         68.0       44.0       54.2181

     factor  segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0  1.272727                 14.0               11.0                11.9653
1  1.200000                 10.0                9.0                 9.7590
2  1.428571                 16.0                7.0                10.8152
3  1.550000                 21.0               12.0                13.0224
```

```
4  1.545455                   6.0                 5.0                    3.9153

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

What is the shape of the loaded dataset ?

[5]:
```
df.shape
```

[5]:
```
(144867, 24)
```

What are the columns present in the dataset?

[6]:
```
df.columns
```

[6]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

What is the datatype of the columns ?

[7]:
```
df.dtypes
```

[7]:
```
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
```

Basic Information about the Dataset

[8]:
```
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
```

Dropping unknown fields

```
[9]: unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor',␣
      ↪'segment_factor']
     df = df.drop(columns = unknown_fields)
```

How many unique entries present in each column ?

```
[10]: 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 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 segment_actual_time            = 747
Unique entries for column segment_osrm_time              = 214
Unique entries for column segment_osrm_distance          = 113799
```

For all those columns where number of unique entries is 2, converting the datatype of columns to category

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

```
[12]: floating_columns = ['actual_distance_to_destination', 'actual_time',␣
       ↪'osrm_time', 'osrm_distance',
                           'segment_actual_time', 'segment_osrm_time',␣
       ↪'segment_osrm_distance']
      for i in floating_columns:
          print(df[i].max())
```

```
1927.4477046975032
4532.0
1686.0
2326.1991000000003
3051.0
```

```
1611.0
2191.4037000000003
```

We can update the datatype to float32 since the maximum value entry is small

[13]:
```python
for i in floating_columns:
    df[i] = df[i].astype('float32')
```

Updating the datatype of the datetime columns

[14]:
```python
datetime_columns = ['trip_creation_time', 'od_start_time', 'od_end_time']
for i in datetime_columns:
    df[i] = pd.to_datetime(df[i])
```

[15]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  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  actual_distance_to_destination  144867 non-null  float32
 13  actual_time                     144867 non-null  float32
 14  osrm_time                       144867 non-null  float32
 15  osrm_distance                   144867 non-null  float32
 16  segment_actual_time             144867 non-null  float32
 17  segment_osrm_time               144867 non-null  float32
 18  segment_osrm_distance           144867 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(7), float64(1), object(6)
memory usage: 15.2+ MB
```

Earlier the dataset was using 25.6+ MB of memory but now it has been reduced to 15.2 + MB. Around 40.63 % reduction in the memory usage.

What is the time period for which the data is given ?

[16]:
```python
df['trip_creation_time'].min(), df['od_end_time'].max()
```

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

1.3 1. Basic data cleaning and exploration:

1.3.1 Handling missing values in the data

Is there any null values present in the dataset ?

[17]:
```
np.any(df.isnull())
```

[17]: True

What is the number of null values present in each column ?

[18]:
```
df.isnull().sum()
```

[18]:
```
data                                0
trip_creation_time                  0
route_schedule_uuid                 0
route_type                          0
trip_uuid                           0
source_center                       0
source_name                       293
destination_center                  0
destination_name                  261
od_start_time                       0
od_end_time                         0
start_scan_to_end_scan              0
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
dtype: int64
```

[19]:
```
missing_source_name = df.loc[df['source_name'].isnull(), 'source_center'].
↪unique()
missing_source_name
```

[19]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

[20]:
```
for i in missing_source_name:
    unique_source_name = df.loc[df['source_center'] == i, 'source_name'].
↪unique()
```

```
    if pd.isna(unique_source_name):
        print("Source Center :", i, "-" * 10, "Source Name :", 'Not Found')
    else :
        print("Source Center :", i, "-" * 10, "Source Name :",␣
↪unique_source_name)
```

```
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
```

[21]:
```
for i in missing_source_name:
    unique_destination_name = df.loc[df['destination_center'] == i,␣
↪'destination_name'].unique()
    if (pd.isna(unique_source_name)) or (unique_source_name.size == 0):
        print("Destination Center :", i, "-" * 10, "Destination Name :", 'Not␣
↪Found')
    else :
        print("Destination Center :", i, "-" * 10, "Destination Name :",␣
↪unique_destination_name)
```

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

[22]:
```
missing_destination_name = df.loc[df['destination_name'].isnull(),␣
↪'destination_center'].unique()
missing_destination_name
```

[22]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

The IDs for which the source name is missing, are all those IDs for destination also missing ?

[23]:
```
np.all(df.loc[df['source_name'].isnull(), 'source_center'].
↪isin(missing_destination_name))
```

[23]: False

Treating missing destination names and source names

[24]:
```
count = 1
for i in missing_destination_name:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.
↪loc[df['destination_center'] == i, 'destination_name'].replace(np.nan,␣
↪f'location_{count}')
    count += 1
```

[25]:
```
d = {}
for i in missing_source_name:
    d[i] = df.loc[df['destination_center'] == i, 'destination_name'].unique()
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
```

[26]:
```
for i in missing_source_name:
    df.loc[df['source_center'] == i, 'source_name'] = df.
↪loc[df['source_center'] == i, 'source_name'].replace(np.nan, d2[i])
```

[27]:
```
df.isna().sum()
```

[27]:
```
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
actual_distance_to_destination   0
actual_time                      0
osrm_time                        0
osrm_distance                    0
segment_actual_time              0
segment_osrm_time                0
segment_osrm_distance            0
dtype: int64
```

Basic Description of the Data

[28]:
```
df.describe()
```

[28]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time
count	144867.000000	144867.000000	144867.000000
mean	961.262986	234.050812	416.929504
std	1037.012769	344.979126	598.096069
min	20.000000	9.000046	9.000000
25%	161.000000	23.355875	51.000000
50%	449.000000	66.126572	132.000000
75%	1634.000000	286.708878	513.000000
max	7898.000000	1927.447754	4532.000000

	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time
count	144867.000000	144867.000000	144867.000000	144867.000000
mean	213.864685	284.768158	36.196110	18.507547
std	308.004333	421.117462	53.566002	14.770471
min	6.000000	9.008200	-244.000000	0.000000
25%	27.000000	29.914701	20.000000	11.000000
50%	64.000000	78.525803	29.000000	17.000000
75%	257.000000	343.193253	40.000000	22.000000
max	1686.000000	2326.199219	3051.000000	1611.000000

	segment_osrm_distance
count	144867.000000
mean	22.829105
std	17.860197
min	0.000000

```
25%                12.070100
50%                23.513000
75%                27.813250
max              2191.403809
```

[29]:
```python
df.describe(include = 'object')
```

[29]:
```
                                        route_schedule_uuid  \
count                                                144867
unique                                                 1504
top     thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f…
freq                                                   1812

                  trip_uuid source_center                  source_name  \
count                144867        144867                       144867
unique                14817          1508                         1508
top     trip-153811219535896559  IND000000ACB  Gurgaon_Bilaspur_HB (Haryana)
freq                    101         23347                        23347

       destination_center               destination_name
count              144867                         144867
unique               1481                           1481
top          IND000000ACB  Gurgaon_Bilaspur_HB (Haryana)
freq                15192                          15192
```

1.3.2 Merging of rows and aggregation of fields

How to begin"

- 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.

[30]:
```python
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' :␣
 ↪'first',
                                                         'source_name' : 'first',
                                                         'destination_name' :␣
 ↪'last',
                                                         'od_start_time' :␣
 ↪'first',
                                                         'od_end_time' : 'first',
                                                         'start_scan_to_end_scan'␣
 ↪: 'first',
```

```
                                                          ␣
↪'actual_distance_to_destination' : 'last',
                                                           'actual_time' : 'last',
                                                           'osrm_time' : 'last',
                                                           'osrm_distance' : 'last',
                                                           'segment_actual_time' :␣
↪'sum',
                                                           'segment_osrm_time' :␣
↪'sum',
                                                           'segment_osrm_distance' :
↪ 'sum'})
df1
```

[30]:
```
                         trip_uuid source_center destination_center      data  \
0      trip-153671041653548748  IND209304AAA       IND000000ACB  training
1      trip-153671041653548748  IND462022AAA       IND209304AAA  training
2      trip-153671042288605164  IND561203AAB       IND562101AAA  training
3      trip-153671042288605164  IND572101AAA       IND561203AAB  training
4      trip-153671043369099517  IND000000ACB       IND160002AAC  training
…                          …           …                …       …
26363  trip-153861115439069069  IND628204AAA       IND627657AAA      test
26364  trip-153861115439069069  IND628613AAA       IND627005AAA      test
26365  trip-153861115439069069  IND628801AAA       IND628204AAA      test
26366  trip-153861118270144424  IND583119AAA       IND583101AAA      test
26367  trip-153861118270144424  IND583201AAA       IND583119AAA      test

      route_type          trip_creation_time  \
0            FTL 2018-09-12 00:00:16.535741
1            FTL 2018-09-12 00:00:16.535741
2        Carting 2018-09-12 00:00:22.886430
3        Carting 2018-09-12 00:00:22.886430
4            FTL 2018-09-12 00:00:33.691250
…            …                        …
26363    Carting 2018-10-03 23:59:14.390954
26364    Carting 2018-10-03 23:59:14.390954
26365    Carting 2018-10-03 23:59:14.390954
26366        FTL 2018-10-03 23:59:42.701692
26367        FTL 2018-10-03 23:59:42.701692

                                source_name  \
0      Kanpur_Central_H_6 (Uttar Pradesh)
1      Bhopal_Trnsport_H (Madhya Pradesh)
2       Doddablpur_ChikaDPP_D (Karnataka)
3           Tumkur_Veersagr_I (Karnataka)
4           Gurgaon_Bilaspur_HB (Haryana)
…                                       …
26363  Tirchchndr_Shnmgprm_D (Tamil Nadu)
```

```
26364   Peikulam_SriVnktpm_D (Tamil Nadu)
26365        Eral_Busstand_D (Tamil Nadu)
26366       Sandur_WrdN1DPP_D (Karnataka)
26367                 Hospet (Karnataka)

                             destination_name              od_start_time  \
0              Gurgaon_Bilaspur_HB (Haryana) 2018-09-12 16:39:46.858469
1         Kanpur_Central_H_6 (Uttar Pradesh) 2018-09-12 00:00:16.535741
2          Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591
3          Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
4             Chandigarh_Mehmdpur_H (Punjab) 2018-09-14 03:40:17.106733
…                                          …                          …
26363  Thisayanvilai_UdnkdiRD_D (Tamil Nadu) 2018-10-04 02:29:04.272194
26364    Tirunelveli_VdkkuSrt_I (Tamil Nadu) 2018-10-04 04:16:39.894872
26365     Tirchchndr_Shnmgprm_D (Tamil Nadu) 2018-10-04 01:44:53.808000
26366                 Bellary_Dc (Karnataka) 2018-10-04 03:58:40.726547
26367           Sandur_WrdN1DPP_D (Karnataka) 2018-10-04 02:51:44.712656

                      od_end_time  start_scan_to_end_scan  \
0     2018-09-13 13:40:23.123744                  1260.0
1     2018-09-12 16:39:46.858469                   999.0
2     2018-09-12 03:01:59.598855                    58.0
3     2018-09-12 02:03:09.655591                   122.0
4     2018-09-14 17:34:55.442454                   834.0
…                              …                      …
26363 2018-10-04 03:31:11.183797                    62.0
26364 2018-10-04 05:47:45.162682                    91.0
26365 2018-10-04 02:29:04.272194                    44.0
26366 2018-10-04 08:46:09.166940                   287.0
26367 2018-10-04 03:58:40.726547                    66.0

       actual_distance_to_destination  actual_time  osrm_time  osrm_distance  \
0                          383.759155        732.0      329.0     446.549591
1                          440.973694        830.0      388.0     544.802673
2                           24.644020         47.0       26.0      28.199400
3                           48.542889         96.0       42.0      56.911598
4                          237.439606        611.0      212.0     281.210907
…                                   …            …          …              …
26363                       33.627182         51.0       41.0      42.521301
26364                       33.673836         90.0       48.0      40.608002
26365                       12.661944         30.0       14.0      16.018499
26366                       40.546738        233.0       42.0      52.530300
26367                       25.534794         42.0       26.0      28.048401

       segment_actual_time  segment_osrm_time  segment_osrm_distance
0                    728.0              534.0             670.620483
1                    820.0              474.0             649.852783
```

```
2                    46.0              26.0               28.199501
3                    95.0              39.0               55.989899
4                   608.0             231.0              317.740784
…                      …                 …                    …
26363                49.0              42.0               42.143101
26364                89.0              77.0               78.586899
26365                29.0              14.0               16.018400
26366               233.0              42.0               52.530300
26367                41.0              25.0               28.048401

[26368 rows x 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

[31]:
```
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_seconds() / 60.0, 2))
df1['od_total_time'].head()
```

[31]:
```
0    1260.60
1     999.51
2      58.83
3     122.78
4     834.64
Name: od_total_time, dtype: float64
```

[32]:
```
df2 = df1.groupby(by = 'trip_uuid', as_index = False).agg({'source_center' :␣
 ↪'first',
                                                          'destination_center'␣
 ↪: 'last',
                                                          'data' : 'first',
                                                          'route_type' :␣
 ↪'first',
                                                          'trip_creation_time'␣
 ↪: 'first',
                                                          'source_name' :␣
 ↪'first',
                                                          'destination_name' :␣
 ↪'last',
                                                          'od_total_time' :␣
 ↪'sum',
                                                         ␣
 ↪'start_scan_to_end_scan' : 'sum',
                                                         ␣
 ↪'actual_distance_to_destination' : 'sum',
```

```
                                                            'actual_time' :␣
↪'sum',
                                                            'osrm_time' : 'sum',
                                                            'osrm_distance' :␣
↪'sum',
                                                           ␣
↪'segment_actual_time' : 'sum',
                                                            'segment_osrm_time' :
↪ 'sum',
                                                           ␣
↪'segment_osrm_distance' : 'sum'})
df2
```

[32]:

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

      route_type          trip_creation_time  \
0            FTL 2018-09-12 00:00:16.535741
1        Carting 2018-09-12 00:00:22.886430
2            FTL 2018-09-12 00:00:33.691250
3        Carting 2018-09-12 00:01:00.113710
4            FTL 2018-09-12 00:02:09.740725
…          …                          …
14812    Carting 2018-10-03 23:55:56.258533
14813    Carting 2018-10-03 23:57:23.863155
14814    Carting 2018-10-03 23:57:44.429324
14815    Carting 2018-10-03 23:59:14.390954
14816        FTL 2018-10-03 23:59:42.701692

                                source_name  \
0       Kanpur_Central_H_6 (Uttar Pradesh)
1        Doddablpur_ChikaDPP_D (Karnataka)
2            Gurgaon_Bilaspur_HB (Haryana)
3                 Mumbai Hub (Maharashtra)
4                   Bellary_Dc (Karnataka)
…                                        …
14812       Chandigarh_Mehmdpur_H (Punjab)
```

```
14813        FBD_Balabhgarh_DPC (Haryana)
14814  Kanpur_GovndNgr_DC (Uttar Pradesh)
14815 Tirunelveli_VdkkuSrt_I (Tamil Nadu)
14816       Sandur_WrdN1DPP_D (Karnataka)

                          destination_name  od_total_time  \
0      Kanpur_Central_H_6 (Uttar Pradesh)        2260.11
1       Doddablpur_ChikaDPP_D (Karnataka)         181.61
2           Gurgaon_Bilaspur_HB (Haryana)        3934.36
3          Mumbai_MiraRd_IP (Maharashtra)         100.49
4           Sandur_WrdN1DPP_D (Karnataka)         718.34
…                                       …              …
14812      Chandigarh_Mehmdpur_H (Punjab)         258.03
14813      Faridabad_Blbgarh_DC (Haryana)          60.59
14814  Kanpur_GovndNgr_DC (Uttar Pradesh)         422.12
14815  Tirchchndr_Shnmgprm_D (Tamil Nadu)         348.52
14816       Sandur_WrdN1DPP_D (Karnataka)         354.40

       start_scan_to_end_scan  actual_distance_to_destination  actual_time  \
0                      2259.0                      824.732849       1562.0
1                       180.0                       73.186905        143.0
2                      3933.0                     1927.404297       3347.0
3                       100.0                       17.175274         59.0
4                       717.0                      127.448502        341.0
…                           …                               …            …
14812                   257.0                       57.762333         83.0
14813                    60.0                       15.513784         21.0
14814                   421.0                       38.684837        282.0
14815                   347.0                      134.723831        264.0
14816                   353.0                       66.081528        275.0

       osrm_time  osrm_distance  segment_actual_time  segment_osrm_time  \
0          717.0     991.352295               1548.0             1008.0
1           68.0      85.111000                141.0               65.0
2         1740.0    2354.066650               3308.0             1941.0
3           15.0      19.680000                 59.0               16.0
4          117.0     146.791794                340.0              115.0
…              …              …                    …                  …
14812       62.0      73.462997                 82.0               62.0
14813       12.0      16.088200                 21.0               11.0
14814       48.0      58.903702                281.0               88.0
14815      179.0     171.110306                258.0              221.0
14816       68.0      80.578705                274.0               67.0

       segment_osrm_distance
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               223.532410
14816                80.578705

[14817 rows x 17 columns]
```

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

1.4.1 Source Name: Split and extract features out of destination. City-place-code (State)

[33]:
```python
def location_name_to_state(x):
    l = x.split('(')
    if len(l) == 1:
        return l[0]
    else:
        return l[1].replace(')', "")
```

[34]:
```python
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]
```

[35]:
```
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]
```

[36]:
```
df2['source_state'] = df2['source_name'].apply(location_name_to_state)
df2['source_state'].unique()
```

[36]:
```
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)
```

[37]:
```
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
```

[37]:
```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Bellary', 'Chennai',
       'Bengaluru', 'Surat', 'Delhi', 'Pune', 'Faridabad', 'Shirala',
       'Hyderabad', 'Thirumalagiri', 'Gulbarga', 'Jaipur', 'Allahabad',
       'Guwahati', 'Narsinghpur', 'Shrirampur', 'Madakasira', 'Sonari',
       'Dindigul', 'Jalandhar', 'Chandigarh', 'Deoli', 'Pandharpur',
       'Kolkata', 'Bhandara', 'Kurnool', 'Bhiwandi', 'Bhatinda',
       'RoopNagar', 'Bantwal', 'Lalru', 'Kadi', 'Shahdol', 'Gangakher',
```

```
       'Durgapur', 'Vapi', 'Jamjodhpur', 'Jetpur', 'Mehsana', 'Jabalpur',
       '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', 'Bilimora',
       'SultnBthry', 'Lucknow', 'Vellore', 'Bhuj', 'Dinhata',
       'Margherita', 'Boisar', 'Vizag', 'Tezpur', 'Koduru', 'Tirupati',
       'Pen', 'Ahmedabad', 'Faizabad', 'Gandhinagar', 'Anantapur',
       'Betul', 'Panskura', 'Rasipurm', 'Sankari', 'Jorhat', 'PNQ',
       'Srikakulam', 'Dehradun', 'Jassur', 'Sawantwadi', 'Shajapur',
       'Ludhiana', 'GreaterThane'], dtype=object)
```

[38]:
```
df2['source_place'] = df2['source_name'].apply(location_name_to_place)
df2['source_place'].unique()[:100]
```

[38]:
```
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', '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)
```

1.4.2 Destination Name: Split and extract features out of destination. City-place-code (State)

[39]:
```
df2['destination_state'] = df2['destination_name'].apply(location_name_to_state)
df2['destination_state'].head(10)
```

```
[39]: 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
```

```
[40]: df2['destination_city'] = df2['destination_name'].apply(location_name_to_city)
      df2['destination_city'].head()
```

```
[40]: 0        Kanpur
      1    Doddablpur
      2       Gurgaon
      3        Mumbai
      4        Sandur
      Name: destination_city, dtype: object
```

```
[41]: df2['destination_place'] = df2['destination_name'].apply(location_name_to_place)
      df2['destination_place'].head()
```

```
[41]: 0    Central_H_6
      1     ChikaDPP_D
      2    Bilaspur_HB
      3      MiraRd_IP
      4     WrdN1DPP_D
      Name: destination_place, dtype: object
```

1.4.3 Trip_creation_time: Extract features like month, year and day etc

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

```
[42]: 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]
```

```
[43]: 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()
```

[43]:
```
0    12
1    12
2    12
3    12
4    12
Name: trip_creation_day, dtype: int8
```

[44]:
```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()
```

[44]:
```
0    9
1    9
2    9
3    9
4    9
Name: trip_creation_month, dtype: int8
```

[45]:
```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()
```

[45]:
```
0    2018
1    2018
2    2018
3    2018
4    2018
Name: trip_creation_year, dtype: int16
```

[46]:
```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()
```

[46]:
```
0    37
1    37
2    37
3    37
4    37
Name: trip_creation_week, dtype: int8
```

[47]:
```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()
```

[47]:
```
0    0
1    0
2    0
```

```
3    0
4    0
Name: trip_creation_hour, dtype: int8
```

Finding the structure of data after data cleaning

[48]:
```python
df2.shape
```

[48]:
```
(14817, 29)
```

[49]:
```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  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
```

[50]:

```
df2.describe().T
```

[50]:

```
                                  count         mean         std          min  \
od_total_time                   14817.0   531.697630  658.868223    23.460000
start_scan_to_end_scan          14817.0   530.810016  658.705957    23.000000
actual_distance_to_destination  14817.0   164.477951  305.388123     9.002461
actual_time                     14817.0   357.143768  561.395020     9.000000
osrm_time                       14817.0   161.384018  271.362549     6.000000
osrm_distance                   14817.0   204.345078  370.395508     9.072900
segment_actual_time             14817.0   353.892273  556.246826     9.000000
segment_osrm_time               14817.0   180.949783  314.541412     6.000000
segment_osrm_distance           14817.0   223.201324  416.628326     9.072900
trip_creation_day               14817.0    18.370790    7.893275     1.000000
trip_creation_month             14817.0     9.120672    0.325757     9.000000
trip_creation_year              14817.0  2018.000000    0.000000  2018.000000
trip_creation_week              14817.0    38.295944    0.967872    37.000000
trip_creation_hour              14817.0    12.449821    7.986553     0.000000

                                        25%          50%          75%  \
od_total_time                    149.930000   280.770000   638.200000
start_scan_to_end_scan           149.000000   280.000000   637.000000
actual_distance_to_destination    22.837238    48.474072   164.583206
actual_time                       67.000000   149.000000   370.000000
osrm_time                         29.000000    60.000000   168.000000
osrm_distance                     30.819201    65.618805   208.475006
segment_actual_time               66.000000   147.000000   367.000000
segment_osrm_time                 31.000000    65.000000   185.000000
segment_osrm_distance             32.654499    70.154404   218.802399
trip_creation_day                 14.000000    19.000000    25.000000
trip_creation_month                9.000000     9.000000     9.000000
trip_creation_year              2018.000000  2018.000000  2018.000000
trip_creation_week                38.000000    38.000000    39.000000
trip_creation_hour                 4.000000    14.000000    20.000000

                                        max
od_total_time                   7898.550000
start_scan_to_end_scan          7898.000000
actual_distance_to_destination  2186.531738
actual_time                     6265.000000
osrm_time                       2032.000000
osrm_distance                   2840.081055
segment_actual_time             6230.000000
segment_osrm_time               2564.000000
segment_osrm_distance           3523.632324
trip_creation_day                 30.000000
```

```
trip_creation_month                10.000000
trip_creation_year               2018.000000
trip_creation_week                 40.000000
trip_creation_hour                 23.000000
```

[51]:
```python
df2.describe(include = object).T
```

[51]:

	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

I am intrested to know how many trips are created on the hourly basis

[52]:
```python
df2['trip_creation_hour'].unique()
```

[52]:
```
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)
```

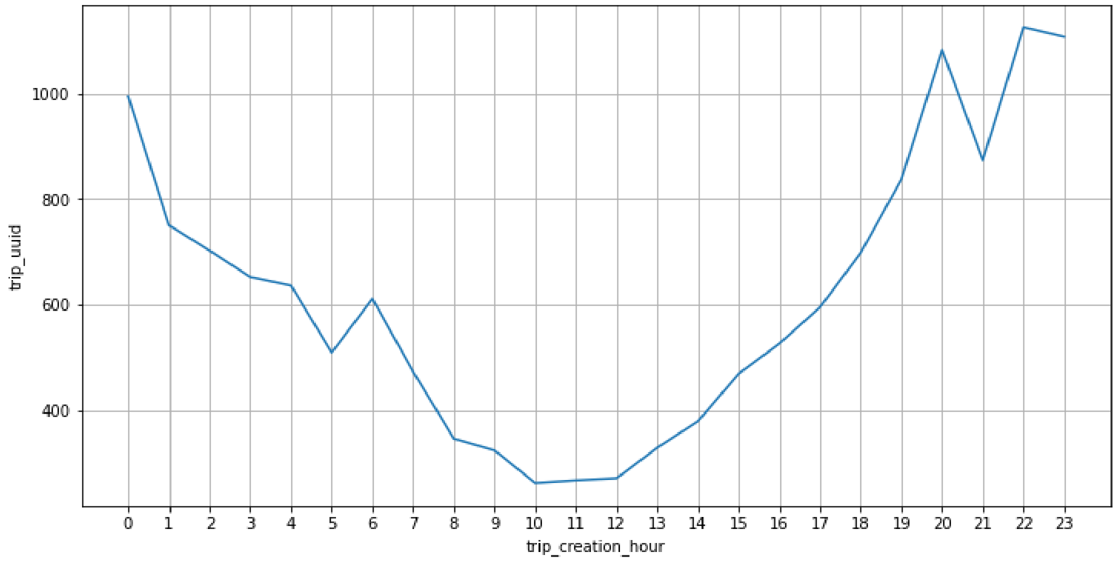
[53]:
```python
df_hour = df2.groupby(by = 'trip_creation_hour')['trip_uuid'].count().
 ↪to_frame().reset_index()
df_hour.head()
```

[53]:

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

[54]:
```python
plt.figure(figsize = (12, 6))
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()
```

[54]: []



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

I am intrested to know how many trips are created for different days of the month

[55]:
```
df2['trip_creation_day'].unique()
```

[55]:
```
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)
```

[56]:
```
df_day = df2.groupby(by = 'trip_creation_day')['trip_uuid'].count().to_frame().
  ↪reset_index()
df_day.head()
```

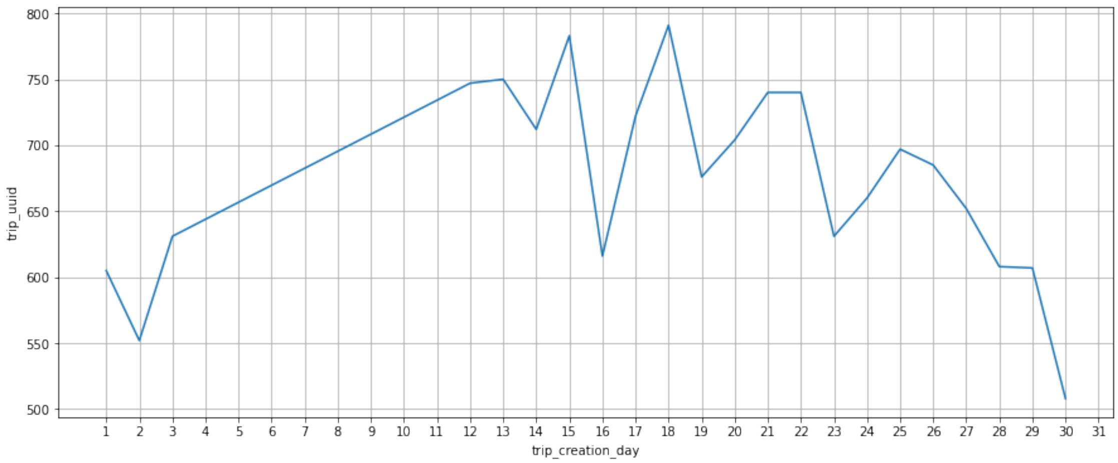
[56]:

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

[57]:
```
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()
```

[57]: []



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

I am intrested to know how many trips are created for different weeks

[58]:
```
df2['trip_creation_week'].unique()
```

[58]: array([37, 38, 39, 40], dtype=int8)

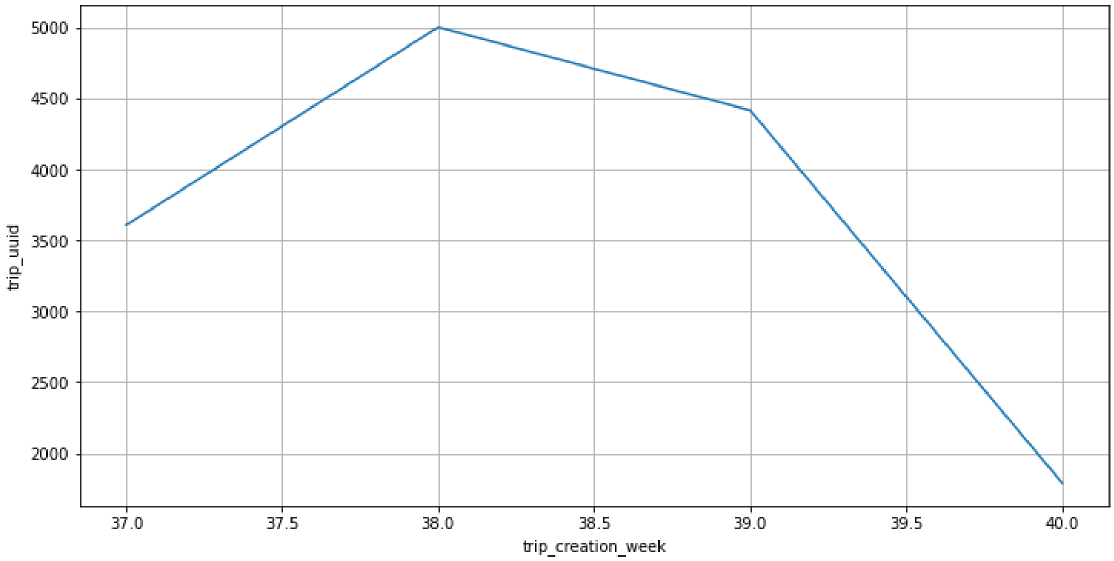
[59]:
```
df_week = df2.groupby(by = 'trip_creation_week')['trip_uuid'].count().
↪to_frame().reset_index()
df_week.head()
```

[59]:

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

[60]:
```
plt.figure(figsize = (12, 6))
sns.lineplot(data = df_week,
             x = df_week['trip_creation_week'],
             y = df_week['trip_uuid'],
             markers = 'o')
plt.grid('both')
plt.plot()
```

[60]: []



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

I am intrested to know how many trips are created in the given two months

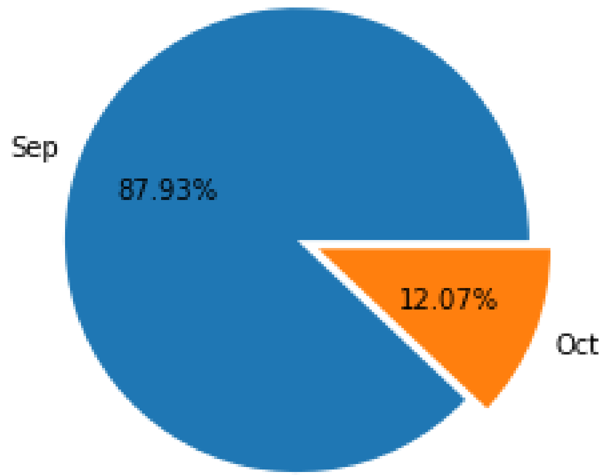
[61]:
```
df_month = df2.groupby(by = 'trip_creation_month')['trip_uuid'].count().
  ↪to_frame().reset_index()
df_month['perc'] = np.round(df_month['trip_uuid'] * 100/ df_month['trip_uuid'].
  ↪sum(), 2)
df_month.head()
```

[61]:

	trip_creation_month	trip_uuid	perc
0	9	13029	87.93
1	10	1788	12.07

[62]:
```
plt.pie(x = df_month['trip_uuid'],
        labels = ['Sep', 'Oct'],
        explode = [0, 0.1],
       autopct = '%.2f%%')
plt.plot()
```

[62]: []



I am interested to know the distribution of trip data for the orders

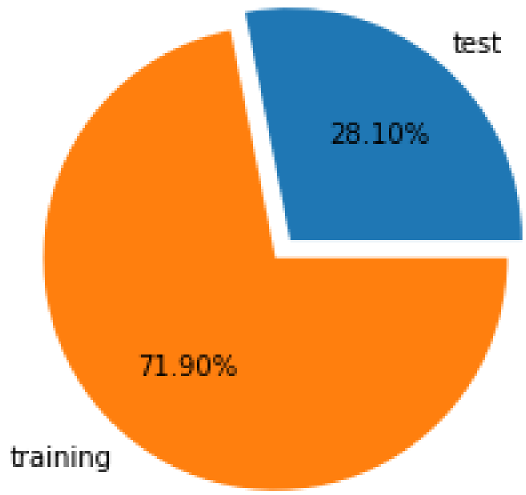
[63]:
```
df_data = df2.groupby(by = 'data')['trip_uuid'].count().to_frame().reset_index()
df_data['perc'] = np.round(df_data['trip_uuid'] * 100/ df_data['trip_uuid'].
    ↪sum(), 2)
df_data.head()
```

[63]:

	data	trip_uuid	perc
0	test	4163	28.1
1	training	10654	71.9

[64]:
```
plt.pie(x = df_data['trip_uuid'],
        labels = df_data['data'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

[64]: []



I am interested to know the distribution of route types for the orders

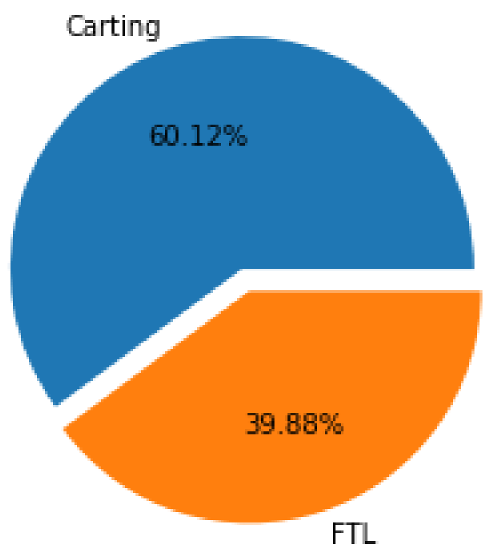
[65]:
```
df_route = df2.groupby(by = 'route_type')['trip_uuid'].count().to_frame().
    ↪reset_index()
df_route['perc'] = np.round(df_route['trip_uuid'] * 100/ df_route['trip_uuid'].
    ↪sum(), 2)
df_route.head()
```

[65]:

	route_type	trip_uuid	perc
0	Carting	8908	60.12
1	FTL	5909	39.88

[66]:
```
plt.pie(x = df_route['trip_uuid'],
        labels = ['Carting', 'FTL'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

[66]: []



I am interested to know what is the distribution of number of trips created from different states

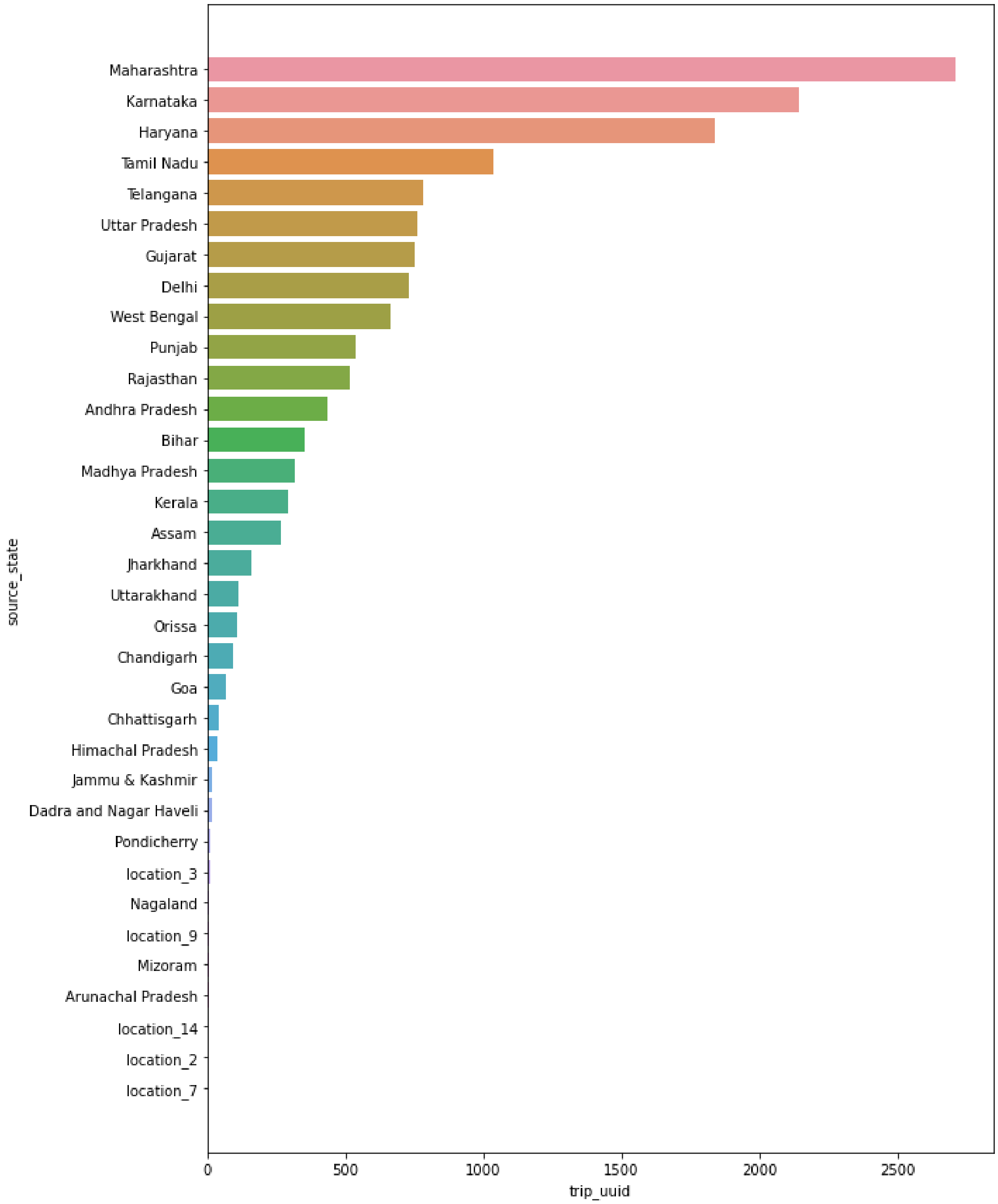
[67]:
```
df_source_state = df2.groupby(by = 'source_state')['trip_uuid'].count().
 ↪to_frame().reset_index()
df_source_state['perc'] = np.round(df_source_state['trip_uuid'] * 100/␣
 ↪df_source_state['trip_uuid'].sum(), 2)
df_source_state = df_source_state.sort_values(by = 'trip_uuid', ascending =␣
 ↪False)
df_source_state.head()
```

[67]:

	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

[68]:
```
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()
```

[68]: []



- It can be seen in the above plot that maximum trips originated from Maharashtra state followed by Karnataka and Haryana. That means that the seller base is strong in these states

I am interested to know top 30 cities based on the number of trips created from different cities

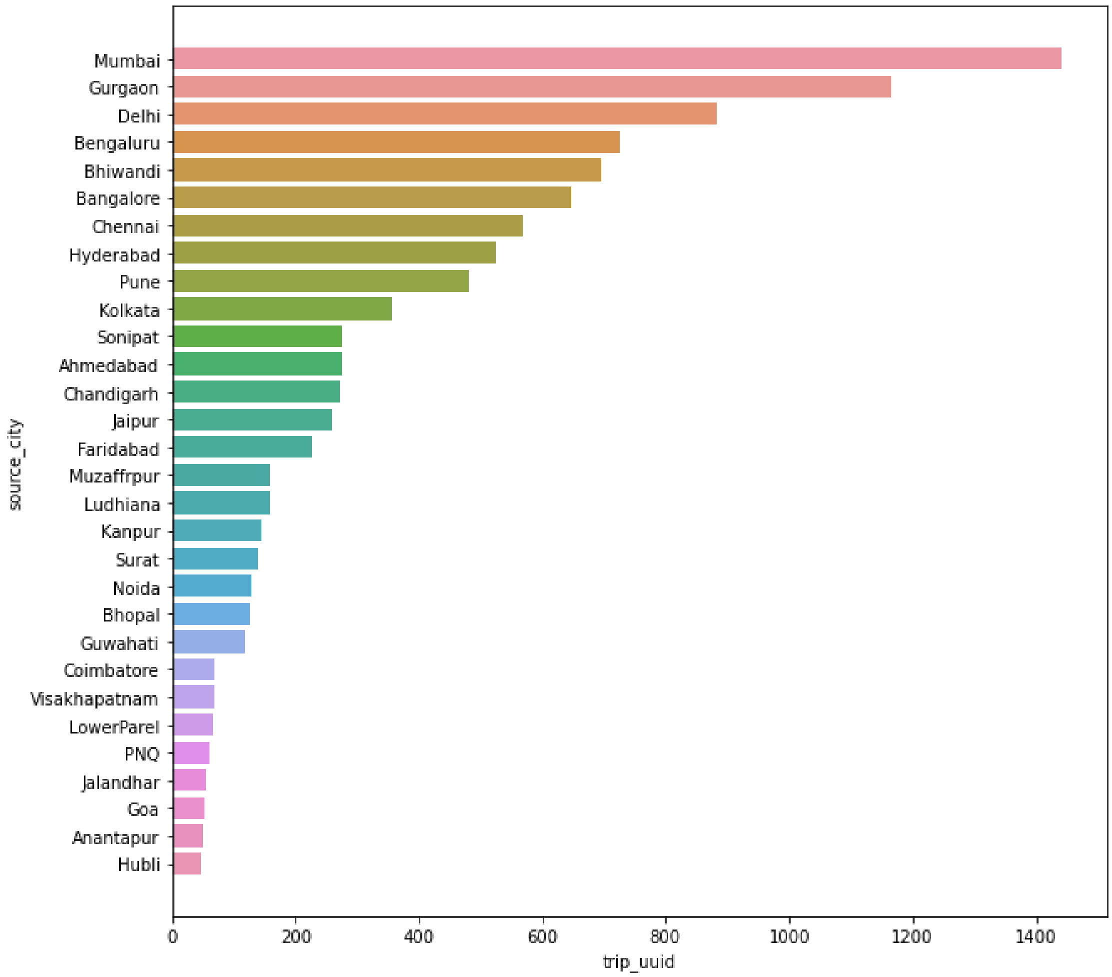
[69]:
```
df_source_city = df2.groupby(by = 'source_city')['trip_uuid'].count().
    ↪to_frame().reset_index()
df_source_city['perc'] = np.round(df_source_city['trip_uuid'] * 100/␣
    ↪df_source_city['trip_uuid'].sum(), 2)
df_source_city = df_source_city.sort_values(by = 'trip_uuid', ascending =␣
    ↪False)[:30]
df_source_city
```

[69]:

	source_city	trip_uuid	perc
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

[70]:
```
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()
```

[70]: []



- It can be seen in the above plot that maximum trips originated from Mumbai city followed by Gurgaon Delhi, Bengaluru and Bhiwandi. That means that the seller base is strong in these cities.

I am interested to know what is the distribution of number of trips which ended in different states

[71]:
```
df_destination_state = df2.groupby(by = 'destination_state')['trip_uuid'].
    ↪count().to_frame().reset_index()
df_destination_state['perc'] = np.round(df_destination_state['trip_uuid'] * 100/
    ↪ df_destination_state['trip_uuid'].sum(), 2)
df_destination_state = df_destination_state.sort_values(by = 'trip_uuid',␣
    ↪ascending = False)
df_destination_state.head()
```

[71]:

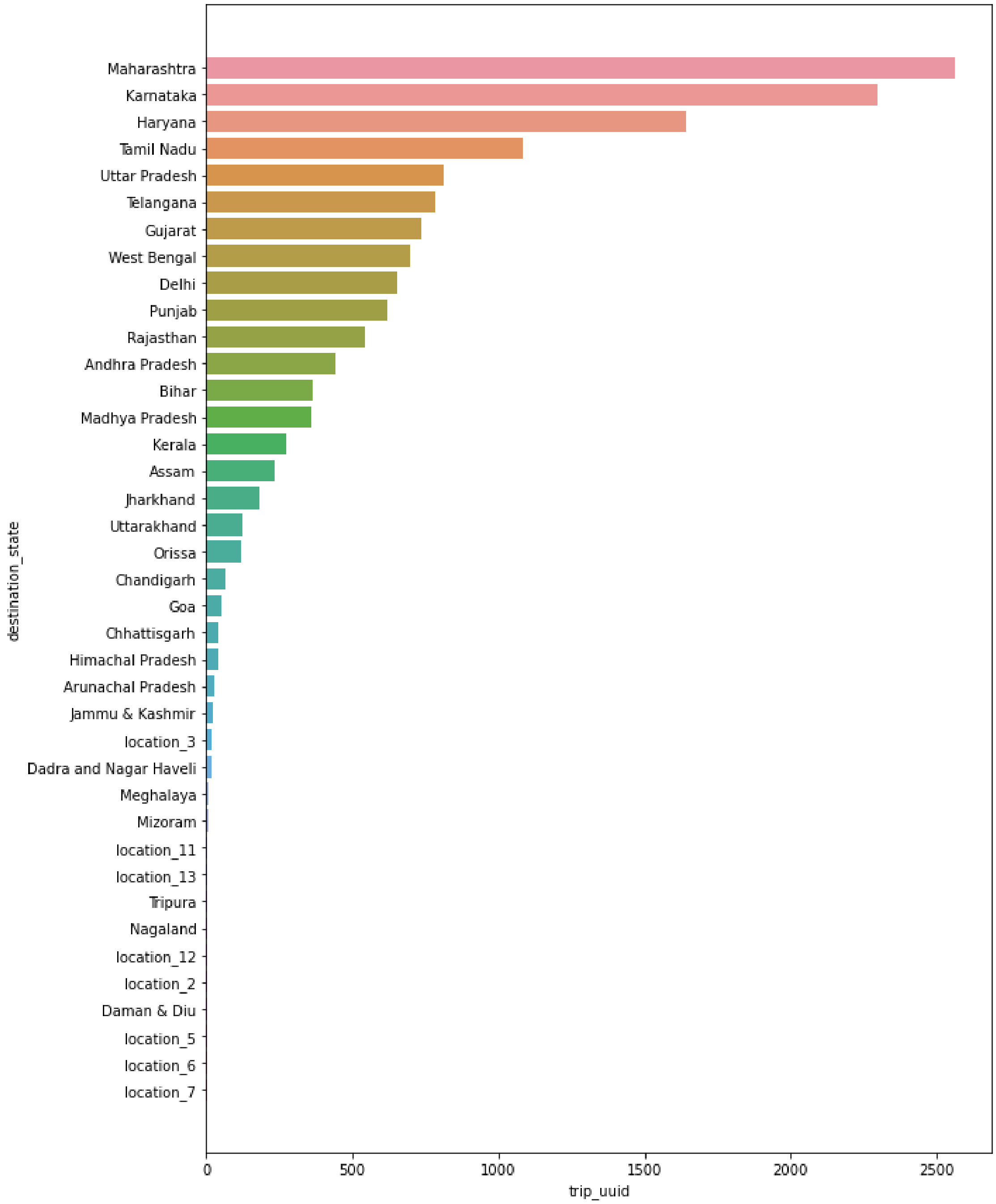
	destination_state	trip_uuid	perc
18	Maharashtra	2561	17.28

```
15         Karnataka       2294  15.48
11           Haryana       1643  11.09
25        Tamil Nadu       1084   7.32
28     Uttar Pradesh        811   5.47
```

[72]:

```
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()
```

[72]: []



- It can be seen in the above plot that maximum 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 in these states.

I am interested to know top 30 cities based on the number of trips ended in different cities

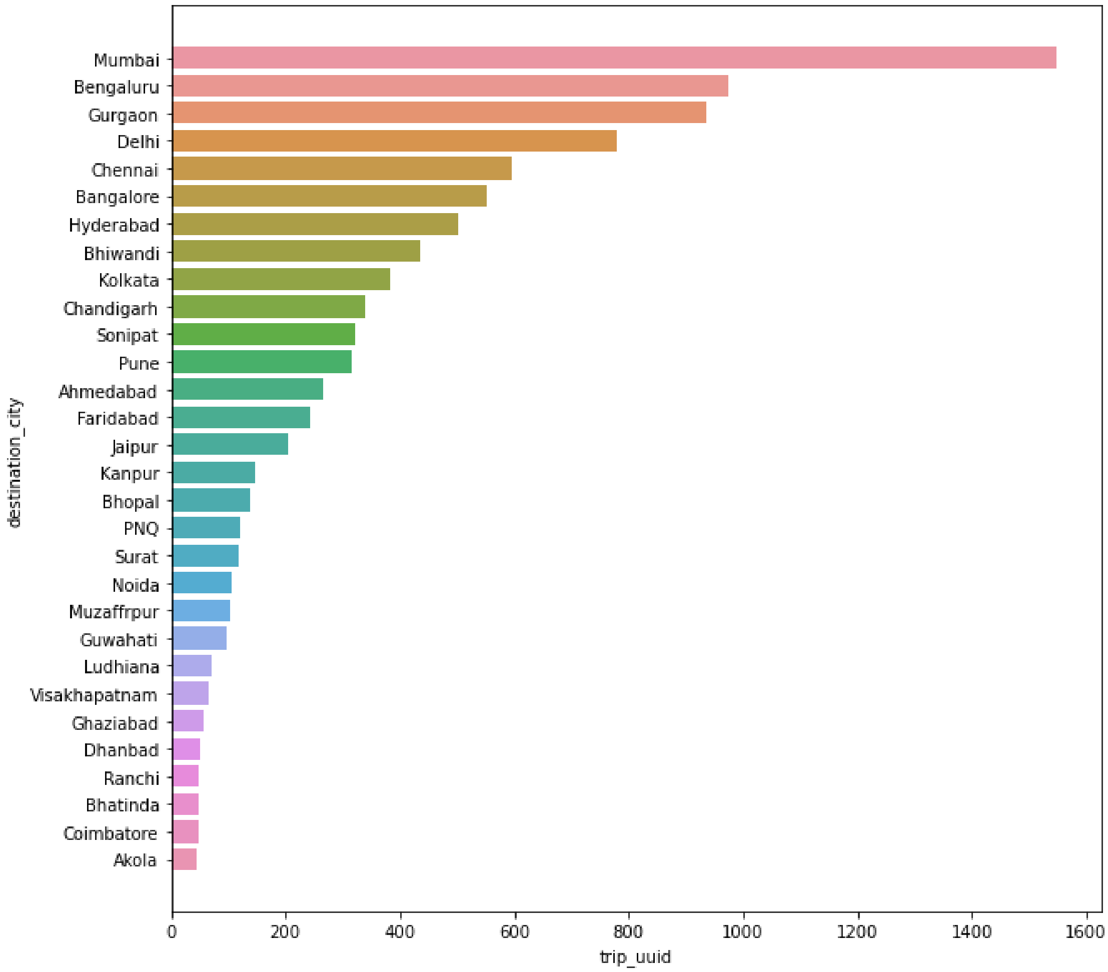
[73]:
```python
df_destination_city = df2.groupby(by = 'destination_city')['trip_uuid'].count().
    ↪to_frame().reset_index()
df_destination_city['perc'] = np.round(df_destination_city['trip_uuid'] * 100/
    ↪df_destination_city['trip_uuid'].sum(), 2)
df_destination_city = df_destination_city.sort_values(by = 'trip_uuid',
    ↪ascending = False)[:30]
df_destination_city
```

[73]:

	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

[74]:
```python
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()
```

[74]: []



- It can be seen in the above plot that maximum 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.

[75]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan',␣
 ↪'actual_distance_to_destination',
                     'actual_time', 'osrm_time', 'osrm_distance',␣
 ↪'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']
sns.pairplot(data = df2,
             vars = numerical_columns,
             kind = 'reg',
             hue = 'route_type',
             markers = '.')
plt.plot()
```

[75]: []

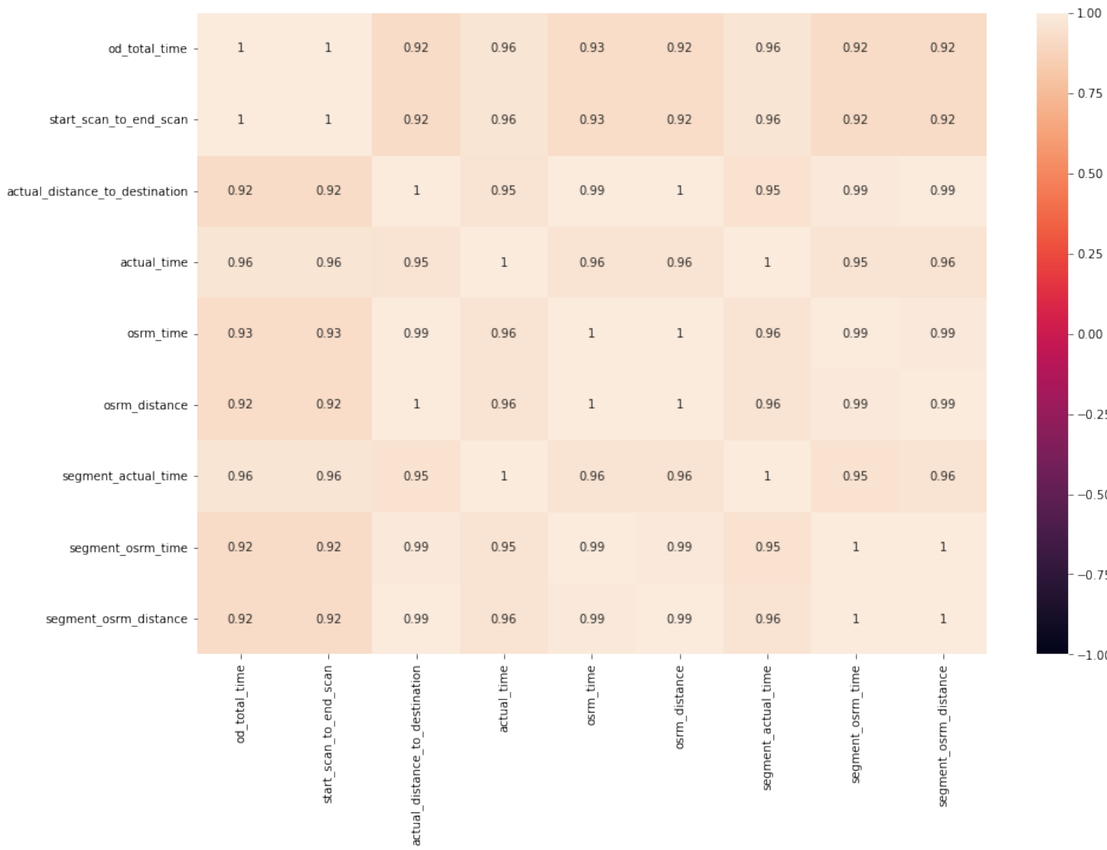
[76]:
```
df_corr = df2[numerical_columns].corr()
df_corr
```

[76]:
```
                                od_total_time  start_scan_to_end_scan  \
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
```

```
                                 actual_distance_to_destination  actual_time  \
od_total_time                                          0.918222     0.961094   
start_scan_to_end_scan                                 0.918308     0.961147   
actual_distance_to_destination                         1.000000     0.953757   
actual_time                                            0.953757     1.000000   
osrm_time                                              0.993561     0.958593   
osrm_distance                                          0.997264     0.959214   
segment_actual_time                                    0.952821     0.999989   
segment_osrm_time                                      0.987538     0.953872   
segment_osrm_distance                                  0.993061     0.956967   

                                 osrm_time  osrm_distance  segment_actual_time  \
od_total_time                     0.926516       0.924219             0.961119   
start_scan_to_end_scan            0.926571       0.924299             0.961171   
actual_distance_to_destination    0.993561       0.997264             0.952821   
actual_time                       0.958593       0.959214             0.999989   
osrm_time                         1.000000       0.997580             0.957765   
osrm_distance                     0.997580       1.000000             0.958353   
segment_actual_time               0.957765       0.958353             1.000000   
segment_osrm_time                 0.993259       0.991798             0.953039   
segment_osrm_distance             0.991608       0.994710             0.956106   

                                 segment_osrm_time  segment_osrm_distance  
od_total_time                             0.918490               0.919199  
start_scan_to_end_scan                    0.918561               0.919291  
actual_distance_to_destination            0.987538               0.993061  
actual_time                               0.953872               0.956967  
osrm_time                                 0.993259               0.991608  
osrm_distance                             0.991798               0.994710  
segment_actual_time                       0.953039               0.956106  
segment_osrm_time                         1.000000               0.996092  
segment_osrm_distance                     0.996092               1.000000  
```

[77]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, vmin = -1, vmax = 1, annot = True)
plt.plot()
```

[77]: []



- Very High Correlation (> 0.9) exists between columns all the numerical columns specified above

1.5 3. In-depth analysis and feature engineering:

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

STEP-1 : 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.

STEP-2 : Checking for basic assumpitons for the hypothesis

- 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.

STEP-4: Compute the p-value and fix value of alpha.

- Based on p-value, we will accept or reject H0.

  1. **p-val > alpha** : Accept H0
  2. **p-val < alpha** : Reject H0

[78]:
```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```

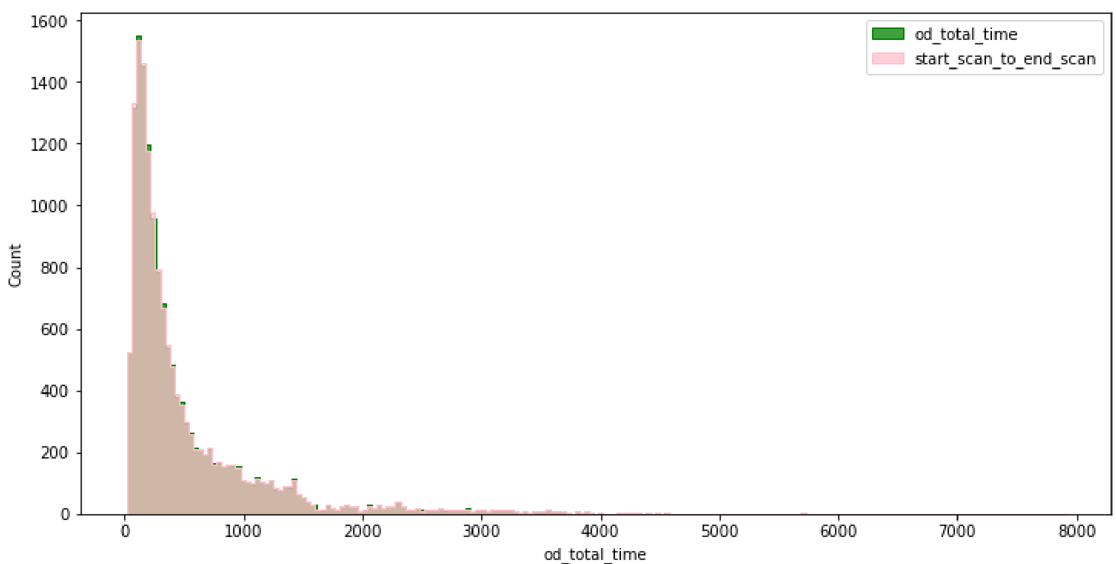
[78]:

	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

- Visual Tests to know if the samples follow normal distribution

[79]:
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['od_total_time'], element = 'step', color = 'green')
sns.histplot(df2['start_scan_to_end_scan'], element = 'step', color = 'pink')
plt.legend(['od_total_time', 'start_scan_to_end_scan'])
plt.plot()
```

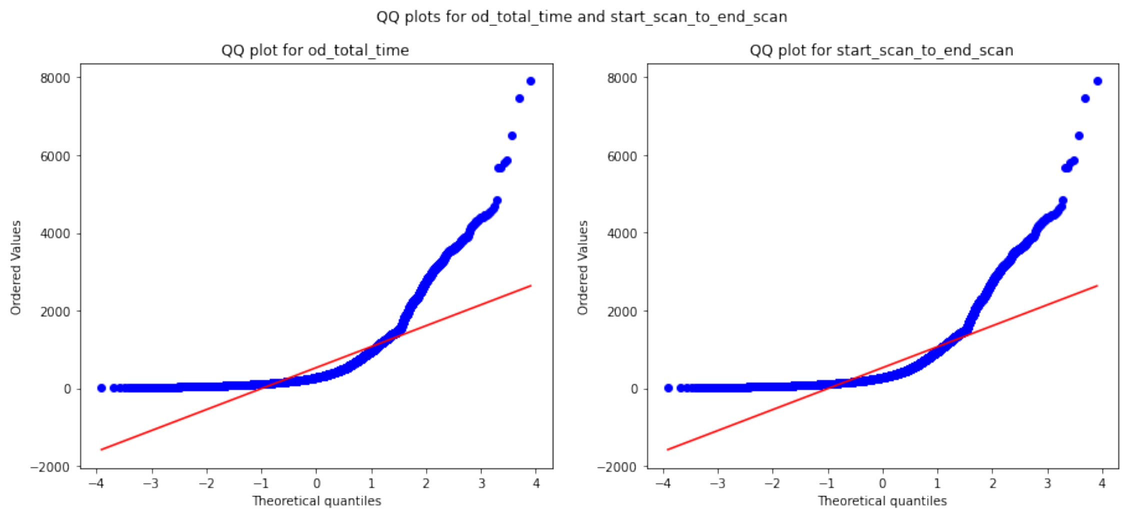
[79]: []



- Distribution check using **QQ Plot**

[80]:
```
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()
```

[80]: []



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

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

[81]:
```
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
```

[82]:
```
test_stat, p_value = spy.shapiro(df2['start_scan_to_end_scan'].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
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

[83]:
```
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 7.21300687930395e-25
The sample does not follow normal distribution
```

[84]:
```
transformed_start_scan_to_end_scan = spy.
 ↪boxcox(df2['start_scan_to_end_scan'])[0]
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 1.0378319150112312e-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.
- Homogeneity of Variances using **Lavene's test**

[85]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['od_total_time'],␣
 ↪df2['start_scan_to_end_scan'])
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.

[86]:
```
test_stat, p_value = spy.mannwhitneyu(df2['od_total_time'],␣
 ↪df2['start_scan_to_end_scan'])
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.

1.5.2 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)

[87]:
```
df2[['actual_time', 'osrm_time']].describe()
```

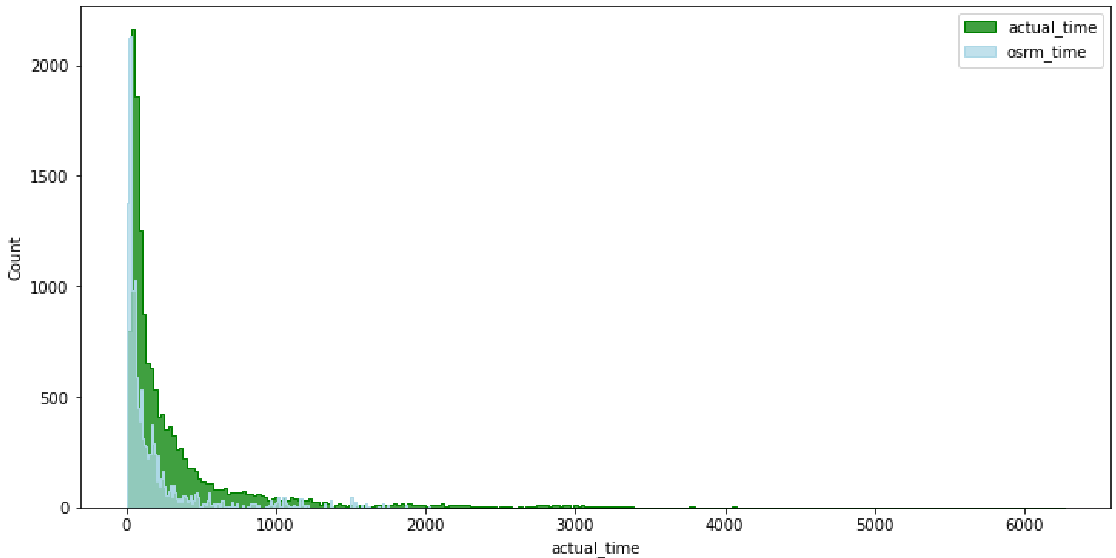
[87]:

	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

- Visual Tests to know if the samples follow normal distribution

[88]:
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'green')
sns.histplot(df2['osrm_time'], element = 'step', color = 'lightblue')
plt.legend(['actual_time', 'osrm_time'])
plt.plot()
```

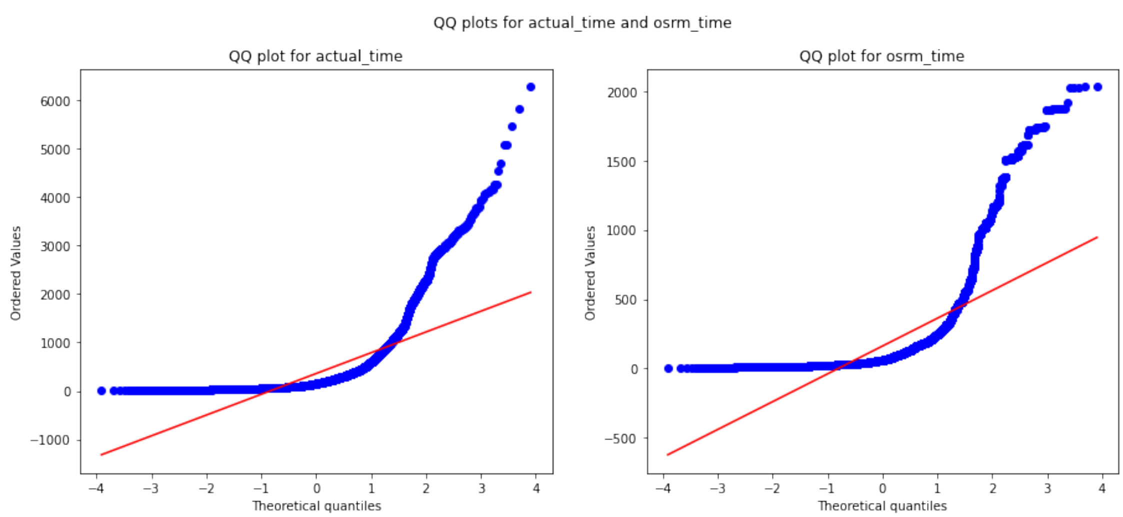
[88]: []



- Distribution check using **QQ Plot**

[89]:
```
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()
```

[89]: []



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

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

[90]:
```
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
```

[91]:
```
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
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

[92]:
```
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 1.0408425976485893e-28
The sample does not follow normal distribution
```

[93]:
```
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 3.271205914895016e-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.
- Homogeneity of Variances using **Lavene's test**

[94]:
```
# Null Hypothesis(HO) - 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.

[95]:
```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'], df2['osrm_time'])
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.

1.5.3 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)

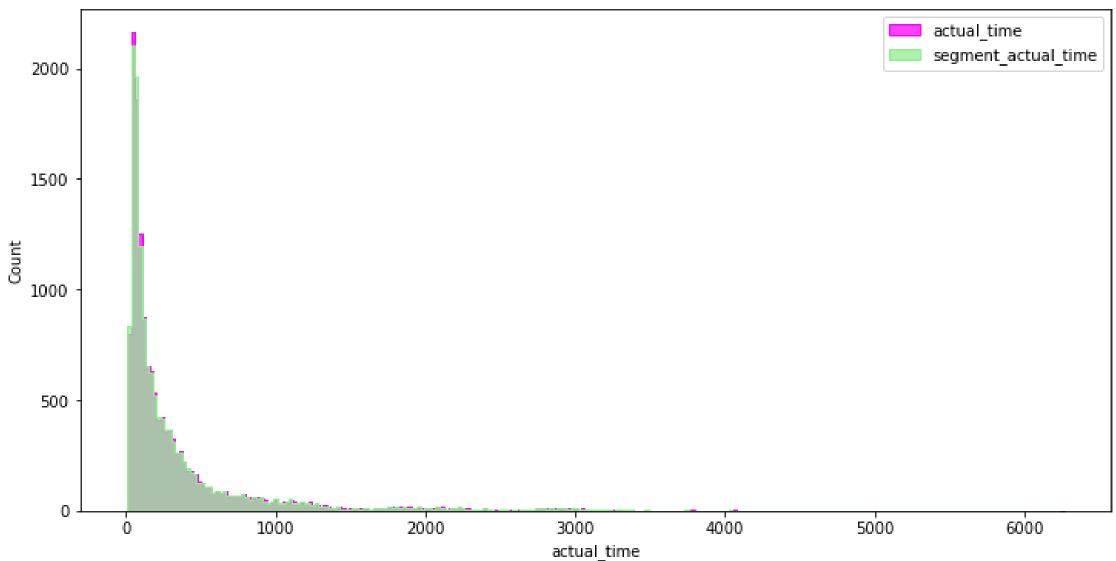
[96]:
```
df2[['actual_time', 'segment_actual_time']].describe()
```

[96]:
```
       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
```

- Visual Tests to know if the samples follow normal distribution

[97]:
```python
plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'magenta')
sns.histplot(df2['segment_actual_time'], element = 'step', color = 'lightgreen')
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```

[97]: []

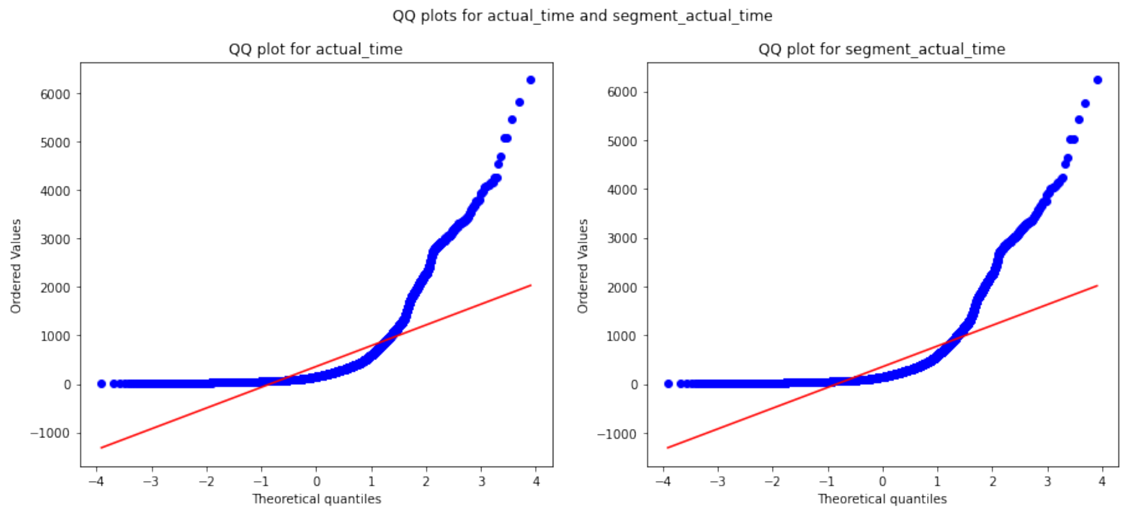


- Distribution check using **QQ Plot**

[98]:
```python
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()
```

[98]: []



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

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

[99]:
```
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
```

[100]:
```
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
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

[101]:
```
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 1.0408425976485893e-28
The sample does not follow normal distribution
```

[102]:
```
transformed_segment_actual_time = spy.boxcox(df2['segment_actual_time'])[0]
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 5.676203648979465e-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.
- Homogeneity of Variances using **Lavene's test**

[103]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'], df2['segment_actual_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 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.

[104]:
```
test_stat, p_value = spy.mannwhitneyu(df2['actual_time'],␣
 ↪df2['segment_actual_time'])
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.

1.5.4 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)

[105]:
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

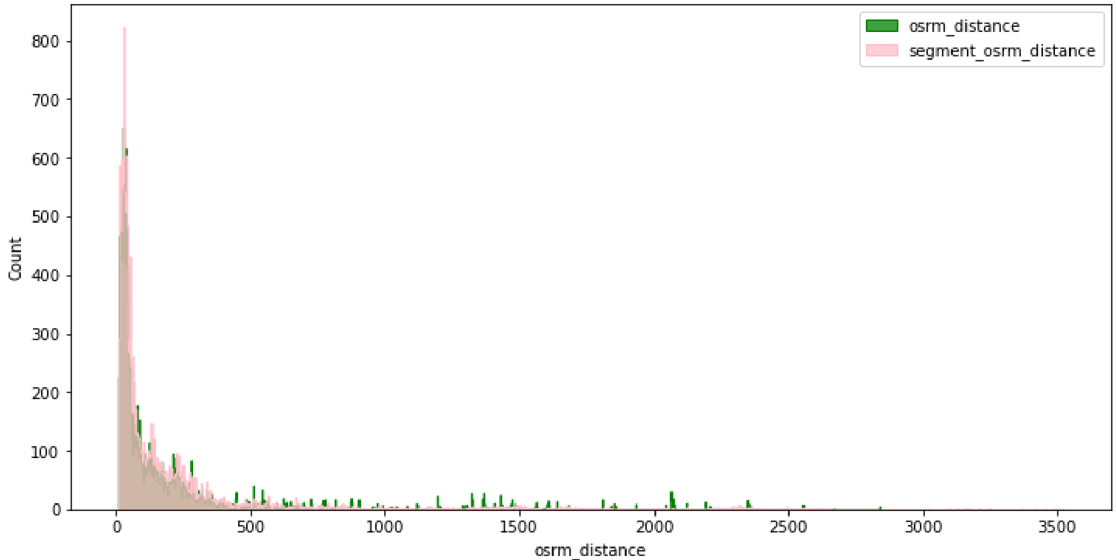
[105]:

	osrm_distance	segment_osrm_distance
count	14817.000000	14817.000000
mean	204.345078	223.201324
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

- Visual Tests to know if the samples follow normal distribution

[106]:
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_distance'], element = 'step', color = 'green', bins =␣
 ↪1000)
sns.histplot(df2['segment_osrm_distance'], element = 'step', color = 'pink',␣
 ↪bins = 1000)
plt.legend(['osrm_distance', 'segment_osrm_distance'])
plt.plot()
```

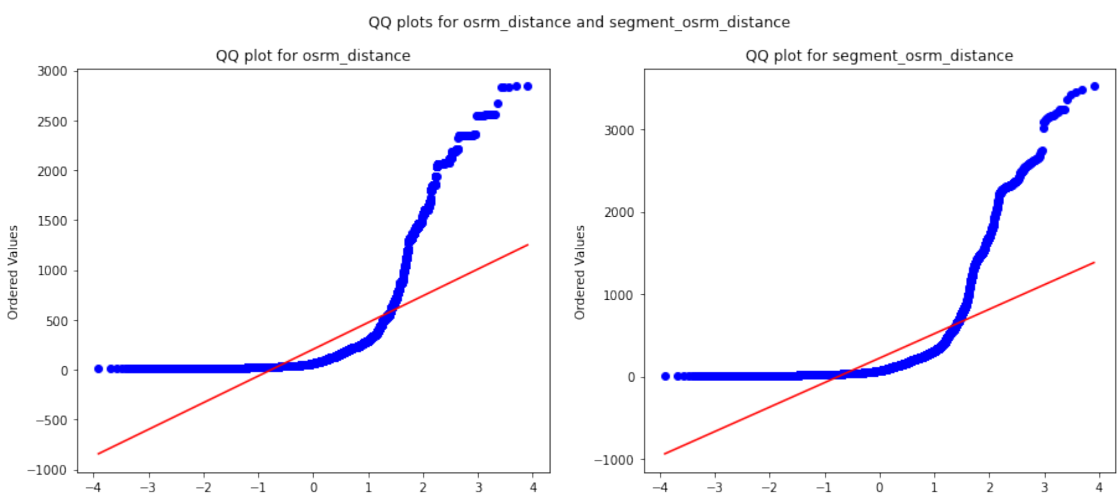
[106]: []



- Distribution check using **QQ Plot**

[107]:
```
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()
```

[107]: []



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

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

[108]:
```
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
```

[109]:
```
test_stat, p_value = spy.shapiro(df2['segment_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
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

[110]:
```
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 7.069971142058e-41
The sample does not follow normal distribution
```

[111]:
```
transformed_segment_osrm_distance = spy.boxcox(df2['segment_osrm_distance'])[0]
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 3.0555416710688996e-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**

[112]:
```
# Null Hypothesis(HO) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['osrm_distance'],␣
 ↪df2['segment_osrm_distance'])
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.

[113]:
```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_distance'],␣
 ↪df2['segment_osrm_distance'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 9.509312191161966e-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.

1.5.5 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)

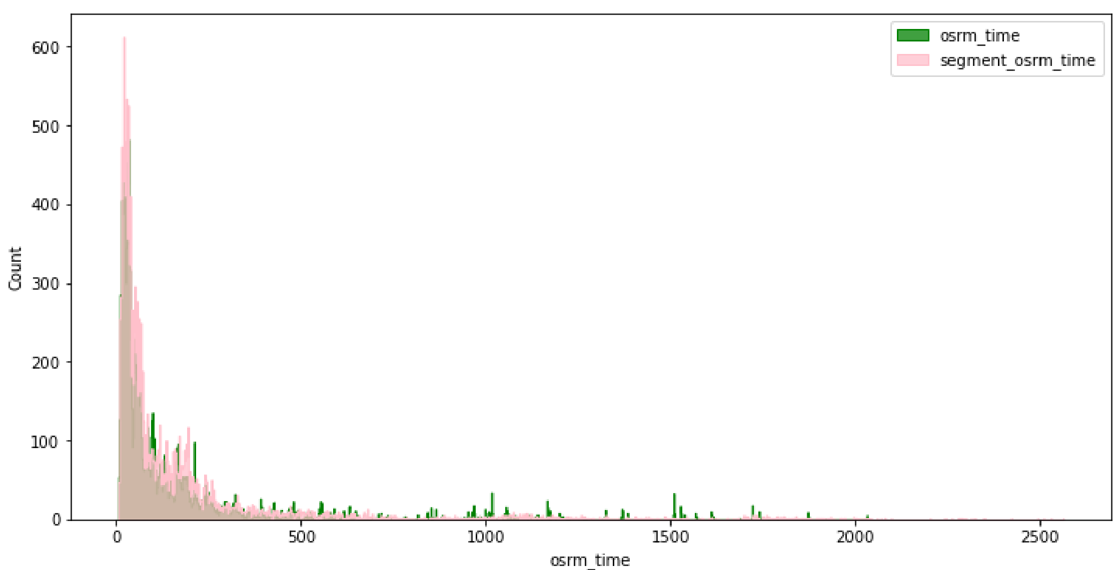
[114]:
```
df2[['osrm_time', 'segment_osrm_time']].describe().T
```

[114]:
```
                     count        mean         std  min   25%   50%    75%  \
osrm_time          14817.0  161.384018  271.362549  6.0  29.0  60.0  168.0
segment_osrm_time  14817.0  180.949783  314.541412  6.0  31.0  65.0  185.0

                      max
osrm_time          2032.0
segment_osrm_time  2564.0
```

- Visual Tests to know if the samples follow normal distribution

[115]:
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['osrm_time'], element = 'step', color = 'green', bins = 1000)
sns.histplot(df2['segment_osrm_time'], element = 'step', color = 'pink', bins =␣
↪1000)
plt.legend(['osrm_time', 'segment_osrm_time'])
plt.plot()
```

[115]: []

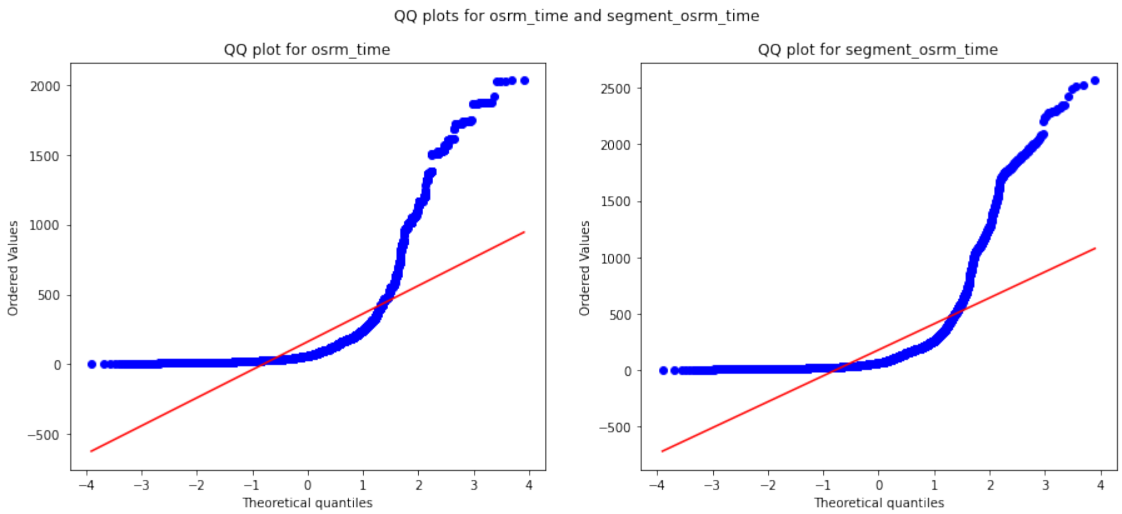


- Distribution check using **QQ Plot**

[116]:
```
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()
```

[116]: []



It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

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

[117]:
```
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
```

[118]:
```
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
```

- Transforming the data using boxcox transformation to check if the transformed data follows normal distribution.

[119]:
```
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 3.271205914895016e-35
The sample does not follow normal distribution
```

[120]:
```
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 4.960995746782918e-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**

[121]:
```
# Null Hypothesis(HO) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['osrm_time'], df2['segment_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 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.

[122]:
```
test_stat, p_value = spy.mannwhitneyu(df2['osrm_time'],␣
 ↪df2['segment_osrm_time'])
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
```

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

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

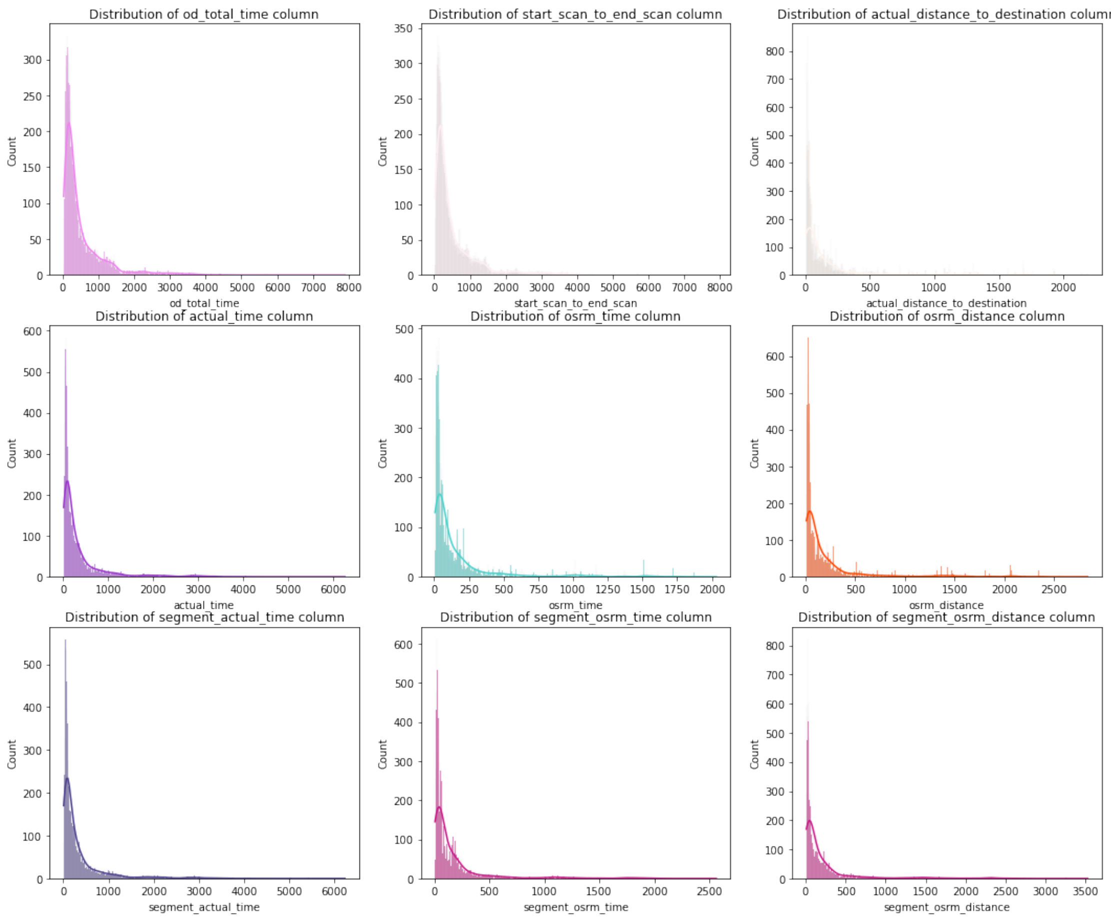
[123]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan',␣
 ↪'actual_distance_to_destination',
                     'actual_time', 'osrm_time', 'osrm_distance',␣
 ↪'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']
df2[numerical_columns].describe().T
```

[123]:
```
                                  count        mean         std        min  \
od_total_time                   14817.0  531.697630  658.868223  23.460000
start_scan_to_end_scan          14817.0  530.810016  658.705957  23.000000
actual_distance_to_destination  14817.0  164.477951  305.388123   9.002461
actual_time                     14817.0  357.143768  561.395020   9.000000
osrm_time                       14817.0  161.384018  271.362549   6.000000
osrm_distance                   14817.0  204.345078  370.395508   9.072900
segment_actual_time             14817.0  353.892273  556.246826   9.000000
segment_osrm_time               14817.0  180.949783  314.541412   6.000000
segment_osrm_distance           14817.0  223.201324  416.628326   9.072900

                                       25%         50%         75%  \
od_total_time                   149.930000  280.770000  638.200000
start_scan_to_end_scan          149.000000  280.000000  637.000000
```

```
actual_distance_to_destination   22.837238   48.474072  164.583206
actual_time                      67.000000  149.000000  370.000000
osrm_time                        29.000000   60.000000  168.000000
osrm_distance                    30.819201   65.618805  208.475006
segment_actual_time              66.000000  147.000000  367.000000
segment_osrm_time                31.000000   65.000000  185.000000
segment_osrm_distance            32.654499   70.154404  218.802399

                                        max
od_total_time                   7898.550000
start_scan_to_end_scan          7898.000000
actual_distance_to_destination  2186.531738
actual_time                     6265.000000
osrm_time                       2032.000000
osrm_distance                   2840.081055
segment_actual_time             6230.000000
segment_osrm_time               2564.000000
segment_osrm_distance           3523.632324
```

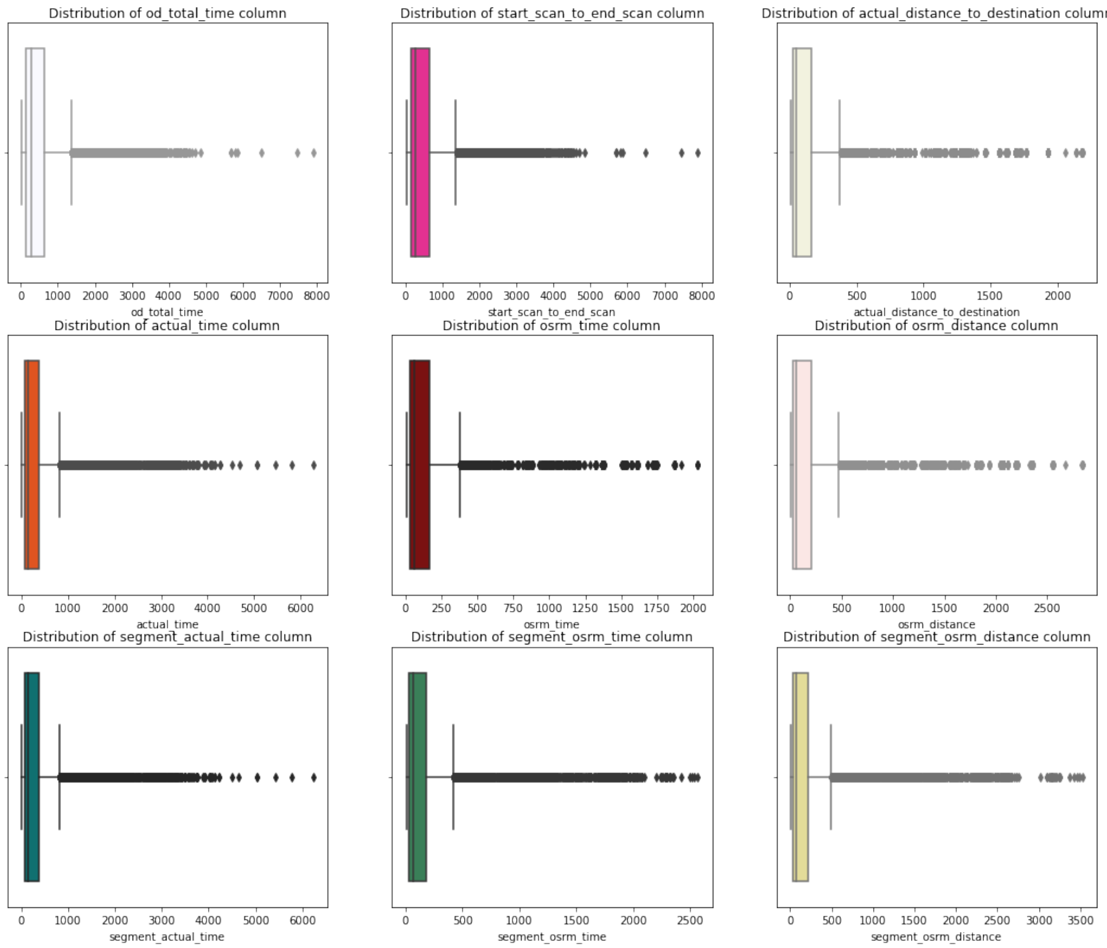
[124]:

```python
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, color = clr)
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



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

[125]:
```
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()
```



- It can be clearly seen in the above plots that there are outliers in all the numerical columns that need to be treated.

[126]:
```
# 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}')
```

```python
    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.

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

[127]:
```
# Get value counts before one-hot encoding

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

[127]:
```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

[128]:
```
# Perform one-hot encoding on categorical column route type
```

[129]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df2['route_type'] = label_encoder.fit_transform(df2['route_type'])
```

[130]:
```python
# Get value counts after one-hot encoding

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

[130]:
```
0    8908
1    5909
Name: route_type, dtype: int64
```

[131]:
```python
# Get value counts of categorical variable 'data' before one-hot encoding

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

[131]:
```
training    10654
test         4163
Name: data, dtype: int64
```

[132]:
```python
# Perform one-hot encoding on categorical variable 'data'
```

[133]:
```python
label_encoder = LabelEncoder()
df2['data'] = label_encoder.fit_transform(df2['data'])
```

[134]:
```python
# Get value counts after one-hot encoding

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

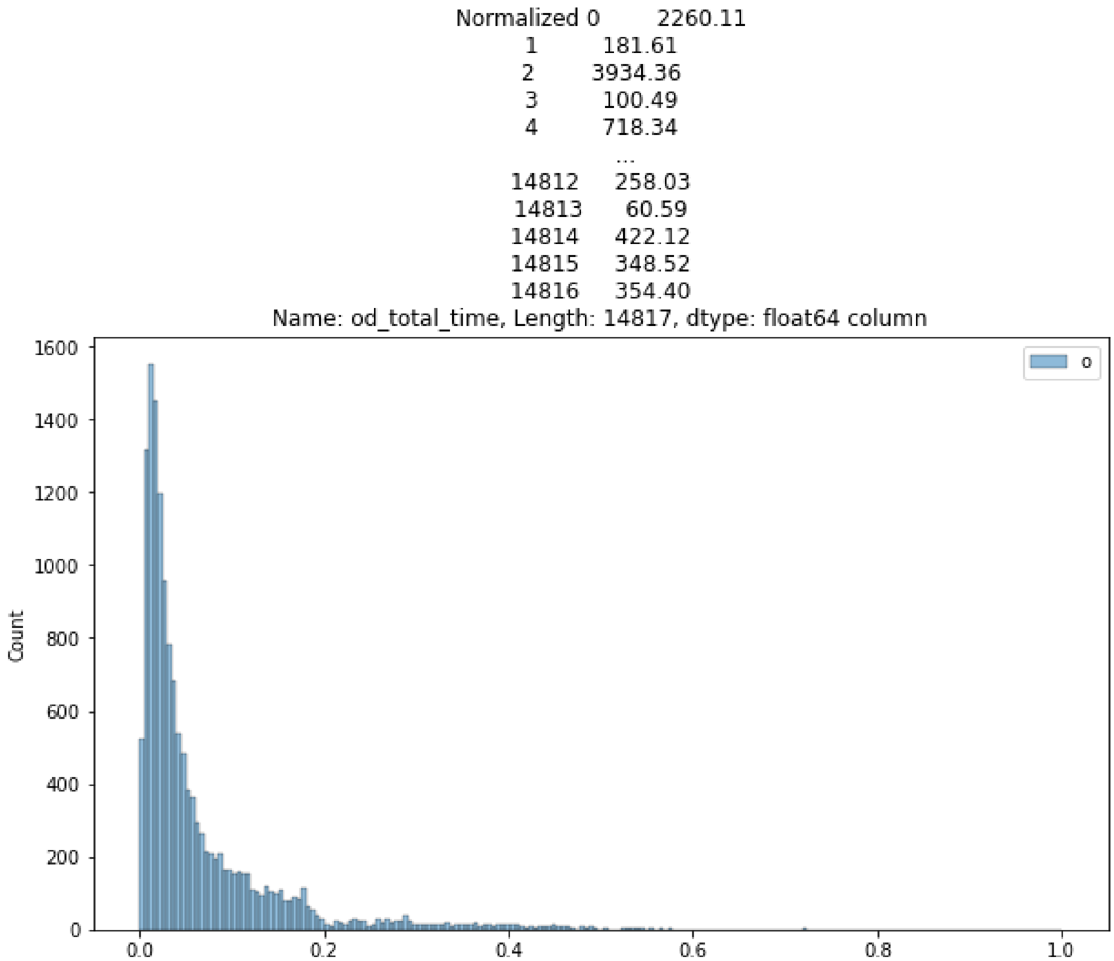
[134]:
```
1    10654
0     4163
Name: data, dtype: int64
```

1.5.8 Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

[135]:
```python
from sklearn.preprocessing import MinMaxScaler
```

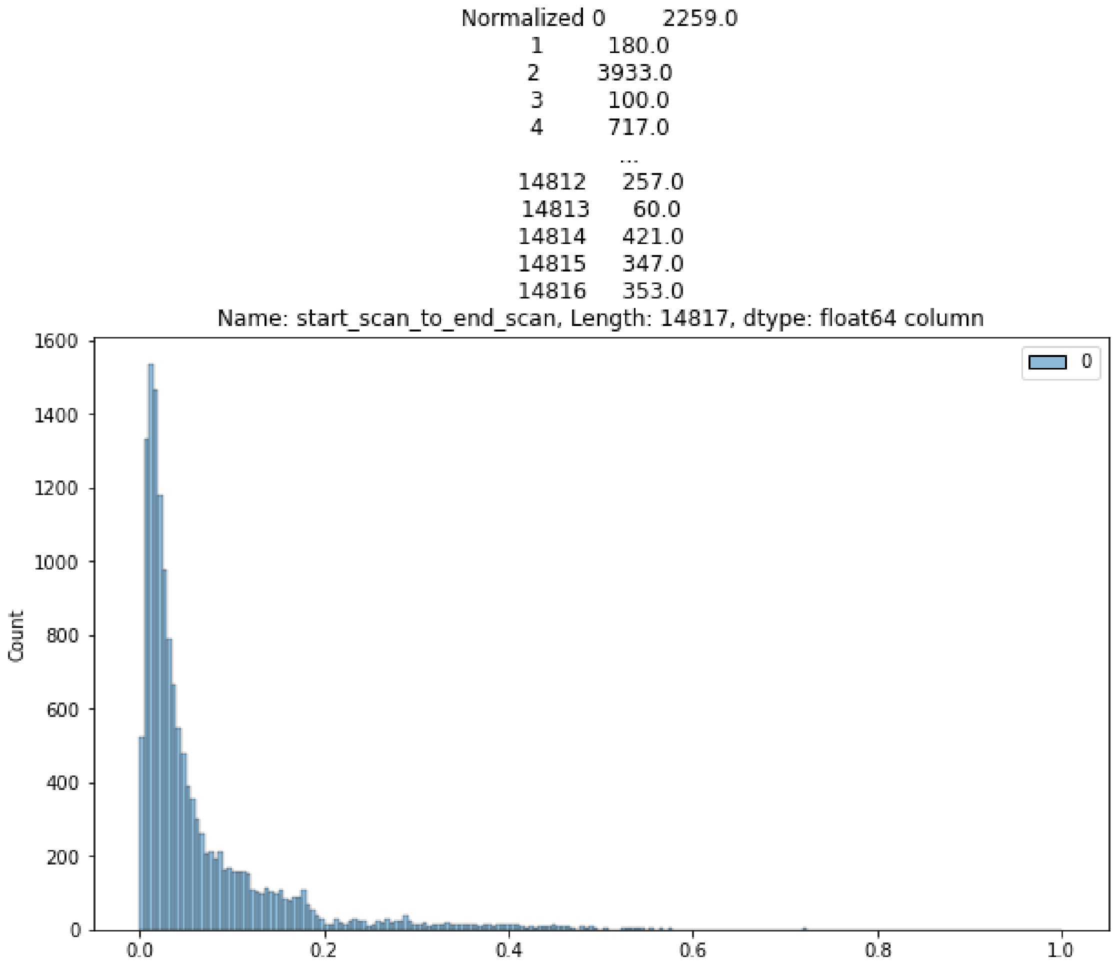
[136]:
```python
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['od_total_time']} column")
plt.legend('od_total_time')
plt.plot()
```

[136]:
```
[]
```



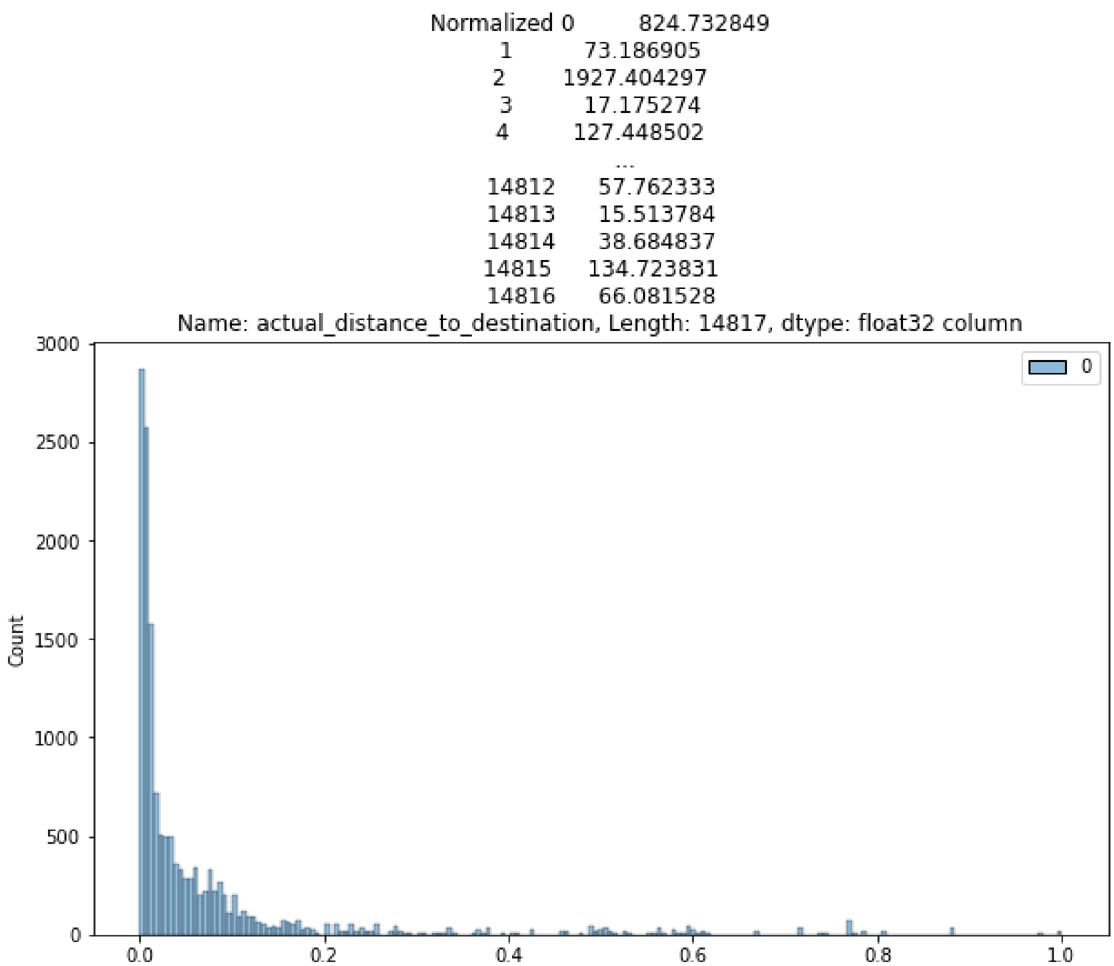
[137]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().
 ↪reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

[137]: []



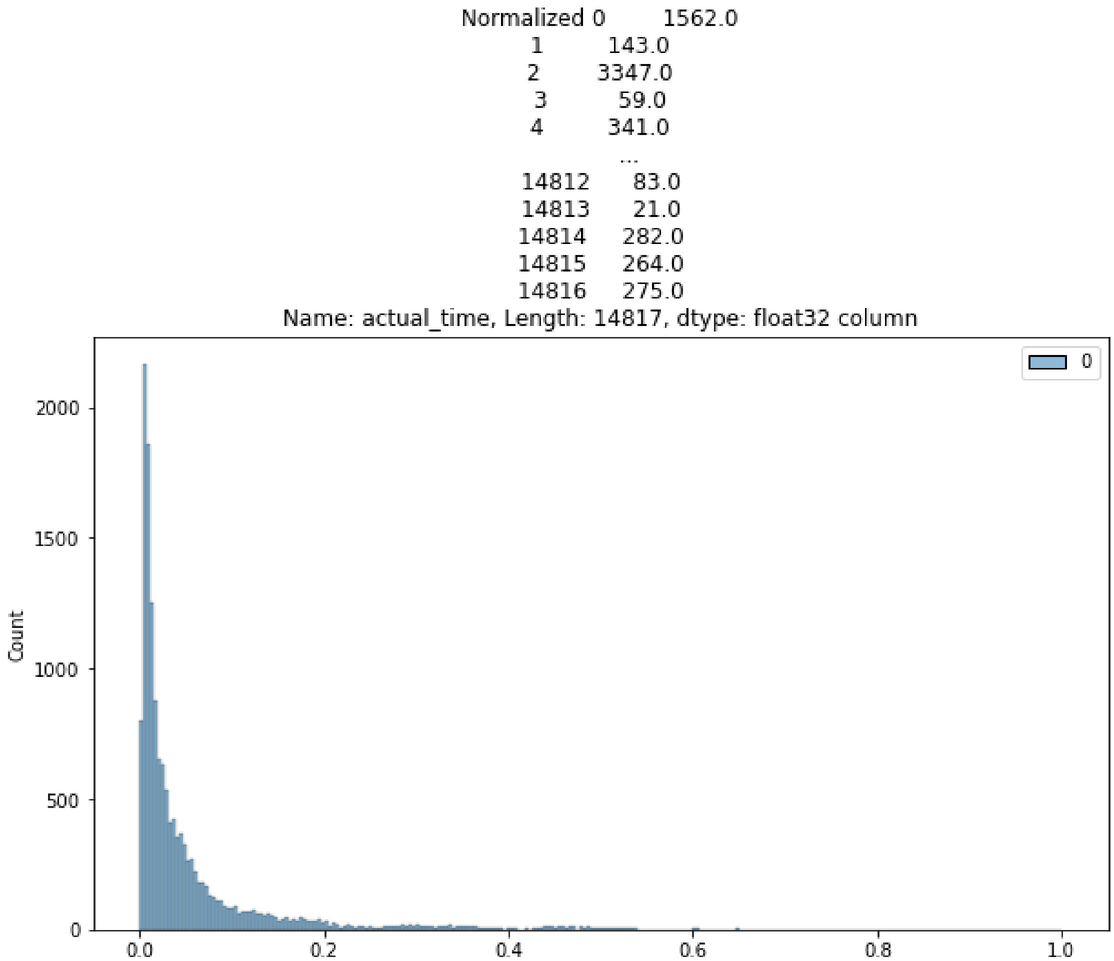
[138]:
```python
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().
 ↪reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_distance_to_destination']} column")
plt.plot()
```

[138]: []




```
[139]: plt.figure(figsize = (10, 6))
       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()
```

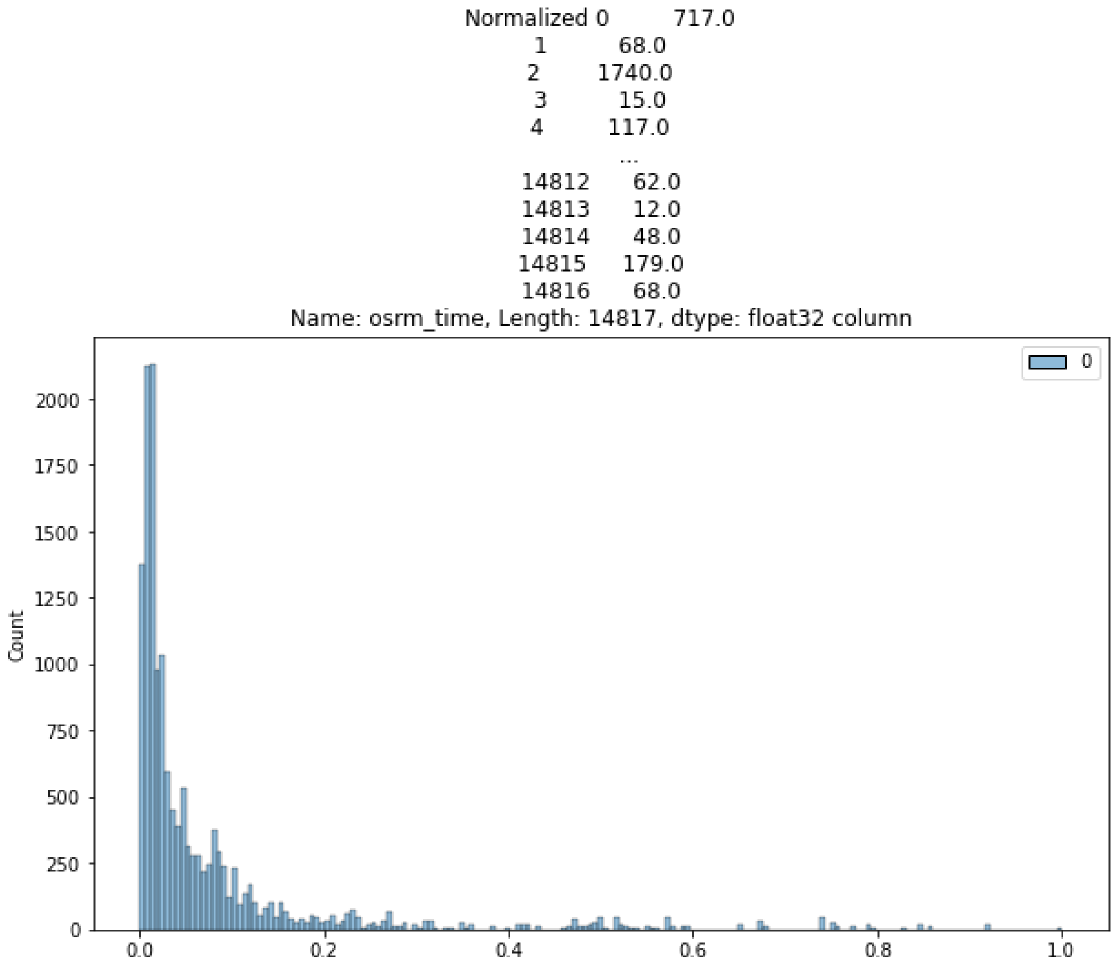
```
[139]: []
```



[ ]:

[140]:
```
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()
```

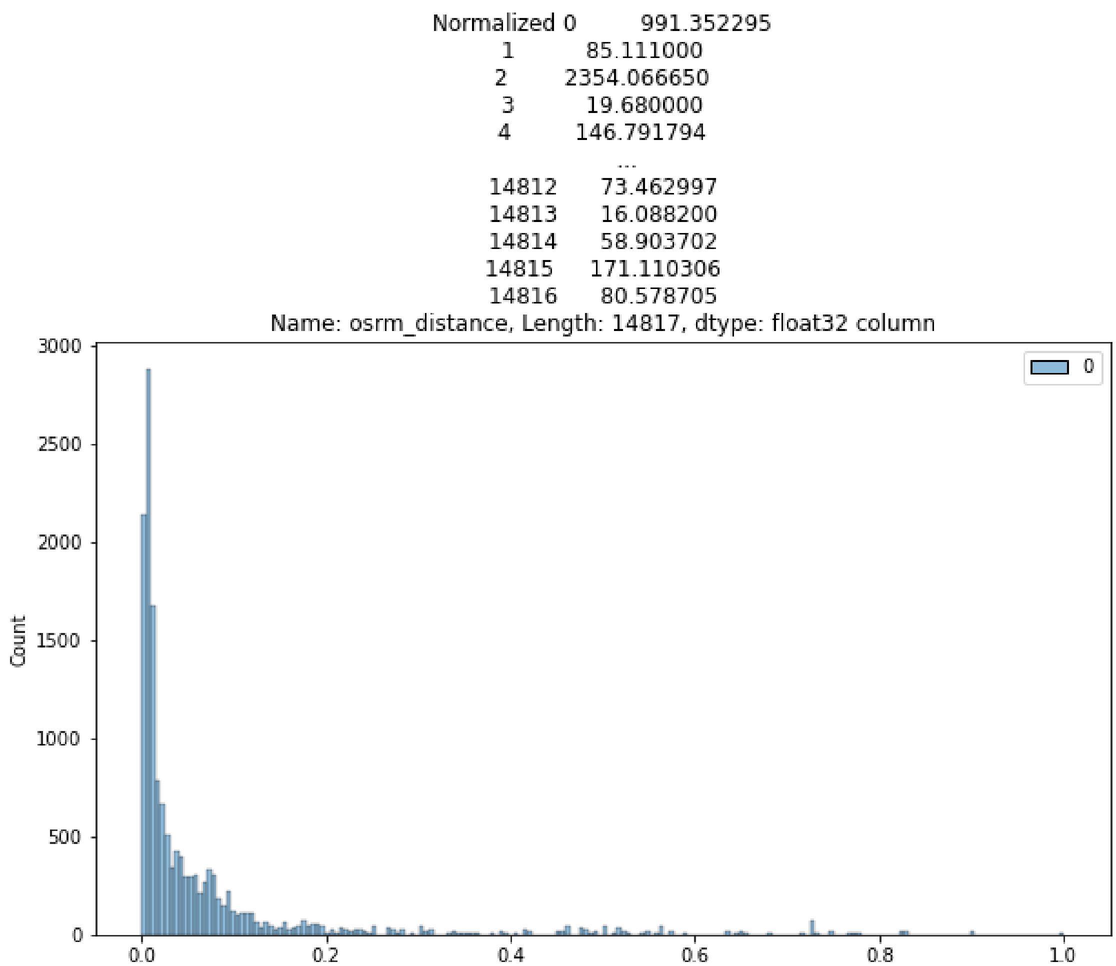
[140]: []



[ ]:

[141]:
```python
plt.figure(figsize = (10, 6))
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()
```

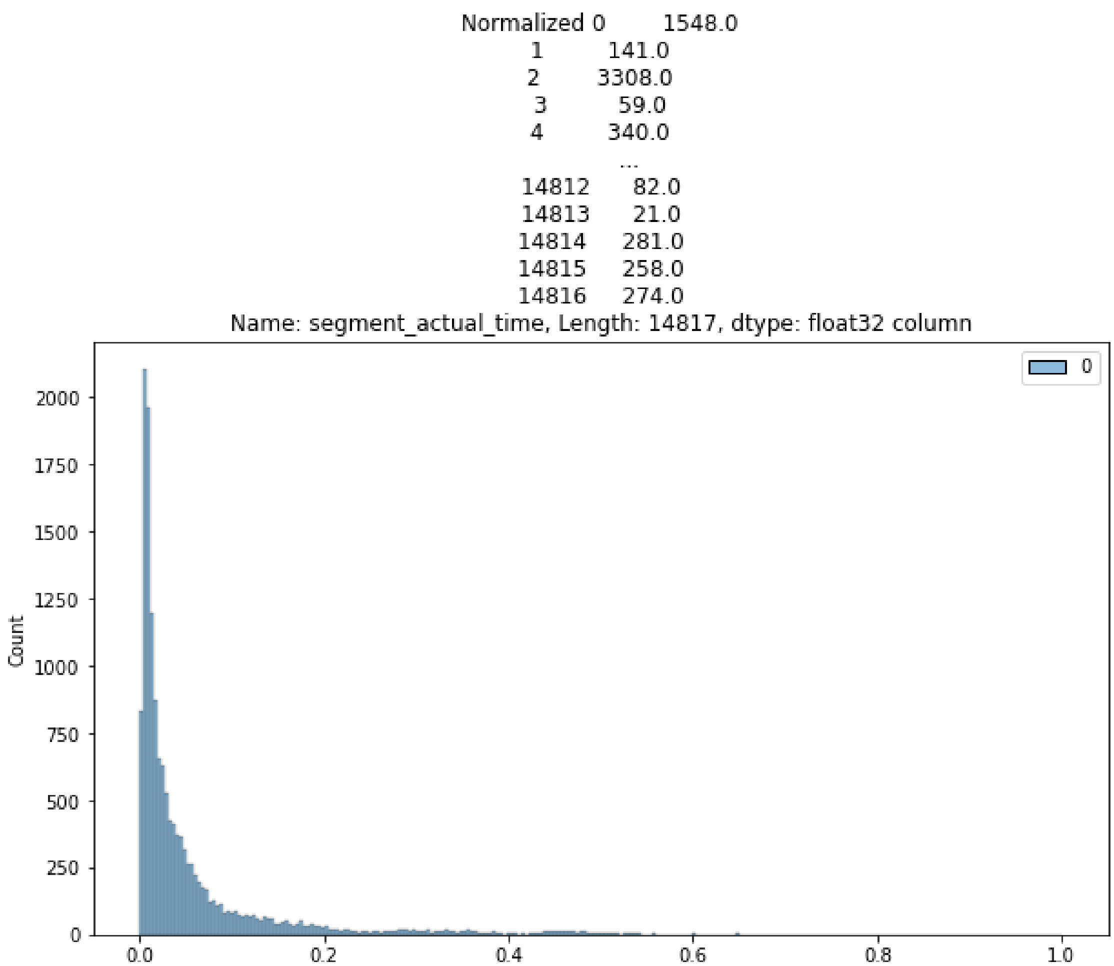
[141]: []



[ ]:

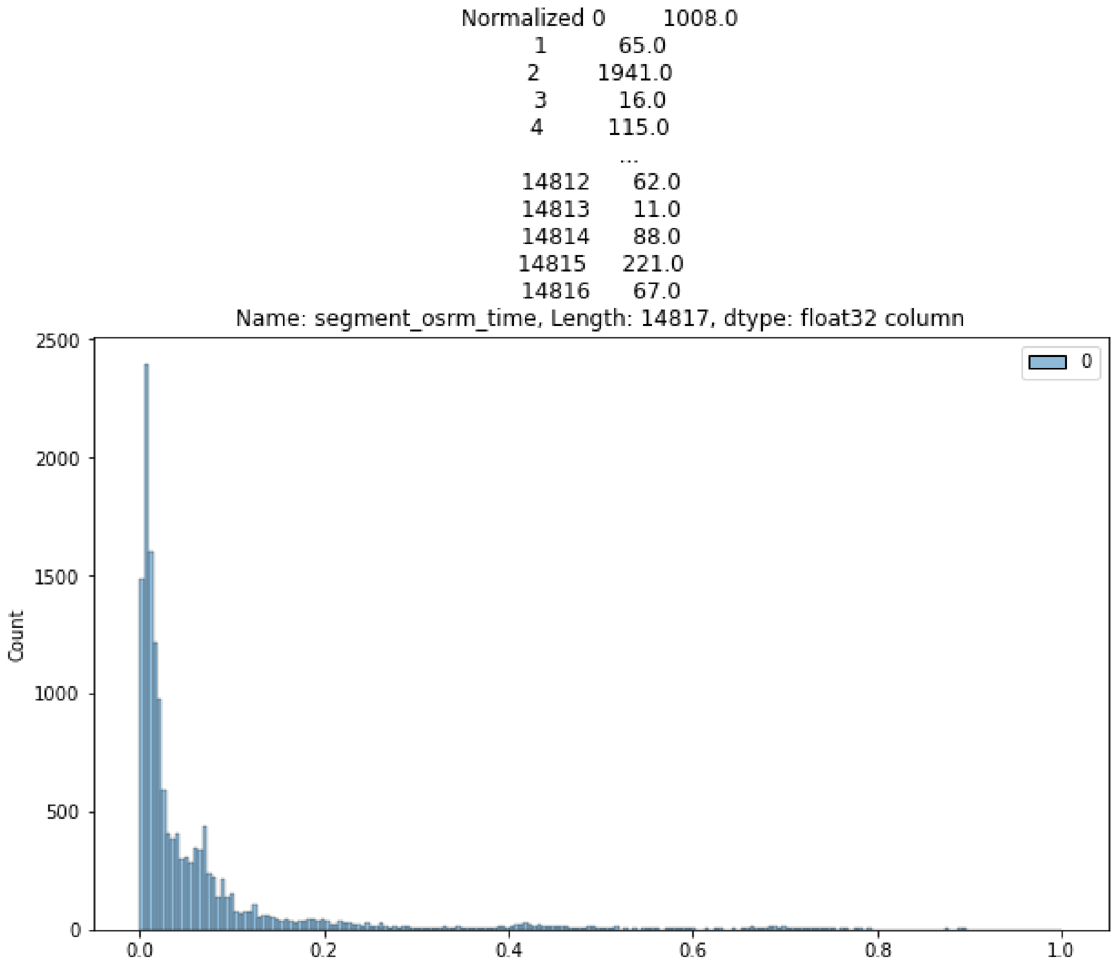
[142]:
```python
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,␣
↪1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_actual_time']} column")
plt.plot()
```

[142]: []



[143]:
```
plt.figure(figsize = (10, 6))
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()
```

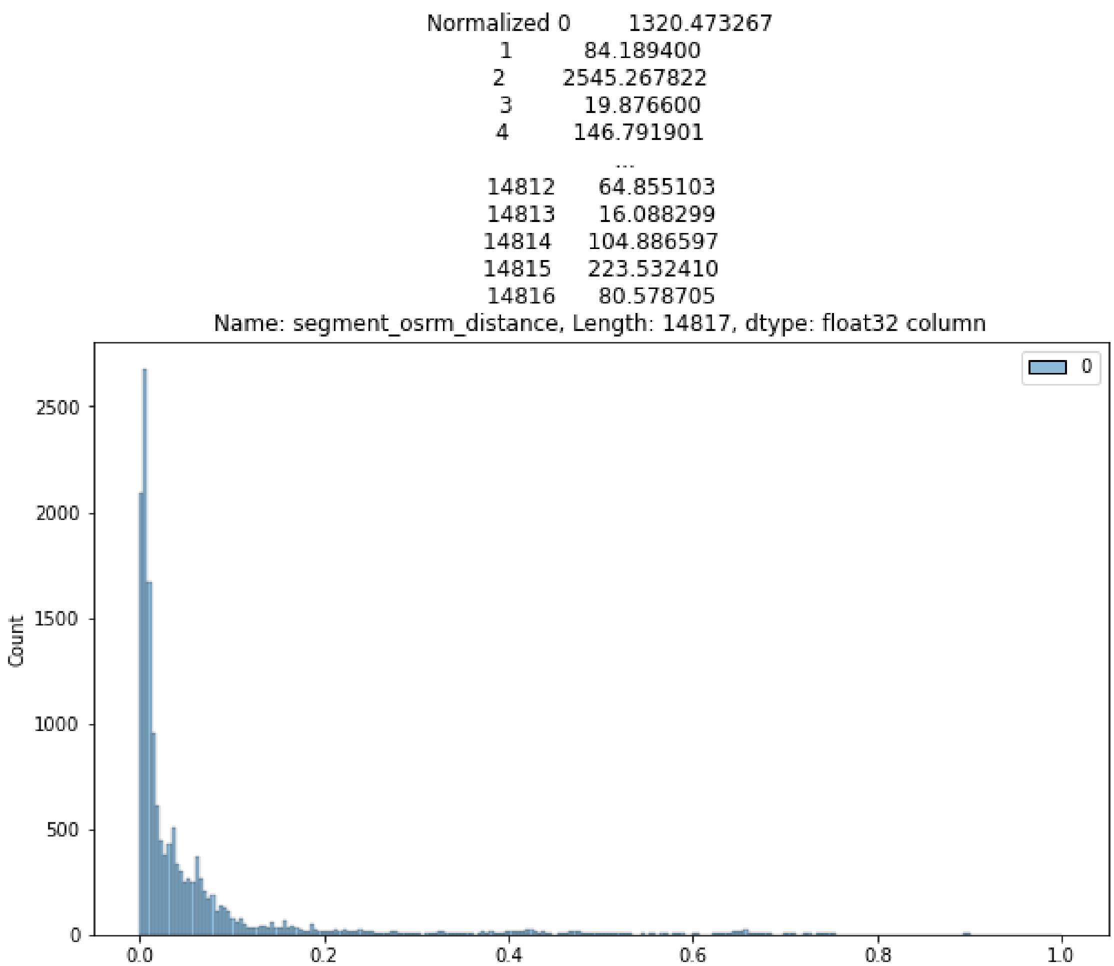
[143]: []



[144]:

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

[144]: []

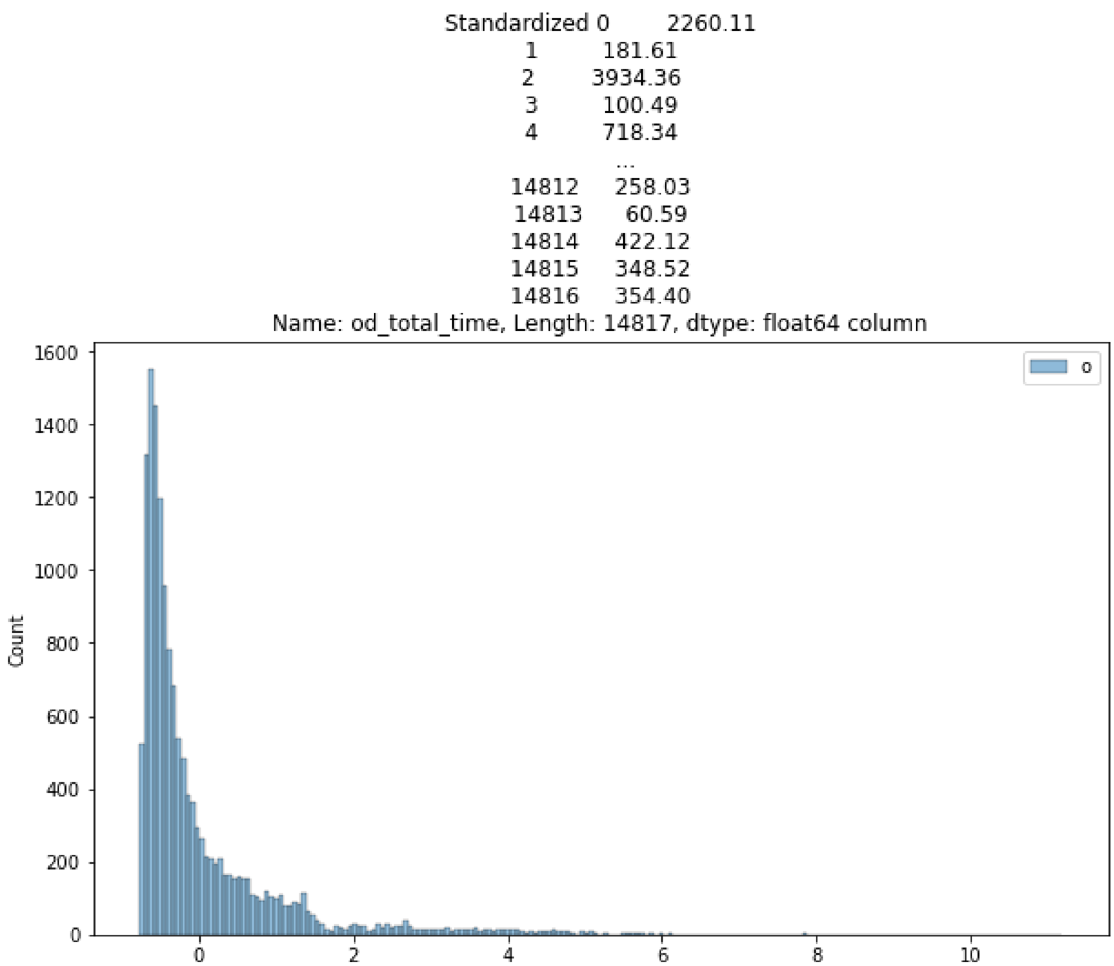


Column Standardization

[145]:
```python
from sklearn.preprocessing import StandardScaler
```

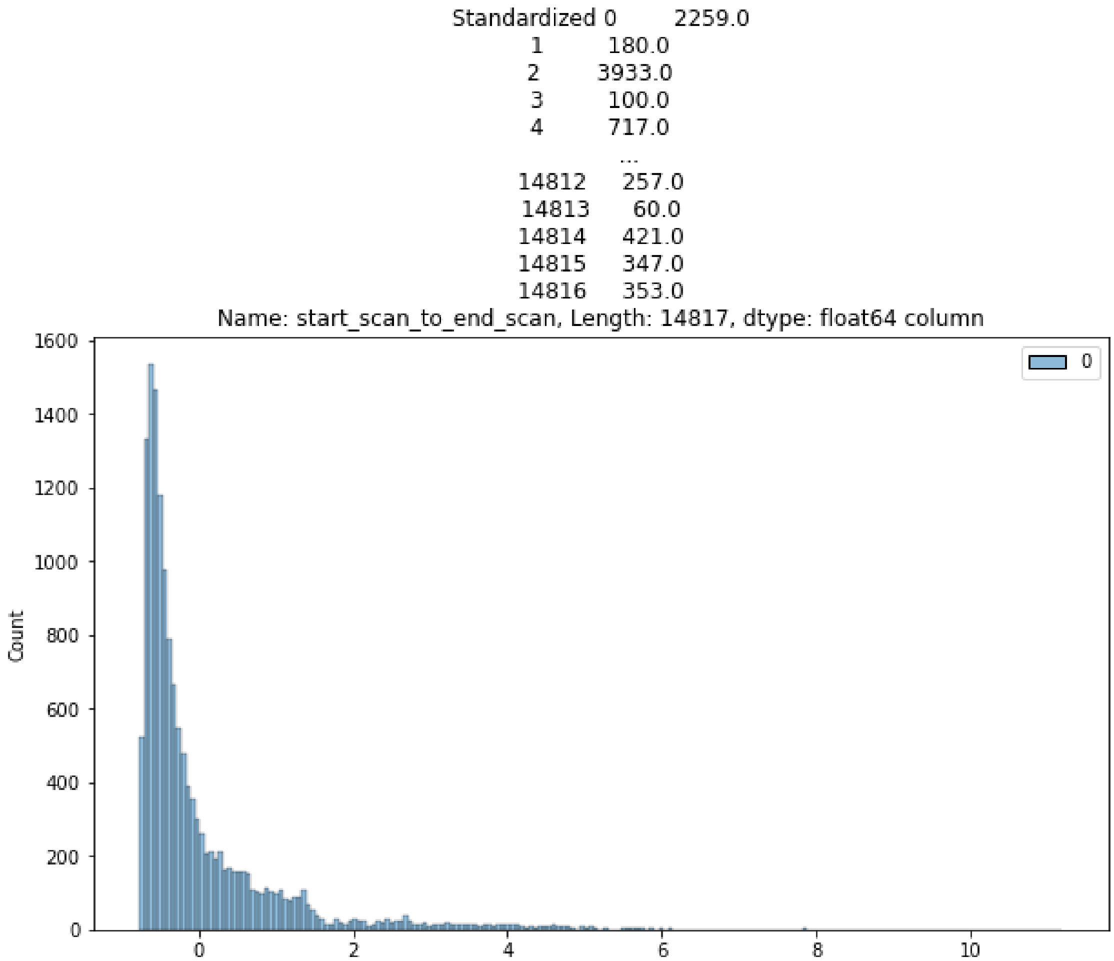
[146]:
```python
plt.figure(figsize = (10, 6))
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['od_total_time']} column")
plt.legend('od_total_time')
plt.plot()
```

[146]: []



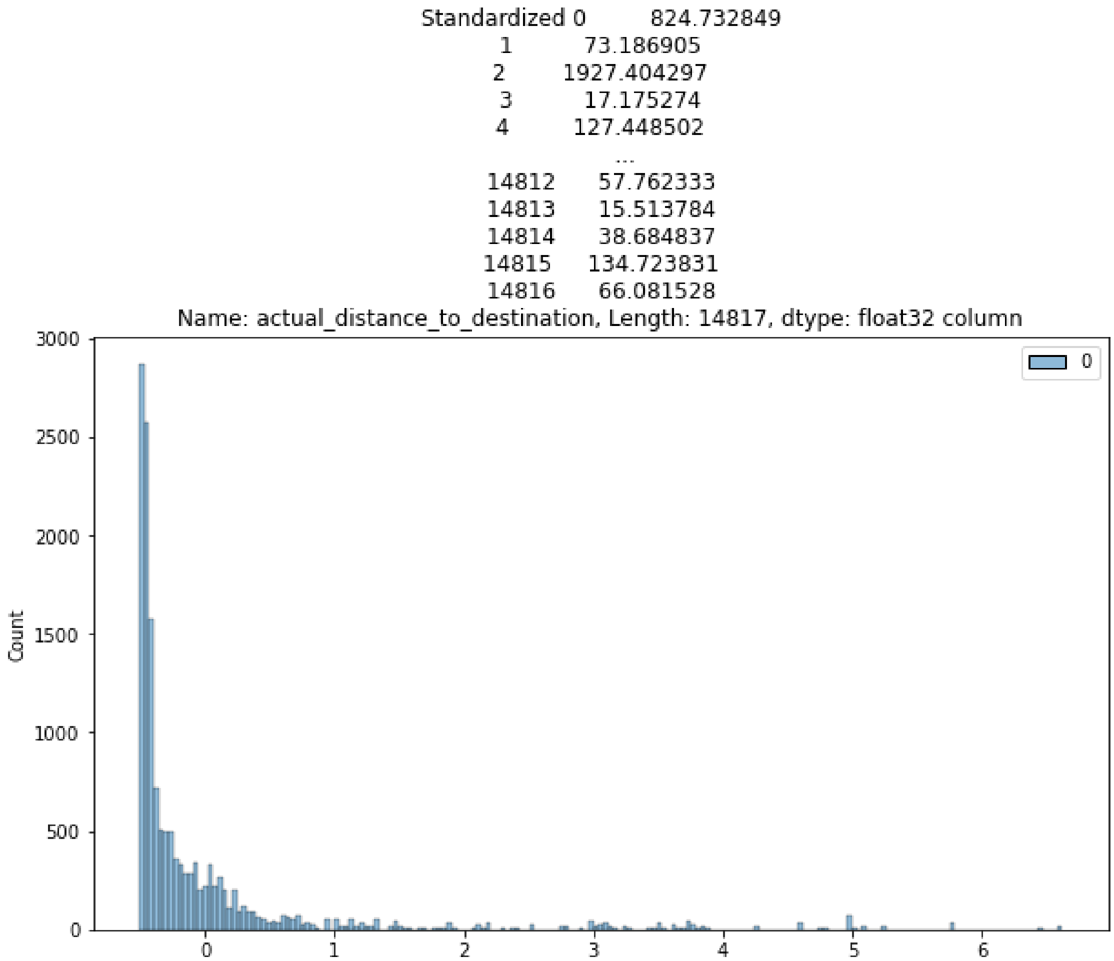
[147]:
```python
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().
 ↪reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['start_scan_to_end_scan']} column")
plt.plot()
```

[147]: []



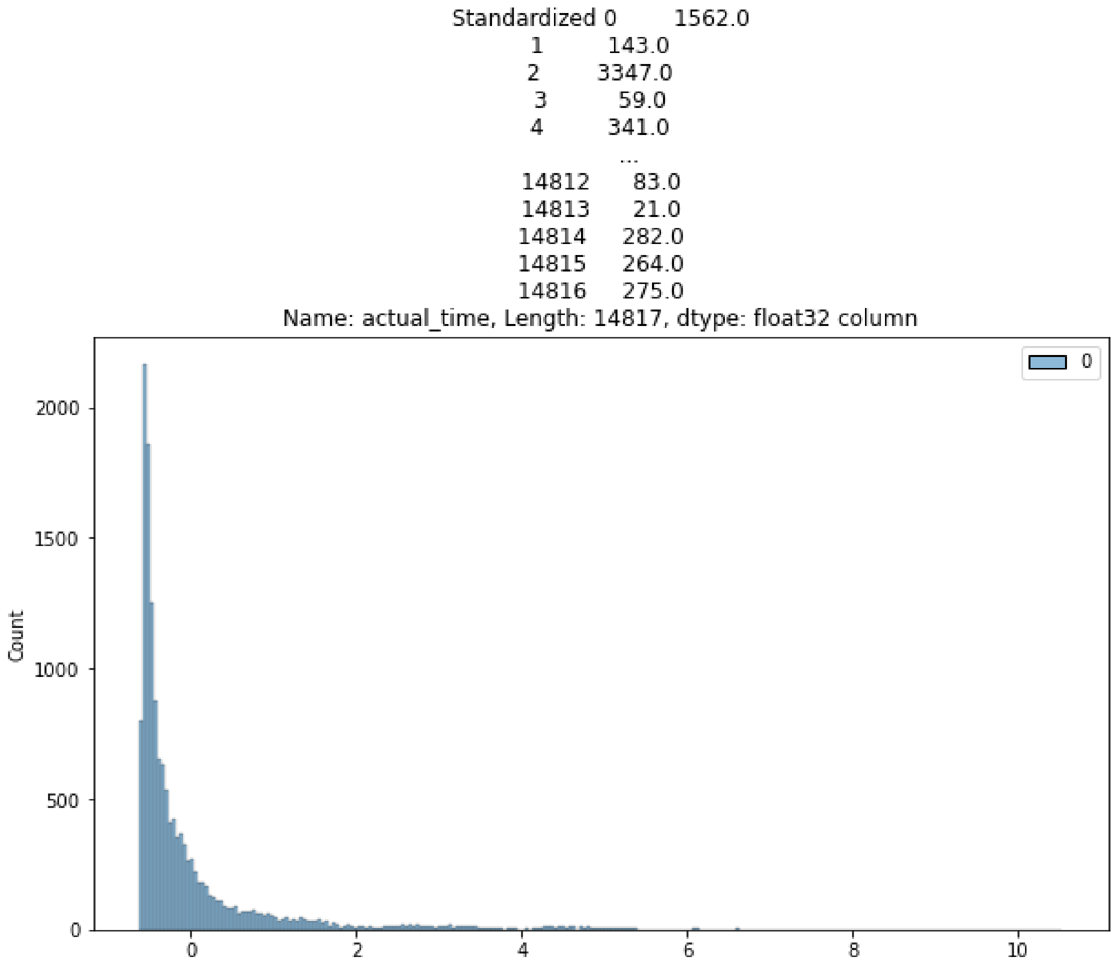
[148]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().
 ↪reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_distance_to_destination']} column")
plt.plot()
```

[148]: []



[149]:
```
plt.figure(figsize = (10, 6))
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()
```

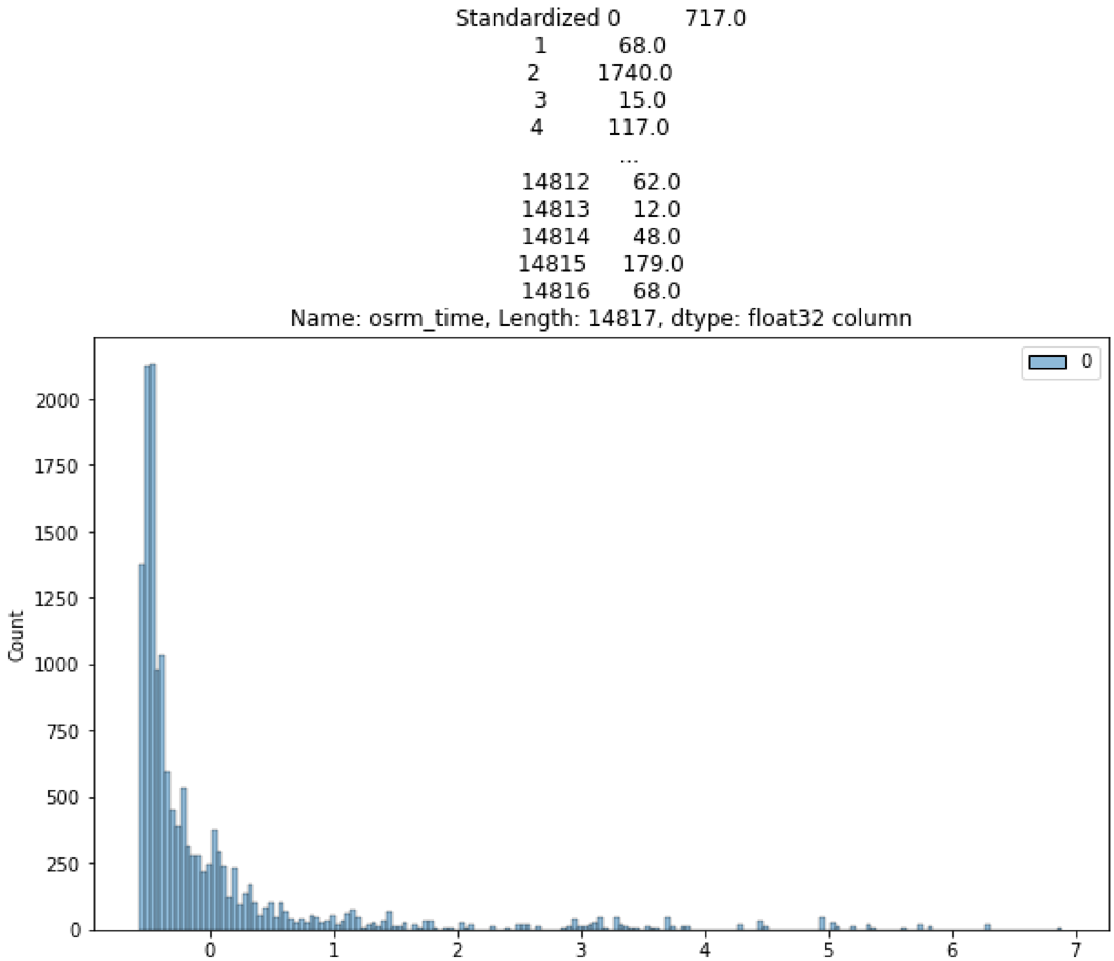
[149]: []



[ ]:

[150]:
```python
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()
```

[150]: []

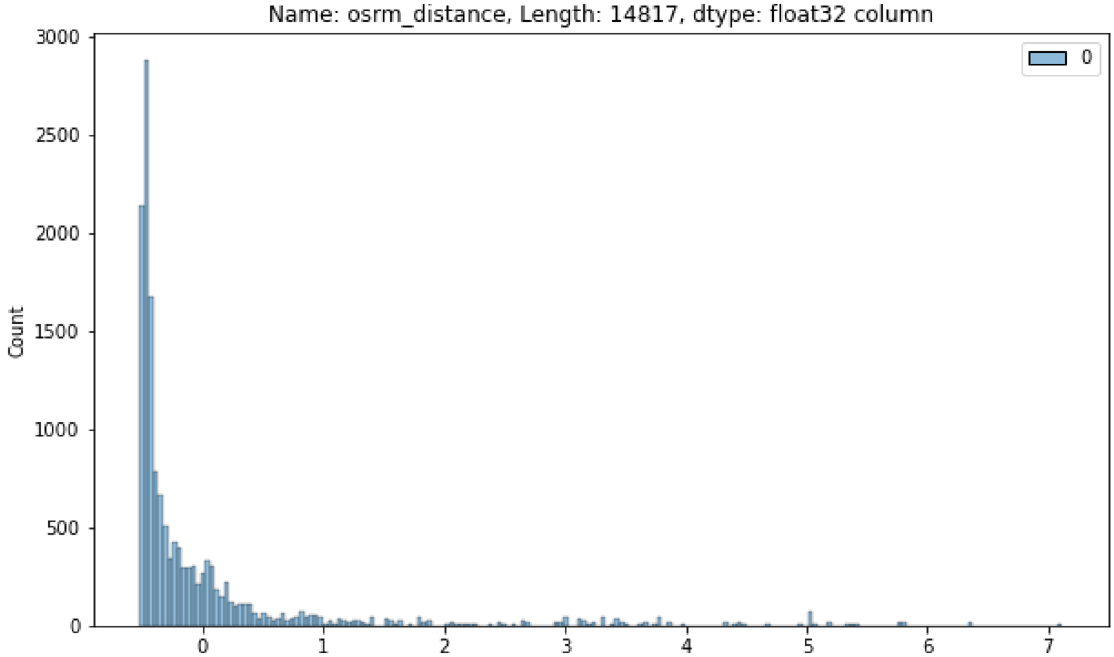


[ ]:

[151]:
```
plt.figure(figsize = (10, 6))
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()
```

[151]: []

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

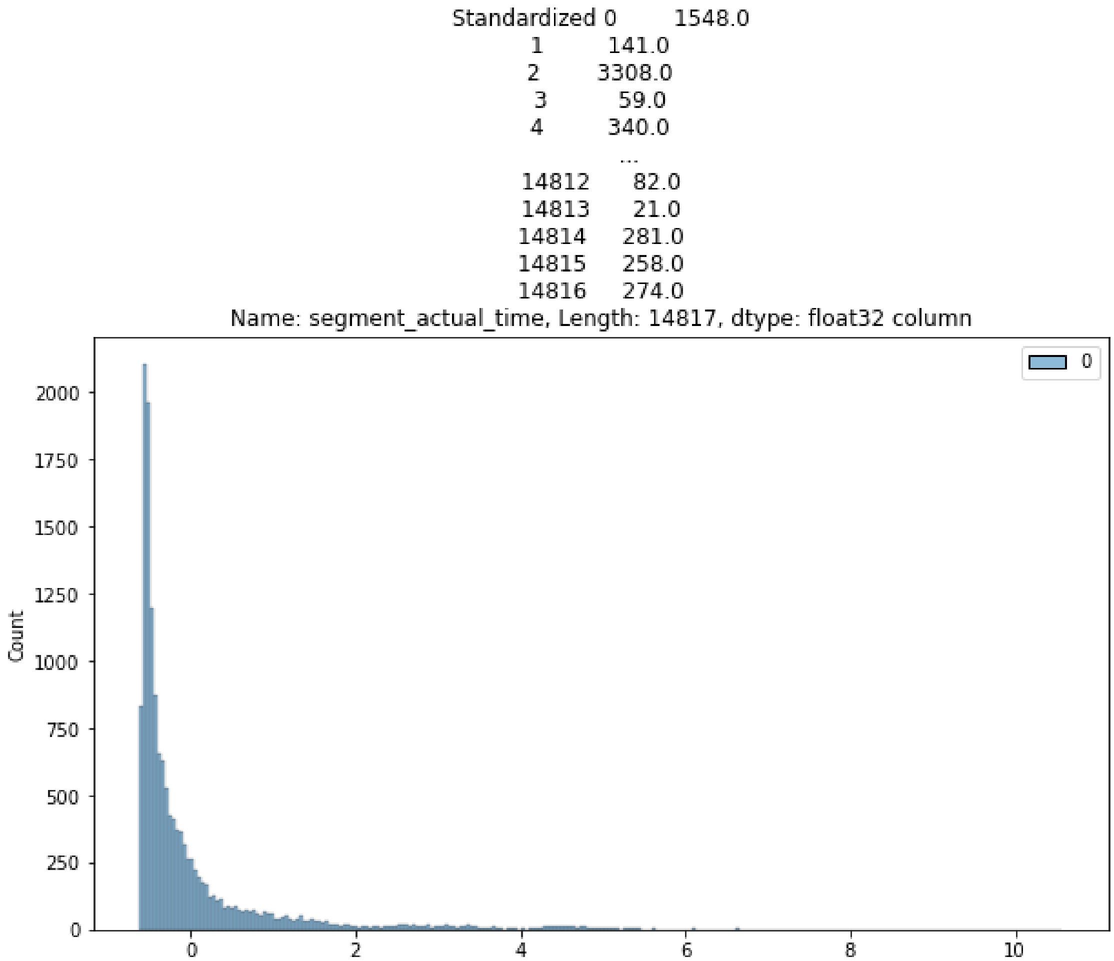


[ ]:

[152]:

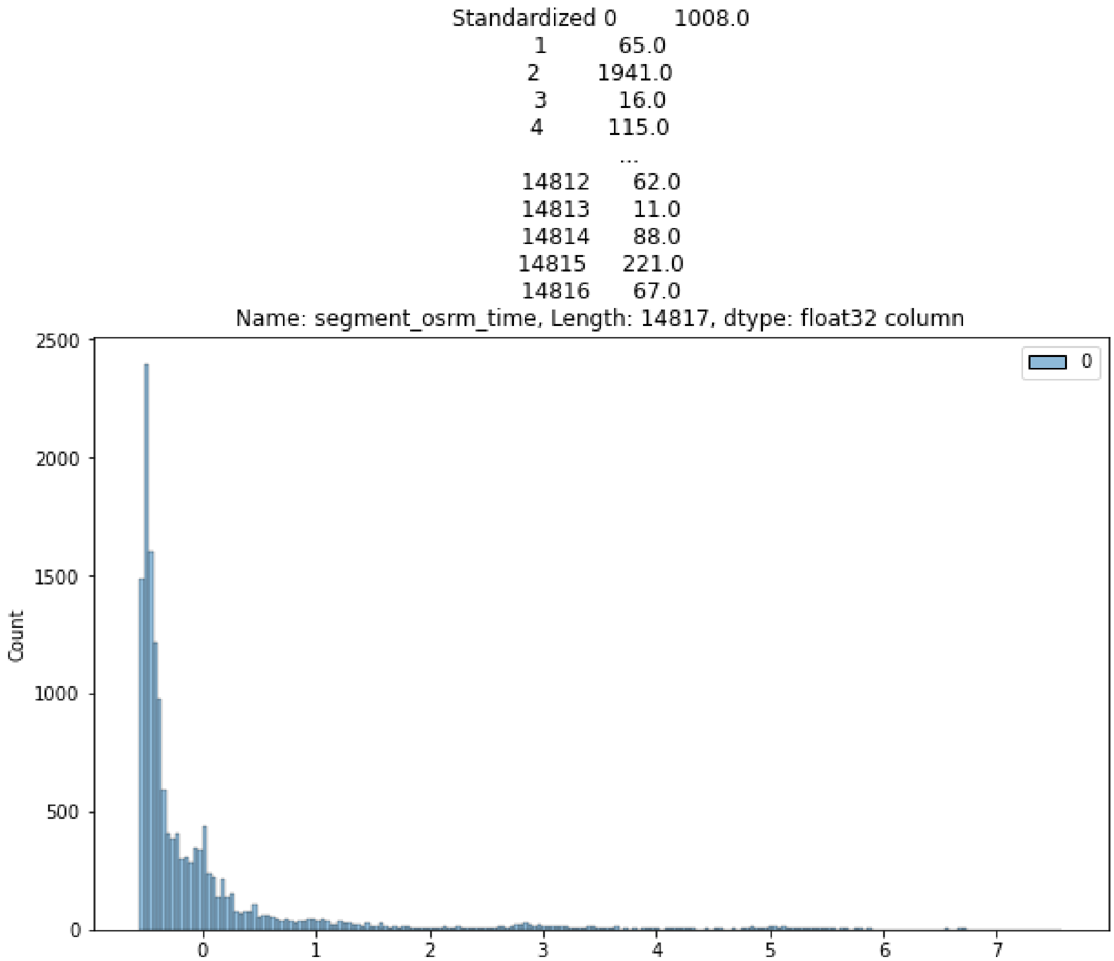
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,
    1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_actual_time']} column")
plt.plot()
```

[152]: []



[153]:
```
plt.figure(figsize = (10, 6))
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()
```

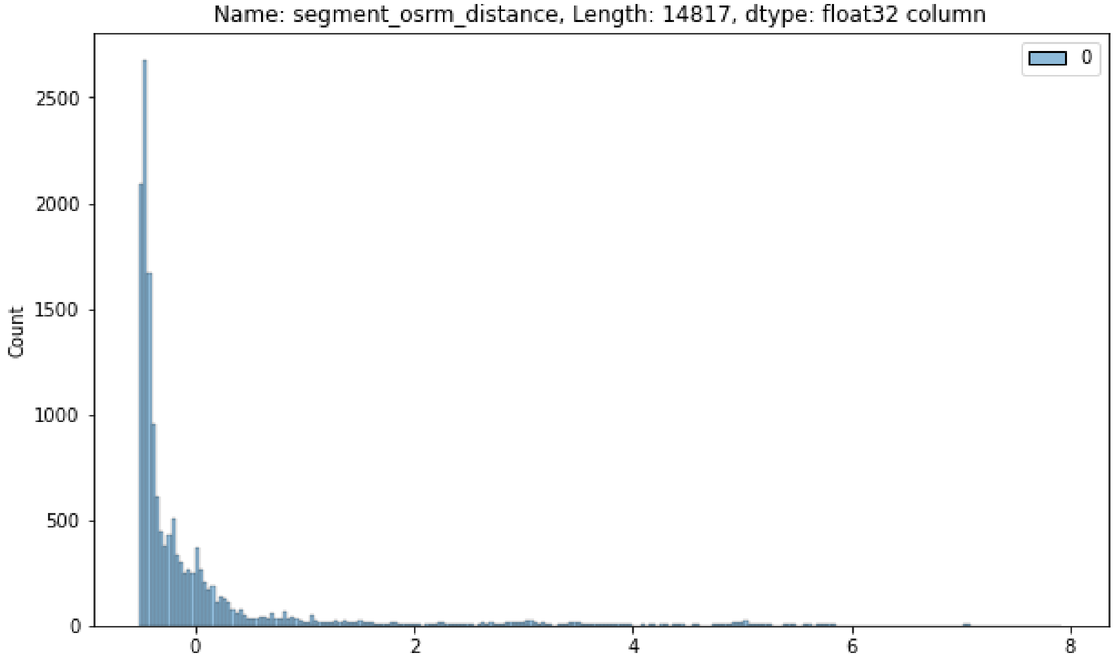
[153]: []



[154]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['segment_osrm_distance'].to_numpy().
↪reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_osrm_distance']} column")
plt.plot()
```

[154]: []

```
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          223.532410
        14816           80.578705
Name: segment_osrm_distance, Length: 14817, dtype: float32 column
```



1.6 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.

1.7 Recommendations

- 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.

[ ]:

```

```