DELHIVERY-FEATURE ENGINEERING

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

DatasetLink:

https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181

Column Profiling:

```
data - tells whether the data is testing or training data
trip_creation_time – Timestamp of trip creation
route_schedule_uuid – Unique Id for a particular route schedule
route_type – Transportation type
FTL – Full Truck Load: FTL shipments get to the destination sooner, as
the truck is making no other pickups or drop-offs along the way
Carting: Handling system consisting of small vehicles (carts)
trip_uuid - Unique ID given to a particular trip (A trip may include
different source and destination centers)
source_center - Source ID of trip origin
source_name - Source Name of trip origin
destination_cente – Destination ID
destination_name – Destination Name
od_start_time – Trip start time
od_end_time – Trip end time
start_scan_to_end_scan – Time taken to deliver from source to
destination
is_cutoff – Unknown field
cutoff_factor – Unknown field
cutoff_timestamp – Unknown field
actual_distance_to_destination – Distance in Kms between source and
destination warehouse
actual_time – Actual time taken to complete the delivery (Cumulative)
osrm_time – An open-source routing engine time calculator which
computes the shortest path between points in a given map (Includes
usual traffic, distance through major and minor roads) and gives the
time (Cumulative)
osrm_distance – An open-source routing engine which computes the
shortest path between points in a given map (Includes usual traffic,
distance through major and minor roads) (Cumulative)
factor – Unknown field
```

```
segment_actual_time – This is a segment time. Time taken by the subset
of the package delivery
segment_osrm_time – This is the OSRM segment time. Time taken by the
subset of the package delivery
segment_osrm_distance – This is the OSRM distance. Distance covered by
subset of the package delivery
segment_factor – Unknown field
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import f_oneway, ttest_ind, shapiro, kruskal,
chi2_contingency, levene
from statsmodels.graphics.gofplots import qqplot
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

Loading Delhivery data

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

delhivery_data.head()
```

```
       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]
```

```
delhivery_data.shape
```

```
(144867, 24)
```

There are 24 fields with 144867 rows. Out of 24 fields 5 are unknown fileds which can be dropped.

is_cutoff – Unknown field

cutoff_factor – Unknown field

cutoff_timestamp – Unknown field

factor – Unknown field

segment_factor - Unknown field

1. BASIC DATA CLEANING AND EXPLORTION

Dropping unknown fields

```
delhivery_data = delhivery_data.drop(['is_cutoff','cutoff_factor','cutoff_timestamp','factor','segment_factor'], axis = 1)
```

```
delhivery_data.shape
```

```
(144867, 19)
```

```
list(delhivery_data.columns)
```

```
['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',
 'actual_distance_to_destination',
 'actual_time',
```

```
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance']
```

```
delhivery_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                           Non-Null Count   Dtype  
---  ------                           --------------   -----  
 0   data                             144867 non-null  object 
 1   trip_creation_time               144867 non-null  object 
 2   route_schedule_uuid              144867 non-null  object 
 3   route_type                       144867 non-null  object 
 4   trip_uuid                        144867 non-null  object 
 5   source_center                    144867 non-null  object 
 6   source_name                      144574 non-null  object 
 7   destination_center               144867 non-null  object 
 8   destination_name                 144606 non-null  object 
 9   od_start_time                    144867 non-null  object 
 10  od_end_time                      144867 non-null  object 
 11  start_scan_to_end_scan           144867 non-null  float64
 12  actual_distance_to_destination   144867 non-null  float64
 13  actual_time                      144867 non-null  float64
 14  osrm_time                        144867 non-null  float64
 15  osrm_distance                    144867 non-null  float64
 16  segment_actual_time              144867 non-null  float64
 17  segment_osrm_time                144867 non-null  float64
 18  segment_osrm_distance            144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

There are three date_time fields of object type which are needed to be converted into date_time.

trip_creation_time

od_start_time

od_end_time

Converting date fields to type dateTime

```
delhivery_data['trip_creation_time']=pd.to_datetime(delhivery_data['trip_creation_time'])
delhivery_data['od_start_time']=pd.to_datetime(delhivery_data['od_start_time'])
```

```
delhivery_data['od_end_time']=pd.to_datetime(delhivery_data['od_end_time'])

delhivery_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                           Non-Null Count   Dtype         
---  ------                           --------------   -----         
 0   data                             144867 non-null  object        
 1   trip_creation_time               144867 non-null  datetime64[ns]
 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  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  float64       
 13  actual_time                      144867 non-null  float64       
 14  osrm_time                        144867 non-null  float64       
 15  osrm_distance                    144867 non-null  float64       
 16  segment_actual_time              144867 non-null  float64       
 17  segment_osrm_time                144867 non-null  float64       
 18  segment_osrm_distance            144867 non-null  float64       
dtypes: datetime64[ns](3), float64(8), object(8)
memory usage: 21.0+ MB
```

```
delhivery_data.isnull().sum()
```

```
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
```

For handling null values in source_name and destination_name, getting list of source_center and destination_center for the corresponding null values.

```
ids_source_name_null =
list(delhivery_data.loc[delhivery_data.source_name.isnull(),'source_ce
nter'].unique())
ids_source_name_null
```

```
['IND342902A1B',
 'IND577116AAA',
 'IND282002AAD',
 'IND465333A1B',
 'IND841301AAC',
 'IND509103AAC',
 'IND126116AAA',
 'IND331022A1B',
 'IND505326AAB',
 'IND852118A1B']
```

```
len(ids_source_name_null)
```

```
10
```

```
ids_destination_name_null =
list(delhivery_data[delhivery_data.destination_name.isnull()].destinat
ion_center.unique())
ids_destination_name_null
```

```
['IND342902A1B',
 'IND577116AAA',
 'IND282002AAD',
 'IND465333A1B',
 'IND841301AAC',
 'IND505326AAB',
 'IND852118A1B',
 'IND126116AAA',
 'IND509103AAC',
 'IND221005A1A',
 'IND250002AAC',
 'IND331001A1C',
 'IND122015AAC']
```

```
len(ids_destination_name_null)
```

```
13
```

```
np.all(ids_source_name_null in ids_destination_name_null)
```

```
False
```

Missing field ids are not same for both souce and destination. so merging them and creating single unique list.

```
missing_name_id =
list(set(ids_destination_name_null+ids_source_name_null))
```

There are some missing area names both in source_name and destination_name in common. We are going to handle this by giving some dummy location name in the same format as 'Anand_VUNagar_DC (Gujarat)'.

```
dummy_values= {}
for i,id in enumerate(missing_name_id):
    dummy_values[id] = f'city{i}_place{i}_code{i}(Unkonwn{i})'

dummy_values
```

```
{'IND221005A1A': 'city0_place0_code0(Unkonwn0)',
 'IND282002AAD': 'city1_place1_code1(Unkonwn1)',
 'IND250002AAC': 'city2_place2_code2(Unkonwn2)',
 'IND122015AAC': 'city3_place3_code3(Unkonwn3)',
 'IND465333A1B': 'city4_place4_code4(Unkonwn4)',
 'IND331001A1C': 'city5_place5_code5(Unkonwn5)',
 'IND577116AAA': 'city6_place6_code6(Unkonwn6)',
 'IND505326AAB': 'city7_place7_code7(Unkonwn7)',
 'IND852118A1B': 'city8_place8_code8(Unkonwn8)',
 'IND509103AAC': 'city9_place9_code9(Unkonwn9)',
 'IND126116AAA': 'city10_place10_code10(Unkonwn10)',
 'IND342902A1B': 'city11_place11_code11(Unkonwn11)',
 'IND841301AAC': 'city12_place12_code12(Unkonwn12)',
 'IND331022A1B': 'city13_place13_code13(Unkonwn13)'}
```

Handling missing values in the data.

Filling null values in source_center and destination_center with dummy values corresponding to their source_center and destination_center

```
for id in ids_source_name_null:
    if id in dummy_values:
        delhivery_data.loc[delhivery_data.source_center == id,
'source_name']= dummy_values[id]

for id in ids_source_name_null:
    if id in dummy_values:
```

```
        print(delhivery_data.loc[delhivery_data.source_center == id,
['source_center','source_name']].value_counts())
```

```
source_center  source_name
IND342902A1B   city11_place11_code11(Unkonwn11)    90
dtype: int64
source_center  source_name
IND577116AAA   city6_place6_code6(Unkonwn6)    16
dtype: int64
source_center  source_name
IND282002AAD   city1_place1_code1(Unkonwn1)    128
dtype: int64
source_center  source_name
IND465333A1B   city4_place4_code4(Unkonwn4)    6
dtype: int64
source_center  source_name
IND841301AAC   city12_place12_code12(Unkonwn12)    5
dtype: int64
source_center  source_name
IND509103AAC   city9_place9_code9(Unkonwn9)    17
dtype: int64
source_center  source_name
IND126116AAA   city10_place10_code10(Unkonwn10)    20
dtype: int64
source_center  source_name
IND331022A1B   city13_place13_code13(Unkonwn13)    3
dtype: int64
source_center  source_name
IND505326AAB   city7_place7_code7(Unkonwn7)    5
dtype: int64
source_center  source_name
IND852118A1B   city8_place8_code8(Unkonwn8)    3
dtype: int64
```

```
for id in ids_destination_name_null:
    if id in dummy_values:
        delhivery_data.loc[delhivery_data.destination_center == id,
'destination_name']= dummy_values[id]
```

```
for id in ids_destination_name_null:
    if id in dummy_values:
        print(delhivery_data.loc[delhivery_data.destination_center ==
id, ['destination_center','destination_name']].value_counts())
```

```
destination_center  destination_name
IND342902A1B        city11_place11_code11(Unkonwn11)    16
dtype: int64
destination_center  destination_name
```

```
IND577116AAA        city6_place6_code6(Unkonwn6)    16
dtype: int64
destination_center  destination_name
IND282002AAD        city1_place1_code1(Unkonwn1)    151
dtype: int64
destination_center  destination_name
IND465333A1B        city4_place4_code4(Unkonwn4)    3
dtype: int64
destination_center  destination_name
IND841301AAC        city12_place12_code12(Unkonwn12)    9
dtype: int64
destination_center  destination_name
IND505326AAB        city7_place7_code7(Unkonwn7)    11
dtype: int64
destination_center  destination_name
IND852118A1B        city8_place8_code8(Unkonwn8)    15
dtype: int64
destination_center  destination_name
IND126116AAA        city10_place10_code10(Unkonwn10)    10
dtype: int64
destination_center  destination_name
IND509103AAC        city9_place9_code9(Unkonwn9)    9
dtype: int64
destination_center  destination_name
IND221005A1A        city0_place0_code0(Unkonwn0)    1
dtype: int64
destination_center  destination_name
IND250002AAC        city2_place2_code2(Unkonwn2)    9
dtype: int64
destination_center  destination_name
IND331001A1C        city5_place5_code5(Unkonwn5)    3
dtype: int64
destination_center  destination_name
IND122015AAC        city3_place3_code3(Unkonwn3)    8
dtype: int64
```

```
delhivery_data.isnull().sum().sum()
```

```
0
```

There are no null values we can proceed with analysis.

```
#Creating a copy of dataframe

delhivery = delhivery_data.copy()

delhivery.head()
```

```
       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  ... actual_time  osrm_time
osrm_distance  \
0 2018-09-20 03:21:32.418600  ...        14.0       11.0
11.9653
1 2018-09-20 03:21:32.418600  ...        24.0       20.0
21.7243
2 2018-09-20 03:21:32.418600  ...        40.0       28.0
32.5395
3 2018-09-20 03:21:32.418600  ...        62.0       40.0
45.5620
4 2018-09-20 03:21:32.418600  ...        68.0       44.0
54.2181

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

```
4                   6.0                   5.0                    3.9153

                                          segment_key
segment_actual_time_sum  \
0  trip-153741093647649320IND388121AAAIND388620AAB
14.0
1  trip-153741093647649320IND388121AAAIND388620AAB
24.0
2  trip-153741093647649320IND388121AAAIND388620AAB
40.0
3  trip-153741093647649320IND388121AAAIND388620AAB
61.0
4  trip-153741093647649320IND388121AAAIND388620AAB
67.0

   segment_osrm_distance_sum segment_osrm_time_sum
0                    11.9653                  11.0
1                    21.7243                  20.0
2                    32.5395                  27.0
3                    45.5619                  39.0
4                    49.4772                  44.0

[5 rows x 23 columns]
```

Exploring each column to segregate categorical and numerical columns

```
delhivery.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                           Non-Null Count   Dtype
---  ------                           --------------   -----
 0   data                             144867 non-null  object
 1   trip_creation_time               144867 non-null  datetime64[ns]
 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                      144867 non-null  object
 7   destination_center               144867 non-null  object
 8   destination_name                 144867 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  float64
```

```
 13  actual_time                   144867 non-null  float64
 14  osrm_time                     144867 non-null  float64
 15  osrm_distance                 144867 non-null  float64
 16  segment_actual_time           144867 non-null  float64
 17  segment_osrm_time             144867 non-null  float64
 18  segment_osrm_distance         144867 non-null  float64
dtypes: datetime64[ns](3), float64(8), object(8)
memory usage: 21.0+ MB
```

```
print("-- 'data' field has",delhivery.data.unique(),"as unique
values")
print("-- The 'trip_creation_time' for the data given is
between",delhivery.trip_creation_time.min().date(),"and",
      delhivery.trip_creation_time.max().date(),"dates")

print("-- 'route_schedule_uuid' field
has",len(list(delhivery.route_schedule_uuid.unique())),"unique
values")

print("-- 'route_type' field
has",list(delhivery.route_type.unique()),"as unique values")

print("-- 'trip_uuid' field
has",len(list(delhivery.trip_uuid.unique())),"unique values")

print("-- 'source_center' field
has",len(list(delhivery.source_center.unique())),"unique values")

print("-- 'source_name' field
has",len(list(delhivery.source_name.unique())),"unique values")

print("-- 'destination_center' field
has",len(list(delhivery.destination_center.unique())),"unique values")

print("-- 'destination_name' field
has",len(list(delhivery.destination_name.unique())),"unique values")

print("-- The 'od_start_time' for the data given is
between",delhivery.od_start_time.min().date(),"and",
      delhivery.od_start_time.max().date(),"dates")

print("-- The 'od_end_time' for the data given is
between",delhivery.od_end_time.min().date(),"and",
      delhivery.od_end_time.max().date(),"dates")
```

```
-- 'data' field has ['training' 'test'] as unique values
-- The 'trip_creation_time' for the data given is between 2018-09-12
and 2018-10-03 dates
-- 'route_schedule_uuid' field has 1504 unique values
-- 'route_type' field has ['Carting', 'FTL'] as unique values
-- 'trip_uuid' field has 14817 unique values
```

```
-- 'source_center' field has 1508 unique values
-- 'source_name' field has 1508 unique values
-- 'destination_center' field has 1481 unique values
-- 'destination_name' field has 1481 unique values
-- The 'od_start_time' for the data given is between 2018-09-12 and
2018-10-06 dates
-- The 'od_end_time' for the data given is between 2018-09-12 and
2018-10-08 dates
```

From the above obeservation 'data' and 'route_type' can be main categorical coulumns and all with nemerical values can be numerical fields.

```
cat_cols = ['data', 'route_type']
delhivery[cat_cols]= delhivery[cat_cols].astype('category')

num_cols = list(delhivery.dtypes.reset_index()
[delhivery.dtypes.reset_index()[0]=='float64']['index'])
num_cols

['start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance']

delhivery.dtypes

data                                    category
trip_creation_time                datetime64[ns]
route_schedule_uuid                       object
route_type                              category
trip_uuid                                 object
source_center                             object
source_name                               object
destination_center                        object
destination_name                          object
od_start_time                     datetime64[ns]
od_end_time                       datetime64[ns]
start_scan_to_end_scan                   float64
actual_distance_to_destination           float64
actual_time                              float64
osrm_time                                float64
osrm_distance                            float64
segment_actual_time                      float64
segment_osrm_time                        float64
segment_osrm_distance                    float64
dtype: object
```

```
delhivery.describe()
```

```
       start_scan_to_end_scan  actual_distance_to_destination
actual_time  \
count           144867.000000                   144867.000000
144867.000000
mean               961.262986                      234.073372
416.927527
std               1037.012769                      344.990009
598.103621
min                 20.000000                        9.000045
9.000000
25%                161.000000                       23.355874
51.000000
50%                449.000000                       66.126571
132.000000
75%               1634.000000                      286.708875
513.000000
max               7898.000000                     1927.447705
4532.000000

           osrm_time  osrm_distance  segment_actual_time
segment_osrm_time  \
count  144867.000000  144867.000000        144867.000000
144867.000000
mean      213.868272     284.771297            36.196111
18.507548
std       308.011085     421.119294            53.571158
14.775960
min         6.000000       9.008200          -244.000000
0.000000
25%        27.000000      29.914700            20.000000
11.000000
50%        64.000000      78.525800            29.000000
17.000000
75%       257.000000     343.193250            40.000000
22.000000
max      1686.000000    2326.199100          3051.000000
1611.000000

       segment_osrm_distance
count           144867.00000
mean                22.82902
std                 17.86066
min                  0.00000
25%                 12.07010
50%                 23.51300
75%                 27.81325
max               2191.40370
```

```
delhivery.describe(include=['object','category'])
```

```
            data                                   route_schedule_uuid  \
count     144867                                                144867
unique         2                                                  1504
top     training  thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...
freq      104858                                                  1812

       route_type                trip_uuid source_center  \
count      144867                   144867        144867
unique          2                    14817          1508
top           FTL  trip-153776597384821516  IND000000ACB
freq        99660                      101         23347

                           source_name destination_center  \
count                           144867             144867
unique                            1508               1481
top     Gurgaon_Bilaspur_HB (Haryana)       IND000000ACB
freq                            23347              15192

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

From the above table we can observe that:

```
- The mode of 'data' field is 'training'
- With frequent 'route_type' as 'FTL'
- With top source and destinations as 'Gurgaon_Bilaspur_HB (Haryana)'
```

2. MERGING ROWS AND AGGREGATING COLUMNS

Grouping by segment

```
delhivery.head()
```

```
       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                 od_end_time  \
0 2018-09-20 03:21:32.418600 2018-09-20 04:47:45.236797
1 2018-09-20 03:21:32.418600 2018-09-20 04:47:45.236797
2 2018-09-20 03:21:32.418600 2018-09-20 04:47:45.236797
3 2018-09-20 03:21:32.418600 2018-09-20 04:47:45.236797
4 2018-09-20 03:21:32.418600 2018-09-20 04:47:45.236797

   start_scan_to_end_scan  actual_distance_to_destination  actual_time
\
0                    86.0                       10.435660         14.0

1                    86.0                       18.936842         24.0

2                    86.0                       27.637279         40.0

3                    86.0                       36.118028         62.0

4                    86.0                       39.386040         68.0


   osrm_time  osrm_distance  segment_actual_time  segment_osrm_time  \
0       11.0        11.9653                 14.0               11.0
1       20.0        21.7243                 10.0                9.0
```

```
2       28.0         32.5395                   16.0                    7.0
3       40.0         45.5620                   21.0                   12.0
4       44.0         54.2181                    6.0                    5.0
```

```
   segment_osrm_distance
0                11.9653
1                 9.7590
2                10.8152
3                13.0224
4                 3.9153
```

```
# grouping by trip_uuid, source_center, destination_center

#creating segment_key for each segment
delhivery['segment_key'] = delhivery['trip_uuid'] +
delhivery['source_center'] + delhivery['destination_center']

#segment_actual_time_sum,segment_osrm_distance_sum,
segment_osrm_time_sum.
delhivery['segment_actual_time_sum'] =
delhivery.groupby('segment_key').agg(segment_actual_time_sum =
('segment_actual_time',np.cumsum))
delhivery['segment_osrm_distance_sum'] =
delhivery.groupby('segment_key').agg(segment_osrm_distance_sum =
('segment_osrm_distance',np.cumsum))
delhivery['segment_osrm_time_sum'] =
delhivery.groupby('segment_key').agg(segment_osrm_time_sum =
('segment_osrm_time',np.cumsum))

delhivery.loc[delhivery.segment_key == 'trip-
153741093647649320IND388121AAAIND388620AAB',
['actual_distance_to_destination','actual_time']]
```

```
   actual_distance_to_destination  actual_time
0                       10.435660         14.0
1                       18.936842         24.0
2                       27.637279         40.0
3                       36.118028         62.0
4                       39.386040         68.0
```

Aggregating at segment level

```
create_segment_dict ={
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_center' : 'first',
```

```
    'source_name' : 'first',

    'destination_center' : 'last',
    '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' : 'last',
    'segment_osrm_distance_sum' : 'last',
    'segment_osrm_time_sum' : 'last',
}

#creating a segment_level dataframe named segment  grouped by
segment_key and the aggregated values
#as in create_segment_dic

segment =
delhivery.groupby('segment_key').agg(create_segment_dict).reset_index(
)

#sorting segment by 'segment_key' and 'od_end_time' ensuring
thatsegments within the same trip are ordered by their
#end timesfrom earliest to latest.

segment = segment.sort_values(by=['segment_key','od_end_time'],
ascending=True).reset_index()

segment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26368 entries, 0 to 26367
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   index                           26368 non-null  int64
 1   segment_key                     26368 non-null  object
 2   data                            26368 non-null  object
 3   trip_creation_time              26368 non-null  datetime64[ns]
 4   route_schedule_uuid             26368 non-null  object
 5   route_type                      26368 non-null  object
 6   trip_uuid                       26368 non-null  object
 7   source_center                   26368 non-null  object
```

```
 8   source_name                       26368 non-null  object
 9   destination_center                26368 non-null  object
 10  destination_name                  26368 non-null  object
 11  od_start_time                     26368 non-null  datetime64[ns]
 12  od_end_time                       26368 non-null  datetime64[ns]
 13  start_scan_to_end_scan            26368 non-null  float64
 14  actual_distance_to_destination    26368 non-null  float64
 15  actual_time                       26368 non-null  float64
 16  osrm_time                         26368 non-null  float64
 17  osrm_distance                     26368 non-null  float64
 18  segment_actual_time_sum           26368 non-null  float64
 19  segment_osrm_distance_sum         26368 non-null  float64
 20  segment_osrm_time_sum             26368 non-null  float64
dtypes: datetime64[ns](3), float64(8), int64(1), object(9)
memory usage: 4.2+ MB
```

Grouping and Aggregating at trip level

```
create_trip_dict ={
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_center' : 'first',
    'source_name' : 'first',

    'destination_center' : 'last',
    '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' : 'last',
    'segment_osrm_distance_sum' : 'last',
    'segment_osrm_time_sum' : 'last',
}

#creating a trip_level dataframe named trip grouped by trip_uuid and
the aggregated values
#as in create_trip_dic
```

```
trip =
segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop=Tr
ue)

trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14817 non-null  object
 1   trip_creation_time              14817 non-null  datetime64[ns]
 2   route_schedule_uuid             14817 non-null  object
 3   route_type                      14817 non-null  object
 4   trip_uuid                       14817 non-null  object
 5   source_center                   14817 non-null  object
 6   source_name                     14817 non-null  object
 7   destination_center              14817 non-null  object
 8   destination_name                14817 non-null  object
 9   od_start_time                   14817 non-null  datetime64[ns]
 10  od_end_time                     14817 non-null  datetime64[ns]
 11  start_scan_to_end_scan          14817 non-null  float64
 12  actual_distance_to_destination  14817 non-null  float64
 13  actual_time                     14817 non-null  float64
 14  osrm_time                       14817 non-null  float64
 15  osrm_distance                   14817 non-null  float64
 16  segment_actual_time_sum         14817 non-null  float64
 17  segment_osrm_distance_sum       14817 non-null  float64
 18  segment_osrm_time_sum           14817 non-null  float64
dtypes: datetime64[ns](3), float64(8), object(8)
memory usage: 2.1+ MB
```

```
#we can check for actual_time, osrm_time,osrm_distance of trip and
cumulatives of segment using this trip dataframe
# and check how far is the reality in sync with the osrm navigation
system

trip[['actual_time','segment_actual_time_sum']].head()
```

	actual_time	segment_actual_time_sum
0	830.0	820.0
1	96.0	95.0
2	2736.0	2700.0
3	59.0	59.0
4	63.0	63.0

```
trip[['osrm_time','segment_osrm_time_sum']].head()
```

```
   osrm_time  segment_osrm_time_sum
0      388.0                  474.0
1       42.0                   39.0
2     1528.0                 1710.0
3       15.0                   16.0
4       27.0                   26.0
```

```
trip[['osrm_distance','segment_osrm_distance_sum']].head()
```

```
   osrm_distance  segment_osrm_distance_sum
0       544.8027                   649.8528
1        56.9116                    55.9899
2      2072.8556                  2227.5270
3        19.6800                    19.8766
4        29.5696                    29.5697
```

3. FEATURE ENGINEERING

```
# Creating new field od_time_diff_hour(time in mins) -time taken
between od_start_time and od_end_time

trip['od_total_time'] = (trip['od_end_time']-
trip['od_start_time']).dt.total_seconds()/60

#dropping fields od_start_time and od_end_time

trip = trip.drop(['od_start_time','od_end_time'], axis = 1)
```

Split and extract features out of source_name and destination_name

```
def location_to_state(loc):
   return loc.split('(')[1][:-1]

def location_to_city(loc):
    l = loc.split('(')[0].split('_')[0]
    if 'CCU' in loc:
        return 'Kolkata'
    elif 'MAA' in loc.upper():
        return 'Chennai'
    elif ('HBR' in loc.upper()) or ('BLR' in loc.upper()):
        return 'Bengaluru'
    elif 'FBD' in loc.upper():
        return 'Faridabad'
    elif 'BOM' in loc.upper():
        return 'Mumbai'
```

```
    elif 'DEL' in loc.upper():
        return 'Delhi'
    elif 'OK' in loc.upper():
        return 'Delhi'
    elif 'GZB' in loc.upper():
        return 'Ghaziabad'
    elif 'GGN' in loc.upper():
        return 'Gurgaon'
    elif 'AMD' in loc.upper():
        return 'Ahmedabad'
    elif 'CJB' in loc.upper():
        return 'Coimbatore'
    elif 'HYD' in loc.upper():
        return 'Hyderabad'
    return l

def location_to_place(loc):
    if 'HBR' in loc:
        return 'HBR Layout PC'
    else:
            # we will remove state
        loc = loc.split('(')[0]

        len_ = len(loc.split('_'))

        if len_ >= 3:
            return loc.split('_')[1]

        # small cities have same city and place name
        if len_ == 2:
            return loc.split('_')[0]

        return loc.split(' ')[0]

def location_to_code(loc):
    # we will remove state
    loc = loc.split('(')[0]

    if len(loc.split('_')) >= 3:
        return loc.split('_')[-1]

    return 'none'

trip['destination_state'] = trip['destination_name'].apply(lambda x:
location_to_state(x))
trip['destination_city']  = trip['destination_name'].apply(lambda x:
location_to_city(x))
trip['destination_place'] = trip['destination_name'].apply(lambda x:
location_to_place(x))
trip['destination_code']  = trip['destination_name'].apply(lambda x:
location_to_code(x))
```

```
trip[['destination_state','destination_city','destination_place','dest
ination_code']].head()
```

```
  destination_state destination_city destination_place
destination_code
0     Uttar Pradesh           Kanpur           Central                6

1         Karnataka       Doddablpur          ChikaDPP                D

2           Haryana          Gurgaon          Bilaspur               HB

3       Maharashtra           Mumbai            MiraRd               IP

4         Karnataka           Sandur          WrdN1DPP                D
```

```
trip['source_state'] = trip['source_name'].apply(lambda x:
location_to_state(x))
trip['source_city']  = trip['source_name'].apply(lambda x:
location_to_city(x))
trip['source_place'] = trip['source_name'].apply(lambda x:
location_to_place(x))
trip['source_code']  = trip['source_name'].apply(lambda x:
location_to_code(x))

trip[['source_state','source_city','source_place','source_code']].head
()
```

```
    source_state  source_city source_place source_code
0  Uttar Pradesh       Kanpur      Central          6
1      Karnataka   Doddablpur     ChikaDPP          D
2        Haryana      Gurgaon     Bilaspur         HB
3    Maharashtra  Mumbai Hub       Mumbai        none
4      Karnataka      Bellary      Bellary        none
```

Splitting trip_creation_time according to year, month, day and hour

```
trip['trip_creation_year'] = trip['trip_creation_time'].dt.year
trip['trip_creation_month']= trip['trip_creation_time'].dt.month
trip['trip_creation_day'] = trip['trip_creation_time'].dt.day
trip['trip_creation_hour'] = trip['trip_creation_time'].dt.hour

trip[['trip_creation_time','trip_creation_year','trip_creation_month',
'trip_creation_day','trip_creation_hour']].head()
```

```
          trip_creation_time  trip_creation_year  trip_creation_month
\
0 2018-09-12 00:00:16.535741                2018                    9
```

```
1 2018-09-12 00:00:22.886430                2018                    9

2 2018-09-12 00:00:33.691250                2018                    9

3 2018-09-12 00:01:00.113710                2018                    9

4 2018-09-12 00:02:09.740725                2018                    9


   trip_creation_day  trip_creation_hour
0                 12                   0
1                 12                   0
2                 12                   0
3                 12                   0
4                 12                   0
```

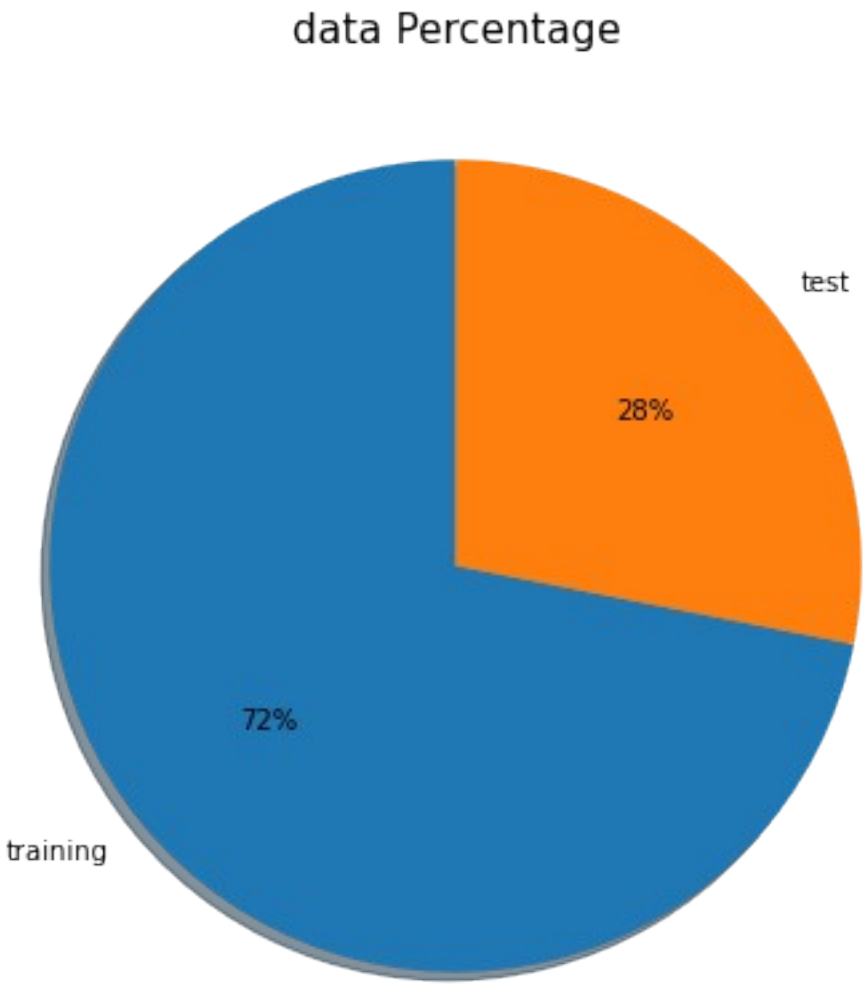
4. IN-DEPTH ANALYSIS

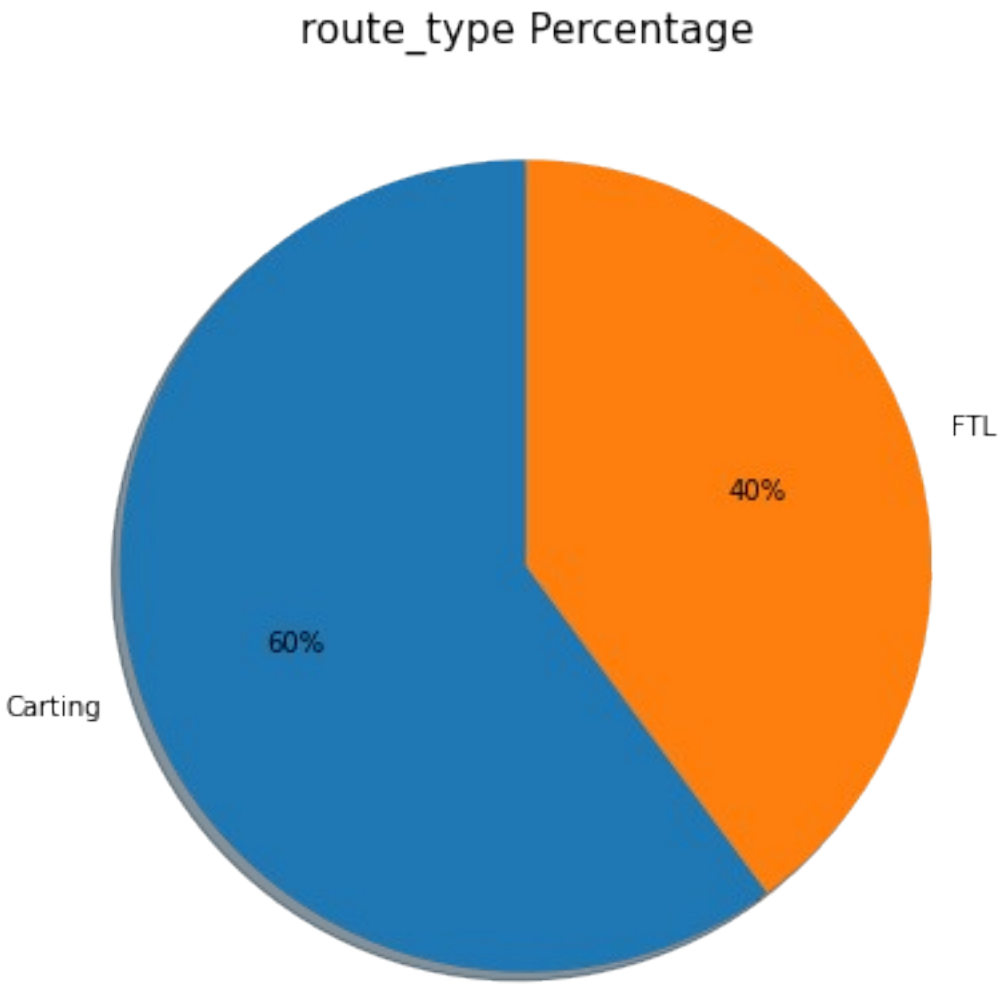
Calculate the time taken between od_start_time and od_end_time and keep it as a feature.

```
trip['od_total_time']

0        1260.604421
1          58.832388
2         834.638929
3         100.494935
4         152.012914
             ...
14812     152.787843
14813      60.590521
14814     248.409092
14815     105.656951
14816     287.474007
Name: od_total_time, Length: 14817, dtype: float64

trip.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 30 columns):
```

```
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   data                              14817 non-null  object
 1   trip_creation_time                14817 non-null  datetime64[ns]
 2   route_schedule_uuid               14817 non-null  object
 3   route_type                        14817 non-null  object
 4   trip_uuid                         14817 non-null  object
 5   source_center                     14817 non-null  object
 6   source_name                       14817 non-null  object
 7   destination_center                14817 non-null  object
 8   destination_name                  14817 non-null  object
 9   start_scan_to_end_scan            14817 non-null  float64
 10  actual_distance_to_destination    14817 non-null  float64
 11  actual_time                       14817 non-null  float64
 12  osrm_time                         14817 non-null  float64
 13  osrm_distance                     14817 non-null  float64
 14  segment_actual_time_sum           14817 non-null  float64
 15  segment_osrm_distance_sum         14817 non-null  float64
 16  segment_osrm_time_sum             14817 non-null  float64
 17  od_total_time                     14817 non-null  float64
 18  destination_state                 14817 non-null  object
 19  destination_city                  14817 non-null  object
 20  destination_place                 14817 non-null  object
 21  destination_code                  14817 non-null  object
 22  source_state                      14817 non-null  object
 23  source_city                       14817 non-null  object
 24  source_place                      14817 non-null  object
 25  source_code                       14817 non-null  object
 26  trip_creation_year                14817 non-null  int64
 27  trip_creation_month               14817 non-null  int64
 28  trip_creation_day                 14817 non-null  int64
 29  trip_creation_hour                14817 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(4), object(16)
memory usage: 3.4+ MB

trip_num_cols = list(trip.dtypes.reset_index()
[(trip.dtypes.reset_index()[0]=='float64')]['index'])
trip_num_cols

['start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time_sum',
 'segment_osrm_distance_sum',
 'segment_osrm_time_sum',
 'od_total_time']
```

```
for col in cat_cols:
    fig = plt.figure(figsize=(10,7))
    plt.pie(trip[col].value_counts().values, labels =
list(trip[col].value_counts().index), autopct='%.0f%%', shadow= True,
                  startangle = 90)
    plt.title(f"{col} Percentage", fontsize=15)
    plt.show()
```



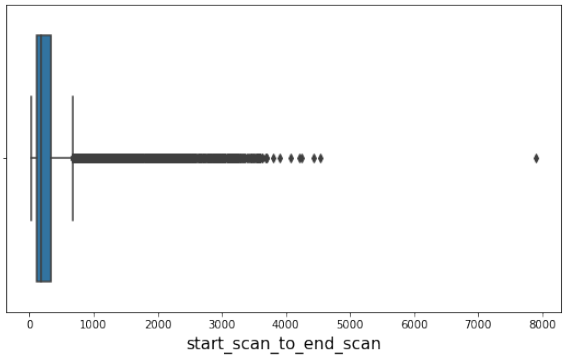


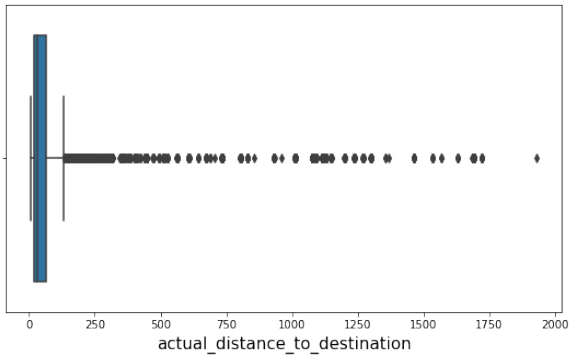
Insight:

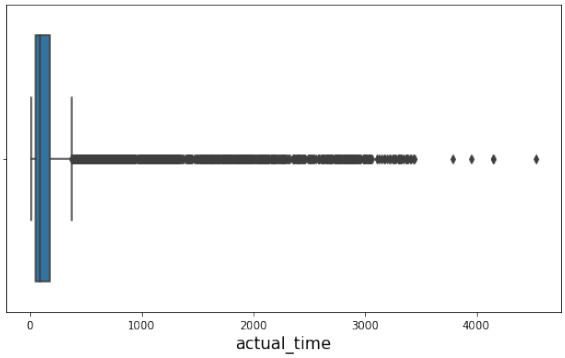
```
Training data is more than testing data.
60% of trasportation type is carting and 40% is Full Truck Load (FTL).
```

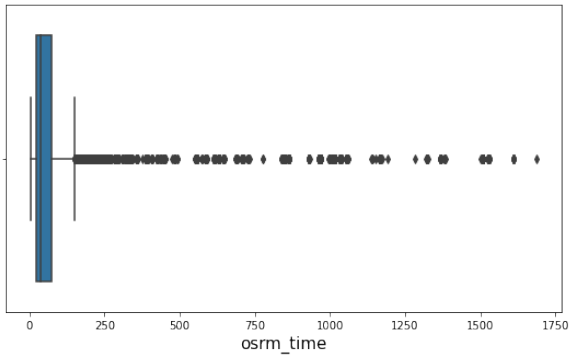
Outlier detection and treatment

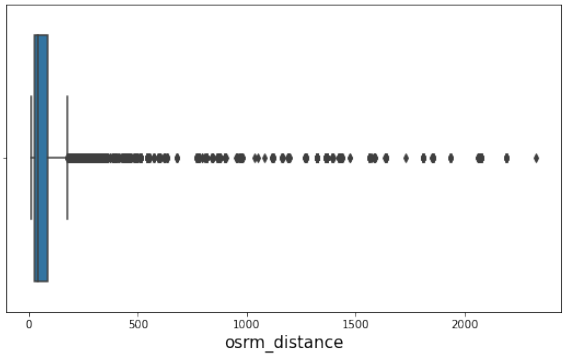
```
plt.figure(figsize=(20,30))
for i,col in enumerate(trip_num_cols):
    plt.subplot(int(len(trip_num_cols)/2)+1, 2, i+1)
    sns.boxplot(x= trip[col])
    plt.xlabel(col, fontsize=15)
plt.show()
```

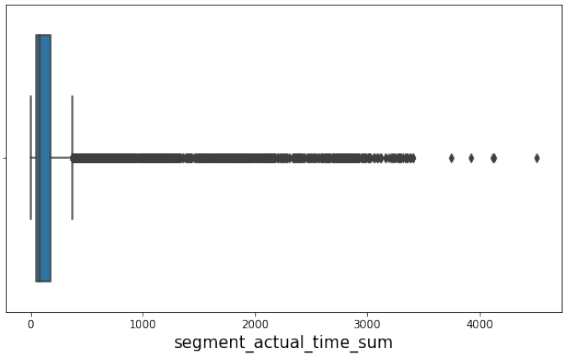

0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan

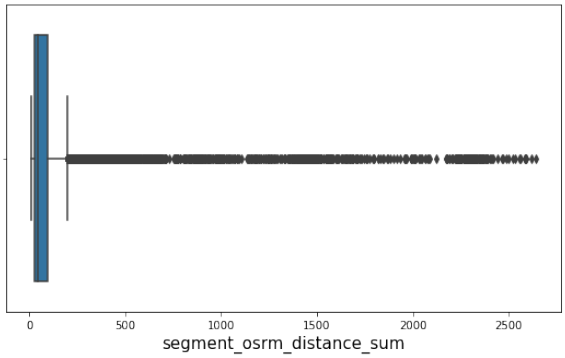

0
250
500
750
1000
1250
1500
1750
2000
actual_distance_to_destination

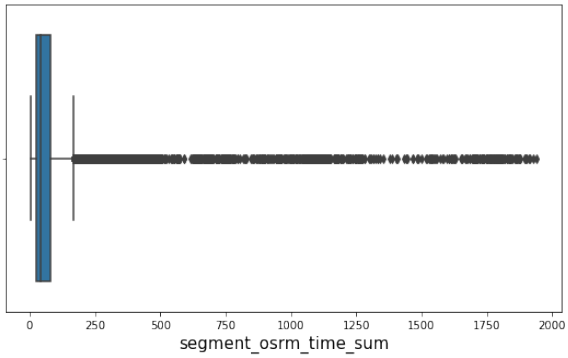

0
1000
2000
3000
4000
actual_time

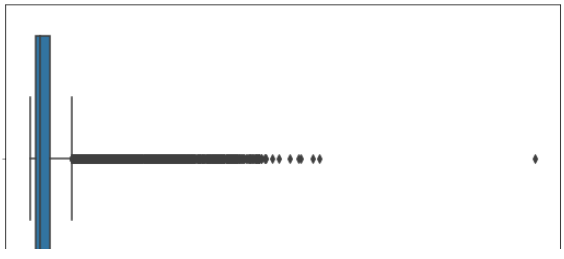

0
250
500
750
1000
1250
1500
1750
osrm_time


0
500
1000
1500
2000
osrm_distance


0
1000
2000
3000
4000
segment_actual_time_sum


0
500
1000
1500
2000
2500
segment_osrm_distance_sum


0
250
500
750
1000
1250
1500
1750
2000
segment_osrm_time_sum



There are many outliers in all the neumerical columns which should be treated.

```
# detecting outliers
for col in trip_num_cols:
    Q1 = trip[col].quantile(0.25)
    Q3 = trip[col].quantile(0.75)
    IQR = Q3-Q1
    LB = Q1-1.5*IQR
    UB = Q3+1.5*IQR
    outliers = trip[(trip[col]<LB) | (trip[col]>UB)]
    print("Column:",col)
    print("Q1:",Q1)
    print("Q3:",Q3)
    print("LB:",LB)
    print("UB:",UB)
    print("IQR:",IQR)
    print("Outliers:", outliers.shape[0])
    print("-"*25)
```

```
Column: start_scan_to_end_scan
Q1: 104.0
Q3: 334.0
LB: -241.0
UB: 679.0
IQR: 230.0
Outliers: 1588
-------------------------
Column: actual_distance_to_destination
Q1: 20.098971767647946
Q3: 65.81230959743395
LB: -48.47103497703106
UB: 134.38231634211297
IQR: 45.713337829786006
Outliers: 2132
-------------------------
Column: actual_time
Q1: 51.0
Q3: 182.0
LB: -145.5
UB: 378.5
IQR: 131.0
Outliers: 1868
-------------------------
Column: osrm_time
Q1: 23.0
Q3: 74.0
LB: -53.5
UB: 150.5
IQR: 51.0
Outliers: 1853
```

```
-------------------------
Column: osrm_distance
Q1: 26.0408
Q3: 86.1399
LB: -64.10784999999998
UB: 176.28855
IQR: 60.09909999999999
Outliers: 1940
-------------------------
Column: segment_actual_time_sum
Q1: 50.0
Q3: 180.0
LB: -145.0
UB: 375.0
IQR: 130.0
Outliers: 1873
-------------------------
Column: segment_osrm_distance_sum
Q1: 27.116
Q3: 94.8538
LB: -74.49070000000002
UB: 196.46050000000002
IQR: 67.73780000000001
Outliers: 1941
-------------------------
Column: segment_osrm_time_sum
Q1: 24.0
Q3: 81.0
LB: -61.5
UB: 166.5
IQR: 57.0
Outliers: 2025
-------------------------
Column: od_total_time
Q1: 104.16200571666667
Q3: 334.75005031666666
LB: -241.7200611833333
UB: 680.6321172166666
IQR: 230.5880446
Outliers: 1584
-------------------------
```

Insight:

```
There are many outliers which might be of many reasons. Sometimes they might be true ones ,
removing them would effect the data. So outliers are left untreated.
```

Performing one-hot encoding on categorical features

```
cat_cols
```

```
['data', 'route_type']
```

```
print("value counts before label encoding")
print("data")
print(trip['data'].value_counts())
print("route_type")
print(trip['route_type'].value_counts())
```

```
value counts before label encoding
data
training    10654
test         4163
Name: data, dtype: int64
route_type
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

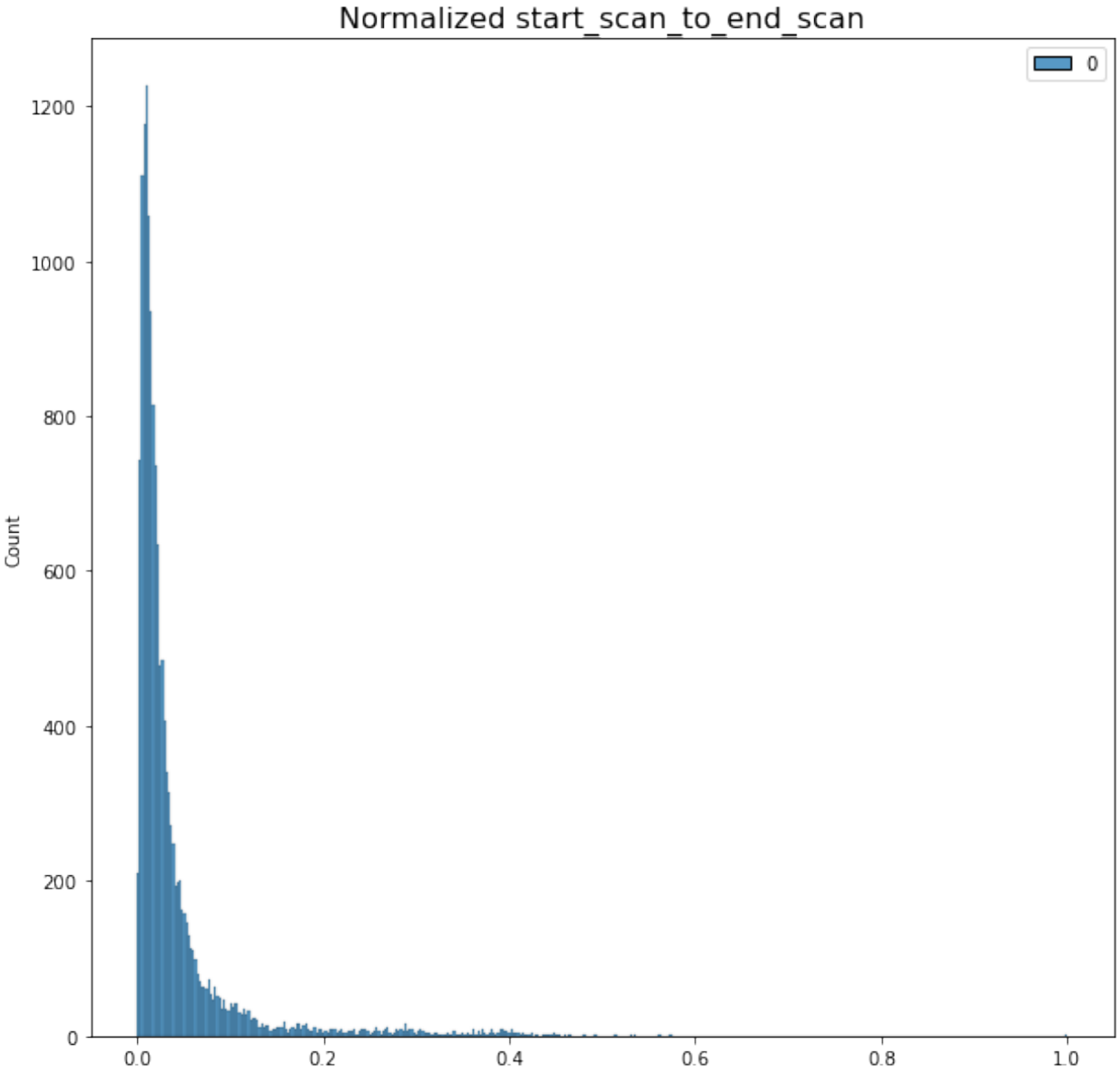
```
label_encoder = LabelEncoder()
trip['data'] = label_encoder.fit_transform(trip['data'])
trip['route_type'] = label_encoder.fit_transform(trip['route_type'])
```

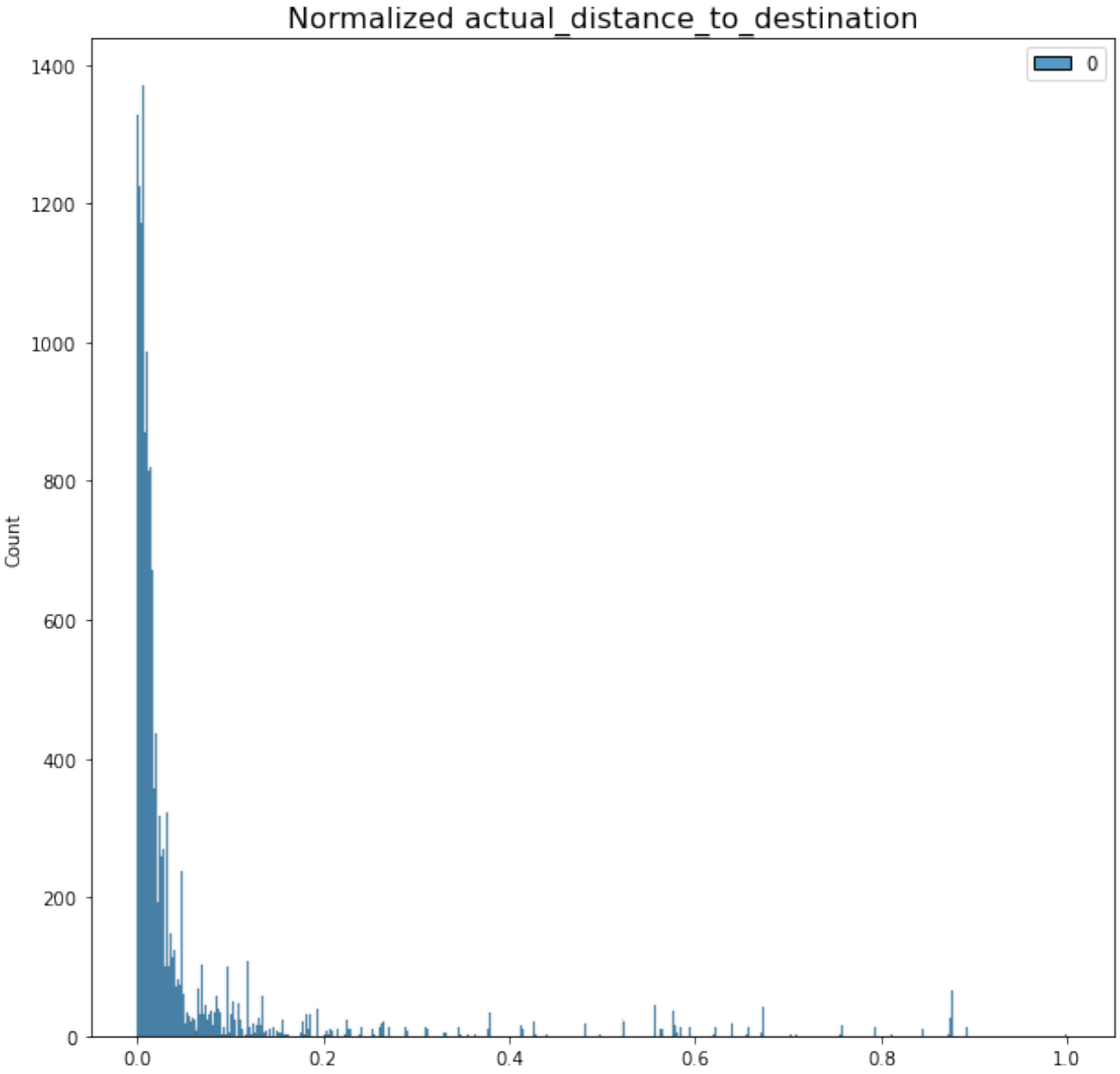
```
print("value counts after label encoding")
print("data")
print(trip['data'].value_counts())
print("route_type")
print(trip['route_type'].value_counts())
```

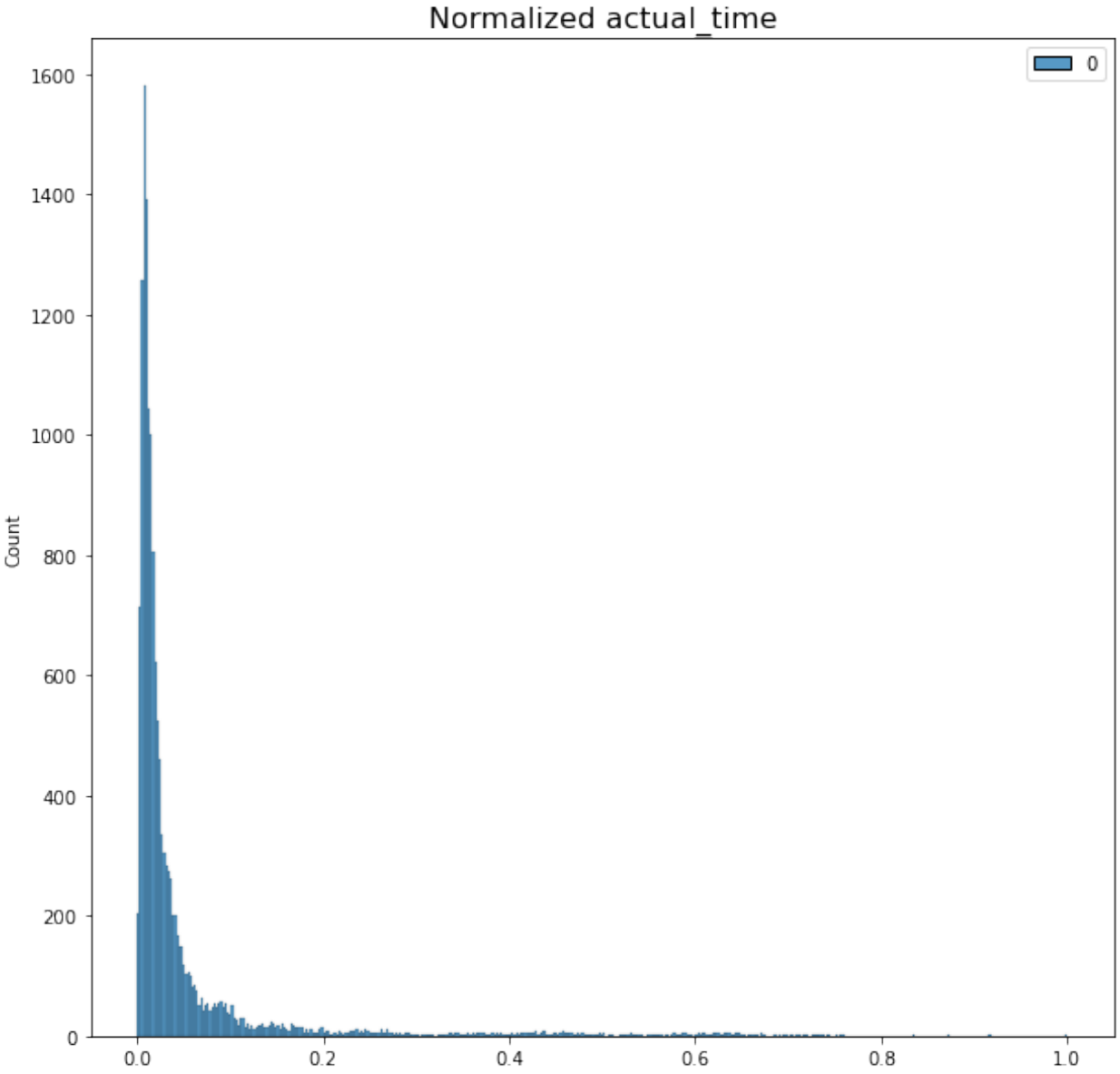
```
value counts after label encoding
data
1    10654
0     4163
Name: data, dtype: int64
route_type
0    8908
1    5909
Name: route_type, dtype: int64
```

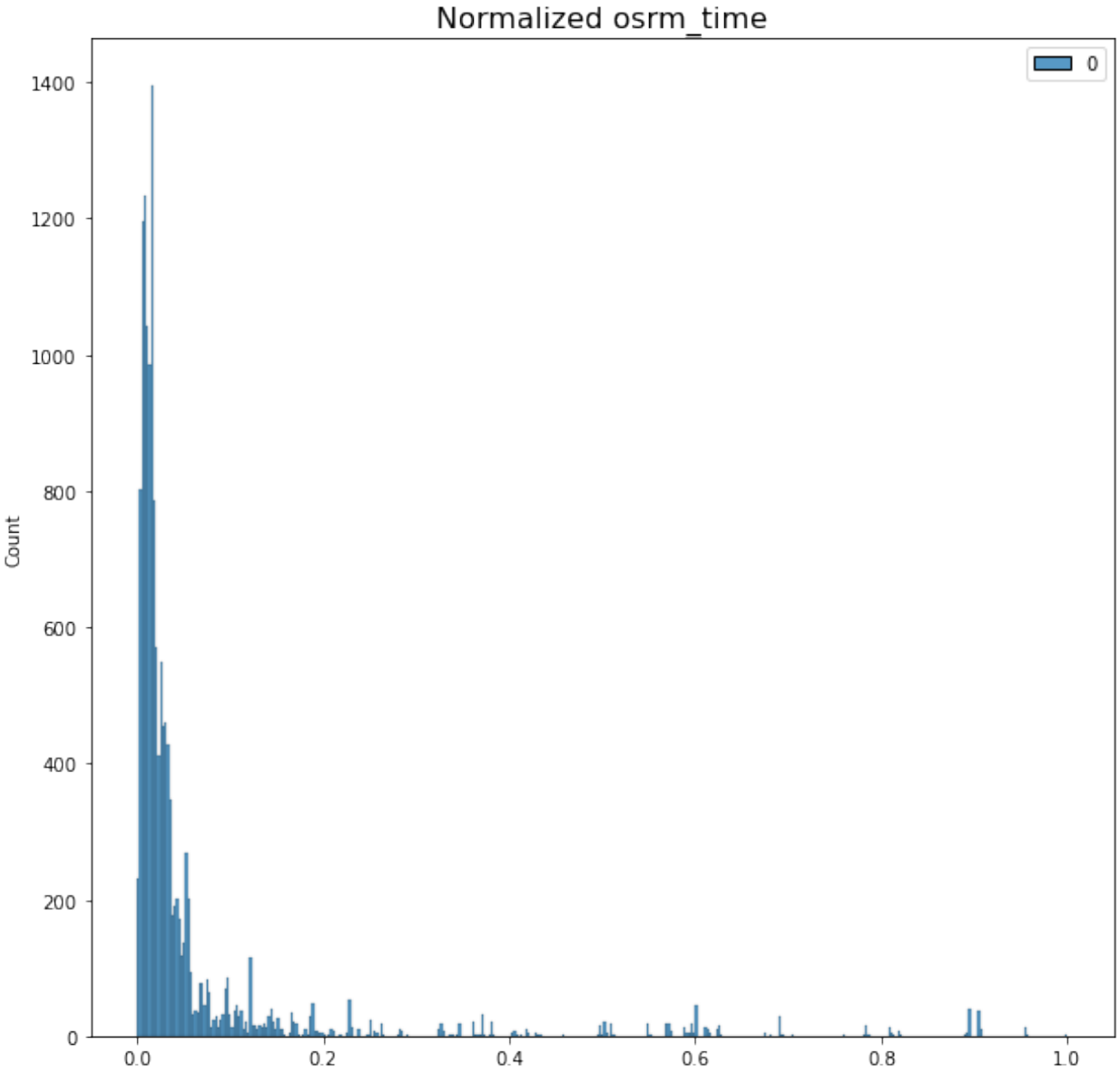
Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

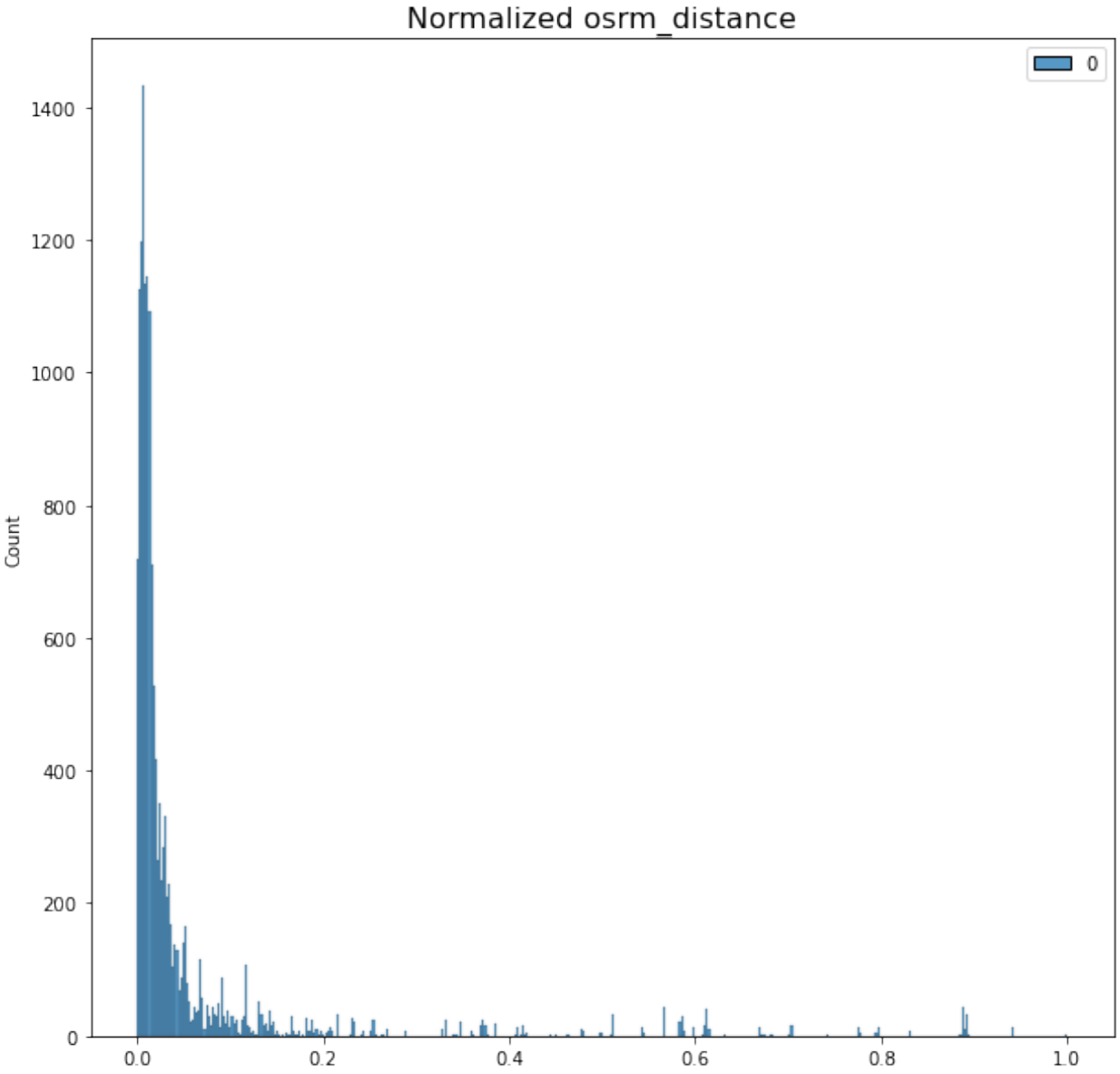
```
# Normalization

for col in trip_num_cols:
    plt.figure(figsize = (10, 10))
    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(trip[col].to_numpy().reshape(-1, 1))
    sns.histplot(scaled)
    plt.title(f"Normalized {col}", fontsize=16)
    #plt.plot()
    plt.show()
```

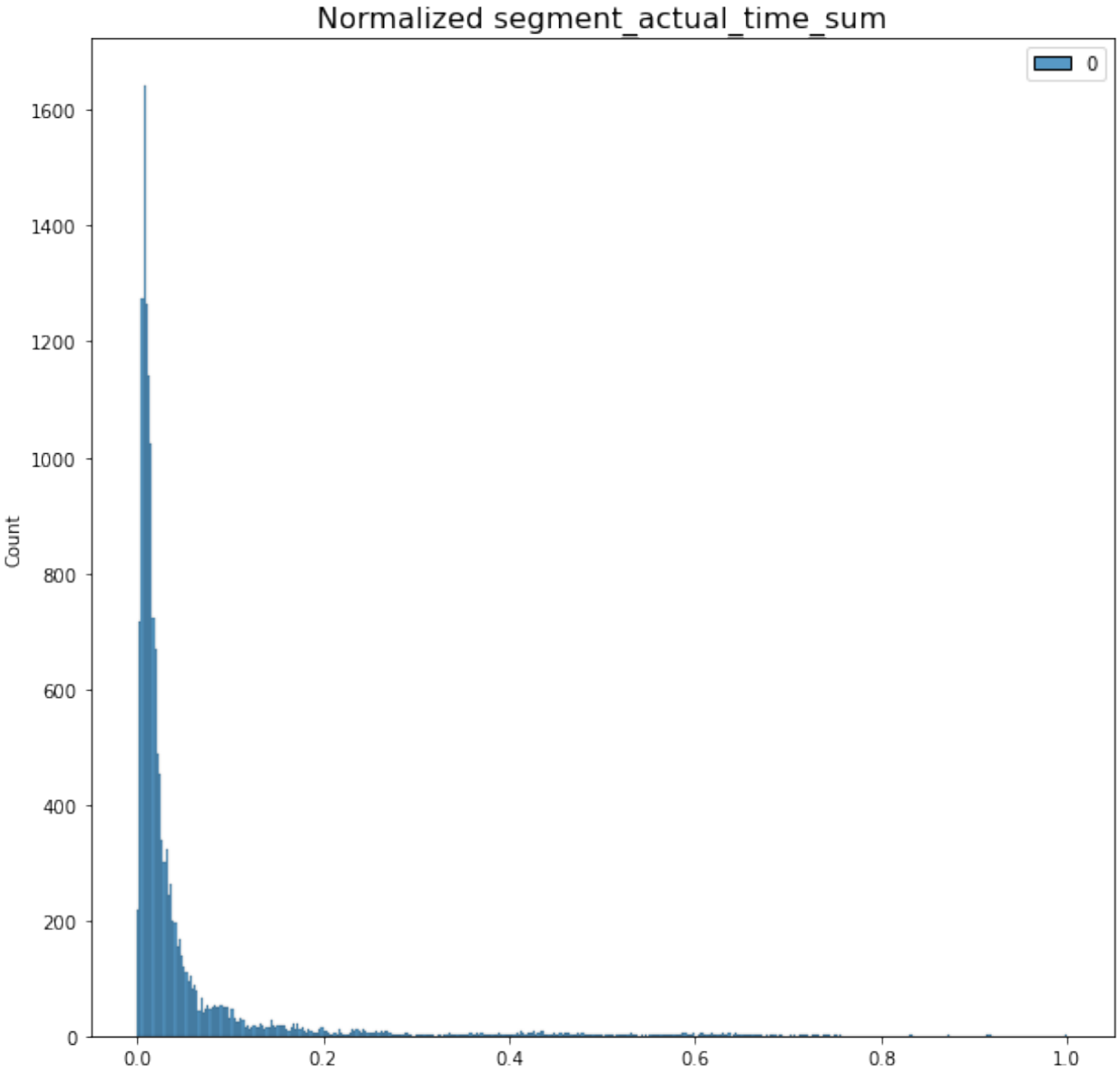

Normalized start_scan_to_end_scan
0
Count
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

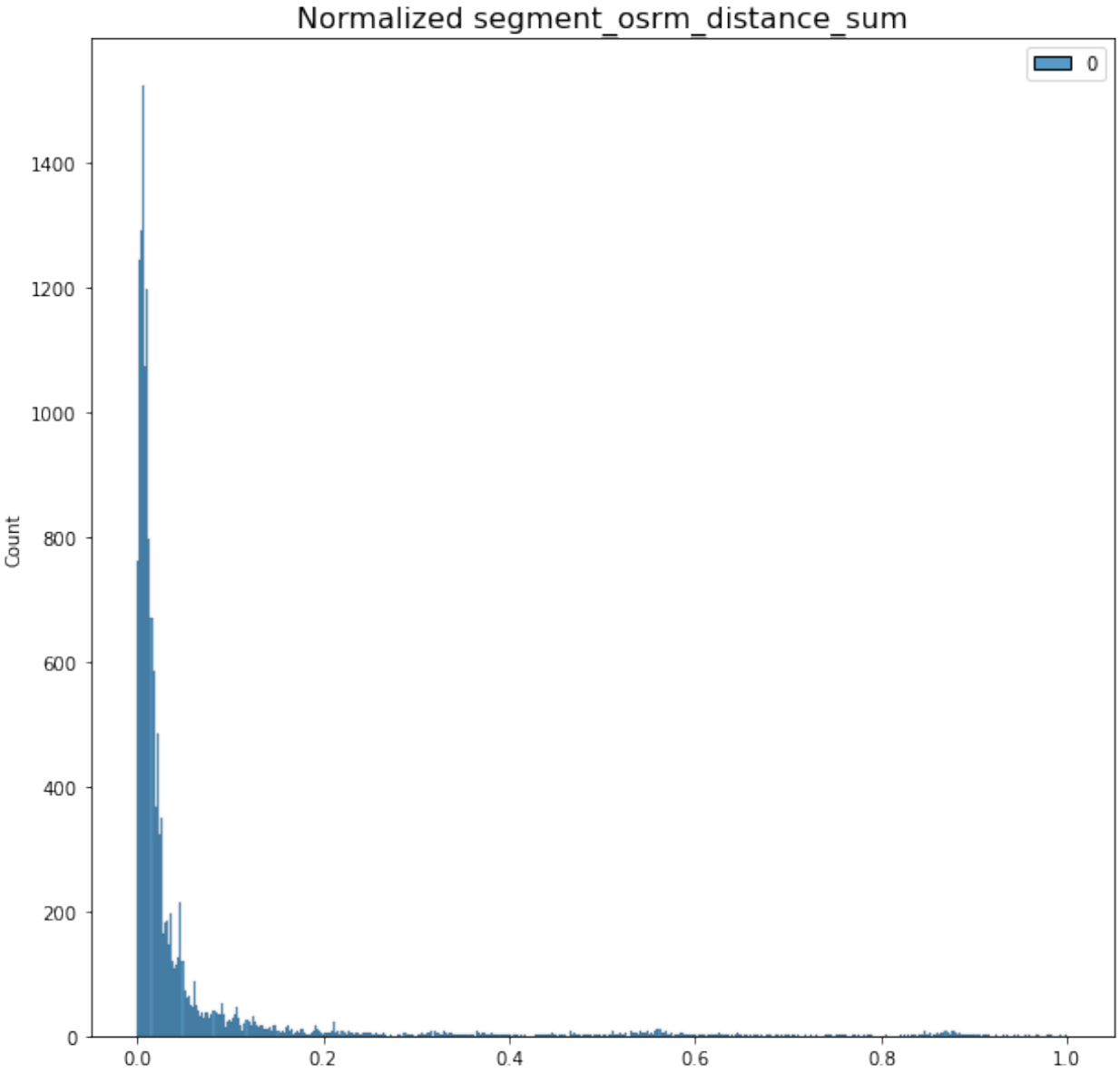

Normalized actual_distance_to_destination
0
Count
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

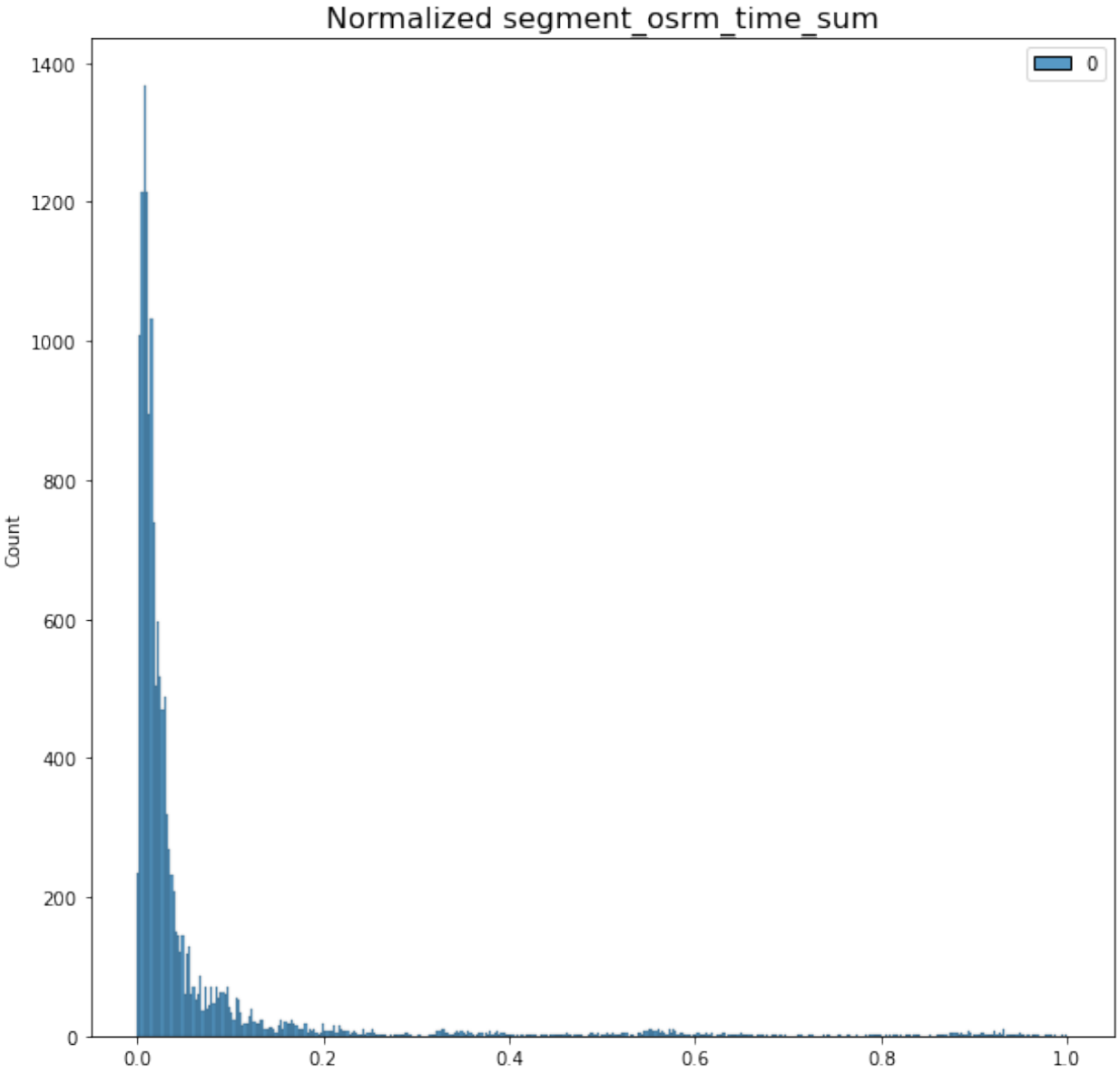

Normalized actual_time
0
Count
1600
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

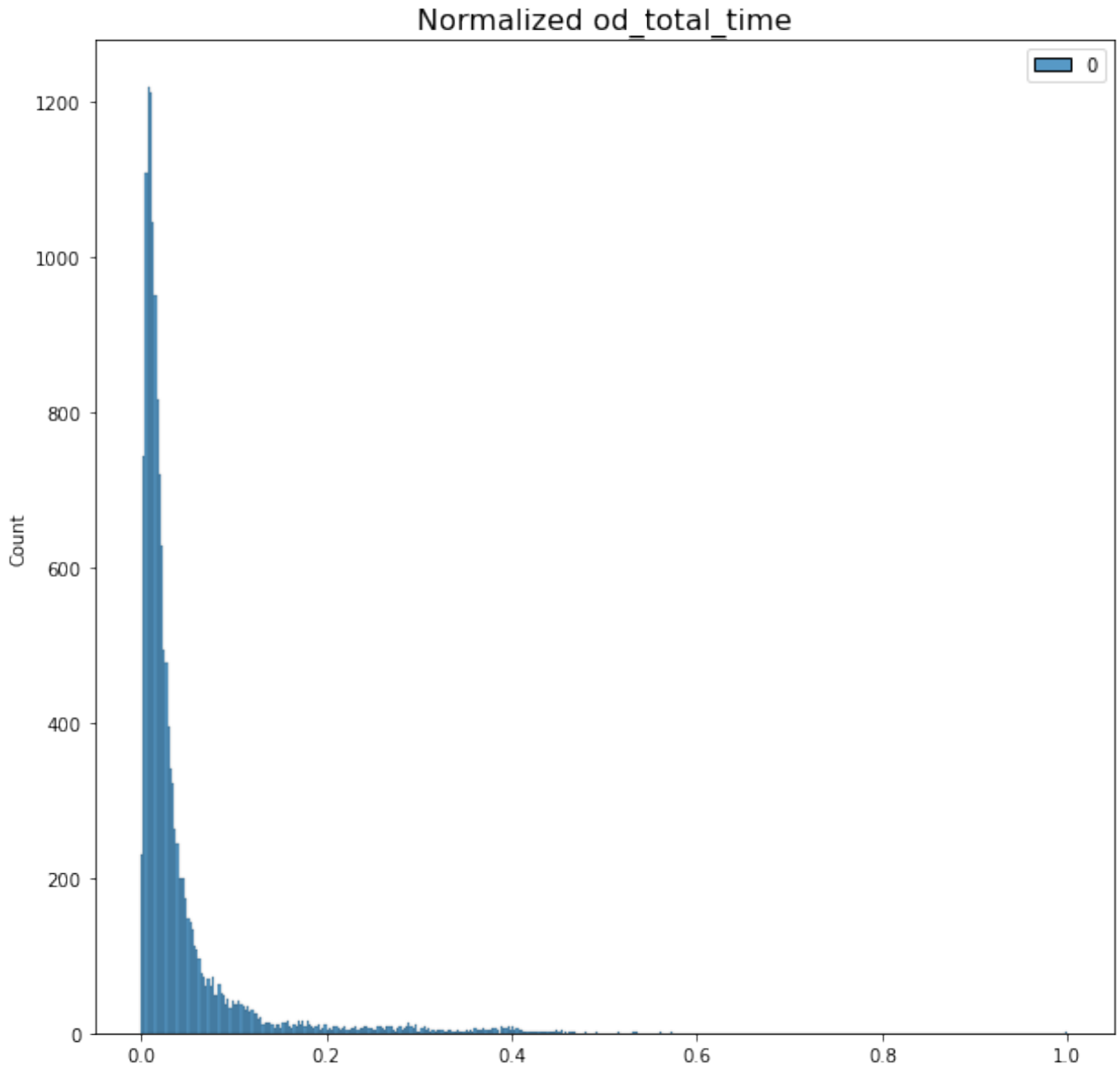

Normalized osrm_time
0
Count
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0


Normalized osrm_distance
0
Count
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

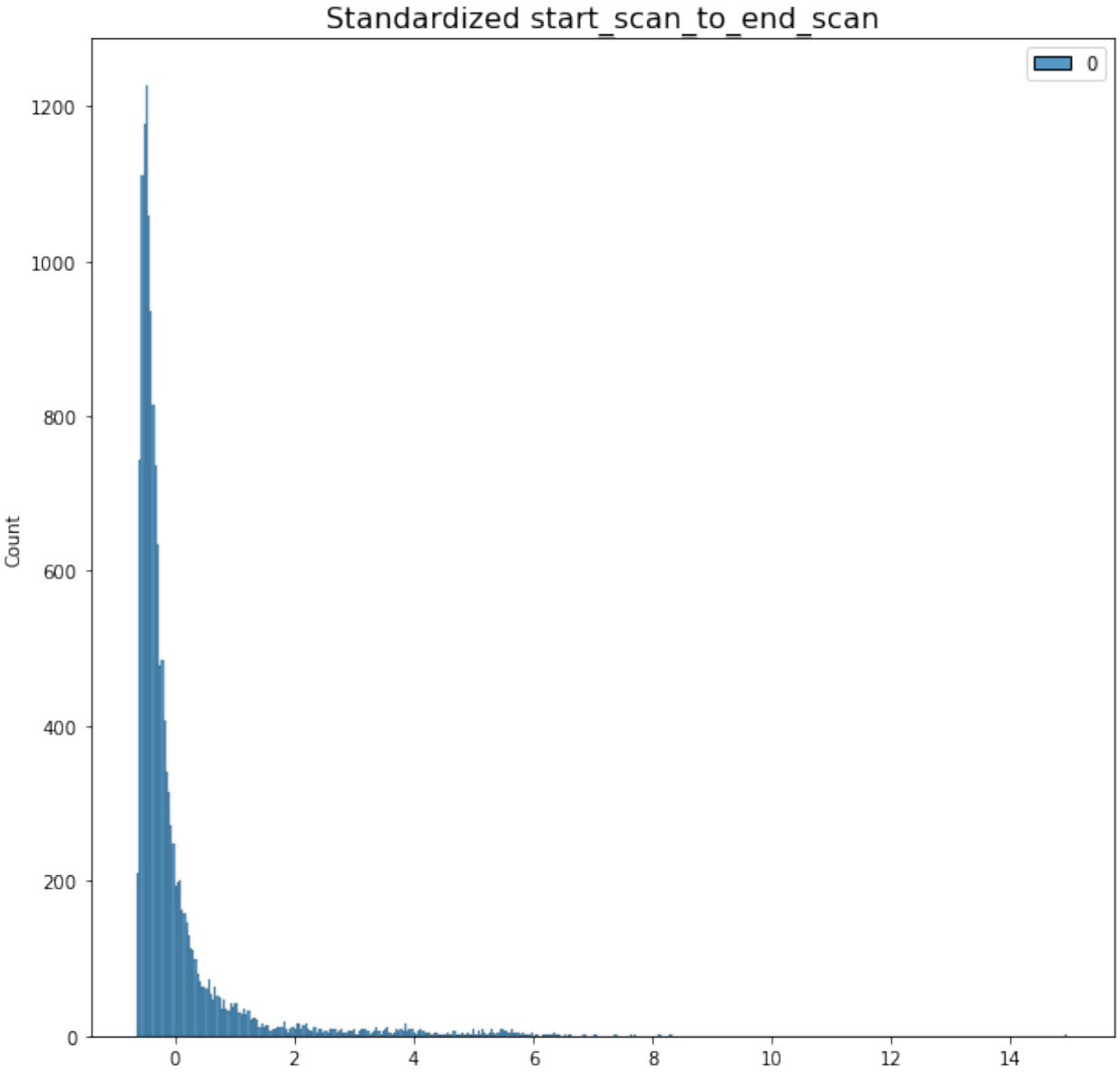

Normalized segment_actual_time_sum
0
Count
1600
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

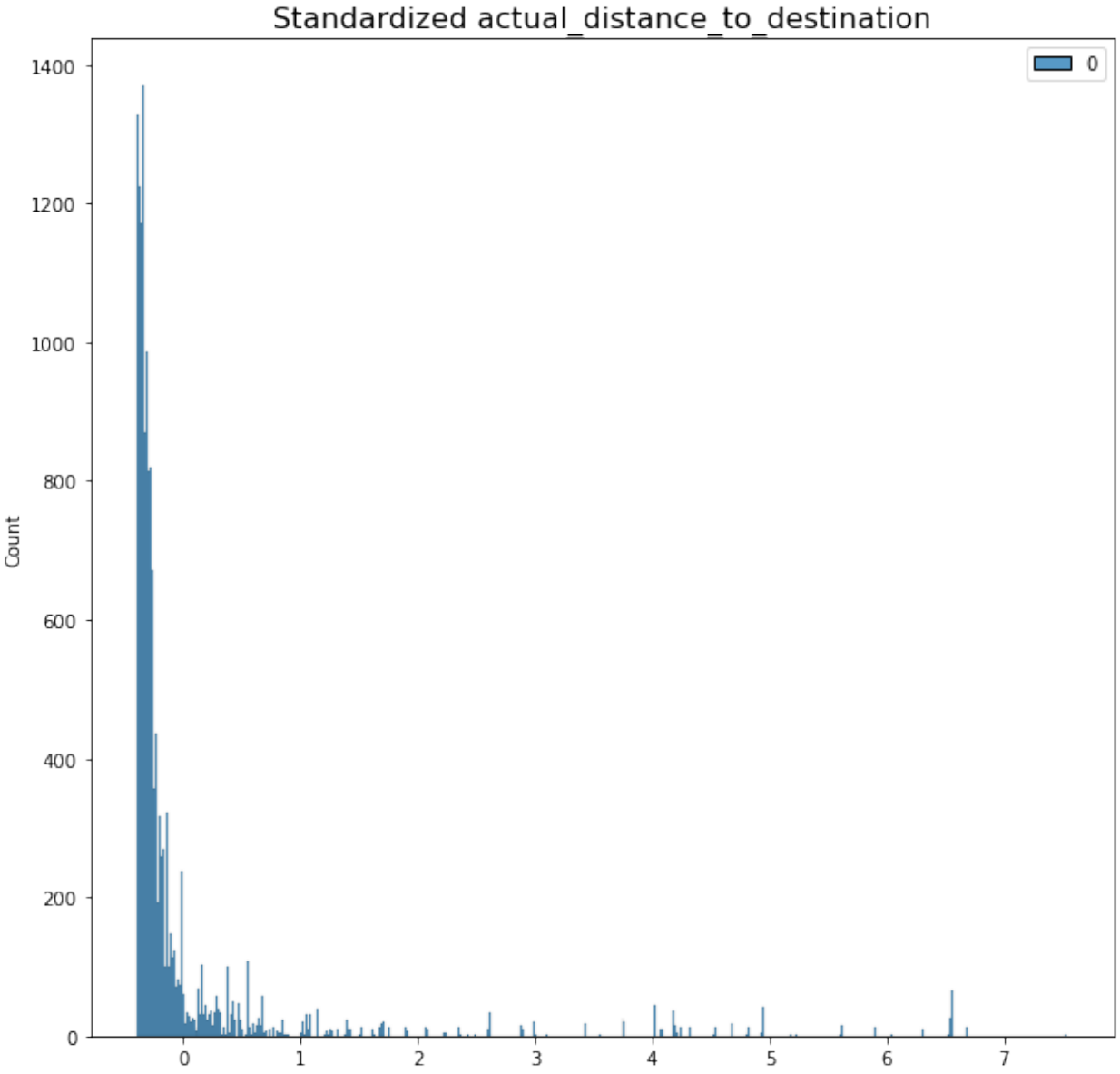

Normalized segment_osrm_distance_sum
0
Count
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

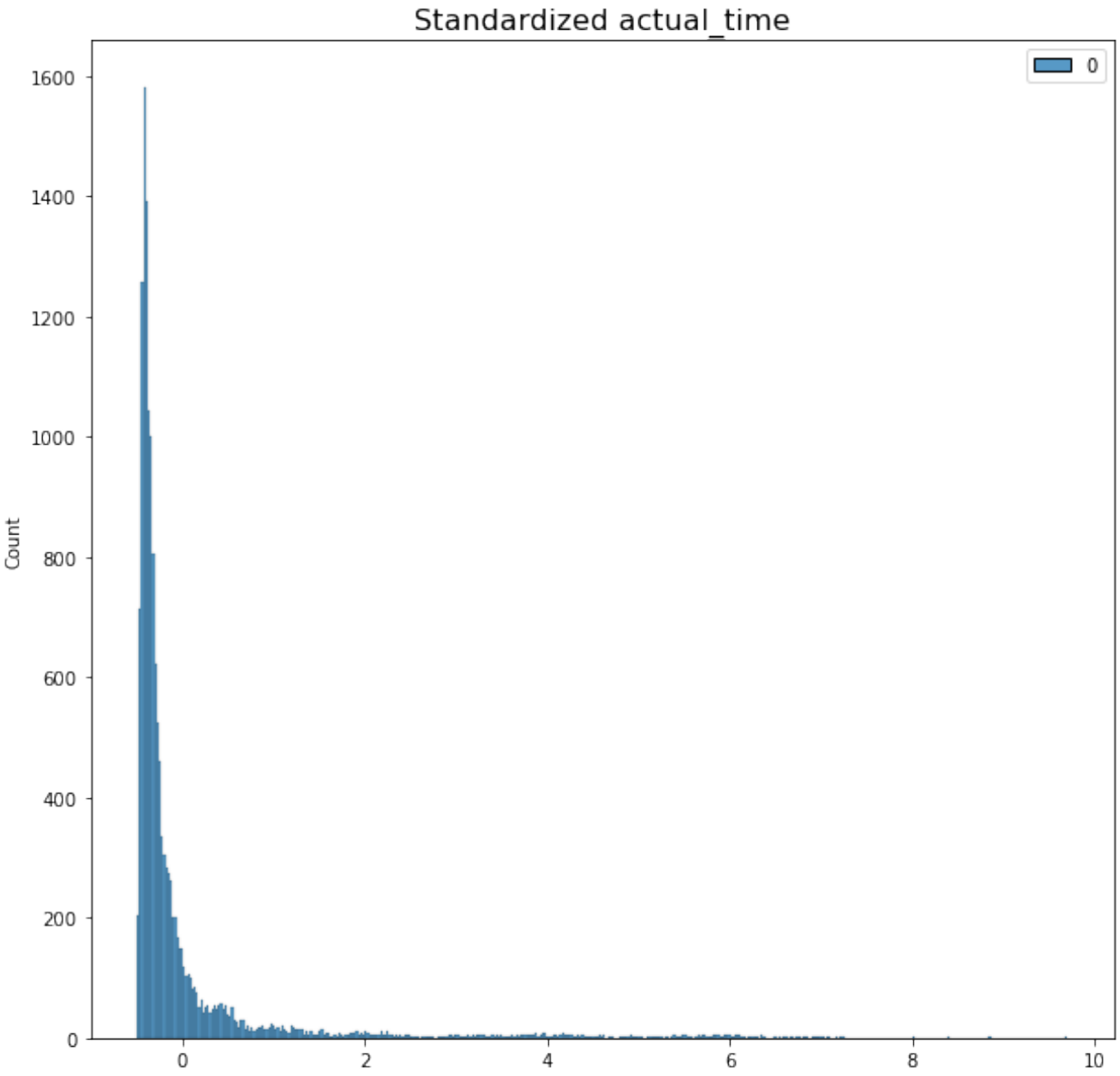

Normalized segment_osrm_time_sum
0
Count
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

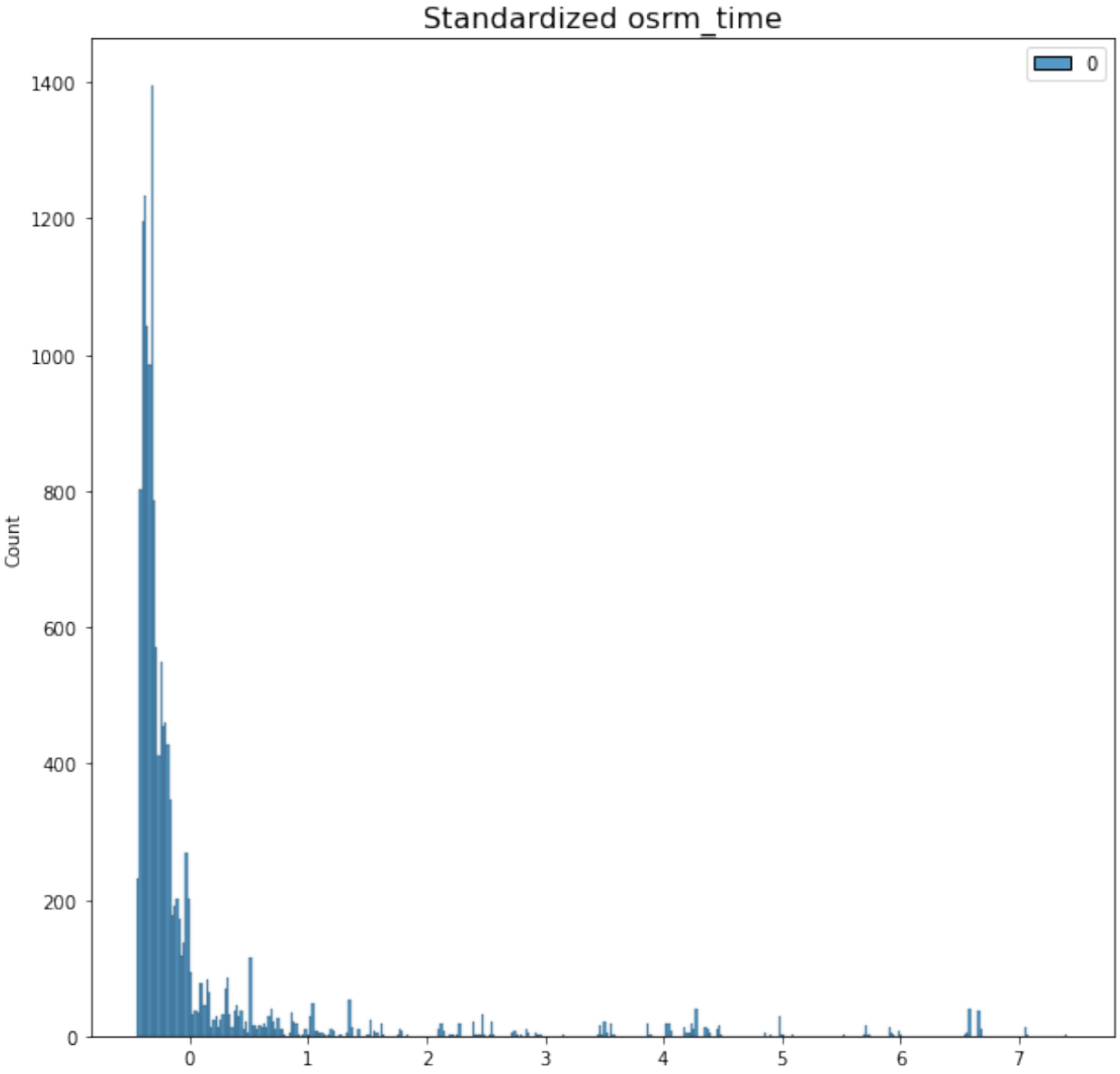


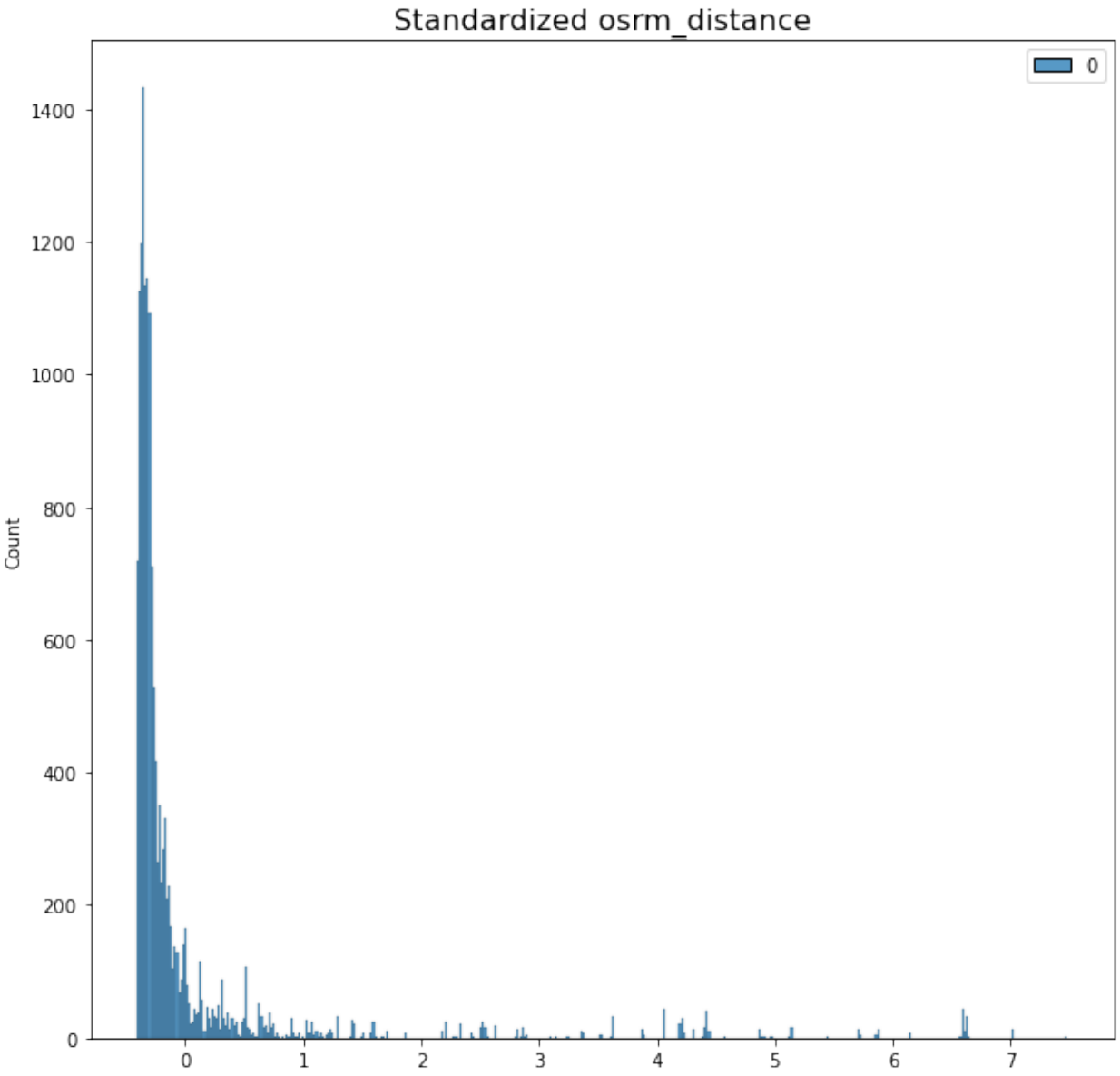

```
# Standardization

for col in trip_num_cols:
    plt.figure(figsize = (10, 10))
    scaler = StandardScaler()
    scaled = scaler.fit_transform(trip[col].to_numpy().reshape(-1, 1))
    sns.histplot(scaled)
    plt.title(f"Standardized {col}", fontsize=16)
    #plt.plot()
    plt.show()
```

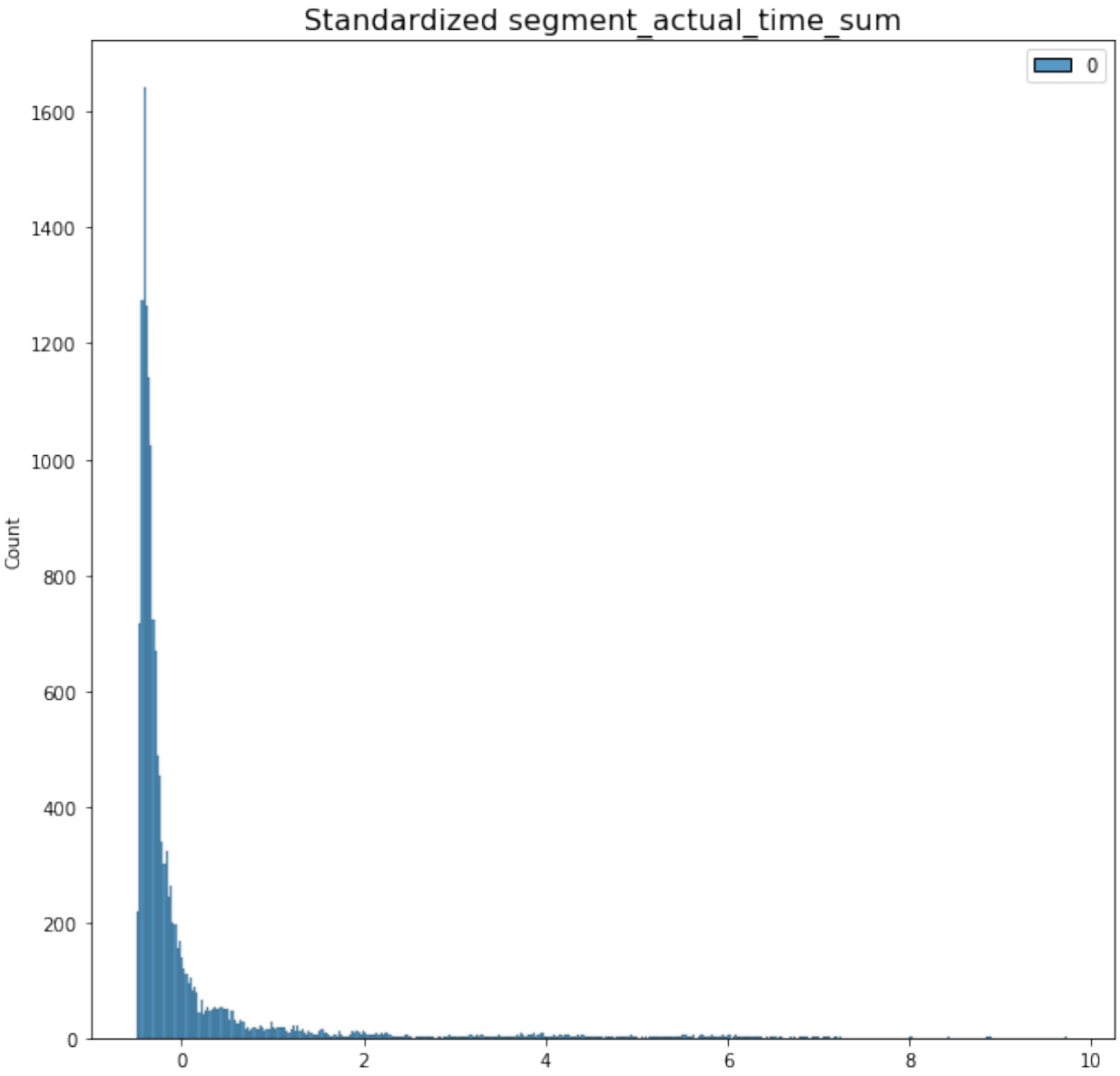

Standardized start_scan_to_end_scan
0
Count
1200
1000
800
600
400
200
0
0
2
4
6
8
10
12
14

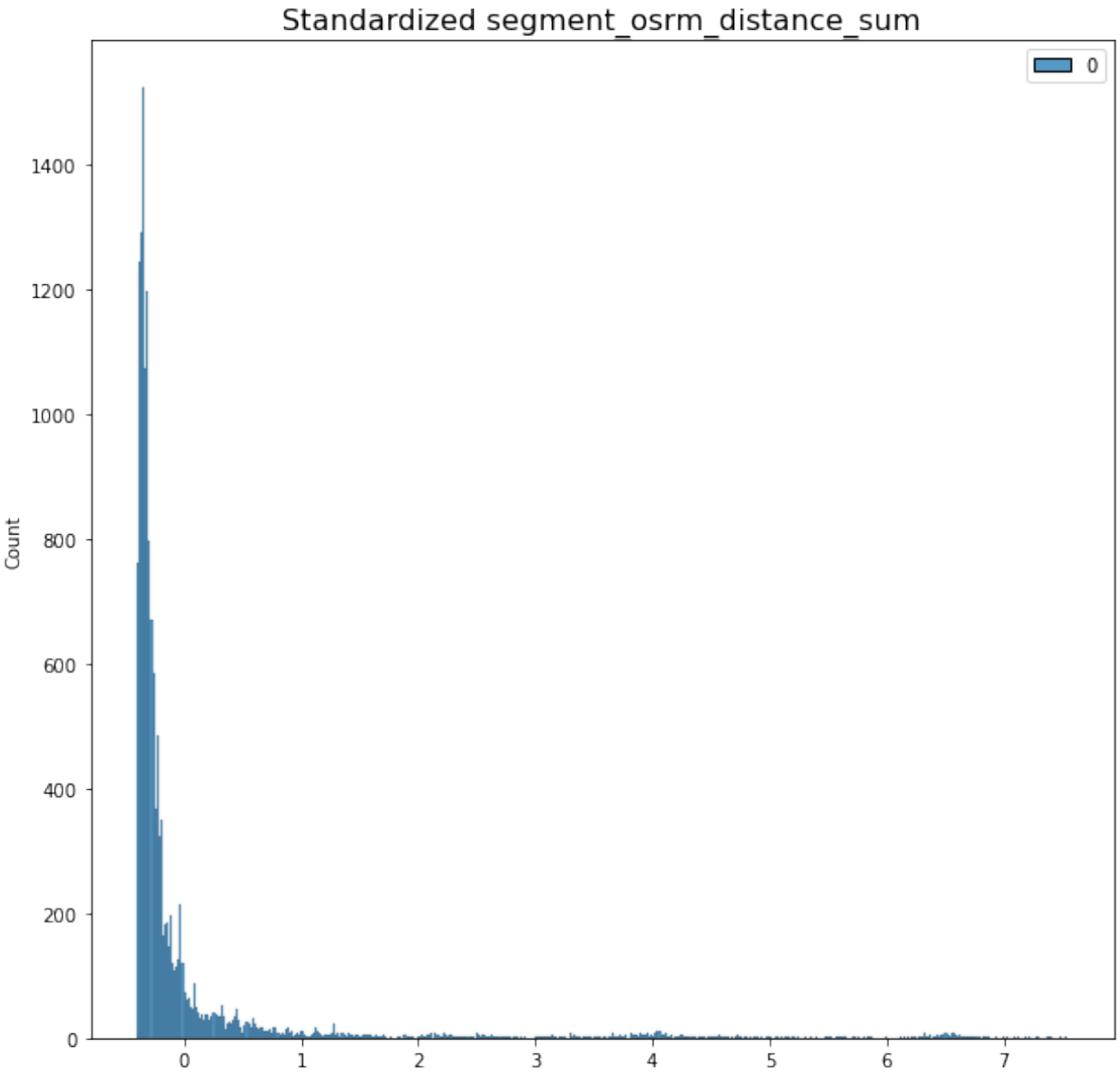

Standardized actual_distance_to_destination
0
Count
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7

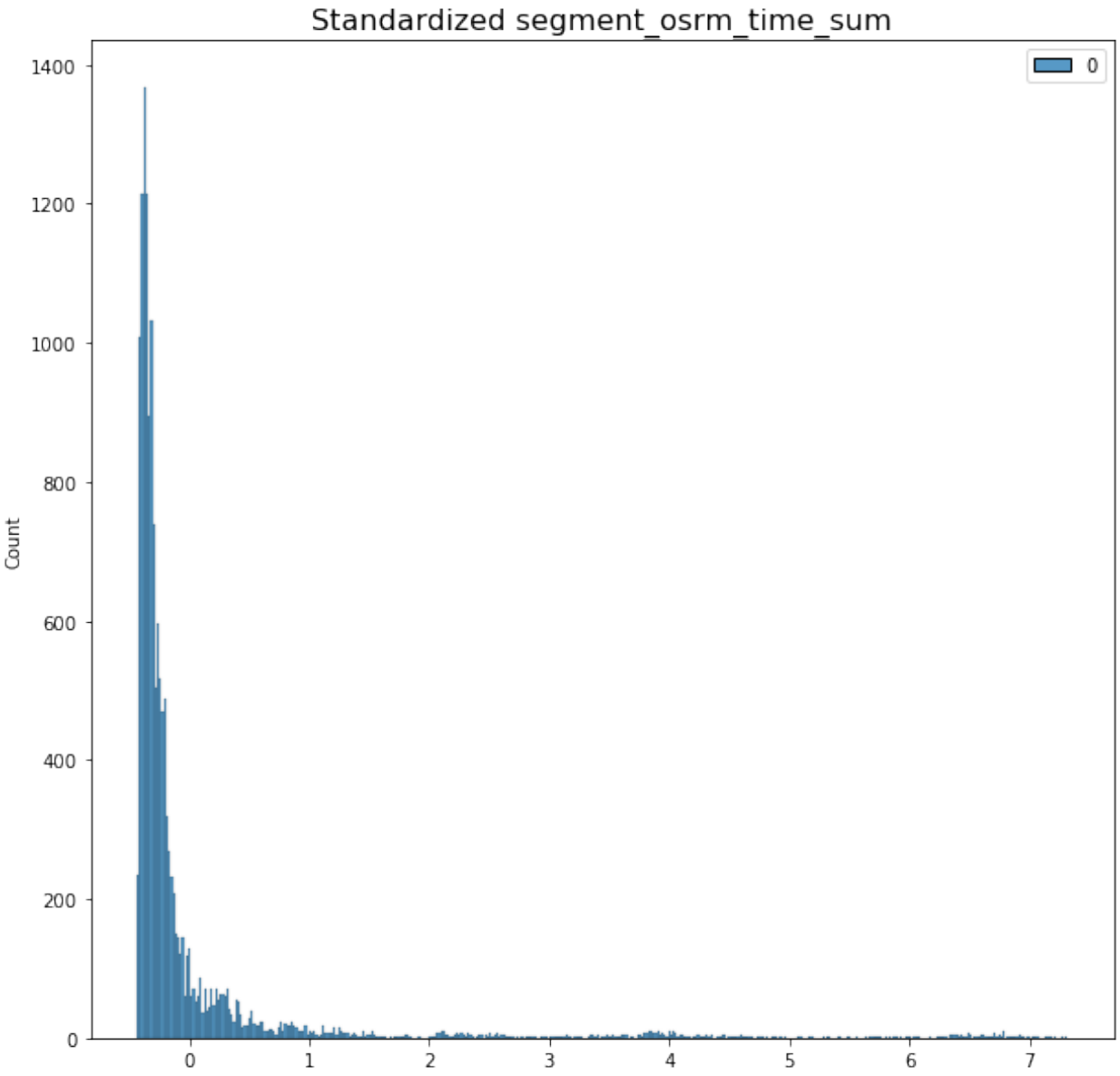

Standardized actual_time
0
Count
1600
1400
1200
1000
800
600
400
200
0
0
2
4
6
8
10

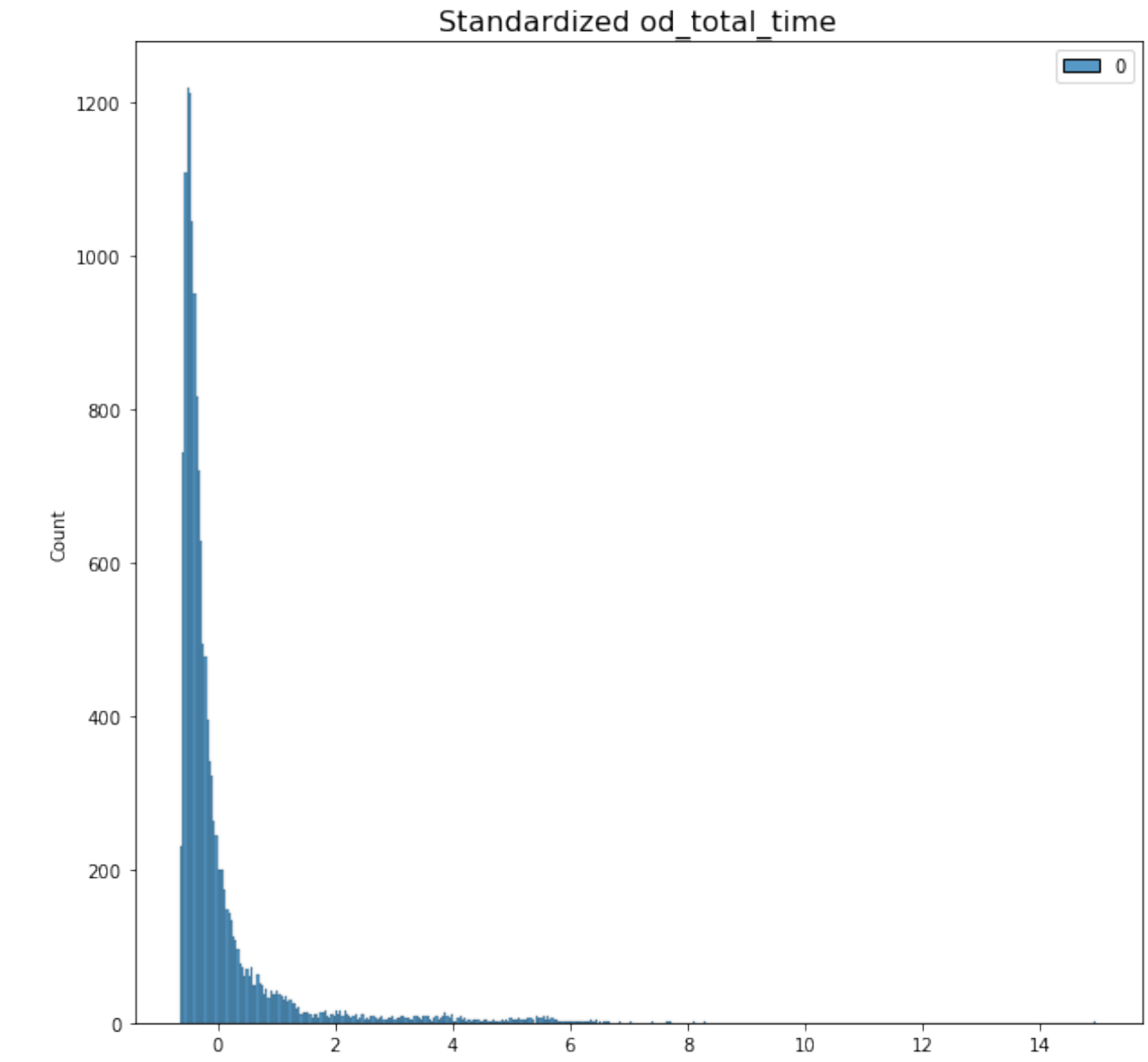

Standardized osrm_time
0
Count
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7


Standardized osrm_distance
0
Count
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7


Standardized segment_actual_time_sum
0
Count
1600
1400
1200
1000
800
600
400
200
0
0
2
4
6
8
10


Standardized segment_osrm_distance_sum
0
Count
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7


Standardized segment_osrm_time_sum
0
Count
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7



5. Hypothesis Testing Check / Visual Analysis

For checking correlation between numeric data we choose correlation Spearman correlation and calculate correlation coefficient by calculatiing rank of each data and then find the corrcoef.

- Null hypothesis is any two numeric columns are independent of each other.
- Alternate hypothesis will be columns are dependent on each other.
- The correration coefficient gives the strength of relationship.
- The range of spearman corrcoef is [-1,1].
- The more +ve the coeff is the more +ve the strength will be.

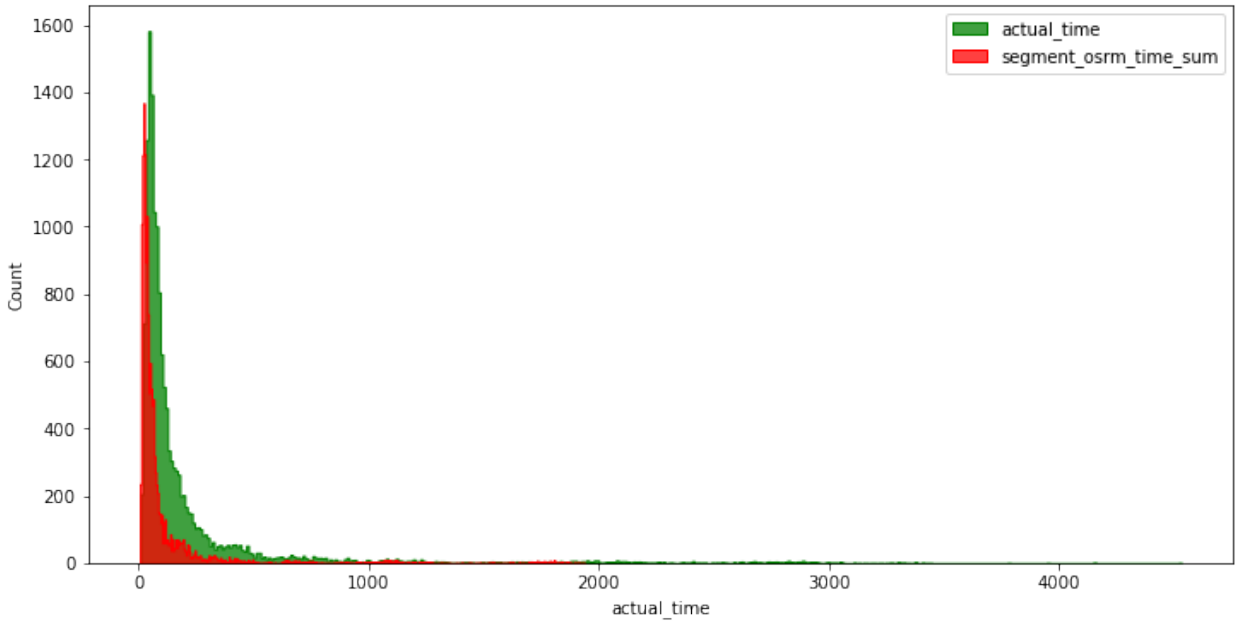
- The more -ve the coeff is, the more -ve the strenght will be.
- If the corrcoeff is 0 then there is no correlation between two numeric columns.

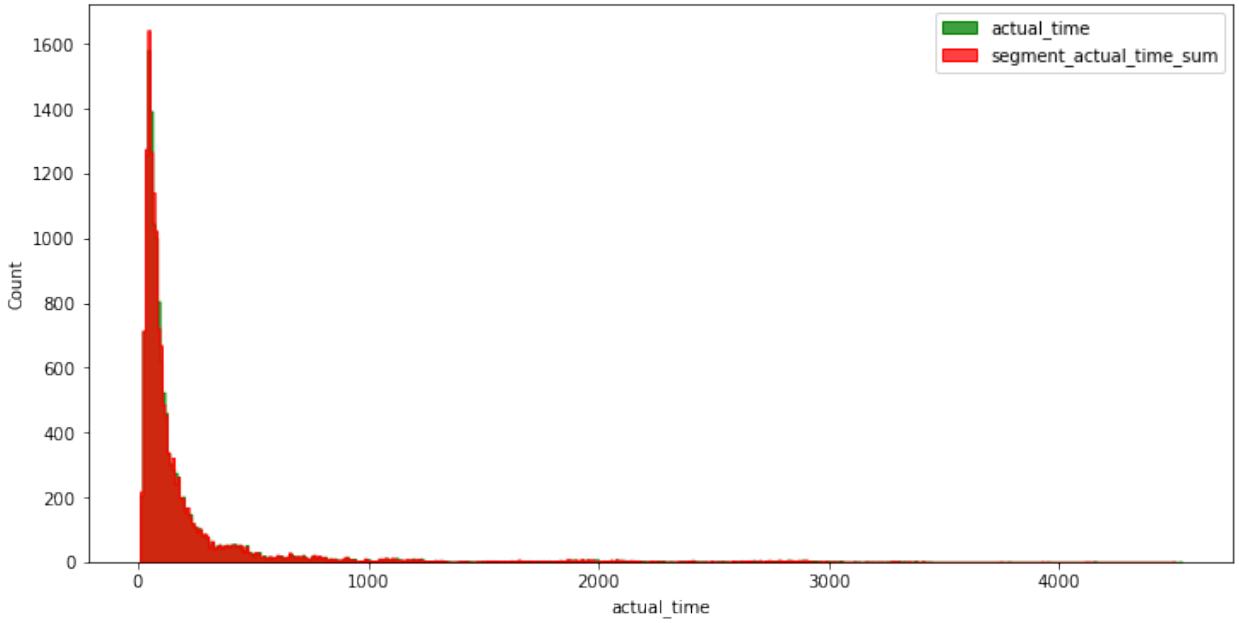
actual_time aggregated value and OSRM time aggregated value

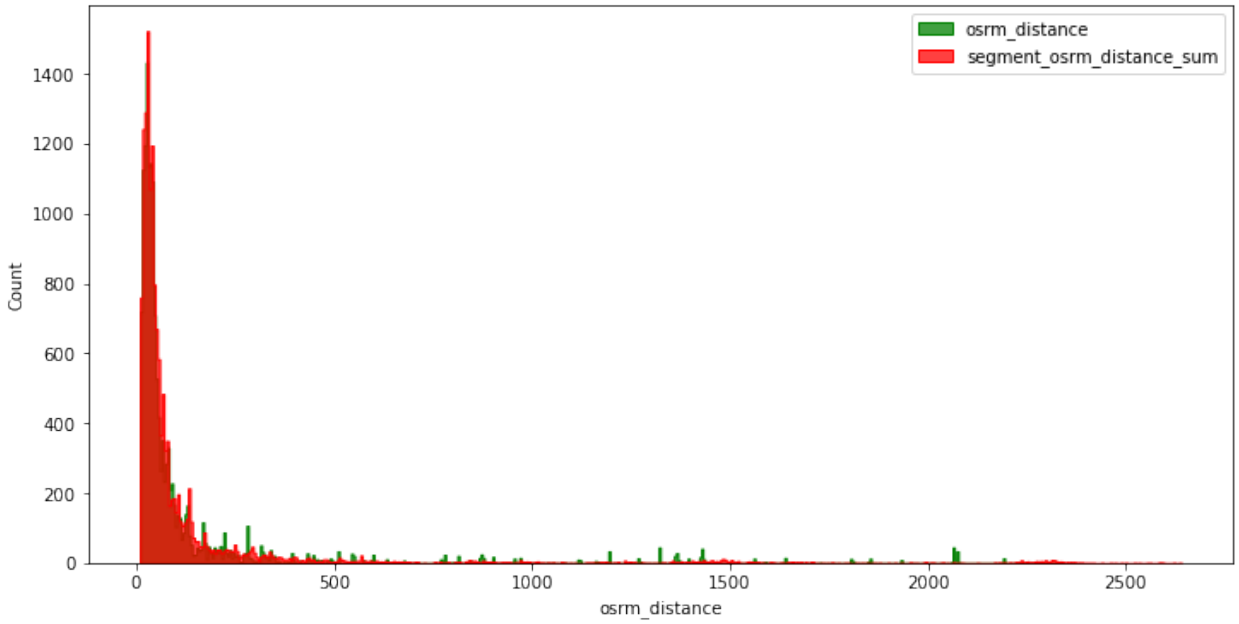
```
corr_elements = [['actual_time','segment_osrm_time_sum'],
['actual_time','segment_actual_time_sum'],
                 ['osrm_distance','segment_osrm_distance_sum'],
['osrm_time','segment_osrm_time_sum']]

for col in corr_elements:
    val = np.corrcoef(trip[col[0]].rank(), trip[col[1]].rank())[0,1]
    if val> 0 :
        print(f'There is +ve relation between {col[0]} and', col[1],"-
corrcoef: ",np.round(val,2))
    if val == 0:
        print(f'There is no relation between {col[0]} and ', col[1],"-
corrcoef: ",np.round(val,2))
    if val< 0:
        print(f'There is -ve relation between {col[0]} and ',
col[1],"- corrcoef: ",np.round(val,2))
```

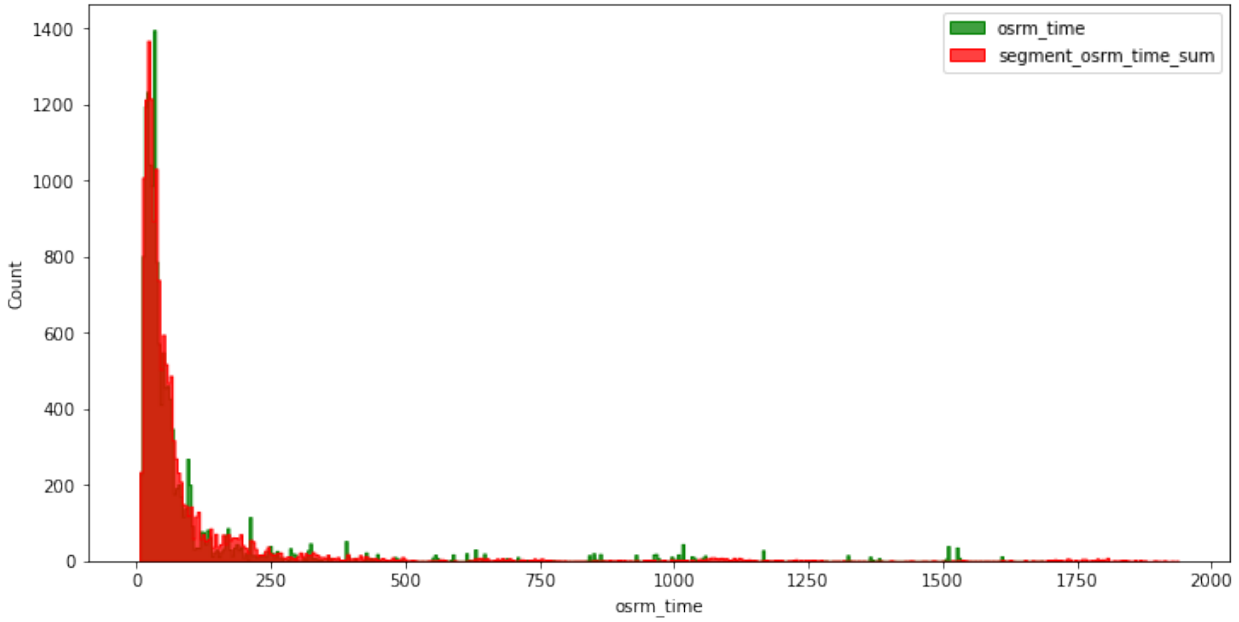
```
There is +ve relation between actual_time and segment_osrm_time_sum -
corrcoef:  0.83
There is +ve relation between actual_time and segment_actual_time_sum
- corrcoef:  1.0
There is +ve relation between osrm_distance and
segment_osrm_distance_sum - corrcoef:  0.99
There is +ve relation between osrm_time and segment_osrm_time_sum -
corrcoef:  0.98
```

```
for col in corr_elements:
    plt.figure(figsize = (12, 6))
    sns.histplot(trip[col[0]], element = 'step', color = 'green')
    sns.histplot(trip[col[1]], element = 'step', color = 'red')
    plt.legend([col[0], col[1]])
    plt.plot()
```


actual_time
segment_osrm_time_sum
Count
actual_time
0
200
400
600
800
1000
1200
1400
1600
0
1000
2000
3000
4000


actual_time
segment_actual_time_sum
Count
actual_time
0
200
400
600
800
1000
1200
1400
1600
0
1000
2000
3000
4000



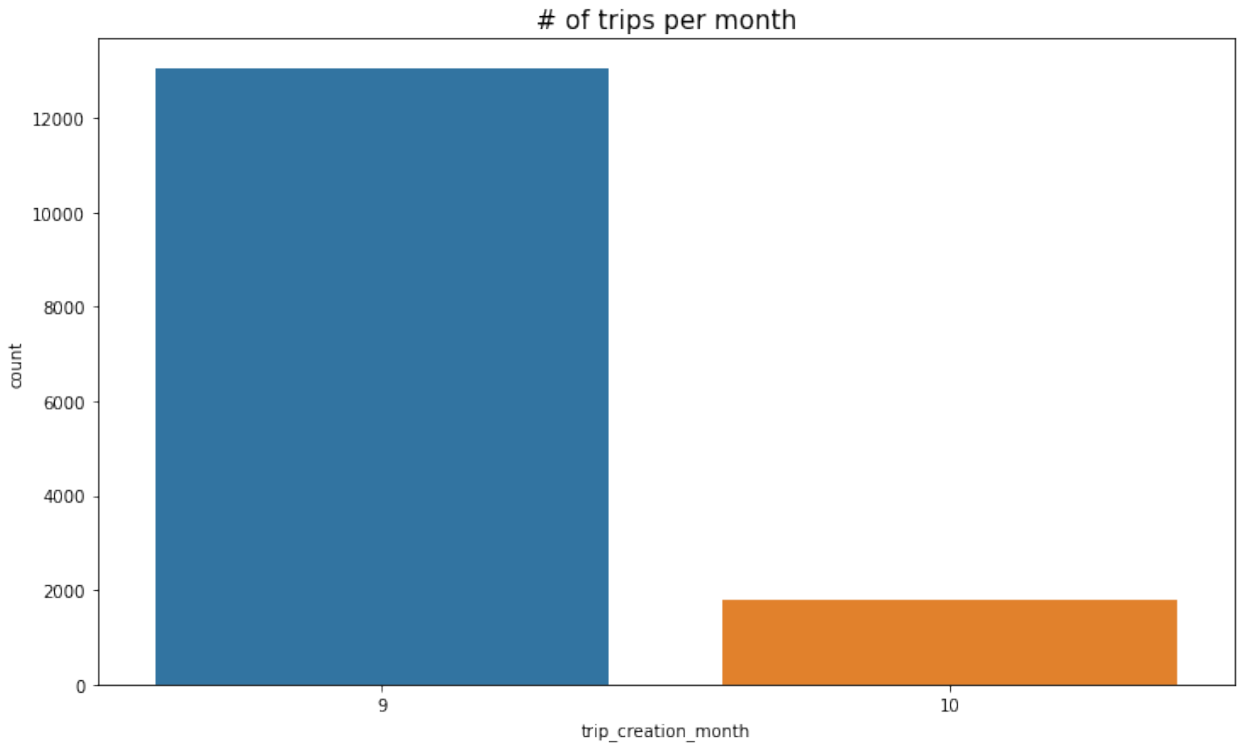



```
Insights:

    From the hypothesis testing and visual analysis we can observe
that actual_time and segment_osrm_time_sum are less correlated than
    others.
    actual_time and segment_actual_time_sum is 100% correlated.
    osrm_distance and segment_osrm_distance_sum and osrm_time and
segment_osrm_time_sum are 98% correlated.
    This indicates ORSM navigation system has some glithces in time
and distance calculation.
```

```
trip.info()
```

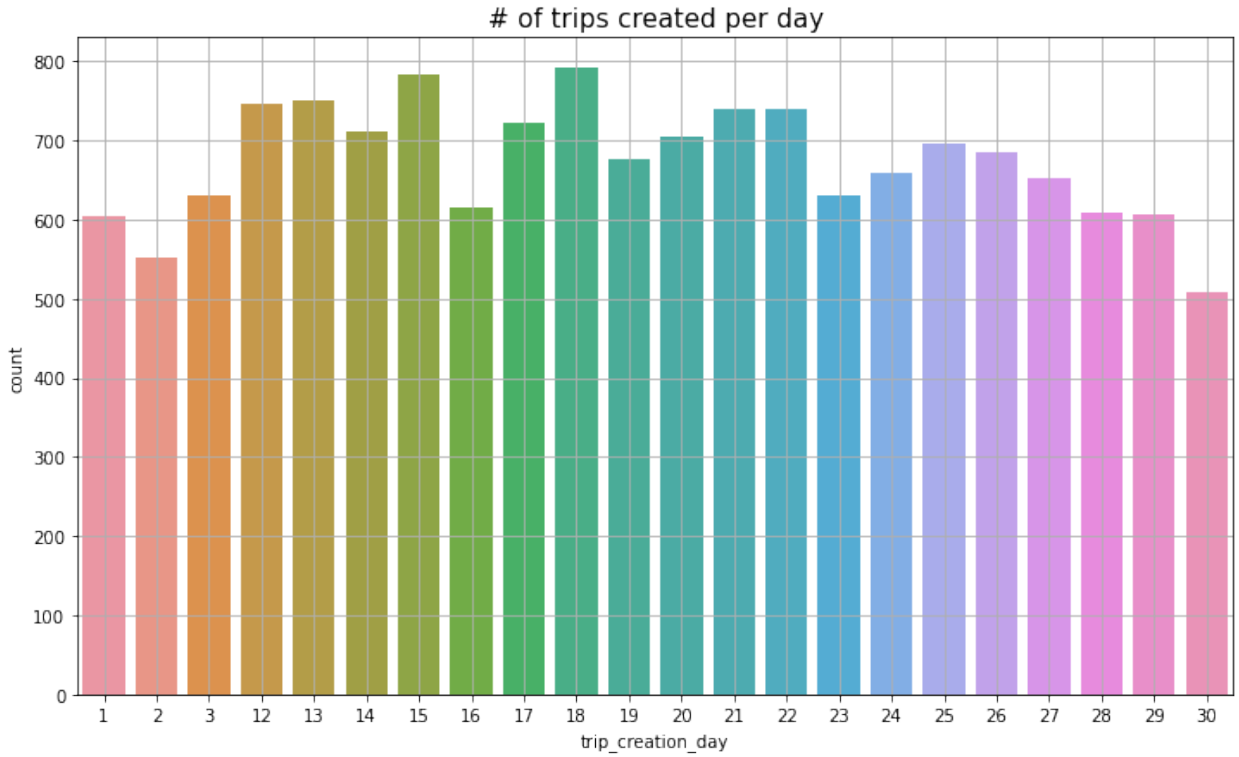
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 30 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   data                              14817 non-null  int64
 1   trip_creation_time                14817 non-null  datetime64[ns]
 2   route_schedule_uuid               14817 non-null  object
 3   route_type                        14817 non-null  int64
 4   trip_uuid                         14817 non-null  object
 5   source_center                     14817 non-null  object
 6   source_name                       14817 non-null  object
 7   destination_center                14817 non-null  object
 8   destination_name                  14817 non-null  object
 9   start_scan_to_end_scan            14817 non-null  float64
 10  actual_distance_to_destination    14817 non-null  float64
 11  actual_time                       14817 non-null  float64
 12  osrm_time                         14817 non-null  float64
 13  osrm_distance                     14817 non-null  float64
 14  segment_actual_time_sum           14817 non-null  float64
 15  segment_osrm_distance_sum         14817 non-null  float64
 16  segment_osrm_time_sum             14817 non-null  float64
 17  od_total_time                     14817 non-null  float64
 18  destination_state                 14817 non-null  object
 19  destination_city                  14817 non-null  object
 20  destination_place                 14817 non-null  object
 21  destination_code                  14817 non-null  object
 22  source_state                      14817 non-null  object
 23  source_city                       14817 non-null  object
 24  source_place                      14817 non-null  object
 25  source_code                       14817 non-null  object
 26  trip_creation_year                14817 non-null  int64
 27  trip_creation_month               14817 non-null  int64
 28  trip_creation_day                 14817 non-null  int64
 29  trip_creation_hour                14817 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(6), object(14)
memory usage: 3.4+ MB
```

```
plt.figure(figsize=(12,7))
sns.countplot(x=trip.trip_creation_month)
plt.title("# of trips per month", fontsize = 15)
plt.show()
```

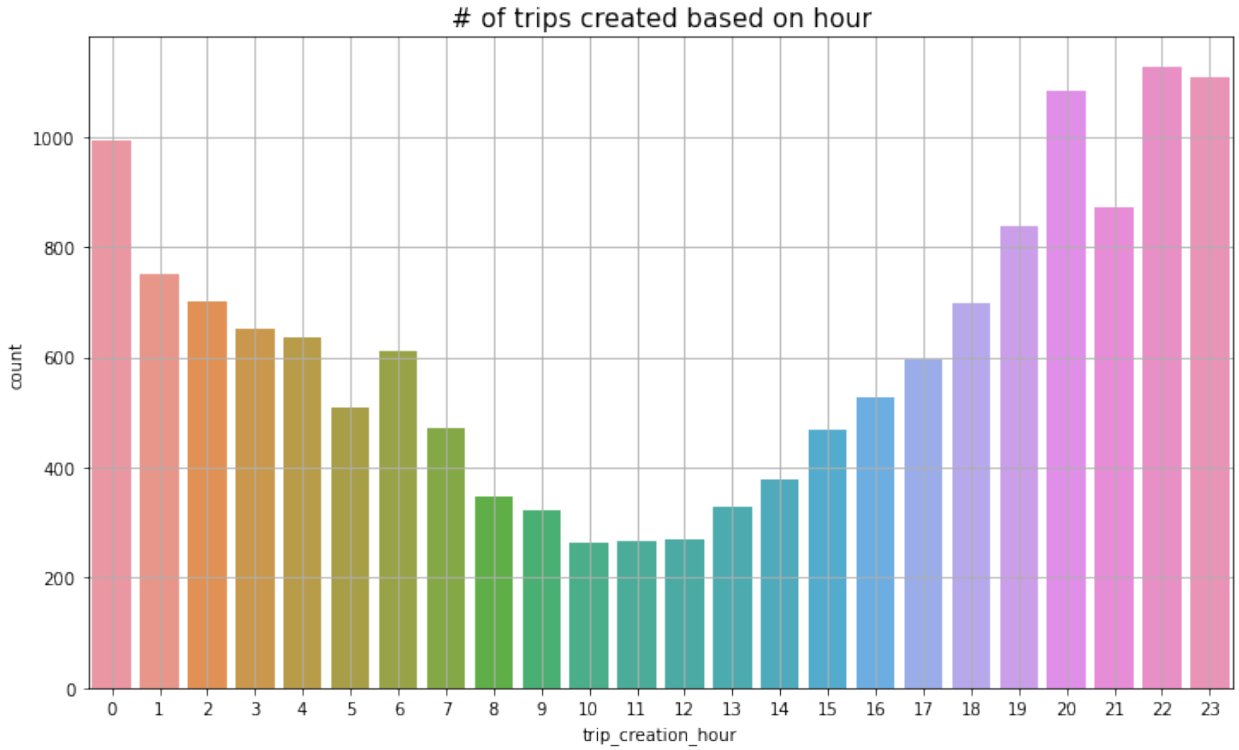


Insight: More trips are created in the month of september.

```
plt.figure(figsize=(12,7))
sns.countplot(x=trip.trip_creation_day)
plt.grid(True)
plt.title("# of trips created per day", fontsize= 15)
plt.show()
```




```
plt.figure(figsize=(12,7))
sns.countplot(x=trip.trip_creation_hour)
plt.grid(True)
plt.title("# of trips created based on hour", fontsize= 15)
plt.show()
```

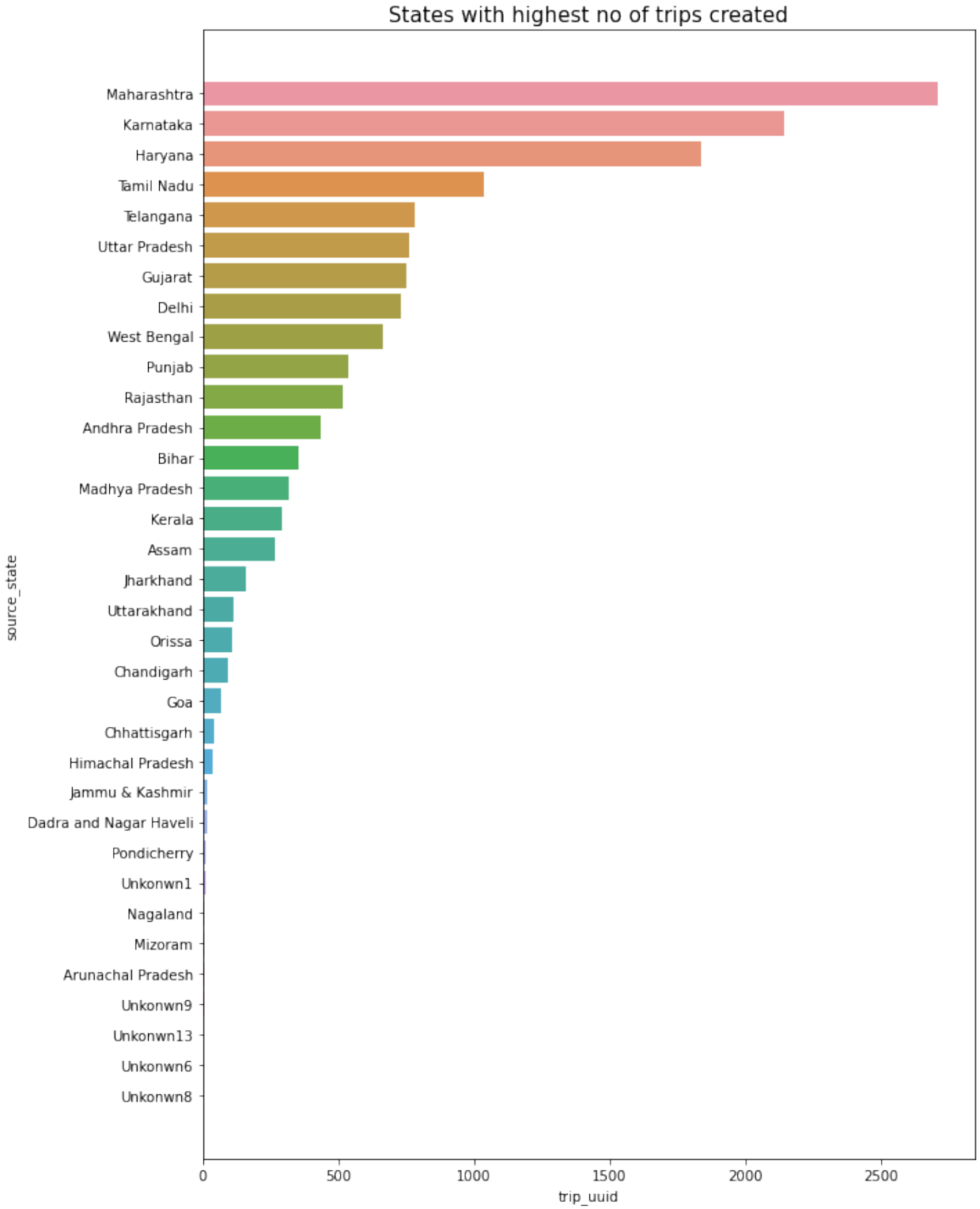


Insights: More trips are created at night hours.

```
# States with highest no of trips created
df_source_state = trip.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 = 'perc', ascending =
False)
df_source_state.head()
```

	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

```
plt.figure(figsize = (10, 15))
sns.barplot(data = df_source_state,
            x = df_source_state['trip_uuid'],
            y = df_source_state['source_state'])
plt.title("States with highest no of trips created", fontsize=15)
plt.plot()
plt.show()
```


States with highest no of trips created
Maharashtra
Karnataka
Haryana
Tamil Nadu
Telangana
Uttar Pradesh
Gujarat
Delhi
West Bengal
Punjab
Rajasthan
Andhra Pradesh
Bihar
Madhya Pradesh
Kerala
Assam
Jharkhand
Uttarakhand
Orissa
Chandigarh
Goa
Chhattisgarh
Himachal Pradesh
Jammu & Kashmir
Dadra and Nagar Haveli
Pondicherry
Unkonwn1
Nagaland
Mizoram
Arunachal Pradesh
Unkonwn9
Unkonwn13
Unkonwn6
Unkonwn8
source_state
0
500
1000
1500
2000
2500
trip_uuid

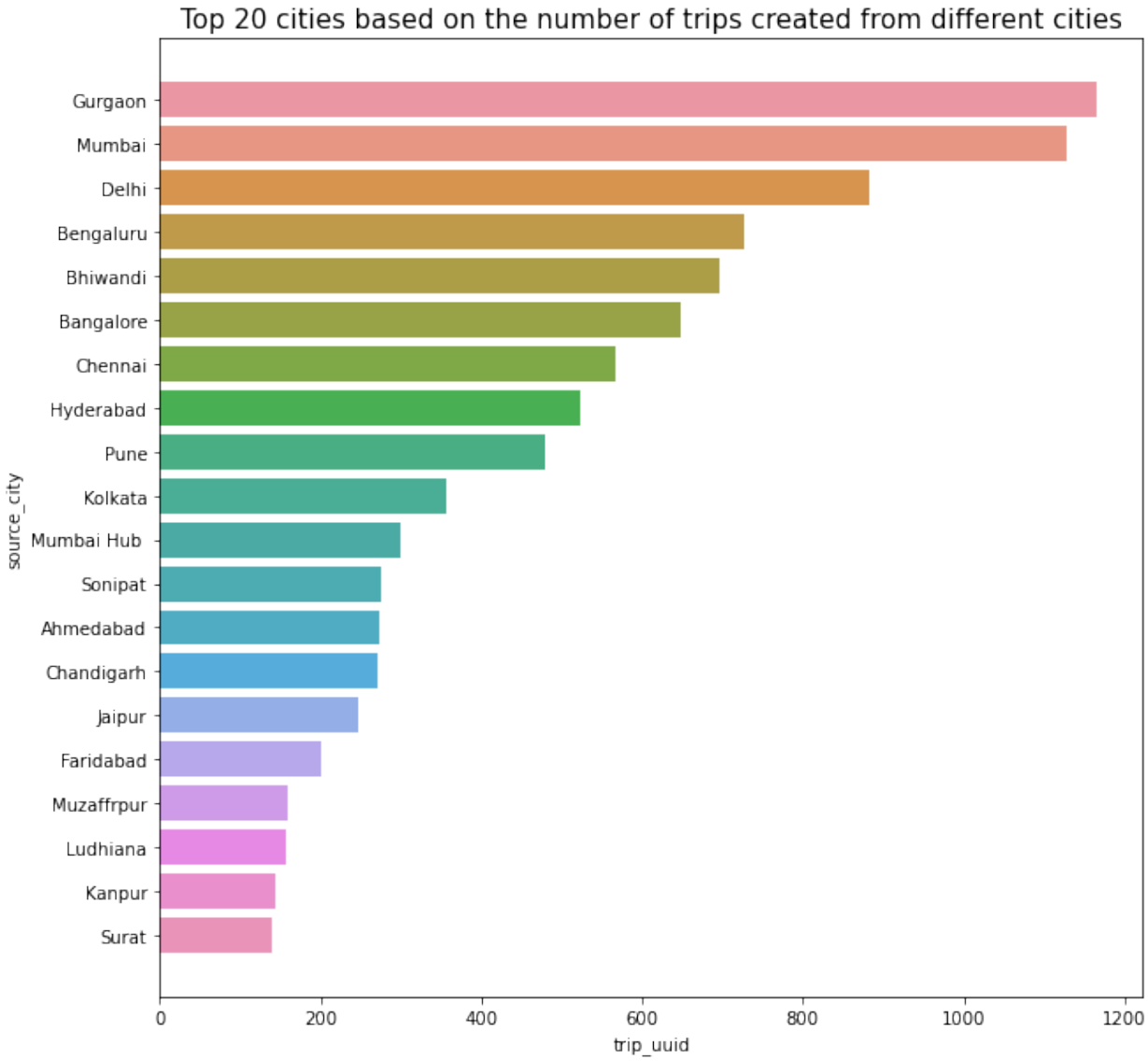
Insights:

Maharashtra is the top state with highest no of trips booed from.
secon and third would be Karnataka and Haryana.
Pondicherry, Dadra and Nagar Haveli are the least booking states.

```
# top 20 cities based on the number of trips created from different
cities
df_source_city = trip.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 = 'perc', ascending =
False)[:20]
df_source_city
```

	source_city	trip_uuid	perc
242	Gurgaon	1165	7.86
446	Mumbai	1128	7.61
173	Delhi	883	5.96
80	Bengaluru	726	4.90
101	Bhiwandi	697	4.70
58	Bangalore	648	4.37
139	Chennai	568	3.83
269	Hyderabad	524	3.54
530	Pune	480	3.24
364	Kolkata	356	2.40
448	Mumbai Hub	300	2.02
625	Sonipat	276	1.86
2	Ahmedabad	274	1.85
135	Chandigarh	272	1.84
276	Jaipur	246	1.66
205	Faridabad	200	1.35
457	Muzaffrpur	159	1.07
389	Ludhiana	158	1.07
327	Kanpur	145	0.98
636	Surat	140	0.94

```
plt.figure(figsize = (10, 10))
sns.barplot(data = df_source_city,
            x = df_source_city['trip_uuid'],
            y = df_source_city['source_city'])
plt.title("Top 20 cities based on the number of trips created from
different cities", fontsize=15)
plt.plot()
plt.show()
```

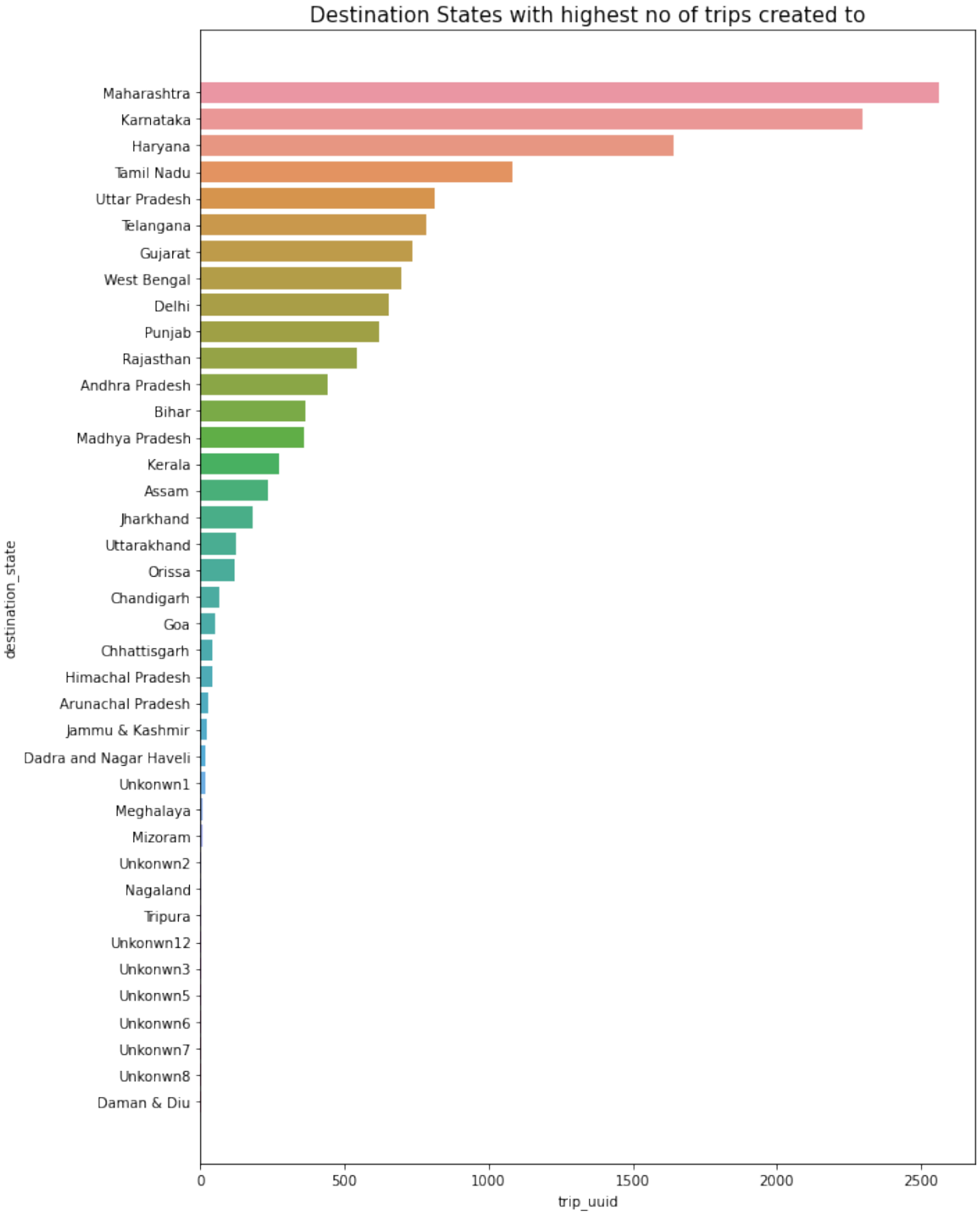


Insights:

```
Gurgon is the top city with highest nuber of trips created followed by
Mumbai and Delhi.

# Destinatoin States with highest no of trips created to
df_destn_state = trip.groupby(by = 'destination_state')
['trip_uuid'].count().to_frame().reset_index()
df_destn_state['perc'] = np.round(df_destn_state['trip_uuid'] * 100/
df_destn_state['trip_uuid'].sum(), 2)
df_destn_state = df_destn_state.sort_values(by = 'perc', ascending =
False)
df_source_state.head()

   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
```

```
plt.figure(figsize = (10, 15))
sns.barplot(data = df_destn_state,
            x = df_destn_state['trip_uuid'],
            y = df_destn_state['destination_state'])
plt.title("Destination States with highest no of trips created to",
fontsize = 15)
plt.plot()
plt.show()
```



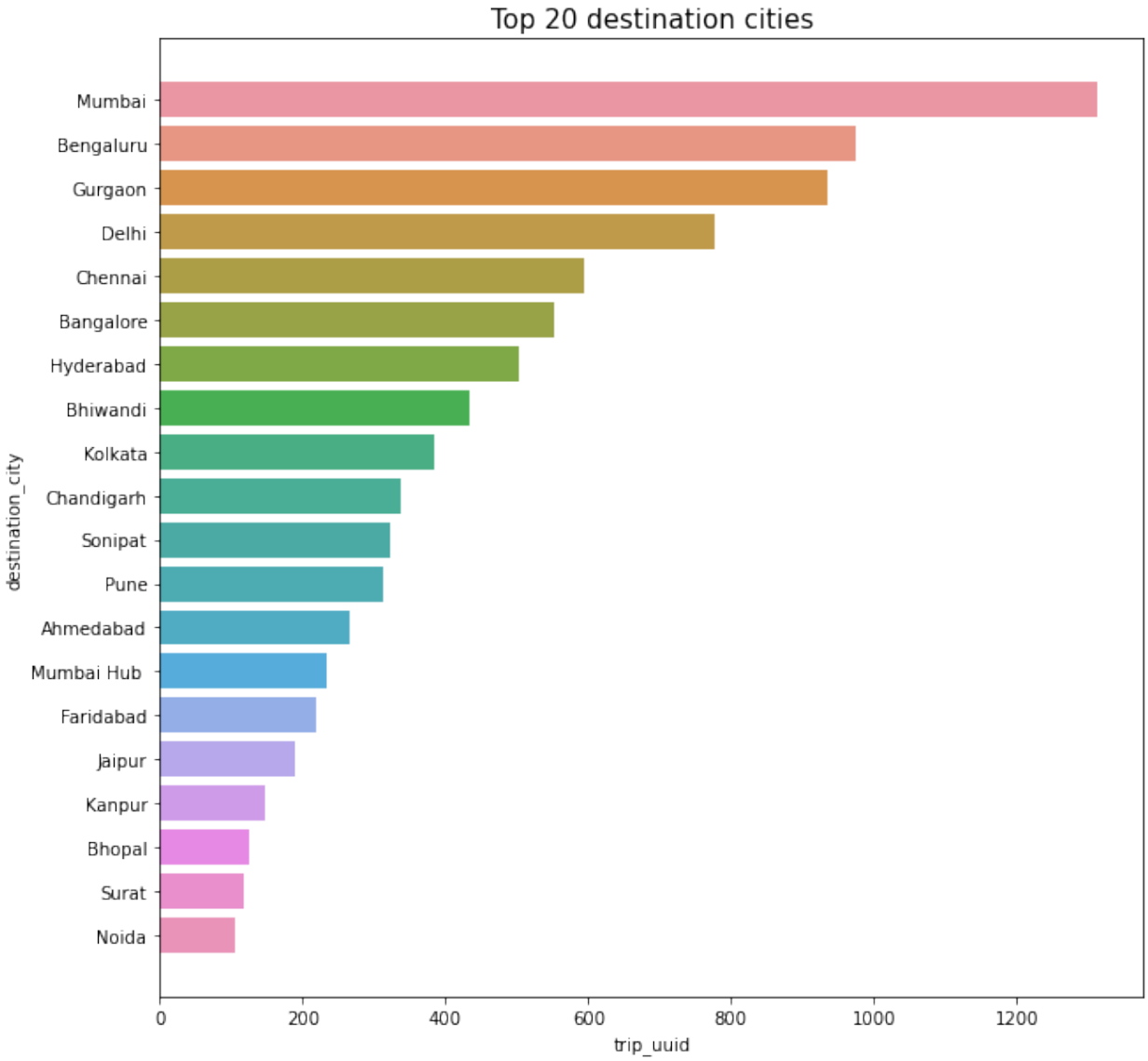
Insights:

Top Destination states are Maharashtra, Karnataka and Haryana.

```
# top 20 cities based on the number of trips created from different
cities
df_destn_city = trip.groupby(by = 'destination_city')
['trip_uuid'].count().to_frame().reset_index()
df_destn_city['perc'] = np.round(df_destn_city['trip_uuid'] * 100/
df_destn_city['trip_uuid'].sum(), 2)
df_destn_city = df_destn_city.sort_values(by = 'perc', ascending =
False)[:20]
df_destn_city
```

	destination_city	trip_uuid	perc
521	Mumbai	1312	8.85
97	Bengaluru	975	6.58
286	Gurgaon	936	6.32
203	Delhi	778	5.25
166	Chennai	595	4.02
73	Bangalore	551	3.72
312	Hyderabad	503	3.39
116	Bhiwandi	434	2.93
424	Kolkata	384	2.59
160	Chandigarh	338	2.28
737	Sonipat	322	2.17
623	Pune	313	2.11
4	Ahmedabad	265	1.79
523	Mumbai Hub	234	1.58
245	Faridabad	220	1.48
323	Jaipur	189	1.28
377	Kanpur	148	1.00
118	Bhopal	124	0.84
752	Surat	117	0.79
560	Noida	106	0.72

```
plt.figure(figsize = (10, 10))
sns.barplot(data = df_destn_city,
            x = df_destn_city['trip_uuid'],
            y = df_destn_city['destination_city'])
plt.title("Top 20 destination cities",fontsize=15 )
plt.plot()
plt.show()
```




```
Insights:

    Mumbai, Gurgon and Delhi are top 3 destination cities.
```