Business Case - Delhivery: Feature Engineering

1. Problem Statement

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.

Objective of the Data Analysis?

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

Importing Required Libraries

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

Loading the given data-set

In [3]:
```
df = pd.read_csv('D:\\Scaler\\Scaler\\Hypothesis Testing\\Business Case\\delhivery_data.csv')
```

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

Out[4]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	s
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_V
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_V
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_V
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_V
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_V

5 rows × 24 columns

2. Data Exploration

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

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

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

Out[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')
```

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

Out[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
```

In [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

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

In [10]:
```
df.info()
```

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

Identifying Category Columns

In order to find the category columns, we will consider columns with 2 unique values

In [11]:
```
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
```

In [12]:
```
cat_cols = ['data', 'route_type']
for col in cat_cols:
    df[col] =  df[col].astype('object')
```

In [13]:
```
floating_columns = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_dista
                    'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance', 'sta
for col in floating_columns:
    print(f"{col:<30} = {df[col].max()}")
```

```
actual_distance_to_destination = 1927.4477046975032
actual_time                    = 4532.0
osrm_time                      = 1686.0
osrm_distance                  = 2326.1991000000003
segment_actual_time            = 3051.0
segment_osrm_time              = 1611.0
segment_osrm_distance          = 2191.4037000000003
start_scan_to_end_scan         = 7898.0
```

Updating the datatype of the datetime columns

In [16]:
```
datetime_columns = ['trip_creation_time', 'od_start_time', 'od_end_time']
for col in datetime_columns:
    df[col] = pd.to_datetime(df[col], unit='ns')
```

In [17]:
```
df['trip_creation_time'].min(), df['od_end_time'].max()
```

Out[17]:
```
(Timestamp('2018-09-12 00:00:16.535741'),
 Timestamp('2018-10-08 03:00:24.353479'))
```

3. Data Cleaning

Handling Missing Values in the data-set

Checking for null values

In [18]:
```
np.any(df.isnull())
```

Out[18]:
```
True
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
data                            0
trip_creation_time              0
route_schedule_uuid             0
route_type                      0
trip_uuid                       0
source_center                   0
source_name                   293
destination_center              0
destination_name              261
od_start_time                   0
od_end_time                     0
start_scan_to_end_scan          0
actual_distance_to_destination  0
actual_time                     0
osrm_time                       0
osrm_distance                   0
segment_actual_time             0
segment_osrm_time               0
segment_osrm_distance           0
dtype: int64
```

In [20]:
```
missing_source_name = df.loc[df['source_name'].isnull(), 'source_center'].unique()
missing_source_name
```

Out[20]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [21]:
```
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
```

In [22]:
```
missing_destination_name = df.loc[df['destination_name'].isnull(), 'destination_center'].uniq
missing_destination_name
```

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

In [23]:
```
for i in missing_destination_name:
    unique_destination_name = df.loc[df['destination_center'] == i, 'destination_name'].uniqu
    if pd.isna(unique_source_name):
        print("destination_center :", i, "-" * 10, "destination_name :", 'Not Found')
    else :
        print("destination_center :", i, "-" * 10, "destination_name :", unique_destination_n
```

```
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 : IND505326AAB ---------- destination_name : Not Found
destination_center : IND852118A1B ---------- destination_name : Not Found
destination_center : IND126116AAA ---------- destination_name : Not Found
destination_center : IND509103AAC ---------- destination_name : Not Found
destination_center : IND221005A1A ---------- destination_name : Not Found
destination_center : IND250002AAC ---------- destination_name : Not Found
destination_center : IND331001A1C ---------- destination_name : Not Found
destination_center : IND122015AAC ---------- destination_name : Not Found
```

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

In [24]:
```
np.all(df.loc[df['source_name'].isnull(), 'source_center'].isin(missing_destination_name))
```

Out[24]:
```
False
```

Treating missing destination names and source names

In [25]:
```
count = 1
for i in missing_destination_name:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.loc[df['destination_center
   'destination_name'].replace(np.nan, f'location_{count}')
    count += 1
```

In [26]:
```
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
```

In [27]:
```
for i in missing_source_name:
    df.loc[df['source_center'] == i, 'source_name'] = df.loc[df['source_center'] == i, 'sourc
```

In [28]:
```
df.isna().sum()
```

Out[28]:
```
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
```

Statistical Summary

In [29]:
```
df.describe()
```

Out[29]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segr
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	
mean	961.262939	234.073380	416.927521	213.868286	284.771301	
std	1036.997803	344.979126	598.096069	308.004333	421.117462	
min	20.000000	9.000046	9.000000	6.000000	9.008200	
25%	161.000000	23.355875	51.000000	27.000000	29.914701	
50%	449.000000	66.126572	132.000000	64.000000	78.525803	
75%	1634.000000	286.708878	513.000000	257.000000	343.193253	
max	7898.000000	1927.447754	4532.000000	1686.000000	2326.199219	

In [30]:
```
df.describe(include = 'object')
```

Out[30]:

	data	route_schedule_uuid	route_type	trip_uuid	source_center	source_name
count	144867	144867	144867	144867	144867	144867
unique	2	1504	2	14817	1508	1508
top	training	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)
freq	104858	1812	99660	101	23347	23347

4. Building features to prepare the data for actual analysis

1. grouping Trip_uuid,'source_center', and'destination_center'

In [31]:
```
combined_package_details = ['trip_uuid', 'source_center', 'destination_center']
#Creating another dataframe to merge the above-mentioned features
df1 = df.groupby(by = combined_package_details, 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' : 'la
                                                                       'actual_time' : 'last',
                                                                       'osrm_time' : 'last',
                                                                       'osrm_distance' : 'last',
                                                                       'segment_actual_time' : 'sum',
                                                                       'segment_osrm_time' : 'sum',
                                                                       'segment_osrm_distance' : 'sum'})
```

In [32]:
```
df1.head()
```

Out[32]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	sc
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	FTL	2018-09-12 00:00:16.535741	Kanpur_ (Ut
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741	Bhopa (Madh
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_
3	trip-153671042288605164	IND572101AAA	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430	Tumku
4	trip-153671043369099517	IND000000ACB	IND160002AAC	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_

Calculating the time taken between od_start_time and od_end_time and creating it as a feature. Also, dropping the original columns

In [33]:
```
#New Feature name is: od_total_time
df1['od_total_time'] = df1['od_end_time'] - df1['od_start_time']
df1.drop(columns = ['od_end_time', 'od_start_time'], inplace = True)
```

```
df1['od_total_time'] = df1['od_total_time'].apply(lambda x: round(x.total_seconds()/60.0,2))
df1['od_total_time'].head()
```

Out[33]:

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

Further aggregate on the basis of just Trip_uuid

In [34]:

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

Out[34]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741	K
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430	Dodda
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018-09-12 00:00:33.691250	Gu
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018-09-12 00:02:09.740725	Bell
...	...	...	...	...	...	...	
14812	trip-153861095625827784	IND160002AAC	IND160002AAC	test	Carting	2018-10-03 23:55:56.258533	Chandig
14813	trip-153861104386292051	IND121004AAB	IND121004AAA	test	Carting	2018-10-03 23:57:23.863155	FBI
14814	trip-153861106442901555	IND208006AAA	IND208006AAA	test	Carting	2018-10-03 23:57:44.429324	Kan
14815	trip-153861115439069069	IND627005AAA	IND628204AAA	test	Carting	2018-10-03 23:59:14.390954	Tiru
14816	trip-153861118270144424	IND583119AAA	IND583119AAA	test	FTL	2018-10-03 23:59:42.701692	San

14817 rows × 17 columns

5. Extracting Features

1. Source Name: Split and extract features out of 'source_name' - State | City | Place | Code

Extracting State

In [35]:
```
def location_name_to_state(x):
    l = x.split('(')
    if len(l) == 1:
        return l[0]
    else:
        return l[1].replace(')', "")
```

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

Out[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)
```

Extracting City

In [38]:
```
def location_name_to_city(x):
    if 'location' in x:
        return 'unknown_city'
    else:
        l = x.split()[0].split('_')
        if 'CCU' in x:
            return 'Kolkata'
        elif 'MAA' in x.upper():
            return 'Chennai'
        elif ('HBR' in x.upper()) or ('BLR' in x.upper()):
            return 'Bengaluru'
        elif 'FBD' in x.upper():
            return 'Faridabad'
        elif 'BOM' in x.upper():
            return 'Mumbai'
        elif 'DEL' in x.upper():
            return 'Delhi'
        elif 'OK' in x.upper():
            return 'Delhi'
        elif 'GZB' in x.upper():
            return 'Ghaziabad'
        elif 'GGN' in x.upper():
            return 'Gurgaon'
        elif 'AMD' in x.upper():
            return 'Ahmedabad'
        elif 'CJB' in x.upper():
            return 'Coimbatore'
        elif 'HYD' in x.upper():
            return 'Hyderabad'
        return l[0]
```

In [39]:
```
df2['source_city'] = df2['source_name'].apply(location_name_to_city)
print('No of source cities :', df2['source_city'].nunique())
df2['source_city'].unique()[:100]
```

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

Out[39]:
```
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)
```

Extracting Place

In [40]:
```python
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]
     '''
    # we will remove state
    x = x.split('(')[0]

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

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

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

    # now we need to deal with edge cases or imporper name convention

    # if len(x.split('_')) == 2:

    return x.split(' ')[0]
```

In [41]:
```python
df2['source_place'] = df2['source_name'].apply(location_name_to_place)
df2['source_place'].unique()[:100]
```

Out[41]:
```
array(['Central', 'ChikaDPP', 'Bilaspur', 'Mumbai', 'Bellary', 'Chennai',
       'Chrompet', 'HBR', 'Lajpat', 'North', 'Balabhgarh', 'Shamshbd',
       'Xroad', 'Nehrugnj', 'Nangli', 'Guwahati', 'KndliDPP', 'DavkharRd',
       'Bandel', 'RTCStand', 'KGAirprt', 'Jalandhar', 'Mthurard',
       'Mullanpr', 'RajCmplx', 'Beliaghata', 'RjnaiDPP', 'AbbasNgr',
       'Mankoli', 'Bhatinda', 'Airport', 'Jaipur', 'Gateway', 'Tathawde',
       'ChotiHvl', 'Trmltmpl', 'OnkarDPP', 'Mehmdpur', 'KaranNGR',
       'Sohagpur', 'Busstand', 'IndEstat', 'Court', 'Jetpur', 'Panchot',
       'Adhartal', 'DumDum', 'Bomsndra', 'Junagadh', 'Swamylyt',
       'Yadvgiri', 'Goa', 'Bhopal', 'Kundli', 'Himmatnagar', 'Vasanthm',
       'Poonamallee', 'VUNagar', 'NlgaonRd', 'Nadiad', 'Bnnrghta',
       'Thirumtr', 'GariDPP', 'Bhubaneshwar', 'Jogshwri', 'KoilStrt',
       'CotnGren', 'Nzbadrd', 'Dwaraka', 'Nelmngla', 'NvygRDPP', 'Hospet',
       'Gndhichk', 'Chowk', 'CharRsta', 'Bilimora', 'Kollgpra', 'Lucknow',
       'Peenya', 'GndhiNgr', 'Sanpada', 'Bhuj', 'WrdN4DPP', 'Sakinaka',
       'CivilHPL', 'OstwlEmp', 'Vizag', 'Mhbhirab', 'MGRoad', 'Balajicly',
       'BljiMrkt', 'Dankuni', 'Trnsport', 'AMD', 'East', 'Mithakal',
       'TrnspNgr', 'Gandhinagar', 'KamaStrt', 'PatelWrd'], dtype=object)
```

Extracting Code

In [42]:
```
def location_name_to_code(x):
    # we will remove state
    x = x.split('(')[0]

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

    return 'none'
```

In [43]:
```
df2['source_code'] = df2['source_name'].apply(location_name_to_code)
df2['source_code'].unique()[:100]
```

Out[43]:
```
array(['6 ', 'D ', 'HB ', 'none', 'DPC ', '12 ', 'IP ', '3 ', 'H ', 'I ',
       '7 ', '1 ', '9 ', '2 ', 'L ', 'DC ', 'M ', 'RP ', '21 ', '4 ',
       'Pc ', 'PC ', 'C ', 'V ', 'CP ', '8 ', 'Dc ', 'P ', '11 ', '5 ',
       '20 ', '15 ', '10 '], dtype=object)
```

Source: State - City - Place - Code Table

In [44]:
```
df2[['source_state','source_city','source_place','source_code']].head(10)
```

Out[44]:

	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	Mumbai	none
4	Karnataka	Bellary	Bellary	none
5	Tamil Nadu	Chennai	Chennai	none
6	Tamil Nadu	Chennai	Chrompet	DPC
7	Karnataka	Bengaluru	HBR	none
8	Gujarat	Surat	Central	12
9	Delhi	Delhi	Lajpat	IP

2. Destination Name: Split and extract features out of 'destination_name' - State | City | Place | Code

In [45]:
```
df2['destination_state'] = df2['destination_name'].apply(location_name_to_state)
df2['destination_state'].head(10)
```

Out[45]:
```
0    Uttar Pradesh
1        Karnataka
2          Haryana
3      Maharashtra
4        Karnataka
5       Tamil Nadu
6       Tamil Nadu
7        Karnataka
8          Gujarat
9            Delhi
Name: destination_state, dtype: object
```

In [46]:
```
df2['destination_city'] = df2['destination_name'].apply(location_name_to_city)
df2['destination_city'].head()
```

Out[46]:
```
0        Kanpur
1    Doddablpur
2       Gurgaon
3        Mumbai
4        Sandur
Name: destination_city, dtype: object
```

In [47]:
```
df2['destination_place'] = df2['destination_name'].apply(location_name_to_place)
df2['destination_place'].head()
```

Out[47]:
```
0     Central
1    ChikaDPP
2    Bilaspur
3      MiraRd
4    WrdN1DPP
Name: destination_place, dtype: object
```

In [48]:
```
df2['destination_code'] = df2['destination_name'].apply(location_name_to_code)
df2['destination_code'].head()
```

Out[48]:
```
0     6
1     D
2    HB
3    IP
4     D
Name: destination_code, dtype: object
```

Destination: State - City - Place - Code Table

In [49]:
```
df2[['destination_state','destination_city','destination_place','destination_code']].head(10)
```

Out[49]:

	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
5	Tamil Nadu	Chennai	Chennai	none
6	Tamil Nadu	Chennai	Vandalur	Dc
7	Karnataka	Bengaluru	HBR	none
8	Gujarat	Surat	Central	3
9	Delhi	Delhi	Delhi	none

3. Extracting features Hour | date | day | week | month | year from 'Trip_creation_time'

In [50]:
```
df2['trip_creation_hour'] = df2['trip_creation_time'].dt.hour
df2['trip_creation_hour'] = df2['trip_creation_hour'].astype('int8')
df2['trip_creation_hour'].head()
```

Out[50]:
```
0    0
1    0
2    0
3    0
4    0
Name: trip_creation_hour, dtype: int8
```

In [51]:
```
df2['trip_creation_date'] = pd.to_datetime(df2['trip_creation_time'].dt.date)
df2['trip_creation_date'].head()
```

Out[51]:
```
0   2018-09-12
1   2018-09-12
2   2018-09-12
3   2018-09-12
4   2018-09-12
Name: trip_creation_date, dtype: datetime64[ns]
```

In [52]:
```
df2['trip_creation_day'] = df2['trip_creation_time'].dt.day
df2['trip_creation_day'] = df2['trip_creation_day'].astype('int8')
df2['trip_creation_day'].head()
```

Out[52]:
```
0    12
1    12
2    12
3    12
4    12
Name: trip_creation_day, dtype: int8
```

In [53]:
```
df2['trip_creation_week'] = df2['trip_creation_time'].dt.isocalendar().week
df2['trip_creation_week'] = df2['trip_creation_week'].astype('int8')
df2['trip_creation_week'].head()
```

Out[53]:
```
0    37
1    37
2    37
3    37
4    37
Name: trip_creation_week, dtype: int8
```

In [54]:
```
df2['trip_creation_month'] = df2['trip_creation_time'].dt.month
df2['trip_creation_month'] = df2['trip_creation_month'].astype('int8')
```

```
df2['trip_creation_month'].head()
```

Out[54]:
```
0    9
1    9
2    9
3    9
4    9
Name: trip_creation_month, dtype: int8
```

In [55]:
```
df2['trip_creation_year'] = df2['trip_creation_time'].dt.year
df2['trip_creation_year'] = df2['trip_creation_year'].astype('int16')
df2['trip_creation_year'].head()
```

Out[55]:
```
0    2018
1    2018
2    2018
3    2018
4    2018
Name: trip_creation_year, dtype: int16
```

Finding the structure of data after data cleaning

In [56]:
```
df2.shape
```

Out[56]:
```
(14817, 31)
```

In [57]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 31 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  object
 4   route_type                      14817 non-null  object
 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  float32
 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  source_code                     14817 non-null  object
 21  destination_state               14817 non-null  object
 22  destination_city                14817 non-null  object
 23  destination_place               14817 non-null  object
 24  destination_code                14817 non-null  object
 25  trip_creation_hour              14817 non-null  int8
 26  trip_creation_date              14817 non-null  datetime64[ns]
 27  trip_creation_day               14817 non-null  int8
 28  trip_creation_week              14817 non-null  int8
 29  trip_creation_month             14817 non-null  int8
 30  trip_creation_year              14817 non-null  int16
dtypes: datetime64[ns](2), float32(8), float64(1), int16(1), int8(4), object(15)
memory usage: 2.6+ MB
```

In [58]:
```
df2.describe().T
```

Out[58]:

	count	mean	std	min	25%	50%	
od_total_time	14817.0	531.697630	658.868223	23.460000	149.930000	280.770000	638.20
start_scan_to_end_scan	14817.0	530.809998	658.707031	23.000000	149.000000	280.000000	637.00
actual_distance_to_destination	14817.0	164.477829	305.388123	9.002461	22.837238	48.474072	164.58
actual_time	14817.0	357.143768	561.395020	9.000000	67.000000	149.000000	370.00
osrm_time	14817.0	161.384018	271.362549	6.000000	29.000000	60.000000	168.00
osrm_distance	14817.0	204.344711	370.395508	9.072900	30.819201	65.618805	208.47
segment_actual_time	14817.0	353.892273	556.246826	9.000000	66.000000	147.000000	367.00
segment_osrm_time	14817.0	180.949783	314.541412	6.000000	31.000000	65.000000	185.00
segment_osrm_distance	14817.0	223.201157	416.628326	9.072900	32.654499	70.154404	218.80
trip_creation_hour	14817.0	12.449821	7.986553	0.000000	4.000000	14.000000	20.00
trip_creation_day	14817.0	18.370790	7.893275	1.000000	14.000000	19.000000	25.00
trip_creation_week	14817.0	38.295944	0.967872	37.000000	38.000000	38.000000	39.00
trip_creation_month	14817.0	9.120672	0.325757	9.000000	9.000000	9.000000	9.00
trip_creation_year	14817.0	2018.000000	0.000000	2018.000000	2018.000000	2018.000000	2018.00

In [59]:
```
df2.describe(include = object).T
```

Out[59]:

	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
source_center	14817	938	IND000000ACB	1063
destination_center	14817	1042	IND000000ACB	821
data	14817	2	training	10654
route_type	14817	2	Carting	8908
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	774	Bilaspur	1085
source_code	14817	33	HB	3222
destination_state	14817	39	Maharashtra	2561
destination_city	14817	806	Mumbai	1548
destination_place	14817	876	Bilaspur	864
destination_code	14817	34	D	2868

6. Data Visualization | Data Analysis

Q1. How many trips are created on the hourly basis?

In [60]:
```
df2['trip_creation_hour'].unique()
```

Out[60]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23], dtype=int8)
```

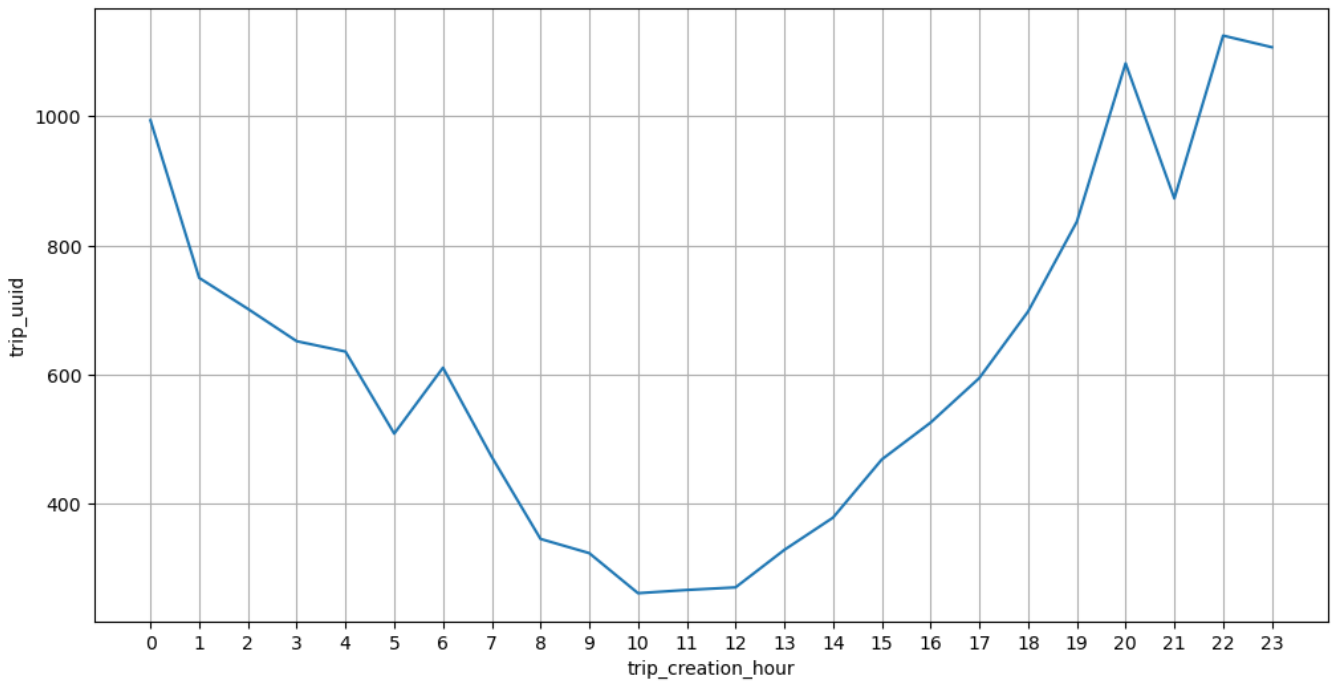
In [61]:
```
df_hour = df2.groupby(by = 'trip_creation_hour')['trip_uuid'].count().to_frame().reset_index(
df_hour.head()
```

Out[61]:

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

In [62]:
```
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()
```

Out[62]:
```
[]
```



Insights:

1. It can be inferred from the above plot that the number of trips start increasing post 12 pm.

2. Number of trips observed to be maximum at 10 P.M and then start decreasing

Q2.How many trips are created for different days of the month?

In [63]:
```
df2['trip_creation_day'].unique()
```

Out[63]:
```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
       29, 30,  1,  2,  3], dtype=int8)
```

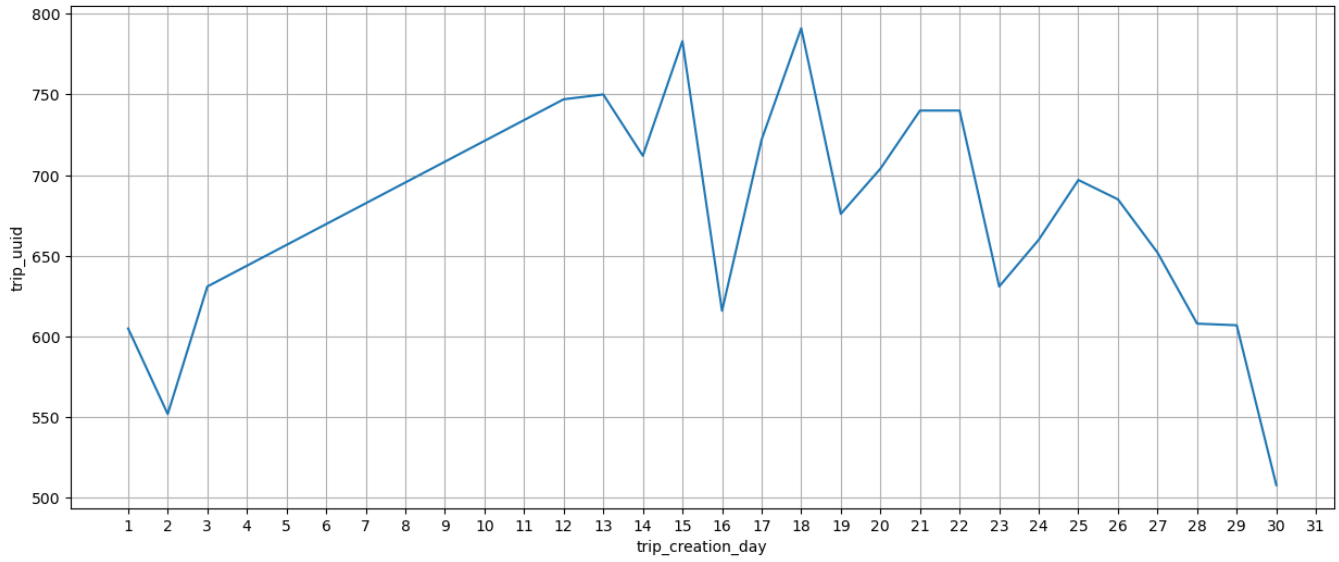
In [64]:
```
df_day = df2.groupby(by = 'trip_creation_day')['trip_uuid'].count().to_frame().reset_index()
df_day.head()
```

Out[64]:

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

In [65]:
```
plt.figure(figsize = (15, 6))
sns.lineplot(data = df_day,
             x = df_day['trip_creation_day'],
             y = df_day['trip_uuid'],
             markers = 'o')
plt.xticks(np.arange(1, 32))
plt.grid('both')
plt.plot()
```

Out[65]:
```
[]
```



Insights:

1. It can be observed from the above plot that most of the trips are created in the mid of the month.

2. That means customers usually request more orders during the mid of the month.

Q3. How many trips are created for different weeks?

In [66]:
```
df2['trip_creation_week'].unique()
```

Out[66]:
```
array([37, 38, 39, 40], dtype=int8)
```

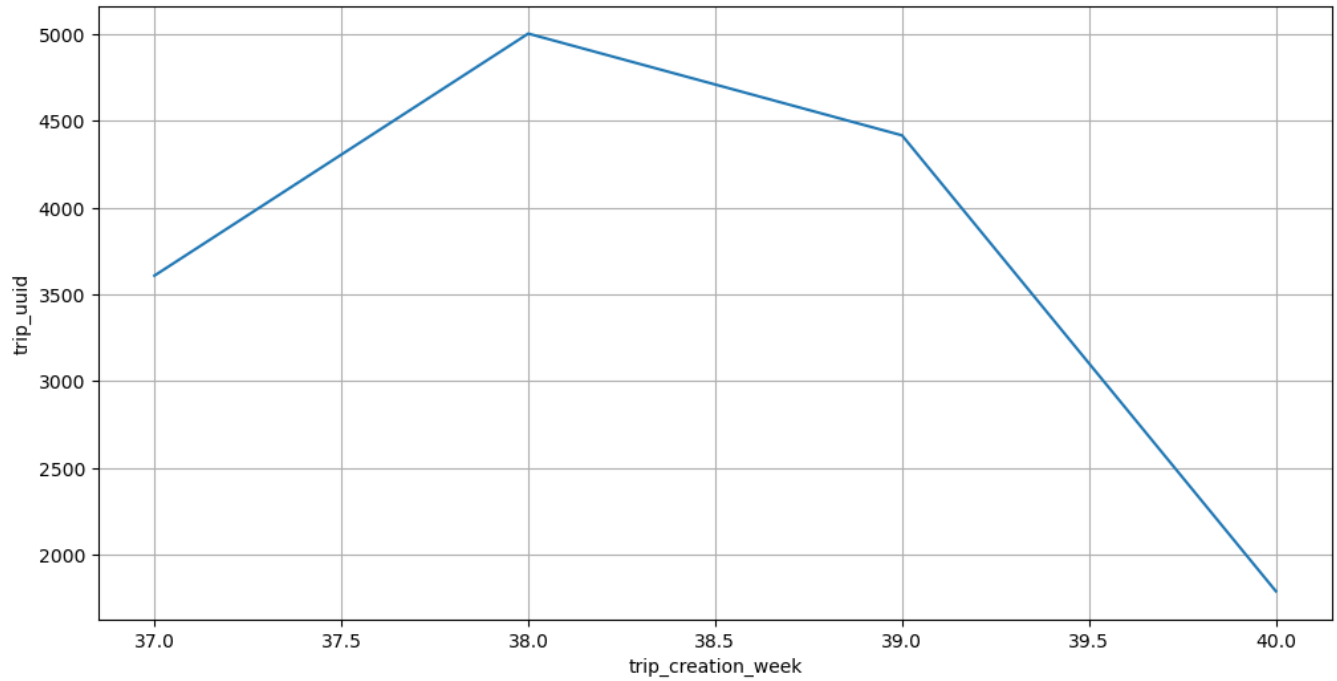
In [67]:
```
df_week = df2.groupby(by = 'trip_creation_week')['trip_uuid'].count().to_frame().reset_index(
df_week.head()
```

Out[67]:

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

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

Out[68]:
```
[]
```



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

Q4. How many trips are created in the given two months?

In [69]:
```
df_month = df2.groupby(by = 'trip_creation_month')['trip_uuid'].count().to_frame().reset_inde
df_month['perc'] = np.round(df_month['trip_uuid'] * 100/ df_month['trip_uuid'].sum(), 2)
df_month.head()
```

Out[69]:

	trip_creation_month	trip_uuid	perc
0	9	13029	87.93
1	10	1788	12.07

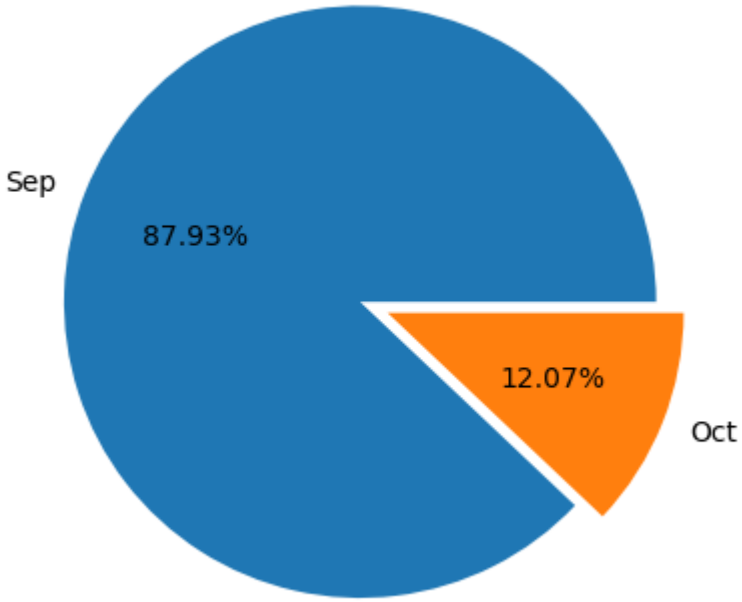
In [70]:
```
df_month = df2.groupby(by = 'trip_creation_month')['trip_uuid'].count().to_frame().reset_inde
df_month['percentage'] = np.round(df_month['trip_uuid'] * 100/ df_month['trip_uuid'].sum(), 2
df_month.head()
```

Out[70]:

	trip_creation_month	trip_uuid	percentage
0	9	13029	87.93
1	10	1788	12.07

In [71]:
```
plt.pie(x = df_month['trip_uuid'],
        labels = ['Sep', 'Oct'],
        explode = [0, 0.1],
       autopct = '%.2f%%')
plt.plot()
```

Out[71]:
```
[]
```



Observations: Trips are created a way more than in September when compared to October

Q5. What is the distribution of trip data for the orders?

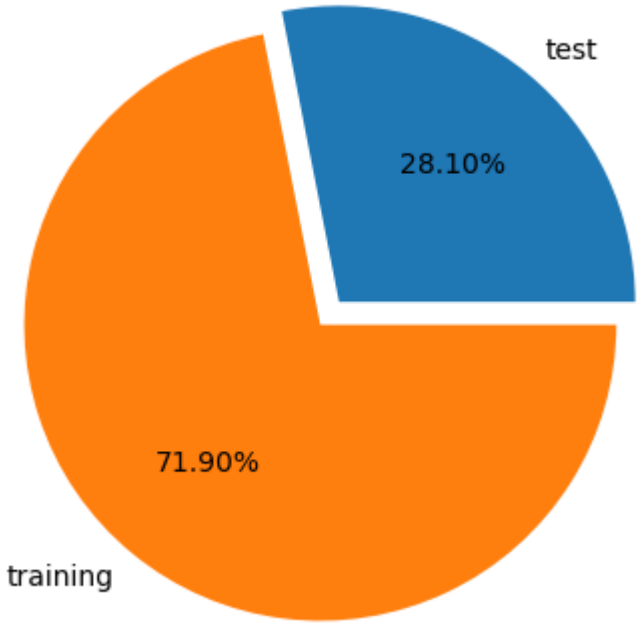
In [72]:
```
df_data = df2.groupby(by = 'data')['trip_uuid'].count().to_frame().reset_index()
df_data['perc'] = np.round(df_data['trip_uuid'] * 100/ df_data['trip_uuid'].sum(), 2)
df_data.head()
```

Out[72]:

	data	trip_uuid	perc
0	test	4163	28.1
1	training	10654	71.9

In [73]:
```
plt.pie(x = df_data['trip_uuid'],
        labels = df_data['data'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

Out[73]:
```
[]
```



Observations: The dataset predominantly comprises the training data.

Q6. What is the distribution of route types for the orders?

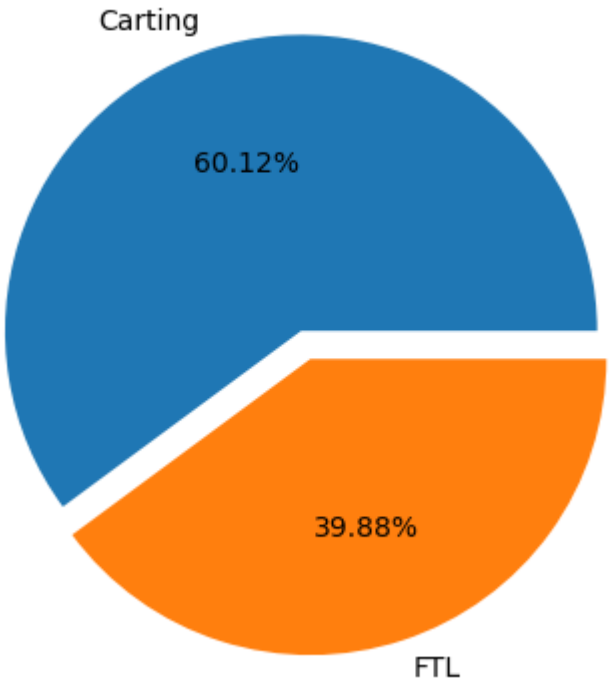
In [74]:
```
df_route = df2.groupby(by = 'route_type')['trip_uuid'].count().to_frame().reset_index()
df_route['percentage'] = np.round(df_route['trip_uuid'] * 100/ df_route['trip_uuid'].sum(), 2
df_route.head()
```

Out[74]:

	route_type	trip_uuid	percentage
0	Carting	8908	60.12
1	FTL	5909	39.88

In [75]:
```
plt.pie(x = df_route['trip_uuid'],
        labels = ['Carting', 'FTL'],
        explode = [0, 0.1],
        autopct = '%.2f%%')
plt.plot()
```

Out[75]:
```
[]
```



Insights:

The distribution of route types reveals a significant prevalence of carting, constituting 60.12% of the total routes, while FTL (Full Truck Load) accounts for the remaining 39.88%. This insight suggests a higher frequency or preference for carting in comparison to FTL, possibly indicating specific operational or logistical considerations within the context of transportation or shipping activities.

Q7. what is the distribution of number of trips created from different states?

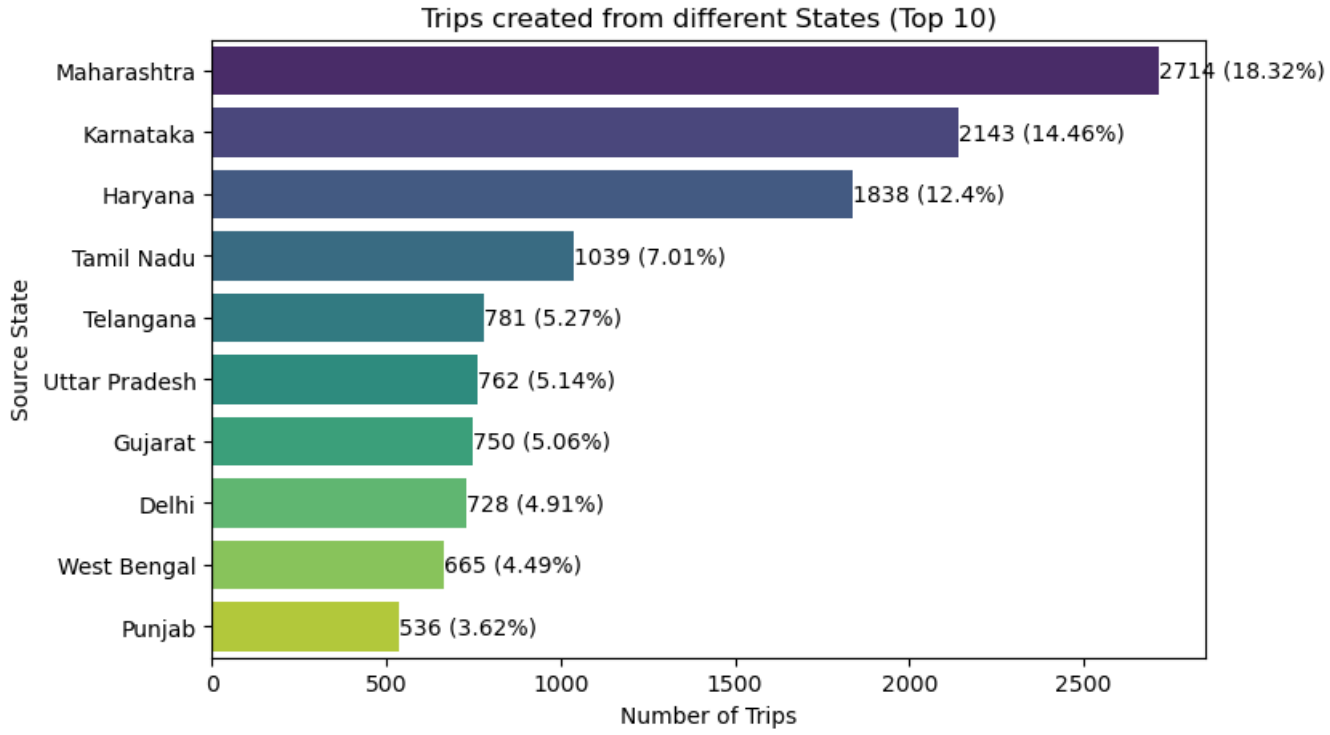
In [76]:
```
df_source_state = df2.groupby(by = 'source_state')['trip_uuid'].count().to_frame().reset_inde
df_source_state['perc'] = np.round(df_source_state['trip_uuid'] * 100/ df_source_state['trip_
df_source_state = df_source_state.sort_values(by = 'trip_uuid', ascending = False)
df_source_state.head()
```

Out[76]:

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

In [95]:
```
plt.figure(figsize=(8, 5))

# Select the top 10 rows
df_top10 = df_source_state.head(10)

# Create a bar plot for the top 10
ax = sns.barplot(data=df_top10,
                 x='trip_uuid',
                 y='source_state',
                 palette='viridis')

# Add labels to each bar
```

```
for index, value in enumerate(df_top10['trip_uuid']):
    ax.text(value, index, f'{value} ({df_top10["perc"].iloc[index]}%)', va='center')

plt.title('Trips created from different States (Top 10)')
plt.xlabel('Number of Trips')
plt.ylabel('Source State')
plt.show()
```



Q8. Find the top 10 cities based on the number of trips created from different cities?

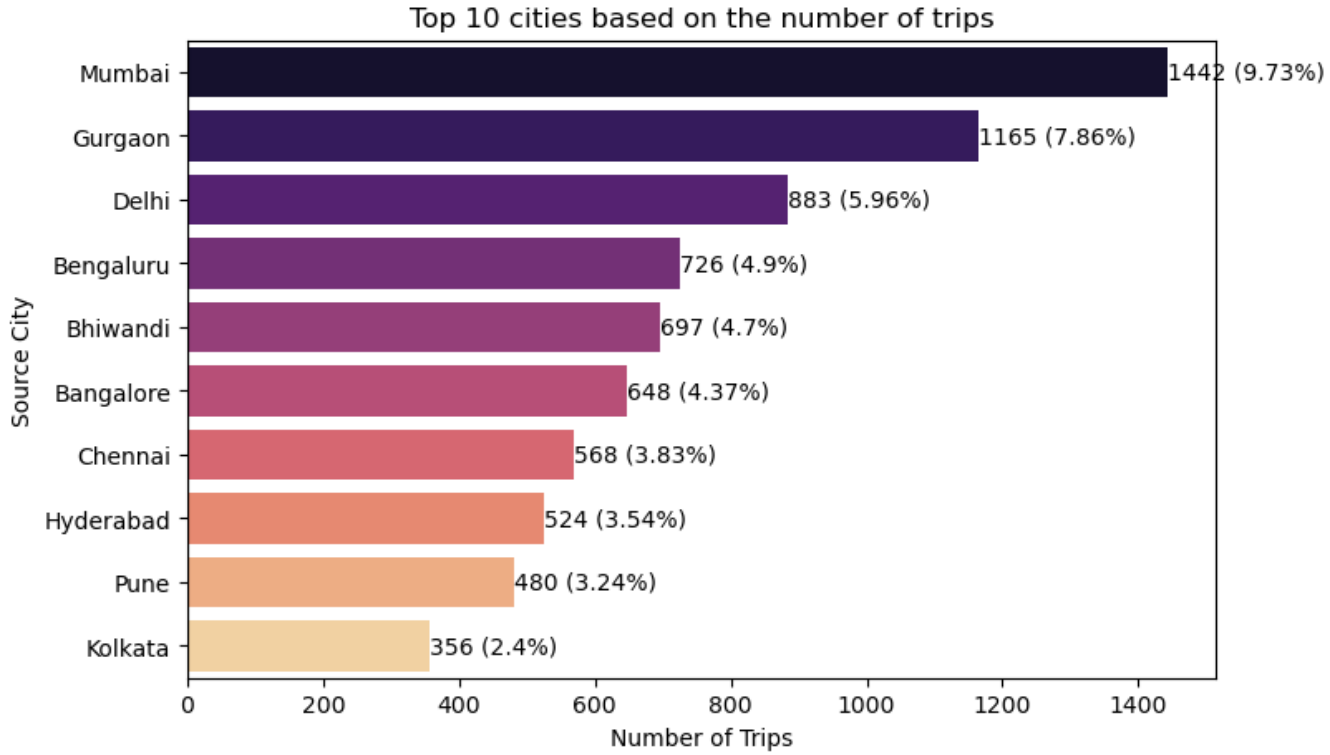
In [87]:
```
df_source_city = df2.groupby(by = 'source_city')['trip_uuid'].count().to_frame().reset_index(
df_source_city['perc'] = np.round(df_source_city['trip_uuid'] * 100/ df_source_city['trip_uui
df_source_city = df_source_city.sort_values(by = 'trip_uuid', ascending = False)[:10]
df_source_city
```

Out[87]:

	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

In [90]:
```
plt.figure(figsize=(8, 5))

# Create a bar plot
ax = sns.barplot(data=df_source_city,
                 x='trip_uuid',
                 y='source_city',
```

```
                palette='magma')

# Add labels to each bar
for index, value in enumerate(df_source_city['trip_uuid']):
    ax.text(value, index, f'{value} ({df_source_city["perc"].iloc[index]}%)', va='center')

plt.title('Top 10 cities based on the number of trips')
plt.xlabel('Number of Trips')
plt.ylabel('Source City')
plt.show()
```



Insights: The plotted data illustrates that the highest number of trips originated from Mumbai, followed by Gurgaon, Delhi, Bengaluru, and Bhiwandi. This pattern indicates a robust seller base in these cities, suggesting a significant presence and potential market strength in these geographical areas.

Q9. what is the distribution of number of trips which ended in different states?

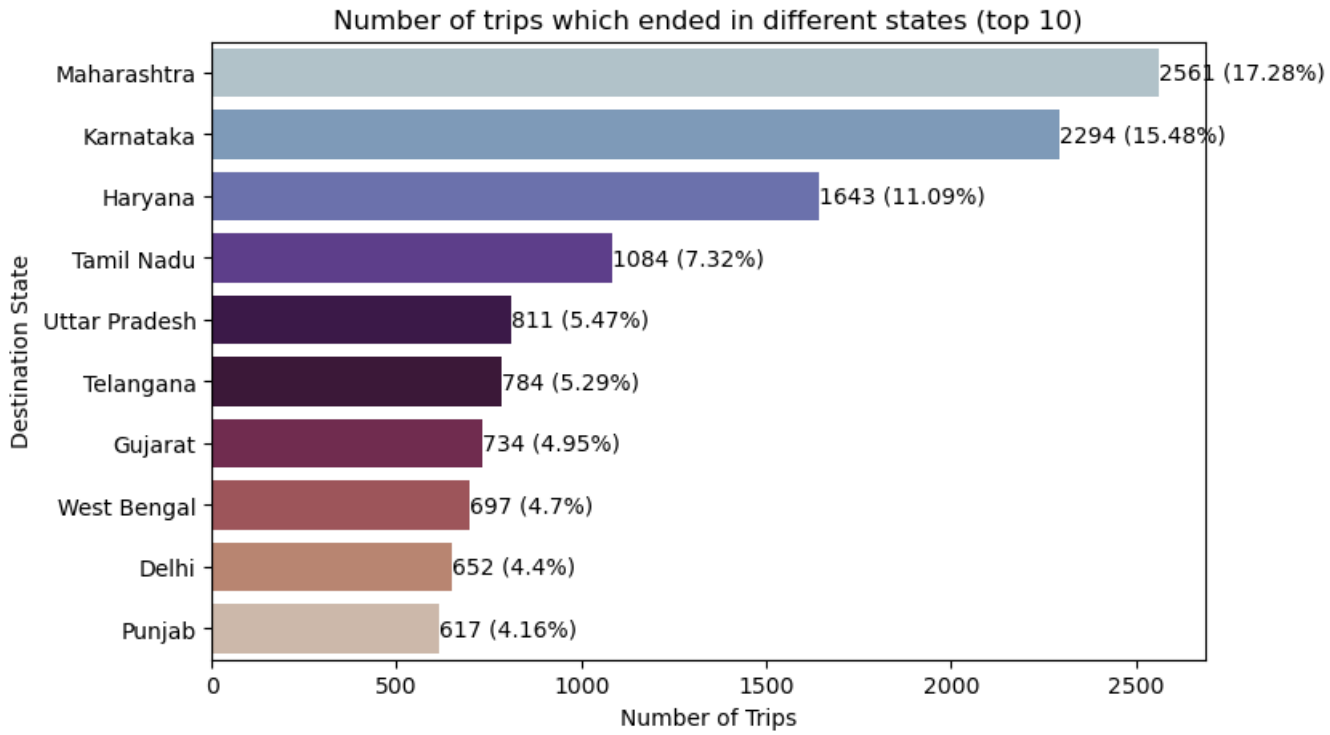
In [98]:
```
df_destination_state = df2.groupby(by = 'destination_state')['trip_uuid'].count().to_frame().
df_destination_state['perc'] = np.round(df_destination_state['trip_uuid'] * 100/ df_destinati
df_destination_state = df_destination_state.sort_values(by = 'trip_uuid', ascending = False)
df_destination_state.head()
```

Out[98]:

	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

In [101…
```
plt.figure(figsize=(8, 5))

df_top10 = df_destination_state.head(10)
```

```
# Create a bar plot
ax = sns.barplot(data=df_top10,
                 x='trip_uuid',
                 y='destination_state',
                 palette='twilight')

# Add labels to each bar
for index, value in enumerate(df_top10['trip_uuid']):
    ax.text(value, index, f'{value} ({df_top10["perc"].iloc[index]}%)', va='center')

plt.title('Number of trips which ended in different states (top 10)')
plt.xlabel('Number of Trips')
plt.ylabel('Destination State')
plt.show()
```



Insights:

The plotted data reveals that the highest number of trips concluded in the Maharashtra state, followed by Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh. This observation indicates a notably elevated frequency of orders in these states, suggesting a substantial demand or transaction volume in these regions.

Q10. Find the top 10 cities based on the number of trips ended in different cities?

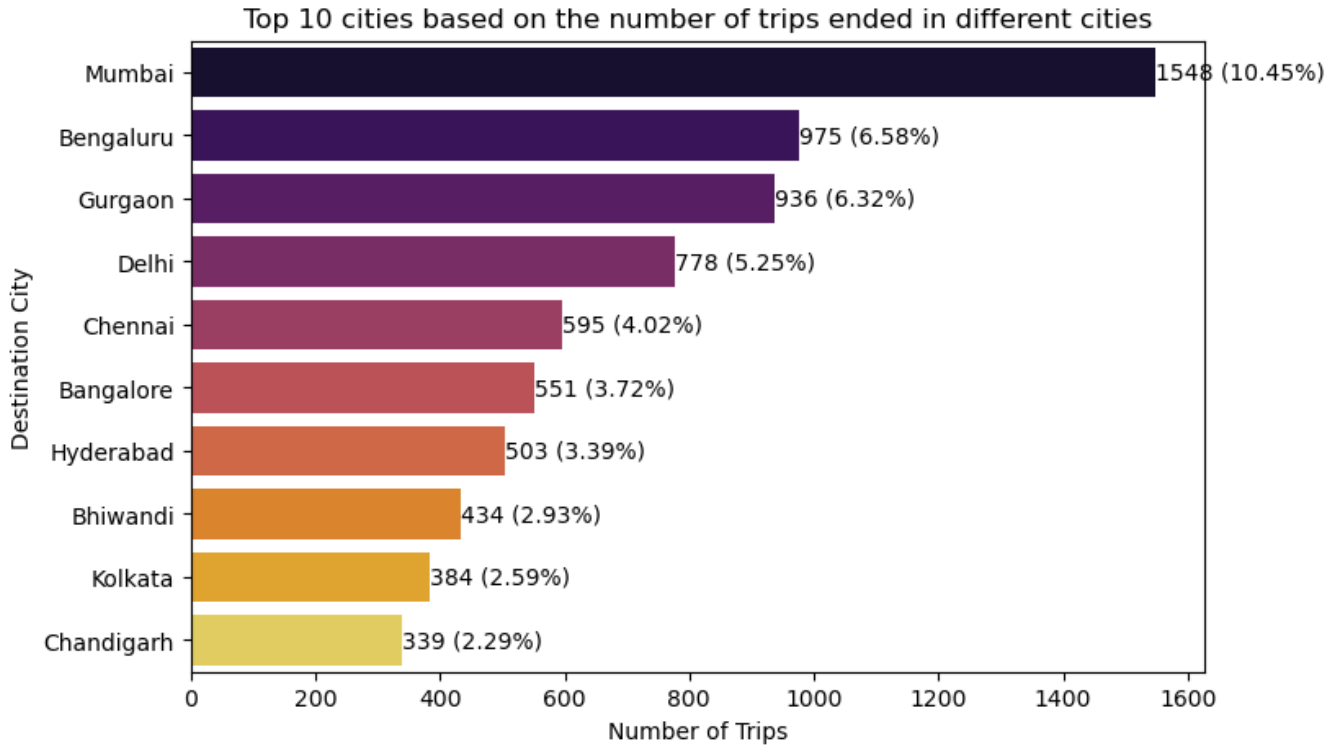
In [91]:
```
df_destination_city = df2.groupby(by = 'destination_city')['trip_uuid'].count().to_frame().re
df_destination_city['perc'] = np.round(df_destination_city['trip_uuid'] * 100/ df_destination
df_destination_city = df_destination_city.sort_values(by = 'trip_uuid', ascending = False)[:1
df_destination_city
```

Out[91]:

	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

In [93]:

```
plt.figure(figsize=(8, 5))

# Create a bar plot
ax = sns.barplot(data=df_destination_city,
                 x='trip_uuid',
                 y='destination_city',
                 palette='inferno')

# Add labels to each bar
for index, value in enumerate(df_destination_city['trip_uuid']):
    ax.text(value, index, f'{value} ({df_destination_city["perc"].iloc[index]}%)', va='center

plt.title('Top 10 cities based on the number of trips ended in different cities')
plt.xlabel('Number of Trips')
plt.ylabel('Destination City')
plt.show()
```



Insights:

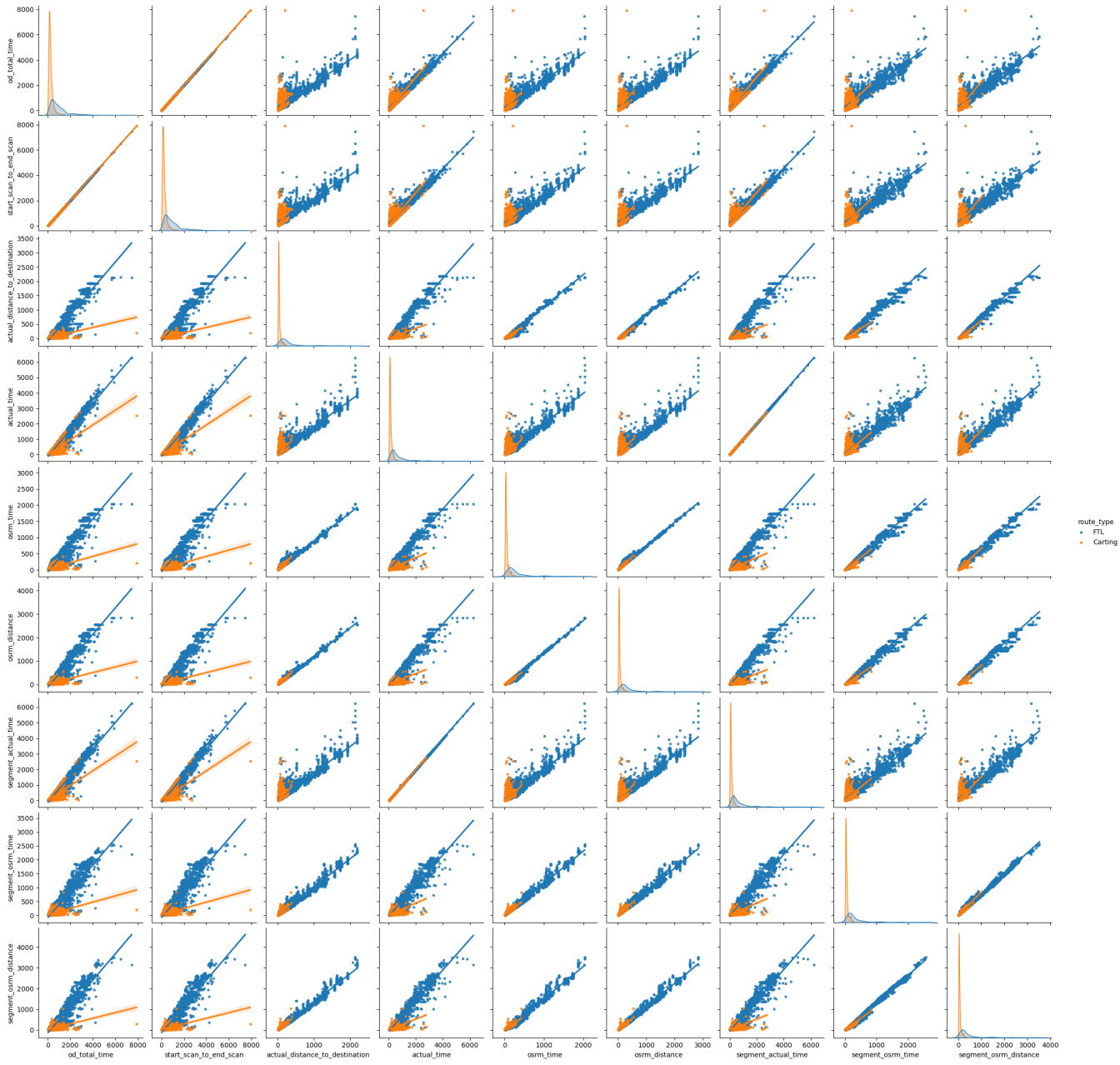
The plotted data indicates that the majority of trips concluded in Mumbai city, with Bengaluru, Gurgaon, Delhi, and Chennai following closely. This pattern suggests a

notably high volume of orders in these cities, highlighting a substantial level of demand or transaction activity.

7. Distribution of the variables and relationship between them

In [84]:
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance_to_destinati
                     '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()
```

Out[84]:
```
[]
```



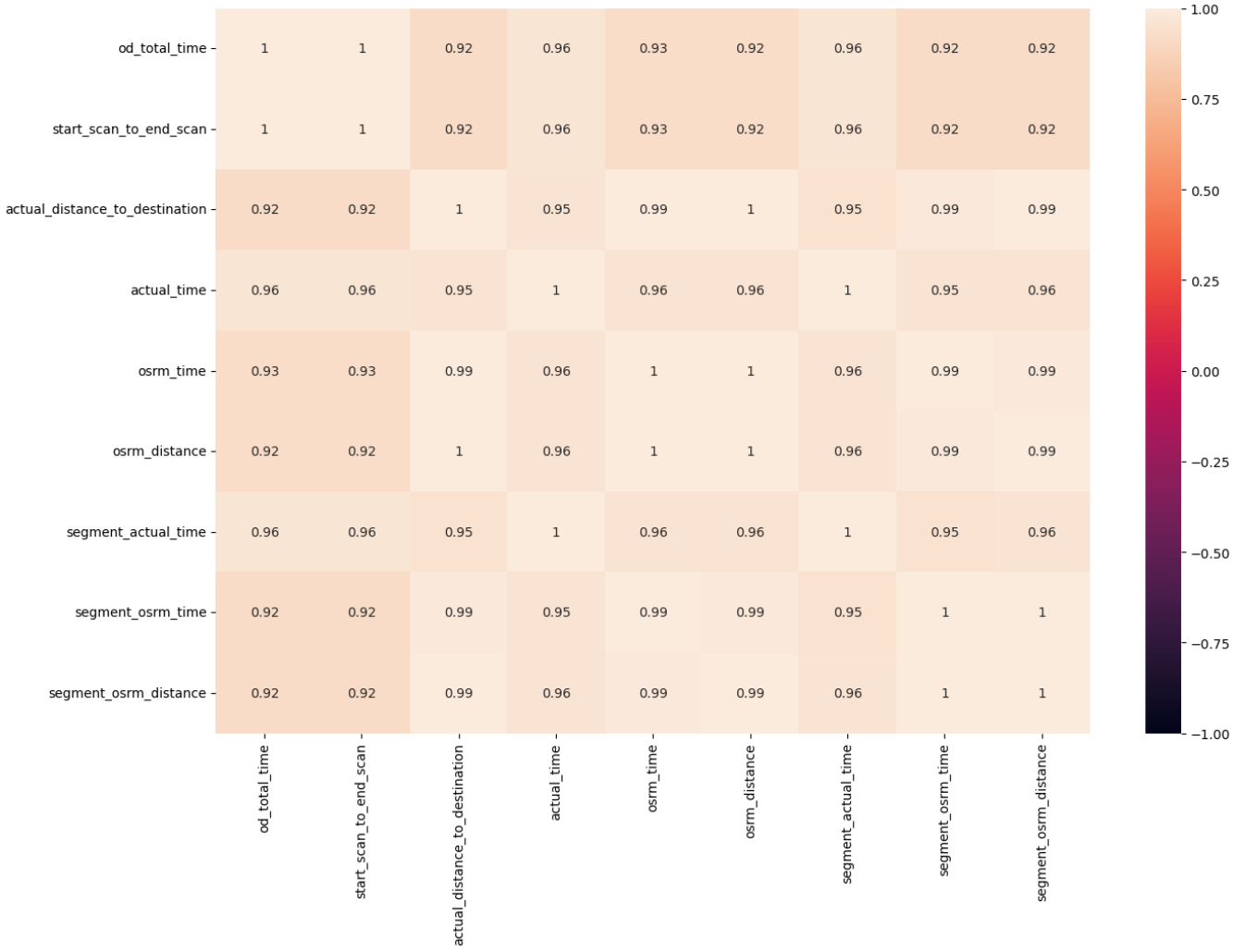
In [85]:
```
df_corr = df2[numerical_columns].corr()
df_corr
```

Out[85]:

	od_total_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time
od_total_time	1.000000	0.999999	0.918222	0.961094
start_scan_to_end_scan	0.999999	1.000000	0.918308	0.961147
actual_distance_to_destination	0.918222	0.918308	1.000000	0.953757
actual_time	0.961094	0.961147	0.953757	1.000000
osrm_time	0.926516	0.926571	0.993561	0.958593
osrm_distance	0.924219	0.924299	0.997264	0.959214
segment_actual_time	0.961119	0.961171	0.952821	0.999989
segment_osrm_time	0.918490	0.918561	0.987538	0.953872
segment_osrm_distance	0.919199	0.919291	0.993061	0.956967

In [86]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, vmin = -1, vmax = 1, annot = True)
plt.plot()
```

Out[86]:
```
[]
```



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

8. In-depth analysis and feature engineering

a.) Comparing 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.

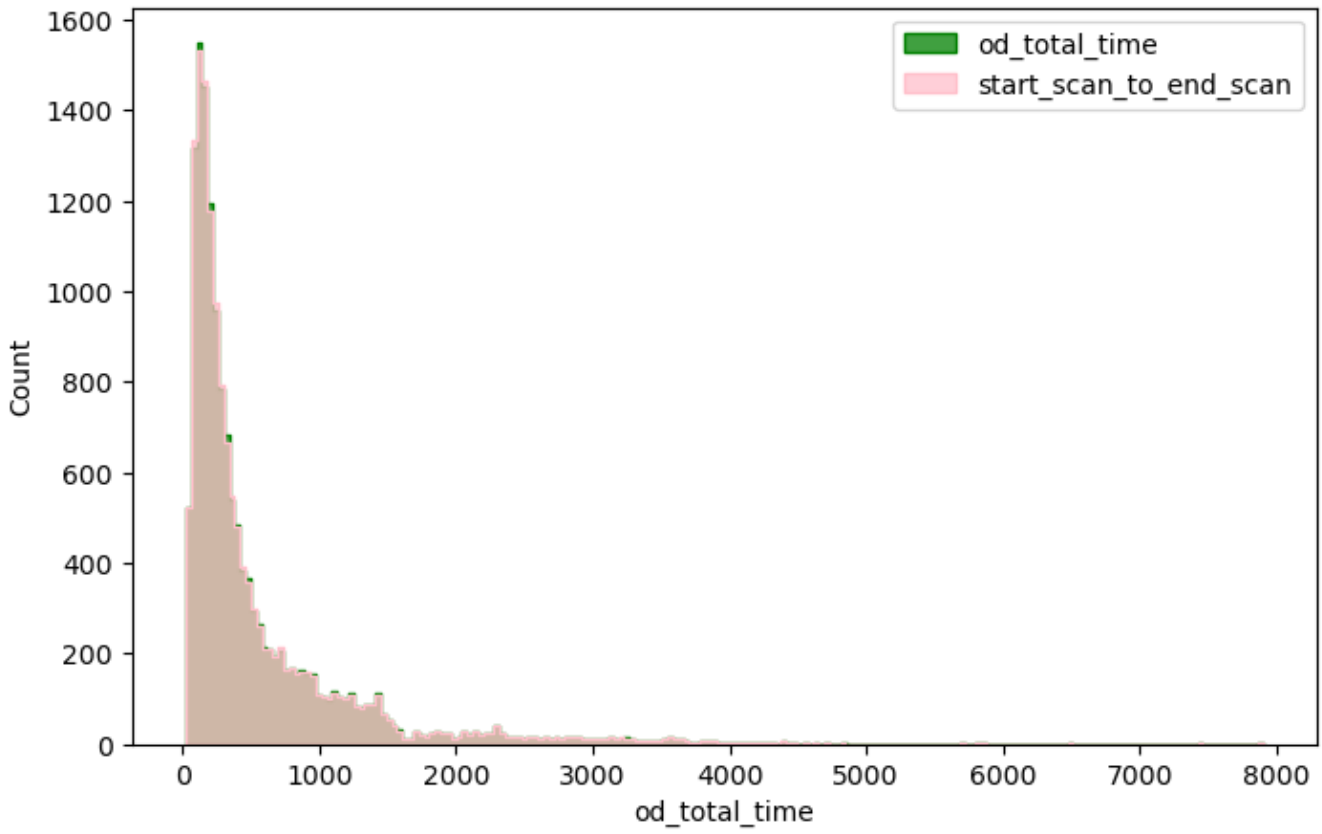
Alternate Hypothesis (HA) - od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected total trip time) are different.

STEP-2: Checking for basic assumpitons for the hypothesis

Visual Tests to know if the samples follow normal distribution

In [109…
```
plt.figure(figsize = (8, 5))
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()
```

Out[109]:
```
[]
```



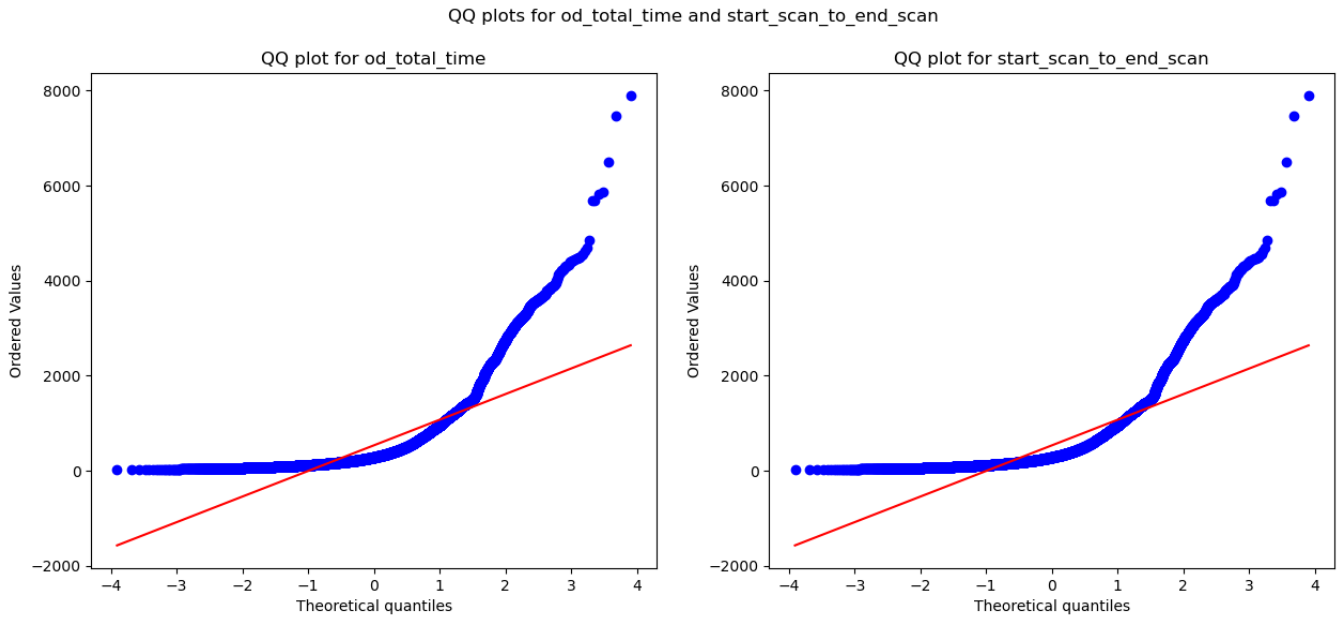
Observation: The above plot does not follow the normal distribution

Distribution check using QQ Plot

In [110…
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for od_total_time and start_scan_to_end_scan')
spy.probplot(df2['od_total_time'], plot = plt, dist = 'norm')
```

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

Out[110]: []



Observation: The above plot that the samples do not come from normal distribution

In [102…
```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```

Out[102]:

	od_total_time	start_scan_to_end_scan
count	14817.000000	14817.000000
mean	531.697630	530.809998
std	658.868223	658.707031
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

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

In [112…
```
test_stat, p_value = spy.shapiro(df2['od_total_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

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

In [113…
```
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
```

Homogeneity of Variances using Lavene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [114…
```
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 0.9668020786253313
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here

we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [115…
```
test_stat, p_value = spy.mannwhitneyu(df2['od_total_time'], df2['start_scan_to_end_scan'])
print('P-value :',p_value)
if p_value < 0.05:
    print('od_total_time and start_scan_to_end_scan are different')
else:
    print('od_total_time and start_scan_to_end_scan are similar')
```

```
P-value : 0.7815123224221716
od_total_time and start_scan_to_end_scan are similar
```

Conclusion: od_total_time and start_scan_to_end_scan are similar

b.) Hypothesis testing / visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values after merging the rows on the basis of trip_uuid)

In [116…
```
df2[['actual_time', 'osrm_time']].describe()
```

Out[116]:

	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

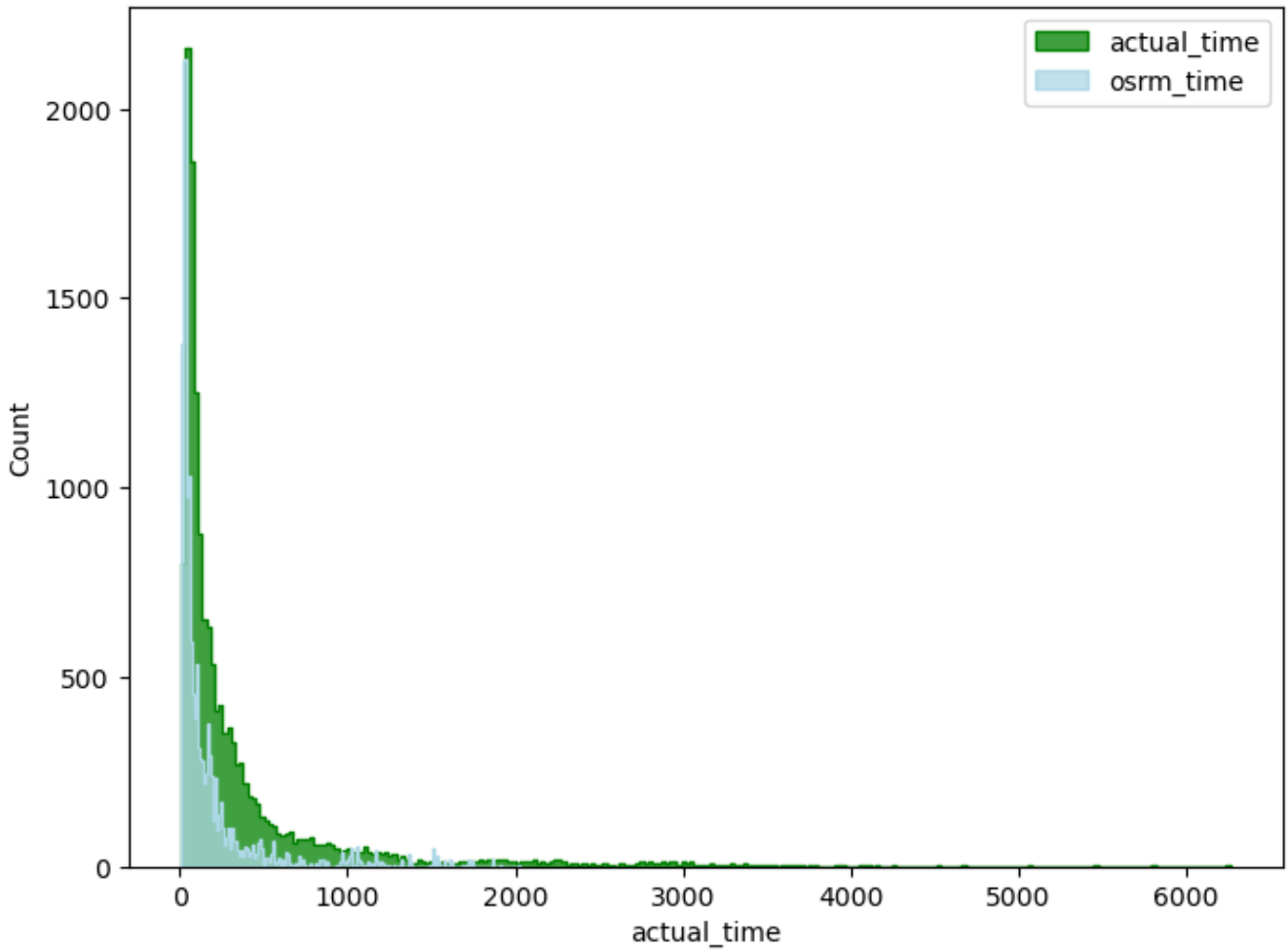
Visual Tests to know if the samples follow normal distribution

In [121…

```
plt.figure(figsize = (8, 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()
```

Out[121]:

```
[]
```



Observation: The above plot does not follow the normal distribution

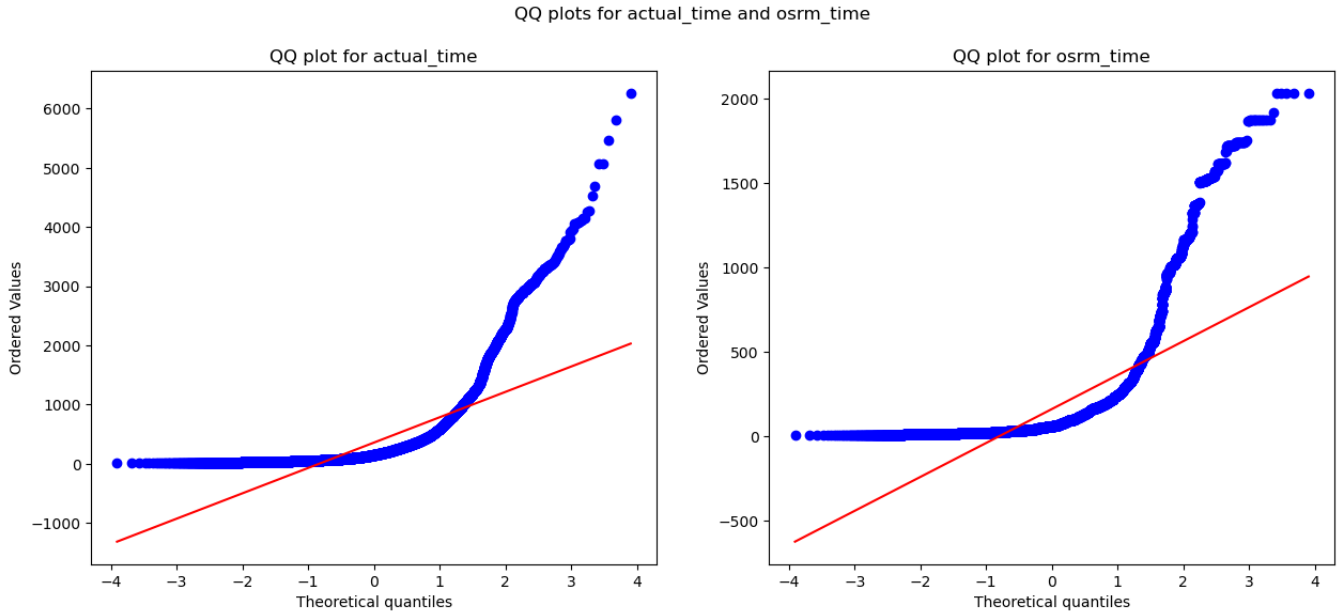
Distribution check using QQ Plot

In [122…

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and osrm_time')
```

```
spy.probplot(df2['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(df2['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.plot()
```

Out[122]: []



The above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

In [124…

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

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

In [125…

```
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
```

Homogeneity of Variances using Lavene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [126…
```
test_stat, p_value = spy.levene(df2['actual_time'], df2['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

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

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

In [127…
```
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
```

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

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

In [128…
```
df2[['actual_time', 'segment_actual_time']].describe()
```

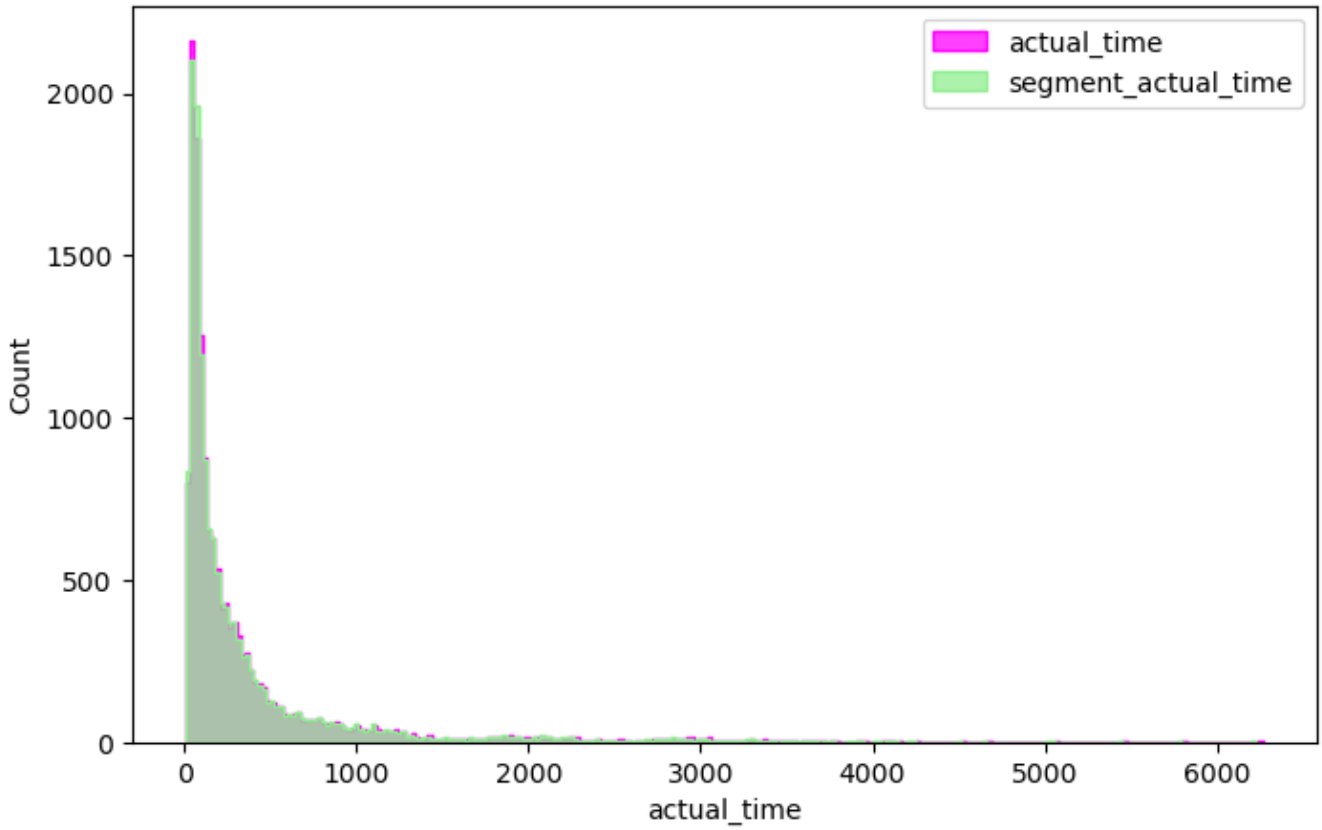
Out[128]:

	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

In [130…
```
plt.figure(figsize = (8, 5))
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()
```

Out[130]:
```
[]
```



Observation: The above plot does not follow the normal distribution

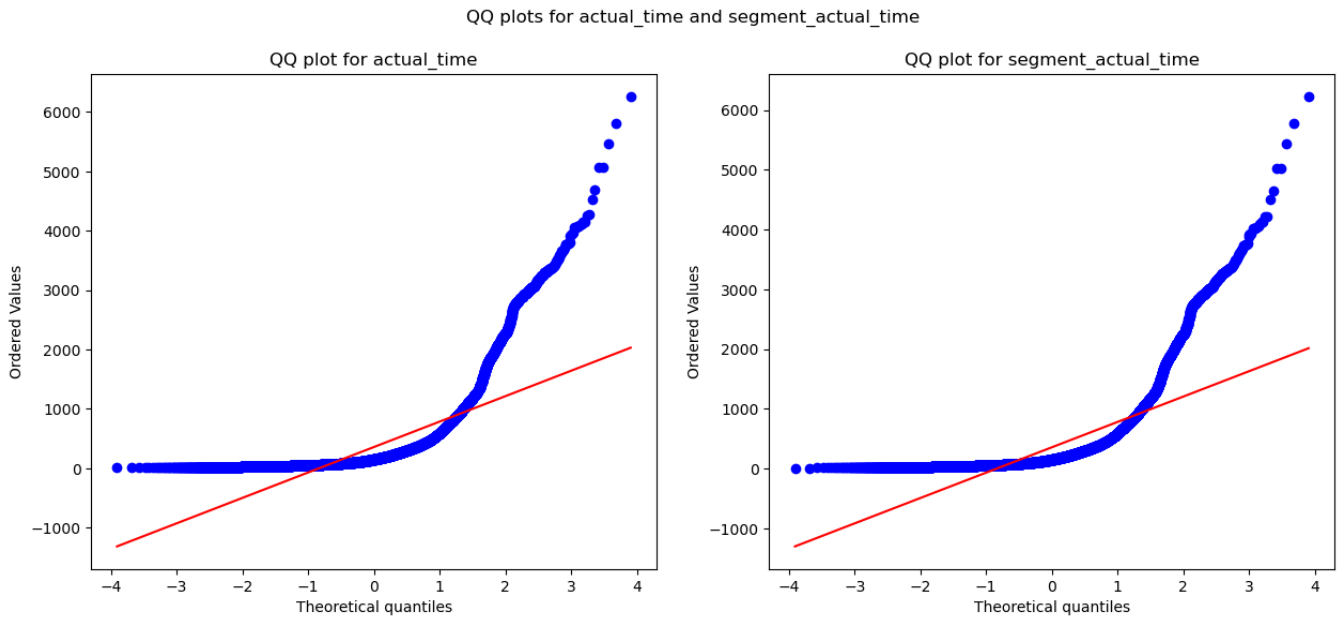
Distribution check using QQ Plot

In [131…

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

Out[131]:

```
[]
```



Observation: The above plot that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

In [133…
```
test_stat, p_value = spy.shapiro(df2['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

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

In [134…
```
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
```

Homogeneity of Variances using Lavene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [135…
```
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.

In [136…
```
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
```

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

d.) hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values after merging the rows on the basis of trip_uuid)

In [138…
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

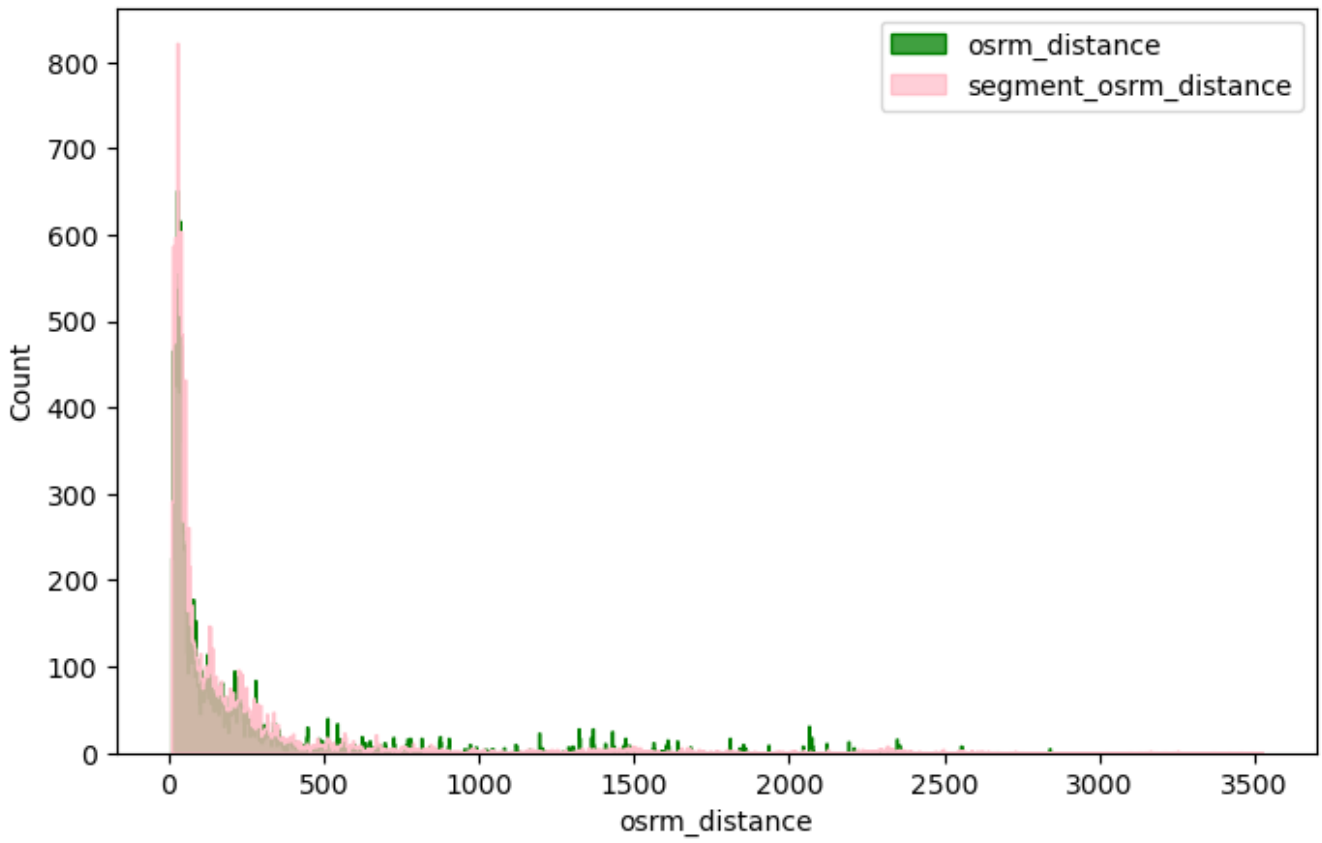
Out[138]:

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

Visual Tests to know if the samples follow normal distribution

In [140…
```
plt.figure(figsize = (8, 5))
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()
```

Out[140]:
```
[]
```

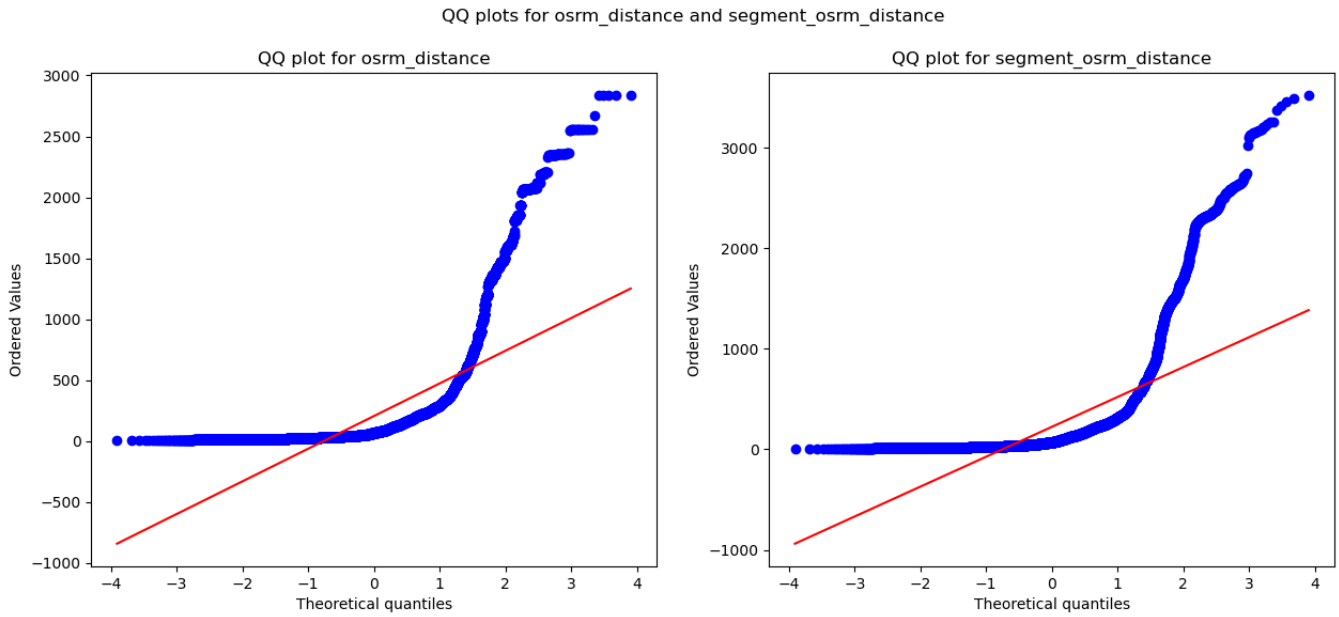


Observation: The above plot does not follow the normal distribution

Distribution check using QQ Plot

In [141…
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_distance and segment_osrm_distance')
spy.probplot(df2['osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_distance')
plt.subplot(1, 2, 2)
spy.probplot(df2['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_distance')
plt.plot()
```

Out[141]:
```
[]
```



Observation: The above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0: The sample follows normal distribution

Ha: The sample does not follow normal distribution

alpha = 0.05

In [143…
```
test_stat, p_value = spy.shapiro(df2['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

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

In [144…
```
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
```

Homogeneity of Variances using Lavene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [146…
```
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.

In [147…
```
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 9.509410818847664e-07
The samples are not similar
```

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

e.) Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values after merging the rows on the basis of trip_uuid)

In [148…
```
df2[['osrm_time', 'segment_osrm_time']].describe().T
```

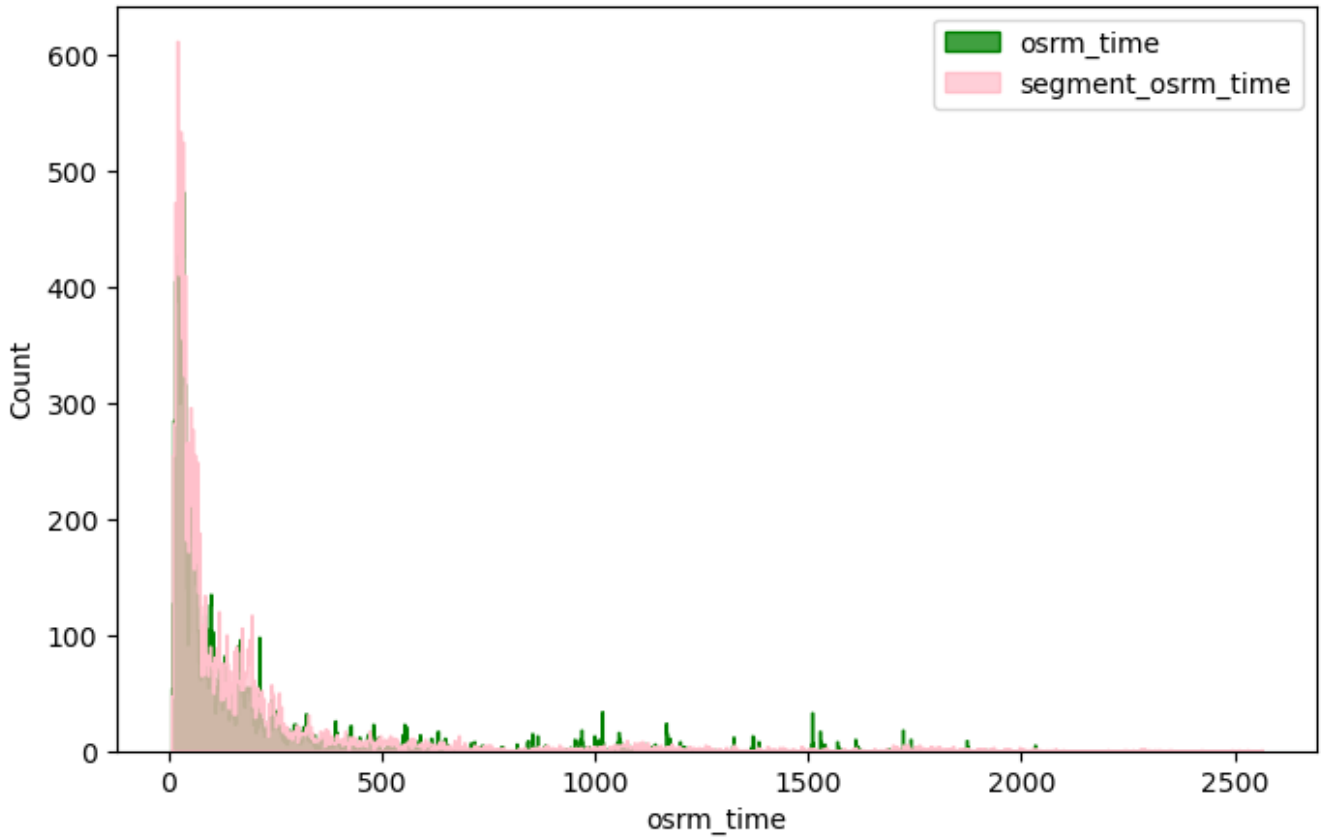
Out[148]:

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

Visual Tests to know if the samples follow normal distribution

In [149…
```
plt.figure(figsize = (8, 5))
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()
```

Out[149]:
```
[]
```



Observation: The above plot does not follow the normal distribution

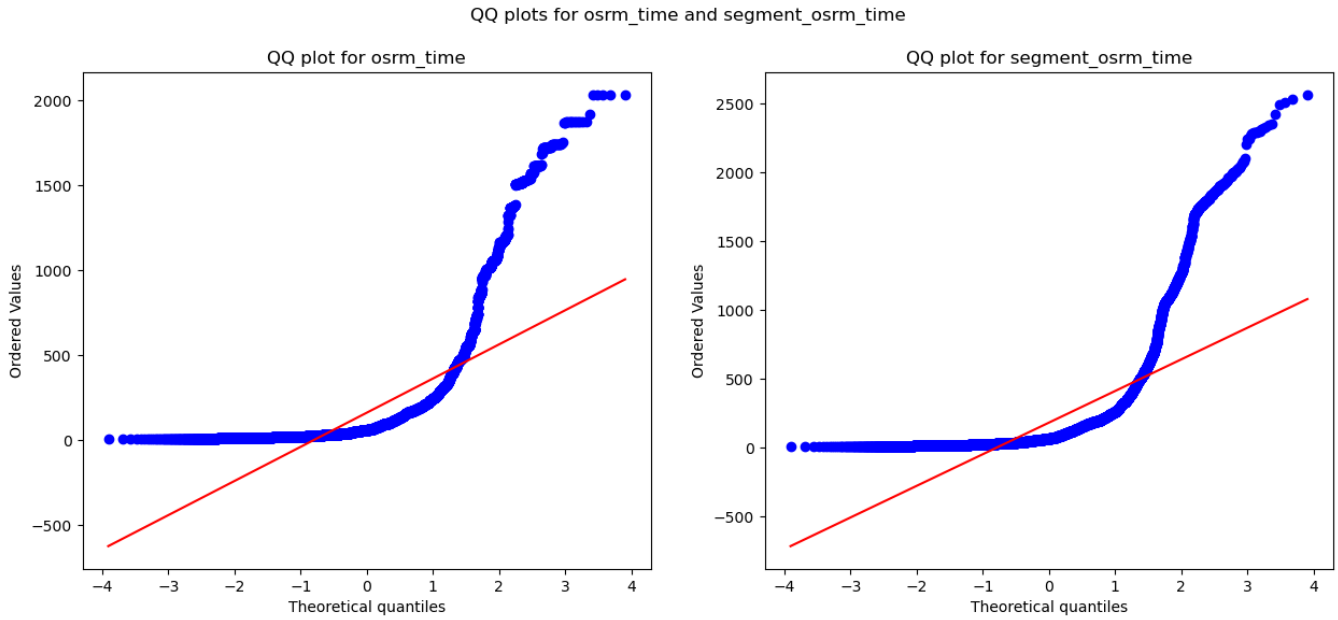
Distribution check using QQ Plot

In [150…

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

Out[150]:

```
[]
```



Observation: The above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0: The sample follows normal distribution

Ha: The sample does not follow normal distribution

alpha = 0.05

In [151…
```
test_stat, p_value = spy.shapiro(df2['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

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

In [152…
```
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
```

Homogeneity of Variances using Lavene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [153…
```
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.

In [154…
```
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
```

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

8. Outlier Treatment

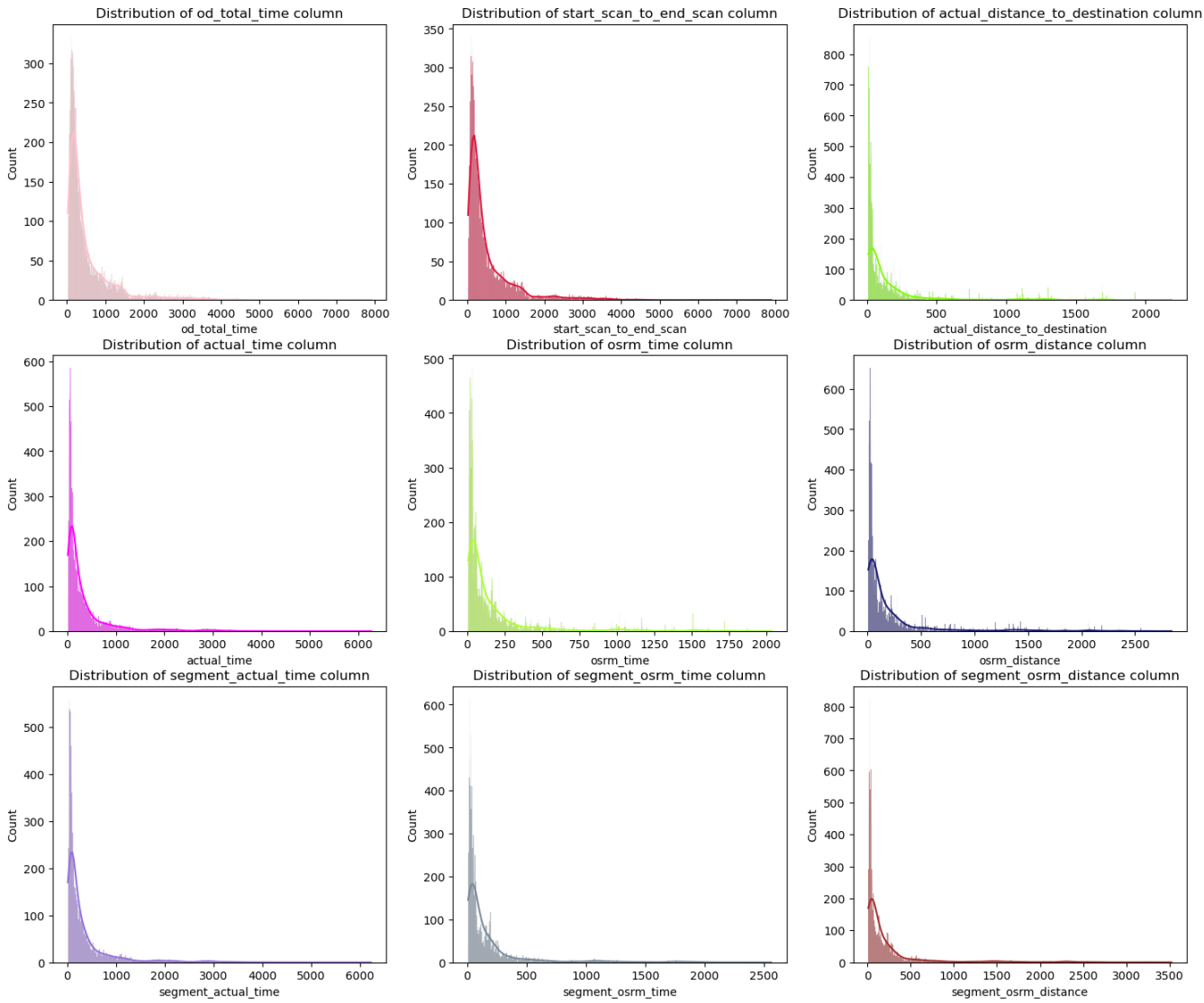
Finding outliers in the numerical variables and checking it using visual analysis

In [155…]
```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance_to_destinati
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']
df2[numerical_columns].describe().T
```

Out[155]:

	count	mean	std	min	25%	50%	75%
od_total_time	14817.0	531.697630	658.868223	23.460000	149.930000	280.770000	638.200000
start_scan_to_end_scan	14817.0	530.809998	658.707031	23.000000	149.000000	280.000000	637.000000
actual_distance_to_destination	14817.0	164.477829	305.388123	9.002461	22.837238	48.474072	164.583206
actual_time	14817.0	357.143768	561.395020	9.000000	67.000000	149.000000	370.000000
osrm_time	14817.0	161.384018	271.362549	6.000000	29.000000	60.000000	168.000000
osrm_distance	14817.0	204.344711	370.395508	9.072900	30.819201	65.618805	208.475006
segment_actual_time	14817.0	353.892273	556.246826	9.000000	66.000000	147.000000	367.000000
segment_osrm_time	14817.0	180.949783	314.541412	6.000000	31.000000	65.000000	185.000000
segment_osrm_distance	14817.0	223.201157	416.628326	9.072900	32.654499	70.154404	218.802399

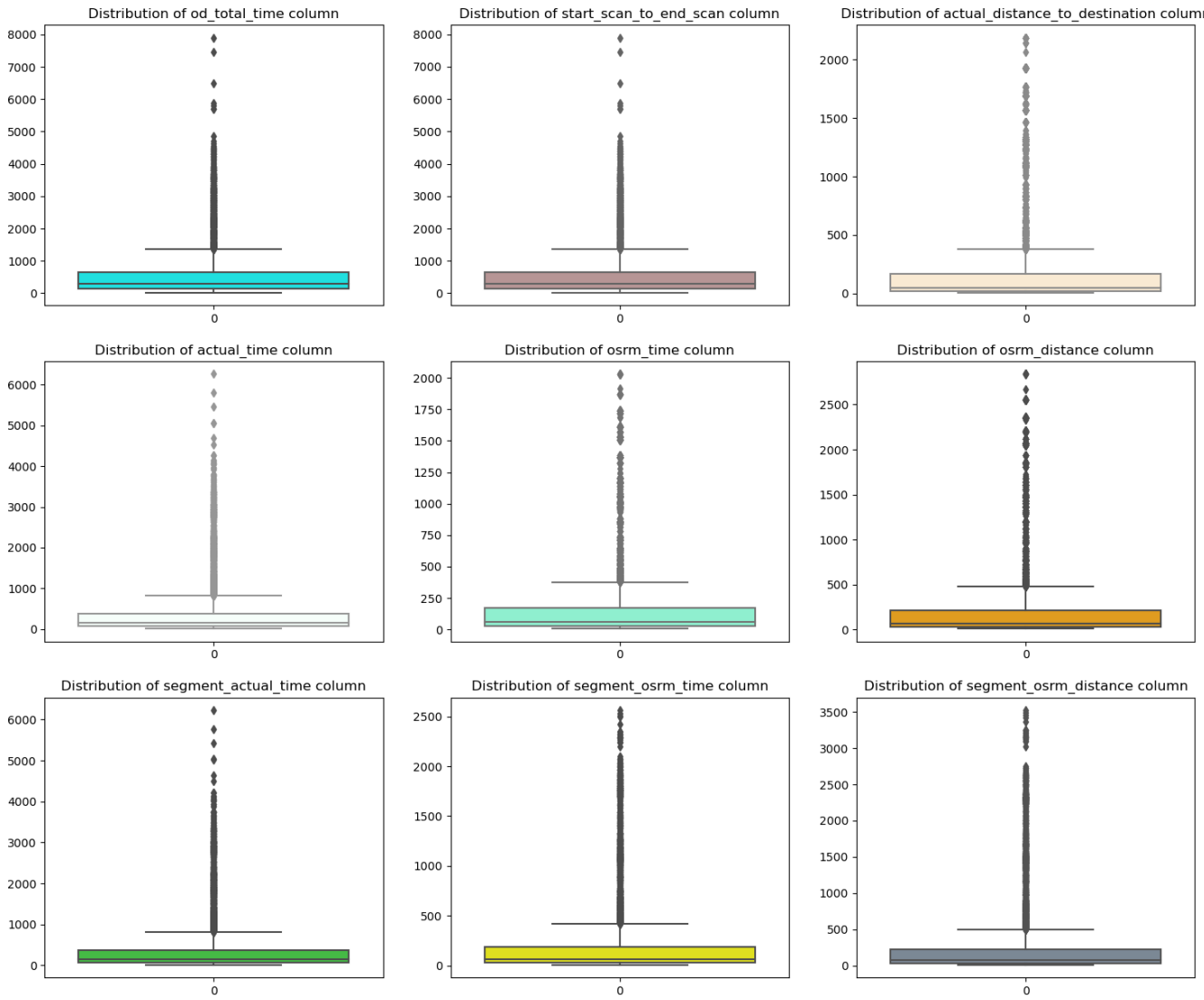
In [156…]
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.histplot(df2[numerical_columns[i]], bins = 1000, kde = True, color = clr)
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



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

In [157…

```
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()
```



Observation: The above plots that there are outliers in all the numerical columns that need to be treated.

Handling the outliers using the IQR method

In [158…

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

```
Column : od_total_time
Q1 : 149.93
Q3 : 638.2
IQR : 488.27000000000004
LB : -582.4750000000001
UB : 1370.605
Number of outliers : 1266
---------------------------------
Column : start_scan_to_end_scan
Q1 : 149.0
Q3 : 637.0
IQR : 488.0
LB : -583.0
UB : 1369.0
Number of outliers : 1267
---------------------------------
Column : actual_distance_to_destination
Q1 : 22.837238311767578
Q3 : 164.5832061767578
IQR : 141.74596786499023
LB : -189.78171348571777
UB : 377.20215797424316
Number of outliers : 1449
---------------------------------
Column : actual_time
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
LB : -387.5
UB : 824.5
Number of outliers : 1643
---------------------------------
Column : osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
LB : -179.5
UB : 376.5
Number of outliers : 1517
---------------------------------
Column : osrm_distance
Q1 : 30.81920051574707
Q3 : 208.47500610351562
IQR : 177.65580558776855
LB : -235.66450786590576
UB : 474.95871448516846
Number of outliers : 1524
---------------------------------
Column : segment_actual_time
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
LB : -385.5
UB : 818.5
Number of outliers : 1643
---------------------------------
Column : segment_osrm_time
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
LB : -200.0
UB : 416.0
Number of outliers : 1492
---------------------------------
Column : segment_osrm_distance
Q1 : 32.65449905395508
Q3 : 218.80239868164062
IQR : 186.14789962768555
LB : -246.56735038757324
```

```
UB : 498.02424812316895
Number of outliers : 1548
---------------------------------
```

Conclusion: 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.

8. Dealing with categorical data by one-hot encoding of categorical variables

In [159…]
```
# Get value counts before one-hot encoding

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

Out[159]:
```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

In [160…]
```
# Perform one-hot encoding on categorical column route type
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df2['route_type'] = label_encoder.fit_transform(df2['route_type'])
```

In [161…]
```
# Get value counts after one-hot encoding

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

Out[161]:
```
0    8908
1    5909
Name: route_type, dtype: int64
```

In [162…]
```
# Get value counts of categorical variable 'data' before one-hot encoding

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

Out[162]:
```
training    10654
test         4163
Name: data, dtype: int64
```

In [163…]
```
# Performing one-hot encoding on categorical variable 'data'
label_encoder = LabelEncoder()
df2['data'] = label_encoder.fit_transform(df2['data'])
```

In [164…]
```
# Get value counts after one-hot encoding

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

Out[164]:
```
1    10654
0     4163
Name: data, dtype: int64
```

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

In [166…]
```
#importing required library
from sklearn.preprocessing import MinMaxScaler
```

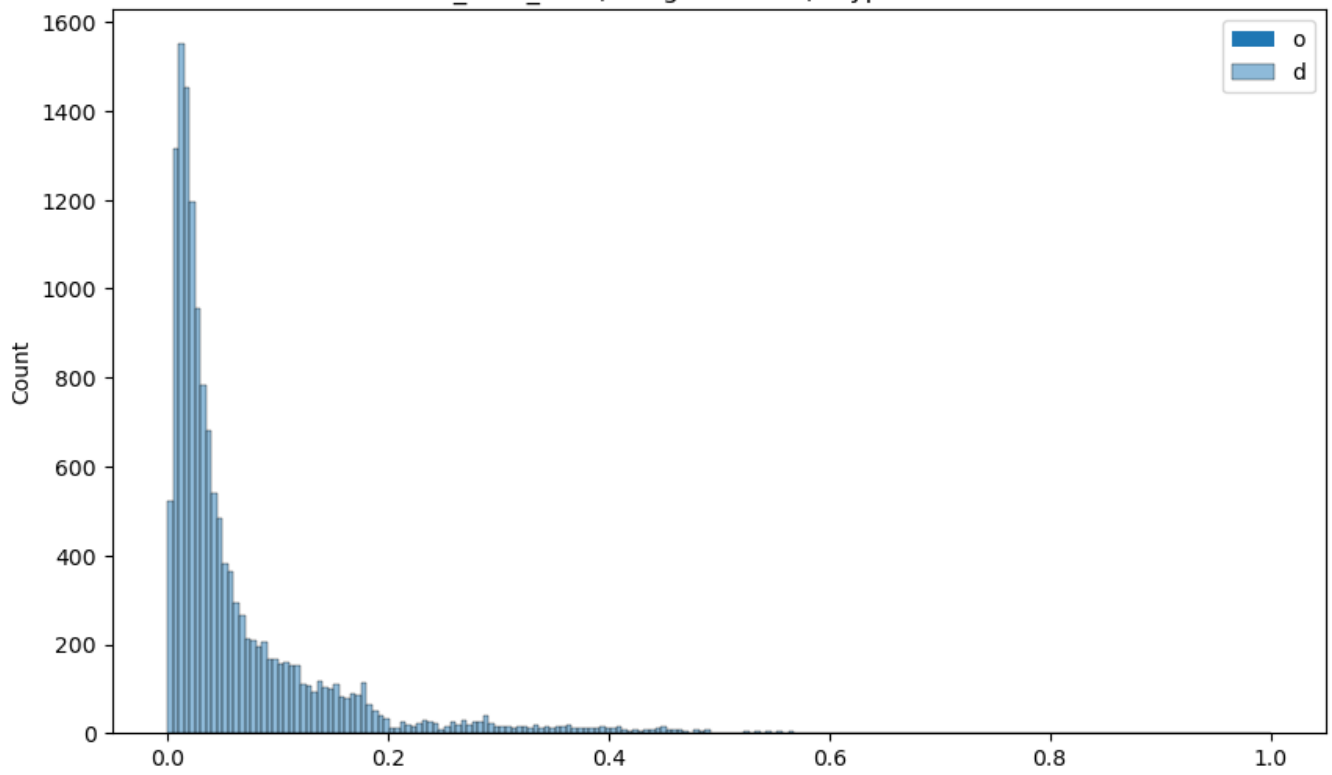
In [167…]
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['od_total_time']} column")
```

```
plt.legend('od_total_time')
plt.plot()
```

Out[167]:

```
[]
```

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



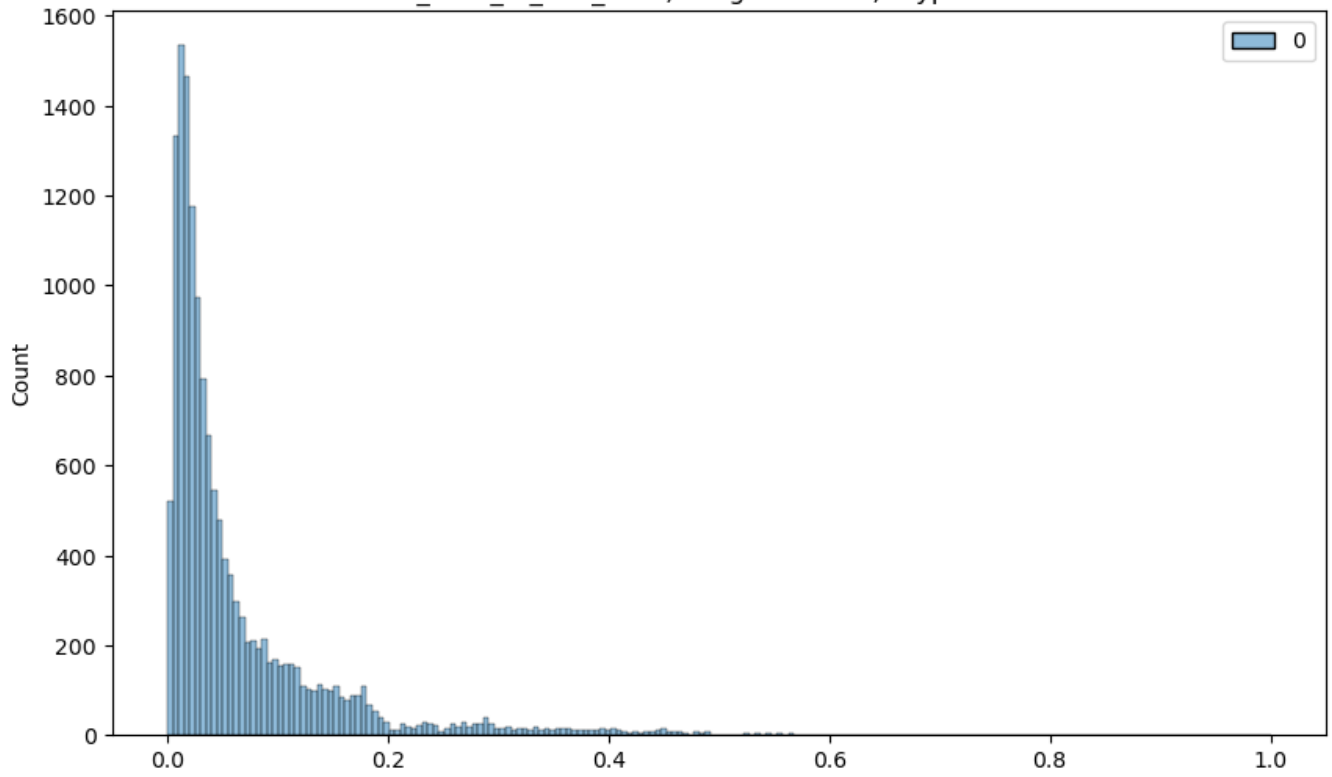
In [168…

```
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()
```

Out[168]:

```
[]
```

```
Normalized 0       2259.0
           1        180.0
           2       3933.0
           3        100.0
           4        717.0
                     ...
       14812        257.0
       14813         60.0
       14814        421.0
       14815        347.0
       14816        353.0
Name: start_scan_to_end_scan, Length: 14817, dtype: float32 column
```



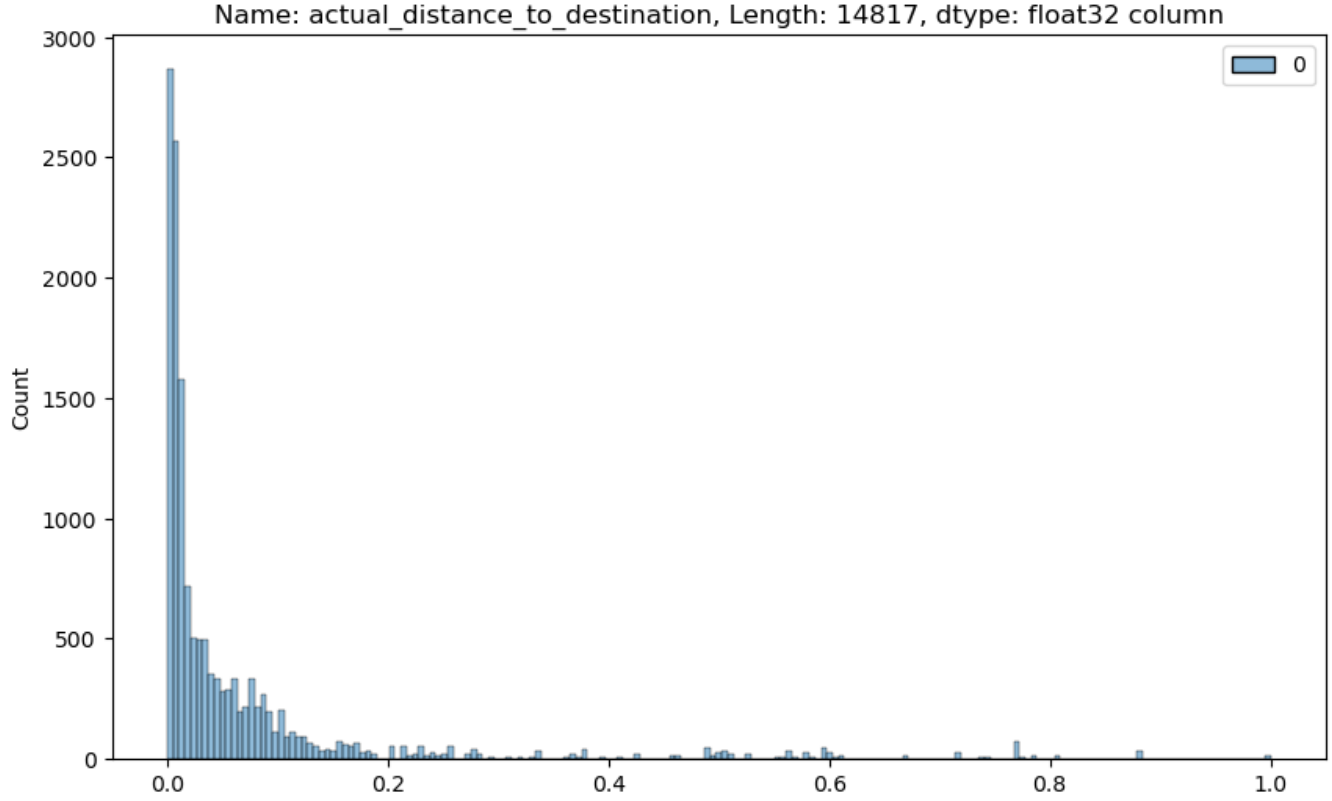
In [169…

```
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()
```

Out[169]:

```
[]
```

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



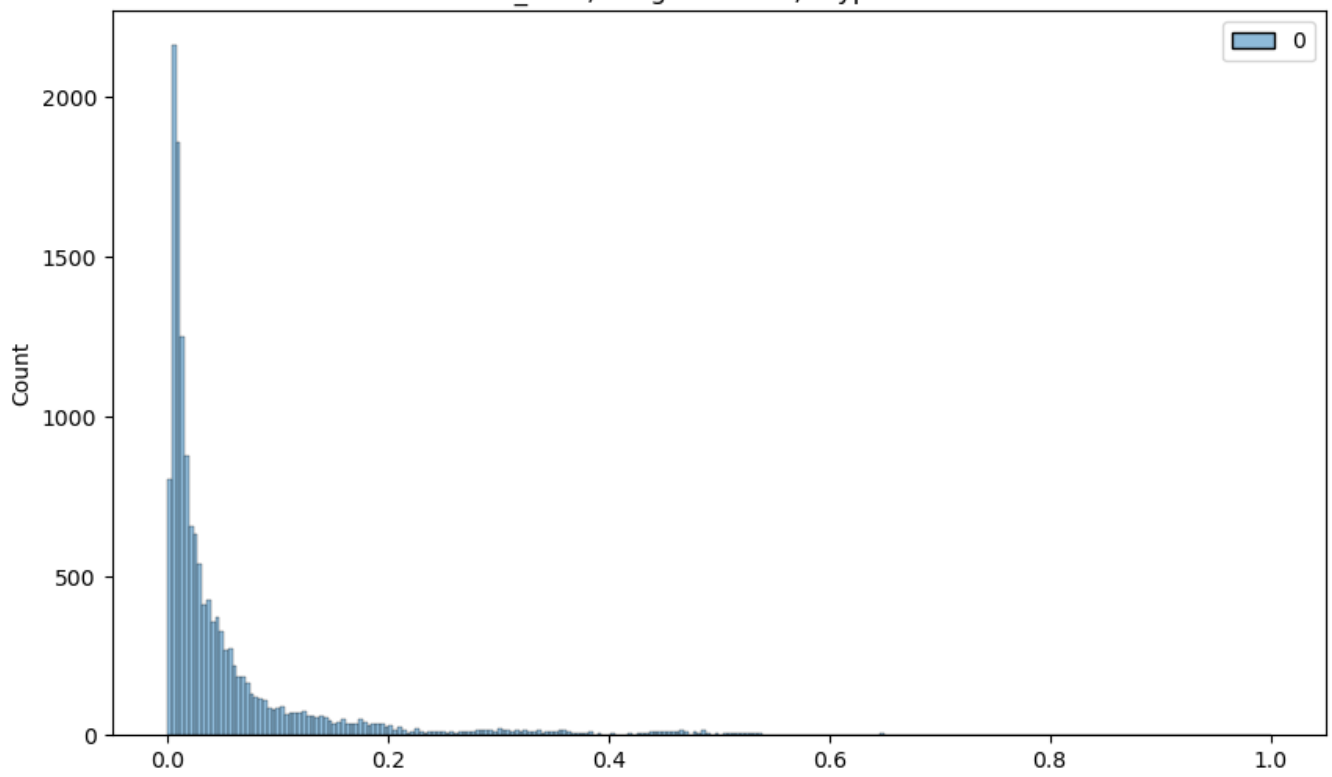
In [170…

```
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()
```

Out[170]:

```
[]
```

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



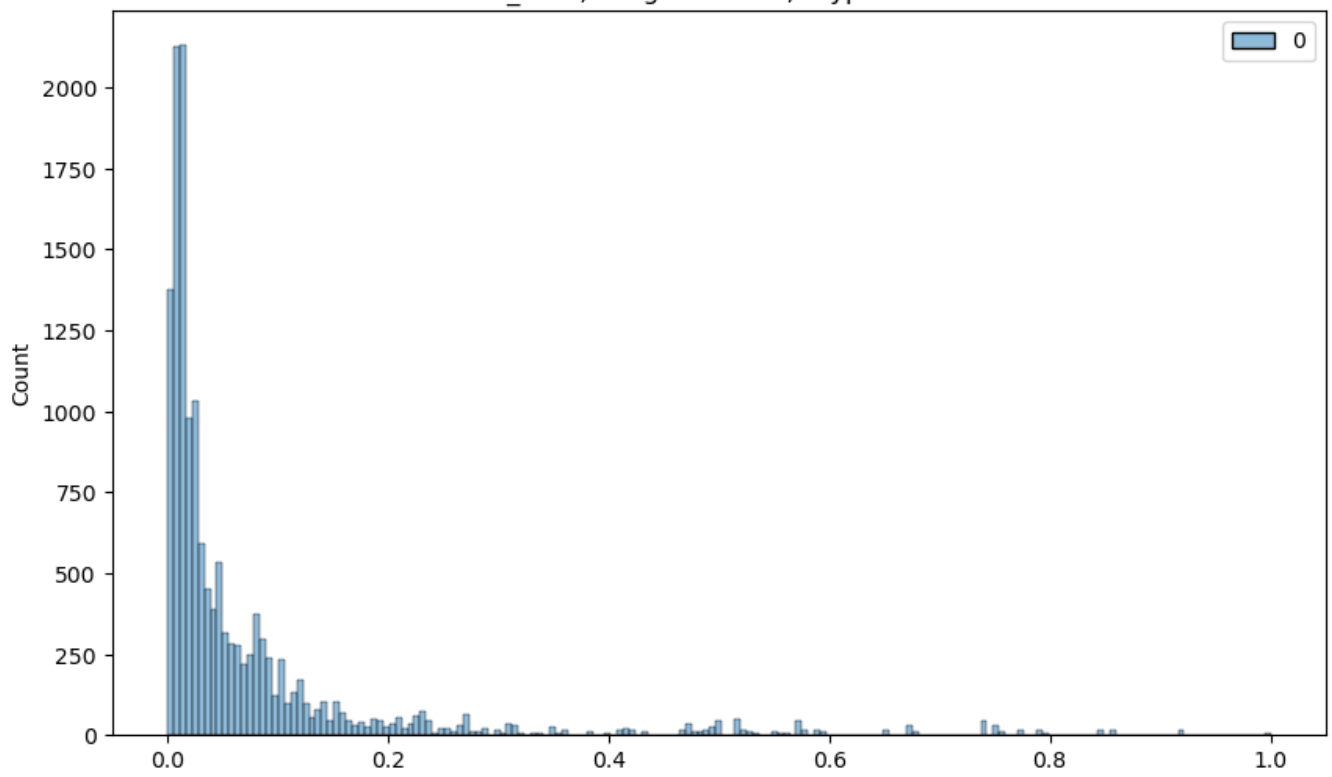
In [171…

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

Out[171]:

```
[]
```

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



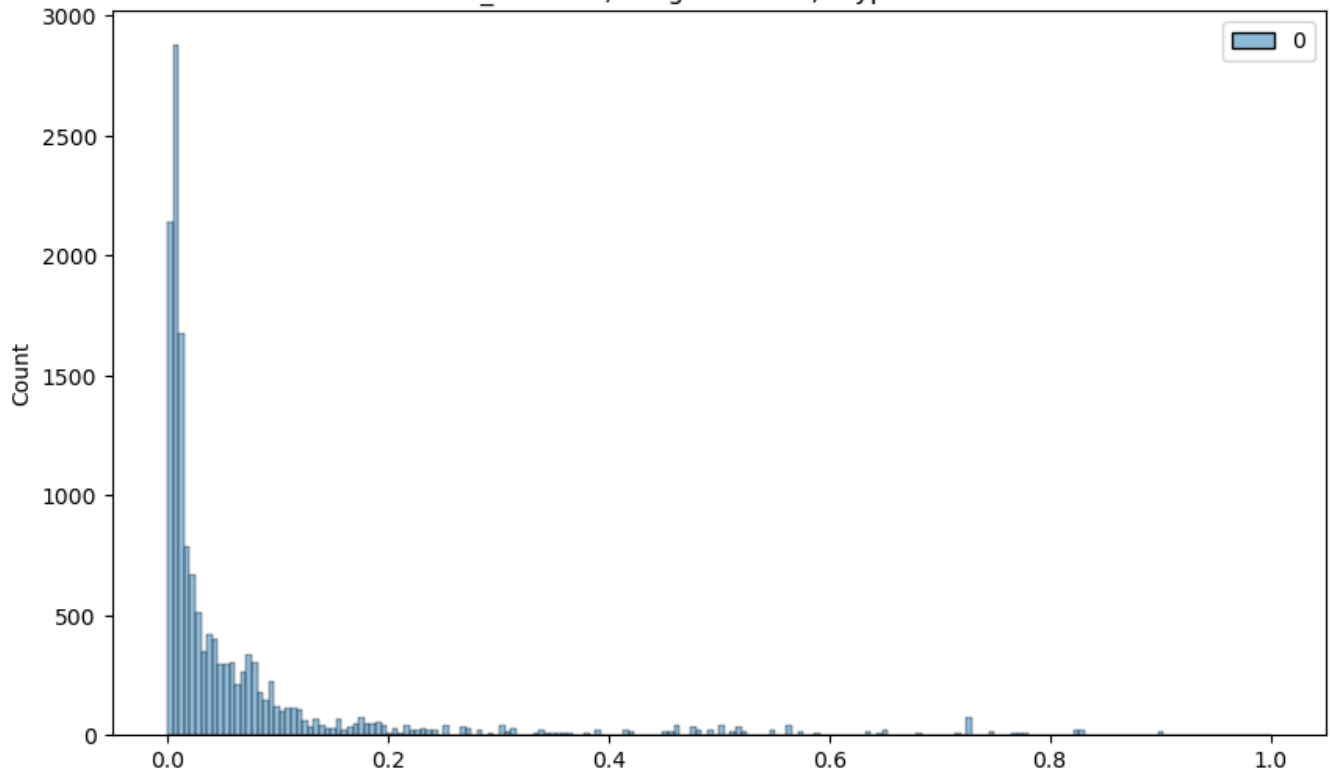
In [172…

```
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()
```

Out[172]:

```
[]
```

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



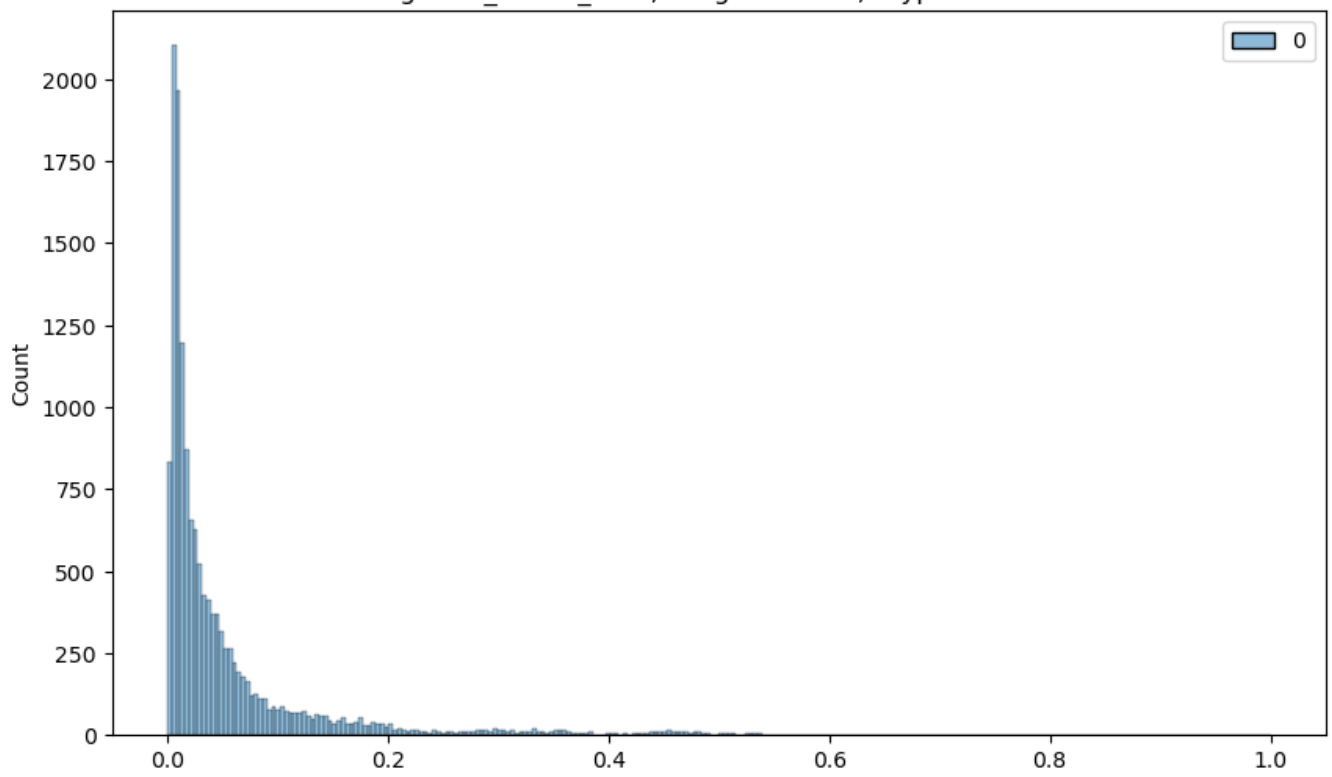
In [173…

```
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()
```

Out[173]:

```
[]
```

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



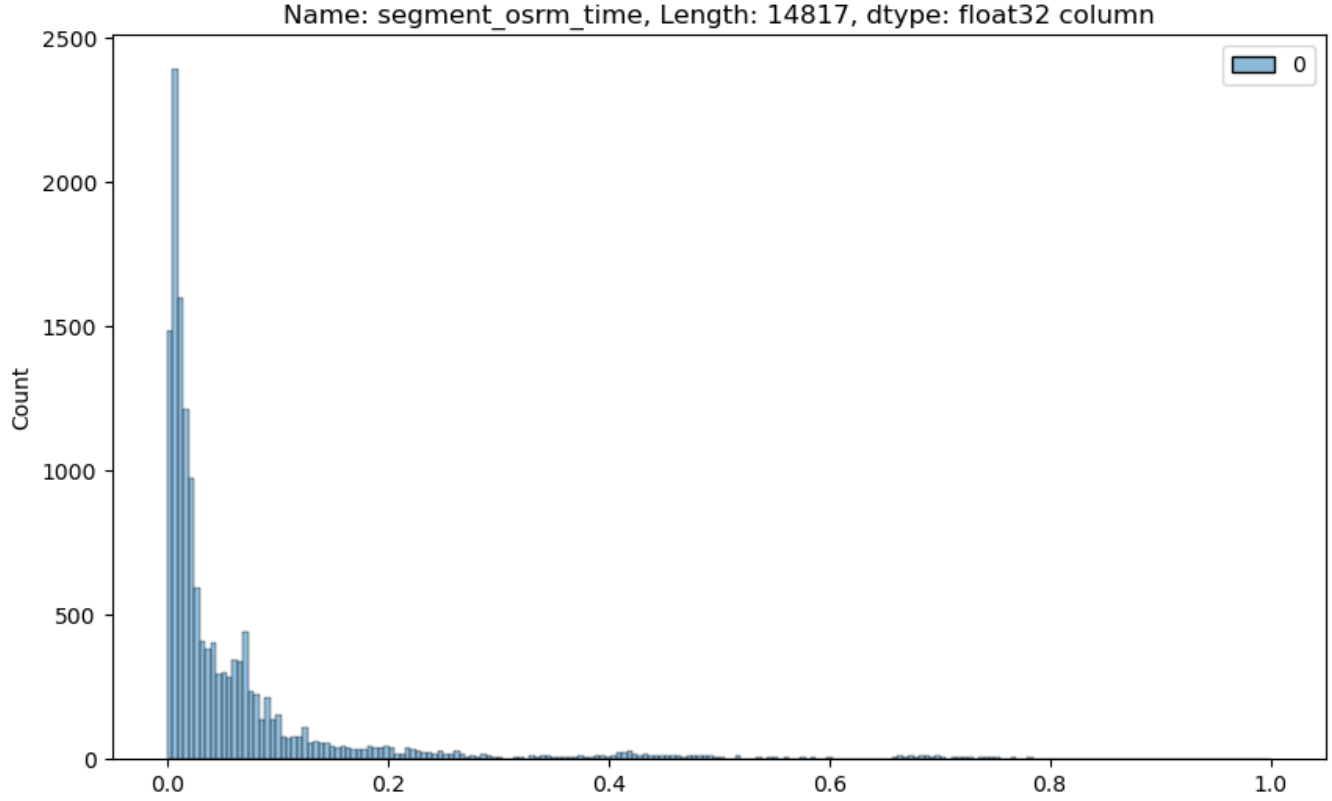
In [174…

```
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()
```

Out[174]:

```
[]
```

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



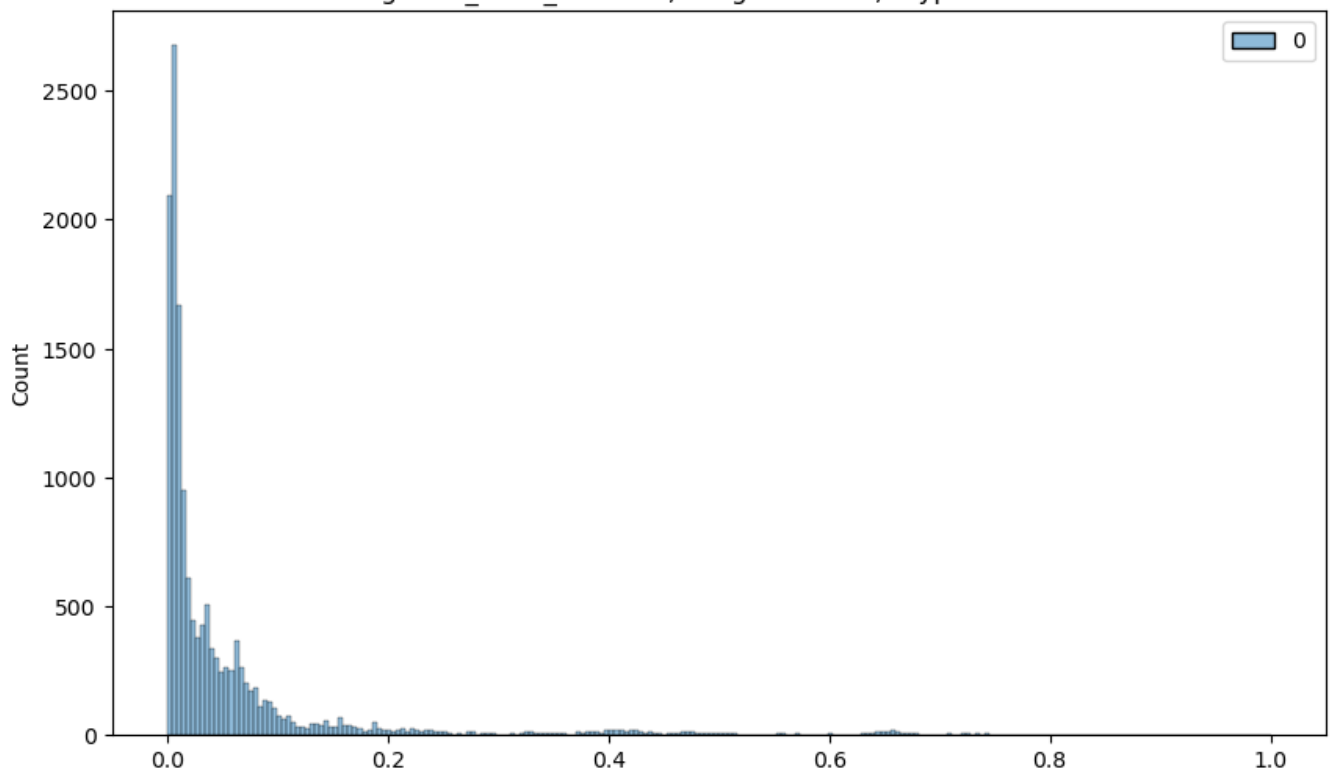
In [175…

```
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()
```

Out[175]:

```
[]
```

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



Column Standardization

In [176…

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

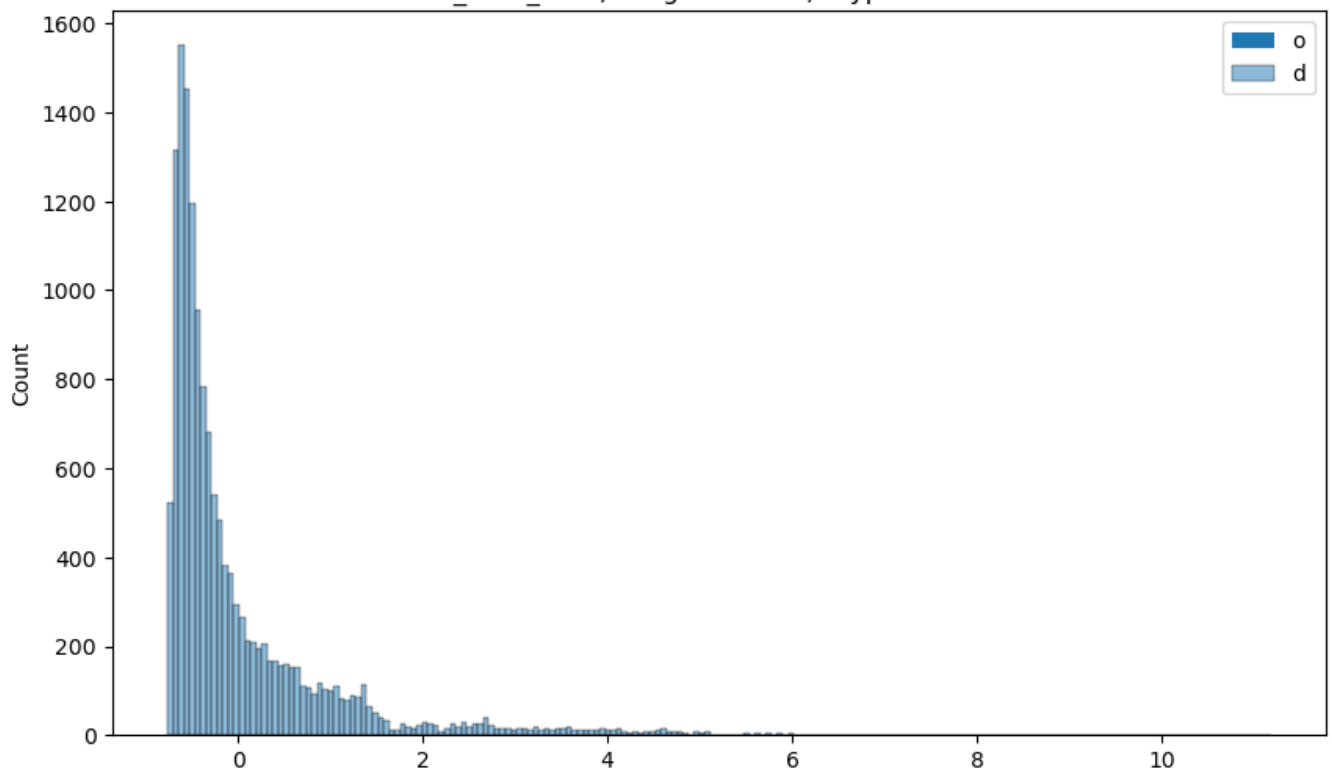
In [177…

```
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()
```

Out[177]:

```
[]
```

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



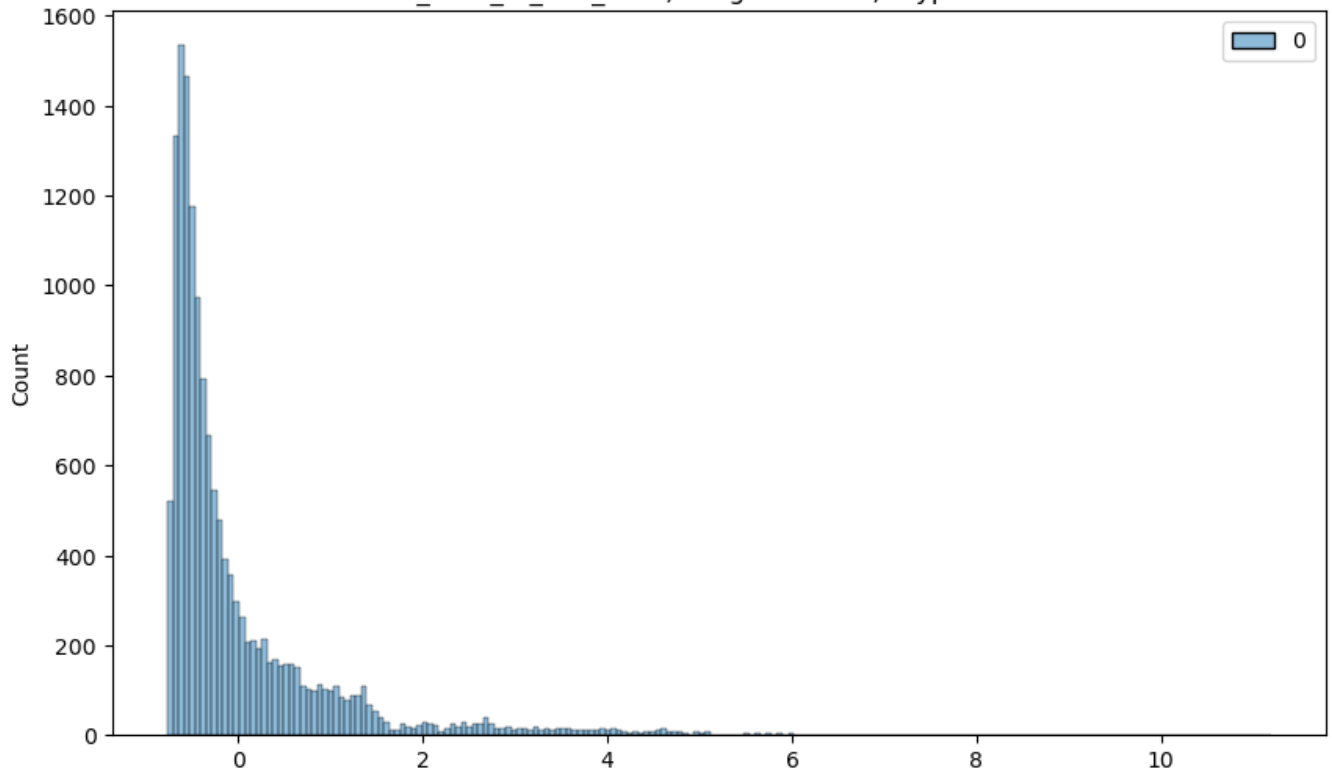
In [178…

```
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()
```

Out[178]:

```
[]
```

```
Standardized 0       2259.0
             1        180.0
             2       3933.0
             3        100.0
             4        717.0
                       ...
         14812        257.0
         14813         60.0
         14814        421.0
         14815        347.0
         14816        353.0
Name: start_scan_to_end_scan, Length: 14817, dtype: float32 column
```



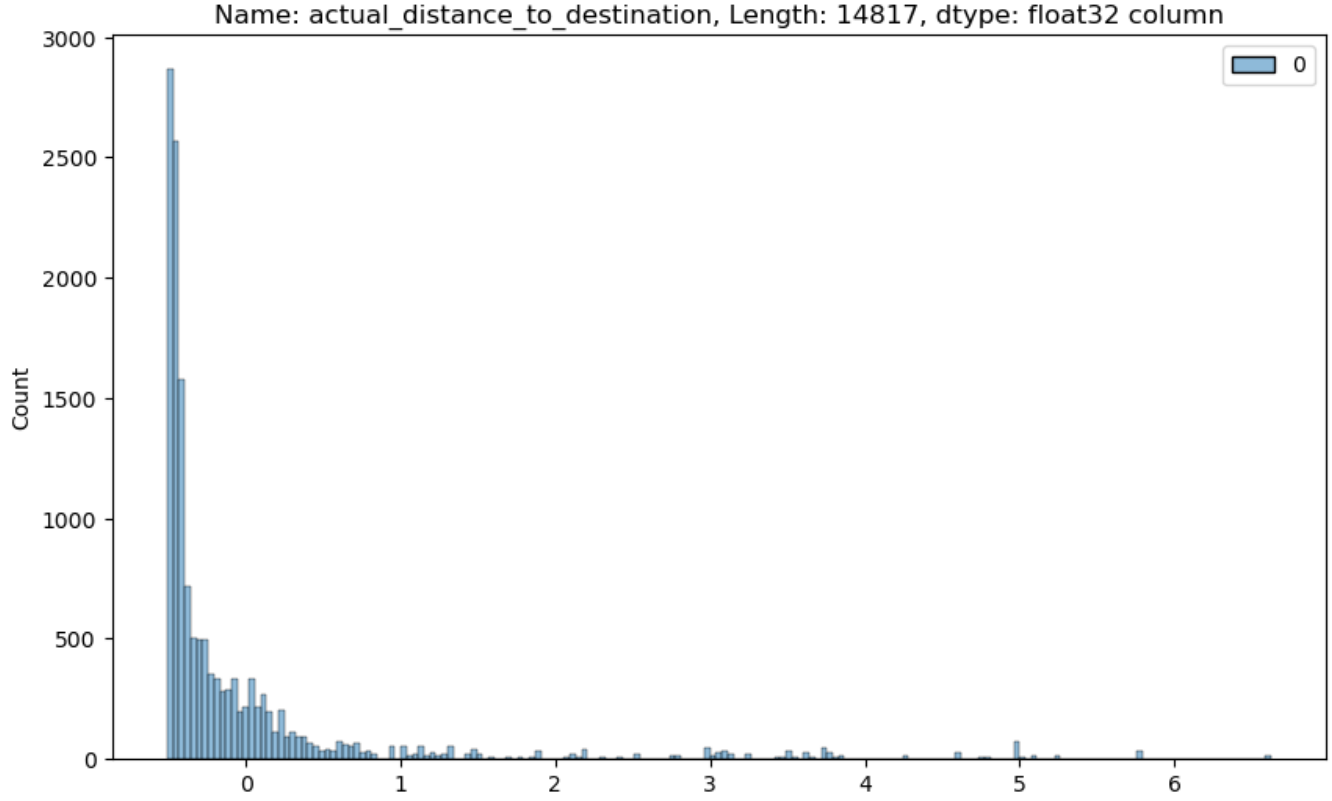
In [179…

```
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()
```

Out[179]:

```
[]
```

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



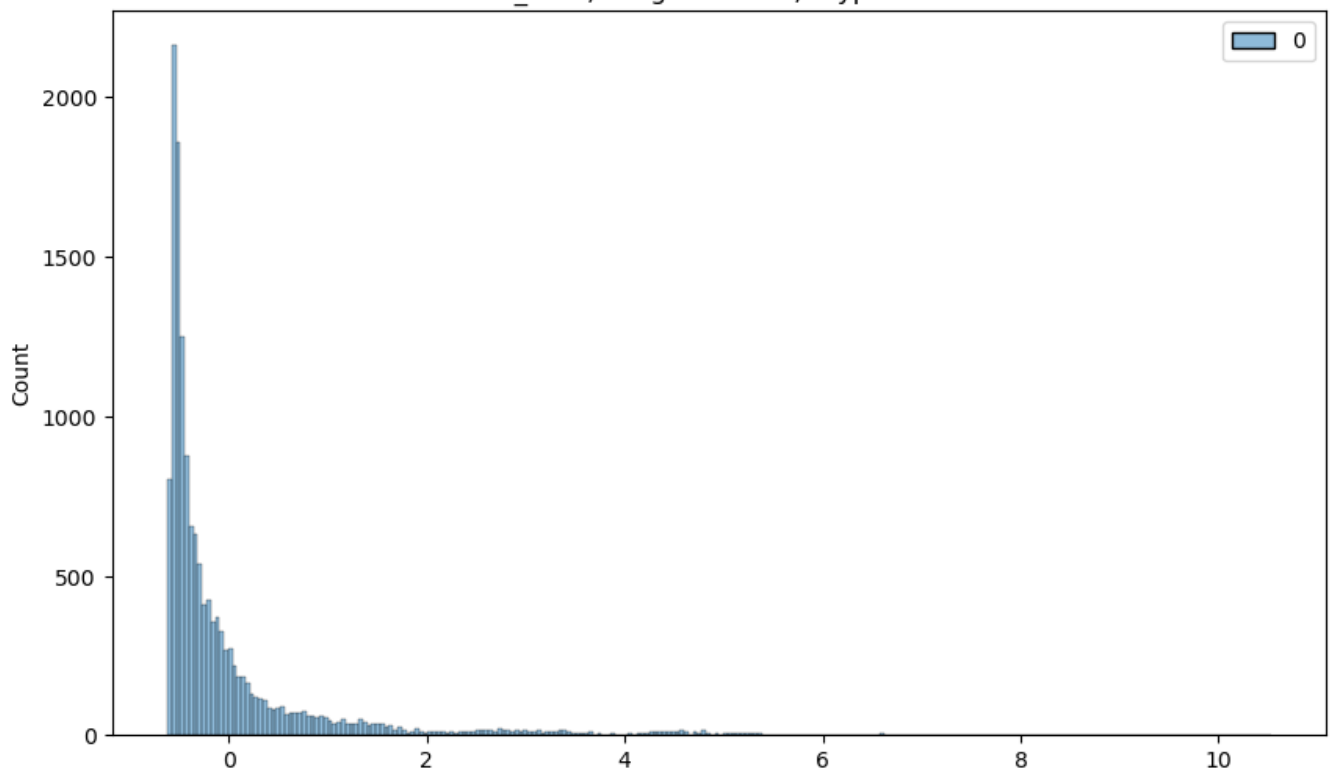
In [180…

```
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()
```

Out[180]:

```
[]
```

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



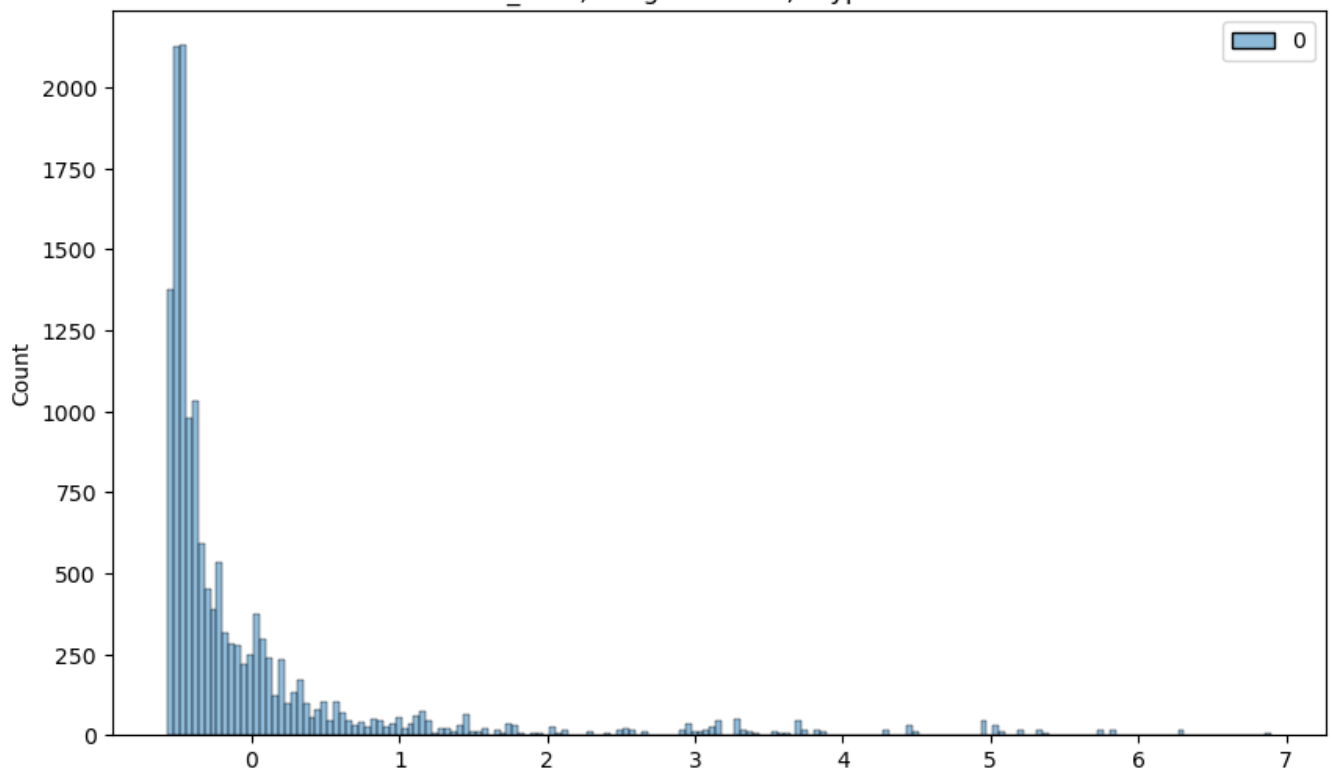
In [181…

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

Out[181]:

```
[]
```

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



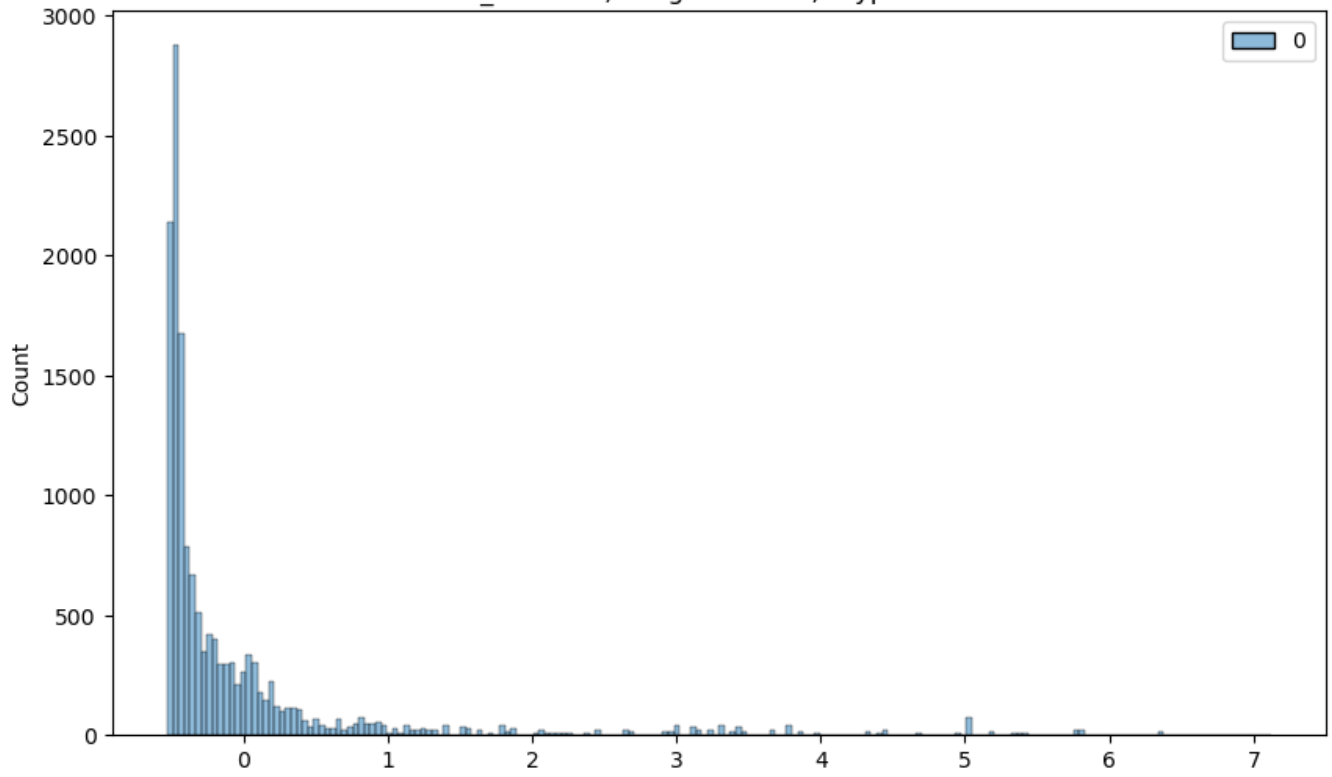
In [182…

```
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()
```

Out[182]:

```
[]
```

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



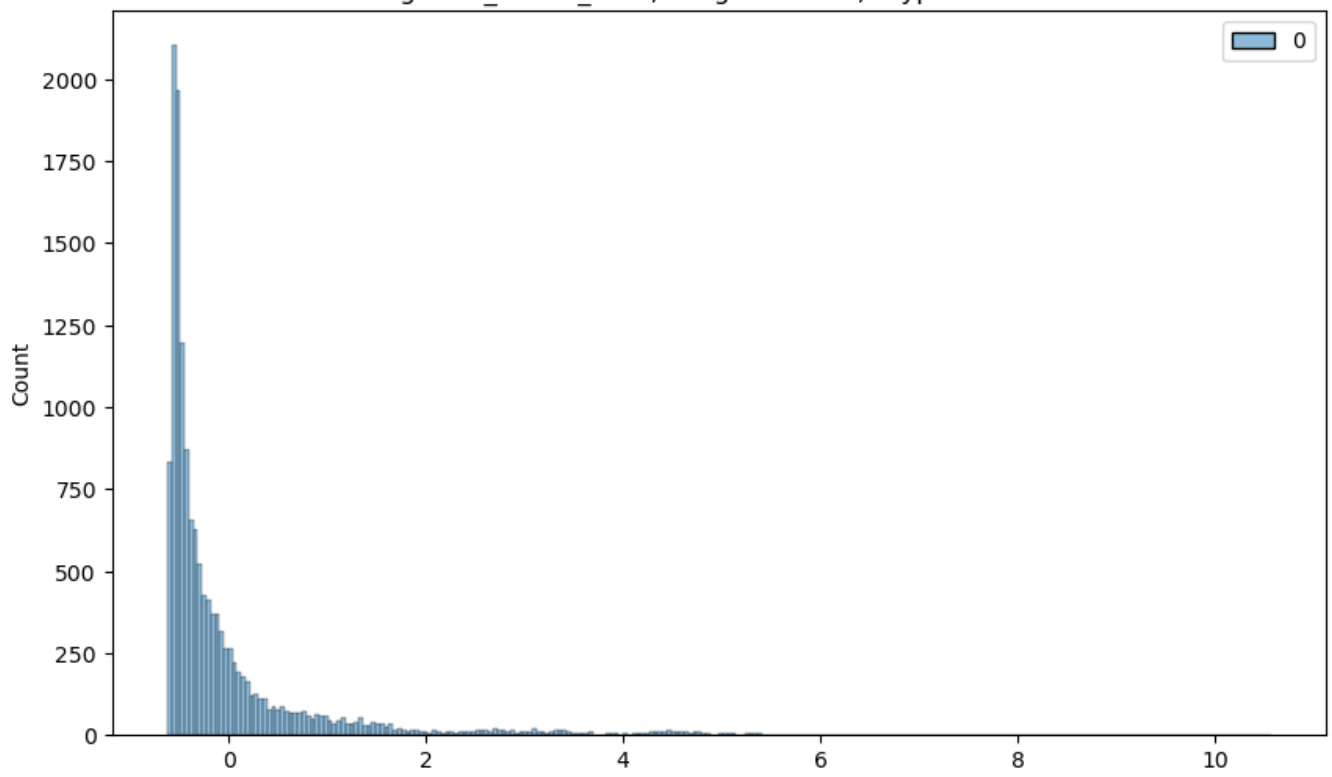
In [183…

```
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()
```

Out[183]:

```
[]
```

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



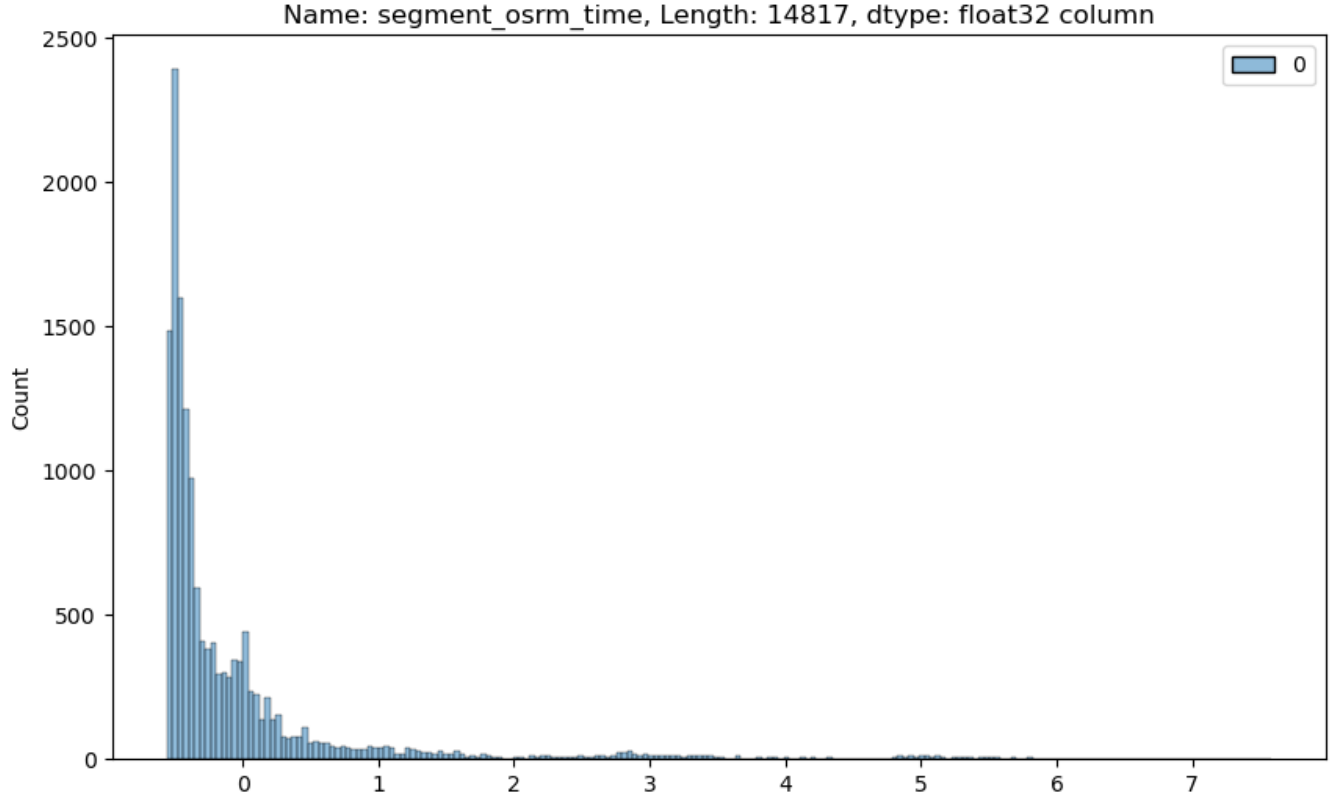
In [184…

```
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()
```

Out[184]:

```
[]
```

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



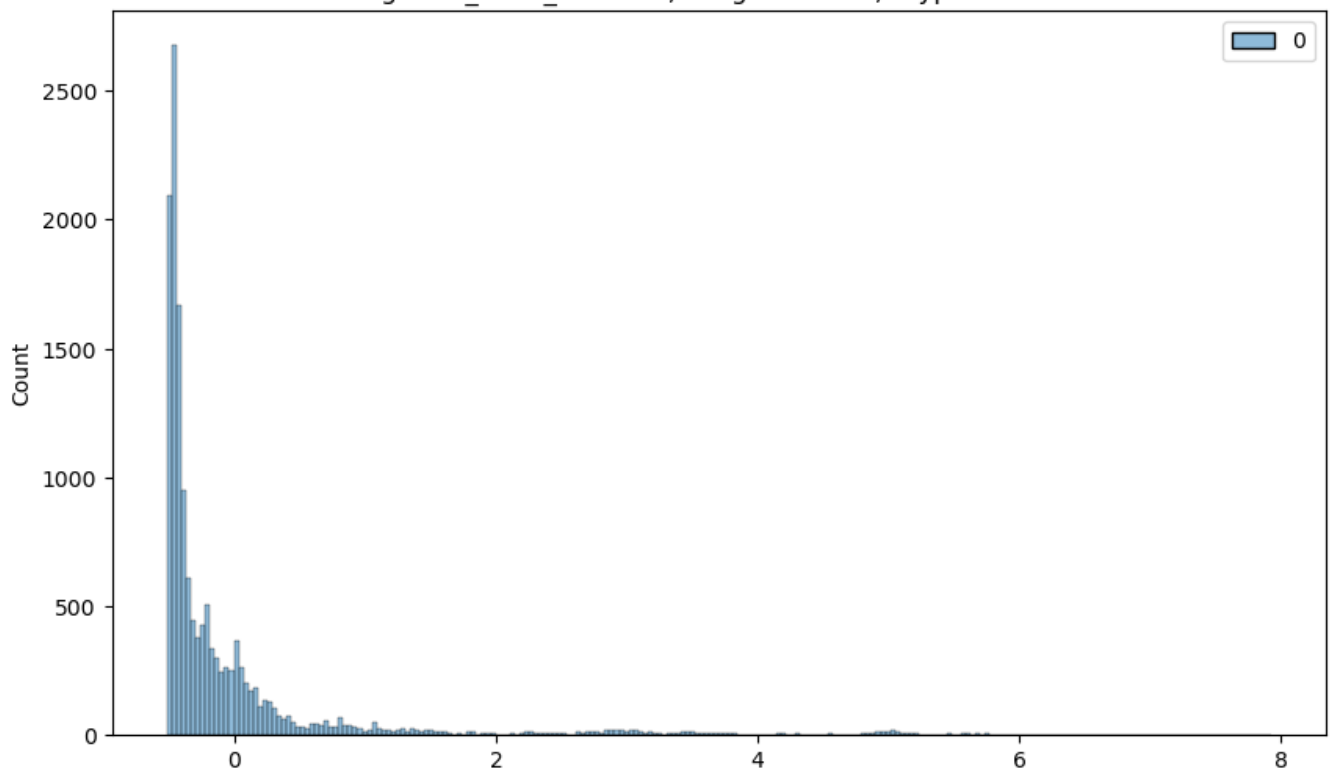
In [185…

```
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()
```

Out[185]:

```
[]
```

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



Business Insights

1. The dataset spans from '2018-09-12 00:00:16' to '2018-10-08 03:00:24'.
2. There are approximately 14,817 unique trip IDs, 1,508 unique source centers, 1,481 unique destination centers, 690 unique source cities, and 806 unique destination cities.
3. Notably, the data is more oriented towards testing than training.
4. The prevalent route type is Carting.
5. Fourteen unique location IDs are absent in the dataset.
6. Trip counts show an upward trend post-noon, peak at 10 P.M., and then decline.
7. The highest number of trips occurred in the 38th week.
8. A concentration of orders tends to occur in the middle of the month.
9. Primary orders originate from states such as Maharashtra, Karnataka, Haryana, Tamil Nadu, and Telangana.
10. The maximum number of trips originates from Mumbai, followed by Gurgaon Delhi, Bengaluru, and Bhiwandi, indicating a strong seller presence in these cities.

11. Most trips conclude in Maharashtra, followed by Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh, pointing to a significantly high order volume in these states.

12. The majority of trips conclude in Mumbai, followed by Bengaluru, Gurgaon, Delhi, and Chennai, indicating substantial order activity in these cities.

13. Regarding destination cities, the highest order frequency is observed in cities like Bengaluru, Mumbai, Gurgaon, Bangalore, and Delhi.

14. The features 'start_scan_to_end_scan' and 'od_total_time' (created feature) exhibit statistical similarity.

15. Conversely, features 'actual_time' and 'osrm_time' are statistically different.

16. Features start_scan_to_end_scan and segment_actual_time are statistically similar.

17. On the other hand, 'osrm_distance' and 'segment_osrm_distance' show statistical differences.

18. Both the osrm_time & segment_osrm_time are not statistically same.

Recommendations:

1. There is a need for enhancements in the OSRM trip planning system to address discrepancies that may affect transporters when the routing engine is configured for optimal outcomes.

2. Observing a variance between 'osrm_time' and 'actual_time,' it is essential for the team to minimize this difference. This reduction is crucial for more accurate delivery time predictions, ensuring greater convenience for customers in anticipating delivery windows.

3. Discrepancies between 'osrm_distance' and the actual distance covered raise concerns. Possible explanations include deviations from predefined routes by delivery personnel or inaccuracies in the OSRM device's prediction of routes based on factors like distance and traffic. Investigation and corrective actions are warranted.

4. A significant portion of orders originates from or is delivered to states such as Maharashtra, Karnataka, Haryana, and Tamil Nadu. To enhance service in these regions, existing corridors can be optimized further.

5. For a deeper understanding of why major orders come from states like Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh, customer profiling is recommended. This analysis aims to improve the overall buying and delivery experience for customers in these states.

6. Considering state-specific factors such as heavy traffic and challenging terrain conditions, this information serves as a valuable indicator for planning and catering to demand, particularly during peak festival seasons.

* End of Project ***

In [ ]: