Delhivery - Feature Engineering

Introduction:

Delhivery, India's leading and rapidly growing integrated player, has set its sights on creating the commerce operating system. They achieve this by utilizing world-class infrastructure, ensuring the highest quality in logistics operations, and harnessing cutting-edge engineering and technology capabilities.

Why this case study?

From Delhivery's Perspective:

- Delhivery aims to establish itself as the premier player in the logistics industry. This case study is of paramount importance as it aligns with the company's core objectives and operational excellence.
- It provides a practical framework for understanding and processing data, which is integral to their operations. By leveraging data engineering pipelines and data analysis techniques, Delhivery can achieve several critical goals.
- First, it allows them to ensure data integrity and quality by addressing missing values and structuring the dataset appropriately.
- Second, it enables the extraction of valuable features from raw data, which can be utilized for building accurate forecasting models.
- Moreover, it facilitates the identification of patterns, insights, and actionable recommendations crucial for optimizing their logistics operations.
- By conducting hypothesis testing and outlier detection, Delhivery can refine their processes

and further enhance the quality of service they provide. From Learners' Perspective:

- Learners will gain hands-on experience in data preprocessing and cleaning, which is often the most time-consuming aspect of data analysis.
- Feature engineering is a critical step in building machine learning models. In this case study, learners will understand how to extract meaningful features from raw data, including datetime manipulation and column splitting.
- The case study introduces learners to the concept of grouping data based on specific keys and then aggregating it. This is a key aspect of data analysis, especially when dealing with time-series data or data with a hierarchical structure.
- Learners will perform hypothesis testing, to validate assumptions and draw insights from data.
- The case study goes beyond data analysis by focusing on deriving actionable insights for a business. Learners will understand how data analysis can drive informed decision-making and recommendations.

In [ ]:

Plan Summery

We will derive three type of data set from original dataset

Original dataset ---> df : Conatins original non aggregated and non grouped data. Actual number of trips

Aggregated at segment ----> seg_agg_data : Contains grouped data based on 'segment_key' i.e on each individual trip and aggreagated.

Aggregated at trip id ----> trip_agg_data : Contains grouped data based on 'trip_id' i.e on complete trip and aggreagated.

We will use original dataset df to get info like number of corridor, pie chart on type of route, pie on data type , diff distribution of trip on hourly and monthly.

We will use Seg_agg_data to info on number cites have max min source city and destination city, correlation between diff features and pairplot.

We will use trip_agg_data to encode, removing outliers for aggreagted columns, standardization and normalization.

Finally, for Hypothesis testing we prefer to use seg_agg_data becuase we want to see relation between each feature at individual segment trip which give us in depth detail analysis.

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

import copy
```

In [2]:
```
```

In [3]:
```
```

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

In [4]:
```
```

Basic data cleaning and exploration:

1. Handle missing values in the data.
2. Converting time columns into pandas datetime.
3. Analyze structure & characteristics of the dataset.

In [5]:
```
#Droping unkoen fields of columns from dataset
```

In [6]:

Out[6]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sourc
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

In [7]:

```
<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
```

In [8]:
```
#changing the data type of date time columns
df['trip_creation_time']=pd.to_datetime(df['trip_creation_time'])
df['od_start_time']=pd.to_datetime(df['od_start_time'])
```

In [9]:
```
#checking columns with only type data in it
for j in df.columns:
    i=df[j].nunique()
```

```
unique entry for data: 2
unique entry for trip_creation_time: 14817
unique entry for route_schedule_uuid: 1504
unique entry for route_type: 2
unique entry for trip_uuid: 14817
unique entry for source_center: 1508
unique entry for source_name: 1498
unique entry for destination_center: 1481
unique entry for destination_name: 1468
unique entry for od_start_time: 26369
unique entry for od_end_time: 26369
unique entry for start_scan_to_end_scan: 1915
unique entry for actual_distance_to_destination: 144515
unique entry for actual_time: 3182
unique entry for osrm_time: 1531
unique entry for osrm_distance: 138046
unique entry for segment_actual_time: 747
unique entry for segment_osrm_time: 214
unique entry for segment_osrm_distance: 113799
```

In [10]:
```
#data and route_type has two type of data and we will change the data type of

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

In [11]:
```
float_col = ['start_scan_to_end_scan','actual_distance_to_destination','actual
```

In [12]:
```
for i in float_col:
```

```
7898.0
1927.4477046975032
4532.0
1686.0
2326.1991000000003
3051.0
1611.0
2191.4037000000003
```

In [13]:

In [14]:
```
for i in float_col:
```

In [15]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                           Non-Null Count   Dtype
---  ------                           --------------   -----
 0   data                             144867 non-null  category
 1   trip_creation_time               144867 non-null  datetime64[ns]
 2   route_schedule_uuid              144867 non-null  object
 3   route_type                       144867 non-null  category
 4   trip_uuid                        144867 non-null  object
 5   source_center                    144867 non-null  object
 6   source_name                      144574 non-null  object
 7   destination_center               144867 non-null  object
 8   destination_name                 144606 non-null  object
 9   od_start_time                    144867 non-null  datetime64[ns]
 10  od_end_time                      144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan           144867 non-null  float32
 12  actual_distance_to_destination   144867 non-null  float32
 13  actual_time                      144867 non-null  float32
 14  osrm_time                        144867 non-null  float32
 15  osrm_distance                    144867 non-null  float32
 16  segment_actual_time              144867 non-null  float32
 17  segment_osrm_time                144867 non-null  float32
 18  segment_osrm_distance            144867 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(8), object(6)
memory usage: 14.6+ MB
```

In [16]:

In [17]:
```
#Time period of data given
date_max = df['trip_creation_time'].dt.date.max()
date_min = df['trip_creation_time'].dt.date.min()
```

Out[17]: (datetime.date(2018, 9, 12), datetime.date(2018, 10, 3))

In [18]:
```
#Checking null values in the dataFrame
```

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]:
```
#Unique source cener name where there is no source name given
missing_source_name_center = df[df['source_name'].isnull()]['source_center'].u
```

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

In [21]:
```
for i in missing_source_name_center:
    unique_source_name = df.loc[df['source_center'] == i, 'source_name'].uniqu
    if pd.isna(unique_source_name):
        print("Source Center :", i, "-" * 10, "Source Name :", 'Not Found')
    else :
```

```
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]:
```
for i in missing_source_name_center:
    unique_destination_name = df.loc[df['destination_center'] == i, 'destinati
    if  (pd.isna(unique_source_name)) or (unique_source_name.size == 0):
        print("Destination Center :", i, "-" * 10, "Destination Name :", 'Not
    else :
```

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

In [23]:
```
missing_destination_name_center = df[df['destination_name'].isnull()]['destina
```

Out[23]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [24]:
```
# All source name center in missing data are there in destination name center
```

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

In [25]:
```
c= 1
for i in missing_destination_name_center:
    df.loc[df['destination_center'] == i,'destination_name'] = df.loc[df['dest
    c+=1
```

In [26]:
```
d={}
for i in missing_source_name_center:
    d[i] = df.loc[df['destination_center']==i,'destination_name'].unique()
for key, value in d.items():
    if len(value)==0:
        d[key] = [f'Location_{c}']
        c+=1

d2 = {}
for i,j in d.items():
    d2[i]=j[0]
for i,j in d2.items():
```

```
IND342902A1B Locatio_1
IND577116AAA Locatio_2
IND282002AAD Locatio_3
IND465333A1B Locatio_4
IND841301AAC Locatio_5
IND509103AAC Locatio_9
IND126116AAA Locatio_8
IND331022A1B Location_14
IND505326AAB Locatio_6
IND852118A1B Locatio_7
```

In [27]:
```
for i in missing_source_name_center:
```

In [28]:
```
```

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
```

In [29]:

Out[29]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osı
count	144867.000000	144867.000000	144867.000000	144867.000000	14
mean	961.271912	234.050812	416.929504	213.864685	
std	1036.997803	344.979126	598.096069	308.004333	
min	20.000000	9.000046	9.000000	6.000000	
25%	161.000000	23.355875	51.000000	27.000000	
50%	449.000000	66.126572	132.000000	64.000000	
75%	1634.000000	286.708878	513.000000	257.000000	
max	7898.000000	1927.447754	4532.000000	1686.000000	

In [30]:

```
C:\Users\hyt1kor\AppData\Local\Temp/ipykernel_21928/2884002236.py:1: FutureWa
rning: Treating datetime data as categorical rather than numeric in `.describ
e` is deprecated and will be removed in a future version of pandas. Specify `
datetime_is_numeric=True` to silence this warning and adopt the future behavi
or now.
  df.describe(include='all')
```

Out[30]:

	data	trip_creation_time	route_schedule_uuid	route_type	t
count	144867	144867	144867	144867	
unique	2	14817	1504	2	
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-15381121953
freq	104858	101	1812	99660	
first	NaN	2018-09-12 00:00:16.535741	NaN	NaN	
last	NaN	2018-10-03 23:59:42.701692	NaN	NaN	
mean	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	

In [31]:

```
df['Year'] = df['trip_creation_time'].dt.year
df['Month'] = df['trip_creation_time'].dt.month
df['Week'] = df['trip_creation_time'].dt.day_name()
```

In [32]:

Out[32]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sourc
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

In [33]:
```
def city(x):
    lst = x.split('_')
    return lst[0]
```

In [34]:

In [35]:

In [36]:

In [37]:

In [38]:

In [39]:

In [40]:
```
df['day'] = df['trip_creation_time'].dt.day
```

In [41]:
```
df['month'] = df['trip_creation_time'].dt.month
```

In [42]:
```
df['hour'] = df['trip_creation_time'].dt.hour
```

In [43]:

Out[43]:

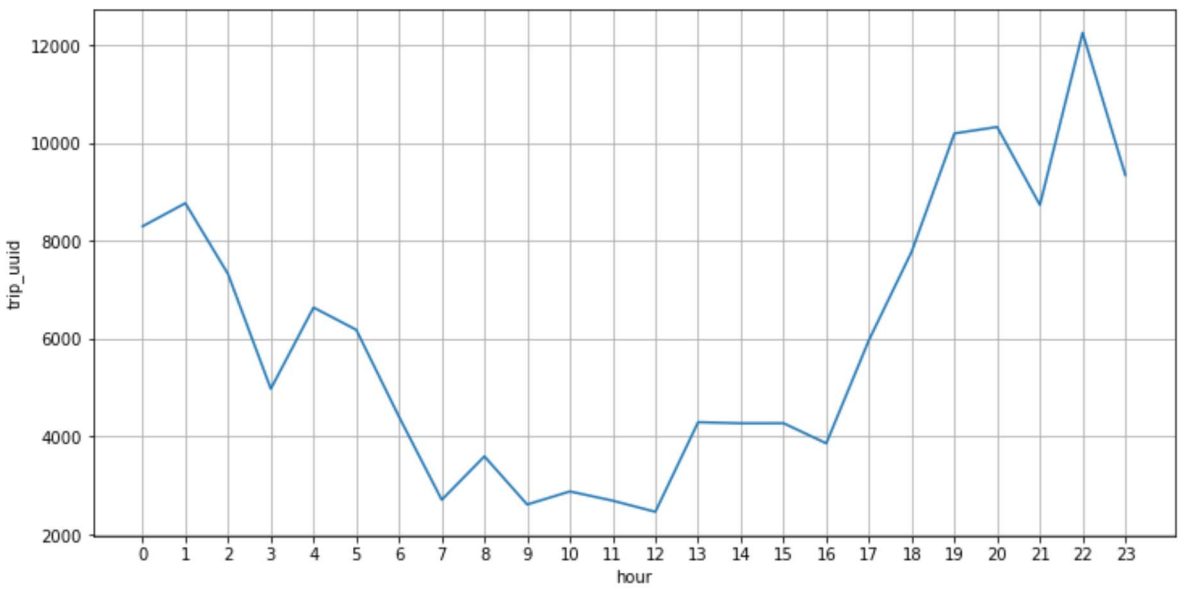
	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sourc
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

5 rows × 29 columns

In [44]:
```
df_hour = df.groupby(by = 'hour')['trip_uuid'].count().to_frame().reset_index(

plt.figure(figsize = (12, 6))
sns.lineplot(data = df_hour,
             x = df_hour['hour'],
             y = df_hour['trip_uuid'],
             markers = '*')
plt.xticks(np.arange(0,24))
plt.grid('both')
```

Out[44]: []



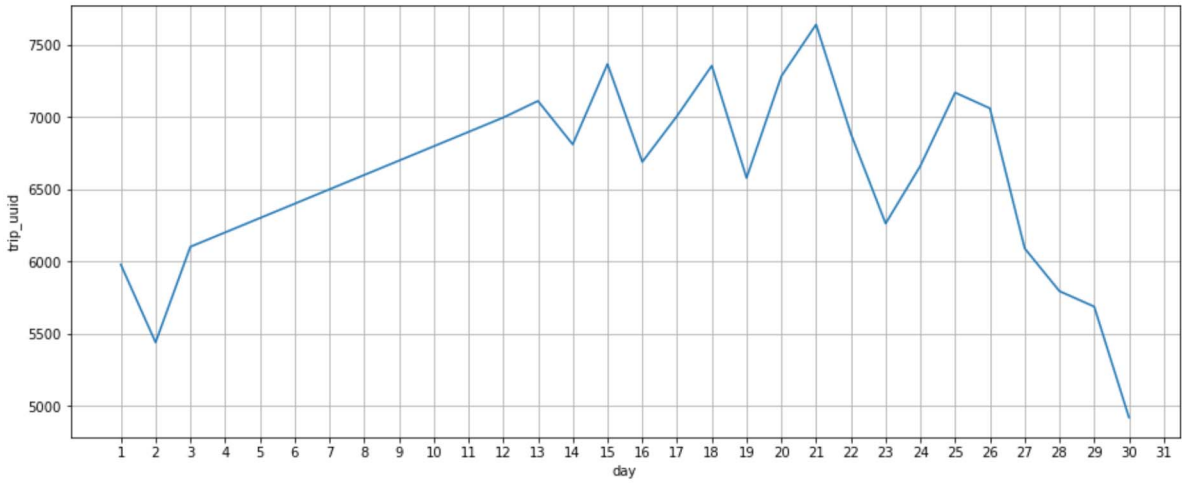
- It can be inferred from the above plot that the number of trips start increasing after the

noon, becomes maximum at 10 P.M and then start decreasing.

In [45]:
```
df_day = df.groupby(by = 'day')['trip_uuid'].count().to_frame().reset_index()

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

Out[45]: []



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

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

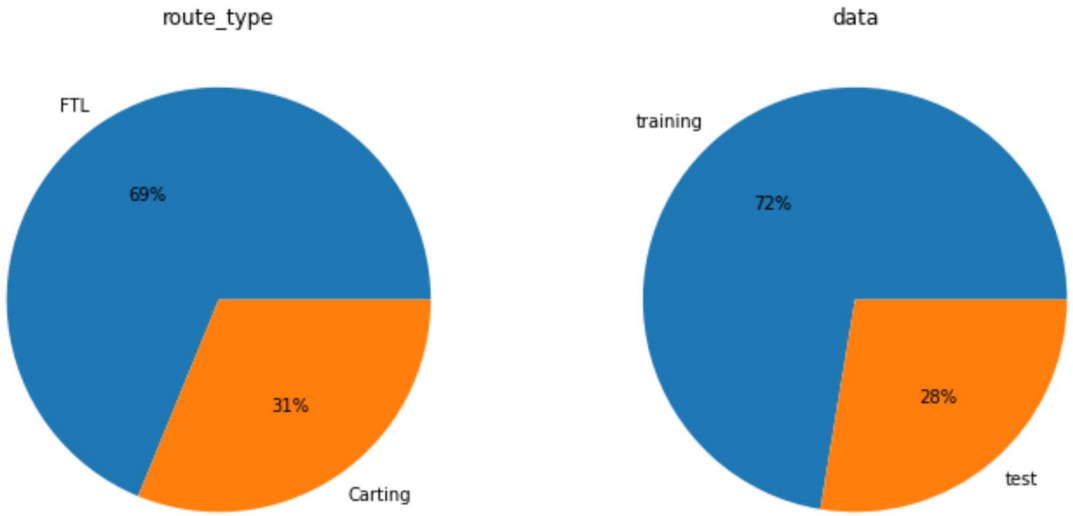
Contribution of route type & data

In [46]:
```
# % of Full truck loading and carting in the whole dataset
x = df['route_type'].value_counts()
y = df['route_type'].value_counts().index

# % of training and testing data in the whole dataset
x1 = df['data'].value_counts()
y1 = df['data'].value_counts().index

plt.figure(figsize=(12,8))
plt.subplot(1,2,1)
plt.pie(x,labels=y, autopct='%.0f%%')
plt.title('route_type')


plt.subplot(1,2,2)
plt.pie(x1,labels=y1, autopct='%.0f%%')
plt.title('data')
plt.show()
```

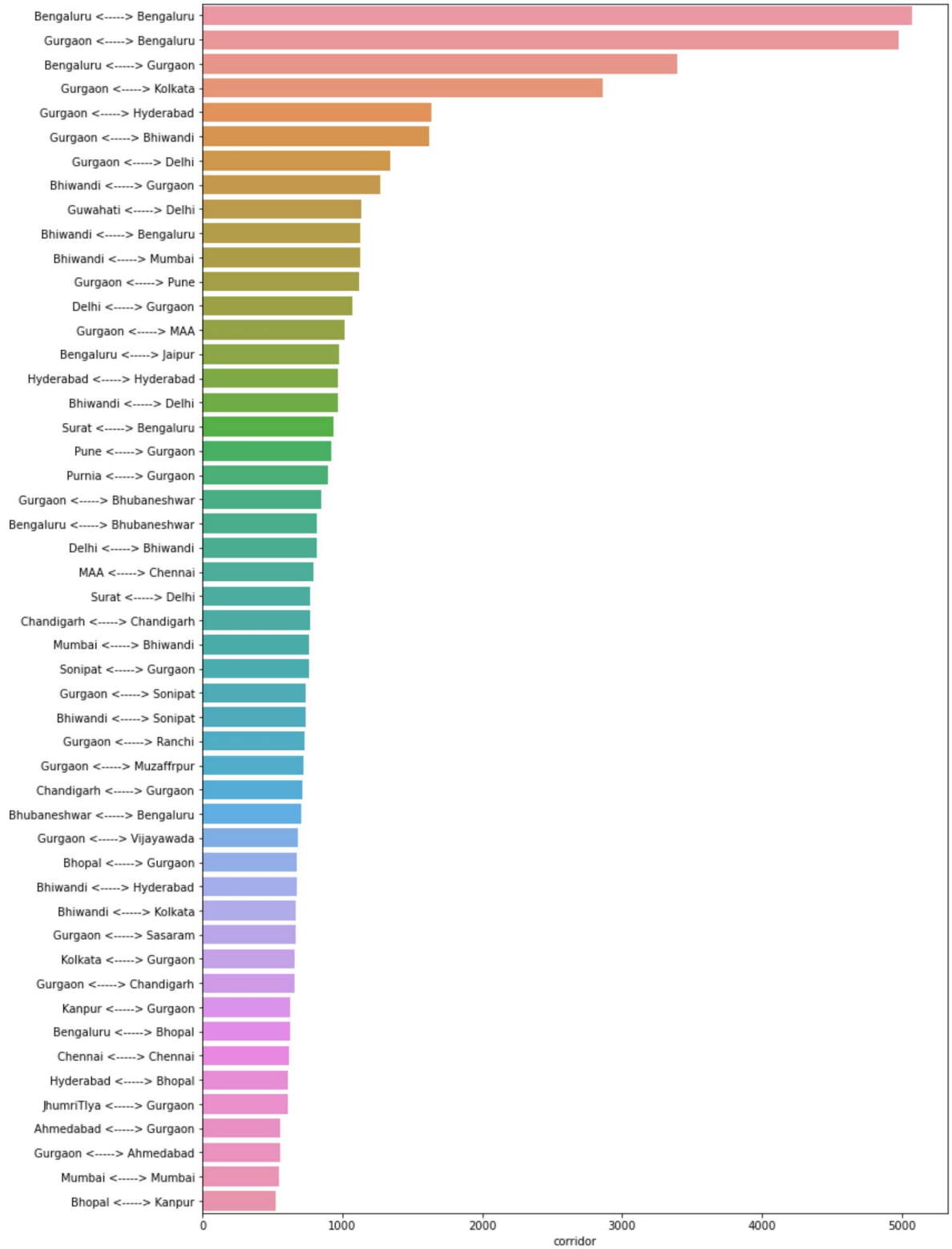


Type *Markdown* and LaTeX: α^2

Segment wise trip counts (Top 50 Corridor)

In [47]:
```
# top 50 corridor in term of number of trip made
x2=df['corridor'].value_counts().sort_values(ascending=False).nlargest(50)
y2=x2.index

plt.figure(figsize=(12,20))
sns.barplot(y=y2, x=x2, data=df)
```

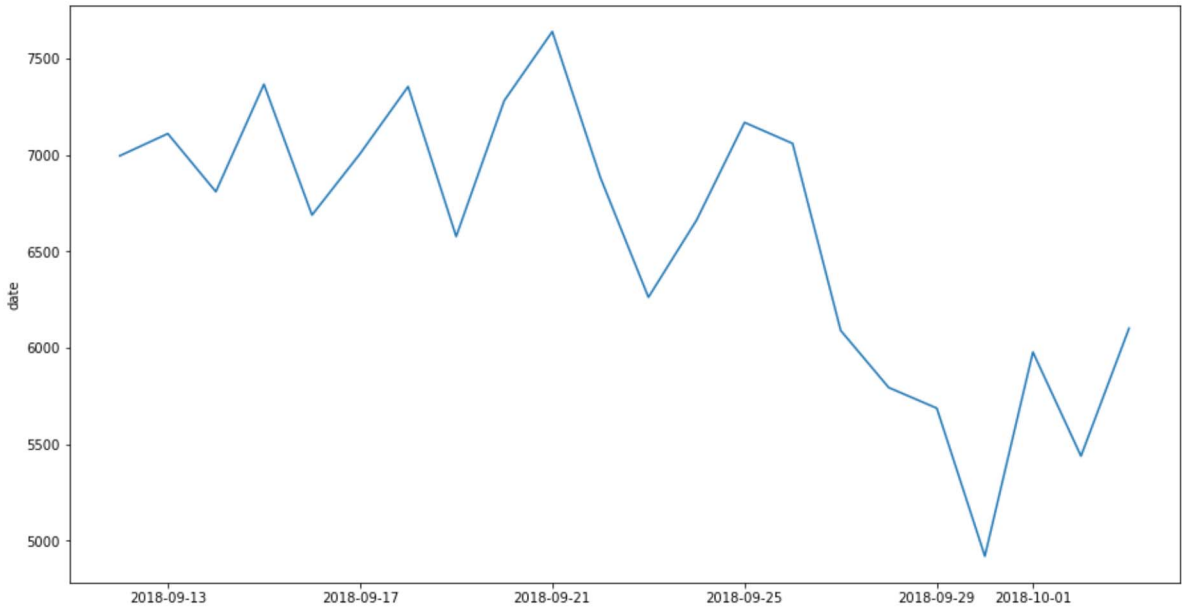


In [ ]:

Trip made between this given duration

In [48]:
```
x3 = df['date'].value_counts().sort_values()
y3=x3.index

plt.figure(figsize=(15,8))
sns.lineplot( x=y3, y=x3)
```



Type *Markdown* and LaTeX: α^2

Merging the rows

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

1. Grouping by segment

- Create a unique identifier for different segments of a trip based on the combination of the trip_uuid, source_center, and destination_center and name it as segment_key.
- You can use inbuilt functions like groupby and aggregations like cumsum() to merge the rows in columns segment_actual_time, segment_osrm_distance, segment_osrm_time based on the segment_key.
- This way you'll get new columns named segment_actual_time_sum, segment_osrm_distance_sum, segment_osrm_time_sum.

2. Aggregating at segment level

- Create a dictionary named create_segment_dict, that defines how to aggregate and select values. i. You can keep the first and last values for some numeric/categorical fields if aggregating them won't make sense.
- Further group the data by segment_key because you want to perform aggregation operations for different segments of each trip based on the segment_key value.
- The aggregation functions specified in the create_segment_dict are applied to each group of rows with the same segment_key.
- Sort the resulting DataFrame segment, by two criteria: i.First, it sorts by segment_key to ensure that segments are ordered consistently. ii. Second, it sorts by od_end_time in ascending order, ensuring that segments within the same trip are ordered by their end

In [49]:
```
segment_key = ['trip_uuid','source_center','destination_center']
segment_col = ['segment_actual_time','segment_osrm_time','segment_osrm_distanc

df['segment_key'] = df['trip_uuid']+'_'+df['source_center']+'_'+df['destinatio

for col in segment_col:
    df[col+'_sum'] = df.groupby('segment_key')[col].cumsum()
```

Out[49]:

	segment_key	segment_actual_time_sum	segmer
0	trip-153741093647649320_IND388121AAA_IND388620AAB	14.0	
1	trip-153741093647649320_IND388121AAA_IND388620AAB	24.0	
2	trip-153741093647649320_IND388121AAA_IND388620AAB	40.0	
3	trip-153741093647649320_IND388121AAA_IND388620AAB	61.0	
4	trip-153741093647649320_IND388121AAA_IND388620AAB	67.0	
...	...	...	
144862	trip-153746066843555182_IND131028AAB_IND000000ACB	92.0	
144863	trip-153746066843555182_IND131028AAB_IND000000ACB	118.0	
144864	trip-153746066843555182_IND131028AAB_IND000000ACB	138.0	
144865	trip-153746066843555182_IND131028AAB_IND000000ACB	155.0	
144866	trip-153746066843555182_IND131028AAB_IND000000ACB	423.0	

144867 rows × 4 columns

In [50]:
```
segment_dict = {
    'trip_uuid' : 'first',
    'data': 'first',
    'route_type': 'first',
    'trip_creation_time': 'first',
    'source_name': 'first',
    'destination_name': 'last',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'first',
    'actual_distance_to_destination': 'last',
    'actual_time': 'last',
    'osrm_time': 'last',
    'osrm_distance': 'last',
    'segment_actual_time' : 'sum',
    'segment_osrm_time' : 'sum',
    'segment_osrm_distance' : 'sum',
    'segment_actual_time_sum': 'last',
    'segment_osrm_time_sum': 'last',
    'segment_osrm_distance_sum': 'last'
}

seg_agg_data = df.groupby('segment_key').agg(segment_dict).reset_index()
seg_agg_data = seg_agg_data.sort_values(by=['segment_key','od_end_time'])
```

Out[50]:

	segment_key	trip_uuid	data	route_
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Ca
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Ca
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	

Type *Markdown* and LaTeX: α^2

Feature Engineering:

Extract features from the below fields:

- Calculate time taken between od_start_time and od_end_time and keep it as a feature named od_time_diff_hour. Drop the original columns, if required.
- Destination Name: Split and extract features out of destination. City-place-code (State)
- Source Name: Split and extract features out of destination. City-place-code (State)
- Trip_creation_time: Extract features like month, year, day, etc.

In [51]:

In [52]:

In [53]:
```
# using regex pattern to seperate the city,place,state
def extract_info(name):
    pattern = r'^(?P<city>[^\s_]+)_?(?P<place>[^\(\)]*)\s?\((?P<state>[A-Za-z\
    match = re.match(pattern, name)
    if match:
        city = match.group('city').strip()
        place = match.group('place').strip() if match.group('place') else city
        state = match.group('state').strip()
        return city, place, state
    else:
```

In [54]:
```
seg_agg_data[['source_city', 'source_place', 'source_state']] = seg_agg_data['
```

In [55]:
```
seg_agg_data[['destination_city', 'destination_place', 'destination_state']] =
```

In [56]:

Out[56]:

	segment_key	trip_uuid	data	route_
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Ca
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Ca
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	

5 rows × 25 columns

In [57]:
```
seg_agg_data['trip_creation_Year'] = seg_agg_data['trip_creation_time'].dt.yea
seg_agg_data['trip_creation_Month'] = seg_agg_data['trip_creation_time'].dt.mo
seg_agg_data['trip_creation_Day'] = seg_agg_data['trip_creation_time'].dt.day
```

In [58]:

Out[58]:

	segment_key	trip_uuid	data	rc
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26363	trip-153861115439069069_IND628204AAA_IND627657AAA	trip-153861115439069069	test	
26364	trip-153861115439069069_IND628613AAA_IND627005AAA	trip-153861115439069069	test	
26365	trip-153861115439069069_IND628801AAA_IND628204AAA	trip-153861115439069069	test	
26366	trip-153861118270144424_IND583119AAA_IND583101AAA	trip-153861118270144424	test	
26367	trip-153861118270144424_IND583201AAA_IND583119AAA	trip-153861118270144424	test	

26368 rows × 28 columns

In [59]:

Out[59]: 26368

In [60]:
```
print(seg_agg_data['source_city'].count())
print(seg_agg_data['source_state'].count())
print(seg_agg_data['source_place'].count())
print(seg_agg_data['destination_city'].count())
print(seg_agg_data['destination_state'].count())
```

```
26302
26302
26302
26287
26287
26287
```

In [61]:

In [62]:
```
seg_agg_data['source_city'].fillna('unkown', inplace=True)
seg_agg_data['source_place'].fillna('unkown', inplace=True)
seg_agg_data['source_state'].fillna('unkown', inplace=True)
seg_agg_data['destination_city'].fillna('unkown', inplace=True)
seg_agg_data['destination_place'].fillna('unkown', inplace=True)
```

In [63]:
```
seg_agg_data.loc[seg_agg_data['source_place']=='','source_place'] = seg_agg_da
```

In [64]:
```
seg_agg_data.loc[seg_agg_data['destination_place']=='','destination_place'] =
```

In [65]:
```
```

Out[65]:
```
segment_key                        0
trip_uuid                          0
data                               0
route_type                         0
trip_creation_time                 0
source_name                        0
destination_name                   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
segment_actual_time_sum            0
segment_osrm_time_sum              0
segment_osrm_distance_sum          0
od_time_diff                       0
source_city                        0
source_place                       0
source_state                       0
destination_city                   0
destination_place                  0
destination_state                  0
trip_creation_Year                 0
trip_creation_Month                0
trip_creation_Day                  0
dtype: int64
```

In [66]:
```
seg_agg_data.loc[seg_agg_data['source_city']=='Bangalore','source_city']='Beng
```

In [67]:

Out[67]:

	segment_key	trip_uuid	data	route_
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Ca
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Ca
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	

5 rows × 28 columns

In [68]:

Out[68]:

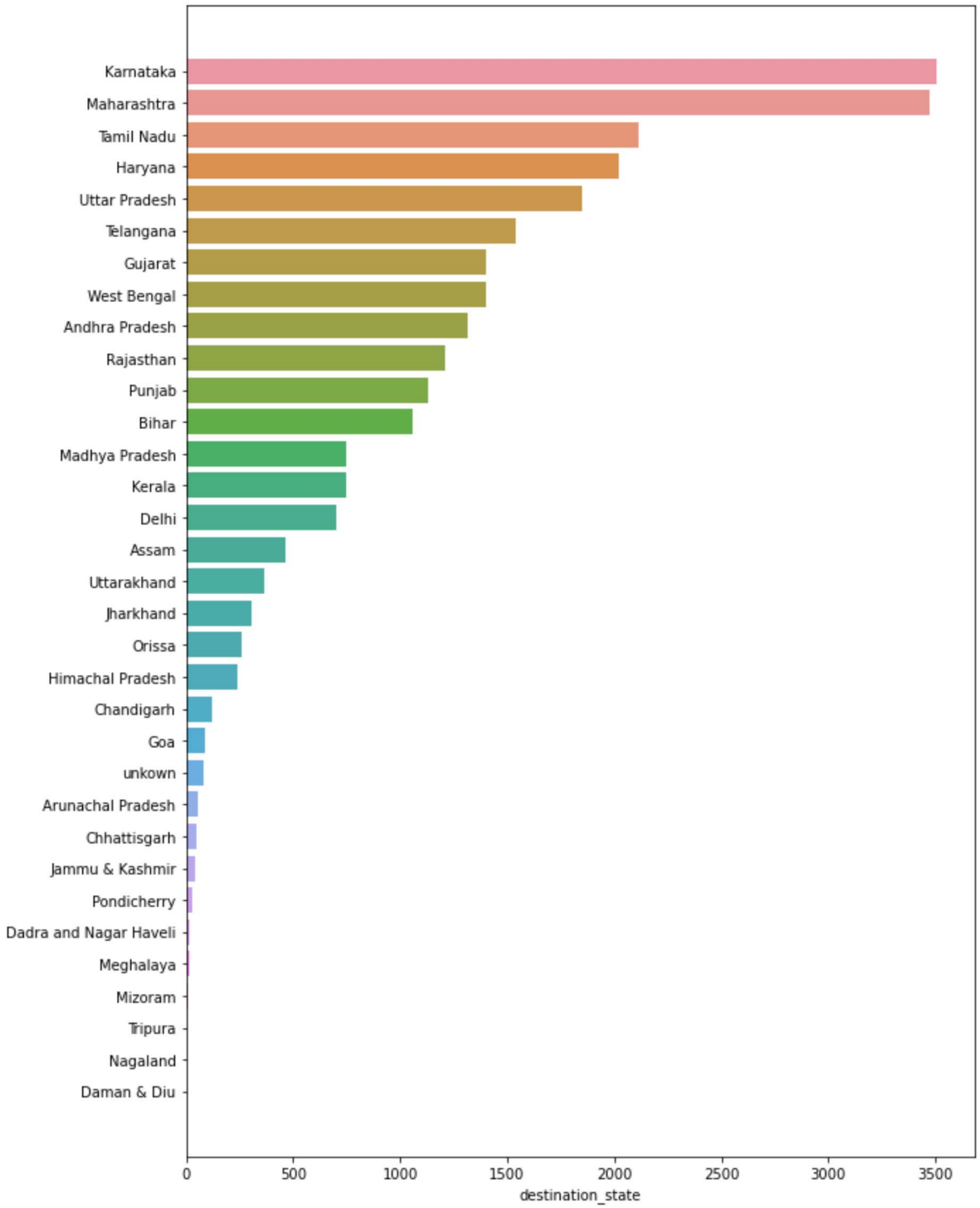
	segment_key	trip_uuid	data	route_
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Ca
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Ca
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	

5 rows × 28 columns

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

In [69]:
```
plt.figure(figsize = (10, 15))
sns.barplot(data = seg_agg_data,
            x = seg_agg_data['destination_state'].value_counts(),
            y = seg_agg_data['destination_state'].value_counts().index)
```

Out[69]: []



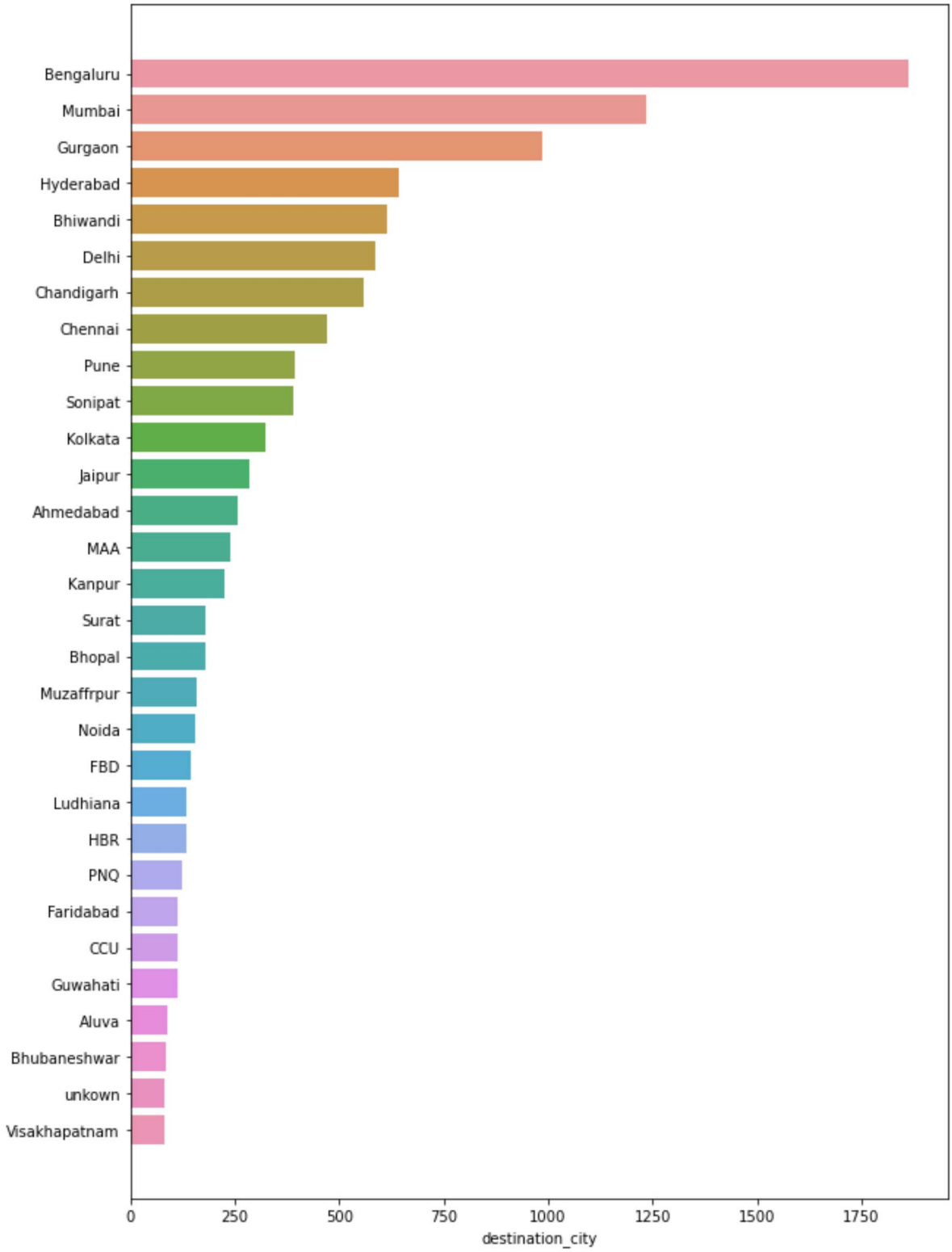
- It can be seen in the above plot that maximum trips originated from Karnataka state followed by Maharashtra and Tamil Nadu. That means that the seller base is strong in these states

In [ ]:

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

In [70]:
```
plt.figure(figsize = (10, 15))
sns.barplot(data = seg_agg_data,
            x = seg_agg_data['destination_city'].value_counts().nlargest(30),
            y = seg_agg_data['destination_city'].value_counts().nlargest(30).i
```

Out[70]: []

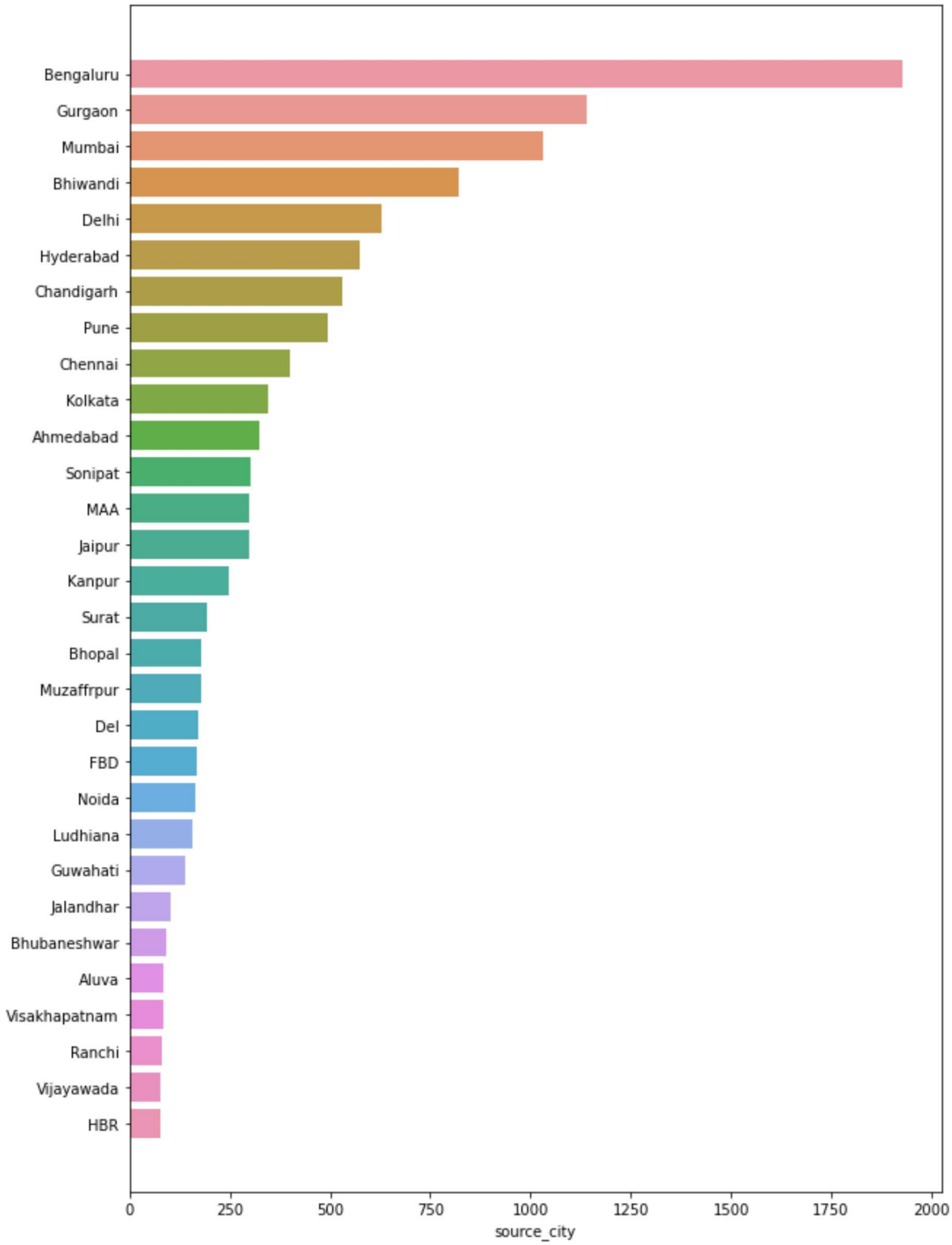


- It can be seen in the above plot that maximum trips ended in Bengaluru city followed by

Mumbai, Gurgaon, Hydrabad and Bhiwandi. That means that the number of orders placed in these cities is significantly high.

In [71]:
```
plt.figure(figsize = (10, 15))
sns.barplot(data = seg_agg_data,
            x = seg_agg_data['source_city'].value_counts().nlargest(30),
            y = seg_agg_data['source_city'].value_counts().nlargest(30).index)
```

Out[71]: []



In [ ]:

In [72]:
```
numerical_columns = ['od_time_diff', 'start_scan_to_end_scan', 'actual_distanc
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actu
                     'segment_osrm_time', 'segment_osrm_distance']
sns.pairplot(data = seg_agg_data,
             vars = numerical_columns,
             kind = 'reg',
             hue = 'route_type',
             markers = '.')
```

Out[72]: []

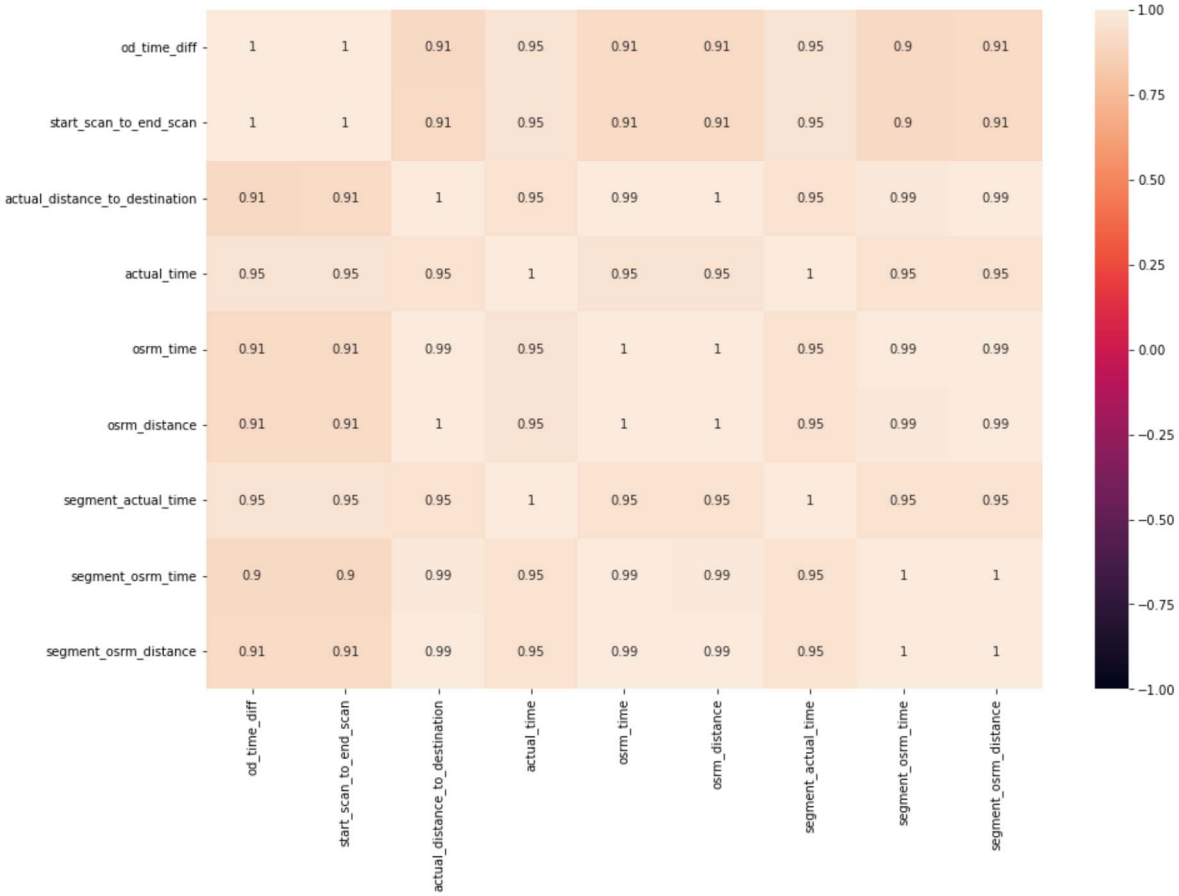
In [73]:
```
df_corr = seg_agg_data[numerical_columns].corr()
```

Out[73]:

	od_time_diff	start_scan_to_end_scan	actual_distance_to_destinatic
od_time_diff	1.000000	0.999796	0.90596
start_scan_to_end_scan	0.999796	1.000000	0.90626
actual_distance_to_destination	0.905961	0.906266	1.00000
actual_time	0.954179	0.954473	0.94899
osrm_time	0.910881	0.911197	0.99489
osrm_distance	0.911050	0.911361	0.99721
segment_actual_time	0.954216	0.954460	0.94798
segment_osrm_time	0.904097	0.904397	0.98716
segment_osrm_distance	0.907828	0.908123	0.99274

In [74]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, vmin = -1, vmax = 1, annot = True)
```

Out[74]: []



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

Type *Markdown* and LaTeX: α^2

4. In-depth analysis:

Grouping and Aggregating at Trip-level

- Groups the segment data by the trip_uuid column to focus on aggregating data at the trip level.
- Apply suitable aggregation functions like first, last, and sum specified in the create_trip_dict dictionary to calculate summary statistics for each trip.

Outlier Detection & Treatment

- Find any existing outliers in numerical features.
- Visualize the outlier values using Boxplot.
- Handle the outliers using the IQR method.
- Perform one-hot encoding on categorical features.
- Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [75]:

In [76]:
```
segment_dict = {
    'data': 'first',
    'route_type': 'first',
    'trip_creation_time': 'first',
    'source_name': 'first',
    'destination_name': 'last',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'first',
    'actual_distance_to_destination': 'last',
    'actual_time': 'last',
    'osrm_time': 'last',
    'osrm_distance': 'last',
    'segment_actual_time' : 'sum',
    'segment_osrm_time' : 'sum',
    'segment_osrm_distance' : 'sum',
    'segment_actual_time_sum': 'last',
    'segment_osrm_time_sum': 'last',
    'segment_osrm_distance_sum': 'last'
}

trip_agg_data = df.groupby('trip_uuid').agg(segment_dict).reset_index()
```

Out[76]:

	trip_uuid	data	route_type	trip_creation_time	source_name
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Bhopal_Trnsport_H (Madhya Pradesh)
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Tumkur_Veersagr_I (Karnataka)
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Bangalore_Nelmngla_H (Karnataka)
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)
...	...	...	...	...	...
14812	trip-153861095625827784	test	Carting	2018-10-03 23:55:56.258533	Chandigarh_Mehmdpur_H (Punjab)
14813	trip-153861104386292051	test	Carting	2018-10-03 23:57:23.863155	FBD_Balabhgarh_DPC (Haryana)
14814	trip-153861106442901555	test	Carting	2018-10-03 23:57:44.429324	Kanpur_Central_H_6 (Uttar Pradesh)
14815	trip-153861115439069069	test	Carting	2018-10-03 23:59:14.390954	Tirunelveli_VdkkuSrt_I (Tamil Nadu)
14816	trip-153861118270144424	test	FTL	2018-10-03 23:59:42.701692	Hospet (Karnataka)

14817 rows × 19 columns

In [77]:

Out[77]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm
count	14817.000000	14817.000000	14817.000000	14817.000000	148
mean	336.474396	111.335724	251.343384	108.337181	1
std	497.933472	247.784592	454.869781	218.236572	2
min	22.000000	9.002461	9.000000	6.000000	
25%	108.000000	20.420677	56.000000	24.000000	
50%	178.000000	35.582874	99.000000	41.000000	
75%	333.000000	76.454971	213.000000	82.000000	
max	7898.000000	1927.447754	4532.000000	1686.000000	23

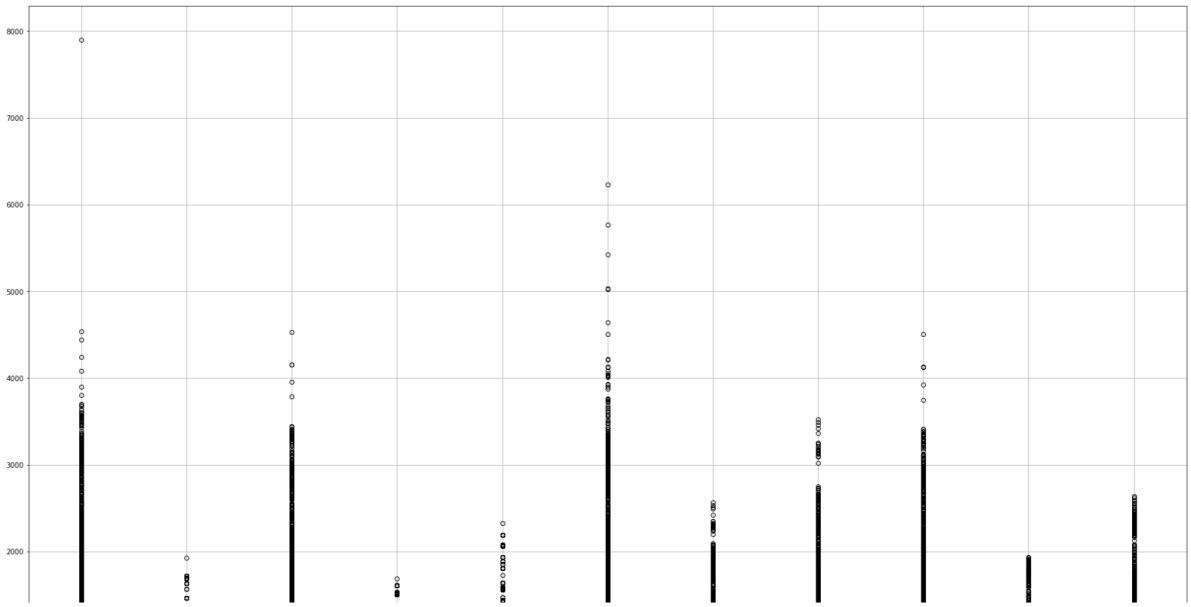
In [78]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14817 non-null  object
 1   data                            14817 non-null  category
 2   route_type                      14817 non-null  category
 3   trip_creation_time              14817 non-null  datetime64[ns]
 4   source_name                     14817 non-null  object
 5   destination_name                14817 non-null  object
 6   od_start_time                   14817 non-null  datetime64[ns]
 7   od_end_time                     14817 non-null  datetime64[ns]
 8   start_scan_to_end_scan          14817 non-null  float32
 9   actual_distance_to_destination  14817 non-null  float32
 10  actual_time                     14817 non-null  float32
 11  osrm_time                       14817 non-null  float32
 12  osrm_distance                   14817 non-null  float32
 13  segment_actual_time             14817 non-null  float32
 14  segment_osrm_time               14817 non-null  float32
 15  segment_osrm_distance           14817 non-null  float32
 16  segment_actual_time_sum         14817 non-null  float32
 17  segment_osrm_time_sum           14817 non-null  float32
 18  segment_osrm_distance_sum       14817 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(11), object(3)
memory usage: 1.3+ MB
```

In [79]:

In [80]:
```
plt.figure(figsize=(30,20))
```

Out[80]: <AxesSubplot:>



In [81]:

In [82]:

In [83]:

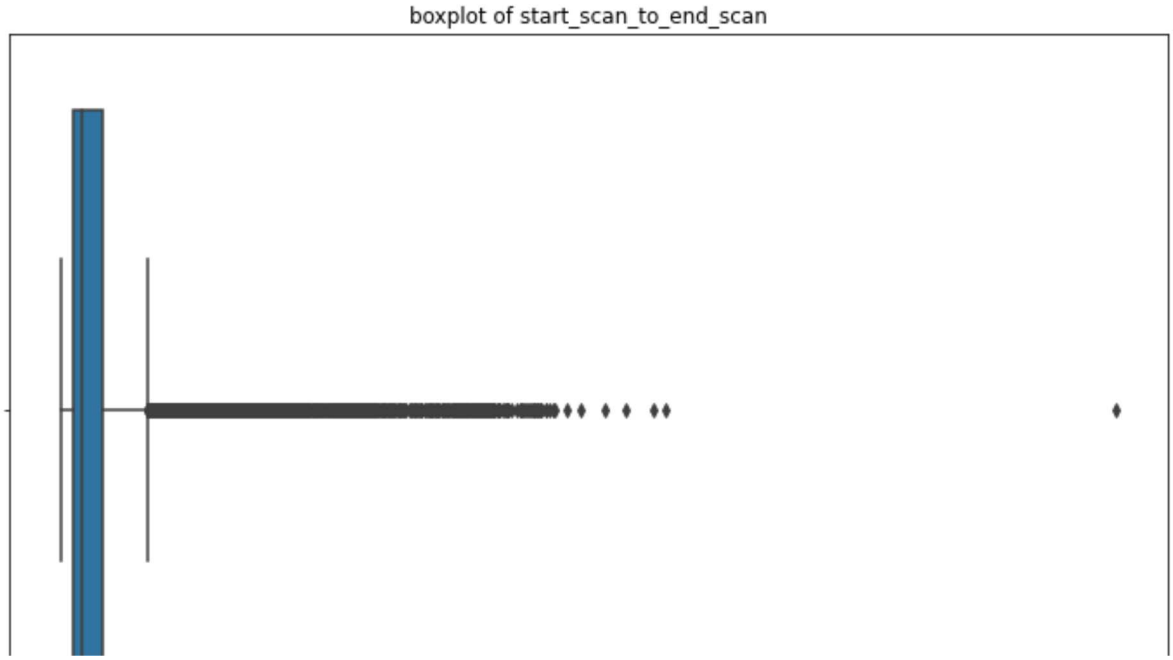
Out[83]:
```
start_scan_to_end_scan            108.000000
actual_distance_to_destination     20.420677
actual_time                        56.000000
osrm_time                          24.000000
osrm_distance                      26.898701
segment_actual_time                66.000000
segment_osrm_time                  31.000000
segment_osrm_distance              32.654499
segment_actual_time_sum            55.000000
segment_osrm_time_sum              25.000000
segment_osrm_distance_sum          27.887499
Name: 0.25, dtype: float64
```

In [84]:

In [85]:
```
IQR
```

Out[85]:
```
start_scan_to_end_scan            225.000000
actual_distance_to_destination     56.034294
actual_time                       157.000000
osrm_time                          58.000000
osrm_distance                      71.704502
segment_actual_time               301.000000
segment_osrm_time                 154.000000
segment_osrm_distance             186.147900
segment_actual_time_sum           157.000000
segment_osrm_time_sum              68.000000
segment_osrm_distance_sum          81.066999
dtype: float64
```

In [86]:
```
for i, col in enumerate(numeric_col):
    plt.figure(figsize=(12,8))
    sns.boxplot(data=numeric_col, x=col)
    plt.title(f'boxplot of {col}')
```



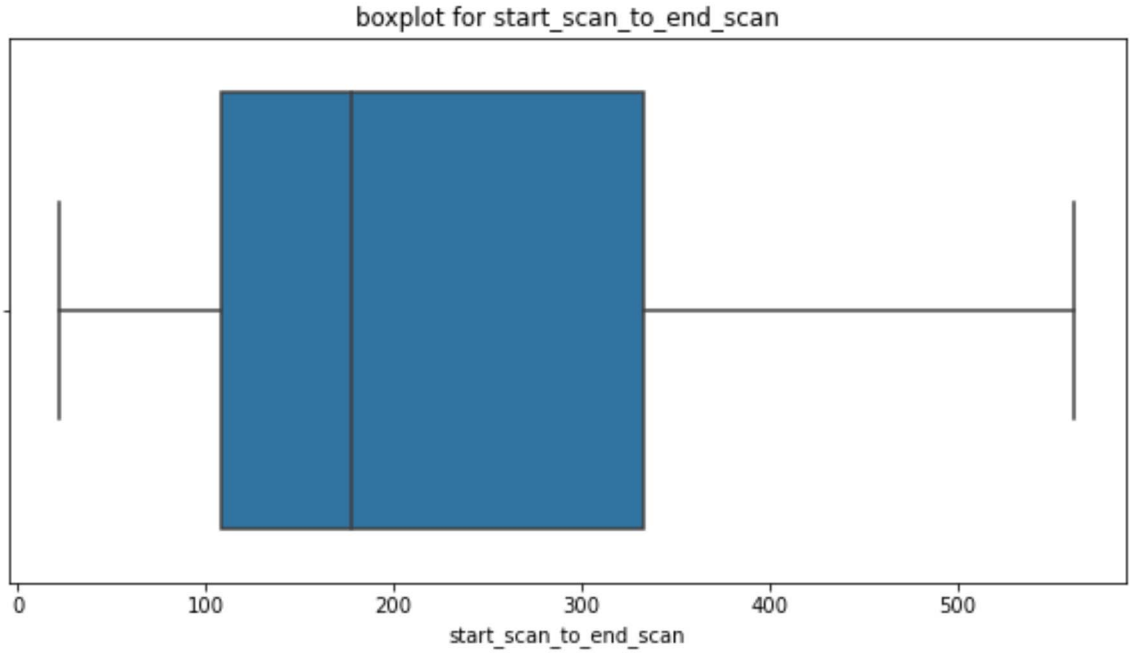
In [87]:
```
for i, col in enumerate(numeric_col):
    q1 = np.percentile(trip_agg_data[col],25)
    q3 = np.percentile(trip_agg_data[col],75)

    iqr = q3 - q1

    lower_bound = iqr - (1.5*iqr)
    upper_bound = iqr + (1.5*iqr)

    clipped = np.clip(trip_agg_data[col], lower_bound, upper_bound)

    plt.figure(figsize=(10,5))
    sns.boxplot(data=clipped.to_frame(),x=col)
    plt.title(f'boxplot for {col}')
```



- 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.

In [ ]:

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

In [88]:
```
# Get value counts before one-hot encoding
```

Out[88]:
```
FTL        13939
Carting    12429
Name: route_type, dtype: int64
```

In [89]:
```
# Perform one-hot encoding on categorical column route type
from sklearn.preprocessing import LabelEncoder
encoded_data = copy.deepcopy(seg_agg_data)
label_encoder = LabelEncoder()
```

In [90]:
```
# Get value counts after one-hot encoding
```

Out[90]:
```
1    13939
0    12429
Name: route_type, dtype: int64
```

In [91]:
```
# Get value counts of categorical variable 'data' before one-hot encoding
```

Out[91]:
```
training    18947
test         7421
Name: data, dtype: int64
```

In [92]:
```
label_encoder = LabelEncoder()
```

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

Out[93]:
```
1    18947
0     7421
Name: data, dtype: int64
```

In [94]:

Out[94]:

	segment_key	trip_uuid	data	route_ty
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	1	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	1	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	1	
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	1	
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	1	

5 rows × 28 columns

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler

In [95]:
```
from sklearn.preprocessing import MinMaxScaler
```

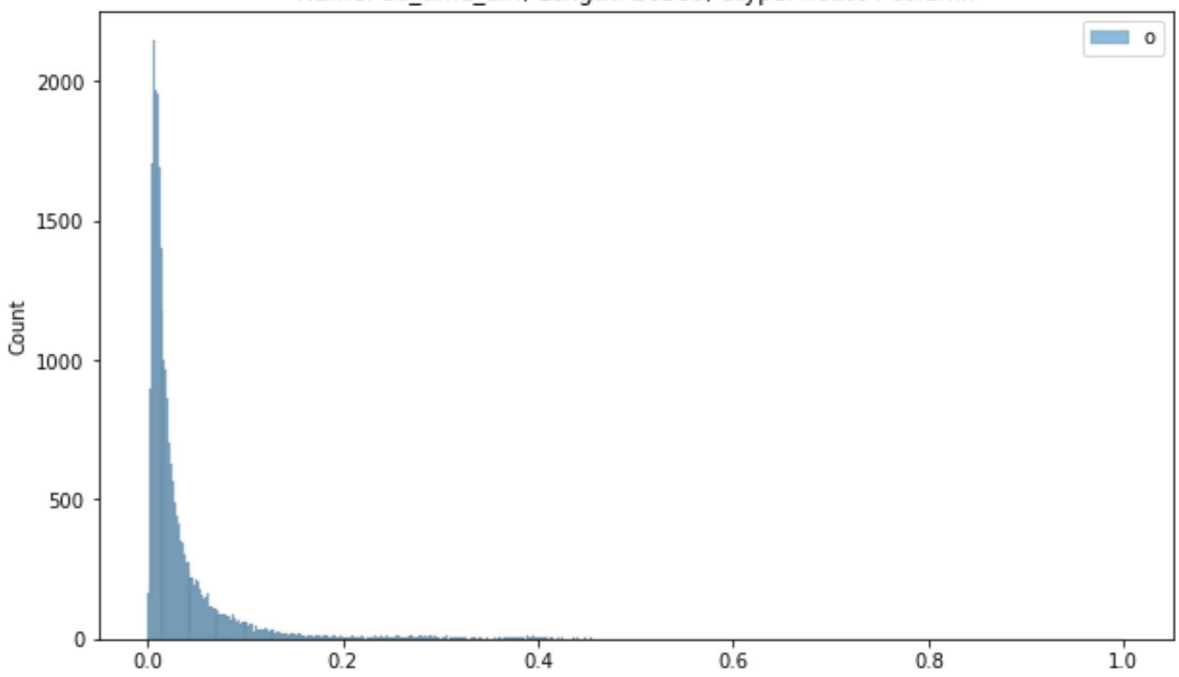
In [96]:

Out[96]:
```
Index(['segment_key', 'trip_uuid', 'data', 'route_type', 'trip_creation_time
',
       'source_name', 'destination_name', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'segment_actual_time_sum',
       'segment_osrm_time_sum', 'segment_osrm_distance_sum', 'od_time_diff',
       'source_city', 'source_place', 'source_state', 'destination_city',
       'destination_place', 'destination_state', 'trip_creation_Year',
       'trip_creation_Month', 'trip_creation_Day'],
      dtype='object')
```

In [97]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['od_time_diff'].to_numpy().reshape(
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['od_time_diff']} column")
plt.legend('od_time_diff')
```

Out[97]:
```
[]
```

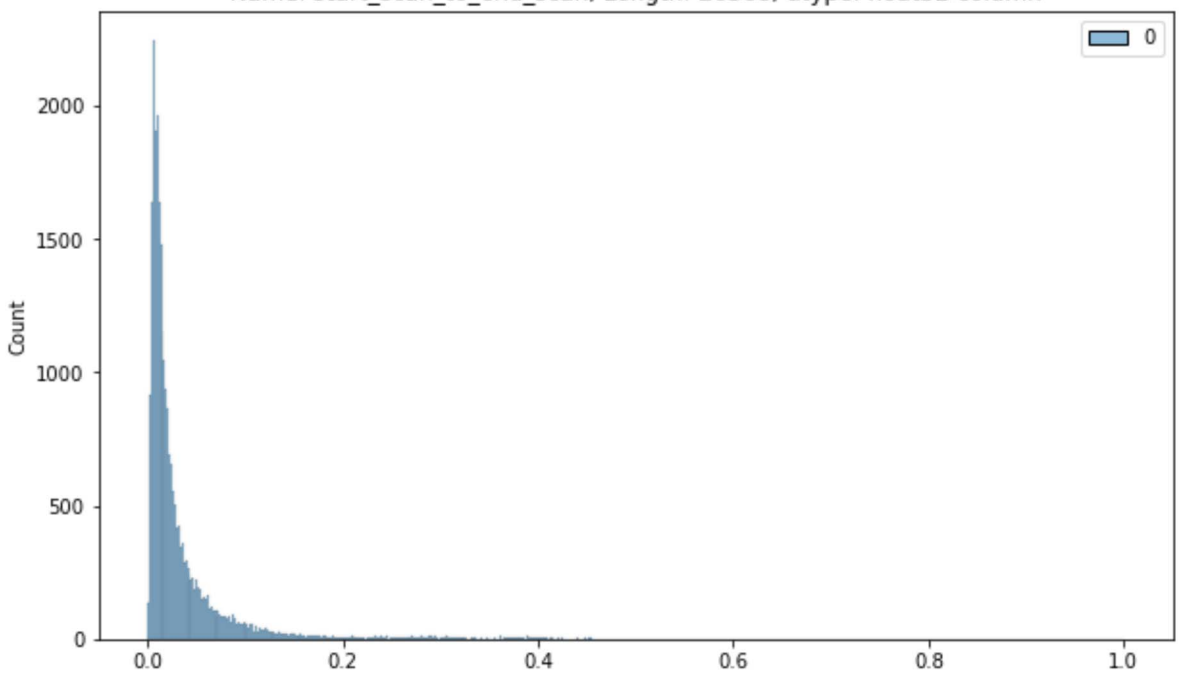
```
Normalized 0      21.01
1      16.66
2       0.98
3       2.05
4      13.91
       ...
26363   1.04
26364   1.52
26365   0.74
26366   4.79
26367   1.12
Name: od_time_diff, Length: 26368, dtype: float64 column
```



In [98]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['start_scan_to_end_scan'].to_numpy(
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['start_scan_to_end_scan']} column")
```

Out[98]: []

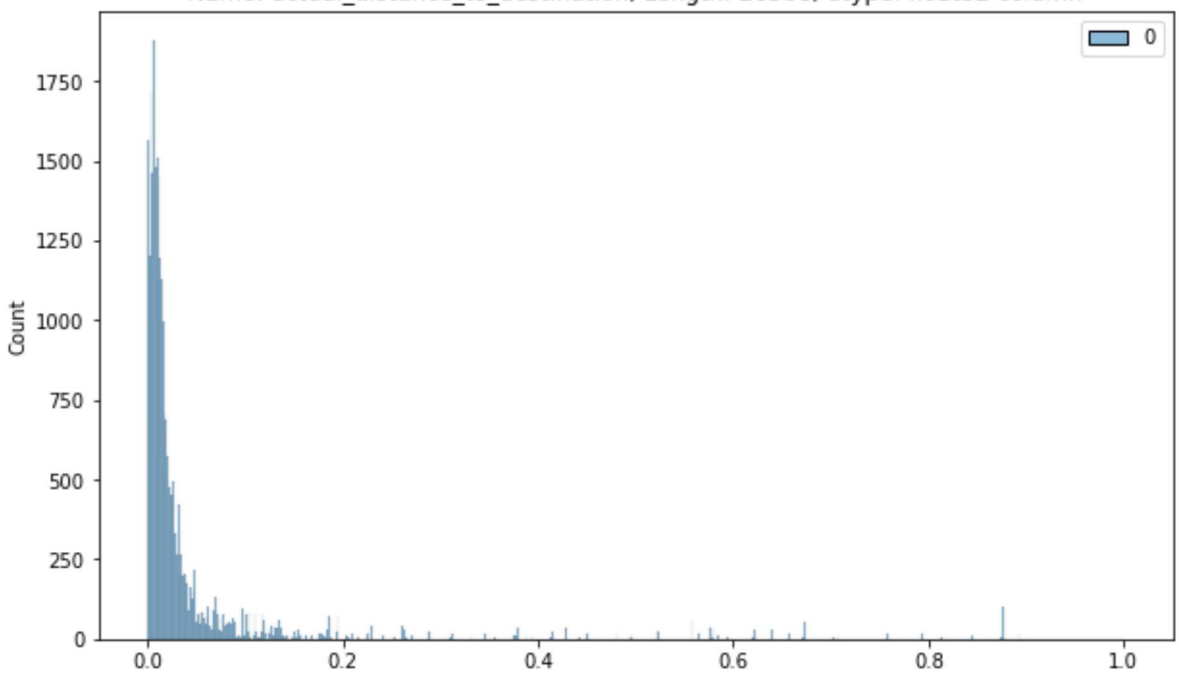
```
Normalized 0      1260.0
1       999.0
2        58.0
3       122.0
4       834.0
        ...
26363    62.0
26364    91.0
26365    44.0
26366   287.0
26367    66.0
Name: start_scan_to_end_scan, Length: 26368, dtype: float32 column
```



In [99]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['actual_distance_to_destination'].t
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['actual_distance_to_destination']} column
```

Out[99]: []

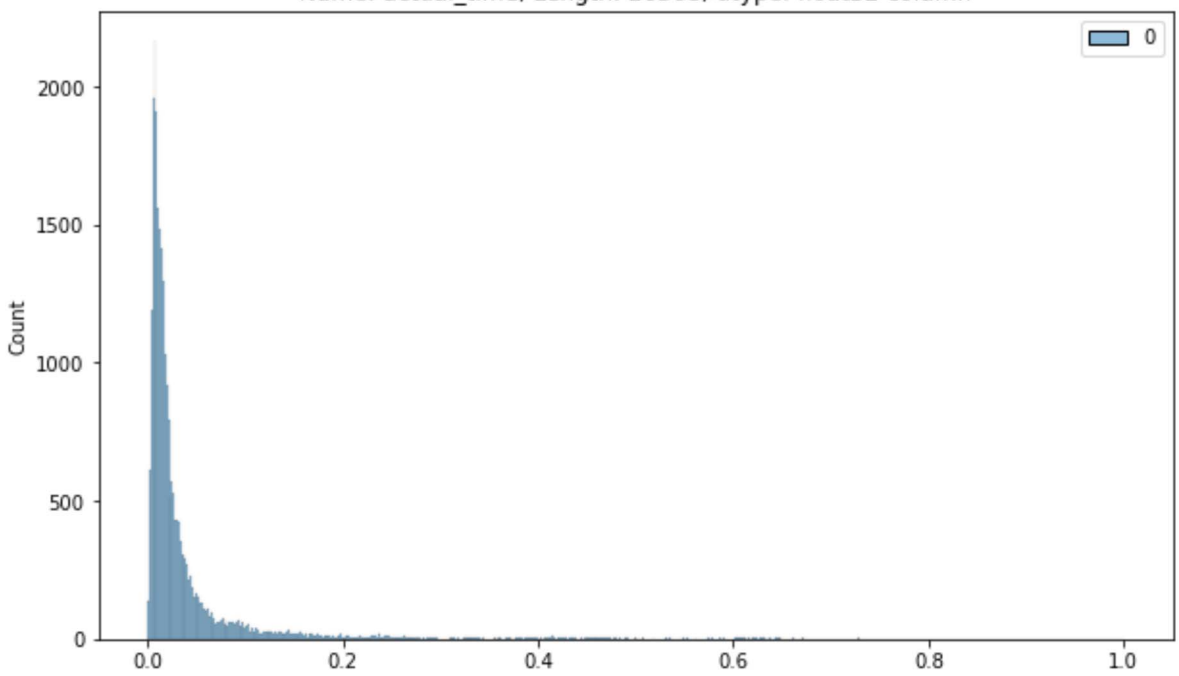
```
Normalized 0      383.759155
           1      440.973694
           2       24.644020
           3       48.542889
           4      237.439606
                     ...
       26363       33.627182
       26364       33.673836
       26365       12.661944
       26366       40.546738
       26367       25.534794
Name: actual_distance_to_destination, Length: 26368, dtype: float32 column
```



In [100]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['actual_time'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['actual_time']} column")
```

Out[100]: []

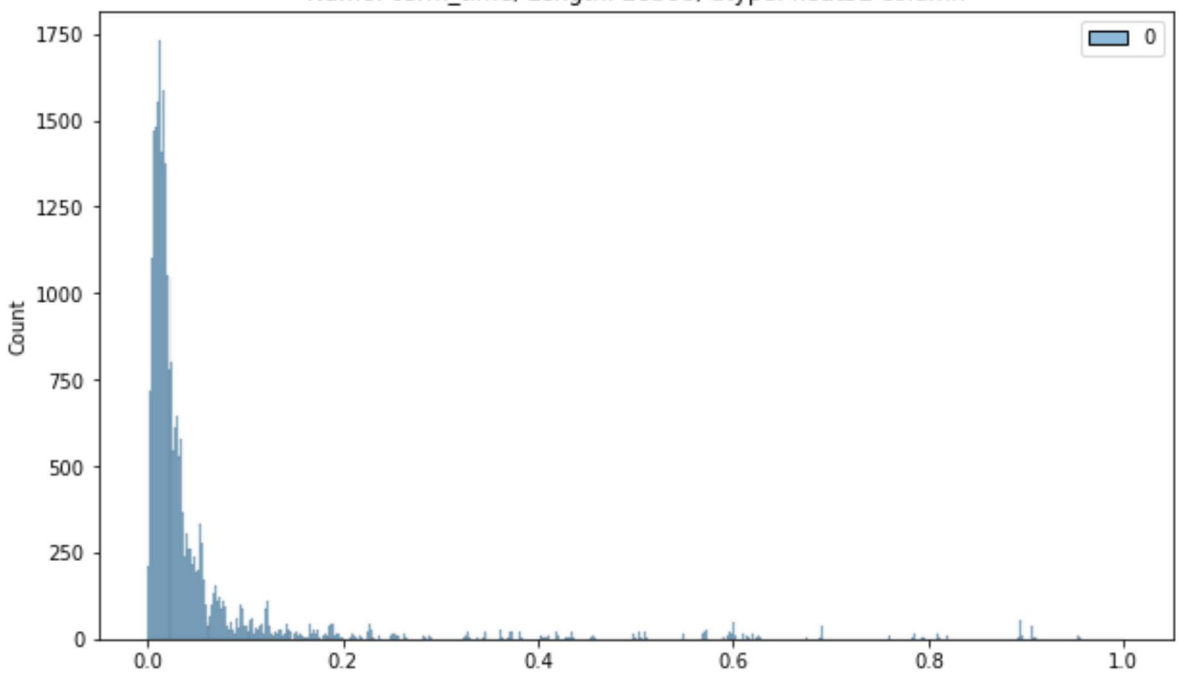
```
Normalized 0      732.0
           1      830.0
           2       47.0
           3       96.0
           4      611.0
                  ...
       26363       51.0
       26364       90.0
       26365       30.0
       26366      233.0
       26367       42.0
Name: actual_time, Length: 26368, dtype: float32 column
```



In [101]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['osrm_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['osrm_time']} column")
```

Out[101]: []

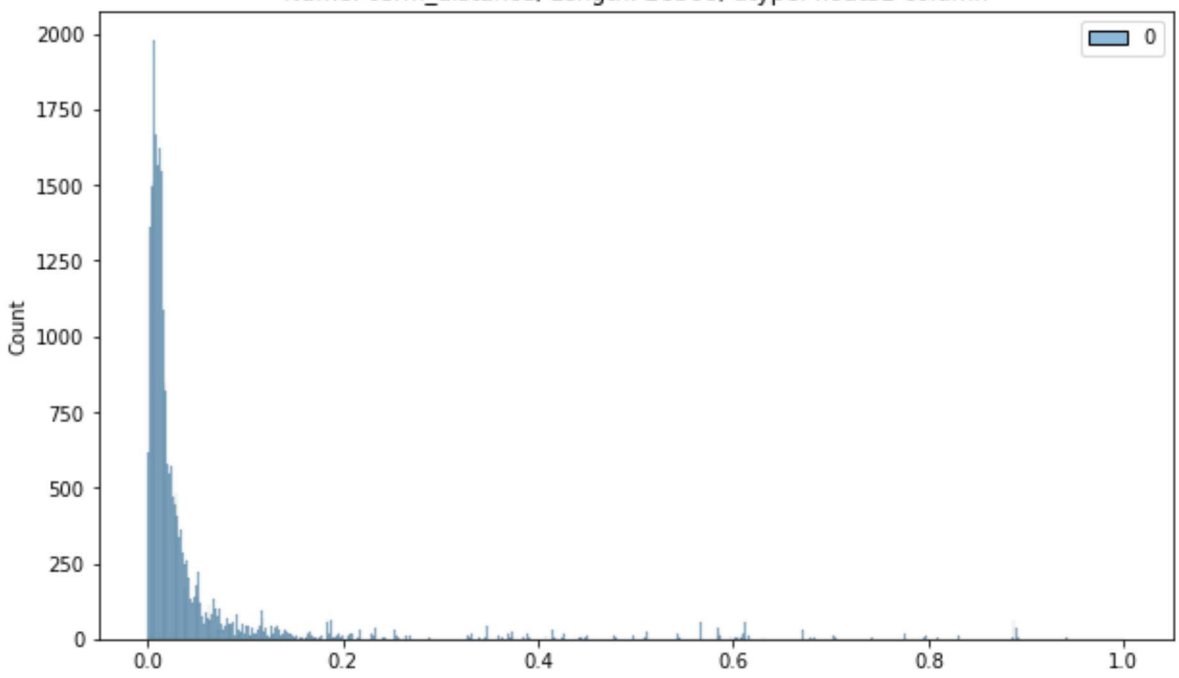
```
Normalized 0       329.0
           1       388.0
           2        26.0
           3        42.0
           4       212.0
                   ...
       26363        41.0
       26364        48.0
       26365        14.0
       26366        42.0
       26367        26.0
Name: osrm_time, Length: 26368, dtype: float32 column
```



In [102]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['osrm_distance'].to_numpy().reshape
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['osrm_distance']} column")
```

Out[102]: []

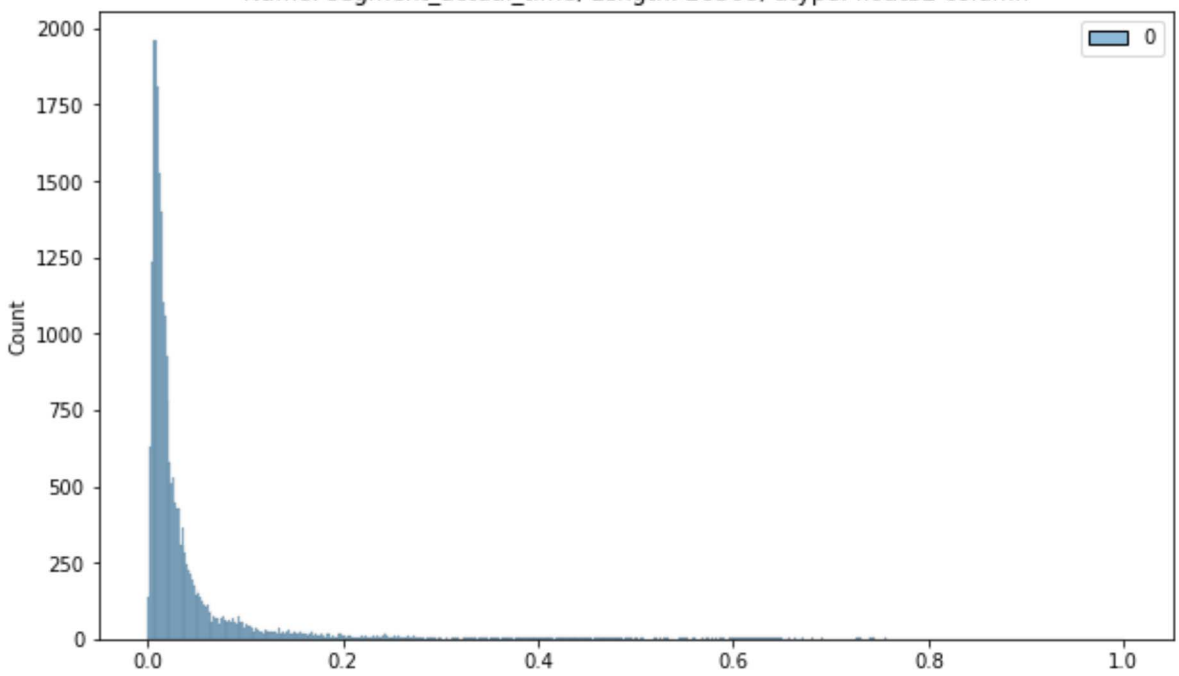
```
Normalized 0      446.549591
           1      544.802673
           2       28.199400
           3       56.911598
           4      281.210907
                     ...
       26363       42.521301
       26364       40.608002
       26365       16.018499
       26366       52.530300
       26367       28.048401
Name: osrm_distance, Length: 26368, dtype: float32 column
```



In [103]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['segment_actual_time'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['segment_actual_time']} column")
```

Out[103]: []

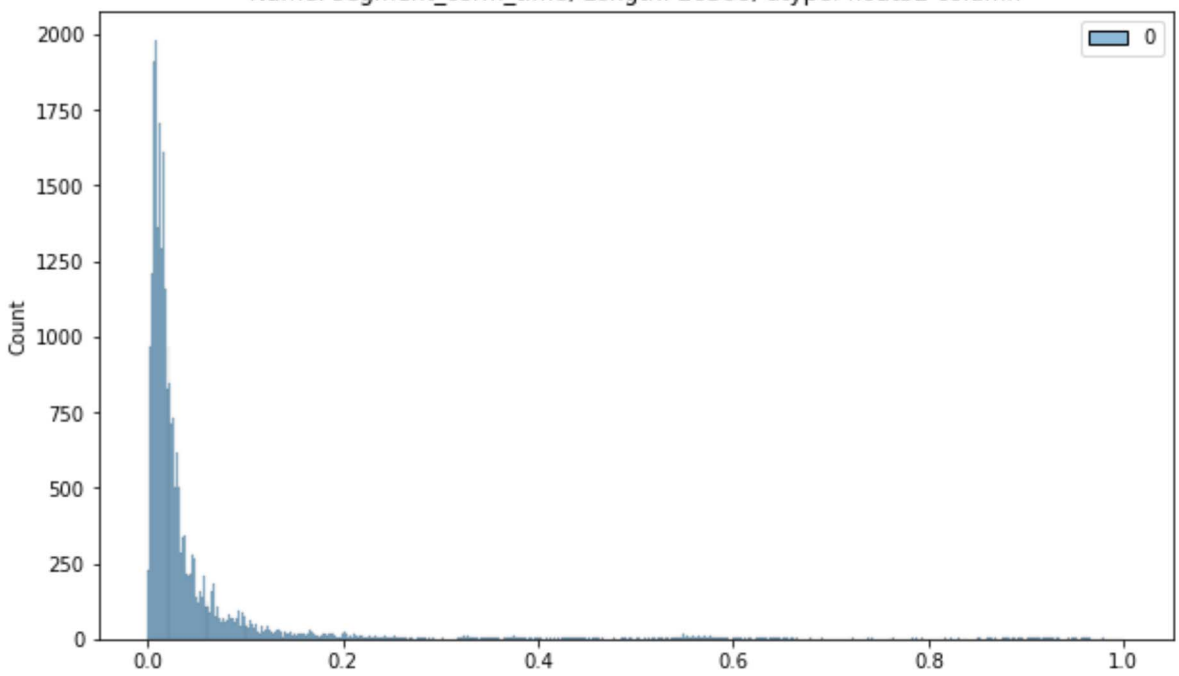
```
Normalized 0       728.0
           1       820.0
           2        46.0
           3        95.0
           4       608.0
                   ...
       26363        49.0
       26364        89.0
       26365        29.0
       26366       233.0
       26367        41.0
Name: segment_actual_time, Length: 26368, dtype: float32 column
```



In [104]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['segment_osrm_time'].to_numpy().res
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['segment_osrm_time']} column")
```

Out[104]: []

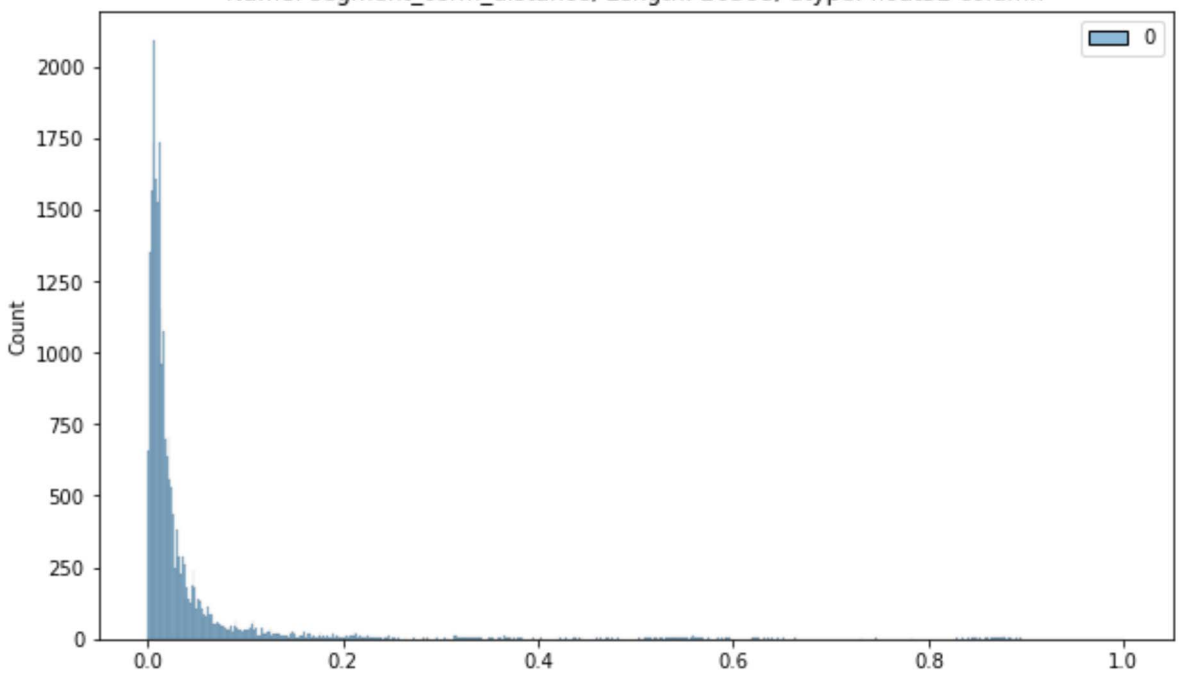
```
Normalized 0      534.0
           1      474.0
           2       26.0
           3       39.0
           4      231.0
                  ...
       26363       42.0
       26364       77.0
       26365       14.0
       26366       42.0
       26367       25.0
Name: segment_osrm_time, Length: 26368, dtype: float32 column
```



In [105]:
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(encoded_data['segment_osrm_distance'].to_numpy()
sns.histplot(scaled)
plt.title(f"Normalized {encoded_data['segment_osrm_distance']} column")
```

Out[105]: []

```
Normalized 0      670.620483
           1      649.852783
           2       28.199501
           3       55.989899
           4      317.740784
                     ...
       26363       42.143101
       26364       78.586899
       26365       16.018400
       26366       52.530300
       26367       28.048401
Name: segment_osrm_distance, Length: 26368, dtype: float32 column
```



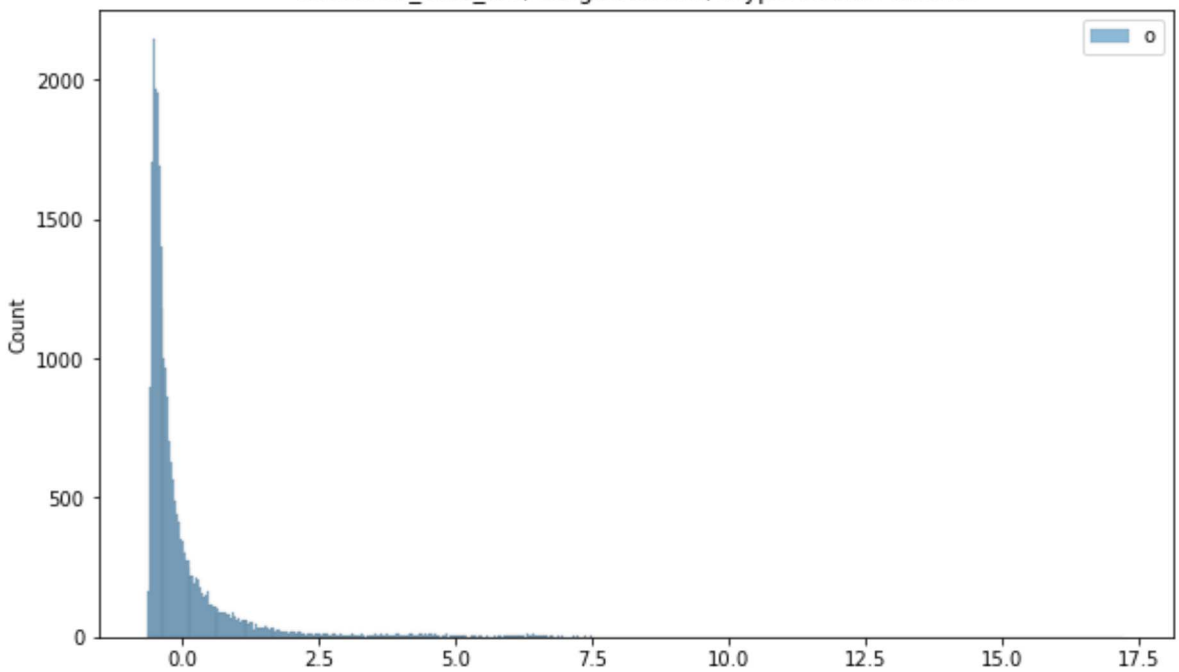
Column Standardization

In [106]:
```
```

In [107]:
```
plt.figure(figsize = (10, 6))
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(encoded_data['od_time_diff'].to_numpy().reshape(
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['od_time_diff']} column")
plt.legend('od_time_diff')
```

Out[107]: []

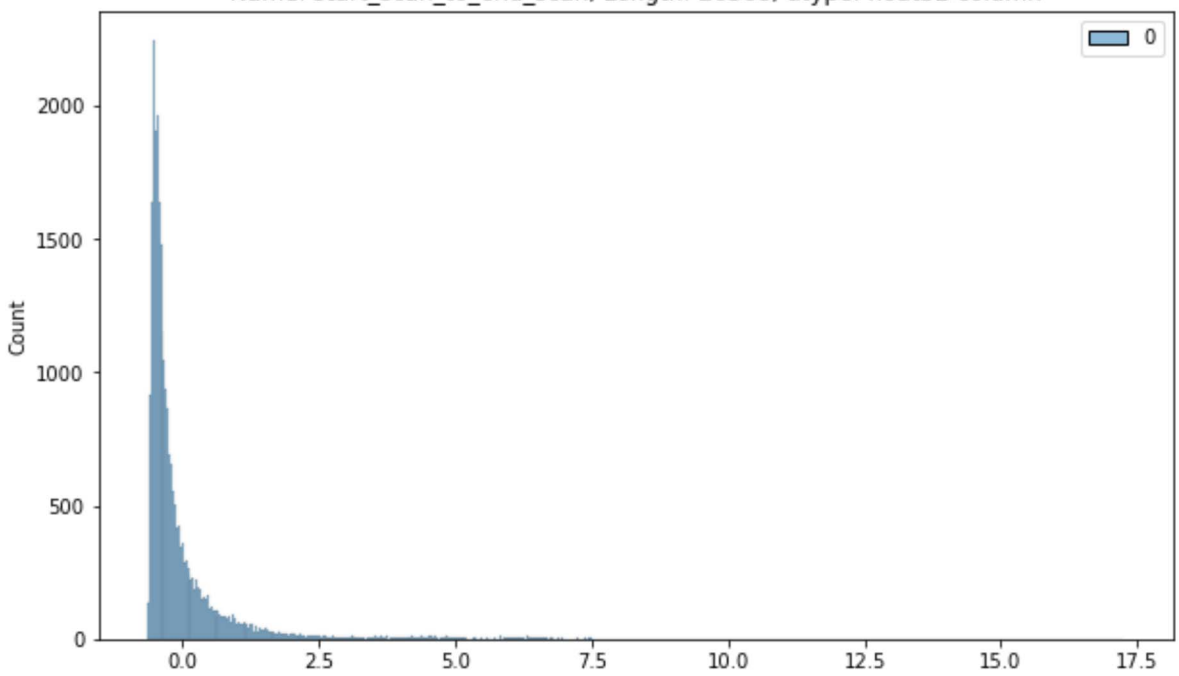
```
Standardized 0      21.01
             1      16.66
             2       0.98
             3       2.05
             4      13.91
                    ...
         26363      1.04
         26364      1.52
         26365      0.74
         26366      4.79
         26367      1.12
Name: od_time_diff, Length: 26368, dtype: float64 column
```



In [108]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['start_scan_to_end_scan'].to_numpy(
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['start_scan_to_end_scan']} column")
```

Out[108]: []

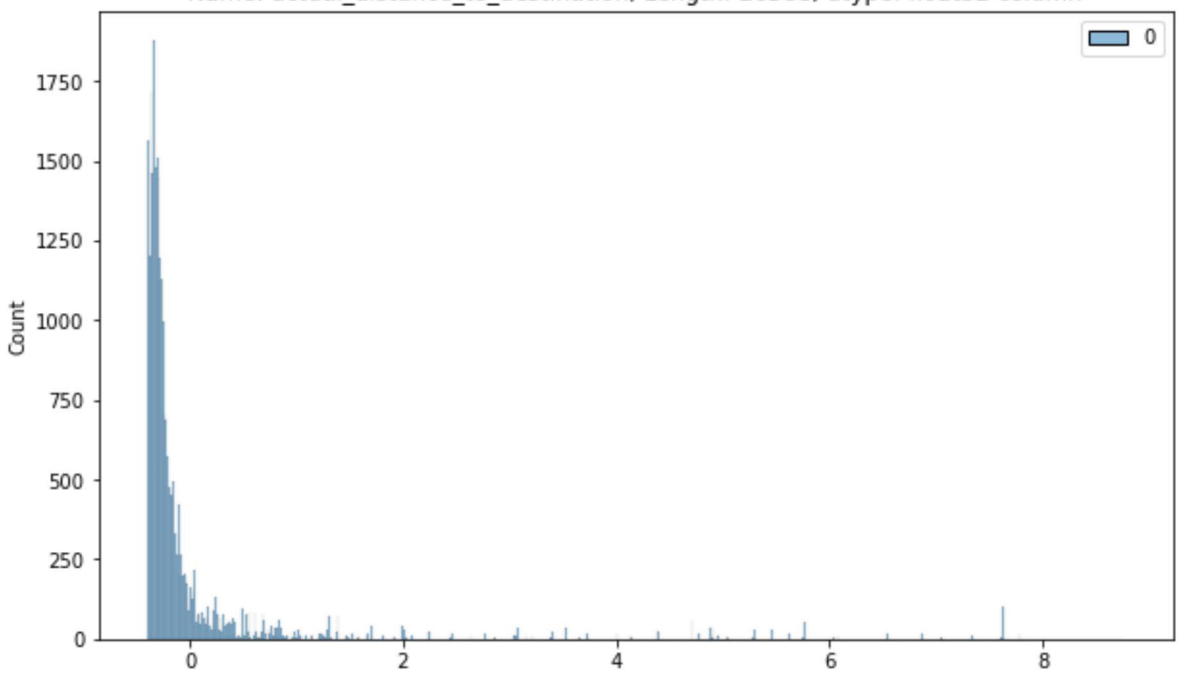
```
Standardized 0      1260.0
             1       999.0
             2        58.0
             3       122.0
             4       834.0
                      ...
         26363        62.0
         26364        91.0
         26365        44.0
         26366       287.0
         26367        66.0
Name: start_scan_to_end_scan, Length: 26368, dtype: float32 column
```



In [109]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['actual_distance_to_destination'].t
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['actual_distance_to_destination']} colu
```

Out[109]: []

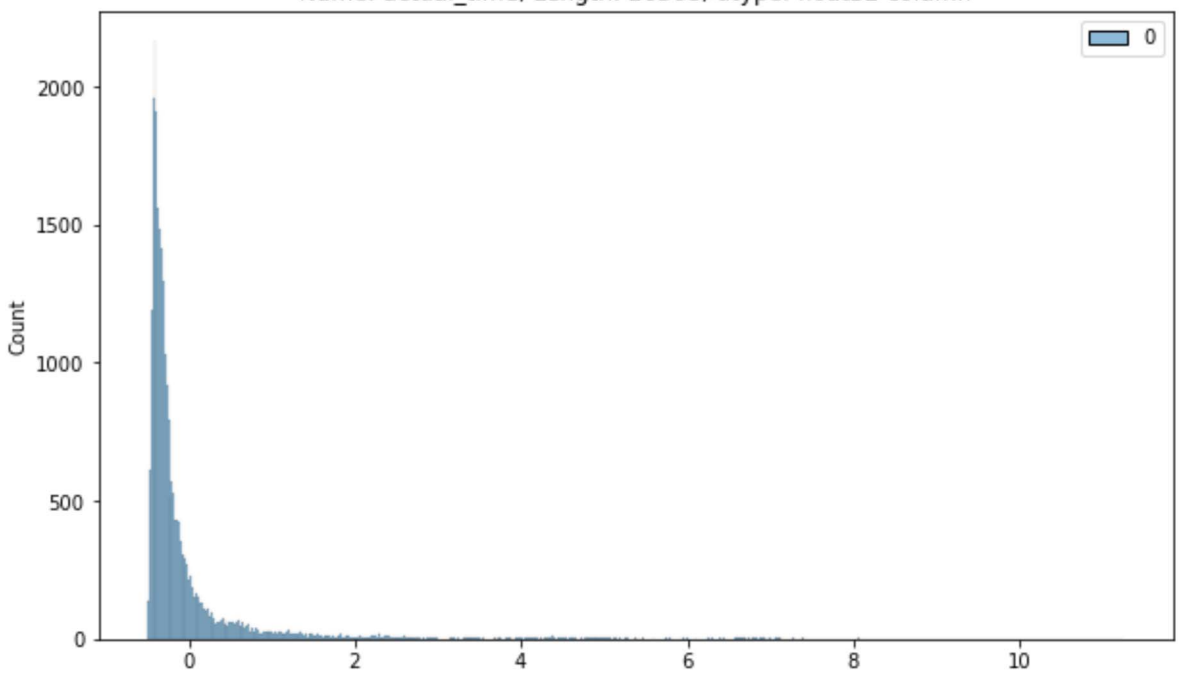
```
Standardized 0      383.759155
             1      440.973694
             2       24.644020
             3       48.542889
             4      237.439606
                        ...
             26363   33.627182
             26364   33.673836
             26365   12.661944
             26366   40.546738
             26367   25.534794
Name: actual_distance_to_destination, Length: 26368, dtype: float32 column
```



In [110]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['actual_time'].to_numpy().reshape(-
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['actual_time']} column")
```

Out[110]: []

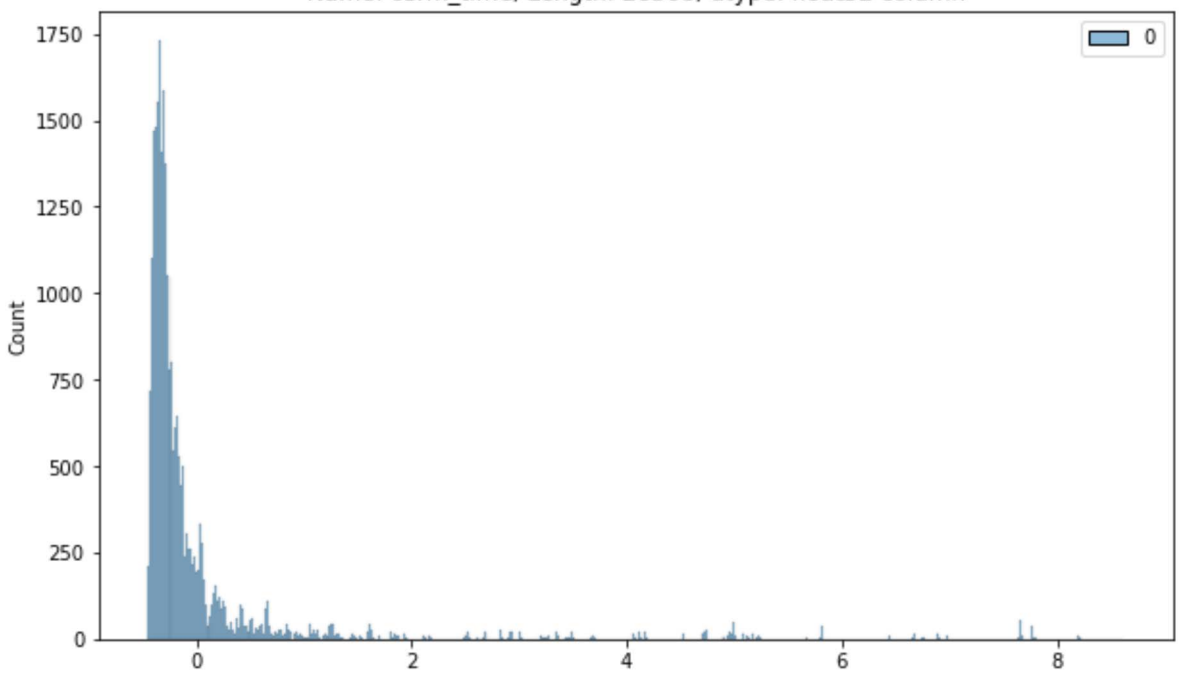
```
Standardized 0      732.0
             1      830.0
             2       47.0
             3       96.0
             4      611.0
                    ...
             26363   51.0
             26364   90.0
             26365   30.0
             26366  233.0
             26367   42.0
Name: actual_time, Length: 26368, dtype: float32 column
```



In [111]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['osrm_time'].to_numpy().reshape(-1,
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['osrm_time']} column")
```

Out[111]: []

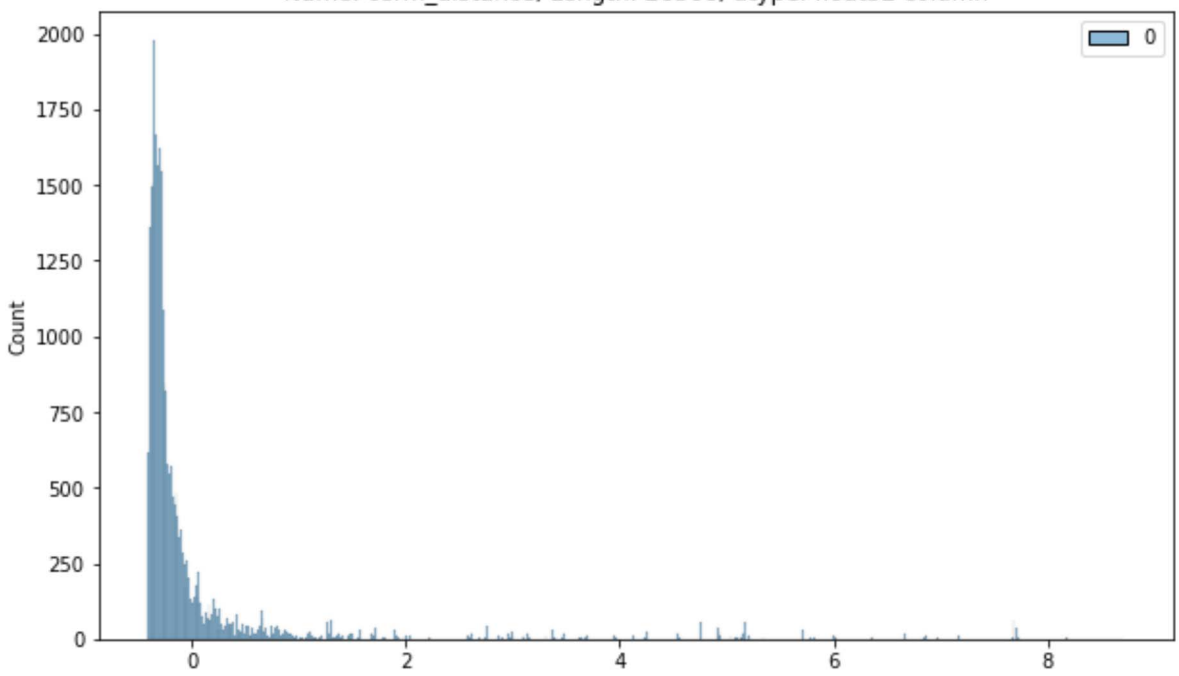
```
Standardized 0      329.0
1      388.0
2       26.0
3       42.0
4      212.0
       ...
26363   41.0
26364   48.0
26365   14.0
26366   42.0
26367   26.0
Name: osrm_time, Length: 26368, dtype: float32 column
```



In [112]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['osrm_distance'].to_numpy().reshape
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['osrm_distance']} column")
```

Out[112]: []

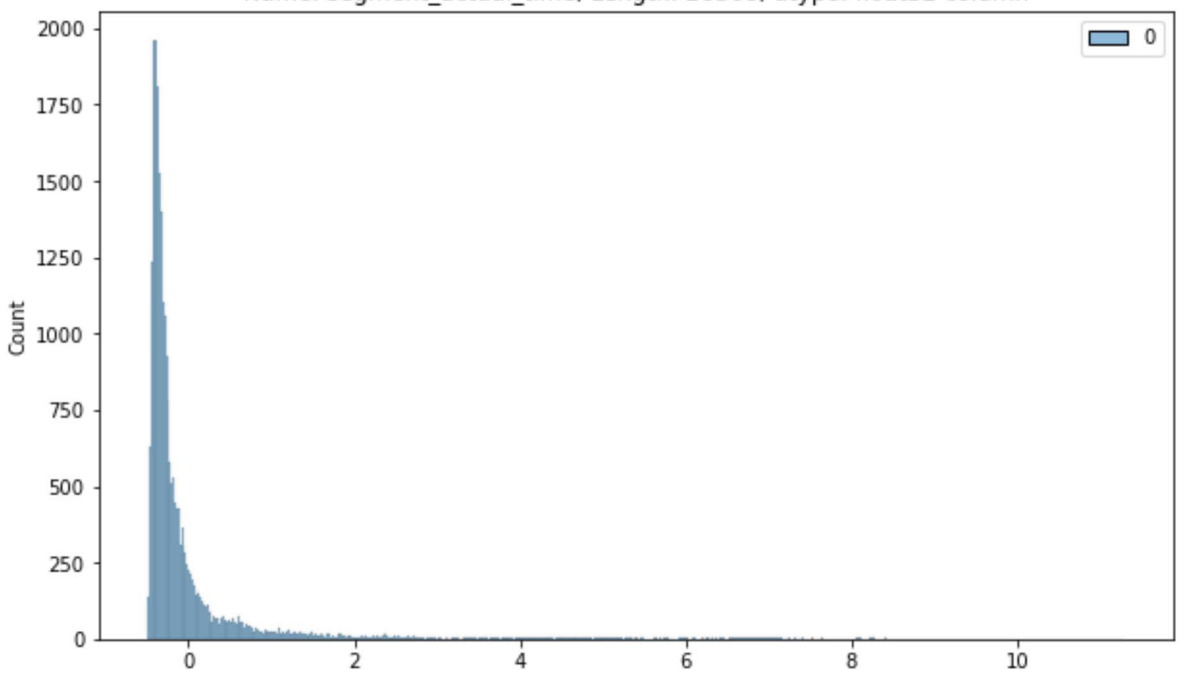
```
Standardized 0      446.549591
             1      544.802673
             2       28.199400
             3       56.911598
             4      281.210907
                        ...
         26363       42.521301
         26364       40.608002
         26365       16.018499
         26366       52.530300
         26367       28.048401
Name: osrm_distance, Length: 26368, dtype: float32 column
```



In [113]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['segment_actual_time'].to_numpy().r
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['segment_actual_time']} column")
```

Out[113]: []

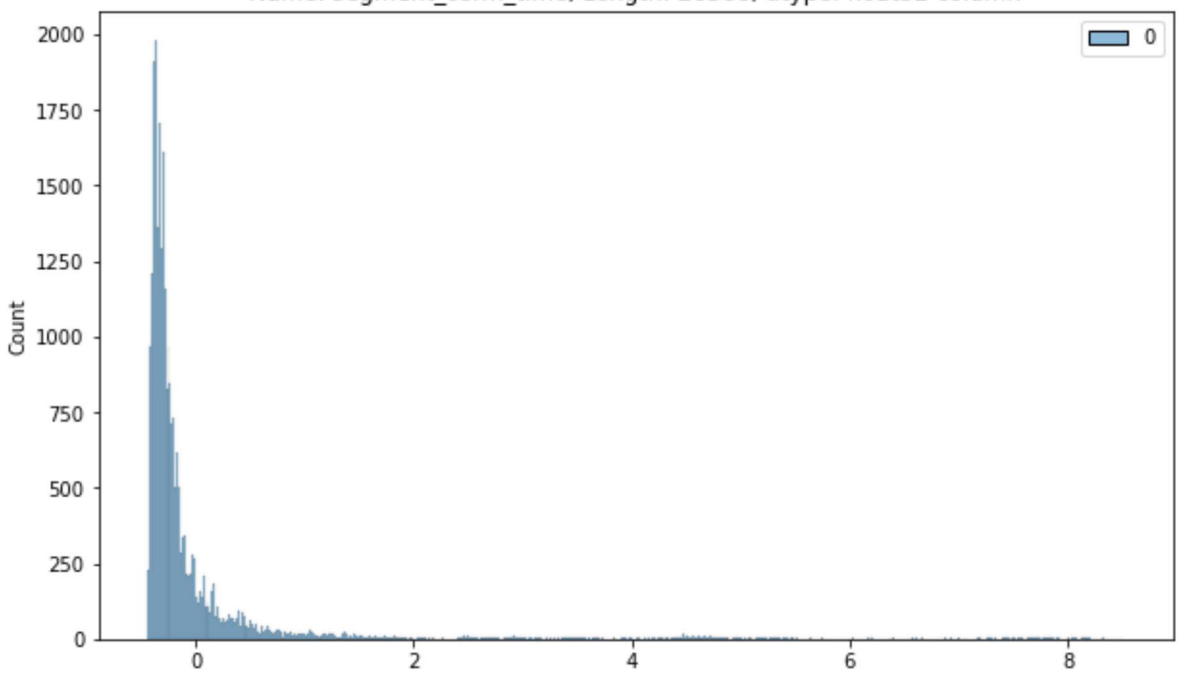
```
Standardized 0       728.0
             1       820.0
             2        46.0
             3        95.0
             4       608.0
                      ...
             26363    49.0
             26364    89.0
             26365    29.0
             26366   233.0
             26367    41.0
Name: segment_actual_time, Length: 26368, dtype: float32 column
```



In [114]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['segment_osrm_time'].to_numpy().res
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['segment_osrm_time']} column")
```

Out[114]: []

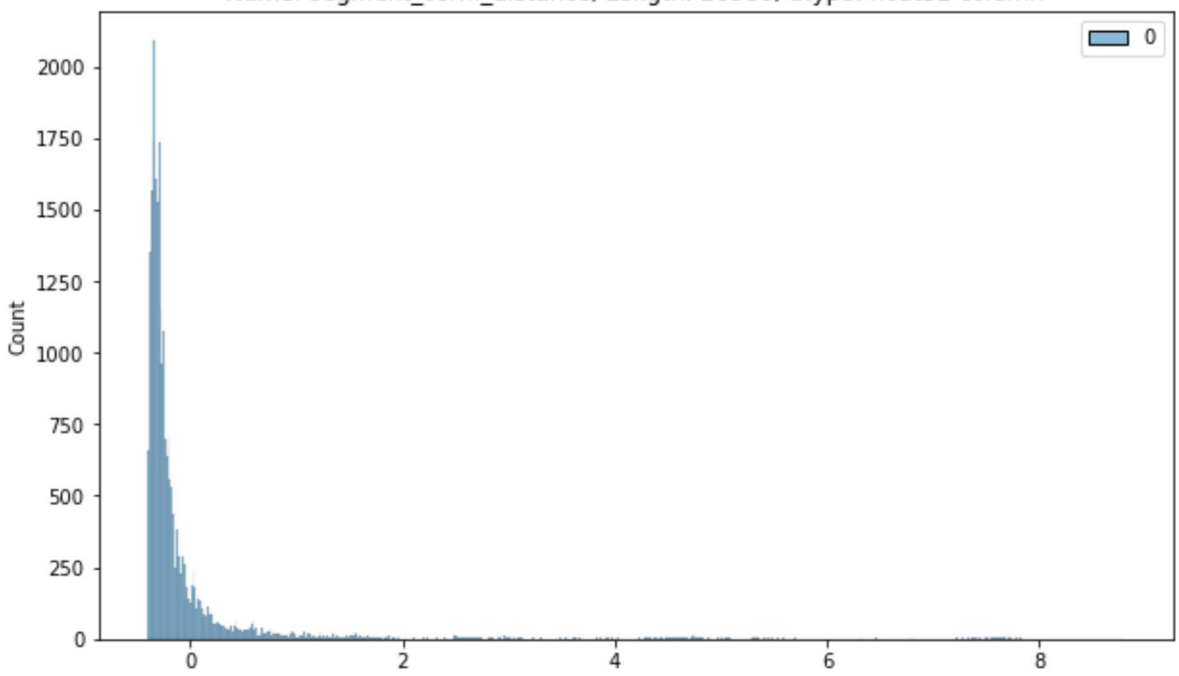
```
Standardized 0      534.0
             1      474.0
             2       26.0
             3       39.0
             4      231.0
                     ...
             26363   42.0
             26364   77.0
             26365   14.0
             26366   42.0
             26367   25.0
Name: segment_osrm_time, Length: 26368, dtype: float32 column
```



In [115]:
```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(encoded_data['segment_osrm_distance'].to_numpy()
sns.histplot(scaled)
plt.title(f"Standardized {encoded_data['segment_osrm_distance']} column")
```

Out[115]: []

```
Standardized 0      670.620483
             1      649.852783
             2       28.199501
             3       55.989899
             4      317.740784
                        ...
         26363       42.143101
         26364       78.586899
         26365       16.018400
         26366       52.530300
         26367       28.048401
Name: segment_osrm_distance, Length: 26368, dtype: float32 column
```



In [ ]:

5. Hypothesis Testing:

Perform hypothesis testing / visual analysis between :

- actual_time aggregated value and OSRM time aggregated value.
- actual_time aggregated value and segment actual time aggregated value.
- OSRM distance aggregated value and segment OSRM distance aggregated value.
- OSRM time aggregated value and segment OSRM time aggregated value.
- Note: Aggregated values are the values you'll get after merging the rows on the basis of trip_uuid.

In [116]:

Hypothesis testing / visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of segment_key)

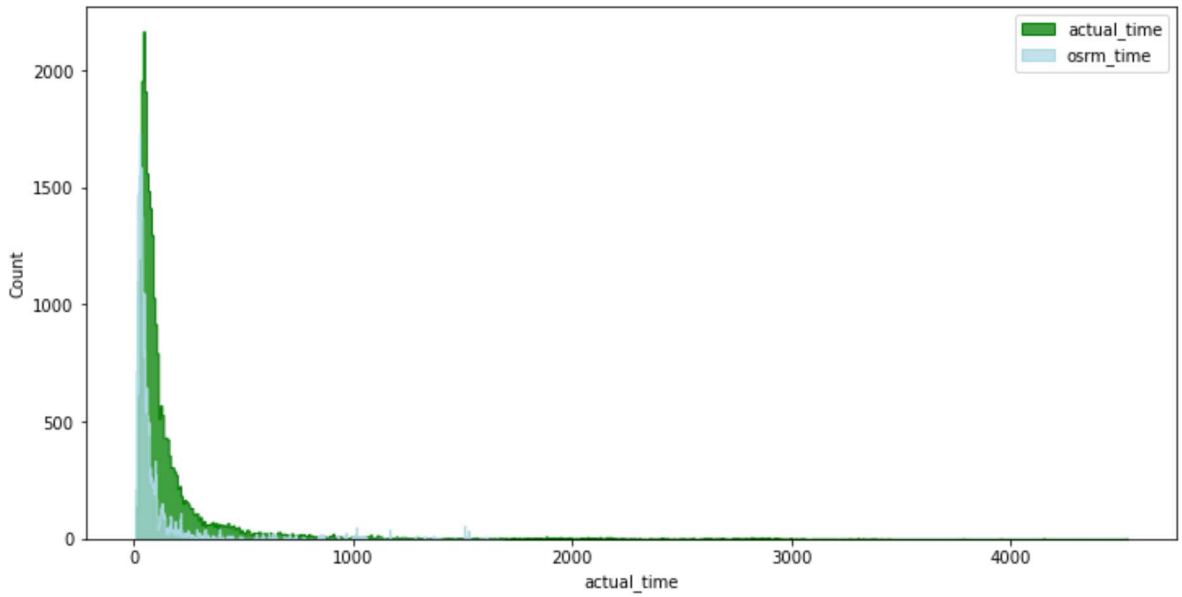
In [117]:

Out[117]:

	actual_time	osrm_time
count	26368.000000	26368.000000
mean	200.690186	90.686707
std	384.854095	185.079559
min	9.000000	6.000000
25%	51.000000	25.000000
50%	84.000000	39.000000
75%	168.000000	72.000000
max	4532.000000	1686.000000

In [118]:
```
plt.figure(figsize = (12, 6))
sns.histplot(seg_agg_data['actual_time'], element = 'step', color = 'green')
sns.histplot(seg_agg_data['osrm_time'], element = 'step', color = 'lightblue')
plt.legend(['actual_time', 'osrm_time'])
```

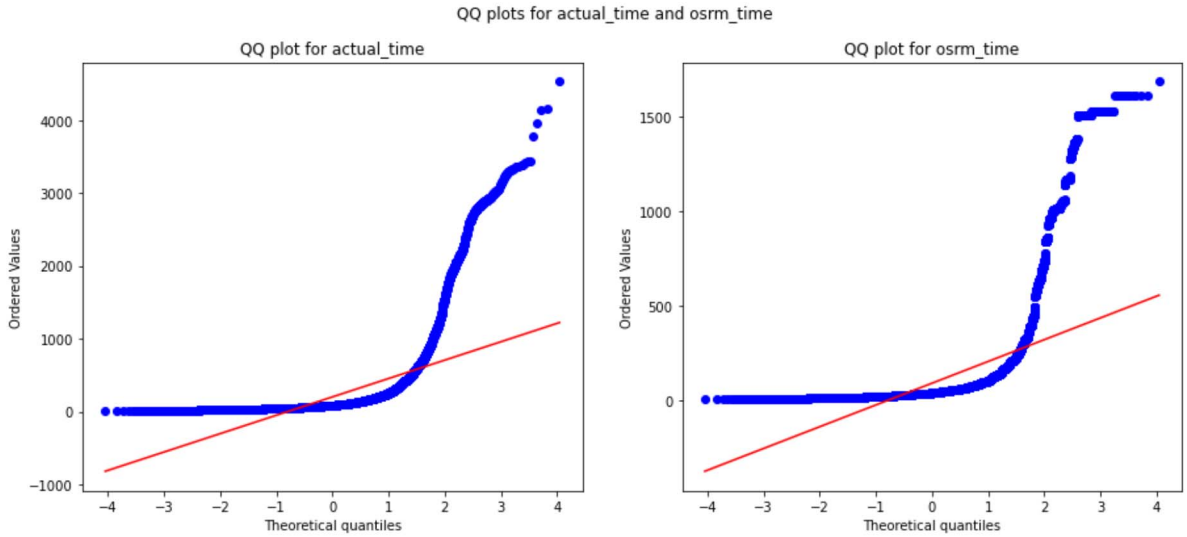
Out[118]: []



- Distribution check using **QQ Plot**

In [119]:
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and osrm_time')
spy.probplot(seg_agg_data['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(seg_agg_data['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
```

Out[119]: []



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

- Applying Shapiro-Wilk test for normality

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

alpha = 0.05

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

In [120]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

In [121]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['start_scan_to_end_scan'].sample
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

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

In [122]:
```
transformed_actual_time = spy.boxcox(seg_agg_data['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 3.599625466189942e-27
The sample does not follow normal distribution

C:\Program Files\Anaconda3\lib\site-packages\scipy\stats\morestats.py:1760: U
serWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

In [123]:
```
transformed_osrm_time = spy.boxcox(seg_agg_data['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 3.1461608973970853e-24
The sample does not follow normal distribution
```

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

- Homogeneity of Variances using **Lavene's test**

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 [124]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

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

```
p-value 1.6971827666500026e-230
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 [125]:
```
test_stat, p_value = spy.mannwhitneyu(seg_agg_data['actual_time'], seg_agg_dat
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
```

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

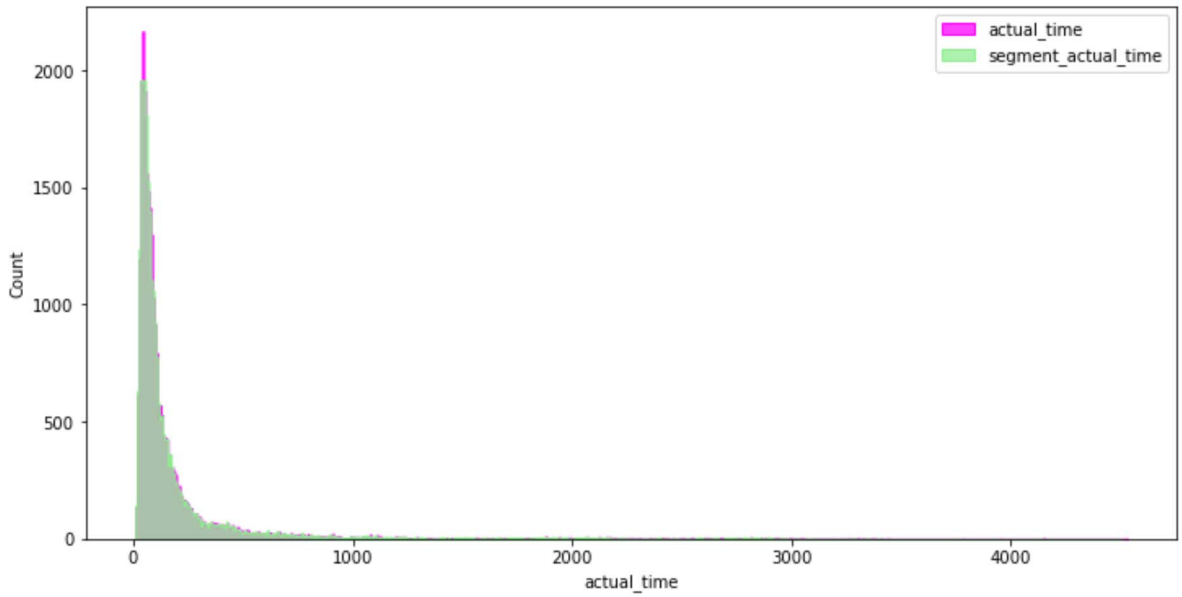
Since p-value < alpha therfore it can be concluded that actual_time and osrm_time are not similar.

Type *Markdown* and LaTeX: α^2

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 segment_key)

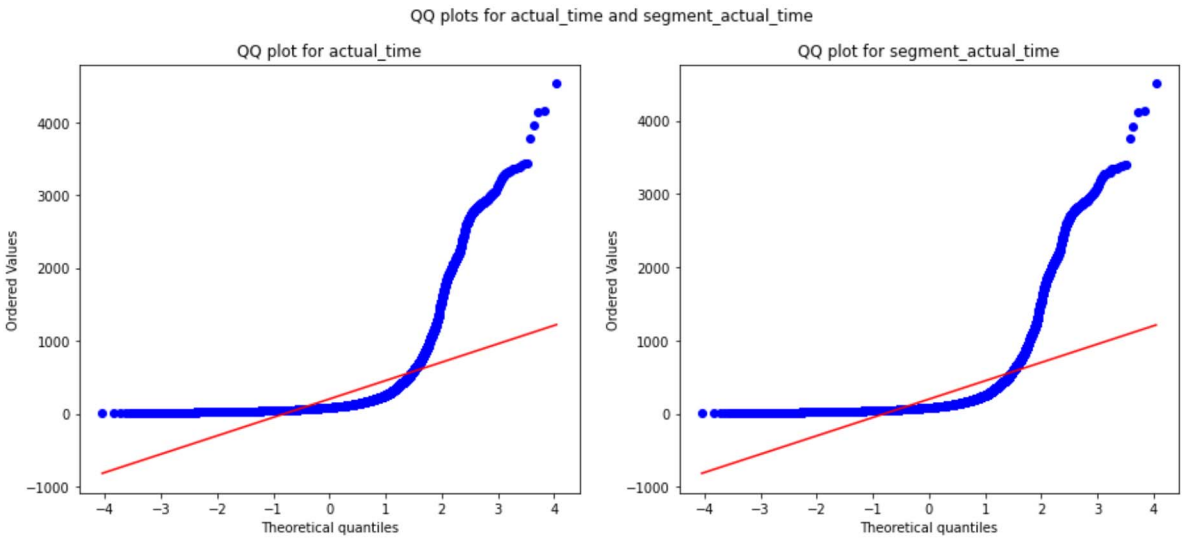
In [126]:
```
plt.figure(figsize = (12, 6))
sns.histplot(seg_agg_data['actual_time'], element = 'step', color = 'magenta')
sns.histplot(seg_agg_data['segment_actual_time'], element = 'step', color = 'l
plt.legend(['actual_time', 'segment_actual_time'])
```

Out[126]: []



In [127]:
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for actual_time and segment_actual_time')
spy.probplot(seg_agg_data['actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for actual_time')
plt.subplot(1, 2, 2)
spy.probplot(seg_agg_data['segment_actual_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_actual_time')
```

Out[127]: []



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

distribution.

- Applying Shapiro-Wilk test for normality

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

alpha = 0.05

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

In [128]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

In [129]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['segment_actual_time'].sample(50
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

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

In [130]:
```
transformed_actual_time = spy.boxcox(seg_agg_data['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 3.599625466189942e-27
The sample does not follow normal distribution

C:\Program Files\Anaconda3\lib\site-packages\scipy\stats\morestats.py:1760: U
serWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

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

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

- Homogeneity of Variances using **Lavene's test**

In [131]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

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

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

```
p-value 0.698556074726726
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 [132]:
```
test_stat, p_value = spy.mannwhitneyu(seg_agg_data['actual_time'], seg_agg_dat
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
```

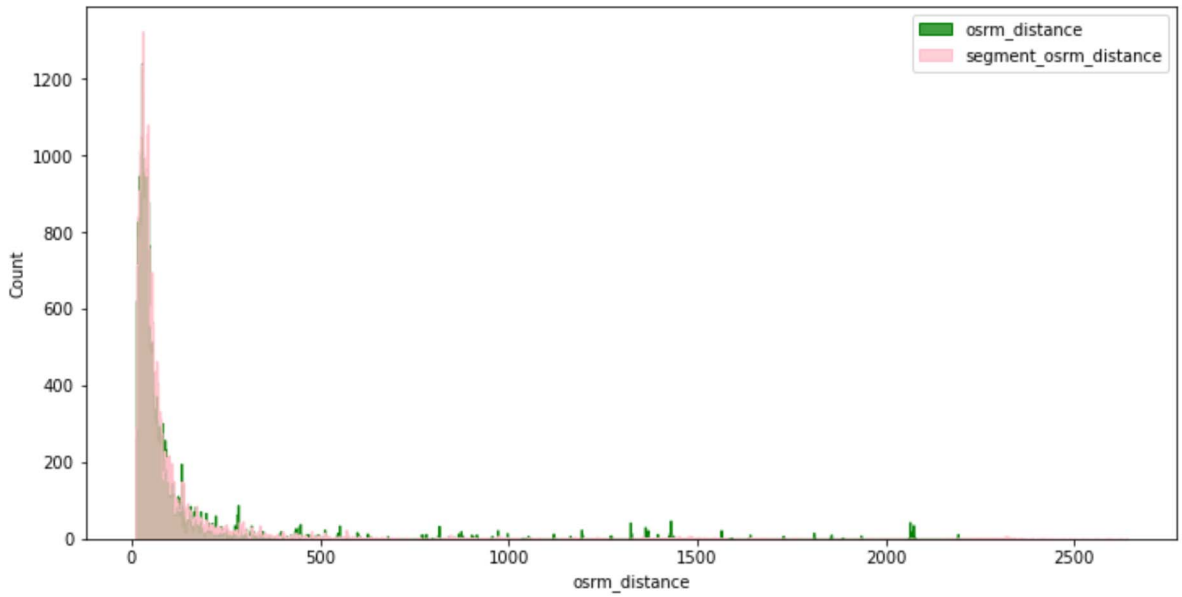
```
p-value 0.16011376950109157
The samples are similar
```

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

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

In [133]:
```
plt.figure(figsize = (12, 6))
sns.histplot(seg_agg_data['osrm_distance'], element = 'step', color = 'green',
sns.histplot(seg_agg_data['segment_osrm_distance'], element = 'step', color =
plt.legend(['osrm_distance', 'segment_osrm_distance'])
```

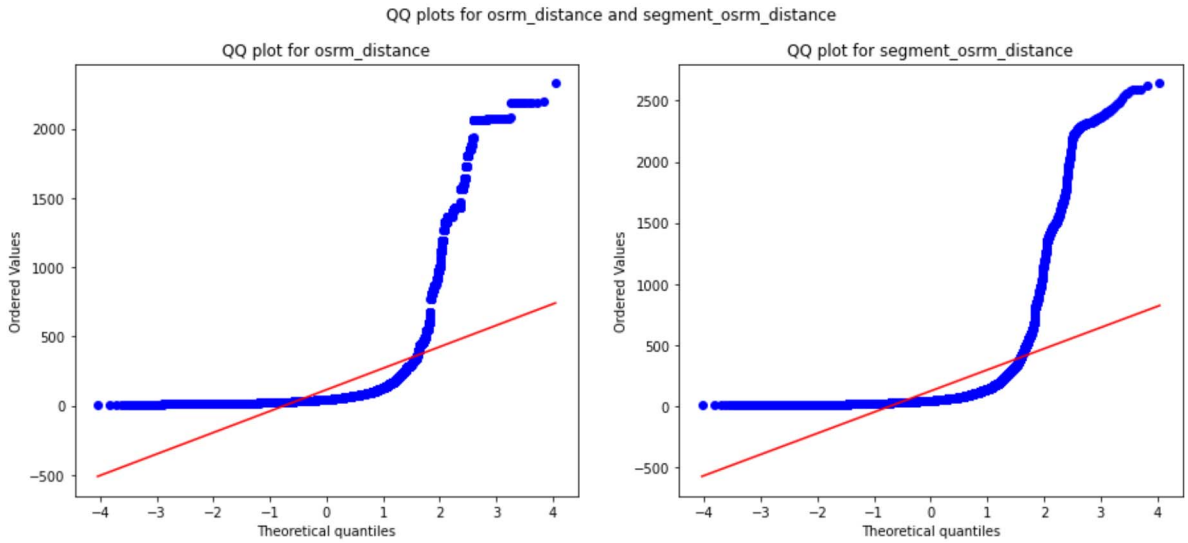
Out[133]: []



- Distribution check using **QQ Plot**

In [134]:
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_distance and segment_osrm_distance')
spy.probplot(seg_agg_data['osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_distance')
plt.subplot(1, 2, 2)
spy.probplot(seg_agg_data['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_distance')
```

Out[134]: []



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

- Applying Shapiro-Wilk test for normality

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

alpha = 0.05

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

In [135]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

In [136]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['segment_osrm_distance'].sample(
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

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

In [137]:
```
transformed_osrm_distance = spy.boxcox(seg_agg_data['osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 6.238992878797227e-29
The sample does not follow normal distribution

C:\Program Files\Anaconda3\lib\site-packages\scipy\stats\morestats.py:1760: U
serWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

In [138]:
```
transformed_segment_osrm_distance = spy.boxcox(seg_agg_data['segment_osrm_dist
test_stat, p_value = spy.shapiro(transformed_segment_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 5.01983769613568e-28
The sample does not follow normal distribution
```

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

- Homogeneity of Variances using **Lavene's test**

In [139]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(seg_agg_data['osrm_distance'], seg_agg_data['s
print('p-value', p_value)

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

```
p-value 3.006816339299108e-05
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 [140]:
```
test_stat, p_value = spy.mannwhitneyu(seg_agg_data['osrm_distance'], seg_agg_d
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
```

```
p-value 3.715809947282301e-10
The samples are not similar
```

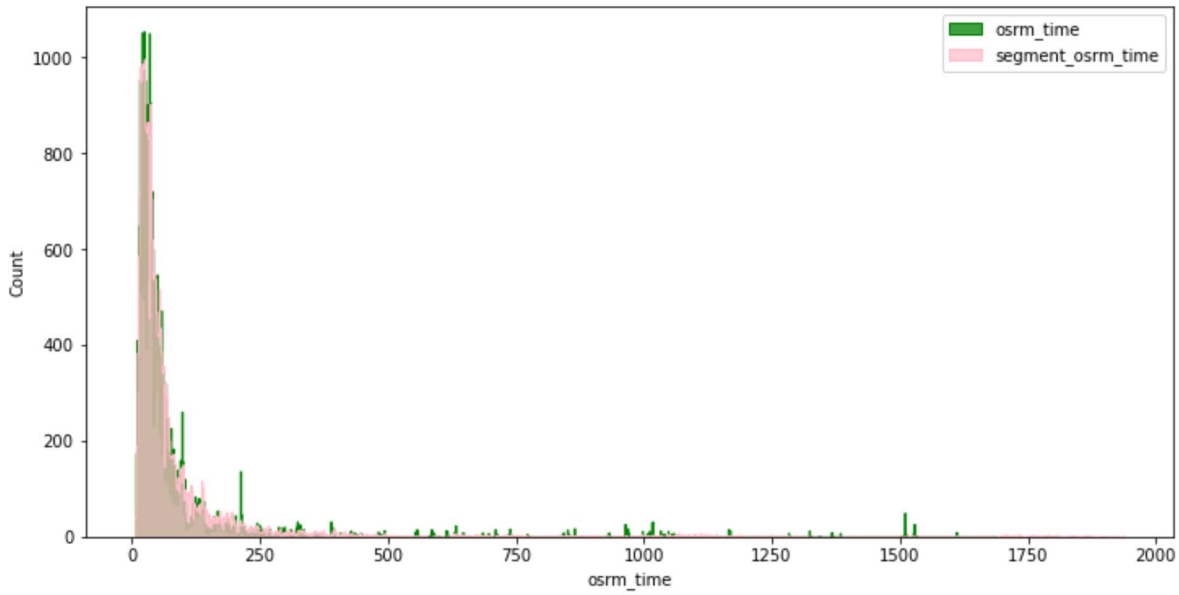
Since p-value < alpha therfore it can be concluded that osrm_distance and segment_osrm_distance are not similar.

In [ ]:

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

In [141]:
```
plt.figure(figsize = (12, 6))
sns.histplot(seg_agg_data['osrm_time'], element = 'step', color = 'green', bin
sns.histplot(seg_agg_data['segment_osrm_time'], element = 'step', color = 'pin
plt.legend(['osrm_time', 'segment_osrm_time'])
```

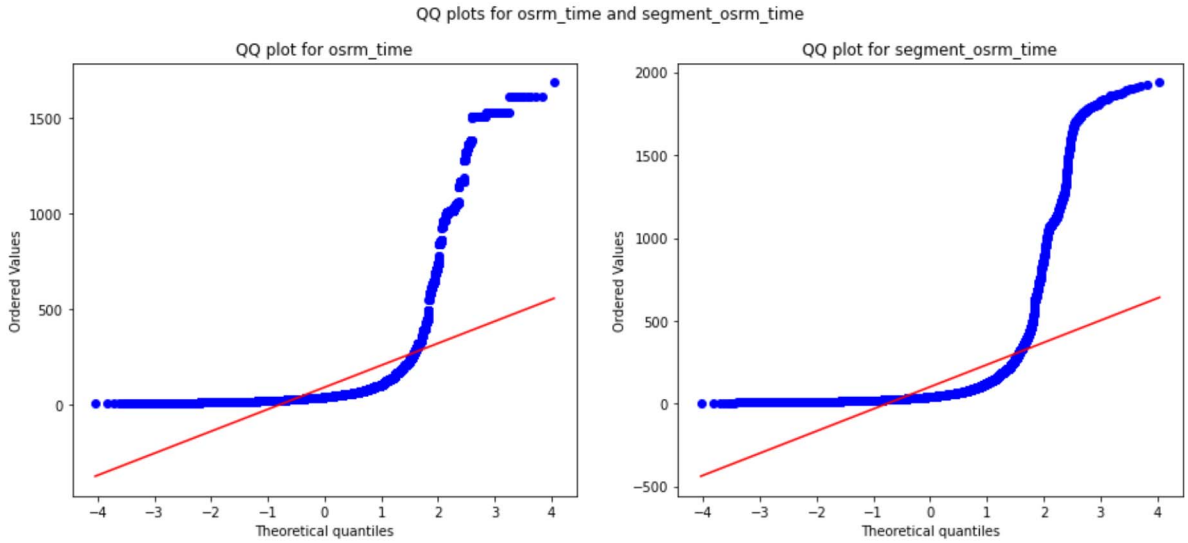
Out[141]: []



- Distribution check using **QQ Plot**

In [143]:
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for osrm_time and segment_osrm_time')
spy.probplot(seg_agg_data['osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for osrm_time')
plt.subplot(1, 2, 2)
spy.probplot(seg_agg_data['segment_osrm_time'], plot = plt, dist = 'norm')
plt.title('QQ plot for segment_osrm_time')
```

Out[143]: []



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

- Applying Shapiro-Wilk test for normality

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

alpha = 0.05

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

In [144]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

In [145]:
```
test_stat, p_value = spy.shapiro(seg_agg_data['segment_osrm_time'].sample(5000
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

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

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

In [146]:
```
transformed_osrm_time = spy.boxcox(seg_agg_data['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 3.1461608973970853e-24
The sample does not follow normal distribution

C:\Program Files\Anaconda3\lib\site-packages\scipy\stats\morestats.py:1760: U
serWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

In [147]:
```
transformed_segment_osrm_time = spy.boxcox(seg_agg_data['segment_osrm_time'])[
test_stat, p_value = spy.shapiro(transformed_segment_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
```

```
p-value 2.085097896474091e-23
The sample does not follow normal distribution
```

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

- Homogeneity of Variances using **Lavene's test**

In [148]:
```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(seg_agg_data['osrm_time'], seg_agg_data['segme
print('p-value', p_value)

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

```
p-value 1.3076142539440769e-09
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 [149]:
```
test_stat, p_value = spy.mannwhitneyu(seg_agg_data['osrm_time'], seg_agg_data[
print('p-value', p_value)
if p_value < 0.05:
    print('The samples are not similar')
else:
```

```
p-value 2.3417605478672297e-11
The samples are not similar
```

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

In [ ]:

Business Insights

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

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

Recommendations

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

In [ ]:

In [ ]: