```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import matplotlib as matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving delhivery_data.csv to delhivery_data.csv
```

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

```
df.head(5)
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.shape
```

```
(144867, 24)
```

- The dataset contains 144,867 rows and 24 columns

```
df.columns
```

```
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')
```

```
df.dtypes
```

```
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
```

```
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
```

- Memory Consumption: Approximately 25.6+ MB

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"start_scan_to_end_scan\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
50669.13363974827,\n        \"min\": 20.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
961.2629860492727,\n          449.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cutoff_factor\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51076.25962961759,\n        \"min\": 9.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
232.926567127089,\n          66.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"actual_distance_to_destination\",\
n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 51076.056012537076,\n        \"min\": 9.00004535977208,\n
\"max\": 144867.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          234.07337161811876,\n
66.12657071764777,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
50924.567951909936,\n        \"min\": 9.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
416.92752662787245,\n          132.0,\n          144867.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51091.78365158944,\n        \"min\": 6.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
213.86827227733025,\n          64.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51047.4793064438,\n        \"min\": 9.0082,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
284.7712965174954,\n          78.5258,\n          144867.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"factor\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 51213.829643415665,\n        \"min\": 0.144,\n
\"max\": 144867.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          2.120107191835053,\n
1.8571428571428568,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_actual_time\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51078.74557749331,\n        \"min\": -244.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
36.19611091552942,\n          29.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51135.69935115408,\n        \"min\": 0.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
18.507548302926132,\n          17.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51107.94242654291,\n        \"min\": 0.0,\n        \"max\": 144867.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
22.829019768477295,\n          23.513,\n          144867.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_factor\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 51190.17322939711,\n        \"min\": -
23.444444444444443,\n        \"max\": 144867.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
2.218368368955117,\n          1.6842105263157894,\n          144867.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


- The start_scan_to_end_scan column has a mean of 961.26 and a median of 449.00, indicating a positively skewed distribution.
- The cutoff_factor column has a mean of 232 and a median of 66, indicating a positively skewed distribution.

Dropping unknown fields

```
df.drop(columns=['is_cutoff', 'cutoff_factor', 'cutoff_timestamp',
'factor', 'segment_factor'], inplace=True) # removing unknown columns
```

- Dropping the columns is_cutoff, cutoff_factor, cutoff_timestamp, factor, and segment_factor as they are not important for our analysis.

Finding the unique values in each column

```
for i in df.columns:
    print(f"Unique entries in {i:<30} = {df[i].nunique()}")
```

```
Unique entries in data                          = 2
Unique entries in trip_creation_time            = 14817
Unique entries in route_schedule_uuid           = 1504
Unique entries in route_type                    = 2
Unique entries in trip_uuid                     = 14817
```

```
Unique entries in source_center                    = 1508
Unique entries in source_name                      = 1498
Unique entries in destination_center               = 1481
Unique entries in destination_name                 = 1468
Unique entries in od_start_time                    = 26369
Unique entries in od_end_time                      = 26369
Unique entries in start_scan_to_end_scan           = 1915
Unique entries in actual_distance_to_destination = 144515
Unique entries in actual_time                      = 3182
Unique entries in osrm_time                        = 1531
Unique entries in osrm_distance                    = 138046
Unique entries in segment_actual_time              = 747
Unique entries in segment_osrm_time                = 214
Unique entries in segment_osrm_distance            = 113799
```

###Converting Columns with Exactly Two Unique Entries to Categorical Data Type

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

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

```
1927
4532
1686
2326
3051
1611
2191
```

Utilizing float32 Datatype for Columns with Small Maximum Value Entries

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

Transitioning DateTime Columns to Appropriate Datatypes

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

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

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

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

- The timestamp range is from September 12, 2018 to October 8, 2018.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float32
 13  actual_time                     144867 non-null  float32
 14  osrm_time                       144867 non-null  float32
 15  osrm_distance                   144867 non-null  float32
 16  segment_actual_time             144867 non-null  float32
 17  segment_osrm_time               144867 non-null  float32
 18  segment_osrm_distance           144867 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(7), float64(1),
object(6)
memory usage: 15.2+ MB
```

- Dataset Memory Optimization: Reduced from 25.6+ MB to 15.2+ MB, Achieving a 40.63% Reduction.

Basic data cleaning and exploration

Detecting the missing values in the data

```
np.any(df.isnull())
```

```
True
```

```
df.isnull().sum()
```

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

- The 'source_name' column has 293 missing values, while the 'destination_name' column has 261 missing values.

Finding missing values in the data

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

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

```
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
```

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

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

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

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

```
np.all(df.loc[df['source_name'].isnull(),
'source_center'].isin(missing_destination_name)) #Checking if the IDs
for which the source name is similar to those IDs for destination name
```

```
False
```

Handling missing values in the data

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

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
```

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

Rechecking for the missing values

```
df.isna().sum()
```

```
data                        0
trip_creation_time          0
route_schedule_uuid         0
route_type                  0
trip_uuid                   0
source_center               0
source_name                 0
destination_center          0
destination_name            0
```

```
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
dtype: int64
```

Basic Descriptive Analysis of the Data

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n{\n      \"column\": \"trip_creation_time\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000144867\",\n        \"max\": \"2018-10-03 23:59:42.701692\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"144867\",\n          \"2018-09-22 13:34:23.659819264\",\n          \"2018-09-27 17:57:56.350054912\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"od_start_time\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000144867\",\n        \"max\": \"2018-10-06 04:27:23.392375\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"144867\",\n          \"2018-09-22 18:02:45.855230720\",\n          \"2018-09-27 22:41:50.285857024\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"od_end_time\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000144867\",\n        \"max\": \"2018-10-08 03:00:24.353479\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"144867\",\n          \"2018-09-23 10:04:31.395393024\",\n          \"2018-09-28 12:49:06.054018048\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"start_scan_to_end_scan\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 50669.13363974827,\n        \"min\": 20.0,\n        \"max\": 144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          961.2629860492727,\n          1634.0,\n          144867.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"actual_distance_to_destination\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 51076.05600901645,\n        \"min\": 9.000045776367188,\n        \"max\": 144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
```

```
234.07337951660156,\n          286.70887756347656,\n
144867.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 50924.567951369456,\n        \"min\":
9.0,\n        \"max\": 144867.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          416.9275207519531,\n          513.0,\n
144867.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 51091.7836512344,\n        \"min\":
6.0,\n        \"max\": 144867.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          213.8682861328125,\n          257.0,\n
144867.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 51047.47930057585,\n        \"min\":
9.008199691772461,\n        \"max\": 144867.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
284.77130126953125,\n          343.19325256347656,\n
144867.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 51078.74557767988,\n        \"min\": -
244.0,\n        \"max\": 144867.0,\n        \"num_unique_values\": 8,\
n        \"samples\": [\n          36.196109771728516,\n
40.0,\n          144867.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"segment_osrm_time\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 51135.69935118459,\n
\"min\": 0.0,\n        \"max\": 144867.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
18.50754737854004,\n          22.0,\n          144867.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51107.942421727195,\n        \"min\": 0.0,\n        \"max\":
144867.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
22.829017639160156,\n          27.813249588012695,\n
144867.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
df.describe(include = 'object')
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"route_schedule_uuid\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          1504,\n          \"1812\",\n
\"144867\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          14817,\n          \"101\",\n          \"144867\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"source_center\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1508,\n
\"23347\",\n          \"144867\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"source_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1508,\n
\"23347\",\n          \"144867\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"destination_center\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1481,\n
\"15192\",\n          \"144867\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"destination_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1481,\n
\"15192\",\n          \"144867\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Merging of rows and aggregation of fields

```
grouping_1 = ['trip_uuid', 'source_center', 'destination_center']
df1 = df.groupby(by = grouping_1, as_index = False).agg({'data' :
'first',

'route_type' : 'first',

'trip_creation_time' : 'first',
                                                        'source_name' :
'first',

'destination_name' : 'last',
                                                        'od_start_time'
: 'first',
                                                        'od_end_time' :
'first',

'start_scan_to_end_scan' : 'first',

'actual_distance_to_destination' : 'last',
                                                        'actual_time' :
'last',
```

```
                                                        'osrm_time' :
'last',
                                                        'osrm_distance'
: 'last',

'segment_actual_time' : 'sum',

'segment_osrm_time' : 'sum',

'segment_osrm_distance' : 'sum'})
df1
```


```
{"summary":"{\n  \"name\": \"df1\",\n  \"rows\": 26368,\n  \"fields\":
[\n    {\n      \"column\": \"trip_uuid\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 14817,\n
\"samples\": [\n          \"trip-153743386446319660\",\n
\"trip-153814691500548801\",\n          \"trip-153702499222437553\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"source_center\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1508,\n        \"samples\": [\n
\"IND853204AAA\",\n          \"IND177001AAA\",\n
\"IND132103AAB\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_center\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1481,\n
\"samples\": [\n          \"IND483501AAA\",\n
\"IND600044AAC\",\n          \"IND680587AAA\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"data\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"test\",\n          \"training\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"route_type\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Carting\",\n          \"FTL\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"trip_creation_time\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2018-09-12 00:00:16.535741\",\n        \"max\": \"2018-10-03
23:59:42.701692\",\n        \"num_unique_values\": 14817,\n
\"samples\": [\n          \"2018-09-20 08:57:44.463490\",\n
\"2018-09-28 15:01:55.005735\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"source_name\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 1508,\n
\"samples\": [\n          \"Naugchia_Vijayght_D (Bihar)\",\n
\"Hamirpur_GutmgrCl_D (Himachal Pradesh)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"destination_name\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1481,\n        \"samples\": [\n
\"Katni_Bargawan_DC (Madhya Pradesh)\",\n
\"Chennai_Chrompet_PC (Tamil Nadu)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"od_start_time\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2018-09-12 00:00:16.535741\",\n        \"max\": \"2018-10-06
04:27:23.392375\",\n        \"num_unique_values\": 26368,\n
\"samples\": [\n          \"2018-09-14 02:45:09.014063\",\n
\"2018-09-21 02:17:46.874554\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"od_end_time\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2018-09-12
00:50:10.814399\",\n        \"max\": \"2018-10-08 03:00:24.353479\",\n
\"num_unique_values\": 26368,\n        \"samples\": [\n
\"2018-09-14 03:46:48.236280\",\n          \"2018-09-21
03:49:05.713120\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_scan_to_end_scan\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 440.56158803319363,\n
\"min\": 20.0,\n        \"max\": 7898.0,\n
\"num_unique_values\": 1915,\n        \"samples\": [\n
1241.0,\n          244.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"actual_distance_to_destination\",\n      \"properties\":
{\n        \"dtype\": \"float32\",\n        \"num_unique_values\":
26332,\n        \"samples\": [\n          21.382129669189453,\n
15.667296409606934\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 1658,\n
\"samples\": [\n          1616.0,\n          893.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 560,\n        \"samples\": [\n
1168.0,\n          1326.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"osrm_distance\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 26015,\n
\"samples\": [\n          17.12849998474121,\n
67.9094009399414\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_actual_time\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 1676,\n
\"samples\": [\n          2345.0,\n          1653.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"segment_osrm_time\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 1102,\n        \"samples\": [\n
1114.0,\n          1689.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"segment_osrm_distance\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 26089,\n
\"samples\": [\n          840.5787963867188,\n
935.7714233398438\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df1"}
```


Feature Engineering

Calculating the Total time taken between od_start_time and od_end_time and keeping it as a feature column.

```
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()
```

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

```
df2 = df1.groupby(by = 'trip_uuid', as_index =
False).agg({'source_center' : 'first',

'destination_center' : 'last',
                                                                'data' :
'first',

'route_type' : 'first',

'trip_creation_time' : 'first',

'source_name' : 'first',

'destination_name' : 'last',

'od_total_time' : 'sum',

'start_scan_to_end_scan' : 'sum',
```

```
'actual_distance_to_destination' : 'sum',

'actual_time' : 'sum',
                                                                'osrm_time'
: 'sum',

'osrm_distance' : 'sum',

'segment_actual_time' : 'sum',

'segment_osrm_time' : 'sum',

'segment_osrm_distance' : 'sum'})
df2
```


```
{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 14817,\n  \"fields\":
[\n    {\n      \"column\": \"trip_uuid\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 14817,\n
\"samples\": [\n          \"trip-153743386446319660\",\n
\"trip-153814691500548801\",\n          \"trip-153702499222437553\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"source_center\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 938,\n        \"samples\": [\n
\"IND413517AAA\",\n          \"IND442001AAA\",\n
\"IND244001AAA\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"destination_center\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1042,\n
\"samples\": [\n          \"IND250611AAA\",\n
\"IND852118A1B\",\n          \"IND736101AAA\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"data\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"test\",\n          \"training\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"route_type\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Carting\",\n          \"FTL\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"trip_creation_time\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2018-09-12 00:00:16.535741\",\n        \"max\": \"2018-10-03
23:59:42.701692\",\n        \"num_unique_values\": 14817,\n
\"samples\": [\n          \"2018-09-20 08:57:44.463490\",\n
\"2018-09-28 15:01:55.005735\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"source_name\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 938,\n
```

```
\"samples\": [\n          \"Udgir_NlgaonRd_D (Maharashtra)\",\n
\"Wardha_RamaNgr_D (Maharashtra)\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"destination_name\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1042,\n        \"samples\": [\n
\"Baraut_SrnprHwy_D (Uttar Pradesh)\",\n          \"location_7\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"od_total_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
658.8682230803469,\n        \"min\": 23.46,\n        \"max\":
7898.55,\n        \"num_unique_values\": 13610,\n        \"samples\":
[\n          460.11,\n          96.37\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"start_scan_to_end_scan\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
658.7059569148138,\n        \"min\": 23.0,\n        \"max\": 7898.0,\n
\"num_unique_values\": 2208,\n        \"samples\": [\n
2245.0,\n          53.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"actual_distance_to_destination\",\n      \"properties\":
{\n        \"dtype\": \"float32\",\n        \"num_unique_values\":
14796,\n        \"samples\": [\n          19.363492965698242,\n
9.699400901794434\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 1853,\n
\"samples\": [\n          377.0,\n          242.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"osrm_time\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 817,\n        \"samples\": [\n          289.0,\
n          522.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_distance\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 14730,\n
\"samples\": [\n          36.480201721191406,\n
35.9359016418457\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_actual_time\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 1890,\n
\"samples\": [\n          2992.0,\n          1220.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_osrm_time\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 1242,\n        \"samples\": [\n
781.0,\n          392.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"segment_osrm_distance\",\n      \"properties\": {\n
```

```
\"dtype\": \"float32\",\n        \"num_unique_values\": 14752,\n
\"samples\": [\n          174.83889770507812,\n
67.8218994140625\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df2"}
```


###Extracting new features like City, Place, Code(State) from the Source Name

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

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]

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]

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

```
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)
```

```
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
```

```
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)

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

array(['Central_H_6', 'ChikaDPP_D', 'Bilaspur_HB', 'unknown_place',
'Dc',
       'Poonamallee', 'Chrompet_DPC', 'HBR Layout PC', 'Central_D_12',
       'Lajpat_IP', 'North_D_3', 'Balabhgarh_DPC', 'Central_DPP_3',
       'Shamshbd_H', 'Xroad_D', 'Nehrugnj_I', 'Central_I_7',
       'Central_H_1', 'Nangli_IP', 'North', 'KndliDPP_D',
'Central_D_9',
       'DavkharRd_D', 'Bandel_D', 'RTCStand_D', 'Central_DPP_1',
       'KGAirprt_HB', 'North_D_2', 'Central_D_1', 'DC', 'Mthurard_L',
       'Mullanpr_DC', 'Central_DPP_2', 'RajCmplx_D', 'Beliaghata_DPC',
       'RjnaiDPP_D', 'AbbasNgr_I', 'Mankoli_HB', 'DPC', 'Airport_H',
       'Hub', 'Gateway_HB', 'Tathawde_H', 'ChotiHvl_DC', 'Trmltmpl_D',
       'OnkarDPP_D', 'Mehmdpur_H', 'KaranNGR_D', 'Sohagpur_D',
       'Chrompet_L', 'Busstand_D', 'Central_I_1', 'IndEstat_I',
'Court_D',
       'Panchot_IP', 'Adhartal_IP', 'DumDum_DPC', 'Bomsndra_HB',
       'Swamylyt_D', 'Yadvgiri_IP', 'Old', 'Kundli_H', 'Central_I_3',
       'Vasanthm_I', 'Poonamallee_HB', 'VUNagar_DC', 'NlgaonRd_D',
       'Bnnrghta_L', 'Thirumtr_IP', 'GariDPP_D', 'Jogshwri_I',
       'KoilStrt_D', 'CotnGren_M', 'Nzbadrd_D', 'Dwaraka_D',
'Nelmngla_H',
       'NvygRDPP_D', 'Gndhichk_D', 'Central_D_3', 'Chowk_D',
'CharRsta_D',
       'Kollgpra_D', 'Peenya_IP', 'GndhiNgr_IP', 'Sanpada_I',
       'WrdN4DPP_D', 'Sakinaka_RP', 'CivilHPL_D', 'OstwlEmp_D',
       'Gajuwaka', 'Mhbhirab_D', 'MGRoad_D', 'Balajicly_I',
'BljiMrkt_D',
       'Dankuni_HB', 'Trnsport_H', 'Rakhial', 'Memnagar', 'East_I_21',
       'Mithakal_D'], dtype=object)
```

###Extracting new features like City, Place, Code(State) from the Destination Name

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

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
```

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

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

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

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

###Extracting new features like Month, Year and Day from the Trip_creation_time

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

```
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]
```

```
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()
```

```
0    12
1    12
2    12
3    12
```

```
4    12
Name: trip_creation_day, dtype: int8
```

```
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()
```

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

```
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()
```

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

```
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()
```

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

- All ride information in the dataset is exclusively from the 37th week of the year 2018 or From 12th September 2018.

##Assessing Data Structure Post Data Cleaning

```
df2.shape
```

```
(14817, 29)
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 29 columns):
```

```
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14817 non-null  object
 1   source_center                   14817 non-null  object
 2   destination_center              14817 non-null  object
 3   data                            14817 non-null  category
 4   route_type                      14817 non-null  category
 5   trip_creation_time              14817 non-null  datetime64[ns]
 6   source_name                     14817 non-null  object
 7   destination_name                14817 non-null  object
 8   od_total_time                   14817 non-null  float64
 9   start_scan_to_end_scan          14817 non-null  float64
 10  actual_distance_to_destination  14817 non-null  float32
 11  actual_time                     14817 non-null  float32
 12  osrm_time                       14817 non-null  float32
 13  osrm_distance                   14817 non-null  float32
 14  segment_actual_time             14817 non-null  float32
 15  segment_osrm_time               14817 non-null  float32
 16  segment_osrm_distance           14817 non-null  float32
 17  source_state                    14817 non-null  object
 18  source_city                     14817 non-null  object
 19  source_place                    14817 non-null  object
 20  destination_state               14817 non-null  object
 21  destination_city                14817 non-null  object
 22  destination_place               14817 non-null  object
 23  trip_creation_date              14817 non-null  datetime64[ns]
 24  trip_creation_day               14817 non-null  int8
 25  trip_creation_month             14817 non-null  int8
 26  trip_creation_year              14817 non-null  int16
 27  trip_creation_week              14817 non-null  int8
 28  trip_creation_hour              14817 non-null  int8
dtypes: category(2), datetime64[ns](2), float32(7), float64(2),
int16(1), int8(4), object(11)
memory usage: 2.2+ MB
```

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


```
{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 16,\n  \"fields\": [\
n    {\n      \"column\": \"count\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"14817\",\n        \"max\":
\"14817\",\n        \"num_unique_values\": 1,\n        \"samples\": [\
n          \"14817\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mean\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": \"1970-01-01 00:00:00.000000009\",\n        \"max\": \"2018-
09-22 12:44:19.555167744\",\n        \"num_unique_values\": 16,\n
\"samples\": [\n          \"2018-09-22 12:44:19.555167744\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"min\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
```

\"min\": \"1970-01-01 00:00:00\",\n   \"max\": \"2018-09-12
00:00:16.535741\",\n   \"num_unique_values\": 12,\n
\"samples\": [\n   37.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"25%\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000000004\",\n
\"max\": \"2018-09-17 02:51:25.129125888\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   \"2018-
09-17 02:51:25.129125888\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"50%\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000000009\",\n
\"max\": \"2018-09-22 04:02:35.066945024\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   \"2018-
09-22 04:02:35.066945024\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"75%\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000000009\",\n
\"max\": \"2018-09-27 19:37:41.898427904\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   \"2018-
09-27 19:37:41.898427904\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"max\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000000010\",\n
\"max\": \"2018-10-03 23:59:42.701692\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   \"2018-
10-03 23:59:42.701692\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"std\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": 0.0,\n   \"max\": 658.8682230803469,\
n   \"num_unique_values\": 14,\n   \"samples\": [\n
7.8932746667826805\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

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

{"summary":"{\n \"name\": \"df2\",\n \"rows\": 11,\n \"fields\": [\
n   {\n   \"column\": \"count\",\n   \"properties\": {\n
\"dtype\": \"date\",\n   \"min\": \"14817\",\n   \"max\":
\"14817\",\n   \"num_unique_values\": 1,\n   \"samples\": [\
n   \"14817\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"unique\",\n   \"properties\": {\n   \"dtype\": \"date\",\n
\"min\": 34,\n   \"max\": 14817,\n   \"num_unique_values\":
9,\n   \"samples\": [\n   806\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"top\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 6,\n
\"samples\": [\n   \"trip-1536710416535487 48\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"freq\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"1\",\n        \"max\":
\"2714\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n
\"1\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Insights

- Trip UUID Uniqueness: Each trip in the dataset has a unique identifier, with a total count of 14,817 trips.
- Center Diversity: Both source and destination centers exhibit a high diversity, with 938 unique centers for the source and 1,042 unique centers for the destination. The most frequent center for both source and destination is 'IND000000ACB', occurring 1,063 and 821 times respectively.
- Location Consistency: Despite the diversity in centers, the most common source and destination names are 'Gurgaon_Bilaspur_HB (Haryana)' for both, suggesting a pattern of repeated trips between these locations.
- State and City Distribution: The data covers trips from 34 unique states for the source and 39 for the destination, with the highest frequency in Maharashtra for both. Mumbai is the most frequent city for both the source and destination, appearing 1,442 and 1,548 times respectively.
- Place Information: The most common source and destination places are 'Bilaspur_HB', with 1,063 and 821 occurrences respectively, indicating a significant portion of trips involving this location.

Analyzing Trip Creation Frequency on an Hourly Basis

```
df2['trip_creation_hour'].unique()
```

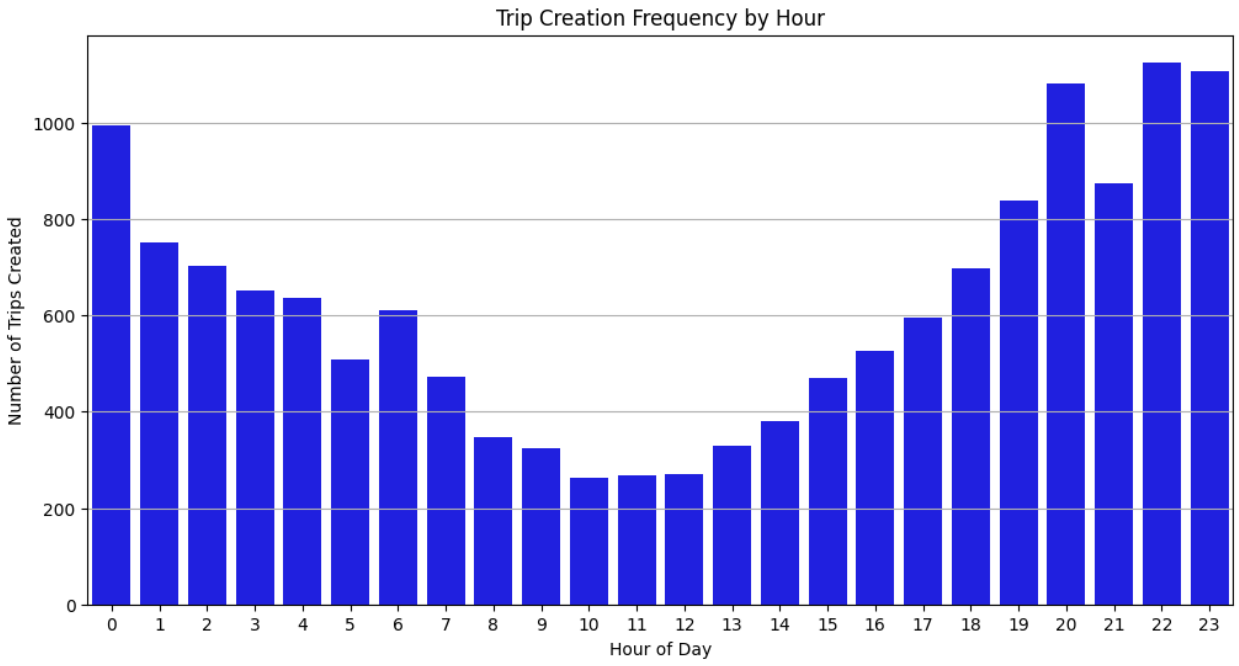
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19, 20, 21, 22, 23], dtype=int8)
```

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


```
{"summary":"{\n  \"name\": \"df_hour\",\n  \"rows\": 24,\n
\"fields\": [\n    {\n      \"column\": \"trip_creation_hour\",\n
\"properties\": {\n        \"dtype\": \"int8\",\n
\"num_unique_values\": 24,\n        \"samples\": [\n          8,\n
16,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
```

\"number\",\n   \"std\": 274,\n   \"min\": 262,\n
\"max\": 1125,\n   \"num_unique_values\": 24,\n
\"samples\": [\n   346,\n   526,\n   994\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"df_hour"}

```
plt.figure(figsize=(12, 6))
sns.barplot(data=df_hour,
            x='trip_creation_hour',
            y='trip_uuid',
            color='blue')
plt.xticks(np.arange(0, 24))
plt.grid(axis='y')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Trips Created')
plt.title('Trip Creation Frequency by Hour')
plt.show()
```



Insights

- Peak Hours: The busiest hours for trip creation are observed in the evening, with a gradual increase starting from around 12 PM and reaching a peak at 10 PM.
- Evening Surge: The increase in trip creation after 12 PM suggests a surge in activity, possibly indicating higher demand or activity during the evening hours.

- Nighttime Activity: The sustained high number of trips created until 10 PM highlights significant activity during the nighttime hours, possibly reflecting late-night travel or other activities.
- Decrease Post-Peak: Following the peak at 12 AM, there is a decrease in trip creation, indicating a decline in activity as the night progresses.

Analyzing Trip Creation Frequency Across Different Days of the Month

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

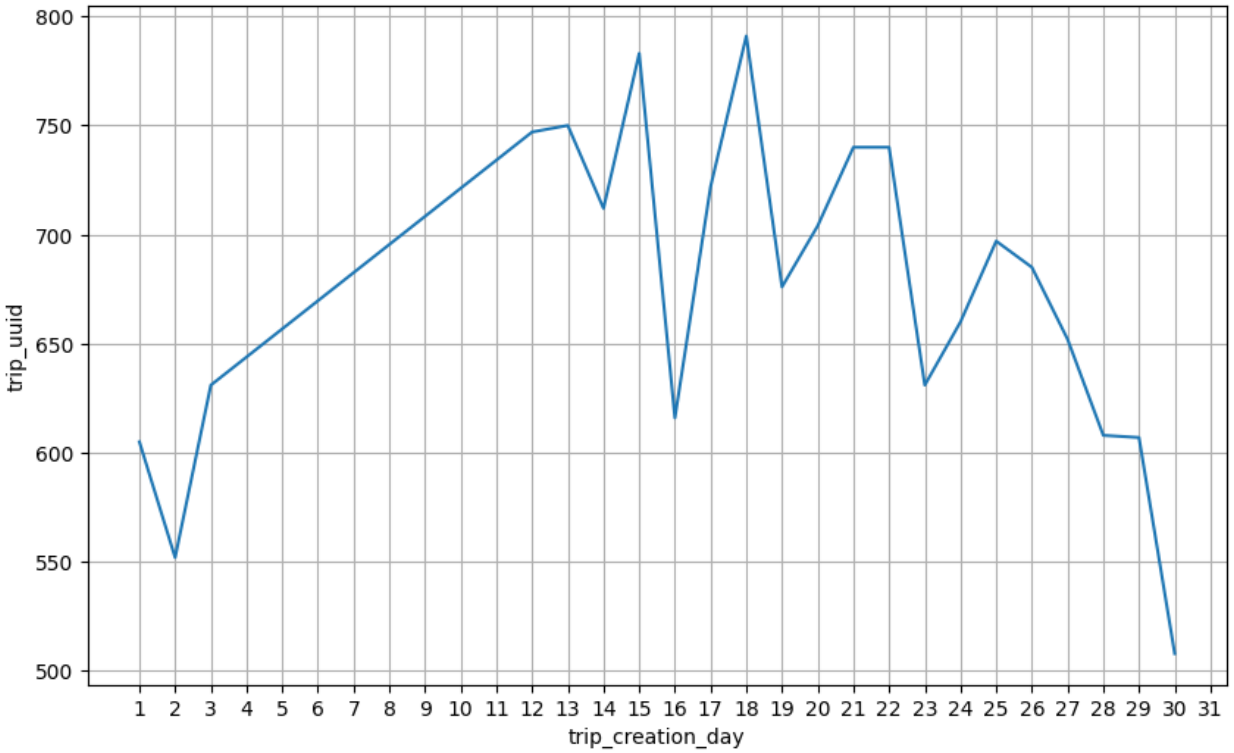
```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28,
       29, 30,  1,  2,  3], dtype=int8)
```

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

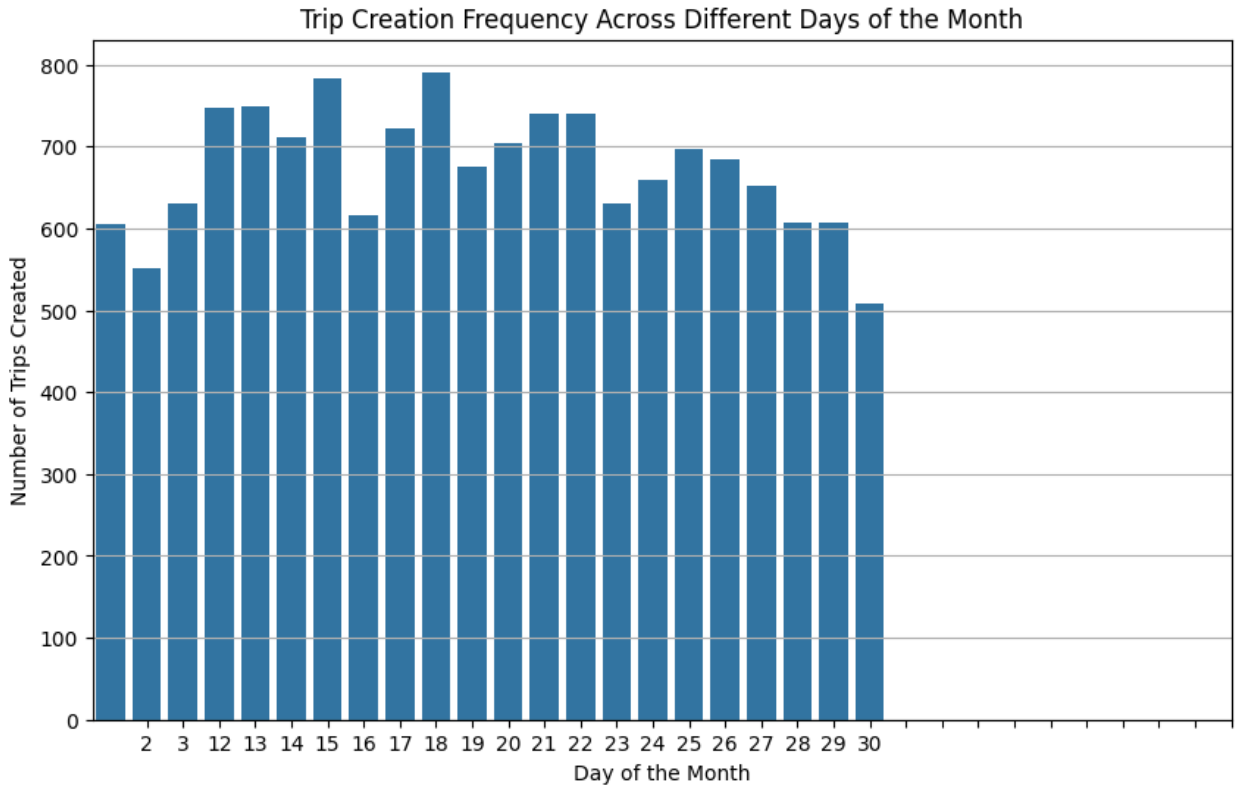

```
{"summary":"{\n  \"name\": \"df_day\",\n  \"rows\": 22,\n  \"fields\":
[\n    {\n      \"column\": \"trip_creation_day\",\n
\"properties\": {\n        \"dtype\": \"int8\",\n
\"num_unique_values\": 22,\n        \"samples\": [\n          1,\n
22,\n          17\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 73,\n        \"min\": 508,\n
\"max\": 791,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          605,\n          608,\n          685\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df_day"}
```


```
plt.figure(figsize = (10, 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()
```

```
[]
```




```
plt.figure(figsize=(10, 6))
sns.barplot(data=df_day,
            x=df_day['trip_creation_day'],
            y=df_day['trip_uuid'])
plt.xticks(np.arange(1, 32))
plt.grid(axis='y')
plt.xlabel('Day of the Month')
plt.ylabel('Number of Trips Created')
plt.title('Trip Creation Frequency Across Different Days of the
Month')
plt.show()
```



- Observing the plot, it's evident that the majority of trips are created around the middle of the month, suggesting a tendency for customers to place more orders during 12th to 22nd day of the month.

###Analyzing Trip Creation Frequency Across Weekly Intervals

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

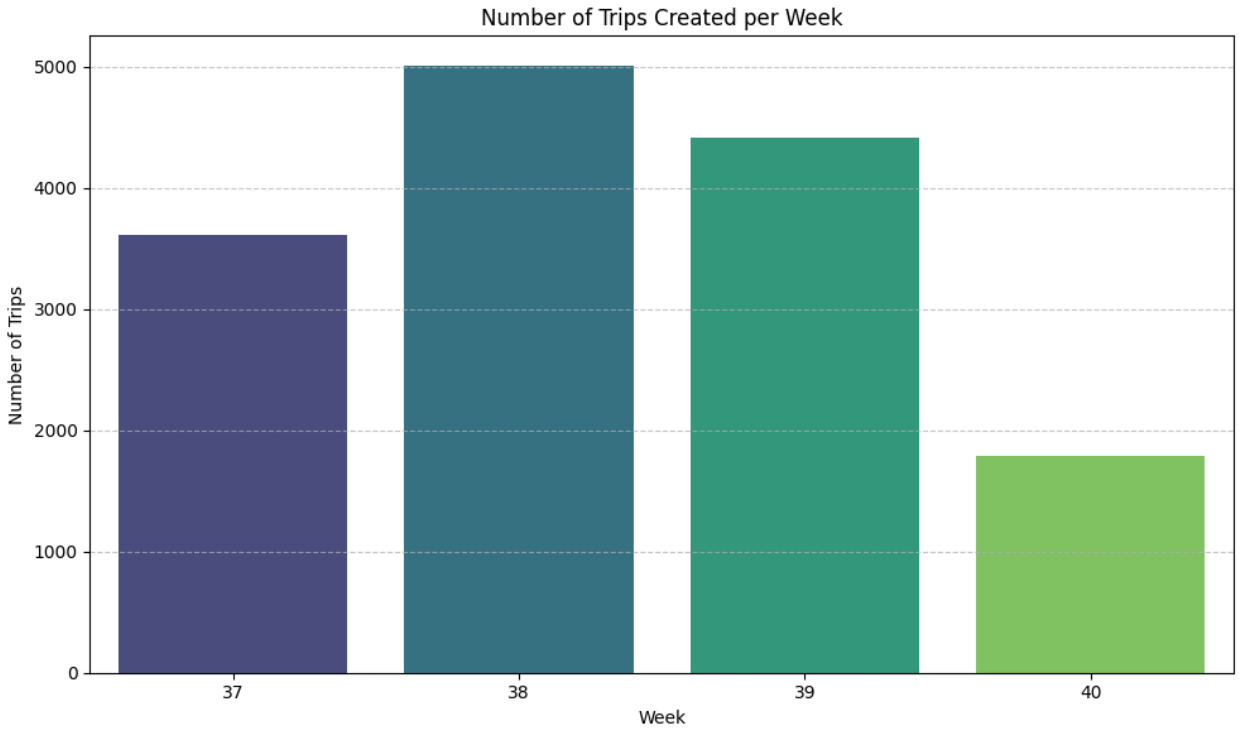
```
array([37, 38, 39, 40], dtype=int8)
```

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


```
{"summary":"{\n  \"name\": \"df_week\",\n  \"rows\": 4,\n  \"fields\":
[\n    {\n      \"column\": \"trip_creation_week\",\n
\"properties\": {\n        \"dtype\": \"int8\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          38,\n
40,\n          37\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1399,\n        \"min\": 1788,\n
\"max\": 5004,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          5004,\n          1788,\n          3608\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df_week"}
```

```
plt.figure(figsize=(10, 6))
sns.barplot(data=df_week,
            x=df_week['trip_creation_week'],
            y=df_week['trip_uuid'],
            ci=None,
            palette='viridis')
plt.xlabel('Week')
plt.ylabel('Number of Trips')
plt.title('Number of Trips Created per Week')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
<ipython-input-107-76fe87b0ff5a>:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  sns.barplot(data=df_week,
<ipython-input-107-76fe87b0ff5a>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=df_week,
```



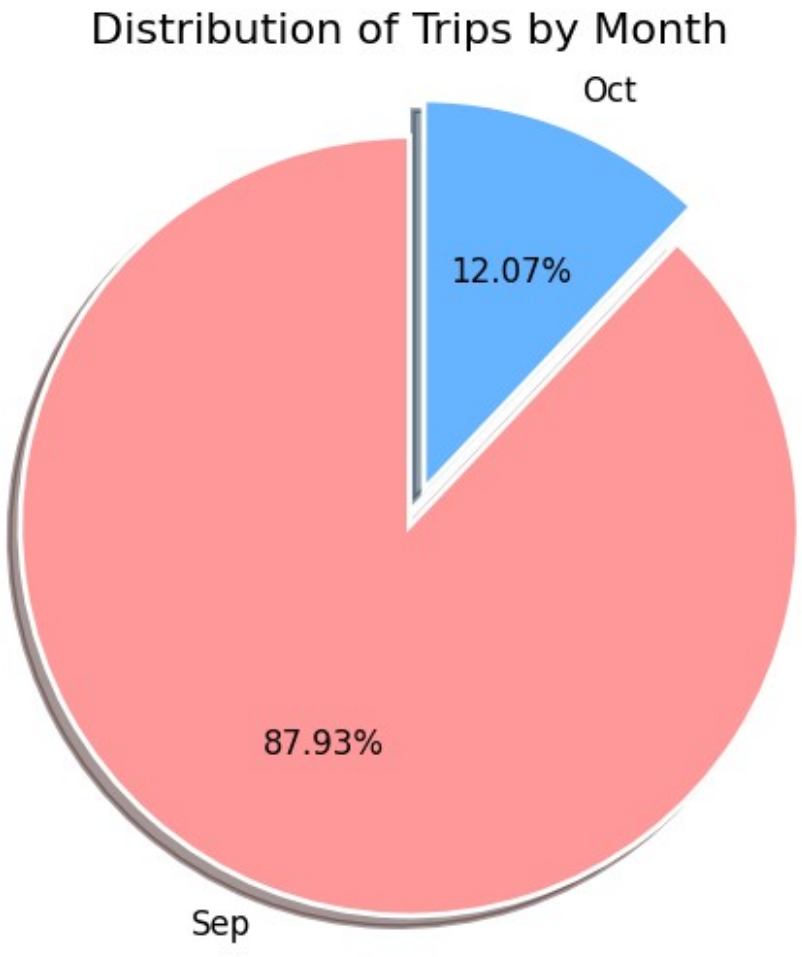
- The data reveals that the majority of trips were created during the 38th week followed by 39th week, indicating a notable peak in trip creation activity during this period.

###Analyzing Trip Creation Frequency Across the given two months Intervals

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


```
{"summary":"{\n  \"name\": \"df_month\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"trip_creation_month\",\n
\"properties\": {\n        \"dtype\": \"int8\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          10,\n
9\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7948,\n        \"min\": 1788,\n
\"max\": 13029,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1788,\n          13029\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"perc\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 53.6411204208115,\n
\"min\": 12.07,\n        \"max\": 87.93,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          12.07,\n
87.93\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_month"}
```


```
colors = ['#ff9999','#66b3ff']
plt.figure(figsize=(6, 6))
plt.pie(x=df_month['trip_uuid'],
        labels=['Sep', 'Oct'],
        explode=[0, 0.1],
        autopct='%.2f%%',
        colors=colors,
        shadow=True,
        startangle=90,
        textprops={'fontsize': 12},
        wedgeprops={'linewidth': 2, 'edgecolor': 'white'})
plt.title('Distribution of Trips by Month', fontsize=16)
plt.axis('equal')
plt.show()
```



Insights

- In September, there were 13,029 trips created, constituting approximately 87.93% of the total trips.
- In October, there were 1,788 trips created, accounting for approximately 12.07% of the total trips.

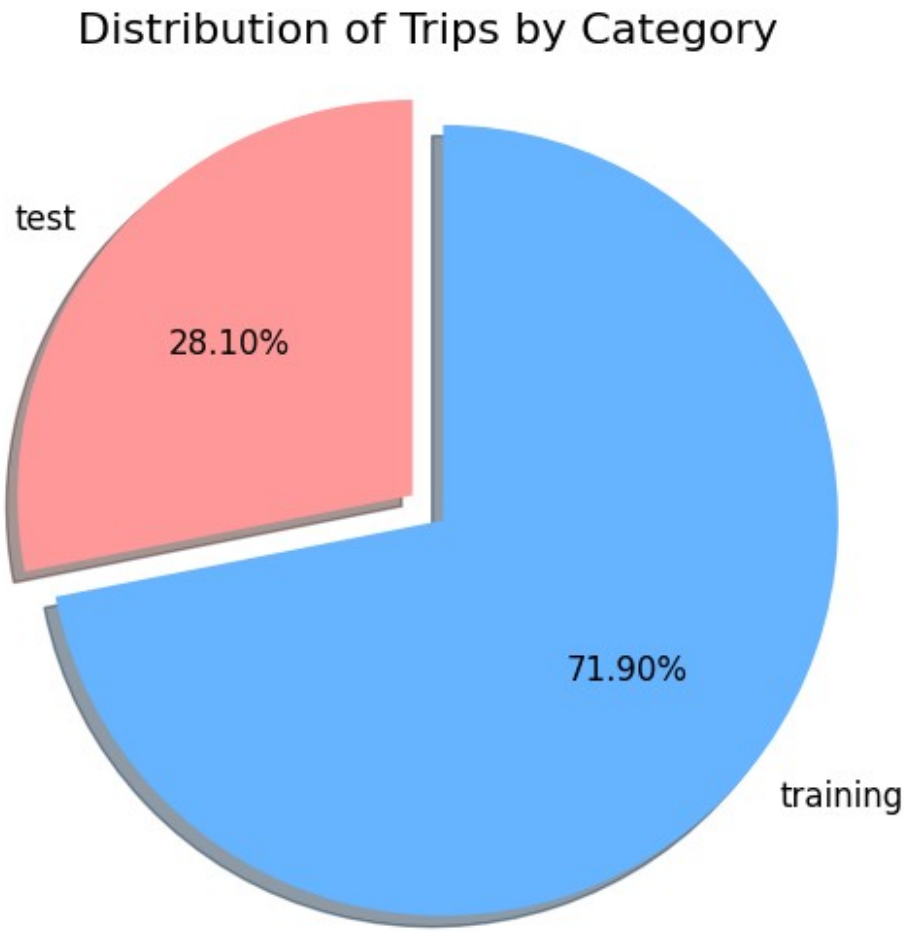
###Analyzing the distribution of trip for the given two months Intervals

```
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()
```


```
{"summary":"{\n  \"name\": \"df_data\",\n  \"rows\": 2,\n  \"fields\":
[\n    {\n      \"column\": \"data\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"training\",\n          \"test\"\
```

```
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4589,\n        \"min\": 4163,\n
\"max\": 10654,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          10654,\n          4163\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"perc\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.971277015970784,\n
\"min\": 28.1,\n        \"max\": 71.9,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          71.9,\n          28.1\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_data"}
```


```
plt.figure(figsize=(6, 6))
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']
plt.pie(x=df_data['trip_uuid'],
        labels=df_data['data'],
        explode=[0, 0.1],
        autopct='%.2f%%',
        colors=colors,
        startangle=90,
        shadow=True,
        textprops={'fontsize': 12})
plt.axis('equal')
plt.title('Distribution of Trips by Category', fontsize=16)
plt.show()
```



Insights

- The majority of trips are utilized for training purposes, as evidenced by the higher percentage of trips in the "training" dataset.
- The "training" dataset contains a significantly larger number of trips compared to the "test" dataset, with approximately 2.5 times as many trips.

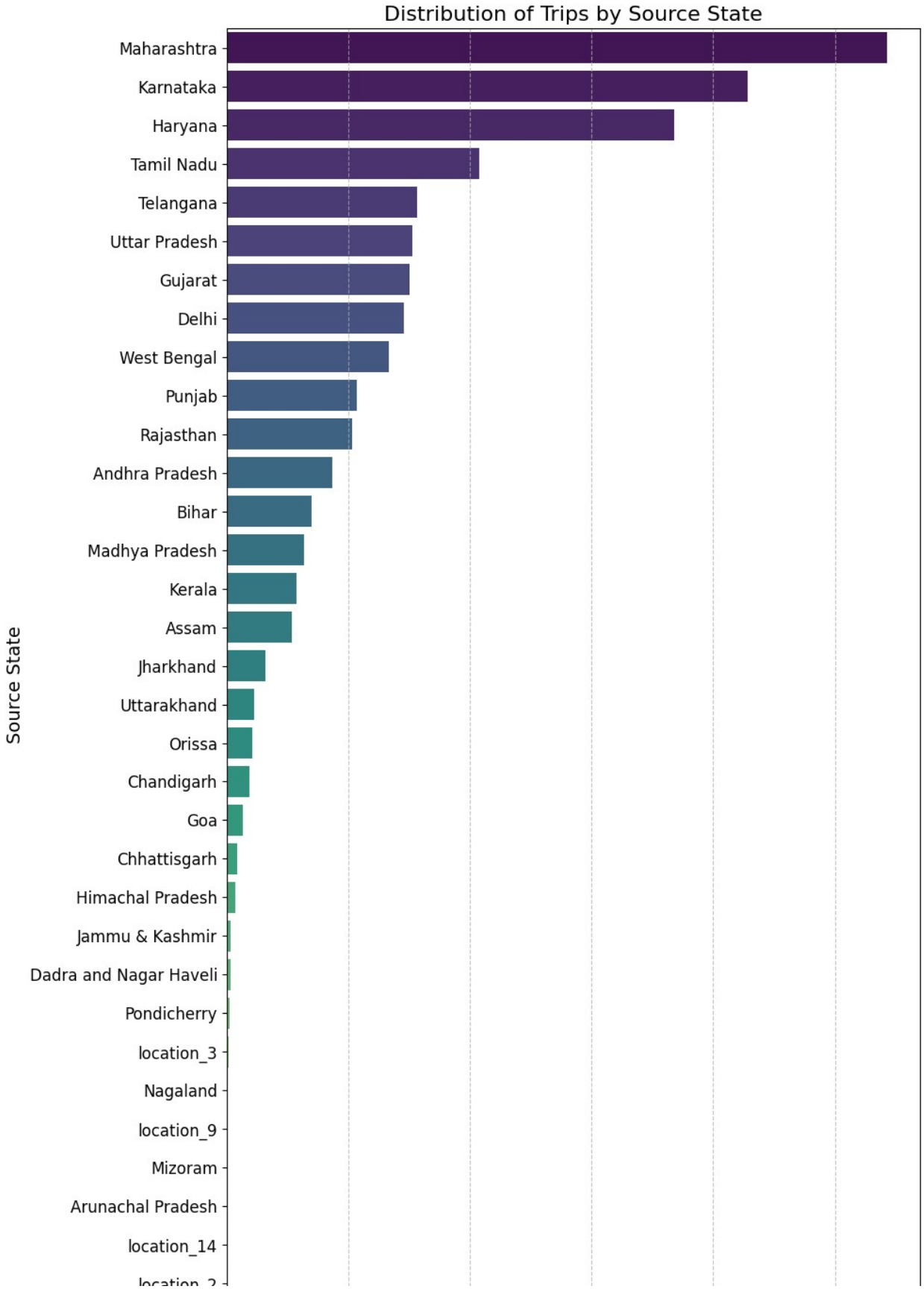
Analyzing Trip Distribution by Source State

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

{"summary":"{\n \"name\": \"df_source_state\",\n \"rows\": 34,\n \"fields\": [\n   {\n   \"column\": \"source_state\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 34,\n   \"samples\": [\n   \"Assam\",\n   \"Chandigarh\",\n   \"Nagaland\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"trip_uuid\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 648,\n   \"min\": 1,\n   \"max\": 2714,\n   \"num_unique_values\": 30,\n   \"samples\": [\n   5,\n   268,\n   17\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"perc\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4.374920513665507,\n   \"min\": 0.01,\n   \"max\": 18.32,\n   \"num_unique_values\": 29,\n   \"samples\": [\n   0.03,\n   1.08,\n   2.36\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df_source_state"}

```
plt.figure(figsize=(10, 15))

sns.barplot(data=df_source_state,
            x='trip_uuid',  # Set the x-axis to 'trip_uuid'
            y='source_state',  # Set the y-axis to 'source_state'
            palette='viridis')  # Choose a color palette

# Add labels and title
plt.xlabel('Number of Trips', fontsize=14)
plt.ylabel('Source State', fontsize=14)
plt.title('Distribution of Trips by Source State', fontsize=16)

# Adjust tick label font size
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add grid lines
plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

```
<ipython-input-118-09c82937f140>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=df_source_state,
```


Distribution of Trips by Source State
Source State
Maharashtra
Karnataka
Haryana
Tamil Nadu
Telangana
Uttar Pradesh
Gujarat
Delhi
West Bengal
Punjab
Rajasthan
Andhra Pradesh
Bihar
Madhya Pradesh
Kerala
Assam
Jharkhand
Uttarakhand
Orissa
Chandigarh
Goa
Chhattisgarh
Himachal Pradesh
Jammu & Kashmir
Dadra and Nagar Haveli
Pondicherry
location_3
Nagaland
location_9
Mizoram
Arunachal Pradesh
location_14
location_2

Insights

- The majority of trips originated from Maharashtra, followed by Karnataka and Haryana. This suggests a robust seller presence in these states, indicating potential areas of market strength and opportunity for further business development.

Exploring the Top 10 Cities by Trip Creation Volume Across Different Cities

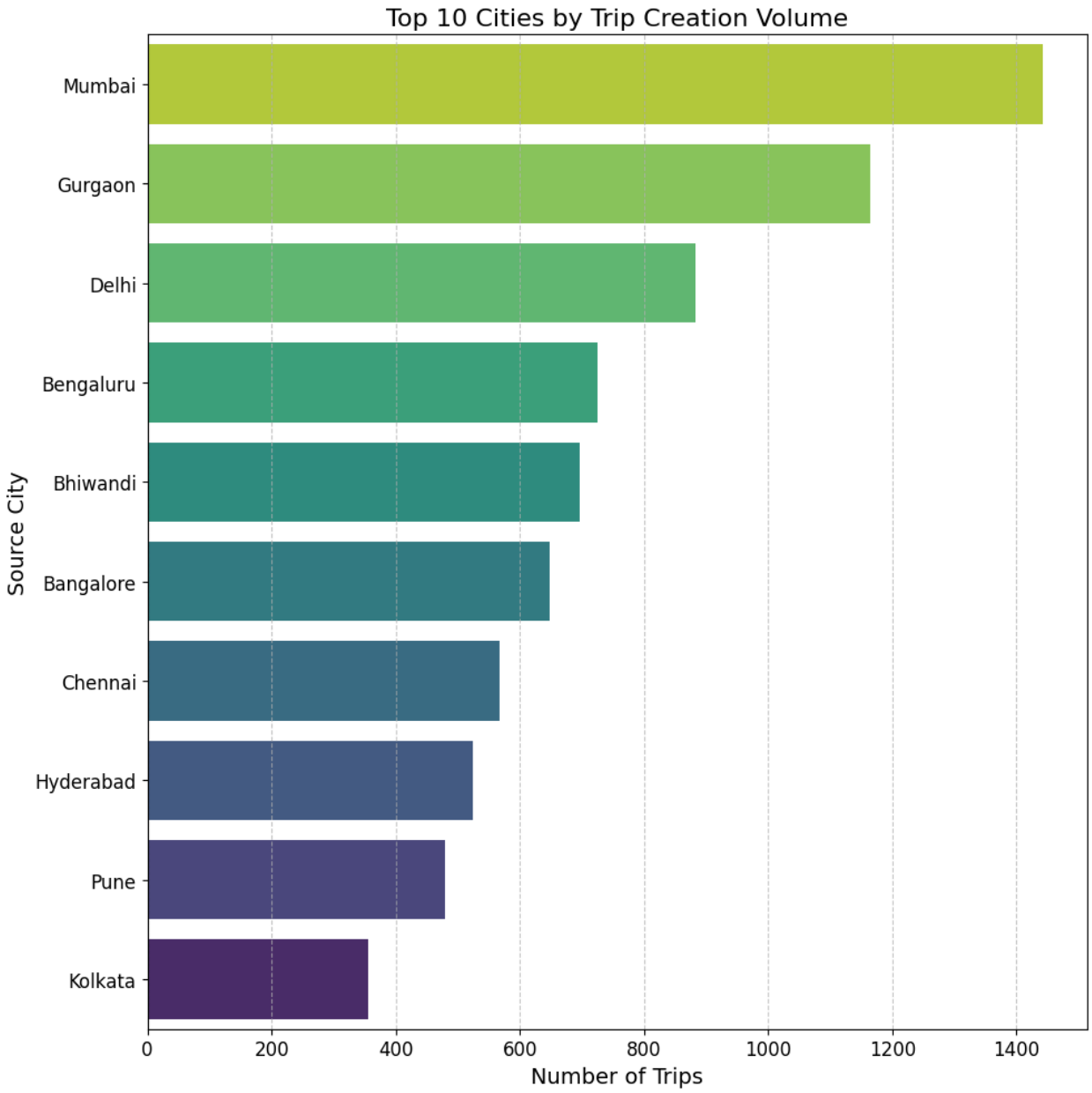
```
df_source_city = df2.groupby(by = 'source_city')
['trip_uuid'].count().to_frame().reset_index()
df_source_city['perc'] = np.round(df_source_city['trip_uuid'] * 100/
df_source_city['trip_uuid'].sum(), 2)
df_source_city = df_source_city.sort_values(by = 'trip_uuid',
ascending = False)[:10]
df_source_city
```

```
{"summary":"{\n  \"name\": \"df_source_city\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"source_city\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Pune\",\n          \"Gurgaon\",\n          \"Bangalore\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_uuid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 332,\n        \"min\": 356,\n
\"max\": 1442,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          480,\n          1165,\n          648\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"perc\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2.2435341663445993,\n        \"min\": 2.4,\n        \"max\": 9.73,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          3.24,\n
7.86,\n          4.37\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_source_city"}
```

```
plt.figure(figsize=(10, 10))

sns.barplot(data=df_source_city.head(10),
            x='trip_uuid',  # Set the x-axis to 'trip_uuid'
            y='source_city',  # Set the y-axis to 'source_city'
            palette='viridis_r')  # Choose a color palette and reverse
it for better contrast

plt.xlabel('Number of Trips', fontsize=14)
plt.ylabel('Source City', fontsize=14)
plt.title('Top 10 Cities by Trip Creation Volume', fontsize=16)
```

```
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

```
<ipython-input-124-1b0126a83d66>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=df_source_city.head(10),
```



- The visualization illustrates that the highest number of trips originated from Mumbai city, followed by Gurgaon Delhi, Bengaluru, and Bhiwandi. This observation indicates a robust seller presence in these cities, suggesting significant market strength and potential areas for further business development.

Exploring the Top 10 Cities by the number of Trips ending in Different Cities

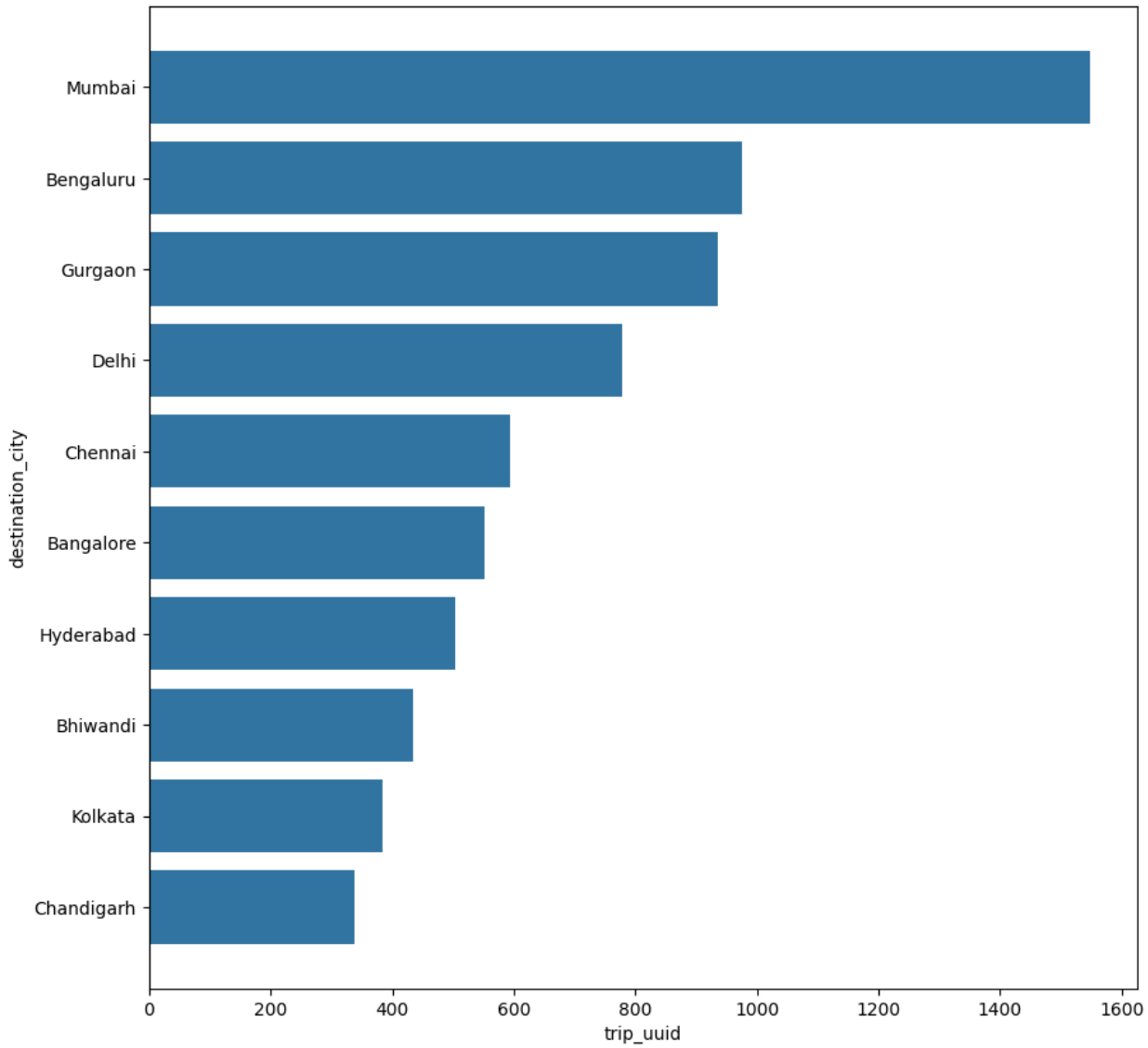
```
df_destination_city = df2.groupby(by = 'destination_city')
['trip_uuid'].count().to_frame().reset_index()
df_destination_city['perc'] =
np.round(df_destination_city['trip_uuid'] * 100/
```

```
df_destination_city['trip_uuid'].sum(), 2)
df_destination_city = df_destination_city.sort_values(by =
'trip_uuid', ascending = False)[:10]
df_destination_city
```

```
{"summary":"{\n  \"name\": \"df_destination_city\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"destination_city\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Kolkata\",\n          \"Bengaluru\",\n          \"Bangalore\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trip_uuid\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
369,\n        \"min\": 339,\n        \"max\": 1548,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          384,\n
975,\n          551\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"perc\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.4950315072604234,\n        \"min\": 2.29,\n        \"max\":
10.45,\n        \"num_unique_values\": 10,\n        \"samples\": [\n
2.59,\n          6.58,\n          3.72\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"df_destination_city"}
```

```
plt.figure(figsize = (10, 10))
sns.barplot(data = df_destination_city,
            x = df_destination_city['trip_uuid'],
            y = df_destination_city['destination_city'])
plt.plot()
```

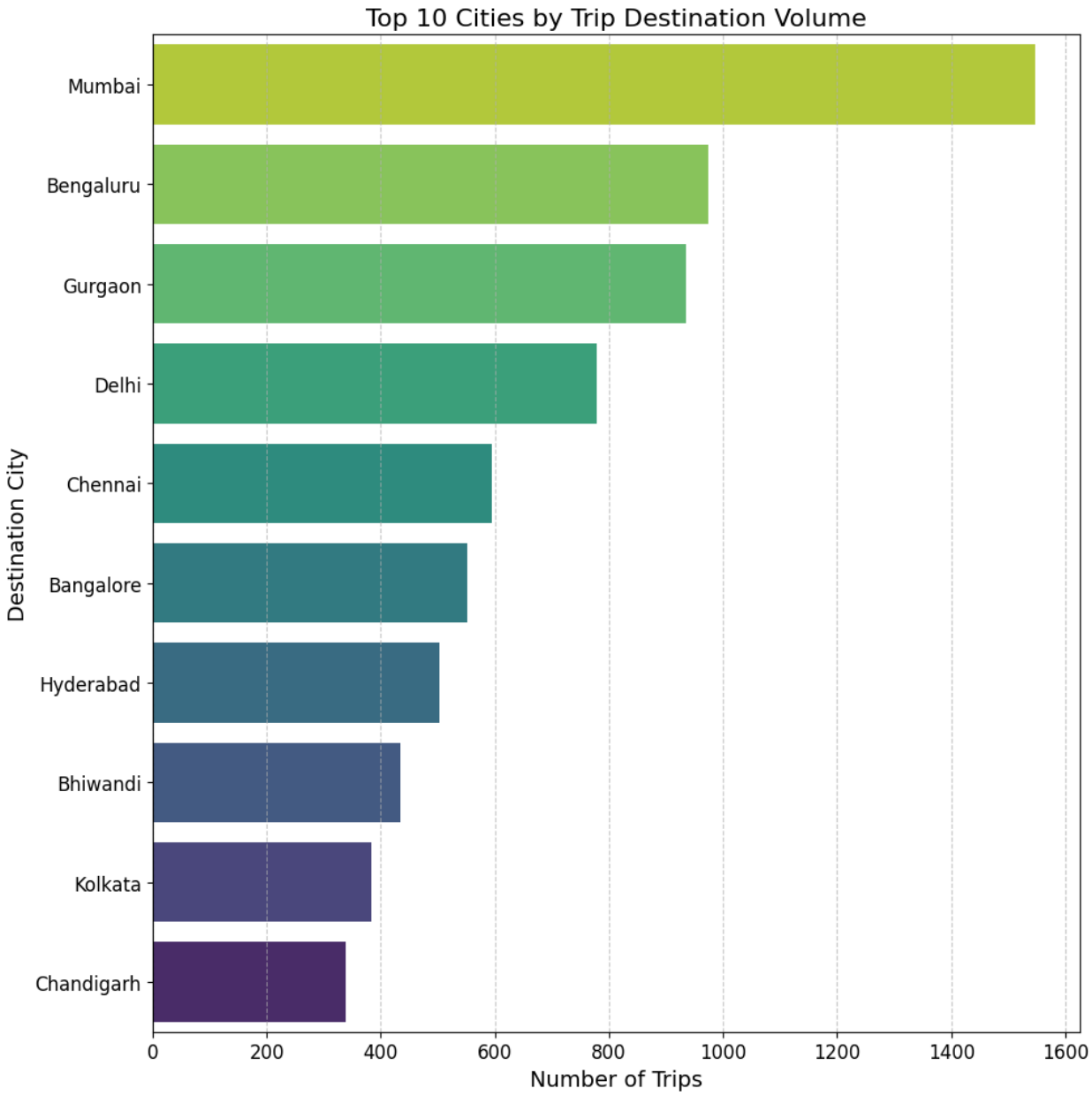
```
[]
```




```
plt.figure(figsize=(10, 10))

sns.barplot(data=df_destination_city.head(10),
            x='trip_uuid',
            y='destination_city',
            palette='viridis_r')

# Add labels and title
plt.xlabel('Number of Trips', fontsize=14)
plt.ylabel('Destination City', fontsize=14)
plt.title('Top 10 Cities by Trip Destination Volume', fontsize=16)

# Adjust tick label font size
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
# Add grid lines
plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

```
<ipython-input-128-0307d67bb1be>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=df_destination_city.head(10),
```



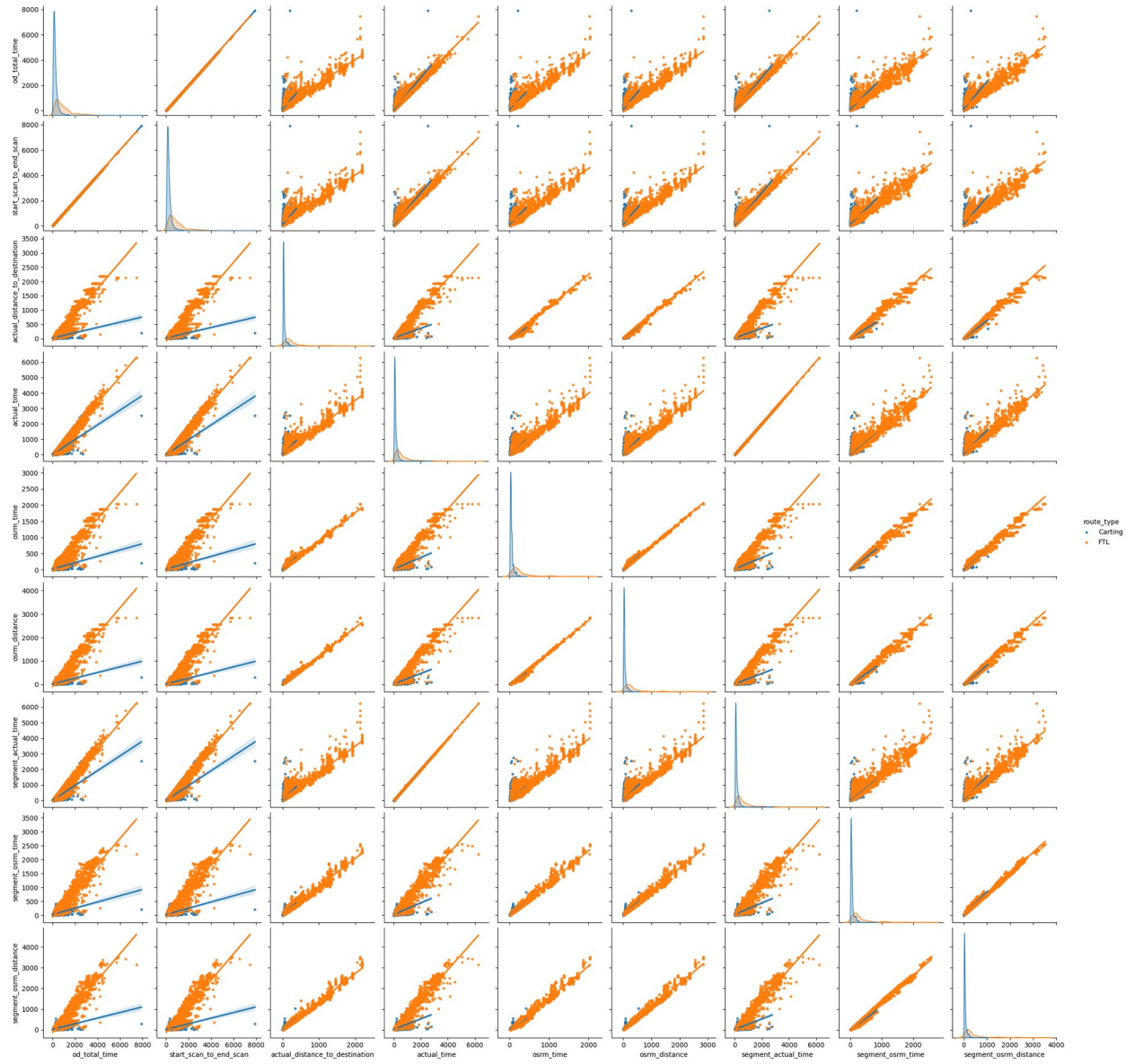
- The highest number of trips concluded in Mumbai city, followed by Bengaluru, Gurgaon, Delhi, and Chennai. This observation suggests a considerable volume of orders placed in these cities, indicating their importance as key destinations for trip completions.

Mulivariate Analysis

```
numerical_columns = ['od_total_time', 'start_scan_to_end_scan',
'actual_distance_to_destination',
                     'actual_time', 'osrm_time', 'osrm_distance',
'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']
sns.pairplot(data = df2,
```

```
            vars = numerical_columns,
            kind = 'reg',
            hue = 'route_type',
            markers = '.')
plt.plot()
```

```
[]
```




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

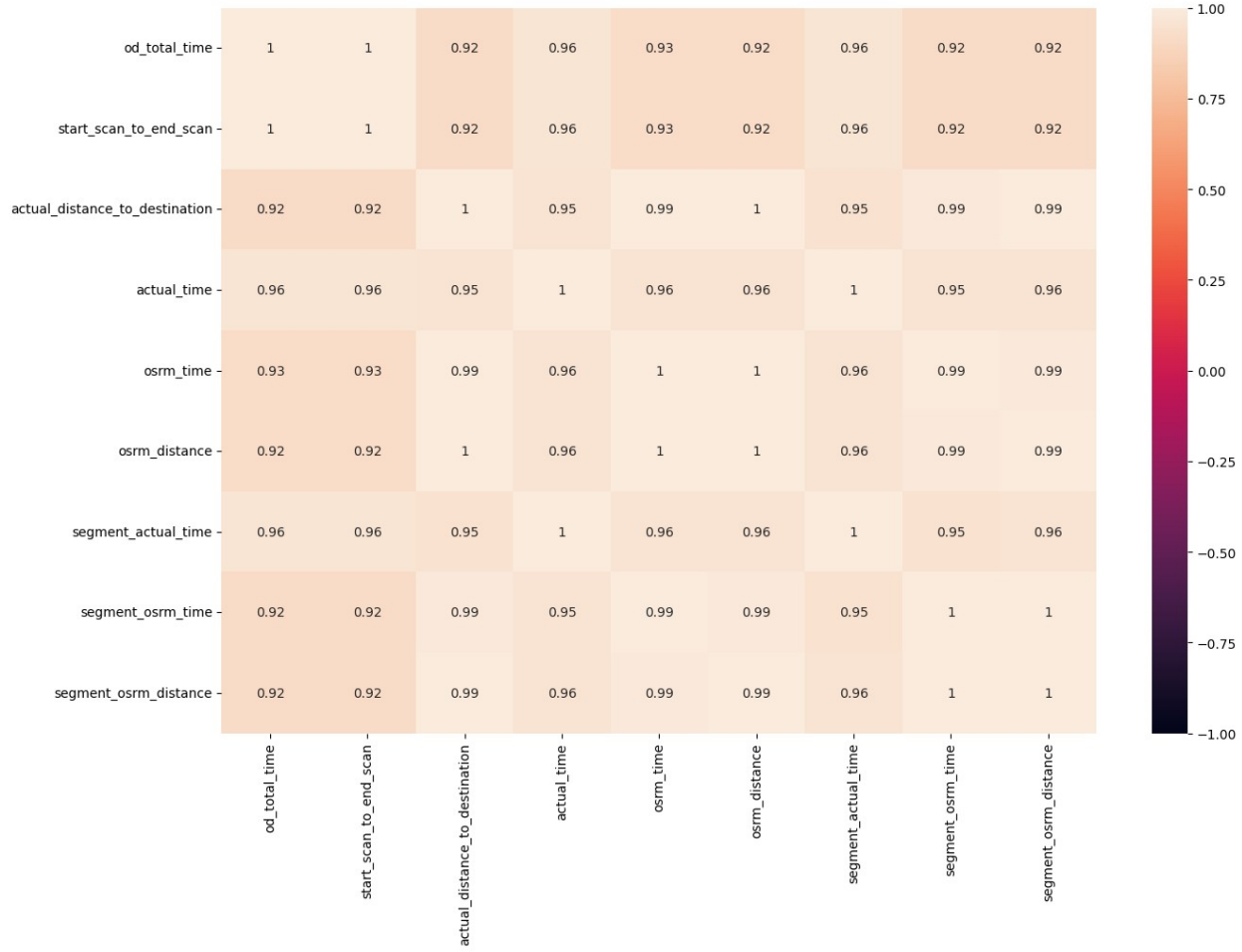
```
{"summary":"{\n  \"name\": \"df_corr\",\n  \"rows\": 9,\n  \"fields\":
[\n    {\n      \"column\": \"od_total_time\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
```

```
0.03421354109985085,\n        \"min\": 0.9182224206780394,\n
\"max\": 1.0,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          0.9184898796116135,\n          0.9999994565451016,\n
0.9242194973757498\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_scan_to_end_scan\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.03418123572738772,\n
\"min\": 0.9183082565871887,\n        \"max\": 1.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.9185611315937687,\n          1.0,\n          0.9242987644854069\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"actual_distance_to_destination\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.03339762040792926,\n
\"min\": 0.9182224206780394,\n        \"max\": 1.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.9875377399033718,\n          0.9183082565871887,\n
0.9972644751354265\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.018794268779785496,\n        \"min\":
0.9537568268413427,\n        \"max\": 1.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.9538724747598699,\n          0.9611466268913078,\n
0.9592140392165971\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.03011192583689132,\n        \"min\":
0.9265162836815423,\n        \"max\": 1.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.9932590034708473,\n          0.9265712231482862,\n
0.9975802833688245\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.03140938312682482,\n        \"min\":
0.9242194973757498,\n        \"max\": 1.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.9917979773483533,\n          0.9242987644854069,\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"segment_actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.01910409184724944,\n        \"min\": 0.9528208857966542,\n
\"max\": 1.0,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          0.9530393742866533,\n          0.9611713167578628,\n
0.9583532596149541\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"segment_osrm_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.033011442841323124,\n        \"min\":
0.9184898796116135,\n        \"max\": 1.0,\n
```

\"num_unique_values\": 9,\n   \"samples\": [\n   1.0,\n
0.9185611315937687,\n   0.9917979773483533\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"segment_osrm_distance\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.032985118476095227,\n   \"min\": 0.9191985864469532,\n
\"max\": 1.0,\n   \"num_unique_values\": 9,\n   \"samples\":
[\n   0.9960917145510623,\n   0.9192913271806832,\n
0.9947096183707673\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df_corr"}

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

```
[]
```



Insights

- High Correlation between Duration Metrics: Columns like od_total_time, start_scan_to_end_scan, actual_time, and segment_actual_time exhibit strong positive correlations, with values close to 1. This suggests that these metrics are highly correlated and likely represent similar aspects of trip duration.
- Strong Relationship between Distance Metrics: The columns related to distance, such as actual_distance_to_destination, osrm_distance, and segment_osrm_distance, also show significant positive correlations close to 1. This indicates a strong relationship between these distance metrics, implying consistency in measuring distances across different methods.
- Correlation between Time and Distance Metrics: Metrics representing time (e.g., actual_time, osrm_time) show moderate positive correlations with distance metrics (actual_distance_to_destination, osrm_distance). This suggests that longer distances generally correspond to longer durations, which aligns with intuitive expectations.
- Segment Metrics Correlation: Columns related to segmented metrics (segment_actual_time, segment_osrm_time, segment_osrm_distance) exhibit positive correlations with their non-segmented counterparts, albeit slightly weaker compared to correlations within each group. This implies that while segmented metrics may capture additional details, they still maintain a relationship with the overall metrics.

Hypothesis Testing

Analyzing the Discrepancy between 'od_total_time' and 'start_scan_to_end_scan': Hypothesis Testing and Visual Examination

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

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavene's test

STEP-3: Define Test statistics; Distribution of T under H0.

- If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

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

- We set our alpha value to be 0.05

STEP-5: Compare p-value and alpha.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

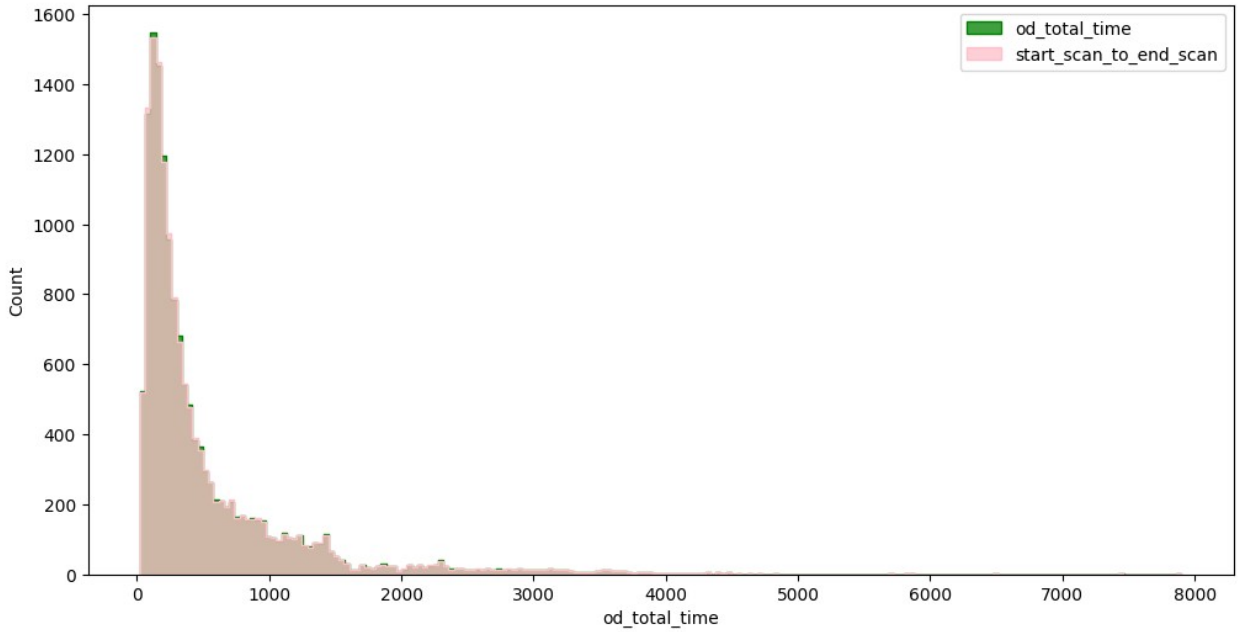
```
df2[['od_total_time', 'start_scan_to_end_scan']].describe()
```


```
{"summary":"{\n  \"name\": \"df2[['od_total_time',
'start_scan_to_end_scan']]\",\n  \"rows\": 8,\n  \"fields\": [\n    {\
n      \"column\": \"od_total_time\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5412.1698427340725,\n
\"min\": 23.46,\n        \"max\": 14817.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
531.6976297496119,\n          280.77,\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"start_scan_to_end_scan\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5412.419114898004,\n        \"min\": 23.0,\n        \"max\": 14817.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
530.8100155227104,\n          280.0,\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


- Visual Tests to know if the samples follow normal distribution

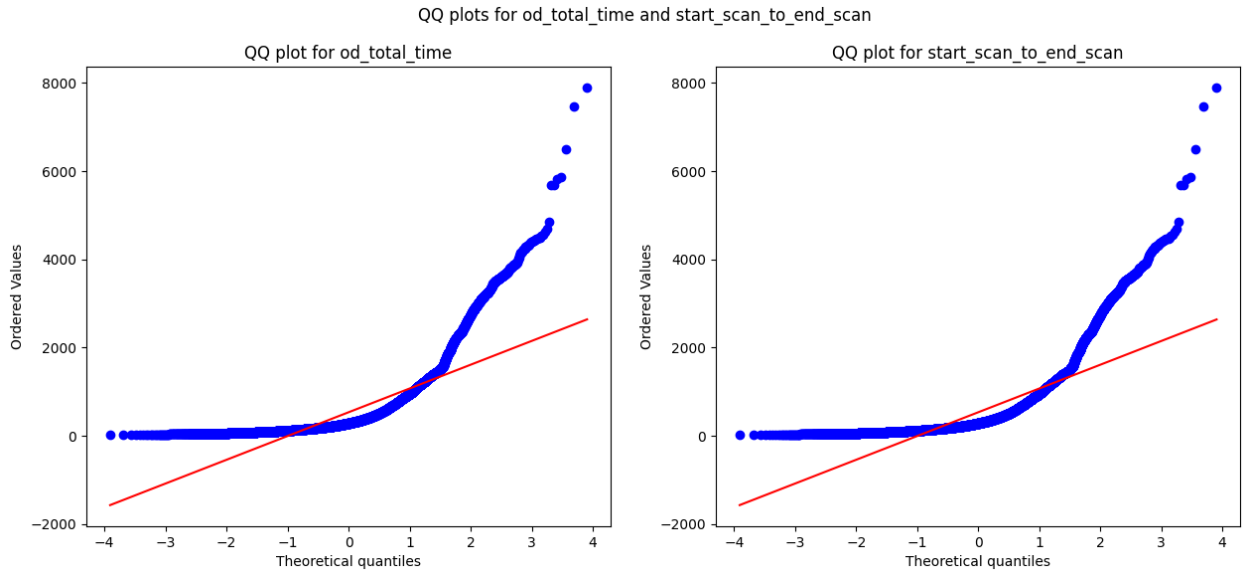
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['od_total_time'], element = 'step', color = 'green')
sns.histplot(df2['start_scan_to_end_scan'], element = 'step', color =
'pink')
plt.legend(['od_total_time', 'start_scan_to_end_scan'])
plt.plot()
```

```
[]
```



- Distribution check using QQ Plot

```
import scipy.stats as spy

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()

[]
```



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

Applying Shapiro-Wilk test for normality

- The sample follows normal distribution
- The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
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
```

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

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

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

```
transformed_od_total_time = spy.boxcox(df2['od_total_time'])[0]
test_stat, p_value = spy.shapiro(transformed_od_total_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.172770042757021e-25
The sample does not follow normal distribution

/usr/local/lib/python3.10/dist-packages/scipy/stats/
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
transformed_start_scan_to_end_scan =
spy.boxcox(df2['start_scan_to_end_scan'])[0]
test_stat, p_value = spy.shapiro(transformed_start_scan_to_end_scan)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.0471322892609475e-24
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "od_total_time" and "start_scan_to_end_scan" columns, the distributions do not follow normal distribution.
- Homogeneity of Variances using Lavene's test
- Null Hypothesis(H0) - Homogenous Variance
- Alternate Hypothesis(HA) - Non Homogenous Variance

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

```
p-value 0.9668007217581142
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

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

```
P-value : 0.7815123224221716
```

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

Hypothesis testing on actual_time aggregated value and OSRM time aggregated value

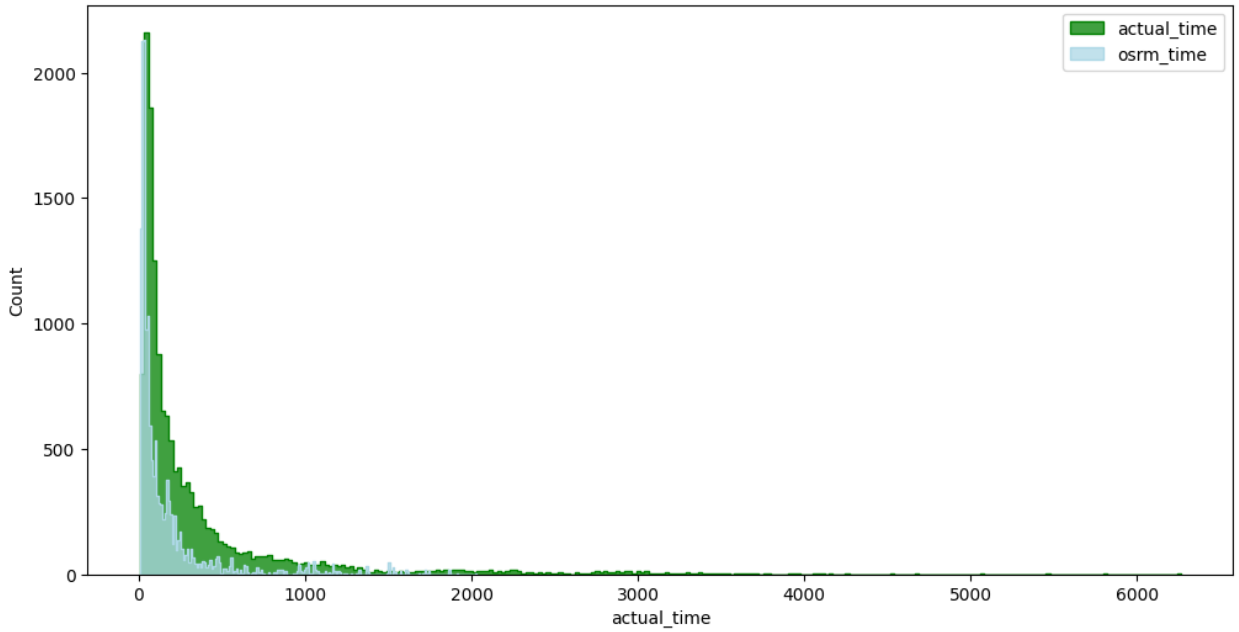
```
df2[['actual_time', 'osrm_time']].describe()
```


```
{"summary":"{\n  \"name\": \"df2[['actual_time', 'osrm_time']]\",\n
\"rows\": 8,\n  \"fields\": [\n    {\n      \"column\":
\"actual_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 5285.834160789364,\n        \"min\":
9.0,\n        \"max\": 14817.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          357.1437683105469,\n          149.0,\n
14817.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"osrm_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 5145.422874535779,\n        \"min\":
6.0,\n        \"max\": 14817.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          161.38401794433594,\n          60.0,\n
14817.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Visual Tests to know if the samples follow normal distribution

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

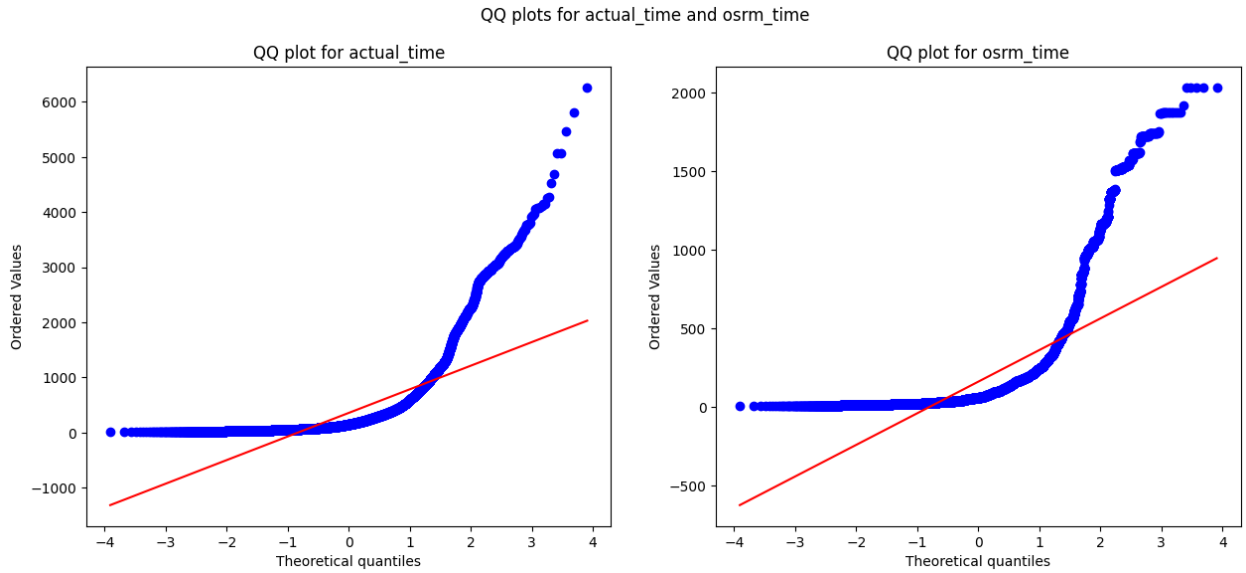
```
[]
```



Distribution check using QQ Plot

```
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()
```

```
[]
```



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

Applying Shapiro-Wilk test for normality

- The sample follows normal distribution
- The sample does not follow normal distribution

Alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
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
```

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

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

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

```
transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
The sample does not follow normal distribution

/usr/local/lib/python3.10/dist-packages/scipy/stats/
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "actual_time" and "osrm_time" columns, the distributions do not follow normal distribution.
- Homogeneity of Variances using Lavene's test

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

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

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

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

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

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

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

Hypothesis testing between actual_time aggregated value and segment actual time aggregated value

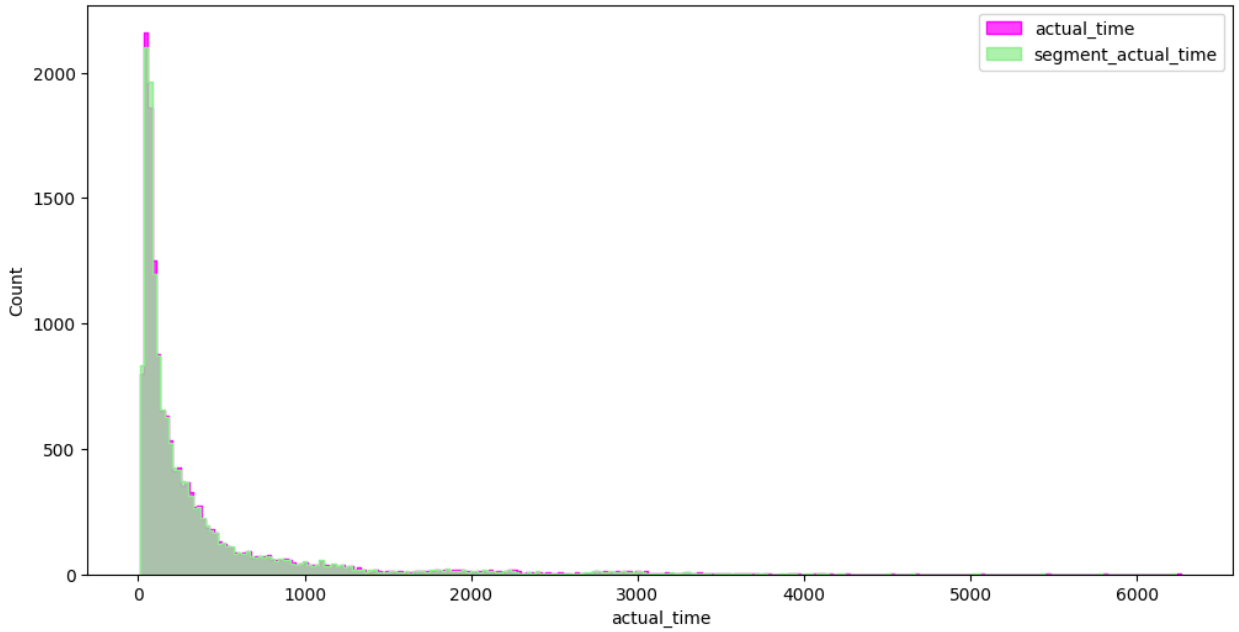
```
df2[['actual_time', 'segment_actual_time']].describe()
```


```
{"summary":"{\n  \"name\": \"df2[['actual_time',
'segment_actual_time']]\",\n  \"rows\": 8,\n  \"fields\": [\n    {\n
\"column\": \"actual_time\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5285.834160789364,\n
\"min\": 9.0,\n        \"max\": 14817.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
357.1437683105469,\n          149.0,\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"segment_actual_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5283.542112306394,\n        \"min\": 9.0,\n        \"max\": 14817.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
353.89227294921875,\n          147.0,\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


- Tests to know if the samples follow normal distribution

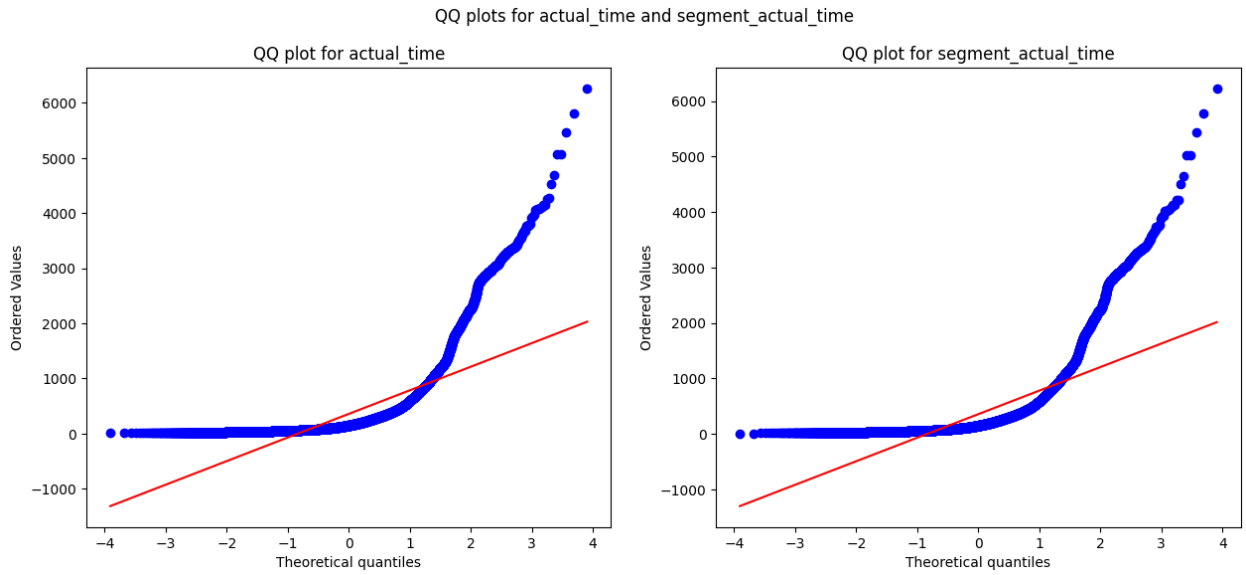
```
plt.figure(figsize = (12, 6))
sns.histplot(df2['actual_time'], element = 'step', color = 'magenta')
sns.histplot(df2['segment_actual_time'], element = 'step', color =
'lightgreen')
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```

```
[]
```



- Distribution check using QQ Plot

```
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()

[]
```



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

Applying Shapiro-Wilk test for normality

- Ho : The sample follows normal distribution
- Ha : The sample does not follow normal distribution

Alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
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
```

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

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

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

```
transformed_actual_time = spy.boxcox(df2['actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
The sample does not follow normal distribution
```

```
transformed_segment_actual_time =
spy.boxcox(df2['segment_actual_time'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_actual_time)
print('p-value', p_value)
if p_value < 0.05:
```

```
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 5.700074948787037e-29
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "actual_time" and "segment_actual_time" columns, the distributions do not follow normal distribution.
- Homogeneity of Variances using Lavene's test

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['actual_time'],
df2['segment_actual_time'])
print('p-value', p_value)

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

```
p-value 0.695502241317651
The samples have Homogenous Variance
```

Since the samples do not come from normal distribution T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

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

```
p-value 0.4164235159622476
The samples are similar
```

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

Hypothesis testing between osrm distance aggregated value and segment osrm distance aggregated value

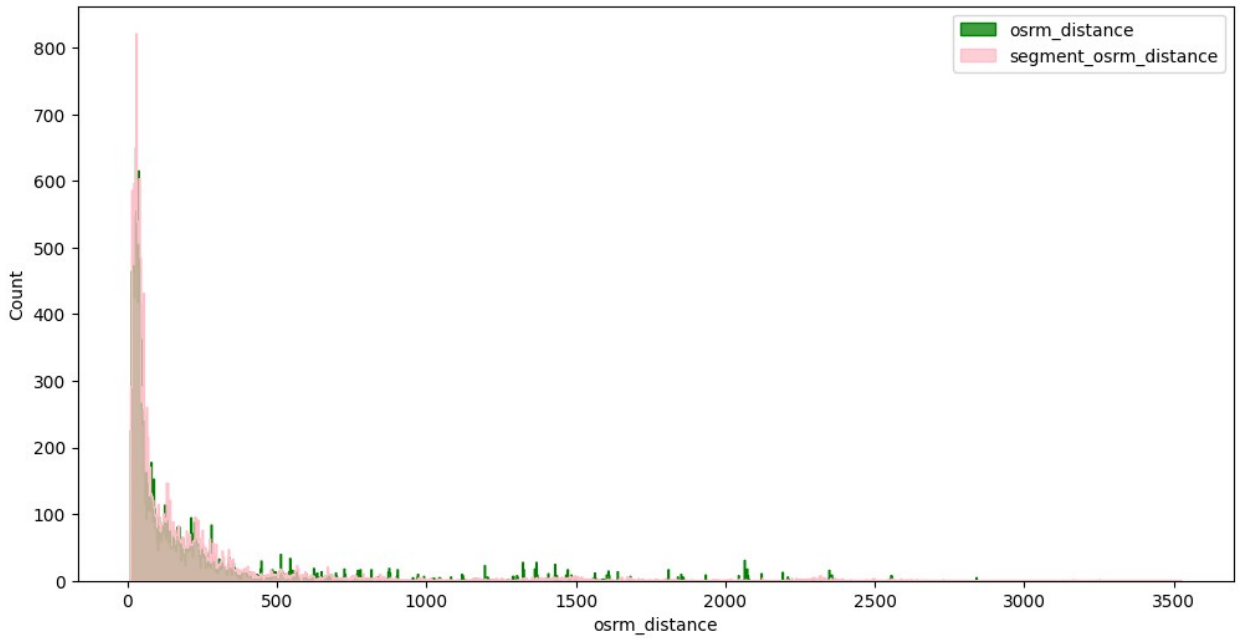
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

```
{"summary":"{\n  \"name\": \"df2[['osrm_distance',
'segment_osrm_distance']]\",\n  \"rows\": 8,\n  \"fields\": [\n    {\n
\"column\": \"osrm_distance\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5138.703672416228,\n
\"min\": 9.07289981842041,\n        \"max\": 14817.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
204.34471130371094,\n          65.61880493164062,\n          14817.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"segment_osrm_distance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5149.504430731242,\n        \"min\": 9.07289981842041,\n
\"max\": 14817.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          223.20115661621094,\n
70.15440368652344,\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


- Tests to know if the samples follow normal distribution

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

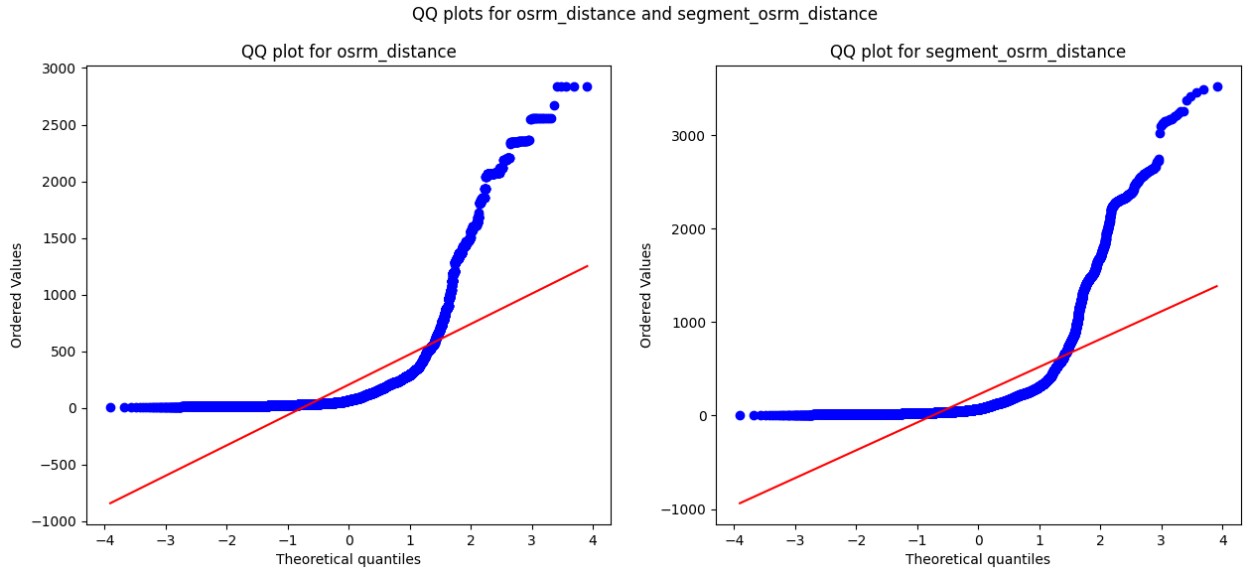
```
[]
```



- Distribution check using QQ Plot

```
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()
```

```
[]
```



- The samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

- Ho: The sample follows normal distribution
- Ha: The sample does not follow normal distribution

Alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
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
```

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

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

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

```
transformed_osrm_distance = spy.boxcox(df2['osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.063104779582808e-41
The sample does not follow normal distribution

/usr/local/lib/python3.10/dist-packages/scipy/stats/
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
transformed_segment_osrm_distance =
spy.boxcox(df2['segment_osrm_distance'])[0]
test_stat, p_value = spy.shapiro(transformed_segment_osrm_distance)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.049169406432229e-38
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "osrm_distance" and "segment_osrm_distance" columns, the distributions do not follow normal distribution.
- Homogeneity of Variances using Lavene's test

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df2['osrm_distance'],
```

```
df2['segment_osrm_distance'])
print('p-value', p_value)

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

```
p-value 0.00020976006524780905
The samples do not have Homogenous Variance
```

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

```
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
```

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

Hypothesis testing between osrm time aggregated value and segment osrm time aggregated value

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

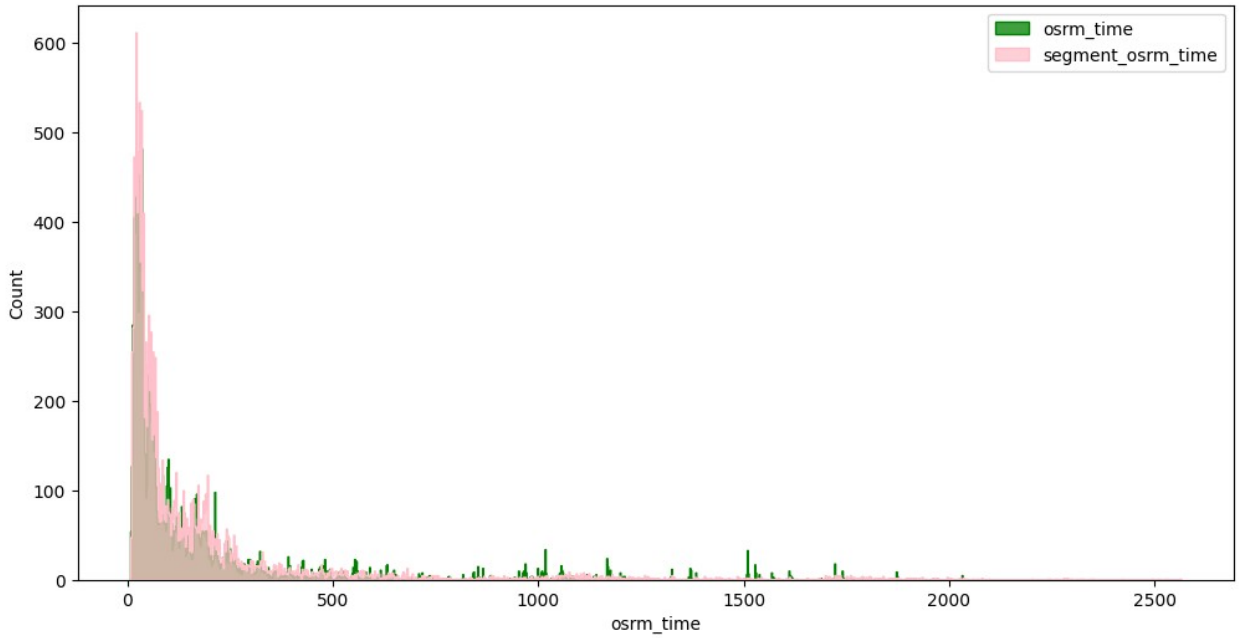
```
{"summary":"{\n  \"name\": \"df2[['osrm_time',
'segment_osrm_time']]\",\n  \"rows\": 2,\n  \"fields\": [\n    {\n
\"column\": \"count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 14817.0,\n
\"max\": 14817.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          14817.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mean\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 13.835085379910657,\n
\"min\": 161.38401794433594,\n        \"max\": 180.9497833251953,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
180.9497833251953\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 30.533620904155693,\n        \"min\": 271.3609924316406,\n
```

```
\"max\": 314.54205322265625,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          314.54205322265625\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"min\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 6.0,\n
\"max\": 6.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"25%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.4142135623730951,\n        \"min\": 29.0,\n        \"max\":
31.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
31.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"50%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.5355339059327378,\n        \"min\": 60.0,\n        \"max\":
65.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
65.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"75%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 12.020815280171307,\n        \"min\": 168.0,\n
\"max\": 185.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          185.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 376.1808075912433,\n
\"min\": 2032.0,\n        \"max\": 2564.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          2564.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


- Tests to know if the samples follow normal distribution

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

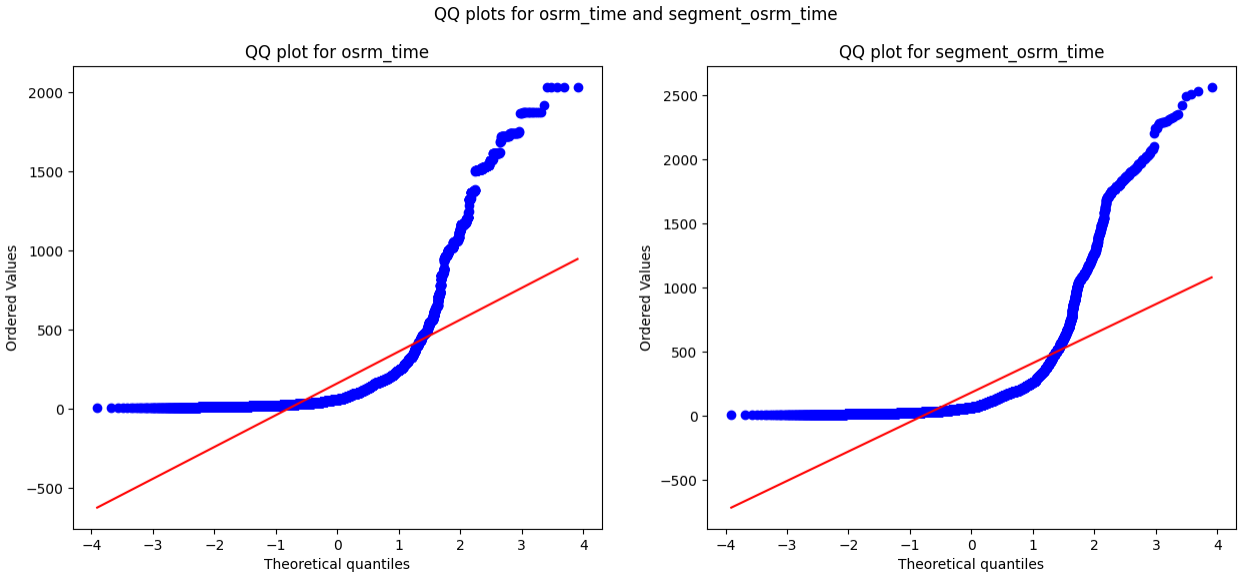
```
[]
```



- Distribution check using QQ Plot

```
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()
```

```
[]
```



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

Applying Shapiro-Wilk test for normality

- Ho: The sample follows normal distribution
- Ha: The sample does not follow normal distribution

Alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
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
```

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

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

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

```
transformed_osrm_time = spy.boxcox(df2['osrm_time'])[0]
test_stat, p_value = spy.shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
The sample does not follow normal distribution

/usr/local/lib/python3.10/dist-packages/scipy/stats/
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
transformed_segment_osrm_time = spy.boxcox(df2['segment_osrm_time'])
[0]
test_stat, p_value = spy.shapiro(transformed_segment_osrm_time)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 4.943039152219146e-34
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "osrm_time" and "segment_osrm_time" columns, the distributions do not follow normal distribution.
- Homogeneity of Variances using Lavene's test
- Null Hypothesis(H0) - Homogenous Variance
- Alternate Hypothesis(HA) - Non Homogenous Variance

```
test_stat, p_value = spy.levene(df2['osrm_time'],
df2['segment_osrm_time'])
print('p-value', p_value)

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

```
p-value 8.349506135727595e-08
The samples do not have Homogenous Variance
```

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

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

```
p-value 2.2995370859748865e-08
The samples are not similar
```

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

Outliers

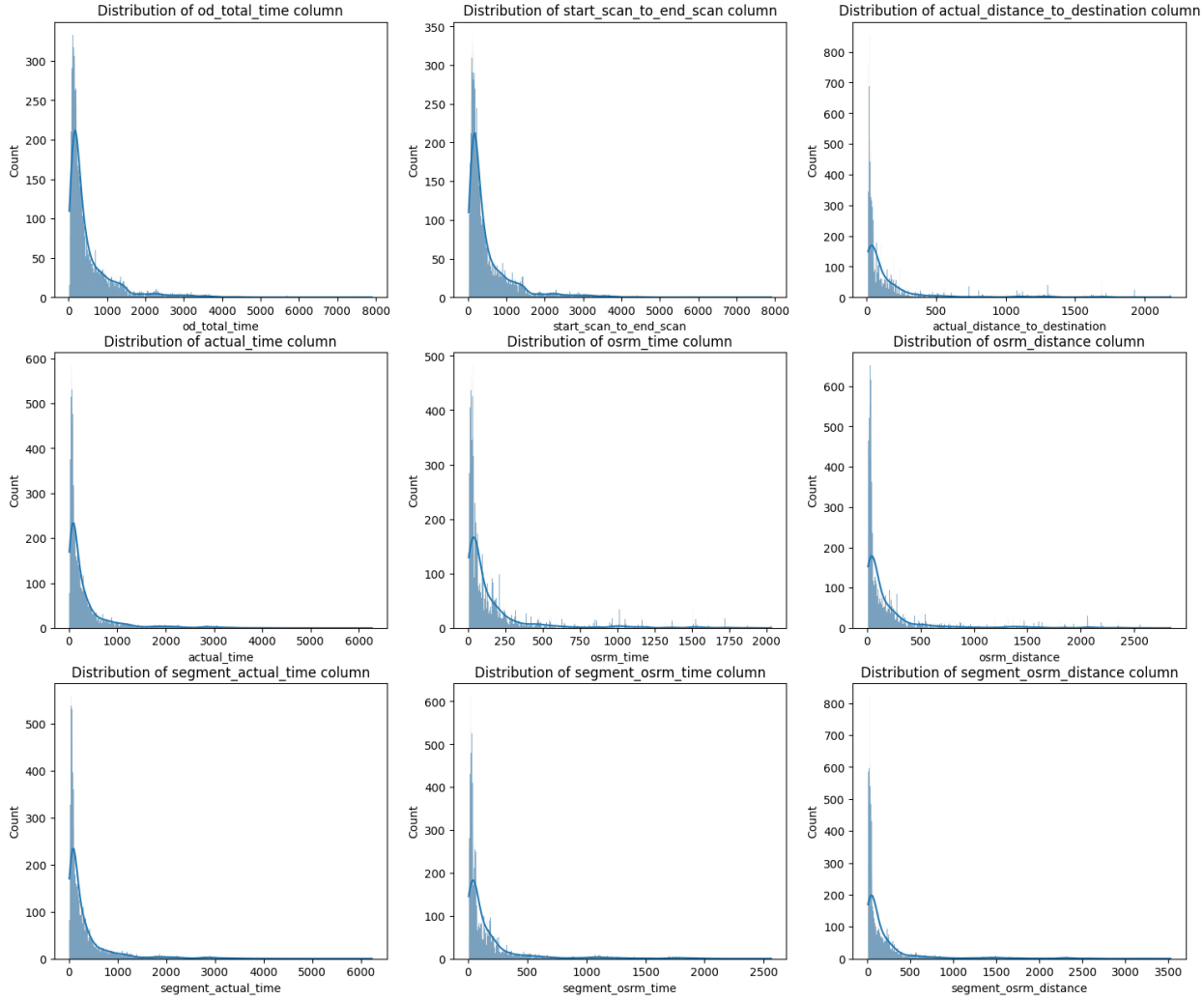
Finding outliers in the numerical variables

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


```
{"summary":"{\n  \"name\": \"df2[numerical_columns]\",\n  \"rows\":
9,\n  \"fields\": [\n    {\n      \"column\": \"count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 14817.0,\n        \"max\": 14817.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          14817.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mean\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
149.96729048507535,\n        \"min\": 161.38401794433594,\n
\"max\": 531.6976297496119,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          180.9497833251953\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 153.727516833242,\n
\"min\": 271.3609924316406,\n        \"max\": 658.8682230803469,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
314.54205322265625\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"min\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6.7667350620627,\n        \"min\": 6.0,\n        \"max\":
23.46,\n        \"num_unique_values\": 6,\n        \"samples\": [\n
23.46\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"25%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 50.883796236755856,\n        \"min\": 22.837238311767578,\n
\"max\": 149.93,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          31.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"50%\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 93.12474237752726,\n        \"min\":
48.47407150268555,\n        \"max\": 280.77,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          65.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"75%\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 191.68710939693733,\
n        \"min\": 164.5832061767578,\n        \"max\": 638.2,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          185.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2449.66409690628,\n
\"min\": 2032.0,\n        \"max\": 7898.55,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          2564.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

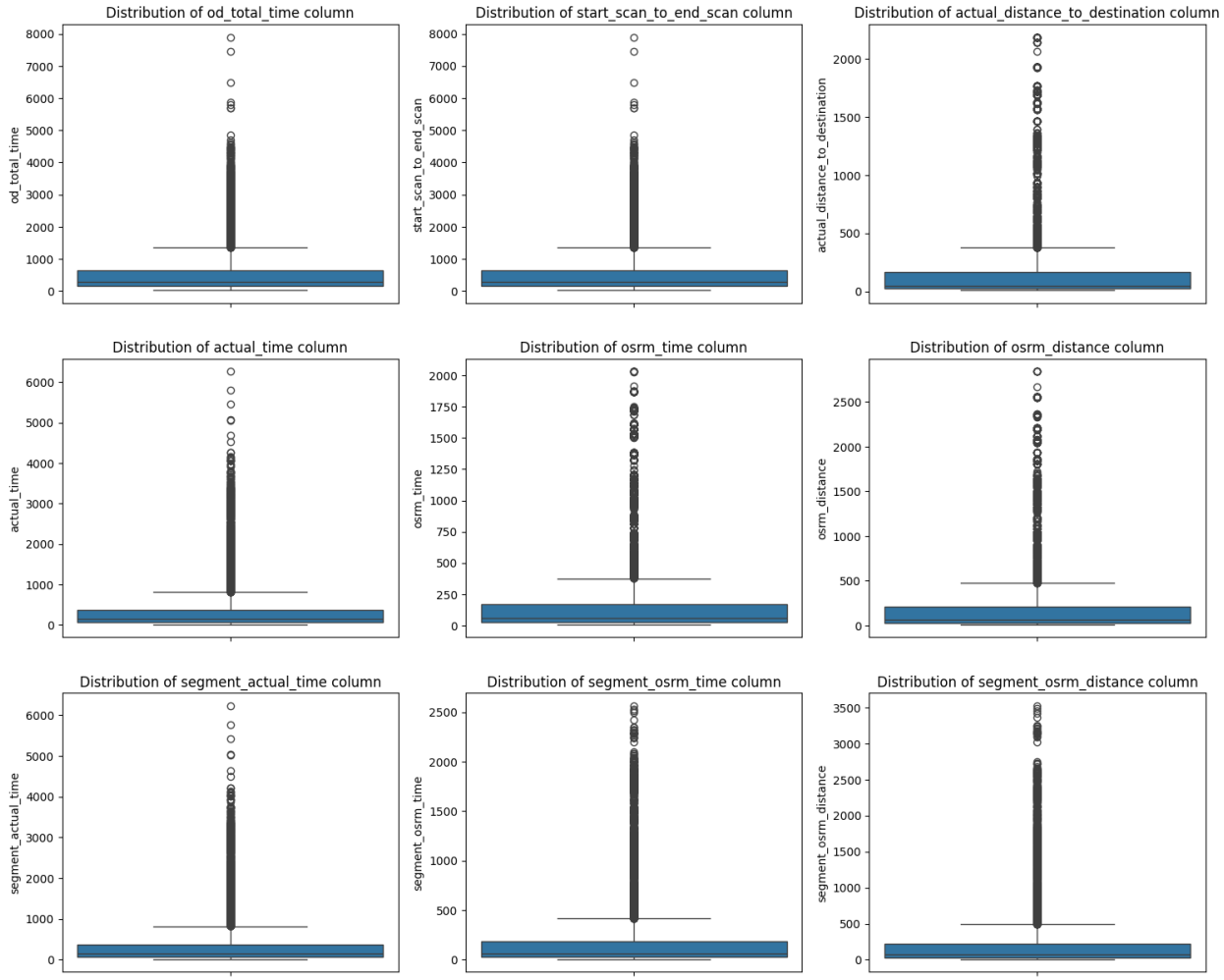
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    sns.histplot(df2[numerical_columns[i]], bins = 1000, kde = True)
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



- All numerical columns in the dataset exhibit right-skewed distributions.

```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
```

```
    sns.boxplot(df2[numerical_columns[i]])
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



- There are outliers in all the numerical columns that need to be treated.

```
# Detecting Outliers

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

```
    print(f'UB : {UB}')
    print(f'Number of outliers : {outliers.shape[0]}')
    print('----------------------------------')
```

```
Column : od_total_time
Q1 : 149.93
Q3 : 638.2
IQR : 488.27000000000004
LB : -582.4750000000001
UB : 1370.605
Number of outliers : 1266
----------------------------------
Column : start_scan_to_end_scan
Q1 : 149.0
Q3 : 637.0
IQR : 488.0
LB : -583.0
UB : 1369.0
Number of outliers : 1267
----------------------------------
Column : actual_distance_to_destination
Q1 : 22.837238311767578
Q3 : 164.5832061767578
IQR : 141.74596786499023
LB : -189.78171348571777
UB : 377.20215797424316
Number of outliers : 1449
----------------------------------
Column : actual_time
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
LB : -387.5
UB : 824.5
Number of outliers : 1643
----------------------------------
Column : osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
LB : -179.5
UB : 376.5
Number of outliers : 1517
----------------------------------
Column : osrm_distance
Q1 : 30.81920051574707
Q3 : 208.47500610351562
IQR : 177.65580558776855
LB : -235.66450786590576
UB : 474.95871448516846
```

```
Number of outliers : 1524
---------------------------------
Column : segment_actual_time
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
LB : -385.5
UB : 818.5
Number of outliers : 1643
---------------------------------
Column : segment_osrm_time
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
LB : -200.0
UB : 416.0
Number of outliers : 1492
---------------------------------
Column : segment_osrm_distance
Q1 : 32.65449905395508
Q3 : 218.80239868164062
IQR : 186.14789962768555
LB : -246.56735038757324
UB : 498.02424812316895
Number of outliers : 1548
---------------------------------
```

The outliers present in our sample data can be the true outliers. It's best to remove outliers only when there is a sound reason for doing so. Some outliers represent natural variations in the population, and they should be left as is in the dataset.

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

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

```
route_type
Carting    8908
FTL        5909
Name: count, dtype: int64
```

```
# 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'])

# Get value counts after one-hot encoding

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

```
route_type
0    8908
1    5909
Name: count, dtype: int64
```

```
# Get value counts of categorical variable 'data' before one-hot
encoding

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

```
data
training    10654
test         4163
Name: count, dtype: int64
```

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

# Get value counts after one-hot encoding

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

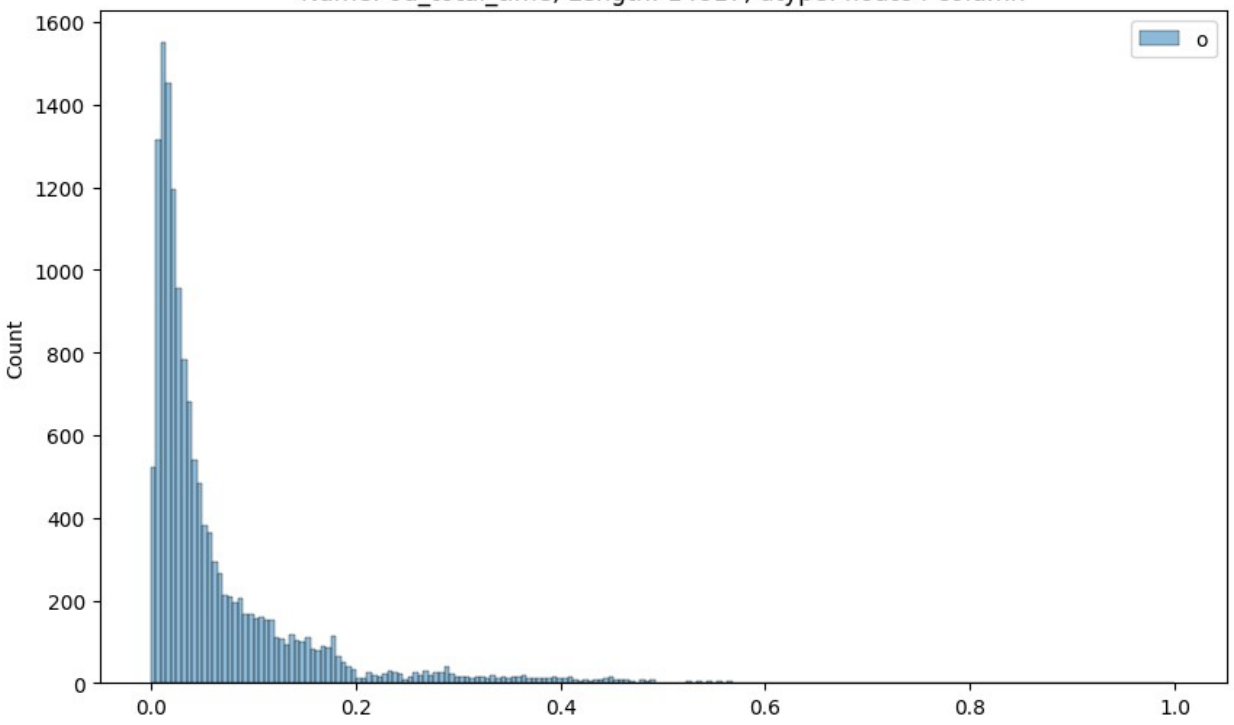
```
data
1    10654
0     4163
Name: count, dtype: int64
```

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

```
from sklearn.preprocessing import MinMaxScaler

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()
```

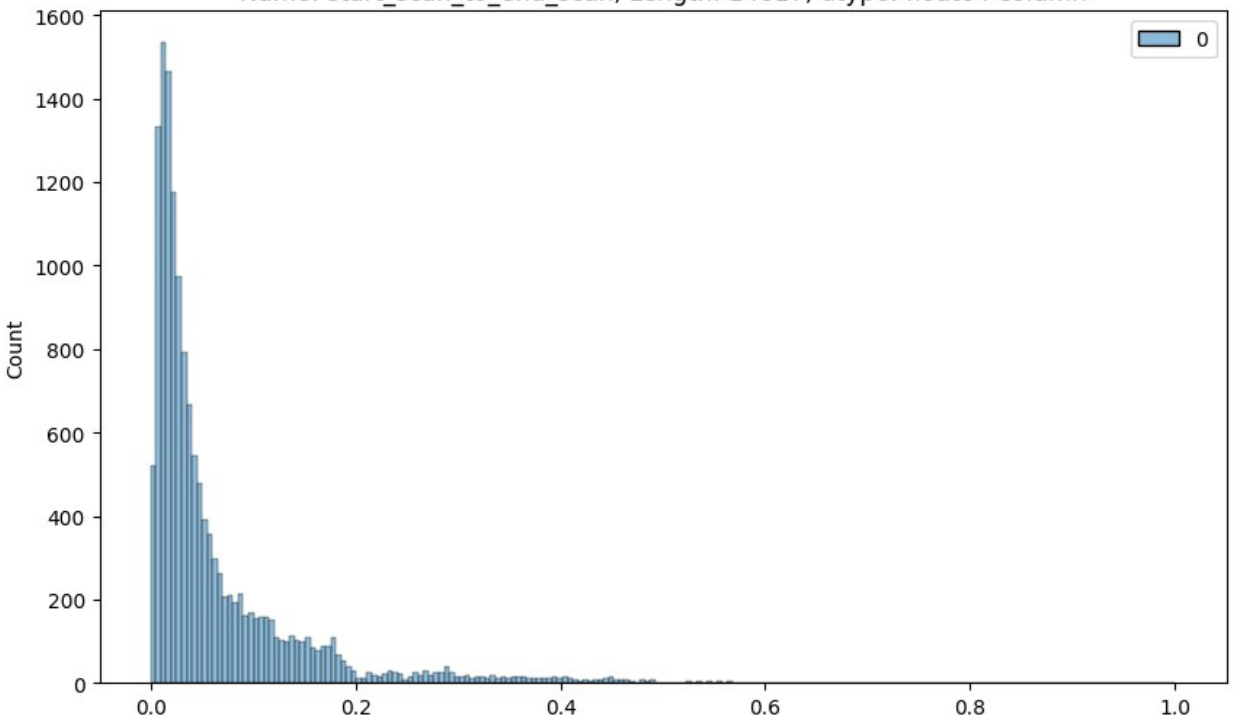
```
[]
```

```
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
```



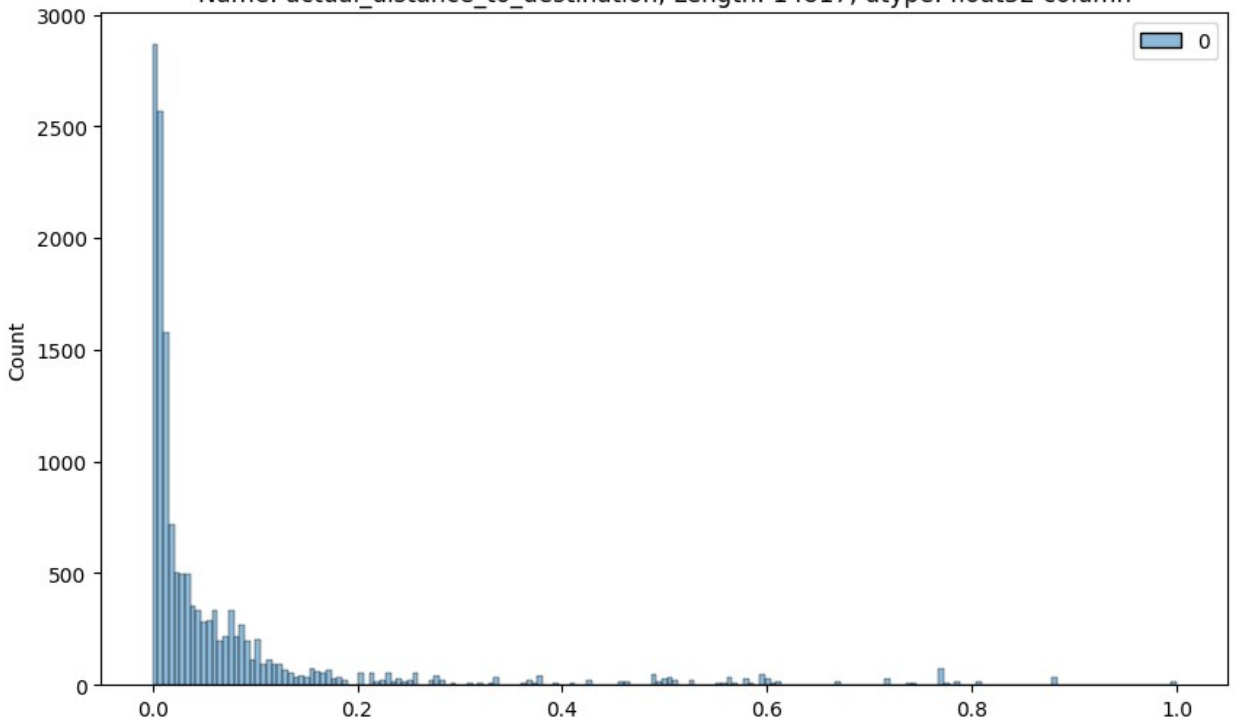

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled =
scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(
-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['start_scan_to_end_scan']} column")
plt.plot()

[]
```

```
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: float64 column
```



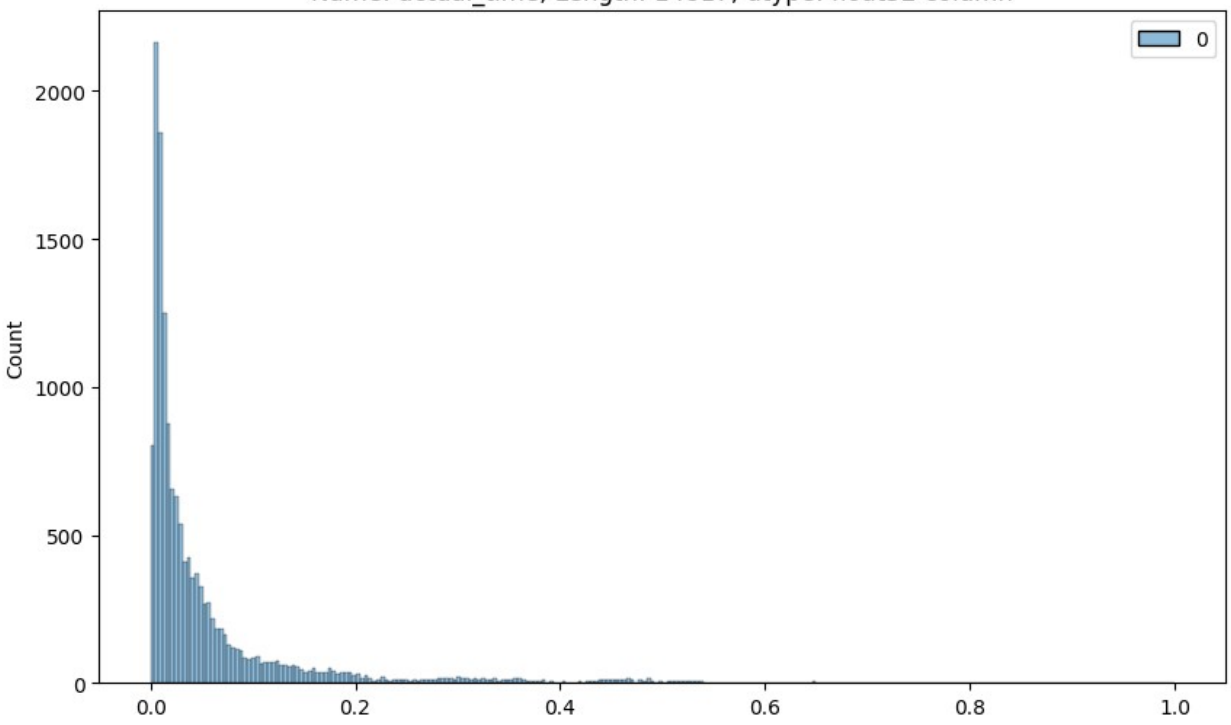

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled =
scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().
reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_distance_to_destination']}
column")
plt.plot()

[]
```

```
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
```



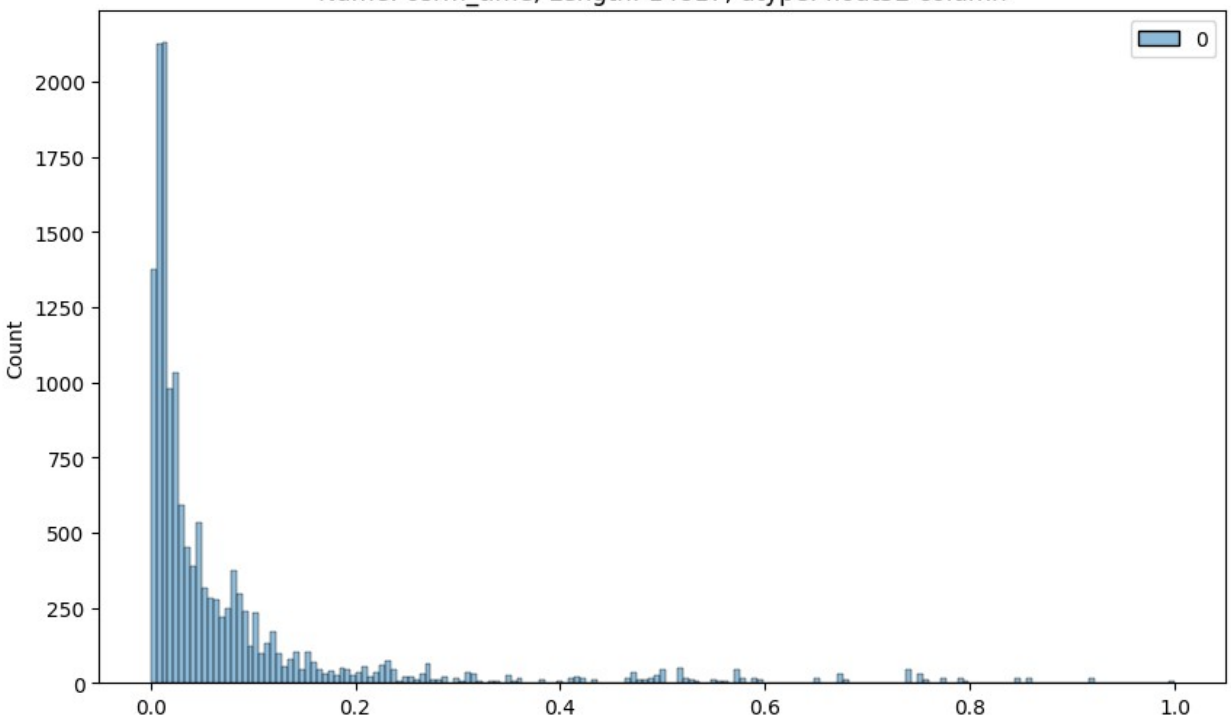

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-
1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_time']} column")
plt.plot()

[]
```

```
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
```



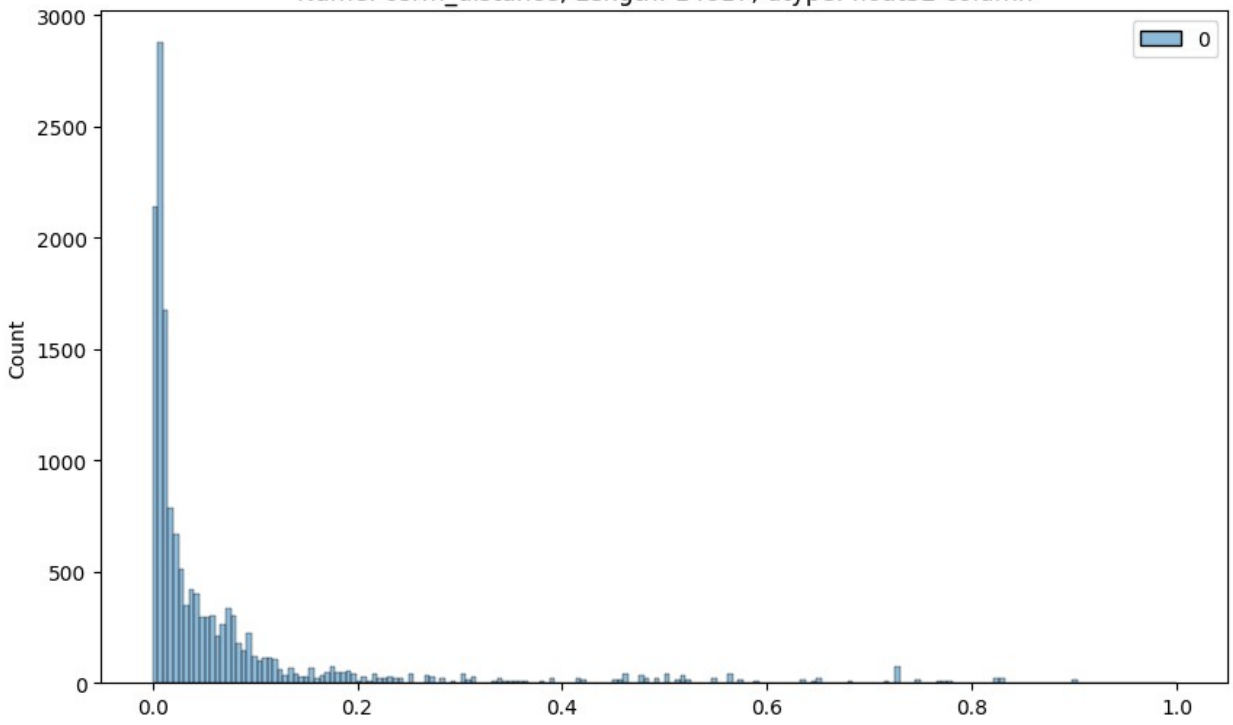

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['osrm_time'].to_numpy().reshape(-1,
1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['osrm_time']} column")
plt.plot()

[]
```

```
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
```



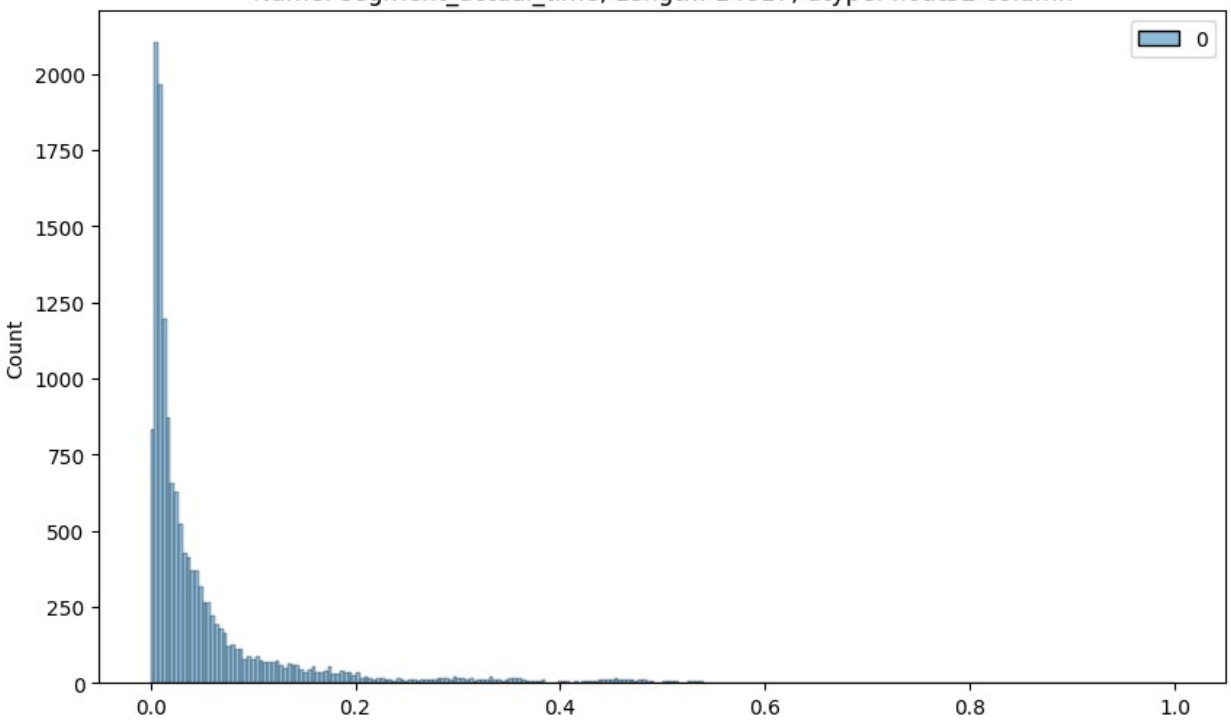

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled =
scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['osrm_distance']} column")
plt.plot()

[]
```

```
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
```



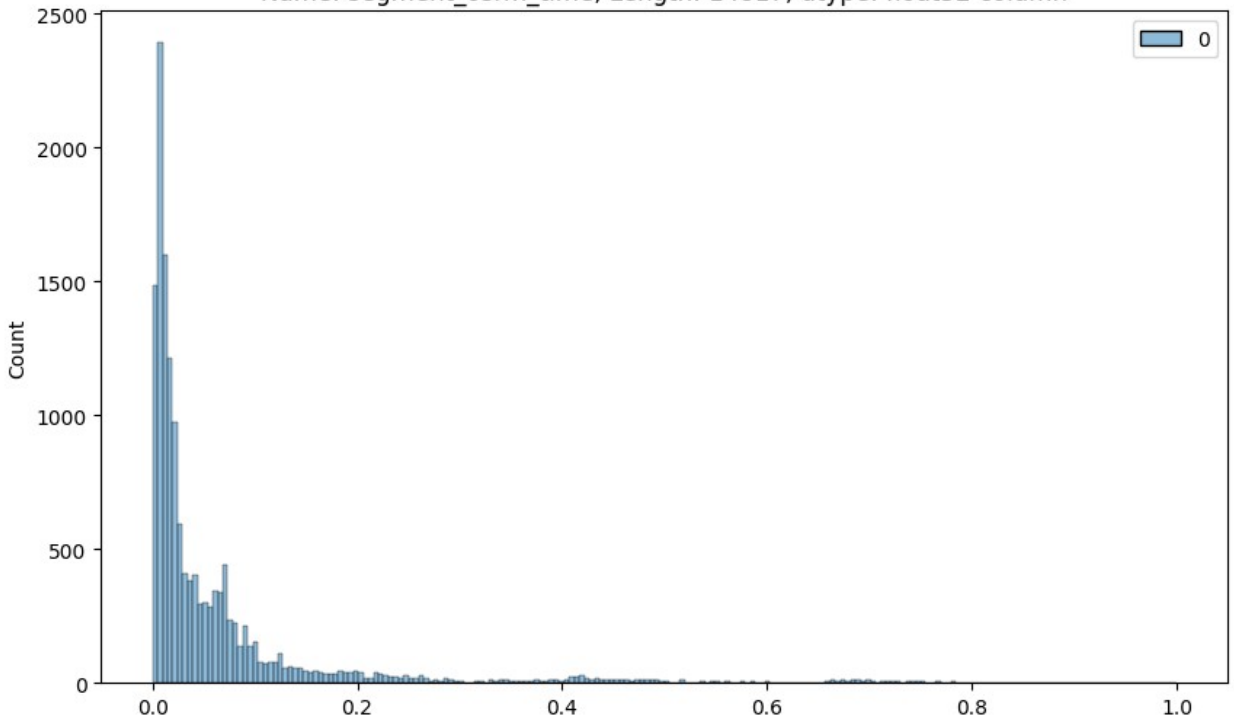

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled =
scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,
1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_actual_time']} column")
plt.plot()

[]
```

```
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
```



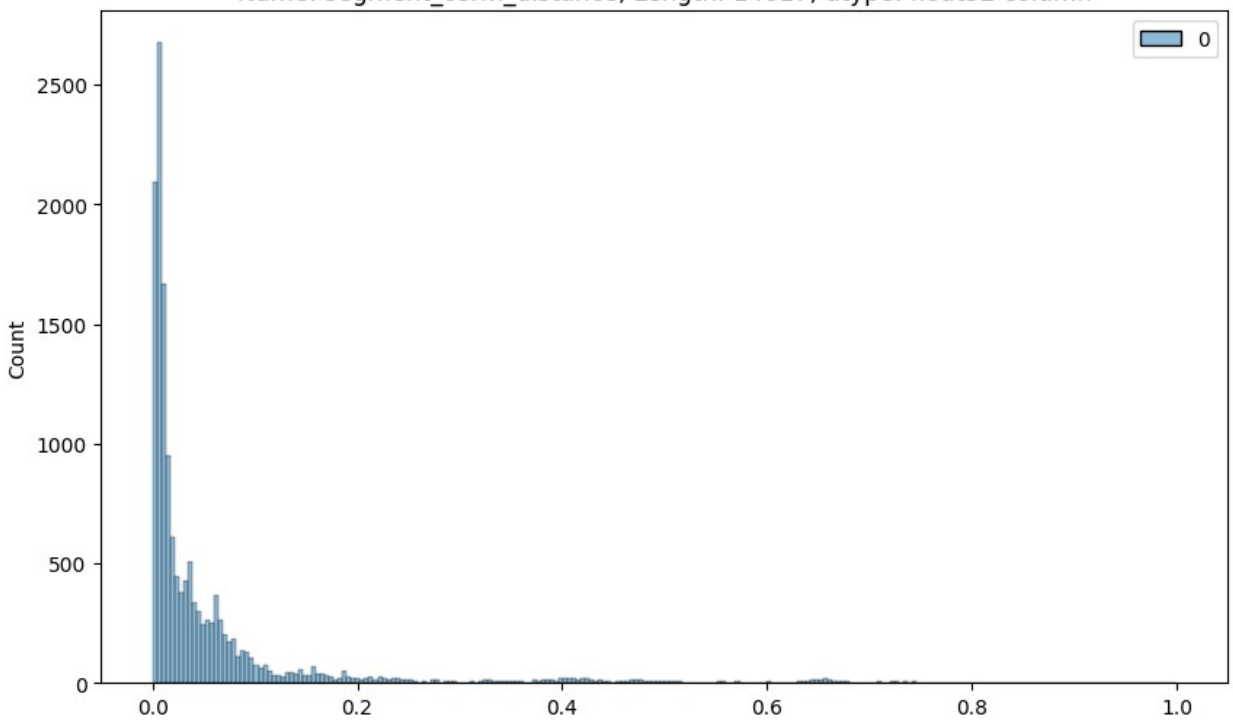

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled =
scaler.fit_transform(df2['segment_osrm_time'].to_numpy().reshape(-1,
1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['segment_osrm_time']} column")
plt.plot()

[]
```

```
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
```




```
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()

[]
```

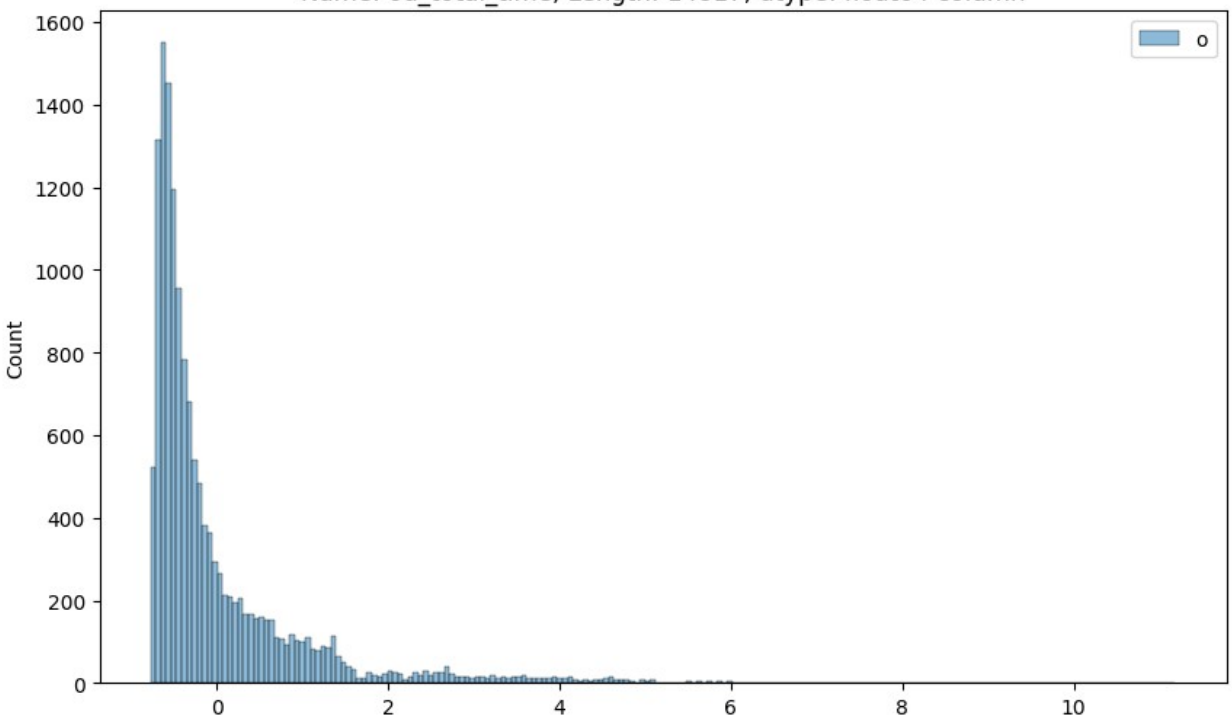
```
Normalized 0       1320.473267
           1         84.189400
           2       2545.267822
           3         19.876600
           4        146.791901
                       ...
       14812         64.855103
       14813         16.088299
       14814        104.886597
       14815        223.532394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



Column Standardization

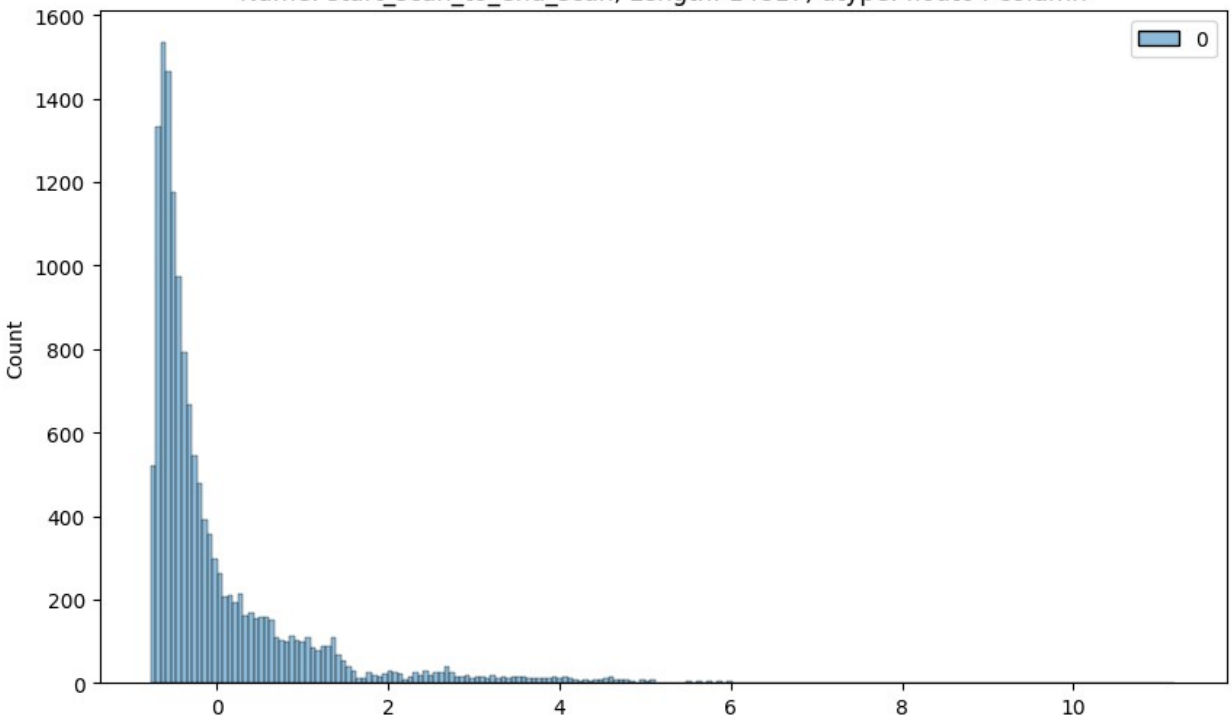
```
from sklearn.preprocessing import StandardScaler

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()

[]
```

```
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
```



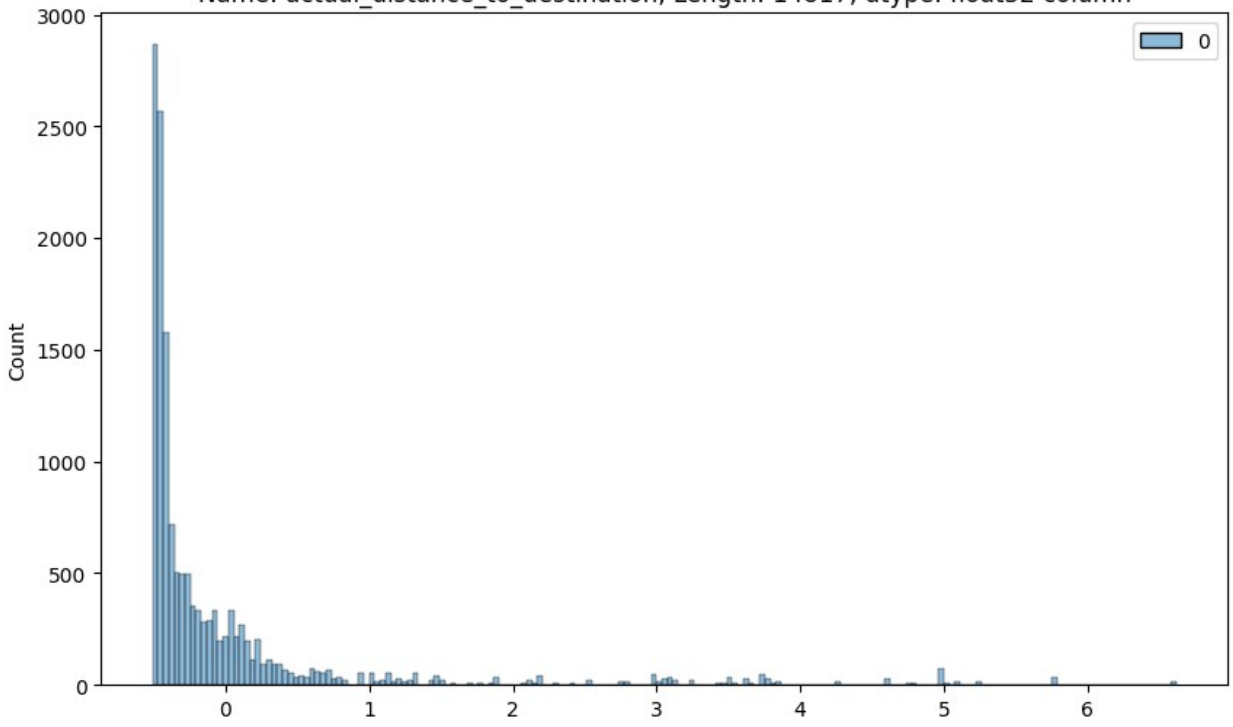

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled =
scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(
-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['start_scan_to_end_scan']} column")
plt.plot()

[]
```

```
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: float64 column
```



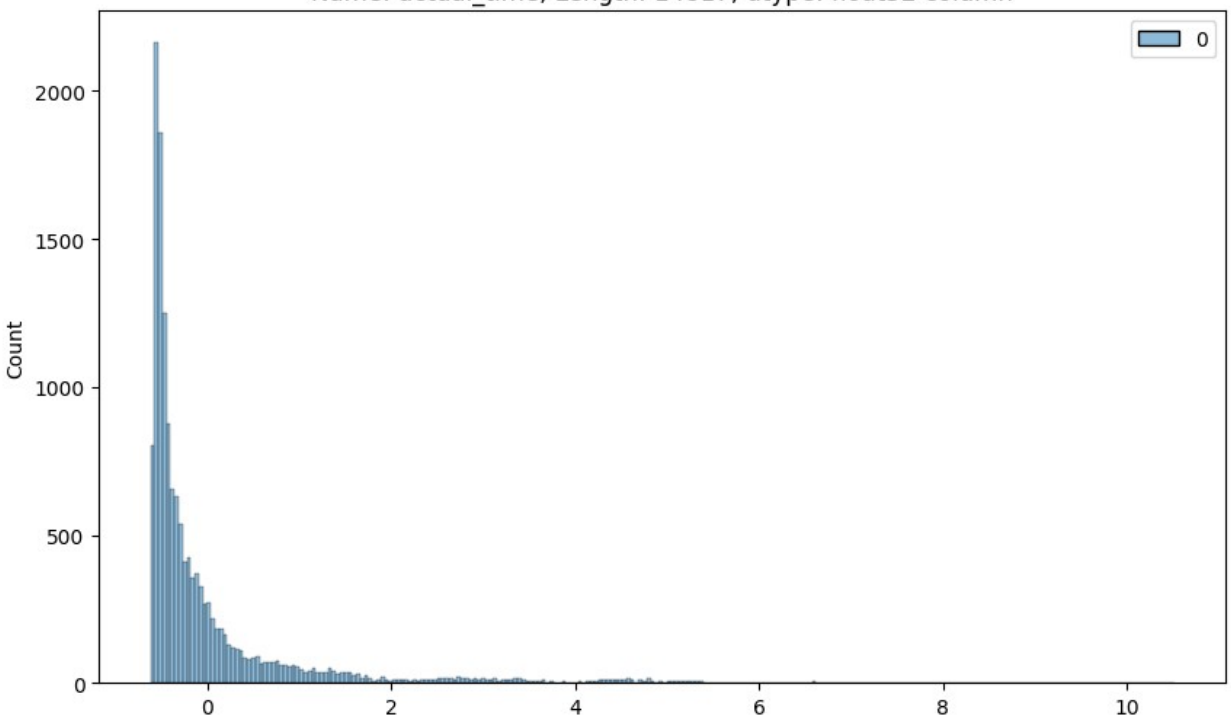

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled =
scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().
reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_distance_to_destination']}
column")
plt.plot()

[]
```

```
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
```



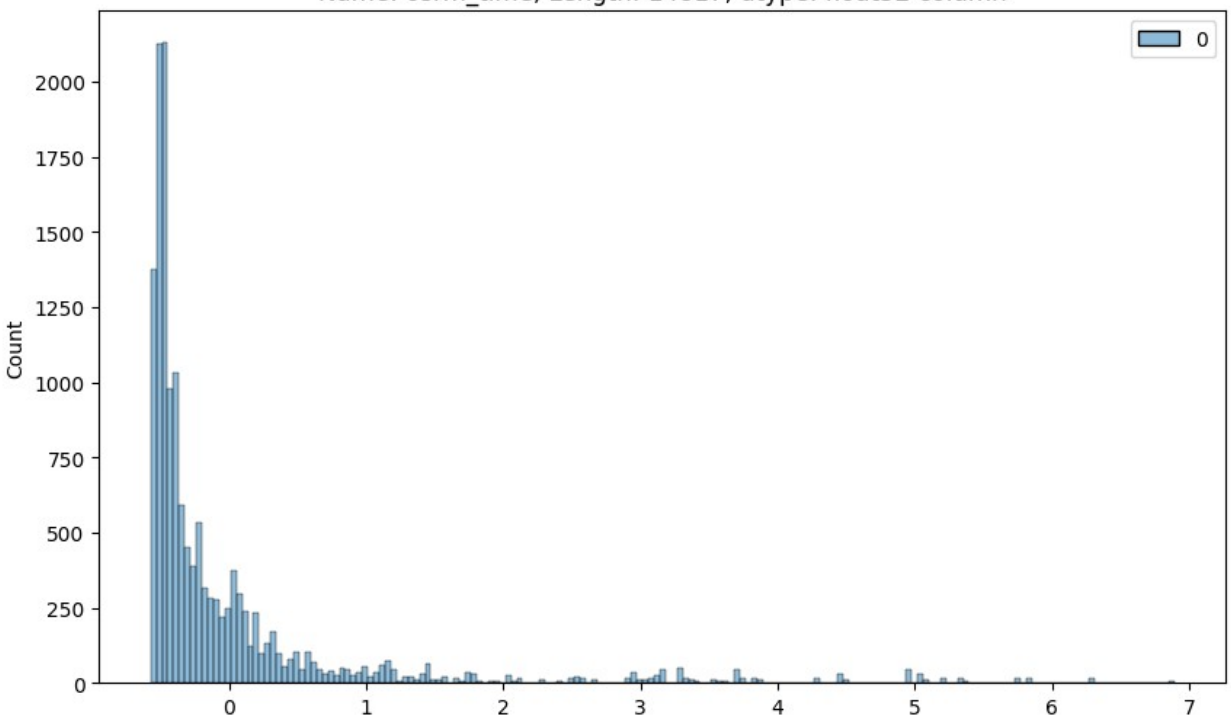

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-
1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['actual_time']} column")
plt.plot()

[]
```

```
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
```



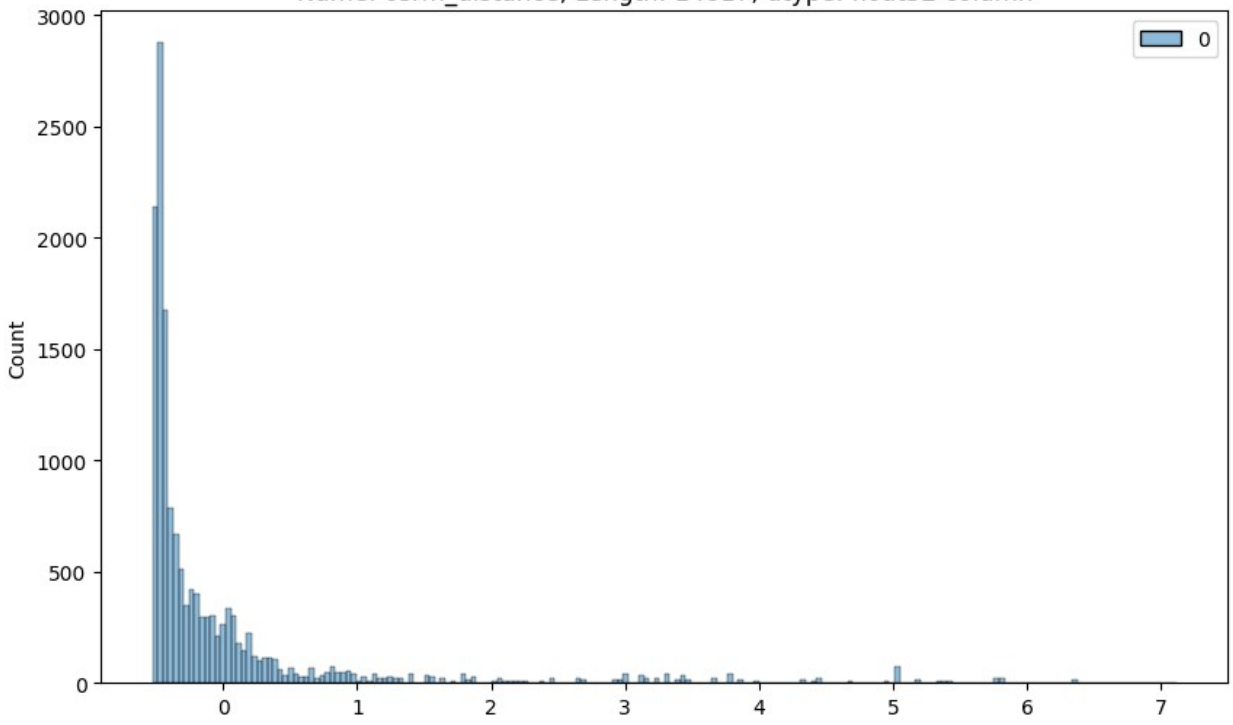

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled = scaler.fit_transform(df2['osrm_time'].to_numpy().reshape(-1,
1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['osrm_time']} column")
plt.plot()

[]
```

```
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
```



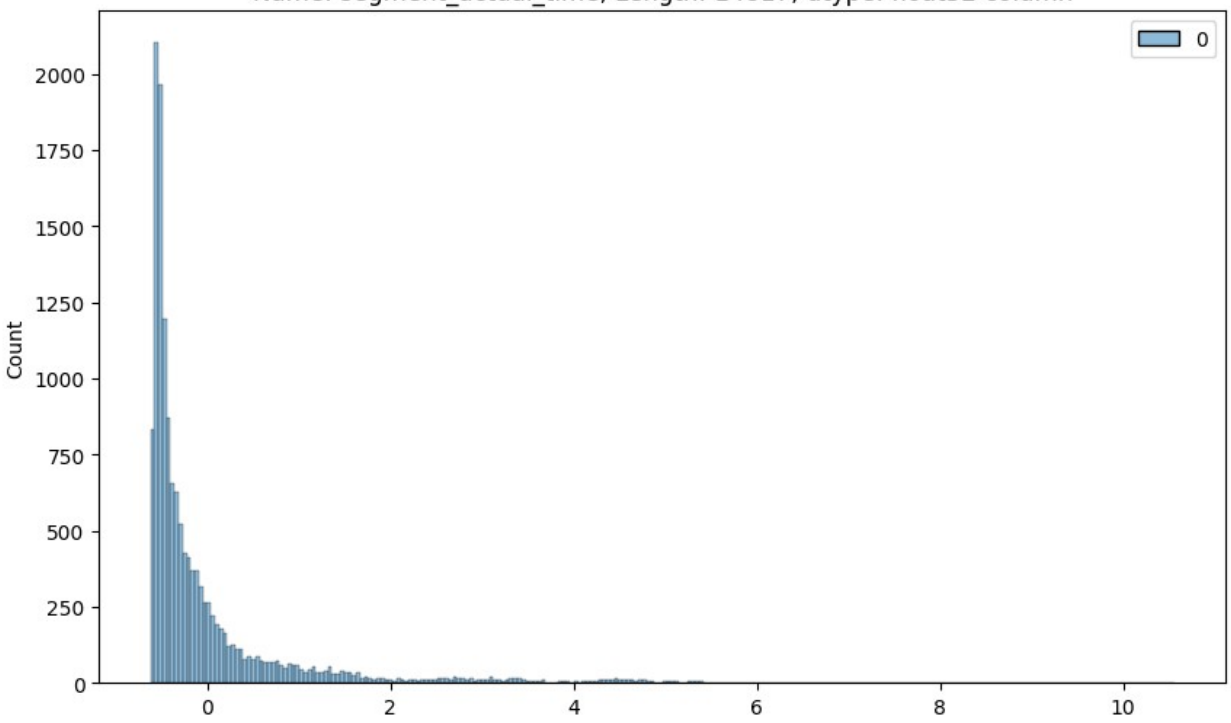

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled =
scaler.fit_transform(df2['osrm_distance'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['osrm_distance']} column")
plt.plot()

[]
```

```
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
```



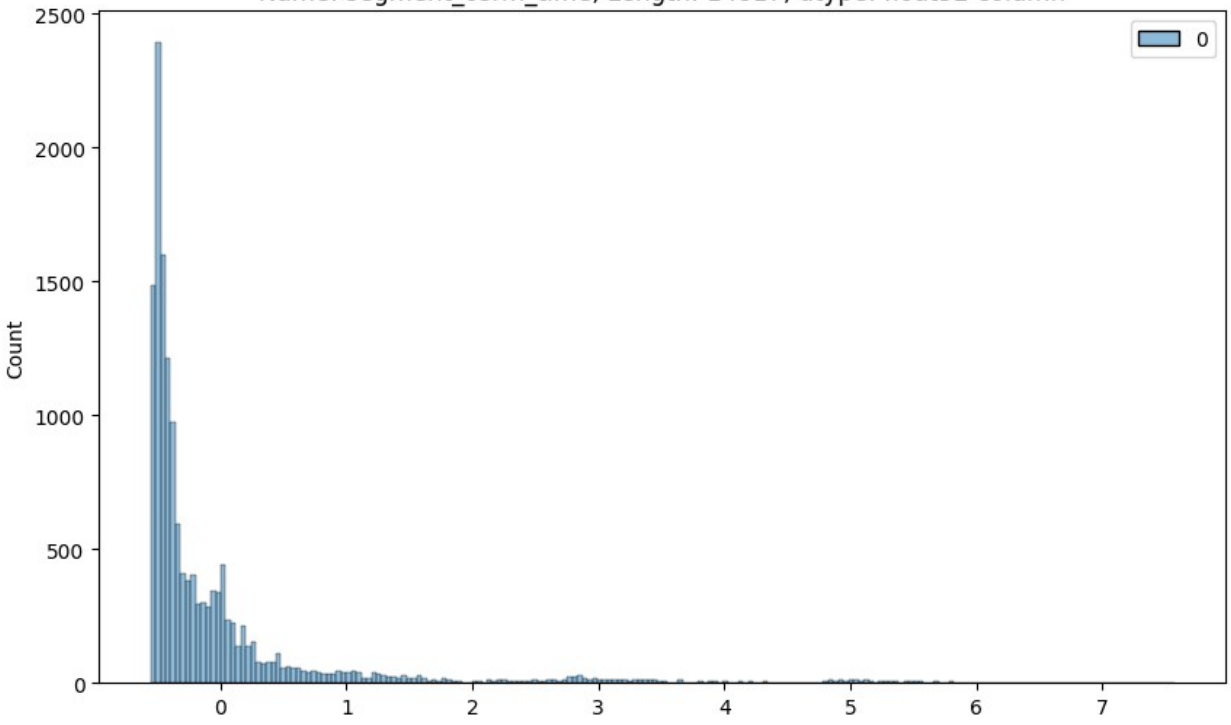

```
plt.figure(figsize = (10, 6))
scaler = StandardScaler()
scaled =
scaler.fit_transform(df2['segment_actual_time'].to_numpy().reshape(-1,
1))
sns.histplot(scaled)
plt.title(f"Standardized {df2['segment_actual_time']} column")
plt.plot()

[]
```

```
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
```




```
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()

[]
```

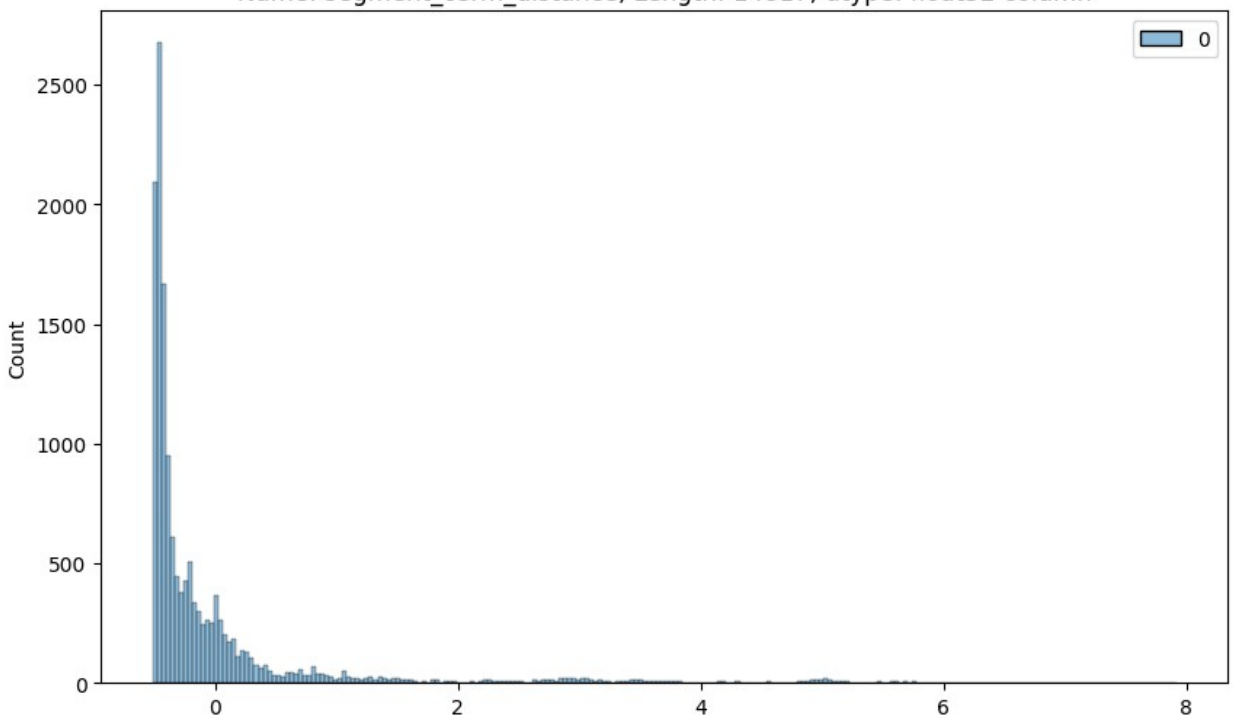
```
Standardized 0       1008.0
             1         65.0
             2       1941.0
             3         16.0
             4        115.0
                       ...
         14812         62.0
         14813         11.0
         14814         88.0
         14815        221.0
         14816   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




```
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()
```

```
[]
```

```
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

Valuable Business Insights:

Data Coverage:

- The dataset spans from '2018-09-12 00:00:16' to '2018-10-08 03:00:24'.

Unique Identifiers:

- There are 14,817 unique trip IDs, 1,508 unique source centers, 1,481 unique destination centers, 690 unique source cities, and 806 unique destination cities.

Data Composition:

- The dataset primarily contains testing data rather than training data.

Route Type:

- The most common route type is Carting.

Missing Data:

- There are 14 unique location IDs missing from the dataset.

Trip Timing:

- Trip activity increases after noon, peaks at 10 P.M., and then decreases.

Weekly Trip Volume:

- The highest number of trips were created in the 38th week. Monthly Order Trends:
- Most orders are placed mid-month, indicating higher customer activity during this period.

Source States:

- The majority of orders originate from Maharashtra, Karnataka, Haryana, Tamil Nadu, and Telangana.

Top Source Cities:

- Mumbai, Gurgaon Delhi, Bengaluru, and Bhiwandi are the leading source cities, suggesting a strong seller base in these locations.

Top Destination States:

- The highest number of trips end in Maharashtra, followed by Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh, indicating significant order volumes in these states.

Top Destination Cities:

- The majority of trips conclude in Mumbai, followed by Bengaluru, Gurgaon, Delhi, and Chennai, reflecting high order volumes in these cities.

Key Destination Cities:

- Key destination cities include Bengaluru, Mumbai, Gurgaon, Bangalore, and Delhi.

Feature Similarity:

- The features start_scan_to_end_scan and od_total_time (created feature) are statistically similar.

Feature Disparity:

- The features actual_time and osrm_time are statistically different.

Recommendations

Recommendations:

- Enhance OSRM Trip Planning System:
  - The OSRM trip planning system requires improvements to address existing discrepancies. Ensuring the routing engine is configured for optimal results will benefit transporters by providing more accurate routes.
- Align OSRM Time with Actual Time:
  - The observed difference between `osrm_time` and `actual_time` needs to be minimized. The team should focus on reducing this gap to enhance delivery time predictions, thereby providing customers with more accurate expected delivery times.
- Ensure Route Accuracy:
  - There is a discrepancy between the `osrm_distance` and the actual distance covered. This could be due to delivery personnel not following the predefined route or the OSRM devices not accurately predicting the route considering distance, traffic, and other factors. The team needs to investigate and address this issue to prevent late deliveries.
- Strengthen Key Corridors:
  - Most orders originate from or are delivered to states like Maharashtra, Karnataka, Haryana, and Tamil Nadu. Enhancing the existing corridors in these areas can further improve market penetration and efficiency.
- Customer Profiling for Major States:
  - Conduct customer profiling for customers in Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh. Understanding why these states generate major orders will help improve the buying and delivery experience for customers.
- Plan for State-Specific Conditions:
  - Consider state-specific conditions such as heavy traffic and poor terrain. This information will be valuable for planning and catering to demand during peak festival seasons, ensuring smoother operations and better customer satisfaction.**Customer Profiling for Major States:**
  - Conduct customer profiling for customers in Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh. Understanding why these states generate major orders will help improve the buying and delivery experience for customers.