Business Case: Delhivery - Feature Engineering

DELIVERY DATA ANALYSIS:

This case study focuses on the logistics sector, specifically examining a leading Indian company in this domain. The study aims to enhance operational efficiency and service quality through in-depth data analysis.

OBJECTIVE

The main objective of feature engineering for Delhivery is to identify, create, and optimize relevant features from the available data sources. This will help in improving the accuracy and effectiveness of various predictive models and analytical tools used in delivery operations. This involves cleaning, manipulating, and understanding large sets of logistics data.

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

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

```
import pandas as pd
df = pd.read_csv("delhivery_data.csv")
df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destin
0	training	35:36.5	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	I
1	training	35:36.5	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	I
2	training	35:36.5	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	I
3	training	35:36.5	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	I
4	training	35:36.5	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	I

5 rows × 24 columns

```
df.shape
```

```
(7178, 24)
```

```
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
```

```
not_required_columns = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']
df=df.drop(columns = not_required_columns)
```

```
df.nunique()
```

```
data                                   2
trip_creation_time                 14817
route_schedule_uuid                 1504
route_type                             2
trip_uuid                          14817
source_center                       1508
source_name                         1498
destination_center                  1481
destination_name                    1468
od_start_time                      26369
od_end_time                        26369
start_scan_to_end_scan              1915
actual_distance_to_destination    144515
actual_time                         3182
osrm_time                           1531
osrm_distance                     138046
segment_actual_time                  747
segment_osrm_time                    214
segment_osrm_distance             113799
dtype: int64
```

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

```
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"])
```

```
df["od_start_time"] = pd.to_datetime(df["od_start_time"])
```

```
df["od_end_time"] = pd.to_datetime(df["od_end_time"])
```

```
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  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 19.1+ MB
```

```
time_period = (df["od_end_time"].max(),df["trip_creation_time"].min())
time_period
```

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

Total 26 days of data are given in the dataset

1. Basic Data cleaning and exploration

```
import numpy as np
np.any(df.isnull())
```

```
True
```

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

```
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
```

```
df.duplicated()
len(df[df.duplicated()])
```

The number of duplicated values in the dataset are 0

```
missed_source_name = df.loc[df["source_name"].isnull(),"source_center"].unique()
missed_source_name
```

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

```
missed_destination_name = df.loc[df["destination_name"].isnull(),"destination_center"].unique()
missed_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()].isin(missed_destination_name))
```

```
False
```

Treating missing names and destination names

```
count = 1
for i in missed_destination_name:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.loc[df['destination_center'] == i, 'destination_name'].replace
    count += 1
print(count)
```

```
14
```

```
d = {}
for i in missed_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 missed_source_name:
    df.loc[df['source_center'] == i, 'source_name'] = df.loc[df['source_center'] == i, 'source_name'].replace(np.nan, d2[i])
```

```
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
```

```
df.describe()
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_tim
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.00000
mean	961.262986	234.073372	416.927527	213.868272	284.771297	36.19611
std	1037.012769	344.990009	598.103621	308.011085	421.119294	53.57115
min	20.000000	9.000045	9.000000	6.000000	9.008200	-244.00000
25%	161.000000	23.355874	51.000000	27.000000	29.914700	20.00000
50%	449.000000	66.126571	132.000000	64.000000	78.525800	29.00000
75%	1634.000000	286.708875	513.000000	257.000000	343.193250	40.00000
max	7898.000000	1927.447705	4532.000000	1686.000000	2326.199100	3051.00000

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

	route_schedule_uuid	trip_uuid	source_center	source_name	destination_center	destination_name
count	144867	144867	144867	144867	144867	144867
unique	1504	14817	1508	1508	1481	1481
top	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)

merging of rows and aggregation of fields

```
df1 = df.groupby(by = ['trip_uuid', 'source_center', 'destination_center'], 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.loc[df1["trip_uuid"]=="trip-153741093647649320"]
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name
10374	trip-153741093647649320	IND388121AAA	IND388620AAB	training	Carting	2018-09-20 02:35:36.476840	Anand_VUNagar_DC (Gujarat)
10375	trip-153741093647649320	IND388620AAB	IND388320AAA	training	Carting	2018-09-20 02:35:36.476840	Khambhat_MotvdDPP_D (Gujarat)

Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

```
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.head()
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)

```
df2.loc[df2["trip_uuid"]=="trip-153741093647649320"]
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name
5919	trip-153741093647649320	IND388121AAA	IND388320AAA	training	Carting	2018-09-20 02:35:36.476840	Anand_VUNagar_DC (Gujarat)

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

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

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

```
df2["state"] = df2["source_name"].apply(name_of_the_state)
df2["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)
```

```
def name_of_the_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]
```

```
df2["city"] = df2["source_name"].apply(name_of_the_city)
df2["city"].unique()[:50]
```

```
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'],
      dtype=object)
```

```
def name_of_the_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["place"] = df2["source_name"].apply(name_of_the_place)
df2["place"].unique()[:50]
```

```
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'], dtype=object)
```

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

```
df2["destination_state"] = df2["destination_name"].apply(name_of_the_state)
df2["destination_state"].unique()
```

```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
       'Madhya Pradesh', 'Assam', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Dadra and Nagar Haveli', 'Orissa',
       'Bihar', 'Jharkhand', 'Goa', 'Uttarakhand', 'Himachal Pradesh',
       'Kerala', 'Arunachal Pradesh', 'Mizoram', 'Chhattisgarh',
       'Jammu & Kashmir', 'Nagaland', 'Meghalaya', 'Tripura',
       'location_13', 'location_6', 'location_2', 'location_7',
       'location_3', 'location_5', 'location_12', 'location_11',
       'Daman & Diu'], dtype=object)
```

```
df2["destination_city"] = df2["destination_name"].apply(name_of_the_city)
df2["destination_city"].unique()[:50]
```

```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Sandur', 'Chennai',
       'Bengaluru', 'Surat', 'Delhi', 'PNQ', 'Faridabad', 'Ratnagiri',
       'Bangalore', 'Hyderabad', 'Aland', 'Jaipur', 'Satna', 'Guwahati',
       'Bareli', 'Nashik', 'Hooghly', 'Sivasagar', 'Palani', 'Jalandhar',
       'Chandigarh', 'Yavatmal', 'Sangola', 'Kolkata', 'Savner',
       'Kurnool', 'Bhatinda', 'Bhiwandi', 'Barnala', 'Murbad', 'Kadaba',
       'Gulbarga', 'Naraingarh', 'Ludhiana', 'Kadi', 'Jabalpur',
       'Gangakher', 'Bankura', 'Silvassa', 'Porbandar', 'Jetpur',
       'Khammam', 'Mehsana', 'Katni', 'Una', 'Malavalli'], dtype=object)
```

```
df2["destination_place"] = df2["destination_name"].apply(name_of_the_place)
df2["destination_place"].unique()[:50]
```

```
array(['Central_H_6', 'ChikaDPP_D', 'Bilaspur_HB', 'MiraRd_IP',
       'WrdN1DPP_D', 'Poonamallee', 'Vandalur_Dc', 'HBR Layout PC',
       'Central_D_3', 'Bhogal', 'unknown_place', 'MjgaonRd_D',
       'Nelmngla_H', 'Uppal_I', 'RazaviRd_D', 'Central_I_7',
       'Central_I_2', 'Hub', 'SourvDPP_D', 'Varachha_DC', 'TgrniaRD_I',
       'DC', 'Gokulam_D', 'Babupaty_D', 'Bomsndra_HB', 'Alwal_I',
       'RjndraRd_D', 'Mehmdpur_H', 'Sanpada_I', 'JajuDPP_D',
       'Central_DPP_2', 'Dankuni_HB', 'Wagodha_D', 'AbbasNgr_I',
       'Balabhgarh_DPC', 'DPC', 'Mankoli_HB', 'Shamshbd_H', 'SnkunDPP_D',
       'Kharar_DC', 'AnugrDPP_D', 'Nehrugnj_I', 'Ward2DPP_D',
       'MilrGanj_HB', 'KaranNGR_D', 'Adhartal_IP', 'Poonamallee_HB',
       'Busstand_D', 'BhowmDPP_D', 'Samrvrni_D'], dtype=object)
```

Trip_creation_time: Extract features like month, year and day etc

```
df2['trip_creation_hour'] = df2['trip_creation_time'].dt.hour
df2['trip_creation_hour'].head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: trip_creation_hour, dtype: int64
```

```
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'].head()
```

```
0    12
1    12
2    12
3    12
4    12
Name: trip_creation_day, dtype: int64
```

```
df2['trip_creation_week'] = df2['trip_creation_time'].dt.isocalendar().week
df2['trip_creation_week'].head()
```

```
0    37
1    37
2    37
3    37
4    37
Name: trip_creation_week, dtype: UInt32
```

```
df2['trip_creation_month'] = df2['trip_creation_time'].dt.month
df2['trip_creation_month'].head()
```

```
0    9
1    9
2    9
3    9
4    9
Name: trip_creation_month, dtype: int64
```

```
df2['trip_creation_year'] = df2['trip_creation_time'].dt.year
df2['trip_creation_year'].head()
```

```
0    2018
1    2018
2    2018
3    2018
4    2018
Name: trip_creation_year, dtype: int64
```

Structure of the Data after 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  float64
 11  actual_time                     14817 non-null  float64
 12  osrm_time                       14817 non-null  float64
 13  osrm_distance                   14817 non-null  float64
 14  segment_actual_time             14817 non-null  float64
 15  segment_osrm_time               14817 non-null  float64
 16  segment_osrm_distance           14817 non-null  float64
 17  state                           14817 non-null  object
 18  city                            14817 non-null  object
 19  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_hour              14817 non-null  int64
 24  trip_creation_date              14817 non-null  datetime64[ns]
 25  trip_creation_day               14817 non-null  int64
 26  trip_creation_week              14817 non-null  UInt32
 27  trip_creation_month             14817 non-null  int64
 28  trip_creation_year              14817 non-null  int64
dtypes: UInt32(1), category(2), datetime64[ns](2), float64(9), int64(4), object(11)
memory usage: 3.0+ MB
```

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

	count	mean	std	min	25%	50%	75%	max
od_total_time	14817.0	531.69763	658.868223	23.46	149.93	280.77	638.2	7898.55
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.0	149.0	280.0	637.0	7898.0
actual_distance_to_destination	14817.0	164.477838	305.388147	9.002461	22.837239	48.474072	164.583208	2186.531787
actual_time	14817.0	357.143754	561.396157	9.0	67.0	149.0	370.0	6265.0
osrm_time	14817.0	161.384018	271.360995	6.0	29.0	60.0	168.0	2032.0
osrm_distance	14817.0	204.344689	370.395573	9.0729	30.8192	65.6188	208.475	2840.081
segment_actual_time	14817.0	353.892286	556.247965	9.0	66.0	147.0	367.0	6230.0
segment_osrm_time	14817.0	180.949787	314.542047	6.0	31.0	65.0	185.0	2564.0
segment_osrm_distance	14817.0	223.201161	416.628374	9.0729	32.6545	70.1544	218.8024	3523.6324
trip_creation_hour	14817.0	12.449821	7.986553	0.0	4.0	14.0	20.0	23.0
trip_creation_day	14817.0	18.37079	7.893275	1.0	14.0	19.0	25.0	30.0
trip_creation_week	14817.0	38.295944	0.967872	37.0	38.0	38.0	39.0	40.0
trip_creation_month	14817.0	9.120672	0.325757	9.0	9.0	9.0	9.0	10.0
trip_creation_year	14817.0	2018.0	0.0	2018.0	2018.0	2018.0	2018.0	2018.0

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

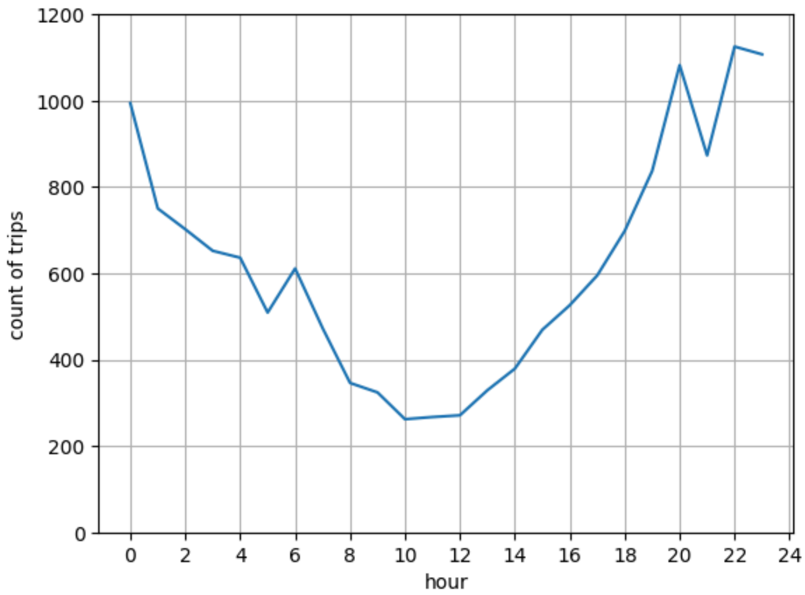
	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
source_center	14817	938	IND000000ACB	1063
destination_center	14817	1042	IND000000ACB	821
source_name	14817	938	Gurgaon_Bilaspur_HB (Haryana)	1063
destination_name	14817	1042	Gurgaon_Bilaspur_HB (Haryana)	821
state	14817	34	Maharashtra	2714
city	14817	690	Mumbai	1442
place	14817	761	Bilaspur_HB	1063
destination_state	14817	39	Maharashtra	2561
destination_city	14817	806	Mumbai	1548
destination_place	14817	850	Bilaspur_HB	821

Trips created on hourly basis

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

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

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.lineplot(data=df_hourly_trip,
             x=df_hourly_trip["trip_creation_hour"],
             y=df_hourly_trip["trip_uuid"])
plt.yticks(np.arange(0,1400,200))
plt.xticks(np.arange(0,26,2))
plt.xlabel("hour")
plt.ylabel("count of trips")
plt.grid("both")
```

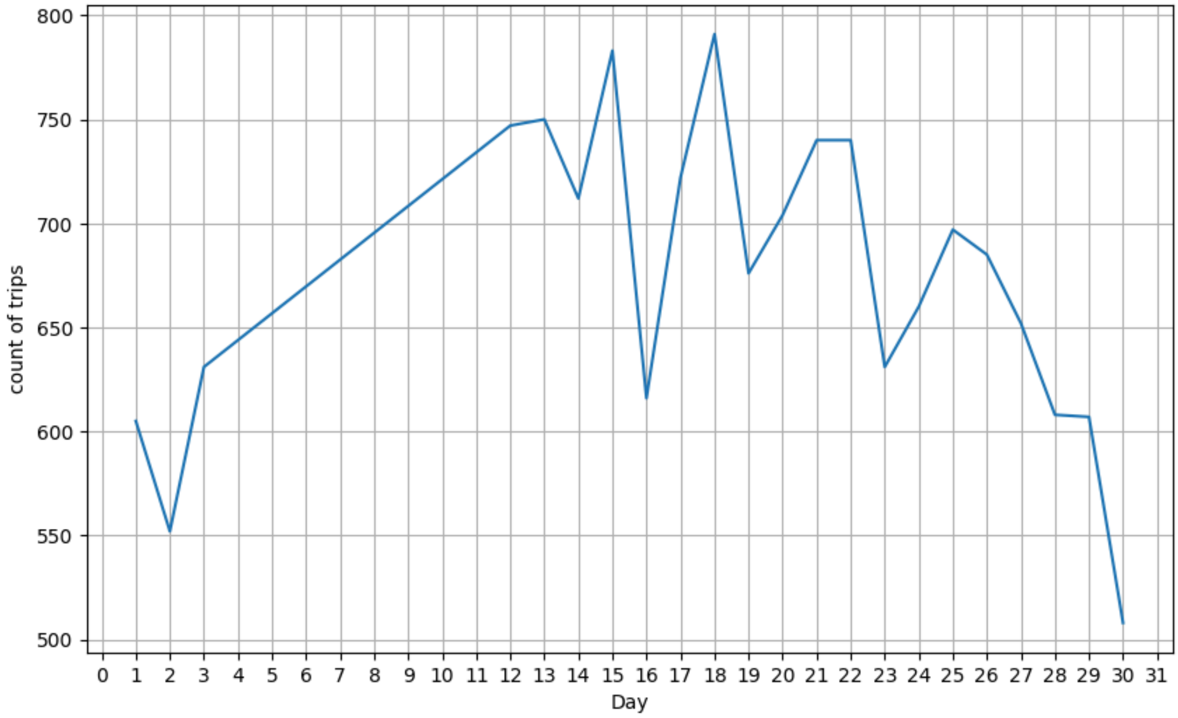


daily trips

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

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

```
plt.figure(figsize=(10,6))
sns.lineplot(data=df_daily_trip,
             x=df_daily_trip["trip_creation_day"],
             y=df_daily_trip["trip_uuid"])
plt.xlabel("Day")
plt.xticks(np.arange(0,32))
plt.ylabel("count of trips")
plt.grid("both")
```

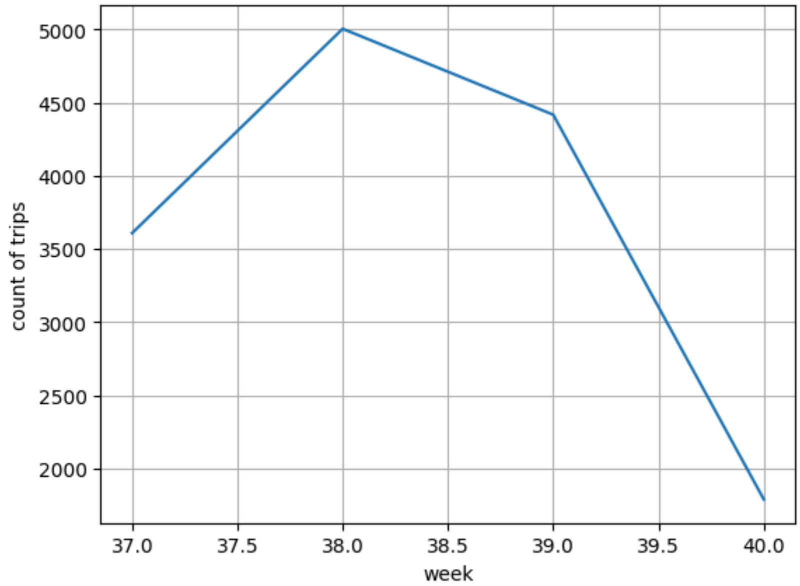


Weekly trips

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

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

```
sns.lineplot(data=df_weekly_trip,
             x=df_weekly_trip["trip_creation_week"],
             y=df_weekly_trip["trip_uuid"])
plt.xlabel("week")
plt.ylabel("count of trips")
plt.grid("both")
```

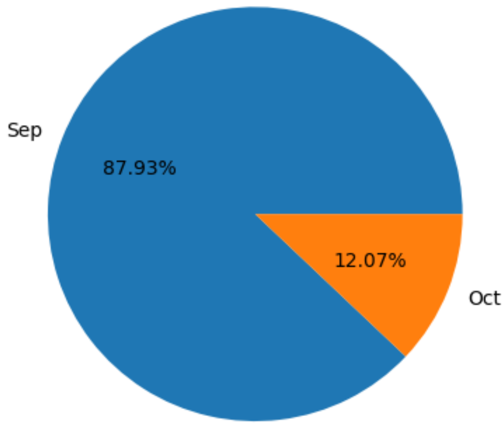


Monthly trips

```
df_monthly_trip = df2.groupby(by="trip_creation_month")["trip_uuid"].count().to_frame().reset_index()
df_monthly_trip["percentage"]=np.round(df_monthly_trip["trip_uuid"]*100/df_monthly_trip["trip_uuid"].sum(),2)
df_monthly_trip
```

	trip_creation_month	trip_uuid	percentage
0	9	13029	87.93
1	10	1788	12.07

```
plt.pie(x=df_monthly_trip["percentage"],
        labels=["Sep","Oct"],
        autopct = '%.2f%%')
plt.show()
```

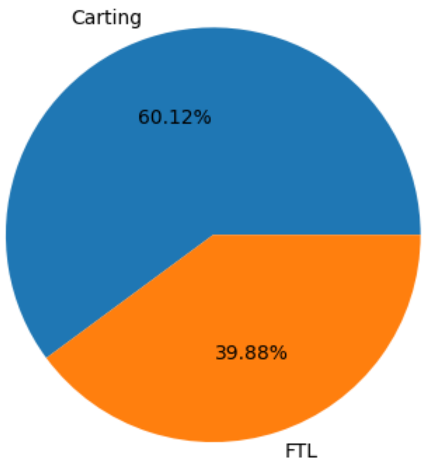


Distribution of orders (Route wise)

```
df_routewise_trip=df2.groupby(by="route_type")["trip_uuid"].count().to_frame().reset_index()
df_routewise_trip["percentage"] = np.round(df_routewise_trip["trip_uuid"]*100/df_routewise_trip["trip_uuid"].sum(),2)
df_routewise_trip
```

	route_type	trip_uuid	percentage
0	Carting	8908	60.12
1	FTL	5909	39.88

```
plt.pie(x=df_routewise_trip["percentage"],
        labels=["Carting","FTL"],
        autopct = "%.2f%%")
plt.show()
```



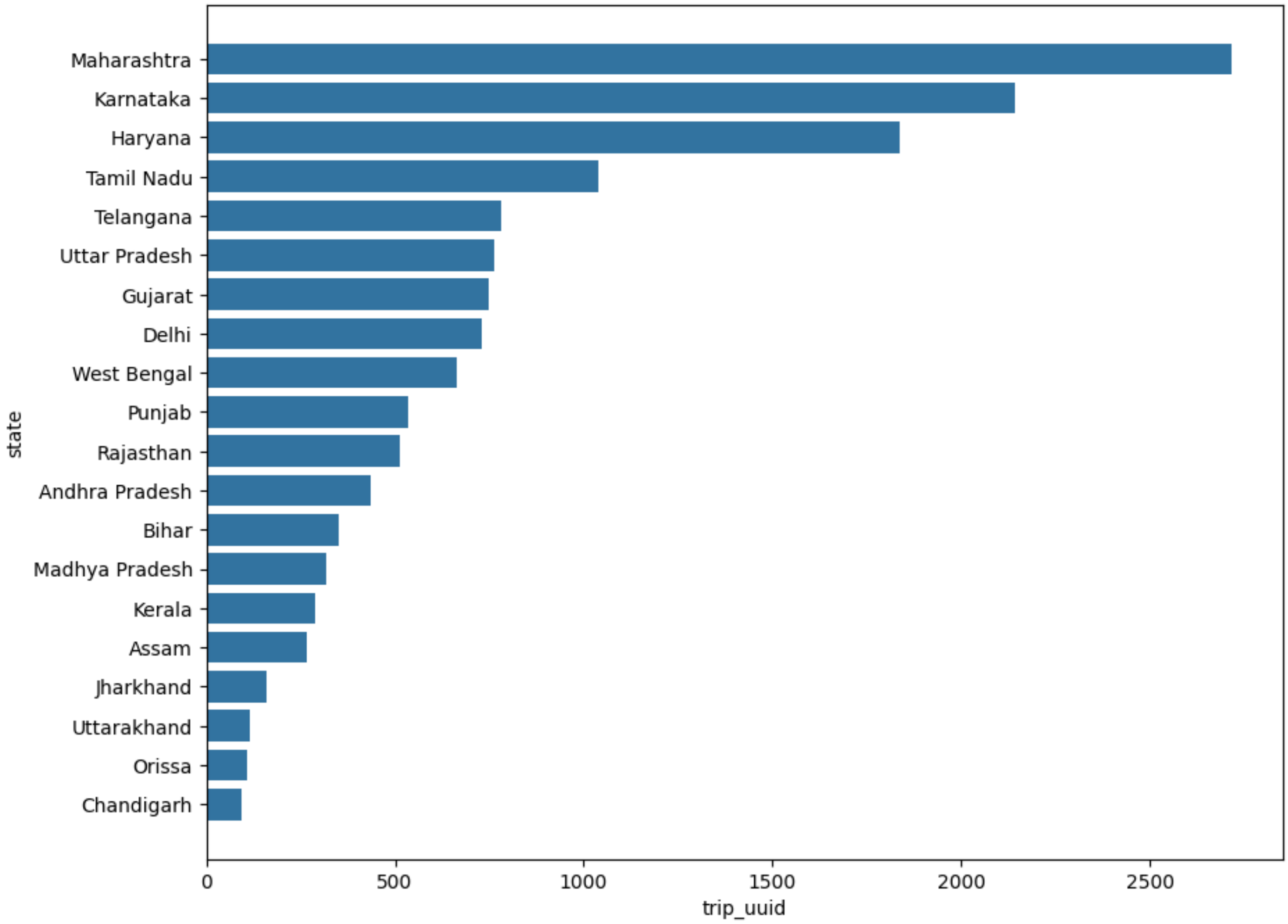
Top 20 States with Maximum Trips

```
df_source_state = df2.groupby(by="state")["trip_uuid"].count().to_frame().reset_index()
df_source_state["percentage"] = 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)[:20]
df_source_state.head()
```

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

```
plt.figure(figsize=(10,8))
sns.barplot(data = df_source_state,
            x = df_source_state["trip_uuid"],
            y = df_source_state["state"])
plt.plot()
```

[]



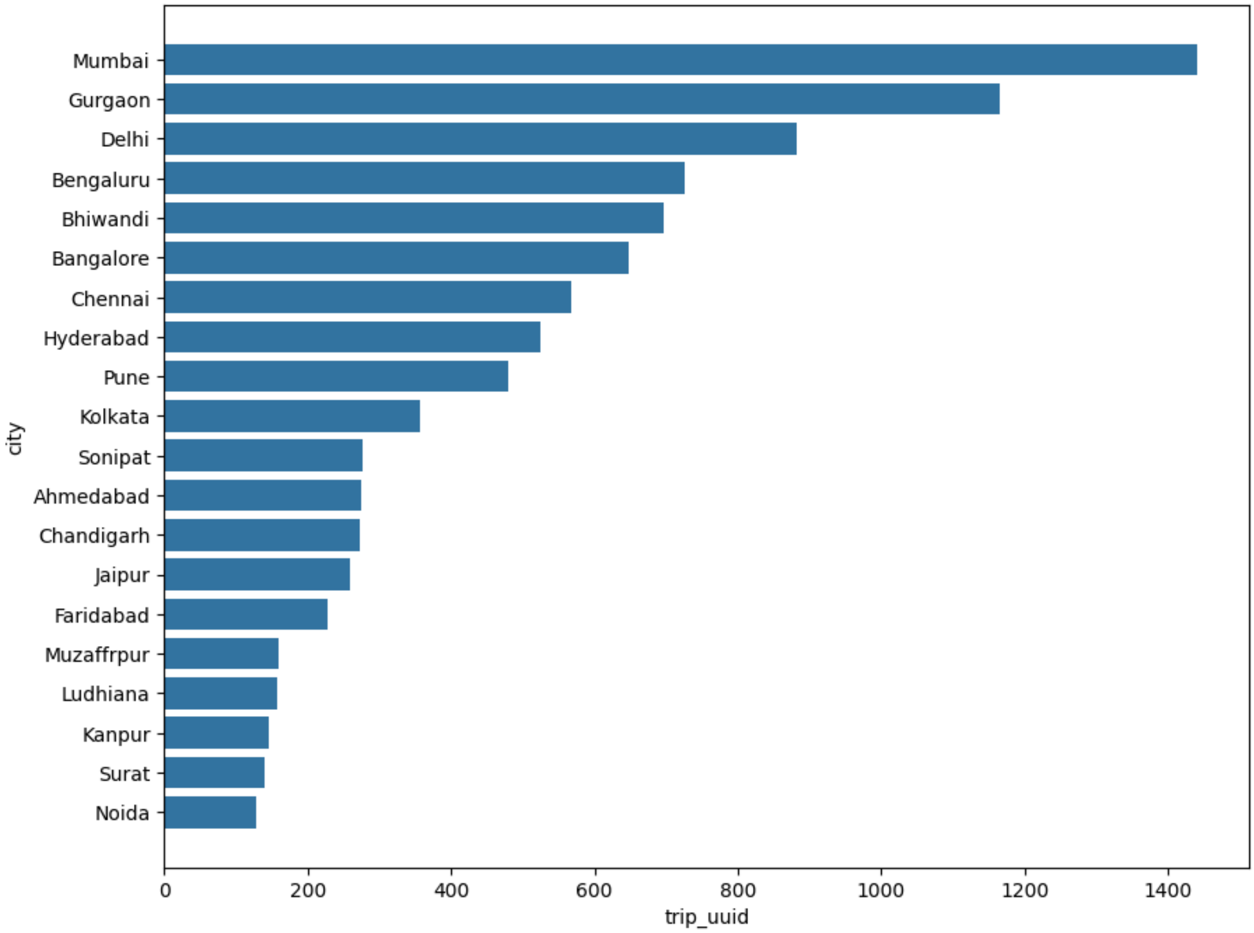
Top 20 cities with Maximum trips

```
df_source_city = df2.groupby(by="city")["trip_uuid"].count().to_frame().reset_index()
df_source_city["percentage"] = 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)[:20]
df_source_city.head()
```

	city	trip_uuid	percentage
439	Mumbai	1442	9.73
237	Gurgaon	1165	7.86
169	Delhi	883	5.96
79	Bengaluru	726	4.90
100	Bhiwandi	697	4.70

```
plt.figure(figsize=(10,8))
sns.barplot(data = df_source_city,
            x = df_source_city["trip_uuid"],
            y = df_source_city["city"])
plt.plot()
```

[]




```
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()
```

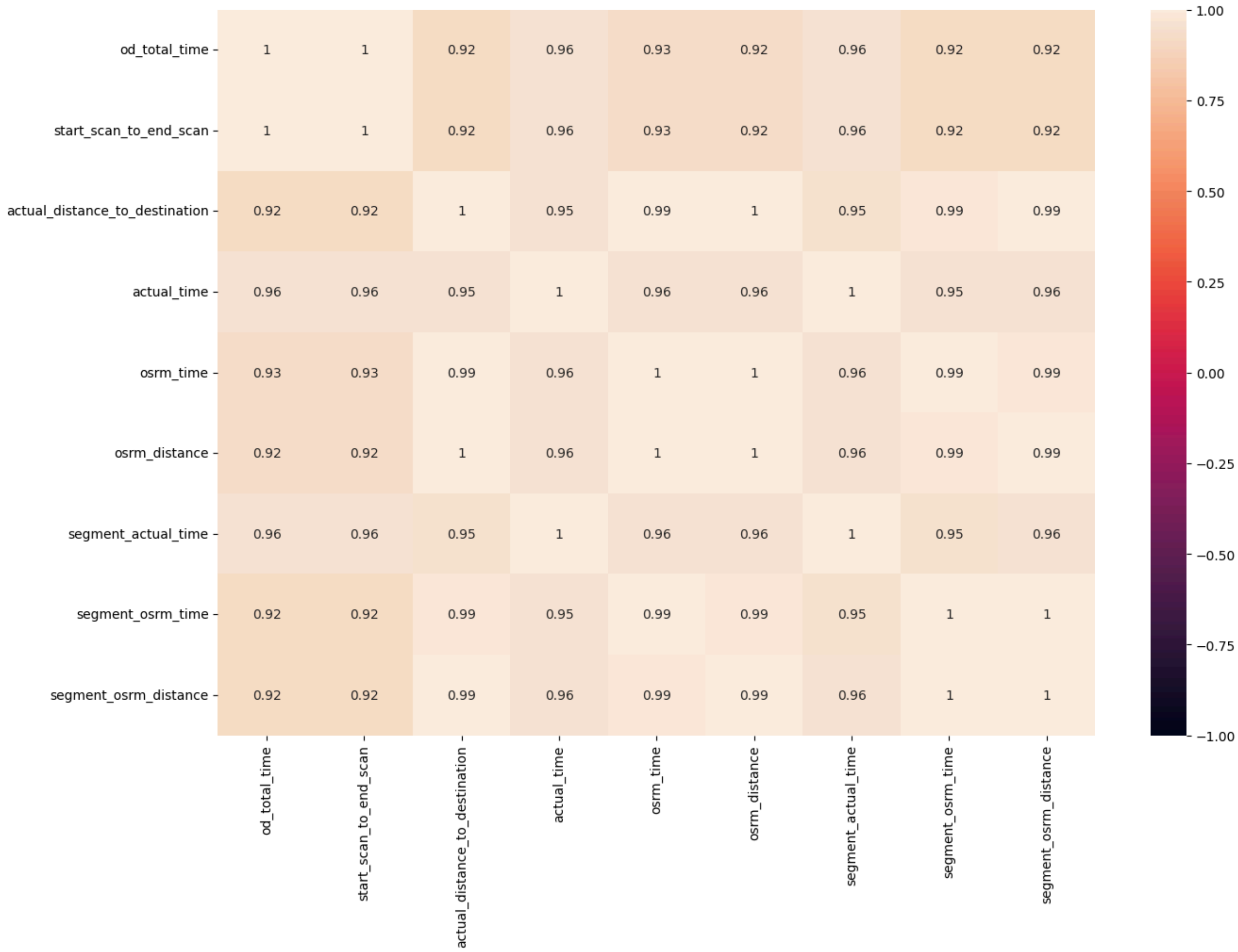
```
[]
```


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

	od_total_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	os
od_total_time	1.000000	0.999999	0.918222	0.961094	0.926516	
start_scan_to_end_scan	0.999999	1.000000	0.918308	0.961147	0.926571	
actual_distance_to_destination	0.918222	0.918308	1.000000	0.953757	0.993561	
actual_time	0.961094	0.961147	0.953757	1.000000	0.958593	
osrm_time	0.926516	0.926571	0.993561	0.958593	1.000000	
osrm_distance	0.924219	0.924299	0.997264	0.959214	0.997580	
segment_actual_time	0.961119	0.961171	0.952821	0.999989	0.957765	
segment_osrm_time	0.918490	0.918561	0.987538	0.953872	0.993259	
segment_osrm_distance	0.919199	0.919291	0.993061	0.956967	0.991608	

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

[ ]



3. In-depth analysis and feature engineering:

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

```
# Steps of Feature Engineering:

# Step 1: Set up Null & Alternate Hypothesis
# H0(Null Hypothesis): mean of od_total_time(Total trip time) and start_scan_to_end (Expected total trip time) are same.
# HA(Alternate Hypothesis): mean of od_total_time(Total trip time) and start_scan_to_end (Expected total trip time) are different

# step 2: Basic assumptions for Hypothesis
# Distribution using QQ plot
# Homogenity of variances using lavene's test

# step 3: Define Test Statistics
# 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
# alpha = 0.05

# step 5: Based on p-value we will reject or accept HO.
# p-val > alpha : Accept HO
# p-val < alpha : Reject HO


# checking mean, median and mode are equal
df2[["od_total_time","start_scan_to_end_scan"]].describe()
```

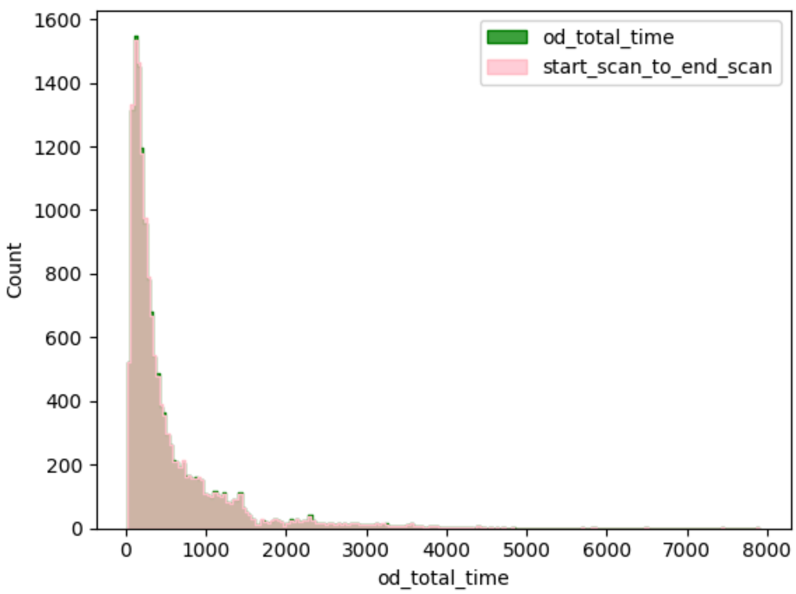
	od_total_time	start_scan_to_end_scan
count	14817.000000	14817.000000
mean	531.697630	530.810016
std	658.868223	658.705957
min	23.460000	23.000000
25%	149.930000	149.000000
50%	280.770000	280.000000
75%	638.200000	637.000000
max	7898.550000	7898.000000

Histogram

```
# If it is following normal distribution, it will form a shape like bell.
```

```
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()
```

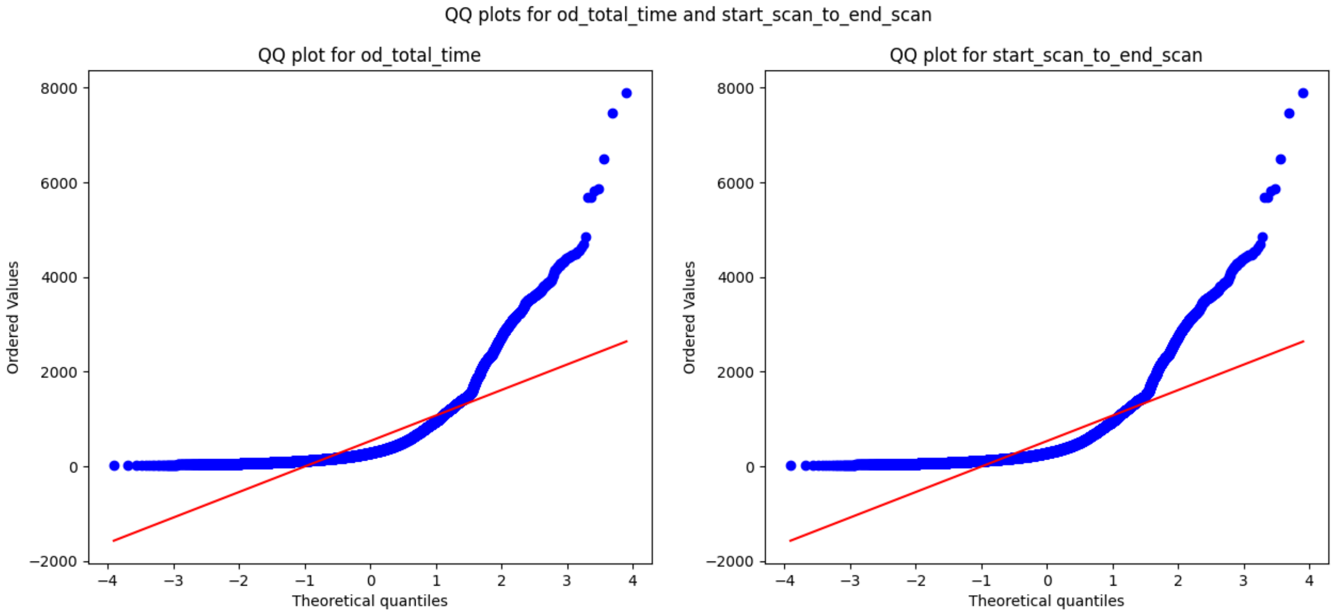
[]



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()
```

[]



From the above both plots, we can infer that it will not follow normal distribution

Applying shapiro-wilk test for normality

H_0 : The sample follows normal distribution

H_1 : The sample does not follow normal distribution

alpha = 0.05

```
test_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
```

Using box-cox transformation

```
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
/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.")
```

Homogenity 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
```

We can not perform T-test here, because the samples are not normally distributed here, we can perform non-parametric equivalent test like Mann-Whitney U rank here for two-independent samples

```
# H0: mean of od_total_time and start_scan_to_end_scan are same.
# HA: mean of od_total_time and start_scan_to_end_scan are different
alpha = 0.05
test_stat, p_value = spy.mannwhitneyu(df2['od_total_time'], df2['start_scan_to_end_scan'])
print('P-value :',p_value)
if p_value > 0.05:
  print("mean of od_total_time and start_scan_to_end_scan are same")
else:
  print("mean of od_total_time and start_scan_to_end_scan are different")
```

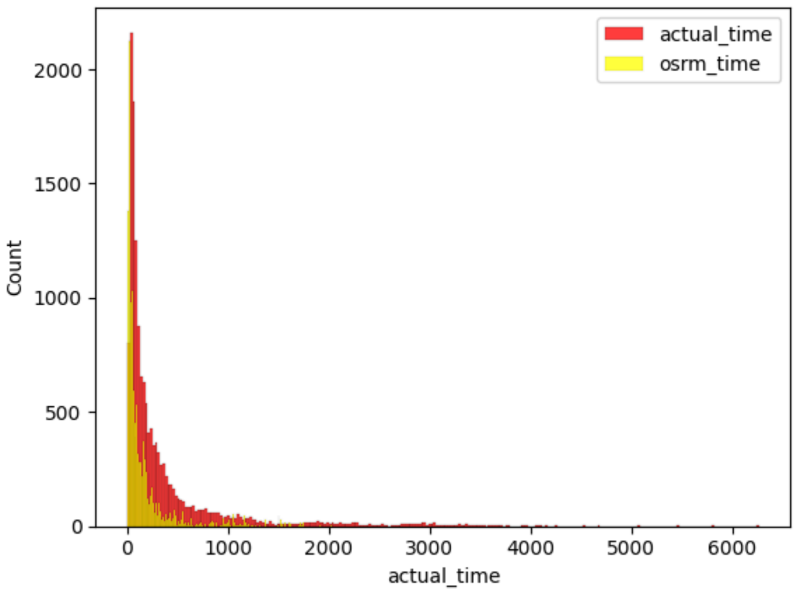
```
P-value : 0.7815123224221716
mean of od_total_time and start_scan_to_end_scan are same
```

mean of time taken by od_total_time and start_scan_to_end_scan are same

Hypothesis testing / visual analysis between actual_time and osrm_time

Visual tests to know if the sample follows normal distribution

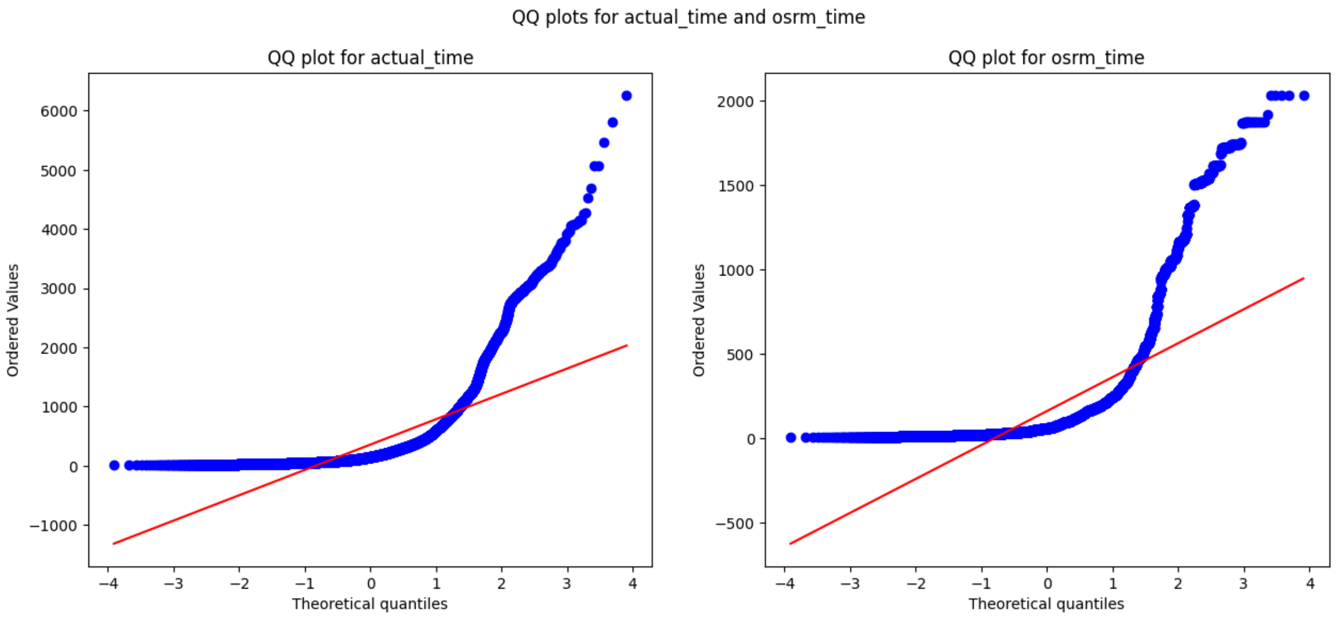
```
sns.histplot(df2["actual_time"], color="red")
sns.histplot(df2["osrm_time"], color="yellow")
plt.legend(["actual_time","osrm_time"])
plt.show()
```



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 inferred that samples do not follow normal distribution

Applying 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
```

boxcox Transformation

```
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 1.021792743086169e-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 > 500(
  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 3.543600614978861e-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 1.871297993683208e-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('mean of actual_time and osrm_time are not similar')
else:
    print('mean of actual_time and osrm_time are similar ')
```

```
p-value 0.0
mean of actual_time and osrm_time are not similar
```

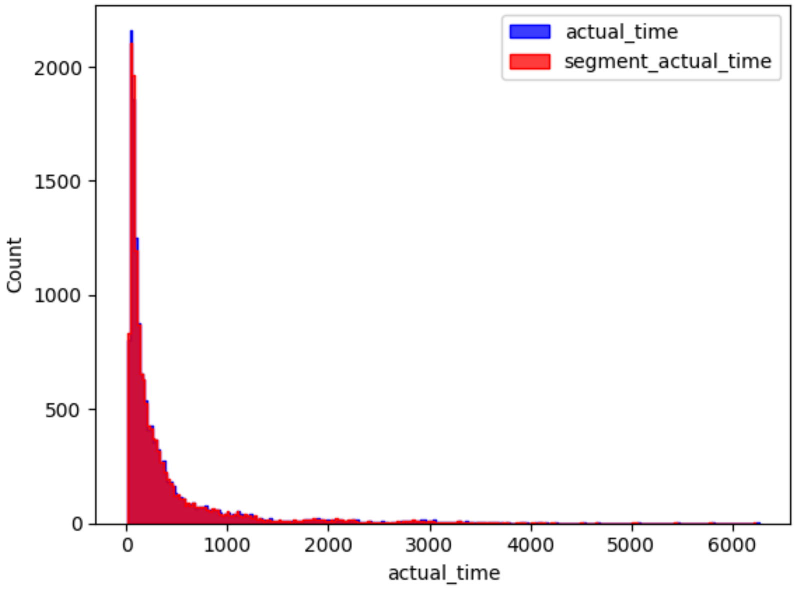
mean of actual_time and osrm_time are not similar

Do Hypothesis testing between actual_time and segment_actual_time

Histogram

```
sns.histplot(df2['actual_time'], element = 'step', color = 'blue')
sns.histplot(df2['segment_actual_time'], element = 'step', color = 'red')
plt.legend(['actual_time', 'segment_actual_time'])
plt.plot()
```

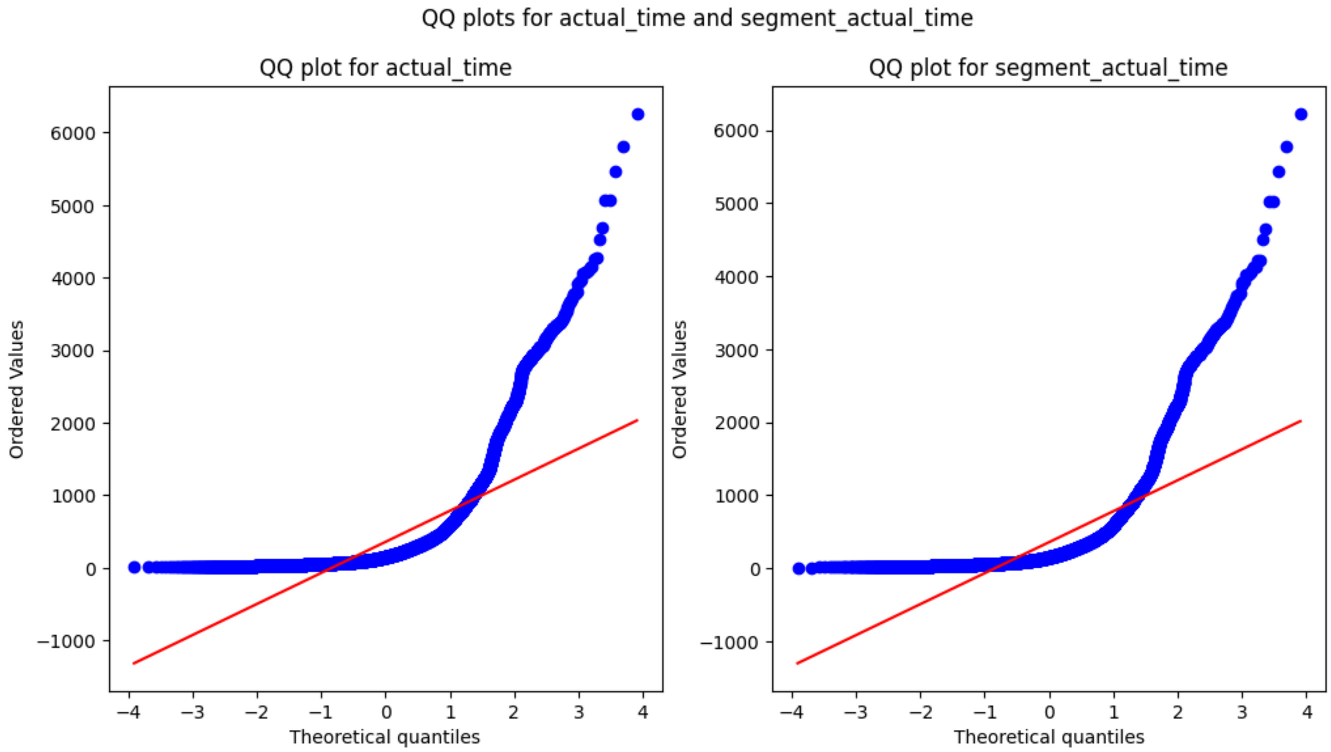
[]



- Distribution check using **QQ Plot**

```
plt.figure(figsize = (12, 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()
```

[]



Samples do not follow normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution**

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

alpha = 0.05

```
test_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 1.021792743086169e-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_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 5.696120172016859e-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 0.6955022668700895
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('mean of actual_time and segment_actual_time are not similar')
else:
    print('mean of actual_time and segment_actual_time are similar ')
```

```
p-value 0.4164235159622476
mean of actual_time and segment_actual_time are similar
```

mean of actual_time and segment_actual_time are similar

Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual 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']
df2[numerical_columns].describe().T
```

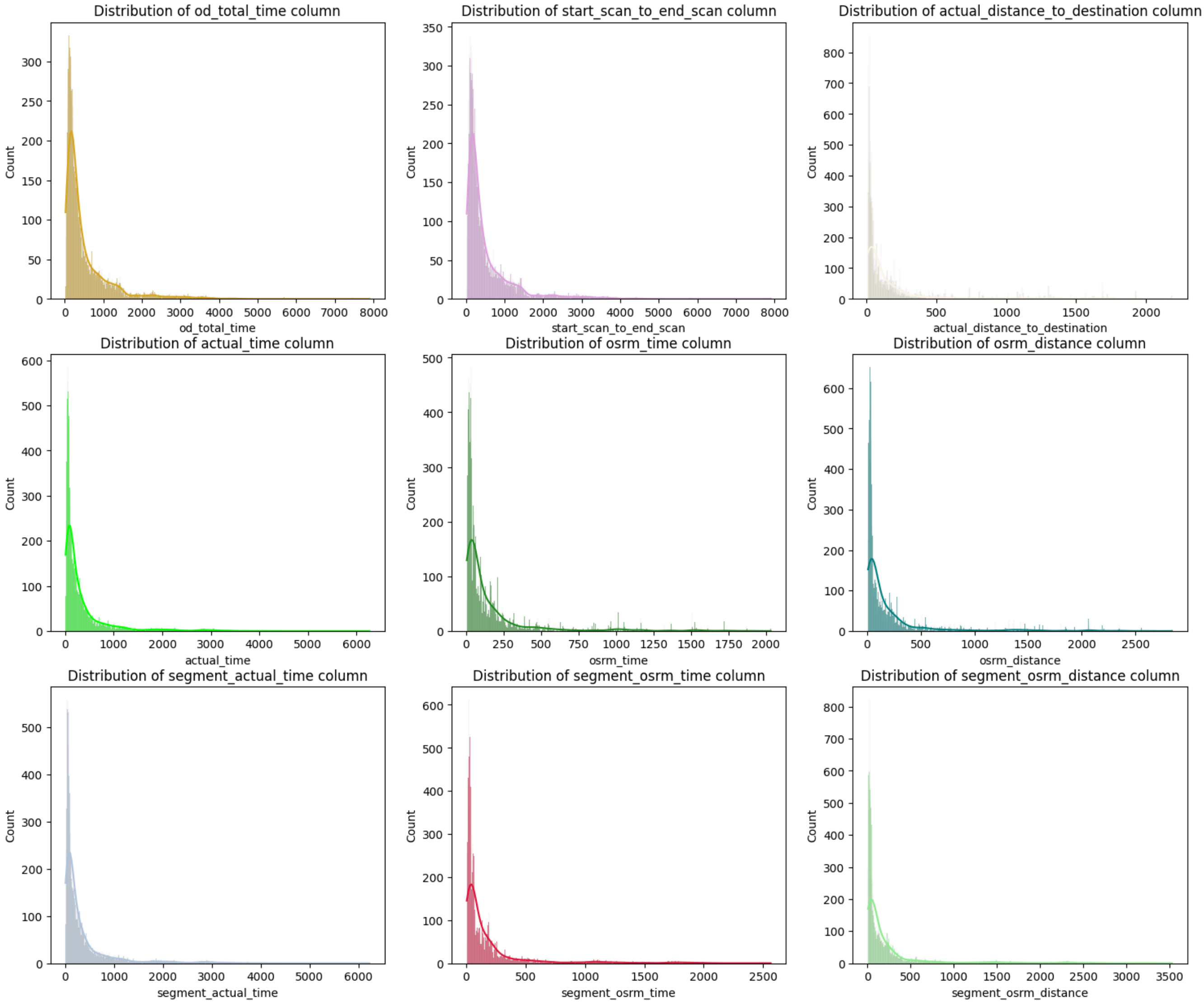
	count	mean	std	min	25%	50%	75%	max
od_total_time	14817.0	531.697630	658.868223	23.460000	149.930000	280.770000	638.200000	7898.550000
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000	280.000000	637.000000	7898.000000
actual_distance_to_destination	14817.0	164.477838	305.388147	9.002461	22.837239	48.474072	164.583208	2186.531787
actual_time	14817.0	357.143754	561.396157	9.000000	67.000000	149.000000	370.000000	6265.000000
osrm_time	14817.0	161.384018	271.360995	6.000000	29.000000	60.000000	168.000000	2032.000000
osrm_distance	14817.0	204.344689	370.395573	9.072900	30.819200	65.618800	208.475000	2840.081000
segment_actual_time	14817.0	353.892286	556.247965	9.000000	66.000000	147.000000	367.000000	6230.000000
segment_osrm_time	14817.0	180.949787	314.542047	6.000000	31.000000	65.000000	185.000000	2564.000000
segment_osrm_distance	14817.0	223.201161	416.628374	9.072900	32.654500	70.154400	218.802400	3523.632400

```
import matplotlib as mpl
colors_list = list(mpl.colors.cnames)
```

```
clr = np.random.choice(colors_list)
clr
```

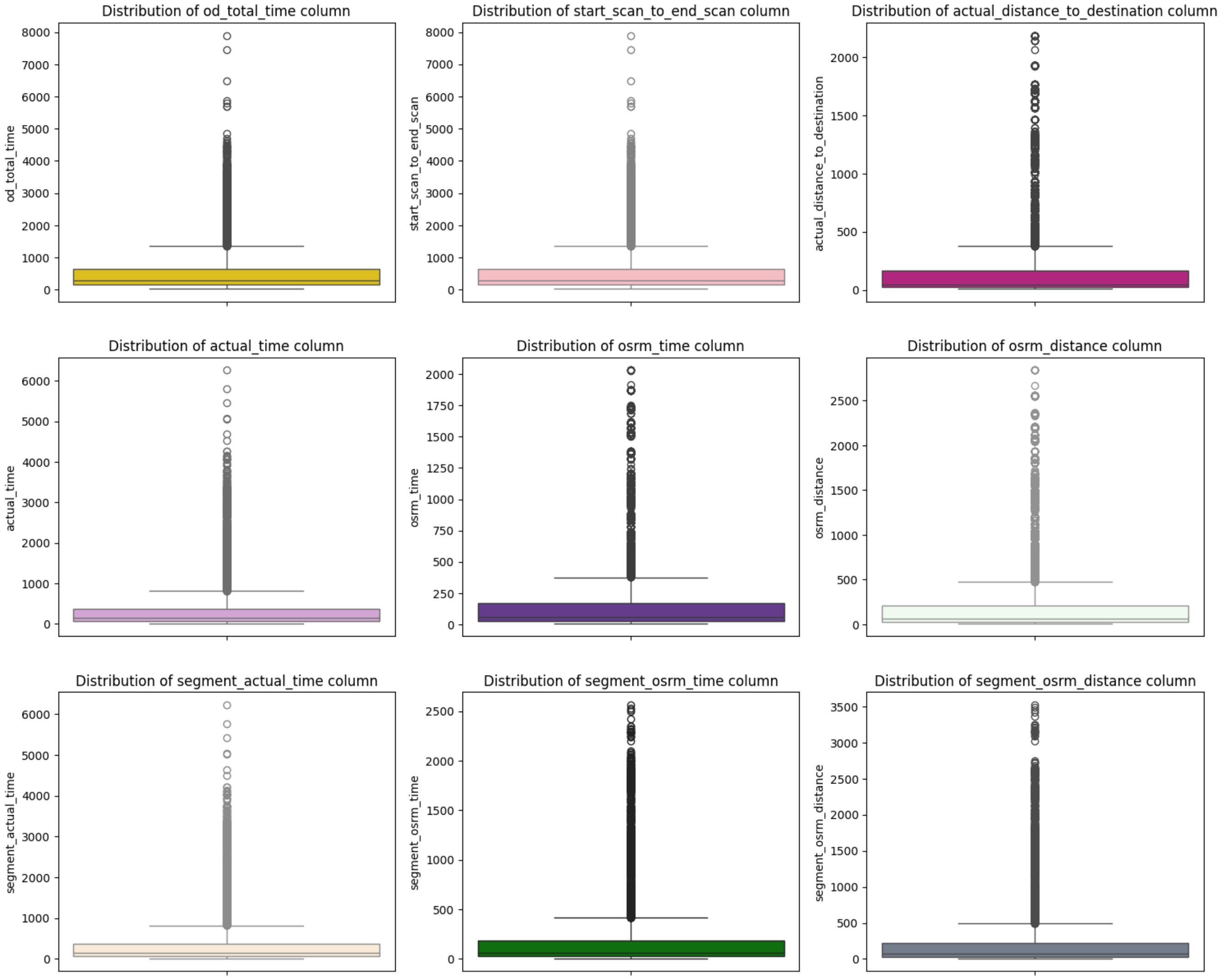
```
'lightslategrey'
```

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



As we can see from the above plots, all the histograms of numerical columns are right skewed, that means there are outliers present in the columns that needs to be cleaned.

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



It can be clearly seen in the above plots that there are outliers in all the numerical columns that can 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 : 22.83723905859321
Q3 : 164.58320763841138
IQR : 141.74596857981817
LB : -189.78171381113404
UB : 377.2021605081386
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 : 30.8192
Q3 : 208.475
IQR : 177.6558
LB : -235.6645
UB : 474.9587
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
```

One Hot encoding is not right suitable in route_type or data column, that is why we used label encoding. Label encoding also has some disadvatages like it expands our dataset, which is not needed.

Label Encoding of categorical variables

```
# Get value counts before Label encoding

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

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

```
# Perform label 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 Label encoding

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

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

```
# Get value counts of categorical variable 'data' before Label encoding

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

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

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

```
# Get value counts after label encoding

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

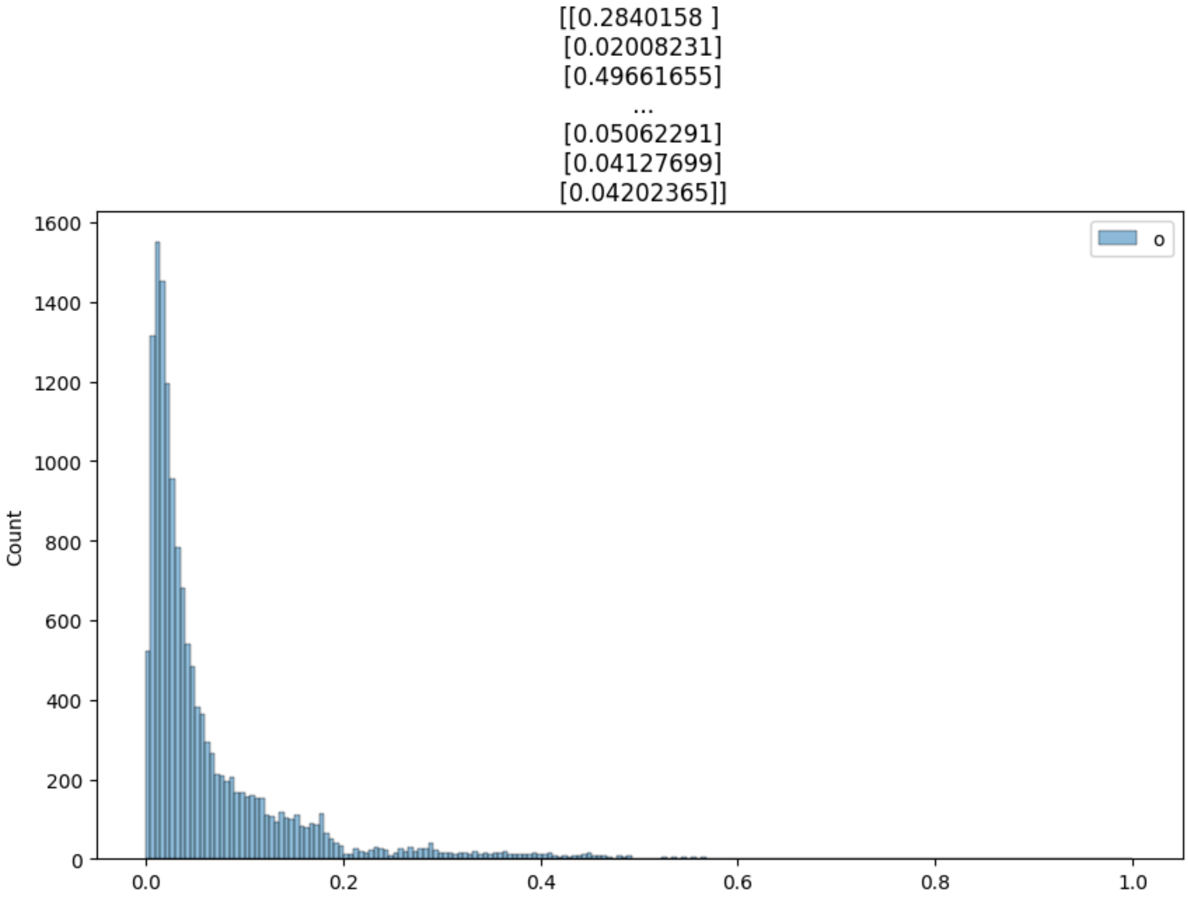
```
1    10654
0     4163
Name: data, dtype: int64
```

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

```
from sklearn.preprocessing import MinMaxScaler
```

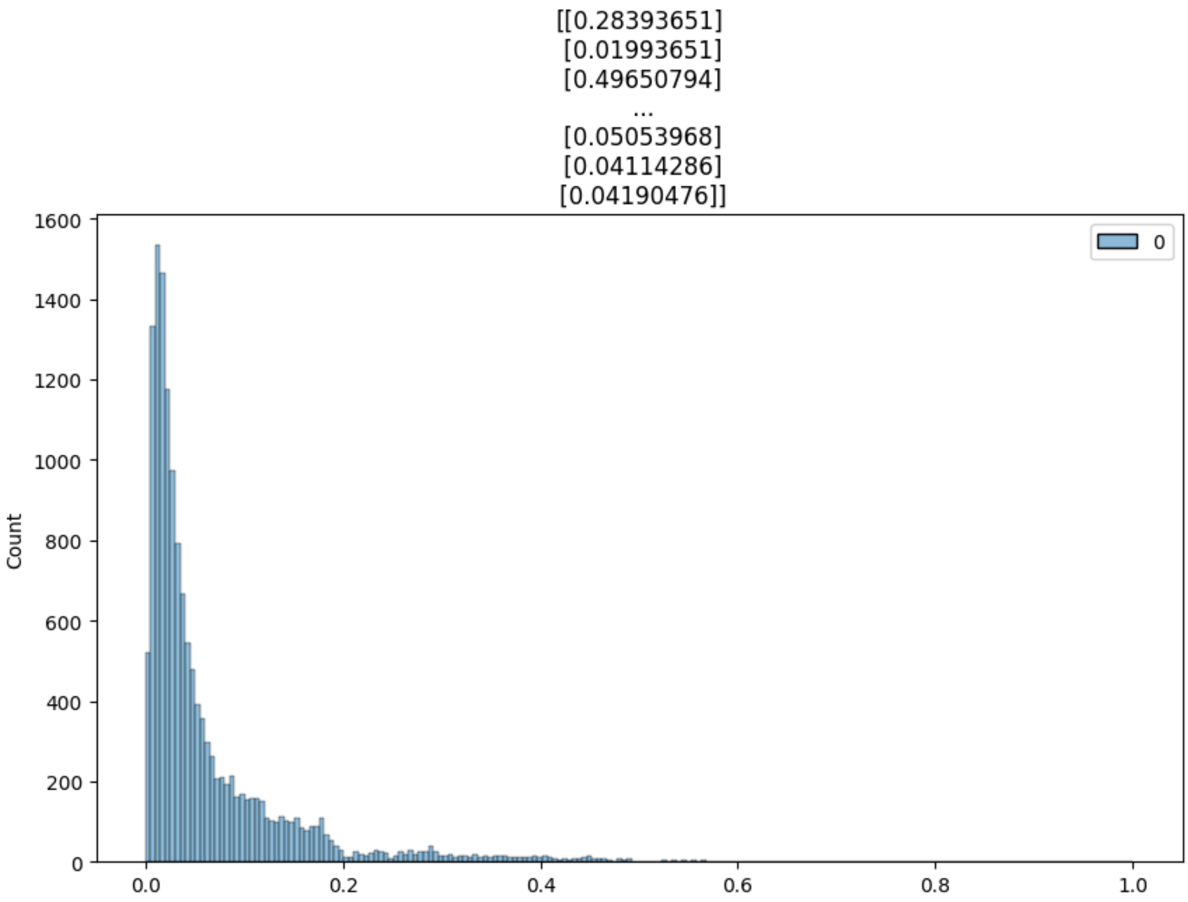
```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"{scaled}")
plt.legend('od_total_time')
plt.plot()
```

```
[]
```



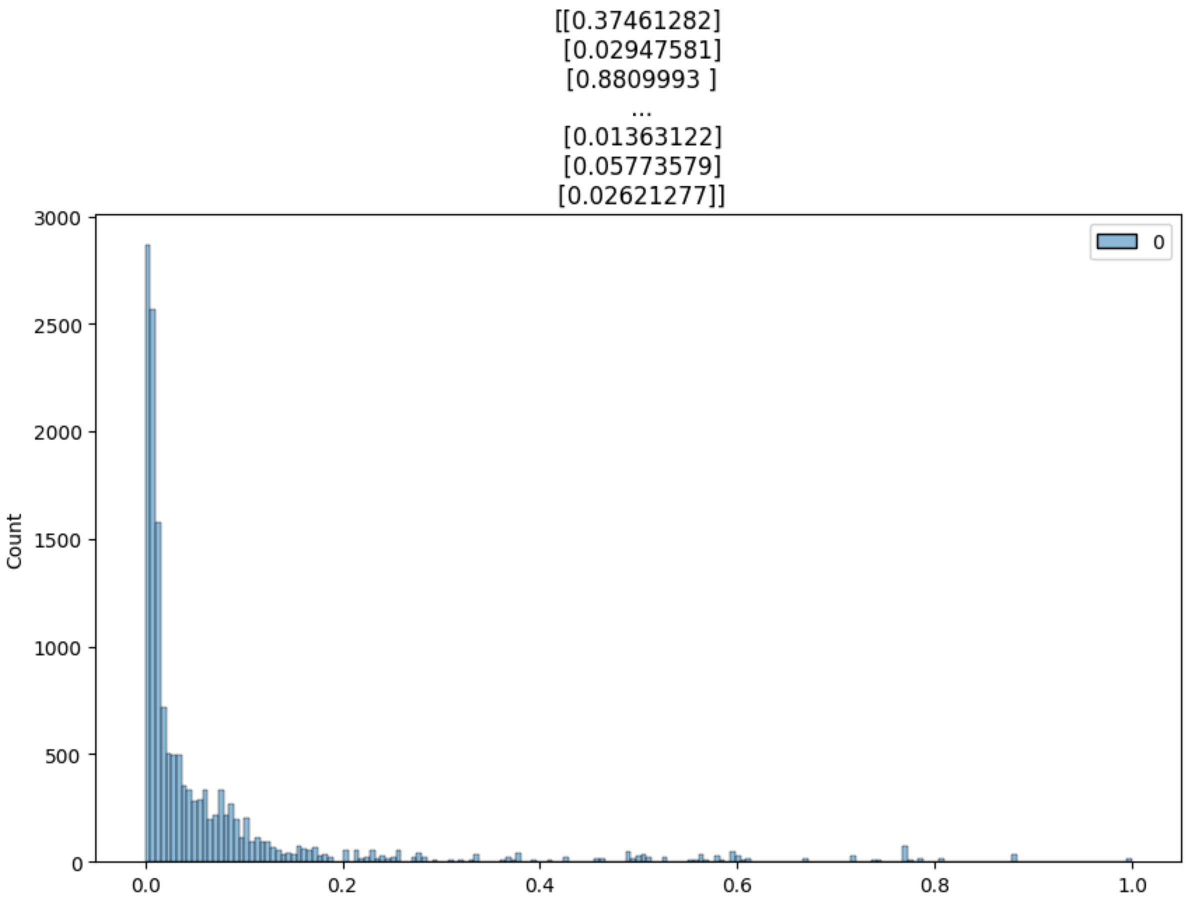

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"{scaled}")
plt.plot()
```

[]



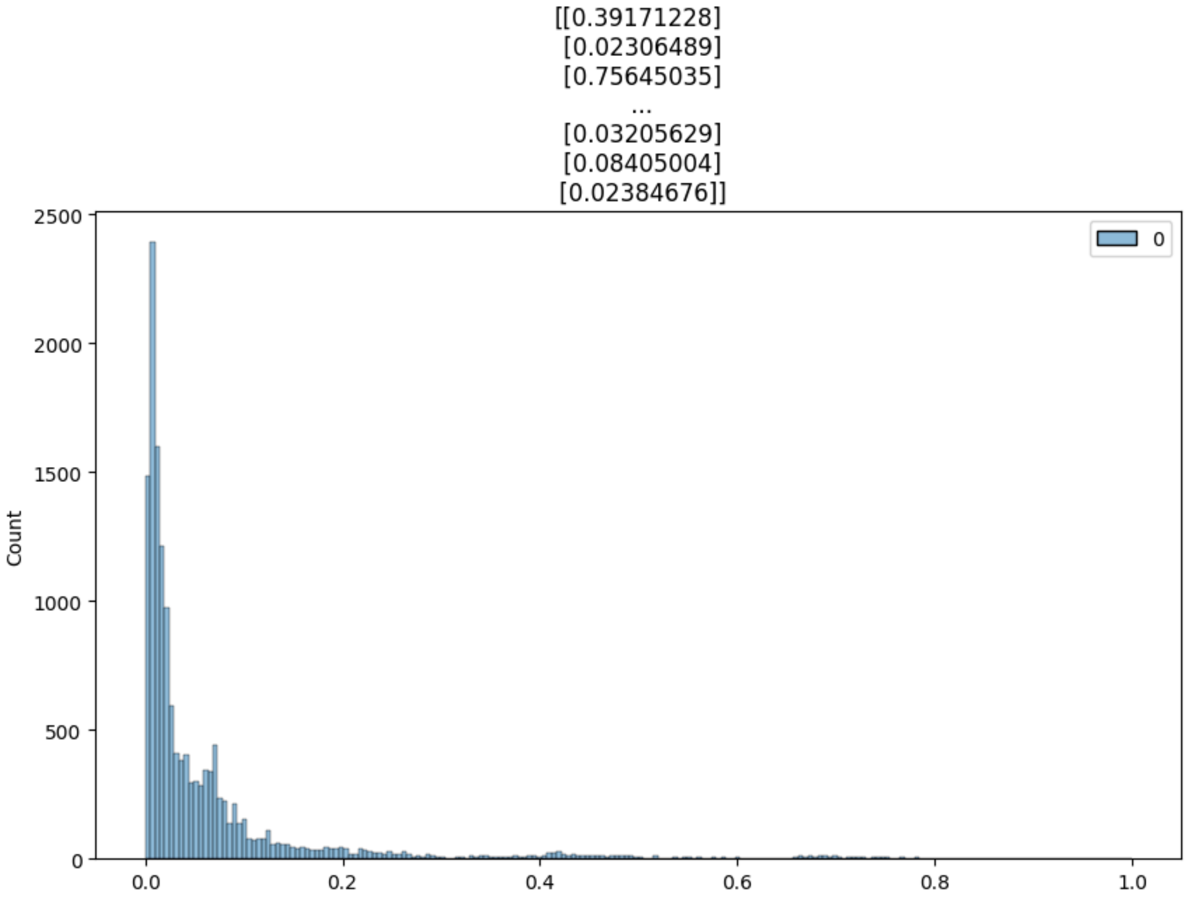

```
plt.figure(figsize = (10, 6))
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().reshape(-1, 1))
sns.histplot(scaled)
plt.title(f"{scaled}")
plt.plot()
```

[]




```
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"{scaled}")
plt.plot()
```

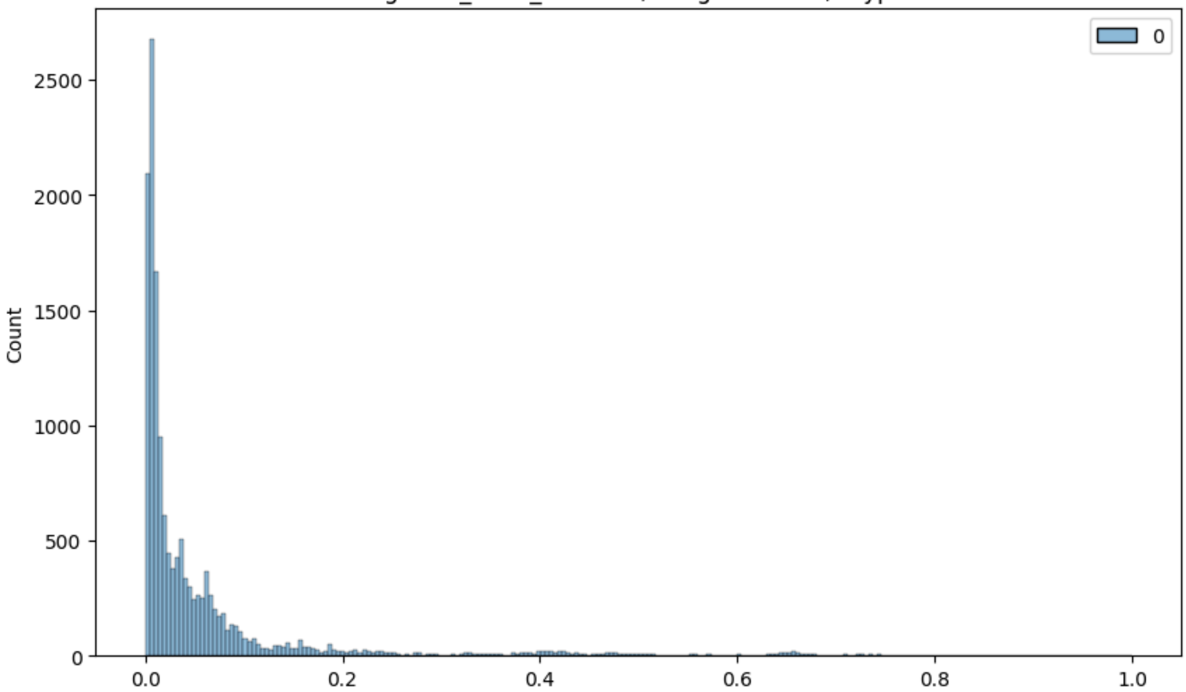
[]




```
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"{df2['segment_osrm_distance']}")
plt.plot()
```

[]

```
0        1320.4733
1          84.1894
2        2545.2678
3          19.8766
4         146.7919
           ...
14812      64.8551
14813      16.0883
14814     104.8866
14815     223.5324
14816      80.5787
Name: segment_osrm_distance, Length: 14817, dtype: float64
```

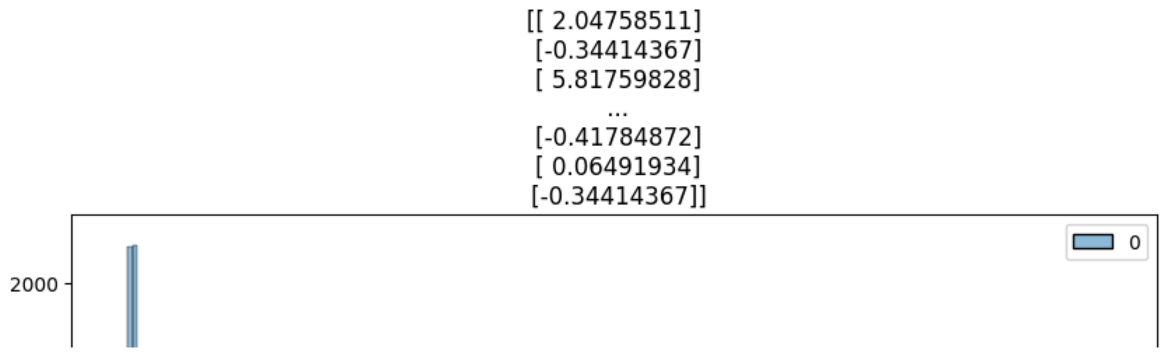


Standard Scaler

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

```
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"{scaled}")
plt.plot()
```

[]




```
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"{scaled}")
plt.plot()
```