Feature Engineering Case Study

In [ ]:
```
# Libraries
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [94]:
```
from scipy.stats import f_oneway,ttest_ind,shapiro,kruskal,chi2_contingency,levene
from statsmodels.graphics.gofplots import qqplot
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

1. Data Cleaning

In [2]:
```
df = pd.read_csv('delhivery_data.csv')
df.head()
```

Out[2]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_cer
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A

5 rows × 24 columns

In [3]:
```
df.shape
```

Out[3]:
```
(144867, 24)
```

In [4]:
```
df.dtypes
```

Out[4]:
```
data                                object
trip_creation_time                  object
route_schedule_uuid                 object
route_type                          object
trip_uuid                           object
source_center                       object
source_name                         object
destination_center                  object
destination_name                    object
od_start_time                       object
od_end_time                         object
start_scan_to_end_scan             float64
is_cutoff                             bool
cutoff_factor                        int64
cutoff_timestamp                    object
actual_distance_to_destination     float64
actual_time                        float64
osrm_time                          float64
osrm_distance                      float64
factor                             float64
segment_actual_time                float64
segment_osrm_time                  float64
segment_osrm_distance              float64
segment_factor                     float64
dtype: object
```

In [5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000
mean	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	2.120107
std	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	1.715421
min	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	0.144000
25%	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	1.604264
50%	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	1.857143
75%	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	2.213483
max	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	77.387097

In [7]:
```
df.describe(include=object)
```

Out[7]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destinati
count	144867	144867	144867	144867	144867	144867	144574	
unique	2	14817	1504	2	14817	1508	1498	
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND(
freq	104858	101	1812	99660	101	23347	23347	

1.1 Handling Null Values

In [8]:
```
round((df.isnull().sum()/df.shape[0])*100,2)
```

Out[8]:
```
data                                0.00
trip_creation_time                  0.00
route_schedule_uuid                 0.00
route_type                          0.00
trip_uuid                           0.00
source_center                       0.00
source_name                         0.20
destination_center                  0.00
destination_name                    0.18
od_start_time                       0.00
od_end_time                         0.00
start_scan_to_end_scan              0.00
is_cutoff                           0.00
cutoff_factor                       0.00
cutoff_timestamp                    0.00
actual_distance_to_destination      0.00
actual_time                         0.00
osrm_time                           0.00
osrm_distance                       0.00
factor                              0.00
segment_actual_time                 0.00
segment_osrm_time                   0.00
segment_osrm_distance               0.00
segment_factor                      0.00
dtype: float64
```

Observation 💡:

- There are some null values in source_name,destination_names approx less than 4% of total rows, so dropping them

In [9]:
```
df.dropna(inplace=True)
```

In [10]:
```
round((df.isnull().sum()/df.shape[0])*100,2)
```

Out[10]:
```
data                              0.0
trip_creation_time                0.0
route_schedule_uuid               0.0
route_type                        0.0
trip_uuid                         0.0
source_center                     0.0
source_name                       0.0
destination_center                0.0
destination_name                  0.0
od_start_time                     0.0
od_end_time                       0.0
start_scan_to_end_scan            0.0
is_cutoff                         0.0
cutoff_factor                     0.0
cutoff_timestamp                  0.0
actual_distance_to_destination    0.0
actual_time                       0.0
osrm_time                         0.0
osrm_distance                     0.0
factor                            0.0
segment_actual_time               0.0
segment_osrm_time                 0.0
segment_osrm_distance             0.0
segment_factor                    0.0
dtype: float64
```

In [11]:
```
pd.set_option('display.max_columns',24)
df[df.duplicated()]
```

Out[11]:

data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name

Observation 💡 :

- No Duplicates Values found

1.2 TypeCasting to Datetime

In [12]:
```
pd.set_option('display.max_columns',24)
df.head(1)
```

Out[12]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_cer
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A

In [13]:
```
time_cols = ['df_creation_time','od_start_time','od_end_time']
```

In [14]:
```
for i in time_cols:
    print(df[i].dtype)
```

```
object
object
object
```

In [15]:
```
for i in time_cols:
    df[i] = pd.to_datetime(df[i])
```

In [16]:
```
for i in time_cols:
    print(df[i].dtype)
```

```
datetime64[ns]
datetime64[ns]
datetime64[ns]
```

1.3 Columns Segmentation

In [17]:
```
categorical_cols,numerical_cols,date_time_cols = [],[],[]
for index,type in enumerate(df.dtypes):
    if type == 'object':
        categorical_cols.append(df.columns[index])
    elif type in ['int64','float64']:
        numerical_cols.append(df.columns[index])
```

```
    else:
        date_time_cols.append(df.columns[index])
```

In [18]:
```
date_time_cols
```

Out[18]:
```
['trip_creation_time', 'od_start_time', 'od_end_time', 'is_cutoff']
```

In [19]:
```
for i in categorical_cols:
    print(i)
    print(df[i].describe())
    print(40*'=')
```

```
data
count       144316
unique           2
top       training
freq        104632
Name: data, dtype: object
========================================
route_schedule_uuid
count                                                144316
unique                                                 1497
top       thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...
freq                                                   1812
Name: route_schedule_uuid, dtype: object
========================================
route_type
count     144316
unique         2
top          FTL
freq       99132
Name: route_type, dtype: object
========================================
trip_uuid
count                      144316
unique                      14787
top       trip-153837029526866991
freq                          101
Name: trip_uuid, dtype: object
========================================
source_center
count           144316
unique            1496
top       IND000000ACB
freq             23267
Name: source_center, dtype: object
========================================
source_name
count                             144316
unique                              1496
top       Gurgaon_Bilaspur_HB (Haryana)
freq                                23267
Name: source_name, dtype: object
```

```
========================================
destination_center
count           144316
unique            1466
top       IND000000ACB
freq             15192
Name: destination_center, dtype: object
========================================
destination_name
count                           144316
unique                            1466
top       Gurgaon_Bilaspur_HB (Haryana)
freq                              15192
Name: destination_name, dtype: object
========================================
cutoff_timestamp
count                  144316
unique                  92894
top       2018-09-24 05:19:20
freq                       39
Name: cutoff_timestamp, dtype: object
========================================
```

Observation 💡 :

- FTL is the Frequent Route
- Top Source and Destinations as 'Gurgaon_Bilaspur_HB (Haryana)'

2.Feature Engineering

In [20]:
```
# Creation of new field od time diff in hours
df['od_total_time'] = (df['od_end_time'] - df['od_start_time']).dt.total_seconds()/60
```

In [21]:
```
# Splitting and extract feature of source_name and destination_name
def location_to_state(loc):
    return loc.split('(')[1][:-1]
```

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

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

    if len_ >= 3:
        return loc.split('_')[1]
    # small cities have same city and place name
    if len_ == 2:
        return loc.split('_')[0]
```

```
    return loc.split(' ')[0]
def location_to_code(loc):
    # we will remove state
    loc = loc.split('(')[0]
    if len(loc.split('_')) >= 3:
        return loc.split('_')[-1]

    return 'none'
```

In [22]:
```
df['destination_state'] = df['destination_name'].apply(lambda x:
location_to_state(x))
df['destination_city'] = df['destination_name'].apply(lambda x:
location_to_city(x))
df['destination_place'] = df['destination_name'].apply(lambda x:
location_to_place(x))
df['destination_code'] = df['destination_name'].apply(lambda x:
location_to_code(x))
```

In [23]:
```
df['source_state'] = df['source_name'].apply(lambda x:
location_to_state(x))
df['source_city'] = df['source_name'].apply(lambda x:
location_to_city(x))
df['source_place'] = df['source_name'].apply(lambda x:
location_to_place(x))
df['source_code'] = df['source_name'].apply(lambda x:
location_to_code(x))
```

In [24]:
```
# Splitting df_Creation_time Acc to Year, Month,Day and Hour
df['creation_year'] = df['df_creation_time'].dt.year
df['creation_month']= df['df_creation_time'].dt.month
df['creation_day'] = df['df_creation_time'].dt.day
df['creation_hour'] = df['df_creation_time'].dt.hour
```

In [25]:
```
df.head()
```

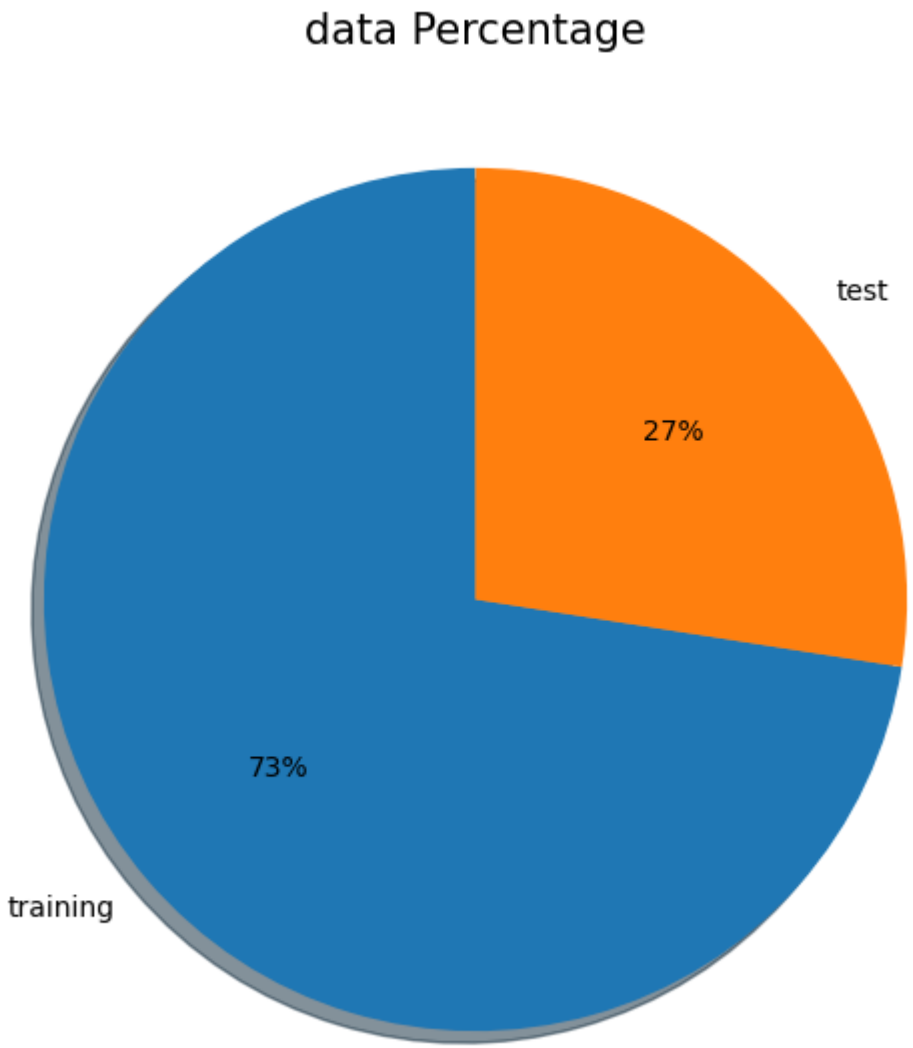
Out[25]:

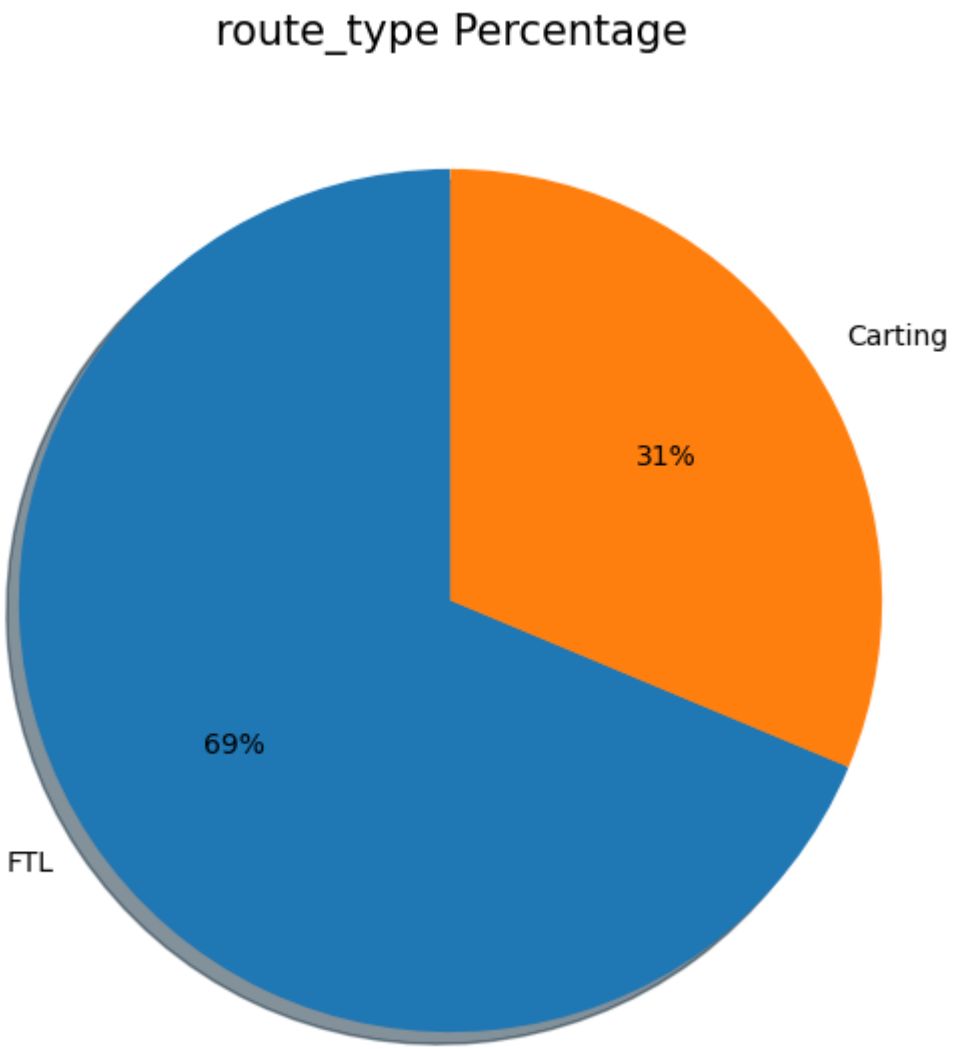
	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_cer
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620A

5 rows × 37 columns

3. In Depth Analysis

In [26]:
```
temp = ['data','route_type']
for col in temp:
    fig = plt.figure(figsize=(10,7))
    plt.pie(df[col].value_counts().values, labels =
    list(df[col].value_counts().index), autopct='%.0f%%', shadow= True,
    startangle = 90)
    plt.title(f"{col} Percentage", fontsize=15)
    plt.show()
```

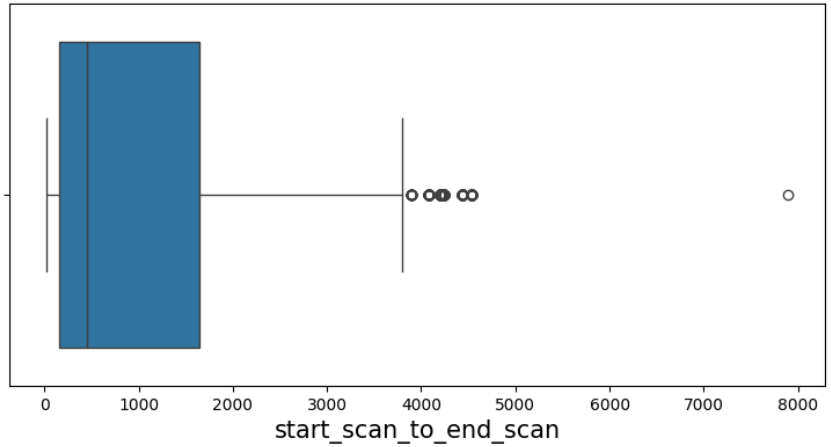

data Percentage
test
27%
73%
training

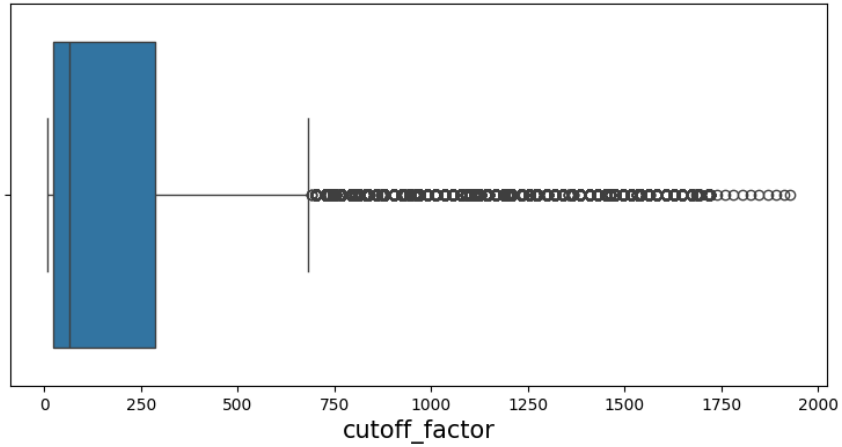


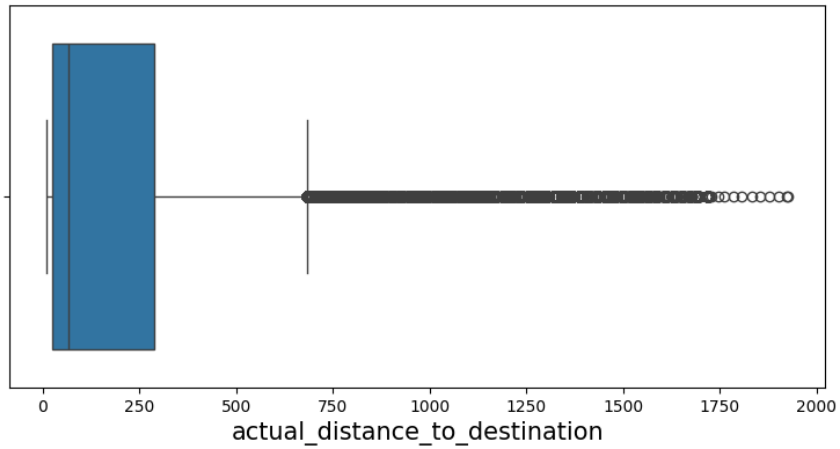
3.1 Outlier Detection

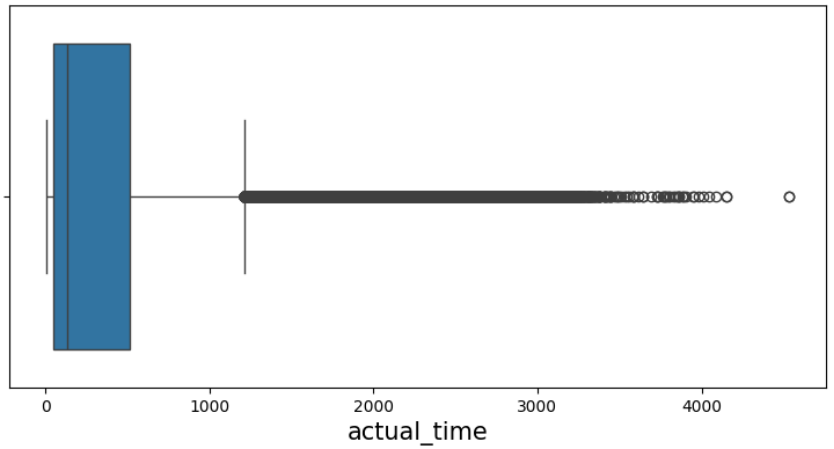
In [27]:
```
plt.figure(figsize=(20,30))
for i,col in enumerate(numerical_cols):
```

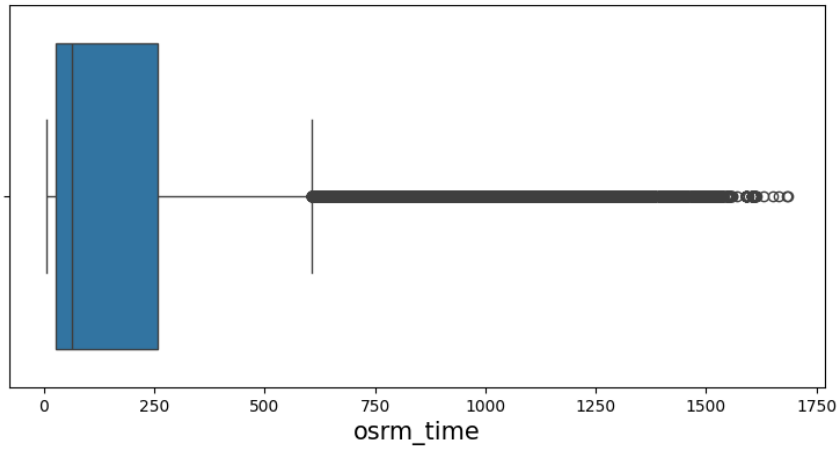
```
    plt.subplot(int(len(numerical_cols)/2)+1, 2, i+1)
    sns.boxplot(x= df[col])
    plt.xlabel(col, fontsize=15)
plt.show()
```

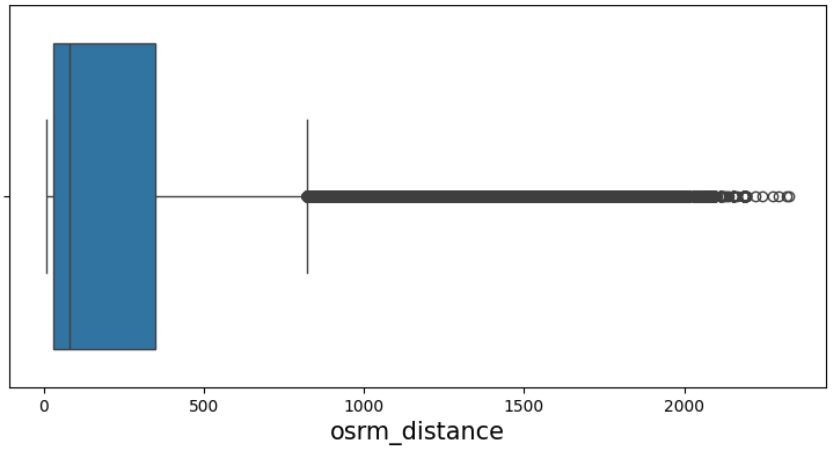

0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan

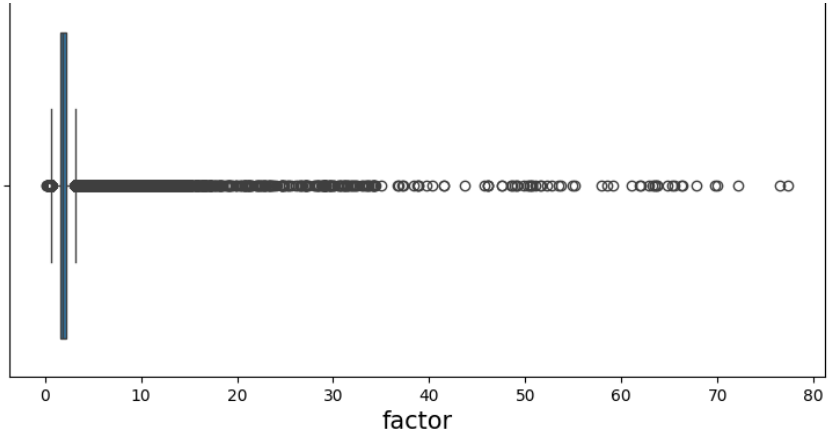

0
250
500
750
1000
1250
1500
1750
2000
cutoff_factor

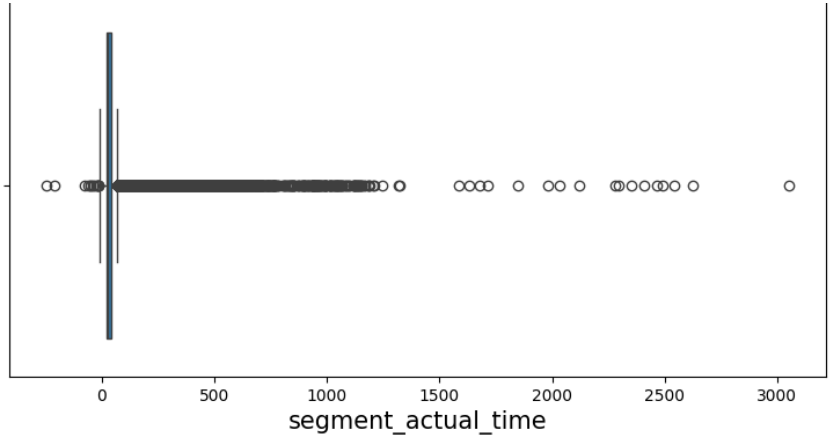

0
250
500
750
1000
1250
1500
1750
2000
actual_distance_to_destination

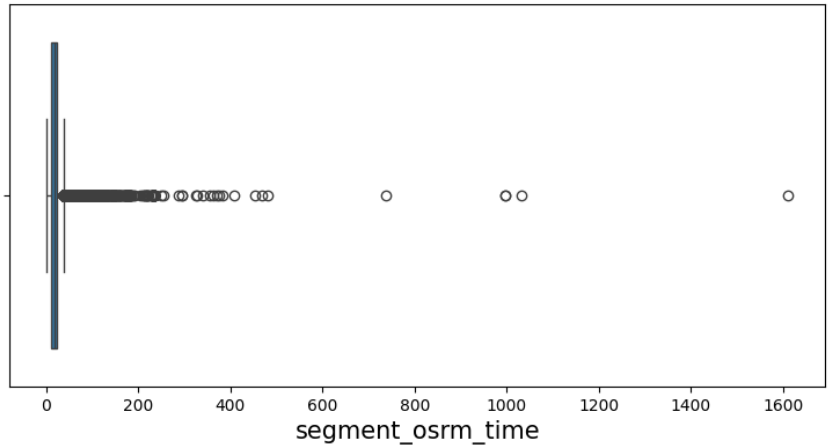

0
1000
2000
3000
4000
actual_time

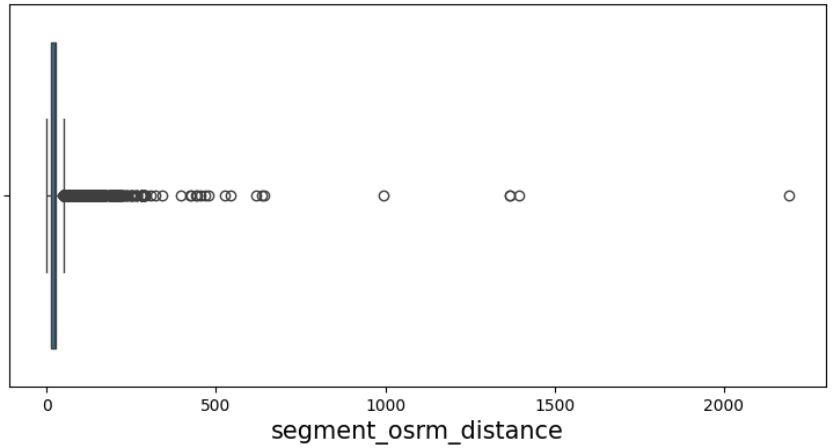

0
250
500
750
1000
1250
1500
1750
osrm_time

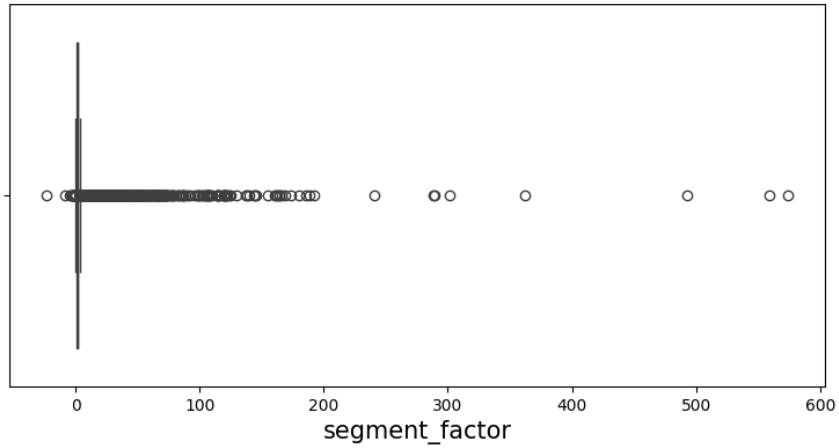

0
500
1000
1500
2000
osrm_distance


0
10
20
30
40
50
60
70
80
factor


0
500
1000
1500
2000
2500
3000
segment_actual_time


0
200
400
600
800
1000
1200
1400
1600
segment_osrm_time


0
500
1000
1500
2000
segment_osrm_distance


0
100
200
300
400
500
600
segment_factor

In [28]:

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

```
Column: start_scan_to_end_scan
Q1: 161.0
Q3: 1645.0
LB: -2065.0
UB: 3871.0
IQR: 1484.0
Outliers: 373
=========================
Column: cutoff_factor
Q1: 22.0
Q3: 286.0
LB: -374.0
UB: 682.0
IQR: 264.0
Outliers: 17246
=========================
Column: actual_distance_to_destination
Q1: 23.352026892994942
Q3: 286.9192937699907
LB: -371.99887342249866
UB: 682.2701940854843
IQR: 263.56726687699575
Outliers: 17818
=========================
Column: actual_time
Q1: 51.0
Q3: 516.0
LB: -646.5
UB: 1213.5
IQR: 465.0
Outliers: 16507
=========================
Column: osrm_time
Q1: 27.0
Q3: 259.0
LB: -321.0
UB: 607.0
IQR: 232.0
Outliers: 17406
=========================
Column: osrm_distance
```

```
Q1: 29.89625
Q3: 346.3054
LB: -444.71747500000004
UB: 820.9191250000001
IQR: 316.40915
Outliers: 17547
=========================
Column: factor
Q1: 1.6045454545454545
Q3: 2.2122803049883686
LB: 0.6929431788810834
UB: 3.1238825806527397
IQR: 0.6077348504429141
Outliers: 11473
=========================
Column: segment_actual_time
Q1: 20.0
Q3: 40.0
LB: -10.0
UB: 70.0
IQR: 20.0
Outliers: 9262
=========================
Column: segment_osrm_time
Q1: 11.0
Q3: 22.0
LB: -5.5
UB: 38.5
IQR: 11.0
Outliers: 6348
=========================
Column: segment_osrm_distance
Q1: 12.053975000000001
Q3: 27.813325000000003
LB: -11.585050000000003
UB: 51.45235000000001
IQR: 15.759350000000001
Outliers: 4295
=========================
Column: segment_factor
Q1: 1.3478260869565215
```

```
Q3: 2.25
LB: -0.005434782608696231
UB: 3.6032608695652177
IQR: 0.9021739130434785
Outliers: 13926
=========================
```

3.2 Encoding on Categorical Features

In [29]:
```
temp
```

Out[29]:
```
['data', 'route_type']
```

In [30]:
```
print("value counts before label encoding")
print(df['data'].value_counts())
print(df['route_type'].value_counts())
```

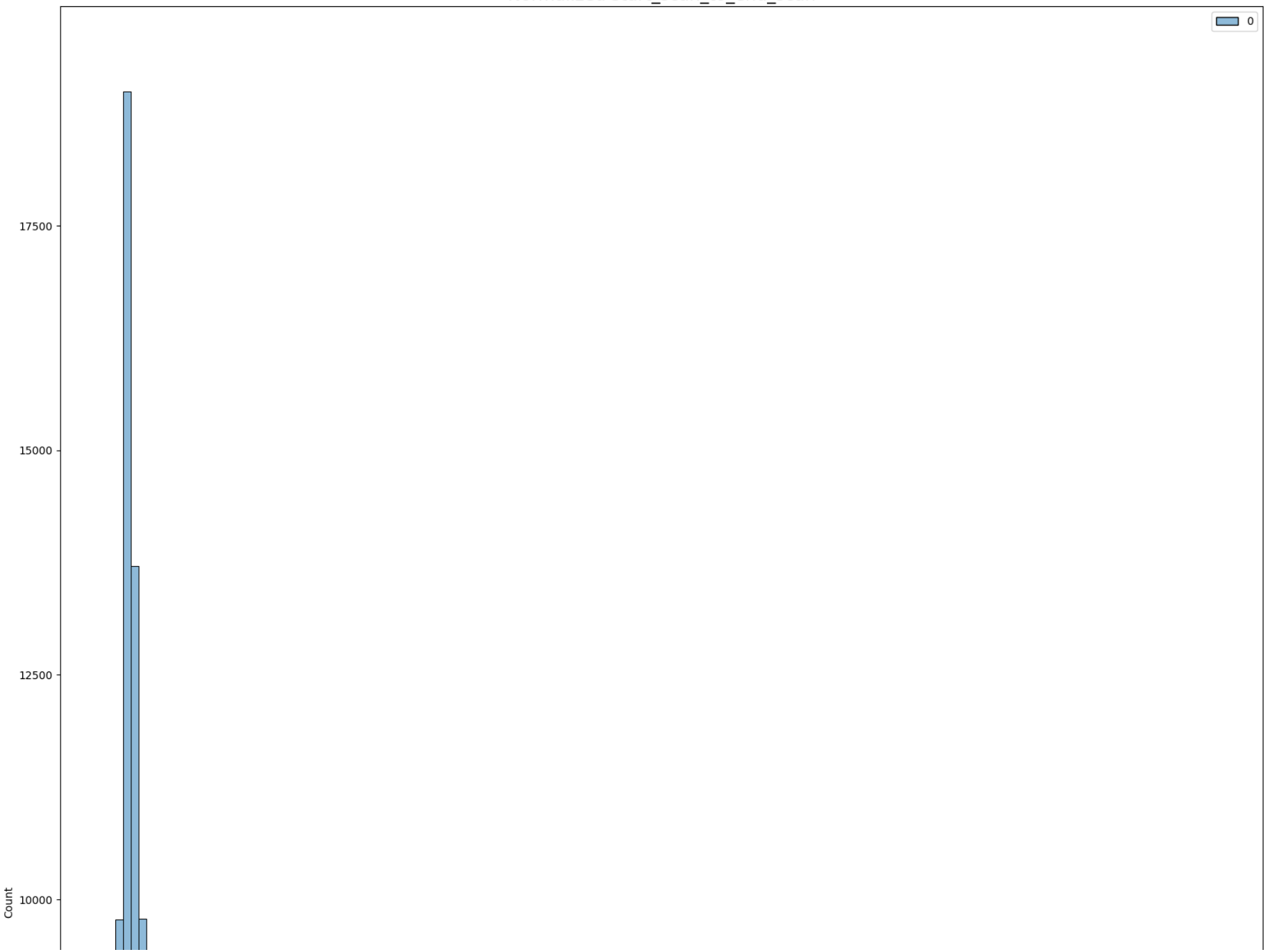
```
value counts before label encoding
data
training    104632
test         39684
Name: count, dtype: int64
route_type
FTL        99132
Carting    45184
Name: count, dtype: int64
```

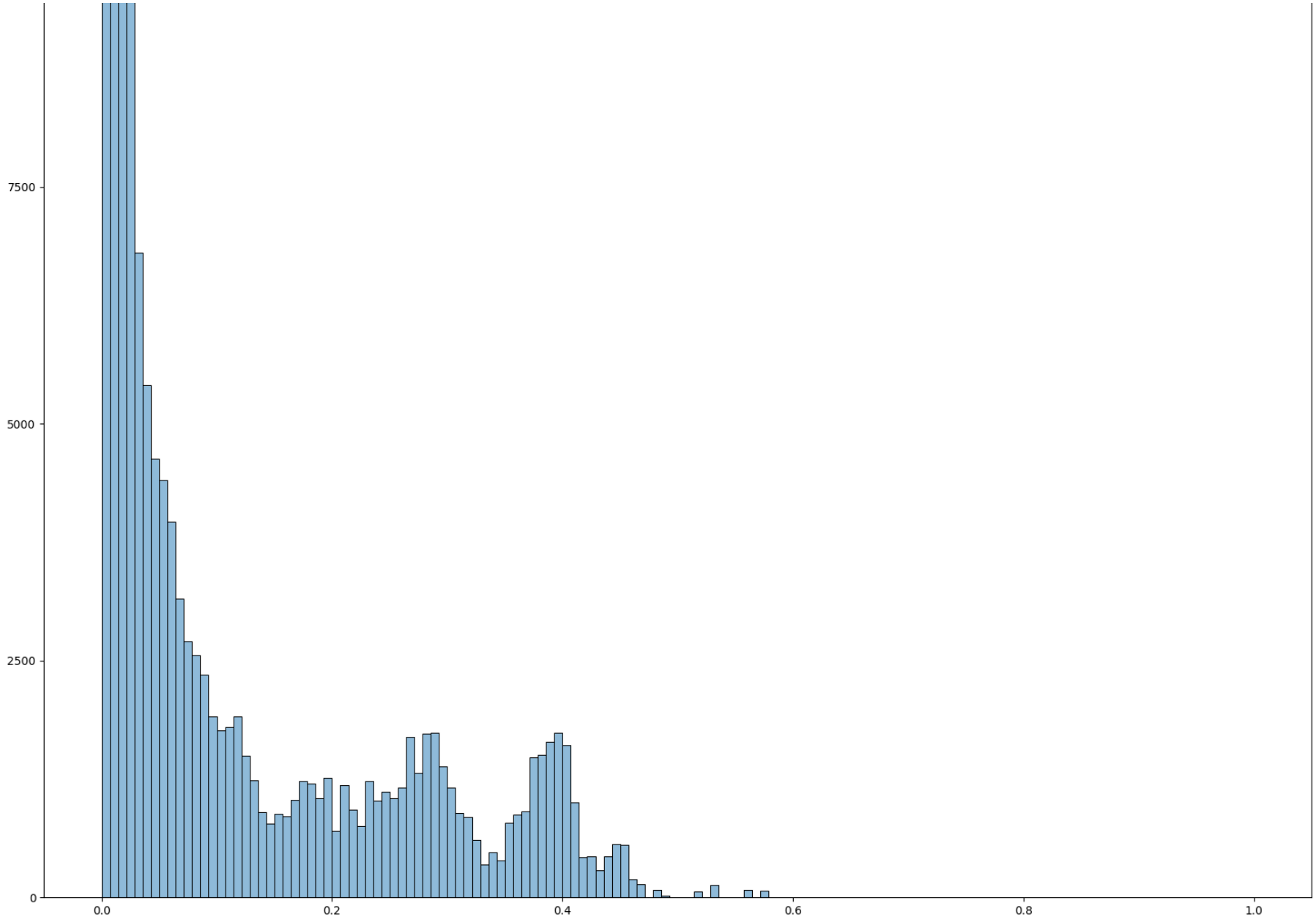
In [31]:
```
label_encoder = LabelEncoder()
df['data'] = label_encoder.fit_transform(df['data'])
df['route_type'] = label_encoder.fit_transform(df['route_type'])
print("value counts after label encoding")
print(df['data'].value_counts())
print(df['route_type'].value_counts())
```

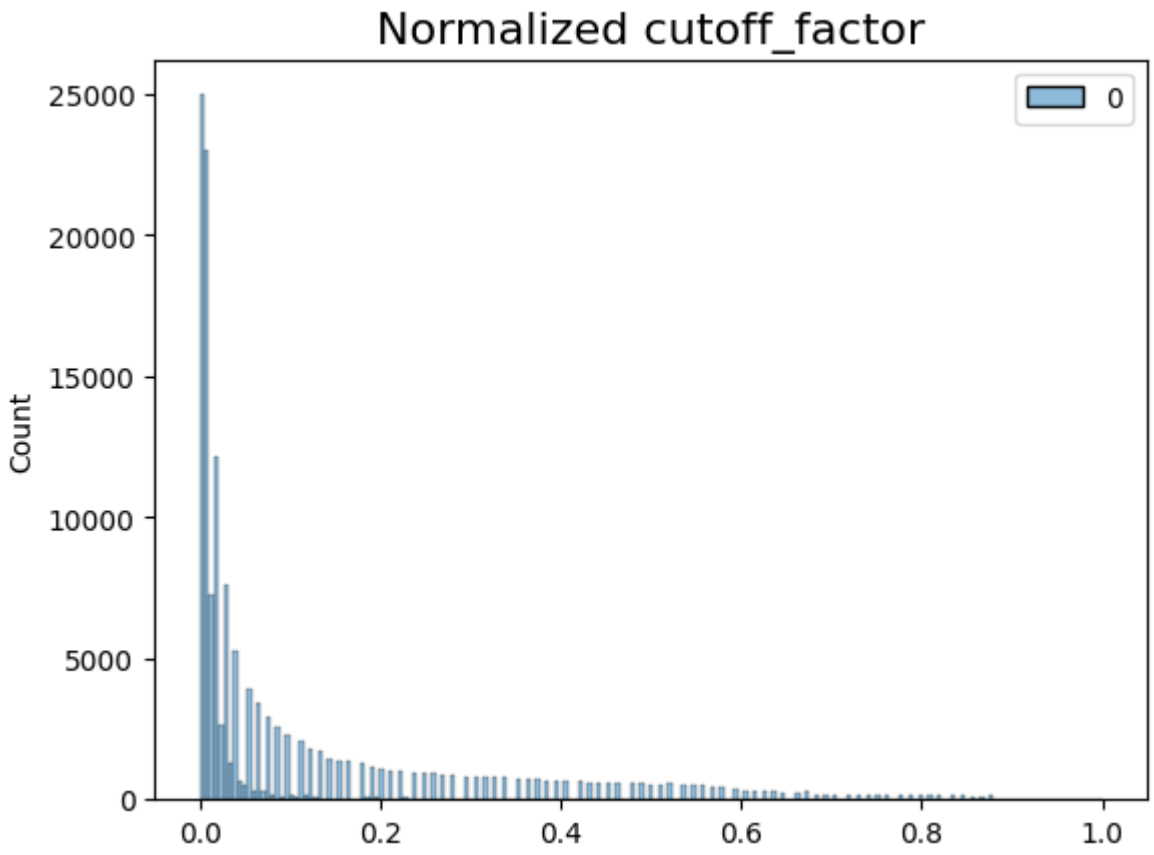
```
value counts after label encoding
data
1    104632
0     39684
Name: count, dtype: int64
route_type
1    99132
0    45184
Name: count, dtype: int64
```

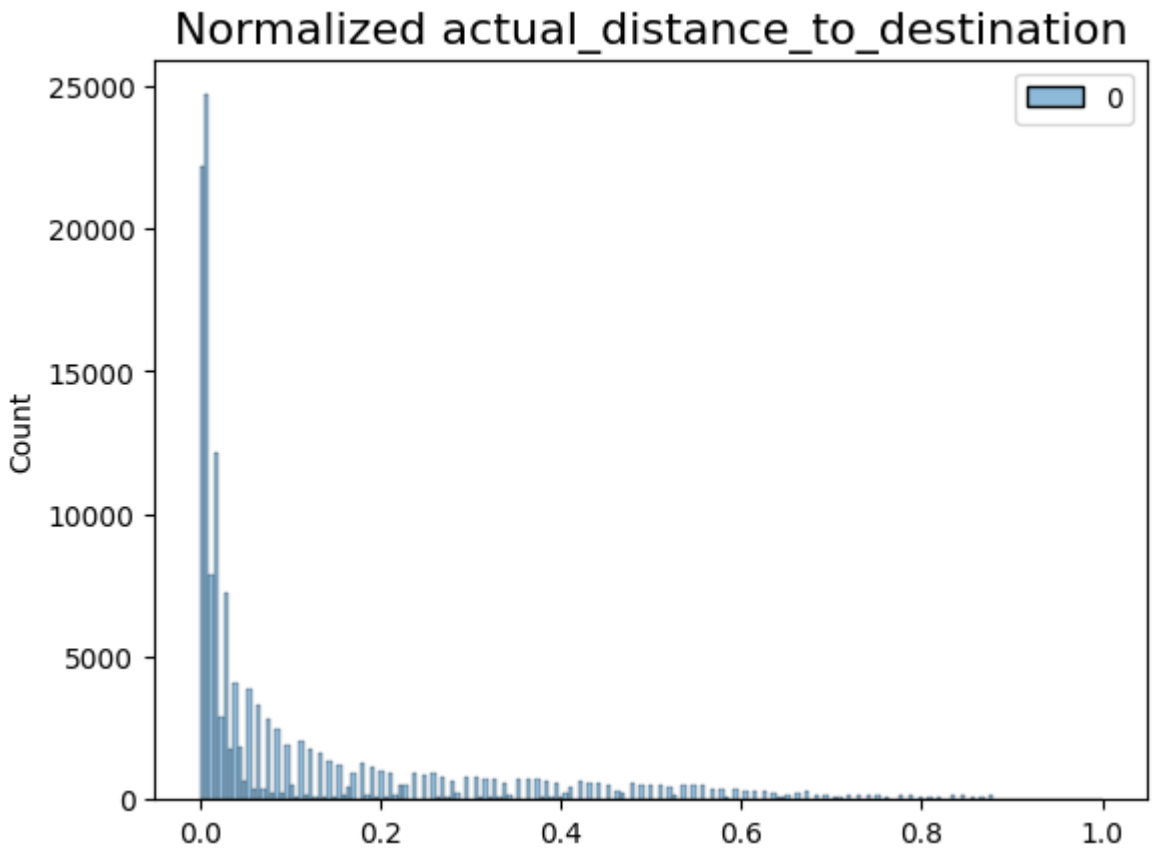
3.3 Normalization of Numerical Features using MinMaxScaler or StandardScaler

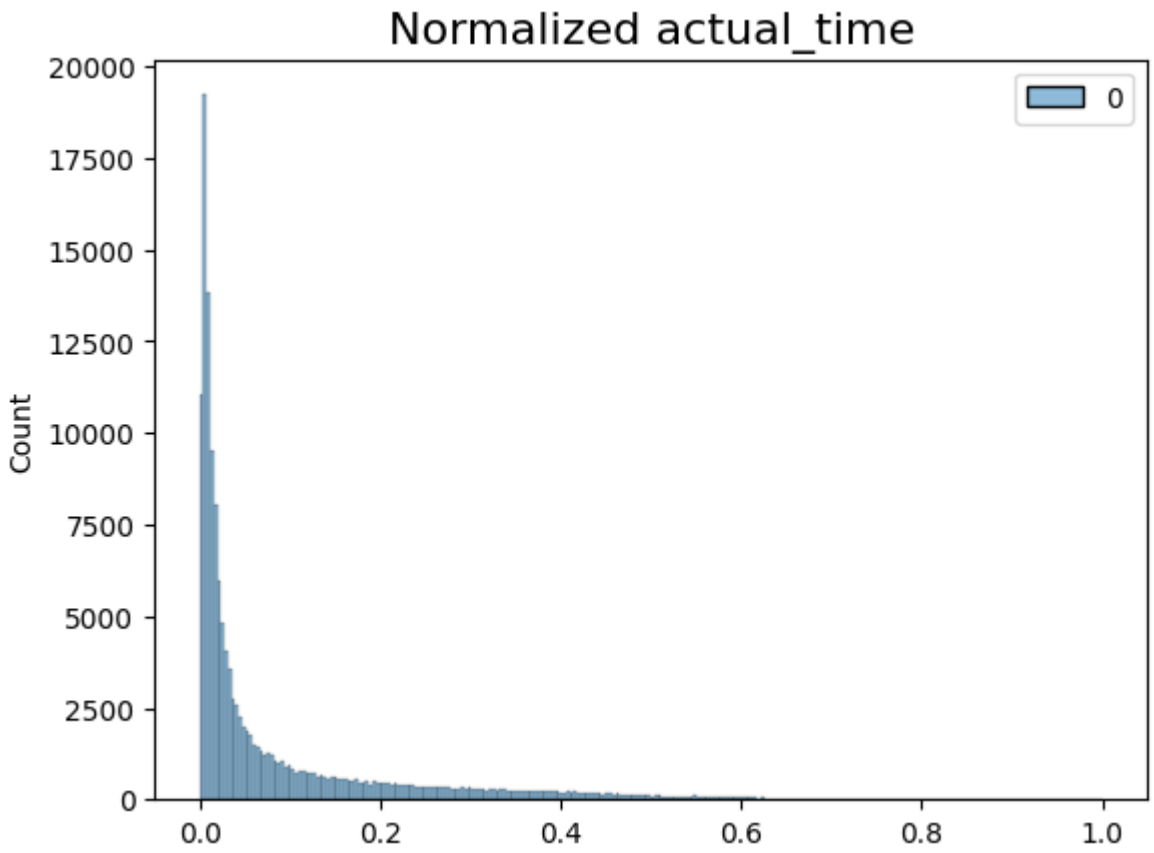
In [32]:
```
plt.figure(figsize=(20,30))
for col in numerical_cols:
    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(df[col].to_numpy().reshape(-1, 1))
    sns.histplot(scaled)
    plt.title(f"Normalized {col}", fontsize=16)
    plt.show()
```

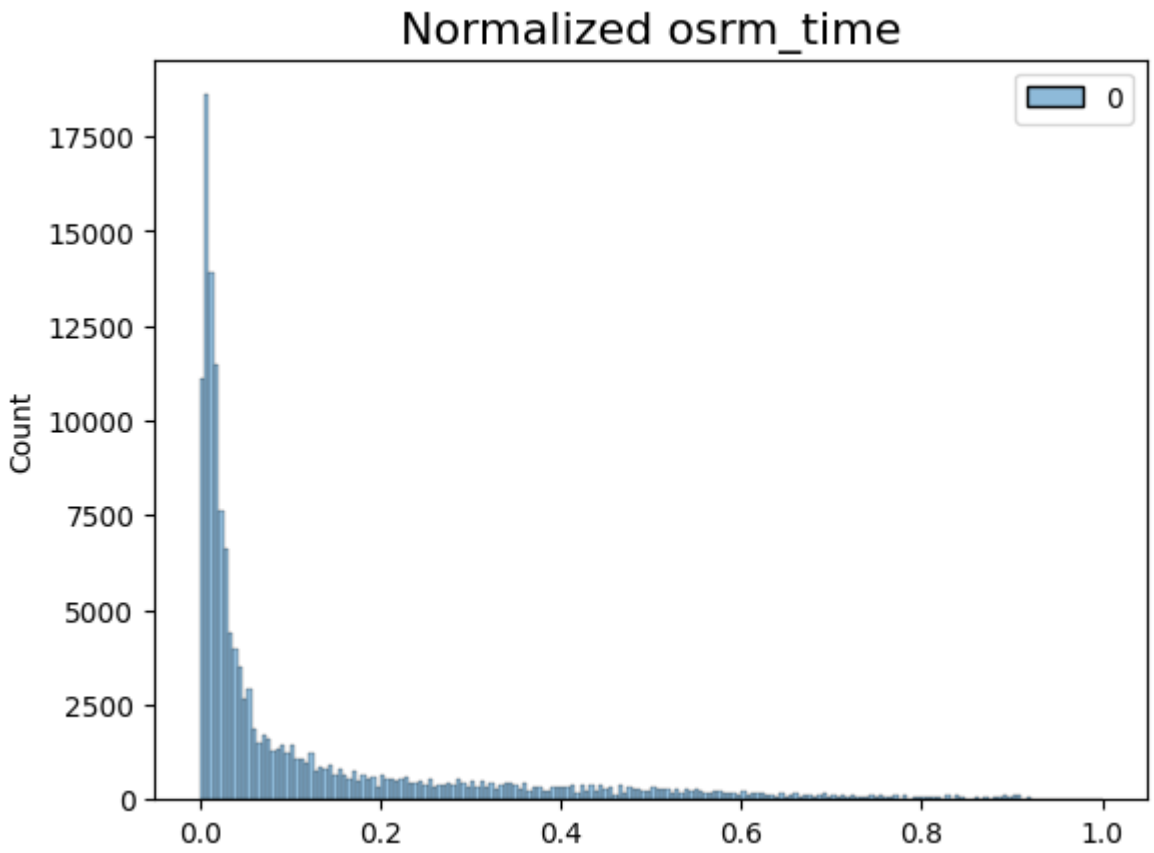

Normalized start_scan_to_end_scan
0
17500
15000
12500
10000
Count

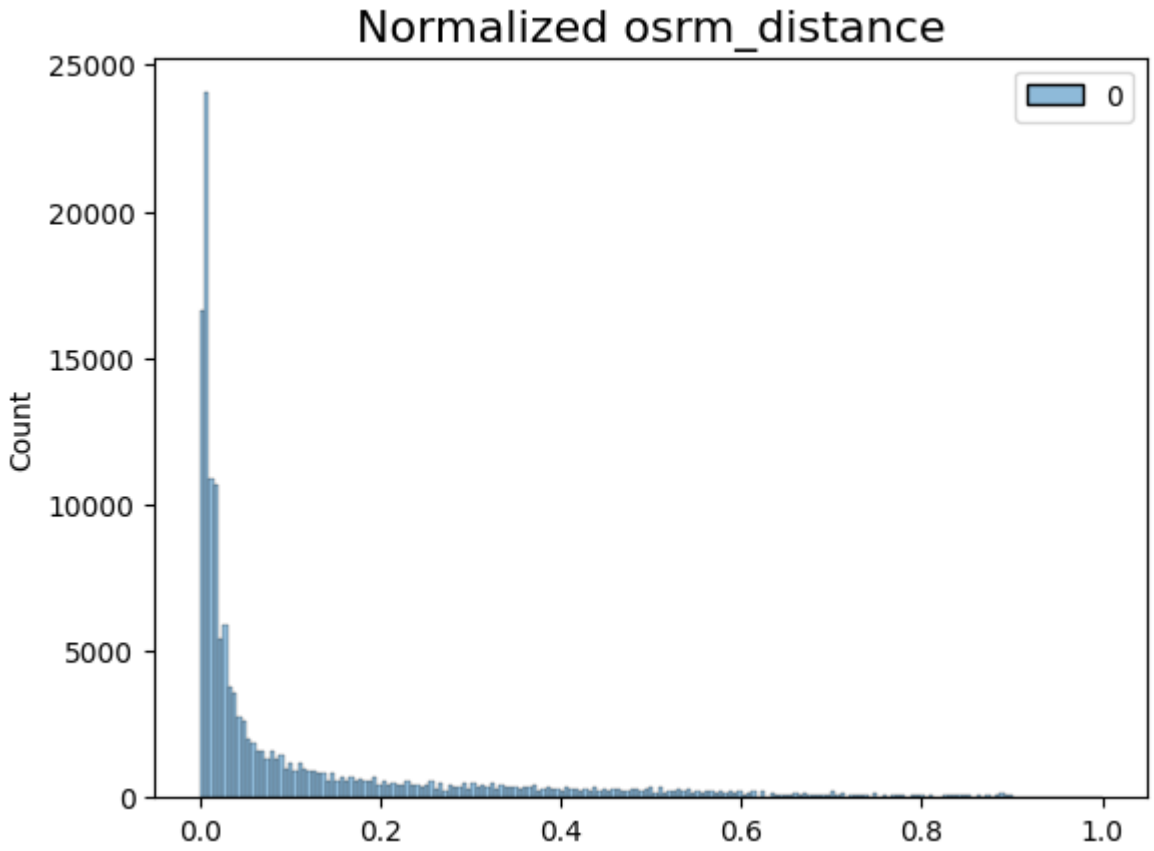

7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

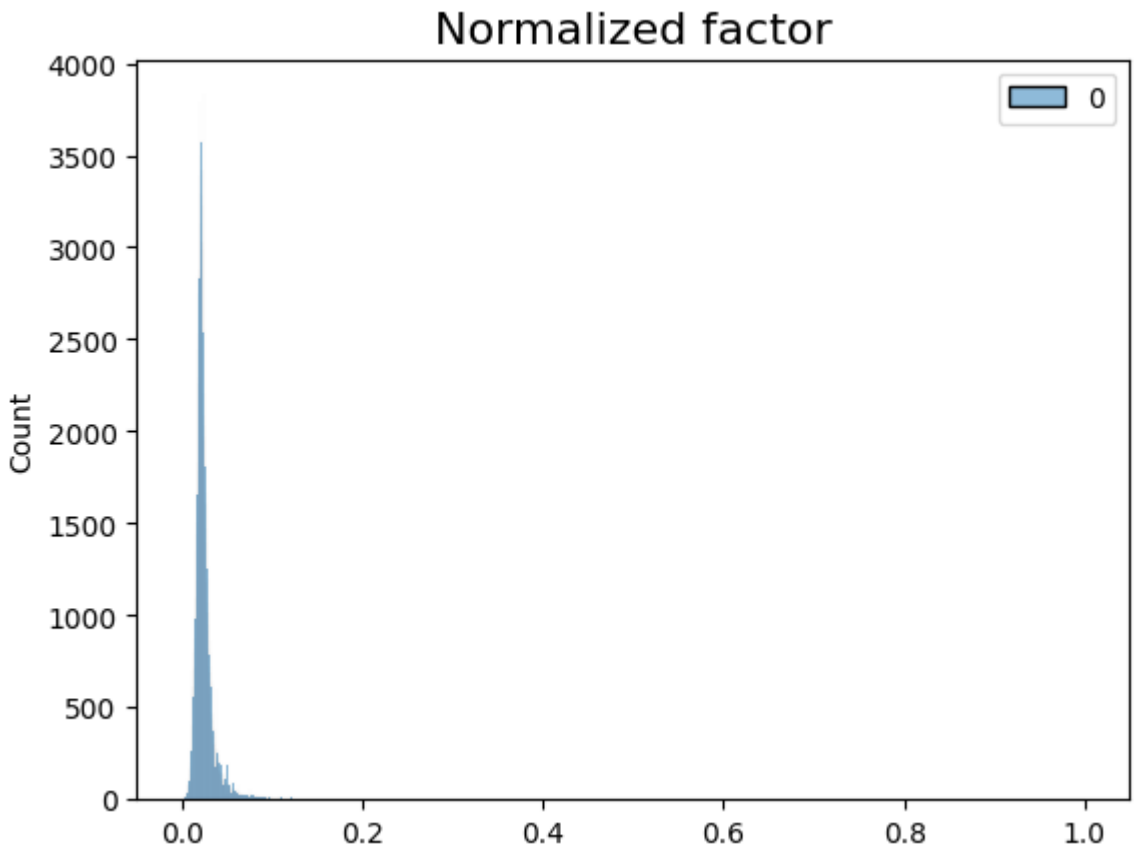

Normalized cutoff_factor
0
Count
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

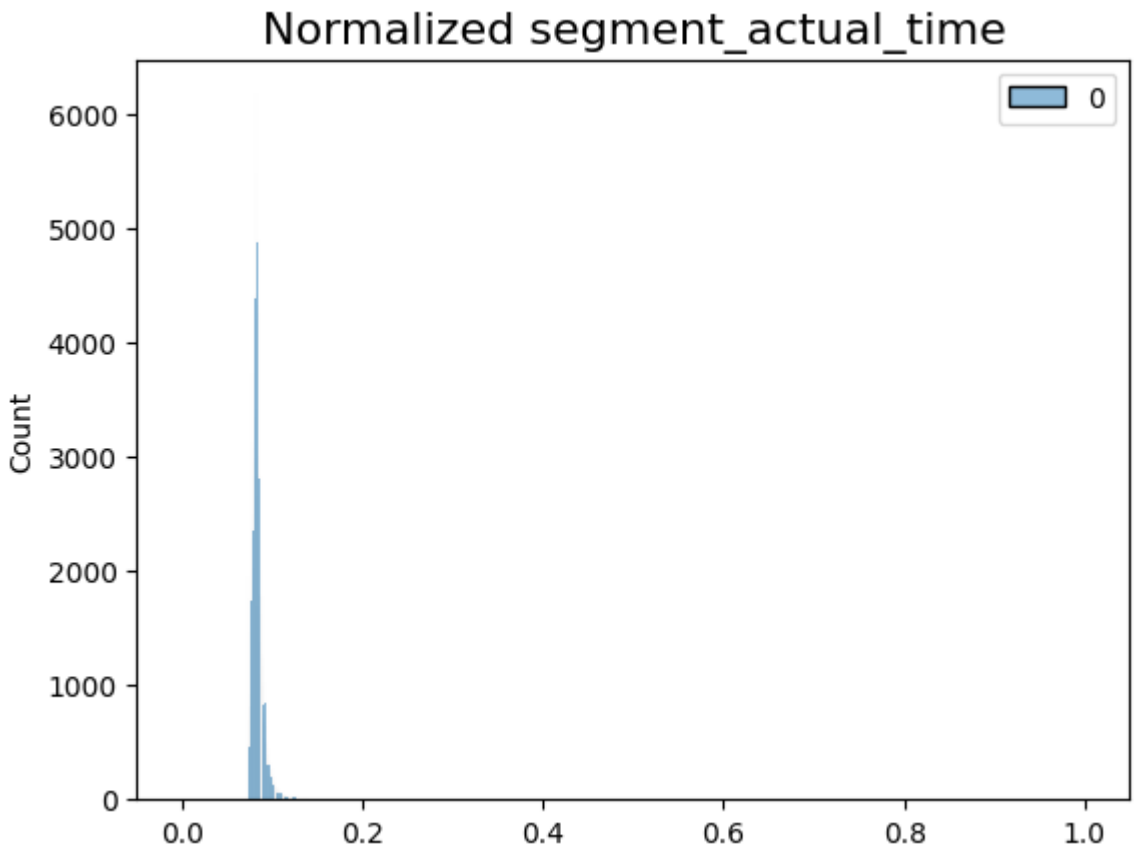

Normalized actual_distance_to_destination
0
Count
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

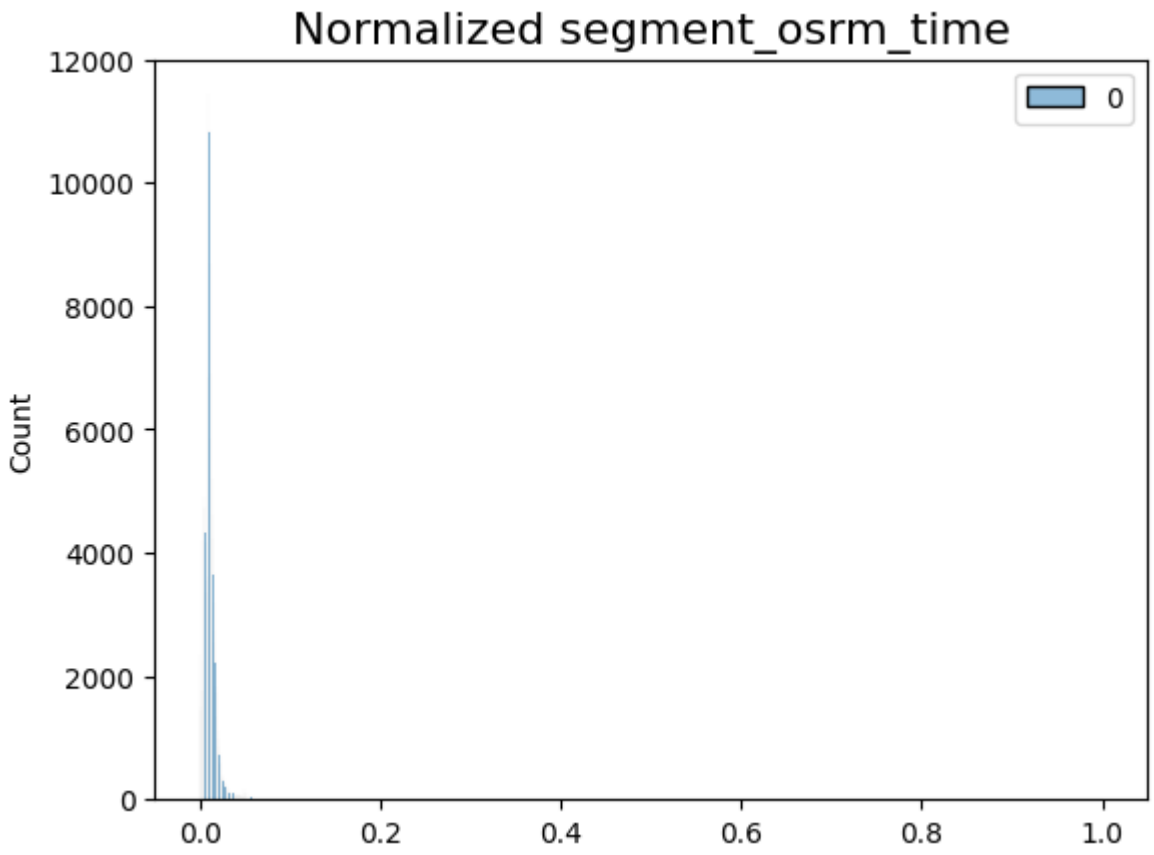

Normalized actual_time
Count
0
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

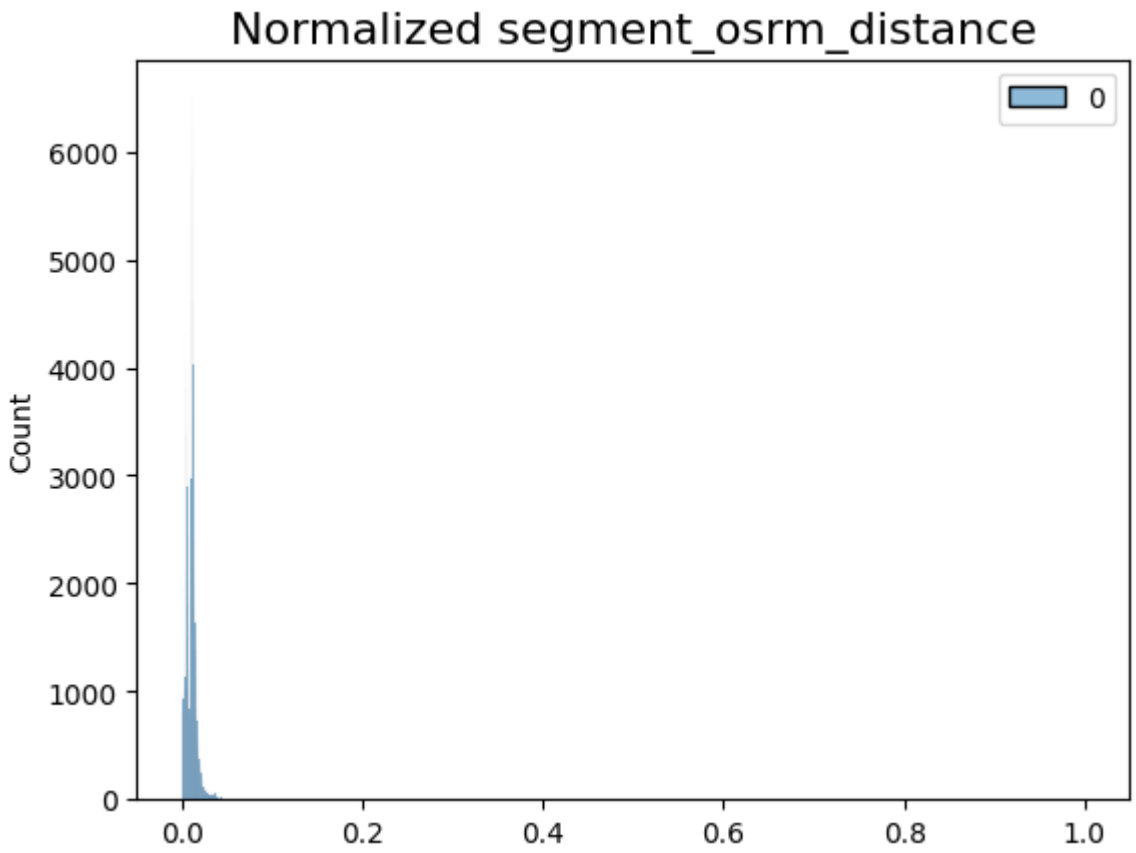

Normalized osrm_time
0
Count
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

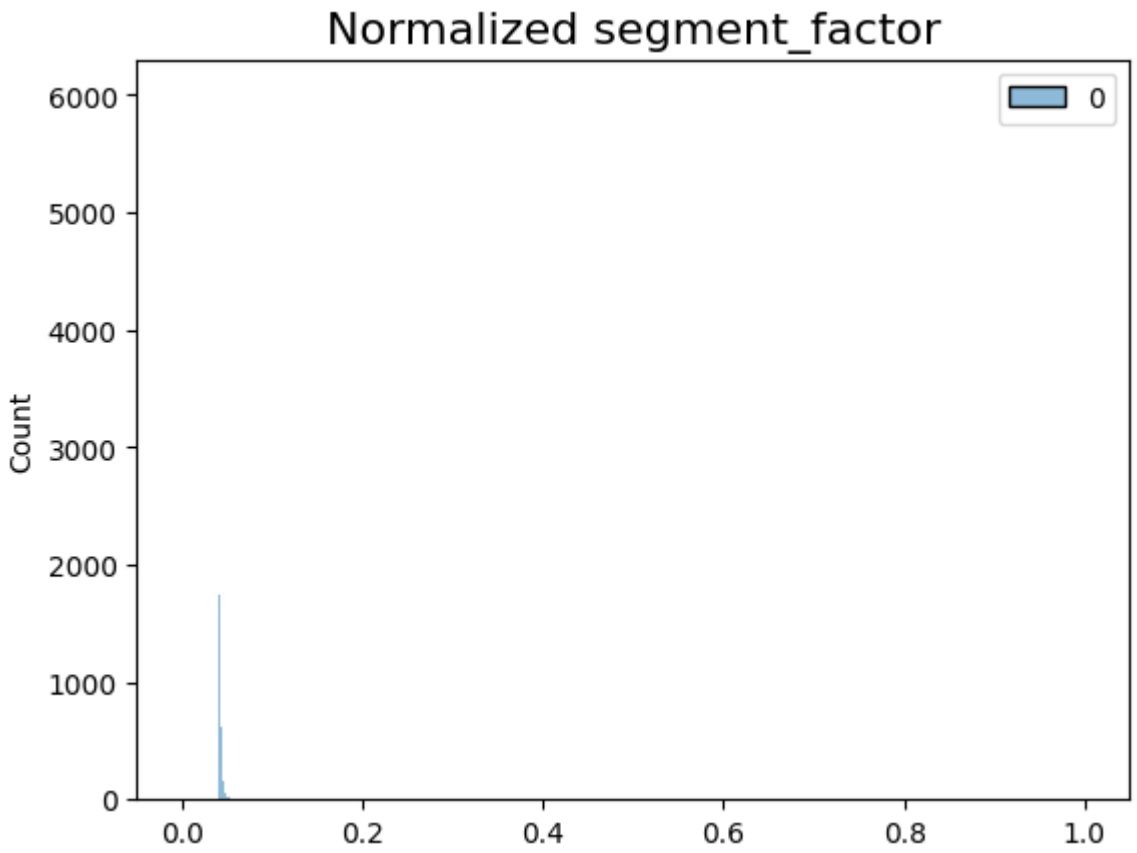

Normalized osrm_distance
Count
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
0


Normalized factor
Count
0
4000
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

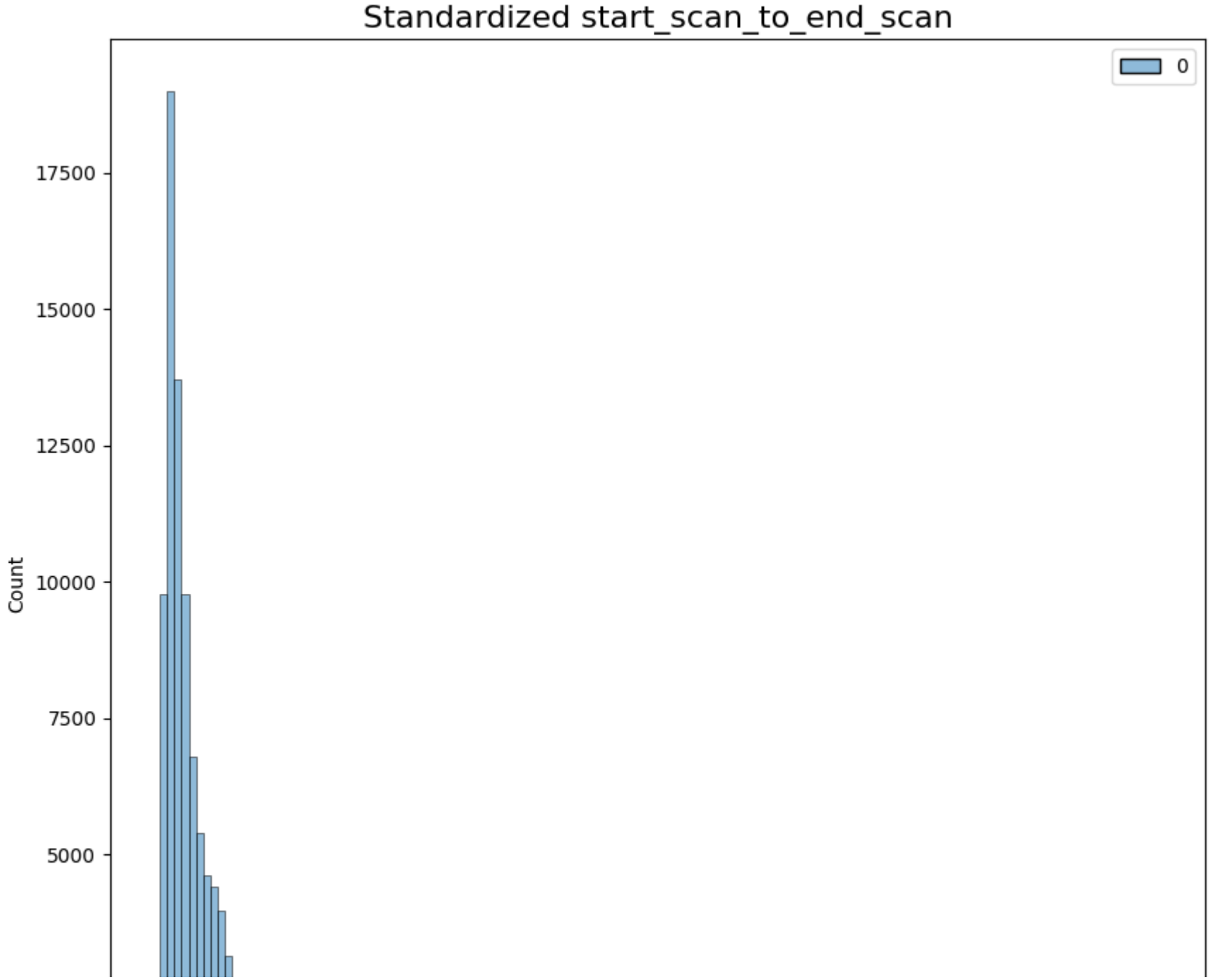

Normalized segment_actual_time
0
Count
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

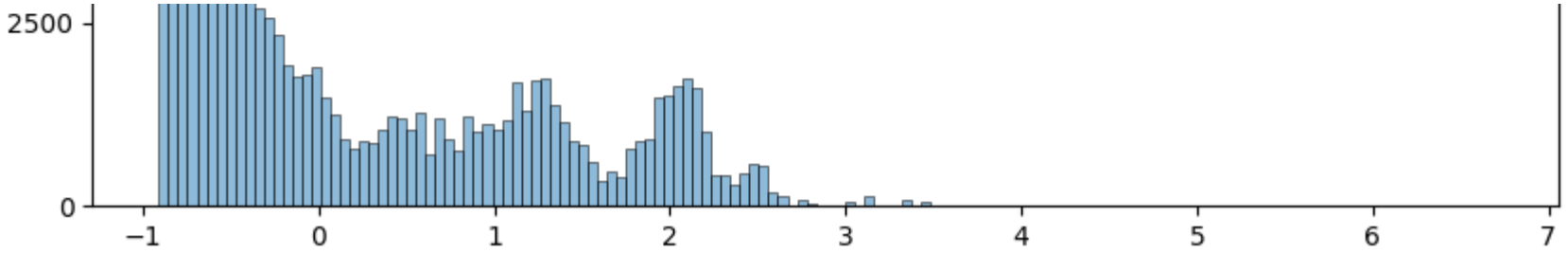

Normalized segment_osrm_time
12000
10000
8000
6000
4000
2000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
0

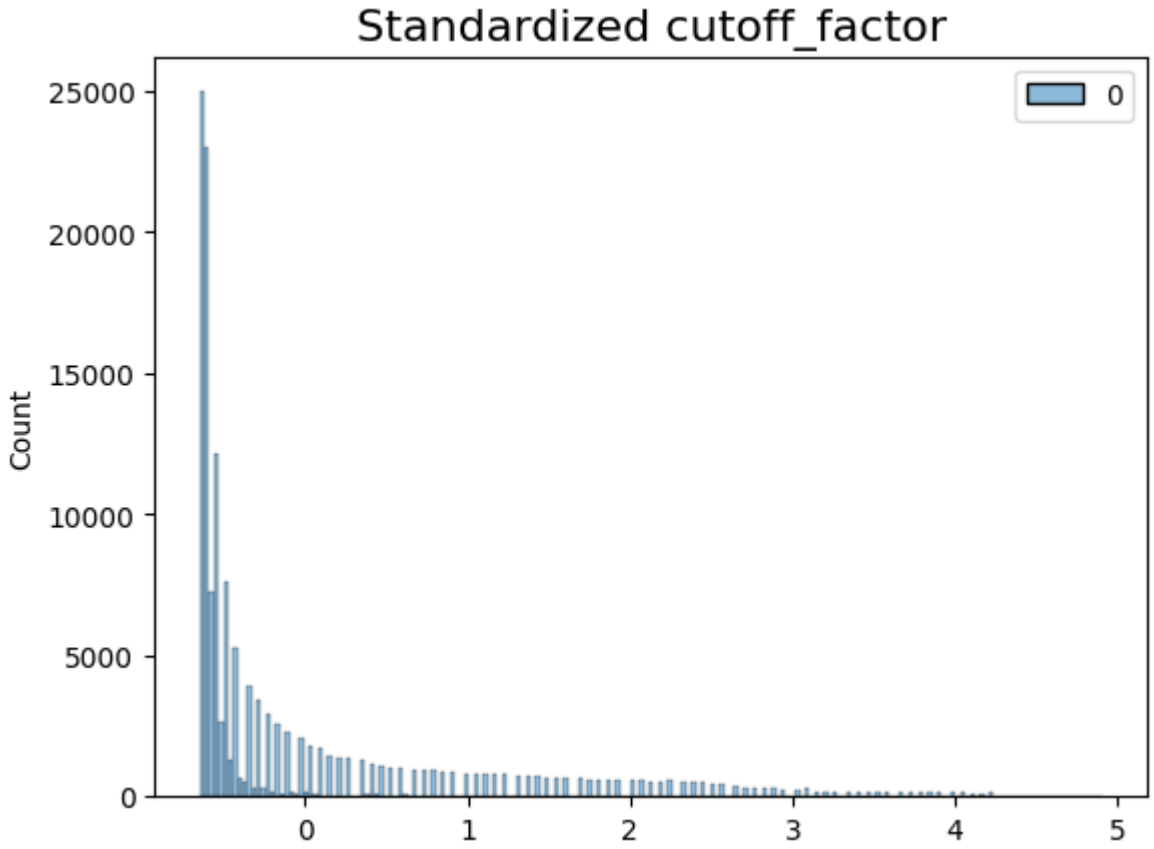

Normalized segment_osrm_distance
0
Count
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

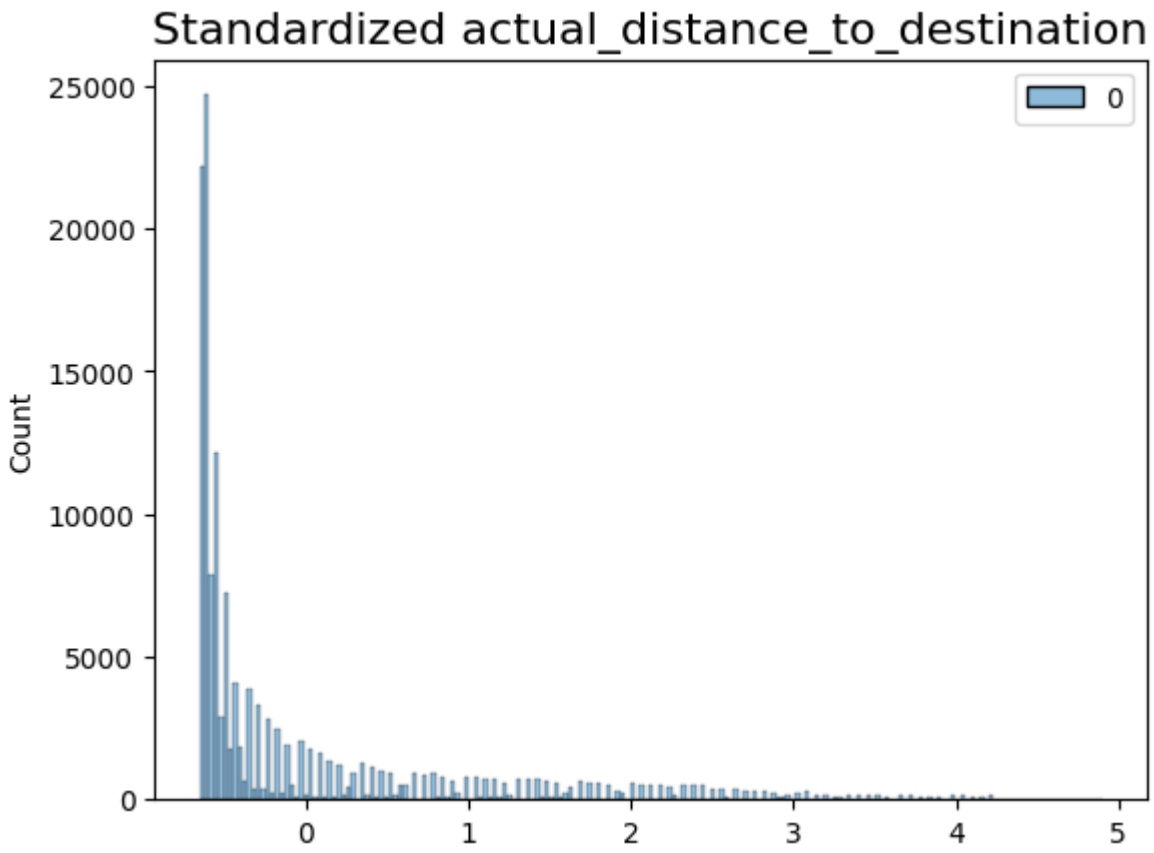


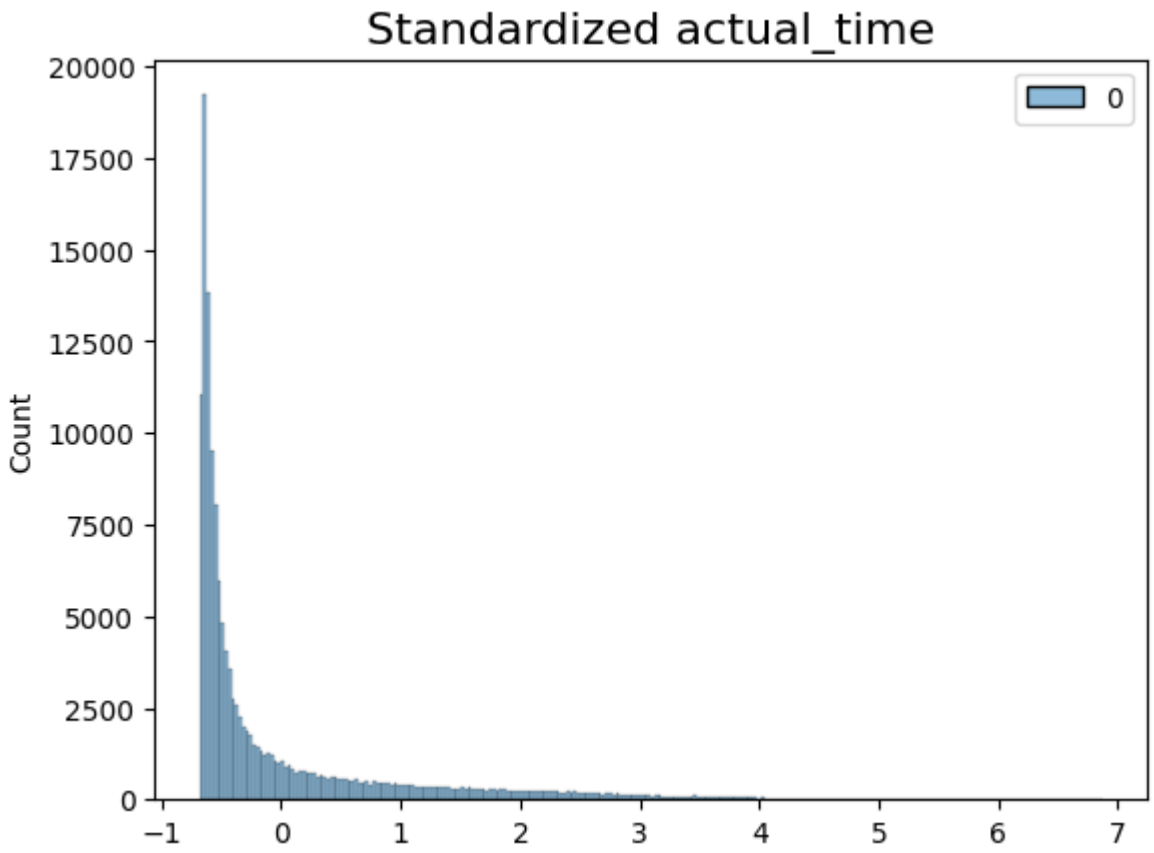
In [33]:
```
plt.figure(figsize = (10, 10))
for col in numerical_cols :
    scaler = StandardScaler()
    scaled = scaler.fit_transform(df[col].to_numpy().reshape(-1, 1))
    sns.histplot(scaled)
    plt.title(f"Standardized {col}", fontsize=16)
    plt.show()
```

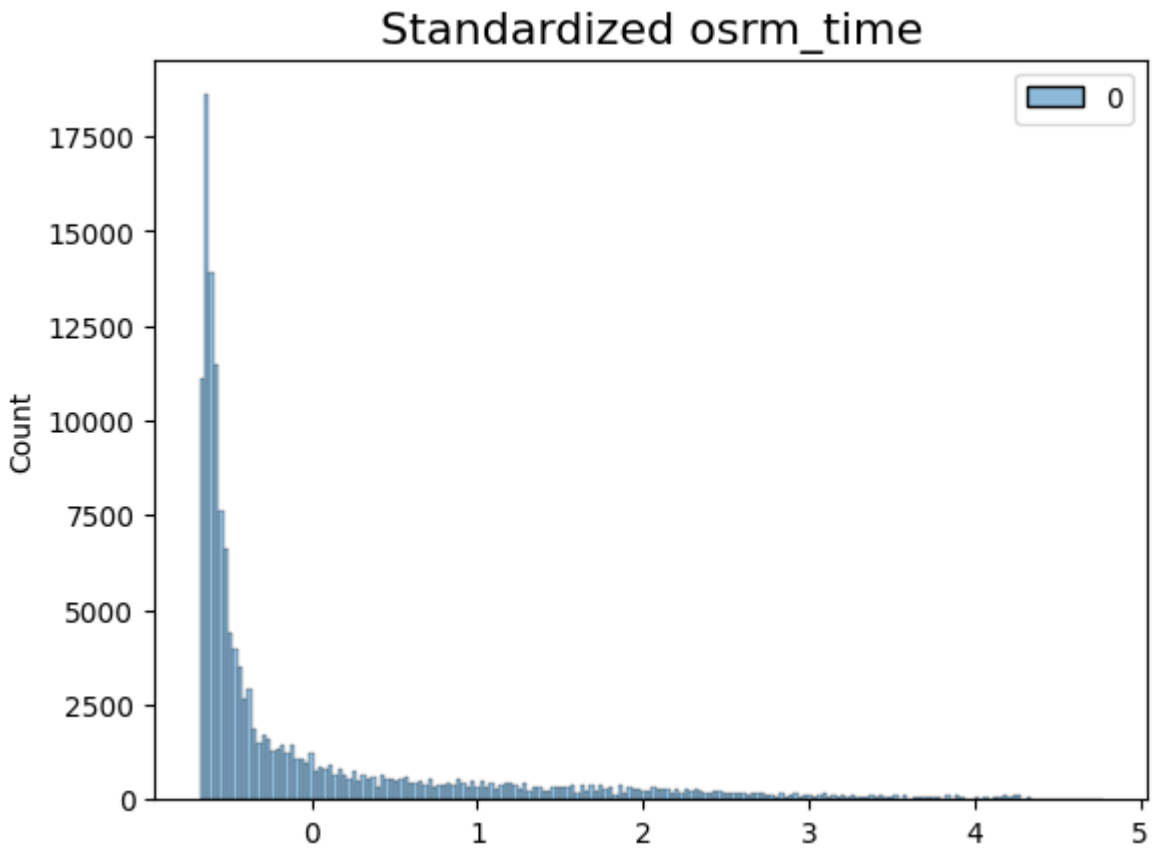

Standardized start_scan_to_end_scan
0
17500
15000
12500
10000
7500
5000
Count

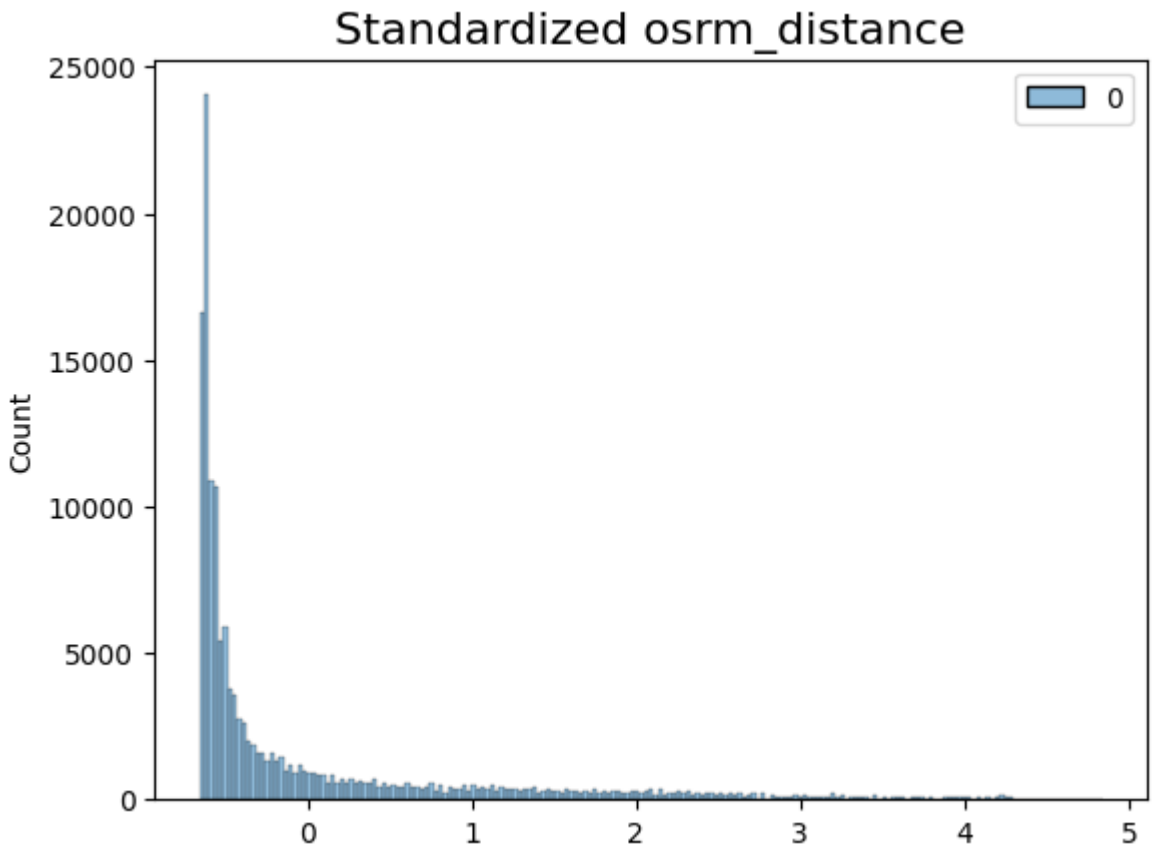

2500
0
−1
0
1
2
3
4
5
6
7

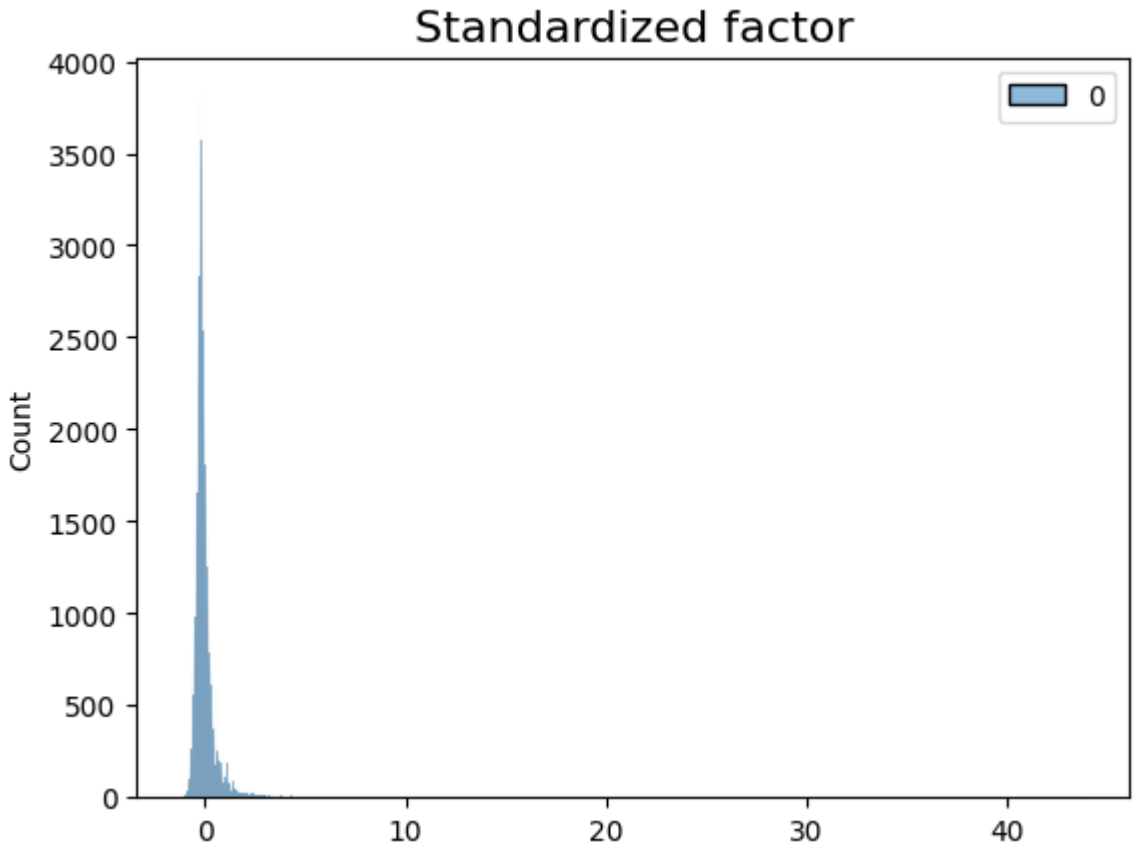

Standardized cutoff_factor
0
25000
20000
15000
10000
5000
0
Count
0
1
2
3
4
5

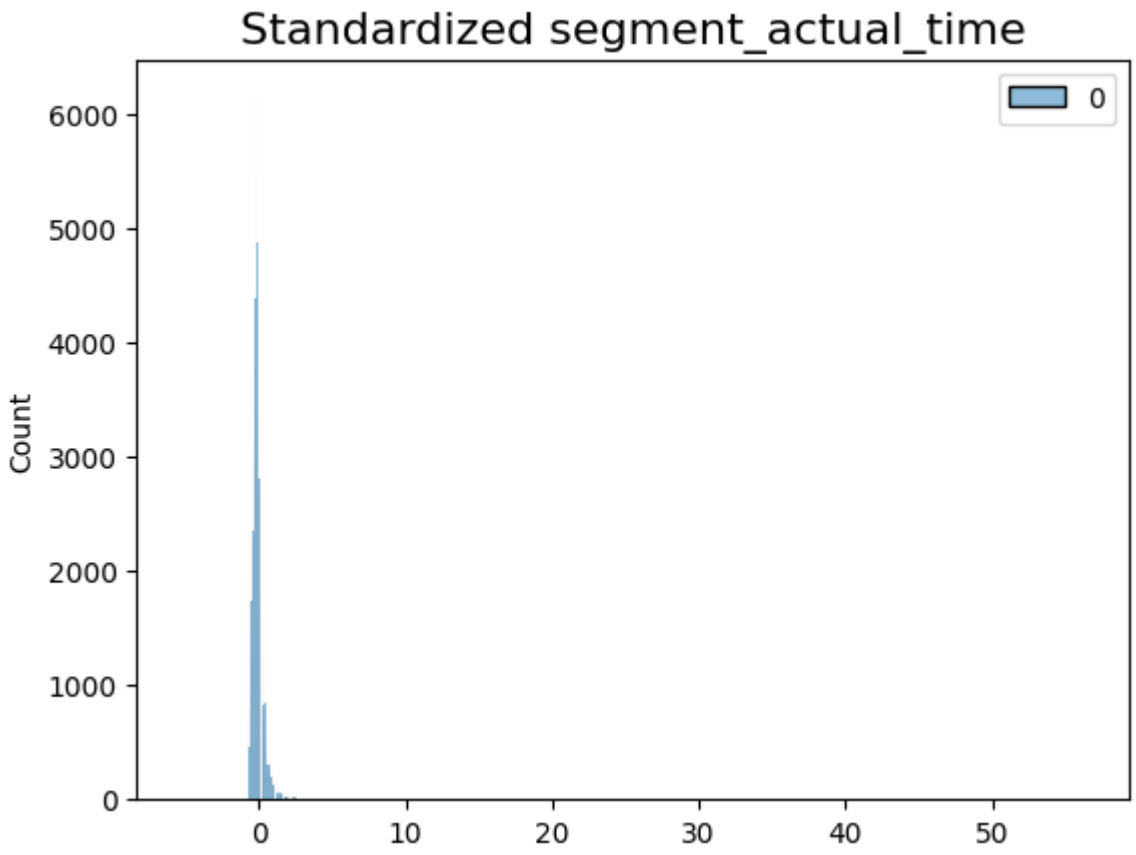

Standardized actual_distance_to_destination
0
Count
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

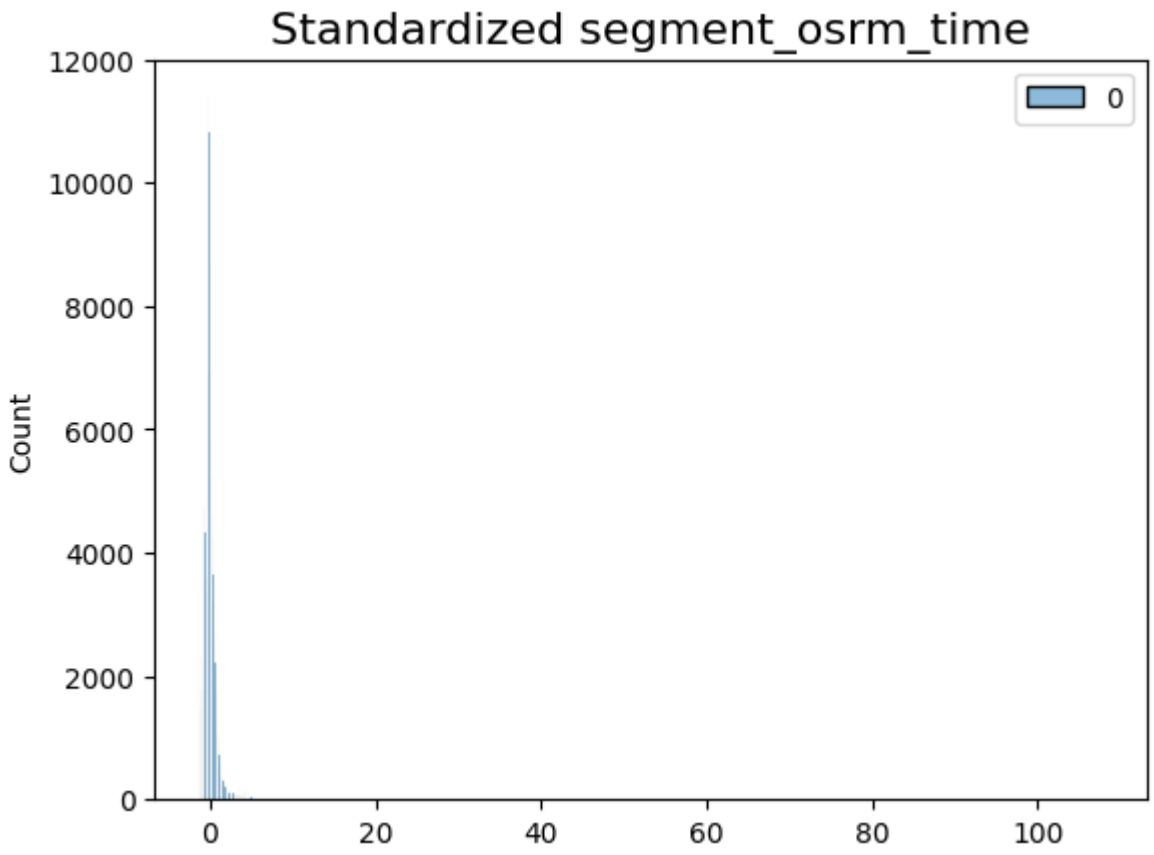

Standardized actual_time
0
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
−1
0
1
2
3
4
5
6
7

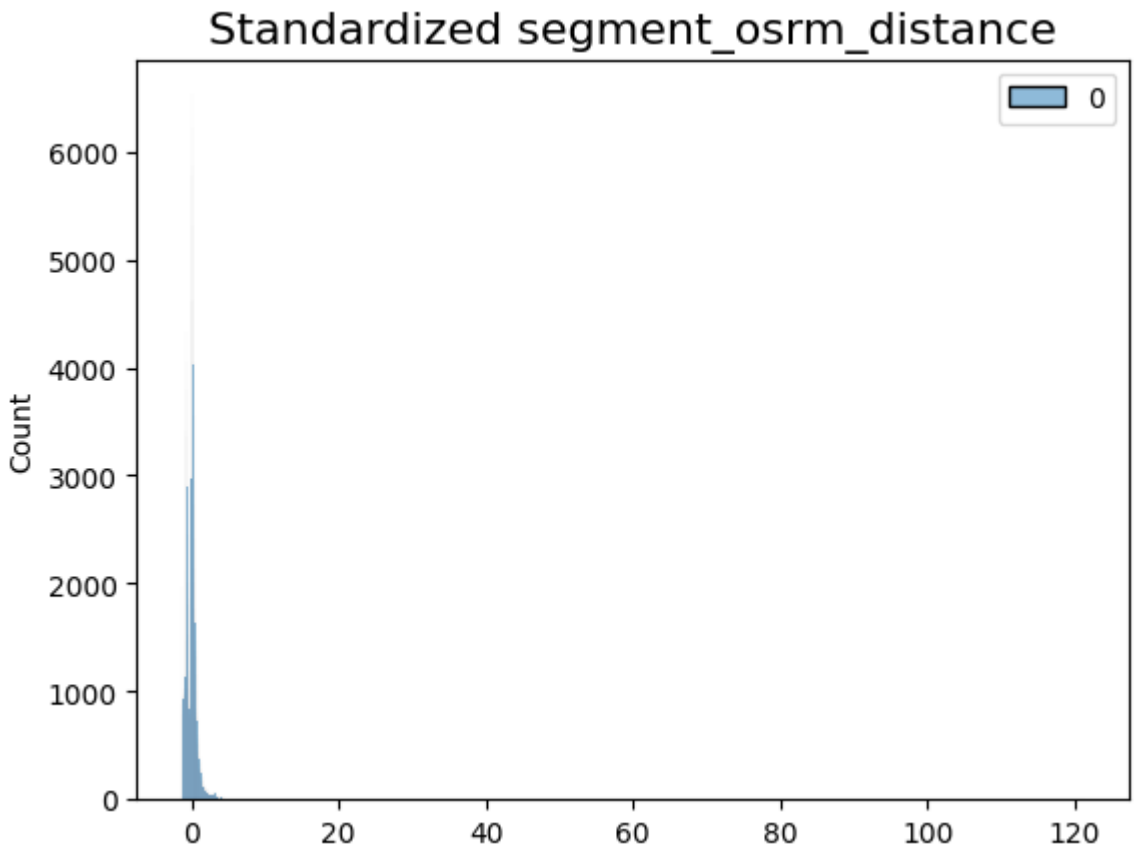

Standardized osrm_time
0
Count
17500
15000
12500
10000
7500
5000
2500
0
0
1
2
3
4
5

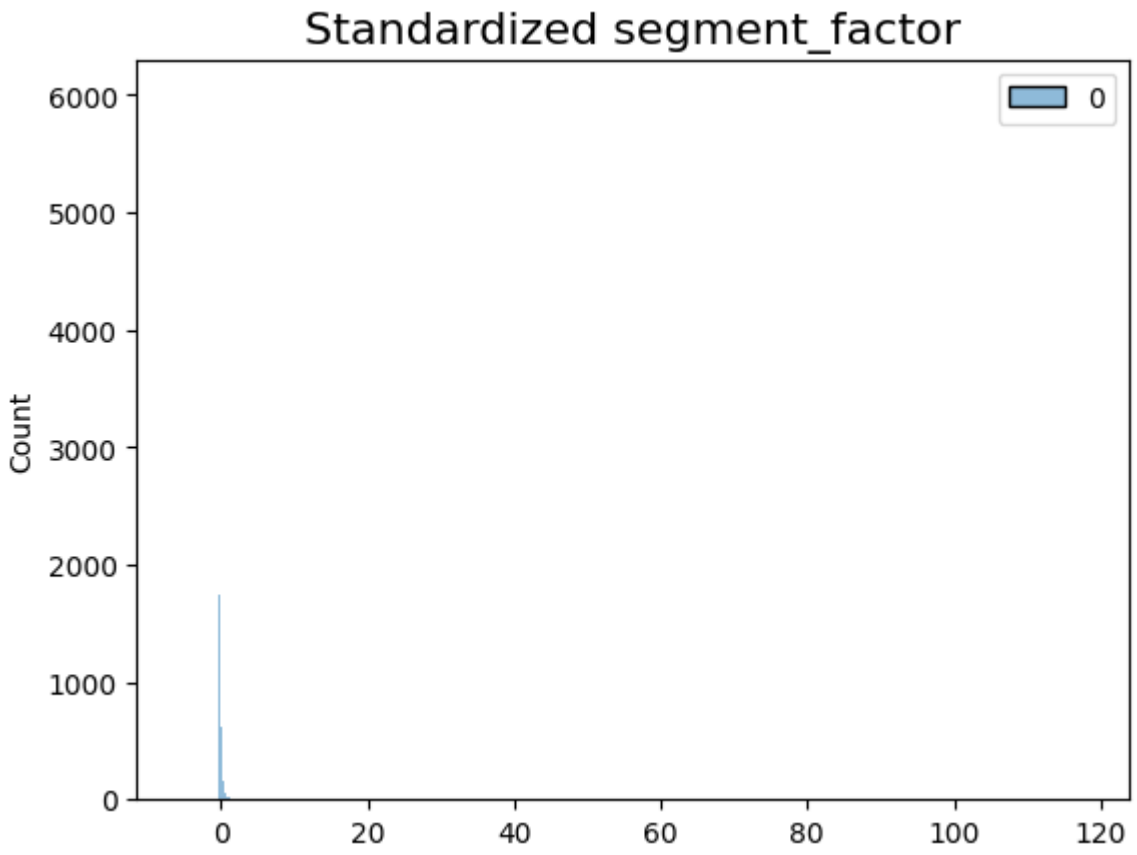

Standardized osrm_distance
25000
20000
15000
10000
5000
0
Count
0
0
1
2
3
4
5


Standardized factor
0
Count
4000
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40


Standardized segment_actual_time
0
Count
6000
5000
4000
3000
2000
1000
0
0
10
20
30
40
50


Standardized segment_osrm_time
0
Count
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100


Standardized segment_osrm_distance
0
Count
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
120



4 Hypothesis Testing Check / Visual Analysis

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

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

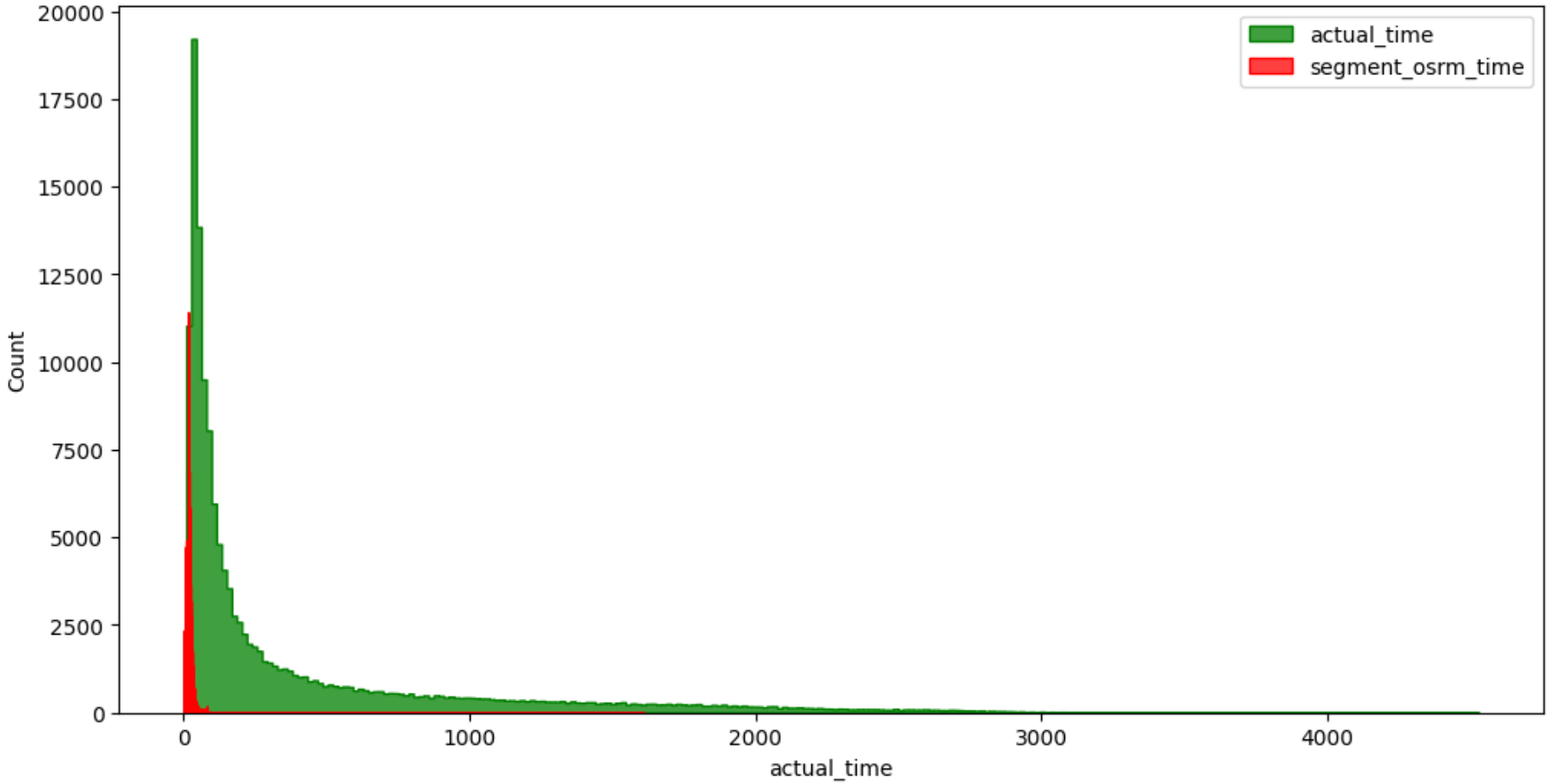
- The more -ve the coeff is, the more -ve the strenght will be.
- If the corrcoeff is 0 then there is no correlation between two numeric columns

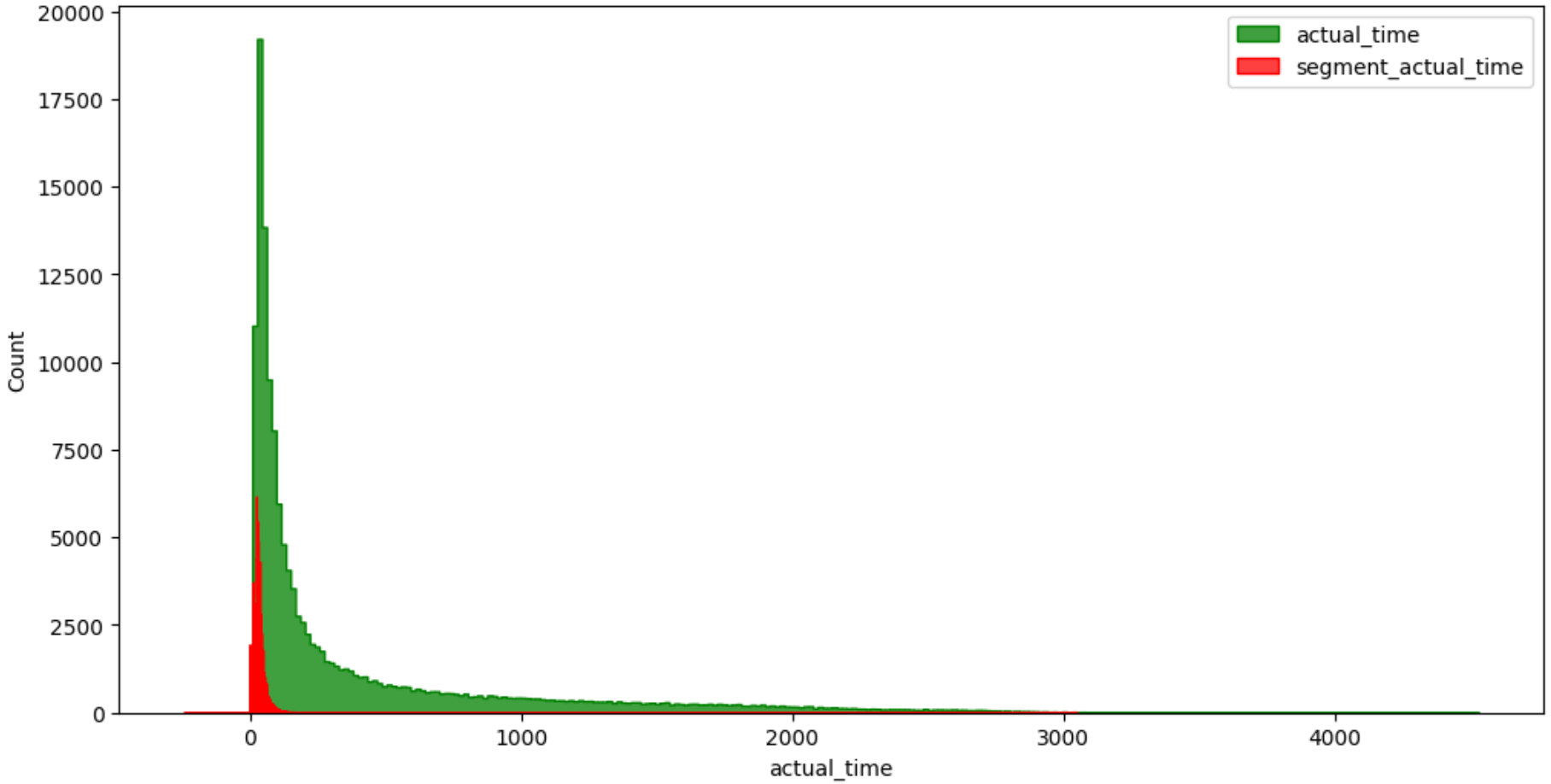
4.1 ActualTime Agg Value and OSRM Time Agg Value

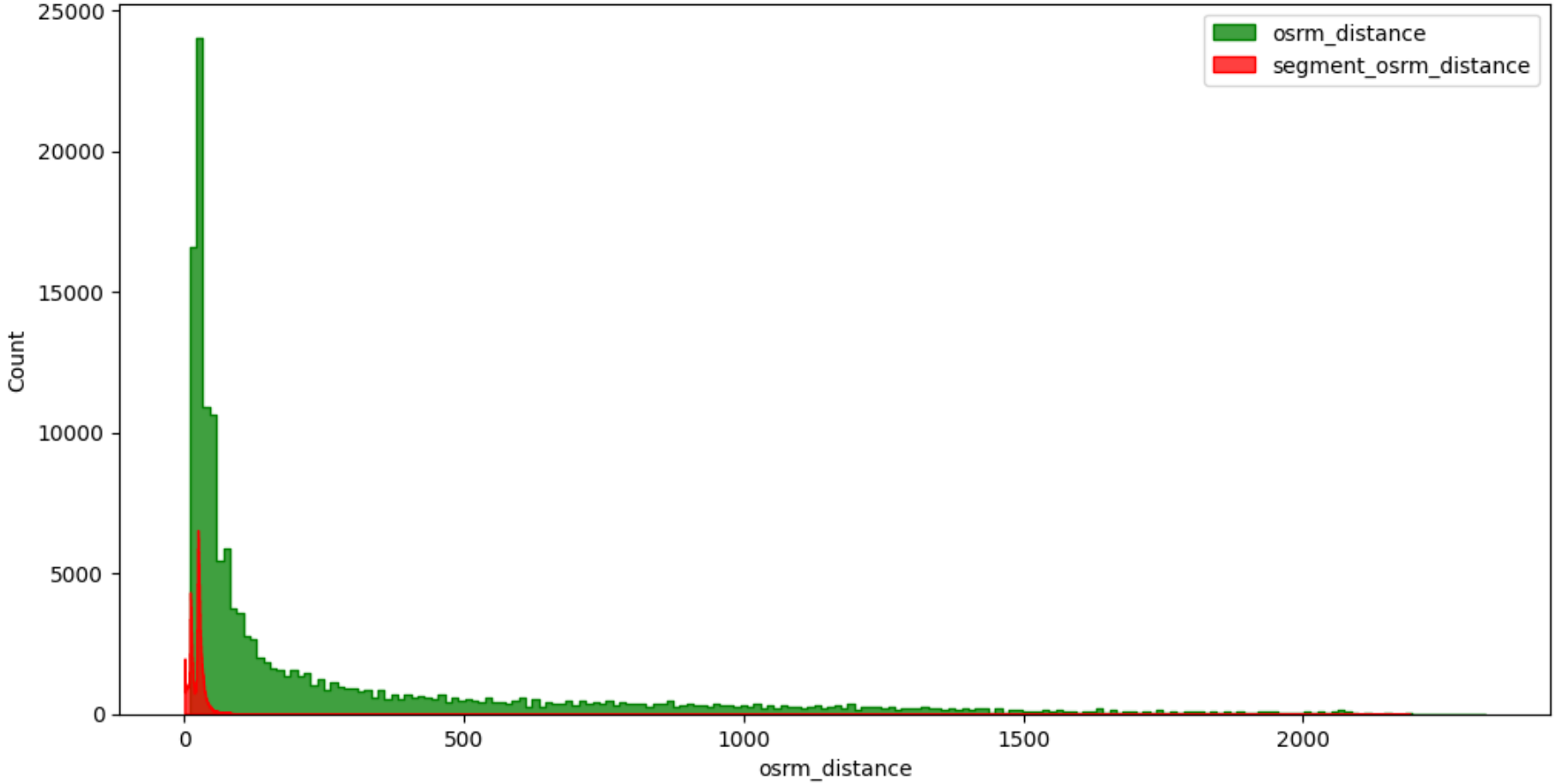
In [34]:
```
corr_elements = [['actual_time','segment_osrm_time'],
['actual_time','segment_actual_time'],
['osrm_distance','segment_osrm_distance'],
['osrm_time','segment_osrm_time']]
for col in corr_elements:
    val = np.corrcoef(df[col[0]].rank(), df[col[1]].rank())[0,1]
    if val> 0 :
        print(f'There is +ve relation between {col[0]} and', col[1],"- corrcoef: ",np.round(val,2))
    if val == 0:
        print(f'There is no relation between {col[0]} and ', col[1],"- corrcoef: ",np.round(val,2))
    if val< 0:
        print(f'There is -ve relation between {col[0]} and ',col[1],"- corrcoef: ",np.round(val,2))
```

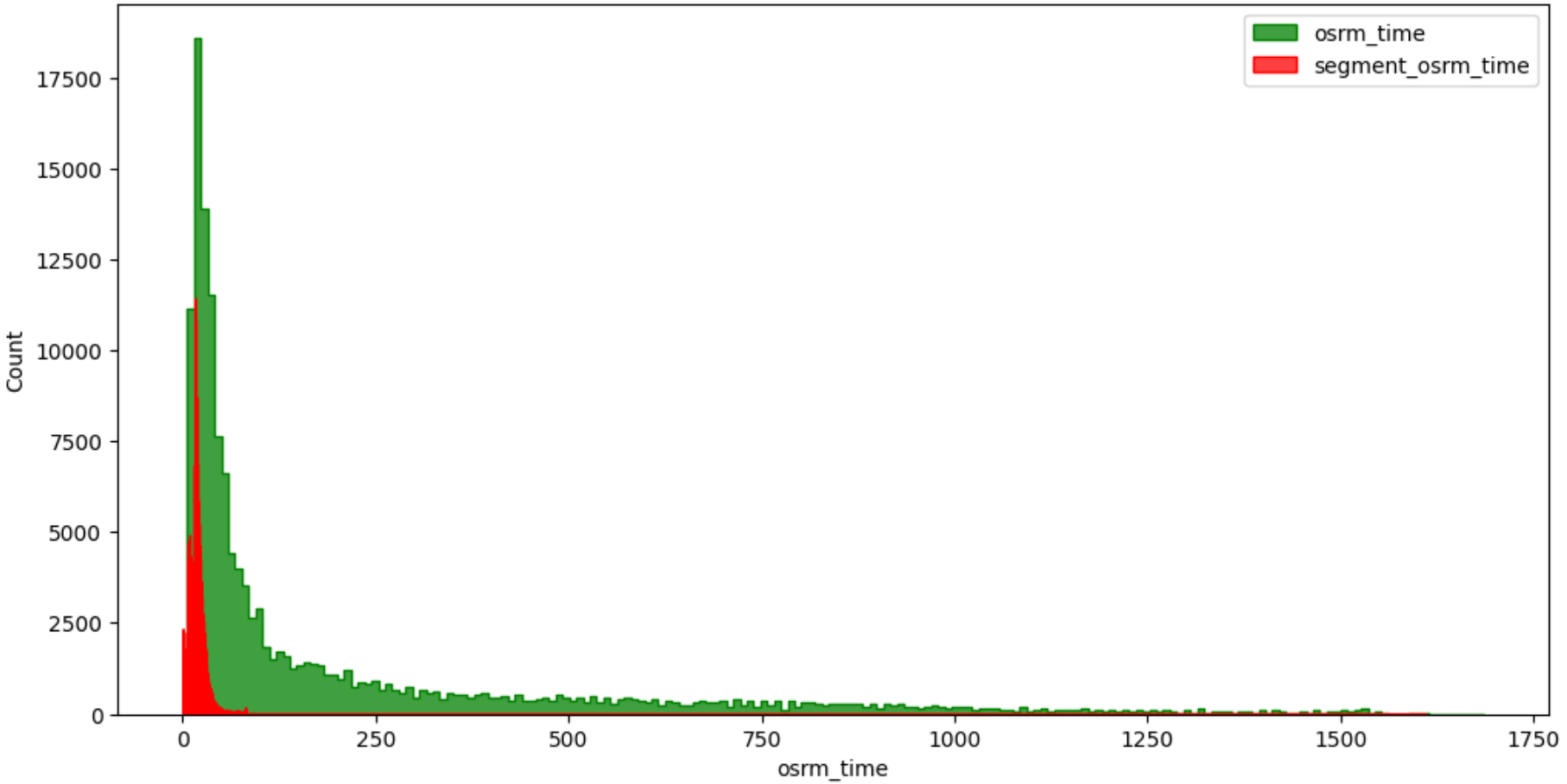
```
There is +ve relation between actual_time and segment_osrm_time - corrcoef:  0.37
There is +ve relation between actual_time and segment_actual_time - corrcoef:  0.36
There is +ve relation between osrm_distance and segment_osrm_distance - corrcoef:  0.52
There is +ve relation between osrm_time and segment_osrm_time - corrcoef:  0.43
```

In [35]:
```
for col in corr_elements:
    plt.figure(figsize = (12, 6))
    sns.histplot(df[col[0]], element = 'step', color = 'green')
    sns.histplot(df[col[1]], element = 'step', color = 'red')
    plt.legend([col[0], col[1]])
    plt.plot()
```


actual_time
segment_osrm_time
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
0
1000
2000
3000
4000
actual_time


actual_time
segment_actual_time
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
0
1000
2000
3000
4000
actual_time


osrm_distance
segment_osrm_distance
Count
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
osrm_distance

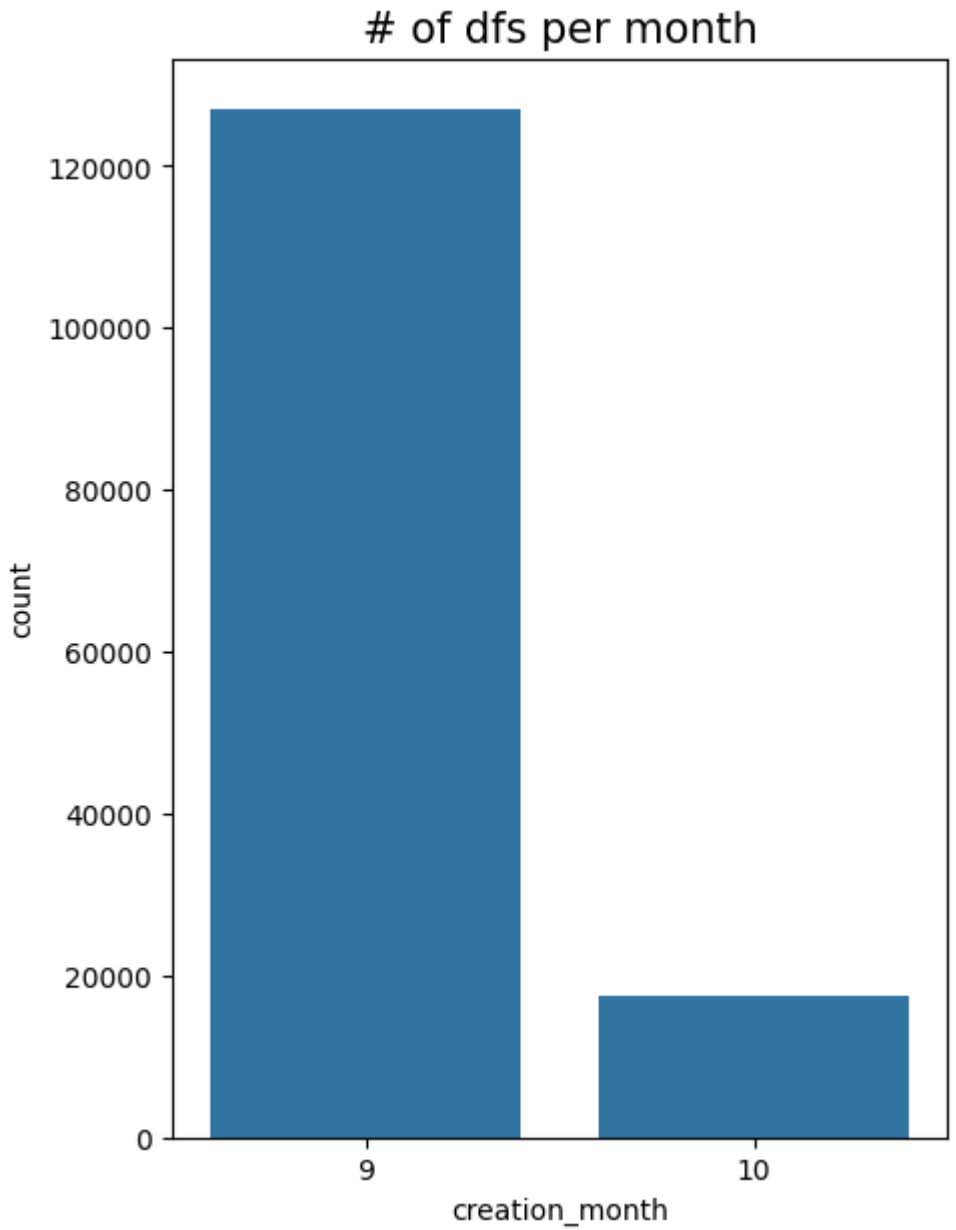


Insights:

- From the hypothesis testing and visual analysis we can observe that actual_time and segment_osrm_time_sum are less correlated than others.
- actual_time and segment_actual_time is 36% correlated.
- osrm_distance and segment_osrm_distance_sum and osrm_time and segment_osrm_time_sum are 43% correlated.
- This indicates ORSM navigation system has some glithces in time and distance calculation.

In [75]:
```
plt.figure(figsize=(5,7))
sns.countplot(x=df.creation_month)
```

```
plt.title("# of dfs per month", fontsize = 15)
plt.show()
```



Observation 💡

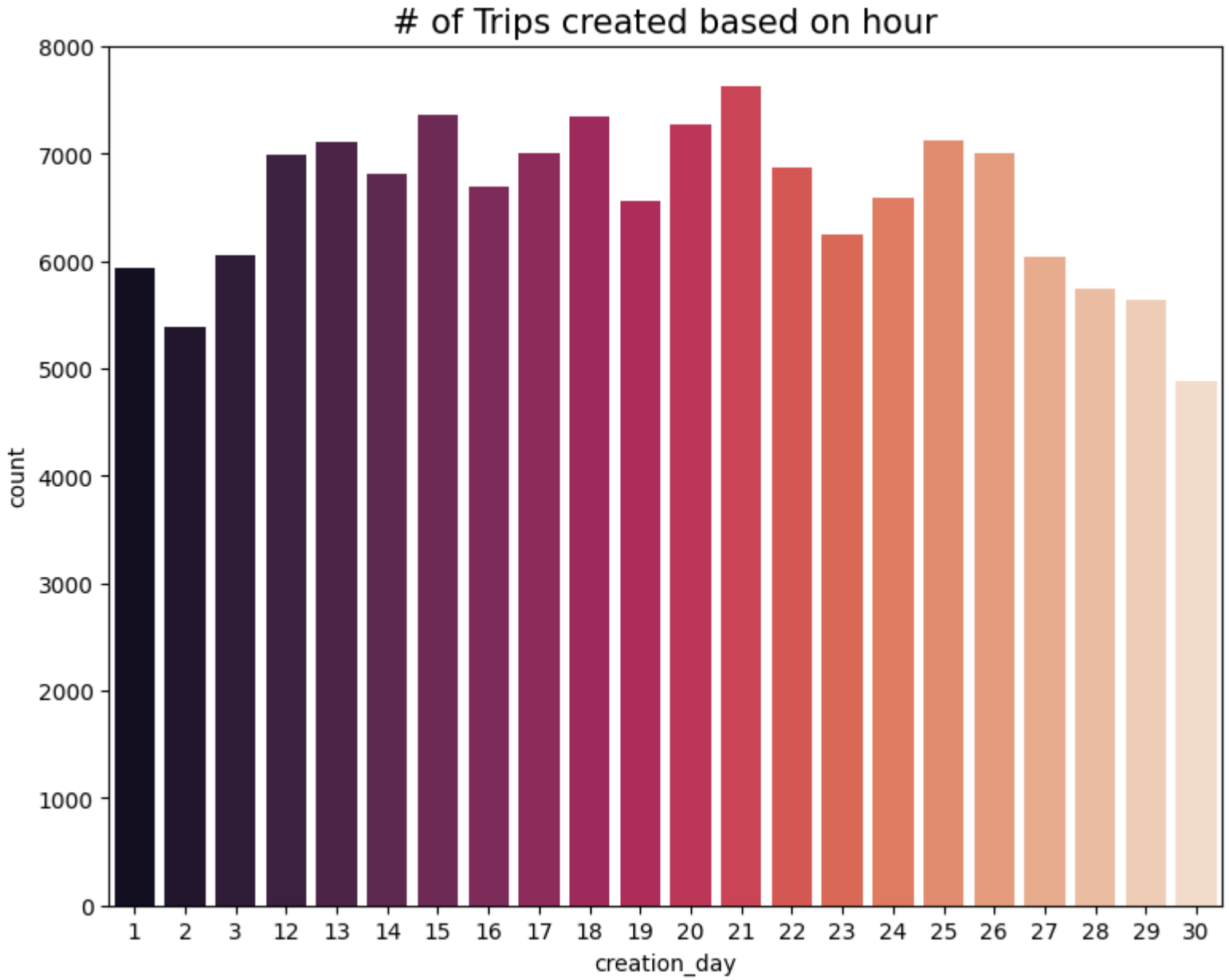
- More trips are created in the month of September

In [87]:
```
plt.figure(figsize=(9,7))
sns.countplot(x=df.creation_day,palette='rocket')
plt.title("# of Trips created based on hour", fontsize= 15)
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_4132\1888778620.py:2: FutureWarning:

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

  sns.countplot(x=df.creation_day,palette='rocket')
```



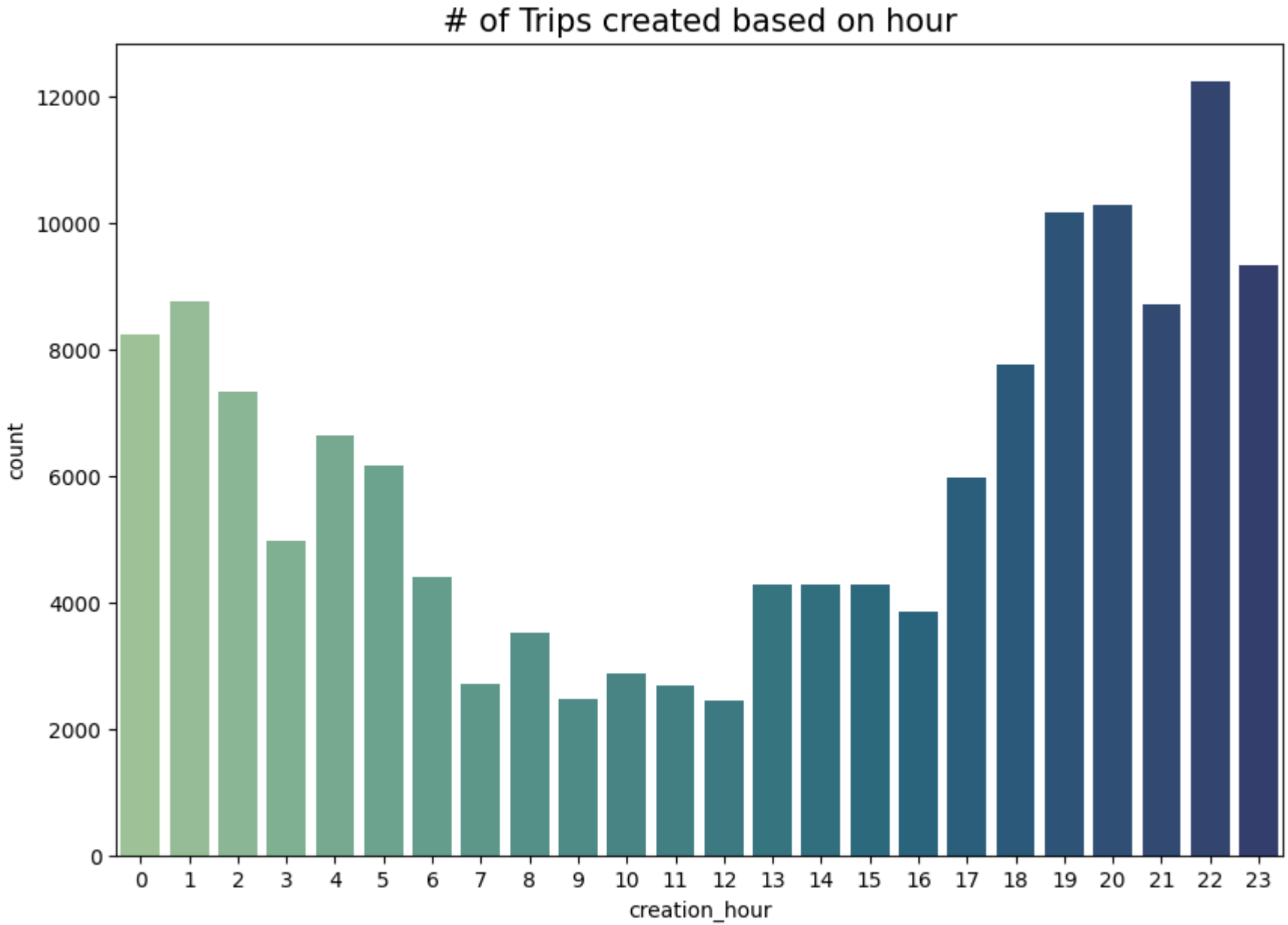
In [77]:
```
plt.figure(figsize=(10,7))
sns.countplot(x=df.creation_hour,palette='crest')
```

```
plt.title("# of Trips created based on hour", fontsize= 15)
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_4132\944955784.py:2: FutureWarning:

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

  sns.countplot(x=df.creation_hour,palette='crest')
```



Observation

- More Trips are created at night hours

In [78]:
```
df_source_state = df.groupby(by = 'source_state')['route_schedule_uuid'].count().to_frame().reset_index()
df_source_state['perc'] = np.round(df_source_state['route_schedule_uuid'] * 100/df_source_state['route_schedule_uuid'].sum(),
df_source_state = df_source_state.sort_values(by = 'perc', ascending =False)
df_source_state.head()
```

Out[78]:

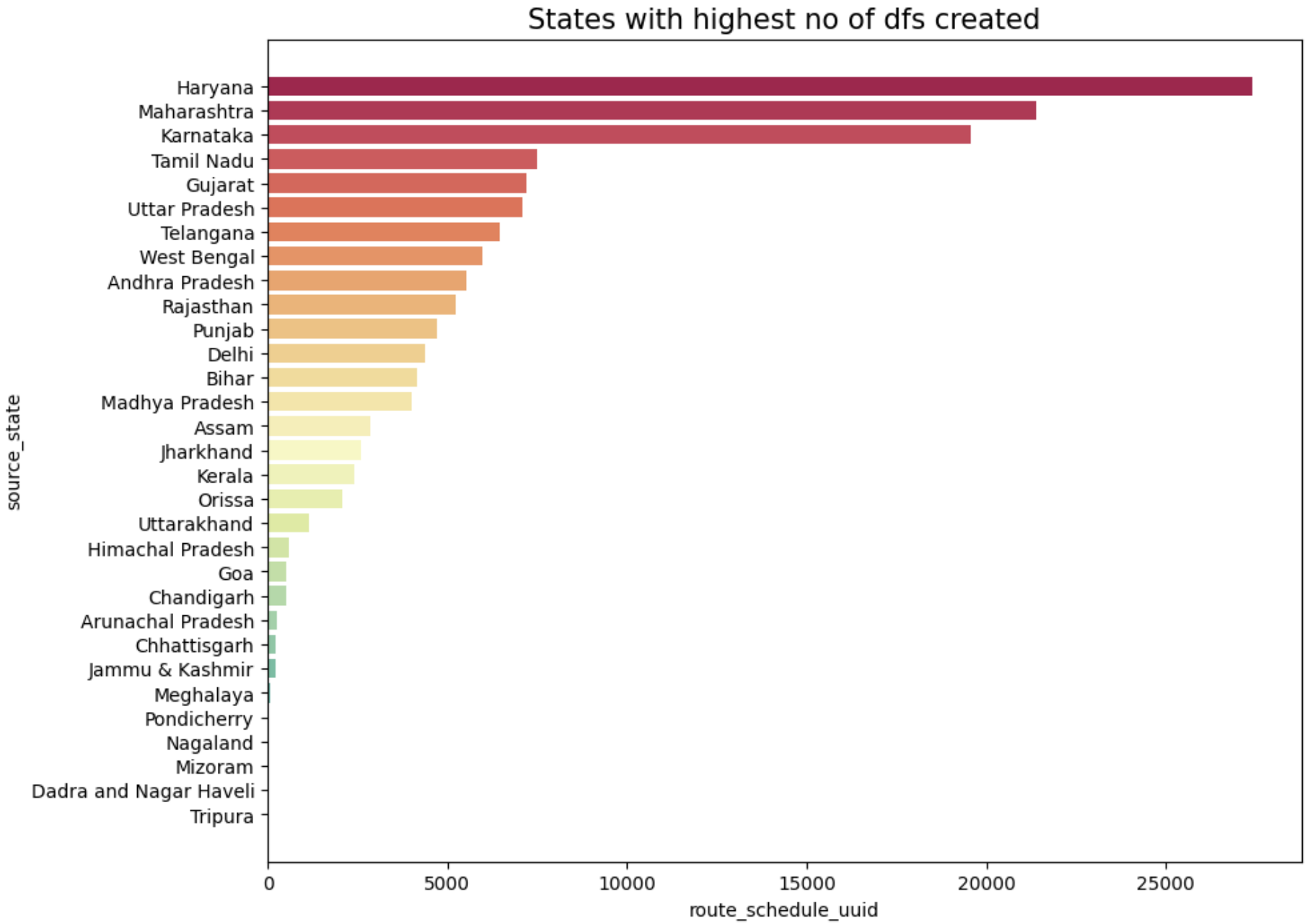
	source_state	route_schedule_uuid	perc
10	Haryana	27408	18.99
17	Maharashtra	21401	14.83
14	Karnataka	19562	13.55
25	Tamil Nadu	7494	5.19
9	Gujarat	7202	4.99

In [80]:
```
plt.figure(figsize = (10, 8))
sns.barplot(data = df_source_state,
    x = df_source_state['route_schedule_uuid'],
    y = df_source_state['source_state'],palette='Spectral')
plt.title("States with highest no of dfs created", fontsize=15)
plt.plot()
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_4132\66871328.py:2: FutureWarning:

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

  sns.barplot(data = df_source_state,
```



Observation 💡

- Haryana is the top state with highest no of dfs booked from.
- Second and third would be Maharastra and Karntaka.
- Jammu Kashmir, Chattisgarh are the least booking states.

In [81]:
```
# top 20 cities based on the number of dfs created from differentcities
df_source_city = df.groupby(by = 'source_city')['route_schedule_uuid'].count().to_frame().reset_index()
df_source_city['perc'] = np.round(df_source_city['route_schedule_uuid'] * 100/
df_source_city['route_schedule_uuid'].sum(), 2)
df_source_city = df_source_city.sort_values(by = 'perc', ascending =False)[:20]
df_source_city[:5]
```

Out[81]:

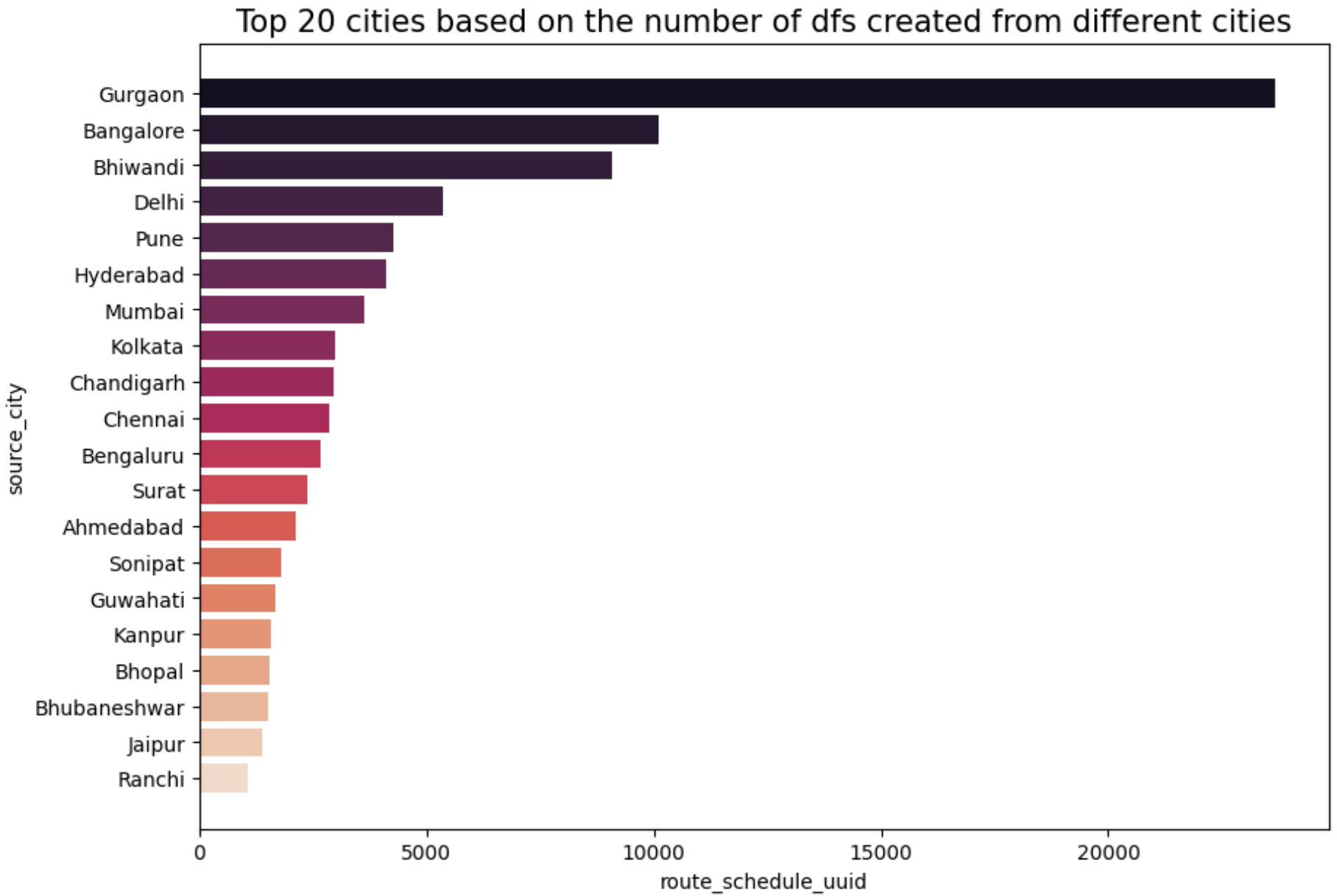
	source_city	route_schedule_uuid	perc
404	Gurgaon	23677	16.41
96	Bangalore	10104	7.00
165	Bhiwandi	9088	6.30
287	Delhi	5364	3.72
923	Pune	4269	2.96

In [83]:
```
plt.figure(figsize = (10, 7))
sns.barplot(data = df_source_city,
    x = df_source_city['route_schedule_uuid'],
    y = df_source_city['source_city'],palette='rocket')
plt.title("Top 20 cities based on the number of dfs created from different cities", fontsize=15)
plt.plot()
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_4132\837529839.py:2: FutureWarning:

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

  sns.barplot(data = df_source_city,
```



Observation 💡

- Gurgaon and Bangalore are Highest no of dfs booked from.

- Jaipur and Ranchi are the least booking states.

In [58]:
```
# Destinatoin States with highest no of dfs created to
df_destn_state = df.groupby(by = 'destination_state')['route_schedule_uuid'].count().to_frame().reset_index()
df_destn_state['perc'] = np.round(df_destn_state['route_schedule_uuid'] * 100/
df_destn_state['route_schedule_uuid'].sum(), 2)
df_destn_state = df_destn_state.sort_values(by = 'perc', ascending =False)
df_source_state.head()
```

Out[58]:

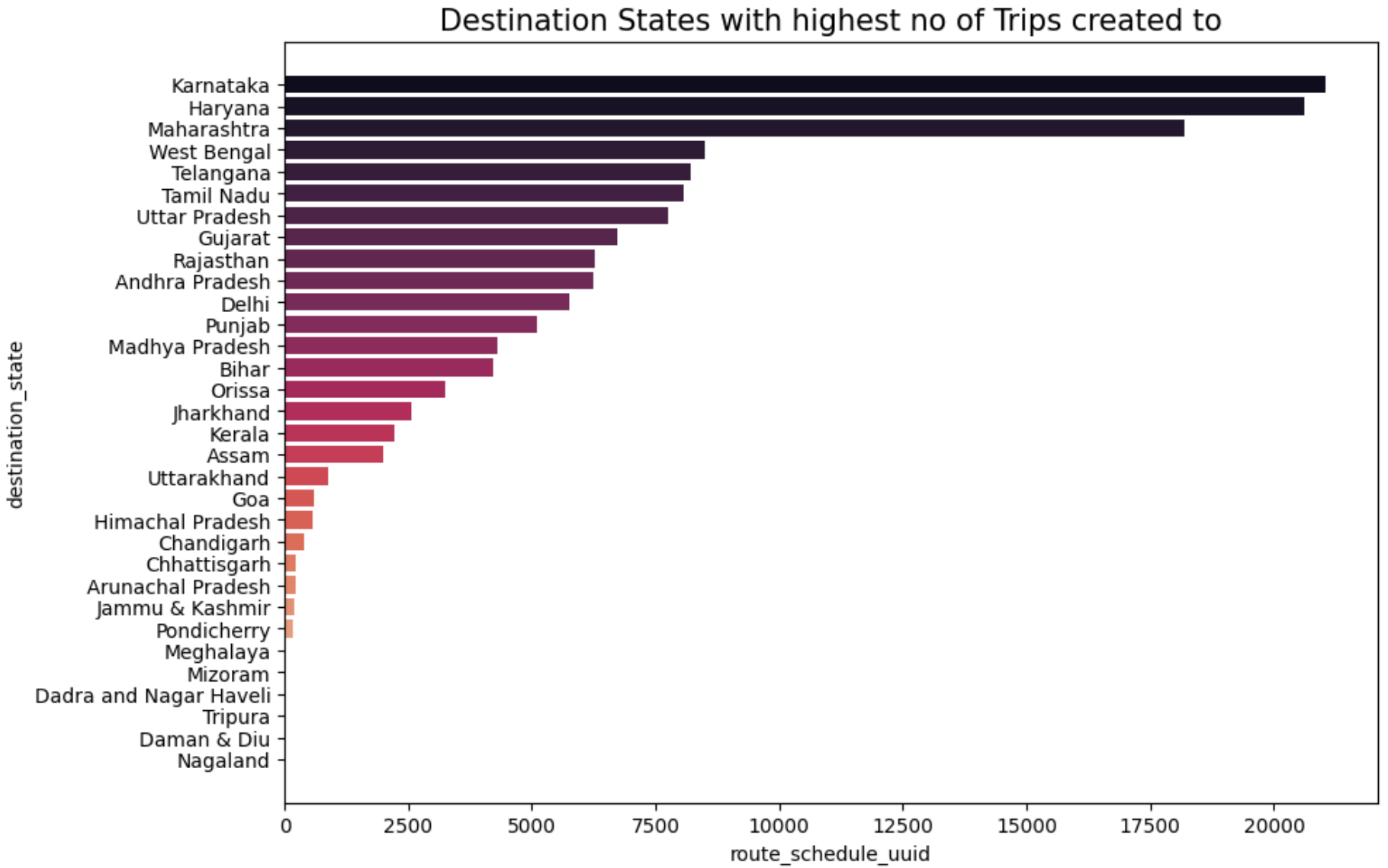
	source_state	route_schedule_uuid	perc
10	Haryana	27408	18.99
17	Maharashtra	21401	14.83
14	Karnataka	19562	13.55
25	Tamil Nadu	7494	5.19
9	Gujarat	7202	4.99

In [86]:
```
plt.figure(figsize = (10, 7))
sns.barplot(data = df_destn_state,
            x = df_destn_state['route_schedule_uuid'],
            y = df_destn_state['destination_state'],palette='rocket')
plt.title("Destination States with highest no of Trips created to",
fontsize = 15)
plt.plot()
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_4132\4054382361.py:2: FutureWarning:

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

  sns.barplot(data = df_destn_state,
```



In [68]:
```
# top 20 cities based on the number of trips created from different cities
df_destn_city = df.groupby(by = 'destination_city')['route_schedule_uuid'].count().to_frame().reset_index()
df_destn_city['perc'] = np.round(df_destn_city['route_schedule_uuid'] * 100/
df_destn_city['route_schedule_uuid'].sum(), 2)
```

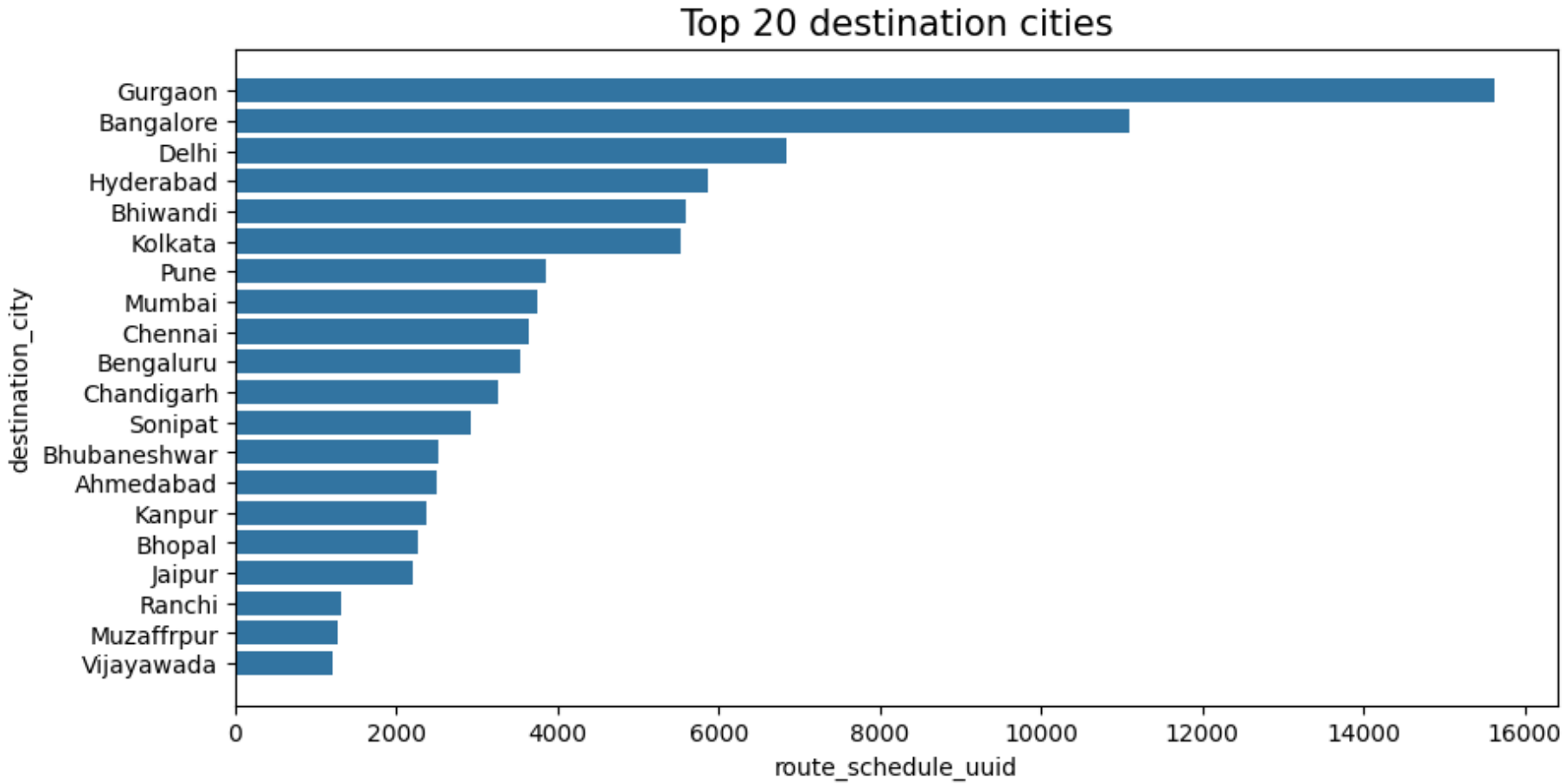
```
df_destn_city = df_destn_city.sort_values(by = 'perc', ascending =False)[:20]
df_destn_city[:5]
```

Out[68]:

	destination_city	route_schedule_uuid	perc
400	Gurgaon	15625	10.83
99	Bangalore	11087	7.68
283	Delhi	6842	4.74
439	Hyderabad	5875	4.07
164	Bhiwandi	5586	3.87

In [71]:

```
plt.figure(figsize = (10, 5))
sns.barplot(data = df_destn_city,
            x = df_destn_city['route_schedule_uuid'],
            y = df_destn_city['destination_city'])
plt.title("Top 20 destination cities",fontsize=15 )
plt.plot()
plt.show()
```


Top 20 destination cities
destination_city
Gurgaon
Bangalore
Delhi
Hyderabad
Bhiwandi
Kolkata
Pune
Mumbai
Chennai
Bengaluru
Chandigarh
Sonipat
Bhubaneshwar
Ahmedabad
Kanpur
Bhopal
Jaipur
Ranchi
Muzaffrpur
Vijayawada
0
2000
4000
6000
8000
10000
12000
14000
16000
route_schedule_uuid