Delhivery Business Case Study

Introduction

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

What is expected

The company wants to understand and process the data coming out of data engineering pipelines: \ • Clean, sanitize and manipulate data to get useful features out of raw fields \ • Make sense out of the raw data and help the data science team to build forecasting models on it.

1. Data

The analysis was done on the data located at - https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181

2. Libraries

Below are the libraries required for analysing and visualizing data

In [1]:
```
# libraries to analyze data
import numpy as np
import pandas as pd
import scipy.stats as sps

# libraries to visualize data
import matplotlib.pyplot as plt
import seaborn as sns

# Misc libraries
import random
```

3. Data loading and exploratory data analysis

Loading the data into Pandas dataframe for easily handling of data

In [2]:
```
# read the file into a pandas dataframe
df = pd.read_csv('delhivery_data.csv')
# look at the datatypes of the columns
print('**************************************************')
print(df.info())
print('**************************************************\n')
print('**************************************************')
```

```python
print(f'Shape of the dataset is {df.shape}')
print('*************************************************\n')
print('*************************************************')
print(f'Number of nan/null values in each column: \n{df.isna().sum()}')
print('*************************************************\n')
print('*************************************************')
print(f'Number of unique values in each column: \n{df.nunique()}')
print('*************************************************\n')
print('*************************************************')
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
print('*************************************************')
```

```
*************************************************
<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
None
*************************************************

*************************************************
Shape of the dataset is (144867, 24)
*************************************************

*************************************************
Number of nan/null values in each column: 
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
is_cutoff                           0
```

```
cutoff_factor                     0
cutoff_timestamp                  0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
factor                            0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
segment_factor                    0
dtype: int64
*************************************************

*************************************************
Number of unique values in each column:
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
is_cutoff                              2
cutoff_factor                        501
cutoff_timestamp                   93180
actual_distance_to_destination    144515
actual_time                         3182
osrm_time                           1531
osrm_distance                     138046
factor                             45641
segment_actual_time                  747
segment_osrm_time                    214
segment_osrm_distance             113799
segment_factor                      5675
dtype: int64
*************************************************

*************************************************
Duplicate entries:
False    144867
Name: count, dtype: int64
*************************************************
```

In [3]:
```
# look at the top 5 rows
df.head()
```

Out[3]:

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

3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUN
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUN

5 rows × 24 columns

Insight

- A quick look at the information of the data reveals that there are **144867 rows and 24 columns** implying 144867 trips have been made with each trip having information such as *trip_creation_time, trip_uuid, source_center, source_name, destination_center, destination_name* to name a few. Most of the datatype are either "object" or "float64" except for *is_cutoff* and *cutoff_factor*.
- We can also infer that **there are 293 missing values or null value in source_name and 261 missing values or null value in destination_name** in the dataset. As these numbers are small compared to dataset size, 144867, it is safe to drop the rows with the missing values
- There are **no duplicate entries**.
- As columns *is_cutoff, cutoff_factor, cutoff_timestamp, factor and segment_factor* are Unknown fields,there is no harm in dropping these columns.
- It makes sense to convert columns *data and route_type* to "category" datatype
- It makes sense to convert columns *trip_creation_time, od_start_time, od_end_time* to "datetime" datatype

In [4]:
```
df = df.dropna(how='any')
df = df.drop(columns = ["is_cutoff", "cutoff_factor", "cutoff_timestamp", "factor", "seg
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"], format='%Y-%m-%d %H:
df["od_start_time"] = pd.to_datetime(df["od_start_time"], format='%Y-%m-%d %H:%M:%S.%f')
df["od_end_time"] = pd.to_datetime(df["od_end_time"], format='%Y-%m-%d %H:%M:%S.%f')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  category
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  category
 4   trip_uuid                       144316 non-null  object
 5   source_center                   144316 non-null  object
 6   source_name                     144316 non-null  object
 7   destination_center              144316 non-null  object
 8   destination_name                144316 non-null  object
 9   od_start_time                   144316 non-null  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144316 non-null  float64
 12  actual_distance_to_destination  144316 non-null  float64
 13  actual_time                     144316 non-null  float64
 14  osrm_time                       144316 non-null  float64
 15  osrm_distance                   144316 non-null  float64
 16  segment_actual_time             144316 non-null  float64
 17  segment_osrm_time               144316 non-null  float64
```

```
 18  segment_osrm_distance         144316 non-null  float64
dtypes: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 20.1+ MB
```

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

Out[6]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destinat
count	144316	144316	144316	144316.000000	144316.000
mean	2018-09-22 13:05:09.454117120	2018-09-22 17:32:42.435769344	2018-09-23 09:36:54.057172224	963.697698	234.708
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.000
25%	2018-09-17 02:46:11.004421120	2018-09-17 07:37:35.014584832	2018-09-18 01:29:56.978912	161.000000	23.352
50%	2018-09-22 03:36:19.186585088	2018-09-22 07:35:23.038482944	2018-09-23 02:49:00.936600064	451.000000	66.135
75%	2018-09-27 17:53:19.027942912	2018-09-27 22:01:30.861209088	2018-09-28 12:13:41.675546112	1645.000000	286.919
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.447
std	NaN	NaN	NaN	1038.082976	345.480

Insight

- The data is provided from **2018-09-12 00:00:16.535741 to 2018-10-03 23:59:42.701692**
- The **average time** taken to deliver from source to destination is **964 mins** with **least time being 20mins** and **maximum time being 7898 mins**
- The **average distance** between source and destination warehouse is **235 Kms** with **least distance being 9 Kms** and **maximum distance being 1927 Kms**

4. Detailed Analysis

4.1. Detecting outliers

4.1.1. Outliers for every continuous variable

In [7]:
```
# helper function to detect outliers
def detectOutliers(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3-q1
    lower_outliers = df[df<(q1-1.5*iqr)]
    higher_outliers = df[df>(q3+1.5*iqr)]
    return lower_outliers, higher_outliers
```

In [8]:
```
numerical_columns = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual
column_outlier_dictionary = {}
for column in numerical_columns:
    print('*'*50)
    print(f'Outliers of \'{column}\' column are:')
```

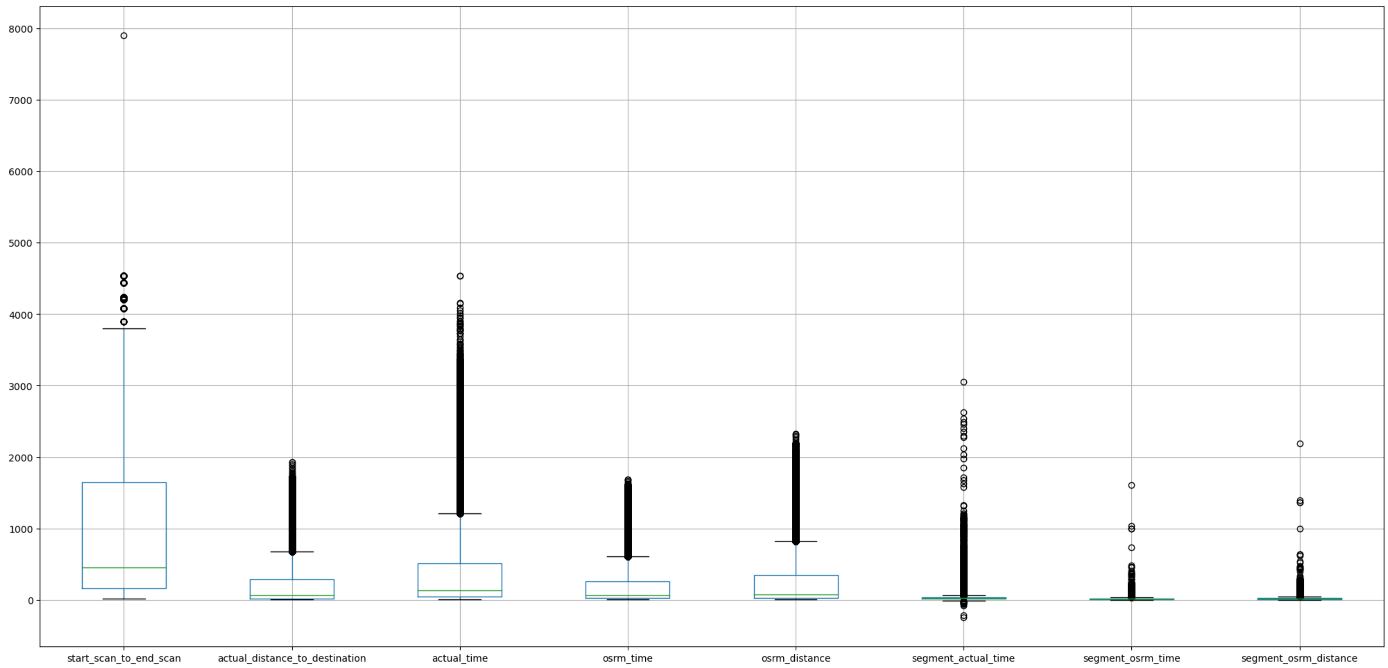
```
    lower_outliers, higher_outliers = detectOutliers(df[column])
    print("Lower outliers:\n", lower_outliers)
    print("Higher outliers:\n", higher_outliers)
    print('*'*50, end="\n")
    column_outlier_dictionary[column] = [lower_outliers, higher_outliers]
```

```
**************************************************
Outliers of 'start_scan_to_end_scan' column are:
Lower outliers:
 Series([], Name: start_scan_to_end_scan, dtype: float64)
Higher outliers:
 32950     3897.0
32951     3897.0
32952     3897.0
32953     3897.0
32954     3897.0
           ...
79524     4239.0
79525     4239.0
79526     4239.0
79527     4239.0
123196    7898.0
Name: start_scan_to_end_scan, Length: 373, dtype: float64
**************************************************
**************************************************
Outliers of 'actual_distance_to_destination' column are:
Lower outliers:
 Series([], Name: actual_distance_to_destination, dtype: float64)
Higher outliers:
 402        704.090688
403        726.181078
404        748.332196
405        770.365887
406        796.335857
             ...
144796    1611.171536
144797    1633.419313
144798    1650.202066
144799    1673.310381
144800    1689.639499
Name: actual_distance_to_destination, Length: 17818, dtype: float64
**************************************************
**************************************************
Outliers of 'actual_time' column are:
Lower outliers:
 Series([], Name: actual_time, dtype: float64)
Higher outliers:
 407       1241.0
408       1277.0
409       1305.0
410       1322.0
411       1352.0
           ...
144796    2640.0
144797    2675.0
144798    2700.0
144799    2736.0
144800    2784.0
Name: actual_time, Length: 16507, dtype: float64
**************************************************
**************************************************
Outliers of 'osrm_time' column are:
Lower outliers:
 Series([], Name: osrm_time, dtype: float64)
Higher outliers:
 405        630.0
```

```
406        641.0
407        655.0
408        671.0
409        696.0
           ...
144796    1492.0
144797    1512.0
144798    1532.0
144799    1549.0
144800    1508.0
Name: osrm_time, Length: 17406, dtype: float64
**************************************************
**************************************************
Outliers of 'osrm_distance' column are:
Lower outliers:
 Series([], Name: osrm_distance, dtype: float64)
Higher outliers:
 405        850.4080
406        865.7213
407        886.1183
408        908.4596
409        944.6344
             ...
144796    1980.0975
144797    2008.9586
144798    2036.3992
144799    2059.0195
144800    2063.7663
Name: osrm_distance, Length: 17547, dtype: float64
**************************************************
**************************************************
Outliers of 'segment_actual_time' column are:
Lower outliers:
 1805      -26.0
3761      -21.0
39825     -58.0
40942    -211.0
56464     -12.0
58697     -36.0
70479     -42.0
73603     -51.0
85042    -244.0
100205    -74.0
119377    -48.0
125821    -16.0
142409    -15.0
Name: segment_actual_time, dtype: float64
Higher outliers:
 21         93.0
34         94.0
72         75.0
73         78.0
106        79.0
          ...
144790     83.0
144819     88.0
144848    302.0
144853     91.0
144866    268.0
Name: segment_actual_time, Length: 9249, dtype: float64
**************************************************
**************************************************
Outliers of 'segment_osrm_time' column are:
Lower outliers:
 Series([], Name: segment_osrm_time, dtype: float64)
Higher outliers:
```

```
 34        70.0
38        45.0
157       81.0
158       81.0
214       44.0
          ...
144802    48.0
144829    74.0
144837    42.0
144843    43.0
144845    54.0
Name: segment_osrm_time, Length: 6348, dtype: float64
**************************************************
**************************************************
Outliers of 'segment_osrm_distance' column are:
Lower outliers:
 Series([], Name: segment_osrm_distance, dtype: float64)
Higher outliers:
 34        72.5561
157       79.6653
158       82.4127
214       52.7136
316       60.0755
           ...
144774    60.6393
144802    61.0445
144829    70.0436
144837    60.4795
144845    55.6993
Name: segment_osrm_distance, Length: 4295, dtype: float64
**************************************************
```

In [9]:
```
df[numerical_columns].boxplot(figsize=(25,12))
plt.show()
```



In [10]:
```
for key, value in column_outlier_dictionary.items():
    print(f'The column \'{key}\' has {len(value[0]) + len(value[1])} outliers')
```

```
The column 'start_scan_to_end_scan' has 373 outliers
The column 'actual_distance_to_destination' has 17818 outliers
The column 'actual_time' has 16507 outliers
The column 'osrm_time' has 17406 outliers
The column 'osrm_distance' has 17547 outliers
The column 'segment_actual_time' has 9262 outliers
```

```
The column 'segment_osrm_time' has 6348 outliers
The column 'segment_osrm_distance' has 4295 outliers
```

Insight

- I will not be removing any outliers now.

4.1.2. Remove the outliers

In [11]:

```
remove_outliers = False
if True == remove_outliers:
    for key, value in column_outlier_dictionary.items():
        lower_outliers = value[0]
        higher_outliers = value[1]
        df.drop(lower_outliers.index, inplace=True)
        df.drop(higher_outliers.index, inplace=True)
else:
    print('Not removing any outliers')
```

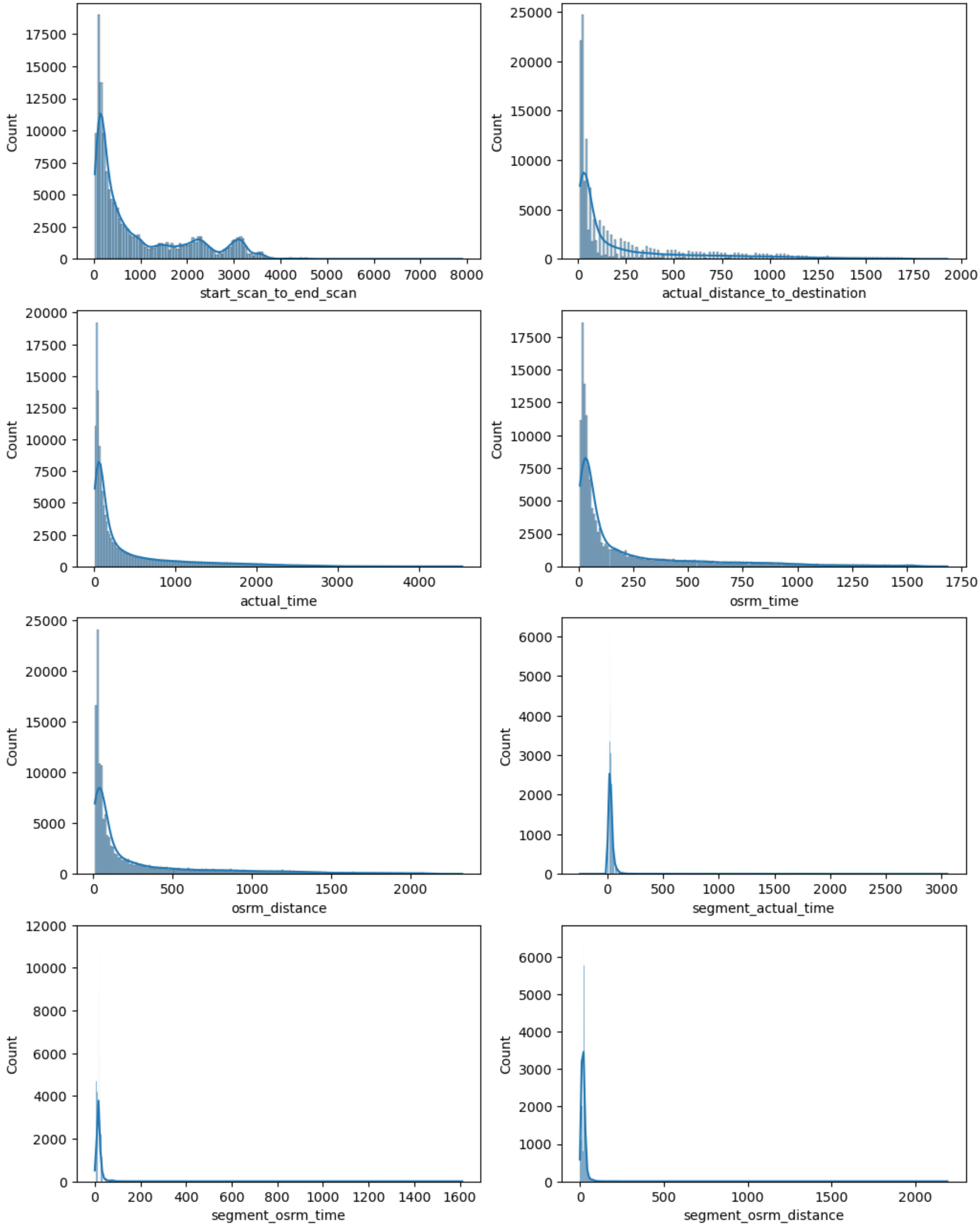
```
Not removing any outliers
```

4.2. Univariate analysis

4.2.1. Numerical Variables

In [12]:

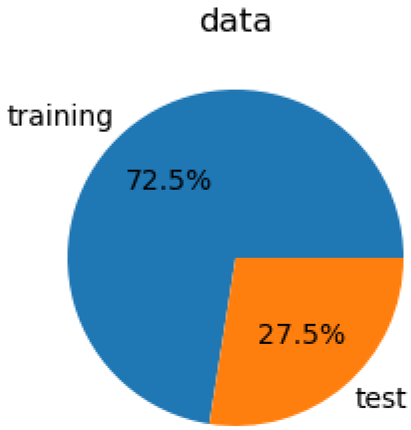
```
fig, ax = plt.subplots(nrows=4, ncols=2, figsize = (12, 16))
sns.histplot(data=df, x = "start_scan_to_end_scan", kde=True, ax=ax[0,0])
sns.histplot(data=df, x = "actual_distance_to_destination", kde=True, ax=ax[0,1])
sns.histplot(data=df, x = "actual_time", kde=True, ax=ax[1,0])
sns.histplot(data=df, x = "osrm_time", kde=True, ax=ax[1,1])
sns.histplot(data=df, x = "osrm_distance", kde=True, ax=ax[2,0])
sns.histplot(data=df, x = "segment_actual_time", kde=True, ax=ax[2,1])
sns.histplot(data=df, x = "segment_osrm_time", kde=True, ax=ax[3,0])
sns.histplot(data=df, x = "segment_osrm_distance", kde=True, ax=ax[3,1])
plt.show()
```

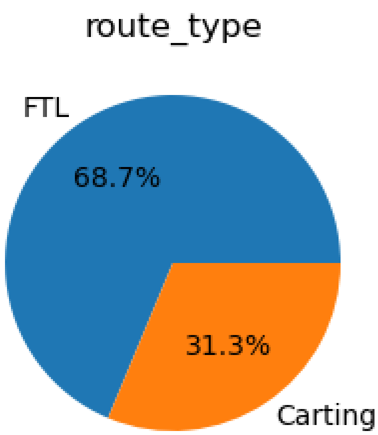


4.2.2. Categorical Variables

In [13]:
```
categorical_columns = ["data", "route_type"]
plt.figure(figsize=(6,6))
plt.subplot(2,1,1)
data = df["data"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("data")
plt.subplot(2,1,2)
```

```
data = df["route_type"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("route_type")
plt.show()
```



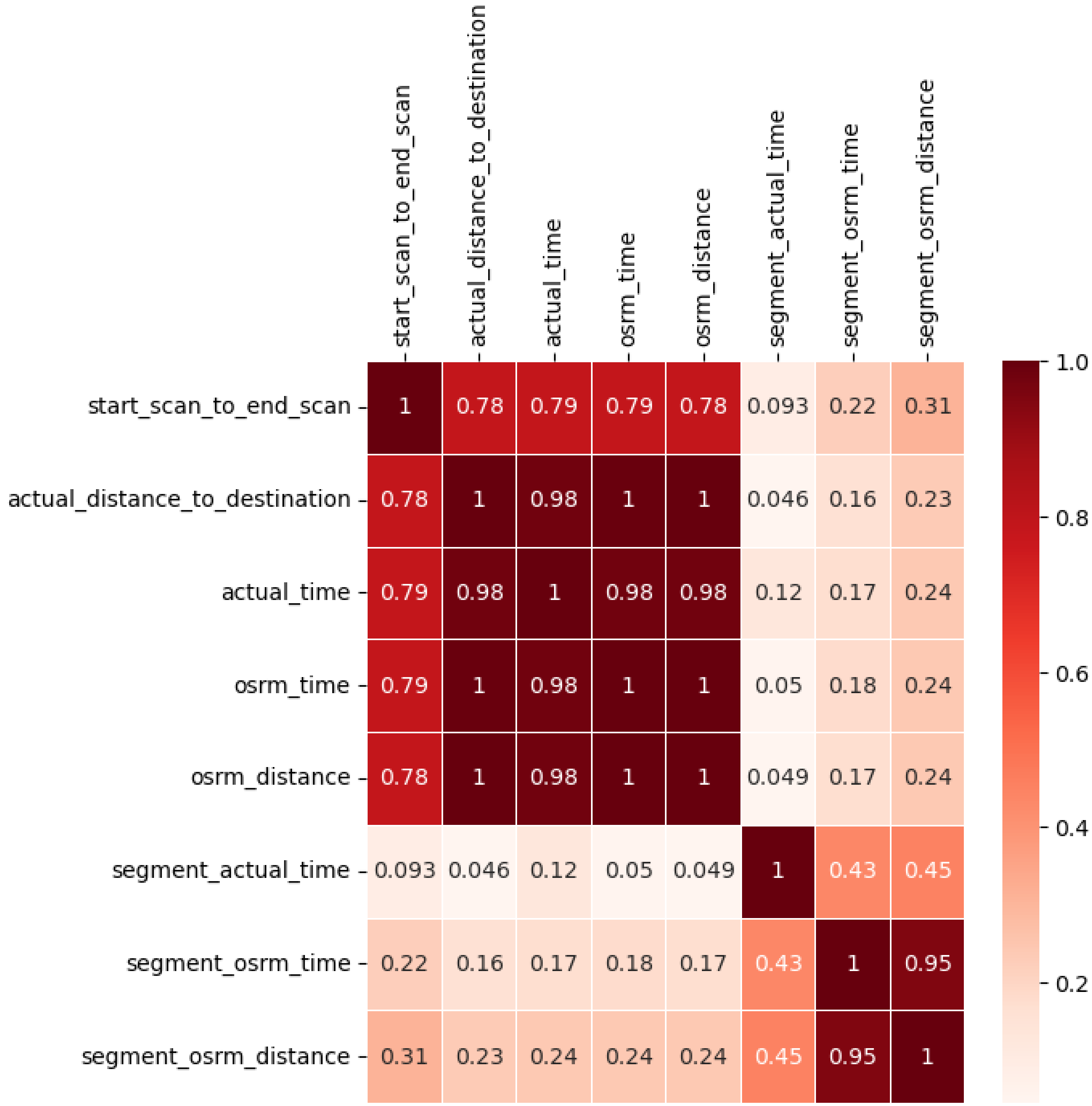


Insight

- The histogram plot of all the **numerical** values show that all the **data is right skewed**
- **72.5%** of the data is **training** data and remaining **27.5%** is **testing** data
- **68.7%** of the delivery is done via **FTL** and remaining **31.3%** through **Carting**

4.3. Multivariate analysis

In [14]:
```
fig, ax = plt.subplots(figsize=(6,6))
sns.heatmap(df.select_dtypes(include=np.number).corr(), annot=True, linewidth=0.5, cmap
ax.xaxis.tick_top()
plt.xticks(rotation=90)
plt.show()
```



Insight

- The heatmap clearly shows high correlation between time and distance. This is expected as the delivery time increases with increase in distance
- *actual_distance_to_destination, actual_time, osrm_time and osrm_distance* are highly correlated
- *segment_osrm_time and segment_osrm_distance* are highly correlated

4.4. Merging rows

The delivery details of one package is divided into several rows. Creating a unique identifier, called *segment_key*, for different segments of a trip based on the combination of *trip_uuid, source_center, and destination_center* and then merge the rows of columns *segment_actual_time, segment_osrm_distance and segment_osrm_time* with same *segment_key* to form new columns *segment_actual_time_sum, segment_osrm_distance_sum, segment_osrm_time_sum*

In [15]:
```
df["segment_key"] = df["trip_uuid"] + '_' + df["source_center"] + '_' + df["destination_
df = df.drop(columns=["source_center", "destination_center"])
segment_columns = ["segment_actual_time", "segment_osrm_distance", "segment_osrm_time"]
for col in segment_columns:
    df[col + "_sum"] = df.groupby("segment_key")[col].cumsum()
```

In [16]:
```
df.head(10)
```

Out[16]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_name	
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Kh
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Kh
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Kh
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Kh
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Anand_VUNagar_DC (Gujarat)	Kh
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	Khambhat_MotvdDPP_D (Gujarat)	

10 rows × 21 columns

Grouping the data by *segment_key*, with aggregation defined for each column, and creating a new dataframe *segment*

In [17]:
```
segment_dict = {
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_name' : 'first',
    'destination_name' : 'last',
    'od_start_time' : 'first',
```

```
    'od_end_time' : 'last',
    'start_scan_to_end_scan' : 'first',
    'actual_distance_to_destination' : 'last',
    'actual_time' : 'last',
    'osrm_time' : 'last',
    'osrm_distance' : 'last',
    'segment_actual_time_sum' : 'last',
    'segment_osrm_distance_sum' : 'last',
    'segment_osrm_time_sum' : 'last',
}

segment_df = df.groupby('segment_key').agg(segment_dict).reset_index()
segment_df = segment_df.sort_values(by=['segment_key', 'od_end_time'], ascending=True).r
```

In [18]:
```
segment_df.head()
```

Out[18]:

	index	segment_key	data	trip_creation_time	route_schedule_uuid	r
0	0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
1	1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
2	2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
3	3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
4	4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	

4.5. Feature Engineering

Extracting features from given data

Extracting time taken between *od_start_time* and *od_end_time*

In [19]:
```
segment_df['od_time_diff_hour'] = (segment_df['od_end_time'] - segment_df['od_start_time
segment_df = segment_df.drop(columns=['od_end_time', 'od_start_time'])
```

In [20]:
```
segment_df.head()
```

Out[20]:

	index	segment_key	data	trip_creation_time	route_schedule_uuid	r
0	0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
1	1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
2	2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	

3	3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...
4	4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...

Extracting city, place, code and state from *source_name* and *destination_name*

In [21]:
```
segment_df['source_state'] = segment_df['source_name'].str.extract(r'\((.*?)\)')
segment_df['source_data'] = segment_df['source_name'].str.extract(r'^(.*?)\(')
segment_df['source_data'] = segment_df['source_data'].str.strip()

segment_df['destination_state'] = segment_df['destination_name'].str.extract(r'\((.*?)\)
segment_df['destination_data'] = segment_df['destination_name'].str.extract(r'^(.*?)\(')
segment_df['destination_data'] =segment_df['destination_data'].str.strip()
```

In [22]:
```
def extract_city_place_code(name):
    parts = name.split('_')
    num_of_parts = len(parts)
    if(num_of_parts == 3):
        city = parts[0]
        place = parts[1]
        code = parts[2]
    elif(num_of_parts == 2):
        city = parts[0]
        place = parts[1]
        code = 'none'
    else:
        city = parts[0]
        place = city
        code = 'none'

    if city == 'Bangalore' or city == 'HBR Layout PC' or city == 'BLR':
        city = 'Bengaluru'
    elif city == 'Mumbai Hub' or city == 'BOM':
        city = 'Mumbai'
    elif city == 'Del':
        city = 'Delhi'
    elif city == 'PNQ Pashan DPC' or city == 'PNQ Vadgaon Sheri DPC':
        city = 'Pune'
    elif city == 'MAA':
        city = 'Chennai'
    elif city == 'FBD':
        city = 'Faridabad'
    elif city == 'CCU':
        city = 'Kolkata'
    elif city == 'AMD':
        city = 'Ahmedabad'
    elif city == 'FBD':
        city = 'Faridabad'
    elif city == 'GGN':
        city = 'Gurgaon'
    elif city == 'GZB':
        city = 'Ghaziabad'

    return [city, place, code]
```

In [23]:
```
extracted_df = segment_df['source_data'].apply(lambda x: extract_city_place_code(x))
segment_df[['source_city','source_place','source_code']] = pd.DataFrame(extracted_df.tol
extracted_df = segment_df['destination_data'].apply(lambda x: extract_city_place_code(x)
```

```
segment_df[['destination_city','destination_place','destination_code']] = pd.DataFrame(e
segment_df = segment_df.drop(columns=['source_name', 'source_data', 'destination_name',
segment_df.head()
```

Out[23]:

	index	segment_key	data	trip_creation_time	route_schedule_uuid	r
0	0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
1	1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
2	2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
3	3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
4	4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	

5 rows × 24 columns

In [24]:

```
segment_df['trip_creation_year'] = segment_df['trip_creation_time'].dt.year
segment_df['trip_creation_month'] = segment_df['trip_creation_time'].dt.month
segment_df['trip_creation_day'] = segment_df['trip_creation_time'].dt.day
segment_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26222 entries, 0 to 26221
Data columns (total 27 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   index                           26222 non-null  int64
 1   segment_key                     26222 non-null  object
 2   data                            26222 non-null  category
 3   trip_creation_time              26222 non-null  datetime64[ns]
 4   route_schedule_uuid             26222 non-null  object
 5   route_type                      26222 non-null  category
 6   trip_uuid                       26222 non-null  object
 7   start_scan_to_end_scan          26222 non-null  float64
 8   actual_distance_to_destination  26222 non-null  float64
 9   actual_time                     26222 non-null  float64
 10  osrm_time                       26222 non-null  float64
 11  osrm_distance                   26222 non-null  float64
 12  segment_actual_time_sum         26222 non-null  float64
 13  segment_osrm_distance_sum       26222 non-null  float64
 14  segment_osrm_time_sum           26222 non-null  float64
 15  od_time_diff_hour               26222 non-null  float64
 16  source_state                    26222 non-null  object
 17  destination_state               26222 non-null  object
 18  source_city                     26222 non-null  object
 19  source_place                    26222 non-null  object
 20  source_code                     26222 non-null  object
 21  destination_city                26222 non-null  object
 22  destination_place               26222 non-null  object
 23  destination_code                26222 non-null  object
 24  trip_creation_year              26222 non-null  int32
 25  trip_creation_month             26222 non-null  int32
 26  trip_creation_day               26222 non-null  int32
```

```
dtypes: category(2), datetime64[ns](1), float64(9), int32(3), int64(1), object(11)
memory usage: 4.8+ MB
```

4.6. In-depth analysis

4.6.1. Grouping and aggregating at trip-level

Group the *segment_df* by *trip_uuid*

In [25]:
```
trip_dict = {
    'segment_key' : 'first',
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'start_scan_to_end_scan' : 'sum',
    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',
    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum',
    'od_time_diff_hour' : 'sum',
    'source_state' : 'first',
    'destination_state' : 'last',
    'source_city' : 'first',
    'source_place' : 'first',
    'source_code' : 'first',
    'destination_city' : 'last',
    'destination_place' : 'last',
    'destination_code' : 'last',
}
trip_df = segment_df.groupby('trip_uuid').agg(trip_dict).reset_index()
```

In [26]:
```
trip_df.head()
```

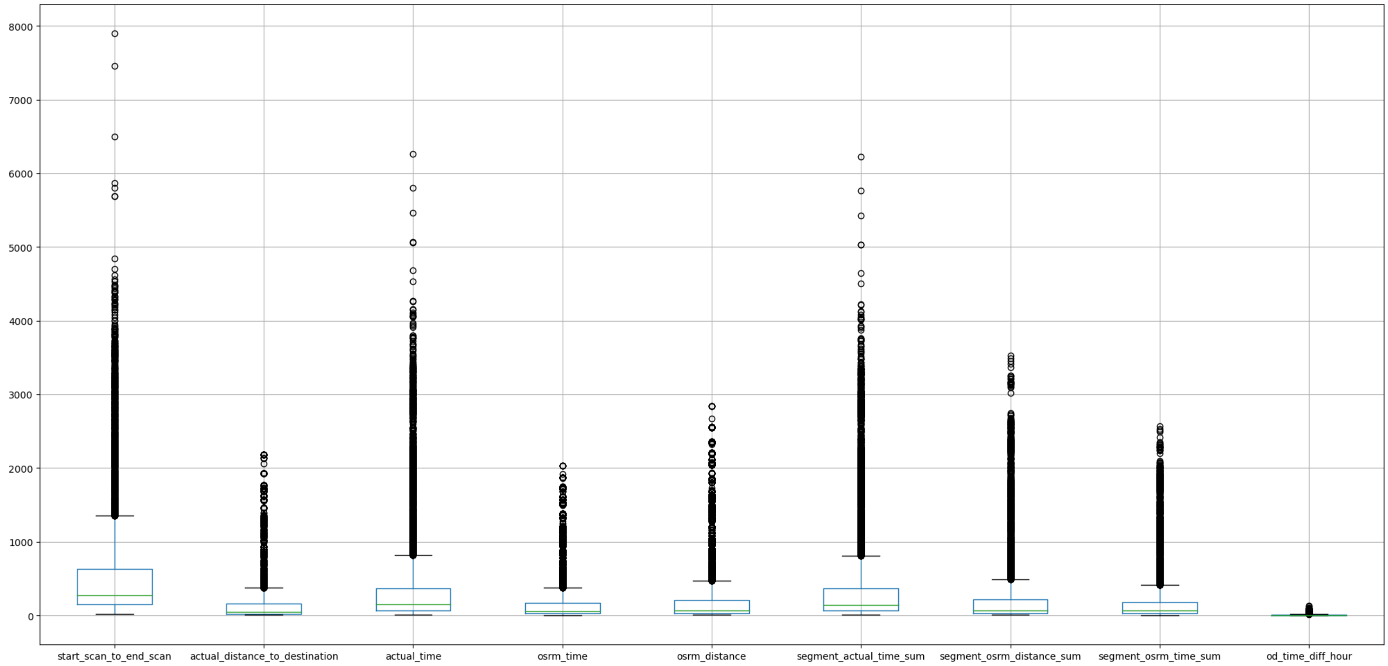
Out[26]:

	trip_uuid	segment_key	data	trip_creation_time	route_scl
0	trip-153671041653548748	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::srout a29b
1	trip-153671042288605164	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::srout bb0b
2	trip-153671043369099517	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::srout 7641
3	trip-153671046011330457	trip-153671046011330457_IND400072AAB_IND401104AAA	training	2018-09-12 00:01:00.113710	thanos::srou a679
4	trip-153671052974046625	trip-153671052974046625_IND583101AAA_IND583201AAA	training	2018-09-12 00:02:09.740725	thanos::srout 65e0

5 rows × 23 columns

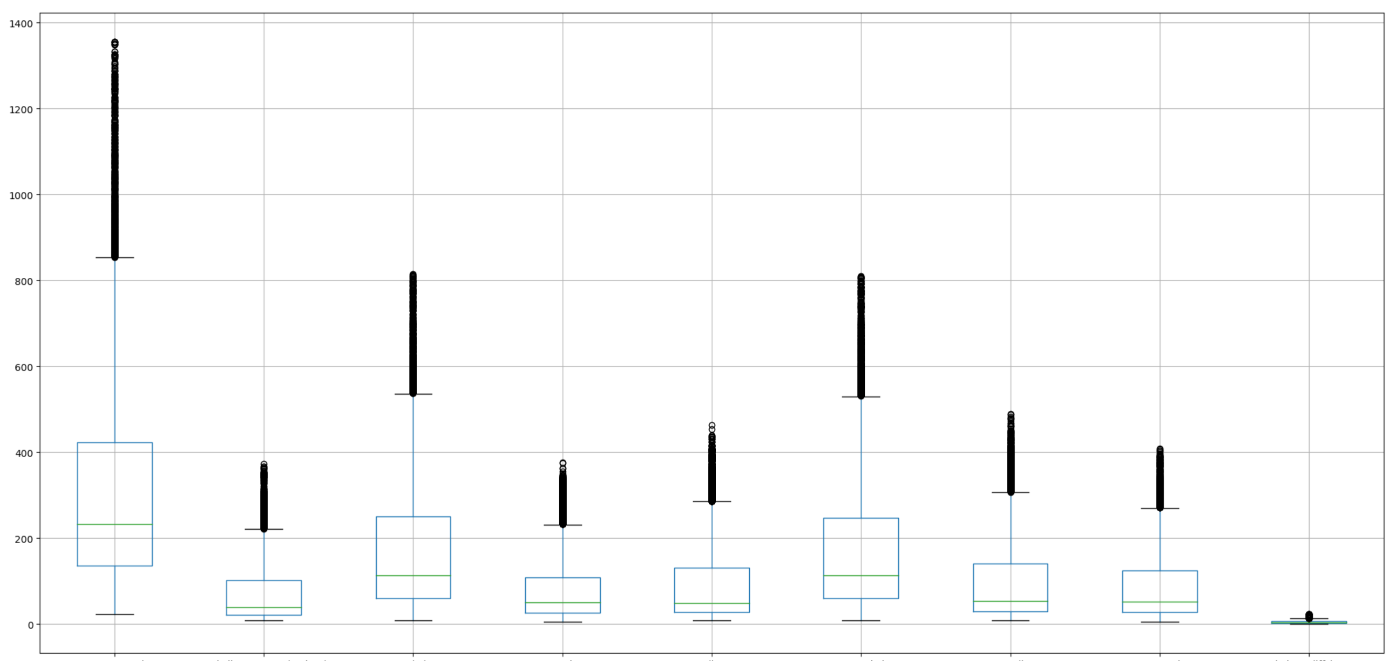
4.6.2. Outlier Detection & Treatment

In [27]:
```
trip_numerical_columns = ['start_scan_to_end_scan', 'actual_distance_to_destination',
                          'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_t
                          'segment_osrm_distance_sum', 'segment_osrm_time_sum', 'od_time
trip_df[trip_numerical_columns].boxplot(figsize=(25,12))
plt.show()
```



In [28]:
```
Q1 = trip_df[trip_numerical_columns].quantile(0.25)
Q3 = trip_df[trip_numerical_columns].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5*IQR
higher_bound = Q3 + 1.5*IQR
trip_df = trip_df[-((trip_df[trip_numerical_columns] < lower_bound) | (trip_df[trip_nume
trip_df = trip_df.reset_index(drop=True)
```

In [29]:
```
trip_df[trip_numerical_columns].boxplot(figsize=(25,12))
plt.show()
```



4.6.3. Perform one-hot encoding on categorical features

route_type is the only categorical feature

In [30]:
```
ohe_df = pd.get_dummies(trip_df['route_type'], dtype='int', prefix='route_type')
trip_df = pd.concat([trip_df, ohe_df], axis=1)
trip_df = trip_df.drop(columns='route_type')
```

In [31]:
```
trip_df.head()
```

Out[31]:

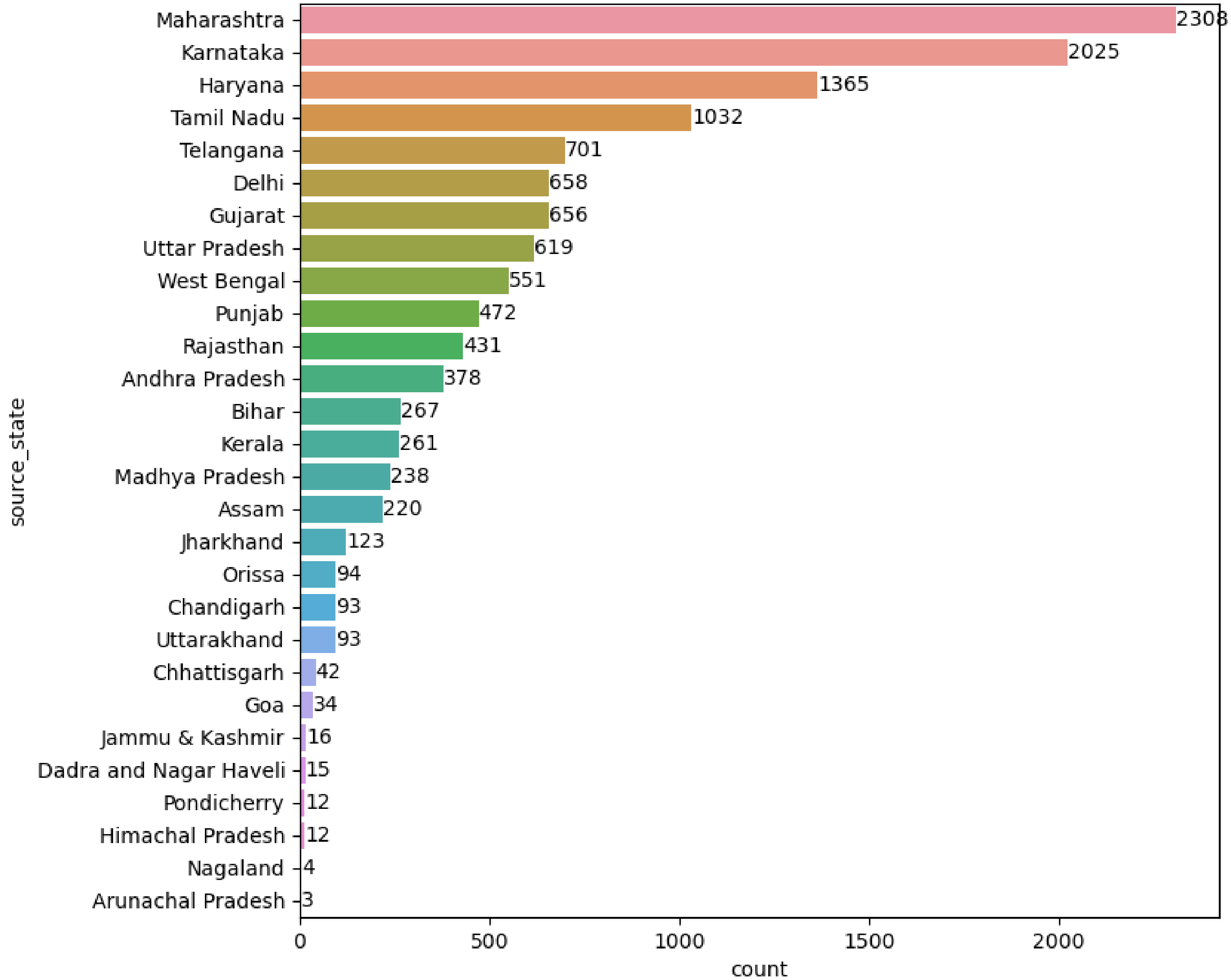
	trip_uuid	segment_key	data	trip_creation_time	route_scl
0	trip-153671042288605164	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::srout bb0k
1	trip-153671046011330457	trip-153671046011330457_IND400072AAB_IND401104AAA	training	2018-09-12 00:01:00.113710	thanos::srou a679
2	trip-153671052974046625	trip-153671052974046625_IND583101AAA_IND583201AAA	training	2018-09-12 00:02:09.740725	thanos::srou 65e(
3	trip-153671055416136166	trip-153671055416136166_IND600056AAA_IND602105AAB	training	2018-09-12 00:02:34.161600	thanos::srou d0a2
4	trip-153671066201138152	trip-153671066201138152_IND600044AAD_IND600048AAA	training	2018-09-12 00:04:22.011653	thanos::srou 846e

5 rows × 24 columns

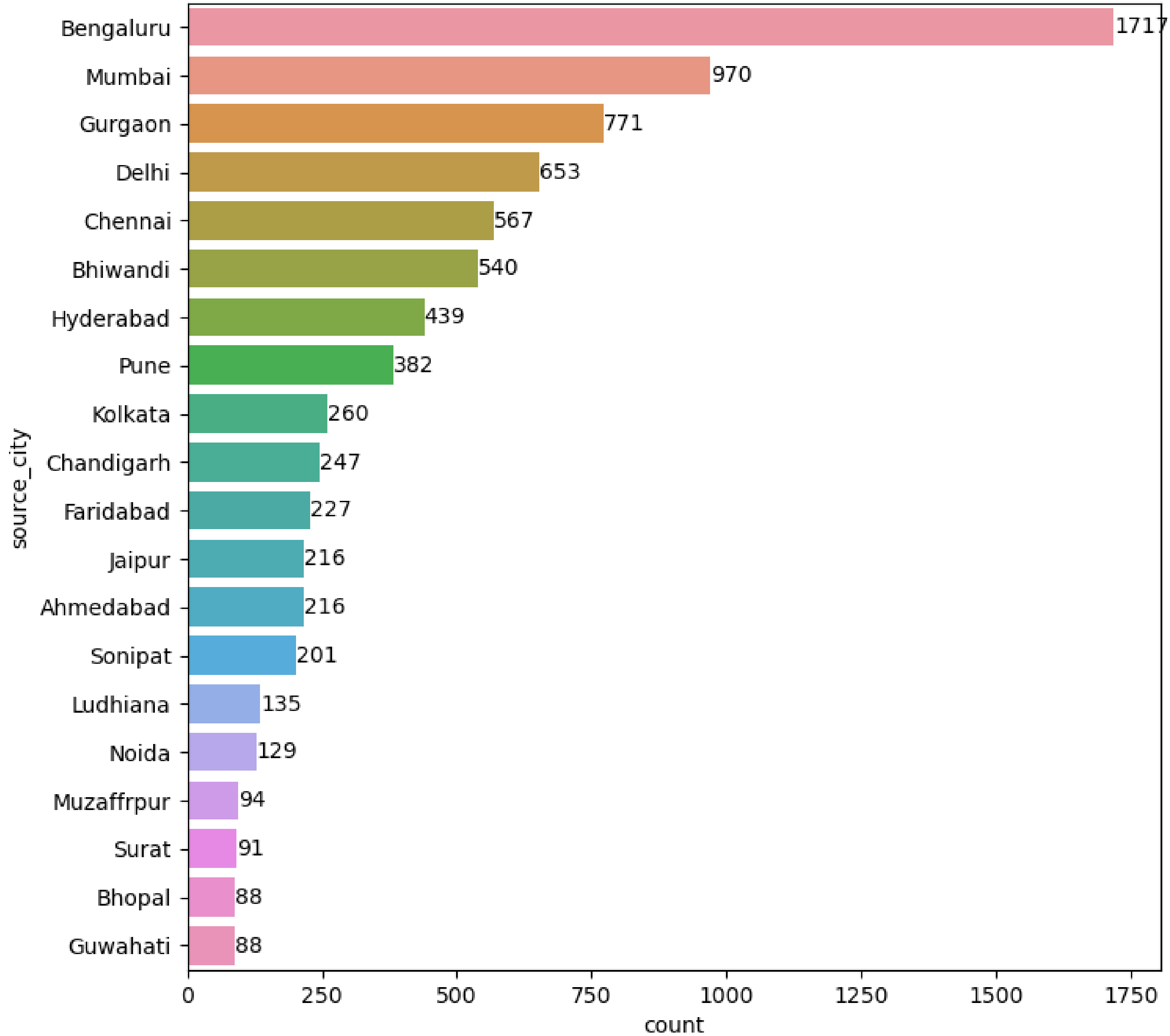
In [32]:
```
print('Number of Carting route is ', trip_df[trip_df['route_type_Carting'] == 1]['route_
print('Number of FTL route is ', trip_df[trip_df['route_type_FTL'] == 1]['route_type_FTL
```

```
Number of Carting route is  8812
Number of FTL route is  3911
```

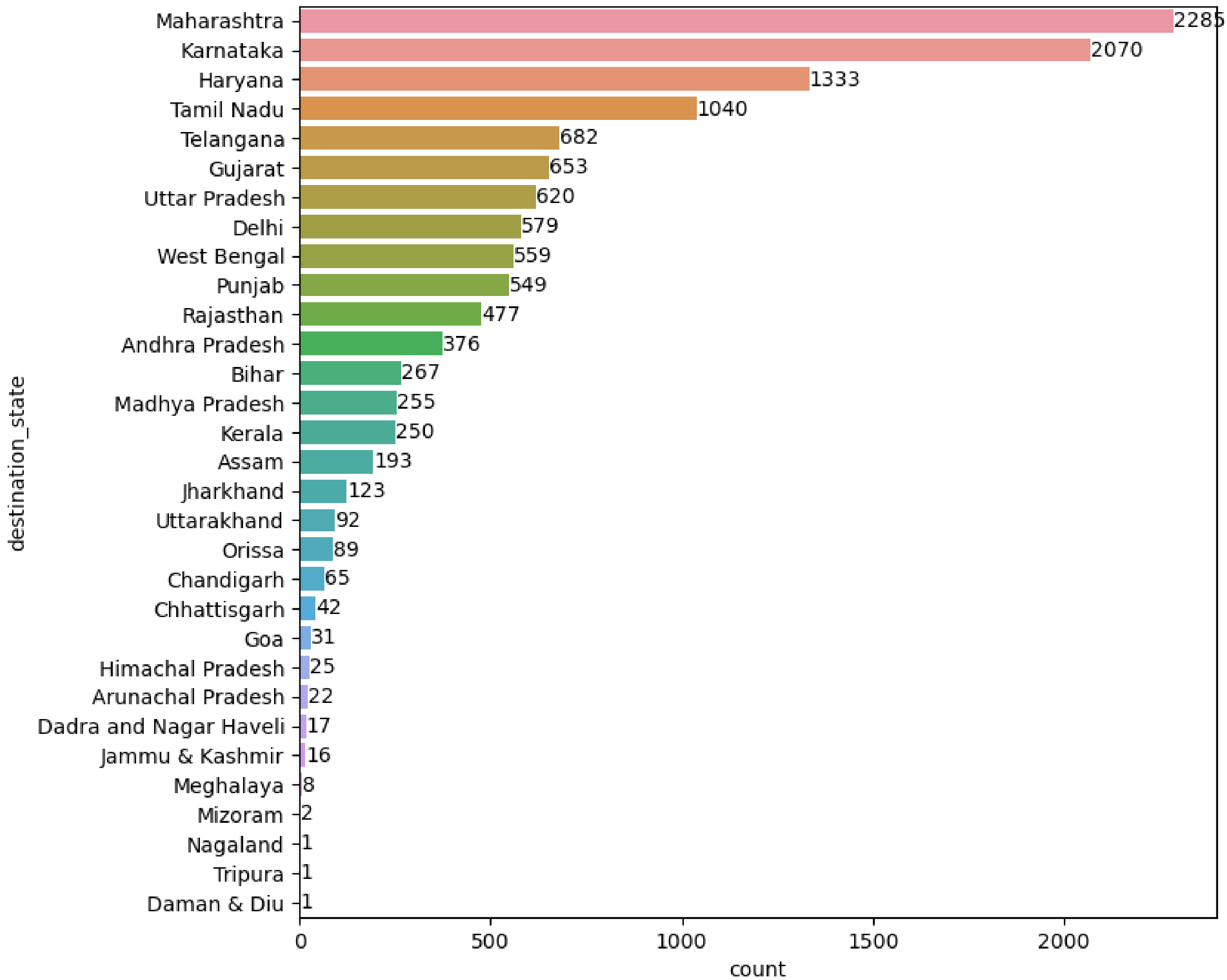
In [33]:
```
plt.figure(figsize=(8,8))
data = trip_df["source_state"]
ax=sns.countplot(y = data, order=data.value_counts().index)
ax.bar_label(ax.containers[0])
plt.show()

plt.figure(figsize=(8,8))
data = trip_df["source_city"]
ax=sns.countplot(y = data, order=data.value_counts()[:20].index)
ax.bar_label(ax.containers[0])
plt.show()

plt.figure(figsize=(8,8))
data = trip_df["destination_state"]
ax=sns.countplot(y = data, order=data.value_counts().index)
ax.bar_label(ax.containers[0])
plt.show()

plt.figure(figsize=(8,8))
data = trip_df["destination_city"]
ax=sns.countplot(y = data, order=data.value_counts()[:20].index)
ax.bar_label(ax.containers[0])
plt.show()
```

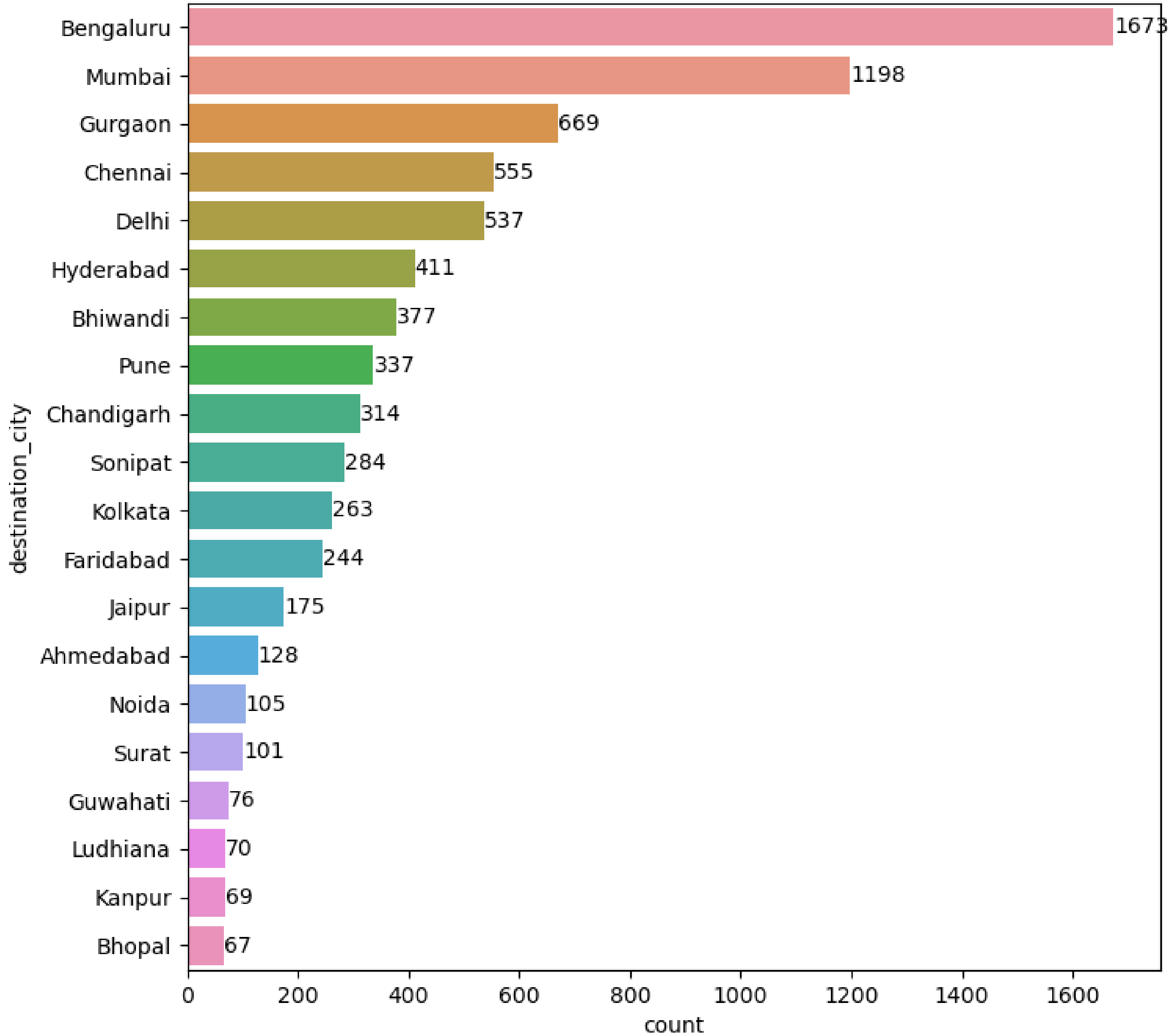


Maharashtra
Karnataka
Haryana
Tamil Nadu
Telangana
Delhi
Gujarat
Uttar Pradesh
West Bengal
Punjab
Rajasthan
Andhra Pradesh
Bihar
Kerala
Madhya Pradesh
Assam
Jharkhand
Orissa
Chandigarh
Uttarakhand
Chhattisgarh
Goa
Jammu & Kashmir
Dadra and Nagar Haveli
Pondicherry
Himachal Pradesh
Nagaland
Arunachal Pradesh
2308
2025
1365
1032
701
658
656
619
551
472
431
378
267
261
238
220
123
94
93
93
42
34
16
15
12
12
4
3
source_state
count
0
500
1000
1500
2000



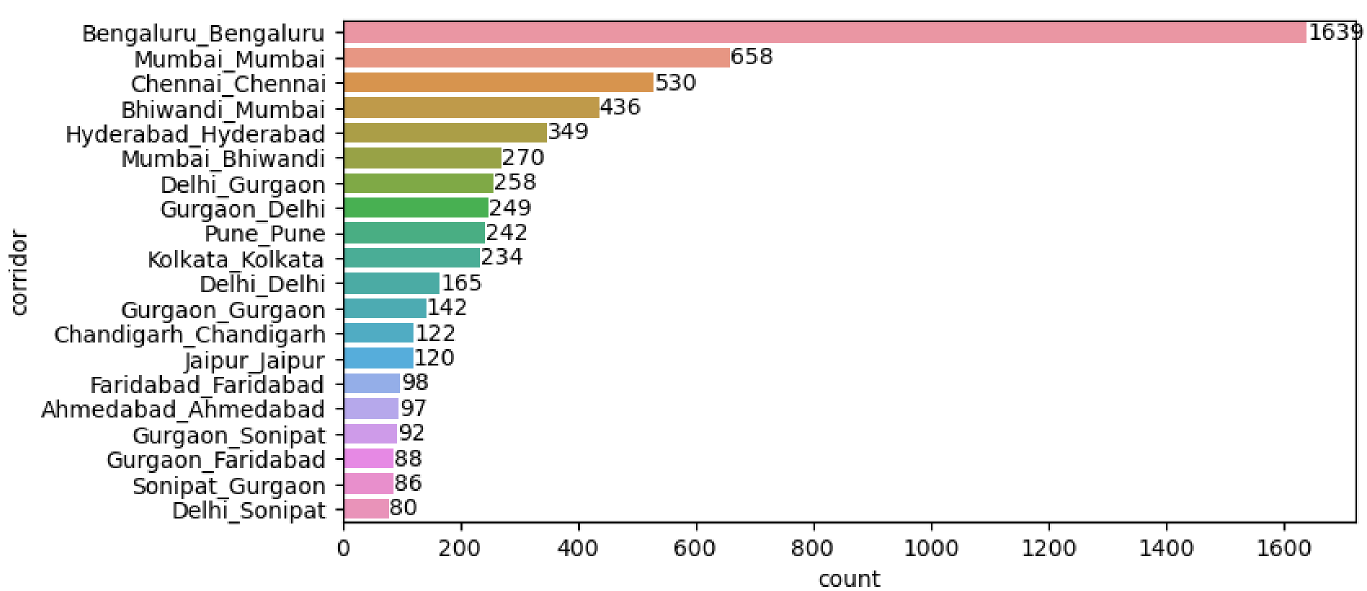
Bengaluru
Mumbai
Gurgaon
Delhi
Chennai
Bhiwandi
Hyderabad
Pune
Kolkata
Chandigarh
Faridabad
Jaipur
Ahmedabad
Sonipat
Ludhiana
Noida
Muzaffrpur
Surat
Bhopal
Guwahati
1717
970
771
653
567
540
439
382
260
247
227
216
216
201
135
129
94
91
88
88
source_city
0
250
500
750
1000
1250
1500
1750
count


destination_state
count
Maharashtra 2285
Karnataka 2070
Haryana 1333
Tamil Nadu 1040
Telangana 682
Gujarat 653
Uttar Pradesh 620
Delhi 579
West Bengal 559
Punjab 549
Rajasthan 477
Andhra Pradesh 376
Bihar 267
Madhya Pradesh 255
Kerala 250
Assam 193
Jharkhand 123
Uttarakhand 92
Orissa 89
Chandigarh 65
Chhattisgarh 42
Goa 31
Himachal Pradesh 25
Arunachal Pradesh 22
Dadra and Nagar Haveli 17
Jammu & Kashmir 16
Meghalaya 8
Mizoram 2
Nagaland 1
Tripura 1
Daman & Diu 1
0
500
1000
1500
2000



In [34]:

```
trip_df["corridor"] = trip_df["source_city"] + '_' + trip_df["destination_city"]
plt.figure(figsize=(8,4))
ax=sns.countplot(y = trip_df["corridor"], order=trip_df["corridor"].value_counts()[:20].
ax.bar_label(ax.containers[0])
plt.show()
```

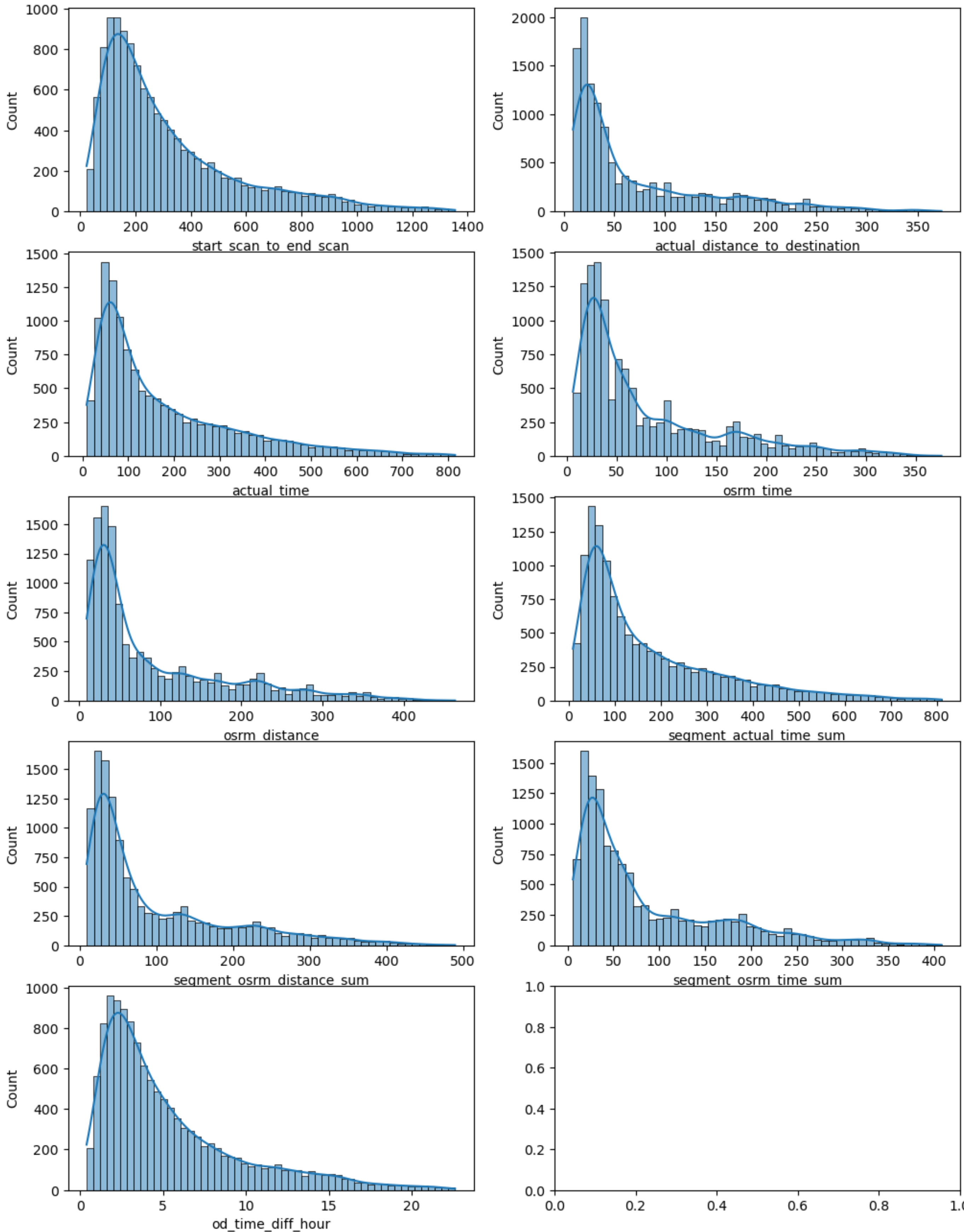


In [35]:
```
Mumbai_Bhiwandi_df = trip_df[((trip_df["corridor"] == "Bhiwandi_Mumbai") | (trip_df["cor
print('Avg time: ', Mumbai_Bhiwandi_df['actual_time'].mean())
print('Avg distance: ', Mumbai_Bhiwandi_df['actual_distance_to_destination'].mean())
```

```
Avg time:  81.8186968838527
Avg distance:  22.218624868058914
```

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

In [36]:
```
fig, ax = plt.subplots(nrows=5, ncols=2, figsize = (12, 16))
sns.histplot(data=trip_df, x = "start_scan_to_end_scan", kde=True, ax=ax[0,0])
sns.histplot(data=trip_df, x = "actual_distance_to_destination", kde=True, ax=ax[0,1])
sns.histplot(data=trip_df, x = "actual_time", kde=True, ax=ax[1,0])
sns.histplot(data=trip_df, x = "osrm_time", kde=True, ax=ax[1,1])
sns.histplot(data=trip_df, x = "osrm_distance", kde=True, ax=ax[2,0])
sns.histplot(data=trip_df, x = "segment_actual_time_sum", kde=True, ax=ax[2,1])
sns.histplot(data=trip_df, x = "segment_osrm_distance_sum", kde=True, ax=ax[3,0])
sns.histplot(data=trip_df, x = "segment_osrm_time_sum", kde=True, ax=ax[3,1])
sns.histplot(data=trip_df, x = "od_time_diff_hour", kde=True, ax=ax[4,0])
plt.show()
```



Insight

- None of the data is gaussian, so we will use MinMaxScaler

In [37]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaler.fit(trip_df[trip_numerical_columns])
trip_df[trip_numerical_columns] = scaler.transform(trip_df[trip_numerical_columns])
```

In [38]:
```
trip_df.describe()
```

Out[38]:

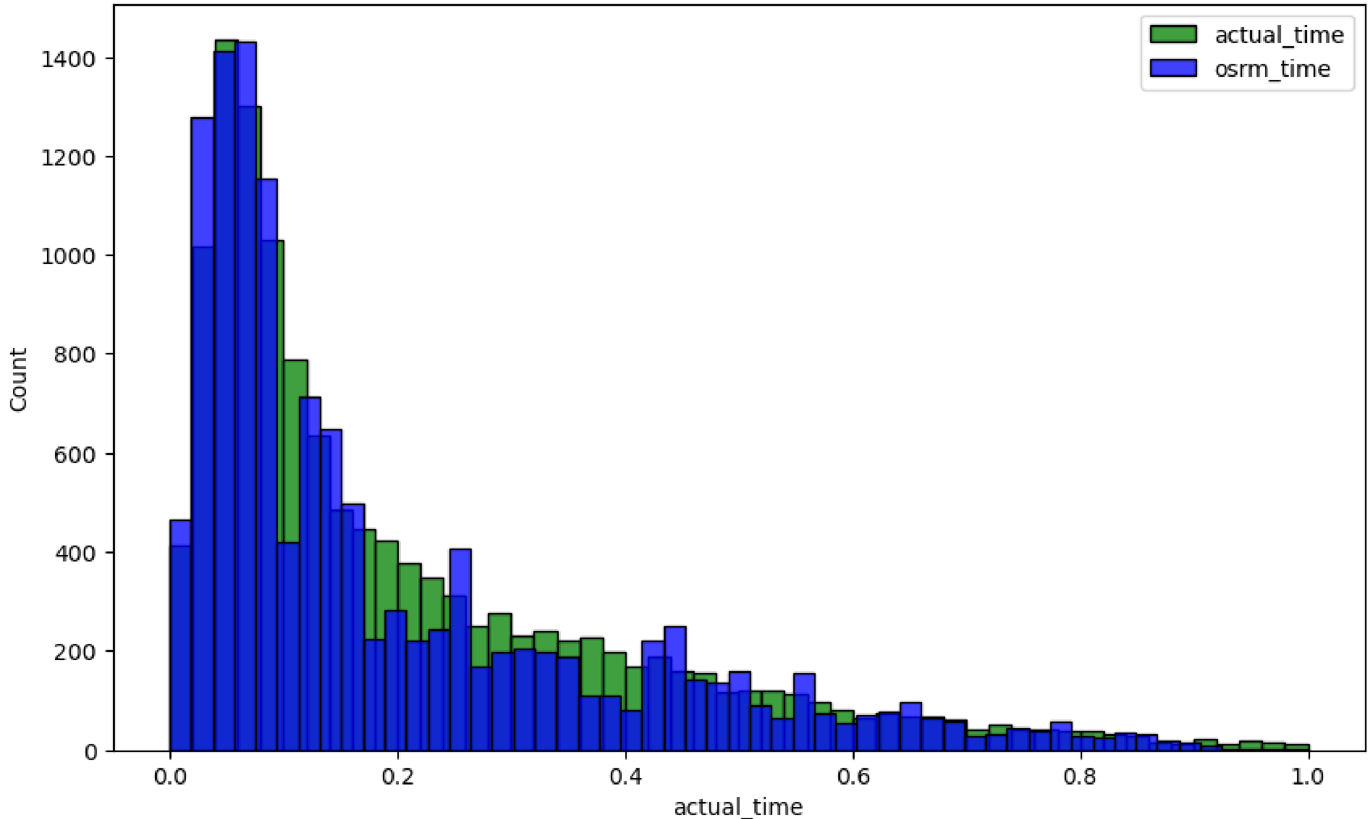
	trip_creation_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm
count	12723	12723.000000	12723.000000	12723.000000	12723.000000	1272
mean	2018-09-22 13:16:08.771620608	0.223107	0.173734	0.208998	0.195785	
min	2018-09-12 00:00:22.886430	0.000000	0.000000	0.000000	0.000000	
25%	2018-09-17 03:12:27.545116928	0.084835	0.034006	0.064516	0.056757	
50%	2018-09-22 04:23:52.568071936	0.157658	0.081009	0.130273	0.118919	
75%	2018-09-27 20:46:53.577142016	0.300300	0.254284	0.300248	0.278378	
max	2018-10-03 23:59:42.701692	1.000000	1.000000	1.000000	1.000000	
std	NaN	0.191859	0.197757	0.196217	0.195496	

4.7. Hypothesis Testing

4.7.1. Are aggregated *actual_time* and aggregated *osrm_time* similar?

H0 : *actual_time* and *osrm_time* are similar \ H1 : *actual_time* and *osrm_time* are different

In [39]:
```
plt.figure(figsize=(10,6))
sns.histplot(trip_df['actual_time'], color='green')
sns.histplot(trip_df['osrm_time'], color='blue')
plt.legend(['actual_time', 'osrm_time'])
plt.show()
```



This is a 2 sample continuous skewed data, so we will use Mann-Whitney U Test

In [40]:
```
statistic, pvalue = sps.mannwhitneyu(trip_df['actual_time'], trip_df['osrm_time'], alter
print('p-value', pvalue)
if pvalue < 0.1:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 1.3094485692382313e-20
The samples are not similar
```

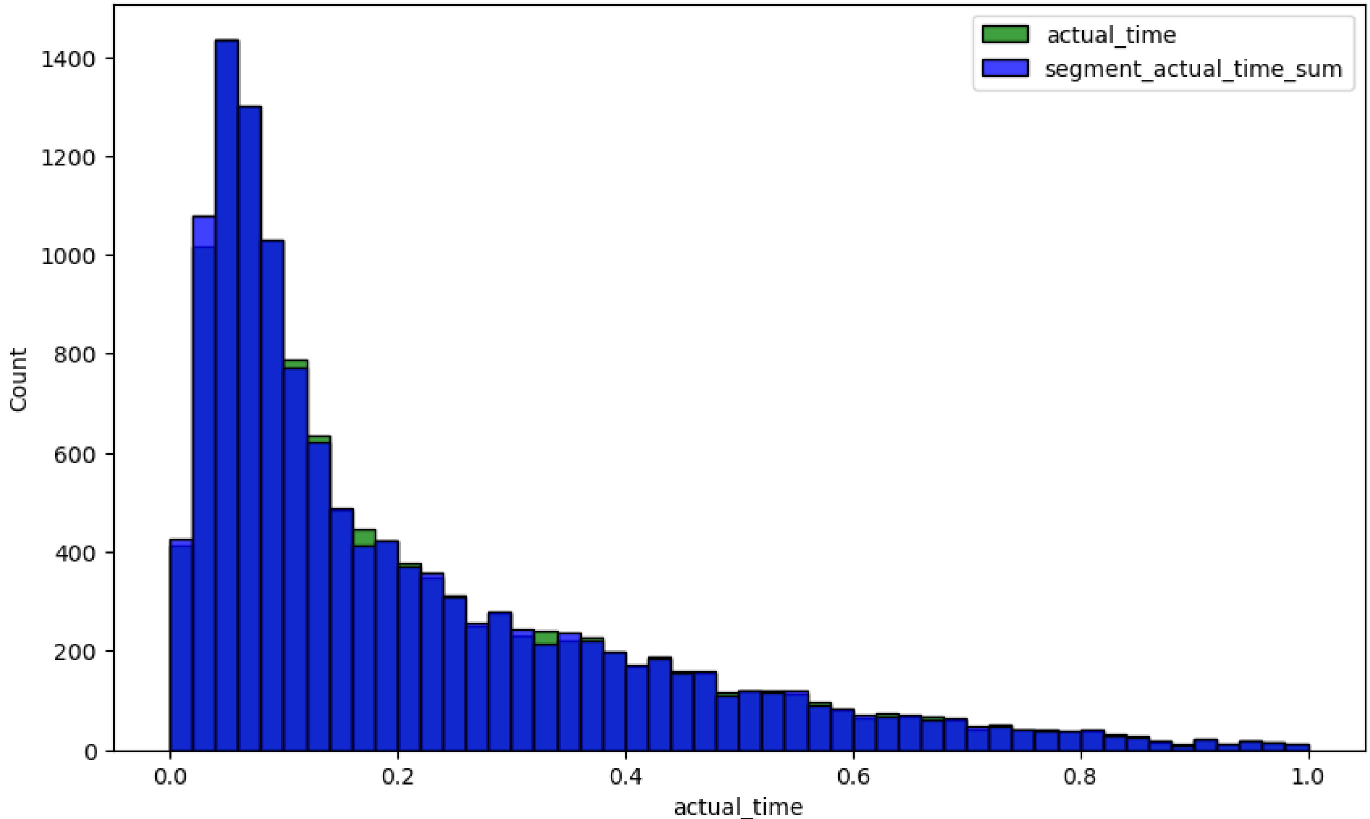
Insight

- *actual_time* and *osrm_time* are different

4.7.2. Are aggregated *actual_time* and aggregated *segment_actual_time* similar?

H0 : *actual_time* and *segment_actual_time* are similar \ H1 : *actual_time* and *segment_actual_time* are different

In [41]:
```
plt.figure(figsize=(10,6))
sns.histplot(trip_df['actual_time'], color='green')
sns.histplot(trip_df['segment_actual_time_sum'], color='blue')
plt.legend(['actual_time', 'segment_actual_time_sum'])
plt.show()
```



This is a 2 sample continuous skewed data, so we will use Mann-Whitney U Test

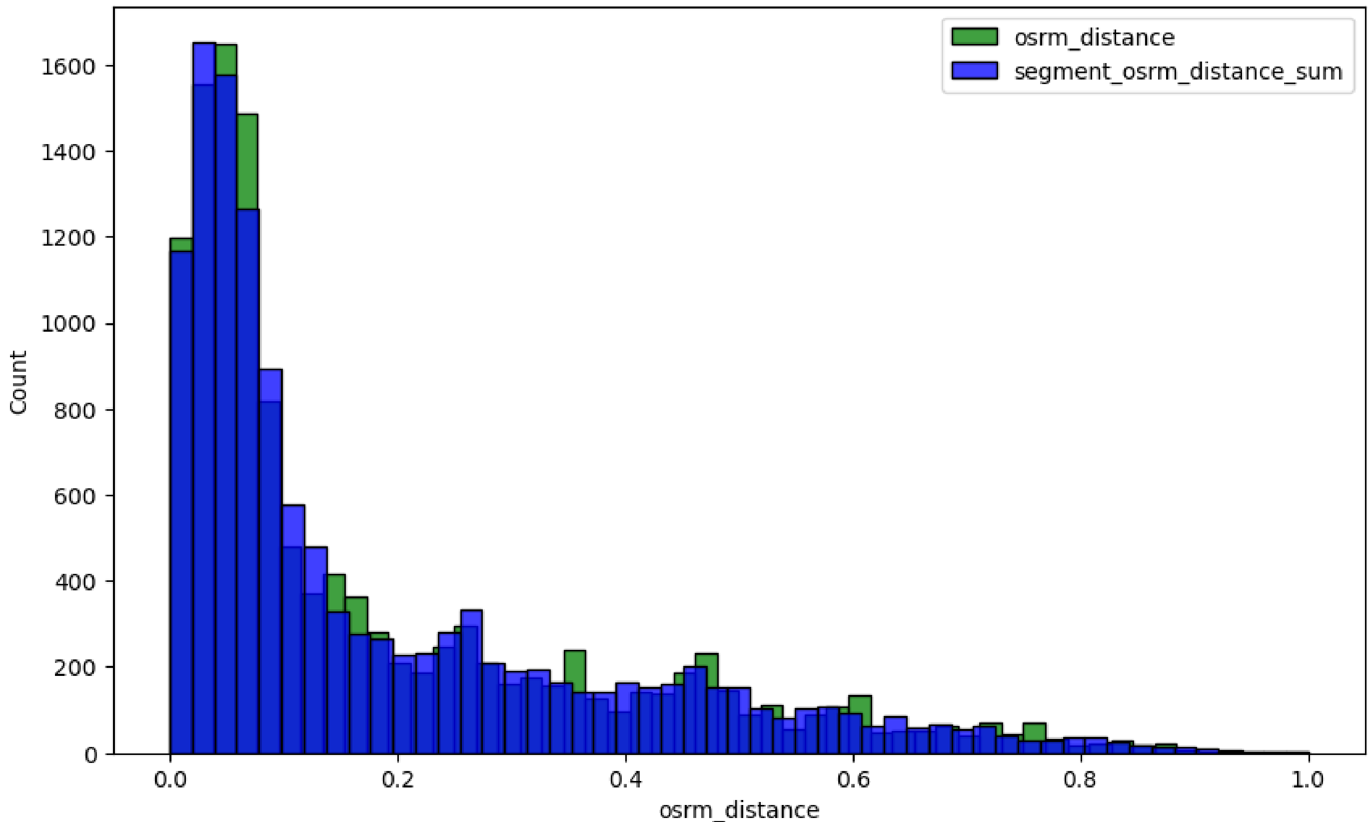
In [42]:
```
statistic, pvalue = sps.mannwhitneyu(trip_df['actual_time'], trip_df['segment_actual_tim
print('p-value', pvalue)
if pvalue < 0.1:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 0.7167057478572094
The samples are similar
```

4.7.3. Are aggregated *osrm_distance* and aggregated *segment_osrm_distance* similar?

H0 : *osrm_distance* and *segment_osrm_distance* are similar \ H1 : *osrm_distance* and *segment_osrm_distance* are different

In [43]:
```
plt.figure(figsize=(10,6))
sns.histplot(trip_df['osrm_distance'], color='green')
sns.histplot(trip_df['segment_osrm_distance_sum'], color='blue')
plt.legend(['osrm_distance', 'segment_osrm_distance_sum'])
plt.show()
```



This is a 2 sample continuous skewed data, so we will use Mann-Whitney U Test

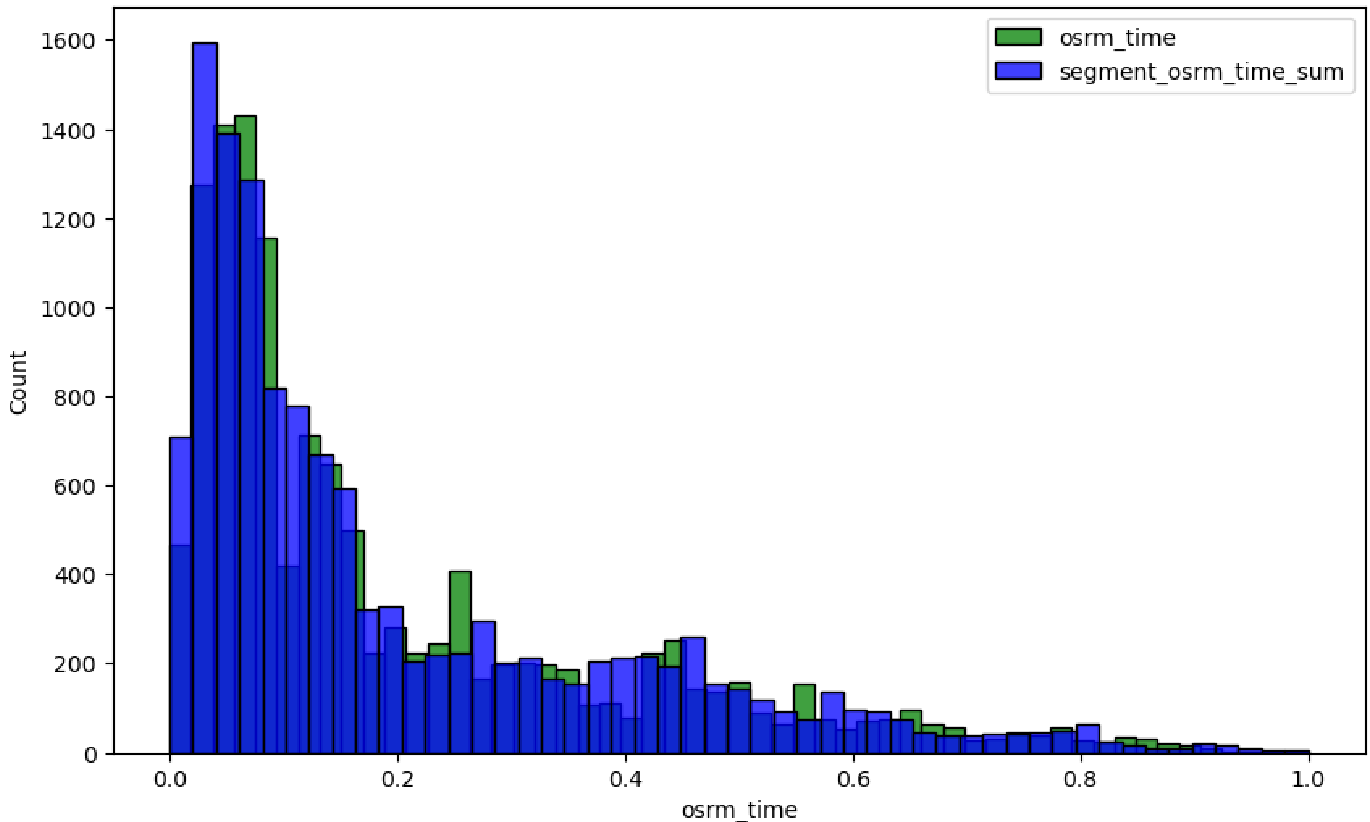
In [44]:
```
statistic, pvalue = sps.mannwhitneyu(trip_df['osrm_distance'], trip_df['segment_osrm_dis
print('p-value', pvalue)
if pvalue < 0.1:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 0.05751040543671224
The samples are not similar
```

4.7.4. Are aggregated *osrm_time* and aggregated *segment_osrm_time* similar?

H0 : *osrm_time* and *segment_osrm_time* are similar \ H1 : *osrm_time* and *segment_osrm_time* are different

In [45]:
```
plt.figure(figsize=(10,6))
sns.histplot(trip_df['osrm_time'], color='green')
sns.histplot(trip_df['segment_osrm_time_sum'], color='blue')
plt.legend(['osrm_time', 'segment_osrm_time_sum'])
plt.show()
```



This is a 2 sample continuous skewed data, so we will use Mann-Whitney U Test

In [46]:
```
statistic, pvalue = sps.mannwhitneyu(trip_df['osrm_time'], trip_df['segment_osrm_time_su
print('p-value', pvalue)
if pvalue < 0.1:
    print('The samples are not similar')
else:
    print('The samples are similar ')
```

```
p-value 0.8230933178296898
The samples are similar
```

5. Business Insights

- The most common route type is **Carting**
- The top 3 **source states** are **Maharastra, Karnataka and Haryana**
- The top 3 **source cities** are **Bengaluru, Mumbai and Gurgaon**
- The top 3 **destination states** are **Maharastra, Karnataka and Haryana**
- The top 3 **destination cities** are **Bengaluru, Mumbai and Gurgaon**
- Most of the packages are sent and received within Bengaluru, Mumbai and Chennai but the **most bussiest corridor is Bhiwandi-Mumbai**
- Aggregated **actual_time** and aggregated **osrm_time** are **not similar**
- Aggregated **actual_time** and aggregated **segment_actual_time** are **similar**
- Aggregated **osrm_distance** and aggregated **segment_osrm_distance** are **not similar**
- Aggregated **osrm_time** and aggregated **segment_osrm_time** are **similar**

6. Recommendation

- The company should advertise more on route type FTL saying it is faster mode of delivery. This way FTL can be suggested to alteast large organization.
- Cities Bengaluru(Karnataka), Mumbai(Maharastra) and Gurgaon(Haryana) send and recieve the majority of the deliveries. The company should keep the customers of these cities satisfied with the better and faster services. This involves improving the OSRM engine to make better delivery time predictions.

In [ ]:

```

```