About this dataset

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

Business Problem

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. Importing libraries

In [84]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
from scipy.stats import norm,ttest_1samp,ttest_ind,ttest_rel,chisquare,chi2_contingency
from scipy.stats import f_oneway,pearsonr,spearmanr,shapiro,kstest
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from category_encoders.target_encoder import TargetEncoder
from sklearn.preprocessing import StandardScaler
from skimpy import skim
from sklearn.preprocessing import OneHotEncoder
from scipy.sparse import hstack
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

2. Data Overview

In [45]:
```
df = pd.read_csv('delhivery_data.txt')
df.head()
```

Out[45]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	desti
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_

5 rows × 24 columns

3. EDA

In [8]:
```
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 [9]:
```
df.columns
```

Out[9]:
```
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')
```

In [10]:
```
df.shape
```

Out[10]:
```
(144867, 24)
```

In [11]:
```
# Lenght of data given
print('Length of Data: ', df.shape[0])
print('Length of Column: ', df.shape[1])
```

```
Length of Data:  144867
Length of Column:  24
```

In [12]:
```
df.describe()
```

Out[12]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor	segment_actual_
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.00
mean	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	2.120107	36.19
std	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	1.715421	53.57
min	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	0.144000	-244.00
25%	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	1.604264	20.00
50%	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	1.857143	29.00
75%	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	2.213483	40.00
max	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	77.387097	3051.00

In [13]:
```
df.nunique()
```

Out[13]:
```
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
```

In [14]:
```
round(100*(df.isnull().sum()/len(df.index)),2).sort_values(ascending=False)
```

Out[14]:
```
source_name                      0.20
destination_name                 0.18
data                             0.00
cutoff_factor                    0.00
segment_osrm_distance            0.00
segment_osrm_time                0.00
segment_actual_time              0.00
factor                           0.00
osrm_distance                    0.00
osrm_time                        0.00
actual_time                      0.00
actual_distance_to_destination   0.00
cutoff_timestamp                 0.00
is_cutoff                        0.00
trip_creation_time               0.00
start_scan_to_end_scan           0.00
od_end_time                      0.00
od_start_time                    0.00
destination_center               0.00
source_center                    0.00
trip_uuid                        0.00
route_type                       0.00
route_schedule_uuid              0.00
segment_factor                   0.00
dtype: float64
```

Data is very good. we have only .20% and .18% data is missing in Source_name and destination_name.

In [15]:
```
skim(df)
```

skimpy summary

Data Summary

dataframe	Values
Number of rows	144867
Number of columns	24

Data Types

Column Type	Count
string	12
float64	10
bool	1
int32	1

number

column_name	NA	NA %	mean	sd	p0	p25	p50	p75	p100	hist
start_scan_to_end_scan	0	0	960	1000	20	160	450	1600	7900	
cutoff_factor	0	0	230	340	9	22	66	290	1900	
actual_distance_to_destination	0	0	230	340	9	23	66	290	1900	
actual_time	0	0	420	600	9	51	130	510	4500	
osrm_time	0	0	210	310	6	27	64	260	1700	
osrm_distance	0	0	280	420	9	30	79	340	2300	
factor	0	0	2.1	1.7	0.14	1.6	1.9	2.2	77	
segment_actual_time	0	0	36	54	-240	20	29	40	3100	
segment_osrm_time	0	0	19	15	0	11	17	22	1600	
segment_osrm_distance	0	0	23	18	0	12	24	28	2200	
segment_factor	0	0	2.2	4.8	-23	1.3	1.7	2.2	570	

bool

column_name	true	true rate	hist
is_cutoff	118749	0.82	

string

column_name	NA	NA %	words per row	total words
data	0	0	1	144867
trip_creation_time	0	0	2	289734
route_schedule_uuid	0	0	1	144867
route_type	0	0	1	144867
trip_uuid	0	0	1	144867
source_center	0	0	1	144867
source_name	293	0.2	2.2	322826
destination_center	0	0	1	144867
destination_name	261	0.18	2.3	328228
od_start_time	0	0	2	289734
od_end_time	0	0	2	289734
cutoff_timestamp	0	0	2	289734

End

In [16]:
```
df.dtypes
```

Out[16]:
```
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
```

Checking for duplicate records:

In [17]:
```
df.duplicated().value_counts()
```

Out[17]:
```
False    144867
Name: count, dtype: int64
```

There are no duplicate rows in this dataset.

Handling missing values

In [46]:
```
simple_imputer = SimpleImputer(strategy='most_frequent')
missing_col = ['source_name','destination_name']
for col in missing_col:
    df[col] = pd.DataFrame(simple_imputer.fit_transform(pd.DataFrame(df[col])))
```

In [20]:
```
df.isna().sum()
```

Out[20]:
```
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
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
```

Coverting columns with 'object' data type to 'datetime' data type.

In [47]:
```
# Convert other datetime columns to datetime objects
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['cutoff_timestamp'] = pd.to_datetime(df['cutoff_timestamp'], format="%Y-%m-%d %H:%M:%S.%f", errors='coerce')

# Parse 'trip_creation_time' without using pd.to_datetime
def parse_trip_creation_time(value):
    try:
        return pd.Timestamp(value)
    except (ValueError, TypeError):
        return pd.NaT

df['trip_creation_time'] = df['trip_creation_time'].apply(parse_trip_creation_time)
df['cutoff_timestamp'] = df['cutoff_timestamp'].apply(parse_trip_creation_time)

# Floor 'trip_creation_time' to the nearest minute
df['trip_creation_time'] = df['trip_creation_time'].dt.floor('T')
df['cutoff_timestamp'] = df['cutoff_timestamp'].dt.floor('T')
# Format datetime columns as strings in the desired format
df['trip_creation_time'] = df['trip_creation_time'].dt.strftime("%m/%d/%Y %H:%M")
df['od_start_time'] = df['od_start_time'].dt.strftime("%m/%d/%Y %H:%M")
df['od_end_time'] = df['od_end_time'].dt.strftime("%m/%d/%Y %H:%M")
df['cutoff_timestamp'] = df['cutoff_timestamp'].dt.strftime("%m/%d/%Y %H:%M")
```

In [48]:
```
Date_col = ['trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_timestamp']

for col in Date_col:
    df[col] = pd.to_datetime(df[col], format='%m/%d/%Y %H:%M')
```

In [49]:
```
df['trip_creation_date'] = df['trip_creation_time'].dt.date
df['od_start_date'] = df['od_start_time'].dt.date
df['od_end_date'] = df['od_end_time'].dt.date
df['cutoff_date'] = df['cutoff_timestamp'].dt.date

df['od_start_date_hour'] = df['od_start_time'].dt.hour



df['trip_year'] = df['trip_creation_time'].dt.year
df['trip_month'] = df['trip_creation_time'].dt.month
df['trip_hour'] = df['trip_creation_time'].dt.hour
df['trip_day'] = df['trip_creation_time'].dt.day
df['trip_week'] = df['trip_creation_time'].dt.isocalendar().week
df['trip_dayofweek'] = df['trip_creation_time'].dt.dayofweek
df['time_category'] = pd.cut(df['trip_hour'],
                             bins=[-1, 4, 12, 16, 24],
```

```
                                labels=['Night', 'Morning', 'Noon', 'Evening'],
                                include_lowest=True)
```

Extracting City name, SubUrb name Division and State from Source and Destination Names

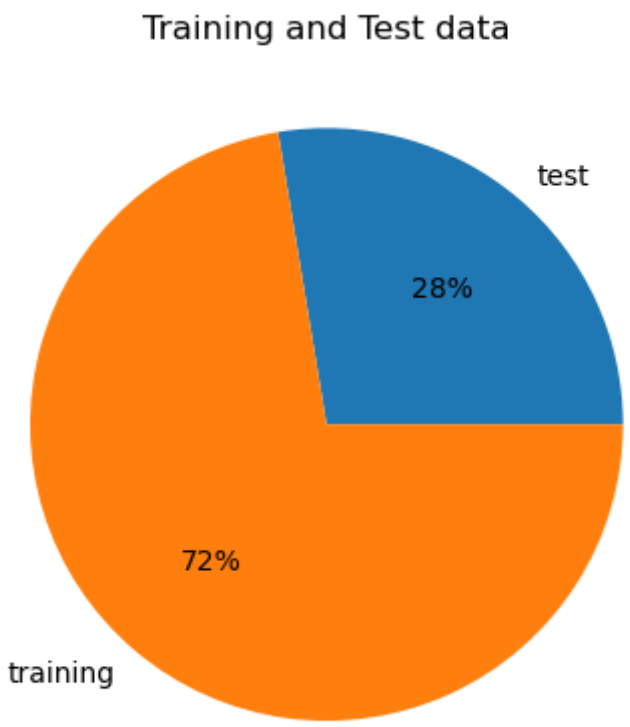
In [50]:
```
df[['City','SubUrbs','Division']] = df['source_name'].str.split('_', expand=True, n=2)
#df.drop('source_name',axis=1,inplace=True)

df['State'] = df['Division'].str.extract(r'\((.*?)\)')

df[['D_City','D_SubUrbs','D_Division']] = df['destination_name'].str.split('_', expand=True, n=2)
#df.drop('destination_name',axis=1,inplace=True)

df['D_State'] = df['D_Division'].str.extract(r'\((.*?)\)')
```

Analyze structure

In [52]:
```
# Type of Data present in dataset
df['data'].value_counts()
```

Out[52]:
```
data
training    104858
test         40009
Name: count, dtype: int64
```

In [75]:
```
data_agg = df.groupby(['data']).aggregate(count=('data','count')).reset_index()
plt.pie(data_agg['count'], labels=data_agg['data'],autopct='%.0f%%')
plt.title('Training and Test data')
plt.show()
```

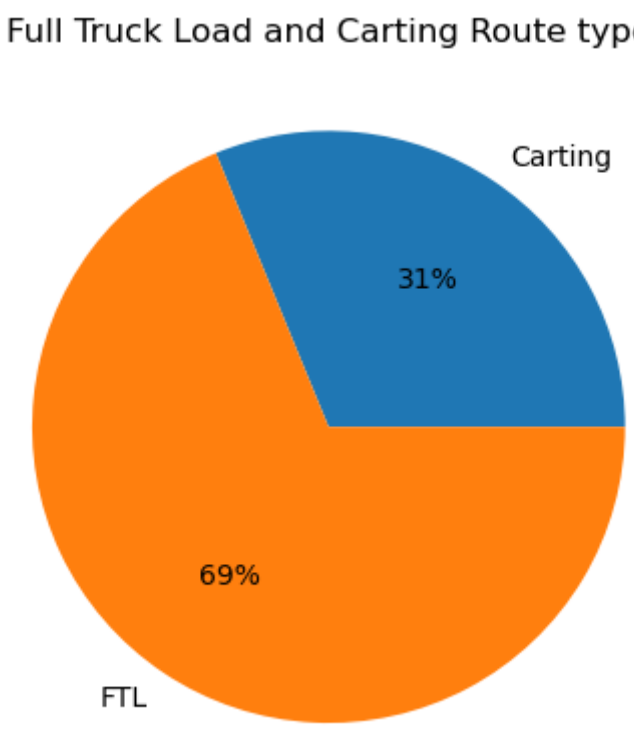


In [76]:
```
route_type = df.groupby(['route_type']).aggregate(count=('route_type','count')).reset_index()
route_type
```

Out[76]:

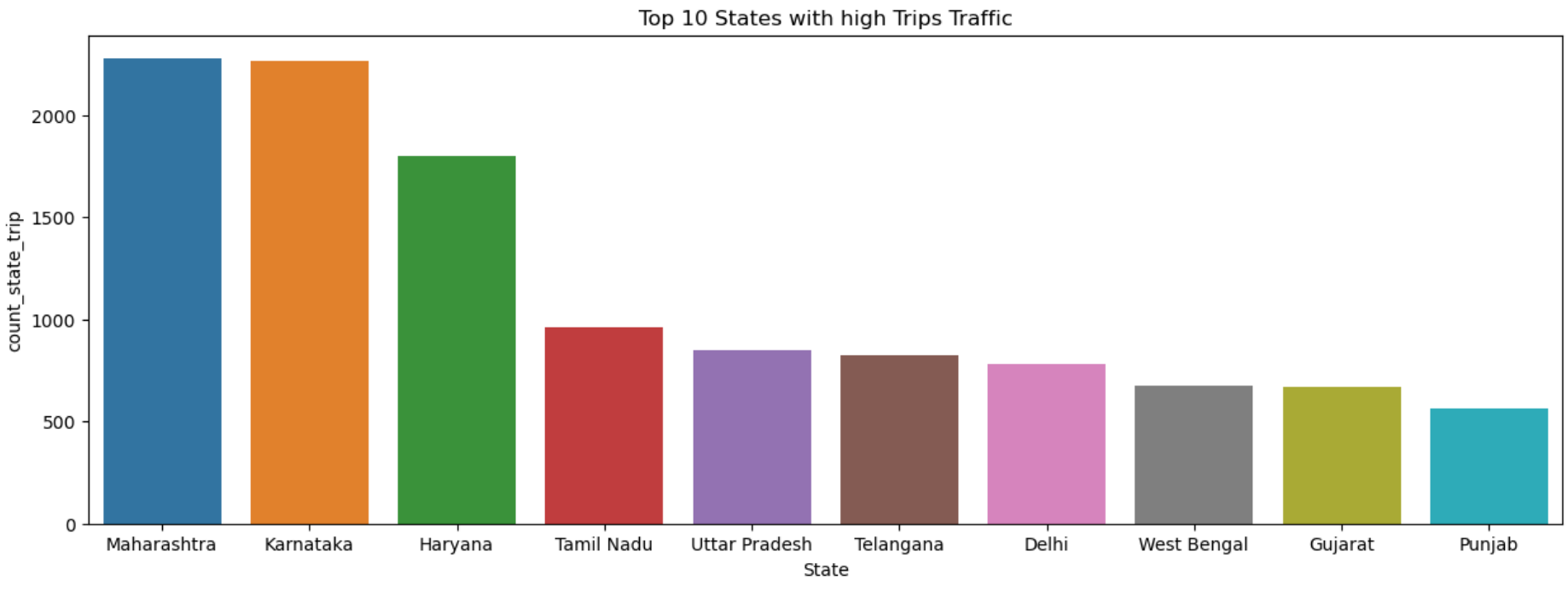
	route_type	count
0	Carting	45207
1	FTL	99660

In [77]:
```
plt.pie(route_type['count'], labels=route_type['route_type'],autopct='%.0f%%')
plt.title('Full Truck Load and Carting Route type')
plt.show()
```



1. Top 10 States having most Trips created for Delivery

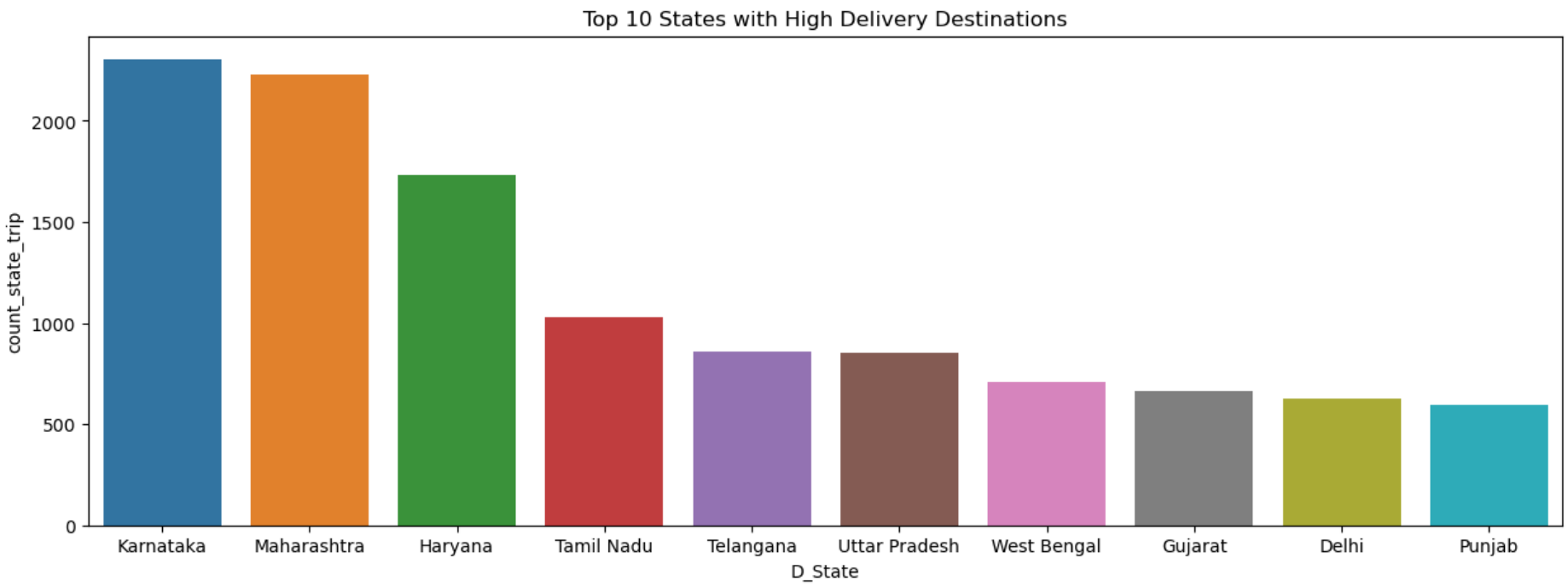
In [78]:
```
unique_trip_uuid = df.groupby(['trip_uuid','State']).aggregate(count=('State','count')).reset_index()
division = unique_trip_uuid.groupby(['State']).aggregate(count_state_trip=('trip_uuid','count')).reset_index()
top_10_State = division.sort_values(['count_state_trip'],ascending=False).head(10)

plt.figure(figsize=(15,5))
sns.barplot(x=top_10_State['State'],y=top_10_State['count_state_trip'])
plt.title('Top 10 States with high Trips Traffic')
plt.show()
```



- Maharashtra and Karnataka have high Delivery traffic followed by Haryana and Tamil Nadu

2. Top 10 Destinations where Delhivery company delivers the most

In [79]:
```
unique_trip_uuid = df.groupby(['trip_uuid','D_State']).aggregate(count=('D_State','count')).reset_index()
division = unique_trip_uuid.groupby(['D_State']).aggregate(count_state_trip=('trip_uuid','count')).reset_index()
top_10_State = division.sort_values(['count_state_trip'],ascending=False).head(10)

plt.figure(figsize=(15,5))
sns.barplot(x=top_10_State['D_State'],y=top_10_State['count_state_trip'])
plt.title('Top 10 States with High Delivery Destinations')
plt.show()
```

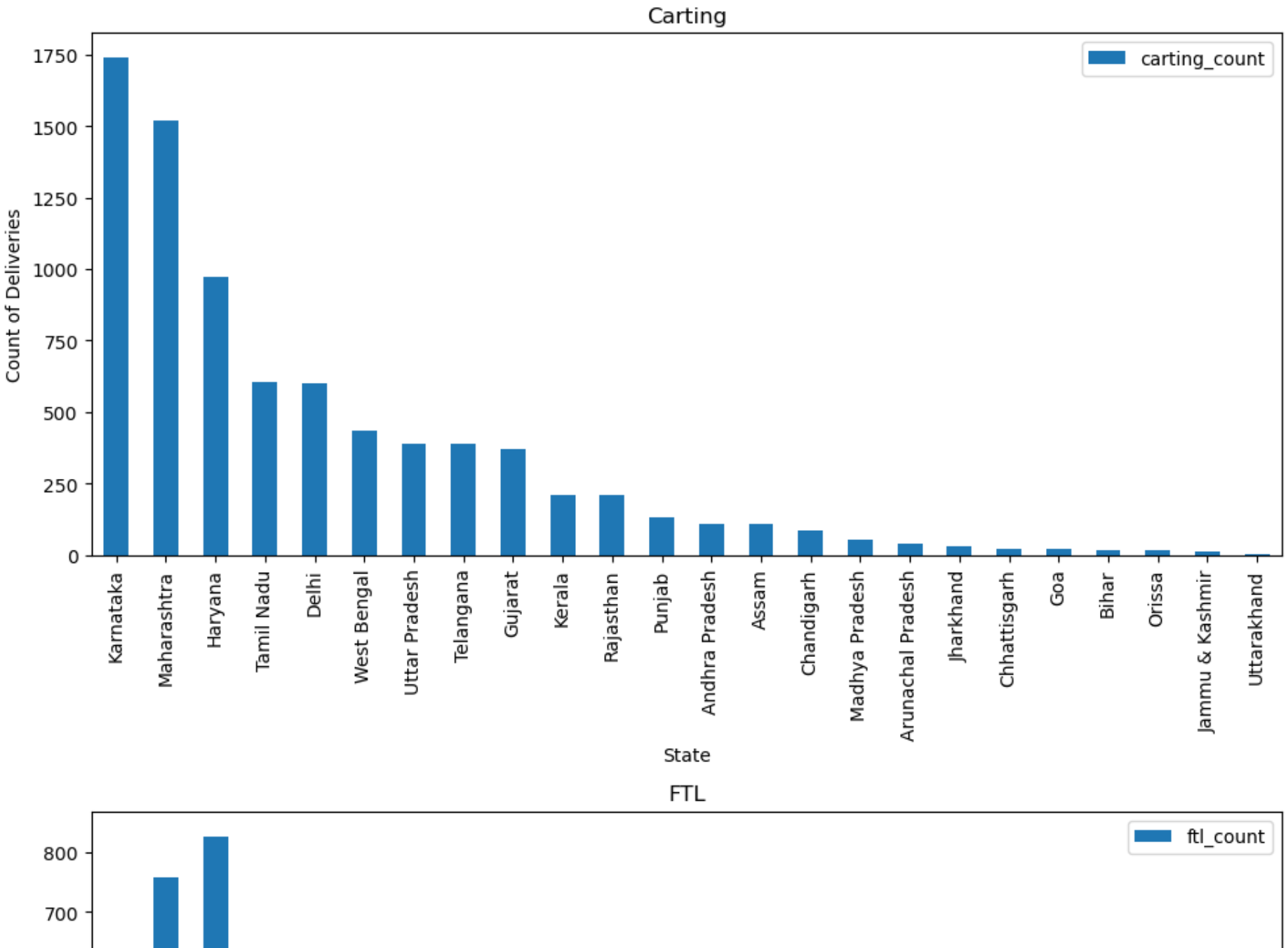


- Karnataka and Maharashtra have high Delivery traffic as Destination followed by Haryana and Tamil Nadu

3. State wise FTL and Carting route type count of trips

In [94]:
```
trip_state = df.groupby(['trip_uuid','State','route_type']).aggregate(count=('trip_uuid','count')).reset_index()
route_type_state = trip_state.groupby(['State','route_type']).aggregate(count_type=('trip_uuid','count')).reset_index()
route_type_state = route_type_state.sort_values(by=['count_type','State'],ascending=[False,True])
```

In [95]:
```
carting = route_type_state[route_type_state['route_type']=='Carting']
ftl = route_type_state[route_type_state['route_type']=='FTL']
carting_ftl = carting.merge(ftl, on=['State'], how='inner').reset_index()
carting_ftl['carting_count'] = carting_ftl['count_type_x']
carting_ftl['ftl_count'] = carting_ftl['count_type_y']
carting_ftl = carting_ftl[['State', 'carting_count', 'ftl_count']]

fig, axes = plt.subplots(nrows=2,ncols=1,figsize=(10,12))

carting_ftl.plot(kind='bar', x='State', y='carting_count', ax=axes[0])
axes[0].set_title('Carting')
axes[0].set_xlabel('State')
axes[0].set_ylabel('Count of Deliveries')

carting_ftl.plot(kind='bar', x='State', y='ftl_count', ax=axes[1])
axes[1].set_title('FTL')
axes[1].set_xlabel('State')
axes[1].set_ylabel('Count of Deliveries')


plt.tight_layout()

plt.show()
```

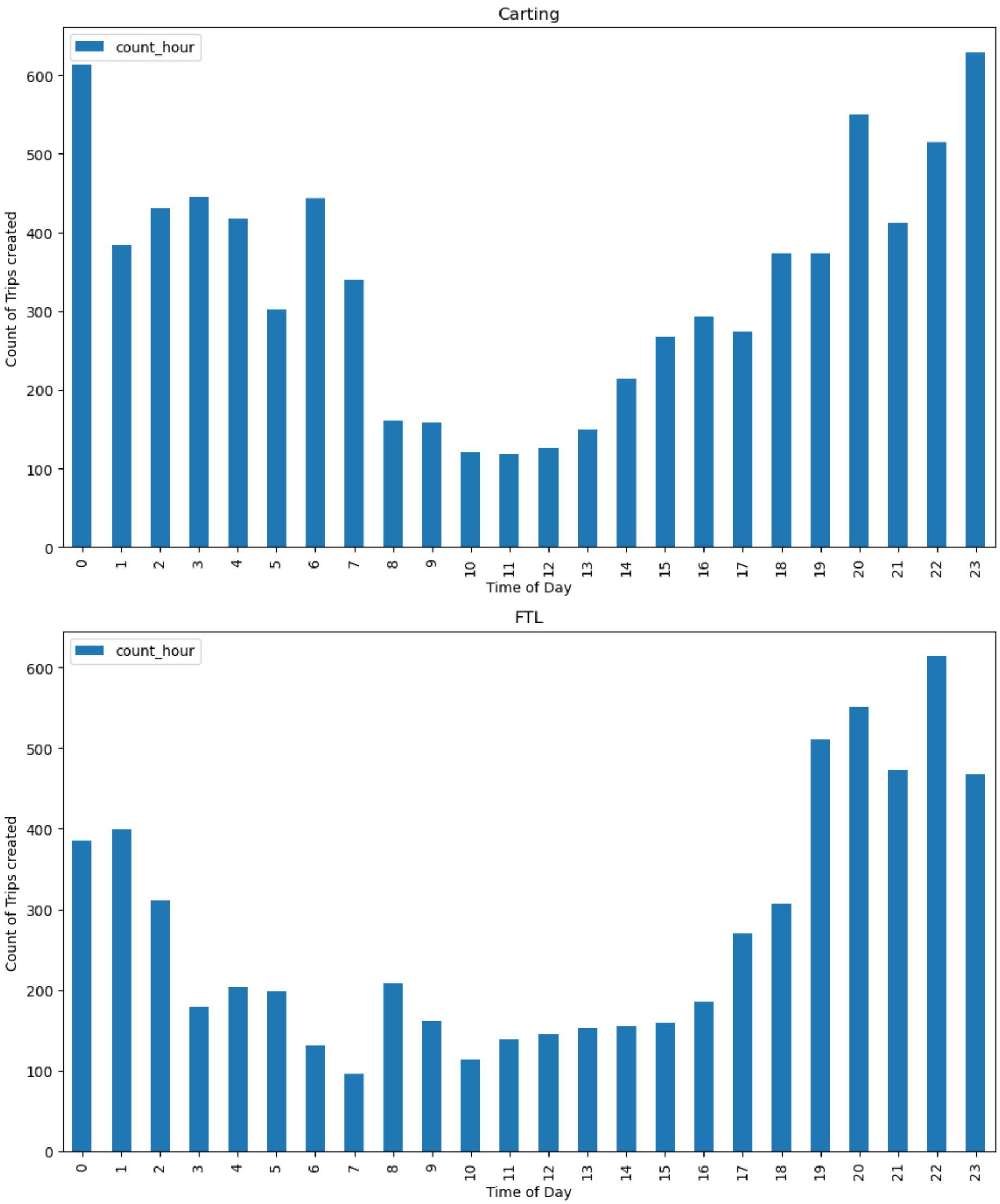


Karnataka and Maharashtra are very high at Carting route type and Haryana and Maharashtra at FTL route type. Overall Maharashtra tops at both Carting and FTL route type.

4.Time based Analysis for Source

In [82]:
```
trip_creation_time = df[['trip_uuid', 'State', 'route_type', 'trip_creation_time']].copy()
trip_creation_time['hour_of_day'] = trip_creation_time['trip_creation_time'].dt.hour
trip_creation_time = trip_creation_time.groupby(['trip_uuid', 'State', 'route_type', 'trip_creation_time',
                                                 'hour_of_day']).aggregate(count=('trip_uuid','count')).reset_index()
carting_ftl = trip_creation_time.groupby(['route_type',
                                          'hour_of_day']).aggregate(count_hour=('route_type','count')).reset_index()
carting = carting_ftl[carting_ftl['route_type']=='Carting']
ftl = carting_ftl[carting_ftl['route_type']=='FTL']

fig, axes = plt.subplots(nrows=2, ncols=1,figsize=(10,12))

carting.plot(kind='bar', x='hour_of_day', y='count_hour', ax=axes[0])
axes[0].set_title('Carting')
axes[0].set_xlabel('Time of Day')
axes[0].set_ylabel('Count of Trips created')
```

```
ftl.plot(kind='bar', x='hour_of_day', y='count_hour', ax=axes[1])
axes[1].set_title('FTL')
axes[1].set_xlabel('Time of Day')
axes[1].set_ylabel('Count of Trips created')

plt.tight_layout()

plt.show()
```

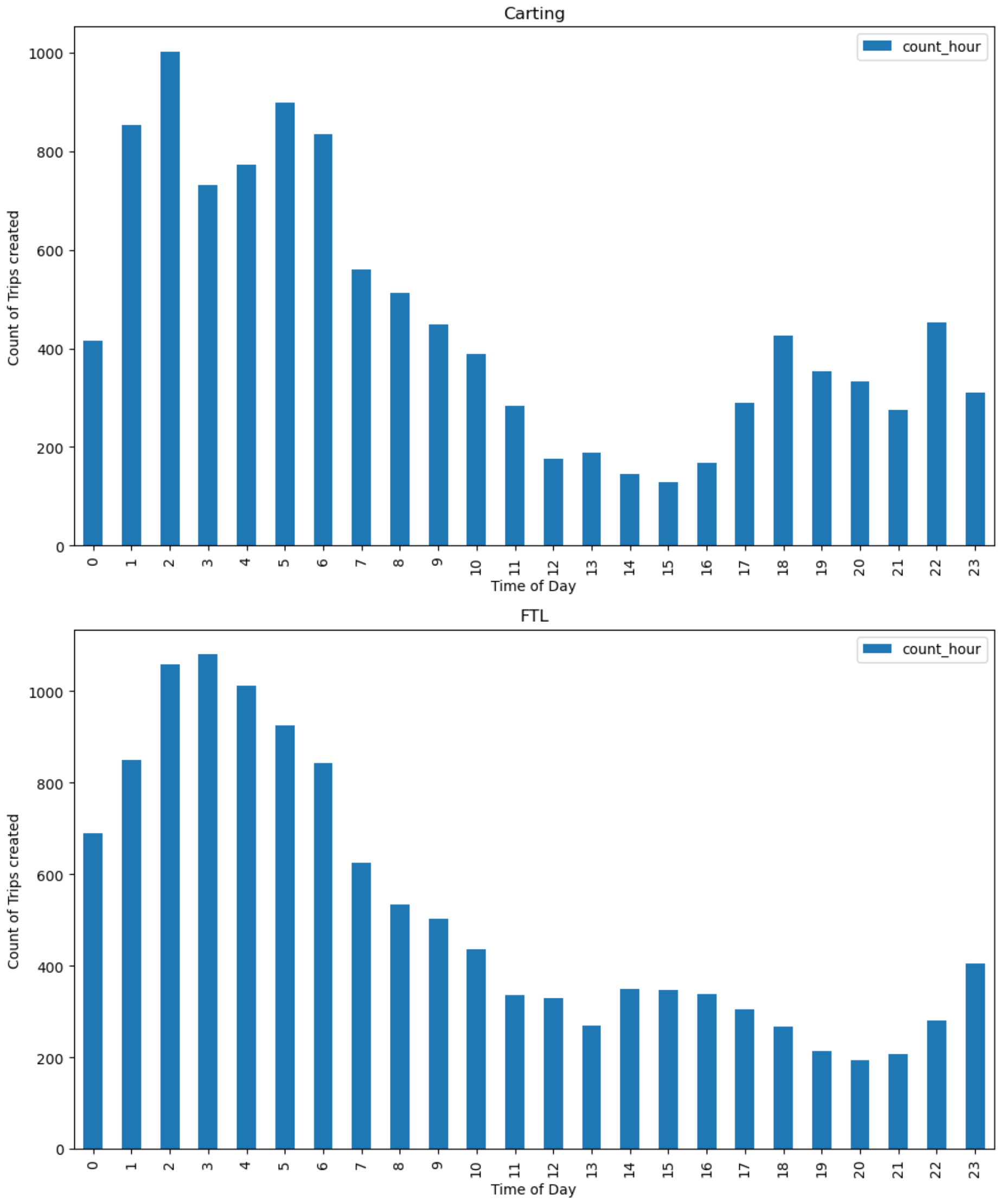


Carting trips are mostly generated at Night after 8 pm till 1 am where as for FTL trips the trips generated are from 7 pm to 12 am.

5. Time based Analysis for Destination

In [83]:
```
trip_creation_time = df[['trip_uuid', 'D_State', 'route_type', 'od_end_time']].copy()
trip_creation_time['hour_of_day'] = trip_creation_time['od_end_time'].dt.hour
trip_creation_time = trip_creation_time.groupby(['trip_uuid', 'D_State', 'route_type', 'od_end_time',
                                                 'hour_of_day']).aggregate(count=('trip_uuid','count')).reset_index()
carting_ftl = trip_creation_time.groupby(['route_type',
                                          'hour_of_day']).aggregate(count_hour=('route_type','count')).reset_index()
```

```
carting = carting_ftl[carting_ftl['route_type']=='Carting']
ftl = carting_ftl[carting_ftl['route_type']=='FTL']

fig,axes = plt.subplots(nrows=2, ncols=1, figsize=(10,12))
carting.plot(kind='bar', x='hour_of_day', y='count_hour', ax=axes[0])
axes[0].set_title('Carting')
axes[0].set_xlabel('Time of Day')
axes[0].set_ylabel('Count of Trips created')

ftl.plot(kind='bar', x='hour_of_day', y='count_hour', ax=axes[1])
axes[1].set_title('FTL')
axes[1].set_xlabel('Time of Day')
axes[1].set_ylabel('Count of Trips created')

plt.tight_layout()

plt.show()
```



Carting trips are mostly delivered to destination in morning from 1 am to 7 am where as for FTL trips they reach the destination from 1 am to 6 am

2. Merging the rows:

Grouping by segment

In [51]:
```
df['segment_key'] = df['trip_uuid'] + df['source_center'] + df['destination_center']
df['segment_actual_time' + '_sum'] = df.groupby('segment_key')['segment_actual_time'].aggregate('cumsum')
df['segment_osrm_distance' + '_sum'] = df.groupby('segment_key')['segment_osrm_distance'].aggregate('cumsum')
df['segment_osrm_time' + '_sum'] = df.groupby('segment_key')['segment_osrm_time'].aggregate('cumsum')
```

In [52]:
```
df[df['trip_uuid']=='trip-153741093647649320'][['trip_uuid', 'source_center',
       'destination_center', 'actual_time',
       'osrm_time', 'segment_actual_time',
       'segment_osrm_time', 'segment_actual_time_sum','segment_osrm_distance_sum','segment_osrm_time_sum']]
```

Out[52]:

	trip_uuid	source_center	destination_center	actual_time	osrm_time	segment_actual_time	segment_osrm_time	segment_actual_time_sum	s
0	trip-153741093647649320	IND388121AAA	IND388620AAB	14.0	11.0	14.0	11.0	14.0	
1	trip-153741093647649320	IND388121AAA	IND388620AAB	24.0	20.0	10.0	9.0	24.0	
2	trip-153741093647649320	IND388121AAA	IND388620AAB	40.0	28.0	16.0	7.0	40.0	
3	trip-153741093647649320	IND388121AAA	IND388620AAB	62.0	40.0	21.0	12.0	61.0	
4	trip-153741093647649320	IND388121AAA	IND388620AAB	68.0	44.0	6.0	5.0	67.0	
5	trip-153741093647649320	IND388620AAB	IND388320AAA	15.0	11.0	15.0	11.0	15.0	
6	trip-153741093647649320	IND388620AAB	IND388320AAA	44.0	17.0	28.0	6.0	43.0	
7	trip-153741093647649320	IND388620AAB	IND388320AAA	65.0	29.0	21.0	11.0	64.0	
8	trip-153741093647649320	IND388620AAB	IND388320AAA	76.0	39.0	10.0	10.0	74.0	
9	trip-153741093647649320	IND388620AAB	IND388320AAA	102.0	45.0	26.0	6.0	100.0	

Aggregating at segment level

In [53]:
```
segment_dict = {
    'data':'first', 'trip_creation_time': 'first', 'route_schedule_uuid' : 'first', 'route_type' : 'first',
    'trip_uuid' : 'first', 'source_center' : 'first', 'source_name' : 'first',

    'destination_center' : 'last', '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' : 'last', 'segment_osrm_distance_sum' : 'last', 'segment_osrm_time_sum' : 'last'
}
```

In [54]:
```
segment = df.groupby('segment_key').agg(segment_dict).reset_index()
segment = segment.sort_values(by=['segment_key','od_end_time'],ascending=True).reset_index()
```

In [55]:
```
segment[segment['trip_uuid']=='trip-153741093647649320'][['trip_uuid', 'source_center', 'destination_center',
                                                          'actual_time', 'osrm_time', 'segment_actual_time_sum',
                                                          'segment_osrm_time_sum']]
```

Out[55]:

	trip_uuid	source_center	destination_center	actual_time	osrm_time	segment_actual_time_sum	segment_osrm_time_sum
10374	trip-153741093647649320	IND388121AAA	IND388620AAB	68.0	44.0	67.0	44.0
10375	trip-153741093647649320	IND388620AAB	IND388320AAA	102.0	45.0	100.0	44.0

In [12]:
```
segment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26368 entries, 0 to 26367
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   index                           26368 non-null  int64
 1   segment_key                     26368 non-null  object
 2   data                            26368 non-null  object
 3   trip_creation_time              26368 non-null  datetime64[ns]
 4   route_schedule_uuid             26368 non-null  object
 5   route_type                      26368 non-null  object
 6   trip_uuid                       26368 non-null  object
 7   source_center                   26368 non-null  object
 8   source_name                     26368 non-null  object
 9   destination_center              26368 non-null  object
 10  destination_name                26368 non-null  object
 11  od_start_time                   26368 non-null  datetime64[ns]
 12  od_end_time                     26368 non-null  datetime64[ns]
 13  start_scan_to_end_scan          26368 non-null  float64
 14  actual_distance_to_destination  26368 non-null  float64
 15  actual_time                     26368 non-null  float64
 16  osrm_time                       26368 non-null  float64
 17  osrm_distance                   26368 non-null  float64
 18  segment_actual_time_sum         26368 non-null  float64
 19  segment_osrm_distance_sum       26368 non-null  float64
 20  segment_osrm_time_sum           26368 non-null  float64
dtypes: datetime64[ns](3), float64(8), int64(1), object(9)
memory usage: 4.2+ MB
```

In [13]:
```
segment.describe().transpose()
```

Out[13]:

	count	mean	min	25%	50%	75%	max	std
index	26368.0	13183.5	0.0	6591.75	13183.5	19775.25	26367.0	7611.930285
trip_creation_time	26368	2018-09-22 14:43:06.143810560	2018-09-12 00:00:00	2018-09-17 04:43:00	2018-09-22 04:42:00	2018-09-27 20:22:15	2018-10-03 23:59:00	NaN
od_start_time	26368	2018-09-22 18:35:02.776092160	2018-09-12 00:00:00	2018-09-17 08:36:00	2018-09-22 08:33:30	2018-09-28 00:13:15	2018-10-06 04:27:00	NaN
od_end_time	26368	2018-09-22 23:33:49.632888576	2018-09-12 00:50:00	2018-09-17 16:26:45	2018-09-22 16:37:30	2018-09-28 03:42:00	2018-10-08 03:00:00	NaN
start_scan_to_end_scan	26368.0	298.278671	20.0	91.0	152.0	307.0	7898.0	440.561588
actual_distance_to_destination	26368.0	92.425217	9.001351	21.684419	35.114228	65.750726	1927.447705	209.415035
actual_time	26368.0	200.690193	9.0	51.0	84.0	168.0	4532.0	384.85364
osrm_time	26368.0	90.686704	6.0	25.0	39.0	72.0	1686.0	185.080423
osrm_distance	26368.0	114.827642	9.0729	27.764725	43.63305	85.566975	2326.1991	253.773765
segment_actual_time_sum	26368.0	198.863092	9.0	50.0	83.0	166.0	4504.0	381.283224
segment_osrm_distance_sum	26368.0	125.42368	9.0729	28.4713	45.9444	91.351975	2640.9247	285.932556
segment_osrm_time_sum	26368.0	101.681318	6.0	25.0	42.0	79.0	1938.0	215.650948

3. Feature Engineering:

Calculate time taken between od_start_time and od_end_time

In [56]:
```
segment['od_time_diff_hour'] = ((segment['od_end_time'] - segment['od_start_time']).dt.total_seconds())/60
segment['od_time_diff_hour']
```

Out[56]:
```
0        1261.0
1         999.0
2          58.0
3         123.0
4         834.0
          ...
26363      62.0
26364      91.0
26365      45.0
26366     288.0
26367      67.0
Name: od_time_diff_hour, Length: 26368, dtype: float64
```

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

In [ ]:
```
df[['D_City','D_SubUrbs','D_Division']] = df['destination_name'].str.split('_', expand=True, n=2)
#df.drop('destination_name',axis=1,inplace=True)

df['D_State'] = df['D_Division'].str.extract(r'\((.*?)\)')
```

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

In [ ]:
```
df[['City','SubUrbs','Division']] = df['source_name'].str.split('_', expand=True, n=2)
#df.drop('source_name',axis=1,inplace=True)

df['State'] = df['Division'].str.extract(r'\((.*?)\)')
```

Trip_creation_time: Extract features like month, year, day, etc.

In [ ]:
```
df['trip_creation_date'] = df['trip_creation_time'].dt.date
df['od_start_date'] = df['od_start_time'].dt.date
df['od_end_date'] = df['od_end_time'].dt.date
df['cutoff_date'] = df['cutoff_timestamp'].dt.date

df['od_start_date_hour'] = df['od_start_time'].dt.hour



df['trip_year'] = df['trip_creation_time'].dt.year
df['trip_month'] = df['trip_creation_time'].dt.month
df['trip_hour'] = df['trip_creation_time'].dt.hour
df['trip_day'] = df['trip_creation_time'].dt.day
df['trip_week'] = df['trip_creation_time'].dt.isocalendar().week
df['trip_dayofweek'] = df['trip_creation_time'].dt.dayofweek
df['time_category'] = pd.cut(df['trip_hour'],
                             bins=[-1, 4, 12, 16, 24],
                             labels=['Night', 'Morning', 'Noon', 'Evening'],
                             include_lowest=True)
```

4. In-depth analysis:

Grouping and Aggregating at Trip-level

In [68]:
```
trip_dict = { 'data' : 'first', 'trip_creation_time': 'first', 'route_schedule_uuid' : 'first', 'route_type' : 'first',
              'trip_uuid' : 'first', 'source_center' : 'first', 'source_name' : 'first',

              'destination_center' : 'last', 'destination_name' : 'last',

              'start_scan_to_end_scan' : 'sum', 'od_time_diff_hour' : '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'

    }
```

In [69]:
```
trip = segment.groupby('trip_uuid').agg(trip_dict).reset_index(drop=True)
trip.head(2)
```

Out[69]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	
0	training	2018-09-12	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	K
1	training	2018-09-12	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	Dodd

In [70]:
```
trip[trip['trip_uuid']=='trip-153741093647649320'][['trip_uuid', 'source_center',
       'destination_center', 'actual_time',
       'osrm_time', 'segment_actual_time_sum','segment_osrm_time_sum']]
```

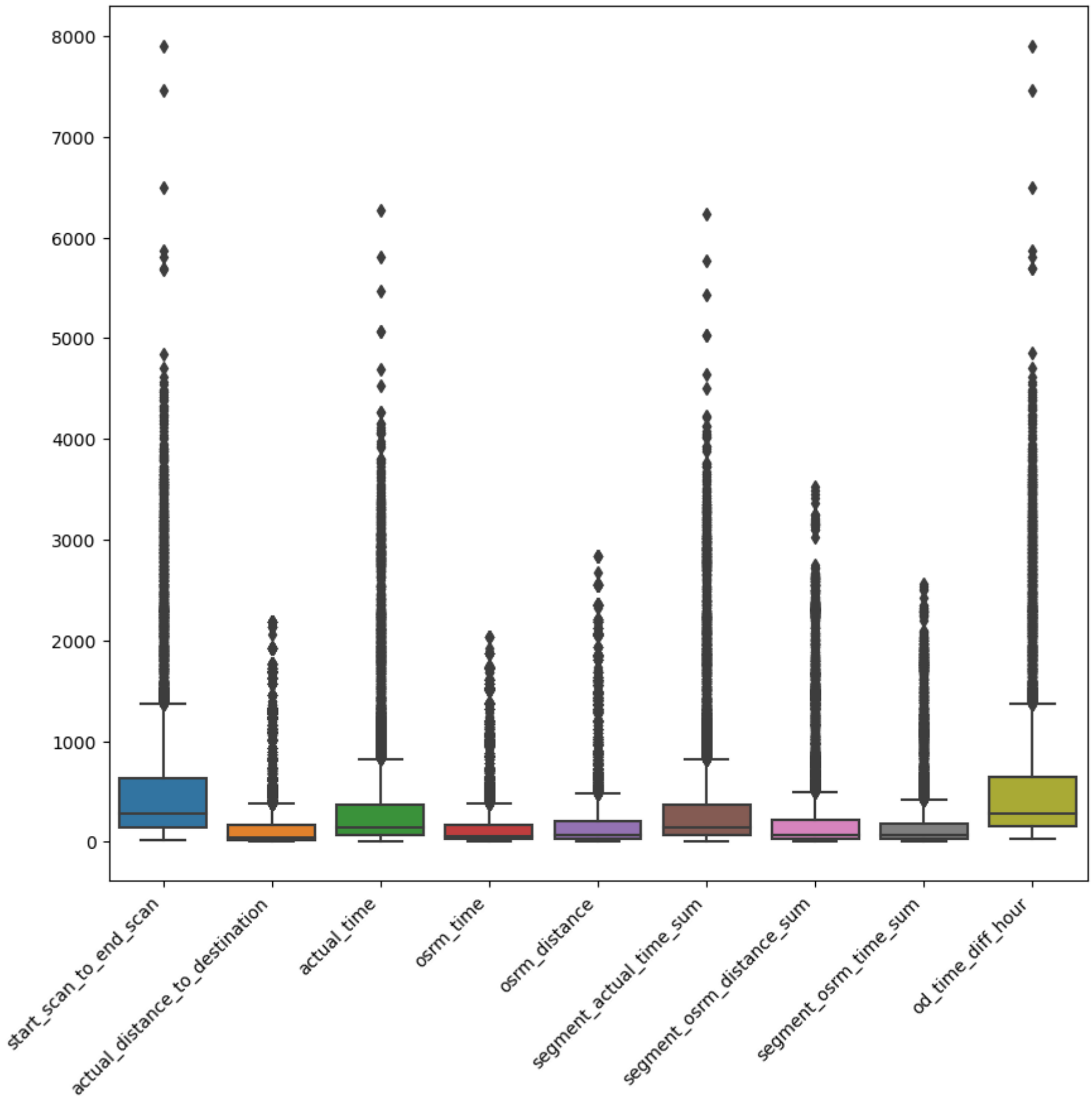
Out[70]:

	trip_uuid	source_center	destination_center	actual_time	osrm_time	segment_actual_time_sum	segment_osrm_time_sum
5919	trip-153741093647649320	IND388121AAA	IND388320AAA	170.0	89.0	167.0	88.0

Outlier Detection & Treatment

In [71]:
```
num_cols = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time',
            'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_distance_sum',
            'segment_osrm_time_sum', 'od_time_diff_hour']
```

In [72]:
```
plt.figure(figsize=(10,9))
sns.boxplot(data=trip[num_cols])
plt.xticks(rotation=45, ha='right')
plt.show()
```



- As we can see from the above box plot there are outliers present for almost all the numeric data we have passed

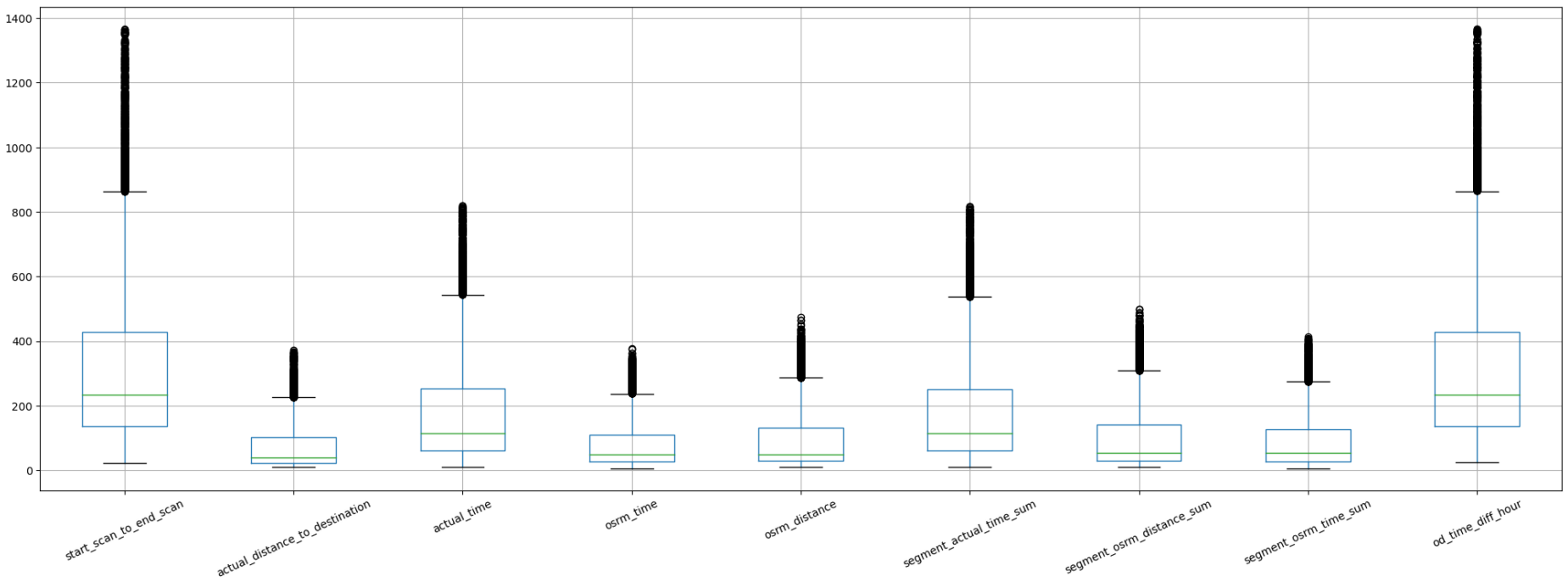
In [73]:
```
Q1 = trip[num_cols].quantile(0.25)
Q3 = trip[num_cols].quantile(0.75)

IQR = Q3-Q1
IQR
```

Out[73]:
```
start_scan_to_end_scan            488.000000
actual_distance_to_destination    141.745969
actual_time                       303.000000
osrm_time                         139.000000
osrm_distance                     177.655800
segment_actual_time_sum           301.000000
segment_osrm_distance_sum         186.147900
segment_osrm_time_sum             154.000000
od_time_diff_hour                 488.000000
dtype: float64
```

In [74]:
```
print(trip.shape , "trip data frame before outliers present")
trip = trip[~((trip[num_cols] < (Q1 - 1.5 * IQR)) | (trip[num_cols] > (Q3 + 1.5 * IQR))).any(axis=1)]

print(trip.shape, " trip data frame after outlier removal")

trip = trip.reset_index(drop=True)
```

```
(14817, 18) trip data frame before outliers present
(12759, 18)  trip data frame after outlier removal
```

In [75]:
```
trip[num_cols].boxplot(rot=25, figsize=(25,8))
plt.show()
```



Perform one-hot encoding on categorical features.

In [81]:
```
# List of categorical columns excluding 'route_schedule_uuid'
categorical_columns = ['route_type', 'source_center', 'source_name',
                       'destination_center', 'destination_name']

# List of numerical columns
numerical_columns = ['start_scan_to_end_scan', 'od_time_diff_hour',
                     'actual_distance_to_destination', 'actual_time',
                     'osrm_time', 'osrm_distance', 'segment_actual_time_sum',
                     'segment_osrm_distance_sum', 'segment_osrm_time_sum']

# Ensure numerical columns have numeric data type
trip[numerical_columns] = trip[numerical_columns].astype(float)

# Initialize OneHotEncoder with sparse_output=True to keep the matrix sparse
encoder = OneHotEncoder(sparse_output=True)

# Fit and transform the categorical columns
encoded_columns = encoder.fit_transform(trip[categorical_columns])

# Convert numerical columns to a sparse matrix
numerical_sparse = trip[numerical_columns].values

# Concatenate the encoded columns with the numerical columns
trip_encoded = hstack([numerical_sparse, encoded_columns])

# Display the shape of the encoded DataFrame
trip_encoded_df = pd.DataFrame(trip_encoded.toarray())
encoded_df = pd.DataFrame(encoded_columns.toarray(), columns=encoder.get_feature_names_out(categorical_columns))

# Concatenate the encoded DataFrame with the original DataFrame
trip_encoded = pd.concat([trip.drop(columns=categorical_columns), encoded_df], axis=1)

# Display the shape of the encoded DataFrame
print(trip_encoded.shape)
trip_encoded.head()
```

```
(12759, 3840)
```

Out[81]:

	data	trip_creation_time	route_schedule_uuid	trip_uuid	start_scan_to_end_scan	od_time_diff_hour	actual_distance_to_destination	actu
0	training	2018-09-12 00:00:00	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	trip-153671042288605164	180.0	181.0	73.186911	
1	training	2018-09-12 00:01:00	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	trip-153671046011330457	100.0	100.0	17.175274	
2	training	2018-09-12 00:02:00	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	trip-153671052974046625	717.0	718.0	127.448500	
3	training	2018-09-12 00:02:00	thanos::sroute:9bf03170-d0a2-4a3f-aa4d-9aaab3d...	trip-153671055416136166	189.0	191.0	24.597048	
4	training	2018-09-12 00:04:00	thanos::sroute:a97698cc-846e-41a7-916b-88b1741...	trip-153671066201138152	98.0	98.0	9.100510	

5 rows × 3840 columns

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler

In [110…
```
scaler = StandardScaler()
scaler.fit(trip[num_cols])
```

Out[110]:
```
▾ StandardScaler
StandardScaler()
```

5. Hypothesis Testing:

In [27]:
```
df1 = df.copy()
df1.head(2)
```

Out[27]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	destination_center	od_start_time	od_end_time	st
0	training	2018-09-20 02:35:00	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	IND388620AAB	2018-09-20 03:21:00	2018-09-20 04:47:00	
1	training	2018-09-20 02:35:00	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	IND388620AAB	2018-09-20 03:21:00	2018-09-20 04:47:00	

2 rows × 35 columns

In [26]:
```
Total_trip = df1['trip_uuid'].nunique()
print("Total Trips we have:", Total_trip)
```

```
Total Trips we have: 14817
```

In [28]:
```
df1=df1[['route_schedule_uuid', 'route_type','trip_creation_date',
       'trip_uuid', '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',  'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor']]
df1.head(2)
```

Out[28]:

	route_schedule_uuid	route_type	trip_creation_date	trip_uuid	od_start_time	od_end_time	start_scan_to_end_scan	is_cutoff	cutoff_factor
0	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	2018-09-20	trip-153741093647649320	2018-09-20 03:21:00	2018-09-20 04:47:00	86.0	True	9
1	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	2018-09-20	trip-153741093647649320	2018-09-20 03:21:00	2018-09-20 04:47:00	86.0	True	18

We compare the Actual time and OSRM time and test the hypothesis for it.

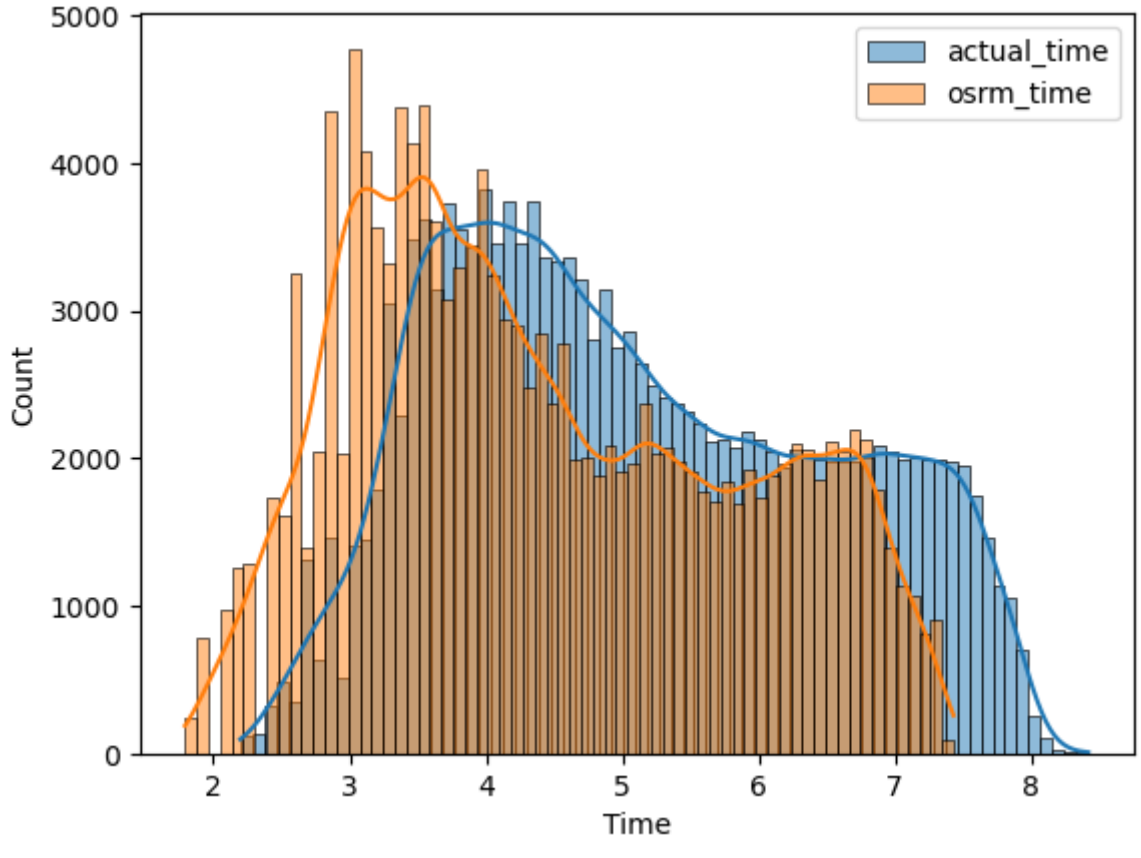
Hypothesis test 1:

- Ho: The actual time taken and predicted time by OSRM is same.
- Ha: The actual time taken is greater than predicted time by OSRM.

In [33]:
```
alpha = 0.05
tstat,pvalue = ttest_ind(df1['actual_time'],df1['osrm_time'],alternative='greater')
#aggregated_df = df.groupby('trip_uuid').agg({'actual_time': 'mean', 'osrm_time': 'mean'}).reset_index()

# Hypothesis Testing (Paired t-test)
t_stat, p_value = ttest_rel(aggregated_df['actual_time'], aggregated_df['osrm_time'])
alpha = 0.05

print(f'T-statistic: {t_stat}\nP-value: {p_value}')
if pvalue<alpha:
    print('We reject Null Hypothesis and conclude that Actual time taken is greater than OSRM predicted time.')
else:
    print('We failed to reject Null Hypothesis.')
```

```
T-statistic: 72.4761096106345
P-value: 0.0
We reject Null Hypothesis and conclude that Actual time taken is greater than OSRM predicted time.
```

In [30]:
```
sns.histplot(np.log(df1['actual_time']), kde=True, label='actual_time')
plt.legend()
sns.histplot(np.log(df1['osrm_time']), kde=True, label='osrm_time')
plt.legend()
plt.xlabel('Time')
plt.show()
```



- We can see that the Actual time taken to deliver is more than predicted time by OSRM.

Hypothesis test of actual_time v/s segment_actual_time_sum

In [111…
```
trip[['actual_time','segment_actual_time_sum']].head()
```

Out[111]:

	actual_time	segment_actual_time_sum
0	143.0	141.0
1	59.0	59.0
2	341.0	340.0
3	61.0	60.0
4	24.0	24.0

In [112…
```
trip['actual_time'].mean(), trip['segment_actual_time_sum'].mean()
```

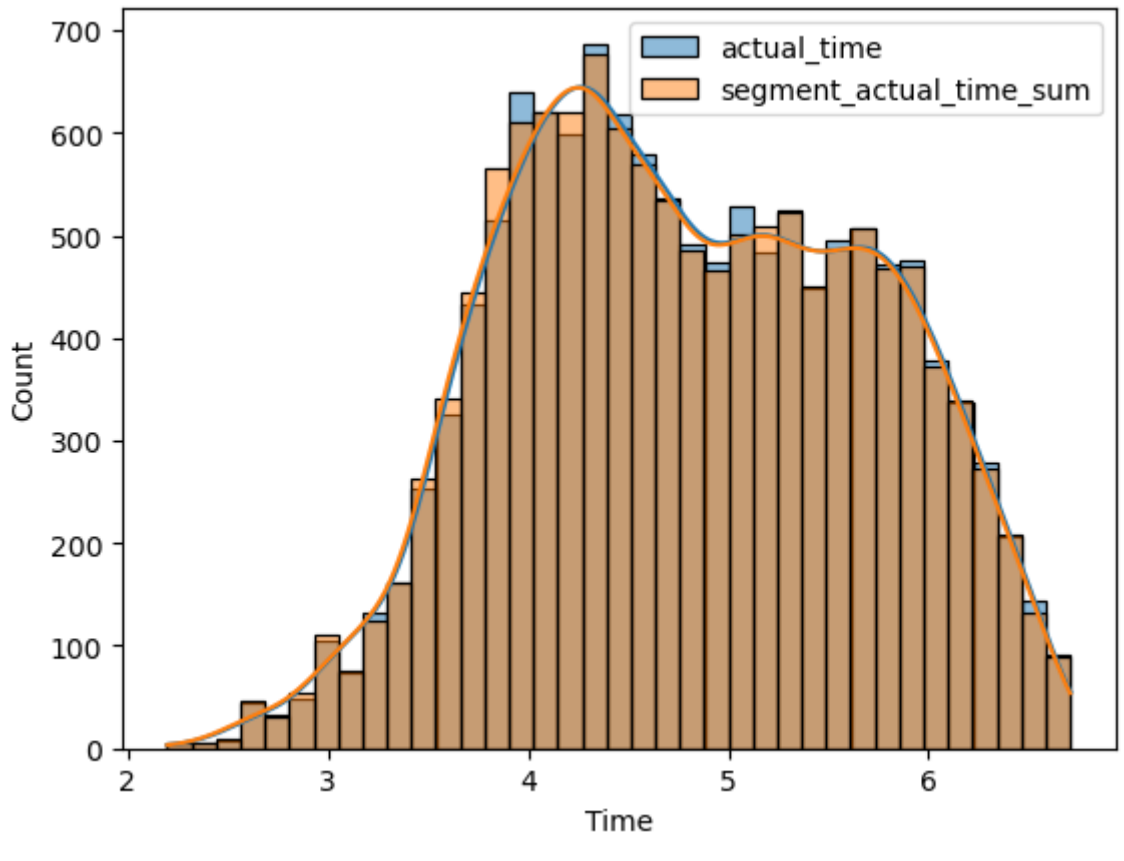
Out[112]:
```
(178.55623481464065, 176.89348695038797)
```

- Null Hypothesis (H0): The means of both times are the same; the difference is not significant.
- Alternative Hypothesis (Ha): The difference is significant; the means of both times are not equal.

In [113…
```
t_stat,p_value = ttest_ind(trip['actual_time'], trip['segment_actual_time_sum'])
print("p_value: ", p_value)

if p_value < 0.05:
    print("Reject H0 : The difference is significant; the means of both times are not equal")
else:
    print("Fail to reject H0 : This means of both times are the same; the difference is not significant")
```

```
p_value:  0.4022851737856503
Fail to reject H0 : This means of both times are the same; the difference is not significant
```

In [116…
```
sns.histplot(np.log(trip['actual_time']), kde=True, label='actual_time')
plt.legend()
sns.histplot(np.log(trip['segment_actual_time_sum']), kde=True, label='segment_actual_time_sum')
plt.legend()
plt.xlabel('Time')
plt.show()
```

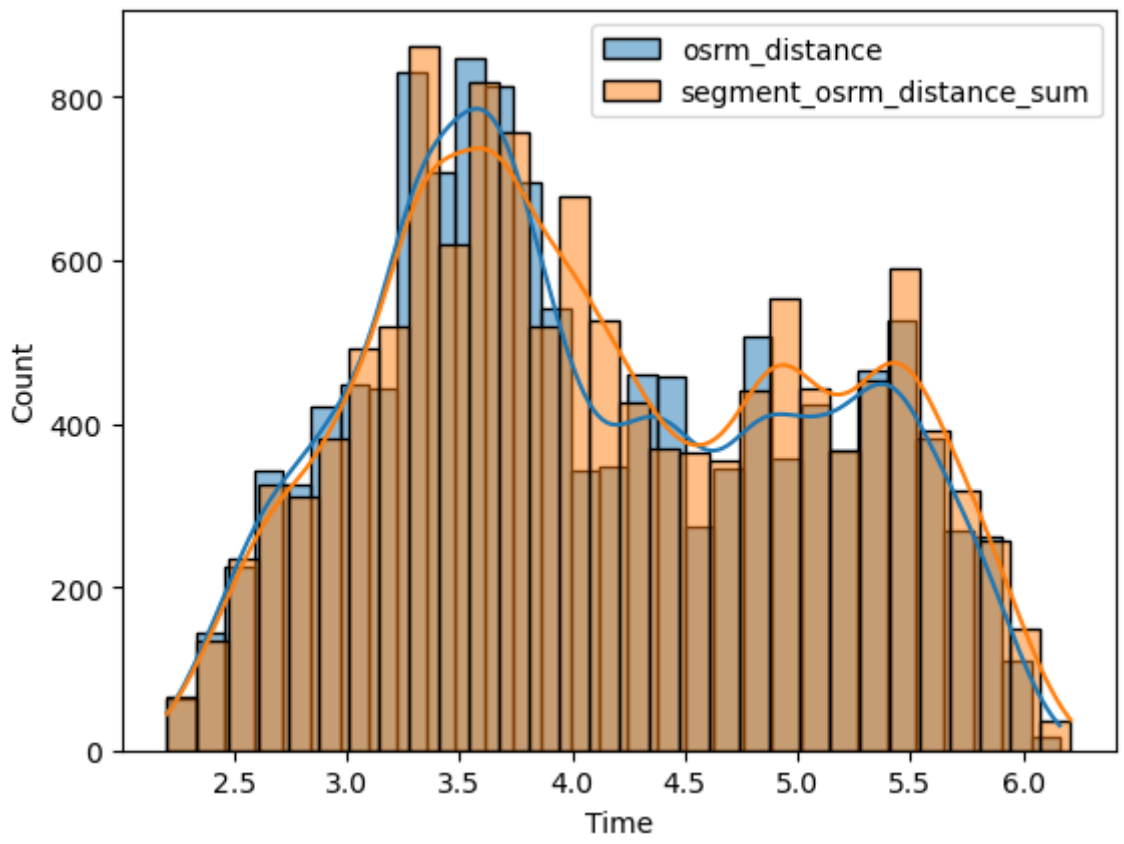


Hypothesis test: OSRM distance aggregated value and segment OSRM distance aggregated value

In [115…
```
t_stat, p_value = ttest_ind(trip['osrm_distance'], trip['segment_osrm_distance_sum'])
print("p_value: ", p_value)

if p_value < 0.05:
    print("Reject H0 : The difference is significant; the means of both times are not equal")
else:
    print("Fail to reject H0 : This means of both times are the same; the difference is not significant")
```

```
p_value:  6.44494577799341e-08
Reject H0 : The difference is significant; the means of both times are not equal
```

In [117…
```
sns.histplot(np.log(trip['osrm_distance']), kde=True, label='osrm_distance')
plt.legend()
sns.histplot(np.log(trip['segment_osrm_distance_sum']), kde=True, label='segment_osrm_distance_sum')
plt.legend()
plt.xlabel('Time')
plt.show()
```



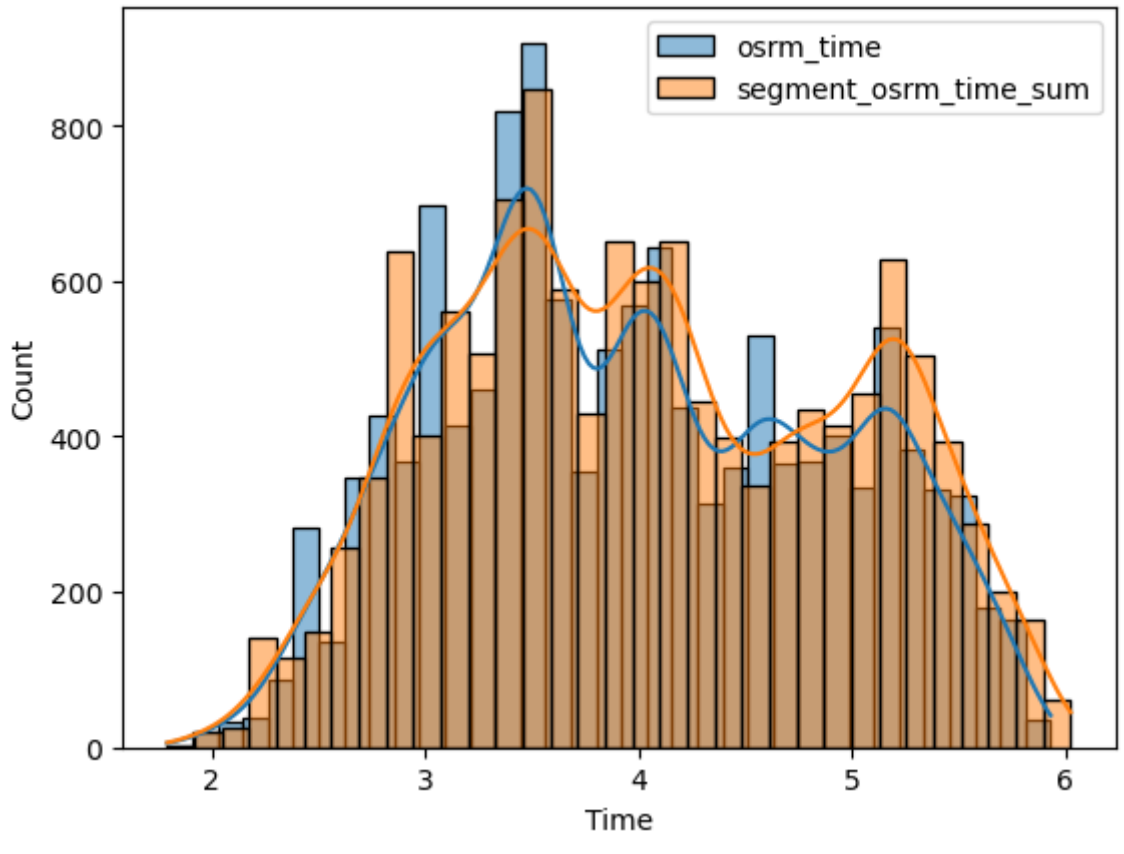
Hypothesis test of: OSRM time aggregated value and segment OSRM time aggregated value

In [114…
```
t_stat, p_value = ttest_ind(trip['osrm_time'], trip['segment_osrm_time_sum'])
print("p_value: ", p_value)

if p_value < 0.05:
    print("Reject H0 : The difference is significant; the means of both times are not equal")
else:
    print("Fail to reject H0 : This means of both times are the same; the difference is not significant")
```

```
p_value:  4.520723895551576e-15
Reject H0 : The difference is significant; the means of both times are not equal
```

In [118…
```
sns.histplot(np.log(trip['osrm_time']), kde=True, label='osrm_time')
plt.legend()
sns.histplot(np.log(trip['segment_osrm_time_sum']), kde=True, label='segment_osrm_time_sum')
```

```
plt.legend()
plt.xlabel('Time')
plt.show()
```



6. Business Insights & Recommendations

- There is a significant difference between OSRM and actual parameters and also there is a significant difference for OSRM actual and segmented values but not significant difference in case of actual & segmented time.

1. **Warehouse Expansion**: It is recommended that Delhivery explore warehouse expansion endeavors in states such as **UP, Telangana, and the outer regions of Delhi**. This strategic initiative is imperative given the discerned trend where order cancellations are notably influenced by delivery time considerations.

2. **Optimizing OSRM Usage**: In optimizing OSRM predictions, a prudent strategy for Delhivery would be to predominantly favor Carting route type deliveries, as this aligns closely with the observed actual time taken. This strategic alignment is anticipated to enhance operational efficiency.

3. **Hypothesis test data and results for Time**: We see that actual time taken to reach destination is more than predicted time by OSRM . The error range lies between -110 min to 190 mins (approx 2hrs early to 3hrs late) for Carting route type and -384 min to 938 min (approx 6 hours early to 15 hrs late) for FTL route type.

4. **Hypothesis test data and results for Distance**: For Actual distance and OSRM distance even though OSRM tries to find best route to reach destination but we can see that many times **actual time taken to reach the destination is less than predicted distance**. The drivers have closer way to reach destination than provided by OSRM.