Delhivery Business Case Study

1. Defining Problem Statement and Analyzing Basic Metrics

1.1 Defining Problem Statement

The goal is to clean, sanitize, and manipulate Delhivery's raw data to generate useful features. This will assist the data science team in building forecasting models to enhance business quality, efficiency, and profitability.

1.2 Analyzing Basic Metrics

In [7]:
```
# Importing required Python libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import scikit_posthocs as sp
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [8]:
```
# Load the dataset
df = pd.read_csv("delhivery_data.csv")
```

In [9]:
```
# Observations on shape of data
print("Shape of the Dataset:")
print(df.shape)
```

```
Shape of the Dataset:
(144867, 24)
```

In [10]:
```
# Data types of all the attributes
print("\nData Types of Columns:")
print(df.info())
```

```
Data Types of Columns:
<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
```

In [11]:

```
# Check for missing values in each column
print("Missing values per column:")
print(df.isnull().sum())
```

```
Missing values per 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
```

In [12]:
```
# Removing null values
df = df.dropna(how='any')
df = df.reset_index(drop=True)
```

1.3 Converting time columns into pandas datatime

In [14]:
```
df['od_start_time']=pd.to_datetime(df['od_start_time'])
df['od_end_time']=pd.to_datetime(df['od_end_time'])
```

In [15]:
```
# Getting first 5 rows of the data
df.head(15)
```

Out[15]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB
10	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG
11	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG
12	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG
13	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG
14	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG

15 rows × 24 columns

In [16]:
```
df.tail(15)
```

Out[16]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
144301	training	2018-09-22 11:30:41.399439	thanos::sroute:d81088e2-9ccd-43e9-9260-3e85633...	FTL	trip-153761584139918815	IND421302
144302	training	2018-09-22 11:30:41.399439	thanos::sroute:d81088e2-9ccd-43e9-9260-3e85633...	FTL	trip-153761584139918815	IND421302
144303	training	2018-09-17 11:35:28.838714	thanos::sroute:d8f74492-4484-412a-887a-61c8e6b...	Carting	trip-153718412883843340	IND600056
144304	training	2018-09-17 11:35:28.838714	thanos::sroute:d8f74492-4484-412a-887a-61c8e6b...	Carting	trip-153718412883843340	IND600056
144305	training	2018-09-17 11:35:28.838714	thanos::sroute:d8f74492-4484-412a-887a-61c8e6b...	Carting	trip-153718412883843340	IND600056
144306	training	2018-09-17 11:35:28.838714	thanos::sroute:d8f74492-4484-412a-887a-61c8e6b...	Carting	trip-153718412883843340	IND600056
144307	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144308	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144309	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144310	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144311	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144312	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144313	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144314	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028
144315	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028

15 rows × 24 columns

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144316 entries, 0 to 144315
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  object
 1   trip_creation_time              144316 non-null  object
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  object
 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  is_cutoff                       144316 non-null  bool
 13  cutoff_factor                   144316 non-null  int64
 14  cutoff_timestamp                144316 non-null  object
 15  actual_distance_to_destination  144316 non-null  float64
 16  actual_time                     144316 non-null  float64
 17  osrm_time                       144316 non-null  float64
 18  osrm_distance                   144316 non-null  float64
 19  factor                          144316 non-null  float64
 20  segment_actual_time             144316 non-null  float64
 21  segment_osrm_time               144316 non-null  float64
 22  segment_osrm_distance           144316 non-null  float64
 23  segment_factor                  144316 non-null  float64
dtypes: bool(1), datetime64[ns](2), float64(10), int64(1), object(10)
memory usage: 25.5+ MB
```

In [18]:

```
# Display statistical summary
print("\nStatistical summary of numerical attributes:")
print(df.describe())
```

```
Statistical summary of numerical attributes:
                      od_start_time                   od_end_time  \
count                        144316                        144316
mean   2018-09-22 17:32:42.435769344  2018-09-23 09:36:54.057172224
min       2018-09-12 00:00:16.535741     2018-09-12 00:50:10.814399
25%    2018-09-17 07:37:35.014584832     2018-09-18 01:29:56.978912
50%    2018-09-22 07:35:23.038482944  2018-09-23 02:49:00.936600064
75%    2018-09-27 22:01:30.861209088  2018-09-28 12:13:41.675546112
max       2018-10-06 04:27:23.392375     2018-10-08 03:00:24.353479
std                              NaN                           NaN

       start_scan_to_end_scan  cutoff_factor  actual_distance_to_destination  \
count           144316.000000  144316.000000                   144316.000000
mean               963.697698     233.561345                      234.708498
min                 20.000000       9.000000                        9.000045
25%                161.000000      22.000000                       23.352027
50%                451.000000      66.000000                       66.135322
75%               1645.000000     286.000000                      286.919294
max               7898.000000    1927.000000                     1927.447705
std               1038.082976     345.245823                      345.480571

         actual_time      osrm_time  osrm_distance         factor  \
count  144316.000000  144316.000000  144316.000000  144316.000000
mean      417.996237     214.437055     285.549785       2.120178
min         9.000000       6.000000       9.008200       0.144000
25%        51.000000      27.000000      29.896250       1.604545
50%       132.000000      64.000000      78.624400       1.857143
75%       516.000000     259.000000     346.305400       2.212280
max      4532.000000    1686.000000    2326.199100      77.387097
std       598.940065     308.448543     421.717826       1.717065

       segment_actual_time  segment_osrm_time  segment_osrm_distance  \
count        144316.000000      144316.000000          144316.000000
mean             36.175379          18.495697              22.818993
min            -244.000000           0.000000               0.000000
25%              20.000000          11.000000              12.053975
50%              28.000000          17.000000              23.508300
75%              40.000000          22.000000              27.813325
max            3051.000000        1611.000000            2191.403700
std              53.524298          14.774008              17.866367

       segment_factor
count   144316.000000
mean         2.218707
min        -23.444444
25%          1.347826
50%          1.684211
75%          2.250000
max        574.250000
std          4.854804
```

1.4 Grouping by sub journey in the trip

In [20]:

```
df['Segment_key']=df['trip_uuid']+df['source_center']+df['destination_center']

segment_cols=['segment_actual_time','segment_osrm_time','segment_osrm_distance']

for col in segment_cols:
      df[col + '_sum'] = df.groupby('Segment_key')[col].cumsum()
df[[col + '_sum' for col in segment_cols]]
```

Out[20]:

	segment_actual_time_sum	segment_osrm_time_sum	segment_osrm_distance_sum
0	14.0	11.0	11.9653
1	24.0	20.0	21.7243
2	40.0	27.0	32.5395
3	61.0	39.0	45.5619
4	67.0	44.0	49.4772
...	...	...	...
144311	92.0	94.0	65.3487
144312	118.0	115.0	82.7212
144313	138.0	149.0	103.4265
144314	155.0	176.0	122.3150
144315	423.0	185.0	131.1238

144316 rows × 3 columns

In [21]:
```
create_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',

    }
```

1.5 Groupby mini-trips, sorting by time

In [23]:
```
segment=df.groupby('Segment_key').agg(create_segment_dict).reset_index()
segment=segment.sort_values(by=['Segment_key','od_end_time'], ascending=True).reset_inde
```

In [24]:
```
segment
```

Out[24]:

	index	Segment_key	data	trip_creation_time	route_sch
0	0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos::route a29b-
1	1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos::route a29b-
2	2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::route bb0b-
3	3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	thanos::route bb0b-
4	4	trip-153671043369099517IND000000ACBIND160002AAC	training	2018-09-12 00:00:33.691250	thanos::route 7641-
...	...	...	...	...	
26217	26217	trip-153861115439069069IND628204AAAIND627657AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26218	26218	trip-153861115439069069IND628613AAAIND627005AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26219	26219	trip-153861115439069069IND628801AAAIND628204AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26220	26220	trip-153861118270144424IND583119AAAIND583101AAA	test	2018-10-03 23:59:42.701692	thanos::srout 6d1f-
26221	26221	trip-153861118270144424IND583201AAAIND583119AAA	test	2018-10-03 23:59:42.701692	thanos::srout 6d1f-

26222 rows × 21 columns

In [25]:
```
segment[segment['trip_uuid']=='trip-153741093647649320']
```

Out[25]:

	index	Segment_key	data	trip_creation_time	route_sche
10370	10370	trip-153741093647649320IND388121AAAIND388620AAB	training	2018-09-20 02:35:36.476840	thanos::sroute b351-4
10371	10371	trip-153741093647649320IND388620AABIND388320AAA	training	2018-09-20 02:35:36.476840	thanos::sroute b351-4

2 rows × 21 columns

In [26]:
```
segment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26222 entries, 0 to 26221
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   index                           26222 non-null  int64
 1   Segment_key                     26222 non-null  object
 2   data                            26222 non-null  object
 3   trip_creation_time              26222 non-null  object
 4   route_schedule_uuid             26222 non-null  object
 5   route_type                      26222 non-null  object
 6   trip_uuid                       26222 non-null  object
 7   source_center                   26222 non-null  object
 8   source_name                     26222 non-null  object
 9   destination_center              26222 non-null  object
 10  destination_name                26222 non-null  object
 11  od_start_time                   26222 non-null  datetime64[ns]
 12  od_end_time                     26222 non-null  datetime64[ns]
 13  start_scan_to_end_scan          26222 non-null  float64
 14  actual_distance_to_destination  26222 non-null  float64
 15  actual_time                     26222 non-null  float64
 16  osrm_time                       26222 non-null  float64
 17  osrm_distance                   26222 non-null  float64
 18  segment_actual_time_sum         26222 non-null  float64
 19  segment_osrm_distance_sum       26222 non-null  float64
 20  segment_osrm_time_sum           26222 non-null  float64
dtypes: datetime64[ns](2), float64(8), int64(1), object(10)
memory usage: 4.2+ MB
```

1.6 Calculate time taken between od_start_time and od_end_time and keep it as a feature.

od_time_diff_hour is matching with start_scan_to_end_scan

In [28]:
```
segment['od_time_diff_mins']=(segment['od_end_time']-segment['od_start_time']).dt.total_
segment['od_time_diff_mins']
```

Out[28]:
```
0        1260.604421
1         999.505379
2          58.832388
3         122.779486
4         834.638929
             ...
26217      62.115193
26218      91.087797
26219      44.174403
26220     287.474007
26221      66.933565
Name: od_time_diff_mins, Length: 26222, dtype: float64
```

In [29]:
```
segment
```

Out[29]:

	index	Segment_key	data	trip_creation_time	route_sch
0	0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos::route a29b-
1	1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos::route a29b-
2	2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::route bb0b-
3	3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	thanos::route bb0b-
4	4	trip-153671043369099517IND000000ACBIND160002AAC	training	2018-09-12 00:00:33.691250	thanos::route 7641-
...	...	...	...	...	
26217	26217	trip-153861115439069069IND628204AAAIND627657AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26218	26218	trip-153861115439069069IND628613AAAIND627005AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26219	26219	trip-153861115439069069IND628801AAAIND628204AAA	test	2018-10-03 23:59:14.390954	thanos::srout 8486-
26220	26220	trip-153861118270144424IND583119AAAIND583101AAA	test	2018-10-03 23:59:42.701692	thanos::srout 6d1f-
26221	26221	trip-153861118270144424IND583201AAAIND583119AAA	test	2018-10-03 23:59:42.701692	thanos::srout 6d1f-

26222 rows × 22 columns

In [30]:

```
create_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',

    'od_time_diff_mins' : 'sum',
    'start_scan_to_end_scan' : 'sum',

    'actual_distance_to_destination' : 'sum',
```

```
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',

    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum',

    }
```

1.7 Groupby trip

In [32]:
```
trip=segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop=True)
```

In [33]:
```
trip
```

Out[33]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cen
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304A
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203A
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000A
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072A
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583101A
...	...	...	...	...	...	
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND160002A
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND121004A
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND208006A
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND627005A
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND583119A

14787 rows × 18 columns

In [34]:
```
trip[['actual_time','segment_actual_time_sum']]
```

Out[34]:

	actual_time	segment_actual_time_sum
0	1562.0	1548.0
1	143.0	141.0
2	3347.0	3308.0
3	59.0	59.0
4	341.0	340.0
...	...	...
14782	83.0	82.0
14783	21.0	21.0
14784	282.0	281.0
14785	264.0	258.0
14786	275.0	274.0

14787 rows × 2 columns

In [35]:
```
trip[trip['trip_uuid']=='trip-153741093647649320']
```

Out[35]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cente
5917	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AA

In [36]:
```
trip[['actual_distance_to_destination','osrm_distance']]
```

Out[36]:

	actual_distance_to_destination	osrm_distance
0	824.732854	991.3523
1	73.186911	85.1110
2	1927.404273	2354.0665
3	17.175274	19.6800
4	127.448500	146.7918
...	...	...
14782	57.762332	73.4630
14783	15.513784	16.0882
14784	38.684839	58.9037
14785	134.723836	171.1103
14786	66.081533	80.5787

14787 rows × 2 columns

In [37]:
```
trip[['actual_time','osrm_time']]
```

Out[37]:

	actual_time	osrm_time
0	1562.0	717.0
1	143.0	68.0
2	3347.0	1740.0
3	59.0	15.0
4	341.0	117.0
...	...	...
14782	83.0	62.0
14783	21.0	12.0
14784	282.0	48.0
14785	264.0	179.0
14786	275.0	68.0

14787 rows × 2 columns

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

Out[38]:

	od_time_diff_mins	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_ti
count	14787.000000	14787.000000	14787.000000	14787.000000	14787.000(
mean	530.313517	529.429025	164.090196	356.306012	160.990(
std	658.415490	658.254936	305.502982	561.517936	271.459-
min	23.461468	23.000000	9.002461	9.000000	6.000(
25%	149.698496	149.000000	22.777099	67.000000	29.000(
50%	279.710750	279.000000	48.287894	148.000000	60.000(
75%	633.537697	632.000000	163.591258	367.000000	168.000(
max	7898.551955	7898.000000	2186.531787	6265.000000	2032.000(

2. Feature Creation

In [40]:
```
#lowering all columns

trip['destination_name'] = trip['destination_name'].str.lower()
trip['source_name'] = trip['source_name'].str.lower()
```

2.1 Split and extract features out of Destination & Source columns

In [42]:
```
def place2state(x):
    if x is None:
        return None
    # transform  "gurgaon_bilaspur_hb (haryana)" into "haryana)""
```

```
    state = x.split('(')[1]

    return state[:-1] #removing ')' from ending

def place2city(x):
    if x is None:
        return None
    # We will remove state
    city = x.split(' (')[0]

    city = city.split('_')[0]

    #Now dealing with edge cases

    if city == 'pnq vadgaon sheri dpc':
      return 'vadgaonsheri'

    # ['PNQ Pashan DPC', 'Bhopal MP Nagar', 'HBR Layout PC',
    #  'PNQ Rahatani DPC', 'Pune Balaji Nagar', 'Mumbai Antop Hill']

    if city in ['pnq pashan dpc','pnq rahatani dpc', 'pune balaji nagar']:
        return 'pune'

    if city == 'hbr layout pc' : return 'bengaluru'
    if city == 'bhopal mp nagar' : return 'bhopal'
    if city == 'mumbai antop hill' : return 'mumbai'


    return city

def place2city_place(x):
    if x is None:
        return None

    # We will remove state
    x = x.split(' (')[0]

    len_ = len(x.split('_'))

    if len_ >= 3:
        return x.split('_')[1]

    # Small cities have same city and place name
    if len_ == 2:
        return x.split('_')[0]


    # Now we need to deal with edge cases or imporper name convention

    #if len(x.split(' ')) == 2:
    #

    return x.split(' ')[0]


def place2code(x):
    if x is None:
        return None
    # We will remove state
    x = x.split(' (')[0]

    if len(x.split('_')) >= 3 :
        return x.split('_')[-1]
```

```
    return 'none'
```

In [43]:
```
trip['destination_state'] = trip['destination_name'].apply(lambda x: place2state(x))
trip['destination_city']  = trip['destination_name'].apply(lambda x: place2city(x))
trip['destination_place'] = trip['destination_name'].apply(lambda x: place2city_place(x
trip['destination_code']  = trip['destination_name'].apply(lambda x: place2code(x))
```

In [44]:
```
trip[['destination_state', 'destination_city', 'destination_place', 'destination_code']]
```

Out[44]:

	destination_state	destination_city	destination_place	destination_code
0	uttar pradesh	kanpur	central	6
1	karnataka	doddablpur	chikadpp	d
2	haryana	gurgaon	bilaspur	hb
3	maharashtra	mumbai	mirard	ip
4	karnataka	sandur	wrdn1dpp	d
...	...	...	...	...
14782	punjab	chandigarh	mehmdpur	h
14783	haryana	faridabad	blbgarh	dc
14784	uttar pradesh	kanpur	govndngr	dc
14785	tamil nadu	tirchchndr	shnmgprm	d
14786	karnataka	sandur	wrdn1dpp	d

14787 rows × 4 columns

In [45]:
```
trip['source_state'] = trip['source_name'].apply(lambda x: place2state(x))
trip['source_city']  = trip['source_name'].apply(lambda x: place2city(x))
trip['source_place'] = trip['source_name'].apply(lambda x: place2city_place(x))
trip['source_code']  = trip['source_name'].apply(lambda x: place2code(x))
```

In [46]:
```
trip[['source_state', 'source_city', 'source_place', 'source_code']]
```

Out[46]:

	source_state	source_city	source_place	source_code
0	uttar pradesh	kanpur	central	6
1	karnataka	doddablpur	chikadpp	d
2	haryana	gurgaon	bilaspur	hb
3	maharashtra	mumbai hub	mumbai	none
4	karnataka	bellary	bellary	none
...	...	...	...	...
14782	punjab	chandigarh	mehmdpur	h
14783	haryana	fbd	balabhgarh	dpc
14784	uttar pradesh	kanpur	govndngr	dc
14785	tamil nadu	tirunelveli	vdkkusrt	i
14786	karnataka	sandur	wrdn1dpp	d

14787 rows × 4 columns

2.2 Extract features like month, year, day, etc from 'Trip_creation_time'

In [48]:
```
trip['trip_creation_time'] =  pd.to_datetime(trip['trip_creation_time'])

trip['trip_year'] = trip['trip_creation_time'].dt.year
trip['trip_month'] = trip['trip_creation_time'].dt.month
trip['trip_hour'] = trip['trip_creation_time'].dt.hour
trip['trip_day'] = trip['trip_creation_time'].dt.day
trip['trip_day_name'] = trip['trip_creation_time'].dt.day_name()
trip['trip_week'] = trip['trip_creation_time'].dt.isocalendar().week
trip['trip_dayofweek'] = trip['trip_creation_time'].dt.dayofweek
```

In [49]:
```
trip[['trip_year', 'trip_month', 'trip_hour', 'trip_day', 'trip_week', 'trip_dayofweek'
```

Out[49]:

	trip_year	trip_month	trip_hour	trip_day	trip_week	trip_dayofweek	trip_day_name
0	2018	9	0	12	37	2	Wednesday
1	2018	9	0	12	37	2	Wednesday
2	2018	9	0	12	37	2	Wednesday
3	2018	9	0	12	37	2	Wednesday
4	2018	9	0	12	37	2	Wednesday
...	...	...	...	...	...	...	...
14782	2018	10	23	3	40	2	Wednesday
14783	2018	10	23	3	40	2	Wednesday
14784	2018	10	23	3	40	2	Wednesday
14785	2018	10	23	3	40	2	Wednesday
14786	2018	10	23	3	40	2	Wednesday

14787 rows × 7 columns

In [50]:
```
trip.head()
```

Out[50]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072AAB
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583101AAA

5 rows × 33 columns

3. Find outliers in numerical & categorical variables

(You might find outliers in almost all the variables), and visualize it using visual analysis.

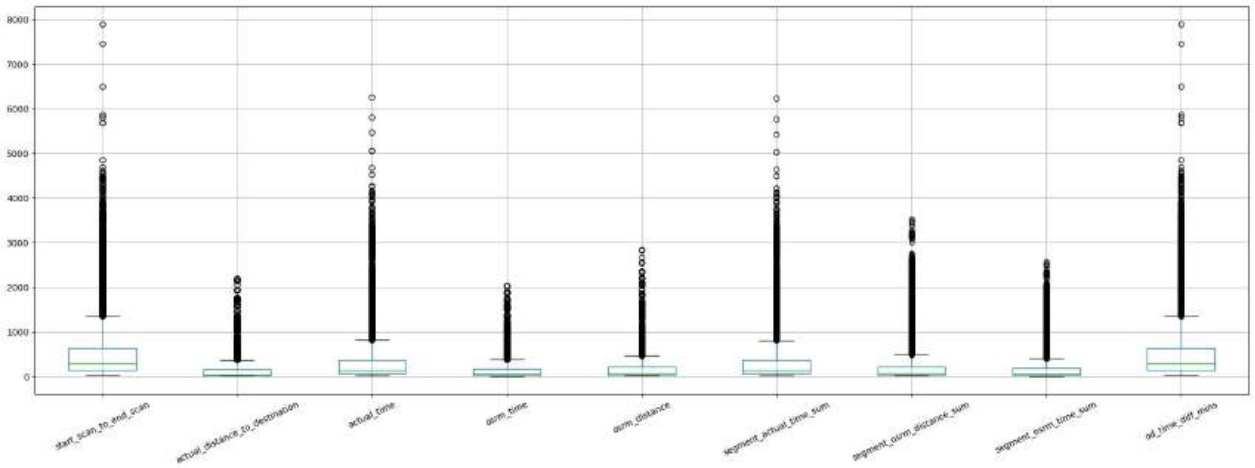
3.1 Find outliers in numerical variables

In [53]:
```
num_cols = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time',
            'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_distance_sum',
```

```
            'segment_osrm_time_sum', 'od_time_diff_mins']
```

In [54]:
```
trip[num_cols].boxplot(rot=25, figsize=(25,8))
```

Out[54]:
```
<Axes: >
```



3.2 Handle the outliers using IQR method

In [56]:
```
Q1 = trip[num_cols].quantile(0.25)
Q3 = trip[num_cols].quantile(0.75)

IQR = Q3 - Q1
```

In [57]:
```
trip = trip[~((trip[num_cols] < (Q1 - 1.5 * IQR)) | (trip[num_cols] > (Q3 + 1.5 * IQR))
trip = trip.reset_index(drop=True)
# ---- and or or
```

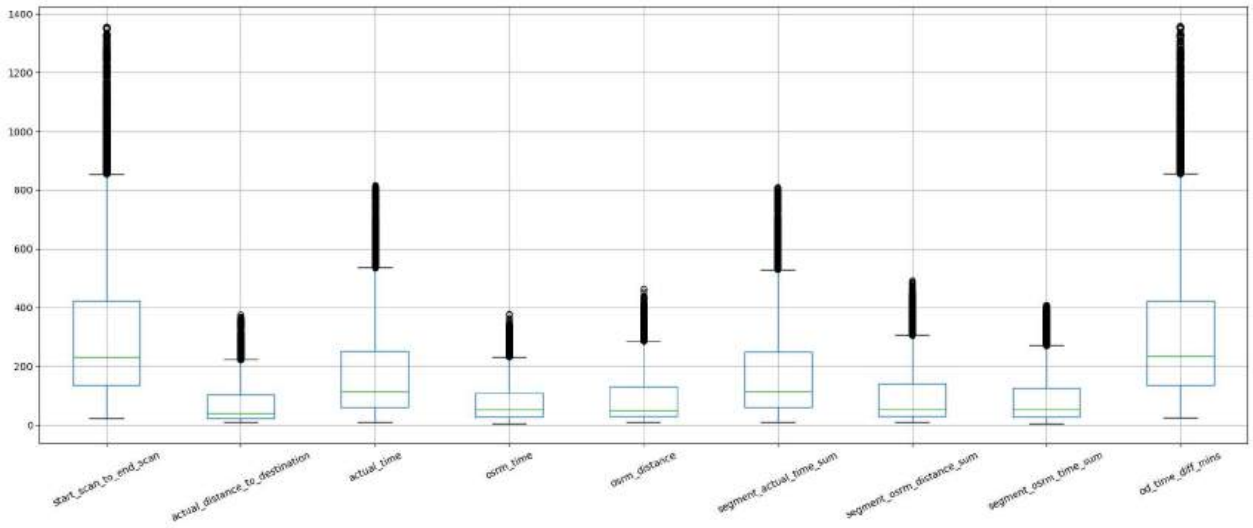
In [58]:
```
trip
```

Out[58]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cer
0	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203A
1	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072A
2	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583101A
3	training	2018-09-12 00:02:34.161600	thanos::sroute:9bf03170-d0a2-4a3f-aa4d-9aaab3d...	Carting	trip-153671055416136166	IND600056A
4	training	2018-09-12 00:04:22.011653	thanos::sroute:a97698cc-846e-41a7-916b-88b1741...	Carting	trip-153671066201138152	IND600044A
...	...	...	...	...	...	
12718	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND160002A
12719	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND121004A
12720	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND208006A
12721	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND627005A
12722	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND583119A

12723 rows × 33 columns

In [59]:
```
trip[num_cols].boxplot(rot=25, figsize=(22,8))
plt.show()
```



3.3 Handling Categorical Variables

3.3.1 Only two route_type – Do one hot encoding

In [62]:
```
trip['route_type'].value_counts()
```

Out[62]:
```
route_type
Carting    8812
FTL        3911
Name: count, dtype: int64
```

In [63]:
```
# One-hot encoding of categorical variables
trip['route_type'] = trip['route_type'].map({'FTL':0, 'Carting':1})
```

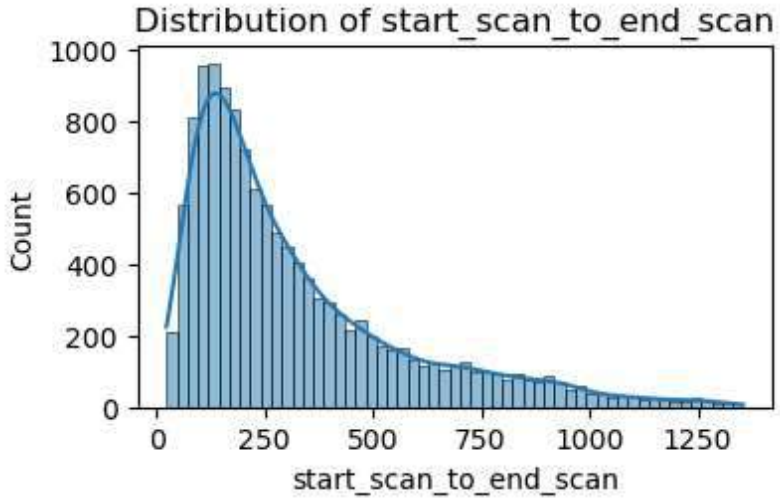
In [64]:
```
trip['route_type'].value_counts()
```

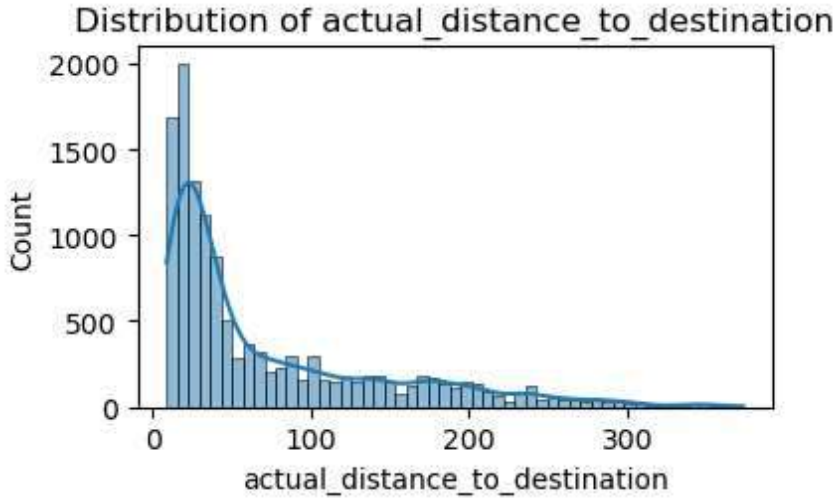
Out[64]:
```
route_type
1    8812
0    3911
Name: count, dtype: int64
```

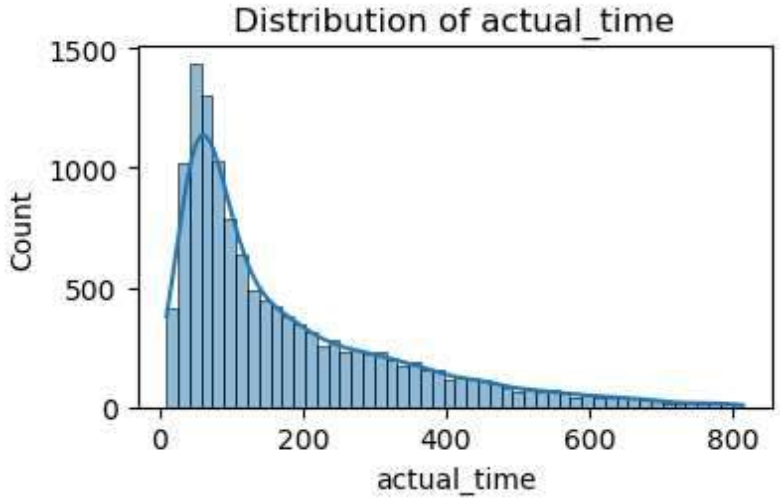
4. Visual Analysis - Univariate & Bivariate

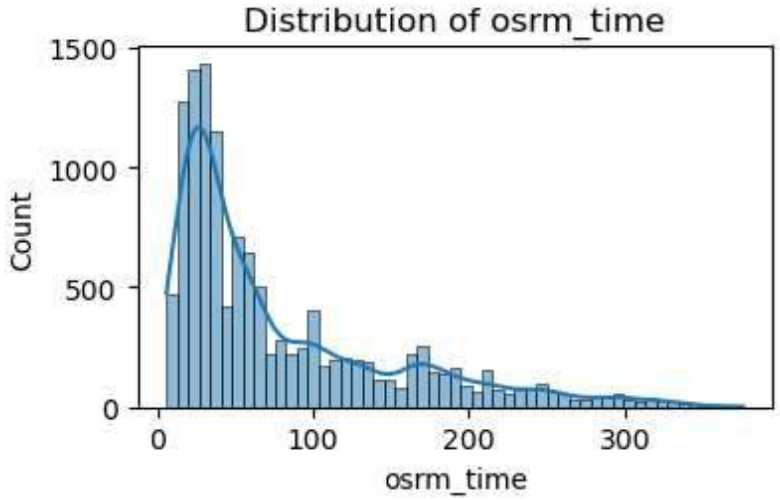
4.1 Univariate Analysis For numerical variables

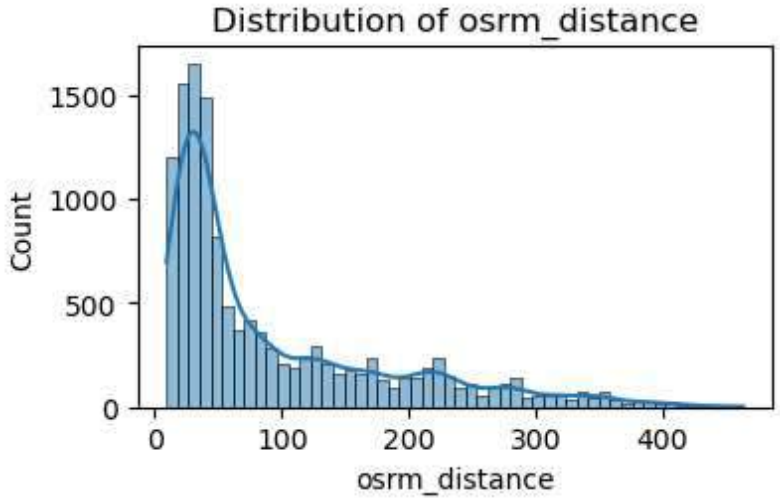
In [66]:
```
# Univariate Analysis for Numerical Variables

# Histogram and KDE for numerical variables
for i, var in enumerate(num_cols):
    plt.figure(figsize=(14, 8))
    plt.subplot(3, 3, i+1)
    sns.histplot(trip[var], kde=True)
    plt.title(f'Distribution of {var}')
plt.tight_layout()
plt.show()
```

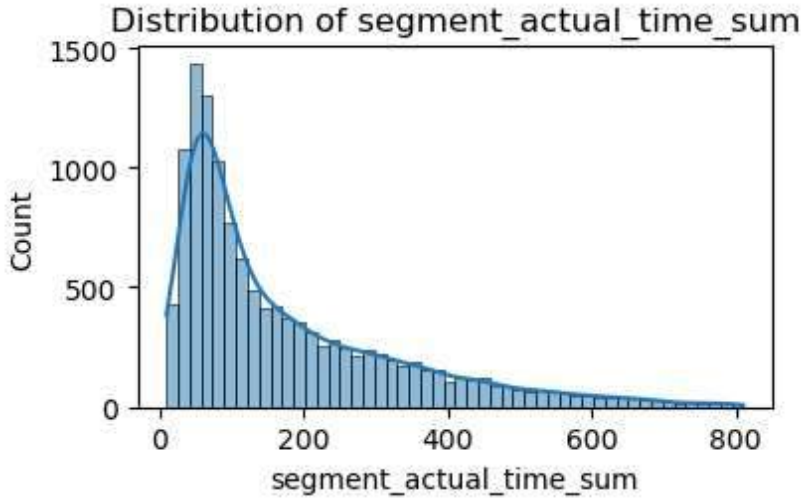

Distribution of start_scan_to_end_scan
Count
start_scan_to_end_scan
0
200
400
600
800
1000
0
250
500
750
1000
1250

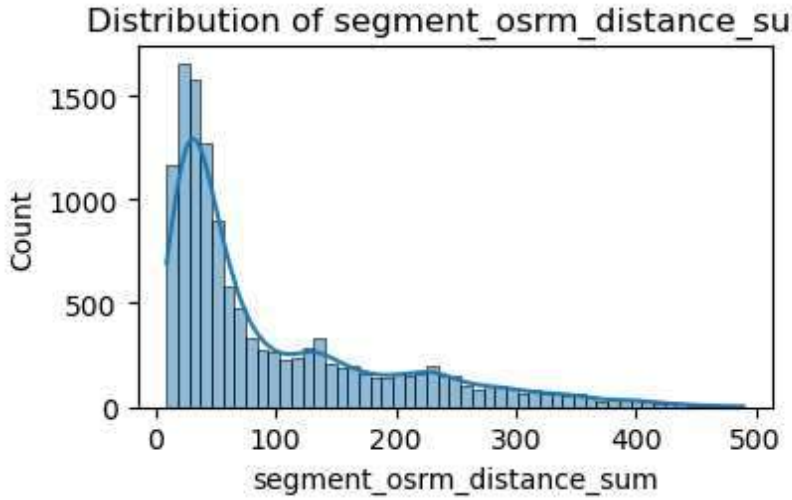

Distribution of actual_distance_to_destination
Count
actual_distance_to_destination
0
500
1000
1500
2000
0
100
200
300

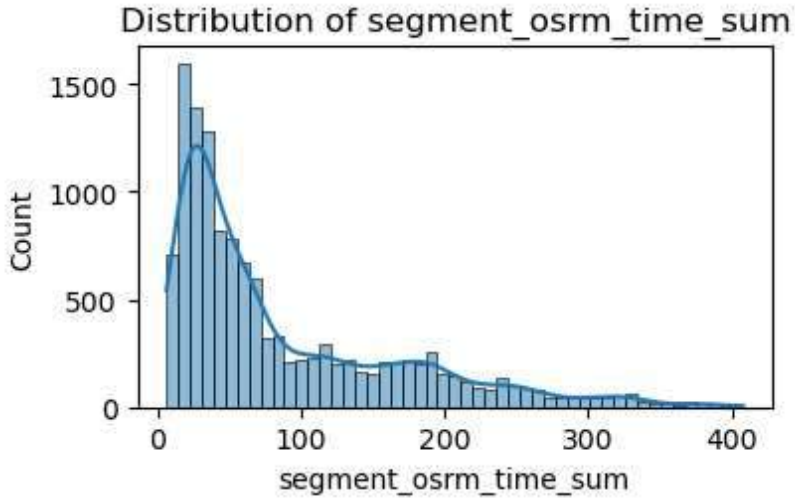

Distribution of actual_time
Count
actual_time
0
500
1000
1500
0
200
400
600
800

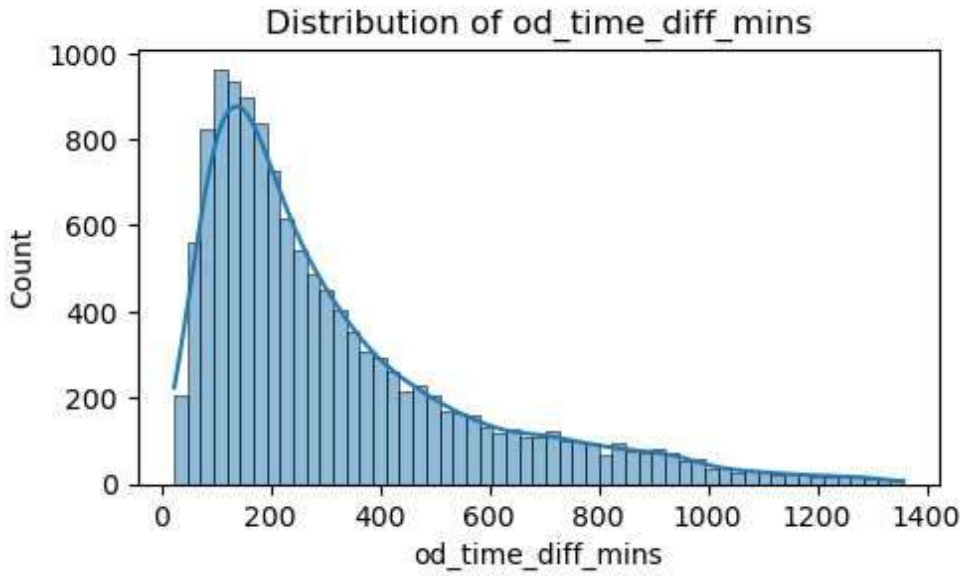

Distribution of osrm_time
Count
1500
1000
500
0
0
100
200
300
osrm_time


Distribution of osrm_distance
Count
1500
1000
500
0
0
100
200
300
400
osrm_distance


Distribution of segment_actual_time_sum
Count
1500
1000
500
0
0
200
400
600
800
segment_actual_time_sum

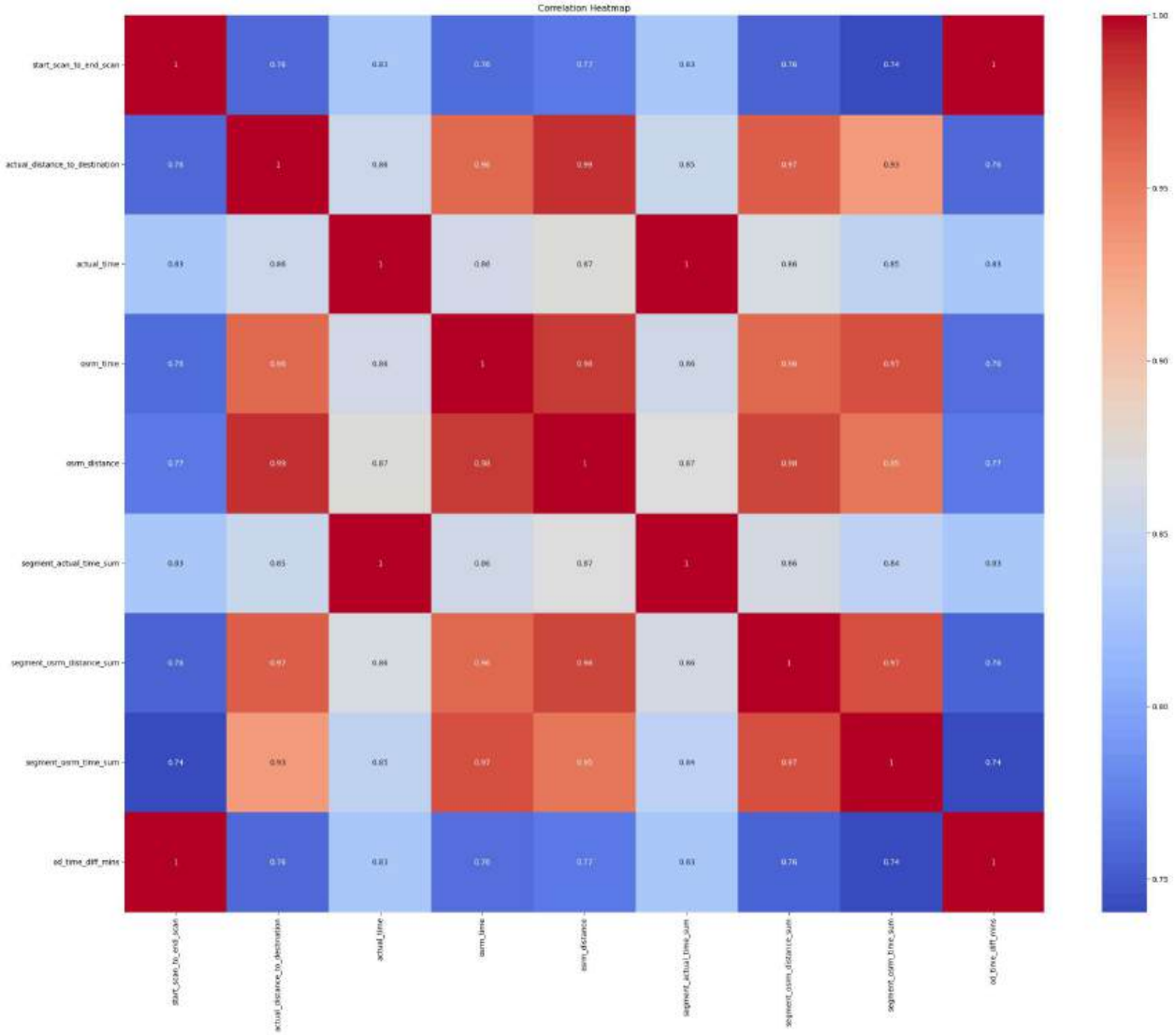






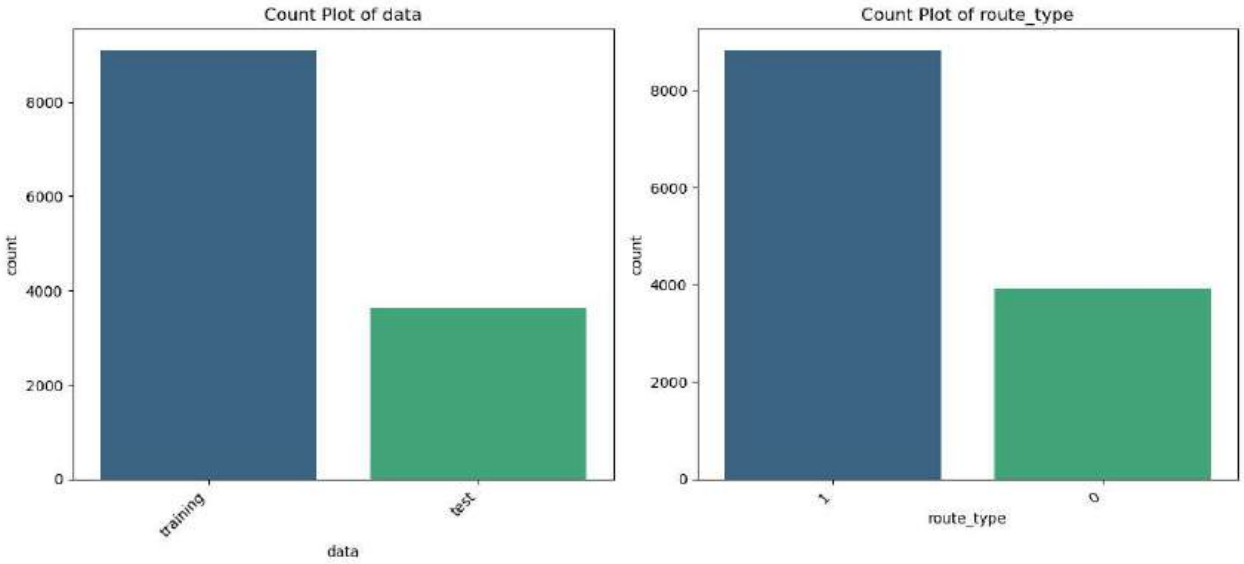
4.2 Correlation Heatmap for Numerical Variables

In [68]:
```
# Correlation Heatmap
plt.figure(figsize=(28, 22))
sns.heatmap(trip[num_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



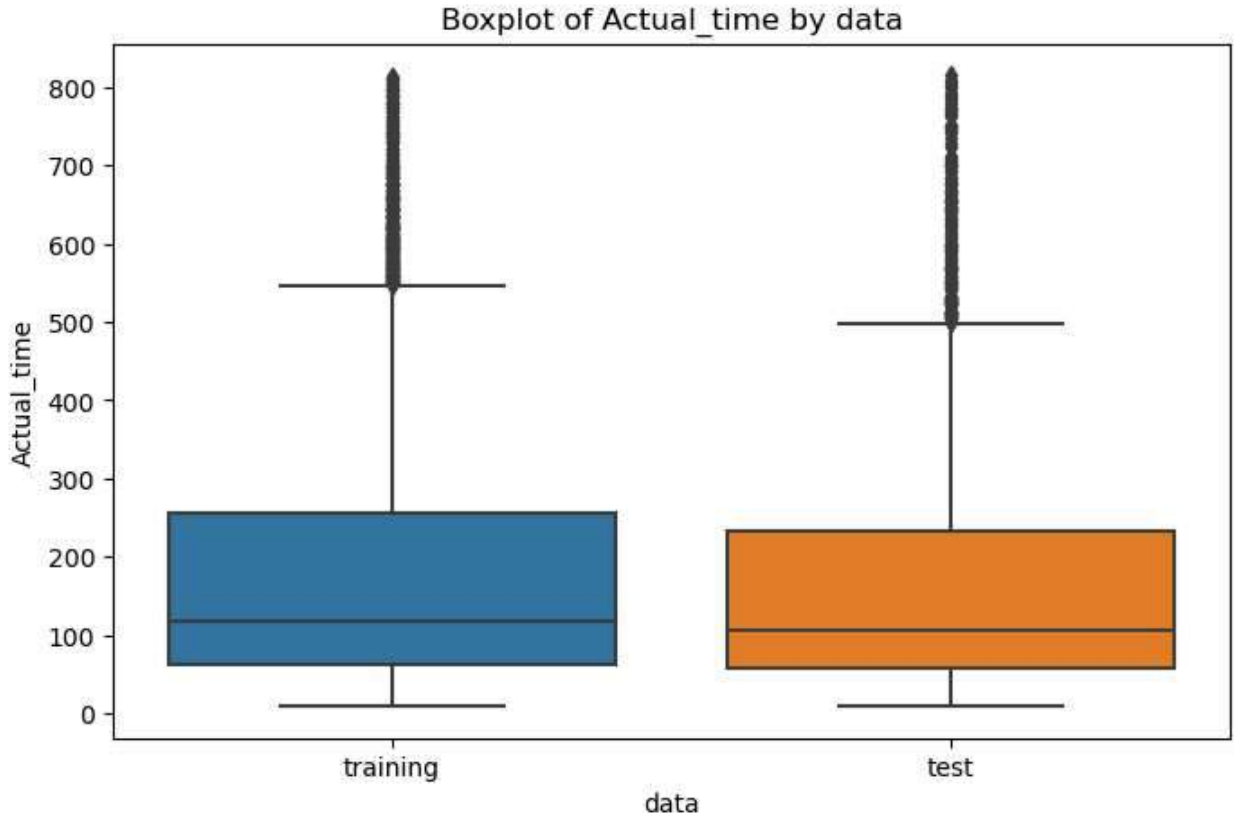
4.3 Univariate Analysis for Categorical Variables

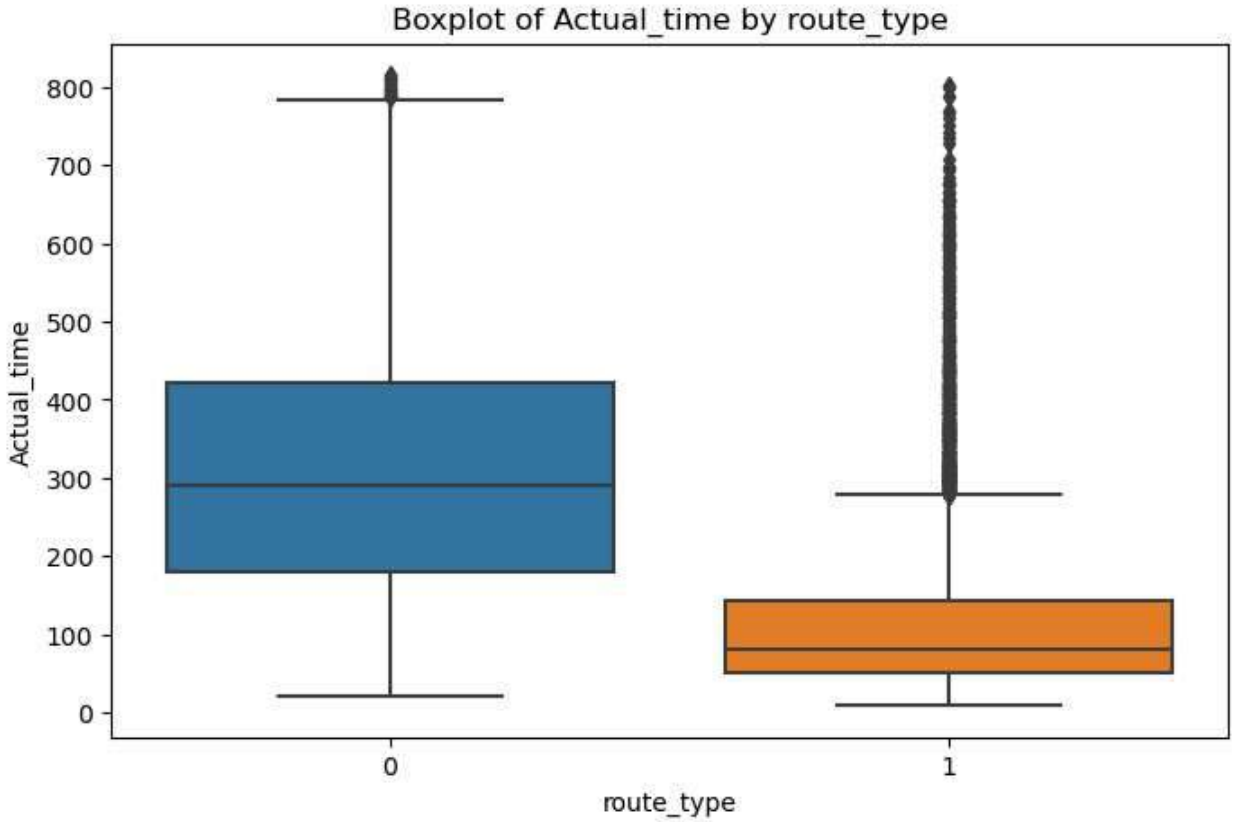
In [70]:
```
# Univariate Analysis of Categorical Variables

cat_cols = ['data', 'route_type']

# Plotting bar plots for categorical variables
plt.figure(figsize=(18, 10))
for i, var in enumerate(cat_cols):
    plt.subplot(2, 3, i+1)
    sns.countplot(x=trip[var], order=trip[var].value_counts().index, palette='viridis')
    plt.xticks(rotation=45, ha='right')
    plt.title(f'Count Plot of {var}')
plt.tight_layout()
plt.show()
```



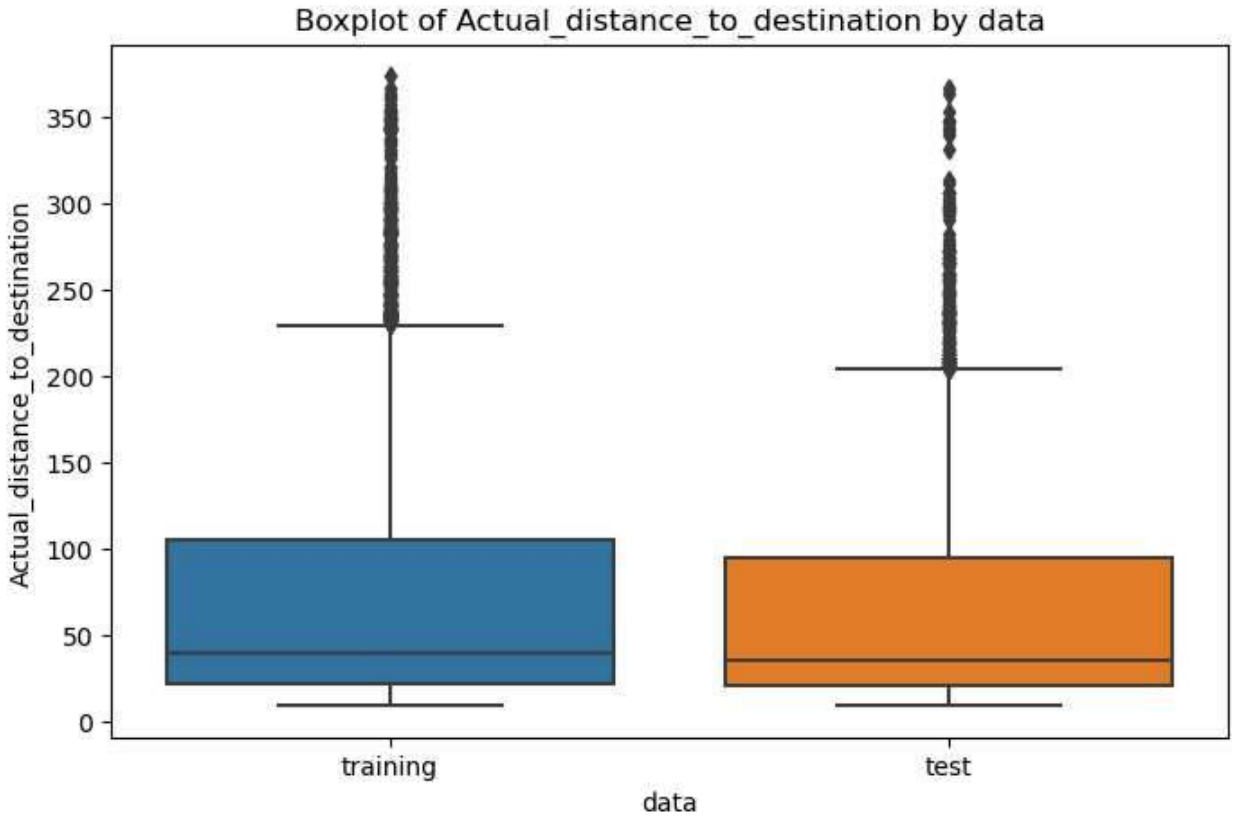
4.4 Bivariate Analysis for Categorical Variables

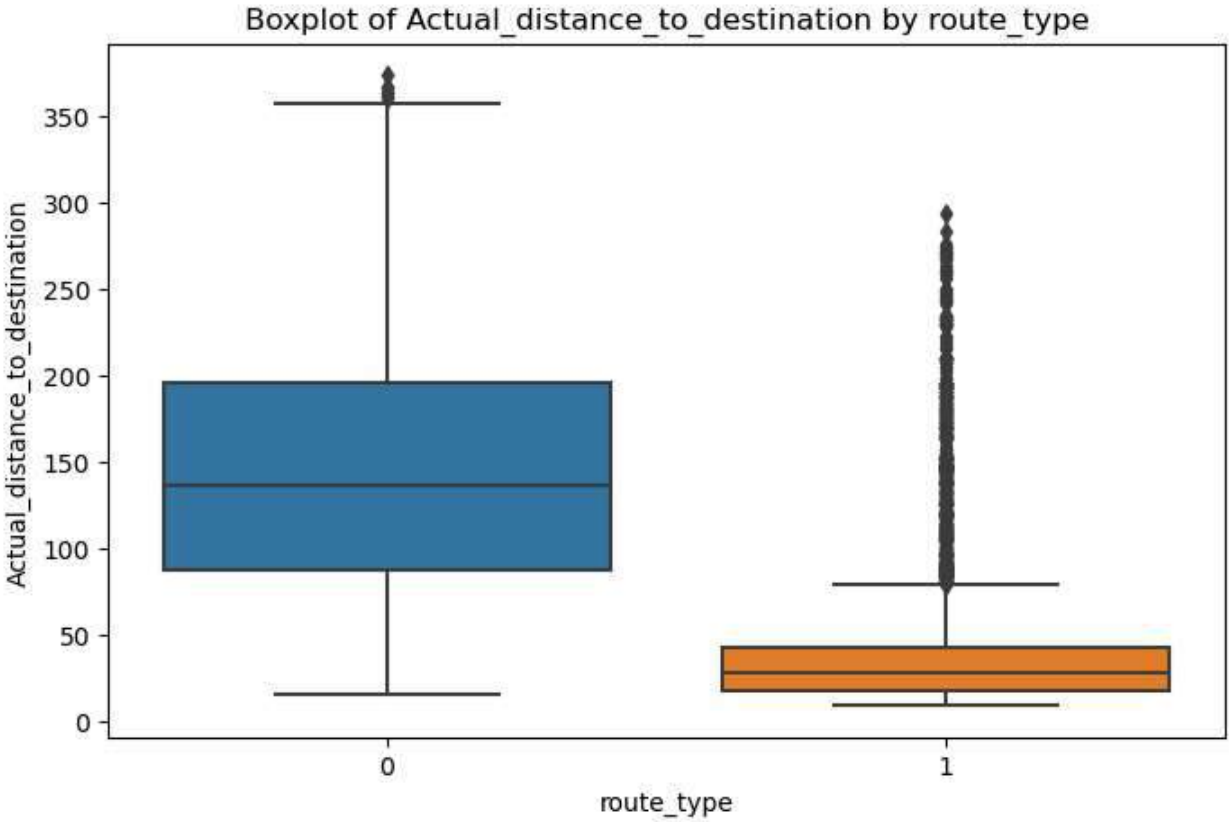
In [72]:
```
# Bivariate Analysis of Categorical Variables

for var in cat_cols:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=var, y='actual_time', data=trip)
    plt.title(f'Boxplot of Actual_time by {var}')
    plt.xlabel(var)
    plt.ylabel('Actual_time')
    plt.show()
```





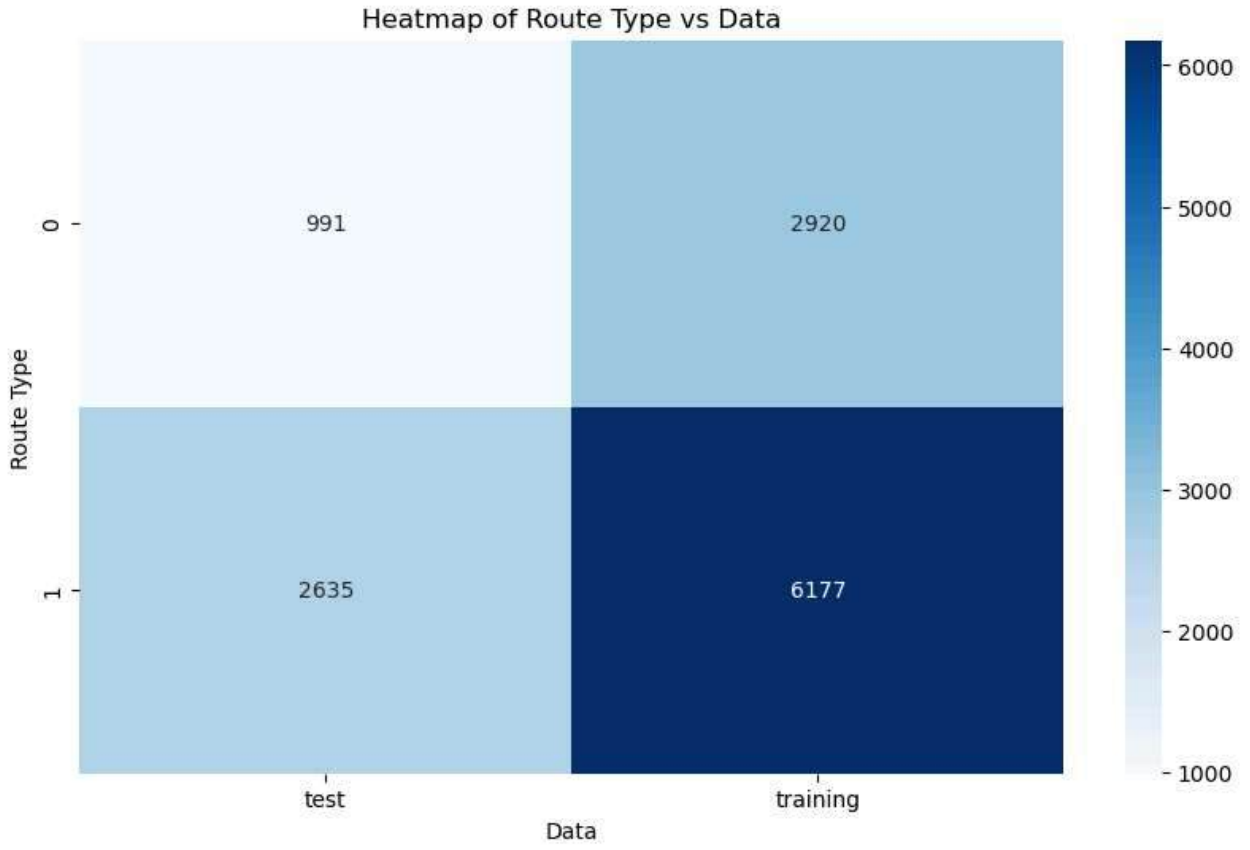
In [73]:
```
# Bivariate Analysis of Categorical Variables

for var in cat_cols:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=var, y='actual_distance_to_destination', data=trip)
    plt.title(f'Boxplot of Actual_distance_to_destination by {var}')
    plt.xlabel(var)
    plt.ylabel('Actual_distance_to_destination')
    plt.show()
```





4.5 Correlation Heatmap for Categorical Variables

In [75]:
```
crosstab = pd.crosstab(trip['route_type'], trip['data'])

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(crosstab, annot=True, fmt='d', cmap='Blues')
plt.title('Heatmap of Route Type vs Data')
plt.xlabel('Data')
plt.ylabel('Route Type')
plt.show()
```



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

In [77]:
```
scaler = MinMaxScaler()
trip[num_cols] = scaler.fit_transform(trip[num_cols])
```

In [78]:
```
trip[num_cols]
```

Out[78]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	seg
0	0.117868	0.176119	0.166253	0.167568	0.167335	
1	0.057808	0.022426	0.062035	0.024324	0.023343	
2	0.521021	0.325009	0.411911	0.300000	0.303075	
3	0.124625	0.042791	0.064516	0.045946	0.041795	
4	0.056306	0.000269	0.018610	0.018919	0.006482	
...	...	...	...	...	...	
12718	0.175676	0.133794	0.091811	0.151351	0.141702	
12719	0.027778	0.017867	0.014888	0.016216	0.015438	
12720	0.298799	0.081447	0.338710	0.113514	0.109662	
12721	0.243243	0.344973	0.316377	0.467568	0.356592	
12722	0.247748	0.156622	0.330025	0.167568	0.157361	

12723 rows × 9 columns

In [79]:
```
trip[num_cols].describe()
```

Out[79]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance
count	12723.000000	12723.000000	12723.000000	12723.000000	12723.000000
mean	0.223107	0.173734	0.208998	0.195785	0.181911
std	0.191859	0.197757	0.196217	0.195496	0.197107
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.084835	0.034006	0.064516	0.056757	0.042410
50%	0.157658	0.081009	0.130273	0.118919	0.086587
75%	0.300300	0.254284	0.300248	0.278378	0.269020
max	1.000000	1.000000	1.000000	1.000000	1.000000

In [80]:
```
trip[['start_scan_to_end_scan','od_time_diff_mins']]
```

Out[80]:

	start_scan_to_end_scan	od_time_diff_mins
0	0.117868	0.118559
1	0.057808	0.057749
2	0.521021	0.520930
3	0.124625	0.125213
4	0.056306	0.055883
...	...	...
12718	0.175676	0.175846
12719	0.027778	0.027834
12720	0.298799	0.298859
12721	0.243243	0.243678
12722	0.247748	0.248097

12723 rows × 2 columns

In [81]:
```
trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12723 entries, 0 to 12722
Data columns (total 33 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            12723 non-null  object
 1   trip_creation_time              12723 non-null  datetime64[ns]
 2   route_schedule_uuid             12723 non-null  object
 3   route_type                      12723 non-null  int64
 4   trip_uuid                       12723 non-null  object
 5   source_center                   12723 non-null  object
 6   source_name                     12723 non-null  object
 7   destination_center              12723 non-null  object
 8   destination_name                12723 non-null  object
 9   od_time_diff_mins               12723 non-null  float64
 10  start_scan_to_end_scan          12723 non-null  float64
 11  actual_distance_to_destination  12723 non-null  float64
 12  actual_time                     12723 non-null  float64
 13  osrm_time                       12723 non-null  float64
 14  osrm_distance                   12723 non-null  float64
 15  segment_actual_time_sum         12723 non-null  float64
 16  segment_osrm_distance_sum       12723 non-null  float64
 17  segment_osrm_time_sum           12723 non-null  float64
 18  destination_state               12723 non-null  object
 19  destination_city                12723 non-null  object
 20  destination_place               12723 non-null  object
 21  destination_code                12723 non-null  object
 22  source_state                    12723 non-null  object
 23  source_city                     12723 non-null  object
 24  source_place                    12723 non-null  object
 25  source_code                     12723 non-null  object
 26  trip_year                       12723 non-null  int32
 27  trip_month                      12723 non-null  int32
 28  trip_hour                       12723 non-null  int32
 29  trip_day                        12723 non-null  int32
 30  trip_day_name                   12723 non-null  object
 31  trip_week                       12723 non-null  UInt32
 32  trip_dayofweek                  12723 non-null  int32
dtypes: UInt32(1), datetime64[ns](1), float64(9), int32(5), int64(1), object(16)
memory usage: 2.9+ MB
```

6. Hypothesis Testing

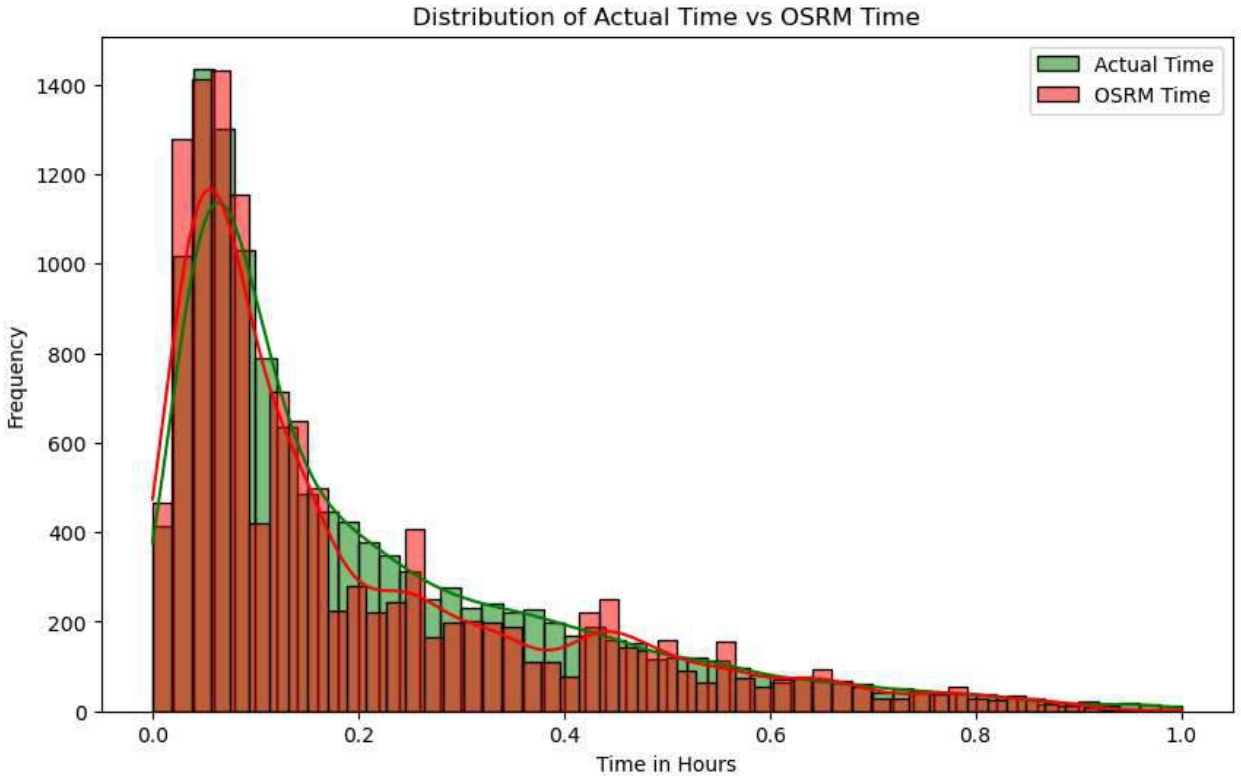
6.1 Perform Hypothesis testing (paired t-test) and visual analysis (histplot) between all the numerical variables

a. actual_time aggregated value and OSRM time aggregated value.

In [85]:

```
# Visual analysis and hypothesis testing between actual_time and osrm_time
plt.figure(figsize=(10, 6))
sns.histplot(trip['actual_time'], color='green', label='Actual Time', kde=True)
sns.histplot(trip['osrm_time'], color='red', label='OSRM Time', kde=True)
plt.xlabel('Time in Hours')
plt.ylabel('Frequency')
plt.title('Distribution of Actual Time vs OSRM Time')
plt.legend()
plt.show()

# Hypothesis testing between actual_time and osrm_time
```

```
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['osrm_time'])
print(f"T-test between actual_time and osrm_time: t-statistic = {t_stat:.2f}, p-value =

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("This implies that the open-source routing engine is inefficient in calculatin
    print("Recommendations: Consider changing OSRM Engine/or look for some faults in OSR
else:
    print("Inferences: There is not enough evidence to suggest that there is no signific
    print("This implies efficiency of OSRM Engine in calculating Estimated overall time
    print("Recommendations: Continue monitoring OSRM Time patterns to see if any trends
```



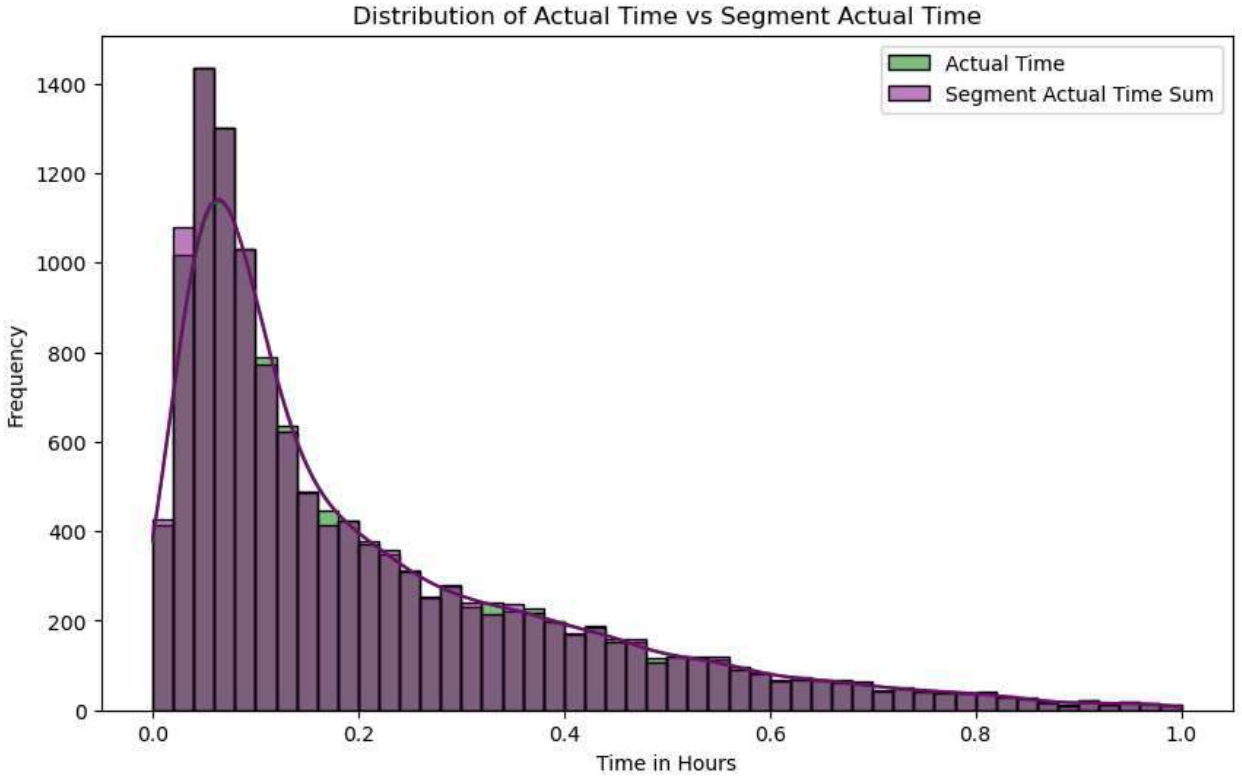

```
T-test between actual_time and osrm_time: t-statistic = 14.36, p-value = 0.0000
Reject Null Hypothesis: There is a significant difference between the aggregated actual
time and OSRM time.
Inferences: There is enough evidence to suggest that there is a significant difference
between the aggregated actual time and OSRM time.
This implies that the open-source routing engine is inefficient in calculating estimate
d time taken to complete the delivery
Recommendations: Consider changing OSRM Engine/or look for some faults in OSRM Engine
```

b. actual_time aggregated value and segment actual time aggregated value.

In [87]:

```
# Visual analysis and hypothesis testing between actual_time and segment_actual_time_sum
plt.figure(figsize=(10, 6))
sns.histplot(trip['actual_time'], color='green', label='Actual Time', kde=True)
sns.histplot(trip['segment_actual_time_sum'], color='purple', label='Segment Actual Time
plt.xlabel('Time in Hours')
```

```
plt.ylabel('Frequency')
plt.title('Distribution of Actual Time vs Segment Actual Time')
plt.legend()
plt.show()

# Hypothesis testing between actual_time and segment_actual_time_sum
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['segment_actual_time_sum'])
print(f"T-test between actual_time and segment_actual_time: t-statistic = {t_stat:.2f},

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("This indicates that there is inefficieny in operations i.e. external factors
    print("Recommendations: I) Consider changing the delivery route based on the least a
    print("II) Consider taking less parcels or increasing the efficiency of truck loadin
else:
    print("Inferences: There is not enough evidence to suggest that there is any signifi
    print("This indicates great efficiency in operations")
    print("Recommendations: Continue monitoring aggregated actual time and segment actua
```

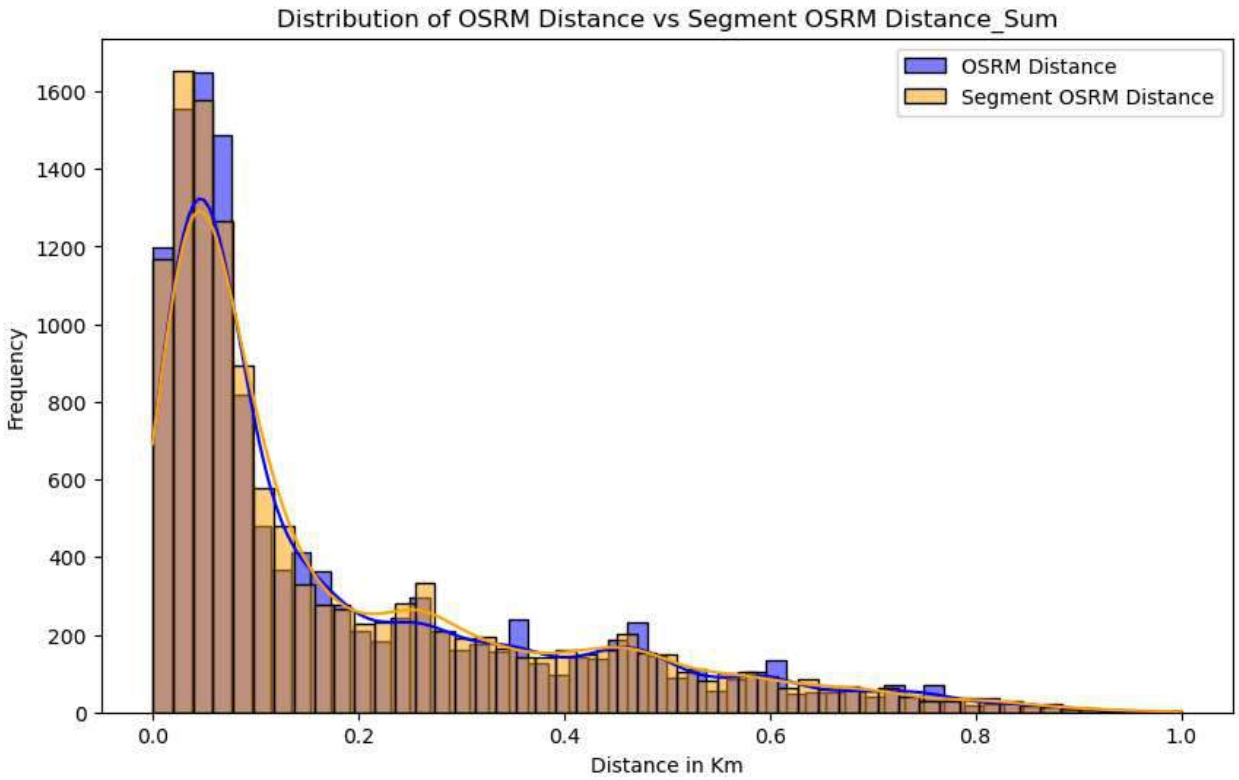


```
T-test between actual_time and segment_actual_time: t-statistic = 49.76, p-value = 0.00
00
Reject Null Hypothesis: There is a significant difference between the aggregated actual
time and segment actual time sum.
Inferences: There is enough evidence to suggest that there is a significant difference
between the aggregated actual time and segment actual time sum.
This indicates that there is inefficieny in operations i.e. external factors such as de
lays(due to traffic), load/unload time are affecting the delivery
Recommendations: I) Consider changing the delivery route based on the least amount of t
raffic, or changing time of delivery to when there is less traffic
II) Consider taking less parcels or increasing the efficiency of truck loading/unloadin
g to improve the pace of delivery
```

c. OSRM distance aggregated value and segment OSRM distance aggregated value

In [89]:

```
# Visual analysis and hypothesis testing between osrm_distance and segment_osrm_distance
plt.figure(figsize=(10, 6))
sns.histplot(trip['osrm_distance'], color='blue', label='OSRM Distance', kde=True)
sns.histplot(trip['segment_osrm_distance_sum'], color='orange', label='Segment OSRM Dist
plt.xlabel('Distance in Km')
plt.ylabel('Frequency')
plt.title('Distribution of OSRM Distance vs Segment OSRM Distance_Sum')
plt.legend()
plt.show()

# Hypothesis testing between osrm_distance and segment_osrm_distance
t_stat, p_value = stats.ttest_rel(trip['osrm_distance'], trip['segment_osrm_distance_sur
print(f"T-test between osrm_distance and segment_osrm_distance: t-statistic = {t_stat:.2

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("This implies that the open-source routing engine is inefficient in calculatir
    print("total distance covered(excluding external factors) to complete the delivery")
    print("Recommendations: Consider changing OSRM Engine/or look for some faults in OSR
else:
    print("Inferences: There is not enough evidence to suggest that there is a significa
    print("This implies efficiency of OSRM Engine in calculating Estimated overall dista
    print("Recommendations: Continue monitoring OSRM distance to see if any trends emerg
```



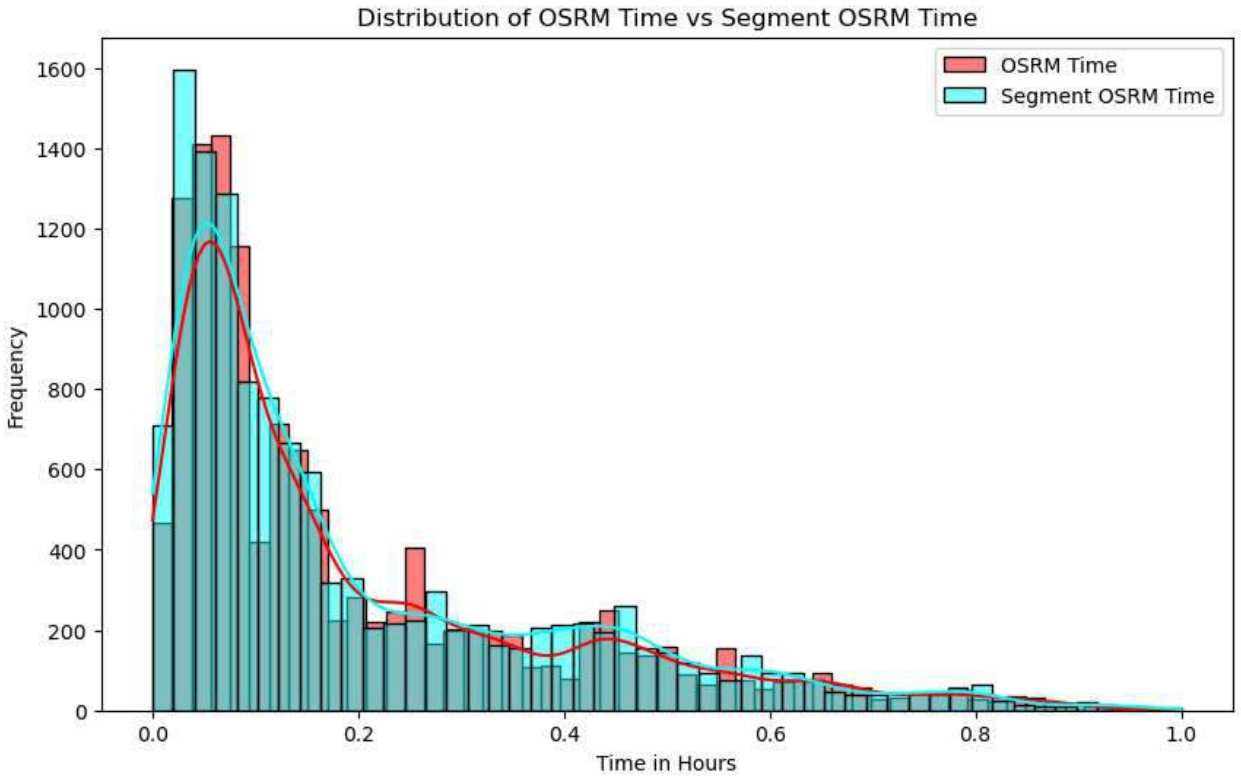

```
T-test between osrm_distance and segment_osrm_distance: t-statistic = -8.76, p-value =
0.0000
Reject Null Hypothesis: There is a significant difference between the aggregated OSRM d
istance and segment OSRM distance sum.
Inferences: There is enough evidence to suggest that there is a significant difference
between the aggregated OSRM distance and segment OSRM distance sum.
This implies that the open-source routing engine is inefficient in calculating both est
imated total distance covered(including external factors) & estimated
total distance covered(excluding external factors) to complete the delivery
Recommendations: Consider changing OSRM Engine/or look for some faults in OSRM Engine a
nd fix them
```

d. OSRM time aggregated value and segment OSRM time aggregated value.

In [91]:

```
# Visual analysis and hypothesis testing between osrm_time and segment_osrm_time
plt.figure(figsize=(10, 6))
sns.histplot(trip['osrm_time'], color='red', label='OSRM Time', kde=True)
sns.histplot(trip['segment_osrm_time_sum'], color='cyan', label='Segment OSRM Time', kd
plt.xlabel('Time in Hours')
plt.ylabel('Frequency')
plt.title('Distribution of OSRM Time vs Segment OSRM Time')
plt.legend()
plt.show()

# Hypothesis testing between osrm_time and segment_osrm_time
t_stat, p_value = stats.ttest_rel(trip['osrm_time'], trip['segment_osrm_time_sum'])
print(f"T-test between osrm_time and segment_osrm_time: t-statistic = {t_stat:.2f}, p-va

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
```

```
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("This implies that the open-source routing engine is inefficient in calculatir
    print("& estimated total distance covered(excluding external factors) to complete th
    print("that there is a significant difference between actual time of delivery and su
    print("This implies inefficiency in operations in the historical data given to the O
    print("Recommendations: Compare OSRM Time with actual time of delivery")
    print("Also compare Segment OSRM Time sum with Segment Actual Time Sum to check the
else:
    print("Inferences: There is not enough evidence to suggest that there is a significa
    print("This implies efficiency of OSRM Engine in calculating Estimated overall dista
    print("Recommendations: Compare both OSRM Time & Segment OSRM Time with Actual Time
    print("Continue monitoring OSRM time and Segment OSRM Time Sum to detect any emergir
    print("or data inputs.")
```

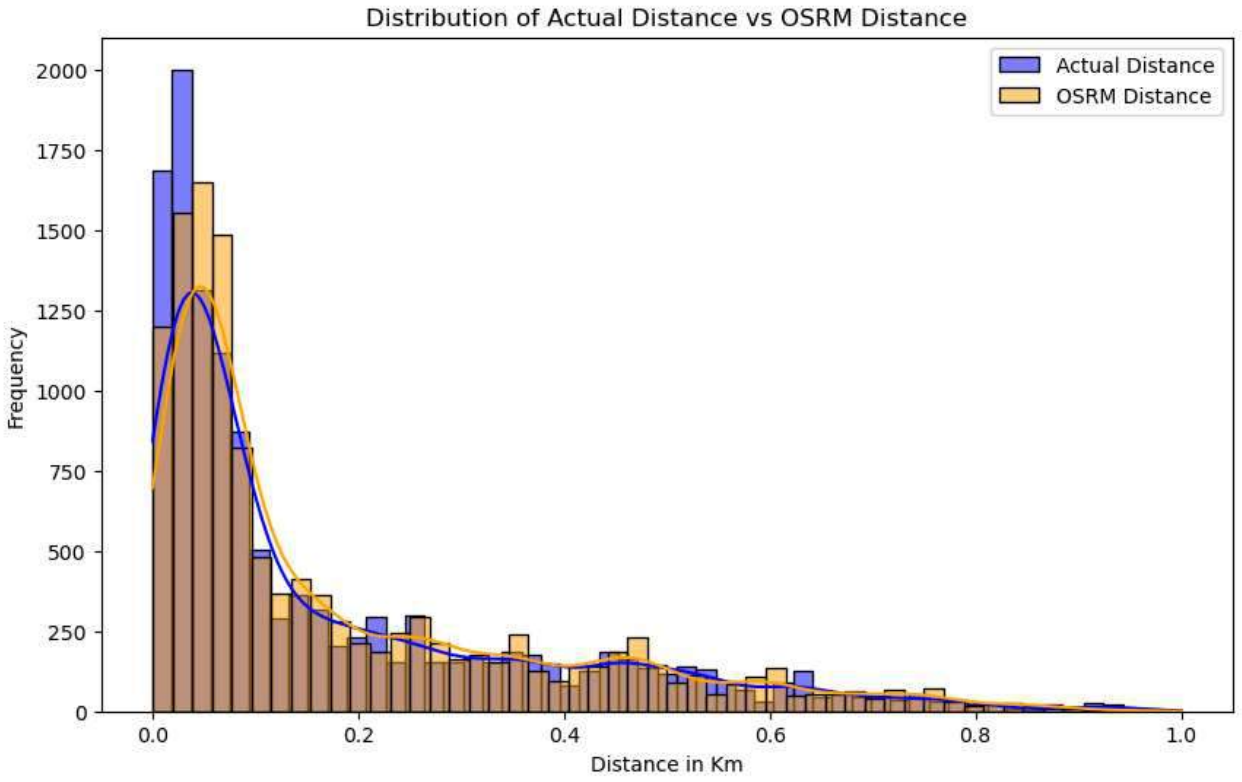



```
T-test between osrm_time and segment_osrm_time: t-statistic = -7.35, p-value = 0.0000
Reject Null Hypothesis: There is a significant difference between the aggregated OSRM t
ime and segment OSRM time sum.
Inferences: There is enough evidence to suggest that there is a significant difference
between the aggregated OSRM time and segment OSRM time sum.
This implies that the open-source routing engine is inefficient in calculating both est
imated total distance covered(including external factors)
& estimated total distance covered(excluding external factors) to complete the delivery
or OSRM Engine has observed(based on historical data)
that there is a significant difference between actual time of delivery and sum of time
taken by the subset of the package delivery
This implies inefficiency in operations in the historical data given to the OSRM
Recommendations: Compare OSRM Time with actual time of delivery
Also compare Segment OSRM Time sum with Segment Actual Time Sum to check the accuracy o
f OSRM Engine
```

e. Actual distance aggregated value and OSRM distance aggregated value

In [93]:
```
# Visual analysis and hypothesis testing between actual_distance_to_destination and osrm
plt.figure(figsize=(10, 6))
sns.histplot(trip['actual_distance_to_destination'], color='blue', label='Actual Distanc
sns.histplot(trip['osrm_distance'], color='orange', label='OSRM Distance', kde=True)
plt.xlabel('Distance in Km')
```

```
plt.ylabel('Frequency')
plt.title('Distribution of Actual Distance vs OSRM Distance')
plt.legend()
plt.show()

# Hypothesis testing between osrm_distance and segment_osrm_distance
t_stat, p_value = stats.ttest_rel(trip['actual_distance_to_destination'], trip['osrm_dis
print(f"T-test between Actual Distance and OSRM distance: t-statistic = {t_stat:.2f}, p

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("There is a significant difference between the aggregated Actual Distance and
    print("may be inefficient in accurately estimating the total distance covered.")
    print("Recommendations: Investigate the OSRM engine to identify potential errors or
    print("Consider alternative routing engines or make necessary adjustments to OSRM to
    print("Conduct additional tests to determine if the discrepancy is consistent acros
else:
    print("Inferences: There is not enough evidence to suggest that there is a signifi
    print("This implies efficiency of OSRM Engine in calculating Estimated overall dist
    print("Recommendations: Continue monitoring OSRM distance to see if any trends emer
```

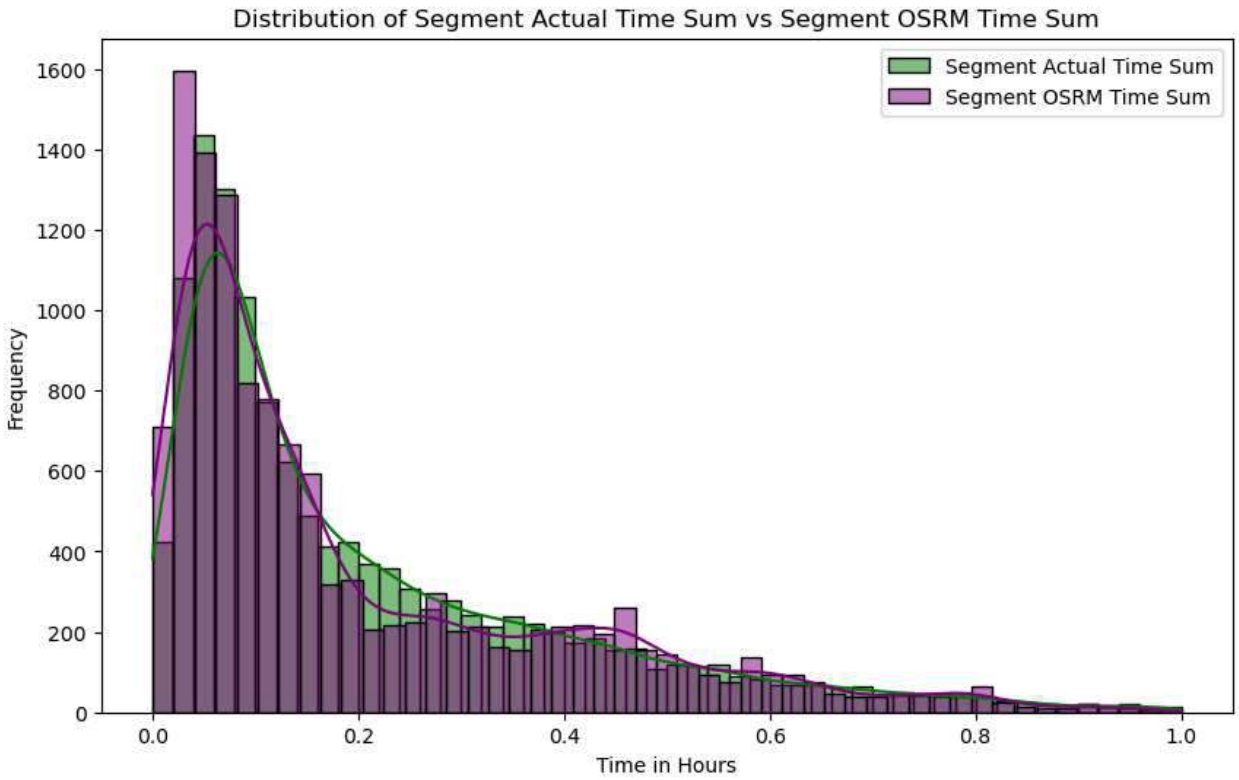


```
T-test between Actual Distance and OSRM distance: t-statistic = -27.77, p-value = 0.000
0
Reject Null Hypothesis: There is a significant difference between the aggregated Actual
Distance and aggregated OSRM distance.
Inferences: There is enough evidence to suggest that there is a significant difference
between the aggregated Actual Distance and aggregated OSRM distance.
There is a significant difference between the aggregated Actual Distance and the aggreg
ated OSRM distance. This suggests that the OSRM engine
may be inefficient in accurately estimating the total distance covered.
Recommendations: Investigate the OSRM engine to identify potential errors or limitation
s in the distance calculation algorithm.
Consider alternative routing engines or make necessary adjustments to OSRM to improve a
ccuracy.
Conduct additional tests to determine if the discrepancy is consistent across different
routes or specific to certain conditions.
```

f. Segment actual time aggregated value and Segment OSRM time aggregated value

In [95]:

```
# Visual analysis and hypothesis testing between segment_actual_time_sum and segment_osr
plt.figure(figsize=(10, 6))
sns.histplot(trip['segment_actual_time_sum'], color='green', label='Segment Actual Time
sns.histplot(trip['segment_osrm_time_sum'], color='purple', label='Segment OSRM Time Sur
plt.xlabel('Time in Hours')
plt.ylabel('Frequency')
plt.title('Distribution of Segment Actual Time Sum vs Segment OSRM Time Sum')
plt.legend()
plt.show()

# Hypothesis testing between actual_time and segment_actual_time_sum
t_stat, p_value = stats.ttest_rel(trip['segment_actual_time_sum'], trip['segment_osrm_t
print(f"T-test between Segment Actual Time Sum and Segment OSRM Time Sum: t-statistic =

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the Segment
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is enough evidence to suggest that there is a significant d
    print("This suggests that the OSRM engine may not be accurately estimating the time
    print("Recommendations: Investigate the causes of this discrepancy within the OSRM e
    print("for individual segments.Explore potential adjustments to the engine or consid
    print("time estimations.Test the OSRM engine's performance under varying conditions
else:
    print("Inferences: There is not enough evidence to suggest that there is a significa
    print("Segment OSRM Time Sum")
    print("This implies efficiency of OSRM Engine in calculating Estimated overall time
    print("Recommendations: Continue monitoring Segment Actual Time Sum and Segment OSRM
```



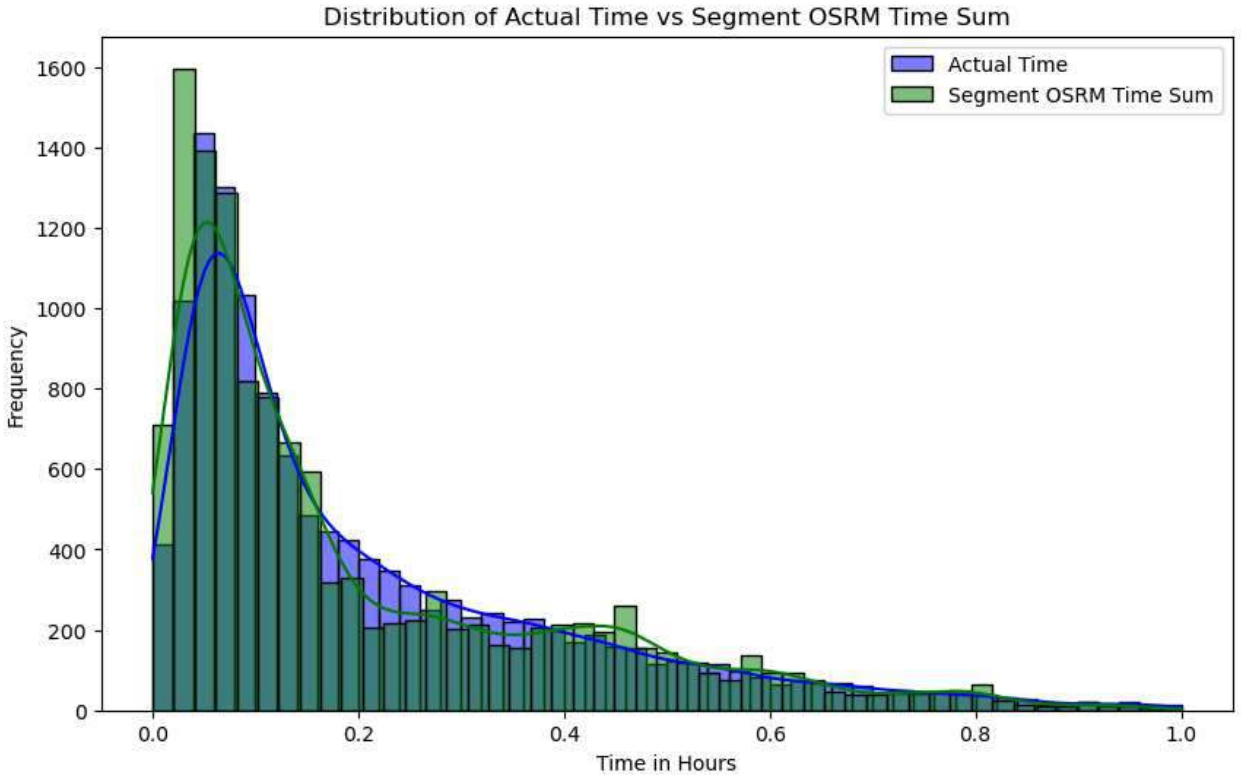

```
T-test between Segment Actual Time Sum and Segment OSRM Time Sum: t-statistic = 9.67, p
-value = 0.0000
Reject Null Hypothesis: There is a significant difference between the Segment Actual Ti
me Sum and Segment OSRM Time Sum
Inferences: There is enough evidence to suggest that there is a significant difference
between the Segment Actual Time Sum and Segment OSRM Time Sum
This suggests that the OSRM engine may not be accurately estimating the time required f
or individual segments, excluding external factors.
Recommendations: Investigate the causes of this discrepancy within the OSRM engine, foc
using on the time estimation algorithms
for individual segments.Explore potential adjustments to the engine or consider alterna
tive solutions that may offer more accurate
time estimations.Test the OSRM engine's performance under varying conditions to identif
y specific scenarios where it underperforms.
```

g. Actual_time aggregated value and Segment OSRM time aggregated value

In [97]:

```
# Visual analysis and hypothesis testing between actual_time and segment_osrm_time_sum
plt.figure(figsize=(10, 6))
sns.histplot(trip['actual_time'], color='blue', label='Actual Time', kde=True)
sns.histplot(trip['segment_osrm_time_sum'], color='green', label='Segment OSRM Time Sum
plt.xlabel('Time in Hours')
plt.ylabel('Frequency')
plt.title('Distribution of Actual Time vs Segment OSRM Time Sum')
plt.legend()
plt.show()

# Hypothesis testing between actual_time and segment_actual_time_sum
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['segment_osrm_time_sum'])
print(f"T-test between Actual Time and Segment OSRM Time Sum: t-statistic = {t_stat:.2f

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
```

```
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is a significant difference between the aggregated Actual T
    print("This implies that the OSRM engine is inefficient in calculating the estimated
    print("to complete the delivery.")
    print("Recommendations: Investigate potential inaccuracies in the OSRM engine's time
    print("Consider refining the OSRM engine or exploring alternative routing engines th
    print("Conduct further testing to understand the conditions under which these discre
else:
    print("Inferences: There is not enough evidence to suggest that there is a significa
    print("This implies efficiency of OSRM Engine in calculating Estimated overall time
    print("Recommendations: Continue monitoring Segment OSRM Time Sum and aggregated Act
```

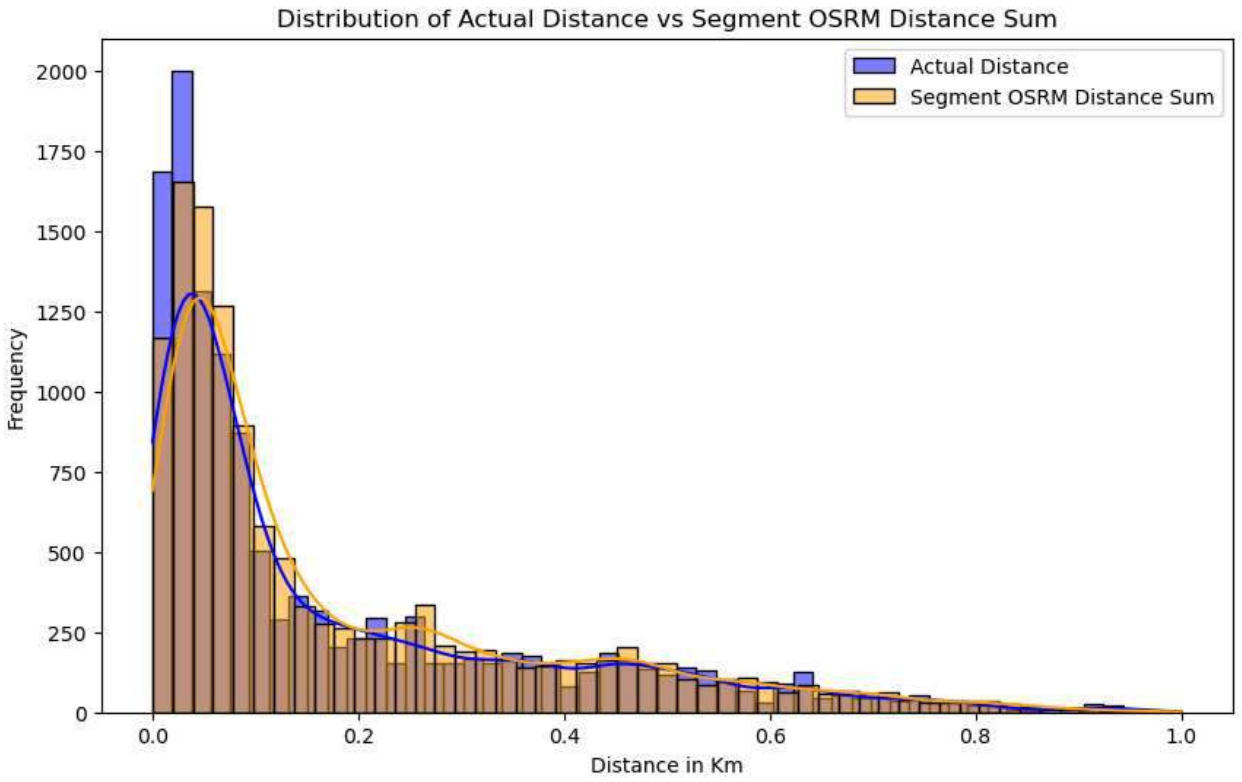



```
T-test between Actual Time and Segment OSRM Time Sum: t-statistic = 10.51, p-value = 0.
0000
Reject Null Hypothesis: There is a significant difference between the aggregated Actual
Time and Segment OSRM Time Sum
Inferences: There is a significant difference between the aggregated Actual Time and th
e Segment OSRM Time Sum.
This implies that the OSRM engine is inefficient in calculating the estimated total tim
e required (excluding external factors)
to complete the delivery.
Recommendations: Investigate potential inaccuracies in the OSRM engine's time estimatio
n algorithm, particularly for segmented journeys.
Consider refining the OSRM engine or exploring alternative routing engines that may pro
vide more accurate time predictions.
Conduct further testing to understand the conditions under which these discrepancies ar
ise and address them accordingly.
```

h. Actual distance aggregated value and Segment OSRM distance aggregated value

In [99]:
```
# Visual analysis and hypothesis testing between actual_distance_to_destination and segm
plt.figure(figsize=(10, 6))
sns.histplot(trip['actual_distance_to_destination'], color='blue', label='Actual Distanc
```

```
sns.histplot(trip['segment_osrm_distance_sum'], color='orange', label='Segment OSRM Dis
plt.xlabel('Distance in Km')
plt.ylabel('Frequency')
plt.title('Distribution of Actual Distance vs Segment OSRM Distance Sum')
plt.legend()
plt.show()

# Hypothesis testing between osrm_distance and segment_osrm_distance
t_stat, p_value = stats.ttest_rel(trip['actual_distance_to_destination'], trip['segment_
print(f"T-test between Actual Distance and Segment OSRM distance Sum: t-statistic = {t_

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is a significant difference between the aggregated Actual D
    print("This suggests that the OSRM engine is inefficient in calculating the estimate
    print("Recommendations: Examine the OSRM engine's distance calculation for segments
    print("Consider using alternative routing solutions if the OSRM engine consistently
    print("Conduct additional analysis to pinpoint the root cause of this discrepancy an
else:
    print("Inferences: There is not enough evidence to suggest that there is a signifi
    print("This implies efficiency of OSRM Engine in calculating Estimated overall dista
    print("Recommendations: Continue monitoring aggregated Actual Distance and Segment C
```

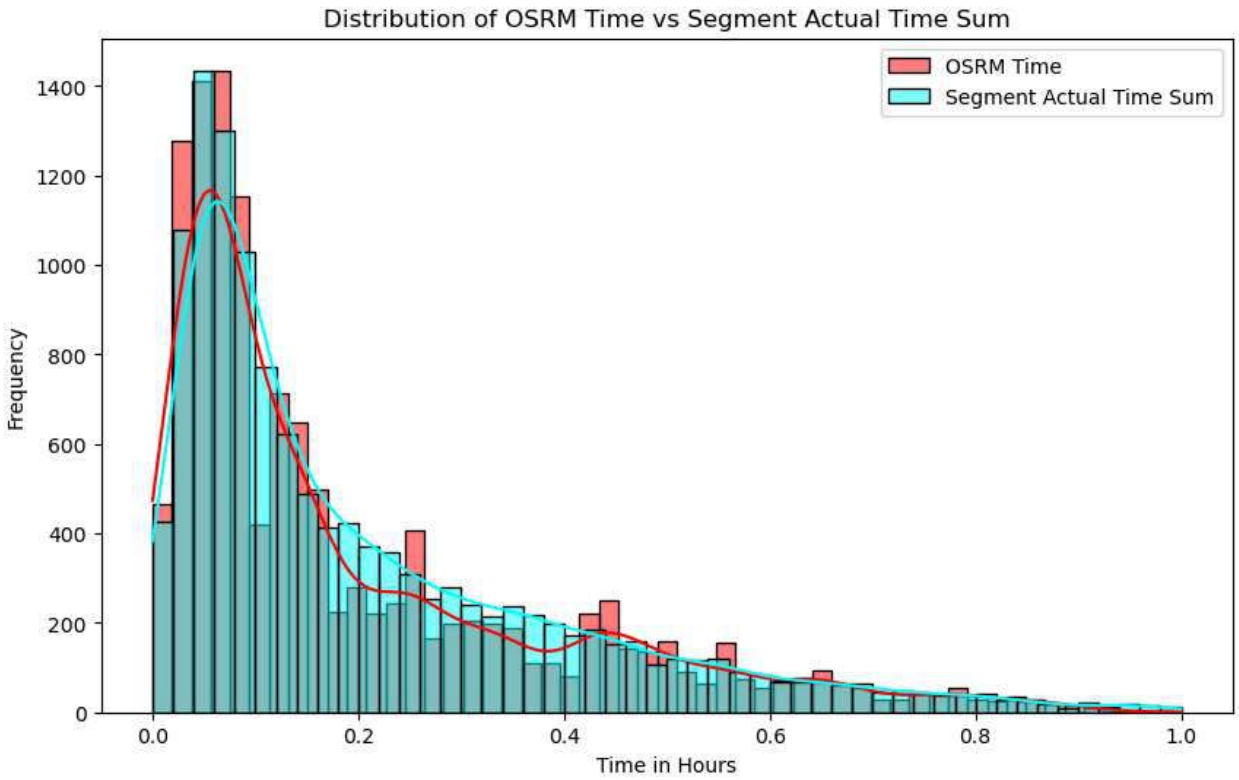


```
T-test between Actual Distance and Segment OSRM distance Sum: t-statistic = -25.09, p-v
alue = 0.0000
Reject Null Hypothesis: There is a significant difference between the aggregated Actual
Distance and Segment OSRM Distance Sum.
Inferences: There is a significant difference between the aggregated Actual Distance an
d the Segment OSRM Distance Sum.
This suggests that the OSRM engine is inefficient in calculating the estimated total di
stance covered (excluding external factors).
Recommendations: Examine the OSRM engine's distance calculation for segments to identif
y and correct any errors.
Consider using alternative routing solutions if the OSRM engine consistently underestim
ates or overestimates distances.
Conduct additional analysis to pinpoint the root cause of this discrepancy and address
it in future iterations of the routing engine.
```

i. OSRM time aggregated value and segment actual time aggregated value.

In [101…

```python
# Visual analysis and hypothesis testing between osrm_time and segment_actual_time_sum
plt.figure(figsize=(10, 6))
sns.histplot(trip['osrm_time'], color='red', label='OSRM Time', kde=True)
sns.histplot(trip['segment_actual_time_sum'], color='cyan', label='Segment Actual Time S
plt.xlabel('Time in Hours')
plt.ylabel('Frequency')
plt.title('Distribution of OSRM Time vs Segment Actual Time Sum')
plt.legend()
plt.show()

# Hypothesis testing between osrm_time and segment_osrm_time
t_stat, p_value = stats.ttest_rel(trip['osrm_time'], trip['segment_actual_time_sum'])
print(f"T-test between OSRM Time and Segment Actual Time Sum: t-statistic = {t_stat:.2f}

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the aggrega
else:
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:
    print("Inferences: There is a significant difference between the aggregated OSRM tir
    print("OSRM engine is inefficient in estimating the total time required (including e
    print("Recommendations: Investigate the reasons behind the OSRM engine's inefficienc
    print("and external factors. Consider enhancements to the OSRM engine or explore oth
    print("real-world conditions. Monitor and test the engine's performance across varic

else:
    print("Inferences: There is not enough evidence to suggest that there is a significa
    print("This implies efficiency of OSRM Engine in calculating Estimated overall time
    print("Continue monitoring OSRM time & Segment Actual Time Sum to see if any trends
```



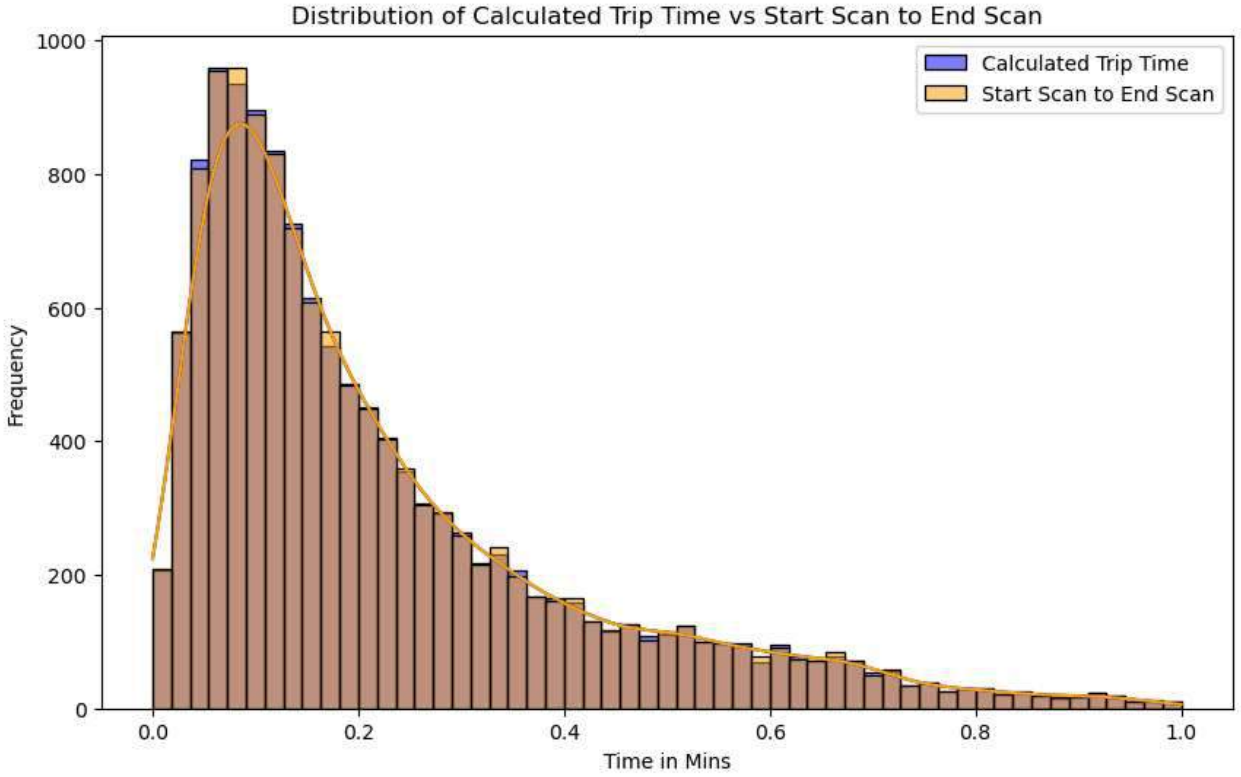

```
T-test between OSRM Time and Segment Actual Time Sum: t-statistic = -13.44, p-value =
0.0000
Reject Null Hypothesis: There is a significant difference between the aggregated OSRM t
ime and Segment Actual Time Sum
Inferences: There is a significant difference between the aggregated OSRM time and Segm
ent Actual Time Sum. This indicates that the
OSRM engine is inefficient in estimating the total time required (including external fa
ctors) to complete the delivery.
Recommendations: Investigate the reasons behind the OSRM engine's inefficiency in time
estimation, considering both internal
and external factors. Consider enhancements to the OSRM engine or explore other routing
engines that may better account for
real-world conditions. Monitor and test the engine's performance across various routes
and conditions to ensure any improvements are effective.
```

j. Calculated Trip Time vs Start_scan_to_end_scan

In [103…

```
# Visual analysis of calculated trip time vs start_scan_to_end_scan
plt.figure(figsize=(10, 6))
sns.histplot(trip['od_time_diff_mins'], color='blue', label='Calculated Trip Time', kde=
sns.histplot(trip['start_scan_to_end_scan'], color='orange', label='Start Scan to End S
plt.xlabel('Time in Mins')
plt.ylabel('Frequency')
plt.title('Distribution of Calculated Trip Time vs Start Scan to End Scan')
plt.legend()
plt.show()

# Hypothesis testing between calculated_trip_time and start_scan_to_end_scan
t_stat, p_value = stats.ttest_rel(trip['od_time_diff_mins'], trip['start_scan_to_end_sca
print(f"T-test between calculated_trip_time and start_scan_to_end_scan: t-statistic = {t

# Set significance level (alpha)
alpha = 0.05

# Decide whether to accept or reject the Null Hypothesis
if p_value <= alpha:
    print("Reject Null Hypothesis: There is a significant difference between the Calcula
else:
```

```
    print("Do not reject Null Hypothesis: There is no significant difference between the

# Draw inferences & conclusions and provide recommendations
if p_value <= alpha:

    print("Inferences: There is enough evidence of a significant difference between the
    print("This indicates operational inefficiencies, such as delays due to traffic or
    print("Recommendations: I) Consider optimizing delivery routes to minimize traffic-
    print("or routes.")
    print("II) Explore ways to improve the efficiency of loading and unloading processes
    print("per delivery to enhance overall delivery speed.")
else:
    print("Inferences: There is no significant difference between the Calculated Trip Ti
    print("operations are functioning efficiently with respect to time estimations.")
    print("This indicates great efficiency in operations")
    print("Recommendations: Continue monitoring the Calculated Trip Time and Start Scan
    print("Use this efficiency as a benchmark for evaluating other time-related metrics
    print("Perform periodic reviews to ensure that any changes in operations do not nega
```




```
T-test between calculated_trip_time and start_scan_to_end_scan: t-statistic = -9.36, p-
value = 0.0000
Reject Null Hypothesis: There is a significant difference between the Calculated Trip T
ime and Start Scan to End Scan
Inferences: There is enough evidence of a significant difference between the Calculated
Trip Time and the Start Scan to End Scan time.
This indicates operational inefficiencies, such as delays due to traffic or loading/unl
oading times, are affecting delivery performance.
Recommendations: I) Consider optimizing delivery routes to minimize traffic-related del
ays, possibly by adjusting delivery times
or routes.
II) Explore ways to improve the efficiency of loading and unloading processes, or reduc
e the number of parcels
per delivery to enhance overall delivery speed.
```

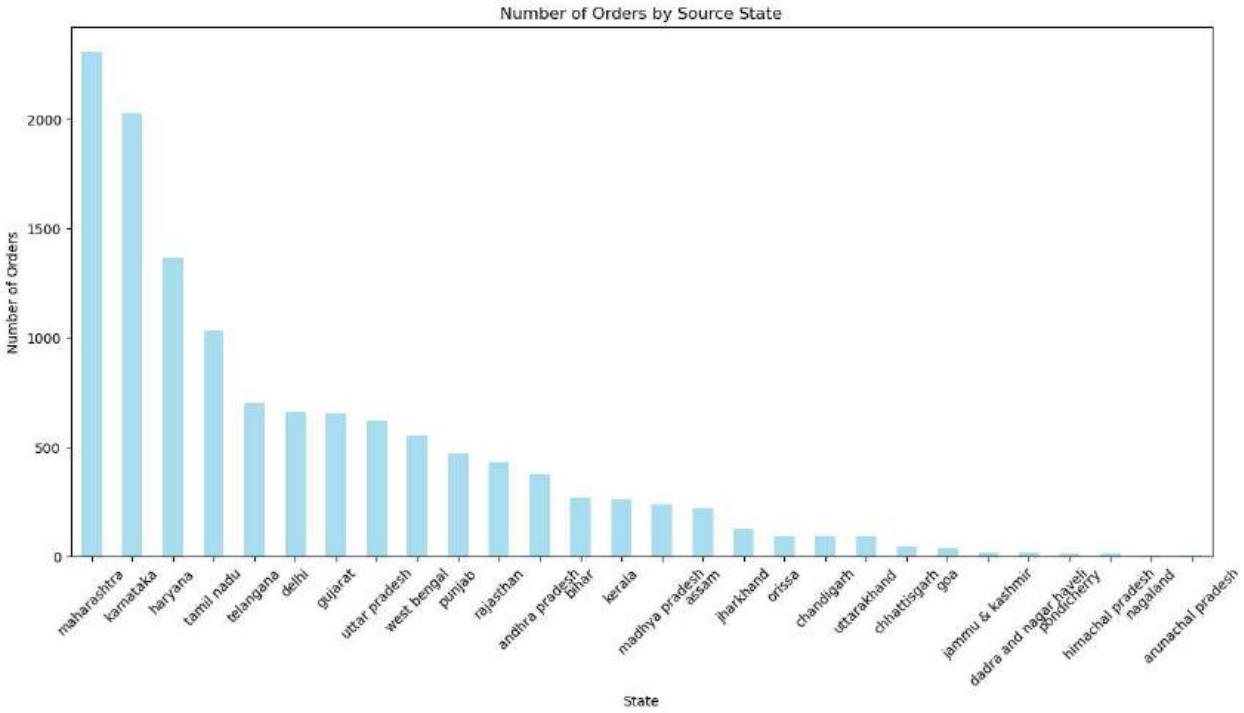
7. Visual Analysis to find Patterns in the data.

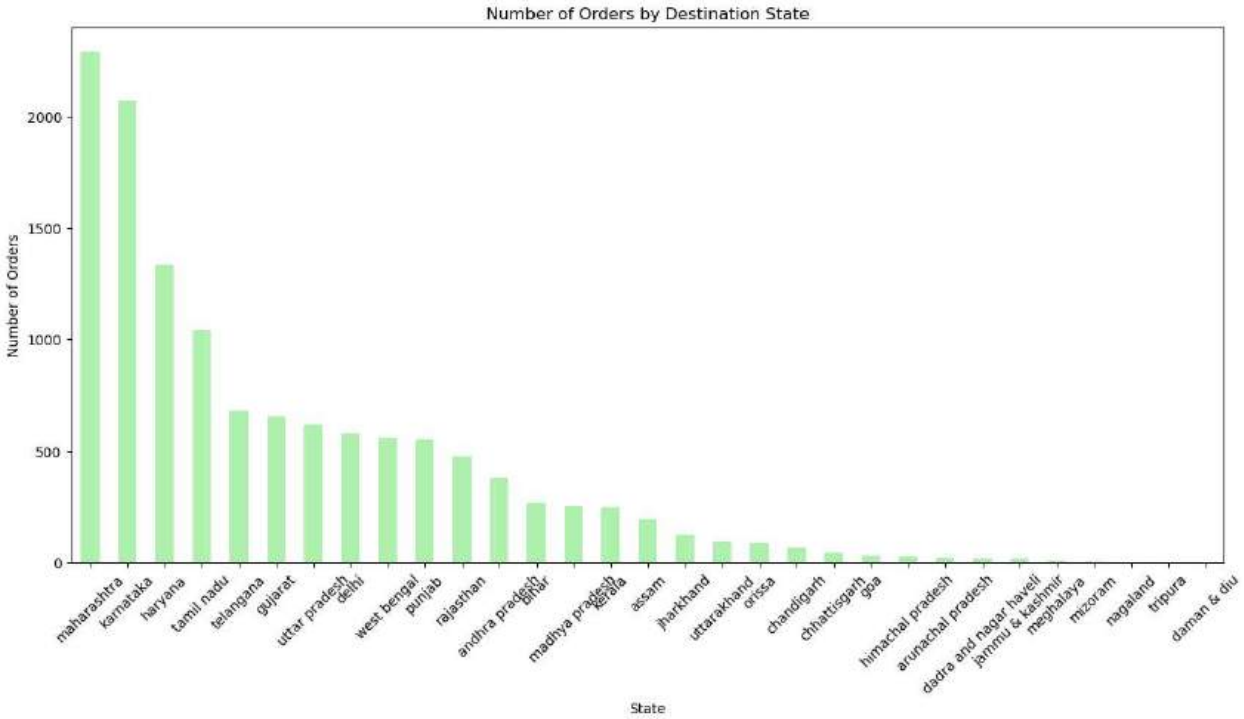
7.1 Visual Analysis to find from where most orders are coming from (State, Corridor, etc.)

7.2 Visual Analysis to find Busiest corridor, avg distance between them, avg time taken, etc.

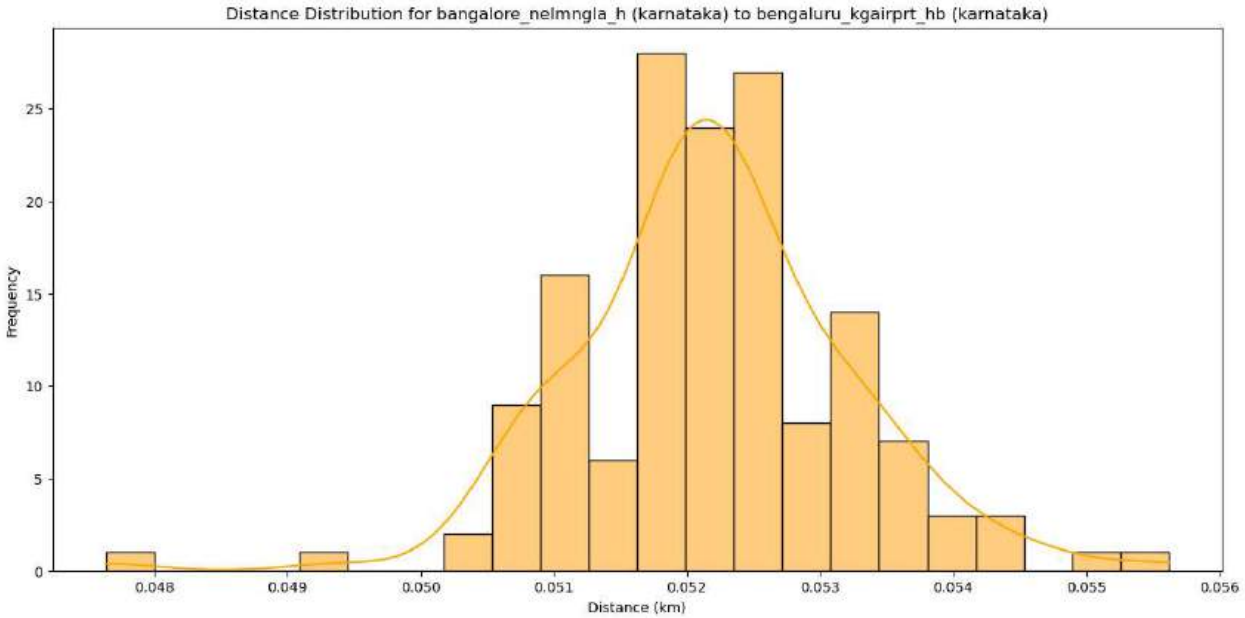
In [106…

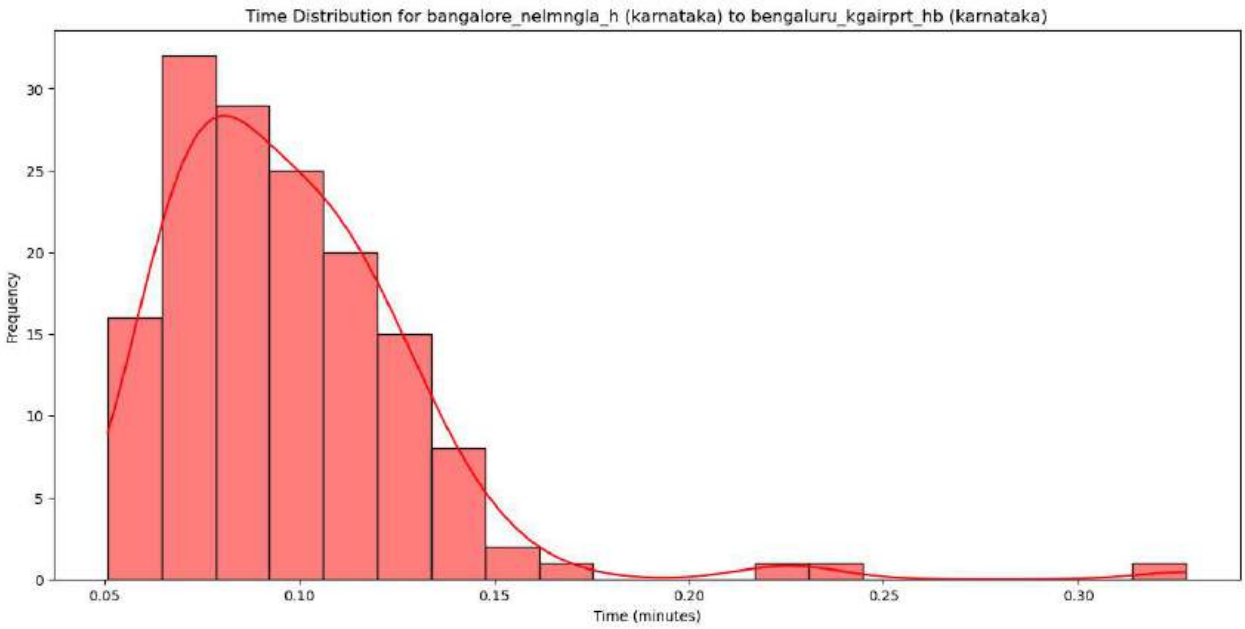
```
# Check where most orders are coming from
source_state_orders = trip['source_state'].value_counts()
destination_state_orders = trip['destination_state'].value_counts()

# Plot the order distribution by state
plt.figure(figsize=(15, 7))
source_state_orders.plot(kind='bar', color='skyblue', alpha=0.7)
plt.title('Number of Orders by Source State')
plt.xlabel('State')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(15, 7))
destination_state_orders.plot(kind='bar', color='lightgreen', alpha=0.7)
plt.title('Number of Orders by Destination State')
plt.xlabel('State')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()


# Identify the busiest corridor based on the most frequent source-destination pairs
corridor_counts = trip.groupby(['source_name', 'destination_name']).size().reset_index(
busiest_corridor = corridor_counts.sort_values(by='count', ascending=False).iloc[0]
busiest_corridor_name = f"{busiest_corridor['source_name']} to {busiest_corridor['destin

# Calculate average distance and time for the busiest corridor
busiest_corridor_data = trip[(trip['source_name'] == busiest_corridor['source_name']) &
avg_distance = busiest_corridor_data['actual_distance_to_destination'].mean()
avg_time = busiest_corridor_data['actual_time'].mean()

print(f"Busiest Corridor: {busiest_corridor_name}")
print(f"Average Distance: {avg_distance:.2f} km")
print(f"Average Time: {avg_time:.2f} minutes")

# Business Recommendations
# 1. Optimize Routes: Use insights to suggest better routing strategies.
# 2. Improve Efficiency: Identify delays and recommend strategies to reduce them.
# 3. Focus Areas: Identify states with high order volumes and suggest infrastructure imp

# Visualization of the busiest corridor's data
plt.figure(figsize=(15, 7))
sns.histplot(busiest_corridor_data['actual_distance_to_destination'], kde=True, color='
plt.title(f'Distance Distribution for {busiest_corridor_name}')
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(15, 7))
sns.histplot(busiest_corridor_data['actual_time'], kde=True, color='red')
plt.title(f'Time Distribution for {busiest_corridor_name}')
plt.xlabel('Time (minutes)')
plt.ylabel('Frequency')
plt.show()
```






```
Busiest Corridor: bangalore_nelmngla_h (karnataka) to bengaluru_kgairprt_hb (karnataka)
Average Distance: 0.05 km
Average Time: 0.10 minutes
```





7.3 Visual Analysis for finding Busiest month, day, week etc.

In [108…

```
# Calculate the number of trips for each month
busiest_month = trip['trip_month'].value_counts().idxmax()
busiest_month_trips = trip['trip_month'].value_counts().max()

# Calculate the number of trips for each day
busiest_day = trip['trip_day'].value_counts().idxmax()
busiest_day_trips = trip['trip_day'].value_counts().max()

# Calculate the number of trips for each day name
busiest_day_name = trip['trip_day_name'].value_counts().idxmax()
busiest_day_name_trips = trip['trip_day_name'].value_counts().max()

# Calculate the number of trips for each week
busiest_week = trip['trip_week'].value_counts().idxmax()
busiest_week_trips = trip['trip_week'].value_counts().max()

# Display the results
print(f"Busiest Month: {busiest_month}th with {busiest_month_trips} trips")
print(f"Busiest Day Name: {busiest_day_name} with {busiest_day_name_trips} trips")
```

```
print(f"Busiest Day: {busiest_day} with {busiest_day_trips} trips")
print(f"Busiest Week: {busiest_week} with {busiest_week_trips} trips")
```

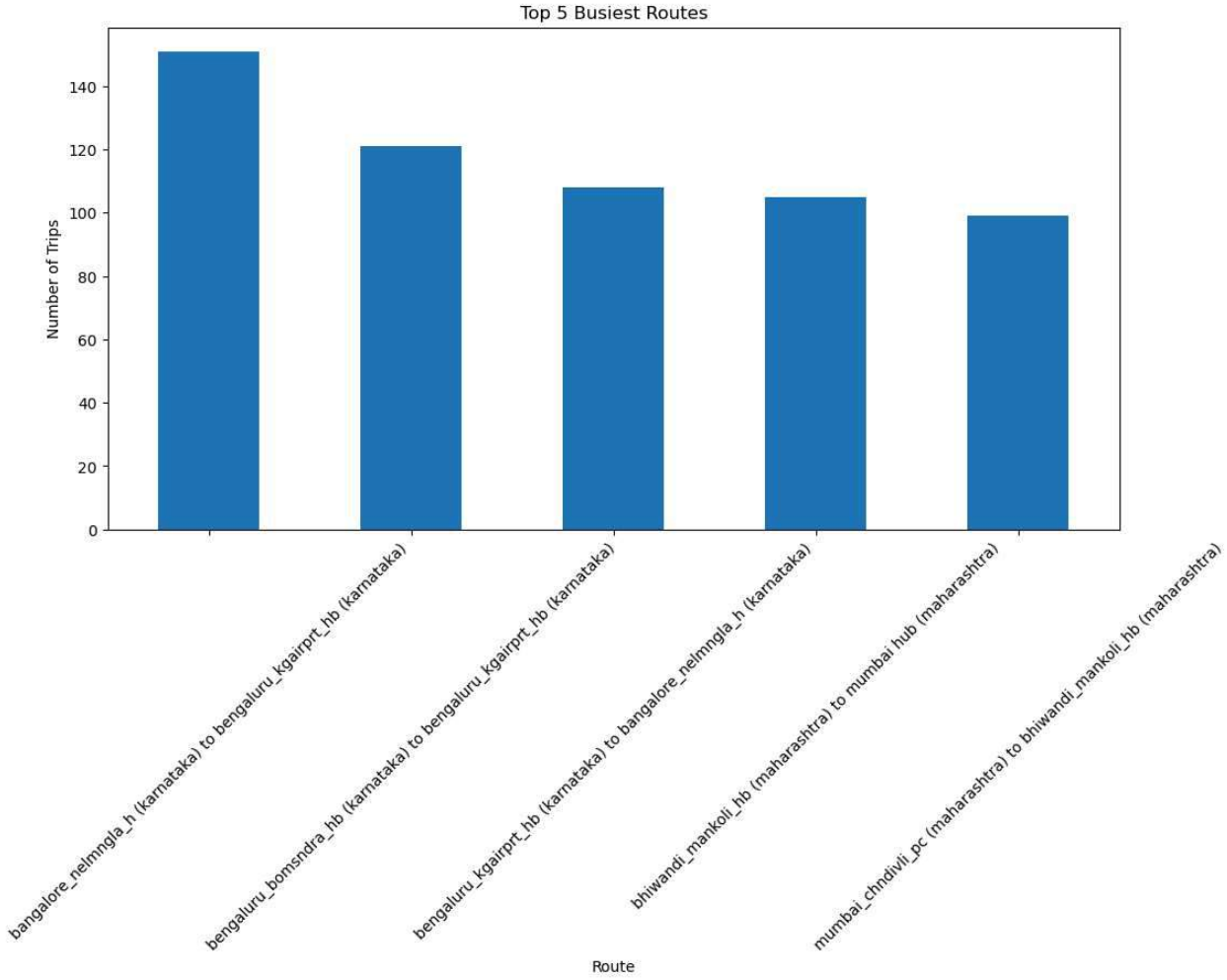
```
Busiest Month: 9th with 11172 trips
Busiest Day Name: Wednesday with 2352 trips
Busiest Day: 18 with 696 trips
Busiest Week: 38 with 4275 trips
```

7.4 Visual Analysis for finding Busiest Route, Popular Route, Emerging Route etc.

In [110…

```
# Create a new column for route by combining source and destination
trip['route'] = trip['source_name'] + ' to ' + trip['destination_name']

# Busiest Route: Route with the highest number of trips
busiest_route = trip['route'].value_counts().idxmax()
busiest_route_trips = trip['route'].value_counts().max()

# Popular Route: Route that consistently has a high number of trips
popular_route = trip.groupby('route').size().idxmax()

# Emerging Route: Route with the highest growth in trips over time
# Calculate monthly growth for each route
route_growth = trip.groupby(['trip_year', 'trip_month', 'route']).size().reset_index(nar
route_growth['previous_month_trip_count'] = route_growth.groupby('route')['trip_count']
route_growth['growth'] = route_growth['trip_count'] - route_growth['previous_month_trip_
route_growth = route_growth.dropna()

# Find the route with the highest growth
emerging_route = route_growth.loc[route_growth['growth'].idxmax(), 'route']
emerging_route_growth = route_growth['growth'].max()

# Display the results
print(f"Busiest Route: {busiest_route} with {busiest_route_trips} trips")
print(f"Popular Route: {popular_route}")
print(f"Emerging Route: {emerging_route} with growth of {emerging_route_growth} trips")

# Visualization of the top 5 routes
top_routes = trip['route'].value_counts().head(5)
plt.figure(figsize=(12, 6))
top_routes.plot(kind='bar')
plt.title('Top 5 Busiest Routes')
plt.xlabel('Route')
plt.ylabel('Number of Trips')
plt.xticks(rotation=45)
plt.show()
```

```
Busiest Route: bangalore_nelmngla_h (karnataka) to bengaluru_kgairprt_hb (karnataka) wi
th 151 trips
Popular Route: bangalore_nelmngla_h (karnataka) to bengaluru_kgairprt_hb (karnataka)
Emerging Route: ccu_new alipore_dpc (west bengal) to ccu_new alipore_dpc (west bengal)
with growth of 2.0 trips
```



8. Business Insights & Recommendations

8.1 Insights on No. of orders by Source & Destination State

- #### High Volume States: -Most orders are from Maharashtra, Karnataka, and Haryana. -Most orders are to Maharashtra, Karnataka, and Haryana as well.
- #### Low Volume States: -The least no. of orders are from Himachal Pradesh, Nagaland, and Arunachal Pradesh. -The least no. of orders are to Nagaland, Tripura and Daman & Diu.
- #### Geographical Trends: -The concentration of high-volume orders in Maharashtra, Karnataka, and Haryana may reflect their economic importance and better-developed logistics infrastructure. -Conversely, the states with the lowest order volumes are generally more remote or less economically developed, which could explain the reduced logistics activity.

8.1.1 Recommendations on No. of orders by Source & Destination State

- #### Focus on High-Volume States: -Optimize Operations: For Maharashtra, Karnataka, and Haryana, focus on optimizing logistics operations to further improve efficiency, given the high volume of orders. Consider investing in better transportation, warehousing, and distribution networks in these states to handle the demand.

-Leverage High Demand: Explore opportunities for expanding services, introducing premium delivery options, or establishing regional hubs in these states to capitalize on the high order volume.

- #### Address Challenges in Low-Volume States: -Improve Connectivity: For states like Himachal Pradesh, Nagaland, Arunachal Pradesh, Tripura, and Daman & Diu, consider studying the logistical challenges—such as poor infrastructure, difficult terrain, or limited demand—that may be contributing to the low order volumes. Look into partnerships with local providers or government initiatives to improve connectivity and infrastructure.

-Targeted Marketing and Incentives: Develop targeted marketing strategies to boost demand in these low-volume states. Offering incentives or discounts could attract more customers and increase order volumes from these regions.

- #### Expand Coverage and Reach: -Identify Growth Opportunities: Look for growth opportunities in under-served or emerging markets. This could involve expanding delivery coverage in low-volume states or exploring new customer segments that might be untapped in these regions.

-Diversify the Logistics Network: Consider diversifying the logistics network by establishing more regional distribution centers or partnering with local logistics providers to enhance service coverage in low-volume states.

- #### Data-Driven Decision Making: -Regular Monitoring: Continuously monitor the order volumes from different states to identify any emerging trends or changes in demand patterns. This will allow for timely adjustments in logistics planning and resource allocation.

-Customer Feedback: Gather and analyze customer feedback from both high- and low-volume states to understand their needs and preferences better, enabling more tailored service offerings.

8.2 Insights on Busiest Corridor, Average Distance, Average Time

- #### Busiest Corridor: -The corridor between Bangalore Nelamangala (Karnataka) and Bengaluru KGAirport (Karnataka) is identified as the busiest. This suggests a high frequency of logistics activities along this route, possibly due to the significance of these locations—one being a major suburban area and the other a key transportation hub (airport).
- #### Average Distance: 0.05 km
- #### Average Time: 0.10 minutes

8.2.1 Recommendations on Busiest Corridor, Average Distance, Average Time

- #### Optimize Operations on the Busiest Corridor: -Resource Allocation: Given that this is the busiest corridor, ensure that resources (vehicles, personnel) are optimally allocated to handle the high volume of activity efficiently.

-Infrastructure Enhancement: If the corridor involves critical areas like airports or industrial zones, consider infrastructure enhancements, such as dedicated lanes or improved scheduling, to further streamline operations.

- #### Investigate Underlying Causes: -Operational Efficiency: Assess whether the logistics operations along this corridor are running at optimal efficiency, or if there are ways to consolidate deliveries or reduce the number of trips.

- #### Monitor and Compare with Other Corridors: -Benchmarking: Compare this corridor's performance with other busy corridors to identify best practices or areas needing improvement. This will help in refining overall logistics strategy and ensuring consistency across different routes.

8.3 Insights- busiest month, day, week

- #### Busiest Month: -Month 9 (September) is the busiest month, with a total of 11,172 trips. This indicates a significant surge in logistics activity during this period.

- #### Busiest Day of the Week: -Wednesday is the busiest day of the week, with 2,352 trips. This suggests that midweek sees the highest volume of logistics operations.

- #### Busiest Specific Day: -The 18th of the month is the busiest specific day, with 696 trips. This might indicate a regular spike in activity around this date each month.

- #### Busiest Week: -Week 38 of the year is the busiest, with 4,275 trips. This suggests a concentrated period of high logistics activity during this particular week.

8.3.1 Recommendations- busiest month, day, week

- #### Resource Allocation:

-Increase Resources During Peak Periods: Given the high volume of trips in September, on Wednesdays, and during Week 38, consider adjusting resource allocation such as personnel, vehicles, and warehouse space—to accommodate the increased demand. This could include scheduling more shifts, adding temporary staff, or increasing fleet availability.

-Operational Efficiency: Ensure that the logistics operations are optimized for these peak times to prevent delays, bottlenecks, or overloading of the system. This could involve streamlining processes, improving coordination between teams, or utilizing advanced logistics software for better planning.

- #### Investigate Causes of Peaks:

-Analyze Demand Drivers: Investigate what causes these peaks in logistics activity, such as seasonal factors, specific customer demands, or promotional events. Understanding these drivers will help in planning for future peaks and possibly smoothing out demand across less busy periods.

-Customer Communication: If the peaks are driven by predictable factors like end-of-month or end-of-quarter business cycles, consider working with clients to distribute demand more evenly across the month or week.

- #### Enhance Predictive Planning:

-Forecasting: Use the data on peak times to enhance predictive planning models. This will allow the logistics team to anticipate and prepare for similar surges in the future, reducing the risk of being caught off guard by sudden spikes in activity.

-Scenario Planning: Develop contingency plans for unexpected increases in demand outside of these known peak times. This could include backup resources or flexible scheduling arrangements.

- #### Explore Automation and Technology:

-Automate Repetitive Tasks: Consider automating repetitive or time-consuming tasks during these peak periods to increase efficiency and reduce the burden on human resources. For example, automated sorting, routing, or dispatch systems could help manage the increased load.

-Technology Investment: Invest in technology solutions that can help manage peak loads, such as dynamic route optimization, real-time tracking, or AI-driven demand forecasting tools.

- #### Monitor and Adjust Over Time:

-Continuous Monitoring: Keep monitoring these trends over time to see if these patterns hold or change. If new peaks emerge, adjust operational strategies accordingly.

-Customer Feedback: Gather feedback from clients during these peak periods to identify areas where service can be improved, and use this information to refine operations and maintain customer satisfaction.

8.4 Insights on- Busiest route, popular route, emerging route

- #### Busiest Route: -The route from Bangalore Nelamangala (Karnataka) to Bengaluru KGAirport (Karnataka) is the busiest, with 151 trips. This indicates a high volume of logistics activity on this corridor, likely due to the significance of the locations as key industrial or transportation hubs.
- #### Popular Route: -The same route, Bangalore Nelamangala to Bengaluru KGAirport, is also the most popular, reaffirming its importance in the logistics network.
- #### Emerging Route: -The route from CCU New Alipore DPC (West Bengal) to CCU New Alipore DPC (West Bengal) is an emerging route, showing a growth of 2 trips. This suggests that this route is beginning to see increased activity, possibly due to recent developments or increased demand in the area.

8.4.1 Recommendations on- Busiest route, popular route, emerging route

- #### Optimize Busiest and Popular Route:

-Enhance Efficiency: Focus on optimizing operations along the Bangalore Nelamangala to Bengaluru KGAirport route by improving traffic management, route planning, and resource allocation to handle the high volume of trips.

-Monitor Congestion: Regularly monitor this route for congestion or delays and consider introducing dedicated lanes or alternative routes to maintain efficiency.

- #### Support Emerging Route:

-Scale Resources: Allocate additional resources to the emerging route from CCU New Alipore DPC to handle the growing demand. This could include increasing fleet availability or improving infrastructure along this route.

-Monitor Growth: Continue monitoring this route's performance to assess whether the growth trend persists and adjust operations accordingly to capitalize on the emerging demand.

- #### Leverage Data for Strategic Planning:

-Identify Trends: Use the insights from these routes to identify similar patterns in other regions and proactively prepare for shifts in logistics demand.

-Customer Targeting: If the emerging route's growth is driven by specific customers or industries, consider targeted marketing efforts to further drive business on this route.

8.5 Insights from Univariate and Bivariate Analysis:

I. Distribution of Continuous Variables:

-start_scan_to_end_scan: The distribution is slightly right-skewed, indicating that most deliveries occur within a certain time frame, with fewer outliers taking longer.

-actual_distance_to_destination: The distribution is also right-skewed, with most trips covering shorter distances and a few covering much longer distances.

-actual_time, osrm_time, segment_actual_time, segment_osrm_time: These variables also show right-skewed distributions, indicating that while most deliveries happen within expected time frames, there are cases where it takes significantly longer.

-osrm_distance, segment_osrm_distance: The distributions show that the majority of segments are short, with fewer long segments.

II. Countplots for Categorical Variables:

-route_type: The Carting type route type is much more common than the Full Truck Load (FTL) type, indicating that Delhivery primarily operates Carting shipments.

-source_center and destination_center: Certain centers are much more active, with a few centers handling the majority of the traffic. This could indicate key hubs or corridors.

III. Correlation Heatmap:

-actual_time and osrm_time: There is a strong positive correlation, indicating that the OSRM time prediction is generally aligned with the actual time taken.

-actual_distance_to_destination and osrm_distance: These variables are also highly correlated, suggesting that the OSRM's distance estimates are accurate relative to actual distances.

-segment_actual_time and segment_osrm_time: The segment times also show a strong positive correlation, reinforcing that OSRM predictions are generally reliable on a segment level as well.

8.5.1 Recommendations from Univariate and Bivariate Analysis:

- #### Route Optimization: -Since FTL is the dominant route type, efforts to optimize these routes further (e.g., reducing outlier delivery times) could significantly improve overall efficiency.
- #### Key Hubs Focus: -Given the concentration of traffic in certain centers, Delhivery should focus on optimizing operations in these key hubs, perhaps by improving infrastructure or increasing resources to handle high volumes efficiently.
- #### Predictive Modeling: -The strong correlation between actual and OSRM times/distances suggests that OSRM can be a reliable tool for forecasting and planning. Further refinement could improve the accuracy of delivery time predictions.
- #### Outlier Management: -The presence of outliers in time and distance data suggests a need for deeper analysis to understand the causes (e.g., traffic, route issues) and implement strategies to mitigate these delays.

8.6 Insights & Recommendations for Hypothesis T-tests between numerical variables

Insights:

There is a significant difference between OSRM and actual parameters.

Recommendations:

-Revisit information fed to routing engine for trip planning. Check for discrepancies with transporters, if the routing engine is configured for optimum results.

-North, South and West Zones corridors have significant traffic of orders. But, we have a smaller presence in Central, Eastern and North-Eastern zone. However it would be difficult to conclude this, by looking at just 2 months data. It is worth investigating and increasing our presence in these regions.

-From state point of view, we have heavy traffic in Mahrashtra followed by Karnataka. This is a good indicator that we need to plan for resources on ground in these 2 states on priority. Especially, during festive seasons.

-We have the least traffic in North eastern states like Arunachal Pradesh & Nagaland. It is worth investigating and increasing our presence in this region.

I. Actual Time vs. OSRM Time:

Insight:

-The significant difference between actual_time and osrm_time suggests that the OSRM (Open Source Routing Machine) estimates might not fully capture the realities of delivery times. Factors such as traffic congestion, road conditions, and unexpected delays could be influencing the actual time taken for deliveries.

-The OSRM time is likely calculated under ideal conditions, which do not always align with real-world scenarios.

Recommendation:

-Delhivery should consider enhancing their routing algorithms by incorporating real-time data, such as traffic patterns and weather conditions, to improve the accuracy of time predictions.

-Continuous monitoring and adjustment of OSRM predictions based on historical data could also reduce discrepancies between estimated and actual delivery times.

II. Actual Time vs. Segment Actual Time:

Insight:

-The significant difference between actual_time and segment_actual_time indicates that the total time for a delivery is not simply the sum of the segment times. This could be due to additional delays that occur when transitioning between segments, such as loading/unloading time, waiting periods, or administrative processes.

Recommendation:

-Delhivery should analyze the transition times between segments to identify bottlenecks and inefficiencies. Streamlining these processes could lead to a reduction in overall delivery times.

-Implementing better coordination and planning between segments might also help in reducing the time lost during transitions.

III. OSRM Distance vs. Segment OSRM Distance:

Insight:

-The significant difference between osrm_distance and segment_osrm_distance suggests that the sum of distances covered in segments does not match the overall route distance. This discrepancy might be due to detours, route changes, or inaccuracies in the routing engine when breaking down the journey into segments.

Recommendation:

-Delhivery should investigate the reasons for these discrepancies by analyzing segment-specific routes. Understanding why the sum of segment distances deviates from the total distance could help in refining route planning and optimizing delivery paths.

-Consider implementing more granular tracking and monitoring of routes to ensure that the segments align better with the overall planned route.

IV. OSRM Time vs. Segment OSRM Time:

Insight:

-The significant difference between osrm_time and segment_osrm_time indicates that the time predicted for segments doesn't align well with the overall predicted time. This could be due to variations in traffic, road conditions, or other factors that affect specific segments differently.

Recommendation:

-Delhivery should refine their segment-specific time predictions by incorporating more localized data, such as traffic information specific to each segment.

-Additionally, improving the accuracy of time estimates for each segment will help in better overall delivery time predictions, leading to more efficient route planning and customer satisfaction.

V. Actual Distance Aggregated Value vs. OSRM Distance Aggregated Value

Insight:

-There is a significant difference between the aggregated Actual Distance and the aggregated OSRM distance.

-This suggests that the OSRM engine may be inefficient in accurately estimating the total distance covered.

Recommendations:

-Investigate the OSRM engine to identify potential errors or limitations in the distance calculation algorithm.

-Consider alternative routing engines or make necessary adjustments to OSRM to improve accuracy.

-Conduct additional tests to determine if the discrepancy is consistent across different routes or specific to certain conditions.

VI. Segment Actual Time Aggregated Value and Segment OSRM Time Aggregated Value

Insight:

-There is a significant difference between the Segment Actual Time Sum and the Segment OSRM Time Sum.

-This suggests that the OSRM engine may not be accurately estimating the time required for individual segments, excluding external factors.

Recommendations:

-Investigate the causes of this discrepancy within the OSRM engine, focusing on the time estimation algorithms for individual segments.

-Explore potential adjustments to the engine or consider alternative solutions that may offer more accurate time estimations.

-Test the OSRM engine's performance under varying conditions to identify specific scenarios where it underperforms.

VII. Actual Time Aggregated Value and Segment OSRM Time Aggregated Value

Insight:

-There is a significant difference between the aggregated Actual Time and the Segment OSRM Time Sum.

-This implies that the OSRM engine is inefficient in calculating the estimated total time required (excluding external factors) to complete the delivery.

Recommendations:

-Investigate potential inaccuracies in the OSRM engine's time estimation algorithm, particularly for segmented journeys.

-Consider refining the OSRM engine or exploring alternative routing engines that may provide more accurate time predictions.

-Conduct further testing to understand the conditions under which these discrepancies arise and address them accordingly.

VIII. Actual Distance Aggregated Value and Segment OSRM Distance Aggregated Value

Insight:

-There is a significant difference between the aggregated Actual Distance and the Segment OSRM Distance Sum.

-This suggests that the OSRM engine is inefficient in calculating the estimated total distance covered (excluding external factors).

Recommendations:

-Examine the OSRM engine's distance calculation for segments to identify and correct any errors.

-Consider using alternative routing solutions if the OSRM engine consistently underestimates or overestimates distances.

-Conduct additional analysis to pinpoint the root cause of this discrepancy and address it in future iterations of the routing engine.

IX. OSRM Time Aggregated Value and Segment Actual Time Aggregated Value

Insight:

-There is a significant difference between the aggregated OSRM time and Segment Actual Time Sum.

-This indicates that the OSRM engine is inefficient in estimating the total time required (including external factors) to complete the delivery.

Recommendations:

-Investigate the reasons behind the OSRM engine's inefficiency in time estimation, considering both internal and external factors.

-Consider enhancements to the OSRM engine or explore other routing engines that may better account for real-world conditions.

-Monitor and test the engine's performance across various routes and conditions to ensure any improvements are effective.

X. Calculated Trip Time vs. Start Scan to End Scan

Insight:

-There is strong evidence of a significant difference between the Calculated Trip Time and the Start Scan to End Scan time. This indicates operational inefficiencies, such as delays due to traffic or loading/unloading times, are affecting delivery performance.

Recommendations:

-Consider optimizing delivery routes to minimize traffic-related delays, possibly by adjusting delivery times or routes.

-Explore ways to improve the efficiency of loading and unloading processes, or reduce the number of parcels per delivery to enhance overall delivery speed.