delhivery1

January 7, 2025

[41]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as skl
```

[42]:
```
df=pd.read_csv("D:\delhivery_data.csv")
```

[43]:
```
df.head()
```

[43]:
```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

                              route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting

                 trip_uuid source_center                source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center             destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
```

```
                od_start_time  …            cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  …         2018-09-20 04:27:55
1  2018-09-20 03:21:32.418600  …         2018-09-20 04:17:55
2  2018-09-20 03:21:32.418600  …  2018-09-20 04:01:19.505586
3  2018-09-20 03:21:32.418600  …         2018-09-20 03:39:57
4  2018-09-20 03:21:32.418600  …         2018-09-20 03:33:55

   actual_distance_to_destination  actual_time  osrm_time osrm_distance  \
0                       10.435660         14.0       11.0       11.9653
1                       18.936842         24.0       20.0       21.7243
2                       27.637279         40.0       28.0       32.5395
3                       36.118028         62.0       40.0       45.5620
4                       39.386040         68.0       44.0       54.2181

     factor  segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0  1.272727                 14.0               11.0                11.9653
1  1.200000                 10.0                9.0                 9.7590
2  1.428571                 16.0                7.0                10.8152
3  1.550000                 21.0               12.0                13.0224
4  1.545455                  6.0                5.0                 3.9153

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

[44]:
```
df.isnull().sum()
```

[44]:
```
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
```

[45]:
```
# Replace missing values in 'source_name' and 'destination_name' with 'Unknown'
df['source_name'].fillna('Unknown', inplace=True)
df['destination_name'].fillna('Unknown', inplace=True)
```

[46]:
```
# Check for missing values again to ensure they're handled
df.isnull().sum()
```

[46]:
```
data                            0
trip_creation_time              0
route_schedule_uuid             0
route_type                      0
trip_uuid                       0
source_center                   0
source_name                     0
destination_center              0
destination_name                0
od_start_time                   0
od_end_time                     0
start_scan_to_end_scan          0
is_cutoff                       0
cutoff_factor                   0
cutoff_timestamp                0
actual_distance_to_destination  0
actual_time                     0
osrm_time                       0
osrm_distance                   0
factor                          0
segment_actual_time             0
segment_osrm_time               0
segment_osrm_distance           0
segment_factor                  0
dtype: int64
```

[47]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  object
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144867 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144867 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
```

[48]:
```
    # Convert time columns into pandas datetime
    df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'])
    df['od_start_time'] = pd.to_datetime(df['od_start_time'])
    df['od_end_time'] = pd.to_datetime(df['od_end_time'])
    df['cutoff_timestamp'] = pd.to_datetime(df['cutoff_timestamp'],␣
 ↪errors='coerce')  # Handling any invalid date entries
```

[49]:
```
# Step 1: Create the segment_key by combining trip_uuid, source_center, and␣
 ↪destination_center
df['segment_key'] = df['trip_uuid'].astype(str) + '_' + df['source_center'].
 ↪astype(str) + '_' + df['destination_center'].astype(str)

# Step 2: Group by segment_key and apply cumsum for the required numeric columns
df['segment_actual_time_sum'] = df.
 ↪groupby('segment_key')['segment_actual_time'].cumsum()
```

```
df['segment_osrm_distance_sum'] = df.
 ↪groupby('segment_key')['segment_osrm_distance'].cumsum()
df['segment_osrm_time_sum'] = df.groupby('segment_key')['segment_osrm_time'].
 ↪cumsum()

# Step 3: Check the result
df[['trip_uuid', 'source_center', 'destination_center', 'segment_key',␣
 ↪'segment_actual_time_sum', 'segment_osrm_distance_sum',␣
 ↪'segment_osrm_time_sum']].head()
```

[49]:

	trip_uuid	source_center	destination_center	segment_key	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_sum
0	trip-153741093647649320	IND388121AAA	IND388620AAB	trip-153741093647649320_IND388121AAA_IND388620AAB	14.0	11.9653	11.0
1	trip-153741093647649320	IND388121AAA	IND388620AAB	trip-153741093647649320_IND388121AAA_IND388620AAB	24.0	21.7243	20.0
2	trip-153741093647649320	IND388121AAA	IND388620AAB	trip-153741093647649320_IND388121AAA_IND388620AAB	40.0	32.5395	27.0
3	trip-153741093647649320	IND388121AAA	IND388620AAB	trip-153741093647649320_IND388121AAA_IND388620AAB	61.0	45.5619	39.0
4	trip-153741093647649320	IND388121AAA	IND388620AAB	trip-153741093647649320_IND388121AAA_IND388620AAB	67.0	49.4772	44.0

[50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 28 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   data                          144867 non-null  object
 1   trip_creation_time            144867 non-null  datetime64[ns]
 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                   144867 non-null  object
 7   destination_center            144867 non-null  object
 8   destination_name              144867 non-null  object
```

```
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool
 13  cutoff_factor                   144867 non-null  int64
 14  cutoff_timestamp                141438 non-null  datetime64[ns]
 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
 24  segment_key                     144867 non-null  object
 25  segment_actual_time_sum         144867 non-null  float64
 26  segment_osrm_distance_sum       144867 non-null  float64
 27  segment_osrm_time_sum           144867 non-null  float64
dtypes: bool(1), datetime64[ns](4), float64(13), int64(1), object(9)
memory usage: 30.0+ MB
```

[51]:
```
df.isnull().sum()
```

[51]:
```
data                                0
trip_creation_time                  0
route_schedule_uuid                 0
route_type                          0
trip_uuid                           0
source_center                       0
source_name                         0
destination_center                  0
destination_name                    0
od_start_time                       0
od_end_time                         0
start_scan_to_end_scan              0
is_cutoff                           0
cutoff_factor                       0
cutoff_timestamp                 3429
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
segment_key                     0
segment_actual_time_sum         0
segment_osrm_distance_sum       0
segment_osrm_time_sum           0
dtype: int64
```

[52]:
```python
# Verify column names
print(df.columns)

# Fix any potential typos or missing columns
create_segment_dict = {
    'segment_actual_time': 'cumsum',
    'segment_osrm_distance': 'cumsum',
    'segment_osrm_time': 'cumsum',
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'trip_uuid': 'first',
    'source_center': 'first',
    'source_name': 'first',
    'destination_center': 'first',  # Fix typo here (from 'destination_cente'␣
 ↪to 'destination_center')
    'destination_name': 'first',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'last',
    'actual_distance_to_destination': 'last',
    'actual_time': 'last',
    'osrm_time': 'last',
    'osrm_distance': 'last',
    'factor': 'first',
    'segment_factor': 'first'
}

# Step 2: Group by segment_key and apply the aggregation functions
df_grouped = df.groupby('trip_uuid').agg(create_segment_dict).reset_index()

# Step 3: Calculate cumulative sums where applicable
df_grouped['segment_actual_time_sum'] = df.
 ↪groupby('trip_uuid')['segment_actual_time'].cumsum()
df_grouped['segment_osrm_distance_sum'] = df.
 ↪groupby('trip_uuid')['segment_osrm_distance'].cumsum()
df_grouped['segment_osrm_time_sum'] = df.
 ↪groupby('trip_uuid')['segment_osrm_time'].cumsum()

# Step 4: Print the result to check
```

```
print(df_grouped.head())
```

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor',
       'segment_key', 'segment_actual_time_sum', 'segment_osrm_distance_sum',
       'segment_osrm_time_sum'],
      dtype='object')
  index  segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0     0                 14.0                11.9653               11.0
1     1                 24.0                21.7243               20.0
2     2                 40.0                32.5395               27.0
3     3                 61.0                45.5619               39.0
4     4                 67.0                49.4772               44.0

  trip_creation_time route_schedule_uuid route_type trip_uuid source_center  \
0                NaT                 NaN        NaN       NaN           NaN
1                NaT                 NaN        NaN       NaN           NaN
2                NaT                 NaN        NaN       NaN           NaN
3                NaT                 NaN        NaN       NaN           NaN
4                NaT                 NaN        NaN       NaN           NaN

  source_name  … start_scan_to_end_scan actual_distance_to_destination  \
0         NaN  …                   NaN                            NaN
1         NaN  …                   NaN                            NaN
2         NaN  …                   NaN                            NaN
3         NaN  …                   NaN                            NaN
4         NaN  …                   NaN                            NaN

  actual_time osrm_time  osrm_distance  factor  segment_factor  \
0         NaN       NaN            NaN     NaN             NaN
1         NaN       NaN            NaN     NaN             NaN
2         NaN       NaN            NaN     NaN             NaN
3         NaN       NaN            NaN     NaN             NaN
4         NaN       NaN            NaN     NaN             NaN

   segment_actual_time_sum  segment_osrm_distance_sum  segment_osrm_time_sum
0                     14.0                    11.9653                   11.0
1                     24.0                    21.7243                   20.0
2                     40.0                    32.5395                   27.0
3                     61.0                    45.5619                   39.0
4                     67.0                    49.4772                   44.0
```

```
[5 rows x 24 columns]
```

[53]:
```
df_grouped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159684 entries, 0 to 159683
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   index                           159684 non-null  object
 1   segment_actual_time             144867 non-null  float64
 2   segment_osrm_distance           144867 non-null  float64
 3   segment_osrm_time               144867 non-null  float64
 4   trip_creation_time              14817 non-null   datetime64[ns]
 5   route_schedule_uuid             14817 non-null   object
 6   route_type                      14817 non-null   object
 7   trip_uuid                       14817 non-null   object
 8   source_center                   14817 non-null   object
 9   source_name                     14817 non-null   object
 10  destination_center              14817 non-null   object
 11  destination_name                14817 non-null   object
 12  od_start_time                   14817 non-null   datetime64[ns]
 13  od_end_time                     14817 non-null   datetime64[ns]
 14  start_scan_to_end_scan          14817 non-null   float64
 15  actual_distance_to_destination  14817 non-null   float64
 16  actual_time                     14817 non-null   float64
 17  osrm_time                       14817 non-null   float64
 18  osrm_distance                   14817 non-null   float64
 19  factor                          14817 non-null   float64
 20  segment_factor                  14817 non-null   float64
 21  segment_actual_time_sum         144867 non-null  float64
 22  segment_osrm_distance_sum       144867 non-null  float64
 23  segment_osrm_time_sum           144867 non-null  float64
dtypes: datetime64[ns](3), float64(13), object(8)
memory usage: 29.2+ MB
```

[54]:
```
# Fill missing values using forward fill method (useful for time-based or␣
 ↪sequence data)
df_grouped.fillna(method='ffill', inplace=True)

# Alternatively, if forward fill doesn't make sense in your context, fill with␣
 ↪a specific value (e.g., 0 or mean)
df_grouped.fillna({
    'segment_actual_time': 0,
    'segment_osrm_distance': 0,
    'segment_osrm_time': 0,
    'segment_factor': 1,
```

```
    'factor': 1
}, inplace=True)

# Verify after filling missing data
print(df_grouped.info())  # Check non-null counts again
print(df_grouped.head())  # Check the first few rows
```

```
C:\Users\samvj\AppData\Local\Temp\ipykernel_13620\3084626157.py:2:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  df_grouped.fillna(method='ffill', inplace=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159684 entries, 0 to 159683
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   index                           159684 non-null  object
 1   segment_actual_time             159684 non-null  float64
 2   segment_osrm_distance           159684 non-null  float64
 3   segment_osrm_time               159684 non-null  float64
 4   trip_creation_time              14817 non-null   datetime64[ns]
 5   route_schedule_uuid             14817 non-null   object
 6   route_type                      14817 non-null   object
 7   trip_uuid                       14817 non-null   object
 8   source_center                   14817 non-null   object
 9   source_name                     14817 non-null   object
 10  destination_center              14817 non-null   object
 11  destination_name                14817 non-null   object
 12  od_start_time                   14817 non-null   datetime64[ns]
 13  od_end_time                     14817 non-null   datetime64[ns]
 14  start_scan_to_end_scan          14817 non-null   float64
 15  actual_distance_to_destination  14817 non-null   float64
 16  actual_time                     14817 non-null   float64
 17  osrm_time                       14817 non-null   float64
 18  osrm_distance                   14817 non-null   float64
 19  factor                          159684 non-null  float64
 20  segment_factor                  159684 non-null  float64
 21  segment_actual_time_sum         159684 non-null  float64
 22  segment_osrm_distance_sum       159684 non-null  float64
 23  segment_osrm_time_sum           159684 non-null  float64
dtypes: datetime64[ns](3), float64(13), object(8)
memory usage: 29.2+ MB
None
  index  segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0     0                 14.0                11.9653               11.0
1     1                 24.0                21.7243               20.0
2     2                 40.0                32.5395               27.0
```

```
3     3                61.0                45.5619               39.0
4     4                67.0                49.4772               44.0

  trip_creation_time route_schedule_uuid route_type trip_uuid source_center  \
0                NaT                 NaN        NaN       NaN           NaN
1                NaT                 NaN        NaN       NaN           NaN
2                NaT                 NaN        NaN       NaN           NaN
3                NaT                 NaN        NaN       NaN           NaN
4                NaT                 NaN        NaN       NaN           NaN

  source_name  … start_scan_to_end_scan actual_distance_to_destination  \
0         NaN  …                   NaN                            NaN
1         NaN  …                   NaN                            NaN
2         NaN  …                   NaN                            NaN
3         NaN  …                   NaN                            NaN
4         NaN  …                   NaN                            NaN

  actual_time osrm_time  osrm_distance  factor  segment_factor  \
0         NaN       NaN            NaN     1.0             1.0
1         NaN       NaN            NaN     1.0             1.0
2         NaN       NaN            NaN     1.0             1.0
3         NaN       NaN            NaN     1.0             1.0
4         NaN       NaN            NaN     1.0             1.0

   segment_actual_time_sum  segment_osrm_distance_sum  segment_osrm_time_sum
0                     14.0                    11.9653                   11.0
1                     24.0                    21.7243                   20.0
2                     40.0                    32.5395                   27.0
3                     61.0                    45.5619                   39.0
4                     67.0                    49.4772                   44.0

[5 rows x 24 columns]
```

[55]:
```
df_grouped.isnull().sum()
```

[55]:
```
index                          0
segment_actual_time            0
segment_osrm_distance          0
segment_osrm_time              0
trip_creation_time         144867
route_schedule_uuid        144867
route_type                 144867
trip_uuid                  144867
source_center              144867
source_name                144867
destination_center         144867
destination_name           144867
```

```
od_start_time                        144867
od_end_time                          144867
start_scan_to_end_scan               144867
actual_distance_to_destination       144867
actual_time                          144867
osrm_time                            144867
osrm_distance                        144867
factor                                    0
segment_factor                            0
segment_actual_time_sum                   0
segment_osrm_distance_sum                 0
segment_osrm_time_sum                     0
dtype: int64
```

[59]:
```
# Forward-fill time columns
df_grouped['trip_creation_time'].ffill(inplace=True)
df_grouped['trip_creation_time'].bfill(inplace=True)
df_grouped['od_start_time'].ffill(inplace=True)
df_grouped['od_start_time'].bfill(inplace=True)
df_grouped['od_end_time'].ffill(inplace=True)
df_grouped['od_end_time'].bfill(inplace=True)
```

[63]:
```
# Fill missing categorical columns with 'Unknown' or the most frequent value
df_grouped['route_schedule_uuid'].fillna('Unknown', inplace=True)
df_grouped['route_type'].fillna('Unknown', inplace=True)
df_grouped['source_center'].fillna('Unknown', inplace=True)
df_grouped['source_name'].fillna('Unknown', inplace=True)
df_grouped['destination_center'].fillna('Unknown', inplace=True)
df_grouped['destination_name'].fillna('Unknown', inplace=True)
df_grouped['trip_uuid'].fillna('Unknown', inplace=True)
```

[64]:
```
# Fill missing numeric columns with the mean (or median)
df_grouped['start_scan_to_end_scan'].
 ↪fillna(df_grouped['start_scan_to_end_scan'].mean(), inplace=True)
df_grouped['actual_distance_to_destination'].
 ↪fillna(df_grouped['actual_distance_to_destination'].mean(), inplace=True)
df_grouped['actual_time'].fillna(df_grouped['actual_time'].mean(), inplace=True)
df_grouped['osrm_time'].fillna(df_grouped['osrm_time'].mean(), inplace=True)
df_grouped['osrm_distance'].fillna(df_grouped['osrm_distance'].mean(),␣
 ↪inplace=True)
```

[65]:
```
df_grouped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159684 entries, 0 to 159683
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
```

```
---  ------                          --------------   -----
 0   index                           159684 non-null  object
 1   segment_actual_time             159684 non-null  float64
 2   segment_osrm_distance           159684 non-null  float64
 3   segment_osrm_time               159684 non-null  float64
 4   trip_creation_time              159684 non-null  datetime64[ns]
 5   route_schedule_uuid             159684 non-null  object
 6   route_type                      159684 non-null  object
 7   trip_uuid                       159684 non-null  object
 8   source_center                   159684 non-null  object
 9   source_name                     159684 non-null  object
 10  destination_center              159684 non-null  object
 11  destination_name                159684 non-null  object
 12  od_start_time                   159684 non-null  datetime64[ns]
 13  od_end_time                     159684 non-null  datetime64[ns]
 14  start_scan_to_end_scan          159684 non-null  float64
 15  actual_distance_to_destination  159684 non-null  float64
 16  actual_time                     159684 non-null  float64
 17  osrm_time                       159684 non-null  float64
 18  osrm_distance                   159684 non-null  float64
 19  factor                          159684 non-null  float64
 20  segment_factor                  159684 non-null  float64
 21  segment_actual_time_sum         159684 non-null  float64
 22  segment_osrm_distance_sum       159684 non-null  float64
 23  segment_osrm_time_sum           159684 non-null  float64
dtypes: datetime64[ns](3), float64(13), object(8)
memory usage: 29.2+ MB
```

[67]:
```
# Calculate the difference and convert it to hours
df_grouped['od_time_diff_hour'] = (df_grouped['od_end_time'] -␣
  ↪df_grouped['od_start_time']).dt.total_seconds() / 3600
print(df_grouped['od_time_diff_hour'])
```

```
0         37.668497
1         37.668497
2         37.668497
3         37.668497
4         37.668497
             …
159679     6.758097
159680     1.009842
159681     7.035331
159682     5.808548
159683     5.906793
Name: od_time_diff_hour, Length: 159684, dtype: float64
```

[69]:
```
df_grouped['Destination_State'] = df_grouped['destination_name'].str.
 ↪extract(r'\((.*?)\)')  # Extract text within parentheses
destination_split = aggregated_df['destination_name'].str.split('_', n=1,␣
 ↪expand=True)  # Split on the first underscore
df_grouped['Destination_City'] = destination_split[0]
df_grouped['Destination_Place_Code'] = destination_split[1].str.replace(r' \(.
 ↪*\)', '', regex=True)  # Remove the state part

# Display the results
print(df_grouped[['Destination_City', 'Destination_Place_Code',␣
 ↪'Destination_State']].head())

# Splitting Source Name
df_grouped['Source_State'] = df_grouped['source_name'].str.extract(r'\((.*?
 ↪)\)')  # Extract text within parentheses
source_split = df_grouped['source_name'].str.split('_', n=1, expand=True)  #␣
 ↪Split on the first underscore
df_grouped['Source_City'] = source_split[0]
df_grouped['Source_Place_Code'] = source_split[1].str.replace(r' \(.*\)', '',␣
 ↪regex=True)  # Remove the state part

# Display the results
print(df_grouped[['Source_City', 'Source_Place_Code', 'Source_State']].head())
```

```
  Destination_City Destination_Place_Code Destination_State
0          Gurgaon            Bilaspur_HB               NaN
1           Kanpur            Central_H_6               NaN
2       Chikblapur             ShntiSgr_D               NaN
3       Doddablpur             ChikaDPP_D               NaN
4       Chandigarh             Mehmdpur_H               NaN
  Source_City Source_Place_Code Source_State
0     Unknown              None          NaN
1     Unknown              None          NaN
2     Unknown              None          NaN
3     Unknown              None          NaN
4     Unknown              None          NaN
```

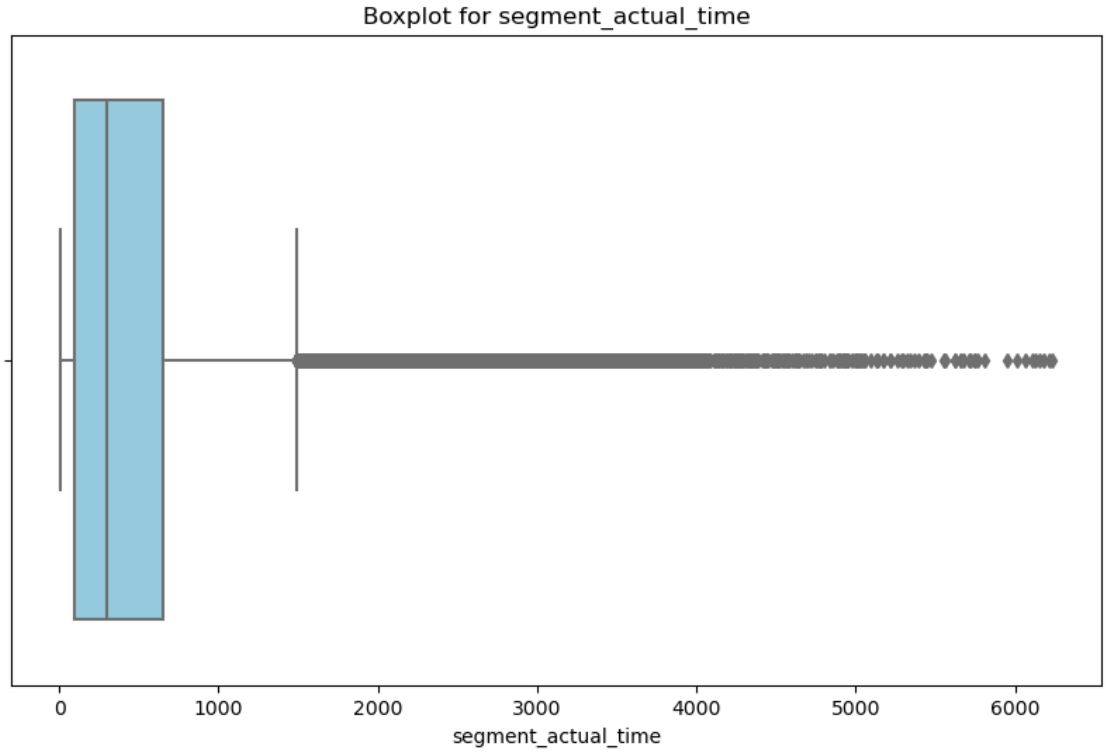
[71]:
```
# Group by trip_uuid and apply the aggregation functions
trip = df_grouped.groupby('trip_uuid').agg(create_segment_dict).reset_index()

# Sort the DataFrame by trip_uuid and od_end_time
df_grouped.sort_values(by=['trip_uuid', 'od_end_time'], ascending=[True, True],␣
 ↪inplace=True)

# Check the result
df_grouped.head()
```

[71]:
```
   index  segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0      0                 14.0                11.9653               11.0
1      1                 24.0                21.7243               20.0
2      2                 40.0                32.5395               27.0
3      3                 61.0                45.5619               39.0
4      4                 67.0                49.4772               44.0

           trip_creation_time route_schedule_uuid route_type trip_uuid  \
0 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
1 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
2 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
3 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
4 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown

  source_center source_name  … segment_actual_time_sum  \
0       Unknown     Unknown  …                   14.0
1       Unknown     Unknown  …                   24.0
2       Unknown     Unknown  …                   40.0
3       Unknown     Unknown  …                   61.0
4       Unknown     Unknown  …                   67.0

  segment_osrm_distance_sum segment_osrm_time_sum od_time_diff_hour  \
0                   11.9653                  11.0         37.668497
1                   21.7243                  20.0         37.668497
2                   32.5395                  27.0         37.668497
3                   45.5619                  39.0         37.668497
4                   49.4772                  44.0         37.668497

   Destination_State  Destination_City  Destination_Place_Code  Source_State  \
0                NaN           Gurgaon             Bilaspur_HB           NaN
1                NaN            Kanpur             Central_H_6           NaN
2                NaN        Chikblapur              ShntiSgr_D           NaN
3                NaN        Doddablpur              ChikaDPP_D           NaN
4                NaN        Chandigarh              Mehmdpur_H           NaN

   Source_City  Source_Place_Code
0      Unknown               None
1      Unknown               None
2      Unknown               None
3      Unknown               None
4      Unknown               None

[5 rows x 31 columns]
```

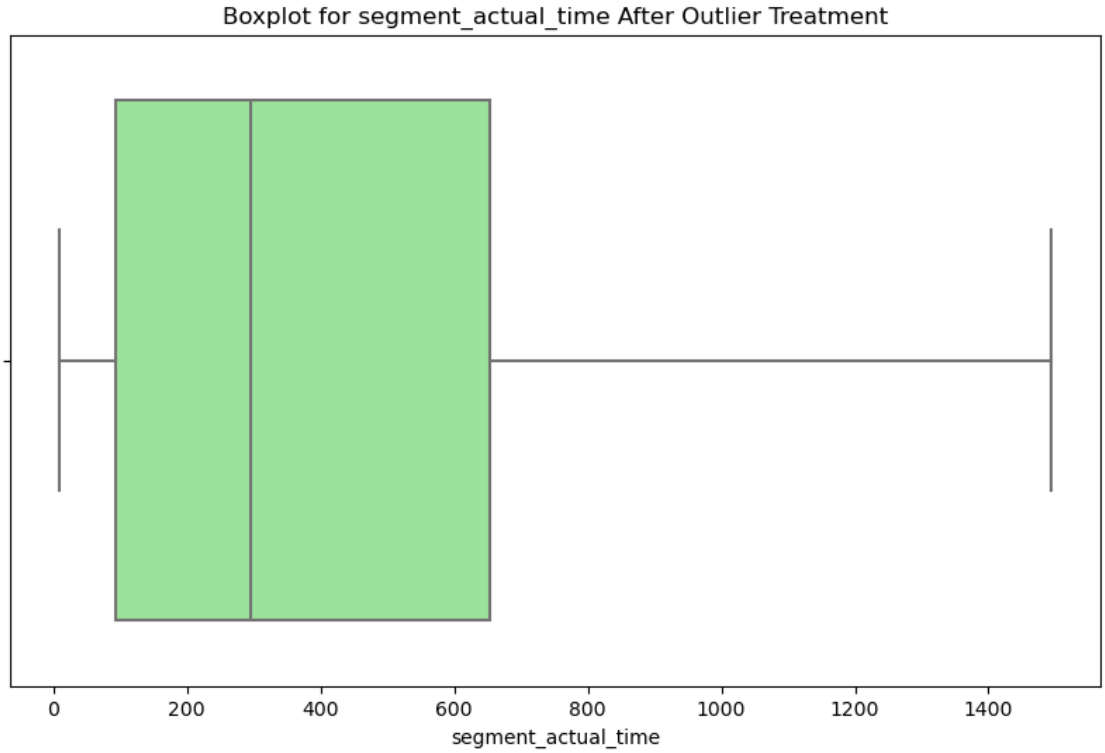
[72]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Select all numerical features
numerical_features = df_grouped.select_dtypes(include=['float64', 'int64']).
 ↪columns

# Function to detect and treat outliers for each numerical feature
for feature in numerical_features:
    print(f"Processing Feature: {feature}")

    # Visualize outliers using a boxplot
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=df_grouped, x=feature, color='skyblue')
    plt.title(f'Boxplot for {feature}')
    plt.xlabel(feature)
    plt.show()

    # Calculate IQR
    Q1 = df_grouped[feature].quantile(0.25)
    Q3 = df_grouped[feature].quantile(0.75)
    IQR = Q3 - Q1

    # Define bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    print(f"{feature}: Lower Bound = {lower_bound}, Upper Bound =␣
 ↪{upper_bound}")

    # Identify outliers
    outliers = df_grouped[(df_grouped[feature] < lower_bound) |␣
 ↪(df_grouped[feature] > upper_bound)]
    print(f"Number of Outliers in {feature}: {len(outliers)}")

    # Treat outliers by capping
    df_grouped[feature] = np.where(
        df_grouped[feature] < lower_bound,
        lower_bound,
        np.where(df_grouped[feature] > upper_bound, upper_bound,␣
 ↪df_grouped[feature])
    )

    # Verify after treatment with boxplot
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=df_grouped, x=feature, color='lightgreen')
    plt.title(f'Boxplot for {feature} After Outlier Treatment')
    plt.xlabel(feature)
    plt.show()
```

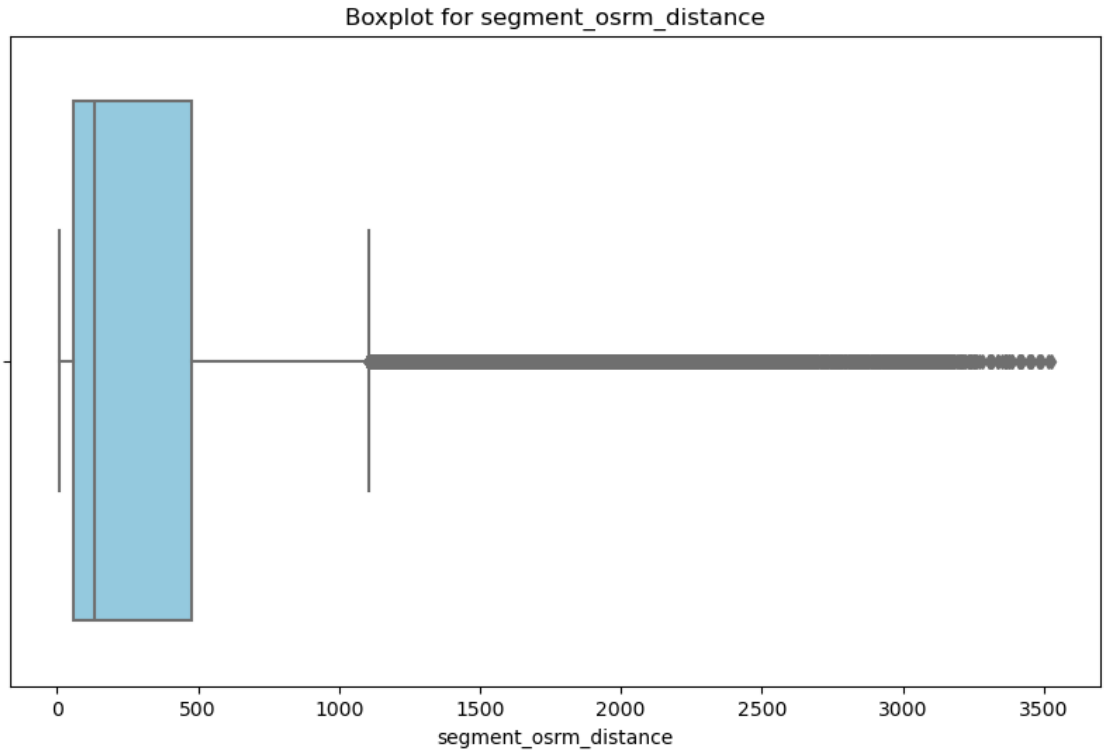
```
Processing Feature: segment_actual_time
```



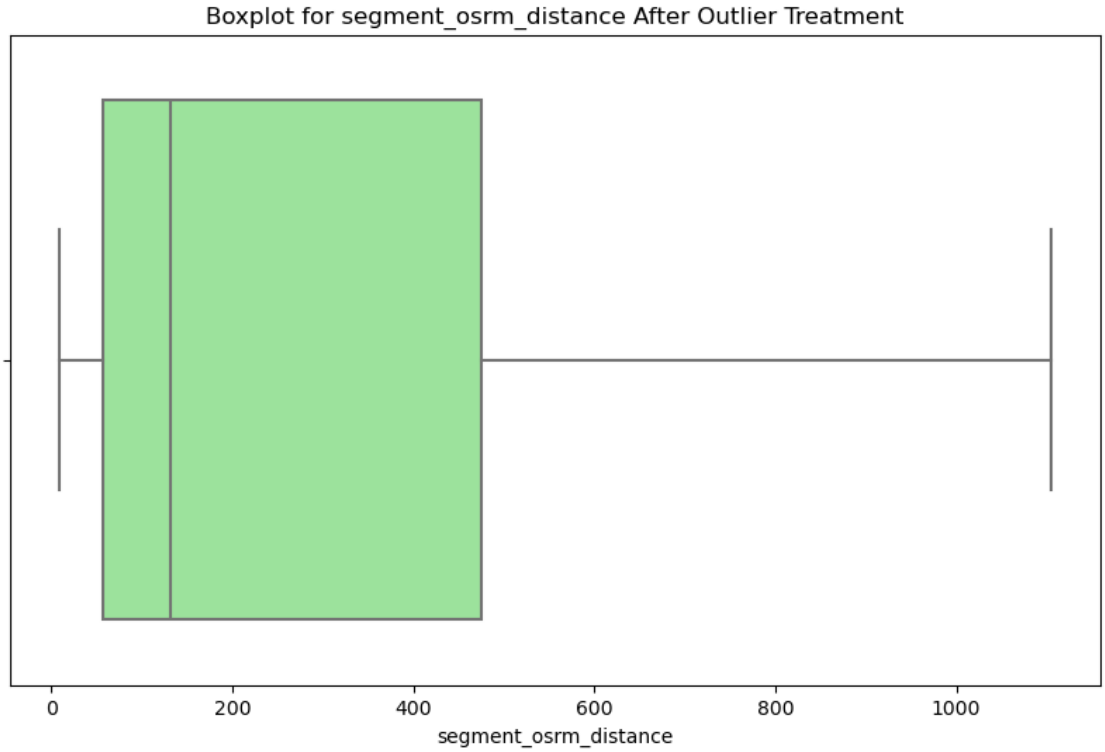

```
segment_actual_time: Lower Bound = -747.0, Upper Bound = 1493.0
Number of Outliers in segment_actual_time: 15846
```



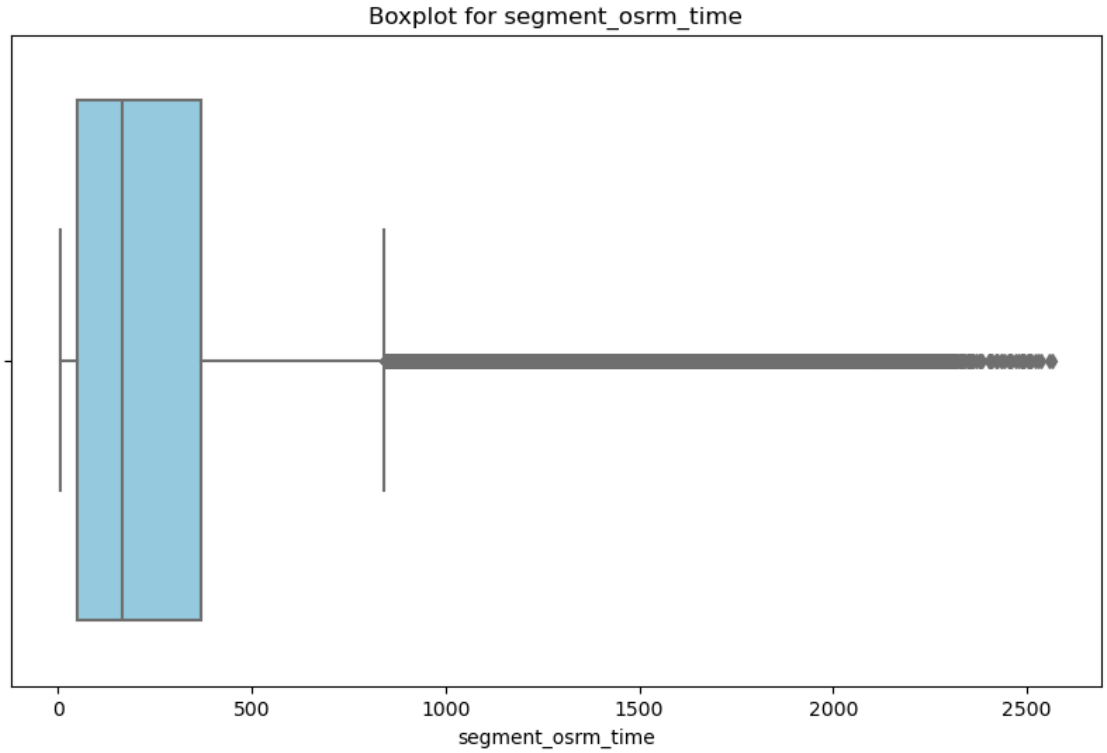
Processing Feature: segment_osrm_distance



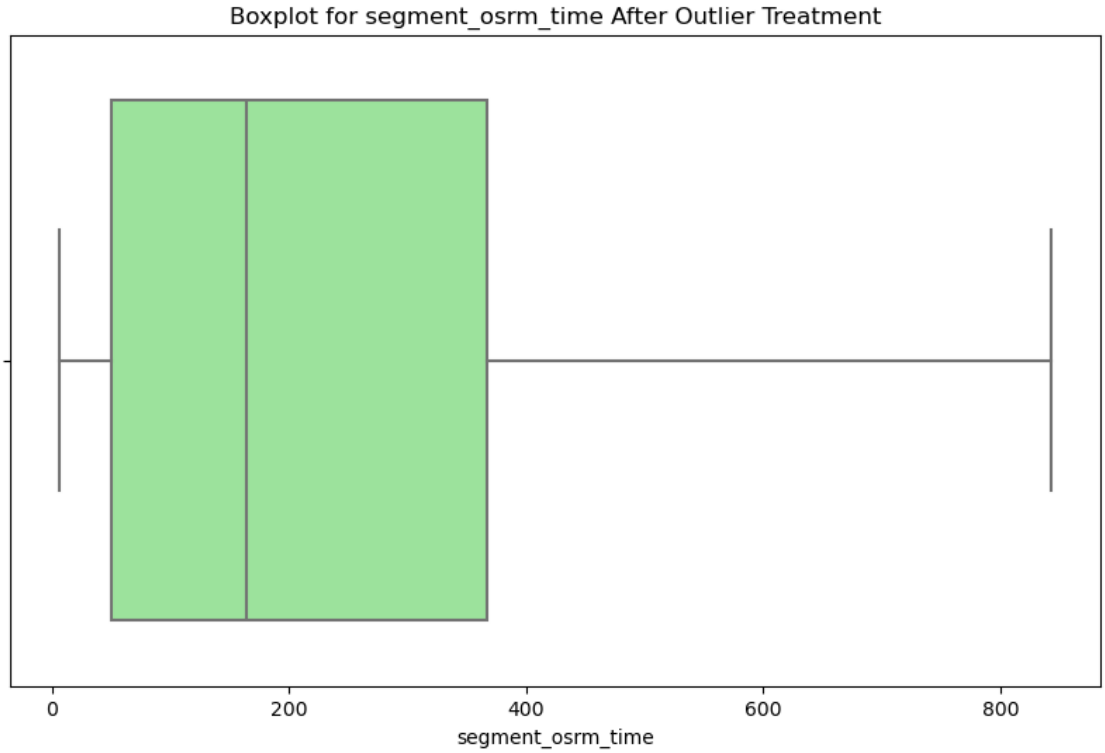
```
segment_osrm_distance: Lower Bound = -572.15005, Upper Bound = 1104.30815
Number of Outliers in segment_osrm_distance: 18006
```



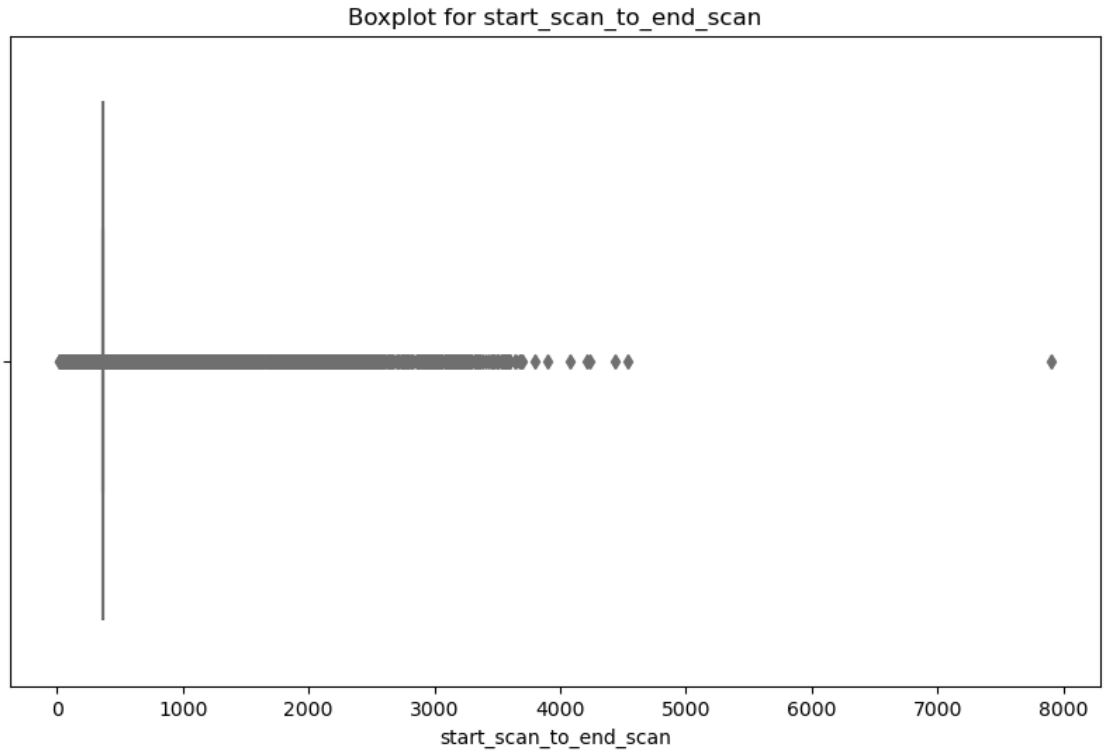

```
Processing Feature: segment_osrm_time
```



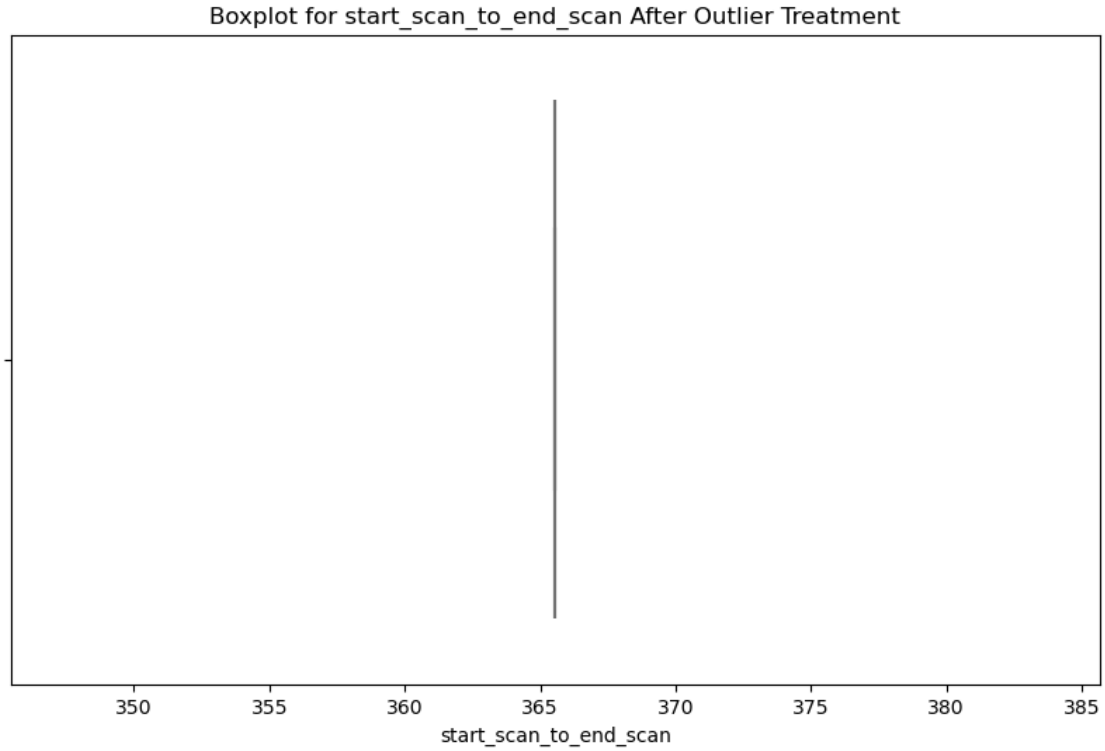

```
segment_osrm_time: Lower Bound = -425.5, Upper Bound = 842.5
Number of Outliers in segment_osrm_time: 17637
```



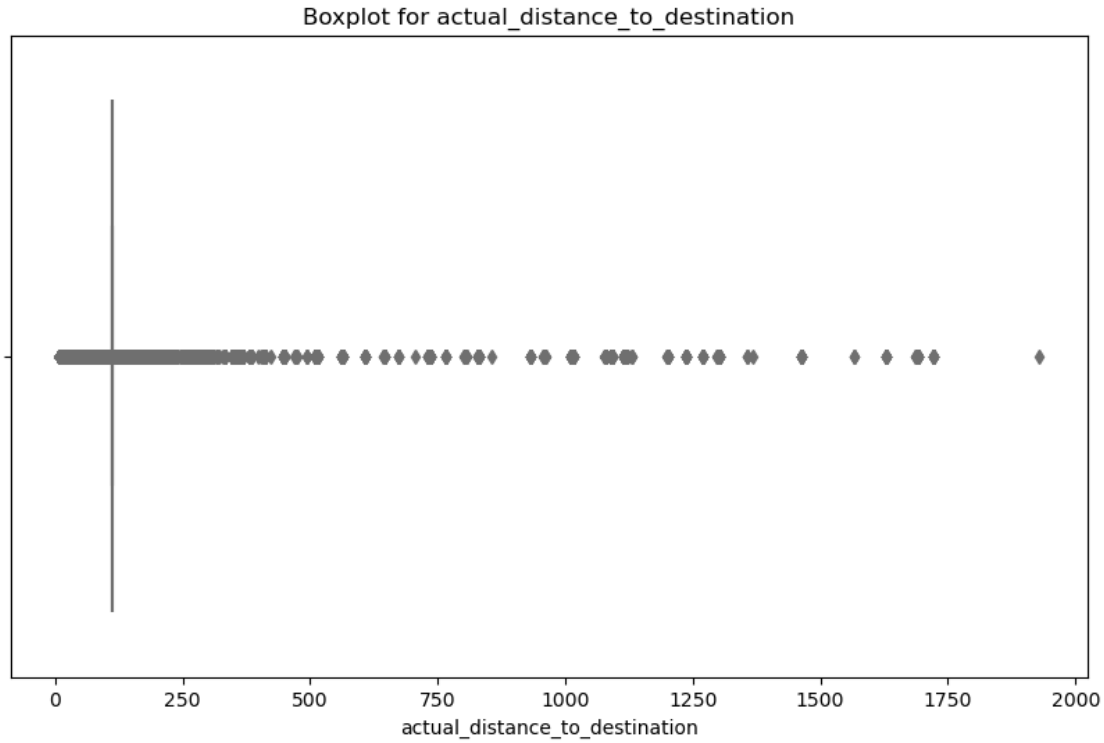

```
Processing Feature: start_scan_to_end_scan
```



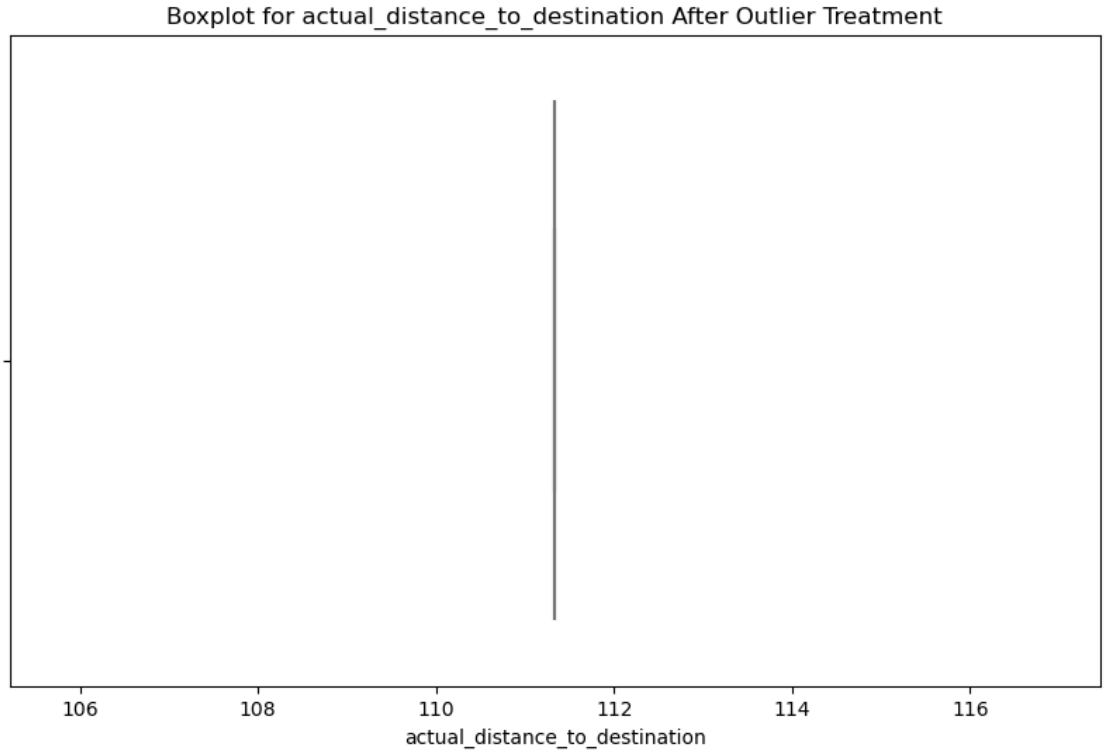
```
start_scan_to_end_scan: Lower Bound = 365.56266450698524, Upper Bound =
365.56266450698524
Number of Outliers in start_scan_to_end_scan: 14817
```



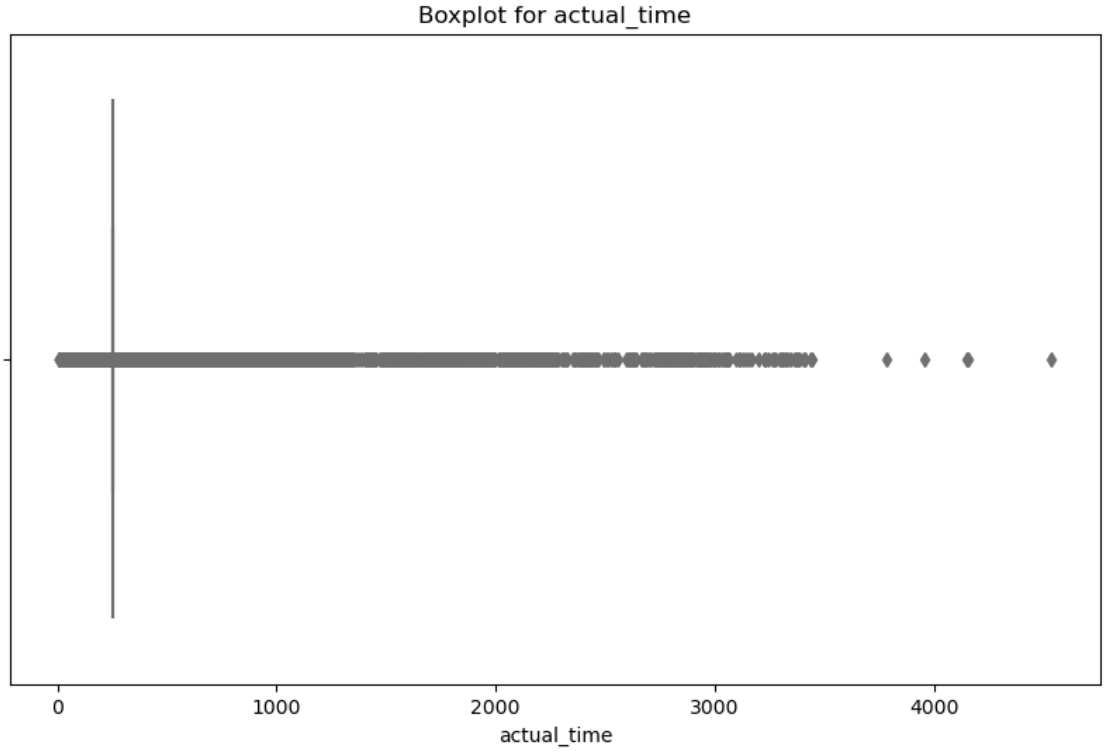

```
Processing Feature: actual_distance_to_destination
```



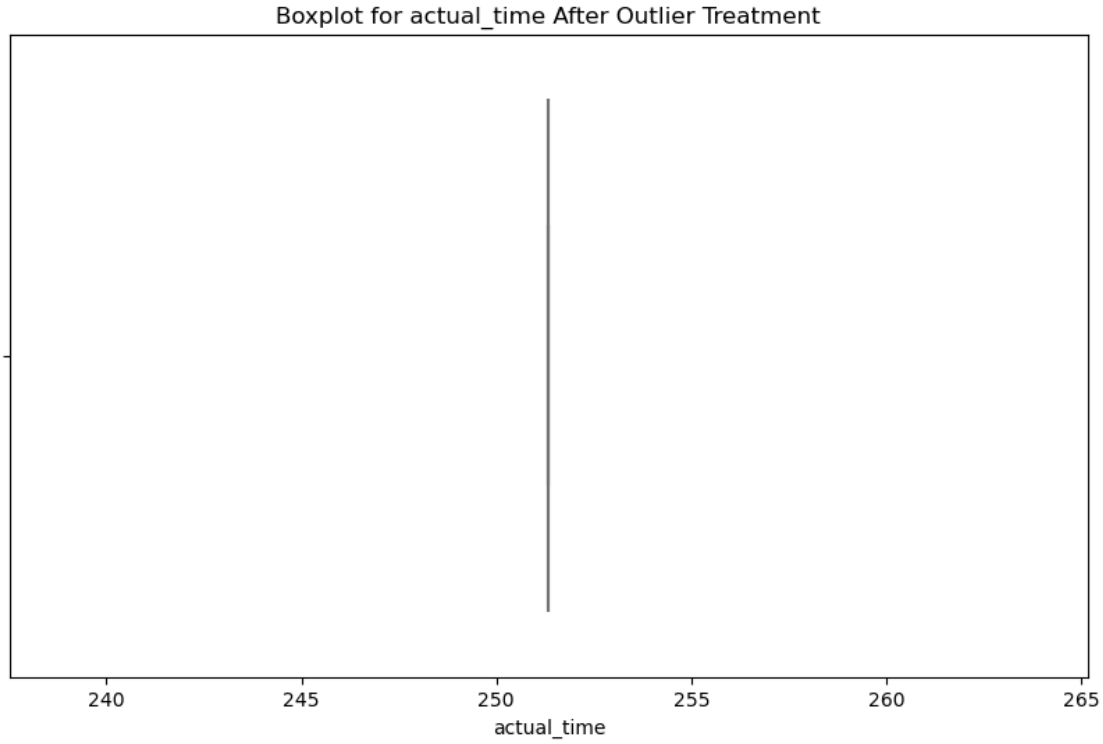

```
actual_distance_to_destination: Lower Bound = 111.33615455284881, Upper Bound =
111.33615455284881
Number of Outliers in actual_distance_to_destination: 14817
```



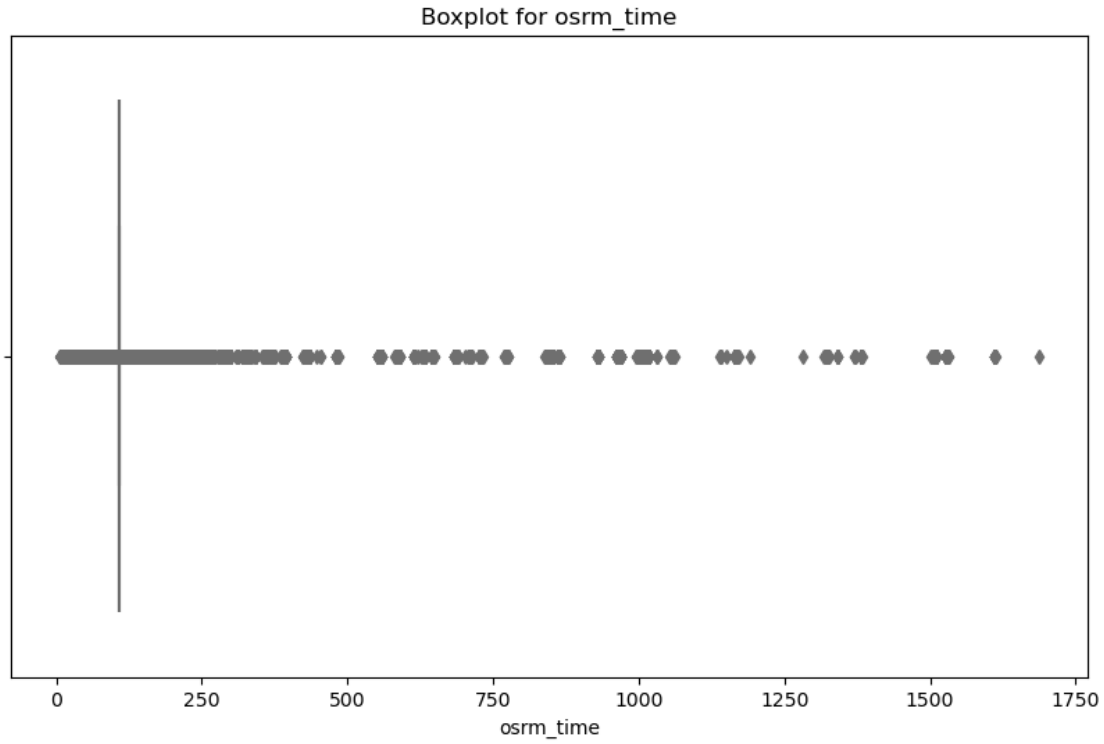

```
Processing Feature: actual_time
```



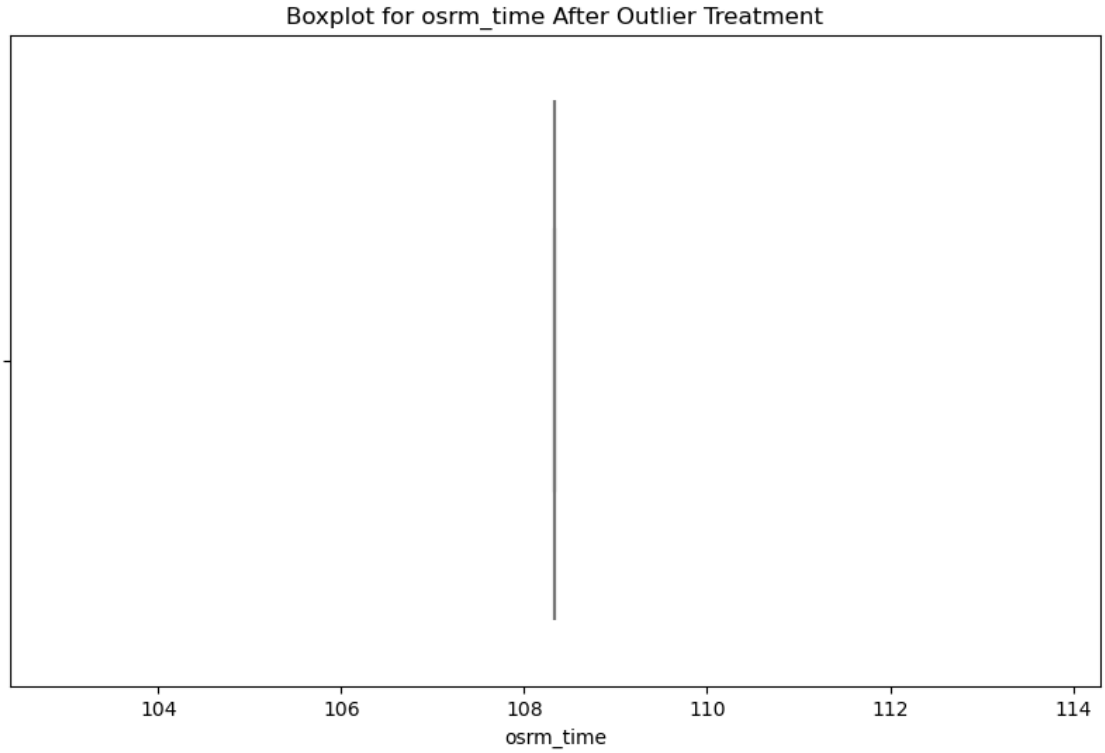
```
actual_time: Lower Bound = 251.34338935007085, Upper Bound = 251.34338935007085
Number of Outliers in actual_time: 14817
```



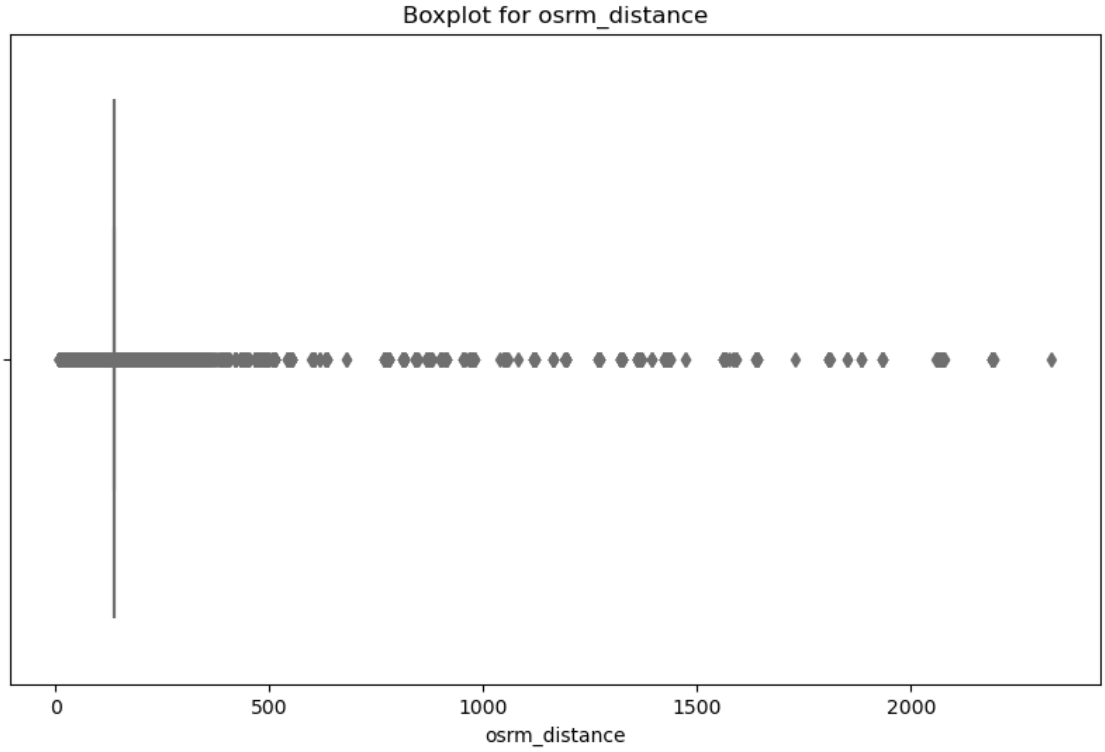

```
Processing Feature: osrm_time
```



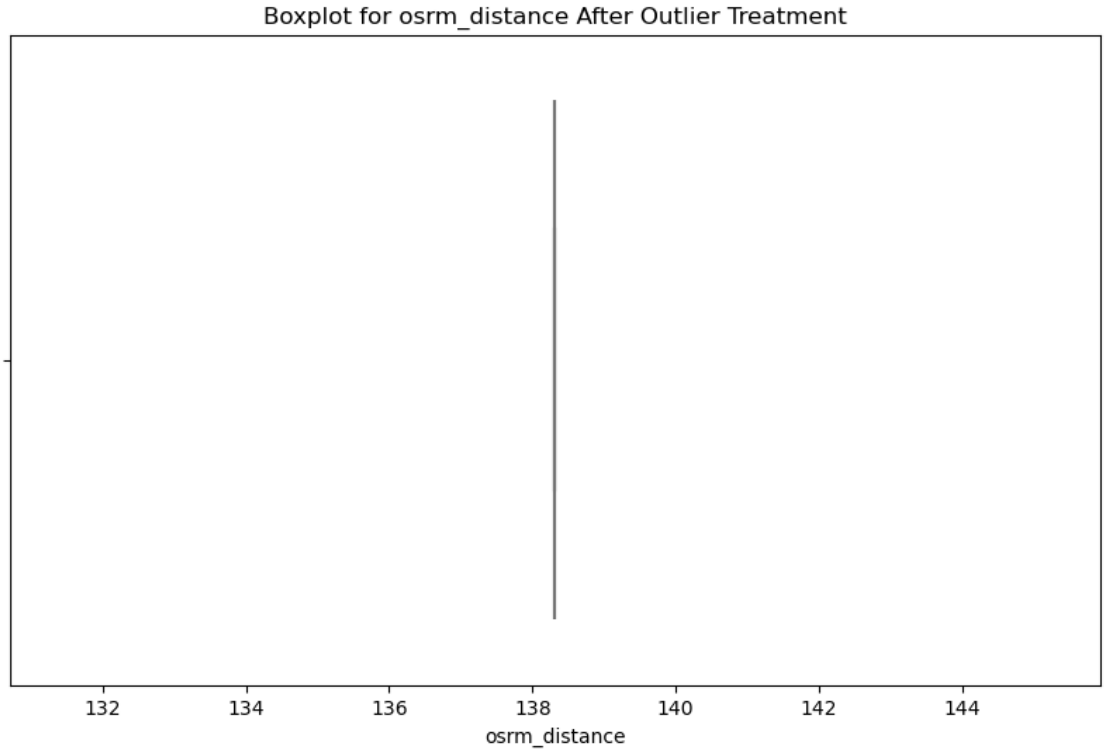

```
osrm_time: Lower Bound = 108.33718026591077, Upper Bound = 108.33718026591077
Number of Outliers in osrm_time: 14817
```



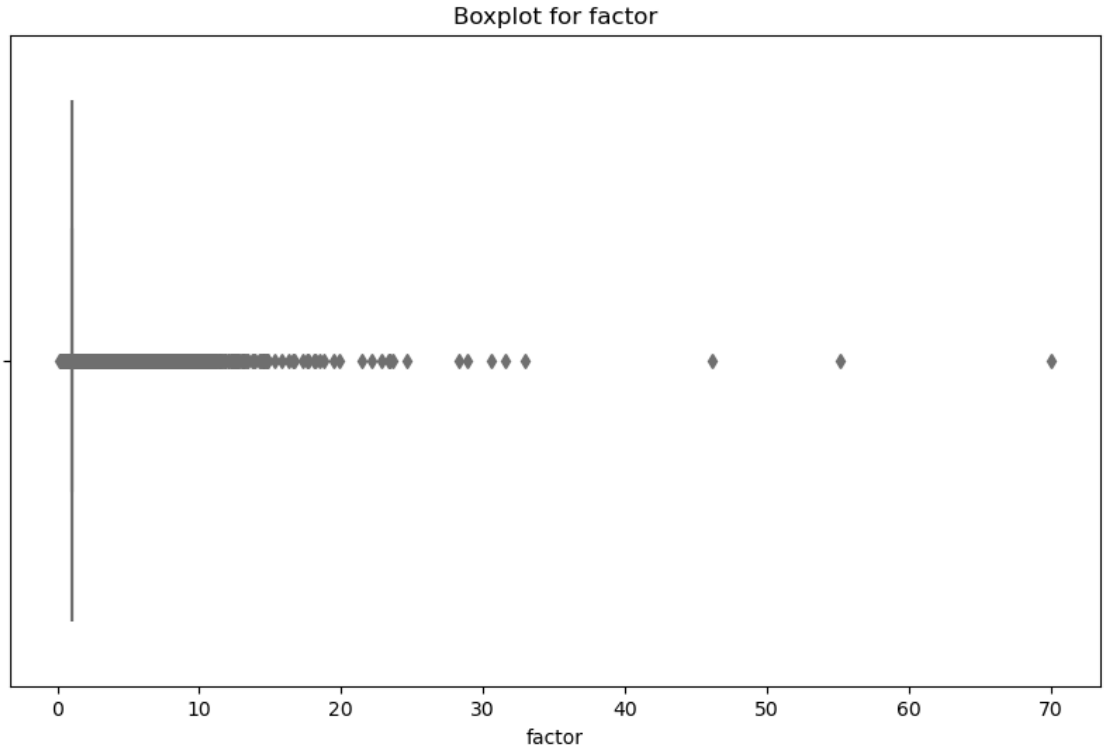
Processing Feature: osrm_distance



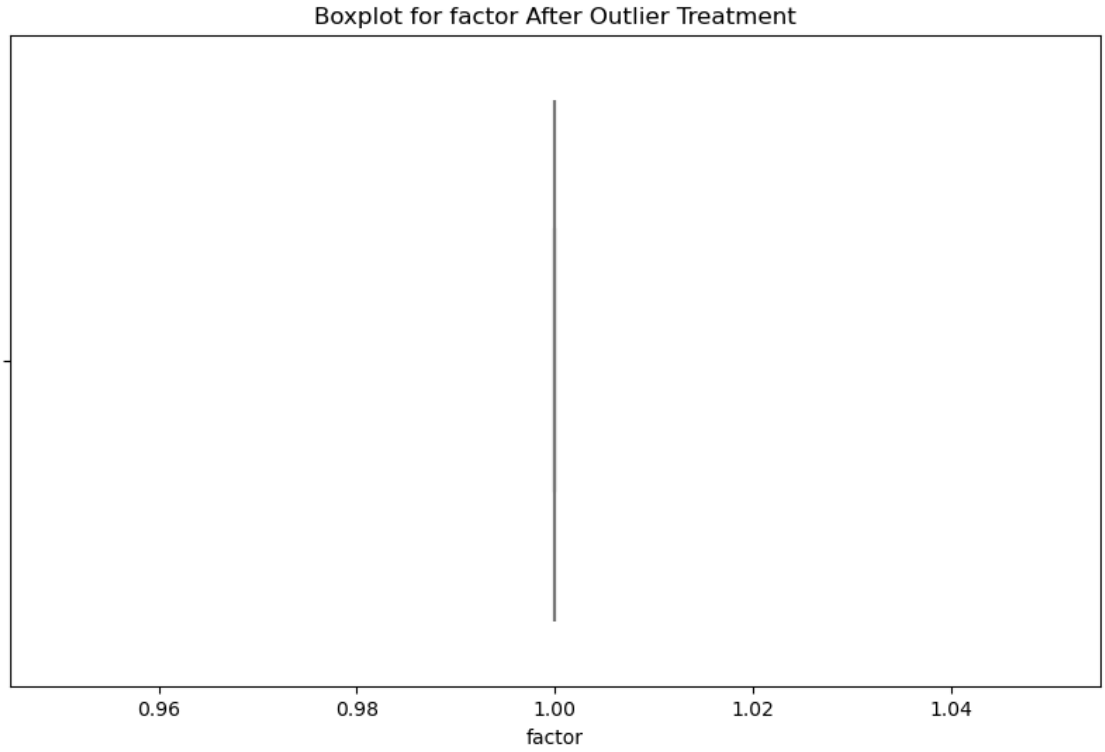
```
osrm_distance: Lower Bound = 138.31516591077815, Upper Bound =
138.31516591077815
Number of Outliers in osrm_distance: 14817
```



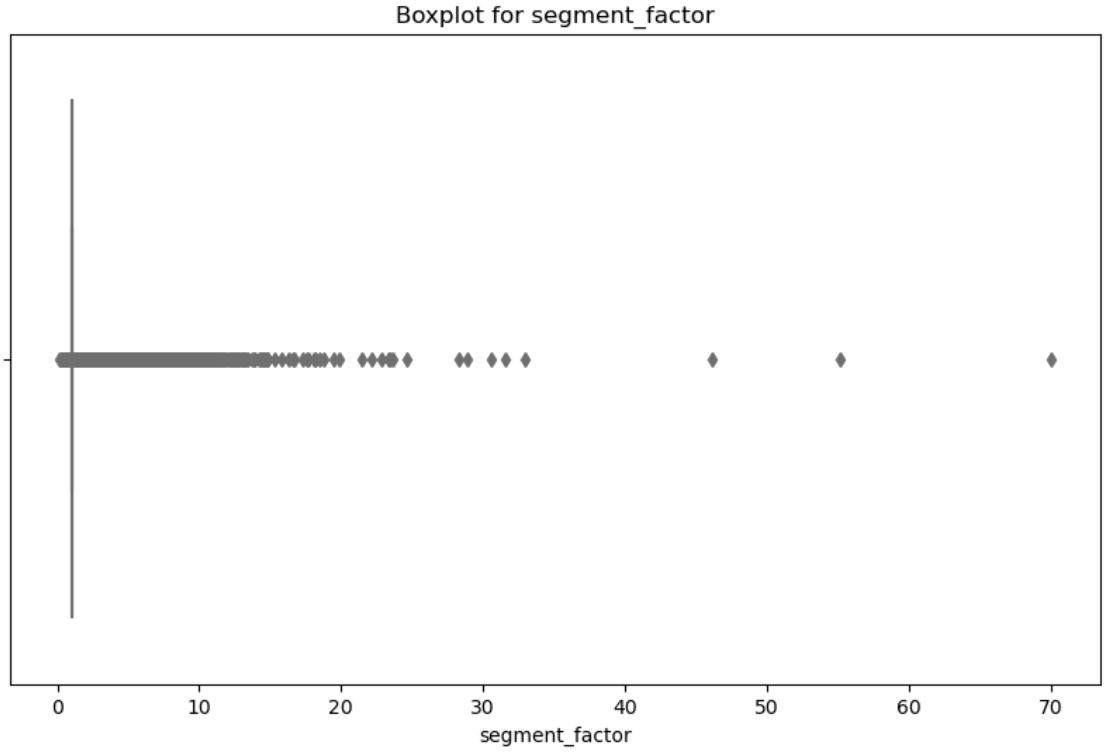

```
Processing Feature: factor
```



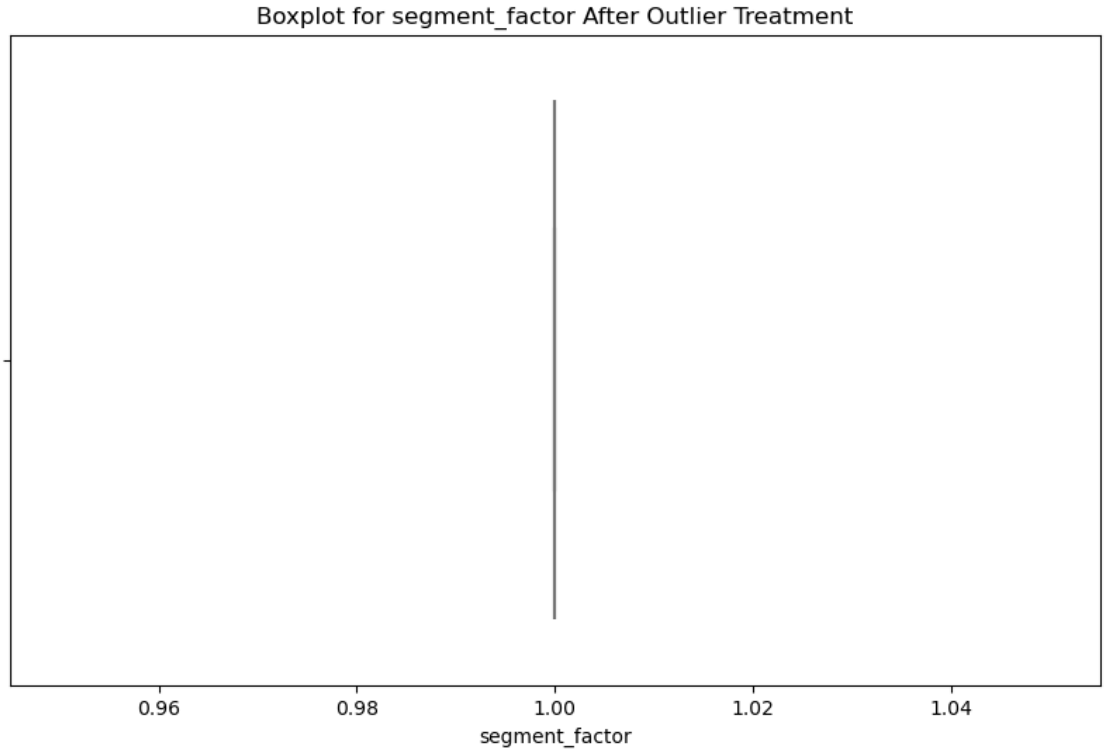

```
factor: Lower Bound = 1.0, Upper Bound = 1.0
Number of Outliers in factor: 14614
```



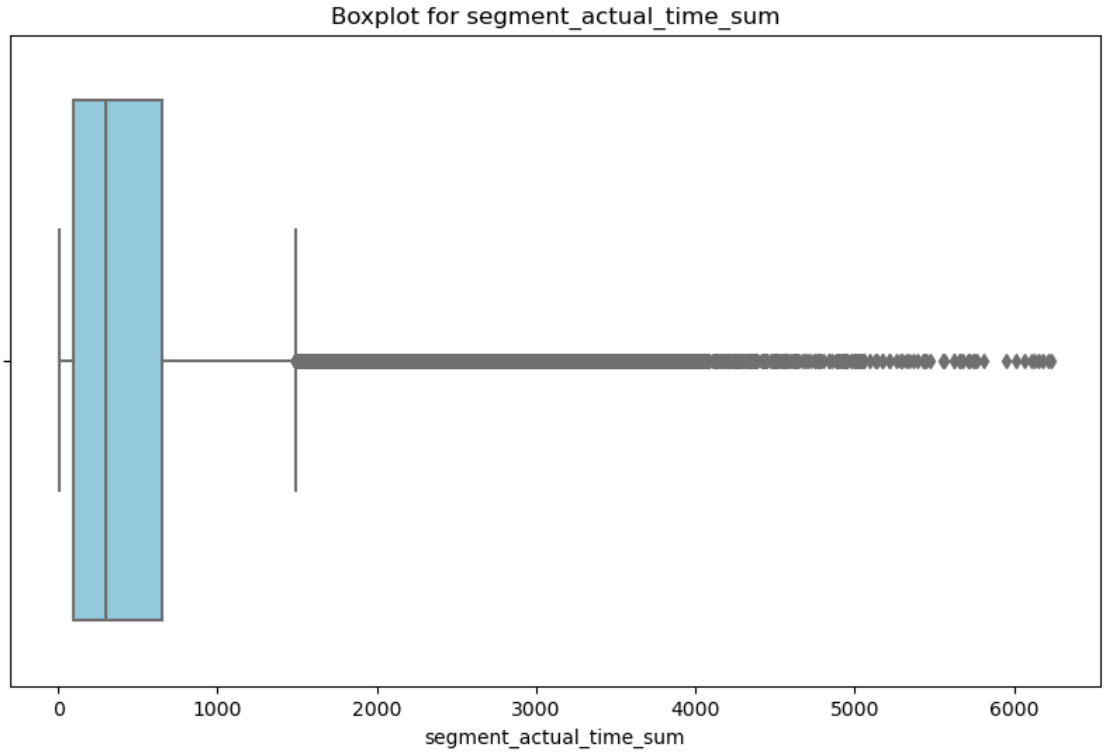

```
Processing Feature: segment_factor
```



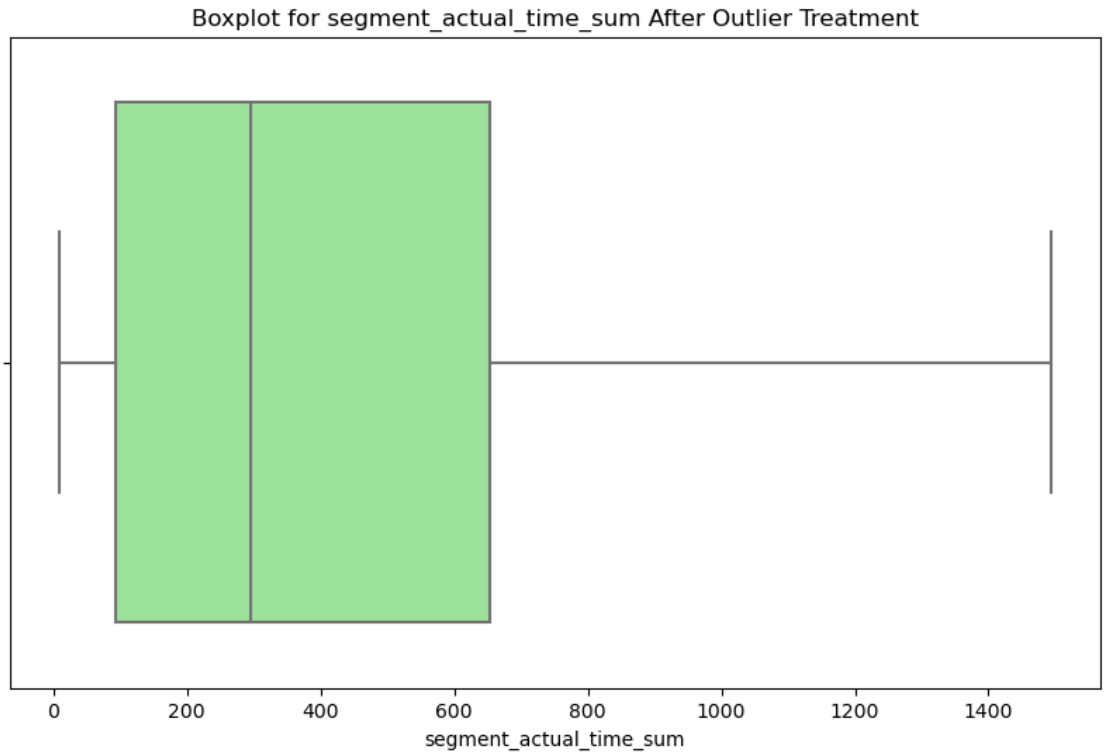
```
segment_factor: Lower Bound = 1.0, Upper Bound = 1.0
Number of Outliers in segment_factor: 14614
```



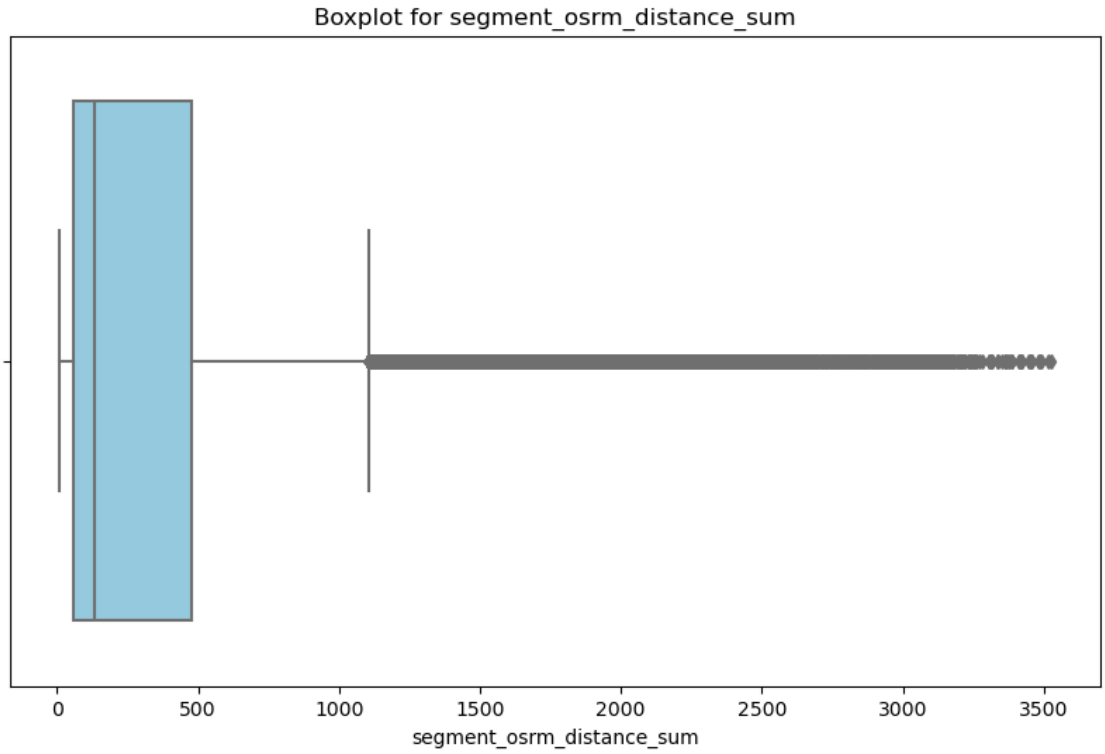

```
Processing Feature: segment_actual_time_sum
```



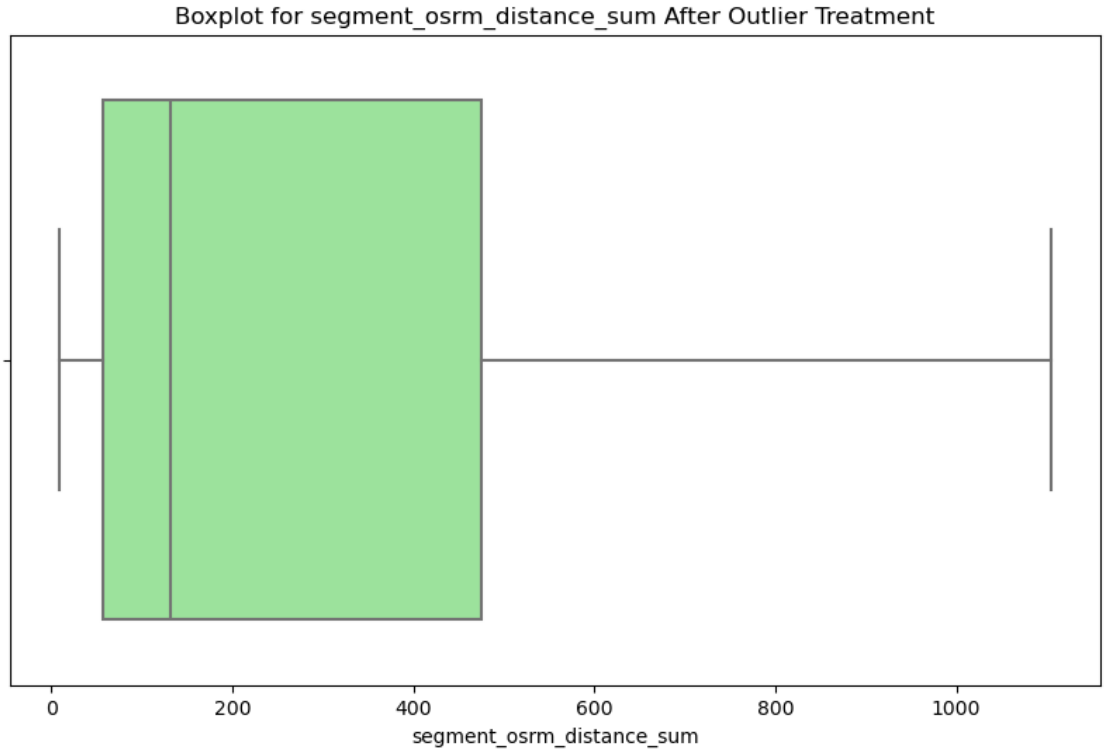

```
segment_actual_time_sum: Lower Bound = -747.0, Upper Bound = 1493.0
Number of Outliers in segment_actual_time_sum: 15846
```



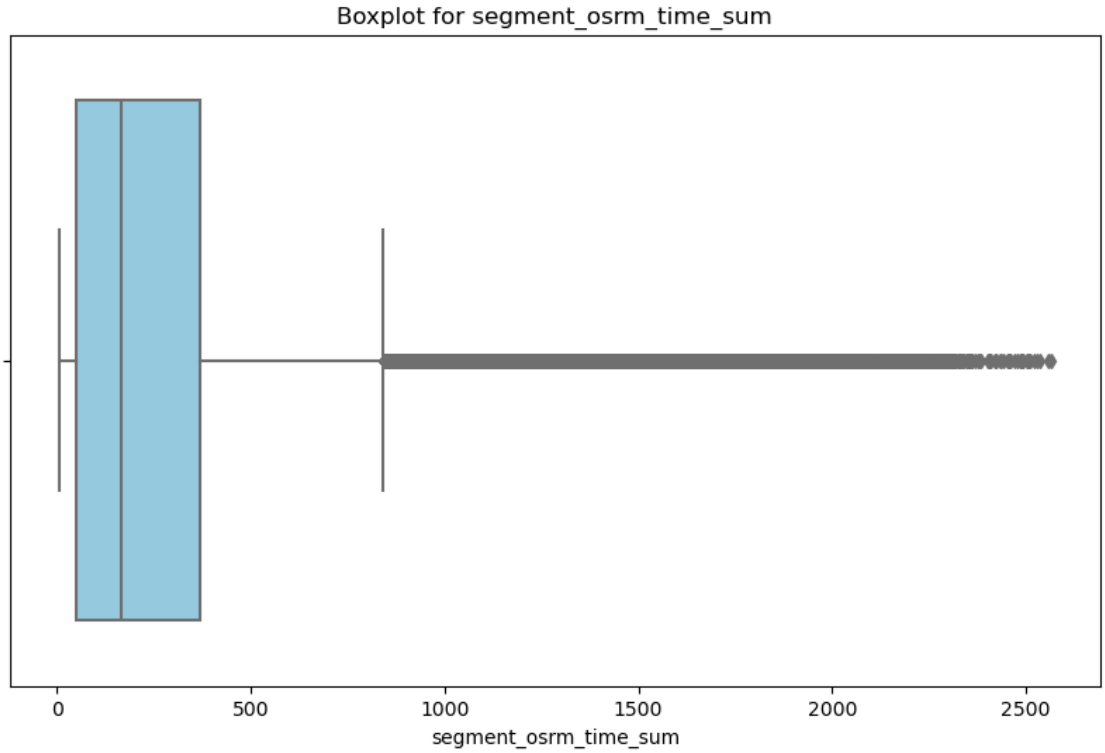
Processing Feature: segment_osrm_distance_sum



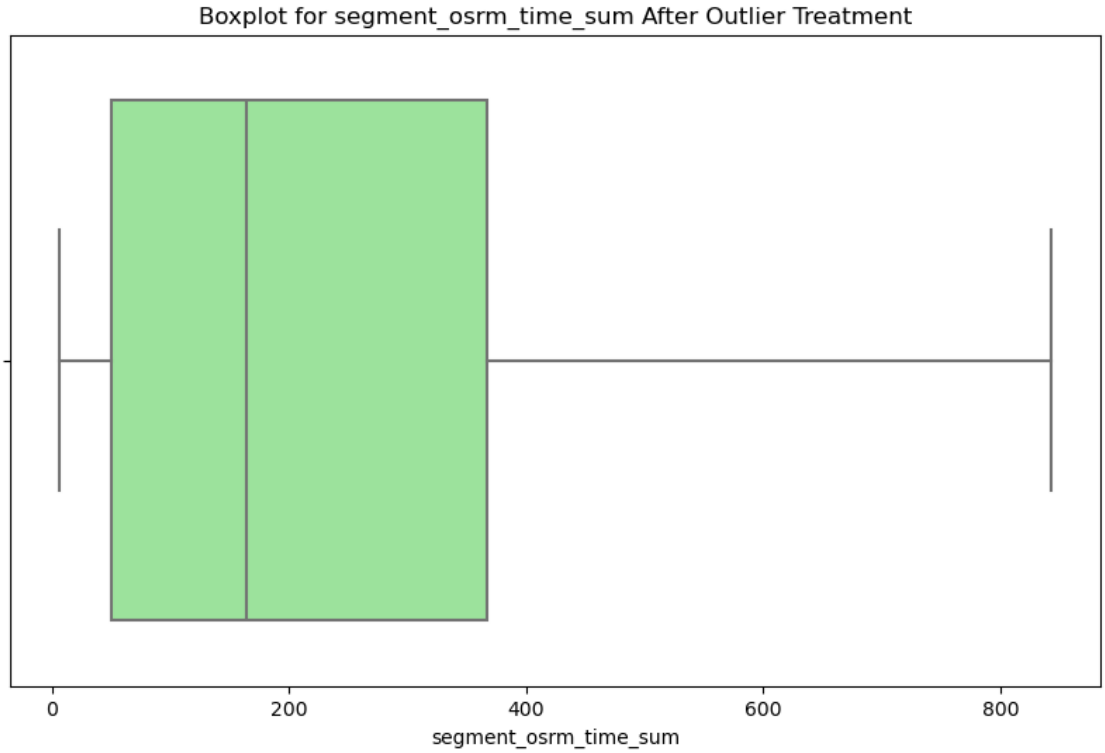
```
segment_osrm_distance_sum: Lower Bound = -572.15005, Upper Bound = 1104.30815
Number of Outliers in segment_osrm_distance_sum: 18006
```



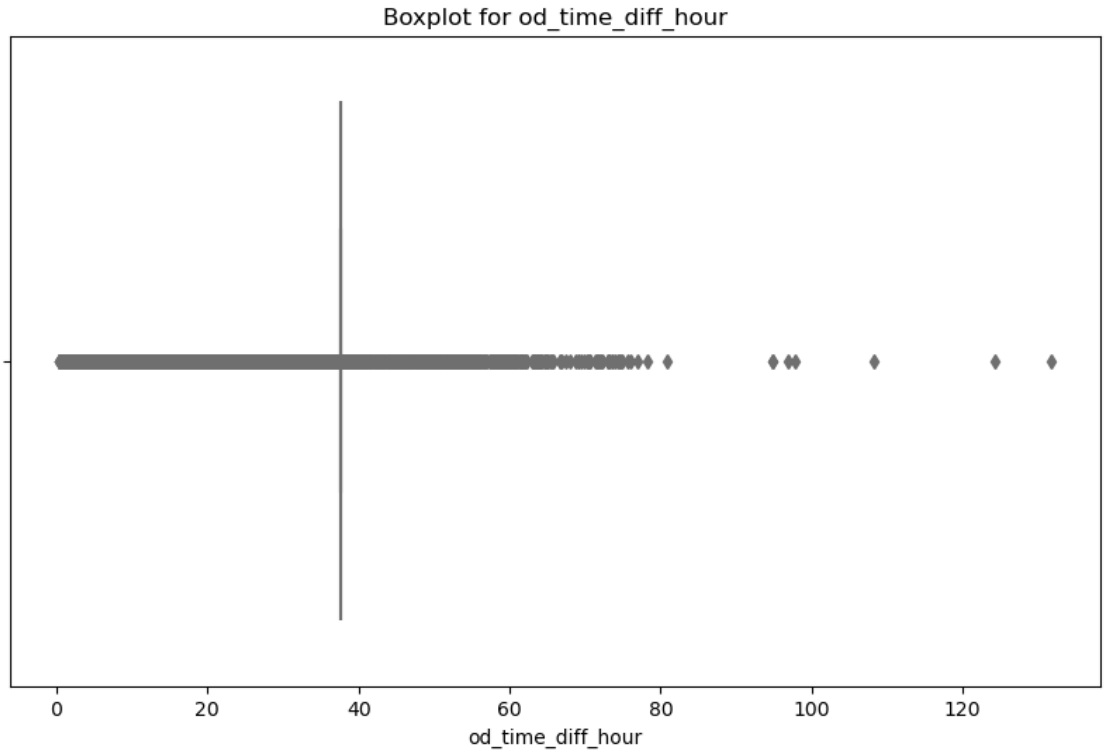

```
Processing Feature: segment_osrm_time_sum
```



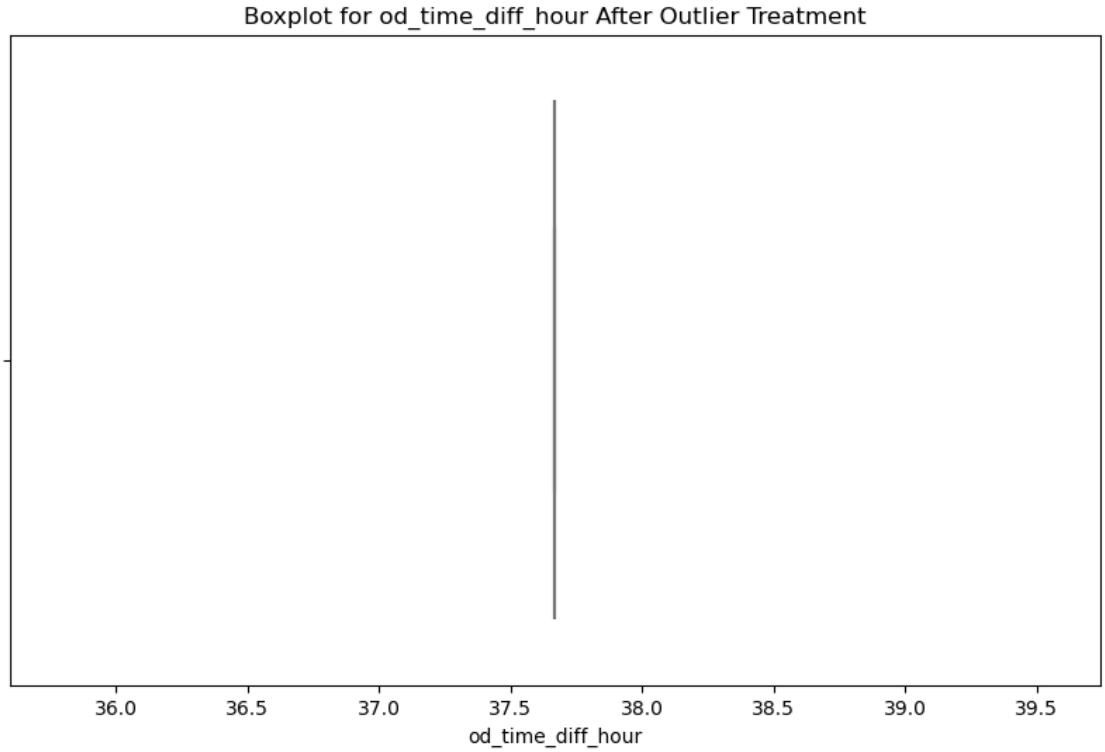

```
segment_osrm_time_sum: Lower Bound = -425.5, Upper Bound = 842.5
Number of Outliers in segment_osrm_time_sum: 17637
```




```
Processing Feature: od_time_diff_hour
```



```
od_time_diff_hour: Lower Bound = 37.6684966675, Upper Bound = 37.6684966675
Number of Outliers in od_time_diff_hour: 14816
```



[74]:
```
# Identify categorical features
categorical_features = df_grouped.select_dtypes(include=['object']).columns

# Check cardinality of each categorical feature
cardinality = {col: df_grouped[col].nunique() for col in categorical_features}
print("Cardinality of Features:", cardinality)

# Select features with low cardinality for one-hot encoding
low_cardinality_features = [col for col, count in cardinality.items() if count␣
 ↪< 50]  # Adjust threshold as needed
print("Features Selected for One-Hot Encoding:", low_cardinality_features)

# Apply one-hot encoding to selected features
df_encoded = pd.get_dummies(df_grouped, columns=low_cardinality_features,␣
 ↪drop_first=True)

# Display results
```

```
print("Shape Before Encoding:", df_grouped.shape)
print("Shape After Encoding:", df_encoded.shape)
```

```
Cardinality of Features: {'index': 159684, 'route_schedule_uuid': 1505,
'route_type': 3, 'trip_uuid': 14818, 'source_center': 869, 'source_name': 865,
'destination_center': 1109, 'destination_name': 1103, 'Destination_State': 32,
'Destination_City': 1259, 'Destination_Place_Code': 1178, 'Source_State': 29,
'Source_City': 680, 'Source_Place_Code': 687}
Features Selected for One-Hot Encoding: ['route_type', 'Destination_State',
'Source_State']
Shape Before Encoding: (159684, 31)
Shape After Encoding: (159684, 89)
```

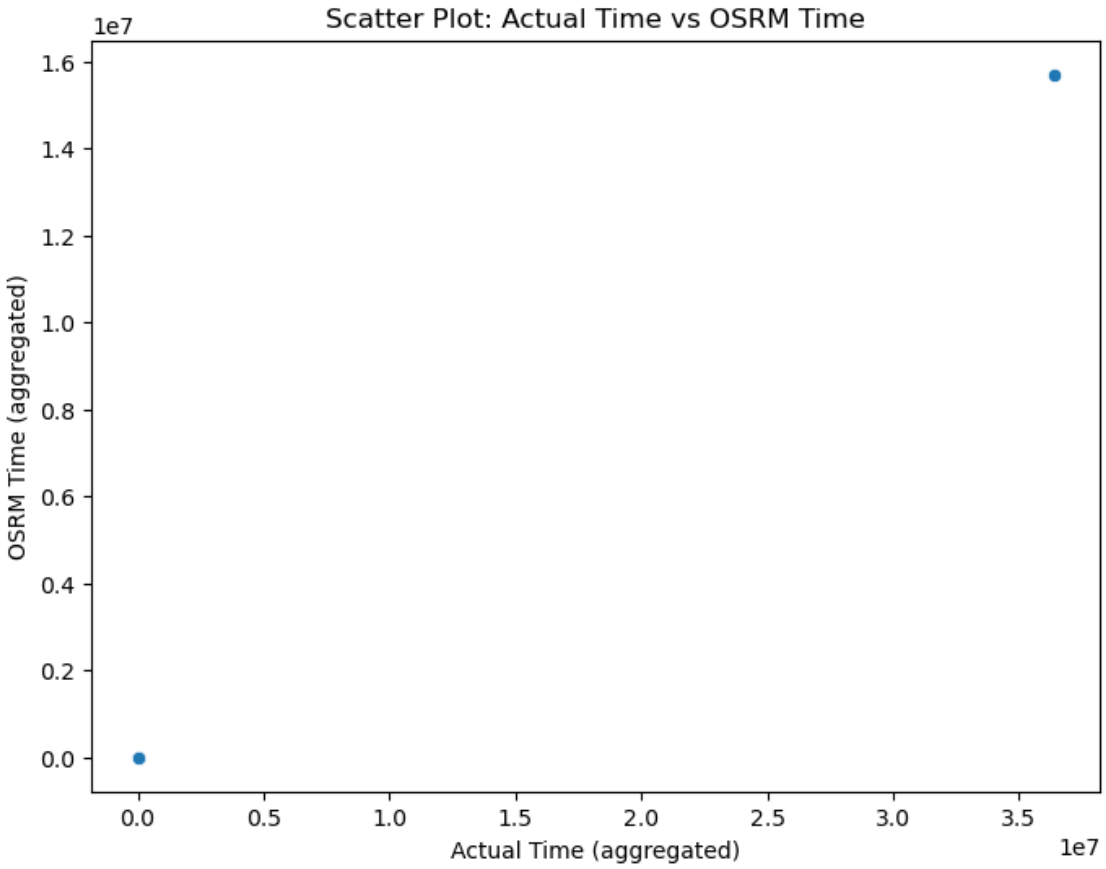
[75]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Identify numerical features to scale (all features of dtype float64)
numerical_features = df_grouped.select_dtypes(include=['float64']).columns

print("Numerical Features to Normalize/Standardize:", numerical_features)

# Create Scalers
min_max_scaler = MinMaxScaler()
standard_scaler = StandardScaler()

# Normalize the numerical features using MinMaxScaler (scales values between 0␣
 ↪and 1)
df_normalized = df_grouped.copy()
df_normalized[numerical_features] = min_max_scaler.
 ↪fit_transform(df_grouped[numerical_features])

# Standardize the numerical features using StandardScaler (mean=0, std=1)
df_standardized = df_grouped.copy()
df_standardized[numerical_features] = standard_scaler.
 ↪fit_transform(df_grouped[numerical_features])

# Display the results
print("Shape Before Scaling:", df_grouped.shape)
print("Shape After Normalization:", df_normalized.shape)
print("Shape After Standardization:", df_standardized.shape)

# Check the first few rows after scaling
print(df_normalized.head())
print(df_standardized.head())
```

```
Numerical Features to Normalize/Standardize: Index(['segment_actual_time',
'segment_osrm_distance', 'segment_osrm_time',
       'start_scan_to_end_scan', 'actual_distance_to_destination',
```

```
       'actual_time', 'osrm_time', 'osrm_distance', 'factor', 'segment_factor',
       'segment_actual_time_sum', 'segment_osrm_distance_sum',
       'segment_osrm_time_sum', 'od_time_diff_hour'],
      dtype='object')
Shape Before Scaling: (159684, 31)
Shape After Normalization: (159684, 31)
Shape After Standardization: (159684, 31)
  index  segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0     0             0.003369               0.002661           0.005977
1     1             0.010108               0.011571           0.016736
2     2             0.020889               0.021446           0.025105
3     3             0.035040               0.033335           0.039450
4     4             0.039084               0.036910           0.045427

          trip_creation_time route_schedule_uuid route_type trip_uuid  \
0 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
1 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
2 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
3 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
4 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown

  source_center source_name  … segment_actual_time_sum  \
0       Unknown     Unknown  …                0.003369
1       Unknown     Unknown  …                0.010108
2       Unknown     Unknown  …                0.020889
3       Unknown     Unknown  …                0.035040
4       Unknown     Unknown  …                0.039084

  segment_osrm_distance_sum segment_osrm_time_sum od_time_diff_hour  \
0                  0.002661              0.005977               0.0
1                  0.011571              0.016736               0.0
2                  0.021446              0.025105               0.0
3                  0.033335              0.039450               0.0
4                  0.036910              0.045427               0.0

   Destination_State  Destination_City  Destination_Place_Code  Source_State  \
0                NaN           Gurgaon             Bilaspur_HB           NaN
1                NaN            Kanpur             Central_H_6           NaN
2                NaN         Chikblapur              ShntiSgr_D           NaN
3                NaN         Doddablpur              ChikaDPP_D           NaN
4                NaN         Chandigarh              Mehmdpur_H           NaN

   Source_City  Source_Place_Code
0      Unknown               None
1      Unknown               None
2      Unknown               None
3      Unknown               None
4      Unknown               None
```

```
[5 rows x 31 columns]
  index  segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0     0            -0.953886              -0.850437          -0.906255
1     1            -0.932782              -0.823832          -0.873415
2     2            -0.899016              -0.794348          -0.847872
3     3            -0.854697              -0.758847          -0.804085
4     4            -0.842035              -0.748174          -0.785840

          trip_creation_time route_schedule_uuid route_type trip_uuid  \
0 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
1 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
2 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
3 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown
4 2018-09-12 00:00:16.535741             Unknown    Unknown   Unknown

  source_center source_name  … segment_actual_time_sum  \
0       Unknown     Unknown  …               -0.953886
1       Unknown     Unknown  …               -0.932782
2       Unknown     Unknown  …               -0.899016
3       Unknown     Unknown  …               -0.854697
4       Unknown     Unknown  …               -0.842035

  segment_osrm_distance_sum segment_osrm_time_sum od_time_diff_hour  \
0                 -0.850437             -0.906255               0.0
1                 -0.823832             -0.873415               0.0
2                 -0.794348             -0.847872               0.0
3                 -0.758847             -0.804085               0.0
4                 -0.748174             -0.785840               0.0

   Destination_State  Destination_City  Destination_Place_Code  Source_State  \
0                NaN           Gurgaon             Bilaspur_HB           NaN
1                NaN            Kanpur             Central_H_6           NaN
2                NaN        Chikblapur              ShntiSgr_D           NaN
3                NaN        Doddablpur              ChikaDPP_D           NaN
4                NaN        Chandigarh              Mehmdpur_H           NaN

   Source_City  Source_Place_Code
0      Unknown               None
1      Unknown               None
2      Unknown               None
3      Unknown               None
4      Unknown               None

[5 rows x 31 columns]
```

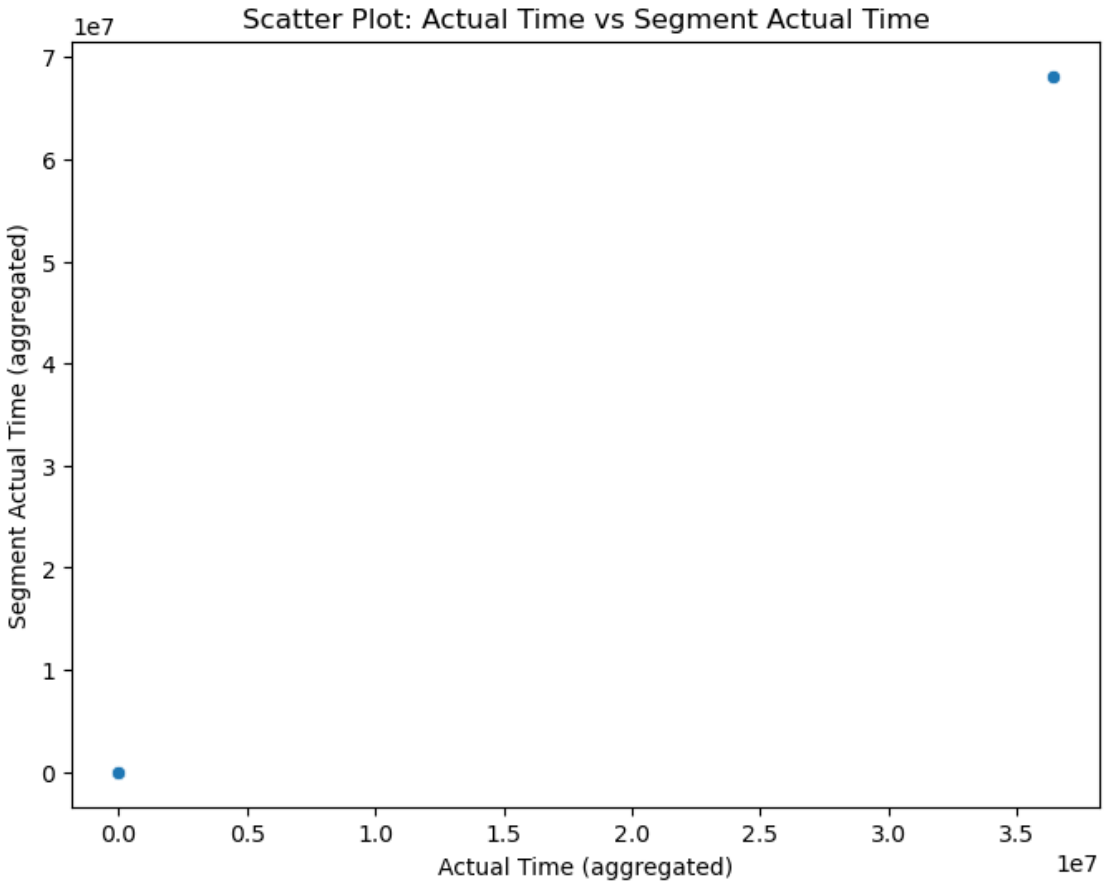
[76]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Assuming 'df_grouped' contains the aggregated data for actual_time and␣
 ↪osrm_time
# If you want the trip-level aggregation, group by 'trip_uuid' first
aggregated_df = df_grouped.groupby('trip_uuid').agg({
    'actual_time': 'sum',  # Summing actual time for each trip
    'osrm_time': 'sum'     # Summing OSRM time for each trip
}).reset_index()

# 1. Visual Analysis: Scatter plot and Correlation Analysis
plt.figure(figsize=(8, 6))
sns.scatterplot(x=aggregated_df['actual_time'], y=aggregated_df['osrm_time'])
plt.title("Scatter Plot: Actual Time vs OSRM Time")
plt.xlabel("Actual Time (aggregated)")
plt.ylabel("OSRM Time (aggregated)")
plt.show()

# Calculate correlation
correlation = aggregated_df['actual_time'].corr(aggregated_df['osrm_time'])
print(f"Correlation between Actual Time and OSRM Time: {correlation:.4f}")

# 2. Hypothesis Testing: Paired t-test
# Null hypothesis (H0): The means of actual_time and osrm_time are equal
# Alternative hypothesis (H1): The means of actual_time and osrm_time are not␣
 ↪equal

t_stat, p_value = stats.ttest_rel(aggregated_df['actual_time'],␣
 ↪aggregated_df['osrm_time'])

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Decision based on p-value
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference␣
 ↪between Actual Time and OSRM Time.")
else:
    print("Fail to reject the null hypothesis: No significant difference␣
 ↪between Actual Time and OSRM Time.")
```




```
Correlation between Actual Time and OSRM Time: 1.0000
T-statistic: 1.1023
P-value: 0.2704
Fail to reject the null hypothesis: No significant difference between Actual
Time and OSRM Time.
```

[78]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Assuming 'df_grouped' contains the data with 'actual_time' and␣
 ↪'segment_actual_time'
# If you want the trip-level aggregation, group by 'trip_uuid' first
aggregated_df = df_grouped.groupby('trip_uuid').agg({
    'actual_time': 'sum',  # Summing actual time for each trip
    'segment_actual_time': 'sum'  # Summing segment actual time for each trip
}).reset_index()

# 1. Visual Analysis: Scatter plot and Correlation Analysis
```

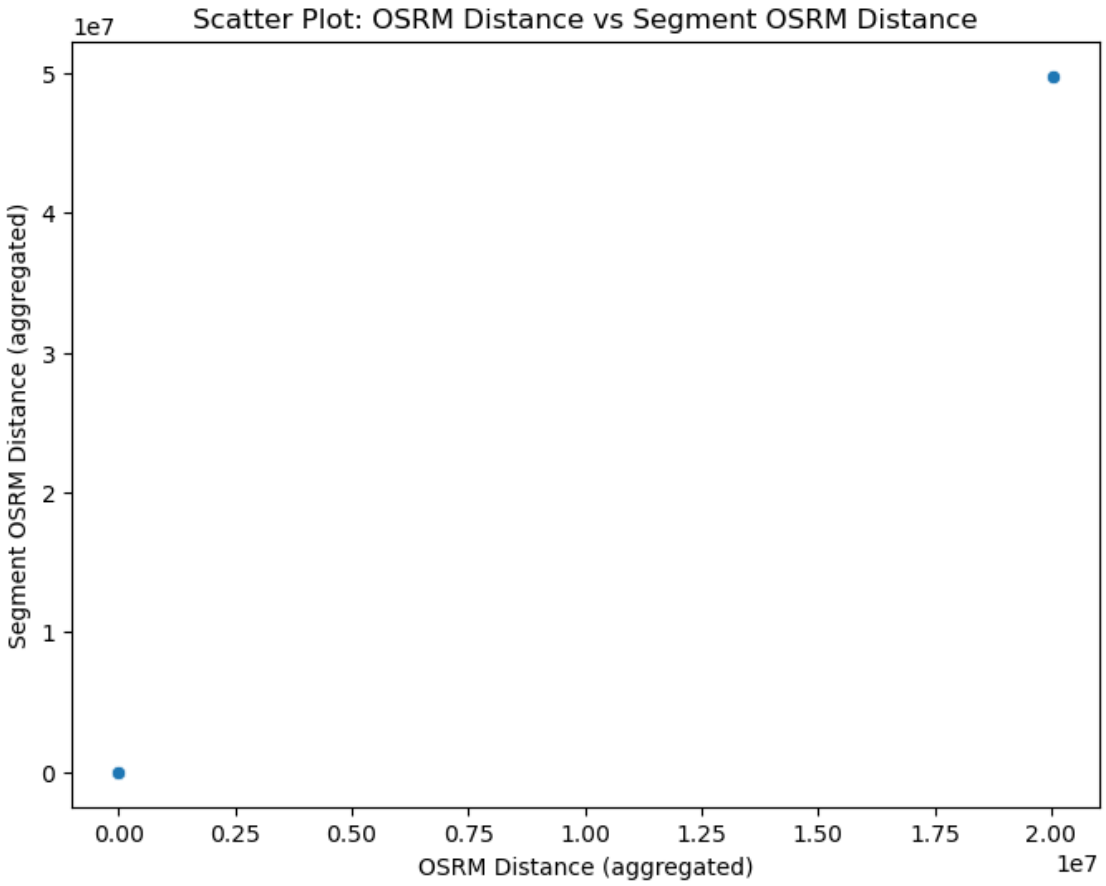
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=aggregated_df['actual_time'],␣
 ↪y=aggregated_df['segment_actual_time'])
plt.title("Scatter Plot: Actual Time vs Segment Actual Time")
plt.xlabel("Actual Time (aggregated)")
plt.ylabel("Segment Actual Time (aggregated)")
plt.show()

# Calculate correlation
correlation = aggregated_df['actual_time'].
 ↪corr(aggregated_df['segment_actual_time'])
print(f"Correlation between Actual Time and Segment Actual Time: {correlation:.
 ↪4f}")

# 2. Hypothesis Testing: Paired t-test
# Null hypothesis (H0): The means of actual_time and segment_actual_time are␣
 ↪equal
# Alternative hypothesis (H1): The means of actual_time and segment_actual_time␣
 ↪are not equal

t_stat, p_value = stats.ttest_rel(aggregated_df['actual_time'],␣
 ↪aggregated_df['segment_actual_time'])

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Decision based on p-value
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference␣
 ↪between Actual Time and Segment Actual Time.")
else:
    print("Fail to reject the null hypothesis: No significant difference␣
 ↪between Actual Time and Segment Actual Time.")
```




```
Correlation between Actual Time and Segment Actual Time: 1.0000
T-statistic: -1.0802
P-value: 0.2801
Fail to reject the null hypothesis: No significant difference between Actual
Time and Segment Actual Time.
```

[81]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Assuming 'df_grouped' contains the data with 'osrm_distance' and␣
 ↪'segment_osrm_distance'
# If you want the trip-level aggregation, group by 'trip_uuid' first
aggregated_df = df_grouped.groupby('trip_uuid').agg({
    'osrm_distance': 'sum',  # Summing OSRM distance for each trip
    'segment_osrm_distance': 'sum'  # Summing segment OSRM distance for each␣
 ↪trip
}).reset_index()
```

```
# 1. Visual Analysis: Scatter plot and Correlation Analysis
plt.figure(figsize=(8, 6))
sns.scatterplot(x=aggregated_df['osrm_distance'],␣
 ↪y=aggregated_df['segment_osrm_distance'])
plt.title("Scatter Plot: OSRM Distance vs Segment OSRM Distance")
plt.xlabel("OSRM Distance (aggregated)")
plt.ylabel("Segment OSRM Distance (aggregated)")
plt.show()

# Calculate correlation
correlation = aggregated_df['osrm_distance'].
 ↪corr(aggregated_df['segment_osrm_distance'])
print(f"Correlation between OSRM Distance and Segment OSRM Distance:␣
 ↪{correlation:.4f}")

# 2. Hypothesis Testing: Paired t-test
# Null hypothesis (H0): The means of osrm_distance and segment_osrm_distance␣
 ↪are equal
# Alternative hypothesis (H1): The means of osrm_distance and␣
 ↪segment_osrm_distance are not equal

t_stat, p_value = stats.ttest_rel(aggregated_df['osrm_distance'],␣
 ↪aggregated_df['segment_osrm_distance'])

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Decision based on p-value
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference␣
 ↪between OSRM Distance and Segment OSRM Distance.")
else:
    print("Fail to reject the null hypothesis: No significant difference␣
 ↪between OSRM Distance and Segment OSRM Distance.")
```



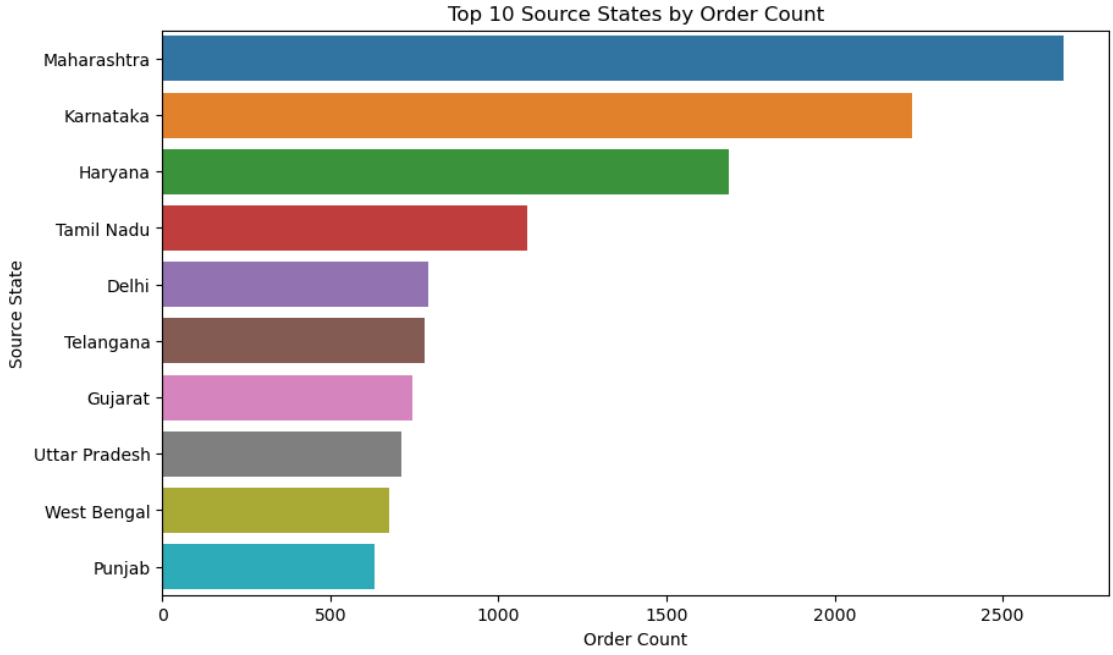

```
Correlation between OSRM Distance and Segment OSRM Distance: 1.0000
T-statistic: -0.9964
P-value: 0.3191
Fail to reject the null hypothesis: No significant difference between OSRM
Distance and Segment OSRM Distance.
```

[82]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Check where most orders are coming from (by Source State)
source_state_counts = df_grouped['Source_State'].value_counts().reset_index()
source_state_counts.columns = ['Source_State', 'Order_Count']

# Display top 10 source states by order count
print(source_state_counts.head(10))

# Visualize the distribution of orders from different states
plt.figure(figsize=(10, 6))
```

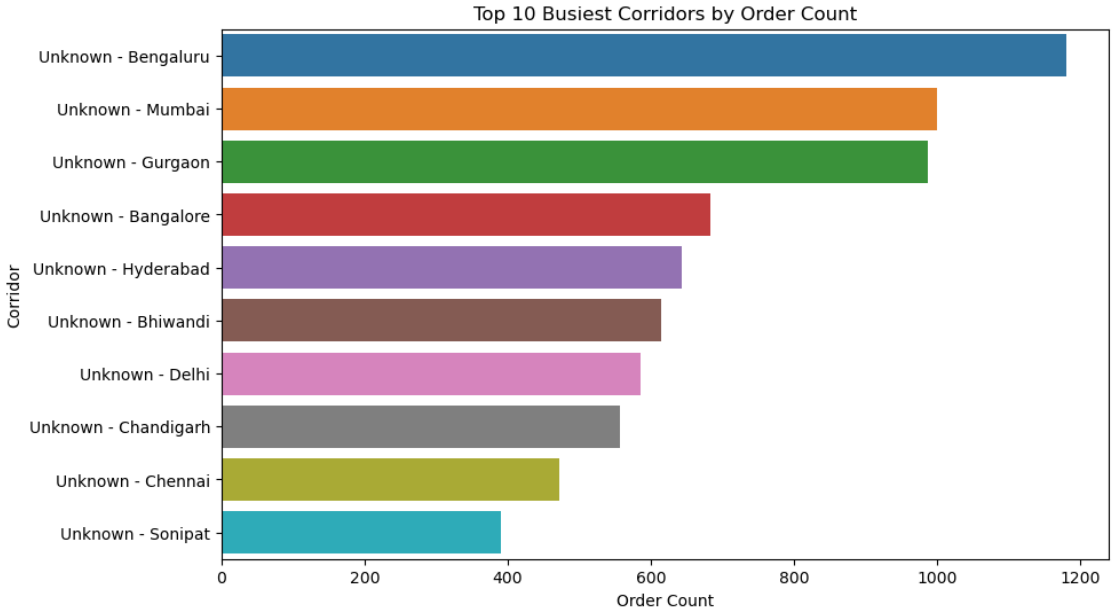
```
sns.barplot(x='Order_Count', y='Source_State', data=source_state_counts.
 ↪head(10))
plt.title("Top 10 Source States by Order Count")
plt.xlabel("Order Count")
plt.ylabel("Source State")
plt.show()

# 2. Check busiest corridors (most common source-destination pairs)
df_grouped['Corridor'] = df_grouped['Source_City'] + " - " +␣
 ↪df_grouped['Destination_City']
corridor_counts = df_grouped['Corridor'].value_counts().reset_index()
corridor_counts.columns = ['Corridor', 'Order_Count']

# Display top 10 busiest corridors
print(corridor_counts.head(10))

# Visualize busiest corridors
plt.figure(figsize=(10, 6))
sns.barplot(x='Order_Count', y='Corridor', data=corridor_counts.head(10))
plt.title("Top 10 Busiest Corridors by Order Count")
plt.xlabel("Order Count")
plt.ylabel("Corridor")
plt.show()

# 3. Average distance and time taken for each corridor
corridor_stats = df_grouped.groupby('Corridor').agg({
    'actual_distance_to_destination': 'mean',  # Average distance
    'actual_time': 'mean'  # Average time
}).reset_index()

# Display top 10 corridors with the highest average distance
print(corridor_stats.nlargest(10, 'actual_distance_to_destination'))

# Visualize average distance and time for each corridor
plt.figure(figsize=(10, 6))
sns.barplot(x='actual_distance_to_destination', y='Corridor',␣
 ↪data=corridor_stats.head(10))
plt.title("Top 10 Corridors by Average Distance")
plt.xlabel("Average Distance")
plt.ylabel("Corridor")
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(x='actual_time', y='Corridor', data=corridor_stats.head(10))
plt.title("Top 10 Corridors by Average Time Taken")
plt.xlabel("Average Time")
plt.ylabel("Corridor")
```

```
plt.show()
```

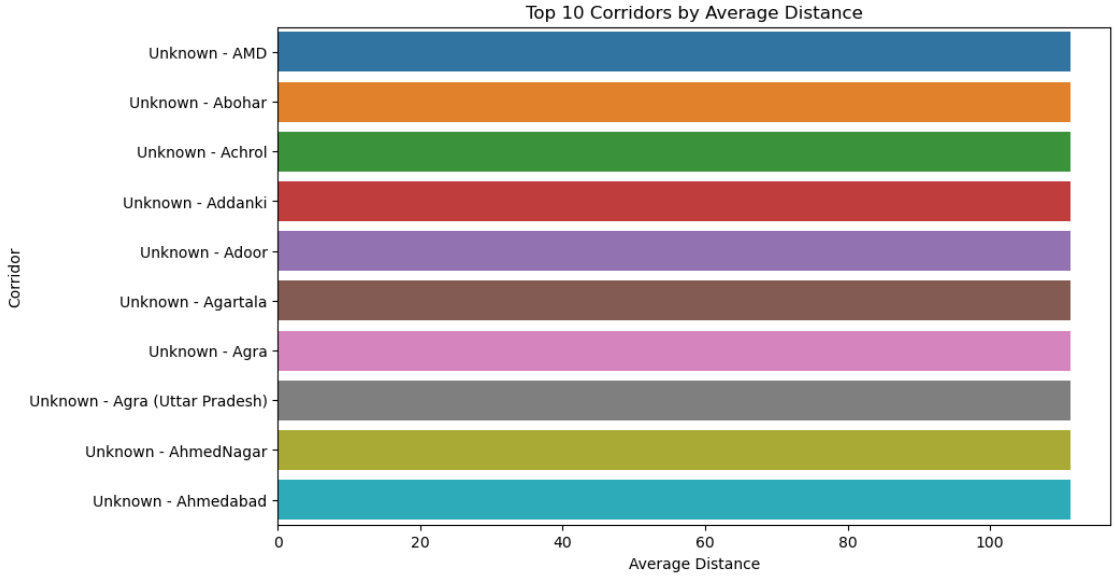
	Source_State	Order_Count
0	Maharashtra	2682
1	Karnataka	2229
2	Haryana	1684
3	Tamil Nadu	1085
4	Delhi	793
5	Telangana	780
6	Gujarat	746
7	Uttar Pradesh	713
8	West Bengal	677
9	Punjab	630

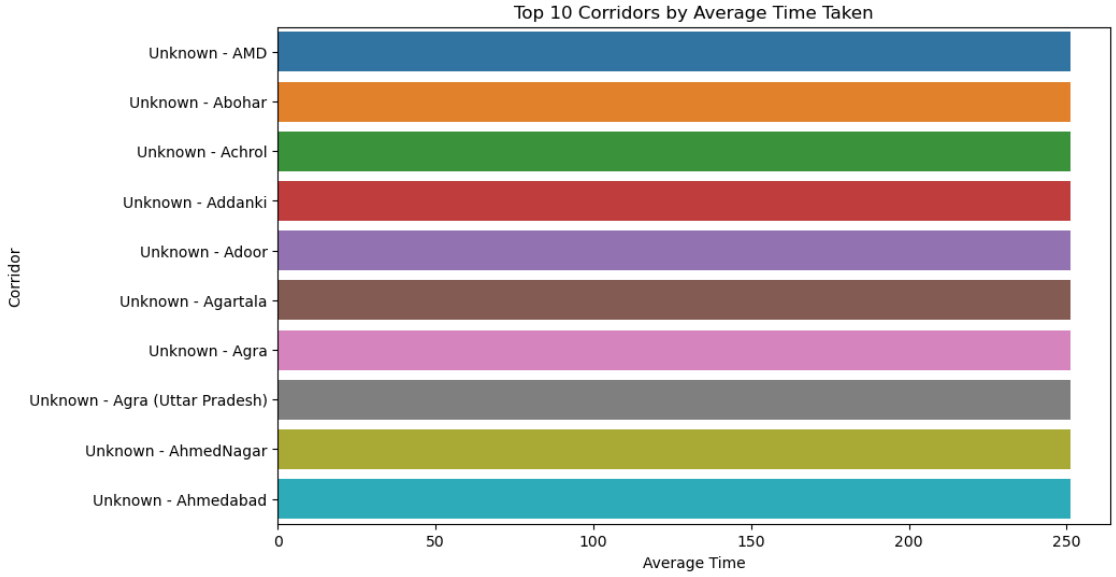


	Corridor	Order_Count
0	Unknown - Bengaluru	1180
1	Unknown - Mumbai	1000
2	Unknown - Gurgaon	986
3	Unknown - Bangalore	683
4	Unknown - Hyderabad	643
5	Unknown - Bhiwandi	614
6	Unknown - Delhi	585
7	Unknown - Chandigarh	556
8	Unknown - Chennai	472
9	Unknown - Sonipat	390



	Corridor	actual_distance_to_destination	actual_time
0	Unknown - AMD	111.336155	251.343389
68	Unknown - Assandh	111.336155	251.343389
72	Unknown - Attingal	111.336155	251.343389
128	Unknown - Barshi	111.336155	251.343389
139	Unknown - Bellary	111.336155	251.343389
141	Unknown - Bengaluru	111.336155	251.343389
143	Unknown - Benipur	111.336155	251.343389
180	Unknown - Bidar (Karnataka)	111.336155	251.343389
257	Unknown - Chittapur	111.336155	251.343389
278	Unknown - Dahod	111.336155	251.343389





[ ]:
```
"""
1. Target High-Volume Source States for Operational Focus
Action: Focus on optimizing operations in high-order-count states like␣
 ↪Maharashtra, Karnataka, Haryana, and Tamil Nadu, as these generate the␣
 ↪highest number of orders.
Initiatives:
Allocate more resources (vehicles, drivers, etc.) to these regions.
Optimize routes and delivery schedules to handle higher demand efficiently.
```

Improve customer service and ensure shorter delivery times in these regions.
Benefit: By enhancing operations in high-order regions, businesses can improve␣
↪operational efficiency, meet customer expectations better, and improve␣
↪profitability in these key markets.
2. Optimize Busiest Corridors to Reduce Congestion
Action: Prioritize improvements to the busiest corridors, such as "Unknown -␣
↪Bengaluru," "Unknown - Mumbai," and "Unknown - Gurgaon." These corridors␣
↪experience the highest order volumes, which may result in delays and service␣
↪issues.
Initiatives:
Assess and optimize traffic flow, delivery windows, and routes on these␣
↪corridors.
Consider partnerships with local logistics providers or traffic management␣
↪solutions to reduce congestion.
Use predictive analytics to foresee traffic or delivery delays and adjust␣
↪routes proactively.
Benefit: Optimizing the busiest corridors will improve delivery speed, reduce␣
↪costs associated with delays, and enhance overall customer satisfaction.
3. Leverage Data for Targeted Marketing Campaigns
Action: Based on the high-order regions (such as Maharashtra, Karnataka, and␣
↪Haryana), initiate targeted marketing campaigns to increase order volume.
Initiatives:
Launch location-specific promotions and discounts.
Partner with local businesses in these high-demand regions to create tailored␣
↪offers.
Benefit: Targeted marketing campaigns can increase demand, drive revenue, and␣
↪help capture a larger market share in high-volume states.
4. Optimize Resource Allocation for High-Demand Corridors
Action: Optimize resource allocation for corridors with high order counts. For␣
↪example, corridors like "Unknown - Bengaluru" and "Unknown - Mumbai" require␣
↪careful planning of vehicle allocation, delivery capacity, and personnel␣
↪resources.
Initiatives:
Forecast demand spikes and adjust staffing levels or fleet availability␣
↪accordingly.
Use historical data to predict busy periods and adjust delivery capacity␣
↪proactively.
Benefit: This optimization ensures that the business is equipped to handle high␣
↪demand without sacrificing service quality or delivery speed.
5. Refine Pricing Strategy for Corridors with Low or No Profitability
Action: Based on insights from average distance and time, identify corridors␣
↪with low profitability (e.g., corridors with low distances but high␣
↪operational costs) and evaluate the feasibility of improving the pricing␣
↪strategy.
Initiatives:

```
Increase delivery charges or offer tiered pricing based on the distance and␣
 ↪complexity of deliveries.
Analyze profitability by corridor to identify potential areas where price␣
 ↪adjustments could optimize margins.
Benefit: Refined pricing strategies will help improve profitability by ensuring␣
 ↪that high-cost corridors generate enough revenue to justify their␣
 ↪operational expenses.
"""
```