Business Case: Delhivery - Feature Engineering

🔮 Introduction

🚚 About Delhivery

Delhivery is a leading Indian logistics service provider that has revolutionized the country's delivery and supply chain industry. Founded in 2010, the company has rapidly expanded its operations to reach a vast network across India, offering a comprehensive suite of logistics solutions.

Delhivery's core services include:

- **Express Delivery:** Providing fast and reliable delivery services for various packages and products.
- **Reverse Logistics:** Handling returns and exchanges for e-commerce businesses.
- **Supply Chain Solutions:** Offering end-to-end supply chain management services, including warehousing, transportation, and distribution.
- **Cross-Border Logistics:** Facilitating international shipping and logistics operations.

With its extensive network, advanced technology, and commitment to customer satisfaction, Delhivery has become a trusted partner for businesses of all sizes, from e-commerce giants to small and medium enterprises.

🎯 Objective

The company wants to understand and process the data coming out of data engineering pipelines:

- Clean, sanitize and manipulate data to get useful features out of raw fields
- Make sense out of the raw data and help the data science team to build forecasting models on it

📁 Dataset

The provided dataset (`delhivery_data.csv`) holds valuable information about package deliveries, including trip details, time taken, distances, and route information.

💾 Features of the dataset.

Feature	Description
data	Tells whether the data is testing or training data
trip_creation_time	Timestamp of trip creation
route_schedule_uuid	Unique Id for a particular route schedule
route_type	Transportation type
FTL	Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
Carting	Handling system consisting of small vehicles (carts)
trip_uuid	Unique ID given to a particular trip (A trip may include different source and destination centers)
source_center	Source ID of trip origin
source_name	Source Name of trip origin
destination_cente	Destination ID
destination_name	Destination Name
od_start_time	Trip start time
od_end_time	Trip end time
start_scan_to_end_scan	Time taken to deliver from source to destination
is_cutoff	Unknown field
cutoff_factor	Unknown field
cutoff_timestamp	Unknown field
actual_distance_to_destination	Distance in Kms between source and destination warehouse

Feature	Description
actual_time	Actual time taken to complete the delivery (Cumulative)
osrm_time	An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through m
osrm_distance	An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor
factor	Unknown field
segment_actual_time	This is a segment time. Time taken by the subset of the package delivery
segment_osrm_time	This is the OSRM segment time. Time taken by the subset of the package delivery
segment_osrm_distance	This is the OSRM distance. Distance covered by subset of the package delivery
segment_factor	Unknown field

📚 Import Necessary Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from scipy.stats import levene
from scipy.stats import probplot
from scipy.stats import wilcoxon
from scipy.stats import mannwhitneyu
from scipy.stats import boxcox, shapiro, anderson
import statsmodels.api as sm
import missingno as msno
import warnings
import re
warnings.filterwarnings('ignore')
```

📁 Loading the Dataset:

```
# Download the data
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181 -O delhivery_dat
```

```
delhivery_data.csv  100%[===================>]  53.04M   116MB/s    in 0.5s
```

🧭 Exploring the Dataset:

```
# Read the CSV file into a Pandas DataFrame
df = pd.read_csv("delhivery_data.csv")

# Display the first few rows of the DataFrame
df.head()
```

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

5 rows × 24 columns

```
# The number of rows and columns given in the dataset
df.shape
```

```
(144867, 24)
```

```
# The characteristics of all columns.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool   
 13  cutoff_factor                   144867 non-null  int64  
 14  cutoff_timestamp                144867 non-null  object 
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

```
# Dropping unknown fields

df = df.drop(columns = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor'])
```

```
df.shape
```

```
(144867, 19)
```

1. 🕵️‍♀️ Basic data cleaning and exploration:

1.1 Handle missing values in the data.

```
# Check for the missing values and find the number of missing values in each column
df.isnull().sum()
```

	0
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
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0

dtype: int64

```
missing_source_name = df.loc[df['source_name'].isnull(), 'source_center'].unique()
missing_source_name
```

```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

```
missing_destination_name = df.loc[df['destination_name'].isnull(), 'destination_center'].unique()
missing_destination_name
```

```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

```
for i in missing_source_name:
    unique_source_name = df.loc[df['source_center'] == i, 'source_name'].unique()
    if pd.isna(unique_source_name):
        print("Source Center :", i, "-" * 7, "Source Name :", 'Not Found')
    else :
        print("Source Center :", i, "-" * 7, "Source Name :", unique_source_name)
```

```
Source Center : IND342902A1B ------- Source Name : Not Found
Source Center : IND577116AAA ------- Source Name : Not Found
Source Center : IND282002AAD ------- Source Name : Not Found
Source Center : IND465333A1B ------- Source Name : Not Found
Source Center : IND841301AAC ------- Source Name : Not Found
Source Center : IND509103AAC ------- Source Name : Not Found
Source Center : IND126116AAA ------- Source Name : Not Found
Source Center : IND331022A1B ------- Source Name : Not Found
Source Center : IND505326AAB ------- Source Name : Not Found
Source Center : IND852118A1B ------- Source Name : Not Found
```

```
for i in missing_destination_name:
    unique_destination_name = df.loc[df['destination_center'] == i, 'destination_name'].unique()
    if (pd.isna(unique_source_name)) or (unique_source_name.size == 0):
        print("Destination Center :", i, "-" * 7, "Destination Name :", 'Not Found')
    else :
        print("Destination Center :", i, "-" * 7, "Destination Name :", unique_destination_name)
```

```
Destination Center : IND342902A1B ------- Destination Name : Not Found
Destination Center : IND577116AAA ------- Destination Name : Not Found
```

```
Destination Center : IND282002AAD ------- Destination Name : Not Found
Destination Center : IND465333A1B ------- Destination Name : Not Found
Destination Center : IND841301AAC ------- Destination Name : Not Found
Destination Center : IND505326AAB ------- Destination Name : Not Found
Destination Center : IND852118A1B ------- Destination Name : Not Found
Destination Center : IND126116AAA ------- Destination Name : Not Found
Destination Center : IND509103AAC ------- Destination Name : Not Found
Destination Center : IND221005A1A ------- Destination Name : Not Found
Destination Center : IND250002AAC ------- Destination Name : Not Found
Destination Center : IND331001A1C ------- Destination Name : Not Found
Destination Center : IND122015AAC ------- Destination Name : Not Found
```

```
# The IDs for which the source name is missing, are all those IDs for destination also missing ?
np.all(df.loc[df['source_name'].isnull(), 'source_center'].isin(missing_destination_name))
```

```
False
```

```
df.source_name.value_counts()
```

source_name	count
Gurgaon_Bilaspur_HB (Haryana)	23347
Bangalore_Nelmngla_H (Karnataka)	9975
Bhiwandi_Mankoli_HB (Maharashtra)	9088
Pune_Tathawde_H (Maharashtra)	4061
Hyderabad_Shamshbd_H (Telangana)	3340
...	...
Shahjhnpur_NavdaCln_D (Uttar Pradesh)	1
Soro_UttarDPP_D (Orissa)	1
Kayamkulam_Bhrnikvu_D (Kerala)	1
Krishnanagar_AnadiDPP_D (West Bengal)	1
Faridabad_Old (Haryana)	1

1498 rows × 1 columns

dtype: int64

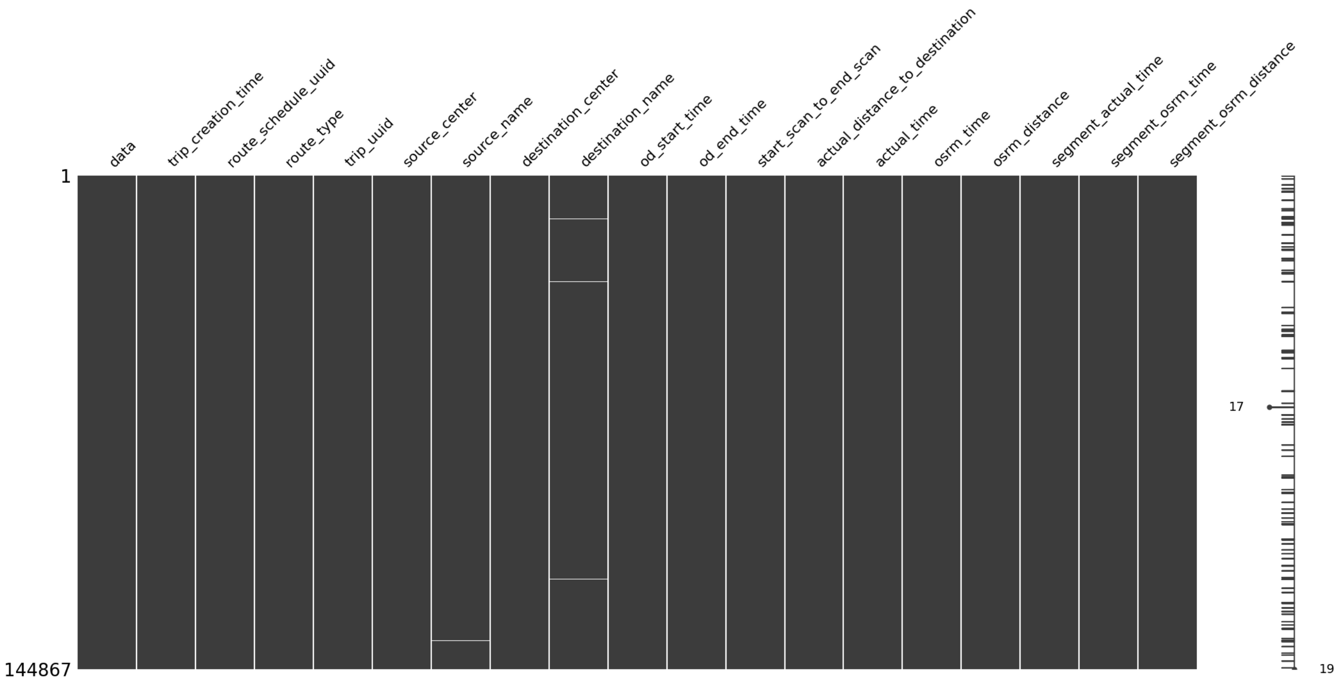
```
df.destination_name.value_counts()
```

destination_name	count
Gurgaon_Bilaspur_HB (Haryana)	15192
Bangalore_Nelmngla_H (Karnataka)	11019
Bhiwandi_Mankoli_HB (Maharashtra)	5492
Hyderabad_Shamshbd_H (Telangana)	5142
Kolkata_Dankuni_HB (West Bengal)	4892
...	...
Hyd_Trimulgherry_Dc (Telangana)	1
Vijayawada (Andhra Pradesh)	1
Baghpat_Barout_D (Uttar Pradesh)	1
Mumbai_Sanpada_CP (Maharashtra)	1
Basta_Central_DPP_1 (Orissa)	1

1468 rows × 1 columns

dtype: int64

```
#graphical represnation of null values
msno.matrix(df)
plt.show()
```




```
# Calculate the percentage of missing values for 'source_name'
source_name_missing_percentage = (df['source_name'].isnull().sum() / len(df)) * 100

# Calculate the percentage of missing values for 'destination_name'
destination_name_missing_percentage = (df['destination_name'].isnull().sum() / len(df)) * 100

print(f"Percentage of missing values for 'source_name': {source_name_missing_percentage:.2f}%")
print(f"Percentage of missing values for 'destination_name': {destination_name_missing_percentage:.2f}%")
```

```
Percentage of missing values for 'source_name': 0.20%
Percentage of missing values for 'destination_name': 0.18%
```

🔍 Insights

- After analyzing the missing values in the `source_name` and `destination_name` columns, we determined that **imputing default values would not significantly improve the data quality.** Given the relatively low percentage of missing data (0.20% and 0.18%, respectively), **removing rows containing null values is a viable option.**
- This approach ensures that our analysis and modeling are based on complete and accurate data, **reducing the risk of introducing biases or inaccuracies due to missing information.** However, it's important to consider the specific context and goals of the analysis to determine if alternative strategies, such as imputation or further investigation, might be more appropriate.

```
# Drop rows with missing values in 'source_name' and 'destination_name'
df = df.dropna(subset=['source_name', 'destination_name'])
```

```
df.isna().sum().any()
```

```
False
```

```
df.shape
```

```
(144316, 19)
```

1.2 Converting time columns into pandas datetime.

```
df.dtypes
```

	0
data	object
trip_creation_time	object
route_schedule_uuid	object
route_type	object
trip_uuid	object
source_center	object
source_name	object
destination_center	object
destination_name	object
od_start_time	object
od_end_time	object
start_scan_to_end_scan	float64
actual_distance_to_destination	float64
actual_time	float64
osrm_time	float64
osrm_distance	float64
segment_actual_time	float64
segment_osrm_time	float64
segment_osrm_distance	float64

dtype: object

```
#checking the unique values for columns
df.nunique()
```

	0
data	2
trip_creation_time	14787
route_schedule_uuid	1497
route_type	2
trip_uuid	14787
source_center	1496
source_name	1496
destination_center	1466
destination_name	1466
od_start_time	26223
od_end_time	26223
start_scan_to_end_scan	1914
actual_distance_to_destination	143965
actual_time	3182
osrm_time	1531
osrm_distance	137544
segment_actual_time	746
segment_osrm_time	214
segment_osrm_distance	113497

dtype: int64

```
# prompt: #checking the unique values for columns

for column in df.columns:
  print(f"Unique values for {column}: \n{df[column].unique()}")
  print('-'*100)
```

```
 1.140e+02 1.840e+02 2.270e+02 1.740e+02 1.320e+02 9.900e+01 9.600e+01
 1.310e+02 1.110e+02 1.040e+02 1.750e+02 2.300e+02 9.500e+01 1.250e+02
 2.950e+02 1.560e+02 1.160e+02 1.460e+02 1.410e+02 1.030e+02 1.170e+02
 2.310e+02 2.540e+02 2.200e+02 2.330e+02 1.810e+02 1.210e+02 1.270e+02
 3.700e+02 3.750e+02 1.500e+02 1.070e+02 1.610e+02 2.320e+02 1.090e+02
 1.200e+02 1.100e+02 9.970e+02 1.790e+02 1.130e+02 1.660e+02 9.960e+02
 1.240e+02 2.150e+02 1.570e+02 3.620e+02 1.430e+02 1.150e+02 1.280e+02
 1.700e+02 1.440e+02 2.350e+02 1.510e+02 3.560e+02 1.180e+02 1.390e+02
 1.710e+02 1.290e+02 1.190e+02 1.690e+02 1.630e+02 2.040e+02 1.480e+02
 1.830e+02 4.810e+02 3.410e+02 3.280e+02 2.130e+02 1.890e+02 1.910e+02
 1.400e+02 1.470e+02 2.080e+02 2.860e+02 2.160e+02 1.720e+02 1.380e+02
 1.670e+02 2.940e+02 1.230e+02 1.260e+02 2.110e+02 1.611e+03 2.190e+02
 2.490e+02 1.850e+02 1.580e+02 3.240e+02 1.770e+02 4.530e+02 1.520e+02
 1.760e+02 7.370e+02 1.730e+02 1.032e+03]
----------------------------------------------------------------------------------------------------
Unique values for segment_osrm_distance:
[11.9653  9.759  10.8152 ... 20.7053 18.8885  8.8088]
----------------------------------------------------------------------------------------------------
```

🔍 Insights

1. Time Columns **(trip_creation_time, od_start_time, od_end_time, cutoff_timestamp)**:
   - These columns should be converted to pandas datetime objects to facilitate time-based analysis and calculations.
2. Categorical Columns **(data, route_type):**
   - These columns should be treated as categorical variables as they only have 2 values.
3. Float64 Columns **(actual_distance_to_destination, start_scan_to_end_scan, actual_time, osrm_time, osrm_distance, segment_actual_time, segment_osrm_time, segment_osrm_distance)**:
   - These columns store numerical data representing distance and time.
   - We can consider changing the datatype to float32 to reduce memory usage, as the precision of float64 might be excessive for our analysis.

```
# datatype changes for time columns into pandas datetime.
for column in ['trip_creation_time', 'od_start_time', 'od_end_time']:
  df[column] = pd.to_datetime(df[column])
```

```
# Convert float64 to float32

for column in ['actual_distance_to_destination', 'start_scan_to_end_scan', 'actual_time', 'osrm_time', 'osrm_distance', 'segment_a
  df[column] = pd.to_numeric(df[column], downcast='float')
```

```
# Convert categorical columns to category dtype
for column in ['data', 'route_type']:
  df[column] = df[column].astype('category')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  category
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  category
 4   trip_uuid                       144316 non-null  object
 5   source_center                   144316 non-null  object
 6   source_name                     144316 non-null  object
 7   destination_center              144316 non-null  object
 8   destination_name                144316 non-null  object
 9   od_start_time                   144316 non-null  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144316 non-null  float32
 12  actual_distance_to_destination  144316 non-null  float32
 13  actual_time                     144316 non-null  float32
 14  osrm_time                       144316 non-null  float32
 15  osrm_distance                   144316 non-null  float32
 16  segment_actual_time             144316 non-null  float32
 17  segment_osrm_time               144316 non-null  float32
 18  segment_osrm_distance           144316 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(8), object(6)
memory usage: 15.7+ MB
```

🔍 Insights

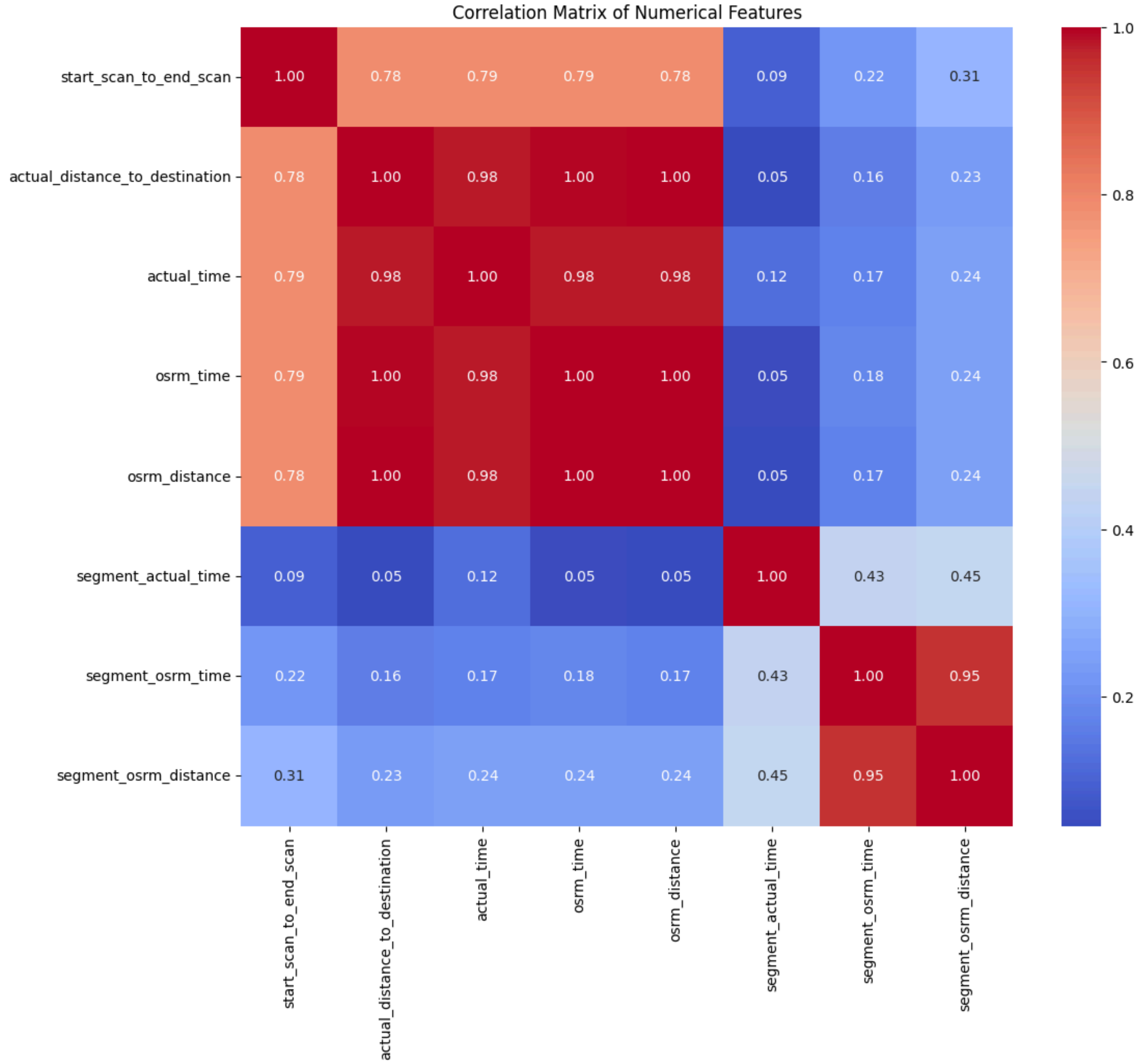
- The dataset's memory usage has been significantly optimized, decreasing from over 25.6 MB to just 15.7 MB. This represents an impressive reduction of approximately 38.67%.

1.3 Analyze structure & characteristics of the dataset.

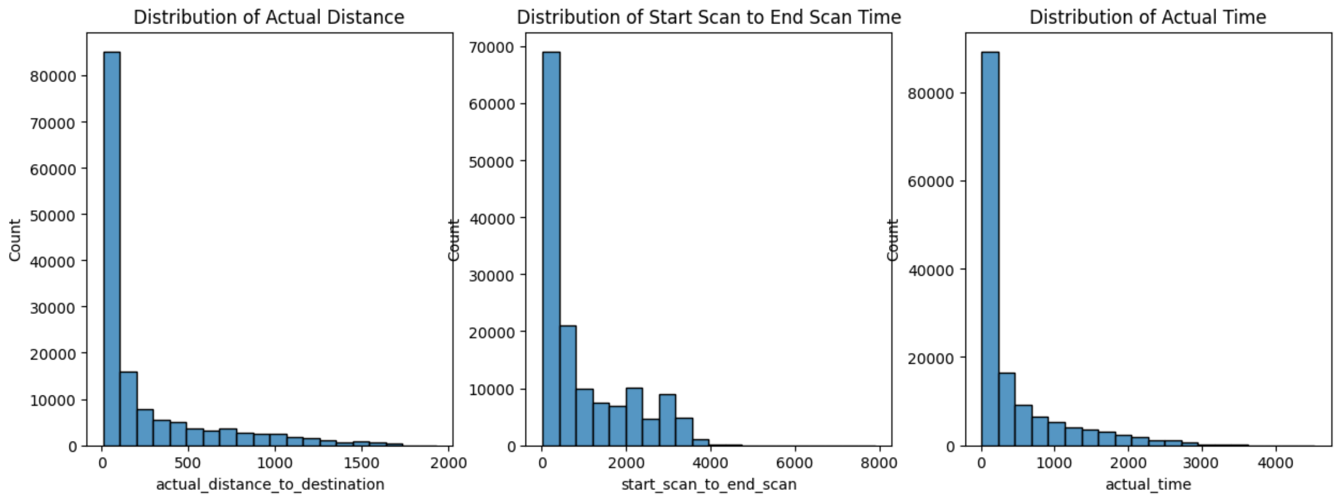
```
# Descriptive statistics for numerical features
numerical_features = df.select_dtypes(include=np.number)
numerical_features.describe()
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time
count	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.00000C
mean	963.697571	234.708496	417.996216	214.437042	285.549805	36.175381
std	1038.097290	345.474426	598.951843	308.438782	421.714020	53.519287
min	20.000000	9.000046	9.000000	6.000000	9.008200	-244.00000C
25%	161.000000	23.352027	51.000000	27.000000	29.896250	20.00000C
50%	451.000000	66.135319	132.000000	64.000000	78.624401	28.00000C
75%	1645.000000	286.919304	516.000000	259.000000	346.305397	40.00000C
max	7898.000000	1927.447754	4532.000000	1686.000000	2326.199219	3051.00000C

```
# Correlation matrix for numerical features
correlation_matrix = numerical_features.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```




```
# Distribution of key features
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.histplot(df['actual_distance_to_destination'], bins=20)
plt.title('Distribution of Actual Distance')

plt.subplot(1, 3, 2)
sns.histplot(df['start_scan_to_end_scan'], bins=20)
plt.title('Distribution of Start Scan to End Scan Time')

plt.subplot(1, 3, 3)
sns.histplot(df['actual_time'], bins=20)
plt.title('Distribution of Actual Time')
plt.show()
```




```
# Unique values for categorical features
for column in df.select_dtypes(include=['category']):
    print(f"Unique values for {column}: {df[column].unique()}")
```

```
Unique values for data: ['training', 'test']
Categories (2, object): ['test', 'training']
Unique values for route_type: ['Carting', 'FTL']
Categories (2, object): ['Carting', 'FTL']
```

🔍 Insights

- **Correlation Analysis:**
  - The correlation matrix reveals strong positive correlations between `actual_time`, `osrm_time`, `segment_actual_time`, and `segment_osrm_time`. This indicates that these features are highly related, as expected, as the total delivery time is influenced by the time taken for individual segments.
  - Similarly, `actual_distance_to_destination` and `osrm_distance` show a strong positive correlation.
  - Understanding these correlations is important for feature selection and model building, as highly correlated features can lead to multicollinearity issues.
- **Distribution Analysis:**
  - The distributions of `actual_distance_to_destination`, `start_scan_to_end_scan`, and `actual_time` show that most deliveries are within a specific range of distances and times, which is important to understand the general operational pattern.
  - The presence of some outliers can be observed in the distributions, which can either be due to unusual delivery circumstances or potentially errors.
- **Categorical Features:**
  - The 'data' feature tells us whether it's test data or train data.
  - The 'route_type' has two values which are mainly 'FTL' and 'Carting'

These insights provide a preliminary understanding of the dataset, which will guide us in further analyses, such as outlier treatment, feature engineering, and model building.

🛠 2. Merging Rows and Aggregating by Segment

🔗 2.1 Grouping by segment

```
# Creating a unique identifier for each segment of a trip
df['segment_key'] = df['trip_uuid'] + '_' + df['source_center'].astype(str) + '_' + df['destination_center']
```

```
# Calculate cumulative sum of segment_actual_time, segment_osrm_distance, and segment_osrm_time within each segment
segment_columns = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time']
```

```
df[ [col + '_sum' for col in segment_columns]] = df.groupby('segment_key')[segment_columns].cumsum()
```

```
df[['segment_key', 'segment_actual_time', 'segment_actual_time_sum','segment_osrm_distance', 'segment_osrm_distance_sum','segment_
```

	segment_key	segment_actual_time	segment_actual_time_sum	segment_osrm_distance	segmen
0	trip-153741093647649320_IND388121AAA_IND388620AAB	14.0	14.0	11.9653	
1	trip-153741093647649320_IND388121AAA_IND388620AAB	10.0	24.0	9.7590	
2	trip-153741093647649320_IND388121AAA_IND388620AAB	16.0	40.0	10.8152	
3	trip-153741093647649320_IND388121AAA_IND388620AAB	21.0	61.0	13.0224	
4	trip-153741093647649320_IND388121AAA_IND388620AAB	6.0	67.0	3.9153	
5	trip-153741093647649320_IND388620AAB_IND388320AAA	15.0	15.0	12.1171	
6	trip-153741093647649320_IND388620AAB_IND388320AAA	28.0	43.0	9.1719	
7	trip-153741093647649320_IND388620AAB_IND388320AAA	21.0	64.0	14.5362	
8	trip-153741093647649320_IND388620AAB_IND388320AAA	10.0	74.0	11.3648	
9	trip-153741093647649320_IND388620AAB_IND388320AAA	26.0	100.0	6.0434	

2.2 Aggregating at segment level

```
# Creating a dictionary for aggregation at segment level

create_segment_dict = {
  'trip_uuid' : 'first',
  'data': 'first',
  'route_type': 'first',
  'trip_creation_time': 'first',
  'source_name': 'first',
  'destination_name': 'last',
  'od_start_time': 'first',
  'od_end_time': 'last',
  'start_scan_to_end_scan': 'first',
  'actual_distance_to_destination': 'last',
  'actual_time': 'last',
  'osrm_time': 'last',
  'osrm_distance': 'last',
  'segment_actual_time' : 'sum',
  'segment_osrm_time' : 'sum',
  'segment_osrm_distance' : 'sum',
  'segment_actual_time_sum': 'last',
  'segment_osrm_time_sum': 'last',
  'segment_osrm_distance_sum': 'last',
}


# Group by segment_key and aggregate
df_segment = df.groupby('segment_key').agg(create_segment_dict).reset_index()


# Sort by segment_key and od_end_time to ensure consistent ordering.
df_segment = df_segment.sort_values(['segment_key', 'od_end_time'])


df_segment.head()
```

	segment_key	trip_uuid	data	route_type	trip_creation_time	sourc
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Cent (Uttar F
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Bhopal_Trn (Madhya F
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_Chika (Kai
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Tumkur_Ve (Kai
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilas (H

Next steps: Generate code with df_segment | View recommended plots | New interactive sheet

🔍 Insights

- By grouping rows based on unique segment identifiers and aggregating relevant features, we create a more concise dataset suitable for analysis and modeling, representing complete trips with cumulative segment-level information.

⚙️ 3. Feature Engineering

⏳ 3.1Calculate Time Difference

```
df_fe = df_segment.copy()
```

```
# Calculate time difference between od_start_time and od_end_time
df_fe['od_time_diff_hour'] = (df_fe['od_end_time'] - df_fe['od_start_time']).dt.total_seconds() / 3600
```

🏙️ 3.2Extract Features from Destination Name

```
# using regex pattern to seperate the city,place,state
def extract_info(name):
    pattern = r'^(?P<city>[^\s_]+)_?(?P<place>[^\(\)]*)\s?\((?P<state>[A-Za-z\s&]+)\)$'
    match = re.match(pattern, name)
    if match:
        city = match.group('city').strip()
        place = match.group('place').strip() if match.group('place') else city
        state = match.group('state').strip()
        return city, place, state
    else:
        return None, None, None
```

```
df_fe[['destination_city', 'destination_place', 'destination_state']] = df_fe['destination_name'].apply(lambda x: pd.Series(extrac
```

🏙️ 3.3 Extract Features from Source Name

```
df_fe[['source_city', 'source_place', 'source_state']] = df_fe['source_name'].apply(lambda x: pd.Series(extract_info(x)))
```

```
df_fe.head()
```

	segment_key	trip_uuid	data	route_type	trip_creation_time	sourc
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Cent (Uttar F
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Bhopal_Trn (Madhya F
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_Chika (Kar
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Tumkur_Ve (Kar
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilas (H

5 rows × 27 columns

```
df_fe[(df_fe['source_place']=='') | (df_fe['destination_place']=='')]
```

	segment_key	trip_uuid	data	route_type	trip_creation_time	sou
7	trip-153671052974046625_IND583101AAA_IND583201AAA	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (K
9	trip-153671052974046625_IND583201AAA_IND583119AAA	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Hospet (K
19	trip-153671110078355292_IND121004AAB_IND121001AAA	trip-153671110078355292	training	Carting	2018-09-12 00:11:40.783923	FBD_Balabhg
33	trip-153671173668736946_IND110043AAA_IND110078AAA	trip-153671173668736946	training	Carting	2018-09-12 00:22:16.687619	Delhi_
80	trip-153671320807895983_IND121004AAB_IND121102AAA	trip-153671320807895983	training	Carting	2018-09-12 00:46:48.079257	FBD_Balabhg
...	...	...	...	...	...	
26118	trip-153860849934816308_IND110078AAA_IND110043AAA	trip-153860849934816308	test	Carting	2018-10-03 23:14:59.348414	Janakp
26153	trip-153860958923357924_IND842003AAB_IND482002AAA	trip-153860958923357924	test	Carting	2018-10-03 23:33:09.233829	Jabalpur_A (Madhya
26180	trip-153861007249500192_IND842001AAA_IND846004AAA	trip-153861007249500192	test	FTL	2018-10-03 23:41:12.495257	Muzaffrpur_
26181	trip-153861007249500192_IND846004AAA_IND847103AAA	trip-153861007249500192	test	FTL	2018-10-03 23:41:12.495257	Darbhan
26221	trip-153861118270144424_IND583201AAA_IND583119AAA	trip-153861118270144424	test	FTL	2018-10-03 23:59:42.701692	Hospet (K

782 rows × 27 columns

```
df_fe.loc[df_fe['source_place']=='','source_place']=df_fe['source_city']
df_fe.loc[df_fe['destination_place']=='','destination_place']=df_fe['destination_city']
```

```
df_fe.isna().sum().any()
```

```
False
```

```
df_fe['source_city'].replace('Bangalore', 'Bengaluru', inplace=True)
df_fe['destination_city'].replace('Bangalore', 'Bengaluru', inplace=True)
```

3.4 📅 Extract Features from Trip Creation Time

```
# Extract month, year, day, etc. from trip_creation_time
df_fe['trip_creation_month'] = df_fe['trip_creation_time'].dt.month
df_fe['trip_creation_year'] = df_fe['trip_creation_time'].dt.year
df_fe['trip_creation_day'] = df_fe['trip_creation_time'].dt.day
df_fe['trip_creation_hour'] = df_fe['trip_creation_time'].dt.hour
df_fe['trip_creation_weekday'] = df_fe['trip_creation_time'].dt.weekday
df_fe['trip_creation_week'] = df_fe['trip_creation_time'].dt.isocalendar().week
```

```
df_fe.head()
```

	segment_key	trip_uuid	data	route_type	trip_creation_time	sourc
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Cent (Uttar F
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Bhopal_Trn (Madhya F
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_Chika (Kar
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Tumkur_Ve (Kar
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilas (H

5 rows × 33 columns

🔍 Insights

- **Time Difference Calculation:** The `od_time_diff_hour` feature is created by finding the difference between the start and end times of a delivery segment and is converted to hours. This feature can be a critical predictor for model accuracy as it is a direct measure of delivery duration.
- **Location Feature Extraction:** We extract City, Place, and State information from source and destination names to create more granular location-based features. This enables us to analyze delivery performance based on different geographic areas.
- **Trip Creation Time Features:** We extracted features such as month, year, day, hour, weekday, and week number from the `trip_creation_time` column. These features can reveal patterns related to the creation of deliveries during specific time periods, days of the week, or months.

These new features can significantly enhance our ability to understand the underlying patterns in the dataset, allowing for more robust predictions.

🔎 4. In-depth analysis

```
df_ida = df_fe.copy()
```

⛓ 4.1 Grouping and Aggregating at Trip-level

```
# Create a dictionary for aggregation at trip level
create_trip_dict={
  'data' : 'first',
  'route_type' : 'first',
  'od_start_time':'first',
  'od_end_time':'last',
  'od_time_diff_hour' : 'sum',
  'trip_creation_time' : 'first',
  'trip_creation_month' : 'first',
  'trip_creation_year' : 'first',
  'trip_creation_day' : 'first',
  'trip_creation_hour' : 'first',
  'trip_creation_weekday' : 'first',
  'trip_creation_week' : 'first',
  'start_scan_to_end_scan' : 'sum',
  'actual_distance_to_destination' : 'sum',
  'actual_time' : 'sum',
  'osrm_time' : 'sum',
  'osrm_distance' : 'sum',
  'segment_actual_time': 'sum',
  'segment_osrm_time': 'sum',
  'segment_osrm_distance': 'sum',
  'segment_actual_time_sum': 'sum',
  'segment_osrm_time_sum': 'sum',
  'segment_osrm_distance_sum': 'sum',
  'source_name': 'first',
  'source_city':'first',
  'source_state':'first',
  'source_place':'first',
  'destination_name': 'first',
  'destination_city':'first',
```

```
  'destination_state':'first',
  'destination_place':'first',
  }

# Group by trip_uuid and aggregate
df_trip = df_ida.groupby('trip_uuid').agg(create_trip_dict).reset_index()

# Sort by trip_uuid and od_end_time to ensure consistent ordering.
df_trip = df_trip.sort_values(['trip_uuid', 'od_end_time'])
df_trip.head()
```

	trip_uuid	data	route_type	od_start_time	od_end_time	od_time_diff_hour	trip_creation_time	trip_creation
0	trip-153671041653548748	training	FTL	2018-09-12 16:39:46.858469	2018-09-12 16:39:46.858469	37.668497	2018-09-12 00:00:16.535741	
1	trip-153671042288605164	training	Carting	2018-09-12 02:03:09.655591	2018-09-12 02:03:09.655591	3.026865	2018-09-12 00:00:22.886430	
2	trip-153671043369099517	training	FTL	2018-09-14 03:40:17.106733	2018-09-14 03:40:17.106733	65.572709	2018-09-12 00:00:33.691250	
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	1.674916	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	2018-09-12 03:54:43.114421	11.972484	2018-09-12 00:02:09.740725	

5 rows × 32 columns

🚨 4.2 Outlier Detection & Treatment

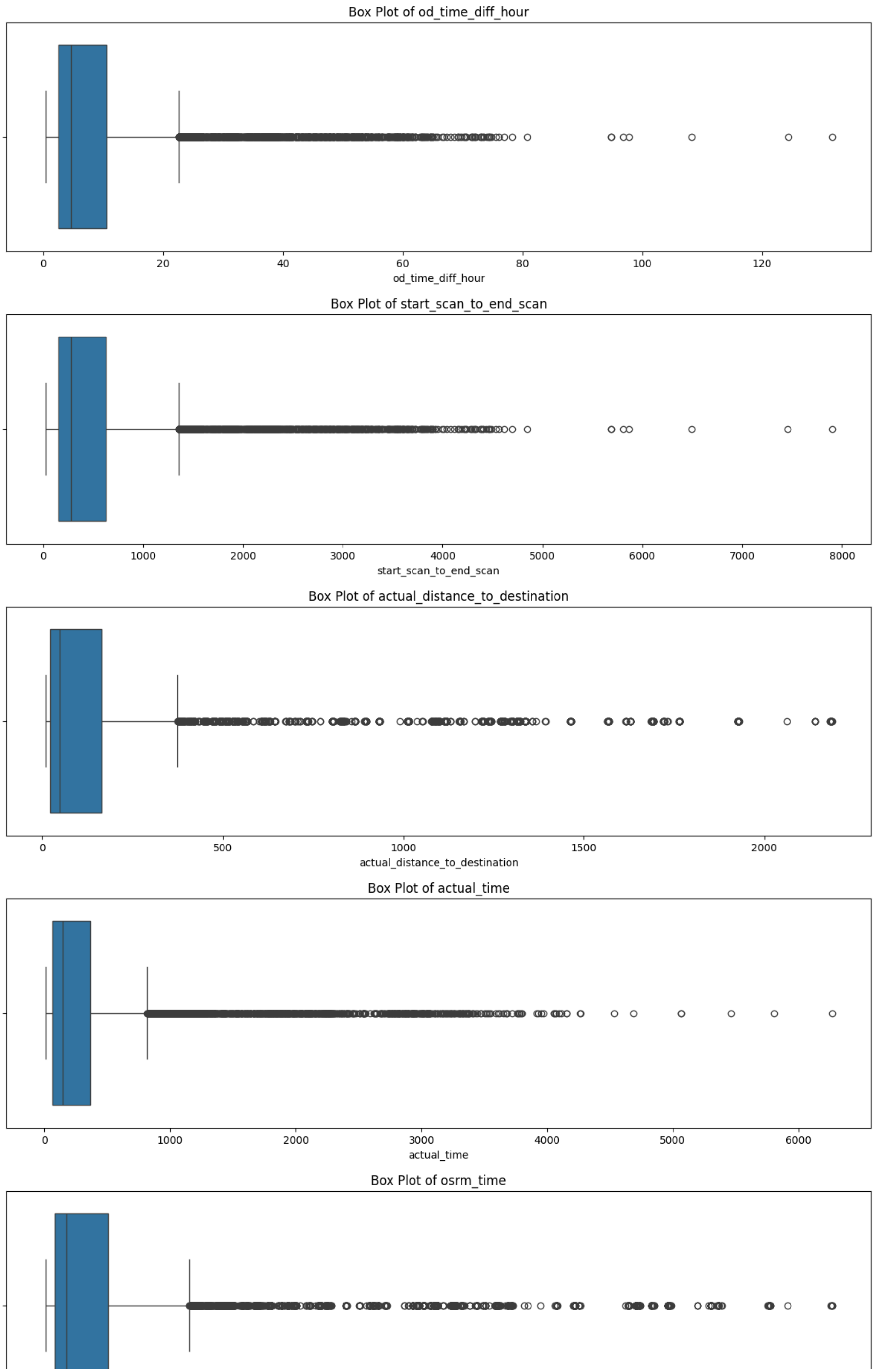
```
numerical_features =  df_trip.select_dtypes(include=[np.float32, np.float64])

# list of numerical columns
numerical_columns = numerical_features.columns.tolist()
numerical_features
```

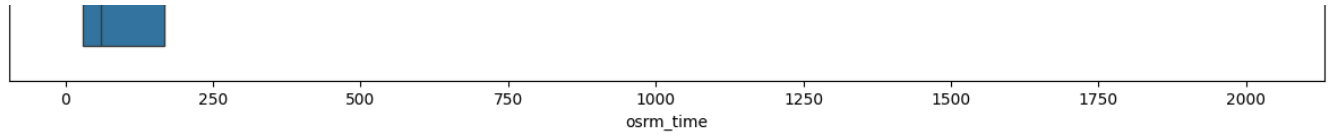
	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	37.668497	2259.0	824.732849	1562.0	717.0	991.352295	
1	3.026865	180.0	73.186905	143.0	68.0	85.111000	
2	65.572709	3933.0	1927.404297	3347.0	1740.0	2354.066650	
3	1.674916	100.0	17.175274	59.0	15.0	19.680000	
4	11.972484	717.0	127.448502	341.0	117.0	146.791794	
...	...	...	...	...	...	...	
14782	4.300482	257.0	57.762333	83.0	62.0	73.462997	
14783	1.009842	60.0	15.513784	21.0	12.0	16.088200	
14784	7.035331	421.0	38.684837	282.0	48.0	58.903702	
14785	5.808548	347.0	134.723831	264.0	179.0	171.110306	
14786	5.906793	353.0	66.081528	275.0	68.0	80.578705	

14787 rows × 12 columns

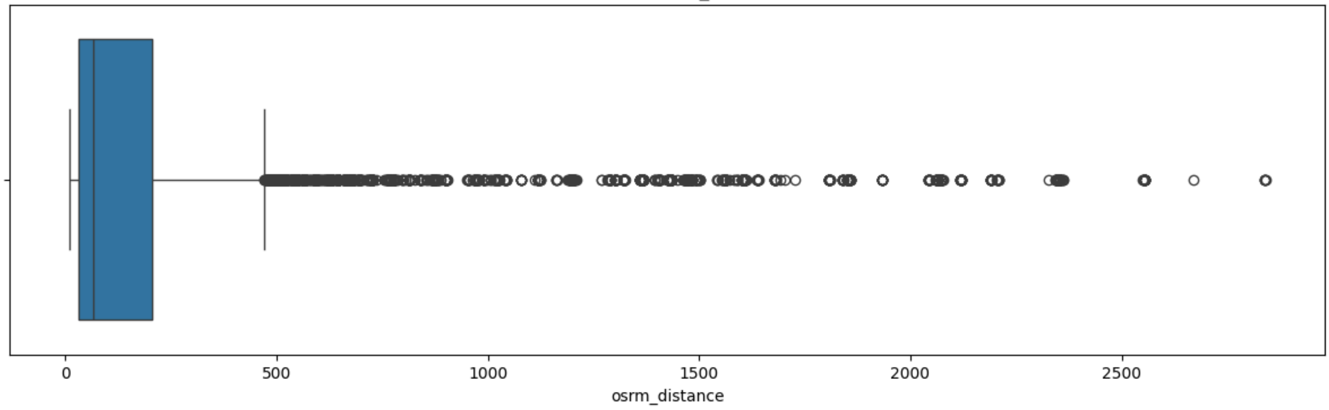
Next steps: Generate code with `numerical_features` | View recommended plots | New interactive sheet

```
for feature in numerical_features.columns:
  plt.figure(figsize=(15, 4))
  sns.boxplot(x=df_trip[feature])
  plt.title(f'Box Plot of {feature}')
  plt.show()
```

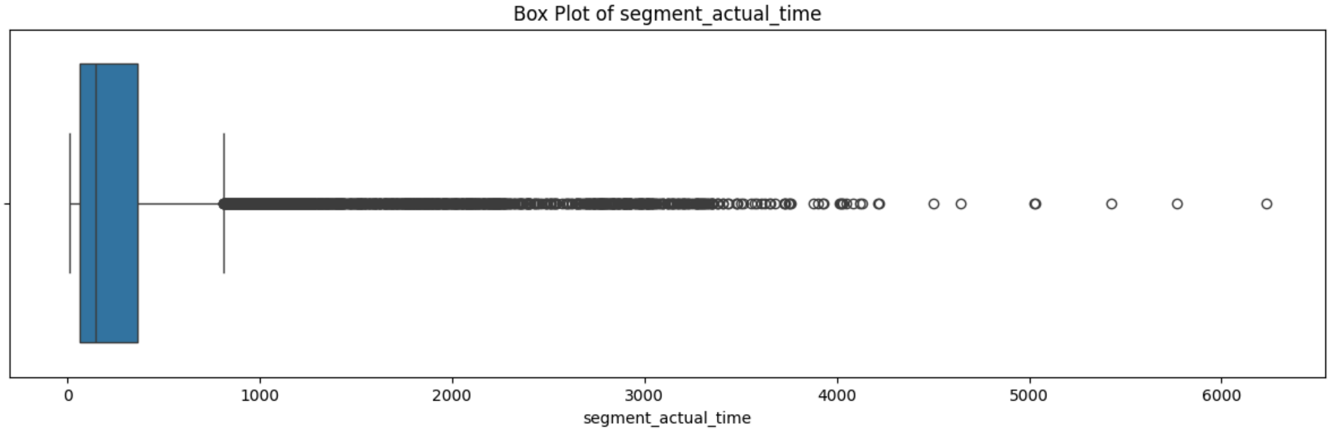

Box Plot of od_time_diff_hour
0 20 40 60 80 100 120
od_time_diff_hour
Box Plot of start_scan_to_end_scan
0 1000 2000 3000 4000 5000 6000 7000 8000
start_scan_to_end_scan
Box Plot of actual_distance_to_destination
0 500 1000 1500 2000
actual_distance_to_destination
Box Plot of actual_time
0 1000 2000 3000 4000 5000 6000
actual_time
Box Plot of osrm_time



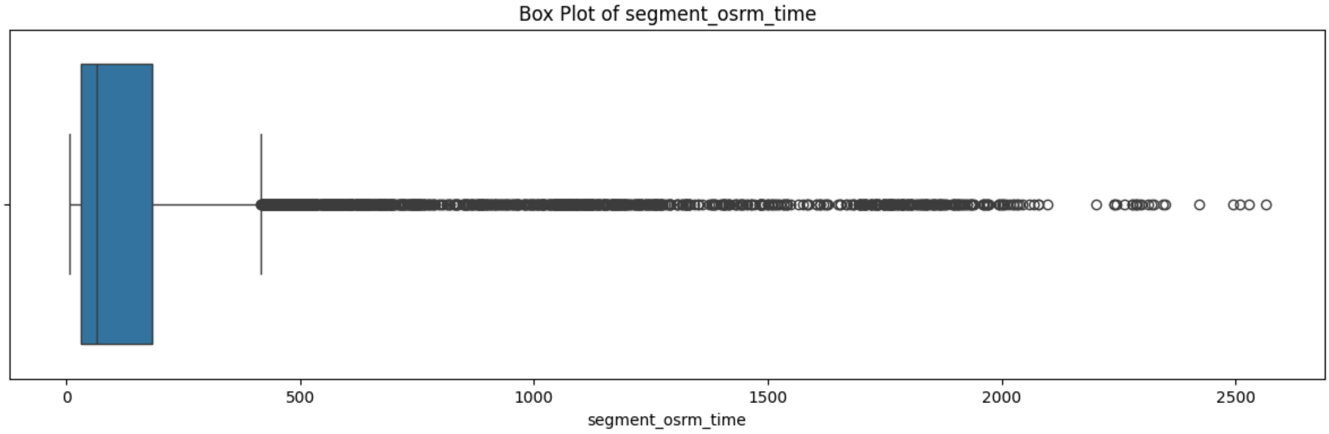
Box Plot of osrm_distance



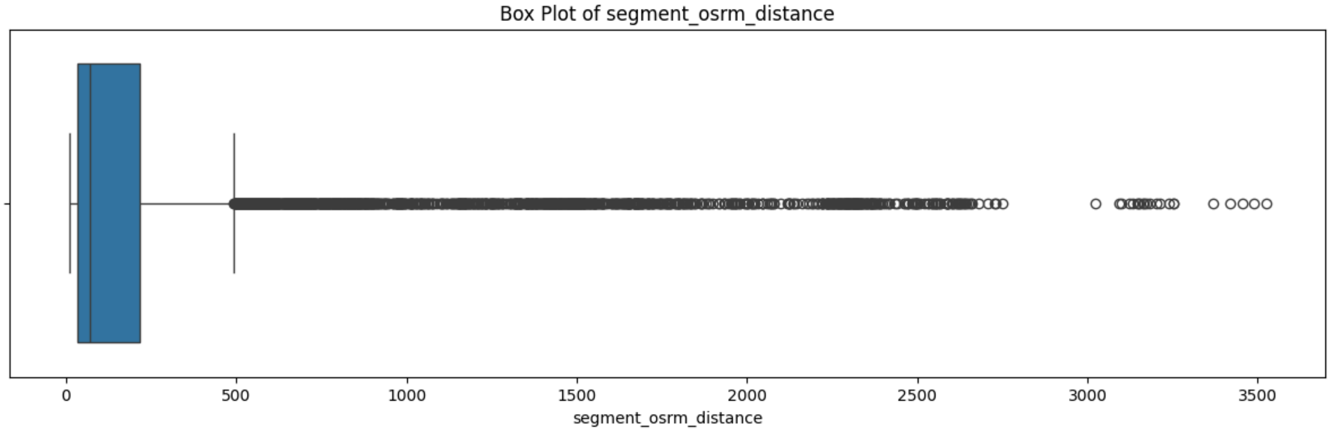
Box Plot of segment_actual_time



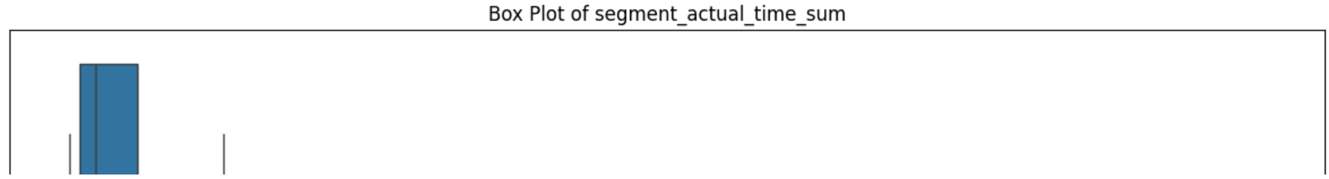
Box Plot of segment_osrm_time

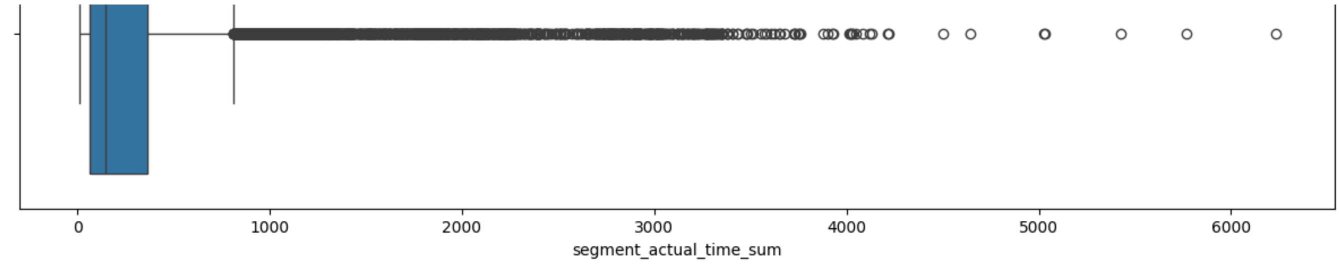


Box Plot of segment_osrm_distance

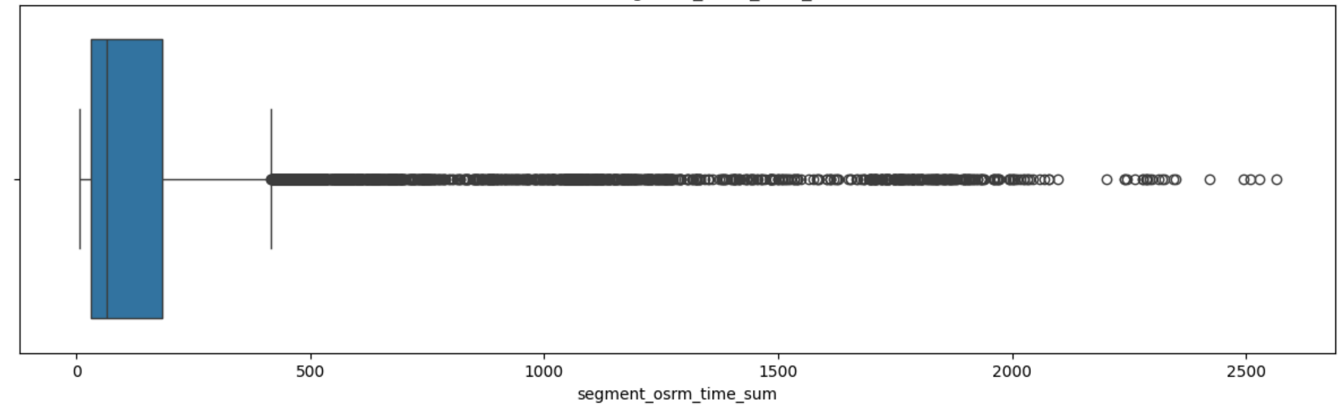


Box Plot of segment_actual_time_sum

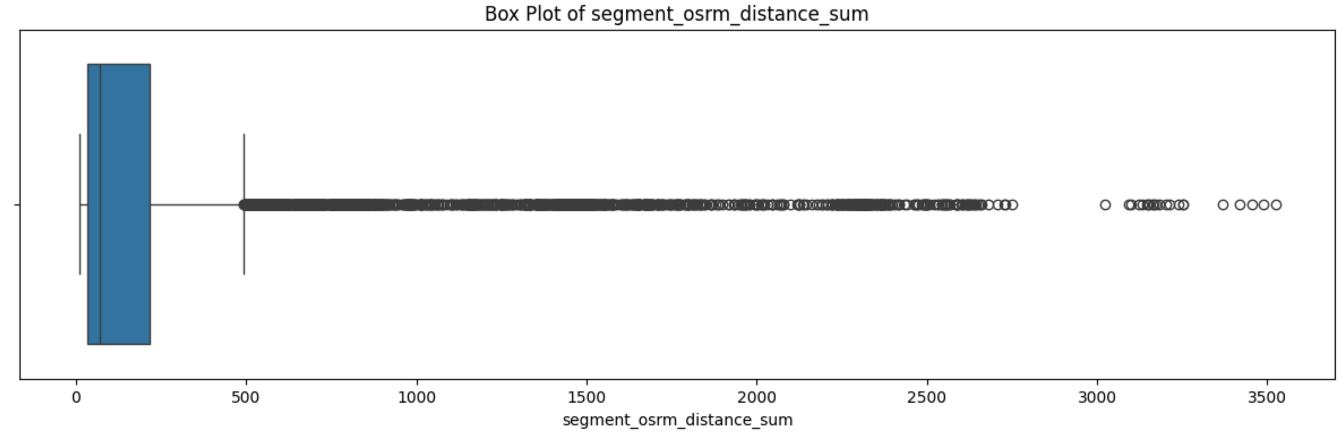




Box Plot of segment_osrm_time_sum



Box Plot of segment_osrm_distance_sum



```
# obtain the first quartile
Q1 = numerical_features.quantile(0.25)

# obtain the third quartile
Q3 = numerical_features.quantile(0.75)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

```
od_time_diff_hour                   8.063987
start_scan_to_end_scan            483.000000
actual_distance_to_destination    140.814157
actual_time                       300.000000
osrm_time                         139.000000
osrm_distance                     175.887303
segment_actual_time               298.000000
segment_osrm_time                 154.000000
segment_osrm_distance             183.981758
segment_actual_time_sum           298.000000
segment_osrm_time_sum             154.000000
segment_osrm_distance_sum         183.981758
dtype: float64
```

```
for i, col in enumerate(numerical_features):
    data = df_trip[col]
    display(data.to_frame())

    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1

    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    clipped_data = np.clip(data, lower_bound, upper_bound)
    print(f'Clipped data of {col}')
    display(clipped_data.to_frame())
    print()

    # Plot boxplot of the clipped data
    plt.figure(figsize=(15, 4))

    plt.subplot(121)
    sns.boxplot(x=clipped_data)
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of clipped {col}', fontfamily='serif', fontweight='bold', fontsize=12)

    filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]
    print(f'Filtered data of {col}')
    display(filtered_data.to_frame())
    print()

    plt.subplot(122)
    sns.boxplot(x=filtered_data)
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of filtered {col}', fontfamily='serif', fontweight='bold', fontsize=12)

    plt.show()
```

	od_time_diff_hour
0	37.668497
1	3.026865
2	65.572709
3	1.674916
4	11.972484
...	...
14782	4.300482
14783	1.009842
14784	7.035331
14785	5.808548
14786	5.906793

14787 rows × 1 columns

```
Clipped data of od_time_diff_hour
```

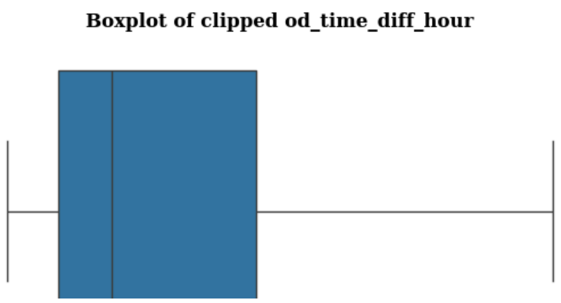
	od_time_diff_hour
0	22.654942
1	3.026865
2	22.654942
3	1.674916
4	11.972484
...	...
14782	4.300482
14783	1.009842
14784	7.035331
14785	5.808548
14786	5.906793

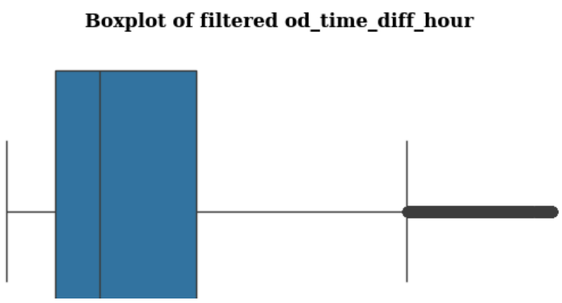
14787 rows × 1 columns

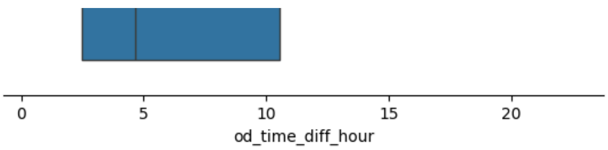
```
Filtered data of od_time_diff_hour
```

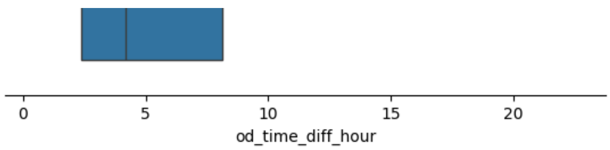
	od_time_diff_hour
1	3.026865
3	1.674916
4	11.972484
5	3.174797
6	1.633427
...	...
14782	4.300482
14783	1.009842
14784	7.035331
14785	5.808548
14786	5.906793

13512 rows × 1 columns









	start_scan_to_end_scan
0	2259.0
1	180.0
2	3933.0
3	100.0
4	717.0
...	...
14782	257.0
14783	60.0
14784	421.0
14785	347.0
14786	353.0

14787 rows × 1 columns

```
Clipped data of start_scan_to_end_scan
```

	start_scan_to_end_scan
0	1356.5
1	180.0
2	1356.5
3	100.0
4	717.0
...	...
14782	257.0
14783	60.0
14784	421.0
14785	347.0
14786	353.0

14787 rows × 1 columns

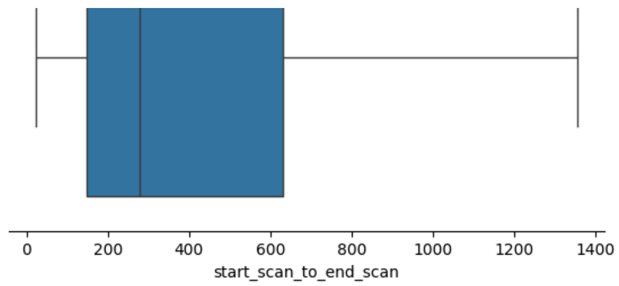
```
Filtered data of start_scan_to_end_scan
```

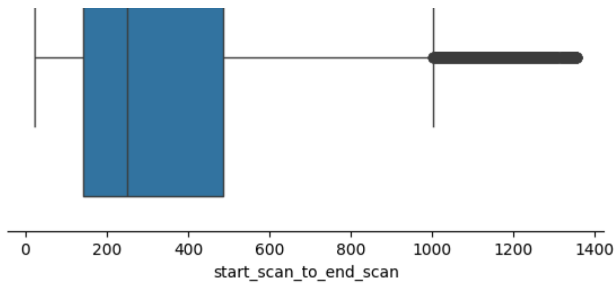
	start_scan_to_end_scan
1	180.0
3	100.0
4	717.0
5	189.0
6	98.0
...	...
14782	257.0
14783	60.0
14784	421.0
14785	347.0
14786	353.0

13505 rows × 1 columns

Boxplot of clipped start_scan_to_end_scan

Boxplot of filtered start_scan_to_end_scan





	actual_distance_to_destination
0	824.732849
1	73.186905
2	1927.404297
3	17.175274
4	127.448502
...	...
14782	57.762333
14783	15.513784
14784	38.684837
14785	134.723831
14786	66.081528

14787 rows × 1 columns

```
Clipped data of actual_distance_to_destination
```

	actual_distance_to_destination
0	374.812490
1	73.186905
2	374.812490
3	17.175274
4	127.448502
...	...
14782	57.762333
14783	15.513784
14784	38.684837
14785	134.723831
14786	66.081528

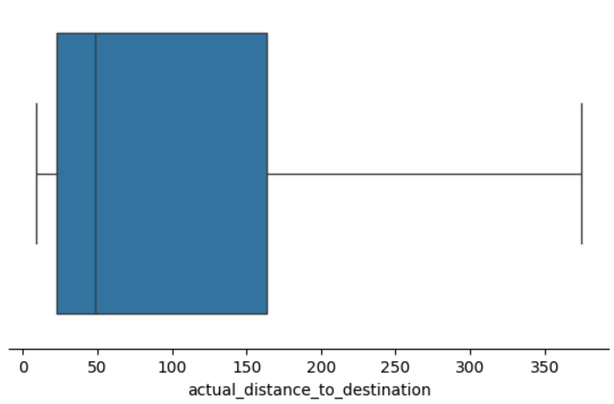
14787 rows × 1 columns

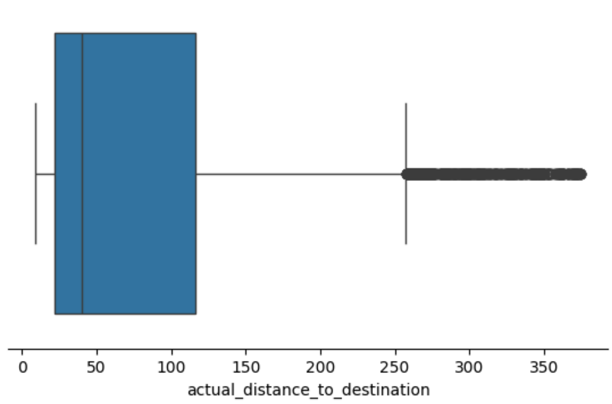
```
Filtered data of actual_distance_to_destination
```

	actual_distance_to_destination
1	73.186905
3	17.175274
4	127.448502
5	24.597050
6	9.100510
...	...
14782	57.762333
14783	15.513784
14784	38.684837
14785	134.723831
14786	66.081528

13335 rows × 1 columns

Boxplot of clipped actual_distance_to_destination **Boxplot of filtered actual_distance_to_destination**





	actual_time
0	1562.0
1	143.0
2	3347.0
3	59.0
4	341.0
...	...
14782	83.0
14783	21.0
14784	282.0
14785	264.0
14786	275.0

14787 rows × 1 columns

```
Clipped data of actual_time
```

	actual_time
0	817.0
1	143.0
2	817.0
3	59.0
4	341.0
...	...
14782	83.0
14783	21.0
14784	282.0
14785	264.0
14786	275.0

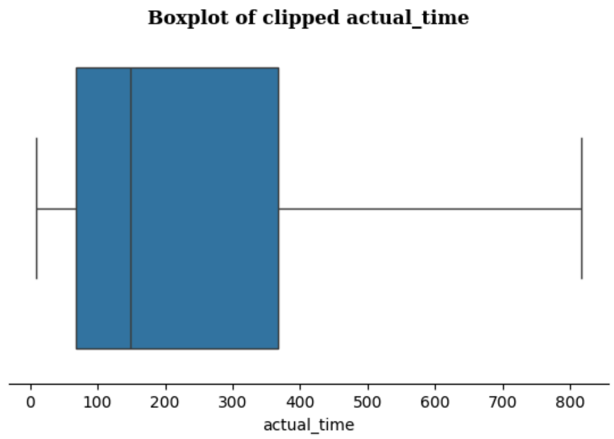
14787 rows × 1 columns

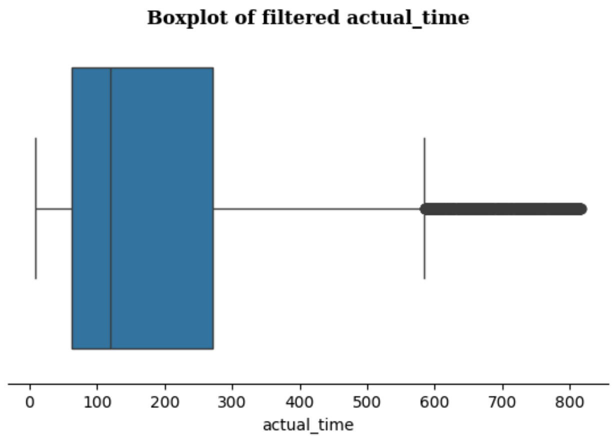
```
Filtered data of actual_time
```

	actual_time
1	143.0
3	59.0
4	341.0
5	61.0
6	24.0
...	...
14782	83.0
14783	21.0
14784	282.0
14785	264.0

14786	275.0

13141 rows × 1 columns





	osrm_time
0	717.0
1	68.0
2	1740.0
3	15.0
4	117.0
...	...
14782	62.0
14783	12.0
14784	48.0
14785	179.0
14786	68.0

14787 rows × 1 columns

```
Clipped data of osrm_time
```

	osrm_time
0	376.5
1	68.0
2	376.5
3	15.0
4	117.0
...	...
14782	62.0
14783	12.0
14784	48.0
14785	179.0
14786	68.0

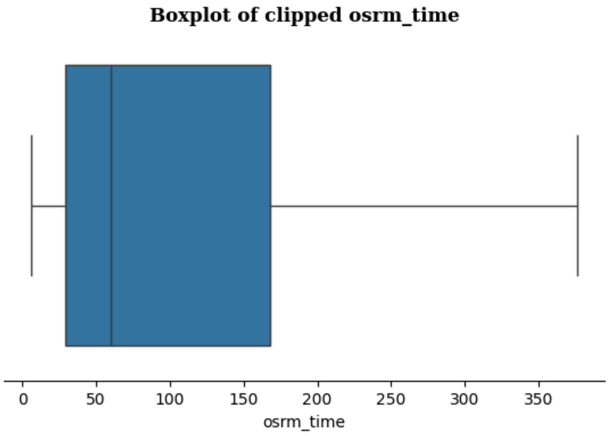
14787 rows × 1 columns

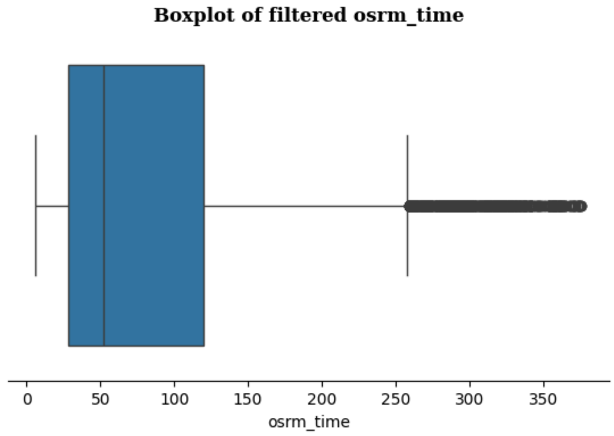
```
Filtered data of osrm_time
```

	osrm_time
1	68.0
3	15.0
4	117.0
5	23.0
6	13.0
...	...
14782	62.0

14783	12.0
14784	48.0
14785	179.0
14786	68.0

13281 rows × 1 columns





	osrm_distance
0	991.352295
1	85.111000
2	2354.066650
3	19.680000
4	146.791794
...	...
14782	73.462997
14783	16.088200
14784	58.903702
14785	171.110306
14786	80.578705

14787 rows × 1 columns

```
Clipped data of osrm_distance
```

	osrm_distance
0	470.475158
1	85.111000
2	470.475158
3	19.680000
4	146.791794
...	...
14782	73.462997
14783	16.088200
14784	58.903702
14785	171.110306
14786	80.578705

14787 rows × 1 columns

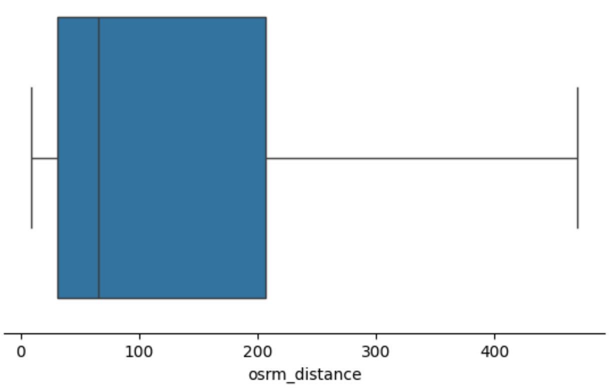
```
Filtered data of osrm_distance
```

	osrm_distance
1	85.111000
3	19.680000
4	146.791794
5	28.064701

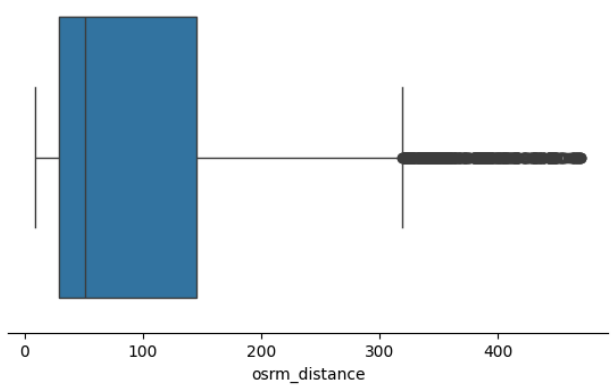
6	12.018400
...	...
14782	73.462997
14783	16.088200
14784	58.903702
14785	171.110306
14786	80.578705

13265 rows × 1 columns

Boxplot of clipped osrm_distance



Boxplot of filtered osrm_distance



	segment_actual_time
0	1548.0
1	141.0
2	3308.0
3	59.0
4	340.0
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

14787 rows × 1 columns

```
Clipped data of segment_actual_time
```

	segment_actual_time
0	811.0
1	141.0
2	811.0
3	59.0
4	340.0
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

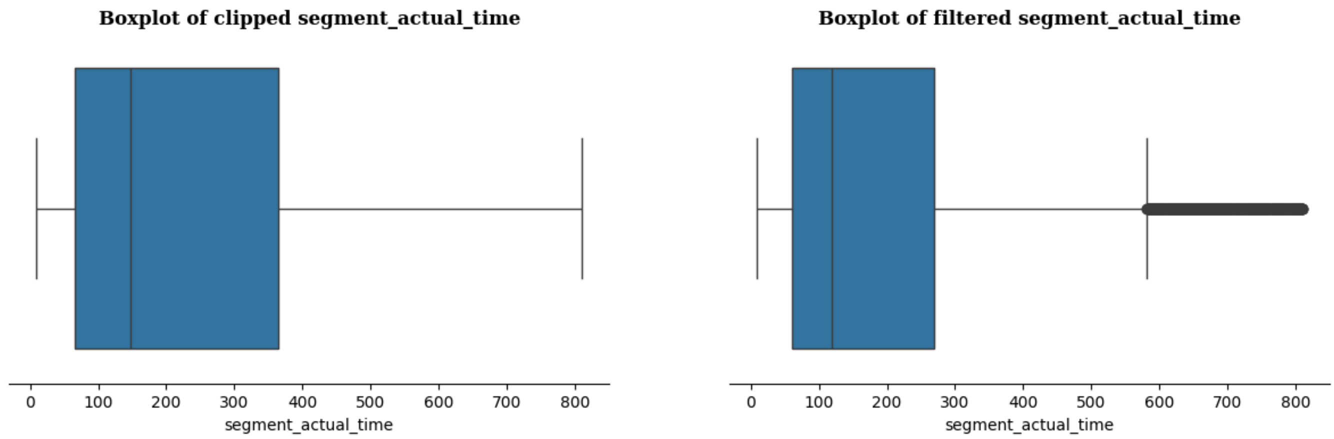
14787 rows × 1 columns

```
Filtered data of segment_actual_time
```

	segment_actual_time

1	141.0
3	59.0
4	340.0
5	60.0
6	24.0
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

13143 rows × 1 columns



	segment_osrm_time
0	1008.0
1	65.0
2	1941.0
3	16.0
4	115.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

14787 rows × 1 columns

```
Clipped data of segment_osrm_time
```

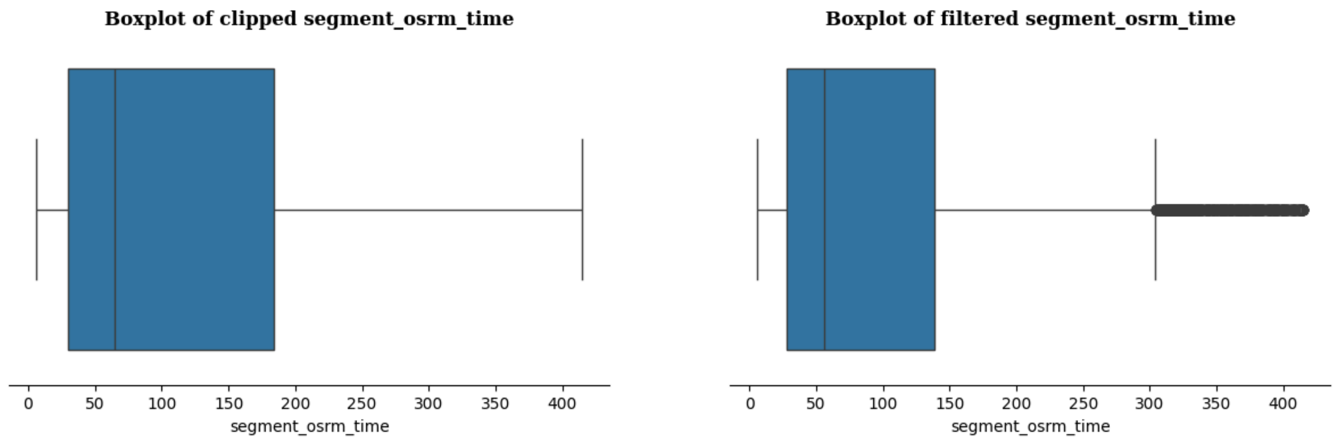
	segment_osrm_time
0	415.0
1	65.0
2	415.0
3	16.0
4	115.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

14787 rows × 1 columns

Filtered data of segment_osrm_time

	segment_osrm_time
1	65.0
3	16.0
4	115.0
5	23.0
6	13.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

13302 rows × 1 columns



	segment_osrm_distance
0	1320.473267
1	84.189400
2	2545.267822
3	19.876600
4	146.791901
...	...
14782	64.855103
14783	16.088299
14784	104.886597
14785	223.532394
14786	80.578705

14787 rows × 1 columns

Clipped data of segment_osrm_distance

	segment_osrm_distance
0	492.533245
1	84.189400
2	492.533245
3	19.876600
4	146.791901
...	...
14782	64.855103
14783	16.088299

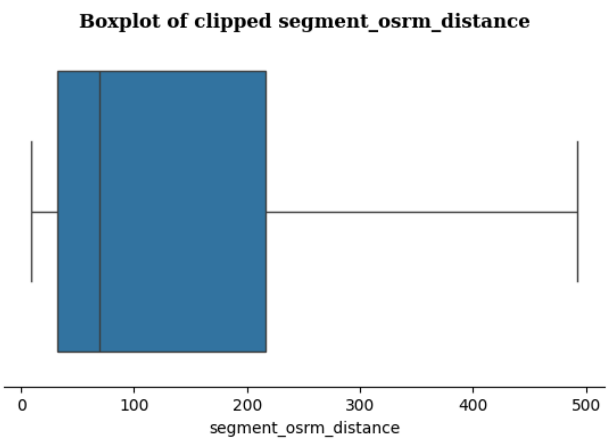
14784	104.886597
14785	223.532394
14786	80.578705

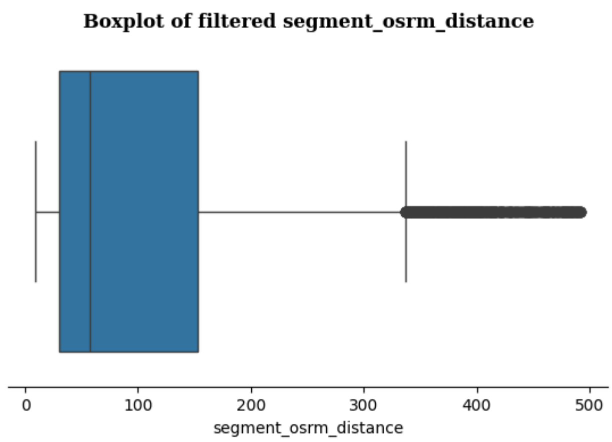
14787 rows × 1 columns

```
Filtered data of segment_osrm_distance
```

	segment_osrm_distance
1	84.189400
3	19.876600
4	146.791901
5	28.064701
6	12.018400
...	...
14782	64.855103
14783	16.088299
14784	104.886597
14785	223.532394
14786	80.578705

13237 rows × 1 columns





	segment_actual_time_sum
0	1548.0
1	141.0
2	3308.0
3	59.0
4	340.0
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

14787 rows × 1 columns

```
Clipped data of segment_actual_time_sum
```

	segment_actual_time_sum
0	811.0
1	141.0
2	811.0
3	59.0
4	340.0

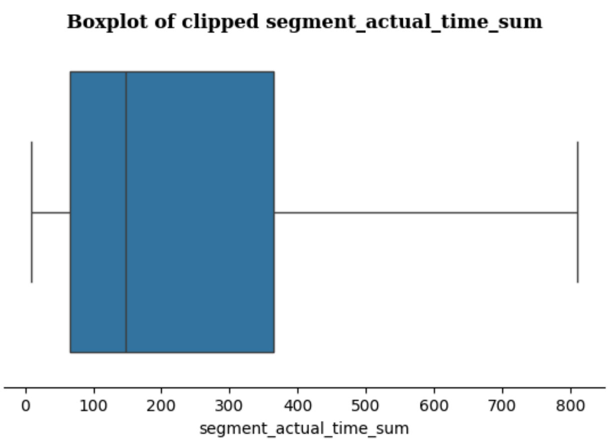
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

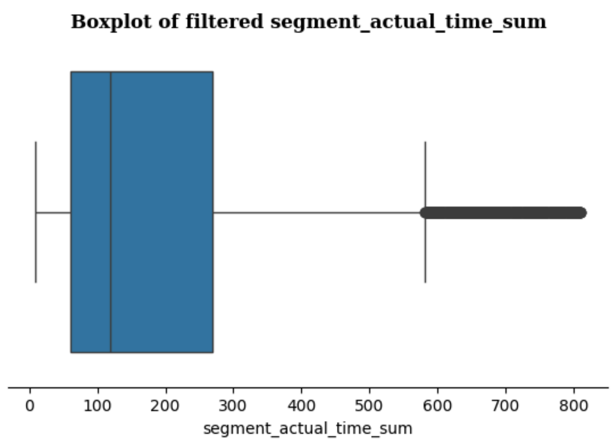
14787 rows × 1 columns

```
Filtered data of segment_actual_time_sum
```

	segment_actual_time_sum
1	141.0
3	59.0
4	340.0
5	60.0
6	24.0
...	...
14782	82.0
14783	21.0
14784	281.0
14785	258.0
14786	274.0

13143 rows × 1 columns





	segment_osrm_time_sum
0	1008.0
1	65.0
2	1941.0
3	16.0
4	115.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

14787 rows × 1 columns

```
Clipped data of segment_osrm_time_sum
```

	segment_osrm_time_sum
0	415.0
1	65.0

2	415.0
3	16.0
4	115.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

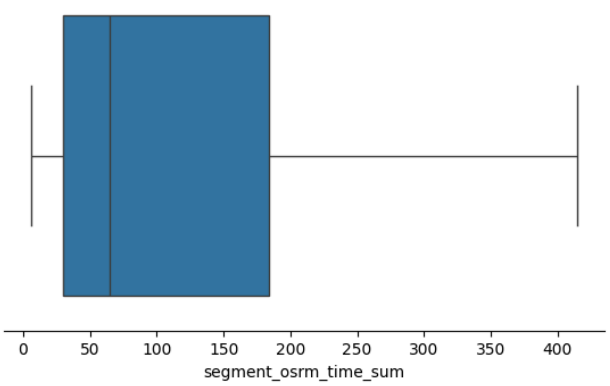
14787 rows × 1 columns

```
Filtered data of segment_osrm_time_sum
```

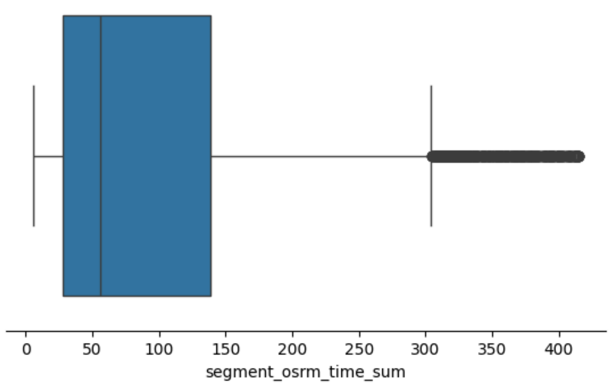
	segment_osrm_time_sum
1	65.0
3	16.0
4	115.0
5	23.0
6	13.0
...	...
14782	62.0
14783	11.0
14784	88.0
14785	221.0
14786	67.0

13302 rows × 1 columns

Boxplot of clipped segment_osrm_time_sum



Boxplot of filtered segment_osrm_time_sum



	segment_osrm_distance_sum
0	1320.473267
1	84.189400
2	2545.267822
3	19.876600
4	146.791901
...	...
14782	64.855103
14783	16.088299
14784	104.886597
14785	223.532394
14786	80.578705

14787 rows × 1 columns

Clipped data of segment_osrm_distance_sum

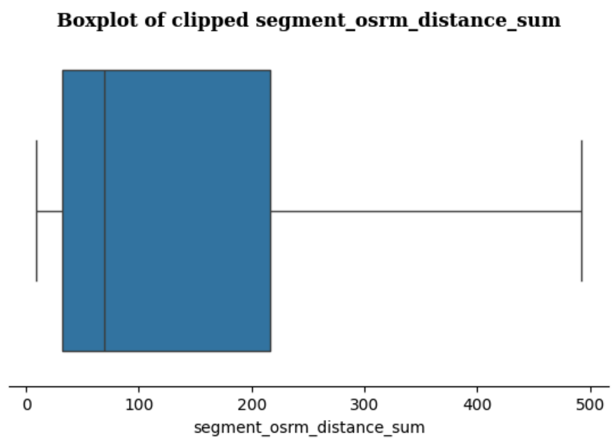
	segment_osrm_distance_sum
0	492.533245
1	84.189400
2	492.533245
3	19.876600
4	146.791901
...	...
14782	64.855103
14783	16.088299
14784	104.886597
14785	223.532394
14786	80.578705

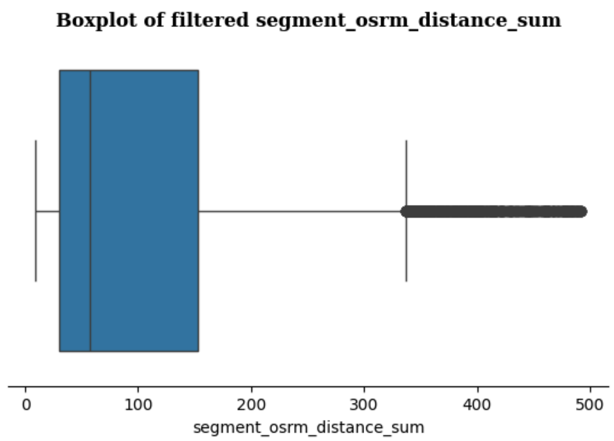
14787 rows × 1 columns

Filtered data of segment_osrm_distance_sum

	segment_osrm_distance_sum
1	84.189400
3	19.876600
4	146.791901
5	28.064701
6	12.018400
...	...
14782	64.855103
14783	16.088299
14784	104.886597
14785	223.532394
14786	80.578705

13237 rows × 1 columns





Insight:

1. Upon examining the data after outlier removal, it is evident that outliers still persist. This highlights the importance of recognizing that the first (Q1) and third (Q3) quartiles do not always have to be the 25th and 75th percentiles, respectively. By adjusting Q1 and Q3 to the 10th and 90th percentiles, we can observe the impact on the data distribution through plotting.
2. Clipped data, which replaces outlier values with specified boundary values, and filtered data, which reduces the number of outliers, were both utilized for further analysis. This dual approach provides a comprehensive understanding of the data's behavior and ensures robust analysis.

```
num_df = numerical_features.copy()
num_df
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	37.668497	2259.0	824.732849	1562.0	717.0	991.352295	
1	3.026865	180.0	73.186905	143.0	68.0	85.111000	
2	65.572709	3933.0	1927.404297	3347.0	1740.0	2354.066650	
3	1.674916	100.0	17.175274	59.0	15.0	19.680000	
4	11.972484	717.0	127.448502	341.0	117.0	146.791794	
...	...	...	...	...	...	...	
14782	4.300482	257.0	57.762333	83.0	62.0	73.462997	
14783	1.009842	60.0	15.513784	21.0	12.0	16.088200	
14784	7.035331	421.0	38.684837	282.0	48.0	58.903702	
14785	5.808548	347.0	134.723831	264.0	179.0	171.110306	
14786	5.906793	353.0	66.081528	275.0	68.0	80.578705	

14787 rows × 12 columns

Next steps: Generate code with `num_df` View recommended plots New interactive sheet

```
Q1 = np.percentile(num_df[numerical_columns], 25)
Q3 = np.percentile(num_df[numerical_columns], 75)
IQR = Q3 - Q1

lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

clipped_num_df = np.clip(num_df[numerical_columns], lower_bound, upper_bound)
display(clipped_num_df)

filtered_num_df = num_df[numerical_columns][(num_df[numerical_columns] >= lower_bound) | (num_df[numerical_columns] <= upper_bound
display(filtered_num_df)
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	37.668497	543.285302	543.285302	543.285302	543.285302	543.285302	
1	3.026865	180.000000	73.186905	143.000000	68.000000	85.111000	
2	65.572709	543.285302	543.285302	543.285302	543.285302	543.285302	
3	1.674916	100.000000	17.175274	59.000000	15.000000	19.680000	
4	11.972484	543.285302	127.448502	341.000000	117.000000	146.791794	
...	...	...	...	...	...	...	
14782	4.300482	257.000000	57.762333	83.000000	62.000000	73.462997	
14783	1.009842	60.000000	15.513784	21.000000	12.000000	16.088200	
14784	7.035331	421.000000	38.684837	282.000000	48.000000	58.903702	
14785	5.808548	347.000000	134.723831	264.000000	179.000000	171.110306	
14786	5.906793	353.000000	66.081528	275.000000	68.000000	80.578705	

14787 rows × 12 columns

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	37.668497	2259.0	824.732849	1562.0	717.0	991.352295	
1	3.026865	180.0	73.186905	143.0	68.0	85.111000	
2	65.572709	3933.0	1927.404297	3347.0	1740.0	2354.066650	
3	1.674916	100.0	17.175274	59.0	15.0	19.680000	
4	11.972484	717.0	127.448502	341.0	117.0	146.791794	
...	...	...	...	...	...	...	
14782	4.300482	257.0	57.762333	83.0	62.0	73.462997	
14783	1.009842	60.0	15.513784	21.0	12.0	16.088200	
14784	7.035331	421.0	38.684837	282.0	48.0	58.903702	
14785	5.808548	347.0	134.723831	264.0	179.0	171.110306	
14786	5.906793	353.0	66.081528	275.0	68.0	80.578705	

14787 rows × 12 columns

Next steps: code clipped_num_df | recommended | interactive | code filtered_num_df | recommended | interactive

4.3 Perform one-hot encoding on categorical features.

```
categorical_cols = ['data', 'route_type']

# Initialize OneHotEncoder
ohe = OneHotEncoder(sparse_output=False)

# Fit and transform the categorical columns
encoded_cat_cols = ohe.fit_transform(df_trip[categorical_cols])

# Create a DataFrame with the encoded columns
categorical_encoded_df = pd.DataFrame(encoded_cat_cols, columns=ohe.get_feature_names_out(categorical_cols))

# Display the encoded DataFrame
display(categorical_encoded_df)
```

	data_test	data_training	route_type_Carting	route_type_FTL
0	0.0	1.0	0.0	1.0
1	0.0	1.0	1.0	0.0
2	0.0	1.0	0.0	1.0
3	0.0	1.0	1.0	0.0
4	0.0	1.0	0.0	1.0
...	...	...	...	...
14782	1.0	0.0	1.0	0.0
14783	1.0	0.0	1.0	0.0
14784	1.0	0.0	1.0	0.0
14785	1.0	0.0	1.0	0.0
14786	1.0	0.0	0.0	1.0

14787 rows × 4 columns

Next steps: Generate code with categorical_encoded_df | View recommended plots | New interactive sheet

```
# Concatenate the original DataFrame with the encoded DataFrame
encoded_df = pd.concat([df_trip, categorical_encoded_df], axis=1)
display(encoded_df)
```

	trip_uuid	data	route_type	od_start_time	od_end_time	od_time_diff_hour	trip_creation_time	trip_crea
0	trip-153671041653548748	training	FTL	2018-09-12 16:39:46.858469	2018-09-12 16:39:46.858469	37.668497	2018-09-12 00:00:16.535741	
1	trip-153671042288605164	training	Carting	2018-09-12 02:03:09.655591	2018-09-12 02:03:09.655591	3.026865	2018-09-12 00:00:22.886430	
2	trip-153671043369099517	training	FTL	2018-09-14 03:40:17.106733	2018-09-14 03:40:17.106733	65.572709	2018-09-12 00:00:33.691250	
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	1.674916	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	2018-09-12 03:54:43.114421	11.972484	2018-09-12 00:02:09.740725	
...	...	...	...	...	...	...	...	
14782	trip-153861095625827784	test	Carting	2018-10-03 23:55:56.258533	2018-10-04 06:41:25.409035	4.300482	2018-10-03 23:55:56.258533	
14783	trip-153861104386292051	test	Carting	2018-10-03 23:57:23.863155	2018-10-04 00:57:59.294434	1.009842	2018-10-03 23:57:23.863155	
14784	trip-153861106442901555	test	Carting	2018-10-04 02:51:27.075797	2018-10-04 02:51:27.075797	7.035331	2018-10-03 23:57:44.429324	
14785	trip-153861115439069069	test	Carting	2018-10-03 23:59:14.390954	2018-10-04 02:29:04.272194	5.808548	2018-10-03 23:59:14.390954	
14786	trip-153861118270144424	test	FTL	2018-10-04 03:58:40.726547	2018-10-04 03:58:40.726547	5.906793	2018-10-03 23:59:42.701692	

14787 rows × 36 columns

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

```
numerical_columns
```

```
['od_time_diff_hour',
 'start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance',
 'segment_actual_time_sum',
 'segment_osrm_time_sum',
 'segment_osrm_distance_sum']
```

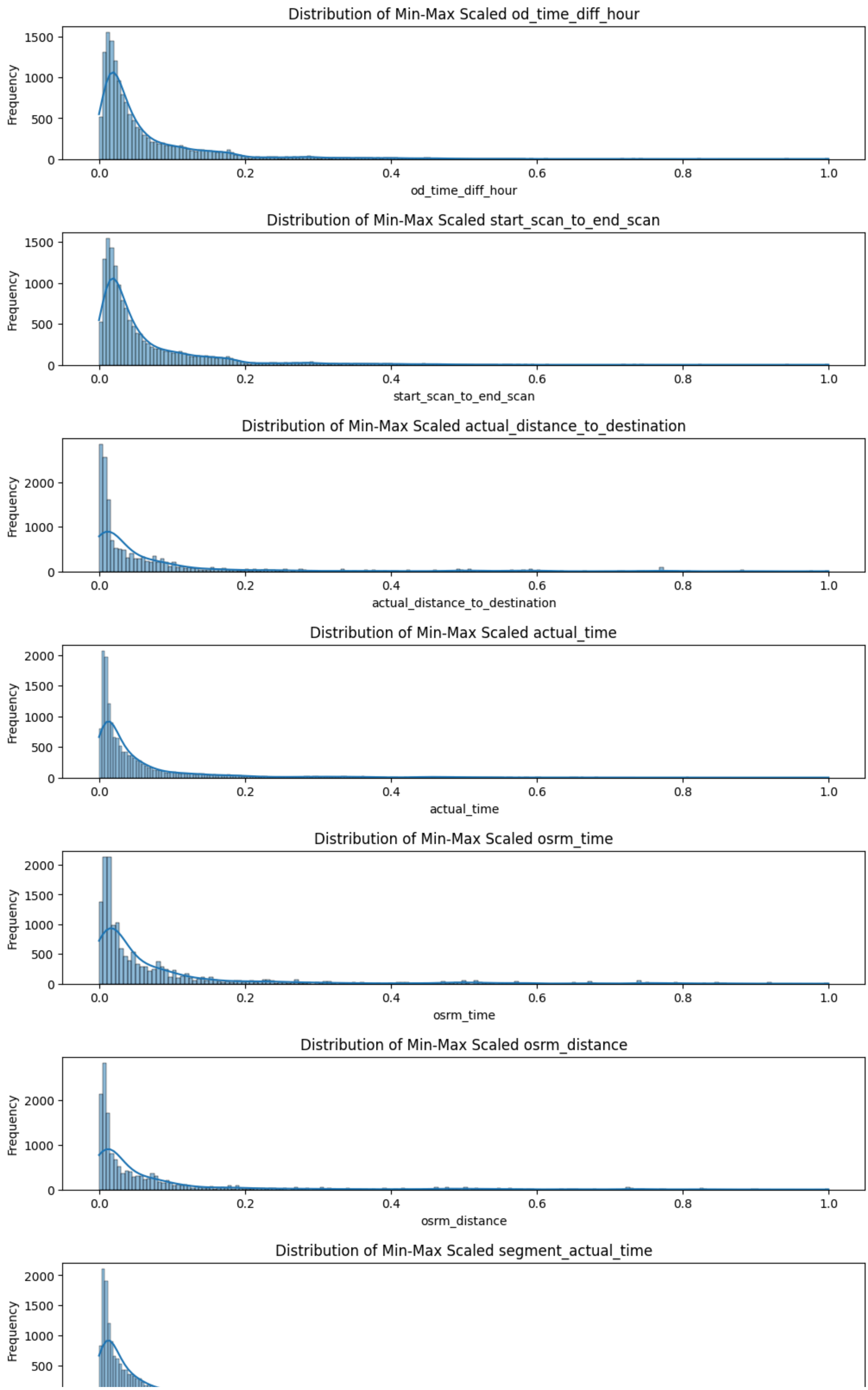
```
# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Transform the numerical features
scaled_numerical_features = scaler.fit_transform(df_trip[numerical_columns])

# Create a new DataFrame with the scaled features
scaled_numerical_df = pd.DataFrame(scaled_numerical_features, columns=numerical_columns)
scaled_numerical_df
```

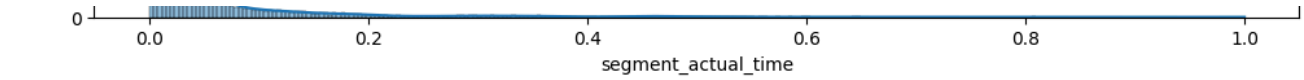
	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	0.284016	0.283937	0.374613	0.248242	0.350938	0.346972	
1	0.020082	0.019937	0.029476	0.021419	0.030602	0.026859	
2	0.496617	0.496508	0.880999	0.533568	0.855874	0.828325	
3	0.009782	0.009778	0.003753	0.007992	0.004442	0.003747	
4	0.088239	0.088127	0.054395	0.053069	0.054788	0.048647	
...	...	...	...	...	...	...	
14782	0.029786	0.029714	0.022392	0.011829	0.027641	0.022745	
14783	0.004715	0.004698	0.002990	0.001918	0.002962	0.002478	
14784	0.050623	0.050540	0.013631	0.043638	0.020731	0.017602	
14785	0.041276	0.041143	0.057736	0.040761	0.085390	0.057237	
14786	0.042024	0.041905	0.026213	0.042519	0.030602	0.025258	

14787 rows × 12 columns

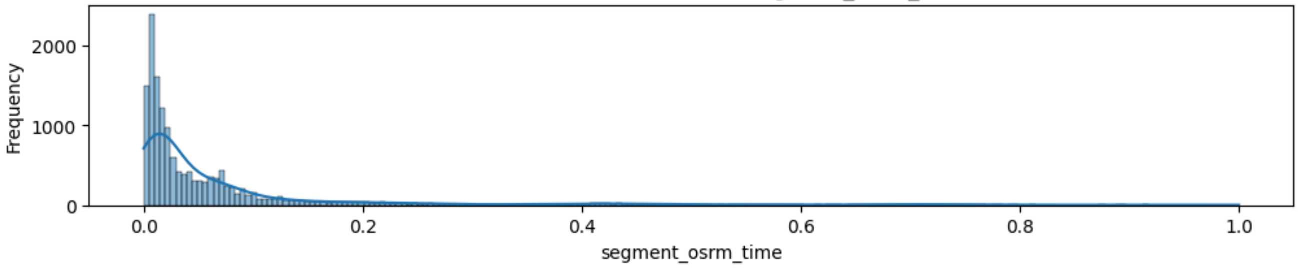
Next steps: Generate code with scaled_numerical_df | View recommended plots | New interactive sheet

```
for i, col in enumerate(numerical_columns):
    plt.figure(figsize=(12, 2))
    sns.histplot(scaled_numerical_df[col], kde=True)
    plt.title(f"Distribution of Min-Max Scaled {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

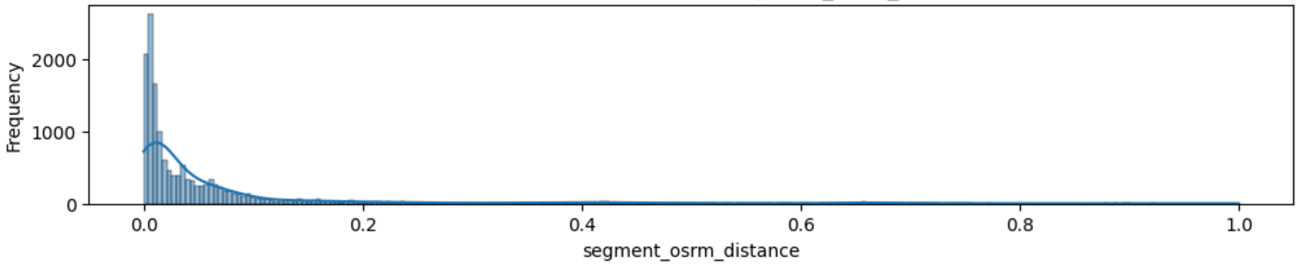

Distribution of Min-Max Scaled od_time_diff_hour
Frequency
od_time_diff_hour
Distribution of Min-Max Scaled start_scan_to_end_scan
Frequency
start_scan_to_end_scan
Distribution of Min-Max Scaled actual_distance_to_destination
Frequency
actual_distance_to_destination
Distribution of Min-Max Scaled actual_time
Frequency
actual_time
Distribution of Min-Max Scaled osrm_time
Frequency
osrm_time
Distribution of Min-Max Scaled osrm_distance
Frequency
osrm_distance
Distribution of Min-Max Scaled segment_actual_time
Frequency



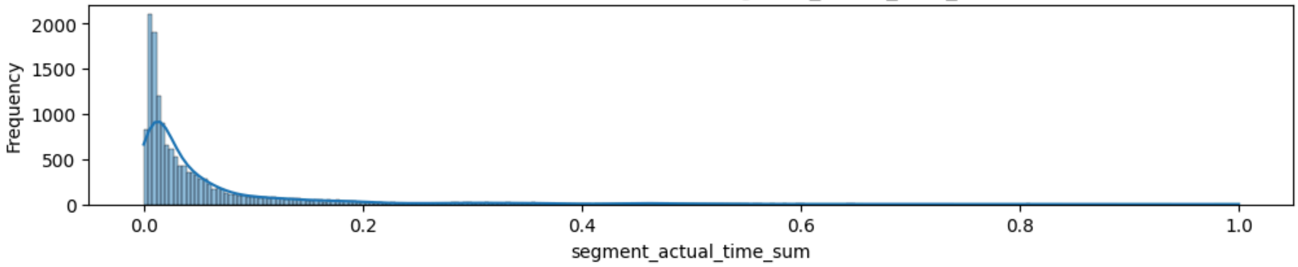
Distribution of Min-Max Scaled segment_osrm_time



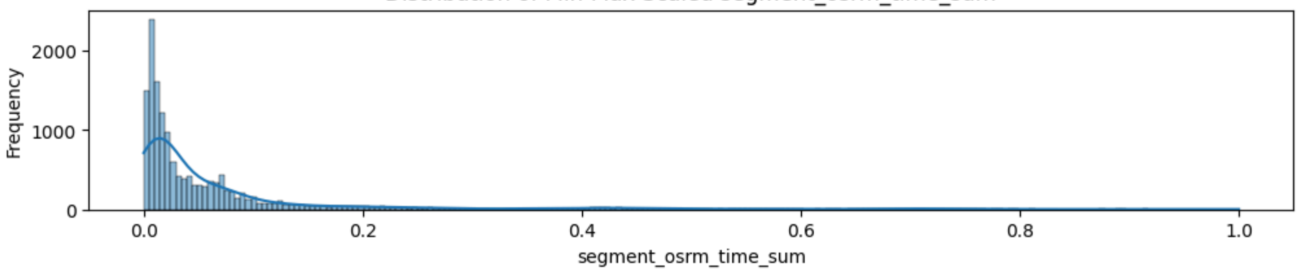
Distribution of Min-Max Scaled segment_osrm_distance



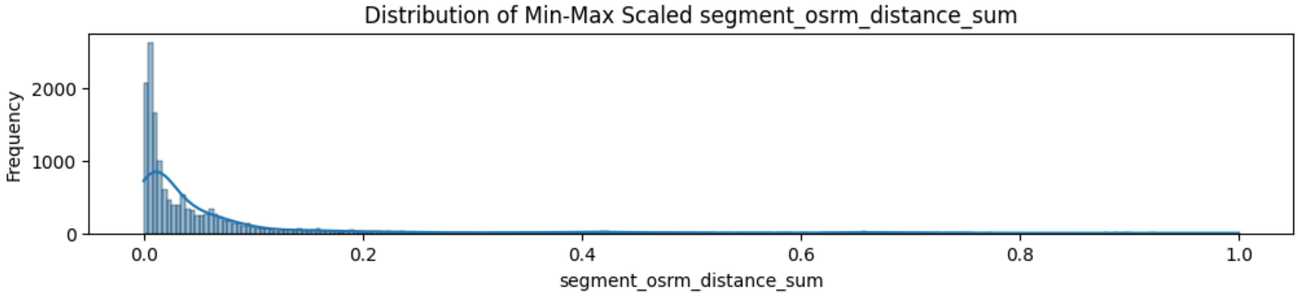
Distribution of Min-Max Scaled segment_actual_time_sum



Distribution of Min-Max Scaled segment_osrm_time_sum



Distribution of Min-Max Scaled segment_osrm_distance_sum



Insight on Standardization and Data Distribution

Standardization is a powerful technique for scaling data, but it's important to note that it works best when the data follows a normal (Gaussian) distribution. If the data is not Gaussian, standardization might not be the most effective approach. Instead, other scaling methods, such as min-max scaling or robust scaling, could be more appropriate for non-Gaussian data distributions. Understanding the underlying distribution of your data is crucial for selecting the right preprocessing technique and ensuring accurate and meaningful analysis.

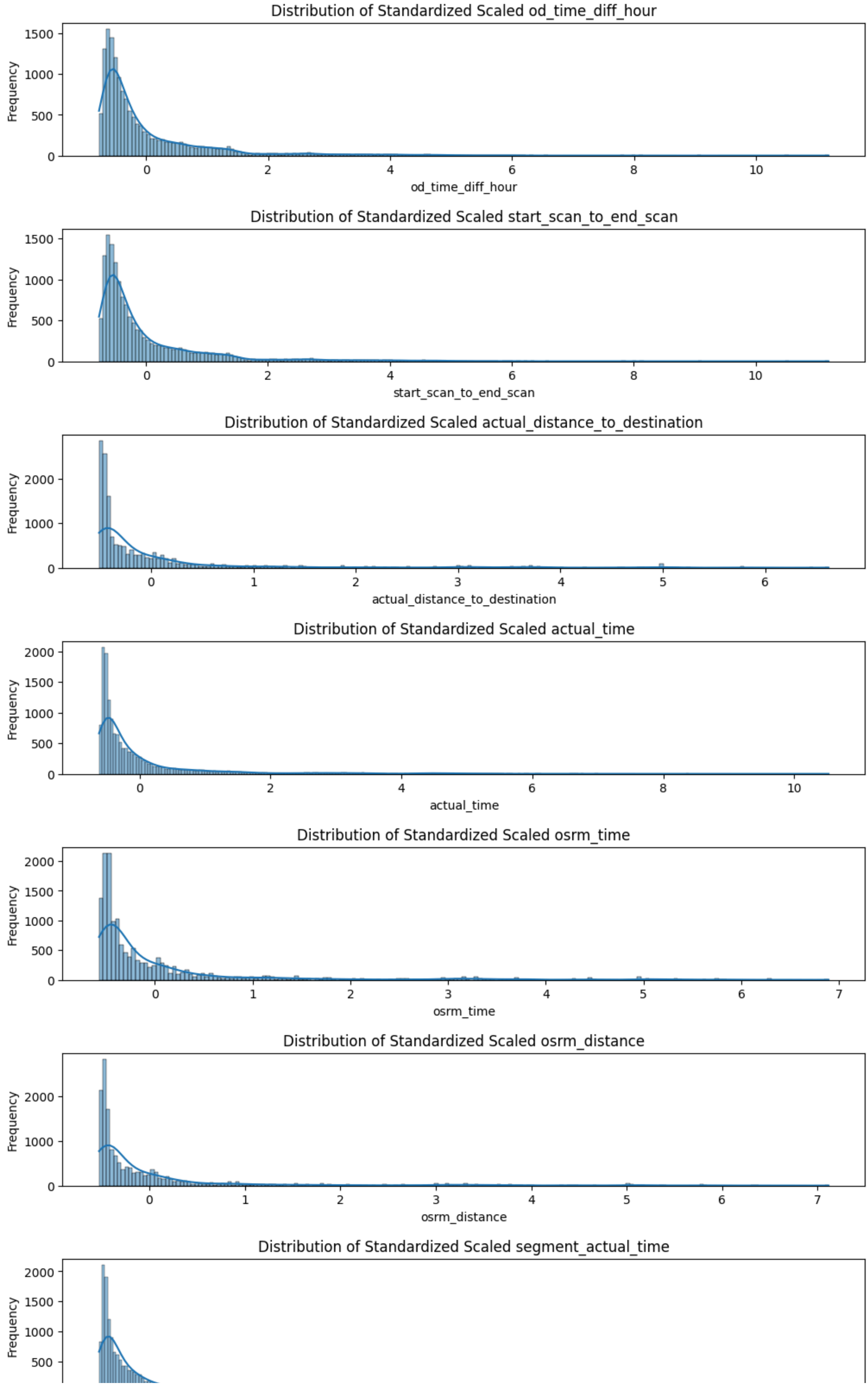
```
# Standardizing the numerical features using StandardScaler
standard_scaler = StandardScaler()
standard_scaled = standard_scaler.fit_transform(df_trip[numerical_columns])
# Converting the scaled features back to a dataframe
standard_scaled_df = pd.DataFrame(standard_scaled, columns=numerical_columns)
standard_scaled_df
```

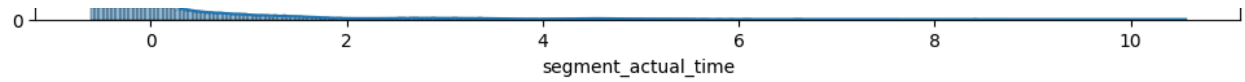
	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segme
0	2.625886	2.627598	2.162548	2.147277	2.048290	2.125107	
1	-0.529518	-0.530859	-0.297563	-0.379887	-0.342571	-0.320538	
2	5.167598	5.170772	5.772034	5.326268	5.816936	5.802622	
3	-0.652664	-0.652397	-0.480911	-0.529486	-0.537818	-0.497115	
4	0.285312	0.284962	-0.119943	-0.027259	-0.162059	-0.154082	
...	...	...	...	...	...	...	
14782	-0.413508	-0.413880	-0.348054	-0.486744	-0.364674	-0.351972	
14783	-0.713243	-0.713166	-0.486350	-0.597162	-0.548870	-0.506808	
14784	-0.164399	-0.164728	-0.410502	-0.132335	-0.416249	-0.391263	
14785	-0.276143	-0.277150	-0.096128	-0.164392	0.066344	-0.088455	
14786	-0.267194	-0.268034	-0.320822	-0.144802	-0.342571	-0.332769	

14787 rows × 12 columns

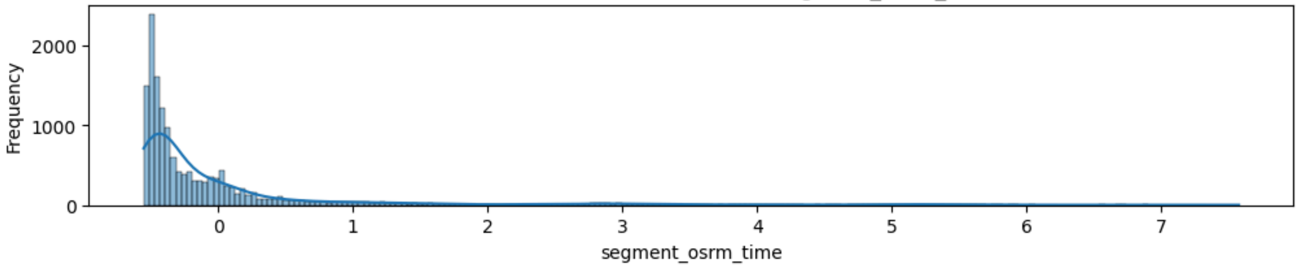
Next steps: Generate code with standard_scaled_df | View recommended plots | New interactive sheet

```
for i, col in enumerate(numerical_columns):
    plt.figure(figsize=(12, 2))
    sns.histplot(standard_scaled_df[col], kde=True)
    plt.title(f"Distribution of Standardized Scaled {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

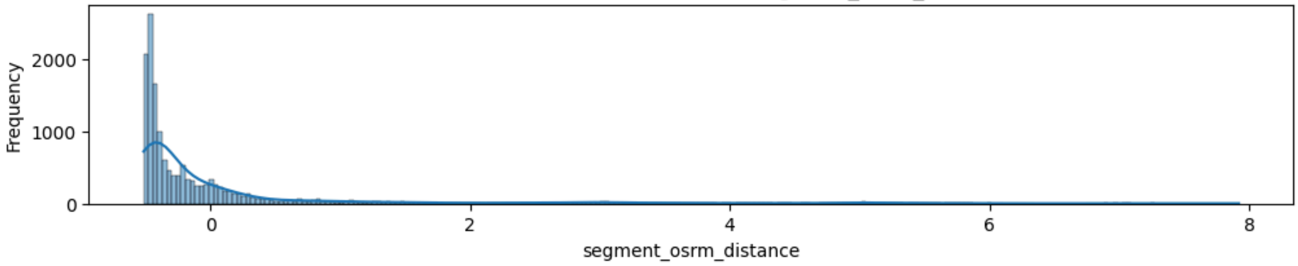

Distribution of Standardized Scaled od_time_diff_hour
Frequency
od_time_diff_hour
Distribution of Standardized Scaled start_scan_to_end_scan
Frequency
start_scan_to_end_scan
Distribution of Standardized Scaled actual_distance_to_destination
Frequency
actual_distance_to_destination
Distribution of Standardized Scaled actual_time
Frequency
actual_time
Distribution of Standardized Scaled osrm_time
Frequency
osrm_time
Distribution of Standardized Scaled osrm_distance
Frequency
osrm_distance
Distribution of Standardized Scaled segment_actual_time
Frequency



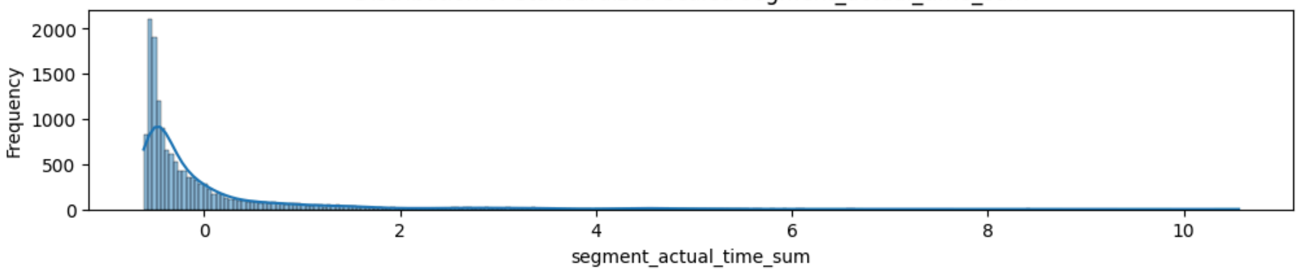
Distribution of Standardized Scaled segment_osrm_time



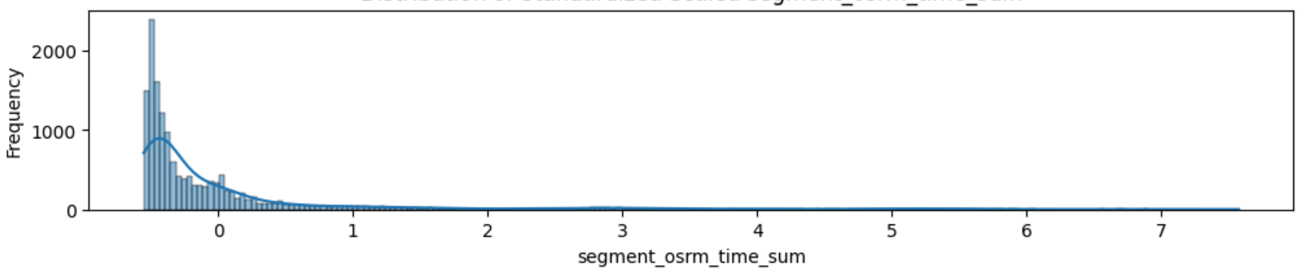
Distribution of Standardized Scaled segment_osrm_distance



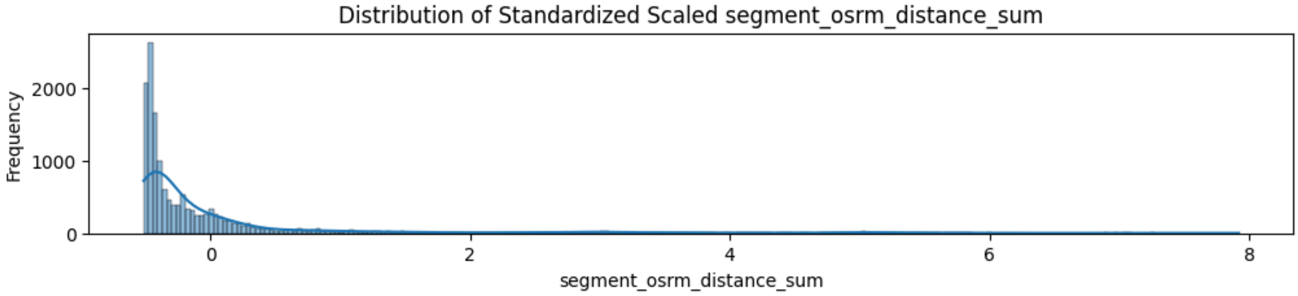
Distribution of Standardized Scaled segment_actual_time_sum



Distribution of Standardized Scaled segment_osrm_time_sum



Distribution of Standardized Scaled segment_osrm_distance_sum



🔍 **Insight:**

1. **Trip-Level Aggregation:** Aggregating data at the trip level provides insights into the overall delivery performance for each trip. For example, we can identify the total duration of a trip, the total distance covered, or the number of segments in a trip.
2. **Outlier Detection and Treatment:** The identification of outliers in numerical features can indicate exceptional cases or potential data errors. Applying appropriate treatment techniques such as clipping or filtering can ensure the robustness of subsequent analyses.
3. **Categorical Feature Encoding:** Applying one-hot encoding to categorical features like 'route_type' and 'data' allows for more effective utilization of this data during modeling, particularly in machine learning algorithms.
4. **Data Scaling and Standardization:** Normalizing or standardizing numerical features can improve the performance of machine learning models, particularly those that are sensitive to the scale of features. In our case, scaling features like 'od_time_diff_hour' and 'actual_time' helps improve model training and prediction accuracy.

These insights provide a deeper understanding of the dataset and guide us in preparing it for modeling and analysis, optimizing the accuracy and reliability of our results.

🧪 5. Hypothesis Testing

5.1 🧠 Perform Hypothesis Testing

Perform hypothesis testing / visual analysis between:

> 📌 **actual_time** aggregated value and **OSRM time** aggregated value.

> 📌 **actual_time** aggregated value and **segment actual time** aggregated value.

> 📌 **OSRM distance** aggregated value and **segment OSRM distance** aggregated value.

> 📌 **OSRM time** aggregated value and **segment OSRM time** aggregated value.

🔻 **STEP 1**: Set up Null Hypothesis

> **Null Hypothesis (Ho)** - There is no significant difference in the mean values between `column1` and `column2`.

- $H_o: \mu_{col1} = \mu_{col2}$

> **Alternate Hypothesis (Ha)** - There is a significant difference in the mean values between `column1` and `column2`.

- $H_a: \mu_{col1} \neq \mu_{col2}$

🔻 **STEP 2**: Check Basic Assumptions for the Hypothesis

- **Normality Checks** :

  > Use **QQ Plot** & **Prob Plot** to visually check distribution.

  - Confirm with **Shapiro-Wilks Test** .
  - Confirm with **Anderson-Darling Test** .
- **Homogeneity of Variances**:

  > Use **Levene's Test** to check if variances are equal between the two groups.

🔻 **STEP 3**: Define Test Statistics; Distribution of T Under Ho

- We know that the test statistic for a **T-Test** follows a **T-distribution**.

> For **independent variables**:

- If data follows normal distribution use **ttest_ind**.
- If not normal use **Mann-Whitney U Test** (Non-Parametric).

> For **dependent variables** (paired T-test):

- If data follows normal distribution use **ttest_rel**.
- If not normal use **Wilcoxon Signed Rank Test** (Non-Parametric).

STEP 4: Decide the Kind of Test

- We will be performing a **Two-Tailed T-Test** ⚖️.

STEP 5: Compute the p-value and Fix Alpha

- Compute the **t-stat** value using the **ttest** function from `scipy.stats` .
- Set **alpha = 0.05** (i.e. confidence level = 95%).

STEP 6: Compare p-value and Alpha

- Based on the **p-value**, we will either **accept** or **reject Ho**:

> **p-val < alpha** 🔴: Reject Ho (Significant Difference).

> **p-val > alpha** 🟢: Accept Ho (No Significant Difference).

```
class NormalityCheck:
    def __init__(self, name, col):
        self.name = name
        self.col = col

    def shapiro_and_anderson(self):
        print(f"Performing SHAPIRO & ANDERSON-DARLING TEST for '{self.name}' column\n")

        # Shapiro-Wilk Test
        shapiro_stat, p_val = shapiro(self.col)
        print(f'Shapiro-Wilk Test\n{self.name} - Data is {"not Gaussian" if p_val < 0.05 else "Gaussian"} (p-value: {p_val})\n')

        # Anderson-Darling Test
        result = anderson(self.col)
        print(f"Anderson-Darling Test\n{self.name} - Data {'does not follow' if result.statistic > result.critical_values[2] else 'f
        print('-'*50)

    def boxcox_transformation(self):
        print(f'Performing BOXCOX transformation on {self.name} column')
        transformed_data, best_lambda = boxcox(self.col)
        self.col = transformed_data  # Update column data with transformed data
        print(f'Best Lambda for {self.name}: {best_lambda}\n')
        self.shapiro_and_anderson()  # Calling shapiro_and_anderson method after transformation


def levene_test(name1, name2, col1, col2):
    levene_stat, p_value = levene(col1, col2)

    print(f'Performing Levene Test for {name1} & {name2}\n')

    print(f'{"Does not have Homogeneous (different) Variance" if p_value < 0.05 else "Have Homogeneous (similar) Variance"} (p-val
    print('-'*50)


def mannwhitneyu_test(name1, name2, col1, col2):
    print(f'Performing Non-parametric Test - Mann-Whitney U for {name1} & {name2}')

    test_stat, p_value = mannwhitneyu(col1, col2)

    result = (
        f"Reject Null Hypothesis\nThere is a significant difference in the Mean values of {name1} and {name2}"
        if p_value < 0.05
        else f"Failed to Reject Null Hypothesis - Accept Ho\nThere is NO significant difference in the Mean values of {name1} and
    )

    print(result)
    print('-' * 50)

    return ""


def normality_plots(name1, name2, name3, name4, col1, col2, col3, col4):
    plt.figure(figsize=(20, 10))
    plt.suptitle("Normality Check - Histplot & QQ Plot", fontsize=16, fontweight="bold", backgroundcolor='black', color='white')

    colors = ['royalblue', 'tomato', 'gold', 'forestgreen']
    cols = [(name1, col1, colors[0]), (name2, col2, colors[1]), (name3, col3, colors[2]), (name4, col4, colors[3])]

    for i, (name, col, color) in enumerate(cols, 1):
        plt.subplot(2, 4, i)
```

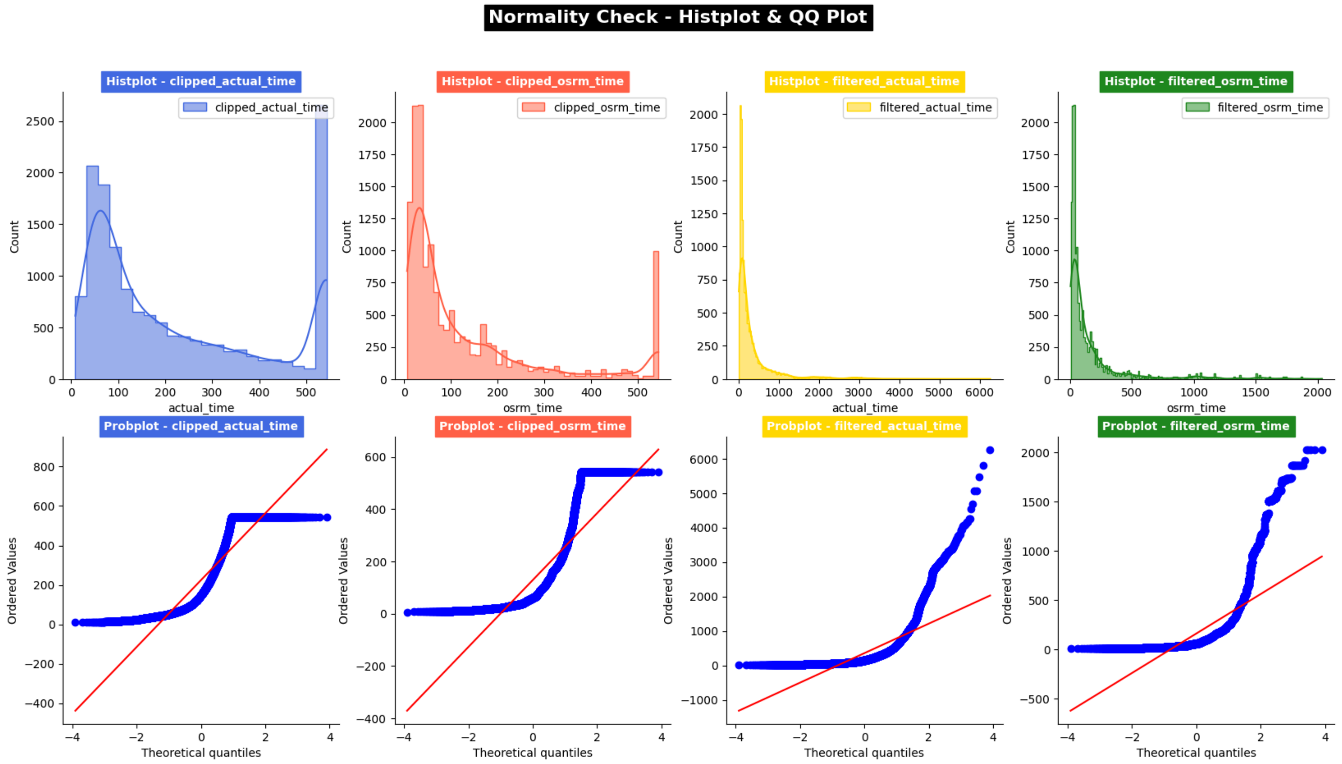
```
        sns.histplot(col, element='step', color=color, kde=True, label=name)
        plt.title(f'Histplot - {name}', fontsize=10, fontweight="bold", backgroundcolor=color, color='white')
        plt.legend()

        plt.subplot(2, 4, i + 4)
        probplot(col, plot=plt, dist='norm')
        plt.title(f'Probplot - {name}', fontsize=10, fontweight="bold", backgroundcolor=color, color='white')

    sns.despine()
    plt.show()
```

5.1.1 ⌛ Actual Time vs. OSRM Time

Compare actual time aggregated value with OSRM time aggregated value.

```
actual_time = clipped_num_df['actual_time']
osrm_time = clipped_num_df['osrm_time']
fil_actual_time = filtered_num_df['actual_time']
fil_osrm_time = filtered_num_df['osrm_time']
```

```
normality_plots('clipped_actual_time','clipped_osrm_time','filtered_actual_time','filtered_osrm_time',actual_time,osrm_time,fil_ac
```




```
col_names= ['clipped_actual_time','clipped_osrm_time','filtered_actual_time','filtered_osrm_time']
cols = [actual_time,osrm_time,fil_actual_time,fil_osrm_time]

for _ in zip(col_names,cols):
    normality = NormalityCheck(_[0],_[1])
    normality.shapiro_and_anderson()
    normality.boxcox_transformation()
```

```
--------------------------------------------------
Performing BOXCOX transformation on clipped_osrm_time column
Best Lambda for clipped_osrm_time: -0.14103790389491522

Performing SHAPIRO & ANDERSON-DARLING TEST for 'clipped_osrm_time' column

Shapiro-Wilk Test
clipped_osrm_time - Data is not Gaussian (p-value: 3.8618622532701243e-47)

Anderson-Darling Test
clipped_osrm_time - Data does not follow a normal distribution (statistic: 105.56334428441187)

--------------------------------------------------
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_actual_time' column

Shapiro-Wilk Test
filtered_actual_time - Data is not Gaussian (p-value: 1.562468303558549e-103)

Anderson-Darling Test
filtered_actual_time - Data does not follow a normal distribution (statistic: 1987.6243628991506)

--------------------------------------------------
Performing BOXCOX transformation on filtered_actual_time column
Best Lambda for filtered_actual_time: -0.1568213597704557

Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_actual_time' column

Shapiro-Wilk Test
filtered_actual_time - Data is not Gaussian (p-value: 1.3667237910318117e-28)

Anderson-Darling Test
filtered_actual_time - Data does not follow a normal distribution (statistic: 34.73776631588407)

--------------------------------------------------
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_osrm_time' column

Shapiro-Wilk Test
filtered_osrm_time - Data is not Gaussian (p-value: 1.4819529577293302e-105)

Anderson-Darling Test
filtered_osrm_time - Data does not follow a normal distribution (statistic: 2211.5469550580383)

--------------------------------------------------
Performing BOXCOX transformation on filtered_osrm_time column
Best Lambda for filtered_osrm_time: -0.21487168152152514

Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_osrm_time' column

Shapiro-Wilk Test
filtered_osrm_time - Data is not Gaussian (p-value: 5.651129981176224e-35)

Anderson-Darling Test
filtered_osrm_time - Data does not follow a normal distribution (statistic: 54.71476624963907)

--------------------------------------------------
```

```
levene_test('clipped_actual_time','clipped_osrm_time',actual_time,osrm_time),
levene_test('filtered_actual_time','filtered_osrm_time',fil_actual_time,fil_osrm_time)
```

```
Performing Levene Test for clipped_actual_time & clipped_osrm_time

Does not have Homogeneous (different) Variance (p-value: 4.662057376491051e-269)

--------------------------------------------------
Performing Levene Test for filtered_actual_time & filtered_osrm_time

Does not have Homogeneous (different) Variance (p-value: 8.744454037320379e-219)

--------------------------------------------------
```

Wilcoxon signed rank test:

With clipped data

H0: aggregated actual time is same as aggregated osrm time

Ha: aggregated actual time is more than the aggregated osrm time

```
alpha = 0.05  #testing at 95% confidence

test_stat , p_value = wilcoxon(fil_actual_time,fil_osrm_time,alternative='greater')
```

```
if p_value < alpha:
    print("Reject Null Hypothesis - The Aggregated Actual_time is More than the Aggregated OSRM_time")
else:
    print("Fail to Reject Null Hypothesis - The Aggregated Actual_time is same as the Aggregated OSRM_time")
```

```
Reject Null Hypothesis - The Aggregated Actual_time is More than the Aggregated OSRM_time
```

With filtered data

```
alpha = 0.05  #testing at 95% confidence

test_stat , p_value = wilcoxon(fil_actual_time,fil_osrm_time,alternative='greater')

if p_value < alpha:
    print("Reject Null Hypothesis - The Aggregated Actual_time is More than the Aggregated OSRM_time")
else:
    print("Fail to Reject Null Hypothesis - The Aggregated Actual_time is same as the Aggregated OSRM_time")
```

```
Reject Null Hypothesis - The Aggregated Actual_time is More than the Aggregated OSRM_time
```

```
### MannWhitney u Rank test

test_cols = [('clipped_actual_time','clipped_osrm_time',actual_time,osrm_time),
       ('filtered_actual_time','filtered_osrm_time',fil_actual_time,fil_osrm_time)]

for _ in test_cols:
    mannwhitneyu_test(_[0],_[1],_[2],_[3])
```

```
Performing Non-parametric Test - Mann-Whitney U for clipped_actual_time & clipped_osrm_time
Reject Null Hypothesis
There is a significant difference in the Mean values of clipped_actual_time and clipped_osrm_time
--------------------------------------------------
Performing Non-parametric Test - Mann-Whitney U for filtered_actual_time & filtered_osrm_time
Reject Null Hypothesis
There is a significant difference in the Mean values of filtered_actual_time and filtered_osrm_time
--------------------------------------------------
```

🔍 Insights

- The Mann-Whitney U test shows a significant difference in the mean values of Aggregated actual_time and Aggregated osrm_time.

> - H_o: $\mu_{Aggregated-actual-time}$!= $\mu_{Aggregated-osrm-time}$

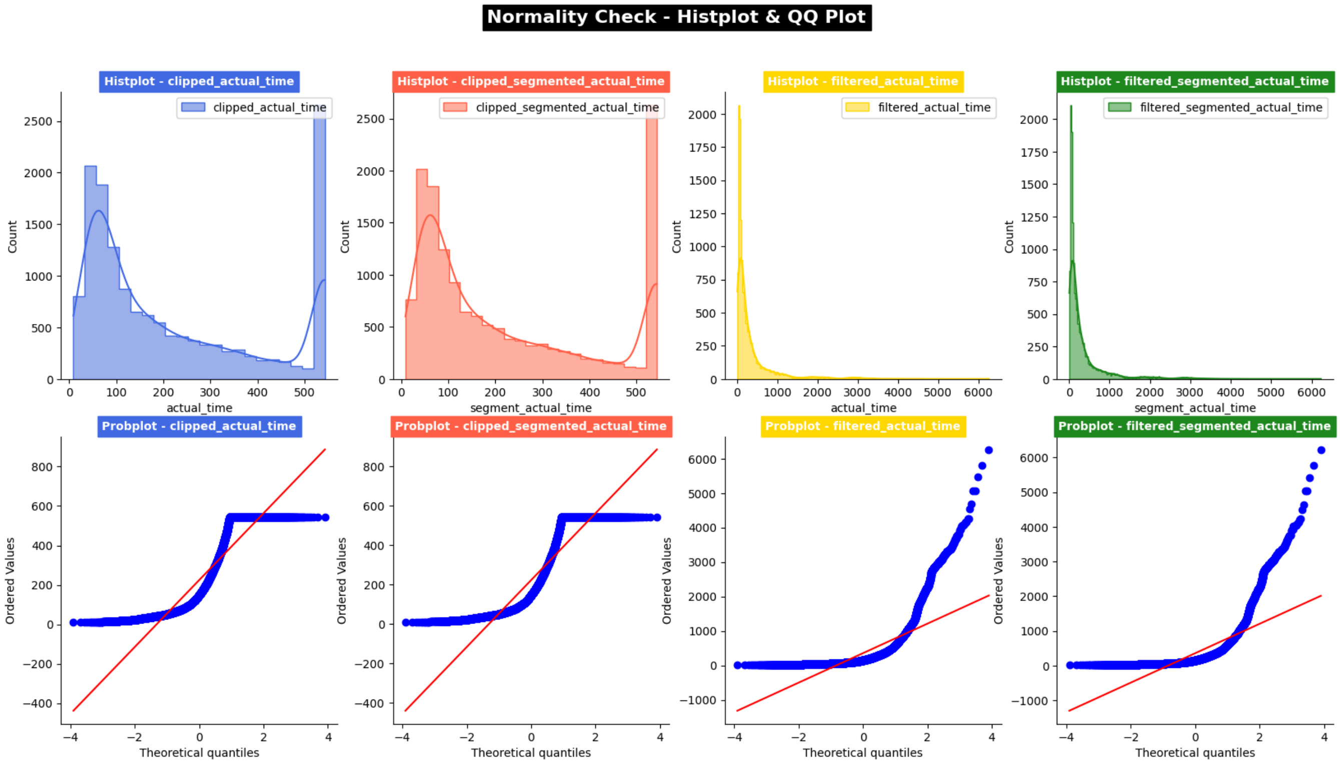
- The Wilcoxon Signed Rank test indicates that Aggregated actual_time is greater than Aggregated osrm_time.

> - H_o: $\mu_{Aggregated-actual-time}$ > $\mu_{Aggregated-osrm-time}$

5.1.2 🛠 Actual Time vs. Segment Actual Time

```
clipped_actual_time = clipped_num_df['actual_time']
clipped_segmented_actual_time = clipped_num_df['segment_actual_time']
filtered_actual_time = filtered_num_df['actual_time']
filtered_segmented_actual_time = filtered_num_df['segment_actual_time']
```

```
normality_plots("clipped_actual_time","clipped_segmented_actual_time","filtered_actual_time","filtered_segmented_actual_time",
                clipped_actual_time,clipped_segmented_actual_time,filtered_actual_time,filtered_segmented_actual_time)
```




```
col_names= ["clipped_actual_time","clipped_segmented_actual_time","filtered_actual_time","filtered_segmented_actual_time"]
cols = [clipped_actual_time,clipped_segmented_actual_time,filtered_actual_time,filtered_segmented_actual_time]

for _ in zip(col_names,cols):
    normality = NormalityCheck(_[0],_[1])
    normality.shapiro_and_anderson()
    normality.boxcox_transformation()
```

```
Anderson-Darling Test
clipped_segmented_actual_time - Data does not follow a normal distribution (statistic: 923.4775383791275)
```

```
Shapiro-Wilk Test
filtered_actual_time - Data is not Gaussian (p-value: 1.3667237910318117e-28)

Anderson-Darling Test
filtered_actual_time - Data does not follow a normal distribution (statistic: 34.73776631588407)

--------------------------------------------------
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_actual_time' column

Shapiro-Wilk Test
filtered_segmented_actual_time - Data is not Gaussian (p-value: 1.6746386852473887e-103)

Anderson-Darling Test
filtered_segmented_actual_time - Data does not follow a normal distribution (statistic: 1985.7691025408603)

--------------------------------------------------
Performing BOXCOX transformation on filtered_segmented_actual_time column
Best Lambda for filtered_segmented_actual_time: -0.156879013041558

Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_actual_time' column

Shapiro-Wilk Test
filtered_segmented_actual_time - Data is not Gaussian (p-value: 7.629078754986035e-29)

Anderson-Darling Test
filtered_segmented_actual_time - Data does not follow a normal distribution (statistic: 35.54653564388536)

--------------------------------------------------
```

```
levene_test("clipped_actual_time","clipped_segmented_actual_time",clipped_actual_time,clipped_segmented_actual_time),
levene_test("filtered_actual_time","filtered_segmented_actual_time",filtered_actual_time,filtered_segmented_actual_time)
```

```
Performing Levene Test for clipped_actual_time & clipped_segmented_actual_time

Have Homogeneous (similar) Variance (p-value: 0.7719645875874674)

--------------------------------------------------
Performing Levene Test for filtered_actual_time & filtered_segmented_actual_time

Have Homogeneous (similar) Variance (p-value: 0.6962696403096398)

--------------------------------------------------
```

```
### MannWhitney u Rank test

test_cols = [("clipped_actual_time","clipped_segmented_actual_time",clipped_actual_time,clipped_segmented_actual_time),
       ("filtered_actual_time","filtered_segmented_actual_time",filtered_actual_time,filtered_segmented_actual_time)]

for _ in test_cols:
    mannwhitneyu_test(_[0],_[1],_[2],_[3])
```

```
Performing Non-parametric Test - Mann-Whitney U for clipped_actual_time & clipped_segmented_actual_time
Failed to Reject Null Hypothesis - Accept Ho
There is NO significant difference in the Mean values of clipped_actual_time and clipped_segmented_actual_time
--------------------------------------------------
Performing Non-parametric Test - Mann-Whitney U for filtered_actual_time & filtered_segmented_actual_time
Failed to Reject Null Hypothesis - Accept Ho
There is NO significant difference in the Mean values of filtered_actual_time and filtered_segmented_actual_time
--------------------------------------------------
```

🔍 Insights

- Data is not Gaussian, but it has similar variance as confirmed by Levene's test.
- The Mann-Whitney U test shows no significant difference in the mean values of Aggregated actual_time and segmented_actual_time.

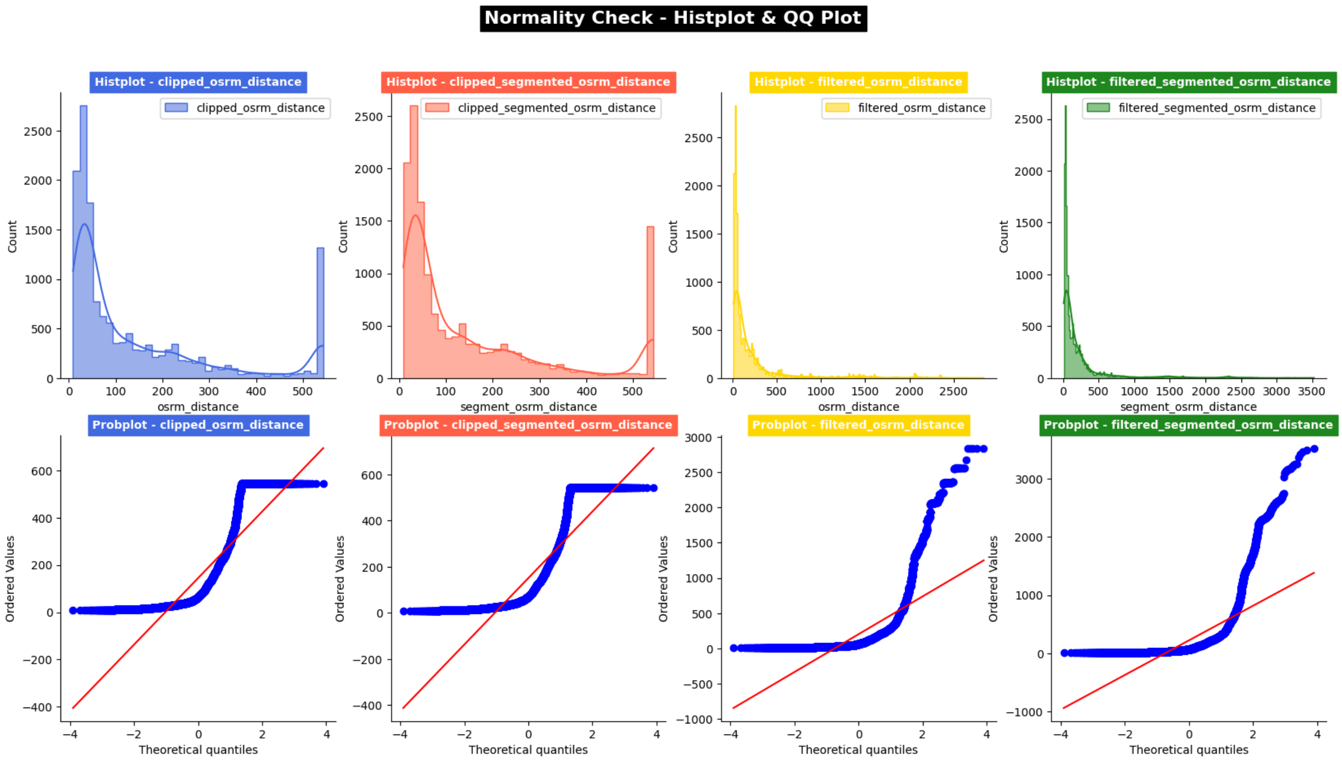
> - H_o: $\mu_{Aggregated-actual-time} = \mu_{Segmented-actual-time}$

5.1.3 📏 OSRM Distance vs. Segment OSRM Distance

```
clipped_osrm_distance = clipped_num_df['osrm_distance']
clipped_segmented_osrm_distance = clipped_num_df['segment_osrm_distance']
filtered_osrm_distance = filtered_num_df['osrm_distance']
filtered_segmented_osrm_distance = filtered_num_df['segment_osrm_distance']
```

```
normality_plots("clipped_osrm_distance","clipped_segmented_osrm_distance","filtered_osrm_distance","filtered_segmented_osrm_distan
```

```
clipped_osrm_distance,clipped_segmented_osrm_distance,filtered_osrm_distance,filtered_segmented_osrm_distance)
```




```
col_names= ["clipped_osrm_distance","clipped_segmented_osrm_distance","filtered_osrm_distance","filtered_segmented_osrm_distance"]
cols = [clipped_osrm_distance,clipped_segmented_osrm_distance,filtered_osrm_distance,filtered_segmented_osrm_distance]

for _ in zip(col_names,cols):
    normality = NormalityCheck(_[0],_[1])
    normality.shapiro_and_anderson()
    normality.boxcox_transformation()
```

```
Anderson-Darling Test
```

```
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_osrm_distance' column

Shapiro-Wilk Test
filtered_osrm_distance - Data is not Gaussian (p-value: 1.0353185470347393e-40)

Anderson-Darling Test
filtered_osrm_distance - Data does not follow a normal distribution (statistic: 87.51631391408955)

--------------------------------------------------
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_osrm_distance' column

Shapiro-Wilk Test
filtered_segmented_osrm_distance - Data is not Gaussian (p-value: 1.1730299913249368e-107)

Anderson-Darling Test
filtered_segmented_osrm_distance - Data does not follow a normal distribution (statistic: 2461.5163434679343)

--------------------------------------------------
Performing BOXCOX transformation on filtered_segmented_osrm_distance column
Best Lambda for filtered_segmented_osrm_distance: -0.2150764819274111

Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_osrm_distance' column

Shapiro-Wilk Test
filtered_segmented_osrm_distance - Data is not Gaussian (p-value: 4.522396861191329e-38)

Anderson-Darling Test
filtered_segmented_osrm_distance - Data does not follow a normal distribution (statistic: 69.05820693672649)

--------------------------------------------------
```

```
levene_test("clipped_osrm_distance","clipped_segmented_osrm_distance",clipped_osrm_distance,clipped_segmented_osrm_distance),
levene_test("filtered_osrm_distance","filtered_segmented_osrm_distance",filtered_osrm_distance,filtered_segmented_osrm_distance)
```

```
Performing Levene Test for clipped_osrm_distance & clipped_segmented_osrm_distance

Does not have Homogeneous (different) Variance (p-value: 0.01826790034030855)

--------------------------------------------------
Performing Levene Test for filtered_osrm_distance & filtered_segmented_osrm_distance

Does not have Homogeneous (different) Variance (p-value: 0.00022171118104902091)

--------------------------------------------------
```

```
# MannWhitney u Rank test

test_cols = [("clipped_osrm_distance","clipped_segmented_osrm_distance",clipped_osrm_distance,clipped_segmented_osrm_distance),
       ("filtered_osrm_distance","filtered_segmented_osrm_distance",filtered_osrm_distance,filtered_segmented_osrm_distance)]

for _ in test_cols:
    mannwhitneyu_test(_[0],_[1],_[2],_[3])
```

```
Performing Non-parametric Test - Mann-Whitney U for clipped_osrm_distance & clipped_segmented_osrm_distance
Reject Null Hypothesis
There is a significant difference in the Mean values of clipped_osrm_distance and clipped_segmented_osrm_distance
--------------------------------------------------
Performing Non-parametric Test - Mann-Whitney U for filtered_osrm_distance & filtered_segmented_osrm_distance
Reject Null Hypothesis
There is a significant difference in the Mean values of filtered_osrm_distance and filtered_segmented_osrm_distance
--------------------------------------------------
```

Insights:

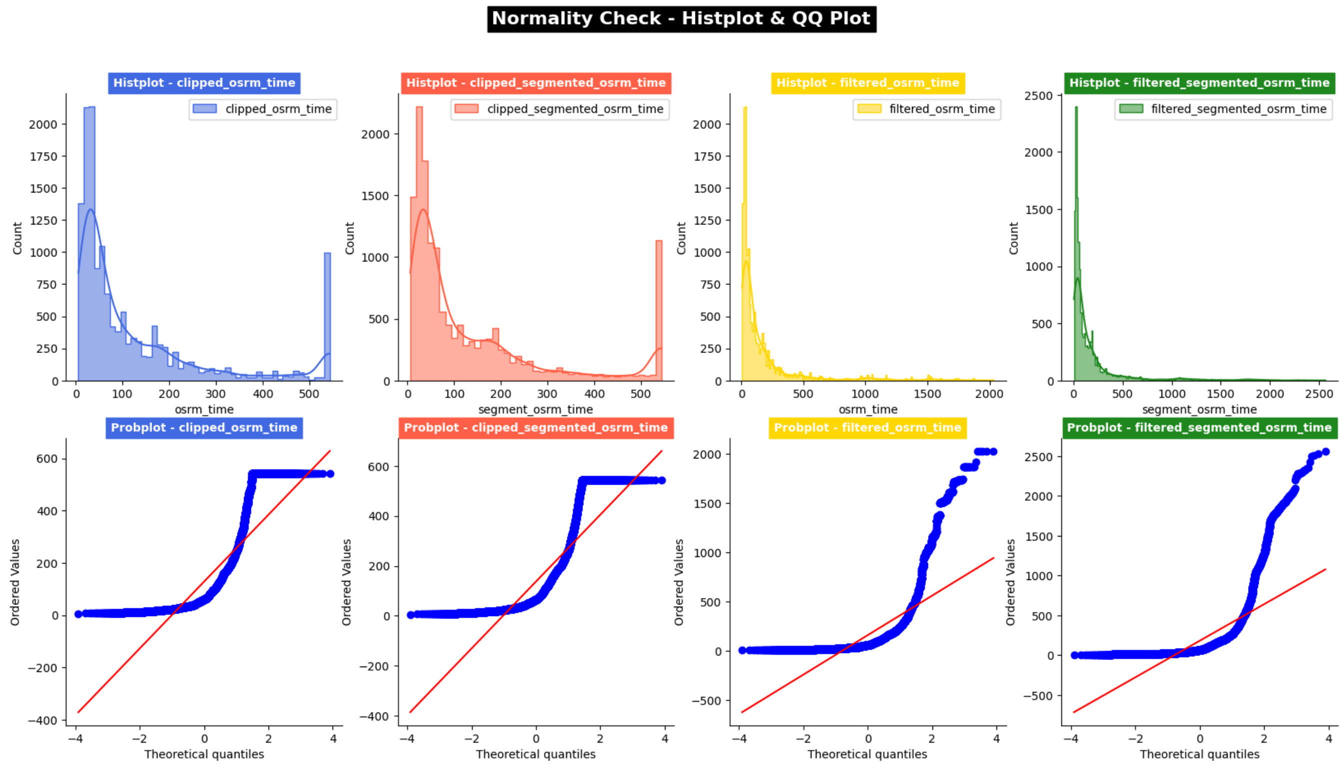
- The Mann-Whitney U test confirms a significant difference in the mean values of osrm_distance and segmented_osrm_distance.

> - H_o: $\mu_{Aggregated-osrm-distance}$!= $\mu_{Segmented-osrm-distance-aggregated}$

5.1.4 🕒 OSRM Time vs. Segment OSRM Time

```
clipped_osrm_time = clipped_num_df['osrm_time']
clipped_segmented_osrm_time = clipped_num_df['segment_osrm_time']
filtered_osrm_time = filtered_num_df['osrm_time']
filtered_segmented_osrm_time = filtered_num_df['segment_osrm_time']
```

```
normality_plots("clipped_osrm_time","clipped_segmented_osrm_time","filtered_osrm_time","filtered_segmented_osrm_time",clipped_osrm
```




```
col_names= ["clipped_osrm_time","clipped_segmented_osrm_time","filtered_osrm_time","filtered_segmented_osrm_time"]
cols = [clipped_osrm_time,clipped_segmented_osrm_time,filtered_osrm_time,filtered_segmented_osrm_time]

for _ in zip(col_names,cols):
    normality = NormalityCheck(_[0],_[1])
    normality.shapiro_and_anderson()
    normality.boxcox_transformation()
```

```
Anderson-Darling Test
clipped_segmented_osrm_time - Data does not follow a normal distribution (statistic: 1385.5093554159357)
```

```
Shapiro-Wilk Test
filtered_osrm_time - Data is not Gaussian (p-value: 5.651129981176224e-35)

Anderson-Darling Test
filtered_osrm_time - Data does not follow a normal distribution (statistic: 54.71476624963907)

--------------------------------------------------
Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_osrm_time' column

Shapiro-Wilk Test
filtered_segmented_osrm_time - Data is not Gaussian (p-value: 2.4712371662349414e-106)

Anderson-Darling Test
filtered_segmented_osrm_time - Data does not follow a normal distribution (statistic: 2274.5763419560826)

--------------------------------------------------
Performing BOXCOX transformation on filtered_segmented_osrm_time column
Best Lambda for filtered_segmented_osrm_time: -0.19359280836918025

Performing SHAPIRO & ANDERSON-DARLING TEST for 'filtered_segmented_osrm_time' column

Shapiro-Wilk Test
filtered_segmented_osrm_time - Data is not Gaussian (p-value: 8.258181658400562e-34)

Anderson-Darling Test
filtered_segmented_osrm_time - Data does not follow a normal distribution (statistic: 49.12475805580834)

--------------------------------------------------
```

```
levene_test("clipped_osrm_time","clipped_segmented_osrm_time",clipped_osrm_time,clipped_segmented_osrm_time),
levene_test("filtered_osrm_time","filtered_segmented_osrm_time",filtered_osrm_time,filtered_segmented_osrm_time)
```

```
Performing Levene Test for clipped_osrm_time & clipped_segmented_osrm_time

Does not have Homogeneous (different) Variance (p-value: 2.7446015225296333e-05)

--------------------------------------------------
Performing Levene Test for filtered_osrm_time & filtered_segmented_osrm_time

Does not have Homogeneous (different) Variance (p-value: 9.250560925676155e-08)

--------------------------------------------------
```

```
# MannWhitney u Rank test

test_cols = [("clipped_osrm_time","clipped_segmented_osrm_time",clipped_osrm_time,clipped_segmented_osrm_time),
       ("filtered_osrm_time","filtered_segmented_osrm_time",filtered_osrm_time,filtered_segmented_osrm_time)]

for _ in test_cols:
    mannwhitneyu_test(_[0],_[1],_[2],_[3])
```

```
Performing Non-parametric Test - Mann-Whitney U for clipped_osrm_time & clipped_segmented_osrm_time
Reject Null Hypothesis
There is a significant difference in the Mean values of clipped_osrm_time and clipped_segmented_osrm_time
--------------------------------------------------
Performing Non-parametric Test - Mann-Whitney U for filtered_osrm_time & filtered_segmented_osrm_time
Reject Null Hypothesis
There is a significant difference in the Mean values of filtered_osrm_time and filtered_segmented_osrm_time
--------------------------------------------------
```

🔍 **Insight:**

- The Mann-Whitney U test confirms a significant difference in the mean values of **osrm_distance** and **segmented_osrm_distance** aggregated.

> - $H_o: \mu_{Aggregated-osrm-distance}$!= $\mu_{Segmented-osrm-distance-aggregated}$

🔍 **Insight:**

1. **Comparison of Aggregated and Segmented Values:** Hypothesis testing was performed to understand the relationship between the aggregated values of trip-level metrics (actual time, OSRM time, OSRM distance) and their corresponding segmented values. These comparisons offer insights into the consistency and accuracy of segment-level data compared to the overall trip metrics.
2. **Non-Parametric Tests:** Given the non-normal distributions of several features, non-parametric tests like the Mann-Whitney U test and the Wilcoxon signed-rank test were employed. These tests allow for robust comparisons of central tendency without relying on assumptions of normality.

3. **Actual vs. OSRM Time:** The results indicate a statistically significant difference between aggregated actual time and OSRM time. This suggests that the OSRM time estimations may not be entirely accurate in capturing the actual travel time. Further analysis may be warranted to investigate the causes of this discrepancy.
4. **OSRM Distance vs. Segmented OSRM Distance:** The comparison of aggregated OSRM distance and segmented OSRM distance also revealed a significant difference. This suggests that the total OSRM distance for a trip might not be accurately represented by the sum of the individual segment distances. It could be a result of factors like routing inaccuracies or data discrepancies.
5. **Actual Time vs. Segmented Actual Time:** In contrast to the other comparisons, the Mann-Whitney U test did not reveal a significant difference between aggregated actual time and segmented actual time. This suggests that the segmented actual time data might be a reliable measure of the trip's actual time.

These insights from hypothesis testing illuminate the relationship between aggregated and segmented trip-level metrics. They can help in assessing the reliability of the data and guide further investigation on the potential sources of discrepancies between estimated and actual values.

For Business Insights

```
# Group by source_state and count the number of orders
orders_by_source = df_trip.groupby('source_state').size().reset_index(name='order_count')

# Find the source state with the most orders
most_orders_source = orders_by_source.loc[orders_by_source['order_count'].idxmax()]

# Display the source state with the most orders
print("Source state with the most orders:")
display(most_orders_source)


# Group by source_state and destination_state to find the busiest corridor
busiest_corridor = df_trip.groupby(['source_state', 'destination_state']).size().reset_index(name='count').nlargest(1, 'count')


# Average distance and time for the busiest corridor
busiest_corridor_details = df_trip.merge(busiest_corridor[['source_state', 'destination_state']], on=['source_state', 'destination
average_distance = busiest_corridor_details['actual_distance_to_destination'].mean()
average_time = busiest_corridor_details['od_time_diff_hour'].mean()

# Display results
print("Busiest corridor:")
display(busiest_corridor)
print("Average distance:", average_distance)
print("Average time (in hours):", average_time)
```

```
Source state with the most orders:
```

	17
source_state	Maharashtra
order_count	2714

dtype: object

```
Busiest corridor:
```

	source_state	destination_state	count
85	Maharashtra	Maharashtra	2458

```
Average distance: 74.852844
Average time (in hours): 5.346577921457034
```

💡 Business Insights 💡

Insights from Exploratory Data Analysis (EDA): 📊

- 🕒 **Data Timeframe:** The data spans **26 days**, from **September 12, 2018** to **October 8, 2018**.
- 📈 **Trip Distribution:** A notable **88% of trips took place in October**, with the remaining **12% in November**.
- 📉 **Data Skewness:** The overall dataset is heavily **right-skewed**.
- 📊 **Feature Correlations:** Nearly all features are **strongly correlated** with each other, which aligns with logical expectations.
- 📅 **Trip Frequency:** Fewer trips were recorded at the **start and end of the month**, with a slight increase in the **mid-month** period.
- ❓ **Missing Trips:** No trips were registered between the **4th and 11th** days of the month, which stands out as unusual.

- 🚚 **Mid-Month Surge:** A greater number of trips are observed **mid-month**, indicating that customers tend to place more orders during this period.

Route Type Analysis: 🚛

- 🚛 **FTL Preference: Full Truck Load (FTL)** shipments are more common compared to carting, pointing towards **faster and more efficient deliveries** when FTL is utilized.

Geographical Focus: 🌍

Identifying busy routes and managing transportation distances can help optimize **logistics operations** and reduce costs.

- 🌍 **Key Source States: Haryana, Maharashtra, and Karnataka** are the most active source states, reflecting significant business activity in these regions.
- 🏙️ **Busy Source Cities: Gurgaon, Bangalore, and Bhiwandi** play critical roles in the business operations, emerging as the most active source cities.
- 📍 **Top Destination Cities: Gurgaon, Bangalore, and Hyderabad** are the busiest destination cities, highlighting their importance in logistics and transportation.
- 🚚 **Busiest Route Corridor:** The route between **Mumbai, Maharashtra** and **Bangalore, Karnataka** is the busiest, with an **average distance of 74.85 km** and an **average travel time of 5.35 hours**.

Delivery Time & Distance Accuracy: ⏱️ 📏

OSRM Time vs. Actual Time:

- 🕒 **Time Difference:** The **mean actual delivery time** exceeds the **mean OSRM estimated time**, suggesting delays or variations in the actual process.
- 🔄 **Optimistic Estimates:** The **OSRM time estimates** are generally shorter than the actual delivery times, indicating that the system tends to provide **optimistic predictions**.

OSRM Distance vs. Actual Distance:

- 📏 **Distance Difference:** The **mean OSRM distance** is slightly higher than the actual distance, implying that OSRM **overestimates the distance** traveled.

Segment-wise Analysis:

- ⏳ **Time Consistency:** The alignment between the **mean actual time** and **segment actual time** shows **accurate time tracking** across delivery segments.
- 📏 **Distance Variability:** The **segment OSRM distance** is higher than the overall distance, suggesting **more conservative estimates** for individual route segments.

Actionable Insights: 💡

- ❓ **Investigate Missing Trips (4th-11th):** It would be worthwhile to explore why there are no trips between the **4th and 11th** of the month. A **focused effort** to drive orders during this period could help close the gap.
- 🚛 **Promote FTL Usage:** With FTL showing **greater efficiency**, consider strategies to **encourage more shipments via FTL** routes.

🌟 Business Recommendations 🌟

Route Optimization: 🚚 🛣️

- Given that the busiest state route is within **Karnataka**, it's essential to **optimize the transportation network** within the state. Implementing **route optimization algorithms** and using **real-time traffic monitoring** can enhance efficiency and reduce congestion.
- Since **Gurgaon** and **Bangalore** are the busiest source and destination cities, city-specific strategies to manage the **high traffic volume** should be prioritized, ensuring smooth transportation and logistics operations.

Operational Efficiency: ⚙️ 📊

- With **OSRM estimated time** being shorter than the **actual delivery time**, businesses should adjust their **customer delivery time expectations** to be more realistic and avoid disappointment.
- The **OSRM estimated distance** is greater than the **actual distance** traveled. This insight can help businesses adjust their **distance estimations** to enhance the accuracy of their logistics planning.
- Noting that **segment OSRM distance** exceeds overall OSRM distance, businesses can use these discrepancies to fine-tune **segment-specific route planning**, improving logistics precision.
- Leveraging **advanced demand forecasting techniques** can help businesses anticipate **peak traffic periods**, allowing better **resource allocation** and minimizing congestion-related delays.

- The analysis points to key areas for operational improvement. By refining **route planning algorithms**, addressing discrepancies in **estimated vs. actual data**, and improving processes across **different delivery stages**, businesses can significantly enhance **overall operational efficiency**.

Customer Satisfaction: 😊 📦

- Ensuring **accurate delivery time estimates** will lead to better **customer satisfaction** by aligning expectations with real-world outcomes.
- Increasing the use of **FTL shipments**, which are faster, can significantly impact **customer satisfaction**. Customers value timely deliveries, and optimizing **FTL** routes supports this expectation by reducing delivery times.

Customer Profiling: 📋 🔍

- A detailed **customer profiling** of those in **Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh** is necessary to understand why major orders come from these states. This can help in **enhancing customer experiences** and improving both their **buying** and **delivery journey**.

Cost Optimization: 💰 📉

- The insights into **estimated vs. actual times and distances** can support **cost optimization** strategies. Accurate logistics planning can lead to **better resource management** and reduce overall operational costs.
- Fine-tuning these processes will result in **better budget allocation** and more efficient use of resources across the supply chain.

Strategic Decision-making: 🧠 📈

- The preference for **FTL over carting** highlights a **strategic move** by logistics management. Continuously evaluating the benefits of this decision will support **long-term strategic planning**, ensuring that evolving business needs are met efficiently.

Collaboration with Stakeholders: 🤝 🚦

- Collaborating with key stakeholders—including **government authorities**, **transportation companies**, and **local communities**—can help businesses develop and implement **comprehensive strategies** for managing transportation in busy corridors and cities, enhancing logistics operations for all parties involved.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.