Delhivery Business Case Study

About Delhivery

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

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

Problem Statement

Delhivery seeks to enhance its data processing by cleaning, aggregating, and analyzing raw logistics data. The data includes timestamps, route details, and distances for each trip, with each trip divided into segments across multiple rows.

The challenge is to handle missing values, outliers, and unknown fields, while aggregating data based on trip UUID and extracting useful features. Key metrics like actual time vs. predicted time (OSRM) and distance need to be compared through visual analysis and hypothesis testing.

The goal is to generate actionable business insights to improve delivery efficiency, optimize routes, and support forecasting models.

Column Profiling:

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_center – 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
actual_distance_to_destination – Distance in Kms between source and destination warehouse
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 major and minor roads) and gives the time (Cumulative)
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 roads) (Cumulative)
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

Unknown Fields: is_cutoff, cutoff_factor, cutoff_timestamp, factor, segment_factor

Data Cleaning and Exploration

Basic Exploration

```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.stats as stats
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
```

```
# Ignore warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

```
# Set pandas options to display all columns and rows
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
# Load the dataset
# df = pd.read_csv('delhivery_data.csv')
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181')
df.sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destinatio
52983	training	2018-09-16 20:26:20.631346	thanos::sroute:9ff9217a-3084-48e0-aab2-b340606...	FTL	trip-153712958063109219	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND38
47216	test	2018-09-30 21:58:16.081990	thanos::sroute:4a82dc91-6e1b-4d75-8b07-dba2aec...	Carting	trip-153834469608177129	IND532001AAB	Srikakulam_Kuslpram_I (Andhra Pradesh)	IND53
130738	training	2018-09-18 15:21:12.692049	thanos::sroute:aa5fea4d-666c-4d4d-a4d0-6a1a7fd...	Carting	trip-153728407269182514	IND834002AAB	Ranchi_Hub (Jharkhand)	IND82
24826	training	2018-09-21 18:46:29.901572	thanos::sroute:83251748-2256-453f-b94f-c1f4dec...	FTL	trip-153755558990119114	IND524003AAA	Nellore_DC (Andhra Pradesh)	IND50
76309	test	2018-09-30 01:00:34.183402	thanos::sroute:f01c8bbd-655d-42ea-9abf-60d5040...	FTL	trip-153826923418313854	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND00

```
row_num, col_num = df.shape
print(f'There are {row_num} rows and {col_num} columns in the dataset')
```

```
There are 144867 rows and 24 columns in the dataset
```

```
df.columns
```

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

```
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
```

Insights:

- As we can see, none of the columns are in datetime format. We need to convert the columns with timestamps to datetime format.
- There are object, bool, float, and integer data types in the dataset. Will convert some of the categorical columns to category data type.

```
# Checking Duplicate Rows
print(f'There are {df.duplicated().sum()} duplicate rows in the dataset')
```

```
There are 0 duplicate rows in the dataset
```

Notes:

- Before doing statistical analysis, will clean up the data. Hence, will be easier to identify the patterns and trends in the data.

Cleaning of Data

```
# Drop the unknown columns
unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']
df.drop(unknown_fields, axis=1, inplace=True)
print(f'Before dropping, the dataset has {row_num} rows and {col_num} columns')
print(f'After dropping, the dataset has {df.shape[0]} rows and {df.shape[1]} columns')
df.sample(5)
```

```
Before dropping, the dataset has 144867 rows and 24 columns
After dropping, the dataset has 144867 rows and 19 columns
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destinatio
104748	training	2018-09-25 06:10:18.418623	thanos::sroute:626fde59-52ba-4d31-ace6-0641841...	FTL	trip-153785581841837896	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND56
118228	test	2018-09-29 13:00:07.223764	thanos::sroute:bc7dbb1d-9379-4674-b8d3-f9c3b96...	FTL	trip-153822600722350680	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND7
19608	training	2018-09-21 14:25:30.123968	thanos::sroute:bc7dbb1d-9379-4674-b8d3-f9c3b96...	FTL	trip-153753993012372153	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND7
130114	test	2018-10-02 08:49:32.876219	thanos::sroute:c9035b9e-78e2-4ffe-831b-d8734fa...	Carting	trip-153847017287596293	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	IND40
80713	training	2018-09-17 22:08:39.150911	thanos::sroute:27e9c98a-7465-43e6-ab0c-3d36b5b...	FTL	trip-153722211915054363	IND562132AAA	Bangalore_Nelmngla_H (Karnataka)	IND42

Insights:

- There are unknown columns in the dataset. We can drop these columns as they do not provide any useful information.
- Hence, 5 columns were dropped from the dataset.

```
# Convert the timestamp columns to datetime
date_time_cols = ['trip_creation_time', 'od_start_time', 'od_end_time']
for col in date_time_cols:
    df[col] = pd.to_datetime(df[col])

# Convert the categorical columns to category
cat_cols = ['data', 'route_type']
for col in cat_cols:
    df[col] = df[col].astype('category')
df.dtypes
```

	0
data	category
trip_creation_time	datetime64[ns]
route_schedule_uuid	object
route_type	category
trip_uuid	object
source_center	object
source_name	object
destination_center	object
destination_name	object
od_start_time	datetime64[ns]
od_end_time	datetime64[ns]
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

Insights:

- Columns are not in the correct format. We need to convert the columns to the correct format.
- Timestamp columns are converted to datetime format. Categorical columns are converted to category data type.
- This will help in retrieving the correct information from the dataset easier.

```
# Finding missing values and their percentage
missing_values = df.isnull().sum()
missing_values
missing_values = missing_values[missing_values > 0]
missing_percentage = missing_values / df.shape[0] * 100
missing_values = pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage})
missing_values
```

	Missing Values	Percentage
source_name	293	0.202254
destination_name	261	0.180165

Insights:

- There are missing values in the dataset.
- Source and Destination Name columns have missing values. It was around .2% and 0.18% of the data respectively.
- Since the missing values are very less, we can drop the rows with missing values.

```
# Finding whether source and destination name was mentioned in the other rows based on Destination ID
###################################################################################################

# Getting the unique source centers with missing source names
src_name_missed_centers = df.loc[df['source_name'].isna(), 'source_center'].unique()
print(f'The source centers with missing source names are: {src_name_missed_centers}')

# Identify the source name by checking if it is available for the source center in other rows
print(df.loc[df['source_center'].isin(src_name_missed_centers), 'source_name'].unique())

print('---------------------------------------------------------------------------------------------------')

# Getting the unique destination centers with missing source names
dest_name_missed_centers = df.loc[df['destination_name'].isna(), 'destination_center'].unique()
print(f'The destination centers with missing source names are: {dest_name_missed_centers}')
```

```
# Identify the source name by checking if it is available for the source center in other rows
print(df.loc[df['destination_center'].isin(dest_name_missed_centers), 'destination_name'].unique())
```

```
The source centers with missing source names are: ['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND509103AAC' 'IND126116AAA' 'IND331022A1B'
 'IND505326AAB' 'IND852118A1B']
[nan]
----------------------------------------------------------------------------------------------------
The destination centers with missing source names are: ['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND505326AAB' 'IND852118A1B' 'IND126116AAA'
 'IND509103AAC' 'IND221005A1A' 'IND250002AAC' 'IND331001A1C'
 'IND122015AAC']
[nan]
```

Insights:

- Source Name is not available for the source center. Hence, we can drop the rows with missing values.
- Destination Name is not available for the destination center. Hence, we can drop the rows with missing values.

```
# Since the source and destination names are missing, we can impute them with the source and destination centers
# But, the percentage of missing values is very low, so we can drop them
print(f'Before dropping, the dataset has {df.shape[0]} rows and {df.shape[1]} columns')
df.dropna(subset=['source_name', 'destination_name'], inplace=True)
print(f'After dropping, the dataset has {df.shape[0]} rows and {df.shape[1]} columns')

# Checking the missing values again
print(f'There are {df.isnull().sum().sum()} missing values in the dataset')
```

```
Before dropping, the dataset has 144867 rows and 19 columns
After dropping, the dataset has 144316 rows and 19 columns
There are 0 missing values in the dataset
```

Statistical Analysis

```
df.describe().T
```

	count	mean	min	25%	50%	75%	
trip_creation_time	144316	2018-09-22 13:05:09.454117120	2018-09-12 00:00:16.535741	2018-09-17 02:46:11.004421120	2018-09-22 03:36:19.186585088	2018-09-27 17:53:19.027942912	2018 23:59:42.
od_start_time	144316	2018-09-22 17:32:42.435769344	2018-09-12 00:00:16.535741	2018-09-17 07:37:35.014584832	2018-09-22 07:35:23.038482944	2018-09-27 22:01:30.861209088	2018 04:27:23.
od_end_time	144316	2018-09-23 09:36:54.057172224	2018-09-12 00:50:10.814399	2018-09-18 01:29:56.978912	2018-09-23 02:49:00.936600064	2018-09-28 12:13:41.675546112	2018 03:00:24.
start_scan_to_end_scan	144316.0	963.697698	20.0	161.0	451.0	1645.0	
actual_distance_to_destination	144316.0	234.708498	9.000045	23.352027	66.135322	286.919294	1927.
actual_time	144316.0	417.996237	9.0	51.0	132.0	516.0	
osrm_time	144316.0	214.437055	6.0	27.0	64.0	259.0	
osrm_distance	144316.0	285.549785	9.0082	29.89625	78.6244	346.3054	232
segment_actual_time	144316.0	36.175379	-244.0	20.0	28.0	40.0	
segment_osrm_time	144316.0	18.495697	0.0	11.0	17.0	22.0	
segment_osrm_distance	144316.0	22.818993	0.0	12.053975	23.5083	27.813325	219

```
df.describe(include=['category', 'object']).T
```

	count	unique	top	freq
data	144316	2	training	104632
route_schedule_uuid	144316	1497	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	1812
route_type	144316	2	FTL	99132
trip_uuid	144316	14787	trip-153837029526866991	101
source_center	144316	1496	IND000000ACB	23267
source_name	144316	1496	Gurgaon_Bilaspur_HB (Haryana)	23267
destination_center	144316	1466	IND000000ACB	15192
destination_name	144316	1466	Gurgaon_Bilaspur_HB (Haryana)	15192

```
# Print the unique values of the categorical columns
for col in cat_cols:
```

```
    print(f'Unique values of {col} are: {df[col].unique().tolist()}')
```

```
Unique values of data are: ['training', 'test']
Unique values of route_type are: ['Carting', 'FTL']
```

Feature Extraction

```
# Since delivery details of one package are divided into several rows, we can group and aggregate them
# Analysing one Carting sample trip
df.loc[df['trip_uuid'] == 'trip-153741093647649320']
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_cer
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320

```
df.loc[df['route_type'] == 'FTL'].sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destinat
14283	training	2018-09-25 13:25:09.517204	thanos::sroute:a4865515-71e9-4dc8-8bae-aec5a36...	FTL	trip-153788190951679687	IND562132AAA	Bangalore_Nelmngla_H (Karnataka)	IND
137949	training	2018-09-25 21:23:10.380886	thanos::sroute:caf62782-95cc-4d47-a071-d1c7038...	FTL	trip-153791059038065877	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND
104262	test	2018-10-02 20:39:04.749899	thanos::sroute:14465f3f-3911-46b4-9256-0ddc7f5...	FTL	trip-153851274474962892	IND533106AAA	Rajamundry_AtoNgrRd_I (Andhra Pradesh)	IND
138875	training	2018-09-26 18:06:53.466526	thanos::sroute:17fc17ec-ffde-4f2a-8ad6-ff420a0...	FTL	trip-153798521346626830	IND382430AAB	Ahmedabad_East_H_1 (Gujarat)	IND
92698	test	2018-10-02 17:55:40.326596	thanos::sroute:c08130ce-88ec-4afe-bee2-13999d5...	FTL	trip-153850294032634589	IND501359AAE	Hyderabad_Shamshbd_H (Telangana)	IND

```
# Analysing one FTL sample trip
df.loc[df['trip_uuid'] == 'trip-153671279791474803']
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_
29081	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144001AAB	Jalandhar_DPC (Punjab)	IND144
29082	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144601AAA	Kapurthala_DC (Punjab)	IND144
29083	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144001AAA	Jalandhar_DC (Punjab)	IND144
29084	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144001AAA	Jalandhar_DC (Punjab)	IND144
29085	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144040AAA	Nakodar_ChowkDPP_D (Punjab)	IND144
29086	training	2018-09-12 00:39:57.915096	thanos::sroute:16abf49b-3f11-48e2-962c-07fcc01...	FTL	trip-153671279791474803	IND144040AAA	Nakodar_ChowkDPP_D (Punjab)	IND144

Insights:

- Took only one trip_uuid for Carting and FLT to analyze the structure of data.
- Here, data, trip_creation_time, route_schedule_uuid, route_type, trip_uuid were the same for all the rows.
- Hence, we can either take the first row or the last row for the analysis.
- Again, source and destination details including source_name, destination_name, od_start_time, od_end_time, start_scan_to_end_scan were the same for few rows. Only the actual, osr and the segment details were different.
- Hence, we can use sum, cumulative sum, mean or some other aggregation method for the actual, osr and segment details to merge it into one row.

```
# Since, all the rows are depend on Trip ID and the source and destination center, will create one new row for grouping
df['trip|src|dest'] = df['trip_uuid'] + '|' + df['source_center'] + '|' + df['destination_center']

# Calculating cumulative sum for the segment columns
segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time']
df[[f'{col}_cum_sum' for col in segment_cols]] = df.groupby('trip_uuid')[segment_cols].cumsum()
df.sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination
969	training	2018-09-17 21:13:57.454736	thanos::sroute:f8c83fd0-6554-44f3-9408-32465bd...	FTL	trip-153721883745449045	IND573116AAA	Channaraya_patna_D (Karnataka)	IND57
111064	test	2018-10-01 21:23:25.878838	thanos::sroute:5c8c056b-47a7-4d82-bc53-107026c...	FTL	trip-153842900587859575	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	IND462
92925	training	2018-09-20 21:43:33.033162	thanos::sroute:532959b3-a854-437a-b00d-551b14b...	Carting	trip-153747981303290283	IND700065AAA	CCU_Beliaghata_DPC (West Bengal)	IND71
116115	training	2018-09-21 20:51:27.259772	thanos::sroute:fa83fd49-3327-4503-8e80-bf58ed6...	FTL	trip-153756308725951466	IND834002AAB	Ranchi_Hub (Jharkhand)	IND000
141286	test	2018-09-28 19:54:45.975911	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153816448597565211	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND562

```
df[['trip|src|dest', 'actual_time', 'segment_actual_time_cum_sum', 'osrm_distance', 'segment_osrm_distance_cum_sum', 'osrm_time', 'segment_os
```

	trip|src|dest	actual_time	segment_actual_time_cum_sum	osrm_distance	segment_osrm_distance_cum_sum
0	trip-153741093647649320|IND388121AAA|IND388620AAB	14.0	14.0	11.9653	11.9653
1	trip-153741093647649320|IND388121AAA|IND388620AAB	24.0	24.0	21.7243	21.7243
2	trip-153741093647649320|IND388121AAA|IND388620AAB	40.0	40.0	32.5395	32.5395

Insights:

- All the segment columns are giving segment level details, hence we can do the cumulative sum of the segment columns.
- As we can see from the data, the actual, osrm and the segment cumulative sum is approximately same for all the rows.
- There are some differences in the actual, osrm and the segment cumulative sum. This was because of the null value handling step where we dropped the rows with missing values.
- Hence, we can use the cumulative sum of the segment columns to merge it into one row and drop the duplicate rows.

```
# Drop the actual and osrm columns
# backup_df = df[['trip|src|dest', 'actual_time', 'segment_actual_time_cum_sum', 'osrm_distance', 'segment_osrm_distance_cum_sum', 'osrm_time
# df.drop(['actual_time', 'osrm_distance', 'osrm_time'], axis=1, inplace=True)

# # Rename the calculated columns to the original name
# df.rename(columns={'segment_actual_time_cum_sum': 'actual_time', 'segment_osrm_distance_cum_sum': 'osrm_distance', 'segment_osrm_time_cum_s
# df.sample(5)
```

```
# Aggregate the data based on the trip, source and destination into one row
segment_dict = {
    'data': 'first',
    'trip_creation_time': 'first',
    'route_type': 'first',
    'trip_uuid' : 'first',
    'source_name': 'first',
    'destination_name': 'last',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'first',
    'actual_distance_to_destination': 'last',
    'actual_time': 'last',
    'osrm_time': 'last',
    'osrm_distance': 'last'
}

agg_df = df.groupby('trip|src|dest').agg(segment_dict).reset_index().sort_values(by=['trip|src|dest', 'od_end_time'])
agg_df.sample(5)
```

	trip|src|dest	data	trip_creation_time	route_type	trip_uuid	source_name	
9523	trip-153737437961982221|IND635802AAB|IND562132AAA	training	2018-09-19 16:26:19.620044	FTL	trip-153737437961982221	Ambur_Central_D_2 (Tamil Nadu)	
17340	trip-153791084714708100|IND742148AAA|IND742225AAA	training	2018-09-25 21:27:27.147353	Carting	trip-153791084714708100	Lalgola_KrsprDPP_D (West Bengal)	Ra
12496	trip-153756603883941583|IND816107AAA|IND826004AAA	training	2018-09-21 21:40:38.839734	FTL	trip-153756603883941583	Pakur_Hatpada_D (Jharkhand)	
10830	trip-153745750546231162|IND600032AAB|IND600056AAB	training	2018-09-20 15:31:45.462567	Carting	trip-153745750546231162	Chennai_Hub (Tamil Nadu)	
14748	trip-153772917731215072|IND600056AAB|IND600016AAC	training	2018-09-23 18:59:37.312406	Carting	trip-153772917731215072	MAA_Poonamallee_HB (Tamil Nadu)	C

Insights:

- Based on the trip_uuid, source_name and destination_name, all the data got merged into one row.
- Here, actual, osrm columns are having the cumulative sum, last values are taken.
- Same for the end time, actual_distance_to_destination

```
# Aggregate the data based on the trip, source and destination into one row
segment_dict = {
    'data': 'first',
    'trip_creation_time': 'first',
    'route_type': 'first',
    'source_name': 'first',
    'source_center': 'first',
    'destination_name': 'last',
    'destination_center': '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'
}
```

```
agg_df_2 = df.groupby('trip_uuid').agg(segment_dict).reset_index().sort_values(by=['trip_uuid', 'od_end_time'])
agg_df_2.sample(5)
```

	trip_uuid	data	trip_creation_time	route_type	source_name	source_center	destination_name	destinatic
12568	trip-153827163308536808	test	2018-09-30 01:40:33.085610	FTL	Chhatarpur_Busstand_D (Madhya Pradesh)	IND471001AAA	Tikamgarh_MndiRoad_D (Madhya Pradesh)	IND4
11952	trip-153818255057771730	test	2018-09-29 00:55:50.577962	Carting	Jaipur_Central_I_7 (Rajasthan)	IND302014AAB	Jaipur_Central_I_7 (Rajasthan)	IND3
14200	trip-153852929394908855	test	2018-10-03 01:14:53.949341	Carting	Bhiwandi_Mankoli_HB (Maharashtra)	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	IND42
9716	trip-153790168210650684	training	2018-09-25 18:54:42.106758	FTL	Bhadrak_Central_I_2 (Orissa)	IND756100AAC	Bhadrak_Central_I_2 (Orissa)	IND75
13875	trip-153848928202349438	test	2018-10-02 14:08:02.023743	Carting	Bengaluru_Bomsndra_HB (Karnataka)	IND560099AAB	Bengaluru_Kengeri_IP (Karnataka)	IND56

Insights:

- The data is now aggregated based on the trip_uuid alone.
- The data is now ready for further analysis.

```
# Dictionary containing IATA codes and their corresponding standardized city names
iata_to_city = {
    'CCU': 'Kolkata',
    'MAA': 'Chennai',
    'BLR': 'Bengaluru',
    'BOM': 'Mumbai',
    'DEL': 'Delhi',
    'HYD': 'Hyderabad',
    'AMD': 'Ahmedabad',
    'CJB': 'Coimbatore',
    'GOI': 'Goa',
    'PNQ': 'Pune',
    'LKO': 'Lucknow',
    'JAI': 'Jaipur',
    'IXC': 'Chandigarh',
    'TRV': 'Thiruvananthapuram',
    'CCJ': 'Kozhikode',
    'IXB': 'Siliguri',
    'IXR': 'Ranchi',
    'BBI': 'Bhubaneswar',
    'GOP': 'Gorakhpur',
    'PAT': 'Patna',
    'VGA': 'Vijayawada',
    'IXM': 'Madurai',
    'IXE': 'Mangalore',
    'NAG': 'Nagpur',
    'GAU': 'Guwahati',
    'UDR': 'Udaipur',
    'VNS': 'Varanasi',
    'SXR': 'Srinagar',
    'BDQ': 'Vadodara',
    'GWL': 'Gwalior',
    'IXJ': 'Jammu',
    'HBX': 'Hubli',
    'VTZ': 'Visakhapatnam',
    'RPR': 'Raipur',
    'BHO': 'Bhopal',
    'IMF': 'Imphal',
    'IXA': 'Agartala',
    'JRH': 'Jorhat',
    'IXS': 'Silchar',
    'DIB': 'Dibrugarh',
    # Adding local names mapping to standard city names
    'Madras': 'Chennai',
    'Calcutta': 'Kolkata',
    'Bangalore': 'Bengaluru',
    'Bombay': 'Mumbai',
    'Thiruvananthapuram': 'Thiruvananthapuram',
    'Calicut': 'Kozhikode',
    'Benares': 'Varanasi',
    'Baroda': 'Vadodara',
    'Vizag': 'Visakhapatnam',
}

def extract_place(data):
    place = ''
    # Split the input data to extract potential IATA or local name
```

```
    if len(data.split('_')) == 1:
        place = data.split('(')[0].split('_')[0].strip()
    else:
        place = data.split("_")[1].strip()

    # Check for IATA code or local name in the dictionary
    if place in iata_to_city:
        return iata_to_city[place]  # Return the standardized city name
    return place  # Return the original place if not found

def extract_state(data):
    return data.split('(')[1].replace(')', '').strip()

def extract_city(data):
    city = data.split('(')[0].split('_')[0].strip()
    if city in iata_to_city:
        return iata_to_city[city]  # Return the standardized city name
    return city
```

Extracting New Features:

```
# Extract the trip duration in hours
agg_df['trip_time(mins)'] = (agg_df['od_end_time'] - agg_df['od_start_time']).dt.total_seconds() / 60
agg_df_2['trip_time(mins)'] = (agg_df_2['od_end_time'] - agg_df_2['od_start_time']).dt.total_seconds() / 60
agg_df.sample(2)
```

	trip|src|dest	data	trip_creation_time	route_type	trip_uuid	source_name	
18035	trip-153794608862696045|IND600032AAB|IND600076AAB	training	2018-09-26 07:14:48.627232	Carting	trip-153794608862696045	Chennai_Hub (Tamil Nadu)	Che
25501	trip-153856446755053750|IND829122AAA|IND825301AAA	test	2018-10-03 11:01:07.551004	FTL	trip-153856446755053750	Ramgarh_HotelPrk_D (Jharkhand)	H

```
# Extract the source and destination city, state and place
agg_df['source_place'] = agg_df['source_name'].apply(extract_place)
agg_df['source_city'] = agg_df['source_name'].apply(extract_city)
agg_df['source_state'] = agg_df['source_name'].apply(extract_state)

agg_df['destination_place'] = agg_df['destination_name'].apply(extract_place)
agg_df['destination_city'] = agg_df['destination_name'].apply(extract_city)
agg_df['destination_state'] = agg_df['destination_name'].apply(extract_state)

agg_df.sample(5)
```

	trip|src|dest	data	trip_creation_time	route_type	trip_uuid	source_name	
8	trip-153671052974046625|IND583119AAA|IND583101AAA	training	2018-09-12 00:02:09.740725	FTL	trip-153671052974046625	Sandur_WrdN1DPP_D (Karnataka)	
15242	trip-153775220795142723|IND384355AAB|IND385001AAA	training	2018-09-24 01:23:27.951679	FTL	trip-153775220795142723	Vadnagar_BsstdDPP_D (Gujarat)	
4197	trip-153698041312290100|IND110037AAK|IND000000ACB	training	2018-09-15 03:00:13.123241	Carting	trip-153698041312290100	Delhi_Kapshera_L (Delhi)	
2347	trip-153687595451422442|IND140501AAA|IND160002AAC	training	2018-09-13 21:59:14.514511	Carting	trip-153687595451422442	Lalru_OnkarDPP_D (Punjab)	C
14764	trip-153773043445518313|IND421802AAA|IND411033AAA	training	2018-09-23 19:20:34.455437	FTL	trip-153773043445518313	Khandala_Satara_D (Maharashtra)	

```
# Trip Creation Time Analysis
agg_df['trip_creation_date'] = pd.to_datetime(agg_df['trip_creation_time'].dt.date)
agg_df['trip_creation_day'] = agg_df['trip_creation_time'].dt.day.astype('int')
agg_df['trip_creation_month'] = agg_df['trip_creation_time'].dt.month.astype('int')
agg_df['trip_creation_week'] = agg_df['trip_creation_time'].dt.isocalendar().week.astype('int')
agg_df['trip_creation_hour'] = agg_df['trip_creation_time'].dt.hour.astype('int')

agg_df.sample(5)
```

	trip|src|dest	data	trip_creation_time	route_type	trip_uuid	source_name	
10227	trip-153740442108450757|IND208012AAA|IND209304AAA	training	2018-09-20 00:47:01.084739	Carting	trip-153740442108450757	Kanpur_Central_D_7 (Uttar Pradesh)	
3307	trip-153694610011270291|IND721102AAA|IND712311AAA	training	2018-09-14 17:28:17.634274	FTL	trip-153694610011270291	Midnapore_Talkui_D (West Bengal)	
4650	trip-153703151748613495|IND600016AAB|IND600044AAC	training	2018-09-15 17:11:57.486514	Carting	trip-153703151748613495	Chennai_Alandur_C (Tamil Nadu)	C
26213	trip-153861106442901555|IND208006AAA|IND209304AAA	test	2018-10-03 23:57:44.429324	Carting	trip-153861106442901555	Kanpur_GovndNgr_DC (Uttar Pradesh)	
12875	trip-153757765074573076|IND500032AAA|IND502295AAA	training	2018-09-22 00:54:10.746119	Carting	trip-153757765074573076	Hyd_Chandanagar_Dc (Telangana)	Sa

```
# Droping unnecessary columns after extracting the required information
agg_df.drop(['trip_creation_time','source_name', 'destination_name', 'od_start_time', 'od_end_time'], axis=1, inplace=True)
agg_df.sample(5)
```

	trip|src|dest	data	route_type	trip_uuid	start_scan_to_end_scan	actual_distance_to_
6231	trip-153713985107076924|IND110064AAA|IND000000ACB	training	Carting	trip-153713985107076924	81.0	
20764	trip-153816699243973025|IND509103AAB|IND518002AAA	test	FTL	trip-153816699243973025	132.0	
13932	trip-153765490850883005|IND843109AAB|IND843126AAA	training	Carting	trip-153765490850883005	81.0	
18857	trip-153800518211472376|IND781022AAA|IND781018AAB	training	Carting	trip-153800518211472376	74.0	
5199	trip-153705600924053371|IND560099AAB|IND560085AAA	training	Carting	trip-153705600924053371	126.0	

Cleaned Data Structural Analysis

```
agg_row, agg_col = agg_df.shape
print(f'There are {agg_row} rows and {agg_col} columns in the aggregated dataset')
```

```
There are 26222 rows and 21 columns in the aggregated dataset
```

```
agg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26222 entries, 0 to 26221
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip|src|dest                   26222 non-null  object
 1   data                            26222 non-null  category
 2   route_type                      26222 non-null  category
 3   trip_uuid                       26222 non-null  object
 4   start_scan_to_end_scan          26222 non-null  float64
 5   actual_distance_to_destination  26222 non-null  float64
 6   actual_time                     26222 non-null  float64
 7   osrm_time                       26222 non-null  float64
 8   osrm_distance                   26222 non-null  float64
 9   trip_time(mins)                 26222 non-null  float64
 10  source_place                    26222 non-null  object
 11  source_city                     26222 non-null  object
 12  source_state                    26222 non-null  object
 13  destination_place               26222 non-null  object
 14  destination_city                26222 non-null  object
 15  destination_state               26222 non-null  object
 16  trip_creation_date              26222 non-null  datetime64[ns]
 17  trip_creation_day               26222 non-null  int64
 18  trip_creation_month             26222 non-null  int64
 19  trip_creation_week              26222 non-null  int64
 20  trip_creation_hour              26222 non-null  int64
dtypes: category(2), datetime64[ns](1), float64(6), int64(4), object(8)
memory usage: 3.9+ MB
```

```
agg_df.describe().T
```

	count	mean	min	25%	50%	75%	max	std
start_scan_to_end_scan	26222.0	298.55339	20.0	90.0	152.0	307.0	7898.0	441.116816
actual_distance_to_destination	26222.0	92.533054	9.001351	21.654149	35.044329	65.557393	1927.447705	209.952355
actual_time	26222.0	200.926588	9.0	51.0	84.0	167.0	4532.0	385.730908
osrm_time	26222.0	90.785333	6.0	25.0	39.0	72.0	1686.0	185.554359
osrm_distance	26222.0	114.975332	9.0729	27.71915	43.54355	85.44395	2326.1991	254.426468
trip_time(mins)	26222.0	299.107278	20.702813	90.977759	152.336732	307.285979	7898.551955	441.249287
trip_creation_date	26222	2018-09-22 00:35:41.713065216	2018-09-12 00:00:00	2018-09-17 00:00:00	2018-09-22 00:00:00	2018-09-27 00:00:00	2018-10-03 00:00:00	NaN
trip_creation_day	26222.0	18.400351	1.0	14.0	19.0	25.0	30.0	7.892191
trip_creation_month	26222.0	9.120815	9.0	9.0	9.0	9.0	10.0	0.325918
trip_creation_week	26222.0	38.302952	37.0	38.0	38.0	39.0	40.0	0.965982

```
agg_df.describe(include=['category', 'object'])
```

	trip|src|dest	data	route_type	trip_uuid	source_place	source_city	source_state	d
count	26222	26222	26222	26222	26222	26222	26222	
unique	26222	2	2	14787	1217	1240	31	
top	153671041653548748|IND209304AAA|IND000000ACB	training	FTL	trip-153717306559016761	Central	Bengaluru	Maharashtra	
freq	1	18893	13798	8	1976	1964	3565	

```
for col in agg_df.columns:
    if agg_df[col].nunique() < 40:
        print(f'Unique values of {col} are: {agg_df[col].nunique()}')
        print(agg_df[col].unique().tolist())
        print('-' * 50)
```

```
Unique values of data are: 2
['training', 'test']
--------------------------------------------------
Unique values of route_type are: 2
['FTL', 'Carting']
--------------------------------------------------
Unique values of source_state are: 31
['Uttar Pradesh', 'Madhya Pradesh', 'Karnataka', 'Haryana', 'Maharashtra', 'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Andhra Pradesh
--------------------------------------------------
Unique values of destination_state are: 32
['Haryana', 'Uttar Pradesh', 'Karnataka', 'Punjab', 'Maharashtra', 'Tamil Nadu', 'Gujarat', 'Delhi', 'Andhra Pradesh', 'Telangana', 'Raja
--------------------------------------------------
Unique values of trip_creation_date are: 22
[Timestamp('2018-09-12 00:00:00'), Timestamp('2018-09-13 00:00:00'), Timestamp('2018-09-14 00:00:00'), Timestamp('2018-09-15 00:00:00'),
--------------------------------------------------
Unique values of trip_creation_day are: 22
[12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 2, 3]
--------------------------------------------------
Unique values of trip_creation_month are: 2
[9, 10]
--------------------------------------------------
Unique values of trip_creation_week are: 4
[37, 38, 39, 40]
--------------------------------------------------
Unique values of trip_creation_hour are: 24
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
--------------------------------------------------
```

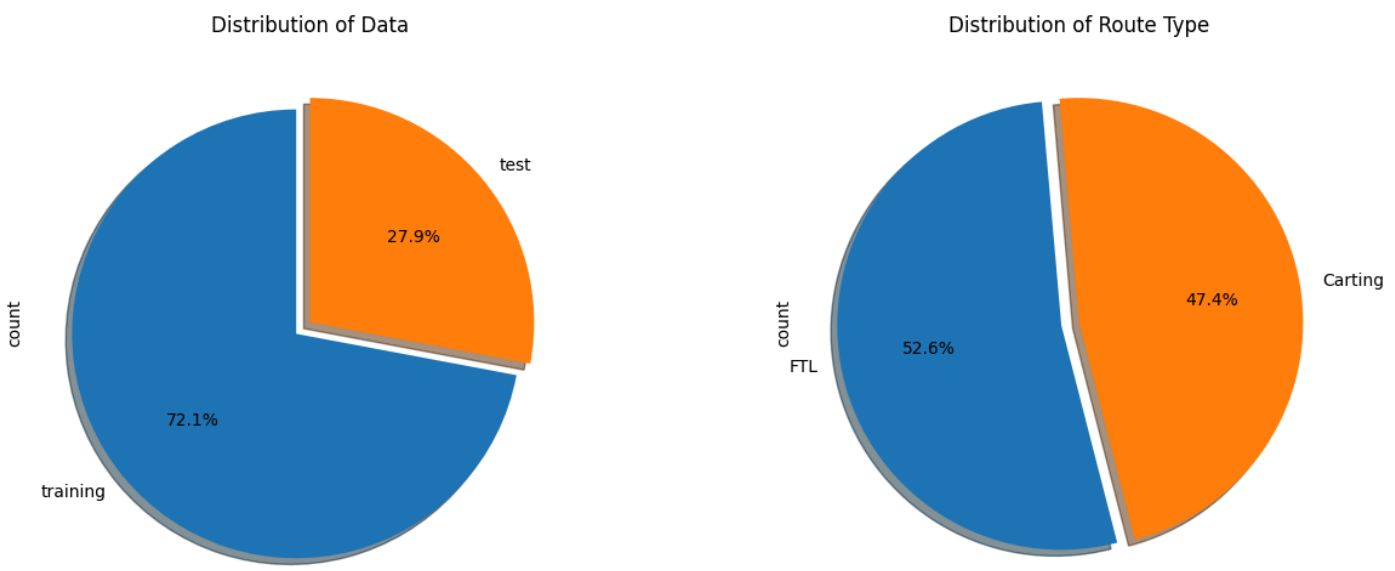
```
# Trip Creation Date range
print(f'The Trip Creation Date range is from {agg_df["trip_creation_date"].min()} to {agg_df["trip_creation_date"].max()}')
print(f'Total duration of the data is {(agg_df["trip_creation_date"].max() - agg_df["trip_creation_date"].min()).days} days')
```

```
The Trip Creation Date range is from 2018-09-12 00:00:00 to 2018-10-03 00:00:00
Total duration of the data is 21 days
```

Data Visualization

Univariate Analysis

```
# Distribution of Data and Route Type (pie chart)
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
agg_df['data'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, explode=(0.08, 0), shadow=True)
plt.title('Distribution of Data')

plt.subplot(1, 2, 2)
agg_df['route_type'].value_counts().plot.pie(autopct='%1.1f%%', startangle=95, explode=(0.08, 0), shadow=True)
plt.title('Distribution of Route Type')
plt.show()
```

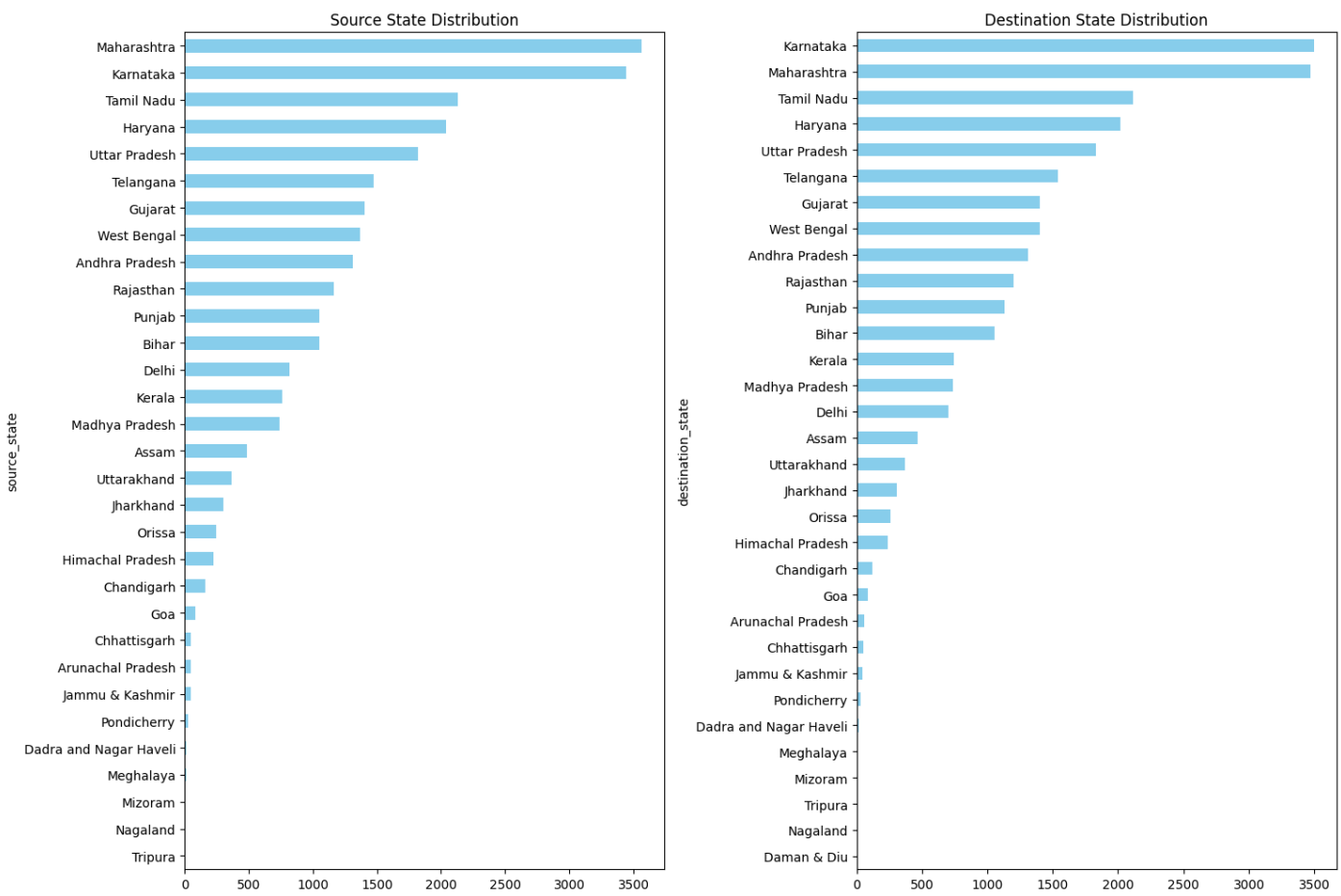


Insights:

- There are more Carting trips than FTL trips. Carting trips are around 53% and FTL trips are around 47%.
- There are more trainings data than testing data. Training data is around 72% and testing data is around 28%.

Bivariate Analysis

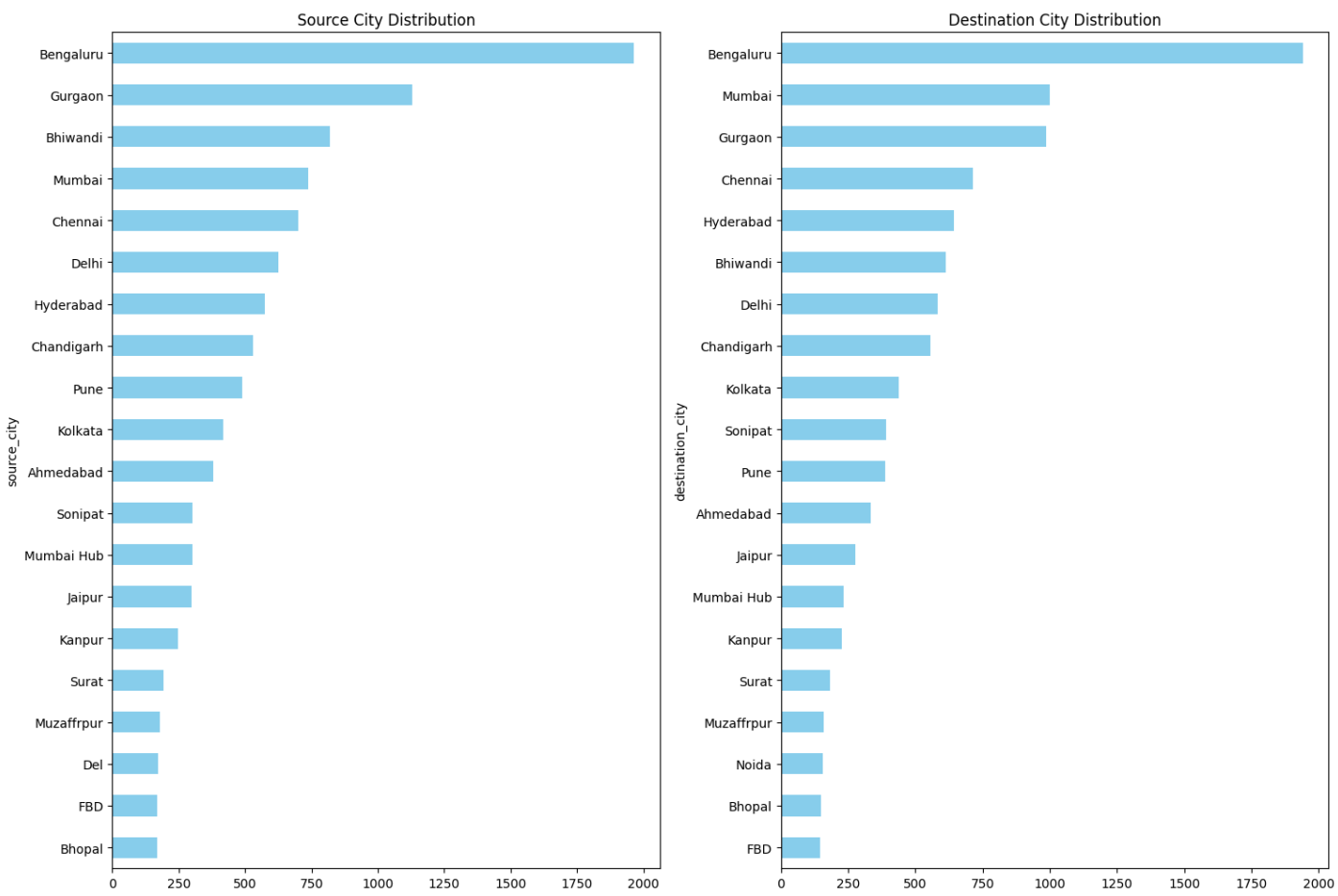
```
# Source and Destination State Analysis
plt.figure(figsize=(15, 10))

plt.subplot(1, 2, 1)
agg_df['source_state'].value_counts().plot(kind='barh', color='skyblue')
plt.title('Source State Distribution')
plt.gca().invert_yaxis()

plt.subplot(1, 2, 2)
agg_df['destination_state'].value_counts().plot(kind='barh', color='skyblue')
plt.title('Destination State Distribution')
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()
```



Insights:

- Maharashtra has the highest number of source centers and second highest number of destination centers.
- Karnataka has the highest number of destination centers and second highest number of source centers.
- Tamil Nadu has the third highest number of source and destination centers.
- There is a consistent pattern in the order of the states with the highest number of source and destination centers.
- Mostly, the source and destination states are in the same order.
- Based on the insights, each state has a similar number of source and destination centers.

```
# Source and Destination City Analysis
plt.figure(figsize=(15, 10))

plt.subplot(1, 2, 1)
agg_df['source_city'].value_counts().head(20).plot(kind='barh', color='skyblue')
plt.title('Source City Distribution')
plt.gca().invert_yaxis()

plt.subplot(1, 2, 2)
agg_df['destination_city'].value_counts().head(20).plot(kind='barh', color='skyblue')
plt.title('Destination City Distribution')
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()
```

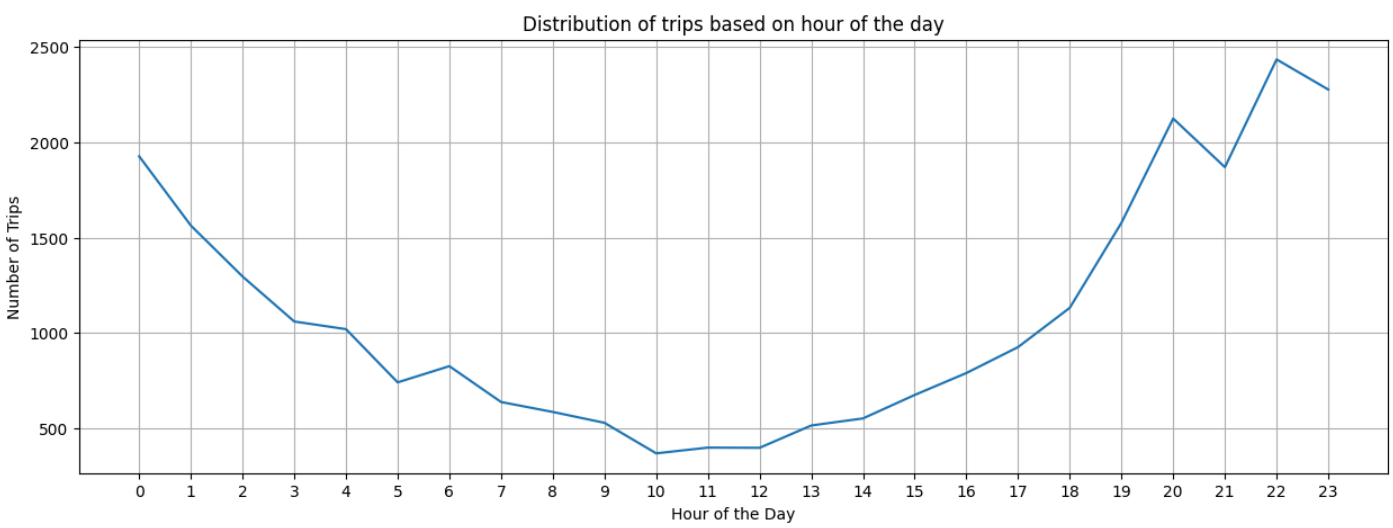


Insights:

- As we can see, the city wise source and destination centers are almost the same as state wise data.
- Hence, proving the state wise insights.

Insights:

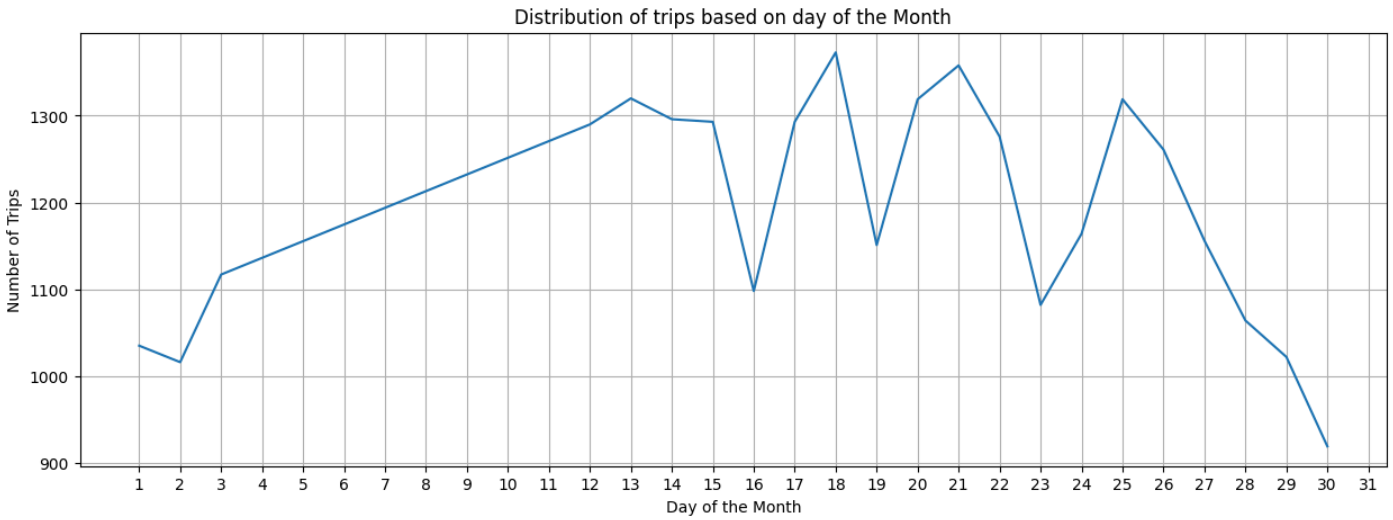
```
# Distribution of trips based on hour of the day
plt.figure(figsize=(15, 5))
trip_counts_by_hour = agg_df['trip_creation_hour'].value_counts().sort_index()
sns.lineplot(x=trip_counts_by_hour.index, y=trip_counts_by_hour.values, palette='viridis')
plt.title('Distribution of trips based on hour of the day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
plt.xticks(np.arange(0,24))
plt.grid('both')
plt.show()
```



Insights:

- The trips are incresing after the noon. The trips are highest at 22:00 hours.
- The trips are declining after 22:00 hours and the trips are lowest at 11:00 hours.

```
# Distribution of trips based on day of the Month
plt.figure(figsize=(15, 5))
trip_counts_by_day = agg_df['trip_creation_day'].value_counts().sort_index()
sns.lineplot(x=trip_counts_by_day.index, y=trip_counts_by_day.values, palette='viridis')
plt.title('Distribution of trips based on day of the Month')
plt.xlabel('Day of the Month')
plt.ylabel('Number of Trips')
plt.xticks(np.arange(1, 32))
plt.grid('both')
plt.show()
```

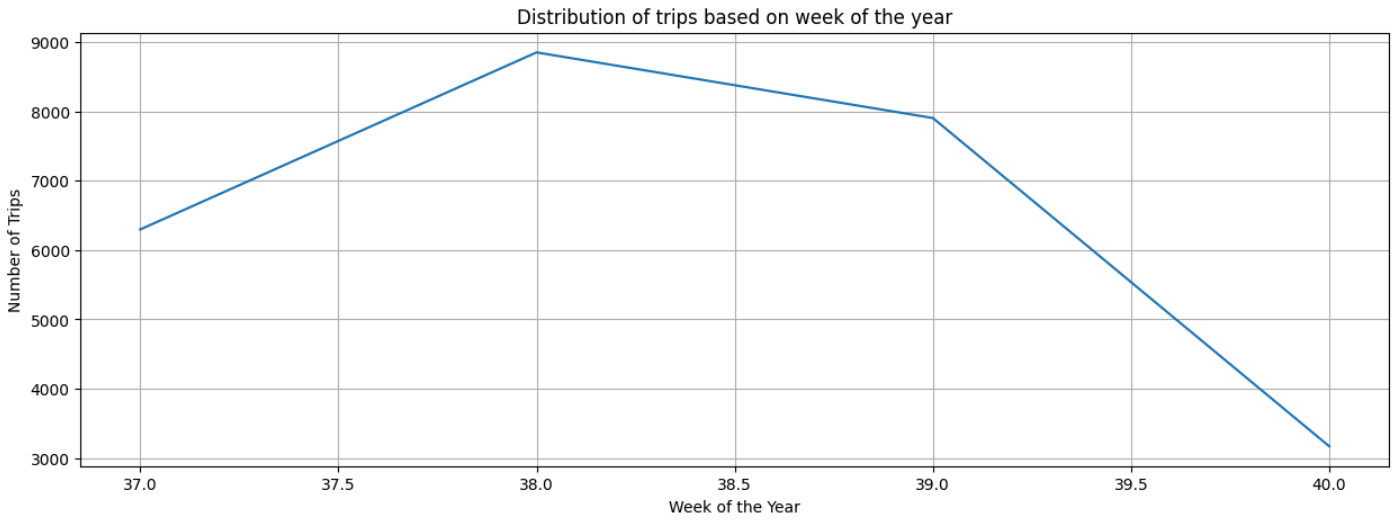


Insights:

- Most of the trips are happened on the middle of the month.
- The trips are highest on the 17th of the month.
- The trips are declining after the 25th of the month and the trips are lowest on the 30st of the month.

```
# Distribution of trips based on week of the year
plt.figure(figsize=(15, 5))
trip_counts_by_week = agg_df['trip_creation_week'].value_counts().sort_index()
sns.lineplot(x=trip_counts_by_week.index, y=trip_counts_by_week.values, palette='viridis')
plt.title('Distribution of trips based on week of the year')
plt.xlabel('Week of the Year')
```

```
plt.ylabel('Number of Trips')
plt.grid('both')
plt.show()
```

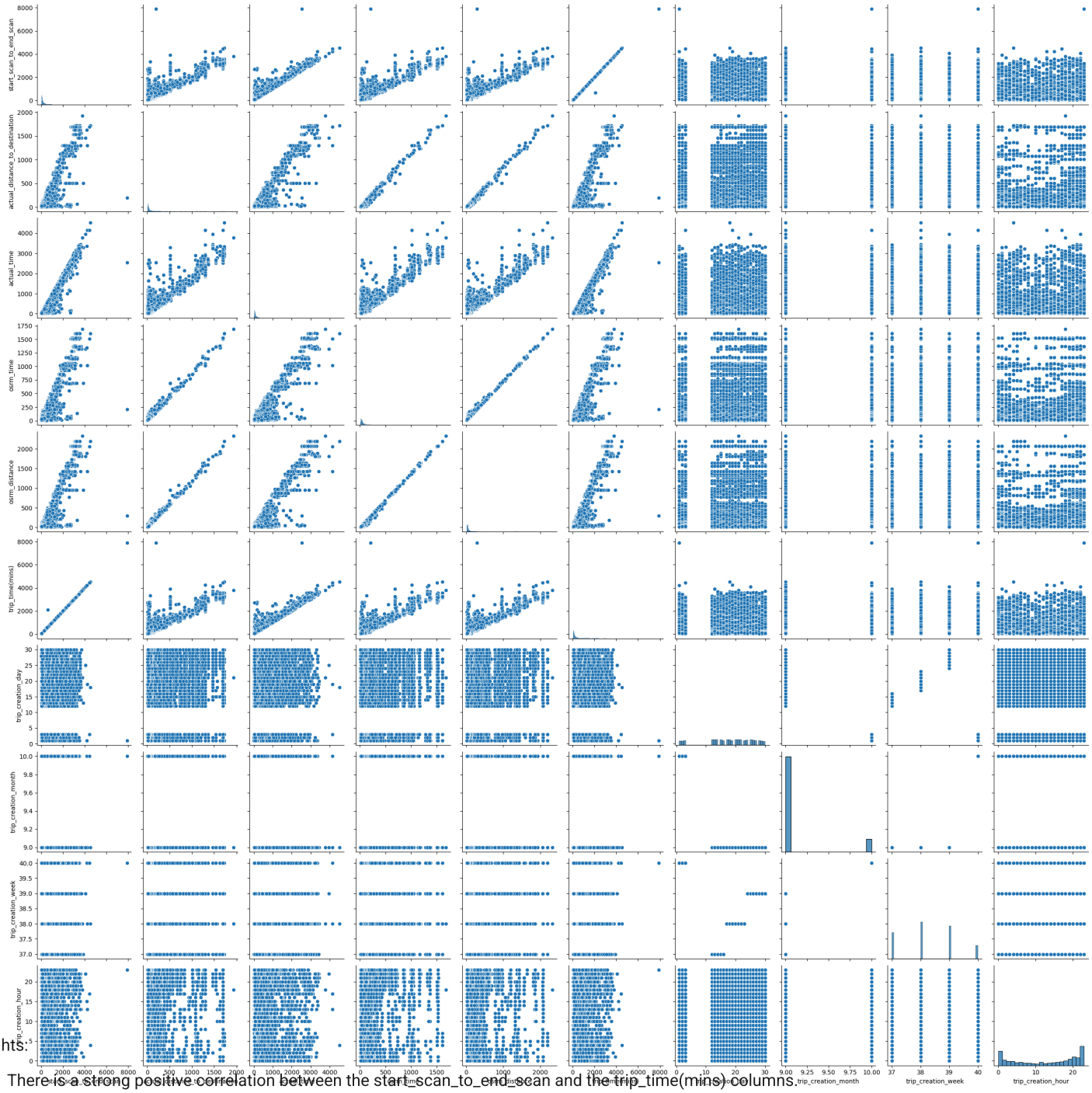


Insights:

- The trips are highest on 38th week of the year.
- The trips are declining after the 38th week of the year.
- The trips are lowest on the 40th week of the year.

```
# pairplot of numerical columns
num_cols = agg_df.select_dtypes(include=['int64', 'float64']).columns.tolist()
plt.figure(figsize=(15, 15))
sns.pairplot(agg_df[num_cols])
plt.show()
```

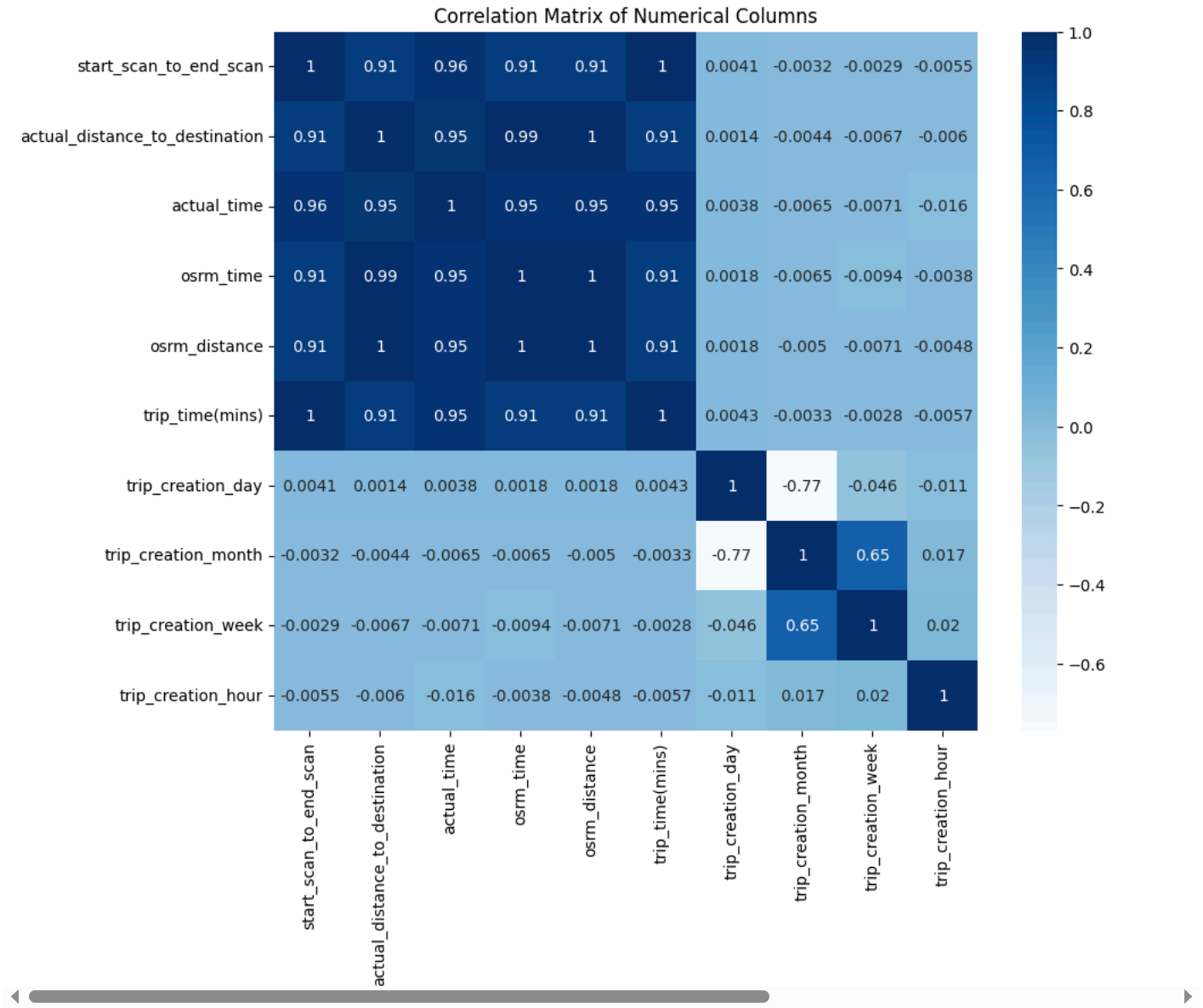
<Figure size 1500x1500 with 0 Axes>



Insights:

- There is a strong positive correlation between the start_scan_to_end_scan and the trip_time(mins) columns.
- There is also a kind of positive correlation between the actual_distance_to_destination and the osrm_distance columns.
- There is also a kind of positive correlation between the actual_time and the osrm_time columns.
- There is a strong positive correlation between the osrm_distance and the osrm_time columns.
- There is also a kind of positive correlation between the actual_distance_to_destination and the actual_time columns.

```
# Correlation matrix of numerical columns
plt.figure(figsize=(10, 8))
sns.heatmap(agg_df[num_cols].corr(), annot=True, cmap='Blues')
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
```



Insights:

- The correleation Matrix proves the Pair Plot insights.
- Trip Time and Start Scan to End Scan are highly correlated.

Indepth Analysis - Feature Engineering

Hypothesis Testing

1. Compare the difference between od_total_time and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: There is no difference between od_total_time and start_scan_to_end_scan.
- Alternate Hypothesis: There is a difference between od_total_time and start_scan_to_end_scan.

Step 2: Identify the test

- Two Sample T-Test
  - To do the two-sample t-test, we need to check the assumptions:
    - Normality Test:
      - QQ Plot
      - Shapiro-Wilk Test
    - Homogeneity of Variance Test:

    - Levene Test
- Kolmogorov-Smirnov Test

Step 3: Decide the significance level

- Significance level is 0.05

Step 4: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

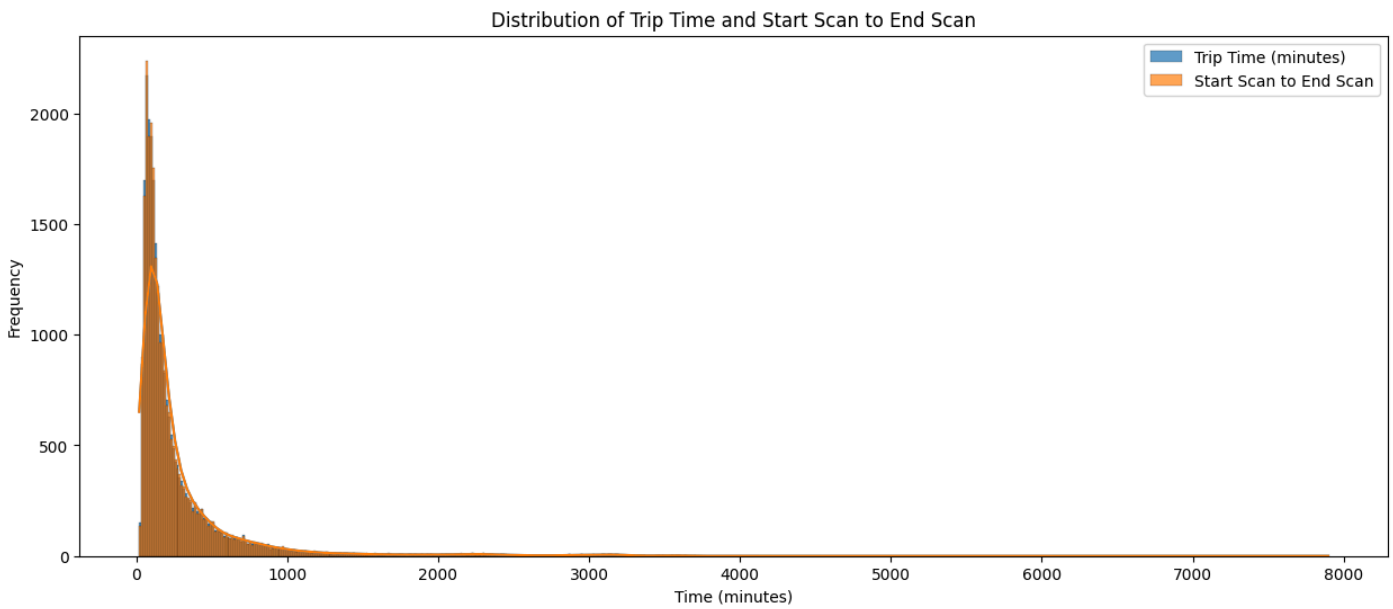
```
agg_df[['trip_time(mins)', 'start_scan_to_end_scan' ]].describe()
```

	trip_time(mins)	start_scan_to_end_scan
count	26222.000000	26222.000000
mean	299.107278	298.553390
std	441.249287	441.116816
min	20.702813	20.000000
25%	90.977759	90.000000
50%	152.336732	152.000000
75%	307.285979	307.000000
max	7898.551955	7898.000000

Insights:

- Based on the statistical analysis, there is no difference between od_total_time and start_scan_to_end_scan.
- Will do the visual analysis to check the difference between od_total_time and start_scan_to_end_scan.

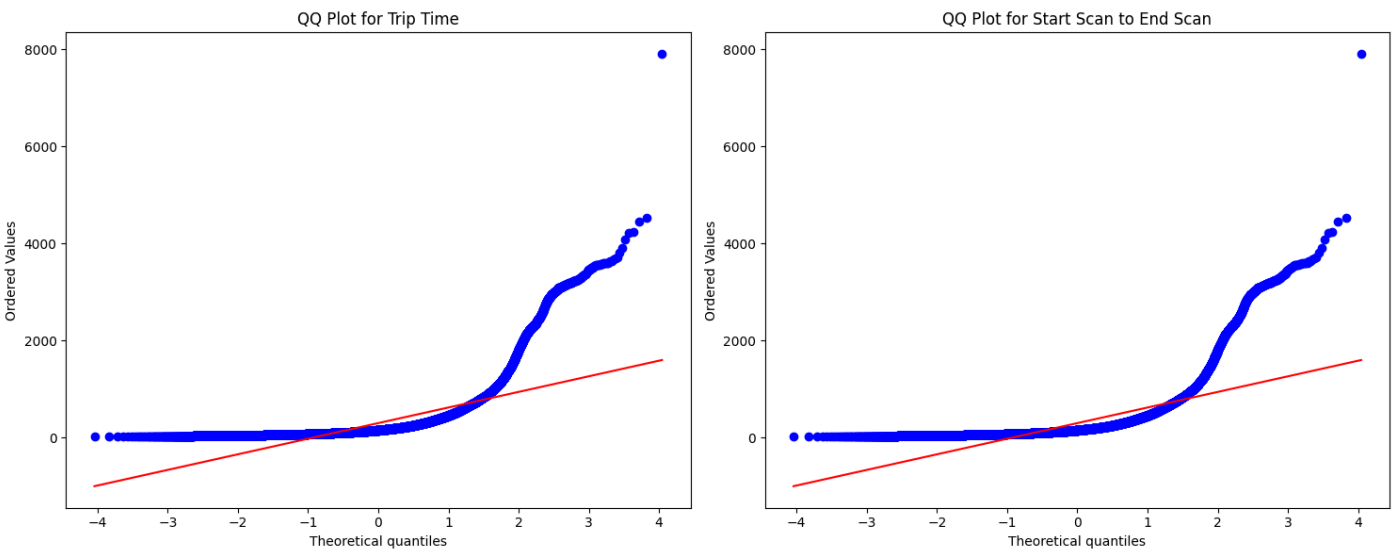
```
def draw_histograms(data1, data2, label1, label2, title, x_label, y_label):
    # Set the figure size
    plt.figure(figsize=(15, 6))

    # Plot the histogram for data1
    sns.histplot(data1, kde=True, alpha=0.7, label=label1)

    # Plot the histogram for data2
    sns.histplot(data2, kde=True, alpha=0.7, label=label2)

    # Add title and labels
    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)

    # Add legend
    plt.legend()
    plt.show()

# Distribution of trip time and start scan to end scan
draw_histograms(agg_df['trip_time(mins)'], agg_df['start_scan_to_end_scan'], 'Trip Time (minutes)', 'Start Scan to End Scan',
                'Distribution of Trip Time and Start Scan to End Scan', 'Time (minutes)', 'Frequency')
```



Insights:

- Visual analysis also proves that there is no difference between od_total_time and start_scan_to_end_scan.

```
def draw_qq_plot(data1, data2, title1, title2):
    # Create a figure and axis
    fig, ax = plt.subplots(1, 2, figsize=(15, 6))

    # Plot the QQ plot for Trip Time
    stats.probplot(data1, dist='norm', plot=ax[0])
    ax[0].set_title(title1)

    # Plot the QQ plot for Start Scan to End Scan
    stats.probplot(data2, dist='norm', plot=ax[1])
    ax[1].set_title(title2)

    plt.tight_layout()
    plt.show()

# QQ Plot for Trip Time and Start Scan to End Scan
draw_qq_plot(agg_df['trip_time(mins)'], agg_df['start_scan_to_end_scan'], 'QQ Plot for Trip Time', 'QQ Plot for Start Scan to End Scan')
```



Insights:

- From the graph, we can see that the both od_total_time and start_scan_to_end_scan are not normally distributed.
- Will quantify the results using Shapiro-Wilk Test.

Shapiro-Wilk Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is normally distributed.
- Alternate Hypothesis: The data is not normally distributed.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Shapiro-Wilk Test for Normality
def shapiro_wilk_test(data):
    stat, p_value = stats.shapiro(data)
    print(f'Shapiro-Wilk Test for Normality')
    print(f'Statistics: {stat}')
    print(f'p-value: {p_value}')
    if p_value < 0.05:
        print('The data is not normally distributed')
    else:
        print('The data is normally distributed')

shapiro_wilk_test(agg_df['trip_time(mins)'])
print('-' * 50)
shapiro_wilk_test(agg_df['start_scan_to_end_scan'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.5304400949805688
p-value: 5.165804867926899e-122
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.5304338308747462
p-value: 5.161369599523079e-122
The data is not normally distributed
```

Levene Test: (Homogeneity of Variance Test)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data has equal variance.
- Alternate Hypothesis: The data does not have equal variance.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Levene's Test for Homogeneity of Variance
def levene_test(data1, data2):
    stat, p_value = stats.levene(data1, data2)
    print(f'Levene\'s Test for Homogeneity of Variance')
    print(f'Statistics: {stat}')
    print(f'p-value: {p_value}')
    if p_value < 0.05:
        print('The variances are not equal')
    else:
        print('The variances are equal')

levene_test(agg_df['trip_time(mins)'], agg_df['start_scan_to_end_scan'])
```

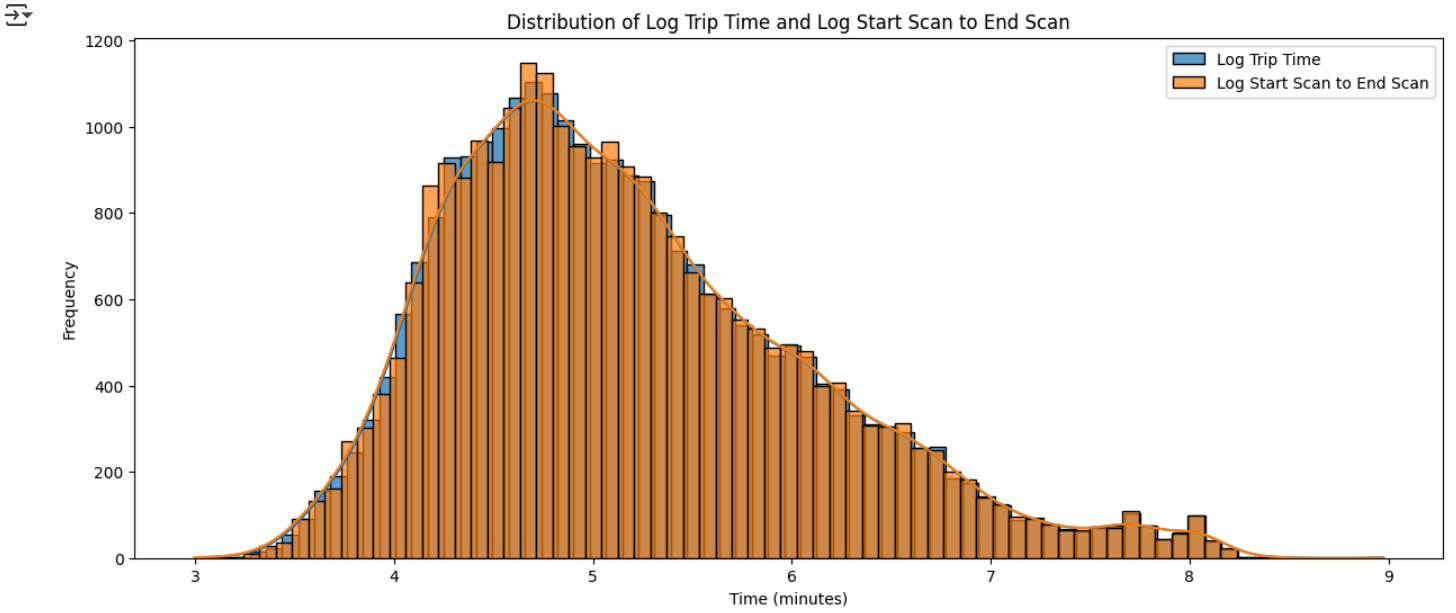
```
Levene's Test for Homogeneity of Variance
Statistics: 0.0002328353382060787
p-value: 0.9878256555192264
The variances are equal
```

Insights:

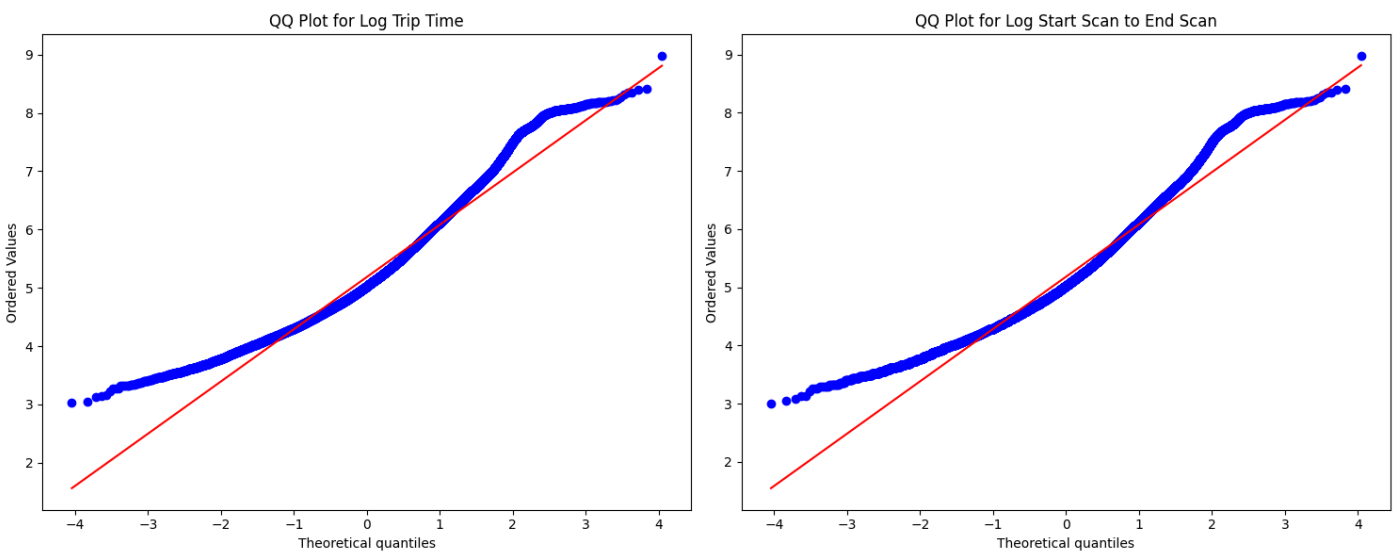
- Based on the statistical analysis, the data is not normally distributed.
- The data does have equal variance.
- Hence, we can't do the two-sample t-test.

Trying to convert the data to normal distribution using log transformation.

```
def convert_to_log_normal_and_visualize(df, data1, data2, label1, label2, title, xlabel, ylabel):
    # Convert into Log Normal Distribution
    df['log_'+data1] = np.log(df[data1])
    df['log_'+data2] = np.log(df[data2])

    # Draw Histograms
    draw_histograms(df['log_'+data1], df['log_'+data2], label1, label2, title, xlabel, ylabel)

convert_to_log_normal_and_visualize(agg_df,'trip_time(mins)', 'start_scan_to_end_scan', 'Log Trip Time', 'Log Start Scan to End Scan', 'Distr
```




```
draw_qq_plot(agg_df['log_trip_time(mins)'], agg_df['log_start_scan_to_end_scan'], 'QQ Plot for Log Trip Time', 'QQ Plot for Log Start Scan to
```




```
# shapiro_wilk_test for Log Normal Distribution
shapiro_wilk_test(agg_df['log_trip_time(mins)'])
print('-' * 50)
shapiro_wilk_test(agg_df['log_start_scan_to_end_scan'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.9590723731359648
p-value: 1.8284089200671066e-63
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.9596914530086159
p-value: 3.689691261086178e-63
The data is not normally distributed
```

Insights:

- Even after log transformation, the data is not normally distributed.
- Hence, we can't do the two-sample t-test.
- Will go with Kolmogorov-Smirnov Test.

```
# Drop the log transformed columns
agg_df.drop(['log_trip_time(mins)', 'log_start_scan_to_end_scan'], axis=1, inplace=True)
```

Kolmogorov-Smirnov Test: (Non-Parametric Test)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is from the same distribution.
- Alternate Hypothesis: The data is not from the same distribution.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# 2 Sample KS Test for Non-Normal Distribution
def ks_test(data1, data2):
    stat, p_value = stats.ks_2samp(data1, data2)
    print(f'2 Sample KS Test for Non-Normal Distribution')
    print(f'Statistics: {stat}')
    print(f'p-value: {p_value}')
    if p_value < 0.05:
        print('The distributions are different')
    else:
```

```
        print('The distributions are same')

ks_test(agg_df['trip_time(mins)'], agg_df['start_scan_to_end_scan'])
```

```
2 Sample KS Test for Non-Normal Distribution
Statistics: 0.006902600869498898
p-value: 0.5575053318743617
The distributions are same
```

Insights:

- Based on the statistical analysis, the data is from the same distribution.
- Hence, there is no difference between od_total_time and start_scan_to_end_scan.

2. Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: There is no difference between actual_time and OSRM time.
- Alternate Hypothesis: There is a difference between actual_time and OSRM time.

Step 2: Identify the test

- Two Sample T-Test
    - To do the two-sample t-test, we need to check the assumptions:
        - Normality Test:
            - QQ Plot
            - Shapiro-Wilk Test
        - Homogeneity of Variance Test:
            - Levene Test
- Kolmogorov-Smirnov Test

Step 3: Decide the significance level

- Significance level is 0.05

Step 4: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

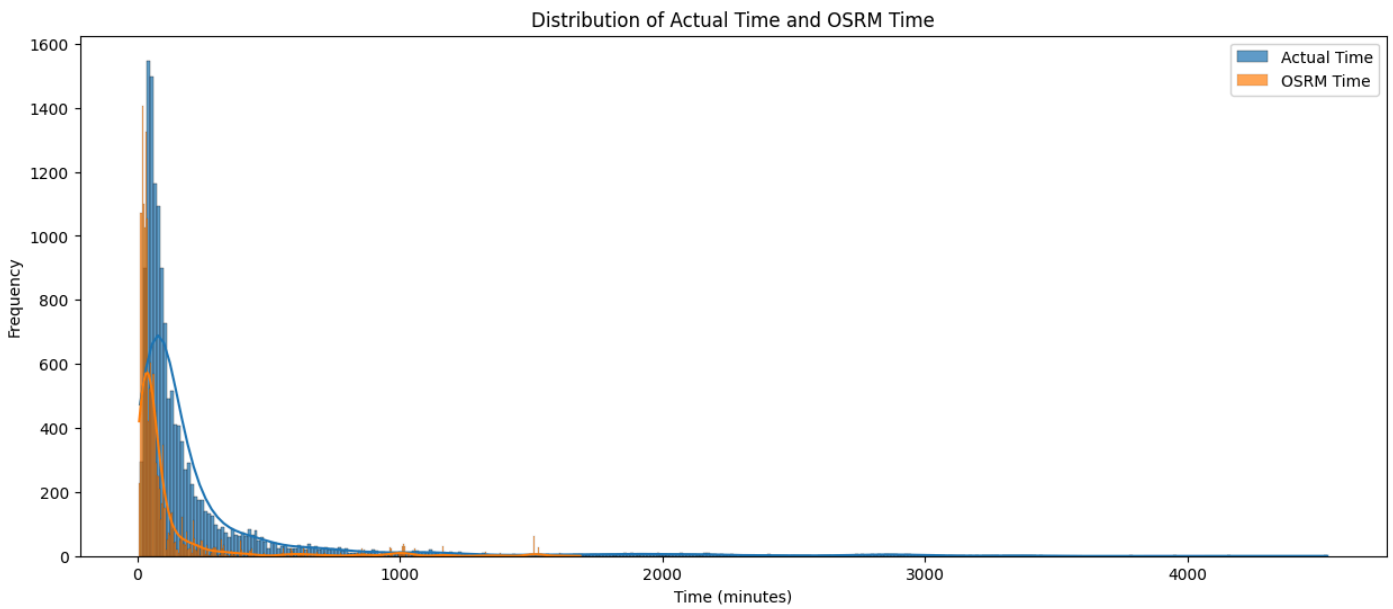
```
agg_df_2[['actual_time', 'osrm_time']].describe()
```

	actual_time	osrm_time
count	14787.000000	14787.000000
mean	251.134375	108.303645
std	455.202399	218.447496
min	9.000000	6.000000
25%	56.000000	24.000000
50%	98.000000	41.000000
75%	213.000000	82.000000
max	4532.000000	1686.000000

Insights:

- Based on the statistical data, there is a difference between actual_time and OSRM time.
- Will do the visual analysis to check the difference between actual_time and OSRM time.

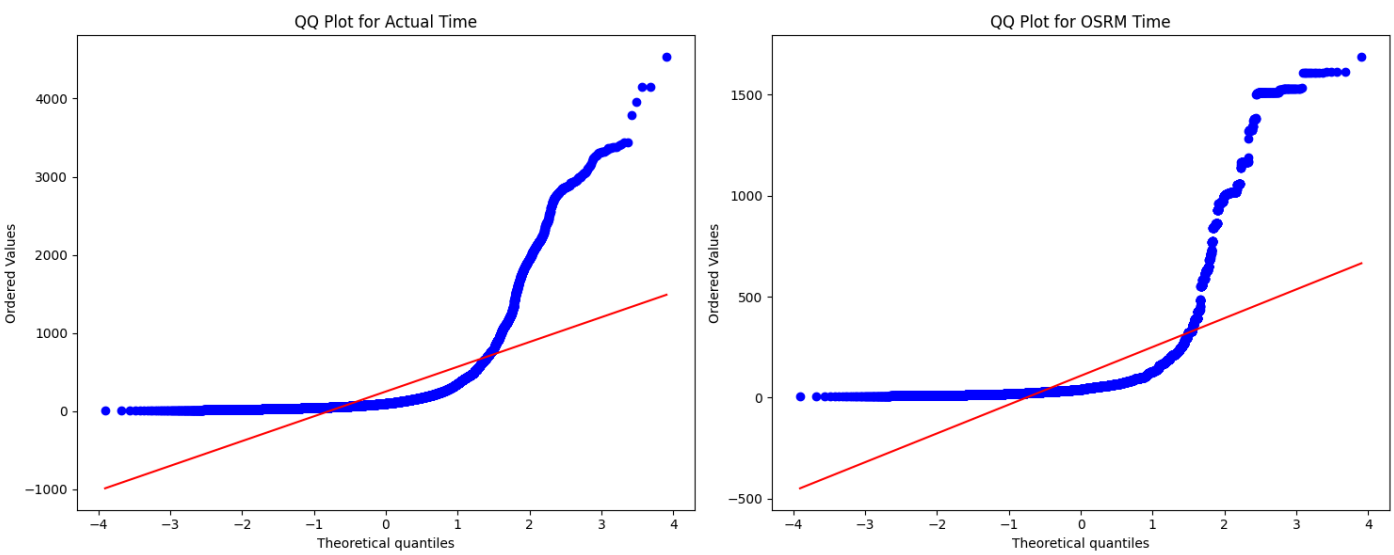
```
# Distribution of Actual Time and OSRM Time
draw_histograms(agg_df_2['actual_time'], agg_df_2['osrm_time'], 'Actual Time', 'OSRM Time',
                'Distribution of Actual Time and OSRM Time', 'Time (minutes)', 'Frequency')
```



Insights:

- Visual analysis also proves that there is a slight difference between actual_time and OSRM time.
- Using Statistical methods, we can test whether the difference is significant or not.

```
# Draw QQ Plot for Actual Time and OSRM Time
draw_qq_plot(agg_df_2['actual_time'], agg_df_2['osrm_time'], 'QQ Plot for Actual Time', 'QQ Plot for OSRM Time')
```



Insights:

- From the graph, we can see that the both actual_time and OSRM time are not normally distributed.
- Will quantify the results using Shapiro-Wilk Test.

Shapiro-Wilk Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is normally distributed.
- Alternate Hypothesis: The data is not normally distributed.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['actual_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['osrm_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.48501426291685024
p-value: 8.503891476364121e-109
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.4258338184867536
p-value: 1.1976397286640894e-111
The data is not normally distributed
```

```
# Levene's Test for Homogeneity of Variance
levene_test(agg_df_2['actual_time'], agg_df_2['osrm_time'])
```

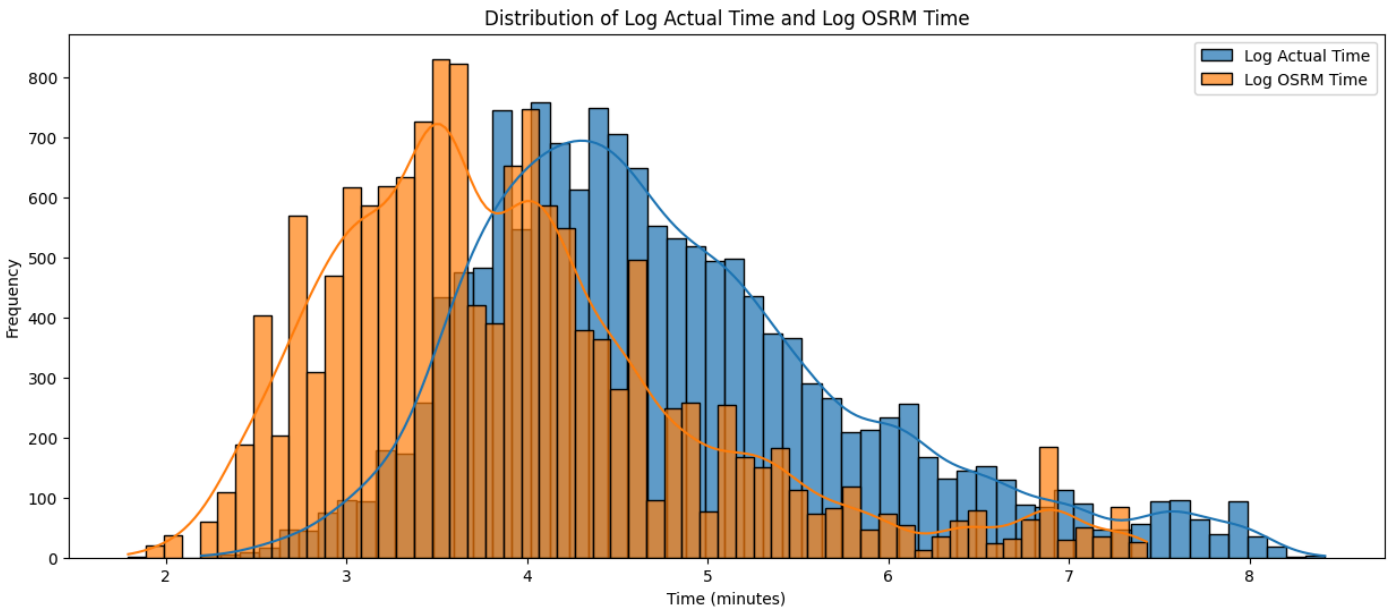
```
Levene's Test for Homogeneity of Variance
Statistics: 759.4074137696683
p-value: 4.417722834578231e-165
The variances are not equal
```

Insights:

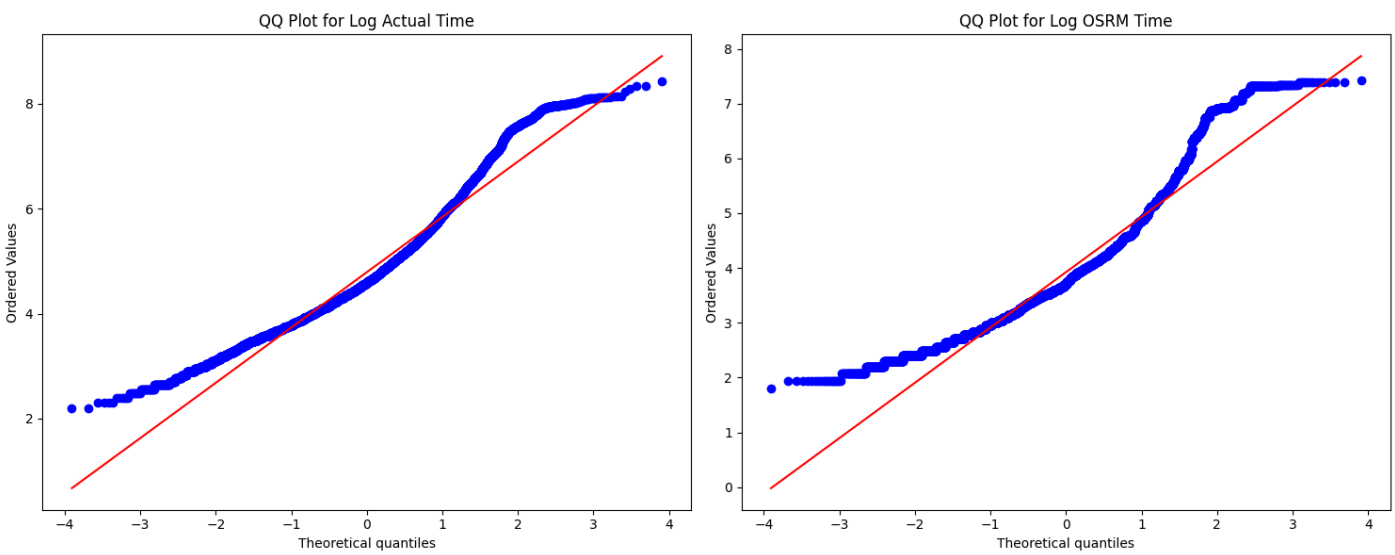
- Based on the statistical analysis, the data is not normally distributed.
- The data does not have equal variance.
- Hence, we can't do the two-sample t-test.

Trying to convert the data to normal distribution using log transformation.

```
# Convert the actual and osrm time into log normal distribution
convert_to_log_normal_and_visualize(agg_df_2, 'actual_time', 'osrm_time', 'Log Actual Time', 'Log OSRM Time', 'Distribution of Log Actual Tim
```




```
# Draw QQ Plot for Log Actual Time and Log OSRM Time
draw_qq_plot(agg_df_2['log_actual_time'], agg_df_2['log_osrm_time'], 'QQ Plot for Log Actual Time', 'QQ Plot for Log OSRM Time')
```




```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['log_actual_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['log_osrm_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.9545744613556878
p-value: 6.567287903162743e-55
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.9302214562901787
p-value: 4.4552379315422194e-63
The data is not normally distributed
```

Insights:

- Even after log transformation, the data is not normally distributed.
- Hence, we can't do the two-sample t-test.
- Will go with Kolmogorov-Smirnov Test.

```
# Drop the log transformed columns
agg_df_2.drop(['log_actual_time', 'log_osrm_time'], axis=1, inplace=True)
```

Kolmogorov-Smirnov Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is from the same distribution.
- Alternate Hypothesis: The data is not from the same distribution.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
#  2 Sample KS Test for Non-Normal Distribution
ks_test(agg_df_2['actual_time'], agg_df_2['osrm_time'])
```

```
2 Sample KS Test for Non-Normal Distribution
Statistics: 0.36904037330087236
p-value: 0.0
The distributions are different
```

Insights:

- Based on the statistical analysis, the data is not from the same distribution.
- Hence, there is a difference between actual_time and OSRM time.

3. Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: There is no difference between actual_time and segment actual time.
- Alternate Hypothesis: There is a difference between actual_time and segment actual time.

Step 2: Identify the test

- Two Sample T-Test
    - To do the two-sample t-test, we need to check the assumptions:
        - Normality Test:
            - QQ Plot
            - Shapiro-Wilk Test
        - Homogeneity of Variance Test:
            - Levene Test
- Kolmogorov-Smirnov Test

Step 3: Decide the significance level

- Significance level is 0.05

Step 4: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

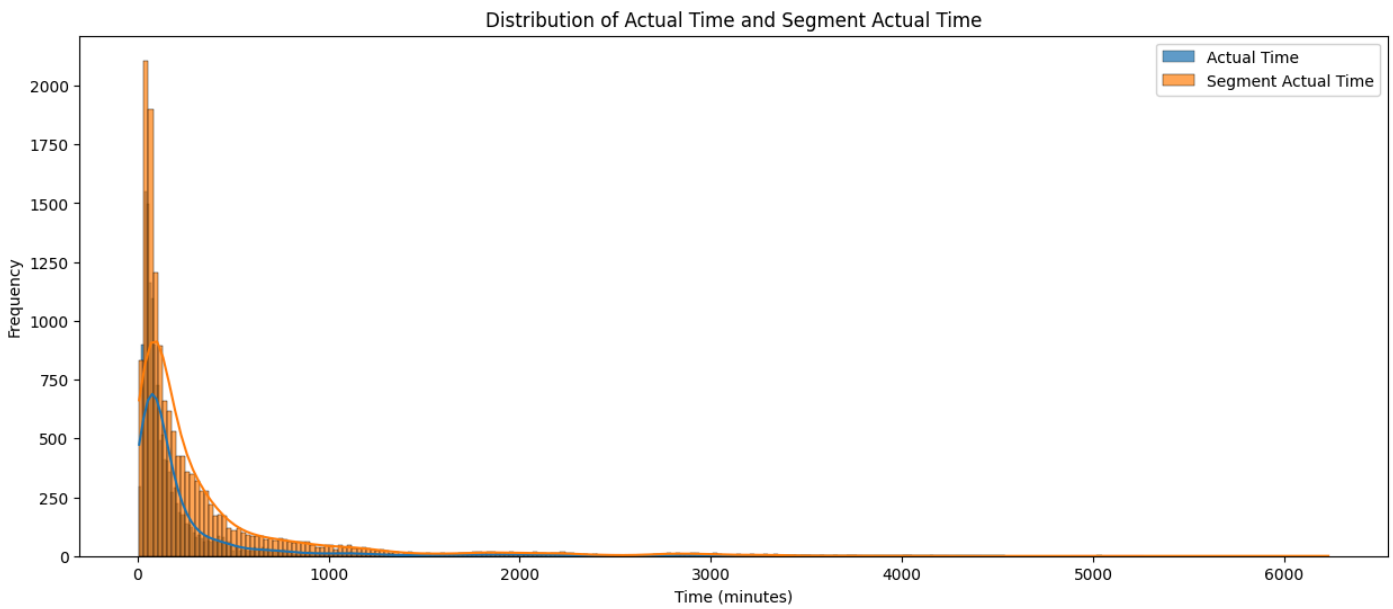
```
agg_df_2[['actual_time', 'segment_actual_time']].describe()
```

	actual_time	segment_actual_time
count	14787.000000	14787.000000
mean	251.134375	353.059174
std	455.202399	556.365911
min	9.000000	9.000000
25%	56.000000	66.000000
50%	98.000000	147.000000
75%	213.000000	364.000000
max	4532.000000	6230.000000

Insights:

- Based on the statistical data, there is no difference between actual_time and segment_actual_time.
- Will do the visual analysis to check the difference between actual_time and segment_actual_time.

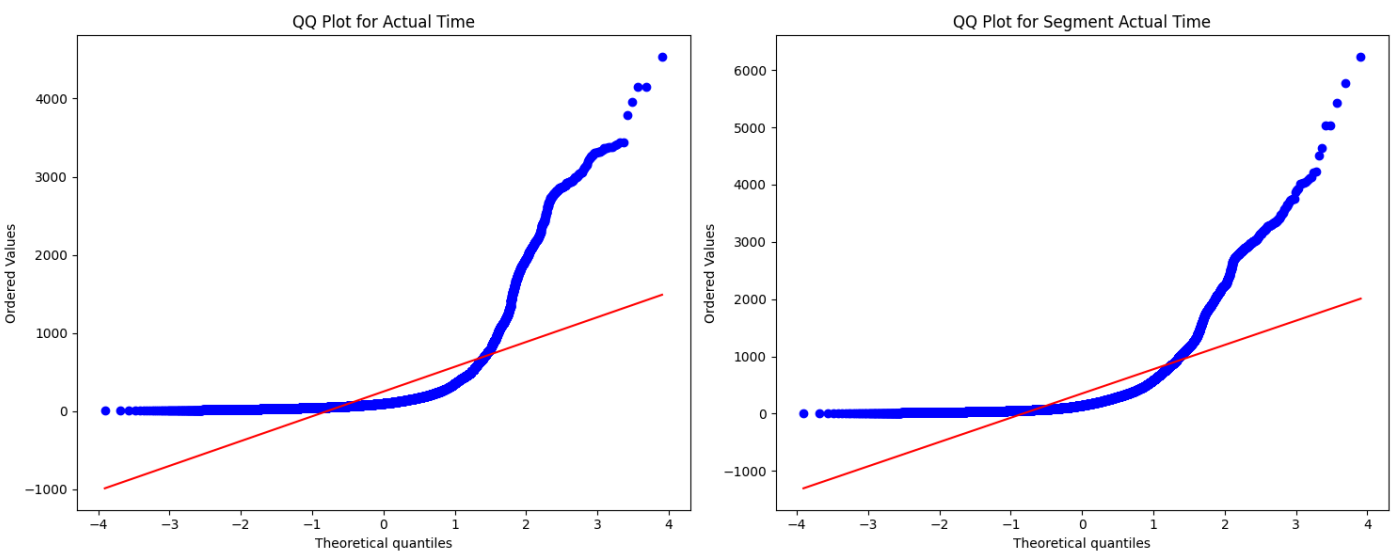
```
# Distribution of Actual Time and Segment Actual Time
draw_histograms(agg_df_2['actual_time'], agg_df_2['segment_actual_time'], 'Actual Time', 'Segment Actual Time',
                'Distribution of Actual Time and Segment Actual Time', 'Time (minutes)', 'Frequency')
```



Insights:

- Visual analysis also proves that there is no difference between actual_time and segment_actual_time.
- Using Statistical methods, we can test whether the conclusion is correct or not.

```
# Draw QQ Plot for Actual Time and Segment Actual Time
draw_qq_plot(agg_df_2['actual_time'], agg_df_2['segment_actual_time'], 'QQ Plot for Actual Time', 'QQ Plot for Segment Actual Time')
```



Insights:

- From the graph, we can see that the both actual_time and segment_actual_time are not normally distributed.
- Will quantify the results using Shapiro-Wilk Test.

Shapiro-Wilk Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is normally distributed.
- Alternate Hypothesis: The data is not normally distributed.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['actual_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['segment_actual_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.48501426291685024
p-value: 8.503891476364121e-109
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.5808456330535781
p-value: 1.6746386852473887e-103
The data is not normally distributed
```

```
# Levene's Test for Homogeneity of Variance
levene_test(agg_df_2['actual_time'], agg_df_2['segment_actual_time'])
```

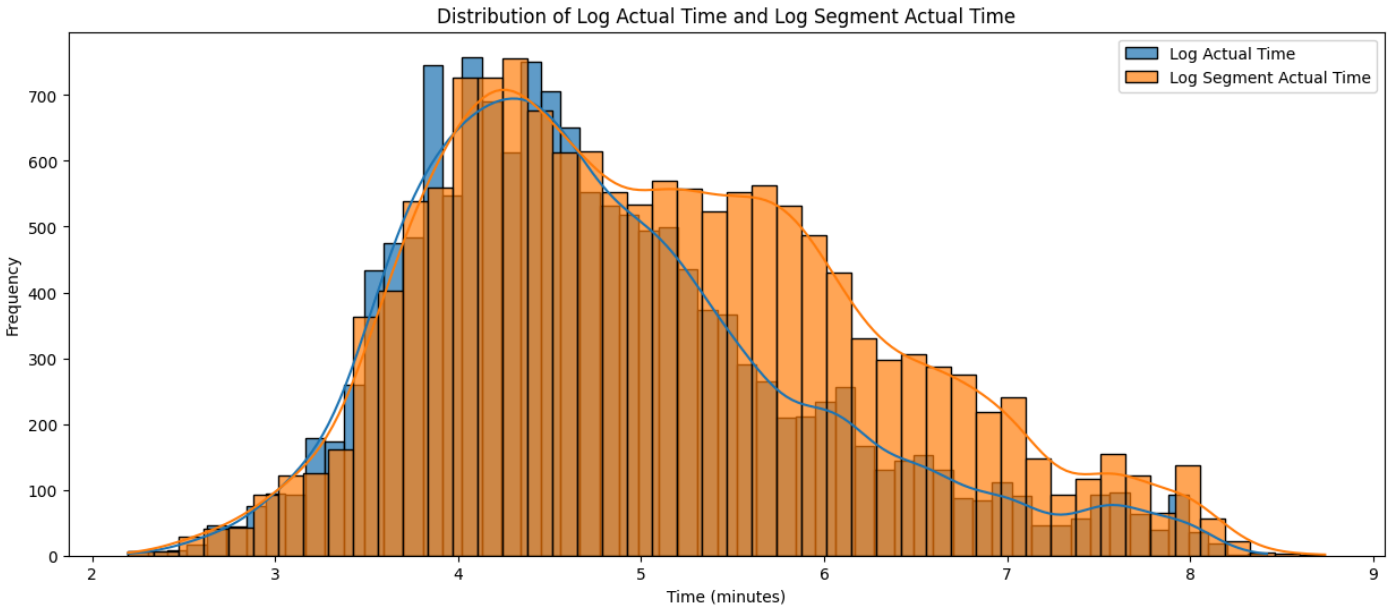
```
Levene's Test for Homogeneity of Variance
Statistics: 243.3695387395901
p-value: 1.1969205169904284e-54
The variances are not equal
```

Insights:

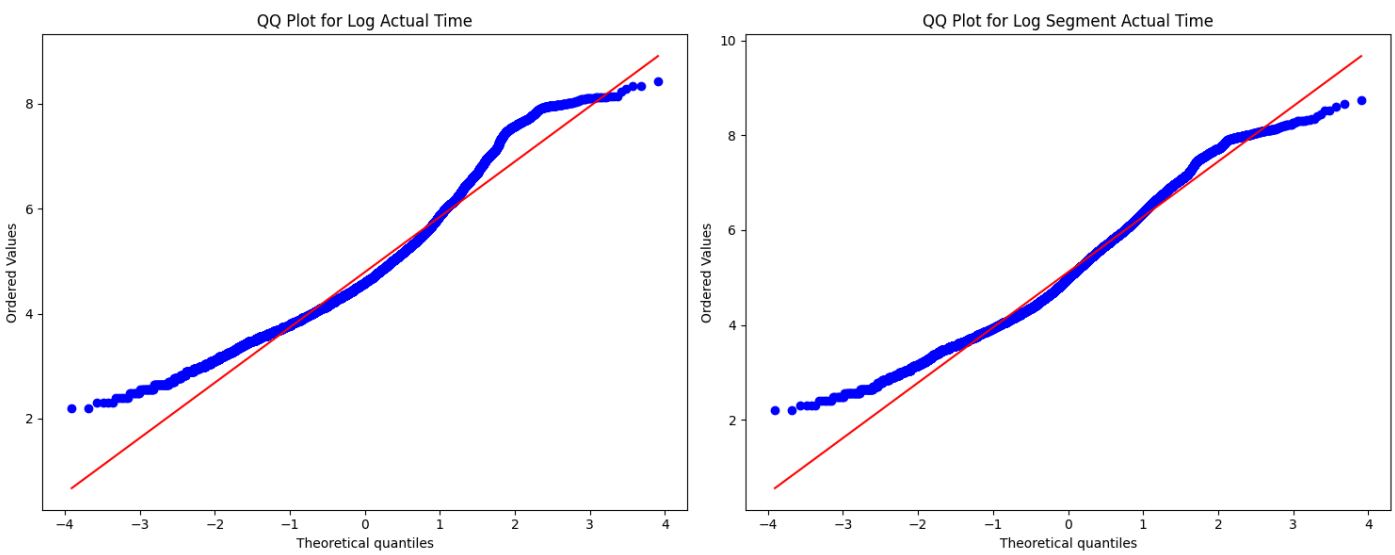
- Based on the statistical analysis, the data is not normally distributed.
- The data does have equal variance.
- Hence, we can't do the two-sample t-test.

Trying to convert the data to normal distribution using log transformation.

```
# convert the actual and segment actual time into log normal distribution
convert_to_log_normal_and_visualize(agg_df_2, 'actual_time', 'segment_actual_time', 'Log Actual Time', 'Log Segment Actual Time', 'Distributi
```




```
# Draw QQ Plot for Log Actual Time and Log Segment Actual Time
draw_qq_plot(agg_df_2['log_actual_time'], agg_df_2['log_segment_actual_time'], 'QQ Plot for Log Actual Time', 'QQ Plot for Log Segment Actual
```




```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['log_actual_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['log_segment_actual_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.9545744613556878
p-value: 6.567287903162743e-55
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.979044236364028
p-value: 1.170653393709495e-41
The data is not normally distributed
```

Insights:

- Even after log transformation, the data is not normally distributed.
- Hence, we can't do the two-sample t-test.
- Will go with Kolmogorov-Smirnov Test.

```
# Drop the log transformed columns
agg_df_2.drop(['log_actual_time', 'log_segment_actual_time'], axis=1, inplace=True)
```

Kolmogorov-Smirnov Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is from the same distribution.
- Alternate Hypothesis: The data is not from the same distribution.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# 2 Sample KS Test for Non-Normal Distribution
ks_test(agg_df_2['actual_time'], agg_df_2['segment_actual_time'])
```

```
2 Sample KS Test for Non-Normal Distribution
Statistics: 0.14749442077500508
p-value: 7.29925803783217e-141
The distributions are different
```

Insights:

- Based on the statistical analysis, the data is from the same distribution.
- Hence, there is no difference between actual_time and segment_actual_time.

4. Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: There is no difference between osrm_distance and segment_osrm_distance.
- Alternate Hypothesis: There is a difference between osrm_distance and segment_osrm_distance.

Step 2: Identify the test

- Two Sample T-Test
    - To do the two-sample t-test, we need to check the assumptions:
        - Normality Test:
            - QQ Plot
            - Shapiro-Wilk Test
        - Homogeneity of Variance Test:
            - Levene Test
- Kolmogorov-Smirnov Test

Step 3: Decide the significance level

- Significance level is 0.05

Step 4: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

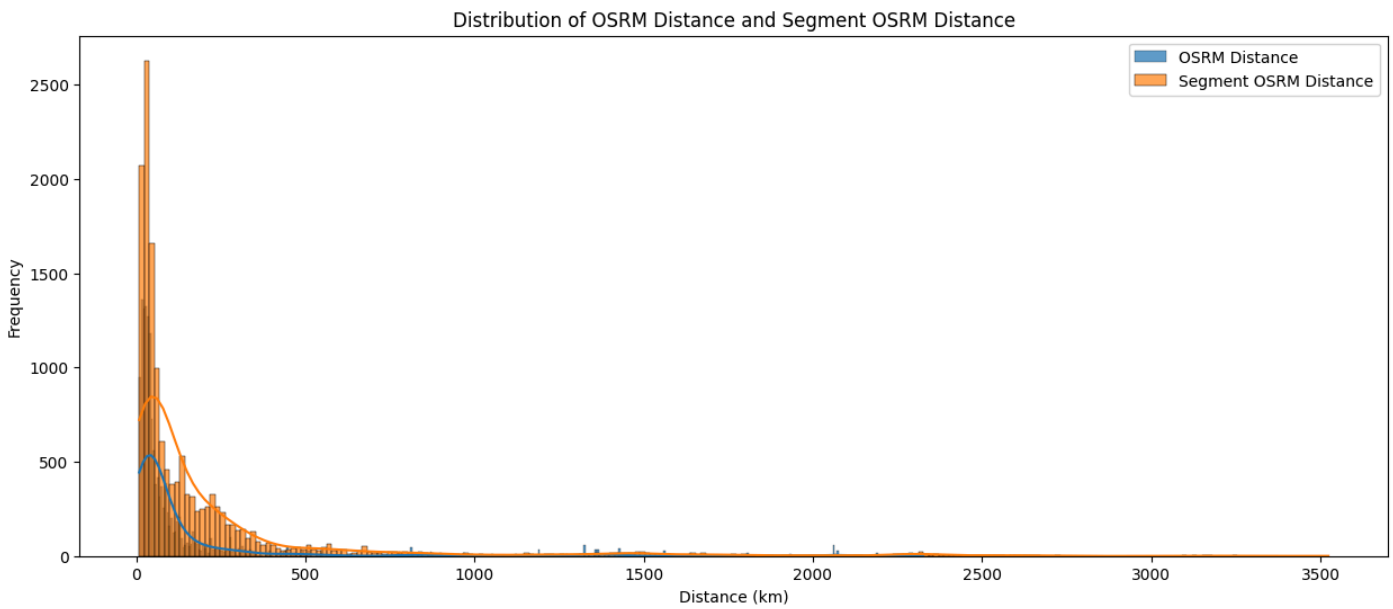
```
agg_df_2[['osrm_distance', 'segment_osrm_distance']].describe()
```

	osrm_distance	segment_osrm_distance
count	14787.000000	14787.000000
mean	138.288962	222.705466
std	300.259409	416.846279
min	9.072900	9.072900
25%	26.871750	32.578850
50%	43.905100	69.784200
75%	98.201250	216.560600
max	2326.199100	3523.632400

Insights:

- Based on the statistical data, there is no difference between osrm_distance and segment_osrm_distance.
- Will do the visual analysis to check the difference between osrm_distance and segment_osrm_distance.

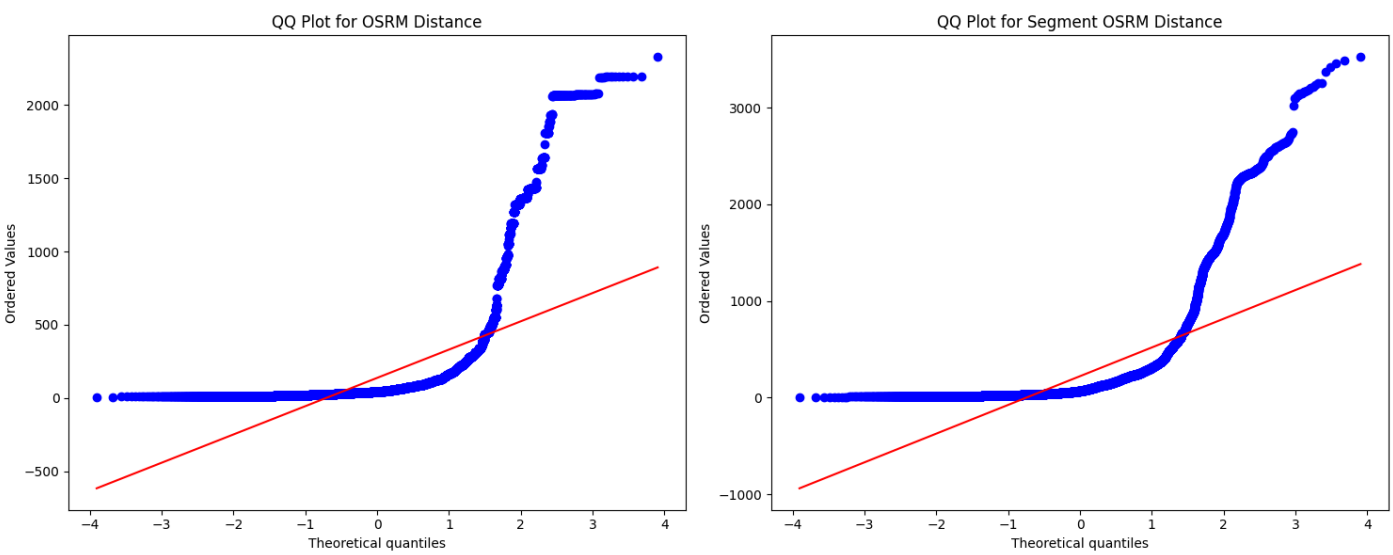
```
# Distribution of OSRM Distance and Segment OSRM Distance
draw_histograms(agg_df_2['osrm_distance'], agg_df_2['segment_osrm_distance'], 'OSRM Distance', 'Segment OSRM Distance',
                'Distribution of OSRM Distance and Segment OSRM Distance', 'Distance (km)', 'Frequency')
```



Insights:

- Visual analysis also proves that there is no difference between osrm_distance and segment_osrm_distance.
- Using Statistical methods, we can test whether the conclusion is correct or not.

```
# Draw QQ Plot for OSRM Distance and Segment OSRM Distance
draw_qq_plot(agg_df_2['osrm_distance'], agg_df_2['segment_osrm_distance'], 'QQ Plot for OSRM Distance', 'QQ Plot for Segment OSRM Distance')
```



Insights:

- From the graph, we can see that the both osrm_distance and segment_osrm_distance are not normally distributed.
- Will quantify the results using Shapiro-Wilk Test.

Shapiro-Wilk Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is normally distributed.
- Alternate Hypothesis: The data is not normally distributed.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['osrm_distance'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['segment_osrm_distance'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.41228074588682406
p-value: 2.896753685020016e-112
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.5071258876719171
p-value: 1.1730299870400521e-107
The data is not normally distributed
```

```
# Levene's Test for Homogeneity of Variance
levene_test(agg_df_2['osrm_distance'], agg_df_2['segment_osrm_distance'])
```

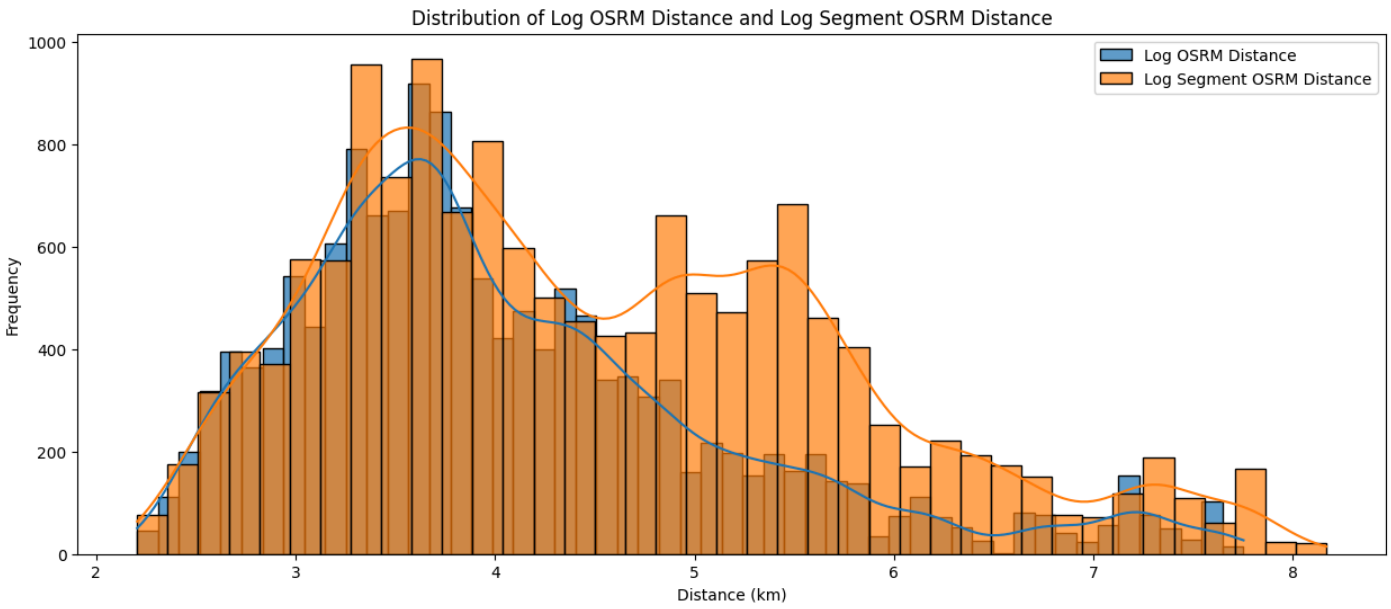
```
Levene's Test for Homogeneity of Variance
Statistics: 349.0792897665722
p-value: 1.879699723806504e-77
The variances are not equal
```

Insights:

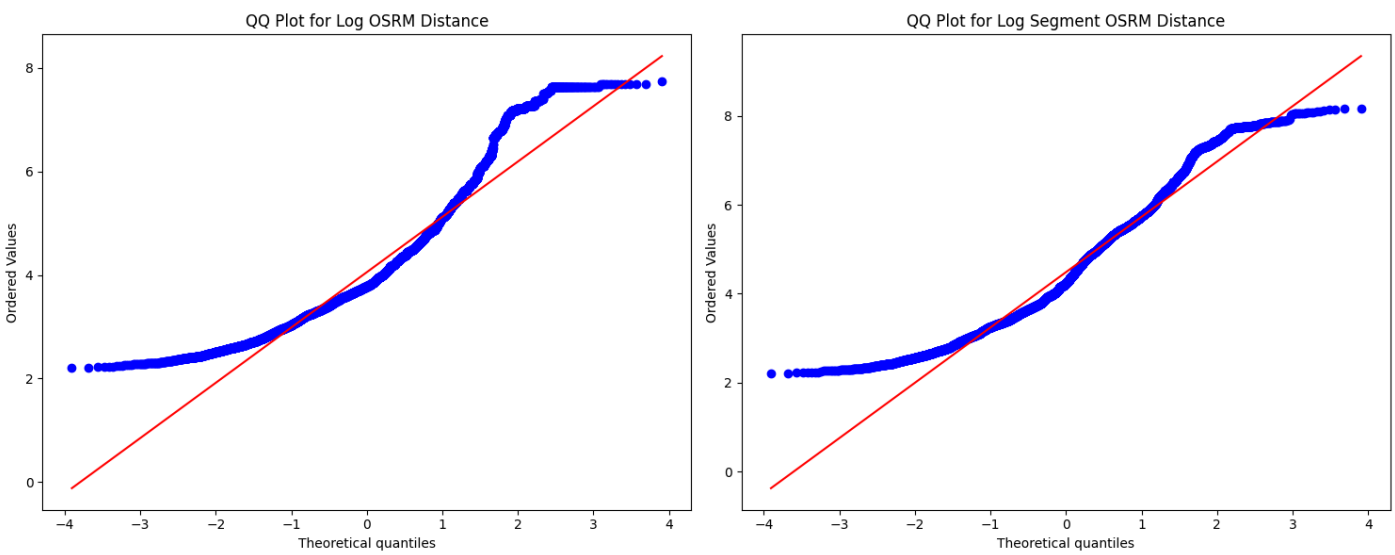
- Based on the statistical analysis, the data is not normally distributed.
- The data does have equal variance.
- Hence, we can't do the two-sample t-test.

Trying to convert the data to normal distribution using log transformation.

```
# convert the osrm and segment osrm distance into log normal distribution
convert_to_log_normal_and_visualize(agg_df_2, 'osrm_distance', 'segment_osrm_distance', 'Log OSRM Distance', 'Log Segment OSRM Distance', 'Di
```




```
# Draw QQ Plot for Log OSRM Distance and Log Segment OSRM Distance
draw_qq_plot(agg_df_2['log_osrm_distance'], agg_df_2['log_segment_osrm_distance'], 'QQ Plot for Log OSRM Distance', 'QQ Plot for Log Segment
```




```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['log_osrm_distance'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['log_segment_osrm_distance'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.9206203747961604
p-value: 1.2073211358372833e-65
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.9612578809307818
p-value: 4.986512109918107e-52
The data is not normally distributed
```

Insights:

- Even after log transformation, the data is not normally distributed.
- Hence, we can't do the two-sample t-test.
- Will go with Kolmogorov-Smirnov Test.

```
# Drop the log transformed columns
agg_df_2.drop(['log_osrm_distance', 'log_segment_osrm_distance'], axis=1, inplace=True)
```

Kolmogorov-Smirnov Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is from the same distribution.
- Alternate Hypothesis: The data is not from the same distribution.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# 2 Sample KS Test for Non-Normal Distribution
ks_test(agg_df_2['osrm_distance'], agg_df_2['segment_osrm_distance'])
```

```
2 Sample KS Test for Non-Normal Distribution
Statistics: 0.18712382498140256
p-value: 3.883692296416939e-227
The distributions are different
```

Insights:

- Based on the statistical analysis, the data is from the same distribution.
- Hence, there is no difference between osrm_distance and segment_osrm_distance.

5. Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: There is no difference between osrm_time and segment_osrm_time.
- Alternate Hypothesis: There is a difference between osrm_time and segment_osrm_time.

Step 2: Identify the test

- Two Sample T-Test
    - To do the two-sample t-test, we need to check the assumptions:
        - Normality Test:
            - QQ Plot
            - Shapiro-Wilk Test
        - Homogeneity of Variance Test:
            - Levene Test
- Kolmogorov-Smirnov Test

Step 3: Decide the significance level

- Significance level is 0.05

Step 4: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

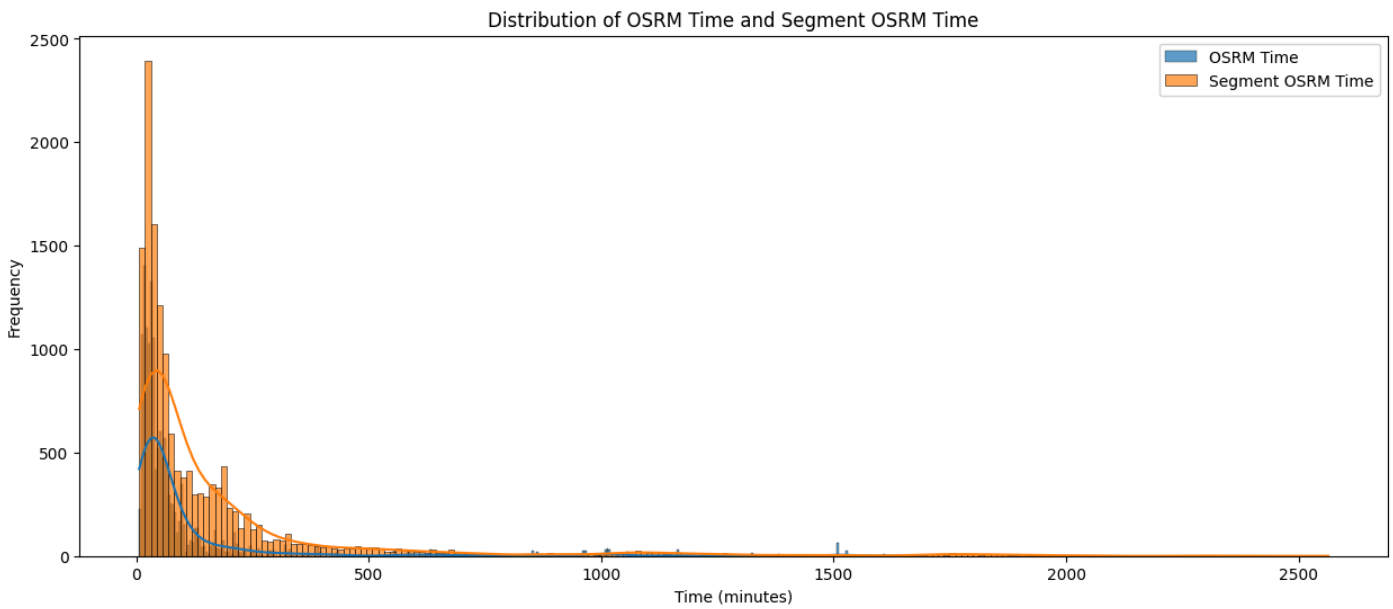
```
agg_df_2[['osrm_time', 'segment_osrm_time']].describe()
```

	osrm_time	segment_osrm_time
count	14787.000000	14787.000000
mean	108.303645	180.511598
std	218.447496	314.679279
min	6.000000	6.000000
25%	24.000000	30.000000
50%	41.000000	65.000000
75%	82.000000	184.000000
max	1686.000000	2564.000000

Insights:

- Based on the statistical data, there is no difference between osrm_time and segment_osrm_time.
- Will do the visual analysis to check the difference between osrm_time and segment_osrm_time.

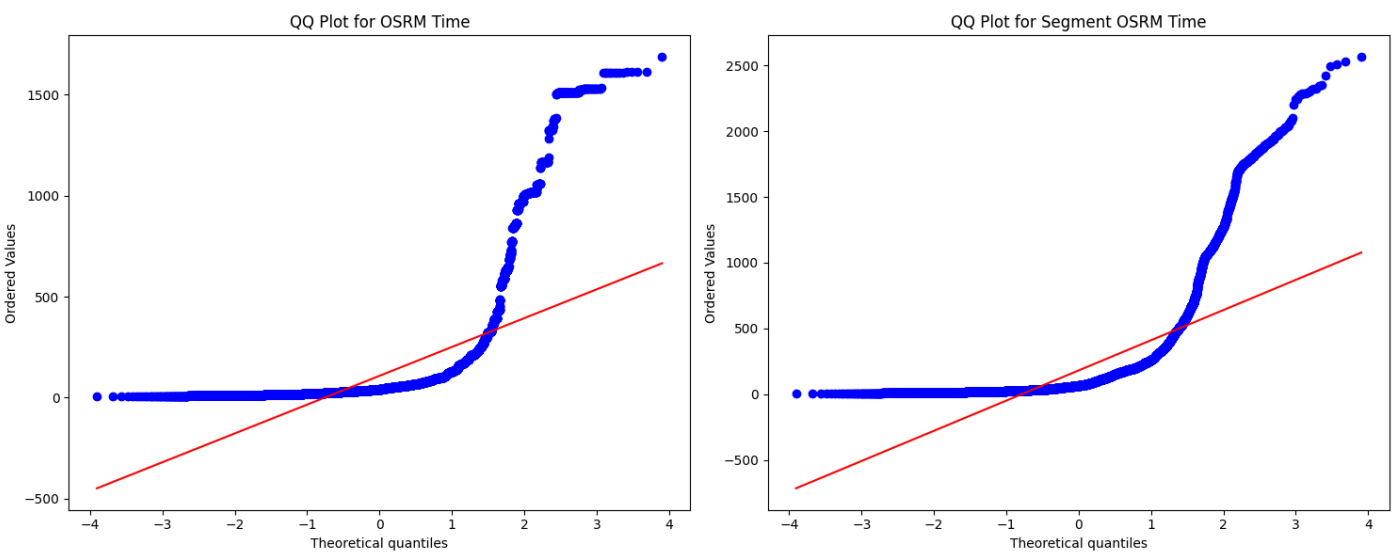
```
# Distribution of OSRM Time and Segment OSRM Time
draw_histograms(agg_df_2['osrm_time'], agg_df_2['segment_osrm_time'], 'OSRM Time', 'Segment OSRM Time',
                'Distribution of OSRM Time and Segment OSRM Time', 'Time (minutes)', 'Frequency')
```



Insights:

- Visual analysis also proves that there is no difference between osrm_time and segment_osrm_time.
- Using Statistical methods, we can test whether the conclusion is correct or not.

```
# Draw QQ Plot for OSRM Time and Segment OSRM Time
draw_qq_plot(agg_df_2['osrm_time'], agg_df_2['segment_osrm_time'], 'QQ Plot for OSRM Time', 'QQ Plot for Segment OSRM Time')
```



Insights:

- From the graph, we can see that the both osrm_time and segment_osrm_time are not normally distributed.
- Will quantify the results using Shapiro-Wilk Test.

Shapiro-Wilk Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is normally distributed.
- Alternate Hypothesis: The data is not normally distributed.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['osrm_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['segment_osrm_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.4258338184867536
p-value: 1.1976397286640894e-111
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.5317562992725007
p-value: 2.4712371662349414e-106
The data is not normally distributed
```

```
# Levene's Test for Homogeneity of Variance
levene_test(agg_df_2['osrm_time'], agg_df_2['segment_osrm_time'])
```

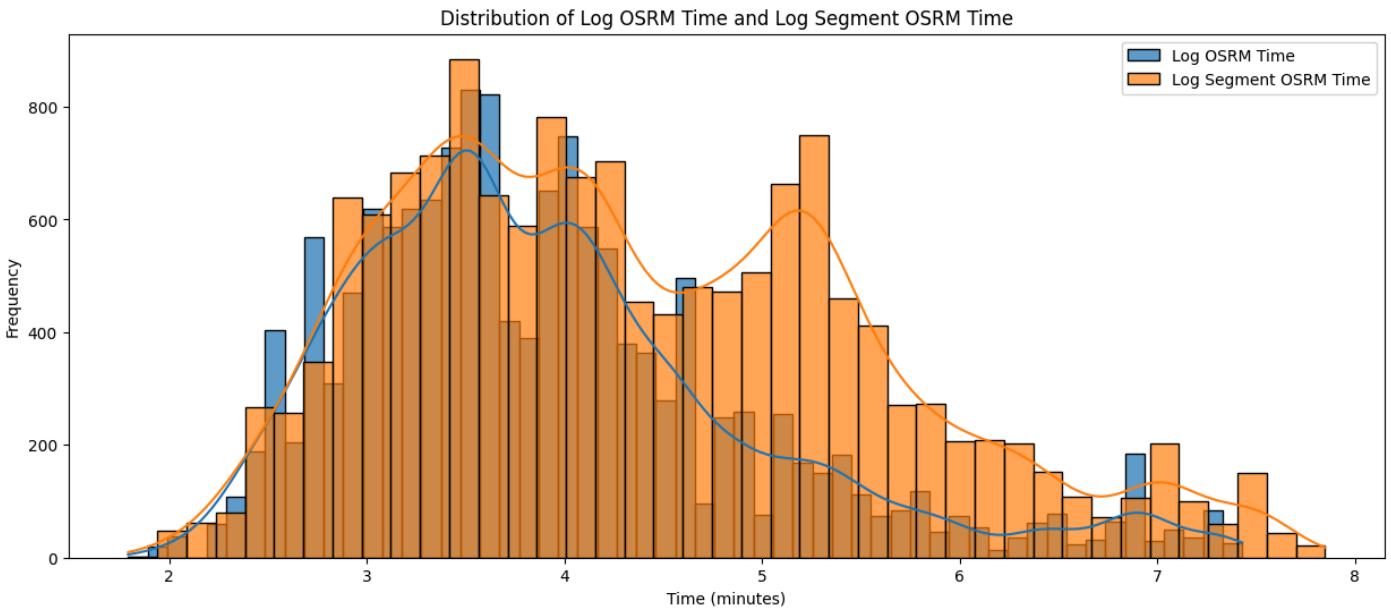
```
Levene's Test for Homogeneity of Variance
Statistics: 448.5894034544343
p-value: 7.939708063078011e-99
The variances are not equal
```

Insights:

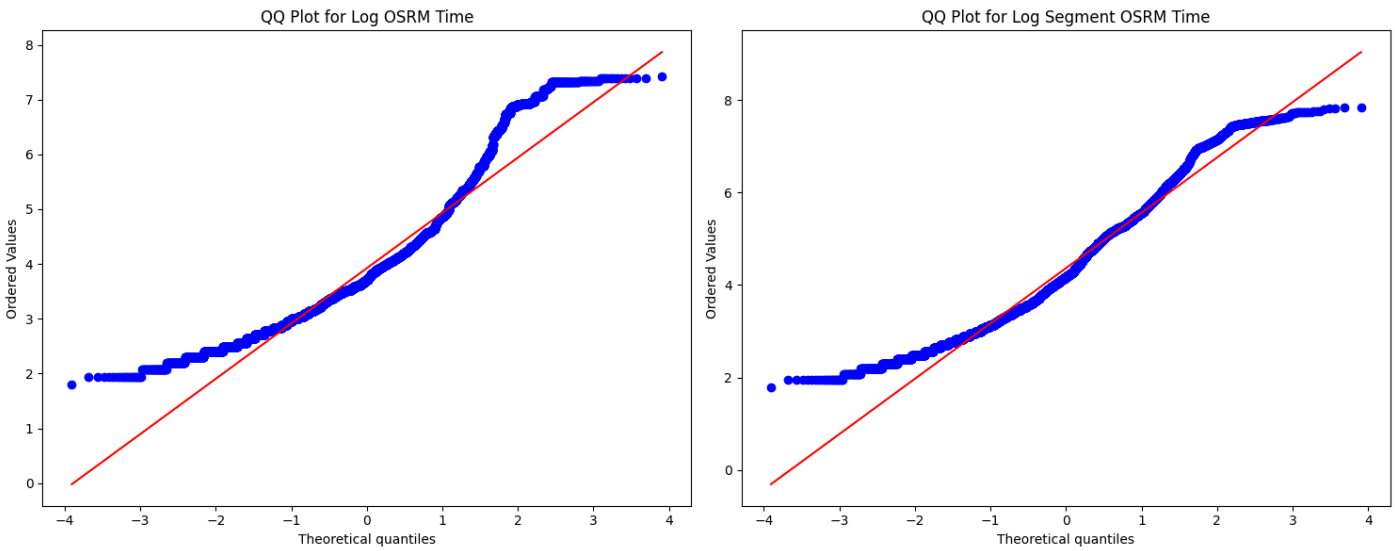
- Based on the statistical analysis, the data is not normally distributed.
- The data does have equal variance.
- Hence, we can't do the two-sample t-test.

Trying to convert the data to normal distribution using log transformation.

```
# Convert the osrm and segment osrm time into log normal distribution
convert_to_log_normal_and_visualize(agg_df_2, 'osrm_time', 'segment_osrm_time', 'Log OSRM Time', 'Log Segment OSRM Time', 'Distribution of Lo
```




```
# Draw QQ Plot for Log OSRM Time and Log Segment OSRM Time
draw_qq_plot(agg_df_2['log_osrm_time'], agg_df_2['log_segment_osrm_time'], 'QQ Plot for Log OSRM Time', 'QQ Plot for Log Segment OSRM Time')
```




```
# Shapiro-Wilk Test for Normality
shapiro_wilk_test(agg_df_2['log_osrm_time'])
print('-' * 50)
shapiro_wilk_test(agg_df_2['log_segment_osrm_time'])
```

```
Shapiro-Wilk Test for Normality
Statistics: 0.9302214562901787
p-value: 4.4552379315422194e-63
The data is not normally distributed
--------------------------------------------------
Shapiro-Wilk Test for Normality
Statistics: 0.9697449114894365
p-value: 1.0331165567871736e-47
The data is not normally distributed
```

Insights:

- Even after log transformation, the data is not normally distributed.
- Hence, we can't do the two-sample t-test.
- Will go with Kolmogorov-Smirnov Test.

```
# Drop the log transformed columns
agg_df_2.drop(['log_osrm_time', 'log_segment_osrm_time'], axis=1, inplace=True)
```

Kolmogorov-Smirnov Test:

Step 1: Define Null and Alternate Hypothesis

- Null Hypothesis: The data is from the same distribution.
- Alternate Hypothesis: The data is not from the same distribution.

Step 2: Decide the significance level

- Significance level is 0.05

Step 3: Calculate and Compare the p-value

- p-value < alpha: Reject the Null Hypothesis
- p-value > alpha: Fail to reject the Null Hypothesis

```
# 2 Sample KS Test for Non-Normal Distribution
ks_test(agg_df_2['osrm_time'], agg_df_2['segment_osrm_time'])
```

```
2 Sample KS Test for Non-Normal Distribution
Statistics: 0.2068032731453303
p-value: 8.103581900075691e-278
The distributions are different
```

Insights:

- Based on the statistical analysis, the data is from the same distribution.
- Hence, there is no difference between osrm_time and segment_osrm_time.

6. Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis

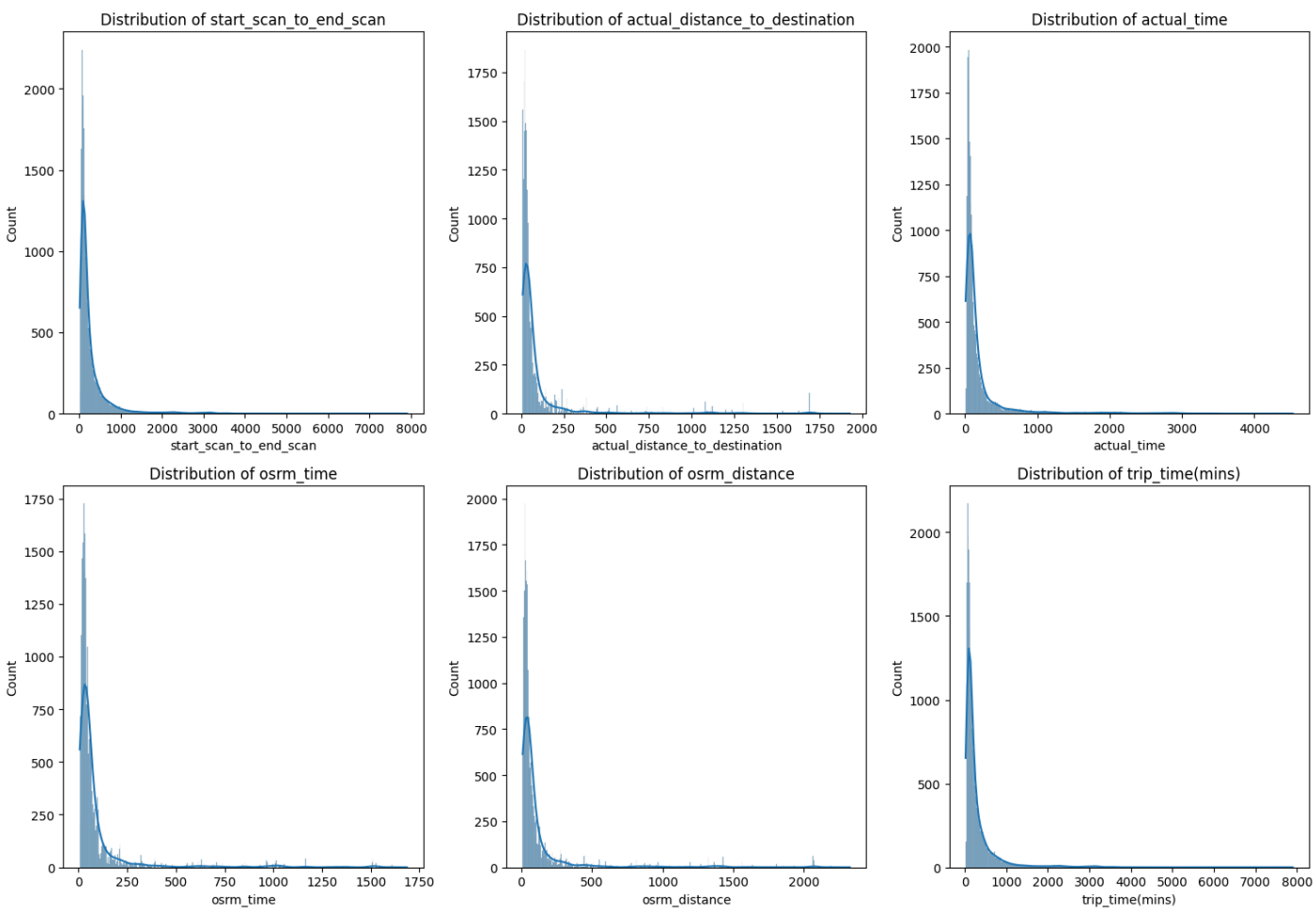
```
num_cols = agg_df.select_dtypes(include=['int64', 'float64']).columns.tolist()
agg_df[num_cols].describe().T
```

	count	mean	std	min	25%	50%	75%	max
start_scan_to_end_scan	26222.0	298.553390	441.116816	20.000000	90.000000	152.000000	307.000000	7898.000000
actual_distance_to_destination	26222.0	92.533054	209.952355	9.001351	21.654149	35.044329	65.557393	1927.447705
actual_time	26222.0	200.926588	385.730908	9.000000	51.000000	84.000000	167.000000	4532.000000
osrm_time	26222.0	90.785333	185.554359	6.000000	25.000000	39.000000	72.000000	1686.000000
osrm_distance	26222.0	114.975332	254.426468	9.072900	27.719150	43.543550	85.443950	2326.199100
trip_time(mins)	26222.0	299.107278	441.249287	20.702813	90.977759	152.336732	307.285979	7898.551955
trip_creation_day	26222.0	18.400351	7.892191	1.000000	14.000000	19.000000	25.000000	30.000000
trip_creation_month	26222.0	9.120815	0.325918	9.000000	9.000000	9.000000	9.000000	10.000000
trip_creation_week	26222.0	38.302952	0.965982	37.000000	38.000000	38.000000	39.000000	40.000000
trip_creation_hour	26222.0	12.880749	8.270751	0.000000	4.000000	16.000000	21.000000	23.000000

Insights:

- For all the numerical columns, Mean is greater than the Median.
- This indicates that the data is right skewed and outliers are present in the data.

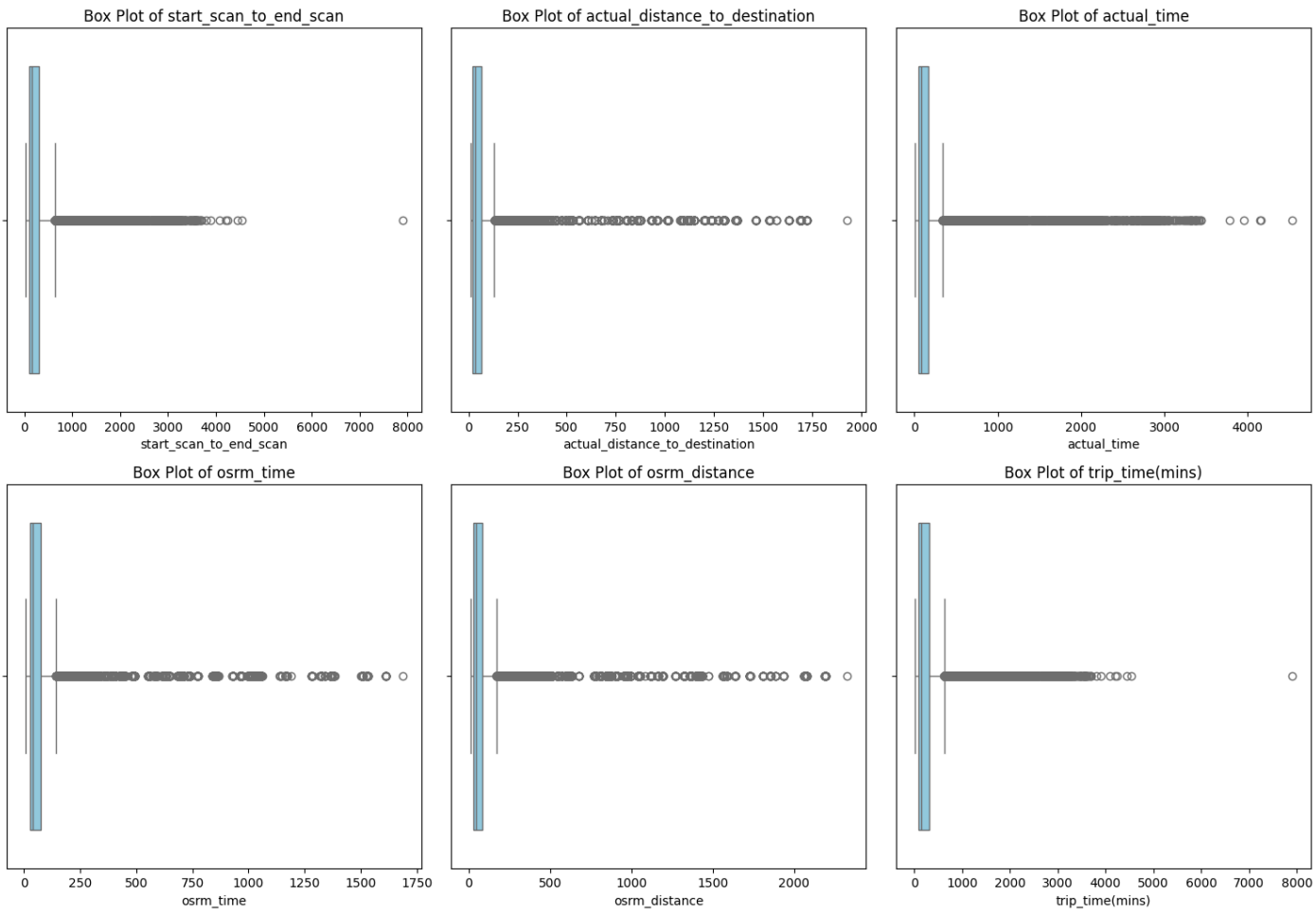
```
# Get numerical columns
num_cols = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'trip_time(mins)']

# Plotting the distribution of numerical columns
plt.figure(figsize=(15, 15))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(agg_df[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



Insights:

- None of the numerical columns are normally distributed.
- The columns are right skewed which means the tail is on the right side of the distribution.
- This indicates that the mean is greater than the median and outliers are present in the data.
- The outlier can be treated on further analysis.

```
# Box Plot of numerical columns
plt.figure(figsize=(15, 15))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(x=agg_df[col], color='skyblue')
    plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()
```



Insights:

- Box Plot proves the insights from the histogram.
- The box plot shows the presence of outliers in the data.

7. Handle the outliers using the IQR method.

```
# Detecting Outliers using IQR
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    print(f'Column: {data.name}')
    print(f'Q1: {Q1}')
    print(f'Q3: {Q3}')
    print(f'IQR: {IQR}')
    print(f'Lower Bound: {lower_bound}')
    print(f'Upper Bound: {upper_bound}')
    print(f'Number of Outliers: {outliers.shape[0]}')
    print('-' * 50)
    return outliers
```

```
# Detecting Outliers using IQR
```

```
outliers_iqr = agg_df[num_cols].apply(detect_outliers_iqr)
```

```
Column: start_scan_to_end_scan
Q1: 90.0
Q3: 307.0
IQR: 217.0
Lower Bound: -235.5
Upper Bound: 632.5
Number of Outliers: 2710
--------------------------------------------------
Column: actual_distance_to_destination
Q1: 21.654149220710377
Q3: 65.55739330743673
IQR: 43.903244086726346
Lower Bound: -44.20071690937914
Upper Bound: 131.41225943752625
Number of Outliers: 3272
--------------------------------------------------
Column: actual_time
Q1: 51.0
Q3: 167.0
IQR: 116.0
Lower Bound: -123.0
Upper Bound: 341.0
Number of Outliers: 3158
--------------------------------------------------
Column: osrm_time
Q1: 25.0
Q3: 72.0
IQR: 47.0
Lower Bound: -45.5
Upper Bound: 142.5
Number of Outliers: 2906
--------------------------------------------------
Column: osrm_distance
Q1: 27.71915
Q3: 85.44395
IQR: 57.7248
Lower Bound: -58.86805
Upper Bound: 172.03115
Number of Outliers: 3079
--------------------------------------------------
Column: trip_time(mins)
Q1: 90.97775862916666
Q3: 307.28597948333334
IQR: 216.30822085416668
Lower Bound: -233.4845726520834
Upper Bound: 631.7483107645834
Number of Outliers: 2720
--------------------------------------------------
```

Insights:

- Outliers in our sample data may be genuine.
- It's advisable to remove outliers only when there is a valid justification.
- Some outliers reflect natural variations within the population and should remain in the dataset.

8. Do one-hot encoding of categorical variables (like route_type)

```
# Previewing the categorical column - route_type
agg_df['route_type'].value_counts()
```

route_type	count
FTL	13798
Carting	12424

dtype: int64

```
# Performing One Hot Encoding for the column 'route_type'
label_encoder = LabelEncoder()
agg_df['route_type'] = label_encoder.fit_transform(agg_df['route_type'])

agg_df['route_type'].value_counts()
```

route_type	count
1	13798
0	12424

dtype: int64

Insights:

- As we can see from the results, FTL and Carting were converted into 1 and 0 encoding.

```
# Previewing the categorical column - data
agg_df['data'].value_counts()
```

data	count
training	18893
test	7329

dtype: int64

```
# Performing One Hot Encoding for the column 'data'
agg_df['data'] = label_encoder.fit_transform(agg_df['data'])

agg_df['data'].value_counts()
```

data	count
1	18893
0	7329

dtype: int64

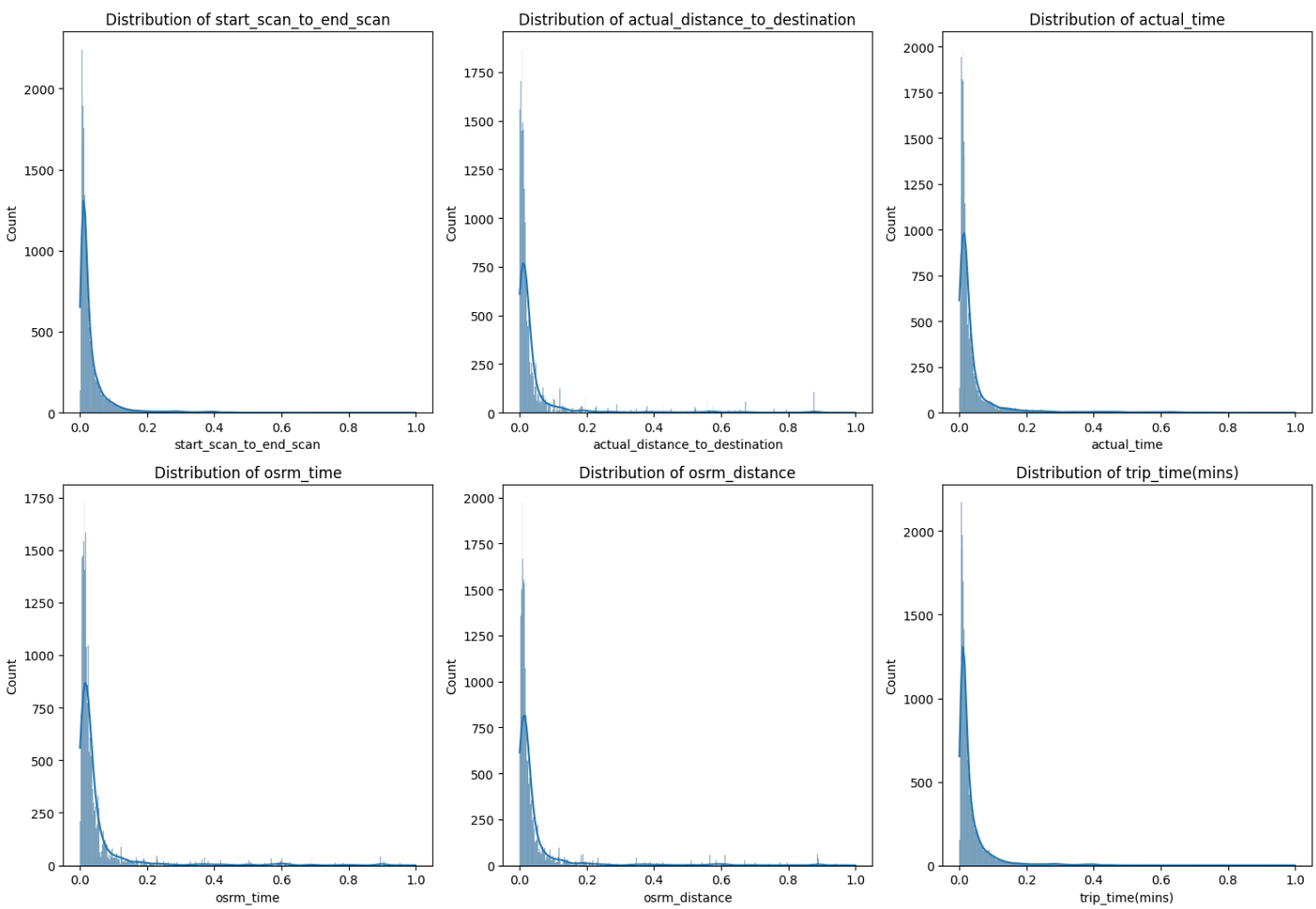
Insights:

- As we can see from the results, training and test data were converted into 1 and 0 encoding.

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

Normalization

```
# Performing Min-Max Scaler
back_up_df = agg_df.copy()
scaler = MinMaxScaler()
back_up_df[num_cols] = scaler.fit_transform(back_up_df[num_cols])

# Histogram of numerical columns after scaling
plt.figure(figsize=(15, 15))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(back_up_df[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

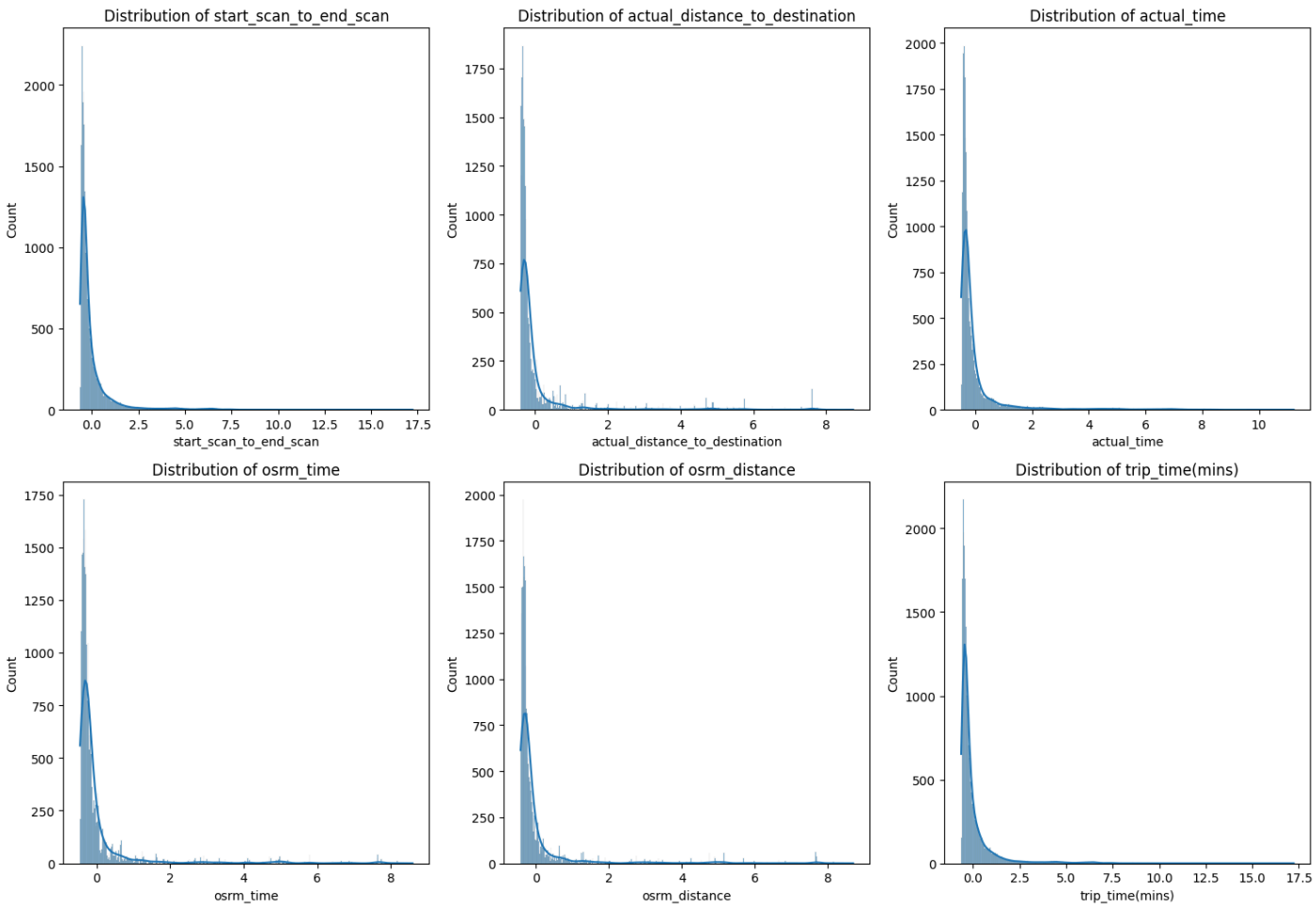


Insights:

- As we can see from the results, the numerical columns were normalized using MinMaxScaler.
- All the numerical columns were scaled between 0 and 1.

Standardization

```
# Perform Standardization
back_up_df = agg_df.copy()
scaler = StandardScaler()
back_up_df[num_cols] = scaler.fit_transform(back_up_df[num_cols])

# Histogram of numerical columns after standardization
plt.figure(figsize=(15, 15))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(back_up_df[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```




```
back_up_df.describe().T
```

	count	mean	min	25%	50%	75%	max	std
data	26222.0	0.720502	0.0	0.0	1.0	1.0	1.0	0.448761
route_type	26222.0	0.526199	0.0	0.0	1.0	1.0	1.0	0.499323
start_scan_to_end_scan	26222.0	-0.0	-0.631485	-0.472794	-0.332239	0.019149	17.22807	1.000019
actual_distance_to_destination	26222.0	-0.0	-0.397868	-0.337602	-0.273823	-0.128487	8.739838	1.000019
actual_time	26222.0	-0.0	-0.497575	-0.388689	-0.303136	-0.087956	11.228439	1.000019
osrm_time	26222.0	0.0	-0.456939	-0.354541	-0.27909	-0.101241	8.597185	1.000019
osrm_distance	26222.0	0.0	-0.416248	-0.342959	-0.280761	-0.116073	8.691179	1.000019
trip_time(mins)	26222.0	0.0	-0.630958	-0.471691	-0.332631	0.018536	17.222894	1.000019
trip_creation_date	26222	2018-09-22 00:35:41.713065216	2018-09-12 00:00:00	2018-09-17 00:00:00	2018-09-22 00:00:00	2018-09-27 00:00:00	2018-10-03 00:00:00	NaN
trip_creation_day	26222.0	18.400351	1.0	14.0	19.0	25.0	30.0	7.892191
trip_creation_month	26222.0	9.120815	9.0	9.0	9.0	9.0	10.0	0.325918
trip_creation_week	26222.0	38.302952	37.0	38.0	38.0	39.0	40.0	0.965982

Insights:

- As we can see from the results, the numerical columns were standardized using StandardScaler.

- All the numerical columns were scaled with mean 0 and standard deviation 1.

Business Insights:

- Timeframe: Data spans from '2018-09-12' to '2018-10-08'.
- Unique Entries:
  - 14787 unique trip IDs
  - 31 unique source states, 32 unique destination states
  - 1240 unique source cities, 1238 unique destination cities
- Data Split: Majority of data is for testing, not training.
- Common Route Type: Full Truck Load (FTL) is the most frequent transportation type.
- Missing Data: 10 unique source names are missing and 13 unique destination names are missing.
- Trip Patterns:
  - Trip volume peaks at 10 P.M., with increasing activity post-noon.
  - Week 38 sees the highest number of trips.
- Ordering Patterns:
  - Most orders are placed mid-month.
  - Key source states: Maharashtra, Karnataka, Tamil Nadu, Haryana, UP.
  - Key source cities: Bengaluru, Gurgaon, Bhiwandi, Mumbai, Chennai.
  - Key destination states: Karnataka, Maharashtra, Tamil Nadu, Haryana, Uttar Pradesh.
  - Key destination cities: Bengaluru, Mumbai, Gurgaon, Chennai, Hydrabad.
- Statistical Insights:
  - Similarity in time features: start_scan_to_end_scan and od_total_time; start_scan_to_end_scan and segment_actual_time.
  - Dissimilarity between actual_time & osrm_time, osrm_distance & segment_osrm_distance.
  - Both osrm_time and segment_osrm_time show significant differences.

Recommendations:

- Improve OSRM System: The OSRM trip planning system needs optimization. The discrepancies between the osrm_time and actual_time should be addressed to enhance delivery time accuracy and provide customers with more precise delivery expectations.
- Reduce Time Discrepancies: The difference between osrm_time and actual_time suggests inefficiencies. Reducing this gap will lead to better delivery time predictions and improve customer satisfaction.
- Address Distance Mismatch: The variation between the osrm_distance and the actual distance covered may indicate delivery personnel not following the predefined route or inaccuracies in OSRM route predictions. This could cause delays, so the team should investigate routing issues considering traffic, terrain, and other factors.
- Enhance Key Corridors: Most orders originate from or are destined for states like Maharashtra, Karnataka, Haryana, and Tamil Nadu. Strengthening existing logistics corridors in these regions can further improve delivery efficiency.
- Customer Profiling: Profiling customers in the states of Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh can help understand why most orders come from these areas and enhance the overall buying and delivery experience.
- Plan for Peak Seasons: Heavy traffic or challenging terrain in certain states could impact deliveries during peak festival seasons, requiring better planning to meet demand.