Business Case: Delhivery - Feature Engineering

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.

How can you help here?

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

• Clean, sanitize and manipulate data to get useful features out of raw fields

• Make sense out of the raw data and help the data science team to build forecasting models on it

Dataset

Dataset Link: Delhivery data

Column Profiling

The dataset at the heart of this exploration. Here are some of the key features:

- **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.
- **is_cutoff**: Unknown field.
- **cutoff_factor**: Unknown field.
- **cutoff_timestamp**: Unknown field.
- **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).
- **factor**: Unknown field.
- **segment_actual_time**: This is a segment time. Time taken by the subset of the package delivery.
- **segment_osrm_time**: This is the OSRM segment time. Time taken by the subset of the package delivery.
- **segment_osrm_distance**: This is the OSRM distance. Distance covered by the subset of the package delivery.
- **segment_factor**: Unknown field.

Concepts Used

- Feature Creation
- Relationship between Features
- Column Normalization /Column Standardization
- Handling categorical values
- Missing values - Outlier treatment / Types of outliers

How to begin:

Since delivery details of one package are divided into several rows (think of it as connecting flights to reach a particular destination). Now think about how we should treat their fields if we combine these rows? What aggregation would make sense if we merge. What would happen to the

numeric fields if we merge the rows?

Hint:

You can use inbuilt functions like `groupby` and aggregations like `sum()`, `cumsum()` to merge some rows based on their

1. `Trip_uuid`, `Source ID` and `Destination ID`
2. Further aggregate on the basis of just `Trip_uuid`. You can also keep the first and last values for some numeric/categorical fields if aggregating them won't make sense.

Basic data cleaning and exploration:

- Handle missing values in the data.
- Analyze the structure of the data.
- Try merging the rows using the hint mentioned above.
- Build some features to prepare the data for actual analysis. Extract features from the below fields:
  - **Destination Name**: Split and extract features out of destination. City-place-code (State)
  - **Source Name**: Split and extract features out of destination. City-place-code (State)
  - **Trip_creation_time**: Extract features like month, year and day etc

In-depth analysis and feature engineering:

- Calculate the time taken between `od_start_time` and `od_end_time` and keep it as a feature. Drop the original columns, if required
- Compare the difference between Point a. and `start_scan_to_end_scan`. Do hypothesis testing/ Visual analysis to check.
- 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`)
- 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`)
- 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`)
- 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`)
- Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis
- Handle the outliers using the **IQR method**.

- Do one-hot encoding of categorical variables (like `route_type`)
- Normalize/ Standardize the numerical features using `MinMaxScaler` or `StandardScaler`.

Evaluation Criteria (100 Points):

- Define Problem Statement and Perform Exploratory Data Analysis (10 points)
    - **Definition of Problem** (as per given problem statement with additional views)
    - **Observations** on:
        - Shape of data
        - Data types of all the attributes
        - Conversion of categorical attributes to 'category' (if required)
        - Missing value detection
        - Statistical summary
    - **Visual Analysis**:
        - Distribution plots of all the continuous variable(s)
        - Boxplots of all the categorical variables
    - **Insights** based on EDA
    - **Comments** on:
        - Range of attributes
        - Outliers of various attributes
        - Distribution of the variables and relationship between them
    - **Comments** for each univariate and bivariate plot
- Feature Creation (10 Points)

- Merging of Rows and Aggregation of Fields (10 Points)
- Comparison & Visualization of Time and Distance Fields (10 Points)
- Missing Values Treatment & Outlier Treatment (10 Points)
- Checking Relationship Between Aggregated Fields (10 Points)
- Handling Categorical Values (10 Points)
- Column Normalization / Column Standardization (10 Points)

- Business Insights (10 Points): Should include patterns observed in the data along with what you can infer from it. Examples:

- Check from where most orders are coming from (State, Corridor, etc.)
- Busiest corridor, average distance between them, average time taken

- Recommendations (10 Points) Actionable items for business. No technical jargon. No complications. Simple action items that everyone can understand.

Solution

Basic data cleaning and exploration:

In [864…
```
#importing the relevent libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [865…
```
#importing the csv file from google drive
!gdown 1ZkF2gGCDkjwQgOTGVBpsqhPpSGg1Fybb
```

```
Downloading...
From: https://drive.google.com/uc?id=1ZkF2gGCDkjwQgOTGVBpsqhPpSGg1Fybb
To: /content/07_delhivery_data.csv
100% 55.6M/55.6M [00:00<00:00, 86.3MB/s]
```

In [866…
```
#load the csv into dataframe
df = pd.read_csv('07_delhivery_data.csv')
```

In [867…
```
df.head(5)
```

Out[867…

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

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

5 rows × 24 columns

Dropping the unknown columns

In [868…

```
# all the columns which are marked as unknown in the column profiling we can remove them

unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']
df.drop(unknown_fields, axis = 1, inplace = True)
```

We can see that time is considered as object and float in no of columns

In [869…

```
df.head(5)
```

Out[869…

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

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

Analyzing the structure of data

In [870…

```
#checking Null values in the columns
df.isna().sum()
```

Out[870…

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0

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

dtype: int64

'Source name' and 'Destination name' have some missing values.

In [871…

```
#unique values in the columns
df.nunique()
```

Out[871…

	0
data	2
trip_creation_time	14817
route_schedule_uuid	1504
route_type	2
trip_uuid	14817
source_center	1508
source_name	1498
destination_center	1481
destination_name	1468
od_start_time	26369
od_end_time	26369
start_scan_to_end_scan	1915

	0
actual_distance_to_destination	144515
actual_time	3182
osrm_time	1531
osrm_distance	138046
segment_actual_time	747
segment_osrm_time	214
segment_osrm_distance	113799

dtype: int64

In [872…

```
# statistical summary of data - Numercial columns
df.describe()
```

Out[872…

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osr
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867
mean	961.262986	234.073372	416.927527	213.868272	284.771297	36.196111	18
std	1037.012769	344.990009	598.103621	308.011085	421.119294	53.571158	14
min	20.000000	9.000045	9.000000	6.000000	9.008200	-244.000000	0
25%	161.000000	23.355874	51.000000	27.000000	29.914700	20.000000	11
50%	449.000000	66.126571	132.000000	64.000000	78.525800	29.000000	17
75%	1634.000000	286.708875	513.000000	257.000000	343.193250	40.000000	22
max	7898.000000	1927.447705	4532.000000	1686.000000	2326.199100	3051.000000	1611

In [873…

```
#statistical summary of data - categorical columns
df.describe(include = object)
```

Out[873…

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_ce
count	144867	144867	144867	144867	144867	144867	144574	14
unique	2	14817	1504	2	14817	1508	1498	
top	training	2018-09-28 05:23:15.359220	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153811219535896559	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND00000(
freq	104858	101	1812	99660	101	23347	23347	1

In [874…

```
#shape of data
df.shape
```

Out[874…

```
(144867, 19)
```

There are 19 columns and 144867 rows in the dataset.

In [875…

```
#checking the datatype of columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 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  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                   144867 non-null  float64
```

```
 14  osrm_time                     144867 non-null  float64
 15  osrm_distance                 144867 non-null  float64
 16  segment_actual_time           144867 non-null  float64
 17  segment_osrm_time             144867 non-null  float64
 18  segment_osrm_distance         144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

Changing he datatype of columns

In [876…

```
#converting the data and route_type as categorical columns

df["data"] = df["data"].astype("category")
df["route_type"] = df["route_type"].astype("category")
```

In [877…

```
# There are few columns which represent time but they in object datatype format those can be convereted to datetime col

datetype_column = ['trip_creation_time', 'od_start_time', 'od_end_time']

for i in datetype_column:
  df[i] = pd.to_datetime(df[i]) #passing the column one by one in the for loop
```

In [878…

```
#checking the column datatypes again
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   data                          144867 non-null  category
 1   trip_creation_time            144867 non-null  datetime64[ns]
 2   route_schedule_uuid           144867 non-null  object
 3   route_type                    144867 non-null  category
 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  datetime64[ns]
```

```
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 19.1+ MB
```

Handling missing values

In [879…

```
# checking for the source_center  for which source_name is null

center_for_missing_source_name = df[df['source_name'].isna()]['source_center'].unique()
center_for_missing_source_name
```

Out[879…

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

This give us all the source_center names for which source_name is not available. Let's check if these source_center name is having source_name

In [880…

```
# Checking if we can get the source_name for above source_centers from other rows in the data
# In short, it's selecting rows where "source_name" has a value, but the "source_center" is associated with some other

df[(df['source_name'].notnull()) & (df['source_center'].isin(df[df['source_name'].isnull()]))]
```

Out[880…

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_sta

It means that source_name for missing values is not available in other rows as well.

In [881…

```
#checking for destination_center for which destination_name is null
center_for_missing_destination_name = df[df['destination_name'].isna()]['destination_center'].unique()
center_for_missing_destination_name
```

Out[881…
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [882…
```
# Checking if we can get the destintion_name for above destinatio_centers from other rows in the data
# In short, it's selecting rows where "destination_name" has a value, but the "destination_center" is associated with s

df[(df['destination_name'].notnull()) & (df['destination_center'].isin(df[df['destination_name'].isnull()]))]
```

Out[882…

data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_sta

Removing incorrect data

It means that destination_name for missing values is not available in other rows as well.

In [883…
```
# We can see from the describe that the values of segment actual time is negative, which can't be true so lets drop tha
df.drop(df[df['segment_actual_time']<0].index, inplace = True)
```

In [884…
```
#now checking the describe data
df.describe()
```

Out[884…

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time	
count	144846	144846	144846	144846.000000	144846.000000	144846.000000	1448
mean	2018-09-22 13:34:27.259366400	2018-09-22 18:02:50.434589952	2018-09-23 10:04:33.787580160	961.226537	234.057171	416.908724	
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.000045	9.000000	
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000	23.354927	51.000000	
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:52:50.639791104	2018-09-23 03:13:03.520212992	449.000000	66.126234	132.000000	
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000	286.706673	513.000000	

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time	
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.447705	4532.000000	1(
std	NaN	NaN	NaN	1036.993595	344.974984	598.085058	

In [885…

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

Out[885…

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0

	0
segment_osrm_distance	0

dtype: int64

Now negative values of negative segment time are removed.

Merging rows and aggregaion of data

In [886…
```
df.head(5)
```

Out[886…

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

Merging row based on trip_uuid, source center and destination center as df1

In [887…
```
# name of columns for aggregation
group_by_columns = ['trip_uuid', 'source_center', 'destination_center']

df1 = df.groupby(by = group_by_columns, as_index= False ).agg(
                                                        {
                                                            'data' : 'first',
```

```
                                                    'trip_creation_time' : 'first',
                                                    'route_type': 'first',
                                                    'source_name' : 'first',
                                                    'destination_name' : 'last',
                                                    'od_start_time' : 'first',
                                                    'od_end_time' : 'first',
                                                    'start_scan_to_end_scan' : 'first',
                                                    'actual_distance_to_destination' : 'last', #cumul
                                                    'actual_time' : 'last', #cumulative time for trip
                                                    'osrm_time' : 'last', #cumualative time
                                                    'osrm_distance' : 'last', #cumulative distance so
                                                    'segment_actual_time' : 'sum', #segment time so d
                                                    'segment_osrm_time' : 'sum', #segment time so doi
                                                    'segment_osrm_distance' : 'sum' #segment distance


                                                }
                                            )
df1.head()
```

Out[887…

	trip_uuid	source_center	destination_center	data	trip_creation_time	route_type	source_name	destination_nar
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	2018-09-12 00:00:16.535741	FTL	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bilaspur_H (Haryar
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	2018-09-12 00:00:16.535741	FTL	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H (Uttar Prades
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	2018-09-12 00:00:22.886430	Carting	Doddablpur_ChikaDPP_D (Karnataka)	Chikblapur_ShntiSgr (Karnatak
3	trip-153671042288605164	IND572101AAA	IND561203AAB	training	2018-09-12 00:00:22.886430	Carting	Tumkur_Veersagr_I (Karnataka)	Doddablpur_ChikaDPP (Karnatak
4	trip-153671043369099517	IND000000ACB	IND160002AAC	training	2018-09-12 00:00:33.691250	FTL	Gurgaon_Bilaspur_HB (Haryana)	Chandigarh_Mehmdpur (Punja

Adding the od_total_time

Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

In [888…

```
#calculating the total trip time
df1['od_total_time'] = df1['od_end_time'] - df1['od_start_time']
```

```
#dropping the original columns from the dataframe
df1.drop(['od_start_time', 'od_end_time'], inplace = True, axis = 1)

#convert the total time into seconds
df1['od_total_time'] = ((df1['od_total_time'].dt.total_seconds())/ 60).round(2)

df1.head(2) #check the value in new column for total trip time
```

Out[888…

	trip_uuid	source_center	destination_center	data	trip_creation_time	route_type	source_name	destination_name	star
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	2018-09-12 00:00:16.535741	FTL	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bilaspur_HB (Haryana)	
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	2018-09-12 00:00:16.535741	FTL	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H_6 (Uttar Pradesh)	

Merging row based on trip_uuid as df2

In [889…

```
df2 = df1.groupby(by = 'trip_uuid', as_index = False).agg({'source_center' : 'first',
                                                           'destination_center' : 'last',
                                                           'data' : 'first',
                                                           'route_type' : 'first',
                                                           'trip_creation_time' : 'first',
                                                           'source_name' : 'first',
                                                           'destination_name' : 'last',
                                                           'od_total_time' : 'sum', #now doing sum after first aggregat
                                                           'start_scan_to_end_scan' : 'sum',
                                                           'actual_distance_to_destination' : 'sum',
                                                           'actual_time' : 'sum',
                                                           'osrm_time' : 'sum',
                                                           'osrm_distance' : 'sum',
                                                           'segment_actual_time' : 'sum',
                                                           'segment_osrm_time' : 'sum',
                                                           'segment_osrm_distance' : 'sum'})
df2.head()
```

Out[889…

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name	destination_nam
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpur_Central_H_ (Uttar Pradesh

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name	destination_nam
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpur_ChikaDPP_ (Karnataka
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgaon_Bilaspur_H (Haryana
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mumbai_MiraRd_I (Maharashtra
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_WrdN1DPP_ (Karnataka

Feature Generation

Extract state, City and Place from Source & Destination

In [890…
```
("Kanpur_Central_H_6 (Uttar Pradesh)").split('_')
```

Out[890…
```
['Kanpur', 'Central', 'H', '6 (Uttar Pradesh)']
```

In [891…
```
("Kanpur_Central_H_6 (Uttar Pradesh)").split('_')[0] #city
```

Out[891…
```
'Kanpur'
```

In [892…
```
("Kanpur_Central_H_6 (Uttar Pradesh)").split('_')[1] #place
```

Out[892…
```
'Central'
```

In [893…
```
("Kanpur_Central_H_6 (Uttar Pradesh)").split('(')[1]
```

Out[893…
```
'Uttar Pradesh)'
```

In [894…
```
("Kanpur_Central_H_6 (Uttar Pradesh)").split('(')[1][:-1]  #state
```

Out[894…
```
'Uttar Pradesh'
```

In [895…
```
#we will be using the df2 dataframe that is been grouped at trip_uuid level

# Extracting state, city and place name from source_name column value e.g. Kanpur_Central_H_6 (Uttar Pradesh)

# Source Name: Split and extract features out of destination. City-place-code (State)
df2['source_state'] = df2['source_name'].apply(lambda x : str(x).split('(')[1][:-1] if '(' in str(x) else None) #state
df2['source_city'] = df2['source_name'].apply(lambda x : str(x).split('_')[0]) #city
#df2['source_place'] = df2['source_name'].apply(lambda x : str(x).split('_')[1]) #place

# Destination Name: Split and extract features out of destination. City-place-code (State)
df2['destination_state'] = df2['destination_name'].apply(lambda x : str(x).split('(')[1][:-1] if '(' in str(x) else Non
df2['destination_city'] = df2['destination_name'].apply(lambda x : str(x).split('_')[0]) #city
#df2['destination_place'] = df2['destination_name'].apply(lambda x : str(x).split('_')[1]) #place

df2.head()
```

Out[895…

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name	destination_nam
0	trip-153671041653548748	IND209304AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpur_Central_H_ (Uttar Pradesh
1	trip-153671042288605164	IND561203AAB	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpur_ChikaDPP_ (Karnataka
2	trip-153671043369099517	IND000000ACB	IND000000ACB	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgaon_Bilaspur_H (Haryana
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mumbai_MiraRd_I (Maharashtra
4	trip-153671052974046625	IND583101AAA	IND583119AAA	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_WrdN1DPP_ (Karnataka

5 rows × 21 columns

In [896…
```
df2['source_state'].unique()
```

Out[896…
```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
```

```
       'Assam', 'Madhya Pradesh', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Goa', 'Jharkhand', 'Pondicherry',
       'Orissa', 'Uttarakhand', 'Himachal Pradesh', 'Kerala',
       'Arunachal Pradesh', 'Bihar', 'Chhattisgarh',
       'Dadra and Nagar Haveli', 'Jammu & Kashmir', 'Mizoram', 'Nagaland',
      None], dtype=object)
```

In [897…

```
df2['destination_state'].unique()
```

Out[897…

```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
       'Madhya Pradesh', 'Assam', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Dadra and Nagar Haveli', 'Orissa',
       'Bihar', 'Jharkhand', 'Goa', 'Uttarakhand', 'Himachal Pradesh',
       'Kerala', 'Arunachal Pradesh', 'Mizoram', 'Chhattisgarh',
       'Jammu & Kashmir', 'Nagaland', 'Meghalaya', 'Tripura', None,
       'Daman & Diu'], dtype=object)
```

State names for source and destination looks good

In [898…

```
df2['source_city'].unique()[:50]
```

Out[898…

```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai Hub (Maharashtra)',
       'Bellary', 'Chennai', 'HBR Layout PC (Karnataka)', 'Surat',
       'Delhi', 'Pune', 'FBD', 'Shirala', 'Hyderabad', 'Thirumalagiri',
       'Gulbarga', 'Jaipur', 'Allahabad', 'Guwahati', 'Narsinghpur',
       'Shrirampur', 'Hoogly', 'Madakasira', 'Sonari', 'Bengaluru',
       'Dindigul', 'Jalandhar', 'Faridabad', 'Chandigarh', 'Deoli',
       'Pandharpur', 'CCU', 'Bhandara', 'Kurnool', 'Bhiwandi', 'Bhatinda',
       'RoopNagar', 'Bantwal', 'Lalru', 'Kadi', 'Shahdol', 'Gangakher',
       'Durgapur', 'Vapi', 'Jamjodhpur', 'Jetpur', 'Mehsana', 'Jabalpur',
       'Junagadh', 'Gundlupet', 'Mysore'], dtype=object)
```

In [899…

```
df2['destination_city'].unique()[:50]
```

Out[899…

```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Sandur', 'Chennai',
       'HBR Layout PC (Karnataka)', 'Surat', 'Delhi',
       'PNQ Rahatani DPC (Maharashtra)', 'Faridabad (Haryana)',
       'Ratnagiri', 'Bangalore', 'Hyderabad', 'Aland', 'Jaipur', 'Satna',
       'Janakpuri (Delhi)', 'Guwahati', 'Bareli', 'Nashik', 'Hooghly',
       'Puttaprthi', 'Sivasagar', 'Bengaluru', 'Palani', 'Jalandhar',
       'Chandigarh', 'Yavatmal', 'Sangola', 'Kolkata', 'Savner',
```

```
       'Kurnool', 'FBD', 'Bhatinda', 'Bhiwandi', 'Barnala', 'Murbad',
       'Kadaba', 'Gulbarga', 'Naraingarh', 'Ludhiana', 'Kadi', 'Jabalpur',
       'MAA', 'Gangakher', 'Bankura', 'Silvassa', 'Porbandar', 'Jetpur'],
      dtype=object)
```

With the city name there is some problem, at some places name of state is added with it.

In [900…

```
df2['source_city'] = df2['source_city'].apply(lambda x : str(x).split()[0])
df2['destination_city'] = df2['destination_city'].apply(lambda x : str(x).split()[0])
df2.sample(10)
```

Out[900…

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name	destination_
6105	trip-153743386446319660	IND302014AAA	IND302014AAA	training	FTL	2018-09-20 08:57:44.463490	Jaipur_Hub (Rajasthan)	Jaipur_Hub (Raja
11614	trip-153814691500548801	IND500039AAC	IND501359AAE	test	Carting	2018-09-28 15:01:55.005735	Hyderabad_Uppal_I (Telangana)	Hyderabad_Shamsł (Telar
2629	trip-153702499222437553	IND411033AAA	IND000000AAL	training	Carting	2018-09-15 15:23:12.224639	Pune_Tathawde_H (Maharashtra)	Pune_PC (Mahara
1350	trip-153687458010856334	IND842001AAA	IND845438AAA	training	FTL	2018-09-13 21:36:20.108791	Muzaffrpur_Bbganj_I (Bihar)	Bettiah_Bypass (
8180	trip-153768266515063150	IND140301AAA	IND160055AAA	training	Carting	2018-09-23 06:04:25.150896	Chandigarh_Kharar_DC (Chandigarh)	Chandigarh_I (Chand
11031	trip-153806673407967942	IND811399AAA	IND110044AAB	test	Carting	2018-09-27 16:45:34.079932	Noida_Sector02_C (Uttar Pradesh)	Del_Okhla_PC (
11881	trip-153817712313897848	IND421302AAG	IND400102AAC	test	Carting	2018-09-28 23:25:23.139238	Bhiwandi_Mankoli_HB (Maharashtra)	Mumbai_Jogs (Mahara
13694	trip-153844549233368752	IND411033AAA	IND411021AAA	test	Carting	2018-10-02 01:58:12.333957	Pune_Tathawde_H (Maharashtra)	PNQ Pasha (Mahara
12817	trip-153833102774531581	IND580028AAA	IND587103AAB	test	FTL	2018-09-30 18:10:27.745704	Hubli_Adargchi_IP (Karnataka)	Bagalkot_Sect (Karn
5220	trip-153732541288726824	IND000000ACB	IND110064AAA	training	Carting	2018-09-19 02:50:12.887540	Gurgaon_Bilaspur_HB (Haryana)	Delhi_Mayapu (

10 rows × 21 columns

Now the city data is also clean. So we are good for some EDA analysis.

In [901…

```
# Replace 'Bangalore' with 'Bengaluru' in the 'city' column
df2['source_city'] = df2['source_city'].replace('Bangalore', 'Bengaluru')
df2['destination_city'] = df2['destination_city'].replace('Bangalore', 'Bengaluru')
```

Extract Date, year, month, day, hour and week from trip creation time

In [902…

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14817 non-null  object
 1   source_center                   14817 non-null  object
 2   destination_center              14817 non-null  object
 3   data                            14817 non-null  category
 4   route_type                      14817 non-null  category
 5   trip_creation_time              14817 non-null  datetime64[ns]
 6   source_name                     14807 non-null  object
 7   destination_name                14809 non-null  object
 8   od_total_time                   14817 non-null  float64
 9   start_scan_to_end_scan          14817 non-null  float64
 10  actual_distance_to_destination  14817 non-null  float64
 11  actual_time                     14817 non-null  float64
 12  osrm_time                       14817 non-null  float64
 13  osrm_distance                   14817 non-null  float64
 14  segment_actual_time             14817 non-null  float64
 15  segment_osrm_time               14817 non-null  float64
 16  segment_osrm_distance           14817 non-null  float64
 17  source_state                    14807 non-null  object
 18  source_city                     14817 non-null  object
 19  destination_state               14809 non-null  object
 20  destination_city                14817 non-null  object
dtypes: category(2), datetime64[ns](1), float64(9), object(9)
memory usage: 2.2+ MB
```

In [903…

```
# since 'trip_creation_time' is already a datetime column so we don't need to use the pd.to_datetime here again.
```

```
df2['trip_creation_date'] = df2['trip_creation_time'].dt.date
df2['trip_creation_year'] = df2['trip_creation_time'].dt.year.astype('int16')
df2['trip_creation_month'] = df2['trip_creation_time'].dt.month.astype('int8')
df2['trip_creation_day'] = df2['trip_creation_time'].dt.day.astype('int8')
df2['trip_creation_week'] = df2['trip_creation_time'].dt.isocalendar().week.astype('int8')
df2['trip_creation_hour'] = df2['trip_creation_time'].dt.hour.astype('int8')

df2.sample(5)
```

Out[903…

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time	source_name	destinat
7063	trip-153756316613557033	IND560300AAA	IND560043AAC	training	Carting	2018-09-21 20:52:46.135830	Bengaluru_KGAirprt_HB (Karnataka)	HBR Layout PC (
13489	trip-153842720855712707	IND110044AAB	IND131028AAB	test	Carting	2018-10-01 20:53:28.557380	Del_Okhla_PC (Delhi)	Sonipa
216	trip-153673138564024513	IND411021AAA	IND411033AAA	training	Carting	2018-09-12 05:49:45.640498	PNQ Pashan DPC (Maharashtra)	Pune_T (Ma
4324	trip-153722834736048396	IND614620AAA	IND623407AAA	training	Carting	2018-09-17 23:52:27.360849	Manamelkudi_TmpleSrt_D (Tamil Nadu)	Thiruvadanai_Ra (T
10844	trip-153802658315990131	IND421302AAG	IND400072AAA	test	Carting	2018-09-27 05:36:23.160178	Bhiwandi_Mankoli_HB (Maharashtra)	Mumba (Ma

5 rows × 27 columns

In [904…

```
df2.describe().T
```

Out[904…

	count	mean	min	25%	50%	75%	
trip_creation_time	14817	2018-09-22 12:44:19.555167744	2018-09-12 00:00:16.535741	2018-09-17 02:51:25.129125888	2018-09-22 04:02:35.066945024	2018-09-27 19:37:41.898427904	2018 23:59:42
od_total_time	14817.0	531.69763	23.46	149.93	280.77	638.2	
start_scan_to_end_scan	14817.0	530.810016	23.0	149.0	280.0	637.0	
actual_distance_to_destination	14817.0	164.477838	9.002461	22.837239	48.474072	164.583208	2186
actual_time	14817.0	357.143754	9.0	67.0	149.0	370.0	
osrm_time	14817.0	161.384018	6.0	29.0	60.0	168.0	

	count	mean	min	25%	50%	75%	
osrm_distance	14817.0	204.344689	9.0729	30.8192	65.6188	208.475	2
segment_actual_time	14817.0	353.95161	9.0	66.0	147.0	367.0	
segment_osrm_time	14817.0	180.921172	6.0	31.0	65.0	185.0	
segment_osrm_distance	14817.0	223.164	9.0729	32.6545	70.1544	218.7102	35
trip_creation_year	14817.0	2018.0	2018.0	2018.0	2018.0	2018.0	
trip_creation_month	14817.0	9.120672	9.0	9.0	9.0	9.0	
trip_creation_day	14817.0	18.37079	1.0	14.0	19.0	25.0	
trip_creation_week	14817.0	38.295944	37.0	38.0	38.0	39.0	
trip_creation_hour	14817.0	12.449821	0.0	4.0	14.0	20.0	

In [905…

```
df2.describe(include = 'object').T
```

Out[905…

	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
source_center	14817	938	IND000000ACB	1063
destination_center	14817	1042	IND000000ACB	821
source_name	14807	933	Gurgaon_Bilaspur_HB (Haryana)	1063
destination_name	14809	1034	Gurgaon_Bilaspur_HB (Haryana)	821
source_state	14807	29	Maharashtra	2714
source_city	14817	716	Bengaluru	1700
destination_state	14809	31	Maharashtra	2561
destination_city	14817	840	Bengaluru	1639
trip_creation_date	14817	22	2018-09-18	791

Univariate Analysis

Top-10 Source states

In [906…

```
top_source_states = df2['source_state'].value_counts(normalize = True).head(10)* 100
top_source_states
```

Out[906…

source_state	proportion
Maharashtra	18.329169
Karnataka	14.472884
Haryana	12.413048
Tamil Nadu	7.016951
Telangana	5.301547
Uttar Pradesh	5.146215
Gujarat	5.065172
Delhi	4.916594
West Bengal	4.491119
Punjab	3.619910

dtype: float64

Maharashtra, karnataka and Haryana contributes to 44% to total trip as source states followed by Tamilnadu and Telangana.

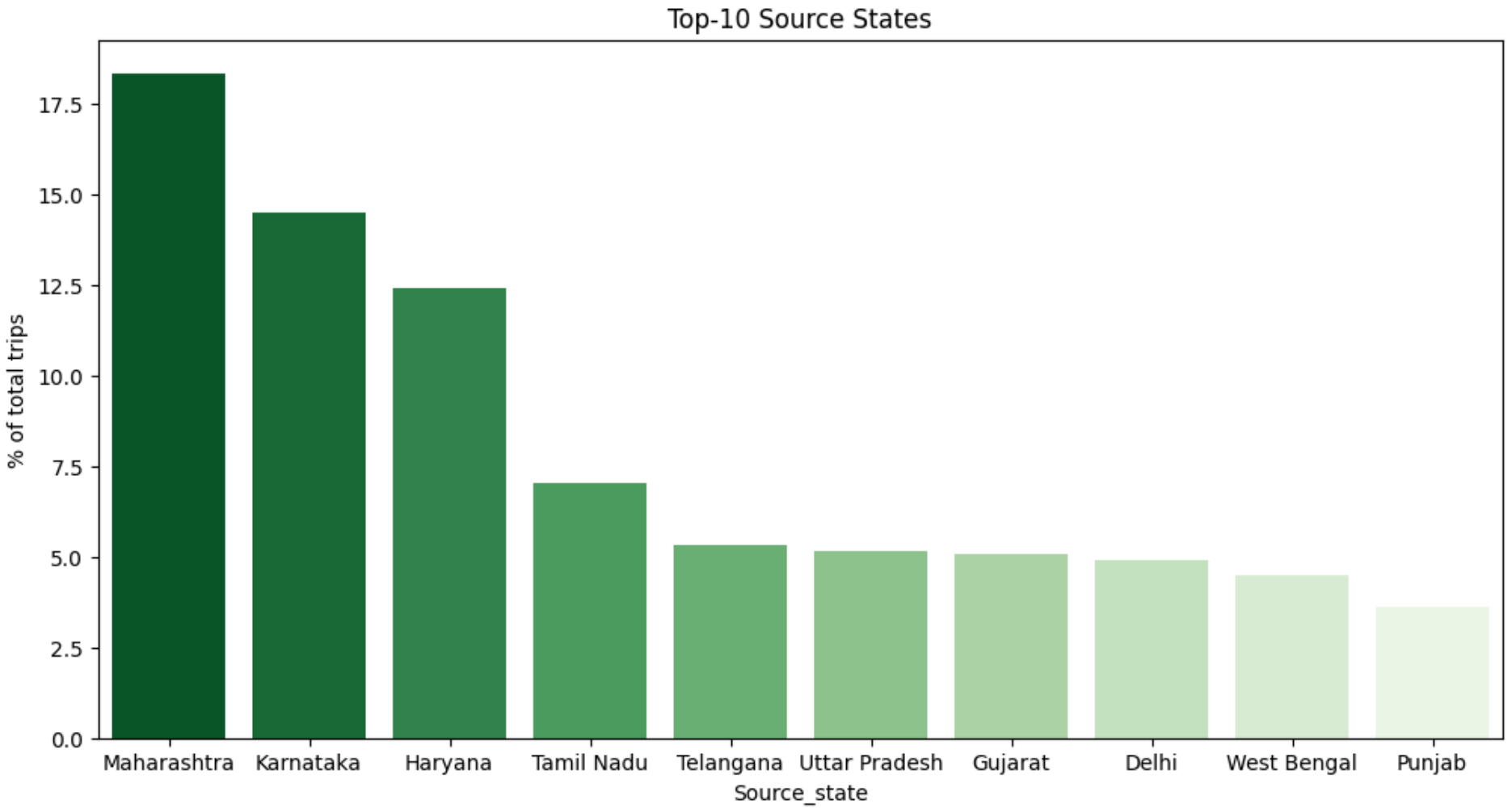
In [907…

```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_source_states.index, y = top_source_states.values, palette='Greens_r')
plt.xlabel('Source_state')
plt.ylabel('% of total trips')
plt.title('Top-10 Source States')
plt.show()
```

```
<ipython-input-907-8dcee287d63c>:3: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
` and set `legend=False` for the same effect.

  sns.barplot(x = top_source_states.index, y = top_source_states.values, palette='Greens_r')
```



Top-10 Destination States

In [908…

```
#getting top-10 states and their counts
top_destination_states = df2['destination_state'].value_counts(normalize = True).head(10) * 100
top_destination_states
```

Out[908…

destination_state	proportion
Maharashtra	17.293538
Karnataka	15.497333
Haryana	11.094605
Tamil Nadu	7.319873
Uttar Pradesh	5.530421
Telangana	5.294078
Gujarat	4.956445
West Bengal	4.706597
Delhi	4.409481
Punjab	4.166385

dtype: float64

Mumbai, karnataka and Haryana contributes to 45% to total trip as destination states followed by Tamilnadu and UttarPradesh.

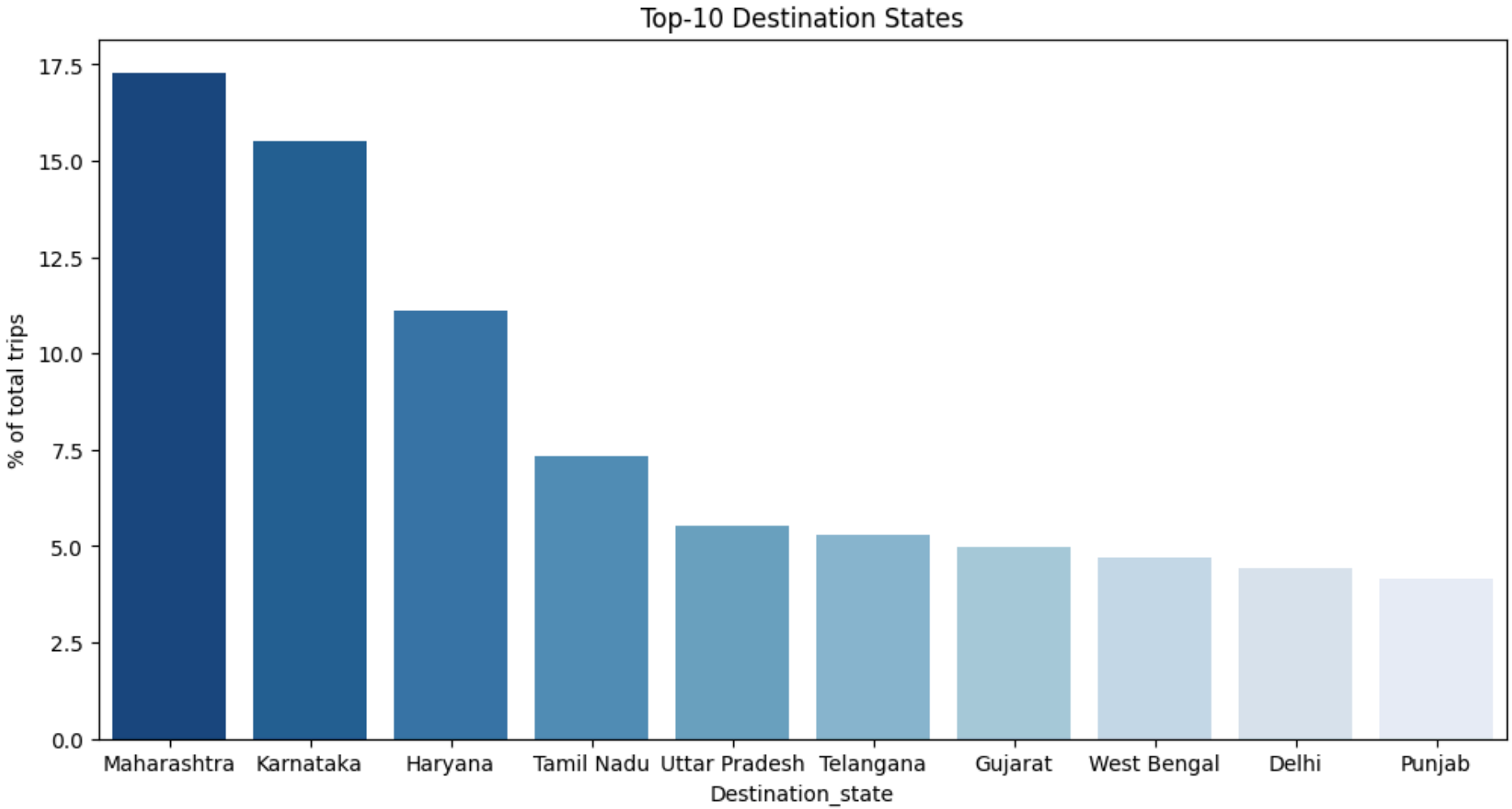
In [909…

```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_destination_states.index, y = top_destination_states.values, palette='Blues_r')
plt.xlabel('Destination_state')
plt.ylabel('% of total trips')
plt.title('Top-10 Destination States')
plt.show()
```

```
<ipython-input-909-f071c97f940b>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
` and set `legend=False` for the same effect.

  sns.barplot(x = top_destination_states.index, y = top_destination_states.values, palette='Blues_r')
```



Top-10 Source Cities

In [910…

```
#getting top-10 cities and their counts
top_source_cities = df2['source_city'].value_counts(normalize = True).head(10)*100
top_source_cities
```

Out[910…

source_city	proportion
Bengaluru	11.473308
Gurgaon	7.687116
Mumbai	6.533036
Bhiwandi	4.704056
Delhi	3.718701
Hyderabad	3.475737
Pune	3.239522
Chennai	2.281164
Kolkata	1.896470
Sonipat	1.862725

dtype: float64

In [100…

```
df2['source_city'].nunique()
```

Out[100…

716

Bengaluru, Gurgaon, Mumbai, Bhiwandi and Delhi are Top-5 source states for trips.

In [911…

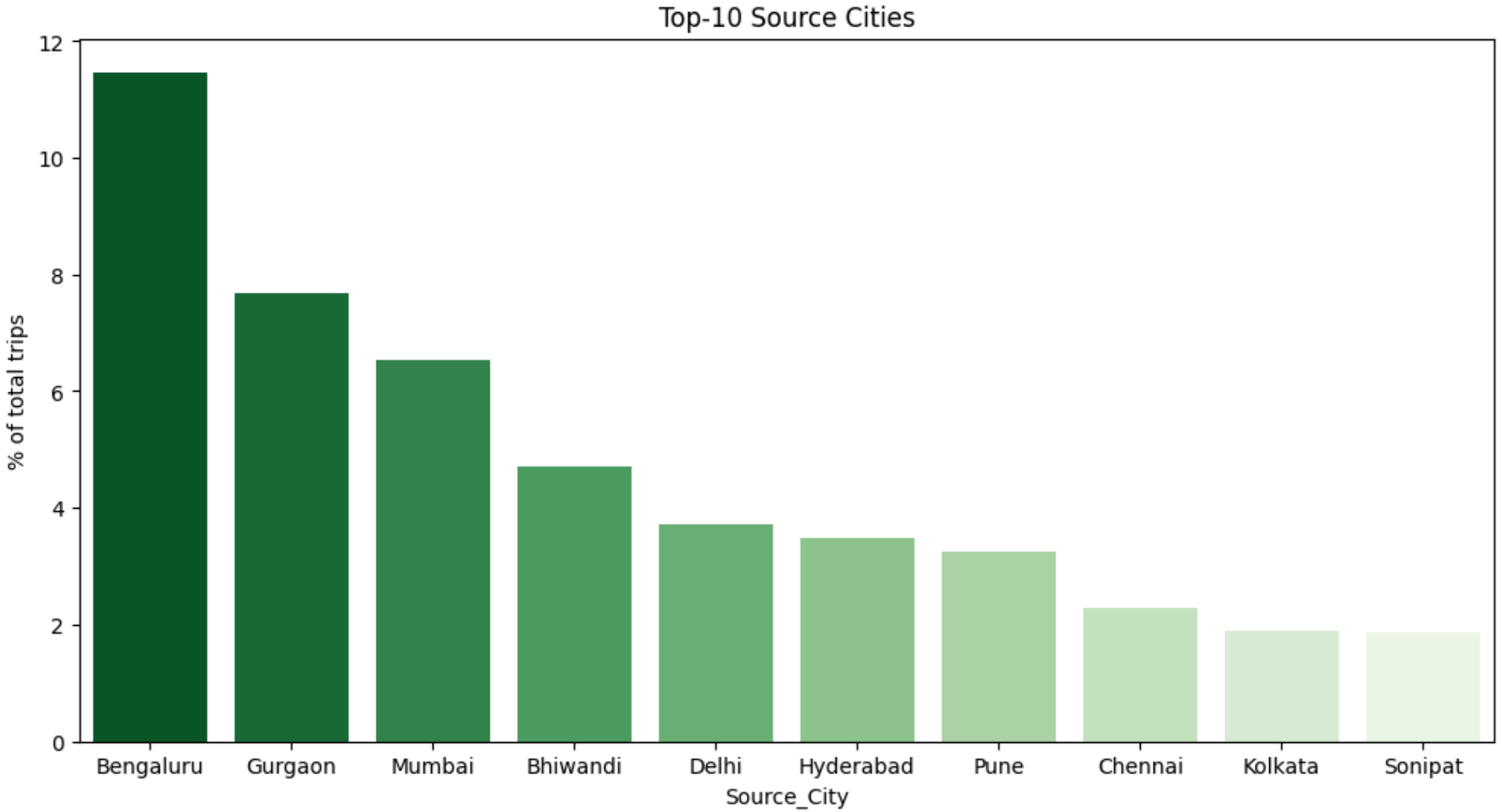
```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_source_cities.index, y = top_source_cities.values, palette='Greens_r')
plt.xlabel('Source_City')
plt.ylabel('% of total trips')
plt.title('Top-10 Source Cities')
plt.show()
```

```
<ipython-input-911-332a90d37220>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
```

```
` and set `legend=False` for the same effect.

  sns.barplot(x = top_source_cities.index, y = top_source_cities.values, palette='Greens_r')
```



Top-10 Destination cities

In [912…

```
#getting top-10 cities and their counts
top_destination_cities = df2['destination_city'].value_counts(normalize = True).head(10)*100
top_destination_cities
```

Out[912…

destination_city	proportion
Bengaluru	11.061618
Mumbai	8.112303
Gurgaon	5.918877
Delhi	3.711952
Hyderabad	3.367753
Bhiwandi	2.929068
Chennai	2.767092
Chandigarh	2.287913
Sonipat	2.173179
Pune	2.139434

dtype: float64

In [100…

```
df2['destination_city'].nunique()
```

Out[100…

```
840
```

Bengaluru top the destination city list, followed by Mumbai, Gurgaon, Delhi and Hyderabad. Highlighting strong demand from these cities.

In [913…

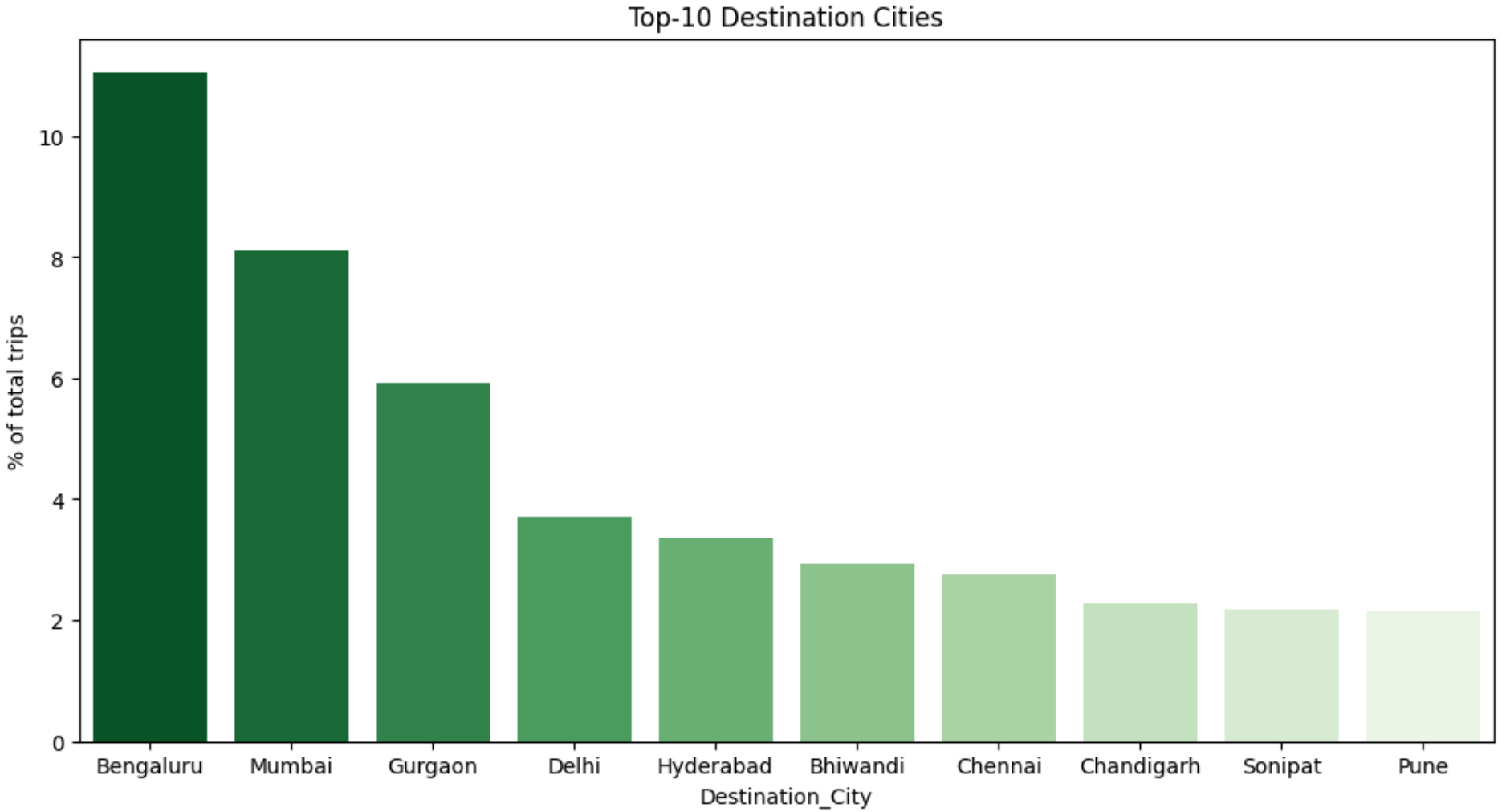
```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_destination_cities.index, y = top_destination_cities.values, palette='Greens_r')
plt.xlabel('Destination_City')
plt.ylabel('% of total trips')
plt.title('Top-10 Destination Cities')
plt.show()
```

```
<ipython-input-913-f6465f6a1d95>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
```

```
` and set `legend=False` for the same effect.

  sns.barplot(x = top_destination_cities.index, y = top_destination_cities.values, palette='Greens_r')
```



Trip count by hour of trip creation time

In [914…

```
#getting the count of trips by hour its booked
hourly_count = df2['trip_creation_hour'].value_counts(normalize = True).sort_index()*100
#displaying the sorted hourly trip count
hourly_count.head()
```

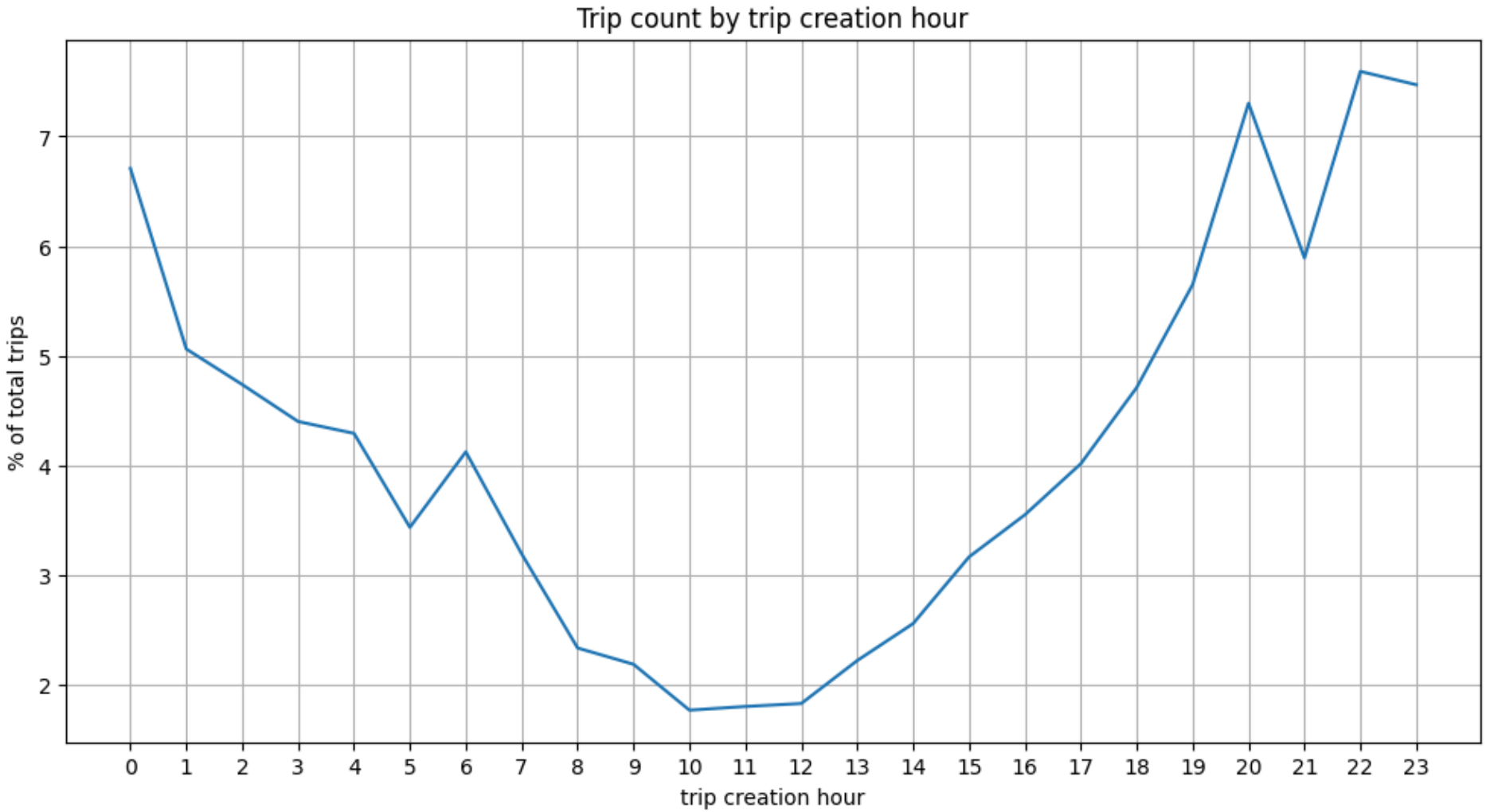
Out[914…

trip_creation_hour	proportion
0	6.708510
1	5.061753
2	4.737801
3	4.400351
4	4.292367

dtype: float64

In [915…

```
#creating a line plot for count of trips
plt.figure(figsize = (12, 6))
sns.lineplot(x = hourly_count.index, y = hourly_count.values)
plt.xlabel('trip creation hour')
plt.ylabel('% of total trips')
plt.title('Trip count by trip creation hour')
plt.xticks(np.arange(0, 24))
plt.grid('x')
plt.show()
```



Most of the trips starts after 12 PM and peaks around 7-11 PM and then start declining.

Trip count by day of the month

In [916…

```
#getting the count of trips by hour its booked
day_count = df2['trip_creation_day'].value_counts(normalize = True).sort_index()*100
day_count.head()
```

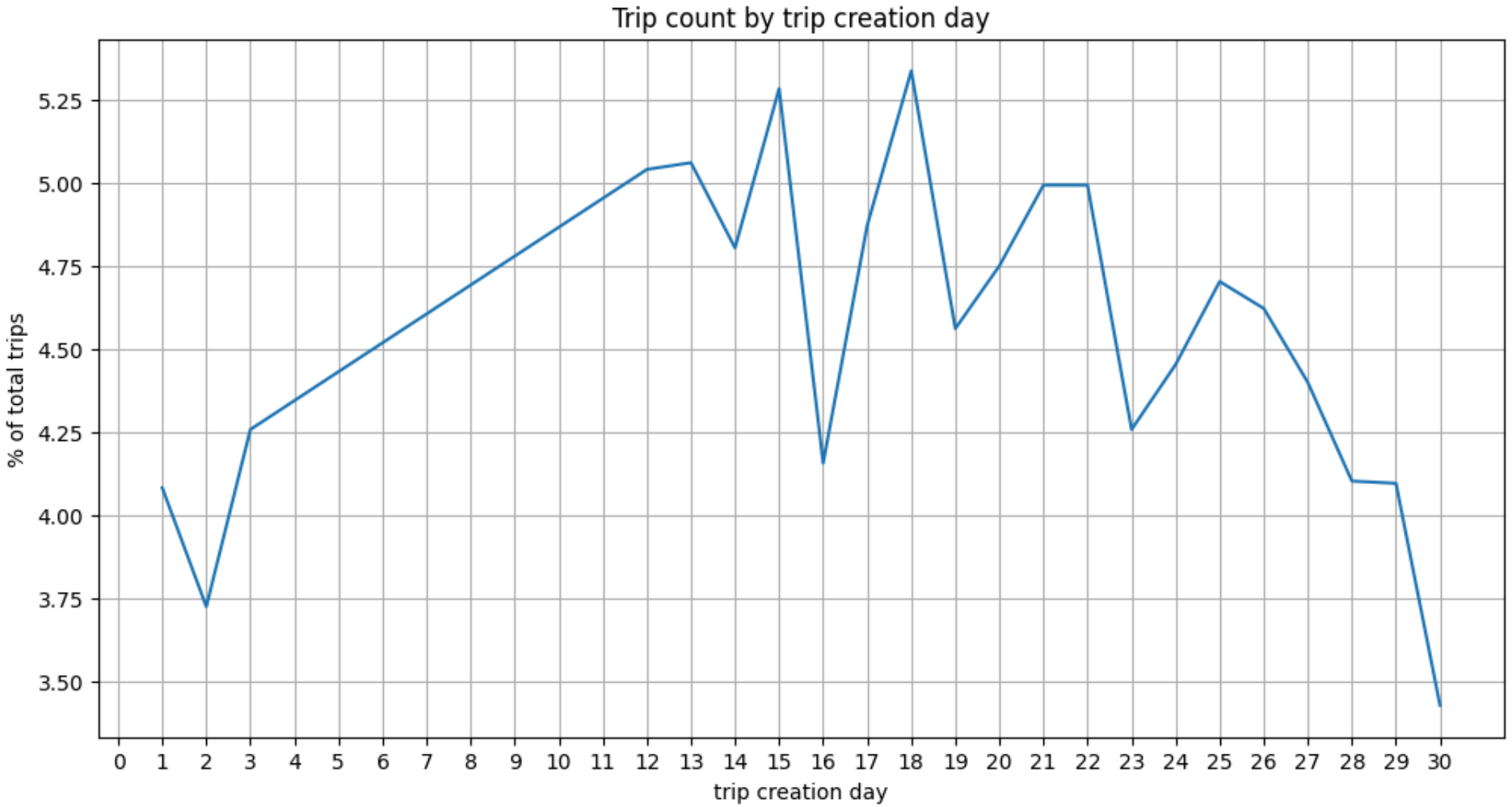
Out [916…

trip_creation_day	proportion
1	4.083148
2	3.725450
3	4.258622
12	5.041506
13	5.061753

dtype: float64

In [917…

```
#creating a line plot for count of trips
plt.figure(figsize = (12, 6))
sns.lineplot(x = day_count.index, y = day_count.values)
plt.xlabel('trip creation day')
plt.ylabel('% of total trips')
plt.title('Trip count by trip creation day')
plt.xticks(np.arange(0, 31))
plt.grid('x')
plt.show()
```



trips booking picks up at the start of the months, peaks in middle and again start declining in %

Trip count by week

In [918…

```
#getting the count of trips by the week no its booked
week_count = df2['trip_creation_week'].value_counts().sort_index()
week_count
```

Out[918…

trip_creation_week	count
37	3608
38	5004
39	4417
40	1788

dtype: int64

Trip count by month of year

In [919…

```
#getting the count of trips by hour its booked
month_count = df2['trip_creation_month'].value_counts().sort_index()
month_count
```

Out[919…

trip_creation_month	count
9	13029
10	1788

dtype: int64

Seems like we have limited data for these two months.

Trip count by year

In [920…

```
#getting the count of trips by hour its booked
year_count = df2['trip_creation_year'].value_counts().sort_index()
year_count
```

Out[920…

trip_creation_year	count
2018	14817

dtype: int64

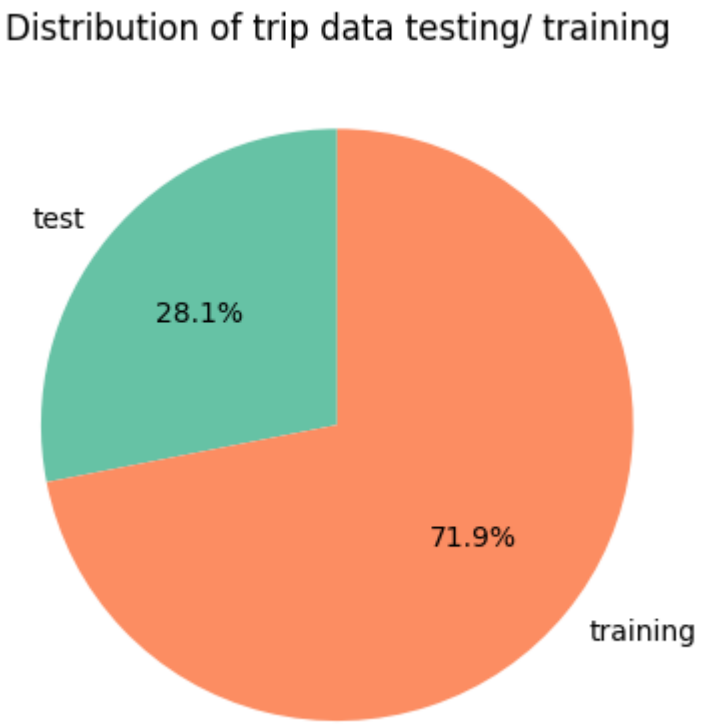
Most of the data belongs to 2018.

Distribution of trip data testing/ training

In [921…

```
#getting the count of trips by data type
data_count = df2['data'].value_counts().sort_index()

plt.pie(data_count, labels = data_count.index, autopct = '%.1f%%', startangle=90, colors=sns.color_palette('Set2'))
plt.title('Distribution of trip data testing/ training')
plt.show()
```

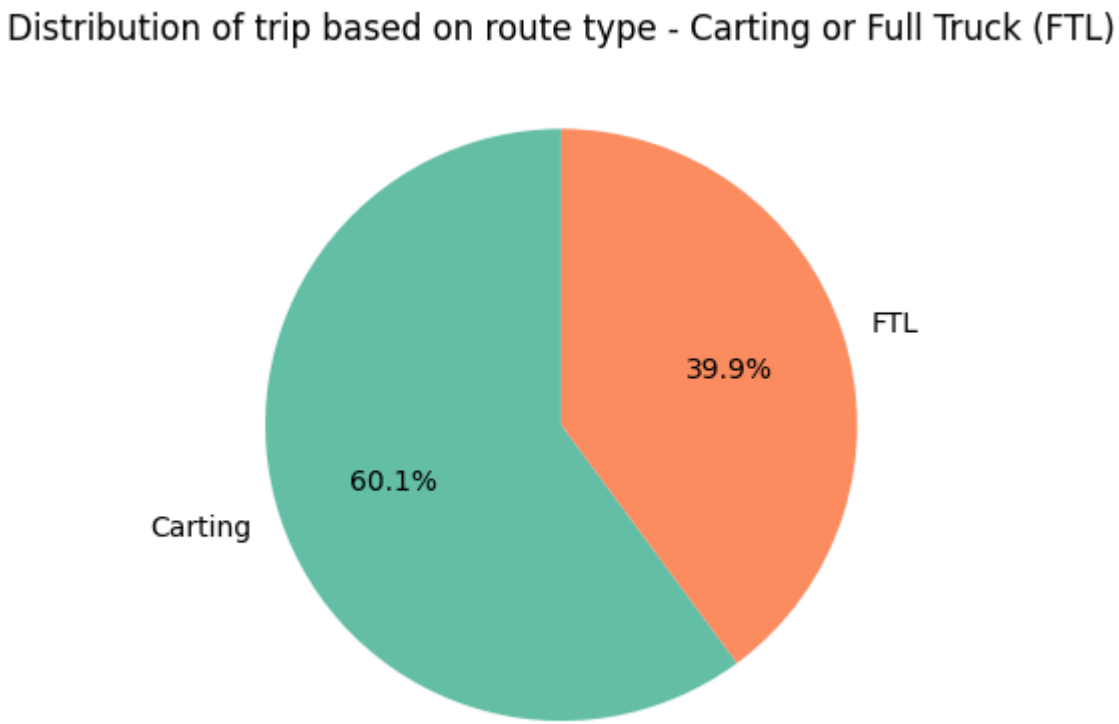


Distribution of trip data by route type

In [922…

```
#getting the count of trips by data type
route_type_count = df2['route_type'].value_counts().sort_index()

plt.pie(route_type_count, labels = route_type_count.index, autopct = '%.1f%%', startangle=90, colors=sns.color_palette(
plt.title('Distribution of trip based on route type - Carting or Full Truck (FTL)')
plt.show()
```



Distribution of inter and intra state trips

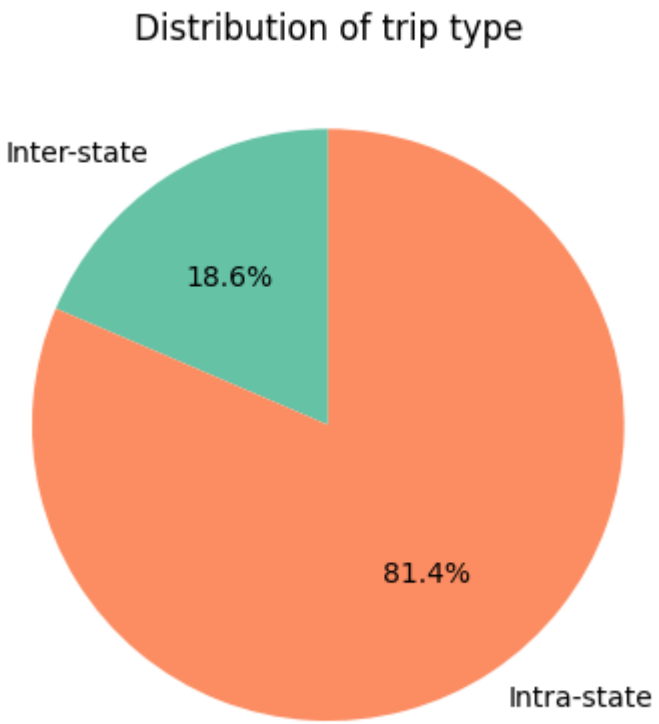
In [923…

```
#creating a column for inter/intra state trip
df2['trip_type'] = np.where(df2['source_state'] == df2['destination_state'], 'Intra-state', 'Inter-state')
```

In [924…

```
#getting the count of trips by data type
trip_type_count = df2['trip_type'].value_counts().sort_index()

plt.pie(trip_type_count, labels = trip_type_count.index, autopct = '%.1f%%', startangle=90, colors=sns.color_palette('S
plt.title('Distribution of trip type')
plt.show()
```



81% of the trips start and end in same state.

Top-10 states with Intra-state trips

In [925…

```
#getting top-10 based on destination states
top_intra_state = df2[df2['trip_type'] == 'Intra-state']['destination_state'].value_counts(normalize = True).head(10)*1
top_intra_state
```

Out[925…

destination_state	proportion
Maharashtra	20.329852
Karnataka	17.047903
Tamil Nadu	8.461793

destination_state	proportion
Haryana	7.865075
Telangana	5.710260
Gujarat	5.212995
West Bengal	5.096967
Uttar Pradesh	4.947787
Rajasthan	3.754351
Andhra Pradesh	3.431129

dtype: float64

Maharastra & Karnataka contributes to more than 37% intra-state trip, followed by Tamilnadu, harayana.

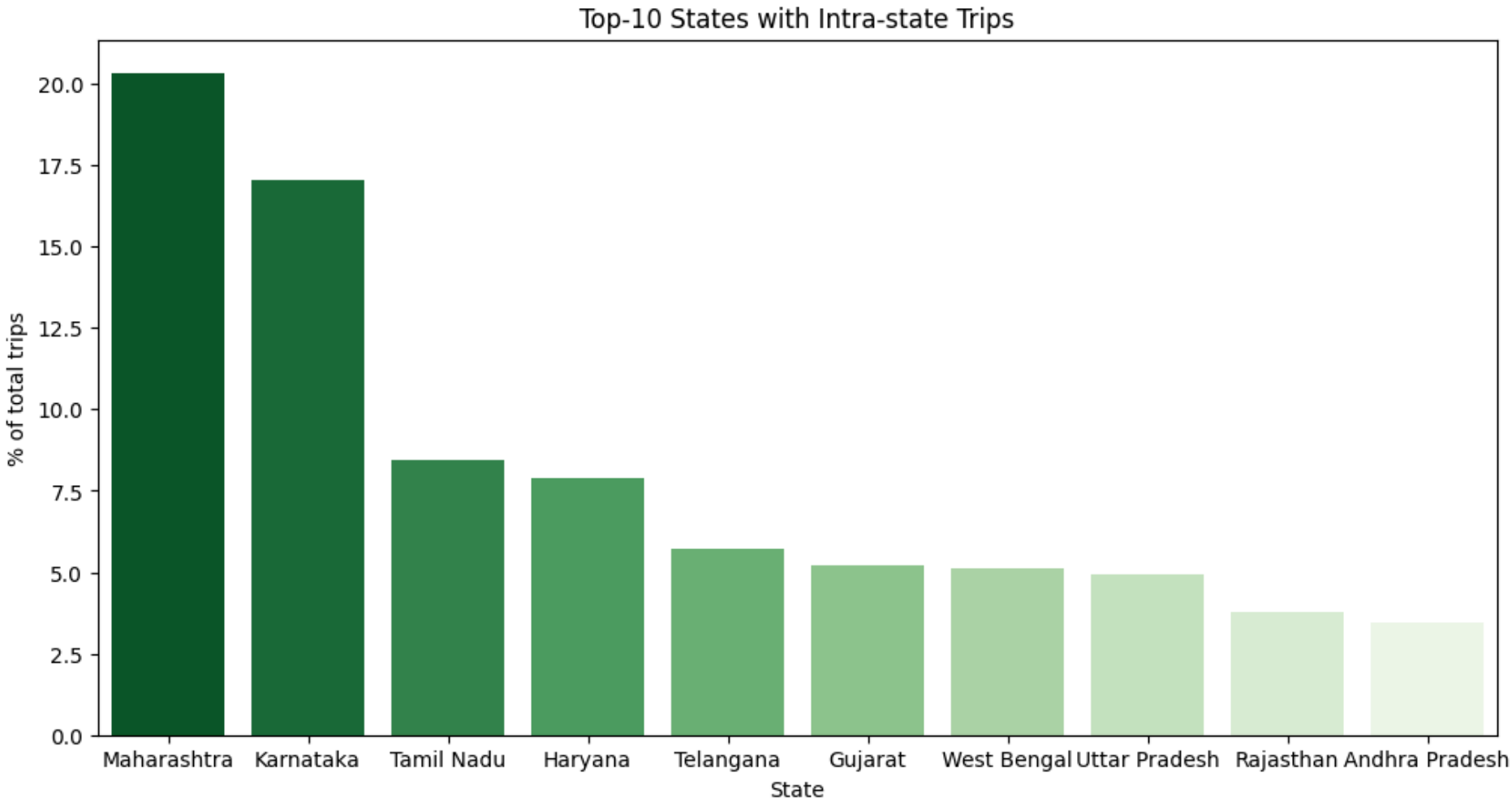
In [926…

```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_intra_state.index, y = top_intra_state.values, palette='Greens_r')
plt.xlabel('State')
plt.ylabel('% of total trips')
plt.title('Top-10 States with Intra-state Trips')
plt.show()
```

```
<ipython-input-926-642847101d14>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
` and set `legend=False` for the same effect.

  sns.barplot(x = top_intra_state.index, y = top_intra_state.values, palette='Greens_r')
```



Top-10 states with Inter-state trips

In [927…

```
#getting top-10 destination states based on destination states
top_inter_destination_source = df2[df2['trip_type'] == 'Inter-state']['destination_state'].value_counts(normalize = Tru
top_inter_destination_source
```

Out[927…

destination_state	proportion
Haryana	25.300766

destination_state	proportion
Delhi	16.296026
Karnataka	8.676631
Uttar Pradesh	8.093328
Punjab	7.619395
Maharashtra	3.937295
Gujarat	3.827926
Rajasthan	3.536274
Telangana	3.463361
Madhya Pradesh	3.171710

dtype: float64

Haryana and Delhi tops the list with destination states for inter-state trips as well contributing more than 40%

In [928…

```
#getting top-10 source states based on destination states
top_inter_source_source = df2[df2['trip_type'] == 'Inter-state']['source_state'].value_counts(normalize = True).head(10
top_inter_source_source
```

Out[928…

source_state	proportion
Haryana	32.433418
Delhi	19.044144
Maharashtra	9.522072
Uttar Pradesh	6.019701
Punjab	4.669829
Gujarat	4.414447
Telangana	3.502371

source_state	proportion
Karnataka	3.137541
Chandigarh	2.699745
Rajasthan	2.225465

dtype: float64

Haryana & Delhi leads source in the Inter-state trips with more 50% trips starts from there, follow2ed by Maharastra and UttarPradesh.

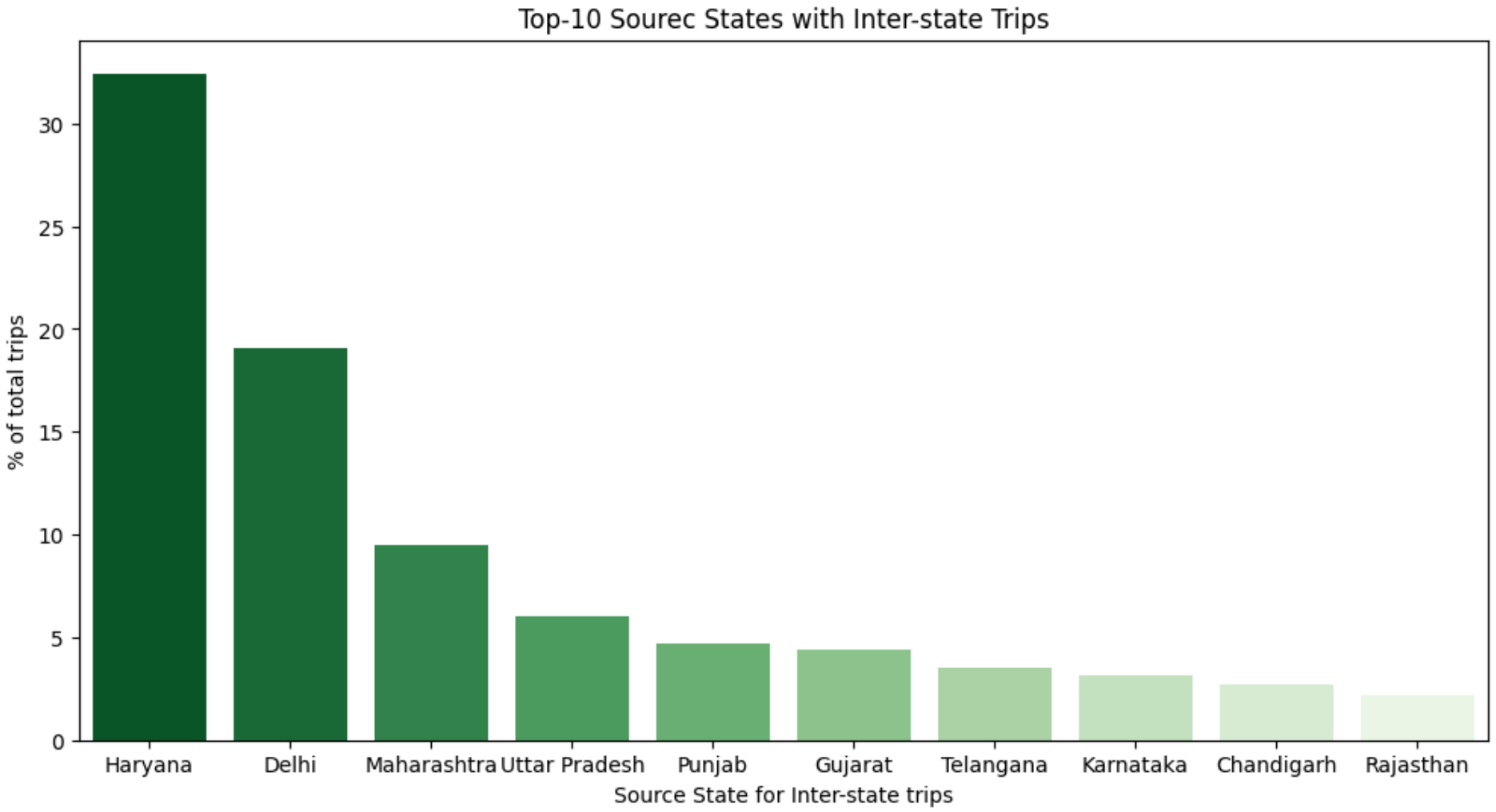
In [929…

```
#creating a bar plot for count of trips by source states
plt.figure(figsize = (12, 6))
sns.barplot(x = top_inter_source_source.index, y = top_inter_source_source.values, palette='Greens_r')
plt.xlabel('Source State for Inter-state trips')
plt.ylabel('% of total trips')
plt.title('Top-10 Source States with Inter-state Trips')
plt.show()
```

```
<ipython-input-929-edf727e6ae88>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue
` and set `legend=False` for the same effect.

  sns.barplot(x = top_inter_source_source.index, y = top_inter_source_source.values, palette='Greens_r')
```



Bi-variate/ Multivariate Analysis

Corelation matrix

In [930…

```
correlation_matrix = df2.corr(numeric_only=True) #only for numeric columns
correlation_matrix
```

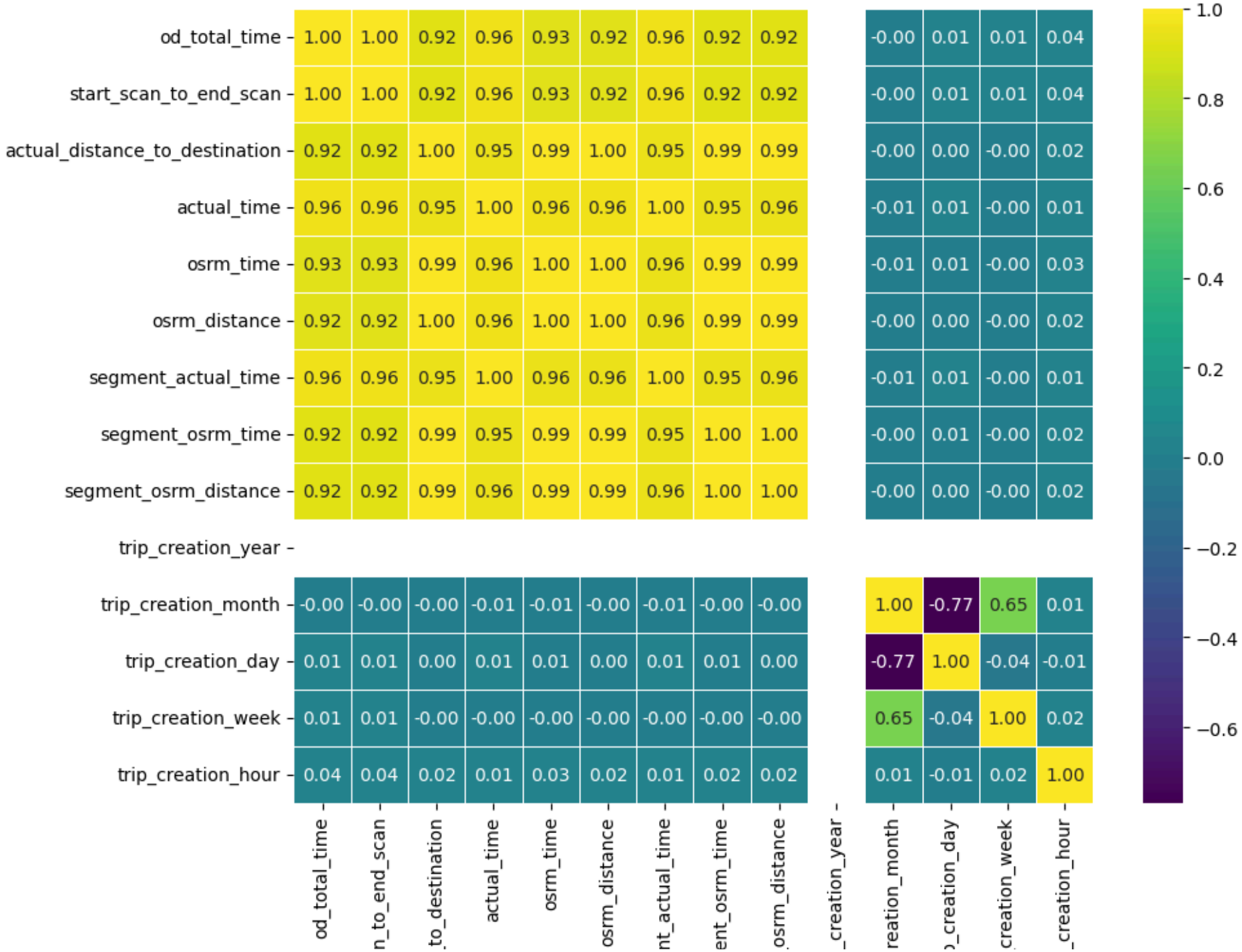
Out[930…

	od_total_time	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	se
od_total_time	1.000000	0.999999	0.918222	0.961094	0.926516	0.924219	
start_scan_to_end_scan	0.999999	1.000000	0.918308	0.961147	0.926571	0.924299	
actual_distance_to_destination	0.918222	0.918308	1.000000	0.953757	0.993561	0.997264	
actual_time	0.961094	0.961147	0.953757	1.000000	0.958593	0.959214	
osrm_time	0.926516	0.926571	0.993561	0.958593	1.000000	0.997580	
osrm_distance	0.924219	0.924299	0.997264	0.959214	0.997580	1.000000	
segment_actual_time	0.961098	0.961151	0.952829	0.999976	0.957771	0.958357	
segment_osrm_time	0.918510	0.918581	0.987530	0.953886	0.993260	0.991797	
segment_osrm_distance	0.919220	0.919313	0.993054	0.956982	0.991609	0.994709	
trip_creation_year	NaN	NaN	NaN	NaN	NaN	NaN	
trip_creation_month	-0.000677	-0.000676	-0.004289	-0.006265	-0.006653	-0.004961	
trip_creation_day	0.007311	0.007303	0.004321	0.007191	0.005034	0.004840	
trip_creation_week	0.006134	0.006122	-0.002246	-0.001533	-0.004816	-0.002587	
trip_creation_hour	0.035781	0.035723	0.020800	0.012795	0.026356	0.022958	

Not very clear by looking at correlation matrix, let's draw heatmap to get visual clariy

In [931…

```
# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt=".2f", linewidths=0.5)
plt.show()
```


od_total_time 1.00 1.00 0.92 0.96 0.93 0.92 0.96 0.92 0.92 -0.00 0.01 0.01 0.04
start_scan_to_end_scan 1.00 1.00 0.92 0.96 0.93 0.92 0.96 0.92 0.92 -0.00 0.01 0.01 0.04
actual_distance_to_destination 0.92 0.92 1.00 0.95 0.99 1.00 0.95 0.99 0.99 -0.00 0.00 -0.00 0.02
actual_time 0.96 0.96 0.95 1.00 0.96 0.96 1.00 0.95 0.96 -0.01 0.01 -0.00 0.01
osrm_time 0.93 0.93 0.99 0.96 1.00 1.00 0.96 0.99 0.99 -0.01 0.01 -0.00 0.03
osrm_distance 0.92 0.92 1.00 0.96 1.00 1.00 0.96 0.99 0.99 -0.00 0.00 -0.00 0.02
segment_actual_time 0.96 0.96 0.95 1.00 0.96 0.96 1.00 0.95 0.96 -0.01 0.01 -0.00 0.01
segment_osrm_time 0.92 0.92 0.99 0.95 0.99 0.99 0.95 1.00 1.00 -0.00 0.01 -0.00 0.02
segment_osrm_distance 0.92 0.92 0.99 0.96 0.99 0.99 0.96 1.00 1.00 -0.00 0.00 -0.00 0.02
trip_creation_year
trip_creation_month -0.00 -0.00 -0.00 -0.01 -0.01 -0.00 -0.01 -0.00 -0.00 1.00 -0.77 0.65 0.01
trip_creation_day 0.01 0.01 0.00 0.01 0.01 0.00 0.01 0.01 0.00 -0.77 1.00 -0.04 -0.01
trip_creation_week 0.01 0.01 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.65 -0.04 1.00 0.02
trip_creation_hour 0.04 0.04 0.02 0.01 0.03 0.02 0.01 0.02 0.02 0.01 -0.01 0.02 1.00
od_total_time
n_to_end_scan
to_destination
actual_time
osrm_time
osrm_distance
nt_actual_time
ent_osrm_time
osrm_distance
_creation_year
reation_month
creation_day
creation_week
_creation_hour
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

There is strong correlation among lot of variables > 0.9

Hypothesis testing

Between Total trip time and start_scan_to_end_scan.

Compare the difference between total trip time and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

Steps

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Total Trip Time and Start to end scan time are same.
- Alternate Hypothesis (HA) - Total Trip Time and start to end scan time are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavenes test

STEP-3: Define Test statistics If the assumptions of the T-Test are satisfied, we can proceed with conducting the T-Test for independent samples. Otherwise, we will resort to performing the non-parametric equivalent of the T-Test for independent samples, namely the Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha. We set our alpha to be 0.05

STEP-5: Compare p-value and alpha. Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

Check for Normality

Visual Analysis

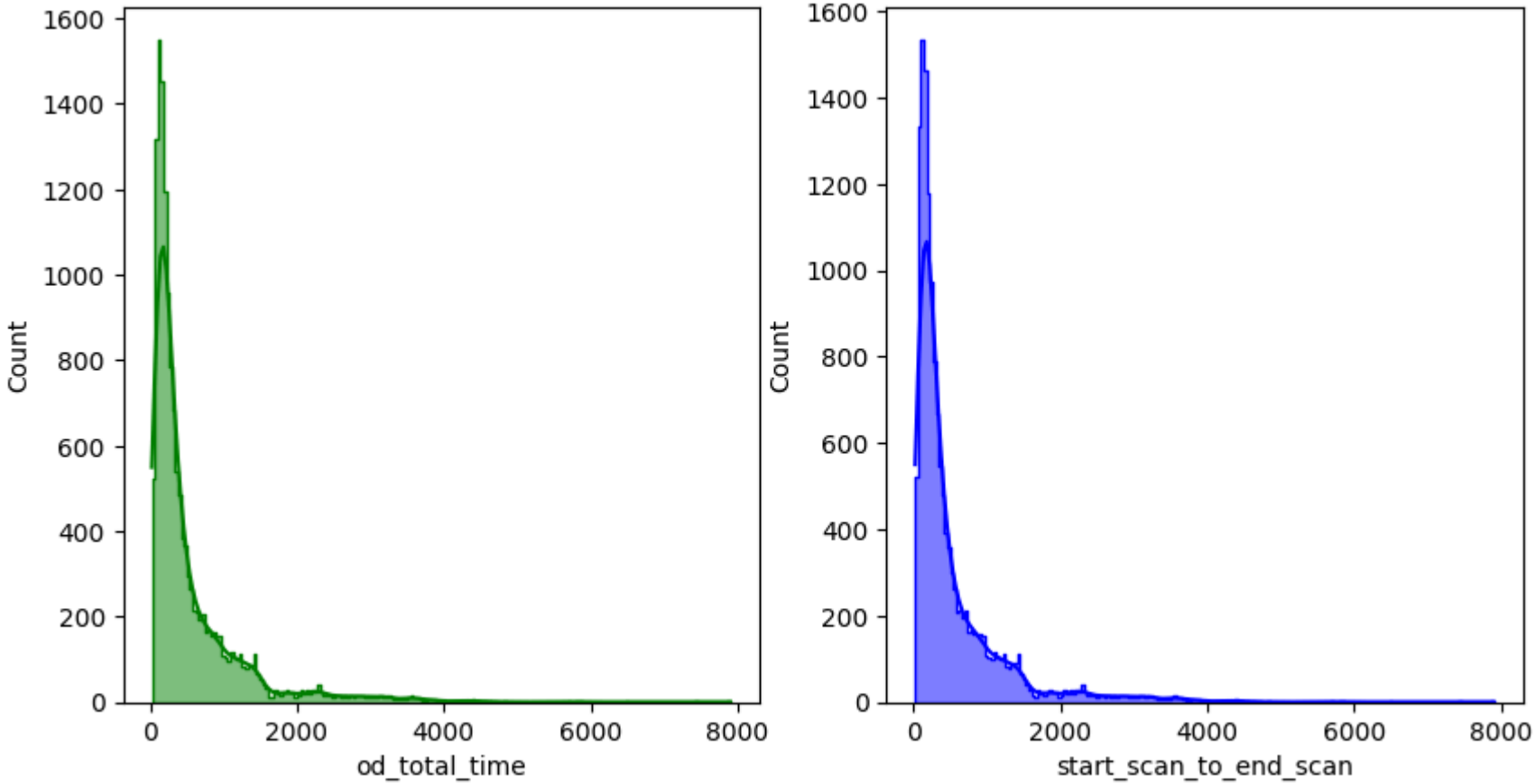
In [932…

```
plt.figure(figsize = (10,5))

plt.subplot(1, 2, 1)
#histogram for Total trip time to check for distribution
sns.histplot(df2['od_total_time'], element = 'step', kde = True, color = 'green', label = 'Total Trip time')

plt.subplot(1, 2, 2)
#histogram for start to end scan to check for distribution
sns.histplot(df2['start_scan_to_end_scan'], element = 'step', kde = True, color = 'blue', label = 'Total time start to

plt.show()
```



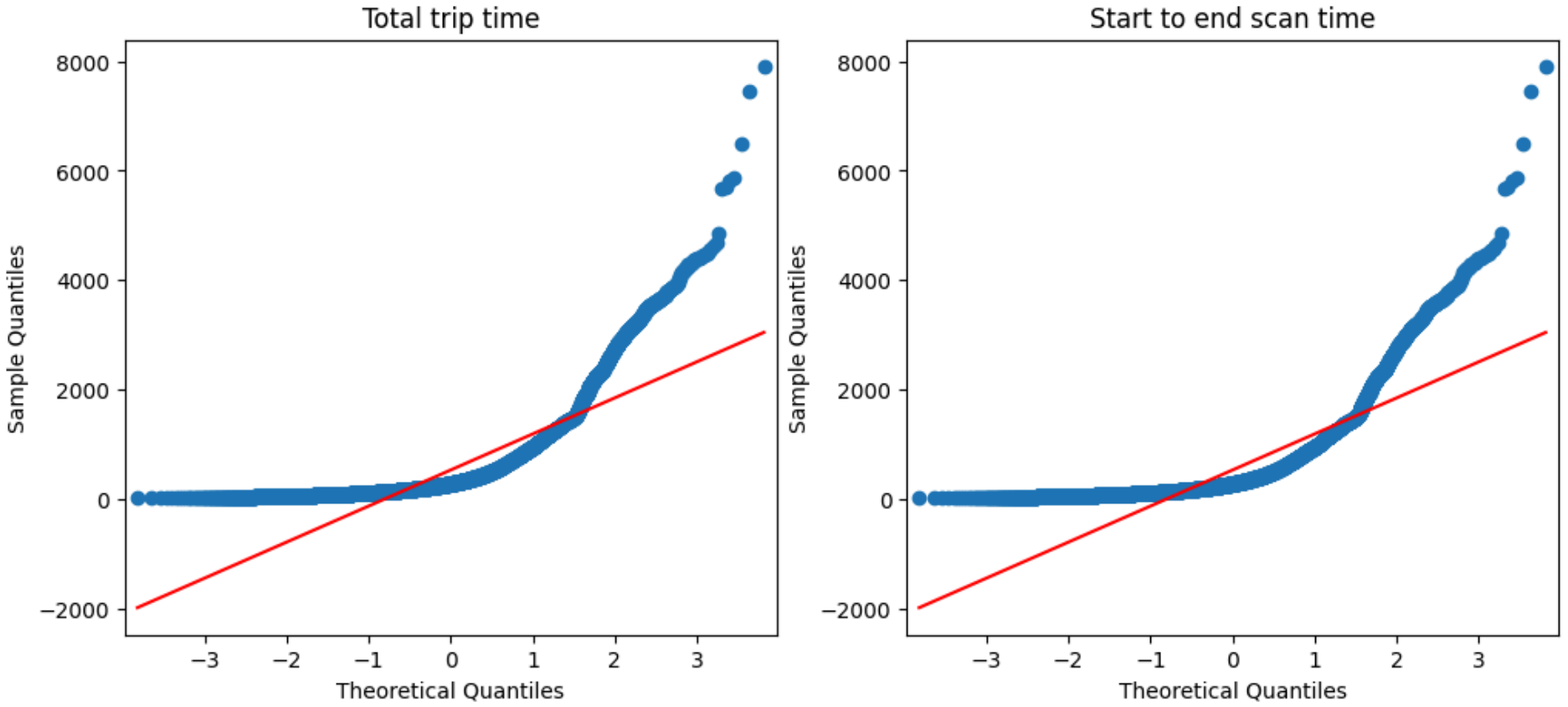
QQ Plot for normality

In [933…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)

#plotting the qq plot for Total trip time
sm.qqplot(df2['od_total_time'], line = 's', ax = ax)
plt.title('Total trip time')

ax = fig.add_subplot(122)
#plotting the qq plot for start to scan end time
sm.qqplot(df2['start_scan_to_end_scan'], line = 's', ax = ax)
plt.title('Start to end scan time')
plt.show()
```



From the above QQ plot data doesn't seem to be following normal distribution.

Shapiro wilk test for normality

In [934…

```
#taking sample for test for both the columns
total_trip_time = df2['od_total_time'].sample(3000)
start_scan_to_end_scan = df2['start_scan_to_end_scan'].sample(3000)
```

For Total Trip Time

In [935…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking Total trip time sample and running the test on that
test_stat, p_value = shapiro(total_trip_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.4888497182002707e-60
Reject the null hypothesis
```

For Total start to end scan time

In [936…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
```

```
np.random.seed(42)

# taking scan start/end sample and running the test on that
test_stat, p_value = shapiro(start_scan_to_end_scan)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.457471790478121e-59
Reject the null hypothesis
```

From above shapiro test we can say that sample for Total trip time and Time for start to end scan doesn't follow normal distribution

Box-cox to transform data to normal

In [937…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_trip_time)
test_stat, p_value = shapiro(transformed1)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.122851670213156e-09
Reject the null hypothesis
```

In [938…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(start_scan_to_end_scan)
test_stat, p_value = shapiro(transformed2)

print(p_value)
```

```
#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

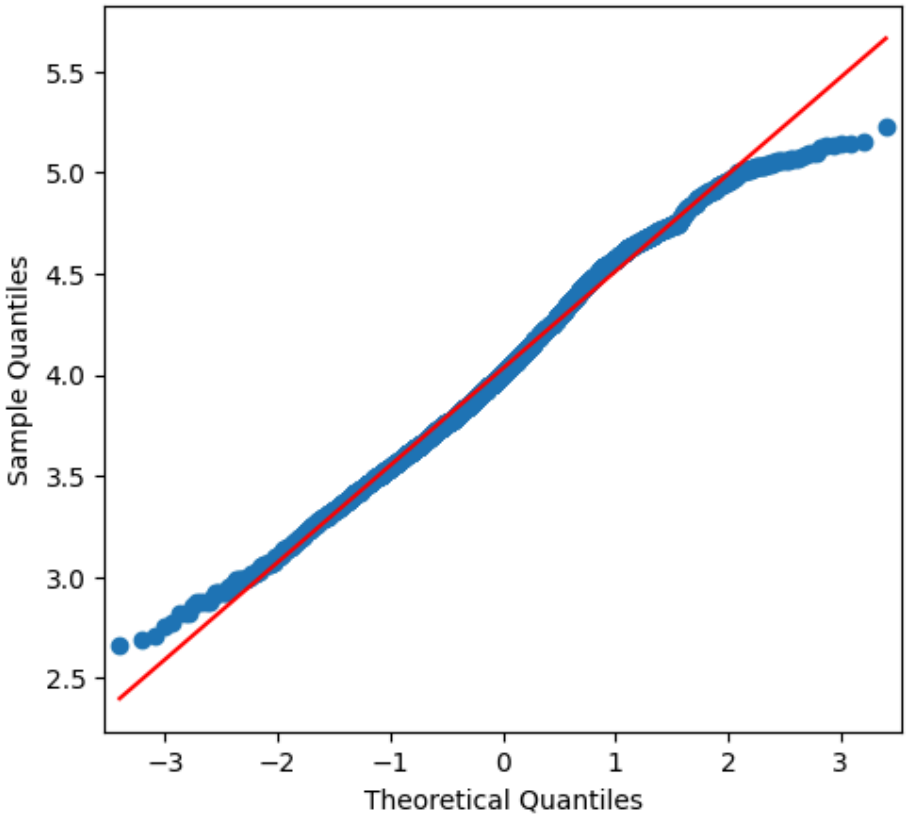
```
6.125800718312231e-10
Reject the null hypothesis
```

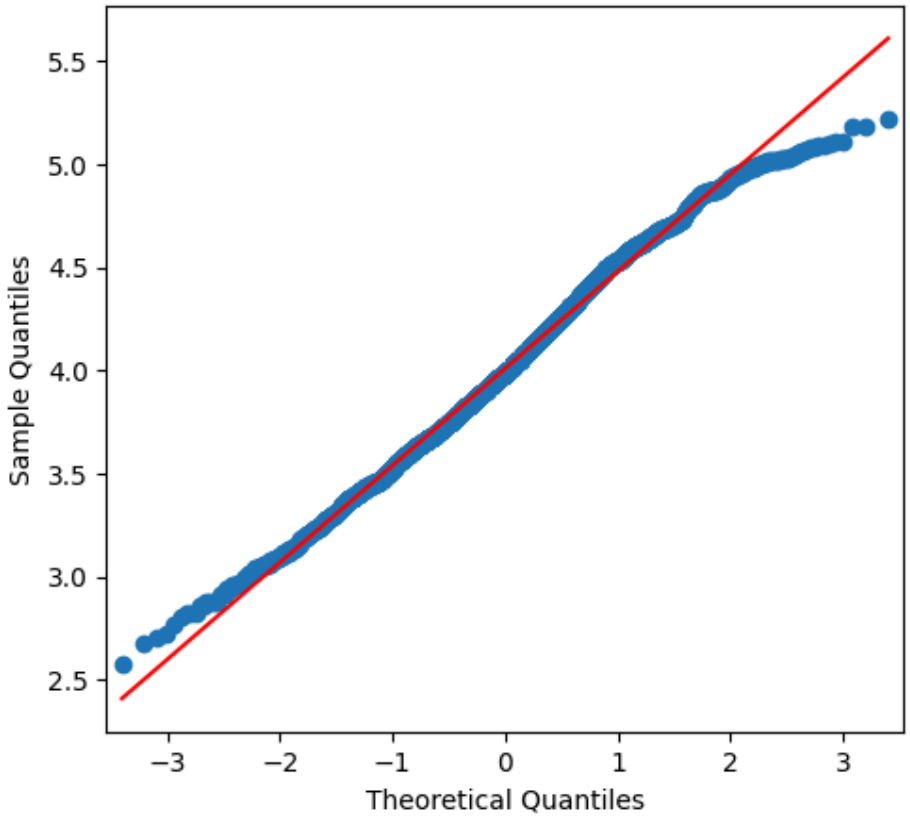
EVen after applying the box-cox transformation data is not normal, which we can verfiy through the qq plot

In [939…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)
#plotting the qq plot for the working day
sm.qqplot(transformed1, line = 's', ax = ax)

ax = fig.add_subplot(122)
#plotting the qq plot for the non-working day
sm.qqplot(transformed2, line = 's', ax = ax)
plt.show()
```





Levene test for homegenity of variance

In [940…

```
#checking the homgenity of the variance using the levene test

from scipy.stats import levene

# Null Hypothesis Ho - homogeneous variance
# Alternate Hypothesis Ha - Non- homogenous variance

test_stat, p_value = levene(total_trip_time, start_scan_to_end_scan)

print(p_value)

if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.6532355299343451
Accept the null hypothesis
```

Samples have homogenous variance

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

Hypothesis test - Nonparametric

In [941…

```
#performing Mann-whiteney test since the data is not normal

# Null Hypothesis ( H0 ) - Total Trip Time and Start to end scan time are same.
# Alternate Hypothesis ( HA ) - Total Trip Time and start to end scan time are different.
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu
test_stat, p_value = mannwhitneyu(total_trip_time, start_scan_to_end_scan)

print(p_value)
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.6757577354402539
Accept the null hypothesis
```

Since p-value is > 0.05 we can say that H0 is True, meaning Total Trip Time and Start to end scan time are same.

The features 'Total Trip Time' and 'Start to end scan time' are not statistically different.

Between aggregated values of Actual time and OSRM time

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)

High level summary

In [942…

```
df2[['actual_time', 'osrm_time']].describe()
```

Out[942...

	actual_time	osrm_time
count	14817.000000	14817.000000
mean	357.143754	161.384018
std	561.396157	271.360995
min	9.000000	6.000000
25%	67.000000	29.000000
50%	149.000000	60.000000
75%	370.000000	168.000000
max	6265.000000	2032.000000

Steps

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Total actual_time aggregated value and OSRM time aggregated value are same.
- Alternate Hypothesis (HA) - actual_time aggregated value and OSRM time aggregated value are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavenes test

STEP-3: Define Test statistics If the assumptions of the T-Test are satisfied, we can proceed with conducting the T-Test for independent samples. Otherwise, we will resort to performing the non-parametric equivalent of the T-Test for independent samples, namely the Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha. We set our alpha to be 0.05

STEP-5: Compare p-value and alpha. Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

Check for Normality

Visual Analysis

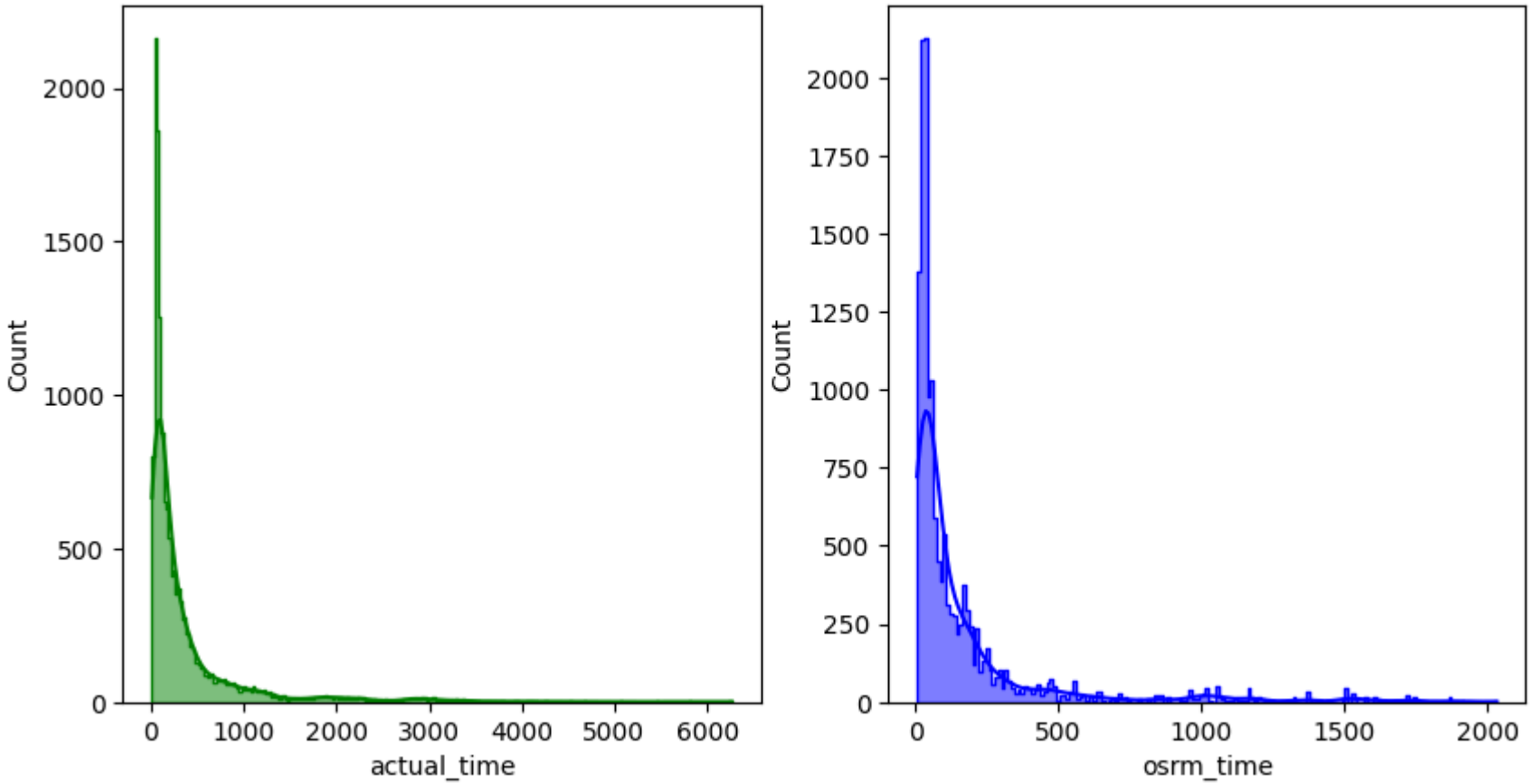
In [943…

```
plt.figure(figsize = (10,5))

plt.subplot(1, 2, 1)
#histogram for Total trip time to check for distribution
sns.histplot(df2['actual_time'], element = 'step', kde = True, color = 'green', label = 'actual_time')

plt.subplot(1, 2, 2)
#histogram for start to end scan to check for distribution
sns.histplot(df2['osrm_time'], element = 'step', kde = True, color = 'blue', label = 'osrm_time')

plt.show()
```



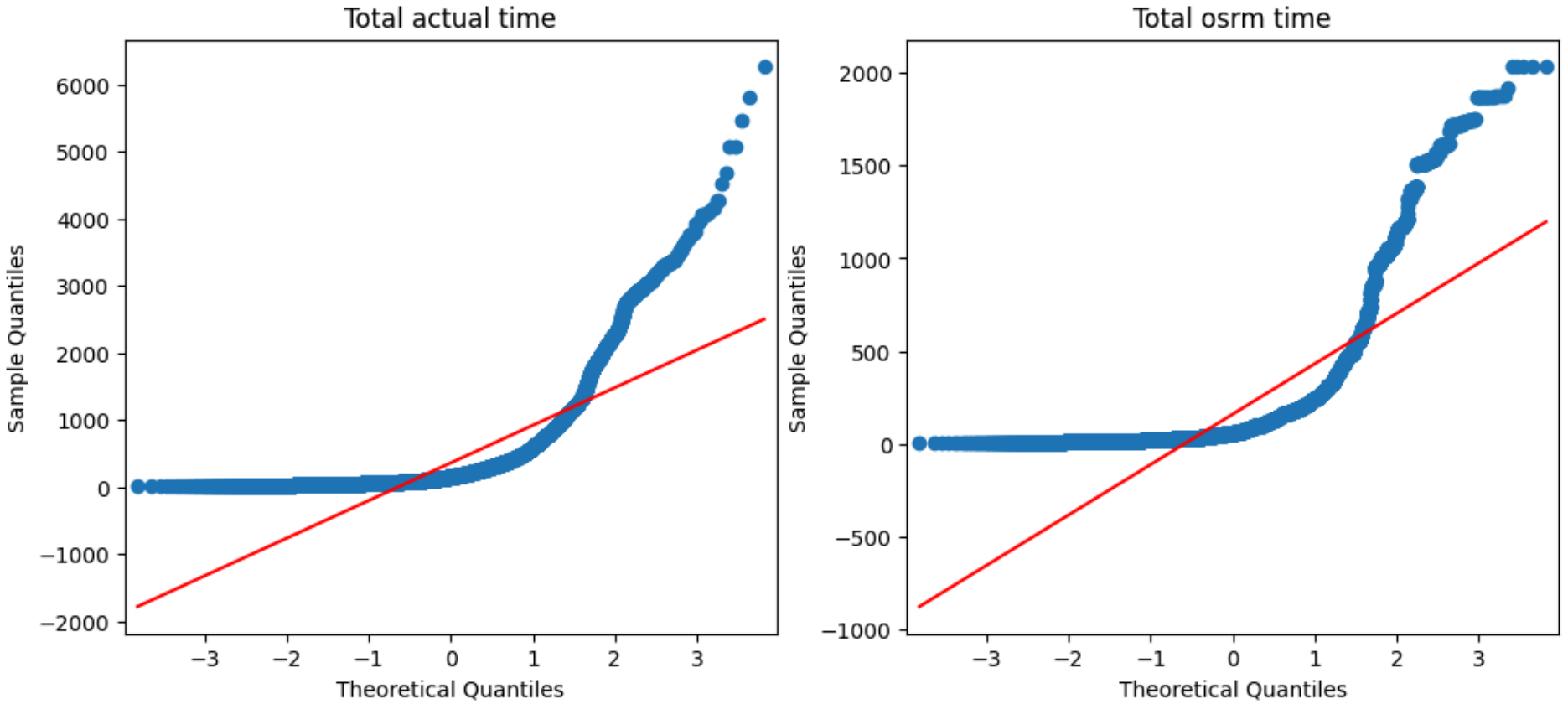
QQ Plot for normality

In [944…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)

#plotting the qq plot for Total trip time
sm.qqplot(df2['actual_time'], line = 's', ax = ax)
plt.title('Total actual time')

ax = fig.add_subplot(122)
#plotting the qq plot for start to scan end time
sm.qqplot(df2['osrm_time'], line = 's', ax = ax)
plt.title('Total osrm time')
plt.show()
```



From the above QQ plot data doesn't seem to be following normal distribution.

Shapiro wilk test for normality

In [945…

```
#taking sample for test for both the columns
total_actual_time = df2['actual_time'].sample(3000)
total_osrm_time = df2['osrm_time'].sample(3000)
```

For Total Actual Time

In [946…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking Total trip time sample and running the test on that
test_stat, p_value = shapiro(total_actual_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.045115166904931e-65
Reject the null hypothesis
```

For Total osrm time

In [947…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
```

```
np.random.seed(42)

# taking scan start/end sample and running the test on that
test_stat, p_value = shapiro(total_osrm_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
2.7245940544573097e-66
Reject the null hypothesis
```

From above shapiro test we can say that sample for Total Actual time and osrm time doesn't follow normal distribution

Box-cox to transform data to normal

In [948…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_actual_time)
test_stat, p_value = shapiro(transformed1)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.0308254643901285e-11
Reject the null hypothesis
```

In [949…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_osrm_time)
test_stat, p_value = shapiro(transformed2)

print(p_value)
```

```
#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

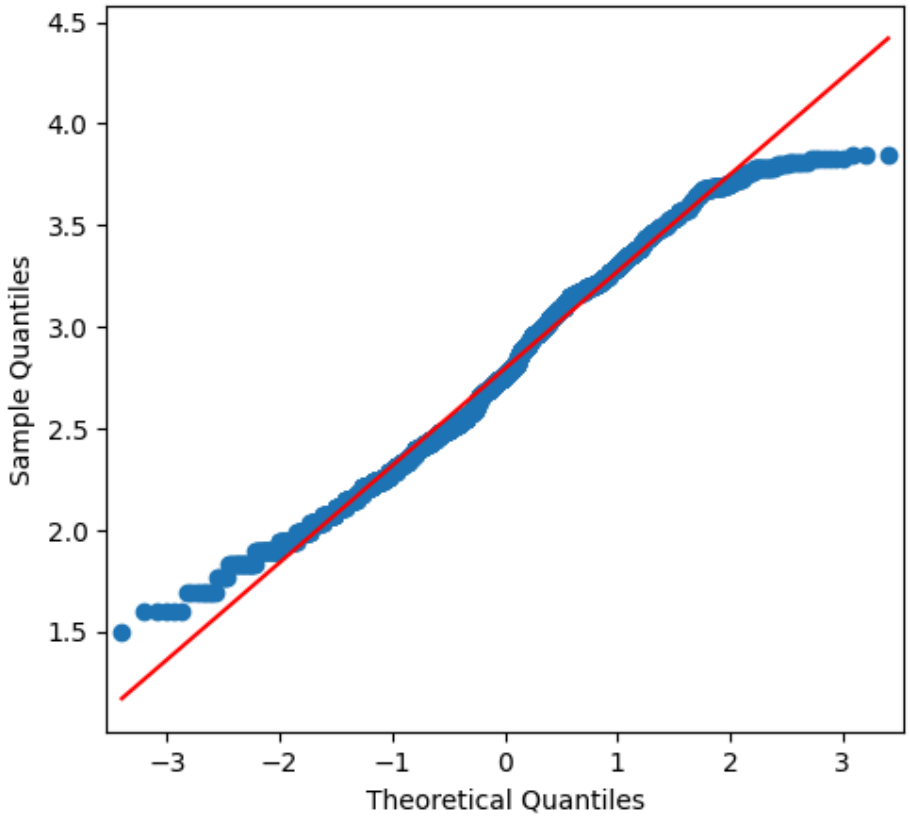
```
6.125800718312231e-10
Reject the null hypothesis
```

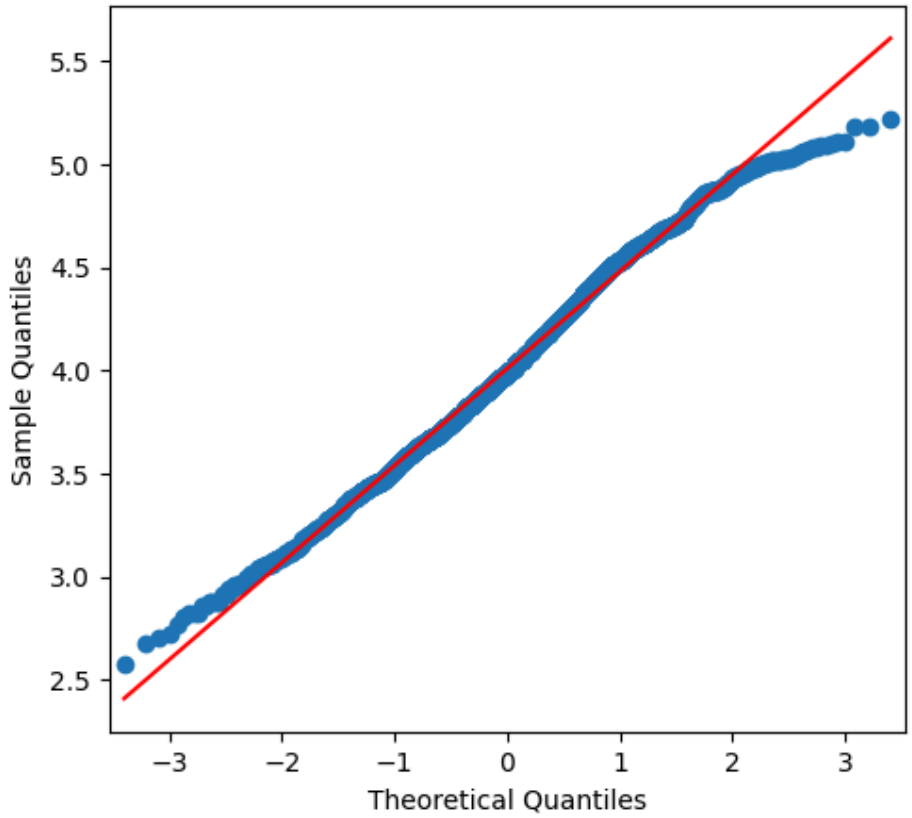
Even after applying the box-cox transformation data is not normal, which we can verfiy through the qq plot

In [950…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)
#plotting the qq plot for the working day
sm.qqplot(transformed1, line = 's', ax = ax)

ax = fig.add_subplot(122)
#plotting the qq plot for the non-working day
sm.qqplot(transformed2, line = 's', ax = ax)
plt.show()
```





Levene test for homegenity of variance

In [951…

```
#checking the homgenity of the variance using the levene test

from scipy.stats import levene

# Null Hypothesis Ho - homogeneous variance
# Alternate Hypothesis Ha - Non- homogenous variance

test_stat, p_value = levene(total_actual_time, total_osrm_time)

print(p_value)

if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.7883917095550236e-37
Reject the null hypothesis
```

Samples don't have homogenous variance

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

Hypothesis test - Nonparametric

In [952…

```
#performing Mann-whiteney test since the data is not normal

# Null Hypothesis ( H0 ) - Total Actual Time and Total osrm time are same.
# Alternate Hypothesis ( HA ) - Total Actual Time and Total osrm timr are different.
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu
test_stat, p_value = mannwhitneyu(total_actual_time, total_osrm_time)

print(p_value)
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.4987136032534865e-142
Reject the null hypothesis
```

Since p-value is less than 0.05 we can say that H0 is rejected, meaning Total actual time and total osrm time are not similar.

- The features 'Total actual Time' and 'total osrm time' are statistically different.

Between aggregated values of Actual time and Segment actual time

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)

High level summary

In [953…
```
df2[['actual_time', 'segment_actual_time']].describe()
```

Out[953…

	actual_time	segment_actual_time
count	14817.000000	14817.000000
mean	357.143754	353.951610
std	561.396157	556.320988
min	9.000000	9.000000
25%	67.000000	66.000000
50%	149.000000	147.000000
75%	370.000000	367.000000
max	6265.000000	6230.000000

since mean is almost same, seems like H0 would be true, let's check in detail

Steps

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Total actual_time aggregated value and segment_actual_time aggregated value are same.
- Alternate Hypothesis (HA) - Total actual_time aggregated value and segment_actual_time aggregated value are not same.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavenes test

STEP-3: Define Test statistics If the assumptions of the T-Test are satisfied, we can proceed with conducting the T-Test for independent samples. Otherwise, we will resort to performing the non-parametric equivalent of the T-Test for independent samples, namely the Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha. We set our alpha to be 0.05

STEP-5: Compare p-value and alpha. Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

Check for Normality

Visual Analysis

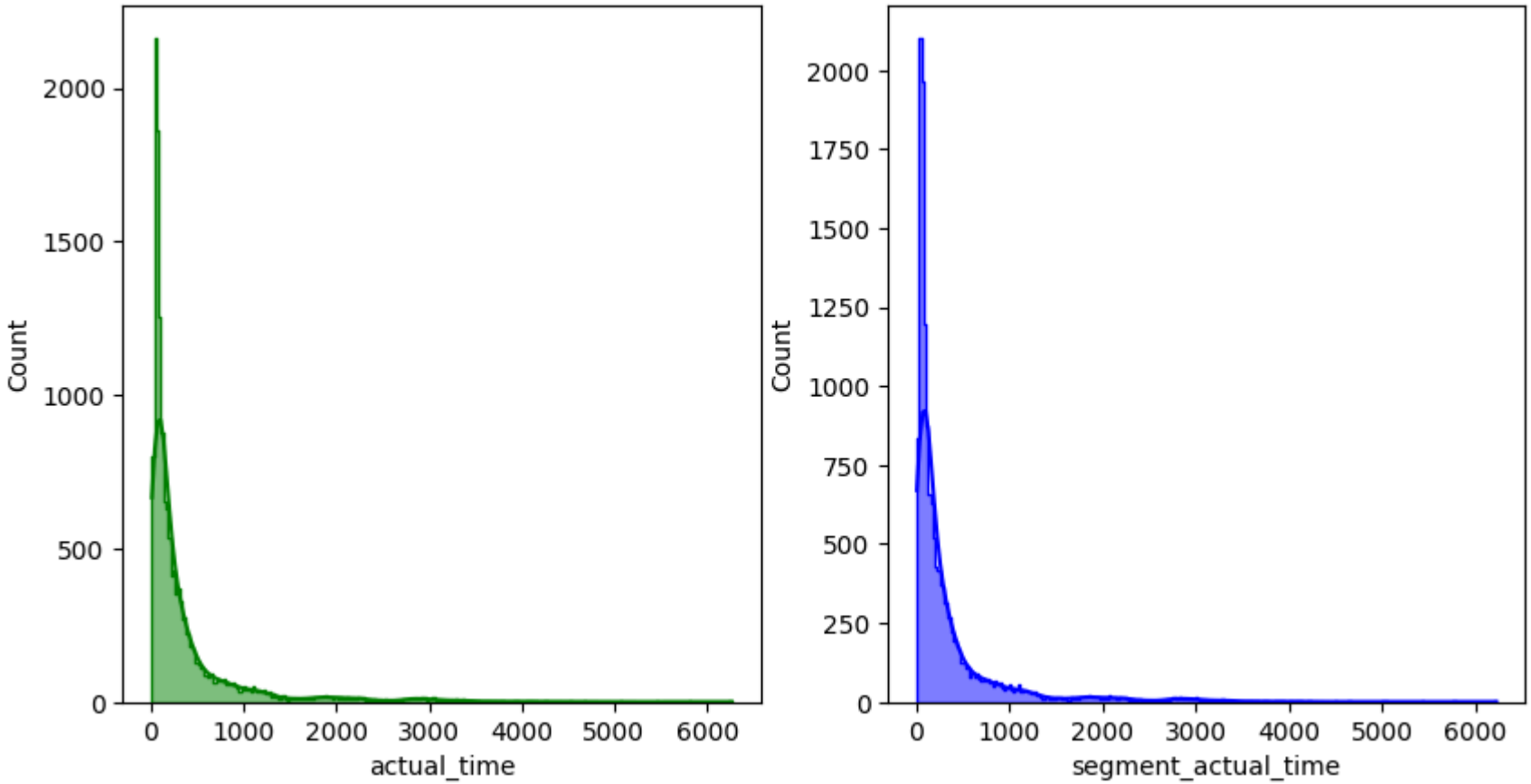
In [954…

```
plt.figure(figsize = (10,5))

plt.subplot(1, 2, 1)
#histogram for Total trip time to check for distribution
sns.histplot(df2['actual_time'], element = 'step', kde = True, color = 'green', label = 'actual_time')

plt.subplot(1, 2, 2)
#histogram for start to end scan to check for distribution
sns.histplot(df2['segment_actual_time'], element = 'step', kde = True, color = 'blue', label = 'segment_actual_time')

plt.show()
```



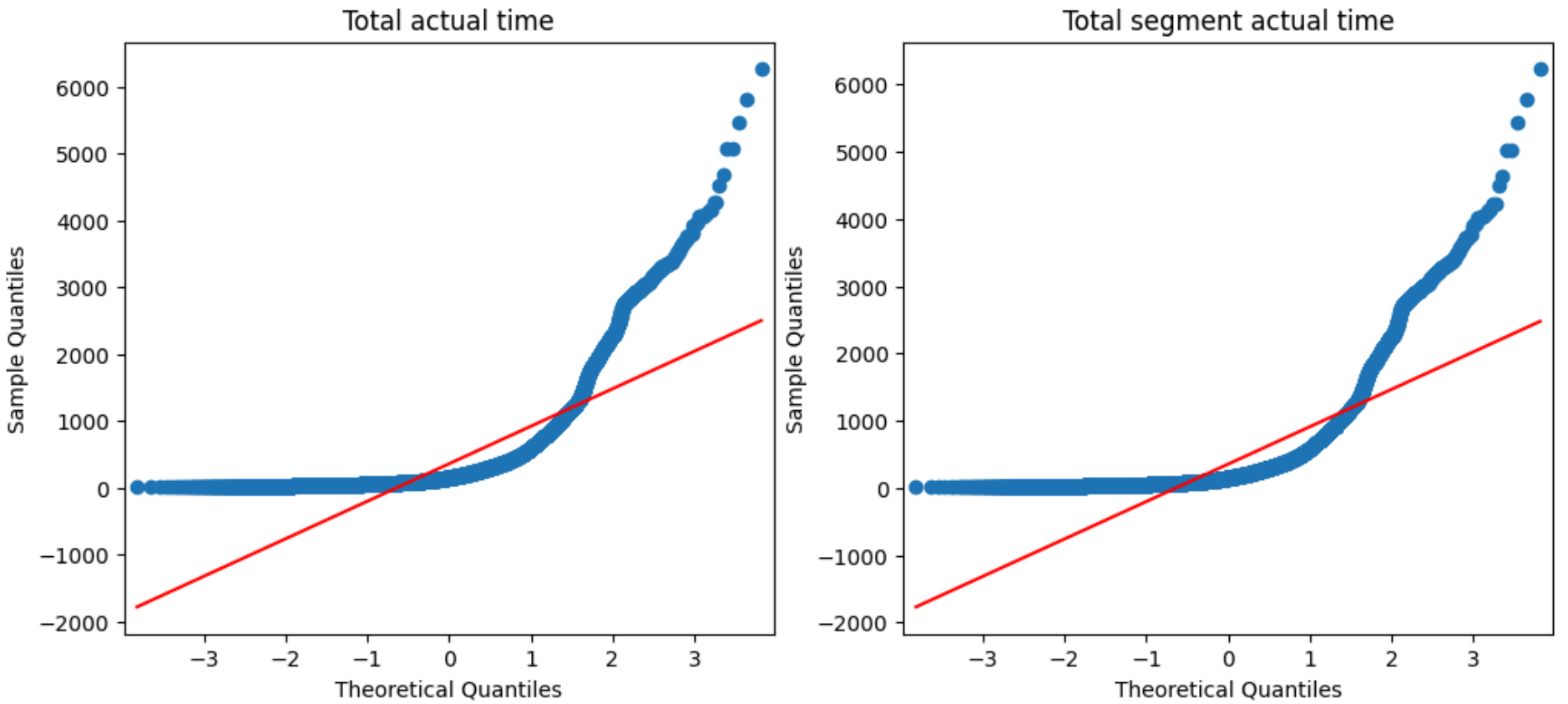
QQ Plot for normality

In [955…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))

ax = fig.add_subplot(121)
#plotting the qq plot for Total trip time
sm.qqplot(df2['actual_time'], line = 's', ax = ax)
plt.title('Total actual time')

ax = fig.add_subplot(122)
#plotting the qq plot for start to scan end time
sm.qqplot(df2['segment_actual_time'], line = 's', ax = ax)
plt.title('Total segment actual time')
plt.show()
```



From the above QQ plot data doesn't seem to be following normal distribution.

Shapiro wilk test for normality

In [956…
```
#taking sample for test for both the columns
total_actual_time = df2['actual_time'].sample(3000)
total_segment_actual_time = df2['segment_actual_time'].sample(3000)
```

For Total actual Time

In [957…
```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
```

```
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking Total trip time sample and running the test on that
test_stat, p_value = shapiro(total_actual_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.045115166904931e-65
Reject the null hypothesis
```

For Total Segment actual time

In [958…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking scan start/end sample and running the test on that
test_stat, p_value = shapiro(total_segment_actual_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
4.871579448423041e-65
Reject the null hypothesis
```

From above shapiro test we can say that sample for Total Actual time and segment time doesn't follow normal distribution

Box-cox to transform data to normal

In [959…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_actual_time)
test_stat, p_value = shapiro(transformed1)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.0308254643901285e-11
Reject the null hypothesis
```

In [960…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_segment_actual_time)
test_stat, p_value = shapiro(transformed2)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
6.125800718312231e-10
Reject the null hypothesis
```

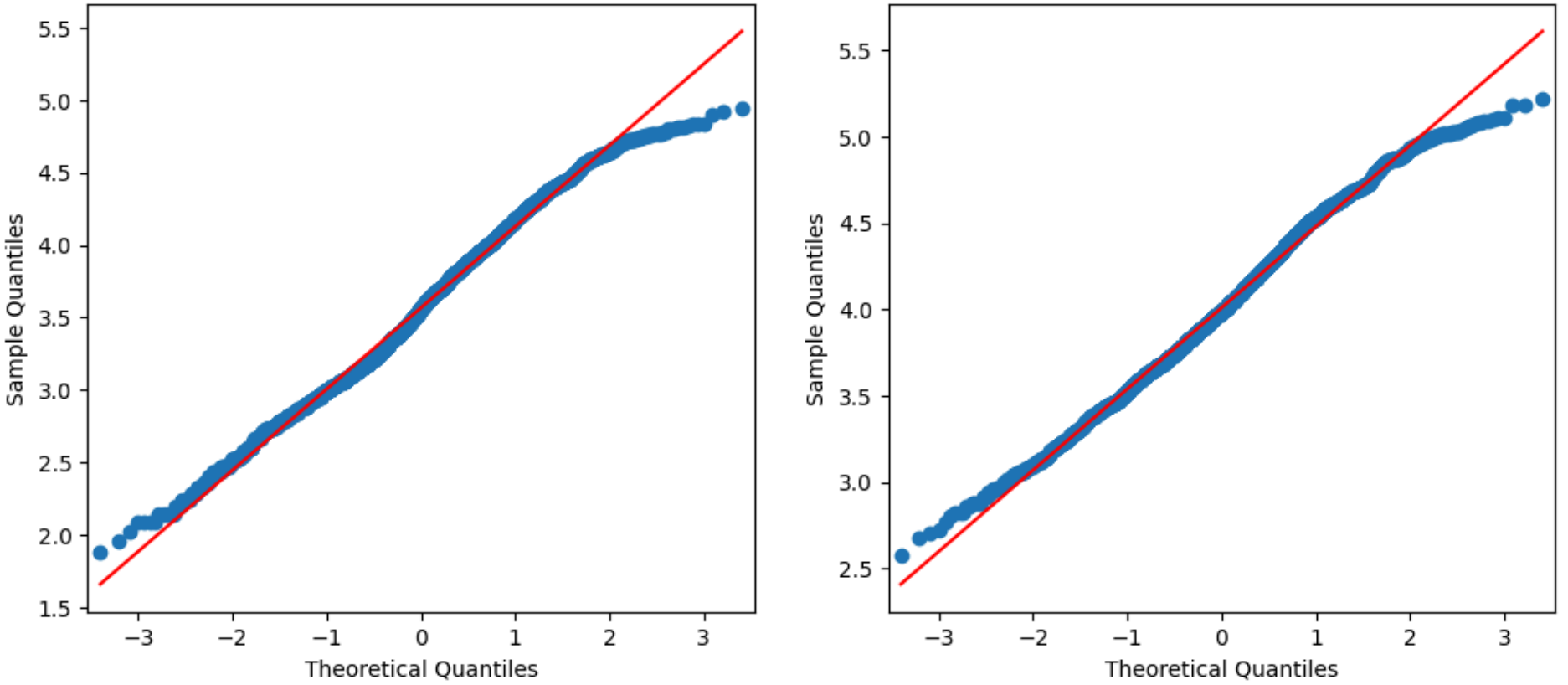
Even after applying the box-cox transformation data is not normal, which we can verfiy through the qq plot

In [961…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)
#plotting the qq plot for the working day
```

```
sm.qqplot(transformed1, line = 's', ax = ax)

ax = fig.add_subplot(122)
#plotting the qq plot for the non-working day
sm.qqplot(transformed2, line = 's', ax = ax)
plt.show()
```



Levene test for homegenity of variance

In [962…

```
#checking the homgenity of the variance using the levene test

from scipy.stats import levene

# Null Hypothesis Ho - homogeneous variance
# Alternate Hypothesis Ha - Non- homogenous variance

test_stat, p_value = levene(total_actual_time, total_segment_actual_time)
```

```
print(p_value)

if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.1663246855991939
Accept the null hypothesis
```

Samples have homogenous variance

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

Hypothesis test - Nonparametric

In [963…

```
#performing Mann-whiteney test since the data is not normal

# Null Hypothesis ( H0 ) - Total Actual Time and Total segment actual time are same.
# Alternate Hypothesis ( HA ) - Total Actual Time and Total segment actual time are not same or they are different.
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu
test_stat, p_value = mannwhitneyu(total_actual_time, total_segment_actual_time)

print(p_value)
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.7786418738130292
Accept the null hypothesis
```

Since p-value is > 0.05 we can say that H0 is accepeted, meaning Total actual time and total segment time are similar, which was visible in mean value in initial decribe function output.

- The features 'Total actual Time' and 'total segment actual time' are not statistically different.

Between aggregated values of OSRM time and segment OSRM time

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)

High level summary

In [964…
```
df2[['osrm_time', 'segment_osrm_time']].describe()
```

Out[964…

	osrm_time	segment_osrm_time
count	14817.000000	14817.000000
mean	161.384018	180.921172
std	271.360995	314.485624
min	6.000000	6.000000
25%	29.000000	31.000000
50%	60.000000	65.000000
75%	168.000000	185.000000
max	2032.000000	2564.000000

since mean 161 vs 181 , seems like H0 might be rejected, but lets see whether statistically it is significant or not. since difference is not big but not small as well.

Steps

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Total osrm_time and segment_osrm_time aggregated value are same.
- Alternate Hypothesis (HA) - Total osrm_time and segment_osrm_time aggregated value are same are not same.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavenes test

STEP-3: Define Test statistics If the assumptions of the T-Test are satisfied, we can proceed with conducting the T-Test for independent samples. Otherwise, we will resort to performing the non-parametric equivalent of the T-Test for independent samples, namely the Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha. We set our alpha to be 0.05

STEP-5: Compare p-value and alpha. Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

Check for Normality

Visual Analysis

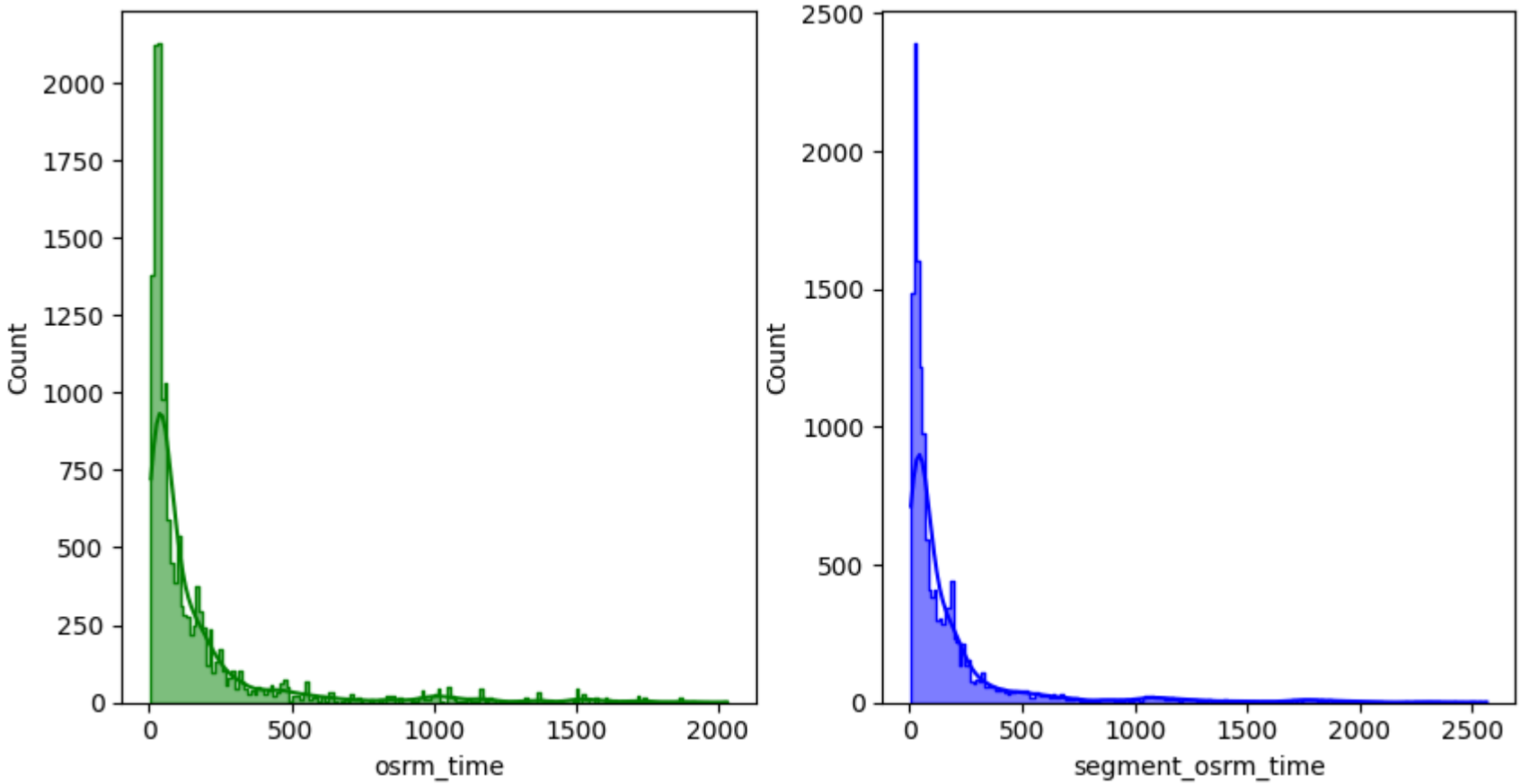
In [965…

```
plt.figure(figsize = (10,5))

plt.subplot(1, 2, 1)
#histogram for Total trip time to check for distribution
sns.histplot(df2['osrm_time'], element = 'step', kde = True, color = 'green', label = 'actual_time')

plt.subplot(1, 2, 2)
#histogram for start to end scan to check for distribution
sns.histplot(df2['segment_osrm_time'], element = 'step', kde = True, color = 'blue', label = 'segment_actual_time')

plt.show()
```



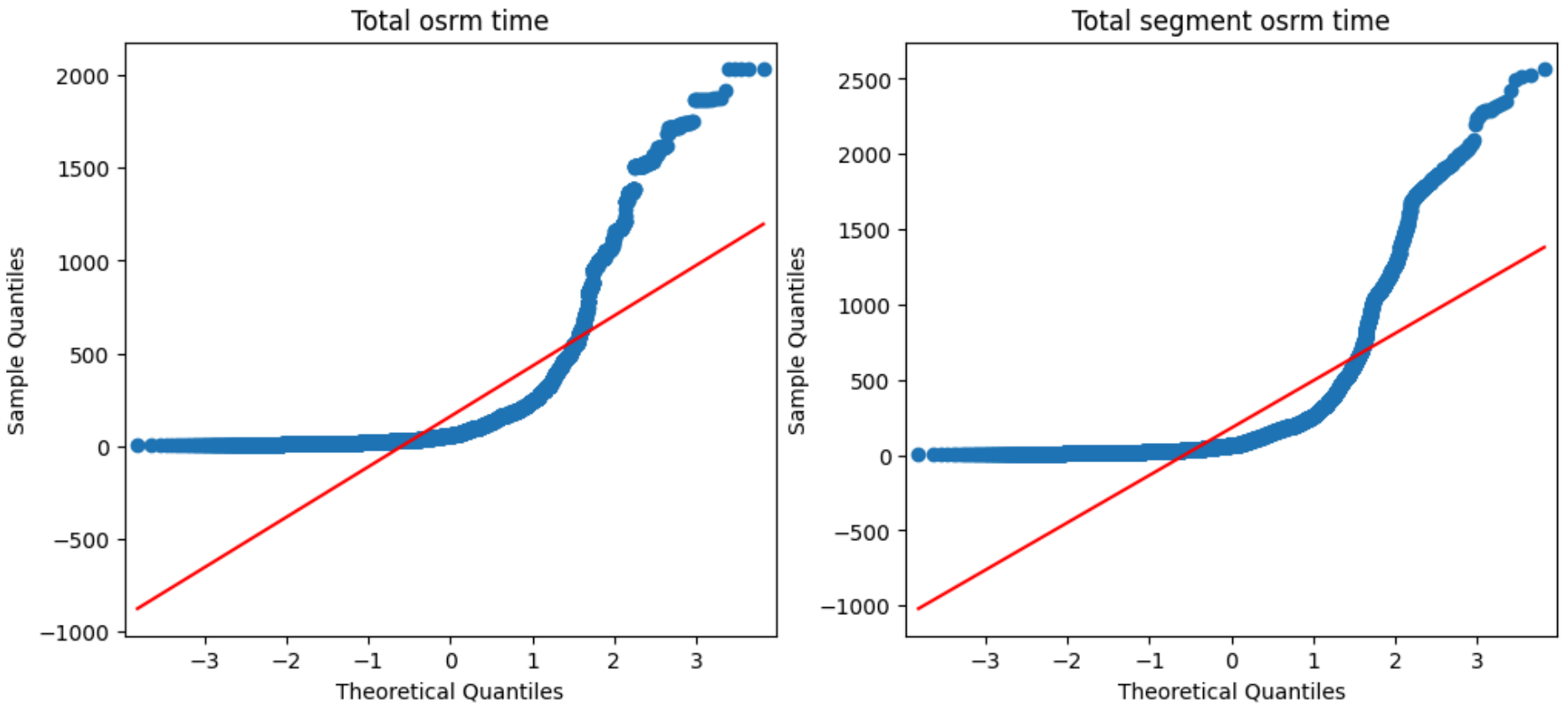
QQ Plot for normality

In [966…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))

ax = fig.add_subplot(121)
#plotting the qq plot for Total trip time
sm.qqplot(df2['osrm_time'], line = 's', ax = ax)
plt.title('Total osrm time')

ax = fig.add_subplot(122)
#plotting the qq plot for start to scan end time
sm.qqplot(df2['segment_osrm_time'], line = 's', ax = ax)
plt.title('Total segment osrm time')
plt.show()
```



From the above QQ plot data doesn't seem to be following normal distribution.

Shapiro wilk test for normality

In [967…
```
#taking sample for test for both the columns
total_osrm_time = df2['osrm_time'].sample(3000)
total_segment_osrm_time = df2['segment_osrm_time'].sample(3000)
```

For Total osrm Time

In [968…
```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
```

```
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking Total trip time sample and running the test on that
test_stat, p_value = shapiro(total_osrm_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
1.013213406539735e-66
Reject the null hypothesis
```

For Total segment osrm time

In [969…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking scan start/end sample and running the test on that
test_stat, p_value = shapiro(total_segment_osrm_time)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
9.664489783740285e-67
Reject the null hypothesis
```

From above shapiro test we can say that sample for Total osrm time and segment osrm time doesn't follow normal distribution

Box-cox to transform data to normal

In [970…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_osrm_time)
test_stat, p_value = shapiro(transformed1)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
3.128712255273457e-17
Reject the null hypothesis
```

In [971…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_segment_osrm_time)
test_stat, p_value = shapiro(transformed2)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
6.125800718312231e-10
Reject the null hypothesis
```

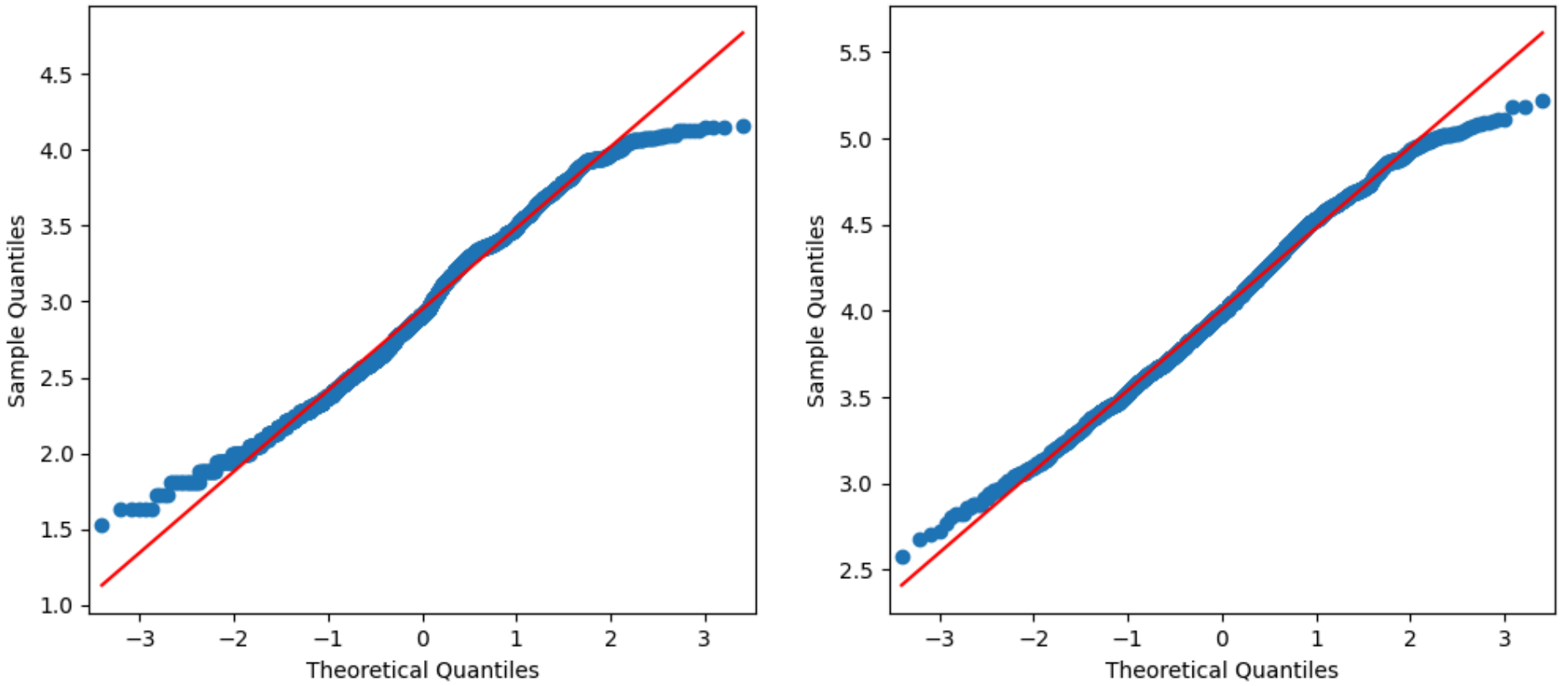
Even after applying the box-cox transformation data is not normal, which we can verfiy through the qq plot

In [972…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)
#plotting the qq plot for the working day
```

```
sm.qqplot(transformed1, line = 's', ax = ax)

ax = fig.add_subplot(122)
#plotting the qq plot for the non-working day
sm.qqplot(transformed2, line = 's', ax = ax)
plt.show()
```



Levene test for homegenity of variance

In [973…

```
#checking the homgenity of the variance using the levene test

from scipy.stats import levene

# Null Hypothesis Ho - homogeneous variance
# Alternate Hypothesis Ha - Non- homogenous variance

test_stat, p_value = levene(total_osrm_time, total_segment_osrm_time)
```

```
print(p_value)

if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.00035092023812268 63
Reject the null hypothesis
```

Samples does't have homogenous variance

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

Hypothesis test - Nonparametric

In [974…

```
#performing Mann-whiteney test since the data is not normal

# Null Hypothesis ( H0 ) - Total Actual Time and Total segment actual time are same.
# Alternate Hypothesis ( HA ) - Total Actual Time and Total segment actual time are not same or they are different.
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu
test_stat, p_value = mannwhitneyu(total_osrm_time, total_segment_osrm_time)

print(p_value)
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.0007081520445951718
Reject the null hypothesis
```

Since p-value is < 0.05 we can say that H0 is rejected, meaning Total osrm time and total segment osrm time are not similar.

- The features 'Total osrm Time' and 'total segment osrm time' are statistically different.

Between aggregated values of OSRM distance and Segment OSRM distance

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)

High level summary

In [975…
```
df2[['osrm_distance', 'segment_osrm_distance']].describe()
```

Out[975…

	osrm_distance	segment_osrm_distance
count	14817.000000	14817.000000
mean	204.344689	223.164000
std	370.395573	416.547252
min	9.072900	9.072900
25%	30.819200	32.654500
50%	65.618800	70.154400
75%	208.475000	218.710200
max	2840.081000	3523.632400

since mean 204 vs 223 , seems like H0 might be accepted

Steps

STEP-1 : Set up Null Hypothesis

- Null Hypothesis (H0) - Total osrm_distance and segment_osrm_distance aggregated value are same.
- Alternate Hypothesis (HA) - Total osrm_distance and segment_osrm_distance aggregated value are not same.

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using QQ Plot
- Homogeneity of Variances using Lavenes test

STEP-3: Define Test statistics If the assumptions of the T-Test are satisfied, we can proceed with conducting the T-Test for independent samples. Otherwise, we will resort to performing the non-parametric equivalent of the T-Test for independent samples, namely the Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha. We set our alpha to be 0.05

STEP-5: Compare p-value and alpha. Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0
- p-val < alpha : Reject H0

Check for Normality

Visual Analysis

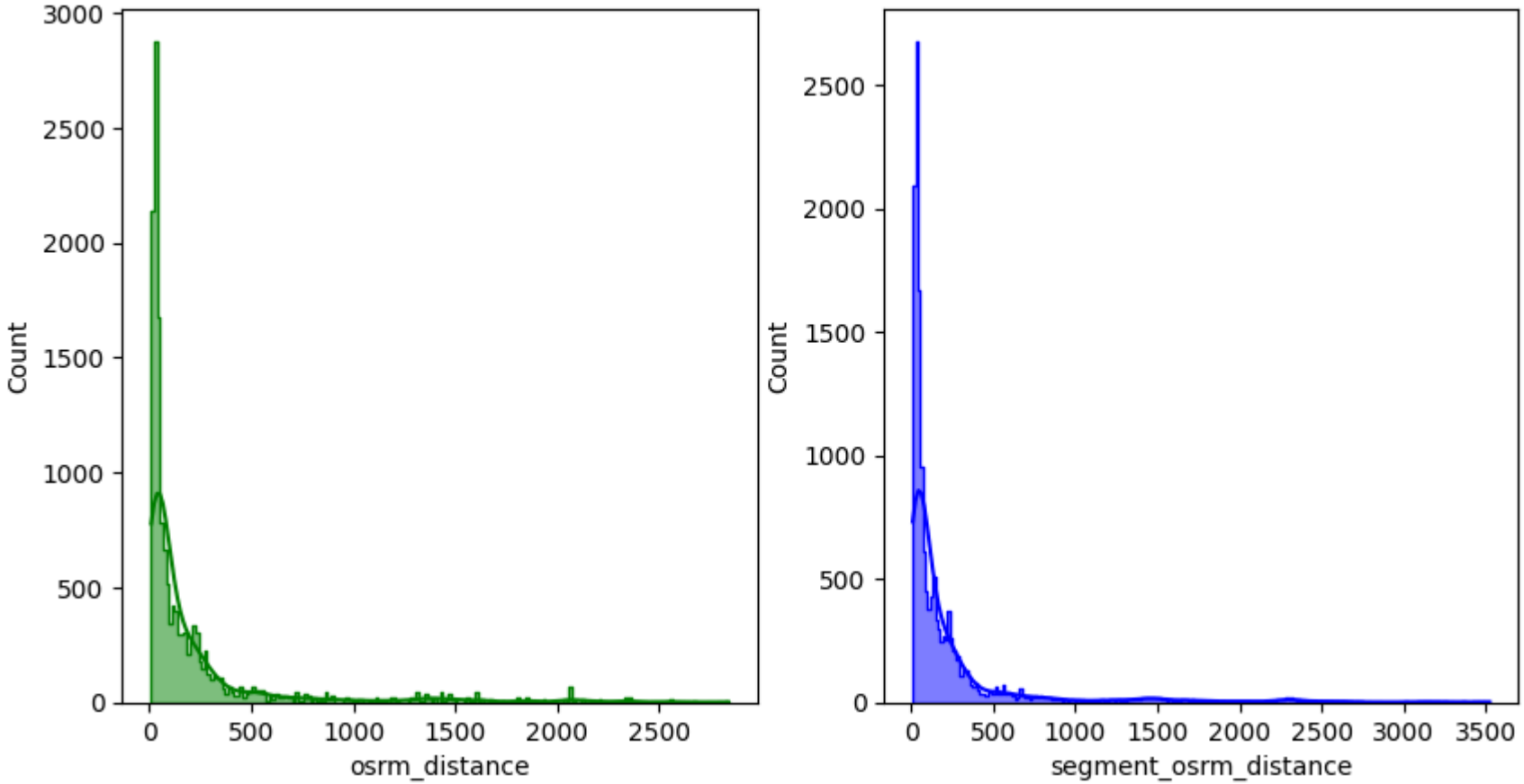
In [976…

```
plt.figure(figsize = (10,5))

plt.subplot(1, 2, 1)
#histogram for Total trip time to check for distribution
sns.histplot(df2['osrm_distance'], element = 'step', kde = True, color = 'green', label = 'actual_time')

plt.subplot(1, 2, 2)
#histogram for start to end scan to check for distribution
sns.histplot(df2['segment_osrm_distance'], element = 'step', kde = True, color = 'blue', label = 'segment_actual_time')

plt.show()
```



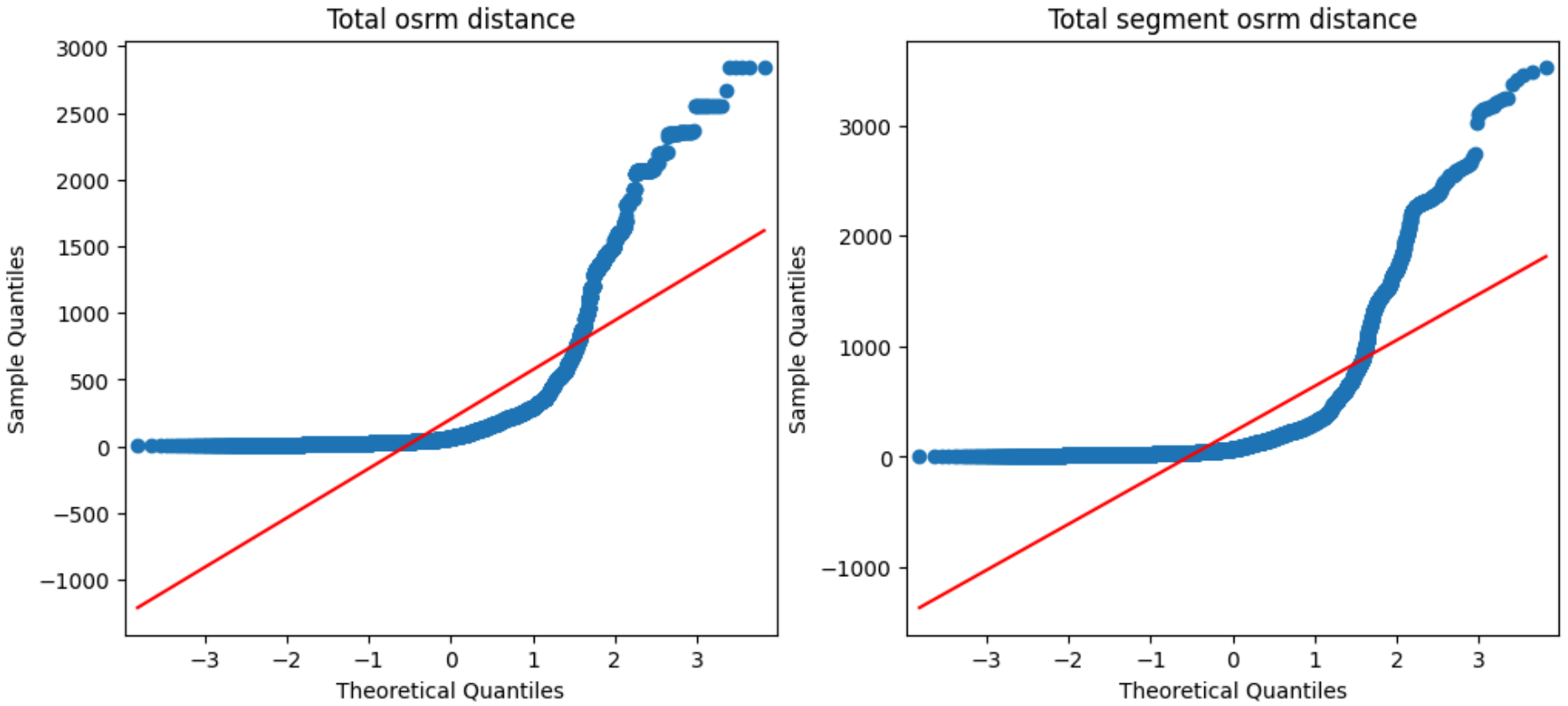
QQ Plot for normality

In [977…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))

ax = fig.add_subplot(121)
#plotting the qq plot for Total trip time
sm.qqplot(df2['osrm_distance'], line = 's', ax = ax)
plt.title('Total osrm distance')

ax = fig.add_subplot(122)
#plotting the qq plot for start to scan end time
sm.qqplot(df2['segment_osrm_distance'], line = 's', ax = ax)
plt.title('Total segment osrm distance')
plt.show()
```



From the above QQ plot data doesn't seem to be following normal distribution.

Shapiro wilk test for normality

In [978…
```
#taking sample for test for both the columns
total_osrm_distance = df2['osrm_distance'].sample(3000)
total_segment_osrm_distance = df2['segment_osrm_distance'].sample(3000)
```

For Total osrm Time

In [979…
```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
```

```
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking Total trip time sample and running the test on that
test_stat, p_value = shapiro(total_osrm_distance)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
9.674626101899342e-68
Reject the null hypothesis
```

For Total segment osrm time

In [980…

```
#H0- assuming sample follows normal distribution
#Ha -Reject H0

#importing the required library for the Shapiro test
from scipy.stats import shapiro

# fixing the random seed, so the sample taken remains same if we re-run
np.random.seed(42)

# taking scan start/end sample and running the test on that
test_stat, p_value = shapiro(total_segment_osrm_distance)


print(p_value)
#checking for hypothesis
if p_value < 0.05: #assuming alpha as 5%
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
7.27504526963356e-68
Reject the null hypothesis
```

From above shapiro test we can say that sample for Total osrm distance and segment osrm distance doesn't follow normal distribution

Box-cox to transform data to normal

In [981…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_osrm_distance)
test_stat, p_value = shapiro(transformed1)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
6.120070861158217e-21
Reject the null hypothesis
```

In [982…

```
from scipy.stats import boxcox

transformed1, best_lambda = boxcox(total_segment_osrm_distance)
test_stat, p_value = shapiro(transformed2)

print(p_value)

#checking for hypothesis
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
6.125800718312231e-10
Reject the null hypothesis
```

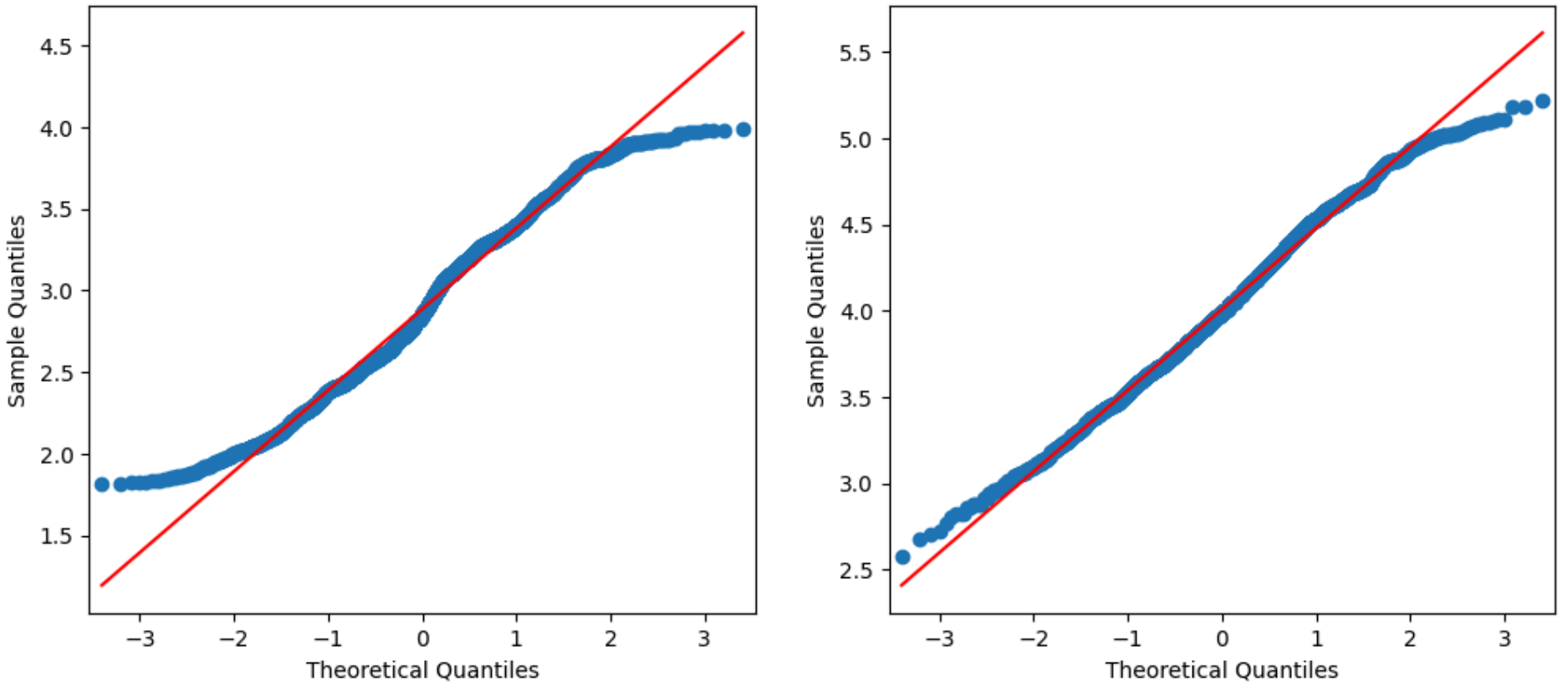
Even after applying the box-cox transformation data is not normal, which we can verfiy through the qq plot

In [983…

```
import statsmodels.api as sm
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(121)
#plotting the qq plot for the working day
```

```
sm.qqplot(transformed1, line = 's', ax = ax)

ax = fig.add_subplot(122)
#plotting the qq plot for the non-working day
sm.qqplot(transformed2, line = 's', ax = ax)
plt.show()
```



Levene test for homegenity of variance

In [984…

```
#checking the homgenity of the variance using the levene test

from scipy.stats import levene

# Null Hypothesis Ho - homogeneous variance
# Alternate Hypothesis Ha - Non- homogenous variance

test_stat, p_value = levene(total_osrm_distance, total_segment_osrm_distance)
```

```
print(p_value)

if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.004595711687792748
Reject the null hypothesis
```

Samples does't have homogenous variance

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

Hypothesis test - Nonparametric

In [985…

```
#performing Mann-whiteney test since the data is not normal

# Null Hypothesis ( H0 ) - Total osrm distance and Total segment osrm distance are same.
# Alternate Hypothesis ( HA ) - Total osrm distance and Total segment osrm distance are not same.
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu
test_stat, p_value = mannwhitneyu(total_osrm_distance, total_segment_osrm_distance)

print(p_value)
if p_value < 0.05:
  print('Reject the null hypothesis')
else:
  print('Accept the null hypothesis')
```

```
0.0022543784321766877
Reject the null hypothesis
```

Since p-value is < 0.05 we can say that H0 is rejected, meaning Total osrm distance and total osrm segment distance are not similar.

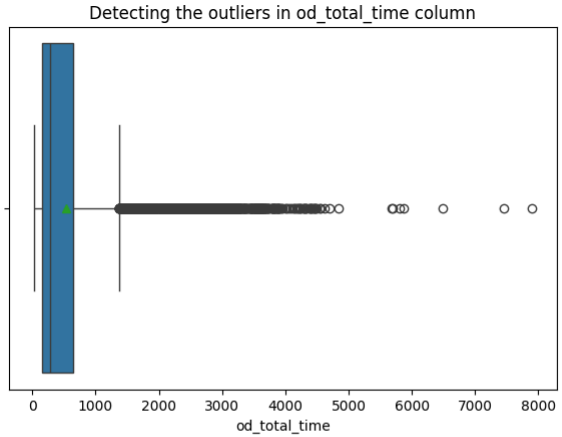
- The features 'Total osrm distance' and 'Total osrm segment distance' are statistically different.

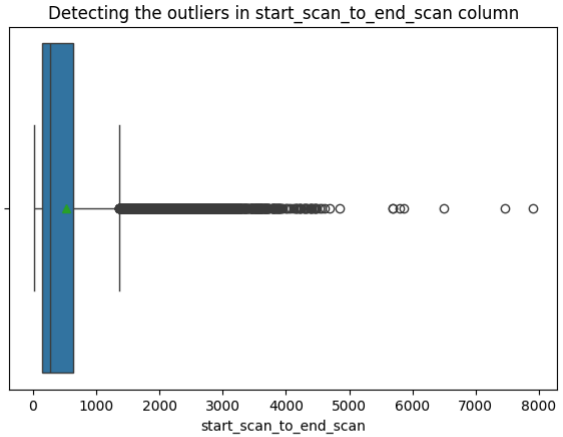
Outlier Detection

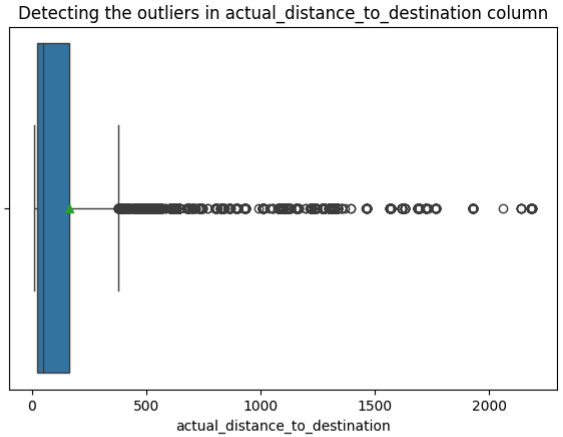
Find outliers in the numerical variables (we might find outliers in almost all the variables), and check it using visual analysis

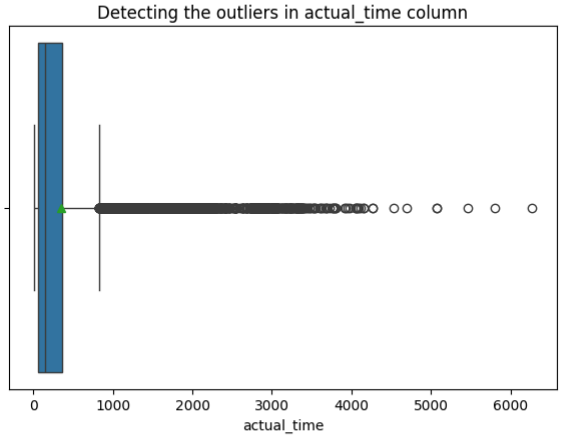
In [986…

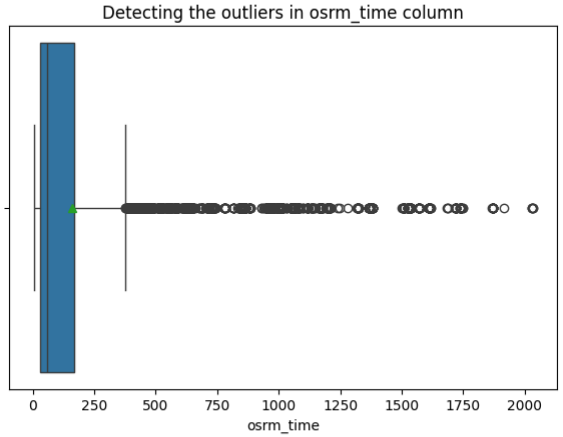
```
#taking all the numerical columns for the outlier detection which are he
numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance_to_destination',
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']
count = 1
plt.figure(figsize = (24, 16))
for i in numerical_columns:
  plt.subplot(3, 3 ,count)
  sns.boxplot(x = df2[i], data = df2, showmeans = True) #plotting the box plot with mean showing on box plot
  plt.title(f'Detecting the outliers in {i} column', size = 12)
  plt.plot()
  count += 1
```

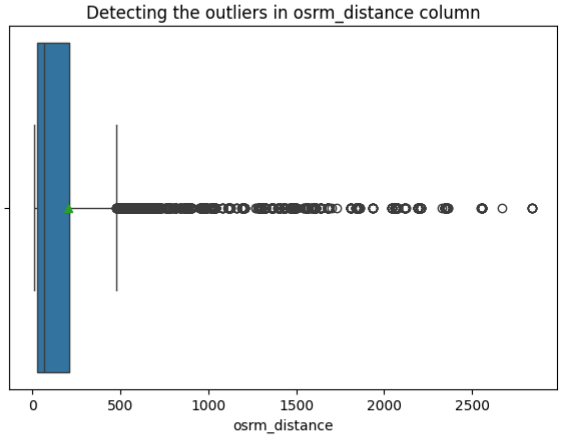

Detecting the outliers in od_total_time column
0 1000 2000 3000 4000 5000 6000 7000 8000
od_total_time

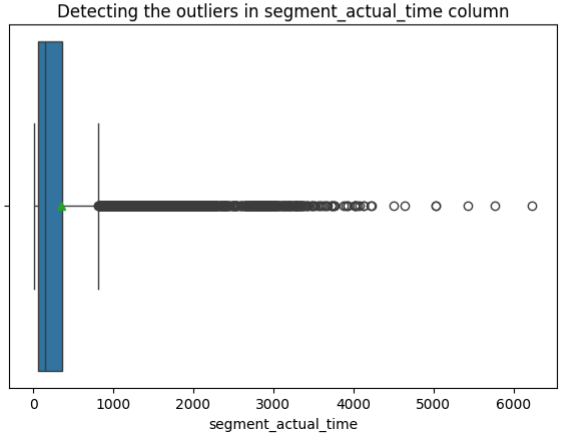

Detecting the outliers in start_scan_to_end_scan column
0 1000 2000 3000 4000 5000 6000 7000 8000
start_scan_to_end_scan

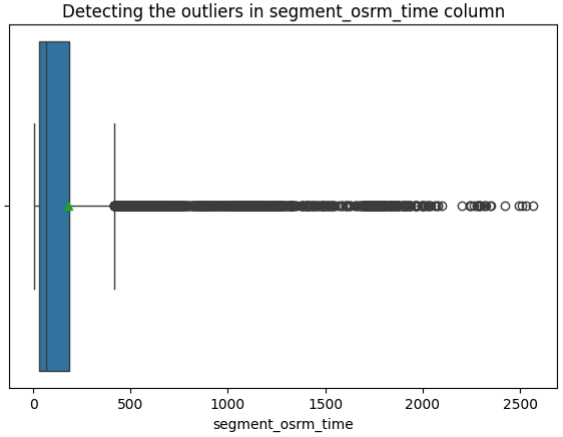

Detecting the outliers in actual_distance_to_destination column
0 500 1000 1500 2000
actual_distance_to_destination

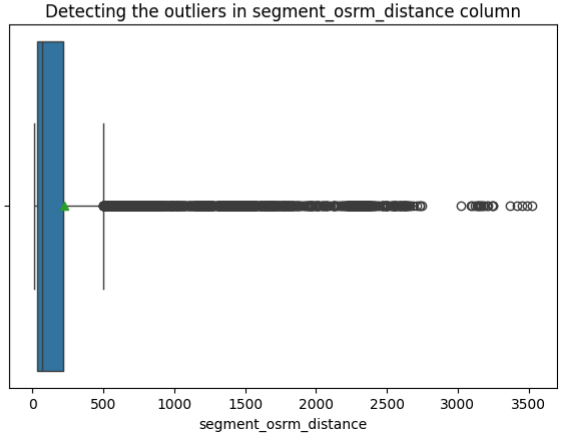

Detecting the outliers in actual_time column
0 1000 2000 3000 4000 5000 6000
actual_time


Detecting the outliers in osrm_time column
0 250 500 750 1000 1250 1500 1750 2000
osrm_time


Detecting the outliers in osrm_distance column
0 500 1000 1500 2000 2500
osrm_distance


Detecting the outliers in segment_actual_time column
0 1000 2000 3000 4000 5000 6000
segment_actual_time


Detecting the outliers in segment_osrm_time column
0 500 1000 1500 2000 2500
segment_osrm_time


Detecting the outliers in segment_osrm_distance column
0 500 1000 1500 2000 2500 3000 3500
segment_osrm_distance

In [987…

```
#Detecting outliers

for i in numerical_columns:
  Q1 = np.quantile(df2[i], 0.25)
  Q3 = np.quantile(df2[i], 0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  df_outliers = df2.loc[( df2[i] < lower_bound) | (df2[i] > upper_bound)] #this df will represent values with outliers

  print(f'Column :{i}')
  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 : {df_outliers.shape[0]}')
  print('-----------------------------------')
```

```
Column :od_total_time
Q1 : 149.93
Q3 : 638.2
IQR : 488.27000000000004
lower_bound : -582.4750000000001
upper_bound : 1370.605
Number of outliers : 1266
-----------------------------------
Column :start_scan_to_end_scan
Q1 : 149.0
Q3 : 637.0
IQR : 488.0
lower_bound : -583.0
upper_bound : 1369.0
Number of outliers : 1267
-----------------------------------
Column :actual_distance_to_destination
Q1 : 22.83723905859321
Q3 : 164.58320763841138
IQR : 141.74596857981817
lower_bound : -189.78171381113404
upper_bound : 377.2021605081386
Number of outliers : 1449
```

```
----------------------------------
Column :actual_time
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
lower_bound : -387.5
upper_bound : 824.5
Number of outliers : 1643
----------------------------------
Column :osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
lower_bound : -179.5
upper_bound : 376.5
Number of outliers : 1517
----------------------------------
Column :osrm_distance
Q1 : 30.8192
Q3 : 208.475
IQR : 177.6558
lower_bound : -235.6645
upper_bound : 474.9587
Number of outliers : 1524
----------------------------------
Column :segment_actual_time
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
lower_bound : -385.5
upper_bound : 818.5
Number of outliers : 1643
----------------------------------
Column :segment_osrm_time
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
lower_bound : -200.0
upper_bound : 416.0
Number of outliers : 1492
----------------------------------
Column :segment_osrm_distance
Q1 : 32.6545
Q3 : 218.7102
```

```
IQR : 186.0557
lower_bound : -246.42905000000002
upper_bound : 497.79375
Number of outliers : 1549
----------------------------------
```

Outlier Treatment

Now that we know there are significant outliers in the data (upto 1643 in segment_actual_time), we can filter out the outliers from dataframe.

Handing the outliers using the IQR method.

In [988…

```
# Select only numeric columns
df2_numeric = df2.select_dtypes(include=[float, int])

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df2_numeric.quantile(0.25)
Q3 = df2_numeric.quantile(0.75)
IQR = Q3 - Q1  # Interquartile range

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out rows with outliers
df2_filtered = df2[~((df2_numeric < lower_bound) | (df2_numeric > upper_bound)).any(axis=1)]

#print the no of row before and after outlier treatment
print(f'Number of rows before outlier treatment : {df2.shape[0]}')
print(f'Number of rows after outlier treatment : {df2_filtered.shape[0]}')
print('No of rows removed during outier treatment: ', df2.shape[0] - df2_filtered.shape[0])
```

```
Number of rows before outlier treatment : 14817
Number of rows after outlier treatment : 12759
No of rows removed during outier treatment:  2058
```

Encoding

One Hot Encoding

Do one-hot encoding of categorical variables (like route_type and data column)

Route type and Data column

In [989…

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder() #created a object of the class LabelEncoder

#onehot encoding for 'route_type'
df2['route_type'] = label_encoder.fit_transform(df2['route_type'])

#onehot encoding for 'data' column
df2['data'] = label_encoder.fit_transform(df2['data'])
```

In [990…

```
df2['route_type'].value_counts(normalize = True)*100
```

Out[990…

route_type	proportion
0	60.120132
1	39.879868

dtype: float64

We can see that route_type is converted in 0 and 1 instead of carting and FTL

In [991…

```
df2['data'].value_counts(normalize = True)*100
```

Out[991…

data	proportion
1	71.903894
0	28.096106

dtype: float64

We can see that column 'data' is converted in 0 and 1 instead of testing and training. (which one is 0 and which one is 1 can be verified through old and new df)

Normalization/ Standardization

Check for Null values

In [992…

```
#checking for Null values in the numeric columns where Normalization/Standardization is to be performed.
df2.isna().sum()
```

Out[992…

	0
trip_uuid	0
source_center	0
destination_center	0
data	0
route_type	0
trip_creation_time	0
source_name	10
destination_name	8
od_total_time	0
start_scan_to_end_scan	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0
source_state	10
source_city	0
destination_state	8

	0
destination_city	0
trip_creation_date	0
trip_creation_year	0
trip_creation_month	0
trip_creation_day	0
trip_creation_week	0
trip_creation_hour	0
trip_type	0

dtype: int64

No null values in numerical columns, otherwise we need to handle those missing values first using fillna() via mean/median whichever works best based on the data.

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [993…
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

#Normalization
scaler = MinMaxScaler() #craeting an object of this MinMaxScaler() class
```

In [994…
```
#this works only for numeric columns with integer or float values
```

Performing Normalization since these time and distance values should greater than zero, in standardization it might go between (-1, 1)

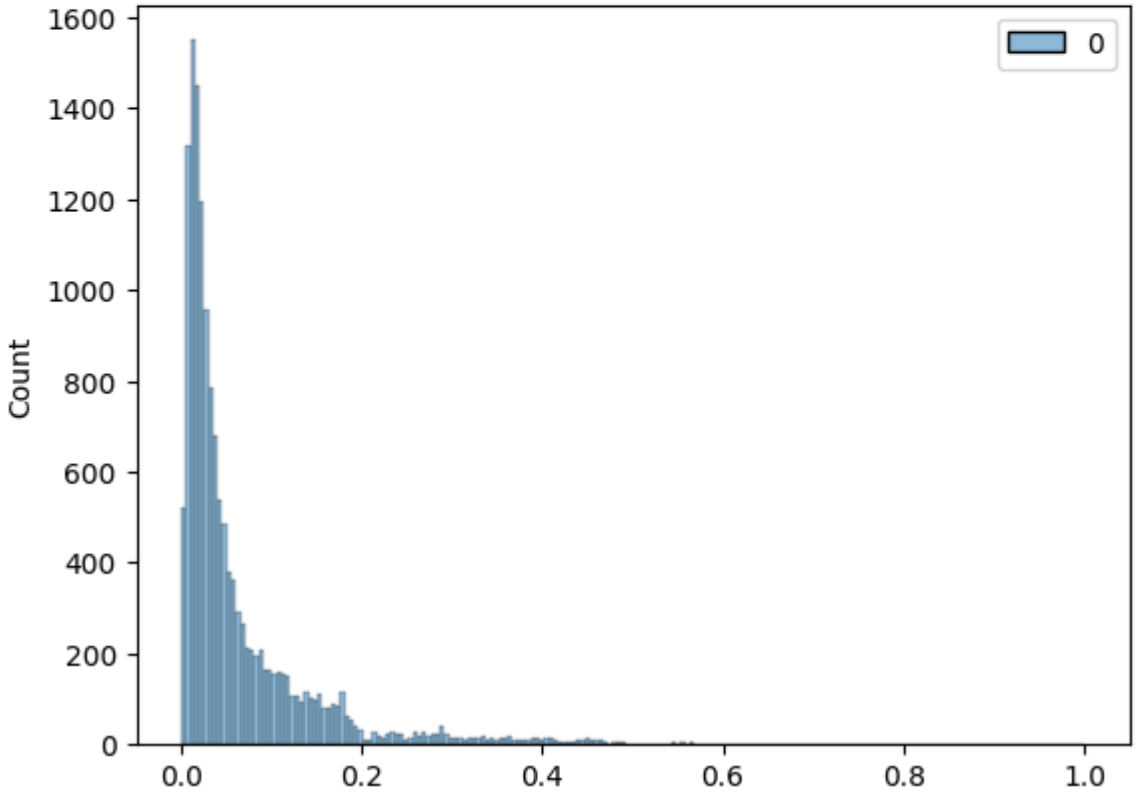
od_total_time

In [995…
```
#Normalization
scaled = scaler.fit_transform(df2['od_total_time'].to_numpy().reshape(-1,1))

# df2['od_total_time'].to_numpy(): This converts the column 'od_total_time' from a pandas DataFrame to a NumPy array.
# .reshape(-1, 1): Reshapes the array to a 2D shape (necessary for fit_transform(), as it expects 2D data).
# scaler.fit_transform(): Scales the values using the chosen scaler (e.g., MinMaxScaler or StandardScaler).
```

```
sns.histplot(scaled)
plt.title(f"Normalized {df2['od_total_time']} column")
plt.show()
```

```
Normalized 0      2260.11
           1       181.61
           2      3934.36
           3       100.49
           4       718.34
                    ...
       14812       258.03
       14813        60.59
       14814       422.12
       14815       348.52
       14816       354.40
Name: od_total_time, Length: 14817, dtype: float64 column
```

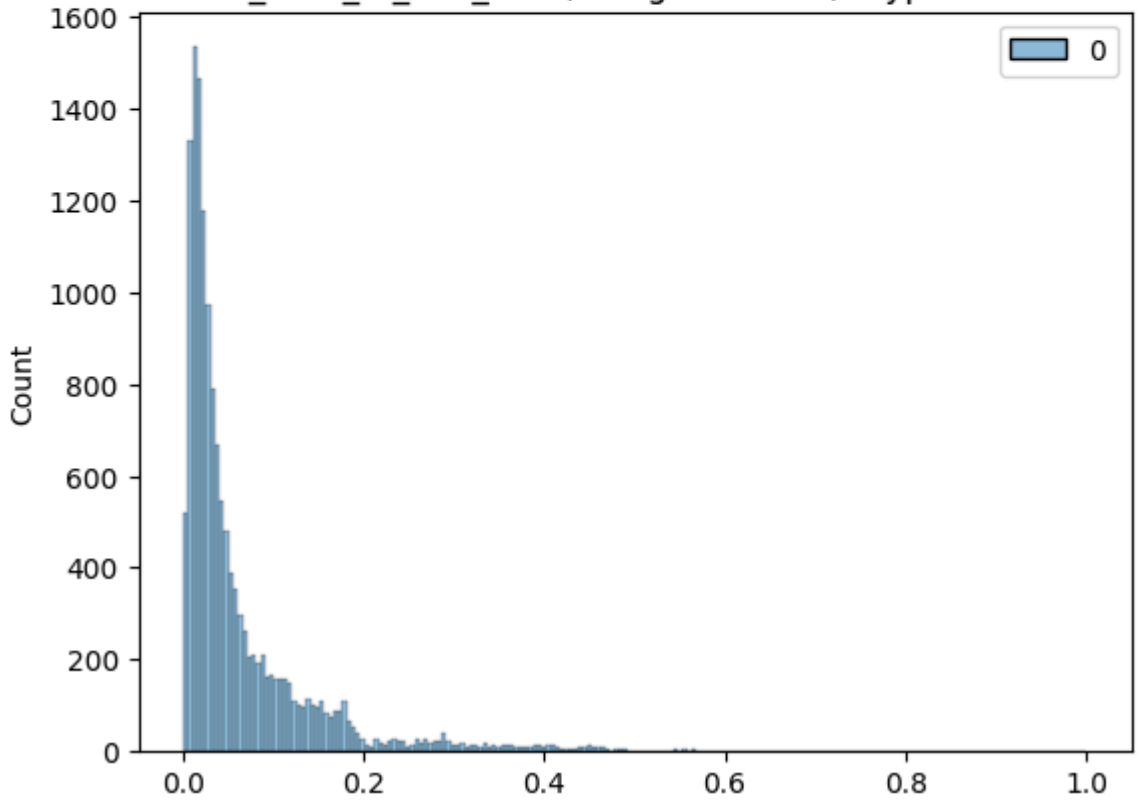


start_scan_to_end_scan

In [996…

```
#Normalization
scaled = scaler.fit_transform(df2['start_scan_to_end_scan'].to_numpy().reshape(-1,1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['start_scan_to_end_scan']} column")
plt.show()
```

```
Normalized 0       2259.0
           1        180.0
           2       3933.0
           3        100.0
           4        717.0
                     ...
       14812        257.0
       14813         60.0
       14814        421.0
       14815        347.0
       14816        353.0
Name: start_scan_to_end_scan, Length: 14817, dtype: float64 column
```

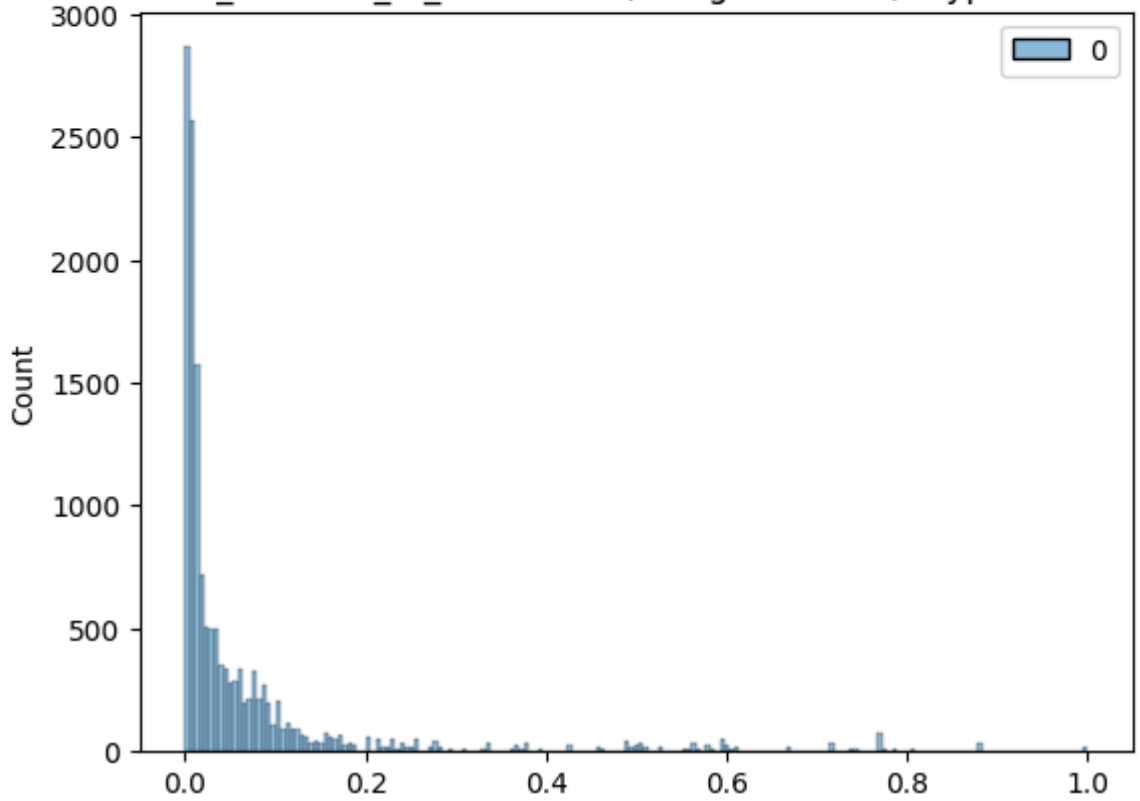


actual_distance_to_destination

In [997…

```
#Normalization
scaled = scaler.fit_transform(df2['actual_distance_to_destination'].to_numpy().reshape(-1,1))
sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_distance_to_destination']} column")
plt.show()
```

```
Normalized 0       824.732854
           1        73.186911
           2      1927.404273
           3        17.175274
           4       127.448500
                      ...
       14812        57.762332
       14813        15.513784
       14814        38.684839
       14815       134.723836
       14816        66.081533
Name: actual_distance_to_destination, Length: 14817, dtype: float64 column
```



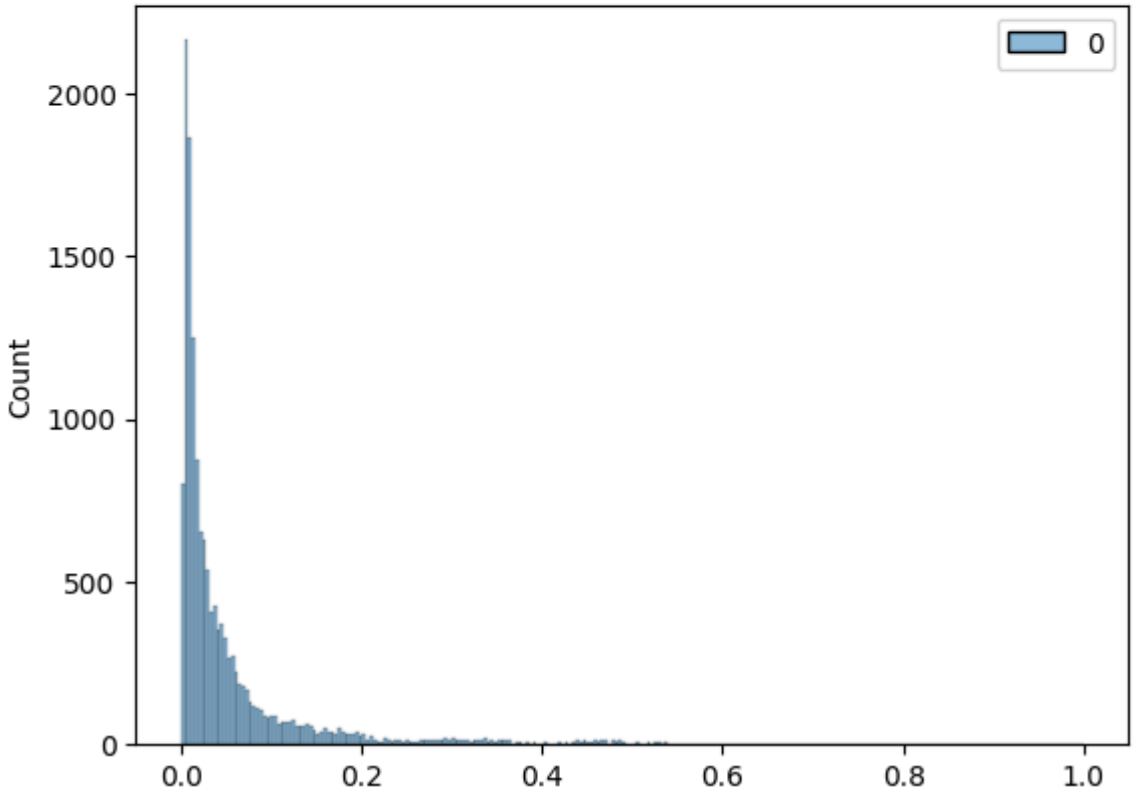
actual_time

In [998…

```
#Normalization
scaled = scaler.fit_transform(df2['actual_time'].to_numpy().reshape(-1,1))

sns.histplot(scaled)
plt.title(f"Normalized {df2['actual_time']} column")
plt.show()
```

```
Normalized 0       1562.0
           1        143.0
           2       3347.0
           3         59.0
           4        341.0
                     ...
       14812         83.0
       14813         21.0
       14814        282.0
       14815        264.0
       14816        275.0
Name: actual_time, Length: 14817, dtype: float64 column
```



Business Insights & Recommendations

Insights

- The data spans from '2018-09-12 00:00:16' to '2018-10-08 03:00:24', involving 14,817 unique trip IDs, 1,508 unique source centers, 1,481 unique destination centers, 716 unique source cities, and 840 unique destination cities.
- Testing data predominates over training data.
- Most of the data is for testing than for training, Most common route type is Carting (Carting 69.1% and Full Truck load 39.9%).
- The names of 14 unique location ids are missing in the data for both source and destination.
- For some cities like Bengaluru and Bangalore both old/ new names are used.
- Maharashtra, karnataka and Haryana contributes to 44% to total trip as source states followed by Tamilnadu and Telangana,
- Mumbai, karnataka and Haryana contributes to 45% to total trip as destination states followed by Tamilnadu and UttarPradesh.
- Bengaluru has the highest number of originated trips, followed by Gurgaon, Mumbai, Bhiwandi and Delhi, indicating a strong seller base in these cities.
- Bengaluru top the destination city list, followed by Mumbai, Gurgaon, Delhi and Hyderabad. Highlighting strong demand from these cities.
- Most of the trips starts after 12 PM and peaks around 7-11 PM and then start declining.
- Trips booking picks up at the start of the months, peaks in middle and again start declining in %
- 81.4% trips start and end in same state mean source and destination state are same.
- Maharastra & Karnataka contributes to more than 37% intra-state trip, followed by Tamilnadu, harayana.
- Haryana and Delhi tops the list as destination states for inter-state trips as well contributing more than 40%.
- Haryana & Delhi leads source in the Inter-state trips with more 50% trips starts from there, follow2ed by Maharastra and UttarPradesh.

Hypoethesis testing results

- The features 'Total Trip Time' and 'Start to end scan time' are not statistically different.
- The features 'Total actual Time' and 'total osrm time' are statistically different.
- The features 'Total actual Time' and 'total segment actual time' are not statistically different.
- The features 'Total osrm Time' and 'total segment osrm time' are statistically different.
- The features 'Total osrm distance' and 'Total osrm segment distance' are statistically different.

Recommendations

Based on the insights from the above business insights, here are tailored recommendations to enhance operational efficiency, data quality, and strategic decision-making:

1. **Data Quality & Standardization:**

- **Missing Location IDs**: The absence of 14 unique location IDs (for both source and destination) can affect the accuracy of route planning and analysis. Investigate why these IDs are missing and fill them in from external datasets or review data collection processes.
- **City Name Standardization**: For cities like **Bengaluru** and **Bangalore**, where old and new names are used interchangeably, standardize city names across the dataset. This will improve data consistency and prevent confusion in reporting and analysis.

2. **Operational Optimization:**

- **Intra-state Trips Focus**: With 81.4% of trips starting and ending in the same state, focusing on optimizing intra-state logistics (especially in Maharashtra, Karnataka, Tamil Nadu, and Haryana) can yield significant operational improvements. Enhance last-mile delivery solutions, consolidate distribution hubs, and optimize routes within these states.
- **Peak Trip Hours Management**: Since most trips start after noon and peak between 7-11 PM, ensure you have adequate fleet availability and resources during these high-demand hours. You could also incentivize customers to book during non-peak hours to balance the load.

3. **Strategic Regional Focus:**

- **Top Source & Destination States**: With **Maharashtra**, **Karnataka**, and **Haryana** being key players for both source and destination trips, strengthen the logistics network in these regions. Consider adding more warehouses or regional hubs to improve service times and reduce operational costs.
- **Inter-state Trip Optimization**: For inter-state trips, particularly those starting from **Haryana** and **Delhi**, invest in better route optimization to improve delivery times. For interstate trips heading to **Haryana** and **Delhi**, ensure smooth cross-border logistics and compliance with state-specific regulations.
- **Bengaluru's Strong Demand and Supply**: As **Bengaluru** leads in both origin and destination trips, focus on enhancing operations in this city. Expanding warehousing, optimizing transportation routes, and offering incentives for sellers can further boost efficiency.

4. **Route Optimization:**

- **Carting as the Most Common Route Type**: Since carting accounts for 69.1% of all routes, focus on optimizing carting logistics. This could involve:
  - Implementing better fleet management and scheduling.
  - Developing predictive models to ensure carting routes are as efficient as possible.
- **Full Truck Load (FTL)**: Though FTL accounts for a smaller share, consider improving operations here by reducing deadhead miles and ensuring full utilization of truck capacity.

5. **Customer Behavior Insights:**

- **Mid-Month Order Peaks**: With trips peaking mid-month, consider aligning marketing campaigns, inventory replenishment, and staffing schedules to match customer behavior. You can also incentivize early-month or end-of-month orders to balance the trip load more evenly.

6. **Feature Discrepancies & Hypothesis Testing:**

- **Discrepancies in Travel Time and Distance Metrics**:
  - Since **total actual time** and **OSRM time** as well as **OSRM distance** and **OSRM segment distance** are statistically different, this indicates room for improvement in route prediction models. Investigate why OSRM (Open Source Routing Machine) time and distance estimates are significantly different from actuals and consider recalibrating the routing algorithms for better accuracy.
- **Non-significant Differences in Some Metrics**: The non-significant differences between **total trip time** and **start-to-end scan time** suggest that these features may not provide additional insights for modeling. Focus on more critical features like **actual time** vs **OSRM time** for optimization.

7. **Model Performance & Testing Data:**

- **Testing Data Predominates**: Since your dataset contains more testing than training data, ensure your machine learning models or analysis pipelines are not overfitting to testing data patterns. Rebalance or validate models to ensure robust performance across both testing and training data.

8. **Regional Logistics Strategy:**

- **Haryana and Delhi as Key Inter-state Hubs**: With over 50% of inter-state trips originating from **Haryana** and **Delhi**, consider building more infrastructure in these states to support the demand. Ensure efficient cross-state transportation, especially for long-haul trips.

9. **Trip Time and Route Optimization:**

- **Time-based Optimization**: Given the statistical difference between **OSRM time** and **actual time**, there is potential to optimize delivery time prediction models. This can help improve ETA accuracy, customer satisfaction, and resource allocation. Fine-tuning route algorithms can lead to better time estimates.
- **Distance Optimization**: The significant difference between **OSRM distance** and **segment OSRM distance** suggests that route segment analysis could reveal inefficiencies. Address these gaps to optimize route selection and reduce fuel costs.

Implementing these recommendations will help optimize your logistics network, improve operational efficiency, and align your services more closely with customer behavior.