```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler
```

```
df = pd.read_csv('delhivery_data.csv.txt')
```

```
#Check the dataframe columns
df.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sou
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	INI
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	INI
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	INI
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	INI
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	INI

```
# Data Info
df.info()
```

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

Insights:

is_cutoff is boolean change into categorical trip_creation_time,od_start_time,od_end_time, cutoff_timestamp is in object change to datetime

```
#Removing Unknown Fields
unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']
df = df.drop(columns = unknown_fields)
```

```
#changing to datetime format
cols=['trip_creation_time','od_start_time','od_end_time']
for col in cols:
  df[col]=pd.to_datetime(df[col])
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  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object        
 3   route_type                      144867 non-null  object        
 4   trip_uuid                       144867 non-null  object        
 5   source_center                   144867 non-null  object        
 6   source_name                     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: datetime64[ns](3), float64(8), object(8)
memory usage: 21.0+ MB
```

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

	count	mean	min	2
trip_creation_time	144867	2018-09-22 13:34:23.659819264	2018-09-12 00:00:16.535741	2018-09 03:20:51.775845
od_start_time	144867	2018-09-22 18:02:45.855230720	2018-09-12 00:00:16.535741	2018-09 08:05:40.886155
od_end_time	144867	2018-09-23 10:04:31.395393024	2018-09-12 00:50:10.814399	2018-09 01:48:06.410121
start_scan_to_end_scan	144867.0	961.262986	20.0	16
actual_distance_to_destination	144867.0	234.073372	9.000045	23.355
actual_time	144867.0	416.927527	9.0	5
osrm_time	144867.0	213.868272	6.0	2
osrm_distance	144867.0	284.771297	9.0082	29.91
segment_actual_time	144867.0	36.196111	-244.0	2
segment_osrm_time	144867.0	18.507548	0.0	1
segment_osrm_distance	144867.0	22.82902	0.0	12.07

```
df.describe(include=object).T
```

	count	unique	top	freq
data	144867	2	training	104858
route_schedule_uuid	144867	1504	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	1812
route_type	144867	2	FTL	99660
trip_uuid	144867	14817	trip-153811219535896559	101
source_center	144867	1508	IND000000ACB	23347
source_name	144574	1498	Gurgaon_Bilaspur_HB (Haryana)	23347
destination_center	144867	1481	IND000000ACB	15192
destination_name	144606	1468	Gurgaon_Bilaspur_HB (Haryana)	15192

Key Data Characteristics

Structure

1. Dimensions: 144,867 rows and 24 columns.
2. Data Types:
    - Categorical or textual: object types.
    - Numerical: float64.
    - Boolean: is_cutoff field.

Missing Values

1. Sparsely distributed:
    - source_name: 293 missing values.
    - destination_name: 261 missing values.
    - All other columns complete.

Statistical Overview

Numerical Columns

1. Wide value ranges observed for start_scan_to_end_scan, actual_distance_to_destination, actual_time, and more.
2. Negative values found in segment_actual_time and segment_factor, warranting further exploration.
3. Variations in factor and segment_factor point to diverse efficiency or performance metrics.

Categorical Columns

1. data: 2 unique values (likely training and testing data).
2. route_schedule_uuid: 1504 unique route schedules.
3. route_type: 2 unique values (e.g., 'Carting', 'FTL').
4. trip_uuid: 14817 unique trip identifiers.
5. source_center and destination_center: Over 1500 unique locations each.
6. source_name and destination_name: Nearly 1500 unique names each.
7. is_cutoff: A boolean field indicating cutoff status.

Notable Observations

1. Wide variations in numerical columns suggest diverse trip characteristics and potential outliers.
2. Negative values in certain columns require investigation for data quality or interpretation nuances.
3. Variations in factor and segment_factor columns highlight potential performance differences.
4. Categorical columns provide context for trip types, schedules, and locations.

```
# Checking for missing values
missing_values = df.isnull().sum()
print("==> Missing values : \n", missing_values)
```

```
==> Missing values :
 data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                     293
destination_center                0
destination_name                261
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
dtype: int64
```

```
#imputing null to constant value
constant_inputer=SimpleImputer(strategy='constant',fill_value='city_place_code (state)')
```

Handle missing values in the data.

Treating missing values

```
cols=['destination_name','source_name']
for col in cols:
  df[col]=pd.DataFrame(constant_inputer.fit_transform(pd.DataFrame(df[col])))
```

```
df.isnull().sum()
```

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

We see no null values

Merging of rows

```
# Grouping by segment

# Creating a unique identifier for each segment of a trip
df['segment_key'] = df['trip_uuid'] + '_' + df['source_center'] + '_' + df['destination_center']

# Using cumsum() to merge the rows for specified columns based on segment_key
cumulative_columns = {
    'segment_actual_time': 'segment_actual_time_sum',
    'segment_osrm_distance': 'segment_osrm_distance_sum',
    'segment_osrm_time': 'segment_osrm_time_sum'
}

for original_col, new_col in cumulative_columns.items():
    df[new_col] = df.groupby('segment_key')[original_col].cumsum()

df[['segment_key', 'segment_actual_time', 'segment_actual_time_sum',
      'segment_osrm_distance', 'segment_osrm_distance_sum',
      'segment_osrm_time', 'segment_osrm_time_sum']]
```

	segment_key	segment_actual_time	segment_ac
0	trip-153741093647649320_IND388121AAA_IND388620AAB	14.0	
1	trip-153741093647649320_IND388121AAA_IND388620AAB	10.0	
2	trip-153741093647649320_IND388121AAA_IND388620AAB	16.0	
3	trip-153741093647649320_IND388121AAA_IND388620AAB	21.0	
4	trip-153741093647649320_IND388121AAA_IND388620AAB	6.0	
...	...	...	
144862	trip-153746066843555182_IND131028AAB_IND000000ACB	12.0	
144863	trip-153746066843555182_IND131028AAB_IND000000ACB	26.0	
144864	trip-153746066843555182_IND131028AAB_IND000000ACB	20.0	
144865	trip-153746066843555182_IND131028AAB_IND000000ACB	17.0	
144866	trip-153746066843555182_IND131028AAB_IND000000ACB	268.0	

144867 rows × 7 columns

```
# Aggregating at segment level

# Creating a dictionary for aggregation at segment level
create_segment_dict = {
  'trip_uuid' : 'first',
  'data': 'first',
  'route_type': 'first',
  'trip_creation_time': 'first',
  'source_name': 'first',
  'destination_name': 'last',
  'od_start_time': 'first',
  'od_end_time': 'last',
  'start_scan_to_end_scan': 'first',
  'actual_distance_to_destination': 'last',
  'actual_time': 'last',
  'osrm_time': 'last',
  'osrm_distance': 'last',
  'segment_actual_time' : 'sum',
  'segment_osrm_time' : 'sum',
  'segment_osrm_distance' : 'sum',
  'segment_actual_time_sum': 'last',
  'segment_osrm_time_sum': 'last',
  'segment_osrm_distance_sum': 'last',
}

# Grouping by segment_key and applying the aggregation operations
seg_agg_data = df.groupby('segment_key').agg(create_segment_dict).reset_index()

# Sorting in ascending order
seg_agg_data = seg_agg_data.sort_values(by=['segment_key','od_end_time'])

seg_agg_data
```

	segment_key	trip_uuid	data	rout
0	trip-153671041653548748_IND209304AAA_IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748_IND462022AAA_IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164_IND561203AAB_IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164_IND572101AAA_IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517_IND000000ACB_IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26363	trip-153861115439069069_IND628204AAA_IND627657AAA	trip-153861115439069069	test	
26364	trip-153861115439069069_IND628613AAA_IND627005AAA	trip-153861115439069069	test	
26365	trip-153861115439069069_IND628801AAA_IND628204AAA	trip-153861115439069069	test	
26366	trip-153861118270144424_IND583119AAA_IND583101AAA	trip-153861118270144424	test	
26367	trip-153861118270144424_IND583201AAA_IND583119AAA	trip-153861118270144424	test	

26368 rows × 20 columns

The rows have been merged based on the unique segment_key, which is a combination of trip_uuid, source_center, and destination_center. The aggregated dataset reflects the total values for each segment of the trip. How the aggregation was performed:

Numerical Fields:

- The Feilds **segment_actual_time**, **segment_osrm_time**, **segment_osrm_distance** were summed up.
- This gives a total measure of time and distance for each unique segment.

Categorical/Boolean Fields:

1. For fields like route_type, the first value in each group was kept.
2. The data field, which distinguishes between training and testing data, was also preserved with the first value.

Source and Destination Names:

1. The source_name field retains the first source name for each segment.
2. The destination_name field holds the last destination name for each segment.

Feature Engineering

Calulating time taken between od_start_time and od_end_time

```
# Preparing for trip-level aggregation

# 1. Calculating time difference between od_start_time and od_end_time
seg_agg_data['od_total_time'] = (
    seg_agg_data['od_end_time'] - seg_agg_data['od_start_time']
).dt.total_seconds() / 60

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

df = seg_agg_data.groupby('trip_uuid').agg(create_trip_dict).reset_index()
df
```

	trip_uuid	data	route_type	od_total_time	trip_creation_time	start_
0	trip-153671041653548748	training	FTL	2260.109800	2018-09-12 00:00:16.535741	
1	trip-153671042288605164	training	Carting	181.611874	2018-09-12 00:00:22.886430	
2	trip-153671043369099517	training	FTL	3934.362520	2018-09-12 00:00:33.691250	
3	trip-153671046011330457	training	Carting	100.494935	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	training	FTL	718.349042	2018-09-12 00:02:09.740725	
...	...	...	...	...	...	
14812	trip-153861095625827784	test	Carting	258.028928	2018-10-03 23:55:56.258533	
14813	trip-153861104386292051	test	Carting	60.590521	2018-10-03 23:57:23.863155	
14814	trip-153861106442901555	test	Carting	422.119867	2018-10-03 23:57:44.429324	
14815	trip-153861115439069069	test	Carting	348.512862	2018-10-03 23:59:14.390954	
14816	trip-153861118270144424	test	FTL	354.407571	2018-10-03 23:59:42.701692	

14817 rows × 18 columns

```
# 4. Extracting features like month, year, day, etc. from Trip_creation_time
df['trip_creation_month'] = df['trip_creation_time'].dt.month
df['trip_creation_year'] = df['trip_creation_time'].dt.year
df['trip_creation_day'] = df['trip_creation_time'].dt.day
df['trip_creation_hour'] = df['trip_creation_time'].dt.hour
df['trip_creation_weekday'] = df['trip_creation_time'].dt.weekday
df['trip_creation_week'] = df['trip_creation_time'].dt.isocalendar().week
df
```

	trip_uuid	data	route_type	od_total_time	trip_creation_time	start_
0	trip-153671041653548748	training	FTL	2260.109800	2018-09-12 00:00:16.535741	
1	trip-153671042288605164	training	Carting	181.611874	2018-09-12 00:00:22.886430	
2	trip-153671043369099517	training	FTL	3934.362520	2018-09-12 00:00:33.691250	
3	trip-153671046011330457	training	Carting	100.494935	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	training	FTL	718.349042	2018-09-12 00:02:09.740725	
...	...	...	...	...	...	
14812	trip-153861095625827784	test	Carting	258.028928	2018-10-03 23:55:56.258533	
14813	trip-153861104386292051	test	Carting	60.590521	2018-10-03 23:57:23.863155	
14814	trip-153861106442901555	test	Carting	422.119867	2018-10-03 23:57:44.429324	
14815	trip-153861115439069069	test	Carting	348.512862	2018-10-03 23:59:14.390954	
14816	trip-153861118270144424	test	FTL	354.407571	2018-10-03 23:59:42.701692	

14817 rows × 24 columns

Extract city, state and place for source and destination.

```
df['destination_city']=df['destination_name'].apply(lambda x: x.split("_")[0])
df['destination_state']=df['destination_name'].apply(lambda x: x.split('(')[1][0:len(x.split('(')[1])-1])
df['destination_place']=df['destination_name'].apply(lambda x:x.split("_")[1][0:(x.split("_")[1]).find('(')] if len(x.split("_"))>1 else 'ur
df['destination_code']=df['destination_name'].apply(lambda x:x.split("_")[2][0:(x.split("_")[2]).find('(')] if len(x.split("_"))>2 else 'unk
```

```
df['source_city']=df['source_name'].apply(lambda x: x.split("_")[0])
df['source_state']=df['source_name'].apply(lambda x: x.split('(')[1][0:len(x.split('(')[1])-1])
df['source_place']=df['source_name'].apply(lambda x:x.split("_")[1][0:(x.split("_")[1]).find('(')] if len(x.split("_"))>1 else 'unknown')
df['source_code']=df['source_name'].apply(lambda x:x.split("_")[2][0:(x.split("_")[2]).find('(')] if len(x.split("_"))>2 else 'unknown')
```

```
df['trip_creation_month'] = df['trip_creation_time'].dt.month
df['trip_creation_year'] = df['trip_creation_time'].dt.year
df['trip_creation_day'] = df['trip_creation_time'].dt.day
df['trip_creation_hour'] = df['trip_creation_time'].dt.hour
df['trip_creation_weekday'] = df['trip_creation_time'].dt.weekday
df['trip_creation_week'] = df['trip_creation_time'].dt.isocalendar().week
```

```
# Dropping the original columns
df = df.drop(['source_name', 'destination_name'], axis=1)
df
```

	trip_uuid	data	route_type	od_total_time	trip_creation_time	start_
0	trip-153671041653548748	training	FTL	2260.109800	2018-09-12 00:00:16.535741	
1	trip-153671042288605164	training	Carting	181.611874	2018-09-12 00:00:22.886430	
2	trip-153671043369099517	training	FTL	3934.362520	2018-09-12 00:00:33.691250	
3	trip-153671046011330457	training	Carting	100.494935	2018-09-12 00:01:00.113710	
4	trip-153671052974046625	training	FTL	718.349042	2018-09-12 00:02:09.740725	
...	...	...	...	...	...	
14812	trip-153861095625827784	test	Carting	258.028928	2018-10-03 23:55:56.258533	
14813	trip-153861104386292051	test	Carting	60.590521	2018-10-03 23:57:23.863155	
14814	trip-153861106442901555	test	Carting	422.119867	2018-10-03 23:57:44.429324	
14815	trip-153861115439069069	test	Carting	348.512862	2018-10-03 23:59:14.390954	
14816	trip-153861118270144424	test	FTL	354.407571	2018-10-03 23:59:42.701692	

14817 rows × 30 columns

This trip-level dataset is ready for further analysis, such as examining patterns in trip durations, distances, and understanding the distribution of trips over time and geography.

4. In-Depth Analysis

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

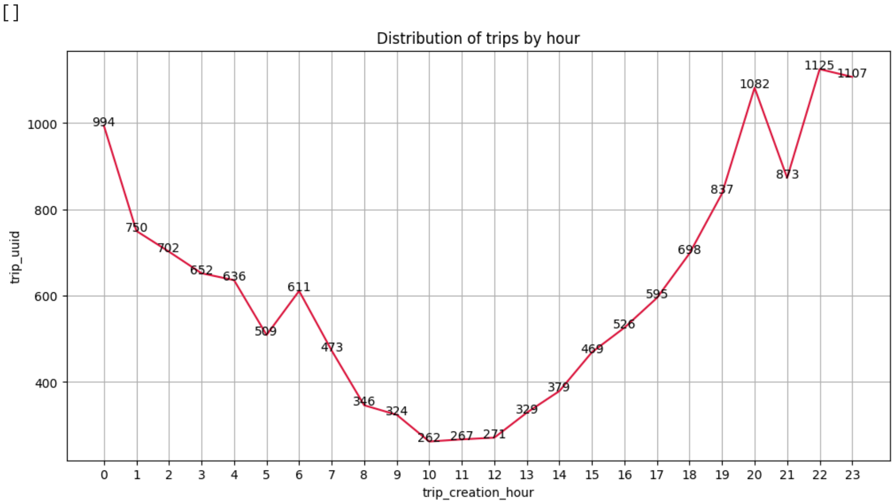
	count	mean	min	25
od_total_time	14817.0	531.795209	23.461468	149.93059
trip_creation_time	14817	2018-09-22 12:44:19.555167744	2018-09-12 00:00:16.535741	2018-09-1 02:51:25.1291258
start_scan_to_end_scan	14817.0	530.810016	23.0	149
actual_distance_to_destination	14817.0	164.477838	9.002461	22.83723
actual_time	14817.0	357.143754	9.0	67
osrm_time	14817.0	161.384018	6.0	29
osrm_distance	14817.0	204.344689	9.0729	30.819
segment_actual_time	14817.0	353.892286	9.0	66
segment_osrm_time	14817.0	180.949787	6.0	31
segment_osrm_distance	14817.0	223.201161	9.0729	32.654
segment_actual_time_sum	14817.0	353.892286	9.0	66
segment_osrm_time_sum	14817.0	180.949787	6.0	31
segment_osrm_distance_sum	14817.0	223.201161	9.0729	32.654
trip_creation_month	14817.0	9.120672	9.0	9
trip_creation_year	14817.0	2018.0	2018.0	2018
trip_creation_day	14817.0	18.37079	1.0	14
trip_creation_hour	14817.0	12.449821	0.0	4
trip_creation_weekday	14817.0	2.919349	0.0	1
trip_creation_week	14817.0	38.295944	37.0	38

```
df.describe(include = object).T
```

	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
data	14817	2	training	10654
route_type	14817	2	Carting	8908
destination_city	14817	856	Bengaluru	1088
destination_state	14817	32	Maharashtra	2561
destination_place	14817	789	Bilaspu	864
destination_code	14817	25	D	2868
source_city	14817	735	Gurgaon	1139
source_state	14817	30	Maharashtra	2714
source_place	14817	707	Bilaspu	1085
source_code	14817	22	HB	3222

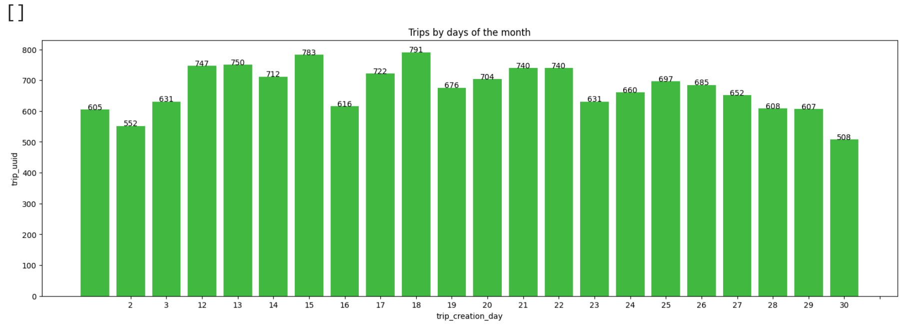
```
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], y[i], ha = 'center')

# Distribution of trips by hour
df_by_hour = df.groupby(by = 'trip_creation_hour')['trip_uuid'].count().to_frame().reset_index()

plt.figure(figsize = (12, 6))
sns.lineplot(data = df,
             x = df_by_hour['trip_creation_hour'],
             y = df_by_hour['trip_uuid'],
             markers = '*', color = 'crimson')
plt.xticks(np.arange(0,24))
plt.grid('both')
addlabels(df_by_hour['trip_creation_hour'],df_by_hour['trip_uuid'])
plt.title('Distribution of trips by hour')
plt.plot()
```

[]



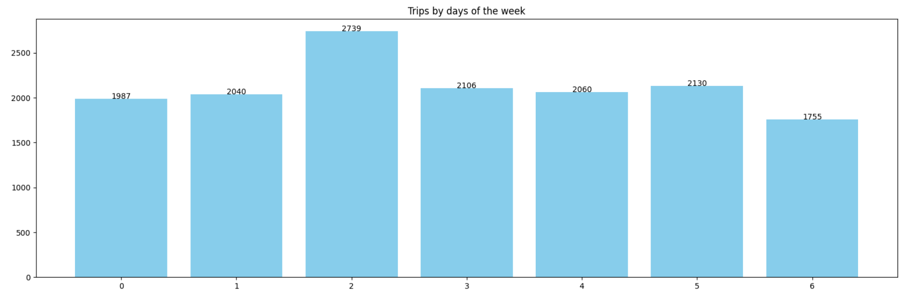

```
# Distribution of trips by days of the month
df_by_day = df.groupby(by = 'trip_creation_day')['trip_uuid'].count().to_frame().reset_index()

plt.figure(figsize = (20, 6))
sns.barplot(data = df,
            x = df_by_day['trip_creation_day'],
            y = df_by_day['trip_uuid'], color = 'limegreen')
plt.xticks(np.arange(1,32))
addlabels(df_by_day['trip_creation_day'],df_by_day['trip_uuid'])
plt.title('Trips by days of the month')
plt.plot()
```

[]



```
# Count of trips by days of the week
df_by_weekday = df.groupby(by = 'trip_creation_weekday')['trip_uuid'].count().to_frame().reset_index()

plt.figure(figsize = (20, 6))
s = plt.bar(df_by_weekday['trip_creation_weekday'], df_by_weekday['trip_uuid'],
            color = 'skyblue')
plt.xticks(np.arange(0,7))
addlabels(df_by_weekday['trip_creation_weekday'],df_by_weekday['trip_uuid'])
plt.title('Trips by days of the week')
plt.plot()
```

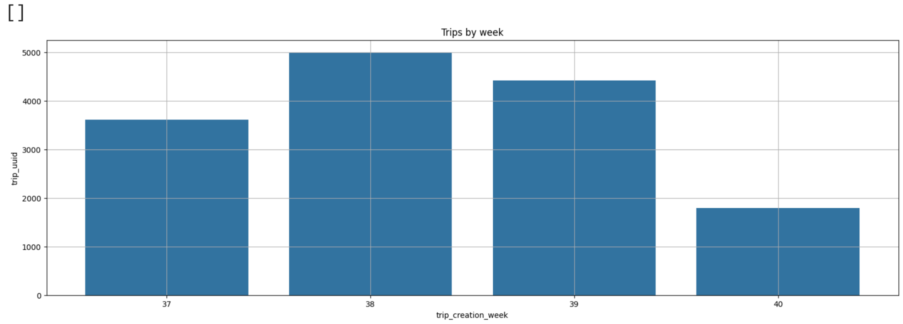
[]



Tuesday seems to be the day with the highest trip creation time. This could be because that there is a day gap given post the weekend when people would make the most purchases so as to consolidate the orders and reduce trips.

```
# Count of trips by week of the year
df_by_week = df.groupby(by = 'trip_creation_week')['trip_uuid'].count().to_frame().reset_index()

plt.figure(figsize = (20, 6))
sns.barplot(data = df,
            x = df_by_week['trip_creation_week'],
            y = df_by_week['trip_uuid'])
plt.grid('both')
plt.title('Trips by week')
plt.plot()
```

[]



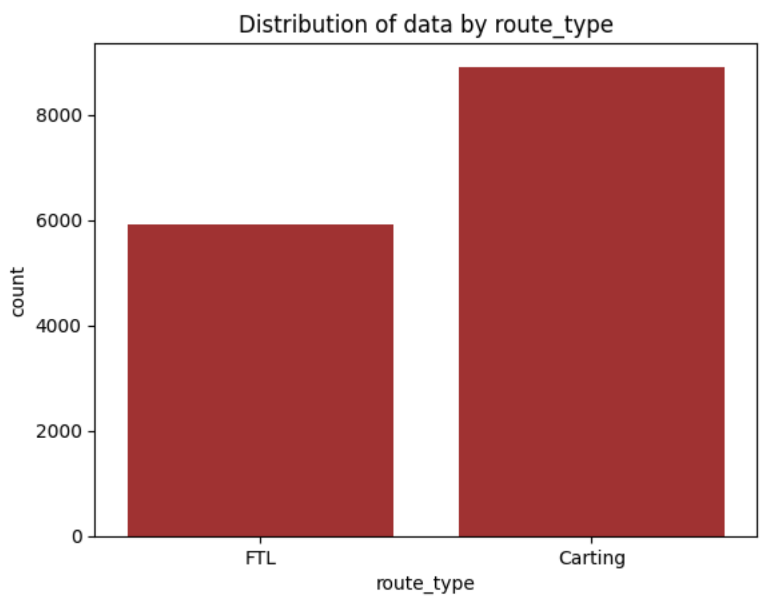
Most trips are created in the 38th week of the year

```
df['trip_creation_month'].value_counts(normalize = True) * 100
```

```
trip_creation_month
9     87.93278
10    12.06722
Name: proportion, dtype: float64
```

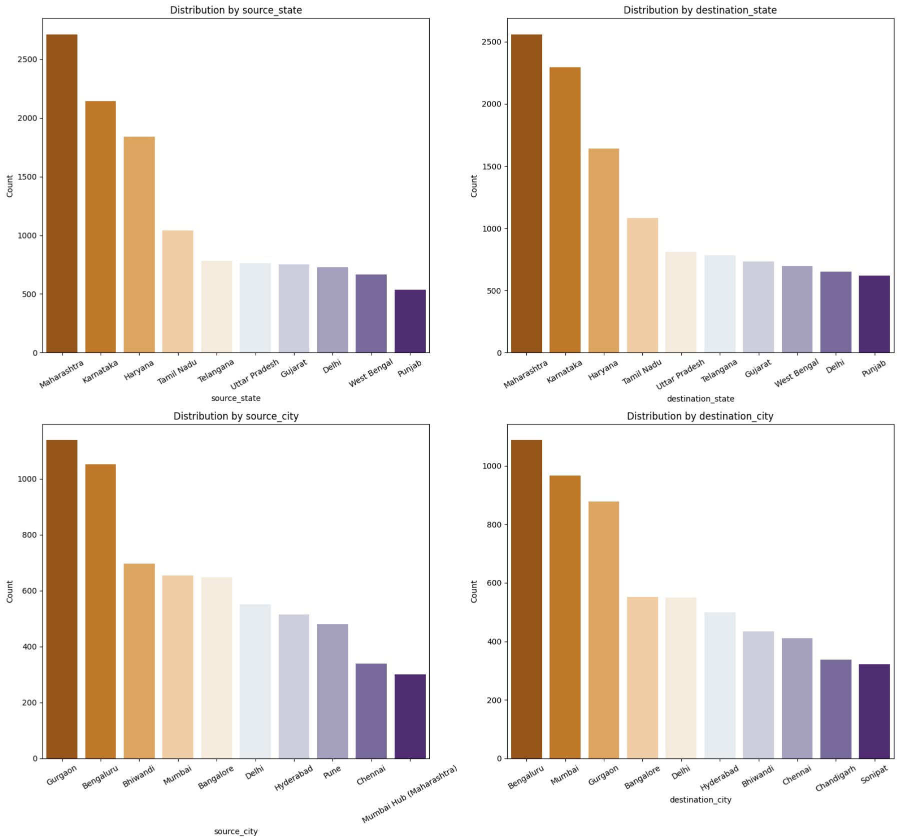
Either September has the most number of trips created, or the data is insufficient to confirm on the monthly trips.

```
# Distribution of data by route_type
sns.countplot(data=df,x='route_type', color= 'firebrick')
plt.title('Distribution of data by route_type')
plt.show()
```



By the given data, more trips are done by the 'Carting' Transportation type.

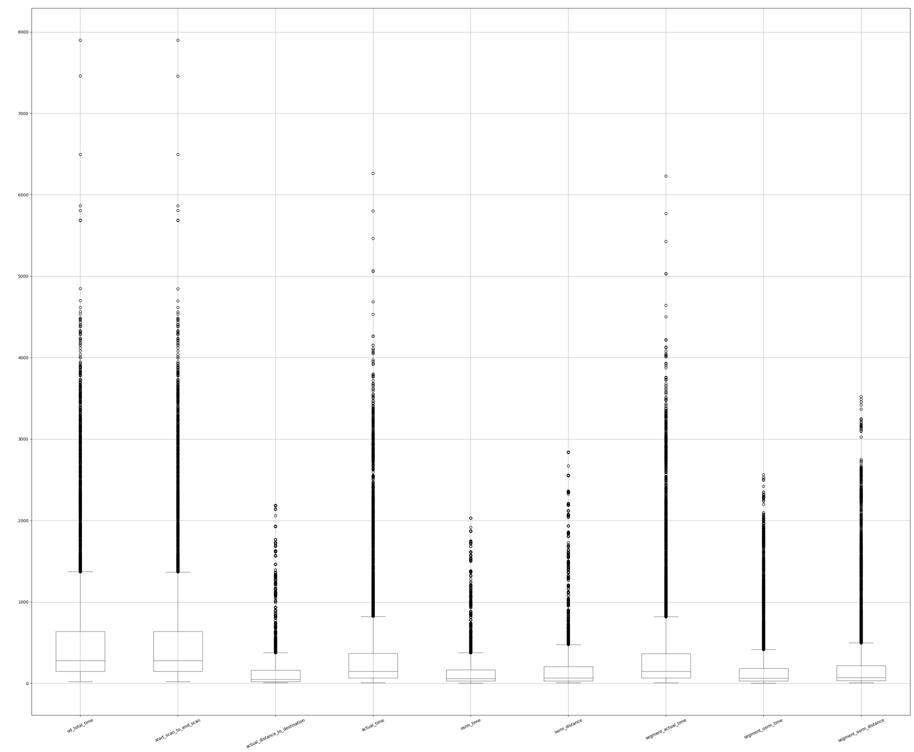
```
# State with most trips

# plt.figure(figsize = (20, 6))
# sns.countplot(data=df, x=df['source_state'])
# plt.xticks(rotation=75)
# plt.tight_layout()
# plt.show()

area_vars= ['source_state', 'destination_state', 'source_city', 'destination_city']
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 15))
for i, var in enumerate(area_vars):
  ax = axes[i // 2, i % 2]
  sns.countplot(data=df, x=var, ax=ax, order=df[var].value_counts(normalize=True).nlargest(10).index, palette='PuOr')
  ax.set_title(f'Distribution by {var}')
  ax.set_ylabel('Count')
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.tight_layout()
plt.show()
```



- These four states – Maharashtra, Karnataka, Haryana, and Tamil Nadu – are major origin and destination points for delivery services.
- These four metropolitan hubs - Mumbai, Gurgaon, Delhi, and Bengaluru – account for a significant share of delivery origin points.
- Packages headed to Mumbai, Bengaluru, Gurgaon, and Delhi make up a significant portion of deliveries nationwide.

Outlier Detection & Treatment

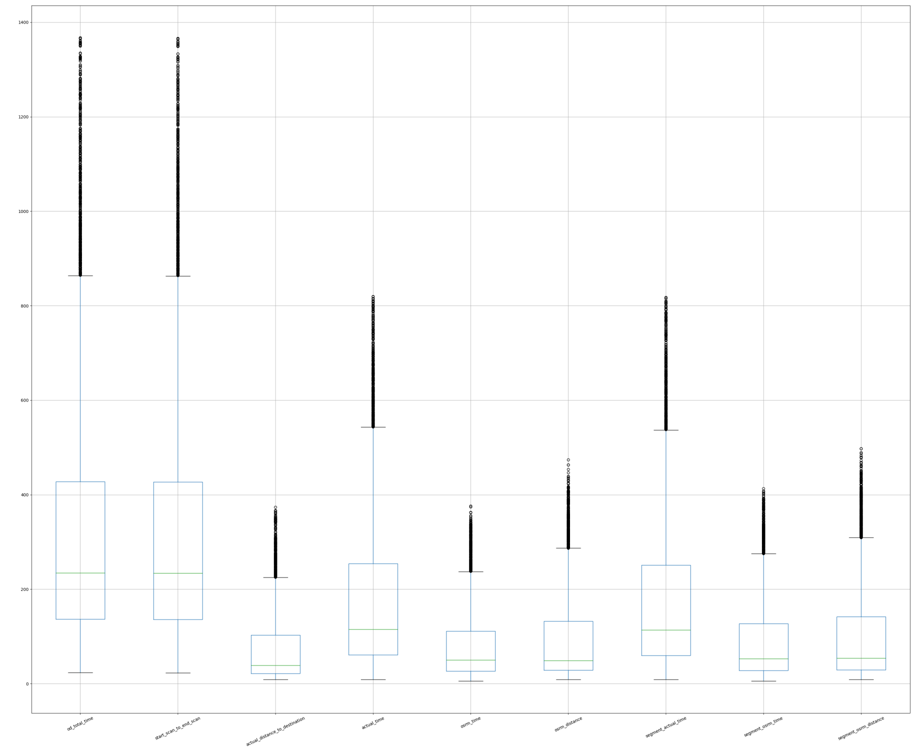
```
# Selecting numerical features for outlier detection
numerical_features = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance_to_destination',
                    'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                    'segment_osrm_time', 'segment_osrm_distance']

# Plotting boxplots for numerical dfatures to visualize outliers
plt.figure(figsize=(30, 25))


df[numerical_features].boxplot(rot=25, figsize=(35,20), color = 'dimgray')
plt.tight_layout()
plt.show()
```



```
# range of outliers are large and also lot in the numerical features.

# Detecting and handling outliers using IQR method
Q1 = df[numerical_features].quantile(0.25)
Q3 = df[numerical_features].quantile(0.75)
IQR = Q3 - Q1

# Filtering out the outliers by keeping only the values that are within 1.5*IQR of Q1 and Q3
df_no_outliers = df[~((df[numerical_features] < (Q1 - 1.5 * IQR)) | (df[numerical_features] > (Q3 + 1.5 * IQR))).any(axis=1)]

# Comparing the shape of the original and outlier-removed dataframes
original_shape = df.shape
outlier_removed_shape = df_no_outliers.shape

original_shape, outlier_removed_shape
```

```
((14817, 30), (12759, 30))
```

```
# After IQR - Plotting boxplots again for numerical features to visualize outliers
plt.figure(figsize=(30, 25))



df_no_outliers[numerical_features].boxplot(rot=25, figsize=(35,20))
plt.tight_layout()
plt.show()
```



After we handled the outliers using IQR method, we can see most outliers are removed as evident in the graph

```
df_corr = df_no_outliers[numerical_features].corr()
df_corr
```

	od_total_time	start_scan_to_end_scan	actual_distance_to_
od_total_time	1.000000	0.999997	
start_scan_to_end_scan	0.999997	1.000000	
actual_distance_to_destination	0.758645	0.758186	
actual_time	0.829296	0.828989	
osrm_time	0.762512	0.761976	
osrm_distance	0.769243	0.768770	
segment_actual_time	0.828951	0.828646	
segment_osrm_time	0.740551	0.740076	
segment_osrm_distance	0.756370	0.755942	

Very high correlation exists between all the columns.

One-hot encoding

```
# Selecting categorical features for one-hot encoding
categorical_features = ['route_type', 'data',
                        'destination_city', 'destination_state',
                        'source_city', 'source_state', 'source_place', 'destination_place']

# Applying one-hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
encoded_categorical = onehot_encoder.fit_transform(df[categorical_features])

# Converting the encoded features back to a dataframe
encoded_categorical_df = pd.DataFrame(encoded_categorical, columns=onehot_encoder.get_feature_names_out(categorical_features))

encoded_categorical_df
```

	route_type_Carting	route_type_FTL	data_test	data_training	destination_city_AN
0	0.0	1.0	0.0	1.0	0
1	1.0	0.0	0.0	1.0	0
2	0.0	1.0	0.0	1.0	0
3	1.0	0.0	0.0	1.0	0
4	0.0	1.0	0.0	1.0	0
...	...	...	...	...	
14812	1.0	0.0	1.0	0.0	0
14813	1.0	0.0	1.0	0.0	0
14814	1.0	0.0	1.0	0.0	0
14815	1.0	0.0	1.0	0.0	0
14816	0.0	1.0	1.0	0.0	0

14817 rows × 3153 columns

Normalize/Standardize the numerical features

```
numerical_features
```

```
['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']
```

```
df.columns
```

```
Index(['trip_uuid', 'data', 'route_type', 'od_total_time',
       'trip_creation_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', 'segment_actual_time_sum',
       'segment_osrm_time_sum', 'segment_osrm_distance_sum',
       'trip_creation_month', 'trip_creation_year', 'trip_creation_day',
       'trip_creation_hour', 'trip_creation_weekday', 'trip_creation_week',
       'destination_city', 'destination_state', 'destination_place',
       'destination_code', 'source_city', 'source_state', 'source_place',
       'source_code'],
      dtype='object')
```

```
# Normalizing/Standardizing the numerical features using MinMaxScaler
min_max_scaler = MinMaxScaler()
min_max_scaled_numerical = min_max_scaler.fit_transform(df[numerical_features])

# Converting the scaled features back to a dataframe
min_max_scaled_numerical_df = pd.DataFrame(min_max_scaled_numerical, columns=numerical_features)

min_max_scaled_numerical_df
```

	od_total_time	start_scan_to_end_scan	actual_distance_to_destination	actual_ti
0	0.284016	0.283937	0.374613	0.2482
1	0.020082	0.019937	0.029476	0.0214
2	0.496617	0.496508	0.880999	0.5335
3	0.009782	0.009778	0.003753	0.0079
4	0.088239	0.088127	0.054395	0.0530
...	...	...	...	
14812	0.029786	0.029714	0.022392	0.0118
14813	0.004715	0.004698	0.002990	0.0019
14814	0.050623	0.050540	0.013631	0.0436
14815	0.041276	0.041143	0.057736	0.0407
14816	0.042024	0.041905	0.026213	0.0425

14817 rows × 9 columns

We can clearly see that the values have been scaled between 0 to 1.

```
# Standardizing the numerical features using StandardScaler
std_scaler = StandardScaler()
std_scaled_numerical = std_scaler.fit_transform(df[numerical_features])

# Converting the scaled features back to a dataframe
std_scaled_numerical_df = pd.DataFrame(std_scaled_numerical, columns=numerical_features)

std_scaled_numerical_df
```

	od_total_time	start_scan_to_end_scan	actual_distance_to_destination	actual_ti
0	2.621986	2.623702	2.162092	2.1462
1	-0.531255	-0.532593	-0.298944	-0.3814
2	5.161957	5.165134	5.772935	5.3259
3	-0.654316	-0.654047	-0.482362	-0.5310
4	0.283017	0.282670	-0.121257	-0.0287
...	...	...	...	
14812	-0.415325	-0.415693	-0.349454	-0.4883
14813	-0.714854	-0.714774	-0.487802	-0.5987
14814	-0.166386	-0.166711	-0.411926	-0.1338
14815	-0.278053	-0.279057	-0.097433	-0.1659
14816	-0.269111	-0.269947	-0.322212	-0.1463

14817 rows × 9 columns

```
# Combining the encoded and scaled features with the rest of the dataset
processed_data = pd.concat([df.drop(categorical_features + numerical_features, axis=1),
                            encoded_categorical_df, min_max_scaled_numerical_df], axis=1)
processed_data
```

	trip_uuid	trip_creation_time	segment_actual_time_sum	segment_osrm_ti
0	trip-153671041653548748	2018-09-12 00:00:16.535741	1548.0	
1	trip-153671042288605164	2018-09-12 00:00:22.886430	141.0	
2	trip-153671043369099517	2018-09-12 00:00:33.691250	3308.0	
3	trip-153671046011330457	2018-09-12 00:01:00.113710	59.0	
4	trip-153671052974046625	2018-09-12 00:02:09.740725	340.0	
...	...	...	...	
14812	trip-153861095625827784	2018-10-03 23:55:56.258533	82.0	
14813	trip-153861104386292051	2018-10-03 23:57:23.863155	21.0	
14814	trip-153861106442901555	2018-10-03 23:57:44.429324	281.0	
14815	trip-153861115439069069	2018-10-03 23:59:14.390954	258.0	
14816	trip-153861118270144424	2018-10-03 23:59:42.701692	274.0	

14817 rows × 3175 columns

Hypothesis Testing

Perform hypothesis testing / visual analysis between

Step 1. State Null Hypothesis

- Null Hypothesis: The actual value and supposed value are same.
- Alternate Hypothesis: The actual value and supposed value are different.

Step 2. Check for basic assumptions

- Distribution check using **QQ Plot**
- Test for Normality using **Shapiro-Wilk** or **Kolmogorov-Smirnov** test
- Homogeneity of Variances using **Levene's test**

Step 3. Define Test statistics; Distribution of T under H0.

- For comparing independent samples: If t-test assumptions hold, we will go for it. Else, we can choose the non-parametric Mann-Whitney U test for its robustness.

Step 4. Compute the p-value and fix value of alpha.

- We will set the alpha value as .05

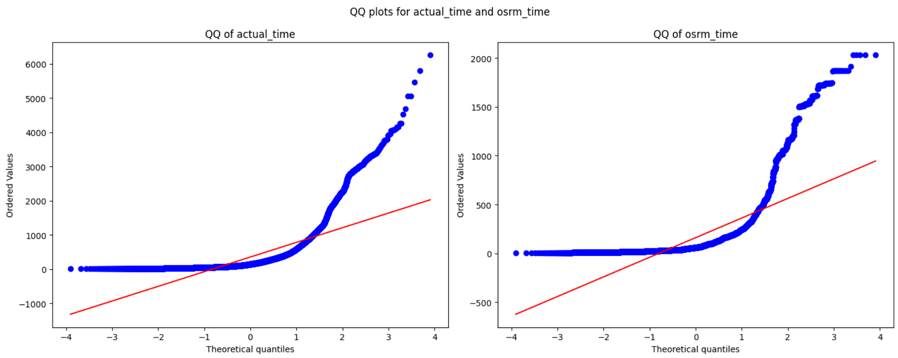
Step 5. Compare p-value and alpha.

- If p_value < alpha : **Reject H0**
- If p_value > alpha : **Accept H0**

```python
alpha = .05

# Creating generic functions to be used

def plot_qq(i,j):
  plt.figure(figsize = (15, 6))
  plt.subplot(1, 2, 1)
  plt.suptitle(f"QQ plots for {i} and {j}")
  stats.probplot(df[i], plot = plt, dist = 'norm')
  plt.title(f"QQ of {i}")
  plt.subplot(1, 2, 2)
  stats.probplot(df[j], plot = plt, dist = 'norm')
  plt.title(f"QQ of {j}")
  plt.tight_layout()
  plt.show()

def test_shapiro_wilk(i):
  # Point to remember: Shapiro-Wilk is basically for smaller sample size.
  # Ho : Sample follows Normal distribution
  # Ha : Sample doesn't follow Normal
  sw_test_stat, p_value = stats.shapiro(df[i].sample(5000))
  print('SW p-value :', p_value)
  if p_value < alpha:
      print(f"Shapiro-Wilk Test: The {i} sample does NOT follow normal distribution")
  else:
      print(f"Shapiro-Wilk Test: The {i} sample follows normal distribution")

def ks_test(i):
  # Point to remember: Kolmogorov-Smirnov test is basically for larger sample size.
  # Ho : Sample follows Normal distribution
  # Ha : Sample doesn't follow Normal distribution

  ks_test_stat, p_value = stats.kstest(df[i], 'norm')
  print('KS p-value :', p_value)
  if p_value < alpha:
      print(f"Kolmogorov-Smirnov Test: The {i} sample does NOT follow normal distribution")
  else:
      print(f"Kolmogorov-Smirnov Test: The {i} sample follows normal distribution")

def levenes_test(i,j):
  # Ho: Variances are equal
  # Ha: Variances are not homogenous

  levene_stat, p_value = stats.levene(df[i], df[j])
  print('Levene p-value :', p_value)
  if p_value < alpha:
      print(f"Levene's Test: The samples {i} and {j} does NOT suggest equal variances")
  else:
      print(f"Levene's Test: The samples {i} and {j} suggest equal variances.")

def mann_whitney_u_test(i,j):
  # Point to note: We are doing with the default alternative which is two-sided

  # Ho: Both the samples have the same median
  # Ha: The samples have different median and thus differs

  m_w_u_stat, p_value = stats.mannwhitneyu(df[i],df[j])
  if p_value < alpha:
      print(f"Mann-Whitney U Test: The samples {i} and {j} are different")
  else:
      print(f"Mann-Whitney U Test: The samples {i} and {j} are similiar.")
```

Actual_time aggregated value and OSRM time aggregated value.

```
# We believe both the columns are same unless proved otherwise by the hypothesis testing

# Tests for normality and homogenity of the samples.

a51 = 'actual_time'
a52 = 'osrm_time'

plot_qq(a51,a52)
test_shapiro_wilk(a51)
test_shapiro_wilk(a52)
ks_test(a51)
ks_test(a52)
levenes_test(a51,a52)
```




```
SW p-value : 0.0
Shapiro-Wilk Test: The actual_time sample does NOT follow normal distribution
SW p-value : 0.0
Shapiro-Wilk Test: The osrm_time sample does NOT follow normal distribution
KS p-value : 0.0
Kolmogorov-Smirnov Test: The actual_time sample does NOT follow normal distribution
KS p-value : 0.0
Kolmogorov-Smirnov Test: The osrm_time sample does NOT follow normal distribution
Levene p-value : 1.871297993683208e-220
Levene's Test: The samples actual_time and osrm_time does NOT suggest equal variances
```

```
# Since the samples do not follow normal distribution and also does not have equal variances, we'll proceed with Mann-Whitney U test.

mann_whitney_u_test(a51,a52)
```

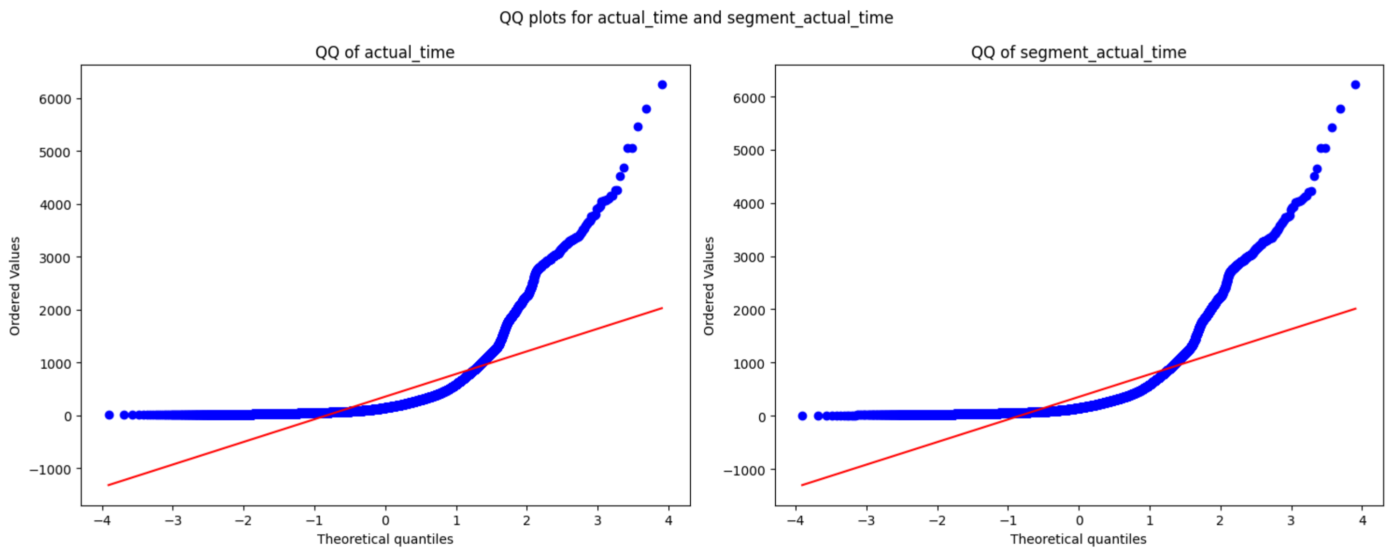
```
Mann-Whitney U Test: The samples actual_time and osrm_time are different
```

Conclusion:

- The columns actual_time and osrm_time do not follow normal distribution and have different variances.
- The columns actual_time and osrm_time are different.

Actual_time aggregated value and segment actual time aggregated value

```
# We believe both the columns are same unless proved otherwise by the hypothesis testing

# Tests for normality and homogenity of the samples.

b51 = 'actual_time'
b52 = 'segment_actual_time'

plot_qq(b51,b52)
test_shapiro_wilk(b51)
test_shapiro_wilk(b52)
ks_test(b51)
ks_test(b52)
levenes_test(b51,b52)
```




```
SW p-value : 0.0
Shapiro-Wilk Test: The actual_time sample does NOT follow normal distribution
SW p-value : 0.0
Shapiro-Wilk Test: The segment_actual_time sample does NOT follow normal distribution
KS p-value : 0.0
Kolmogorov-Smirnov Test: The actual_time sample does NOT follow normal distribution
KS p-value : 0.0
Kolmogorov-Smirnov Test: The segment_actual_time sample does NOT follow normal distribution
Levene p-value : 0.6955022668700895
Levene's Test: The samples actual_time and segment_actual_time suggest equal variances.
```

+ Code + Text

```
# Although they have equal variances, they do not follow normal distribution and T-test cannot be applied. We'll proceed with Mann-Whitney U t

mann_whitney_u_test(b51,b52)
```

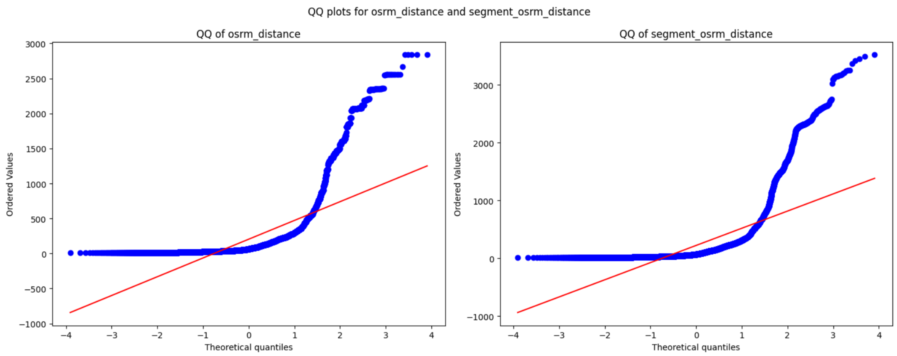
```
Mann-Whitney U Test: The samples actual_time and segment_actual_time are similiar.
```

Conclusion:

- The columns actual_time and segment_actual_time do not follow normal distribution and have equal variances.
- The columns actual_time and segment_actual_time are similiar.

OSRM distance aggregated value and segment OSRM distance aggregated value.

```
# We believe both the columns are same unless proved otherwise by the hypothesis testing

# Tests for normality and homogenity of the samples.

c51 = 'osrm_distance'
c52 = 'segment_osrm_distance'

plot_qq(c51,c52)
test_shapiro_wilk(c51)
test_shapiro_wilk(c52)
ks_test(c51)
ks_test(c52)
levenes_test(c51,c52)
```




```
SW p-value : 0.0
```

```
# Proceeding with Mann-Whitney U test

mann_whitney_u_test(c51,c52)
```

```
Mann-Whitney U Test: The samples osrm_distance and segment_osrm_distance are different
```

Conclusion:

- The columns osrm_time and segment_osrm_time does not follow normal distribution and does not have equal variances.
- The columns osrm_time and segment_osrm_time are different.

OSRM time aggregated value and segment OSRM time aggregated value.

```
# We believe both the columns are same unless proved otherwise by the hypothesis testing

# Tests for normality and homogenity of the samples.

d51 = 'osrm_time'
d52 = 'segment_osrm_time'

plot_qq(d51,d52)
test_shapiro_wilk(d51)
test_shapiro_wilk(d52)
ks_test(d51)
ks_test(d52)
levenes_test(d51,d52)
```