Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as spy
from scipy.stats import norm,t,binom,expon,chi2,chisquare,chi2_contingency,ttest_1samp,ttest
from scipy.stats import kruskal,levene,shapiro,kstest, probplot
```

Loading Dataset

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhiv
```

```
--2024-01-12 12:32:57--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.172.139.94
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.172.139.9
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: ‘delhivery_data’

delhivery_data      100%[===================>]  53.04M   202MB/s    in 0.3s

2024-01-12 12:32:57 (202 MB/s) - ‘delhivery_data’ saved [55617130/55617130]
```

Analysis of Dataset

```
df = pd.read_csv("delhivery_data")
```

```
df.head()
```

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

thanos::sroute:eb7bfc78-

Basic data cleaning and exploration:

Basic Information

```
df.shape
```

```
(144867, 24)
```

5 rows × 24 columns

```
df.columns
```

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

```
df.info()
```

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

```
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

Removing Unknown Fields

```
unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_facto
df = df.drop(columns = unknown_fields)
```

Finding the Categorical Attributes

```
#find out unique entries
for i in df.columns:
    print(f"Unique entries for column {i:35} = {df[i].nunique()}")
```

```
Unique entries for column data                                = 2
Unique entries for column trip_creation_time                  = 14817
Unique entries for column route_schedule_uuid                 = 1504
Unique entries for column route_type                          = 2
Unique entries for column trip_uuid                           = 14817
Unique entries for column source_center                       = 1508
Unique entries for column source_name                         = 1498
Unique entries for column destination_center                  = 1481
Unique entries for column destination_name                    = 1468
Unique entries for column od_start_time                       = 26369
Unique entries for column od_end_time                         = 26369
Unique entries for column start_scan_to_end_scan              = 1915
Unique entries for column actual_distance_to_destination      = 144515
Unique entries for column actual_time                         = 3182
Unique entries for column osrm_time                           = 1531
Unique entries for column osrm_distance                       = 138046
Unique entries for column segment_actual_time                 = 747
Unique entries for column segment_osrm_time                   = 214
Unique entries for column segment_osrm_distance               = 113799
```

Conversion of Categorical Attributes to Category

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

Updating datatype of Date-time columns

```
datetime_columns= ['trip_creation_time', 'od_start_time', 'od_end_time']
for i in datetime_columns:
  df[i] = pd.to_datetime(df[i])
```

Updating float64 to float32 columns

```
floating_columns = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_dist
                    'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance']
for i in floating_columns:
    df[i] = df[i].astype('float32')
```

```
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  float32
 13  actual_time                     144867 non-null  float32
 14  osrm_time                       144867 non-null  float32
 15  osrm_distance                   144867 non-null  float32
 16  segment_actual_time             144867 non-null  float32
 17  segment_osrm_time               144867 non-null  float32
 18  segment_osrm_distance           144867 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(7), float64(1), object(6)
memory usage: 15.2+ MB
```

Missing values in dataset

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

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

```
actual_time                  0
osrm_time                    0
osrm_distance                0
segment_actual_time          0
segment_osrm_time            0
segment_osrm_distance        0
dtype: int64
```

```
#missing source name
m_source_name = df.loc[df.source_name.isnull(), 'source_center'].unique()
m_source_name
```

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

```
#checking the missing source name in destination name
for i in m_source_name:
  if np.all(df['destination_center'] == i):
    print(f"{i} is {df['destination_name']}")
  else:
    print(f"{i} is not found")
```

```
IND342902A1B is not found
IND577116AAA is not found
IND282002AAD is not found
IND465333A1B is not found
IND841301AAC is not found
IND509103AAC is not found
IND126116AAA is not found
IND331022A1B is not found
IND505326AAB is not found
IND852118A1B is not found
```

```
#missing destination name
m_destination_name = df.loc[df.destination_name.isnull(), 'destination_center'].unique()
m_destination_name
```

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

```
#checking the missing destination name in source name
for i in m_destination_name:
  if np.all(df['source_center'] == i):
    print(f"{i} is {df['source_name']}")
  else:
    print(f"{i} is not found")
```

```
IND342902A1B is not found
IND577116AAA is not found
IND282002AAD is not found
IND465333A1B is not found
IND841301AAC is not found
IND505326AAB is not found
IND852118A1B is not found
IND126116AAA is not found
IND509103AAC is not found
IND221005A1A is not found
IND250002AAC is not found
IND331001A1C is not found
IND122015AAC is not found
```

```
#replace the missing source and destination name by their source centre and destination cent
for i in m_source_name:
  df.loc[df['source_center'] == i, 'source_name'] = df.loc[df['source_center'] == i, 'source

for j in m_destination_name:
  df.loc[df['destination_center'] == j, 'destination_name'] = df.loc[df['destination_center'
```

```
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
```

```
df.duplicated().sum()
```

```
0
```

Merging of rows and aggregation of fields

```
df_n = df.groupby(by = ['trip_uuid','source_center', 'destination_center'], as_index = False
```

```
df_n = df_n.sort_values(['trip_uuid', 'od_start_time'])
df_nw = df_n.groupby(by = 'trip_uuid', as_index = False).agg({'source_center' : 'first', 'de
                                                              'data' : 'first', 'route_typ
                                                              'trip_creation_time' : 'firs
                                                              'destination_name' : 'last',
                                                              'od_end_time' : 'last', 'sta
                                                              'actual_distance_to_destinat
                                                              'osrm_time' : 'sum', 'osrm_c
                                                              'segment_actual_time' : 'sum
                                                              'segment_osrm_distance' : 's
```

```
df_nw.head()
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creat
0	trip-153671041653548748	IND462022AAA	IND000000ACB	training	FTL	2 00:00:
1	trip-153671042288605164	IND572101AAA	IND562101AAA	training	Carting	2 00:00:
2	trip-153671043369099517	IND562132AAA	IND160002AAC	training	FTL	2 00:00:
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2 00:01:
4	trip-153671052974046625	IND583101AAA	IND583101AAA	training	FTL	2 00:02:

Build some features to prepare the data for actual analysis

```
def extract_state(x):
  l = x.split('(')
  if len(l) == 1:
    return 'Unknown'
```

```
  else:
    return l[1].replace(')', "")


def extract_city(x):
  if (x in m_source_name) or (x in m_destination_name):
    return 'Unknown'
  else:
    return x.split()[0].split('_')[0]


def extract_place(x):
    if (x in m_source_name) or (x in m_destination_name):
        return 'Unknown'
    elif len(x.split()[0].split('_')) == 1:
        return 'Unknown_Place'
    else:
        return x.split()[0].split('_')[1]
```

Split and extract features from Source name. City-place-code (State)

```
#extract State
df_nw['source_state'] = df_nw['source_name'].apply(extract_state)

#extract City
df_nw['source_city'] = df_nw['source_name'].apply(extract_city)

#extract Place
df_nw['source_place'] = df_nw['source_name'].apply(extract_place)

print("Top 5 source states are: ", df_nw['source_state'].value_counts()[:5])
print("Top 5 source cities are: ", df_nw['source_city'].value_counts()[:5])
print("Top 5 source places are: ", df_nw['source_place'].value_counts()[:5])
```

```
Top 5 source states are:  Maharashtra    2682
Karnataka      2229
Haryana        1684
Tamil Nadu     1085
Delhi           793
Name: source_state, dtype: int64
Top 5 source cities are:  Gurgaon      1024
Bengaluru    1015
Mumbai        893
Bhiwandi      811
Bangalore     755
Name: source_city, dtype: int64
Top 5 source places are:  Central          1039
Bilaspur          970
Mankoli           811
Nelmngla          732
```

```
Unknown_Place     642
Name: source_place, dtype: int64
```

Split and extract features from Destination name. City-place-code (State)

```
#extract State
df_nw['destination_state'] = df_nw['destination_name'].apply(extract_state)

#extract City
df_nw['destination_city'] = df_nw['destination_name'].apply(extract_city)

#extract Place
df_nw['destination_place'] = df_nw['destination_name'].apply(extract_place)

print("Top 5 destination states are: ", df_nw['destination_state'].value_counts()[:5])
print("Top 5 destination cities are: ", df_nw['destination_city'].value_counts()[:5])
print("Top 5 destination places are: ", df_nw['destination_place'].value_counts()[:5])
```

```
Top 5 destination states are:  Maharashtra    2591
Karnataka      2275
Haryana        1667
Tamil Nadu     1072
Telangana       838
Name: destination_state, dtype: int64
Top 5 destination cities are:  Mumbai       1127
Bengaluru    1056
Gurgaon       869
Bangalore     646
Hyderabad     630
Name: destination_city, dtype: int64
Top 5 destination places are:  Central          921
Bilaspur         856
Unknown_Place    725
Nelmngla         628
Mankoli          604
Name: destination_place, dtype: int64
```

Trip_creation_time: Extract features like month, year and day

```
#extract date
df_nw['trip_creation_date'] = df_nw['trip_creation_time'].dt.date
df_nw['trip_creation_date'] = pd.to_datetime(df_nw['trip_creation_date']

#extract year
df_nw['trip_creation_year'] = df_nw['trip_creation_time'].dt.year

#extract month
df_nw['trip_creation_month'] = df_nw['trip_creation_time'].dt.month

#extract day
```

```
df_nw['trip_creation_day'] = df_nw['trip_creation_time'].dt.day

#extract hour
```

Time taken between od_start_time and od_end_time

```
df_nw['od_total_time'] = df_nw['od_end_time'] - df_nw['od_start_time']
df_nw.drop(columns = ['od_end_time', 'od_start_time'], inplace = True)
df_nw['od_total_time'] = df_nw['od_total_time'].apply(lambda x : round(x.total_seconds() / 6
df_nw['od_total_time'].head()
```

```
0    2260.11
1     181.61
2    3934.36
3     100.49
4     718.35
Name: od_total_time, dtype: float64
```

After Cleaning, Merging and Features extraction, the analysis of the structure of new data

```
df_nw.head(3)
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creat
0	trip-153671041653548748	IND462022AAA	IND000000ACB	training	FTL	2 00:00:
1	trip-153671042288605164	IND572101AAA	IND562101AAA	training	Carting	2 00:00:
2	trip-153671043369099517	IND562132AAA	IND160002AAC	training	FTL	2 00:00:

3 rows × 28 columns

```
df_nw.shape
```

```
(14817, 28)
```

```
df_nw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 28 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                     14817 non-null  object
 7   destination_name                14817 non-null  object
 8   start_scan_to_end_scan          14817 non-null  float64
 9   actual_distance_to_destination  14817 non-null  float32
 10  actual_time                     14817 non-null  float32
 11  osrm_time                       14817 non-null  float32
 12  osrm_distance                   14817 non-null  float32
 13  segment_actual_time             14817 non-null  float32
 14  segment_osrm_time               14817 non-null  float32
 15  segment_osrm_distance           14817 non-null  float32
 16  source_state                    14817 non-null  object
 17  source_city                     14817 non-null  object
 18  source_place                    14817 non-null  object
 19  destination_state               14817 non-null  object
 20  destination_city                14817 non-null  object
 21  destination_place               14817 non-null  object
 22  trip_creation_date              14817 non-null  datetime64[ns]
 23  trip_creation_year              14817 non-null  int64
 24  trip_creation_month             14817 non-null  int64
 25  trip_creation_day               14817 non-null  int64
 26  trip_creation_hour              14817 non-null  int64
 27  od_total_time                   14817 non-null  float64
dtypes: category(2), datetime64[ns](2), float32(7), float64(2), int64(4), object(11)
memory usage: 2.6+ MB
```

Statistical Summary

```
df_nw.describe().T
```

	count	mean	std	min	25%
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000
actual_distance_to_destination	14817.0	164.477829	305.388153	9.002461	22.837238
actual_time	14817.0	357.143768	561.396118	9.000000	67.000000
osrm_time	14817.0	161.384018	271.360992	6.000000	29.000000
osrm_distance	14817.0	204.344711	370.395569	9.072900	30.819201
segment_actual_time	14817.0	353.892273	556.247925	9.000000	66.000000
segment_osrm_time	14817.0	180.949783	314.542053	6.000000	31.000000
segment_osrm_distance	14817.0	223.201157	416.628387	9.072900	32.654499
trip_creation_year	14817.0	2018.000000	0.000000	2018.000000	2018.000000
trip_creation_month	14817.0	9.120672	0.325757	9.000000	9.000000
trip_creation_day	14817.0	18.370790	7.893275	1.000000	14.000000
trip_creation_hour	14817.0	12.449821	7.986553	0.000000	4.000000
od_total_time	14817.0	547.462995	668.655943	23.460000	151.160000

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

	count	unique	top	freq
trip_uuid	14817	14817	trip-153671041653548748	1
source_center	14817	868	IND000000ACB	948
destination_center	14817	956	IND000000ACB	813
source_name	14817	868	Gurgaon_Bilaspur_HB (Haryana)	948
destination_name	14817	956	Gurgaon_Bilaspur_HB (Haryana)	813
source_state	14817	30	Maharashtra	2682
source_city	14817	665	Gurgaon	1024
source_place	14817	639	Central	1039
destination_state	14817	33	Maharashtra	2591
destination_city	14817	759	Mumbai	1127
destination_place	14817	721	Central	921

```
df_nw.describe(include = 'category')
```

	data	route_type
count	14817	14817
unique	2	2
top	training	Carting
freq	10654	8908

```
# date time
from_d = df_nw.trip_creation_date.min().date()
to_d = df_nw.trip_creation_date.max().date()
delta = to_d - from_d
print("The data is given from date:",from_d, "to date:", to_d,"and" )
print("Total", delta.days, "days data is given in the dataset.")
```

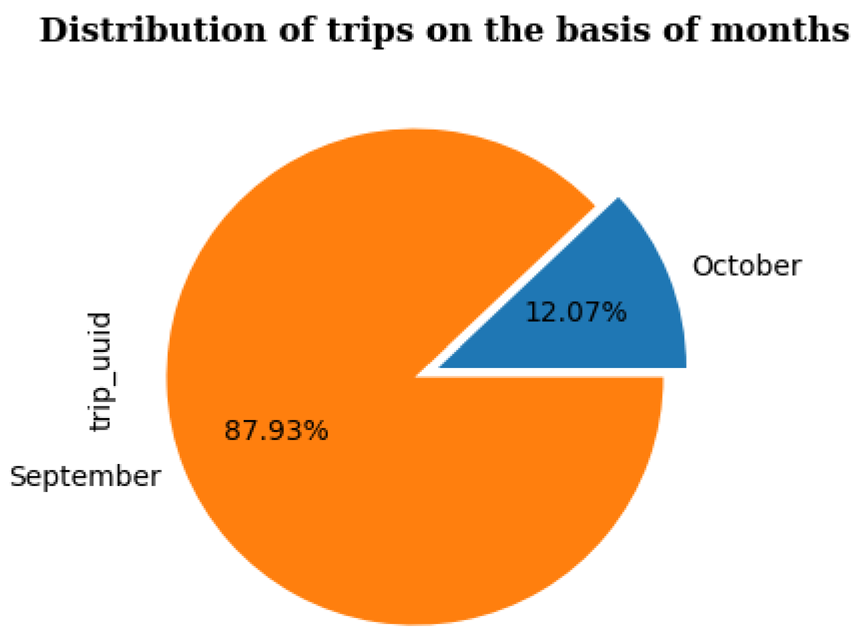
```
The data is given from date: 2018-09-12 to date: 2018-10-03 and
Total 21 days data is given in the dataset.
```

Distribution of trips on the basis of month

```
#trips monthly basis
df_nw.trip_creation_date.dt.month_name().value_counts()
```

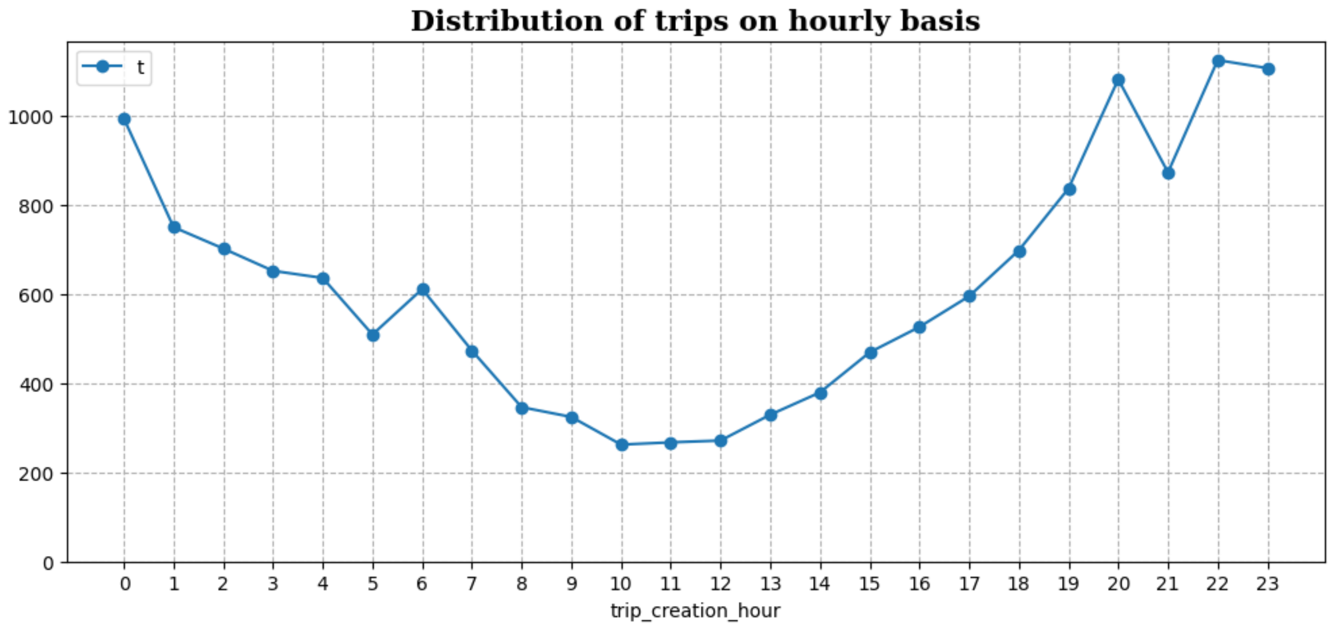
```
September    13029
October       1788
Name: trip_creation_date, dtype: int64
```

```
#trips created in monthly basis
plt.figure(figsize=(4,6))
df_nw.groupby(by = df_nw.trip_creation_date.dt.month_name())['trip_uuid'].count().plot(kind=

plt.title(" Distribution of trips on the basis of months", font='serif', size=12, weight='bo
plt.show()
```



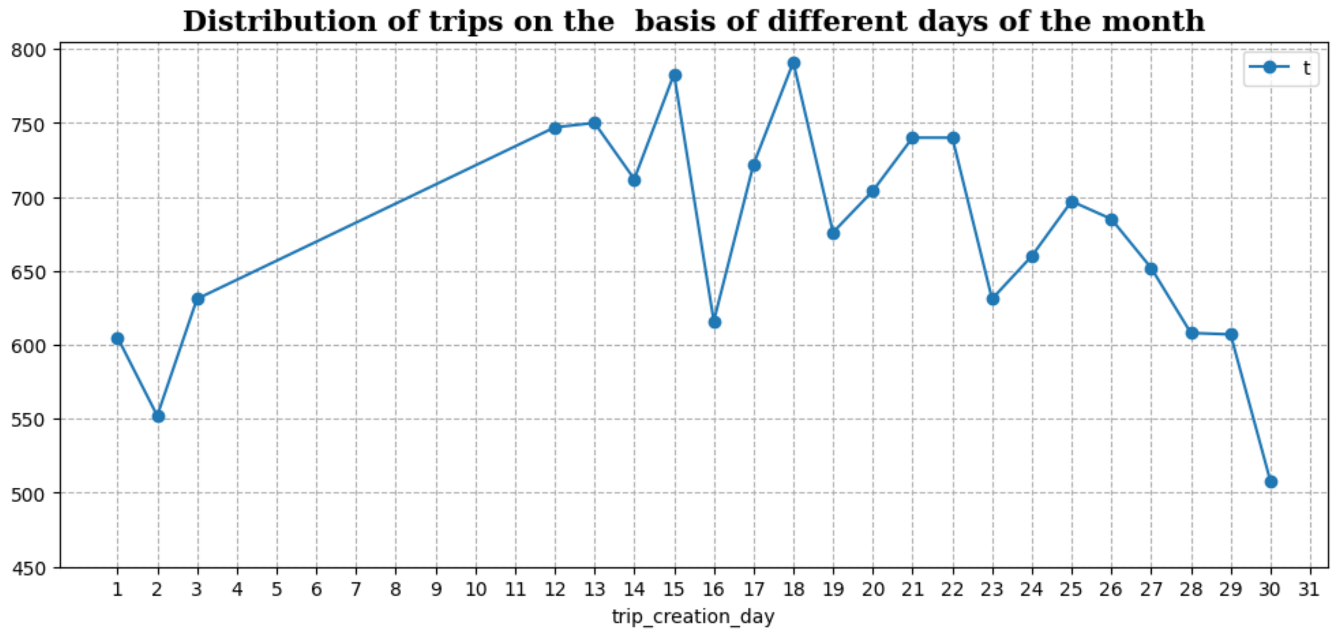
Distribution of trips on the basis of hour

```
# distribution of trips creation on an hourly basis
plt.figure(figsize = (12, 5))
plt.title("Distribution of trips on hourly basis", font='serif',size=15, weight='bold')
df_nw.groupby(by = df_nw['trip_creation_hour'])['trip_uuid'].count().plot(kind = 'line', mar
plt.ylim(0,)
plt.xticks(np.arange(0, 24))
plt.legend('trips')
plt.grid(axis = 'both', linestyle = '--')
plt.show()
```



Distribution of trips on the basis of different days of month

```
# distribution of trips creation on days basis
plt.figure(figsize = (12, 5))
plt.title("Distribution of trips on the  basis of different days of the month", font='serif'
df_nw.groupby(by = df_nw['trip_creation_day'])['trip_uuid'].count().plot(kind = 'line', mark
plt.ylim(450,)
plt.xticks(np.arange(1, 32))
plt.legend('trips')
plt.grid(axis = 'both', linestyle = '--')
plt.show()
```



Categorical Variables:

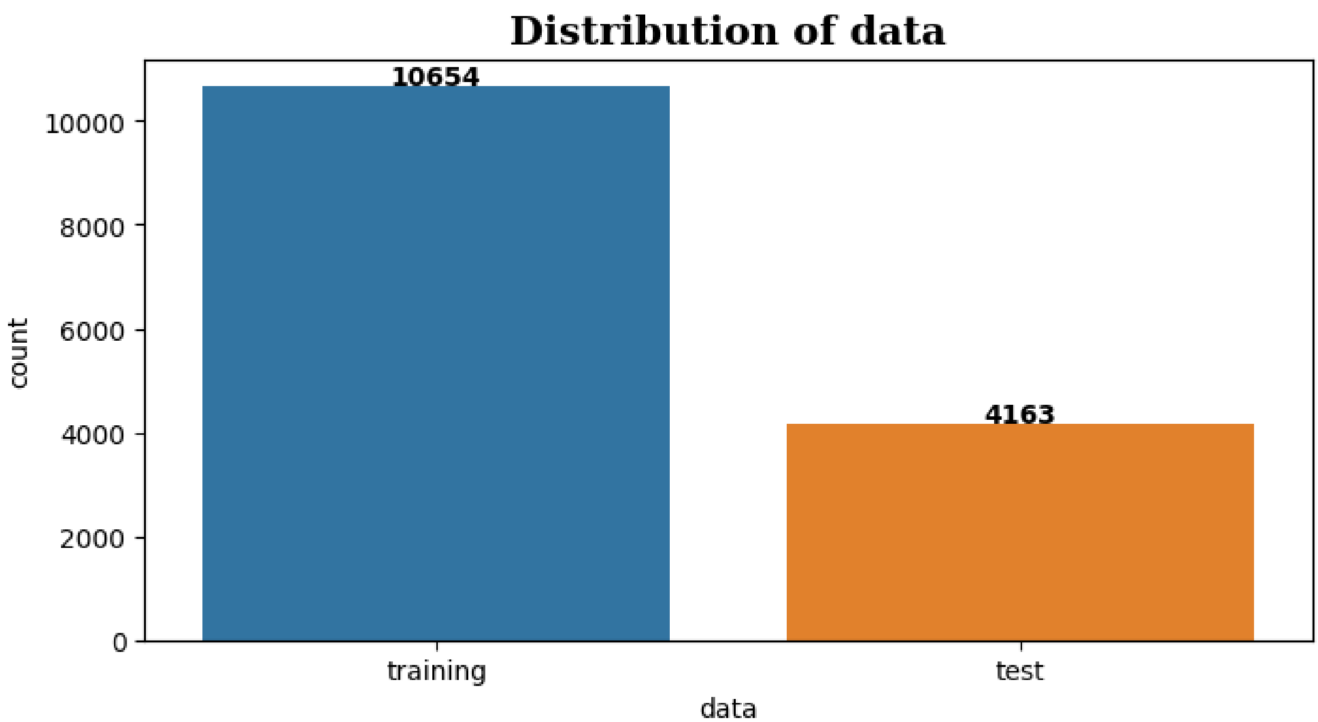
Distribution of data columns for the trip orders

```
data = df_nw.groupby(by = df_nw['data'])['trip_uuid'].count()
```

```
data
```

```
data
test         4163
training    10654
Name: trip_uuid, dtype: int64
```

```
fig = plt.figure(figsize=(8,4))

ht= df_nw.data.value_counts(ascending=False)

sns.countplot(data=df_nw, x='data', order= ht.index)
plt.title('Distribution of data', {'font':'serif', 'size':15,'weight':'bold'})


for i in range(len(ht.index)):
  plt.text(i,ht[i]+20, ht[i], ha='center', va='baseline',fontsize=10, weight= 'bold')

plt.show()
```



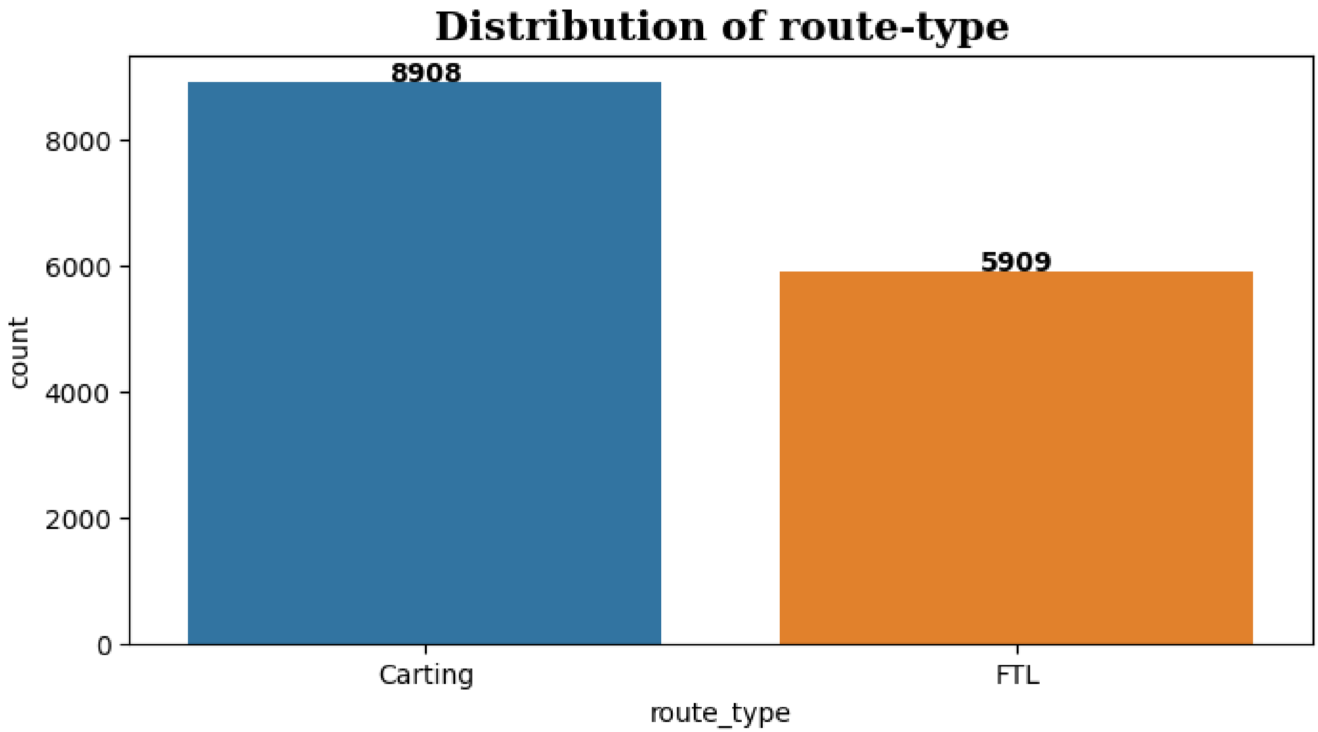
Distribution of route-type

```
route_type = df_nw.groupby(df_nw.route_type)['trip_uuid'].count()
```

```
route_type
```

```
route_type
Carting    8908
FTL        5909
Name: trip_uuid, dtype: int64
```

```
fig = plt.figure(figsize=(8,4))

rt= df_nw.route_type.value_counts(ascending=False)

sns.countplot(data=df_nw, x='route_type', order= rt.index)
plt.title('Distribution of route-type', {'font':'serif', 'size':15,'weight':'bold'})


for i in range(len(rt.index)):
  plt.text(i,rt[i]+22, rt[i], ha='center', va= 'baseline', fontsize=10, weight= 'bold')

plt.show()
```



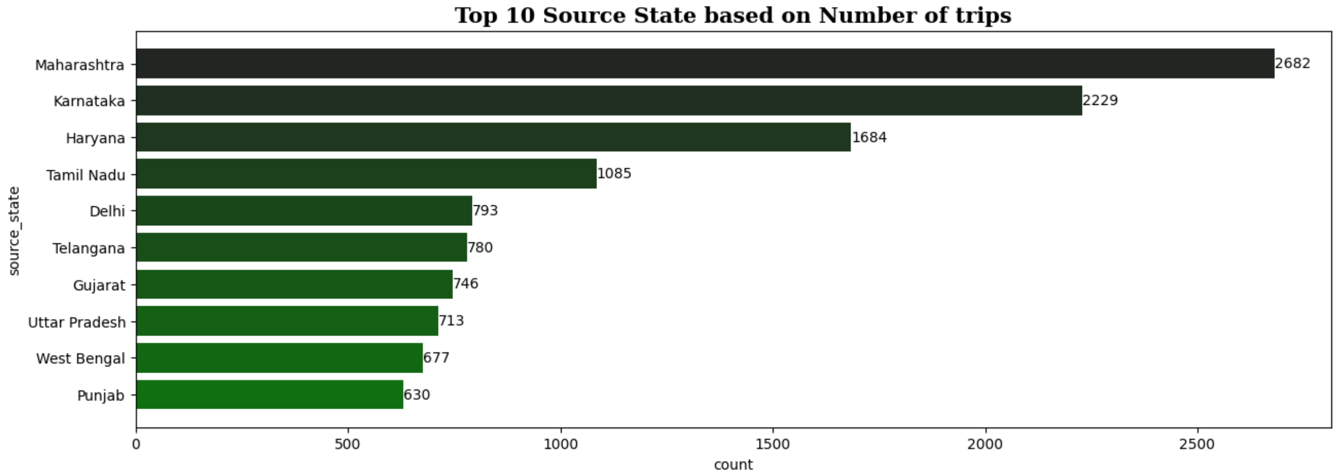
Distribution of trips based on Source-state

```
#Top 10 source-state
T10_source_state = df_nw.groupby(df_nw.source_state)['trip_uuid'].count().sort_values(ascend
T10_source_state[:10]
```

```
source_state
Maharashtra      2682
Karnataka        2229
Haryana          1684
Tamil Nadu       1085
Delhi             793
Telangana         780
Gujarat           746
Uttar Pradesh     713
West Bengal       677
Punjab            630
Name: trip_uuid, dtype: int64
```

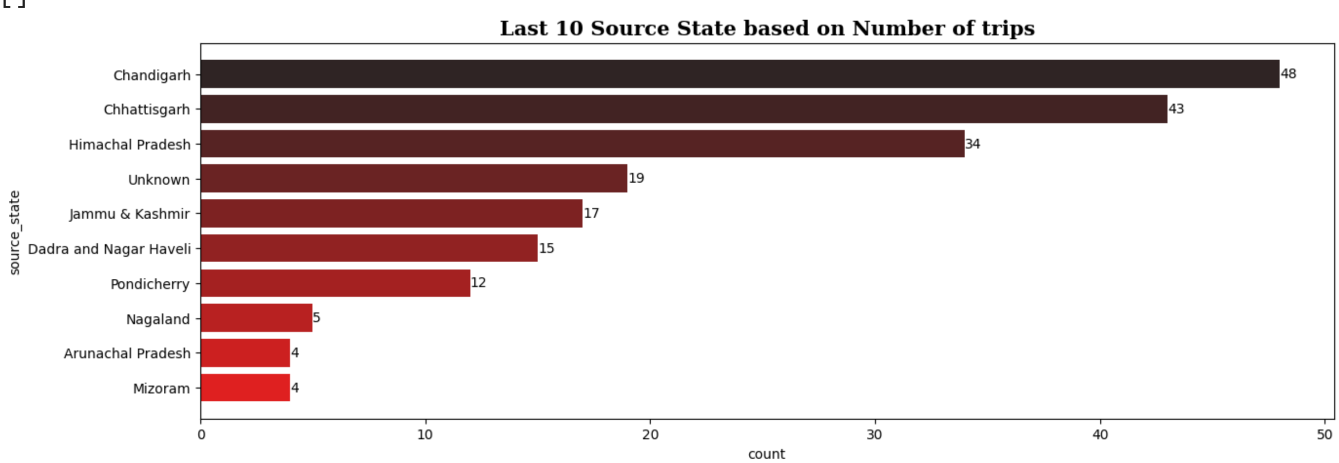
```
plt.figure(figsize = (15, 5))

s = sns.countplot(data = df_nw, order = df_nw['source_state'].value_counts().index[:10], y =

plt.title("Top 10 Source State based on Number of trips", {'font':'serif', 'weight': 'bold',

plt.bar_label(container=s.containers[0])

plt.plot()
```

[]




```
plt.figure(figsize = (15, 5))

t = sns.countplot(data = df_nw, order = df_nw['source_state'].value_counts().index[-10:], y

plt.title("Last 10 Source State based on Number of trips", {'font':'serif', 'weight': 'bold'

plt.bar_label(container=t.containers[0])

plt.plot()
```

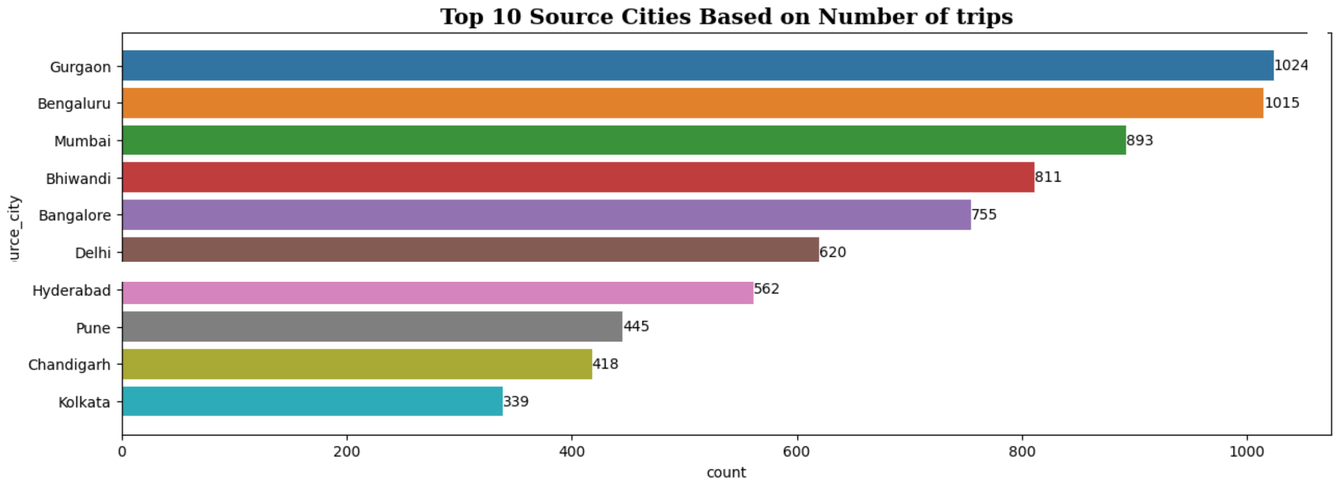
[]



Distribution of trips based on Source-Cities

```
plt.figure(figsize = (15, 5))

s = sns.countplot(data = df_nw, order = df_nw['source_city'].value_counts().index[:10], y

plt.title("Top 10 Source Cities Based on Number of trips", {'font':'serif', 'weight': 'bol

plt.bar_label(container=s.containers[0])
```

[]



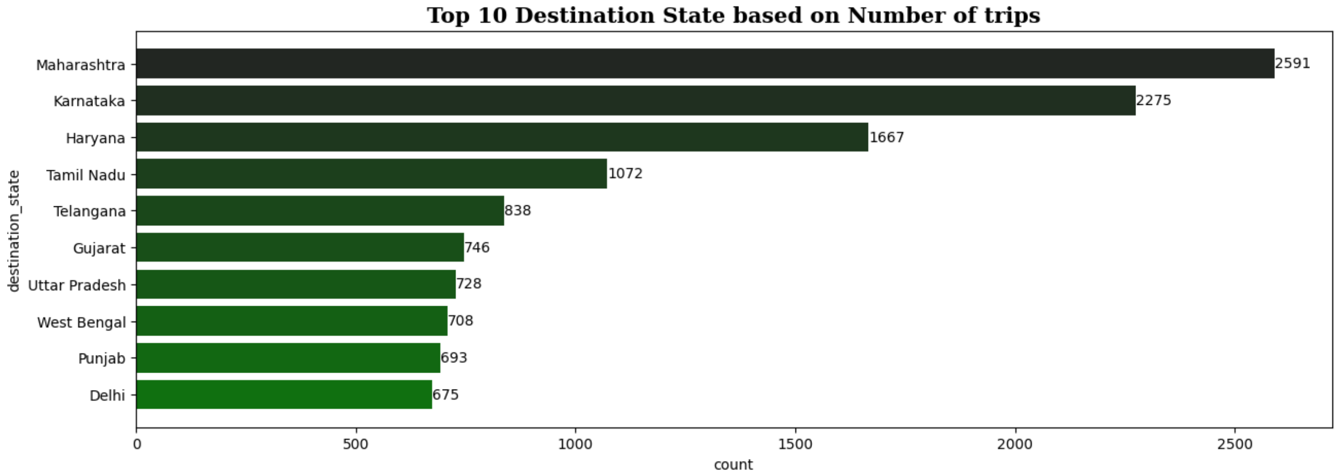
Distribution of trips based on Destination-state

```
#Top 10 source-state
T10_destination_state = df_nw.groupby(df_nw.destination_state)['trip_uuid'].count().sort_val
T10_destination_state[:10]
```

```
destination_state
Maharashtra      2591
Karnataka        2275
Haryana          1667
Tamil Nadu       1072
Telangana         838
Gujarat           746
Uttar Pradesh     728
West Bengal       708
Punjab            693
Delhi             675
Name: trip_uuid, dtype: int64
```

```
plt.figure(figsize = (15, 5))

d = sns.countplot(data = df_nw, order = df_nw['destination_state'].value_counts().index[:10]

plt.title("Top 10 Destination State based on Number of trips", {'font':'serif', 'weight': 'b

plt.bar_label(container=d.containers[0])

plt.plot()
```

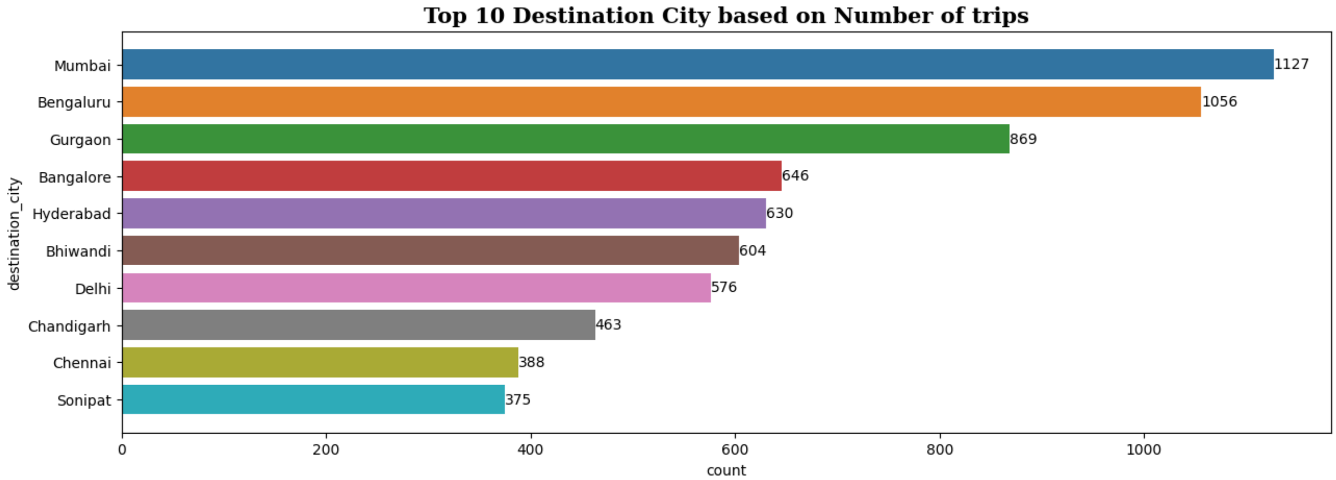
```
[]
```



Distribution of trips based on Destination-city

```
plt.figure(figsize = (15, 5))

c = sns.countplot(data = df_nw, order = df_nw['destination_city'].value_counts().index[:10],

plt.title("Top 10 Destination City based on Number of trips", {'font':'serif', 'weight': 'bo

plt.bar_label(container=c.containers[0])

plt.plot()
```

[]



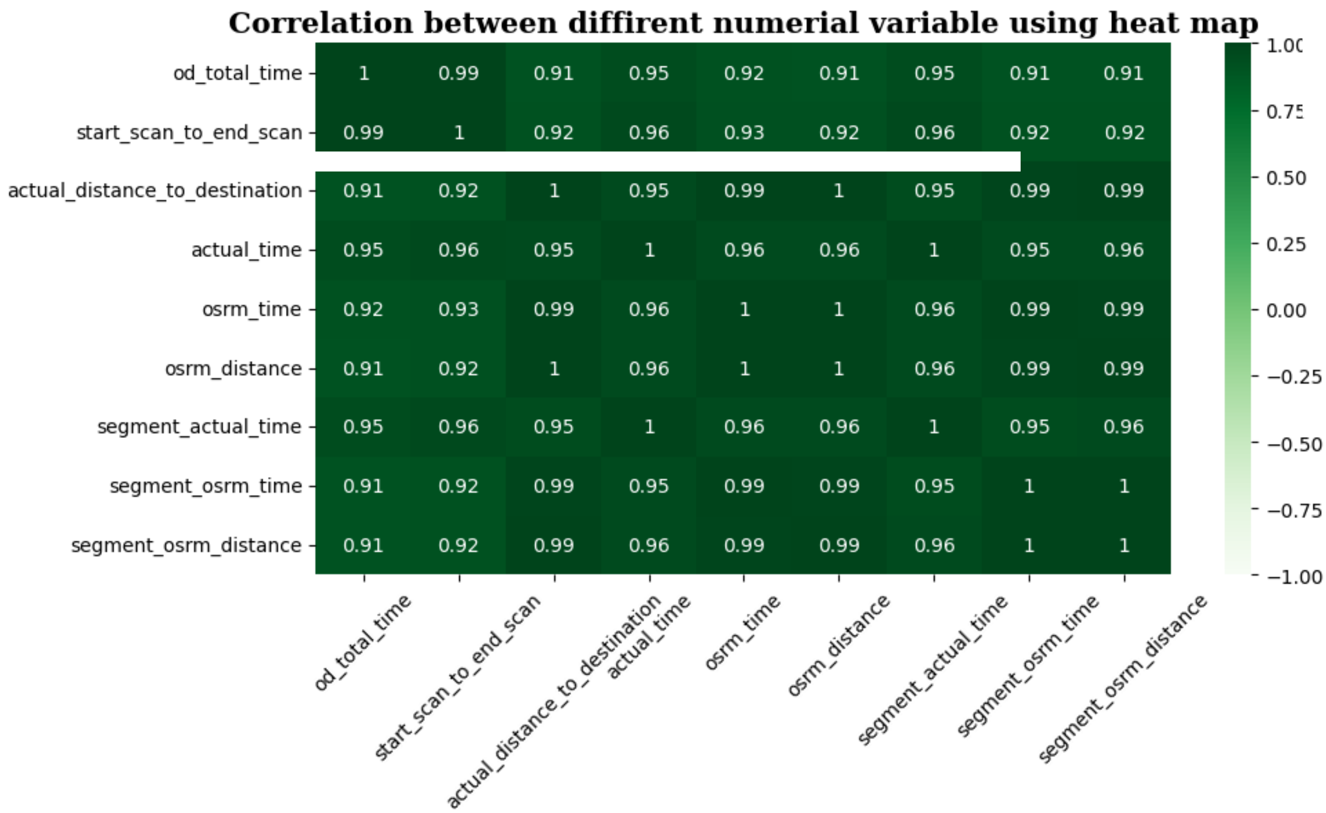
Numerical Variables

```
#Heat Map

numerical_columns = ['od_total_time', 'start_scan_to_end_scan', 'actual_distance_to_destinat
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance']

df_corr = df_nw[numerical_columns].corr()
df_corr
```

	od_total_time	start_scan_to_end_scan	actual_distance_to_
od_total_time	1.000000	0.993619	
start_scan_to_end_scan	0.993619	1.000000	
actual_distance_to_destination	0.906813	0.918308	
actual_time	0.952580	0.961147	
osrm_time	0.916065	0.926571	
osrm_distance	0.913205	0.924299	
segment_actual_time	0.952656	0.961171	
segment_osrm_time	0.907616	0.918561	
segment_osrm_distance	0.907676	0.919291	

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

sns.heatmap(data = df_corr, cmap = 'Greens', annot = True, vmin = -1, vmax = 1)
```

```
plt.title('Correlation between diffirent numerial variable using heat map', font='serif',
plt.xticks(rotation=45)
```

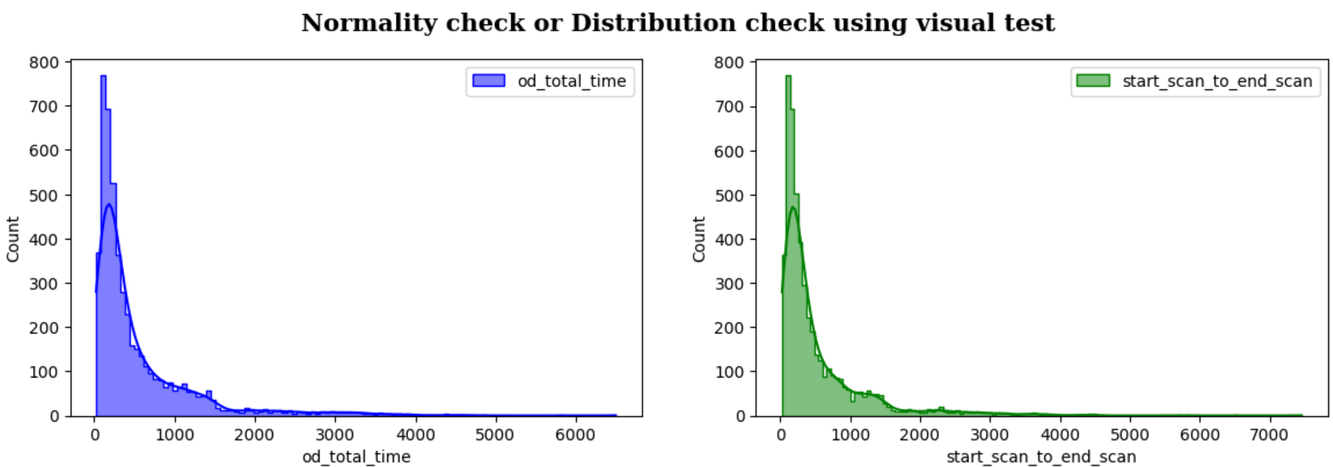


In-depth analysis and feature engineering:

Question: Compare the difference between od_total_time and start_scan_to_end_scan. Hypothesis Testing/Visual Analysis

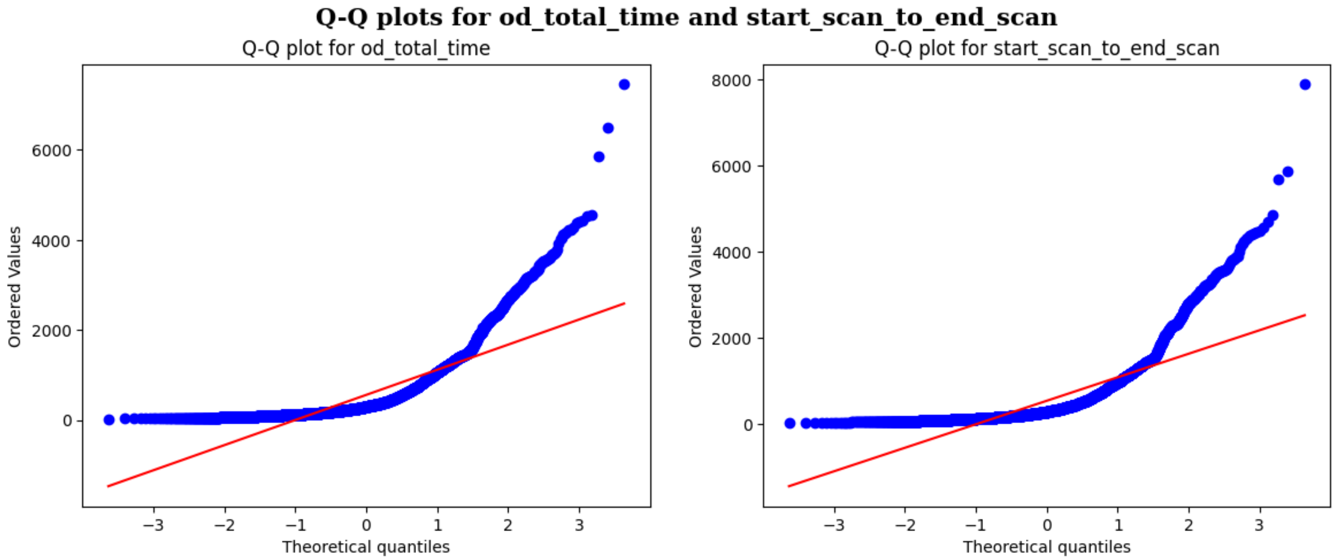
Distribution check or Normality check by Visual Tests

```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df_nw['od_total_time'].sample(5000), element = 'step', kde = True, color = 'blu
plt.legend()

plt.subplot(1, 2, 2)
sns.histplot(df_nw['start_scan_to_end_scan'].sample(5000), element = 'step', kde = True, col
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=1
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 5))
plt.suptitle('Q-Q plots for od_total_time and start_scan_to_end_scan', font='serif', size=15

plt.subplot(1, 2, 1)
probplot(df_nw['od_total_time'].sample(5000), plot = plt, dist = 'norm')
plt.title('Q-Q plot for od_total_time')

plt.subplot(1, 2, 2)
probplot(df_nw['start_scan_to_end_scan'].sample(5000), plot = plt, dist = 'norm')
plt.title('Q-Q plot for start_scan_to_end_scan')
plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for od_total_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['od_total_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for start_scan_to_end_scan)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['start_scan_to_end_scan'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

Normality Check or Distribution Check after boxcox transformation:

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for od_total_time)

transformed_od_total_time = spy.boxcox(df_nw['od_total_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_od_total_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 6.041245108788226e-27
Reject Ho. The sample does not follow normal distribution
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning: p-v
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for start_scan_to_end_

transformed_start_scan_to_end_scan = spy.boxcox(df_nw['start_scan_to_end_scan'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_start_scan_to_end_scan)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 1.0471322892609475e-24
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance
od_total_time_sample =df_nw['od_total_time'].sample(5000)
start_scan_to_end_scan_sample = df_nw['start_scan_to_end_scan'].sample(5000)
```

```
alpha = 0.05
test_stat, p_value = levene(od_total_time_sample, start_scan_to_end_scan_sample
print('p-value', p_value)
if p_value < alpha:
    print('reject Ho: The samples do not have  Homogenous Variance')
else:
```

```
p-value 0.010677279688423613
reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test
ks_stat,p_value = kstest(df_nw['od_total_time'], df_nw['start_scan_to_end_scan'])

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.01626510089761768
P value is: 0.039264199380179554
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): od_total_time (Total Trip Time) and start_scan_to_end_scan (Expectec
#   Alternative Hypothesis (Ha): od_total_time (Total Trip Time) and start_scan_to_end_scan
```

```
alpha = 0.05
if p_value < alpha:
  print('Reject Ho: od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected tot
else:
  print('Accept Ho: od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected tot
```

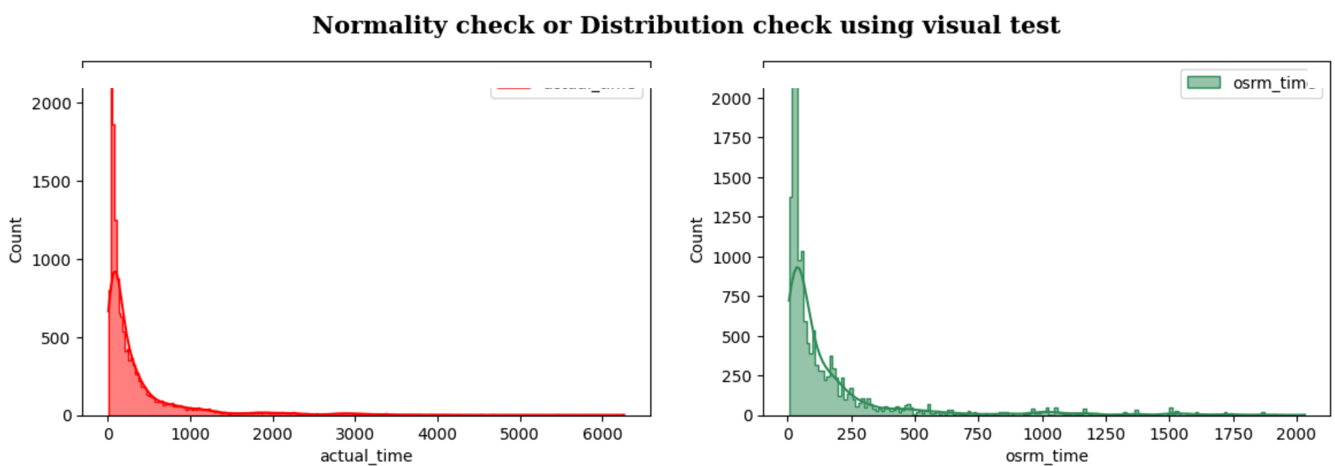
```
Reject Ho: od_total_time (Total Trip Time) and start_scan_to_end_scan (Expected total tr
```

Question: Hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value

Normality Check or Distribution Check using Histogram or visual

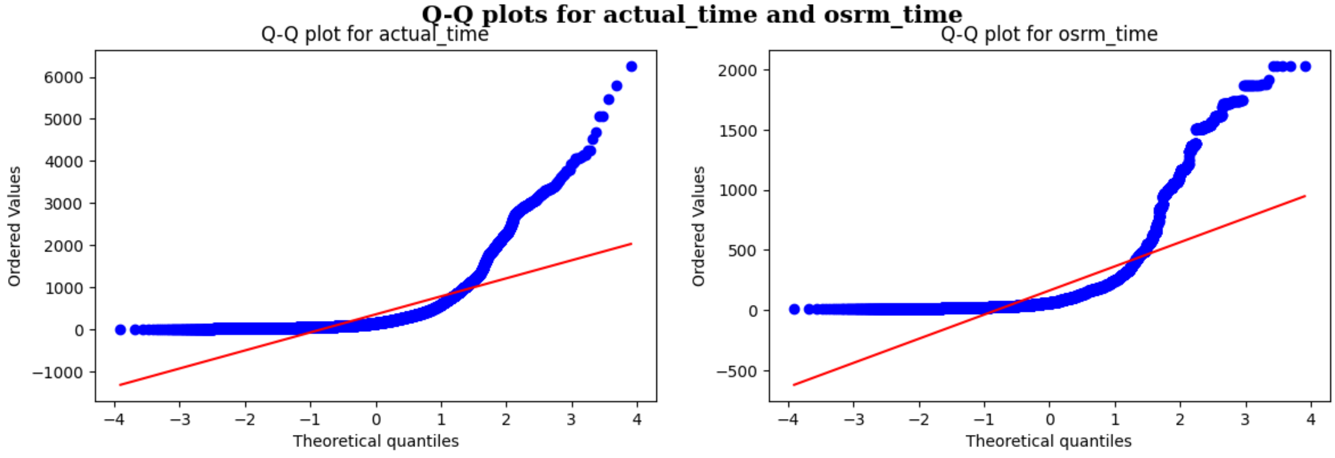
```
# Distribution check or Normality check by Visual Tests
```

```
plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df_nw['actual_time'], element = 'step', color = 'red', kde = True, label = 'a
plt.legend()

plt.subplot(1, 2, 2)
sns.histplot(df_nw['osrm_time'], element = 'step', color = 'seagreen', kde = True, label =
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 4))
plt.suptitle('Q-Q plots for actual_time and osrm_time', font='serif', size=15, weight='bold'

plt.subplot(1, 2, 1)
probplot(df_nw['actual_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for actual_time')

plt.subplot(1, 2, 2)
probplot(df_nw['osrm_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for osrm_time')

plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for actual_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for osrm_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

Normality Check or Distribution Check after boxcox transformation:

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for actual_time)

transformed_actual_time = spy.boxcox(df_nw['actual_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
Reject Ho. The sample does not follow normal distribution
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning: p-v
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for osrm_time)

transformed_osrm_time = spy.boxcox(df_nw['osrm_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance
actual_time_sample =df_nw['actual_time'].sample(5000)
osrm_time_sample = df_nw['osrm_time'].sample(5000)

alpha = 0.05
test_stat, p_value = levene(actual_time_sample, osrm_time_sample)
```

```
print('p-value', p_value)
if p_value < alpha:
    print('reject Ho: The samples do not have  Homogenous Variance')
else:
```

```
p-value 9.747074590283541e-77
reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test
ks_stat,p_value = kstest(df_nw['actual_time'], df_nw['osrm_time'])

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.2973611392319633
P value is: 0.0
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): actual_time aggregated value and osrm_time aggregated value are same
# Alternative Hypothesis (Ha): actual_time aggregated value and osrm_time aggregated value a

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: actual_time aggregated value and osrm_time aggregated value are not same
else:
  print('Accept Ho: actual_time aggregated value and osrm_time aggregated value are same.')
```

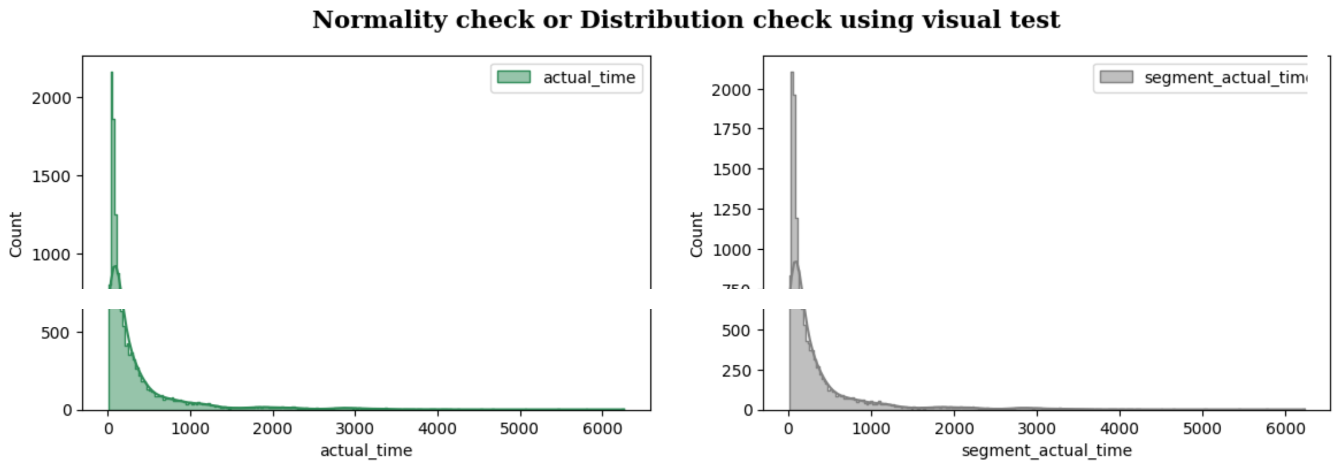
```
Reject Ho: actual_time aggregated value and osrm_time aggregated value are not same
```

Question: Hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value

Normality Check or Distribution Check using Histogram or visual

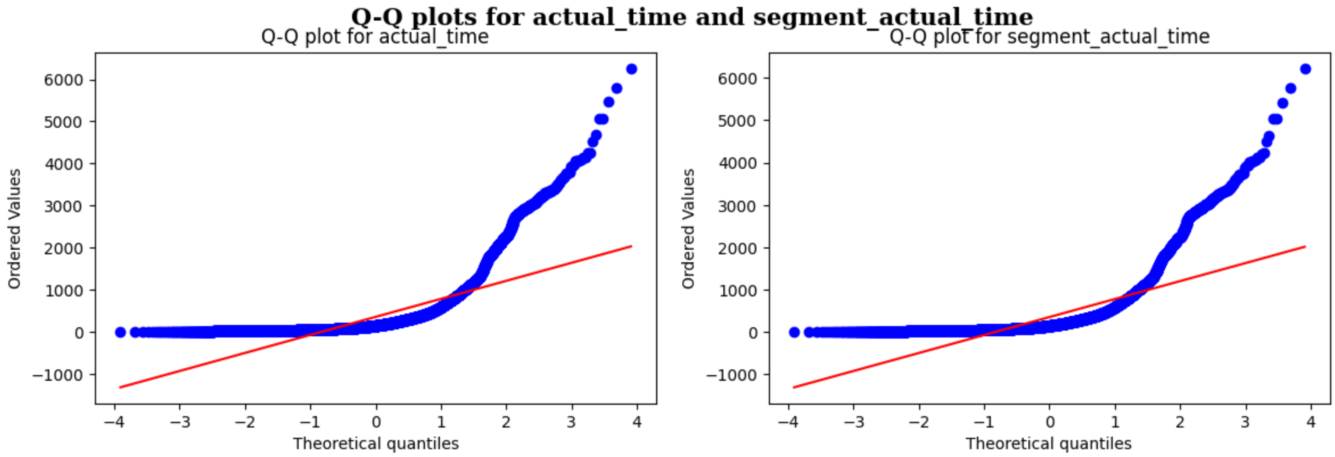
```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df_nw['actual_time'], element = 'step', color = 'seagreen', kde = True, label
plt.legend()
```

```
plt.subplot(1, 2, 2)
sns.histplot(df_nw['segment_actual_time'], element = 'step', color = 'grey', kde = True, la
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 4))
plt.suptitle('Q-Q plots for actual_time and segment_actual_time', font='serif', size=15, wei

plt.subplot(1, 2, 1)
probplot(df_nw['actual_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for actual_time')

plt.subplot(1, 2, 2)
probplot(df_nw['segment_actual_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for segment_actual_time')

plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for actual_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['actual_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for segment_actual_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['segment_actual_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

Normality Check or Distribution Check after boxcox transformation:

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for actual_time)

transformed_actual_time = spy.boxcox(df_nw['actual_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_actual_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 1.020620453603145e-28
Reject Ho. The sample does not follow normal distribution
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning: p-v
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for segment_actual_tim

transformed_segment_actual_time = spy.boxcox(df_nw['segment_actual_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_segment_actual_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 5.700074948787037e-29
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance

alpha = 0.05
test_stat, p_value = levene(df_nw['actual_time'], df_nw['segment_actual_time']
print('p-value', p_value)
if p_value < alpha:
```

```
    print('Reject Ho: The samples do not have  Homogenous Variance')
else:
```

```
p-value 0.695502241317651
Fail to Reject Ho: The samples have Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test
ks_stat,p_value = kstest(df_nw['actual_time'], df_nw['segment_actual_time'])

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.006344064250523029
P value is: 0.9248583909392553
```

Decision to Accept or Reject Null hypothesis

```
# Null Hypothesis (Ho): actual_time aggregated value and segment_actual_time aggregated valu
# Alternative Hypothesis (Ha): actual_time aggregated value and segment_actual_time aggregat

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: actual_time aggregated value and segment_actual_time aggregated value ar
else:
  print('Accept Ho: actual_time aggregated value and segment_actual_time aggregated value ar
```

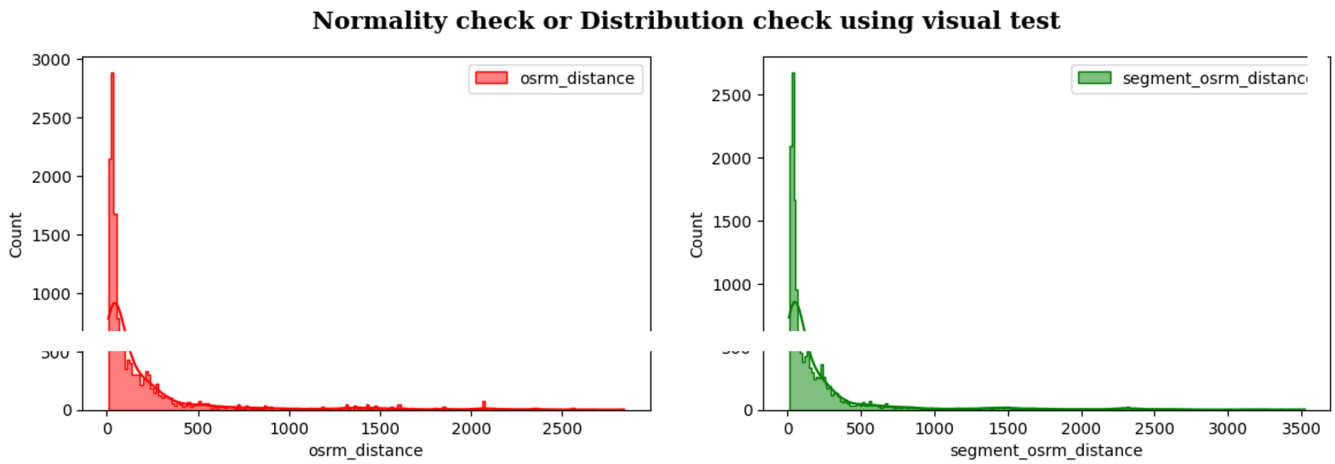
```
Accept Ho: actual_time aggregated value and segment_actual_time aggregated value are sar
```

Question: Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

Normality Check or Distribution Check using Histogram or visual

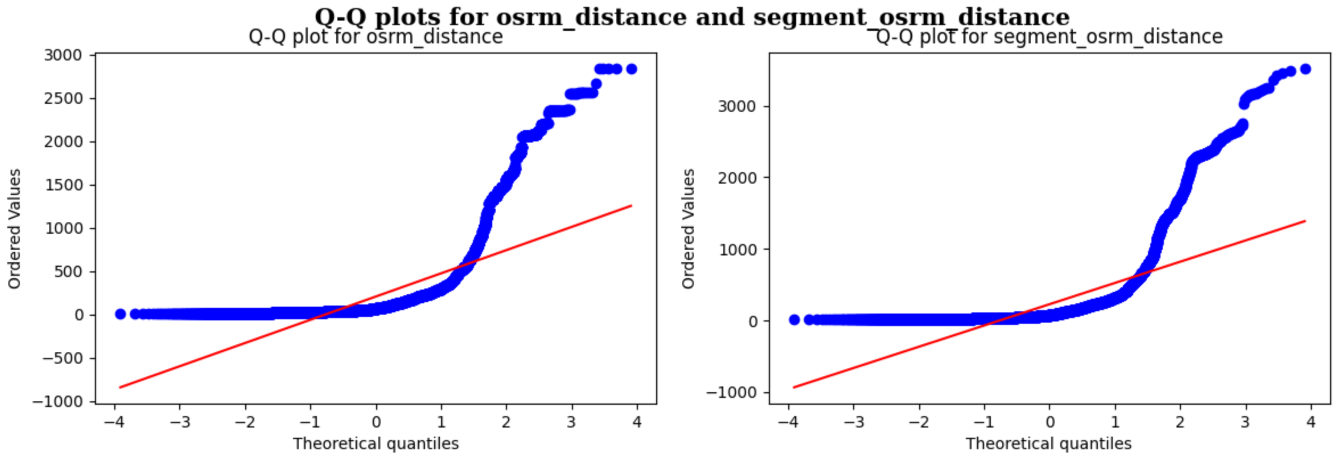
```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df_nw['osrm_distance'], element = 'step', color = 'red', kde = True, label =
plt.legend()

plt.subplot(1, 2, 2)
```

```
sns.histplot(df_nw['segment_osrm_distance'], element = 'step', color = 'green', kde = True
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 4))
plt.suptitle('Q-Q plots for osrm_distance and segment_osrm_distance', font='serif', size=15,

plt.subplot(1, 2, 1)
probplot(df_nw['osrm_distance'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for osrm_distance')

plt.subplot(1, 2, 2)
probplot(df_nw['segment_osrm_distance'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for segment_osrm_distance')

plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for osrm_distance)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['osrm_distance'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for segment_osrm_distance)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['segment_osrm_distance'].sample(5000)
print('p-value', p_value)
```

```
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

Normality Check or Distribution Check after boxcox transformation:

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for osrm_distance)

transformed_osrm_distance = spy.boxcox(df_nw['osrm_distance'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_osrm_distance)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 7.061423221425618e-41
Reject Ho. The sample does not follow normal distribution
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning: p-v
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for osrm_distance)

transformed_segment_osrm_distance = spy.boxcox(df_nw['segment_osrm_distance'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_segment_osrm_distance)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 3.049169406432229e-38
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance

alpha = 0.05
test_stat, p_value = levene(df_nw['osrm_distance'], df_nw['segment_osrm_distance'])
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho: The samples do not have  Homogenous Variance')
else:
    print('Fail to Reject Ho: The samples have Homogenous Variance ')
```

```
p-value 0.00020976006524780905
Reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test
ks_stat,p_value = kstest(df_nw['osrm_distance'], df_nw['segment_osrm_distance'])

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.0416413578997098
P value is: 1.3413627761631081e-11
```

Decision to Accept or Reject Null Hypothesis

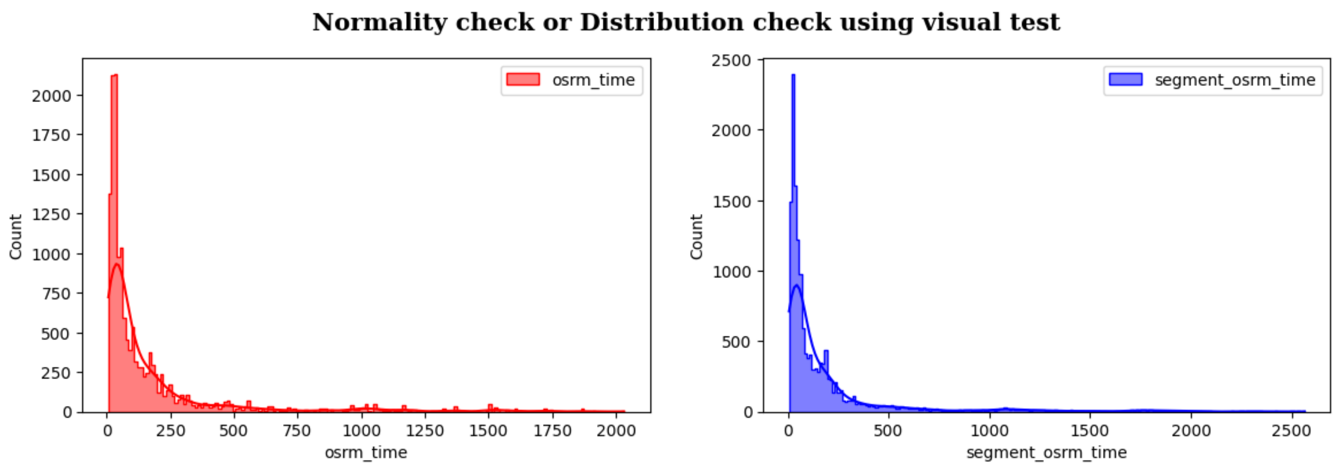
```
# Null Hypothesis (Ho): osrm_distance aggregated value and segment_osrm_distance aggregated
# Alternative Hypothesis (Ha): osrm_distance aggregated value and segment_osrm_distance aggr

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: osrm_distance aggregated value and segment_osrm_distance aggregated valu
else:
  print('Accept Ho: osrm_distance aggregated value and segment_osrm_distance aggregated valu
```

```
Reject Ho: osrm_distance aggregated value and segment_osrm_distance aggregated value are
```

Question: Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value

Normality Check or Distribution Check using Histogram or visual

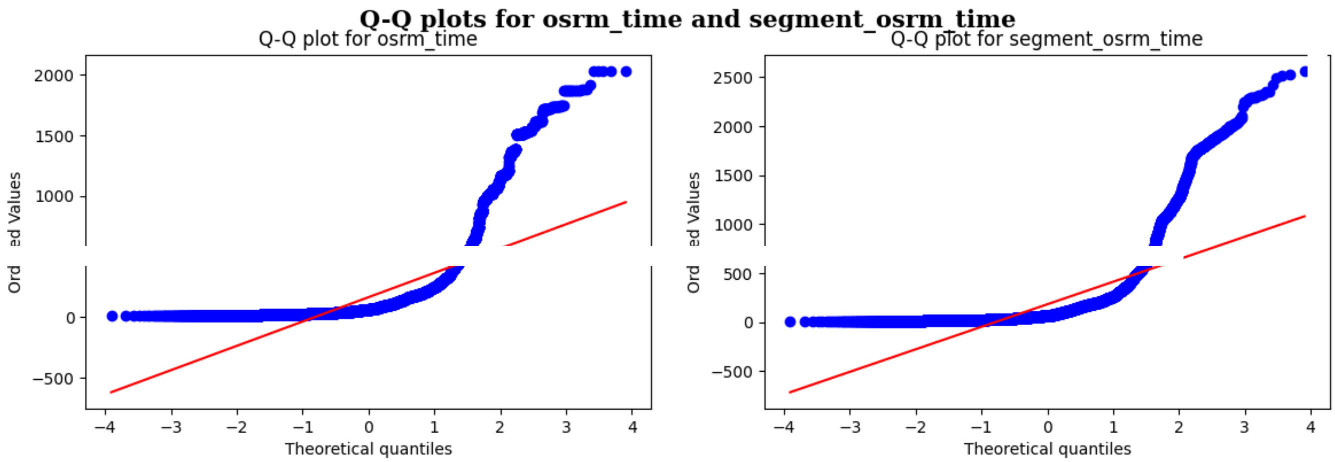
```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df_nw['osrm_time'], element = 'step', color = 'red', kde = True, label = 'osrm_
plt.legend()

plt.subplot(1, 2, 2)
sns.histplot(df_nw['segment_osrm_time'],  element = 'step', color = 'blue', kde = True, labe
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=1
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 4))
plt.suptitle('Q-Q plots for osrm_time and segment_osrm_time', font='serif', size=15, weigh

plt.subplot(1, 2, 1)
```

```
probplot(df_nw['osrm_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for osrm_time')

plt.subplot(1, 2, 2)
probplot(df_nw['segment_osrm_time'], plot = plt, dist = 'norm')
plt.title('Q-Q plot for segment_osrm_time')
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for osrm_time)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for segment_osrm_time)
```

```
# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df_nw['segment_osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution'
else:
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

Normality Check or Distribution Check after boxcox transformation:

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for osrm_time)

transformed_osrm_time = spy.boxcox(df_nw['osrm_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_osrm_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 3.5882550510138333e-35
Reject Ho. The sample does not follow normal distribution
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning: p-v
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
#Normality Check using Shapiro-Wilk test after Boxcox transformation (for segment_osrm_tim

transformed_segment_osrm_time = spy.boxcox(df_nw['segment_osrm_time'])[0]

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(transformed_segment_osrm_time)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
```

```
p-value 4.943039152219146e-34
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance

alpha = 0.05
test_stat, p_value = levene(df_nw['osrm_time'], df_nw['segment_osrm_time'])
print('p-value', p_value)
if p_value < alpha:
    print('reject Ho: The samples do not have  Homogenous Variance')
else:
    print('Fail to Reject Ho: The samples have Homogenous Variance ')
```

```
p-value 8.349506135727595e-08
reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test

ks_stat,p_value = kstest(df_nw['osrm_time'], df_nw['segment_osrm_time'])

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.0363096443274617
P value is: 6.383943701595088e-09
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): osrm_time aggregated value and segment_osrm_time aggregated value ar
# Alternative Hypothesis (Ha): osrm_time aggregated value and segment_osrm_time aggregated v


alpha = 0.05
if p_value < alpha:
  print('Reject Ho: osrm_time aggregated value and segment_osrm_time aggregated value are nc
else:
  print('Accept Ho: osrm_time aggregated value and segment_osrm_time aggregated value are sa
```

```
Reject Ho: osrm_time aggregated value and segment_osrm_time aggregated value are not san
```

Finding outliers in the numerical variables

```
df_nw.columns
```

```
Index(['trip_uuid', 'source_center', 'destination_center', 'data',
       'route_type', 'trip_creation_time', 'source_name', 'destination_name',
       'start_scan_to_end_scan', 'actual_distance_to_destination',
       'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'source_state',
       'source_city', 'source_place', 'destination_state', 'destination_city',
       'destination_place', 'trip_creation_date', 'trip_creation_year',
       'trip_creation_month', 'trip_creation_day', 'trip_creation_hour',
       'od_total_time'],
      dtype='object')
```

Finding outliers by data analysis

```
numerical_columns = ['start_scan_to_end_scan', 'actual_distance_to_destination',
                     'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                     'segment_osrm_time', 'segment_osrm_distance','od_total_time']
df_nw[numerical_columns].describe().T
```

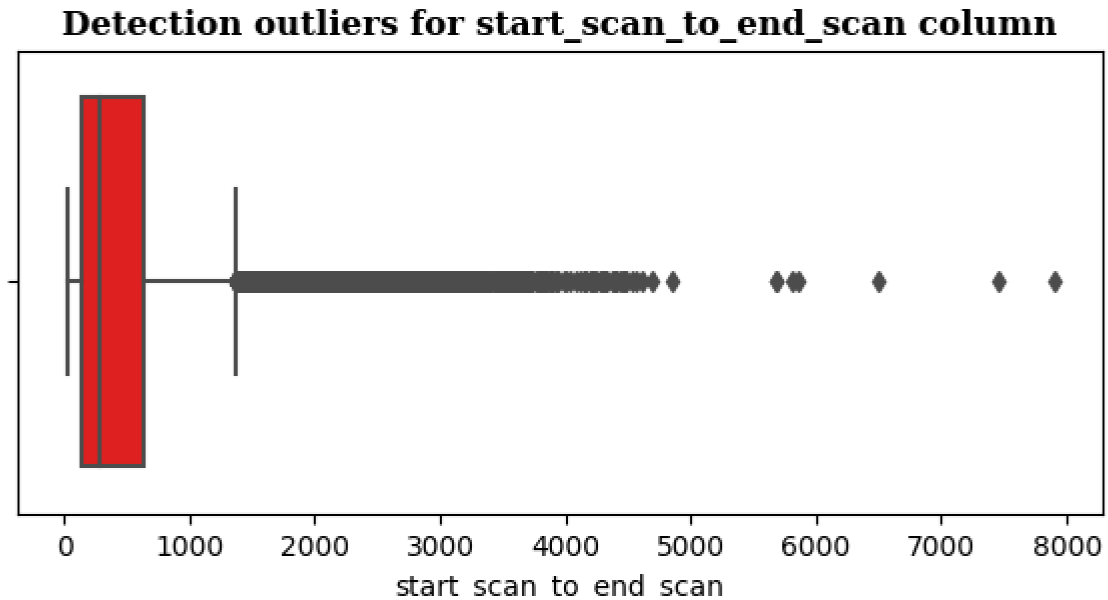
	count	mean	std	min	25%	
start_scan_to_end_scan	14817.0	530.810016	658.705957	23.000000	149.000000	280.0
actual_distance_to_destination	14817.0	164.477829	305.388153	9.002461	22.837238	48.4
actual_time	14817.0	357.143768	561.396118	9.000000	67.000000	149.0
osrm_time	14817.0	161.384018	271.360992	6.000000	29.000000	60.0
osrm_distance	14817.0	204.344711	370.395569	9.072900	30.819201	65.6
segment_actual_time	14817.0	353.892273	556.247925	9.000000	66.000000	147.0
segment_osrm_time	14817.0	180.949783	314.542053	6.000000	31.000000	65.0
segment_osrm_distance	14817.0	223.201157	416.628387	9.072900	32.654499	70.1
od_total_time	14817.0	547.462995	668.655943	23.460000	151.160000	288.5

Outliers detection by Boxplot:

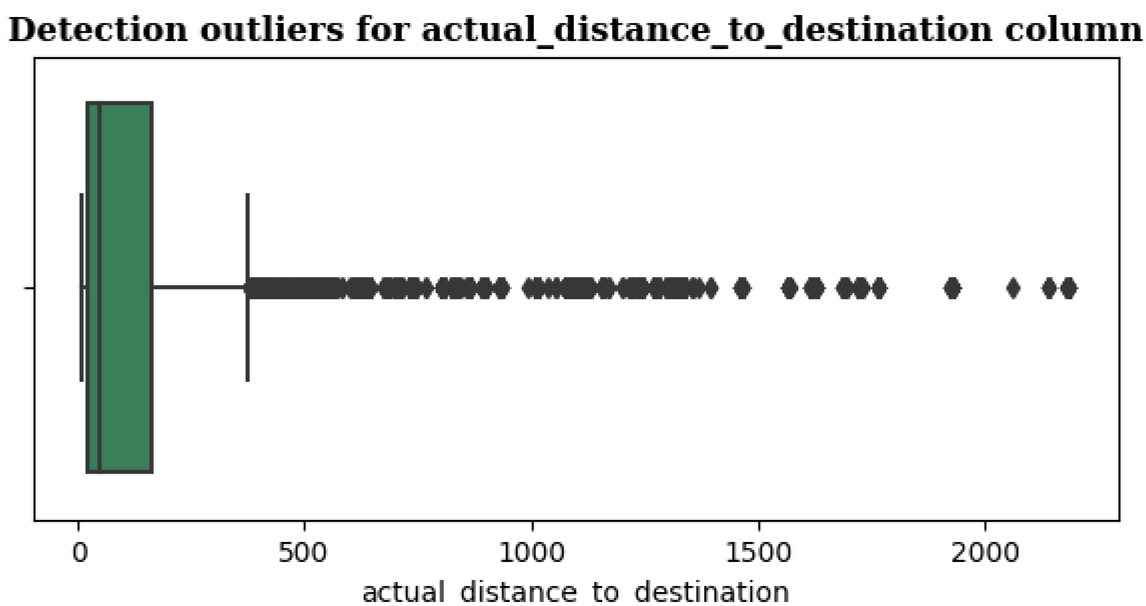
```
def outliers(x, clr):
  plt.figure(figsize = (7, 3))

  sns.boxplot(x = df_nw[x], color = clr)
```

```
    plt.title(f"Detection outliers for {x} column", font='serif', weight='bold', size=12
```

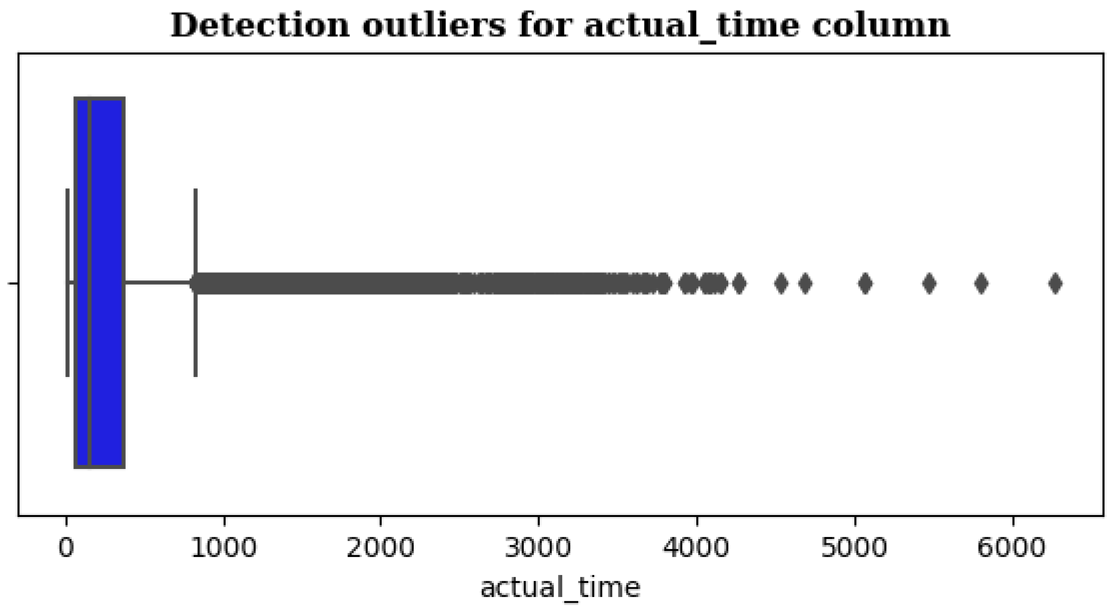
```
outliers('start_scan_to_end_scan','red')
```



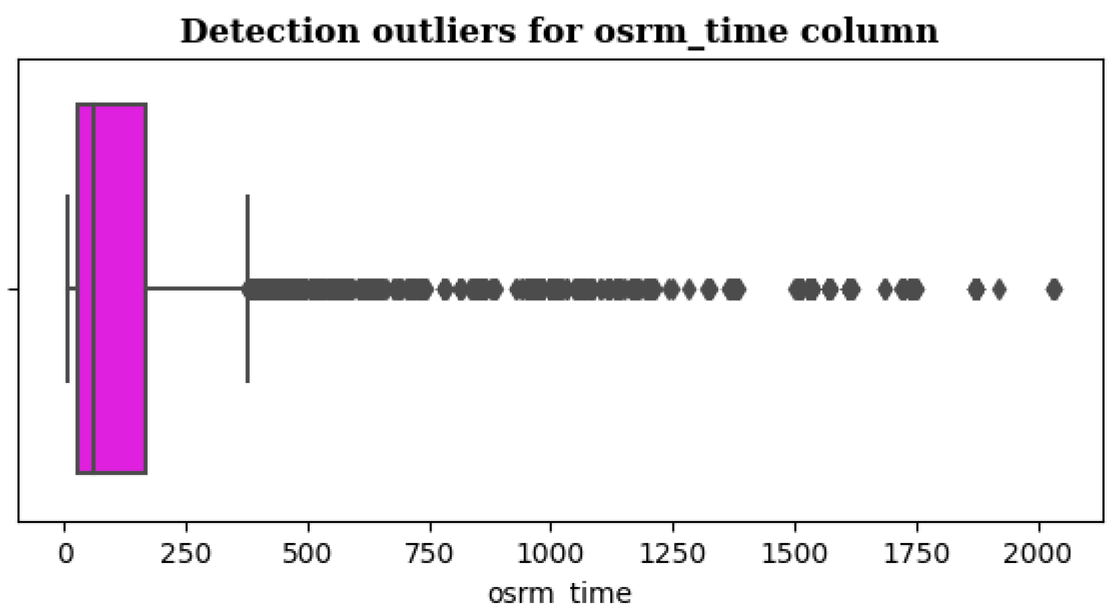

```
outliers('actual_distance_to_destination','seagreen')
```



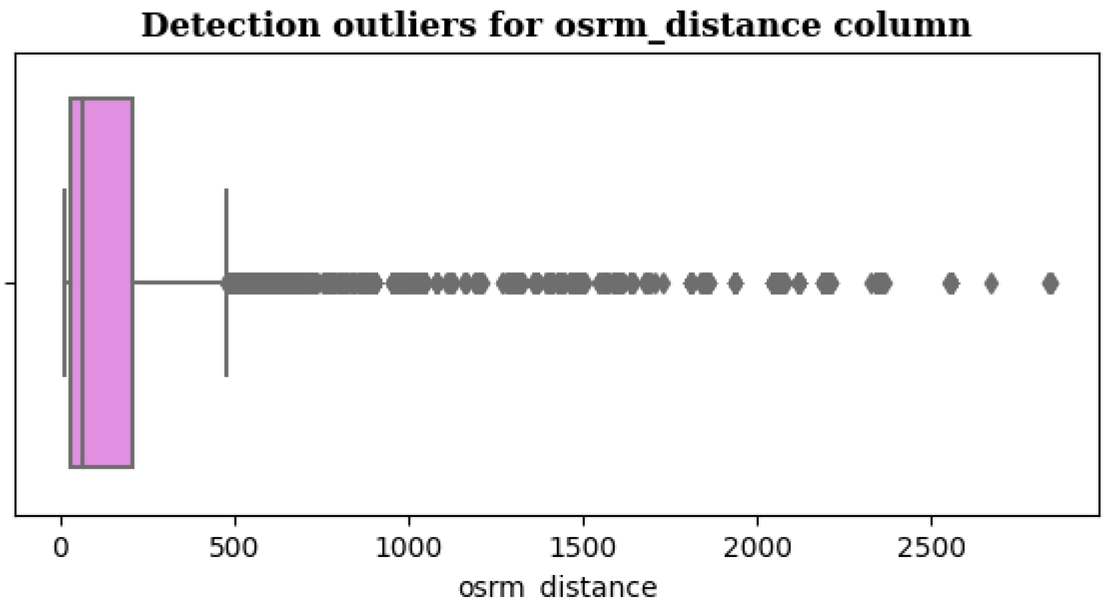

```
outliers('actual_time','blue')
```



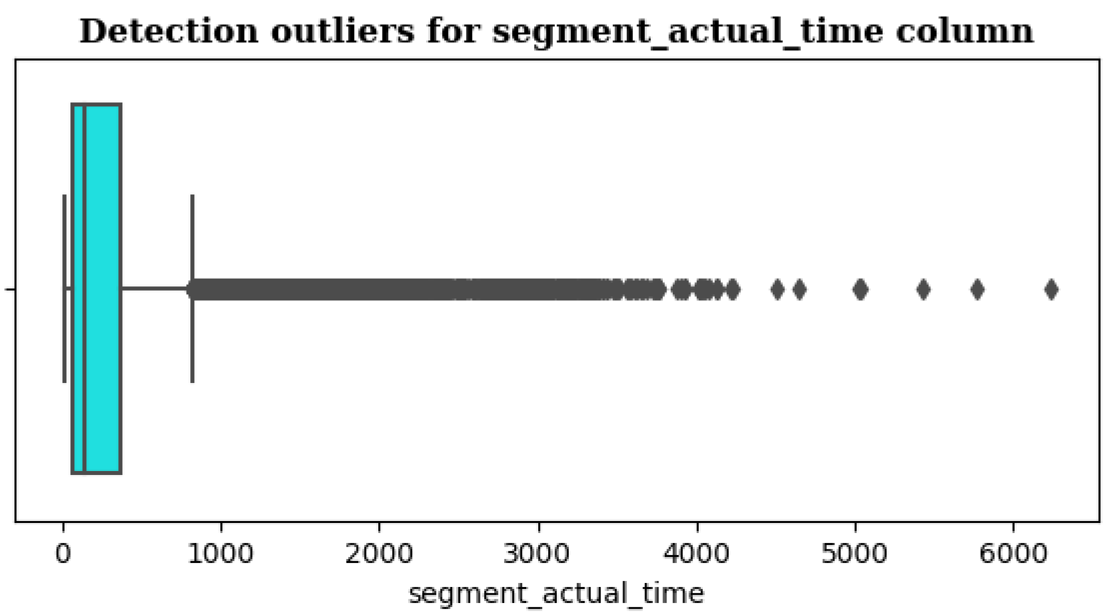

```
outliers('osrm_time','magenta')
```



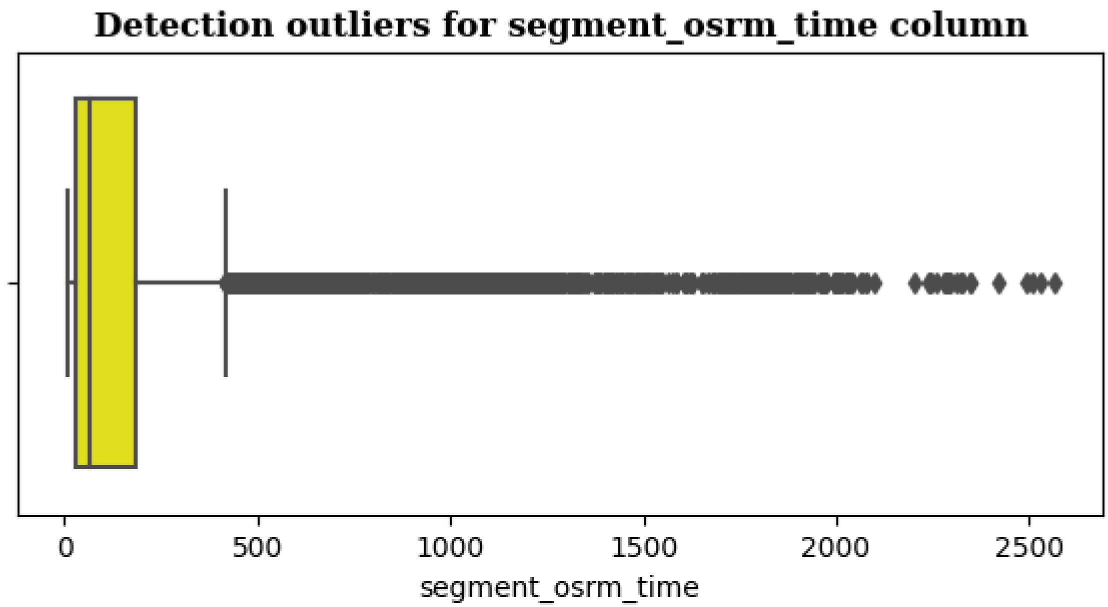

```
outliers('osrm_distance','violet')
```



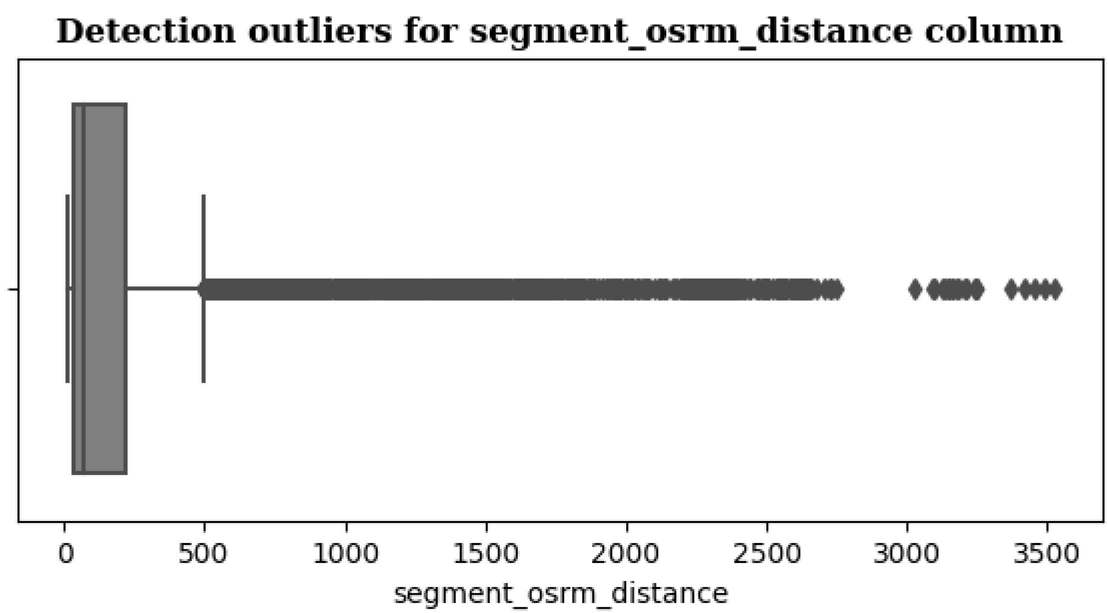

```
outliers('segment_actual_time','cyan')
```



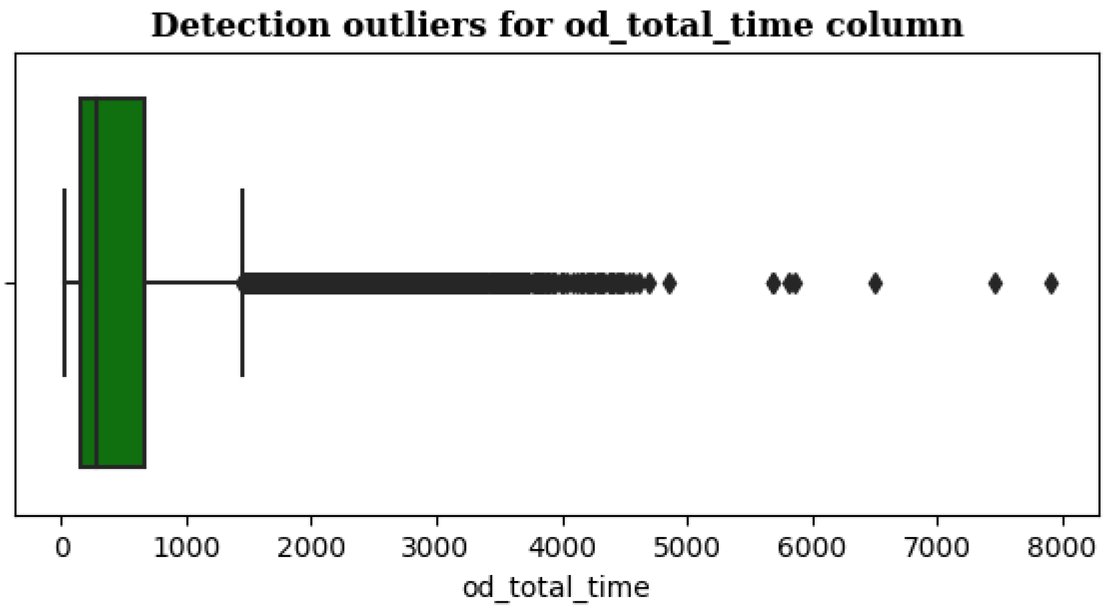

```
outliers('segment_osrm_time','yellow')
```




```
outliers('segment_osrm_distance','grey')
```




```
outliers('od_total_time','green')
```



Handling the outliers using IQR method

Detecting outliers using IQR

```
#Detecting IQR
def detect_by_IQR(i):
  Q1 = np.quantile(df_nw[i], 0.25)
  Q3 = np.quantile(df_nw[i], 0.75)
  IQR = Q3 - Q1
  Lower_outlier = Q1 - 1.5*IQR
  Higher_outlier = Q3 + 1.5*IQR
  outliers = df_nw.loc[(df_nw[i] < Lower_outlier) | (df_nw[i] > Higher_outlier)]
  print('Column :', i)
  print(f'Q1 : {Q1}')
  print(f'Q3 : {Q3}')
  print(f'IQR : {IQR}')
  print(f'Lower outlier : {Lower_outlier}')
  print(f'Upper outlier : {Higher_outlier}')
  print(f'Number of outliers : {outliers.shape[0]}')
  print('----------------------------------')
```

```
numerical_columns = ['start_scan_to_end_scan', 'actual_distance_to_destination',
                    'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
                    'segment_osrm_time', 'segment_osrm_distance','od_total_time']
for i in numerical_columns:
  detect_by_IQR(i)
```

```
Column : start_scan_to_end_scan
Q1 : 149.0
Q3 : 637.0
```

```
IQR : 488.0
Lower outlier : -583.0
Upper outlier : 1369.0
Number of outliers : 1267
----------------------------------
Column : actual_distance_to_destination
Q1 : 22.837238311767578
Q3 : 164.5832061767578
IQR : 141.74596786499023
Lower outlier : -189.78171348571777
Upper outlier : 377.20215797424316
Number of outliers : 1449
----------------------------------
Column : actual_time
Q1 : 67.0
Q3 : 370.0
IQR : 303.0
Lower outlier : -387.5
Upper outlier : 824.5
Number of outliers : 1643
----------------------------------
Column : osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
Lower outlier : -179.5
Upper outlier : 376.5
Number of outliers : 1517
----------------------------------
Column : osrm_distance
Q1 : 30.81920051574707
Q3 : 208.47500610351562
IQR : 177.65580558776855
Lower outlier : -235.66450786590576
Upper outlier : 474.95871448516846
Number of outliers : 1524
----------------------------------
Column : segment_actual_time
Q1 : 66.0
Q3 : 367.0
IQR : 301.0
Lower outlier : -385.5
Upper outlier : 818.5
Number of outliers : 1643
----------------------------------
Column : segment_osrm_time
Q1 : 31.0
Q3 : 185.0
IQR : 154.0
Lower outlier : -200.0
Upper outlier : 416.0
Number of outliers : 1492
----------------------------------
Column : segment_osrm_distance
Q1 : 32.65449905395508
```

Handling the outliers using IQR by trimming method

```
df_without_out = df_nw.copy(deep = True)

for i in numerical_columns:
  Q1 = np.quantile(df_without_out[i], 0.25)
  Q3 = np.quantile(df_without_out[i], 0.75)
  IQR = Q3 - Q1
  Lower_outlier = Q1 - 1.5*IQR
  Higher_outlier =Q3 + 1.5*IQR
    #outliers = df_without_out.loc[(df_without_out[i] < Lower_outlier) | (df_without_out[i]
  #new dataframe without outliers
  df_without_out = df_without_out.loc[(df_without_out[i] > Lower_outlier) & (df_without_out[
  print(f'Shape of dataframe after removing outliers from {i} column : {df_without_out.shape
print('-'*30)
print('Shape of new dataframe after removing all outliers is: ', df_without_out.shape)
```

```
Shape of dataframe after removing outliers from start_scan_to_end_scan column : (13550,
Shape of dataframe after removing outliers from actual_distance_to_destination column :
Shape of dataframe after removing outliers from actual_time column : (12113, 28)
Shape of dataframe after removing outliers from osrm_time column : (11506, 28)
Shape of dataframe after removing outliers from osrm_distance column : (10782, 28)
Shape of dataframe after removing outliers from segment_actual_time column : (10444, 28)
Shape of dataframe after removing outliers from segment_osrm_time column : (9753, 28)
Shape of dataframe after removing outliers from segment_osrm_distance column : (9153, 28
Shape of dataframe after removing outliers from od_total_time column : (8701, 28)
------------------------------
Shape of new dataframe after removing all outliers is:  (8701, 28)
```

One-Hot Encoding of Categorical Variables

One-Hot Encoding for route types & data columns:

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

```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

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

```
training    10654
test         4163
Name: data, dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder

# Assigning numerical values and storing it in another columns
df_nw['route_type_nw'] = df_nw['route_type'].cat.codes
df_nw['data_nw'] = df_nw['data'].cat.codes

# Create an instance of One-hot-encoder
ohe = OneHotEncoder()

# Passing encoded columns
ohe_data = pd.DataFrame(ohe.fit_transform(df_nw[['route_type_nw', 'data_nw']]).toarray())

# Merge with main
New_df = df_nw.join(ohe_data)

New_df.head()
```

	trip_uuid	source_center	destination_center	data	route_type	trip_creat
0	trip-153671041653548748	IND462022AAA	IND000000ACB	training	FTL	2 00:00:
1	trip-153671042288605164	IND572101AAA	IND562101AAA	training	Carting	2 00:00:
2	trip-153671043369099517	IND562132AAA	IND160002AAC	training	FTL	2 00:00:
3	trip-153671046011330457	IND400072AAB	IND401104AAA	training	Carting	2 00:01:
4	trip-153671052974046625	IND583101AAA	IND583101AAA	training	FTL	2 00:02:

5 rows × 34 columns

```
df_nw['route_type_nw'].value_counts()
```

```
0    8908
1    5909
Name: route_type_nw, dtype: int64
```

```
df_nw['data_nw'].value_counts()
```

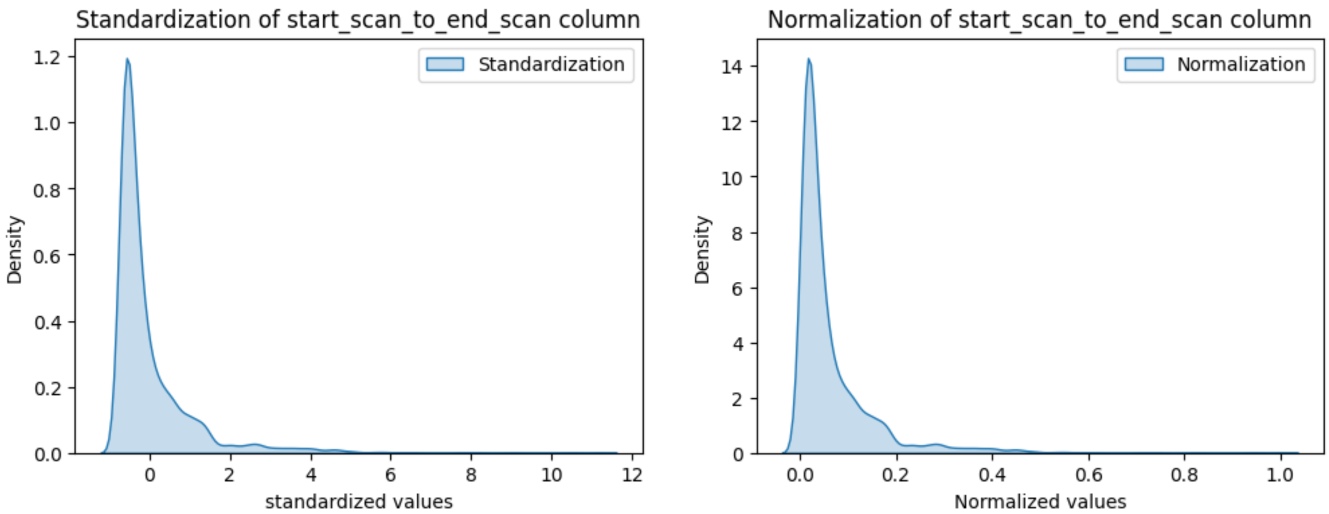
```
1    10654
0     4163
Name: data_nw, dtype: int64
```

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler

```
#importing standardscaler and minmaxscaler

from sklearn.preprocessing import StandardScaler, MinMaxScaler


def norm_stand(i):
  #normalization by MinMaxScaler
  df_normal = MinMaxScaler().fit_transform(df_nw[i].to_numpy().reshape(-1,1))

  #standardization by StandardScaler
  df_standard = StandardScaler().fit_transform(df_nw[i].to_numpy().reshape(-1, 1))

  #plot for normalization/standardization
  plt.figure(figsize=(12,4))

  plt.subplot(1,2,1)
  sns.kdeplot(df_standard, label= 'Standardization', fill = True, color="Red")
  plt.title(f'Standardization of {i} column')
  plt.xlabel('standardized values')
  plt.legend()

  plt.subplot(1,2,2)
  sns.kdeplot(df_normal, label= 'Normalization', fill = True, color="Green")
  plt.title(f'Normalization of {i} column')
  plt.xlabel('Normalized values')
  plt.legend()

  plt.show()
```

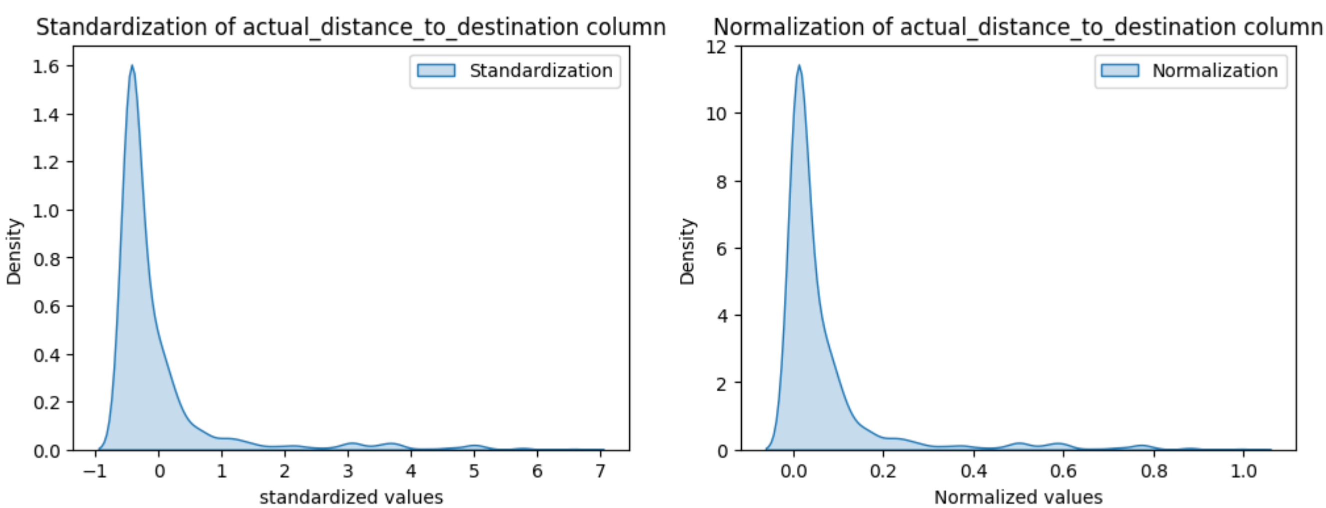
Normalization &Standardization for start_scan_to_end_scan column:

```
norm_stand("start_scan_to_end_scan")
```



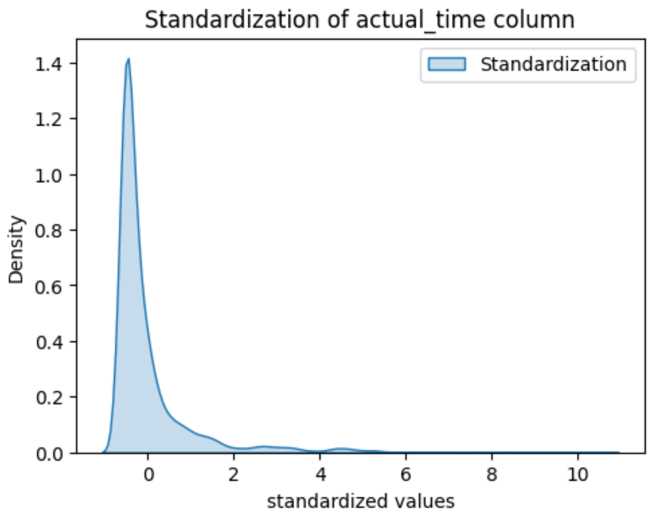
Normalization & Standardization for actual_distance_to_destination column:

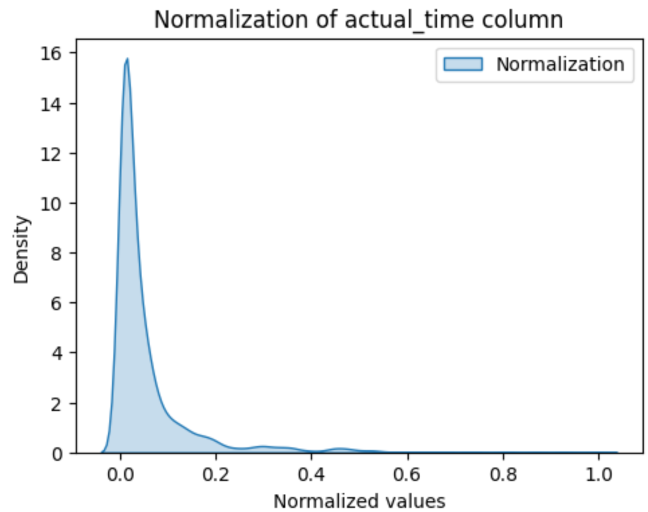
```
norm_stand("actual_distance_to_destination")
```



Normalization & Standardization for actual_time column:

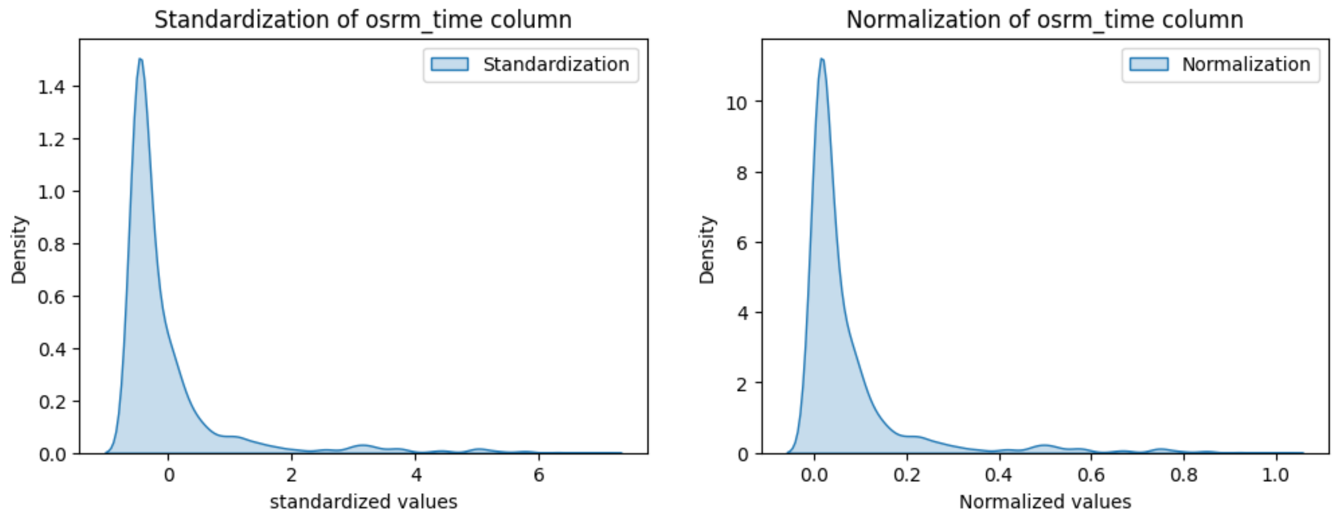
```
norm_stand("actual_time")
```





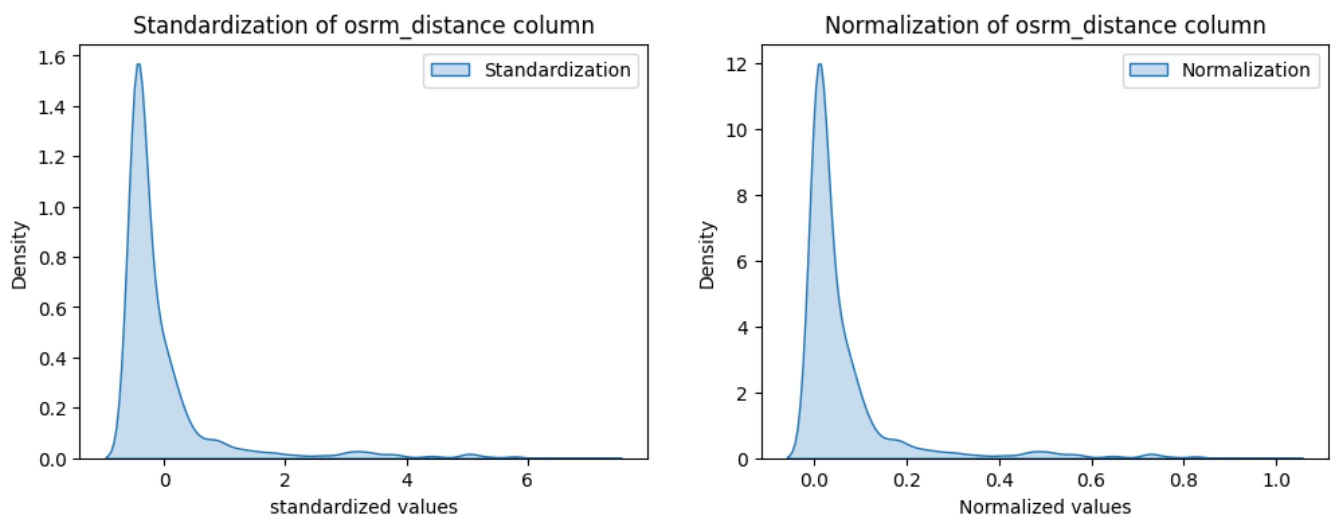
Normalization & Standardization for osrm_time column:

```
norm_stand("osrm_time")
```



Normalization & Standardization for osrm_distance column:

```
norm_stand("osrm_distance")
```



```
norm_stand("segment_actual_time")
```