In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import mannwhitneyu
```

In [2]:
```
df=pd.read_csv("Delhivery_data.csv")
```

In [3]:
```
df.head(10)
```

Out[3]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sourc
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38

10 rows × 24 columns

In [4]:
```
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
```

In [5]:
```
df.describe()
```

Out[5]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time
count	144867.000000	144867.000000	144867.000000	144867.000000
mean	961.262986	232.926567	234.073372	416.927527
std	1037.012769	344.755577	344.990009	598.103621
min	20.000000	9.000000	9.000045	9.000000
25%	161.000000	22.000000	23.355874	51.000000
50%	449.000000	66.000000	66.126571	132.000000
75%	1634.000000	286.000000	286.708875	513.000000
max	7898.000000	1927.000000	1927.447705	4532.000000

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
is_cutoff                           0
cutoff_factor                       0
cutoff_timestamp                    0
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
factor                              0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
segment_factor                      0
dtype: int64
```

In [7]:
```
missing_source = df.loc[df['source_name'].isnull(),'source_center'].unique(
missing_source
```

Out[7]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [8]:
```
for i in missing_source:
    name= df.loc[df['source_center']==i, 'source_name'].unique()
    if pd.isna(name):
        print(f'Source name not found for {i}')
```

```
Source name not found for IND342902A1B
Source name not found for IND577116AAA
Source name not found for IND282002AAD
Source name not found for IND465333A1B
Source name not found for IND841301AAC
Source name not found for IND509103AAC
Source name not found for IND126116AAA
Source name not found for IND331022A1B
Source name not found for IND505326AAB
Source name not found for IND852118A1B
```

In [9]:
```
missing_destination = df.loc[df['destination_name'].isnull(),'destination_c
missing_destination
```

Out[9]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [10]:
```
for i in missing_destination:
    dest_name= df.loc[df['destination_center']==i, 'destination_name'].uniq
    if pd.isna(dest_name):
        print(f'Destination name not found for {i}')
```

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

Source_name and destination_name columns have null values. Unable able to impute values using source_center and destination_center data as there is no known data for those source and destination centers.

So Removing all the rows that have null values as we cant work with rows that dont have source and destination names.

In [11]:
```
df.dropna(how='any',inplace=True)
df.reset_index(inplace=True)
```

In [12]:
```
# Changing the data types of columns 'data' and 'route_type' to category.

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

In [13]:
```
# Changing data types of columns od_start_time and od_end_time to datetime.

df['od_start_time'] =  pd.to_datetime(df['od_start_time'])
df['od_end_time']  =  pd.to_datetime(df['od_end_time'])
df['trip_creation_time']  =  pd.to_datetime(df['trip_creation_time'])
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144316 entries, 0 to 144315
Data columns (total 25 columns):
 #   Column                          Non-Null Count   Dtype         
---  ------                          --------------   -----         
 0   index                           144316 non-null  int64         
 1   data                            144316 non-null  category      
 2   trip_creation_time              144316 non-null  datetime64[ns]
 3   route_schedule_uuid             144316 non-null  object        
 4   route_type                      144316 non-null  category      
 5   trip_uuid                       144316 non-null  object        
 6   source_center                   144316 non-null  object        
 7   source_name                     144316 non-null  object        
 8   destination_center              144316 non-null  object        
 9   destination_name                144316 non-null  object        
 10  od_start_time                   144316 non-null  datetime64[ns]
 11  od_end_time                     144316 non-null  datetime64[ns]
 12  start_scan_to_end_scan          144316 non-null  float64       
 13  is_cutoff                       144316 non-null  bool          
 14  cutoff_factor                   144316 non-null  int64         
 15  cutoff_timestamp                144316 non-null  object        
 16  actual_distance_to_destination  144316 non-null  float64       
 17  actual_time                     144316 non-null  float64       
 18  osrm_time                       144316 non-null  float64       
 19  osrm_distance                   144316 non-null  float64       
 20  factor                          144316 non-null  float64       
 21  segment_actual_time             144316 non-null  float64       
 22  segment_osrm_time               144316 non-null  float64       
 23  segment_osrm_distance           144316 non-null  float64       
 24  segment_factor                  144316 non-null  float64       
dtypes: bool(1), category(2), datetime64[ns](3), float64(10), int64(2), ob
ject(7)
memory usage: 24.6+ MB
```

Merging rows to make the dataset compact without loosing data

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

Grouping by 'trip_uuid', 'source_center' and 'destination_center'.

And taking the sums of 'segment_actual_time', 'segment_osrm_distance' and 'segment_osrm_time' for each group.

In [15]:
```
df_grouped = df.groupby(by=['trip_uuid','source_center','destination_center

    'data' : 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_center' : 'first',
    'source_name' : 'first',

    'destination_center' : 'last',
    'destination_name' : 'last',

    'od_start_time' : 'first',
    'od_end_time' : 'first',
    '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_distance' : 'sum',
    'segment_osrm_time' : 'sum',

    })
```

In [16]:
```
df_grouped.rename(columns={'segment_actual_time':'total_segment_actual_time
```

In [17]:
```
df_grouped.head(10)
```

Out[17]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sour
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND2(
1	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND4(
2	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND5(
3	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND57
4	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND0(
5	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND5(
6	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND4(
7	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND58
8	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND58
9	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND58

In [18]:
```
df_grouped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26222 entries, 0 to 26221
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            26222 non-null  category
 1   trip_creation_time              26222 non-null  datetime64[ns]
 2   route_schedule_uuid             26222 non-null  object
 3   route_type                      26222 non-null  category
 4   trip_uuid                       26222 non-null  object
 5   source_center                   26222 non-null  object
 6   source_name                     26222 non-null  object
 7   destination_center              26222 non-null  object
 8   destination_name                26222 non-null  object
 9   od_start_time                   26222 non-null  datetime64[ns]
 10  od_end_time                     26222 non-null  datetime64[ns]
 11  start_scan_to_end_scan          26222 non-null  float64
 12  actual_distance_to_destination  26222 non-null  float64
 13  actual_time                     26222 non-null  float64
 14  osrm_time                       26222 non-null  float64
 15  osrm_distance                   26222 non-null  float64
 16  total_segment_actual_time       26222 non-null  float64
 17  total_segment_osrm_distance     26222 non-null  float64
 18  total_segment_osrm_time         26222 non-null  float64
dtypes: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 3.5+ MB
```

In [19]:
```
# Creating a feature of the time difference of trip start and end time.

df_grouped['od_total_time']=(df_grouped['od_end_time']-df_grouped['od_start
```

In [20]:
```
trip_dict = {

    'data' : 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',

    'source_center' : 'first',
    'source_name' : 'first',

    'destination_center' : 'last',
    'destination_name' : 'last',

    'start_scan_to_end_scan' : 'sum',
    'od_total_time' : 'sum',

    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',

    'total_segment_actual_time' : 'sum',
    'total_segment_osrm_distance' : 'sum',
    'total_segment_osrm_time' : 'sum',
    }
```

In [21]:
```
trip=df_grouped.groupby('trip_uuid',as_index=False).agg(trip_dict)
```

In [22]:
```
trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14787 non-null  category
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  category
 4   trip_uuid                       14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  od_total_time                   14787 non-null  float64
 11  actual_distance_to_destination  14787 non-null  float64
 12  actual_time                     14787 non-null  float64
 13  osrm_time                       14787 non-null  float64
 14  osrm_distance                   14787 non-null  float64
 15  total_segment_actual_time       14787 non-null  float64
 16  total_segment_osrm_distance     14787 non-null  float64
 17  total_segment_osrm_time         14787 non-null  float64
dtypes: category(2), datetime64[ns](1), float64(9), object(6)
memory usage: 1.8+ MB
```

We can observe that original data with more than 1.45 lakh rows are reduced to less than 15k rows.

In [23]:
```
trip.head()
```

Out[23]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sour
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND2(
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND56
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND0(
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND4(
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND5(

Splitting source_name and destination_name to create new features

In [24]:
```
trip['destination_name'] = trip['destination_name'].str.lower() #lowering a
trip['source_name'] = trip['source_name'].str.lower()
```

In [25]:
```
def place_to_state(x):
    # transforming  "kanpur_central_h_6 (uttar pradesh)" into "uttar prades
    state = x.split('(')[1]

    return state[:-1] #removing ')' from ending
```

In [26]:
```
def place_to_city(x):
     # transforming  "kanpur_central_h_6 (uttar pradesh)" into "kanpur_cent
    city=x.split(' (')[0]
    # transforming "kanpur_central_h_6" to "kanpur"
    city=city.split('_')[0]

    #Now dealing with edge cases

    if city == 'pnq vadgaon sheri dpc':
      return 'vadgaonsheri'

    # ['PNQ Pashan DPC', 'Bhopal MP Nagar','PNQ Rahatani DPC',
    #  'HBR Layout PC', 'Pune Balaji Nagar', 'Mumbai Antop Hill']

    if city in ['pnq pashan dpc','pnq rahatani dpc', 'pune balaji nagar']:
        return 'pune'

    if city == 'hbr layout pc' : return 'bengaluru'
    if city == 'bhopal mp nagar' : return 'bhopal'
    if city == 'mumbai antop hill' : return 'mumbai'

    return city

def place_to_city_place(x):

    # We will remove state from the string
    x = x.split(' (')[0]

    len_ = len(x.split('_'))

    if len_ >= 3:
        return x.split('_')[1]

    # Small cities have same city and also a place name
    if len_ == 2:
        return x.split('_')[0]


    # Now we need to deal with edge cases or imporper name convention

    #if len(x.split(' ')) == 2:
    #

    return x.split(' ')[0]
```

In [27]:
```
def place_to_code(x):
    # We will remove state from the string
    x = x.split(' (')[0]

    # selecting the place code from the string with the help of its length.
    if len(x.split('_')) >= 3 :
        return x.split('_')[-1]

    return 'none'
```

In [28]:
```
trip['source_state'] = trip['source_name'].apply(place_to_state)
trip['source_city']  = trip['source_name'].apply(place_to_city)
trip['source_place'] = trip['source_name'].apply(place_to_city_place)
trip['source_code']  = trip['source_name'].apply(place_to_code)
```

In [29]:
```
trip['destination_state'] = trip['destination_name'].apply(place_to_state)
trip['destination_city']  = trip['destination_name'].apply(place_to_city)
trip['destination_place'] = trip['destination_name'].apply(place_to_city_pl
trip['destination_code']  = trip['destination_name'].apply(place_to_code)
```

Splitting trip_creation_date to create new features

In [30]:
```
trip['trip_year'] = trip['trip_creation_time'].dt.year
trip['trip_month'] = trip['trip_creation_time'].dt.month
trip['trip_hour'] = trip['trip_creation_time'].dt.hour
trip['trip_day'] = trip['trip_creation_time'].dt.day
trip['trip_week'] = trip['trip_creation_time'].dt.isocalendar().week
trip['trip_dayofweek'] = trip['trip_creation_time'].dt.dayofweek
```

In [31]:
```
trip[['trip_year', 'trip_month', 'trip_hour', 'trip_day', 'trip_week', 'tri
```

Out[31]:

	trip_year	trip_month	trip_hour	trip_day	trip_week	trip_dayofweek
0	2018	9	0	12	37	2
1	2018	9	0	12	37	2
2	2018	9	0	12	37	2
3	2018	9	0	12	37	2
4	2018	9	0	12	37	2
...	...	...	...	...	...	...
14782	2018	10	23	3	40	2
14783	2018	10	23	3	40	2
14784	2018	10	23	3	40	2
14785	2018	10	23	3	40	2
14786	2018	10	23	3	40	2

14787 rows × 6 columns

EDA on the final dataset

In [32]:
```
trip.head(5)
```

Out[32]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sour
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND2(
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND56
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND0(
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND4(
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND58

5 rows × 32 columns

In [33]:
```
trip.shape
```

Out[33]:
```
(14787, 32)
```

In [34]:
```
trip.info()
```

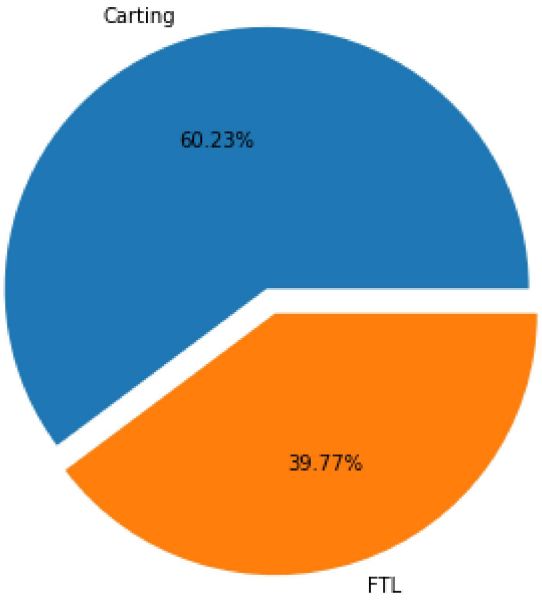
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 32 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14787 non-null  category
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  category
 4   trip_uuid                       14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  od_total_time                   14787 non-null  float64
 11  actual_distance_to_destination  14787 non-null  float64
 12  actual_time                     14787 non-null  float64
 13  osrm_time                       14787 non-null  float64
 14  osrm_distance                   14787 non-null  float64
 15  total_segment_actual_time       14787 non-null  float64
 16  total_segment_osrm_distance     14787 non-null  float64
 17  total_segment_osrm_time         14787 non-null  float64
 18  source_state                    14787 non-null  object
 19  source_city                     14787 non-null  object
 20  source_place                    14787 non-null  object
 21  source_code                     14787 non-null  object
 22  destination_state               14787 non-null  object
 23  destination_city                14787 non-null  object
 24  destination_place               14787 non-null  object
 25  destination_code                14787 non-null  object
 26  trip_year                       14787 non-null  int32
 27  trip_month                      14787 non-null  int32
 28  trip_hour                       14787 non-null  int32
 29  trip_day                        14787 non-null  int32
 30  trip_week                       14787 non-null  UInt32
 31  trip_dayofweek                  14787 non-null  int32
dtypes: UInt32(1), category(2), datetime64[ns](1), float64(9), int32(5), o
bject(14)
memory usage: 3.1+ MB
```

In [35]:
```
trip['trip_creation_time'].min(),trip['trip_creation_time'].max()
```

Out[35]:
```
(Timestamp('2018-09-12 00:00:16.535741'),
 Timestamp('2018-10-03 23:59:42.701692'))
```

The data given to us is of a month duration.

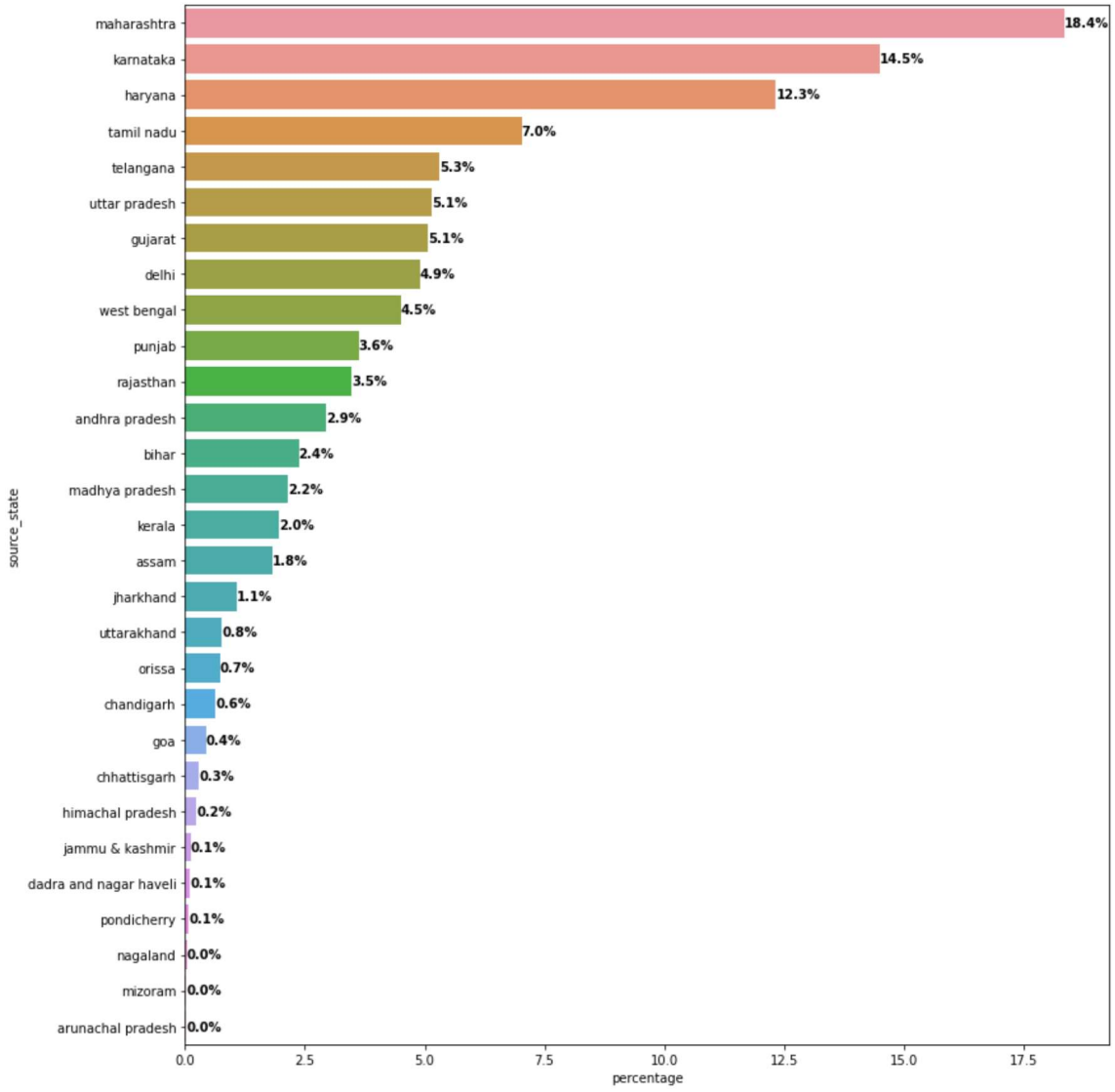
In [36]:
```
trip_route_type=trip.groupby('route_type')['trip_uuid'].nunique().reset_ind
trip_route_type['percentage']= trip_route_type['trip_uuid']/(trip_route_typ

plt.figure(figsize=(8,6))
plt.pie(x=trip_route_type['percentage'],labels=trip_route_type['route_type'
plt.show()
```



More than 60% of the trips are for carting.

In [37]:
```
trip_source_state= trip.groupby(['source_state'])['trip_uuid'].nunique().re
trip_source_state['percentage'] = trip_source_state['trip_uuid']/(trip_sour
trip_source_state= trip_source_state.sort_values(by='trip_uuid', ascending=
```

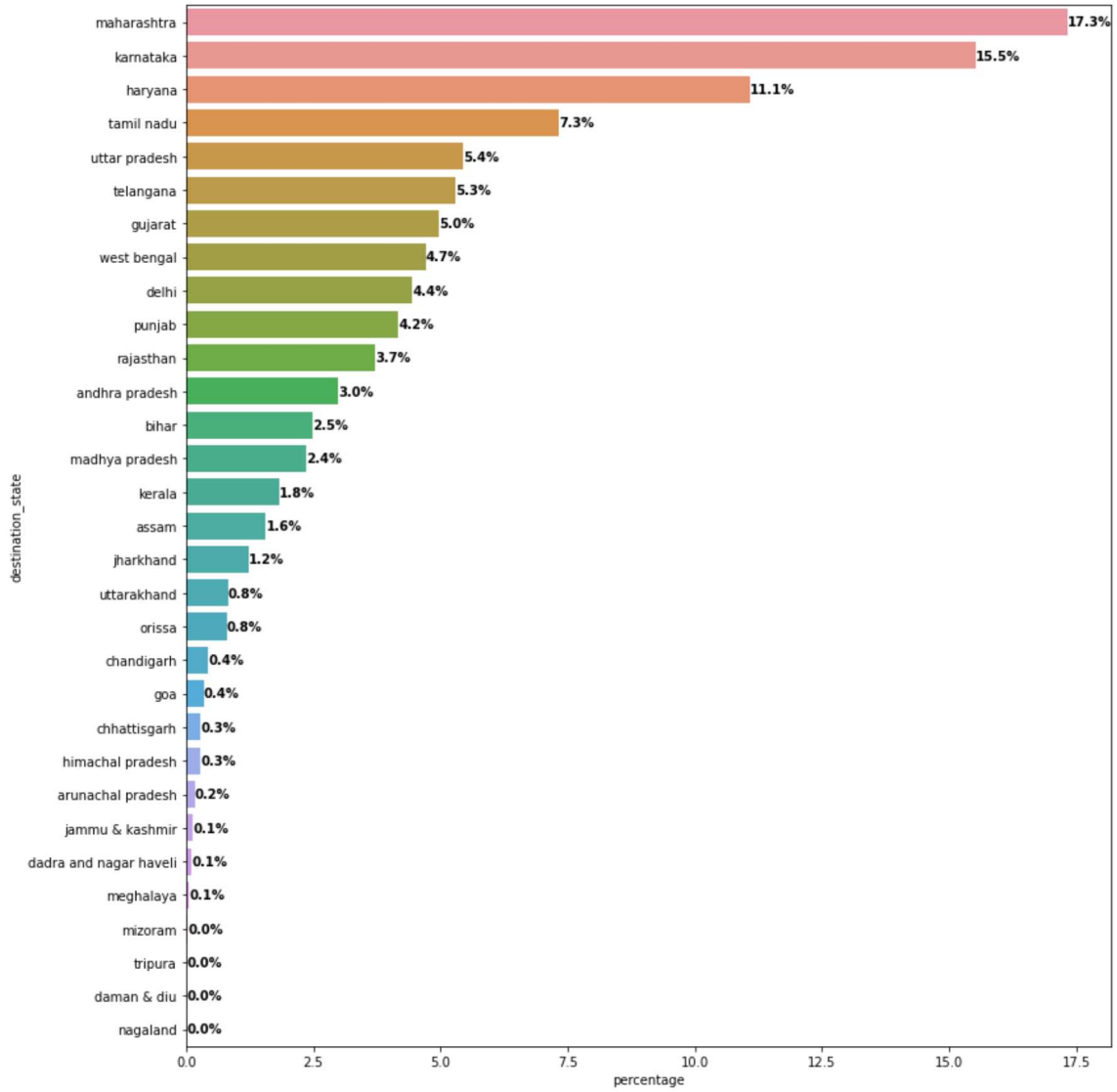
In [38]:
```
plt.figure(figsize=(13,15))
ax=sns.barplot(y= trip_source_state['source_state'] ,x= trip_source_state['
# Annotating each bar with its percentage value
for index, value in enumerate(trip_source_state['percentage']):
    ax.text(value, index, f'{value:.1f}%', ha='left', va='center', color='b
plt.show()
```



Most orders are sourced from the states Maharashtra, Karnataka, Haryana, Tamil Nadu, Telangana.

In [39]:
```
trip_destination_state= trip.groupby(['destination_state'])['trip_uuid'].nu
trip_destination_state['percentage'] = trip_destination_state['trip_uuid']/
trip_destination_state= trip_destination_state.sort_values(by='trip_uuid',
```

In [40]:
```
plt.figure(figsize=(13,15))
ax=sns.barplot(y= trip_destination_state['destination_state'] ,x= trip_dest
# Annotating each bar with its percentage value
for index, value in enumerate(trip_destination_state['percentage']):
    ax.text(value, index, f'{value:.1f}%', ha='left', va='center', color='b
plt.show()
```



Most orders are destined to Maharashtra, Karnataka followed by Haryana, Tamil Nadu, Telangana.

Hypothesis testing and Visual analysis

Comparing 'od_total_time' and 'start_scan_to_end_scan'

In [41]:
```
trip[['od_total_time','start_scan_to_end_scan']].describe()
```

Out[41]:

	od_total_time	start_scan_to_end_scan
count	14787.000000	14787.000000
mean	530.313517	529.429025
std	658.415490	658.254936
min	23.461468	23.000000
25%	149.698496	149.000000
50%	279.710750	279.000000
75%	633.537697	632.000000
max	7898.551955	7898.000000

From the above initial comparision, total trip time and start_scan_to_end_scan seems to be very similar.

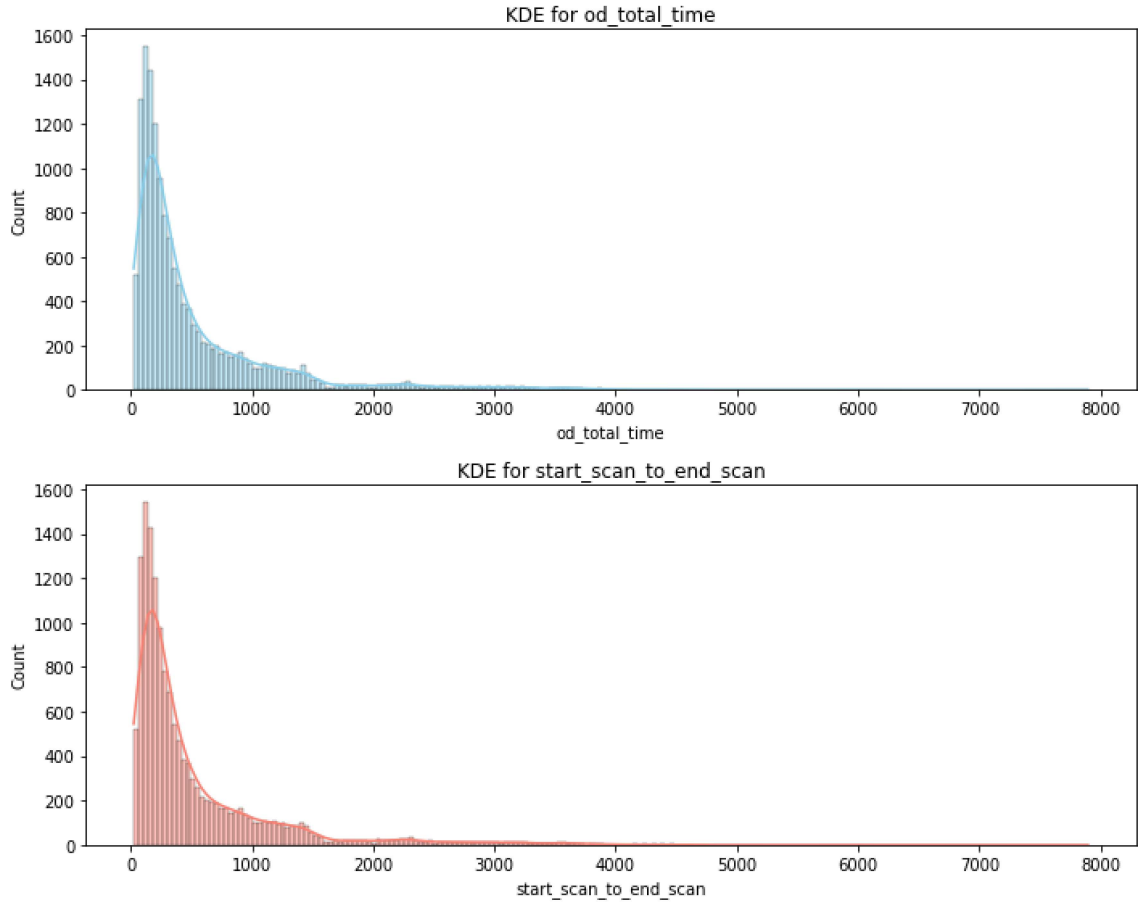
So lets do Hypothesis testing to find out whether they are indeed same or not.

Null hypothesis (H0) : od_total_time(total trip time) & start_scan_to_end_scan(expected trip time) are same.

Alternate hypothesis (Ha) : od_total_time(total trip time) & start_scan_to_end_scan(expected trip time) are different.

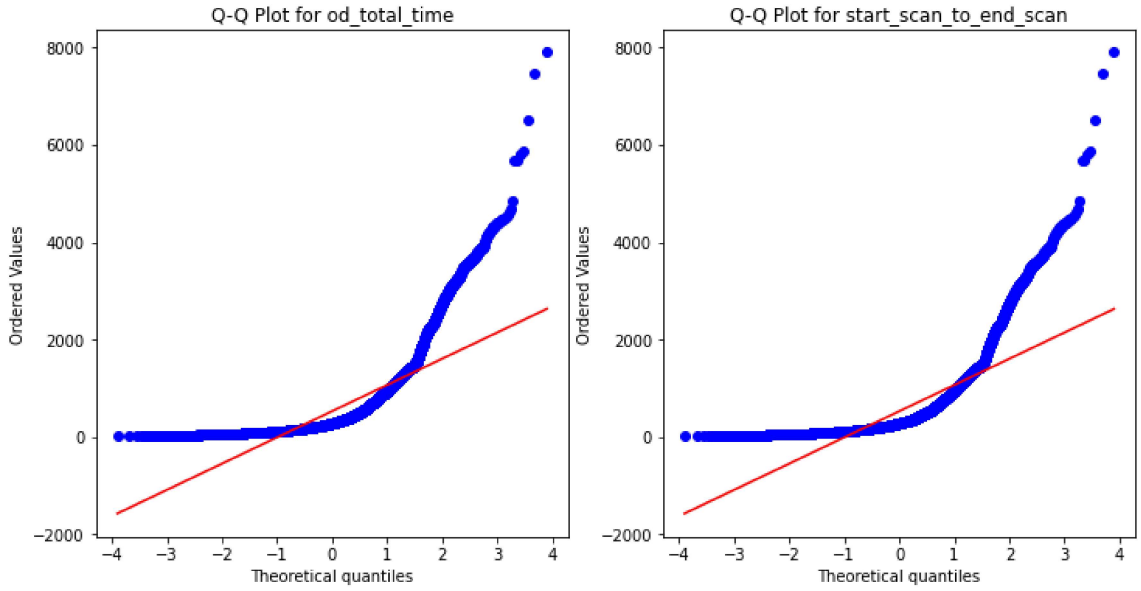
Let's set alpha as 0.05

In [42]:
```
# To check whether these two are same or different we can conduct a 2-sampl
# But for this the basic assumptions has to be satisfied.

# So let's check whether the data is following a normal distribution or not

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
sns.histplot(data=trip, x='od_total_time', kde=True , ax=axes[0], fill=True
axes[0].set_title('KDE for od_total_time')

sns.histplot(data=trip, x='start_scan_to_end_scan', kde=True, ax=axes[1], f
axes[1].set_title('KDE for start_scan_to_end_scan')

plt.tight_layout()
```



The above plots doesnt seems to be normal distribution. So let's confirm the same with qq plots.

In [43]:
```
# Let's plot q-q plot for columns 'od_total_time' and 'start_scan_to_end_sc

plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
stats.probplot(trip['od_total_time'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for od_total_time')

plt.subplot(1,2,2)
stats.probplot(trip['start_scan_to_end_scan'], dist="norm", plot=sns.mpl.py
plt.title('Q-Q Plot for start_scan_to_end_scan')
plt.show()
```



From the above Q-Q plots we can deduce that the above data is not following a normal distribution.

So, we can't conduct 'Two-sample T-test' instead we can do the Mann-Whitney U test.

In [44]:
```
test_stat,p = mannwhitneyu(trip['od_total_time'],trip['start_scan_to_end_sc
print("P-Value :", p)
if p<0.05:
    print("The samples are not similar.")
else:
    print("The samples are similar.")
```

```
P-Value : 0.7809958014596654
The samples are similar.
```

As the P-value is greater than the significance level, we can conclude that both 'od_total_time' and 'start_scan_to_end_scan' are similar.

Comparing actual_time aggregated value and OSRM time aggregated value

In [45]:
```
trip[['actual_time','osrm_time']].describe()
```

Out[45]:

	actual_time	osrm_time
count	14787.000000	14787.000000
mean	356.306012	160.990938
std	561.517936	271.459495
min	9.000000	6.000000
25%	67.000000	29.000000
50%	148.000000	60.000000
75%	367.000000	168.000000
max	6265.000000	2032.000000

From the above initial comparision 'actual_time' and 'osrm_time' seems to be very different. Actual time seems to be greater than the osrm_time.

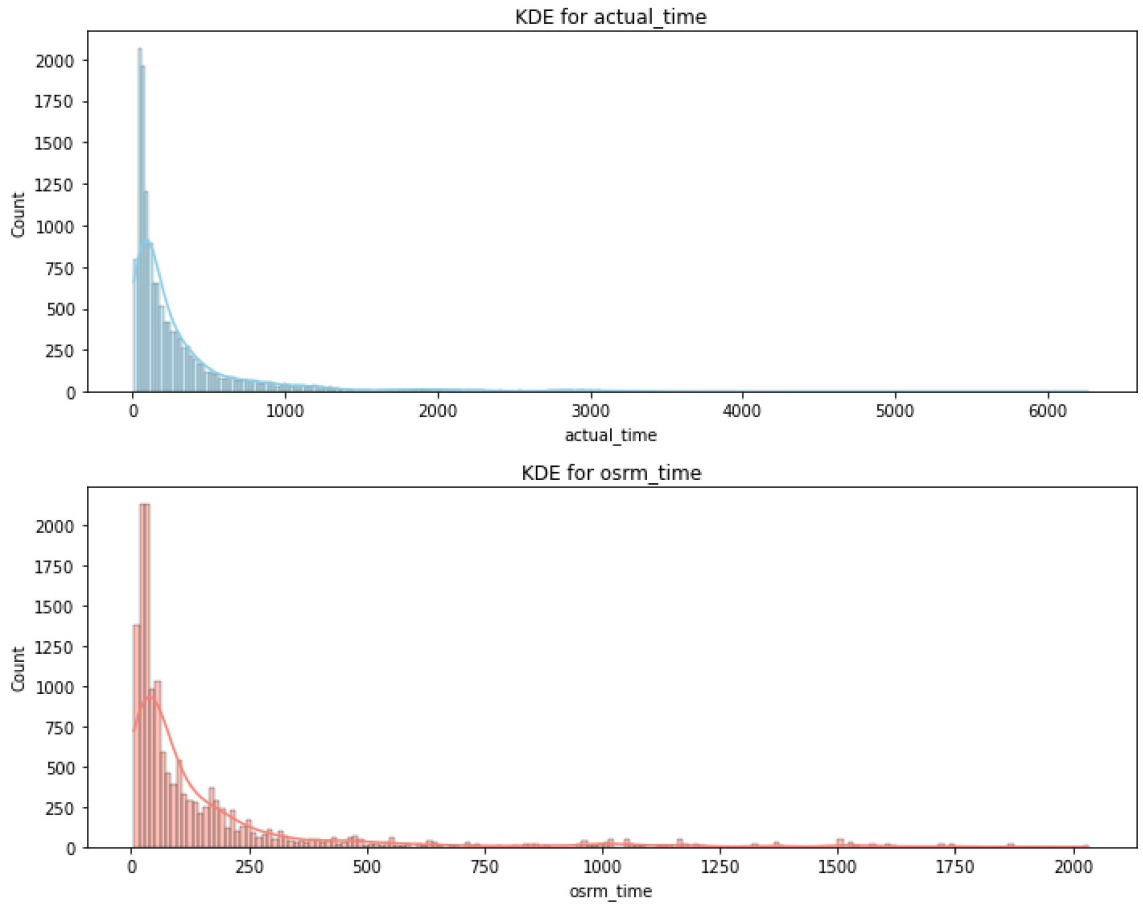
So let's do Hypothesis testing to find out whether they are same or not.

Null hypothesis (H0) : actual_time aggregated value & osrm_time aggregated value are same.

Alternate hypothesis (Ha) : actual_time aggregated value & osrm_time aggregated value are different.

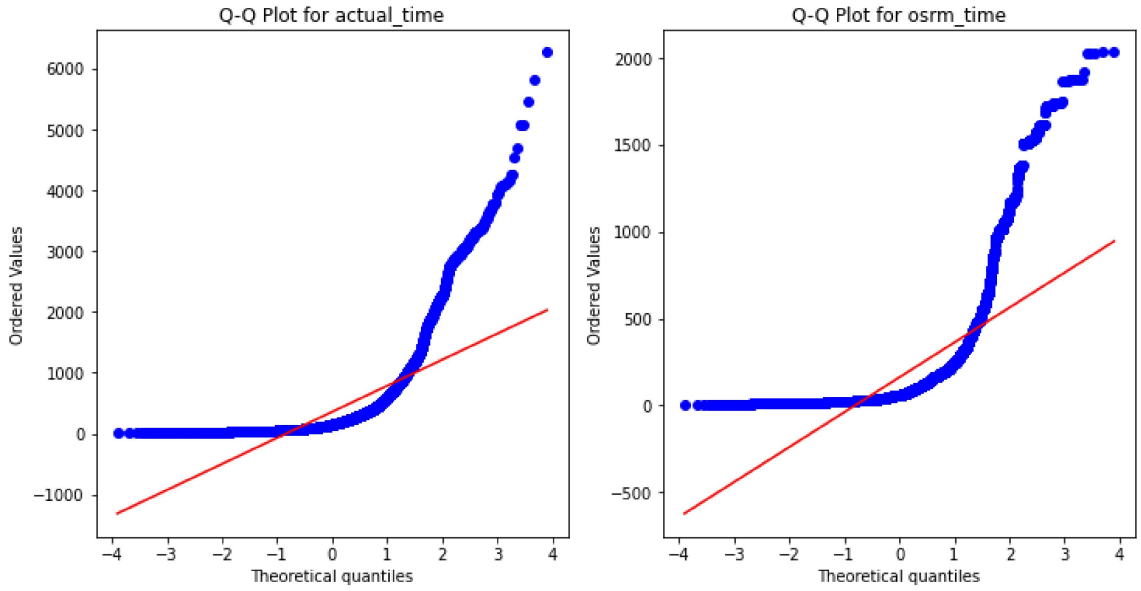
Let's set alpha as 0.05

In [46]:
```
# To check whether these two are same or different we can conduct a 2-sampl
# But for this the basic assumptions has to be satisfied.

# So let's check whether the data is following a normal distribution or not

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
sns.histplot(data=trip, x='actual_time', kde=True , ax=axes[0], fill=True,
axes[0].set_title('KDE for actual_time')

sns.histplot(data=trip, x='osrm_time', kde=True, ax=axes[1], fill=True, col
axes[1].set_title('KDE for osrm_time')

plt.tight_layout()
```



The above plots doesnt seems to be normal distribution. So let's confirm the same with qq plots.

In [47]:
```
# Let's plot q-q plot for columns 'actual_time' and 'osrm_time'

plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
stats.probplot(trip['actual_time'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for actual_time')

plt.subplot(1,2,2)
stats.probplot(trip['osrm_time'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for osrm_time')
plt.show()
```



From the above Q-Q plots we can deduce that the above data is not following a normal distribution.

So, we can't conduct 'Two-sample T-test' instead we can do the Mann-Whitney U test.

In [48]:
```
test_stat,p = mannwhitneyu(trip['actual_time'],trip['osrm_time'])
print("P-Value :", p)
if p<0.05:
    print("The samples are not similar.")
else:
    print("The samples are similar.")
```

```
P-Value : 0.0
The samples are not similar.
```

In [49]:
```
# As the samples are not similar, lets perform a 'one-tailed' Mann-Whitney
# to find whether 1 data is greater or less than the other dataset.

test_stat,p = mannwhitneyu(trip['actual_time'],trip['osrm_time'], alternati
print("P-Value :", p)
if p<0.05:
    print("actual_time is greater than osrm_time")
else:
    print("The samples are similar.")
```

```
P-Value : 0.0
actual_time is greater than osrm_time
```

As the P-value is less than the significance level, we can conclude that 'actual_time' is greater than the 'osrm_time'.

Comparing actual_time aggregated value and segment_actual_time aggregated value

In [50]:
```
trip[['actual_time','total_segment_actual_time']].describe()
```

Out[50]:

	actual_time	total_segment_actual_time
count	14787.000000	14787.000000
mean	356.306012	353.059174
std	561.517936	556.365911
min	9.000000	9.000000
25%	67.000000	66.000000
50%	148.000000	147.000000
75%	367.000000	364.000000
max	6265.000000	6230.000000

From the above initial comparision 'actual_time' and 'total_segment_actual_time' seems to be similar.

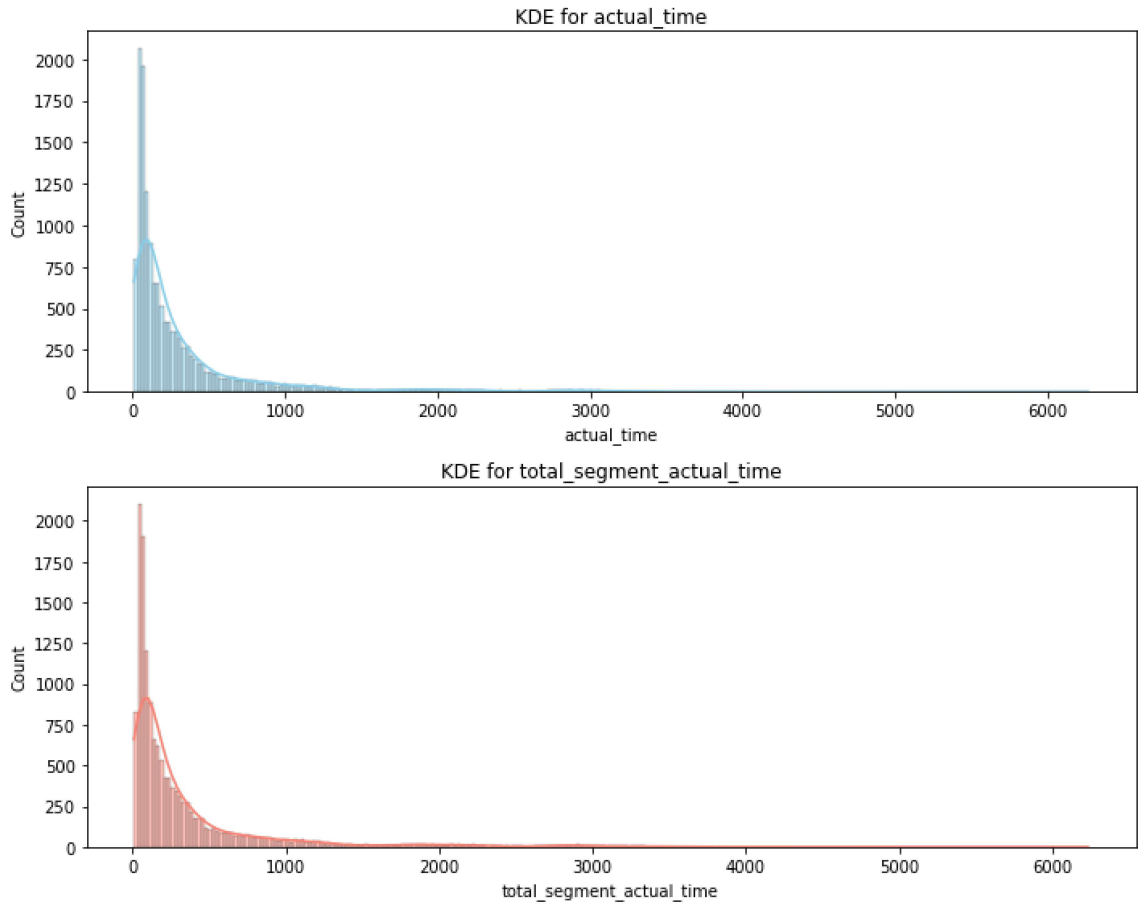
So let's do Hypothesis testing to find out whether they are indeed same or not.

Null hypothesis (H0) : actual_time aggregated value & total_segment_actual_time are same.

Alternate hypothesis (Ha) : actual_time aggregated value & total_segment_actual_time are different.

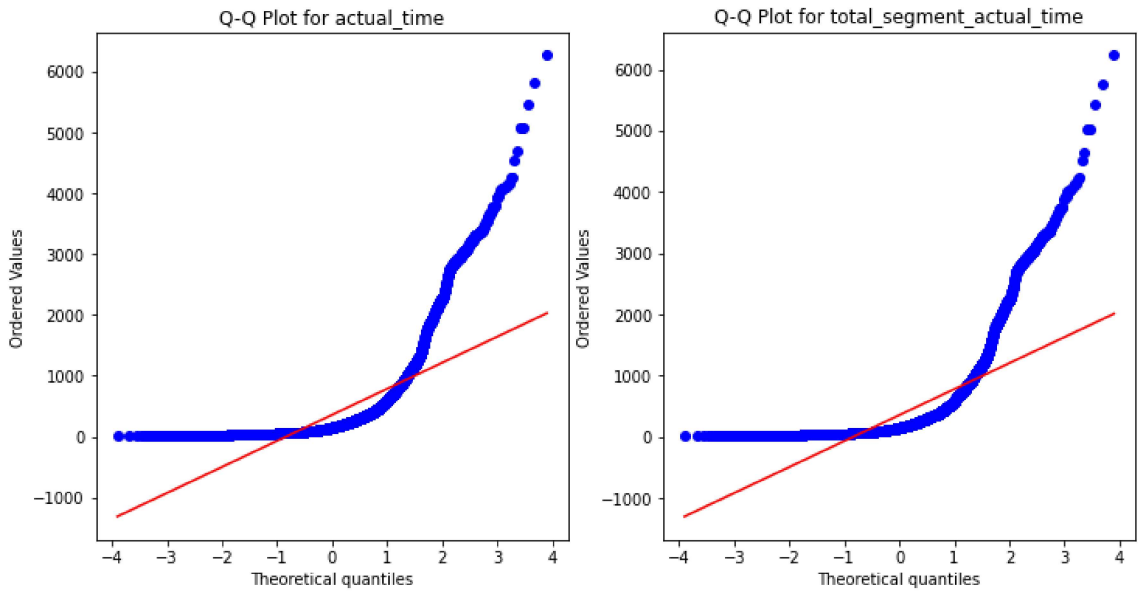
Let's set alpha as 0.05

In [51]:
```
# To check whether these two are same or different we can conduct a 2-sampl
# But for this the basic assumptions has to be satisfied.

# So let's check whether the data is following a normal distribution or not

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
sns.histplot(data=trip, x='actual_time', kde=True , ax=axes[0], fill=True,
axes[0].set_title('KDE for actual_time')

sns.histplot(data=trip, x='total_segment_actual_time', kde=True, ax=axes[1]
axes[1].set_title('KDE for total_segment_actual_time')

plt.tight_layout()
```



The above plots doesnt seems to be normal distribution. So let's confirm the same with qq plots.

In [52]:
```
# Let's plot q-q plot for columns 'actual_time' and 'total_segment_actual_t

plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
stats.probplot(trip['actual_time'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for actual_time')

plt.subplot(1,2,2)
stats.probplot(trip['total_segment_actual_time'], dist="norm", plot=sns.mpl
plt.title('Q-Q Plot for total_segment_actual_time')
plt.show()
```



From the above Q-Q plots we can deduce that the above data is not following a normal distribution.

So, we can't conduct 'Two-sample T-test' instead we can do the Mann-Whitney U test.

In [53]:
```
test_stat,p = mannwhitneyu(trip['actual_time'],trip['total_segment_actual_t
print("P-Value :", p)
if p<0.05:
    print("The samples are not similar.")
else:
    print("The samples are similar.")
```

```
P-Value : 0.41578601931625214
The samples are similar.
```

As the P-value is greater than the significance level, we can conclude that both 'actual_time' and 'total_segment_actual_time' are similar.

Comparing osrm_distance aggregated value and segment_osrm_distance aggregated value

In [54]:
```
trip[['osrm_distance','total_segment_osrm_distance']].describe()
```

Out[54]:

	osrm_distance	total_segment_osrm_distance
count	14787.000000	14787.000000
mean	203.887411	222.705466
std	370.565564	416.846279
min	9.072900	9.072900
25%	30.756900	32.578850
50%	65.302800	69.784200
75%	206.644200	216.560600
max	2840.081000	3523.632400

From the above initial comparision 'osrm_distance' and 'total_segment_osrm_distance' seems to be similar.

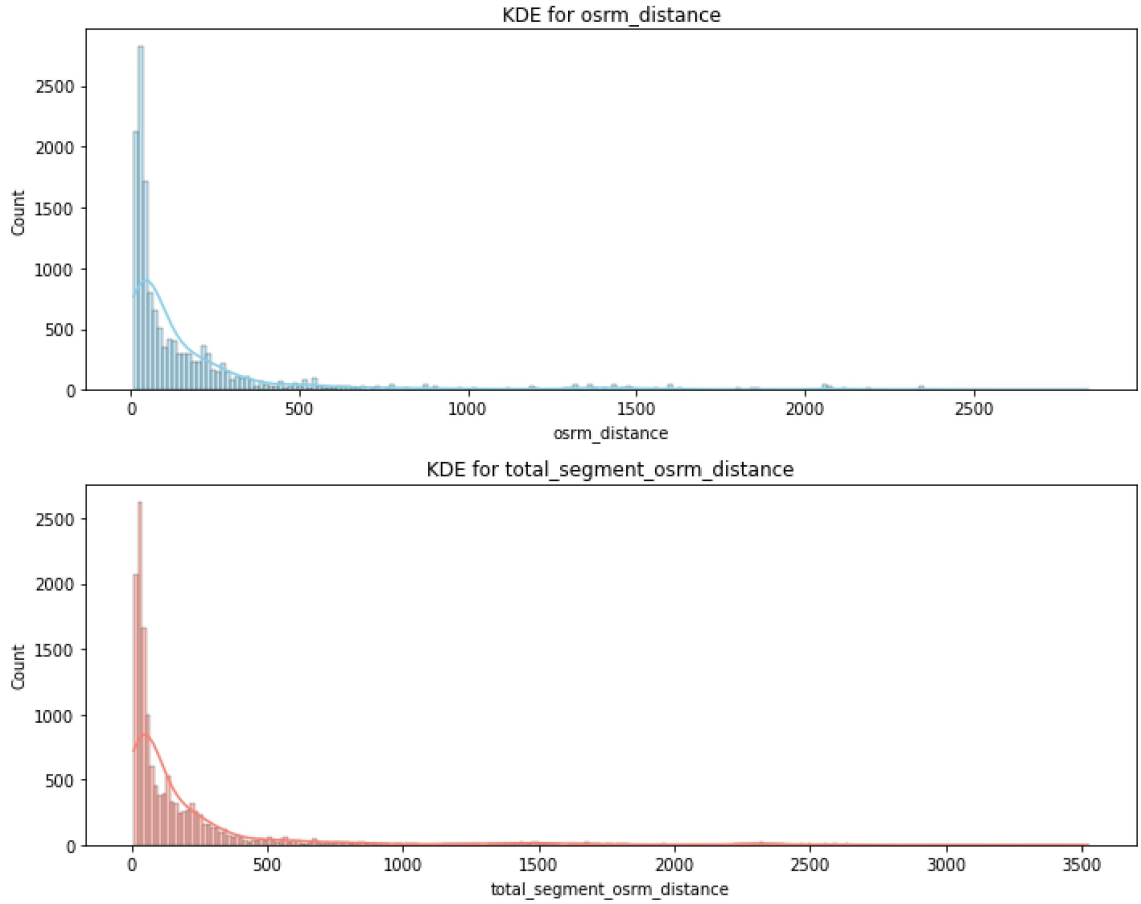
So let's do Hypothesis testing to find out whether they are indeed same or not.

Null hypothesis (H0) : osrm_distance aggregated value & total_segment_osrm_distance are same.

Alternate hypothesis (Ha) : osrm_distance aggregated value & total_segment_osrm_distance are different.

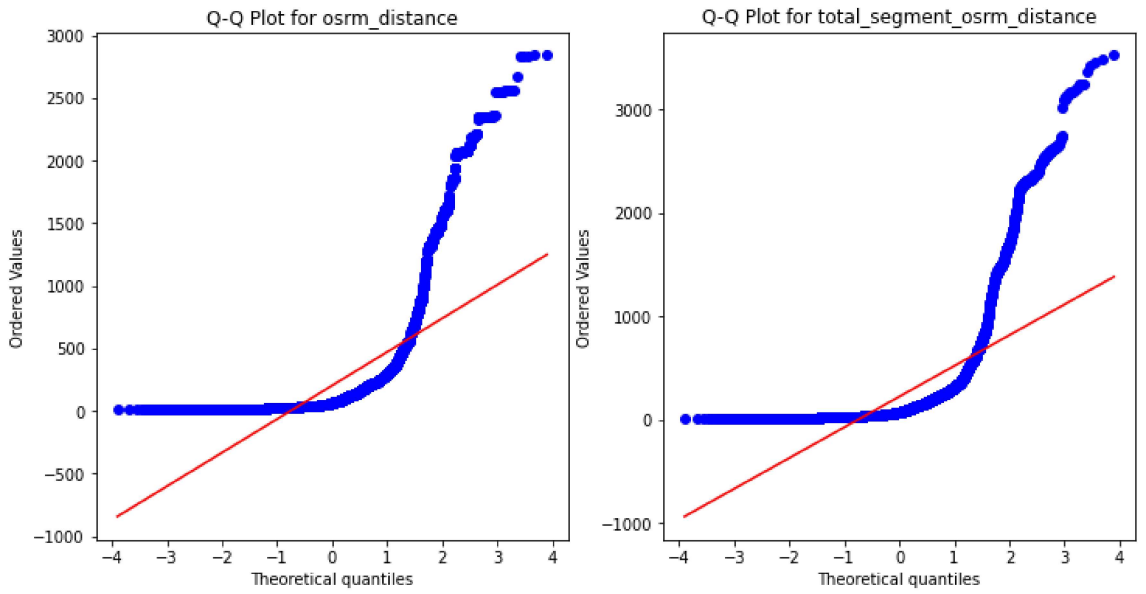
Let's set alpha as 0.05

In [55]:
```
# To check whether these two are same or different we can conduct a 2-sampl
# But for this the basic assumptions has to be satisfied.

# So let's check whether the data is following a normal distribution or not

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
sns.histplot(data=trip, x='osrm_distance', kde=True , ax=axes[0], fill=True
axes[0].set_title('KDE for osrm_distance')

sns.histplot(data=trip, x='total_segment_osrm_distance', kde=True, ax=axes[
axes[1].set_title('KDE for total_segment_osrm_distance')

plt.tight_layout()
```



The above plots doesnt seems to be normal distribution. So let's confirm the same with qq plots.

In [56]:
```
# Let's plot q-q plot for columns 'osrm_distance' and 'total_segment_osrm_d

plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
stats.probplot(trip['osrm_distance'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for osrm_distance')

plt.subplot(1,2,2)
stats.probplot(trip['total_segment_osrm_distance'], dist="norm", plot=sns.m
plt.title('Q-Q Plot for total_segment_osrm_distance')
plt.show()
```



From the above Q-Q plots we can deduce that the above data is not following a normal distribution.

So, we can't conduct 'Two-sample T-test' instead we can do the Mann-Whitney U test.

In [57]:
```
test_stat,p = mannwhitneyu(trip['osrm_distance'],trip['total_segment_osrm_d
print("P-Value :", p)
if p<0.05:
    print("The samples are not similar.")
else:
    print("The samples are similar.")
```

```
P-Value : 1.000108765909207e-06
The samples are not similar.
```

In [58]:
```
# As the samples are not similar, lets perform a 'one-tailed' Mann-Whitney
# to find whether 1 data is greater or less than the other dataset.

test_stat,p = mannwhitneyu(trip['osrm_distance'],trip['total_segment_osrm_d
print("P-Value :", p)
if p<0.05:
    print("osrm_distance is less than total_segment_osrm_distance")
else:
    print("The samples are similar.")
```

```
P-Value : 5.000543829546035e-07
osrm_distance is less than total_segment_osrm_distance
```

As the P-value is less than the significance level, we can conclude that 'osrm_distance' is less than the 'total_segment_osrm_distance'.

Comparing osrm time aggregated value and segment osrm time aggregated value

In [59]:
```
trip[['osrm_time','total_segment_osrm_time']].describe()
```

Out[59]:

	osrm_time	total_segment_osrm_time
count	14787.000000	14787.000000
mean	160.990938	180.511598
std	271.459495	314.679279
min	6.000000	6.000000
25%	29.000000	30.000000
50%	60.000000	65.000000
75%	168.000000	184.000000
max	2032.000000	2564.000000

From the above initial comparision 'osrm_time' and 'total_segment_osrm_time' seems to be a bit different.

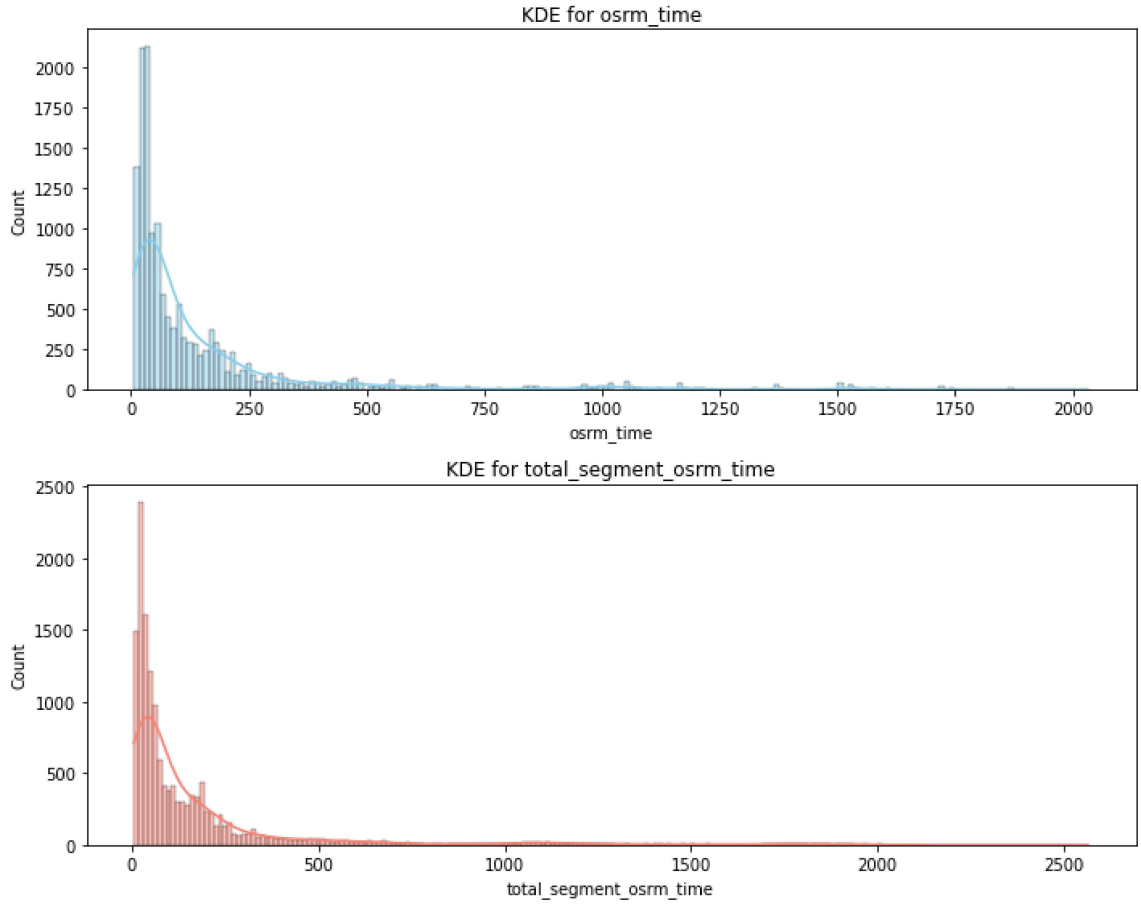
So let's do Hypothesis testing to find out whether they are indeed same or not.

Null hypothesis (H0) : osrm_time aggregated value & total_segment_osrm_time are same.

Alternate hypothesis (Ha) : osrm_time aggregated value & total_segment_osrm_time are different.

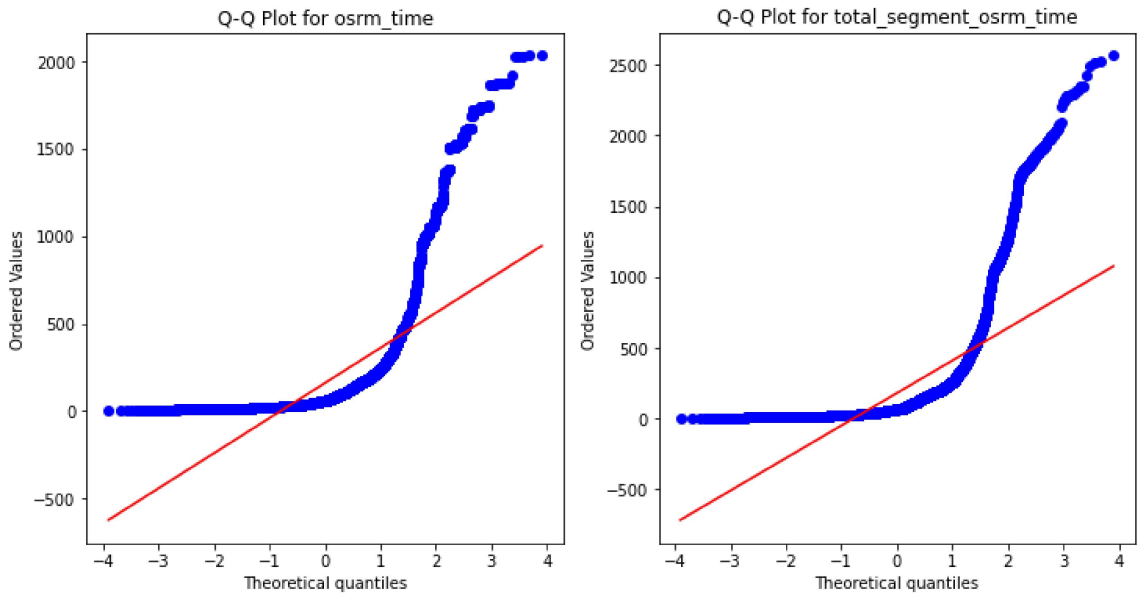
Let's set alpha as 0.05

In [60]:
```
# To check whether these two are same or different we can conduct a 2-sampl
# But for this the basic assumptions has to be satisfied.

# So let's check whether the data is following a normal distribution or not

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
sns.histplot(data=trip, x='osrm_time', kde=True , ax=axes[0], fill=True, co
axes[0].set_title('KDE for osrm_time')

sns.histplot(data=trip, x='total_segment_osrm_time', kde=True, ax=axes[1],
axes[1].set_title('KDE for total_segment_osrm_time')

plt.tight_layout()
```



The above plots doesnt seems to be normal distribution. So let's confirm the same with qq plots.

In [61]:
```
# Let's plot q-q plot for columns 'osrm_time' and 'total_segment_osrm_time'

plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
stats.probplot(trip['osrm_time'], dist="norm", plot=sns.mpl.pyplot)
plt.title('Q-Q Plot for osrm_time')

plt.subplot(1,2,2)
stats.probplot(trip['total_segment_osrm_time'], dist="norm", plot=sns.mpl.p
plt.title('Q-Q Plot for total_segment_osrm_time')
plt.show()
```



From the above Q-Q plots we can deduce that the above data is not following a normal distribution.

So, we can't conduct 'Two-sample T-test' instead we can do the Mann-Whitney U test.

In [62]:
```
test_stat,p = mannwhitneyu(trip['osrm_time'],trip['total_segment_osrm_time'
print("P-Value :", p)
if p<0.05:
    print("The samples are not similar.")
else:
    print("The samples are similar.")
```

```
P-Value : 2.4893531591323577e-08
The samples are not similar.
```

In [63]:
```
# As the samples are not similar, lets perform a 'one-tailed' Mann-Whitney
# to find whether 1 data is greater or less than the other dataset.

test_stat,p = mannwhitneyu(trip['osrm_time'],trip['total_segment_osrm_time'
print("P-Value :", p)
if p<0.05:
    print("osrm_time is less than total_segment_osrm_time")
else:
    print("The samples are similar.")
```

```
P-Value : 1.2446765795661789e-08
osrm_time is less than total_segment_osrm_time
```

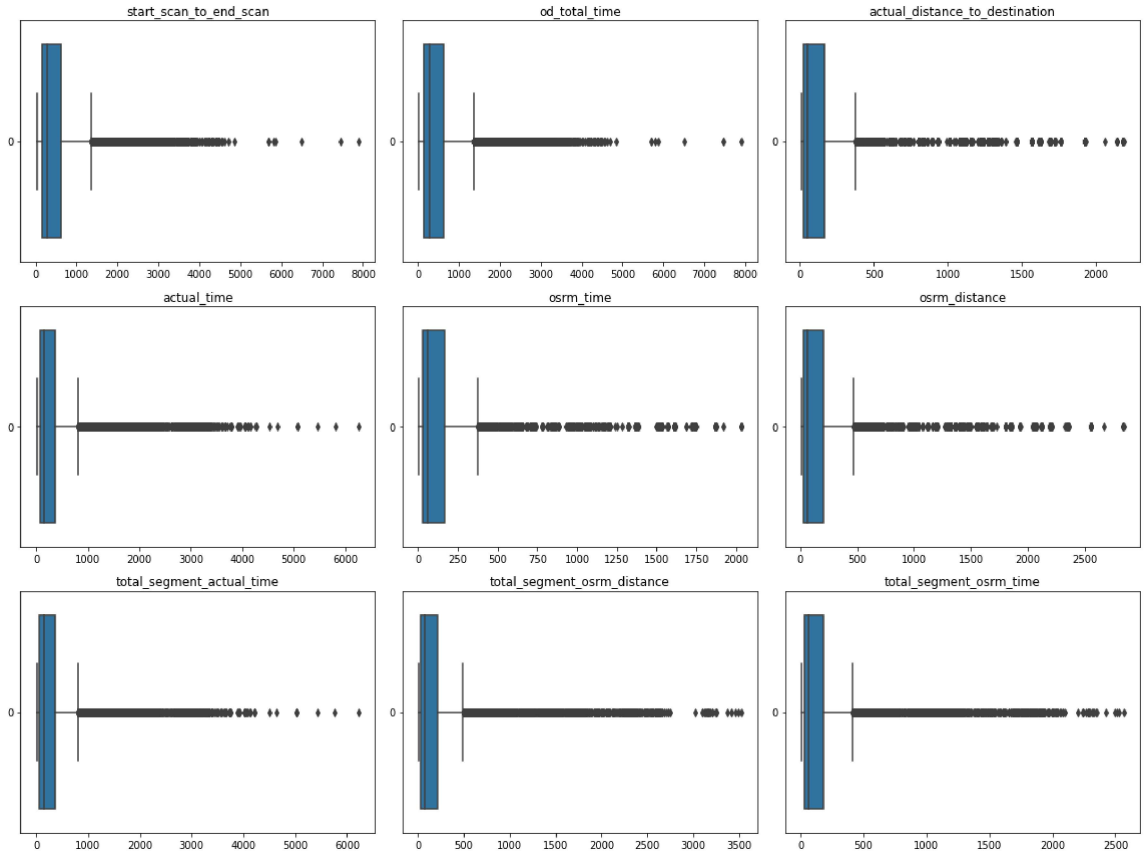
As the P-value is less than the significance level, we can conclude that 'osrm_time' is less than the 'total_segment_osrm_time'.

Finding outliers in all the numerical data columns.

In [64]:
```
# Let's create a list of all the column names with numerical data.

numerical_col=trip.dtypes[(trip.dtypes== 'float64')].index.to_list()
numerical_col
```

Out[64]:
```
['start_scan_to_end_scan',
 'od_total_time',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'total_segment_actual_time',
 'total_segment_osrm_distance',
 'total_segment_osrm_time']
```

In [65]:
```
plt.figure(figsize=(16, 12))
for i in range(len(numerical_col)):
    plt.subplot(3,3,i+1)
    sns.boxplot(trip[numerical_col[i]],orient='h')
    plt.title(f"{numerical_col[i]}")
plt.tight_layout()
```



It can be observed that there are outliers in all the columns.

Handling outliers using IQR method

In [66]:
```
# Calculating IQR

Q1 = trip[numerical_col].quantile(0.25)
Q3 = trip[numerical_col].quantile(0.75)

IQR = Q3 - Q1
```

In [67]:
```
trip = trip[~((trip[numerical_col] < (Q1 - 1.5 * IQR)) | (trip[numerical_co
trip = trip.reset_index(drop=True)
```

In [68]:
```
trip
```

Out[68]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	s
0	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	I
1	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	I
2	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	I
3	training	2018-09-12 00:02:34.161600	thanos::sroute:9bf03170-d0a2-4a3f-aa4d-9aaab3d...	Carting	trip-153671055416136166	I
4	training	2018-09-12 00:04:22.011653	thanos::sroute:a97698cc-846e-41a7-916b-88b1741...	Carting	trip-153671066201138152	II
...	...	...	...	...	...	
12718	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	II
12719	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	I
12720	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	I
12721	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	I
12722	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	I

12723 rows × 32 columns

One-hot encoding for categorical column

In [69]:
```
trip['route_type'].value_counts()
```

Out[69]:
```
route_type
Carting    8812
FTL        3911
Name: count, dtype: int64
```

In [71]:
```
# As there are only 2 distinct values, they can be easily encoded using map

trip['route_type'] = trip['route_type'].map({'FTL':0, 'Carting':1})
```

In [72]:
```
trip['route_type'].value_counts()
```

Out[72]:
```
route_type
1    8812
0    3911
Name: count, dtype: int64
```

Standardizing the columns using StandardScaler

In [73]:
```
from sklearn.preprocessing import StandardScaler
```

In [74]:
```
scaler = StandardScaler()
scaler.fit(trip[numerical_col])
```

Out[74]:
```
StandardScaler

StandardScaler()
```

In [75]:
```
trip[numerical_col] = scaler.transform(trip[numerical_col])
```

In [76]:
```
trip[numerical_col]
```

Out[76]:

	start_scan_to_end_scan	od_total_time	actual_distance_to_destination	actual_time	osı
0	-0.548546	-0.544839	0.012060	-0.217856	-0
1	-0.861602	-0.861856	-0.765152	-0.749015	-0
2	1.552838	1.552812	0.764988	1.034163	0
3	-0.513328	-0.510150	-0.662169	-0.736369	-0
4	-0.869428	-0.871585	-0.877197	-0.970332	-0
...	...	...	...	...	
12718	-0.247231	-0.246189	-0.201970	-0.597255	-0
12719	-1.018130	-1.017809	-0.788207	-0.989302	-0
12720	0.394533	0.395103	-0.466688	0.661086	-0
12721	0.104957	0.107436	0.865940	0.547267	1
12722	0.128436	0.130473	-0.086534	0.616823	-0

12723 rows × 9 columns

Recommendations:

- There is a significant difference in OSRM time, distance and actual values. Actual values are higher than the algorithm predicted values. Check for any discrepancies with the transporters team and check if the routing algorithm is equipped to handle these for optimum results. If not, design a better algorithm to predict time and distance more accurately.

- Most of the orders are coming from/reaching to states like Maharashtra, Karnataka, Haryana and Tamil Nadu. Enhance the existing corridors by minimizing the transportation losses to improve the penetration in these areas.

- North, South and West Zones corridors have significant traffic of orders. But, we have a smaller presence in Central, Eastern and North-Eastern zone. However it would be difficult to conclude this, by looking at just 2 months data. It is worth investigating and increasing our presence in these regions.

- From state's data, we have heavy traffic in Mahrashtra followed by Karnataka. This is a good indicator that we need to plan for resources on ground in these 2 states on priority. Especially, during festive seasons.

- Customer profiling of the customers belonging to the states Maharashtra, Karnataka, Haryana, Tamil Nadu and Uttar Pradesh has to be done to get to know why major

- The data provided to us is of only a month duration. So, request for a more broader data to find the seasonality of the orders to determine the traffic distribution across different seasons.

In [ ]: