About Delhivery

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

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

Problem statement

How can you help here?

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

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

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

In [180]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
DLRYPD = pd.read_csv(r"H:\Scaler\Dehlivery\delhivery_data.csv")
```

In [3]:
```
DLRYPD.head()
```

Out[3]:

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

5 rows × 24 columns

CHECKING THE DATA STRUCTURE

In [4]:
```
DLRYPD.shape
```

Out[4]:
```
(144867, 24)
```

The data frame has 144867 rows and 24 columns

In [5]:
```
DLRYPD.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
```

There appears to be null values in all the columns Out of the 24 columns 12 columns are of object data type 10 columns are of float datatype 1 column is of bool data type and 1 column is of int64 datatype

Converting datatype of the columns trip_creation_time,od_start_time,od_end_time,cutoff_timestamp to date time.

In [6]:
```
DLRYPDDT = ['trip_creation_time','od_start_time','od_end_time','cutoff_timestamp']

for i in DLRYPDDT:
   DLRYPD[i] = pd.to_datetime(DLRYPD[i])
```

In [7]:
```
 DLRYPD.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  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object        
 3   route_type                      144867 non-null  object        
 4   trip_uuid                       144867 non-null  object        
 5   source_center                   144867 non-null  object        
 6   source_name                     144574 non-null  object        
 7   destination_center              144867 non-null  object        
 8   destination_name                144606 non-null  object        
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64       
 12  is_cutoff                       144867 non-null  bool          
 13  cutoff_factor                   144867 non-null  int64         
 14  cutoff_timestamp                144867 non-null  datetime64[ns]
 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), datetime64[ns](4), float64(10), int64(1), object(8)
memory usage: 25.6+ MB
```

In [8]:
```
#The first trip creation time is
print(f"First Trip creation time : {DLRYPD['trip_creation_time'].min()}")
```

```
First Trip creation time : 2018-09-12 00:00:16.535741
```

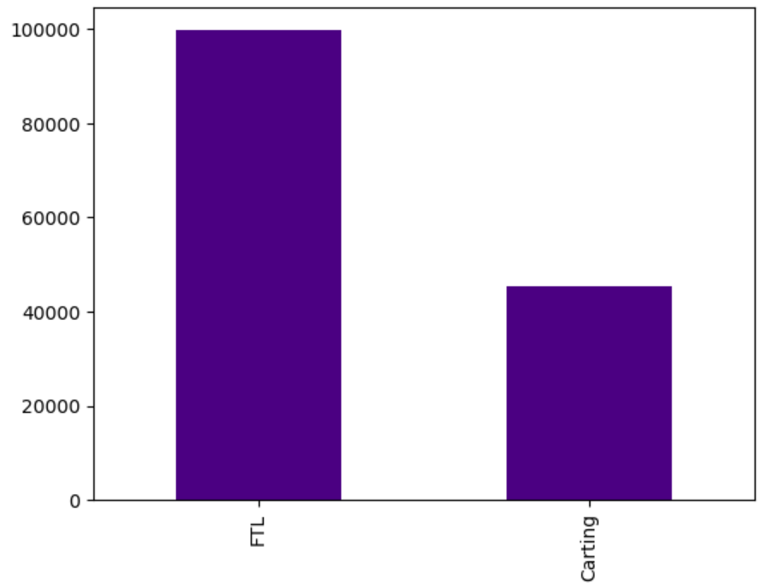
In [9]:
```
#The last trip creation time is
print(f"Last Trip creation time : {DLRYPD['trip_creation_time'].max()}")
```

```
Last Trip creation time : 2018-10-03 23:59:42.701692
```

In [10]:
```
#Checking count values of different route types
DLRYPD['route_type'].value_counts()
```

Out[10]:
```
FTL         99660
Carting     45207
Name: route_type, dtype: int64
```

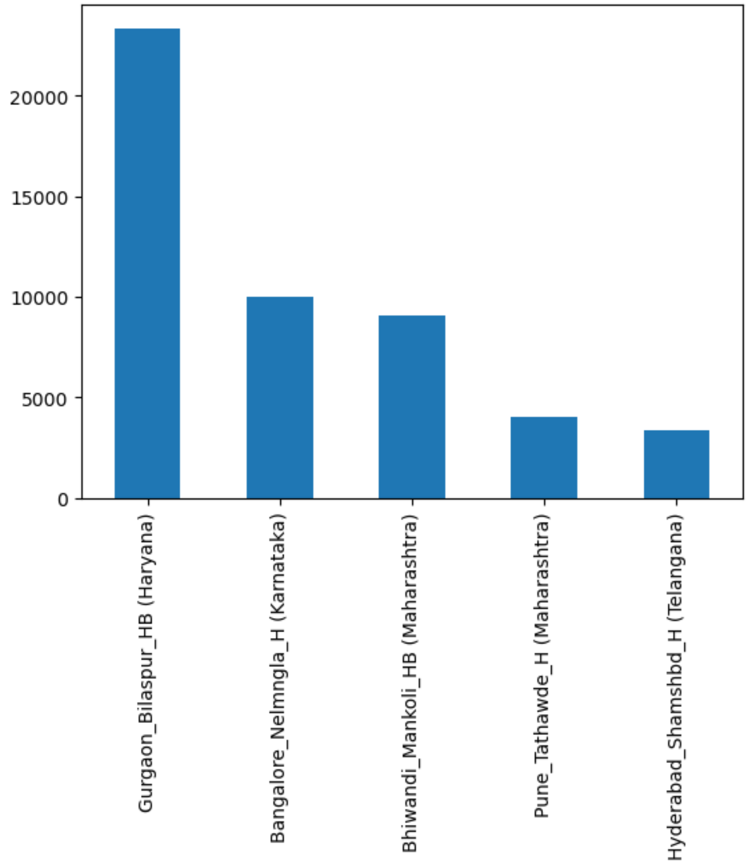
In [11]:
```
DLRYPD['route_type'].value_counts().plot(kind='bar',color='indigo')
plt.show()
```



In [12]:
```
# Checking the source name-wise data count
DLRYPD['source_name'].value_counts()
```

Out[12]:
```
Gurgaon_Bilaspur_HB (Haryana)           23347
Bangalore_Nelmngla_H (Karnataka)         9975
Bhiwandi_Mankoli_HB (Maharashtra)        9088
Pune_Tathawde_H (Maharashtra)            4061
Hyderabad_Shamshbd_H (Telangana)         3340
                                         ...
Shahjhnpur_NavdaCln_D (Uttar Pradesh)       1
Soro_UttarDPP_D (Orissa)                    1
Kayamkulam_Bhrnikvu_D (Kerala)              1
Krishnanagar_AnadiDPP_D (West Bengal)       1
Faridabad_Old (Haryana)                     1
Name: source_name, Length: 1498, dtype: int64
```

In [13]:
```
DLRYPD['source_name'].value_counts().head().plot(kind='bar')
plt.show()
```



In [14]:
```
DLRYPD['source_name'].value_counts().head()
```

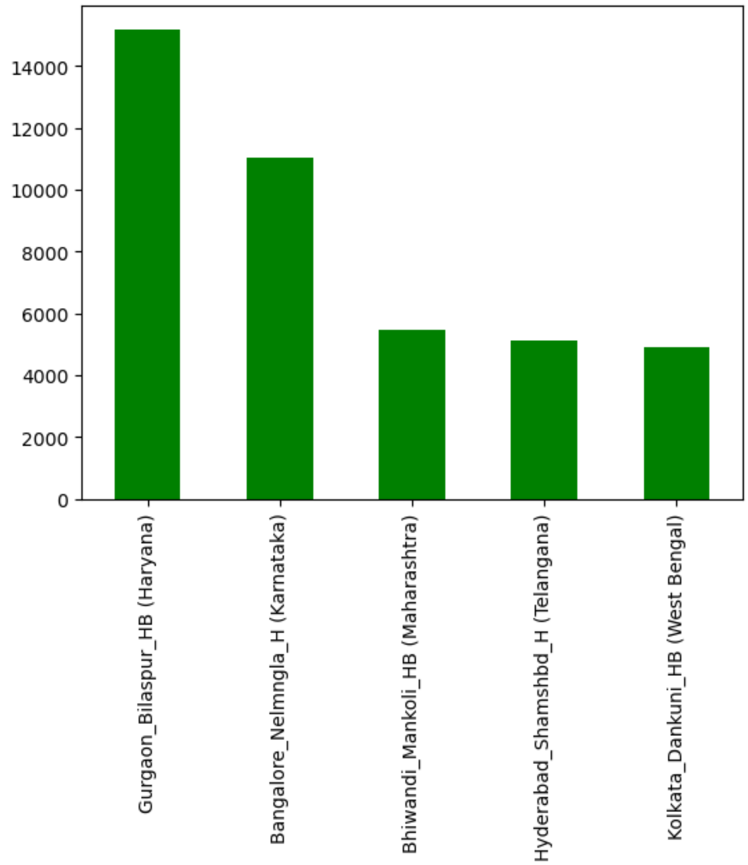
Out[14]:
```
Gurgaon_Bilaspur_HB (Haryana)        23347
Bangalore_Nelmngla_H (Karnataka)      9975
Bhiwandi_Mankoli_HB (Maharashtra)     9088
Pune_Tathawde_H (Maharashtra)         4061
Hyderabad_Shamshbd_H (Telangana)      3340
Name: source_name, dtype: int64
```

In [15]:
```
DLRYPD['destination_name'].value_counts()
```

Out[15]:
```
Gurgaon_Bilaspur_HB (Haryana)        15192
Bangalore_Nelmngla_H (Karnataka)     11019
Bhiwandi_Mankoli_HB (Maharashtra)     5492
Hyderabad_Shamshbd_H (Telangana)      5142
Kolkata_Dankuni_HB (West Bengal)      4892
                                     ...
Hyd_Trimulgherry_Dc (Telangana)          1
Vijayawada (Andhra Pradesh)              1
Baghpat_Barout_D (Uttar Pradesh)         1
Mumbai_Sanpada_CP (Maharashtra)          1
Basta_Central_DPP_1 (Orissa)             1
Name: destination_name, Length: 1468, dtype: int64
```

In [16]:
```
DLRYPD['destination_name'].value_counts().head().plot(kind='bar',color='green')
```

Out[16]: <AxesSubplot:>



In [17]:
```
DLRYPD['destination_name'].value_counts().head()
```

Out[17]:
```
Gurgaon_Bilaspur_HB (Haryana)        15192
Bangalore_Nelmngla_H (Karnataka)     11019
Bhiwandi_Mankoli_HB (Maharashtra)     5492
Hyderabad_Shamshbd_H (Telangana)      5142
Kolkata_Dankuni_HB (West Bengal)      4892
Name: destination_name, dtype: int64
```

Checking for null values

Checking the percentage of null values in each column

In [18]:
```
DLRYPD.isnull().mean() * 100
```

Out[18]:
```
data                              0.000000
trip_creation_time                0.000000
route_schedule_uuid               0.000000
route_type                        0.000000
trip_uuid                         0.000000
source_center                     0.000000
source_name                       0.202254
destination_center                0.000000
destination_name                  0.180165
od_start_time                     0.000000
od_end_time                       0.000000
start_scan_to_end_scan            0.000000
is_cutoff                         0.000000
cutoff_factor                     0.000000
cutoff_timestamp                  0.000000
actual_distance_to_destination    0.000000
actual_time                       0.000000
osrm_time                         0.000000
osrm_distance                     0.000000
factor                            0.000000
segment_actual_time               0.000000
segment_osrm_time                 0.000000
segment_osrm_distance             0.000000
segment_factor                    0.000000
dtype: float64
```

We can see that null values are present in the columns source_name and destination_name . The percentage of null values is small hence the null values are being dropped

In [19]:
```
DLRYPD = DLRYPD.dropna(how = 'any').reset_index(drop = True)
```

Checking for null values again

In [20]:
```
DLRYPD.isnull().mean() * 100
```

Out[20]:
```
data                              0.0
trip_creation_time                0.0
route_schedule_uuid               0.0
route_type                        0.0
trip_uuid                         0.0
source_center                     0.0
source_name                       0.0
destination_center                0.0
destination_name                  0.0
od_start_time                     0.0
od_end_time                       0.0
start_scan_to_end_scan            0.0
is_cutoff                         0.0
cutoff_factor                     0.0
cutoff_timestamp                  0.0
actual_distance_to_destination    0.0
actual_time                       0.0
osrm_time                         0.0
osrm_distance                     0.0
factor                            0.0
segment_actual_time               0.0
segment_osrm_time                 0.0
segment_osrm_distance             0.0
segment_factor                    0.0
dtype: float64
```

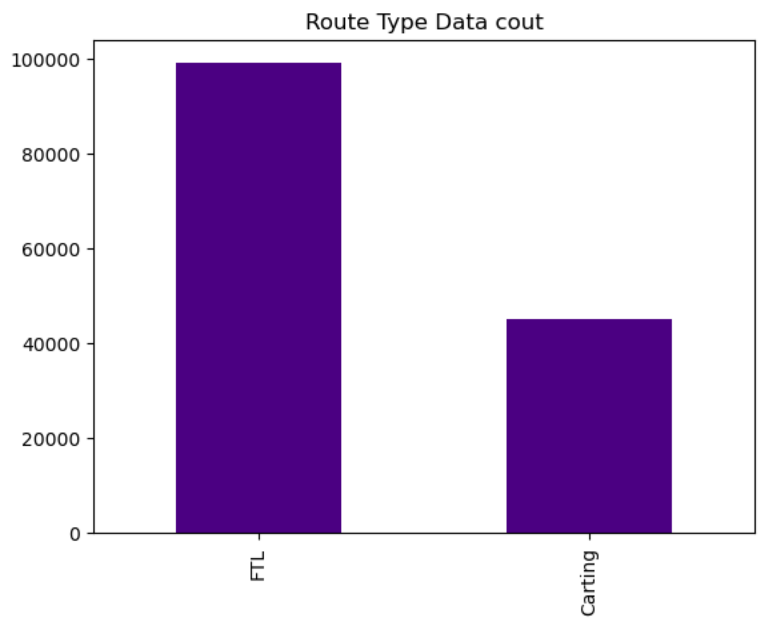
Performing Univariate analysis

In [21]:
```
DLRYPD.head()
```

Out[21]:

:tual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor	segment_actual_time	segment_osrm_time	segment_osrm_distance	segment_factor
10.435660	14.0	11.0	11.9653	1.272727	14.0	11.0	11.9653	1.272727
18.936842	24.0	20.0	21.7243	1.200000	10.0	9.0	9.7590	1.111111
27.637279	40.0	28.0	32.5395	1.428571	16.0	7.0	10.8152	2.285714
36.118028	62.0	40.0	45.5620	1.550000	21.0	12.0	13.0224	1.750000
39.386040	68.0	44.0	54.2181	1.545455	6.0	5.0	3.9153	1.200000

In [22]:
```
DLRYPD['route_type'].value_counts().plot(kind='bar',color='indigo')
plt.title('Route Type Data cout')
plt.show()
```

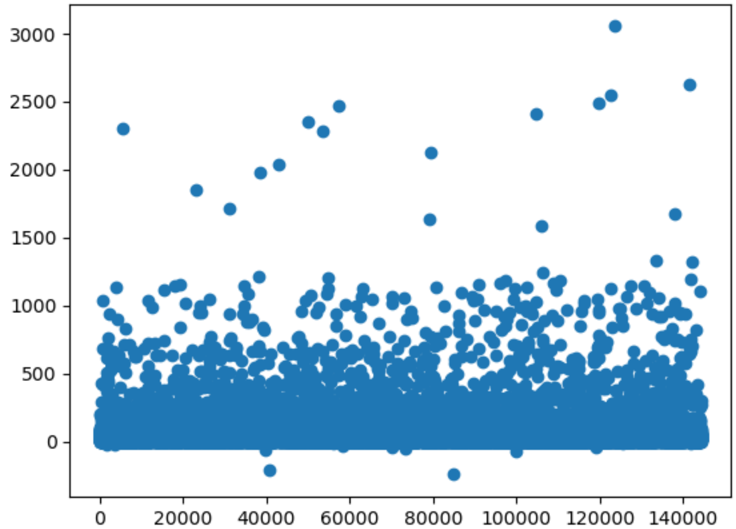


In [23]:
```
DLRYPD['route_type'].value_counts()
```

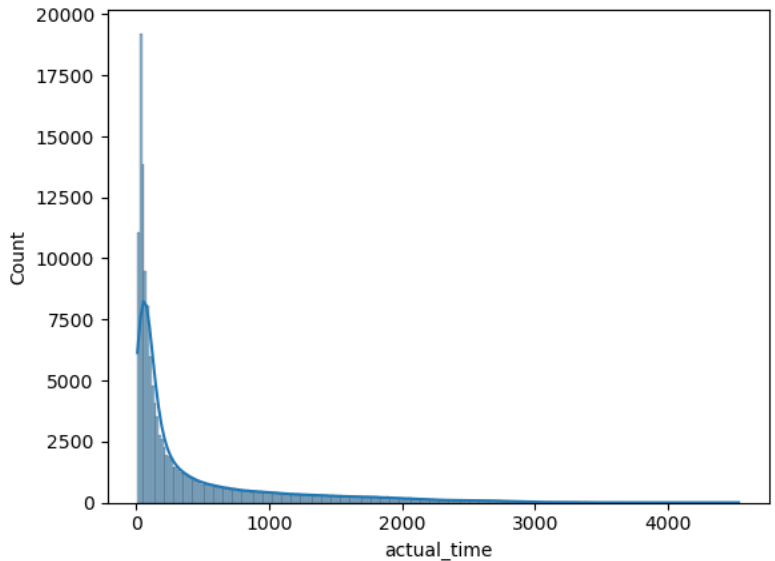
Out[23]:
```
FTL        99132
Carting    45184
Name: route_type, dtype: int64
```

It can be seen that the route type FTL has 99132 count. Carting route type has 45184 count. FTL has more route types than carting

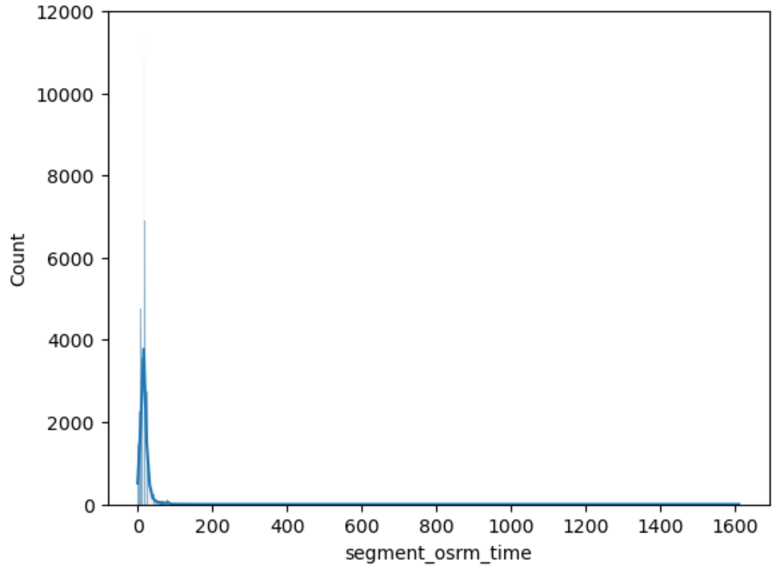
In [24]:
```
plt.scatter(DLRYPD.index,DLRYPD['segment_actual_time'])
plt.show()
```



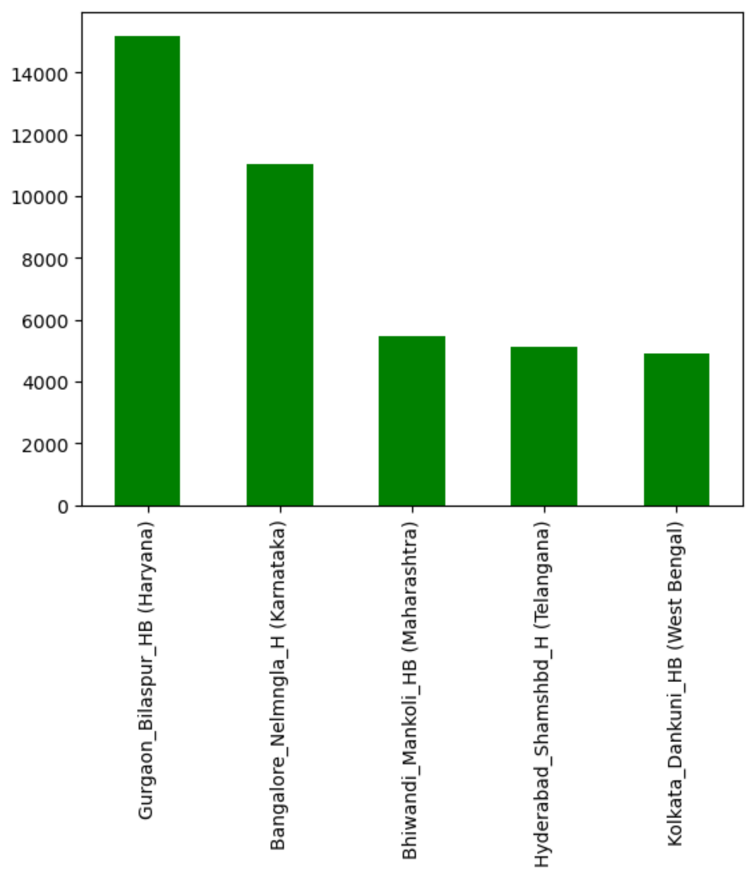
In [25]:
```
#DLRYPD['actual_time'].plot(kind='density')
sns.histplot(x='actual_time', data=DLRYPD, kde=True)
plt.show()
```



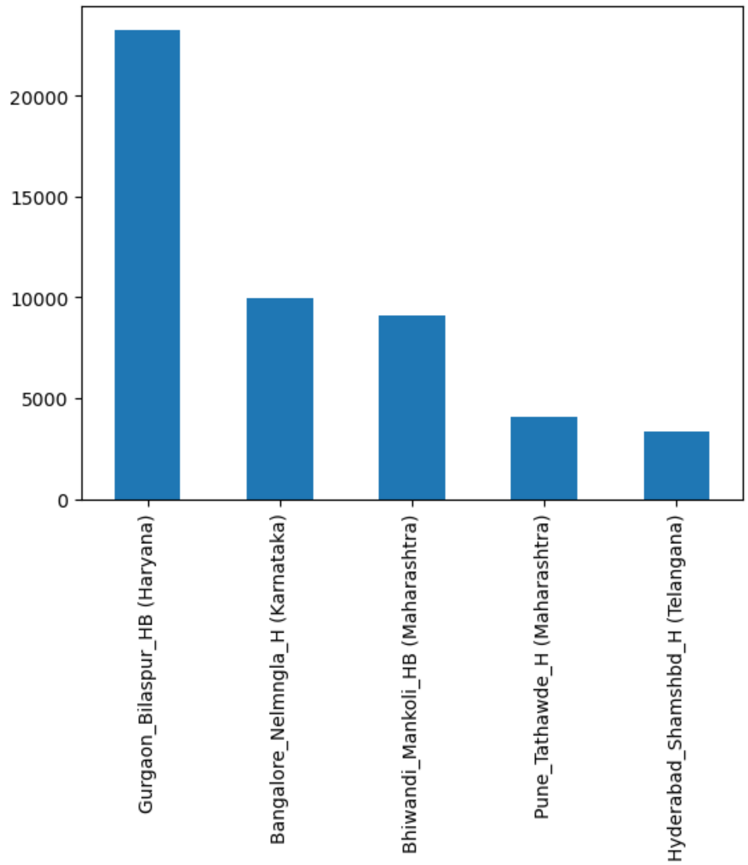
In [33]:
```
sns.histplot(x='segment_osrm_time', data=DLRYPD, kde=True)
plt.show()
```



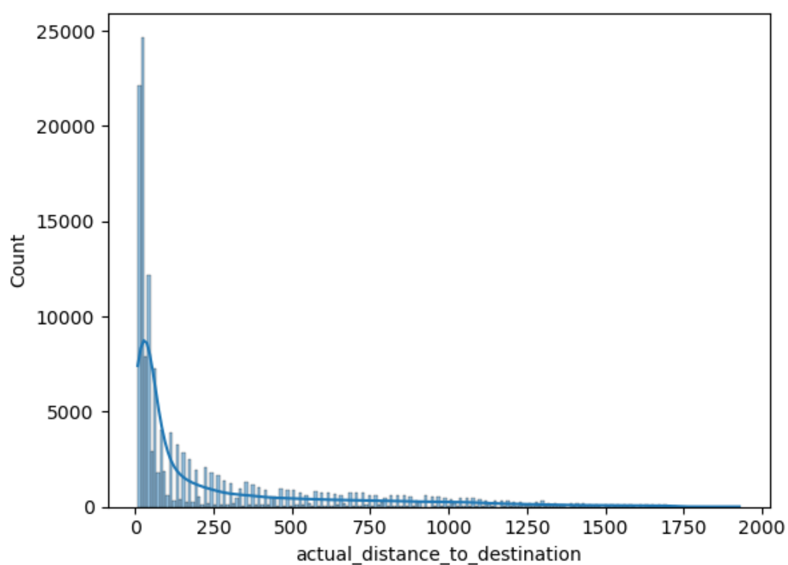
In [34]:
```
DLRYPD['destination_name'].value_counts().head().plot(kind='bar',color='green')
plt.show()
```



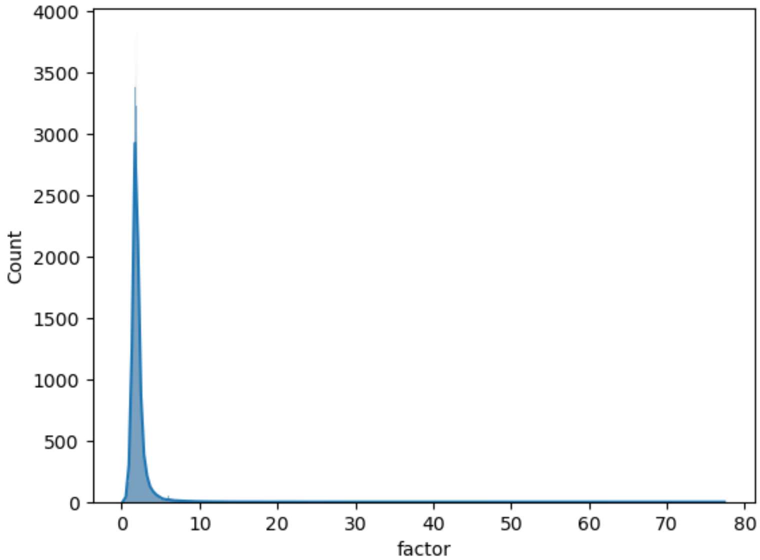
In [35]:
```
DLRYPD['source_name'].value_counts().head().plot(kind='bar')
plt.show()
```



In [36]:
```
sns.histplot(x='actual_distance_to_destination', data=DLRYPD, kde=True)
plt.show()
```



In [37]:
```
sns.histplot(x='factor', data=DLRYPD, kde=True)
plt.show()
```



Data Pre-Processing

Feature Creation and Deletion of Unnecessary Features

Creating a new column called "SegmentID"

In [42]:
```
DLRYPD['segment_id'] = DLRYPD['trip_uuid'] + "*" + DLRYPD['source_center'] + "*" + DLRYPD['destination_center']
```

In [43]:
```
DLRYPD.head()
```

Out[43]:

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

5 rows × 25 columns

In [45]:
```
# Grouping and calculating the cumulative sum of columns 'sum_seg_actual_time'
DLRYPD['sum_seg_actual_time'] = DLRYPD.groupby('segment_id')['segment_actual_time'].cumsum()
```

In [46]:
```
# Grouping and calculating the cumulative sum of columns 'sum_seg_osrm_distance'
DLRYPD['sum_seg_osrm_distance'] = DLRYPD.groupby('segment_id')['segment_osrm_distance'].cumsum()
```

In [47]:
```
# Grouping and calculating the cumulative sum of columns 'sum_seg_osrm_time'
DLRYPD['sum_seg_osrm_time'] = DLRYPD.groupby('segment_id')['segment_osrm_time'].cumsum()
```

In [48]:
```
DLRYPD.head()
```

Out[48]:

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

5 rows × 28 columns

Creating a dictionary for segment wise aggregation

In [49]:
```
dict_segment = {
    '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',

    'sum_seg_actual_time' : 'last',
    'sum_seg_osrm_distance' : 'last',
    'sum_seg_osrm_time' : 'last'
}
```

In [51]:
```
# Grouping the rows as per 'segment_id' and sorting the dataset by mentioned columns
DLRYPD_segment_wise = DLRYPD.groupby('segment_id').agg(dict_segment).reset_index()
DLRYPD_segment_wise = DLRYPD_segment_wise.sort_values(by = ['segment_id','od_end_time'], ascending = True).reset_index(drop = Tru
```

In [53]:
```
# Adding a new column 'od_time_diff_minutes'
DLRYPD_segment_wise['od_time_diff_minutes'] = (DLRYPD_segment_wise['od_end_time'] - DLRYPD_segment_wise['od_start_time']).dt.tota
```

In [55]:
```
# Dropping columns 'od_start_time' and 'od_end_time'
DLRYPD_segment_wise = DLRYPD_segment_wise.drop(['od_start_time','od_end_time'], axis = 1)
```

In [56]:
```
DLRYPD_segment_wise.head()
```

Out[56]:

	segment_id	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	
0	trip-153671041653548748*IND209304AAA*IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	K
1	trip-153671041653548748*IND462022AAA*IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	
2	trip-153671042288605164*IND561203AAB*IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Dodda
3	trip-153671042288605164*IND572101AAA*IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND572101AAA	
4	trip-153671043369099517*IND000000ACB*IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB	Gu

Trip wise aggregation

In [57]:
```
# Creating a dictionary for trip-wise aggregation
dict_trip = {
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',

    'source_center' : 'last',
    'source_name' : 'last',

    'destination_center' : 'first',
    'destination_name' : 'first',

    'start_scan_to_end_scan' : 'sum',
    'od_time_diff_minutes' : 'sum',

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

    'sum_seg_actual_time' : 'sum',
    'sum_seg_osrm_distance' : 'sum',
    'sum_seg_osrm_time' : 'sum'
}
```

In [59]:
```
# Trip-wise Aggregation of data
DLRYPD_trip_wise = DLRYPD_segment_wise.groupby('trip_uuid').agg(dict_trip).reset_index(drop = True)
```

In [61]:
```
DLRYPD_trip_wise.head()
```

Out[61]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destinati
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND000000ACB	Gurgaon_Bil
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND572101AAA	Tumkur_Veersagr_I (Karnataka)	IND562101AAA	Chikblapur_S (K
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND562132AAA	Bangalore_Nelmngla_H (Karnataka)	IND160002AAC	Chandigarh_Meh
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072AAB	Mumbai Hub (Maharashtra)	IND401104AAA	Mumbai_M (Mal
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583201AAA	Hospet (Karnataka)	IND583201AAA	Hospet (K

In [62]:
```
DLRYPD_trip_wise.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  object
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  object
 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_time_diff_minutes            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  sum_seg_actual_time             14787 non-null  float64
 16  sum_seg_osrm_distance           14787 non-null  float64
 17  sum_seg_osrm_time               14787 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(8)
memory usage: 2.0+ MB
```

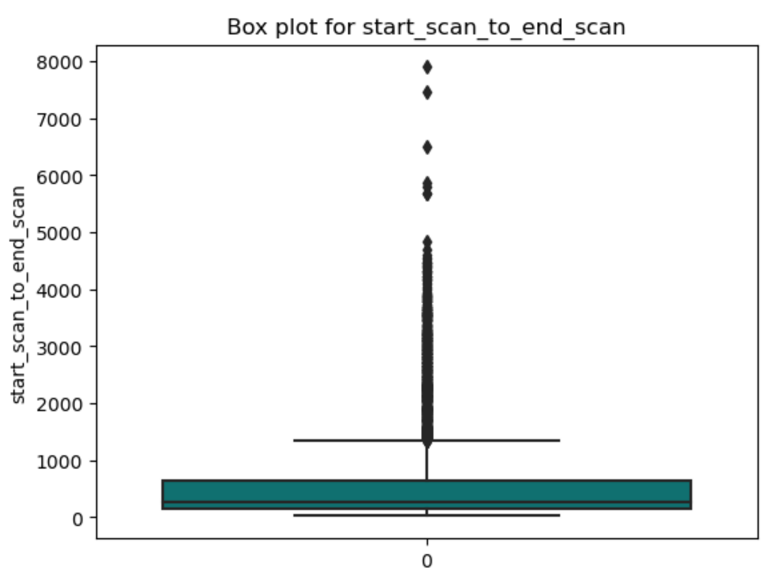
OUTLIER DETECTION

In [63]:
```
def find_outliers_IQR(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    IQR = q3 - q1
    outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
    return outliers
```

In [67]:
```
start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['start_scan_to_end_scan'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1282
max outlier value: 7898.0
min outlier values: 1357.0
```

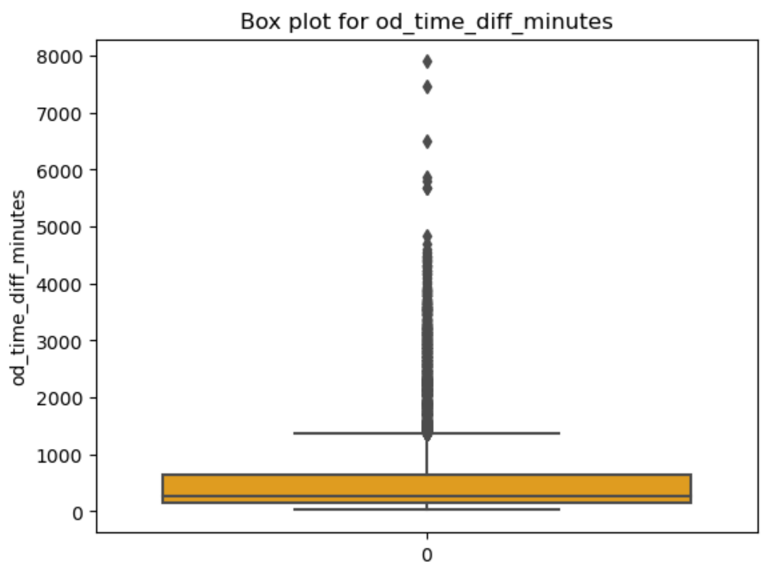
In [73]:
```
#Visualizing the outliers in the start_scan_to_end_scan column using box plot

sns.boxplot(data=DLRYPD_trip_wise['start_scan_to_end_scan'],color = 'teal')
plt.ylabel('start_scan_to_end_scan')
plt.title("Box plot for start_scan_to_end_scan")
plt.show()
```



In [69]:
```
#Checking outliers for 'od_time_diff_minutes'

start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['od_time_diff_minutes'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1275
max outlier value: 7898.551954566667
min outlier values: 1359.5605082166667
```

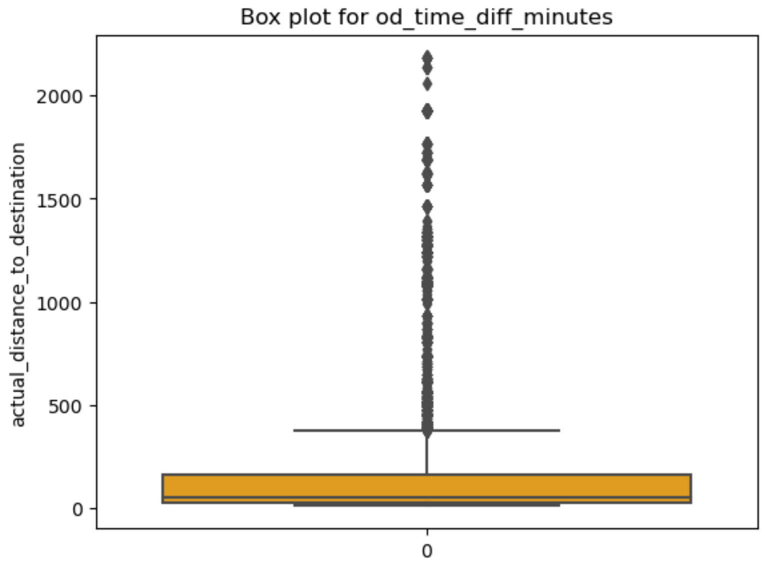
In [76]:
```
#Visualizing the outliers in the od_time_diff_minutes column using box plot

sns.boxplot(data=DLRYPD_trip_wise['od_time_diff_minutes'],color = 'orange')
plt.ylabel('od_time_diff_minutes')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [77]:
```
#Checking outliers for 'actual_distance_to_destination'
start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['actual_distance_to_destination'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1452
max outlier value: 2186.531787238833
min outlier values: 374.9746646495524
```

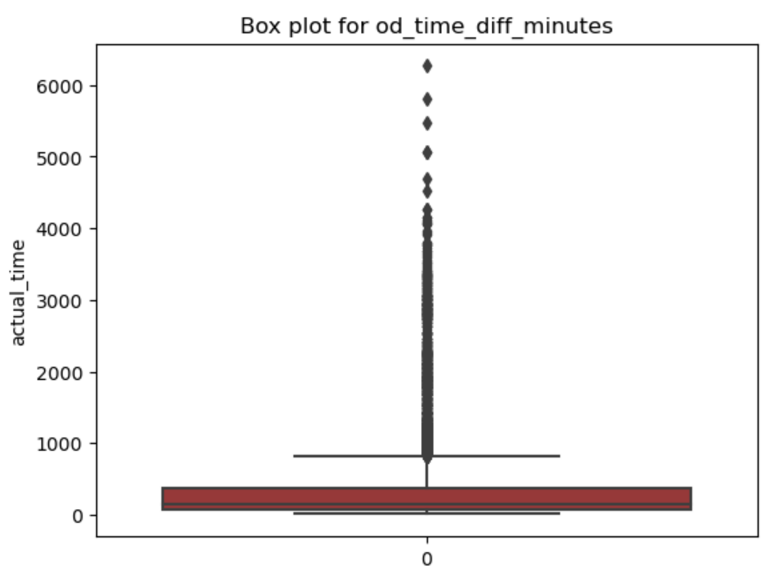
In [78]:
```
#Visualizing the outliers in the 'actual_distance_to_destination'

sns.boxplot(data=DLRYPD_trip_wise['actual_distance_to_destination'],color = 'orange')
plt.ylabel('actual_distance_to_destination')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [79]:
```
#Checking outliers for 'actual_time'
start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['actual_time'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1646
max outlier value: 6265.0
min outlier values: 818.0
```

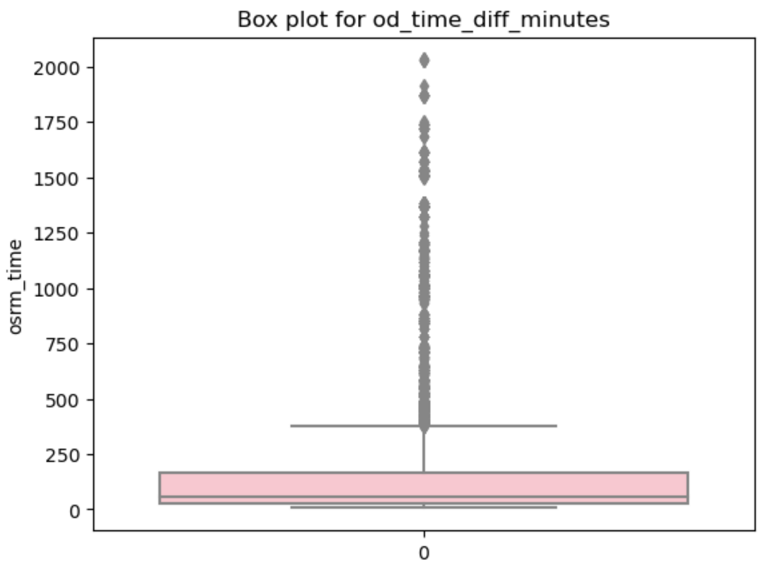
In [81]:
```
#Visualizing the outliers in the 'actual_time'

sns.boxplot(data=DLRYPD_trip_wise['actual_time'],color = 'brown')
plt.ylabel('actual_time')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [82]:
```
#Checking outliers for 'osrm_time'
start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['osrm_time'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1506
max outlier value: 2032.0
min outlier values: 377.0
```

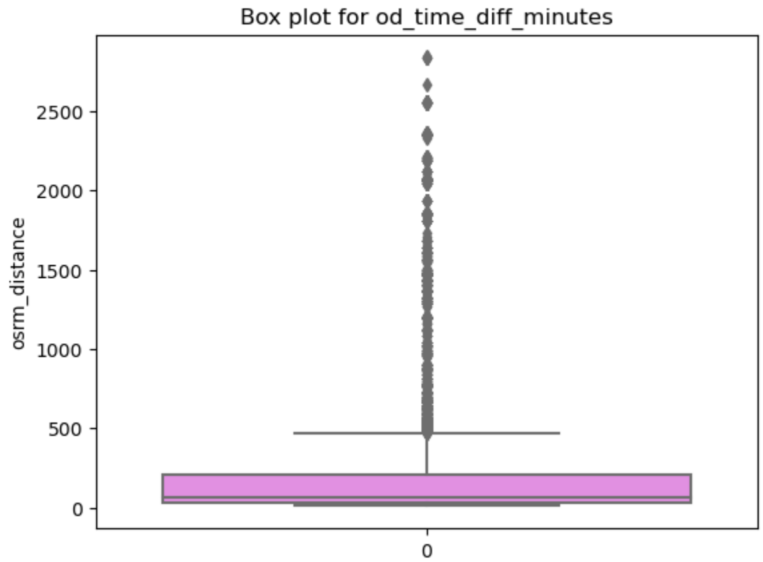
In [84]:
```
#Visualizing the outliers in the 'osrm_time'

sns.boxplot(data=DLRYPD_trip_wise['osrm_time'],color = 'pink')
plt.ylabel('osrm_time')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [85]:
```
#Checking outliers for 'osrm_distance'

start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['osrm_distance'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1522
max outlier value: 2840.081
min outlier values: 470.57349999999997
```

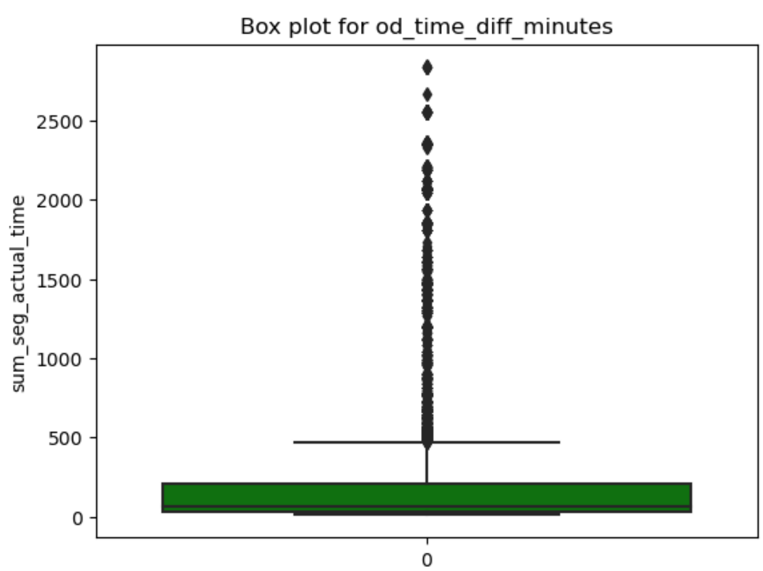
In [86]:
```
#Visualizing the outliers in the 'osrm_distance'

sns.boxplot(data=DLRYPD_trip_wise['osrm_distance'],color = 'violet')
plt.ylabel('osrm_distance')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [87]:
```
#Checking outliers for 'sum_seg_actual_time'

start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['sum_seg_actual_time'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1644
max outlier value: 6230.0
min outlier values: 813.0
```

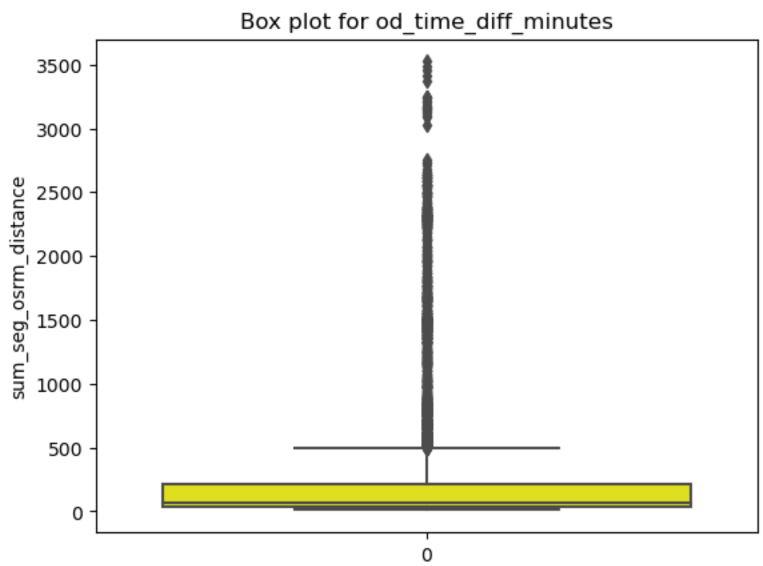
In [89]:
```
#Visualizing the outliers in the 'sum_seg_actual_time'

sns.boxplot(data=DLRYPD_trip_wise['osrm_distance'],color = 'green')
plt.ylabel('sum_seg_actual_time')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [90]:
```
#Checking outliers for 'sum_seg_osrm_distance'

start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['sum_seg_osrm_distance'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1550
max outlier value: 3523.6323999999995
min outlier values: 493.5402
```

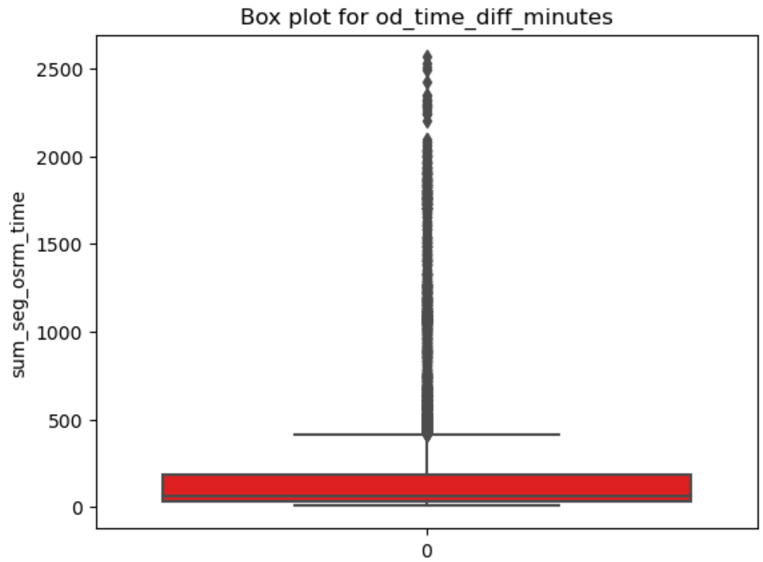
In [92]:
```
#Visualizing the outliers in the 'sum_seg_osrm_distance'

sns.boxplot(data=DLRYPD_trip_wise['sum_seg_osrm_distance'],color = 'yellow')
plt.ylabel('sum_seg_osrm_distance')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



In [93]:
```
#Checking outliers for 'sum_seg_osrm_time'

start_scan_to_end_scan_OUTLIER = find_outliers_IQR(DLRYPD_trip_wise['sum_seg_osrm_time'])

print('number of outliers: '+ str(len(start_scan_to_end_scan_OUTLIER)))

print('max outlier value: '+ str(start_scan_to_end_scan_OUTLIER.max()))

print('min outlier values: '+ str(start_scan_to_end_scan_OUTLIER.min()))
```

```
number of outliers: 1485
max outlier value: 2564.0
min outlier values: 416.0
```

In [95]:
```
#Visualizing the outliers in the 'sum_seg_osrm_time'

sns.boxplot(data=DLRYPD_trip_wise['sum_seg_osrm_time'],color = 'red')
plt.ylabel('sum_seg_osrm_time')
plt.title("Box plot for od_time_diff_minutes")
plt.show()
```



Observations from outlier detection

1. All the numerical features available in the dataset have a large number of potential outliers.
2. For feature 'start_scan_to_end_scan', 'od_time_diff_minutes', median point is around 250.
3. For feature 'actual_distance_to_destination', the median point is around 75-100.
4. For feature 'actual_time' and 'osrm_time', the median point is around 100 and seems visually pretty close to each other.
5. The median of feature 'sum_seg_osrm_time' seems visually more than the feature 'sum_seg_actual_time'.

Outlier Treatment and Column Standardization

In [97]:
```
tripwise_copy = DLRYPD_trip_wise.copy(deep = True)
tripwise_copy.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  object
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  object
 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_time_diff_minutes            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  sum_seg_actual_time             14787 non-null  float64
 16  sum_seg_osrm_distance           14787 non-null  float64
 17  sum_seg_osrm_time               14787 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(8)
memory usage: 2.0+ MB
```

In [98]:
```
tripwise_copy.describe()
```

Out[98]:

	start_scan_to_end_scan	od_time_diff_minutes	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	sum_seg_actual_time	sum_seg
count	14787.000000	14787.000000	14787.000000	14787.000000	14787.000000	14787.000000	14787.000000	
mean	529.429025	530.313517	164.090196	356.306012	160.990938	203.887411	353.059174	
std	658.254936	658.415490	305.502982	561.517936	271.459495	370.565564	556.365911	
min	23.000000	23.461468	9.002461	9.000000	6.000000	9.072900	9.000000	
25%	149.000000	149.698496	22.777099	67.000000	29.000000	30.756900	66.000000	
50%	279.000000	279.710750	48.287894	148.000000	60.000000	65.302800	147.000000	
75%	632.000000	633.537697	163.591258	367.000000	168.000000	206.644200	364.000000	
max	7898.000000	7898.551955	2186.531787	6265.000000	2032.000000	2840.081000	6230.000000	

In [99]:
```
#Using IQR method to treat outliers

f_cols = ['start_scan_to_end_scan','od_time_diff_minutes','actual_distance_to_destination','actual_time','osrm_time','osrm_distar
for i in f_cols:
    q1 = DLRYPD_trip_wise[i].quantile(0.25)
    q3 = DLRYPD_trip_wise[i].quantile(0.75)
    iqr = q3 - q1
    tripwise_copy = tripwise_copy[(tripwise_copy[i] > (q1 - 1.5*iqr)) & (tripwise_copy[i] < (q3 + 1.5*iqr))]
```

In [100]:
```
tripwise_copy.info()
```

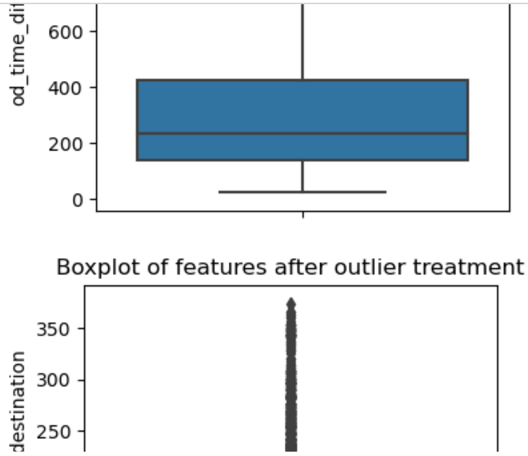
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12723 entries, 1 to 14786
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            12723 non-null  object
 1   trip_creation_time              12723 non-null  datetime64[ns]
 2   route_schedule_uuid             12723 non-null  object
 3   route_type                      12723 non-null  object
 4   trip_uuid                       12723 non-null  object
 5   source_center                   12723 non-null  object
 6   source_name                     12723 non-null  object
 7   destination_center              12723 non-null  object
 8   destination_name                12723 non-null  object
 9   start_scan_to_end_scan          12723 non-null  float64
 10  od_time_diff_minutes            12723 non-null  float64
 11  actual_distance_to_destination  12723 non-null  float64
 12  actual_time                     12723 non-null  float64
 13  osrm_time                       12723 non-null  float64
 14  osrm_distance                   12723 non-null  float64
 15  sum_seg_actual_time             12723 non-null  float64
 16  sum_seg_osrm_distance           12723 non-null  float64
 17  sum_seg_osrm_time               12723 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(8)
memory usage: 1.8+ MB
```

In [101]:
```
tripwise_copy.describe()
```

Out[101]:

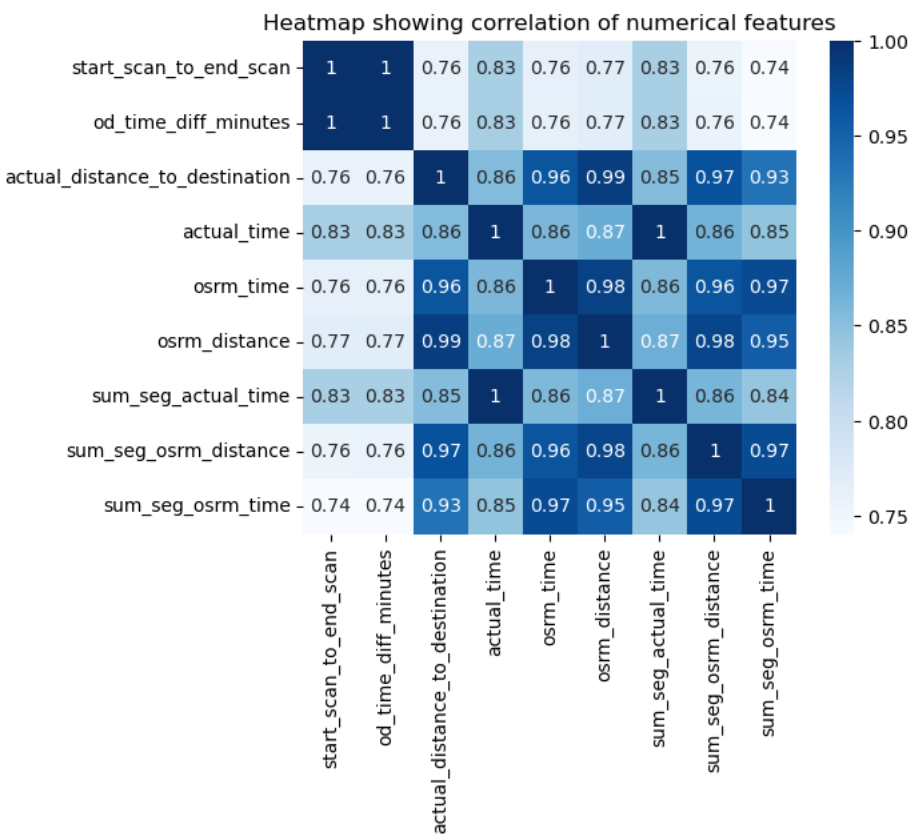
	start_scan_to_end_scan	od_time_diff_minutes	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	sum_seg_actual_time	sum_seg
count	12723.000000	12723.000000	12723.000000	12723.000000	12723.000000	12723.000000	12723.000000	
mean	320.178731	321.022701	72.317812	177.452723	78.440305	91.734030	175.796274	
std	255.555831	255.885432	72.070232	158.150841	72.333674	89.566572	157.099770	
min	23.000000	23.461468	9.002461	9.000000	6.000000	9.072900	9.000000	
25%	136.000000	136.523359	21.395561	61.000000	27.000000	28.344450	60.000000	
50%	233.000000	233.549105	38.525319	114.000000	50.000000	48.418300	113.000000	
75%	423.000000	423.905113	101.673567	251.000000	109.000000	131.316850	248.000000	
max	1355.000000	1357.397291	373.441224	815.000000	376.000000	463.478100	810.000000	

In [102]:
```
# Checking the plots of the numerical features after outlier treatment
for j in f_cols:
    fig, ax = plt.subplots(figsize = (4,4))
    sns.boxplot(data = tripwise_copy, y = j, ax = ax).set(title = 'Boxplot of features after outlier treatment')
    plt.show()
```



In [103]:
```
#Plotting the heatmap for numerical features
```

In [104]:
```
sns.heatmap(tripwise_copy.corr(method = 'pearson'), square = True, annot = True, cmap = 'Blues').set(title = 'Heatmap showing cor
plt.show()
```



Hypothesis Testing & Bi-Variate Visual Analysis

Comparison of Time and Distance Fields and Checking relationship between Aggregated Fields

A. Hypothesis Testing to compare the difference between 'start_scan_to_end_scan' and 'od_time_diff_minutes'

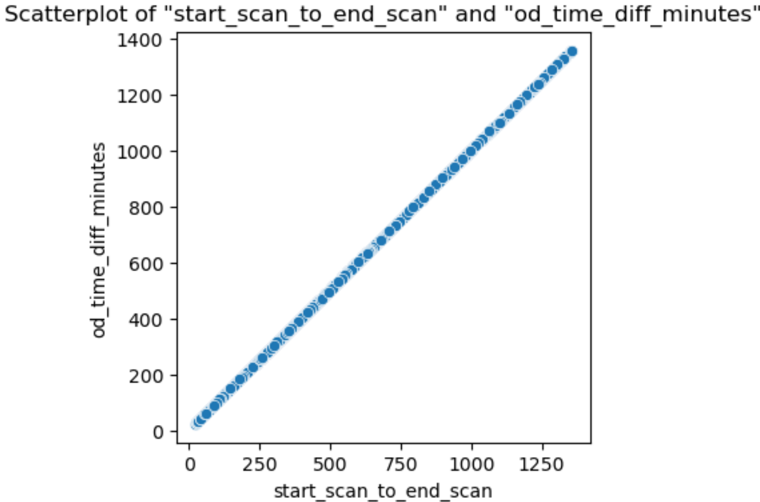
Null Hypothesis (H0): The means of both the independent samples i.e. 'start_scan_to_end_scan' and 'od_time_diff_minutes' are equal i.e. distributions of both samples are equal.

Alternate Hypothesis(Ha): The means of both the independent samples i.e.'start_scan_to_end_scan' and 'od_time_diff_minutes' are not equal i.e. distributions of both samples are not equal

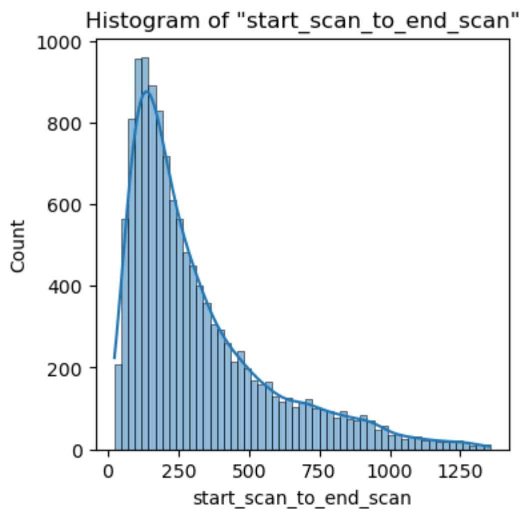
Assumed Significance level (alpha): 0.05

This means that if the p-value of the tests is less than the assumed significance level, we will REJECT the Null Hypothesis and vice-versa.

In [106]:
```
# Plot showing 'start_scan_to_end_scan' vs. 'od_time_diff_minutes'
fig, ax = plt.subplots(figsize = (4,4))
sns.scatterplot(data = tripwise_copy, x = 'start_scan_to_end_scan', y = 'od_time_diff_minutes', ax = ax).set(title = 'Scatterplot
plt.show()
```

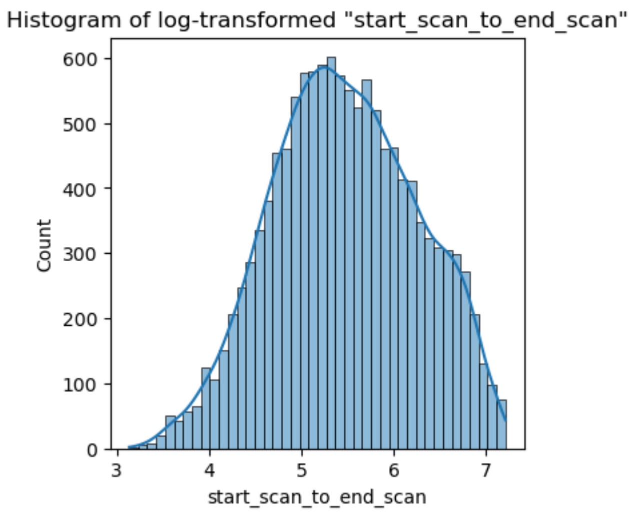


In [107]:
```
# Histogram showing 'start_scan_to_end_scan'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'start_scan_to_end_scan', ax = ax, kde = True).set(title = 'Histogram of "start_scan_to_er
plt.show()
```

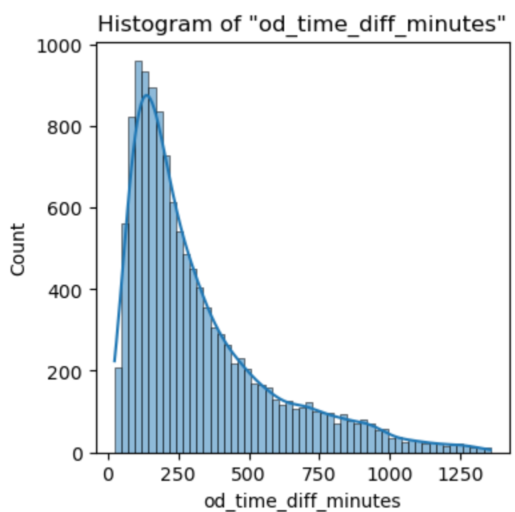


The above histogram of feature 'start_scan_to_end_scan' is right-skewed, This feature could follow a log normal distribution and to check this we are going to plot a histogram of the log-transformed data. This help us to understand if the plot after log normal transformation follows gaussian distribution

In [110]:
```
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = np.log(tripwise_copy['start_scan_to_end_scan']), ax = ax, kde = True).set(title = 'Histogram of log-transformed
plt.show()
```

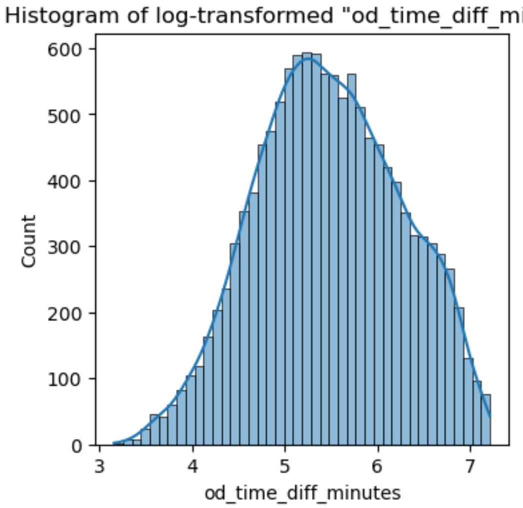


In [111]:
```
# Histogram of 'od_time_diff_minutes'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'od_time_diff_minutes', ax = ax, kde = True).set(title = 'Histogram of "od_time_diff_minut
plt.show()
```



The above histogram of feature 'od_time_diff_minutes' is right-skewed, This feature could follow a log normal distribution and to check this we are going to plot a histogram of the log-transformed data. This help us to understand if the plot after log normal transformation follows gaussian distribution

In [112]:
```
# Histogram of log-transformed values of 'od_time_diff_minutes' feature
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = np.log(tripwise_copy['od_time_diff_minutes']), ax = ax, kde = True).set(title = 'Histogram of log-transformed "c
plt.show()
```



The log-transformed plots for both the features seems to be Gaussian. But there seems to be a minor peak in the histograms of both the log-transformed plots. So using a statistical test to test both the cases. To perform 2 sample t test we need to check the assumption of a T-test"

Case-relevant Assumptions of 2-Sample T-Test

1. Data in each group should be NORMALLY Distributed.
2. Data values should be independent.
3. The Variances of the two independent groups should be EQUAL.

For Normality, we will do SHAPIRO-WILK Test. For Equi-Variance, we will do LEVENE'S Test.

SHAPIRO-WILK'S TEST:

For Shapiro-Test:

Null Hypothesis: Sample follows a gaussian distribution.

Alternate Hypothesis: Sample does not follow a gaussian distribution.

In [114]:
```
#Importing the required librarires

from scipy.stats import shapiro
from scipy.stats import levene
from scipy.stats import ttest_ind
from scipy.stats import kruskal
from sklearn import preprocessing
```

In [116]:
```
test_stat_sh1, p_val_1 = shapiro(np.log(tripwise_copy['start_scan_to_end_scan']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh1}, P-value of Shapiro Test: {p_val_1}")
if p_val_1 < 0.05:
    print('Sample follows a Gaussian Distribution')
else:
    print('Sample does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9921913743019104, P-value of Shapiro Test: 1.03336088559351 97e-25
Sample follows a Gaussian Distribution

C:\Users\india\anaconda3\lib\site-packages\scipy\stats\_morestats.py:1800: UserWarning: p-value may not be accurate for N > 500
0.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

In [118]:
```
# Shapiro-Wilk's Test for 'od_time_diff_minutes'
test_stat_sh2, p_val_2 = shapiro(np.log(tripwise_copy['od_time_diff_minutes']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh2}, P-value of Shapiro Test: {p_val_2}")
if p_val_2 < 0.05:
    print('Sample follows a Gaussian Distribution')
else:
    print('Sample does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9920996427536011, P-value of Shapiro Test: 7.419760050250907e-26
Sample follows a Gaussian Distribution
```

As per SHAPIRO-WILK'S TEST of both the features i.e. 'start_scan_to_end_scan' and 'od_time_diff_minutes' we can see that they follow GAUSSIAN Distribution.

LEVENE'S TEST:

For Levene's Test:

Null Hypothesis (H0): The variance of features 'start_scan_to_end_scan' and 'od_time_diff_minutes' will be equal.

Alternate Hypothesis (Ha): The variance of features will not be equal.

In [121]:
```
# Levene's test for checking Equi-Variance of features mentioned
test_stat_l1, p_val_l1 = levene(np.log(tripwise_copy['start_scan_to_end_scan']), np.log(tripwise_copy['od_time_diff_minutes']), c
print(f"Test Statistic for Levene's Test: {test_stat_l1}, P-value for Levene's Test: {p_val_l1}")
if p_val_l1 < 0.05:
    print('Both the samples do not have equal variance')
else:
    print('Both the samples have equal variance')
```

```
Test Statistic for Levene's Test: 0.05066491477212913, P-value for Levene's Test: 0.8219120931501083
Both the samples have equal variance
```

We can proceed with the 2 sample t test as we can see that assumptions of Normality and Equi-Variance have been satisfied.

In [123]:
```
# Hypothesis Testing to check if the "start_scan_to_end_scan" and "od_time_diff_minutes" are from same distributions
t_test_stat_1, t_p_val_1 = ttest_ind(np.log(tripwise_copy['start_scan_to_end_scan']), np.log(tripwise_copy['od_time_diff_minutes'
print(f"Test Statistic for 2-Sample T-Test: {t_test_stat_1}, P-value of 2-Sample T-Test: {t_p_val_1}")
if t_p_val_1 < 0.05:
    print('Distributions of both the samples are not EQUAL.')
else:
    print('Distributions of both the samples are EQUAL')
```

```
Test Statistic for 2-Sample T-Test: -0.37408119821664204, P-value of 2-Sample T-Test: 0.7083470236477065
Distributions of both the samples are EQUAL
```

B. Hypothesis Testing to compare the difference between 'actual_time' aggregated value and 'osrm_time' aggregated value

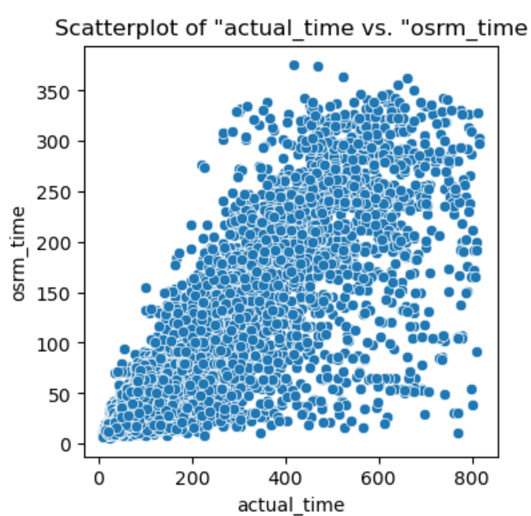
Null Hypothesis (H0): The means of both the independent samples i.e. 'actual_time' aggregated value and 'osrm_time' aggregated value are equal i.e. distributions of both samples are equal.

Alternate Hypothesis(Ha): The means of both the independent samples i.e.'actual_time' aggregated value and 'osrm_time' aggregated value are not equal i.e. distributions of both samples are not equal

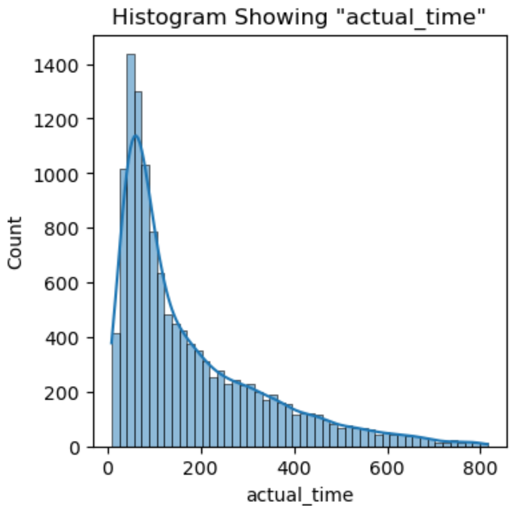
Assumed Significance level (alpha): 0.05

This means that if the p-value of the tests is less than the assumed significance level, we will REJECT the Null Hypothesis and vice-versa.

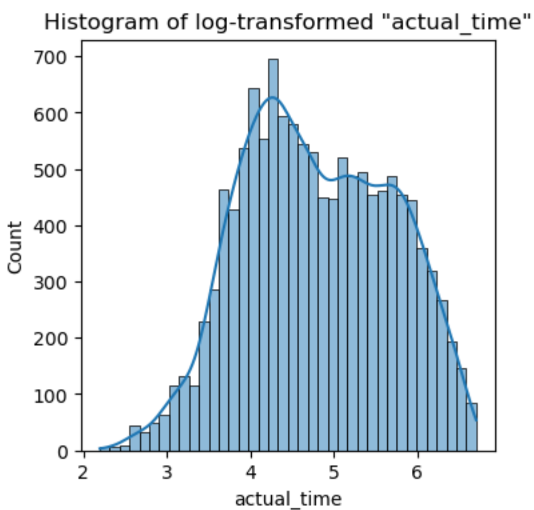
In [124]:
```
# Plot showing 'actual_time' vs. 'osrm_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.scatterplot(data = tripwise_copy, x = 'actual_time', y = 'osrm_time', ax = ax).set(title = 'Scatterplot of "actual_time vs. '
plt.show()
```



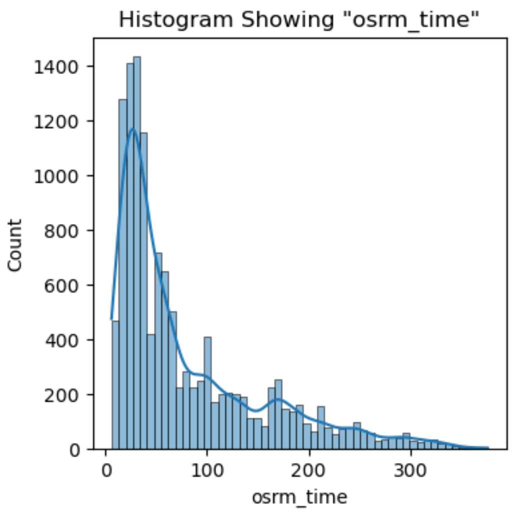
In [125]:
```
# Histogram of "actual_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'actual_time', ax = ax, kde = True).set(title = 'Histogram Showing "actual_time"')
plt.show()
```



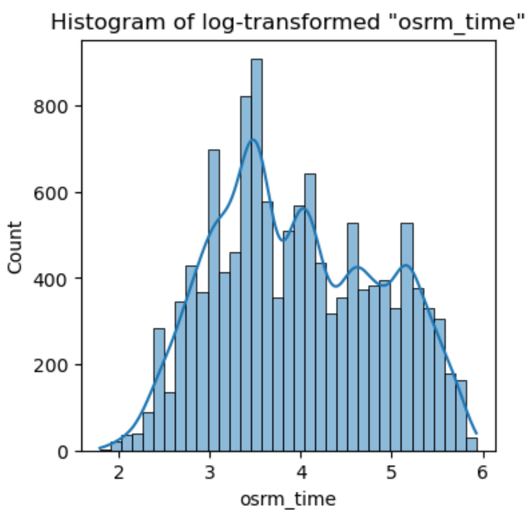
In [127]:
```
# log-transformed histogram plot feature 'actual_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['actual_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "actual_t
plt.show()
```



In [129]:
```
# Histogram of "osrm_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'osrm_time', ax = ax, kde = True).set(title = 'Histogram Showing "osrm_time"')
plt.show()
```



In [130]:
```
# Histogram of log-transformed feature 'osrm_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['osrm_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "osrm_time'
plt.show()
```



The log-transformed plots for both the features seems to be Gaussian. But there seems to be a minor peak in the histograms of both the log-transformed plots. So using a statistical test to test both the cases. To perform 2 sample t test we need to check the assumption of a T-test"

Case-relevant Assumptions of 2-Sample T-Test

Data in each group should be NORMALLY Distributed. Data values should be independent. The Variances of the two independent groups should be EQUAL. For Normality, we will do SHAPIRO-WILK Test. For Equi-Variance, we will do LEVENE'S Test.

SHAPIRO-WILK'S TEST:

For Shapiro-Test:

Null Hypothesis: Sample follows a gaussian distribution.

Alternate Hypothesis: Sample does not follow a gaussian distribution.

In [134]:
```
# Shapiro-Wilk's Test for 'actual_time'
test_stat_sh3, p_val_3 = shapiro(np.log(tripwise_copy['actual_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh3}, P-value of Shapiro Test: {p_val_3}")
if p_val_3 < 0.05:
    print('Sample "actual_time" follows a Gaussian Distribution')
else:
    print('Sample "actual_time" does not follow a Gaussian Distribution')
# Shapiro-Wilk's Test for 'osrm_time'
test_stat_sh4, p_val_4 = shapiro(np.log(tripwise_copy['osrm_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh4}, P-value of Shapiro Test: {p_val_4}")
if p_val_4 < 0.05:
    print('Sample "osrm_time" follows a Gaussian Distribution')
else:
    print('Sample "osrm_time" does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9844387769699097, P-value of Shapiro Test: 6.172019184770069e-35
Sample "actual_time" follows a Gaussian Distribution
Test Statistics of Shapiro Test: 0.9736400246620178, P-value of Shapiro Test: 5.4510510262235384e-43
Sample "osrm_time" follows a Gaussian Distribution
```

LEVENE'S TEST:

For Levene's Test:

Null Hypothesis (H0): The variance of features 'actual_time' and 'osrm_time' will be equal. Alternate Hypothesis (Ha): The variance of features will not be equal.

In [138]:
```
# Levene's test for checking Equi-Variance of features mentioned
test_stat_l2, p_val_l2 = levene(np.log(tripwise_copy['actual_time']), np.log(tripwise_copy['osrm_time']), center = 'median')
print(f"Test Statistic for Levene's Test: {test_stat_l2}, P-value for Levene's Test: {p_val_l2}")
if p_val_l2 < 0.05:
    print('Both the samples do not have equal variance')
else:
    print('Both the samples have equal variance')
```

```
Test Statistic for Levene's Test: 0.1773082369398479, P-value for Levene's Test: 0.6737003343148753
Both the samples have equal variance
```

CONCLUSION

The results of the Levene's Test shows that both the samples have equal variance.

So we can proceed with the 2-Sample T-Test since both the assumptions of Normality and Equi-Variance have been satisfied.

In [141]:
```
# Hypothesis Testing to check if the "actual_time" aggregated value and "osrm_time" aggregated value are from same distributions
t_test_stat_2, t_p_val_2 = ttest_ind(np.log(tripwise_copy['actual_time']), np.log(tripwise_copy['osrm_time']), alternative = 'two
print(f"Test Statistic for 2-Sample T-Test: {t_test_stat_2}, P-value of 2-Sample T-Test: {t_p_val_2}")
if t_p_val_2 < 0.05:
    print('Distributions of both the samples are not EQUAL.')
else:
    print('Distributions of both the samples are EQUAL')
```

```
Test Statistic for 2-Sample T-Test: 74.07939353363732, P-value of 2-Sample T-Test: 0.0
Distributions of both the samples are not EQUAL.
```

Non parametric test (Kruskal-Wallis Test) is required to be performed as the parametric test has yielded a p value as 0

In [143]:
```
# Doing Kruskal-Wallis Test just to double-check the above results
kr_test_stat_1, kr_p_val_1 = kruskal(tripwise_copy['actual_time'],tripwise_copy['osrm_time'])
if kr_p_val_1 < 0.05:
    print("The Distributions of the given samples are NOT EQUAL")
else:
    print("The Distributions of the given samples are EQUAL")
```

```
The Distributions of the given samples are NOT EQUAL
```

C. Hypothesis Testing to compare the difference between 'actual_time' aggregated value and 'sum_seg_actual_time'

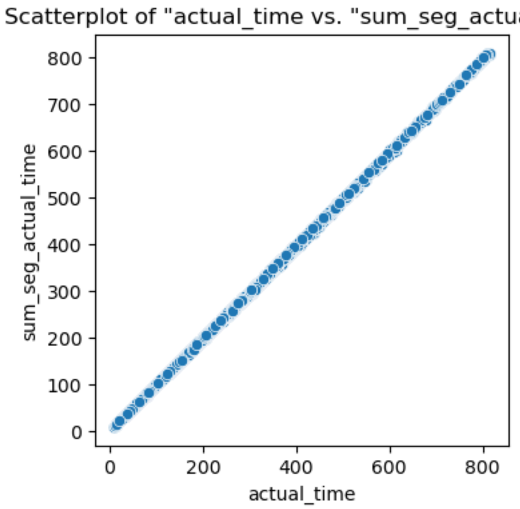
Null Hypothesis (H0): The means of both the independent samples i.e. 'actual_time' aggregated value and 'sum_seg_actual_time' are equal i.e. distributions of both samples are equal.

Alternate Hypothesis(Ha): The means of both the independent samples i.e.'actual_time' aggregated value and 'sum_seg_actual_time' are not equal i.e. distributions of both samples are not equal

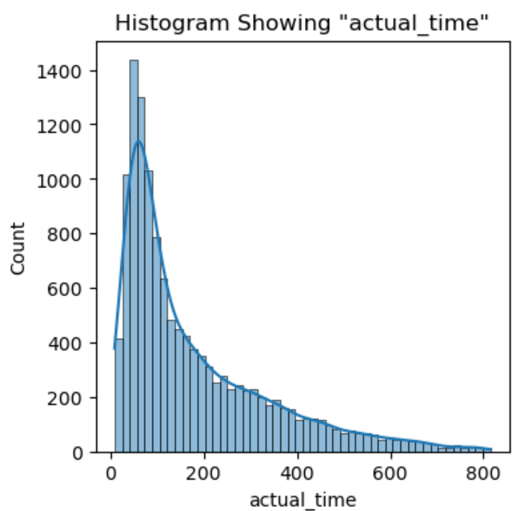
Assumed Significance level (alpha): 0.05

This means that if the p-value of the tests is less than the assumed significance level, we will REJECT the Null Hypothesis and vice-versa.

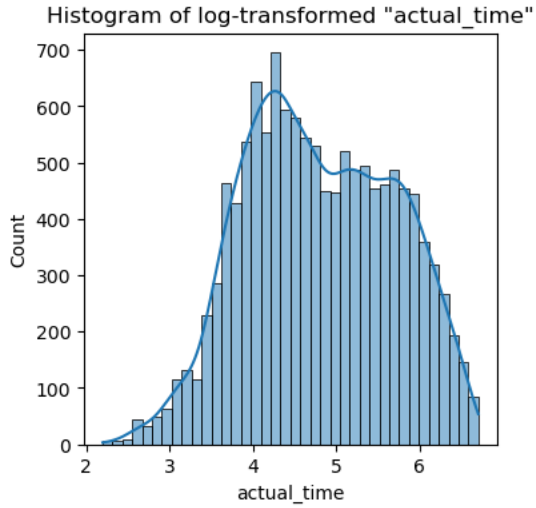
In [145]:
```
# Plot showing 'actual_time' vs. 'sum_seg_actual_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.scatterplot(data = tripwise_copy, x = 'actual_time', y = 'sum_seg_actual_time', ax = ax).set(title = 'Scatterplot of "actual_
plt.show()
```



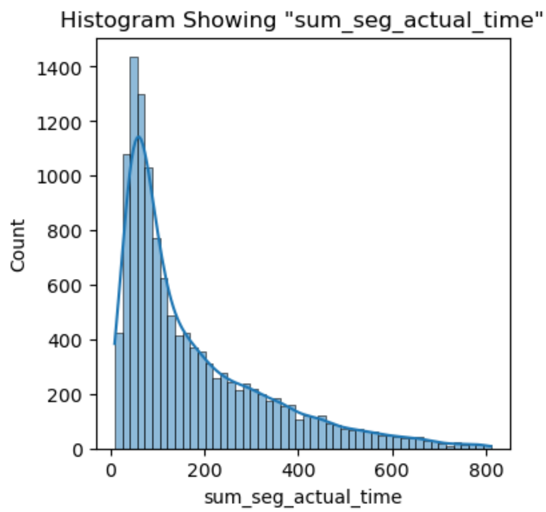
In [146]:
```
# Histogram of "actual_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'actual_time', ax = ax, kde = True).set(title = 'Histogram Showing "actual_time"')
plt.show()
```



In [148]:
```
# Histogram of log-transformed feature 'actual_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['actual_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "actual_t
plt.show()
```



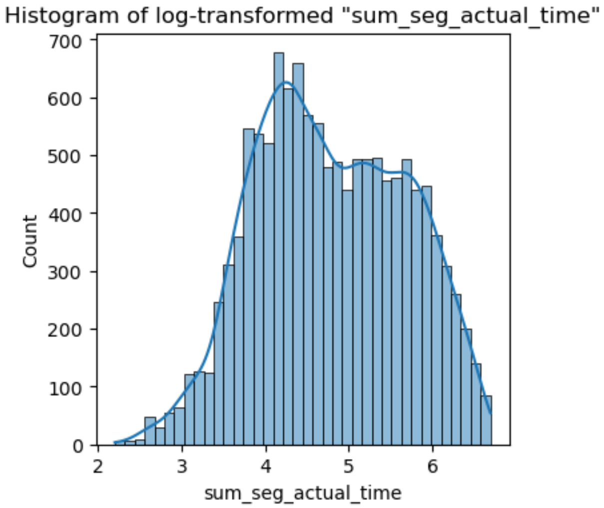
In [149]:
```
# Histogram of "sum_seg_actual_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'sum_seg_actual_time', ax = ax, kde = True).set(title = 'Histogram Showing "sum_seg_actual
plt.show()
```



Since the above histogram of feature 'sum_seg_actual_time' is right-skewed, there are chances of this feature following a log-normal distribution.

Plotting a histogram of the log-transformed data to see if the data is Gaussian.

In [150]:
```
# Histogram of log-transformed feature 'sum_seg_actual_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['sum_seg_actual_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed '
plt.show()
```



The log-transformed plots for both the features i.e. 'actual_time' and 'sum_seg_actual_time' does/does not seem Gaussian in quick view as the histogram seems to have minor peaks.

Thus a statistical test is the best option to test both these cases.

As statistical test is required to test both the cases checking the conditions for the relevant assumptions of 2 sample T-Test. Case-relevant Assumptions of 2-Sample T-Test

1. Data in each group should be NORMALLY Distributed.
2. Data values should be independent.
3. The Variances of the two independent groups should be EQUAL.

Thus we will need to do NORMALITY Test as well as EQUI-VARIANCE Test. For Normality, we will do SHAPIRO-WILK Test.

For Equi-Variance, we will do LEVENE'S Test.

SHAPIRO-WILK'S TEST: For Shapiro-Test: Null Hypothesis: Sample follows a gaussian distribution. Alternate Hypothesis: Sample does not follow a gaussian distribution.

In [151]:
```
# Shapiro-Wilk's Test for 'actual_time'
test_stat_sh5, p_val_5 = shapiro(np.log(tripwise_copy['actual_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh5}, P-value of Shapiro Test: {p_val_5}")
if p_val_5 < 0.05:
    print('Sample "actual_time" follows a Gaussian Distribution')
else:
    print('Sample "actual_time" does not follow a Gaussian Distribution')
# Shapiro-Wilk's Test for 'sum_seg_actual_time'
test_stat_sh6, p_val_6 = shapiro(np.log(tripwise_copy['sum_seg_actual_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh6}, P-value of Shapiro Test: {p_val_6}")
if p_val_6 < 0.05:
    print('Sample "sum_seg_actual_time" follows a Gaussian Distribution')
else:
    print('Sample "sum_seg_actual_time" does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9844387769699097, P-value of Shapiro Test: 6.172019184770069e-35
Sample "actual_time" follows a Gaussian Distribution
Test Statistics of Shapiro Test: 0.9843399524688721, P-value of Shapiro Test: 4.998404017787914e-35
Sample "sum_seg_actual_time" follows a Gaussian Distribution

C:\Users\india\anaconda3\lib\site-packages\scipy\stats\_morestats.py:1800: UserWarning: p-value may not be accurate for N > 500
0.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

LEVENE'S TEST:

For Levene's Test:

Null Hypothesis (H0): The variance of features 'actual_time' and 'sum_seg_actual_time' will be equal.

Alternate Hypothesis (Ha): The variance of features will not be equal.

In [152]:
```
# Levene's test for checking Equi-Variance of features mentioned
test_stat_l3, p_val_l3 = levene(np.log(tripwise_copy['actual_time']), np.log(tripwise_copy['sum_seg_actual_time']), center = 'mec
print(f"Test Statistic for Levene's Test: {test_stat_l3}, P-value for Levene's Test: {p_val_l3}")
if p_val_l3 < 0.05:
    print('Both the samples do not have equal variance')
else:
    print('Both the samples have equal variance')
```

```
Test Statistic for Levene's Test: 0.06224811418824465, P-value for Levene's Test: 0.8029793653927775
Both the samples have equal variance
```

As conditions for 2 sample T-Test have been satisfied we can proceed with the test.

In [153]:
```
# Hypothesis Testing to check if the "actual_time" aggregated value and "sum_seg_actual_time" aggregated value are from same dist
t_test_stat_3, t_p_val_3 = ttest_ind(np.log(tripwise_copy['actual_time']), np.log(tripwise_copy['sum_seg_actual_time']), alternat
print(f"Test Statistic for 2-Sample T-Test: {t_test_stat_3}, P-value of 2-Sample T-Test: {t_p_val_3}")
if t_p_val_3 < 0.05:
    print('Distributions of both the samples are not EQUAL.')
else:
    print('Distributions of both the samples are EQUAL')
```

```
Test Statistic for 2-Sample T-Test: 0.9677833919578374, P-value of 2-Sample T-Test: 0.3331617586609119
Distributions of both the samples are EQUAL
```

D. Hypothesis Testing to compare the difference between 'osrm_distance' aggregated value and 'sum_seg_osrm_distance'

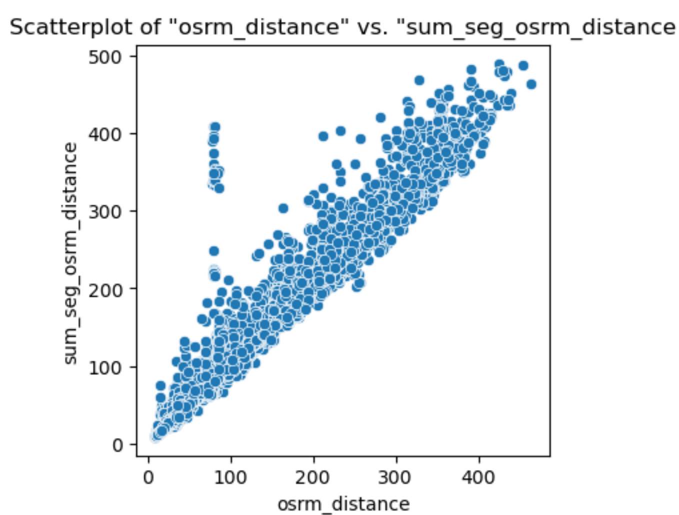
Null Hypothesis (H0): The means of both the independent samples i.e. 'osrm_distance' aggregated value and 'sum_seg_osrm_distance' are equal i.e. distributions of both samples are equal.

Alternate Hypothesis(Ha): The means of both the independent samples i.e.'osrm_distance' aggregated value and 'sum_seg_osrm_distance' are not equal i.e. distributions of both samples are not equal

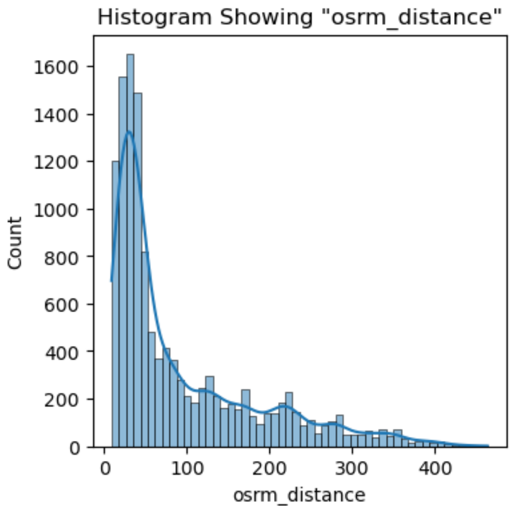
Assumed Significance level (alpha): 0.05

This means that if the p-value of the tests is less than the assumed significance level, we will REJECT the Null Hypothesis and vice-versa.

In [155]:
```
# Plot showing 'osrm_distance' vs. 'sum_seg_osrm_distance'
fig, ax = plt.subplots(figsize = (4,4))
sns.scatterplot(data = tripwise_copy, x = 'osrm_distance', y = 'sum_seg_osrm_distance', ax = ax).set(title = 'Scatterplot of "osr
plt.show()
```

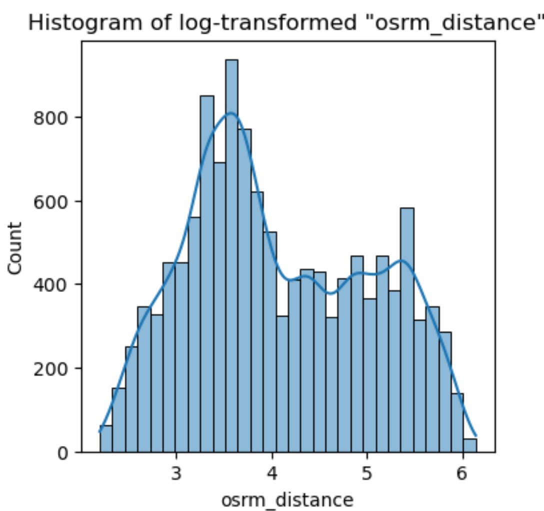


In [156]:
```
# Histogram of "osrm_distance"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'osrm_distance', ax = ax, kde = True).set(title = 'Histogram Showing "osrm_distance"')
plt.show()
```

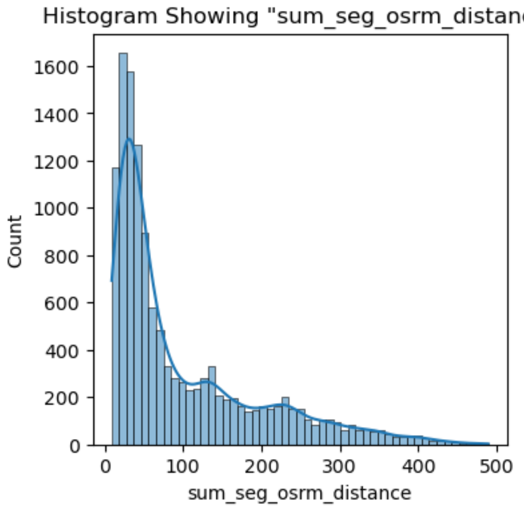


The histogram is once again right skewed so we proceed to check if the histogram plot of the lognormal transformation is Gaussian

In [157]:
```
# Histogram of log-transformed feature 'osrm_distance'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['osrm_distance'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "osrm_d
plt.show()
```



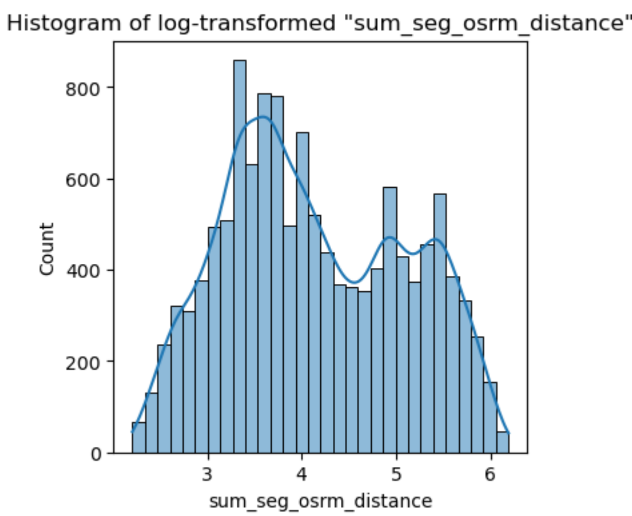
In [158]:
```
# Histogram of "sum_seg_osrm_distance"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'sum_seg_osrm_distance', ax = ax, kde = True).set(title = 'Histogram Showing "sum_seg_osrm
plt.show()
```



Histogram of feature 'sum_seg_osrm_distance' is once again right skewed.

Plotting the histogram of the log normal transformation to check if the its Gaussian.

In [159]:
```
# Histogram of log-transformed feature 'sum_seg_osrm_distance'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['sum_seg_osrm_distance'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed
plt.show()
```



The log-transformed plots for both the features i.e. 'actual_time' and 'sum_seg_actual_time' does/does not seem Gaussian in quick view as the histogram seems to have minor peaks.

Thus a statistical test is the best option to test both these cases.

As statistical test is required to test both the cases checking the conditions for the relevant assumptions of 2 sample T-Test.

Case-relevant Assumptions of 2-Sample T-Test

Data in each group should be NORMALLY Distributed. Data values should be independent. The Variances of the two independent groups should be EQUAL.

Thus we will need to do NORMALITY Test as well as EQUI-VARIANCE Test. For Normality, we will do SHAPIRO-WILK Test.

For Equi-Variance, we will do LEVENE'S Test.

SHAPIRO-WILK'S TEST: For Shapiro-Test: Null Hypothesis: Sample follows a gaussian distribution. Alternate Hypothesis: Sample does not follow a gaussian distribution.

SHAPIRO-WILK'S TEST:

For Shapiro-Test:

Null Hypothesis: Sample follows a gaussian distribution.

Alternate Hypothesis: Sample does not follow a gaussian distribution.

In [161]:
```
# Shapiro-Wilk's Test for 'osrm_distance'
test_stat_sh7, p_val_7 = shapiro(np.log(tripwise_copy['osrm_distance']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh7}, P-value of Shapiro Test: {p_val_7}")
if p_val_7 < 0.05:
    print('Sample "osrm_distance" follows a Gaussian Distribution')
else:
    print('Sample "osrm_distance" does not follow a Gaussian Distribution')
# Shapiro-Wilk's Test for 'sum_seg_osrm_distance'
test_stat_sh8, p_val_8 = shapiro(np.log(tripwise_copy['sum_seg_osrm_distance']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh8}, P-value of Shapiro Test: {p_val_8}")
if p_val_8 < 0.05:
    print('Sample "sum_seg_osrm_distance" follows a Gaussian Distribution')
else:
    print('Sample "sum_seg_osrm_distance" does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9623979330062866, P-value of Shapiro Test: 0.0
Sample "osrm_distance" follows a Gaussian Distribution
Test Statistics of Shapiro Test: 0.9668007493019104, P-value of Shapiro Test: 0.0
Sample "sum_seg_osrm_distance" follows a Gaussian Distribution

C:\Users\india\anaconda3\lib\site-packages\scipy\stats\_morestats.py:1800: UserWarning: p-value may not be accurate for N > 500
0.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

As the value is zero we need to perform a Non-Parametric test (i.e. Kruskal-Wallis Test) instead of Parametric Test

In [163]:
```
# Doing Kruskal-Wallis Test just to test the similarity between the distributions of the given samples
kr_test_stat_2, kr_p_val_2 = kruskal(tripwise_copy['osrm_distance'],tripwise_copy['sum_seg_osrm_distance'])
if kr_p_val_2 < 0.05:
    print("The Distributions of the given samples are NOT EQUAL")
else:
    print("The Distributions of the given samples are EQUAL")
```

```
The Distributions of the given samples are NOT EQUAL
```

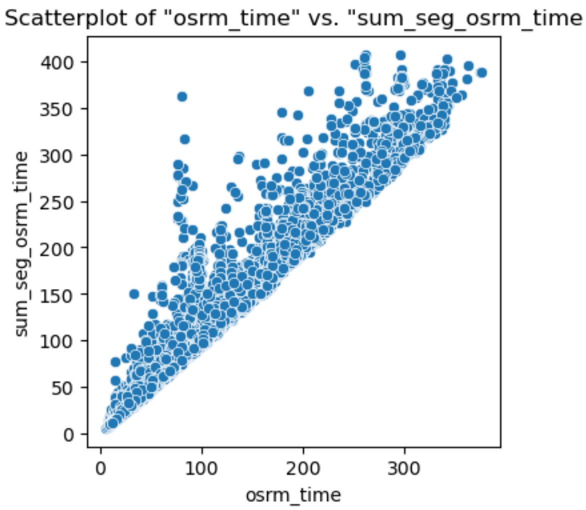
E. Hypothesis Testing to compare the difference between 'osrm_time' aggregated value and 'sum_seg_osrm_time'

Null Hypothesis (H0): The means of both the independent samples i.e. 'osrm_time' aggregated value and 'sum_seg_osrm_time' are equal i.e. distributions of both samples are equal.

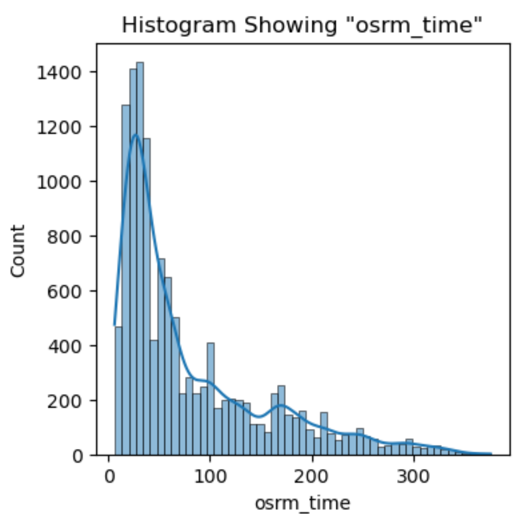
Alternate Hypothesis(Ha): The means of both the independent samples i.e.'osrm_time' aggregated value and 'sum_seg_osrm_time' are not equal i.e. distributions of both samples are not equal.

Assumed Significance level (alpha): 0.05 This means that if the p-value of the tests is less than the assumed significance level, we will REJECT the Null Hypothesis and vice-versa.

In [164]:
```
# Plot showing 'osrm_time' vs. 'sum_seg_osrm_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.scatterplot(data = tripwise_copy, x = 'osrm_time', y = 'sum_seg_osrm_time', ax = ax).set(title = 'Scatterplot of "osrm_time"
plt.show()
```



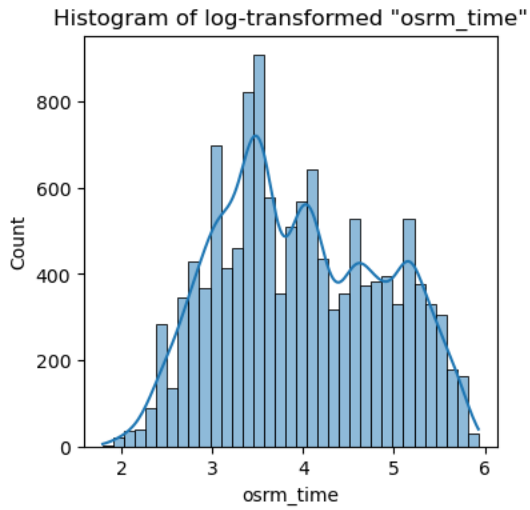
In [165]:
```
# Histogram of "osrm_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'osrm_time', ax = ax, kde = True).set(title = 'Histogram Showing "osrm_time"')
plt.show()
```



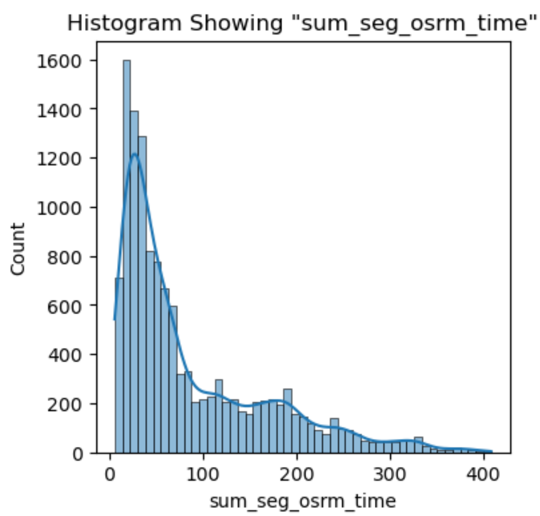
Histogram of feature "osrm_time" is once again right skewed.

Plotting the histogram of the log normal transformation to check if the its Gaussian.

In [168]:
```
# Histogram of log-transformed feature 'osrm_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['osrm_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "osrm_time'
plt.show()
```



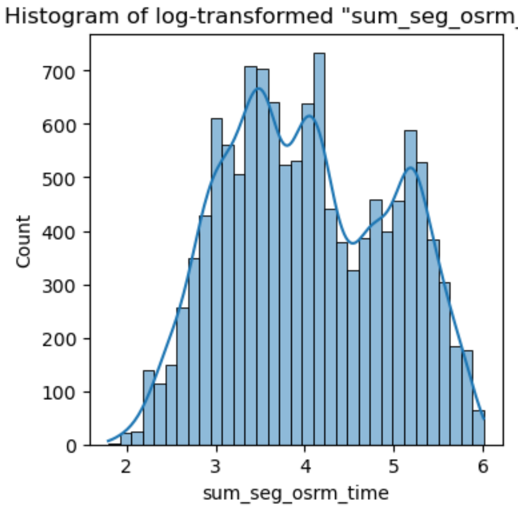
In [169]:
```
# Histogram of "sum_seg_osrm_time"
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(data = tripwise_copy, x = 'sum_seg_osrm_time', ax = ax, kde = True).set(title = 'Histogram Showing "sum_seg_osrm_tim
plt.show()
```



Histogram of feature "sum_seg_osrm_time" is once again right skewed.

Plotting the histogram of the log normal transformation to check if the its Gaussian.

In [171]:
```
# Histogram of log-transformed feature 'sum_seg_osrm_time'
fig, ax = plt.subplots(figsize = (4,4))
sns.histplot(x = (np.log(tripwise_copy['sum_seg_osrm_time'])), kde = True, ax = ax).set(title = 'Histogram of log-transformed "su
plt.show()
```



The log-transformed plots for both the features i.e. 'osrm_time' and 'sum_seg_osrm_time' does not seem Gaussian in quick view, as there seems another minor peaks in the histogram plotted in above images. Thus a statistical test is the best option to test both these cases. Here first we will try to do a 2-Sample T-Test. But for doing this, we need to check the assumptions of a T-Test.

Case-relevant Assumptions of 2-Sample T-Test

1. Data in each group should be NORMALLY Distributed.
2. Data values should be independent.
3. The Variances of the two independent groups should be EQUAL.

Thus we will need to do NORMALITY Test as well as EQUI-VARIANCE Test. For Normality, we will do SHAPIRO-WILK Test.

For Equi-Variance, we will do LEVENE'S Test.

SHAPIRO-WILK'S TEST:

For Shapiro-Test:

Null Hypothesis: Sample follows a gaussian distribution.

Alternate Hypothesis: Sample does not follow a gaussian distribution.

In [173]:
```
# Shapiro-Wilk's Test for 'osrm_time'
test_stat_sh9, p_val_9 = shapiro(np.log(tripwise_copy['osrm_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh9}, P-value of Shapiro Test: {p_val_9}")
if p_val_9 < 0.05:
    print('Sample "osrm_time" follows a Gaussian Distribution')
else:
    print('Sample "osrm_time" does not follow a Gaussian Distribution')
# Shapiro-Wilk's Test for 'sum_seg_osrm_time'
test_stat_sh10, p_val_10 = shapiro(np.log(tripwise_copy['sum_seg_osrm_time']))
print(f"Test Statistics of Shapiro Test: {test_stat_sh10}, P-value of Shapiro Test: {p_val_10}")
if p_val_10 < 0.05:
    print('Sample "sum_seg_osrm_time" follows a Gaussian Distribution')
else:
    print('Sample "sum_seg_osrm_time" does not follow a Gaussian Distribution')
```

```
Test Statistics of Shapiro Test: 0.9736400246620178, P-value of Shapiro Test: 5.4510510262235384e-43
Sample "osrm_time" follows a Gaussian Distribution
Test Statistics of Shapiro Test: 0.9738816022872925, P-value of Shapiro Test: 7.66510259985675e-43
Sample "sum_seg_osrm_time" follows a Gaussian Distribution
```

```
C:\Users\india\anaconda3\lib\site-packages\scipy\stats\_morestats.py:1800: UserWarning: p-value may not be accurate for N > 500
0.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

LEVENE'S TEST:

For Levene's Test:

Null Hypothesis (H0): The variance of features 'osrm_time' and 'sum_seg_osrm_time' will be equal.

Alternate Hypothesis (Ha): The variance of features will not be equal.

In [174]:
```
# Levene's test for checking Equi-Variance of features mentioned
test_stat_l4, p_val_l4 = levene(np.log(tripwise_copy['osrm_time']), np.log(tripwise_copy['sum_seg_osrm_time']), center = 'median'
print(f"Test Statistic for Levene's Test: {test_stat_l4}, P-value for Levene's Test: {p_val_l4}")
if p_val_l4 < 0.05:
    print('Both the samples do not have equal variance')
else:
    print('Both the samples have equal variance')
```

```
Test Statistic for Levene's Test: 15.708312566529548, P-value for Levene's Test: 7.41003845574868e-05
Both the samples do not have equal variance
```

The Levene's test show that the samples do not have equal variance.

So the assumptions of a 2-Sample T-Test does not follow here.

So a Non-Parametric Test, Kruskal-Wallis Test, is to be used here.

In [176]:
```
# Doing Kruskal-Wallis Test just to test the similarity between the distributions of the given samples
kr_test_stat_3, kr_p_val_3 = kruskal(tripwise_copy['osrm_time'],tripwise_copy['sum_seg_osrm_time'])
if kr_p_val_3 < 0.05:
    print("The Distributions of the given samples are NOT EQUAL")
else:
    print("The Distributions of the given samples are EQUAL")
```

```
The Distributions of the given samples are NOT EQUAL
```

Observations/Insights from the Hypothesis Testing

1) The distributions of the features 'start_scan_to_end_scan' and 'od_time_diff_minutes' are EQUAL.

2) The distributions of features 'actual_time' aggregated value and 'osrm_time' aggregated value are NOT EQUAL.

3) The distributions of features 'actual_time' aggregated value and segment actual time aggregated value are EQUAL.

4) The distributions of features 'osrm_distance' aggregated value and segment osrm distance aggregated value are NOT EQUAL.

5) The distributions of both the samples i.e. of features 'osrm_time' aggregated value and segment osrm time aggregated value are NOT EQUAL.

Recommendations

1) There is a significant difference in the actual cumulative time taken to delivery and the shortest cumulative time for delivery .It is recommended that the accuracy of the open source routing engine calculator needs to be improved so that the estimated time for delivery can be accurately predicted.

2) A difference is observed in the distance calculated by the open source routing engine calculator and the aggregate of the subset of package delivery.To solve this appropriate mechanisms can be applied so as to make the segment distance nearly equal to the estimated distance.

3) There is also a significant difference in the aggregated estimated time for package delivery and the aggregated estimated segment time for package delivery. This indicates lapse in the calculation of aggregated time of different segments of a trip and it needs to be accurately predicted.

4) Appropriate warehousing facilities need to be planned as per the demand/most frequent source and destination cities, localities and states as well.

In [ ]: