Delhivery_feature_engineering_solution

September 3, 2022

1 Business Case: Delhivery - Feature Engineering

1.1 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.

1.2 Dataset column Profiling:

- *data* - tells whether the data is testing or training data
- *trip_creation_time* – Timestamp of trip creation
- *route_schedule_uuid* – Unique Id for a particular route schedule
- *route_type* – Transportation type
  - FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
  - Carting: Handling system consisting of small vehicles (carts)
- *trip_uuid* - Unique ID given to a particular trip (A trip may include different source and destination centers)
- *source_center* - Source ID of trip origin
- *source_name* - Source Name of trip origin
- *destination_cente* – Destination ID
- *destination_name* – Destination Name
- *od_start_time* – Trip start time
- *od_end_time* – Trip end time
- *start_scan_to_end_scan* – Time taken to deliver from source to destination
- *is_cutoff* – Unknown field
- *cutoff_factor* – Unknown field
- *cutoff_timestamp* – Unknown field
- *actual_distance_to_destination* – Distance in Kms between source and destination warehouse
- *actual_time* – Actual time taken to complete the delivery (Cumulative)
- *osrm_time* – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
- *osrm_distance* – An open-source routing engine which computes the shortest path between

points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)

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

1.3 Business problem

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

1.4 Our approach to the businesss problem.

The focus of this case-study is to use various feature engineering techniques to generate a dataset which can be used by the Data Science team at Delhivery to build forecasting models. We begin with preliminary EDA to analyze the structure of the given data-set, understand relationship between the given features, spot missing values and outliers, perform uni-variate analysis to understand distributions for continuous features and count distribution for categorical features, and perform bi-variate/multi-variate analysis to understand relations among various features. We then perform aggregation; We first aggregate at (trip, source, destination) level. We then further aggregate at (trip) level and use several visual/hypothesis tests to determine relationship between various aggregated features and also remove outliers wherever necessary. We then use various feature engineering techniques on the aggregated data-set to create new features. Finally, we examine the dataset to further generate business insights and recommendations.

Basic roadmap 1. Basic data cleaning and exploration: - Handle missing values in the data. - Analyze the structure of the data. - Try merging the rows using the hint mentioned above. 2. Build some features to prepare the data for actual analysis. Extract features from the below fields: - Destination Name: Split and extract features out of destination. City-place-code (State) - Source Name: Split and extract features out of destination. City-place-code (State) - Trip_creation_time: Extract features like month, year and day etc 3. In-depth analysis and feature engineering: - Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required - Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check. - Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid) - Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid) - Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid) - Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid) - Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis - Handle the outliers using the IQR method. - Do one-hot encoding of categorical variables (like route_type) - Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

2 Solution

2.1 Read data and understand its structure

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#read dataset
data_original = pd.read_csv('data/delhivery_data.csv')
df = data_original.copy()
df.head()
```

[1]:
```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

                                 route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3…    Carting

                 trip_uuid source_center                 source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center               destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
```

```
                 od_start_time  ...           cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  ...        2018-09-20 04:27:55
1  2018-09-20 03:21:32.418600  ...        2018-09-20 04:17:55
2  2018-09-20 03:21:32.418600  ...  2018-09-20 04:01:19.505586
3  2018-09-20 03:21:32.418600  ...        2018-09-20 03:39:57
4  2018-09-20 03:21:32.418600  ...        2018-09-20 03:33:55

   actual_distance_to_destination  actual_time  osrm_time osrm_distance  \
0                       10.435660         14.0       11.0       11.9653
1                       18.936842         24.0       20.0       21.7243
2                       27.637279         40.0       28.0       32.5395
3                       36.118028         62.0       40.0       45.5620
4                       39.386040         68.0       44.0       54.2181

     factor  segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0  1.272727                 14.0               11.0                11.9653
1  1.200000                 10.0                9.0                 9.7590
2  1.428571                 16.0                7.0                10.8152
3  1.550000                 21.0               12.0                13.0224
4  1.545455                  6.0                5.0                 3.9153

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

[2]:
```
df.shape
```

[2]:
```
(144867, 24)
```

[3]:
```
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
```

[4]:
```
#check for missing values
df.isna().sum()
```

[4]:
```
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
```

[5]:
```python
df.nunique()
```

[5]:
```
data                                 2
trip_creation_time               14817
route_schedule_uuid               1504
route_type                           2
trip_uuid                        14817
source_center                     1508
source_name                       1498
destination_center                1481
destination_name                  1468
od_start_time                    26369
od_end_time                      26369
start_scan_to_end_scan            1915
is_cutoff                            2
cutoff_factor                      501
cutoff_timestamp                 93180
actual_distance_to_destination  144515
actual_time                       3182
osrm_time                         1531
osrm_distance                   138046
factor                           45641
segment_actual_time                747
segment_osrm_time                  214
segment_osrm_distance           113799
segment_factor                    5675
dtype: int64
```

[6]:
```python
#check possible values for the categorical variables
for col in ['data', 'route_type', 'is_cutoff']:
    print(df[col].value_counts())
    print('\n')
```

```
training    104858
test         40009
Name: data, dtype: int64



FTL        99660
Carting    45207
Name: route_type, dtype: int64



True     118749
```

```
False     26118
Name: is_cutoff, dtype: int64
```

[7]:
```python
df.describe()
```

[7]:

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

	actual_time	osrm_time	osrm_distance	factor
count	144867.000000	144867.000000	144867.000000	144867.000000
mean	416.927527	213.868272	284.771297	2.120107
std	598.103621	308.011085	421.119294	1.715421
min	9.000000	6.000000	9.008200	0.144000
25%	51.000000	27.000000	29.914700	1.604264
50%	132.000000	64.000000	78.525800	1.857143
75%	513.000000	257.000000	343.193250	2.213483
max	4532.000000	1686.000000	2326.199100	77.387097

	segment_actual_time	segment_osrm_time	segment_osrm_distance
count	144867.000000	144867.000000	144867.00000
mean	36.196111	18.507548	22.82902
std	53.571158	14.775960	17.86066
min	-244.000000	0.000000	0.00000
25%	20.000000	11.000000	12.07010
50%	29.000000	17.000000	23.51300
75%	40.000000	22.000000	27.81325
max	3051.000000	1611.000000	2191.40370

	segment_factor
count	144867.000000
mean	2.218368
std	4.847530
min	-23.444444
25%	1.347826
50%	1.684211
75%	2.250000
max	574.250000

Observations

1. The given dataset has 144867 rows and 24 columns.
2. 'start_scan_to_end_scan', 'cutoff_factor', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance', and 'segment_factor' are continuous variables.
3. 'trip_creation_time', 'od_start_time', 'od_end_time', and 'cutoff_timestamp' are continuous variables representing datetime. We will convert them from object to datetime64 dtype in the next section.
4. 'data', 'is_cutoff', and 'route_type' are dichotomous categorical variables.
5. 'source_center', 'source_name', 'destination_center', and 'destination_name' are categorical variables.
6. 'route_schedule_uuid' is a categorical variable representing route schedule for each trip. Similarly 'trip_uuid' is a categorical variable representing a trip. Once we aggregate data at trip level, it will become an identifier column.
7. 'source_name' and 'destination_name' columns contain 293 and 261 missing values respectively. We treat them in the section on missing value treatment.
8. There are 21 rows where the value of 'segment_actual_time' is negative. Ideally, we expect time to be always a positive quantity. However, in the absence of any domain knowledge, we cannot conlude if negative values are indeed errors or they can be valid values in certain conditions. We will anyway perform most of the analysis at aggregate level, so we ignore these negative values.

2.2 Converting relevant columns to datetime64

[8]:
```
col_convertto_datetime = ['trip_creation_time', 'od_start_time', 'od_end_time',␣
↪'cutoff_timestamp']
for col in col_convertto_datetime:
    df[col] = pd.to_datetime(df[col])

df[col_convertto_datetime].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 4 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   trip_creation_time  144867 non-null  datetime64[ns]
 1   od_start_time       144867 non-null  datetime64[ns]
 2   od_end_time         144867 non-null  datetime64[ns]
 3   cutoff_timestamp    144867 non-null  datetime64[ns]
dtypes: datetime64[ns](4)
memory usage: 4.4 MB
```

2.3 Dropping columns unrelated to the analysis.

'is_cutoff' and 'cutoff_factor' are unknown fields and we drop them from the further analysis. 'cuttoff_timestamp', on the other hand, seems useful to obtain ordering within each trip group. We will however not include it in the aggregated data.

[9]:
```
df.drop(labels=['is_cutoff', 'cutoff_factor'], axis=1, inplace=True)
```

2.4 Identify relationships between various columns

In this section, we attempt to understand cardinality relationships between various columns. This will help us select appropriate function during agggregation.

[10]:
```
#confirm many:one relation between trip_uuid and trip_creation_time
df.groupby('trip_uuid')['trip_creation_time'].nunique().value_counts()
```

[10]:
```
1    14817
Name: trip_creation_time, dtype: int64
```

[11]:
```
#confirm many:one relation between trip_uuid and data attribute
df.groupby('trip_uuid')['data'].nunique().value_counts()
```

[11]:
```
1    14817
Name: data, dtype: int64
```

[12]:
```
#confirm many:one relation between trip_uuid and route_schedule_uuid
print(df.groupby('trip_uuid')['route_schedule_uuid'].nunique().value_counts())
```

```
1    14817
Name: route_schedule_uuid, dtype: int64
```

[13]:
```
#relationship between route_type and route_schedule_uuid
df.groupby('route_schedule_uuid')['route_type'].nunique().value_counts()
```

[13]:
```
1    1504
Name: route_type, dtype: int64
```

[14]:
```
#confirm if factor is ratio of actual_time and osrm_time (we round both ratios␣
↪to two decimal places to ignore minute differences)
df.apply(lambda row: np.round(row['factor'],2) == np.round(row['actual_time'] /␣
↪row['osrm_time'],2), axis=1).value_counts()
```

[14]:
```
True    144867
dtype: int64
```

[15]:
```
#confirm if segment_factor is ratio of segment_actual_time and segment_osrm_time
```

```
print(df[df['segment_osrm_time'] != 0].apply(lambda row: np.
↪round(row['segment_factor'],2) == np.round(row['segment_actual_time'] /␣
↪row['segment_osrm_time'],2), axis=1).value_counts())
print('\n')
#check segment_factor when segment_osrm_time = 0
print(df[df['segment_osrm_time'] == 0]['segment_factor'].value_counts())
```

```
True    142520
dtype: int64


-1.0    2347
Name: segment_factor, dtype: int64
```

[16]:
```
#confirm if start_scan_to_end_scan is difference of od_start_time and␣
↪od_end_time
df.apply(lambda row: np.floor(row['start_scan_to_end_scan']) == np.
↪floor((row['od_end_time'] - row['od_start_time']).total_seconds()/60),␣
↪axis=1).value_counts()
```

[16]:
```
True    144867
dtype: int64
```

[17]:
```
#understand actual_time variable and its relation with segment_actual_time
df[df['trip_uuid'] == 'trip-153671042288605164'].sort_values(['od_start_time',␣
↪'cutoff_timestamp'], ascending=[True, True])[['trip_uuid', 'source_name',␣
↪'destination_name', 'od_start_time', 'od_end_time',␣
↪'start_scan_to_end_scan', 'actual_time', 'segment_actual_time', 'osrm_time',␣
↪'segment_osrm_time', 'osrm_distance', 'segment_osrm_distance',␣
↪'actual_distance_to_destination']]
```

[17]:
```
                    trip_uuid                          source_name  \
66267  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66266  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66265  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66264  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66263  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66262  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66270  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)
66269  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)
66268  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)

                        destination_name              od_start_time  \
66267  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66266  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66265  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66264  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
```

```
66263  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66262  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66270  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591
66269  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591
66268  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591

                    od_end_time  start_scan_to_end_scan  actual_time  \
66267 2018-09-12 02:03:09.655591                   122.0         96.0
66266 2018-09-12 02:03:09.655591                   122.0         76.0
66265 2018-09-12 02:03:09.655591                   122.0         53.0
66264 2018-09-12 02:03:09.655591                   122.0         40.0
66263 2018-09-12 02:03:09.655591                   122.0         24.0
66262 2018-09-12 02:03:09.655591                   122.0         14.0
66270 2018-09-12 03:01:59.598855                    58.0         47.0
66269 2018-09-12 03:01:59.598855                    58.0         31.0
66268 2018-09-12 03:01:59.598855                    58.0         18.0

       segment_actual_time  osrm_time  segment_osrm_time  osrm_distance  \
66267                 20.0       42.0                3.0        56.9116
66266                 22.0       39.0                8.0        52.1825
66265                 13.0       30.0                9.0        40.8990
66264                 16.0       21.0                5.0        28.6245
66263                 10.0       15.0                6.0        20.1431
66262                 14.0        8.0                8.0        10.3544
66270                 15.0       26.0                7.0        28.1994
66269                 13.0       19.0                9.0        21.2530
66268                 18.0       10.0               10.0        10.8633

       segment_osrm_distance  actual_distance_to_destination
66267                 3.8074                       48.542890
66266                11.2834                       45.182529
66265                12.2746                       36.472941
66264                 8.4814                       27.421490
66263                 9.7887                       19.445815
66262                10.3544                        9.832310
66270                 6.9464                       24.644021
66269                10.3898                       19.308677
66268                10.8633                        9.357635
```

In the query above, for a given trip, we order all the rows by 'od_start_time'(increasing) and 'cutoff_timestamp'(decreasing). This effectively gives us all the rows in the order in which different checkpoints are visited from trip-source to trip-destination. In this sorted data-set, we observe that 'actual_time' continuously decreases for a given {source, destination} combination. This means that actual_time is a cumulative variable which represents the actual 'remaining' distance to the destination, and therefore, is a decreasing variable.

[18]:
```
#confirm if osrm_time is also a 'decreasing' variable similar to the actual_time
grps = df.sort_values(['trip_uuid', 'od_start_time', 'cutoff_timestamp'],␣
↪ascending=[True, True, True]).groupby(['trip_uuid', 'source_center',␣
↪'destination_center']).groups

cnt=0
cnt2=0
for grp in grps.values():
    minsofar = 1000000
    for index in grp:
        row = df.iloc[index]
        val = row['osrm_time']
        if(val > minsofar):
            cnt += 1
        else:
            cnt2 += 1
            minsofar = val

print(f'osrm_time is descreasing for {cnt2} rows, increasing for {cnt} rows')
```

```
osrm_time is descreasing for 141543 rows, increasing for 3324 rows
```

Observations

1. All the rows for a given trip share the same 'trip_creation_time'.
2. All the rows for a given trip share the same 'data' value (either training or test but not mixed).
3. All the rows for a given trip share the same 'route_schedule_uuid'.
4. Each route is associated with one 'route_type'. Since each trip is associated with one 'route', by transitivity, all the rows for a given trip share the same 'route_type'.
5. 'factor' is the ratio of 'actual_time' and 'osrm_time'.
6. 'Segment_factor' is a ratio of 'segment_actual_time' and 'segment_osrm_time'. For rows where 'segment_osrm_time' is zero, 'Segment_factor' is set to -1.
7. 'start_scan_to_end_scan' is the difference of 'od_end_time' and 'od_start_time'
8. For all the rows for a given group of {trip, source, destination}, 'actual_time' is a cumulative variable which seems to represent the actual 'remaining' time to the destination. Thus it is a 'strictly decreasing' variable. 'osrm_time', on the other hand, is not 'strictly' decreasing variable inside each group. There are several instances where 'osrm_time' time at one checkpoint is greater than the same calculated at the previous checkpoint. This makes sense, as unlike 'actual_time', 'osrm_time' is a dynamic time calculated based on various parameters. We will revisit this point when we aggregate data.

2.5 Missing value treatment

'source_name' and 'destination_name' columns contain 293 and 261 missing values respectively. We observe that the valid values for these columns follow the this format: **<city><*place*><code>(<State>)**. In the feature engineering section, we will create new features by extracting individual components from these values. For convenience, we replace all the missing values with the text **NA_NA_NA(NA)**,so that individual features extracted for the missing value rows will have text 'NA' for each component.

[19]:
```
#replace missing values in both source_name and destination_name by text␣
↪NA_NA_NA(NA)
df.fillna('NA_NA_NA(NA)', inplace=True)
df.isna().sum()
```

[19]:
```
data                             0
trip_creation_time               0
route_schedule_uuid              0
route_type                       0
trip_uuid                        0
source_center                    0
source_name                      0
destination_center               0
destination_name                 0
od_start_time                    0
od_end_time                      0
start_scan_to_end_scan           0
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
```

2.6 Data aggregation

In the given data-set, each trip consists of one or more sub-trips. Each sub-trip is defined by a source and destination and in turn consists of several checkpoints. Thus each row represents a checkpoint. We aggregate data in two step process.

1. **First level aggregation** - We aggregate rows at sub-trip level. That is, we group rows by {'trip_uuid', 'source_center', 'destination_center'}. We then order rows within each group by 'od_start_time'(increasing) and 'actual_distance_to_destination'(decreasing). This effectively sorts each rows (checkpoints) within each trip in the order in which they are visited. We then aggregate columns as follow.

a. **group level columns** such as data, trip_creation_time, route_schedule_uuid, route_type, source_name, destination_name, od_start_time, od_end_time, start_scan_to_end_scan. All rows within a group shares the same value for these columns. We use **first** aggregate function.

b. **cumulative time and distance columns** such as actual_time, osrm_time, osrm_distance. We use the **first** function to take the largest value within the group.

c. **factor ratio** - we take the **first** value (as factor is ratio of actual_distance and osrm_distance)

d. **segment time and distance columns** - we use **sum** function to aggregate them.

2. **Final aggregation** - We take output from the first step and perform further aggregation. We group by {trip_uuid} and aggregate as follow.

a. **group level columns** such as data, trip_creation_time, route_schedule_uuid, route_type, source_name, destination_name. We use **first** aggregate function.

b. **source_name, source_center, od_start_time** - take **first** value.

c. **destination_name, destination_center, od_end_time** - take **last** value.

d. **other time and distance columns** such as actual_time_agg, segment_actual_time_agg, osrm_time_agg, segment_osrm_time_agg, actual_distance_to_destination_agg, osrm_distance_agg, segment_osrm_distance_agg. We aggregate values using **sum** function.

e. **factor and start_scan_to_end_scan columns** - we **recompute** them using the aggregated values.

[20]:
```
#Level 1 aggregation
df_agg_first_level = df.sort_values(['trip_uuid', 'od_start_time',␣
↪'actual_distance_to_destination'], ascending=[True, True, False]).
↪groupby(['trip_uuid', 'source_center', 'destination_center']).agg({
    'data': 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'source_name': 'first',
    'destination_name': 'first',
    'od_start_time': 'first',
    'od_end_time': 'first',
    'start_scan_to_end_scan': 'first',
    'actual_time': 'first',
    'segment_actual_time': 'sum',
    'osrm_time': 'first',
    'segment_osrm_time': 'sum',
    'factor': 'first',
    'actual_distance_to_destination': 'first',
    'osrm_distance': 'first',
    'segment_osrm_distance': 'sum'
```

```
}).rename(columns={
    'actual_time': 'actual_time_agg',
    'segment_actual_time' : 'segment_actual_time_agg',
    'osrm_time' : 'osrm_time_agg',
    'segment_osrm_time' : 'segment_osrm_time_agg',
    'actual_distance_to_destination' : 'actual_distance_to_destination_agg',
    'osrm_distance' : 'osrm_distance_agg',
    'segment_osrm_distance' : 'segment_osrm_distance_agg'
}).reset_index().sort_values(['trip_uuid', 'od_start_time'], ascending=[True,␣
↪True])
```

[21]:
```
#Level 2 aggregation
df_aggr2 = df_agg_first_level.groupby('trip_uuid').agg({
    'data': 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'source_center': 'first',
    'source_name': 'first',
    'destination_center': 'last',
    'destination_name': 'last',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'actual_time_agg': 'sum',
    'segment_actual_time_agg': 'sum',
    'osrm_time_agg': 'sum',
    'segment_osrm_time_agg': 'sum',
    'actual_distance_to_destination_agg': 'sum',
    'osrm_distance_agg': 'sum',
    'segment_osrm_distance_agg': 'sum'
}).reset_index()

#recompute start_scan_to_end_scan
df_aggr2['start_scan_to_end_scan'] = np.floor((df_aggr2['od_end_time'] -␣
↪df_aggr2['od_start_time']).dt.total_seconds()/60)

#recompute factor as the ratio of aggregated actual_time and odrm_time
df_aggr2['factor'] = np.round(df_aggr2['actual_time_agg'] /␣
↪df_aggr2['osrm_time_agg'],2)
```

2.6.1 Verification - confirm details for 'trip-153671042288605164' in the original and aggregated data-sets

'trip-153671042288605164' in the original data-set

[22]:

```
df[df['trip_uuid'] == 'trip-153671042288605164'][['trip_uuid', 'source_name',
 ↪'destination_name', 'od_start_time', 'od_end_time',
 ↪'start_scan_to_end_scan', 'actual_time', 'segment_actual_time', 'osrm_time',
 ↪'segment_osrm_time', 'osrm_distance', 'segment_osrm_distance',
 ↪'actual_distance_to_destination']]
```

[22]:

```
                    trip_uuid                        source_name  \
66262  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66263  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66264  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66265  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66266  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66267  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
66268  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)
66269  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)
66270  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)

                       destination_name              od_start_time  \
66262  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66263  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66264  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66265  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66266  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66267  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
66268  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591
66269  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591
66270  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591

                    od_end_time  start_scan_to_end_scan  actual_time  \
66262 2018-09-12 02:03:09.655591                   122.0         14.0
66263 2018-09-12 02:03:09.655591                   122.0         24.0
66264 2018-09-12 02:03:09.655591                   122.0         40.0
66265 2018-09-12 02:03:09.655591                   122.0         53.0
66266 2018-09-12 02:03:09.655591                   122.0         76.0
66267 2018-09-12 02:03:09.655591                   122.0         96.0
66268 2018-09-12 03:01:59.598855                    58.0         18.0
66269 2018-09-12 03:01:59.598855                    58.0         31.0
66270 2018-09-12 03:01:59.598855                    58.0         47.0

       segment_actual_time  osrm_time  segment_osrm_time  osrm_distance  \
66262                 14.0        8.0                8.0        10.3544
66263                 10.0       15.0                6.0        20.1431
66264                 16.0       21.0                5.0        28.6245
66265                 13.0       30.0                9.0        40.8990
66266                 22.0       39.0                8.0        52.1825
66267                 20.0       42.0                3.0        56.9116
66268                 18.0       10.0               10.0        10.8633
```

```
66269                  13.0      19.0               9.0        21.2530
66270                  15.0      26.0               7.0        28.1994

       segment_osrm_distance  actual_distance_to_destination
66262                10.3544                        9.832310
66263                 9.7887                       19.445815
66264                 8.4814                       27.421490
66265                12.2746                       36.472941
66266                11.2834                       45.182529
66267                 3.8074                       48.542890
66268                10.8633                        9.357635
66269                10.3898                       19.308677
66270                 6.9464                       24.644021
```

'trip-153671042288605164' in the First level aggregated data-set

[23]:
```python
df_agg_first_level[df_agg_first_level['trip_uuid'] ==
 ↪'trip-153671042288605164'][['trip_uuid', 'source_name', 'destination_name',
 ↪'od_start_time', 'od_end_time', 'start_scan_to_end_scan', 'actual_time_agg',
 ↪'segment_actual_time_agg', 'osrm_time_agg', 'segment_osrm_time_agg',
 ↪'osrm_distance_agg', 'segment_osrm_distance_agg',
 ↪'actual_distance_to_destination_agg']]
```

[23]:
```
                trip_uuid                       source_name  \
3  trip-153671042288605164      Tumkur_Veersagr_I (Karnataka)
2  trip-153671042288605164  Doddablpur_ChikaDPP_D (Karnataka)

                   destination_name              od_start_time  \
3  Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430
2  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591

                od_end_time  start_scan_to_end_scan  actual_time_agg  \
3 2018-09-12 02:03:09.655591                   122.0             96.0
2 2018-09-12 03:01:59.598855                    58.0             47.0

   segment_actual_time_agg  osrm_time_agg  segment_osrm_time_agg  \
3                     95.0           42.0                   39.0
2                     46.0           26.0                   26.0

   osrm_distance_agg  segment_osrm_distance_agg  \
3            56.9116                    55.9899
2            28.1994                    28.1995

   actual_distance_to_destination_agg
3                           48.542890
2                           24.644021
```

'trip-153671042288605164' in the final aggregated data-set

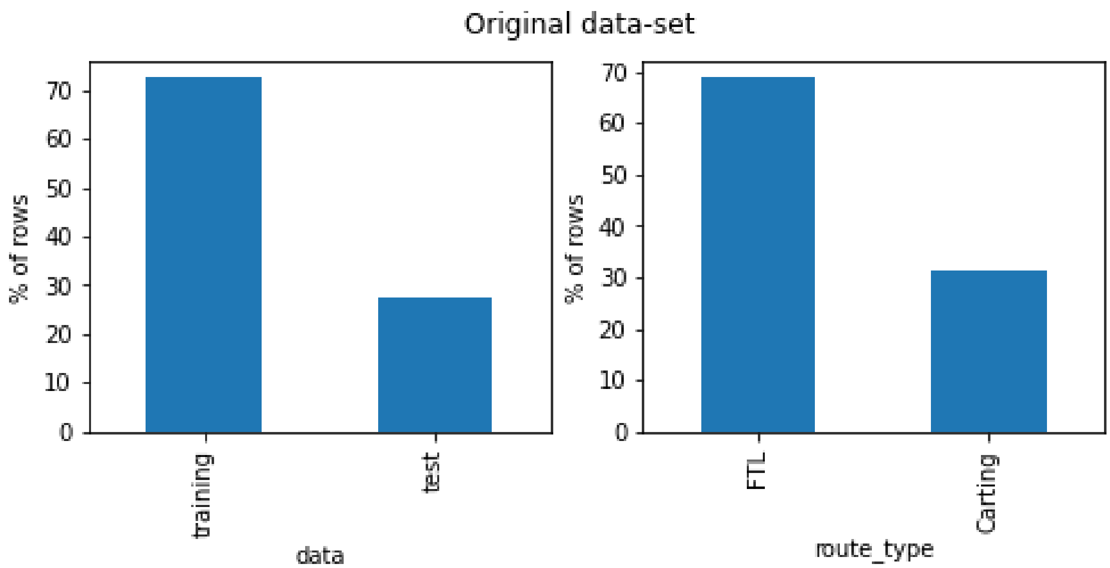
[24]:
```
df_aggr2[df_aggr2['trip_uuid'] == 'trip-153671042288605164'][['trip_uuid',␣
↪'source_name', 'destination_name', 'od_start_time', 'od_end_time',␣
↪'start_scan_to_end_scan', 'actual_time_agg', 'segment_actual_time_agg',␣
↪'osrm_time_agg', 'segment_osrm_time_agg', 'osrm_distance_agg',␣
↪'segment_osrm_distance_agg', 'actual_distance_to_destination_agg']]
```

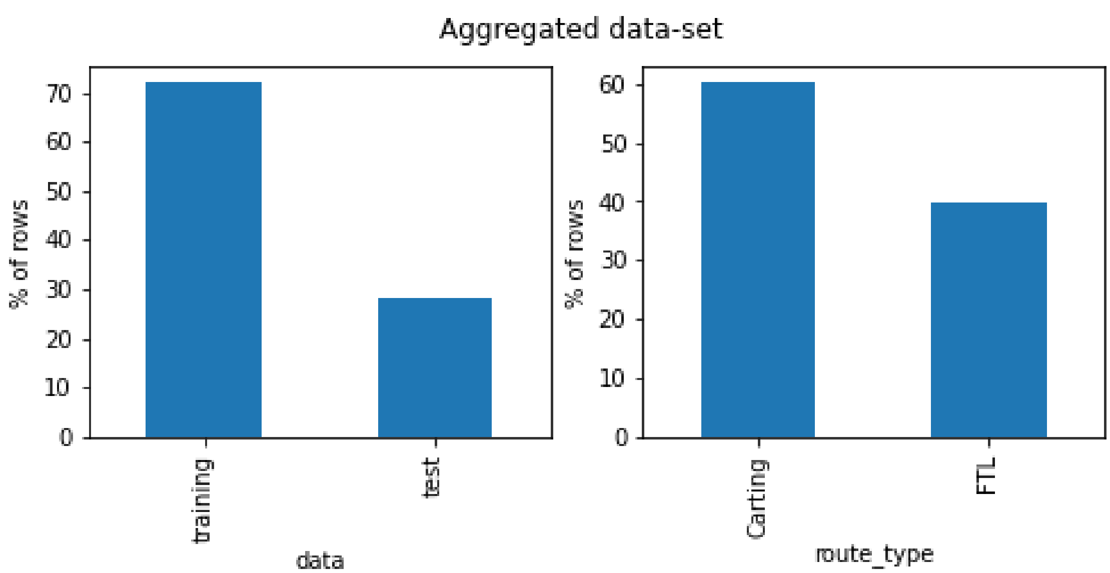
[24]:
```
                trip_uuid                    source_name  \
1  trip-153671042288605164  Tumkur_Veersagr_I (Karnataka)

                  destination_name              od_start_time  \
1  Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 00:00:22.886430

                od_end_time  start_scan_to_end_scan  actual_time_agg  \
1 2018-09-12 03:01:59.598855                   181.0            143.0

   segment_actual_time_agg  osrm_time_agg  segment_osrm_time_agg  \
1                    141.0           68.0                   65.0

   osrm_distance_agg  segment_osrm_distance_agg  \
1             85.111                    84.1894

   actual_distance_to_destination_agg
1                           73.186911
```

2.7 Uni-variate analysis

2.7.1 Categorical variables

[25]:
```
data = df
fig, ax = plt.subplots(1, 2, figsize=(8, 3))
fig.suptitle('Original data-set')
data['data'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
↪xlabel='data', ylabel='% of rows' ,ax = ax[0])
data['route_type'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
↪xlabel='route_type', ylabel='% of rows', ax = ax[1])

data = df_aggr2
fig, ax = plt.subplots(1, 2, figsize=(8, 3))
fig.suptitle('Aggregated data-set')
data['data'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
↪xlabel='data', ylabel='% of rows' ,ax = ax[0])
data['route_type'].value_counts(normalize=True).mul(100).plot(kind='bar',␣
↪xlabel='route_type', ylabel='% of rows', ax = ax[1])

plt.show()
```





Observations

1. The original data-set contains approximately 70% training data and 30% test data. After we aggregate data at trip level, the ratio of training/test data remains identical.
2. The original data-set contains close to 70% rows where route_type is 'FTL' and 30% rows where route_type is 'Carting'. After aggregating the data-set at trip level, the percent of 'Carting' rwos goes up to approximately 60% while that of 'FTL' comes down to around 40%. This implies that there are more trips with 'Carting' route_type than FTL, however, average number of checkpoints in 'FTL' trips are much higher than that in 'Carting' trips.

2.7.2 Continuous variables

Continuous variables in the original data-set

[26]:
```
#check statistical parameters for various countnuous variables.
def getnumcols(df):
    return [col for col in df.columns if (df[col].dtype.kind in 'iufc')]
cols_cont = getnumcols(df)

df[cols_cont].describe()
```

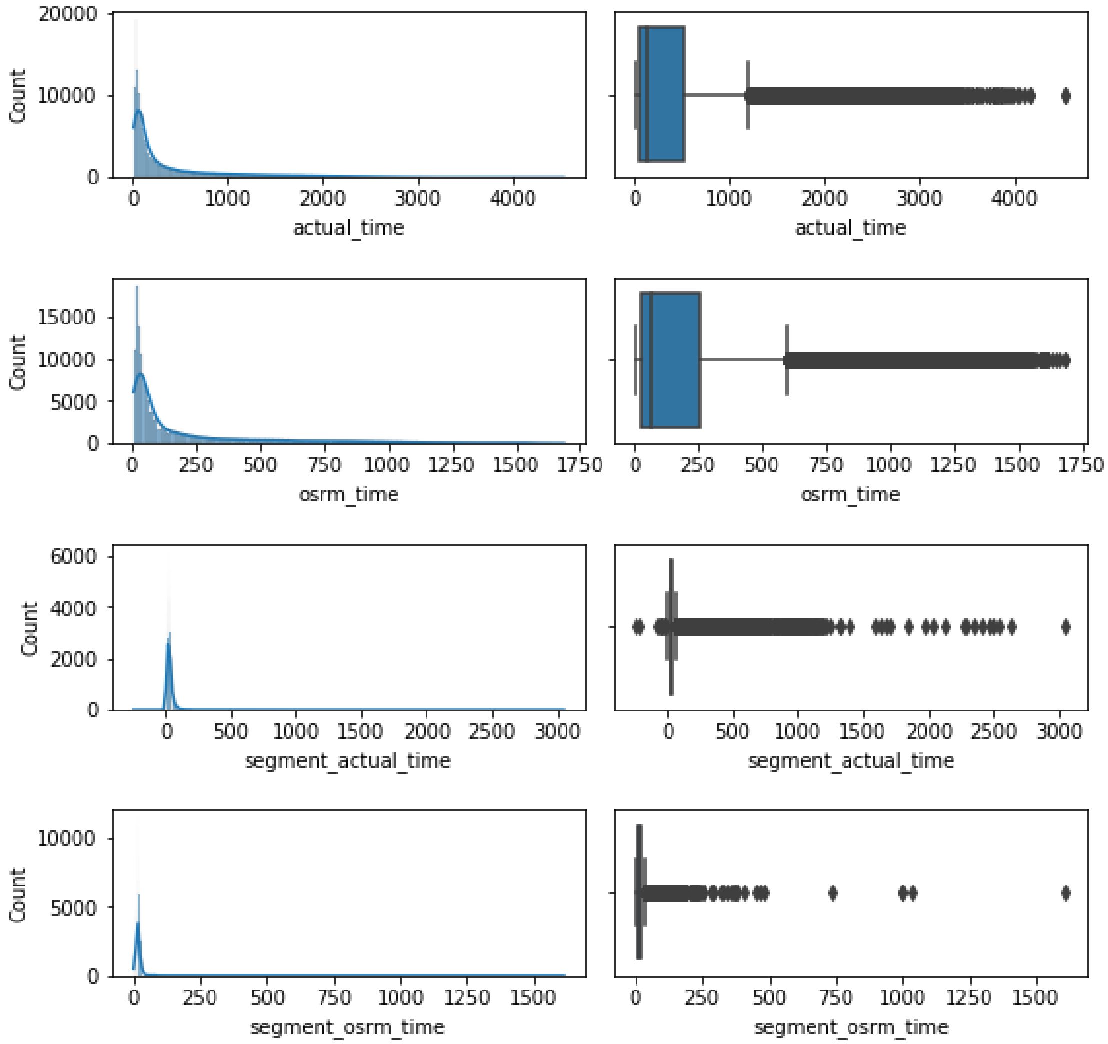
[26]:
```
       start_scan_to_end_scan  actual_distance_to_destination    actual_time  \
count           144867.000000                   144867.000000  144867.000000
mean               961.262986                      234.073372     416.927527
std               1037.012769                      344.990009     598.103621
min                 20.000000                        9.000045       9.000000
25%                161.000000                       23.355874      51.000000
50%                449.000000                       66.126571     132.000000
75%               1634.000000                      286.708875     513.000000
max               7898.000000                     1927.447705    4532.000000

           osrm_time  osrm_distance         factor  segment_actual_time  \
count  144867.000000  144867.000000  144867.000000        144867.000000
mean      213.868272     284.771297       2.120107            36.196111
std       308.011085     421.119294       1.715421            53.571158
min         6.000000       9.008200       0.144000          -244.000000
25%        27.000000      29.914700       1.604264            20.000000
50%        64.000000      78.525800       1.857143            29.000000
75%       257.000000     343.193250       2.213483            40.000000
max      1686.000000    2326.199100      77.387097          3051.000000

       segment_osrm_time  segment_osrm_distance  segment_factor
count      144867.000000           144867.00000   144867.000000
mean           18.507548               22.82902        2.218368
std            14.775960               17.86066        4.847530
min             0.000000                0.00000      -23.444444
25%            11.000000               12.07010        1.347826
50%            17.000000               23.51300        1.684211
75%            22.000000               27.81325        2.250000
max          1611.000000             2191.40370      574.250000
```

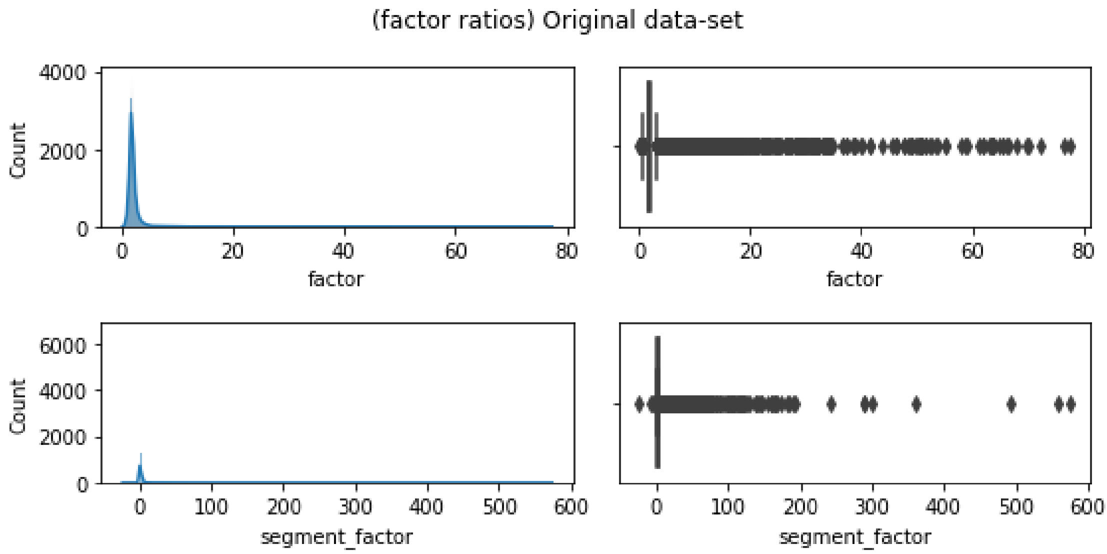
We now look at distributions of 'actual_time' (cumulative), 'osrm_time' (cumulative), 'segment_actual_time', 'segment_osrm_time', 'osrm_distance', 'segment_osrm_distance', 'actual_distance_to_destination'

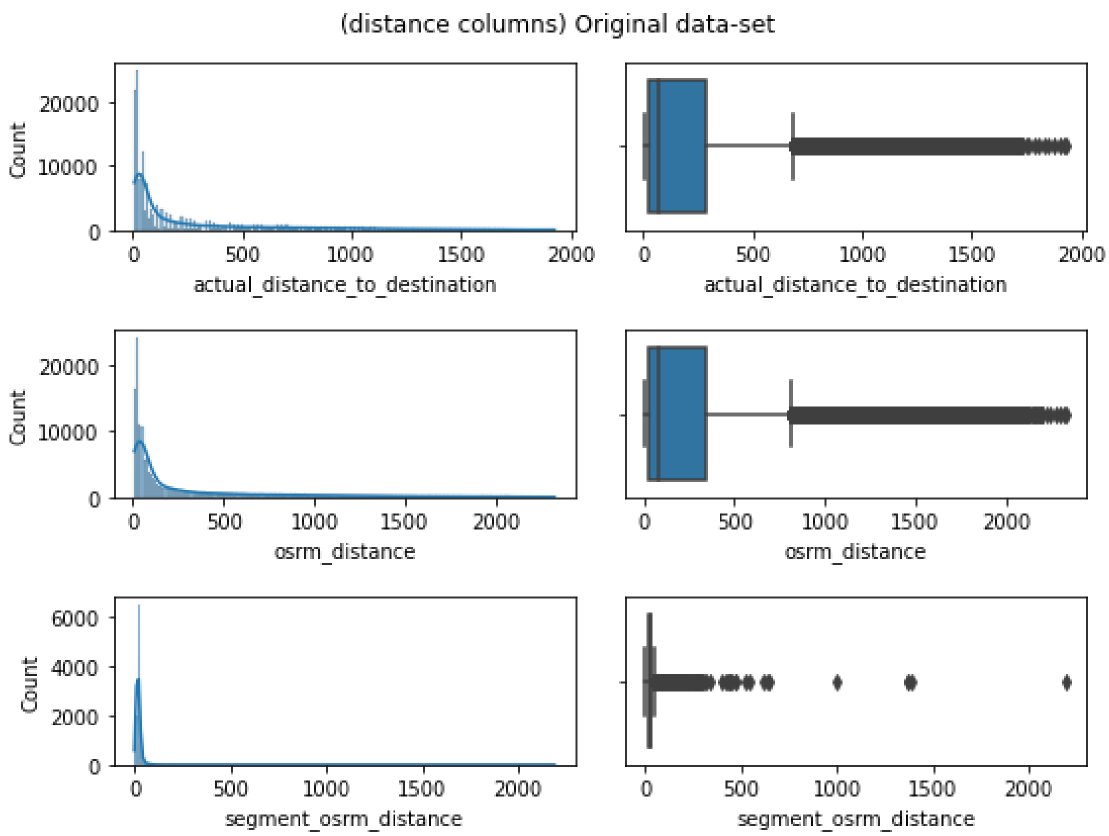
[27]:
```
#Analyze indepedent continuous columns
def plotdist(data, cols, title):
    n = len(cols)
    fig, ax = plt.subplots(n, 2, figsize=(8, 2*n))
```

```
    fig.suptitle(title)
    for i in range(len(cols)):
        col = cols[i]
        axis = ax[i] if(n > 1) else ax
        sns.histplot(data=data[col], kde=True, ax=axis[0])
        sns.boxplot(x=col, data=data, orient="horizontal", ax=axis[1])
    fig.tight_layout()
    plt.subplots_adjust(hspace=0.6)
    plt.show()

cols = ['actual_time', 'osrm_time', 'segment_actual_time', 'segment_osrm_time']
plotdist(df, cols, '(time columns) Original data-set')

cols = ['factor', 'segment_factor']
plotdist(df, cols, '(factor ratios) Original data-set')

cols = ['actual_distance_to_destination', 'osrm_distance',␣
 ↪'segment_osrm_distance']
plotdist(df, cols, '(distance columns) Original data-set')
```

(time columns) Original data-set







[28]:
```
#check skewness and kurtoises
```

```
pd.concat([df[cols_cont].skew(), df[cols_cont].kurt()], axis = 1).
↪rename(columns={0:'skew', 1:'kurtoises'})
```

[28]:

	skew	kurtoises
start_scan_to_end_scan	1.110624	-0.051149
actual_distance_to_destination	1.991105	3.397661
actual_time	2.068065	3.962915
osrm_time	2.045166	3.755795
osrm_distance	2.048236	3.734448
factor	17.490811	491.177641
segment_actual_time	16.827413	494.496398
segment_osrm_time	19.639634	1444.561243
segment_osrm_distance	26.585363	2317.333310
segment_factor	47.372766	4037.388713

Satistical parameters for continuous variables in the aggregated data-set

[29]:
```
#check statistical parameters for various countnuous variables.
cols_cont = [col for col in df_aggr2.columns if (df_aggr2[col].dtype.kind in␣
↪'iufc')]
df_aggr2[cols_cont].describe().T
```

[29]:

	count	mean	std
actual_time_agg	14817.0	353.003172	557.325096
segment_actual_time_agg	14817.0	353.892286	556.247965
osrm_time_agg	14817.0	161.138355	271.062452
segment_osrm_time_agg	14817.0	180.949787	314.542047
actual_distance_to_destination_agg	14817.0	164.682943	305.561548
osrm_distance_agg	14817.0	204.249399	370.183781
segment_osrm_distance_agg	14817.0	223.201161	416.628374
start_scan_to_end_scan	14817.0	546.960991	668.656823
factor	14817.0	2.611391	2.174422

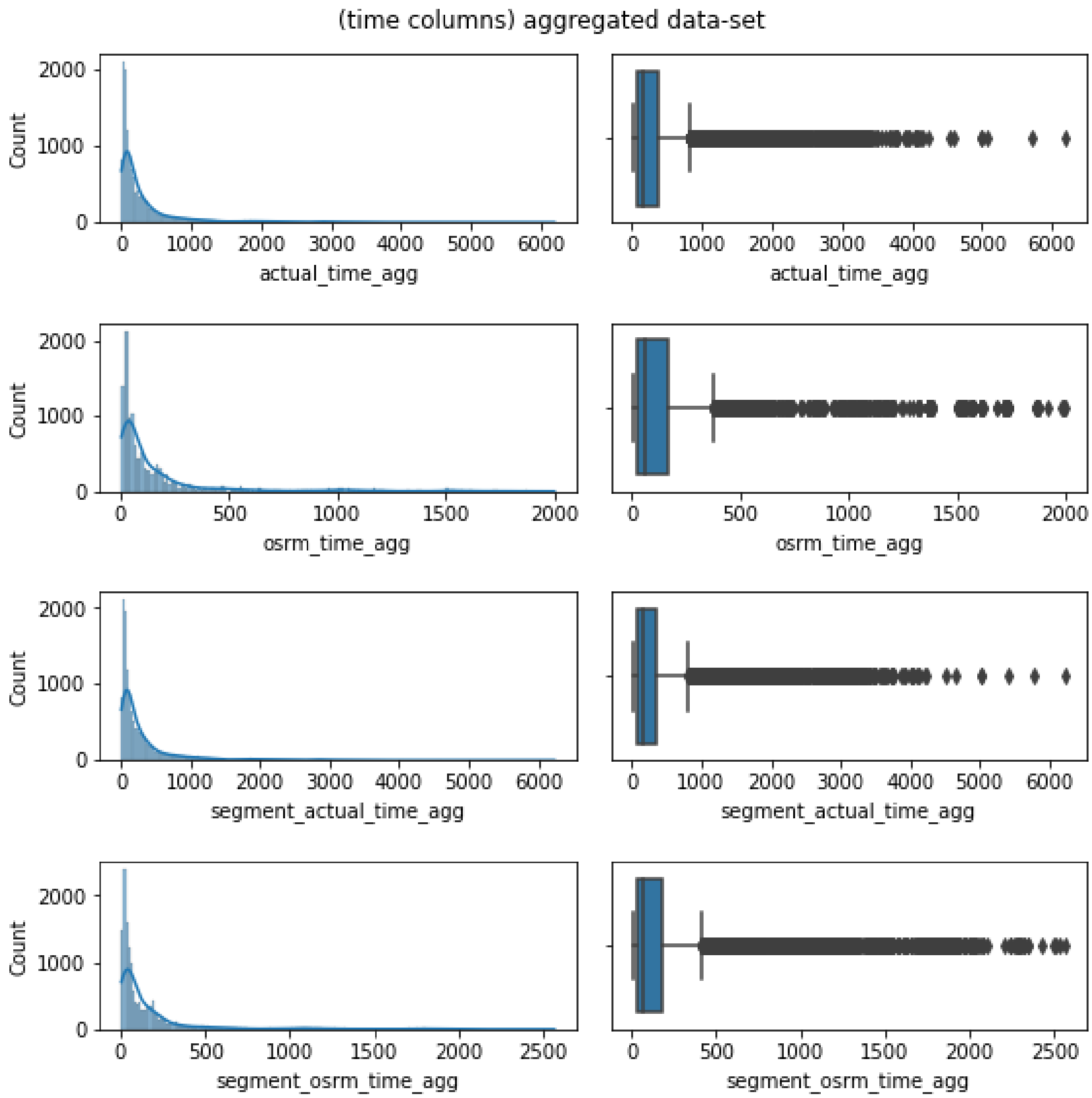
	min	25%	50%
actual_time_agg	9.000000	66.00000	145.000000
segment_actual_time_agg	9.000000	66.00000	147.000000
osrm_time_agg	6.000000	29.00000	60.000000
segment_osrm_time_agg	6.000000	31.00000	65.000000
actual_distance_to_destination_agg	9.002461	22.86003	48.499937
osrm_distance_agg	9.072900	30.84630	65.606900
segment_osrm_distance_agg	9.072900	32.65450	70.154400
start_scan_to_end_scan	23.000000	151.00000	288.000000
factor	0.270000	1.74000	2.090000

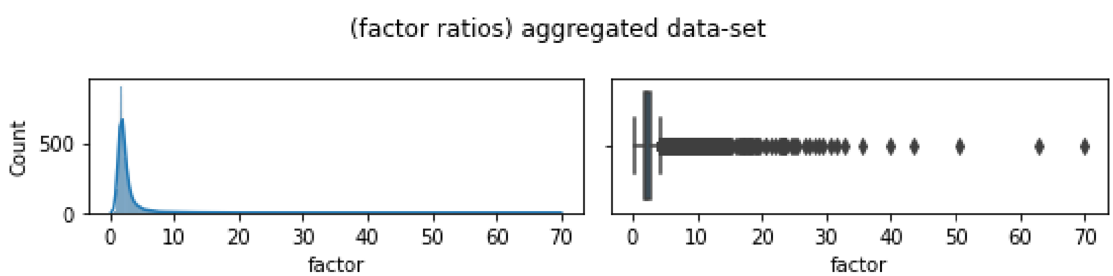
	75%	max
actual_time_agg	367.000000	6187.000000
segment_actual_time_agg	367.000000	6230.000000

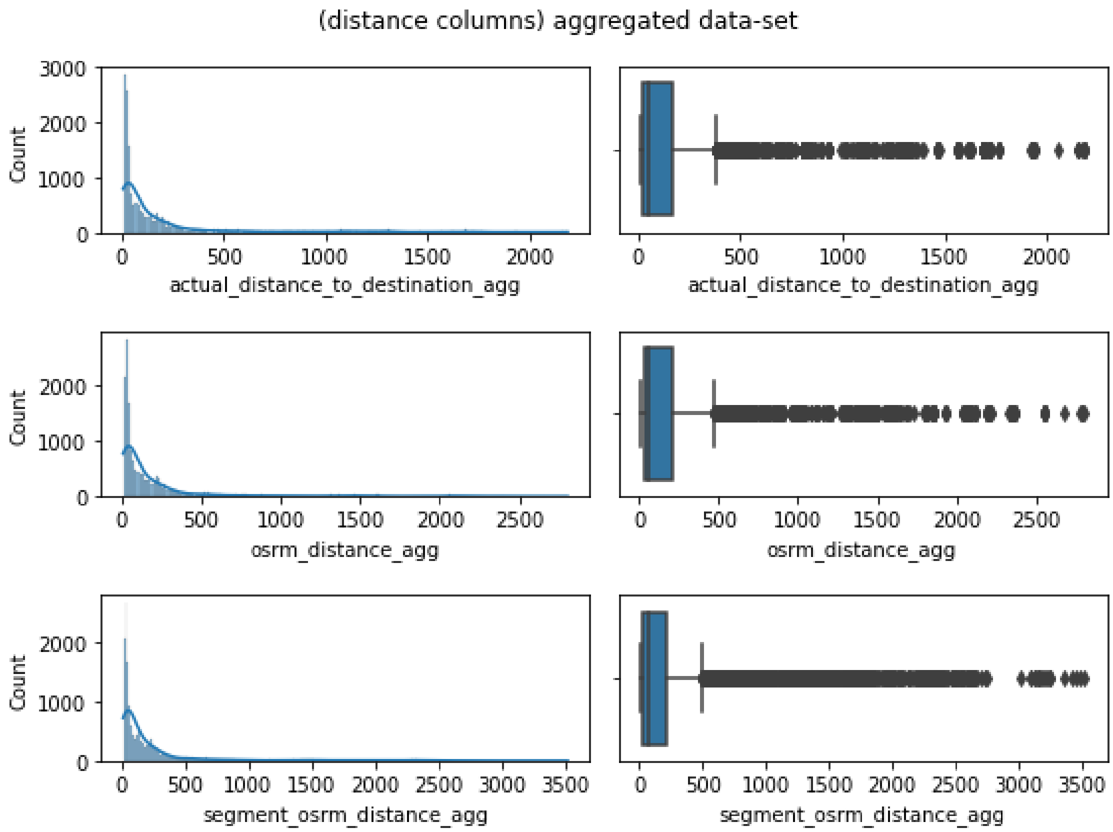
```
osrm_time_agg                      168.000000  1998.000000
segment_osrm_time_agg              185.000000  2564.000000
actual_distance_to_destination_agg 164.853324  2187.483994
osrm_distance_agg                  207.521100  2804.709500
segment_osrm_distance_agg          218.802400  3523.632400
start_scan_to_end_scan             673.000000  7898.000000
factor                               2.720000    70.000000
```

[30]:

```
#plot distributions of variables in the aggregated data-set
cols = ['actual_time_agg', 'osrm_time_agg', 'segment_actual_time_agg',␣
↪'segment_osrm_time_agg']
plotdist(df_aggr2, cols, '(time columns) aggregated data-set')

cols = ['factor']
plotdist(df_aggr2, cols, '(factor ratios) aggregated data-set')

cols = ['actual_distance_to_destination_agg', 'osrm_distance_agg',␣
↪'segment_osrm_distance_agg']
plotdist(df_aggr2, cols, '(distance columns) aggregated data-set')
```


(time columns) aggregated data-set
Count
actual_time_agg
actual_time_agg
Count
osrm_time_agg
osrm_time_agg
Count
segment_actual_time_agg
segment_actual_time_agg
Count
segment_osrm_time_agg
segment_osrm_time_agg


(factor ratios) aggregated data-set
Count
factor
factor



[31]:
```
#check skewness and kurtoises
pd.concat([df_aggr2[cols_cont].skew(), df_aggr2[cols_cont].kurt()], axis = 1).
 ↪rename(columns={0:'skew', 1:'kurtoises'})
```

[31]:

	skew	kurtoises
actual_time_agg	3.361614	13.617328
segment_actual_time_agg	3.365921	13.798969
osrm_time_agg	3.448921	13.193313
segment_osrm_time_agg	3.597829	14.547415
actual_distance_to_destination_agg	3.558780	13.694012
osrm_distance_agg	3.550099	13.793937
segment_osrm_distance_agg	3.710119	15.357322
start_scan_to_end_scan	2.768892	10.188852
factor	9.111707	165.133653

Observations

1. In the **original data-set**, we observe that distribution of various time variables (actual, osrm, segment_actual, and segment_osrm), distance variables (osrm_distance, segment_odrm_distance, actual_distance_to_destination), and factor ratios (factor and segment_factor) are all **right skewed with several outliers.**

2. Similarly, in the **aggregated data-set** as well, all aggregated time, distance, and factor ratio columns are right skewed with several outliers.

Notes:

1. In the outliers analysis section, we further check the number of outliers and examine ways to treat them.

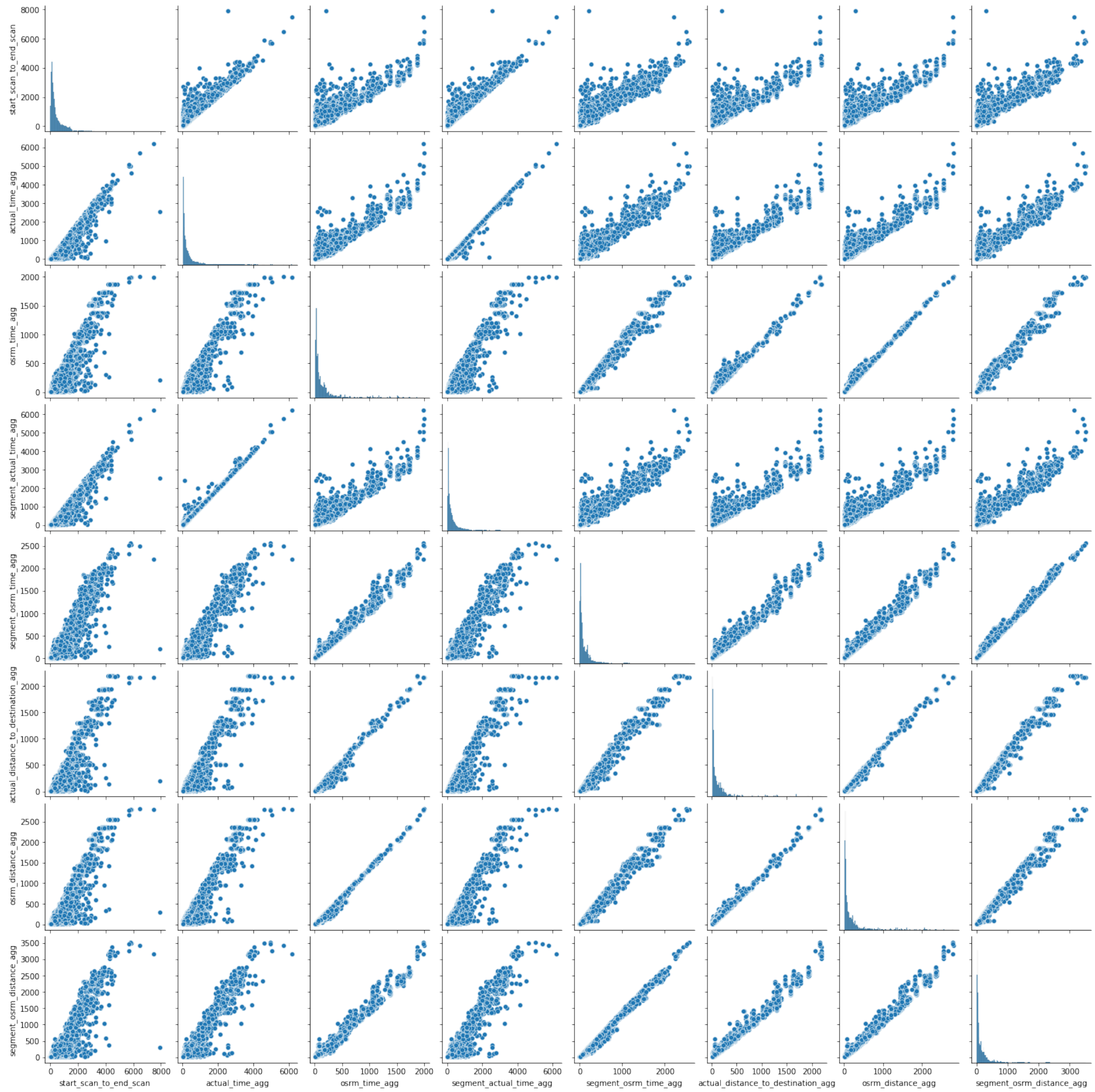
2. Our data values do not follow normal distribution. In the hypothesis testing section, we check various alternatives to deal with non-normality of the data.

2.8 Bi-variate analysis

We will perform bi-variate analysis only on the aggregated data-set.

2.8.1 pair plots and correlation (for continuous variables)

[32]:
```
#pair plot to view correlation among various time and distance variables.
df_time_dist_features = df_aggr2[['start_scan_to_end_scan', 'actual_time_agg',␣
 ↪'osrm_time_agg', 'segment_actual_time_agg', 'segment_osrm_time_agg',␣
 ↪'actual_distance_to_destination_agg', 'osrm_distance_agg',␣
 ↪'segment_osrm_distance_agg']]
sns.pairplot(df_time_dist_features)
plt.show()
```



[33]:
```
corr_df = df_time_dist_features.corr(method='pearson')
corr_df
```

[33]:

	start_scan_to_end_scan	actual_time_agg	\
start_scan_to_end_scan	1.000000	0.950480	
actual_time_agg	0.950480	1.000000	
osrm_time_agg	0.915690	0.960458	
segment_actual_time_agg	0.952656	0.997866	
segment_osrm_time_agg	0.907616	0.955157	
actual_distance_to_destination_agg	0.907187	0.956301	
osrm_distance_agg	0.912889	0.961056	
segment_osrm_distance_agg	0.907676	0.958266	

```
                                   osrm_time_agg  segment_actual_time_agg  \
start_scan_to_end_scan                  0.915690                 0.952656
actual_time_agg                         0.960458                 0.997866
osrm_time_agg                           1.000000                 0.957542
segment_actual_time_agg                 0.957542                 1.000000
segment_osrm_time_agg                   0.993268                 0.953039
actual_distance_to_destination_agg      0.993813                 0.953141
osrm_distance_agg                       0.997602                 0.958154
segment_osrm_distance_agg               0.991609                 0.956106

                                   segment_osrm_time_agg  \
start_scan_to_end_scan                          0.907616
actual_time_agg                                 0.955157
osrm_time_agg                                   0.993268
segment_actual_time_agg                         0.953039
segment_osrm_time_agg                           1.000000
actual_distance_to_destination_agg              0.987727
osrm_distance_agg                               0.991853
segment_osrm_distance_agg                       0.996092

                                   actual_distance_to_destination_agg  \
start_scan_to_end_scan                                       0.907187
actual_time_agg                                              0.956301
osrm_time_agg                                                0.993813
segment_actual_time_agg                                      0.953141
segment_osrm_time_agg                                        0.987727
actual_distance_to_destination_agg                           1.000000
osrm_distance_agg                                            0.997461
segment_osrm_distance_agg                                    0.993257

                                   osrm_distance_agg  \
start_scan_to_end_scan                      0.912889
actual_time_agg                             0.961056
osrm_time_agg                               0.997602
segment_actual_time_agg                     0.958154
segment_osrm_time_agg                       0.991853
actual_distance_to_destination_agg          0.997461
osrm_distance_agg                           1.000000
segment_osrm_distance_agg                   0.994717

                                   segment_osrm_distance_agg
start_scan_to_end_scan                              0.907676
actual_time_agg                                     0.958266
osrm_time_agg                                       0.991609
segment_actual_time_agg                             0.956106
segment_osrm_time_agg                               0.996092
actual_distance_to_destination_agg                  0.993257
```

```
osrm_distance_agg                                   0.994717
segment_osrm_distance_agg                           1.000000
```

[34]:
```
#plot heatmap
plt.figure(figsize=(15,6))
sns.heatmap(corr_df, cmap="YlGnBu", annot=True)
plt.show()
```

Observations

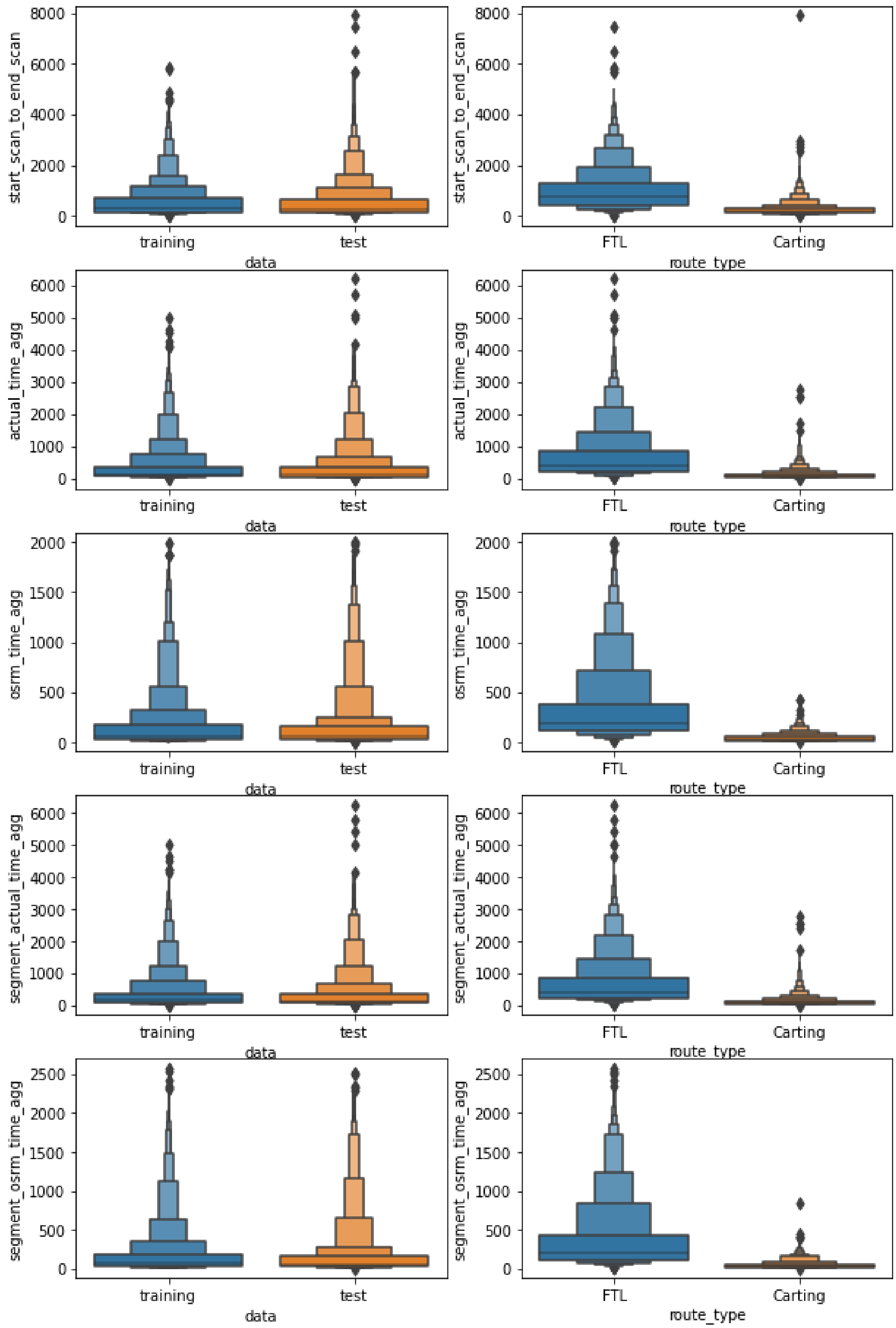
1. We see strong correlation (> 0.9) among all time variables (both cumulative and non-cumulative) - start_scan_to_end_scan, actual_time_agg, osrm_time_agg, segment_actual_time_agg, segment_osrm_time_agg.
2. We see very high correlation (> 0.99) among all distance variables (both cumulative and non-cumulative) - actual_distance_to_destination_agg, osrm_distance_agg, segment_osrm_distance_agg.
3. We also see strong correlation between time and distance variables (both cumulative and non-cumulative). In fact, we see all pairs of time and distance variables highly correlated. This confirms our general understanding that in general, longer distance takes longer time and vice-versa. We can visually see the correlation in the pair plot.

2.8.2 Time variables analysis

Distributions of time variables for various categorical variables

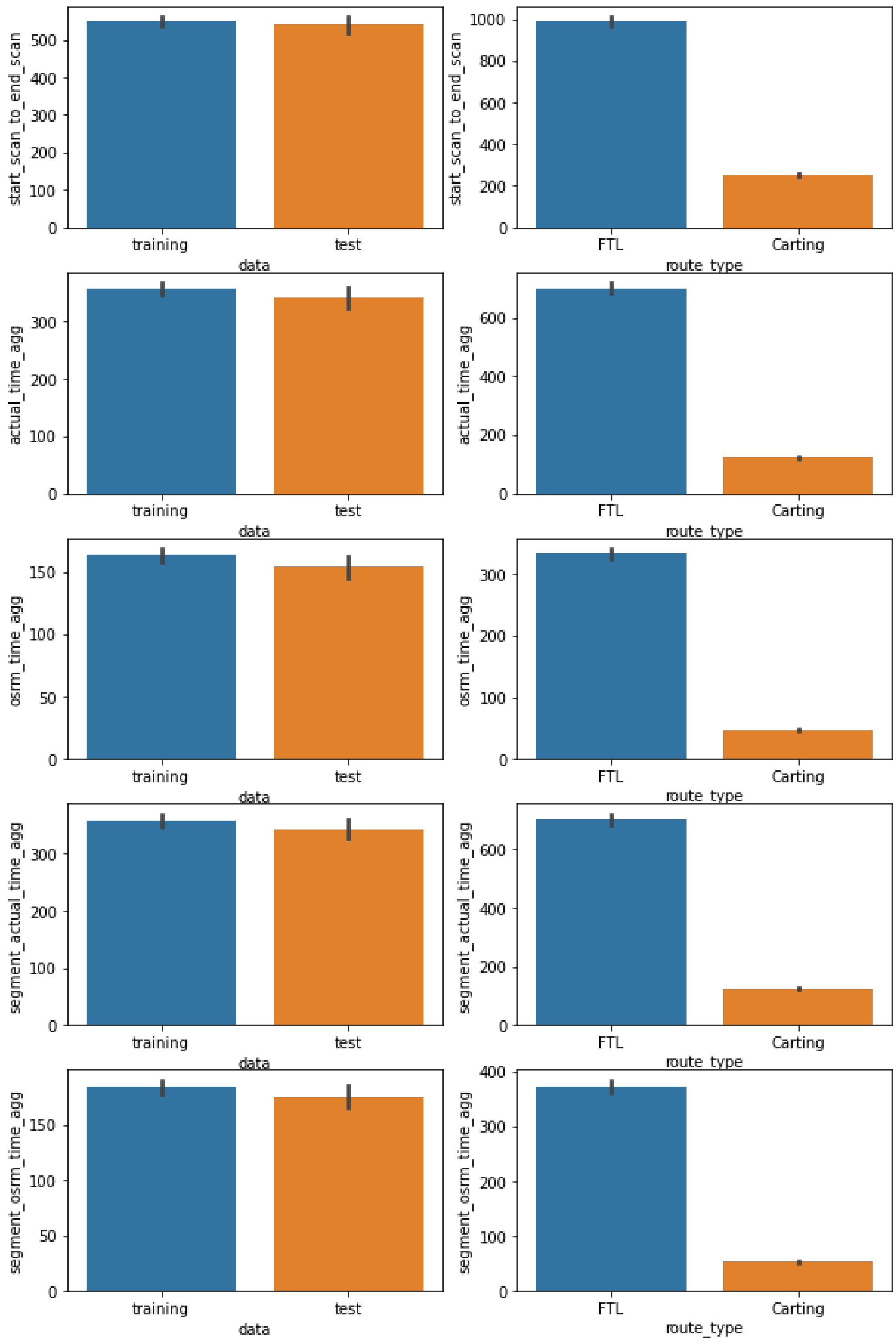
[35]:
```
data = df_aggr2
```

```
fig, ax = plt.subplots(5, 2, figsize=(10, 16))

sns.boxenplot(x='data', y='start_scan_to_end_scan', data=data, ax=ax[0][0])
sns.boxenplot(x='route_type', y='start_scan_to_end_scan', data=data, ax=ax[0][1])

sns.boxenplot(x='data', y='actual_time_agg', data=data, ax=ax[1][0])
sns.boxenplot(x='route_type', y='actual_time_agg', data=data, ax=ax[1][1])

sns.boxenplot(x='data', y='osrm_time_agg', data=data, ax=ax[2][0])
sns.boxenplot(x='route_type', y='osrm_time_agg', data=data, ax=ax[2][1])

sns.boxenplot(x='data', y='segment_actual_time_agg', data=data, ax=ax[3][0])
sns.boxenplot(x='route_type', y='segment_actual_time_agg', data=data, ax=ax[3][1])

sns.boxenplot(x='data', y='segment_osrm_time_agg', data=data, ax=ax[4][0])
sns.boxenplot(x='route_type', y='segment_osrm_time_agg', data=data, ax=ax[4][1])

plt.show()
```


start_scan_to_end_scan
actual_time_agg
osrm_time_agg
segment_actual_time_agg
segment_osrm_time_agg
training
test
data
FTL
Carting
route_type

mean value for time variables for each categorical variables

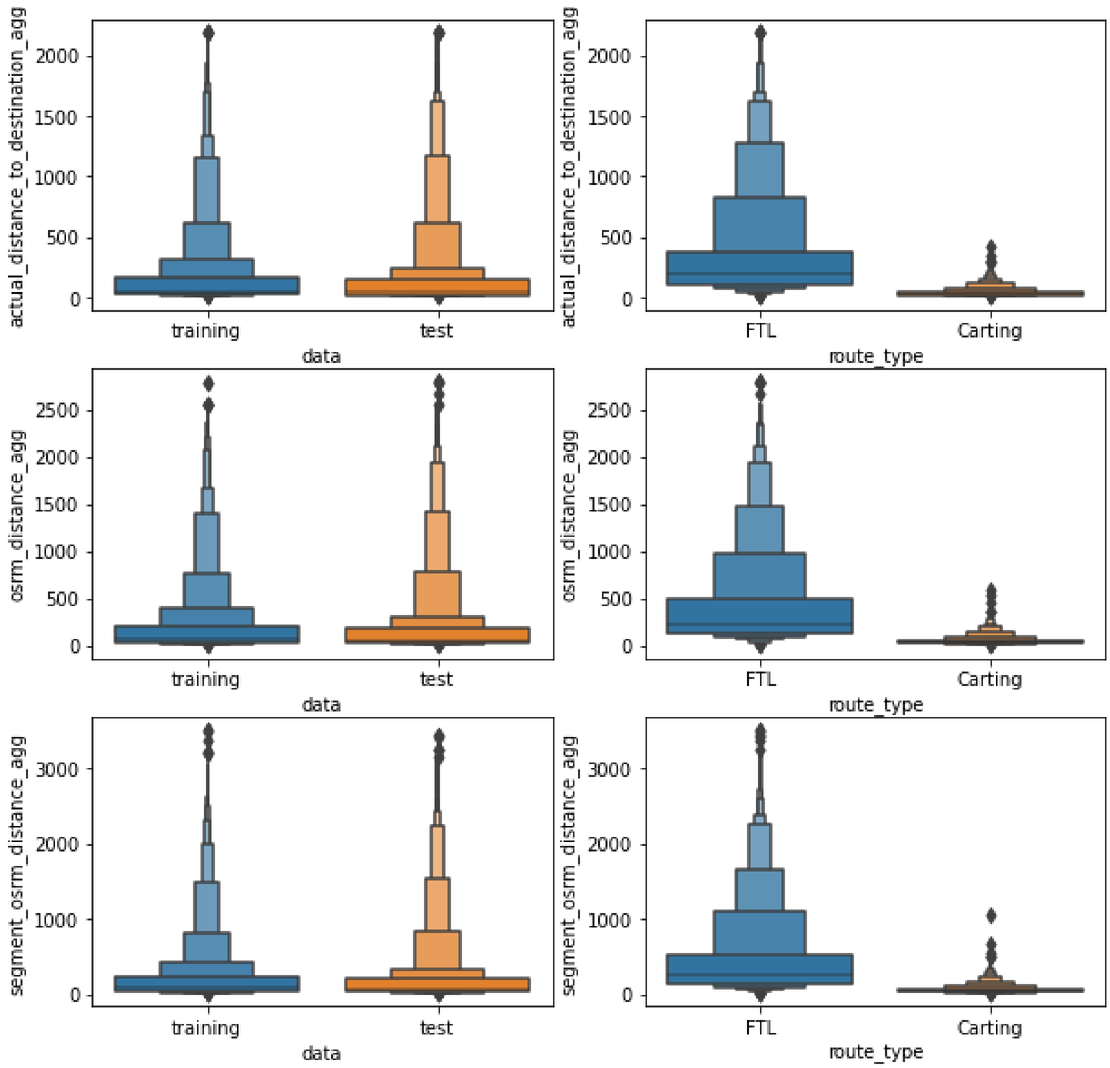
[36]:
```
fig, ax = plt.subplots(5, 2, figsize=(10, 16))

sns.barplot(x='data', y='start_scan_to_end_scan',ci=95, data=data, ax=ax[0][0])
sns.barplot(x='route_type', y='start_scan_to_end_scan', ci=95, data=data,␣
↪ax=ax[0][1])

sns.barplot(x='data', y='actual_time_agg',ci=95, data=data, ax=ax[1][0])
sns.barplot(x='route_type', y='actual_time_agg', ci=95, data=data, ax=ax[1][1])

sns.barplot(x='data', y='osrm_time_agg',ci=95, data=data, ax=ax[2][0])
sns.barplot(x='route_type', y='osrm_time_agg', ci=95, data=data, ax=ax[2][1])

sns.barplot(x='data', y='segment_actual_time_agg',ci=95, data=data, ax=ax[3][0])
sns.barplot(x='route_type', y='segment_actual_time_agg', ci=95, data=data,␣
↪ax=ax[3][1])

sns.barplot(x='data', y='segment_osrm_time_agg',ci=95, data=data, ax=ax[4][0])
sns.barplot(x='route_type', y='segment_osrm_time_agg', ci=95, data=data,␣
↪ax=ax[4][1])

plt.show()
```


start_scan_to_end_scan
training
test
data
FTL
Carting
route_type
actual_time_agg
osrm_time_agg
segment_actual_time_agg
segment_osrm_time_agg

2.8.3 Distance variables analysis

Distributions of distance variables for various categorical variables

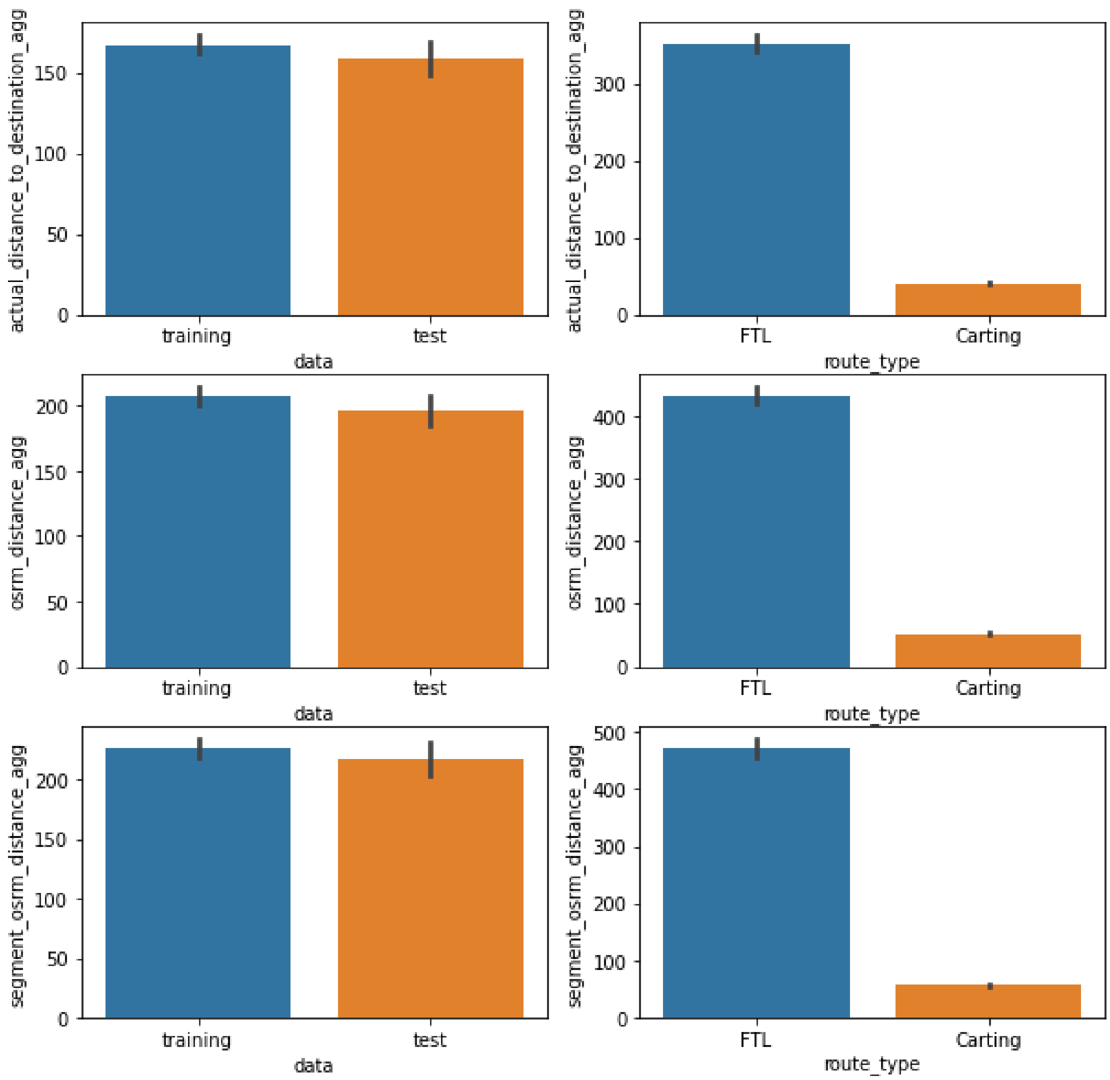
[37]:
```
fig, ax = plt.subplots(3, 2, figsize=(10, 10))

sns.boxenplot(x='data', y='actual_distance_to_destination_agg', data=data,␣
 ↪ax=ax[0][0])
sns.boxenplot(x='route_type', y='actual_distance_to_destination_agg',␣
 ↪data=data, ax=ax[0][1])

sns.boxenplot(x='data', y='osrm_distance_agg', data=data, ax=ax[1][0])
sns.boxenplot(x='route_type', y='osrm_distance_agg', data=data, ax=ax[1][1])

sns.boxenplot(x='data', y='segment_osrm_distance_agg', data=data, ax=ax[2][0])
sns.boxenplot(x='route_type', y='segment_osrm_distance_agg', data=data,␣
 ↪ax=ax[2][1])

plt.show()
```



mean value of distance variables for various categorical variables

[38]:
```
fig, ax = plt.subplots(3, 2, figsize=(10, 10))

sns.barplot(x='data', y='actual_distance_to_destination_agg',ci=95, data=data,␣
 ↪ax=ax[0][0])
sns.barplot(x='route_type', y='actual_distance_to_destination_agg', ci=95,␣
 ↪data=data, ax=ax[0][1])

sns.barplot(x='data', y='osrm_distance_agg',ci=95, data=data, ax=ax[1][0])
sns.barplot(x='route_type', y='osrm_distance_agg', ci=95, data=data,␣
 ↪ax=ax[1][1])
```

```
sns.barplot(x='data', y='segment_osrm_distance_agg',ci=95, data=data,␣
↪ax=ax[2][0])
sns.barplot(x='route_type', y='segment_osrm_distance_agg', ci=95, data=data,␣
↪ax=ax[2][1])

plt.show()
```



Observations

1. Based on distribution plots for both time and distance variables, we observe that *median value* for 'Carting' route_type data is less than that of 'FTL' route_type. Similarly, based on the mean value plot for both time and distance variables, we observe that *mean value* for 'Carting' route_type data is significantly lower than that of 'FTL' route_type. This implies that **trips with route_type 'Carting' are associated with shorter time and**

distance, whereas, trips with route_type 'FTL' are associated with longer time and distance.

2. We see that for all time and distance variables, the distributions of training and test data are identical. Similarly, for all variables, mean values are identical between the training and test data. **Thus division of training and test data seem reasonable.**

2.9 Outliers analysis

In this section, we focus only on aggregated data-set. We consider as outliers all the values which fall outside of the interval **[q1 - 1.5 * IQR, q3 + 1.5 * IQR]**, where q1 is 25% percentile value, q3 is 75% percentile value, and IQR is interquartile range which is equal to (q3-q1). We find outliers for the original aggregated data as well as for log-transformed, sqrt-transformed, and cube-root transformed data.

[39]:
```
def findoutliers(arr):
    q3 = np.percentile(arr, 75)
    q1 = np.percentile(arr, 25)
    iqr = q3-q1
    ulim = q3 + 1.5*iqr
    llim = q1 - 1.5*iqr
    return pd.Series([True if((ele > ulim) or (ele < llim)) else False for ele␣
 ↪in arr])

def makepositive(s, pos_val=0.01):
    return s.transform(lambda val: val if(val > 0) else pos_val)
```

[40]:
```
data = df_aggr2
outliers = []
transformations = [
    ('  original', lambda s:s),
    ('sqrt', lambda s: makepositive(s)**(1/2)),
    ('cuberoot', lambda s: makepositive(s)**(1/3)),
    (' log', lambda s: np.log(makepositive(s)))]

total_n = data.shape[0]
for col in getnumcols(data):
    for trans_name, trans_fn in transformations:
        ret = findoutliers(trans_fn(data[col]))
        outliers_n = ret.sum()
        outliers.append([col, trans_name, outliers_n, np.round((outliers_n /␣
 ↪total_n) * 100, 2)])

print(f'total number of rows in aggregated dataset: {total_n}')
outliers_df = pd.DataFrame(data=outliers, columns=['column', 'transformation',␣
 ↪'outliers count', 'outliers as % of total rows'])

outliers_df = outliers_df.set_index(keys=['column', 'transformation'])
```

```
outliers_df.unstack()
```

```
total number of rows in aggregated dataset: 14817
```

[40]:

```
                                    outliers count                        \
transformation                            original  log cuberoot  sqrt
column
actual_distance_to_destination_agg            1449    0      702   889
actual_time_agg                               1625    5      660   864
factor                                        1335  819     1002  1094
osrm_distance_agg                             1527    0      697   924
osrm_time_agg                                 1511    0      696   857
segment_actual_time_agg                       1643    5      659   861
segment_osrm_distance_agg                     1548    0      745   934
segment_osrm_time_agg                         1492    0      721   895
start_scan_to_end_scan                        1115    3      362   624

                                    outliers as % of total rows                \
transformation                                         original   log cuberoot
column
actual_distance_to_destination_agg                         9.78  0.00     4.74
actual_time_agg                                           10.97  0.03     4.45
factor                                                     9.01  5.53     6.76
osrm_distance_agg                                         10.31  0.00     4.70
osrm_time_agg                                             10.20  0.00     4.70
segment_actual_time_agg                                   11.09  0.03     4.45
segment_osrm_distance_agg                                 10.45  0.00     5.03
segment_osrm_time_agg                                     10.07  0.00     4.87
start_scan_to_end_scan                                     7.53  0.02     2.44


transformation                      sqrt
column
actual_distance_to_destination_agg  6.00
actual_time_agg                     5.83
factor                              7.38
osrm_distance_agg                   6.24
osrm_time_agg                       5.78
segment_actual_time_agg             5.81
segment_osrm_distance_agg           6.30
segment_osrm_time_agg               6.04
start_scan_to_end_scan              4.21
```

Observations

1. The table above shows outlier percentage for each column of the original aggregated data-set as well as the log transformed, square-root transformed, and cube root transformed data. We observe that in the original aggregated data-set, the percentage of outliers across various

columns ranges from **the highest value of 11.09% (for 'segment_actual_time_agg') to the lowest value of 7.53% (for 'start_scan_to_end_scan'). Thus the overall pecentage of outliers in the overall aggregated data-set is rather high.**

2. We also apply log, sqrt, and cube-root transformations on the aggregated data-set before counting the number of outliers. We observe that **log transformation is very effective here in eliminating or reducing the outliers considerably in all the columns except the 'factor' column.**

2.10 Outliers treatment

There are a few potential options in dealing with outliers data, each having specific pros and cons.

1. Removing outliers - If the outliers represent noise/error and form a relatively small portion of the actual data, we can consider removing/trimming them. In the current data-set, however, the percentage of outliers is rather large (ranging from 7.53% to 11.09% across columns and the overall combined percentage could be even higher).

2. Replacing outliers(Winsorization) - The other option could be to replace the outliers with the suitable percentile value of the data.

3. Apply log transformation - As observed in the previous section, the log transformation seems quite effective in removing/reducing the number of outliers significantly (Except for the factor column). If the further intended statistical analysis can work effectively with log-transformed data, we can consider choosing this option. However, there are certain statistical analysis techniques which may not quite yield the same result on log-transformed data as on the original data. For instance, the t-test on the log-transformed data compares geometric means, not the (usual) arithmetic means.

In this case-study, we will create a copy of the aggregated data-set and replace outliers with appropriate percentile values (option 2). In the code below, we evaluate various percentile values for winsorization and select the lowest value which addresses all the outliers. We will use this modified data-set in the hypothesis test section. However, We will not change the original aggregated data-set.

[41]:
```
df_outlier_treated = df_aggr2.copy()

import random
from scipy.stats.mstats import winsorize

winsorizedarr = winsorize(df_outlier_treated['actual_time_agg'],(0.05,0.05))

data = df_outlier_treated
outliers = []
transformations = [
    ('    original', lambda s:s),
    ('   90% winsorize', lambda s:winsorize(s,(0.05, 0.05))),
    ('  80% winsorize', lambda s:winsorize(s,(0.1, 0.1))),
    (' 78% winsorize', lambda s:winsorize(s,(0.11, 0.11))),
    ('75% winsorize', lambda s:winsorize(s,(0.125, 0.125))),
```

```
]

total_n = data.shape[0]
for col in getnumcols(data):
    for trans_name, trans_fn in transformations:
        ret = findoutliers(trans_fn(data[col]))
        outliers_n = ret.sum()
        outliers.append([col, trans_name, outliers_n, np.round((outliers_n /␣
↪total_n) * 100, 2)])

print(f'total number of rows in aggregated dataset: {total_n}')
outliers_df = pd.DataFrame(data=outliers, columns=['column', 'winsorization',␣
↪'outliers count', 'outliers as % of total rows'])

outliers_df = outliers_df.set_index(keys=['column', 'winsorization'])
outliers_df.unstack()
```

```
total number of rows in aggregated dataset: 14817
```

[41]:

```
                                  outliers count                  \
winsorization                           original   90% winsorize
column
actual_distance_to_destination_agg          1449            1449
actual_time_agg                             1625            1625
factor                                      1335            1335
osrm_distance_agg                           1527            1527
osrm_time_agg                               1511            1511
segment_actual_time_agg                     1643            1643
segment_osrm_distance_agg                   1548            1548
segment_osrm_time_agg                       1492            1492
start_scan_to_end_scan                      1115            1115

                                                                 \
winsorization                        80% winsorize  78% winsorize
column
actual_distance_to_destination_agg               0              0
actual_time_agg                               1625              0
factor                                           0              0
osrm_distance_agg                             1527              0
osrm_time_agg                                 1511              0
segment_actual_time_agg                       1643           1643
segment_osrm_distance_agg                     1548              0
segment_osrm_time_agg                         1492              0
start_scan_to_end_scan                           0              0

                                              outliers as % of total rows  \
winsorization                      75% winsorize                  original
```

```
column
actual_distance_to_destination_agg            0                        9.78
actual_time_agg                               0                       10.97
factor                                        0                        9.01
osrm_distance_agg                             0                       10.31
osrm_time_agg                                 0                       10.20
segment_actual_time_agg                       0                       11.09
segment_osrm_distance_agg                     0                       10.45
segment_osrm_time_agg                         0                       10.07
start_scan_to_end_scan                        0                        7.53

                                                                    \
winsorization                       90% winsorize   80% winsorize
column
actual_distance_to_destination_agg           9.78            0.00
actual_time_agg                             10.97           10.97
factor                                       9.01            0.00
osrm_distance_agg                           10.31           10.31
osrm_time_agg                               10.20           10.20
segment_actual_time_agg                     11.09           11.09
segment_osrm_distance_agg                   10.45           10.45
segment_osrm_time_agg                       10.07           10.07
start_scan_to_end_scan                       7.53            0.00


winsorization                      78% winsorize 75% winsorize
column
actual_distance_to_destination_agg          0.00           0.0
actual_time_agg                             0.00           0.0
factor                                      0.00           0.0
osrm_distance_agg                           0.00           0.0
osrm_time_agg                               0.00           0.0
segment_actual_time_agg                    11.09           0.0
segment_osrm_distance_agg                   0.00           0.0
segment_osrm_time_agg                       0.00           0.0
start_scan_to_end_scan                      0.00           0.0
```

[42]:
```
# At around 75% winsorization, we see zero outliers across all the columns. So␣
↪we will use this value.
for col in getnumcols(df_outlier_treated):
    df_outlier_treated[col] = winsorize(df_outlier_treated[col],(0.125,0.125))

df_outlier_treated2 = df_aggr2.copy()

ret = [False] * len(df_outlier_treated2)
for col in ['actual_time_agg', 'segment_actual_time_agg', 'osrm_time_agg',␣
↪'segment_osrm_time_agg']:
```

```
    ret = ret | findoutliers(df_outlier_treated2[col])

df_outlier_treated2 = df_outlier_treated2[~ret]

print(len(df_outlier_treated2)/len(df_aggr2) * 100)
```

```
87.14314638590808
```

2.11 Checking relationship between aggregated fields

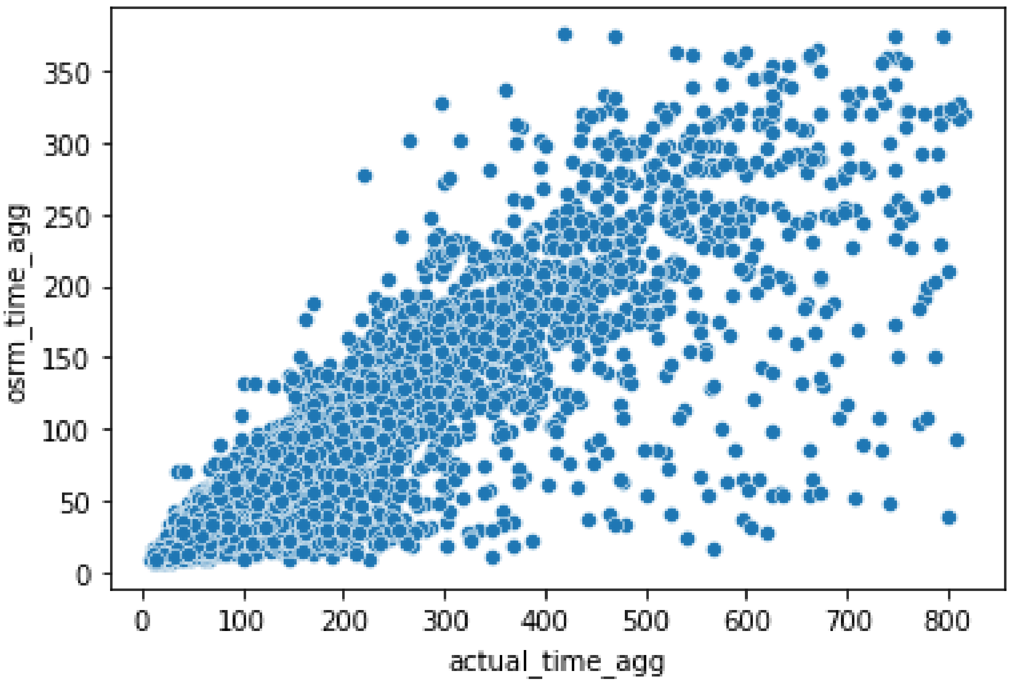
In this section, we compare relationships between various related time and distance fields visually(scatter plot) and using hypothesis tests. As seen in the previous sections, the various time and distance variables are right skewed and contain several outliers. Since two-sample paired t-test makes an assumption of normally distributed data, it is not suitable for our data-set. We instead use the following two tests.

1. Hypothesis testing using bootstrap CI - We use bootstrap resampling (With replacement) to generate sampling distribution of mean of the difference and also compute confidence interval for the mean of difference. If that confidence interval does not contain zero value (as the null hypothesis assumes zero difference), we get a statistically significant result and reject the null hypothesis of zero difference.
2. wilcoxon's signed rank test - We use this non-parametric test as it doesn't assume normally distributed data.

Note: - We use the aggregated dataset without outliers for the hypothesis test. Also, the data-set given to us contain most of its data from sept-2018. In order to generate more random and independent sample, we take a smaller random sample of 5000 rows and perform our hypothesis tests on it.

2.11.1 Comparing actual_time aggregated value and OSRM time aggregated value

[43]:
```
import scipy.stats as stats

#use data without outliers and take a smaller sample
data = df_outlier_treated2.sample(5000)

sns.scatterplot(x='actual_time_agg', y='osrm_time_agg', data=data)
plt.show()
```



Two-sample paired hypothesis test using bootstrapping CI

H0: The mean difference between 'actual_time_agg' and 'osrm_time_agg' is zero.

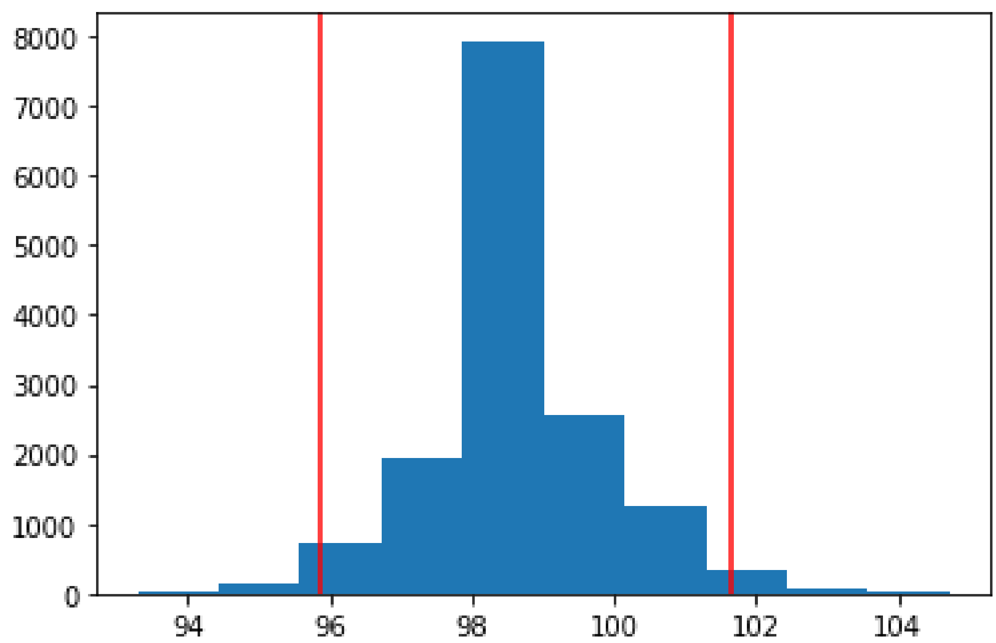
H1: The mean difference is not zero.

[44]:
```
#custom bootstrap function
#def bootstrap_mean(data, sample_size=None, n_rep=9999, random_state=None):
#    sample_size = sample_size if (sample_size != None) else len(data)
#    bootstrapped_means = []
#    for rep in range(n_rep):
#        resample = data.sample(n=sample_size, replace=True, random_state =␣
↪random_state)
#        bootstrapped_means.append(np.mean(resample))
#    conf_interval = np.percentile(bootstrapped_means, [2.5, 97.5])
#    se = np.std(bootstrapped_means)
#    return (conf_interval, se, bootstrapped_means)
```

[46]:
```
d = (data['actual_time_agg'] - data['osrm_time_agg'])

sample_means = []
def meanfn(sample):
    s_mean = np.mean(sample)
    sample_means.append(s_mean)
    return s_mean
```

```
rng = np.random.default_rng(seed=37)
res = stats.bootstrap((d,), meanfn, confidence_level=0.95, vectorized=False,
                random_state=rng, method='BCa')

print(res.confidence_interval)

plt.hist(sample_means)
plt.axvline(x=res.confidence_interval.low, c="r")
plt.axvline(x=res.confidence_interval.high, c="r")
plt.show()
```

```
ConfidenceInterval(low=95.86241306408151, high=101.63992246689878)
```



Wilcoxon's signed rank test

[47]:
```
stats.wilcoxon(d)
```

[47]:
```
WilcoxonResult(statistic=14907.0, pvalue=0.0)
```

Observations

1. Hoever, the bootstrap confidence interval for the difference between aggregated actual and osrm time doesn't include the expected null hypothesis difference of zero. Thus we reject the null hypothesis of zero difference.
2. Similarly, the pvalue returned from Wilcoxon's signed rank test is close to zero. Thus we reject the null hypothesis of zero difference.

3. While there indeed is high correlation between aggregated actual and osrm time, we also observe that the *mean of the paired difference is significantly different from zero*(null hypothesis value).
4. Thus we can state at 95% confidence level that there is a significant difference between the aggregated actual time and osrm time. This is also visible in the scatter plot where we see a large deviation away from the diagonal line x=y.

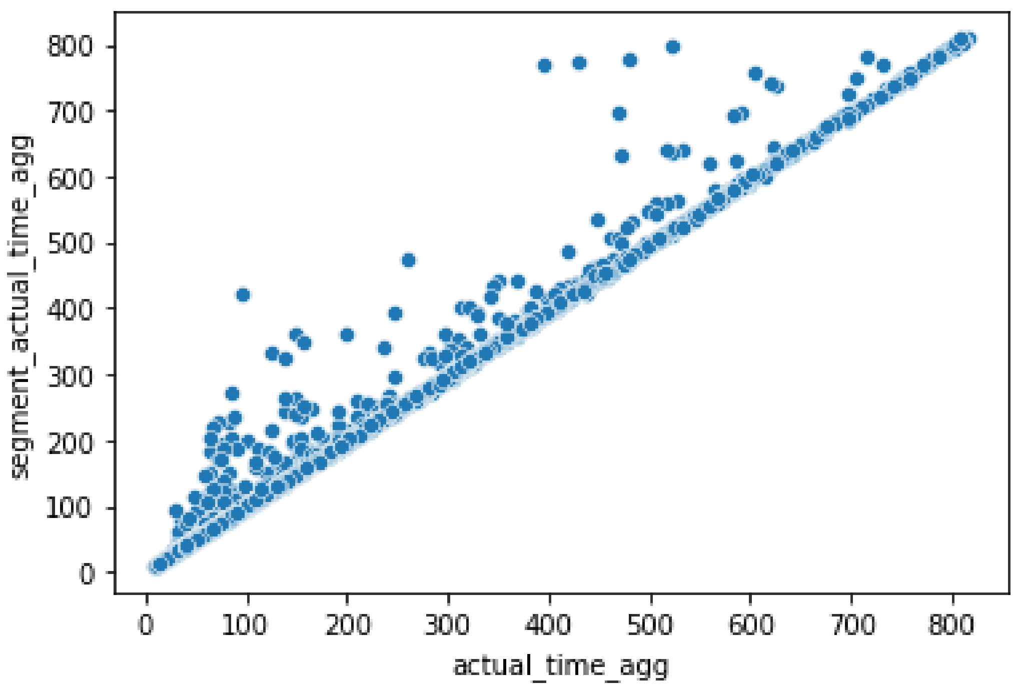
Recommendation for DS team

There is an overall significant difference between the actual delivery time versus the predicted delivery time (from OSRM). This presents a potential opportunity for the data science team to adjust osrm prediction with expected delays based on the historical data. At the same time, this is an opportunity for Logistics team to find and eliminate any factors which may be contributing to the overall delay.

2.11.2 Comparing actual_time aggregated value and segment actual time aggregated value

[48]:
```
sns.scatterplot(x='actual_time_agg', y='segment_actual_time_agg', data=data)
```

[48]: <AxesSubplot:xlabel='actual_time_agg', ylabel='segment_actual_time_agg'>



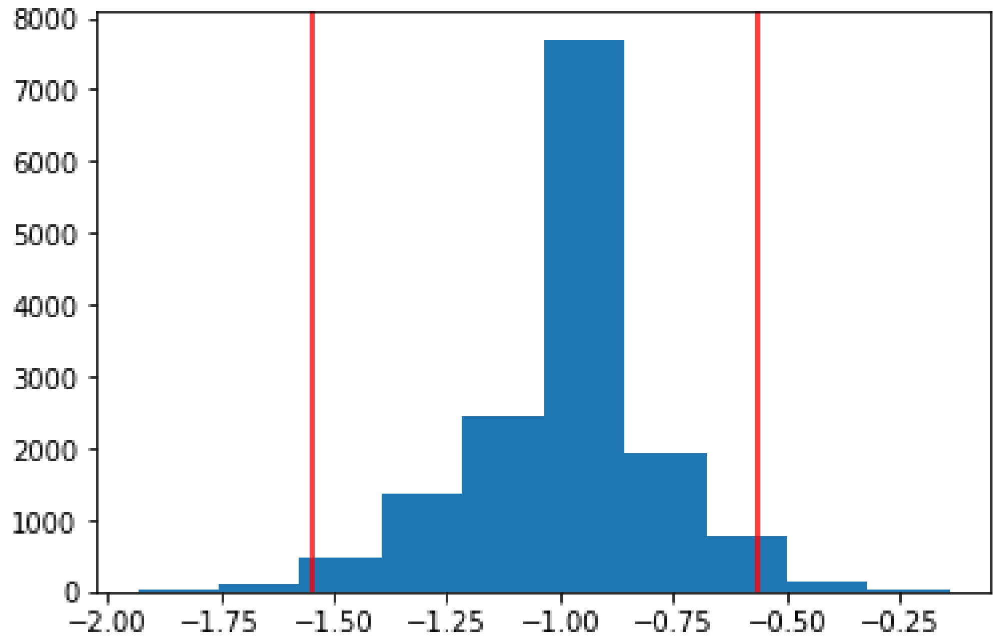
Two-sample paired hypothesis test using bootstrapping CI

H0: The mean difference between 'actual_time_agg' and 'segment_actual_time_agg' is zero.

H1: The mean difference is not zero.

[50]:
```
d = (data['actual_time_agg'] - data['segment_actual_time_agg'])

sample_means = []

rng = np.random.default_rng(seed=37)
res = stats.bootstrap((d,), meanfn, confidence_level=0.95, vectorized=False,
                random_state=rng, method='BCa')

print(res.confidence_interval)

plt.hist(sample_means)
plt.axvline(x=res.confidence_interval.low, c="r")
plt.axvline(x=res.confidence_interval.high, c="r")
plt.show()
```

```
ConfidenceInterval(low=-1.5475363959194206, high=-0.5629428013248543)
```



Wilcoxon's signed rank test

[51]:
```
stats.wilcoxon(d)
```

[51]:
```
WilcoxonResult(statistic=902412.0, pvalue=0.0)
```

Observations

1. The bootstrap confidence interval for the difference between aggregated actual and aggregated segment actual time doesn't include the expected null hypothesis difference of zero. Thus we

reject the null hypothesis of zero difference.

2. Similarly, the pvalue returned from Wilcoxon's signed rank test is close to zero. Thus we reject the null hypothesis of zero difference.

3. While there indeed is high correlation between aggregated actual and aggregated segment actual time, we also observe that the *mean of the paired difference is significantly different from zero*(null hypothesis value).

4. Thus we can state at 95% confidence level that there is a significant difference between the aggregated actual time and aggregated segment actual time. We can also verify this visually. In the scatterplot between the two variables, while there are many observations which fall near the diagonal line x=y, there are also high number of deviations away from the diagonal line.

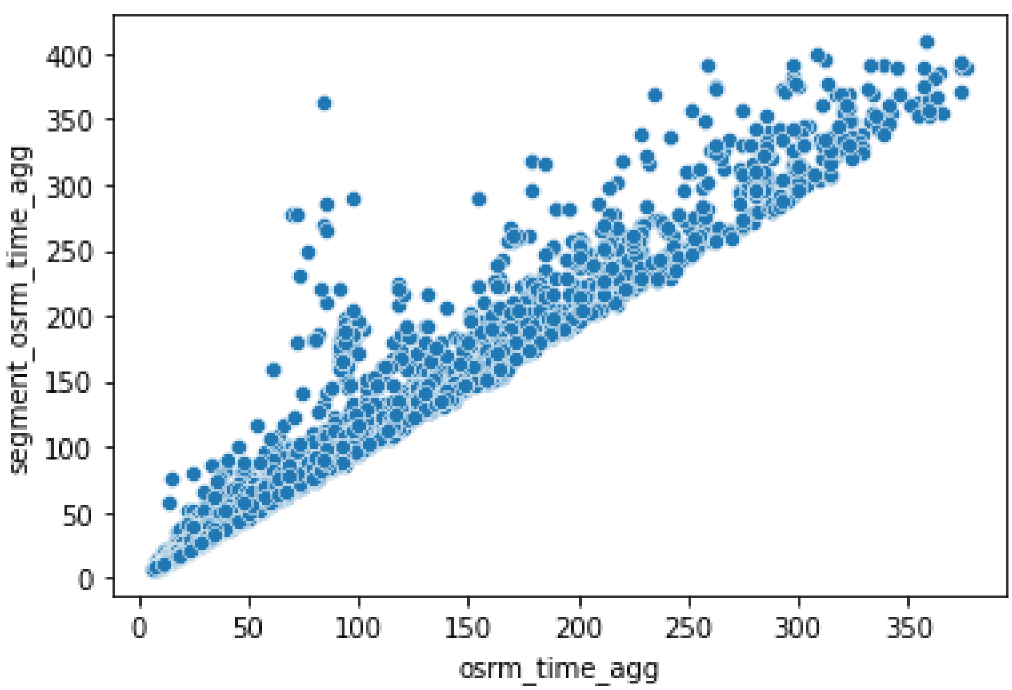
Recommendation for DS team

There is an overall significant difference between the aggregated actual delivery time versus the aggregated segment actual delivery time (from OSRM). This is an unexpected finding. This presents a potential opportunity for the data science team to investigate why the sum of segmented time is different from the cumulative actual time.

2.11.3 Comparing osrm time aggregated value and segment osrm time aggregated value

[52]:
```
sns.scatterplot(x='osrm_time_agg', y='segment_osrm_time_agg', data=data)
```

[52]: <AxesSubplot:xlabel='osrm_time_agg', ylabel='segment_osrm_time_agg'>



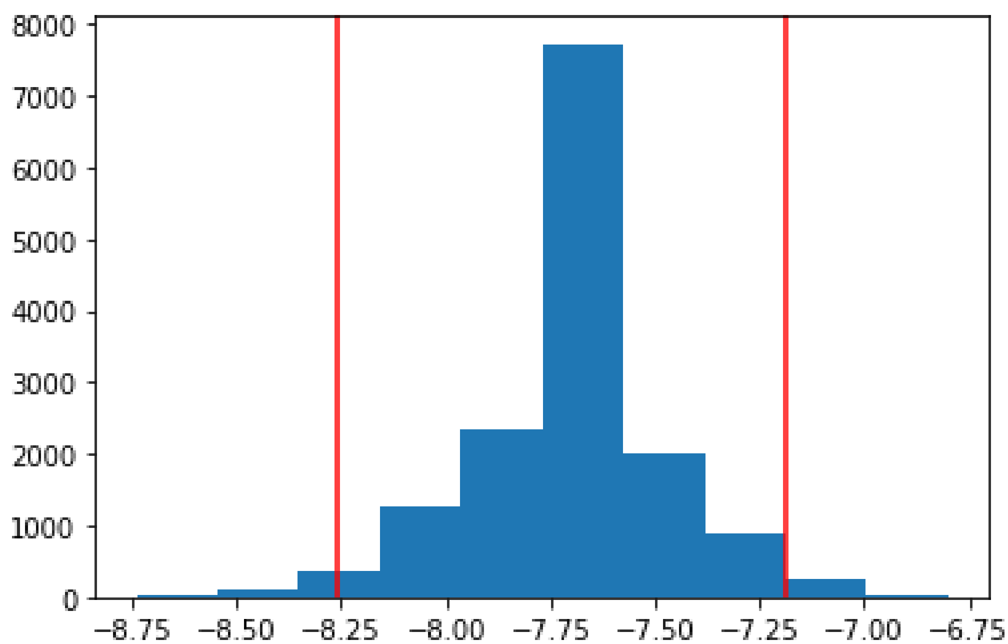
Two-sample paired hypothesis test using bootstrapping CI

H0: The mean difference between 'osrm_time_agg' and 'segment_osrm_time_agg' is zero.

H1: The mean difference is not zero.

[53]:
```
d = (data['osrm_time_agg'] - data['segment_osrm_time_agg'])

sample_means = []

rng = np.random.default_rng(seed=37)
res = stats.bootstrap((d,), meanfn, confidence_level=0.95, vectorized=False,
                random_state=rng, method='BCa')

print(res.confidence_interval)

plt.hist(sample_means)
plt.axvline(x=res.confidence_interval.low, c="r")
plt.axvline(x=res.confidence_interval.high, c="r")
plt.show()
```

```
ConfidenceInterval(low=-8.257253212059139, high=-7.185702402042958)
```



Wilcoxon's signed rank test

[54]:
```
stats.wilcoxon(d)
```

[54]: WilcoxonResult(statistic=1836839.5, pvalue=4.4693343307340444e-206)

Observations

1. The bootstrap confidence interval for the difference between aggregated osrm and aggregated segment osrm time doesn't include the expected null hypothesis difference of zero. Thus we reject the null hypothesis of zero difference.
2. Similarly, the pvalue returned from Wilcoxon's signed rank test is close to zero. Thus we reject the null hypothesis of zero difference.
3. While there indeed is high correlation between aggregated osrm and aggregated segment osrm time, we also observe that the *mean of the paired difference is significantly different from zero*(null hypothesis value).
4. Thus we can state at 95% confidence level that there is a significant difference between the aggregated osrm time and aggregated segment osrm time. We can also verify this visually. In the scatterplot between the two variables, while there are many observations which fall near the diagonal line x=y, there are also high number of deviations away from the diagonal line.

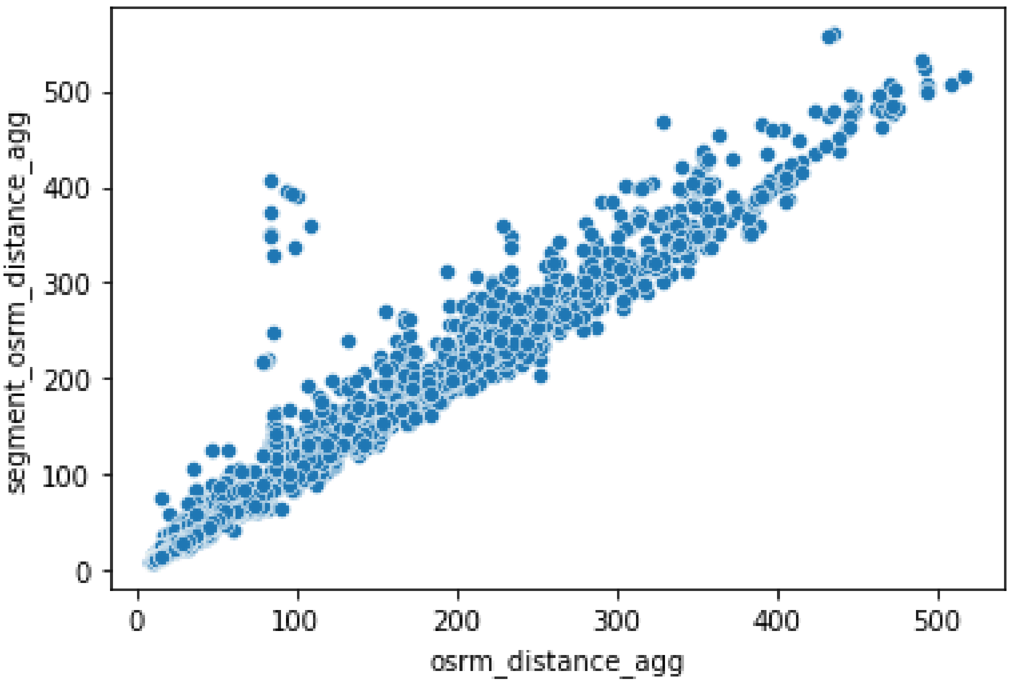
Recommendation for DS team

There is an overall significant difference between the aggregated osrm delivery time versus the aggregated segment osrm delivery time (from OSRM). Since OSRM is dynamic in nature, this may or may not be unexpected depending on how the cumulative osrm values are computed. This presents a potential opportunity for the data science team to investigate further.

2.11.4 Comparing osrm distance aggregated value and segment osrm distance aggregated value

[55]:
```
sns.scatterplot(x='osrm_distance_agg', y='segment_osrm_distance_agg', data=data)
```

[55]: <AxesSubplot:xlabel='osrm_distance_agg', ylabel='segment_osrm_distance_agg'>



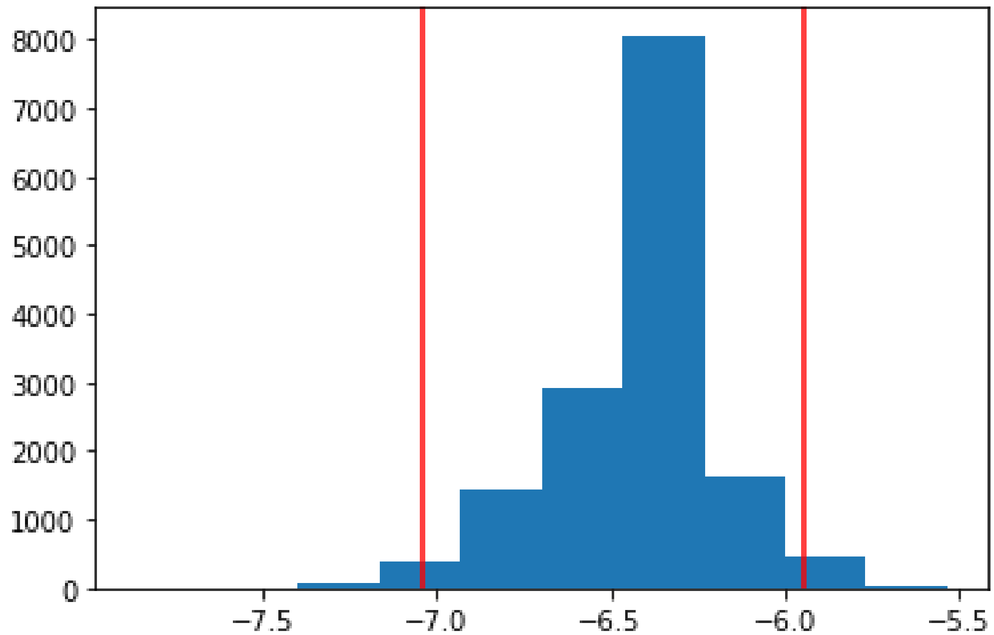
Two-sample paired hypothesis test using bootstrapping CI

H0: The mean difference between 'osrm_ditance_agg' and 'segment_osrm_distance_agg' is zero.

H1: The mean difference is not zero.

[56]:
```
d = (data['osrm_distance_agg'] - data['segment_osrm_distance_agg'])

sample_means = []

rng = np.random.default_rng(seed=37)
res = stats.bootstrap((d,), meanfn, confidence_level=0.95, vectorized=False,
                random_state=rng, method='BCa')

print(res.confidence_interval)

plt.hist(sample_means)
plt.axvline(x=res.confidence_interval.low, c="r")
plt.axvline(x=res.confidence_interval.high, c="r")
plt.show()
```

```
ConfidenceInterval(low=-7.035239380201518, high=-5.943864044429519)
```



Wilcoxon's signed rank test

[57]:
```
stats.wilcoxon(d)
```

[57]: `WilcoxonResult(statistic=1482896.5, pvalue=6.028593832429975e-267)`

Observations

1. The bootstrap confidence interval for the difference between aggregated osrm and aggregated segment osrm distance doesn't include the expected null hypothesis difference of zero. Thus we reject the null hypothesis of zero difference.
2. Similarly, the pvalue returned from Wilcoxon's signed rank test is close to zero. Thus we reject the null hypothesis of zero difference.
3. While there indeed is high correlation between aggregated osrm and aggregated segment osrm distance, we also observe that the *mean of the paired difference is significantly different from zero*(null hypothesis value).
4. Thus we can state at 95% confidence level that there is a significant difference between the aggregated osrm distance and aggregated segment osrm distance. We can also verify this visually. In the scatterplot between the two variables, while there are many observations which fall near the diagonal line x=y, there are also high number of deviations away from the diagonal line.

Recommendation for DS team

There is an overall significant difference between the aggregated osrm delivery time versus the aggregated segment osrm delivery time (from OSRM). Since OSRM is dynamic in nature, this may or may not be unexpected depending on how the cumulative osrm values are computed. This presents a potential opportunity for the data science team to investigate further.

2.12 Feature creation

1. Destination Name: Split and extract features out of destination. City-place-code (State)
2. Source Name: Split and extract features out of destination. City-place-code (State)
3. Trip_creation_time: Extract features like month, year and day etc

2.12.1 Create features from source name and destination name

The usual format of source_name and destination_name values is **<city><*place*><code>(state)**. We use regular expressions to extract individual components and create 8 new features; city, place, code, and state for both source and destination.

Note: There are some values where character '_' comes in the actual name. For instance 'Surat_Central_I_4 (Gujarat)' and 'GZB_Mohan_Nagar_DPC (Uttar Pradesh)'. Since there's is no clearly defined rule to correctly identify each components in the presence of additional underscores in the data, we follow the rule stated below.

All the characters preceding the first underscore represent 'city' value. Similarly, all the characters between the first and second underscores represent 'place' component. All the character between the second underscore and the left parenthesis (including any additional underscores) are considered 'code' components. Finally, the value inside the round parenthesis is considered state component.

[58]:
```python
#Destination Name: Split and extract features out of destination. City_place_code(State)

data = df_aggr2

import re
regx = re.compile(r'^([^_]*)(?:_([^_]*))?(?:_(.*))?\((.*)\)$')

def getmatchcomp(str, grpno):
    m = regx.match(str)
    if(m != None and m.groups != None):
        grps = m.groups()
        if(grps != None and grpno < len(grps) and grps[grpno] != None):
            return grps[grpno]
    return 'NA'

def process_name(val):
    if(val != None):
        #convert to lower
        val = val.lower()
        #remove spaces
        val = val.replace(' ', '')
    return val
```

```python
def correct_city(city):
    city = process_name(city)

    city_map = {
        'bengaluru': 'bengaluru',
        'bangalore': 'bengaluru',
        'nowda': 'noida'
    }
    #TODO: add more corrections in map above
    if(city in city_map.keys()):
        city=city_map[city]

    return city

for data in [df, df_agg_first_level, df_aggr2]:
    data['s_city'] = data['source_name'].apply(lambda x:␣
↪correct_city(getmatchcomp(x, 0)))
    data['s_place'] = data['source_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 1)))
    data['s_code'] = data['source_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 2)))
    data['s_state'] = data['source_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 3)))

    data['d_city'] = data['destination_name'].apply(lambda x:␣
↪correct_city(getmatchcomp(x, 0)))
    data['d_place'] = data['destination_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 1)))
    data['d_code'] = data['destination_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 2)))
    data['d_state'] = data['destination_name'].apply(lambda x:␣
↪process_name(getmatchcomp(x, 3)))

data[['source_name', 's_city', 's_place', 's_code', 's_state',␣
↪'destination_name', 'd_city', 'd_place', 'd_code', 'd_state']].head(5)
```

[58]:
```
                           source_name     s_city   s_place s_code  \
0  Bhopal_Trnsport_H (Madhya Pradesh)     bhopal  trnsport      h
1       Tumkur_Veersagr_I (Karnataka)     tumkur  veersagr      i
2    Bangalore_Nelmngla_H (Karnataka)  bengaluru  nelmngla      h
3            Mumbai Hub (Maharashtra)   mumbaihub        na     na
4              Bellary_Dc (Karnataka)    bellary        dc     na

         s_state                   destination_name      d_city   d_place  \
0  madhyapradesh      Gurgaon_Bilaspur_HB (Haryana)     gurgaon  bilaspur
1      karnataka  Chikblapur_ShntiSgr_D (Karnataka)  chikblapur  shntisgr
```

```
2      karnataka     Chandigarh_Mehmdpur_H (Punjab)  chandigarh  mehmdpur
3    maharashtra     Mumbai_MiraRd_IP (Maharashtra)      mumbai    mirard
4      karnataka             Bellary_Dc (Karnataka)     bellary        dc

  d_code      d_state
0     hb      haryana
1      d    karnataka
2      h       punjab
3     ip  maharashtra
4     na    karnataka
```

2.12.2 Create features from Trip_creation_time

[59]:

```
#create year, month,and day features from trip_creation_time
data['ct_year'] = data['trip_creation_time'].dt.year
data['ct_month'] = data['trip_creation_time'].dt.month
data['ct_day'] = data['trip_creation_time'].dt.day

print(data['ct_year'].value_counts())
print('\n')
print(data['ct_month'].value_counts())
print('\n')
print(data['ct_day'].value_counts())

data[['trip_creation_time', 'ct_year', 'ct_month', 'ct_day']].sample(5)
```

```
2018    14817
Name: ct_year, dtype: int64



9     13029
10     1788
Name: ct_month, dtype: int64



18    791
15    783
13    750
12    747
21    740
22    740
17    722
14    712
20    704
25    697
26    685
19    676
```

```
24    660
27    652
23    631
3     631
16    616
28    608
29    607
1     605
2     552
30    508
Name: ct_day, dtype: int64
```

[59]:

	trip_creation_time	ct_year	ct_month	ct_day
11987	2018-09-29 01:30:51.318072	2018	9	29
2460	2018-09-15 06:05:41.613364	2018	9	15
466	2018-09-12 17:33:39.498320	2018	9	12
11978	2018-09-29 01:15:44.703806	2018	9	29
4045	2018-09-17 18:20:42.819693	2018	9	17

2.12.3 One-hot encoding of categorical variables

We one-hot encode route_type and data variables.

[60]:
```
df_aggr2 = pd.get_dummies(df_aggr2, columns = ['route_type', 'data'])
```

2.12.4 Standardization/Normalization

As seen in the previous sections, our data contains a lot of outiers. Standardization usually is more robust against outliers compared to min-max normalization. We will standardize all distance and time columns in the aggregated data set.

[61]:
```
data = df_aggr2

# standardization
from sklearn.preprocessing import StandardScaler
zscaler = StandardScaler()

for col in ['actual_time_agg', 'osrm_time_agg', 'segment_actual_time_agg',␣
 ↪'segment_osrm_time_agg', 'osrm_distance_agg', 'segment_osrm_distance_agg',␣
 ↪'actual_distance_to_destination_agg', 'start_scan_to_end_scan']:
    colname = col + '_z'
    arr = data[col].to_numpy().reshape(-1,1)
    data[colname] = zscaler.fit_transform(arr)
```

2.12.5 Drop unnecessary columns

[62]:
```
df_aggr2.
↪drop(labels=['trip_uuid','route_schedule_uuid','source_name','destination_name'],␣
↪axis=1, inplace=True)
```

[63]:
```
### Save final aggregated data-set
df_aggr2.to_csv('delhivery_aggregated_data.csv', index=False)

df_aggr2.head(5).T
```

[63]:
```
                                                          0  \
trip_creation_time                   2018-09-12 00:00:16.535741
source_center                                      IND462022AAA
destination_center                                 IND000000ACB
od_start_time                        2018-09-12 00:00:16.535741
od_end_time                          2018-09-13 13:40:23.123744
actual_time_agg                                          1562.0
segment_actual_time_agg                                  1548.0
osrm_time_agg                                             717.0
segment_osrm_time_agg                                    1008.0
actual_distance_to_destination_agg                   824.732854
osrm_distance_agg                                      991.3523
segment_osrm_distance_agg                             1320.4733
start_scan_to_end_scan                                   2260.0
factor                                                     2.18
s_city                                                   bhopal
s_place                                                trnsport
s_code                                                        h
s_state                                           madhyapradesh
d_city                                                  gurgaon
d_place                                                bilaspur
d_code                                                       hb
d_state                                                 haryana
ct_year                                                    2018
ct_month                                                      9
ct_day                                                       12
route_type_Carting                                            0
route_type_FTL                                                1
data_test                                                     0
data_training                                                 1
actual_time_agg_z                                      2.169358
osrm_time_agg_z                                        2.050747
segment_actual_time_agg_z                              2.146791
segment_osrm_time_agg_z                                2.629468
osrm_distance_agg_z                                    2.126321
segment_osrm_distance_agg_z                            2.633784
```

```
actual_distance_to_destination_agg_z                     2.160194
start_scan_to_end_scan_z                                 2.561997

                                                                1  \
trip_creation_time                     2018-09-12 00:00:22.886430
source_center                                        IND572101AAA
destination_center                                   IND562101AAA
od_start_time                          2018-09-12 00:00:22.886430
od_end_time                            2018-09-12 03:01:59.598855
actual_time_agg                                             143.0
segment_actual_time_agg                                     141.0
osrm_time_agg                                                68.0
segment_osrm_time_agg                                        65.0
actual_distance_to_destination_agg                      73.186911
osrm_distance_agg                                          85.111
segment_osrm_distance_agg                                 84.1894
start_scan_to_end_scan                                      181.0
factor                                                        2.1
s_city                                                     tumkur
s_place                                                  veersagr
s_code                                                          i
s_state                                                 karnataka
d_city                                                 chikblapur
d_place                                                  shntisgr
d_code                                                          d
d_state                                                 karnataka
ct_year                                                      2018
ct_month                                                        9
ct_day                                                         12
route_type_Carting                                              1
route_type_FTL                                                  0
data_test                                                       0
data_training                                                   1
actual_time_agg_z                                       -0.376818
osrm_time_agg_z                                         -0.343616
segment_actual_time_agg_z                               -0.382742
segment_osrm_time_agg_z                                 -0.368643
osrm_distance_agg_z                                     -0.321847
segment_osrm_distance_agg_z                              -0.33367
actual_distance_to_destination_agg_z                    -0.299446
start_scan_to_end_scan_z                                -0.547326

                                                                2  \
trip_creation_time                     2018-09-12 00:00:33.691250
source_center                                        IND562132AAA
destination_center                                   IND160002AAC
od_start_time                          2018-09-12 00:00:33.691250
```

```
od_end_time                          2018-09-14 17:34:55.442454
actual_time_agg                                          3072.0
segment_actual_time_agg                                  3308.0
osrm_time_agg                                            1726.0
segment_osrm_time_agg                                    1941.0
actual_distance_to_destination_agg                  1932.273969
osrm_distance_agg                                     2348.3098
segment_osrm_distance_agg                             2545.2678
start_scan_to_end_scan                                   3934.0
factor                                                     1.78
s_city                                                bengaluru
s_place                                                nelmngla
s_code                                                        h
s_state                                               karnataka
d_city                                               chandigarh
d_place                                                mehmdpur
d_code                                                        h
d_state                                                  punjab
ct_year                                                    2018
ct_month                                                      9
ct_day                                                       12
route_type_Carting                                            0
route_type_FTL                                                1
data_test                                                     0
data_training                                                 1
actual_time_agg_z                                       4.87882
osrm_time_agg_z                                        5.773262
segment_actual_time_agg_z                              5.310954
segment_osrm_time_agg_z                                5.595785
osrm_distance_agg_z                                    5.792076
segment_osrm_distance_agg_z                             5.57366
actual_distance_to_destination_agg_z                   5.784925
start_scan_to_end_scan_z                               5.065608

                                                              3  \
trip_creation_time                   2018-09-12 00:01:00.113710
source_center                                      IND400072AAB
destination_center                                 IND401104AAA
od_start_time                        2018-09-12 00:01:00.113710
od_end_time                          2018-09-12 01:41:29.809822
actual_time_agg                                            59.0
segment_actual_time_agg                                    59.0
osrm_time_agg                                              15.0
segment_osrm_time_agg                                      16.0
actual_distance_to_destination_agg                    17.175274
osrm_distance_agg                                         19.68
segment_osrm_distance_agg                               19.8766
```

start_scan_to_end_scan	100.0
factor	3.93
s_city	mumbaihub
s_place	na
s_code	na
s_state	maharashtra
d_city	mumbai
d_place	mirard
d_code	ip
d_state	maharashtra
ct_year	2018
ct_month	9
ct_day	12
route_type_Carting	1
route_type_FTL	0
data_test	0
data_training	1
actual_time_agg_z	-0.527543
osrm_time_agg_z	-0.53915
segment_actual_time_agg_z	-0.530163
segment_osrm_time_agg_z	-0.52443
osrm_distance_agg_z	-0.498605
segment_osrm_distance_agg_z	-0.48804
actual_distance_to_destination_agg_z	-0.482759
start_scan_to_end_scan_z	-0.668469

	4
trip_creation_time	2018-09-12 00:02:09.740725
source_center	IND583101AAA
destination_center	IND583101AAA
od_start_time	2018-09-12 00:02:09.740725
od_end_time	2018-09-12 12:00:30.683231
actual_time_agg	341.0
segment_actual_time_agg	340.0
osrm_time_agg	117.0
segment_osrm_time_agg	115.0
actual_distance_to_destination_agg	127.4485
osrm_distance_agg	146.7918
segment_osrm_distance_agg	146.7919
start_scan_to_end_scan	718.0
factor	2.91
s_city	bellary
s_place	dc
s_code	na
s_state	karnataka
d_city	bellary
d_place	dc

```
d_code                                                   na
d_state                                           karnataka
ct_year                                                2018
ct_month                                                  9
ct_day                                                   12
route_type_Carting                                        0
route_type_FTL                                            1
data_test                                                 0
data_training                                             1
actual_time_agg_z                                 -0.021538
osrm_time_agg_z                                    -0.16284
segment_actual_time_agg_z                         -0.024976
segment_osrm_time_agg_z                           -0.209676
osrm_distance_agg_z                               -0.155219
segment_osrm_distance_agg_z                       -0.183405
actual_distance_to_destination_agg_z               -0.12186
start_scan_to_end_scan_z                           0.255804
```

2.13 Additional Business Insights

In this section, we use the features created in the previous sections to analyze data, answer speific questions, and draw potentially useful insights/recommendations.

2.13.1 Check from where most orders originate (State, city etc)

[64]:
```
total_rows = len(df)
total_trips = len(df_aggr2)

for col in ['s_state', 's_city']:
    print('\n' + str(df_aggr2.groupby(col)[col].agg(lambda x: np.round((len(x)␣
↪* 100 / total_trips),2)).sort_values(ascending=False).head(5)))
```

```
s_state
maharashtra    18.10
karnataka      15.04
haryana        11.37
tamilnadu       7.32
delhi           5.35
Name: s_state, dtype: float64

s_city
bengaluru    11.95
gurgaon       6.91
bhiwandi      5.47
delhi         4.18
mumbai        3.91
```

```
Name: s_city, dtype: float64
```

Observations

1. Top 3 states where the most number of orders originate are Maharashtra(18%), Karnataka(15%), and Haryana(11.37%).
2. Top 3 cities where the most number of orders originate are Benguluru(11.95%), Gurgaon(6.91%), and Bhiwandi(5.47%).

Recommendations

1. Maharashtra, Karnataka, and Haryana are the top three states shipping 18%, 15%, and 11% of the total shipment orders. The business should continue to invest resources in these states as they contribute significantly to the overall revenues.
2. Bangaluru, Gurgaon, and Haryana are the top three cities shipping 11.95%, 6.91%, and 5.45% of the total shipment orders. The business should continue to invest resources in these cities as they contribute significantly to the overall revenues.

2.13.2 Check where the most orders are delivered (State, city etc)

[65]:
```
for col in ['d_state', 'd_city']:
    print(f'\n {df_aggr2.groupby(col)[col].agg(lambda x: np.round((len(x) * 100␣
↪/ total_trips),2)).sort_values(ascending=False).head(3)}')
```

```
 d_state
maharashtra    17.49
karnataka      15.35
haryana        11.25
Name: d_state, dtype: float64

 d_city
bengaluru    11.49
mumbai        6.01
gurgaon       5.86
Name: d_city, dtype: float64
```

Observations

1. Top 3 states where the most number of orders are delivered are Maharashtra(17.49%), Karnataka(15.35%), and Haryana(11.25%).
2. Top 3 cities where the most number of orders are delivered are Bengaluru(11.49%), Mumbai(6%), and Gurgaon(5.86%)

Recommendations

1. Maharashtra, Karnataka, and Haryana are the top three states receiving 17.49%, 15.35%, and 11% of the total shipments delivered. The business should continue to invest resources in these states as they contribute significantly to the overall revenues.

2. Bangaluru, Gurgaon, and Haryana are the top three cities receiving 11.45%, 6%, and 5.86% of the total shipments delivered. The business should continue to invest resources in these cities as they contribute significantly to the overall revenues.

2.13.3 Busiest, longest, and slowest inter-state corridors

Note: We define an inter-state corridor as a combination of a source state and a destination state such that both of them are different. They represent one-way corridor. This means that (Maharashtra to Gujarat) is different from (Gujarat to Maharashtra).

[66]:
```
#we use segment osrm times and distances while aggregating at inter-state level.
#inter-state
filter = (df['s_state'] != df['d_state']) & (df['s_state'] != 'na') &␣
↪(df['d_state'] != 'na')
df_interstate = df[filter].groupby(['s_state', 'd_state']).
↪agg(count=('d_state', 'count'), avg_distance_osrm=('segment_osrm_distance',␣
↪np.mean), avg_time_osrm=('segment_osrm_time', np.mean)).reset_index()

res = df_interstate.sort_values(by=['count'])[['s_state', 'd_state', 'count',␣
↪'avg_distance_osrm', 'avg_time_osrm']]
print('\nBusiest inter-state corridors')
print(res.tail(3).sort_values(by='count', ascending=False))

res = df_interstate.sort_values(by=['avg_distance_osrm'])[['s_state',␣
↪'d_state', 'count', 'avg_distance_osrm', 'avg_time_osrm']]
print('\ninter-state corridors with longest average distance')
print(res.tail(3).sort_values(by='avg_distance_osrm', ascending=False))

res = df_interstate.sort_values(by=['avg_time_osrm'])[['s_state', 'd_state',␣
↪'count', 'avg_distance_osrm', 'avg_time_osrm']]
print('\ninter-state corridors with longest average time')
print(res.tail(3).sort_values(by='avg_time_osrm', ascending=False))
```

```
Busiest inter-state corridors
        s_state    d_state  count  avg_distance_osrm  avg_time_osrm
37      haryana  karnataka   4976          30.352581      23.066519
58    karnataka    haryana   3394          30.660306      23.691514
78  maharashtra    haryana   2934          28.669487      21.880709

inter-state corridors with longest average distance
        s_state     d_state  count  avg_distance_osrm  avg_time_osrm
121  westbengal       assam    203          44.466977      31.674877
7         assam   meghalaya     33          42.667479      33.151515
11        assam  westbengal    149          41.793932      29.577181

inter-state corridors with longest average time
       s_state    d_state  count  avg_distance_osrm  avg_time_osrm
```

```
105  tamilnadu     kerala     16            30.478794      39.062500
68      kerala  tamilnadu     45            28.830993      33.666667
7        assam  meghalaya     33            42.667479      33.151515
```

Recommendations

1. (Hariyana to Karnataka), (Karnataka to Hariyana), and (Maharashtra to Karnataka) are the top three busiest inter-state corridors. The business should continue to invest resources on these corridors.
2. (westbengal to assam), (assam to meghalaya), and (assam to westbengal) are the top three inter-state corridors covering maximum average distance (between checkpoints). The business/logistics team can consider adding more check-points if needed.
3. (tamilnadu to kerala), (kerala to tamilnadu), and (assam to meghalaya) are top three corridors with maxium average time taken (between checkpoints). The business/logistics team can consider adding more check-points if needed.

2.13.4 Busiest, longest, and slowest inter-city corridors

Note: We define an inter-city corridor as a combination of a source city and a destination city such that both of them are different. They represent one-way corridor. This means that (Mumbai to Pune) is different from (Pune to Mumbai).

[67]:
```
filter = (df['s_city'] != df['d_city']) & (df['s_city'] != 'na') &␣
 ↪(df['d_city'] != 'na')
df_intercity = df[filter].groupby(['s_city', 'd_city']).agg(count=('d_city',␣
 ↪'count'), avg_distance_osrm=('segment_osrm_distance', np.mean),␣
 ↪avg_time_osrm=('segment_osrm_time', np.mean)).reset_index()

res = df_intercity.sort_values(by=['count'])[['s_city', 'd_city', 'count',␣
 ↪'avg_distance_osrm', 'avg_time_osrm']]
print('\nBusiest inter-city corridors')
print(res.tail(3).sort_values(by='count', ascending=False))

res = df_intercity.sort_values(by=['avg_distance_osrm'])[['s_city', 'd_city',␣
 ↪'count', 'avg_distance_osrm', 'avg_time_osrm']]
print('\ninter-city corridors with longest average distance')
print(res.tail(3).sort_values(by='avg_distance_osrm', ascending=False))

res = df_intercity.sort_values(by=['avg_time_osrm'])[['s_city', 'd_city',␣
 ↪'count', 'avg_distance_osrm', 'avg_time_osrm']]
print('\ninter-city corridors with longest average time')
print(res.tail(3).sort_values(by='avg_time_osrm', ascending=False))
```

```
Busiest inter-city corridors
        s_city     d_city  count  avg_distance_osrm  avg_time_osrm
761    gurgaon  bengaluru   4976          30.352581      23.066519
235  bengaluru    gurgaon   3394          30.660306      23.691514
```

```
783     gurgaon     kolkata   2862          26.755446      18.998602

inter-city corridors with longest average distance
         s_city      d_city  count  avg_distance_osrm  avg_time_osrm
1278   lalitpur     gwalior      1           223.2655          208.0
1531  nandurbar       dhule      1           109.1615           79.0
2014      solan  chandigarh      1           101.7296           95.0

inter-city corridors with longest average time
         s_city      d_city  count  avg_distance_osrm  avg_time_osrm
1278   lalitpur     gwalior      1           223.2655          208.0
2014      solan  chandigarh      1           101.7296           95.0
1531  nandurbar       dhule      1           109.1615           79.0
```

Recommendations

1. (Gurgaon to Bengaluru), (Bengaluru to Gurgaon), and (Gurgaon to kolkata) are the top three busiest inter-state corridors. The business should continue to invest resources on these corridors.
2. (lalitpur to gwalior), (nandurbar to dhule), and (solan to chandigarh) are the top three inter-state corridors covering maximum average distance (between checkpoints). On the other hand, (lalitpur to gwalior), (solan to chandigarh), and (nandurbar to dhule) are top three corridors with maximum average time taken (between checkpoints). The business/logistics team can consider adding more check-points if needed.

2.13.5 Inter-city corridors experiencing maxium average delay

we consider the difference between the segment_actual_time and segment_osrm_time as the delay.

[68]:
```python
df_temp = df.copy()
filter = (df_temp['s_city'] != df_temp['d_city']) & (df_temp['s_city'] != 'na')␣
↪& (df_temp['d_city'] != 'na')
df_temp['delay_time'] = df_temp['segment_actual_time'] -␣
↪df_temp['segment_osrm_time']
df_intercity = df_temp[filter].groupby(['s_city', 'd_city']).
↪agg(avg_delay=('delay_time', np.mean)).reset_index()
df_intercity.sort_values(by='avg_delay', ascending=False).head()
```

[68]:

	s_city	d_city	avg_delay
1164	khatra	hura	1251.0
855	helencha	kolkata	1059.8
1531	nandurbar	dhule	1014.0
1875	sakri	dhule	884.5
1932	shahada	dhule	820.0

Recommendation

(Khatra to hura), (helencha to kolkata), and (nadurbar to dhule) are top three inter-city corridors with maxium delays. If these corridors have potential to generate more demands, the business should consider investing more resources to improve delivery time.

```
[ ]:
```