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.

Problem Statement:

1. Clean, sanitize and manipulate data to get useful features out of raw fields
2. Make sense out of the raw data and help the data science team to build forecasting models on it

In [1]:
```
# Importing required packages for analysis
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from scipy.stats import chi2_contingency,ttest_ind,pearsonr,normaltest
import re
```

In [2]:
```
# Initial pandas & matplotlib setup
pd.options.display.max_rows = 50
pd.options.display.max_columns = 50
np.set_printoptions(precision=2,suppress=True)
pd.options.display.max_colwidth = 3000
sns.set_palette("muted")
```

In [3]:
```
# To increase jupyter notebook cell width
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [73]:
```
# To plot clear graphs
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
```

In [7]:
```
# Importing the given dataset to pandas dataframe
data = pd.read_csv("./delhivery_data.txt")
df = data.copy()
df.head(9)
```

Out[7]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797

Insights:

1. It is apparent that there are more than 1 row for each trip_uuid. In the above example, first 5 records have same source_center, destination_center, od_start_time, and od_end_time. However, value in columns with prefix as segment has different values.
2. Way to understand it as follows -- For a deivery to move from source_center to destination_center, there are 5 segments in between and from data shown above, we can calculate the total time/distance by aggregating segment times, and segment distances. Therefore, first 5 records can be aggregated to 1 record
3. Same can be done with last 4 records. Then, we can club the resultant 2 records as one by grouping on trip_uuid. - It means that the record is about a delivery from source_center as Anand_VUNagar_DC and destination_center as Anand_Vaghasi_IP
4. Therefore, we need to group and aggregate the data to perform the analysis and glean insights from it.

In [8]:
```
# To get the shape of the dataset
print(f"Number of records : {df.shape[0]}")
print(f"Total Features:  {df.shape[1]}")
```

```
Number of records : 144867
Total Features:  24
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  object
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
```

```
 8   destination_name               144606 non-null  object
 9   od_start_time                  144867 non-null  object
 10  od_end_time                    144867 non-null  object
 11  start_scan_to_end_scan         144867 non-null  float64
 12  is_cutoff                      144867 non-null  bool
 13  cutoff_factor                  144867 non-null  int64
 14  cutoff_timestamp               144867 non-null  object
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                    144867 non-null  float64
 17  osrm_time                      144867 non-null  float64
 18  osrm_distance                  144867 non-null  float64
 19  factor                         144867 non-null  float64
 20  segment_actual_time            144867 non-null  float64
 21  segment_osrm_time              144867 non-null  float64
 22  segment_osrm_distance          144867 non-null  float64
 23  segment_factor                 144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

Insights:

1. Except for the features like trip_creation_time, od_start_time, od_end_time, and cutoff_timestamp, all other features have correct datatypes
2. There are 24 features with ~1.5 lakh records in the dataset
3. Except for the columns source_name, and destination_name, no other columns have missing data

In [10]:
```
# What percentage/proportion of data is missing
df.isna().sum()
```

Out[10]:
```
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
```

Insights:

1. Field source_name has 293 missing values and destination_name has 261 missing values. It might reduce even further after merging
2. Check and see if source_center mapped to null source_name have any non-null source_name in other rows. Same process with the destination_center and destination_name

In [11]:
```python
# Get the source_centers where source_name is null

condition = (df["source_name"].isna() == True)
source_centers = df.loc[condition, "source_center"].unique()
source_centers
```

Out[11]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [12]:
```python
# For the above source centers, are there any non-null source names ?
np.any(df.loc[df["source_center"].isin(source_centers),"source_name"].isna() == False)
```

Out[12]:
```
False
```

In [13]:
```python
# Get the destination_centers where destination_name is null

condition = (df["destination_name"].isna() == True)
destination_centers = df.loc[condition, "destination_center"].unique()
destination_centers
```

Out[13]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [14]:
```python
# For the above destination centers, are there any non-null destination names ?
np.any(df.loc[df["destination_center"].isin(destination_centers),"destination_name"].isna() == False)
```

Out[14]:
```
False
```

Insights:

1. None of the above listed source_centers/destination_centers have a non-null source_name/destination_name.
2. We can either drop/fill the rows with source_center/destination_center.

In [15]:
```python
# Changing datatypes of all columns with datetime data as discussed above
datetime_features = ["trip_creation_time",
                     "od_start_time",
                     "od_end_time",
                     "cutoff_timestamp",]

for feature in datetime_features:
    df[feature] = pd.to_datetime(df[feature])


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  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
```

```
 5   source_center                  144867 non-null  object
 6   source_name                    144574 non-null  object
 7   destination_center             144867 non-null  object
 8   destination_name               144606 non-null  object
 9   od_start_time                  144867 non-null  datetime64[ns]
 10  od_end_time                    144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan         144867 non-null  float64
 12  is_cutoff                      144867 non-null  bool
 13  cutoff_factor                  144867 non-null  int64
 14  cutoff_timestamp               144867 non-null  datetime64[ns]
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                    144867 non-null  float64
 17  osrm_time                      144867 non-null  float64
 18  osrm_distance                  144867 non-null  float64
 19  factor                         144867 non-null  float64
 20  segment_actual_time            144867 non-null  float64
 21  segment_osrm_time              144867 non-null  float64
 22  segment_osrm_distance          144867 non-null  float64
 23  segment_factor                 144867 non-null  float64
dtypes: bool(1), datetime64[ns](4), float64(10), int64(1), object(8)
memory usage: 25.6+ MB
```

In [16]:

```python
# No duplicate records in the dataset
df.loc[df.duplicated()].sum(numeric_only=True).sum()
```

Out[16]:

```
0.0
```

Insights:

1. There are no duplicates in the dataset
2. As shown below, cutoff_factor and actual_distance_to_destination are highly correlated, therefore it can be dropped

In [17]:

```python
# Null Hypothesis, H0: cutoff_factor and actual_distance_to_destination are not correlated
# Alternate Hypothesis, Ha: cutoff_factor and actual_distance_to_destination are correlated

teststatistic, pvalue = pearsonr(x=df["cutoff_factor"], y = df["actual_distance_to_destination"],)

print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")


print()
print("----------------------------XXX----------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX----------------------------

Hypothesis test performed:  pearsonr
```

```
TestStatistic:1.0, Pvalue:0.0
Reject H0. Two features are correlated
```

In [18]:
```
# Dropping features that are unknown or highly correlated (cutoff_factor)
to_be_dropped = ['cutoff_timestamp','factor','segment_factor',"cutoff_factor","is_cutoff"]
df.drop(columns=to_be_dropped,inplace=True)
```

In [19]:
```
# Create lists of Categorical, datetime, & Numerical features
cat_cols = df.select_dtypes(include=["object"]).columns.tolist()
num_cols = df.select_dtypes(include=["int","float"]).columns.tolist()
date_cols =  df.select_dtypes(include=["datetime"]).columns.tolist()

print(f"Categorical Columns: {cat_cols}")
print(f"Datetime Columns: {date_cols}")
print(f"Numerical Columns: {num_cols}")
```

```
Categorical Columns: ['data', 'route_schedule_uuid', 'route_type', 'trip_uuid', 'source_center', 'source_name', 'destination_center', 'destination_name']
Datetime Columns: ['trip_creation_time', 'od_start_time', 'od_end_time']
Numerical Columns: ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time', 'segme
nt_osrm_time', 'segment_osrm_distance']
```

In [20]:
```
# Create a new column with just date portion
df["trip_creation_date"] = df["trip_creation_time"].dt.date

min_trip_date = df["trip_creation_date"].min()
max_trip_date = df["trip_creation_date"].max()

print(min_trip_date.strftime("%d %B %Y"), "till", max_trip_date.strftime("%d %B %Y"))
```

```
12 September 2018 till 03 October 2018
```

Insights:

1. We have data related to the trips created from September 12, 2018 till October 3,2018
2. All the features are segregated into different groups based on their datatypes

In [21]:
```
def print_unique_values_and_counts(cols_list,df):

    """
    Given a list of columns & dataframe, print unique values and counts

    """
    print("Unique Values & Unique Value Counts:")
    print()

    for column in cols_list:
        print(f"{column} :\n Unique Values: {df[column].unique()},\n Unique Value Counts: {df[column].nunique()}")
        print("----------------------------XXX----------------------------")
        print()

    return

# Calling the above function with categorical columns list & data
print_unique_values_and_counts(cat_cols,df)
```

```
Unique Values & Unique Value Counts:

data :
 Unique Values: ['training' 'test'],
 Unique Value Counts: 2
```

```
----------------------------XXX----------------------------

route_schedule_uuid :
 Unique Values: ['thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef'
 'thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc211728881b'
 'thanos::sroute:a16bfa03-3462-4bce-9c82-5784c7d315e6' ...
 'thanos::sroute:72cf9feb-f4e3-4a55-b92a-0b686ee8fabc'
 'thanos::sroute:5e08be79-8a4c-4a91-a514-5350403c0e31'
 'thanos::sroute:a3c30562-87e5-471c-9646-0ed49c150996'],
 Unique Value Counts: 1504
----------------------------XXX----------------------------

route_type :
 Unique Values: ['Carting' 'FTL'],
 Unique Value Counts: 2
----------------------------XXX----------------------------

trip_uuid :
 Unique Values: ['trip-153741093647649320' 'trip-153768492602129387'
 'trip-153693976643699843' ... 'trip-153761584139918815'
 'trip-153718412883843340' 'trip-153746066843555182'],
 Unique Value Counts: 14817
----------------------------XXX----------------------------

source_center :
 Unique Values: ['IND388121AAA' 'IND388620AAB' 'IND421302AAG' ... 'IND361335AAA'
 'IND562132AAC' 'IND639104AAB'],
 Unique Value Counts: 1508
----------------------------XXX----------------------------

source_name :
 Unique Values: ['Anand_VUNagar_DC (Gujarat)' 'Khambhat_MotvdDPP_D (Gujarat)'
 'Bhiwandi_Mankoli_HB (Maharashtra)' ... 'Dwarka_StnRoad_DC (Gujarat)'
 'Bengaluru_Nelmngla_L (Karnataka)' 'Kulithalai_AnnaNGR_D (Tamil Nadu)'],
 Unique Value Counts: 1498
----------------------------XXX----------------------------

destination_center :
 Unique Values: ['IND388620AAB' 'IND388320AAA' 'IND411033AAA' ... 'IND600004AAA'
 'IND134203AAA' 'IND400701AAA'],
 Unique Value Counts: 1481
----------------------------XXX----------------------------

destination_name :
 Unique Values: ['Khambhat_MotvdDPP_D (Gujarat)' 'Anand_Vaghasi_IP (Gujarat)'
 'Pune_Tathawde_H (Maharashtra)' ... 'Chennai_Mylapore (Tamil Nadu)'
 'Naraingarh_Ward2DPP_D (Haryana)' 'Mumbai_Ghansoli_DC (Maharashtra)'],
 Unique Value Counts: 1468
----------------------------XXX----------------------------
```

Insights:

1. There are two types of data - training, and test as expected
2. Two different route types as expected - carting and FTL/Full Truck Load
3. There are on 14817 trips based on trip_uuid. Each trip is separated into multiple segments
4. Trips have started from 1508 different centres and delivered to 1481 different centers
5. Source name, source center counts do not match. 1508 and 1498. Either there are same source names for different source centers or there is incorrect data
6. Same as point 5, could be said for destination name, destination center
7. There are 1504 different route schedules available

In [22]:
```
df.describe(include=[int,float])
```

Out[22]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.00000
mean	961.262986	234.073372	416.927527	213.868272	284.771297	36.196111	18.507548	22.82902
std	1037.012769	344.990009	598.103621	308.011085	421.119294	53.571158	14.775960	17.86066
min	20.000000	9.000045	9.000000	6.000000	9.008200	-244.000000	0.000000	0.00000
25%	161.000000	23.355874	51.000000	27.000000	29.914700	20.000000	11.000000	12.07010
50%	449.000000	66.126571	132.000000	64.000000	78.525800	29.000000	17.000000	23.51300
75%	1634.000000	286.708875	513.000000	257.000000	343.193250	40.000000	22.000000	27.81325
max	7898.000000	1927.447705	4532.000000	1686.000000	2326.199100	3051.000000	1611.000000	2191.40370

Insights:

1. Comparing Means & Medians of the numerical columns, it is evident that there are outliers. Majorly in start_scan_to_end_scan, cutoff_factor, actual_distance_to_destination, actual_time
2. Other columns excluding above have outliers too but may be not to the extent of columns in point 1
3. 75 percent of values in start_scan_to_end_scan column are under 1634 minutes with max value at 7898 minutes
4. Column cutoff_factor and actual_distance_to_destination seem to have same mean and median values, including quartiles. Is cutoff_factor same as actual_distance_to_destination ?
5. segment_actual_time seem to have more outliers than segment_osrm_time based on mean and median values
6. No comments on other features like factor, segment_factor as very little is known about them

Merging/Aggregation:

1. Dataset is related to Trips of Delhivery. However, per above analysis, it is evident that each trip is broken into multiple segments.
2. Therefore, for analysis purposes, it is better to bring the data to Trip level and that can be done by aggregating the columns

In [23]:
```
# To merge the data, we have to aggregate different columns differently
# Therefore, it is better to create a hashmap that maps column with the function to aggregate with

map_col_to_func = {}

for column in df.columns:
    # lets map all columns to first function. Later change it as required
    map_col_to_func[column] = "first"

map_col_to_func
```

Out[23]:
```
{'data': 'first',
 'trip_creation_time': 'first',
 'route_schedule_uuid': 'first',
 'route_type': 'first',
 'trip_uuid': 'first',
 'source_center': 'first',
 'source_name': 'first',
 'destination_center': 'first',
 'destination_name': 'first',
 'od_start_time': 'first',
 'od_end_time': 'first',
```

```
 'start_scan_to_end_scan': 'first',
 'actual_distance_to_destination': 'first',
 'actual_time': 'first',
 'osrm_time': 'first',
 'osrm_distance': 'first',
 'segment_actual_time': 'first',
 'segment_osrm_time': 'first',
 'segment_osrm_distance': 'first',
 'trip_creation_date': 'first'}
```

In [24]:
```
# Create another hashmap to map features to correct functions based on our knowledge of columns
# For cumulative features like actual time, osrm_time, osrm_distance, it is enough to pick the last value

change_func_dict = {

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

    'actual_distance_to_destination': 'last',
    'actual_time' : 'last',
    'osrm_distance' : 'last',
    'osrm_time': 'last',

    'segment_actual_time' : 'sum',
    'segment_osrm_time' : 'sum',
    'segment_osrm_distance':  'sum'

}
```

In [25]:
```
# Using above dict, update map_col_to_func
map_col_to_func.update(change_func_dict)
map_col_to_func
```

Out[25]:
```
{'data': 'first',
 'trip_creation_time': 'first',
 'route_schedule_uuid': 'first',
 'route_type': 'first',
 'trip_uuid': 'first',
 'source_center': 'first',
 'source_name': 'first',
 'destination_center': 'last',
 'destination_name': 'last',
 'od_start_time': 'first',
 'od_end_time': 'last',
 'start_scan_to_end_scan': 'first',
 'actual_distance_to_destination': 'last',
 'actual_time': 'last',
 'osrm_time': 'last',
 'osrm_distance': 'last',
 'segment_actual_time': 'sum',
 'segment_osrm_time': 'sum',
 'segment_osrm_distance': 'sum',
 'trip_creation_date': 'first'}
```

In [26]:
```
# Looking at the data, it seems logical to group the trips by trip_uuid, source_center, and destination_center
# We can include od_start_time and od_end_time as well to to do the grouping.

by = ["trip_uuid","source_center","destination_center"]
trips_df = df.groupby(by=by, as_index= False, ).aggregate(map_col_to_func).copy()
```

Insights:

1. For features related to segment like segment_actual_time, segment_osrm_time, segment_osrm_distance - aggregation function selected is sum as we need the data for entire trip
2. For features like data, trip_creation_time, route_schedule_uuid, route_type, trip_uuid - it doesn't matter if we use first or last as aggregation function
3. As the name suggests, for source_center, source_name, destination_center, destination_name - first and last are aggregation functions respectively
4. od_start_time, od_end_time has data about entire trip and it is same for the entire trip. Therefore first function is used
5. For features that are cumulative like actual_time, osrm_time, osrm_distance - last function is used to get the total trip data

In [27]:

```
df.loc[df["trip_uuid"]=='trip-153741093647649320']
```

Out[27]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 04:47:45.236797

In [28]:

```
trips_df.loc[trips_df["trip_uuid"]=='trip-153741093647649320'].sort_values(by=["trip_uuid","od_start_time","od_end_time"])
```

Out[28]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_t
10374	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09 03:21:32.418

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_t
			fa3d5c3297ef							
10375	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09 04:47:45.236

Insights:

1. Shown above is an example of how data related to Trip trip-153741093647649320 is merged/aggregated to just two rows
2. As discussed in the initial analysis, let do another grouping using just trip_uuid to merge the data further and get it to trip level
3. Once that is done, segment related data is aggregated, let's rename the features accordingly

In [29]:

```
# Sort the data by trip, od_start_time, and od_end_time to not mess up the order
trips_df = trips_df.sort_values(by=["trip_uuid","od_start_time","od_end_time"]).reset_index(drop=True)
trips_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26368 entries, 0 to 26367
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            26368 non-null  object
 1   trip_creation_time              26368 non-null  datetime64[ns]
 2   route_schedule_uuid             26368 non-null  object
 3   route_type                      26368 non-null  object
 4   trip_uuid                       26368 non-null  object
 5   source_center                   26368 non-null  object
 6   source_name                     26302 non-null  object
 7   destination_center              26368 non-null  object
 8   destination_name                26287 non-null  object
 9   od_start_time                   26368 non-null  datetime64[ns]
 10  od_end_time                     26368 non-null  datetime64[ns]
 11  start_scan_to_end_scan          26368 non-null  float64
 12  actual_distance_to_destination  26368 non-null  float64
 13  actual_time                     26368 non-null  float64
 14  osrm_time                       26368 non-null  float64
 15  osrm_distance                   26368 non-null  float64
 16  segment_actual_time             26368 non-null  float64
 17  segment_osrm_time               26368 non-null  float64
 18  segment_osrm_distance           26368 non-null  float64
 19  trip_creation_date              26368 non-null  object
dtypes: datetime64[ns](3), float64(8), object(9)
memory usage: 4.0+ MB
```

In [30]:

```
# Change existing map_col_to_func hashmap for performing second aggregation


change_func_dict = {
    'start_scan_to_end_scan': 'sum',
    'actual_distance_to_destination': 'sum',
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
```

```
}


map_col_to_func.update(change_func_dict)
```

Out[30]:
```
{'data': 'first',
 'trip_creation_time': 'first',
 'route_schedule_uuid': 'first',
 'route_type': 'first',
 'trip_uuid': 'first',
 'source_center': 'first',
 'source_name': 'first',
 'destination_center': 'last',
 'destination_name': 'last',
 'od_start_time': 'first',
 'od_end_time': 'last',
 'start_scan_to_end_scan': 'sum',
 'actual_distance_to_destination': 'sum',
 'actual_time': 'sum',
 'osrm_time': 'sum',
 'osrm_distance': 'sum',
 'segment_actual_time': 'sum',
 'segment_osrm_time': 'sum',
 'segment_osrm_distance': 'sum',
 'trip_creation_date': 'first'}
```

In [31]:
```
# perform grouping at trip_uuid level and merge the columns
by =  "trip_uuid"
trips_df = trips_df.groupby(by=by,as_index= False).aggregate(map_col_to_func).copy()
```

In [32]:
```
# rename the columns
rename_map = {"segment_actual_time": "agg_segment_actual_time",
              "segment_osrm_time": "agg_segment_osrm_time",
              "segment_osrm_distance": "agg_segment_osrm_distance"
             }
trips_df.rename(rename_map,axis=1,inplace=True)
```

In [33]:
```
# As shown below, each trip now has just one record
# Below is the example of the trip that we analyzed earlier
trips_df.loc[trips_df["trip_uuid"]=='trip-153741093647649320']
```

Out[33]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time	o
5919	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388320AAA	Anand_Vaghasi_IP (Gujarat)	2018-09-20 03:21:32.418600	06:3

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14817 non-null  object
 1   trip_creation_time              14817 non-null  datetime64[ns]
 2   route_schedule_uuid             14817 non-null  object
 3   route_type                      14817 non-null  object
 4   trip_uuid                       14817 non-null  object
```

```
 5   source_center                  14817 non-null  object
 6   source_name                    14807 non-null  object
 7   destination_center             14817 non-null  object
 8   destination_name               14809 non-null  object
 9   od_start_time                  14817 non-null  datetime64[ns]
 10  od_end_time                    14817 non-null  datetime64[ns]
 11  start_scan_to_end_scan         14817 non-null  float64
 12  actual_distance_to_destination 14817 non-null  float64
 13  actual_time                    14817 non-null  float64
 14  osrm_time                      14817 non-null  float64
 15  osrm_distance                  14817 non-null  float64
 16  agg_segment_actual_time        14817 non-null  float64
 17  agg_segment_osrm_time          14817 non-null  float64
 18  agg_segment_osrm_distance      14817 non-null  float64
 19  trip_creation_date             14817 non-null  object
dtypes: datetime64[ns](3), float64(8), object(9)
memory usage: 2.3+ MB
```

Data Cleaning:

As the proportion of null values is close to a quarter percent of trips data, we can safely drop them.

In [35]:
```
# Check to see the proportion of null valus in source_name, destination_name columns
trips_df.isna().sum(numeric_only=True)*100/trips_df.shape[0]
```

Out[35]:
```
data                              0.000000
trip_creation_time                0.000000
route_schedule_uuid               0.000000
route_type                        0.000000
trip_uuid                         0.000000
source_center                     0.000000
source_name                       0.067490
destination_center                0.000000
destination_name                  0.053992
od_start_time                     0.000000
od_end_time                       0.000000
start_scan_to_end_scan            0.000000
actual_distance_to_destination    0.000000
actual_time                       0.000000
osrm_time                         0.000000
osrm_distance                     0.000000
agg_segment_actual_time           0.000000
agg_segment_osrm_time             0.000000
agg_segment_osrm_distance         0.000000
trip_creation_date                0.000000
dtype: float64
```

In [36]:
```
# Dropping all the rows with either source_name or destination_name as null
trips_df.dropna(axis=0, inplace=True)
```

In [37]:
```
# Create/Update lists of Categorical, datetime, & Numerical features
cat_cols = trips_df.select_dtypes(include=["object"]).columns.tolist()
num_cols = trips_df.select_dtypes(include=["int","float"]).columns.tolist()
date_cols =  trips_df.select_dtypes(include=["datetime"]).columns.tolist()

print(f"Categorical Columns: {cat_cols}")
print(f"Datetime Columns: {date_cols}")
print(f"Numerical Columns: {num_cols}")
```

```
Categorical Columns: ['data', 'route_schedule_uuid', 'route_type', 'trip_uuid', 'source_center', 'source_name', 'destination_center', 'destination_name', 
'trip_creation_date']
Datetime Columns: ['trip_creation_time', 'od_start_time', 'od_end_time']
```

```
Numerical Columns: ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'agg_segment_actual_time', 'a
gg_segment_osrm_time', 'agg_segment_osrm_distance']
```

In [38]:

```python
from math import inf

def calculate_quartiles(df,column,unit= None,minCap=-inf,):

    """
    Given a dataframe and numerical column, calculate quartiles, IQR

    """
    quartile1 = np.percentile(df[column],25)
    quartile3 = np.percentile(df[column],75)
    IQR = quartile3-quartile1
    minimum = max(quartile1-1.5*IQR,minCap)
    maximum = quartile3+1.5*IQR

    print(f"Quartile 1:  {unit}{quartile1}")
    print(f"Quartile 3:  {unit}{quartile3}")
    print(f"IQR (Inter Quartile Range): {unit}{np.round(quartile3-quartile1)}")
    print(f"Minimum {column}: {unit}{minimum}\nMaximum {column}: {unit}{maximum}")

    return minimum, maximum
```

In [39]:

```python
# Calculating the quartiles and percentage of outliers

def calculate_outlier_stats(data,num_cols,minCap= 0):

    '''
        Given a dataframe and list of numerical columns, print outlier stats
        data : Dataframe
        num_cols: list of numerical columns
        minCap: Cap on lower_limit or Q1 - 1.5* IQR

    '''

    for column in num_cols:

        print(f"{column}:")
        print()

        lower_limit, upper_limit = calculate_quartiles(data,column,unit='',minCap= minCap)

        top_outliers_cnt = data.loc[data[column] > upper_limit,column].shape[0]
        bottom_outliers_cnt = data.loc[data[column] < lower_limit,column].shape[0]
        total_outliers_cnt = top_outliers_cnt + bottom_outliers_cnt

        outliers_percentage = total_outliers_cnt*100/data[column].shape[0]

        print(f"Total count of Outliers: {total_outliers_cnt} out of {data.shape[0]} records")
        print(f"Percentage of Outliers in the dataset: {np.round(outliers_percentage,2)}%")

        print()
        print("----------------------------XXX----------------------------")
        print()

calculate_outlier_stats(trips_df,num_cols)
```

```
start_scan_to_end_scan:
```

```
Quartile 1:  149.0
Quartile 3:  638.0
IQR (Inter Quartile Range): 489.0
Minimum start_scan_to_end_scan: 0
Maximum start_scan_to_end_scan: 1371.5
Total count of Outliers: 1261 out of 14800 records
Percentage of Outliers in the dataset: 8.52%

----------------------------XXX----------------------------

actual_distance_to_destination:

Quartile 1:  22.786366191115015
Quartile 3:  164.70555055540507
IQR (Inter Quartile Range): 142.0
Minimum actual_distance_to_destination: 0
Maximum actual_distance_to_destination: 377.58432710184013
Total count of Outliers: 1449 out of 14800 records
Percentage of Outliers in the dataset: 9.79%

----------------------------XXX----------------------------

actual_time:

Quartile 1:  67.0
Quartile 3:  370.0
IQR (Inter Quartile Range): 303.0
Minimum actual_time: 0
Maximum actual_time: 824.5
Total count of Outliers: 1642 out of 14800 records
Percentage of Outliers in the dataset: 11.09%

----------------------------XXX----------------------------

osrm_time:

Quartile 1:  29.0
Quartile 3:  168.25
IQR (Inter Quartile Range): 139.0
Minimum osrm_time: 0
Maximum osrm_time: 377.125
Total count of Outliers: 1515 out of 14800 records
Percentage of Outliers in the dataset: 10.24%

----------------------------XXX----------------------------

osrm_distance:

Quartile 1:  30.775025
Quartile 3:  208.63277499999998
IQR (Inter Quartile Range): 178.0
Minimum osrm_distance: 0
Maximum osrm_distance: 475.41939999999994
Total count of Outliers: 1524 out of 14800 records
Percentage of Outliers in the dataset: 10.3%

----------------------------XXX----------------------------

agg_segment_actual_time:

Quartile 1:  66.0
Quartile 3:  367.0
IQR (Inter Quartile Range): 301.0
Minimum agg_segment_actual_time: 0
```

```
Maximum agg_segment_actual_time: 818.5
Total count of Outliers: 1642 out of 14800 records
Percentage of Outliers in the dataset: 11.09%

----------------------------XXX---------------------------

agg_segment_osrm_time:

Quartile 1:  30.0
Quartile 3:  185.0
IQR (Inter Quartile Range): 155.0
Minimum agg_segment_osrm_time: 0
Maximum agg_segment_osrm_time: 417.5
Total count of Outliers: 1487 out of 14800 records
Percentage of Outliers in the dataset: 10.05%

----------------------------XXX---------------------------

agg_segment_osrm_distance:

Quartile 1:  32.6177
Quartile 3:  218.917675
IQR (Inter Quartile Range): 186.0
Minimum agg_segment_osrm_distance: 0
Maximum agg_segment_osrm_distance: 498.3676375
Total count of Outliers: 1544 out of 14800 records
Percentage of Outliers in the dataset: 10.43%

----------------------------XXX---------------------------
```

Insights:

1. As shown above, every numerical feature has ~10 percent of outliers. Therefore, cannot drop them
2. More exploration is required to see if the outliers are possible data points or real outliers
3. For this analysis, lets leave the outliers as is.

Feature Engineering:

In [40]:

```
# copy trips_df to df object for easier access
df = trips_df.copy()
```

In [41]:

```
# Create different features like date, day, weekday, hour, month, year, quarter for trip_creation_time
# These features could help us understand the frequency of trips

df["trip_creation_day"] = df["trip_creation_time"].dt.day
df["trip_creation_month"] = df["trip_creation_time"].dt.month
df["trip_creation_year"] = df["trip_creation_time"].dt.year
df["trip_creation_weekday"] = df["trip_creation_time"].dt.weekday
df["trip_creation_quarter"] = df["trip_creation_time"].dt.quarter
df["trip_creation_hour"] = df["trip_creation_time"].dt.hour
```

In [42]:

```
# Get 5 sample records
df.loc[:,["trip_creation_time","trip_creation_date","trip_creation_hour","trip_creation_day","trip_creation_weekday","trip_creation_month","trip_creation_
```

Out[42]:

	trip_creation_time	trip_creation_date	trip_creation_hour	trip_creation_day	trip_creation_weekday	trip_creation_month	trip_creation_year	trip_creation_quarter
2185	2018-09-14 23:33:58.459607	2018-09-14	23	14	4	9	2018	3

	trip_creation_time	trip_creation_date	trip_creation_hour	trip_creation_day	trip_creation_weekday	trip_creation_month	trip_creation_year	trip_creation_quarter
13299	2018-10-01 11:40:42.787446	2018-10-01	11	1	0	10	2018	4
6486	2018-09-20 23:42:47.099157	2018-09-20	23	20	3	9	2018	3
11985	2018-09-29 01:28:34.272131	2018-09-29	1	29	5	9	2018	3
3569	2018-09-16 22:54:13.479572	2018-09-16	22	16	6	9	2018	3

In [43]:

```
# Create new features called source_state & destination_state
state_pattern = re.compile(r'\((.*?)\)')

#Extract state using regex Pattern object
df["source_state"] =  df["source_name"].apply(lambda k: state_pattern.findall(k)[0])
df["destination_state"] =  df["destination_name"].apply(lambda k: state_pattern.findall(k)[0])
```

In [44]:

```
# Data looks good. No duplicates or repeats
df["source_state"].unique()
```

Out[44]:

```
array(['Madhya Pradesh', 'Karnataka', 'Maharashtra', 'Tamil Nadu',
       'Gujarat', 'Delhi', 'Haryana', 'Telangana', 'Rajasthan',
       'Uttar Pradesh', 'Assam', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Goa', 'Jharkhand', 'Pondicherry', 'Orissa',
       'Uttarakhand', 'Himachal Pradesh', 'Kerala', 'Arunachal Pradesh',
       'Bihar', 'Chandigarh', 'Chhattisgarh', 'Dadra and Nagar Haveli',
       'Jammu & Kashmir', 'Mizoram', 'Nagaland'], dtype=object)
```

In [45]:

```
# Data looks good. No duplicates or repeats
df["destination_state"].unique()
```

Out[45]:

```
array(['Haryana', 'Karnataka', 'Punjab', 'Maharashtra', 'Tamil Nadu',
       'Gujarat', 'Delhi', 'Telangana', 'Rajasthan', 'Madhya Pradesh',
       'Assam', 'Uttar Pradesh', 'West Bengal', 'Andhra Pradesh',
       'Dadra and Nagar Haveli', 'Orissa', 'Bihar', 'Jharkhand',
       'Pondicherry', 'Goa', 'Chandigarh', 'Uttarakhand',
       'Himachal Pradesh', 'Kerala', 'Arunachal Pradesh', 'Mizoram',
       'Chhattisgarh', 'Nagaland', 'Meghalaya', 'Jammu & Kashmir',
       'Tripura', 'Daman & Diu'], dtype=object)
```

In [46]:

```
# Get 5 sample records
df.loc[:,["source_name","source_state"]].sample(5)
```

Out[46]:

	source_name	source_state
2733	Kakinada_DC (Andhra Pradesh)	Andhra Pradesh
9779	Mumbai Hub (Maharashtra)	Maharashtra
4474	Delhi_Airport_H (Delhi)	Delhi
1230	Hyderabad_Tolichwk_I (Telangana)	Telangana
14656	Bangalore_Nelmngla_H (Karnataka)	Karnataka

In [47]:

```
# Get 5 sample records
df.loc[:,["destination_name","destination_state"]].sample(5)
```

Out[47]:

	destination_name	destination_state
5603	Mumbai_MiraRd_IP (Maharashtra)	Maharashtra

	destination_name	destination_state
3481	Gurgaon_Bilaspur_HB (Haryana)	Haryana
6742	Hyderabad_Shamshbd_H (Telangana)	Telangana
1694	Mumbai_Ulhasngr_DC (Maharashtra)	Maharashtra
11910	Chennai_Vandalur_Dc (Tamil Nadu)	Tamil Nadu

In [48]:

```
# Create new features called source_city & destination_city
df["source_city"] = df["source_name"].apply(lambda k: k.split("(")[0].strip().replace(" ","_").split("_",1)[0].strip())
df["destination_city"] = df["destination_name"].apply(lambda k: k.split("(")[0].strip().replace(" ","_").split("_",1)[0].strip())
```

In [49]:

```
# Replace Bengaluru with Bangalore, Hyd with Hyderabad etc

df["source_city"]= df["source_city"].str.replace("Bengaluru","Bangalore")
df["destination_city"] = df["destination_city"].str.replace("Bengaluru","Bangalore")

df["source_city"]= df["source_city"].str.replace(r"\bHyd\b","Hyderabad")
df["destination_city"] = df["destination_city"].str.replace(r"\bHyd\b","Hyderabad")

df["source_city"]= df["source_city"].str.replace("Amd","AMD")
df["destination_city"] = df["destination_city"].str.replace("Amd","AMD")
```

```
<ipython-input-49-3f3d21caed6a>:6: FutureWarning: The default value of regex will change from True to False in a future version.
  df["source_city"]= df["source_city"].str.replace(r"\bHyd\b","Hyderabad")
<ipython-input-49-3f3d21caed6a>:7: FutureWarning: The default value of regex will change from True to False in a future version.
  df["destination_city"] = df["destination_city"].str.replace(r"\bHyd\b","Hyderabad")
```

In [50]:

```
# Get 5 sample records
df.loc[:,["source_name","source_city"]].sample(5)
```

Out[50]:

	source_name	source_city
8544	Cjb_West_Dc (Tamil Nadu)	Cjb
14145	Delhi_Mayapuri_PC (Delhi)	Delhi
12013	Hyderabad_Shamshbd_H (Telangana)	Hyderabad
2697	Del_Okhla_PC (Delhi)	Del
3204	Gurgaon_Kadipur (Haryana)	Gurgaon

In [51]:

```
# Get 5 sample records
df.loc[:,["destination_name","destination_city"]].sample(5)
```

Out[51]:

	destination_name	destination_city
11542	Gokak_Bsavangr_D (Karnataka)	Gokak
11736	Mahasamund_RajpurRD_D (Chhattisgarh)	Mahasamund
1564	Gulbarga_Nehrugnj_I (Karnataka)	Gulbarga
10807	Srikakulam_Kuslpram_I (Andhra Pradesh)	Srikakulam
7654	Bhiwandi_Mankoli_HB (Maharashtra)	Bhiwandi

In [52]:
```python
def get_code(x):

    '''
    Given a source_name or destination_name string, splits it by separator '_'
    Based on the length and datatype of last element in the list, returns place_code

    '''

    temp = x.split("(")[0].split("_")

    if len(temp[-1].strip()) > 3:
        return np.NaN
    elif temp[-1].strip().upper() == "HUB":
        return np.NaN

    if temp[-1].strip().isnumeric() == True:
        return (temp[-2].strip() + str(temp[-1]).strip()).upper()
    else:
        return temp[-1].strip().upper()
```

In [53]:
```python
# Create new features called source_place_code & destination_place_code
# There are source and destination names where separator is not underscore
# or that do not have city or place or code - Extracted feature is not 100% clean

df["source_place_code"] = df["source_name"].apply(get_code)
df["destination_place_code"] = df["destination_name"].apply(get_code)
```

In [54]:
```python
# Get 5 sample records
df.loc[:,["source_name","source_place_code"]].sample(5)
```

Out[54]:

	source_name	source_place_code
14631	Nipani_AkkolRD_D (Karnataka)	D
12762	Bhadrak_Central_I_2 (Orissa)	I2
10365	CCU_Lake Avenue_DPC (West Bengal)	DPC
12784	Muzaffrngr_MhmodNgr_D (Uttar Pradesh)	D
540	Gurgaon_Bilaspur_HB (Haryana)	HB

In [55]:
```python
# Get 5 sample records
df.loc[:,["destination_name","destination_place_code"]].sample(5)
```

Out[55]:

	destination_name	destination_place_code
466	CCU_Lake Avenue_DPC (West Bengal)	DPC
12953	Bangalore_East_I_20 (Karnataka)	I20
7291	PNQ Vadgaon Sheri DPC (Maharashtra)	NaN
755	Delhi_Patparganj_DPC (Delhi)	DPC
7540	Medchal_MROoffce_D (Telangana)	D

In [56]:
```python
def get_place(x):
```

```
    '''
    Given a source_name or destination_name string, splits it by separator '_'
    Returns place

    '''
    pattern = re.compile(r"[_](.*?)[_]")
    temp = pattern.findall(x)

    if len(temp) == 0:
        return np.NaN
    else:
        return temp[0].strip()
```

In [57]:
```
# Create new features called source_place & destination_place
# There are source and destination names where separator is not underscore
# or that do not have city or place or code - Extracted feature is not 100% clean

df["source_place"] =  df["source_name"].apply(get_place)
df["destination_place"] = df["destination_name"].apply(get_place)
```

In [58]:
```
# Get 5 sample records
df.loc[:,["source_name","source_place"]].sample(5)
```

Out[58]:

	source_name	source_place
11430	BOM_Sakinaka_RP (Maharashtra)	Sakinaka
3524	Purnia_Central_H_2 (Bihar)	Central
545	Bangalore_Nelmngla_H (Karnataka)	Nelmngla
6945	Bengaluru_Hoodi_IP (Karnataka)	Hoodi
323	Delhi_Gateway_HB (Delhi)	Gateway

In [59]:
```
# Get 5 sample records
df.loc[:,["destination_name","destination_place"]].sample(5)
```

Out[59]:

	destination_name	destination_place
9699	Jaipur_Hub (Rajasthan)	NaN
344	Kanpur_Central_H_6 (Uttar Pradesh)	Central
2433	Bangalore_Nelmngla_H (Karnataka)	Nelmngla
9614	Delhi_Kishangarh_DPC (Delhi)	Kishangarh
1846	Hisar_IndstlAr_I (Haryana)	IndstlAr

In [60]:
```
# Create another feature by calculating minutes between od_start_time and od_end_time
df["od_start_end_time_diff"] = (df["od_end_time"]-df["od_start_time"]).dt.total_seconds()/60

# Drop od_start_time, and od_end_time
df.drop(columns=["od_start_time","od_end_time"],inplace=True)

# Add od_start_end_time_diff to num_cols list
num_cols.append("od_start_end_time_diff")
```

Insights:

1. Created multiple features like date, day, weekday, month, year, quarter, and hour from trip_creation_time column
2. From source_name, and destination_name columns, created features like city, place, code, and state as shown above
3. Features from source_name, and destination_name need lot of cleaning. Like mapping cities spelled little differently together - Benguluru & Bangalore, Hyd & Hyderbad for examples
4. Also, there are some source_names, destination*names with different separator like ' ' instead of *''. It will affect the features

In [61]:

```
# Top 5 destination states
df["destination_state"].value_counts().head(5)
```

Out[61]:

```
Maharashtra    2591
Karnataka      2275
Haryana        1667
Tamil Nadu     1072
Telangana       838
Name: destination_state, dtype: int64
```

In [62]:

```
# Top 5 source states
df["source_state"].value_counts().head(5)
```

Out[62]:

```
Maharashtra    2682
Karnataka      2229
Haryana        1681
Tamil Nadu     1085
Delhi           791
Name: source_state, dtype: int64
```

In [63]:

```
# Top 5 destination cities
df["destination_city"].value_counts().head(5)
```

Out[63]:

```
Bangalore    1702
Mumbai       1127
Gurgaon       869
Hyderabad     632
Bhiwandi      604
Name: destination_city, dtype: int64
```

In [64]:

```
# Top 5 source cities
df["source_city"].value_counts().head(5)
```

Out[64]:

```
Bangalore    1770
Gurgaon      1022
Mumbai        893
Bhiwandi      811
Delhi         618
Name: source_city, dtype: int64
```

In [65]:

```
# Between which states, majority of deliveries happen
df[["source_state","destination_state"]].value_counts().head(5)
```

Out[65]:

```
source_state  destination_state
Maharashtra   Maharashtra          2406
Karnataka     Karnataka            2015
Tamil Nadu    Tamil Nadu           1016
Haryana       Haryana               871
Telangana     Telangana             655
dtype: int64
```

In [66]:

```
# Between which cities majority of deliveries happen
df[["source_city","destination_city"]].value_counts().head(5)
```

Out[66]:
```
source_city  destination_city
Bangalore    Bangalore           1376
Mumbai       Mumbai               600
Bhiwandi     Mumbai               437
Hyderabad    Hyderabad            404
Mumbai       Bhiwandi             270
dtype: int64
```

Insights:

1. Top 5 busiest states are Karnataka, Maharastra, Tamilnadu, Haryana, and Uttar Pradesh
2. Top 5 busiest cities are Bangalore, Mumbai, Gurgaon, Bhiwandi, Hyderabad, Delhi
3. Maharashtra is the busisest corridor followed by Karnataka and TamilNadu
4. In cities, there are more deliveries across Bangalore, followed by Mumbai
5. Shown below is the mean time, and mean distance for the deliveries

In [67]:
```python
# To get the average time and average distance between busiest corridor/state
df.loc[(df["source_state"] == "Maharashtra") & (df["destination_state"] == "Maharashtra"),["actual_time","actual_distance_to_destination"]].mean()
```

Out[67]:
```
actual_time                       164.041563
actual_distance_to_destination     60.110614
dtype: float64
```

In [68]:
```python
# To get the average time and average distance between busiest corridor/city
df.loc[(df["source_city"] == "Bangalore") & (df["destination_city"] == "Bangalore"),["actual_time","actual_distance_to_destination"]].mean()
```

Out[68]:
```
actual_time                       99.492733
actual_distance_to_destination    37.211491
dtype: float64
```

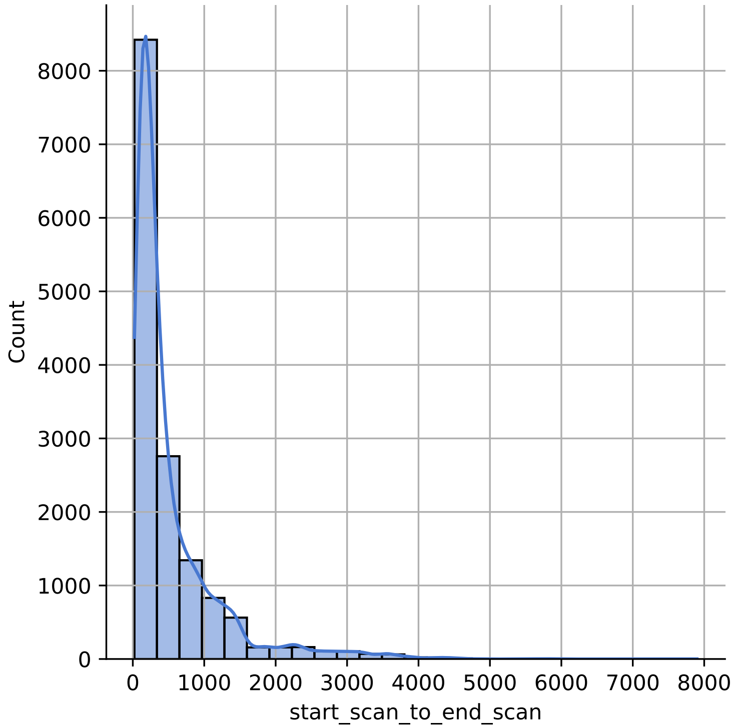
Insights:

1. As shown above, busiest state is Maharastra with average time of 164 minutes and average distance of 60 kms
2. Busiest city is Bangalore with average time of 100 minutes and average distance of 37 kms

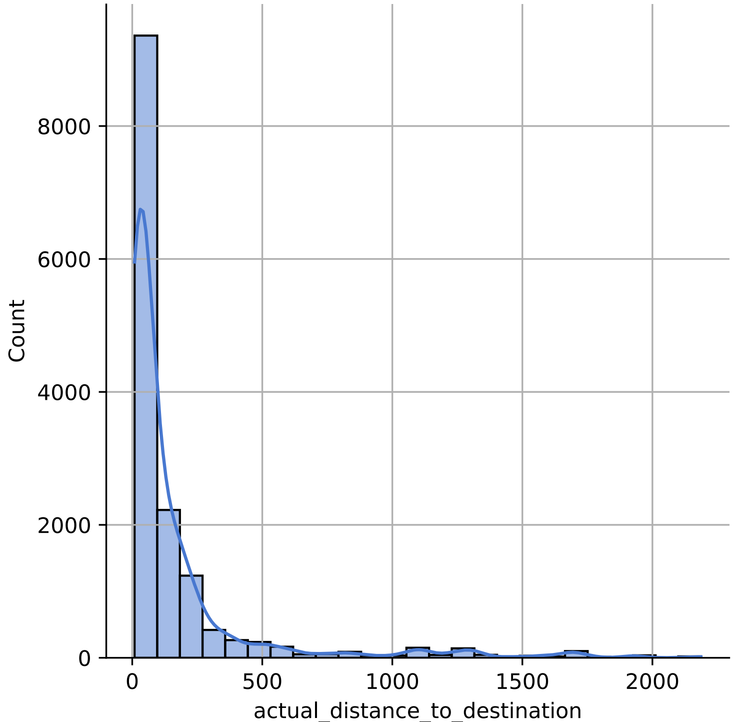
Visual Analysis

Univariate Analysis:

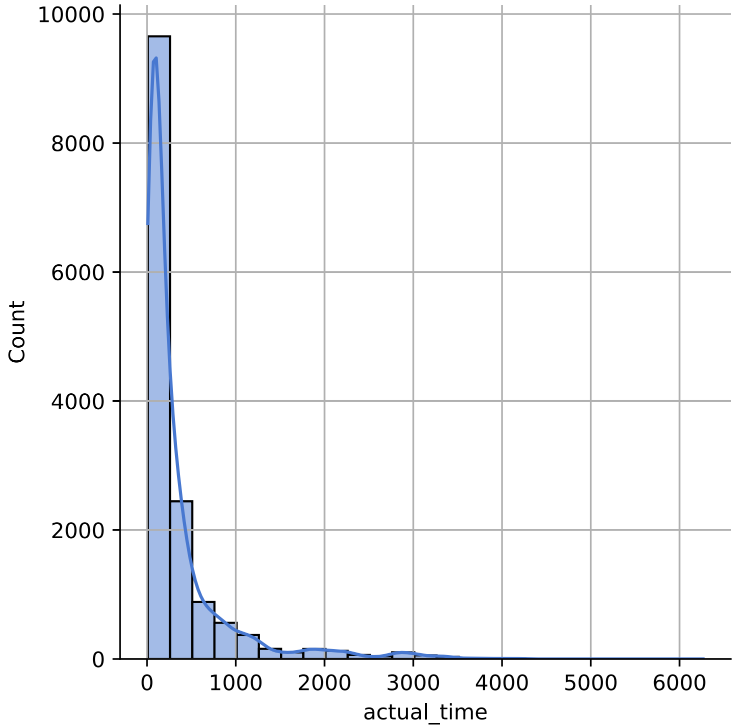
In [74]:
```python
# To plot the Kernel density distribution and histogram of numerical columns
for column in num_cols:
    sns.displot(df[column],kde=True,bins=25)
    plt.grid(True)
    plt.show()
```



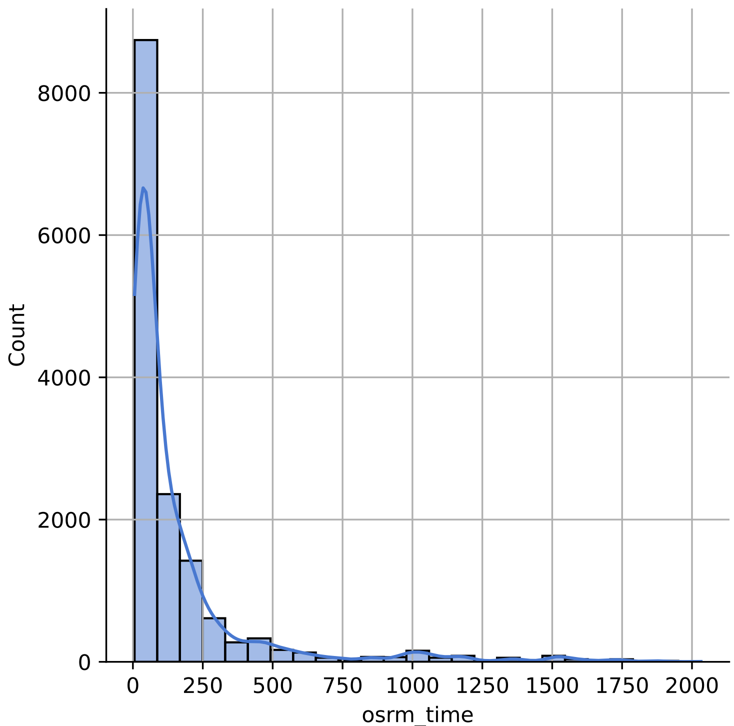
Count
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan



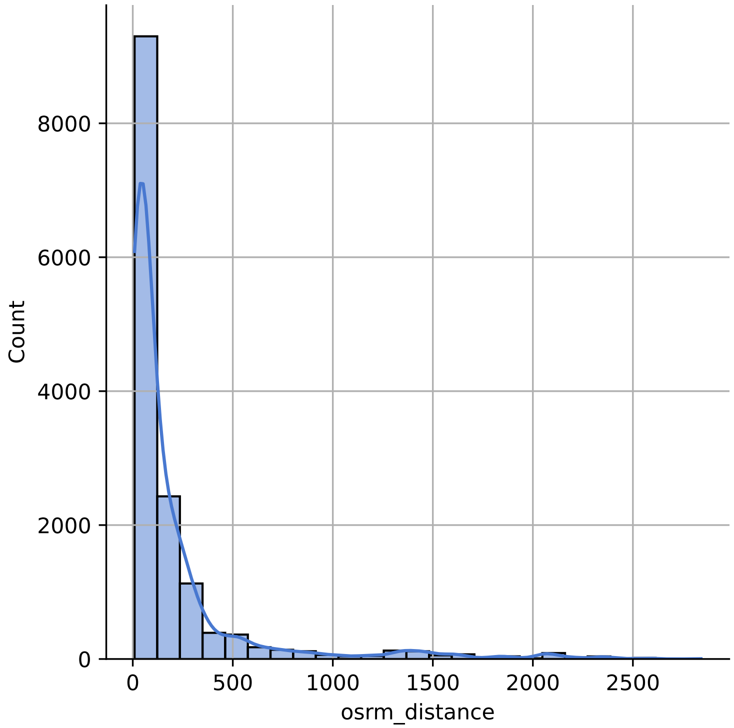
Count
8000
6000
4000
2000
0
0
500
1000
1500
2000
actual_distance_to_destination



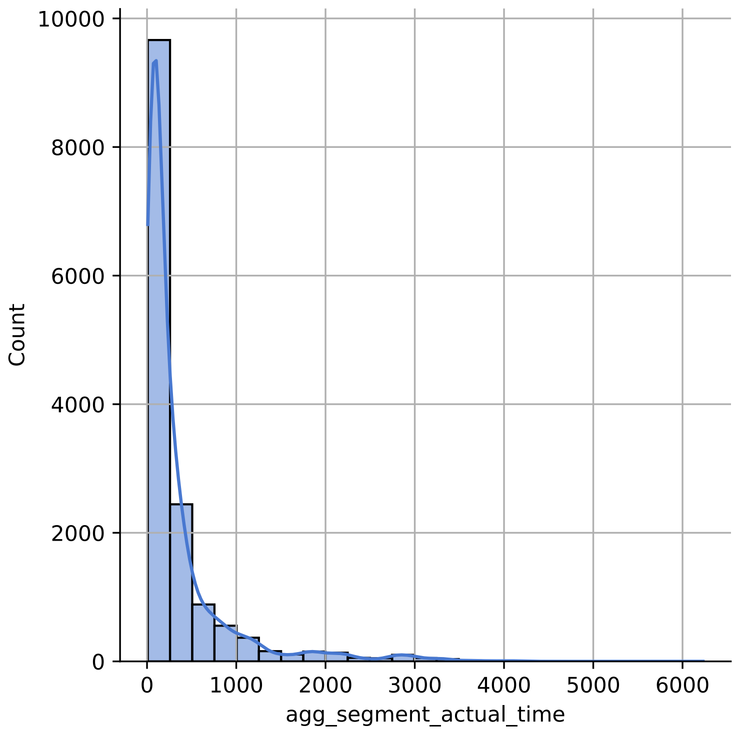
10000
8000
6000
4000
2000
0
Count
0
1000
2000
3000
4000
5000
6000
actual_time



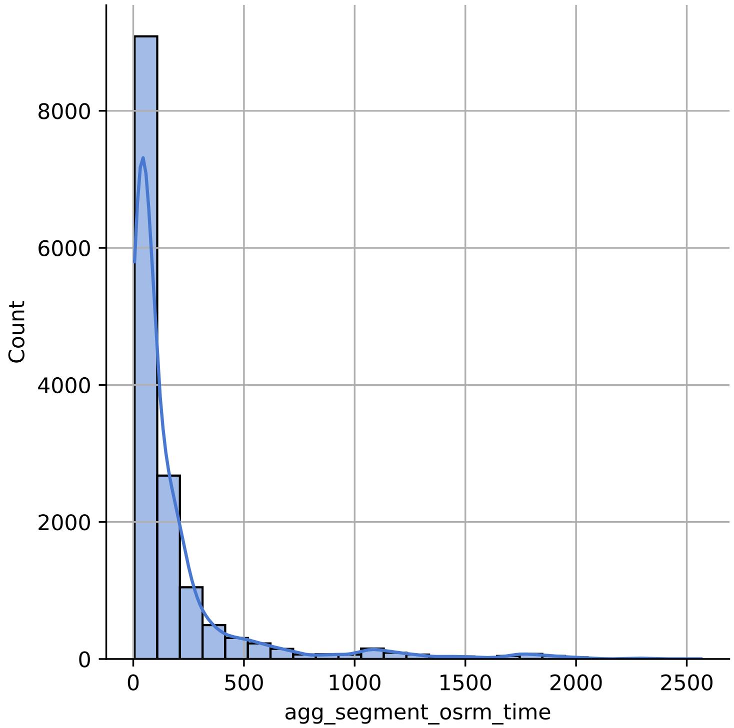
8000
6000
4000
2000
0
Count
0
250
500
750
1000
1250
1500
1750
2000
osrm_time



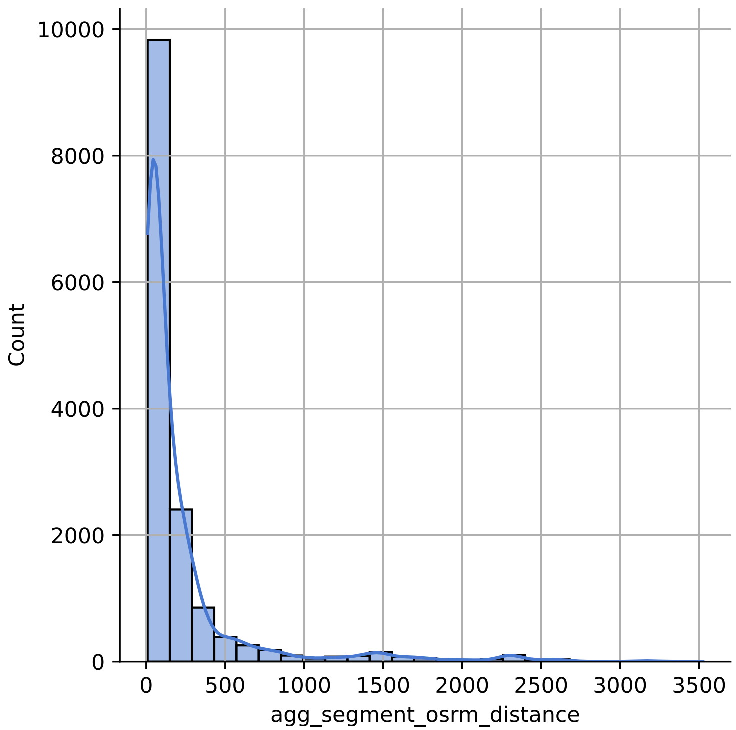
Count
8000
6000
4000
2000
0
0
500
1000
1500
2000
2500
osrm_distance



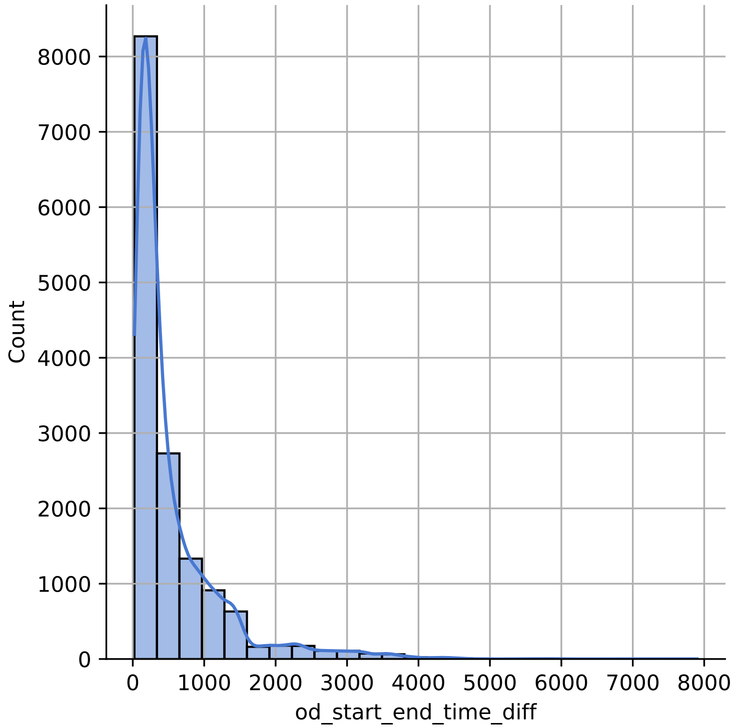
Count
10000
8000
6000
4000
2000
0
0
1000
2000
3000
4000
5000
6000
agg_segment_actual_time



Count
8000
6000
4000
2000
0
0
500
1000
1500
2000
2500
agg_segment_osrm_time



Count
10000
8000
6000
4000
2000
0
0
500
1000
1500
2000
2500
3000
3500
agg_segment_osrm_distance

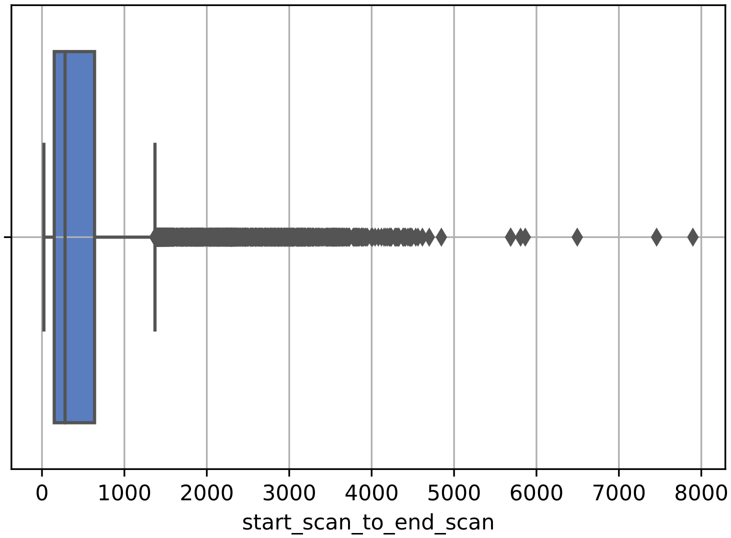


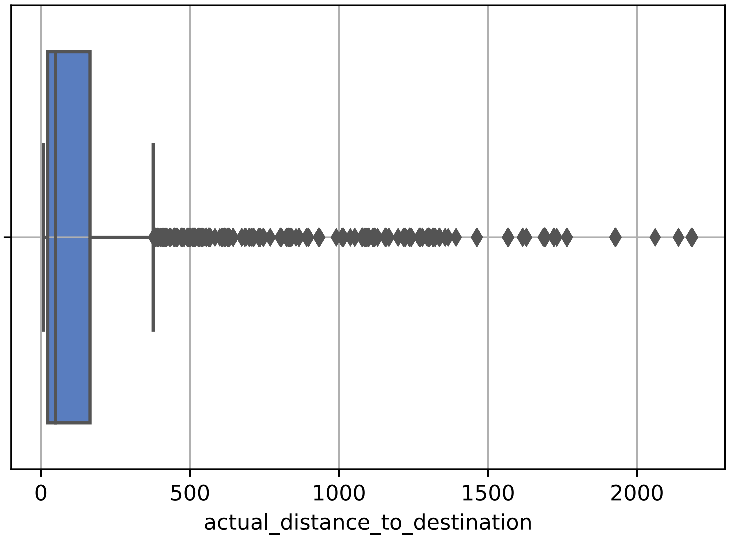
Insights:

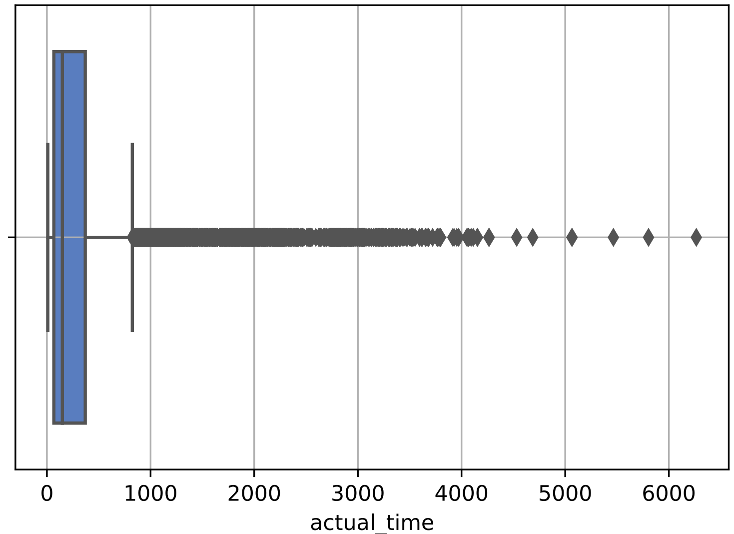
1. All numerical features have positive/right skewed distributions. Therefore, it is necessary to test normality, and variances before proceeding with any hypothesis tests (to infer about population)

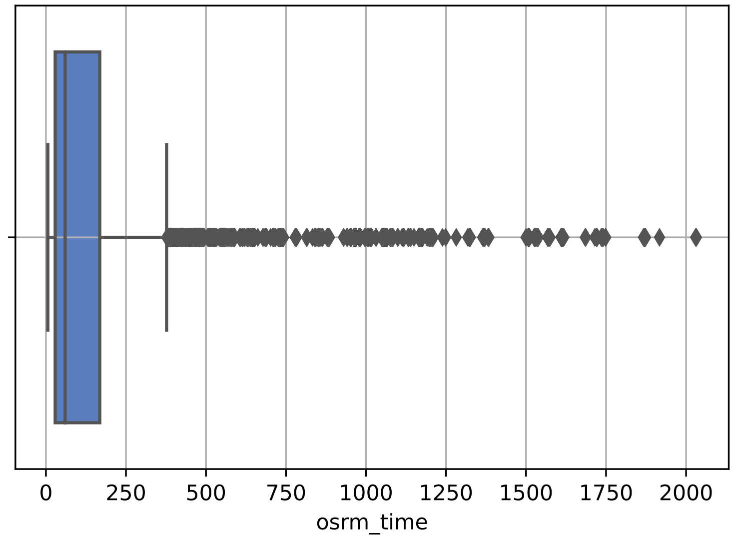
In [75]:

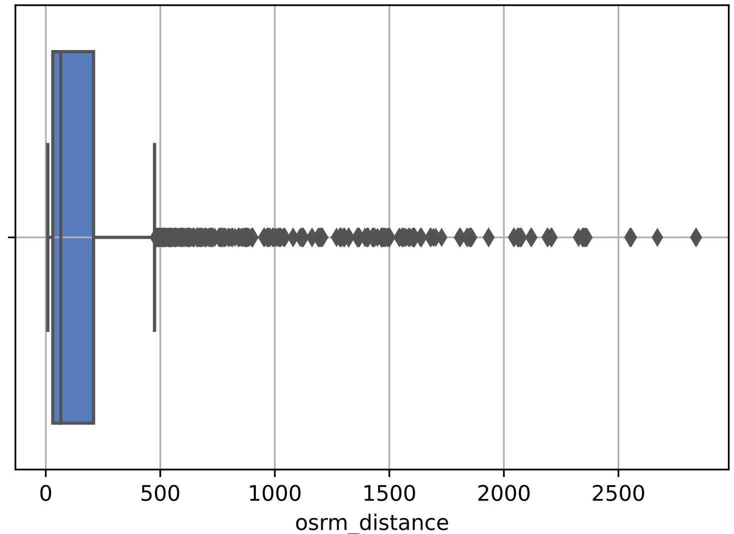
```
# Generate box plots for all numerical columns

for column in num_cols:
    sns.boxplot(data=df,x=column)
    plt.xlabel(column)
    plt.grid(True)
    plt.show()
```

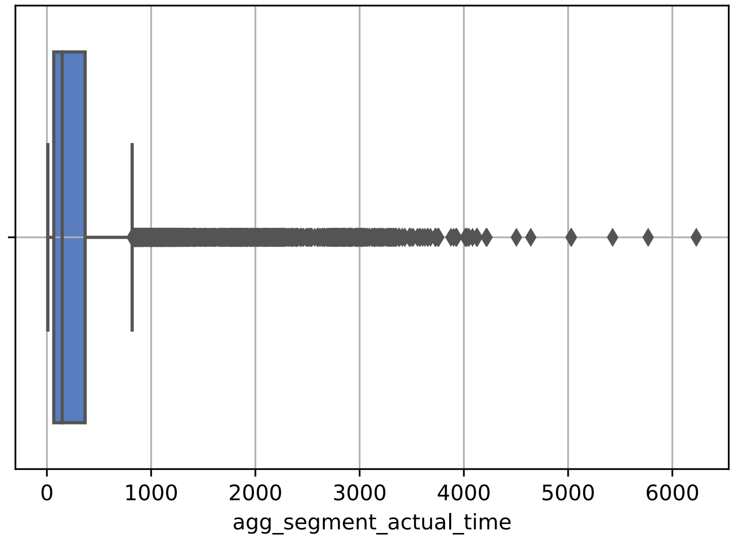

0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan

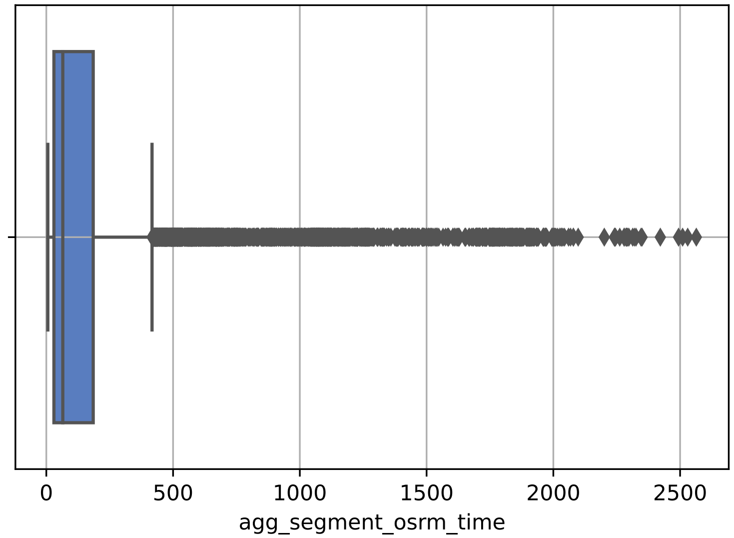

0
500
1000
1500
2000
actual_distance_to_destination

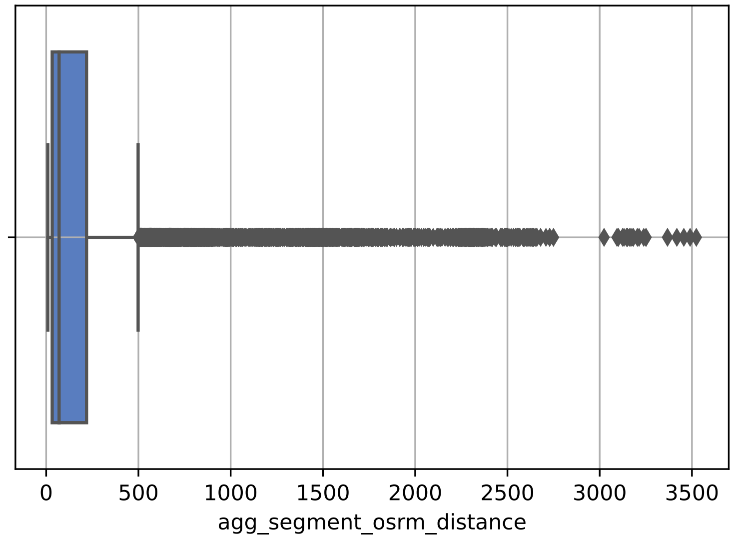

0
1000
2000
3000
4000
5000
6000
actual_time

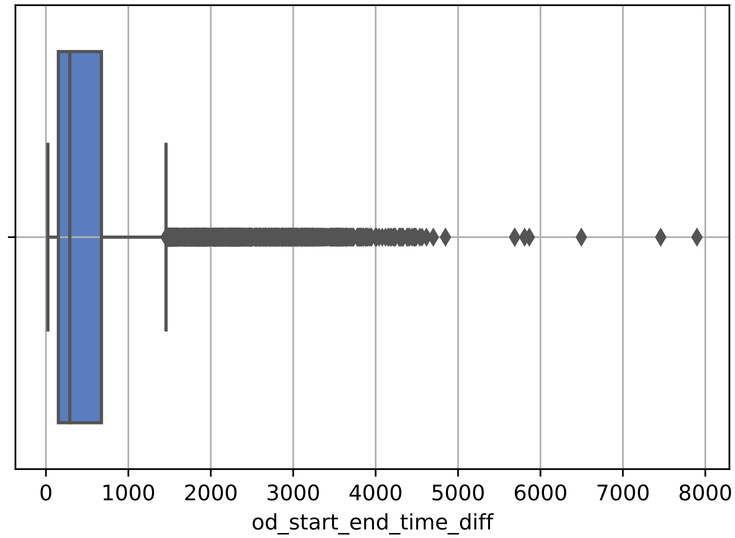

0
250
500
750
1000
1250
1500
1750
2000
osrm_time


0
500
1000
1500
2000
2500
osrm_distance


0
1000
2000
3000
4000
5000
6000
agg_segment_actual_time


0
500
1000
1500
2000
2500
agg_segment_osrm_time


0
500
1000
1500
2000
2500
3000
3500
agg_segment_osrm_distance

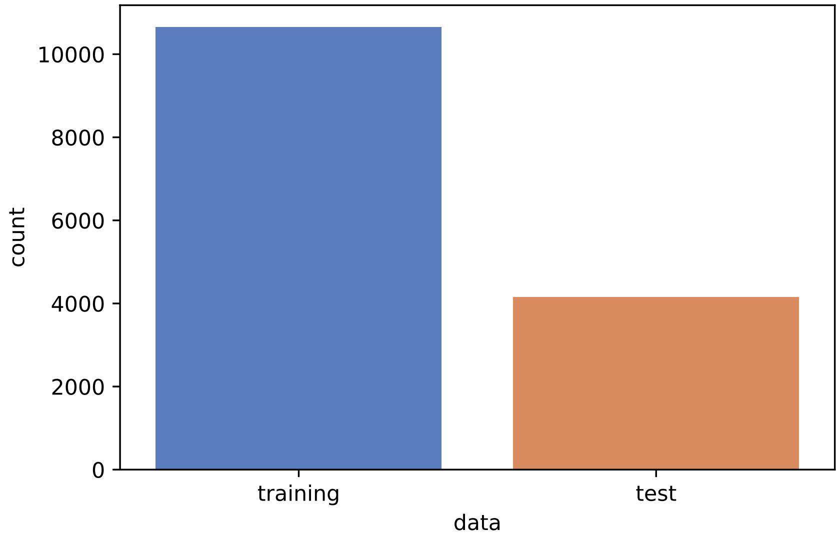


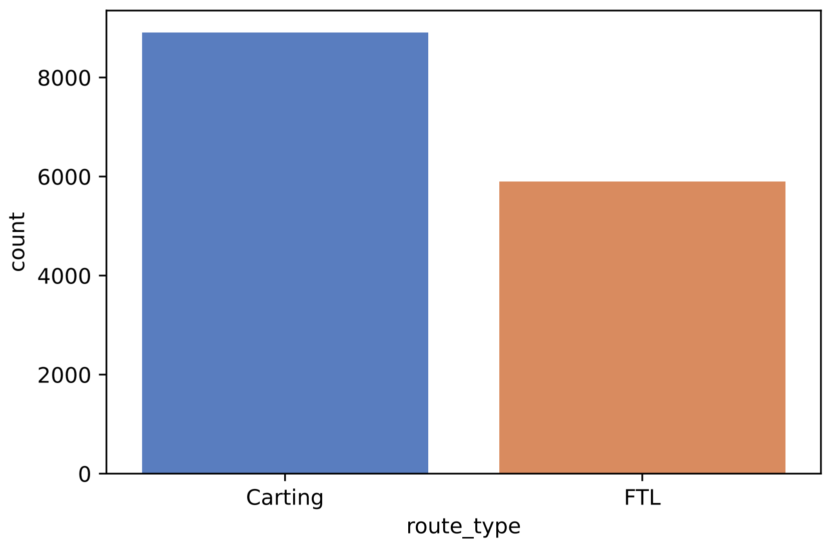
Insights:

1. As expected, numerical features have lot of outliers, also distribution is positive/right skewed
2. Upperlimit of agg_segment_osrm_time,and agg_segment_actual_time is ~500 minutes
3. For agg_segment_osrm_distance and osrm_distance - upper limit is ~500 kms
4. osrm_time & actual_time distributions seem way different. actual_time is more spread out giving an impression that estimated delivery times are way off

In [76]:
```
def print_count_plots(df,cols_list,limit=15):

    """
    Given the dataframe, categorical columns list, returns the count plots for all
    Caveat: If there are more than 100 unique categories for a column, it is skipped

    """
    for col in cat_cols:
        if df[col].nunique() <= limit:
            sns.countplot(data=df,x=col,order=df[col].value_counts().index)
            plt.show()

    return
```

In [77]:
```
# To generate count plots for all categorical variables
print_count_plots(df,cat_cols)
```



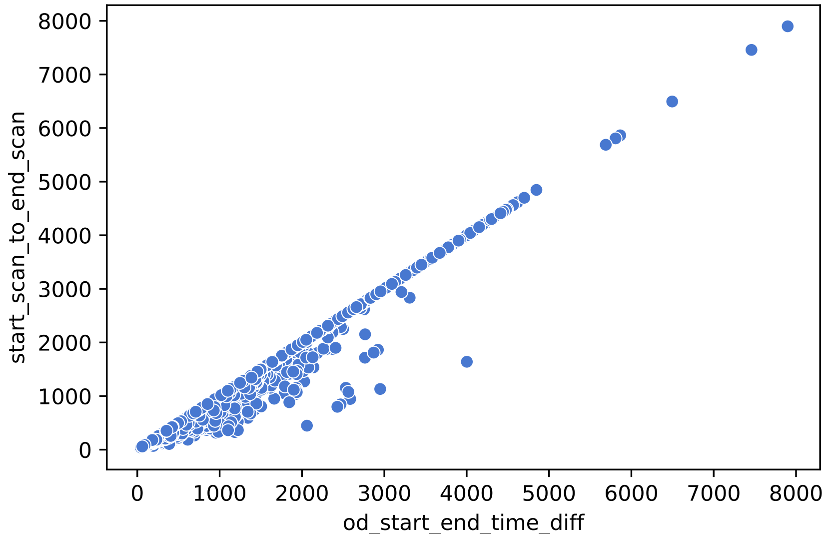


Insights:

1. There is around ~3 times more training data than test data
2. Number of FTL & Carting route_types is almost same.

Bivariate Analysis:

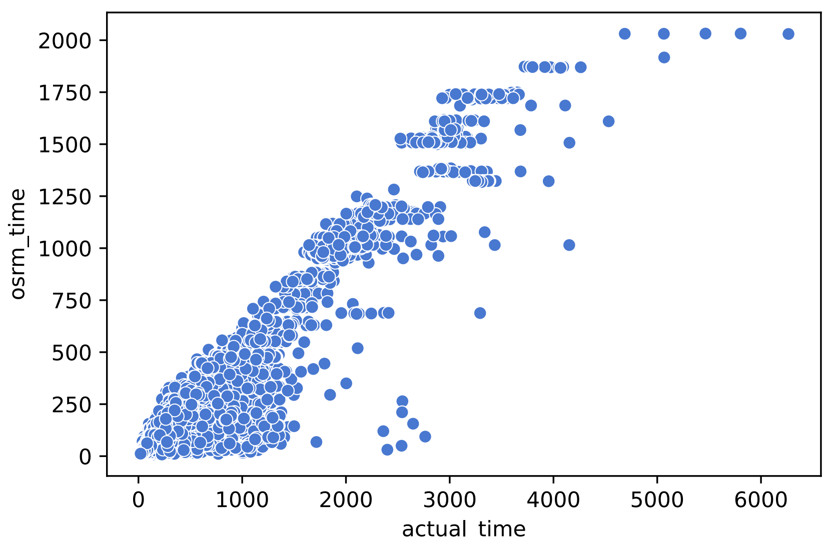
In [78]:
```
# scatterplot to show the correlation of below features
sns.scatterplot(x=df["od_start_end_time_diff"], y = df["start_scan_to_end_scan"])
plt.show()
```



Insights:

1. It looks positively correlated. However, there are deliveries especially under 3000 min where difference between start_scan_to_end_scan & od_start_end_time_diff is more
2. In the next section, lets check if the difference is significant on an average for the population
3. Are the differences due to Carting ? Let's verify that in multivriate visual analysis

In [79]:
```
# scatterplot to show the correlation of below features
sns.scatterplot(x=df["actual_time"],y=df["osrm_time"])
plt.show()
```

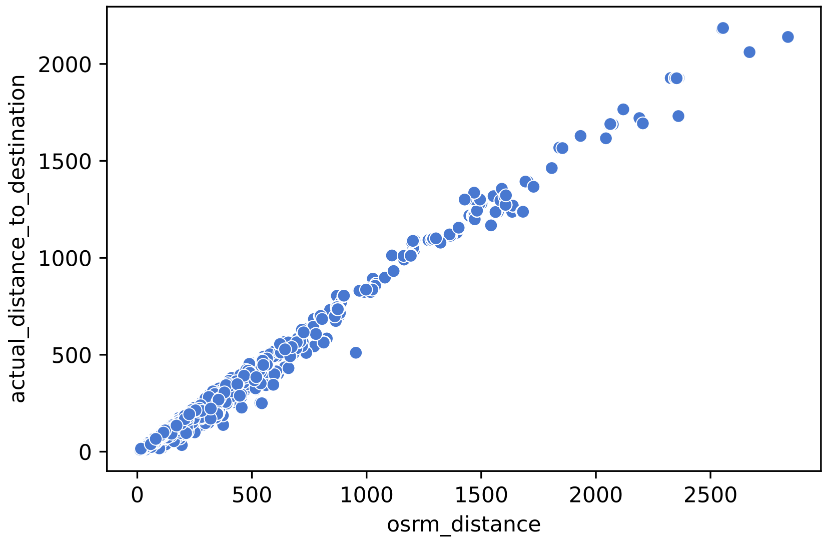


Insights:

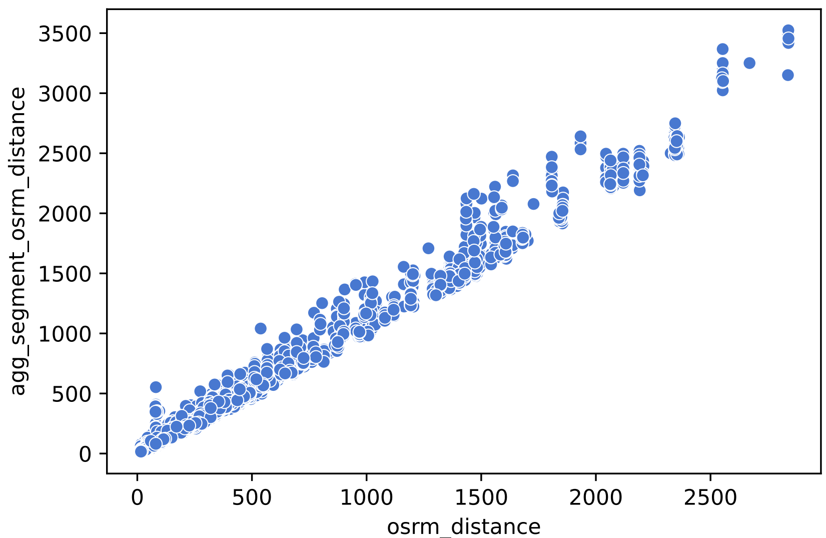
3. On the other hand, osrm_distance seem to be higher than the actual_distance_to_destination. Let's explore these two points later if it is significant

1. It's evident that actual_time and osrm_time are positively correlated.
2. actual_time seem to be way more than osrm_time. Either there are delays in delivery or osrm_time is incorrect.
3. On the other hand, osrm_distance seem to be higher than the actual_distance_to_destination. Let's explore these two points later if it is significant

In [80]:
```
# scatterplot to show the correlation of below features
sns.scatterplot(x=df["osrm_distance"],y=df["actual_distance_to_destination"])
plt.show()
```



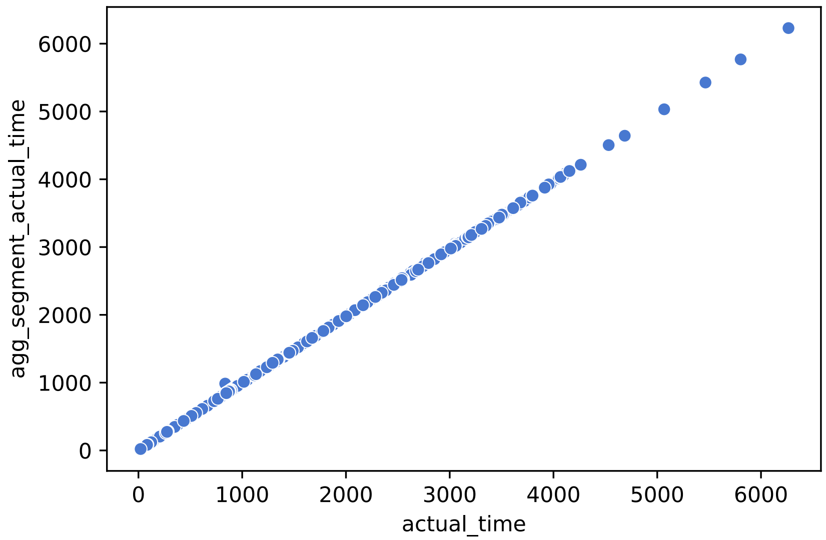
In [81]:
```
# scatterplot to show the correlation of below features
sns.scatterplot(x=df["osrm_distance"],y=df["agg_segment_osrm_distance"])
plt.show()
```



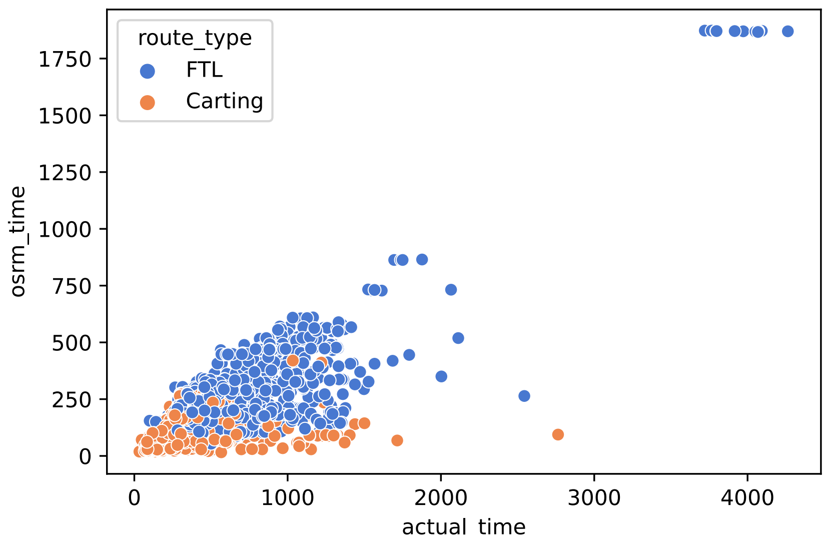
Insights:

1. osrm_distance and agg_segment_osrm_distance seem to have positive correlation
2. Both values seem to be almost similar for most of the trips. Same can be said about actual_time and agg_segment_actual_time as shown below

In [82]:
```
# scatterplot to show the correlation of below features
sns.scatterplot(x=df["actual_time"],y=df["agg_segment_actual_time"])
plt.show()
```



In [84]:
```
# Deliveries with source and destination center being the same
sns.scatterplot(data = df[df["source_center"] == df["destination_center"]], x = "actual_time", y = "osrm_time", hue="route_type")
plt.show()
```

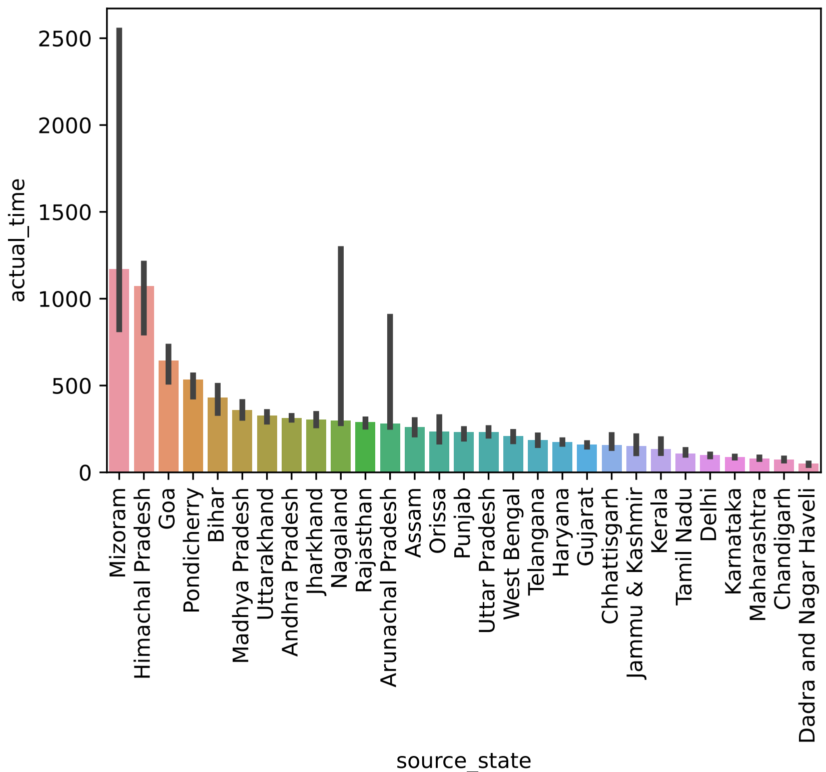


Insights:

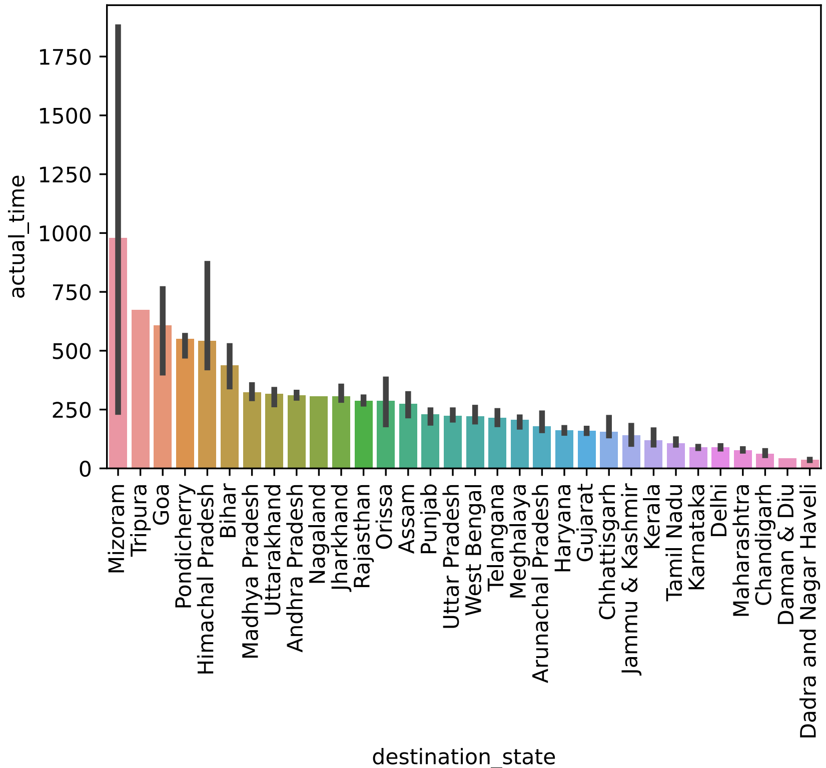
1. Actual time is way more than osrm_time for deliveries returned to source. FTL route_type trips took longer than Carting

Multivariate analysis:

In [85]:
```
# To observe where the median duration time lies for deliveries started at a source state
sns.barplot(data=df, x="source_state", y = "actual_time", estimator= np.median,
            order= df.groupby(by="source_state")["actual_time"].median().sort_values(ascending=False).index)
plt.xticks(rotation=90)
plt.show()
```



In [86]:
```
# To observe where the median duration time lies for deliveries reaching a destination state
sns.barplot(data=df, x="destination_state", y = "actual_time", estimator= np.median,
            order= df.groupby(by="destination_state")["actual_time"].median().sort_values(ascending=False).index)
plt.xticks(rotation=90)
plt.show()
```

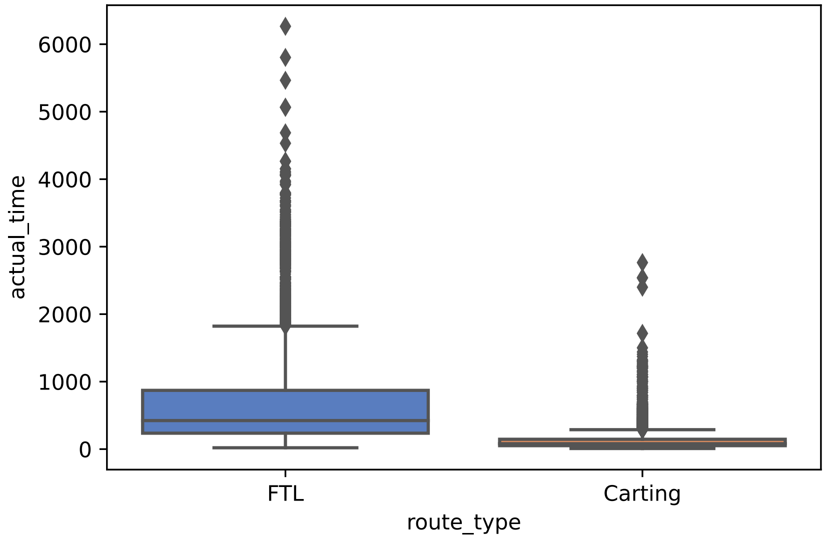


Insights:

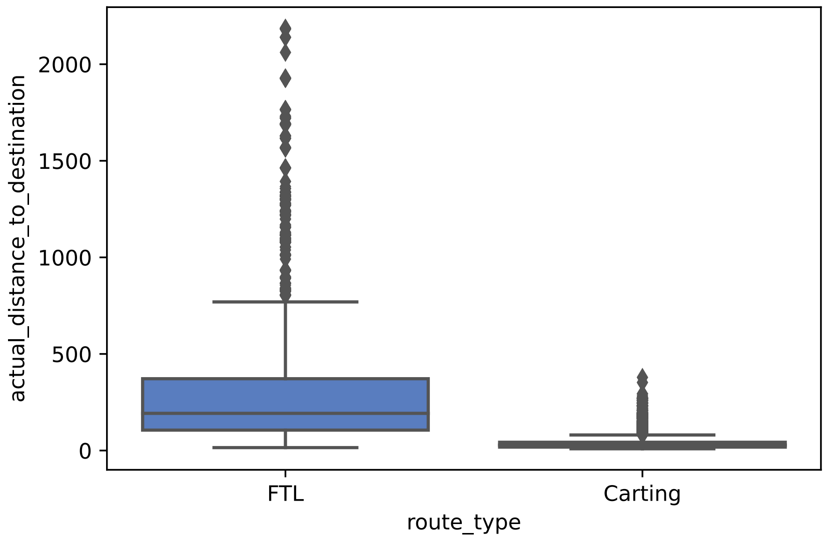
1. From the above two graphs, it is apparent that, median actual time for deliveries starting a source state or delivered to a destination state does not wary much with exceptions being Mizoram, Arunachal Pradesh, Nagaland
2. We do not have much data on Mizoram. Most of the deliveries are returned to the source_center

In [87]:

```
# To check how times differ between different route types
sns.boxplot(x=df["route_type"], y=df["actual_time"])
plt.show()
```



In [88]:
```
# To check how distances differ between different route types
sns.boxplot(x=df["route_type"], y=df["actual_distance_to_destination"])
plt.show()
```

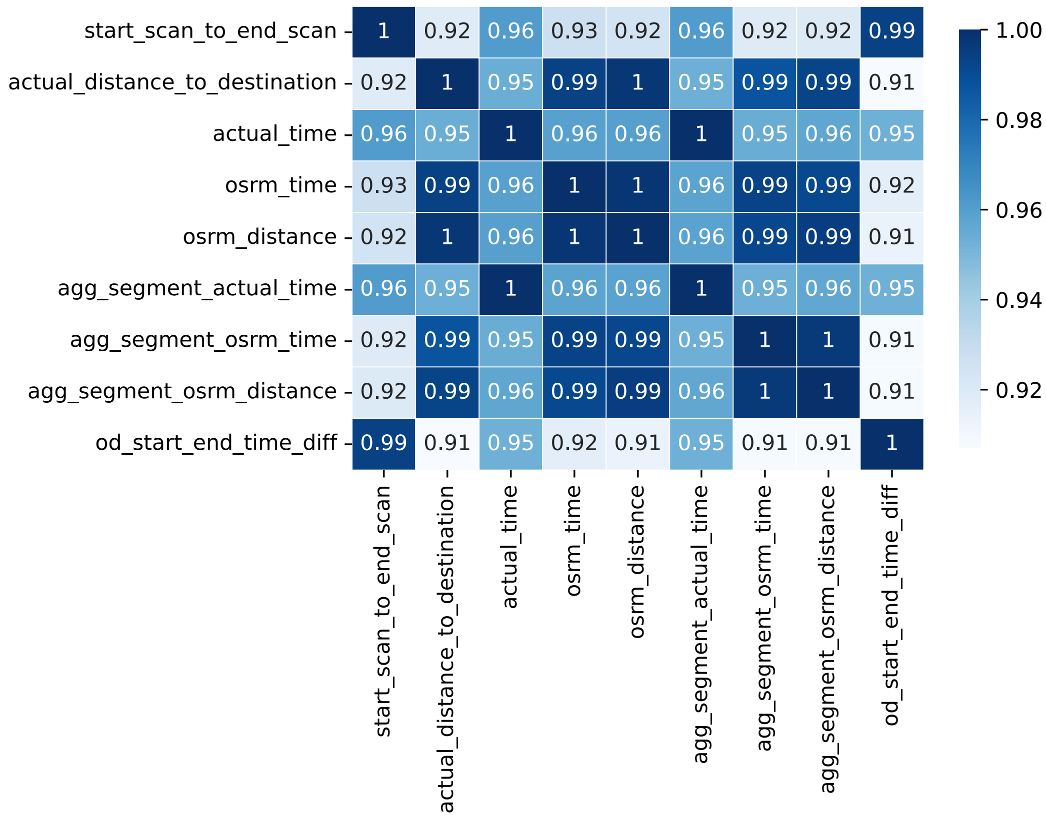


Insights:

1. It is evident that carting route type is used for distances upto 500 km or less

In [89]:
```
#Heatmap of correlation between different features:
sns.heatmap(data= df[num_cols].corr(method="pearson",numeric_only=True)
            ,cbar_kws={"shrink": .9},linewidths=0.4,cmap="Blues",annot=True,
           )
```

```
plt.yticks(rotation = 0)
```



Insights:

1. As shown in the heatmap above, majority of the features are highly correlated. We can check further to see if same can be inferred about the population via hypothesis tests
2. Based on hypothesis test result, we can infer about correlation of features in population as well
3. Also lets compare the means of different features using hypothesis tests

In-depth Analysis

In [90]:
```
def check_normality(data, alpha = 0.05):

    # Null Hypothesis, H0: Given data comes from normal distribution
    # Alternate Hypothesis, Ha: Given data does not come normal distribution

    from scipy.stats import normaltest

    teststatistic, pvalue = normaltest(data)

    print("Null Hypothesis, H0: Given data comes from normal distribution")
    print("Alternate Hypothesis, Ha: Given data does not come from normal distribution")


    if pvalue < alpha:
        print("Reject H0. Therefore, Given data does not come normal distribution")
```

```python
    else:
        print("Unable to reject H0. Therefore, Given data comes from normal distribution")

    print()
    print("----------------------------XXX---------------------------")
    print()

    print("Hypothesis test performed: ", normaltest.__name__)
    print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")
```

In [91]:
```python
def check_equal_variance(df, column1, column2, normality = True, alpha=0.05):

    # Null Hypothesis, H0:  column1 & column2 have equal variances
    # Alternate Hypothesis, Ha: column1 & column2 do not have equal variances

    from scipy.stats import levene

    if normality:
        center = "median"
    else:
        center = "trimmed"

    teststatistic, pvalue = levene(df[column1],df[column2],center = center)


    print("Null Hypothesis, H0:  Sample1 & Sample2 have equal variances")
    print("Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances")


    print("Result: ", end=" ")
    if pvalue < alpha:
        print("Reject H0. Do not have equal variances")
    else:
        print("Unable to reject H0. Given data have equal variance")

    print()
    print("----------------------------XXX---------------------------")
    print()

    print("Hypothesis test performed: ", levene.__name__)
    print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")
```

In [93]:
```python
def compare_two_means(sample1, sample2, normality = True, equal_var = True, alpha = 0.05, alternative = "two_sided"):

    '''
    Conducts ttest_ind or mannwhitneyu test based on normality and variance of the samples
    By providing alternative value, either one sided or two sided test can be conducted

    '''


    if alternative not in ["two-sided","less","greater"]:
        print("selected alternative is incorrect")
        return

    from scipy.stats import ttest_ind, mannwhitneyu
```

```
    if normality and equal_var:
        func = ttest_ind
    else:
        func = mannwhitneyu


    teststatistic, pvalue = func(sample1, sample2, alternative = alternative)

    if alternative == "greater":

        print("Null Hypothesis,H0: sample1 mean/median (mu1) <= sample2 mean/median (mu2)")
        print("Alternate Hypothesis,Ha: mu1 > mu2")

        print("Result: ", end=" ")
        if pvalue < alpha:
            print("Reject H0. mu1 > mu2")
        else:
            print("Unable to reject H0, mu1 <= mu2")


    elif alternative == "less":

        print("Null Hypothesis,H0: sample1 mean/median (mu1) >= sample2 mean/median (mu2)")
        print("Alternate Hypothesis,Ha: mu1 < mu2")

        print("Result: ", end=" ")
        if pvalue < alpha:
            print("Reject H0. mu1 < mu2")
        else:
            print("Unable to reject H0, mu1 >= mu2")

    else:

        print("Null Hypothesis,H0: sample1 mean/median (mu1) = sample2 mean/median (mu2)")
        print("Alternate Hypothesis,Ha: mu1 != mu2")

        print("Result: ", end=" ")
        if pvalue < alpha:
            print("Reject H0. mu1 != mu2")
        else:
            print("Unable to reject H0, mu1 = mu2")

    print()
    print("----------------------------XXX---------------------------")
    print()

    print("Hypothesis test performed: ", func.__name__)
    print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")
```

Hypothesis Testing 1

In [94]:

```
check_normality(df["od_start_end_time_diff"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX----------------------------
```

```
Hypothesis test performed:  normaltest
TestStatistic:8775.0006, Pvalue:0.0
```

In [95]:
```
check_normality(df["start_scan_to_end_scan"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX---------------------------

Hypothesis test performed:  normaltest
TestStatistic:9149.1143, Pvalue:0.0
```

In [96]:
```
check_equal_variance(df,"od_start_end_time_diff","start_scan_to_end_scan")
```

```
Null Hypothesis, H0:  Sample1 & Sample2 have equal variances
Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances
Result:  Reject H0. Do not have equal variances

----------------------------XXX---------------------------

Hypothesis test performed:  levene
TestStatistic:4.0108, Pvalue:0.0452
```

In [97]:
```
#Random Variable: Time in minutes
compare_two_means(df["od_start_end_time_diff"],df["start_scan_to_end_scan"],normality=False,equal_var=False,alternative="greater")
```

```
Null Hypothesis,H0: sample1 mean/median (mu1) <= sample2 mean/median (mu2)
Alternate Hypothesis,Ha: mu1 > mu2
Result:  Reject H0. mu1 > mu2

----------------------------XXX---------------------------

Hypothesis test performed:  mannwhitneyu
TestStatistic:111208635.0, Pvalue:0.0108
```

In [98]:
```
# Null Hypothesis, H0: od_start_end_time_diff and start_scan_to_end_scan are not correlated
# Alternate Hypothesis, Ha: od_start_end_time_diff and start_scan_to_end_scan are correlated


print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")

teststatistic, pvalue = pearsonr(x=df["od_start_end_time_diff"], y = df["start_scan_to_end_scan"])

print()
print("----------------------------XXX---------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX---------------------------
```

```
Hypothesis test performed:  pearsonr
TestStatistic:0.9936, Pvalue:0.0
Reject H0. Two features are correlated
```

Insights:

1. We can infer that od_start_end_time_diff and start_scan_to_end_scan are correlated for the population data as well
2. average od_start_end_time_diff seem to be significantly higher than start_scan_to_end_scan

Hypothesis Testing 2

In [99]:
```
check_normality(df["actual_time"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX---------------------------

Hypothesis test performed:  normaltest
TestStatistic:10489.5932, Pvalue:0.0
```

In [100...
```
check_normality(df["osrm_time"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX---------------------------

Hypothesis test performed:  normaltest
TestStatistic:10583.3646, Pvalue:0.0
```

In [101...
```
check_equal_variance(df,"actual_time","osrm_time", normality=False)
```

```
Null Hypothesis, H0:  Sample1 & Sample2 have equal variances
Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances
Result:  Reject H0. Do not have equal variances

----------------------------XXX---------------------------

Hypothesis test performed:  levene
TestStatistic:4030.0601, Pvalue:0.0
```

In [102...
```
#Random Variable: Time in minutes
compare_two_means(df["actual_time"],df["osrm_time"],normality=False,equal_var=False,alternative="greater")
```

```
Null Hypothesis,H0: sample1 mean/median (mu1) <= sample2 mean/median (mu2)
Alternate Hypothesis,Ha: mu1 > mu2
Result:  Reject H0. mu1 > mu2

----------------------------XXX---------------------------

Hypothesis test performed:  mannwhitneyu
TestStatistic:152381350.5, Pvalue:0.0
```

In [103...
```
print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")

teststatistic, pvalue = pearsonr(x=df["actual_time"], y = df["osrm_time"])
```

```python
print()
print("----------------------------XXX----------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX----------------------------

Hypothesis test performed:  pearsonr
TestStatistic:0.9588, Pvalue:0.0
Reject H0. Two features are correlated
```

Insights:

1. We can infer that mean actual time taken for deliveries is more than mean osrm_time
2. Both actual_time and osrm_time are correlated

Hypothesis Testing 3

In [104…

```python
check_normality(df["osrm_distance"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX----------------------------

Hypothesis test performed:  normaltest
TestStatistic:10845.0997, Pvalue:0.0
```

In [105…

```python
check_normality(df["agg_segment_osrm_distance"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX----------------------------

Hypothesis test performed:  normaltest
TestStatistic:11317.3545, Pvalue:0.0
```

In [106…

```python
check_equal_variance(df,"osrm_distance","agg_segment_osrm_distance",normality=False)
```

```
Null Hypothesis, H0:  Sample1 & Sample2 have equal variances
Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances
Result:  Reject H0. Do not have equal variances

----------------------------XXX----------------------------

Hypothesis test performed:  levene
TestStatistic:27.0822, Pvalue:0.0
```

In [107…

```
#Random Variable: distance
compare_two_means(df["osrm_distance"],df["agg_segment_osrm_distance"],normality=False,equal_var=False,alternative="less")
```

```
Null Hypothesis,H0: sample1 mean/median (mu1) >= sample2 mean/median (mu2)
Alternate Hypothesis,Ha: mu1 < mu2
Result:  Reject H0. mu1 < mu2

----------------------------XXX----------------------------

Hypothesis test performed:  mannwhitneyu
TestStatistic:105920611.5, Pvalue:0.0
```

In [108…

```
print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")

teststatistic, pvalue = pearsonr(x=df["osrm_distance"], y = df["agg_segment_osrm_distance"])

print()
print("----------------------------XXX----------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX----------------------------

Hypothesis test performed:  pearsonr
TestStatistic:0.9947, Pvalue:0.0
Reject H0. Two features are correlated
```

Insights:

1. Mean osrm_distance is less than mean agg_segment_osrm_distance for the population
2. Both features are correlated

Hypothesis Testing 4

In [109…

```
check_normality(df["agg_segment_actual_time"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution

----------------------------XXX----------------------------

Hypothesis test performed:  normaltest
TestStatistic:10482.874, Pvalue:0.0
```

In [110…

```
check_equal_variance(df,"actual_time","agg_segment_actual_time",normality=False)
```

```
Null Hypothesis, H0:  Sample1 & Sample2 have equal variances
Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances
Result:  Unable to reject H0. Given data have equal variance

----------------------------XXX----------------------------

Hypothesis test performed:  levene
TestStatistic:0.3915, Pvalue:0.5315
```

In [111…

```
#Random Variable: Time in minutes
compare_two_means(df["actual_time"],df["agg_segment_actual_time"],normality=False,equal_var=True,alternative="two-sided")
```

```
Null Hypothesis,H0: sample1 mean/median (mu1) = sample2 mean/median (mu2)
Alternate Hypothesis,Ha: mu1 != mu2
Result:  Unable to reject H0, mu1 = mu2

----------------------------XXX----------------------------

Hypothesis test performed:  mannwhitneyu
TestStatistic:110117032.5, Pvalue:0.4167
```

In [112…

```
print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")

teststatistic, pvalue = pearsonr(x=df["actual_time"], y = df["agg_segment_actual_time"])

print()
print("----------------------------XXX----------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX----------------------------

Hypothesis test performed:  pearsonr
TestStatistic:1.0, Pvalue:0.0
Reject H0. Two features are correlated
```

Insights:

1. Mean actual_time is equal to mean agg_segment_actual_time as expected
2. Both are 100% correlated

Hypothesis Testing 5

In [113…

```
check_normality(df["agg_segment_osrm_time"])
```

```
Null Hypothesis, H0: Given data comes from normal distribution
Alternate Hypothesis, Ha: Given data does not come from normal distribution
Reject H0. Therefore, Given data does not come normal distribution
```

```
----------------------------XXX----------------------------

Hypothesis test performed:  normaltest
TestStatistic:11018.6735, Pvalue:0.0
```

In [114…

```python
check_equal_variance(df,"agg_segment_osrm_time","osrm_time",normality=False)
```

```
Null Hypothesis, H0:  Sample1 & Sample2 have equal variances
Alternate Hypothesis, Ha: Sample1 & Sample2 do not have equal variances
Result:  Reject H0. Do not have equal variances

----------------------------XXX----------------------------

Hypothesis test performed:  levene
TestStatistic:80.0469, Pvalue:0.0
```

In [115…

```python
#Random Variable: Time in minutes
compare_two_means(df["agg_segment_osrm_time"],df["osrm_time"],normality=False,equal_var=False,alternative="greater")
```

```
Null Hypothesis,H0: sample1 mean/median (mu1) <= sample2 mean/median (mu2)
Alternate Hypothesis,Ha: mu1 > mu2
Result:  Reject H0. mu1 > mu2

----------------------------XXX----------------------------

Hypothesis test performed:  mannwhitneyu
TestStatistic:113622396.5, Pvalue:0.0
```

In [116…

```python
print("Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated")
print("Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated")

teststatistic, pvalue = pearsonr(x=df["agg_segment_osrm_time"], y = df["osrm_time"])

print()
print("----------------------------XXX----------------------------")
print()

print("Hypothesis test performed: ", pearsonr.__name__)
print(f"TestStatistic:{np.round(teststatistic,4)}, Pvalue:{np.round(pvalue,4)}")


if pvalue < 0.05:
    print("Reject H0. Two features are correlated")
else:
    print("Unable to reject H0")
```

```
Null Hypothesis, H0: Sample 1 and Sample 2 are not correlated
Alternate Hypothesis, Ha: Sample 1 and Sample 2 are correlated

----------------------------XXX----------------------------

Hypothesis test performed:  pearsonr
TestStatistic:0.9933, Pvalue:0.0
Reject H0. Two features are correlated
```

Insights:

1. Both are correlated positively. However mean agg_segment_osrm_time is more than mean osrm_time
2. Based on the above tests, we can drop agg_segment_actual_time and keep just the actual_time as both are totally the same

In [117…

```
# Dropping the agg_segment_actual_time column
df.drop(columns=["agg_segment_actual_time"], inplace=True)

# removing it from num_cols list
num_cols.remove("agg_segment_actual_time")
```

Scaling:

In [118…

```
# All numerical columns are standardized
# As there are outliers, considered to perform standardization using Standard Scaler instead of MinMaxScaler
df2 = df.copy()
for column in num_cols:
    df2[column] = StandardScaler().fit_transform(df2[[column]])
```

In [119…

```
# Resultant dataframe after standard scaling
df2[num_cols].describe()
```

Out[119…

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	agg_segment_osrm_time	agg_segment_osrm_distance	od_start_end_time_diff
count	1.480000e+04	1.480000e+04	1.480000e+04	1.480000e+04	1.480000e+04	1.480000e+04	1.480000e+04	1.480000e+04
mean	1.440289e-17	7.921591e-18	-2.688540e-17	1.056212e-17	-3.504704e-17	1.920386e-18	-1.824366e-17	-1.094620e-16
std	1.000034e+00	1.000034e+00	1.000034e+00	1.000034e+00	1.000034e+00	1.000034e+00	1.000034e+00	1.000034e+00
min	-7.711622e-01	-5.092113e-01	-6.201882e-01	-5.726888e-01	-5.272914e-01	-5.562771e-01	-5.140449e-01	-7.839216e-01
25%	-5.798734e-01	-4.640970e-01	-5.169075e-01	-4.879709e-01	-4.687275e-01	-4.800122e-01	-4.575592e-01	-5.930000e-01
50%	-3.809938e-01	-3.800571e-01	-3.708899e-01	-3.737859e-01	-3.747748e-01	-3.687925e-01	-3.676043e-01	-3.870166e-01
75%	1.625092e-01	3.999615e-04	2.264536e-02	2.494076e-02	1.122734e-02	1.253196e-02	-1.061236e-02	1.883625e-01
max	1.118439e+01	6.617773e+00	1.051989e+01	6.889854e+00	7.112275e+00	7.572290e+00	7.917633e+00	1.099373e+01

One-hot encoding:

In [120…

```
df2 = pd.concat([df2,pd.get_dummies(df2["route_type"])], axis = 1)
df2.drop(columns = ["route_type"], inplace=True)
```

In [121…

```
df2.head(5)
```

Out[121…

	data	trip_creation_time	route_schedule_uuid	trip_uuid	source_center	source_name	destination_center	destination_name	start_scan_to_end_scan	ac
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc60074b	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2.623454	
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0d68b9	trip-153671042288605164	IND572101AAA	Tumkur_Veersagr_I (Karnataka)	IND562101AAA	Chikblapur_ShntiSgr_D (Karnataka)	-0.532810	
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e5720d	trip-153671043369099517	IND562132AAA	Bangalore_Nelmngla_H (Karnataka)	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab)	5.164863	

	data	trip_creation_time	route_schedule_uuid	trip_uuid	source_center	source_name	destination_center	destination_name	start_scan_to_end_scan	ac
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f9f442	trip-153671046011330457	IND400072AAB	Mumbai Hub (Maharashtra)	IND401104AAA	Mumbai_MiraRd_IP (Maharashtra)	-0.654264	
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df0613461b0f	trip-153671052974046625	IND583101AAA	Bellary_Dc (Karnataka)	IND583101AAA	Bellary_Dc (Karnataka)	0.282444	

Final Insights:

Observed Patterns:

1. Busiest state is Maharastra with average time of 164 minutes and average distance of 60 kms
2. Busiest city is Bangalore with average time of 100 minutes and average distance of 37 kms
3. Top 5 busiest states are Karnataka, Maharastra, Tamilnadu, Haryana, and Uttar Pradesh
4. Top busiest cities are Bangalore, Mumbai, Gurgaon, Bhiwandi, Hyderabad, Delhi

Inferences:

1. Aggregated osrm time for each segment of a trip is greater than than of the osrm time of the entire trip
2. Mean osrm time of a given trip is significantly less than the actual time taken by the delivery. Above could be one of the reasons
3. Mean osrm distance calculated by the open source routing machine is higher than the actual distance to destination
4. Mean difference of Trip start and end times is more than mean time taken to deliver from source to destination
5. There are features that are highly correlated. Consider dropping them for ML training as it would affect results

Recommendations:

1. Better data collection process to prevent missing/bad data in source_name and destination_name columns.
2. Highly recommend looking into Open Source routing engine as it is under-estimating delivery times & over-estimating shortest distances between two points. Better estimation will help in improving the quality of the service provided to customers
3. Adding more trucks (consider electric) to busiest states/cities would help reduce the delivery times even further and provide a competing edge against other companies
4. Look into why actual delivery times are way higher than predicted times. It might help in reducing the delivery timelines and help in competing with the other logistics companies
5. Looking into the reasons why there are deliveries returning to source centers could help improving the efficiency of the logistics
6. Collecting datapoints on the delivery trucks and drivers would help in further analysis

In [ ]: