```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

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

```
df
```

	data	trip_creation_time	route_schedule_uuid	route_type	tri
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	15374606684
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	15374606684
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	15374606684
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	15374606684
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	15374606684

144867 rows × 24 columns

Problem statement:

Clean, sanitize and manipulate the raw data of Delhivery to extract meaningful insights and carry out feature engineering.

```
df.shape
```

```
(144867, 24)
```

```
# Statistical summary
df.describe()
```

	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

```
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
```

```
# We will convert the data type of few columns
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"],format = "%Y-%m-%d %H:%M:%S.%f")
df["od_start_time"] = pd.to_datetime(df["od_start_time"],format = "%Y-%m-%d %H:%M:%S.%f")
df["od_end_time"] = pd.to_datetime(df["od_end_time"],format = '%Y-%m-%d %H:%M:%S.%f')
df["cutoff_timestamp"] = pd.to_datetime(df["cutoff_timestamp"],format = "mixed")
df[["data","route_type"]] = df[["data","route_type"]].astype("category")
```

```
# We will rename few columns for easier readability
df.rename(columns = {"segment_factor":"segment_act_time/osmr_time",
                    "factor": "actual_time/osmr_time",
                    "cutoff_timestamp": "actual_delivery_time"}, inplace =True)
```

```
# We will drop is_cutoff since it is of no use, we will also drop "cutoff factor" since this number
# is already being reflected in the column "actual_distance_to_destination"
df.drop(columns = ["is_cutoff","cutoff_factor"], inplace = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 22 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_delivery_time            144867 non-null  datetime64[ns]
 13  actual_distance_to_destination  144867 non-null  float64
 14  actual_time                     144867 non-null  float64
 15  osrm_time                       144867 non-null  float64
 16  osrm_distance                   144867 non-null  float64
 17  actual_time/osmr_time           144867 non-null  float64
```

```
 18  segment_actual_time             144867 non-null  float64
 19  segment_osrm_time               144867 non-null  float64
 20  segment_osrm_distance           144867 non-null  float64
 21  segment_act_time/osmr_time      144867 non-null  float64
dtypes: category(2), datetime64[ns](4), float64(10), object(6)
memory usage: 22.4+ MB
```

```
# Checking for missing values
df.isnull().sum()
```

```
data                                0
trip_creation_time                  0
route_schedule_uuid                 0
route_type                          0
trip_uuid                           0
source_center                       0
source_name                       293
destination_center                  0
destination_name                  261
od_start_time                       0
od_end_time                         0
start_scan_to_end_scan              0
actual_delivery_time                0
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
actual_time/osmr_time               0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
segment_act_time/osmr_time          0
dtype: int64
```

```
source = df[["source_center","source_name"]].drop_duplicates()
destination = df[["destination_center","destination_name"]].drop_duplicates()
```

```
merged = df.merge(source, on = "source_center", how = "left").merge(destination, on = "destination_center", how = "left").dr
```

```
merged.isna().sum()
```

```
data                                0
trip_creation_time                  0
route_schedule_uuid                 0
route_type                          0
trip_uuid                           0
source_center                       0
destination_center                  0
od_start_time                       0
od_end_time                         0
start_scan_to_end_scan              0
actual_delivery_time                0
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
actual_time/osmr_time               0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
segment_act_time/osmr_time          0
source_name_y                     293
destination_name_y                261
dtype: int64
```

Here we see that, even after obtaining source name and destination name from source_center and destination_center resp. we still have the same no. of null values. Therefore, all the null value belong to the same source_center and destination_center. So we will have to impute the null values with the mode if the entire data, since it is not a numerical data.

```
df["source_name"] = df["source_name"].fillna(df["source_name"].mode()[0])
df["destination_name"] = df["destination_name"].fillna(df["destination_name"].mode()[0])
```

```
df["source_name"].mode()
```

```
0    Gurgaon_Bilaspur_HB (Haryana)
Name: source_name, dtype: object
```

```
df["destination_name"].mode()
```

```
0    Gurgaon_Bilaspur_HB (Haryana)
Name: destination_name, dtype: object
```

```
df.isna().sum()
# Now there are no null values
```

```
data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                       0
destination_center                0
destination_name                  0
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
actual_delivery_time              0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
actual_time/osmr_time             0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
segment_act_time/osmr_time        0
dtype: int64
```

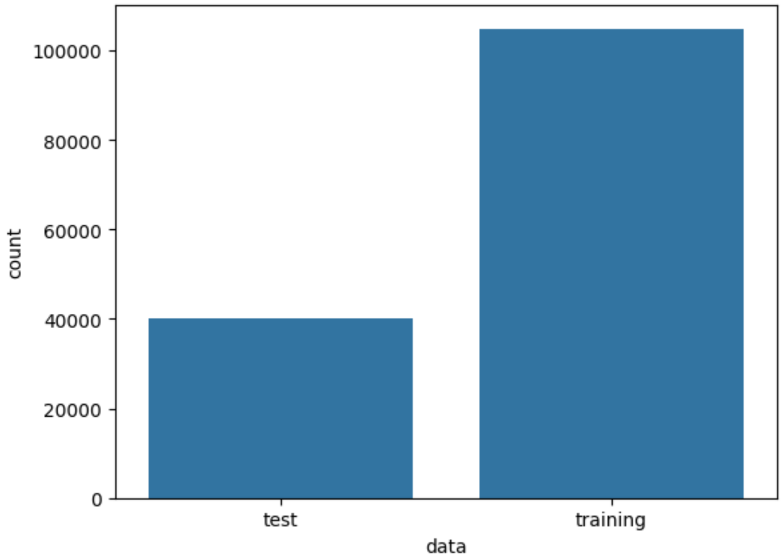
```
df["source_name"].mode()
```

```
0    Gurgaon_Bilaspur_HB (Haryana)
Name: source_name, dtype: object
```

EDA - Basic visual analysis

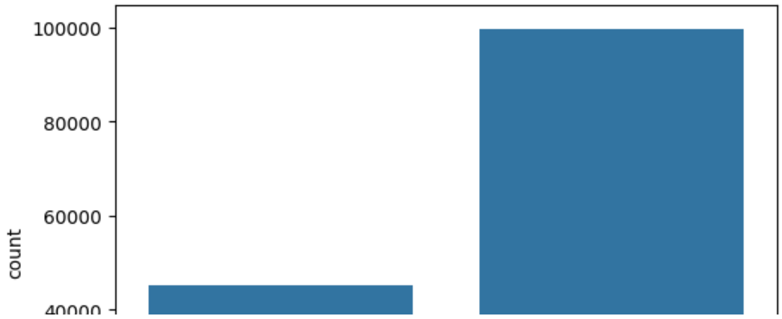
```
# Lets see how much is the distribution of training and test data
sns.countplot(data = df, x = df["data"])
```

```
<Axes: xlabel='data', ylabel='count'>
```




```
# Lets see how much is the distribution of route_type
sns.countplot(data = df, x = df["route_type"])
```

```
<Axes: xlabel='route_type', ylabel='count'>
```

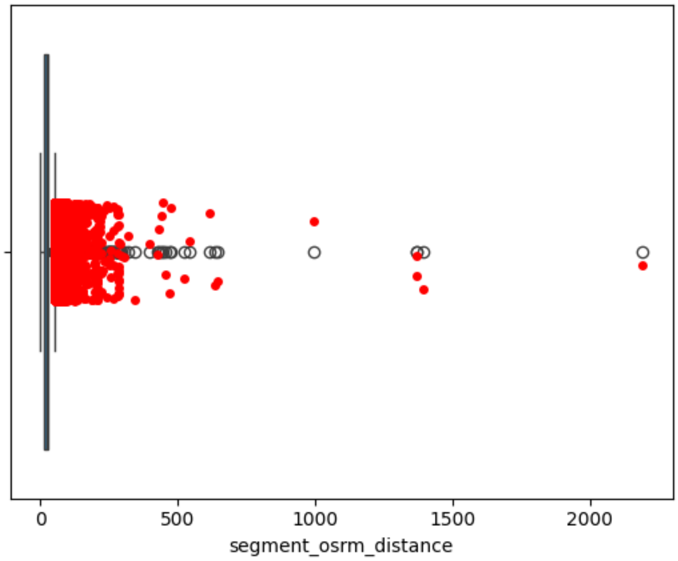



```
def outlier_func(arr):
    q1 = np.percentile(arr,25)
    q2 = np.percentile(arr,75)
    iqr = 1.5*(q2 - q1)
    low = max(q1 - iqr,0)
    high = q2 + iqr
    outliers = arr[(arr>high) | (arr<low)]
    return outliers
```

route_type

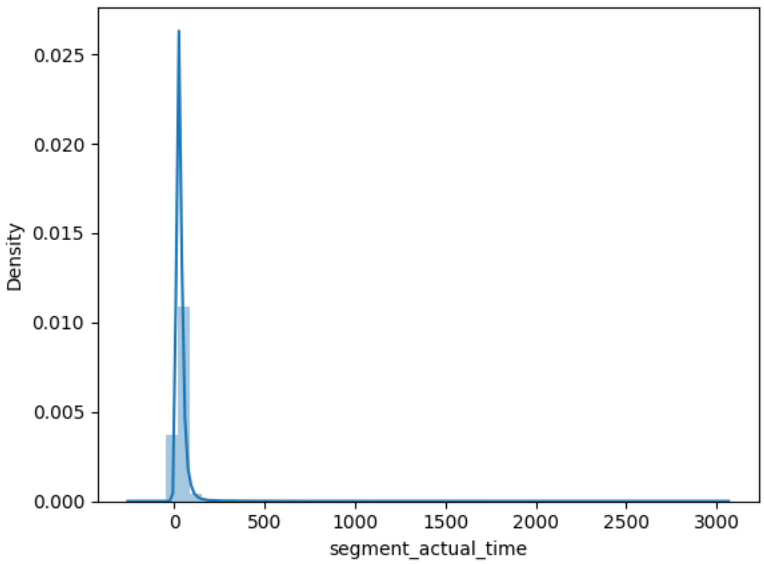
```
# Outliers in "segment_osrm_distance" column
sns.boxplot(x = df["segment_osrm_distance"])
sns.stripplot(x = outlier_func(df["segment_osrm_distance"]), color = "red")
# We find out that there are a lot of outliers above the lower whisker.
```

```
<Axes: xlabel='segment_osrm_distance'>
```




```
# Univariate analysis of ridership
sns.distplot(df["segment_actual_time"])
# We see that the actual time for each segment has a very big range, but most of the order are completed within 50h
```

```
<Axes: xlabel='segment_actual_time', ylabel='Density'>
```



```
df.describe()
# From the overall data we see that, the first trip is on 2018-09-22 and the last trip is on 2018-10-08.
# The median time taken from source to destination is 449.
# The distance ranges from 9 to 1927km.
# The median time between each segment is 29
```

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end
count	144867	144867	144867	144867.(
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.:
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.(
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.(
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.(
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.(
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.(
std	NaN	NaN	NaN	1037.(

Feature creation

```
# Let us get the city name and place name from source_name
df[["source_city", "source_place"]] = df["source_name"].str.split("_", expand = True).drop(columns = [2,3])
```

```
# Let us get the city name and place name from destination_name
df[["destination_city", "destination_place"]] = df["destination_name"].str.split("_", expand = True).drop(columns = [2,3])
```

```
# Let us get the state name which is within the brackets from source_name
df['source_name'] = df['source_name'].astype('str')
df["source_state"] = df["source_name"].apply(lambda st: st[st.find("(")+1:st.find(")")])
```

```
# Let us get the state name which is within the brackets from source name
df['destination_name'] = df['destination_name'].astype('str')
df["destination_state"] = df["destination_name"].apply(lambda st: st[st.find("(")+1:st.find(")")])
```

```
# Let us create columns for year, month and date from trip_creation_time
df["trip_creation_year"] = df["trip_creation_time"].dt.year
df["trip_creation_month"] = df["trip_creation_time"].dt.month
df["trip_creation_day"] = df["trip_creation_time"].dt.day
```

```
df['od_trip_duration'] = df["od_end_time"] - df["od_start_time"]
```

```
df['od_trip_duration']
```

```
0        0 days 01:26:12.818197
1        0 days 01:26:12.818197
2        0 days 01:26:12.818197
3        0 days 01:26:12.818197
4        0 days 01:26:12.818197
                    ...
144862   0 days 07:07:41.181838
144863   0 days 07:07:41.181838
144864   0 days 07:07:41.181838
144865   0 days 07:07:41.181838
144866   0 days 07:07:41.181838
Name: od_trip_duration, Length: 144867, dtype: timedelta64[ns]
```

```
# Dropping the unnecessary columns
df.drop(columns = ["source_center", "source_name", "destination_center", "destination_name", "od_end_time", "od_start_time"],
```

Aggregating data on the basis of tip_uuid

```
df_agg = df.groupby("trip_uuid").agg({
    "data": 'first',
    "trip_creation_time": 'first',
    "route_schedule_uuid": 'first',
    "route_type": 'first',
    "start_scan_to_end_scan": 'first',
    "source_city": 'first',
    "source_place": 'first',
    "destination_city": 'first',
    "destination_place": 'first',
    "source_state": 'first',
    "destination_state": 'first',
    "trip_creation_year": 'first',
    "trip_creation_month": 'first',
    "trip_creation_day": 'first',
    "od_trip_duration": 'first',
    "actual_distance_to_destination": 'max',
    "actual_time": 'max',
    "osrm_time": 'max',
    "osrm_distance": "max",
    "actual_time/osmr_time": 'mean',
    "segment_actual_time": 'sum',
    "segment_osrm_time": 'sum',
    "segment_osrm_distance": 'sum',
    "segment_act_time/osmr_time": 'mean'
}).reset_index()
```

```
df_agg
```

	trip_uuid	data	trip_creation_time	route_schedule_uuid	route
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
2	trip-153671043369099517	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	
3	trip-153671046011330457	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	
4	trip-153671052974046625	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	
...	...	...	...	...	
14812	trip-153861095625827784	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	
14813	trip-153861104386292051	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	
14814	trip-153861106442901555	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	
14815	trip-153861115439069069	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	
14816	trip-153861118270144424	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	

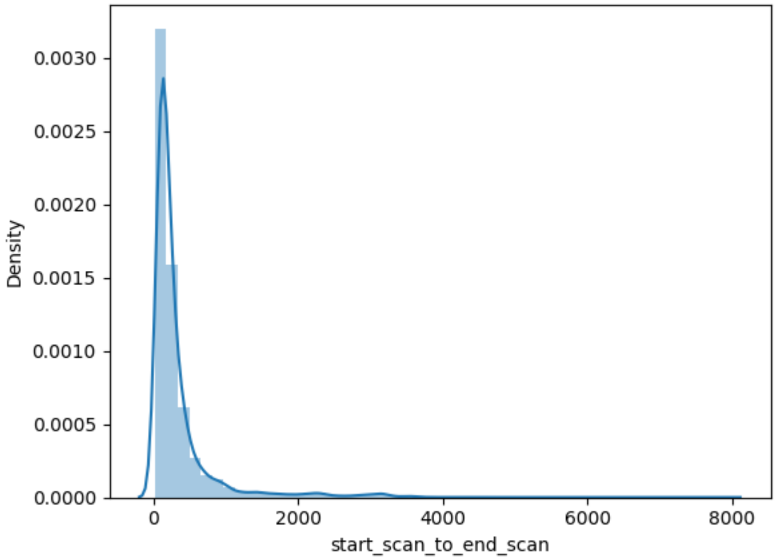
14817 rows × 25 columns

1. Comparing od_trip_duration with start_scan_to_end_scan

```
# We will convert the data type of the column "od_trip_duration" into seconds, so that it can be plotted
df_agg["od_trip_duration"] = df_agg["od_trip_duration"].astype("int64")
```

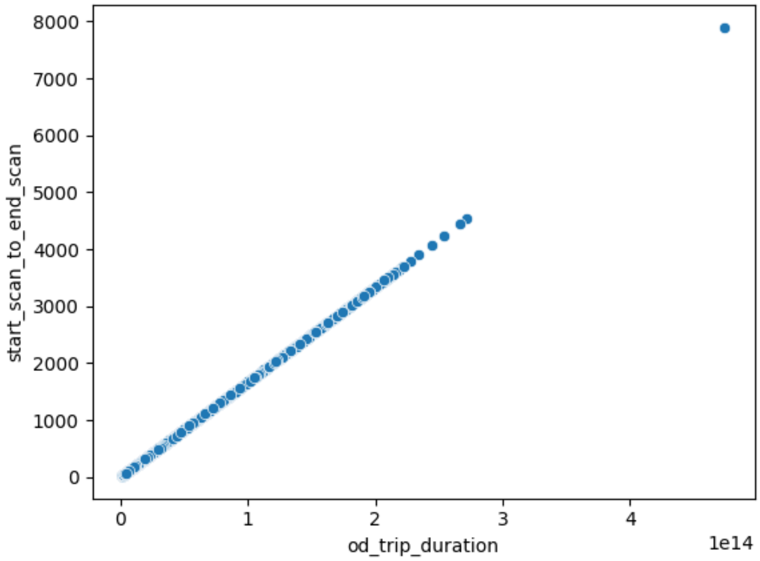
```
#Univariate analysis of df_agg["start_scan_to_end_scan"]
sns.distplot(df_agg["start_scan_to_end_scan"])
```

```
<Axes: xlabel='start_scan_to_end_scan', ylabel='Density'>
```




```
sns.scatterplot(data = df_agg, x = "od_trip_duration", y =  "start_scan_to_end_scan")
```

```
<Axes: xlabel='od_trip_duration', ylabel='start_scan_to_end_scan'>
```




```
df_agg["od_trip_duration"].corr(df_agg["start_scan_to_end_scan"])
```

```
0.9999998327770195
```

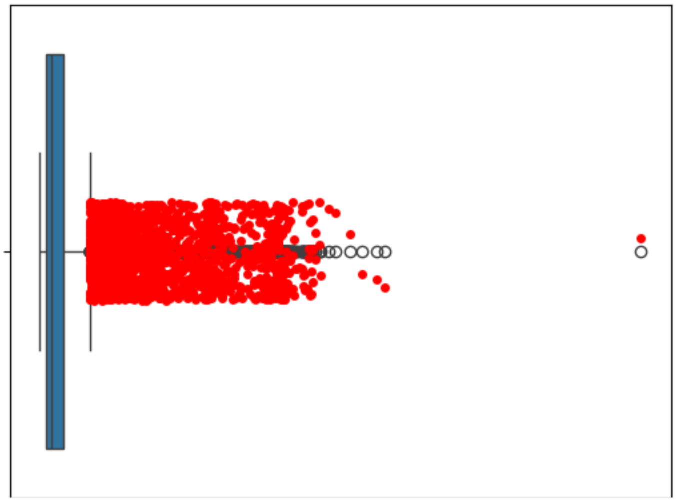
We observe that the columns od_trip_duration and start_scan_to_end_scan are directly correlated with the correlation coefficient almost equal to 1. Even from the graph we can observe that. So basically both these columns are one and the same, we can drop any one of them for training our ML model.

```
df_agg.drop(columns = "od_trip_duration", inplace = True)
```

Detecting Outliers

```
# Outliers in "segment_osrm_distance" column
sns.boxplot(x = df_agg["start_scan_to_end_scan"])
sns.stripplot(x = outlier_func(df_agg["start_scan_to_end_scan"]), color = "red")
# We find out that there are a lot of outliers above the lower whisker.
```

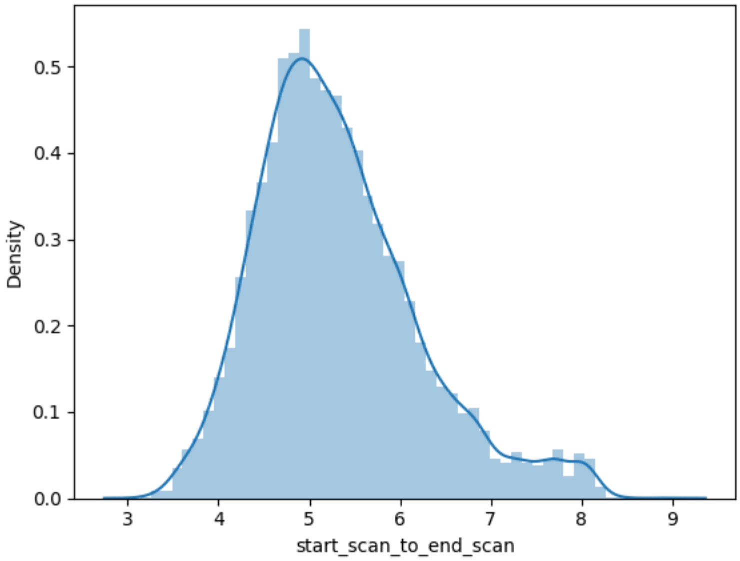
```
<Axes: xlabel='start_scan_to_end_scan'>
```



Treating outliers

```
a = np.log1p(df_agg["start_scan_to_end_scan"])
sns.distplot(a)
```

```
<Axes: xlabel='start_scan_to_end_scan', ylabel='Density'>
```




```
# Outliers in "segment_osrm_distance" column
sns.boxplot(x = a)
sns.stripplot(x = outlier_func(a), color = "red")
# We find out that there are a lot of outliers above the lower whisker.
```

```
<Axes: xlabel='start_scan_to_end_scan'>
```

Even inspite of converting into log normal scale, we still see some outliers. So let us impute the outliers with median values.

```
def imputer(a):
    median_value = a.median()
    # Calculate the interquartile range (IQR)
    Q1 =  a.quantile(0.25)
    Q3 =  a.quantile(0.75)
    IQR = Q3 - Q1

    # Define the upper and lower bounds for identifying outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = (a < lower_bound) | (a > upper_bound)

    # Impute outliers with the median value
    a[outliers] = median_value
    return a
```

```
df_agg["start_scan_to_end_scan_imputed"] = imputer(df_agg["start_scan_to_end_scan"])
```

Normalizing/standardizing using min - max scaler

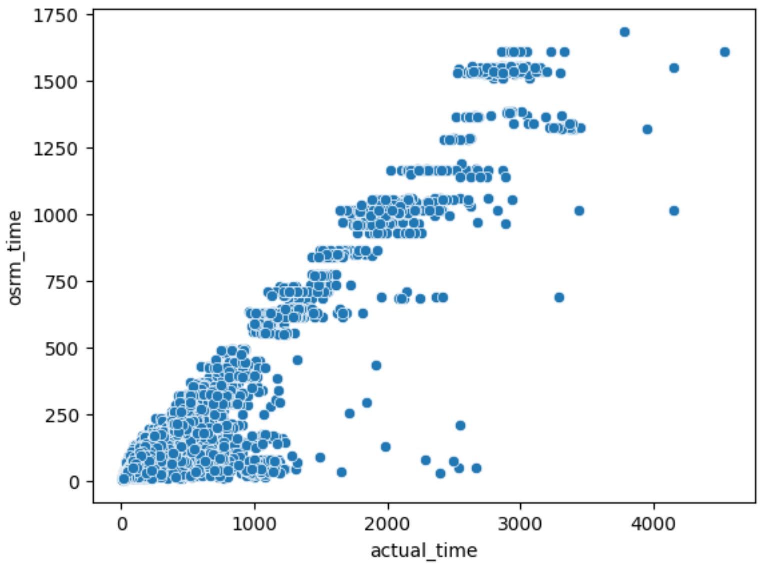
```
def minmax_scaler(a):
    return (a-a.min())/(a.max()-a.min())
```

```
df_agg["start_scan_to_end_scan_imputed"] = minmax_scaler(df_agg["start_scan_to_end_scan_imputed"])
```

2. Comparing actual_time with osrm_time

```
sns.scatterplot(data = df_agg, x = "actual_time", y =  "osrm_time")
```

```
<Axes: xlabel='actual_time', ylabel='osrm_time'>
```




```
df_agg["actual_time"].corr(df_agg["osrm_time"])
```

```
0.9538911998326526
```

We observe high correlation between actual_time and osrm_time

```
a = df_agg["actual_time"]/(df_agg["actual_time"].median())
a
```

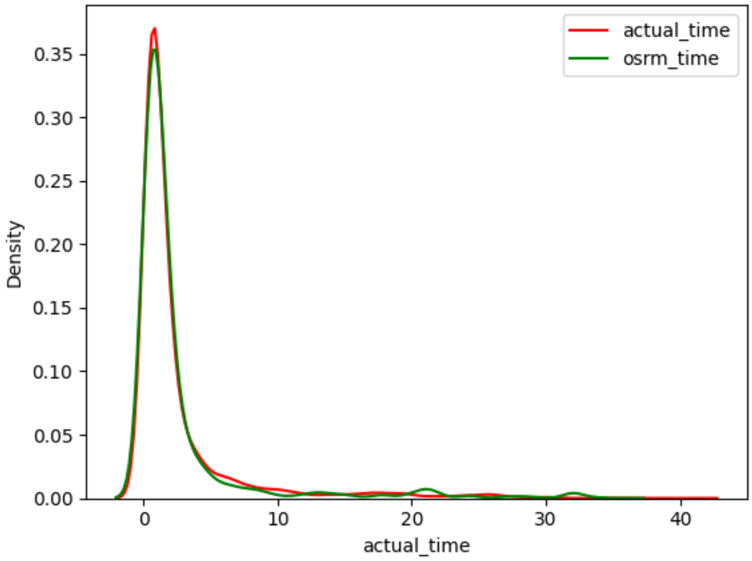
```
0         7.477477
1         0.864865
2        24.648649
3         0.531532
4         1.324324
            ...
14812     0.441441
14813     0.189189
14814     1.711712
14815     0.810811
14816     2.099099
Name: actual_time, Length: 14817, dtype: float64
```

```
b = df_agg["osrm_time"]/(df_agg["osrm_time"].median())
b
```

```
0         8.208333
1         0.875000
2        31.854167
3         0.312500
4         0.958333
            ...
14812     0.708333
14813     0.250000
14814     0.604167
14815     1.041667
14816     0.875000
Name: osrm_time, Length: 14817, dtype: float64
```

```
sns.kdeplot(a, color = "red", label = "actual_time")
sns.kdeplot(b, color = 'green', label = "osrm_time")
plt.legend()
# With KDE plot we can do side by side comparision of the two curves.
```

```
<matplotlib.legend.Legend at 0x13b287e00>
```

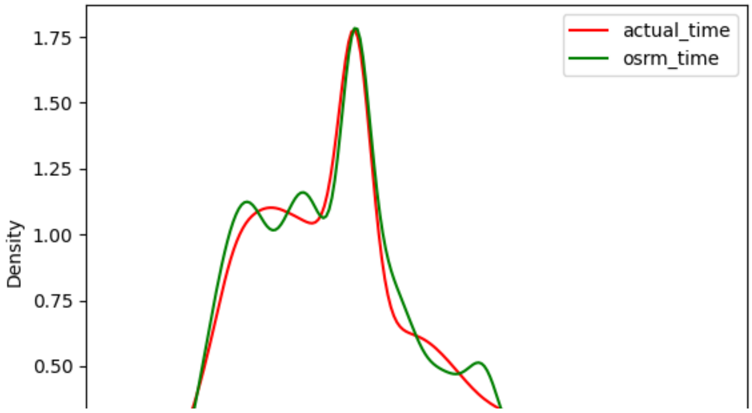


Since there are a lot of outliers, Let us impute the outliers with median values.

```
df_agg["actual_time_imputed"] = imputer(df_agg["actual_time"])
df_agg["osrm_time_imputed"] = imputer(df_agg["osrm_time"])
```

```
# Let us plot the kde plot after imputing
a0 = df_agg["actual_time_imputed"]/(df_agg["actual_time_imputed"].median())
b0 = df_agg["osrm_time_imputed"]/(df_agg["osrm_time_imputed"].median())
a0 = np.log1p(a0)
b0 = np.log1p(b0)
sns.kdeplot(a0, color = "red", label = "actual_time")
sns.kdeplot(b0, color = 'green', label = "osrm_time")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x13970a060>
```



It seems that the curve doesn't follow normal distribution, but actual_time and Osrm_time do follow each other. There is some degree of similiarity. Since the two variables are numerical data, and they don't appear to follow normal distribution we will use KS test.

Null Hypothesis: Actual time aggregated value and OSRM time aggregated value are taken from the same distribution

Alternate Hypothesis: Actual time aggregated value and OSRM time aggregated value are different

Test: KS test for independant samples.

significance level: 0.05

```
stats.kstest(a0,b0)
```

```
KstestResult(statistic=0.02929067962475529, pvalue=5.904917350307763e-06, statistic_location=1.1460788259070334,
statistic_sign=-1)
```

p<<0.05, therefore the alternate hypothesis is true.

The results of the test make it clear that the two samples come from different distributions.

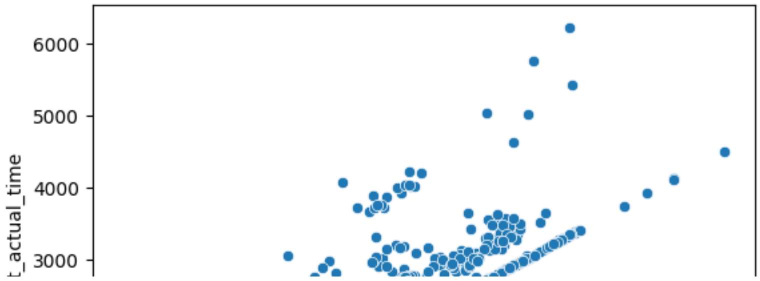
Normalizing/standardizing using min - max scaler

```
df_agg["actual_time_imputed"] = minmax_scaler(a0)
df_agg["osrm_time_imputed"] = minmax_scaler(b0)
```

3. Comparing actual_time with segment_actual_time

```
sns.scatterplot(data = df_agg, x = "actual_time", y =  "segment_actual_time")
```

```
<Axes: xlabel='actual_time', ylabel='segment_actual_time'>
```




```
df_agg["actual_time"].corr(df_agg["segment_actual_time"])
```

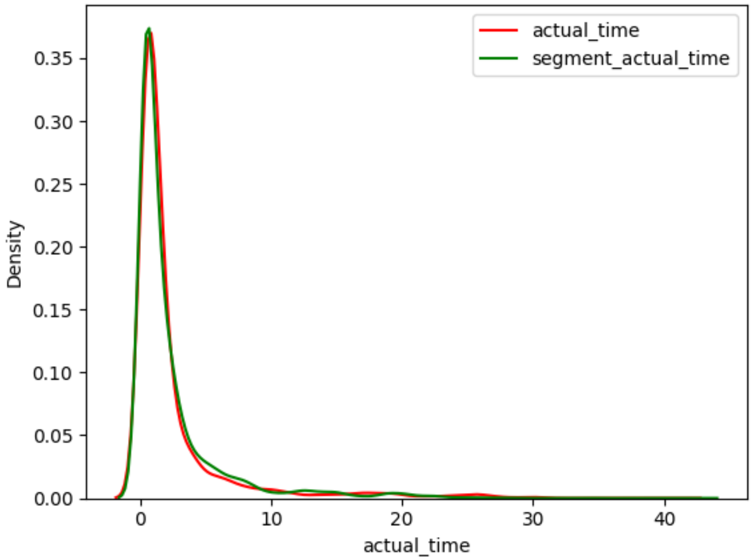
```
0.9585323357636127
```

We observe high correlation between actual_time and segment_actual_time

```
a = df_agg["actual_time"]/(df_agg["actual_time"].median())
b = df_agg["segment_actual_time"]/(df_agg["segment_actual_time"].median())


sns.kdeplot(a, color = "red", label = "actual_time")
sns.kdeplot(b, color = 'green', label = "segment_actual_time")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1338cfda0>
```



We observe that the curves are right skewed. So we will be converting them into log normal scale.

```
a = np.log1p(df_agg["segment_actual_time"])
sns.distplot(a)
```

```
<Axes: xlabel='segment_actual_time', ylabel='Density'>
```

Let us test whether the curve is normal or not using the Wilkin Shapiro test

Null Hypothesis: the distribution is taken from a normal distribution.

Alternate hypothesis: the distribution is not normal

p = 0.05

```
## Let us test whether the curve is normal or not
stats.shapiro(a)
```

```
ShapiroResult(statistic=0.9779465198516846, pvalue=1.661939978689233e-42)
```

segment actual time

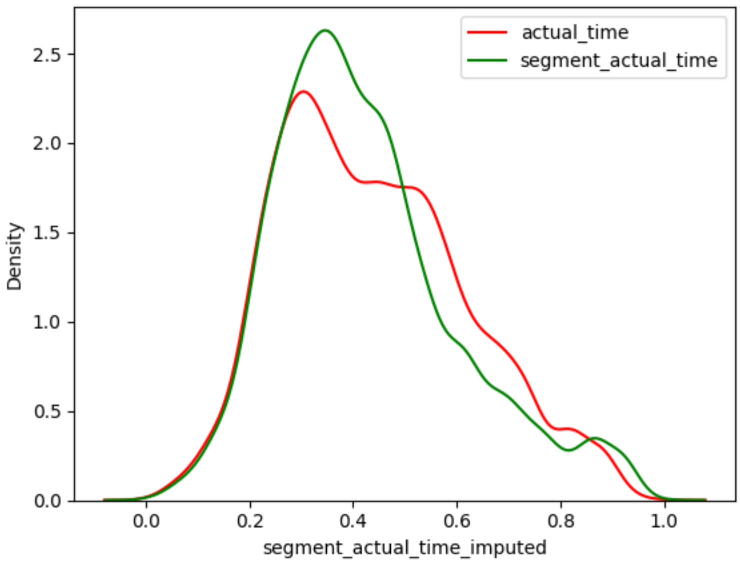
Since p is smaller than 0.05, we can say that we will be rejecting the null hypothesis, so the distribution is not normal.

Scaling the variables

```
df_agg["segment_actual_time_imputed"] = minmax_scaler(a)
df_agg["actual_time_imputed"] = minmax_scaler(np.log1p(df_agg["actual_time"]))
```

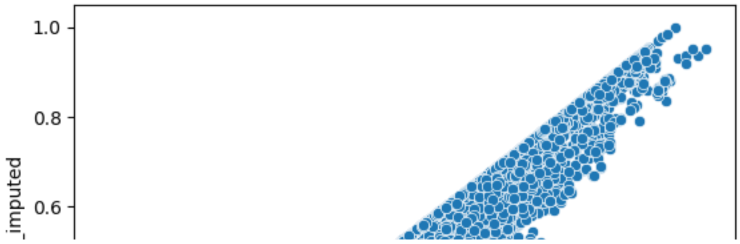
```
sns.kdeplot(df_agg["segment_actual_time_imputed"], color = "red", label = "actual_time")
sns.kdeplot(df_agg["actual_time_imputed"], color = 'green', label = "segment_actual_time")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1397093a0>
```




```
sns.scatterplot(data = df_agg, x = "segment_actual_time_imputed", y =  "actual_time_imputed")
```

```
<Axes: xlabel='segment_actual_time_imputed', ylabel='actual_time_imputed'>
```




```
df_agg["actual_time_imputed"].corr(df_agg["segment_actual_time_imputed"])
```

```
0.9566229910465973
```

It seems that the curve doesn't follow normal distribution, but actual_time and segment_actual_time do follow each other. There is some degree of similiarity. Since the two variables are numerical data, and they don't appear to follow normal distribution we will use KS test.

Null Hypothesis: Actual time aggregated value and segment actual time aggregated value are taken from the same distribution

Alternate Hypothesis: Actual time aggregated value and segment actual time aggregated value are different

Test: KS test for independant samples.

significance level: 0.05

```
stats.kstest(df_agg["segment_actual_time_imputed"],df_agg["actual_time_imputed"])
```

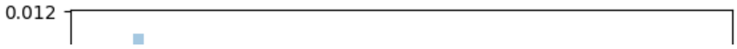
```
KstestResult(statistic=0.08301275561854626, pvalue=7.399132107794807e-45, statistic_location=0.4905899200069058,
statistic_sign=-1)
```

p<<0.05, therefore the alternate hypothesis is true. The results of the test make it clear that the two samples come from different distributions.

4. Comparing osrm distance aggregated value and segment osrm distance aggregated value

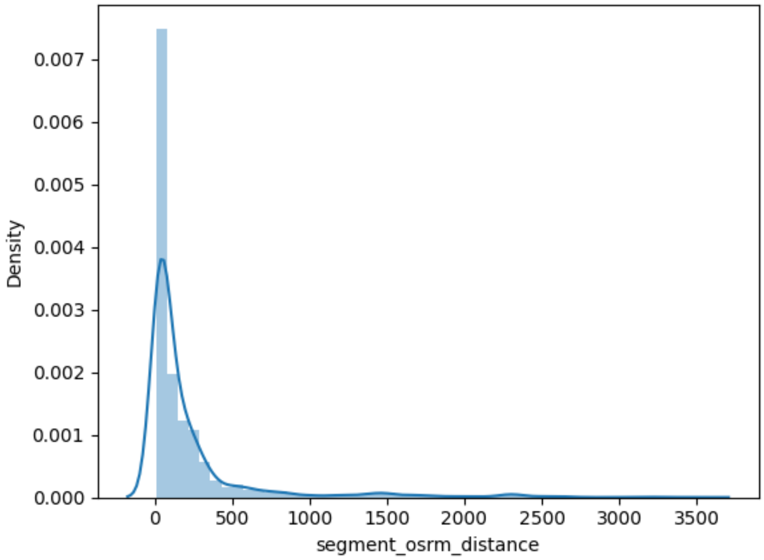
```
#Univariate analysis of df_agg["osrm_distance"]
sns.distplot(df_agg["osrm_distance"])
```

<Axes: xlabel='osrm_distance', ylabel='Density'>



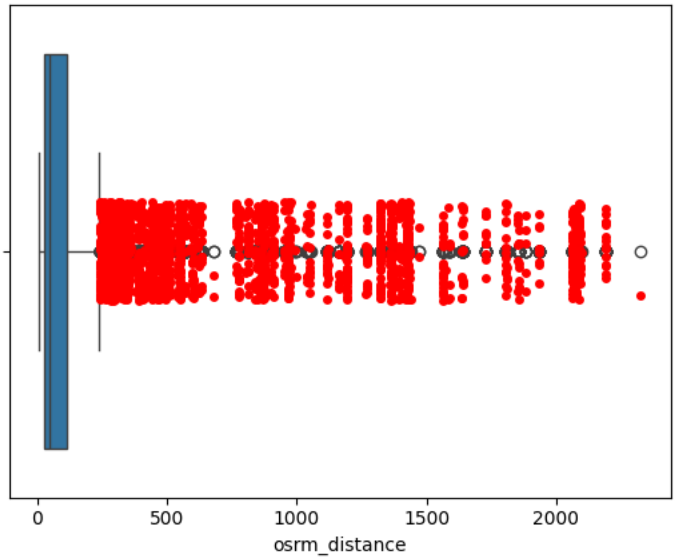

```
#Univariate analysis of df_agg["segment_osrm_distance"]
sns.distplot(df_agg["segment_osrm_distance"])
```

<Axes: xlabel='segment_osrm_distance', ylabel='Density'>



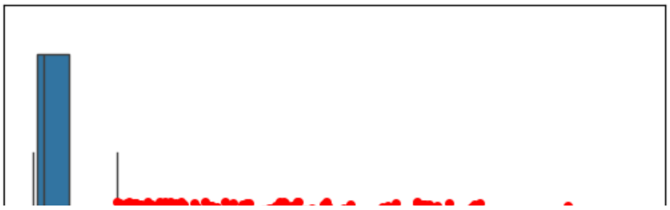

```
# Outliers in "osrm_distance" column
sns.boxplot(x = df_agg["osrm_distance"])
sns.stripplot(x = outlier_func(df_agg["osrm_distance"]), color = "red")
# We find out that there are a lot of outliers above the upper whisker.
```

<Axes: xlabel='osrm_distance'>




```
# Outliers in "segment_osrm_distance" column
sns.boxplot(x = df_agg["segment_osrm_distance"])
sns.stripplot(x = outlier_func(df_agg["segment_osrm_distance"]), color = "red")
# We find out that there are a lot of outliers above the upper whisker.
```

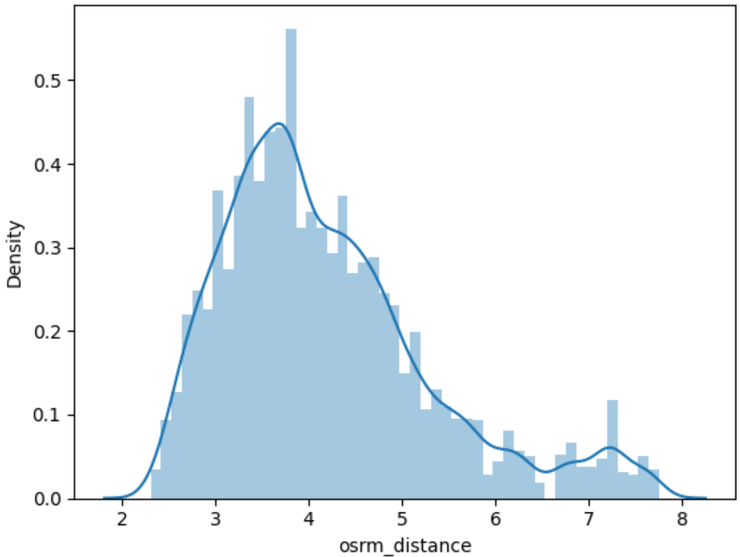
```
<Axes: xlabel='segment_osrm_distance'>
```



Treating outliers

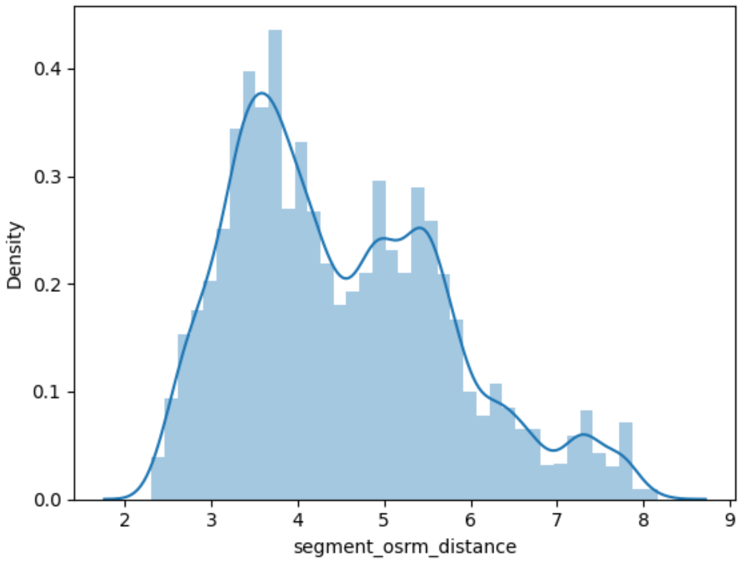
```
a = np.log1p(df_agg["osrm_distance"])
sns.distplot(a)
```

```
<Axes: xlabel='osrm_distance', ylabel='Density'>
```




```
b = np.log1p(df_agg["segment_osrm_distance"])
sns.distplot(b)
```

```
<Axes: xlabel='segment_osrm_distance', ylabel='Density'>
```



Normalizing/standardizing using min - max scaler

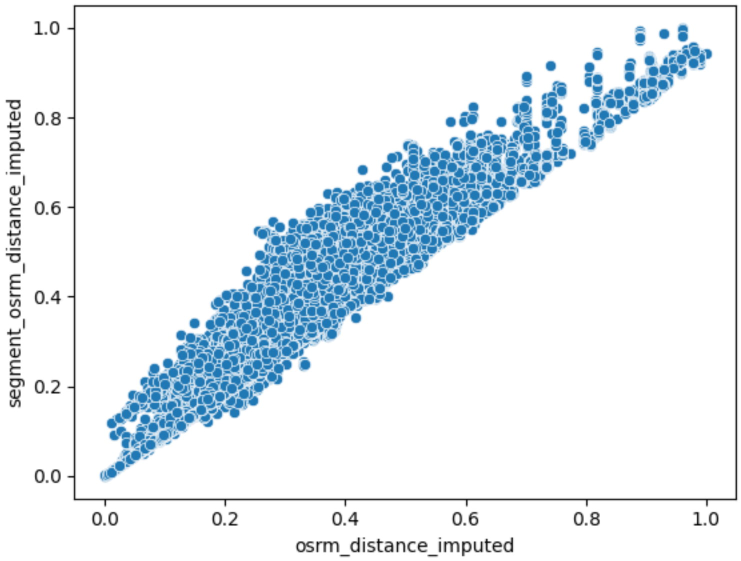
```
df_agg["osrm_distance_imputed"] = minmax_scaler(a)
df_agg["segment_osrm_distance_imputed"] = minmax_scaler(b)
```

```
df_agg["osrm_distance_imputed"].corr(df_agg["segment_osrm_distance_imputed"])
```

```
0.9550972784100195
```

```
sns.scatterplot(data = df_agg, x = "osrm_distance_imputed", y =  "segment_osrm_distance_imputed")
```

```
<Axes: xlabel='osrm_distance_imputed',
ylabel='segment_osrm_distance_imputed'>
```



It seems that the curve doesn't follow normal distribution, but actual_time and segment_actual_time do follow each other. There is some degree of similiarity. Since the two variables are numerical data, and they don't appear to follow normal distribution we will use KS test.

Null Hypothesis: Actual time aggregated value and segment actual time aggregated value are taken from the same distribution

Alternate Hypothesis: Actual time aggregated value and segment actual time aggregated value are different

Test: KS test for independant samples.

significance level: 0.05

```
stats.kstest(df_agg["osrm_distance_imputed"],df_agg["segment_osrm_distance_imputed"])
```

```
KstestResult(statistic=0.11675777822771138, pvalue=1.9249076547636587e-88, statistic_location=0.4374474698869129,
statistic_sign=1)
```

p<<0.05, therefore the alternate hypothesis is true. The results of the test make it clear that the two samples come from different distributions.

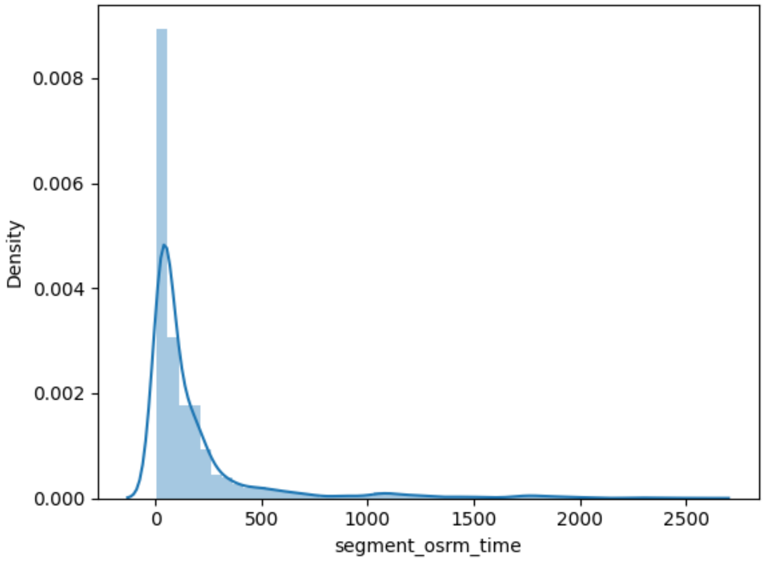
5. Comparing osrm time aggregated value and segment osrm time aggregated value

```
#Univariate analysis of df_agg["osrm_distance"]
sns.distplot(df_agg["osrm_time"])
```

```
<Axes: xlabel='osrm_time', ylabel='Density'>
```

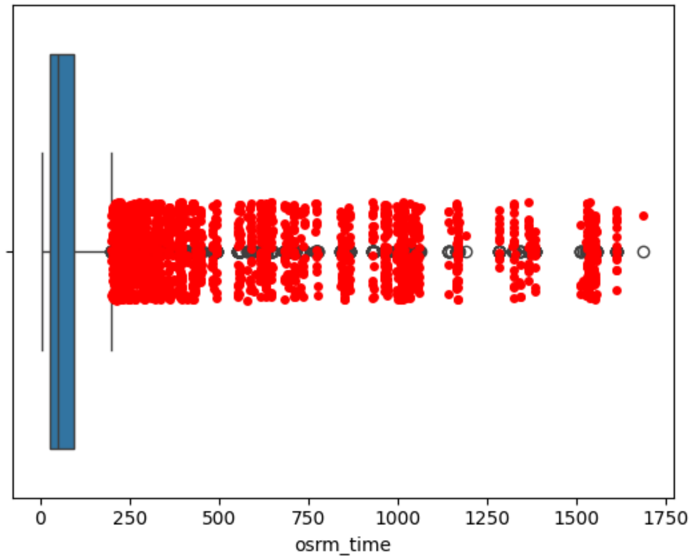

```
#Univariate analysis of df_agg["osrm_distance"]
sns.distplot(df_agg["segment_osrm_time"])
```

```
<Axes: xlabel='segment_osrm_time', ylabel='Density'>
```



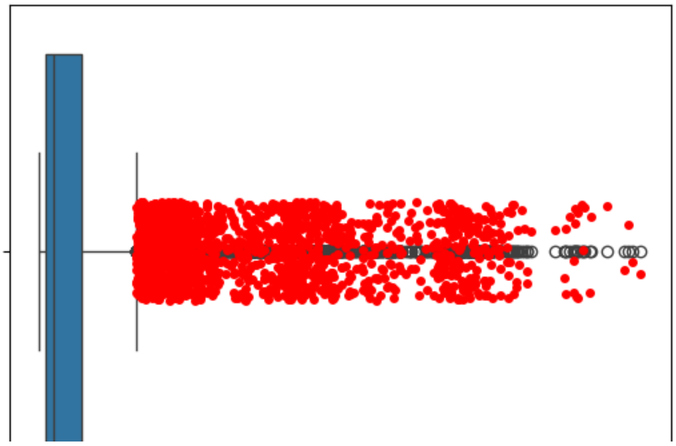

```
# Outliers in "osrm_time" column
sns.boxplot(x = df_agg["osrm_time"])
sns.stripplot(x = outlier_func(df_agg["osrm_time"]), color = "red")
# We find out that there are a lot of outliers above the upper whisker.
```

```
<Axes: xlabel='osrm_time'>
```




```
# Outliers in "segment_osrm_time" column
sns.boxplot(x = df_agg["segment_osrm_time"])
sns.stripplot(x = outlier_func(df_agg["segment_osrm_time"]), color = "red")
# We find out that there are a lot of outliers above the upper whisker.
```

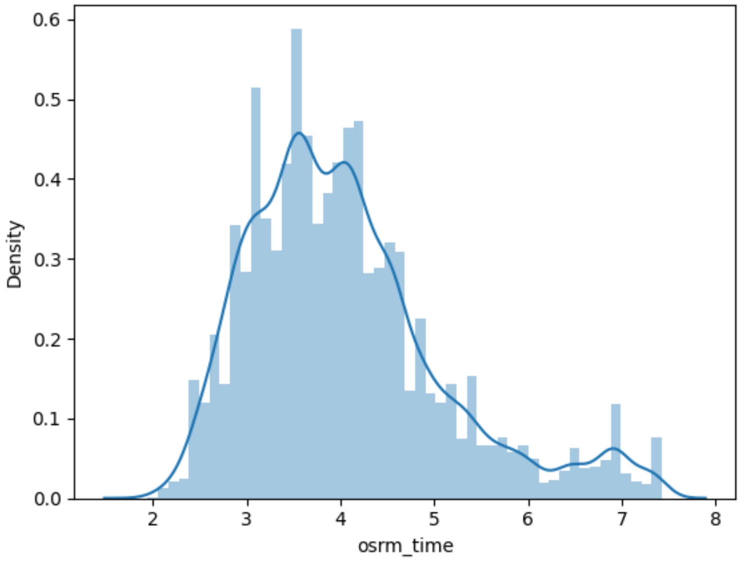
```
<Axes: xlabel='segment_osrm_time'>
```



Treating outliers

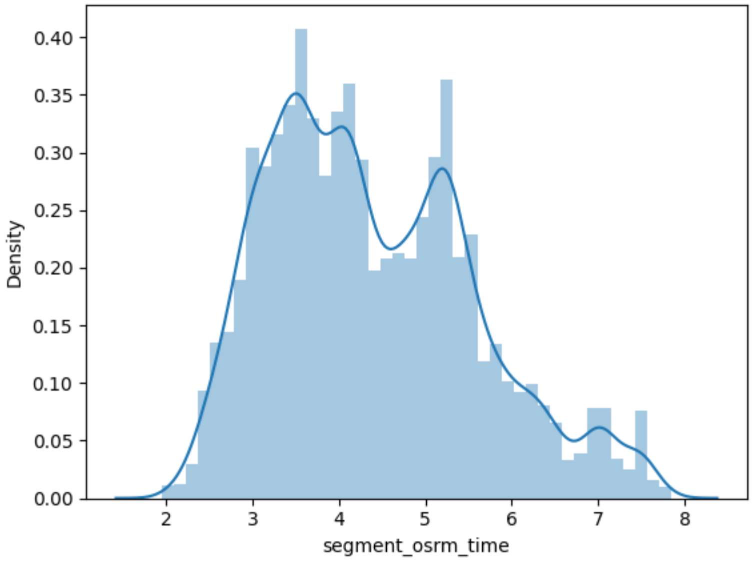
```
a = np.log1p(df_agg["osrm_time"])
sns.distplot(a)
```

```
<Axes: xlabel='osrm_time', ylabel='Density'>
```




```
b = np.log1p(df_agg["segment_osrm_time"])
sns.distplot(b)
```

```
<Axes: xlabel='segment_osrm_time', ylabel='Density'>
```



Normalizing/standardizing using min - max scaler

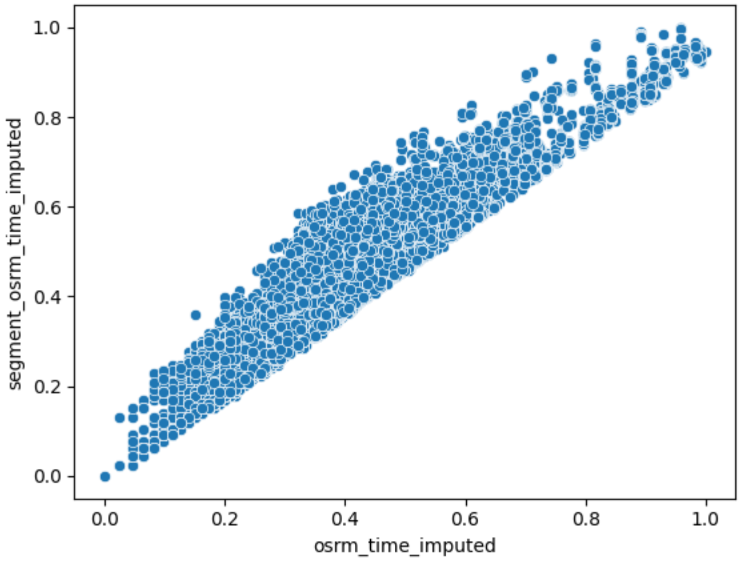
```
df_agg["osrm_time_imputed"] = minmax_scaler(a)
df_agg["segment_osrm_time_imputed"] = minmax_scaler(b)
```

```
df_agg["osrm_time_imputed"].corr(df_agg["segment_osrm_time_imputed"])
```

```
0.948940474436486
```

```
sns.scatterplot(data = df_agg, x = "osrm_time_imputed", y =  "segment_osrm_time_imputed")
```

```
<Axes: xlabel='osrm_time_imputed', ylabel='segment_osrm_time_imputed'>
```



It seems that the curve doesn't follow normal distribution, but segment_osrm_time_imputed and osrm_time_imputed do follow each other. There is some degree of similiarity. Since the two variables are numerical data, and they don't appear to follow normal distribution we will use KS test.

Null Hypothesis: osrm_time_imputed and segment_osrm_time_imputed are taken from the same distribution

Alternate Hypothesis: osrm_time_imputed and segment_osrm_time_imputed are different

Test: KS test for independant samples.

significance level: 0.05

```
stats.kstest(df_agg["osrm_time_imputed"],df_agg["segment_osrm_time_imputed"])
```

```
KstestResult(statistic=0.1368698117027738, pvalue=1.6523278092778216e-121, statistic_location=0.49199282722919,
statistic_sign=1)
```

p<<0.05, therefore the alternate hypothesis is true. The results of the test make it clear that the two samples come from different distributions.

One hot encoding of the columns - 'data' and 'route_type'

```
df_agg = pd.get_dummies(df_agg, columns = ["data", "route_type"])
df_agg
```

	trip_uuid	trip_creation_time	route_schedule_uuid	start_scan_t
0	trip-153671041653548748	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	
1	trip-153671042288605164	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	
2	trip-153671043369099517	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	
3	trip-153671046011330457	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	
4	trip-153671052974046625	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	
...	...	...	...	
14812	trip-153861095625827784	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	
14813	trip-153861104386292051	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	
14814	trip-153861106442901555	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	
14815	trip-153861115439069069	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	
14816	trip-153861118270144424	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	

14817 rows × 33 columns

Business Insights:

1. Full Truck Load orders are twice as many as Carting orders.

2. Most orders are coming from Gurgaon Bilaspur - Harayana.

3. Most orders are going to Gurgaon Bilaspur - Harayana.

4. Actual time for each segment has a very big range, but most of the order are completed within 50h

5. Segment actual time is usually always more than segment OSRM time. Therefore it is evident that orders always gets delayed from the estimated time.

```
np.mean(df_agg["actual_distance_to_destination"]/df_agg["osrm_distance"])
```

```
0.7795901078631176
```

6. On average actual distance is less than OSRM distance.

```
df_agg["source_state"].mode()
```

```
0    Maharashtra
Name: source_state, dtype: object
```

7. The state with highest number of orders sourced from is Maharashtra.

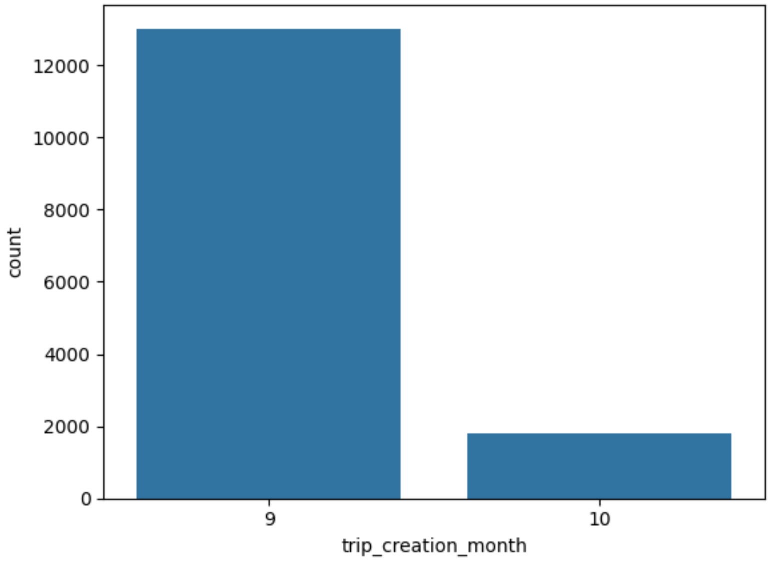
```
df_agg["destination_state"].mode()
```

```
0    Maharashtra
Name: destination_state, dtype: object
```

8. The state with highest number of orders delivered to is also Maharashtra.

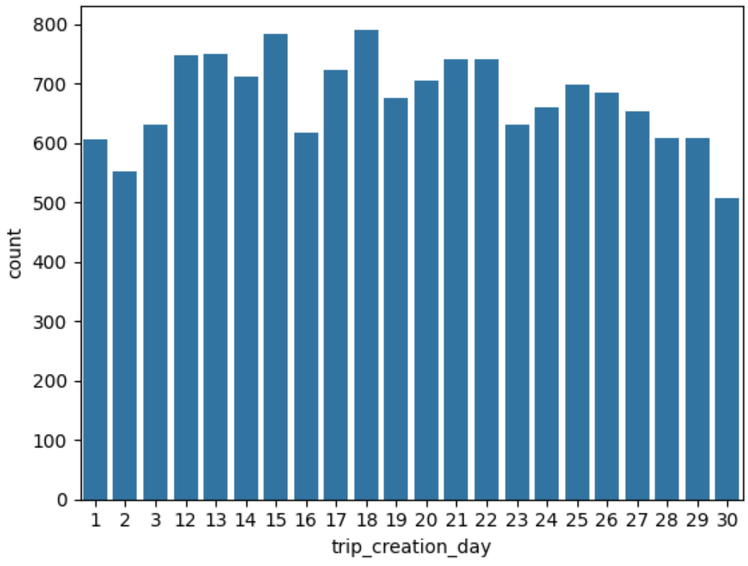
```
sns.countplot(data = df_agg, x = "trip_creation_month")
```

```
<Axes: xlabel='trip_creation_month', ylabel='count'>
```




```
sns.countplot(data = df_agg, x = "trip_creation_day")
```

```
<Axes: xlabel='trip_creation_day', ylabel='count'>
```



9. Most of the trips are created in september month.

```
df_agg.describe()
```

	trip_creation_time	start_scan_to_end_scan	trip_creation_year	trip_cr
count	14817	14817.000000	14817.0	
mean	2018-09-22 12:44:19.555167744	199.613012	2018.0	
min	2018-09-12 00:00:16.535741	22.000000	2018.0	
25%	2018-09-17 02:51:25.129125888	108.000000	2018.0	

10. Average distance between source and destination is 125 Km, where as the average time is 278h

2018-10-03

```
df_agg.groupby(["source_city", "destination_city"])["trip_uuid"].count().sort_values(ascending = False)
```

```
source_city         destination_city
Bengaluru           Bengaluru           528
Bangalore           Bengaluru           492
Bhiwandi            Mumbai              407
Bengaluru           Bangalore           336
Hyderabad           Hyderabad           308
                                       ...
Almora              Pithorgarh            1
                    Ranikhet              1
Vadodara (Gujarat)  Vadodara              1
Vaijiapur           Nashik                1
Vapi                Daman                 1
Name: trip_uuid, Length: 1407, dtype: int64
```

11. We see that the busiest routes are from Bangalore to Bangalore and Bhiwandi to Mumbai.

Recommendations

1. Delhivery should optimise and prioritize FTL packages as they make up signicantly more number of orders.

2. Delhivery should decentralize and develop new warehouses around Gurgaon, since it handles the highest number of orders in and out. So that it is not over burdened and over dependent.

4. Actual time for some of the orders are very large, so delhivery should make warehouses in remote areas where it is taking a lot of time to deliver

5. Since actual time is always more than OSRM time, we need to improve the OSRM time estimation which is not able to estimate time properly.

6. Since actual time is always more than OSRM time, we need to investigate why there are frequent delays and build necessary logistic capabilities to reduce the time.

7. Since actual distance is always more than OSRM distance, we need to improve the OSRM distance estimation which is not able to estimate distance properly.

8. Special emphasis needs to be given to warehouses in Maharastra, logistic facility needs to be upgraded since a lot of orders are sourced and delivered here.

9. Special emphasis needs to be given to routes from Bangalore to Bangalore and Bhiwandi to Mumbai. Logistic facility needs to be upgraded since a lot of orders are sourced and delivered here.

10. Average distance is 125Km and average time is 278h, which gives around 0.5km travelled for every hour. Therefore speed is very slow. Delhivery needs to ask its drivers to increase speed, decrease warehouse downtime which eats away a lot of time.