Delhivery

April 25, 2023

[125]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="darkgrid")
import warnings
warnings.filterwarnings('ignore')

from scipy.stats import pearsonr, spearmanr # For correlation testing
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[126]:
```
df = pd.read_csv("delhivery_data.csv")
```

[127]:
```
df.head(5)
```

[127]:
```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

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

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

  destination_center              destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
```

```
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)

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

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

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

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

1 1. Basic data cleaning and exploration:

1.0.1 1. Analyze the structure of the data.

[128]:
```python
df.shape
```

[128]:
```
(144867, 24)
```

[129]:
```python
df.describe()
```

[129]:
```
       start_scan_to_end_scan  cutoff_factor  actual_distance_to_destination  \
count           144867.000000  144867.000000                   144867.000000
```

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

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

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

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

[130]:
```python
df.describe(include="object")
```

[130]:
```
           data           trip_creation_time  \
count    144867                       144867
unique        2                        14817
top    training  2018-09-28 05:23:15.359220
freq     104858                          101

                                      route_schedule_uuid route_type  \
```

```
count                                               144867     144867
unique                                                1504          2
top     thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f…        FTL
freq                                                  1812      99660

                      trip_uuid source_center                 source_name  \
count                    144867        144867                      144574
unique                    14817          1508                        1498
top     trip-153811219535896559  IND000000ACB  Gurgaon_Bilaspur_HB (Haryana)
freq                        101         23347                       23347

       destination_center               destination_name  \
count              144867                         144606
unique               1481                           1468
top          IND000000ACB  Gurgaon_Bilaspur_HB (Haryana)
freq                15192                          15192

                     od_start_time                 od_end_time  \
count                       144867                      144867
unique                       26369                       26369
top     2018-09-21 18:37:09.322207  2018-09-24 09:59:15.691618
freq                            81                          81

           cutoff_timestamp
count                144867
unique                93180
top     2018-09-24 05:19:20
freq                     40
```

[131]:
```
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
```

1.0.2 2. Handle missing values in the data.

[132]:
```
df.isnull().sum()
# Missing values are in source_name and destination_name
# we can check
```

[132]:
```
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
```

[133]:
```
## Condtion we created for this kind of missing values ##
# *******************************************************

# Splitted the data by missing values and count the number of missing values␣
↪for each particular source_center
# Then for each source_center, check if there any matching values in␣
↪source_center and source_center column of main data
# Calculate the number of missing count of each center and compare with whole␣
↪data count if both are same then that particular data won't be anywhere in␣
↪our dataset
# If the above condition is true we can drop the null values
```

Missing values check for source_name feature

[134]:
```
miss_count = df[df['source_name'].apply(pd.isna)]["source_center"].
↪value_counts().reset_index()
cant_modify = []
can_modify = []
for i in miss_count["index"]:
    if (df[df["source_center"] == f"{i}"]["source_name"].apply(pd.isna).sum()␣
↪== miss_count[miss_count["index"] == f"{i}"]["source_center"].values[0]) and␣
↪(df[df["source_center"] == f"{i}"]["source_name"].apply(pd.isna).sum() ==␣
↪df[df["source_center"] == f"{i}"].shape[0]):
        cant_modify.append(i)
    else:
        can_modify.append(i)
if can_modify == []:
    print("We can drop the rows of source_name which are having missing values,␣
↪there is no way to assume any value there")
```

```
We can drop the rows of source_name which are having missing values, there is no
way to assume any value there
```

Missing values check for source_name feature

[135]:
```
miss_count = df[df['source_name'].apply(pd.isna)]["source_center"].
↪value_counts().reset_index()
cant_modify = []
can_modify = []
for i in miss_count["index"]:
    if df[df["source_center"] == f"{i}"]["source_name"].apply(pd.isna).sum() ==␣
↪miss_count[miss_count["index"] == f"{i}"]["source_center"].values[0] and␣
↪(df[df["source_center"] == f"{i}"]["source_name"].apply(pd.isna).sum() ==␣
↪df[df["source_center"] == f"{i}"].shape[0]):
        cant_modify.append(i)
    else:
        can_modify.append(i)
if can_modify == []:
```

```
    print("We can drop the rows of source_name which are having missing values,␣
↪there is no way to assume any value there")
```

```
We can drop the rows of source_name which are having missing values, there is no
way to assume any value there
```

[136]:
```
# Dropping the missing values
df = df.dropna(axis=0)
```

[137]:
```
df.isnull().sum()
```

[137]:
```
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
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
```

[138]:
```
# All missing values are dropped, there is no way to fill up with other values␣
↪or with any aggregation values
# Hence we dropped missing values
```

1.0.3 3. Merging the rows

[139]:
```
# Merging the rows with groupby of trip id, source center, destination center␣
↪and aggregate sum by segment time and max by actual cumulative time
# So that the we can able to fetch data of source and destination with their␣
↪actual time taken and total segment time taken
```

[140]:
```
groupby_trip_source_dest = df.
 ↪groupby(["trip_uuid","source_name","destination_name"]).agg(
    {
        "segment_actual_time":"sum",
        "segment_osrm_time":"sum",
        "segment_osrm_distance":"sum",
        "actual_time":"max",
        "osrm_time":"max",
        "osrm_distance":"max"
    }).reset_index()
```

[141]:
```
merged_data = groupby_trip_source_dest.groupby("trip_uuid").agg(
    {
        "source_name":"first",
        "destination_name":"last",
        "segment_actual_time":"sum",
        "segment_osrm_time":"sum",
        "segment_osrm_distance":"sum",
        "actual_time":"sum",
        "osrm_time":"sum",
        "osrm_distance":"sum"
    }).reset_index()
```

[ ]:
```

```

[142]:
```
merged_data
```

[142]:
```
                     trip_uuid                         source_name  \
0      trip-153671041653548748  Bhopal_Trnsport_H (Madhya Pradesh)
1      trip-153671042288605164   Doddablpur_ChikaDPP_D (Karnataka)
2      trip-153671043369099517    Bangalore_Nelmngla_H (Karnataka)
3      trip-153671046011330457            Mumbai Hub (Maharashtra)
4      trip-153671052974046625                Bellary_Dc (Karnataka)
…                          …                                   …
14782  trip-153861095625827784        Chandigarh_Mehmdpur_H (Punjab)
14783  trip-153861104386292051          FBD_Balabhgarh_DPC (Haryana)
14784  trip-153861106442901555  Kanpur_Central_H_6 (Uttar Pradesh)
14785  trip-153861115439069069          Eral_Busstand_D (Tamil Nadu)
14786  trip-153861118270144424                   Hospet (Karnataka)

                        destination_name  segment_actual_time  \
0            Gurgaon_Bilaspur_HB (Haryana)               1548.0
1        Doddablpur_ChikaDPP_D (Karnataka)                141.0
2           Chandigarh_Mehmdpur_H (Punjab)               3308.0
3           Mumbai_MiraRd_IP (Maharashtra)                 59.0
4                   Bellary_Dc (Karnataka)                340.0
…                                    …                   …
```

```
14782      Chandigarh_Mehmdpur_H (Punjab)                 82.0
14783      Faridabad_Blbgarh_DC (Haryana)                 21.0
14784  Kanpur_Central_H_6 (Uttar Pradesh)                281.0
14785        Eral_Busstand_D (Tamil Nadu)                258.0
14786              Bellary_Dc (Karnataka)                274.0

       segment_osrm_time  segment_osrm_distance  actual_time  osrm_time  \
0                 1008.0              1320.4733       1562.0      743.0
1                   65.0                84.1894        143.0       68.0
2                 1941.0              2545.2678       3347.0     1741.0
3                   16.0                19.8766         59.0       15.0
4                  115.0               146.7919        341.0      117.0
…                    …                    …            …         …
14782               62.0                64.8551         83.0       62.0
14783               11.0                16.0883         21.0       12.0
14784               88.0               104.8866        282.0       54.0
14785              221.0               223.5324        264.0      184.0
14786               67.0                80.5787        275.0       68.0

       osrm_distance
0           991.3523
1            85.1110
2          2372.0852
3            19.6800
4           146.7918
…             …
14782        73.4630
14783        16.0882
14784        63.2841
14785       177.6635
14786        80.5787

[14787 rows x 9 columns]
```

[ ]:

2 2. Build some features to prepare the data for actual analysis. Extract features from the below fields:

2.0.1 1. Destination Name: Split and extract features out of destination. City-place-code (State)

Seperator function to split

[143]:
```
def seperator(x):
    res = x.split("_")
    if len(res) == 2:
        second_split = res[1].split(" ")
```

```
        res.pop()
        for i in second_split:
            res.append(i)
    elif len(res) == 1:
        third_split = res[0].split(" ")
        res.pop()
        for i in third_split:
            res.append(i)
        if len(res) <= 2:
            res.append(third_split[-1])
    return res if len(res) == 3 else res[:3]
```

[144]:
```
df["destination_name"]
```

[144]:
```
0          Khambhat_MotvdDPP_D (Gujarat)
1          Khambhat_MotvdDPP_D (Gujarat)
2          Khambhat_MotvdDPP_D (Gujarat)
3          Khambhat_MotvdDPP_D (Gujarat)
4          Khambhat_MotvdDPP_D (Gujarat)
                      …
144862    Gurgaon_Bilaspur_HB (Haryana)
144863    Gurgaon_Bilaspur_HB (Haryana)
144864    Gurgaon_Bilaspur_HB (Haryana)
144865    Gurgaon_Bilaspur_HB (Haryana)
144866    Gurgaon_Bilaspur_HB (Haryana)
Name: destination_name, Length: 144316, dtype: object
```

[145]:
```
# Split the destination name column with "_" where we observerd this is the␣
↪delimiter for destination name.
# Dropped the unwanted columns
# Appending the data to Main Dataframe
destination = pd.DataFrame(df["destination_name"].apply(seperator).tolist(),␣
↪index=df.trip_uuid).reset_index()
destination.columns = ["id","City","Place","State"]
df["destination_city"] = destination["City"].to_numpy()
df["destination_place"] = destination["Place"].to_numpy()
df["destination_state"] = destination["State"].to_numpy()
```

2.0.2 2. Source Name: Split and extract features out of destination. City-place-code (State)

[146]:
```
df["source_name"]
```

[146]:
```
0          Anand_VUNagar_DC (Gujarat)
1          Anand_VUNagar_DC (Gujarat)
2          Anand_VUNagar_DC (Gujarat)
3          Anand_VUNagar_DC (Gujarat)
```

```
4         Anand_VUNagar_DC (Gujarat)
                     …
144862    Sonipat_Kundli_H (Haryana)
144863    Sonipat_Kundli_H (Haryana)
144864    Sonipat_Kundli_H (Haryana)
144865    Sonipat_Kundli_H (Haryana)
144866    Sonipat_Kundli_H (Haryana)
Name: source_name, Length: 144316, dtype: object
```

[147]:
```
# Split the source name column with "_" where we observerd this is the␣
↪delimiter for source name.
# Dropped the unwanted columns
# Appending the data to Main Dataframe
source = pd.DataFrame(df["source_name"].apply(seperator).tolist(), index=df.
↪trip_uuid).reset_index()
source.columns = ["id","City","Place","Code"]
df["source_city"] = source["City"].to_numpy()
df["source_place"] = source["Place"].to_numpy()
df["source_code"] = source["Code"].to_numpy()
```

[148]:
```
df.isna().sum()
```

[148]:
```
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
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
destination_city                  0
```

```
destination_place           0
destination_state           0
source_city                 0
source_place                0
source_code                 0
dtype: int64
```

2.0.3 3. Trip_creation_time: Extract features like month, year and day etc

[149]:
```
# First will convert the whole column into datetime dtype
# Then will split this into multiple features
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"])
df["Trip_Year"] = df["trip_creation_time"].dt.year
df["Trip_Month"] = df["trip_creation_time"].dt.month_name()
df["Trip_day"] = df["trip_creation_time"].dt.day
```

3 3. In-depth analysis and feature engineering:

3.0.1 1. Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

[150]:
```
# od_start_time - Trip start time
# od_end_time - Trip end time
# For calculating difference between od_start and od_end we can find out the␣
↪original time taken by the order.
df["od_start_time"] = pd.to_datetime(df["od_start_time"])
df["od_end_time"] = pd.to_datetime(df["od_end_time"])
df["time_diff_min"] = (df["od_end_time"] - df["od_start_time"]).dt.
↪total_seconds()/60
```

3.0.2 2. Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

[151]:
```
# start_scan_to_end_scan - Time taken to deliver from source to destination
# time_diff_min - Calculated time diff by their actual timings
```

Hypothetical testing for calculated timing and actual timing

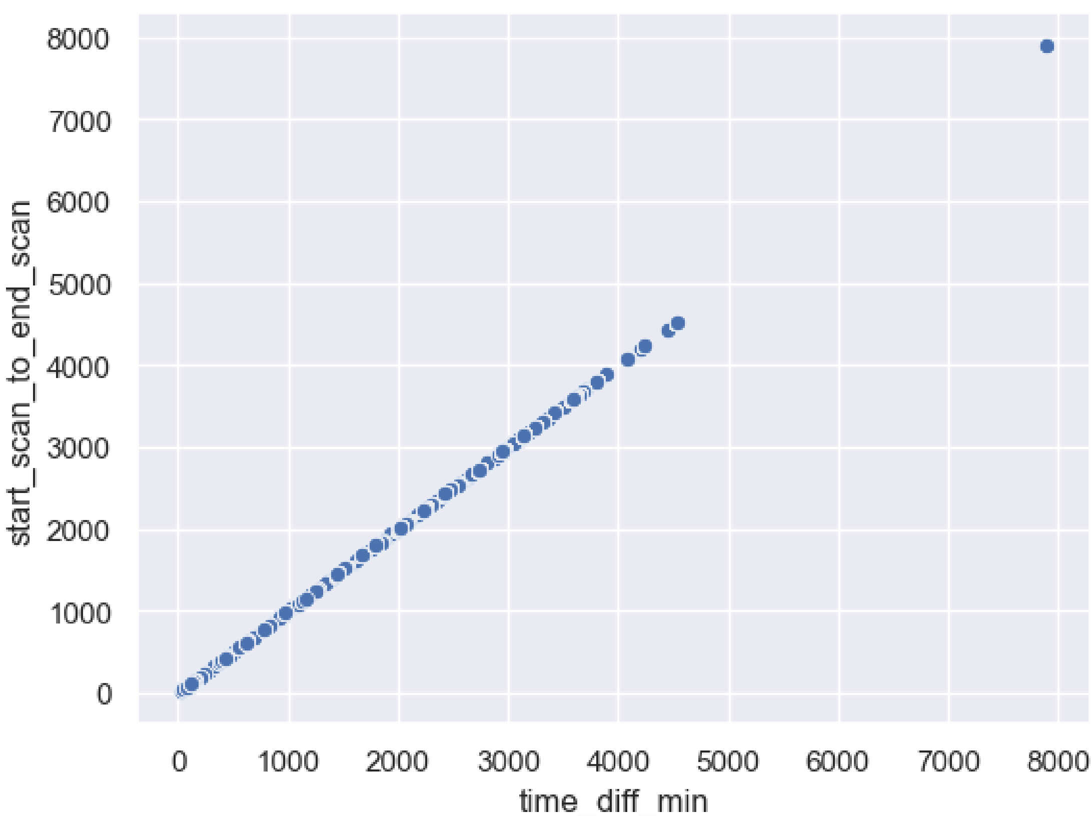
[152]:
```
# H0: Both are not correlated
# Ha: Both are correlated

## properties of two variables ##
# 1. Both features are continuous variables
# 2. Data is right skewed in nature

## Correlation Testing ##
alpha = 0.05
```

```
corr_stat, p_value = pearsonr(df["start_scan_to_end_scan"], df["time_diff_min"])
if p_value<alpha:
    print("Reject Null Hypothesis")
else:
    print("Fail to reject Null Hypothesis")
print("Test Statistic Value: ",corr_stat)
print("P_value:",p_value)
```

```
Reject Null Hypothesis
Test Statistic Value:  0.9999999609905782
P_value: 0.0
```

[153]:
```
sns.scatterplot(data=df,x="time_diff_min", y="start_scan_to_end_scan")
```

[153]: <AxesSubplot:xlabel='time_diff_min', ylabel='start_scan_to_end_scan'>



[154]:
```
# Inference
# Both features are highly correlated
# Even test confirm the same and graph also tells the same
# Our calculated timings and actual timings both are same there is high␣
↪colinearity
```

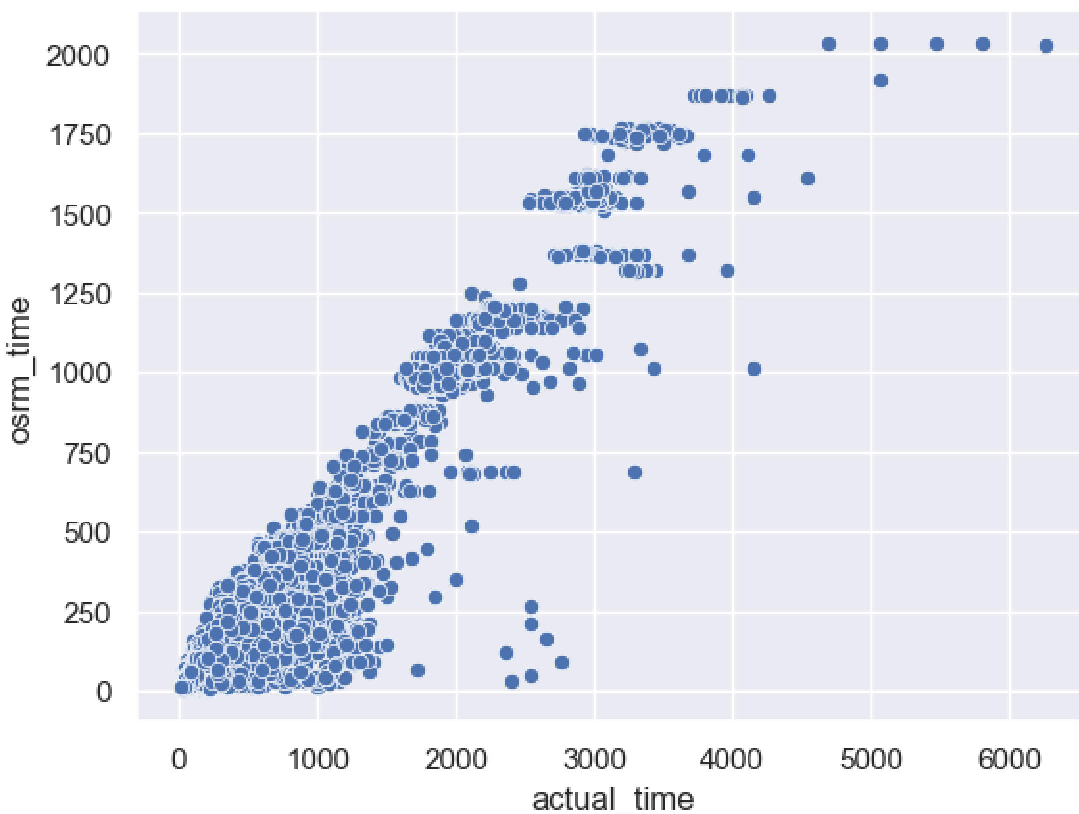
```
# we have created a another feature with 95 % confident
```

3.0.3 3. Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Hypothesis Testing for actual_time aggregated value and OSRM time aggregated value

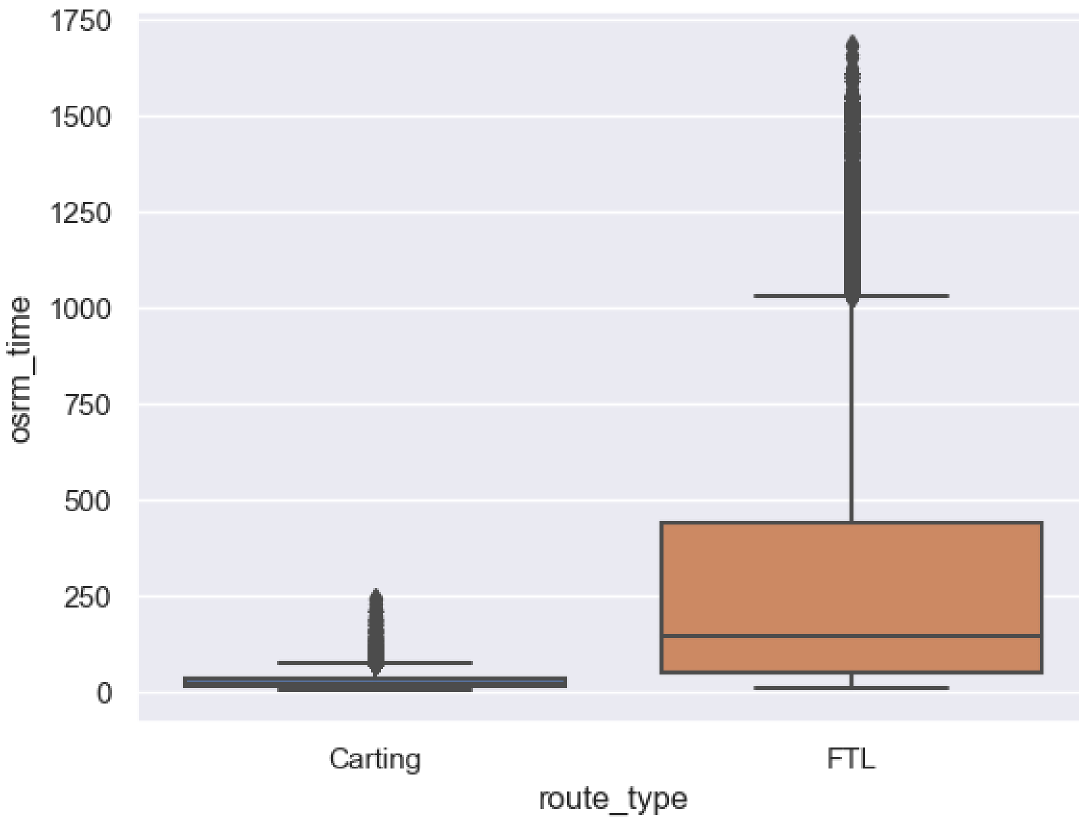
[155]:
```
# Lets calculate this with visual analysis
# Both data is continuous. Hence will use scatter plot to analyze
sns.scatterplot(data=merged_data,x="actual_time",y="osrm_time")
```

[155]: <AxesSubplot:xlabel='actual_time', ylabel='osrm_time'>



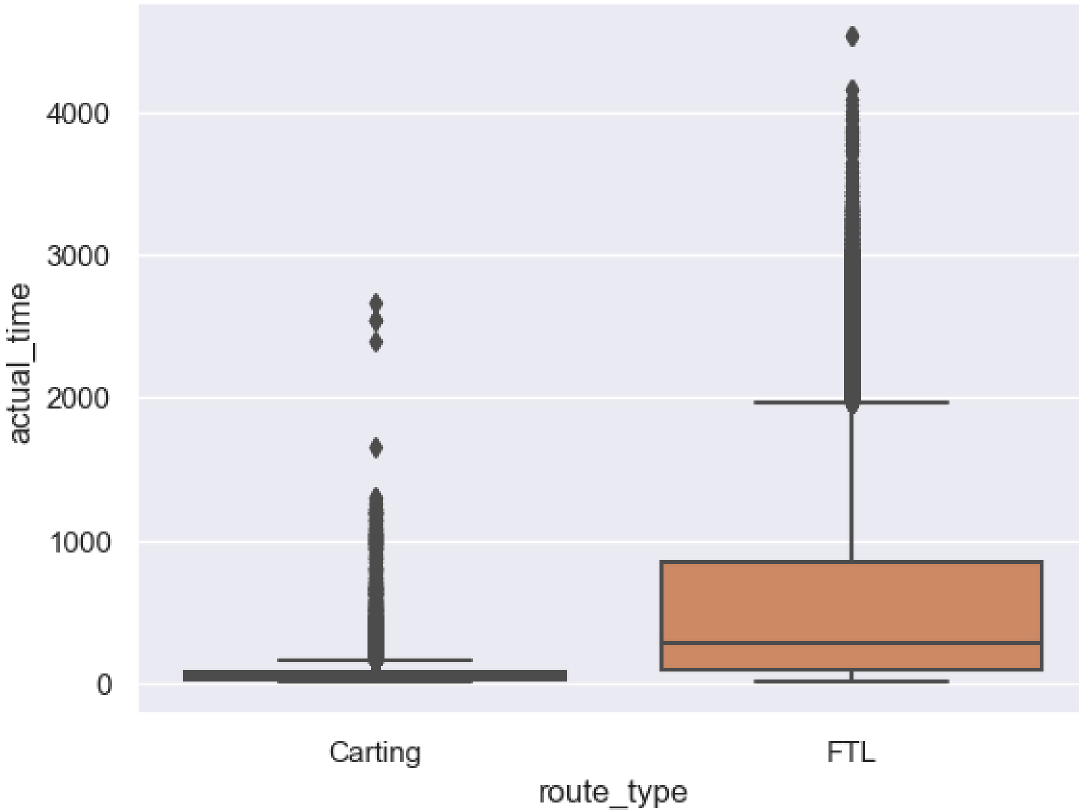
[156]:
```
sns.boxplot(data=df,x="route_type",y="osrm_time")
```

[156]: <AxesSubplot:xlabel='route_type', ylabel='osrm_time'>



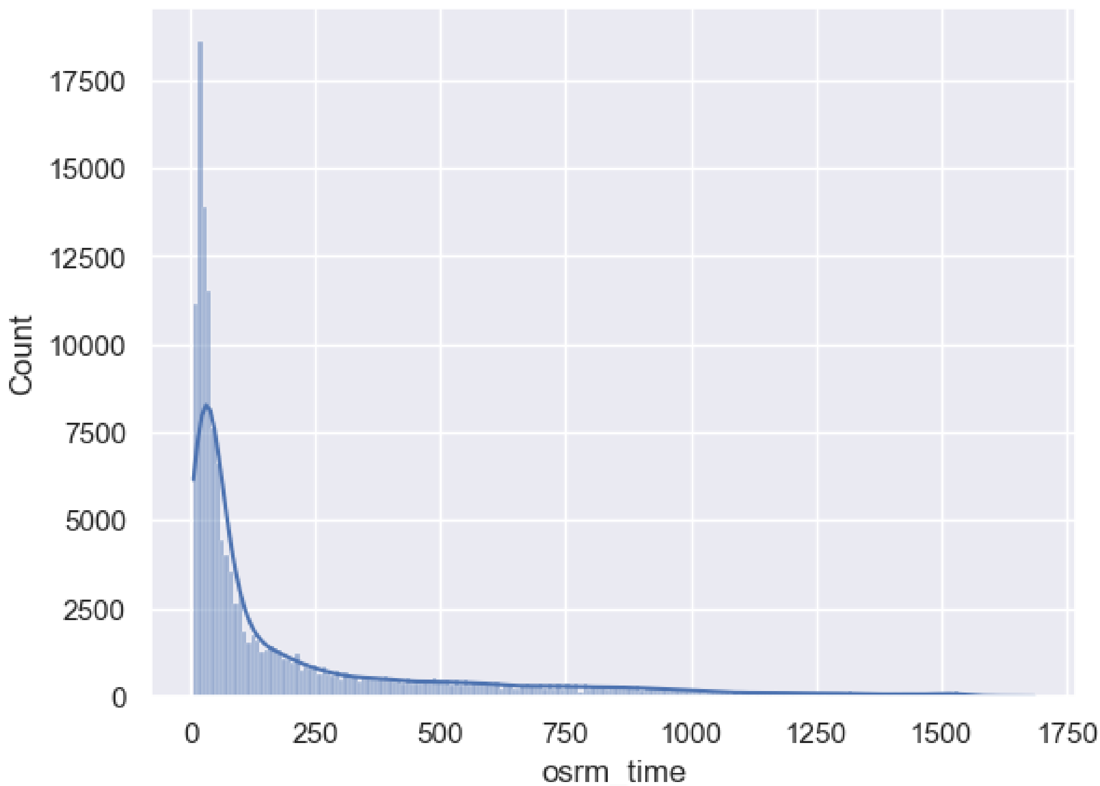
[157]:
```
sns.boxplot(data=df,x="route_type",y="actual_time")
```

[157]: <AxesSubplot:xlabel='route_type', ylabel='actual_time'>



[158]:
```
sns.histplot(data=df,x="osrm_time",kde=True)
```

[158]: <AxesSubplot:xlabel='osrm_time', ylabel='Count'>



[159]:
```
# Inference of Visual Plot

# Plot is look like positive correlation between those variables
# But looks like there is some outliers are present in data
# Anyhow lets test our data to hypothetical testing
```

[160]:
```
# H0: Both are not correlated
# Ha: Both are correlated

## properties of two variables ##
# 1. Both features are continuous variables
# 2. Data is right skewed in nature
# 3. There is lots of outliers are there

## Correlation Testing ##
alpha = 0.05
corr_stat, p_value = pearsonr(merged_data["actual_time"],␣
 ↪merged_data["osrm_time"])
if p_value<alpha:
    print("Reject Null Hypothesis")
else:
```

```
    print("Fail to reject Null Hypothesis")
print("Test Statistic Value: ",corr_stat)
print("P_value:",p_value)
```

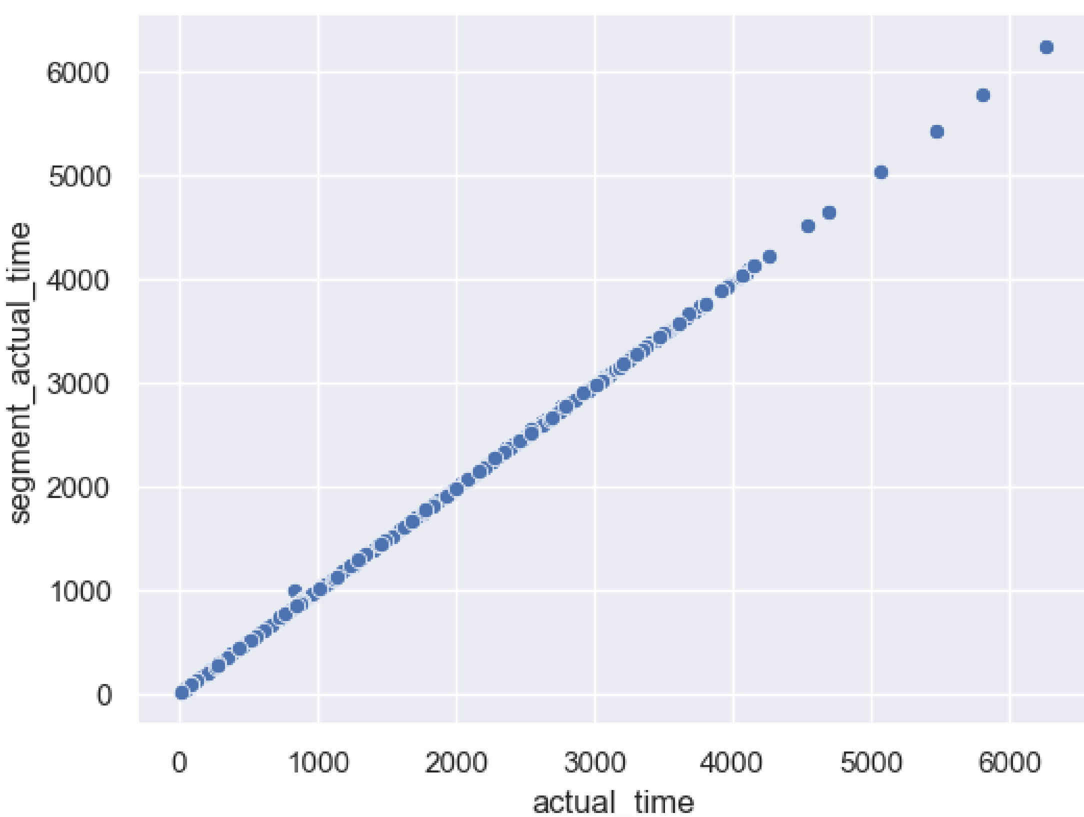
```
Reject Null Hypothesis
Test Statistic Value:  0.9587749744242271
P_value: 0.0
```

[161]:
```
# Test_Results
# 1. There is a high correlation between these variables
# 2. Visually also its proved and hypothetically also its proved
# 3. The actual time and open-source routing engine timings both are same
```

3.0.4 4. Hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value

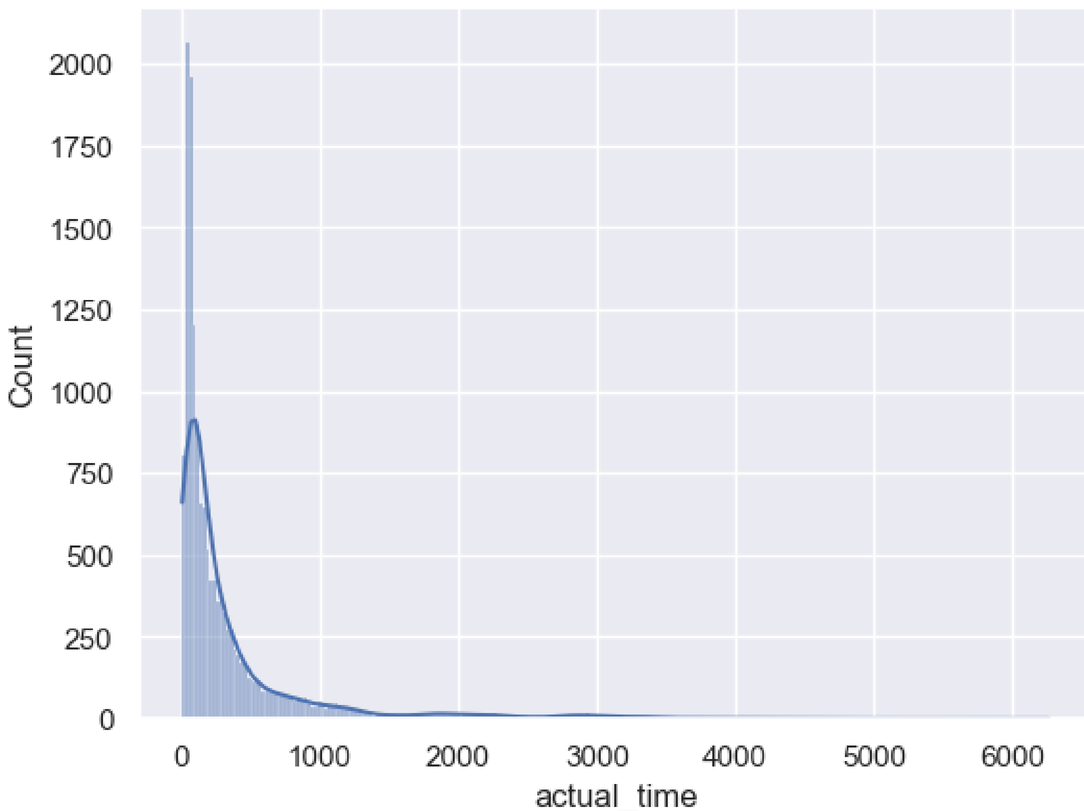
[162]:
```
# Visual Analysis
sns.scatterplot(data=merged_data,x="actual_time",y="segment_actual_time")

# From ploting we can find out the both actual time and segment actual time are␣
↪almost same
# There is high correlation between these variables
```

[162]:
```
<AxesSubplot:xlabel='actual_time', ylabel='segment_actual_time'>
```



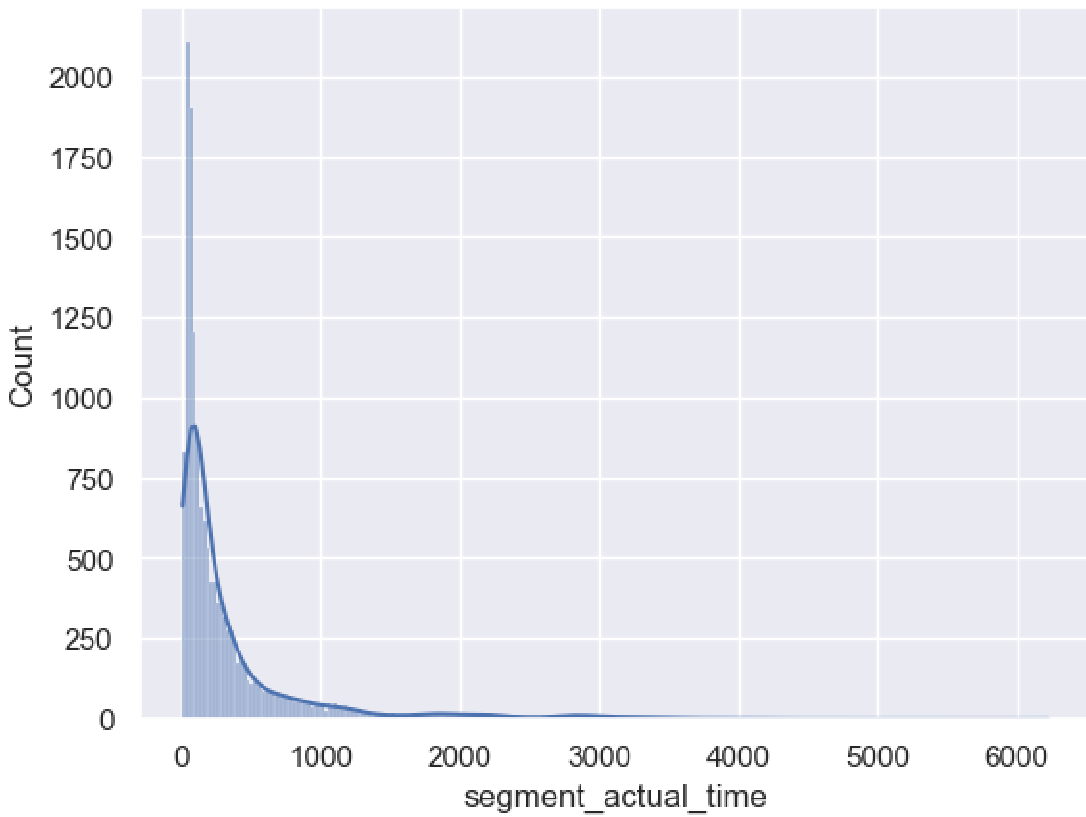
[163]:
```
sns.histplot(data=merged_data,x="actual_time",kde=True)
```

[163]: <AxesSubplot:xlabel='actual_time', ylabel='Count'>



[164]:
```
sns.histplot(data=merged_data,x="segment_actual_time",kde=True)
```

[164]: <AxesSubplot:xlabel='segment_actual_time', ylabel='Count'>



[165]:
```
# H0: Both are not correlated
# Ha: Both are correlated

## properties of two variables ##
# 1. Both features are continuous variables
# 2. Data is right skewed in nature
# 3. There is lots of outliers are there

## Correlation Testing ##
alpha = 0.05
corr_stat, p_value = pearsonr(merged_data["actual_time"],␣
 ↪merged_data["segment_actual_time"])
if p_value<alpha:
    print("Reject Null Hypothesis")
else:
    print("Fail to reject Null Hypothesis")
print("Test Statistic Value: ",corr_stat)
print("P_value:",p_value)
```

```
Reject Null Hypothesis
Test Statistic Value:  0.9999889423463791
```

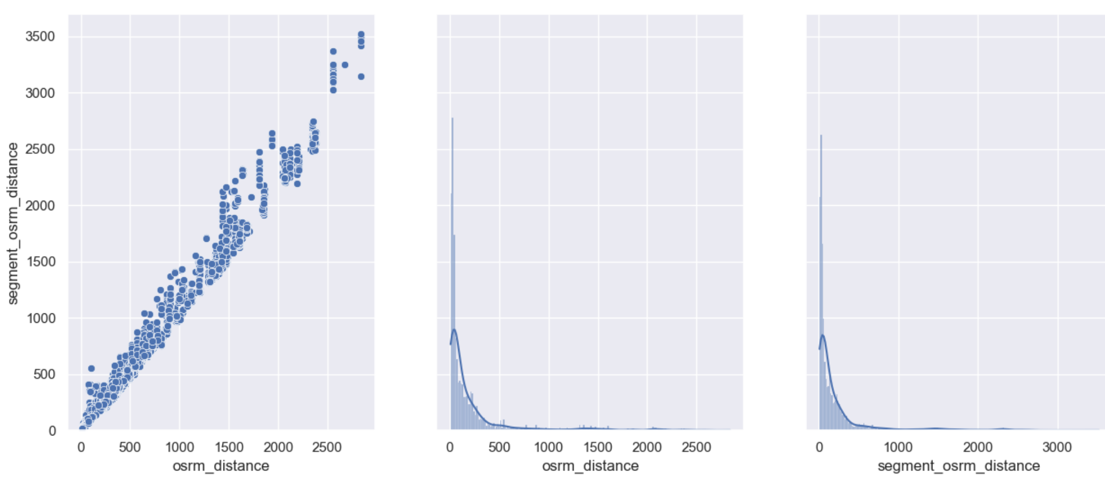
```
P_value: 0.0
```

[166]:
```
# Test_Results
# 1. There is a high correlation between these variables
# 2. Visually also its proved and hypothetically also its proved
# 3. The actual time and segment_actual_time both are same
```

3.0.5 5. Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

[167]:
```
# Visual Analysis
fig, axes = plt.subplots(1,3,figsize=(15,6), sharey=True)
sns.scatterplot(ax= axes[0], data=merged_data,x="osrm_distance",y="segment_osrm_distance")

sns.histplot(ax= axes[1], data=merged_data,x="osrm_distance",kde=True)

sns.histplot(ax= axes[2], data=merged_data,kde=True,x="segment_osrm_distance")

# From ploting we can find out the both osrm_distance and segment_osrm_distance are almost same
# There is high correlation between these variables
```

[167]:
```
<AxesSubplot:xlabel='segment_osrm_distance', ylabel='Count'>
```



[168]:
```
# H0: Both are not correlated
# Ha: Both are correlated

## properties of two variables ##
# 1. Both features are continuous variables
# 2. Data is right skewed in nature
```

```
# 3. There is lots of outliers are there

## Correlation Testing ##
alpha = 0.05
corr_stat, p_value = pearsonr(merged_data["osrm_distance"],␣
 ↪merged_data["segment_osrm_distance"])
if p_value<alpha:
    print("Reject Null Hypothesis")
else:
    print("Fail to reject Null Hypothesis")
print("Test Statistic Value: ",corr_stat)
print("P_value:",p_value)
```

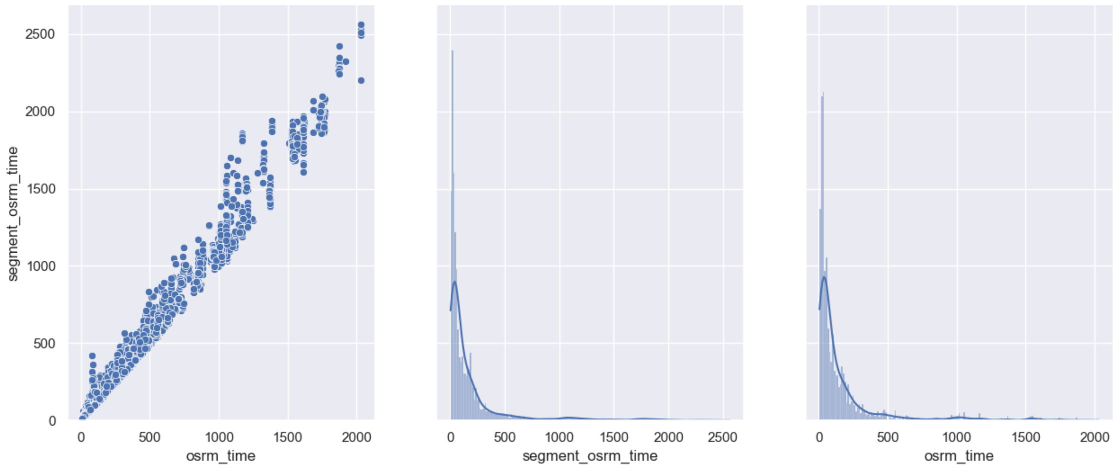
```
Reject Null Hypothesis
Test Statistic Value:  0.99496426416308
P_value: 0.0
```

[169]:
```
# Test_Results
# 1. There is a high correlation between these variables
# 2. Visually also its proved and hypothetically also its proved
# 3. The actual time and open-source routing engine timings both are same
```

3.0.6 6. Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value

[170]:
```
# Visual Analysis
fig, axes = plt.subplots(1,3,figsize=(15,6), sharey=True)
sns.scatterplot(ax= axes[0],␣
 ↪data=merged_data,x="osrm_time",y="segment_osrm_time")

sns.histplot(ax= axes[1], data=merged_data,x="segment_osrm_time",kde=True)

sns.histplot(ax= axes[2], data=merged_data,kde=True,x="osrm_time")

# From ploting we can find out the both osrm_time and segment_osrm_time are␣
 ↪almost same
# There is high correlation between these variables
```

[170]:
```
<AxesSubplot:xlabel='osrm_time', ylabel='Count'>
```



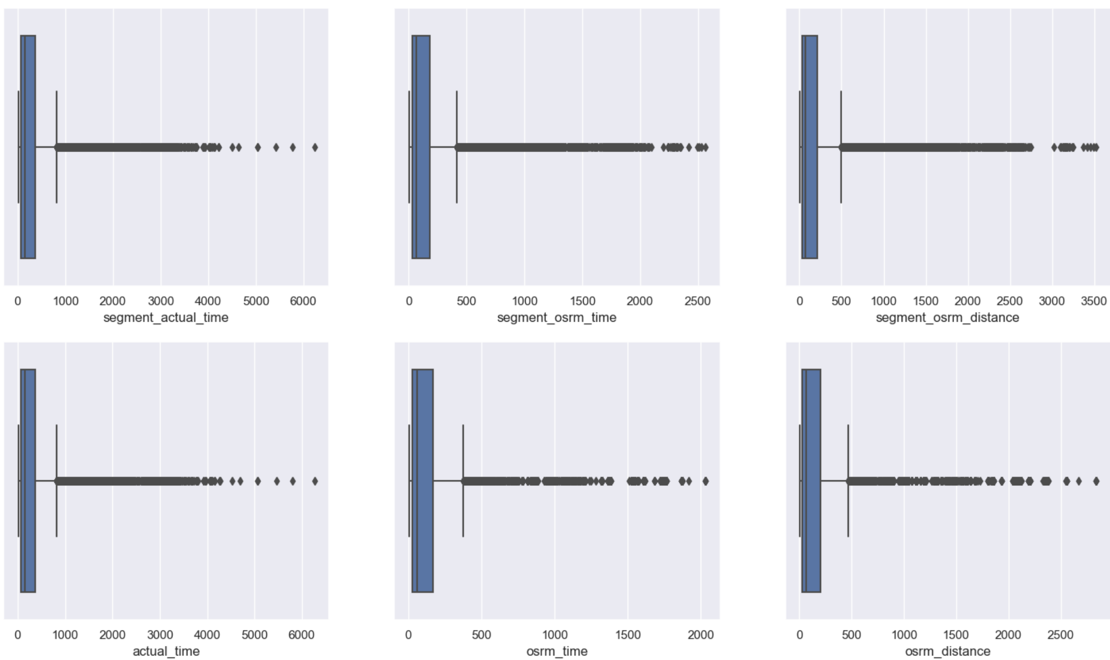
[171]:
```
# H0: Both are not correlated
# Ha: Both are correlated

## properties of two variables ##
# 1. Both features are continuous variables
# 2. Data is right skewed in nature
# 3. There is lots of outliers are there

## Correlation Testing ##
alpha = 0.05
corr_stat, p_value = pearsonr(merged_data["osrm_time"],␣
↪merged_data["segment_osrm_time"])
if p_value<alpha:
    print("Reject Null Hypothesis")
else:
    print("Fail to reject Null Hypothesis")
print("Test Statistic Value: ",corr_stat)
print("P_value:",p_value)
```

```
Reject Null Hypothesis
Test Statistic Value:  0.9935532802444722
P_value: 0.0
```

[172]:
```
# Test_Results
# 1. There is a high correlation between these variables
# 2. Visually also its proved and hypothetically also its proved
# 3. The osrm_time and segment_osrm_time both are same
```

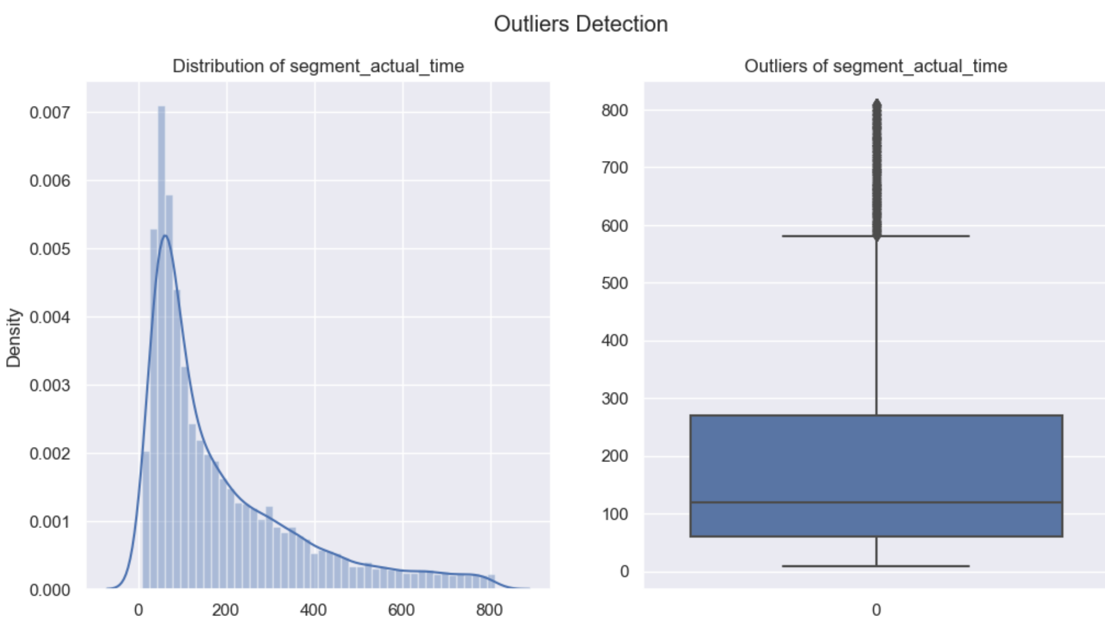
3.0.7 7. Find outliers in the numerical variables

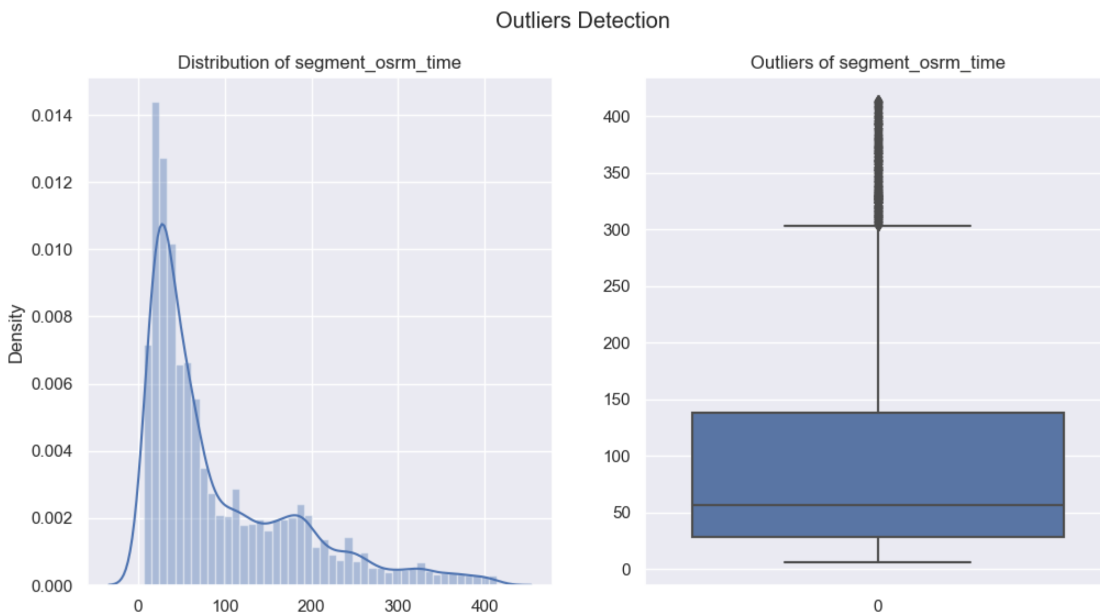
[173]:
```
numerical_variable = ["segment_actual_time"        ,"segment_osrm_time",␣
↪"segment_osrm_distance","actual_time","osrm_time","osrm_distance"]
fig, axes = plt.subplots(2,3, figsize=(18,10))
for i in range(len(numerical_variable)):
    if i < 3:
        sns.boxplot(ax = axes[0,i], data = merged_data, x =␣
↪numerical_variable[i])
    else:
        i -= 3
        sns.boxplot(ax = axes[1,i], data = merged_data, x =␣
↪numerical_variable[i+3])
```

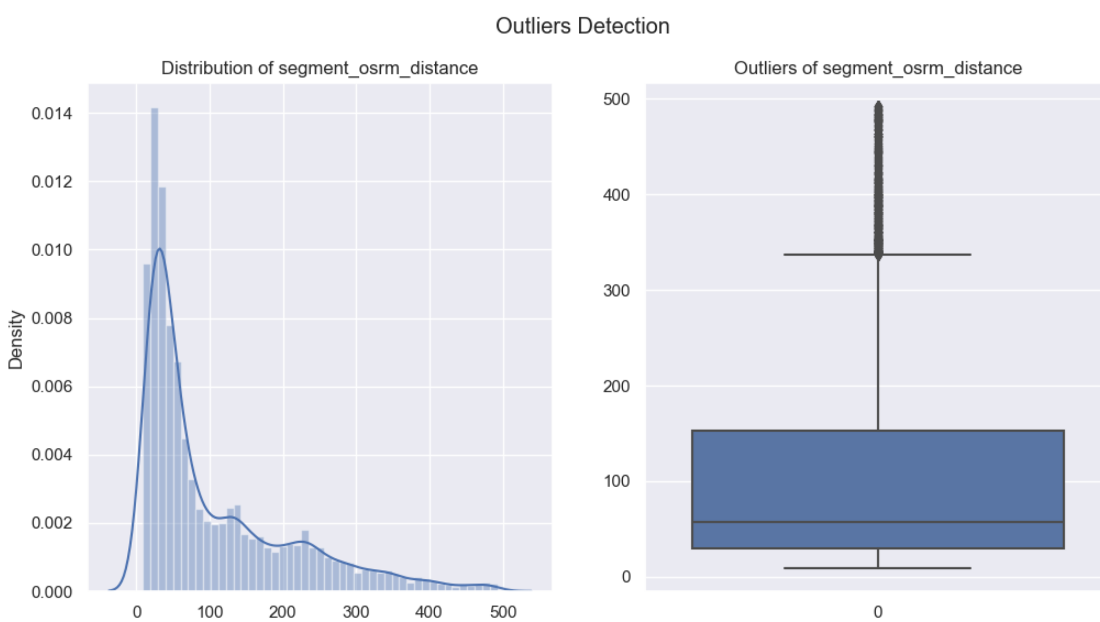


[174]:
```
numerical_variable = ["segment_actual_time"        ,"segment_osrm_time",␣
↪"segment_osrm_distance","actual_time","osrm_time","osrm_distance"]
for i in range(len(numerical_variable)):
    upper = merged_data[numerical_variable[i]].quantile(.75)
    lower = merged_data[numerical_variable[i]].quantile(.25)
    iqr = upper - lower
    upper_limit = upper + 1.5 * iqr
    lower_limit = lower - 1.5 * iqr

    #Non_outlier data
```

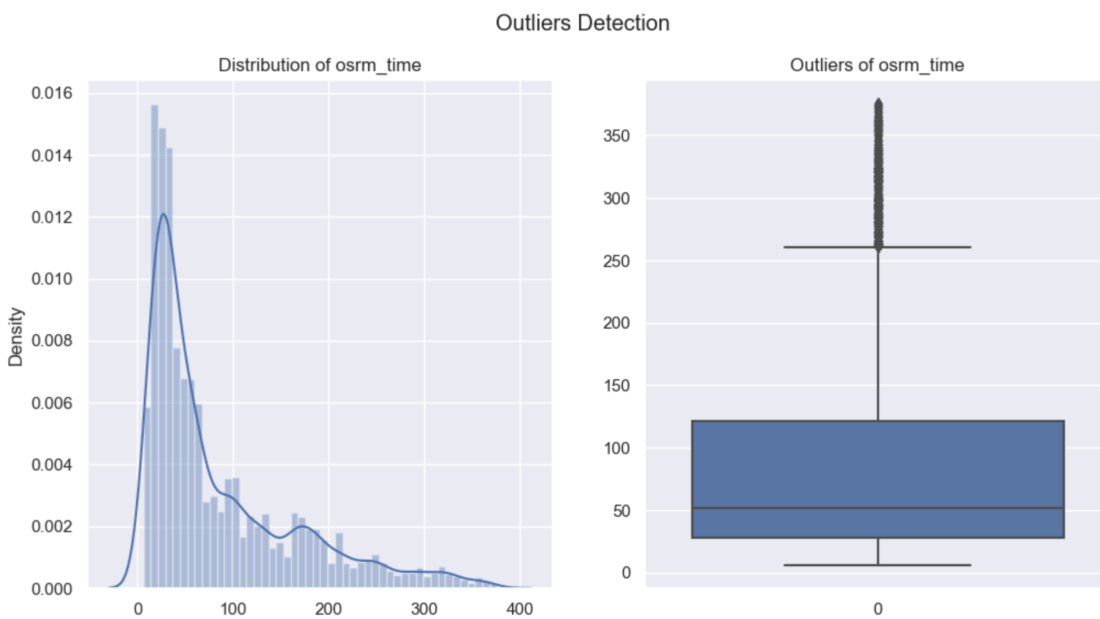
```
    non_outlier_data = np.array(merged_data[(merged_data[numerical_variable[i]] < upper_limit) & (merged_data[numerical_variable[i]] > lower_limit)][numerical_variable[i]]).reshape(1,-1)
    # Even though we have filtered outliers based on IQR range
    # But data still have outliers values, this can be ignorable

    fig, axes = plt.subplots(1,2, figsize=(12,6))
    fig.suptitle("Outliers Detection")
    sns.distplot(ax = axes[0],a=non_outlier_data)
    axes[0].set_title(f"Distribution of {numerical_variable[i]}")
    sns.boxplot(ax = axes[1],data = non_outlier_data)
    axes[1].set_title(f"Outliers of {numerical_variable[i]}")


# Inference
# Even we have filtered outliers based on IQR range
# There is some outliers present in data this can't be removed or fileterd again
```




Outliers Detection
Distribution of segment_osrm_time
Outliers of segment_osrm_time
Density
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
400
350
250
150
50


Outliers Detection
Distribution of segment_osrm_distance
Outliers of segment_osrm_distance
Density
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
400
500


Outliers Detection
Distribution of actual_time
Outliers of actual_time
Density
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0
200
400
600
800
0
100
200
300
400
500
600
700
800
0


Outliers Detection
Distribution of osrm_time
Outliers of osrm_time
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
0
100
200
300
400
0
50
100
150
200
250
300
350
0

3.0.8 8. One hot Encoding for Categorical variable

[175]:
```python
# Figuring out the categorical variable
df.nunique().reset_index()

# route_type and is_cutoff is the teo categorical variable
# Let's do OneHot encoding for those
dummies = pd.get_dummies(df.route_type)
df = pd.concat([df,dummies],axis=1)
```

3.0.9 9. Normalize/ Standardize the numerical features

[179]:
```python
numerical_variable = ["segment_actual_time"        ,"segment_osrm_time","segment_osrm_distance","actual_time","osrm_time","osrm_distance"]

# Initialize standard scaler
standard = StandardScaler()
data_fornormalize = df.copy()
data_fornormalize[numerical_variable] = standard.fit_transform(data_fornormalize[numerical_variable])
```

[181]:
```python
# Initialize standard scaler
MinMax = MinMaxScaler()
data_forMinmax = df.copy()
data_forMinmax[numerical_variable] = MinMax.fit_transform(data_forMinmax[numerical_variable])
```

```
[ ]: df.
      ↪groupby(["trip_uuid","source_name","destination_name"])["actual_distance_to_destination",␣
      ↪"actual_time"].mean().reset_index().
      ↪sort_values(by="actual_time",ascending=False).head(50)
```

4 Business Insights

```
[ ]: # Most of the orders are packed from haryana,Karnataka,maharashtra, Telangana,␣
      ↪Uttar pratesh
     # Order packed from major cities are Gurgaon, Bangalore, Bhiwandi, Pune,␣
      ↪Hyderabad
     # Less number of orders are packed in the states of eastern india and Delhi, goa
     # Orders packed in least city was Bhadra, jetpur, krishnanagar, etc.
     # Most people ordered from Haryana, Karnataka, Maharashtra, Delhi, Telangana
     # Eastern side of india people was not ordered that much in delhivery
     # Hills side area have taken more time to delivery, there are multiple␣
      ↪dependencies
     # The delhivery almost delivery all the products equal to open source time␣
      ↪calculator
     # Even there is no difference beteen delhivery distance and OSRM distance,␣
      ↪logistics are travelling in correct way and there is no scam happened
     # there are some outliers in data, which tells that delhivery delivered some␣
      ↪products in extreme condition also
```

5 Recommendations

```
[ ]: # Delhivery is faster in major cities, if they develop their business to tier-3␣
      ↪cities, it will helpfull to increase business growth
     # In some places the intermediate time taken between two cities have taking␣
      ↪more than usual timings, which delhivery should take care
     # Several condition delhivery logistics, but there rare cases where delhivery␣
      ↪makes to deliver products as soon as possible
```