+ Code + Markdown ··· Select Kernel

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Python

```python
plt.rcParams['figure.figsize'] = (31, 9)
sns.set()
```

Python

```python
df= pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181')
```

Python

Problem Statement

Delhivery uses to data to build sophisticated systems to run their day to day services and would require the data to be cleaned, sanitized and transformed to get useful features out of the raw fields. And use this transformed data to build data products and especially forcasting models to help predict relevent outcomes for the business.

Exploratory Data Analysis

- There are 24 features and 144K observations in the dataset.
- The data seems to be a mix of different data types.Most seem to be float, with a few object data types and one int64 and bool data type. We'll explore if these need to be transformed to another data type.
- Transforming 'trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_timestamp' to date time
- Transforming 'data' and 'route_type' to categorical types
- There are two columns 'source_name' and 'destination_name' have some missing values.
- Looks like values in source_name and destiname_name are missing at random and isn't tied to any specific feature in the dataset. We will explore how to treat these missing values.
-

```python
df.shape
```

Python

```
(144867, 24)
```

```python
df.head(5)
```

Python

```python
cols_to_datetime = ['trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_timestamp']
for col in cols_to_datetime:
    df[col] = pd.to_datetime(df[col])

cols_to_categorical = ['data','route_type']

for col in cols_to_categorical:
    df[col] = pd.Categorical(df[col], ordered= False)

df.info()
```

Python

```
<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  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  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
...
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), category(2), datetime64[ns](4), float64(10), int64(1), object(6)
memory usage: 23.6+ MB
```

```python
df.columns[df.columns != 'source_name']
```

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'destination_center', 'destination_name',
       'od_start_time', 'od_end_time', 'start_scan_to_end_scan', 'is_cutoff',
       'cutoff_factor', 'cutoff_timestamp', 'actual_distance_to_destination',
       'actual_time', 'osrm_time', 'osrm_distance', 'factor',
       'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance',
       'segment_factor'],
      dtype='object')
```

```python
for col in df.columns[df.columns != 'source_name']:
    print(col)
    print('Actual cardinality:', df[col].nunique())
    print(df.loc[df['source_name'].isna(), col].nunique())
    print('-' * 25)
```

```
data
Actual cardinality: 2
2
-------------------------
trip_creation_time
Actual cardinality: 14817
66
-------------------------
route_schedule_uuid
Actual cardinality: 1504
18
-------------------------
route_type
Actual cardinality: 2
2
-------------------------
trip_uuid
Actual cardinality: 14817
66
-------------------------
source_center
Actual cardinality: 1508
10
-------------------------
destination_center
...
```

```python
for col in df.columns[df.columns != 'destination_name']:
    print(col)
    print('Actual cardinality:', df[col].nunique())
    print(df.loc[df['destination_name'].isna(), col].nunique())
    print('-' * 25)
```

```
data
Actual cardinality: 2
2
-------------------------
trip_creation_time
Actual cardinality: 14817
80
-------------------------
route_schedule_uuid
Actual cardinality: 1504
22
-------------------------
route_type
Actual cardinality: 2
2
-------------------------
trip_uuid
Actual cardinality: 14817
80
-------------------------
source_center
Actual cardinality: 1508
24
-------------------------
source_name
...
segment_factor
```

Data summary

- start_scan_to_end_scan: Don't have information about the unit of measurement. Min seems to be 20 and goes till 7898 with and average of around 961.
- cutoff_factor: Don't have information on the unit of measurement. Min seems to be 9 and max seems to be 1927 with and average of 232.9
- actual_distance_to_destination: Min amount KMs delivered 9 and max seems to be 1927 and the average is 234. This seems to closely resemble cutoff_factor feature.
- osrm_time is the time that is calculated by a system to generate the time taken to deliver on the shortest path. Min seems like six and doesn't look like this time was achieved. Max seems like 1686 and the lower than the actual time. Most likely delhivery didn't achieve this target. The average time on osrm seem to be 232 and actual time is around 416. Seems like Delhivery doesn't achieve their target most of the time.
- osrm_distance: is the shortest distance generated by the osrm system and the average is higher than the actual distance and still reaches faster than the actual time.
- segment_actual_time: lowest seems to be -244 and the max seems to be 3051 and the average is 36. should there be a negative value in the time feature.
- segment_osrm_time: like the osrm feilds this seems lower than the actual time.
- segment_osrm_distance: There doesn't seem to be a corresponding actual segment distanced. Min: 0 Max: 2191 and the average seem to be 22

```python
df.describe()
```

Python

Data summary for object data types

- Looks like most object features are just ID columns or features with high cardinality
- Looks like date time fields don't have any particular pattern.

```python
df.describe(include= 'object')
```

Python

...

```python
df.describe(include= 'datetime', datetime_is_numeric= True)
```

Python

...

Exploratory Data Analysis

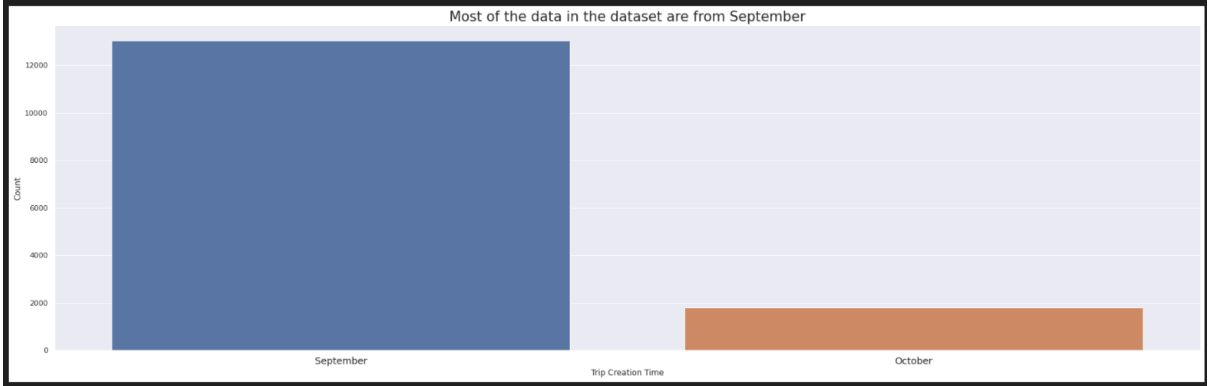
- Most of the data are for September and few in October.
- Most trips are created on a Wednesday and least on a Sunday.
- There are a total of 14,817 unique trip UUIDs and total of 1504 route Ids.
- 448 routes make up ~80% of all the deliveries undertaken by Delhivery.
- There are 1508 unique source centers and about 274 (17.5%) make up ~80% of trip creations for Delhivery.
- There are 1508 unique source center Ids and 1498 unique source centers. Are Ids duplicated?
- There' doesn't seem to be any duplicates. All 10 Ids are assigned to missing source names.
- We see similar trends for destination centers. Almost ~17.5% receive 80% of deliveries.
- We see similar trends for start and end time for trips in Delhivery. Infact, we see similar trends for route type. Most trips start and end in the earlier hours of the day or the late of the day.
- Time between start and end scan seem to follow an exponential distribution.
- Distance between source and destination center seem to follow exponential distribution.
- Segment actual and osrm times seem to follow Exponential Distribution as well.

```python
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  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  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
...
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), category(2), datetime64[ns](4), float64(10), int64(1), object(6)
memory usage: 23.6+ MB
```

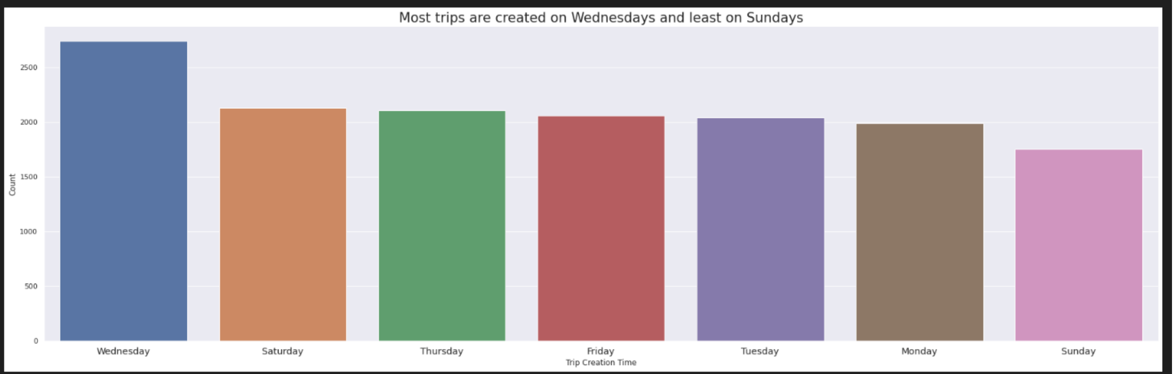
```python
temp= df.groupby('trip_uuid')['trip_creation_time'].first()
ax= sns.countplot(x= temp.dt.month_name())
ax.set_title(r'Most of the data in the dataset are from September', fontsize= 'xx-large')
ax.set_xlabel('Trip Creation Time')
ax.set_xticklabels(fontsize= 'large', labels= ['September', 'October'])
ax.set_ylabel('Count')
plt.show()
```

Python




```python
temp= df.groupby('trip_uuid')['trip_creation_time'].first()
ax= sns.countplot(x= temp.dt.day_name(), order= temp.dt.day_name().value_counts().index)
ax.set_title('Most trips are created on Wednesdays and least on Sundays', fontsize= 'xx-large')
ax.set_xlabel('Trip Creation Time')
ax.set_xticklabels(fontsize= 'large', labels= temp.dt.day_name().value_counts().index)
ax.set_ylabel('Count')
plt.show()
```

Python




```python
print(df['route_schedule_uuid'].nunique())
temp= df['route_schedule_uuid'].value_counts(normalize= True)
temp[temp.cumsum() < 0.8].shape
```

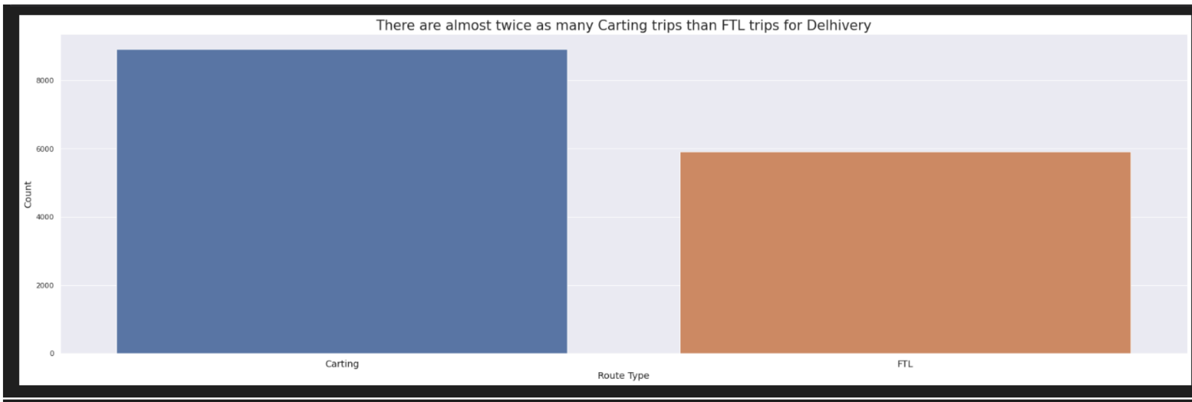
Python

```
1504
```

```
(488,)
```

```python
temp= df.groupby('route_type')['trip_uuid'].nunique().reset_index()
ax= sns.barplot(data= temp, x= 'route_type', y= 'trip_uuid', order= temp['route_type'].unique())
ax.set_title('There are almost twice as many Carting trips than FTL trips for Delhivery', fontsize= 'xx-large')
ax.set_xlabel('Route Type', fontsize= 'large')
ax.set_ylabel('Count', fontsize= 'large')
ax.set_xticklabels(labels= temp['route_type'].unique(), fontsize= 'large')
plt.show()
```

Python




```
print(df['source_center'].nunique())
print(df['source_name'].nunique())
temp= df['source_center'].value_counts(normalize= True)
temp[temp.cumsum() < 0.8].shape[0]
```

```
1508
1498

274
```

```
df.loc[df['source_name'].isna(), 'source_center'].nunique()
```

```
10
```

```
print(df['destination_center'].nunique())
print(df['destination_name'].nunique())
temp= df['destination_center'].value_counts(normalize= True)
temp[temp.cumsum() < 0.8].shape[0]
```

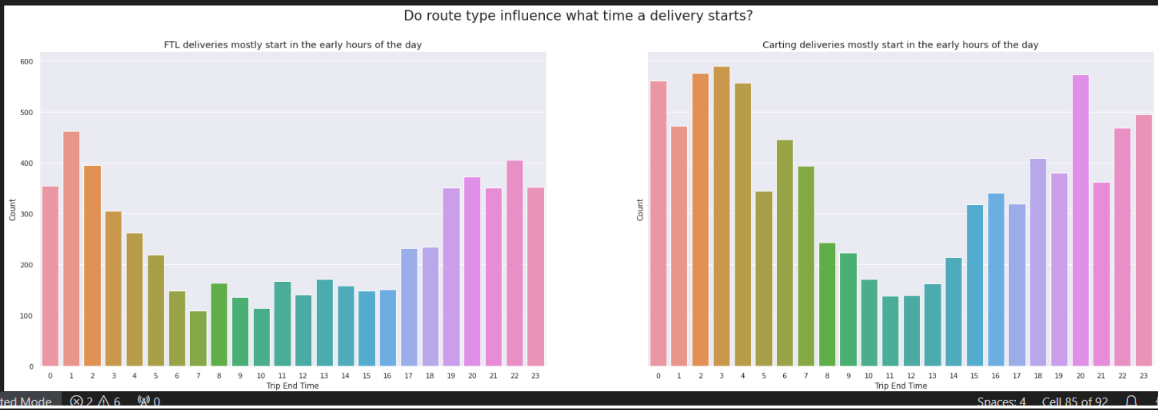
```
1481
1468

257
```

```
df.loc[df['destination_name'].isna(), 'destination_center'].nunique()
```

```
... 13
```

```python
temp= df.groupby('trip_uuid').first()
fig, (ax1, ax2) = plt.subplots(1, 2, sharey= True)
sns.countplot(x= temp.loc[temp['route_type'] == 'FTL', 'od_start_time'].dt.hour, ax= ax1)
sns.countplot(x= temp.loc[temp['route_type'] == 'Carting', 'od_start_time'].dt.hour, ax= ax2)
plt.suptitle('Do route type influence what time a delivery starts?', fontsize= 'xx-large')
ax1.set_title('FTL deliveries mostly start in the early hours of the day', fontsize= 'large')
ax2.set_title('Carting deliveries mostly start in the early hours of the day', fontsize= 'large')
ax1.set_xlabel('Trip End Time')
ax2.set_xlabel('Trip End Time')
ax1.set_ylabel('Count')
ax2.set_ylabel('Count')
plt.show()
```

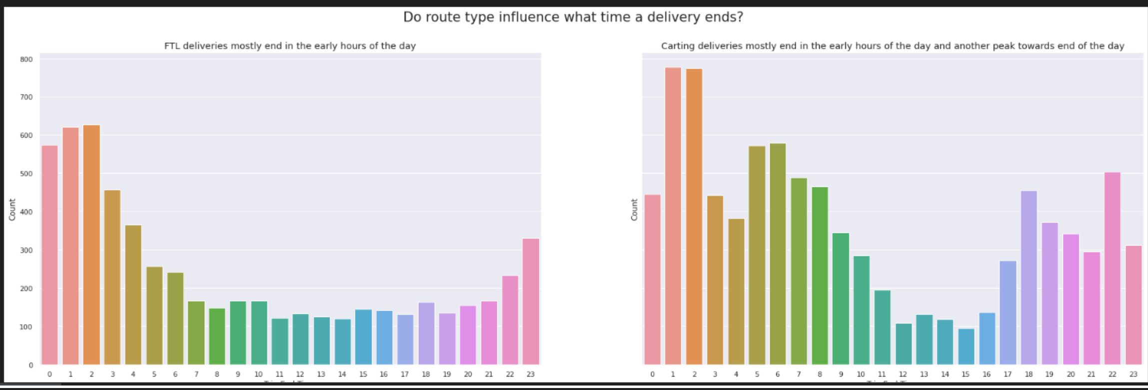
Python



Restricted Mode ⊗ 2 ⚠ 6 Spaces: 4 Cell 85 of 92

```python
temp= df.groupby('trip_uuid').first()
fig, (ax1, ax2) = plt.subplots(1, 2, sharey= True)
sns.countplot(x= temp.loc[temp['route_type'] == 'FTL', 'od_end_time'].dt.hour, ax= ax1)
sns.countplot(x= temp.loc[temp['route_type'] == 'Carting', 'od_end_time'].dt.hour, ax= ax2)
plt.suptitle('Do route type influence what time a delivery ends?', fontsize= 'xx-large')
ax1.set_title('FTL deliveries mostly end in the early hours of the day', fontsize= 'large')
ax2.set_title('Carting deliveries mostly end in the early hours of the day and another peak towards end of the day', fontsize= 'large')
ax1.set_xlabel('Trip End Time')
ax2.set_xlabel('Trip End Time')
ax1.set_ylabel('Count')
ax2.set_ylabel('Count')
plt.show()
```

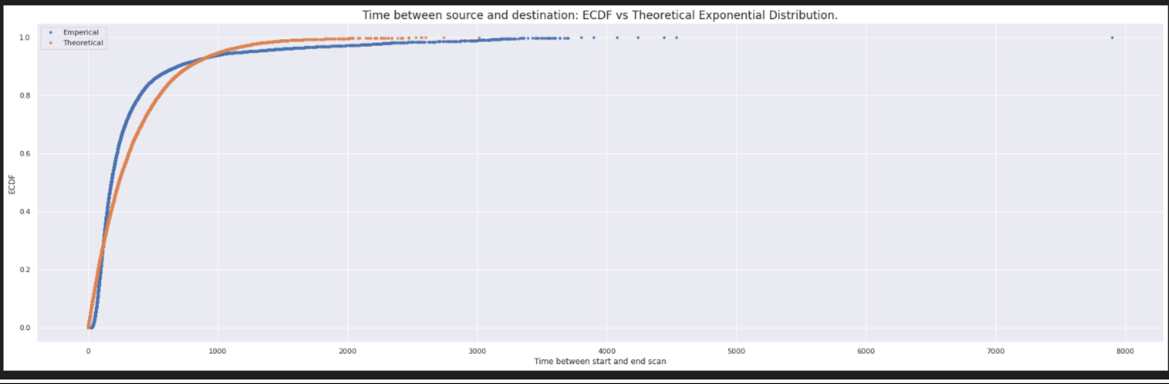
Python




```python
def ecdf(data):
    n = len(data)
    x = np.sort(data)
    y = np.arange(1, n+1) / n
    return x, y
```

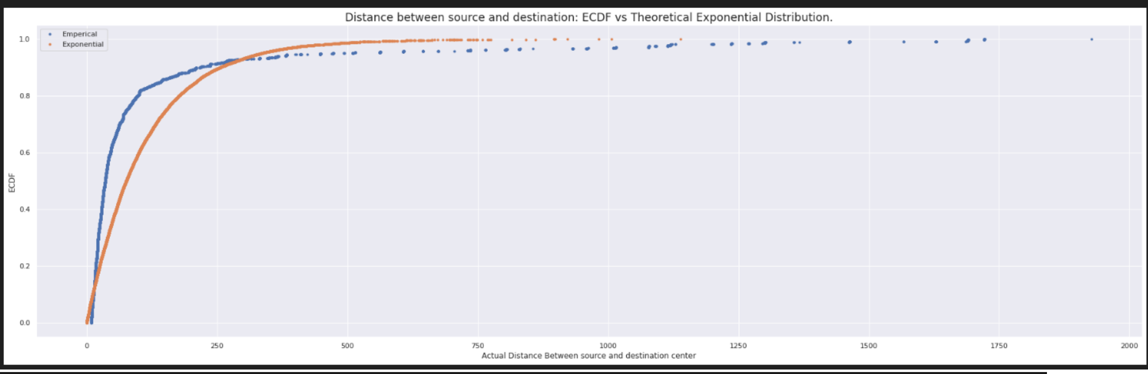
```python
temp = df.groupby('trip_uuid').first()['start_scan_to_end_scan']
temp_x, temp_y= ecdf(temp)
theo_x, theo_y= ecdf(np.random.exponential(temp.mean(), len(temp_x)))

plt.plot(temp_x, temp_y, marker= '.', linestyle= 'none')
plt.plot(theo_x, theo_y, marker= '.', linestyle= 'none')
plt.xlabel('Time between start and end scan')
plt.ylabel('ECDF')
plt.title('Time between source and destination: ECDF vs Theoretical Exponential Distribution.', fontsize= 'x-large')
plt.legend(['Emperical', 'Theoretical'])
plt.show()
```

Python



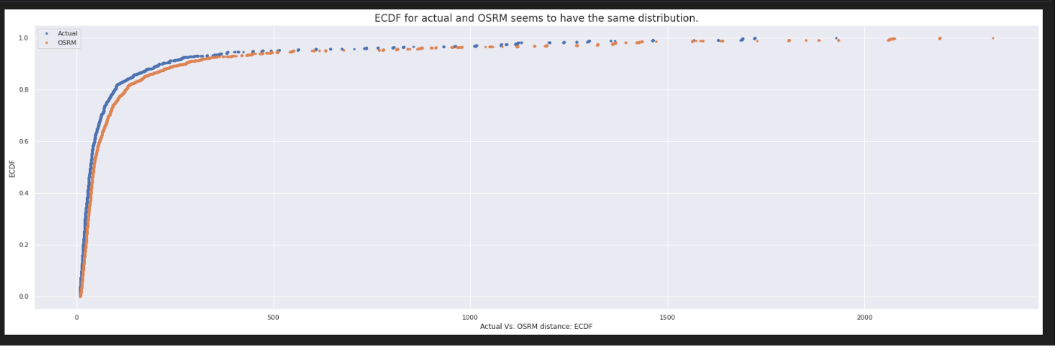

```python
temp= df.groupby('trip_uuid')['actual_distance_to_destination'].last()
emp_x, emp_y= ecdf(temp)
exp_x, exp_y= ecdf(np.random.exponential(temp.mean(), len(emp_x)))

plt.xlabel('Actual Distance Between source and destination center')
plt.ylabel('ECDF')
plt.plot(emp_x, emp_y, marker= '.', linestyle = 'none', label= 'Emperical')
plt.plot(exp_x, exp_y, marker= '.', linestyle = 'none', label= 'Exponential')
plt.title('Distance between source and destination: ECDF vs Theoretical Exponential Distribution.', fontsize= 'x-large')
plt.legend()
plt.show()
```

Python



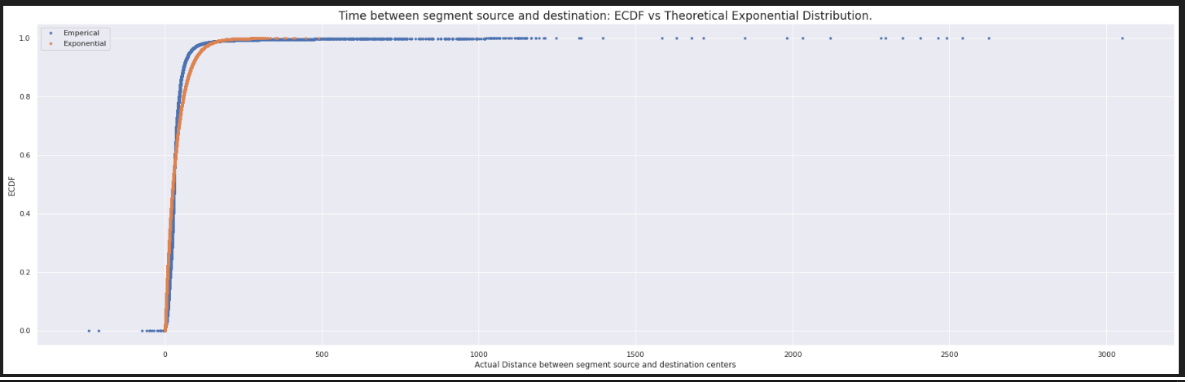

```python
actual= df.groupby('trip_uuid')['actual_distance_to_destination'].last()
osrm= df.groupby('trip_uuid')['osrm_distance'].last()
actual_x, actual_y= ecdf(actual)
osrm_x, osrm_y= ecdf(osrm)

plt.xlabel('Actual Vs. OSRM distance: ECDF')
plt.ylabel('ECDF')
plt.plot(actual_x, actual_y, marker= '.', linestyle = 'none', label= 'Actual')
plt.plot(osrm_x, osrm_y, marker= '.', linestyle = 'none', label= 'OSRM')
plt.title('ECDF for actual and OSRM seems to have the same distribution.', fontsize= 'x-large')
plt.legend()
plt.show()
```

Python



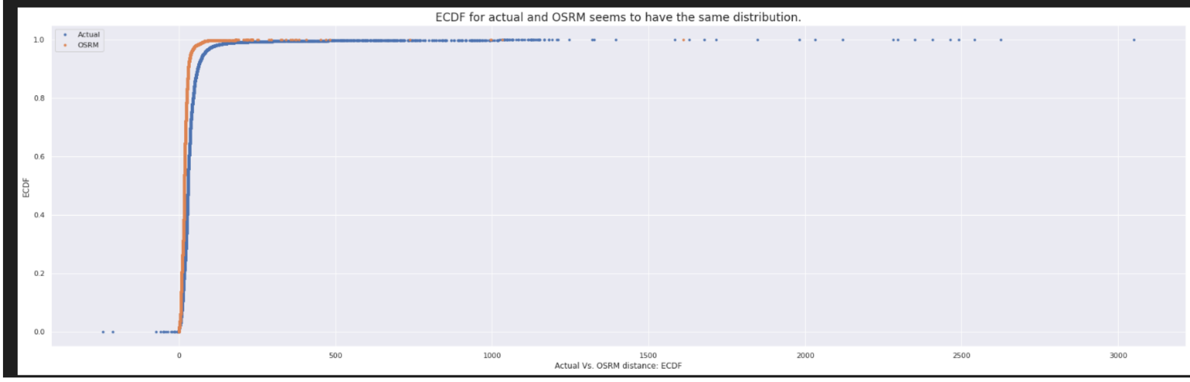
```python
temp= df['segment_actual_time']
emp_x, emp_y= ecdf(temp)
exp_x, exp_y= ecdf(np.random.exponential(temp.mean(), len(emp_x)))

plt.xlabel('Actual Distance between segment source and destination centers')
plt.ylabel('ECDF')
plt.plot(emp_x, emp_y, marker= '.', linestyle = 'none', label= 'Emperical')
plt.plot(exp_x, exp_y, marker= '.', linestyle = 'none', label= 'Exponential')
plt.title('Time between segment source and destination: ECDF vs Theoretical Exponential Distribution.', fontsize= 'x-large')
plt.legend()
plt.show()
```

Python




```python
actual= df['segment_actual_time']
osrm= df['segment_osrm_time']
actual_x, actual_y= ecdf(actual)
osrm_x, osrm_y= ecdf(osrm)

plt.xlabel('Actual Vs. OSRM distance: ECDF')
plt.ylabel('ECDF')
plt.plot(actual_x, actual_y, marker= '.', linestyle = 'none', label= 'Actual')
plt.plot(osrm_x, osrm_y, marker= '.', linestyle = 'none', label= 'OSRM')
plt.title('ECDF for actual and OSRM seems to have the same distribution.', fontsize= 'x-large')
plt.legend()
plt.show()
```

Python



Merging the rows to make the dataset easier to work with.

- We will merge by the trip_uuid, source_center and destination_center. This seems like the unique identifier for this dataset.
- We will get the first or the last value for most fields depending on their nature like start time or end time.
- We will capture sum of fields like segment actual and osrm time

```python
group= ['trip_uuid', 'source_center', 'destination_center']
```

```python
aggregations = {
    'data': 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'source_center': 'first',
    'source_name': 'first',
    'destination_center': 'first',
    'destination_name': 'first',
    'od_start_time': 'min',
    'od_end_time': 'max',
    'start_scan_to_end_scan': 'last',
    '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'
}
```

```python
cleaned= df.groupby(group, as_index= False).agg(aggregations)
```

```python
cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26368 entries, 0 to 26367
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       26368 non-null  object
 1   data                            26368 non-null  category
 2   trip_creation_time              26368 non-null  datetime64[ns]
 3   route_schedule_uuid             26368 non-null  object
 4   route_type                      26368 non-null  category
 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
dtypes: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 3.5+ MB
```

Feature Engineering

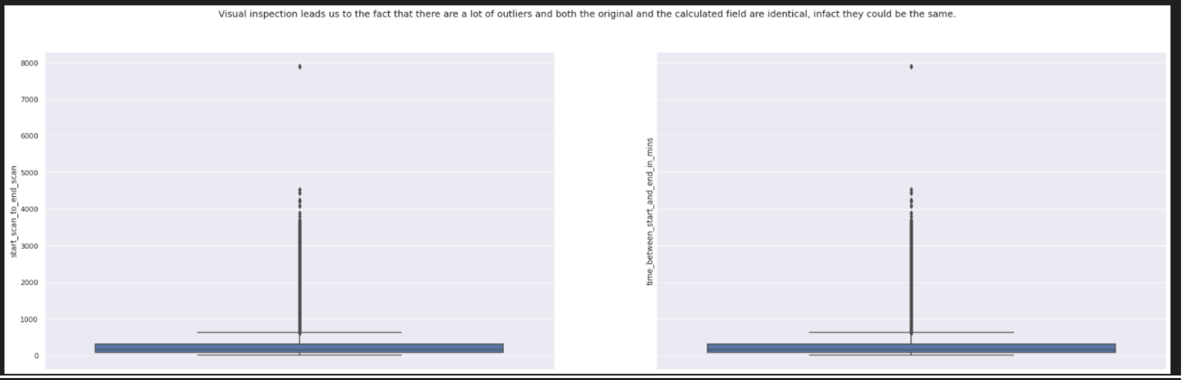
- Used regex to extract the first part from destination and source name and state. Used split to extract city and place from first part and added columns city, place and state for both source and destination name
- Year, month, day and hour of trip creation was extracted and added to the cleaned data set.
- Calculating time between end and start od time in minutes gives us the same result as start_scan_to_end_scan.

```python
cleaned[['part1', 'destination_state']]= cleaned['destination_name'].str.extract(r'(.*_.*_.*)\s\((.*)\)')
cleaned[['destination_city', 'destination_place'] ] = cleaned['part1'].str.split('_', expand= True).loc[:, [0, 1]]
cleaned.drop(['part1', 'destination_name'], axis= 1, inplace= True)

cleaned[['part1', 'source_state']]= cleaned['source_name'].str.extract(r'(.*_.*_.*)\s\((.*)\)')
cleaned[['source_city', 'source_place'] ] = cleaned['part1'].str.split('_', expand= True).loc[:, [0, 1]]
cleaned.drop(['part1', 'source_name'], axis= 1, inplace= True)

cleaned['trip_creation_year'] = cleaned['trip_creation_time'].dt.year
cleaned['trip_creation_month'] = cleaned['trip_creation_time'].dt.month
cleaned['trip_creation_day'] = cleaned['trip_creation_time'].dt.day
cleaned['trip_creation_hour'] = cleaned['trip_creation_time'].dt.hour

cleaned['time_between_start_and_end_in_mins']= ((cleaned['od_end_time'] - cleaned['od_start_time']) / np.timedelta64(1, 'm'))

cleaned.head()
```

Python

```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned['start_scan_to_end_scan'], ax= ax1)
sns.boxplot(y= cleaned['time_between_start_and_end_in_mins'], ax= ax2)
plt.suptitle('Visual inspection leads us to the fact that there are a lot of outliers and both the original and the calculated field are identical,

plt.show()
```

Python



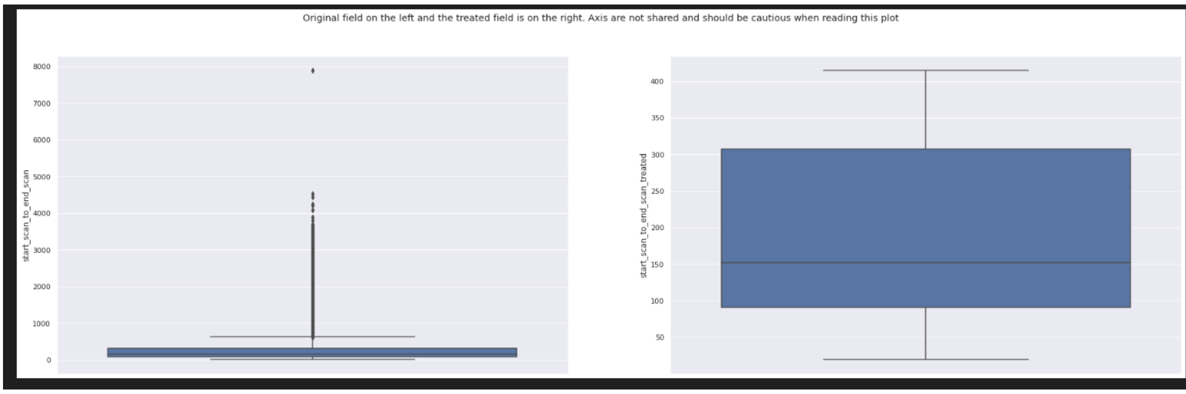
Treating outliers

```python
q1= cleaned['start_scan_to_end_scan'].quantile(.25)
q3= cleaned['start_scan_to_end_scan'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

cleaned['start_scan_to_end_scan_treated']= cleaned['start_scan_to_end_scan'].clip(lower, upper)
```

Python

```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= False)
sns.boxplot(y= cleaned['start_scan_to_end_scan'], ax= ax1)
sns.boxplot(y= cleaned['start_scan_to_end_scan_treated'], ax= ax2)
plt.suptitle('Original field on the left and the treated field is on the right. Axis are not shared and should be cautious when reading this plot')

plt.show()
```

Python

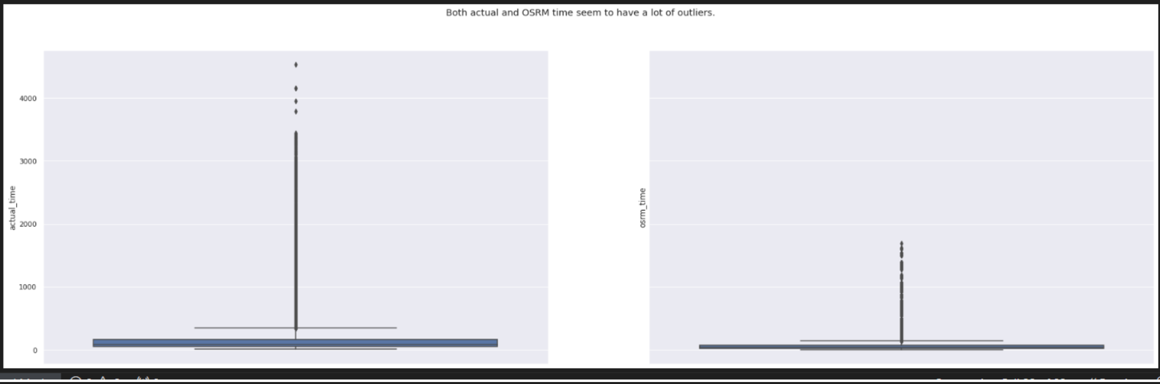


Hypothesis testing / visual analysis between actual time aggregated value and OSRM aggregated value.

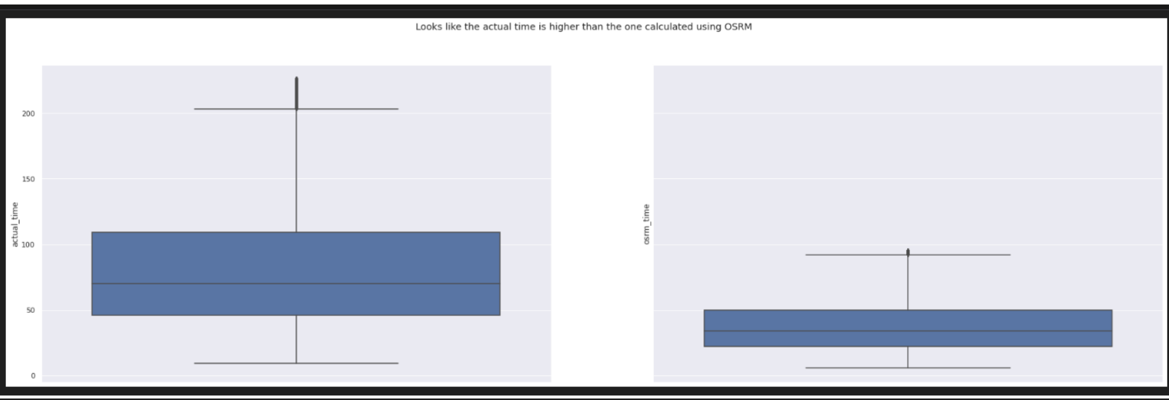
As the pvalue is less than 0.05 we reject the null hypothesis that actual time is equal to OSRM time. We can see this visually as well.

```
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned['actual_time'], ax= ax1)
sns.boxplot(y= cleaned['osrm_time'], ax= ax2)
plt.suptitle('Both actual and OSRM time seem to have a lot of outliers.')

plt.show()
```

Python

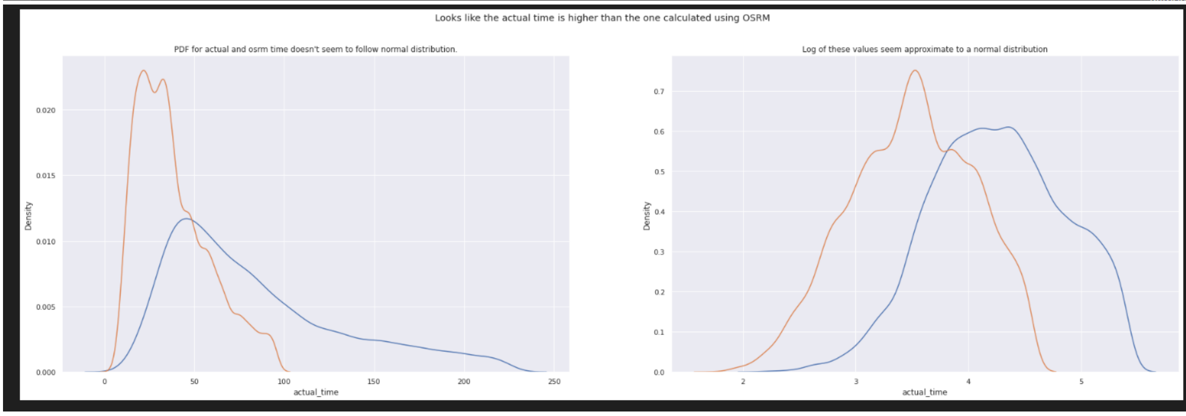



```
# Treating outliers in actual time.
q1= cleaned['actual_time'].quantile(.25)
q3= cleaned['actual_time'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

actual_idx= (cleaned['actual_time'] > lower) & (cleaned['actual_time'] < upper)

# Treating outliers in OSRM time.
q1= cleaned['osrm_time'].quantile(.25)
q3= cleaned['osrm_time'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

osrm_idx= (cleaned['osrm_time'] < upper)

# Plotting the treated fields.

fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned.loc[actual_idx, 'actual_time'], ax= ax1)
sns.boxplot(y= cleaned.loc[osrm_idx, 'osrm_time'], ax= ax2)
plt.suptitle('Looks like the actual time is higher than the one calculated using OSRM')

plt.show()
```



Based on the visual analysis alone we can figure out that OSRM time is lower than the actual time.

```
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= False)
sns.kdeplot(x= cleaned.loc[actual_idx, 'actual_time'], ax= ax1)
sns.kdeplot(x= cleaned.loc[osrm_idx, 'osrm_time'], ax= ax1)
ax1.set_title('PDF for actual and osrm time doesn\'t seem to follow normal distribution.')

sns.kdeplot(x= np.log(cleaned.loc[actual_idx, 'actual_time']), ax= ax2)
sns.kdeplot(x= np.log(cleaned.loc[osrm_idx, 'osrm_time']), ax= ax2)
ax2.set_title('Log of these values seem approximate to a normal distribution')

plt.suptitle('Looks like the actual time is higher than the one calculated using OSRM')

plt.show()
```




```
print('average actual time:', cleaned.loc[actual_idx, 'actual_time'].mean())
print('standard deviation:', cleaned.loc[actual_idx, 'actual_time'].std())
print('average osrm time:', cleaned.loc[osrm_idx, 'osrm_time'].mean())
print('standard deviation:', cleaned.loc[osrm_idx, 'osrm_time'].std())
```

```
average actual time: 83.58569513153637
standard deviation: 48.70340878883891
average osrm time: 38.14671546372649
standard deviation: 20.232805741507132
```

```
from scipy.stats import ttest_ind, ttest_ind_from_stats
```

```python
# Null hypothesis: actual time and osrm time are actually the same.
# Alternate hypothesis: OSRM time is actually lower than actual time.

# test-statistic = average time

# one-tailed T-test to check if osrm is lower than actual time.

stat, pvalue= ttest_ind(cleaned['actual_time'], cleaned['osrm_time'], equal_var= False, alternative= 'greater')

print(stat)
print(pvalue)

if pvalue < 0.05:
    print('we reject the null hypothesis.')
else:
    print('we fail to reject the null hypothesis.')
```

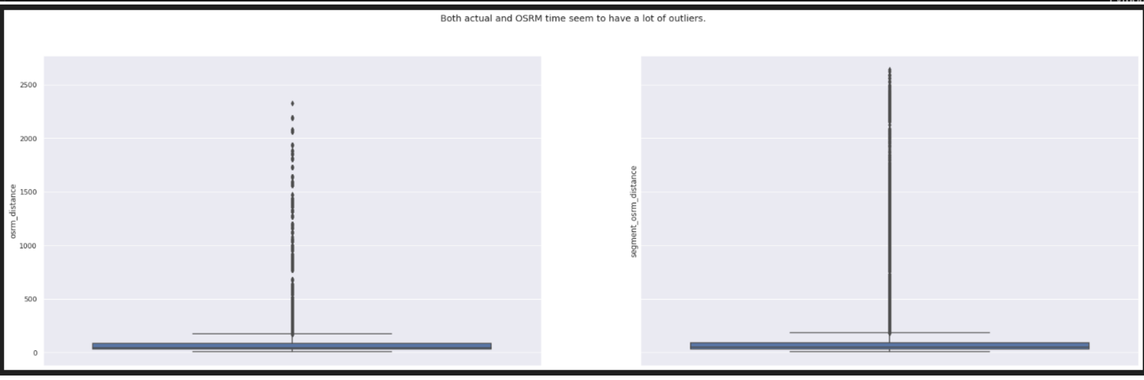
```
41.82845508363711
0.0
we reject the null hypothesis.
```

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

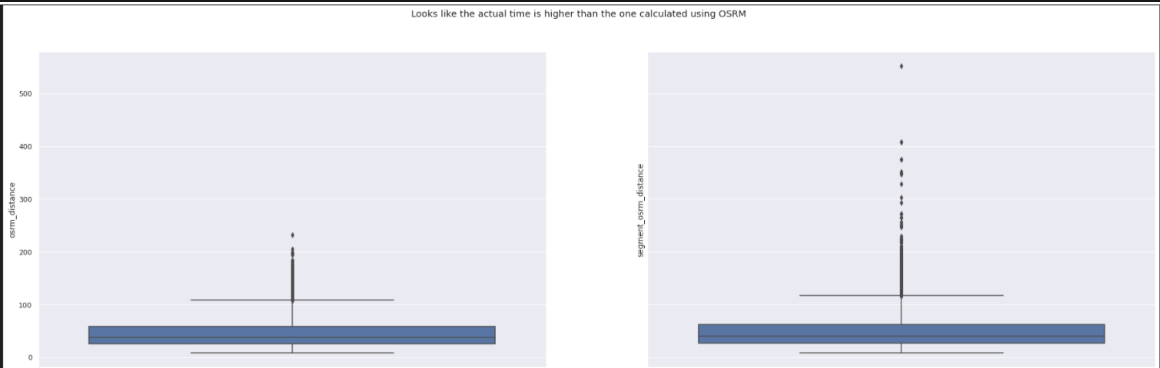
- Alhtough visually they look similar we reject the null hypothesis that the osrm_distance and segement_osrm_distance is the same.

```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned['osrm_distance'], ax= ax1)
sns.boxplot(y= cleaned['segment_osrm_distance'], ax= ax2)
plt.suptitle('Both actual and OSRM time seem to have a lot of outliers.')

plt.show()
```

Python

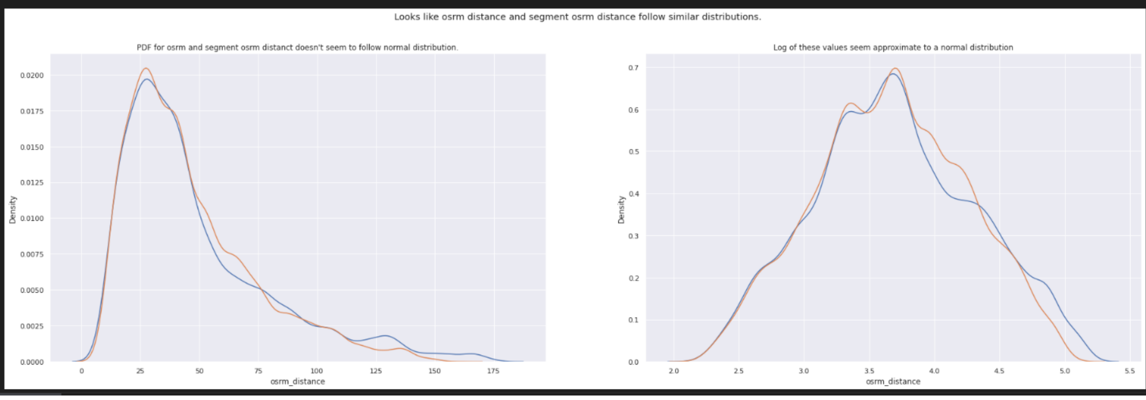


```python
# Treating outliers in actual time.
q1= cleaned['osrm_distance'].quantile(.25)
q3= cleaned['osrm_distance'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

osrm_idx= (cleaned['osrm_distance'] > lower) & (cleaned['osrm_distance'] < upper)

# Treating outliers in OSRM time.
q1= cleaned['segment_osrm_distance'].quantile(.25)
q3= cleaned['segment_osrm_distance'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

segment_osrm_idx= (cleaned['segment_osrm_distance'] > lower) & (cleaned['segment_osrm_distance'] < upper)

# Plotting the treated fields.

fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned.loc[osrm_idx, 'osrm_distance'], ax= ax1)
sns.boxplot(y= cleaned.loc[segment_osrm_idx, 'segment_osrm_distance'], ax= ax2)
plt.suptitle('Looks like the actual time is higher than the one calculated using OSRM')

plt.show()
```




```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= False)
sns.kdeplot(x= cleaned.loc[osrm_idx, 'osrm_distance'], ax= ax1)
sns.kdeplot(x= cleaned.loc[segment_osrm_idx, 'segment_osrm_distance'], ax= ax1)
ax1.set_title('PDF for osrm and segment osrm distanct doesn\'t seem to follow normal distribution.')

sns.kdeplot(x= np.log(cleaned.loc[osrm_idx, 'osrm_distance']), ax= ax2)
sns.kdeplot(x= np.log(cleaned.loc[segment_osrm_idx, 'segment_osrm_distance']), ax= ax2)
ax2.set_title('Log of these values seem approximate to a normal distribution')

plt.suptitle('Looks like osrm distance and segment osrm distance follow similar distributions.')

plt.show()
```

Python



```python
print('average actual time:', cleaned.loc[osrm_idx, 'osrm_distance'].mean())
print('standard deviation:', cleaned.loc[osrm_idx, 'osrm_distance'].std())
print('average osrm time:', cleaned.loc[segment_osrm_idx, 'segment_osrm_distance'].mean())
print('standard deviation:', cleaned.loc[segment_osrm_idx, 'segment_osrm_distance'].std())
```

```
average actual time: 43.536091128397466
standard deviation: 24.123013295033985
average osrm time: 45.40590915407574
standard deviation: 25.67995768485745
```

```python
actual_sample_size= cleaned.loc[osrm_idx, 'osrm_distance'].shape[0]
segment_actual_sample_size= cleaned.loc[segment_osrm_idx, 'segment_osrm_distance'].shape[0]
print(actual_sample_size, segment_actual_sample_size)
```

```
21597 21444
```

```python
# Null hypothesis: osrm distance and segment osrm distance are actually the same.
# Alternate hypothesis: osrm distance and segment osrm distance time are different.

# test-statistic = average time

# H0: avg. osrm distance = avg. segement osrm distance
# H1: avg. osrm distance != avg. segement osrm distance

# two-tailed T-test to check if actual and segment actual time are actually different.

stat, pvalue= ttest_ind(cleaned.loc[osrm_idx, 'osrm_distance'].sample(segment_actual_sample_size), cleaned.loc[segment_osrm_idx, 'segment_osrm_dista

print(stat)
print(pvalue)

if pvalue < 0.05:
    print('we reject the null hypothesis.')
else:
    print('we fail to reject the null hypothesis.')
```

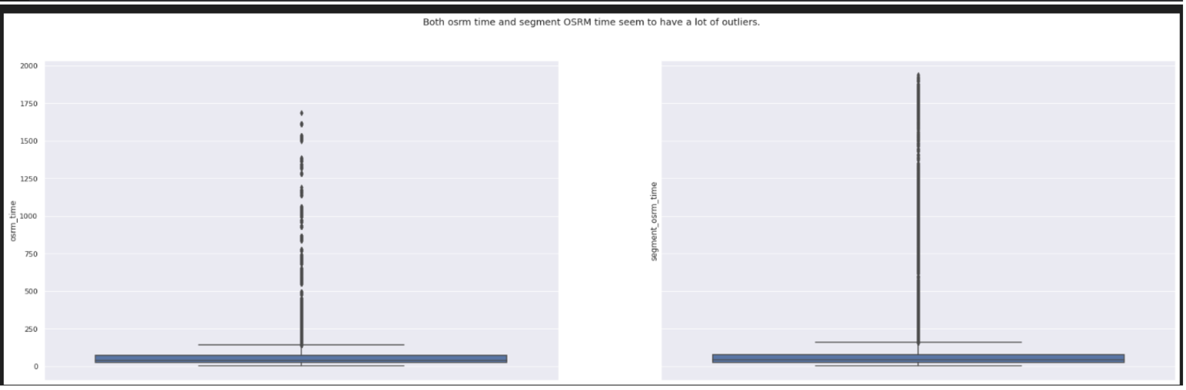
Python

```
-7.7498615495659084
9.401910584760783e-15
we reject the null hypothesis.
```

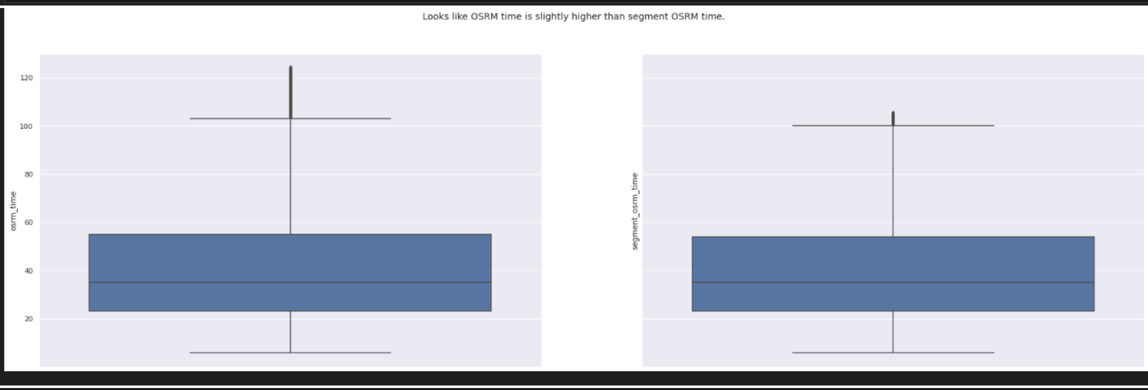
Hypothesis testing / visual analysis between osrm time aggregated value and segment osrm time aggregated value

- We saw that visually the two distribution seems to be somewhat different.
- We also reject the null hypothesis because the p value is less than 0.05.

```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned['osrm_time'], ax= ax1)
sns.boxplot(y= cleaned['segment_osrm_time'], ax= ax2)
plt.suptitle('Both osrm time and segment OSRM time seem to have a lot of outliers.')

plt.show()
```

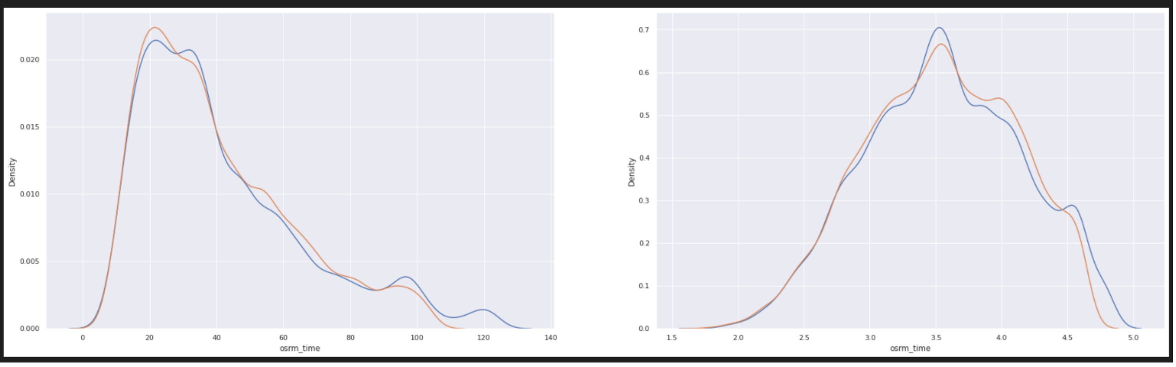


```python
# Treating outliers in actual time.
q1= cleaned['osrm_time'].quantile(.25)
q3= cleaned['segment_osrm_distance'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

osrm_idx= (cleaned['osrm_time'] > lower) & (cleaned['osrm_time'] < upper)

# Treating outliers in OSRM time.
q1= cleaned['segment_osrm_time'].quantile(.25)
q3= cleaned['segment_osrm_time'].quantile(.75)

iqr= q3 - q1

lower= q1 - 1.5 * iqr
upper= q1 + 1.5 * iqr

segment_osrm_idx= (cleaned['segment_osrm_time'] > lower) & (cleaned['segment_osrm_time'] < upper)

# Plotting the treated fields.

fig, (ax1, ax2)= plt.subplots(1, 2, sharey= True)
sns.boxplot(y= cleaned.loc[osrm_idx, 'osrm_time'], ax= ax1)
sns.boxplot(y= cleaned.loc[segment_osrm_idx, 'segment_osrm_time'], ax= ax2)
plt.suptitle('Looks like OSRM time is slightly higher than segment OSRM time.')

plt.show()
```




```python
fig, (ax1, ax2)= plt.subplots(1, 2, sharey= False)
sns.kdeplot(x= cleaned.loc[osrm_idx, 'osrm_time'], ax= ax1)
sns.kdeplot(x= cleaned.loc[segment_osrm_idx, 'segment_osrm_time'], ax= ax1)
# ax1.set_title('PDF for osrm and segment osrm distanct doesn\'t seem to follow normal distribution.')

sns.kdeplot(x= np.log(cleaned.loc[osrm_idx, 'osrm_time']), ax= ax2)
sns.kdeplot(x= np.log(cleaned.loc[segment_osrm_idx, 'segment_osrm_time']), ax= ax2)
ax2.set_title('Log of these values seem approximate to a normal distribution')

plt.suptitle('Looks like osrm time and segment osrm time follow similar distributions.')

plt.show()
```

Python



```python
print('average actual time:', cleaned.loc[osrm_idx, 'osrm_time'].mean())
print('standard deviation:', cleaned.loc[osrm_idx, 'osrm_time'].std())
print('average osrm time:', cleaned.loc[segment_osrm_idx, 'segment_osrm_time'].mean())
print('standard deviation:', cleaned.loc[segment_osrm_idx, 'segment_osrm_time'].std())
```

```
average actual time: 41.855978975032855
standard deviation: 25.112815198253877
average osrm time: 40.156046208310045
standard deviation: 22.39847489675932
```

```python
actual_sample_size= cleaned.loc[osrm_idx, 'osrm_time'].shape[0]
segment_actual_sample_size= cleaned.loc[segment_osrm_idx, 'segment_osrm_time'].shape[0]
print(actual_sample_size, segment_actual_sample_size)
```

```
22830 21468
```

```python
# Null hypothesis: osrm time and segment osrm time are actually the same.
# Alternate hypothesis: osrm time and segment osrm time are different.

# test-statistic = average time

# H0: avg. osrm time = avg. segement osrm time
# H1: avg. osrm time != avg. segement osrm time

# two-tailed T-test to check if actual and segment actual time are actually different.

stat, pvalue= ttest_ind(cleaned.loc[osrm_idx, 'osrm_time'].sample(segment_actual_sample_size), cleaned.loc[segment_osrm_idx, 'segment_osrm_time'], e

print(stat)
print(pvalue)

if pvalue < 0.05:
  print('we reject the null hypothesis.')
else:
  print('we fail to reject the null hypothesis.')
```

Python

```
7.629012511583253
2.414952103970721e-14
we reject the null hypothesis.
```

```python
Click to add a breakpoint cessing import StandardScaler, OneHotEncoder
```

```python
encoder= OneHotEncoder(handle_unknown= 'ignore', drop= 'first', sparse= False)

encoder.fit(cleaned[['route_type']])

one_hot_encoded = encoder.transform(cleaned[['route_type']])

cleaned['route_type'] = pd.DataFrame(one_hot_encoded).astype(int)
```

We have selected just the dtype that are floats using the pandas selecte dtypes method

```python
numeric_columns= cleaned.select_dtypes(include = 'float').columns
```

Python

```
Index(['start_scan_to_end_scan', 'actual_distance_to_destination',
       'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance',
       'time_between_start_and_end_in_mins', 'start_scan_to_end_scan_treated'],
      dtype='object')
```

```
Click to add a breakpoint ler()

scaler.fit(cleaned[numeric_columns])

cleaned[numeric_columns] = scaler.transform(cleaned[numeric_columns])
cleaned.head()
```

...

We can see the scaled columns below.

```
cleaned[numeric_columns].head()
```

Business Insights

Clearly from the visualizations we can see that Delhivery consistently goes over the OSRM time predicted by the software.

The team should look in to understanding whether the software consistently under predict time or if Delhivery has hard time aligning with the time.

Close to 80% of all deliveries are sent or recieved across just 17% of centeres. Delhivery should concentrate on improving the time for these centeres.

We can see that most deliveries happen across Maharashtra and Karnataka

Surprisingly Gurgaon seems to top the chart when it comes to city and Banglore as expected since the Karantaka was also listed in the top state.

We can also see that Central and Bilsapur was the most common corridors that Delihivery served.

```
print(cleaned.groupby('source_state')['trip_uuid'].nunique().sort_values(ascending= False).head(5))
print('_' * 25)
cleaned.groupby('destination_state')['trip_uuid'].nunique().sort_values(ascending= False).head(5)
```

```
source_state
Maharashtra      2278
Karnataka        2264
Haryana          1743
Tamil Nadu        959
Uttar Pradesh     850
Name: trip_uuid, dtype: int64
_________________________

destination_state
Karnataka      2304
Maharashtra    2231
Haryana        1660
Tamil Nadu     1027
Telangana       856
Name: trip_uuid, dtype: int64
```

```python
print(cleaned.groupby('source_city')['trip_uuid'].nunique().sort_values(ascending= False).head(5))
print('_' * 25)
cleaned.groupby('destination_city')['trip_uuid'].nunique().sort_values(ascending= False).head(5)
```

```
source_city
Gurgaon      1131
Bengaluru    1099
Bhiwandi      821
Bangalore     792
Mumbai        643
Name: trip_uuid, dtype: int64
_________________________

destination_city
Bengaluru    1143
Gurgaon       972
Mumbai        966
Bangalore     683
Hyderabad     635
Name: trip_uuid, dtype: int64
```

```python
print(cleaned.groupby('source_place')['trip_uuid'].nunique().sort_values(ascending= False).head(5))
print('_' * 25)
cleaned.groupby('destination_place')['trip_uuid'].nunique().sort_values(ascending= False).head(5)
```

```
source_place
Central     1696
Bilaspur    1085
Mankoli      821
Nelmngla     769
Bomsndra     468
Name: trip_uuid, dtype: int64
_________________________

destination_place
Central     1575
Bilaspur     971
Nelmngla     665
Mankoli      614
Shamshbd     464
Name: trip_uuid, dtype: int64
```

- **Recommendations**

Delihivery should review the actual time taken and the time recommended by the OSRM system. If the OSRM is consistently under reporting time taken. If these times are used to inform customers of their expected delivery and the deliveries are delayed this could lead to bad customer expereince.

It is also clear that segement time and the actual time are not equal This could indicate possible data issues with Delhiveries system. Delhivery team should review this.

Since there is clear evidence that 80% of deliveries are done across just 20% of centeres. Delihivery should work on optimizing the processes across these centers. This could potentially bring great improvements across the whole business as this improve experience for a great proportion of customers.