Business Problem Statement

The company wants to understand and process the data coming out of data engineering pipelines: Clean, sanitize and manipulate data to get useful features out of raw fields Make sense out of the raw data and help the data science team to build forecasting models on it.

The aim is to find is there significant differences among variables like expected delivery time and actual dilivery time. It crucial for business because if the algorithm that predicts time is not working correctly, then customers will get wrong estimation and if the customers will not get delivery on time then, they will be unsetisfied and raise customer service complaints. That may lead to increase operational overload, decrease revenue, decrease company reputation.

Exploratory Data Analysis

```
import numpy as np
import pandas as pd
from scipy import stats

import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('/content/delhivery_data.txt')
```

```
pd.set_option('max_columns',100)
pd.set_option('min_rows', 8)
df
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_
...	...	...	...	...	...	...	
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sor
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sor
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sor
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sor

144867 rows × 24 columns

```
df.shape
# 144867 rows, 24 columns
```

```
(144867, 24)
```

```
# Drop unnecessary/Unknown field columns
unknown = df.iloc[:,df.columns.str.contains('factor|cutoff')].columns
for i in unknown:
  df.drop(i,axis=1,inplace=True)
```

```
df_na = pd.DataFrame(df.isna().sum())
df_na['percent'] = df.isna().sum() *100/len(df)
```

```
df_na['percent'] = df_na['percent'].round(3)
df_na
# Minor rows with missing value, we can drop those rows.
```

	0	percent
data	0	0.000
trip_creation_time	0	0.000
route_schedule_uuid	0	0.000
route_type	0	0.000
trip_uuid	0	0.000
source_center	0	0.000
source_name	293	0.202
destination_center	0	0.000
destination_name	261	0.180
od_start_time	0	0.000
od_end_time	0	0.000
start_scan_to_end_scan	0	0.000
actual_distance_to_destination	0	0.000
actual_time	0	0.000
osrm_time	0	0.000
osrm_distance	0	0.000
segment_actual_time	0	0.000
segment_osrm_time	0	0.000
segment_osrm_distance	0	0.000

```
df.dropna(how='any', inplace=True)
# removed rows with null values
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 144316 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144316 non-null  object 
 1   trip_creation_time              144316 non-null  object 
 2   route_schedule_uuid             144316 non-null  object 
 3   route_type                      144316 non-null  object 
 4   trip_uuid                       144316 non-null  object 
 5   source_center                   144316 non-null  object 
 6   source_name                     144316 non-null  object 
 7   destination_center              144316 non-null  object 
 8   destination_name                144316 non-null  object 
 9   od_start_time                   144316 non-null  object 
 10  od_end_time                     144316 non-null  object 
 11  start_scan_to_end_scan          144316 non-null  float64
 12  actual_distance_to_destination  144316 non-null  float64
 13  actual_time                     144316 non-null  float64
 14  osrm_time                       144316 non-null  float64
 15  osrm_distance                   144316 non-null  float64
 16  segment_actual_time             144316 non-null  float64
 17  segment_osrm_time               144316 non-null  float64
 18  segment_osrm_distance           144316 non-null  float64
dtypes: float64(8), object(11)
memory usage: 22.0+ MB
```

```
df['od_start_time'] = pd.to_datetime(df['od_start_time'])
df['od_end_time'] = pd.to_datetime(df['od_end_time'])
# Converting the data type of columns to appropriate data type
```

```
df.nunique()
# Only few columns are categorical, mostly data is numeric
```

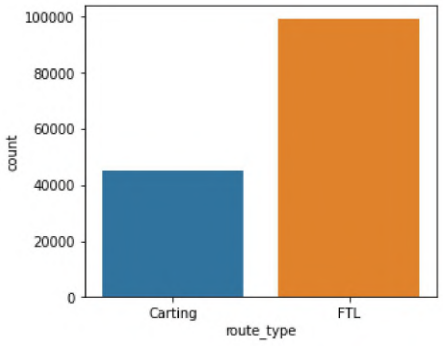
```
data                                2
trip_creation_time              14787
route_schedule_uuid              1497
route_type                          2
trip_uuid                       14787
source_center                    1496
```

```
source_name                          1496
destination_center                   1466
destination_name                     1466
od_start_time                       26223
od_end_time                         26223
start_scan_to_end_scan               1914
actual_distance_to_destination     143965
actual_time                          3182
osrm_time                            1531
osrm_distance                      137544
segment_actual_time                   746
segment_osrm_time                     214
segment_osrm_distance              113497
dtype: int64
```

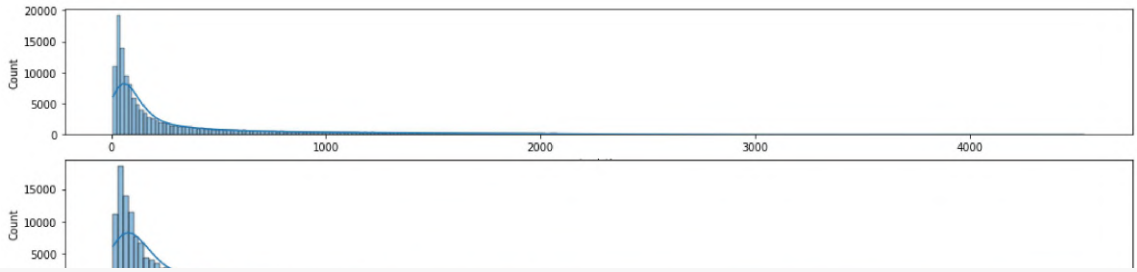
```
#Checking how the data is spread in categoric columns
df2=df.copy()
df2
categ_cols = ['route_type']
cat_count = df2[categ_cols].melt().groupby(['variable', 'value'])[['value']].size().reset_index(name='counts')
s = df2[categ_cols].melt().variable.value_counts()
cat_count['Percent'] = cat_count['counts'].div(cat_count['variable'].map(s)).mul(100).round().astype('int')
cat_count.groupby(['variable', 'value']).first()
# FTL Transportation used 69%
# Carting Transportation 31%
```

		counts	Percent
variable	**value**		
route_type	**Carting**	45184	31
	FTL	99132	69

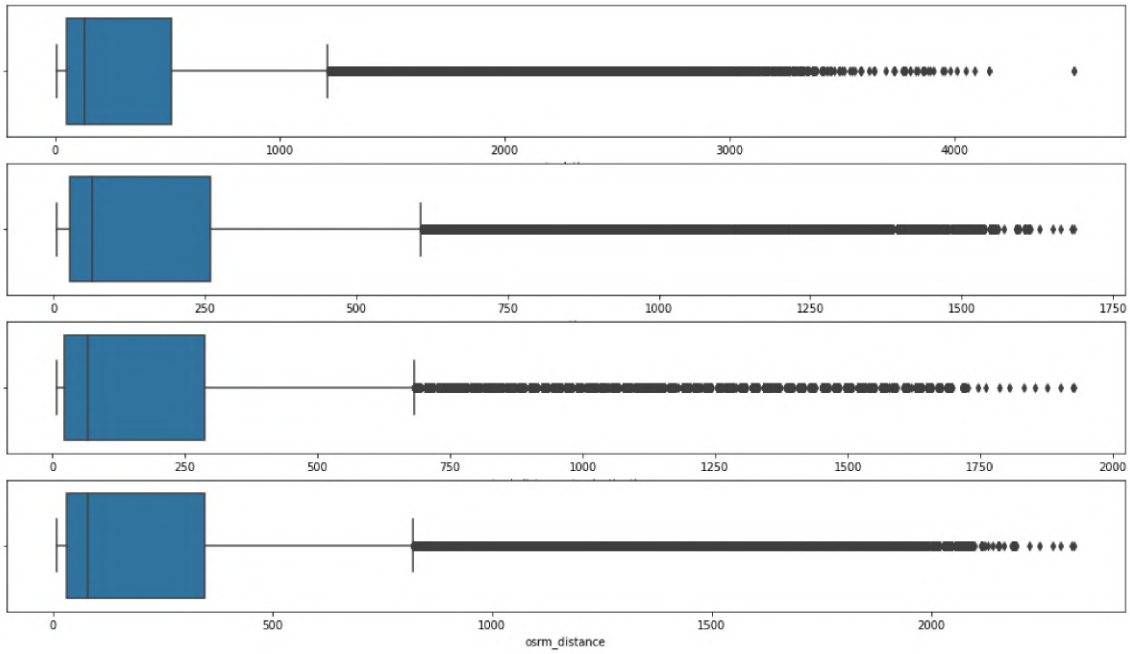
```
plt.figure(figsize = [16,4])
cat_cols = ['route_type']
for i in range (len(cat_cols)):
  plt.subplot(1, 3, i+1)
  sns.countplot(data=df, x=cat_cols[i])
```




```
plt.figure(figsize = [18,10])
num_cols = ['actual_time','osrm_time','actual_distance_to_destination','osrm_distance']
for i in range (len(num_cols)):
  plt.subplot(len(num_cols),1, i+1)
  sns.histplot(data=df, x=num_cols[i], kde=True)
```




```
plt.figure(figsize = [18,10])
num_cols = ['actual_time','osrm_time','actual_distance_to_destination','osrm_distance']
for i in range (len(num_cols)):
  plt.subplot(len(num_cols),1, i+1)
  sns.boxplot(data=df, x=num_cols[i])
```




```
df.describe()

# Data contains extreame outliers
# Variables showing are highly right skew distribution
# segment_actual_time showing negetive value that not making sense
# Data cleaning is needed
# Data contains similer segment vaulues, need to group them
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segm
count	130868.000000	130868.000000	130868.000000	130868.000000	130868.000000	
mean	941.958057	231.120919	404.065012	209.825603	279.976641	
std	1031.953320	342.810982	591.574488	305.929392	418.394128	
min	20.000000	9.000045	9.000000	6.000000	9.008200	
25%	151.000000	23.124073	48.000000	26.000000	28.792675	
50%	426.000000	63.811633	119.000000	61.000000	75.009050	
75%	1597.000000	286.273519	488.000000	251.000000	335.047625	
max	4535.000000	1924.221719	4532.000000	1682.000000	2319.934400	

groupby and aggregations

```
# Grouping the data at journey level, Creating data with unique index value as segment_key.
df['segment_key'] = df['trip_uuid'] + df['source_center'] + df['destination_center']
segment_cols = ['segment_actual_time','segment_osrm_distance','segment_osrm_time']
```

```
for col in segment_cols:
  df[col+'_sum'] = df.groupby('segment_key')[col].cumsum()
df[[col + '_sum' for col in segment_cols]].head()
```

	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_sum
0	14.0	11.9653	11.0
1	24.0	21.7243	20.0
2	40.0	32.5395	27.0
3	61.0	45.5619	39.0
4	67.0	49.4772	44.0

```
# Aggregating at sub-journey level
create_segment_dict = {
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',

    'source_center' : 'first',
    'source_name' : 'first',

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

    'od_start_time' : 'first',
    'od_end_time' : 'first',
    'start_scan_to_end_scan' : 'first',

    'actual_distance_to_destination' : 'last',
    'actual_time' : 'last',

    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',

    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum'
}
```

```
segment = df.groupby('segment_key').agg(create_segment_dict).reset_index()
segment = segment.sort_values(by=['segment_key','od_end_time'],ascending=True).reset_index()
```

feature cration and merging of rows

```
# Feature creation: Calculating time taken between od_start_time and od_end_time, keep it as feature.
segment['od_time_diff_hour'] = (segment['od_end_time'] - segment['od_start_time']).dt.total_seconds()/60
```

```
segment
# od_time_diff_hour is matching with start_scan_to_end_scan
```

	index	segment_key	data	trip_creation_time	route_schedule_uuid
0	0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...
1	1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...
2	2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-

```
segment.nunique()
#now we have the data without duplicates, means we have unique value for ever row as segment_key.
```

```
index                           26222
segment_key                     26222
data                                2
trip_creation_time              14787
route_schedule_uuid              1497
route_type                          2
trip_uuid                       14787
source_center                    1496
source_name                      1496
destination_center               1466
destination_name                 1466
od_start_time                   26222
od_end_time                     26222
start_scan_to_end_scan           1914
actual_distance_to_destination  26193
actual_time                      1657
osrm_time                        2723
osrm_distance                   26089
segment_actual_time_sum          3839
segment_osrm_distance_sum       26102
segment_osrm_time_sum            2964
od_time_diff_hour               26222
dtype: int64
```

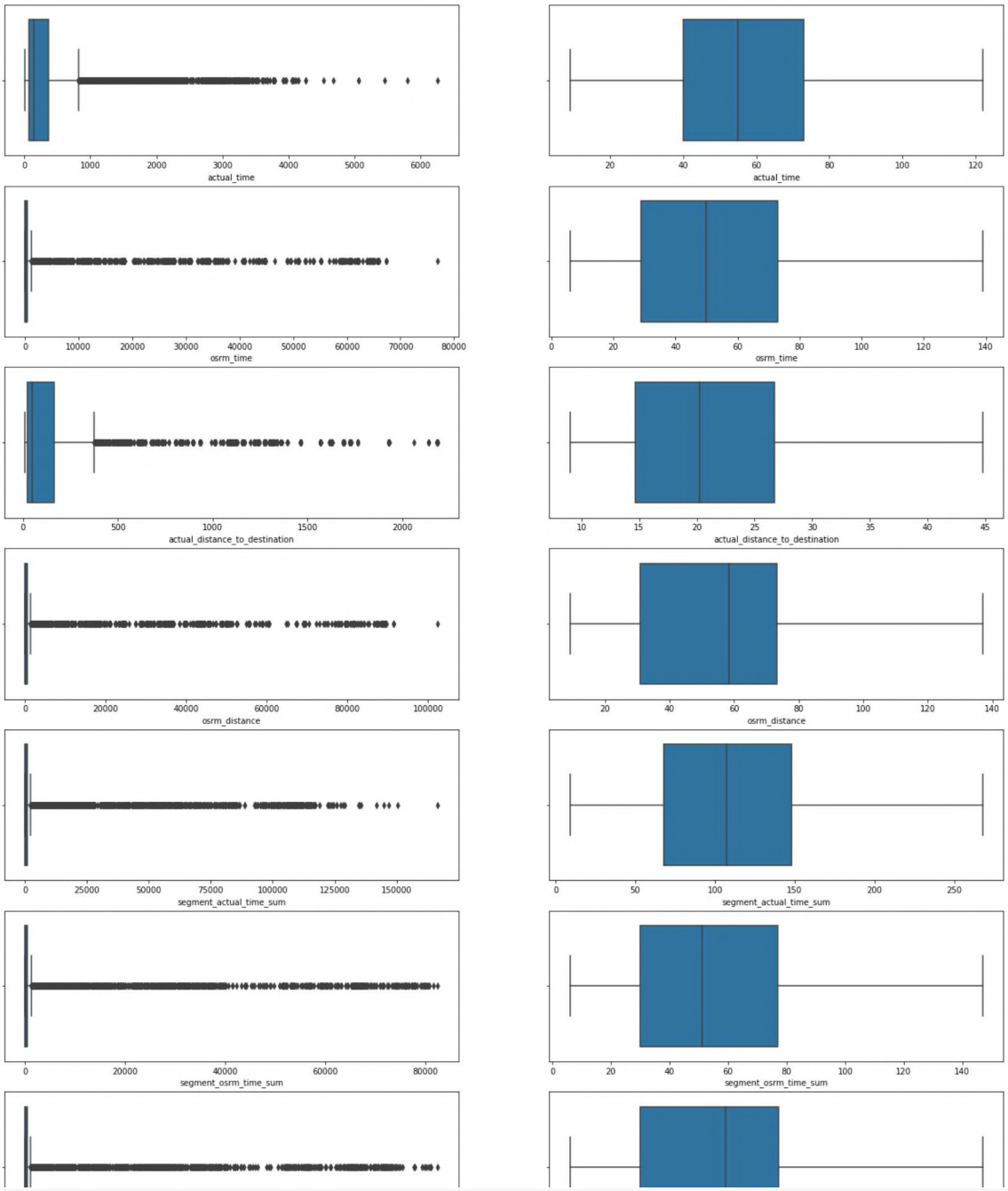
```
create_trip_dict = {
    'data' : 'first',
    'trip_creation_time' : 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',

    'source_center' : 'first',
    'source_name' : 'first',

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

    'start_scan_to_end_scan' : 'sum',
    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',

    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum'
}
```

```
trip = segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop=True)
trip
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	Ka
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Doddab
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB	Gur
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072AAB	
...	...	...	...	...	...	...	

Outlier treatment

```
trip_o = trip.copy()

def find_outliers_IQR(col):
  q1=col.quantile(0.25)
  q3=col.quantile(0.75)
  IQR=q3-q1
  outliers = trip[((col<(q1-1.5*IQR)) | (col>(q3+1.5*IQR)))]
  return outliers

cols = ['actual_time', 'osrm_time','actual_distance_to_destination','osrm_distance','segment_actual_time_sum','segment_osrm_time_sum',

n=1
while n!=0:
  n=0
  for x in cols:
    outliers = find_outliers_IQR(trip[x]).index
    trip.drop(outliers,inplace=True)
    n+=len(outliers)


fig, axis = plt.subplots(nrows=len(cols), ncols=2, figsize=(22, len(cols)*4))
for i in range (len(cols)):
  for j in ([0,1]):
    if j==0:
      sns.boxplot(data=trip_o, x=cols[i], ax=axis[i, j])
    else:
      sns.boxplot(data=trip, x=cols[i], ax=axis[i, j])
# Left side plots: boxplot distribution before removing outliers
# Right side plots: boxplot distribution after removing outliers
```




```
1 trip_o.describe()
2 # before outlier treatment there was high difference between median and mean values.
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segmer
count	14787.000000	14787.000000	14787.000000	14787.000000	14787.000000	
mean	529.429025	164.090196	356.306012	2092.831406	2786.867034	
std	658.254936	305.502982	561.517936	7964.769556	10769.741065	
min	23.000000	9.002461	9.000000	6.000000	9.072900	
25%	149.000000	22.777099	67.000000	61.000000	65.635550	
50%	279.000000	48.287894	148.000000	167.000000	172.886300	
75%	632.000000	163.591258	367.000000	510.500000	603.955400	
max	7898.000000	2186.531787	6265.000000	76953.000000	102415.868000	

```
1 trip.describe()
2 # after outlier treatment there the mean values is closer to median.
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment
count	5160.000000	5160.000000	5160.000000	5160.000000	5160.000000	
mean	155.022481	20.817280	57.434690	54.112209	57.274155	
std	121.042528	8.006889	23.920582	29.317440	29.436978	
min	23.000000	9.002461	9.000000	6.000000	9.072900	
25%	89.000000	14.641301	40.000000	29.000000	30.853800	
50%	127.000000	20.266599	55.000000	50.000000	58.327150	
75%	182.000000	26.713161	73.000000	73.000000	73.351800	

```
len(trip_o),len(trip)
```

```
(14787, 5160)
```

Hypothesis testing Comparisons

```
# hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value

# Checking: Does segment_actual_time is similer as segment_osrm_time?

from scipy.stats import ttest_ind
null_hypothesis = 'mean of actual_time is not higher than mean of osrm_time'
alternative_hypothesis = 'mean of actual_time is higher than mean of osrm_time'

sample1 = trip['actual_time']
sample2 = trip['osrm_time']
t_stat, p_value = ttest_ind(sample1, sample2, equal_var=False, alternative='greater')
print(t_stat, p_value)

if(p_value < 0.05):
  print('Since, p-value < 0.05, the null hypothesis is rejected')
  print(alternative_hypothesis)
else:
  print('Since p-value > 0.05, we fail to reject null hypothesis')
  print(null_hypothesis)

# conclusion: mean of actual_time is higher than mean of osrm_time
```

```
6.307544212335001 1.4779513885349015e-10
Since, p-value < 0.05, the null hypothesis is rejected
mean of actual_time is higher than mean of osrm_time
```

```
# hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value

# Checking: Does actual_time is similer as segment_osrm_time?

from scipy.stats import ttest_ind
null_hypothesis = 'mean of actual_time is similer as segment_actual_time'
alternative_hypothesis = 'mean of actual_time is different than mean of segment_osrm_time'

sample1 = trip['actual_time']
sample2 = trip['segment_actual_time_sum']
t_stat, p_value = ttest_ind(sample1, sample2)
print(t_stat, p_value)

if(p_value < 0.05):
  print('Since, p-value < 0.05, the null hypothesis is rejected')
  print(alternative_hypothesis)
else:
  print('Since p-value > 0.05, we fail to reject null hypothesis')
  print(null_hypothesis)

# conclusion: mean of actual_time is different than mean of segment_osrm_time
```

```
-65.79091499324902 0.0
Since, p-value < 0.05, the null hypothesis is rejected
mean of actual_time is different than mean of segment_osrm_time
```

```
# Checking: Does osrm_distance is similer as segment_osrm_distance_sum?

from scipy.stats import ttest_ind
null_hypothesis = 'mean of osrm_distance is similer as mean of segment_osrm_distance_sum'
alternative_hypothesis = 'mean of osrm_distance is higher than mean of segment_osrm_distance_sum'

```

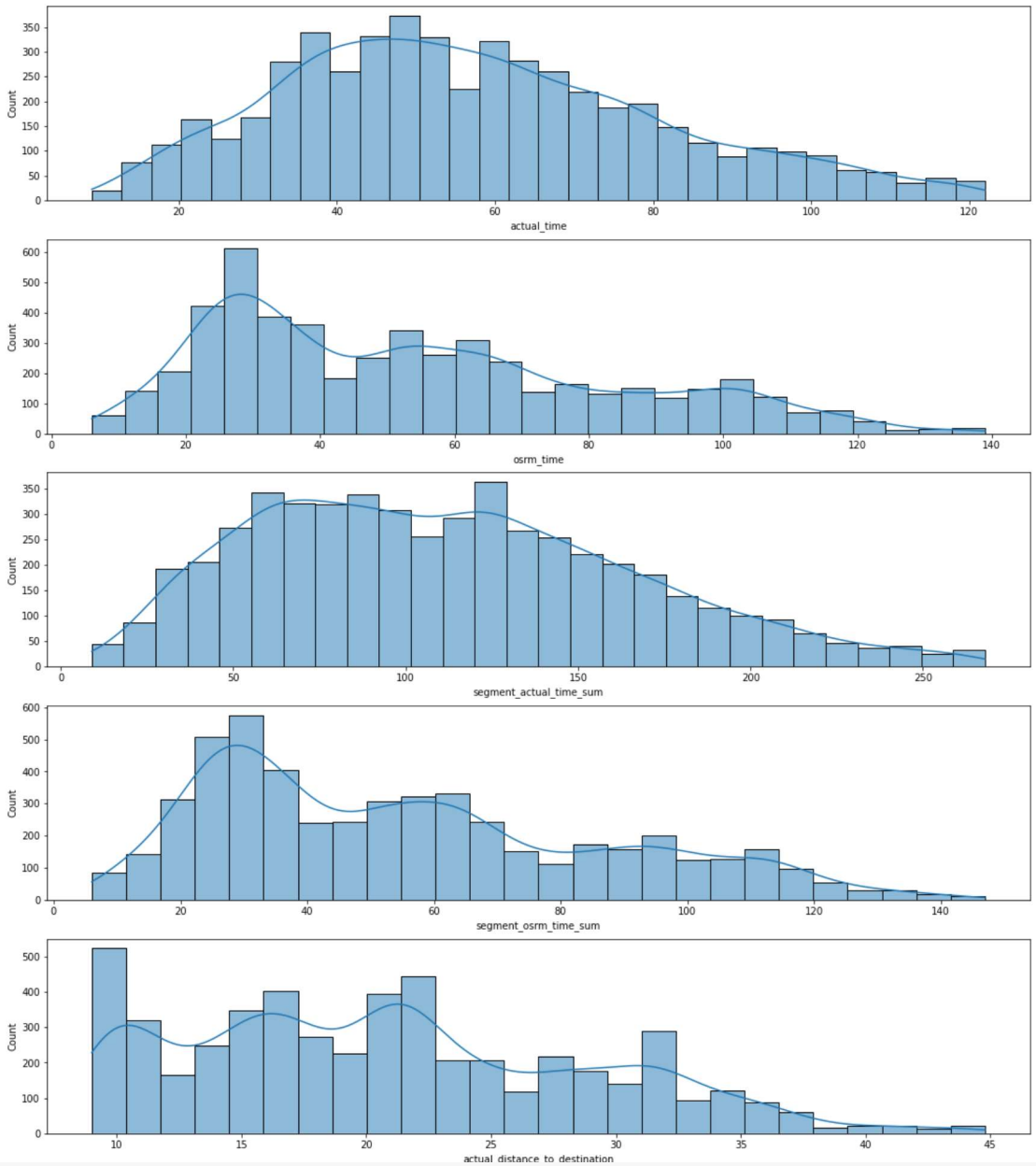
```
sample1 = trip['osrm_distance']
sample2 = trip['segment_osrm_distance_sum']
t_stat, p_value = ttest_ind(sample1, sample2, equal_var=False, alternative='greater')
print(t_stat, p_value)

if(p_value < 0.05):
  print('Since, p-value < 0.05, the null hypothesis is rejected')
  print(alternative_hypothesis)
else:
  print('Since p-value > 0.05, we fail to reject null hypothesis')
  print(null_hypothesis)

# conclusion: mean of osrm_distance is similer as mean of segment_osrm_distance_sum
```

```
-3.4156110488999936 0.9996805751383507
Since p-value > 0.05, we fail to reject null hypothesis
mean of osrm_distance is similer as mean of segment_osrm_distance_sum
```

```
num_cols = ['actual_time','osrm_time','segment_actual_time_sum','segment_osrm_time_sum','actual_distance_to_destination','osrm_distanc
for i in (num_cols):
  stat, p_value = stats.shapiro(sample1)
  if(p_value < 0.05):
    print(i, ": sample is not normally distributed, do non parametric test")
  else:
    print(i, ": sample is normally distributed, can do parametric test")
```

```
actual_time : sample is not normally distributed, do non parametric test
osrm_time : sample is not normally distributed, do non parametric test
segment_actual_time_sum : sample is not normally distributed, do non parametric test
segment_osrm_time_sum : sample is not normally distributed, do non parametric test
actual_distance_to_destination : sample is not normally distributed, do non parametric test
osrm_distance : sample is not normally distributed, do non parametric test
segment_osrm_distance_sum : sample is not normally distributed, do non parametric test
/usr/local/lib/python3.8/dist-packages/scipy/stats/morestats.py:1760: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
plt.figure(figsize = [18,30])
num_cols = ['actual_time','osrm_time','segment_actual_time_sum','segment_osrm_time_sum','actual_distance_to_destination','osrm_distanc
for i in range (len(num_cols)):
  plt.subplot(len(num_cols),1, i+1)
  sns.histplot(data=trip, x=num_cols[i], kde=True)
# Distribution are not normally distributed, we will do non parametric tests
```




```
# H0: mean of both samples are similer
# Ha: means of both samples are different
from scipy.stats import mannwhitneyu

sample1 = trip['actual_time']
sample2 = trip['osrm_time']
# perform mann whitney test
stat, p_value = mannwhitneyu(sample1, sample2)
print('Statistics=%.2f, p=%.2f' % (stat, p_value))
# Level of significance
alpha = 0.05
# conclusion
if p_value < alpha:
    print('Reject Null Hypothesis (Significant difference between two samples)')
else:
    print('Do not Reject Null Hypothesis (No significant difference between two samples)')
```

```
Statistics=14829121.00, p=0.00
Reject Null Hypothesis (Significant difference between two samples)
```

```
# H0: mean of both samples are similer
# Ha: means of both samples are different
from scipy.stats import mannwhitneyu

sample1 = trip['osrm_distance']
sample2 = trip['segment_osrm_distance_sum']
# perform mann whitney test
stat, p_value = mannwhitneyu(sample1, sample2)
print('Statistics=%.2f, p=%.2f' % (stat, p_value))
# Level of significance
alpha = 0.05
# conclusion
```

```
if p_value < alpha:
    print('Reject Null Hypothesis (Significant difference between two samples)')
else:
    print('Do not Reject Null Hypothesis (No significant difference between two samples)')
```

```
Statistics=12813705.50, p=0.00
Reject Null Hypothesis (Significant difference between two samples)
```

Standardization, Normalization

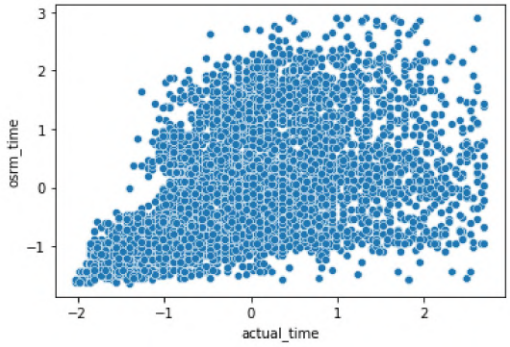
```
df_ao = trip[["actual_time", "osrm_time"]]
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
df_ao_ss = StandardScaler().fit_transform(df_ao) # ss--> standard scaler z-score
```

```
df_ao_ss = pd.DataFrame(df_ao_ss, columns=["actual_time", "osrm_time"])
```

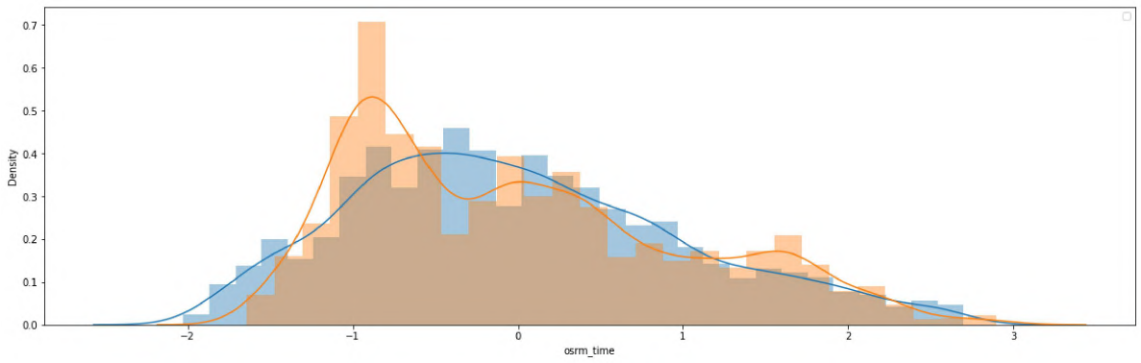
```
sns.scatterplot(x=df_ao_ss["actual_time"], y=df_ao_ss["osrm_time"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c55bdeca0>
```




```
plt.figure(figsize = (20,6))

sns.distplot(df_ao_ss['actual_time'])
sns.distplot(df_ao_ss['osrm_time'])

plt.legend()
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated
  warnings.warn(msg, FutureWarning)
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```




```
# hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value

# Checking: Does segment_actual_time is similer as segment_osrm_time?

from scipy.stats import ttest_ind
null_hypothesis = 'mean of actual_time is similer to osrm_time'
alternative_hypothesis = 'mean of actual_time is different than osrm_time'
```

```
8
9 sample1 = df_ao_ss['actual_time']
10 sample2 = df_ao_ss['osrm_time']
11 t_stat, p_value = ttest_ind(sample1, sample2)
12 print(t_stat, p_value)
13
14 if(p_value < 0.05):
15   print('Since, p-value < 0.05, the null hypothesis is rejected')
16   print(alternative_hypothesis)
17 else:
18   print('Since p-value > 0.05, we fail to reject null hypothesis')
19   print(null_hypothesis)
20
21 # conclusion: mean of actual_time is similer to osrm_time (with following the standardization approach)
```

```
-4.301138070642346e-15 0.9999999999999966
Since p-value > 0.05, we fail to reject null hypothesis
mean of actual_time is similer to osrm_time
```

```
1 df_ao_ss.mean()
```

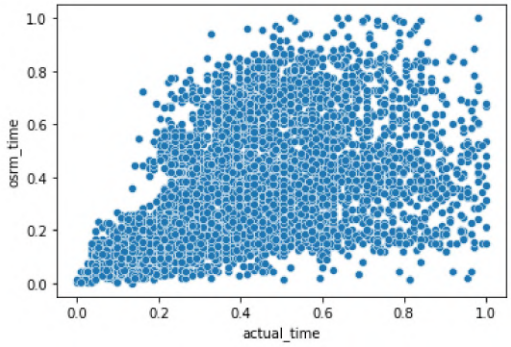
```
actual_time   -8.950635e-17
osrm_time     -4.819573e-18
dtype: float64
```

```
1 df_ao_mm.mean()
```

```
actual_time    0.428626
osrm_time      0.361746
dtype: float64
```

```
1 df_ao_mm = MinMaxScaler().fit_transform(df_ao)
2 df_ao_mm = pd.DataFrame(df_ao_mm, columns=["actual_time", "osrm_time"])
3 sns.scatterplot(x=df_ao_mm["actual_time"], y=df_ao_mm["osrm_time"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c537513a0>
```




```
1 plt.figure(figsize = (20,6))
2
3 sns.distplot(df_ao_mm['actual_time'])
4 sns.distplot(df_ao_mm['osrm_time'])
5
6 plt.legend()
7 plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated
  warnings.warn(msg, FutureWarning)
```

Handling categorical values

```
trip.nunique()
```

```
data                                2
trip_creation_time               5160
route_schedule_uuid               746
route_type                          2
trip_uuid                        5160
source_center                     396
source_name                       396
destination_center                415
destination_name                  415
start_scan_to_end_scan            500
actual_distance_to_destination   5151
actual_time                       114
osrm_time                         134
osrm_distance                    5140
segment_actual_time_sum           260
segment_osrm_distance_sum        5142
segment_osrm_time_sum             141
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
trip[col] = label_encoder.fit_transform(trip['route_type'])
trip[col].value_counts()
```

```
0    4907
1     253
Name: route_type, dtype: int64
```

```
trip['data'].value_counts()
```

```
training    3599
test        1561
Name: data, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
trip[col] = label_encoder.fit_transform(trip['data'])
trip[col].value_counts()
```

```
1    3599
0    1561
Name: route_type, dtype: int64
```

Handling missing values

```
from sklearn.impute import SimpleImputer
dm = pd.read_csv('/content/delhivery_data.txt')
dm.head(3)

# Because data contains very few missing values, so we removed them.
# If we would get data with many missing value, then we would imputation approach:
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	sour
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNa (
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNa (
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNa (

```
dm.isna().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
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
```

```
dm['source_name']  = SimpleImputer(strategy="most_frequent").fit_transform(dm[['source_name']])
```

```
dm['destination_name'] = SimpleImputer(strategy="most_frequent").fit_transform(dm[['destination_name']])
```

```
dm.isna().sum()
```

```
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
```

Columns split

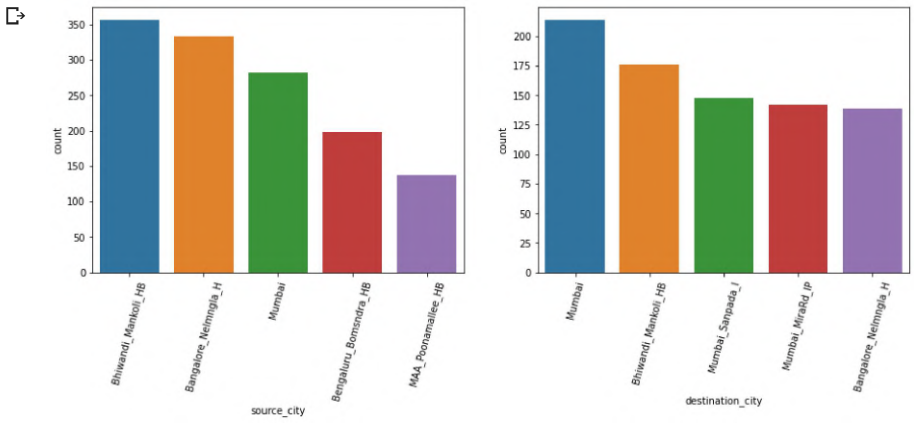
```
ds = trip[['destination_name']].copy()

new = trip['source_name'].str.split(" ", n = 1, expand = True)
ds['source_city']= new[0]
ds['source_state']= new[1].str[1:-1]

new = trip['destination_name'].str.split(" ", n = 1, expand = True)
ds['destination_city']= new[0]
ds['destination_state']= new[1].str[1:-1]

ds['Corridor'] = ds['source_city']+" To "+ds['destination_city']

ds
```

	destination_name	source_city	source_state	destination_city	destination_state
3	Mumbai_MiraRd_IP (Maharashtra)	Mumbai	ub (Maharashtra	Mumbai_MiraRd_IP	Maharashtra
5	Chennai_Poonamallee (Tamil Nadu)	Chennai_Poonamallee	Tamil Nadu	Chennai_Poonamallee	Tamil Nadu
6	Chennai_Vandalur_Dc (Tamil Nadu)	Chennai_Chrompet_DPC	Tamil Nadu	Chennai_Vandalur_Dc	Tamil Nadu
7	HBR Layout PC (Karnataka)	HBR	ayout PC (Karnataka	HBR	ayout PC (Karnataka
...	...	...	...	...	...
14779	Chennai_Thiruvlr_DC (Tamil Nadu)	Chennai_Porur_DPC	Tamil Nadu	Chennai_Thiruvlr_DC	Tamil Nadu

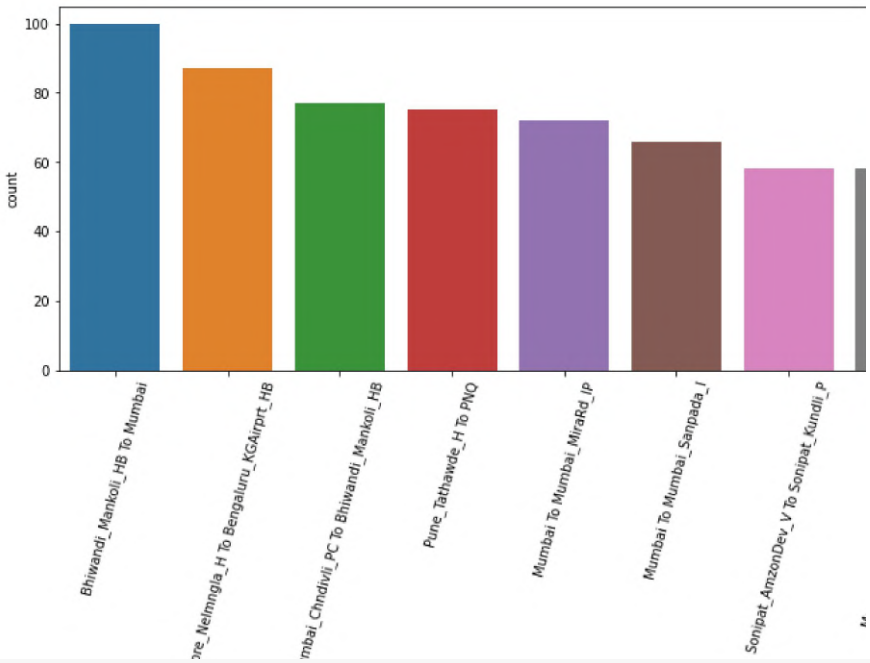
```
ds['Corridor'].value_counts()
# There are 658 routes
```

```
Bhiwandi_Mankoli_HB To Mumbai                    100
Bangalore_Nelmngla_H To Bengaluru_KGAirprt_HB     87
Mumbai_Chndivli_PC To Bhiwandi_Mankoli_HB         77
Pune_Tathawde_H To PNQ                            75
                                                 ...
Delhi_Rohini_DPC To Delhi_Barwala                  1
Wardha_RamaNgr_D To Deoli_Central_DPP_2            1
Nadiad_DC To Nadiad_DC                             1
Janakpuri To Delhi_Nangli_IP                       1
Name: Corridor, Length: 658, dtype: int64
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.countplot(data= ds, x='source_city', order=ds['source_city'].value_counts().nlargest(5).index)
plt.xticks(rotation = 75)

plt.subplot(1,2,2)
sns.countplot(data= ds, x='destination_city', order=ds['destination_city'].value_counts().nlargest(5).index)
plt.xticks(rotation = 75)
plt.show()

# most orders are coming from Bhindwandi_Mankoli_HB
# most orders are going to Mumbai
# Left plot: top 5 cities acting as source point
# right plot: top 5 cities acting as destination point
```

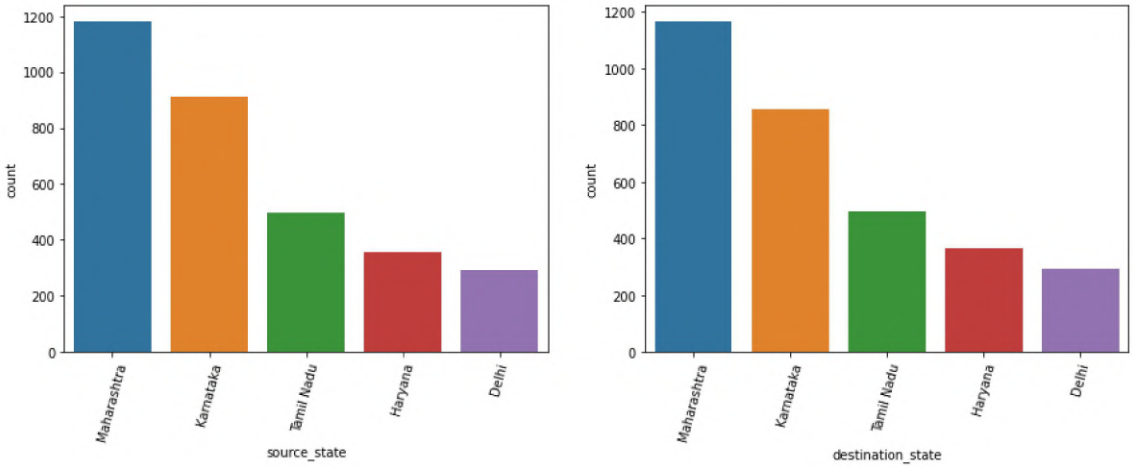



```
plt.figure(figsize=(15,5))
sns.countplot(data= ds, x='Corridor', order=ds['Corridor'].value_counts().nlargest(10).index)
plt.xticks(rotation = 75)
plt.show()

# the busiest route is Bhindwandi_Mankoli_HB to Mumbai
# Top 10 busiest routes
```



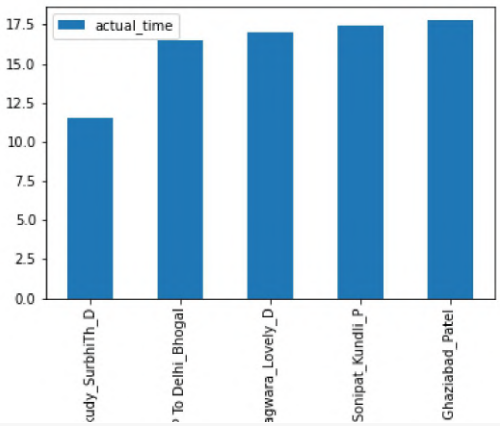

```
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.countplot(data= ds, x='source_state', order=ds['source_state'].value_counts().nlargest(5).index)
plt.xticks(rotation = 75)

plt.subplot(1,2,2)
sns.countplot(data= ds, x='destination_state', order=ds['destination_state'].value_counts().nlargest(5).index)
plt.xticks(rotation = 75)

# most orders are coming from Maharashtra state
# most orders are going to  Maharashtra state
# Left plot: top 5 cities acting as source point
# right plot: top 5 cities acting as destination point
```

```
(array([0, 1, 2, 3, 4]), <a list of 5 Text major ticklabel objects>)
```

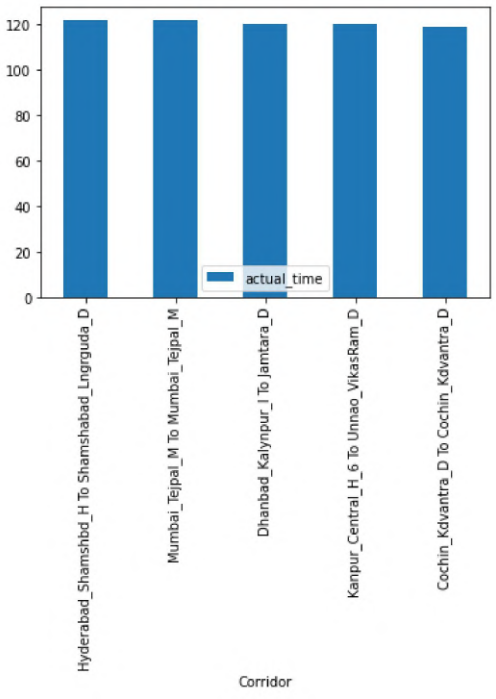



```
dn=pd.concat([trip,ds],axis=1)

```

```
dn.groupby('Corridor').agg({'actual_time':'mean'}).nsmallest(5,columns='actual_time').plot(kind='bar')
plt.show()

# trip between cities Angamaly to Chalakudy saw the least avg time for completion
```




```
dn.groupby('Corridor').agg({'actual_time':'mean'}).nlargest(5,columns='actual_time').plot(kind='bar')
plt.show()

# trip between cities Hyderabad to Shamshabad saw the highest avg time for completion
```




```
ds.describe()
# 1179 orders are coming from state - Maharastra
```

	destination_name	source_city	source_state	destination_city	destination_state	Corr
count	5160	5160	5160	5160	5160	
unique	415	391	42	411	44	
top	Mumbai Hub (Maharashtra)	Bhiwandi_Mankoli_HB	Maharashtra	Mumbai	Maharashtra	Bhiwandi_Manko To Mu
freq	212	357	1179	214	1164	

```
dn.describe()
# Average actual time for a trip is 57 Hour
```

	route_type	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_dista
count	5160.000000	5160.000000	5160.000000	5160.000000	5160.000000	5160.000
mean	0.697481	155.022481	20.817280	57.434690	54.112209	57.274

```
trip_cr = df[['trip_creation_time']].copy()
trip_cr['trip_creation_time'] = pd.to_datetime(trip_cr['trip_creation_time'])
trip_cr['year'] = trip_cr['trip_creation_time'].dt.year
trip_cr['month'] = trip_cr['trip_creation_time'].dt.month
trip_cr['day'] = trip_cr['trip_creation_time'].dt.day
trip_cr
# Trip creation time variable is not making much sense
```

	trip_creation_time	year	month	day
0	2018-09-20 02:35:36.476840	2018	9	20
1	2018-09-20 02:35:36.476840	2018	9	20
2	2018-09-20 02:35:36.476840	2018	9	20
3	2018-09-20 02:35:36.476840	2018	9	20
...	...	...	...	...
144862	2018-09-20 16:24:28.436231	2018	9	20
144863	2018-09-20 16:24:28.436231	2018	9	20
144864	2018-09-20 16:24:28.436231	2018	9	20
144865	2018-09-20 16:24:28.436231	2018	9	20

130868 rows × 4 columns

```
trip_cr['year'].value_counts()
# Data contains order from only 2018
```

```
2018    130868
Name: year, dtype: int64
```

```
trip_cr['month'].value_counts()
# Data contains order from september and october month.
```

```
9     114979
10     15889
Name: month, dtype: int64
```

Business Insights and Recommendations

```
# variables showing are highly right skew distribution
# mean of actual_time is higher than mean of osrm_time
# mean of actual_time is different than mean of segment_osrm_time
# mean of osrm_distance is similar as mean of segment_osrm_distance
# FTL Transportation used 69%
# Carting Transportation 31%
# There are 658 routes connecting sources to destinations
# most orders are coming from Bhindwandi_Mankoli_HB
# most orders are going to Mumbai
# the busiest route is Bhindwandi_Mankoli_HB to Mumbai
# company can use more transportation vehicles on this route
# most orders are coming from Maharashtra state
# most orders are going to  Maharashtra state
# trip between cities Angamaly to Chalakudy saw the least avg time for completion
# trip between cities Hyderabad to Shamshabad saw the highest avg time for completion
# By visulaizing plots, we obserbed that there is not much difference between pair variable
# All the appropriate insights and recommendations are mentioned with relevent code and plots
```