About Delhivery

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors

Purpose

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

Column Profiling:

- data - tells whether the data is testing or training data
- trip_creation_time – Timestamp of trip creation
- route_schedule_uuid – Unique Id for a particular route schedule
- route_type – Transportation type
- FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
- Carting: Handling system consisting of small vehicles (carts)
- trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
- source_center - Source ID of trip origin
- source_name - Source Name of trip origin
- destination_cente – Destination ID
- destination_name – Destination Name
- od_start_time – Trip start time
- od_end_time – Trip end time
- start_scan_to_end_scan – Time taken to deliver from source to destination
- is_cutoff – Unknown field
- cutoff_factor – Unknown field
- cutoff_timestamp – Unknown field

- actual_distance_to_destination – Distance in Kms between source and destination warehouse
- actual_time – Actual time taken to complete the delivery (Cumulative)
- osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
- osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
- factor – Unknown field
- segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
- segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
- segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
- segment_factor – Unknown field

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

```
dt = pd.read_csv('delhivery_data.csv')
```

```
dt.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
```

```
dt.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
```

```
r = dt.shape[0]
c = dt.shape[1]
print(f'The Total number of Rows - {r} and Total number of Columns = {c}')
```

```
The Total number of Rows - 144867 and Total number of Columns = 24
```

Statistical Summary

```
dt.describe()
```

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actua
count	144867.000000	144867.000000	144867.000000	144867
mean	961.262986	232.926567	234.073372	416
std	1037.012769	344.755577	344.990009	598
min	20.000000	9.000000	9.000045	9
25%	161.000000	22.000000	23.355874	51
50%	449.000000	66.000000	66.126571	132

Dropping the Unknown Columns

```
dt.drop(['segment_factor', 'factor', 'cutoff_timestamp', 'cutoff_factor', 'is_cutoff'], ax
```

Handling / Imputing the null Values

```
dt['source_name'].fillna('City_place_code (State)', inplace = True)
dt['destination_name'].fillna('City_place_code (State)', inplace = True)
```

Outlier detection

```
def outlier(data):
    '''
    Function to Identify Outliers
    '''
    for i in data:
        if type(data[i].values[0])!=str:
            iqr = np.percentile(data[i], 75) - np.percentile(data[i], 25)
            upper_limit = np.percentile(data[i], 75) + iqr*1.5
            lower_limit = np.percentile(data[i], 25) - iqr*1.5
            l = len(data[i][(data[i]<lower_limit) | (data[i]>upper_limit)])
            print(f'Number of possible outliers in {i} =',l)
```

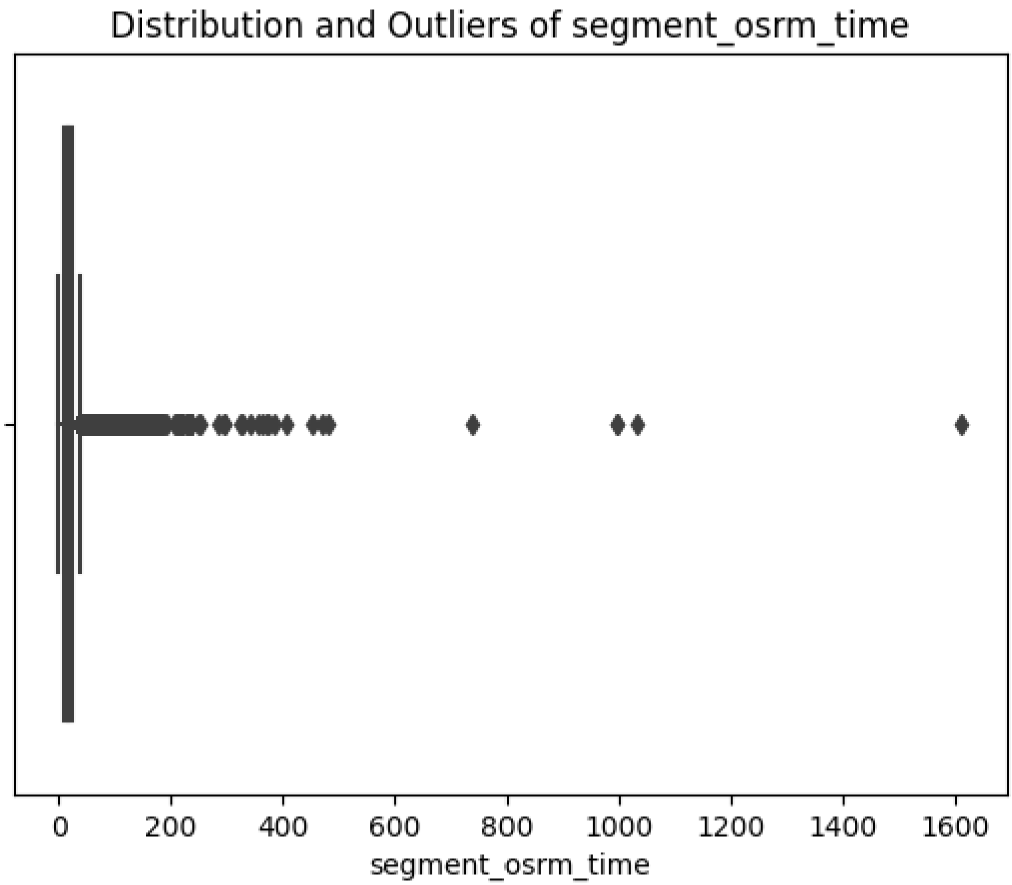
```
outlier(dt)
```

```
Number of possible outliers in start_scan_to_end_scan = 373
Number of possible outliers in actual_distance_to_destination = 17992
Number of possible outliers in actual_time = 16633
Number of possible outliers in osrm_time = 17603
Number of possible outliers in osrm_distance = 17816
Number of possible outliers in segment_actual_time = 9298
Number of possible outliers in segment_osrm_time = 6378
Number of possible outliers in segment_osrm_distance = 4315
```

Visualizing Data and Outliers

```
sns.boxplot(data = dt, x ='segment_osrm_time' )
plt.title('Distribution and Outliers of segment_osrm_time')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of segment_osrm_time')
```

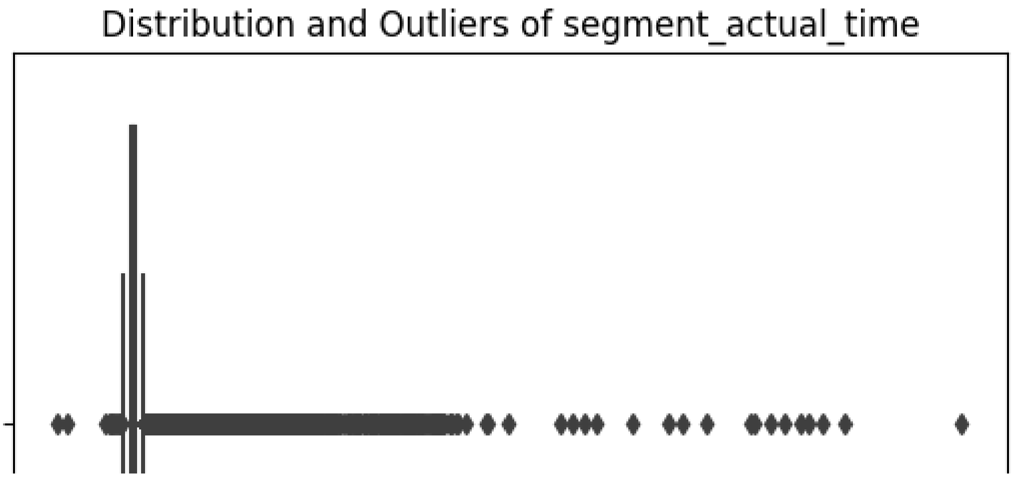



```
mi = min(dt['segment_osrm_time'])
ma = max(dt['segment_osrm_time'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = 0.0 and Maximum value = 1611.0
```

```
sns.boxplot(data = dt, x ='segment_actual_time' )
plt.title('Distribution and Outliers of segment_actual_time')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of segment_actual_time')
```

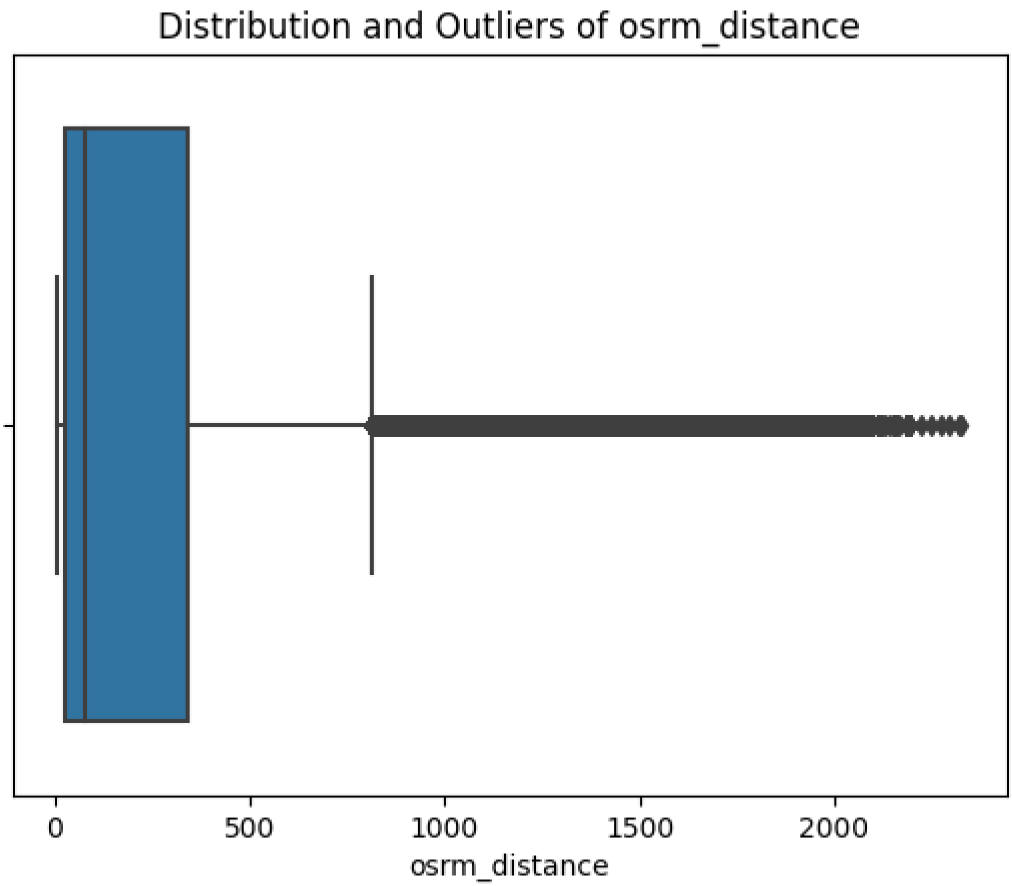



```
mi = min(dt['segment_actual_time'])
ma = max(dt['segment_actual_time'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = -244.0 and Maximum value = 3051.0
```

```
sns.boxplot(data = dt, x ='osrm_distance' )
plt.title('Distribution and Outliers of osrm_distance')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of osrm_distance')
```

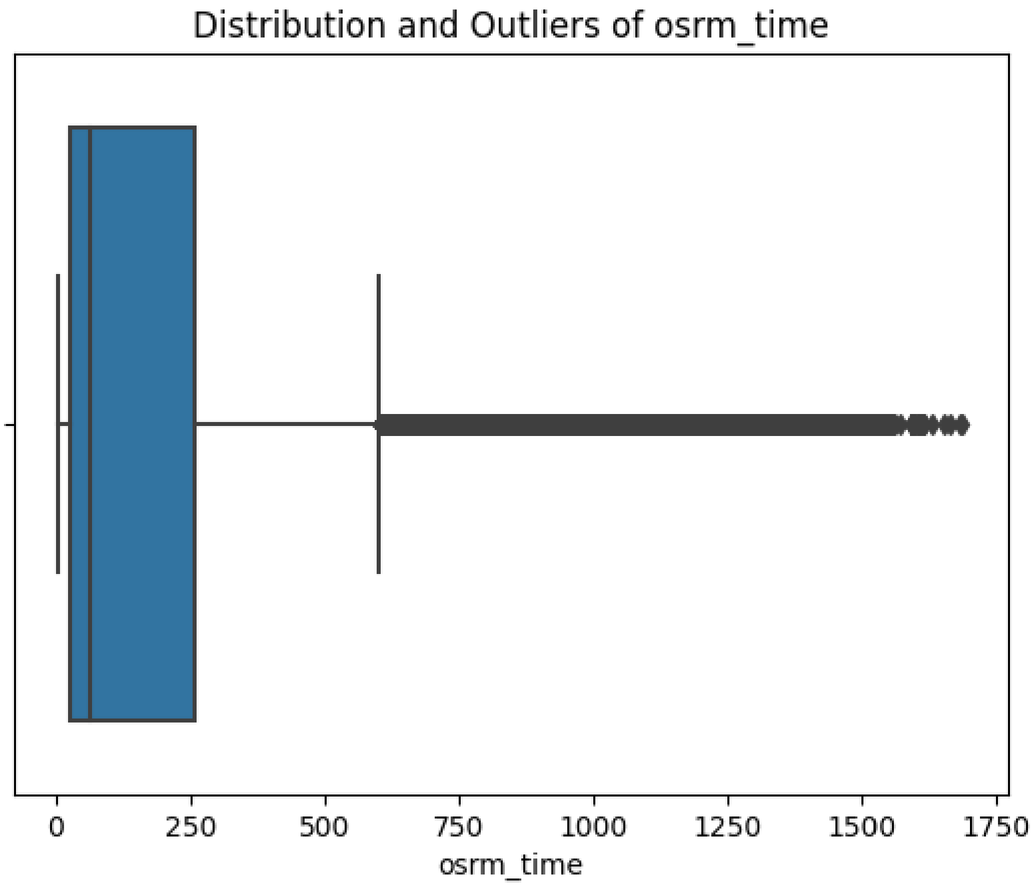



```
mi = min(dt['osrm_distance'])
ma = max(dt['osrm_distance'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = 9.0082 and Maximum value = 2326.1991000000003
```

```
sns.boxplot(data = dt, x ='osrm_time' )
plt.title('Distribution and Outliers of osrm_time')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of osrm_time')
```

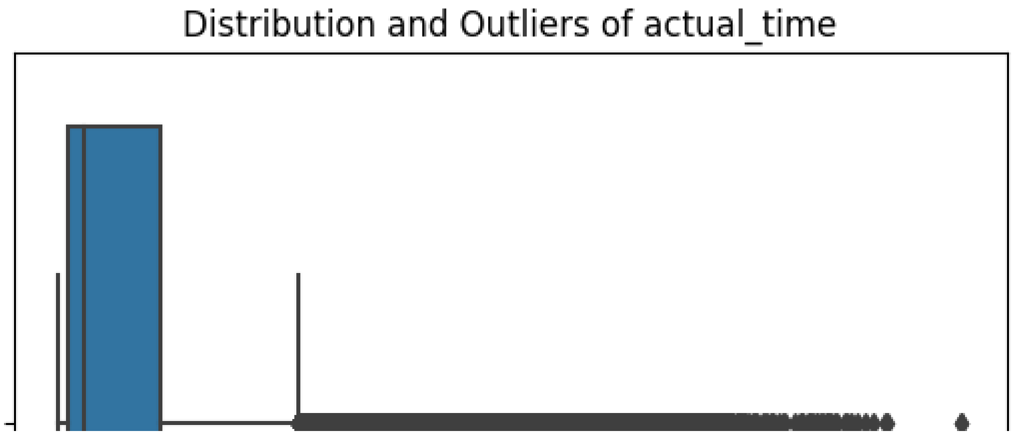



```
mi = min(dt['osrm_time'])
ma = max(dt['osrm_time'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = 6.0 and Maximum value = 1686.0
```

```
sns.boxplot(data = dt, x ='actual_time' )
plt.title('Distribution and Outliers of actual_time')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of actual_time')
```

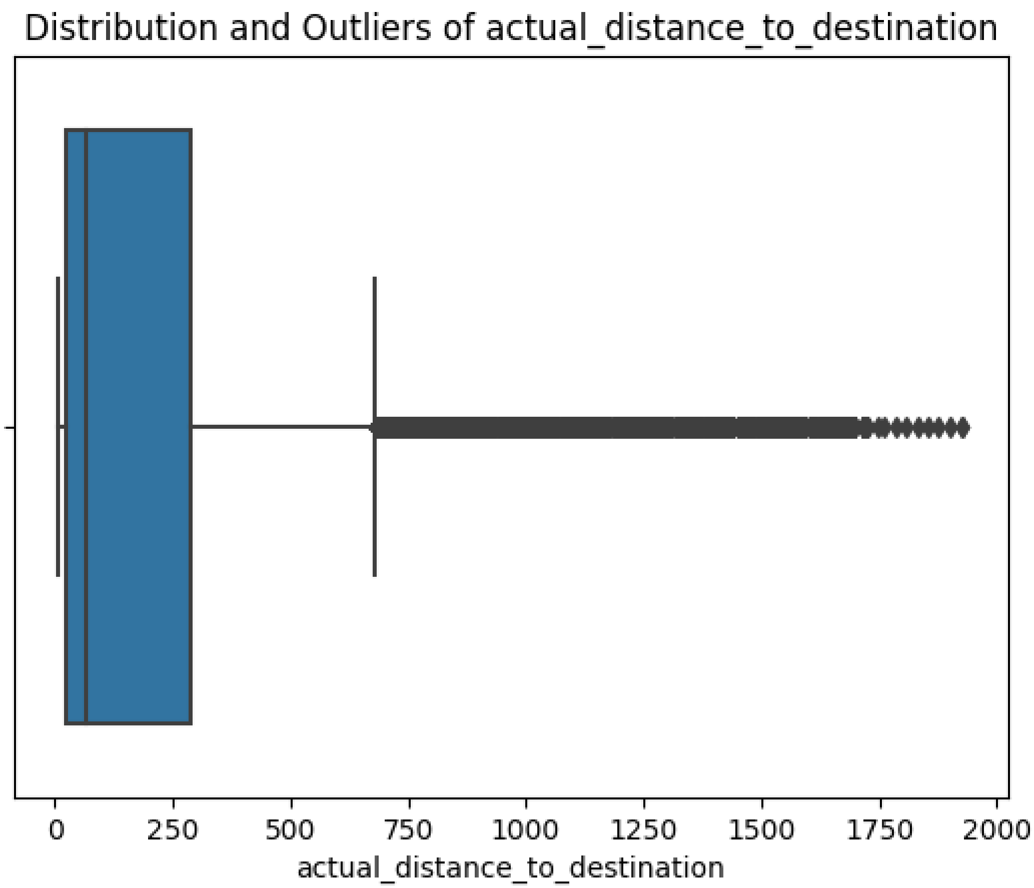



```
mi = min(dt['actual_time'])
ma = max(dt['actual_time'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = 9.0 and Maximum value = 4532.0
```

```
sns.boxplot(data = dt, x ='actual_distance_to_destination' )
plt.title('Distribution and Outliers of actual_distance_to_destination')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of actual_distance_to_destination')
```

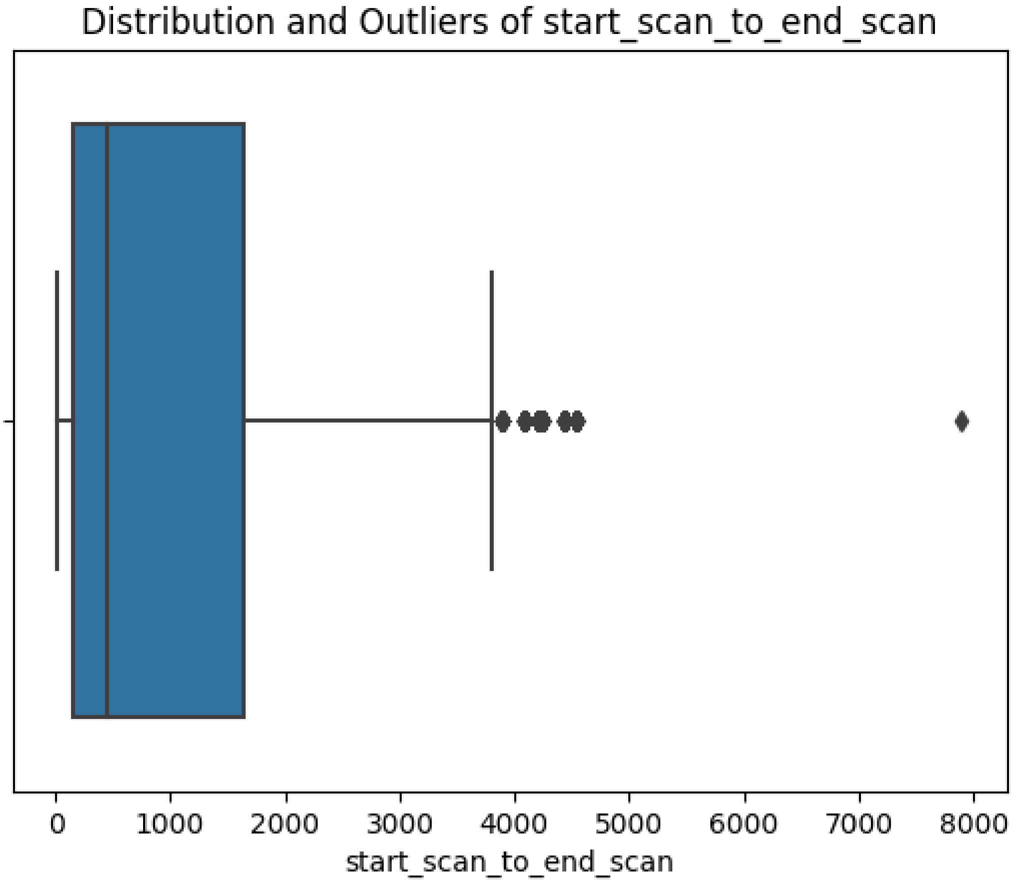



```
mi = min(dt['actual_distance_to_destination'])
ma = max(dt['actual_distance_to_destination'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
Minimum value = 9.00004535977208 and Maximum value = 1927.4477046975032
```

```
sns.boxplot(data = dt, x ='start_scan_to_end_scan' )
plt.title('Distribution and Outliers of start_scan_to_end_scan')
```

```
Text(0.5, 1.0, 'Distribution and Outliers of start_scan_to_end_scan')
```




```
mi = min(dt['actual_distance_to_destination'])
ma = max(dt['actual_distance_to_destination'])
print(f'Minimum value = {mi} and Maximum value = {ma}')
```

```
dt.columns
```

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

Extracting City, State and Place_Code

```
#dt[dt['trip_uuid'] == 'trip-153861115439069069']
```

```
dt['source_city'] = dt['source_name'].str.split('_').apply(lambda x:x[0].split()[0])
```

```
dt['source_state'] = dt['source_name'].str.split('_').apply(lambda x:x[-1]).apply(lambda x
```

```
dt['source_place-code'] = dt['source_name'].str.split('(').apply(lambda x:x[0]).str.split(
```

```
dt[['source_city', 'source_state', 'source_place-code']].value_counts().head(3)
```

```
source_city  source_state  source_place-code
Gurgaon      Haryana       Bilaspur_HB          23347
Bangalore    Karnataka     Nelmngla_H            9975
Bhiwandi     Maharashtra   Mankoli_HB            9088
dtype: int64
```

```
dt['destination_city'] = dt['destination_name'].str.split('_').apply(lambda x:x[0].split()
```

```
dt['destination_state'] = dt['destination_name'].str.split('_').apply(lambda x:x[-1]).appl
```

```
dt['destination_place-code'] = dt['destination_name'].str.split('(').apply(lambda x:x[0]).
```

```
dt[['destination_city', 'destination_state', 'destination_place-code']].value_counts().hea
```

```
destination_city  destination_state  destination_place-code
Gurgaon           Haryana            Bilaspur_HB               15192
Bangalore         Karnataka          Nelmngla_H                11019
Bhiwandi          Maharashtra        Mankoli_HB                 5492
dtype: int64
```

Extract features like month, year and day from trip_creation_time

```
dt['trip_creation_time'] = pd.to_datetime(dt['trip_creation_time'])
dt['trip_creation_time_month'] = dt['trip_creation_time'].dt.month
dt['trip_creation_time_day'] = dt['trip_creation_time'].dt.day
dt['trip_creation_time_year'] = dt['trip_creation_time'].dt.year
```

```
dt[['trip_creation_time', 'trip_creation_time_month', 'trip_creation_time_day', 'trip_crea
```

	trip_creation_time	trip_creation_time_month	trip_creation_time_day	trip_c
0	2018-09-20 02:35:36.476840	9	20	
1	2018-09-20 02:35:36.476840	9	20	

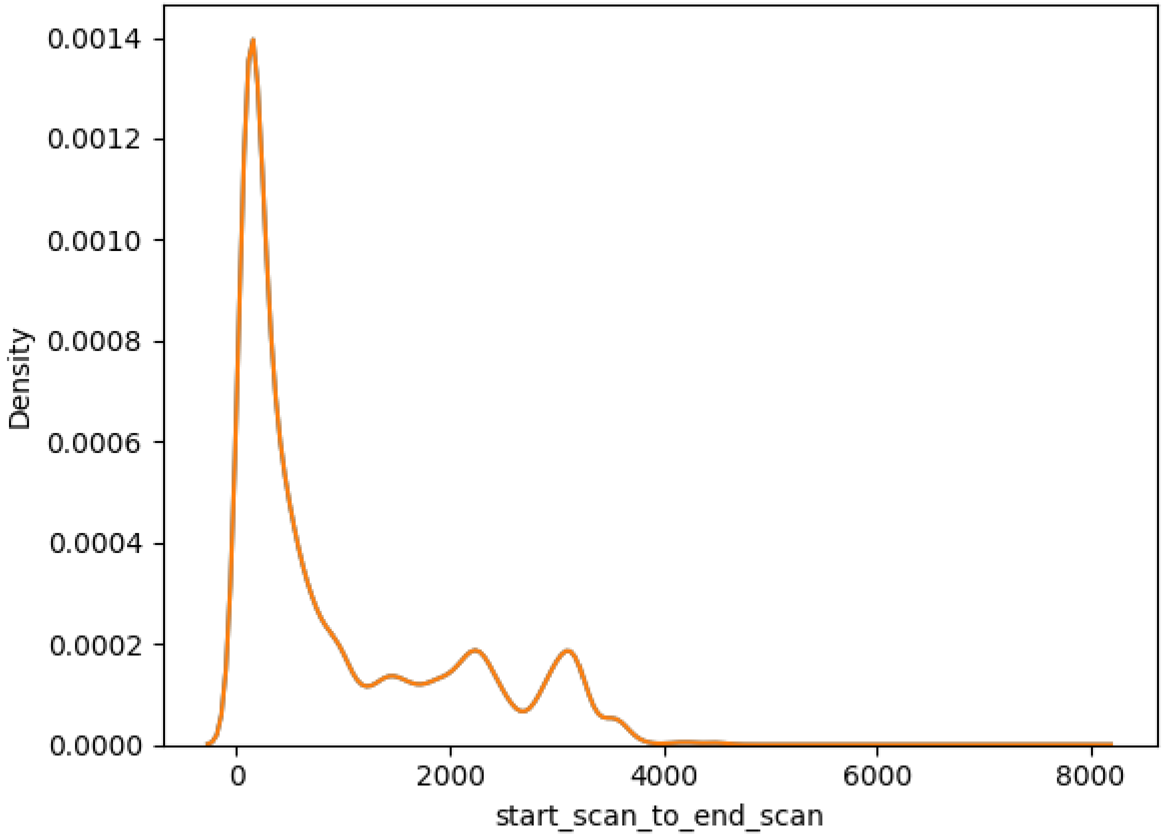
Time between od_start_time and od_end_time

```
dt['od_start_time'] = pd.to_datetime(dt['od_start_time'])
dt['od_end_time'] = pd.to_datetime(dt['od_end_time'])
```

```
dt['start_end_diff'] = (dt['od_end_time'] - dt['od_start_time']).dt.total_seconds()/60
```

Comparing start_scan_to_end_scan and start_end_diff (Newly formed feature)

```
sns.kdeplot(data = dt, x = 'start_scan_to_end_scan')
sns.kdeplot(data = dt, x = 'start_end_diff')
plt.show()
```



We do Hypothesis testing to Check difference between start_end_diff and start_scan_to_end_scan

- Null Hypothesis - There is no difference between start_end_diff and start_scan_to_end_scan
- Alternative Hypothesis - There is significant difference between start_end_diff and start_scan_to_end_scan

```
ttest_ind(dt['start_scan_to_end_scan'], dt['start_end_diff'], permutations = 100)
```

```
Ttest_indResult(statistic=-0.12873063959303033, pvalue=0.84)
```

Because P Value is greater than 0.05, we Fail to reject the Null Hypothesis

The difference between start_scan_to_end_scan and start_end_diff is not significant

```
# dropping route_schedule_uuid because does give much information
dt.drop('route_schedule_uuid', axis = 1, inplace = True)
```

Grouping the data with respect to Trip_uuid , Source_center and Destination center

```
# Dictionary for Aggregating the Values

groupby_dict = {

'data':'first',
'trip_creation_time' : 'first',
'route_type' : 'first',
'trip_uuid' : 'first',
'source_name' : 'first',
'destination_name' : 'last',
'od_start_time' : 'first',
'od_end_time' : 'last',
'start_scan_to_end_scan' : 'mean', # difference between od_start_time and od_end_time
'actual_distance_to_destination':'sum',
'actual_time':'last',# Since this is the Cumulative sum and the last value will give the s
'osrm_time' :'last',   # Since they are Cumulative sums, we only take last one
'osrm_distance':'last',  # Since they are Cumulative sums, we only take last one
'segment_actual_time' : 'sum',
'segment_osrm_time': 'sum',
'segment_osrm_distance':'sum',
'source_city':'first',
'source_state':'first',
'destination_city':'first',
'destination_state':'first',
'trip_creation_time_month': 'first',
'trip_creation_time_day':'first',
'trip_creation_time_year':'first',
'source_place-code' : 'first',
'destination_place-code': 'first'
   }
```

```
dt_grouped = dt.groupby(['trip_uuid', 'source_center', 'destination_center']).aggregate(gr
```

```
dt_grouped.head(2)
```

data	trip_creation_time	route_type	trip_uuid	source_name

Grouping the data with respect to Trip_uuid alone

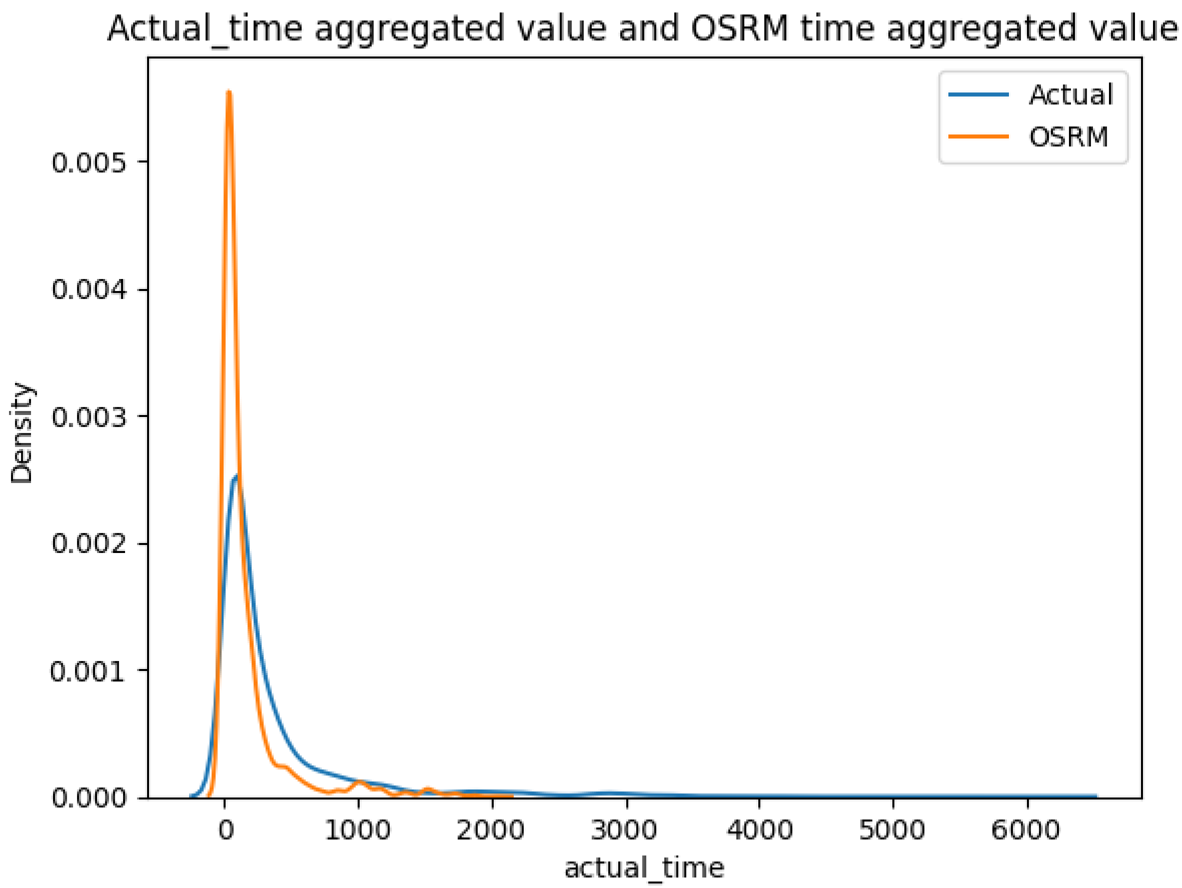
1	training	00:00:16.535741	FTL	153671041653548748	(Madhya Pradesh)

```
# Dictionary for Aggregating the Values

groupby_dict = {

'data':'first',
'trip_creation_time' : 'first',
'route_type' : 'first',
'trip_uuid' : 'first',
'source_name' : 'first',
'destination_name' : 'last',
'od_start_time' : 'first',
'od_end_time' : 'last',
'start_scan_to_end_scan' : 'sum', # difference between od_start_time and od_end_time
'actual_distance_to_destination':'sum',
'actual_time':'sum', # Now we will sum all cumulative time of subsets
'osrm_time' :'sum',   # We will sum all cumulative time of subsets
'osrm_distance':'sum',  # We will sum all cumulative distances of subsets
'segment_actual_time' : 'sum',
'segment_osrm_time': 'sum',
'segment_osrm_distance':'sum',
'source_city':'first',
'source_state':'first',
'destination_city':'first',
'destination_state':'first',
'trip_creation_time_month': 'first',
'trip_creation_time_day':'first',
'trip_creation_time_year':'first',
'source_place-code' : 'first',
'destination_place-code': 'first'
   }



dt_grouped = dt_grouped.groupby(['trip_uuid']).aggregate(groupby_dict).reset_index(drop =


dt_grouped.head()
```

	data	trip_creation_time	route_type	trip_uuid	source_name
0	training	2018-09-12 00:00:16.535741	FTL	trip-153671041653548748	Kanpur_Central_H_6 (Uttar Pradesh)
1	training	2018-09-12 00:00:22.886430	Carting	trip-153671042288605164	Doddablpur_ChikaDPP_D (Karnataka)
2	training	2018-09-12 00:01:00.113710	FTL	trip-153671046011330457	Gurgaon_Bilaspur_HB (Maharashtra)

Comparing actual_time aggregated value and OSRM time aggregated value

```
sns.kdeplot(data = dt_grouped, x = 'actual_time', label = 'Actual')
sns.kdeplot(data = dt_grouped, x = 'osrm_time', label = 'OSRM')
plt.legend()
plt.title('Actual_time aggregated value and OSRM time aggregated value')
plt.show()
```



We do Hypothesis testing to Check difference between Actual_time aggregated value and OSRM time aggregated value

- Null Hypothesis - There is no difference between Actual_time aggregated value and OSRM time aggregated value
- Alternative Hypothesis - There is significant difference Actual_time aggregated value and OSRM time aggregated value

```
ttest_ind(dt_grouped['actual_time'], dt_grouped['segment_osrm_time'], permutations = 100)
```

```
Ttest_indResult(statistic=33.32861975300905, pvalue=0.0)
```

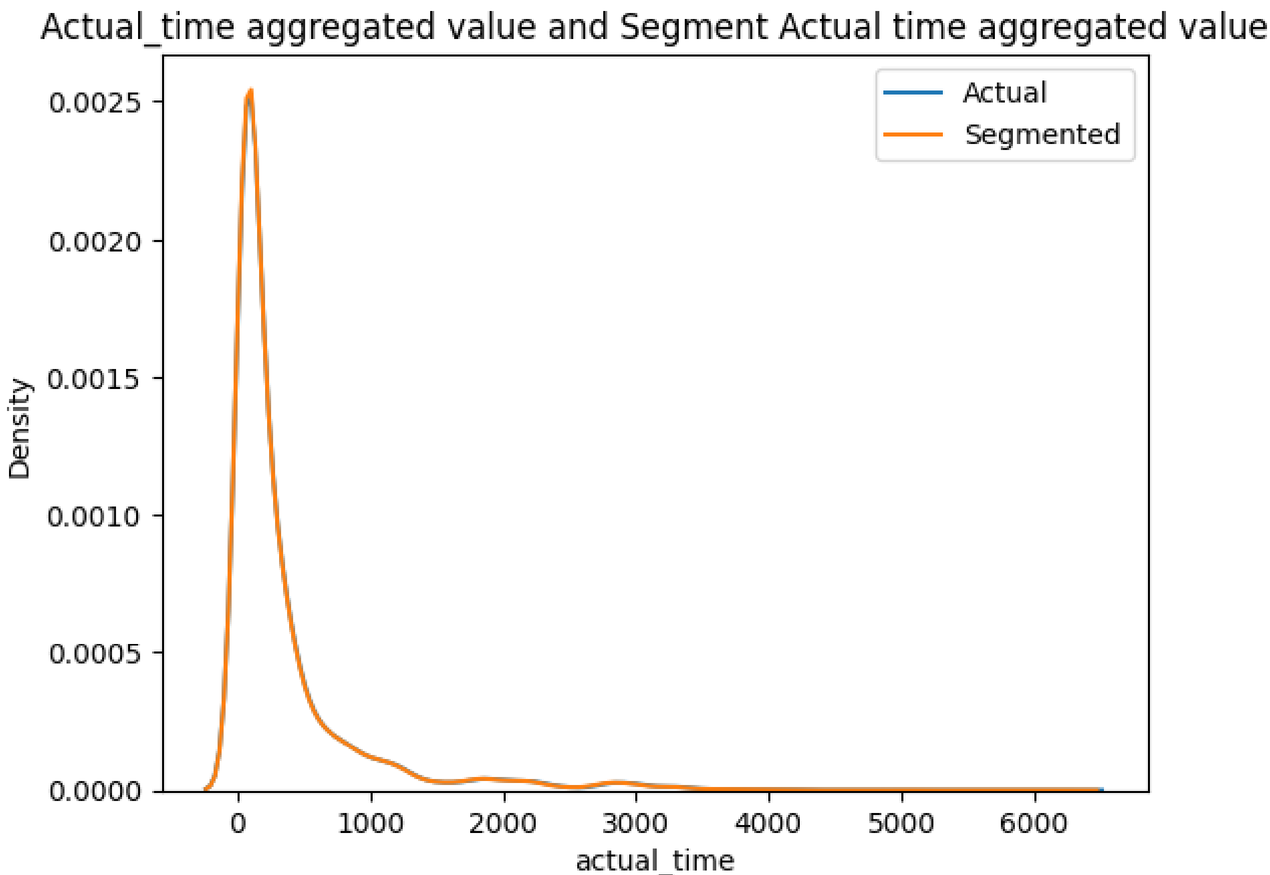
Actual Time > OSRM Time

Becuase P Value is less than 0.05, we reject the Null Hypothesis

We can conclude that the difference between Actual_time aggregated value and OSRM time aggregated is Significant

Comparing actual_time aggregated value and segment actual time aggregated value

```
sns.kdeplot(data = dt_grouped, x = 'actual_time', label = 'Actual')
sns.kdeplot(data = dt_grouped, x = 'segment_actual_time', label = 'Segmented')
plt.legend()
plt.title('Actual_time aggregated value and Segment Actual time aggregated value')
plt.show()
```



We do Hypothesis testing to Check difference between Actual_time aggregated value and Segment Actual time aggregated value

- Null Hypothesis - There is no difference between Actual_time aggregated value and Segment Actual time aggregated value
- Alternative Hypothesis - There is significant difference between Actual_time aggregated value and Segment Actual time aggregated value

```
ttest_ind(dt_grouped['actual_time'], dt_grouped['segment_actual_time'], permutations = 100
```

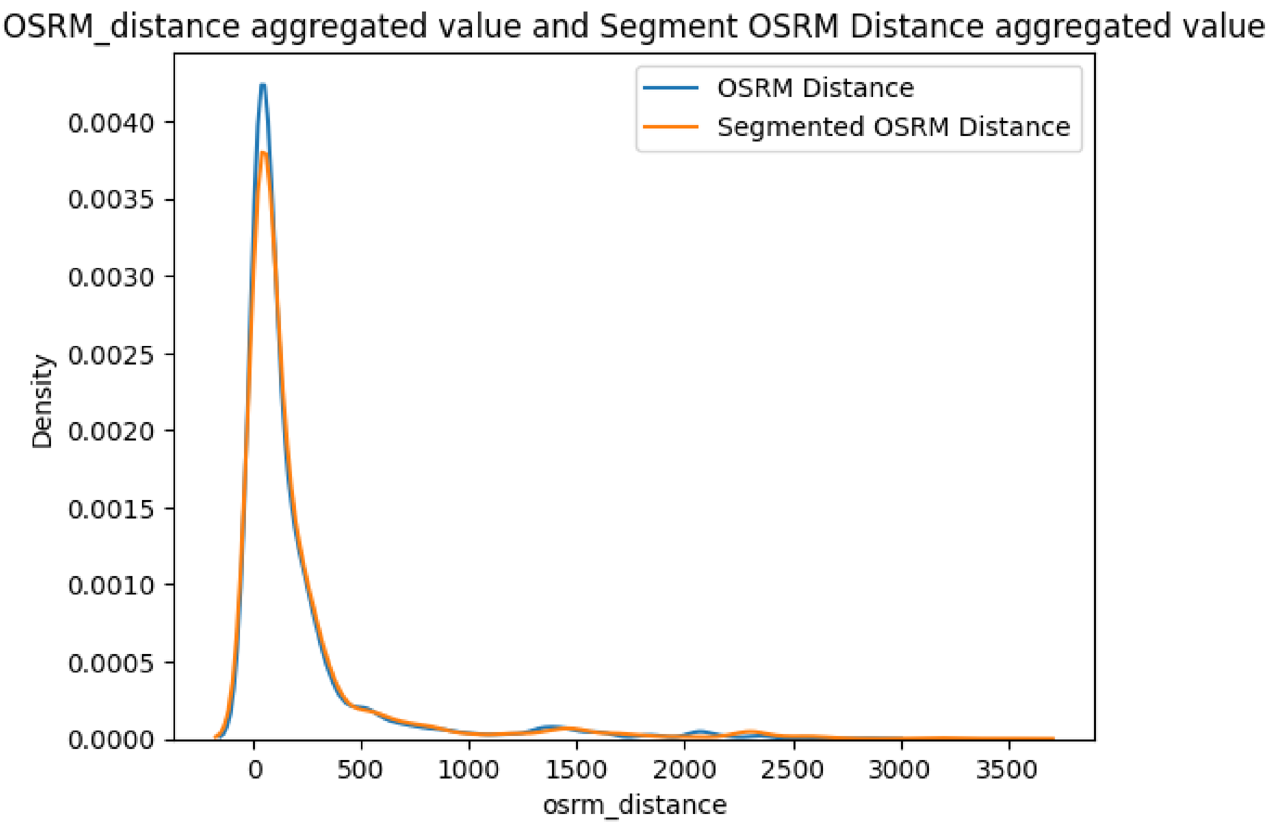
```
Ttest_indResult(statistic=0.5008024728897531, pvalue=0.6)
```

Becuase P Value is greater than 0.05, we fail to reject the Null Hypothesis

We can conclude that the difference between Actual_time aggregated value and Segment Actual time aggregated value is not Significant

osrm distance aggregated value and segment osrm distance aggregated value

```
sns.kdeplot(data = dt_grouped, x = 'osrm_distance', label = 'OSRM Distance')
sns.kdeplot(data = dt_grouped, x = 'segment_osrm_distance', label = 'Segmented OSRM Distan
plt.legend()
plt.title('OSRM_distance aggregated value and Segment OSRM Distance aggregated value')
plt.show()
```



We do Hypothesis testing to Check difference between OSRM_distance aggregated value and Segment OSRM Distance aggregated value

- Null Hypothesis - There is no difference between OSRM_distance aggregated value and Segment OSRM Distance aggregated value
- Alternative Hypothesis - There is significant difference OSRM_distance aggregated value and Segment OSRM Distance aggregated value

```
ttest_ind(dt_grouped['osrm_distance'], dt_grouped['segment_osrm_distance'], permutations =
```

```
Ttest_indResult(statistic=-4.117367046483823, pvalue=0.0)
```

Becuase P Value is lesser than 0.05, we reject the Null Hypothesis

We can conclude that the difference between OSRM_distance aggregated value and Segment OSRM Distance aggregated value is Significant

osrm time aggregated value and segment osrm time aggregated value

```
sns.kdeplot(data = dt_grouped, x = 'osrm_time', label = 'OSRM Time')
sns.kdeplot(data = dt_grouped, x = 'segment_osrm_time', label = 'Segmented OSRM Time')
plt.legend()
plt.title('OSRM_time aggregated value and Segment OSRM time aggregated value')
plt.show()
```

OSRM_time aggregated value and Segment OSRM time aggregated value

We do Hypothesis testing to Check difference between OSRM_time aggregated value and Segment OSRM time aggregated value

- Null Hypothesis - There is no difference between OSRM_time aggregated value and Segment OSRM time aggregated value
- Alternative Hypothesis - There is significant difference between OSRM_time aggregated value and Segment OSRM time aggregated value

```
ttest_ind(dt_grouped['osrm_time'], dt_grouped['segment_osrm_time'], permutations = 100)
```

```
Ttest_indResult(statistic=-5.733106696963521, pvalue=0.0)
```

Becuase P Value is lesser than 0.05, we reject the Null Hypothesis

We can conclude that the difference between two value is Significant.

Handling Categorical Values - One Hot Encoding

```
dt_grouped['route_Carting'] = pd.get_dummies(dt_grouped.route_type, prefix='route').loc[:,
dt_grouped['route_FTL'] = pd.get_dummies(dt_grouped.route_type, prefix='route').loc[:, 'ro
```

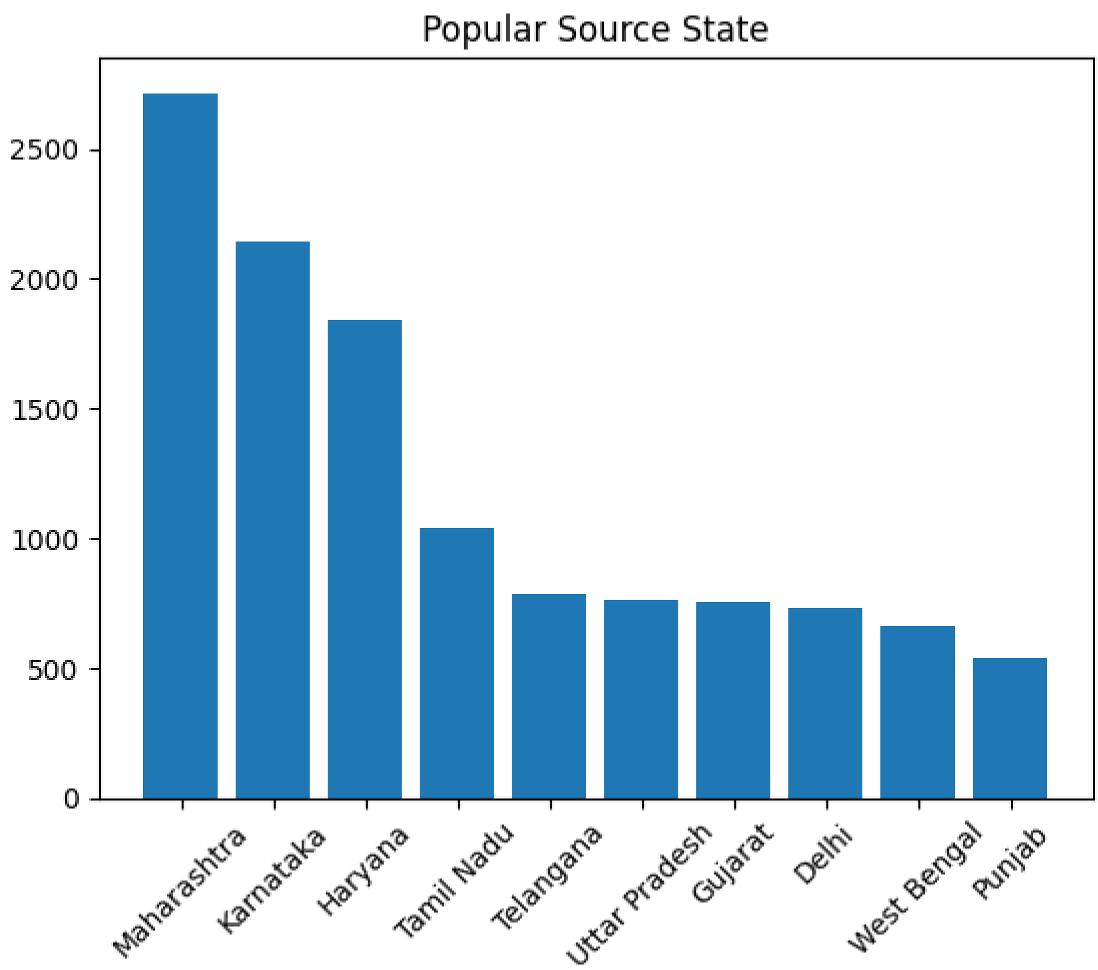
```
dt_grouped[['route_Carting', 'route_FTL']].head(2)
```

	route_Carting	route_FTL
0	0	1
1	1	0

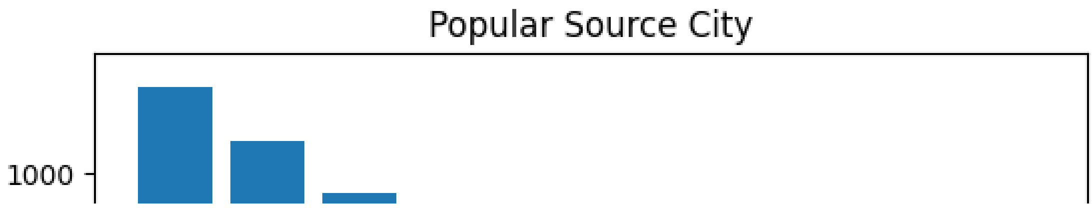
Similarly we can One Hot encode other Categorical Features like Day, Month , Year using the same above code

Visualizing Data

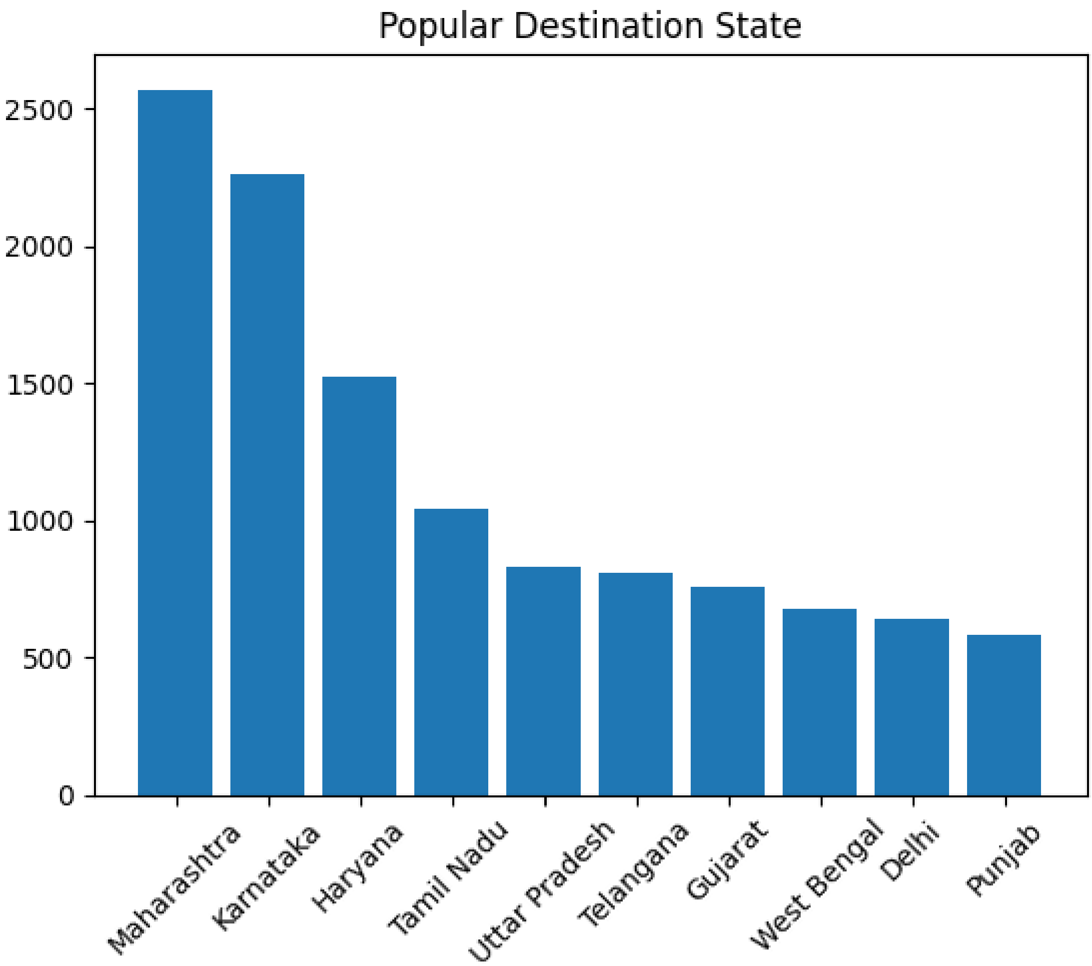
```
x = dt_grouped['source_state'].value_counts().iloc[:10].index
y = dt_grouped['source_state'].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 45)
plt.title('Popular Source State')
plt.show()
```



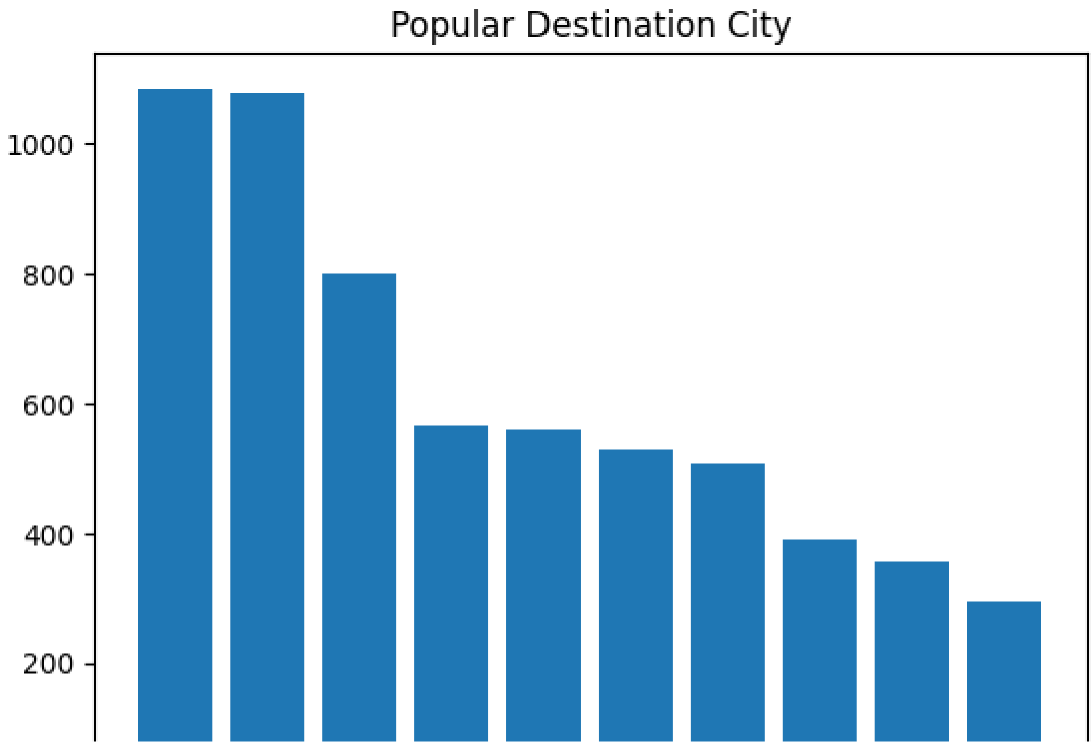

```
x = dt_grouped['source_city'].value_counts().iloc[:10].index
y = dt_grouped['source_city'].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 45)
plt.title('Popular Source City')
plt.show()
```



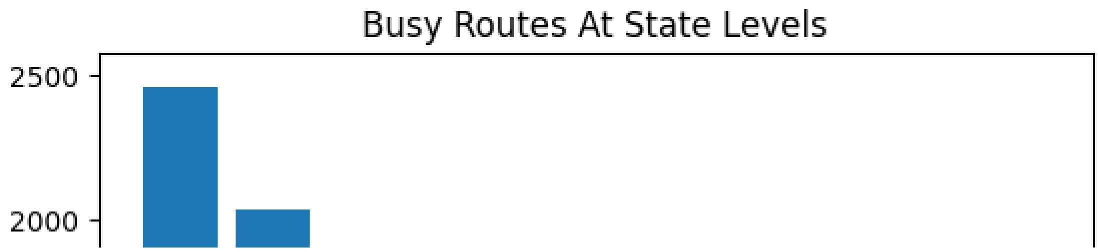

```
x = dt_grouped['destination_state'].value_counts().iloc[:10].index
y = dt_grouped['destination_state'].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 45)
plt.title('Popular Destination State')
plt.show()
```



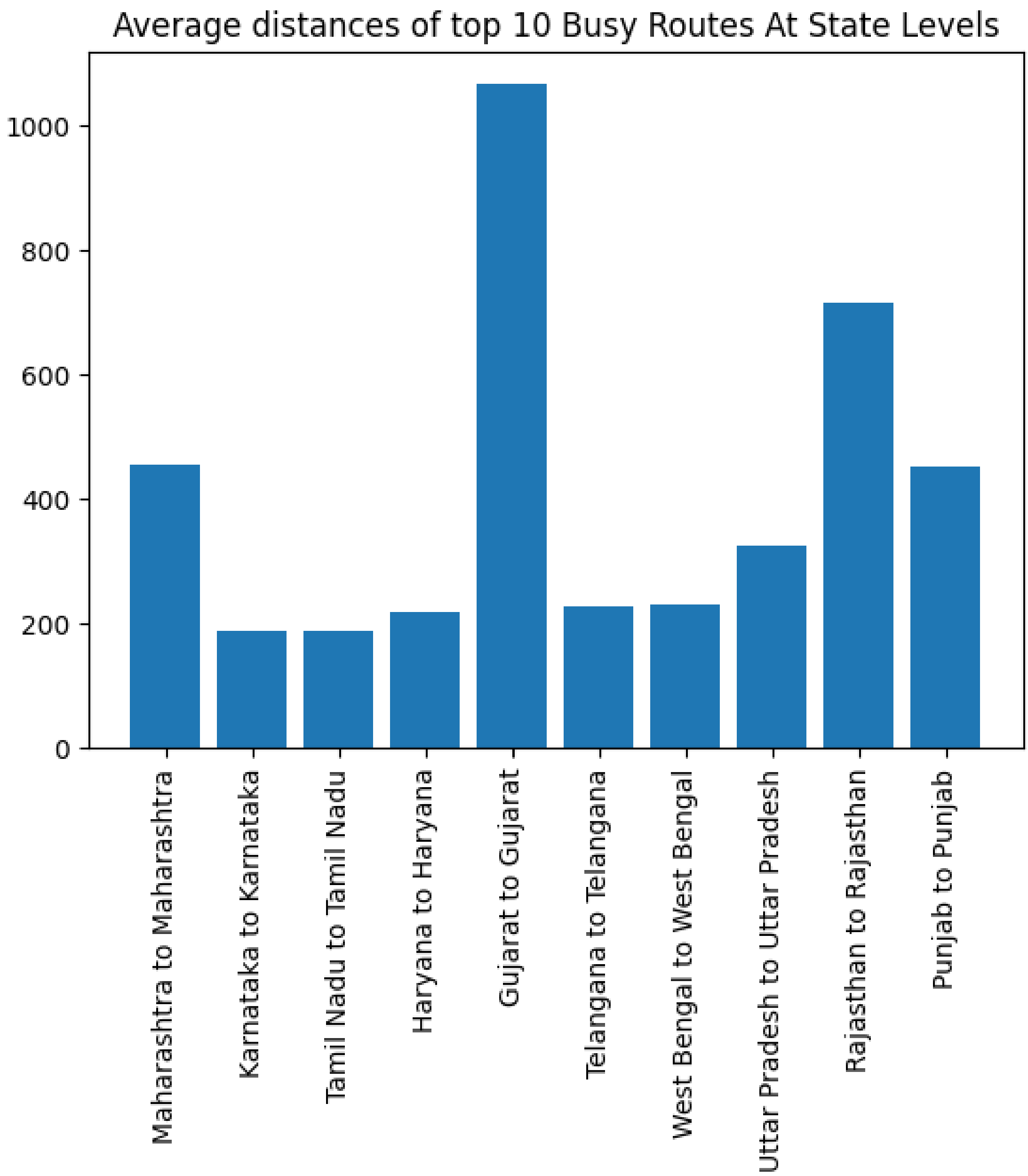

```
x = dt_grouped['destination_city'].value_counts().iloc[:10].index
y = dt_grouped['destination_city'].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 45)
plt.title('Popular Destination City')
plt.show()
```



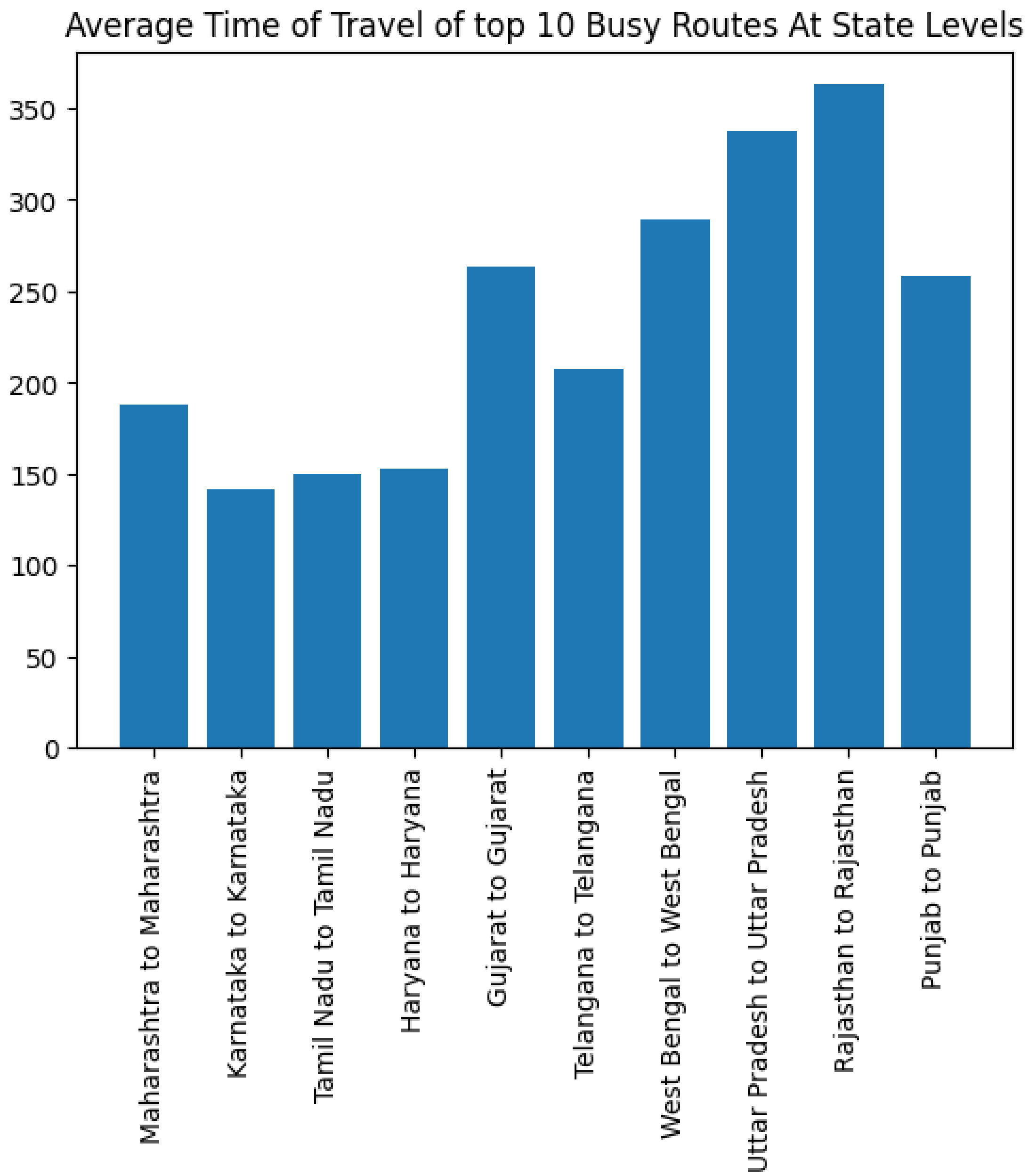

```
x = [i[0]+' to '+i[1] for i in dt_grouped[['source_state', 'destination_state']].value_cou
y = dt_grouped[['source_state', 'destination_state']].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Busy Routes At State Levels')
plt.show()
```



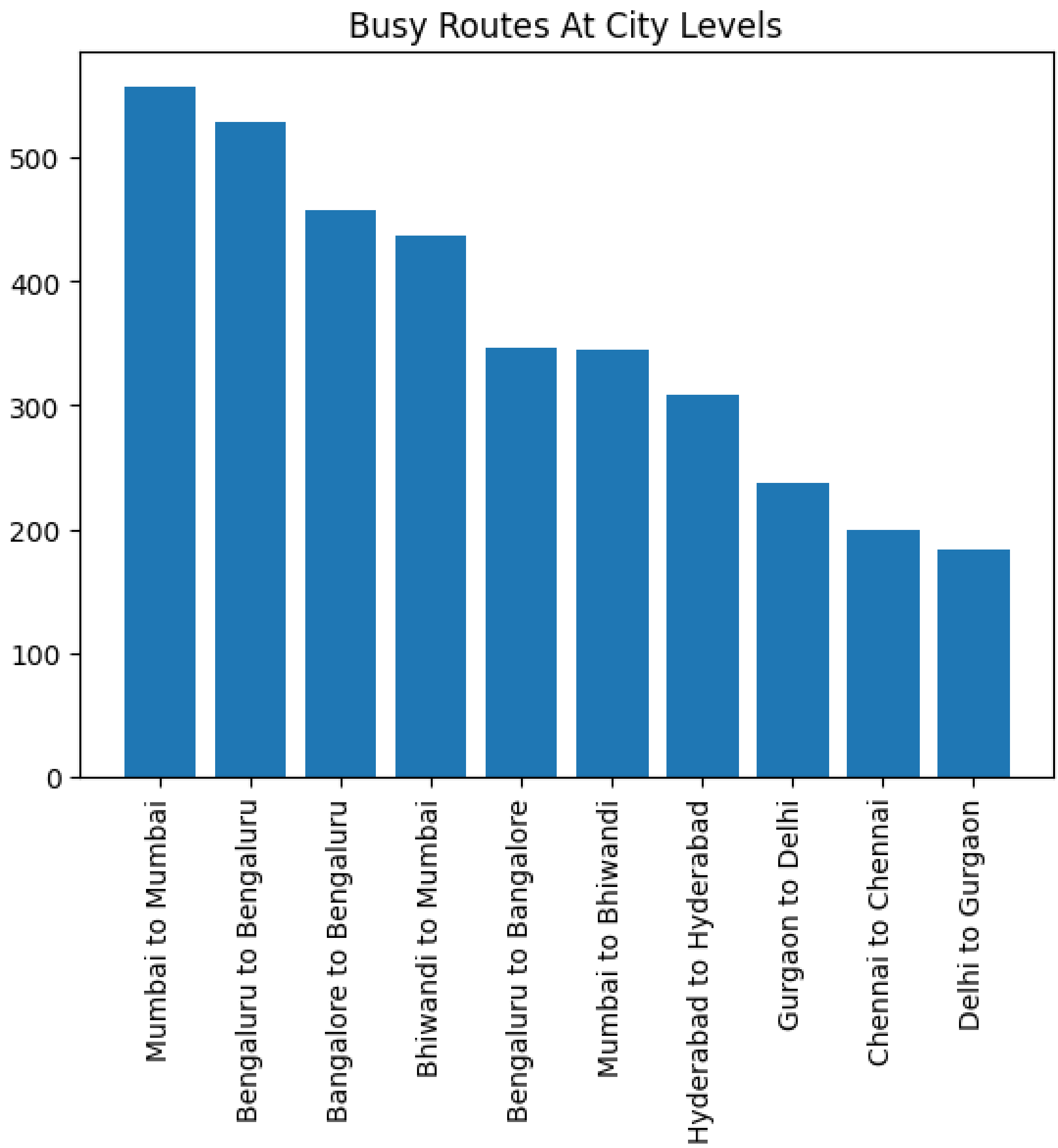

```
d = dt_grouped[['source_state', 'destination_state', 'actual_distance_to_destination']].gr
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_distance_to_destination_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_state', 'destination_state']].values[:10]]
y =  [i[0] for i in d[['actual_distance_to_destination_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average distances of top 10 Busy Routes At State Levels')
plt.show()
```



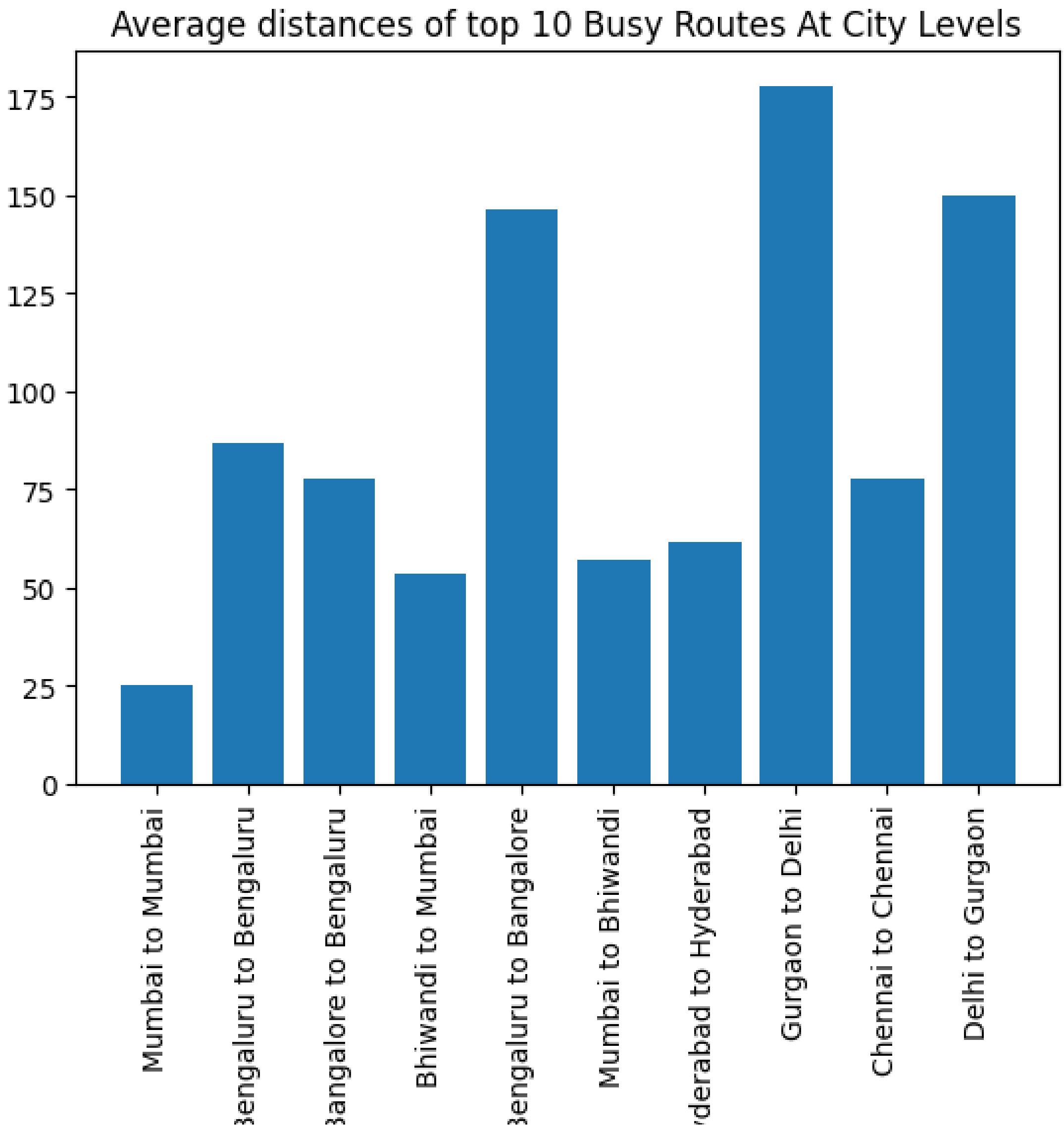
```
d = dt_grouped[['source_state', 'destination_state', 'actual_time']].groupby(['source_stat
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_time_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_state', 'destination_state']].values[:10]]
y =  [i[0] for i in d[['actual_time_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average Time of Travel of top 10 Busy Routes At State Levels')
plt.show()
```



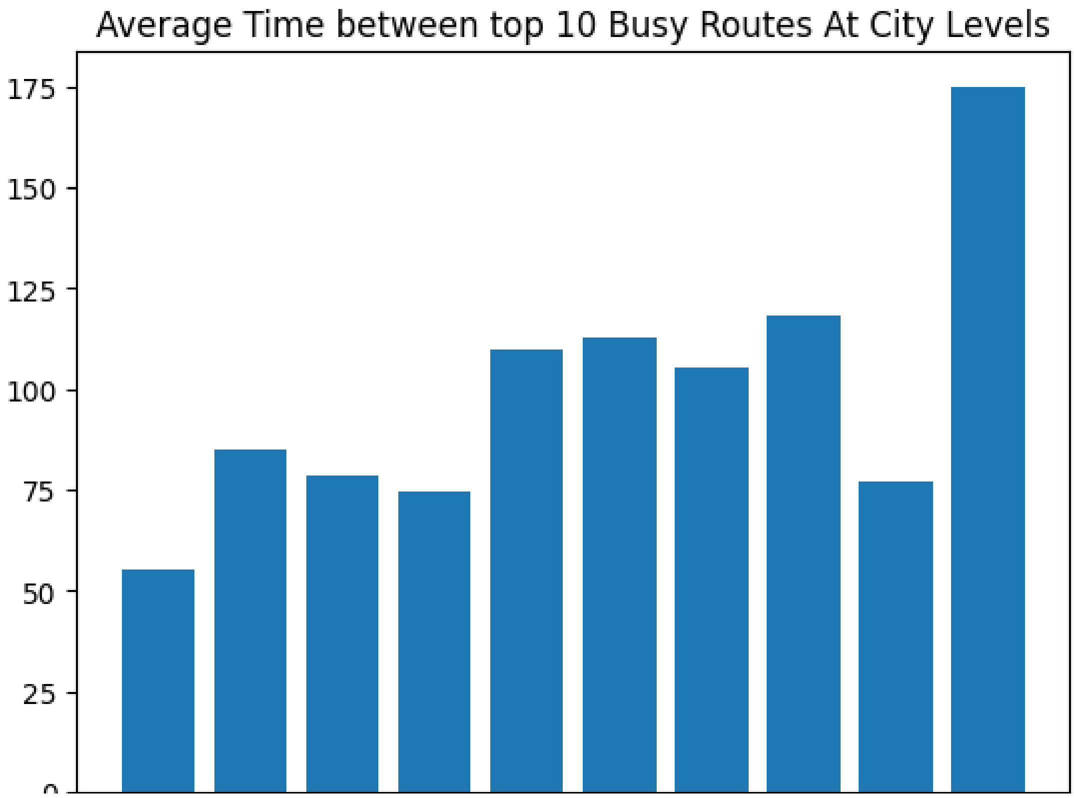
```
x = [i[0]+' to '+i[1] for i in dt_grouped[['source_city', 'destination_city']].value_count
y = dt_grouped[['source_city', 'destination_city']].value_counts().iloc[:10].values
plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Busy Routes At City Levels')
plt.show()
```



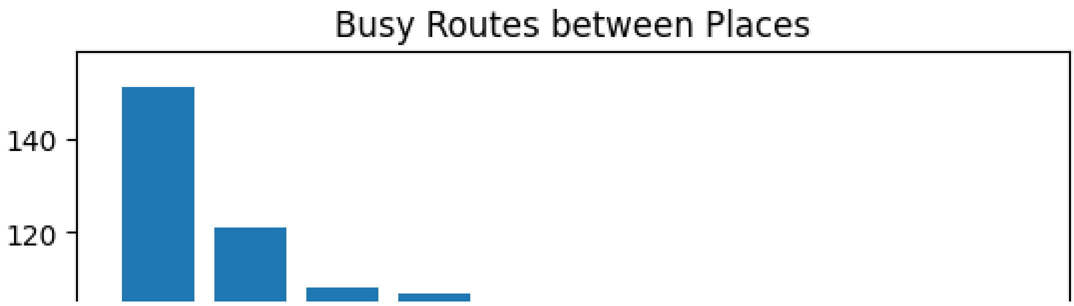

```
d = dt_grouped[['source_city', 'destination_city', 'actual_distance_to_destination']].grou
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_distance_to_destination_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_city', 'destination_city']].values[:10]]
y =  [i[0] for i in d[['actual_distance_to_destination_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average distances of top 10 Busy Routes At City Levels')
plt.show()
```




```
d = dt_grouped[['source_city', 'destination_city', 'actual_time']].groupby(['source_city',
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_time_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_city', 'destination_city']].values[:10]]
y =  [i[0] for i in d[['actual_time_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average Time between top 10 Busy Routes At City Levels')
plt.show()
```



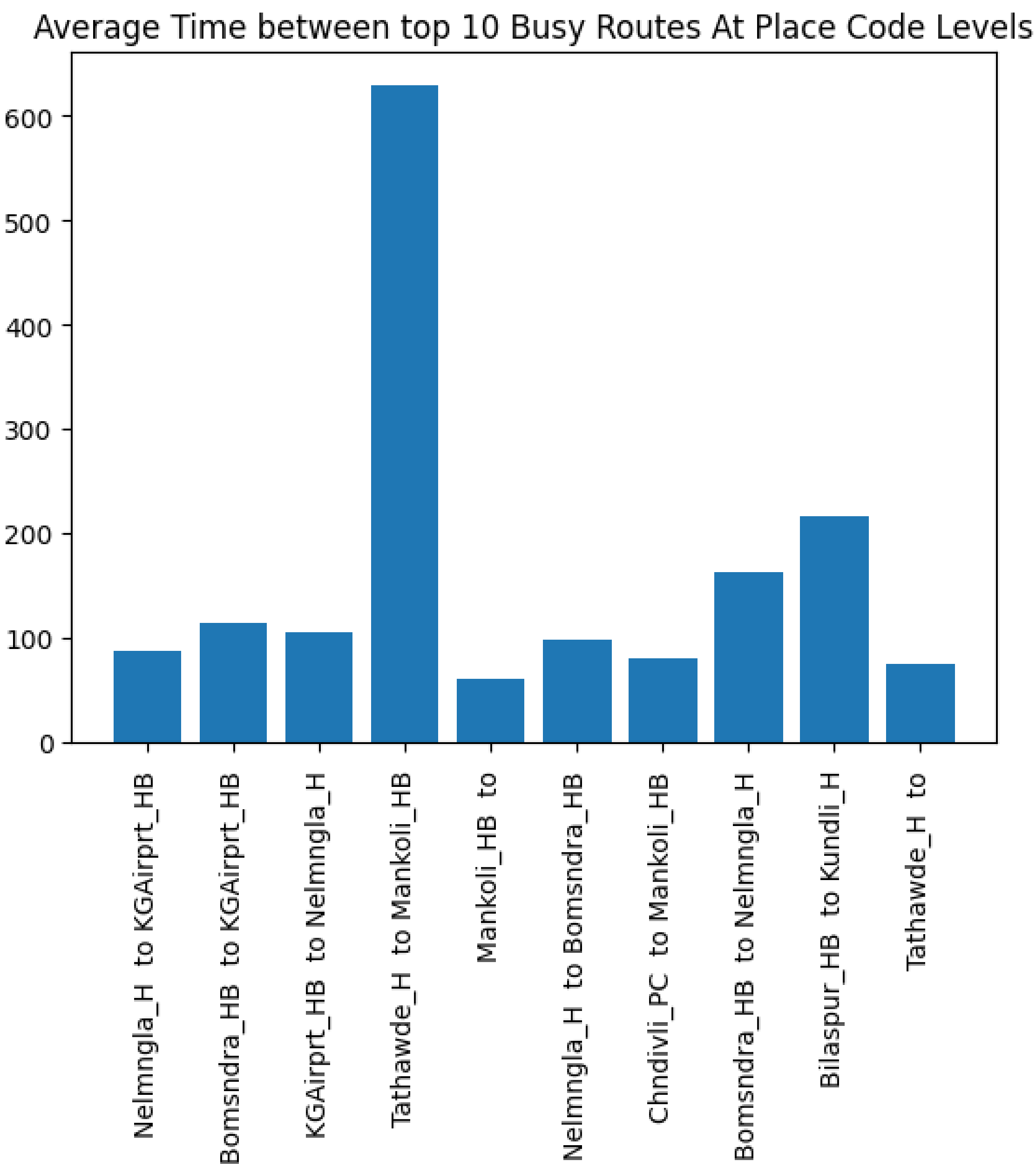

```
x = [i[0]+' to '+i[1] for i in dt_grouped[['source_place-code', 'destination_place-code']]
y = dt_grouped[['source_place-code', 'destination_place-code']].value_counts().iloc[:10].v
plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Busy Routes between Places')
plt.show()
```




```
d = dt_grouped[['source_place-code', 'destination_place-code', 'actual_distance_to_destina
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_distance_to_destination_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_place-code', 'destination_place-code']].values[
y =  [i[0] for i in d[['actual_distance_to_destination_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average distances between top 10 Busy Routes At Place Code Levels')
plt.show()
```

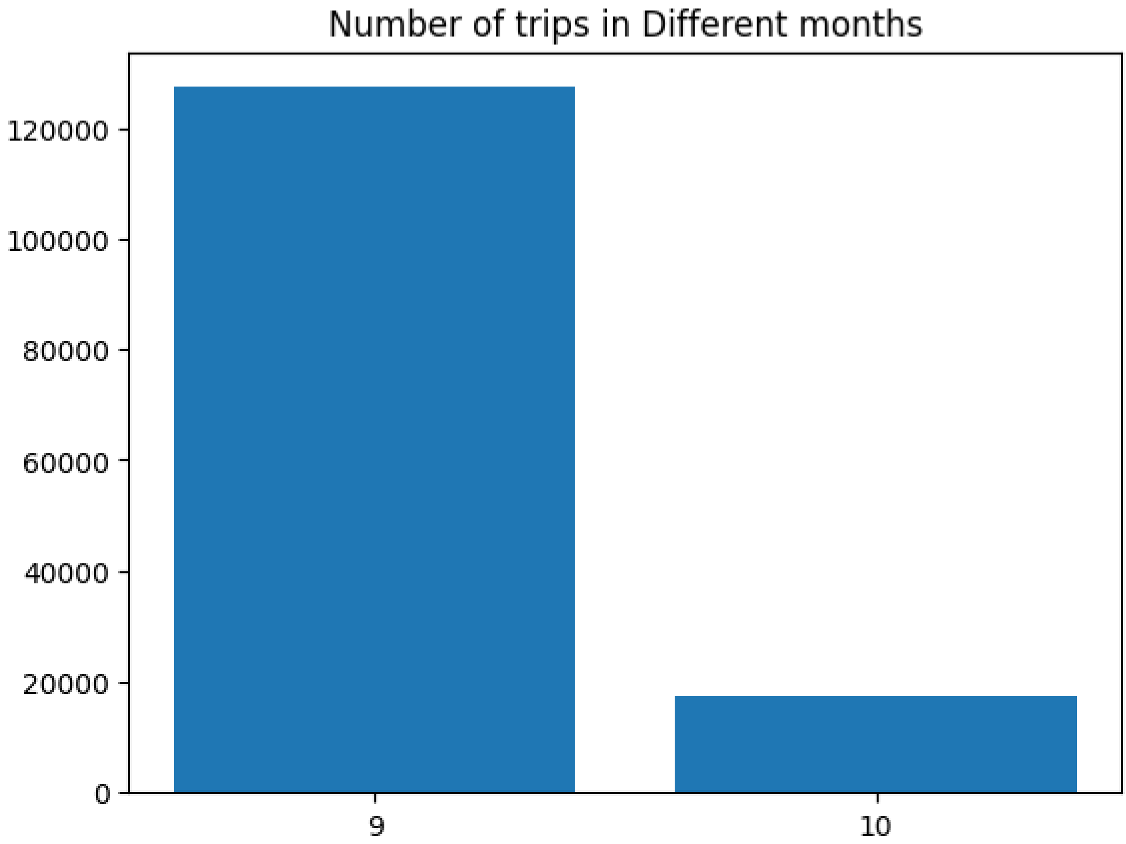
Average distances between top 10 Busy Routes At Place Code Levels

```
d = dt_grouped[['source_place-code', 'destination_place-code', 'actual_time']].groupby(['s
d.columns = ['_'.join(i) for i in d.columns]
d = d.reset_index()
d = d.sort_values(by = 'actual_time_count', ascending = False)

x =  [i[0]+' to '+i[1] for i in d[['source_place-code', 'destination_place-code']].values[
y =  [i[0] for i in d[['actual_time_mean']].values[:10]]

plt.bar(x,height=y)
plt.xticks(rotation = 90)
plt.title('Average Time between top 10 Busy Routes At Place Code Levels')
plt.show()
```




```
x = dt['trip_creation_time_month'].value_counts().index
y = dt['trip_creation_time_month'].value_counts().values
plt.bar(x,height=y)
plt.title('Number of trips in Different months')
```

```
plt.xticks( list(x))
plt.show()
```



MinMax Scaler

Before scaling the data, we will split it into Train and Test,

We will fit the method on Train data and transform both Train and Test Data.

Following are the Columns which we will Scale -

- start_scan_to_end_scan
- actual_distance_to_destination
- segment_osrm_distance
- actual_time
- osrm_time
- osrm_distance
- segment_actual_time
- segment_osrm_time

```
dt_grouped_train = dt_grouped[dt_grouped['data']=='training']
dt_grouped_test = dt_grouped[dt_grouped['data']=='test']

MMS = MinMaxScaler()
```

```
# Fitting the Standard Sclaer
MMS.fit(dt_grouped_train[['start_scan_to_end_scan', 'actual_distance_to_destination', 'seg
```

```
MinMaxScaler()
```

```
# Trasnforming the Data using MinMax Sclaer
dt_grouped_train.loc[:, ['start_scan_to_end_scan', 'actual_distance_to_destination', 'segm

dt_grouped_test.loc[:, ['start_scan_to_end_scan', 'actual_distance_to_destination', 'segme
```

```
dt_grouped_trained[['start_scan_to_end_scan', 'actual_distance_to_destination', 'segment_o
```

	start_scan_to_end_scan	actual_distance_to_destination	segment_osrm_distance	ac
0	0.382811	0.104014	0.373134	
1	0.026879	0.002717	0.021373	

Handling the outliers using the IQR method

- Here the Outliers are because of some locations which lie very far away.

```
def outlier(data):
    '''
    Function to Identify Outliers and Impute the Outliers using IQR Method
    '''
    for i in data:
        if (dt[i].dtype == np.float) | (dt[i].dtype == np.int):

            iqr = np.percentile(data[i].values, 75) - np.percentile(data[i].values, 25)
            upper_limit = np.percentile(data[i].values, 75) + iqr*1.5
            lower_limit = np.percentile(data[i].values, 25) - iqr*1.5

            data[i][(data[i]<lower_limit)] = lower_limit
            data[i][(data[i]>upper_limit)] = upper_limit


    return data
```

```
dt_cleaned = outlier(dt)
```

Business Insights

- Top 10 Popular Source States are - 'Maharashtra' , 'Karnataka', 'Haryana', 'Tamil Nadu', 'Telangana','Uttar Pradesh', 'Gujarat', 'Delhi', 'West Bengal', 'Punjab'

- Top 10 Popular Source Cities are - 'Gurgaon', 'Bengaluru', 'Mumbai', 'Bhiwandi', 'Bangalore', 'Delhi', 'Hyderabad', 'Pune', 'Chennai', 'Kolkata'
- Top 10 Popular Destination States are - 'Maharashtra', 'Karnataka', 'Haryana', 'Tamil Nadu', 'Uttar Pradesh', 'Telangana', 'Gujarat', 'West Bengal', 'Delhi', 'Punjab'
- Top 10 Popular Destination Cities are - 'Mumbai', 'Bengaluru', 'Gurgaon', 'Bhiwandi', 'Bangalore', 'Delhi', 'Hyderabad', 'Chennai', 'Chandigarh', 'Pune'
- Top 10 Busiest Routes at State level - 'Maharashtra to Maharashtra', 'Karnataka to Karnataka', 'Tamil Nadu to Tamil Nadu', 'Haryana to Haryana', 'Gujarat to Gujarat', 'Telangana to Telangana', 'West Bengal to West Bengal', 'Uttar Pradesh to Uttar Pradesh', 'Rajasthan to Rajasthan', 'Punjab to Punjab'
- Top 10 Busiest Routes at City Level - 'Mumbai to Mumbai', 'Bengaluru to Bengaluru', 'Bangalore to Bengaluru', 'Bhiwandi to Mumbai', 'Bengaluru to Bangalore', 'Mumbai to Bhiwandi', 'Hyderabad to Hyderabad', 'Gurgaon to Delhi', 'Chennai to Chennai','Delhi to Gurgaon'
- Top Busiest Routes between places - 'Nelmngla_H to KGAirprt_HB ', 'Bomsndra_HB to KGAirprt_HB ', 'KGAirprt_HB to Nelmngla_H ', 'Tathawde_H to Mankoli_HB ', 'Nelmngla_H to Bomsndra_HB ', 'Chndivli_PC to Mankoli_HB ', 'Bomsndra_HB to Nelmngla_H ', 'Bilaspur_HB to Kundli_H '.

- Among the Busiest routes at City level, Gurgaon to Delhi has the highest average distance.
- Among the Busiest routes between Places, Tathawade_H to Mankoli_HB is has the heighest avegare distance.

The below points may lead to mis calculations -

- The Actual Time and OSRM time have significant difference between them.
- The difference between OSRM distance and OSRM Segmented Distance is also Significant
- Similarly the difference between OSRM time and OSRM time aggregated is significant.

- Among the Busiest routes at State Level, Rajasthan to Rajastha is most time taking followed by Uttar Pradesh to Uttar Pradesh.
- Among the Busiest routes at City level, Delhi to Gurgaon is most Time taking trip.
- Among the Busiest routes at Place level, Tathawde_H to Mankoli_HB is the most time taking trip.

Recomendations -

- The difference between Actual Time and OSRM is Significant. This can cause a very major impact on the comapny. Becuase the OSRM wrongly predicts the estimated time, the customers might get a wrong estimate, and might receive their packages with delay.

- This wrong estimation of time might also result in disruptions of Operations.
- OSRM_distance aggregated value and Segment OSRM Distance AND OSRM_time aggregated value and Segment OSRM time aggregated value have significant difference. These may lead to a major issues in the operations.
- Mumbai, Banglore and Gurgaon are the Cities which has Most number of delivery services, hence its important to make sure that proper facilities and work force is always there to handle it.
- Its also evident that the routes within Mahrashtra, Karnataka and Tamil Nadu are one of the most busy paths. Its important to know good number of back up paths to move between Source and Destination.
- When it comes to time of travel, travels within Rajasthan, Uttarpradesh and West Bengal is more time taking. Hence its important to optimise these travels, like travelling during less Traffic . This is will save Fuel and Time.