```
#Downloading the dataset and storing it
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv
```

```
--2023-03-18 14:38:16--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.65.40.200, 18.65.40.33, 18.65.40.189, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.65.40.200|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: 'delhivery_data.csv.15'

delhivery_data.csv. 100%[===================>]  53.04M   121MB/s    in 0.4s

2023-03-18 14:38:17 (121 MB/s) - 'delhivery_data.csv.15' saved [55617130/55617130]
```

```
#Importing all the required packages
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.stats import ttest_ind, pearsonr
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import re
import warnings
```

```
#to ignore the warnings
warnings.filterwarnings("ignore")
```

```
#Store the dataset in the dataframe
df = pd.read_csv("delhivery_data.csv")
```

```
#View first 5 records using the head command
df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time	...	cutoff_timestamp	actual_distance_to_destination	actual_time	osrm_time	osrm
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:27:55	10.435660	14.0	11.0	
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:17:55	18.936842	24.0	20.0	
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:01:19.505586	27.637279	40.0	28.0	
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:39:57	36.118028	62.0	40.0	
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:33:55	39.386040	68.0	44.0	

5 rows × 24 columns

Provided Columns and thier description

1. data - tells whether the data is testing or training data
2. trip_creation_time – Timestamp of trip creation
3. route_schedule_uuid – Unique Id for a particular route schedule
4. route_type – Transportation type
5. 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
6. Carting: Handling system consisting of small vehicles (carts)
7. trip_uuid - Unique ID given to a particular trip (A trip may include . different source and destination centers)
8. source_center - Source ID of trip origin
9. source_name - Source Name of trip origin
10. destination_cente – Destination ID
11. destination_name – Destination Name
12. od_start_time – Trip start time
13. od_end_time – Trip end time
14. start_scan_to_end_scan – Time taken to deliver from source to destination
15. is_cutoff – Unknown field
16. cutoff_factor – Unknown field
17. cutoff_timestamp – Unknown field
18. actual_distance_to_destination – Distance in Kms between source and destination warehouse
19. actual_time – Actual time taken to complete the delivery (Cumulative)
20. 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)
21. 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)
22. factor – Unknown field
23. segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
24. segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
25. segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
26. segment_factor – Unknown field

```
#Drop the columns which are not clear and which are unknown
df.drop(["is_cutoff","cutoff_factor","cutoff_timestamp","factor","segment_factor"], axis=1, inplace=True)
```

```
df.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')
```

```
#After droping the unwanted columns, the dataframe consists of 19 columns and 144867 rows
df.shape
```

```
(144867, 19)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 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  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

```
#Checking for the null values using the isna() function
df.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
actual_distance_to_destination      0
actual_time                         0
osrm_time                           0
osrm_distance                       0
segment_actual_time                 0
segment_osrm_time                   0
segment_osrm_distance               0
dtype: int64
```

293 null values found in source_name column and 261 null values found in destination_name columns

```
#Droping the null values
df.dropna(inplace=True)
```

There are 144867 rows and out of which 554 rows contains null values, which is 0.3%, hence we can delete those entires

```
df.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
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
dtype: int64
```

```
df.shape
```

```
(144316, 19)
```

```
#Checking for any duplicate entires
df.duplicated().any()
```

```
False
```

There are no duplicate entires found

```
#The describe function used to show the Mean, Median, percentile, min and max values of all numerical columns
df.describe()
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance
count	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000	144316.000000
mean	963.697698	234.708498	417.996237	214.437055	285.549785	36.175379	18.495697	22.818993
std	1038.082976	345.480571	598.940065	308.448543	421.717826	53.524298	14.774008	17.866367
min	20.000000	9.000045	9.000000	6.000000	9.008200	-244.000000	0.000000	0.000000
25%	161.000000	23.352027	51.000000	27.000000	29.896250	20.000000	11.000000	12.053975
50%	451.000000	66.135322	132.000000	64.000000	78.624400	28.000000	17.000000	23.508300

```
#The describe function with include="object", shows the count, unique and freq of the columns whose type is object.
df.describe(include="object")
```

	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
count	144316	144316	144316	144316	144316	144316	144316	144316	144316	144316	144316
unique	2	14787	1497	2	14787	1496	1496	1466	1466	26223	26223
top	training	2018-10-01 05:04:55.268931	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153837029526866991	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-21 18:37:09.322207	2018-09-24 09:59:15.691618
freq	104632	101	1812	99132	101	23267	23267	15192	15192	81	81

```
#Converting the datatype from object to datetime
df['od_start_time'] =  pd.to_datetime(df['od_start_time'])
df['od_end_time'] =  pd.to_datetime(df['od_end_time'])
df['trip_creation_time'] =  pd.to_datetime(df['trip_creation_time'])
```

```
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  datetime64[ns]
 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  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 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: datetime64[ns](3), float64(8), object(8)
memory usage: 22.0+ MB
```

```
#Calculating the time taken between strat time and the end time and storing it as od_total_time
df["od_total_time"] = (df.od_end_time-df.od_start_time).astype('timedelta64[m]')
```

```
df.head()
```

	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
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	86.0	10.435660	14.0
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	86.0	18.936842	24.0
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	86.0	27.637279	40.0
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	86.0	36.118028	62.0
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	86.0	39.386040	68.0

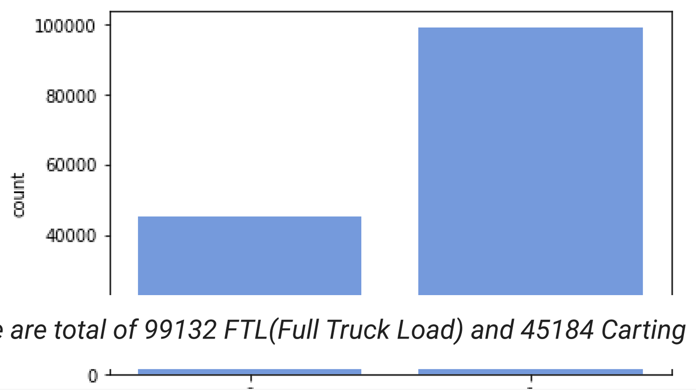
```
#Performing one hot encoding for the categorical column route_type
print(df["route_type"].value_counts())
label_encoder = LabelEncoder()
df["route_type"] = label_encoder.fit_transform(df["route_type"])
print(df["route_type"].value_counts())
```

```
FTL        99132
Carting    45184
Name: route_type, dtype: int64
1    99132
0    45184
Name: route_type, dtype: int64
```

After one hot encoding, FTL is marked as 1 and Carting is marked as 0

```
sns.countplot(x = 'route_type', data = df, color='cornflowerblue')
```

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



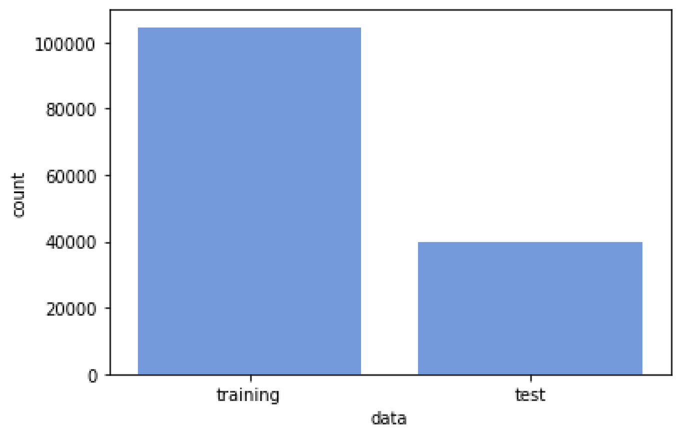
There are total of 99132 FTL(Full Truck Load) and 45184 Carting

```
print(df["data"].value_counts())
```

```
training    104632
test         39684
Name: data, dtype: int64
```

```
sns.countplot(x = 'data', data = df, color='cornflowerblue')
```

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



There are total of 104632 Training data and 39684 Testing data

```
df[["temp_source","Soruce_State"]]=df['source_name'].str.split('(',expand=True)
df['Soruce_State'] = df['Soruce_State'].str.replace(")","")
df[["Source_City", "Soruce_Place", "Soruce_Code"]] = df['temp_source'].str.split('_',expand=True).drop(columns=[3])
df[["temp_destination","Destination_State"]]=df['destination_name'].str.split('(',expand=True)
df['Destination_State'] = df['Destination_State'].str.replace(")","")
df[["Destination_City", "Destination_Place", "Destination_Code"]] = df['temp_destination'].str.split('_',expand=True).drop(columns=[3])
df.drop(["temp_destination","temp_source"], axis=1, inplace=True)
```

- *Splitting the source_name to Soruce_State, Source_City, Soruce_Place, Soruce_Code and storing it in each columns*
- *Splitting the destination_name to Destination_State, Destination_City, Destination_Place, Destination_Code and storing it in each columns*

```
df['trip_year'] = df['trip_creation_time'].dt.year
df['trip_month'] = df['trip_creation_time'].dt.month
df['trip_day'] = df['trip_creation_time'].dt.day
```

Splitting the trip_creation_time to trip_year, trip_month, trip_day and storing it in each columns

```
df["source_to_destination_state"] = df['Soruce_State']+" to "+df ['Destination_State']
df["source_to_destination_city"] = df['Source_City']+" to "+df ['Destination_City']
df["source_to_destination_place"] = df['Soruce_Place']+" to "+df ['Destination_Place']
```

Merging each Source and Destination State, City and Place and sotring it in each columns

```
df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time	...	Destination_State	Destination_City	Destination_Place	Destination_Code	trip
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	0	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	Gujarat	Khambhat	MotvdDPP	D	
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	0	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	Gujarat	Khambhat	MotvdDPP	D	
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	0	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	Gujarat	Khambhat	MotvdDPP	D	
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	0	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	Gujarat	Khambhat	MotvdDPP	D	
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	0	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	Gujarat	Khambhat	MotvdDPP	D	

5 rows × 34 columns

```
df[["Source_City", "Soruce_Place", "Soruce_Code","Soruce_State","Destination_City", "Destination_Place", "Destination_Code","Destination_State","source_to_destination_state","source_to_destination_city","source_to_destination_place"]].describe()
```

	Source_City	Soruce_Place	Soruce_Code	Soruce_State	Destination_City	Destination_Place	Destination_Code	Destination_State	source_to_destination_state	source_to_destination_city	source_to_destination_place
count	144316	142218	129688	144316	144316	141884	128883	144316	144316	144316	139818
unique	1260	1153	24	31	1256	1130	27	32	155	2355	2377
top	Gurgaon	Bilaspur	HB	Haryana	Gurgaon	Bilaspur	H	Karnataka	Maharashtra to Maharashtra	Gurgaon to Bangalore	Bilaspur to Nelmngla
freq	23585	23384	41097	27408	15393	15363	34572	21049	11876	4976	4976

- There are 31 unique Source State
- There are 32 unique Destination State
- There are 155 unique source_to_destination_state
- There are 2355 unique source_to_destination_state
- There are 2377 unique source_to_destination_state

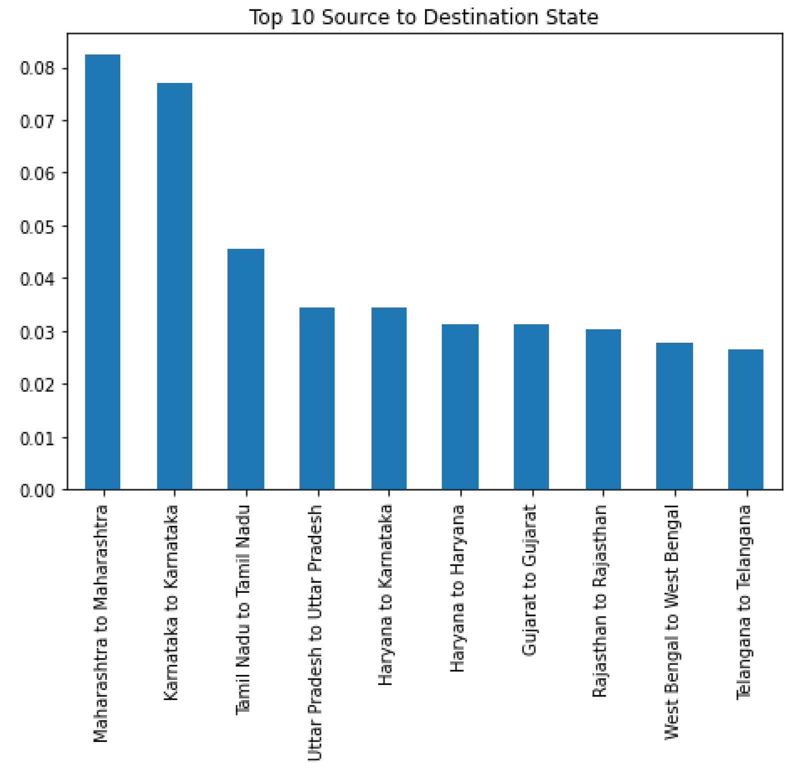
```
df[["trip_year","trip_month","trip_day"]].describe()
```

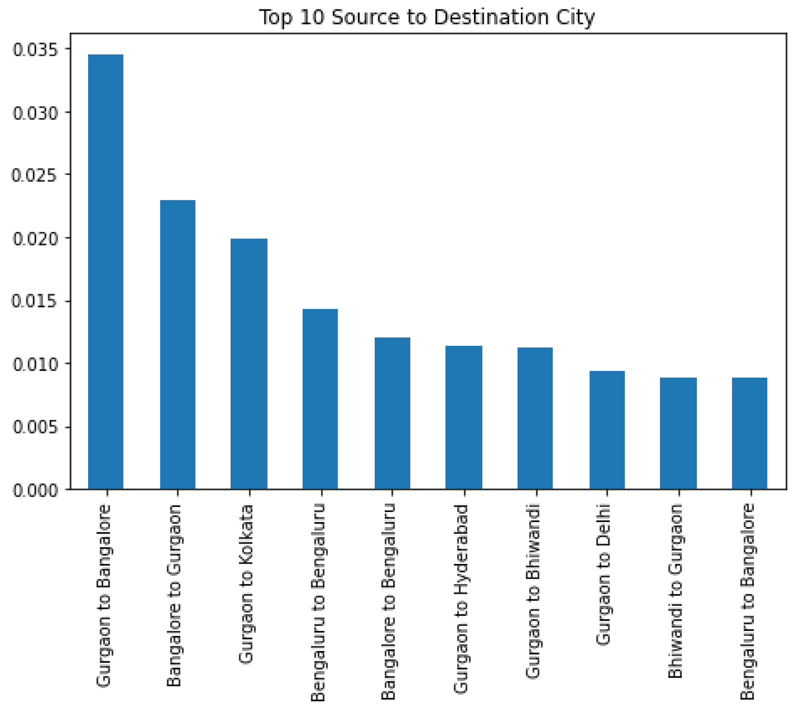
	trip_year	trip_month	trip_day
count	144316.0	144316.000000	144316.000000
mean	2018.0	9.120458	18.383111
std	0.0	0.325497	7.865996
min	2018.0	9.000000	1.000000
25%	2018.0	9.000000	14.000000
50%	2018.0	9.000000	19.000000
75%	2018.0	9.000000	25.000000
max	2018.0	10.000000	30.000000

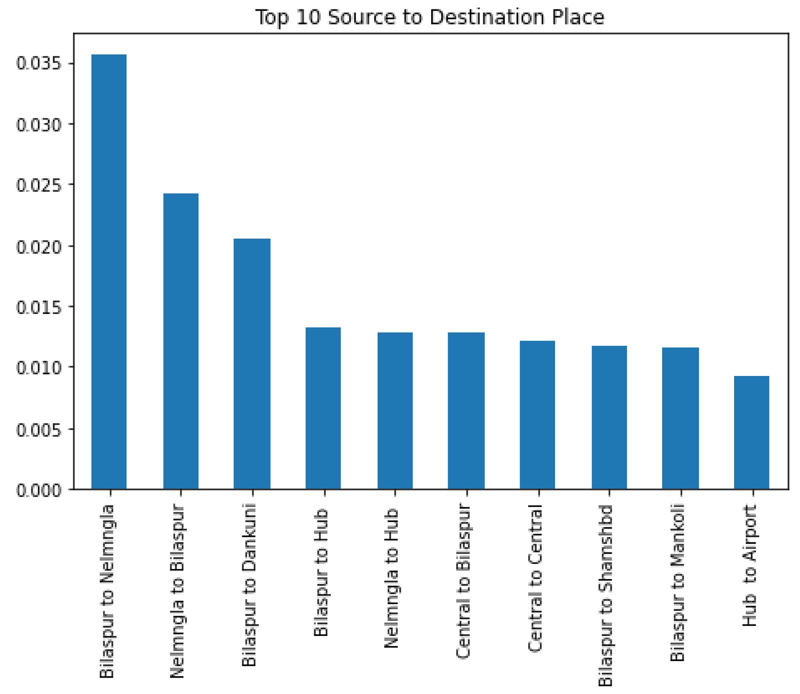
The given dataset is for the year 2018 for the month 9 and 10

```
plt.figure(figsize=(26,5))
plt.subplot(131)
plt.title("Top 10 Source to Destination State")
df['source_to_destination_state'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(132)
plt.title("Top 10 Source to Destination City")
df['source_to_destination_city'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(133)
plt.title("Top 10 Source to Destination Place")
df['source_to_destination_place'].value_counts(1)[:10].plot(kind='bar')
```

```
<Axes: title={'center': 'Top 10 Source to Destination Place'}>
```



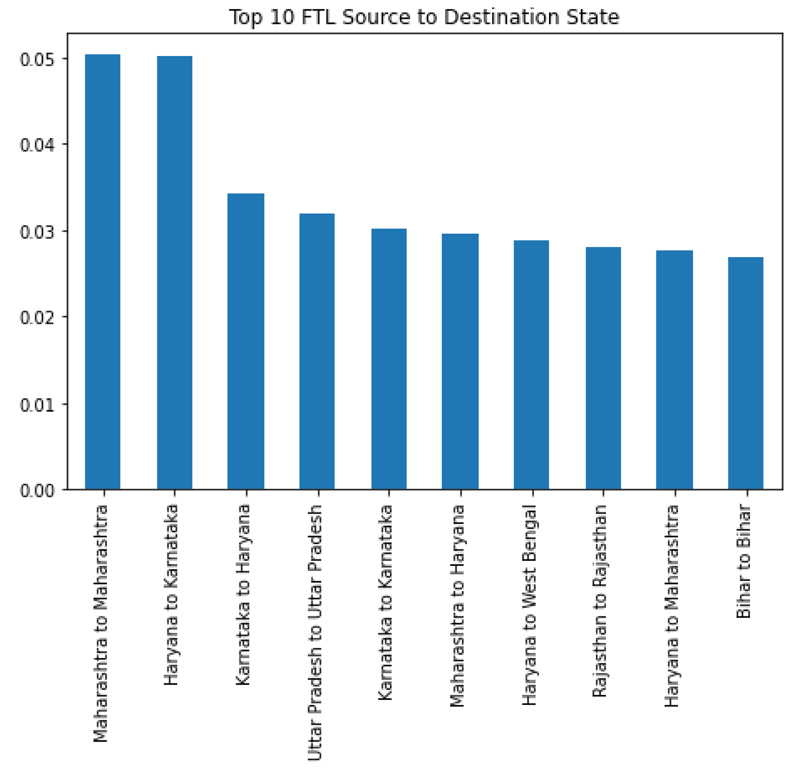


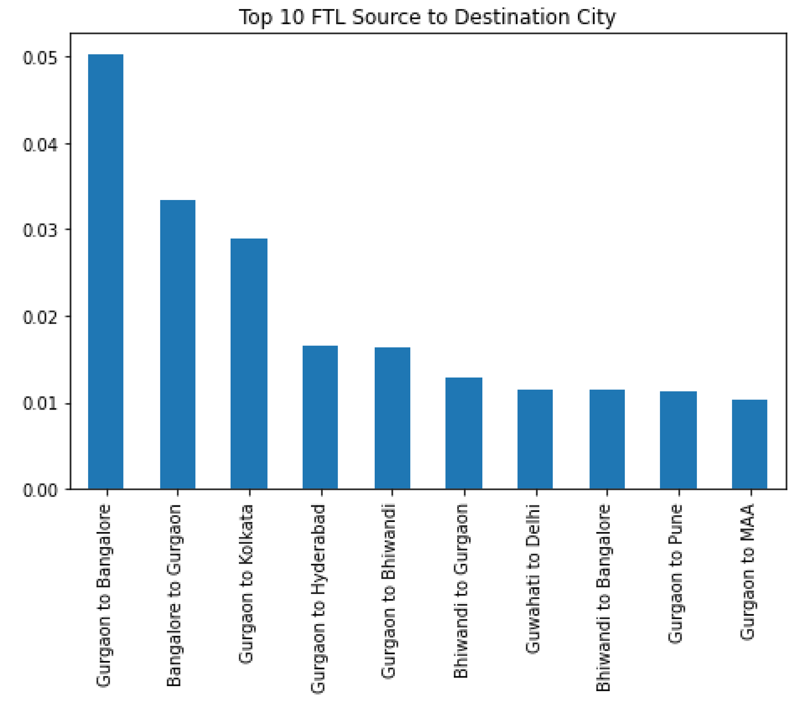


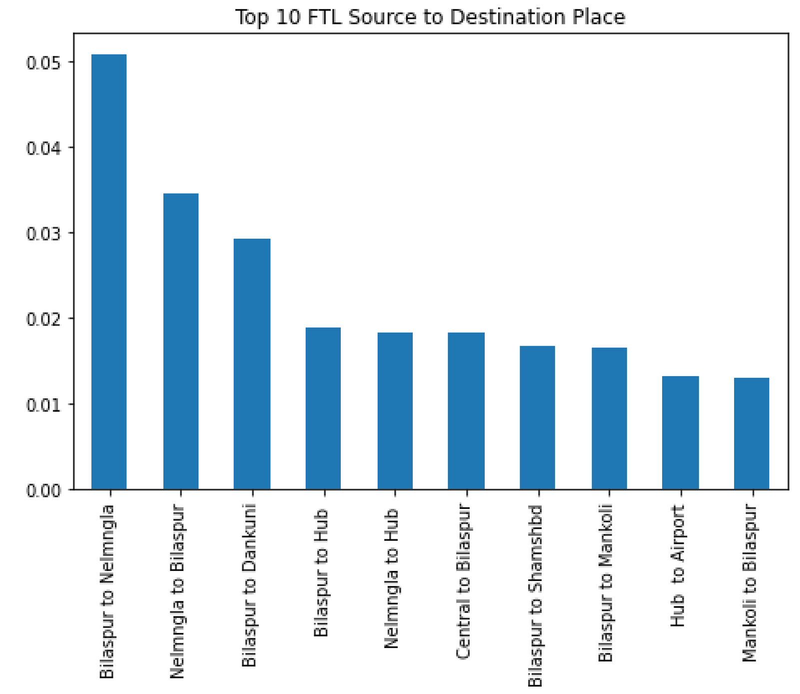
In the above graph we can see the TOP 10 Source to Destination State, City and Place

```
plt.figure(figsize=(26,5))
plt.subplot(131)
plt.title("Top 10 FTL Source to Destination State")
df[df["route_type"]==1]['source_to_destination_state'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(132)
plt.title("Top 10 FTL Source to Destination City")
df[df["route_type"]==1]['source_to_destination_city'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(133)
plt.title("Top 10 FTL Source to Destination Place")
df[df["route_type"]==1]['source_to_destination_place'].value_counts(1)[:10].plot(kind='bar')
```

```
<Axes: title={'center': 'Top 10 FTL Source to Destination Place'}>
```





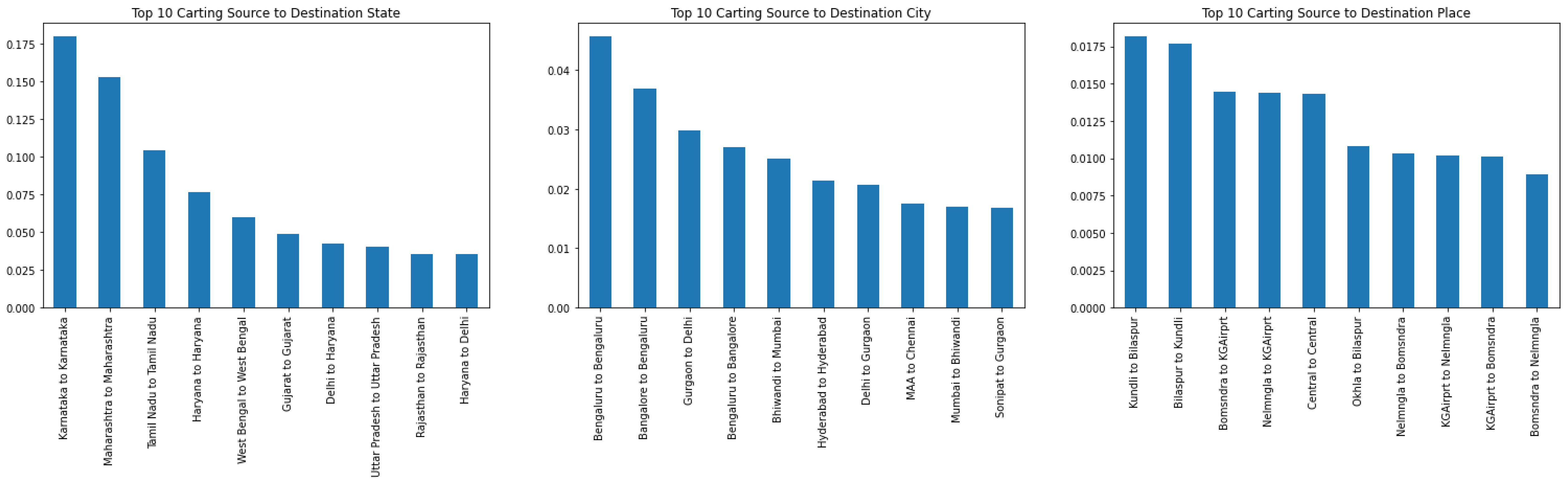


In the above graph we can see the TOP 10 FTL Source to Destination State, City and Place

```
plt.figure(figsize=(26,5))
plt.subplot(131)
```

```
plt.title("Top 10 Carting Source to Destination State")
df[df["route_type"]==0]['source_to_destination_state'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(132)
plt.title("Top 10 Carting Source to Destination City")
df[df["route_type"]==0]['source_to_destination_city'].value_counts(1)[:10].plot(kind='bar')
plt.subplot(133)
plt.title("Top 10 Carting Source to Destination Place")
df[df["route_type"]==0]['source_to_destination_place'].value_counts(1)[:10].plot(kind='bar')
```

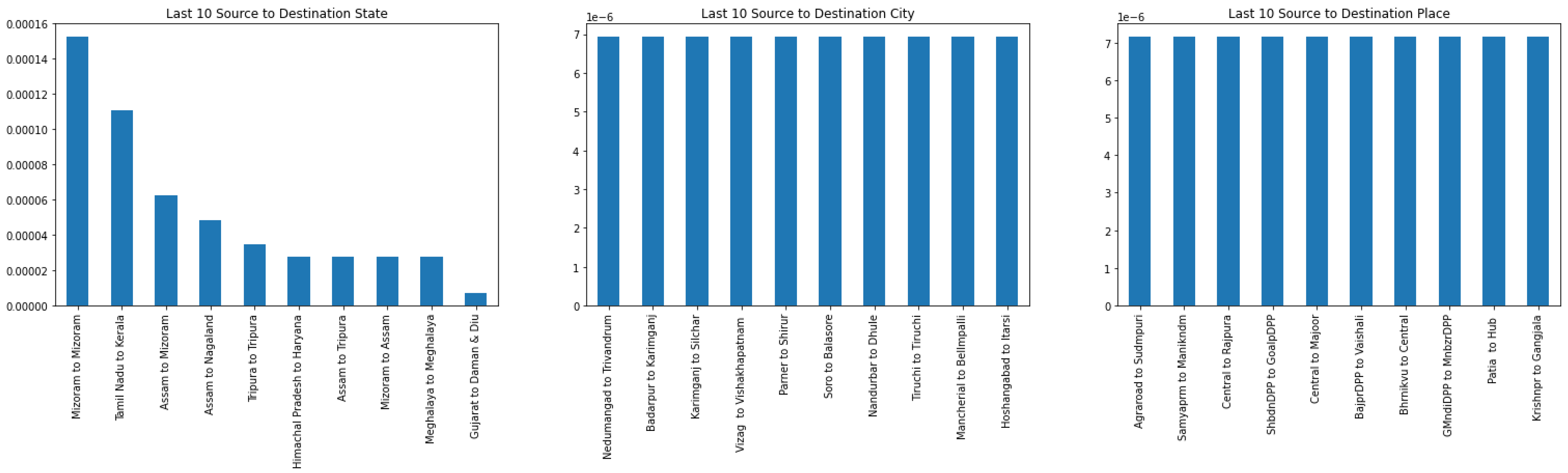
```
<Axes: title={'center': 'Top 10 Carting Source to Destination Place'}>
```



In the above graph we can see the TOP 10 Carting Source to Destination State, City and Place

```
plt.figure(figsize=(26,5))
plt.subplot(131)
plt.title("Last 10 Source to Destination State")
df['source_to_destination_state'].value_counts(1)[-10:].plot(kind='bar')
plt.subplot(132)
plt.title("Last 10 Source to Destination City")
df['source_to_destination_city'].value_counts(1)[-10:].plot(kind='bar')
plt.subplot(133)
plt.title("Last 10 Source to Destination Place")
df['source_to_destination_place'].value_counts(1)[-10:].plot(kind='bar')
```

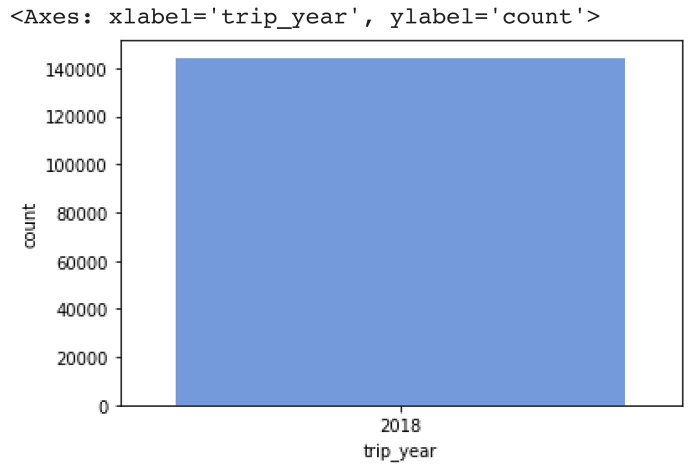
```
<Axes: title={'center': 'Last 10 Source to Destination Place'}>
```



In the above graph we can see the Least 10 Source to Destination State, City and Place

```
sns.countplot(x = 'trip_year', data = df, color='cornflowerblue')
```

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



In the given data set, the data is given for the year 2018

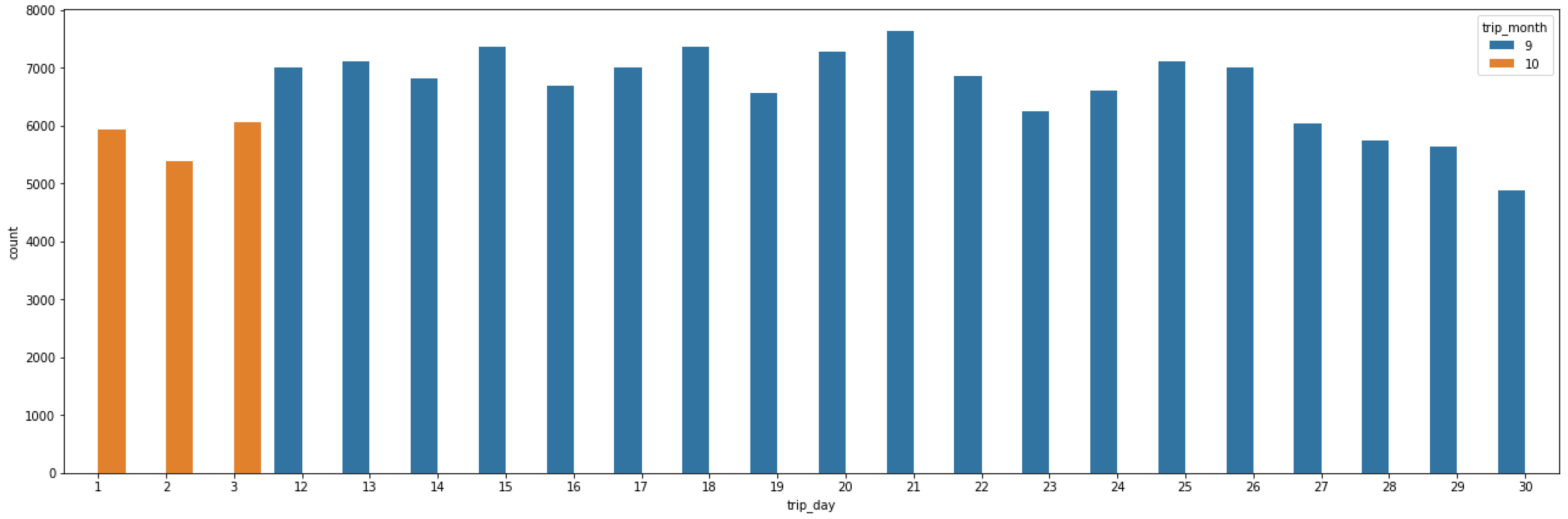
```
sns.countplot(x = 'trip_month', data = df, color='cornflowerblue')
```

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

In the given data set, the data is given for the month 9 and 10

```
plt.figure(figsize=(22, 7))
sns.countplot(data=df, x="trip_day", hue="trip_month")
```

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



From the above graph we can see that, the maximum trips is happening in the 9th month on 15, 18 and 21

```
trip_state_time_distance = df.groupby(["source_to_destination_state"]).agg({'start_scan_to_end_scan':'mean','actual_distance_to_destination':'mean'}).reset_index()
trip_state_time_distance.sort_values('start_scan_to_end_scan',ascending = False).head(10)
```

	source_to_destination_state	start_scan_to_end_scan	actual_distance_to_destination
117	Punjab to Karnataka	3802.000000	1050.751668
22	Delhi to Assam	3702.000000	759.661163
152	West Bengal to Maharashtra	3630.310160	837.091932
80	Karnataka to West Bengal	3595.000000	804.067604
9	Assam to Delhi	3573.291117	748.599138
53	Haryana to Tamil Nadu	3355.808867	876.628008
118	Punjab to Maharashtra	3297.000000	660.579031
101	Maharashtra to West Bengal	3167.635682	831.090075
71	Karnataka to Haryana	3125.825869	869.096298
47	Haryana to Karnataka	3097.507637	859.827666

From the above table we can see that, the top 10 most time taking trip from the source to destination state

```
trip_state_time_distance.sort_values('actual_distance_to_destination',ascending = False).head(10)
```

	source_to_destination_state	start_scan_to_end_scan	actual_distance_to_destination
117	Punjab to Karnataka	3802.000000	1050.751668
53	Haryana to Tamil Nadu	3355.808867	876.628008
71	Karnataka to Haryana	3125.825869	869.096298
47	Haryana to Karnataka	3097.507637	859.827666
152	West Bengal to Maharashtra	3630.310160	837.091932
101	Maharashtra to West Bengal	3167.635682	831.090075
80	Karnataka to West Bengal	3595.000000	804.067604
77	Karnataka to Rajasthan	3076.875256	781.405522
22	Delhi to Assam	3702.000000	759.661163
9	Assam to Delhi	3573.291117	748.599138

From the above table we can see that, the top 10 most long distance trip from the source to destination state

```
trip_city_time_distance = df.groupby(["source_to_destination_city"]).agg({'start_scan_to_end_scan':'mean','actual_distance_to_destination':'mean'}).reset_index()
trip_city_time_distance.sort_values('start_scan_to_end_scan',ascending = False).head(10)
```

	source_to_destination_city	start_scan_to_end_scan	actual_distance_to_destination
449	Chandigarh to Bangalore	3802.000000	1050.751668
582	Delhi to Guwahati	3702.000000	759.661163
1265	Kolkata to Bhiwandi	3630.310160	837.091932
207	Bangalore to Kolkata	3595.000000	804.067604
851	Guwahati to Delhi	3573.291117	748.599138
834	Gurgaon to MAA	3355.808867	876.628008
452	Chandigarh to Bhiwandi	3297.000000	660.579031
336	Bhiwandi to Kolkata	3167.635682	831.090075
271	Bengaluru to Gurgaon	3140.000000	870.118858
198	Bangalore to Gurgaon	3125.492461	869.072245

From the above table we can see that, the top 10 most time taking trip from the source to destination city

```
trip_city_time_distance.sort_values('actual_distance_to_destination',ascending = False).head(10)
```

	source_to_destination_city	start_scan_to_end_scan	actual_distance_to_destination
449	Chandigarh to Bangalore	3802.000000	1050.751668
834	Gurgaon to MAA	3355.808867	876.628008
271	Bengaluru to Gurgaon	3140.000000	870.118858
198	Bangalore to Gurgaon	3125.492461	869.072245
810	Gurgaon to Bangalore	3097.507637	859.827666
1265	Kolkata to Bhiwandi	3630.310160	837.091932
336	Bhiwandi to Kolkata	3167.635682	831.090075
207	Bangalore to Kolkata	3595.000000	804.067604
204	Bangalore to Jaipur	3076.875256	781.405522
582	Delhi to Guwahati	3702.000000	759.661163

From the above table we can see that, the top 10 most long distance trip from the source to destination city

```
df.drop(["od_end_time","od_start_time"], axis=1, inplace=True)
```

```
#Defining the alpha value to 0.05
alpha = 0.05

#Function to plot the graph
def analysis_plot(new_df,col_a,col_b):
  plt.figure(figsize=(25,5))
  plt.subplot(131)
  plt.title(col_a+" vs "+col_b)
  sns.scatterplot(data=new_df, x=new_df[col_a], y=new_df[col_b])
  plt.subplot(132)
  plt.title(col_a)
  sns.histplot(new_df[col_a],kde=True)
  plt.subplot(133)
  plt.title(col_b)
  sns.histplot(new_df[col_b],kde=True)

#Function to perform the hypothesis testing
def feature_analysis(new_df,col_a,col_b):
  H0 = col_a+" and "+col_b+" are Equal"
  Ha = col_a+" and "+col_b+" are Not Equal"
  print("Given H0 --> "+ H0)
  print("Given Ha --> "+ Ha)
  stat, pvalue = ttest_ind(new_df[col_a], new_df[col_b])
  print("Statistics --> "+str(stat))
  print("Pvalue --> "+str(pvalue))
  if(pvalue<alpha):
    print("H0 is Rejected, Therefore "+Ha)
  else:
    print("H0 is Accepted, Therefore "+H0)

#Function to plot the box plot
def box_plot(new_df,col_a,col_b,col_c):
  plt.figure(figsize=(25,5))
  plt.subplot(131)
  sns.boxplot(y=col_a,data=new_df)
  print("\n"+"*"*20+" "+col_a+" "+"*"*20)
  outlier_detection(new_df,col_a)
  plt.subplot(132)
  sns.boxplot(y=col_b,data=new_df)
  print("\n"+"*"*20+" "+col_b+" "+"*"*20)
  outlier_detection(new_df,col_b)
  plt.subplot(133)
  sns.boxplot(y=col_c,data=new_df)
  print("\n"+"*"*20+" "+col_c+" "+"*"*20)
  outlier_detection(new_df,col_c)

#Function to detect the outliers and count number of outliers
def outlier_detection(new_df,col_name):
  p_25 = np.percentile(new_df[col_name], 25)
  p_50 = np.percentile(new_df[col_name], 50)
  p_75 = np.percentile(new_df[col_name], 75)

  print("First Quartile: ", p_25) # p = 25%
  print("Second Quartile: ", p_50)# p = 50%
  print("Third Quartile: ", p_75) # p = 75%

  print("IQR: ", p_75 - p_25)

  left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
  right_whis = p_75 + 1.5 * (p_75 - p_25)

  print("Left: ", left_whis)
  print("Right: ", right_whis)

  num_outliers = len(new_df[new_df[col_name] > right_whis])
  print(col_name+" outliers: ", num_outliers)

  print(col_name+" std: ", new_df[col_name].std())
```

```
trip_time_distance = df.groupby(["trip_uuid"]).agg({'actual_time':'max','osrm_time':'max','od_total_time':'max','start_scan_to_end_scan':'max','segment_actual_time':'sum',
                               'osrm_distance':'max','segment_osrm_distance':'sum','segment_osrm_time':'sum','actual_distance_to_destination':'max'}).reset_index()
```

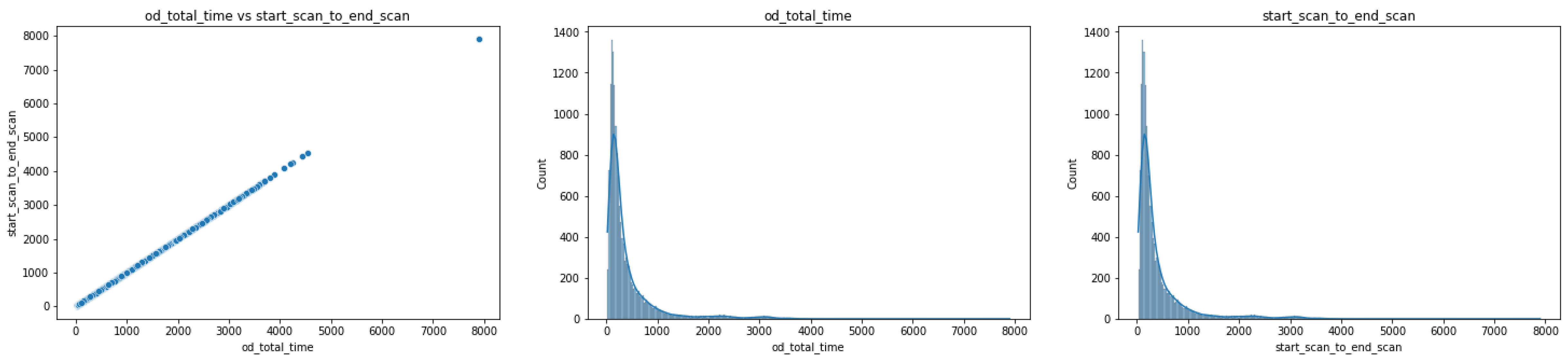
Grouping by the trip_uuid and calculating the Aggregated actual_time, osrm_time, od_total_time, start_scan_to_end_scan, segment_actual_time, osrm_distance, segment_osrm_distance, segment_osrm_time, actual_distance_to_destination

```
trip_time_distance
```

	trip_uuid	actual_time	osrm_time	od_total_time	start_scan_to_end_scan	segment_actual_time	osrm_distance	segment_osrm_distance	segment_osrm_time	actual_distance_to_destination
0	trip-153671041653548748	830.0	394.0	1260.0	1260.0	1548.0	544.8027	1320.4733	1008.0	440.973689
1	trip-153671042288605164	96.0	42.0	122.0	122.0	141.0	56.9116	84.1894	65.0	48.542890
2	trip-153671043369099517	2736.0	1529.0	3099.0	3099.0	3308.0	2090.8743	2545.2678	1941.0	1689.964663
3	trip-153671046011330457	59.0	15.0	100.0	100.0	59.0	19.6800	19.8766	16.0	17.175274
4	trip-153671052974046625	147.0	46.0	485.0	485.0	340.0	63.6461	146.7919	115.0	59.530350
...	...	...	...	...	...	...	...	...	...	...
14782	trip-153861095625827784	49.0	34.0	152.0	152.0	82.0	44.5639	64.8551	62.0	31.261599
14783	trip-153861104386292051	21.0	12.0	60.0	60.0	21.0	16.0882	16.0883	11.0	15.513784
14784	trip-153861106442901555	190.0	29.0	248.0	248.0	281.0	32.2277	104.8866	88.0	19.349008
14785	trip-153861115439069069	90.0	50.0	105.0	105.0	258.0	52.8070	223.5324	221.0	37.387664
14786	trip-153861118270144424	233.0	42.0	287.0	287.0	274.0	52.5303	80.5787	67.0	40.546740

14787 rows × 10 columns

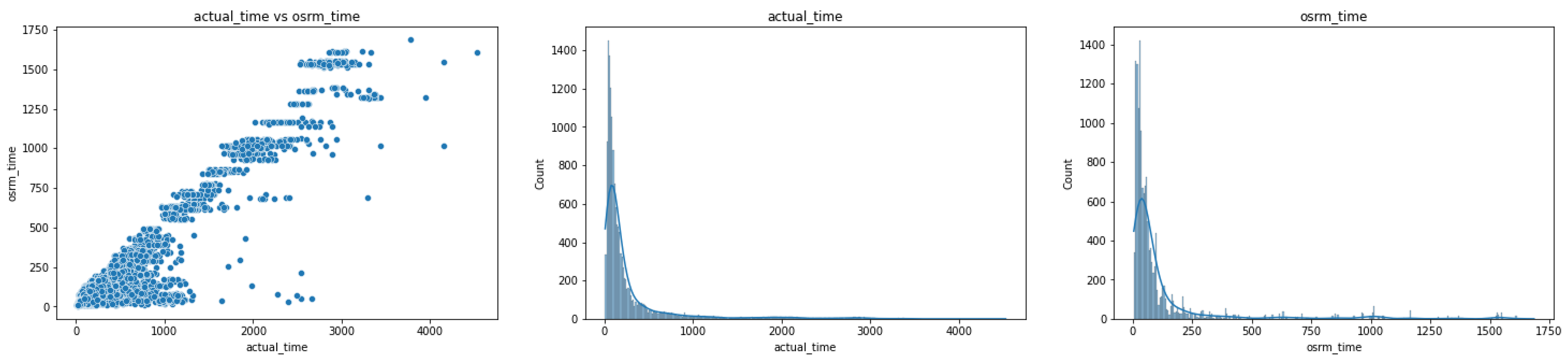
```
analysis_plot(trip_time_distance,"od_total_time","start_scan_to_end_scan")
```




```
feature_analysis(trip_time_distance,"od_total_time","start_scan_to_end_scan")
```

```
Given H0 --> od_total_time and start_scan_to_end_scan are Equal
Given Ha --> od_total_time and start_scan_to_end_scan are Not Equal
Statistics --> 0.0
Pvalue --> 1.0
H0 is Accepted, Therefore od_total_time and start_scan_to_end_scan are Equal
```

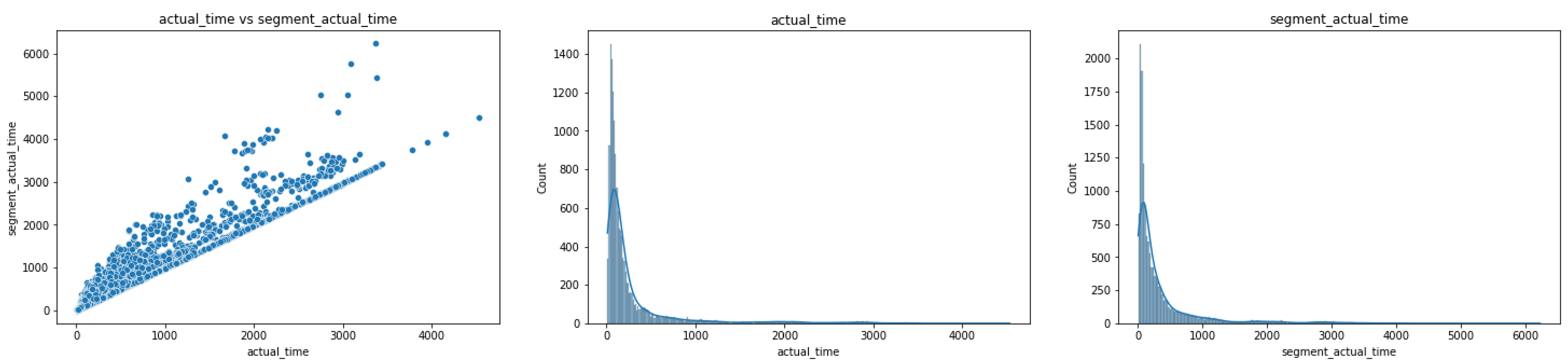
```
analysis_plot(trip_time_distance,"actual_time","osrm_time")
```




```
feature_analysis(trip_time_distance,"actual_time","osrm_time")
```

```
Given H0 --> actual_time and osrm_time are Equal
Given Ha --> actual_time and osrm_time are Not Equal
Statistics --> 35.08988288617061
Pvalue --> 2.5609809806142685e-264
H0 is Rejected, Therefore actual_time and osrm_time are Not Equal
```

```
analysis_plot(trip_time_distance,"actual_time","segment_actual_time")
```

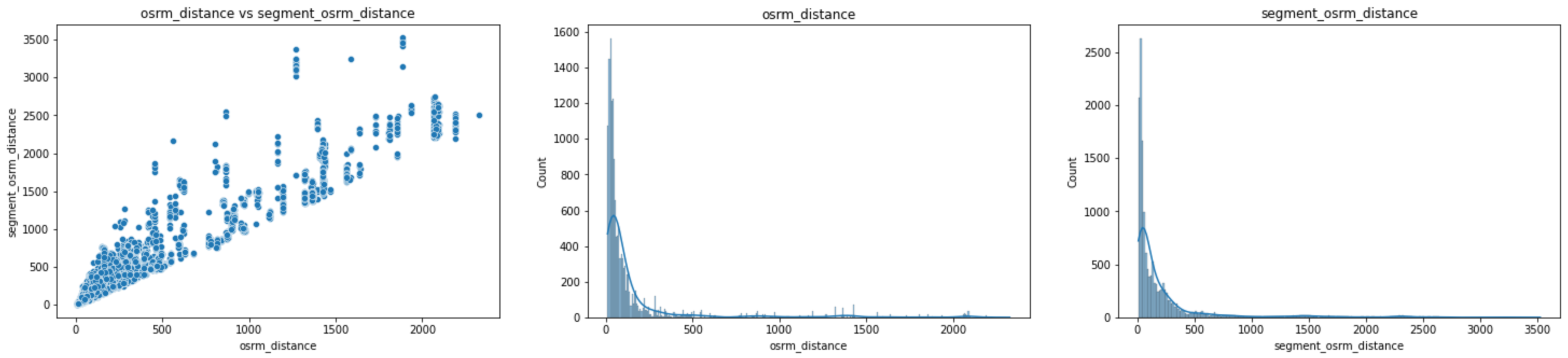



```
feature_analysis(trip_time_distance,"segment_actual_time","actual_time")
```

```
Given H0 --> segment_actual_time and actual_time are Equal
Given Ha --> segment_actual_time and actual_time are Not Equal
```

```
Statistics --> 12.366405130681308
Pvalue --> 4.849541124158181e-35
H0 is Rejected, Therefore segment_actual_time and actual_time are Not Equal
```

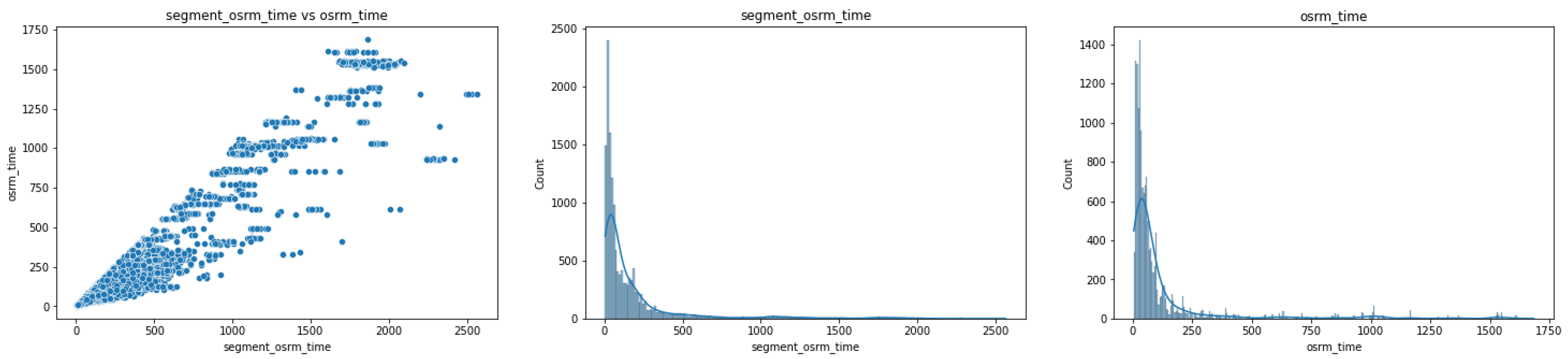
```
analysis_plot(trip_time_distance,"osrm_distance","segment_osrm_distance")
```




```
feature_analysis(trip_time_distance,"segment_osrm_distance","osrm_distance")
```

```
Given H0 --> segment_osrm_distance and osrm_distance are Equal
Given Ha --> segment_osrm_distance and osrm_distance are Not Equal
Statistics --> 15.373808827249137
Pvalue --> 3.9396914792427207e-53
H0 is Rejected, Therefore segment_osrm_distance and osrm_distance are Not Equal
```

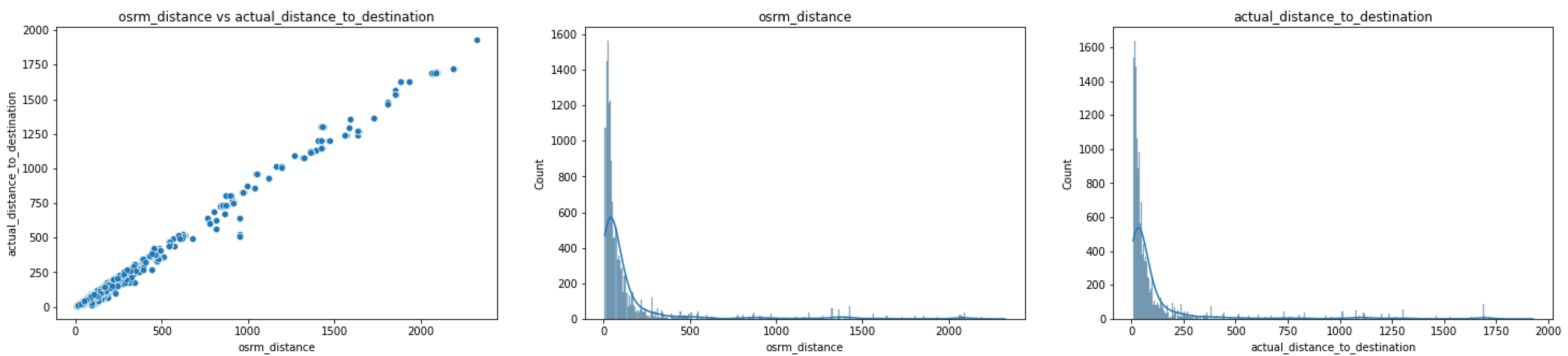
```
analysis_plot(trip_time_distance,"segment_osrm_time","osrm_time")
```




```
feature_analysis(trip_time_distance,"segment_osrm_time","osrm_time")
```

```
Given H0 --> segment_osrm_time and osrm_time are Equal
Given Ha --> segment_osrm_time and osrm_time are Not Equal
Statistics --> 18.108706618672148
Pvalue --> 6.745046176314656e-73
H0 is Rejected, Therefore segment_osrm_time and osrm_time are Not Equal
```

```
analysis_plot(trip_time_distance,"osrm_distance","actual_distance_to_destination")
```




```
feature_analysis(trip_time_distance,"osrm_distance","actual_distance_to_destination")
```

```
Given H0 --> osrm_distance and actual_distance_to_destination are Equal
Given Ha --> osrm_distance and actual_distance_to_destination are Not Equal
Statistics --> 8.84614890355335
Pvalue --> 9.552148425013404e-19
H0 is Rejected, Therefore osrm_distance and actual_distance_to_destination are Not Equal
```

```
trip_time_distance.columns
```

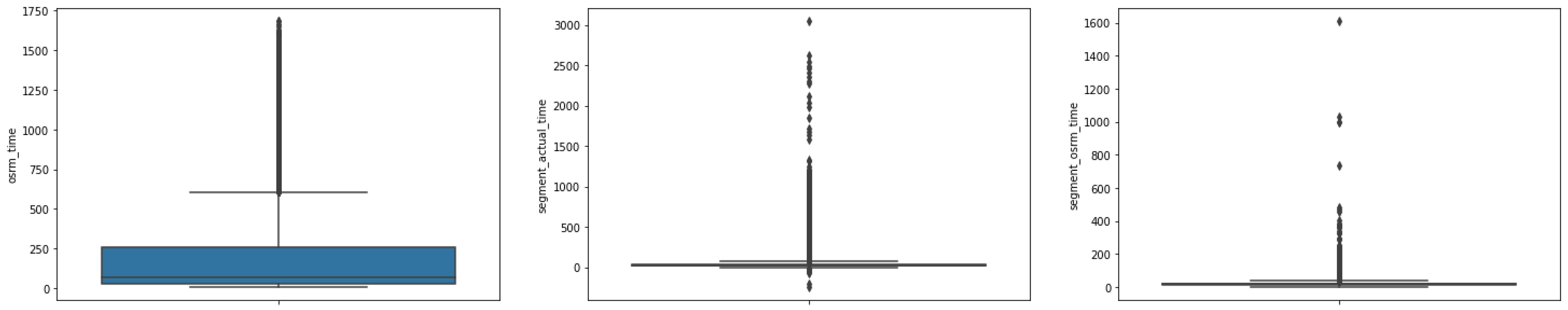
```
Index(['trip_uuid', 'actual_time', 'osrm_time', 'od_total_time',
       'start_scan_to_end_scan', 'segment_actual_time', 'osrm_distance',
       'segment_osrm_distance', 'segment_osrm_time',
       'actual_distance_to_destination'],
      dtype='object')
```

```
box_plot(df,"actual_time","od_total_time","start_scan_to_end_scan")
```

```
******************** actual_time ********************
First Quartile:  51.0
Second Quartile:  132.0
Third Quartile:  516.0
IQR:  465.0
Left:  0
Right:  1213.5
actual_time outliers:  16507
actual_time std:  598.9400647682617

******************** od_total_time ********************
First Quartile:  161.0
Second Quartile:  451.0
Third Quartile:  1645.0
IQR:  1484.0
Left:  0
Right:  3871.0
od_total_time outliers:  373
od_total_time std:  1038.0829762943533

******************** start_scan_to_end_scan ********************
First Quartile:  161.0
Second Quartile:  451.0
Third Quartile:  1645.0
IQR:  1484.0
Left:  0
Right:  3871.0
start_scan_to_end_scan outliers:  373
start_scan_to_end_scan std:  1038.0829762943533
```

```
box_plot(df,"osrm_time","segment_actual_time","segment_osrm_time")
```

```
******************** osrm_time ********************
First Quartile:  27.0
Second Quartile:  64.0
Third Quartile:  259.0
IQR:  232.0
Left:  0
Right:  607.0
osrm_time outliers:  17406
osrm_time std:  308.4485430314111

******************** segment_actual_time ********************
First Quartile:  20.0
Second Quartile:  28.0
Third Quartile:  40.0
IQR:  20.0
Left:  0
Right:  70.0
segment_actual_time outliers:  9249
segment_actual_time std:  53.52429778355094

******************** segment_osrm_time ********************
First Quartile:  11.0
Second Quartile:  17.0
Third Quartile:  22.0
IQR:  11.0
Left:  0
Right:  38.5
segment_osrm_time outliers:  6348
segment_osrm_time std:  14.774007972115733
```




```
box_plot(df,"osrm_distance","segment_osrm_distance","actual_distance_to_destination")
```

```
******************** osrm_distance ********************
First Quartile:  29.89625
Second Quartile:  78.6244
Third Quartile:  346.3054
IQR:  316.40915
Left:  0
Right:  820.9191250000001
osrm_distance outliers:  17547
osrm_distance std:  421.7178256506773

******************** segment_osrm_distance ********************
First Quartile:  12.053975000000001
Second Quartile:  23.5083
```

```
trip_time_distance.drop(["trip_uuid"], axis=1, inplace=True)
```

```
Left:  0
```

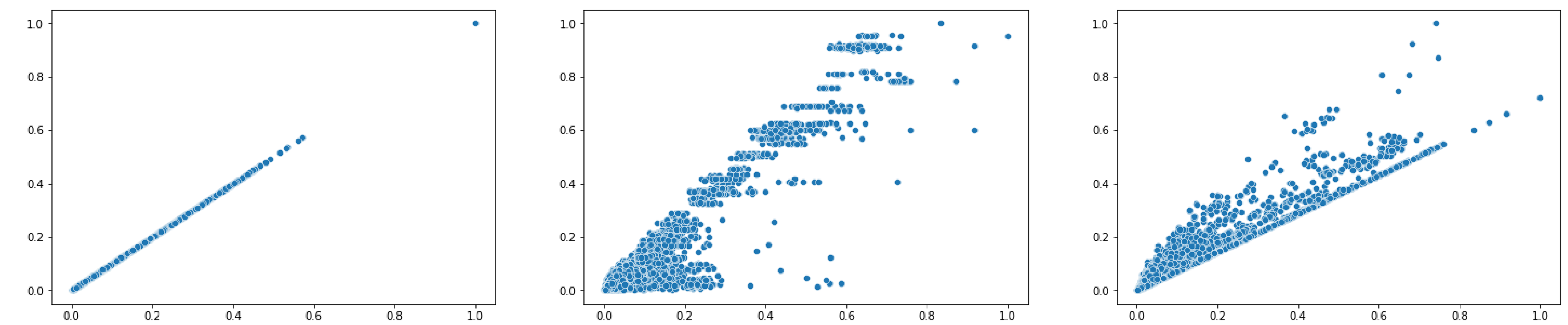
```
trip_time_distance_normal = MinMaxScaler().fit_transform(trip_time_distance)
trip_time_distance_normal
```

```
array([[0.18151669, 0.23095238, 0.15707937, ..., 0.37313365, 0.39171228,
        0.22516735],
       [0.01923502, 0.02142857, 0.01257143, ..., 0.02137295, 0.02306489,
        0.02061066],
       [0.60291842, 0.90654762, 0.39060317, ..., 0.72162526, 0.75645035,
        0.87621067],
       ...,
       [0.04001769, 0.01369048, 0.02857143, ..., 0.02726194, 0.03205629,
        0.00539319],
       [0.01790847, 0.02619048, 0.0104127 , ..., 0.06102031, 0.08405004,
        0.01479594],
       [0.04952465, 0.02142857, 0.03352381, ..., 0.02034559, 0.02384676,
        0.01644263]])
```

- *Normalizing the numerical values using fit_transform and MinMaxScaler*
- *The below Graph shows the Normalized Scatter Graph for the numerical values*

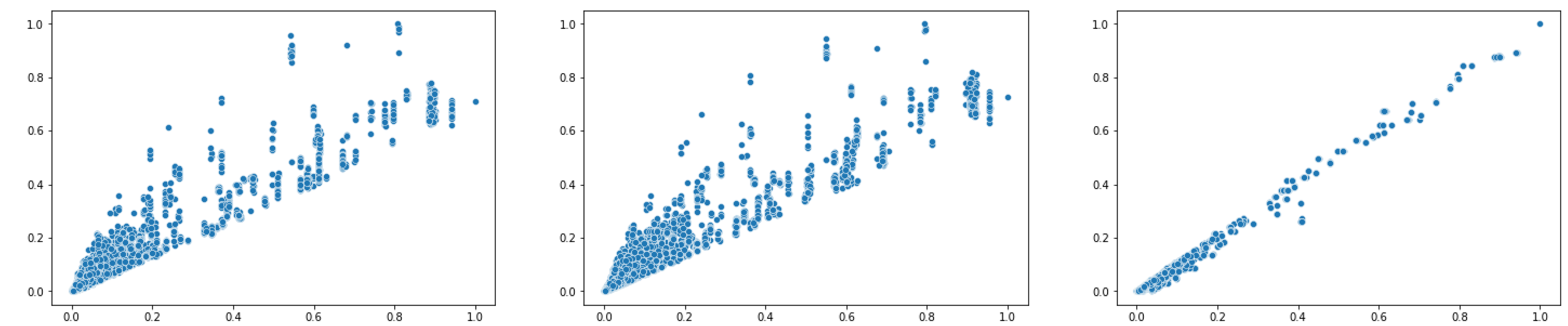
```
plt.figure(figsize=(25,5))
plt.subplot(131)
sns.scatterplot(x=trip_time_distance_normal[:,2], y=trip_time_distance_normal[:,3])
plt.subplot(132)
sns.scatterplot(x=trip_time_distance_normal[:,0], y=trip_time_distance_normal[:,1])
plt.subplot(133)
sns.scatterplot(x=trip_time_distance_normal[:,0], y=trip_time_distance_normal[:,4])
```

```
<Axes: >
```

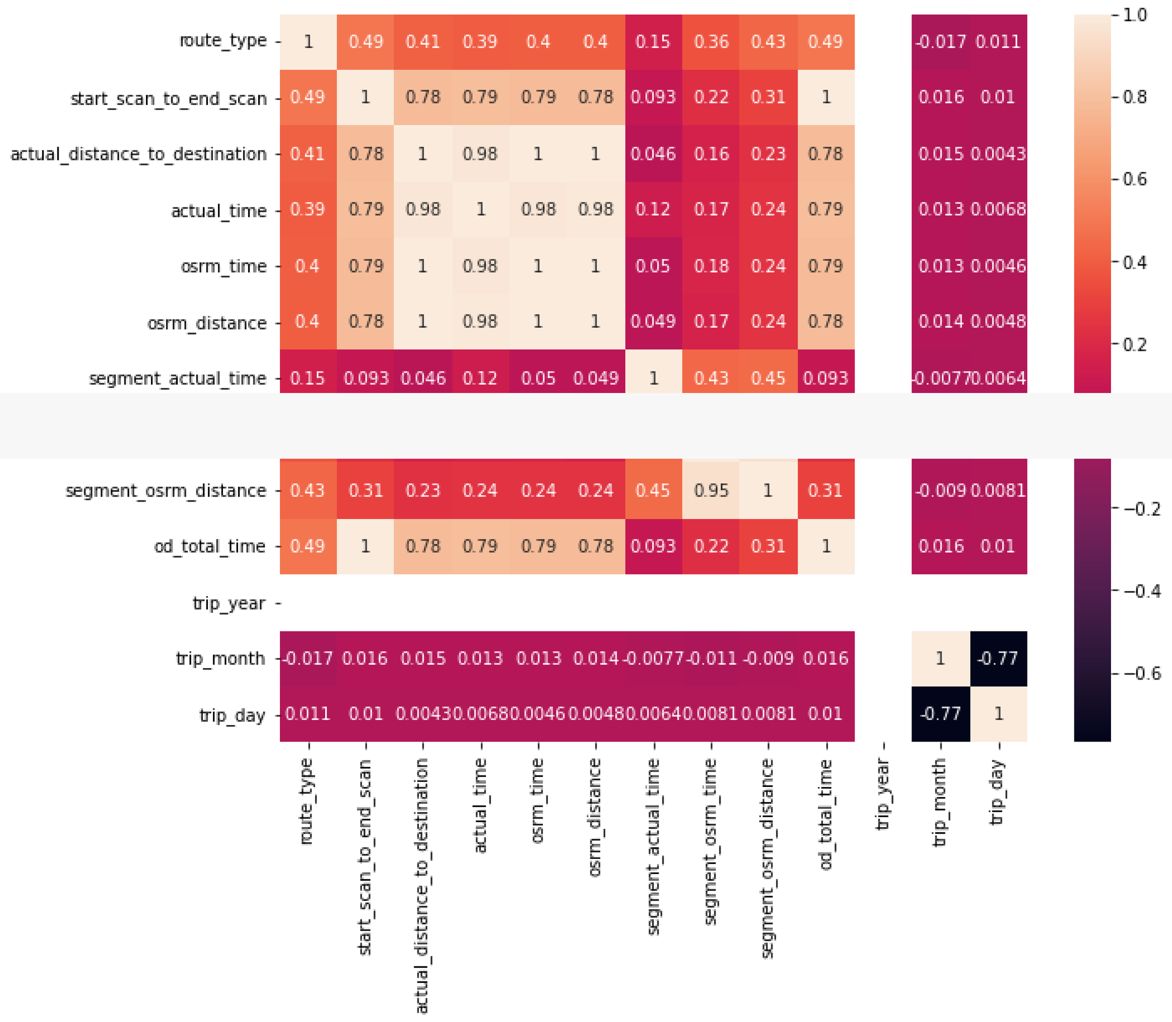


```
plt.figure(figsize=(25,5))
plt.subplot(131)
sns.scatterplot(x=trip_time_distance_normal[:,5], y=trip_time_distance_normal[:,6])
plt.subplot(132)
sns.scatterplot(x=trip_time_distance_normal[:,1], y=trip_time_distance_normal[:,7])
plt.subplot(133)
sns.scatterplot(x=trip_time_distance_normal[:,5], y=trip_time_distance_normal[:,8])
```

```
<Axes: >
```



```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True);
```



✓ 2s completed at 8:43 PM