Business Case: Delhivery - Feature Engineering Delhivery - Feature Engineering

Nikhil K A

SCALER DSML

Business Case: Delhivery - Feature Engineering

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.

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

1. Defining Problem Statement and perform Exploratory Data Analysis
    a) Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), statistical summary

Delhivery, a leading logistics provider in India, faces the challenge of effectively harnessing data from its engineering pipelines to enhance operational efficiency and competitiveness. The Data team is tasked with cleaning, sanitizing, and manipulating this raw data to extract meaningful features. The business case leverages Python's robust data analytics and visualization capabilities to extract valuable insights from the data set. By harnessing feature engineering, hypothesis testing and Python libraries such as Pandas, NumPy, SciPy, Matplotlib, and Seaborn, the case aims to gain a comprehensive understanding of the various metrices. By transforming unrefined data into structured insights, they aim to support the data science team in developing accurate forecasting models. This process is crucial for enabling Delhivery to maintain its edge in quality, efficiency, and profitability in a rapidly evolving market.

The data set have the following columns:

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 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)
5	trip_uuid	:	Unique ID given to a particular trip (A trip may include different source and destination centers)
6	source_center	:	Source ID of trip origin
7	source_name	:	Source Name of trip origin
8	destination_cente	:	Destination ID
9	destination_name	:	Destination Name
10	od_start_time	:	Trip start time
11	od_end_time	:	Trip end time
12	start_scan_to_end_scan	:	Time taken to deliver from source to destination
13	is_cutoff	:	Unknown field
14	cutoff_factor	:	Unknown field
15	cutoff_timestamp	:	Unknown field
16	actual_distance_to_destination	:	Distance in Kms between source and destination warehouse
17	actual_time	:	Actual time taken to complete the delivery (Cumulative)
18	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)
19	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)
20	factor	:	Unknown field
21	segment_actual_time	:	This is a segment time. Time taken by the subset of the package delivery
22	segment_osrm_time	:	This is the OSRM segment time. Time taken by the subset of the package delivery
23	segment_osrm_distance	:	This is the OSRM distance. Distance covered by subset of the package delivery
24	segment_factor	:	Unknown field

The data set is downloaded as 'delhivery_data.csv' and saved as dataframe named 'df'.

```
[2] !wget --no-check-certificate  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv
```

```
--2024-09-21 10:21:22--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 65.8.234.72, 65.8.234.174, 65.8.234.36, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|65.8.234.72|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: 'delhivery_data.csv'

delhivery_data.csv  100%[===================>]  53.04M   102MB/s    in 0.5s

2024-09-21 10:21:22 (102 MB/s) - 'delhivery_data.csv' saved [55617130/55617130]
```

```
[3] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[5] df=pd.read_csv('delhivery_data.csv')
```

The data sample is observed by df.head()

```
[7] df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center
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
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
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
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
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

5 rows × 24 columns

destination_name	od_start_time	...	cutoff_timestamp	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:27:55	10.435660	14.0	11.0	11.9653	1.272727
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:17:55	18.936842	24.0	20.0	21.7243	1.200000
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 04:01:19.505586	27.637279	40.0	28.0	32.5395	1.428571
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:39:57	36.118028	62.0	40.0	45.5620	1.550000
Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	2018-09-20 03:33:55	39.386040	68.0	44.0	54.2181	1.545455

segment_actual_time	segment_osrm_time	segment_osrm_distance	segment_factor
14.0	11.0	11.9653	1.272727
10.0	9.0	9.7590	1.111111
16.0	7.0	10.8152	2.285714
21.0	12.0	13.0224	1.750000
6.0	5.0	3.9153	1.200000

The data is divided into 24 columns and there are 144867 rows in the dataset.

Shape of the dataframe : df.shape showed

```
[9] df.shape
(144867, 24)
```

The basic information about dataframe. df.info()

```
[11] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  object
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  object
 10  od_end_time                     144867 non-null  object
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool
 13  cutoff_factor                   144867 non-null  int64
 14  cutoff_timestamp                144867 non-null  object
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

Data type of 12 of the 24 columns are object type, 10 of them are float64, one of the columns being int64 and one bool.

Detecting missing values by isna().

```
[13] df.isna().sum()
```

	0
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

It shows there are 293 null values or missing values in the source_name column and 261 in the destination_name column in the dataset.

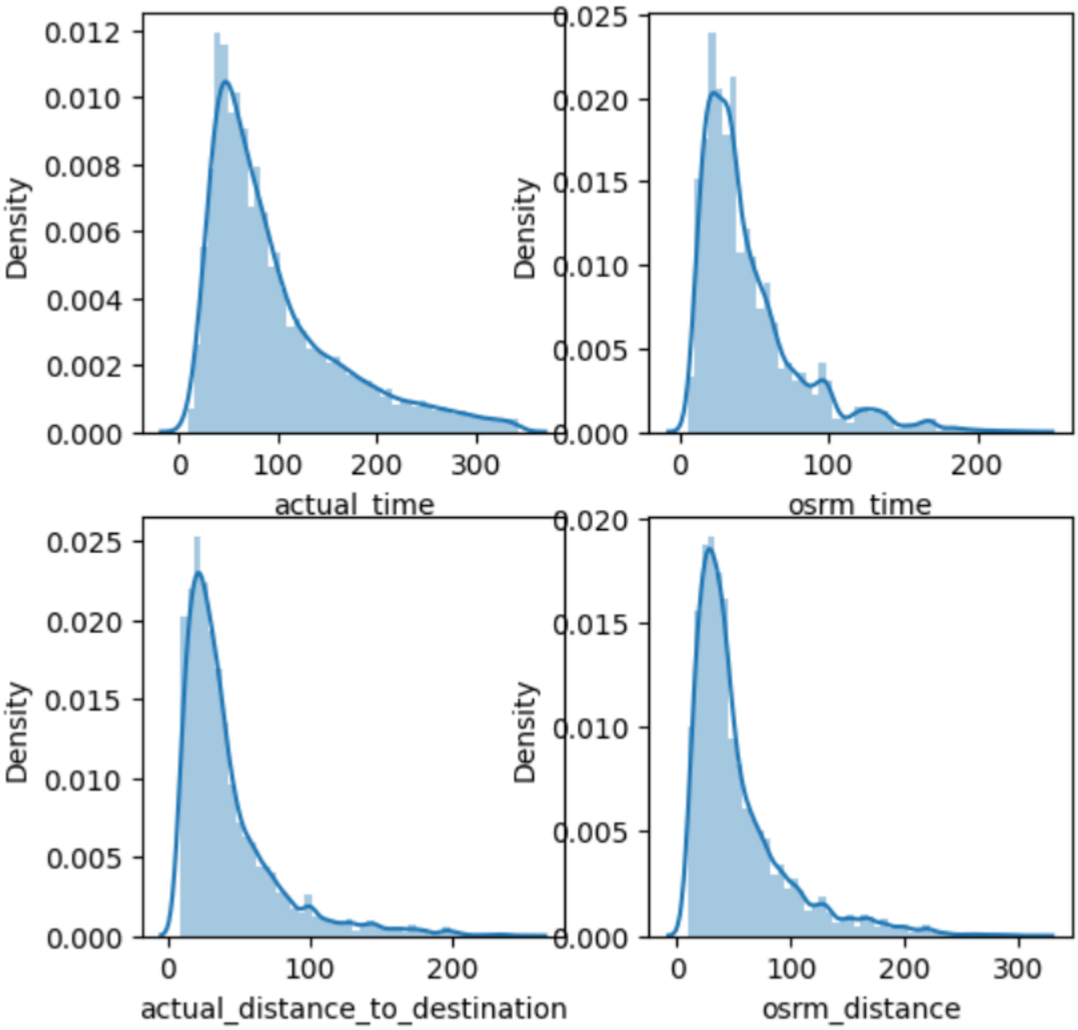
The unknown fields are removed using drop function

```
[5] df.drop(["is_cutoff","cutoff_factor","cutoff_timestamp","factor","segment_factor"],axis=1,inplace=True)
```

b) Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the categorical variables)

Distribution of actual and osrm time and distances are plotted using distplot.

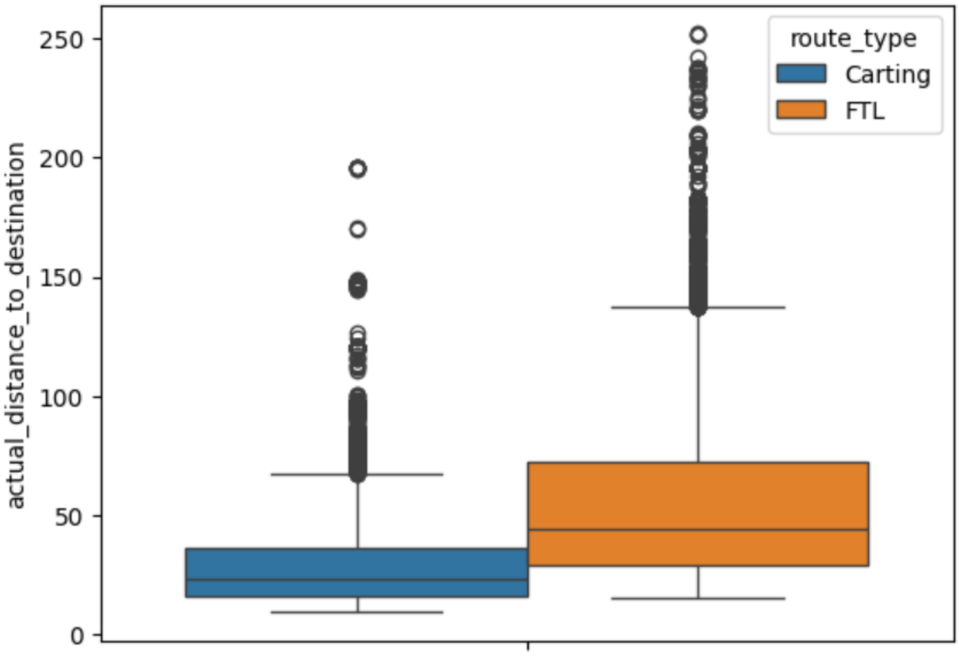
```
plt.figure(figsize=(6,6))
plt.subplot(2,2,1)
sns.distplot(df1['actual_time'])
plt.subplot(2,2,2)
sns.distplot(df1['osrm_time'])
plt.subplot(2,2,3)
sns.distplot(df1['actual_distance_to_destination'])
plt.subplot(2,2,4)
sns.distplot(df1['osrm_distance'])
```



Boxplot of the data of the distance of trips with respect to different modes of transport:

```
[53] sns.boxplot(data=df1,y='actual_distance_to_destination',hue='route_type')
```

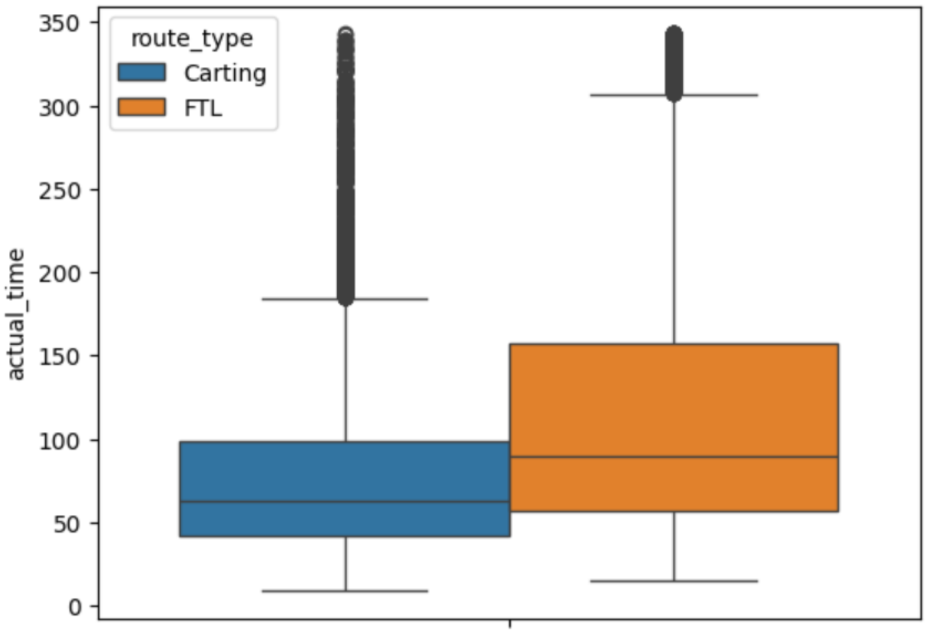
```
<Axes: ylabel='actual_distance_to_destination'>
```



Box plot of the time taken for trips with respect to different modes of transport:

```
[54] sns.boxplot(data=df1,y='actual_time',hue='route_type')
```

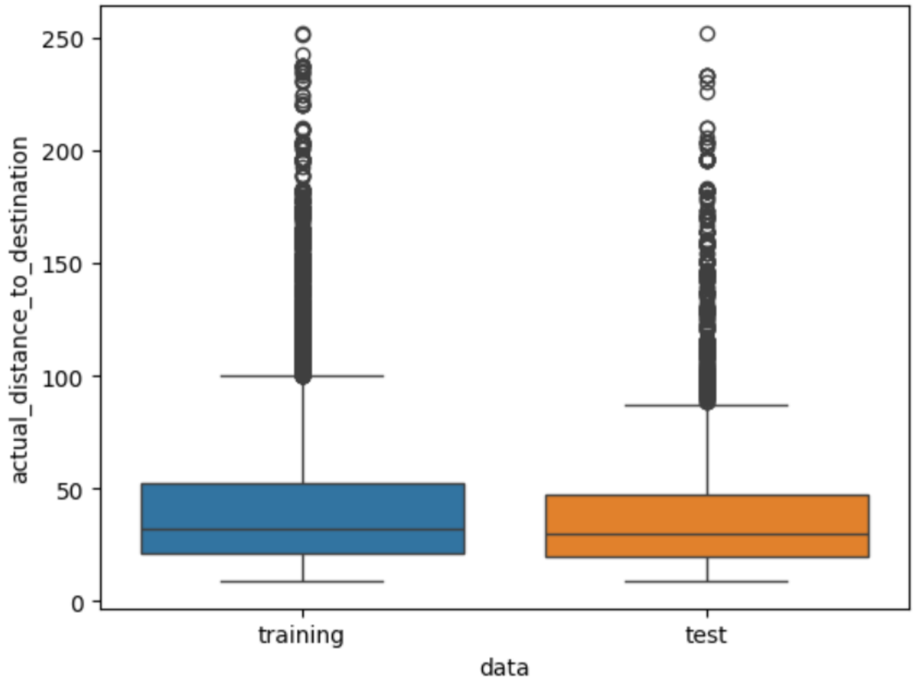
```
<Axes: ylabel='actual_time'>
```



Box plot for distance of trips with respect to the type of data:

```
sns.boxplot(data=df1,x='data',y='actual_distance_to_destination',hue='data')
```

```
<Axes: xlabel='data', ylabel='actual_distance_to_destination'>
```



c) Insights based on EDA: Comments on range of attributes, outliers of various attributes, Comments on the distribution of the variables and relationship between them, Comments for each univariate and bivariate plot

- The data contains 144867 records of different segments of delivery trips.
- There are data of 14787 different trips.
- The data is regarding the delivery trips from 12-09-2018 to 03-10-2018.
- The distance covered in the trips are from 9 km short to 367 kms long.
- From visual analysis it has been recognised that there are outliers in almost all the variables.
- The trips are spread across different states and coridors. Among them, Karnataka, Maharashtra, Tamilnadu and Haryana are having the greatest number of trips (either as source or as destination). Contributing 40% of the total trips.
- The time taken for completion of trip is found to be following a right skewed distribution in the distribution plot. Both the actual time and osrm time are following similar distribution in histogram. This shows that there is a higher number of shorter trips than longer trips.
- Similarly, the distribution of distance of trips, both actual and osrm are found to be following a right skewed distribution.

- Among different route types, carting is found to be more used for shorter trips when compared to FTL.

2) Feature Creation

The start time ‘od_start_time’ and end time ‘od_end_time’ are of object datatype. They are converted to datetime format by using ‘to_datetime’ function.

```
[30] df1['od_start_time']=pd.to_datetime(df1['od_start_time'])
     df1['od_end_time']=pd.to_datetime(df1['od_end_time'])
```

The difference between start time and end time is calculated and the hour difference is saved as ‘od_time_diff_hour’

```
[31] df1['od_time_diff_hour']=(df1['od_end_time']-df1['od_start_time']).dt.total_seconds() / 3600
```

The source and destination states are extracted from the source and destination name by using apply function. Similarly, source and destination city also extracted and created additional columns in the name source_state,destination_state,source_city and destination_city respectively.

```
[10] df1['source_state']=df1['source_name'].apply(lambda X: X.split('(')[1].split(')')[0])
     df1['source_city']=df1['source_name'].apply(lambda X: X.split('_')[0])
     df1['destination_state']=df1['destination_name'].apply(lambda X: X.split('(')[1].split(')')[0])
     df1['destination_city']=df1['destination_name'].apply(lambda X: X.split('_')[0])
```

```
df1
```

:ime	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance	od_time_diff_hour	source_state	source_city	destination_state	destination_city
94.0	544.8027	820.0	474.0	649.8528	16.658423	Madhya Pradesh	Bhopal	Uttar Pradesh	Kanpur
49.0	446.5496	728.0	534.0	670.6205	21.010074	Uttar Pradesh	Kanpur	Haryana	Gurgaon
26.0	28.1994	46.0	26.0	28.1995	0.980540	Karnataka	Doddablpur	Karnataka	Chikblapur
42.0	56.9116	95.0	39.0	55.9899	2.046325	Karnataka	Tumkur	Karnataka	Doddablpur
29.0	2090.8743	2700.0	1710.0	2227.5270	51.662060	Karnataka	Bangalore	Haryana	Gurgaon
...	...	...	...	...	...	...	...	...	...
31.0	25.7087	32.0	30.0	25.7087	0.757975	Tamil Nadu	Thisayanvilai	Tamil Nadu	Peikulam
41.0	42.5213	49.0	42.0	42.1431	1.035253	Tamil Nadu	Tirchchndr	Tamil Nadu	Thisayanvilai
50.0	52.8070	59.0	58.0	61.0753	1.760949	Tamil Nadu	Tirunelveli	Tamil Nadu	Eral
26.0	28.0484	41.0	25.0	28.0484	1.115559	Karnataka	Hospet (Karnataka)	Karnataka	Sandur

The trip creation time is converted into datetime format from object data type.

```
[13] df1['trip_creation_time']=pd.to_datetime(df1['trip_creation_time'])
```

The date of trip creation is used to extract useful data like trip creation year, month and day. These data are saved under additional columns in the name trip_creation_year, trip_creation_month and trip_creation_day respectively.

```
[15] df1['trip_creation_year']=df1['trip_creation_time'].dt.year
     df1['trip_creation_month']=df1['trip_creation_time'].dt.month
     df1['trip_creation_day']=df1['trip_creation_time'].dt.day
```

```
[17] df1.head()
```

segment_osrm_distance	od_time_diff_hour	source_state	source_city	destination_state	destination_city	trip_creation_year	trip_creation_month	trip_creation_day
649.8528	16.658423	Madhya Pradesh	Bhopal	Uttar Pradesh	Kanpur	2018	9	12
670.6205	21.010074	Uttar Pradesh	Kanpur	Haryana	Gurgaon	2018	9	12
28.1995	0.980540	Karnataka	Doddablpur	Karnataka	Chikblapur	2018	9	12
55.9899	2.046325	Karnataka	Tumkur	Karnataka	Doddablpur	2018	9	12
2227.5270	51.662060	Karnataka	Bangalore	Haryana	Gurgaon	2018	9	12

3) Merging of rows and aggregation of fields

The dataframe contains details of trip with segments which shows how many kms are remaining to the destination, time required to reach destinations and time and distance of each segment. Thus, the dataframe contains multiple fields for the same trip. The data can be grouped on basis of trip_uuid in order to aggregate the data regarding different trips across segments.

The data is grouped by keeping the relevant columns like source center, destination center, starts can to end scan etc. aggregating the time and distances using max and sum functions.

```
df1=df.groupby(['trip_uuid',’trip_creation_time’,'source_name','source_center',’destination_name’,'destination_center','data','route_type','od_start_time','od_end_time','start_scan_to_end_scan']).aggregate({'actual_distance_to_destination':'max','actual_time':'max','osrm_time':'max','osrm_distance':'max','segment_actual_time':'sum','segment_osrm_time':'sum','segment_osrm_distance':'sum'})
```

```
[29] df1.reset_index(inplace=True)
     df1.head()
```

	trip_uuid	trip_creation_time	source_name	source_center	destination_name	destination_center	data	route_type	od_start_time
0	trip-153671041653548748	2018-09-12 00:00:16.535741	Bhopal_Trnsport_H (Madhya Pradesh)	IND462022AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741
1	trip-153671041653548748	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	training	FTL	2018-09-12 16:39:46.858469
2	trip-153671042288605164	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	Chikblapur_ShntiSgr_D (Karnataka)	IND562101AAA	training	Carting	2018-09-12 02:03:09.655591
3	trip-153671042288605164	2018-09-12 00:00:22.886430	Tumkur_Veersagr_I (Karnataka)	IND572101AAA	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430
4	trip-153671043369099517	2018-09-12 00:00:33.691250	Bangalore_Nelmngla_H (Karnataka)	IND562132AAA	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	training	FTL	2018-09-12 00:00:33.691250

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_distanc
2018-09-12 16:39:46.858469	999.0	440.973689	830.0	394.0	544.8027	820.0	474.0	649.852
2018-09-13 13:40:23.123744	1260.0	383.759164	732.0	349.0	446.5496	728.0	534.0	670.620
2018-09-12 03:01:59.598855	58.0	24.644021	47.0	26.0	28.1994	46.0	26.0	28.199
2018-09-12 02:03:09.655591	122.0	48.542890	96.0	42.0	56.9116	95.0	39.0	55.989
2018-09-14 03:40:17.106733	3099.0	1689.964663	2736.0	1529.0	2090.8743	2700.0	1710.0	2227.527

Here the actual distance to destination and osrm distance and corresponding time taken can be compared.

4) Comparison & Visualization of time and distance fields

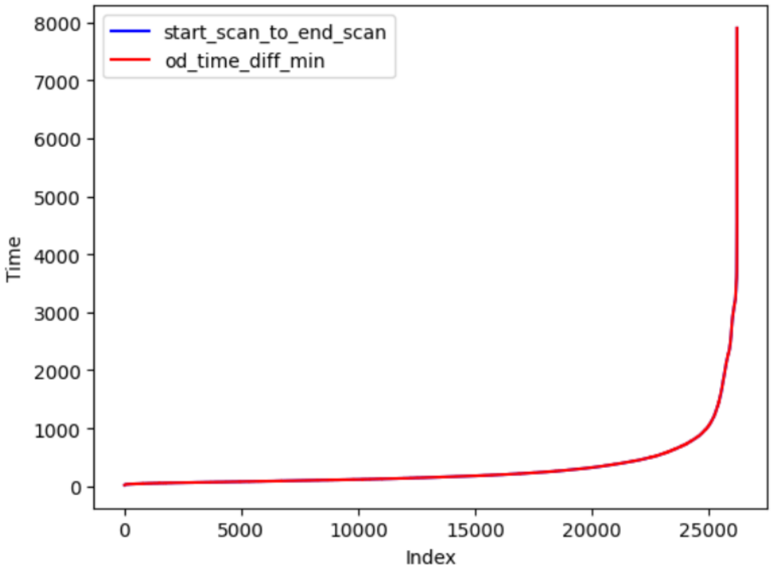
The od_time_diff_hour column is converted into minutes because all the other time columns are in minutes.

```
[27] df1['od_time_diff_min']=df1['od_time_diff_hour']*60
```

This value is compared with start scan to end scan time.

```
[42] dff=df1.sort_values(by='start_scan_to_end_scan')
     dff.reset_index(inplace=True)
```

```
sns.lineplot(data=dff,x=dff.index,y='start_scan_to_end_scan',label='start_scan_to_end_scan',color='blue')
sns.lineplot(data=dff,x=dff.index,y='od_time_diff_min',label='od_time_diff_min',color='red')
plt.xlabel('Index')
plt.ylabel('Time')
plt.legend()
plt.show()
```



In visual analysis, these values don't show any difference. Hypothesis testing is done to check this inference.

H_0 = The od_time_diff and start_scan_to_end_scan time are same

H_a = There is significant difference between the time intervals.

t test is done to test the hypothesis:

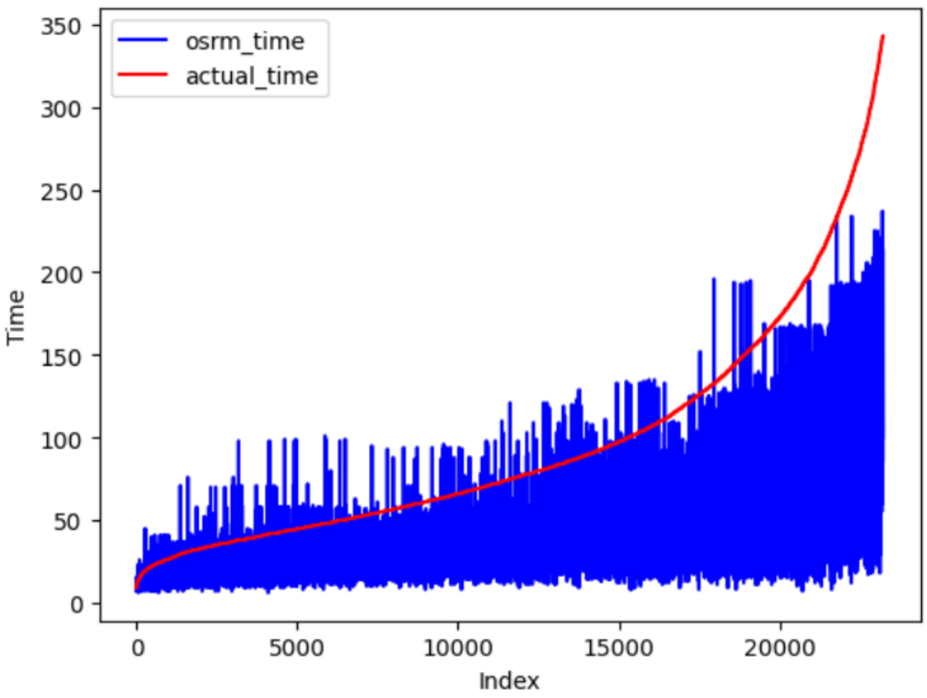
```
[29] ttest_ind(df1['od_time_diff_min'],df1['start_scan_to_end_scan'])

TtestResult(statistic=0.12947850360324997, pvalue=0.8969795299490583, df=52444.0)
```

The p-value is 0.897. For a confidence level of 95%, the test failed to reject null hypothesis. Hence, there is no significant difference between od_time_diff and start_scan_to_end_scan.

The actual time and osrm_time columns are compared visually:

```
[ ] dff=df1.sort_values(by='actual_time')
    dff.reset_index(inplace=True)
```

```
sns.lineplot(data=dff,x=dff.index,y='osrm_time',label='osrm_time',color='blue')
sns.lineplot(data=dff,x=dff.index,y='actual_time',label='actual_time',color='red')
plt.xlabel('Index')
plt.ylabel('Time')
plt.legend()
plt.show()
```



It is observed that osrm time is generally lesser than actual time. Hypothesis testing is used to test this hypothesis.

H_0 = The osrm_time and actual_time are same

H_a = There is significant difference between osrm_time and actual time

t-test for independent variables is done:

```
[40] ttest_ind(df1['osrm_time'],df1['actual_time'])
```

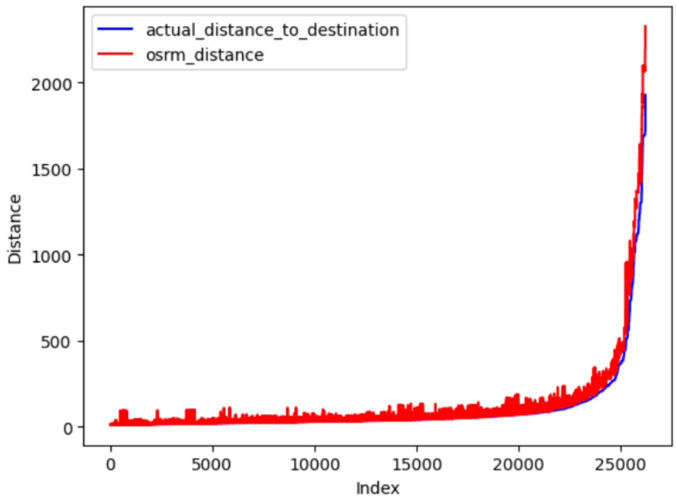
```
TtestResult(statistic=-41.49451885578905, pvalue=0.0, df=52444.0)
```

p-value is zero that is, null hypothesis is rejected and there is significant difference between osrm_time and actual time. The test statistics is negative that shows that osrm_time is lesser than actual time in general.

The actual_distance_to_destination and osrm_distance columns are compared visually:

```
[44] dff=df1.sort_values(by='actual_distance_to_destination')
     dff.reset_index(inplace=True)
```

```
sns.lineplot(data=dff,x=dff.index,y='actual_distance_to_destination',label='actual_distance_to_destination',color='blue')
sns.lineplot(data=dff,x=dff.index,y='osrm_distance',label='osrm_distance',color='red')
plt.xlabel('Index')
plt.ylabel('Distance')
plt.legend()
plt.show()
```



It is observed that osrm distance is generally higher than actual distance. Hypothesis testing is used to test this hypothesis.

H_0 = The osrm_distance and actual_distance_to_destination are same

H_a= There is significant difference between osrm_distance and actual_distance_to_destination

t-test for independent variables is done:

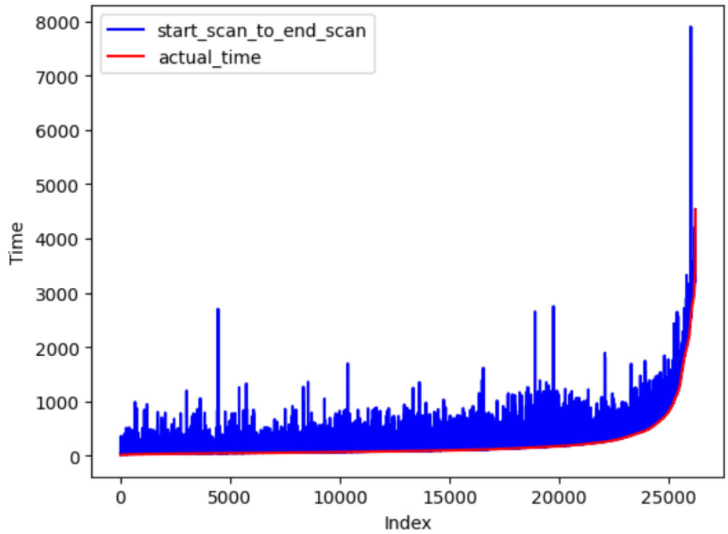
```
[46] ttest_ind(df1['osrm_distance'],df1['actual_distance_to_destination'])

TtestResult(statistic=11.159639893008887, pvalue=6.925536775666083e-29, df=52444.0)
```

p-value is very close to zero that is, null hypothesis is rejected and there is significant difference between osrm_distance and actual_distance_to_destination. The test statistics is negative that shows that osrm_distance is lesser than actual_distance_to_destination in general.

Actual time and start scan to end scan is compared. The time taken between start scan and end scan will always be higher than the actual_time which is the time taken for transit alone. Visual analysis is done:

```
[37] dff=df1.sort_values(by='actual_time')
     dff.reset_index(inplace=True)
```

```
[38] sns.lineplot(data=dff,x=dff.index,y='start_scan_to_end_scan',label='start_scan_to_end_scan',color='blue')
     sns.lineplot(data=dff,x=dff.index,y='actual_time',label='actual_time',color='red')
     plt.xlabel('Index')
     plt.ylabel('Time')
     plt.legend()
     plt.show()
```



From visual analysis it is obvious that the start-end scan time is higher than actual time of transit. T-test between the two gave p-value of zero.

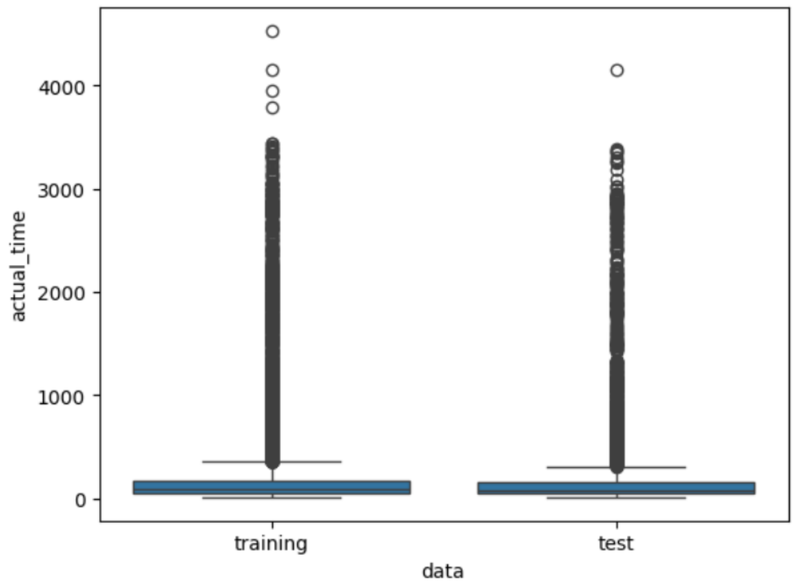
```
[41] ttest_rel(df1['actual_time'],df1['start_scan_to_end_scan'])
```

```
TtestResult(statistic=-116.68172665963952, pvalue=0.0, df=26222)
```

5) Missing values Treatment & Outlier treatment

The missing values are observed in the fields of source name and destination name. Which doesn't affect analysis of the data.

```
sns.boxplot(data=df1,x='data',y='actual_time')
<Axes: xlabel='data', ylabel='actual_time'>
```



The outliers are found out by using IQR method.

The outliers in total time to destination and total distance to destination in the cumulative dataframe is found out using equations for q1,q3 and IQR. Now the upper limit is calculated by using the formula

Upper_limit = q3 + 1.5 * IQR

```
[11] q1t=df1['actual_time'].quantile(0.25)
     q3t=df1['actual_time'].quantile(0.75)
     iqrt=q3t-q1t
     upper_limit_t=q3t+1.5*iqrt
```

```
[12] df1=df1[df1['actual_time']<upper_limit_t]
```

The outliers are removed from the data.

```
[13] q1d=df1['actual_distance_to_destination'].quantile(0.25)
     q3d=df1['actual_distance_to_destination'].quantile(0.75)
     iqrd=q3d-q1d
     upper_limit_d=q3d+1.5*iqrd
```

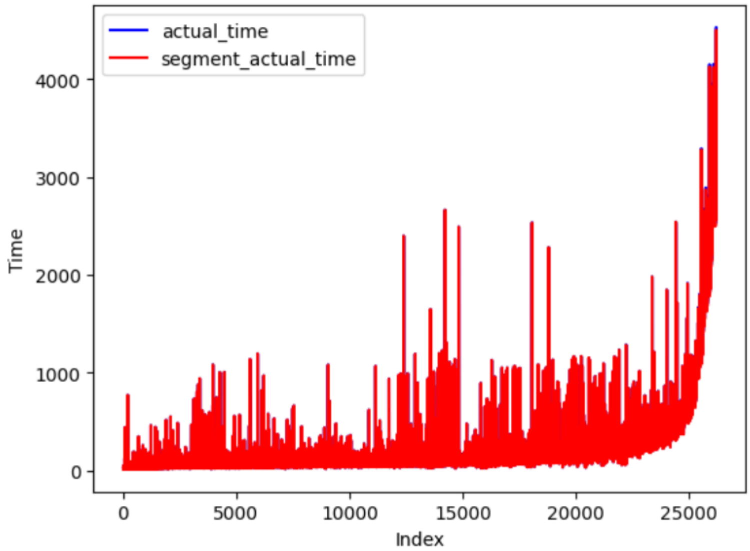
```
[14] df1=df1[df1['actual_distance_to_destination']<upper_limit_t]
```

6) Checking relationship between aggregated fields

Aggregate segment_actual_time and actual time is compared. In visual comparison, the aggregate segment_actual_time varies about actual time and is not possible to draw out a comparison.

```
[ ] dff=df1.sort_values(by='actual_time')
    dff.reset_index(inplace=True)
```

```
[54] sns.lineplot(data=dff,x=dff.index,y='actual_time',label='actual_time',color='blue')
     sns.lineplot(data=dff,x=dff.index,y='segment_actual_time',label='segment_actual_time',color='red')
     plt.xlabel('Index')
     plt.ylabel('Time')
     plt.legend()
     plt.show()
```



Hypothesis testing is used to compare the same.

H_0 = Aggregate segment_actual_time and actual time are same.

H_a = There is significant difference between them

t-test is used.

```
ttest_ind(df1['actual_time'],df1['segment_actual_time'])
```

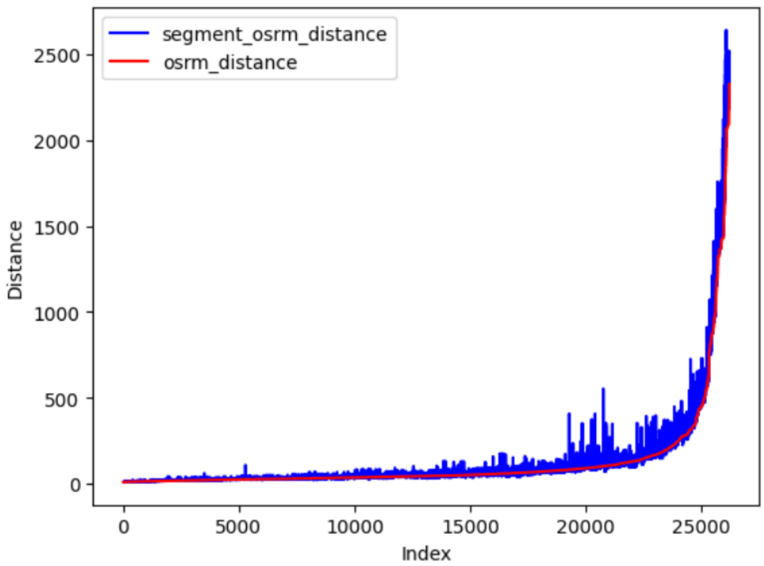
```
TtestResult(statistic=0.5477570535931394, pvalue=0.5838610621303091, df=52444.0)
```

The p-value is higher than α for a significance level of 90%. Hence failed to reject null hypothesis. The aggregate segment_actual_time and actual time are same.

Aggregate segment_osrm_distance and aggregate osrm distance are compared. In visual comparison, segment_osrm_distance is found to be generally higher than osrm_distance.

```
[50] dff=df1.sort_values(by='osrm_distance')
     dff.reset_index(inplace=True)
```

```
[51] sns.lineplot(data=dff,x=dff.index,y='segment_osrm_distance',label='segment_osrm_distance',color='blue')
     sns.lineplot(data=dff,x=dff.index,y='osrm_distance',label='osrm_distance',color='red')
     plt.xlabel('Index')
     plt.ylabel('Distance')
     plt.legend()
     plt.show()
```



Hypothesis testing using t-test is done:

H_0 = aggregate osrm_distance and aggregate_sefment_osrm distance are same.

H_a = there is significant difference between them.

```
[52] ttest_ind(df1['osrm_distance'],df1['segment_osrm_distance'])
```

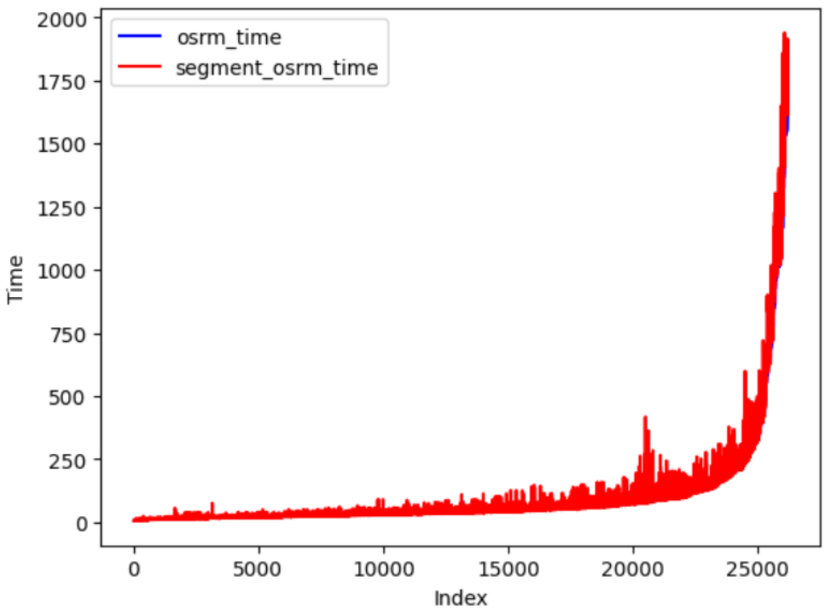
```
TtestResult(statistic=-4.3016929613942825, pvalue=1.698042984329447e-05, df=52444.0)
```

p-value is less than α for a confidence level of 95%. Thus, null hypothesis is rejected and the stat value shows that the segment_osrm_distance larger than osrm_distance.

Aggregate segment_osrm_time and osrm_time is compared. In visual comparison, the aggregate segment_ osrm_time varies about osrm_time and is not possible to draw out a comparison.

```
[57] dff=df1.sort_values(by='osrm_time')
     dff.reset_index(inplace=True)
```

```
[58] sns.lineplot(data=dff,x=dff.index,y='osrm_time',label='osrm_time',color='blue')
     sns.lineplot(data=dff,x=dff.index,y='segment_osrm_time',label='segment_osrm_time',color='red')
     plt.xlabel('Index')
     plt.ylabel('Time')
     plt.legend()
     plt.show()
```



Hypothesis testing is used to compare the same.

H_0 = Aggregate segment_ osrm_time and osrm_time are same.

H_a = There is significant difference between them

t-test is used.

```
[59] ttest_ind(df1['osrm_time'],df1['segment_osrm_time'])
```

```
TtestResult(statistic=-6.0285565458936, pvalue=1.6653197347173302e-09, df=52444.0)
```

The p-value is lower than α for a significance level of 95%. Hence rejected null hypothesis. There is significant difference between aggregate segment_actual_time and actual time. The stat value is less than zero that implies aggregate segment_osrm_time is greater.

7) Handling categorical values

One-hot encoding is used for representing categorical variables. The categorical columns in which encoding is used are data and route_type. Get_dummies() function is used for one-hot encoding of these categorical columns.

```
[60] df2=pd.get_dummies(df1,columns=['route_type','data'])
     df2
```

on_city	trip_creation_year	trip_creation_month	trip_creation_day	start_to_end_hrs	od_time_diff_min	route_type_Carting	route_type_FTL	data_test	data_training
Kanpur	2018	9	12	16.650000	999.505379	False	True	False	True
Gurgaon	2018	9	12	21.000000	1260.604421	False	True	False	True
ikblapur	2018	9	12	0.966667	58.832388	True	False	False	True
ıdablpur	2018	9	12	2.033333	122.779486	True	False	False	True
Gurgaon	2018	9	12	51.650000	3099.723591	False	True	False	True

Now, these columns are in Boolean data type. astype function is used to change the data type into integer.

```
df2['route_type_FTL']=df2['route_type_FTL'].astype(int)
df2['route_type_Carting']=df2['route_type_Carting'].astype(int)
df2['data_training']=df2['data_training'].astype(int)
df2['data_test']=df2['data_test'].astype(int)
df2
```

on_city	trip_creation_year	trip_creation_month	trip_creation_day	start_to_end_hrs	od_time_diff_min	route_type_Carting	route_type_FTL	data_test	data_training
Kanpur	2018	9	12	16.650000	999.505379	0	1	0	1
Gurgaon	2018	9	12	21.000000	1260.604421	0	1	0	1
ikblapur	2018	9	12	0.966667	58.832388	1	0	0	1
ıdablpur	2018	9	12	2.033333	122.779486	1	0	0	1
Gurgaon	2018	9	12	51.650000	3099.723591	0	1	0	1

8) Column Normalization /Column Standardization

Minmax scaler is used for standardize the numerical columns in the data frame. Minmax scaler is a tool from sklearn library.

```
#import minmax scaler
from sklearn.preprocessing import MinMaxScaler
```

```
[68]
# instantiate the MinMaxScaler
min_max=MinMaxScaler()
```

Now the minmax scaler is used to get standardised values for all the numerical columns in the dataframe.

```
[72] df2['start_scan_to_end_scan']=min_max.fit_transform(df2[['start_scan_to_end_scan']])
     df2['od_time_diff_min']=min_max.fit_transform(df2[['od_time_diff_min']])
     df2['actual_time']=min_max.fit_transform(df2[['actual_time']])
     df2['actual_distance_to_destination']=min_max.fit_transform(df2[['actual_distance_to_destination']])
     df2['osrm_time']=min_max.fit_transform(df2[['osrm_time']])
     df2['osrm_distance']=min_max.fit_transform(df2[['osrm_distance']])
     df2['segment_actual_time']=min_max.fit_transform(df2[['segment_actual_time']])
     df2['segment_osrm_time']=min_max.fit_transform(df2[['segment_osrm_time']])
     df2['segment_osrm_distance']=min_max.fit_transform(df2[['segment_osrm_distance']])
```

The output is:

```
df2
```

_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance	..
0.124270	0.225168	0.181517	0.230952	0.231204	0.180423	0.242236	0.243471	..
0.157400	0.195344	0.159850	0.204167	0.188801	0.159956	0.273292	0.251362	..
0.004824	0.008154	0.008402	0.011905	0.008254	0.008231	0.010352	0.007267	..
0.012947	0.020611	0.019235	0.021429	0.020646	0.019132	0.017081	0.017827	..
0.390835	0.876211	0.602918	0.906548	0.898441	0.598665	0.881988	0.842925	..

9) Business Insights

- The time duration between od_start time and od_end time is found to be equal to the time of start_scan_to_end_scan.

- The actual time for a trip completion is found to be significantly higher than the open-source routing engine calculated time (osrm_time). This indicates the trip completion in actual takes more than the shortest time required for the trip. This can be due to trip segmentation and related delays.

- Further, the actual distance to destination and the open-source routing engine calculated distance are found to be different. The actual distance being significantly higher than the osrm distance. This adds to the above mentioned insight.

- Actual time taken for the trip and time between the start and end scans are compared and the latter is found to be significantly larger than the former.

- The actual time taken for the trip and aggregate segment actual time are found to be equal. This shows that the latency in segmentation is taken into account in the actual time taken for the trip.

- The aggregate segment osrm time is found to be significantly larger than the osrm time for the trip. This shows that the osrm engine contributes to inaccuracy while going through segmentation.

- In analysis, the major contributors to the trips in terms of source and destination state are found to be Karnataka and Maharashtra, contributing about 27% of all the trips.

- Busiest corridors are Bengaluru-Bengaluru and Bhivandi-Mumbai. Both are intra-state and intra-city trips. the average time taken for intra-Bangalore trips is 81.74 minutes while for the latter is 80.12.

- In comparison FTL has a greater number of trips than carting. While the actual time and actual distance have shown higher in the case of FTL. FTL is found to be providing slightly faster trip completion than Carting.

2) Recommendations.

- Integrate real-time traffic and environmental data to refine routing algorithms. This could help in reducing discrepancies between actual and calculated distances/times.
- Custom Routing Solutions: Consider developing or adopting a proprietary routing engine that factors in real-world conditions more accurately, especially for segmented trips.
- Review Segmentation Practices: Analyze the causes of segmentation delays and explore solutions such as streamlining processes or increasing vehicle readiness.
- Utilize advanced GPS and tracking technology to monitor trip progress and identify delays in real-time.
- Focus on Major Corridors: Prioritize enhancements on the busiest corridors (Bengaluru-Bengaluru and Bhivandi-Mumbai) to improve average trip times, potentially through dedicated lanes or optimized traffic signals.
- Provide training for drivers on efficient navigation and time management, particularly for intra-city trips.
- Analyze High-Volume Regions: Given that Karnataka and Maharashtra are the major contributors to trips, consider localized strategies that cater specifically to these states, such as targeted marketing campaigns or improved service offerings.

- Regularly analyze trip completion times and distances to identify trends and optimize performance across different segments.
- Fleet Optimization: Assess fleet composition and utilization to ensure that the right vehicles are used for the appropriate trip types (FTL vs. carting). Explore expanding the FTL fleet if it consistently shows better efficiency.
- Implement predictive maintenance programs to reduce vehicle downtime and delays.
- Real-time Updates: Improve customer communication by providing real-time updates on trip status and potential delays, which could enhance customer satisfaction and trust.