🚚DELHIVERY - Business CaseStudy🚚

Feature Engineering

Analysed by : **PAVAN ELETI**

Introduction:

- 🚚 **Delhivery**, established in 2011, is India's foremost logistics and supply chain service provider, offering a comprehensive range of solutions including express parcel transportation, warehousing, and last-mile delivery.
- Leveraging advanced technology and a vast delivery network, Delhivery efficiently manages nationwide movement of goods, earning trust across businesses of all sizes for its dedication to innovation and customer satisfaction.
- As the largest fully integrated player in India by revenue in Fiscal 2021, Delhivery aims to lead the industry by pioneering the commerce operating system, driven by top-tier infrastructure, logistics operations, and innovative data intelligence initiatives led by its Data team.

🔷 Why this case study?

> Delhivery aims to establish itself as the premier player in the logistics industry. This case study is of paramount importance as it aligns with the company's core objectives and operational excellence.

> It provides a practical framework for understanding and processing data, which is integral to their operations. By leveraging data engineering pipelines and data analysis techniques, Delhivery can achieve several critical goals.

> First, it allows them to ensure data integrity and quality by addressing missing values and structuring the dataset appropriately.

> Second, it enables the extraction of valuable features from raw data, which can be utilized for building accurate forecasting models.

> Moreover, it facilitates the identification of patterns, insights, and actionable recommendations crucial for optimizing their logistics operations.

> Byconducting hypothesis testing and outlier detection, Delhivery can refine their processes and further enhance the quality of service they provide.

How can you help here?

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.

📃 Features of the dataset:

- Column Profiling:

Feature	Description
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**
a. FTL–Full Truck Load	FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-of
b. 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_center	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 poir
osrm_distance	An open-source routing engine which computes the shortest path between points in a given ma

Feature	Description
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

```
# importing the required modules and packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import re
from scipy.stats import norm,zscore,boxcox,probplot
from scipy.stats import ttest_ind,ttest_rel,mannwhitneyu,wilcoxon
from scipy.stats import shapiro,levene,kstest,anderson
import statsmodels.api as sm
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler , MinMaxScaler , OneHotEncoder
import warnings
warnings.filterwarnings('ignore')


!gdown 1DzFkNxhyWLo-v4Uco3B3rFdxEbgxQSVI
```

```
Downloading...
From: https://drive.google.com/uc?id=1DzFkNxhyWLo-v4Uco3B3rFdxEbgxQSVI
To: /content/delhivery_data.csv
100% 55.6M/55.6M [00:00<00:00, 70.6MB/s]
```

```
# pd_reading the data
delhivery_data = pd.read_csv('delhivery_data.csv')


# setting the option of displaying all the columns
pd.set_option('display.max_columns', 50)


# making a deep copy for backup
dd = delhivery_data.copy()
dd.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uu
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	tr 1537410936476493
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	tr 1537410936476493
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	tr 1537410936476493
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	tr 1537410936476493
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	tr 1537410936476493

🧐Exploration of data :

```
dd.shape
```

```
(144867, 24)
```

```
dd.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', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

```
dd.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
```

📝 Statistical Summary

```
dd.describe().T
```

	count	mean	std	min	25%
start_scan_to_end_scan	144867.0	961.262986	1037.012769	20.000000	161.000000
cutoff_factor	144867.0	232.926567	344.755577	9.000000	22.000000
actual_distance_to_destination	144867.0	234.073372	344.990009	9.000045	23.355874
actual_time	144867.0	416.927527	598.103621	9.000000	51.000000
osrm_time	144867.0	213.868272	308.011085	6.000000	27.000000
osrm_distance	144867.0	284.771297	421.119294	9.008200	29.914700
factor	144867.0	2.120107	1.715421	0.144000	1.604264
segment_actual_time	144867.0	36.196111	53.571158	-244.000000	20.000000
segment_osrm_time	144867.0	18.507548	14.775960	0.000000	11.000000
segment_osrm_distance	144867.0	22.829020	17.860660	0.000000	12.070100

```
dd.describe(include=object).T
```

	count	unique	top	freq
data	144867	2	training	104858
trip_creation_time	144867	14817	2018-09-28 05:23:15.359220	101
route_schedule_uuid	144867	1504	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	1812
route_type	144867	2	FTL	99660
trip_uuid	144867	14817	trip-153811219535896559	101
source_center	144867	1508	IND000000ACB	23347
source_name	144574	1498	Gurgaon_Bilaspur_HB (Haryana)	23347
destination_center	144867	1481	IND000000ACB	15192
destination_name	144606	1468	Gurgaon_Bilaspur_HB (Haryana)	15192
od_start_time	144867	26369	2018-09-21 18:37:09.322207	81
od_end_time	144867	26369	2018-09-24 09:59:15.691618	81
cutoff_timestamp	144867	93180	2018-09-24 05:19:20	40

📑📑Duplicate Detection

```
dd[dd.duplicated()]
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_

🏷️ Insights

- The dataset does not contain any duplicates.

🔶❓ Null Detection

```
dd.isna().any()
```

```
data                      False
trip_creation_time        False
route_schedule_uuid       False
route_type                False
trip_uuid                 False
source_center             False
source_name                True
destination_center        False
destination_name           True
od_start_time             False
od_end_time               False
start_scan_to_end_scan    False
```

```
is_cutoff                         False
cutoff_factor                     False
cutoff_timestamp                  False
actual_distance_to_destination    False
actual_time                       False
osrm_time                         False
osrm_distance                     False
factor                            False
segment_actual_time               False
segment_osrm_time                 False
segment_osrm_distance             False
segment_factor                    False
dtype: bool
```

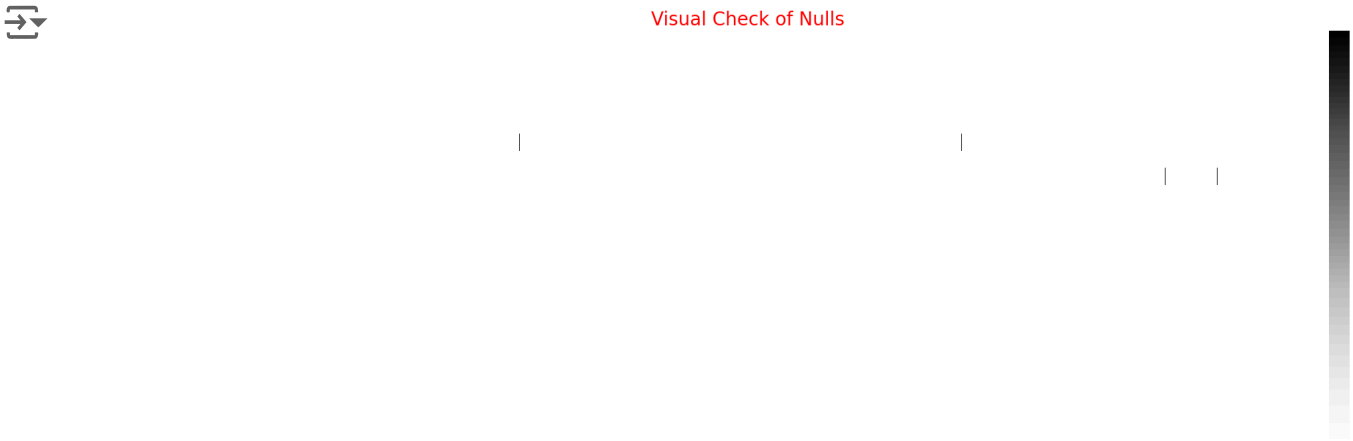
```
dd.isnull().sum()
```

```
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
```

```
def missing_data(df):
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isna().count()*100).sort_values(as
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, k
    return missing_data_df

missing_pct = missing_data(dd)
missing_pct[missing_pct['Total']>0]
```

	Total	Percent
source_name	293	0.202254
destination_name	261	0.180165

```
plt.figure(figsize=(25,8))
plt.style.use('dark_background')
sns.heatmap(dd.isnull().T,cmap='Greys')
plt.title('Visual Check of Nulls',fontsize=20,color='r')
plt.show()
```




```
# Dropping unknown fields

unknown_fields = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'se
dd = dd.drop(columns = unknown_fields)


dd.sample()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip
79715	training	2018-09-23 09:14:49.745585	thanos::sroute:f6912dfd-a9cf-4ea7-a7ce-1403d4a...	FTL	153769408974

```
dd.shape
```

```
(144867, 19)
```

```
#checking the unique values for columns
for _ in dd.columns:
    print()
    print(f'Total Unique Values in {_} column are :- {dd[_].nunique()}')
    print(f'Unique Values in {_} column are :-\n {dd[_].unique()}')
    print()
    print('-'*120)
```

```

Total Unique Values in data column are :- 2
Unique Values in data column are :-
 ['training' 'test']

------------------------------------------------------------------------------------------------------------------------

Total Unique Values in trip_creation_time column are :- 14817
Unique Values in trip_creation_time column are :-
 ['2018-09-20 02:35:36.476840' '2018-09-23 06:42:06.021680'
 '2018-09-14 15:42:46.437249' ... '2018-09-22 11:30:41.399439'
 '2018-09-17 11:35:28.838714' '2018-09-20 16:24:28.436231']

------------------------------------------------------------------------------------------------------------------------

Total Unique Values in route_schedule_uuid column are :- 1504
Unique Values in route_schedule_uuid column are :-
 ['thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3297ef'
 'thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc211728881b'
 'thanos::sroute:a16bfa03-3462-4bce-9c82-5784c7d315e6' ...
 'thanos::sroute:72cf9feb-f4e3-4a55-b92a-0b686ee8fabc'
 'thanos::sroute:5e08be79-8a4c-4a91-a514-5350403c0e31'
 'thanos::sroute:a3c30562-87e5-471c-9646-0ed49c150996']

------------------------------------------------------------------------------------------------------------------------

Total Unique Values in route_type column are :- 2
Unique Values in route_type column are :-
 ['Carting' 'FTL']

------------------------------------------------------------------------------------------------------------------------

Total Unique Values in trip_uuid column are :- 14817
Unique Values in trip_uuid column are :-
 ['trip-153741093647649320' 'trip-153768492602129387'
 'trip-153693976643699843' ... 'trip-153761584139918815'
 'trip-153718412883843340' 'trip-153746066843555182']
```

```
--------------------------------------------------------------------------------

Total Unique Values in source_center column are :- 1508
Unique Values in source_center column are :-
 ['IND388121AAA' 'IND388620AAB' 'IND421302AAG' ... 'IND361335AAA'
 'IND562132AAC' 'IND639104AAB']

--------------------------------------------------------------------------------

Total Unique Values in source_name column are :- 1498
Unique Values in source_name column are :-
 ['Anand_VUNagar_DC (Gujarat)' 'Khambhat_MotvdDPP_D (Gujarat)'
 'Bhiwandi_Mankoli_HB (Maharashtra)' ... 'Dwarka_StnRoad_DC (Gujarat)'
 'Bengaluru_Nelmngla_L (Karnataka)' 'Kulithalai_AnnaNGR_D (Tamil Nadu)']

--------------------------------------------------------------------------------

Total Unique Values in destination center column are :- 1481
```

🔄 Changing the Datatype of Columns

```
dd.sample()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip
25373	training	2018-09-23 05:04:13.738416	thanos::sroute:25dd334d-e30d-4cfa-b6e8-192bcfe...	FTL	153767905373

```
dd.dtypes
```

```
data                            object
trip_creation_time              object
route_schedule_uuid             object
route_type                      object
trip_uuid                       object
source_center                   object
source_name                     object
destination_center              object
destination_name                object
od_start_time                   object
od_end_time                     object
start_scan_to_end_scan         float64
actual_distance_to_destination float64
actual_time                    float64
osrm_time                      float64
osrm_distance                  float64
segment_actual_time            float64
segment_osrm_time              float64
segment_osrm_distance          float64
dtype: object
```

```
# Converting the datatypes to category for columns like data and route_type as th
dd['data'] = dd['data'].astype('category')
dd['route_type'] = dd['route_type'].astype('category')

# Converting time columns to datetime format
datetime_cols = ['trip_creation_time', 'od_start_time', 'od_end_time']
for _ in datetime_cols:
    dd[_] = pd.to_datetime(dd[_])


dd.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  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  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: category(2), datetime64[ns](3), float64(8), object(6)
memory usage: 19.1+ MB
```

```
float_cols = []
for _ in dd.columns:
    if isinstance(dd[_],'float64'):
        float_cols.append(_)
float_cols
```

see y it didnt work

```
float_cols = []
for _ in dd.columns:
    if dd[_].dtype=='float64':
        float_cols.append(_)
float_cols
```

```
['start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance']
```

```
# reducing the float64 to float32 to save memory
for _ in float_cols:
    dd[_] = dd[_].astype('float32')
```

```
dd.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  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144574 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144606 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float32
 12  actual_distance_to_destination  144867 non-null  float32
 13  actual_time                     144867 non-null  float32
 14  osrm_time                       144867 non-null  float32
 15  osrm_distance                   144867 non-null  float32
 16  segment_actual_time             144867 non-null  float32
 17  segment_osrm_time               144867 non-null  float32
 18  segment_osrm_distance           144867 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(8), object(6)
memory usage: 14.6+ MB
```

Insights:

- Earlier the dataset was using 25.6+ MB of memory but now it has been reduced to 14.6 + MB. Around 40.63 % reduction in the memory usage.

```
# Time period of data
dd['trip_creation_time'].max(), dd['trip_creation_time'].min() , dd['trip_creatio
```

```
(Timestamp('2018-10-03 23:59:42.701692'),
 Timestamp('2018-09-12 00:00:16.535741'),
 Timedelta('21 days 23:59:26.165951'))
```

```
# Time period of data
dd['od_start_time'].max(), dd['od_start_time'].min(), dd['od_start_time'].max() -
```

```
(Timestamp('2018-10-06 04:27:23.392375'),
 Timestamp('2018-09-12 00:00:16.535741'),
 Timedelta('24 days 04:27:06.856634'))
```

```
# Time period of data
dd['od_end_time'].max(), dd['od_end_time'].min(), dd['od_end_time'].max() - dd['o
```

```
(Timestamp('2018-10-08 03:00:24.353479'),
 Timestamp('2018-09-12 00:50:10.814399'),
 Timedelta('26 days 02:10:13.539080'))
```

```
data_time_frame =  dd['od_end_time'].max() - dd['trip_creation_time'].min()
data_time_frame
```

```
Timedelta('26 days 03:00:07.817738')
```

🔴Null Treatment:

Replace null values in 'source_name' and 'destination_name' columns with 'unknown' through scikit imputation

columns_to_impute = ['source_name', 'destination_name']
imputer = SimpleImputer(strategy='constant', fill_value='unknown')
dd[columns_to_impute] = imputer.fit_transform(dd[columns_to_impute])

but 'unknown' transactions will be more and to be omitted while analyzing ... Hence `no use of imputing`....

```
dd[(dd.source_name.isna())&(dd.destination_name.isna())]
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip
68006	training	2018-09-26 22:21:56.619259	thanos::sroute:cfb575b8-df26-48f5-8427-6f48f9d...	FTL	153800051661
68007	training	2018-09-26 22:21:56.619259	thanos::sroute:cfb575b8-df26-48f5-8427-6f48f9d...	FTL	153800051661
68008	training	2018-09-26 22:21:56.619259	thanos::sroute:cfb575b8-df26-48f5-8427-6f48f9d...	FTL	153800051661

```
dd[dd.source_name.isna()]
```

	data	trip_creation_time	route_schedule_uuid	route_type	tri
112	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
113	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
114	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
115	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
116	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
...	...	...	...	...	
144484	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144485	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144486	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144487	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144488	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666

293 rows × 19 columns

```
dd[dd.destination_name.isna()]
```

	data	trip_creation_time	route_schedule_uuid	route_type	tri
110	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
111	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	15378655843
982	test	2018-10-01 20:56:18.155260	thanos::sroute:d0ebdacd-e09b-47d3-be77-c9c4a05...	FTL	15384273781
983	test	2018-10-01 20:56:18.155260	thanos::sroute:d0ebdacd-e09b-47d3-be77-c9c4a05...	FTL	15384273781
4882	training	2018-09-24 07:18:06.087341	thanos::sroute:2f43f11e-d3ba-4590-9355-82928e1...	FTL	15377734860
...	...	...	...	...	
144478	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144479	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144480	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144481	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666
144482	test	2018-10-03 09:06:06.690094	thanos::sroute:cbef3b6a-79ea-4d5e-a215-b558a70...	FTL	15385575666

261 rows × 19 columns

```
ddd = dd.copy()


missing_source_name = ddd.loc[ddd['source_name'].isnull(), 'source_center'].uniqu
missing_source_name
```

```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

```
missing_destination_name = ddd.loc[ddd['destination_name'].isnull(), 'destination
missing_destination_name
```

```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

```
# checking if element of one np.array isin another np.array

#np.all(df.loc[df['source_name'].isnull(), 'source_center'].isin(missing_destinat

np.in1d(missing_source_name, missing_destination_name).all()
```

```
False
```

```
for _ in missing_source_name:
    unique_source_name = ddd.loc[ddd['source_center'] == _, 'source_name'].unique
    if pd.isna(unique_source_name):
        print("Source Center :", _, "-" * 5, "Source Name :", 'NA')
    else :
        print("Source Center :", _, "-" * 5, "Source Name :", unique_source_name)
```

```
Source Center : IND342902A1B ----- Source Name : NA
Source Center : IND577116AAA ----- Source Name : NA
Source Center : IND282002AAD ----- Source Name : NA
Source Center : IND465333A1B ----- Source Name : NA
Source Center : IND841301AAC ----- Source Name : NA
Source Center : IND509103AAC ----- Source Name : NA
Source Center : IND126116AAA ----- Source Name : NA
Source Center : IND331022A1B ----- Source Name : NA
Source Center : IND505326AAB ----- Source Name : NA
Source Center : IND852118A1B ----- Source Name : NA
```

```
for _ in missing_destination_name:
    unique_destination_name = ddd.loc[ddd['destination_center'] == _, 'destinatio
    if pd.isna(unique_destination_name):
        print("Destination Center :", _, "-" * 5, "Destination Name :", 'NA')
    else :
        print("Destination Center :", _, "-" * 5, "Destination Name :", unique_de
```

```
Destination Center : IND342902A1B ----- Destination Name : NA
Destination Center : IND577116AAA ----- Destination Name : NA
Destination Center : IND282002AAD ----- Destination Name : NA
Destination Center : IND465333A1B ----- Destination Name : NA
Destination Center : IND841301AAC ----- Destination Name : NA
Destination Center : IND505326AAB ----- Destination Name : NA
Destination Center : IND852118A1B ----- Destination Name : NA
Destination Center : IND126116AAA ----- Destination Name : NA
Destination Center : IND509103AAC ----- Destination Name : NA
Destination Center : IND221005A1A ----- Destination Name : NA
Destination Center : IND250002AAC ----- Destination Name : NA
Destination Center : IND331001A1C ----- Destination Name : NA
Destination Center : IND122015AAC ----- Destination Name : NA
```

```
count = 1
for i in missing_destination_name:
    ddd.loc[ddd['destination_center'] == i, 'destination_name'] = ddd.loc[ddd['de
                                    'destination_name'].replace(np.nan, f'locati
    count += 1


d = {}
for i in missing_source_name:
    d[i] = ddd.loc[ddd['destination_center'] == i, 'destination_name'].unique()
for idx, val in d.items():
    if len(val) == 0:
        d[idx] = [f'location_{count}']
        count += 1
d2 = {}
for idx, val in d.items():
    d2[idx] = val[0]
for i, v in d2.items():
    print(i, v)
```

```
IND342902A1B location_1
IND577116AAA location_2
IND282002AAD location_3
IND465333A1B location_4
IND841301AAC location_5
IND509103AAC location_9
IND126116AAA location_8
IND331022A1B location_14
IND505326AAB location_6
IND852118A1B location_7
```

```
for i in missing_source_name:
    ddd.loc[ddd['source_center'] == i, 'source_name'] = ddd.loc[ddd['source_cente


ddd.source_name.value_counts()
```

```
source_name
Gurgaon_Bilaspur_HB (Haryana)            23347
Bangalore_Nelmngla_H (Karnataka)          9975
Bhiwandi_Mankoli_HB (Maharashtra)         9088
Pune_Tathawde_H (Maharashtra)             4061
Hyderabad_Shamshbd_H (Telangana)          3340
                                         ...
Badkulla_Central_DPP_1 (West Bengal)         1
Kasganj_BnkrGate_D (Uttar Pradesh)           1
Shahjhnpur_NavdaCln_D (Uttar Pradesh)        1
Jaunpur_Katghara_D (Uttar Pradesh)           1
Krishnanagar_AnadiDPP_D (West Bengal)        1
Name: count, Length: 1508, dtype: int64
```

```
ddd.destination_name.value_counts()
```

```
destination_name
Gurgaon_Bilaspur_HB (Haryana)        15192
Bangalore_Nelmngla_H (Karnataka)     11019
```

```
Bhiwandi_Mankoli_HB (Maharashtra)     5492
Hyderabad_Shamshbd_H (Telangana)      5142
Kolkata_Dankuni_HB (West Bengal)      4892
                                     ...
Vijayawada (Andhra Pradesh)              1
Ranaghat_ArickDPP_D (West Bengal)        1
Mumbai_Sanpada_CP (Maharashtra)          1
Delhi_Lajwanti (Delhi)                   1
Luxettipet_ShivaDPP_D (Telangana)        1
Name: count, Length: 1481, dtype: int64
```

even if we replace these nulls with some values, those are not gonna have any impact on the data.... so we can drop it as well..

```
261+290
```

```
551
```

```
len(delhivery_data)
```

```
144867
```

```
144867 - 551
```

```
144316
```

```
df = dd.dropna()
```

```
df.isna().sum().any()
```

```
False
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  category
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  category
 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  float32
 12  actual_distance_to_destination  144316 non-null  float32
```

```
 13  actual_time                   144316 non-null  float32
 14  osrm_time                     144316 non-null  float32
 15  osrm_distance                 144316 non-null  float32
 16  segment_actual_time           144316 non-null  float32
 17  segment_osrm_time             144316 non-null  float32
 18  segment_osrm_distance         144316 non-null  float32
dtypes: category(2), datetime64[ns](3), float32(8), object(6)
memory usage: 15.7+ MB
```

🏷️Insights:

- Only two fields have a tiny fraction of missing values, less than 0.05% of the whole dataset.
- Since we have plenty of data to work with, we're choosing to just get rid of the missing values instead of trying to guess them using methods like using the average or most common value.
- I'm dropping the missing values to keep things simple and not mess up how the features are spread out. But if a lot more data was missing, we could have used other methods like guessing based on what's there or using the most common values.

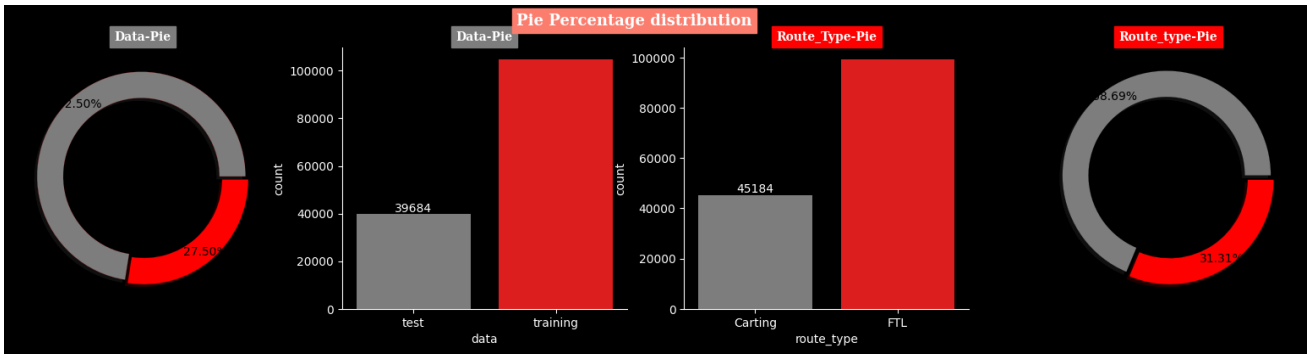
🕵️Exploratory Data Analysis

```
cp = ['gray','red','dimgrey','tomato','dimgray','orangered','k','salmon','gray','
```

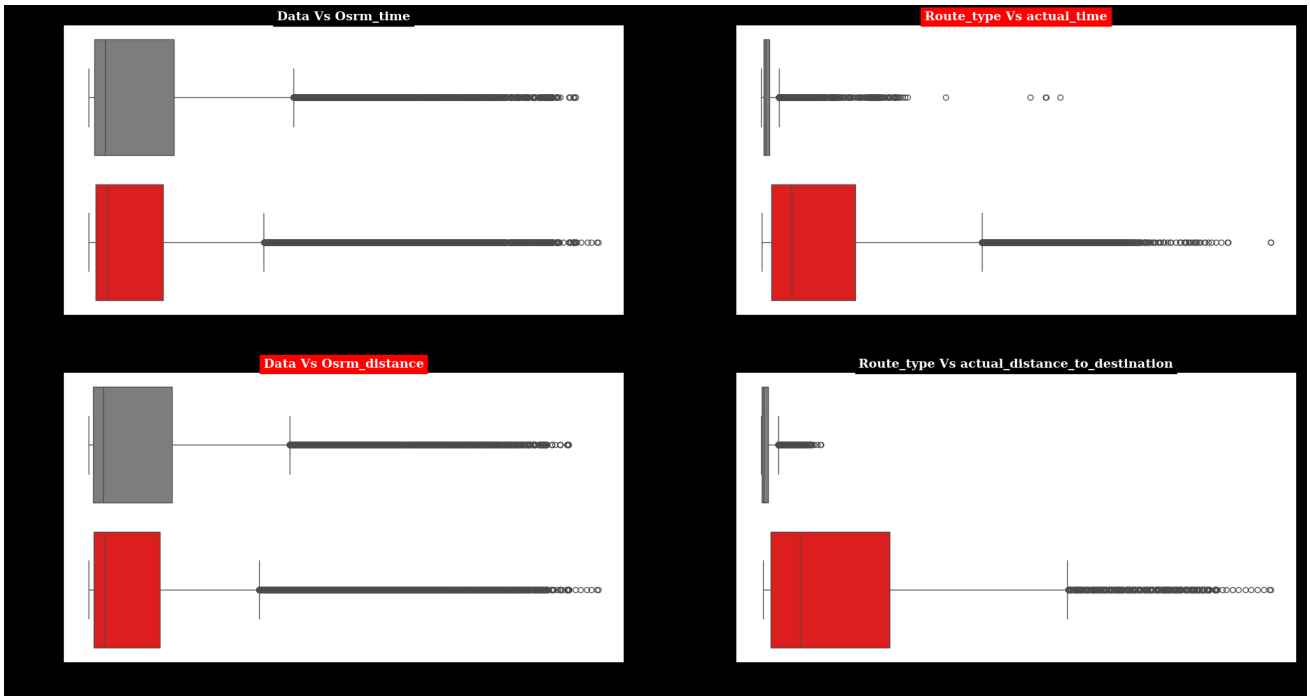
```
df.sample()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
9661	training	2018-09-14 05:14:51.329917	thanos::sroute:14058670-577e-4371-a06f-fde230f...	FTL	1536902091329

```
plt.figure(figsize=(20,4))
plt.suptitle('Pie Percentage distribution',fontsize=13,fontfamily='serif',fontwei

plt.subplot(141)
plt.pie(df['data'].value_counts(), labels=df['data'].value_counts().index, colors
       textprops={'color':'k','fontsize':10} , shadow=True, radius=1,wedgeprops=d
plt.title('Data-Pie',fontsize=10,fontfamily='serif',fontweight='bold',backgroundc

plt.subplot(142)
a = sns.barplot(x=df['data'].value_counts().index, y=df['data'].value_counts(), p
a.bar_label(a.containers[0], label_type='edge',fmt='%d')
plt.title('Data-Pie',fontsize=10,fontfamily='serif',fontweight='bold',backgroundc

plt.subplot(143)
b = sns.barplot(x=df['route_type'].value_counts().index, y=df['route_type'].value
b.bar_label(b.containers[0], label_type='edge',fmt='%d')
plt.title('Route_Type-Pie',fontsize=10,fontfamily='serif',fontweight='bold',backg

plt.subplot(144)
plt.pie(df['route_type'].value_counts(), labels=df['route_type'].value_counts().i
       textprops={'color':'k','fontsize':10} , shadow=True, radius=1,wedgeprops=d
plt.title('Route_type-Pie',fontsize=10,fontfamily='serif',fontweight='bold',backg

sns.despine()
plt.show()
```



```
    plt.figure(figsize=(25,13))
    plt.style.use('default')
    plt.style.use('seaborn-bright')

    plt.subplot(221)
    sns.boxplot(data=df,y='data',x='osrm_time',palette=cp)
    plt.title('Data Vs Osrm_time',fontsize=14,fontfamily='serif',fontweight='bold',ba

    plt.subplot(222)
    sns.boxplot(data=df,y='route_type',x='actual_time',palette=cp)
    plt.title('Route_type Vs actual_time',fontsize=14,fontfamily='serif',fontweight='

    plt.subplot(223)
    sns.boxplot(data=df,y='data',x='osrm_distance',palette=cp)
    plt.title('Data Vs Osrm_distance',fontsize=14,fontfamily='serif',fontweight='bold

    plt.subplot(224)
    sns.boxplot(data=df,y='route_type',x='actual_distance_to_destination',palette=cp)
    plt.title('Route_type Vs actual_distance_to_destination',fontsize=14,fontfamily='
    sns.despine()
    plt.show()
```


Data Vs Osrm_time
Route_type Vs actual_time
Data Vs Osrm_distance
Route_type Vs actual_distance_to_destination

Observations:

- Both training and test data have the same range of osrm time recorded
- FTL route type has more actual time compared to Carting. This can also be since FTL is used a lot more than carting in the data available to us
- Both training and test data have the same range of osrm distance recorded
- FTL route type has more actual distance compared to Carting. This can also be since FTL is used a lot more than carting in the data available to us

---

2.Merging of rows and aggregation of fields

Merging of rows and aggregation of fields

- Since delivery details of one package are divided into several rows (think of it as connecting flights to reach a particular destination). Now think about how we should treat their fields if we combine these rows? What aggregation would make sense if we merge. What would happen to the numeric fields if we merge the rows.

```
# Grouping by segment
# Creating a unique identifier for each segment of a trip

segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_tim

df['segment_key'] = df['trip_uuid'] + '+' + df['source_center'] + '+' + df['desti

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

df[['segment_key', 'segment_actual_time', 'segment_actual_time_sum','segment_osrm
```

	segment_key	segment_actual_time	segme
0	trip-153741093647649320+IND388121AAA+IND388620AAB	14.0	
1	trip-153741093647649320+IND388121AAA+IND388620AAB	10.0	
2	trip-153741093647649320+IND388121AAA+IND388620AAB	16.0	
3	trip-153741093647649320+IND388121AAA+IND388620AAB	21.0	
4	trip-153741093647649320+IND388121AAA+IND388620AAB	6.0	
...	...	...	
144862	trip-153746066843555182+IND131028AAB+IND000000ACB	12.0	
144863	trip-153746066843555182+IND131028AAB+IND000000ACB	26.0	

```
# Aggregating at segment level & Creating a dictionary for aggregation at segment

segment_dict = {
  'trip_uuid' : 'first',
  'data': 'first',
  'route_type': 'first',
  'trip_creation_time': 'first',
  'source_name': 'first',
  'destination_name': 'last',
  'od_start_time': 'first',
  'od_end_time': 'last',
  'start_scan_to_end_scan': 'first',
  'actual_distance_to_destination': 'last',
  'actual_time': 'last',
  'osrm_time': 'last',
  'osrm_distance': 'last',
  'segment_actual_time' : 'sum',
  'segment_osrm_time' : 'sum',
  'segment_osrm_distance' : 'sum',
  'segment_actual_time_sum': 'last',
  'segment_osrm_time_sum': 'last',
  'segment_osrm_distance_sum': 'last',
}

# Grouping by segment_key and aggregating
segment_agg_data = df.groupby('segment_key').agg(segment_dict).reset_index()
segment_agg_data = segment_agg_data.sort_values(by=['segment_key','od_end_time'])
segment_agg_data
```

	segment_key	trip_uuid	data	r
0	trip-153671041653548748+IND209304AAA+IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748+IND462022AAA+IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164+IND561203AAB+IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164+IND572101AAA+IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517+IND000000ACB+IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26217	trip-153861115439069069+IND628204AAA+IND627657AAA	trip-153861115439069069	test	
26218	trip-153861115439069069+IND628613AAA+IND627005AAA	trip-153861115439069069	test	
26219	trip-153861115439069069+IND628801AAA+IND628204AAA	trip-153861115439069069	test	
26220	trip-153861118270144424+IND583119AAA+IND583101AAA	trip-153861118270144424	test	
26221	trip-153861118270144424+IND583201AAA+IND583119AAA	trip-153861118270144424	test	

26222 rows × 20 columns

🔷 Understanding:

> The rows have been merged based on the unique segment_key, which is a combination of trip_uuid, source_center, and destination_center.

> The aggregated dataset reflects the total values for each segment of the trip.

📂 Feature Engineering

```
# 1. Calculating time difference between od_start_time and od_end_time
segment_agg_data['od_total_time']=(segment_agg_data['od_end_time'] - segment_agg_
segment_agg_data['od_time_diff_hour'] = (segment_agg_data['od_total_time']).dt.to
segment_agg_data
```

	segment_key	trip_uuid	data	r
0	trip-153671041653548748+IND209304AAA+IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748+IND462022AAA+IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164+IND561203AAB+IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164+IND572101AAA+IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517+IND000000ACB+IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26217	trip-153861115439069069+IND628204AAA+IND627657AAA	trip-153861115439069069	test	
26218	trip-153861115439069069+IND628613AAA+IND627005AAA	trip-153861115439069069	test	
26219	trip-153861115439069069+IND628801AAA+IND628204AAA	trip-153861115439069069	test	
26220	trip-153861118270144424+IND583119AAA+IND583101AAA	trip-153861118270144424	test	
26221	trip-153861118270144424+IND583201AAA+IND583119AAA	trip-153861118270144424	test	

26222 rows × 22 columns

```
segment_agg_data.sample()
```

	segment_key	trip_uuid	data	ro
8717	trip-153730784169518637+IND521105AAA+IND534001AAA	trip-153730784169518637	training	

```
# de = segment_agg_data.drop(columns=['source_city','source_state','source_place'
de = segment_agg_data.copy()
```

```
de.sample()
```

	segment_key	trip_uuid	data	rou
19756	trip-153808171245619718+IND421302AAG+IND401104AAA	trip-153808171245619718	test	

```
# # could have done this --- but some major error ... check it....
# sad = segment_agg_data.copy()
# sad["source_city"] = sad["source_name"].str.split(" ",n=1,expand=True)[0].str.s
# sad["source_state"] = sad["source_name"].str.split(" ",n=1,expand=True)[1].str.

# sad["destination_city"] = sad["destination_name"].str.split(" ",n=1,expand=True
# sad["destination_state"] = sad["destination_name"].str.split(" ",n=1,expand=Tru

# sad["source_place"] = sad["source_name"].str.split("_",n=2,expand=True)[1]
# sad["destination_place"] = sad["destination_name"].str.split("_",n=2,expand=Tru


# using regex pattern to seperate the city,place,state
def extract_info(name):
    pattern = r'^(?P<city>[^\s_]+)_?(?P<place>[^\(\)]*)\s?\((?P<state>[A-Za-z\s&]
    match = re.match(pattern, name)
    if match:
        city = match.group('city').strip()
        place = match.group('place').strip() if match.group('place') else city
        state = match.group('state').strip()
        return city, place, state
    else:
        return None, None, None


de[['source_city', 'source_place', 'source_state']] = de['source_name'].apply(lam


de[['destination_city', 'destination_place', 'destination_state']] = de['destinat


de
```

	segment_key	trip_uuid	data	r
0	trip-153671041653548748+IND209304AAA+IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748+IND462022AAA+IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164+IND561203AAB+IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164+IND572101AAA+IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517+IND000000ACB+IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26217	trip-153861115439069069+IND628204AAA+IND627657AAA	trip-153861115439069069	test	
26218	trip-153861115439069069+IND628613AAA+IND627005AAA	trip-153861115439069069	test	
26219	trip-153861115439069069+IND628801AAA+IND628204AAA	trip-153861115439069069	test	
26220	trip-153861118270144424+IND583119AAA+IND583101AAA	trip-153861118270144424	test	
26221	trip-153861118270144424+IND583201AAA+IND583119AAA	trip-153861118270144424	test	

26222 rows × 28 columns

```
de[(de['source_place']=='') | (de['destination_place']=='')]
```

	segment_key	trip_uuid	data	r
7	trip-153671052974046625+IND583101AAA+IND583201AAA	trip-153671052974046625	training	
9	trip-153671052974046625+IND583201AAA+IND583119AAA	trip-153671052974046625	training	
19	trip-153671110078355292+IND121004AAB+IND121001AAA	trip-153671110078355292	training	
33	trip-153671173668736946+IND110043AAA+IND110078AAA	trip-153671173668736946	training	
80	trip-153671320807895983+IND121004AAB+IND121102AAA	trip-153671320807895983	training	
...	...	...	...	
26118	trip-153860849934816308+IND110078AAA+IND110043AAA	trip-153860849934816308	test	
26153	trip-153860958923357924+IND842003AAB+IND482002AAA	trip-153860958923357924	test	
26180	trip-153861007249500192+IND842001AAA+IND846004AAA	trip-153861007249500192	test	
26181	trip-153861007249500192+IND846004AAA+IND847103AAA	trip-153861007249500192	test	
26221	trip-153861118270144424+IND583201AAA+IND583119AAA	trip-153861118270144424	test	

782 rows × 28 columns

```
de.loc[de['source_place']=='','source_place']=de['source_city']
de.loc[de['destination_place']=='','destination_place']=de['destination_city']
```

```
de[de.source_place.isna()]
```

	segment_key	trip_uuid	data	route_type	trip_creation_time	source_name	de

```
de.isna().sum()
```

```
segment_key                0
trip_uuid                  0
data                       0
route_type                 0
trip_creation_time         0
source_name                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
segment_actual_time_sum           0
segment_osrm_time_sum             0
segment_osrm_distance_sum         0
od_total_time                     0
od_time_diff_hour                 0
source_city                       0
source_place                      0
source_state                      0
destination_city                  0
destination_place                 0
destination_state                 0
dtype: int64
```

```
#de = de.drop(columns=['source_city','source_state','source_place','destination_c
```

```
de.loc[de.source_city=='Bangalore','source_city']='Bengaluru'
de.loc[de.destination_city=='Bangalore','destination_city']='Bengaluru'
```

```
np.set_printoptions(threshold=np.inf)
```

```
de['source_city'].unique()
```

```
array(['Kanpur', 'Bhopal', 'Doddablpur', 'Tumkur', 'Gurgaon',
'Bengaluru',
       'Mumbai', 'Bellary', 'Sandur', 'Hospet', 'Chennai', 'HBR',
'Surat',
       'Delhi', 'Pune', 'FBD', 'Shirala', 'Ratnagiri', 'Kolhapur',
       'Hyderabad', 'Anantapur', 'Thirumalagiri', 'Gulbarga', 'Aland',
       'Sindagi', 'Indi', 'Jaipur', 'Allahabad', 'Guwahati', 'Unnao',
       'Narsinghpur', 'Gadarwara', 'Shrirampur', 'Nashik', 'Sinnar',
       'Sangamner', 'Shirdi', 'Kopargaon', 'Vaijiapur', 'Hoogly',
       'Hooghly', 'Kolkata', 'Madakasira', 'Pavagada', 'Sonari',
       'Medchal', 'Dindigul', 'Kodaikanal', 'Batlagundu', 'Palani',
       'Oddnchtram', 'Jalandhar', 'Nakodar', 'Kapurthala', 'Faridabad',
       'Chandigarh', 'Deoli', 'Pandharpur', 'Atapadi', 'CCU', 'Bhandara',
       'Kurnool', 'Palwal', 'Bhiwandi', 'Bhatinda', 'TalwandiSabo',
       'Mansa', 'Jhunir', 'RoopNagar', 'AnandprShb', 'Bantwal', 'Kadaba',
       'Sullia', 'Chittapur', 'Sedam', 'Chincholi', 'Lalru', 'Kadi',
       'Mehsana', 'Shahdol', 'Dola', 'Gangakher', 'Parli', 'Ambajogai',
       'Nanded', 'Loha', 'Durgapur', 'Bankura', 'Barjora', 'Vapi',
       'Jamjodhpur', 'Porbandar', 'Junagadh', 'Jetpur', 'Dhoraji',
       'Khammam', 'Nalgonda', 'Miryalguda', 'Suryapet', 'Choutuppal',
       'Vijayawada', 'Vadnagar', 'Palanpur', 'Deesa', 'Jabalpur',
       'Talala', 'Veraval', 'Una', 'Kodinar', 'Gundlupet',
'Tirumakudalu',
       'Chamarjngr', 'Malavalli', 'Kollegala', 'Mysore', 'Hunsur',
       'HDKote', 'Gonikoppal', 'Patan', 'Bhabhar', 'Vizag', 'Rajamundry',
       'Goa', 'Sawantwadi', 'Kankavali', 'Sonipat', 'Moradabad',
       'Rudrapur', 'Himmatnagar', 'Khedbrahma', 'Modasa', 'Jamshedpur',
```

```
       'Pondicherry', 'Cuddalore', 'Chidambaram', 'Sirkazhi', 'Karaikal',
       'Thiruvarur', 'Nagapttinm', 'MAA', 'Anand', 'Khambhat', 'Udgir',
       'Latur', 'Degloor', 'Nadiad', 'Umreth', 'Villupuram',
'Virudhchlm',
       'Pennadam', 'Chinnasalem', 'Panruti', 'Neyveli', 'Purulia',
'Hura',
       'Rghunthpur', 'Jhalda', 'Bhubaneshwar', 'Bamangola', 'Meham',
       'Tiruppattur', 'Ambur', 'Kotdwara', 'Haridwar', 'Medak',
       'Narsapur', 'Kamareddy', 'Yellareddy', 'Bodhan', 'Banswada',
       'Dhrangadhra', 'Halvad', 'Gandhidham', 'Gangavathi', 'Koppal',
       'Ghumarwin', 'JognderNgr', 'Jahu', 'ChandroknaRD', 'Kharagpur',
       'AmaDubi', 'Agra', 'Sitapur', 'Biswan', 'Lakhimpur', 'Gola',
       'Canacona', 'Bilimora', 'SultnBthry', 'Lucknow', 'Vellore',
'Bhuj',
       'Anjar', 'Dinhata', 'BOM', 'Margherita', 'Boisar', 'Dahanu',
       'Chodavaram', 'Tezpur', 'Bomdila', 'Koduru', 'Rajampet',
       'Tirupati', 'Puttur', 'Sriklahsti', 'Gudur', 'Venktagiri', 'Pen',
       'Roha', 'Mahad', 'AMD', 'Ahmedabad', 'Faizabad', 'Gosainganj',
       'Gandhinagar', 'Muzaffrpur', 'Phagwara', 'Betul', 'Itarsi',
       'Pandhurna', 'Panskura', 'Haldia', 'Tamluk', 'Karur', 'BLR',
       'Rasipurm', 'Sankari', 'Jorhat', 'Bokakhat', 'PNQ', 'Aligarh',
       'Mainpuri', 'Shikohabad', 'Firozabad', 'Srikakulam', 'Rajam',
       'Palakonda', 'Parvathipuram', 'Bobbili', 'Tekkali', 'Palasa',
       'Narasnpeta', 'Paralakhemundi', 'Dehradun', 'Hajo', 'NOI',
       'Jassur', 'Dalhousie', 'Chamba', 'Baharampur', 'Dhulian',
       'Hoskote', 'Shajapur', 'Shujalpur', 'Pachore', 'OK', 'Ludhiana',
       'GreaterThane', 'Janakpuri', 'Amritsar', 'Tirupur', 'Attur',
       'Salem', 'Tirchngode', 'Darjeeling', 'Mirik', 'Tiruchi', 'Noida',
       'Rangia', 'Dhubri', 'Bilasipara', 'Kokrajhar', 'Tura', 'Ajmer',
       'Pali', 'Jodhpur', 'Amroha', 'Gajraula', 'Rampur', 'Jhansi',
       'Dholpur', 'Gwalior', 'Thirthurpondi', 'Pushpavanam', 'Ranchi',
```

```
de['source_place'].unique()
```

```
array(['Central_H_6', 'Trnsport_H', 'ChikaDPP_D', 'Veersagr_I',
       'Bilaspur_HB', 'Nelmngla_H', 'Hub', 'Dc', 'WrdN1DPP_D', 'Hospet',
       'Poonamallee', 'Porur_DPC', 'Chrompet_DPC', 'Layout PC',
       'Bagaluru_D', 'Central_D_12', 'Central_I_4', 'Lajpat_IP',
       'North_D_3', 'Balabhgarh_DPC', 'Central_DPP_3', 'MjgaonRd_D',
       'Shivaji_I', 'Shamshbd_H', 'KamaStrt_I', 'Xroad_D', 'Nehrugnj_I',
       'RazaviRd_D', 'KalyanNg_D', 'SindgiRD_D', 'Central_I_7',
       'VarunCly_DC', 'Central_H_1', 'Nangli_IP', 'North', 'VikasRam_D',
       'KndliDPP_D', 'MPward_D', 'Central_D_9', 'DavkharRd_D',
       'TgrniaRD_I', 'Central_D_1', 'JawanChk_D', 'SaiBansi_D',
       'NkshtrPz_D', 'YeolaRD_D', 'Bandel_D', 'DC', 'North_I_4',
       'RTCStand_D', 'PnukndRD_D', 'Central_DPP_1', 'KGAirprt_HB',
       'North_D_2', 'MROoffce_D', 'Athithnr_DC', 'RTOofice_D',
       'RjndraRd_D', 'Palani_D', 'DPC', 'ChowkDPP_D', 'Mthurard_L',
       'Mullanpr_DC', 'Mehmdpur_H', 'Mohali', 'Central_DPP_2',
       'RajCmplx_D', 'Vidyangr_D', 'Beliaghata_DPC', 'RjnaiDPP_D',
       'KaremDPP_D', 'AbbasNgr_I', 'Palwal', 'Mankoli_HB', 'Wardno3_D',
       'GreenVly_D', 'BhaRDDPP_D', 'Airport_H', 'Gateway_HB',
       'Tathawde_H', 'ChotiHvl_DC', 'PnjPiara_D', 'Kharar_DC',
       'Trmltmpl_D', 'AnugrDPP_D', 'Srirampt_D', 'GlbrgaRD_D',
       'DBRCmplx_D', 'Mainroad_D', 'OnkarDPP_D', 'KaranNGR_D',
       'Panchot_IP', 'Sohagpur_D', 'ChainDPP_D', 'Chrompet_L',
       'Busstand_D', 'JirgeNgr_D', 'BnsllNgr_D', 'Aswningr_I',
       'SivjiCWK_D', 'Central_I_1', 'KeranDPP_D', 'IndEstat_I',
'Court_D',
       'JmnvadRd_DC', 'NSTRoad_I', 'HydRoad_DC', 'Ragvendr_D',
```

```
       'Krishna_D', 'Nagaram_D', 'Rynapadu_H', 'BsstdDPP_D',
       'HawaiPlr_DC', 'Adhartal_IP', 'DumDum_DPC', 'Bomsndra_HB',
       'SsnRdDPP_D', 'Mamlatdr_DC', 'NCplxDPP_D', 'Swamylyt_D',
       'Narasipr_D', 'BrDbleRd_D', 'SulthnRd_D', 'NarsipuraRd_D',
       'Yadvgiri_IP', 'RatnaDPP_D', 'BegurRD_D', 'Thomas_D',
       'Central_D_2', 'TirupDPP_D', 'Gajuwaka_L', 'AtoNgrRd_I',
       'LaxmiNgr_D', 'NrdawDPP_D', 'Old City', 'Kundli_H', 'UdhamNgr_H',
       'Patelfli_D', 'Central_I_3', 'Vasanthm_I', 'KtsiGrsm_D',
       'ARBNorth_DC', 'Pngktgudi_D', 'Thalthru_DC', 'Bypasrd_D',
       'Sttyapar_D', 'Poonamallee_HB', 'VUNagar_DC', 'MotvdDPP_D',
       'NlgaonRd_D', 'Srnwsngr_D', 'Dakor_DC', 'Vaghasi_IP',
'Bnnrghta_L',
       'Thirumtr_IP', 'SelamRd_D', 'EastmnRD_D', 'VkotRoad_D',
       'GandhiRd_D', 'NJVNgr_D', 'GariDPP_D', 'barkarRd_D', 'GMukrDPP_D',
       'MP Nagar', 'Central_D_3', 'Jogshwri_I', 'GModDPP_D',
'KoilStrt_D',
       'CotnGren_M', 'Nzbadrd_D', 'Haridwar', 'Dwaraka_D', 'Devenply_I',
       'JKRoad_D', 'SuryaDPP_D', 'Menagrdn_D', 'NvygRDPP_D', 'CrossRD_D',
       'Sector1A_IP', 'PhrmPlza_D', 'Banikatt_D', 'Gndhichk_D',
'Dhelu_D',
       'Sulgwan_D', 'Bulabeda_D', 'Chowk_D', 'PatelNGR_D', 'Maheva_D',
       'BkgnRoad_D', 'CharRsta_D', 'Kollgpra_D', 'Peenya_IP',
       'GndhiNgr_IP', 'Sanpada_I', 'WrdN4DPP_D', 'Sakinaka_RP',
       'CivilHPL_D', 'OstwlEmp_D', 'KetyDPP_D', 'Gajuwaka', 'BaljiDPP_D',
       'Mhbhirab_D', 'CTRoad_D', 'MGRoad_D', 'RSRoad_D', 'Balajicly_I',
       'Artmclny_D', 'SDKNgr_D', 'Bngisheb_D', 'TirupthiRd_D',
       'BljiMrkt_D', 'DataSagr_D', 'Govndsgr_D', 'Dankuni_HB', 'Rakhial',
       'East_H_1', 'Memnagar', 'East_I_21', 'Mithakal_D', 'TrnspNgr_D',
       'Pakrela_D', 'Bbganj_I', 'Bilaspur_RP', 'Lovely_D', 'PatelWrd_D',
       'DivrsnRd_D', 'Mataward_D', 'CottonGreen_DPC', 'Pawane_L',
'Karur',
       'JPNagar Pc', 'Knrpati D', 'Trchngrd D', 'Kengeri IP', 'KHRoad I',
```

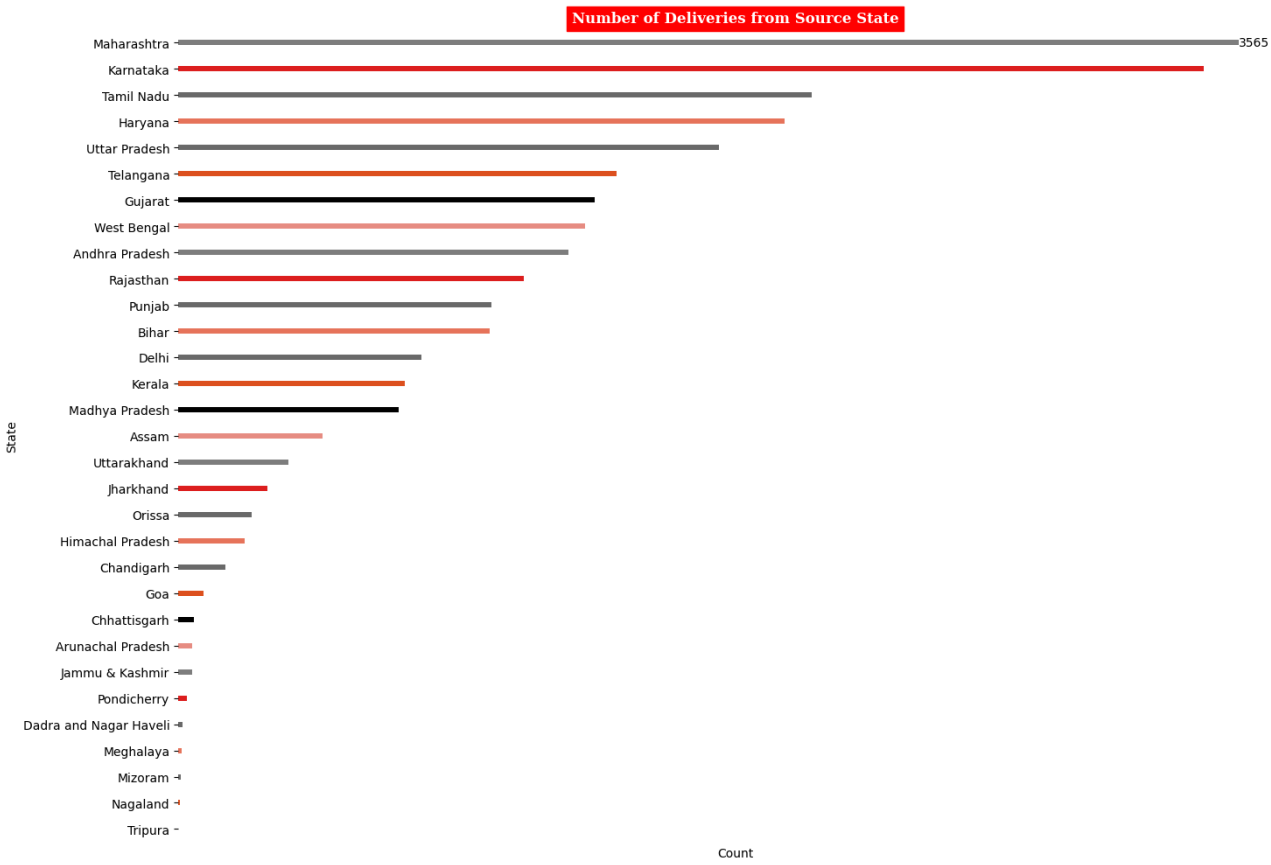
```
de['source_state'].unique()
```

```
array(['Uttar Pradesh', 'Madhya Pradesh', 'Karnataka', 'Haryana',
       'Maharashtra', 'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana',
       'Andhra Pradesh', 'Rajasthan', 'Assam', 'West Bengal', 'Punjab',
       'Chandigarh', 'Goa', 'Uttarakhand', 'Jharkhand', 'Pondicherry',
       'Orissa', 'Himachal Pradesh', 'Kerala', 'Arunachal Pradesh',
       'Bihar', 'Meghalaya', 'Chhattisgarh', 'Jammu & Kashmir',
       'Dadra and Nagar Haveli', 'Mizoram', 'Tripura', 'Nagaland'],
      dtype=object)
```

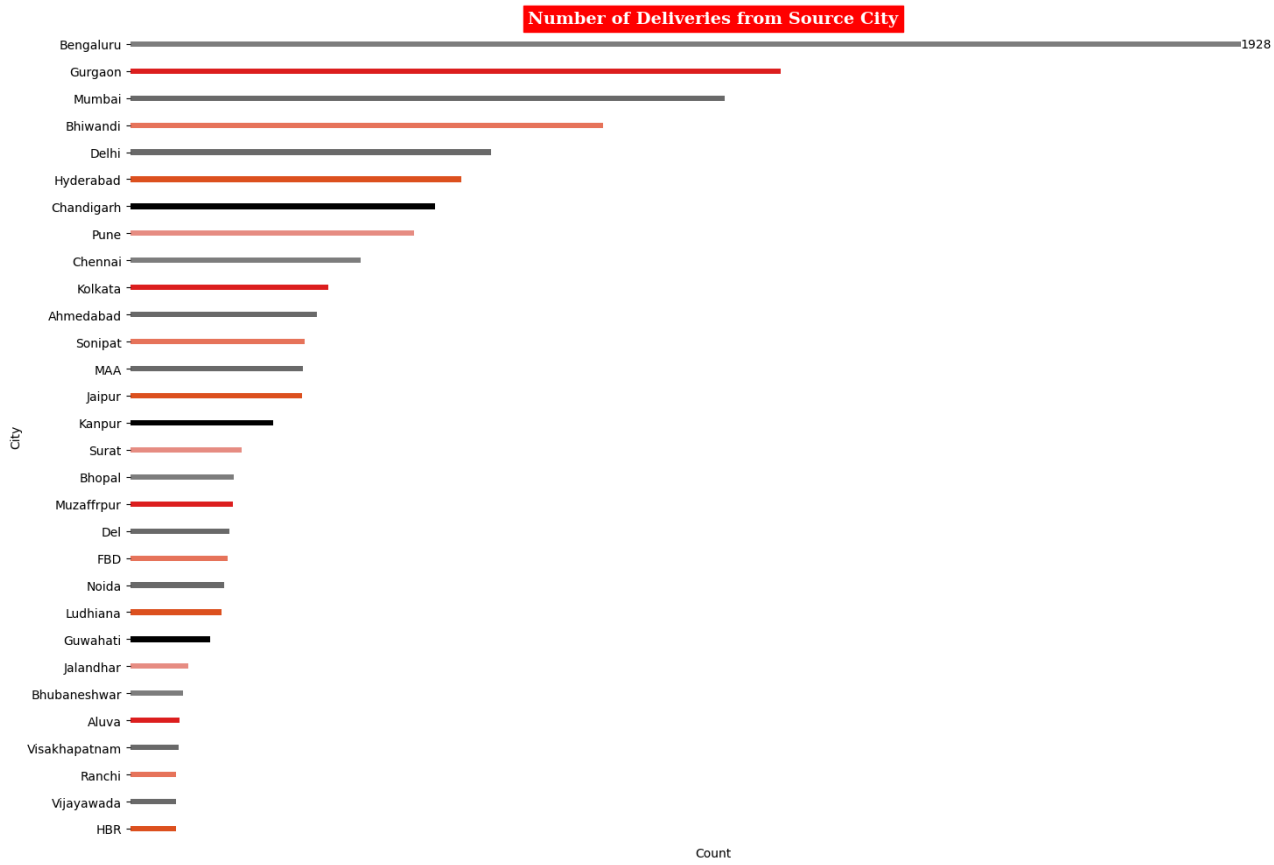
```
de['source_state'].value_counts().to_frame().style.background_gradient(cmap='Reds
```

source_state	count
Maharashtra	3565
Karnataka	3445
Tamil Nadu	2130
Haryana	2039
Uttar Pradesh	1818
Telangana	1474
Gujarat	1401
West Bengal	1368
Andhra Pradesh	1310
Rajasthan	1162
Punjab	1052
Bihar	1048
Delhi	818
Kerala	762
Madhya Pradesh	740
Assam	484
Uttarakhand	369
Jharkhand	301
Orissa	248
Himachal Pradesh	223
Chandigarh	160
Goa	86
Chhattisgarh	52
Arunachal Pradesh	48
Jammu & Kashmir	47
Pondicherry	30
Dadra and Nagar Haveli	15
Meghalaya	13
Mizoram	8
Nagaland	5

```
state_counts = de['source_state'].value_counts().to_frame().reset_index()
state_counts.columns = ['State', 'Count']

plt.figure(figsize=(15,10))
a = sns.barplot(y='State', x='Count', data=state_counts,palette=cp,width=0.2)
a.bar_label(a.containers[0], label_type='edge')
plt.xticks([])
plt.ylabel('State')
plt.xlabel('Count')
plt.title('Number of Deliveries from Source State',fontsize=12,fontfamily='serif'
plt.tight_layout()
sns.despine(bottom=True,left=True)
plt.show()
```


Number of Deliveries from Source State
Maharashtra
Karnataka
Tamil Nadu
Haryana
Uttar Pradesh
Telangana
Gujarat
West Bengal
Andhra Pradesh
Rajasthan
Punjab
Bihar
Delhi
Kerala
Madhya Pradesh
Assam
Uttarakhand
Jharkhand
Orissa
Himachal Pradesh
Chandigarh
Goa
Chhattisgarh
Arunachal Pradesh
Jammu & Kashmir
Pondicherry
Dadra and Nagar Haveli
Meghalaya
Mizoram
Nagaland
Tripura
3565
State
Count

```
city_counts = de['source_city'].value_counts().to_frame().reset_index()[:30]
city_counts.columns = ['City', 'Count']

plt.figure(figsize=(15,10))
a = sns.barplot(y='City', x='Count', data=city_counts,palette=cp,width=0.2)
a.bar_label(a.containers[0], label_type='edge')
plt.xticks([])
plt.ylabel('City')
plt.xlabel('Count')
plt.title('Number of Deliveries from Source City',fontsize=14,fontfamily='serif',
plt.tight_layout()
sns.despine(bottom=True,left=True)
plt.show()
```


Number of Deliveries from Source City
Bengaluru
1928
Gurgaon
Mumbai
Bhiwandi
Delhi
Hyderabad
Chandigarh
Pune
Chennai
Kolkata
Ahmedabad
Sonipat
MAA
Jaipur
Kanpur
Surat
Bhopal
Muzaffrpur
Del
FBD
Noida
Ludhiana
Guwahati
Jalandhar
Bhubaneshwar
Aluva
Visakhapatnam
Ranchi
Vijayawada
HBR
City
Count

Insights:

> Source State contributors

- **Maharashtra**, **Karnataka** ,**Tamil Nadu**, **Haryana**, and **Uttar Pradesh** are the top contributors where maximum bookings are recorded in this month indicating significant engagement.

> Source City contributors

- Cities like **Bengaluru**, **Gurgaon**, **Mumbai**, **Bhiwandi**, **Delhi**, **Hyderabad** where the major no.of booking are recorded.

```
de.describe().T
```

	count	mean	min	
trip_creation_time	26222	2018-09-22 13:58:56.740969728	2018-09-12 00:00:16.535741	2018-(03:57:50.90041
od_start_time	26222	2018-09-22 17:49:54.012840448	2018-09-12 00:00:16.535741	2018-(07:43:36.52578
od_end_time	26222	2018-09-22 22:49:00.449498112	2018-09-12 00:50:10.814399	2018-(15:10:35.61536
start_scan_to_end_scan	26222.0	298.553375	20.0	
actual_distance_to_destination	26222.0	92.533051	9.001351	21.6
actual_time	26222.0	200.92659	9.0	
osrm_time	26222.0	90.785332	6.0	
osrm_distance	26222.0	114.975334	9.0729	27.7
segment_actual_time	26222.0	199.095642	9.0	
segment_osrm_time	26222.0	101.793343	6.0	
segment_osrm_distance	26222.0	125.587128	9.0729	28.42
segment_actual_time_sum	26222.0	199.095642	9.0	
segment_osrm_time_sum	26222.0	101.793343	6.0	
segment_osrm_distance_sum	26222.0	125.587128	9.0729	28.42

```
de.describe(include='object').T
```

	count	unique	top	fre
segment_key	26222	26222	trip-153671041653548748+IND209304AAA+IND000000ACB	
trip_uuid	26222	14787	trip-153717306559016761	
source_name	26222	1496	Gurgaon_Bilaspur_HB (Haryana)	105
destination_name	26222	1466	Gurgaon_Bilaspur_HB (Haryana)	92
source_city	26222	1239	Bengaluru	192
source_place	26222	1246	Bilaspur_HB	105
source_state	26222	31	Maharashtra	356
destination_city	26222	1236	Bengaluru	186
destination_place	26222	1217	Bilaspur_HB	92

```
de['destination_city'].unique()
```

```
array(['Gurgaon', 'Kanpur', 'Chikblapur', 'Doddablpur', 'Chandigarh',
       'Mumbai', 'Hospet', 'Bellary', 'Sandur', 'Chennai', 'Bengaluru',
       'HBR', 'Surat', 'Delhi', 'PNQ', 'Faridabad', 'Kolhapur',
'Shirala',
       'Ratnagiri', 'Anantapur', 'Hyderabad', 'Sindagi', 'Gulbarga',
       'Indi', 'Aland', 'Jaipur', 'Satna', 'Janakpuri', 'Guwahati',
       'Unnao', 'Gadarwara', 'Bareli', 'Shirdi', 'Sinnar', 'Sangamner',
       'Shrirampur', 'Kopargaon', 'Vaijiapur', 'Nashik', 'Kolkata',
       'Hoogly', 'Hooghly', 'Pavagada', 'Puttaprthi', 'Sivasagar',
       'Medchal', 'Oddnchtram', 'Batlagundu', 'Vadipatti', 'Kodaikanal',
       'Palani', 'Nakodar', 'Kapurthala', 'Jalandhar', 'Yavatmal',
       'Atapadi', 'Sangola', 'Bhandara', 'Savner', 'Kurnool', 'Palwal',
       'FBD', 'TalwandiSabo', 'Mansa', 'Jhunir', 'Bhatinda', 'Bhiwandi',
       'Barnala', 'Murbad', 'AnandprShb', 'RoopNagar', 'Puttur',
'Kadaba',
       'Chittapur', 'Sedam', 'Chincholi', 'Naraingarh', 'Ludhiana',
       'Ahmedabad', 'Kadi', 'Dola', 'Jabalpur', 'MAA', 'Parli',
       'Ambajogai', 'Pune', 'Loha', 'Gangakher', 'Barjora', 'Bishnupur',
       'Bankura', 'Silvassa', 'Junagadh', 'Bhanvad', 'Porbandar',
       'Dhoraji', 'Upleta', 'Jetpur', 'Choutuppal', 'Suryapet',
       'Vijayawada', 'Nalgonda', 'Miryalguda', 'Khammam', 'Vadnagar',
       'Palanpur', 'Deesa', 'Mehsana', 'Katni', 'Kodinar', 'Talala',
       'Una', 'Chamarjngr', 'Mysore', 'Kollegala', 'Tirumakudalu',
       'Malavalli', 'Krishnarajngr', 'Gonikoppal', 'HDKote', 'Unjha',
       'Bhabhar', 'Radhanpur', 'Rajamundry', 'Visakhapatnam',
       'Sawantwadi', 'Kankavali', 'Bhopal', 'Bhubaneshwar', 'Allahabad',
       'Moradabad', 'Rudrapur', 'Sonipat', 'Modasa', 'Khedbrahma',
       'Himmatnagar', 'Sasaram', 'Ranchi', 'Cuddalore', 'Chidambaram',
       'Sirkazhi', 'Karaikal', 'Nagapttinm', 'Pondicherry', 'Thiruvarur',
       'GZB', 'Khambhat', 'Anand', 'Degloor', 'Udgir', 'Latur', 'Nanded',
       'Noida', 'Umreth', 'Nadiad', 'Panruti', 'Pennadam', 'Chinnasalem',
       'Villupuram', 'Neyveli', 'Virudhchlm', 'Jhalda', 'Purulia',
'Hura',
       'Durgapur', 'Bhadrak', 'Goa', 'Balurghat', 'Meham', 'Hisar',
       'Ambur', 'Tiruppattur', 'Haridwar', 'Kotdwara', 'Narsapur',
       'Kamareddy', 'Bodhan', 'Medak', 'Banswada', 'Yellareddy',
```

```
       'Dhrangadhra', 'Halvad', 'Gangavathi', 'Koppal', 'Jahu',
       'BilaspurHP', 'JognderNgr', 'Kharagpur', 'Jhargram',
       'ChandroknaRD', 'Kirauli', 'BLR', 'Lakhimpur', 'Sitapur', 'Gola',
       'Dhaurahara', 'Mangalore', 'Canacona', 'Vansda', 'Mananthavady',
       'Lucknow', 'Silchar', 'Nakhatrana', 'Bhuj', 'Pundibari',
       'LowerParel', 'Changlang', 'Dahanu', 'Boisar', 'Chodavaram',
       'Bhalukpong', 'Tezpur', 'Rajampet', 'Tirupati', 'Koduru', 'GGN',
       'Sriklahsti', 'Venktagiri', 'Gudur', 'Roha', 'Mahad', 'Pen',
'CCU',
       'Amdavad', 'AMD', 'Gosainganj', 'Akbarpur', 'Muzaffrpur',
'Purnia',
       'Aurangabad', 'KN', 'Phagwara', 'Pandhurna', 'Betul', 'Sausar',
       'Tamluk', 'Panskura', 'Haldia', 'Madurai', 'Dindigul', 'Namakkal',
       'Erode', 'Aligarh', 'Mainpuri', 'Shikohabad', 'Firozabad',
'Rajam',
       'Salur', 'Srikakulam', 'Palakonda', 'Parvathipuram', 'Narasnpeta',
       'Palasa', 'Paralakhemundi', 'Tekkali', 'Nalasopara', 'Hajo',
'NOI',
       'Dalhousie', 'Chamba', 'Pathankot', 'Baharampur', 'Dhulian',
       'Malda', 'Malvan', 'Hoskote', 'Shujalpur', 'Shajapur', 'Ambabadi',
       'OK', 'Amritsar', 'Coimbatore', 'Jasai', 'Tirchngode', 'Mettur',
       'Kurseong', 'Darjeeling', 'Tiruchi', 'Dadri', 'Del', 'Rangia',
```

```
de['destination_place'].unique()
```

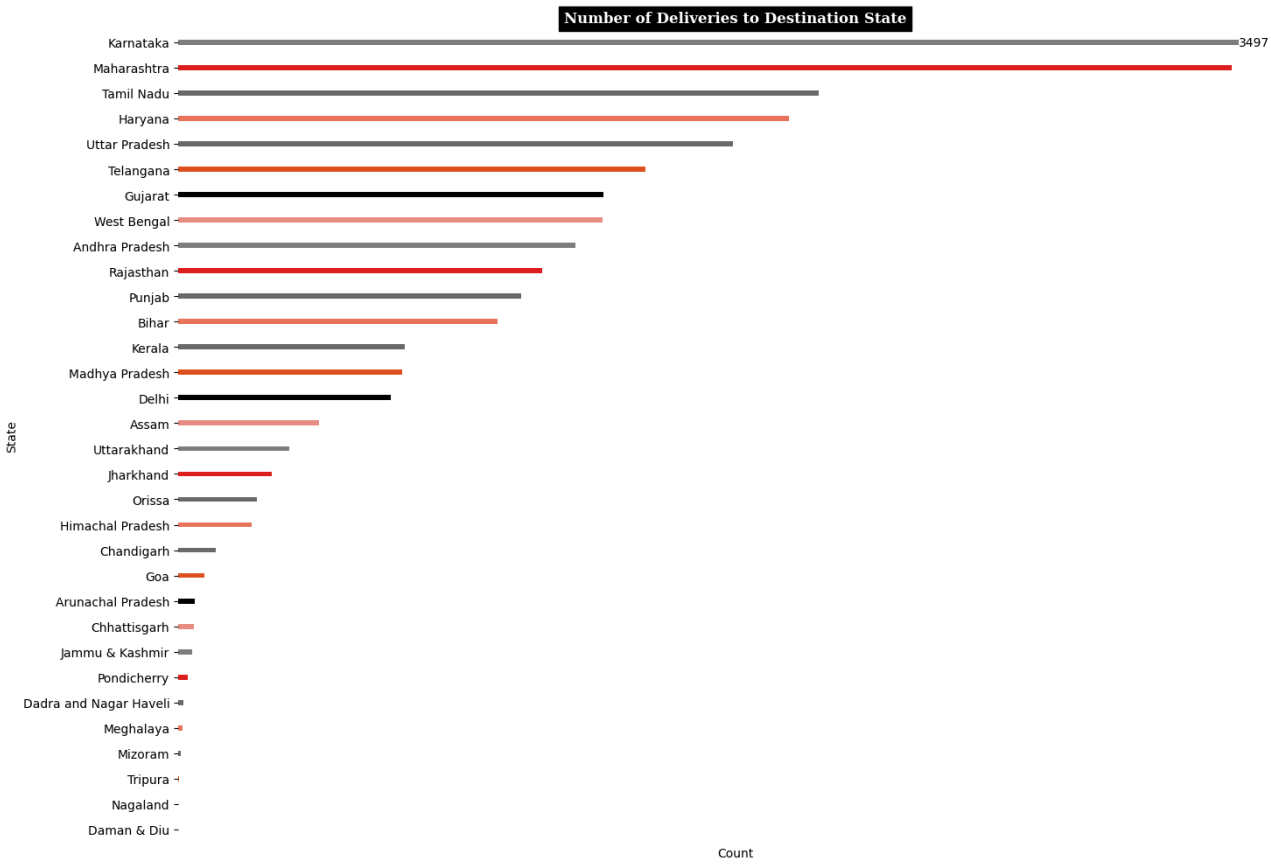
```
array(['Bilaspur_HB', 'Central_H_6', 'ShntiSgr_D', 'ChikaDPP_D',
       'Mehmdpur_H', 'MiraRd_IP', 'Hospet', 'Dc', 'WrdN1DPP_D',
       'Sriperumbudur_Dc', 'Poonamallee', 'Vandalur_Dc', 'NwYlhnka_DC',
       'Layout PC', 'Central_I_4', 'Central_D_3', 'Bhogal',
       'Rahatani DPC', 'Faridabad', 'Shivaji_I', 'Central_DPP_3',
       'MjgaonRd_D', 'KamaStrt_I', 'Nelmngla_H', 'Uppal_I', 'KalyanNg_D',
       'Nehrugnj_I', 'SindgiRD_D', 'RazaviRd_D', 'Bhankrot_DC',
       'Central_I_7', 'Central_I_2', 'Janakpuri', 'Hub', 'VikasRam_D',
       'MPward_D', 'SourvDPP_D', 'Varachha_DC', 'SaiBansi_D',
       'Central_D_1', 'JawanChk_D', 'DavkharRd_D', 'NkshtrPz_D',
       'YeolaRD_D', 'TgrniaRD_I', 'North_I_4', 'Bandel_D', 'DC',
       'PnukndRD_D', 'Gokulam_D', 'Babupaty_D', 'Bomsndra_HB',
       'MROoffce_D', 'Alwal_I', 'Palani_D', 'RTOofice_D', 'lalaNGR_D',
       'Athithnr_DC', 'RjndraRd_D', 'ChowkDPP_D', 'DPC', 'Mohali',
       'Mullanpr_DC', 'Sanpada_I', 'JajuDPP_D', 'Vidyangr_D',
       'Central_DPP_2', 'Dankuni_HB', 'KaremDPP_D', 'Wagodha_D',
       'Shamshbd_H', 'AbbasNgr_I', 'Palwal', 'Balabhgarh_DPC',
       'Wardno3_D', 'GreenVly_D', 'BhaRDDPP_D', 'Mankoli_HB',
       'SnkunDPP_D', 'PnjPiara_D', 'ChotiHvl_DC', 'Kharar_DC',
'Darbe_DC',
       'AnugrDPP_D', 'GlbrgaRD_D', 'DBRCmplx_D', 'Mainroad_D',
       'Ward2DPP_D', 'MilrGanj_HB', 'East_H_1', 'KaranNGR_D',
       'ChainDPP_D', 'Adhartal_IP', 'Poonamallee_HB', 'JirgeNgr_D',
       'BnsllNgr_D', 'Tathawde_H', 'SivjiCWK_D', 'Busstand_D',
       'Central_DPP_1', 'StnRdDPP_D', 'BhowmDPP_D', 'Samrvrni_D',
       'JmnvadRd_DC', 'AdrshSt_DC', 'Nagaram_D', 'Krishna_D',
       'Rynapadu_H', 'HydRoad_DC', 'Ragvendr_D', 'NSTRoad_I',
       'BsstdDPP_D', 'HawaiPlr_DC', 'Panchot_IP', 'Bargawan_DC',
       'KGAirprt_HB', 'keshod_DC', 'NCplxDPP_D', 'SsnRdDPP_D',
       'Mamlatdr_DC', 'BrDbleRd_D', 'Yadvgiri_IP', 'NarsipuraRd_D',
       'Narasipr_D', 'SulthnRd_D', 'Jogeshwri_L', 'JublieRD_D',
       'Thomas_D', 'BegurRD_D', 'TirupDPP_D', 'Santalpr_D', 'AtoNgrRd_I',
       'Gajuwaka_IP', 'LaxmiNgr_D', 'NrdawDPP_D', 'Trnsport_H',
       'Central_H_1', 'UdhamNgr_H', 'Kundli_H', 'Patelfli_D',
```

```
       'Rohini_DPC', 'KtsiGrsm_D', 'ARBNorth_DC', 'Pngktgudi_D',
       'Thalthru_DC', 'Sttyapar_D', 'Vasanthm_I', 'Bypasrd_D',
       'Mohan_Nagar_DPC', 'Madhavaram_L', 'MotvdDPP_D', 'Vaghasi_IP',
       'NlgaonRd_D', 'Srnwsngr_D', 'Aswningr_I', 'Sec 02_DPC',
'Dakor_DC',
       'GandhiRd_D', 'EastmnRD_D', 'VkotRoad_D', 'Thirumtr_IP',
       'NJVNgr_D', 'SelamRd_D', 'GMukrDPP_D', 'GariDPP_D', 'Central_I_1',
       'MP Nagar', 'Phaphamu_DC', 'Porur_DPC', 'Perungudi_DPC',
       'AkhirDPP_D', 'GModDPP_D', 'IndstlAr_I', 'Raiprvlg_L',
'Jhilmil_L',
       'Central_D_2', 'KoilStrt_D', 'Haridwar', 'Nzbadrd_D', 'Xroad_D',
       'Devenply_I', 'SuryaDPP_D', 'Dwaraka_D', 'Menagrdn_D', 'JKRoad_D',
       'Mayapuri_PC', 'Hoodi_IP', 'NvygRDPP_D', 'CrossRD_D',
'PhrmPlza_D',
       'Banikatt_D', 'Sulgwan_D', 'Indsarea_D', 'Dhelu_D', 'UBamdDPP_D',
       'AchneraRD_D', 'JPNagar_Pc', 'KHRoad_I', 'Maheva_D', 'Chowk_D',
       'BkgnRoad_D', 'TahsilRD_D', 'Kishangarh_DPC', 'Kuntikna_H',
       'CharRsta_D', 'CottonGreen_DPC', 'CikhliRD_D', 'PunjabiB_L',
       'Kengeri_IP', 'Indira Nagar', 'Peenya_IP', 'Sirikona_H',
       'Khandeshwar_Dc', 'ClgRDDPP_D', 'Alwal_L', 'StatonRD_D',
       'Pashan DPC', 'CP', 'KetyDPP_D', 'OstwlEmp_D', 'BaljiDPP_D',
       'Khenewa_D', 'Mhbhirab_D', 'RSRoad_D', 'Balajicly_I',
'Artmclny_D',
```

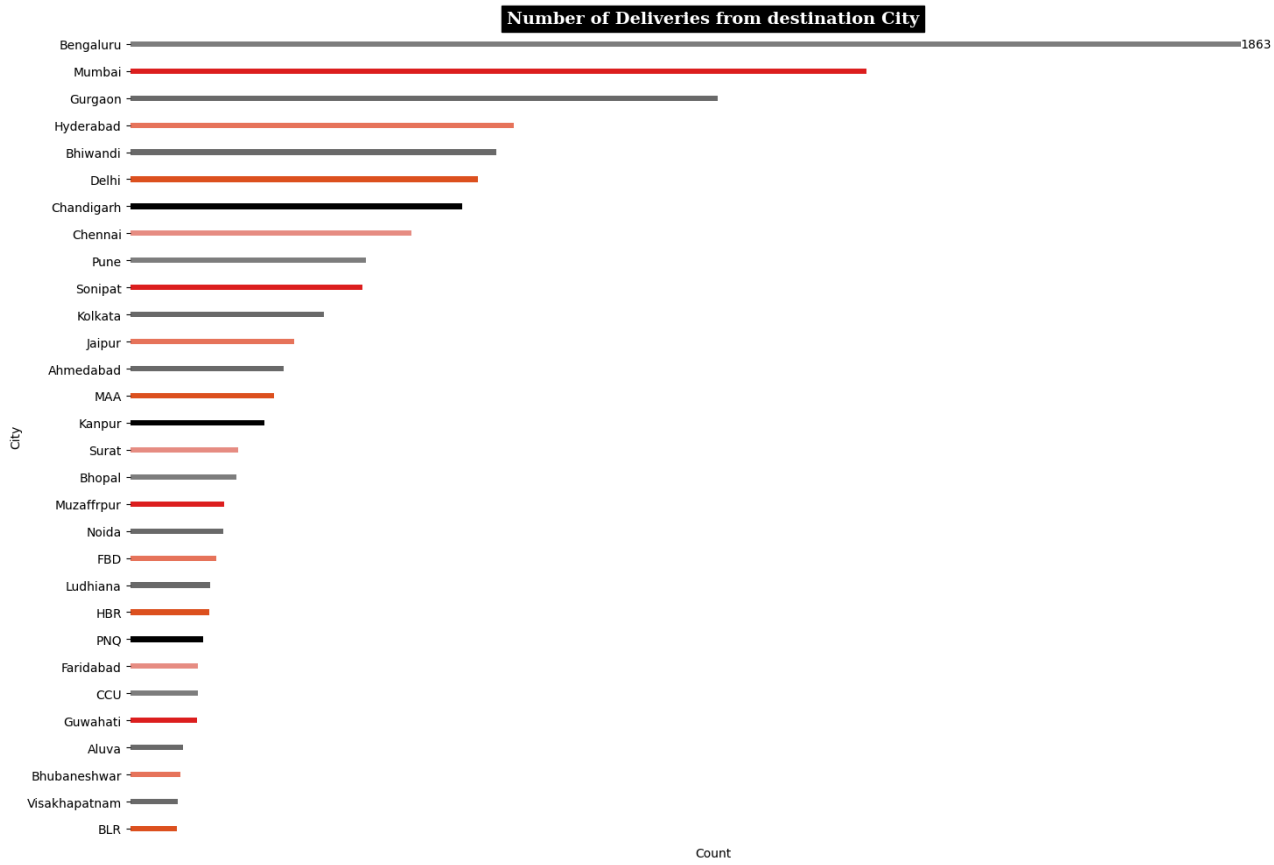
```
de['destination_state'].unique()
```

```
array(['Haryana', 'Uttar Pradesh', 'Karnataka', 'Punjab', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Andhra Pradesh', 'Telangana',
       'Rajasthan', 'Madhya Pradesh', 'Assam', 'West Bengal',
       'Chandigarh', 'Dadra and Nagar Haveli', 'Orissa', 'Uttarakhand',
       'Bihar', 'Jharkhand', 'Pondicherry', 'Goa', 'Himachal Pradesh',
       'Kerala', 'Arunachal Pradesh', 'Mizoram', 'Chhattisgarh',
       'Jammu & Kashmir', 'Meghalaya', 'Nagaland', 'Tripura',
       'Daman & Diu'], dtype=object)
```

```
state_counts = de['destination_state'].value_counts().to_frame().reset_index()
state_counts.columns = ['State', 'Count']

plt.figure(figsize=(15,10))
a = sns.barplot(y='State', x='Count', data=state_counts,palette=cp,width=0.2)
a.bar_label(a.containers[0], label_type='edge')
plt.xticks([])
plt.ylabel('State')
plt.xlabel('Count')
plt.title('Number of Deliveries to Destination State',fontsize=12,fontfamily='ser
plt.tight_layout()
sns.despine(bottom=True,left=True)
plt.show()
```


Number of Deliveries to Destination State
Karnataka
Maharashtra
Tamil Nadu
Haryana
Uttar Pradesh
Telangana
Gujarat
West Bengal
Andhra Pradesh
Rajasthan
Punjab
Bihar
Kerala
Madhya Pradesh
Delhi
Assam
Uttarakhand
Jharkhand
Orissa
Himachal Pradesh
Chandigarh
Goa
Arunachal Pradesh
Chhattisgarh
Jammu & Kashmir
Pondicherry
Dadra and Nagar Haveli
Meghalaya
Mizoram
Tripura
Nagaland
Daman & Diu
3497
State
Count

```
city_counts = de['destination_city'].value_counts().to_frame().reset_index()[:30]
city_counts.columns = ['City', 'Count']

plt.figure(figsize=(15,10))
a = sns.barplot(y='City', x='Count', data=city_counts,palette=cp,width=0.2)
a.bar_label(a.containers[0], label_type='edge')
plt.xticks([])
plt.ylabel('City')
plt.xlabel('Count')
plt.title('Number of Deliveries from destination City',fontsize=14,fontfamily='se
plt.tight_layout()
sns.despine(bottom=True,left=True)
plt.show()
```


Number of Deliveries from destination City
Bengaluru
Mumbai
Gurgaon
Hyderabad
Bhiwandi
Delhi
Chandigarh
Chennai
Pune
Sonipat
Kolkata
Jaipur
Ahmedabad
MAA
Kanpur
Surat
Bhopal
Muzaffrpur
Noida
FBD
Ludhiana
HBR
PNQ
Faridabad
CCU
Guwahati
Aluva
Bhubaneshwar
Visakhapatnam
BLR
1863
City
Count

Insights:

> Destination State

- States like **Karnataka**, **Maharashtra**, **Tamil Nadu**, **Haryana**, and **Uttar Pradesh** where maximum packages are received in this month indicating significant engagement.

> Destination City

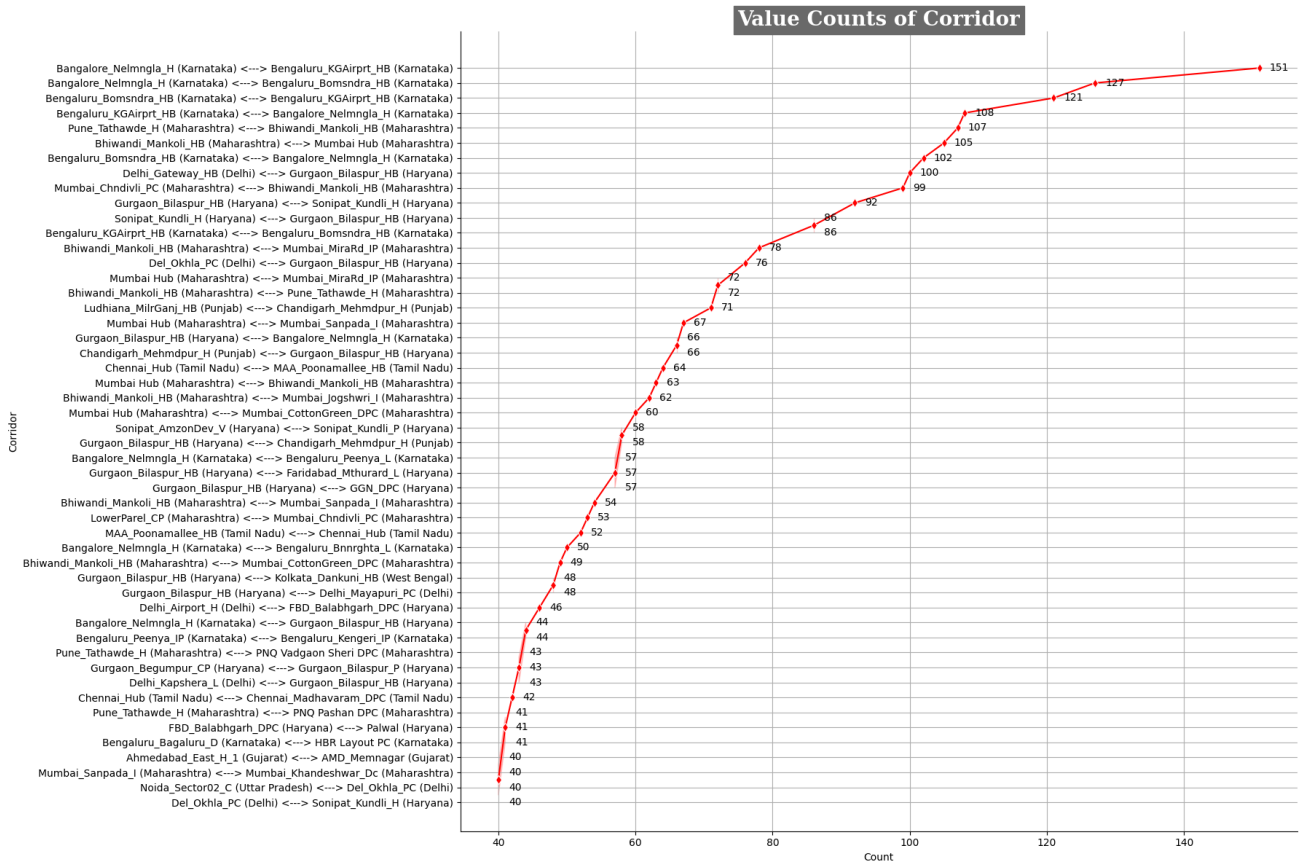
- Cities like **Bengaluru**, **Mumbai**, **Gurgaon**, **Bhiwandi**, **Hyderabad**, **Delhi** where the major no.of booking are received.

```
np.set_printoptions(threshold=np.inf)
```

```
de['corridor'] = de['source_name'] +' <---> '+ de['destination_name']
de['corridor'].value_counts()
```

```
corridor
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_KGAirprt_HB (Karnataka)
151
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_Bomsndra_HB (Karnataka)
127
Bengaluru_Bomsndra_HB (Karnataka) <---> Bengaluru_KGAirprt_HB (Karnataka)
121
Bengaluru_KGAirprt_HB (Karnataka) <---> Bangalore_Nelmngla_H (Karnataka)
108
Pune_Tathawde_H (Maharashtra) <---> Bhiwandi_Mankoli_HB (Maharashtra)
107

...
Ongole_SubhVRTL_I (Andhra Pradesh) <---> Kandukur_LICOffce_D (Andhra Pradesh)
1
Madnapalle_PngnrRd_D (Andhra Pradesh) <---> Palamaner_Lakshmi_D (Andhra
Pradesh)      1
Dharmavram_SaiNgr_D (Andhra Pradesh) <---> Kadiri_GVManu_D (Andhra Pradesh)
1
Baharampur_Chuanpur_I (West Bengal) <---> Chapra_NagarDPP_D (West Bengal)
1
Jaipur_NgrNigam_DC (Rajasthan) <---> Jaipur_Central_D_1 (Rajasthan)
1
Name: count, Length: 2741, dtype: int64
```

```
corridor_counts = de['corridor'].value_counts()[:50]

plt.figure(figsize=(18,12))
#corridor_counts.plot(kind='line', marker='d', color='r')
sns.lineplot(y=corridor_counts.index, x=corridor_counts.values, marker='d', color
plt.title('Value Counts of Corridor',fontsize=20,fontfamily='serif',fontweight='b
plt.ylabel('Corridor')
plt.xlabel('Count')
plt.tight_layout()
sns.despine()
plt.grid(True)

for i, count in enumerate(corridor_counts.values):
    plt.text(count+1.5, corridor_counts.index[i], str(count), ha='left', va='cent

plt.show()
```


Value Counts of Corridor
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_KGAirprt_HB (Karnataka) 151
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_Bomsndra_HB (Karnataka) 127
Bengaluru_Bomsndra_HB (Karnataka) <---> Bengaluru_KGAirprt_HB (Karnataka) 121
Bengaluru_KGAirprt_HB (Karnataka) <---> Bangalore_Nelmngla_H (Karnataka) 108
Pune_Tathawde_H (Maharashtra) <---> Bhiwandi_Mankoli_HB (Maharashtra) 107
Bhiwandi_Mankoli_HB (Maharashtra) <---> Mumbai Hub (Maharashtra) 105
Bengaluru_Bomsndra_HB (Karnataka) <---> Bangalore_Nelmngla_H (Karnataka) 102
Delhi_Gateway_HB (Delhi) <---> Gurgaon_Bilaspur_HB (Haryana) 100
Mumbai_Chndivli_PC (Maharashtra) <---> Bhiwandi_Mankoli_HB (Maharashtra) 99
Gurgaon_Bilaspur_HB (Haryana) <---> Sonipat_Kundli_H (Haryana) 92
Sonipat_Kundli_H (Haryana) <---> Gurgaon_Bilaspur_HB (Haryana) 86
Bengaluru_KGAirprt_HB (Karnataka) <---> Bengaluru_Bomsndra_HB (Karnataka) 86
Bhiwandi_Mankoli_HB (Maharashtra) <---> Mumbai_MiraRd_IP (Maharashtra) 78
Del_Okhla_PC (Delhi) <---> Gurgaon_Bilaspur_HB (Haryana) 76
Mumbai Hub (Maharashtra) <---> Mumbai_MiraRd_IP (Maharashtra) 72
Bhiwandi_Mankoli_HB (Maharashtra) <---> Pune_Tathawde_H (Maharashtra) 72
Ludhiana_MilrGanj_HB (Punjab) <---> Chandigarh_Mehmdpur_H (Punjab) 71
Mumbai Hub (Maharashtra) <---> Mumbai_Sanpada_I (Maharashtra) 67
Gurgaon_Bilaspur_HB (Haryana) <---> Bangalore_Nelmngla_H (Karnataka) 66
Chandigarh_Mehmdpur_H (Punjab) <---> Gurgaon_Bilaspur_HB (Haryana) 66
Chennai_Hub (Tamil Nadu) <---> MAA_Poonamallee_HB (Tamil Nadu) 64
Mumbai Hub (Maharashtra) <---> Bhiwandi_Mankoli_HB (Maharashtra) 63
Bhiwandi_Mankoli_HB (Maharashtra) <---> Mumbai_Jogshwri_I (Maharashtra) 62
Mumbai Hub (Maharashtra) <---> Mumbai_CottonGreen_DPC (Maharashtra) 60
Sonipat_AmzonDev_V (Haryana) <---> Sonipat_Kundli_P (Haryana) 58
Gurgaon_Bilaspur_HB (Haryana) <---> Chandigarh_Mehmdpur_H (Punjab) 58
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_Peenya_L (Karnataka) 57
Gurgaon_Bilaspur_HB (Haryana) <---> Faridabad_Mthurard_L (Haryana) 57
Gurgaon_Bilaspur_HB (Haryana) <---> GGN_DPC (Haryana) 57
Bhiwandi_Mankoli_HB (Maharashtra) <---> Mumbai_Sanpada_I (Maharashtra) 54
LowerParel_CP (Maharashtra) <---> Mumbai_Chndivli_PC (Maharashtra) 53
MAA_Poonamallee_HB (Tamil Nadu) <---> Chennai_Hub (Tamil Nadu) 52
Bangalore_Nelmngla_H (Karnataka) <---> Bengaluru_Bnrghta_L (Karnataka) 50
Bhiwandi_Mankoli_HB (Maharashtra) <---> Mumbai_CottonGreen_DPC (Maharashtra) 49
Gurgaon_Bilaspur_HB (Haryana) <---> Kolkata_Dankuni_HB (West Bengal) 48
Gurgaon_Bilaspur_HB (Haryana) <---> Delhi_Mayapuri_PC (Delhi) 48
Delhi_Airport_H (Delhi) <---> FBD_Balabhgarh_DPC (Haryana) 46
Bangalore_Nelmngla_H (Karnataka) <---> Gurgaon_Bilaspur_HB (Haryana) 44
Bengaluru_Peenya_IP (Karnataka) <---> Bengaluru_Kengeri_IP (Karnataka) 44
Pune_Tathawde_H (Maharashtra) <---> PNQ Vadgaon Sheri DPC (Maharashtra) 43
Gurgaon_Begumpur_CP (Haryana) <---> Gurgaon_Bilaspur_P (Haryana) 43
Delhi_Kapshera_L (Delhi) <---> Gurgaon_Bilaspur_HB (Haryana) 43
Chennai_Hub (Tamil Nadu) <---> Chennai_Madhavaram_DPC (Tamil Nadu) 42
Pune_Tathawde_H (Maharashtra) <---> PNQ Pashan DPC (Maharashtra) 41
FBD_Balabhgarh_DPC (Haryana) <---> Palwal (Haryana) 41
Bengaluru_Bagaluru_D (Karnataka) <---> HBR Layout PC (Karnataka) 41
Ahmedabad_East_H_1 (Gujarat) <---> AMD_Memnagar (Gujarat) 40
Mumbai_Sanpada_I (Maharashtra) <---> Mumbai_Khandeshwar_Dc (Maharashtra) 40
Noida_Sector02_C (Uttar Pradesh) <---> Del_Okhla_PC (Delhi) 40
Del_Okhla_PC (Delhi) <---> Sonipat_Kundli_H (Haryana) 40
Corridor
40
60
80
100
120
140
Count

Insights:

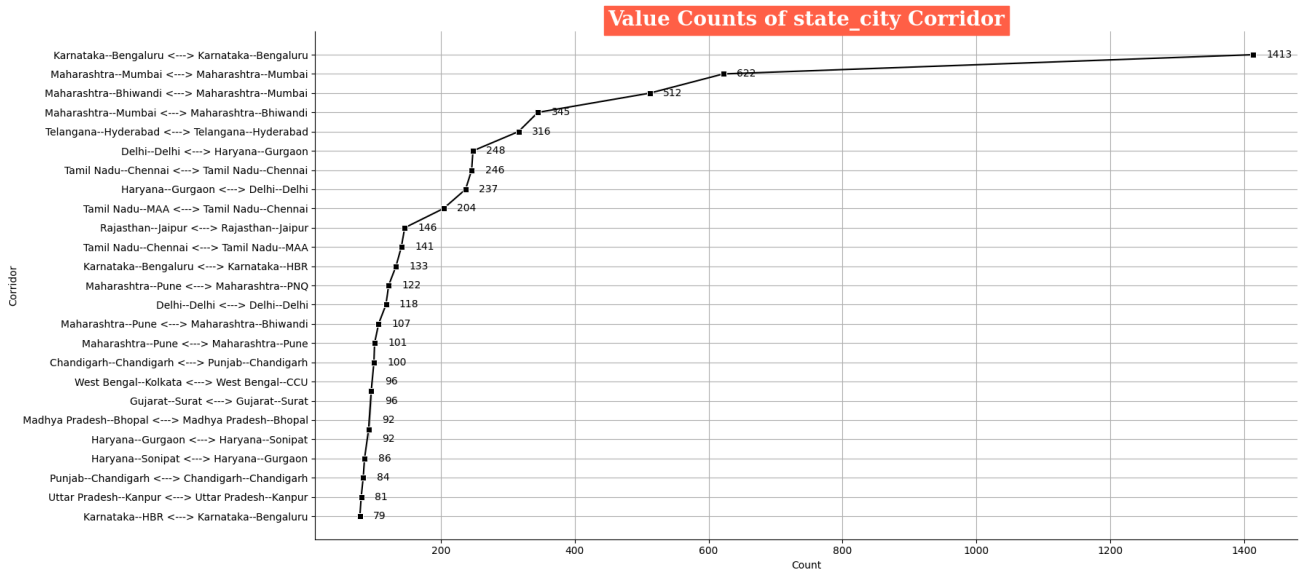
- The route between **Bangalore_Nelamangala_H** to **Bengaluru_KGAirport_HB**,**Bengaluru_Bomsndra_HB** sees the highest package volume, with 151 and 127 packages sent respectively.
- **Bengaluru_Bommasandra_HB** to **Bengaluru_KGAirport_HB** is also popular, with 121 packages sent.
- **Bengaluru_KGAirport_HB** to **Bangalore_Nelamangala_H** has moderate activity, with 108 packages sent.

4. The data indicates Bengaluru's importance as a transportation hub `Corridor` within ***Karnataka***, handling significant package traffic.

```
de['state_corridor'] = de['source_state']+'--'+de['source_city'] +' <---> '+ de['
de['state_corridor'].value_counts()
```

```
state_corridor
Karnataka--Bengaluru <---> Karnataka--Bengaluru                  1413
Maharashtra--Mumbai <---> Maharashtra--Mumbai                     622
Maharashtra--Bhiwandi <---> Maharashtra--Mumbai                   512
Maharashtra--Mumbai <---> Maharashtra--Bhiwandi                   345
Telangana--Hyderabad <---> Telangana--Hyderabad                   316
                                                                 ...
Gujarat--Jetpur <---> Gujarat--Dhoraji                              1
Andhra Pradesh--Anakapalle <---> Andhra Pradesh--Visakhapatnam      1
Andhra Pradesh--Narsiptnm <---> Andhra Pradesh--Anakapalle          1
West Bengal--MirzapurWB <---> West Bengal--Kolkata                  1
Uttar Pradesh--Anandnagar <---> Uttar Pradesh--Gorakhpur            1
Name: count, Length: 2302, dtype: int64
```

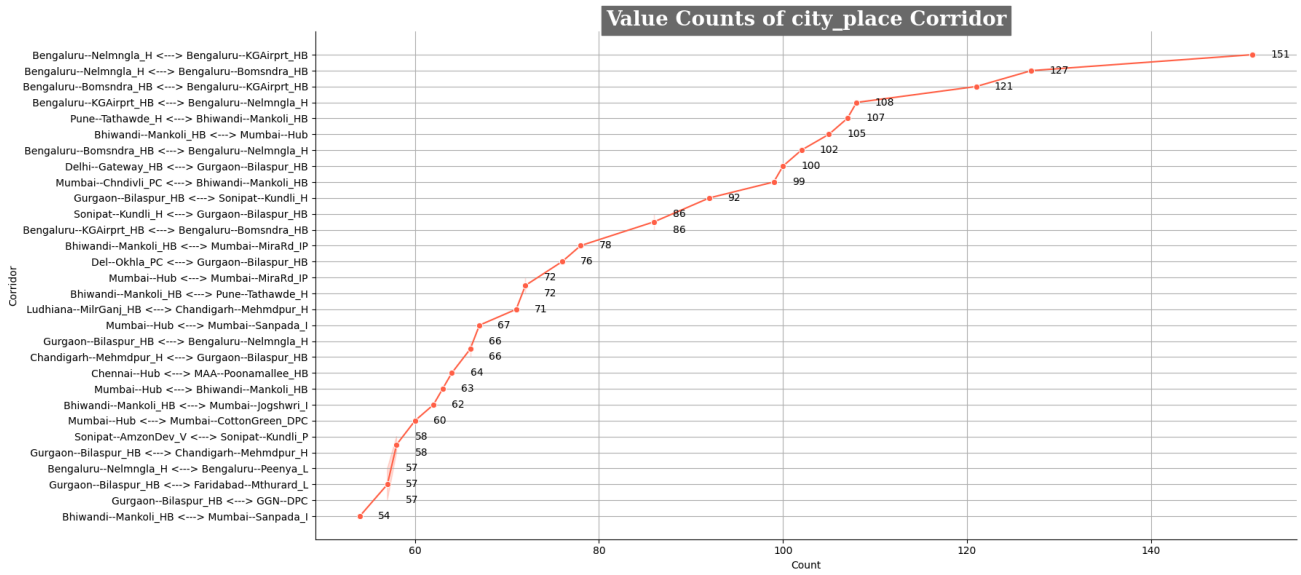
```
state_corridor_counts = de['state_corridor'].value_counts()[:25]

plt.figure(figsize=(18,8))
sns.lineplot(y=state_corridor_counts.index, x=state_corridor_counts.values, marke
plt.title('Value Counts of state_city Corridor',fontsize=20,fontfamily='serif',fo
plt.ylabel('Corridor')
plt.xlabel('Count')
plt.tight_layout()
sns.despine()
plt.grid(True)

for i, count in enumerate(state_corridor_counts.values):
    plt.text(count+20, state_corridor_counts.index[i], str(count), ha='left', va=

plt.show()
```




```
de['city_corridor'] = de['source_city']+'--'+de['source_place'] +' <---> '+ de['d
display(de['city_corridor'].value_counts())
```

```
city_corridor
Bengaluru--Nelmngla_H <---> Bengaluru--KGAirprt_HB     151
Bengaluru--Nelmngla_H <---> Bengaluru--Bomsndra_HB     127
Bengaluru--Bomsndra_HB <---> Bengaluru--KGAirprt_HB    121
Bengaluru--KGAirprt_HB <---> Bengaluru--Nelmngla_H     108
Pune--Tathawde_H <---> Bhiwandi--Mankoli_HB            107
                                                      ...
Ongole--SubhVRTL_I <---> Kandukur--LICOffce_D            1
Madnapalle--PngnrRd_D <---> Palamaner--Lakshmi_D         1
Dharmavram--SaiNgr_D <---> Kadiri--GVManu_D              1
Baharampur--Chuanpur_I <---> Chapra--NagarDPP_D          1
Jaipur--NgrNigam_DC <---> Jaipur--Central_D_1            1
Name: count, Length: 2741, dtype: int64
```

```
city_corridor_counts = de['city_corridor'].value_counts()[:30]

plt.figure(figsize=(18,8))
sns.lineplot(y=city_corridor_counts.index, x=city_corridor_counts.values, marker=
plt.title('Value Counts of city_place Corridor',fontsize=20,fontfamily='serif',fo
plt.ylabel('Corridor')
plt.xlabel('Count')
plt.tight_layout()
sns.despine()
plt.grid(True)

for i, count in enumerate(city_corridor_counts.values):
    plt.text(count+2, city_corridor_counts.index[i], str(count), ha='left', va='c

plt.show()
```



Insights:

- Maharashtra, Karnataka, Haryana, and Tamil Nadu serve as key starting and ending locations for delivery services.
- Mumbai, Gurgaon, Delhi, and Bengaluru are major metropolitan centers from where many deliveries originate.
- A large proportion of nationwide deliveries are destined for Mumbai, Bengaluru, Gurgaon, and Delhi.

```
# 4. Extracting features like month, year, day, etc. from Trip_creation_time
de['trip_creation_month'] = de['trip_creation_time'].dt.month
de['trip_creation_year'] = de['trip_creation_time'].dt.year
de['trip_creation_day'] = de['trip_creation_time'].dt.day
de['trip_creation_hour'] = de['trip_creation_time'].dt.hour
de['trip_creation_weekday'] = de['trip_creation_time'].dt.weekday
de['trip_creation_week'] = de['trip_creation_time'].dt.isocalendar().week
de
```

	segment_key	trip_uuid	data	r
0	trip-153671041653548748+IND209304AAA+IND000000ACB	trip-153671041653548748	training	
1	trip-153671041653548748+IND462022AAA+IND209304AAA	trip-153671041653548748	training	
2	trip-153671042288605164+IND561203AAB+IND562101AAA	trip-153671042288605164	training	
3	trip-153671042288605164+IND572101AAA+IND561203AAB	trip-153671042288605164	training	
4	trip-153671043369099517+IND000000ACB+IND160002AAC	trip-153671043369099517	training	
...	...	...	...	
26217	trip-153861115439069069+IND628204AAA+IND627657AAA	trip-153861115439069069	test	
26218	trip-153861115439069069+IND628613AAA+IND627005AAA	trip-153861115439069069	test	
26219	trip-153861115439069069+IND628801AAA+IND628204AAA	trip-153861115439069069	test	
26220	trip-153861118270144424+IND583119AAA+IND583101AAA	trip-153861118270144424	test	
26221	trip-153861118270144424+IND583201AAA+IND583119AAA	trip-153861118270144424	test	

26222 rows × 37 columns

🔷In-Depth Analysis

```
new_df = de.copy()
```

```
new_df.columns
```

```
Index(['segment_key', 'trip_uuid', 'data', 'route_type',
'trip_creation_time',
       'source_name', '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',
'segment_actual_time_sum',
       'segment_osrm_time_sum', 'segment_osrm_distance_sum', 'od_total_time',
       'od_time_diff_hour', 'source_city', 'source_place', 'source_state',
       'destination_city', 'destination_place', 'destination_state',
       'corridor', 'state_corridor', 'city_corridor', 'trip_creation_month',
       'trip_creation_year', 'trip_creation_day', 'trip_creation_hour',
       'trip_creation_weekday', 'trip_creation_week'],
      dtype='object')
```

```
new_df.sample(2)
```

	segment_key	trip_uuid	data	r
7726	trip-153723402751707419+IND792103AAB+IND792001AAA	trip-153723402751707419	training	
24122	trip-153844014707745337+IND141109AAA+IND000000ACA	trip-153844014707745337	test	

```
create_trip_dict={
  'data' : 'first',
  'route_type' : 'first',
  'od_start_time':'first',
  'od_end_time':'last',
  'od_time_diff_hour' : 'sum',
  'trip_creation_time' : 'first',
  'trip_creation_month' : 'first',
  'trip_creation_year' : 'first',
  'trip_creation_day' : 'first',
  'trip_creation_hour' : 'first',
  'trip_creation_weekday' : 'first',
  'trip_creation_week' : 'first',
  'start_scan_to_end_scan' : 'sum',
  'actual_distance_to_destination' : 'sum',
  'actual_time' : 'sum',
  'osrm_time' : 'sum',
  'osrm_distance' : 'sum',
  'segment_actual_time': 'sum',
  'segment_osrm_time': 'sum',
  'segment_osrm_distance': 'sum',
  'segment_actual_time_sum': 'sum',
  'segment_osrm_time_sum': 'sum',
  'segment_osrm_distance_sum': 'sum',
  'source_name': 'first',
  'source_city':'first',
  'source_state':'first',
  'source_place':'first',
  'destination_name': 'first',
  'destination_city':'first',
  'destination_state':'first',
  'destination_place':'first',
  'corridor':'first',
  'state_corridor':'first',
  'city_corridor':'first'
  }

trip_agg_df = new_df.groupby('trip_uuid').agg(create_trip_dict).reset_index()
trip_agg_df
```

	trip_uuid	data	route_type	od_start_time	od_end_time	od_t
0	trip-153671041653548748	training	FTL	2018-09-12 16:39:46.858469	2018-09-12 16:39:46.858469	
1	trip-153671042288605164	training	Carting	2018-09-12 02:03:09.655591	2018-09-12 02:03:09.655591	
2	trip-153671043369099517	training	FTL	2018-09-14 03:40:17.106733	2018-09-14 03:40:17.106733	
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	2018-09-12 03:54:43.114421	
...	...	...	...	...	...	
14782	trip-153861095625827784	test	Carting	2018-10-03 23:55:56.258533	2018-10-04 06:41:25.409035	
14783	trip-153861104386292051	test	Carting	2018-10-03 23:57:23.863155	2018-10-04 00:57:59.294434	
14784	trip-153861106442901555	test	Carting	2018-10-04 02:51:27.075797	2018-10-04 02:51:27.075797	
14785	trip-153861115439069069	test	Carting	2018-10-03 23:59:14.390954	2018-10-04 02:29:04.272194	
14786	trip-153861118270144424	test	FTL	2018-10-04 03:58:40.726547	2018-10-04 03:58:40.726547	

14787 rows × 35 columns

```
numerical_columns = trip_agg_df.select_dtypes(include=[np.float32, np.float64])
numerical_columns
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	37.668497	2259.0	824.7328
1	3.026865	180.0	73.1869
2	65.572709	3933.0	1927.4042
3	1.674916	100.0	17.1752
4	11.972484	717.0	127.4485
...	...	...	
14782	4.300482	257.0	57.7623
14783	1.009842	60.0	15.5137
14784	7.035331	421.0	38.6848
14785	5.808548	347.0	134.7238
14786	5.906793	353.0	66.0815

14787 rows × 12 columns

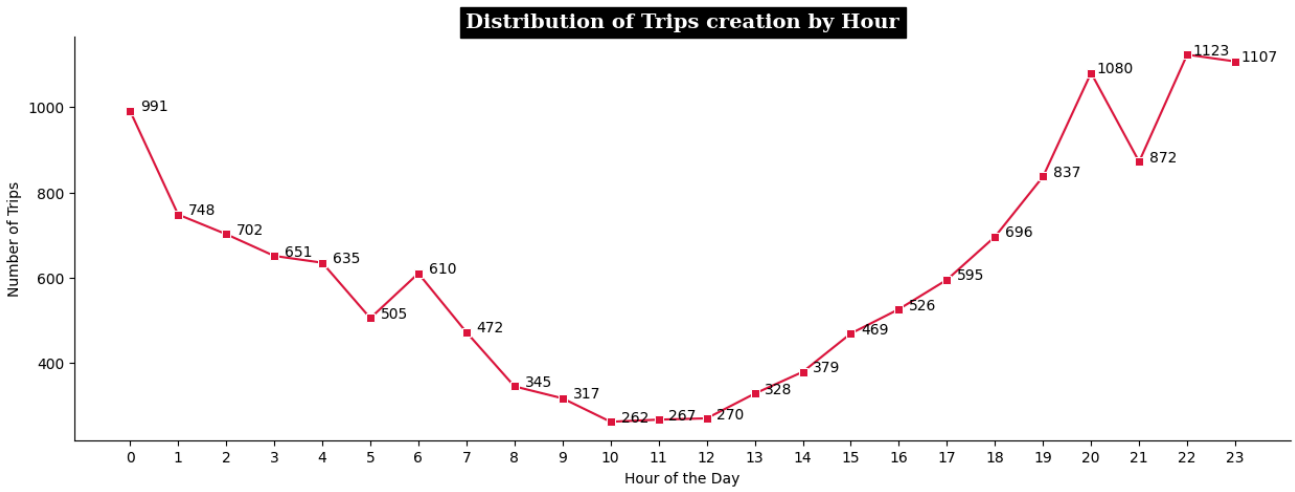
```
numerical_columns.describe().T
```

	count	mean	std	min	25%	
od_time_diff_hour	14787.0	8.840187	10.978880	0.391024	2.494975	
start_scan_to_end_scan	14787.0	529.429016	658.254944	23.000000	149.000000	27
actual_distance_to_destination	14787.0	164.090195	305.502991	9.002461	22.777099	4
actual_time	14787.0	356.306000	561.517944	9.000000	67.000000	14
osrm_time	14787.0	160.990936	271.459503	6.000000	29.000000	6
osrm_distance	14787.0	203.887405	370.565552	9.072900	30.756900	6
segment_actual_time	14787.0	353.059174	556.365906	9.000000	66.000000	14
segment_osrm_time	14787.0	180.511597	314.679291	6.000000	30.000000	6
segment_osrm_distance	14787.0	222.705444	416.846283	9.072900	32.578850	6
segment_actual_time_sum	14787.0	353.059174	556.365906	9.000000	66.000000	14
segment_osrm_time_sum	14787.0	180.511597	314.679291	6.000000	30.000000	6

```
trip_agg_df.describe(include = object).T
```

	count	unique	top	freq
trip_uuid	14787	14787	trip-153671041653548748	1
source_name	14787	930	Gurgaon_Bilaspur_HB (Haryana)	1052
source_city	14787	713	Bengaluru	1700
source_state	14787	29	Maharashtra	2714
source_place	14787	788	Bilaspur_HB	1052
destination_name	14787	1042	Gurgaon_Bilaspur_HB (Haryana)	745
destination_city	14787	851	Bengaluru	1633
destination_state	14787	32	Maharashtra	2569
destination_place	14787	866	Bilaspur_HB	745
corridor	14787	1737	Bangalore_Nelmngla_H (Karnataka) <---> Bengalu...	151
state_corridor	14787	1366	Karnataka--Bengaluru <---> Karnataka--Bengaluru	1333
city_corridor	14787	1737	Bengaluru--Nelmngla_H <---> Bengaluru--KGAirpr...	151

```
trip_df = trip_agg_df.copy()


trip_creation_by_hour = trip_df.groupby(by='trip_creation_hour')['trip_uuid'].cou

plt.figure(figsize=(15,5))
sns.lineplot(data=trip_creation_by_hour, x='trip_creation_hour', y='trip_uuid', m
plt.xticks(np.arange(0, 24))

for i, count in enumerate(trip_creation_by_hour['trip_uuid']):
    plt.text(trip_creation_by_hour['trip_creation_hour'][i]+0.5, count, count, ha

plt.title('Distribution of Trips creation by Hour',fontsize=14,fontfamily='serif'
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
sns.despine()
plt.show()
```




```
trip_df.sample()
```

	trip_uuid	data	route_type	od_start_time	od_end_time	od_tim
11573	trip-153814251919205555	test	Carting	2018-09-28 13:48:39.192312	2018-09-28 18:02:28.698462	

```
trip_df.trip_creation_year.value_counts()
```

```
trip_creation_year
2018    14787
Name: count, dtype: int64
```

```
trip_df.trip_creation_month.value_counts()
```

```
trip_creation_month
9     13011
10     1776
Name: count, dtype: int64
```

```
trip_df['trip_creation_month'].value_counts(normalize = True) * 100
```

```
trip_creation_month
9     87.98945
```

```
10    12.01055
Name: proportion, dtype: float64
```

```
trip_df.trip_creation_week.value_counts()
```

```
trip_creation_week
38    5001
39    4402
37    3608
40    1776
Name: count, dtype: Int64
```

```
trip_df.trip_creation_weekday.value_counts(ascending=True)
```

```
trip_creation_weekday
6    1753
0    1980
1    2035
4    2057
3    2103
5    2128
2    2731
Name: count, dtype: int64
```

```
trip_df['trip_creation_day_week'] = trip_df['trip_creation_time'].dt.day_name()
```

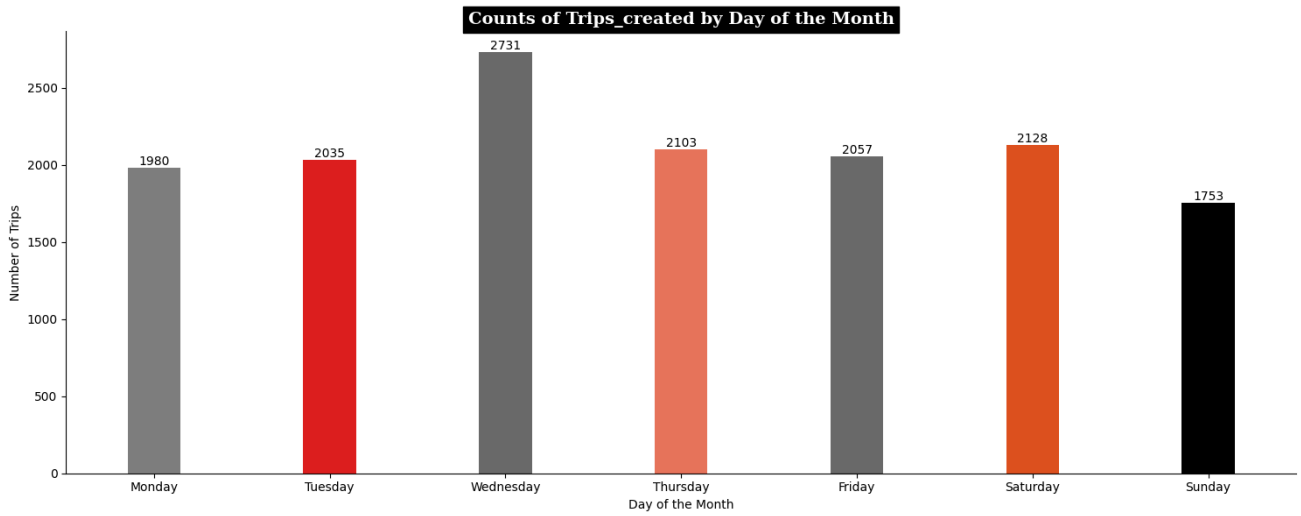
```
trip_df.trip_creation_day.value_counts()
```

```
trip_creation_day
18    791
15    783
13    750
12    747
21    740
22    740
17    722
14    712
20    703
25    695
26    683
19    674
24    658
27    650
23    631
3     627
16    616
28    605
29    605
1     600
2     549
30    506
Name: count, dtype: int64
```

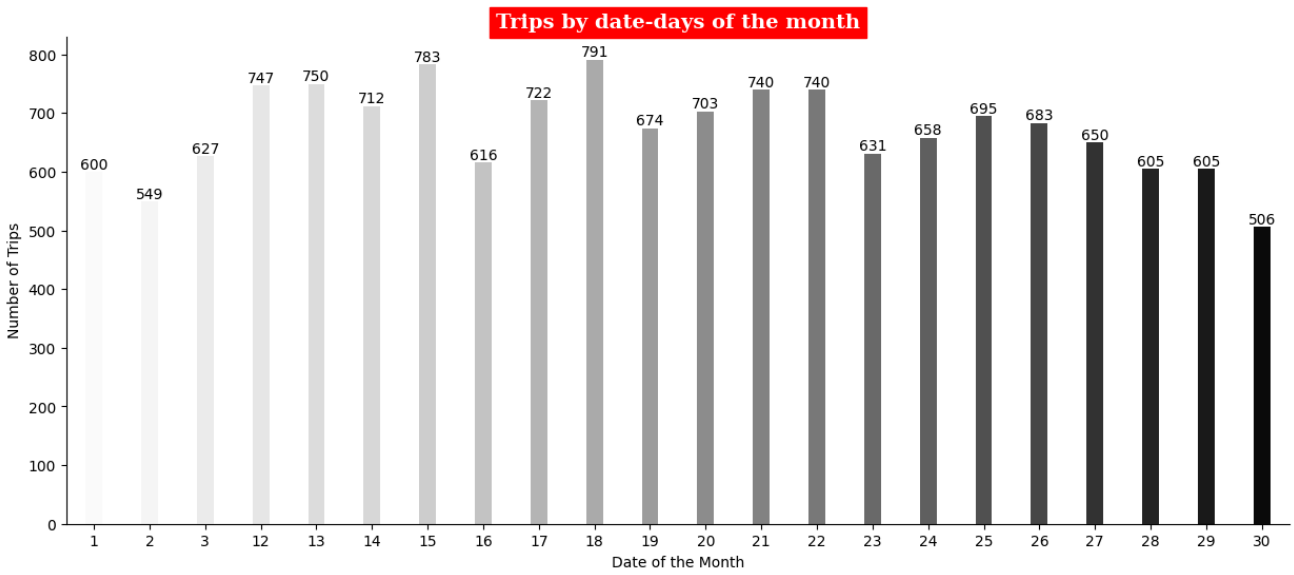
```
trip_df.sample()
```

	trip_uuid	data	route_type	od_start_time	od_end_time	od_tim
10648	trip-153800659468028518	test	Carting	2018-09-27 02:37:15.362086	2018-09-27 04:21:45.871140	

```
plt.figure(figsize=(15,6))

weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturda
day_counts = trip_df['trip_creation_day_week'].value_counts().reindex(weekday_ord

sns.barplot(x=day_counts.index, y=day_counts.values, palette=cp,width=0.3)
for i, count in enumerate(day_counts.values):
    plt.text(i, count, str(count), ha='center', va='bottom')
plt.title('Counts of Trips_created by Day of the Month',fontsize=14,fontfamily='s
plt.xlabel('Day of the Month')
plt.ylabel('Number of Trips')
plt.tight_layout()
sns.despine()
plt.show()
```




```
trip_df['trip_creation_dayofdate'] = trip_df['trip_creation_time'].dt.day


trips_by_dateday = trip_df.groupby(by = 'trip_creation_dayofdate')['trip_uuid'].c

plt.figure(figsize = (15, 6))
sns.barplot(data =trip_df,x = trips_by_dateday['trip_creation_dayofdate'],y = tri
for i, count in enumerate(trips_by_dateday['trip_uuid']):
    plt.text(i, count, str(count), ha='center', va='bottom')
plt.title('Trips by date-days of the month',fontsize=14,fontfamily='serif',fontwe
plt.xlabel('Date of the Month')
plt.ylabel('Number of Trips')
sns.despine()
plt.show()
```



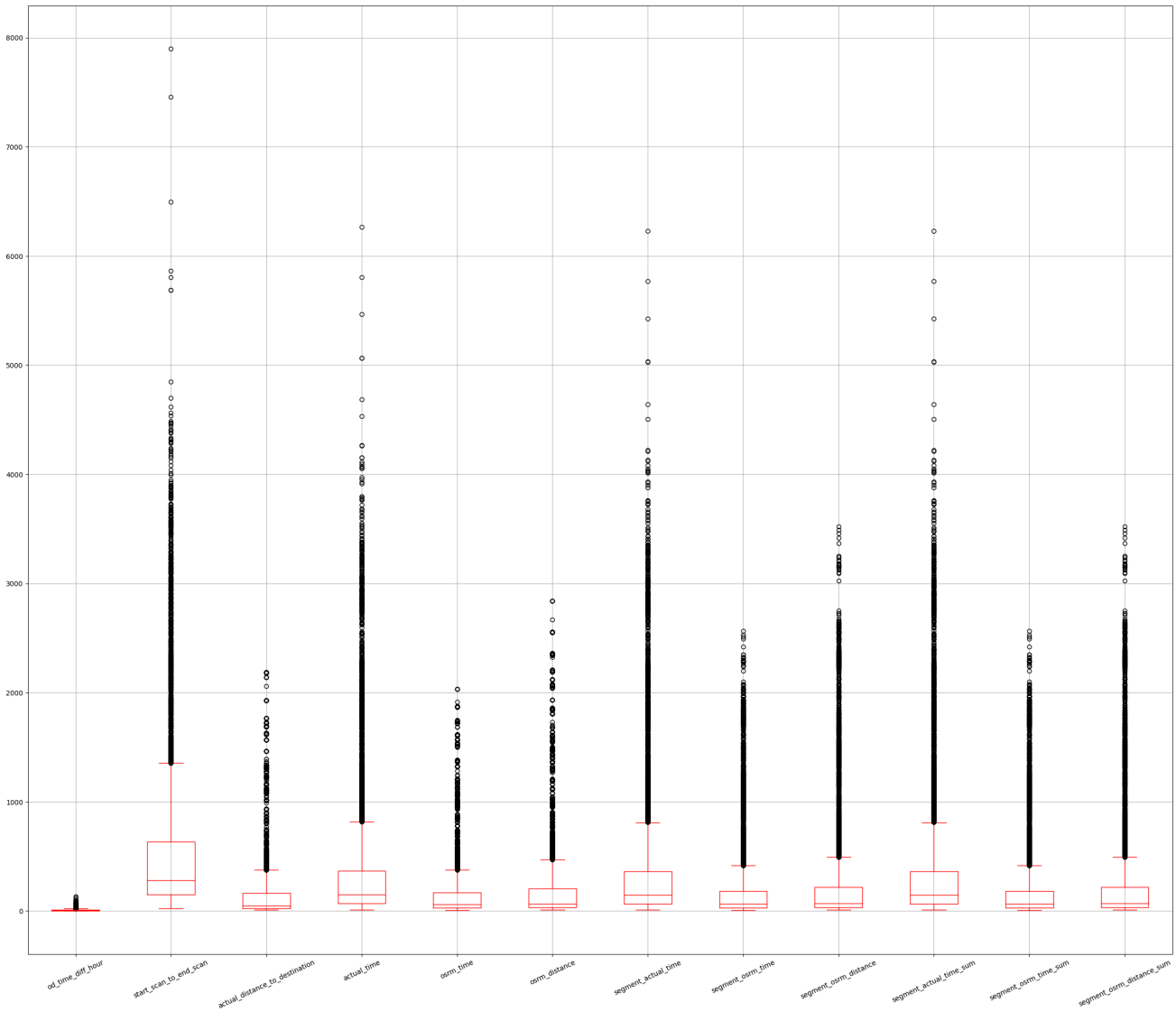
📈Outlier treatment

`numerical_columns`

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	37.668497	2259.0	824.7328
1	3.026865	180.0	73.1869
2	65.572709	3933.0	1927.4042
3	1.674916	100.0	17.1752
4	11.972484	717.0	127.4485
...	...	...	
14782	4.300482	257.0	57.7623
14783	1.009842	60.0	15.5137
14784	7.035331	421.0	38.6848
14785	5.808548	347.0	134.7238
14786	5.906793	353.0	66.0815

14787 rows × 12 columns

```
plt.figure(figsize=(30, 25))
numerical_columns.boxplot(rot=25, figsize=(35,20), color = 'r')
plt.grid('off')
plt.show()
```




```
numerical_columns.columns
```

```
Index(['od_time_diff_hour', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'segment_actual_time_sum',
       'segment_osrm_time_sum', 'segment_osrm_distance_sum'],
      dtype='object')
```

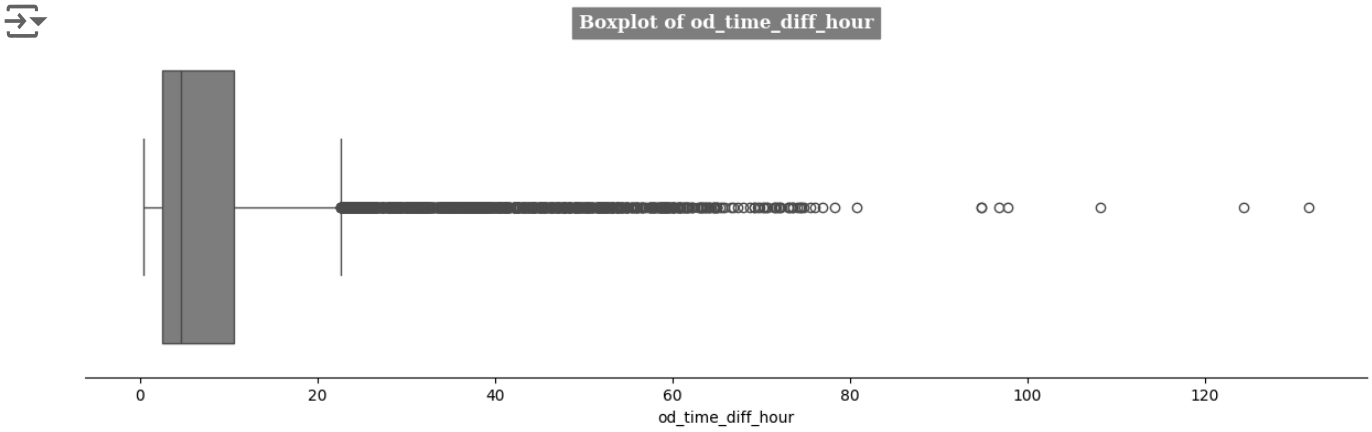
```
num_cols = numerical_columns.columns.tolist()
num_cols
```

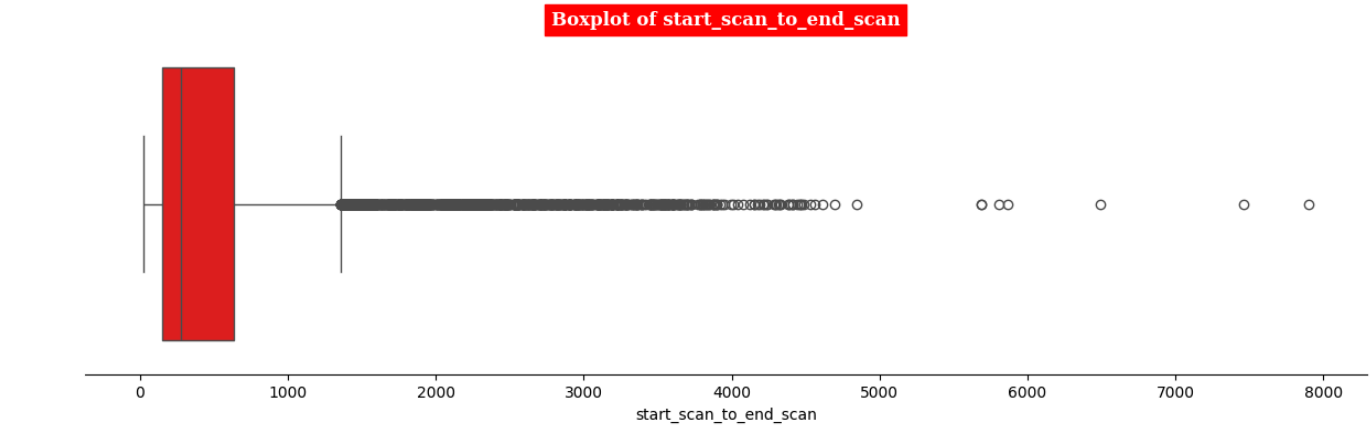
```
['od_time_diff_hour',
 'start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance',
 'segment_actual_time_sum',
 'segment_osrm_time_sum',
 'segment_osrm_distance_sum']
```

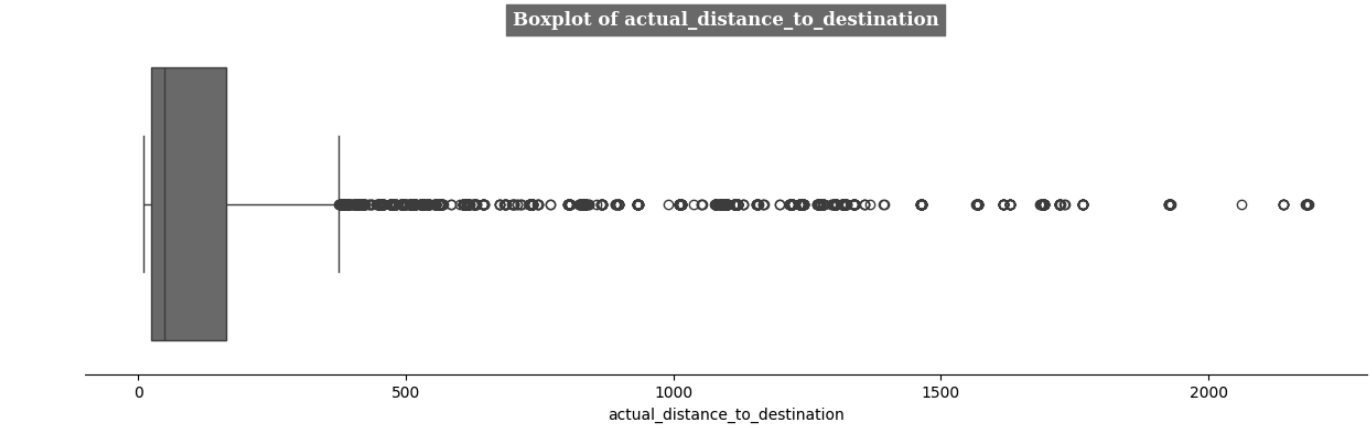
```
# obtain the first quartile
Q1 = numerical_columns.quantile(0.25)

# obtain the third quartile
Q3 = numerical_columns.quantile(0.75)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

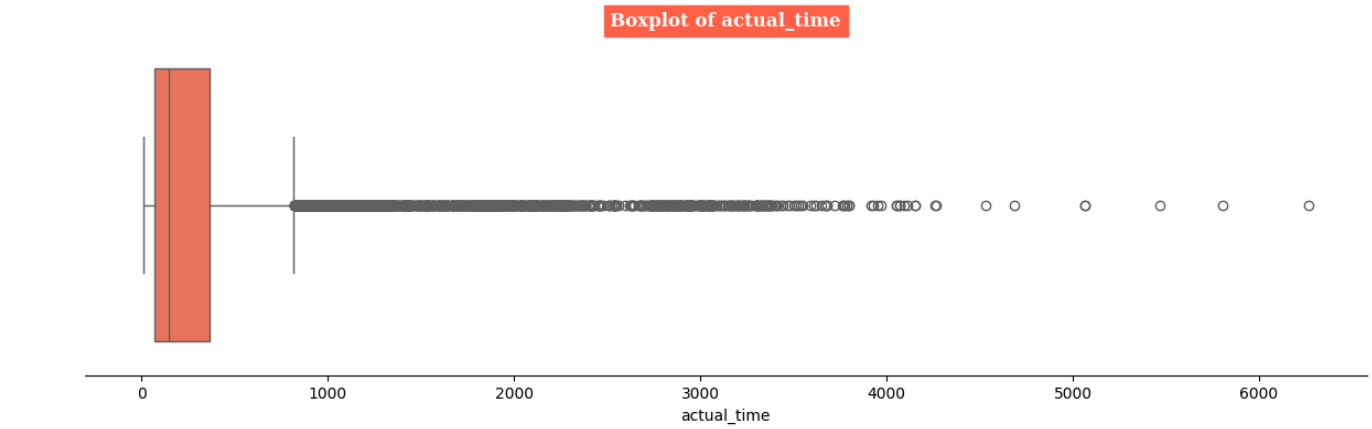
```
od_time_diff_hour                  8.063987
start_scan_to_end_scan           483.000000
actual_distance_to_destination   140.814157
actual_time                      300.000000
osrm_time                        139.000000
osrm_distance                    175.887303
segment_actual_time              298.000000
segment_osrm_time                154.000000
segment_osrm_distance            183.981758
segment_actual_time_sum          298.000000
segment_osrm_time_sum            154.000000
segment_osrm_distance_sum        183.981758
dtype: float64
```

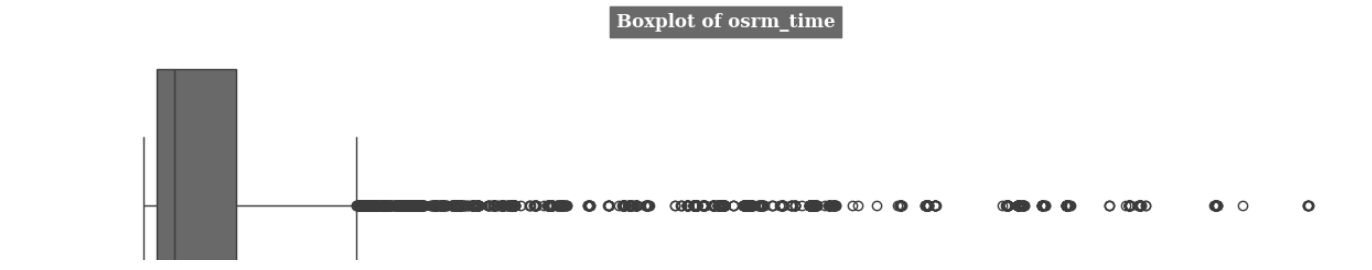
```
    for i,col in enumerate(numerical_columns):
        plt.figure(figsize=(15,4))
        sns.boxplot(x=col, data=numerical_columns,color=cp[i])
        sns.despine(left=True)
        plt.yticks([])
        plt.title(f'Boxplot of {col}',fontfamily='serif',fontweight='bold',fontsize=1
        plt.show()
```


Boxplot of od_time_diff_hour
0
20
40
60
80
100
120
od_time_diff_hour


Boxplot of start_scan_to_end_scan
0
1000
2000
3000
4000
5000
6000
7000
8000
start_scan_to_end_scan


Boxplot of actual_distance_to_destination
0
500
1000
1500
2000
actual_distance_to_destination


Boxplot of actual_time
0
1000
2000
3000
4000
5000
6000
actual_time


Boxplot of osrm_time

✅Outlier Removal

```
for i, col in enumerate(numerical_columns):

    data = trip_df[col]
    display(data.to_frame())


    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1


    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    clipped_data = np.clip(data, lower_bound, upper_bound)
    print(f'Clipped data of {col}')
    display(clipped_data.to_frame())
    print()

    # Plot boxplot of the clipped data
    plt.figure(figsize=(15, 4))
    plt.subplot(121)
    sns.boxplot(x=clipped_data, color=cp[i])
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of clipped {col}', fontfamily='serif', fontweight='bold',

    filtered_data = data.loc[(data >= lower_bound) | (data <= upper_bound)]
    print(f'Filtered data of {col}')
    display(filtered_data.to_frame())
    print()

    plt.subplot(122)
    sns.boxplot(x=filtered_data, color=cp[i])
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of filtered {col}', fontfamily='serif', fontweight='bold'

    plt.show()
```

	od_time_diff_hour
0	37.668497
1	3.026865
2	65.572709
3	1.674916
4	11.972484
...	...
14782	4.300482
14783	1.009842
14784	7.035331
14785	5.808548
14786	5.906793

14787 rows × 1 columns

```
Clipped data of od_time_diff_hour
```

	od_time_diff_hour
0	22.654942
1	3.026865
2	22.654942
3	1.674916
4	11.972484
...	...
14782	4.300482
14783	1.009842
14784	7.035331
14785	5.808548
14786	5.906793

14787 rows × 1 columns

```
Filtered data of od_time_diff_hour
```

	od_time_diff_hour
0	37.668497
1	3.026865
2	65.572709

Understanding:

- Here we see that the data after removing outliers has outliers. It has to be understood that q1 and q3 dont have to be always 25th percentile and 75th percentile. ***Try changing q1 and q3 to 10th percentile to 90th percentile and plot and see...***

- Clipped data replaces the outlier values with specified values.

- Here, I have proceeded with both clipped and filtered data(with reduced outliers) for further analysis.

```
num_df = numerical_columns.copy()
num_df
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	37.668497	2259.0	824.7328
1	3.026865	180.0	73.1869
2	65.572709	3933.0	1927.4042
3	1.674916	100.0	17.1752
4	11.972484	717.0	127.4485
...	...	...	
14782	4.300482	257.0	57.7623
14783	1.009842	60.0	15.5137
14784	7.035331	421.0	38.6848
14785	5.808548	347.0	134.7238
14786	5.906793	353.0	66.0815

14787 rows × 12 columns

14784 421.0

```
num_cols
```

```
['od_time_diff_hour',
 'start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance',
 'segment_actual_time_sum',
 'segment_osrm_time_sum',
 'segment_osrm_distance_sum']
```

3 100.0

```
Q1 = np.percentile(num_df[num_cols], 25)
Q3 = np.percentile(num_df[num_cols], 75)
IQR = Q3 - Q1

lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

clipped_num_df = np.clip(num_df[num_cols], lower_bound, upper_bound)
display(clipped_num_df)

filtered_num_df = num_df[num_cols][(num_df[num_cols] >= lower_bound) | (num_df[nu
display(filtered_num_df)
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	37.668497	543.285302	543.2853
1	3.026865	180.000000	73.1869
2	65.572709	543.285302	543.2853
3	1.674916	100.000000	17.1752
4	11.972484	543.285302	127.4485
...	...	...	
14782	4.300482	257.000000	57.7623
14783	1.009842	60.000000	15.5137
14784	7.035331	421.000000	38.6848
14785	5.808548	347.000000	134.7238
14786	5.906793	353.000000	66.0815

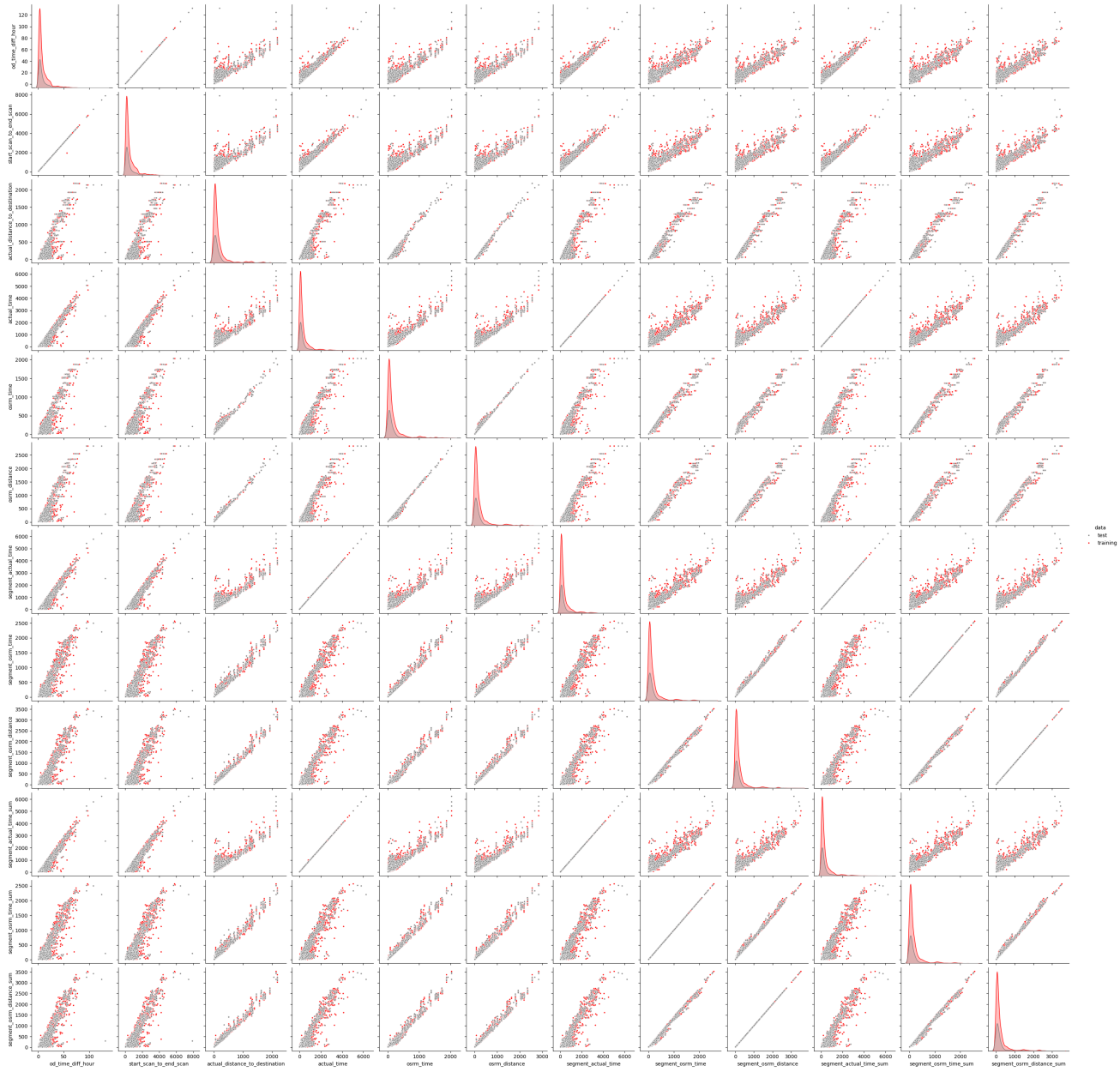
14787 rows × 12 columns

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	37.668497	2259.0	824.7328
1	3.026865	180.0	73.1869
2	65.572709	3933.0	1927.4042
3	1.674916	100.0	17.1752
4	11.972484	717.0	127.4485
...	...	...	
14782	4.300482	257.0	57.7623
14783	1.009842	60.0	15.5137
14784	7.035331	421.0	38.6848
14785	5.808548	347.0	134.7238
14786	5.906793	353.0	66.0815

14787 rows × 12 columns

```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.title(f' Pairplot Analysis',fontfamily='serif',fontweight='bold',fontsize=15,
sns.pairplot(data = trip_df,vars = num_cols,hue='data',markers = '.',palette=cp)
plt.show()
```

Pairplot Analysis

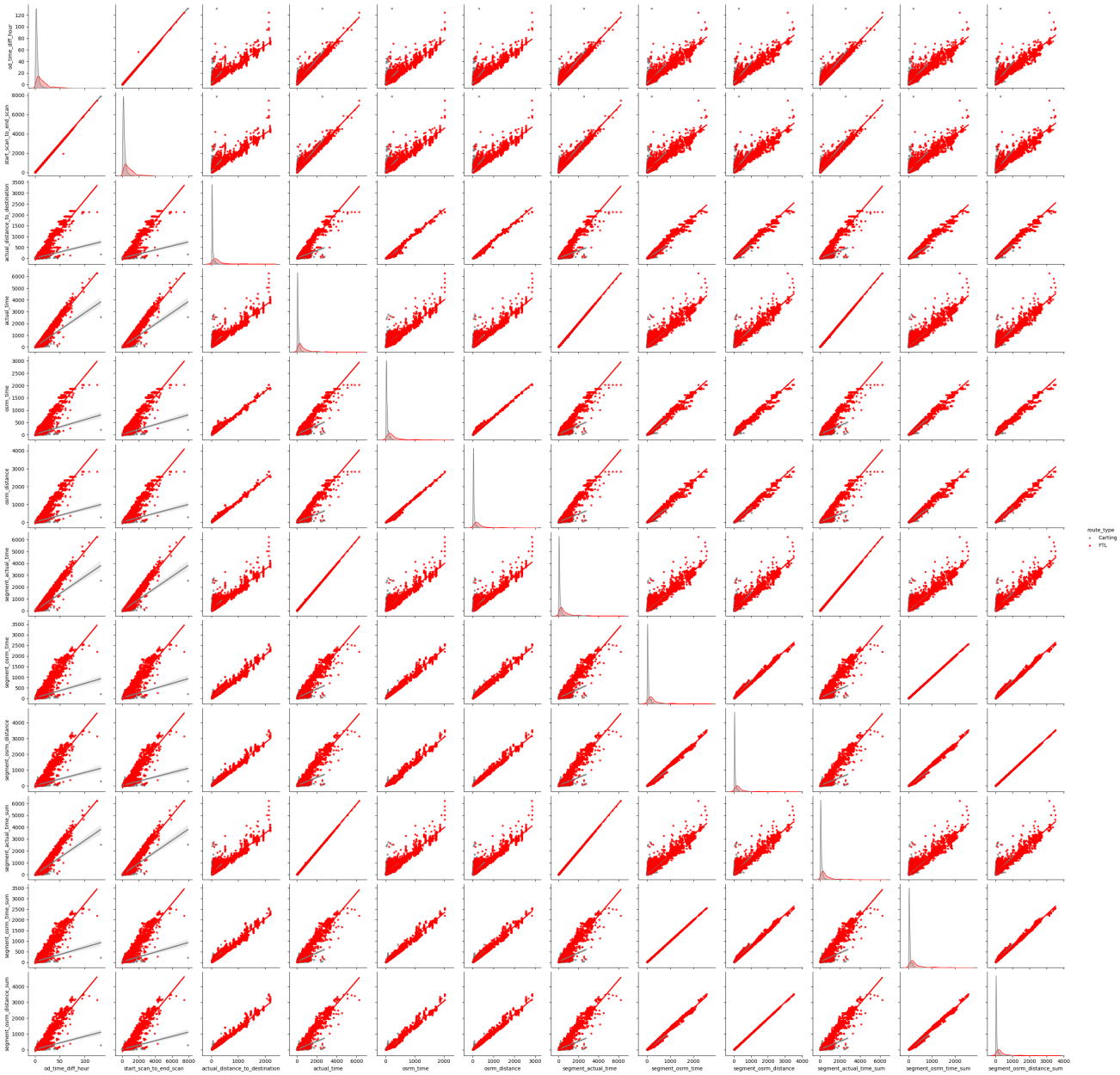


1	68.0
2	1740.0
3	15.0
4	117.0
...	...
14782	62.0
14783	18.0

14783 12.0

```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.title(f' Pairplot Analysis',fontfamily='serif',fontweight='bold',fontsize=15,
sns.pairplot(data = trip_df,vars=num_cols,kind = 'reg',hue='route_type',markers =
plt.show()
```

Pairplot Analysis

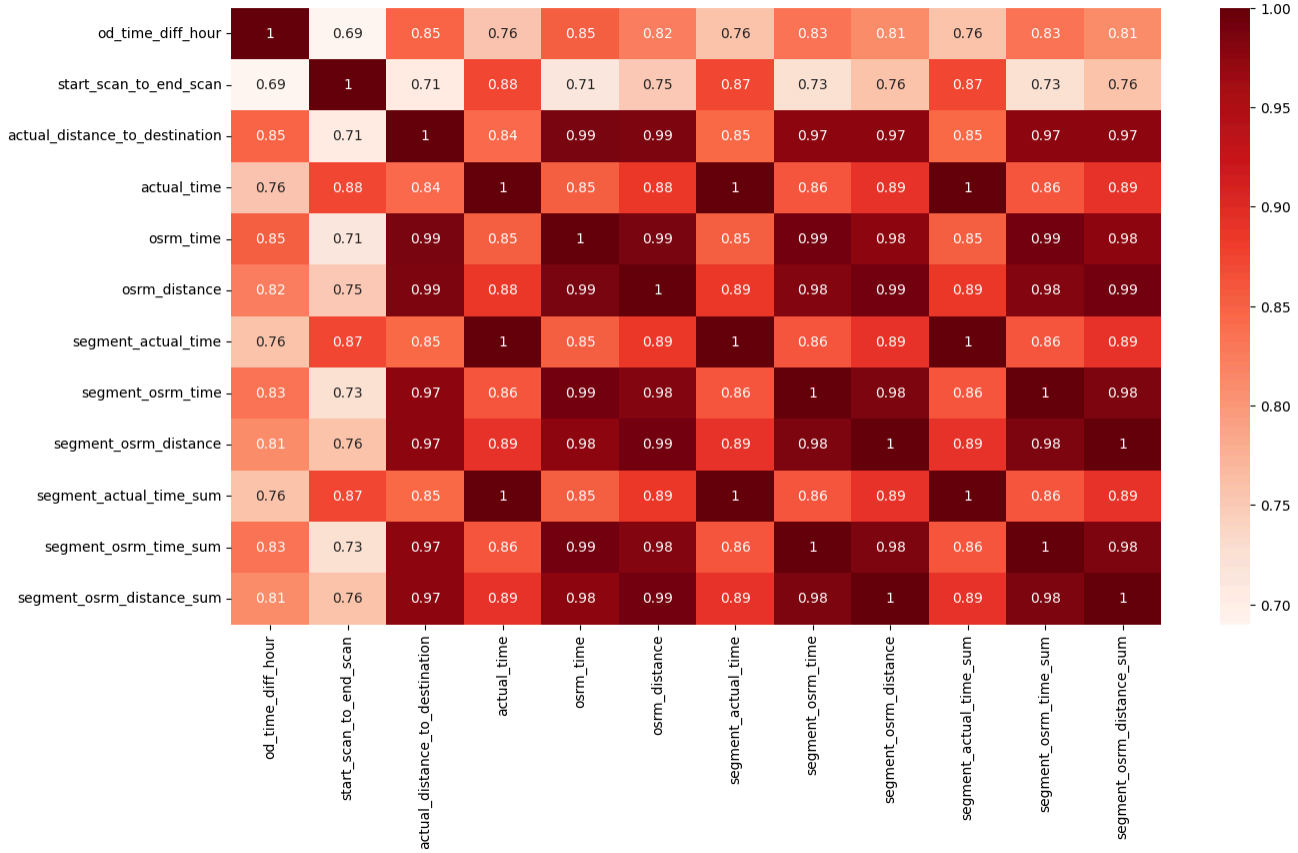


```
clipped_df_corr = clipped_num_df.corr()
clipped_df_corr

filtered_df_corr = filtered_num_df.corr()
filtered_df_corr
```

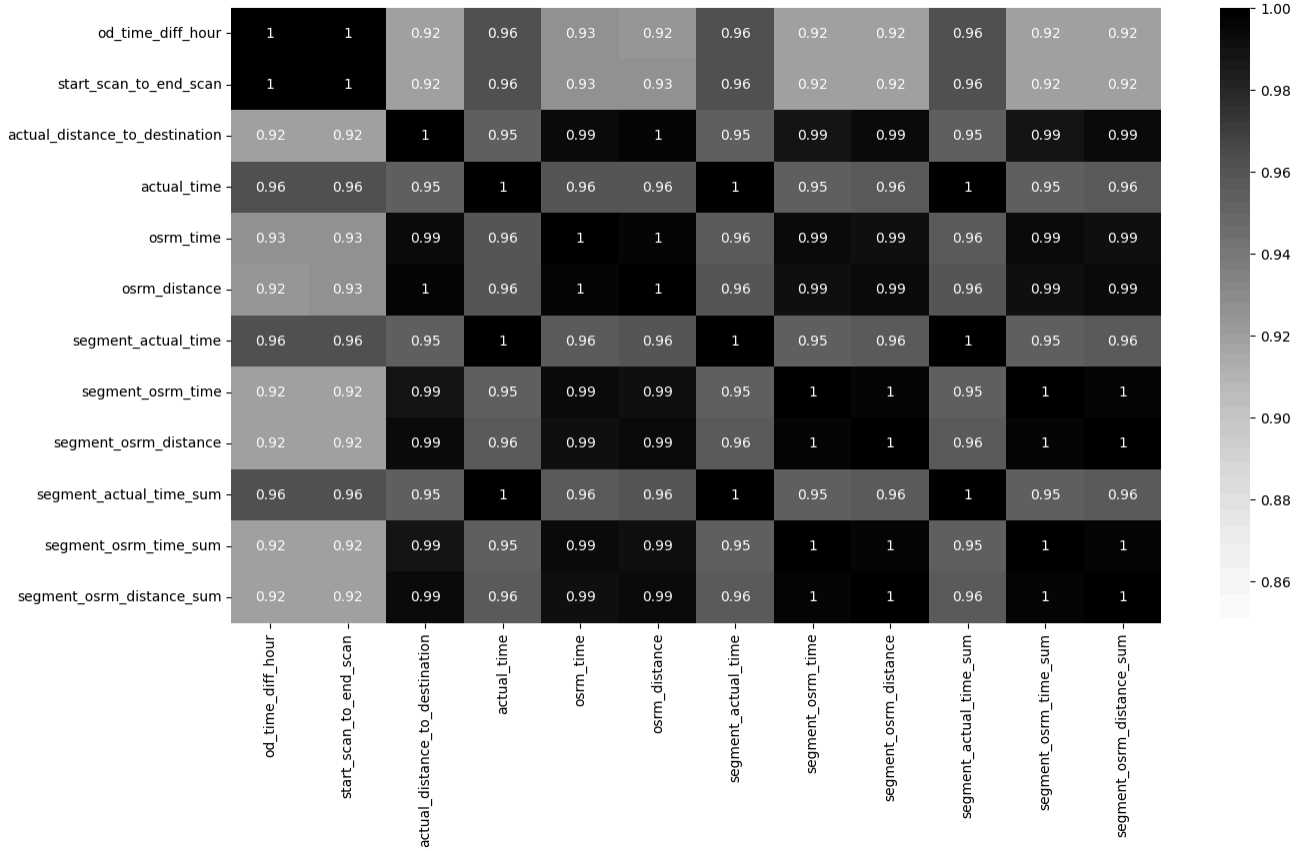
	od_time_diff_hour	start_scan_to_end_scan	actual_d
od_time_diff_hour	1.000000	0.999837	
start_scan_to_end_scan	0.999837	1.000000	
actual_distance_to_destination	0.918644	0.919159	
actual_time	0.961223	0.961612	
osrm_time	0.926973	0.927471	
osrm_distance	0.924683	0.925205	
segment_actual_time	0.961288	0.961634	
segment_osrm_time	0.918921	0.919429	
segment_osrm_distance	0.919665	0.920191	
segment_actual_time_sum	0.961288	0.961634	
segment_osrm_time_sum	0.918921	0.919429	
segment_osrm_distance_sum	0.919665	0.920191	

```
plt.figure(figsize = (15,8))
plt.suptitle(f'Correlation Analysis- clipped_df',fontfamily='serif',fontweight='b
sns.heatmap(data = clipped_df_corr,vmin=0.69, annot = True, cmap='Reds')
plt.show()

plt.figure(figsize = (15,8))
plt.suptitle(f'Correlation Analysis - filtered_df',fontfamily='serif',fontweight=
sns.heatmap(data = filtered_df_corr,vmin=0.85, annot = True, cmap='Greys')
plt.show()
```

Correlation Analysis- clipped_df



Correlation Analysis - filtered_df



Insights:

- Very High Correlation exists between all the numerical columns.

```
trip_df.skew(numeric_only = True)
```

```
od_time_diff_hour                 2.89355
trip_creation_month               2.337439
trip_creation_year                     0.0
trip_creation_day                -0.695241
trip_creation_hour               -0.206092
trip_creation_weekday             0.065904
trip_creation_week                0.181308
start_scan_to_end_scan            2.895337
actual_distance_to_destination    3.562931
actual_time                       3.375178
osrm_time                         3.455256
osrm_distance                     3.553619
segment_actual_time               3.372043
segment_osrm_time                 3.602915
segment_osrm_distance             3.714016
segment_actual_time_sum           3.372043
segment_osrm_time_sum             3.602915
segment_osrm_distance_sum         3.714016
trip_creation_dayofdate          -0.695241
dtype: Float64
```

Insights:

- We can see that Many of the data is ***Right-Skewed***.

1️⃣ one-hot encoding

```
4                 115.0
```

```
trip_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 37 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   trip_uuid                     14787 non-null  object
 1   data                          14787 non-null  category
 2   route_type                    14787 non-null  category
 3   od_start_time                 14787 non-null  datetime64[ns]
 4   od_end_time                   14787 non-null  datetime64[ns]
 5   od_time_diff_hour             14787 non-null  float64
 6   trip_creation_time            14787 non-null  datetime64[ns]
 7   trip_creation_month           14787 non-null  int32
 8   trip_creation_year            14787 non-null  int32
 9   trip_creation_day             14787 non-null  int32
 10  trip_creation_hour            14787 non-null  int32
 11  trip_creation_weekday         14787 non-null  int32
 12  trip_creation_week            14787 non-null  UInt32
 13  start_scan_to_end_scan        14787 non-null  float32
```

```
 14  actual_distance_to_destination  14787 non-null  float32
 15  actual_time                     14787 non-null  float32
 16  osrm_time                       14787 non-null  float32
 17  osrm_distance                   14787 non-null  float32
 18  segment_actual_time             14787 non-null  float32
 19  segment_osrm_time               14787 non-null  float32
 20  segment_osrm_distance           14787 non-null  float32
 21  segment_actual_time_sum         14787 non-null  float32
 22  segment_osrm_time_sum           14787 non-null  float32
 23  segment_osrm_distance_sum       14787 non-null  float32
 24  source_name                     14787 non-null  object
 25  source_city                     14787 non-null  object
 26  source_state                    14787 non-null  object
 27  source_place                    14787 non-null  object
 28  destination_name                14787 non-null  object
 29  destination_city                14787 non-null  object
 30  destination_state               14787 non-null  object
 31  destination_place               14787 non-null  object
 32  corridor                        14787 non-null  object
 33  state_corridor                  14787 non-null  object
 34  city_corridor                   14787 non-null  object
 35  trip_creation_day_week          14787 non-null  object
 36  trip_creation_dayofdate         14787 non-null  int32
dtypes: UInt32(1), category(2), datetime64[ns](3), float32(11), float64(1), i
memory usage: 3.0+ MB
```

```
categorical_cols = ['data','route_type']
```

segment_osrm_distance

```
# one hot encoding the categorical features
ohe = OneHotEncoder(sparse=False)
encoded_cat_cols = ohe.fit_transform(trip_df[categorical_cols])

categorical_encoded_df = pd.DataFrame(encoded_cat_cols, columns=ohe.get_feature_n
display(categorical_encoded_df)

encoded_df = pd.concat([trip_df,categorical_encoded_df],axis=1)
encoded_df
```

	data_test	data_training	route_type_Carting	route_type_FTL
0	0.0	1.0	0.0	1.0
1	0.0	1.0	1.0	0.0
2	0.0	1.0	0.0	1.0
3	0.0	1.0	1.0	0.0
4	0.0	1.0	0.0	1.0
...	...	...	...	...
14782	1.0	0.0	1.0	0.0
14783	1.0	0.0	1.0	0.0
14784	1.0	0.0	1.0	0.0
14785	1.0	0.0	1.0	0.0
14786	1.0	0.0	0.0	1.0

14787 rows × 4 columns

	trip_uuid	data	route_type	od_start_time	od_end_time	c
0	trip-153671041653548748	training	FTL	2018-09-12 16:39:46.858469	2018-09-12 16:39:46.858469	
1	trip-153671042288605164	training	Carting	2018-09-12 02:03:09.655591	2018-09-12 02:03:09.655591	
2	trip-153671043369099517	training	FTL	2018-09-14 03:40:17.106733	2018-09-14 03:40:17.106733	
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	2018-09-12 03:54:43.114421	
...	...	...	...	...	...	
14782	trip-153861095625827784	test	Carting	2018-10-03 23:55:56.258533	2018-10-04 06:41:25.409035	
14783	trip-153861104386292051	test	Carting	2018-10-03 23:57:23.863155	2018-10-04 00:57:59.294434	

0️⃣-1️⃣ Minmax scaler

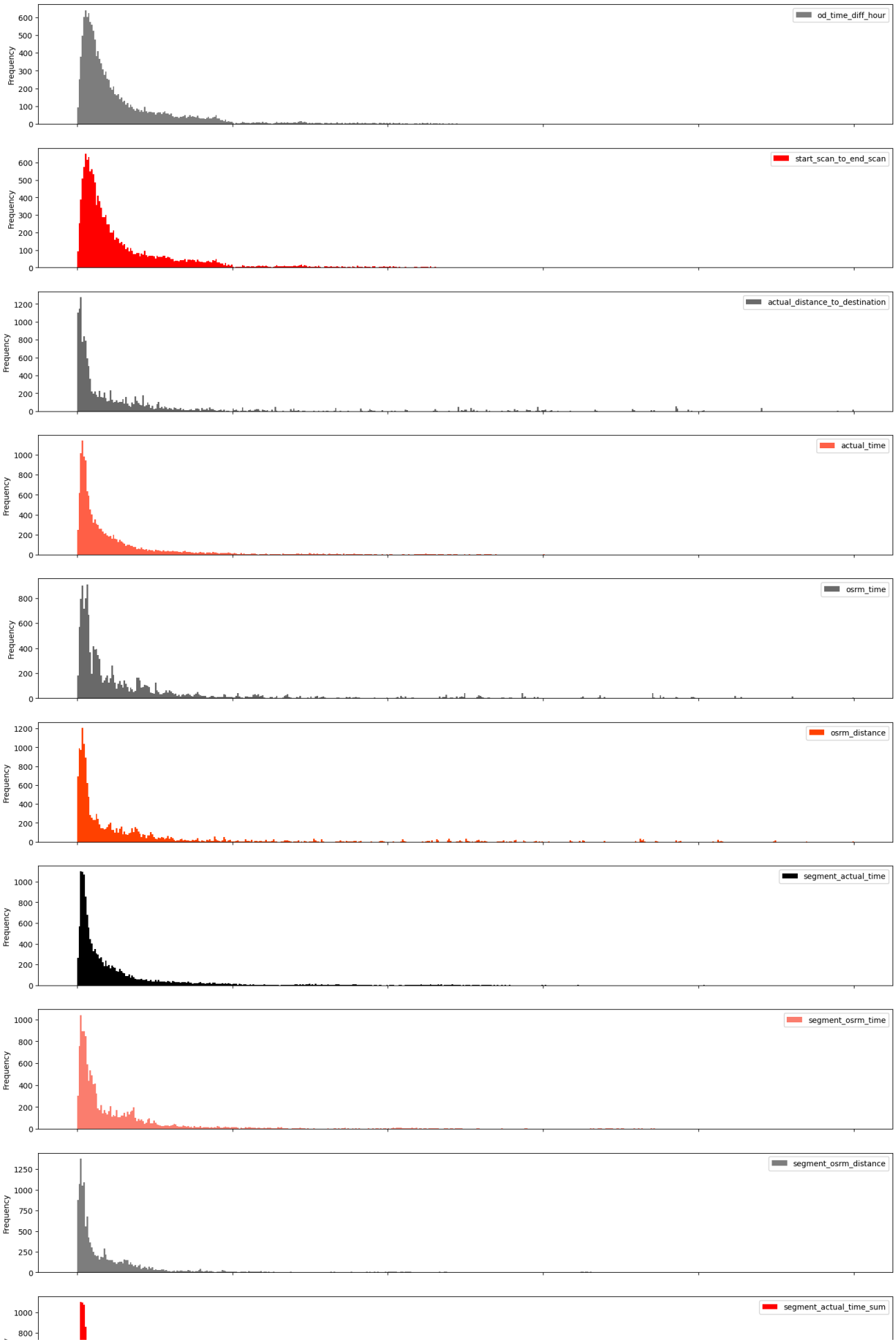
🔷 Most appropriate since the data is not gaussian

```
# Normalizing/Standardizing the numerical features using MinMaxScaler
min_max_scaler = MinMaxScaler()
min_max_scaled_numerical = min_max_scaler.fit_transform(trip_df[num_cols])

# Converting the scaled features back to a dataframe
min_max_scaled_df = pd.DataFrame(min_max_scaled_numerical, columns=num_cols)
min_max_scaled_df
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	0.284016	0.283937	0.3746
1	0.020082	0.019937	0.0294
2	0.496617	0.496508	0.8809
3	0.009782	0.009778	0.0037
4	0.088239	0.088127	0.0543
...	...	...	
14782	0.029786	0.029714	0.0223
14783	0.004715	0.004698	0.0029
14784	0.050623	0.050540	0.0136
14785	0.041276	0.041143	0.0577
14786	0.042024	0.041905	0.0262

14787 rows × 12 columns

```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.suptitle(f'Min-Max scaled visualization of num_cols',fontfamily='serif',fontw
min_max_scaled_df.plot(kind='hist', figsize=(20,40),subplots=True,color=cp,bins=5
plt.show()
```

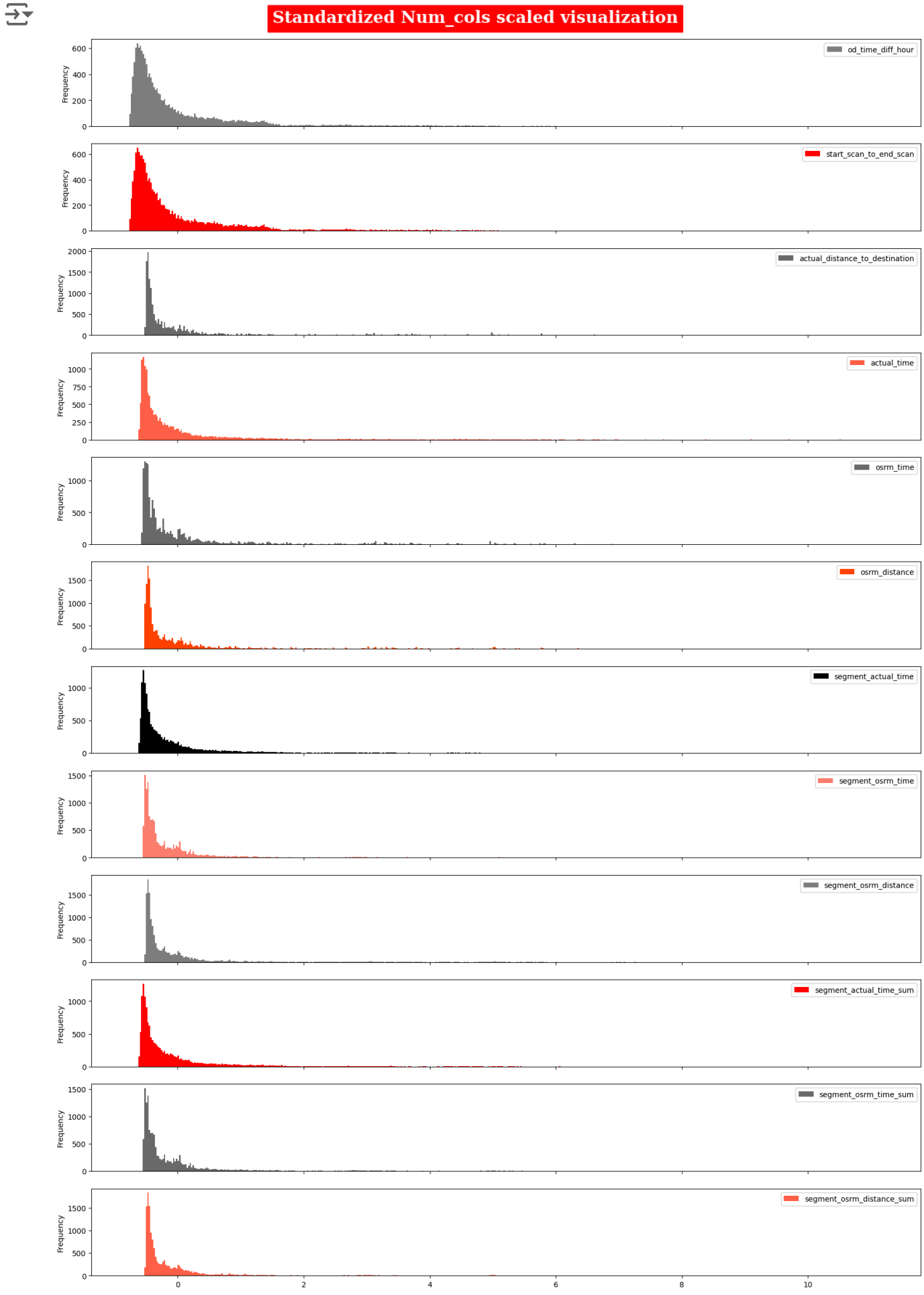

Min-Max scaled visualization of num_cols
od_time_diff_hour
start_scan_to_end_scan
actual_distance_to_destination
actual_time
osrm_time
osrm_distance
segment_actual_time
segment_osrm_time
segment_osrm_distance
segment_actual_time_sum
Frequency

```
# Just so to know ... cant do this as `data is not gaussian`
# Standardization works only with data which follows normal distribution
# Standardizing the numerical features using StandardScaler
std_scaler = StandardScaler()
std_scaled = std_scaler.fit_transform(trip_df[num_cols])

# Converting the scaled features back to a dataframe
std_scaled_df = pd.DataFrame(std_scaled, columns=num_cols)
std_scaled_df
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinati
0	2.625886	2.627598	2.1625
1	-0.529518	-0.530859	-0.2975
2	5.167598	5.170772	5.7720
3	-0.652664	-0.652397	-0.4809
4	0.285312	0.284962	-0.1199
...	...	...	
14782	-0.413508	-0.413880	-0.3480
14783	-0.713243	-0.713166	-0.4863
14784	-0.164399	-0.164728	-0.4105
14785	-0.276143	-0.277150	-0.0961
14786	-0.267194	-0.268034	-0.3208

14787 rows × 12 columns

```
plt.figure(figsize=(14,0.05))
plt.axis('off')
plt.suptitle(f'Standardized Num_cols scaled visualization',fontfamily='serif',fon
std_scaled_df.plot(kind='hist', figsize=(20,30),subplots=True,color=cp,bins=500)
plt.show()
```


Standardized Num_cols scaled visualization
od_time_diff_hour
start_scan_to_end_scan
actual_distance_to_destination
actual_time
osrm_time
osrm_distance
segment_actual_time
segment_osrm_time
segment_osrm_distance
segment_actual_time_sum
segment_osrm_time_sum
segment_osrm_distance_sum
Frequency

☝️Hypothesis Testing:

- Perform hypothesis testing / visual analysis between :

> ⚡ actual_time aggregated value and OSRM time aggregated value.

> ⚡ actual_time aggregated value and segment actual time aggregated value.

> ⚡ OSRM distance aggregated value and segment OSRM distance aggregated value.

> ⚡ OSRM time aggregated value and segment OSRM time aggregated value.

- Note: Aggregated values are the values you'll get after merging the rows on the basis of trip_uuid.

Assumptions of T-Test

1. The sample size should be less than 30.
2. The population variance is unknown.
3. The population mean and standard deviation are finite.
4. The means of the two populations being compared should follow normal distributions.
5. If using Student's original definition of the t-test, the two populations being compared should have the same variance. If the sample sizes in the two groups being compared are equal, Student's original t-test is highly robust to the presence of unequal variances.

💉 ***STEP-1*** : Set up Null Hypothesis

> **Null Hypothesis (Ho)** - There is no significant difference in the mean values between column1 and column2
> H_{o}: $μ*{col1}$ = $μ*{col2}$

> **Alternate Hypothesis (Ha)** - There is a significant difference in the mean values between column1 and column2
> H_{a}: $μ*{col1}$!= $μ*{col2}$

💉 ***STEP-2*** : Checking for basic assumpitons for the hypothesis

> Normality checks
>
> - Distribution check using **QQ Plot** & **prob Plot**

- Confirmation by **Shapiro-wilks Test**
- Confirmation by **Anderson-darling Test**
- Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under Ho.

- We know that the test statistic while performing a T-Test follows T-distribution.

> for independent variables:
>
> > If data follows normal distribution we go with **ttest_ind**
> > Else we will go with **Mannwhitney_u test** (Non - Parametric test)

> for dependent variables: (paired T-test)
>
> > If data follows normal distribution we go with **ttest_rel**
> > Else we will go with **Wilcoxon signed rank test** (Non - Parametric test)

STEP-4: Decide the kind of test.

- We will be performing **Two tailed t-test**

STEP-5: Compute the p-value and fix value of alpha.

- we will be computing the t-stat value using the ttest function using scipy.stats.
- We set our ***alpha to be 0.05 (i.e) confidence level = 95%***

STEP-6: Compare p-value and alpha.

- Based on p-value, we will accept or reject H0.

> **p-val < alpha** : Reject Ho

> **p-val > alpha** : Accept Ho

```
# def shapiro_and_anderson(name,col):

#     print(f"Performing SHAPIRO & ANDERSON DARLING TEST for {name} column")
#     print()
#     print('Shapiro Wilks Test')
#     shapiro_stat , p_val = shapiro(col)
#     if p_val < 0.05:
#         print(f'{name} - Data is not Gaussian')
#     else:
#         print(f'{name} - Data is Gaussian')
#     print()

#     print("As shapiro is sensitive, we go with ANDERSON DARLING TEST")
#     result = anderson(col)
#     if result.statistic > result.critical_values[2]:
#         print(f"{name} - Data does not follow normal distribution.")
#     else:
#         print(f"{name} - Data follows normal distribution.")
#     print()
#     print('-'*50)

#     def boxcox_tranformation(name,col):

#         print(f'Performing BOXCOX transformation on {name} column')
#         transformed_data,best_lambda = boxcox(col)

    #     return " "


# class NormalCheck:
#     def __init__(self, name, col):
#         self.name = name
#         self.col = col

#     def perform_checks(self):
#         boxcox_transformation(self.name, self.col)
#         shapiro_and_anderson(self.name, self.col)
```

```
class Normality_check:
    def __init__(self, name, col):
        self.name = name
        self.col = col

    def shapiro_and_anderson(self):
        print(f"Performing SHAPIRO & ANDERSON-DARLING TEST for {self.name} column
        print()

        # Shapiro-Wilk Test
        print('Shapiro-Wilk Test')
        shapiro_stat, p_val = shapiro(self.col)
        if p_val < 0.05:
            print(f'{self.name} - Data is not Gaussian')
        else:
            print(f'{self.name} - Data is Gaussian')
        print()

        # Using Anderson-Darling Test
        print("Since Shapiro-Wilk test is sensitive, we go with Anderson-Darling
        result = anderson(self.col)
        if result.statistic > result.critical_values[2]:
            print(f"{self.name} - Data does not follow a normal distribution.")
        else:
            print(f"{self.name} - Data follows a normal distribution.")
        print()
        print('-'*50)

    def boxcox_transformation(self):
        print(f'Performing BOXCOX transformation on {self.name} column')
        transformed_data, best_lambda = boxcox(self.col)
        self.col = transformed_data  # Update column data with transformed data
        self.shapiro_and_anderson()  # Calling shapiro_and_anderson method after


# normality_check = NormalCheck(name, col)
# normality_check.boxcox_transformation()


def levene_test(name1,name2,col1,col2):

    levene_stat, p_value = levene(col1,col2)

    print(f'Performing Levene Test for {name1} & {name2}')

    if p_value < 0.05:
        print('Does not have Homogenous (different) Variance')
    else:
        print('Have Homogenous (similar) variance')
    print()
    print('-'*50)
    print()
    return ""
```

```
## MannWhitney u Rank test
### Test statistics : Mann-Whitney U rank test for two independent samples

def mannwhitneyu_test(name1,name2,col1,col2):

    print(f'Performing Non-parametric Test - MannWhitneyU for {name1} & {name2}')
    test_stat, p_value = mannwhitneyu(col1,col2)

    if p_value < 0.05:
        print("Reject Null Hypothesis")
        print(f'There is a significant difference in the Mean values of {name1} a
    else:
        print("Failed to Reject Null Hypothesis - Accept Ho")
        print(f'There is NO significant difference in the Mean values of {name1}

    print()
    print('-'*50)
    print()

    return ""
```

```
def normality_plots(name1,name2,name3,name4,col1,col2,col3,col4):

    plt.figure(figsize = (20,10))
    plt.suptitle("Normality check - Histplot & QQ(prob)plot",fontsize=16,fontweig

    plt.subplot(241)
    sns.histplot(col1, element = 'step', color =cp[1], kde = True, label = name1)
    plt.title(f'Histplot - {name1}',fontsize=10,fontweight="bold",backgroundcolor
    plt.legend()

    plt.subplot(242)
    sns.histplot(col3,element = 'step', color = cp[2], kde = True, label = name3
    plt.title(f'Histplot - {name3}',fontsize=10,fontweight="bold",backgroundcolor
    plt.legend()

    plt.subplot(243)
    probplot(col1, plot = plt, dist = 'norm')
    plt.title(f'Probplot - {name1}',fontsize=10,fontweight="bold",backgroundcolor

    plt.subplot(244)
    probplot(col3, plot = plt, dist = 'norm')
    plt.title(f'Probplot - {name3}',fontsize=10,fontweight="bold",backgroundcolor

    plt.subplot(245)
    sns.histplot(col2, element = 'step', color =cp[1], kde = True, label = name2
    plt.title(f'Histplot - {name2}',fontsize=10,fontweight="bold",backgroundcolor
    plt.legend()

    plt.subplot(246)
    sns.histplot(col4,element = 'step', color = cp[2], kde = True, label = name4)
    plt.title(f'Histplot - {name4}',fontsize=10,fontweight="bold",backgroundcolor
    plt.legend()

    plt.subplot(247)
    probplot(col2, plot = plt, dist = 'norm')
    plt.title(f'Probplot - {name2}',fontsize=10,fontweight="bold",backgroundcolor

    plt.subplot(248)
    probplot(col4, plot = plt, dist = 'norm')
    plt.title(f'Probplot - {name4}',fontsize=10,fontweight="bold",backgroundcolor

    sns.despine()
    plt.show()
```

⚡ Hypothesis testing - actual_time aggregated value and OSRM time aggregated value.

```
clipped_num_df.sample(3)
```

	od_time_diff_hour	start_scan_to_end_scan	actual_distance_to_destinatio
2762	8.076220	483.0	106.23396
2365	2.338220	140.0	24.86736
2424	1.762951	105.0	14.41372

```
clipped_num_df[['actual_time','osrm_time']].describe().T
```

	count	mean	std	min	25%	50%	75%	max
actual_time	14787.0	224.212577	185.922840	9.0	67.0	148.0	367.0	543.285302
osrm_time	14787.0	128.190912	150.301267	6.0	29.0	60.0	168.0	543.285302

```
filtered_num_df[['actual_time','osrm_time']].describe().T
```

	count	mean	std	min	25%	50%	75%	max
actual_time	14787.0	356.306000	561.517944	9.0	67.0	148.0	367.0	6265.0
osrm_time	14787.0	160.990936	271.459503	6.0	29.0	60.0	168.0	2032.0

```
actual_time = clipped_num_df['actual_time']
osrm_time = clipped_num_df['osrm_time']
fil_actual_time = filtered_num_df['actual_time']
fil_osrm_time = filtered_num_df['osrm_time']

normality_plots('clipped_actual_time','clipped_osrm_time','filtered_actual_time',
```

Normality check - Histplot & QQ(prob)plot