Importing relevant libraries -

```
import numpy as np
import pandas as pd
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import shapiro
from scipy.stats import levene
from scipy.stats import kruskal
from sklearn.preprocessing import MinMaxScaler
```

```
data = pd.read_csv("delhivery_data.csv")
```

```
data.head()
```

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

5 rows × 24 columns

```
data.shape
```

```
(144867, 24)
```

```
data.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
```

Handling missing values -

```
#Checking for missing values -
data.isna().sum()
```

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
is_cutoff	0
cutoff_factor	0
cutoff_timestamp	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
factor	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0
segment_factor	0

dtype: int64

```
# Replacing the missing values of the source and destination names with their respective center ids -
data["source_name"] = data["source_name"].fillna(data["source_center"])
data["destination_name"] = data["destination_name"].fillna(data["destination_center"])
```

```
#Confirming for missing values replacement-
data.isna().sum().sum()
```

```
0
```

All missing values handled.

```
#Conversion of date time columns from object type to datetime format -
data['trip_creation_time']=pd.to_datetime(data['trip_creation_time'])
data['od_start_time']=pd.to_datetime(data['od_start_time'])
data['od_end_time']=pd.to_datetime(data['od_end_time'])
```

```
data.describe()
```

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	
count	144867	144867	144867	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	2
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	0
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	1
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	1
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	2
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	77
std	NaN	NaN	NaN	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	1

```
data.nunique()
```

	0
data	2
trip_creation_time	14817
route_schedule_uuid	1504
route_type	2
trip_uuid	14817
source_center	1508
source_name	1508
destination_center	1481
destination_name	1481
od_start_time	26369
od_end_time	26369
start_scan_to_end_scan	1915
is_cutoff	2
cutoff_factor	501
cutoff_timestamp	93180
actual_distance_to_destination	144515
actual_time	3182
osrm_time	1531
osrm_distance	138046
factor	45641
segment_actual_time	747
segment_osrm_time	214
segment_osrm_distance	113799
segment_factor	5675

dtype: int64

Merging of rows -

```
#Merging the rows by grouping at trip id,source and destination levels,individual packages and their segments are aggregated with the appropriate aggregation function.
data_merge=data.groupby(['trip_uuid','source_center','destination_center']).agg({'data':'first',
                                                                              'trip_creation_time':'first',
                                                                             'route_schedule_uuid':'first',
                                                                              'route_type':'first',
                                                                              'source_name':'first',
                                                                              'destination_name':'last',
                                                                              'od_start_time':'first',
                                                                              'od_end_time':'last',
                                                                              'start_scan_to_end_scan':'max',
                                                                              'actual_distance_to_destination':'max',
                                                                              'actual_time':'max',
                                                                              'osrm_time':'max',
                                                                              'osrm_distance':'max',
                                                                              'segment_actual_time':'sum',
                                                                              'segment_osrm_time':'sum',
                                                                              'segment_osrm_distance':'sum'}).reset_index()
data_merge
# In the dataset actual and osrm times are cumulative values, hence using max agg func, whereas the segment times are not cumulative, hence using sum.
```

	trip_uuid	source_center	destination_center	data	trip_creation_time	route_schedule_uuid	route_type	source_name	destination_name	od_start_tim
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bilaspur_HB (Haryana)	2018-09-12 16:39:46.85846
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 00:00:16.53574
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	Doddablpur_ChikaDPP_D (Karnataka)	Chikblapur_ShntiSgr_D (Karnataka)	2018-09-12 02:03:09.65559
3	trip-153671042288605164	IND572101AAA	IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	Tumkur_Veersagr_I (Karnataka)	Doddablpur_ChikaDPP_D (Karnataka)	2018-09-12 00:00:22.88643
4	trip-153671043369099517	IND000000ACB	IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	Gurgaon_Bilaspur_HB (Haryana)	Chandigarh_Mehmdpur_H (Punjab)	2018-09-14 03:40:17.10673
...	...	...	...	...	...	...	...	...	...	...
26363	trip-153861115439069069	IND628204AAA	IND627657AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	Tirchchndr_Shnmgprm_D (Tamil Nadu)	Thisayanvilai_UdnkdiRD_D (Tamil Nadu)	2018-10-04 02:29:04.27219
26364	trip-153861115439069069	IND628613AAA	IND627005AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	Peikulam_SriVnktpm_D (Tamil Nadu)	Tirunelveli_VdkkuSrt_I (Tamil Nadu)	2018-10-04 04:16:39.89487
26365	trip-153861115439069069	IND628801AAA	IND628204AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	Eral_Busstand_D (Tamil Nadu)	Tirchchndr_Shnmgprm_D (Tamil Nadu)	2018-10-04 01:44:53.80800
26366	trip-153861118270144424	IND583119AAA	IND583101AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	Sandur_WrdN1DPP_D (Karnataka)	Bellary_Dc (Karnataka)	2018-10-04 03:58:40.72654
26367	trip-153861118270144424	IND583201AAA	IND583119AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	Hospet (Karnataka)	Sandur_WrdN1DPP_D (Karnataka)	2018-10-04 02:51:44.71265

26368 rows × 19 columns

```
df = data_merge.groupby("trip_uuid").agg({"data": "first", "trip_creation_time": "first",
                                          "route_schedule_uuid": "first", "route_type": "first",
                                          "source_center": "first", "source_name": "first",
                                          "destination_center": "last", "destination_name": "last",
                                          "od_start_time": "first", "od_end_time": "last",
                                          "start_scan_to_end_scan":"sum", "actual_distance_to_destination": "sum",
                                          "actual_time": "sum", "osrm_time": "sum", "osrm_distance": "sum",
                                          "segment_actual_time": "sum", "segment_osrm_time": "sum", "segment_osrm_distance": "sum"}).reset_index()
df
```

	trip_uuid	data	trip_creation_time	route_schedule_uuid	route_type	source_center	source_name	destination_center	destination_name	od_start_time
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 16:39:46.858469
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	2018-09-12 02:03:09.655591
2	trip-153671043369099517	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-14 03:40:17.106733
3	trip-153671046011330457	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	IND400072AAB	Mumbai Hub (Maharashtra)	IND401104AAA	Mumbai_MiraRd_IP (Maharashtra)	2018-09-12 00:01:00.113710
4	trip-153671052974046625	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	IND583101AAA	Bellary_Dc (Karnataka)	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka)	2018-09-12 00:02:09.740725
...	...	...	...	...	...	...	...	...	...	...
14812	trip-153861095625827784	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab)	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab)	2018-10-03 23:55:56.258533
14813	trip-153861104386292051	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	IND121004AAB	FBD_Balabhgarh_DPC (Haryana)	IND121004AAA	Faridabad_Blbgarh_DC (Haryana)	2018-10-03 23:57:23.863155
14814	trip-153861106442901555	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	IND208006AAA	Kanpur_GovndNgr_DC (Uttar Pradesh)	IND208006AAA	Kanpur_GovndNgr_DC (Uttar Pradesh)	2018-10-04 02:51:27.075797
14815	trip-153861115439069069	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	IND627005AAA	Tirunelveli_VdkkuSrt_I (Tamil Nadu)	IND628204AAA	Tirchchndr_Shnmgprm_D (Tamil Nadu)	2018-10-03 23:59:14.390954
14816	trip-153861118270144424	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka)	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka)	2018-10-04 03:58:40.726547

14817 rows × 19 columns

Feature build for data analysis -

```
df[["source", "source_state"]] = df["source_name"].str.split("(", n=1, expand = True)
df["source_state"] = df["source_state"].str.strip(")")
df[["source_city", "source_place", "source_code"]]= df["source"].str.split("_", n=2, expand = True)
df[["dest", "dest_state"]] = df["destination_name"].str.split("(", n=1, expand = True)
df["dest_state"] = df["dest_state"].str.strip(")")
df[["dest_city", "dest_place", "dest_code"]]= df["dest"].str.split("_", n=2, expand = True)
```

```
df["trip_creation_year"] = df["trip_creation_time"].dt.year
df["trip_creation_month"] = df["trip_creation_time"].dt.month_name()
df["trip_creation_day"] = df["trip_creation_time"].dt.day
```

```
df.head()
```

	trip_uuid	data	trip_creation_time	route_schedule_uuid	route_type	source_center	source_name	destination_center	destination_name	od_start_time	...
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 16:39:46.858469	...
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	2018-09-12 02:03:09.655591	...
2	trip-153671043369099517	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-14 03:40:17.106733	...
3	trip-153671046011330457	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	IND400072AAB	Mumbai Hub (Maharashtra)	IND401104AAA	Mumbai_MiraRd_IP (Maharashtra)	2018-09-12 00:01:00.113710	...
4	trip-153671052974046625	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	IND583101AAA	Bellary_Dc (Karnataka)	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka)	2018-09-12 00:02:09.740725	...

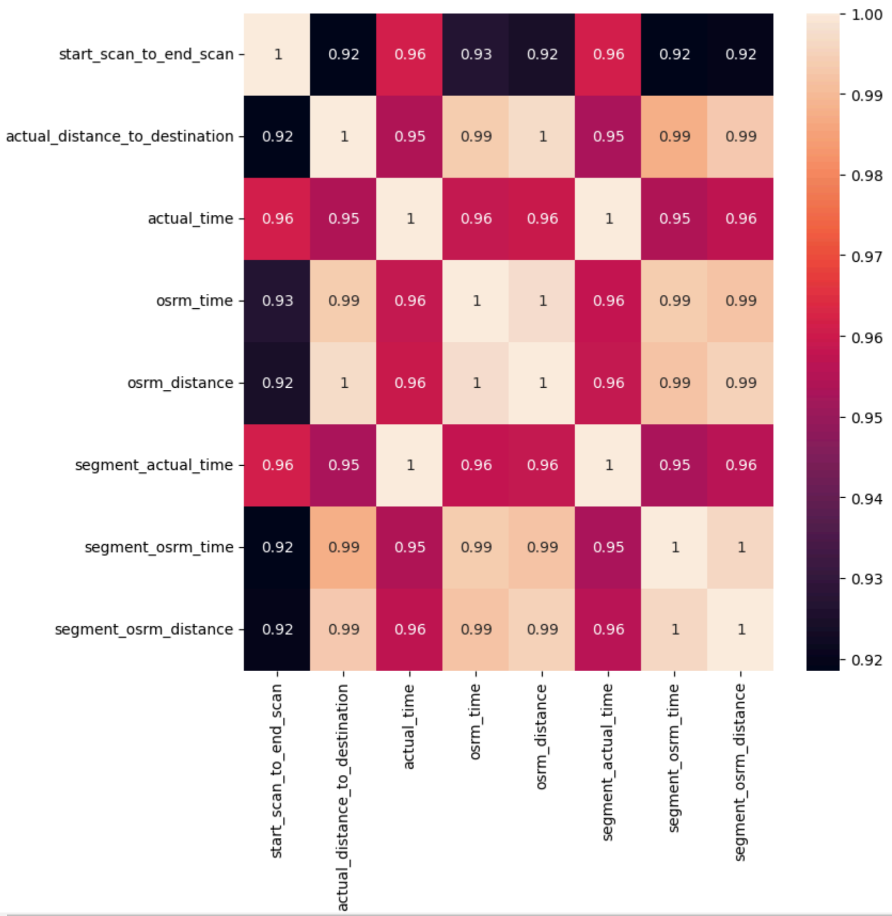
5 rows × 32 columns

Data Visualization:

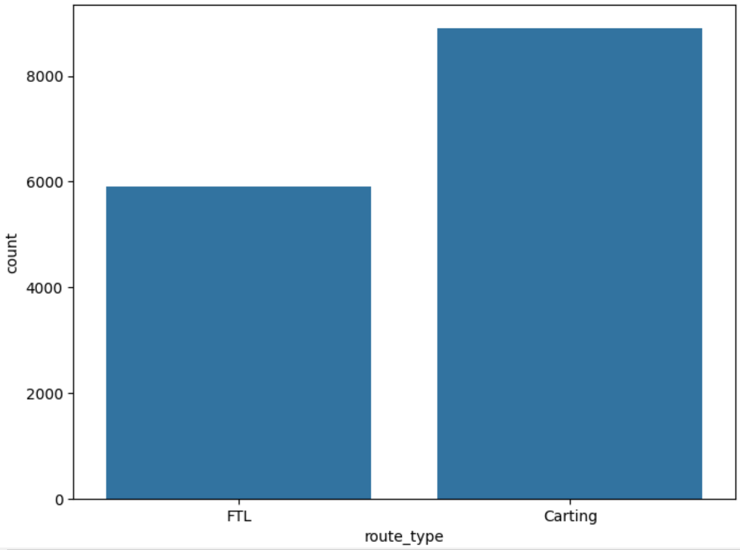
```
num_col = df.select_dtypes(include = ["int", "float64"]).columns
num_col.drop(["trip_creation_year",
       "trip_creation_day"])
```

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

```
df_corr = df[num_col.drop(["trip_creation_year",
       "trip_creation_day"])].corr()
plt.figure(figsize = (8, 8))
sns.heatmap(df_corr, annot = True)
plt.show()
```



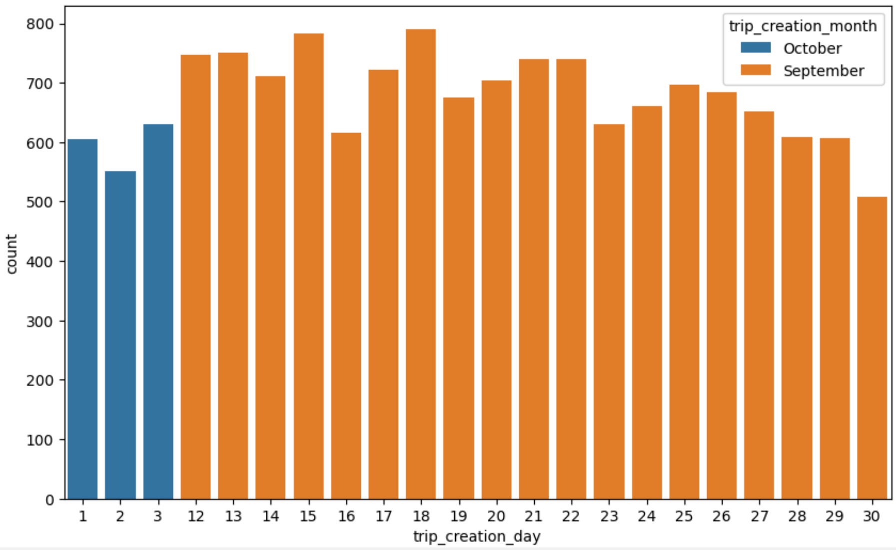

```
#Most preferred route_type  :

plt.figure(figsize = (8,6))
bar = sns.countplot(x=df["route_type"])
plt.show()
```



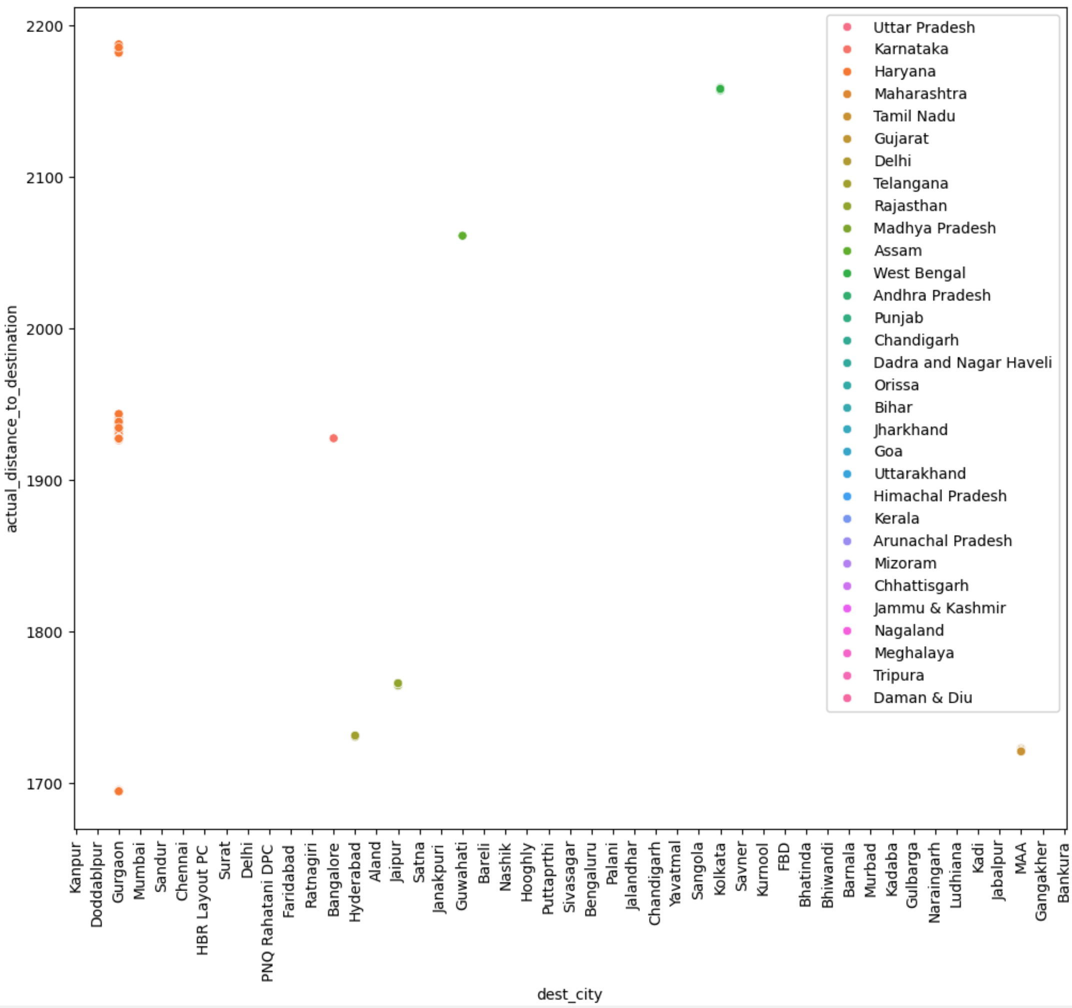
Inference:

- Most of the shipments use "carting" as the route type.
- Count of carting is close to 9k and that of FTL is around 6k (i.e 2/3rd).

```
#Trip creation day and month -
plt.figure(figsize = (10,6))
sns.countplot(x = df["trip_creation_day"], hue = df["trip_creation_month"])
plt.show()
```




```
# Actual distance to destination vs city

top_city = df["actual_distance_to_destination"].sort_values(ascending = False).head(100)
plt.figure(figsize = (12, 10))
plt.xticks(rotation = 90)
sns.scatterplot(x=df["dest_city"], y = top_city, hue= df["dest_state"])
plt.legend(loc = "upper right")
plt.show()
```

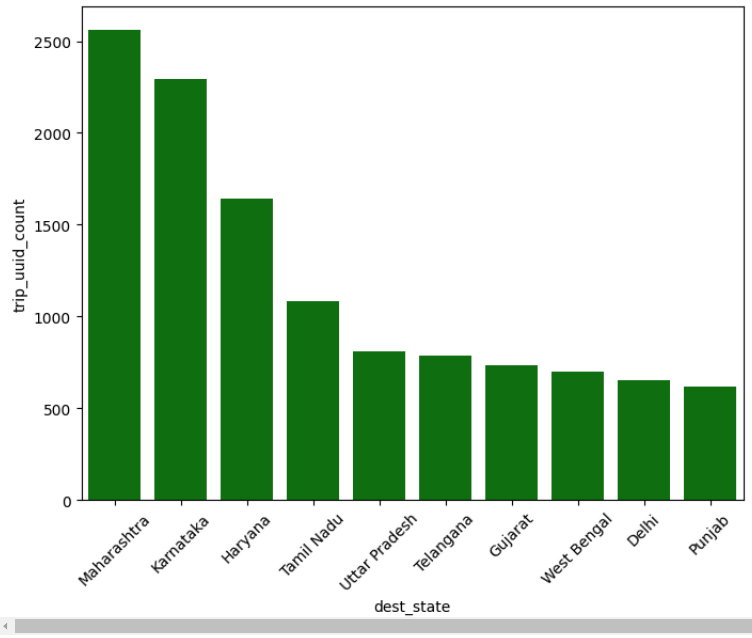


Gurgaon, Hariyana is the destination city to which "distance to reach destination" is the highest compared to other cities in the states.

```
# Destination states with delivery counts -
states_dest = df.groupby("dest_state")["trip_uuid"].count().sort_values(ascending = False).head(10).to_frame("trip_uuid_count").reset_index()
states_dest
```

	dest_state	trip_uuid_count
0	Maharashtra	2561
1	Karnataka	2294
2	Haryana	1643
3	Tamil Nadu	1084
4	Uttar Pradesh	811
5	Telangana	784
6	Gujarat	734
7	West Bengal	697
8	Delhi	652
9	Punjab	617

```
plt.figure(figsize = (8,6))
sns.barplot(x="dest_state", y = "trip_uuid_count", data = states_dest, color = "g")
plt.xticks(rotation = 45)
plt.show()
```



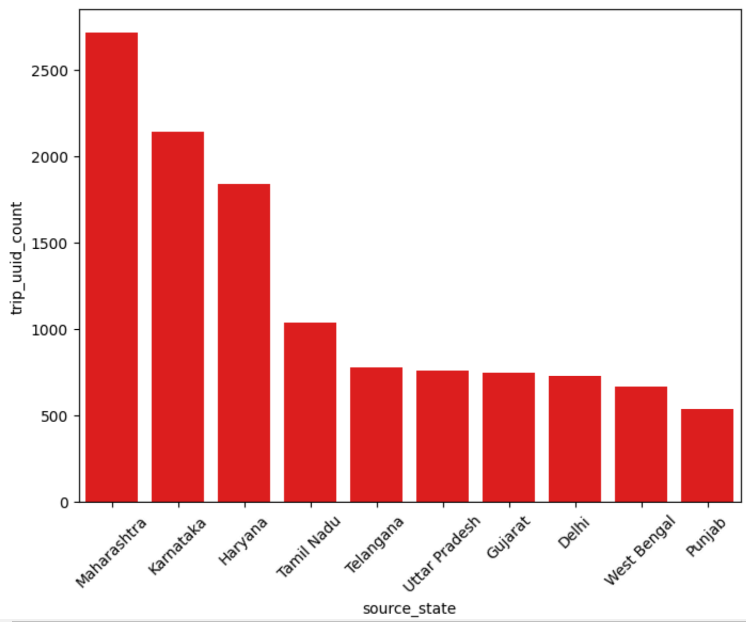
Inference:

- Highest deliveries are occuring in Maharashtra (2561) and then followed by Karnataka (2294) in a close second.
- Haryana, TN and UP are also in the top 5 states.

```
# Source states with delivery counts -
states_source = df.groupby("source_state")["trip_uuid"].count().sort_values(ascending = False).head(10).to_frame("trip_uuid_count").reset_index()
states_source
```

	source_state	trip_uuid_count
0	Maharashtra	2714
1	Karnataka	2143
2	Haryana	1838
3	Tamil Nadu	1039
4	Telangana	781
5	Uttar Pradesh	762
6	Gujarat	750
7	Delhi	728
8	West Bengal	665
9	Punjab	536

```
plt.figure(figsize = (8,6))
sns.barplot(x="source_state", y = "trip_uuid_count", data = states_source, color = "r")
plt.xticks(rotation = 45)
plt.show()
```



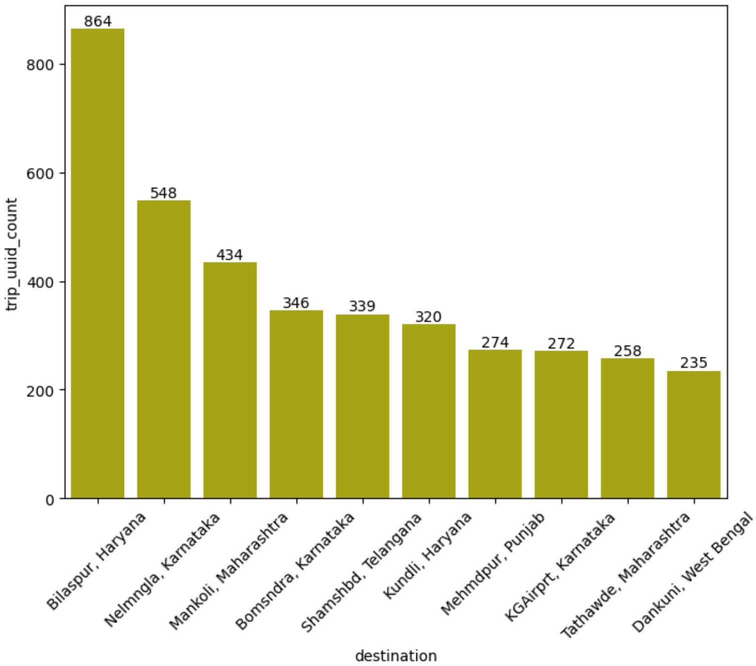
- In case of source states for delivery, trend is similar to the destination delivery count, with Maharashtra (2714) at the top and Karnataka (2143) at second position.
- Haryana, TN and Telangana are in the top 5 states.

```
# Analysis of destination places in states with highest delivery:
```

```
d = df.groupby(["dest_place", "dest_state"])["trip_uuid"].count().sort_values(ascending = False).head(10).to_frame("trip_uuid_count").reset_index()
d
```

	dest_place	dest_state	trip_uuid_count
0	Bilaspur	Haryana	864
1	Nelmngla	Karnataka	548
2	Mankoli	Maharashtra	434
3	Bomsndra	Karnataka	346
4	Shamshbd	Telangana	339
5	Kundli	Haryana	320
6	Mehmdpur	Punjab	274
7	KGAirprt	Karnataka	272
8	Tathawde	Maharashtra	258
9	Dankuni	West Bengal	235

```
plt.figure(figsize = (8,6))
d["destination"] = d["dest_place"] + ", " + d["dest_state"]
ax = sns.barplot(x="destination", y = "trip_uuid_count", data = d, color = "y")
for i in ax.patches:
    ax.annotate(format(i.get_height(), '.0f'), (i.get_x() + i.get_width() / 2., i.get_height()),  ha = 'center', va = 'center', xytext = (0, 5), textcoords = 'offset points')
plt.xticks(rotation = 45)
plt.show()
```



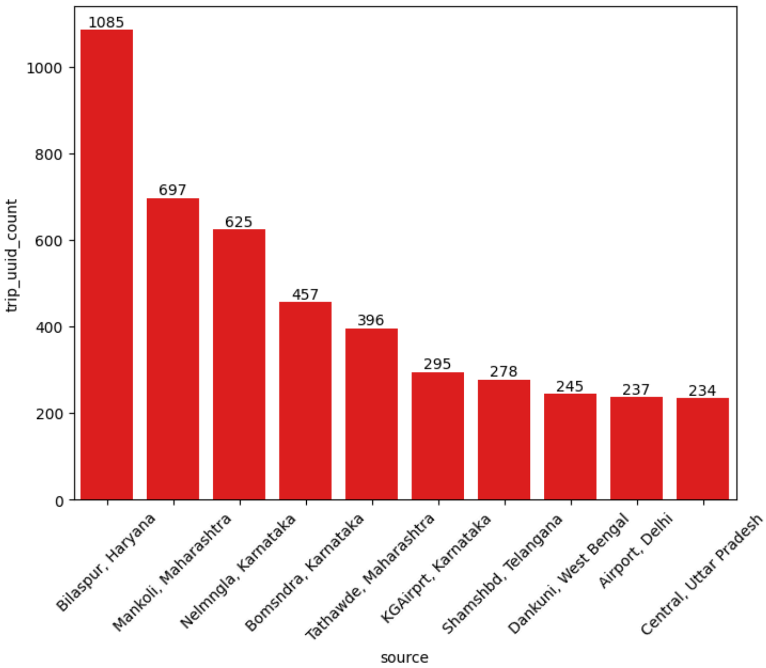
Inference:

- Bilaspur in Haryana is the most delivered place in that state. Nelamangala in Karnataka is at second.
- In the top 10, there are 3 places in the state Karnatake which to which most deliveries are done.

```
# Analysis of source places in states with highest delivery:
s = df.groupby(["source_place", "source_state"])["trip_uuid"].count().sort_values(ascending = False).head(10).to_frame("trip_uuid_count").reset_index()
s
```

	source_place	source_state	trip_uuid_count
0	Bilaspur	Haryana	1085
1	Mankoli	Maharashtra	697
2	Nelmngla	Karnataka	625
3	Bomsndra	Karnataka	457
4	Tathawde	Maharashtra	396
5	KGAirprt	Karnataka	295
6	Shamshbd	Telangana	278
7	Dankuni	West Bengal	245
8	Airport	Delhi	237
9	Central	Uttar Pradesh	234

```
plt.figure(figsize = (8,6))
s["source"] = s["source_place"] + ", " + s["source_state"]
ax = sns.barplot(x="source", y = "trip_uuid_count", data = s, color = "r")
for i in ax.patches:
    ax.annotate(format(i.get_height(), '.0f'), (i.get_x() + i.get_width() / 2., i.get_height()),  ha = 'center', va = 'center', xytext = (0, 5), textcoords = 'offset points')
plt.xticks(rotation = 45)
plt.show()
```



Inference:

- Bilaspur in Haryana is the most sourced place in that state. Mankoli in Maharashtra is at second.
- There are 3 places in Karnataka and 2 in Maharashtra in the top 10 source places.

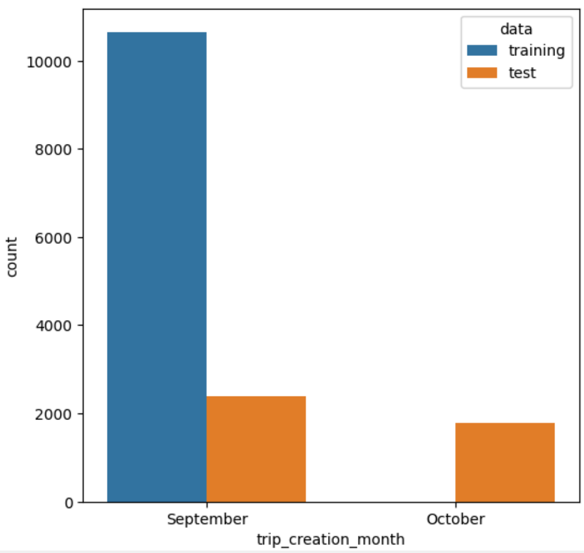
```
# Analysis of destination cities in states with highest delivery:
d = df.groupby(["dest_city", "dest_state"])["trip_uuid"].count().sort_values(ascending = False).head(10).to_frame("trip_uuid_count").reset_index()
d
```

	dest_city	dest_state	trip_uuid_count
0	Bengaluru	Karnataka	1088
1	Mumbai	Maharashtra	966
2	Gurgaon	Haryana	877
3	Bangalore	Karnataka	551
4	Delhi	Delhi	549
5	Hyderabad	Telangana	499
6	Bhiwandi	Maharashtra	434
7	Chennai	Tamil Nadu	410
8	Sonipat	Haryana	322
9	Pune	Maharashtra	313

Inference -

- Bangalore/ Bengaluru in karnataka has the highest deliveries in the country.
- Top 5 cities also include Mumbai, Delhi, Gurgaon, Hyderabad.

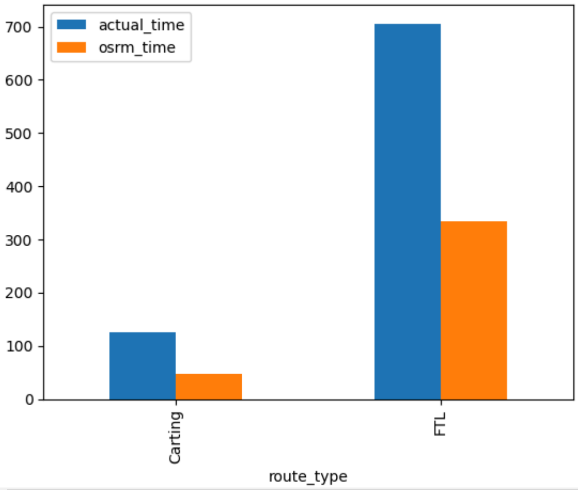
```
#Month with highest trip creation:
plt.figure(figsize = (6,6))
sns.countplot(x=df["trip_creation_month"], hue = df["data"])
plt.show()
```



Inference -

- Majority of trip creation has occured in September.
- And most of the trip creation in september is part of training.

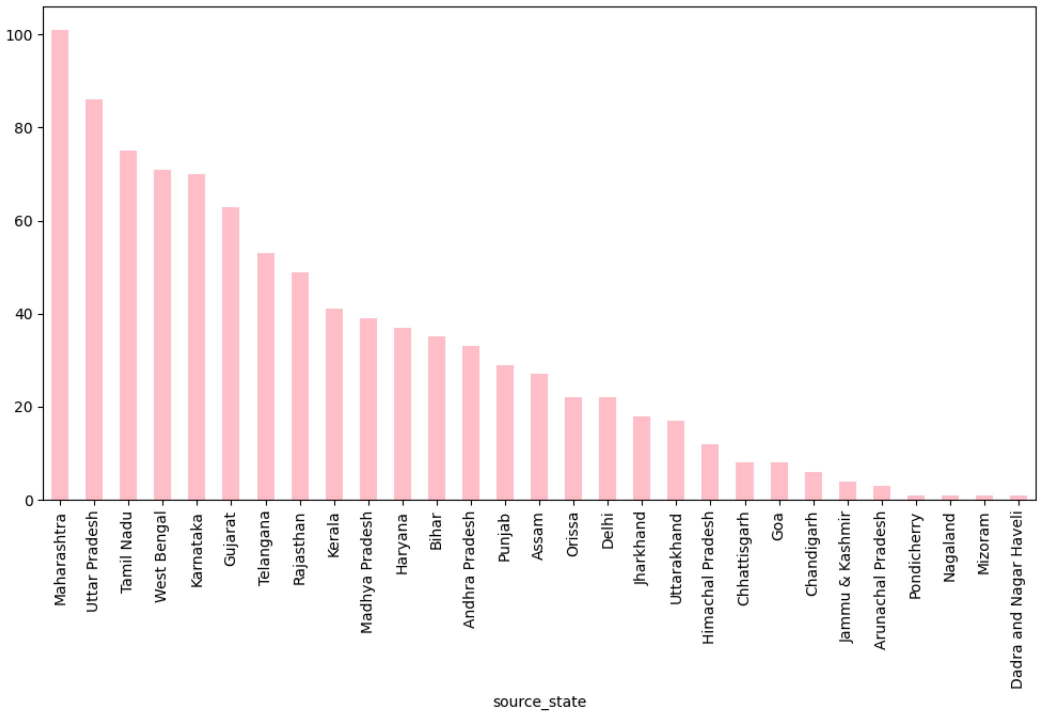
```
# Route type that is more time efficient:

df.groupby("route_type").aggregate({"actual_time" : "mean", "osrm_time": "mean"}).plot(kind = "bar")
plt.show()
```



Inference:

- Full truck load, that is FTL takes more time to deliver than small vehicles(carts).
- Actual mean time of carting is lesser by 6 times whereas osrm mean time of carting is lesser by 3 times compared to FTL.

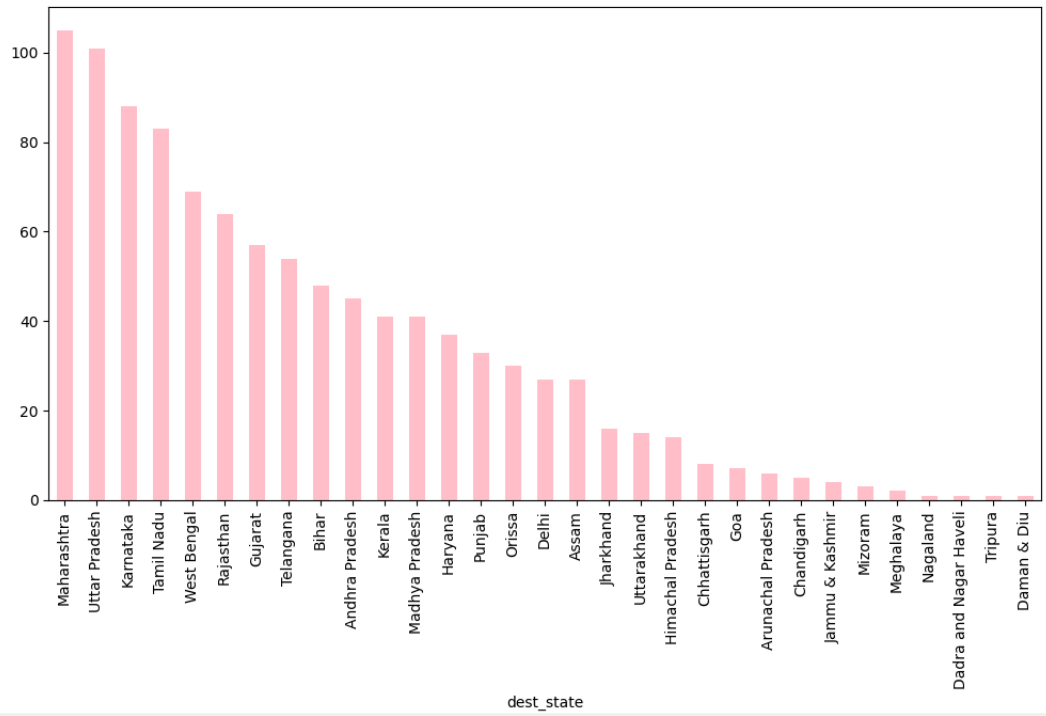
```
# States and the number of source centers -
plt.figure(figsize = (12,6))
df.groupby("source_state")["source_center"].nunique().sort_values(ascending = False).plot(kind = "bar", color = "pink")
plt.show()
```



Inference:

- Maharashtra has highest number of source centers followed by UP and TN.
- WB and Karnataka are also in the top 5 states.

```
# States and the number of destination centers -
plt.figure(figsize = (12,6))
df.groupby("dest_state")["destination_center"].nunique().sort_values(ascending = False).plot(kind = "bar", color = "pink")
plt.show()
```



Inference:

- Maharashtra has highest number of source centers followed by UP and Karnataka.
- WB and TN are also in the top 5 states.
- The same 5 states are in the top 5 list of most source centers and destination centers.

```
# Day with most deliveries:

df["weekday"] = df["trip_creation_time"].dt.day_name()
df.head()
```

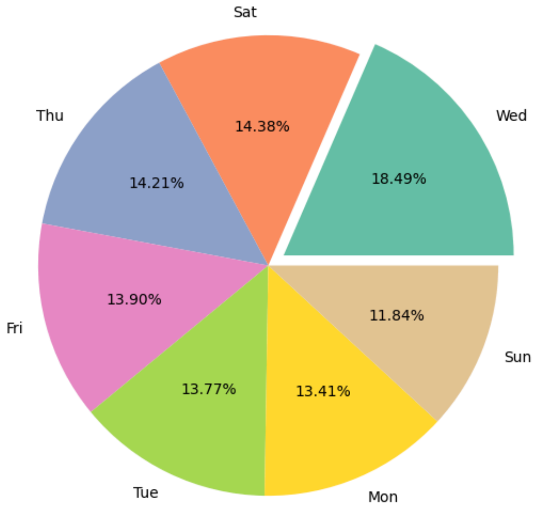
	trip_uuid	data	trip_creation_time	route_schedule_uuid	route_type	source_center	source_name	destination_center	destination_name	od_start_time	...
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 16:39:46.858469	...
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	2018-09-12 02:03:09.655591	...
2	trip-153671043369099517	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-14 03:40:17.106733	...
3	trip-153671046011330457	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	IND400072AAB	Mumbai Hub (Maharashtra)	IND401104AAA	Mumbai_MiraRd_IP (Maharashtra)	2018-09-12 00:01:00.113710	...
4	trip-153671052974046625	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	IND583101AAA	Bellary_Dc (Karnataka)	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka)	2018-09-12 00:02:09.740725	...

5 rows × 33 columns

```
day_data = df["weekday"].value_counts().to_frame("count").reset_index()
day_data
```

	weekday	count
0	Wednesday	2739
1	Saturday	2130
2	Thursday	2106
3	Friday	2060
4	Tuesday	2040
5	Monday	1987
6	Sunday	1755

```
plt.figure(figsize = (10,7))
palette_color = sns.color_palette("Set2")
plt.pie(data = day_data, x = day_data["count"], colors = palette_color, labels = ["Wed", "Sat", "Thu", "Fri", "Tue", "Mon", "Sun"], explode = (0.08,0,0,0,0,0,0), autopct = "%0.2f%%")
plt.show()
```



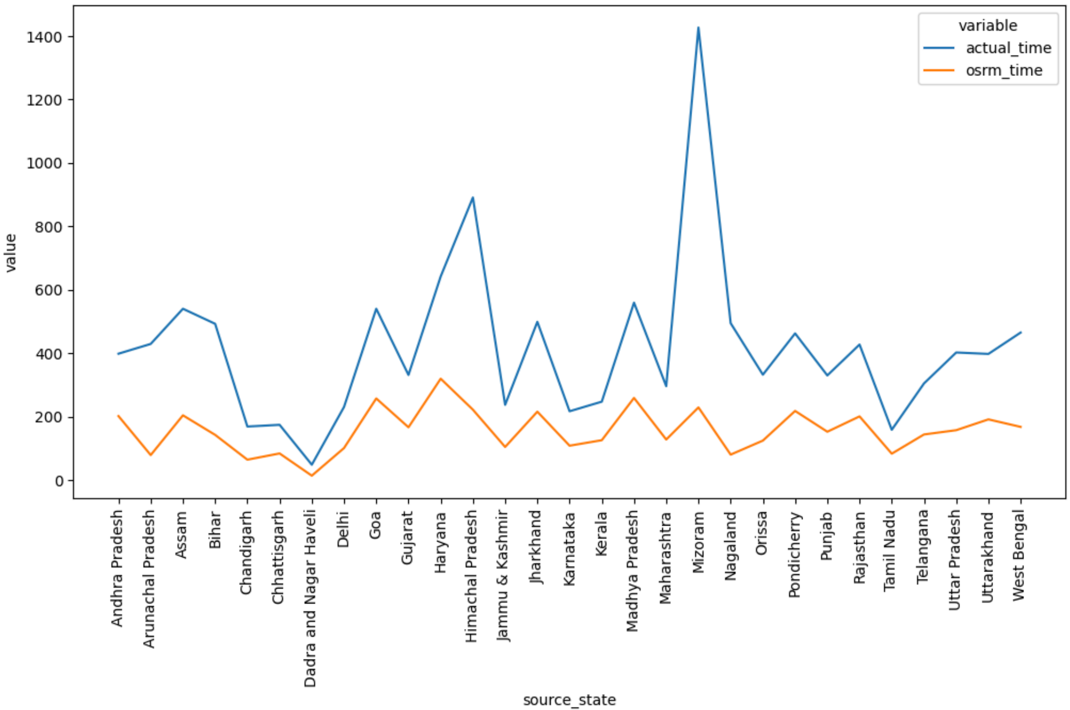
Inference:

- Highest deliveries happen on Wednesday (18.5%).
- Followed by Saturday (14.38%) and Thursday (14.2%).
- Least deliveries happen on Monday and Sunday.

```
#Analyzing actual time vs osrm time for each state:

t = df.groupby("source_state").aggregate({"actual_time": "mean", "osrm_time": "mean"}).reset_index()
t = pd.melt(t, id_vars = ["source_state"], value_vars = ["actual_time", "osrm_time"])
t.head()
```

	source_state	variable	value
0	Andhra Pradesh	actual_time	398.435484
1	Arunachal Pradesh	actual_time	429.250000
2	Assam	actual_time	540.171642
3	Bihar	actual_time	492.645714
4	Chandigarh	actual_time	168.741935

```
plt.figure(figsize = (12, 6))
sns.lineplot(y = "value", x = "source_state", data = t, hue = "variable")
plt.xticks(rotation = 90)
plt.show()
```



Inference:

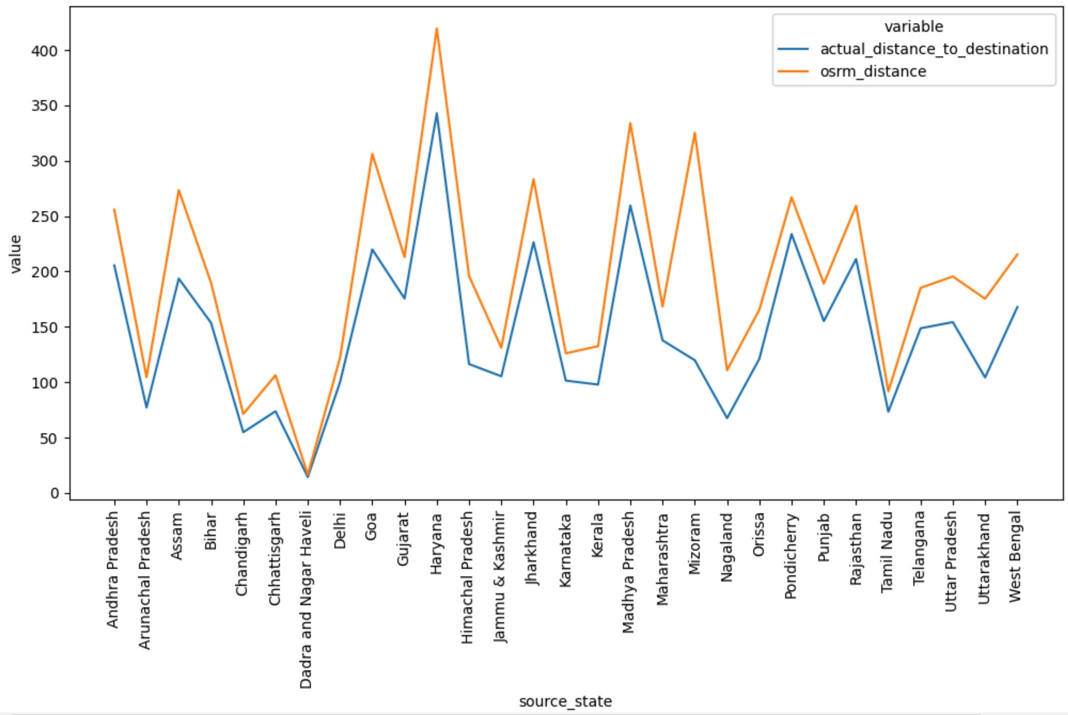
- In states/ UT Dadra & Nagar Haveli the difference between actual mean time and osrm mean time is almost nill.
- The difference is minimal in TN as well.
- The variation is highest in case of Mizoram and followed by Haryana.

```
#Analyzing actual distance vs osrm distance for each state:

d = df.groupby("source_state").aggregate({"actual_distance_to_destination": "mean", "osrm_distance": "mean"}).reset_index()
d = pd.melt(d, id_vars = ["source_state"], value_vars = ["actual_distance_to_destination", "osrm_distance"])
d
```

	source_state	variable	value
0	Andhra Pradesh	actual_distance_to_destination	205.461342
1	Arunachal Pradesh	actual_distance_to_destination	77.150532
2	Assam	actual_distance_to_destination	193.532247
3	Bihar	actual_distance_to_destination	153.588085
4	Chandigarh	actual_distance_to_destination	54.811432
5	Chhattisgarh	actual_distance_to_destination	73.734984
6	Dadra and Nagar Haveli	actual_distance_to_destination	14.349976
7	Delhi	actual_distance_to_destination	100.054953
8	Goa	actual_distance_to_destination	219.883526
9	Gujarat	actual_distance_to_destination	175.549300
10	Haryana	actual_distance_to_destination	342.895103
11	Himachal Pradesh	actual_distance_to_destination	116.370463
12	Jammu & Kashmir	actual_distance_to_destination	105.300217
13	Jharkhand	actual_distance_to_destination	226.406040
14	Karnataka	actual_distance_to_destination	101.412585
15	Kerala	actual_distance_to_destination	97.875658
16	Madhya Pradesh	actual_distance_to_destination	259.562780
17	Maharashtra	actual_distance_to_destination	137.783489
18	Mizoram	actual_distance_to_destination	119.774782
19	Nagaland	actual_distance_to_destination	67.510835
20	Orissa	actual_distance_to_destination	121.063622
21	Pondicherry	actual_distance_to_destination	233.749084
22	Punjab	actual_distance_to_destination	155.165304
23	Rajasthan	actual_distance_to_destination	211.217983
24	Tamil Nadu	actual_distance_to_destination	73.311371
25	Telangana	actual_distance_to_destination	148.657690
26	Uttar Pradesh	actual_distance_to_destination	154.254277
27	Uttarakhand	actual_distance_to_destination	104.240000
28	West Bengal	actual_distance_to_destination	167.877726
29	Andhra Pradesh	osrm_distance	255.804854
30	Arunachal Pradesh	osrm_distance	104.452450
31	Assam	osrm_distance	273.490164
32	Bihar	osrm_distance	189.872446
33	Chandigarh	osrm_distance	71.311419
34	Chhattisgarh	osrm_distance	106.256274
35	Dadra and Nagar Haveli	osrm_distance	16.592587
36	Delhi	osrm_distance	122.053160
37	Goa	osrm_distance	306.198908
38	Gujarat	osrm_distance	212.907671
39	Haryana	osrm_distance	419.345857
40	Himachal Pradesh	osrm_distance	195.963321
41	Jammu & Kashmir	osrm_distance	131.068324
42	Jharkhand	osrm_distance	283.488592
43	Karnataka	osrm_distance	126.073458
44	Kerala	osrm_distance	132.494697
45	Madhya Pradesh	osrm_distance	333.969259
46	Maharashtra	osrm_distance	168.249902
47	Mizoram	osrm_distance	325.281675
48	Nagaland	osrm_distance	110.817940
49	Orissa	osrm_distance	165.384751
50	Pondicherry	osrm_distance	266.931242
51	Punjab	osrm_distance	189.070016
52	Rajasthan	osrm_distance	259.339060
53	Tamil Nadu	osrm_distance	91.682308
54	Telangana	osrm_distance	185.170089
55	Uttar Pradesh	osrm_distance	195.463270
56	Uttarakhand	osrm_distance	175.364052
57	West Bengal	osrm_distance	215.371641

```
plt.figure(figsize = (12, 6))
sns.lineplot(y = "value", x = "source_state", data = d, hue = "variable")
```

```
plt.xticks(rotation = 90)
plt.show()
```



Inference:

- The differences in mean actual and osrm distances are overall lesser compared to the differences in times.
- Again for Dadra and Nagar Haveli, Delhi and TN the difference is almost negligible.
- The differences are highest in case of Mizoram, Haryana, Goa and Assam.

Outliers

```
#Percentage of outliers:
Q1 = df["actual_time"].quantile(0.25)
Q3 = df["actual_time"].quantile(0.75)
IQR = Q3-Q1
upper_bound = Q3 + 1.5*IQR
lower_bound = Q1 - 1.5*IQR
upper_outliers = df[df["actual_time"] > upper_bound]
round(len(upper_outliers)*100/len(df),2)
```

```
11.09
```

Inference -

- Nearly 11% of the data is outliers, hence clipping them is not right.
- Before performing hypothesis testing, we can try to transform the data using log normal function and see its impact.
- Outlier treatment will be done post that.

Feature Engineering:

```
# Creating a feature (column) which is the difference between od_end_time and od_start_time
df["total_min_diff"] = (df["od_end_time"] - df["od_start_time"])/pd.Timedelta(minutes = 1) # time difference between the two datetime columns for each row in the DataFrame and
                                                                                            # the result is a pandas Timedelta object.When you divide the Timedelta object (time difference)
                                                                                            #by pd.Timedelta(minutes=1), the result is the total time difference in minutes.
```

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

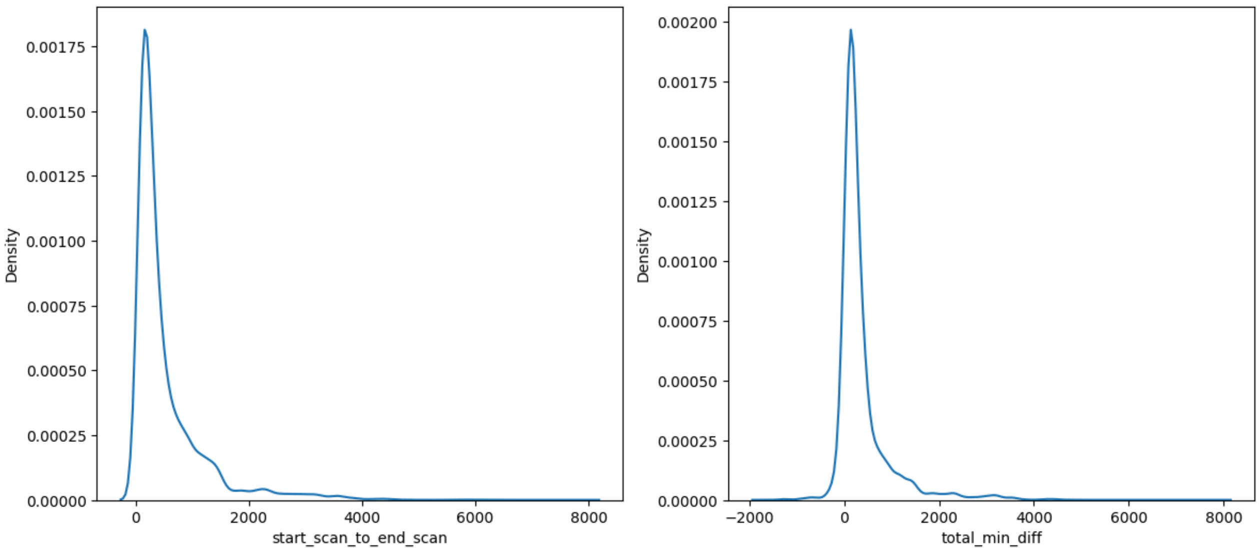
```
#Comparing the total min difference and the start scan to end scan and statistically check if they are similar -
df[["start_scan_to_end_scan", "total_min_diff"]]
```

	start_scan_to_end_scan	total_min_diff
0	2259.0	0.000000
1	180.0	0.000000
2	3933.0	0.000000
3	100.0	100.494935
4	717.0	232.556228
...	...	...
14812	257.0	405.485842
14813	60.0	60.590521
14814	421.0	0.000000
14815	347.0	149.831354
14816	353.0	0.000000

14817 rows × 2 columns

```
# Hypothesis testing:
#  H0 - mean(start_scan_to_end_scan) == mean(total_min_diff)
# Ha -  mean(start_scan_to_end_scan) != mean(total_min_diff)
```

```
#Checking for normality -
plt.figure(figsize = (14,6))
plt.subplot(1,2,1)
sns.kdeplot(df["start_scan_to_end_scan"])

plt.subplot(1,2,2)
sns.kdeplot(df["total_min_diff"])
plt.show()
```



Inference:

- It appears that both data are right skewed and that there are presence of outliers.
- Shapiro test cannot be done as it cannot handle large datasets.
- Parametric tests cannot be used to check the statistical significance.

```
# Non - parametric tests:
# Kruskal wallis test:

stat, p_val = kruskal(df["start_scan_to_end_scan"], df["total_min_diff"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

```
KW statistic is: 1111.6501037556222 and P value is: 9.700927751107437e-244
Reject H0: Medians are significantly different
```

Inference:

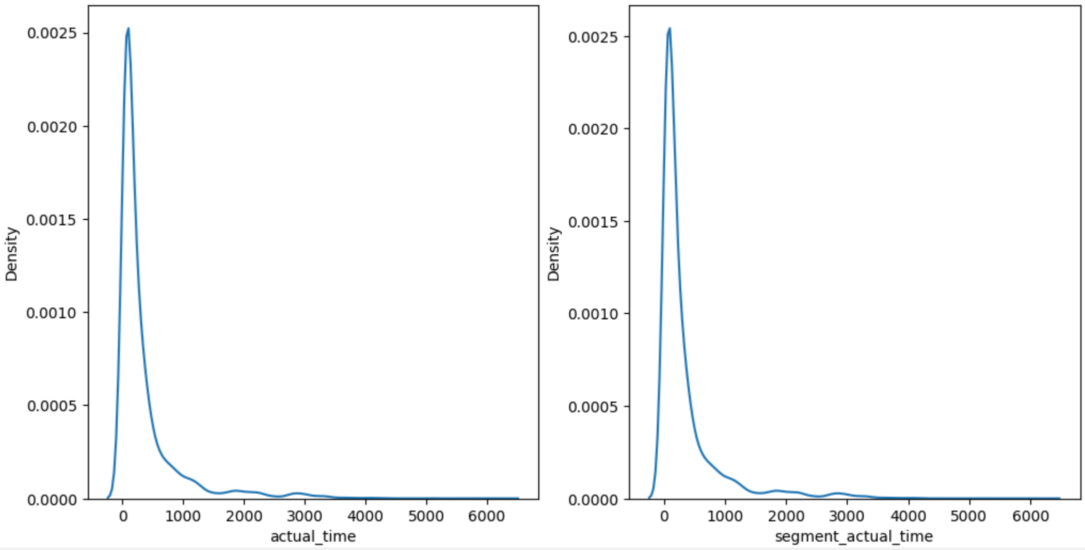
- From the hypothesis testing, it is confirmed that the means of the start_scan_to_end_scan and total_min_diff are significantly different.
- This difference indicates that there could be a data collection issue at source due to human or system error.
- Additionally, there could be some assumptions that were taken incorrectly. For example, total_min_diff is based on differences between trip and odometer times, whereas start_scan_to_end_scan might include delays or processes not accounted for in total_min_diff.

```
df[["trip_uuid", "actual_time", "segment_actual_time"]]
```

	trip_uuid	actual_time	segment_actual_time
0	trip-153671041653548748	1562.0	1548.0
1	trip-153671042288605164	143.0	141.0
2	trip-153671043369099517	3347.0	3308.0
3	trip-153671046011330457	59.0	59.0
4	trip-153671052974046625	341.0	340.0
...	...	...	...
14812	trip-153861095625827784	83.0	82.0
14813	trip-153861104386292051	21.0	21.0
14814	trip-153861106442901555	282.0	281.0
14815	trip-153861115439069069	264.0	258.0
14816	trip-153861118270144424	275.0	274.0

14817 rows × 3 columns

```
# Analysis between actual_time aggregated value and segment actual time aggregated value:

# checking normality by plotting the density plots-
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["actual_time"])
plt.subplot(1,2,2)
sns.kdeplot(df["segment_actual_time"])
plt.show()
```



Inference -

- Both are appearing to have a long tail, hence right skewed distribution.
- Shapiro test cannot be done as it cannot handle large datasets.
- Parametric tests cannot be used to check the statistical significance.
- Log normal function also couldnt make it completely normal.

```
# Non - parametric tests:
# Kruskal wallis test:

stat, p_val = kruskal(df["actual_time"], df["segment_actual_time"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

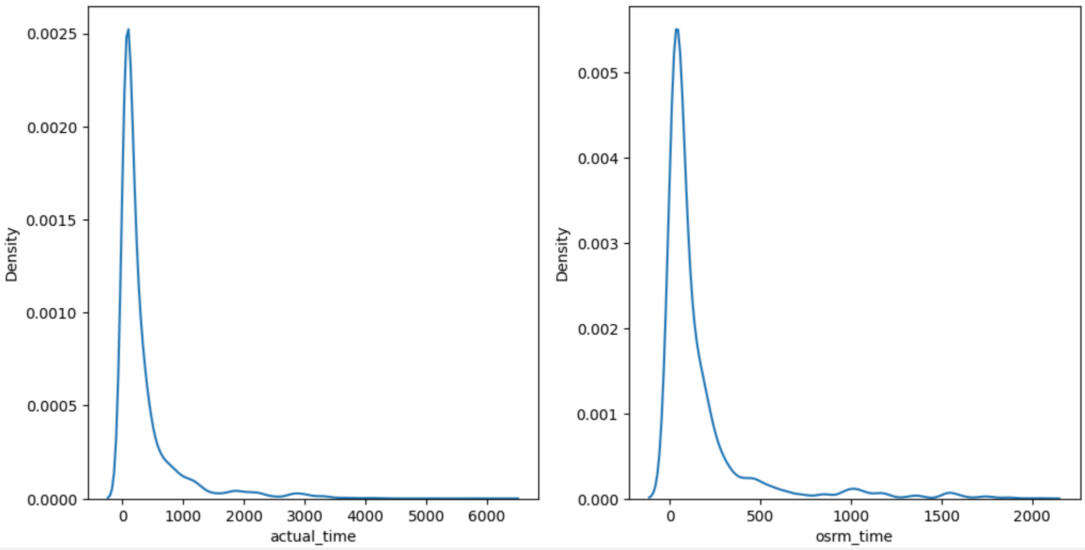
```
KW statistic is: 0.6603876360800134 and P value is: 0.4164231265256002
Accept H0: Medians are not significantly different
```

Inference -

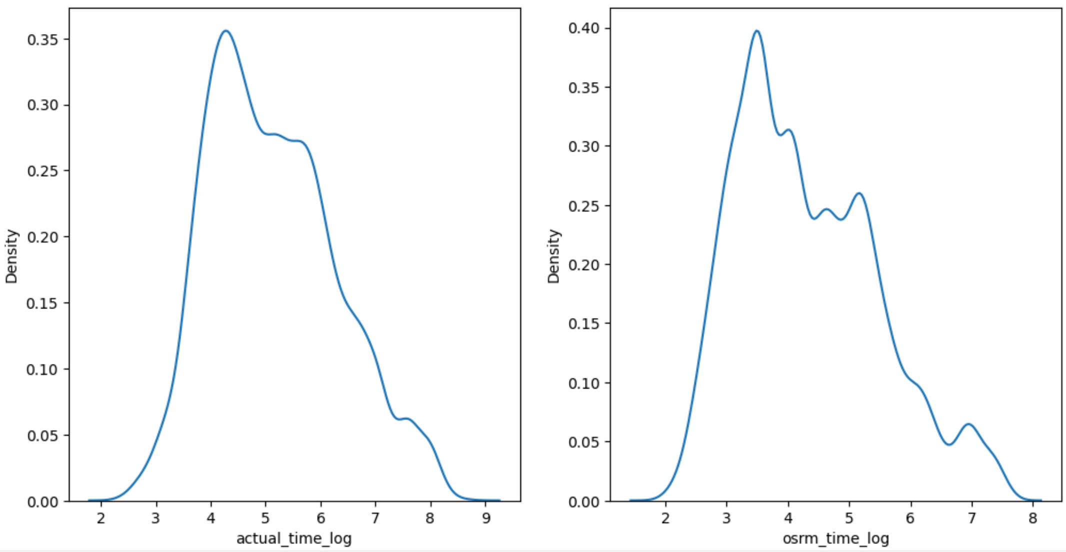
- The P values is not less than 5% and this means that Null hypothesis cannot be rejected.
- The actual time and segment actual time have means that are significantly similar.
- Since time and distance metrics (segment, osrm actual) are strongly correlated (0.96), and we have already tested the significance of the difference in actual time and segment time, the distance metrics are likely to follow a very similar pattern. In most cases, testing actual distance vs. segment distance would not yield additional insights because the high correlation implies that the behavior of the distances is already aligned with that of the times.

```
# Analysis between actual_time aggregated value and osrm time aggregated value:

# checking normality by plotting the density plots-
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["actual_time"])
plt.subplot(1,2,2)
```

```
sns.kdeplot(df["osrm_time"])
plt.show()
```




```
#Let us try converting data to log normal and see if we can make it normal.
df["actual_time_log"] = np.log1p(df["actual_time"])
df["osrm_time_log"] = np.log1p(df["osrm_time"])

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["actual_time_log"])
plt.subplot(1,2,2)
sns.kdeplot(df["osrm_time_log"])
plt.show()
```



Inference -

- Both are appearing to have a long tail, hence right skewed distribution.
- Shapiro test cannot be done as it cannot handle large datasets.
- Parametric tests cannot be used to check the statistical significance.
- Log normal function also couldnt make it completely normal.

```
# Non - parametric tests:
# Kruskal wallis test:

stat, p_val = kruskal(df["actual_time"], df["osrm_time"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

```
KW statistic is: 3363.9317112678514 and P value is: 0.0
Reject H0: Medians are significantly different
```

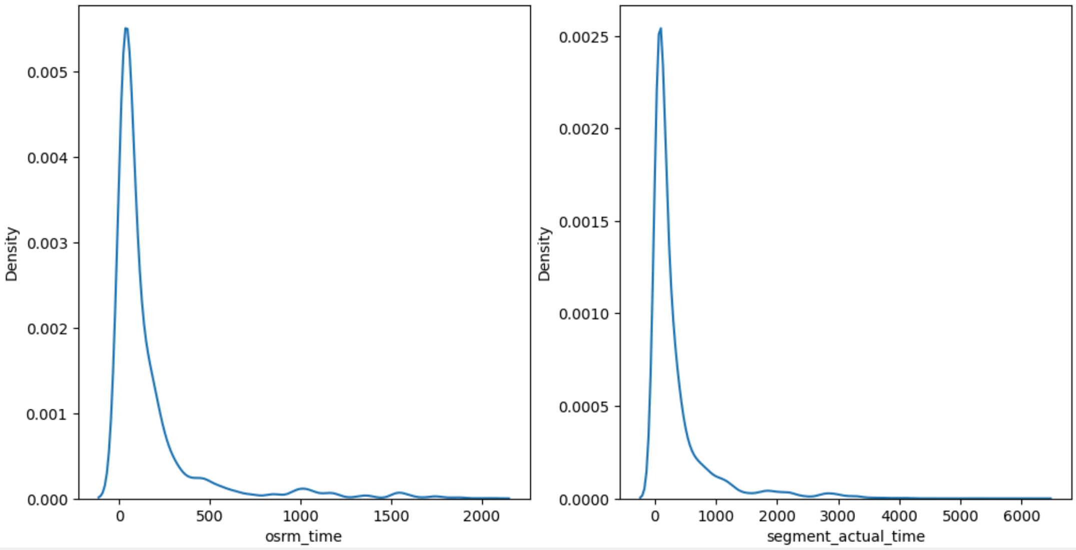
```
# Trying to check the significance using log normal transformed data -
# Kruskal wallis test:

stat, p_val = kruskal(df["actual_time_log"], df["osrm_time_log"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

```
KW statistic is: 3363.9317112678514 and P value is: 0.0
Reject H0: Medians are significantly different
```

Inference -

- The P value is less than 5% and this means that Null hypothesis can be rejected.
- The actual time and osrm time means that are significantly different.
- Since time and distance metrics (segment, osrm actual) are strongly correlated (0.96), and we have already tested the significance of the difference in actual time and segment time, the distance metrics are likely to follow a very similar pattern. In most cases, testing actual distance vs. segment distance would not yield additional insights because the high correlation implies that the behavior of the distances is already aligned with that of the times.

```
# Analysis between osrm aggregated value and segmented time aggregated value:

# checking normality by plotting the density plots-
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["osrm_time"])
plt.subplot(1,2,2)
sns.kdeplot(df["segment_actual_time"])
plt.show()
```



Inference -

- Both are appearing to have a long tail, hence right skewed distribution.
- Shapiro test cannot be done as it cannot handle large datasets.
- Parametric tests cannot be used to check the statistical significance.
- Log normal function also couldnt make it completely normal.

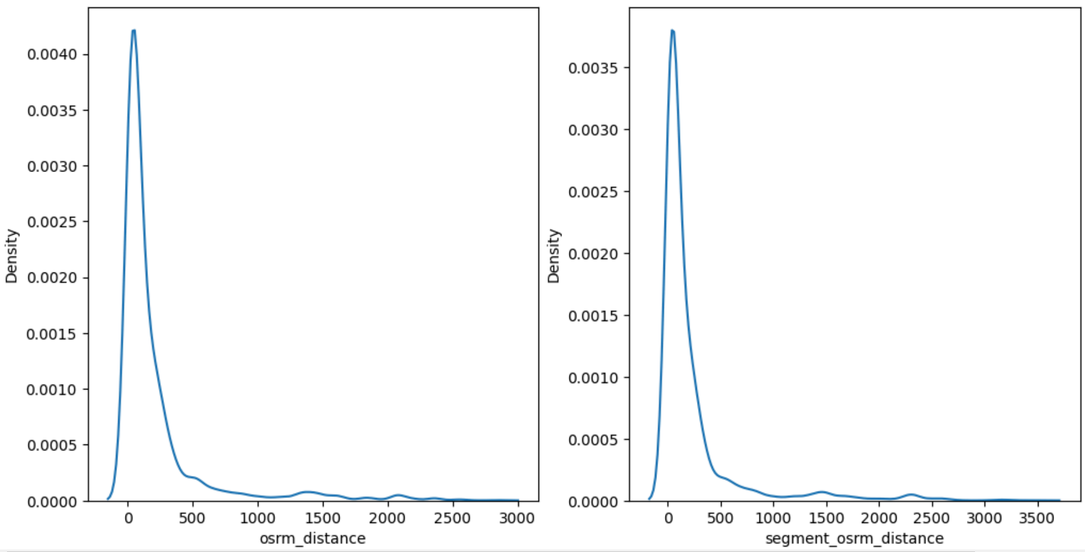
```
# Non - parametric tests:
# Kruskal wallis test:

stat, p_val = kruskal(df["segment_actual_time"], df["osrm_time"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

```
KW statistic is: 3285.9259815010146 and P value is: 0.0
Reject H0: Medians are significantly different
```

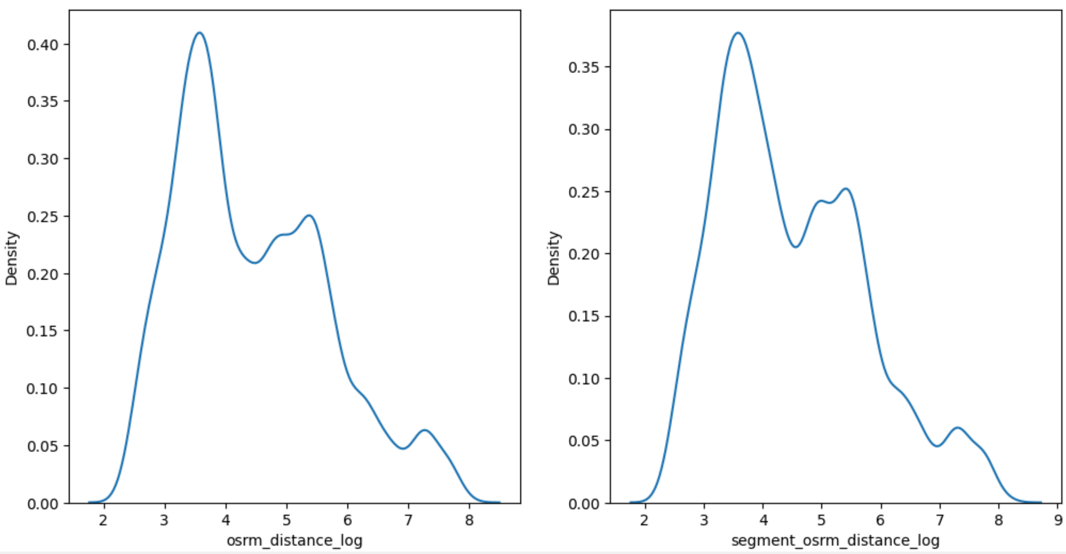
Inference -

- The P value is less than 5% and this means that Null hypothesis can be rejected.
- The segment actual time and osrm time means that are significantly different.

```
# Analysis between osrm distance aggregated value and segment osrm distance aggregated value:

# checking normality by plotting the density plots-
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["osrm_distance"])
plt.subplot(1,2,2)
sns.kdeplot(df["segment_osrm_distance"])
plt.show()
```




```
#Let us try converting data to log normal and see if we can make it normal.
df["osrm_distance_log"] = np.log1p(df["osrm_distance"])
df["segment_osrm_distance_log"] = np.log1p(df["segment_osrm_distance"])

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.kdeplot(df["osrm_distance_log"])
plt.subplot(1,2,2)
sns.kdeplot(df["segment_osrm_distance_log"])
plt.show()
```



Inference -

- Both are appearing to have a long tail, hence right skewed distribution.
- Shapiro test cannot be done as it cannot handle large datasets.
- Parametric tests cannot be used to check the statistical significance.
- Log normal function also couldnt make it completely normal.

```
# Non - parametric tests:
# Kruskal wallis test:

stat, p_val = kruskal(df["osrm_distance"], df["segment_osrm_distance"])
print("KW statistic is:", stat, "and P value is:", p_val)

if p_val < 0.05:
  print("Reject H0: Medians are significantly different")
else:
  print("Accept H0: Medians are not significantly different")
```

```
KW statistic is: 19.189169264591182 and P value is: 1.1838316476705933e-05
Reject H0: Medians are significantly different
```

Inference -

- The P value is less than 5% and this means that Null hypothesis can be rejected.
- The segment osrm distance and osrm distance means are significantly different.

Outlier Treatment -

```
data.describe()
```

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	
count	144867	144867	144867	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	2
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	0
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	1
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	1
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	2
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	77
std	NaN	NaN	NaN	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	1

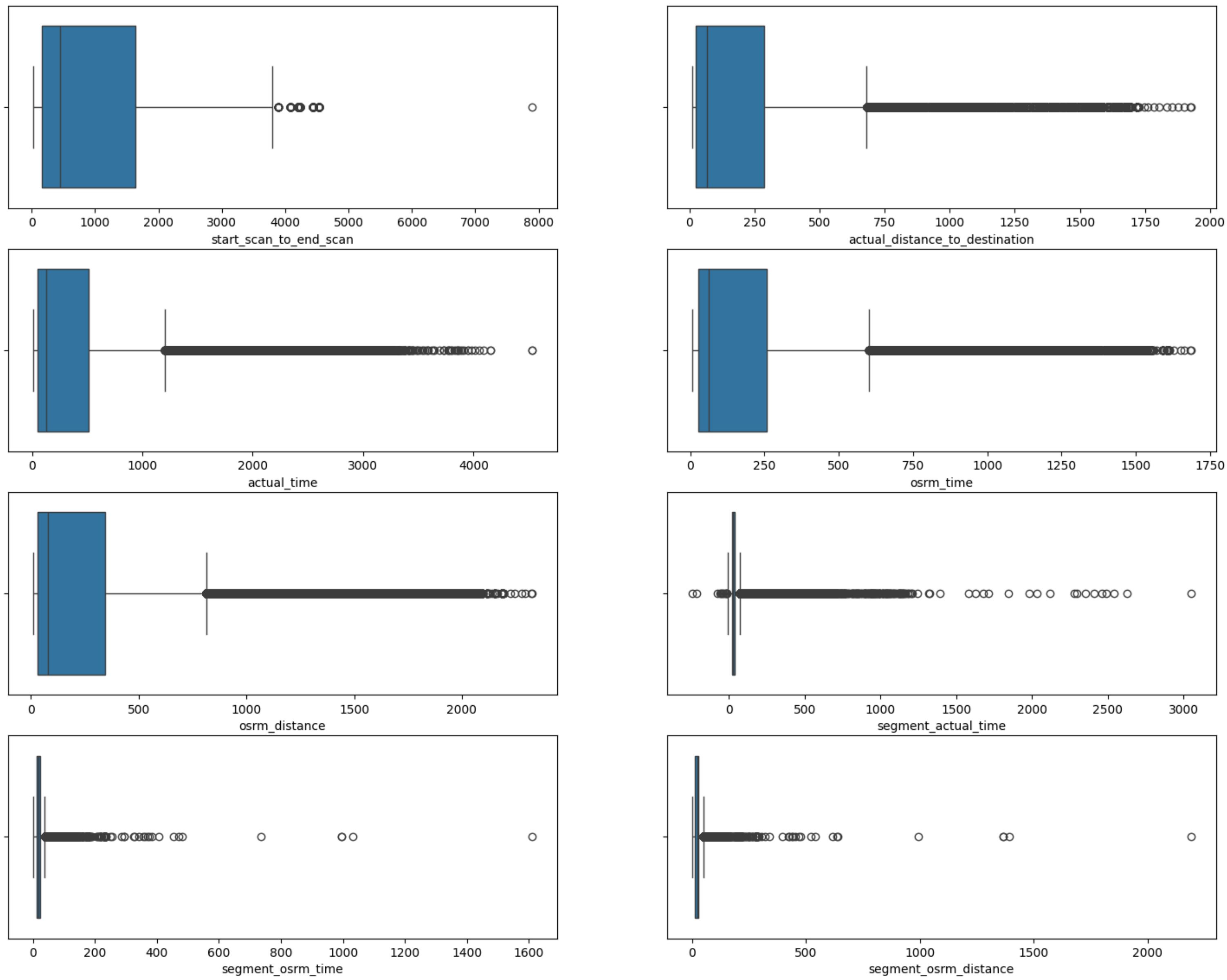
```
data.columns
```

```
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')
```

```
num_col = data.select_dtypes(include = ["int", "float64"])
num_col.drop(columns = ["cutoff_factor", "factor", "segment_factor"], inplace = True)
num_list = list(num_col.columns)
num_list
```

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

```
# Analysing the outliers in the numerical columns -

plt.figure(figsize = (18,14))
plt.subplot(4,2,1)
sns.boxplot(data = num_col, x = "start_scan_to_end_scan")
plt.subplot(4,2,2)
sns.boxplot(data = num_col, x = "actual_distance_to_destination")
plt.subplot(4,2,3)
sns.boxplot(data = num_col, x = "actual_time")
plt.subplot(4,2,4)
sns.boxplot(data = num_col, x = "osrm_time")
plt.subplot(4,2,5)
sns.boxplot(data = num_col, x = "osrm_distance")
plt.subplot(4,2,6)
sns.boxplot(data = num_col, x = "segment_actual_time")
plt.subplot(4,2,7)
sns.boxplot(data = num_col, x = "segment_osrm_time")
plt.subplot(4,2,8)
sns.boxplot(data = num_col, x = "segment_osrm_distance")
plt.show()
```




```
# Creating a dataset by clipping the dataset at minimum and maximum value:
data_iqr = data
for i in num_list:
  q1 = data[i].quantile(0.25)
  q3 = data[i].quantile(0.75)
  IQR = q3- q1
  upper = q3 + (1.5*IQR)
  lower = q1 - (1.5*IQR)
  print(f"lower limit of {i} = {lower}")
  print(f"upper limit of {i} = {upper}")
  print("----------------------------------------------------------------------------")
  data_iqr = data_iqr[~((data_iqr[i] < lower) | (data_iqr[i] > upper))]

data_iqr
```

```
lower limit of start_scan_to_end_scan = -2048.5
upper limit of start_scan_to_end_scan = 3843.5
---------------------------------------------------------------------------
lower limit of actual_distance_to_destination = -371.6736259929169
upper limit of actual_distance_to_destination = 681.7383749520162
---------------------------------------------------------------------------
lower limit of actual_time = -642.0
upper limit of actual_time = 1206.0
---------------------------------------------------------------------------
lower limit of osrm_time = -318.0
upper limit of osrm_time = 602.0
---------------------------------------------------------------------------
lower limit of osrm_distance = -440.0031250000001
upper limit of osrm_distance = 813.1110750000001
---------------------------------------------------------------------------
lower limit of segment_actual_time = -10.0
upper limit of segment_actual_time = 70.0
---------------------------------------------------------------------------
lower limit of segment_osrm_time = -5.5
upper limit of segment_osrm_time = 38.5
---------------------------------------------------------------------------
lower limit of segment_osrm_distance = -11.544625
upper limit of segment_osrm_distance = 51.427975
---------------------------------------------------------------------------
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time	...
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...
...	...	...	...	...	...	...	...	...	...	...	...
144861	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-20 16:24:28.436231	...
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-20 16:24:28.436231	...
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-20 16:24:28.436231	...
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-20 16:24:28.436231	...
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	2018-09-20 16:24:28.436231	...

114085 rows × 24 columns

```
data.select_dtypes(include = "object").columns
```

```
Index(['data', 'route_schedule_uuid', 'route_type', 'trip_uuid',
       'source_center', 'source_name', 'destination_center',
       'destination_name', 'cutoff_timestamp'],
      dtype='object')
```

```
# encoding of Route type and data using 1HE -

df_encoded = pd.get_dummies(data = data, columns = ["route_type", "data"], prefix = ["route_type", "data"])
```

```
df_encoded[["route_type_Carting",   "route_type_FTL",   "data_test", "data_training"]].value_counts()
```

route_type_Carting	route_type_FTL	data_test	data_training	count
False	**True**	**False**	**True**	73108
True	**False**	**False**	**True**	31750
False	**True**	**True**	**False**	26552
True	**False**	**True**	**False**	13457

dtype: int64

```
df_new = df_encoded
df_new.drop(columns = ["trip_creation_time", "source_name", "is_cutoff", "destination_name", "cutoff_factor", "factor", "cutoff_timestamp"], inplace = True)
```

Normalizing/ Standardizing the numerical features using MinMaxScaler or StandardScaler -

```
data[num_list] = MinMaxScaler().fit_transform(data[num_list])
data[num_list]
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance
0	0.008378	0.000748	0.001105	0.002976	0.001276	0.078300	0.006828	0.005460
1	0.008378	0.005180	0.003316	0.008333	0.005488	0.077086	0.005587	0.004453
2	0.008378	0.009715	0.006854	0.013095	0.010155	0.078907	0.004345	0.004935
3	0.008378	0.014135	0.011718	0.020238	0.015775	0.080425	0.007449	0.005942
4	0.008378	0.015839	0.013044	0.022619	0.019511	0.075873	0.003104	0.001787
...	...	...	...	...	...	...	...	...
144862	0.051663	0.018900	0.018793	0.032143	0.025427	0.077693	0.007449	0.003735
144863	0.051663	0.023505	0.024541	0.041667	0.033090	0.081942	0.013035	0.007928
144864	0.051663	0.029797	0.028963	0.048810	0.038014	0.080121	0.021105	0.009448
144865	0.051663	0.033715	0.032943	0.054762	0.044132	0.079211	0.016760	0.008619
144866	0.051663	0.031817	0.092195	0.052976	0.034405	0.155387	0.005587	0.004020

144867 rows × 8 columns

Recommendations and Insights summary:

1. For deliver the products 61% prioritize carting shipments route type because FTL shipments Full Truck Load takes more time to deliver product and time taken to actual and osrm higher than that of rout type. Using carting shipments helps to reach destination sooner provided it is feasible.
2. Most of the products were delivered during wednesday (18.5%) followed by saturday and thursday. Least number of products were delivered on sunday Delhivery could optimize resource allocation and scheduling to handle peak delivery days, particularly on Wednesday, while exploring strategies to boost efficiency and demand on the lower-performing days, such as Monday and Sunday.
3. It is observed that Gurgaon, Hariyana is the destination city to which "distance to reach destination" is the highest compared to other cities in the states. Delhivery could consider optimizing routes or exploring alternative transportation methods for shipments to Gurgaon, Haryana, to reduce delivery times and improve cost efficiency for long-distance deliveries.
4. The highest number of deliveries and sources are occurring in Maharashtra followed closely by Karnataka. Delhivery could focus on scaling operations and resource allocation in Maharashtra and Karnataka to meet the high delivery demand and ensure optimized performance in these key regions.
5. Full Truck Load (FTL) takes more time to deliver than small vehicles (carts). The actual mean time for carting is 6 times less, and the OSRM mean time for carting is 3 times less compared to FTL. Delhivery could explore increasing the use of carting for shorter routes or time-sensitive deliveries to optimize delivery times, while reserving FTL for larger or longer distance shipments.
6. In states/ UT Dadra & Nagar Haveli and TN the difference between actual mean time and osrm mean time is almost nill. Delhivery could investigate the factors contributing to the minimal difference in Dadra & Nagar Haveli and Tamil Nadu to replicate these efficiencies in other regions, while also focusing on understanding the causes of the high variation in Mizoram and Haryana to improve routing accuracy and delivery times.
7. Delhivery could focus on optimizing time predictions by studying the regions with high variations in time (Mizoram, Haryana, Goa, and Assam), while leveraging the regions with minimal distance discrepancies (Dadra & Nagar Haveli, Delhi, Tamil Nadu) as benchmarks for improving time and distance forecasting models.
8. Means of "start_scan_to_end_scan" and "total_min_diff" are significantly different. Delhivery should investigate the factors driving this difference as this could uncover inefficiencies in the scanning process or reveal opportunities for optimizing delivery timelines. It should conduct a thorough review of the data collection process to identify potential human or system errors.
9. The actual time and segment actual time have means that are significantly similar. Delhivery could evaluate whether the granularity of segment-based analysis is necessary, as the similarity between actual time and segment actual time suggests that segmenting may not be providing additional value in optimizing delivery times.
10. All the numerical columns are right-skewed in distribution, with almost 11% of the data containing outliers. Delhivery should investigate the causes of outliers to determine whether they should be removed or managed through robust statistical techniques.