In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

About Delhivery

- Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities. The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

Loading the Dataset

In [2]:
```
df=pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181')
df.head()
```

Out[2]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388

5 rows × 24 columns

What is the shape of the loaded dataset ?

In [3]:
```
df.shape
```

Out[3]: (144867, 24)

What are the columns present in the dataset?

In [4]:
```
df.columns
```

Out[4]:
```
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_tim
e',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

Basic Information about the Dataset

In [5]:
```
df.dtypes
```

Out[5]:

	0
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
is_cutoff	bool
cutoff_factor	int64
cutoff_timestamp	object
actual_distance_to_destination	float64
actual_time	float64
osrm_time	float64
osrm_distance	float64
factor	float64
segment_actual_time	float64
segment_osrm_time	float64
segment_osrm_distance	float64
segment_factor	float64

dtype: object

In [6]:
```
df.info()
```

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

Dropping unknown fields

In [7]:
```
unknown_fields=['is_cutoff', 'cutoff_factor','cutoff_timestamp','factor','s
egment_factor']
df.drop(unknown_fields,axis=1,inplace=True)
```

How many unique entries present in each column ?

In [8]:
```
for i in df.columns:
    print(f"Unique entries for column {i:<30} = {df[i].nunique()}")
```

```
Unique entries for column data                           = 2
Unique entries for column trip_creation_time             = 14817
Unique entries for column route_schedule_uuid            = 1504
Unique entries for column route_type                     = 2
Unique entries for column trip_uuid                      = 14817
Unique entries for column source_center                  = 1508
Unique entries for column source_name                    = 1498
Unique entries for column destination_center             = 1481
Unique entries for column destination_name               = 1468
Unique entries for column od_start_time                  = 26369
Unique entries for column od_end_time                    = 26369
Unique entries for column start_scan_to_end_scan         = 1915
Unique entries for column actual_distance_to_destination = 144515
Unique entries for column actual_time                    = 3182
Unique entries for column osrm_time                      = 1531
Unique entries for column osrm_distance                  = 138046
Unique entries for column segment_actual_time            = 747
Unique entries for column segment_osrm_time              = 214
Unique entries for column segment_osrm_distance          = 113799
```

Updating the datatype of the datetime columns

In [9]:
```
datetime_columns = ['trip_creation_time', 'od_start_time', 'od_end_time']
for i in datetime_columns:
    df[i] = pd.to_datetime(df[i])
```

1. Basic data cleaning and exploration:

1. Handle missing values in the data.

In [10]:
```
df.isnull().sum()
```

Out[10]:

	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
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0

dtype: int64

In [11]:
```
df = df.dropna(how='any')
df = df.reset_index(drop=True)
```

In [12]:
```
df.isnull().sum()
```

Out[12]:

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	0
destination_center	0
destination_name	0
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0

dtype: int64

In [14]:
```
df.describe()
```

Out[14]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	ac
count	144316	144316	144316	144316.000000	
mean	2018-09-22 13:05:09.454117120	2018-09-22 17:32:42.435769344	2018-09-23 09:36:54.057172224	963.697698	
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	
25%	2018-09-17 02:46:11.004421120	2018-09-17 07:37:35.014584832	2018-09-18 01:29:56.978912	161.000000	
50%	2018-09-22 03:36:19.186585088	2018-09-22 07:35:23.038482944	2018-09-23 02:49:00.936600064	451.000000	
75%	2018-09-27 17:53:19.027942912	2018-09-27 22:01:30.861209088	2018-09-28 12:13:41.675546112	1645.000000	
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	
std	NaN	NaN	NaN	1038.082976	

In [15]:
```
df.describe(include='object')
```

Out[15]:

	data	route_schedule_uuid	route_type	trip_uuid	source_center	
count	144316	144316	144316	144316	144316	
unique	2	1497	2	14787	1496	
top	training	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	FTL	trip-153837029526866991	IND000000ACB	Gur
freq	104632	1812	99132	101	23267	

1. **Merging the rows.**

1. Grouping by segment

a. Create a unique identifier for different segments of a trip based on the combination of the trip_uuid, source_center, and destination_center and it as segment_key.

In [16]:
```
df['segment_key'] = df['trip_uuid'] + df['source_center'] + df['destination
_center']

segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_os
rm_time']

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


df[[col + '_sum' for col in segment_cols]]
```

Out[16]:

	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_sum
0	14.0	11.9653	11.0
1	24.0	21.7243	20.0
2	40.0	32.5395	27.0
3	61.0	45.5619	39.0
4	67.0	49.4772	44.0
...	...	...	...
144311	92.0	65.3487	94.0
144312	118.0	82.7212	115.0
144313	138.0	103.4265	149.0
144314	155.0	122.3150	176.0
144315	423.0	131.1238	185.0

144316 rows × 3 columns

1. Aggregating at segment level

   a. Create a dictionary named **create_segment_dict**, that defines how to aggregate and select values.

In [17]:
```
create_segment_dict = {

    'data' : 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_center' : 'first',
    'source_name' : 'first',
    'destination_center' : 'last',
    'destination_name' : 'last',
    'od_start_time' : 'first',
    'od_end_time' : 'first',
    'start_scan_to_end_scan' : 'first',
    'actual_distance_to_destination' : 'last',
    'actual_time' : 'last',
    'osrm_time' : 'last',
    'osrm_distance' : 'last',
    'segment_actual_time_sum' : 'last',
    'segment_osrm_distance_sum' : 'last',
    'segment_osrm_time_sum' : 'last',

    }
```

In [18]:
```
segment = df.groupby('segment_key').agg(create_segment_dict).reset_index()
segment = segment.sort_values(by=['segment_key','od_end_time'], ascending=True).reset_index()
```

In [19]:
```
segment
```

Out[19]:

	index	segment_key	data	trip_creation_time	
0	0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	tha
1	1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	tha
2	2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	tha
3	3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	tha
4	4	trip-153671043369099517IND000000ACBIND160002AAC	training	2018-09-12 00:00:33.691250	tha
...	...	...	...	...	
26217	26217	trip-153861115439069069IND628204AAAIND627657AAA	test	2018-10-03 23:59:14.390954	th
26218	26218	trip-153861115439069069IND628613AAAIND627005AAA	test	2018-10-03 23:59:14.390954	th
26219	26219	trip-153861115439069069IND628801AAAIND628204AAA	test	2018-10-03 23:59:14.390954	th
26220	26220	trip-153861118270144424IND583119AAAIND583101AAA	test	2018-10-03 23:59:42.701692	tha
26221	26221	trip-153861118270144424IND583201AAAIND583119AAA	test	2018-10-03 23:59:42.701692	tha

26222 rows × 21 columns

3. Feature Engineering:

Extract features from the below fields:

1. Calculate time taken between od_start_time and od_end_time and keep it as a feature named od_time_diff_hour. Drop the original columns, if required.

In [20]:
```
#Calculate time taken between od_start_time and od_end_time and keep it as
a feature named od_time_diff_hour.
segment['od_time_diff_hour'] = (segment['od_end_time'] - segment['od_start_
time']).dt.total_seconds() / (3600)
segment['od_time_diff_hour']
```

Out[20]:

	od_time_diff_hour
0	21.010074
1	16.658423
2	0.980540
3	2.046325
4	13.910649
...	...
26217	1.035253
26218	1.518130
26219	0.736240
26220	4.791233
26221	1.115559

26222 rows × 1 columns

dtype: float64

In [21]:
```
#Drop the original columns
segment.drop(['od_start_time', 'od_end_time'], axis=1, inplace=True)
```

4. In-depth analysis:

1. Grouping and Aggregating at Trip-level

   a. Groups the segment data by the **trip_uuid** column to focus on aggregating data at the trip level.

   b. Apply suitable aggregation functions like first, last, and sum specified in the **create_trip_dict** dictionary to calculate summary statistics for each trip.

In [22]:
```
create_trip_dict = {

    'data' : 'first',
    'trip_creation_time': 'first',
    'route_schedule_uuid' : 'first',
    'route_type' : 'first',
    'trip_uuid' : 'first',
    'source_center' : 'first',
    'source_name' : 'first',
    'destination_center' : 'last',
    'destination_name' : 'last',
    'start_scan_to_end_scan' : 'sum',
    'od_time_diff_hour' : 'sum',
    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',
    'segment_actual_time_sum' : 'sum',
    'segment_osrm_distance_sum' : 'sum',
    'segment_osrm_time_sum' : 'sum',

    }
```

In [23]:
```
trip = segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop
= True)
```

In [24]:
```
trip
```

Out[24]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	s
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IN
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IN
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IN
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IN
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IN
...	...	...	...	...	...	
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IN
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IN
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IN
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IN
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IN

14787 rows × 18 columns

In [25]:
```
trip[['actual_time', 'segment_actual_time_sum']]
```

Out[25]:

	actual_time	segment_actual_time_sum
0	1562.0	1548.0
1	143.0	141.0
2	3347.0	3308.0
3	59.0	59.0
4	341.0	340.0
...	...	...
14782	83.0	82.0
14783	21.0	21.0
14784	282.0	281.0
14785	264.0	258.0
14786	275.0	274.0

14787 rows × 2 columns

In [26]:
```
def location_name_to_state(x):
  l=x.split('(')
  if len(l)==1:
    return l[0]
  else:
    return l[1].replace(')','')
```

In [27]:
```
#Source Name: Split and extract features out of destination. City-place-cod
e(State)
trip['source_state']=trip['source_name'].apply(location_name_to_state)
trip['source_city'] = trip['source_name'].apply(lambda x: x.split('_')[0])
#Destination Name: Split and extract features out of destination. City-plac
e-code(State)
trip['destination_state']=trip['destination_name'].apply(location_name_to_s
tate)
trip['destination_city'] = trip['destination_name'].apply(lambda x: x.split
('_')[0])
```

In [28]:
```
trip[['source_state','destination_state','source_city','destination_city']]
```

Out[28]:

	source_state	destination_state	source_city	destination_city
0	Uttar Pradesh	Uttar Pradesh	Kanpur	Kanpur
1	Karnataka	Karnataka	Doddablpur	Doddablpur
2	Haryana	Haryana	Gurgaon	Gurgaon
3	Maharashtra	Maharashtra	Mumbai Hub (Maharashtra)	Mumbai
4	Karnataka	Karnataka	Bellary	Sandur
...	...	...	...	...
14782	Punjab	Punjab	Chandigarh	Chandigarh
14783	Haryana	Haryana	FBD	Faridabad
14784	Uttar Pradesh	Uttar Pradesh	Kanpur	Kanpur
14785	Tamil Nadu	Tamil Nadu	Tirunelveli	Tirchchndr
14786	Karnataka	Karnataka	Sandur	Sandur

14787 rows × 4 columns

In [30]:
```
#Trip_creation_time: Extract features like month, year, day, etc.
trip['trip_creation_month']=trip['trip_creation_time'].dt.month
trip['trip_creation_year']=trip['trip_creation_time'].dt.year
trip['trip_creation_day']=trip['trip_creation_time'].dt.day
```

In [31]:
```
trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14787 non-null  object
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  object
 4   trip_uuid                       14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  od_time_diff_hour               14787 non-null  float64
 11  actual_distance_to_destination  14787 non-null  float64
 12  actual_time                     14787 non-null  float64
 13  osrm_time                       14787 non-null  float64
 14  osrm_distance                   14787 non-null  float64
 15  segment_actual_time_sum         14787 non-null  float64
 16  segment_osrm_distance_sum       14787 non-null  float64
 17  segment_osrm_time_sum           14787 non-null  float64
 18  source_state                    14787 non-null  object
 19  source_city                     14787 non-null  object
 20  destination_state               14787 non-null  object
 21  destination_city                14787 non-null  object
 22  trip_creation_month             14787 non-null  int32
 23  trip_creation_year              14787 non-null  int32
 24  trip_creation_day               14787 non-null  int32
dtypes: datetime64[ns](1), float64(9), int32(3), object(12)
memory usage: 2.7+ MB
```

In [32]:
```
trip.describe().T
```

Out[32]:

	count	mean	min	25%
trip_creation_time	14787	2018-09-22 12:26:28.269885696	2018-09-12 00:00:16.535741	2018-09-17 02:38:18.12843187
start_scan_to_end_scan	14787.0	529.429025	23.0	149.0
od_time_diff_hour	14787.0	8.838559	0.391024	2.49497
actual_distance_to_destination	14787.0	164.090196	9.002461	22.77709
actual_time	14787.0	356.306012	9.0	67.0
osrm_time	14787.0	160.990938	6.0	29.0
osrm_distance	14787.0	203.887411	9.0729	30.756
segment_actual_time_sum	14787.0	353.059174	9.0	66.0
segment_osrm_distance_sum	14787.0	222.705466	9.0729	32.5788
segment_osrm_time_sum	14787.0	180.511598	6.0	30.0
trip_creation_month	14787.0	9.120105	9.0	9.0
trip_creation_year	14787.0	2018.0	2018.0	2018.0
trip_creation_day	14787.0	18.375127	1.0	14.0

In [33]:
```
trip.describe(include='object').T
```

Out[33]:

	count	unique	top	freq
data	14787	2	training	10645
route_schedule_uuid	14787	1497	thanos::sroute:a16bfa03-3462-4bce-9c82-5784c7d...	53
route_type	14787	2	Carting	8906
trip_uuid	14787	14787	trip-153671041653548748	1
source_center	14787	930	IND000000ACB	1052
source_name	14787	930	Gurgaon_Bilaspur_HB (Haryana)	1052
destination_center	14787	1035	IND000000ACB	821
destination_name	14787	1035	Gurgaon_Bilaspur_HB (Haryana)	821
source_state	14787	29	Maharashtra	2714
source_city	14787	731	Gurgaon	1128
destination_state	14787	31	Maharashtra	2561
destination_city	14787	856	Bengaluru	1088

I am interested to know what is the distribution of number of trips created from different states

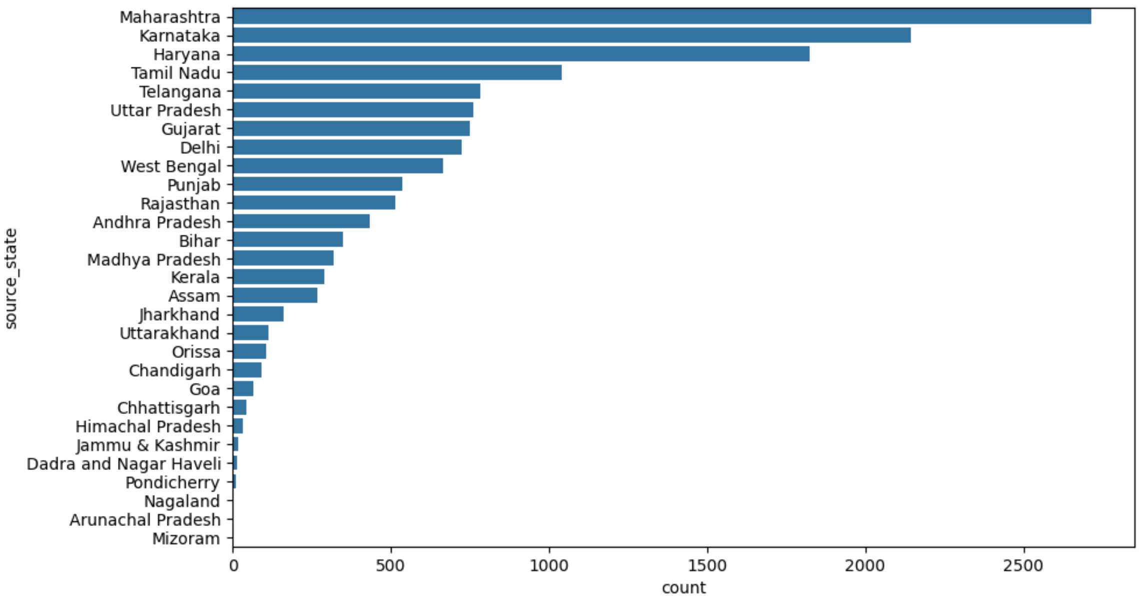
In [36]:
```
df_source_state=trip['source_state'].value_counts().reset_index()
df_source_state[:5]
```

Out[36]:

	source_state	count
0	Maharashtra	2714
1	Karnataka	2143
2	Haryana	1823
3	Tamil Nadu	1039
4	Telangana	784

In [38]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='source_state', data=df_source_state)
```

Out[38]: <Axes: xlabel='count', ylabel='source_state'>



- It can be seen in the above plot that maximum trips originated from Maharashtra state followed by Karnataka and Haryana. That means that the seller base is strong in these states.

I am interested to know top 30 cities based on the number of trips created from different cities

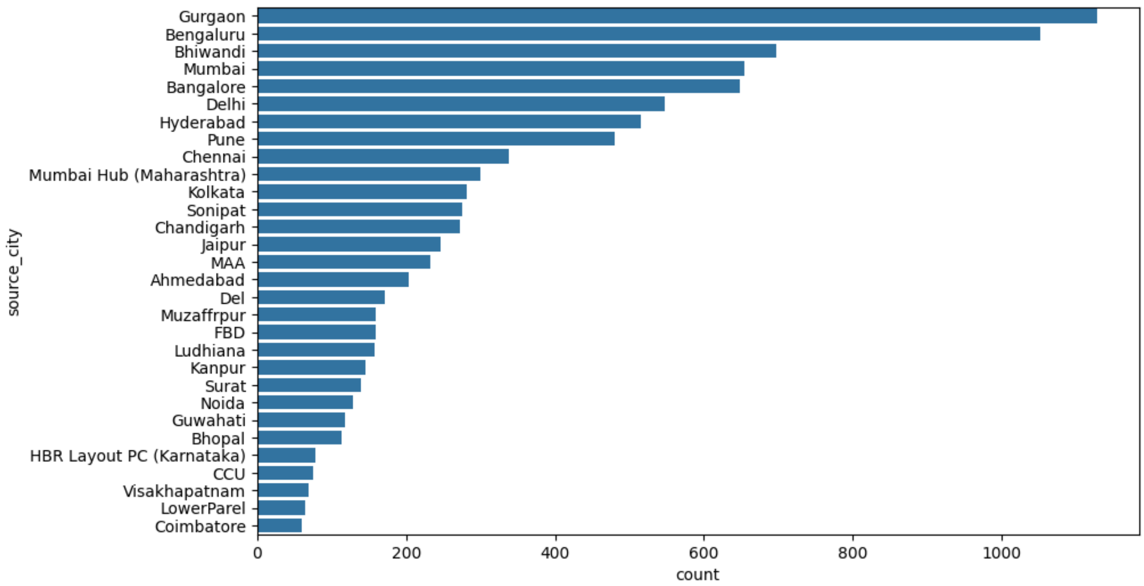
In [39]:
```
df_source_city=trip['source_city'].value_counts().reset_index()
df_source_city[:30]
```

Out[39]:

	source_city	count
0	Gurgaon	1128
1	Bengaluru	1052
2	Bhiwandi	697
3	Mumbai	654
4	Bangalore	648
5	Delhi	548
6	Hyderabad	515
7	Pune	480
8	Chennai	338
9	Mumbai Hub (Maharashtra)	300
10	Kolkata	281
11	Sonipat	275
12	Chandigarh	272
13	Jaipur	246
14	MAA	232
15	Ahmedabad	204
16	Del	172
17	Muzaffrpur	159
18	FBD	159
19	Ludhiana	158
20	Kanpur	145
21	Surat	140
22	Noida	129
23	Guwahati	118
24	Bhopal	114
25	HBR Layout PC (Karnataka)	79
26	CCU	75
27	Visakhapatnam	69
28	LowerParel	65
29	Coimbatore	60

In [40]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='source_city', data=df_source_city[:30])
```

Out[40]: <Axes: xlabel='count', ylabel='source_city'>



- It can be seen in the above plot that maximum trips originated from Gurgaon city followed by Bengaluru,Bhiwandi and Mumbai. That means that the seller base is strong in these cities.

I am interested to know what is the distribution of number of trips which ended in different states

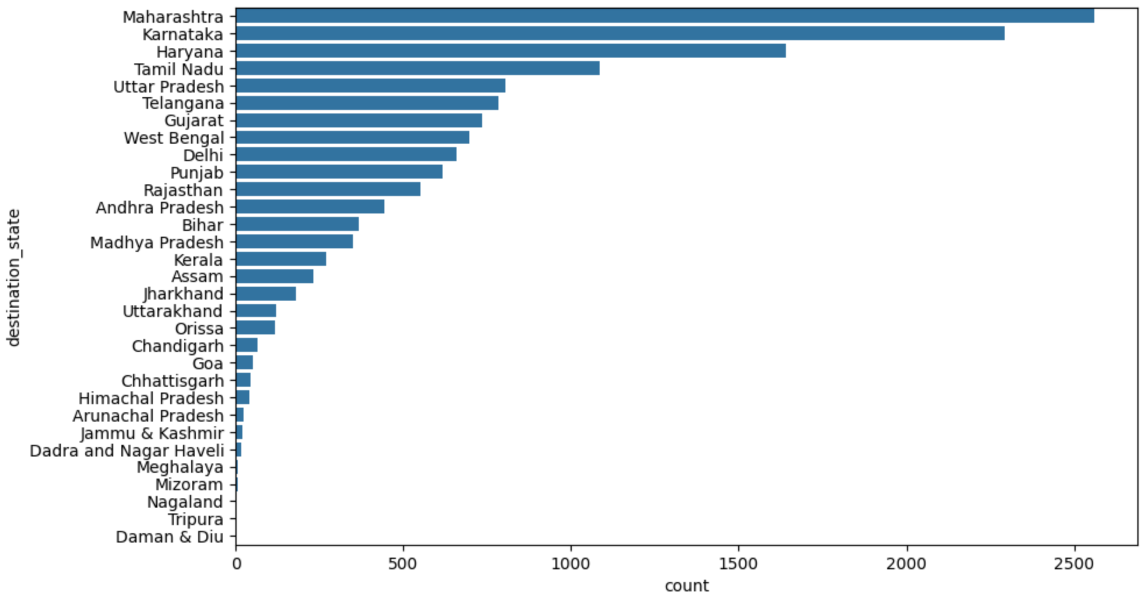
In [41]:
```
df_destination_state=trip['destination_state'].value_counts().reset_index()
df_destination_state[:5]
```

Out[41]:

	destination_state	count
0	Maharashtra	2561
1	Karnataka	2294
2	Haryana	1640
3	Tamil Nadu	1084
4	Uttar Pradesh	805

In [42]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='destination_state', data=df_destination_state)
```

Out[42]: <Axes: xlabel='count', ylabel='destination_state'>



- It can be seen in the above plot that maximum trips ended in Maharashtra state followed by Karnataka, Haryana, Tamil Nadu and Uttar Pradesh. That means that the number of orders placed in these states is significantly high in these states.

I am interested to know top 30 cities based on the number of trips ended in different cities

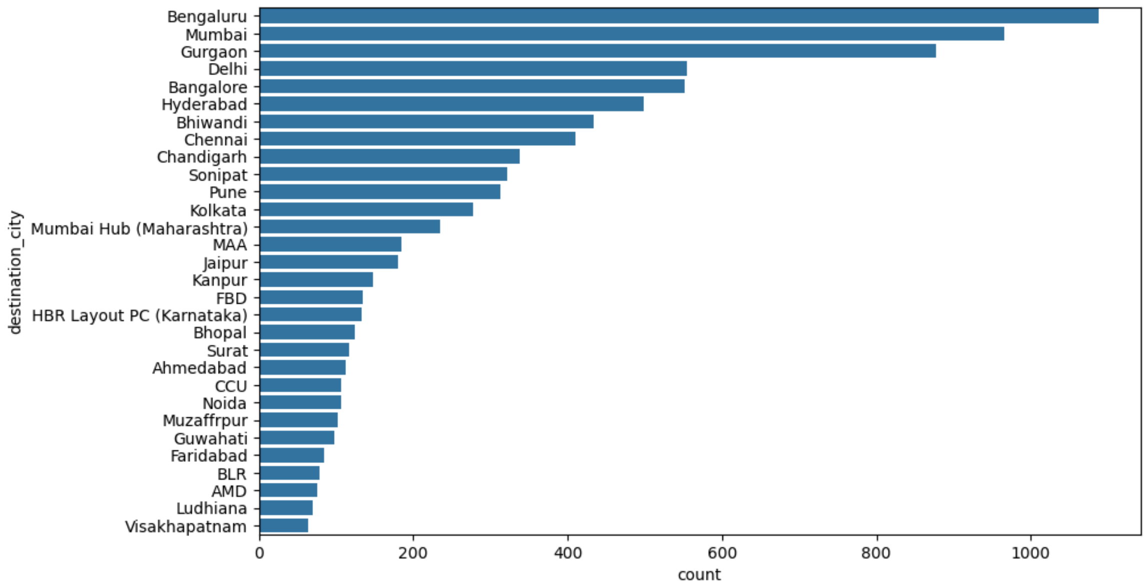
In [43]:
```
df_destination_city=trip['destination_city'].value_counts().reset_index()
df_destination_city[:30]
```

Out[43]:

	destination_city	count
0	Bengaluru	1088
1	Mumbai	966
2	Gurgaon	877
3	Delhi	554
4	Bangalore	551
5	Hyderabad	499
6	Bhiwandi	434
7	Chennai	410
8	Chandigarh	338
9	Sonipat	322
10	Pune	313
11	Kolkata	277
12	Mumbai Hub (Maharashtra)	234
13	MAA	185
14	Jaipur	180
15	Kanpur	148
16	FBD	135
17	HBR Layout PC (Karnataka)	133
18	Bhopal	124
19	Surat	117
20	Ahmedabad	113
21	CCU	107
22	Noida	106
23	Muzaffrpur	102
24	Guwahati	98
25	Faridabad	85
26	BLR	78
27	AMD	75
28	Ludhiana	70
29	Visakhapatnam	64

In [44]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='destination_city', data=df_destination_city[:30])
```

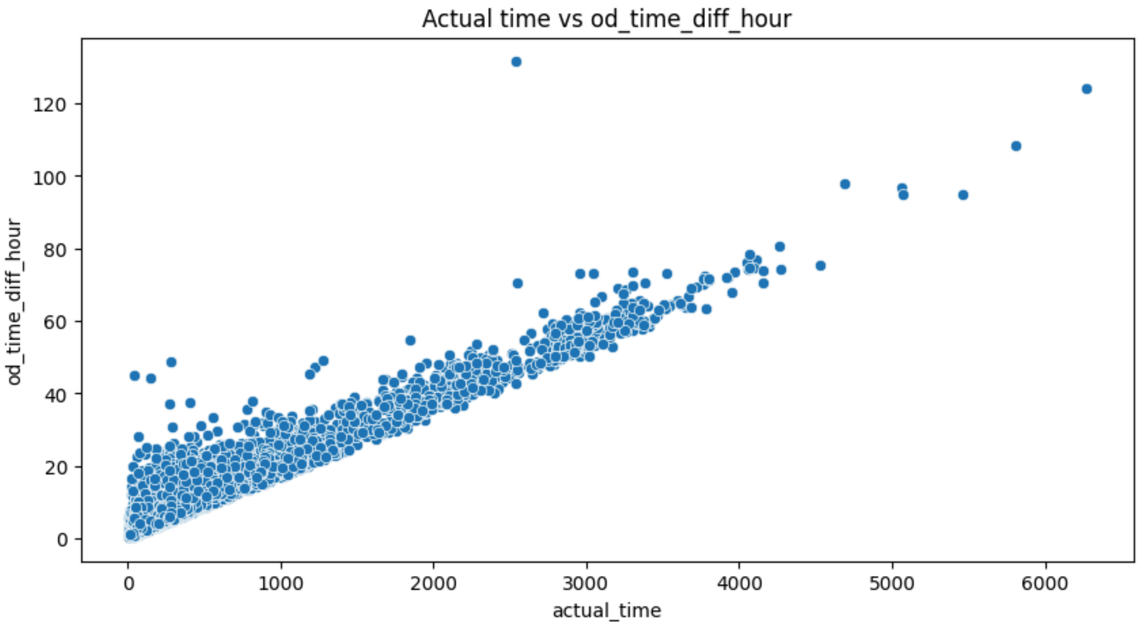
Out[44]: <Axes: xlabel='count', ylabel='destination_city'>



- It can be seen in the above plot that maximum trips ended in Bengaluru city followed by Mumbai, Gurgaon, Delhi. That means that the number of orders placed in these cities is significantly high.

5. Hypothesis Testing:

In [ ]:
```
# 3.2 Hypothesis Testing and Visual Analysis
# Compare actual_time with start_scan_to_end_scan using visual analysis
plt.figure(figsize=(10, 5))
sns.scatterplot(x=trip['actual_time'], y=trip['od_time_diff_hour'])
plt.title('Actual time vs od_time_diff_hour')
plt.show()
```



Null Hypothesis (H0): There is no significant difference between od_time_diff_hour (Point a) and start_scan_to_end_scan.

Alternate Hypothesis (H1): There is a significant difference between od_time_diff_hour (Point a) and start_scan_to_end_scan.

In [ ]:
```
# Hypothesis testing
from scipy import stats
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['od_time_diff_h
our'])
print(f'T-statistic: {t_stat}, P-value: {p_value}')

if p_value<0.05:
  print("Reject Null Hypothesis")
else:
  print("Accept Null Hypothesis")
```

```
T-statistic: 76.68717434417965, P-value: 0.0
Reject Null Hypothesis
```

a. actual_time aggregated value and OSRM time aggregated value.

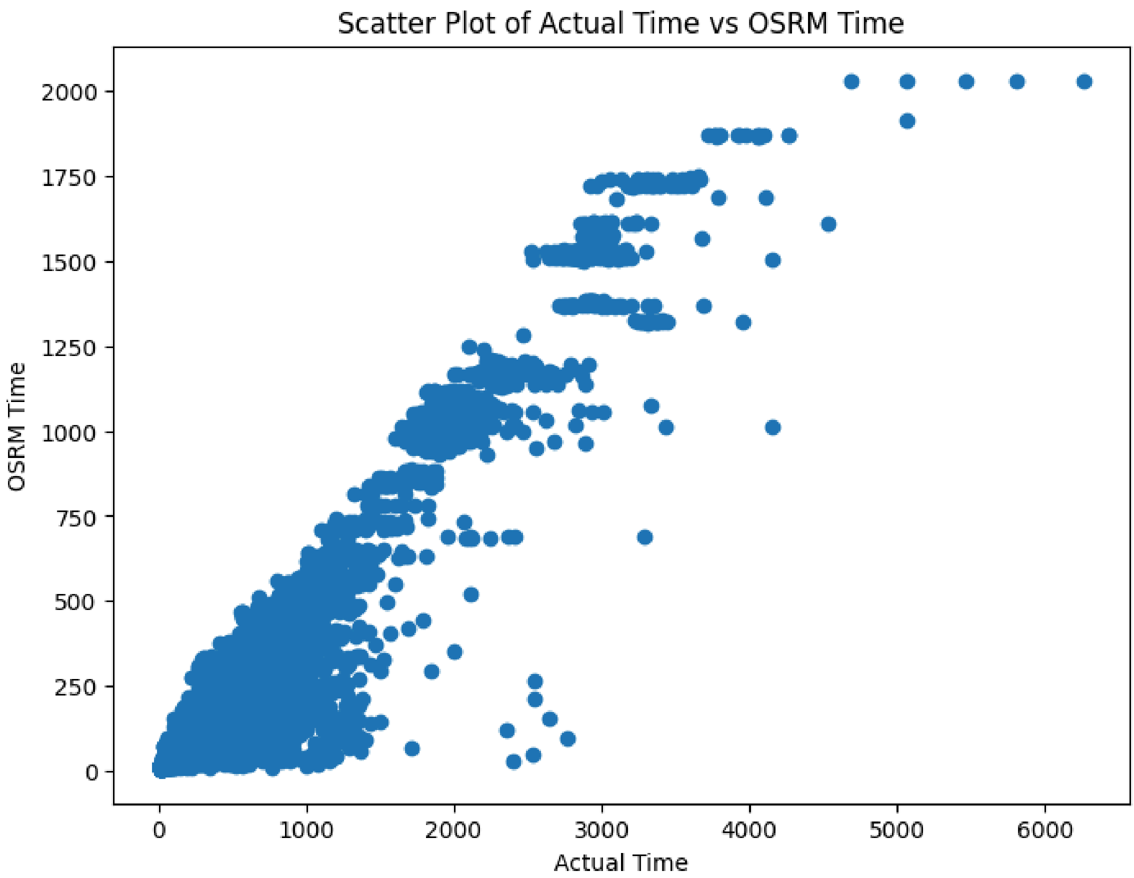
In [ ]:
```
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['osrm_time'])

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

alpha=0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference be
tween the actual_time and osrm_time.")
else:
    print("Fail to reject the null hypothesis. There is no significant diff
erence between the actual_time and osrm_time.")
```

```
T-statistic: 76.37699387098537
P-value: 0.0
Reject the null hypothesis. There is a significant difference between the
actual_time and osrm_time.
```

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(trip['actual_time'], trip['osrm_time'])
plt.xlabel('Actual Time')
plt.ylabel('OSRM Time')
plt.title('Scatter Plot of Actual Time vs OSRM Time')
plt.show()
```



b. actual_time aggregated value and segment actual time aggregated value.

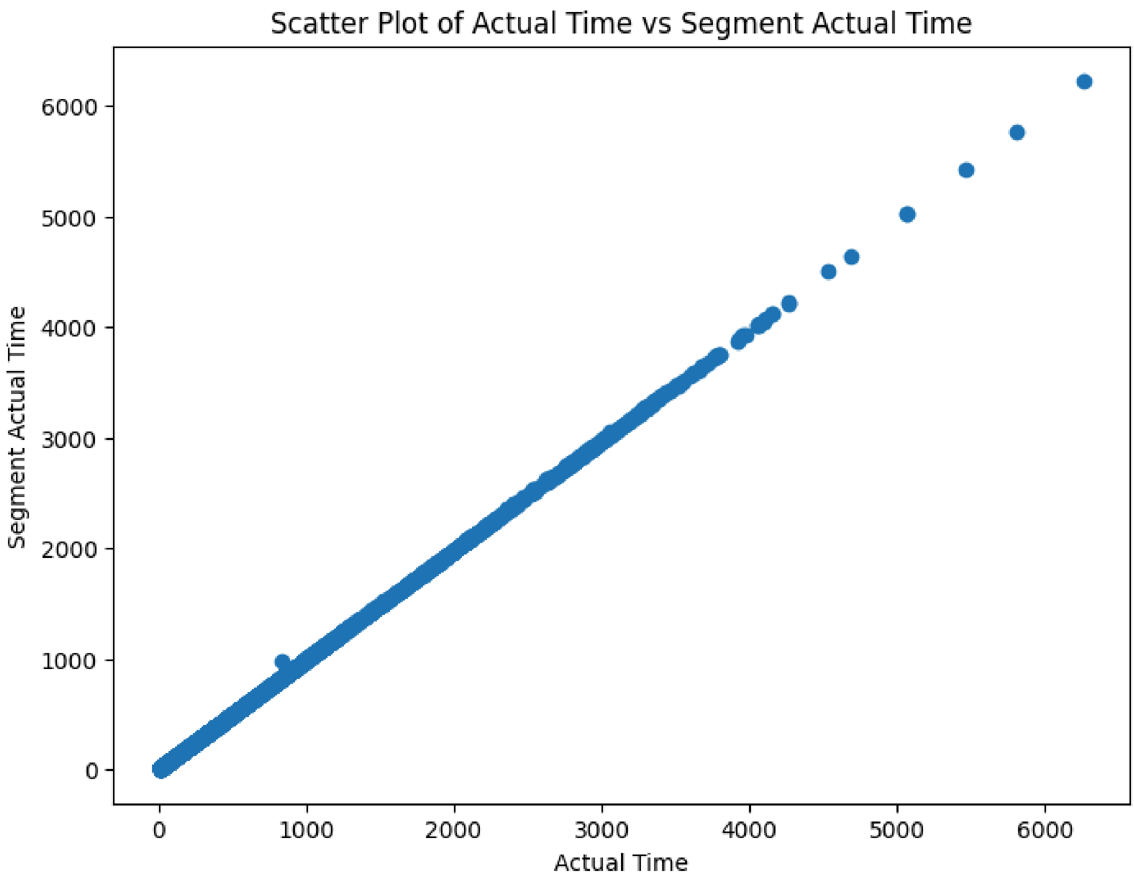
In [ ]:
```
t_stat, p_value = stats.ttest_rel(trip['actual_time'], trip['segment_actual
_time_sum'])

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

alpha=0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference be
tween the actual_time and segment_actual_time.")
else:
    print("Fail to reject the null hypothesis. There is no significant diff
erence between the actual_time and segment_actual_time.")
```

```
T-statistic: 68.26327799172758
P-value: 0.0
Reject the null hypothesis. There is a significant difference between the
actual_time and segment_actual_time.
```

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(trip['actual_time'], trip['segment_actual_time_sum'])
plt.xlabel('Actual Time')
plt.ylabel('Segment Actual Time')
plt.title('Scatter Plot of Actual Time vs Segment Actual Time')
plt.show()
```



c. OSRM distance aggregated value and segment OSRM distance aggregated value.

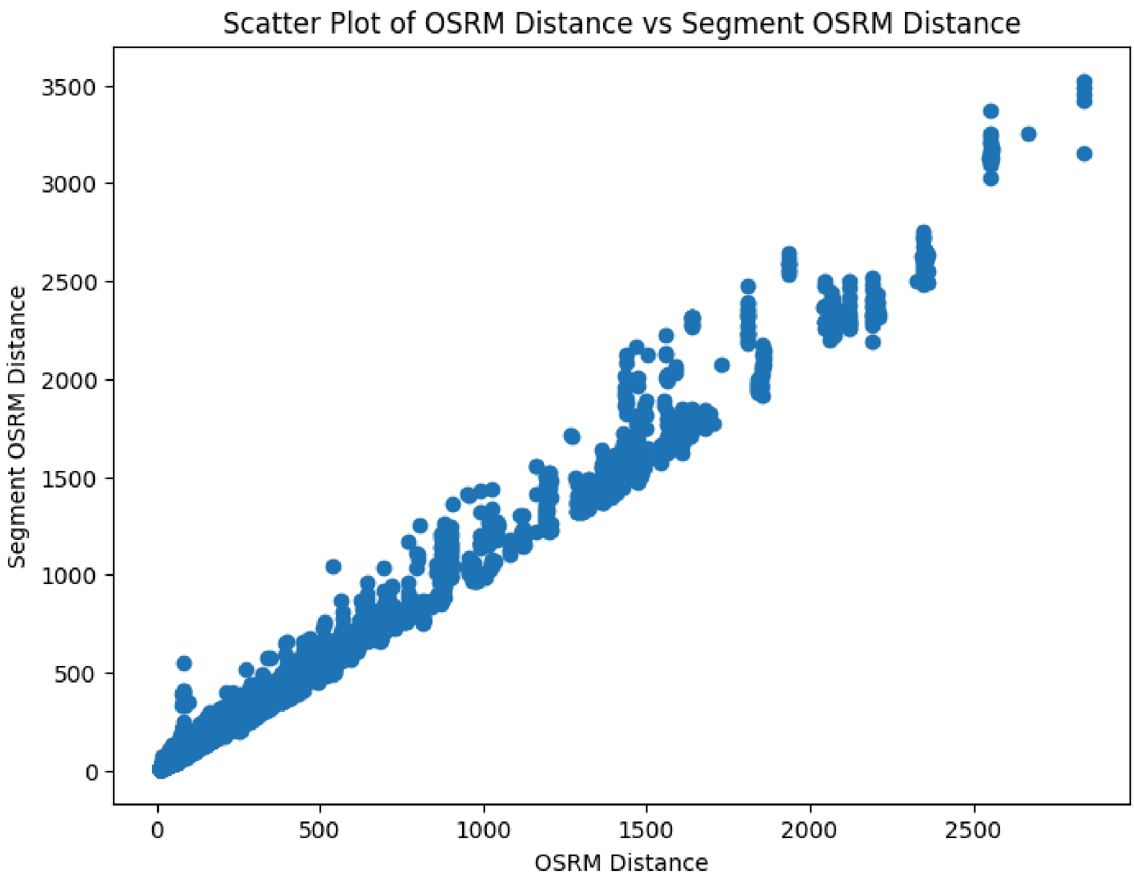
In [ ]:
```
t_stat, p_value = stats.ttest_rel(trip['osrm_distance'], trip['segment_osrm
_distance_sum'])

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

alpha=0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference be
tween the osrm_distance and segment_osrm_distance.")
else:
    print("Fail to reject the null hypothesis. There is no significant diff
erence between the osrm_distance and segment_osrm_distance.")
```

```
T-statistic: -37.24135017557747
P-value: 3.046090802811484e-290
Reject the null hypothesis. There is a significant difference between the
osrm_distance and segment_osrm_distance.
```

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(trip['osrm_distance'], trip['segment_osrm_distance_sum'])
plt.xlabel('OSRM Distance')
plt.ylabel('Segment OSRM Distance')
plt.title('Scatter Plot of OSRM Distance vs Segment OSRM Distance')
plt.show()
```



d. OSRM time aggregated value and segment OSRM time aggregated value.

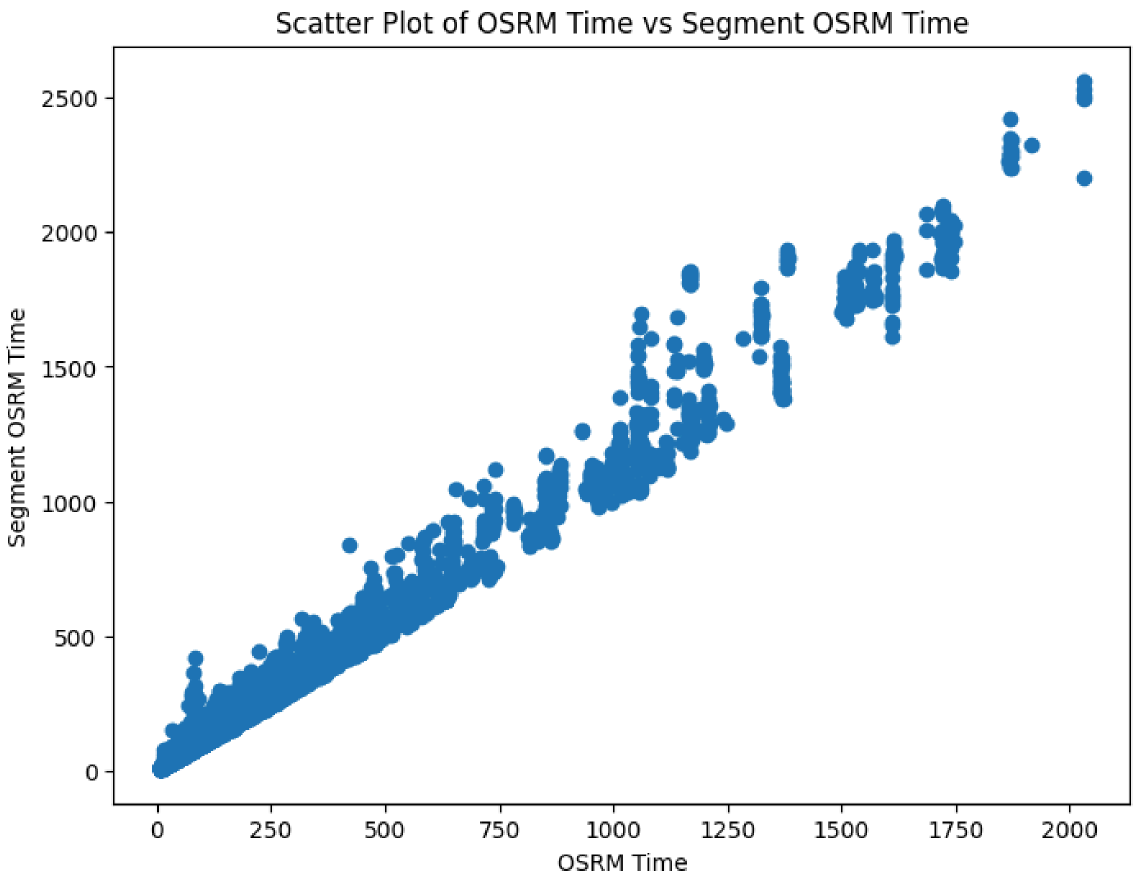
In [ ]:
```
t_stat, p_value = stats.ttest_rel(trip['osrm_time'], trip['segment_osrm_tim
e_sum'])

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

alpha=0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference be
tween the osrm_time and segment_osrm_time.")
else:
    print("Fail to reject the null hypothesis. There is no significant diff
erence between the osrm_time and segment_osrm_time.")
```

```
T-statistic: -43.20294257463631
P-value: 0.0
Reject the null hypothesis. There is a significant difference between the
osrm_time and segment_osrm_time.
```

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(trip['osrm_time'], trip['segment_osrm_time_sum'])
plt.xlabel('OSRM Time')
plt.ylabel('Segment OSRM Time')
plt.title('Scatter Plot of OSRM Time vs Segment OSRM Time')
plt.show()
```



In [ ]:
```
numerical_columns = ['od_time_diff_hour', 'start_scan_to_end_scan', 'actual_distance_to_destination',
                    'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time_sum',
                    'segment_osrm_time_sum', 'segment_osrm_distance_sum']
trip[numerical_columns].describe().T
```

Out[ ]:

	count	mean	std	min	25%	
od_time_diff_hour	14787.0	8.838559	10.973591	0.391024	2.494975	4.66
start_scan_to_end_scan	14787.0	529.429025	658.254936	23.000000	149.000000	279.00
actual_distance_to_destination	14787.0	164.090196	305.502982	9.002461	22.777099	48.28
actual_time	14787.0	356.306012	561.517936	9.000000	67.000000	148.00
osrm_time	14787.0	160.990938	271.459495	6.000000	29.000000	60.00
osrm_distance	14787.0	203.887411	370.565564	9.072900	30.756900	65.30
segment_actual_time_sum	14787.0	353.059174	556.365911	9.000000	66.000000	147.00
segment_osrm_time_sum	14787.0	180.511598	314.679279	6.000000	30.000000	65.00
segment_osrm_distance_sum	14787.0	222.705466	416.846279	9.072900	32.578850	69.78

(4)2. Outlier Detection & Treatment

```
a. Find any existing outliers in numerical features.
b. Visualize the outlier values using Boxplot.
c. Handle the outliers using the IQR method.
```

1. Perform one-hot encoding on categorical features.
2. Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

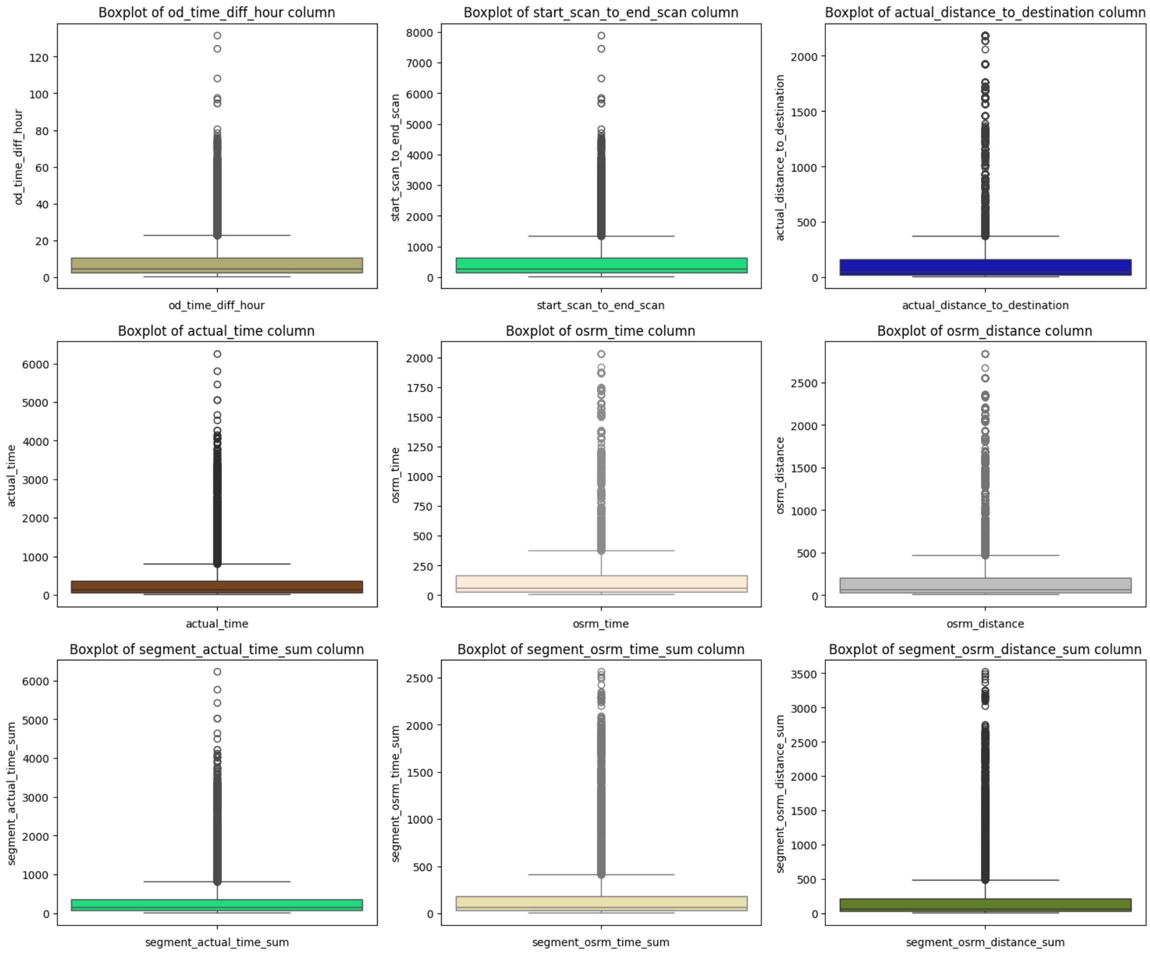
In [ ]:
```
for i in numerical_columns:
    Q1 = np.quantile(trip[i], 0.25)
    Q3 = np.quantile(trip[i], 0.75)
    IQR = Q3 - Q1
    LB = Q1 - 1.5 * IQR
    UB = Q3 + 1.5 * IQR
    outliers = trip.loc[(trip[i] < LB) | (trip[i] > UB)]
    print('Column :', i)
    print(f'Q1 : {Q1}')
    print(f'Q3 : {Q3}')
    print(f'IQR : {IQR}')
    print(f'LB : {LB}')
    print(f'UB : {UB}')
    print(f'Number of outliers : {outliers.shape[0]}')
    print('----------------------------------')
```

```
Column : od_time_diff_hour
Q1 : 2.494974930972222
Q3 : 10.558961618055555
IQR : 8.063986687083332
LB : -9.601005099652777
UB : 22.654941648680555
Number of outliers : 1275
----------------------------------
Column : start_scan_to_end_scan
Q1 : 149.0
Q3 : 632.0
IQR : 483.0
LB : -575.5
UB : 1356.5
Number of outliers : 1282
----------------------------------
Column : actual_distance_to_destination
Q1 : 22.777098943155323
Q3 : 163.5912581579725
IQR : 140.81415921481718
LB : -188.44413987907043
UB : 374.81249698019826
Number of outliers : 1452
----------------------------------
Column : actual_time
Q1 : 67.0
Q3 : 367.0
IQR : 300.0
LB : -383.0
UB : 817.0
Number of outliers : 1646
----------------------------------
Column : osrm_time
Q1 : 29.0
Q3 : 168.0
IQR : 139.0
LB : -179.5
UB : 376.5
Number of outliers : 1506
----------------------------------
Column : osrm_distance
Q1 : 30.7569
Q3 : 206.6442
IQR : 175.8873
LB : -233.07405000000003
UB : 470.47515000000004
Number of outliers : 1522
----------------------------------
Column : segment_actual_time_sum
Q1 : 66.0
Q3 : 364.0
IQR : 298.0
LB : -381.0
UB : 811.0
Number of outliers : 1644
----------------------------------
Column : segment_osrm_time_sum
Q1 : 30.0
Q3 : 184.0
IQR : 154.0
LB : -201.0
```

```
UB : 415.0
Number of outliers : 1485
----------------------------------
Column : segment_osrm_distance_sum
Q1 : 32.57885
Q3 : 216.5606
IQR : 183.98174999999998
LB : -243.393775
UB : 492.533225
Number of outliers : 1550
----------------------------------
```

In [ ]:

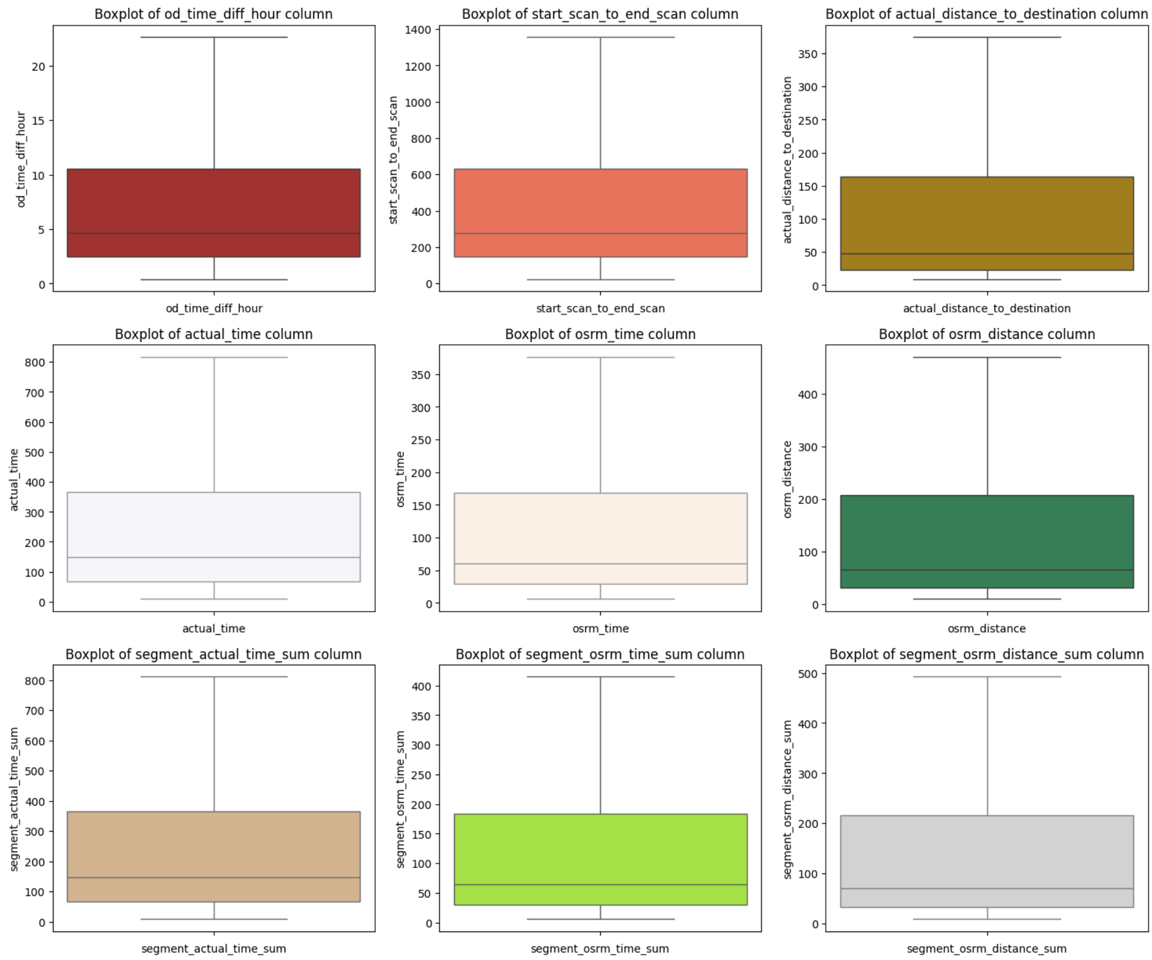
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.boxplot(trip[numerical_columns[i]],color = clr)
    plt.xlabel(numerical_columns[i])
    plt.title(f"Boxplot of {numerical_columns[i]} column")
    plt.plot()
```



In [ ]:
```
def handle_outliers_iqr(df, column):
  Q1 = np.quantile(df[column], 0.25)
  Q3 = np.quantile(df[column], 0.75)
  IQR = Q3 - Q1
  LB = Q1 - 1.5 * IQR
  UB = Q3 + 1.5 * IQR
  df[column] = np.where(df[column] < LB, LB, df[column])
  df[column] = np.where(df[column] > UB, UB, df[column])
  return df

for column in numerical_columns:
  trip = handle_outliers_iqr(trip, column)
```

In [ ]:
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.boxplot(trip[numerical_columns[i]],color=clr)
    plt.xlabel(numerical_columns[i])
    plt.title(f"Boxplot of {numerical_columns[i]} column")
    plt.plot()
```



Do one-hot encoding of categorical variables (like route_type)

In [ ]:
```
# Get value counts before one-hot encoding

trip['route_type'].value_counts()
```

Out[ ]:

	count
route_type	
Carting	8906
FTL	5881

dtype: int64

In [ ]:
```
# Perform one-hot encoding on categorical column route type
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
trip['route_type'] = label_encoder.fit_transform(trip['route_type'])
```

In [ ]:
```
# Get value counts after one-hot encoding

trip['route_type'].value_counts()
```

Out[ ]:

	count
route_type	
0	8906
1	5881

dtype: int64

In [ ]:
```
# Get value counts of categorical variable 'data' before one-hot encoding

trip['data'].value_counts()
```

Out[ ]:

	count
data	
training	10645
test	4142

dtype: int64

In [ ]:
```
# Perform one-hot encoding on categorical variable 'data'
label_encoder = LabelEncoder()
trip['data'] = label_encoder.fit_transform(trip['data'])
```

In [ ]:
```
# Get value counts after one-hot encoding

trip['data'].value_counts()
```

Out[ ]:

data	count
1	10645
0	4142

dtype: int64

In [ ]:
```
trip_num=trip[numerical_columns]
```

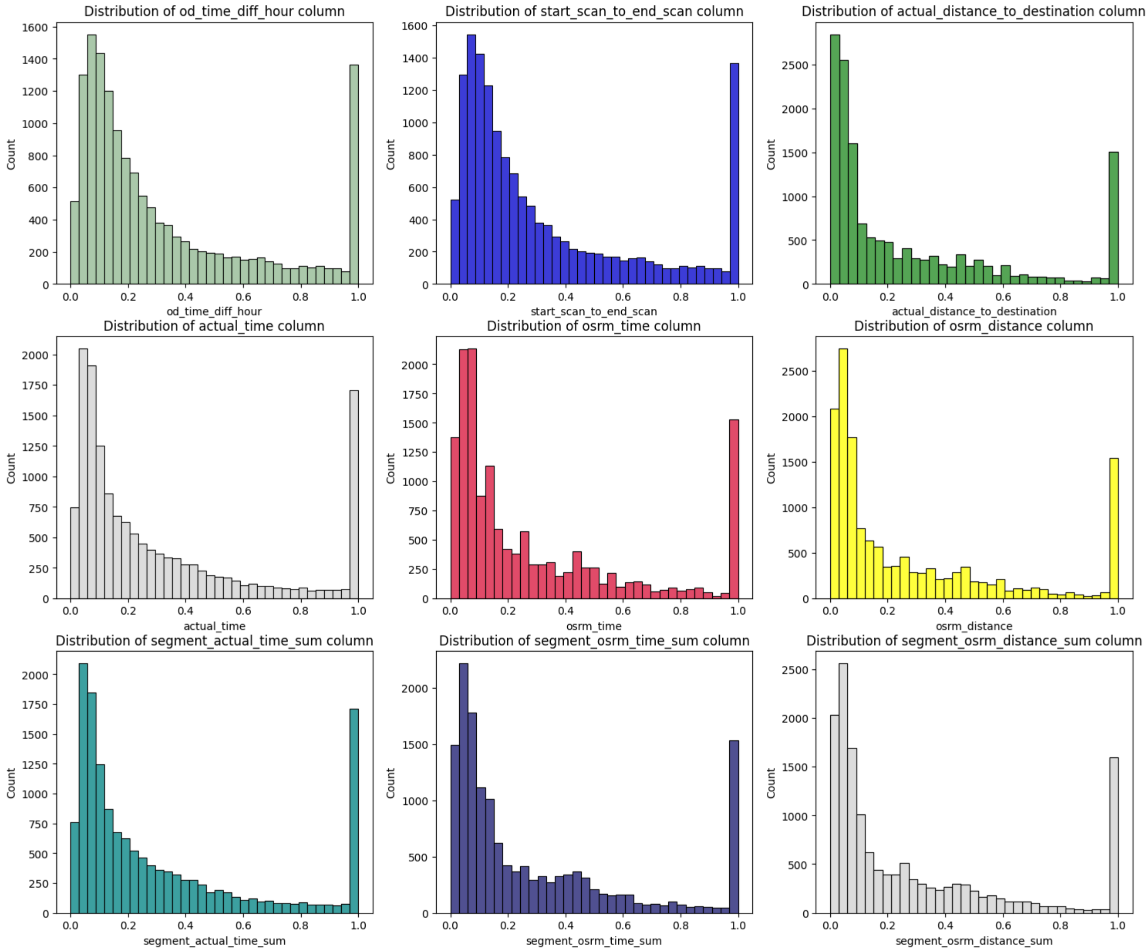
Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler


# Choose a scaler (MinMaxScaler or StandardScaler)
scaler = MinMaxScaler()  # Or StandardScaler()

# Fit and transform the numerical features
trip_num[numerical_columns] = scaler.fit_transform(trip[numerical_columns])
```

In [ ]:
```
plt.figure(figsize = (18, 15))
for i in range(len(numerical_columns)):
    plt.subplot(3, 3, i + 1)
    clr = np.random.choice(list(mpl.colors.cnames))
    sns.histplot(trip_num[numerical_columns[i]], color = clr)
    plt.title(f"Distribution of {numerical_columns[i]} column")
    plt.plot()
```



Business Insights

- There are about 14817 unique trip IDs, 1508 unique source centers, 1481 unique destination_centers, 731 unique source cities, 856 unique destination cities.
- Most orders are sourced from the states like Maharashtra, Karnataka, Haryana, Tamil Nadu, Telangana
- Maximum number of trips originated from Gurgaon city followed by Bengaluru,Bhiwandi and Mumbai. That means that the seller base is strong in these cities.
- Maximum number of trips ended in Maharashtra state followed by Karnataka, Haryana, Tamil Nadu and Uttar Pradesh. That means that the number of orders placed in these states is significantly high.
- Maximum number of trips ended in Bengaluru city followed by Mumbai, Gurgaon, Delhi and Bangalore. That means that the number of orders placed in these cities is significantly high.
- Most orders in terms of destination are coming from cities like bengaluru, mumbai, gurgaon, bangalore, Delhi.
- Features actual_time & osrm_time are statitically different.
- Features actual_time and segment actual time are statistically different from each other.
- Features osrm_distance and segment_osrm_distance are statistically different from each other.
- Both the osrm_time & segment_osrm_time are not statistically same.

Recommendations

- The OSRM trip planning system needs to be improved. Discrepancies need to be catered to for transporters, if the routing engine is configured for optimum results.
- osrm_time and actual_time are different. Team needs to make sure this difference is reduced, so that better delivery time prediction can be made and it becomes convenient for the customer to expect an accurate delivery time.
- The osrm distance and actual distance covered are also not same i.e. maybe the delivery person is not following the predefined route which may lead to late deliveries or the osrm devices is not properly predicting the route based on distance, traffic and other factors. Team needs to look into it.
- Most of the orders are coming from/reaching to states like Maharashtra, Karnataka, Haryana and Tamil Nadu. The existing corridors can be further enhanced to improve the penetration in these areas.
- Customer profiling of the customers belonging to the states Maharashtra, Karnataka, Haryana, Tamil Nadu and Uttar Pradesh has to be done to get to know why major orders are coming from these states and to improve customers' buying and delivery experience.
- From state point of view, we might have very heavy traffic in certain states and bad terrain conditions in certain states. This will be a good indicator to plan and cater to demand during peak festival seasons.