Step 1. Defining Problem Statement.

Problem statement Introduction:

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

Business Problem:

Delhivery, a leading logistics company in India, faces challenges in maximizing the value of its extensive data resources.

The key issues are:

1. Efficiently cleaning and transforming raw data to extract useful features.
2. Supporting the data science team with well-processed data to develop reliable forecasting models.

Addressing these issues will enhance the company's operational efficiency, competitiveness, and profitability.

1. Basic data cleaning and exploration:

In [582…

```
#importing required libraries.
import pandas as pd
import numpy as np
```

In [583…

```
#Loading dataset dataset.
df=pd.read_csv("/content/delhivery_data.csv")
```

In [584…

```
df.head()
```

Out[584]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121/
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121/
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121/
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121/
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121/

5 rows × 24 columns

In [585…

```
df.shape
```

Out[585]:

```
(144867, 24)
```

In [586…

```
df.columns
```

Out[586]:

```
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')
```

In [587…

```
df.dtypes
```

Out[587]:
```
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 [588…
```python
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
```

In [589…
```python
unknwn_columns = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segm
df.drop(columns = unknwn_columns,inplace=True)
```

In [590…
```python
df.shape
```

Out[590]:
```
(144867, 19)
```

1. Handling missing values in the data.

In [591…
```
#Checking for missing values in the dataset.
df.isna().sum()
```

Out[591]:
```
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 [592…
```
source_name_missing = df.loc[df['source_name'].isnull(), 'source_center'].unique()
print(source_name_missing)
```

```
['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND509103AAC' 'IND126116AAA' 'IND331022A1B'
 'IND505326AAB' 'IND852118A1B']
```

In [593…
```
destination_name_missing = df.loc[df['destination_name'].isnull(), 'destination_cer
print(destination_name_missing)
```

```
['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND505326AAB' 'IND852118A1B' 'IND126116AAA'
 'IND509103AAC' 'IND221005A1A' 'IND250002AAC' 'IND331001A1C'
 'IND122015AAC']
```

In [594…
```
count = 1

# Replace missing destination_name based on destination_center
for i in destination_name_missing:
    df.loc[df['destination_center'] == i, 'destination_name'] = df.loc[df['destinat
    count += 1
```

In [595…
```
# Replace missing source_name based on source_center using a dictionary
d = {}
for i in source_name_missing:
    d[i] = df.loc[df['source_center'] == i, 'source_name'].dropna().unique()

# Handle cases where no unique values are found for missing source_name
for idx, val in d.items():
    if len(val) == 0:
        d[idx] = [f'location_{count}']
        count += 1
```

```
# Map source_center to the corresponding destination_name
d2 = {k: v[0] for k, v in d.items()}
```

In [596…

```
# Replace missing source_name using the mapped values in d2
for i in source_name_missing:
    df.loc[df['source_center'] == i, 'source_name'] = df.loc[df['source_center'] ==
```

In [597…

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

Out[597]:

```
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
```

2. Converting time columns into pandas datetime.

In [598…

```
#Converting time columns into pandas datetime.
df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'])
df['od_start_time'] = pd.to_datetime(df['od_start_time'])
df['od_end_time'] = pd.to_datetime(df['od_end_time'])
```

In [599…

```
#Converting data type of categorical columns
df['route_type'] = df['route_type'].astype('category')
```

3. Analyze structure & characteristics of the dataset.

In [600…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144867 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144867 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: category(1), datetime64[ns](3), float64(8), object(7)
memory usage: 20.0+ MB
```

In [601…

```
df.describe()
```

Out[601]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	actual_dis
count	144867	144867	144867	144867.000000	
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.262986	
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 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000	
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.000000	
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000	
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	1037.012769	

2. Try merging the rows using the hint mentioned below.

1. Grouping by segment

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

In [602…

```
df['segment_key'] = df['trip_uuid'] + df['source_center'] + df['destination_center'
```

b. Using inbuilt functions like groupby and aggregations like cumsum() to merge the rows in columns segment_actual_time,segment_osrm_distance, segment_osrm_time based on the segment_key.

In [603…

```
segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time'

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

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

Out[603]:

	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
...	...	...	...
144862	92.0	65.3487	94.0
144863	118.0	82.7212	115.0
144864	138.0	103.4265	149.0
144865	155.0	122.3150	176.0
144866	423.0	131.1238	185.0

144867 rows × 3 columns

In [604…

```
df.head()
```

Out[604]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121

5 rows × 23 columns

2. Aggregating at segment level

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

i. keeping the first and last values for some numeric/categorical fields if aggregating them won’t make sense.

In [605…

```
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',

    }
```

b & c. Grouping the data by segment_key because you want to perform aggregation operations for different segments of each trip based on the segment_key value & The aggregation functions specified in the create_segment_dict are applied to each group of rows with the same segment_key.

In [606…

```
segment = df.groupby('segment_key').agg(create_segment_dict).reset_index()
```

d. Sorting the resulting DataFrame segment, by two criteria:

i. First, sorting it by segment_key to ensure that segments are ordered consistently.

ii. Second, sorting it by od_end_time in ascending order, ensuring that segments within the same trip are ordered by their end times from earliest to latest.

In [607…

```
segment = segment.sort_values(by=['segment_key','od_end_time'], ascending=True).res
```

In [608…

```
segment
```

Out[608]:

	index	segment_key	data	trip_creation_time	rou
0	0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos:
1	1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos:
2	2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::
3	3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	thanos::
4	4	trip-153671043369099517IND000000ACBIND160002AAC	training	2018-09-12 00:00:33.691250	thanos:
...	...	...	...	...	
26363	26363	trip-153861115439069069IND628204AAAIND627657AAA	test	2018-10-03 23:59:14.390954	thanos
26364	26364	trip-153861115439069069IND628613AAAIND627005AAA	test	2018-10-03 23:59:14.390954	thanos
26365	26365	trip-153861115439069069IND628801AAAIND628204AAA	test	2018-10-03 23:59:14.390954	thanos
26366	26366	trip-153861118270144424IND583119AAAIND583101AAA	test	2018-10-03 23:59:42.701692	thanos
26367	26367	trip-153861118270144424IND583201AAAIND583119AAA	test	2018-10-03 23:59:42.701692	thanos

26368 rows × 21 columns

3. Feature Engineering:

1. Calculating time taken between od_start_time and od_end_time and keeping it as a feature named od_time_diff_hour.

In [609…

```
segment['od_time_diff_hour'] = (segment['od_end_time'] - segment['od_start_time']).
segment['od_time_diff_hour']
```

Out[609]:
```
0        1260.604421
1         999.505379
2          58.832388
3         122.779486
4         834.638929
            ...
26363      62.115193
26364      91.087797
26365      44.174403
26366     287.474007
26367      66.933565
Name: od_time_diff_hour, Length: 26368, dtype: float64
```

2. Grouping and Aggregating at Trip-level

Creating create_trip_dict dictionary.

In [610…
```
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',

    }
```

b. Grouping the segment data by the trip_uuid column to focus on aggregating data at the trip level.

In [611…
```
trip = segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop = True)
```

In [612…
```
trip[['actual_time', 'segment_actual_time_sum']]
```

Out[612]:

	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
...	...	...
14812	83.0	82.0
14813	21.0	21.0
14814	282.0	281.0
14815	264.0	258.0
14816	275.0	274.0

14817 rows × 2 columns

In [613…

```
trip
```

Out[613]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	sour
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND2(
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND5(
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND0(
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND4(
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND5ᵋ
...	...	...	...	...	...	
14812	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND1(
14813	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND1:
14814	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND2(
14815	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND6:
14816	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND5ᵋ

14817 rows × 18 columns

3. Destination Name: Split and extract features out of destination. City-place-code(State)

In [614...

```
trip['destination_name'] = trip['destination_name'].str.lower() #Lowering all colum
trip['source_name'] = trip['source_name'].str.lower()
```

In [615...

```
def place2state(x):
    # transform  "gurgaon_bilaspur_hb (haryana)" into "haryana)""
    state = x.split('(')
    if len(state)==1:
      return state[0]
    else:
      return state[1].replace(')', "") #removing ')' from ending

def place2city(x):
```

```
    # We will remove state
    city = x.split(' (')[0]

    city = city.split('_')[0]

    #Now dealing with edge cases

    if city == 'pnq vadgaon sheri dpc':
      return 'vadgaonsheri'

    # ['PNQ Pashan DPC', 'Bhopal MP Nagar', 'HBR Layout PC',
    #  'PNQ Rahatani DPC', 'Pune Balaji Nagar', 'Mumbai Antop Hill']

    if city in ['pnq pashan dpc','pnq rahatani dpc', 'pune balaji nagar']:
        return 'pune'

    if city == 'hbr layout pc' : return 'bengaluru'
    if city == 'bhopal mp nagar' : return 'bhopal'
    if city == 'mumbai antop hill' : return 'mumbai'


    return city

def place2city_place(x):

    # We will remove state
    x = x.split(' (')[0]

    len_ = len(x.split('_'))

    if len_ >= 3:
        return x.split('_')[1]

    # Small cities have same city and place name
    if len_ == 2:
        return x.split('_')[0]


    # Now we need to deal with edge cases or imporper name convention

    #if len(x.split(' ')) == 2:
    #

    return x.split(' ')[0]


def place2code(x):
    # We will remove state
    x = x.split(' (')[0]

    if len(x.split('_')) >= 3 :
        return x.split('_')[-1]

    return 'none'
```

In [616…

```
trip['destination_state'] = trip['destination_name'].apply(lambda x: place2state(x)
trip['destination_city']  = trip['destination_name'].apply(lambda x: place2city(x))
trip['destination_place'] = trip['destination_name'].apply(lambda x: place2city_pla
trip['destination_code']  = trip['destination_name'].apply(lambda x: place2code(x))
```

In [617…

```
trip[['destination_state', 'destination_city', 'destination_place', 'destination_cc
```

Out[617]:

	destination_state	destination_city	destination_place	destination_code
0	uttar pradesh	kanpur	central	6
1	karnataka	doddablpur	chikadpp	d
2	haryana	gurgaon	bilaspur	hb
3	maharashtra	mumbai	mirard	ip
4	karnataka	sandur	wrdn1dpp	d
...	...	...	...	...
14812	punjab	chandigarh	mehmdpur	h
14813	haryana	faridabad	blbgarh	dc
14814	uttar pradesh	kanpur	govndngr	dc
14815	tamil nadu	tirchchndr	shnmgprm	d
14816	karnataka	sandur	wrdn1dpp	d

14817 rows × 4 columns

4. Source Name: Split and extract features out of destination. City-place-code (State)

In [618…

```
trip['source_state'] = trip['source_name'].apply(lambda x: place2state(x))
trip['source_city']  = trip['source_name'].apply(lambda x: place2city(x))
trip['source_place'] = trip['source_name'].apply(lambda x: place2city_place(x))
trip['source_code']  = trip['source_name'].apply(lambda x: place2code(x))
```

In [619…

```
trip[['source_state', 'source_city', 'source_place', 'source_code']]
```

Out[619]:

	source_state	source_city	source_place	source_code
0	uttar pradesh	kanpur	central	6
1	karnataka	doddablpur	chikadpp	d
2	haryana	gurgaon	bilaspur	hb
3	maharashtra	mumbai hub	mumbai	none
4	karnataka	bellary	bellary	none
...	...	...	...	...
14812	punjab	chandigarh	mehmdpur	h
14813	haryana	fbd	balabhgarh	dpc
14814	uttar pradesh	kanpur	govndngr	dc
14815	tamil nadu	tirunelveli	vdkkusrt	i
14816	karnataka	sandur	wrdn1dpp	d

14817 rows × 4 columns

5. Trip_creation_time: Extract features like month, year, day, etc.

In [620…

```
trip['trip_year'] = trip['trip_creation_time'].dt.year
trip['trip_month'] = trip['trip_creation_time'].dt.month
trip['trip_hour'] = trip['trip_creation_time'].dt.hour
trip['trip_day'] = trip['trip_creation_time'].dt.day
trip['trip_week'] = trip['trip_creation_time'].dt.isocalendar().week
trip['trip_dayofweek'] = trip['trip_creation_time'].dt.dayofweek
```

In [621…

```
trip[['trip_year', 'trip_month', 'trip_hour', 'trip_day', 'trip_week', 'trip_dayofw
```

Out[621]:

	trip_year	trip_month	trip_hour	trip_day	trip_week	trip_dayofweek
0	2018	9	0	12	37	2
1	2018	9	0	12	37	2
2	2018	9	0	12	37	2
3	2018	9	0	12	37	2
4	2018	9	0	12	37	2
...	...	...	...	...	...	...
14812	2018	10	23	3	40	2
14813	2018	10	23	3	40	2
14814	2018	10	23	3	40	2
14815	2018	10	23	3	40	2
14816	2018	10	23	3	40	2

14817 rows × 6 columns

In [622…

```
trip.describe().T
```

Out[622]:

	count	mean	min	25%
trip_creation_time	14817	2018-09-22 12:44:19.555167744	2018-09-12 00:00:16.535741	2018-09-17 02:51:25.129125888
start_scan_to_end_scan	14817.0	530.810016	23.0	149.0
od_time_diff_hour	14817.0	531.697682	23.461468	149.930591
actual_distance_to_destination	14817.0	164.477838	9.002461	22.837239
actual_time	14817.0	357.143754	9.0	67.0
osrm_time	14817.0	161.384018	6.0	29.0
osrm_distance	14817.0	204.344689	9.0729	30.8192
segment_actual_time_sum	14817.0	353.892286	9.0	66.0
segment_osrm_distance_sum	14817.0	223.201161	9.0729	32.6545
segment_osrm_time_sum	14817.0	180.949787	6.0	31.0
trip_year	14817.0	2018.0	2018.0	2018.0
trip_month	14817.0	9.120672	9.0	9.0
trip_hour	14817.0	12.449821	0.0	4.0
trip_day	14817.0	18.37079	1.0	14.0
trip_week	14817.0	38.295944	37.0	38.0
trip_dayofweek	14817.0	2.919349	0.0	1.0

4. In-depth analysis:

1. Outlier Detection & Treatment

a. Finding any existing outliers in numerical features.

In [623…

```
trip.dtypes
```

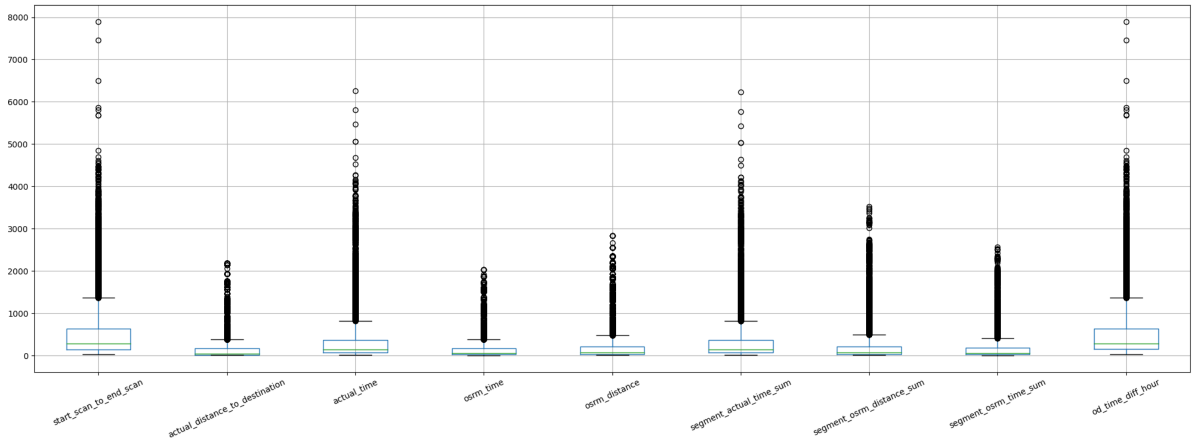
Out[623]:
```
data                                    object
trip_creation_time              datetime64[ns]
route_schedule_uuid                     object
route_type                            category
trip_uuid                               object
source_center                           object
source_name                             object
destination_center                      object
destination_name                        object
start_scan_to_end_scan                 float64
od_time_diff_hour                      float64
actual_distance_to_destination         float64
actual_time                            float64
osrm_time                              float64
osrm_distance                          float64
segment_actual_time_sum                float64
segment_osrm_distance_sum              float64
segment_osrm_time_sum                  float64
destination_state                       object
destination_city                        object
destination_place                       object
destination_code                        object
source_state                            object
source_city                             object
source_place                            object
source_code                             object
trip_year                                int32
trip_month                               int32
trip_hour                                int32
trip_day                                 int32
trip_week                               UInt32
trip_dayofweek                           int32
dtype: object
```

In [624…
```
num_cols = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_tim
            'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_distance_sum'
            'segment_osrm_time_sum', 'od_time_diff_hour']
```

b. Visualizing the outlier values using Boxplot.

In [625…
```
trip[num_cols].boxplot(rot=25, figsize=(25,8))
```

Out[625]: <Axes: >



c. Handling the outliers using the IQR method.

In [626…
```
Q1 = trip[num_cols].quantile(0.25)
Q3 = trip[num_cols].quantile(0.75)

IQR = Q3 - Q1
```

In [627…
```
trip = trip[~((trip[num_cols] < (Q1 - 1.5 * IQR)) | (trip[num_cols] > (Q3 + 1.5 * I
trip = trip.reset_index(drop=True)
```

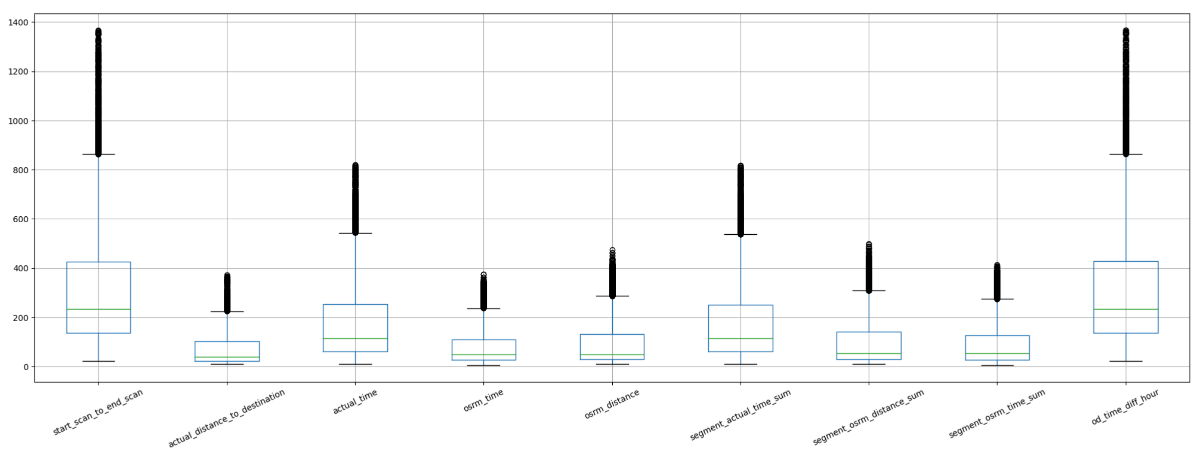
In [628…
```
trip.describe().T
```

Out[628]:

	count	mean	min	25%
trip_creation_time	12759	2018-09-22 13:31:40.651586816	2018-09-12 00:00:22.886430	2018-09-17 03:24:12.859975936
start_scan_to_end_scan	12759.0	322.025237	23.0	136.0
od_time_diff_hour	12759.0	322.872651	23.461468	136.846184
actual_distance_to_destination	12759.0	72.82579	9.002461	21.410516
actual_time	12759.0	178.556235	9.0	61.0
osrm_time	12759.0	78.977506	6.0	27.0
osrm_distance	12759.0	92.380262	9.0729	28.38
segment_actual_time_sum	12759.0	176.893487	9.0	60.0
segment_osrm_distance_sum	12759.0	98.668152	9.0729	29.4891
segment_osrm_time_sum	12759.0	86.500039	6.0	28.0
trip_year	12759.0	2018.0	2018.0	2018.0
trip_month	12759.0	9.122345	9.0	9.0
trip_hour	12759.0	12.416255	0.0	4.0
trip_day	12759.0	18.354887	1.0	14.0
trip_week	12759.0	38.301748	37.0	38.0
trip_dayofweek	12759.0	2.913003	0.0	1.0

In [629…
```
trip[num_cols].boxplot(rot=25, figsize=(25,8))
```

Out[629]: <Axes: >



2. Perform one-hot encoding on categorical features.

In [630…

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

Out[630]:

```
route_type
Carting     8817
FTL         3942
Name: count, dtype: int64
```

Insights: Most common route type is Carting.

In [631…

```
trip['route_type'] = trip['route_type'].map({'FTL':0, 'Carting':1})
```

In [632…

```
trip[num_cols]
```

Out[632]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distanc
0	180.0	73.186911	143.0	68.0	85.111
1	100.0	17.175274	59.0	15.0	19.680
2	717.0	127.448500	341.0	117.0	146.791
3	189.0	24.597048	61.0	23.0	28.064
4	98.0	9.100510	24.0	13.0	12.018
...	...	...	...	...	
12754	257.0	57.762332	83.0	62.0	73.463
12755	60.0	15.513784	21.0	12.0	16.088
12756	421.0	38.684839	282.0	48.0	58.903
12757	347.0	134.723836	264.0	179.0	171.110
12758	353.0	66.081533	275.0	68.0	80.578

12759 rows × 9 columns

In [633…

```
trip[num_cols].describe()
```

Out[633]:

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_dis
count	12759.000000	12759.000000	12759.000000	12759.000000	12759.0(
mean	322.025237	72.825790	178.556235	78.977506	92.38
std	257.404103	72.570289	159.088778	72.855650	90.1
min	23.000000	9.002461	9.000000	6.000000	9.07
25%	136.000000	21.410516	61.000000	27.000000	28.38
50%	234.000000	38.672808	115.000000	50.000000	48.7
75%	427.000000	102.959653	254.000000	111.000000	131.9(
max	1366.000000	373.441224	820.000000	376.000000	474.1

5. Hypothesis Testing:

1. Perform hypothesis testing / visual analysis between :

a. actual_time aggregated value and OSRM time aggregated value.

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

Null Hypothesis - There is no difference between actual_time and osrm_time.

Alternative Hypothesis - There is significant difference between actual_time and osrm_time

In [634…

```
trip[['actual_time', 'osrm_time']].describe()
```

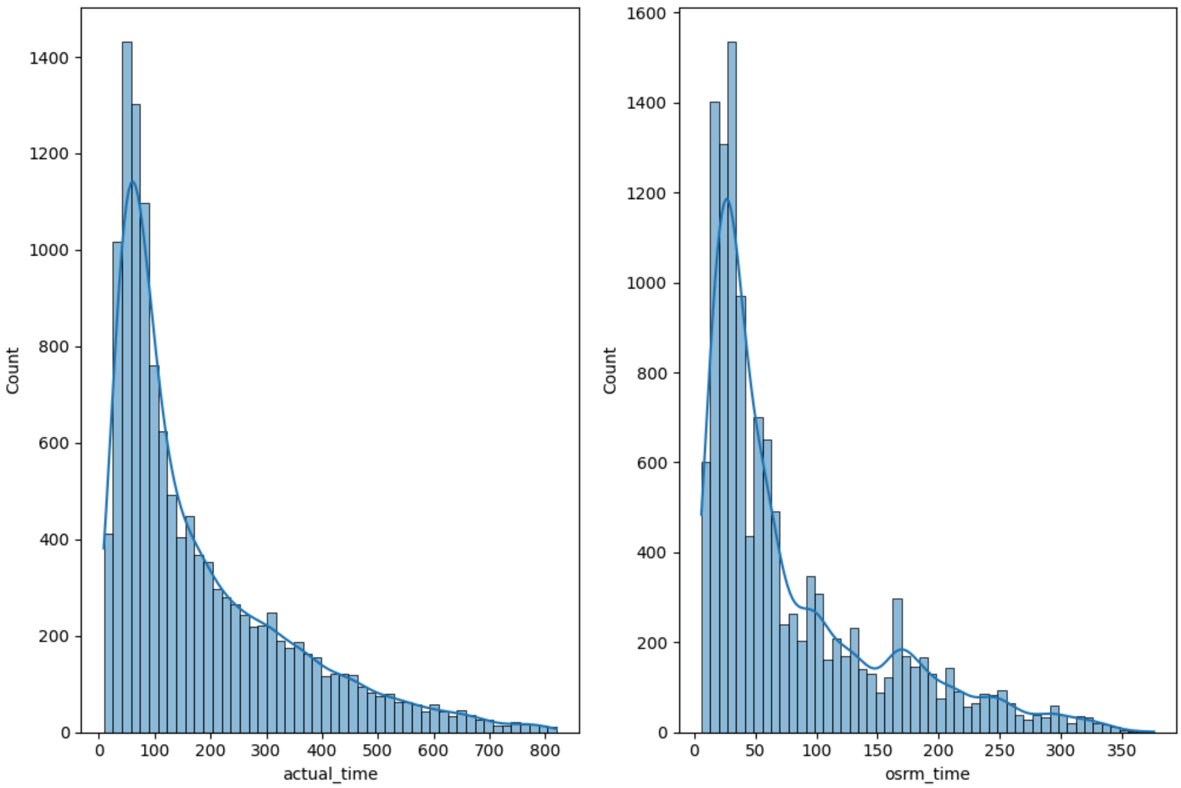
Out[634]:

	actual_time	osrm_time
count	12759.000000	12759.000000
mean	178.556235	78.977506
std	159.088778	72.855650
min	9.000000	6.000000
25%	61.000000	27.000000
50%	115.000000	50.000000
75%	254.000000	111.000000
max	820.000000	376.000000

In [635…

```
#distribution of actual time and osrm time
import matplotlib.pyplot as plt
import seaborn as sns
```

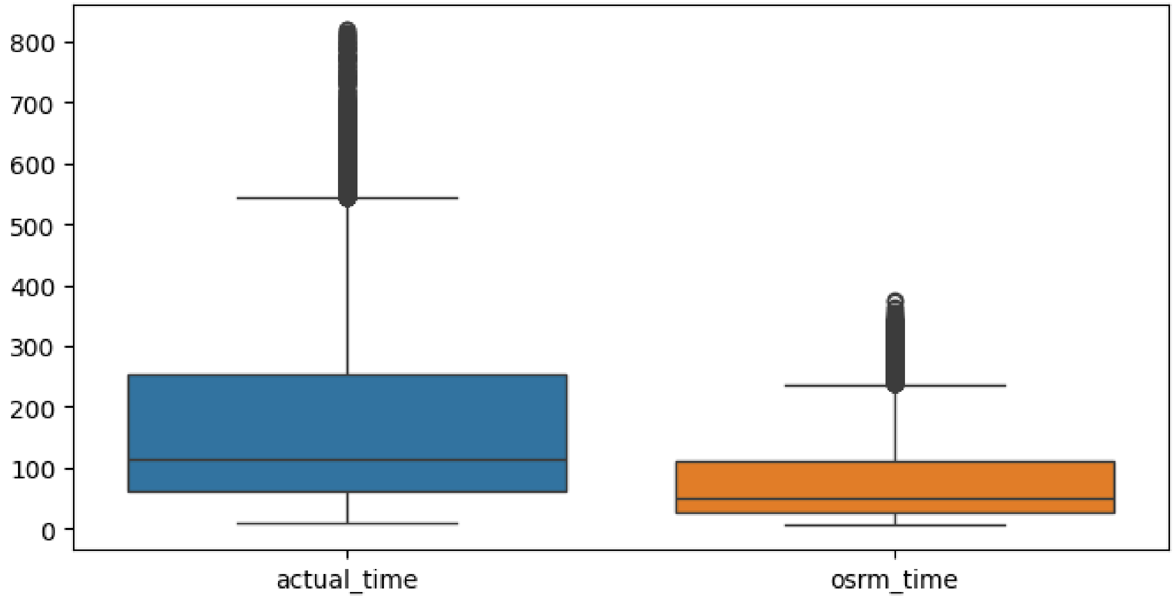
```
plt.figure(figsize = (12, 8))
plt.subplot(1,2,1)
sns.histplot(data = trip['actual_time'], kde=True)
plt.subplot(1,2,2)
sns.histplot(data=trip['osrm_time'], kde=True)
plt.show()
```



Insights:The histograms show that both actual_time and osrm_time is right skewed and not nrmally distributed

In [636…

```
plt.figure(figsize = (8, 4))
sns.boxplot(data = trip[['actual_time','osrm_time']])
plt.show()
```

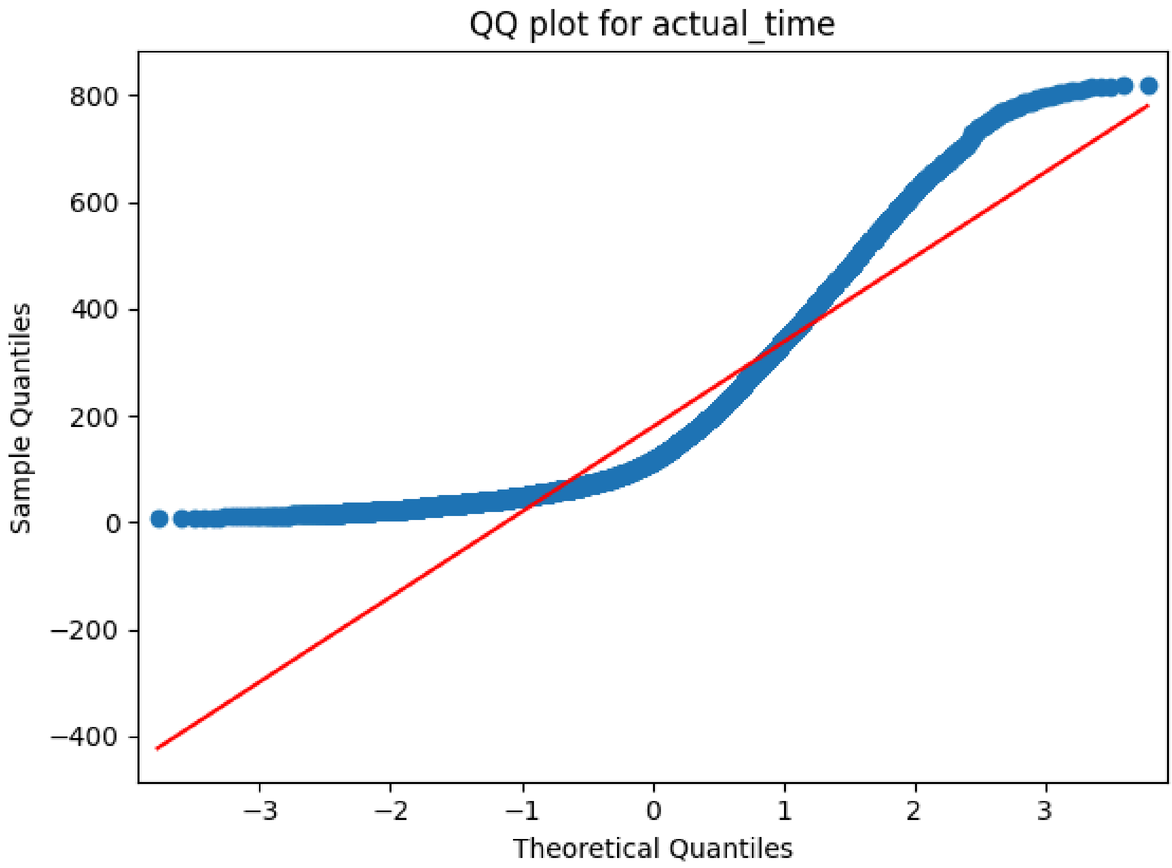


Insights: We can clearly see from the box plot that the actual time is much higher than the OSRM time.

In [637…

```
import statsmodels.api as sm
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['actual_time'], line="s")
plt.title('QQ plot for actual_time')
plt.tight_layout()
plt.show()
```

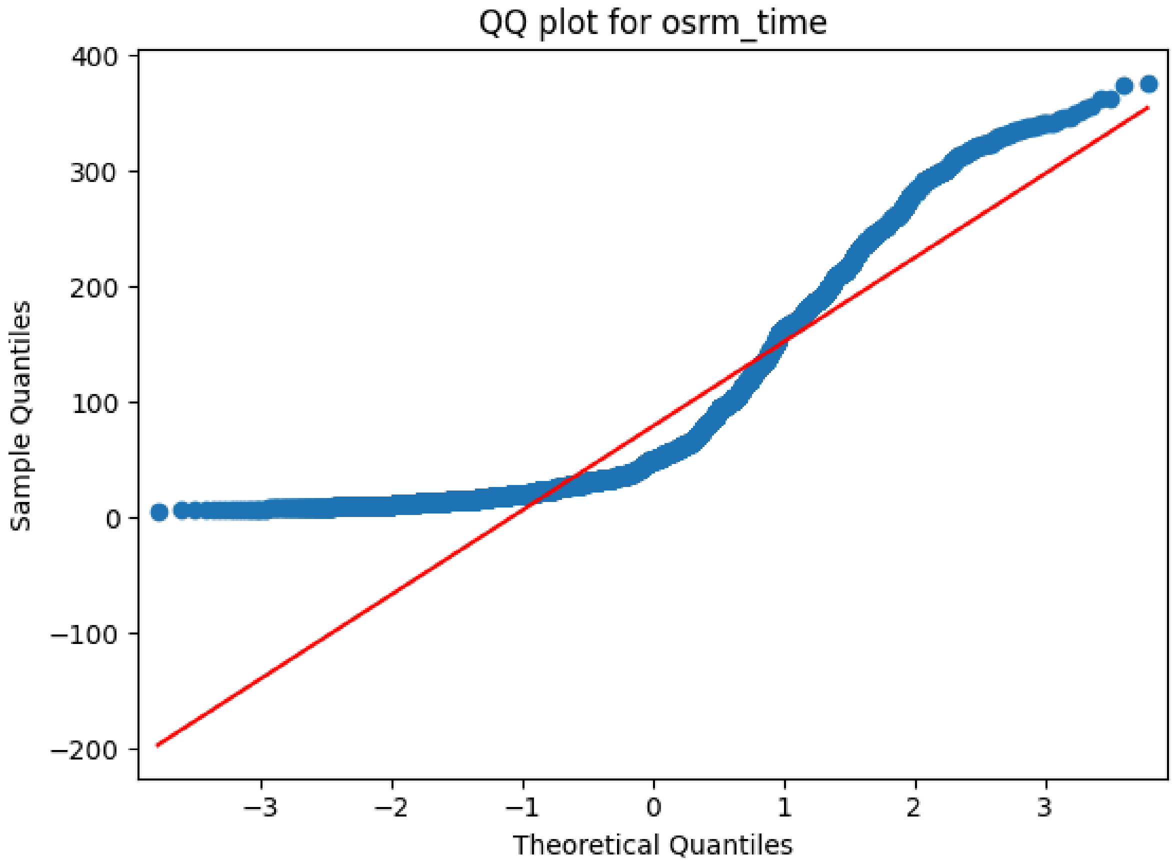
```
<Figure size 1000x400 with 0 Axes>
```



In [638…

```
plt.figure(figsize=(10, 4))
sm.qqplot(trip['osrm_time'], line="s")
plt.title('QQ plot for osrm_time')
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```



samples do not follow normal distribution

Applying Shapiro-Wilk test

Ho: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [639…

```
from scipy.stats import shapiro
test_stat, p_value = shapiro(trip['actual_time'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.729785448040269e-49
The sample does not follow normal distribution
```

In [640…

```
test_stat, p_value = shapiro(trip['osrm_time'].sample(5000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 5.465429629112036e-60
The sample does not follow normal distribution
```

Homogeneity of Variances using Lavene's test

In [641…

```
from scipy.stats import levene
test_stat, p_value = levene(trip['actual_time'], trip['osrm_time'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.0
The samples do not have  Homogenous Variance
```

As the samples do not exhibit a normal distribution, the application of the T-Test is not suitable in this context. Instead, we can utilize its non-parametric equivalent, namely the Mann-Whitney U rank test, for comparing two independent samples.

In [642…

```
from scipy.stats import mannwhitneyu

H0="There is no difference between actual_time and osrm_time."
Ha="There is significant difference between actual_time and osrm_time."
alpha = 0.05

u_stat, p_value = mannwhitneyu(trip['actual_time'], trip['osrm_time'])

print('Test Statistic:', u_stat)
print('P value:', p_value)

if p_value < alpha:
  print("Result: \nReject null hypothesis. \n", Ha)
else:
  print("Result: \nFail to reject null hypothesis. \n ", H0)
```

```
Test Statistic: 120080730.5
P value: 0.0
Result:
Reject null hypothesis.
 There is significant difference between actual_time and osrm_time.
```

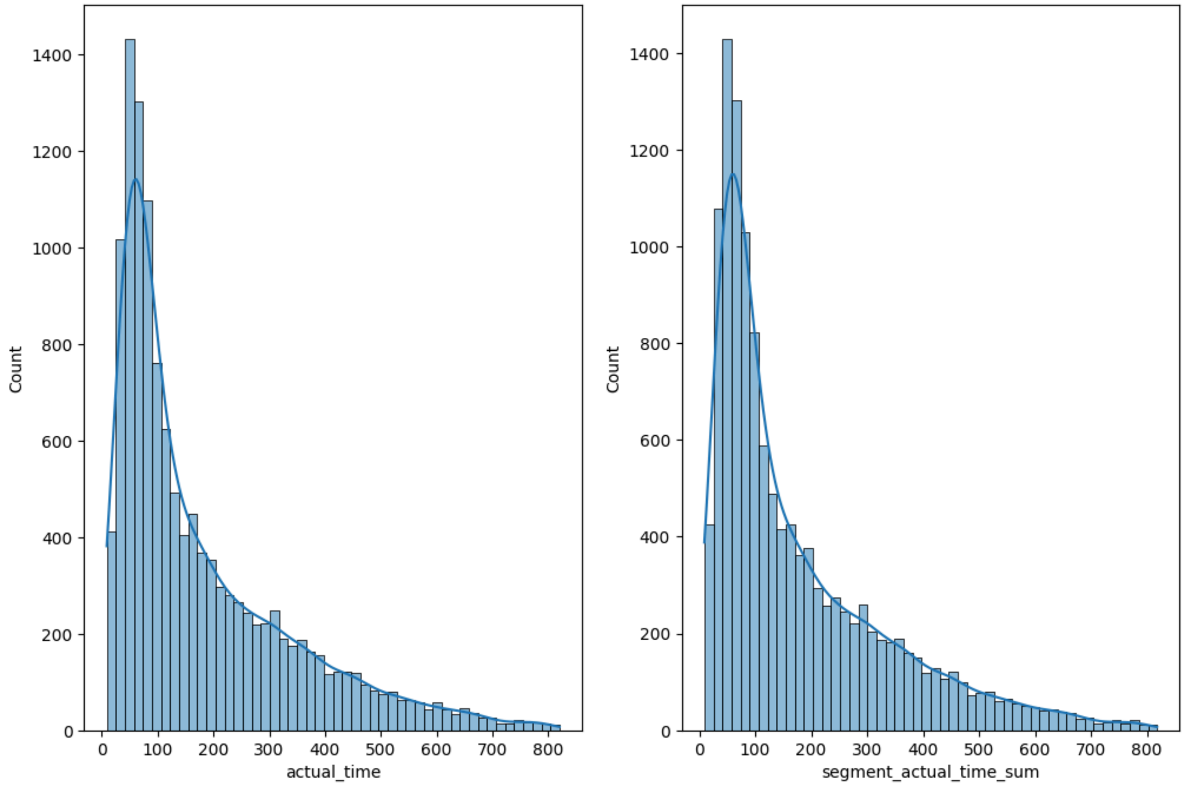
b. actual_time aggregated value and segment actual time aggregated value.

Null Hypothesis: There is no difference between actual_time and segment_actual_time

Alternative Hypothesis: There is a difference between actual_time and segment_actual_time

In [643…

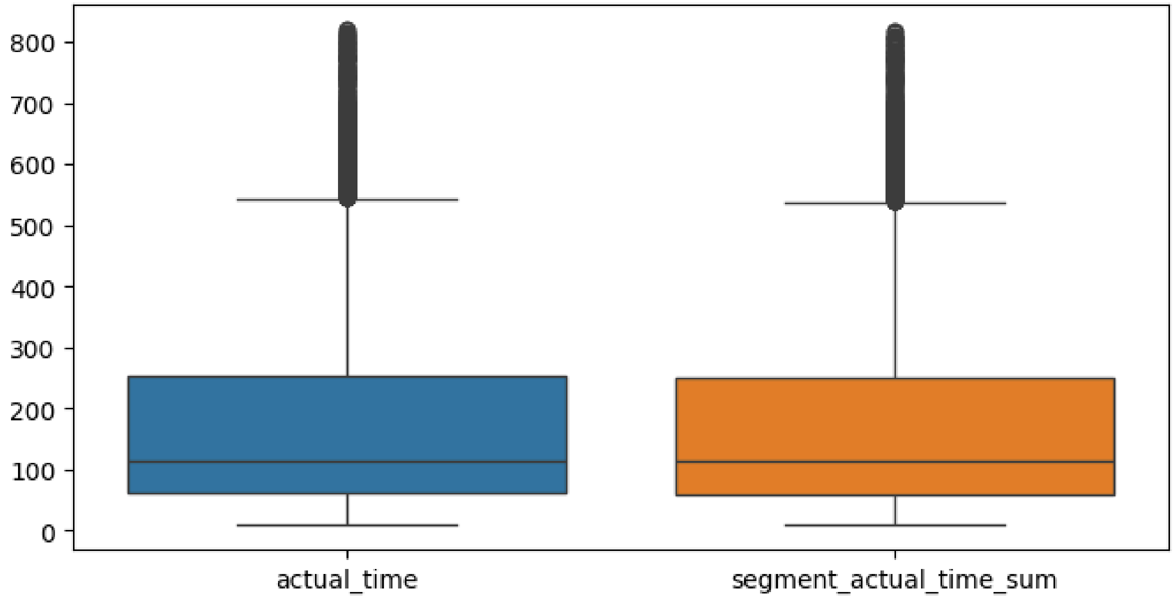
```
#distribution of actual time and segment actual time

plt.figure(figsize = (12, 8))
plt.subplot(1,2,1)
sns.histplot(data = trip['actual_time'], kde=True)
plt.subplot(1,2,2)
sns.histplot(data=trip['segment_actual_time_sum'], kde=True)
plt.show()
```



Both actual time and segment actual time are right-skewed and not normally distributed.

In [644...

```
plt.figure(figsize = (8, 4))
sns.boxplot(data = trip[['actual_time','segment_actual_time_sum']])
plt.show()
```

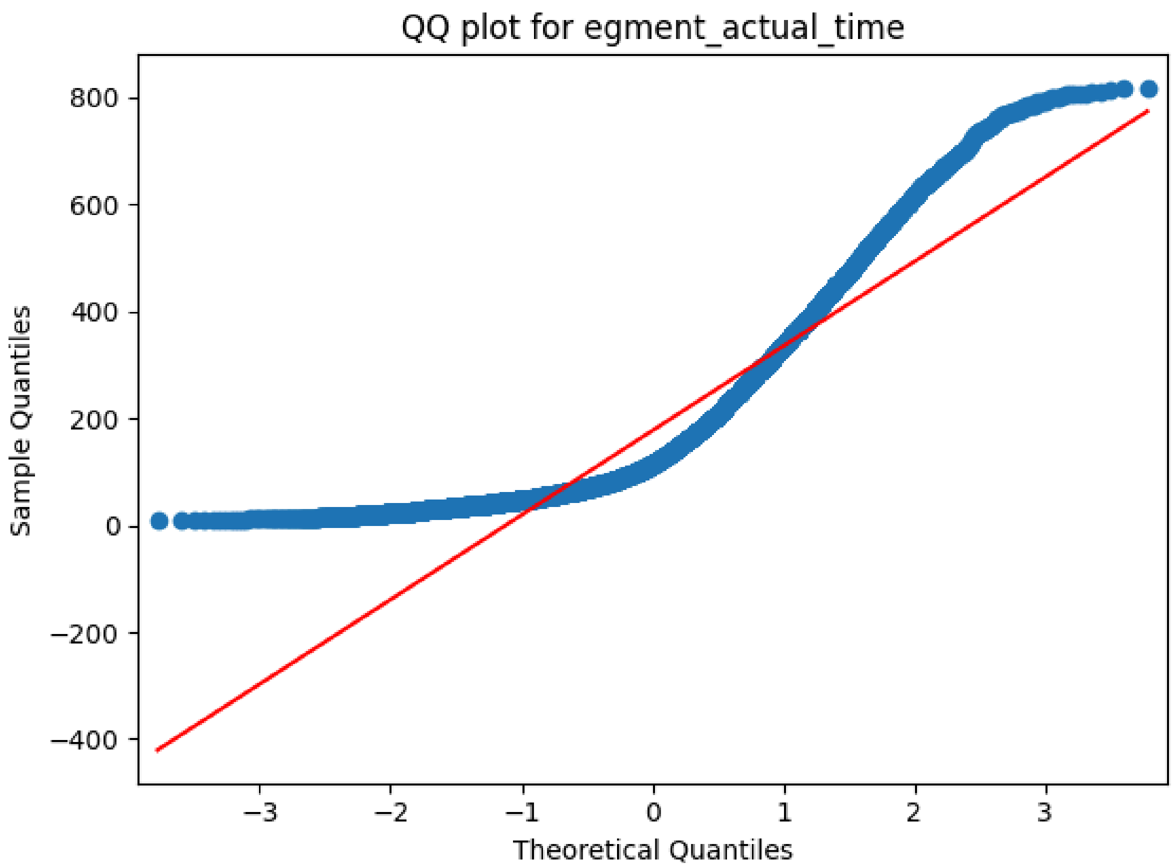


We can see from the boxplot that the actual time and segment actual time do not differ much.

In [645...

```
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['segment_actual_time_sum'], line="s")
plt.title('QQ plot for egment_actual_time')
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```



Apply Shapiro-Wilk test

Ho: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [646…

```
from scipy.stats import shapiro
test_stat, p_value = shapiro(trip['segment_actual_time_sum'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 8.3100134161197e-49
The sample does not follow normal distribution
```

Homogeneity of Variances using Lavene's test

In [647…

```
from scipy.stats import levene
test_stat, p_value = levene(trip['actual_time'], trip['segment_actual_time_sum'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.5781800861905502
The samples have Homogenous Variance
```

Since the samples do not come from normal distribution T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

Null Hypothesis: There is no difference between actual_time and segment_actual_time

Alternative Hypothesis: There is a difference between actual_time and segment_actual_time

In [648…

```
H0="There is no difference between actual_time and segment_actual_time"
Ha="There is a difference between actual_time and segment_actual_time"
alpha=0.05
test_stat, p_value = mannwhitneyu(trip['actual_time'], trip['segment_actual_time_su
print('p-value', p_value)
if p_value < alpha:
    print("Result: \nReject null hypothesis. \n",Ha)
else:
    print("Result: \nFail to reject null hypothesis. \n",H0)
```

```
p-value 0.3350750092211148
Result:
Fail to reject null hypothesis.
 There is no difference between actual_time and segment_actual_time
```

The hypothesis test result confirms our observation from the visual analysis

Insights: it can be concluded that actual_time and segment_actual_time are similar

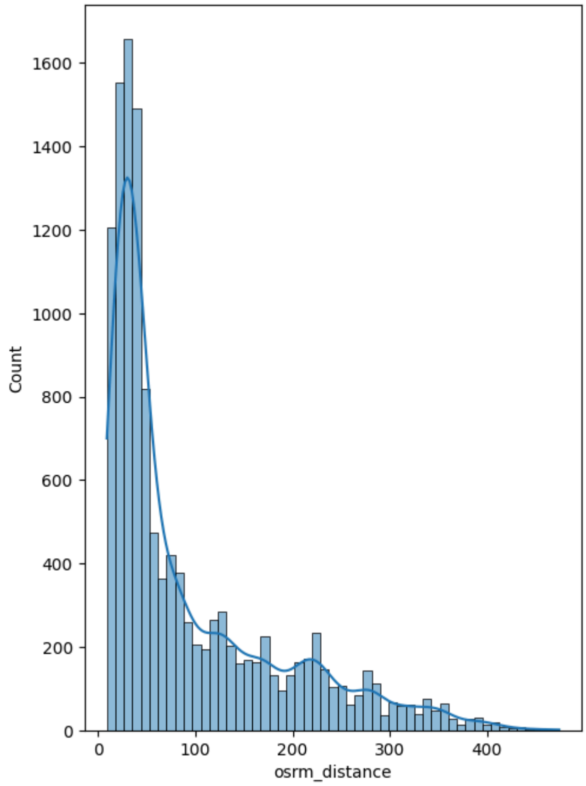
c. OSRM distance aggregated value and segment OSRM distance aggregated value.

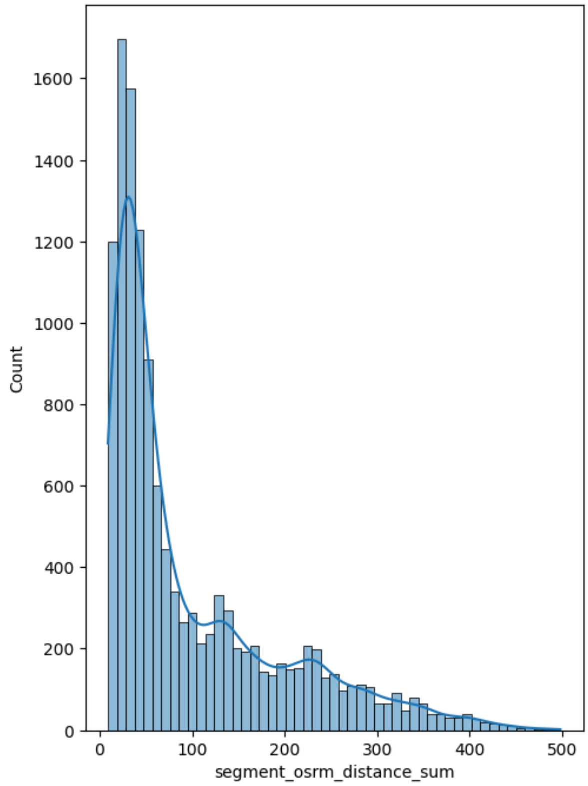
Null Hypothesis - There is no difference between osrm distance and segment_osrm distance.

Alternative Hypothesis - There is significant difference between osrm distance and segment_osrm distance.

In [649…

```
#distribution of osrm distance and segment osrm distance
plt.figure(figsize = (12, 8))
plt.subplot(1,2,1)
sns.histplot(data = trip['osrm_distance'], kde=True)
plt.subplot(1,2,2)
sns.histplot(data=trip['segment_osrm_distance_sum'], kde=True)
plt.show()
```

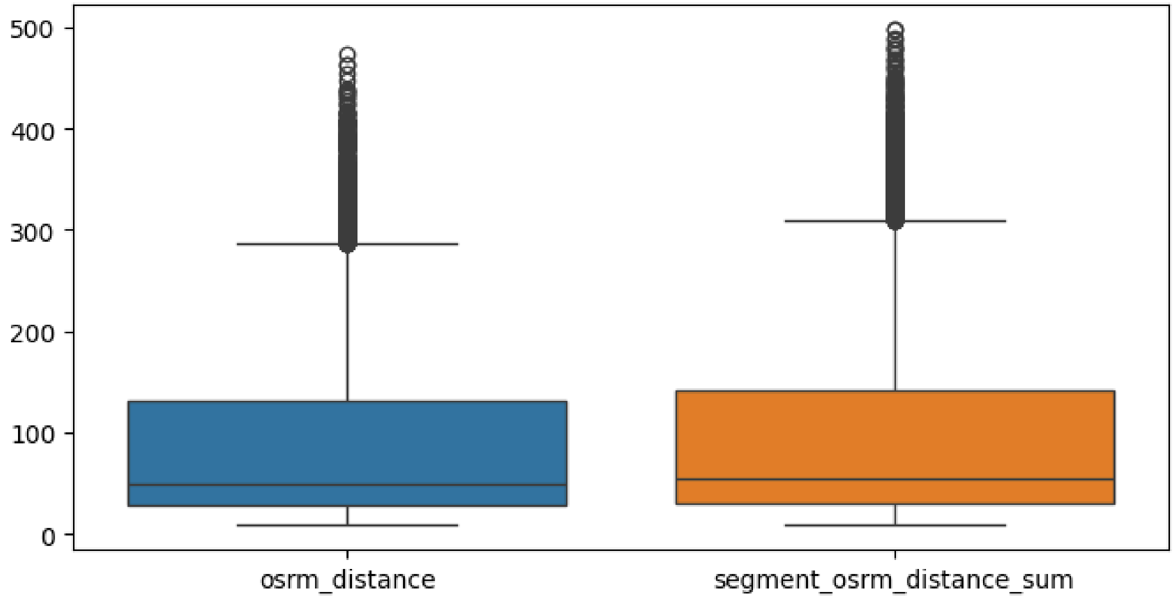




Distributions for both parameters are very similar with right-skew

In [650…

```
plt.figure(figsize = (8, 4))
sns.boxplot(data = trip[['osrm_distance','segment_osrm_distance_sum']])
plt.show()
```

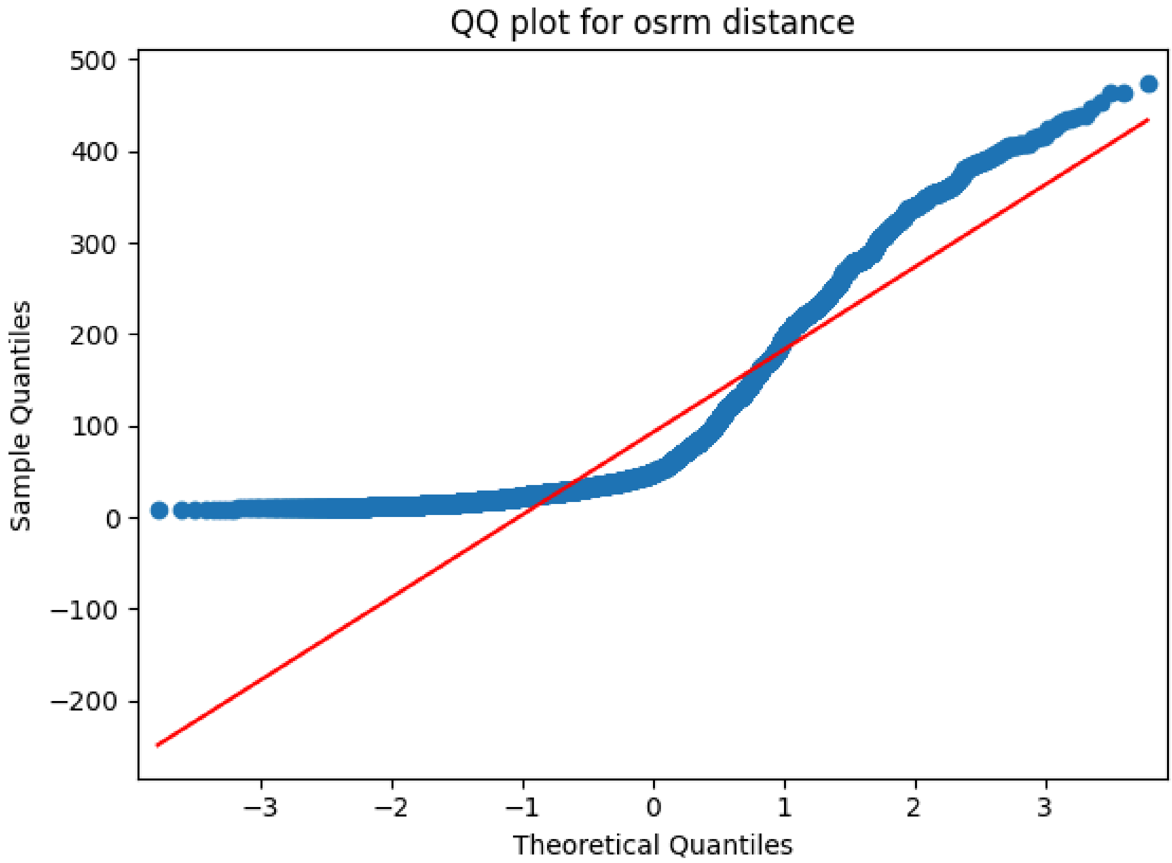


The box plot shows a small difference between the mean values of osrm distance and segment osrm distance

In [651…

```
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['osrm_distance'],line="s")
plt.title('QQ plot for osrm distance')
plt.tight_layout()
plt.show()
```

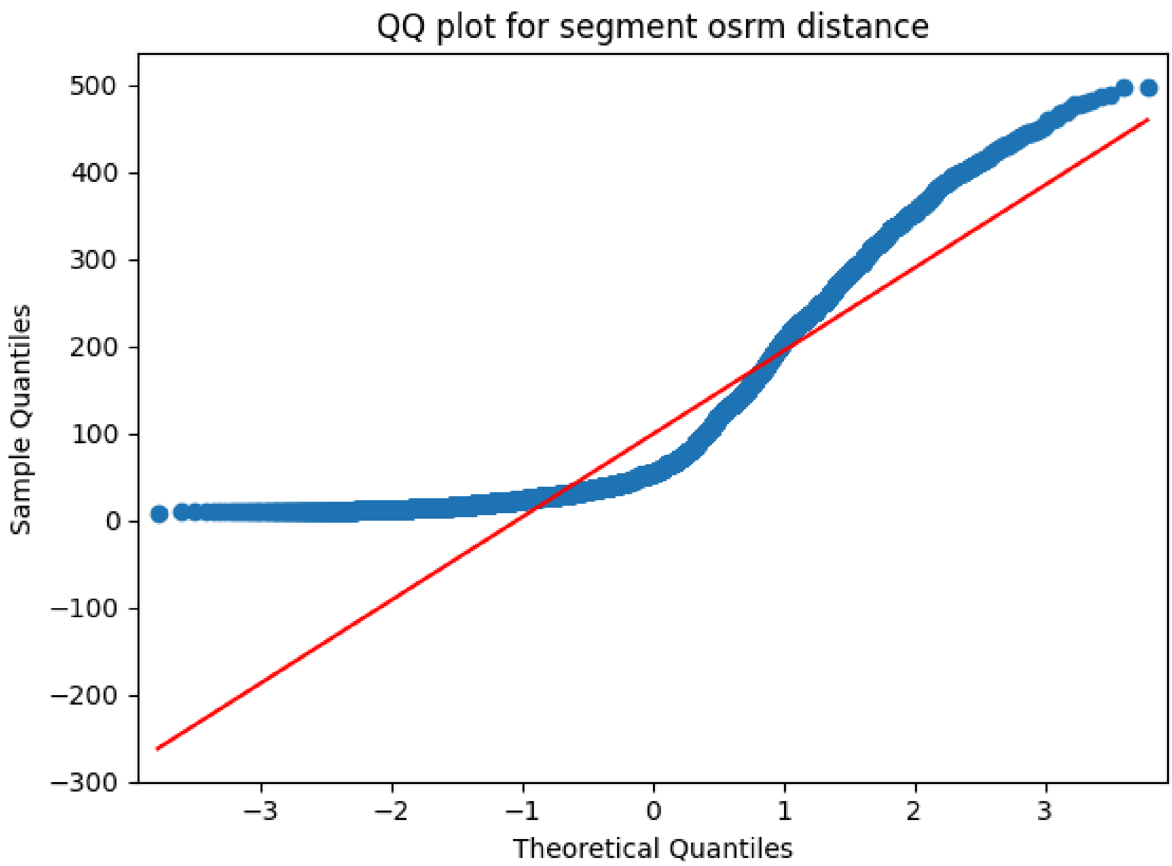
```
<Figure size 1000x400 with 0 Axes>
```



In [652…

```
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['segment_osrm_distance_sum'], line="s")
plt.title('QQ plot for segment osrm distance')
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```



Samples do not follow normal distribution

Apply Shapiro-Wilk test

Ho: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [653...

```
test_stat, p_value = shapiro(trip['osrm_distance'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 2.8203009811114866e-51
The sample does not follow normal distribution
```

In [654...

```
test_stat, p_value = shapiro(trip['segment_osrm_distance_sum'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 8.618720532885107e-52
The sample does not follow normal distribution
```

Homogeneity of Variances using Lavene's test

In [655...

```
from scipy.stats import levene
test_stat, p_value = levene(trip['osrm_distance'], trip['segment_osrm_distance_sum'
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 1.246011259053651e-05
The samples do not have  Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

In [656...

```
H0="There is no difference between osrm distance and segment_osrm distance."

Ha="There is significant difference between osrm distance and segment_osrm distance
alpha = 0.05

u_stat, p_value = mannwhitneyu(trip['osrm_distance'], trip['segment_osrm_distance_s

print('Test Statistic:', u_stat)
print('P value:', p_value)

if p_value < alpha:
```

```
    print("Result: \nReject null hypothesis. \nThere is significant difference betwee
else:
    print("Result: \nFail to reject null hypothesis. \n There is no difference betwee
```

```
Test Statistic: 78081822.0
P value: 1.773217099903382e-08
Result:
Reject null hypothesis.
There is significant difference between osrm distance and segment_osrm distance.
```

Insights: it can be concluded that osrm_distance and segment_osrm_distance are not similar.

The hypothesis test result confirms our observation from the visual analysis

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

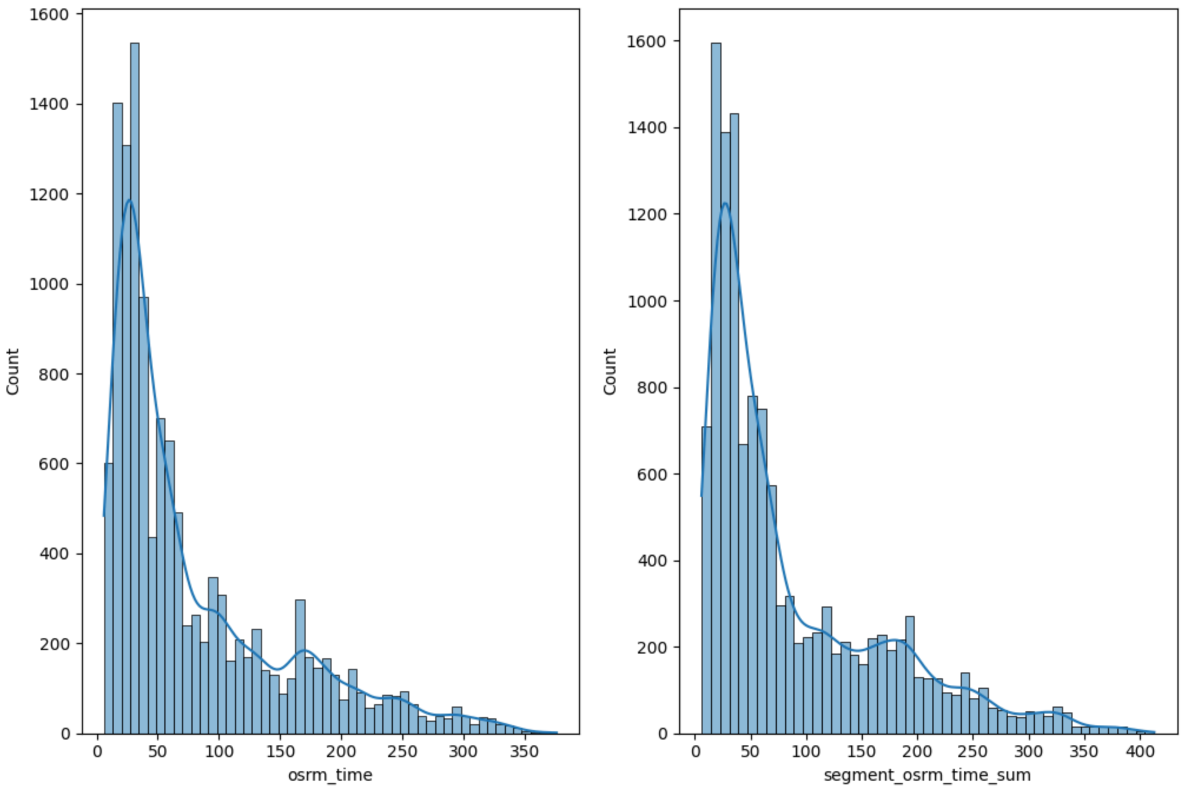
Null Hypothesis - There is no difference between osrm time and segment_osrm time.

Alternative Hypothesis - There is significant difference between osrm time and segment_osrm time.

Visual Tests to know if the samples follow normal distribution

In [657…

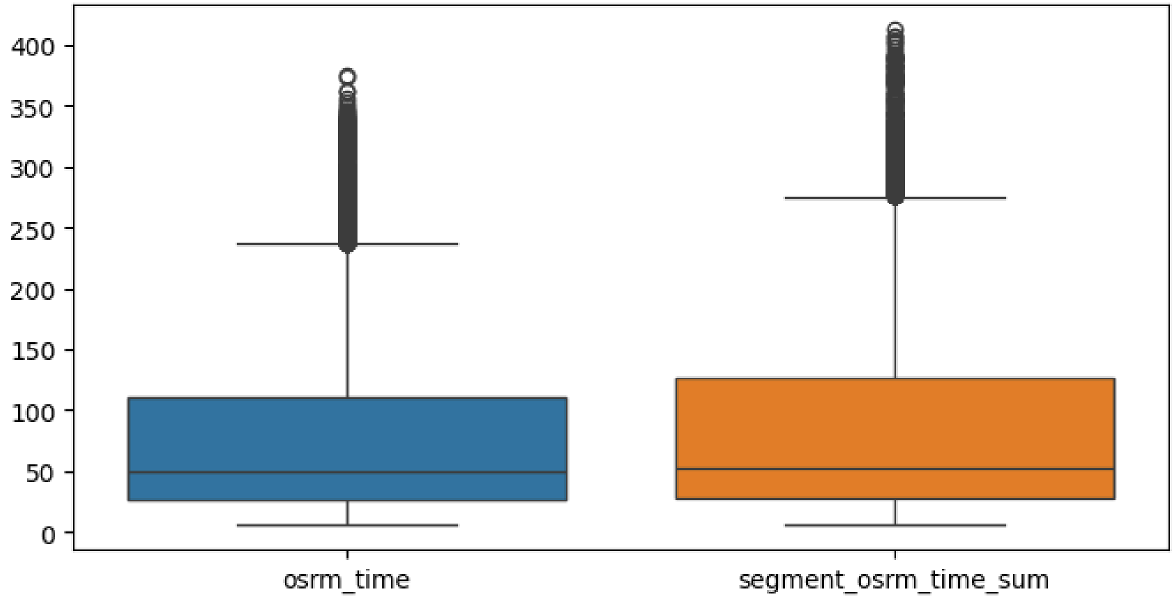
```
#distribution of osrm distance and segment osrm distance
plt.figure(figsize = (12, 8))
plt.subplot(1,2,1)
sns.histplot(data = trip['osrm_time'], kde=True)
plt.subplot(1,2,2)
sns.histplot(data=trip['segment_osrm_time_sum'], kde=True)
plt.show()
```



The distributions are right skewed

In [658...

```
plt.figure(figsize = (8, 4))
sns.boxplot(data = trip[['osrm_time','segment_osrm_time_sum']])
plt.show()
```

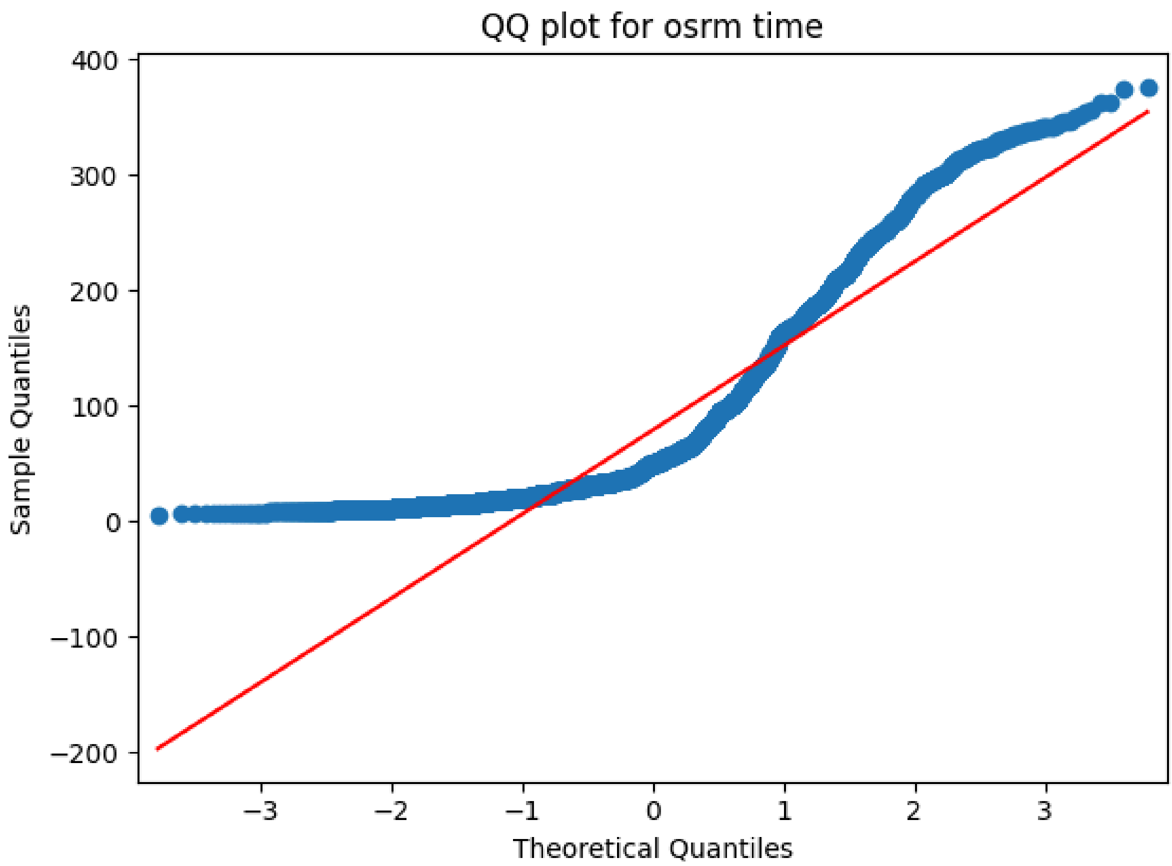


The boxplot and the lineplot of 1000 trips shows that osrm_time is lesser than segment_osrm_time

In [659...

```
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['osrm_time'],line="s")
plt.title('QQ plot for osrm time')
plt.tight_layout()
plt.show()
```

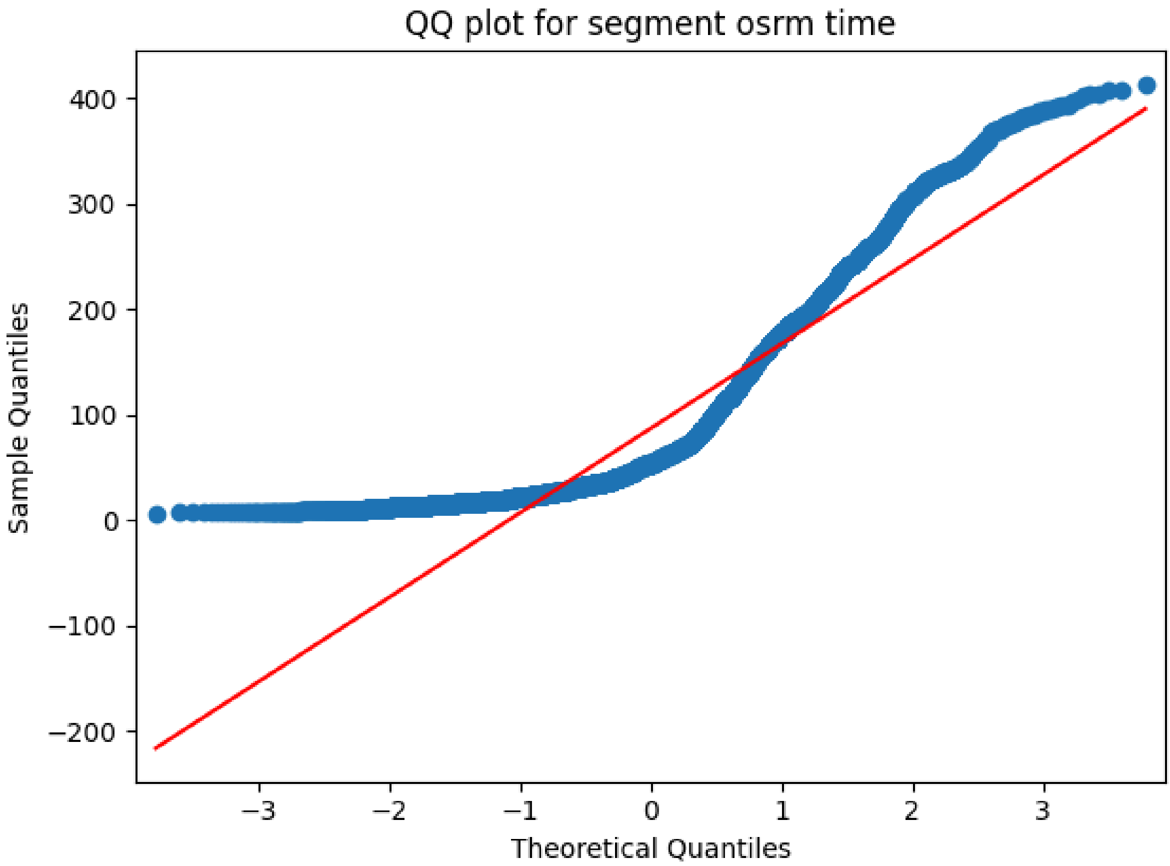
<Figure size 1000x400 with 0 Axes>



In [660…

```
#Distribution check using QQ Plot
plt.figure(figsize=(10, 4))
sm.qqplot(trip['segment_osrm_time_sum'], line="s")
plt.title('QQ plot for segment osrm time')
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```



samples do not follow normal distribution

Apply Shapiro-Wilk test

Ho: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [661…

```
test_stat, p_value = shapiro(trip['osrm_time'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 9.11797801017103e-50
The sample does not follow normal distribution
```

In [662…

```
test_stat, p_value = shapiro(trip['segment_osrm_time_sum'].sample(3000))
print('p-value', p_value)
if p_value < 0.05:
```

```
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 2.4392109875757093e-49
The sample does not follow normal distribution
```

Homogeneity of Variances using Lavene's test

In [663...

```
from scipy.stats import levene
test_stat, p_value = levene(trip['osrm_time'], trip['segment_osrm_time_sum'])
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 7.146289584700569e-14
The samples do not have  Homogenous Variance
```

Since the samples do not follow any of the assumptions, T-Test cannot be applied here. We can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples

In [664...

```
Ho="There is no difference between osrm time and segment_osrm time."

Ha="There is significant difference between osrm time and segment_osrm time."
alpha = 0.05

u_stat, p_value = mannwhitneyu(trip['osrm_time'], trip['segment_osrm_time_sum'])

print('Test Statistic:', u_stat)
print('P value:', p_value)

if p_value < alpha:
  print("Result: \nReject null hypothesis. \nThere is significant difference betwee
else:
  print("Result: \nFail to reject null hypothesis. \n There is no difference betwee
```

```
Test Statistic: 77704262.0
P value: 3.501031561380257e-10
Result:
Reject null hypothesis.
There is significant difference between osrm time and segment_osrm time.
```

Insights: It can be concluded that osrm_time and segment_osrm_time are not similar

The hypothesis test result confirms our observation from the visual analysis

2. Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

In [665...

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
trip[num_cols] = scaler.fit_transform(trip[num_cols])
```

6. Business Insights & Recommendations

Patterns observed in the data along with what you can infer from them.

1. Checking from where most orders are coming from (State, Corridor, etc.)

In [666…

```
trip[["trip_uuid","source_center", "destination_center","source_city", "destinatior
```

Out[666]:

```
trip_uuid             12759
source_center           909
destination_center     1010
source_city             692
destination_city        812
dtype: int64
```

In [667…

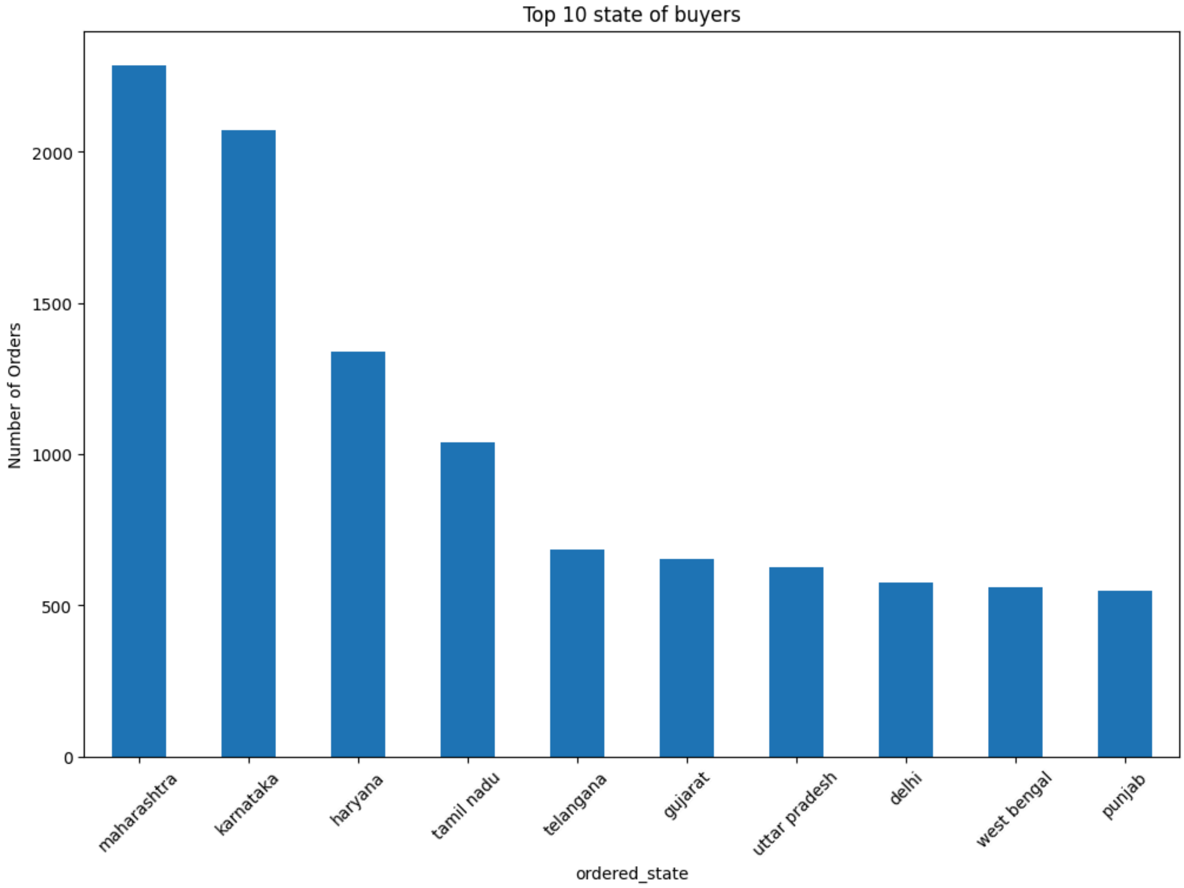
```
order_state= trip["destination_state"].value_counts()
order_state.head(10)
```

Out[667]:

```
destination_state
maharashtra      2286
karnataka        2070
haryana          1337
tamil nadu       1040
telangana         682
gujarat           653
uttar pradesh     625
delhi             574
west bengal       559
punjab            549
Name: count, dtype: int64
```

In [668…

```
# Plot the top 10 states
plt.figure(figsize=(12, 8))
order_state.head(10).plot(kind='bar')
plt.title('Top 10 state of buyers')
plt.xlabel('ordered_state')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()
```



Insights: highest number of orders comes from Maharashtra, followed by Karnataka, Haryana, Tamil Nadu, and telangana.

In [669…

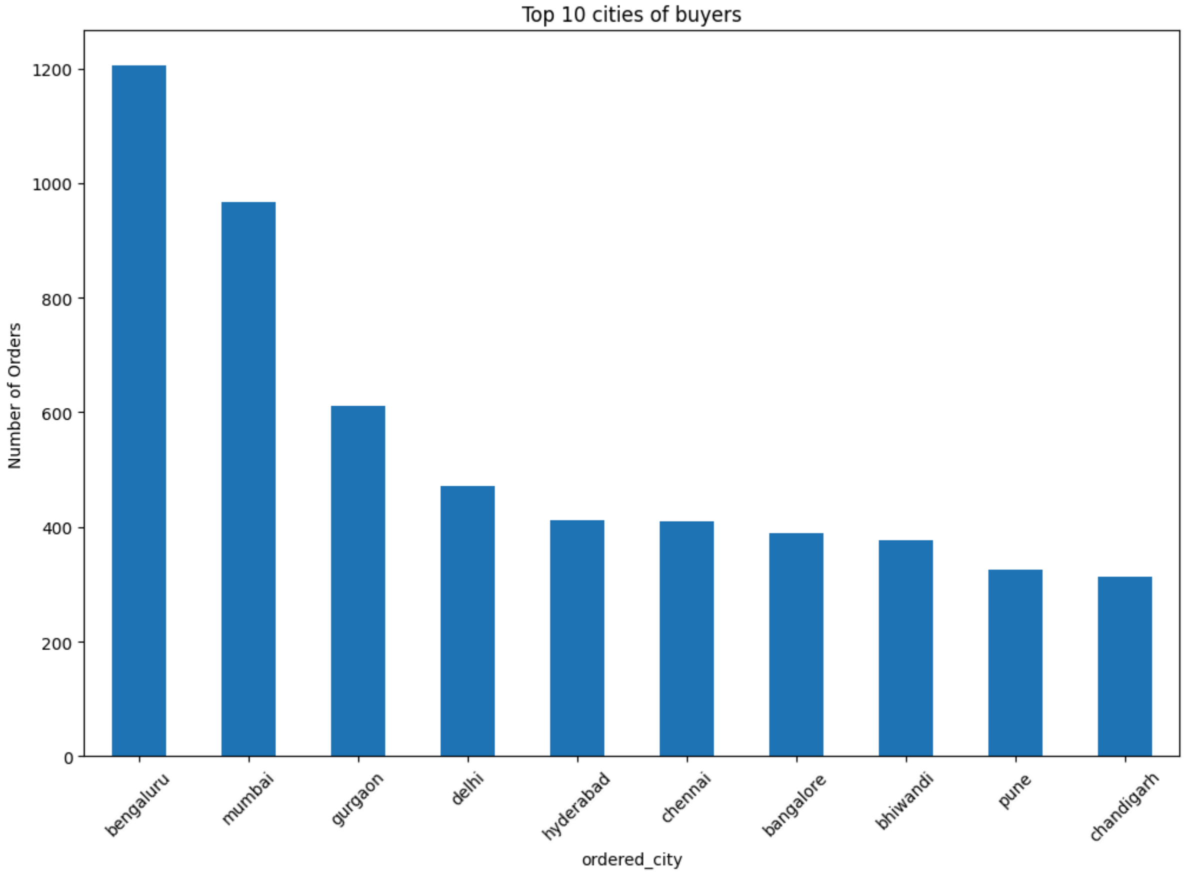
```
order_state= trip["destination_city"].value_counts()
order_state.head(10)
```

Out[669]:

```
destination_city
bengaluru      1206
mumbai          967
gurgaon         611
delhi           471
hyderabad       411
chennai         410
bangalore       389
bhiwandi        377
pune            326
chandigarh      314
Name: count, dtype: int64
```

In [670…

```
# Plot the top 10 cities
plt.figure(figsize=(12, 8))
order_state.head(10).plot(kind='bar')
plt.title('Top 10 cities of buyers')
plt.xlabel('ordered_city')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()
```



Insights: Most of the orders come from cities like Bengaluru, Mumbai, Gurgaon, Delhi and Hydrabad.

2. Busiest corridor, avg distance between them, avg time taken, etc

To find the busiest corridor, look at the source_name and destination_name fields and count trips between these locations.

In [671…

```
busiest_corridor=trip.groupby(['source_name','destination_name'])['trip_uuid'].cour
busiest_corridor.head(10)
```

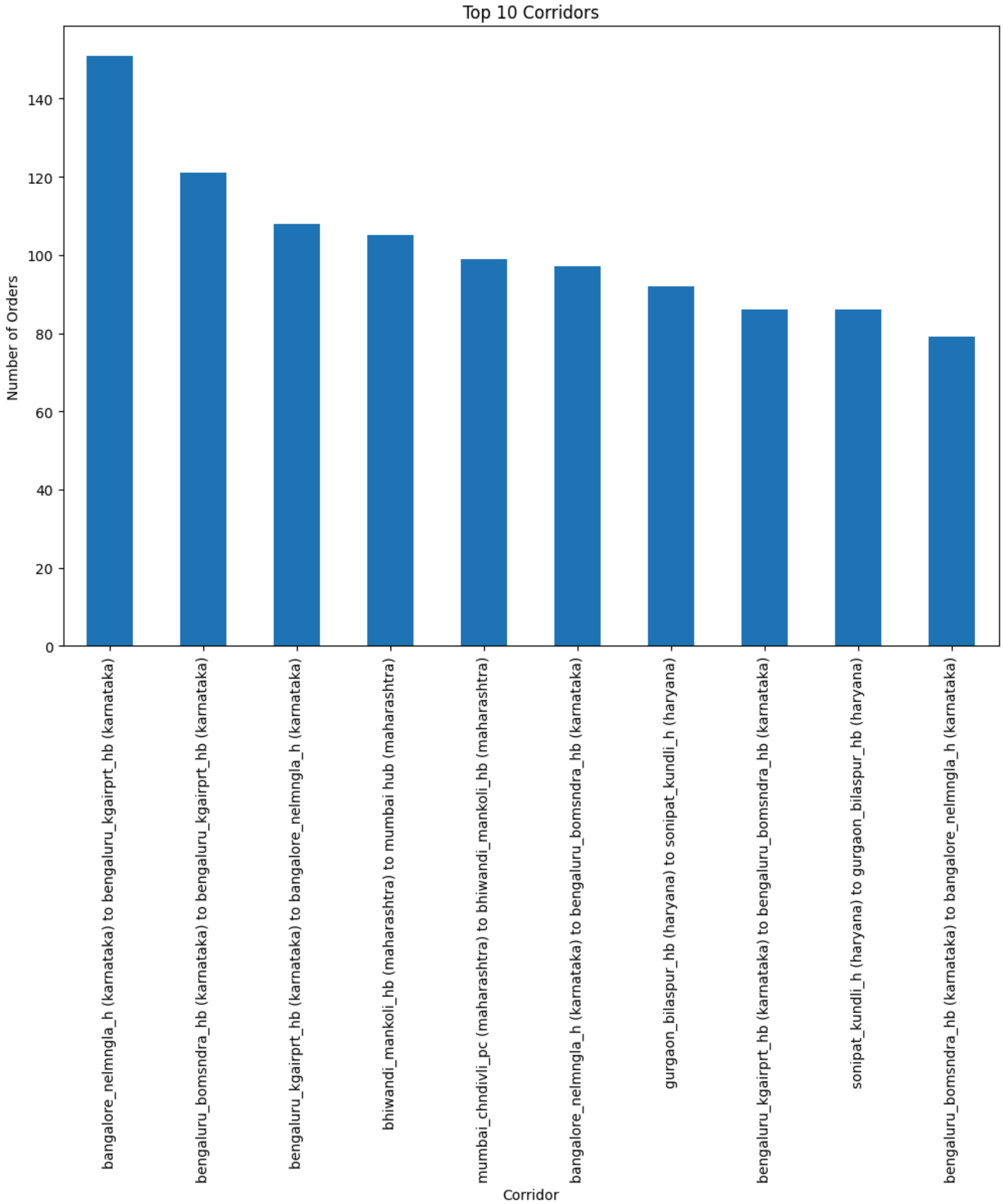
Out[671]:

	source_name	destination_name	trip_uuid
0	bangalore_nelmngla_h (karnataka)	bengaluru_kgairprt_hb (karnataka)	151
1	bengaluru_bomsndra_hb (karnataka)	bengaluru_kgairprt_hb (karnataka)	121
2	bengaluru_kgairprt_hb (karnataka)	bangalore_nelmngla_h (karnataka)	108
3	bhiwandi_mankoli_hb (maharashtra)	mumbai hub (maharashtra)	105
4	mumbai_chndivli_pc (maharashtra)	bhiwandi_mankoli_hb (maharashtra)	99
5	bangalore_nelmngla_h (karnataka)	bengaluru_bomsndra_hb (karnataka)	97
6	gurgaon_bilaspur_hb (haryana)	sonipat_kundli_h (haryana)	92
7	sonipat_kundli_h (haryana)	gurgaon_bilaspur_hb (haryana)	86
8	bengaluru_kgairprt_hb (karnataka)	bengaluru_bomsndra_hb (karnataka)	86
9	bengaluru_bomsndra_hb (karnataka)	bangalore_nelmngla_h (karnataka)	79

In [672…

```
# Create a 'corridor' field combining source and destination
trip['corridor'] = trip['source_name'] + ' to ' + trip['destination_name']

# Count the number of trips for each corridor
corridor_counts = trip['corridor'].value_counts()

# Display the top 10 corridors
print(corridor_counts.head(10))

# Plot the top 10 corridors
plt.figure(figsize=(12, 8))
corridor_counts.head(10).plot(kind='bar')
plt.title('Top 10 Corridors')
plt.xlabel('Corridor')
plt.ylabel('Number of Orders')
plt.xticks(rotation=90)
plt.show()
```

```
corridor
bangalore_nelmngla_h (karnataka) to bengaluru_kgairprt_hb (karnataka)     151
bengaluru_bomsndra_hb (karnataka) to bengaluru_kgairprt_hb (karnataka)    121
bengaluru_kgairprt_hb (karnataka) to bangalore_nelmngla_h (karnataka)     108
bhiwandi_mankoli_hb (maharashtra) to mumbai hub (maharashtra)             105
mumbai_chndivli_pc (maharashtra) to bhiwandi_mankoli_hb (maharashtra)      99
bangalore_nelmngla_h (karnataka) to bengaluru_bomsndra_hb (karnataka)      97
gurgaon_bilaspur_hb (haryana) to sonipat_kundli_h (haryana)                92
bengaluru_kgairprt_hb (karnataka) to bengaluru_bomsndra_hb (karnataka)     86
sonipat_kundli_h (haryana) to gurgaon_bilaspur_hb (haryana)                86
bengaluru_bomsndra_hb (karnataka) to bangalore_nelmngla_h (karnataka)      79
Name: count, dtype: int64
```



In [673…

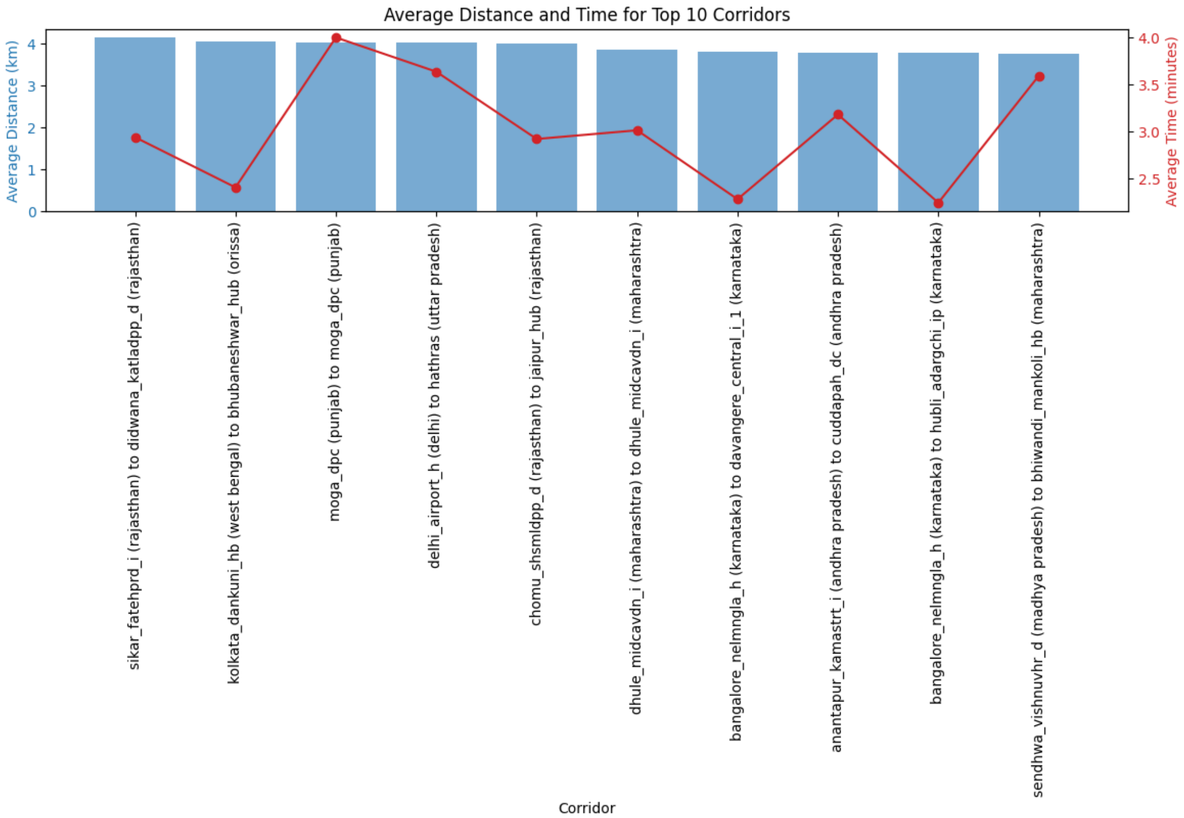
```
# Group by corridor and calculate average distance and time
corridor_stats = trip.groupby('corridor').agg({'actual_distance_to_destination': 'm

# Display the top 10 corridors by average distance
print(corridor_stats.sort_values(by='actual_distance_to_destination', ascending=Fal
```

```
                                               corridor  \
1772  sikar_fatehprd_i (rajasthan) to didwana_katlad...
1167  kolkata_dankuni_hb (west bengal) to bhubaneshw...
1375             moga_dpc (punjab) to moga_dpc (punjab)
575   delhi_airport_h (delhi) to hathras (uttar prad...
512   chomu_shsmldpp_d (rajasthan) to jaipur_hub (ra...
639   dhule_midcavdn_i (maharashtra) to dhule_midcav...
171   bangalore_nelmngla_h (karnataka) to davangere_...
84    anantapur_kamastrt_i (andhra pradesh) to cudda...
174   bangalore_nelmngla_h (karnataka) to hubli_adar...
1742  sendhwa_vishnuvhr_d (madhya pradesh) to bhiwan...

      actual_distance_to_destination  actual_time
1772                        4.142566     2.938372
1167                        4.046282     2.404059
1375                        4.039271     4.000714
575                         4.034927     3.636123
512                         3.997119     2.919514
639                         3.860131     3.012233
171                         3.817676     2.278338
84                          3.792224     3.180385
174                         3.777372     2.240621
1742                        3.768202     3.592121
```

In [674…

```python
# Plot average distance and time for top 10 corridors
top_corridors = corridor_stats.sort_values(by='actual_distance_to_destination', asc

fig, ax1 = plt.subplots(figsize=(12, 8))

color = 'tab:blue'
ax1.set_xlabel('Corridor')
ax1.set_ylabel('Average Distance (km)', color=color)
ax1.bar(top_corridors['corridor'], top_corridors['actual_distance_to_destination'],
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_xticklabels(top_corridors['corridor'], rotation=90)

ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Average Time (minutes)', color=color)
ax2.plot(top_corridors['corridor'], top_corridors['actual_time'], color=color, mark
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()
plt.title('Average Distance and Time for Top 10 Corridors')
plt.show()
```

```
<ipython-input-674-de7dbd16666b>:11: UserWarning: FixedFormatter should only be us
ed together with FixedLocator
  ax1.set_xticklabels(top_corridors['corridor'], rotation=90)
```



Insights: bangalore_nelmngla_h (karnataka) to davangere_central_i *1*(karnataka) require less time as the distance is short.

These average time and distrance mitrices help to highlight areas like moga_dpc(panjab) to moga_dpc(panjab) for improvement in route planning and logistics.

Business Insights:

- Most of the data is used for testing rather than training, with Carting being the most common route type.
- The data covers the period from September 12, 2018, to October 8, 2018, and includes 12,759 unique trip IDs, 909 source centers, 1,010 destination centers, 692 source cities, and 812 destination cities. Testing data is more common than training data.
- The actual_time and osrm_time features show significant differences.
- The actual_time and segment_actual_time features are quite similar.
- List item
- The osrm_distance and segment_osrm_distance features show significant differences from each other.
- The osrm time and segment_osrm time features also show significant differences from each other.
- Most orders come from states such as Maharashtra, Karnataka, Haryana, Tamil Nadu and Smaller states like Arunachal Pradesh, Nagaland, Himachal, Goa etc have the lowest

volumes as expected.

- Most orders come from cities such as Bengaluru, Mumbai, Gurgaon, Delhi, and Hyderabad.
- Trips mostly start from states like Maharashtra, Karnataka, Haryana, Tamil Nadu, and Telangana.
- Mumbai has the highest number of trips starting there, followed by Gurgaon, Delhi, Bengaluru, and Bhiwandi, showing these cities have a strong seller presence.
- South, North and West Zones corridors have significant traffic of orders. But, we have a smaller presence in Central, Eastern and North-Eastern zone.
- Average time and distance metrics suggest that areas like Moga_DPC in Punjab could improve route planning and logistics.

Recommendations:

- A large volume of orders either starts from or is directed to states such as Maharashtra, Karnataka, Haryana, and Tamil Nadu. Improving the efficiency of current routes could boost service coverage in these regions.
- Profiling customers in Maharashtra, Karnataka, Haryana, Tamil Nadu, and Uttar Pradesh is important. Gaining insights into why these states generate a high number of orders can help enhance both the purchasing and delivery processes for customers.
- When planning, it’s important to consider state-specific challenges such as heavy traffic and difficult terrain, especially during busy festival periods, to better meet demand.
- The OSRM trip planning system needs improvements to fix discrepancies, particularly for transporters who depend on this system for accurate routing.
- There is a noticeable gap between osrm_time and actual_time. Reducing this discrepancy is essential to improve delivery time predictions and provide more accurate estimates to customers.
- Optimise routes along corridors with maximum average speed to shorten delivery time

Actionable Items for the business.

Optimize Routes:

If certain corridors have higher average times or distances, consider optimizing these routes to improve efficiency.

Focus on High-Volume Sources:

Increase resources or improve services in regions that are the primary sources of orders.

Address Bottlenecks:

For corridors with unusually high average times or distances, investigate potential issues such as traffic patterns or inefficiencies.

Improve Forecasting:

Use insights from busy corridors and high-volume sources to better forecast demand and plan logistics.

By implementing these analyses and insights, businesses can make data-driven decisions to enhance operational efficiency and customer satisfaction.