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.

Column Profiling:

data - tells whether the data is testing or training data

trip_creation_time – Timestamp of trip creation

route_schedule_uuid – Unique Id for a particular route schedule

route_type – Transportation type

FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way Carting: Handling system consisting of small vehicles (carts)

trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)

source_center - Source ID of trip origin

source_name - Source Name of trip origin

destination_cente – Destination ID

destination_name – Destination Name

od_start_time – Trip start time

od_end_time – Trip end time

start_scan_to_end_scan – Time taken to deliver from source to destination is_cutoff – Unknown field

cutoff_factor – Unknown field

cutoff_timestamp – Unknown field

actual_distance_to_destination – Distance in Kms between source and destination warehouse

actual_time – Actual time taken to complete the delivery (Cumulative) osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)

osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)

factor – Unknown field

segment_actual_time – This is a segment time. Time taken by the subset of the package delivery

segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery

segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery

segment_factor – Unknown field

Problem Statement

The company wants to understand and process the data coming out of data engineering pipelines:

• Clean, sanitize and manipulate data to get useful features out of raw fields

• Make sense out of the raw data and help the data science team to build forecasting models on it

Concepts Used:

Feature Creation

Relationship between Features

Column Normalization /Column Standardization

Handling categorical values

Missing values - Outlier treatment / Types of outliers

Importing necessary Libraries

```
Start coding or generate with AI.
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from scipy.stats import ttest_ind,ttest_1samp,ttest_rel
```

```
!wget "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/ori
```

```
--2024-08-10 18:51:17--  https://d2beiqkhq929f0.cloudfront.net/public_assets/a
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 13
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|1
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: ‘delhivery_data.csv’

delhivery_data.csv  100%[===================>]  53.04M   199MB/s    in 0.3s

2024-08-10 18:51:17 (199 MB/s) - ‘delhivery_data.csv’ saved [55617130/5561713
```

1.Basic data cleaning and exploration

```
df=pd.read_csv("delhivery_data.csv")
df.head()
```

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

5 rows × 24 columns

```
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
```

```
df.size
```

```
3476808
```

```
df.shape
```

```
(144867, 24)
```

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

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

dtype: int64

```
# Dropping unknown columns
df.drop(['is_cutoff','cutoff_factor','cutoff_timestamp','factor','segment_factor'
```

```
# Drop null values
df.dropna(inplace=True)
```

```
df.duplicated().value_counts()
```

	count
False	144316

dtype: int64

```
df.nunique()
```

	0
data	2
trip_creation_time	14787
route_schedule_uuid	1497
route_type	2
trip_uuid	14787
source_center	1496
source_name	1496
destination_center	1466
destination_name	1466
od_start_time	26223
od_end_time	26223
start_scan_to_end_scan	1914
actual_distance_to_destination	143965
actual_time	3182
osrm_time	1531
osrm_distance	137544
segment_actual_time	746
segment_osrm_time	214
segment_osrm_distance	113497

dtype: int64

```
df['data'].unique()
```

```
array(['training', 'test'], dtype=object)
```

```
df['route_type'].unique()
```

```
array(['Carting', 'FTL'], dtype=object)
```

```
df.columns
```

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'actual_distance_to_destination',
       'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance'],
      dtype='object')
```

```
#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'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  object
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  object
 4   trip_uuid                       144316 non-null  object
 5   source_center                   144316 non-null  object
 6   source_name                     144316 non-null  object
 7   destination_center              144316 non-null  object
 8   destination_name                144316 non-null  object
 9   od_start_time                   144316 non-null  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144316 non-null  float64
 12  actual_distance_to_destination  144316 non-null  float64
 13  actual_time                     144316 non-null  float64
 14  osrm_time                       144316 non-null  float64
 15  osrm_distance                   144316 non-null  float64
 16  segment_actual_time             144316 non-null  float64
 17  segment_osrm_time               144316 non-null  float64
 18  segment_osrm_distance           144316 non-null  float64
dtypes: datetime64[ns](3), float64(8), object(8)
memory usage: 22.0+ MB
```

```
df_original=df.copy()
```

2.Build some features to prepare the data for actual analysis.

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 name it as segment_key. b. You can use 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. c. This way you'll get new columns named segment_actual_time_sum, segment_osrm_distance_sum, segment_osrm_time_sum.
2. Aggregating at segment level a. Create a dictionary named create_segment_dict, that defines how to aggregate and select values. i. You can keep the first and last values for some numeric/categorical fields if aggregating them won't make sense. b. Further group the data by segment_key because you want to perform aggregation operations for different segments of each trip based on the segment_key value. c. The aggregation functions specified in the create_segment_dict are applied to each group of rows with the same segment_key. d. Sort the resulting DataFrame segment, by two criteria: i. First, it sorts by segment_key to ensure that segments are ordered consistently. ii. Second, it sorts by od_end_time in ascending order, ensuring that segments within the same trip are ordered by their end times from earliest to latest.

```
df['trip_uuid']=df['trip_uuid'].astype(str).str.split('-').str[1]
df['trip_uuid'].head()
```

	trip_uuid
0	153741093647649320
1	153741093647649320
2	153741093647649320
3	153741093647649320
4	153741093647649320

dtype: object

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

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

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

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

	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm_time_
0	14.0	11.9653	
1	24.0	21.7243	
2	40.0	32.5395	
3	61.0	45.5619	
4	67.0	49.4772	
...	...	...	
144862	92.0	65.3487	
144863	118.0	82.7212	1
144864	138.0	103.4265	1
144865	155.0	122.3150	1
144866	423.0	131.1238	1

144316 rows × 3 columns

```
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',

}
```

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

	index	segment_key	data	trip_creation_time
0	0	153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741
1	1	153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741
2	2	153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430
3	3	153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430
4	4	153671043369099517IND000000ACBIND160002AAC	training	2018-09-12 00:00:33.691250

5 rows × 21 columns

```
segment['od_time_diff_hour'] = (pd.to_datetime(segment['od_end_time']) - pd.to_da
segment['od_time_diff_hour']
```

	od_time_diff_hour
0	1260.604421
1	999.505379
2	58.832388
3	122.779486
4	834.638929
...	...
26217	62.115193
26218	91.087797
26219	44.174403
26220	287.474007
26221	66.933565

26222 rows × 1 columns

dtype: float64

```
segment
```

	index	segment_key	data	trip_creation_t
0	0	153671041653548748IND209304AAAIND000000ACB	training	2018-0 00:00:16.53
1	1	153671041653548748IND462022AAAIND209304AAA	training	2018-0 00:00:16.53
2	2	153671042288605164IND561203AABIND562101AAA	training	2018-0 00:00:22.88
3	3	153671042288605164IND572101AAAIND561203AAB	training	2018-0 00:00:22.88
4	4	153671043369099517IND000000ACBIND160002AAC	training	2018-0 00:00:33.69
...	...	...	...	
26217	26217	153861115439069069IND628204AAAIND627657AAA	test	2018-1 23:59:14.39
26218	26218	153861115439069069IND628613AAAIND627005AAA	test	2018-1 23:59:14.39
26219	26219	153861115439069069IND628801AAAIND628204AAA	test	2018-1 23:59:14.39
26220	26220	153861118270144424IND583119AAAIND583101AAA	test	2018-1 23:59:42.70
26221	26221	153861118270144424IND583201AAAIND583119AAA	test	2018-1 23:59:42.70

26222 rows × 22 columns

```
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',

}


trip= segment.groupby('trip_uuid').agg(create_trip_dict).reset_index(drop=True)
trip.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uu
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	1536710416535487
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	1536710422886051
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	1536710433690995
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	1536710460113304
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	1536710529740466

Next steps: Generate code with `trip` View recommended plots New interactive sheet

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.
2. Destination Name: Split and extract features out of destination. City-place-code (State)
3. Source Name: Split and extract features out of destination. City-place-code (State)
4. Trip_creation_time: Extract features like month, year, day, etc.

```
def extract_state(x):
    # transform "gurgaon_bilaspur_hb (haryana)" into "haryana"
    state = x.split('(')[1]

    return state[:-1] #removing ')' from ending

def extract_city(x):
    #we will remove state
    city = x.split(' (')[0]

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


    return city

def extract_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 extract_code(x):
    # we will remove state
    x = x.split('(')[0]

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

    return 'none'


trip['destination_state'] = trip['destination_name'].apply(lambda x: extract_stat
trip['destination_city']  = trip['destination_name'].apply(lambda x: extract_city
trip['destination_place'] = trip['destination_name'].apply(lambda x: extract_plac
trip['destination_code']  = trip['destination_name'].apply(lambda x: extract_code


trip['source_state'] = trip['source_name'].apply(lambda x: extract_state(x))
trip['source_city']  = trip['source_name'].apply(lambda x: extract_city(x))
trip['source_place'] = trip['source_name'].apply(lambda x: extract_place(x))
trip['source_code']  = trip['source_name'].apply(lambda x: extract_code(x))
```

```
trip
```

	data	trip_creation_time	route_schedule_uuid	route_type	tri
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	153671041653
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	153671042288
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	153671043369
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	153671046011
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	153671052974
...	...	...	...	...	
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	153861095625
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	153861104386
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	153861106442
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	153861115439
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	153861118270

14787 rows × 26 columns

```
trip['trip_creation_time'] = pd.to_datetime(trip['trip_creation_time'])

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

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

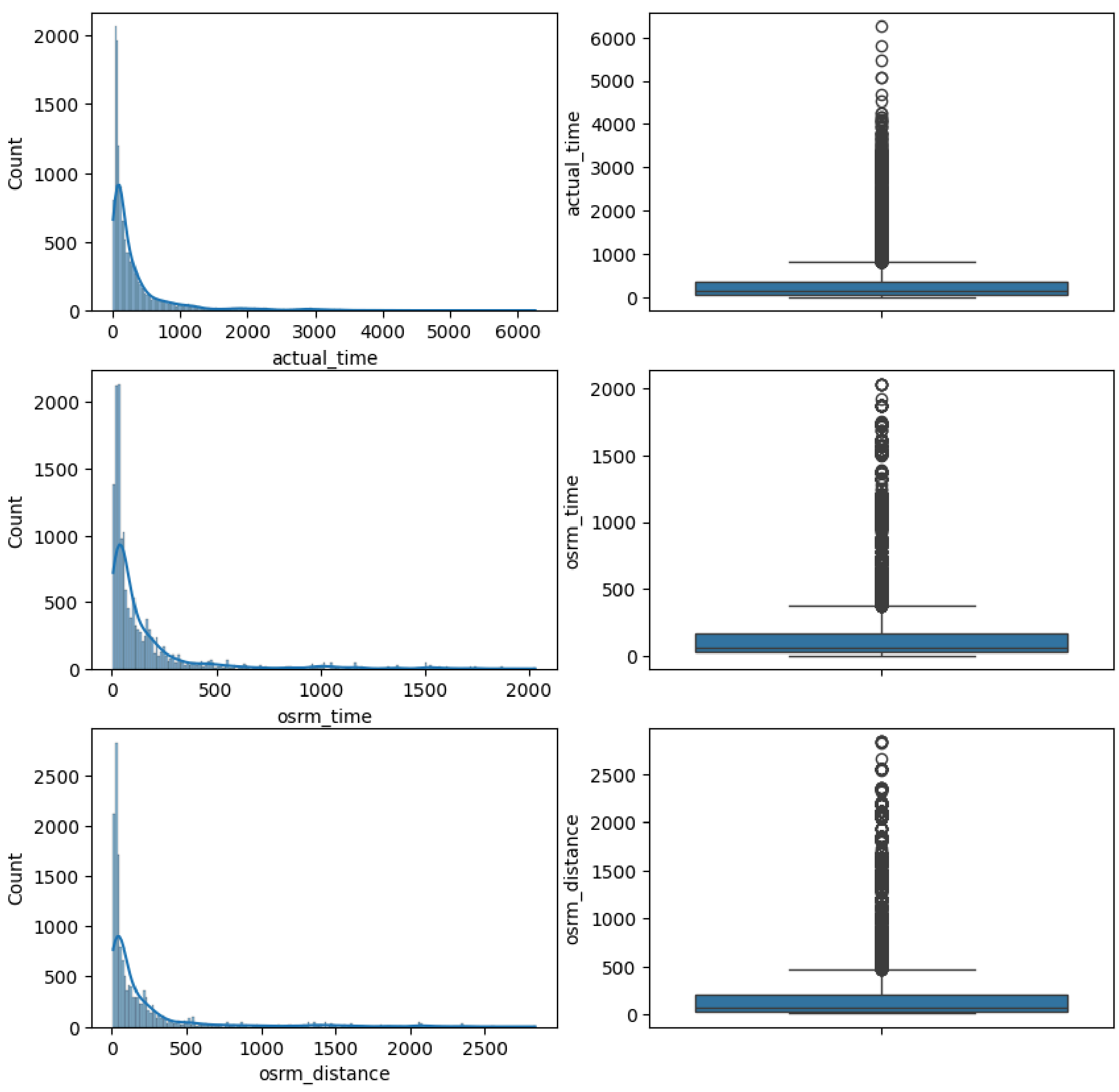
	trip_year	trip_month	trip_hour	trip_day	trip_week	trip_dayofweek
0	2018	9	0	12	37	2
1	2018	9	0	12	37	2
2	2018	9	0	12	37	2
3	2018	9	0	12	37	2
4	2018	9	0	12	37	2
...	...	...	...	...	...	...
14782	2018	10	23	3	40	2
14783	2018	10	23	3	40	2
14784	2018	10	23	3	40	2
14785	2018	10	23	3	40	2
14786	2018	10	23	3	40	2

14787 rows × 6 columns

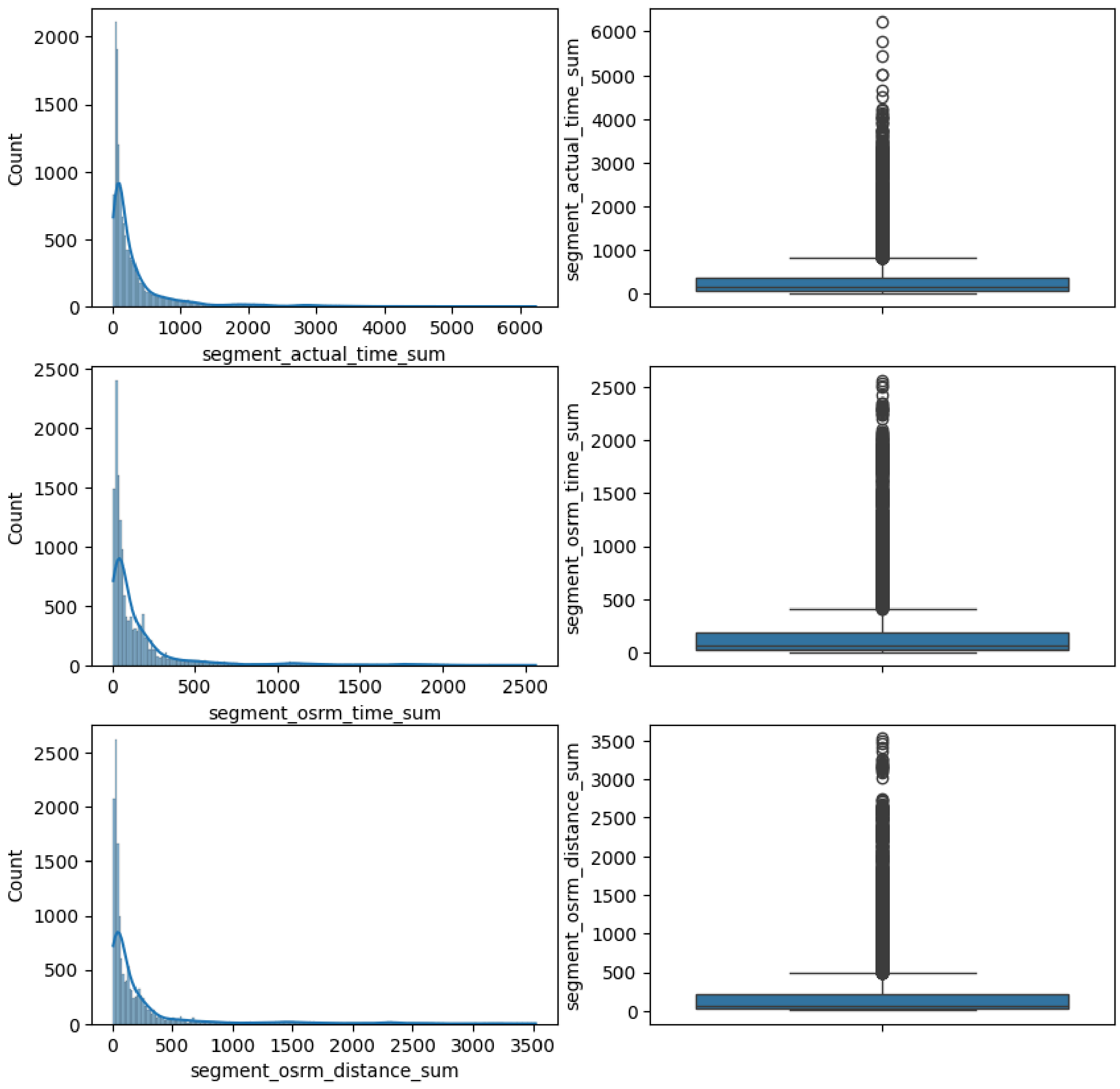
4. In-depth analysis:

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

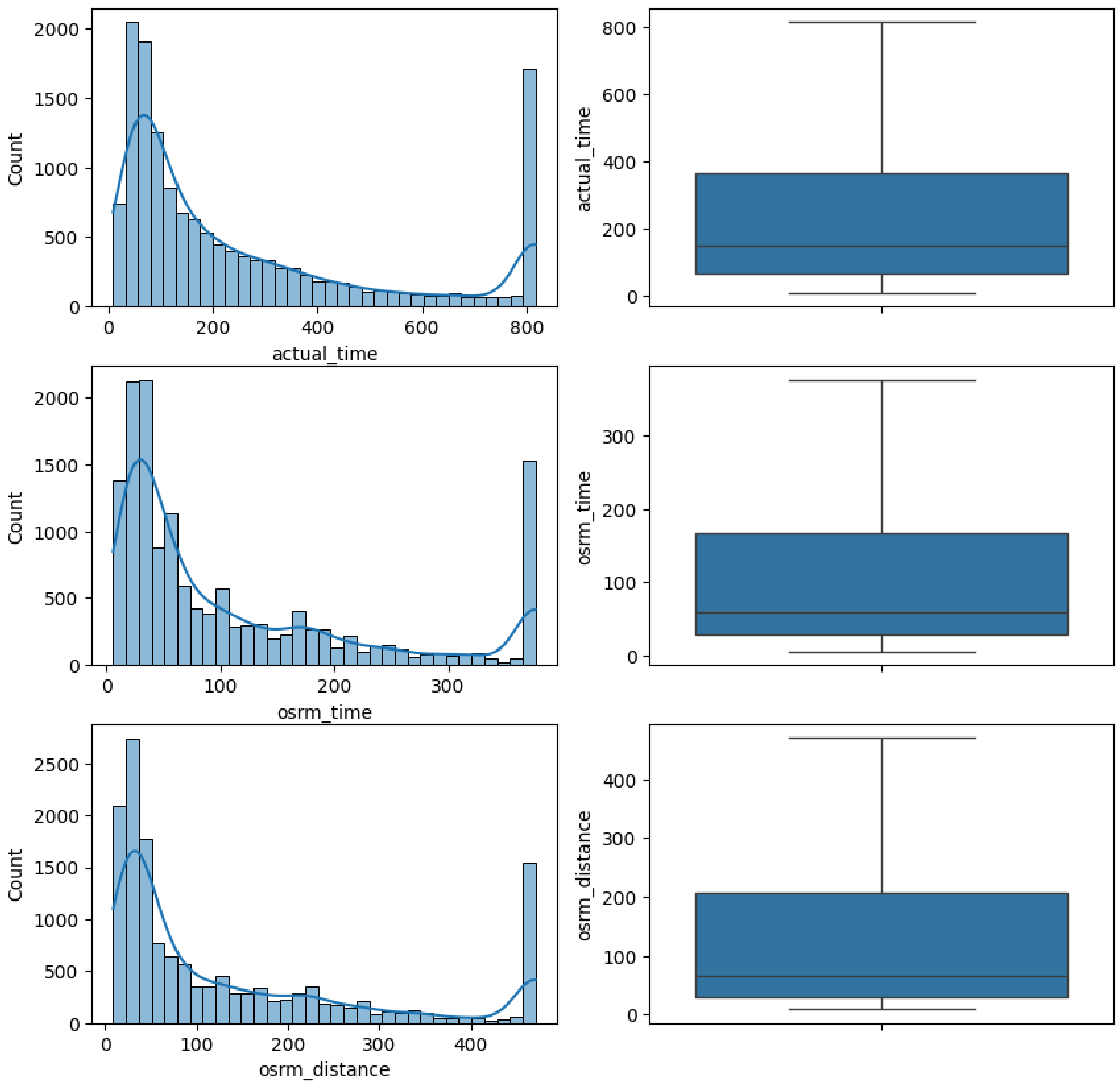
```
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
fig, axs = plt.subplots(3, 2,  figsize=(10,10))
sns.histplot(ax=axs[0,0],data= trip['actual_time'],kde=True)
sns.boxplot(ax=axs[0,1],data=trip['actual_time'])

sns.histplot(ax=axs[1,0],data= trip['osrm_time'],kde=True)
sns.boxplot(ax=axs[1,1],data= trip['osrm_time'])

sns.histplot(ax=axs[2,0],data= trip['osrm_distance'],kde=True)
sns.boxplot(ax=axs[2,1],data= trip['osrm_distance'])


plt.show()
```


Count
actual_time
actual_time
Count
osrm_time
osrm_time
Count
osrm_distance
osrm_distance

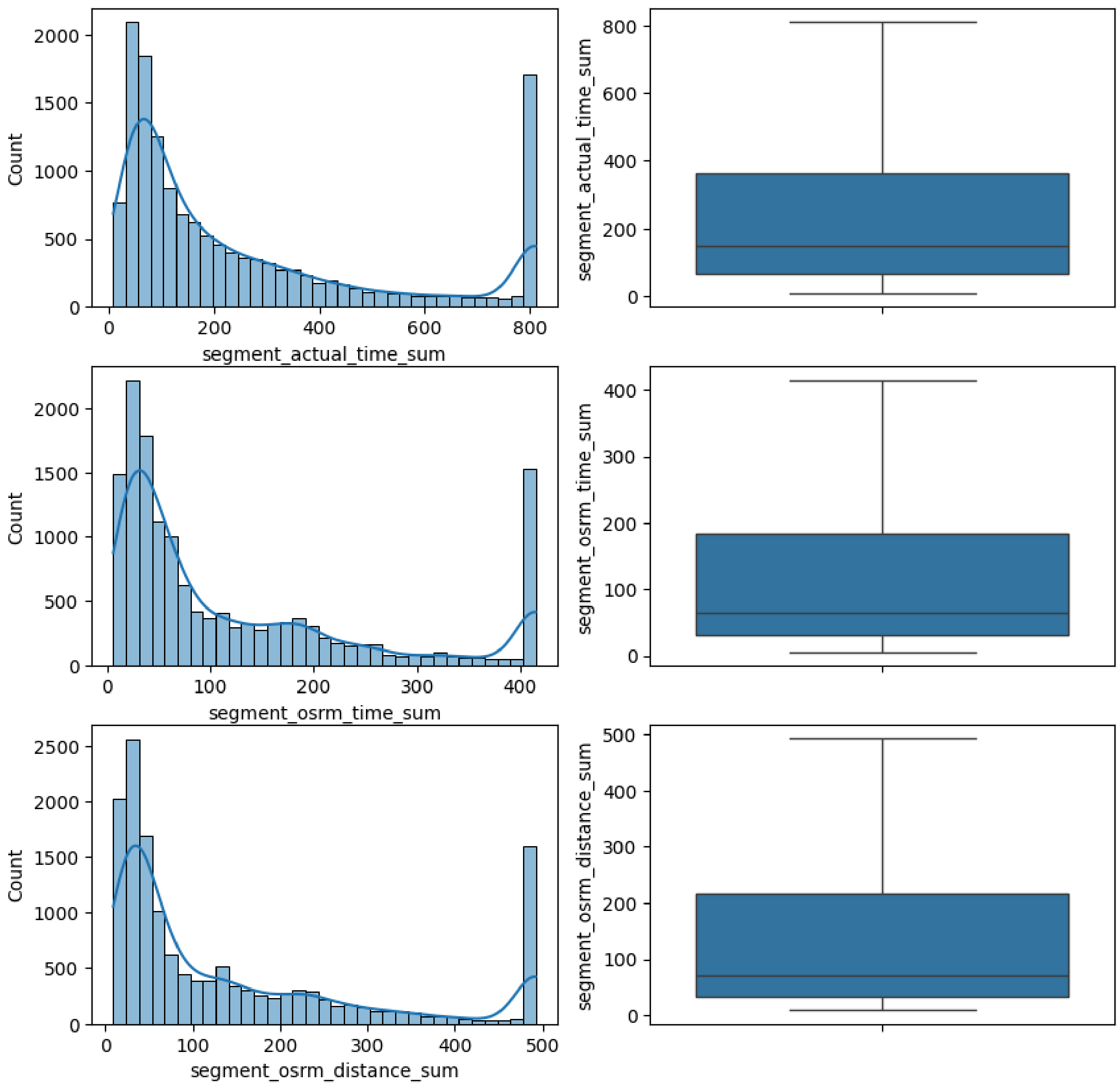
```
warnings.filterwarnings("ignore")
fig, axs = plt.subplots(3, 2,  figsize=(10,10))
sns.histplot(ax=axs[0,0],data= trip['segment_actual_time_sum'],kde=True)
sns.boxplot(ax=axs[0,1],data=trip['segment_actual_time_sum'])

sns.histplot(ax=axs[1,0],data= trip['segment_osrm_time_sum'],kde=True)
sns.boxplot(ax=axs[1,1],data= trip['segment_osrm_time_sum'])

sns.histplot(ax=axs[2,0],data= trip['segment_osrm_distance_sum'],kde=True)
sns.boxplot(ax=axs[2,1],data= trip['segment_osrm_distance_sum'])


plt.show()
```


Count
segment_actual_time_sum
Count
segment_osrm_time_sum
Count
segment_osrm_distance_sum
segment_actual_time_sum
segment_osrm_time_sum
segment_osrm_distance_sum

```
def clip_value_helper(row,cl, Q1,Q3, minval, maxval):
    #     Q1=row[cl].quantile(0.25)
    #     Q3=row[cl].quantile(0.75)
    #     minval=min(row[cl])
    #     maxval=max(row[cl])
    IQR=Q3-Q1
    if row[cl]<Q1-1.5*IQR:
        return min(minval, Q1-1.5*IQR)
    elif row[cl] > Q3+1.5*IQR:
        return min(maxval, Q3+1.5*IQR)
    else:
        return row[cl]


for cl in ['actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time_sum'
    Q1=trip[cl].quantile(0.25)
    Q3=trip[cl].quantile(0.75)
    minval=min(trip[cl])
    maxval=max(trip[cl])
    trip[cl]=trip.apply(lambda row:clip_value_helper(row,cl,Q1,Q3,minval, maxval)


import warnings
import seaborn as sns
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
fig, axs = plt.subplots(3, 2,  figsize=(10,10))
sns.histplot(ax=axs[0,0],data= trip['actual_time'],kde=True)
sns.boxplot(ax=axs[0,1],data=trip['actual_time'])

sns.histplot(ax=axs[1,0],data= trip['osrm_time'],kde=True)
sns.boxplot(ax=axs[1,1],data= trip['osrm_time'])

sns.histplot(ax=axs[2,0],data= trip['osrm_distance'],kde=True)
sns.boxplot(ax=axs[2,1],data= trip['osrm_distance'])


plt.show()
```


Count
actual_time
actual_time
Count
osrm_time
osrm_time
Count
osrm_distance
osrm_distance

```
warnings.filterwarnings("ignore")
fig, axs = plt.subplots(3, 2,  figsize=(10,10))
sns.histplot(ax=axs[0,0],data= trip['segment_actual_time_sum'],kde=True)
sns.boxplot(ax=axs[0,1],data=trip['segment_actual_time_sum'])

sns.histplot(ax=axs[1,0],data= trip['segment_osrm_time_sum'],kde=True)
sns.boxplot(ax=axs[1,1],data= trip['segment_osrm_time_sum'])

sns.histplot(ax=axs[2,0],data= trip['segment_osrm_distance_sum'],kde=True)
sns.boxplot(ax=axs[2,1],data= trip['segment_osrm_distance_sum'])


plt.show()
```



```
#Here We will use label encoder for encoding route_type column
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, One
le = LabelEncoder()
trip['route_type'] = le.fit_transform(trip['route_type'])
trip['route_type'].value_counts()
```

	count
route_type	
0	8906
1	5881

dtype: int64

5. Hypothesis Testing:

1. Perform hypothesis testing / visual analysis between :

   a. actual_time aggregated value and OSRM time aggregated value.

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

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

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

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

```
trip[['actual_time','osrm_time']]
```

	actual_time	osrm_time
0	817.0	376.5
1	143.0	68.0
2	817.0	376.5
3	59.0	15.0
4	341.0	117.0
...	...	...
14782	83.0	62.0
14783	21.0	12.0
14784	282.0	48.0
14785	264.0	179.0
14786	275.0	68.0

14787 rows × 2 columns

```
from scipy.stats import ttest_ind,ttest_1samp,ttest_rel
```

```
# we will use ttest  sample test to know if there is significant difference in ac

# HO : mean Actual time to deliver package from source to destination is lesser t

# HA: mean Actual time to deliver package from source to destination is greater t

ttest_value,p_value= ttest_ind(trip['actual_time'],trip['osrm_time'],equal_var=Fa

print("ttest statistic value ", ttest_value)
print("p-value ", p_value)

if(p_value<0.05):
  print("Reject Null Hypothesis, indicates that mean actual time is greater than
else:
  print("Fail to reject  Null Hypothesis,indicates that mean actual time is less
```

```
ttest statistic value  63.30545280574021
p-value  0.0
Reject Null Hypothesis, indicates that mean actual time is greater than the m
```

```
trip[['actual_time','segment_actual_time_sum']]
```

	actual_time	segment_actual_time_sum
0	817.0	811.0
1	143.0	141.0
2	817.0	811.0
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

```
# we will use ttest sample test to know if there is significant difference in act

# HO : mean Actual aggregated trip time to deliver package from source to destina

# HA: mean Actual aggregated trip time to deliver package from source to destinat

ttest_value,p_value= ttest_ind(trip['actual_time'],trip['segment_actual_time_sum'

print("ttest statistic value ", ttest_value)
print("p-value ", p_value)

if(p_value<0.05):
  print("Reject Null Hypothesis, indicates that mean actual time is lesser than t
else:
  print("Fail to reject  Null Hypothesis,indicates that mean actual time is great
```

```
ttest statistic value  0.7566645099710447
p-value  0.7753715448578429
Fail to reject  Null Hypothesis,indicates that mean actual time is greater thi
```

```
trip[['osrm_distance','segment_osrm_distance_sum']]
```

	osrm_distance	segment_osrm_distance_sum
0	470.47515	492.533225
1	85.11100	84.189400
2	470.47515	492.533225
3	19.68000	19.876600
4	146.79180	146.791900
...	...	...
14782	73.46300	64.855100
14783	16.08820	16.088300
14784	58.90370	104.886600
14785	171.11030	223.532400
14786	80.57870	80.578700

14787 rows × 2 columns

```
# We will use ttest_ind test to know if there significant difference in  OSRM dis

# H0: Mean osrm_distance aggregated value is less than the segment _osrm_distance
# Ha: Mean osrm_distance aggregated value is greater than sement_osrm_distance_su

ttest_value,p_value= ttest_ind(trip['osrm_distance'],trip['segment_osrm_distance_

print("ttest statistic value ", ttest_value)
print("p-value ", p_value)

if(p_value<0.05):
  print("Reject Null Hypothesis, indicates that mean osrm_distance is greater tha
else:
  print("Fail to reject  Null Hypothesis,indicates that mean osrm_distance is les
```

```
ttest statistic value  -4.735638441691023
p-value  0.9999989030967289
Fail to reject  Null Hypothesis,indicates that mean osrm_distance is less thai
```

```
trip[['osrm_time','segment_osrm_time_sum']]
```

	osrm_time	segment_osrm_time_sum
0	376.5	415.0
1	68.0	65.0
2	376.5	415.0
3	15.0	16.0
4	117.0	115.0
...	...	...
14782	62.0	62.0
14783	12.0	11.0
14784	48.0	88.0
14785	179.0	221.0
14786	68.0	67.0

14787 rows × 2 columns

```
# We will use ttest_ind test to know if there significant difference in  osrm_tim

# H0: Mean osrm_time aggregated value is less than the segment_osrm_time_sum
# Ha: Mean osrm_time aggregated value is greater than sement_osrm_time_sum

ttest_value,p_value= ttest_ind(trip['osrm_time'],trip['segment_osrm_time_sum'],eq

print("ttest statistic value ", ttest_value)
print("p-value ", p_value)

if(p_value<0.05):
  print("Reject Null Hypothesis, indicates that mean osrm_time is greater than th
else:
  print("Fail to reject  Null Hypothesis,indicates that mean osrm_time is less th
```

```
ttest statistic value  -7.807941938846417
p-value  0.9999999999999997
Fail to reject  Null Hypothesis,indicates that mean osrm_time is less than th
```

```
#Here We will use label encoder for encoding route_type column
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, One
le = LabelEncoder()
trip['route_type'] = le.fit_transform(trip['route_type'])
trip['route_type'].value_counts()
```

	count
route_type	
0	8906
1	5881

dtype: int64

```
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']
```

```
scaler=MinMaxScaler()

trip[num_cols]=scaler.fit_transform(trip[num_cols])
```

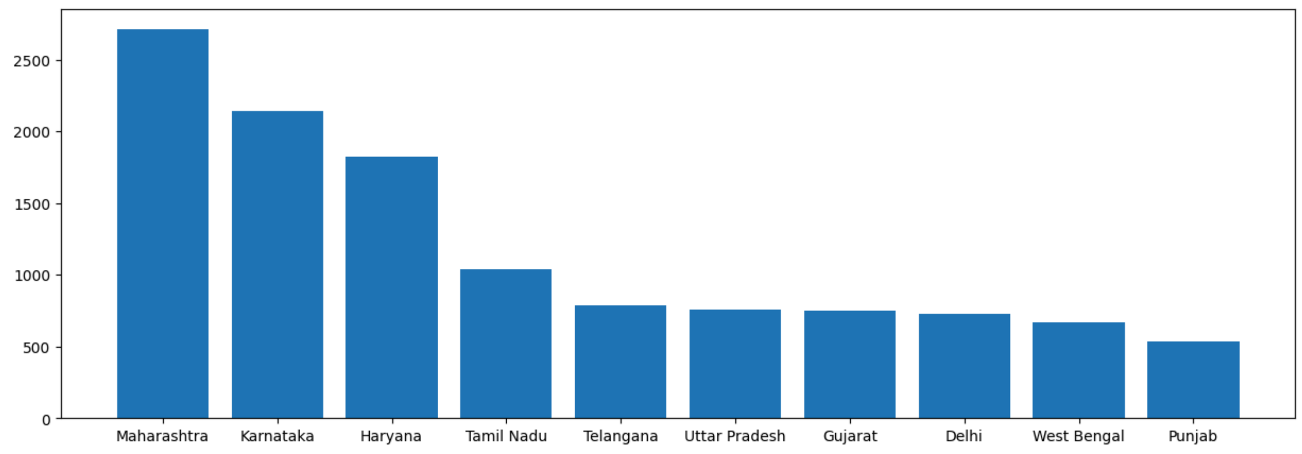
```
trip[num_cols]
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	os
0	0.283937	0.374613	1.000000	
1	0.019937	0.029476	0.165842	
2	0.496508	0.880999	1.000000	
3	0.009778	0.003753	0.061881	
4	0.088127	0.054395	0.410891	
...	...	...	...	
14782	0.029714	0.022392	0.091584	
14783	0.004698	0.002990	0.014851	
14784	0.050540	0.013631	0.337871	
14785	0.041143	0.057736	0.315594	
14786	0.041905	0.026213	0.329208	

14787 rows × 9 columns

6. Business Insights & Recommendations

```
# Top 10 states from where Delhivery is  getting orders
plt.figure(figsize=(15,5))
plt.bar(trip['source_state'].value_counts()[:10].index,trip['source_state'].value
plt.show()
```

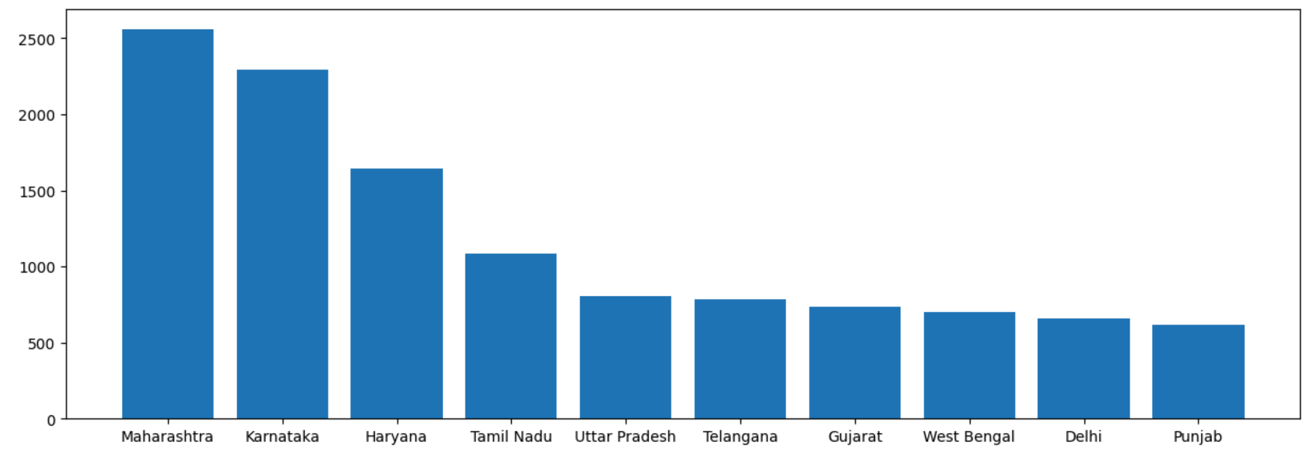


```
trip['destination_state'].value_counts()
```

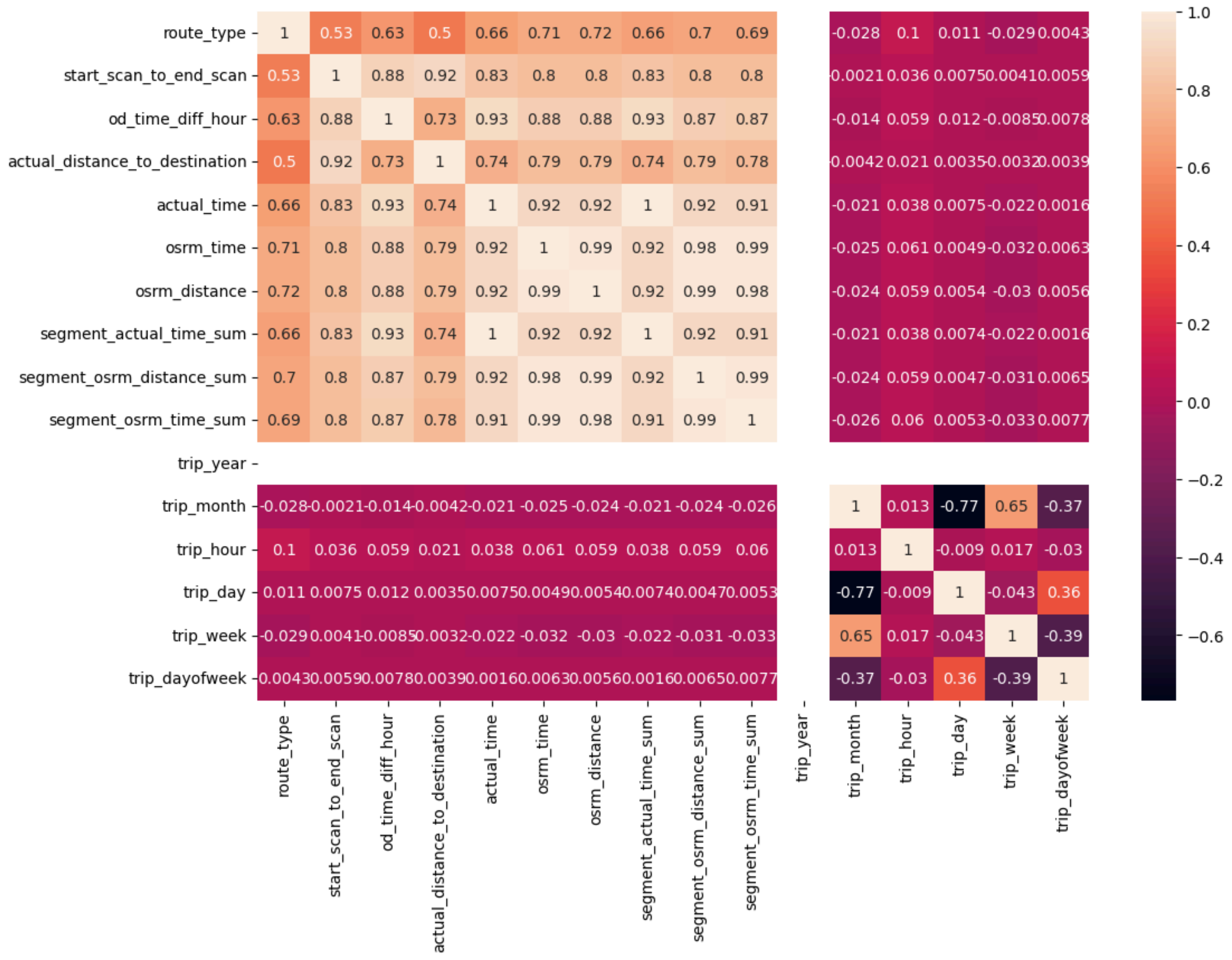
destination_state	count
Maharashtra	2561
Karnataka	2294
Haryana	1640
Tamil Nadu	1084
Uttar Pradesh	805
Telangana	784
Gujarat	734
West Bengal	697
Delhi	657
Punjab	617
Rajasthan	550
Andhra Pradesh	442
Bihar	367
Madhya Pradesh	350
Kerala	270
Assam	232
Jharkhand	181
Uttarakhand	122
Orissa	119
Chandigarh	65
Goa	52
Chhattisgarh	43
Himachal Pradesh	42
Arunachal Pradesh	25
Jammu & Kashmir	20
Dadra and Nagar Haveli	17
Meghalaya	8
Mizoram	6
Nagaland	1
Tripura	1
Daman & Diu	1

dtype: int64

```
## Top 10 states from destination states
plt.figure(figsize=(15,5))
plt.bar(trip['destination_state'].value_counts()[:10].index,trip['destination_sta
plt.show()
```



```
# Select only numeric columns before calculating correlation
numeric_trip = trip.select_dtypes(include=['number'])

# Calculate and plot the correlation matrix
plt.figure(figsize=(12, 8))  # Adjust figure size as needed
sns.heatmap(numeric_trip.corr(), annot=True)
plt.show()
```



Business Insights

By doing Hypothesis testing between osrm data and actual data, we can observe that mean of both data is not the same.

Distance and time attributes are highly correlated, so its obvious that distance between places will matter in speedy delivery

Maximum orders are found from Maharashtra, so we can say more customers in the state.

Minimum trips are from North-Eastern states so business needs improvement in that states

Recommendations

From the above analysis, It can be observed that the actual time taken for delivery is higher compared to osrm time. So we can optimize our services using osrm.

In Maharashtra, we have the highest number of trips, so we should increase outlets in the state.

In North-Eastern states, we have very less business, so we need to optimize their condition and also provide marketing to increase services.

Revisit information fed to routing engine for trip planning. Check for discrepancies with transporters, if the routing engine is configured for optimum results.

If Actual delivery time is higher than osrm time then should focus on hops which are causing delays, if delays are related to processing or logistic that should be quickly fixed.

If Issue is not related to delivery and logistic process then should focus on identifying best route to move packages quickly.

Double-click (or enter) to edit