Business Case: Delhivery - Feature Engineering

Problem Statement

- Delhivery, India's largest integrated logistics provider, aims to enhance its delivery efficiency and operational performance. With a vision to become the backbone of commerce, Delhivery is focused on building a robust operating system through world-class infrastructure, high-quality logistics operations, and advanced technology capabilities.
- To stay ahead of competitors and bridge performance gaps, the company wants to better understand and analyze data emerging from its data engineering pipelines. This requires identifying both well-performing and underperforming delivery routes, enabling the business to optimize operations and improve decision-making.

Objective

The objective of this project is to process and analyze the delivery data to uncover meaningful patterns and insights, and to provide actionable recommendations help Delhivery outperform competitors. By leveraging data, the goal is to make operations smarter, faster, and more cost-effective — from optimizing delivery routes to improving demand forecasting. Specifically, the analysis aims to:

- Clean, sanitize, and transform raw data into meaningful features for analysis.
- Support the data science team by preparing data suitable for modeling.
- Aggregation and Grouping: Group data by keys for detailed, level-wise analysis.
- Hypothesis Testing: Compare actual vs. predicted time and distance metrics to validate performance gaps.

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
delhi_df=pd.read_csv('delhivery_dataset.csv')
```

```
delhi_df.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cent
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A

5 rows × 24 columns

Basic Analysis of Data

```
delhi_df.shape
```

```
(144867, 24)
```

```
delhi_df['trip_uuid'].nunique()
```

```
14817
```

```
delhi_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', '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')
```

```
delhi_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19130 entries, 0 to 19129
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            19130 non-null  object
 1   trip_creation_time              19130 non-null  object
 2   route_schedule_uuid             19130 non-null  object
 3   route_type                      19130 non-null  object
 4   trip_uuid                       19130 non-null  object
 5   source_center                   19130 non-null  object
 6   source_name                     19069 non-null  object
 7   destination_center              19129 non-null  object
```

```
 8   destination_name                19087 non-null  object
 9   od_start_time                   19129 non-null  object
 10  od_end_time                     19129 non-null  object
 11  start_scan_to_end_scan          19129 non-null  float64
 12  is_cutoff                       19129 non-null  object
 13  cutoff_factor                   19129 non-null  float64
 14  cutoff_timestamp                19129 non-null  object
 15  actual_distance_to_destination  19129 non-null  float64
 16  actual_time                     19129 non-null  float64
 17  osrm_time                       19129 non-null  float64
 18  osrm_distance                   19129 non-null  float64
 19  factor                          19129 non-null  float64
 20  segment_actual_time             19129 non-null  float64
 21  segment_osrm_time               19129 non-null  float64
 22  segment_osrm_distance           19129 non-null  float64
 23  segment_factor                  19129 non-null  float64
dtypes: float64(11), object(13)
memory usage: 3.5+ MB
```

```
#checking duplicates
print("Number of duplicates:", delhi_df.duplicated().sum())
```

```
Number of duplicates: 0
```

```
delhi_df.describe(include='all').T
```

	count	unique	top	freq	mean	s
data	144867	2	training	104858	NaN	Na
trip_creation_time	144867	14817	2018-09-22 04:55:04.835022	101	NaN	Na
route_schedule_uuid	144867	1504	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	1812	NaN	Na
route_type	144867	2	FTL	99660	NaN	Na
trip_uuid	144867	14817	trip-153759210483476123	101	NaN	Na
source_center	144867	1508	IND000000ACB	23347	NaN	Na
source_name	144574	1498	Gurgaon_Bilaspur_HB (Haryana)	23347	NaN	Na
destination_center	144867	1481	IND000000ACB	15192	NaN	Na
destination_name	144606	1468	Gurgaon_Bilaspur_HB (Haryana)	15192	NaN	Na
od_start_time	144867	26369	2018-09-21 18:37:09.322207	81	NaN	Na
od_end_time	144867	26369	2018-09-24 09:59:15.691618	81	NaN	Na
start_scan_to_end_scan	144867.0	NaN	NaN	NaN	961.262986	1037.0127
is_cutoff	144867	2	True	118749	NaN	Na
cutoff_factor	144867.0	NaN	NaN	NaN	232.926567	344.7555
cutoff_timestamp	144867	93180	2018-09-24 05:19:20	40	NaN	Na
actual_distance_to_destination	144867.0	NaN	NaN	NaN	234.073372	344.9900
actual_time	144867.0	NaN	NaN	NaN	416.927527	598.1036
osrm_time	144867.0	NaN	NaN	NaN	213.868272	308.0110
osrm_distance	144867.0	NaN	NaN	NaN	284.771297	421.1192
factor	144867.0	NaN	NaN	NaN	2.120107	1.7154
segment_actual_time	144867.0	NaN	NaN	NaN	36.196111	53.5711

The total number of rows is 144867, but source_name and destination_name have fewer non-null entries (144574 and 144606). That means these columns have missing values.

```
#Checking missing values
delhi_df.isnull().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

```
missing_per = (delhi_df.isnull().mean()*100).round(2)
missing_per = missing_per[missing_per > 0]
missing_per
```

	0
source_name	0.20
destination_name	0.18

dtype: float64

handling missing values

- The dataset contains missing values in the source_name and destination_name columns, accounting for 0.2% and 0.18% respectively.
- Since the proportion of missing values is very low relative to the overall dataset, dropping these rows is unlikely to impact the analysis significantly.

```
df=delhi_df.copy()
df = df.dropna(subset=['source_name','destination_name'])
```

```
print("Number of missing value:", int(df.isnull().sum().sum()))
```

```
Number of missing value: 0
```

Data Type Conversion

```
date_col = ['trip_creation_time','od_start_time', 'od_end_time','cutoff_timestamp']
for i in date_col:
     df[i] = pd.to_datetime(df[i], format = 'mixed')
```

```
cat = ['data','route_type' ]
for i in cat:
  df[i] = df[i].astype('category')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  category
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  category
 4   trip_uuid                       144316 non-null  object
 5   source_center                   144316 non-null  object
 6   source_name                     144316 non-null  object
 7   destination_center              144316 non-null  object
 8   destination_name                144316 non-null  object
 9   od_start_time                   144316 non-null  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144316 non-null  float64
 12  is_cutoff                       144316 non-null  bool
 13  cutoff_factor                   144316 non-null  int64
 14  cutoff_timestamp                144316 non-null  datetime64[ns]
 15  actual_distance_to_destination  144316 non-null  float64
 16  actual_time                     144316 non-null  float64
 17  osrm_time                       144316 non-null  float64
 18  osrm_distance                   144316 non-null  float64
 19  factor                          144316 non-null  float64
 20  segment_actual_time             144316 non-null  float64
 21  segment_osrm_time               144316 non-null  float64
 22  segment_osrm_distance           144316 non-null  float64
 23  segment_factor                  144316 non-null  float64
dtypes: bool(1), category(2), datetime64[ns](4), float64(10), int64(1), object(6)
memory usage: 24.6+ MB
```

```
df[df['trip_uuid'] ==  'trip-153741093647649320']
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_cent
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121A
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620A
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620A
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620A
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620A
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620A

10 rows × 24 columns

Since a single trip_uuid can have multiple connection points or segments, we merge or aggregate these segments to treat the entire trip as one complete delivery segment.

Groupby and aggregations on segment

```
df['segment_key'] = df['trip_uuid'].astype(str) + "_" + df['source_center'] + "_" + df['destination_
segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time']

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


create_segment_dict = {
    'data': 'first',
    'trip_creation_time': '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_level_df = df.groupby('segment_key').agg(create_segment_dict).reset_index()
segment_level_df = segment_level_df.sort_values(by=['segment_key', 'od_end_time']).reset_index()
```

```
segment_level_df.head()
```

	index	segment_key	data	trip_creation_time	route_type
0	0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	FTL
1	1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	FTL
2	2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	Carting
3	3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	Carting
4	4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	FTL

Next steps: Generate code with segment_level_df | View recommended plots | New interactive sheet

Feature Engineering

Calculate the time taken between od_start_time and od_end_time

```
segment_level_df['od_time_diff_hour']=(segment_level_df['od_end_time']-
                                       segment_level_df['od_start_time']).dt.total_seconds()/3600
```

Extracting features from source name and destination

```
segment_level_df['source_name'].head(10)
```

	source_name
0	Delhi_Lajpat_IP (Delhi)
1	FBD_Balabhgarh_DPC (Haryana)
2	Narsinghpur_KndliDPP_D (Madhya Pradesh)
3	Gadarwara_MPward_D (Madhya Pradesh)
4	Sonari_Central_DPP_1 (Assam)
5	Hyderabad_North_D_2 (Telangana)
6	Medchal_MROoffce_D (Telangana)
7	Dindigul_Central_D_1 (Tamil Nadu)
8	Kodaikanal_Athithnr_DC (Tamil Nadu)
9	Batlagundu_RTOofice_D (Tamil Nadu)

dtype: object

```
city_mapping = {
    'Ahd':'Ahmedabad',
    'Bangalore': 'Bengaluru',
    'Blr':'Bengaluru',
     'Hbr Layout Pc':'Bengaluru',
    'Mumbai Hub':'Mumbai',
    'Maa':'Chennai',
    'Hyd':'Hyderabad',
    'Ccu':'Kolkata',
    'Ggn':'Gurgaon',
    'Del':'Delhi',
    'Fbd':'Faridabad,'
}
# SOURCE
segment_level_df['source_state'] = segment_level_df['source_name'].str.extract(r'\((.*?)\)')
source_clean = segment_level_df['source_name'].str.replace(r'\s*\(.*?\)', '', regex=True)
source_clean = source_clean.str.strip().str.title()
source_clean = source_clean.replace(city_mapping)
source_split = source_clean.str.split('_')
segment_level_df['source_city'] = source_split.str.get(0).replace(city_mapping)
segment_level_df['source_code'] = source_split.str.get(1)

# DESTINATION
segment_level_df['destination_state'] = segment_level_df['destination_name'].str.extract(r'\((.*?)\)
cleaned = segment_level_df['destination_name'].str.replace(r'\s*\(.*?\)', '', regex=True)
cleaned = cleaned.str.strip().str.title()
cleaned = cleaned.replace(city_mapping)
split_parts = cleaned.str.split('_')
segment_level_df['destination_city'] = split_parts.str.get(0).replace(city_mapping)
segment_level_df['destination_code'] = split_parts.str.get(1)
```

Extract features month, year and day

```
segment_level_df['year'] = segment_level_df['trip_creation_time'].dt.year
segment_level_df['month']=segment_level_df['trip_creation_time'].dt.month_name()
segment_level_df['day']=segment_level_df['trip_creation_time'].dt.day_name()
```

Removing Redundant Columns

```
segment_level_df.drop(columns=['trip_creation_time','source_name', 'destination_name'], inplace=True
```

Grouping and Aggregating at Trip-level

```
trip_dict = {'segment_key':'first',
             'data':'first',
             'route_type':'first',
             'start_scan_to_end_scan':'sum',
             'actual_distance_to_destination':'sum',
             'actual_time':'first',
             'osrm_time':'first',
             'osrm_distance':'first',
             'segment_actual_time_sum':'sum',
             'segment_osrm_distance_sum':'sum',
             'segment_osrm_time_sum':'sum',
             'od_time_diff_hour':'sum',
             'source_state':'first',
             'source_city':'first',
             'source_code':'first',
             'destination_state':'first',
             'destination_code':'first',
             'destination_city':'first',
             'year':'first', 'month':'first', 'day':'first'}
```

```
trip_level_df = segment_level_df.groupby('trip_uuid').agg(trip_dict)
trip_level_df.head()
```

	segment_key	data	route_type	start_sc
trip_uuid				
trip-153671041653548748	trip-153671041653548748_IND209304AAA_IND000000ACB	training	FTL	
trip-153671042288605164	trip-153671042288605164_IND561203AAB_IND562101AAA	training	Carting	
trip-153671043369099517	trip-153671043369099517_IND000000ACB_IND160002AAC	training	FTL	
trip-153671046011330457	trip-153671046011330457_IND400072AAB_IND401104AAA	training	Carting	
trip-153671052974046625	trip-153671052974046625_IND583101AAA_IND583201AAA	training	FTL	

5 rows × 21 columns

Statistical summary

```
trip_level_df.describe(include='O').T
```

	count	unique	top	freq
segment_key	2285	2285	trip-153861118270144424_IND583119AAA_IND583101AAA	1
source_state	2285	27	Maharashtra	423
source_city	2285	373	Bengaluru	283
source_code	2209	395	Bilaspur	163
destination_state	2285	27	Maharashtra	409
destination_code	2182	439	Central	117
destination_city	2285	454	Bengaluru	272
month	2285	2	September	2022
day	2285	7	Wednesday	392

Observation

- There are two unique values in the data column, with **'training'** being the most frequent.
- There are two route types and the **'Carting'** type occurs most frequently in this dataset.
- The majority of deliveries originated from the state of **Maharashtra**, which appears most frequently in the source_state column.
- The most common delivery city is **Bengaluru**, based on the highest frequency in the destination_city column.
- There are only two unique months represented in the dataset, with **September** accounting for the majority of deliveries.
- **Wednesda**y is the most frequent delivery day, indicating a peak mid-week delivery trend.

```
trip_level_df.describe().T
```

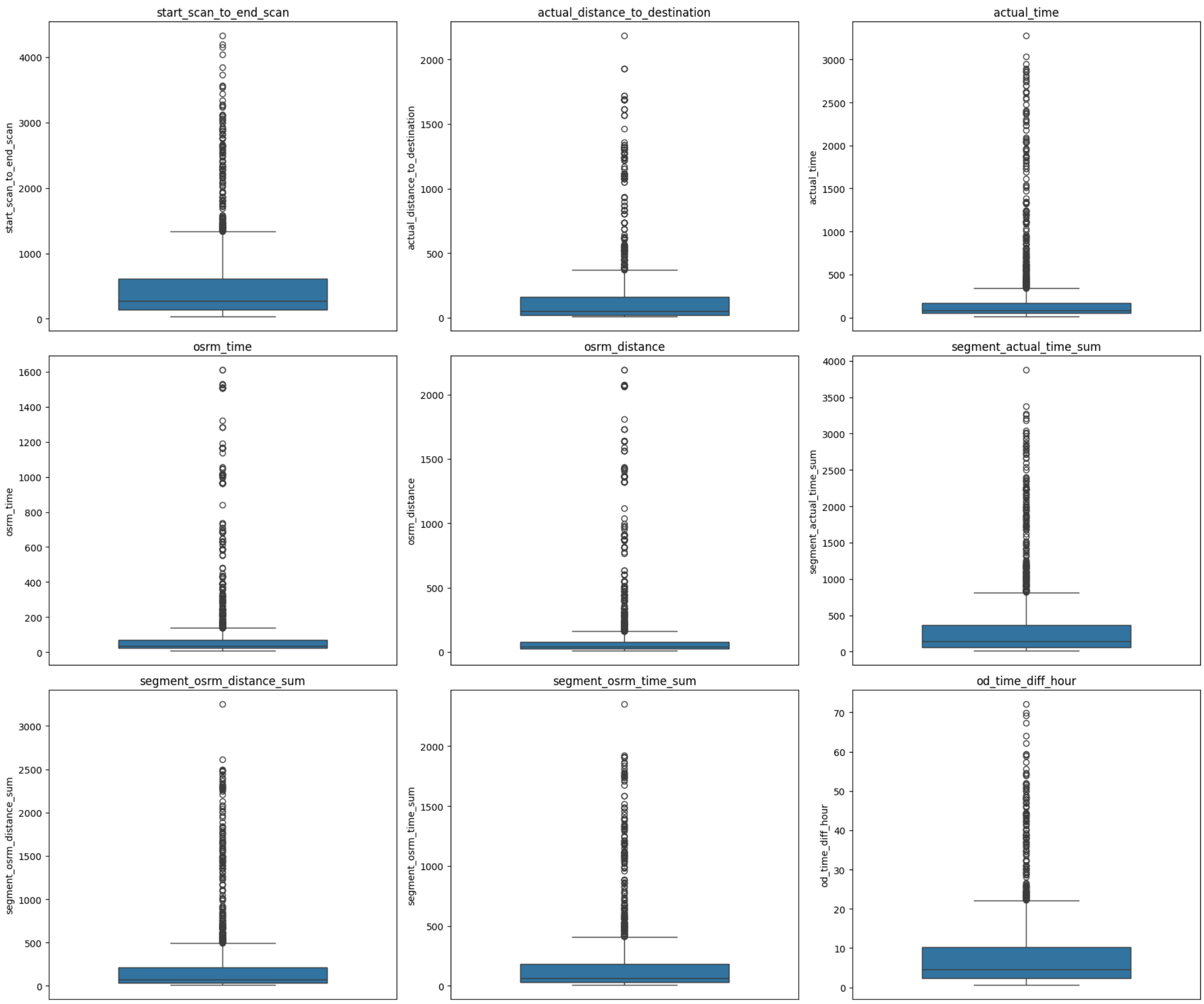
	count	mean	std	min	25%	50%
start_scan_to_end_scan	2285.0	508.899781	617.375059	34.000000	143.000000	269.000000
actual_distance_to_destination	2285.0	158.397730	288.570555	9.006255	22.042272	47.771576
actual_time	2285.0	211.357549	417.606623	11.000000	50.000000	83.000000
osrm_time	2285.0	98.632385	209.768794	7.000000	23.000000	37.000000
osrm_distance	2285.0	125.479342	288.023162	9.136400	25.673500	40.283300
segment_actual_time_sum	2285.0	336.092341	514.428163	11.000000	63.000000	140.000000
segment_osrm_distance_sum	2285.0	214.279915	392.409966	9.136400	31.837000	68.728400
segment_osrm_time_sum	2285.0	174.017505	295.657553	7.000000	30.000000	65.000000
od_time_diff_hour	2285.0	8.496773	10.292473	0.575371	2.385239	4.499494
year	2285.0	2018.000000	0.000000	2018.000000	2018.000000	2018.000000

Observation

- It can be observed that the actual delivery time **(actual_time, mean ≈ 228 minutes)** is significantly higher than the predicted time **(osrm_time, mean ≈ 104 minutes)**.
- The time difference between dispatch and reaching the customer (od_time_diff_hour) ranges from ~0.39 hours (23 minutes) to ~131.6 hours (5.5 days).
- Many time and distance-related columns show a significant gap between the mean and median, and also have high standard deviations. These indicate the presence of outliers.

Outlier Detection

```
num_col=['start_scan_to_end_scan','actual_distance_to_destination','actual_time',
         'osrm_time','osrm_distance','segment_actual_time_sum','segment_osrm_distance_sum',
         'segment_osrm_time_sum','od_time_diff_hour']

plt.figure(figsize=(18,15))
for i, col in enumerate(num_col,1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=trip_level_df[col],width = 0.6)
    plt.title(col, fontsize=12)
    plt.xticks([])
plt.tight_layout()
```


start_scan_to_end_scan
actual_distance_to_destination
actual_time
osrm_time
osrm_distance
segment_actual_time_sum
segment_osrm_distance_sum
segment_osrm_time_sum
od_time_diff_hour

Insights

- Time and distance-related columns show a high number of outliers and narrow IQRs.
- However, there are many values far outside this small range.
- This means most trips have similar times or distances, but there are also many trips that are unusually long or short, which are outside the expected pattern.
- To improve interpretability, we use the IQR method to filter out these extreme values.

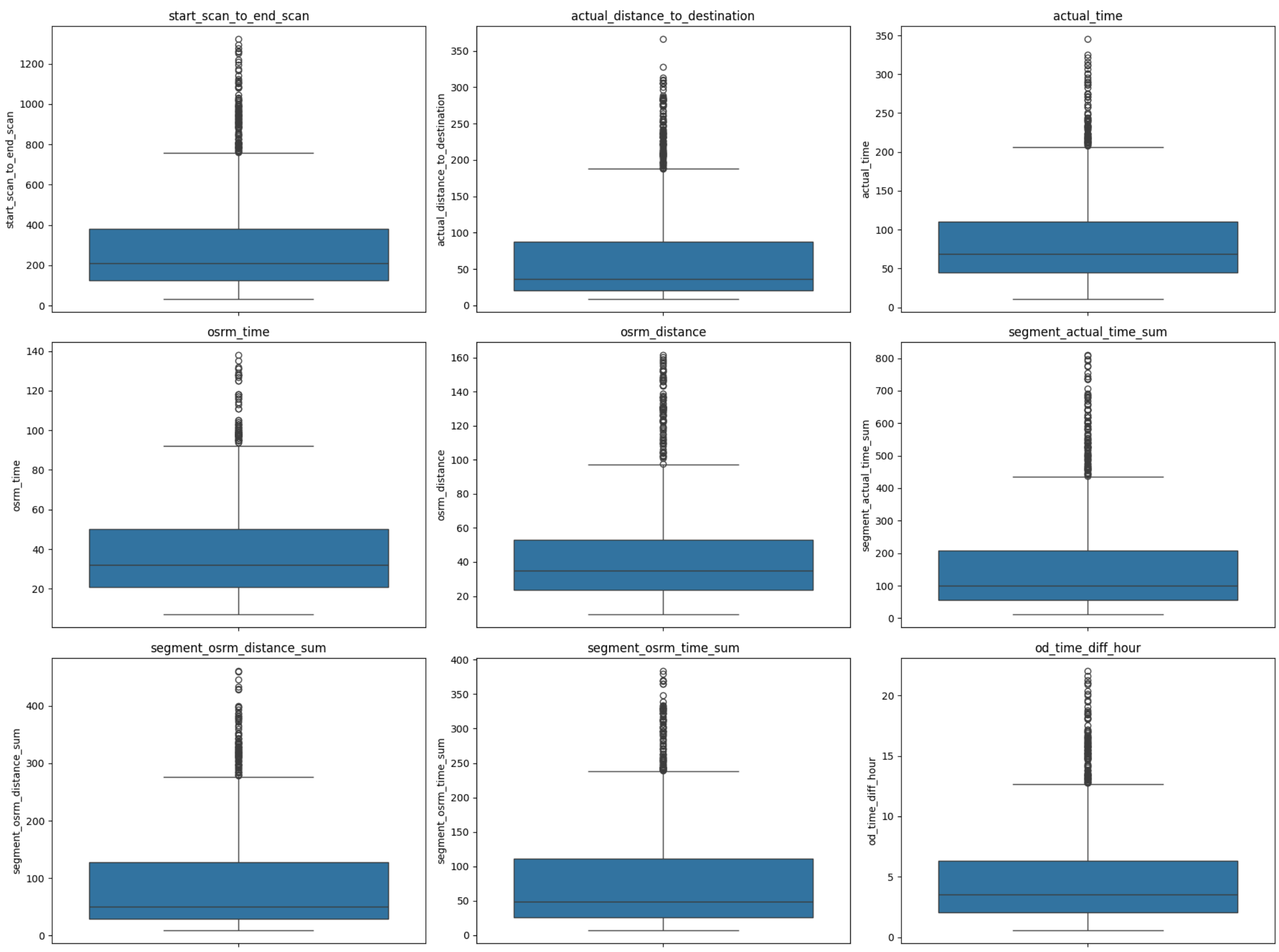
Handle the outliers using the IQR method.

```
iqr_df=trip_level_df[num_col]
Q1=iqr_df.quantile(0.25)
Q2 = iqr_df.quantile(0.50)
Q3 = iqr_df.quantile(0.75)
iqr = Q3-Q1
lower = Q1 - (1.5 * iqr)
upper = Q3 + (1.5 * iqr)
```

```
filtered_df = iqr_df[~((iqr_df < lower) | (iqr_df > upper)).any(axis=1)]
```

```
cols = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time',
        'osrm_time', 'osrm_distance', 'segment_actual_time_sum', 'segment_osrm_distance_sum',
        'segment_osrm_time_sum', 'od_time_diff_hour']

plt.figure(figsize=(18, 15))
plt.suptitle("Box plot after removing outliers", fontsize=20)

for i, c in enumerate(cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=filtered_df[c])
    plt.title(c)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

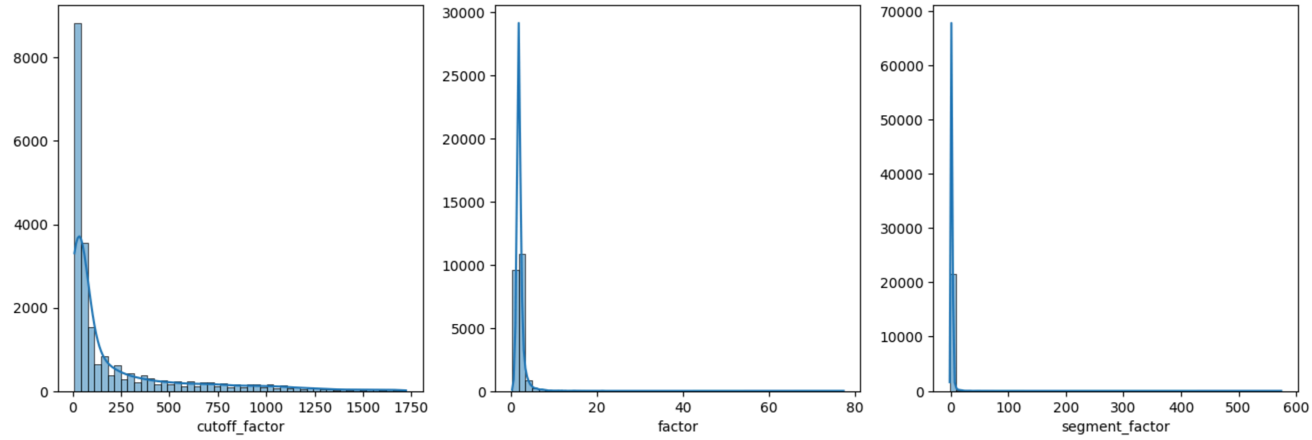
Box plot after removing outliers



Univariate Analysis

```
c=['cutoff_factor','factor','segment_factor']
plt.figure(figsize=(16,5))
```

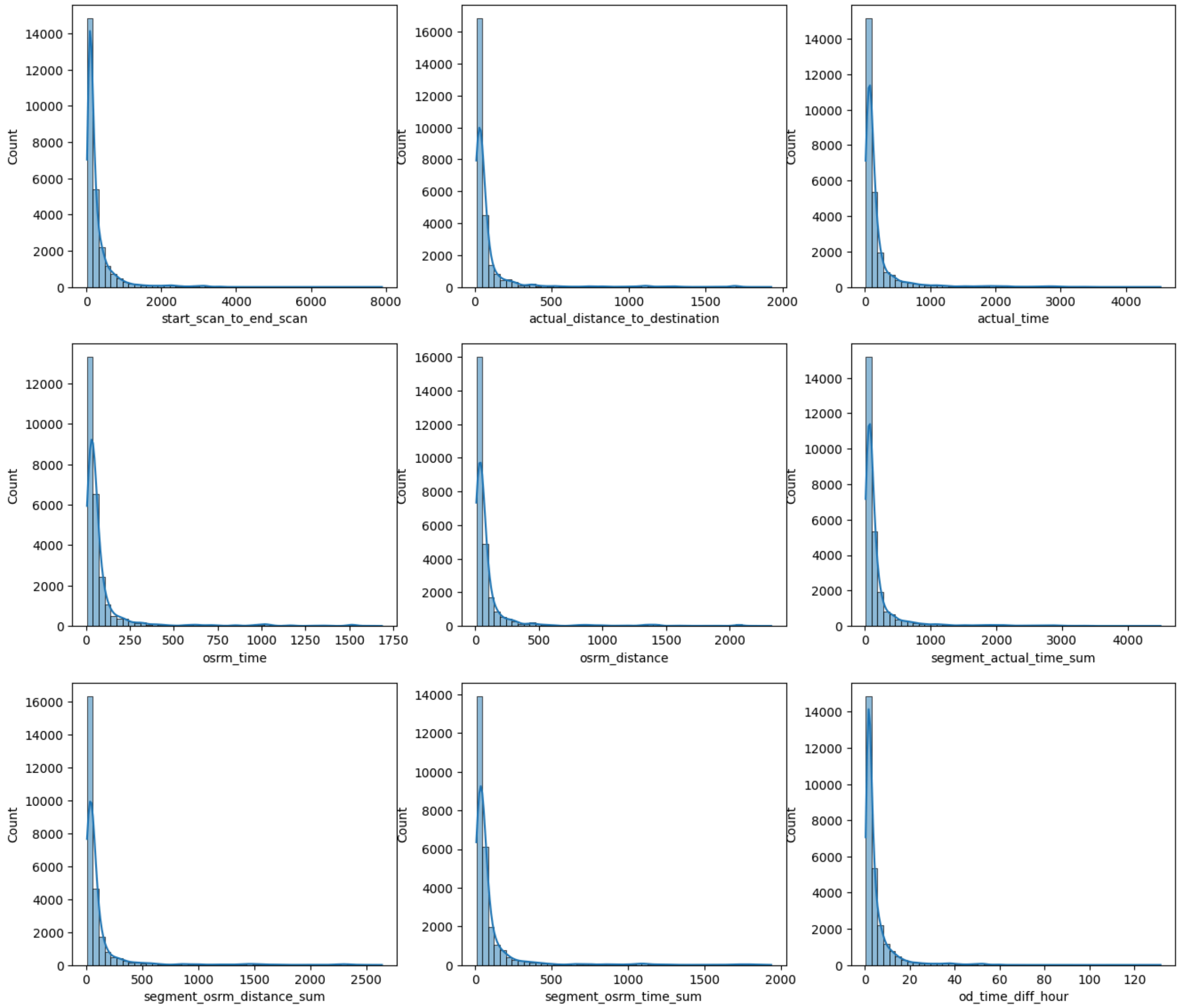
```
for i,c in enumerate(c,1):
  plt.subplot(1,3,i)
  sns.histplot(x=c,data=df,kde=True,bins=50)
  plt.ylabel("")
```



Insights

- All three distributions are right-skewed (positively skewed), meaning most of the values are concentrated on the lower end (closer to 0), with a long tail extending toward higher values.
- Spikes near 0 in all three plots suggest that a large number of records have small values for these variables.
- However, the exact meaning and role of these columns in the delivery process is unclear.

```
hist_col=['start_scan_to_end_scan','actual_distance_to_destination','actual_time',
          'osrm_time','osrm_distance','segment_actual_time_sum','segment_osrm_distance_sum',
          'segment_osrm_time_sum','od_time_diff_hour']

plt.figure(figsize=(16,14))
for i,hist_col in enumerate(hist_col,1):
  plt.subplot(3,3,i)
  sns.histplot(x=hist_col,data=segment_level_df,kde=True,bins=50)
```


Count
start_scan_to_end_scan
actual_distance_to_destination
actual_time
osrm_time
osrm_distance
segment_actual_time_sum
segment_osrm_distance_sum
segment_osrm_time_sum
od_time_diff_hour

Insights

- All of the time and distance-related distributions are right-skewed, indicating that most deliveries are completed within a short duration. However, a smaller number of deliveries take significantly longer, which may be due to external factors such as weather conditions, traffic congestion, or road quality.
- This pattern highlights the need for further in-depth analysis to identify the root causes behind these unusually long delivery times.

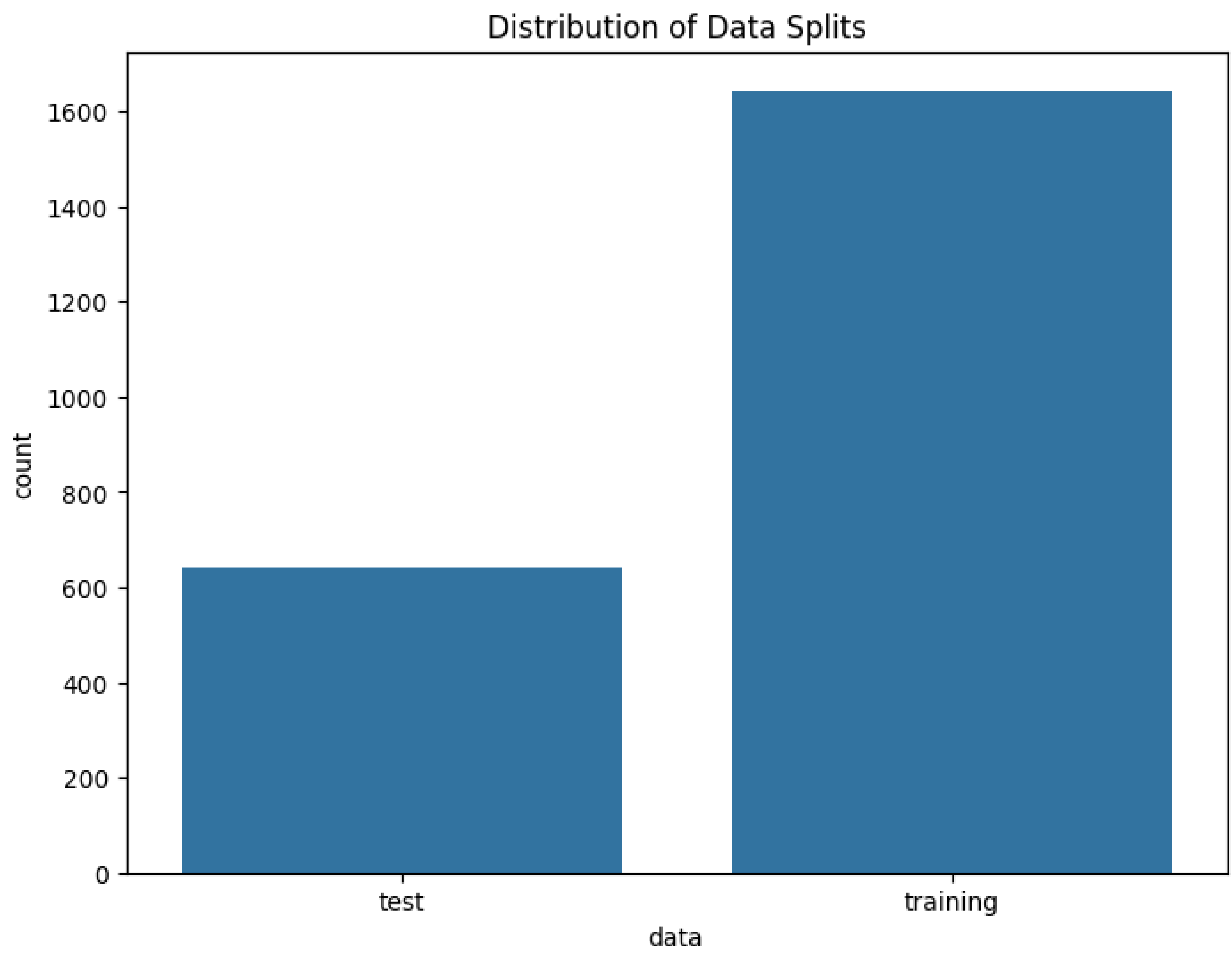
Frequency Distribution of Categorical Columns

```
trip_level_df['data'].value_counts(normalize=True).mul(100).round(2)
```

	proportion
data	
training	72.58
test	27.42

dtype: float64

```
plt.figure(figsize=(8,6))
sns.countplot(data=trip_level_df, x='data')
plt.title('Distribution of Data Splits')
plt.show()
```

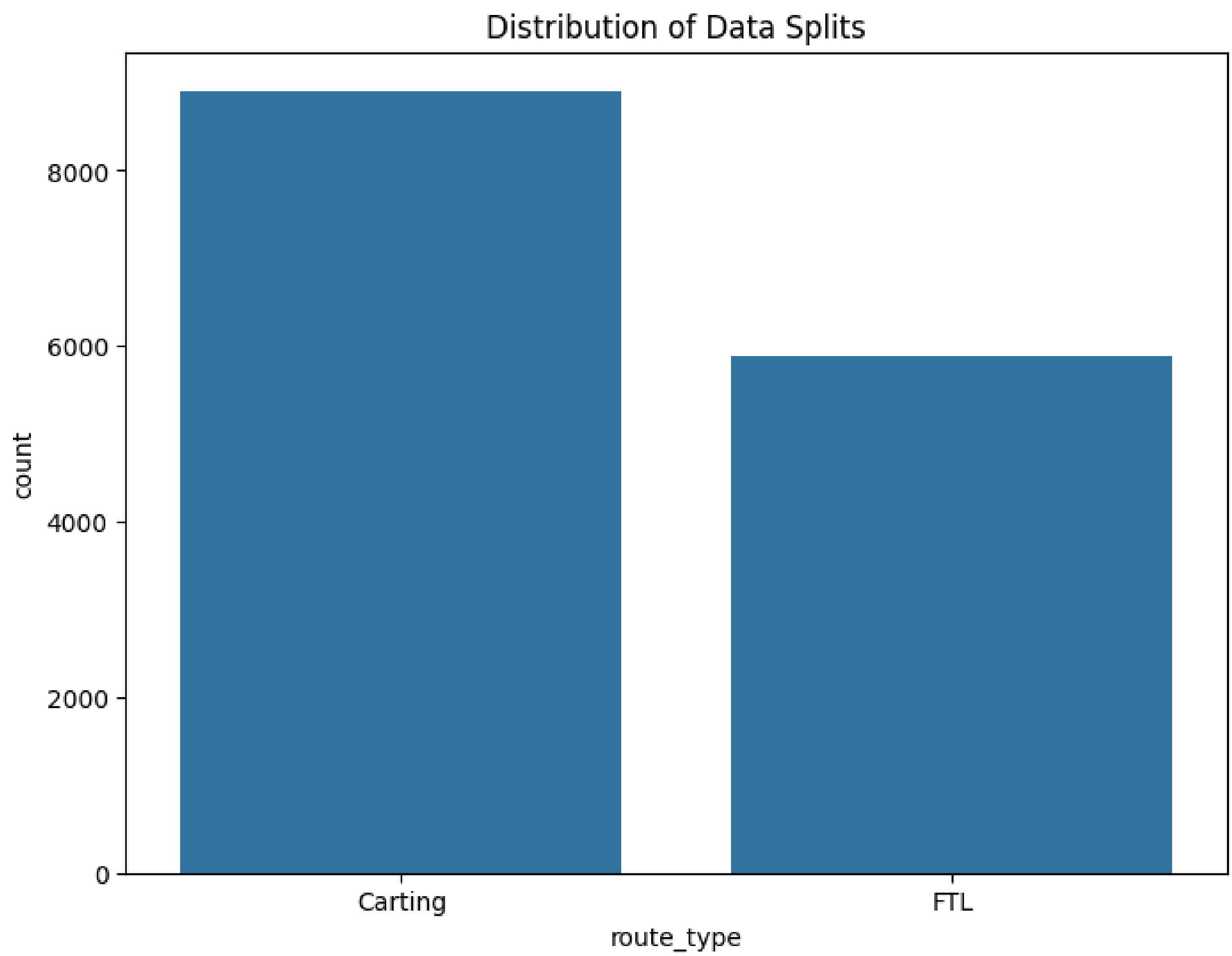



```
trip_level_df['route_type'].value_counts(normalize=True).mul(100).round(2)
```

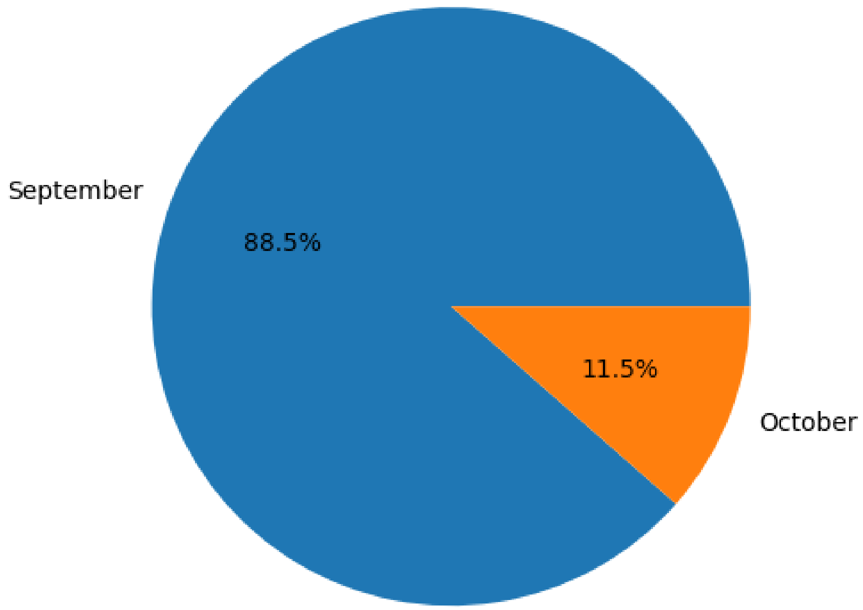
route_type	proportion
Carting	60.07
FTL	39.93

dtype: float64

```
plt.figure(figsize=(8,6))
sns.countplot(data=trip_level_df, x='route_type')
plt.title('Distribution of Data Splits')
plt.show()
```



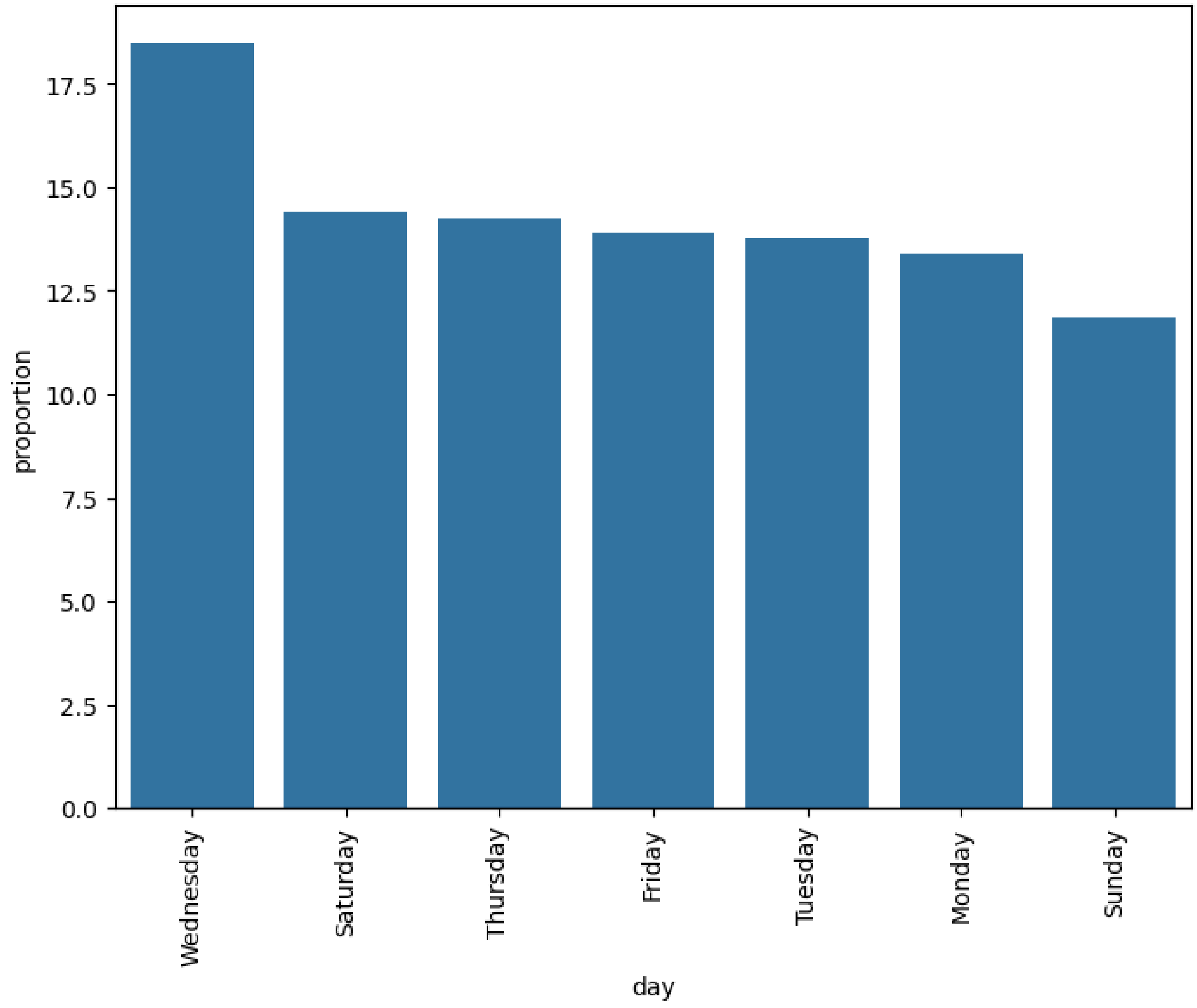

```
month=trip_level_df['month'].value_counts(normalize=True).mul(100).round(2)
plt.pie(month,labels=month.index,autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```



Insights

- About **72**% of the data belongs to the **training** category.
- Around **60%** of all trips are of **Carting** type, showing that Carting is the dominant mode of delivery in the network.
- **88%** of the trips occurred in the month of **September.**

```
plt.figure(figsize=(8,6))
days = trip_level_df['day'].value_counts(normalize=True).mul(100).round(2).reset_index()
sns.barplot(x='day',y='proportion',data=days)
plt.xticks(rotation = 90)
plt.show()
```



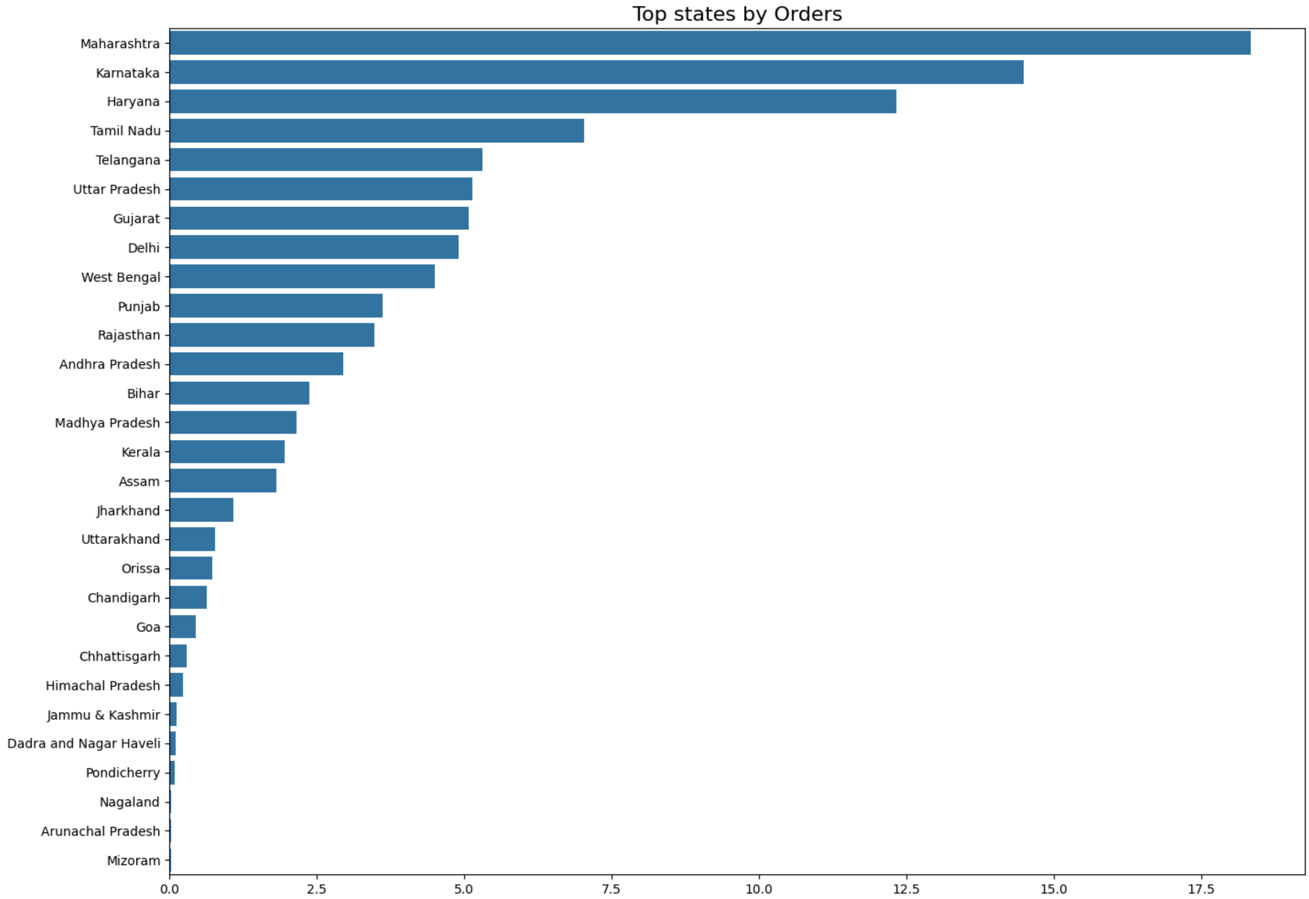
Insights

- Approximately 17% of trips are created on Wednesday, followed by Friday (15%) and Saturday (14%).
- Compared to weekdays, fewer trips are created during the weekend—especially on Sunday, which accounts for just 12% of the total.

Source State-wise Trip Count

```
order= trip_level_df['source_state'].value_counts(normalize=True).mul(100).round(2).reset_index()
```

```
plt.figure(figsize=(16,12))
ax = sns.barplot(y='source_state',x='proportion',data=order)
plt.title("Top states by Orders",fontsize=16)
plt.xlabel("")
plt.ylabel("")
plt.show()
```



Insights

- Most of the orders come from the state of **Maharashtra**, followed by Karnataka and Haryana.
- In contrast **Nagaland** contributes the least, accounting for only 0.05% of the total orders.
- This could be due to its low population density and hilly terrain, which make logistics more challenging and demand relatively low.

Top 5 Bussiest Corridor

```
city_route =trip_level_df['source_city']+ '-' + trip_level_df['destination_city']
```

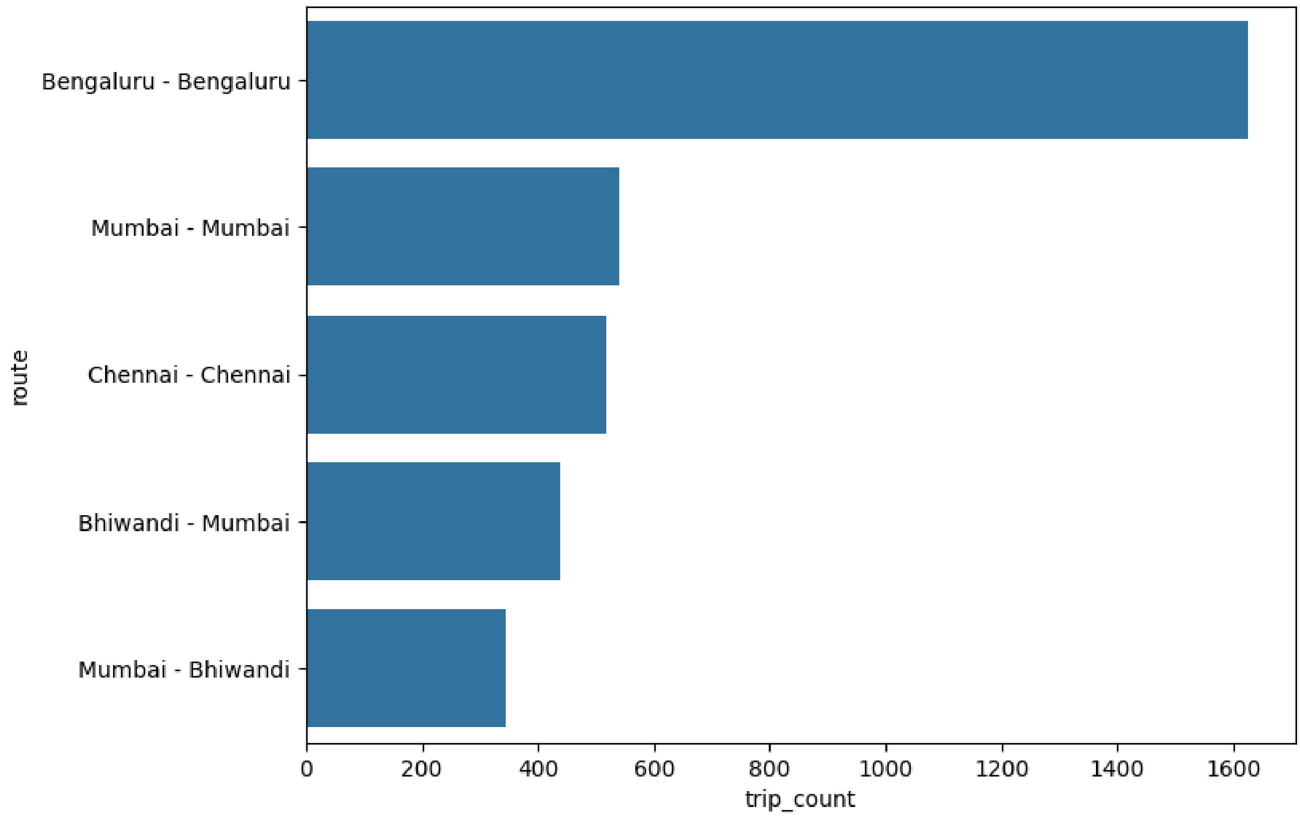
```
city_route
```

	source_city	destination_city	trip_count	route
0	Bengaluru	Bengaluru	1626	Bengaluru - Bengaluru
1	Mumbai	Mumbai	541	Mumbai - Mumbai
2	Chennai	Chennai	519	Chennai - Chennai
3	Bhiwandi	Mumbai	437	Bhiwandi - Mumbai
4	Mumbai	Bhiwandi	345	Mumbai - Bhiwandi
...	...	...	...	...
1344	Gobicheti	Anthiyour	1	Gobicheti - Anthiyour
1345	Vellore	Bengaluru	1	Vellore - Bengaluru
1346	Vikarabad	Hyderabad	1	Vikarabad - Hyderabad
1347	Aliganj	Mainpuri	1	Aliganj - Mainpuri
1348	Aligarh	Delhi	1	Aligarh - Delhi

1349 rows × 4 columns

Next steps: Generate code with city_route | View recommended plots | New interactive sheet

```
plt.figure(figsize=(8,6))
city_route=trip_level_df.groupby(['source_city','destination_city']).size().sort_values(ascending=Fa
city_route['route'] = city_route['source_city']+ ' - '+city_route['destination_city']
sns.barplot(y='route',x='trip_count',data=city_route.head(5))
plt.show()
```



Insights

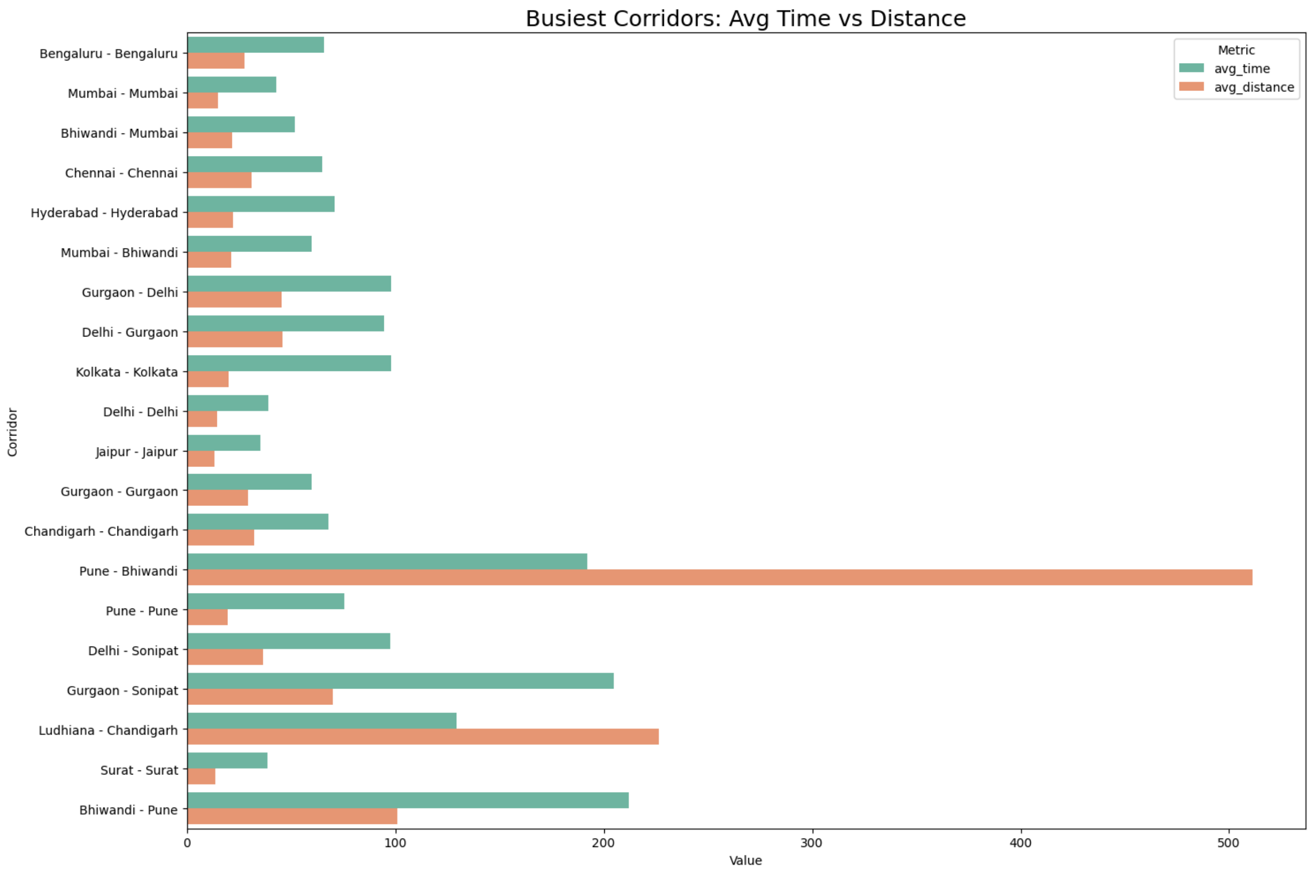
- **Bangaluru - Bangaluru** is the most active delivery corridor, with a trip count nearly triple that of any other city pair. This suggests a high concentration of intra-city deliveries within Bangaluru.
- There could be multiple warehouses or centers in the city.

```
merge = pd.merge(city_route,trip_level_df,on=['source_city','destination_city'],how='left')
final = merge.groupby(['route','trip_count']).agg(avg_time=('actual_time','median'),
                    avg_distance=('actual_distance_to_destination','median')).reset_index().sort_valu
```

```
final.head(10)
```

	route	trip_count	avg_time	avg_distance
58	Bengaluru - Bengaluru	254	66.0	27.902506
451	Mumbai - Mumbai	84	43.0	15.136192
93	Bhiwandi - Mumbai	75	52.0	21.579702
138	Chennai - Chennai	67	65.0	31.157974
304	Hyderabad - Hyderabad	55	71.0	22.066472
448	Mumbai - Bhiwandi	43	60.0	21.264406
244	Gurgaon - Delhi	38	98.0	45.654502
176	Delhi - Gurgaon	38	94.5	45.953875
384	Kolkata - Kolkata	35	98.0	20.088265
173	Delhi - Delhi	26	39.0	14.443220

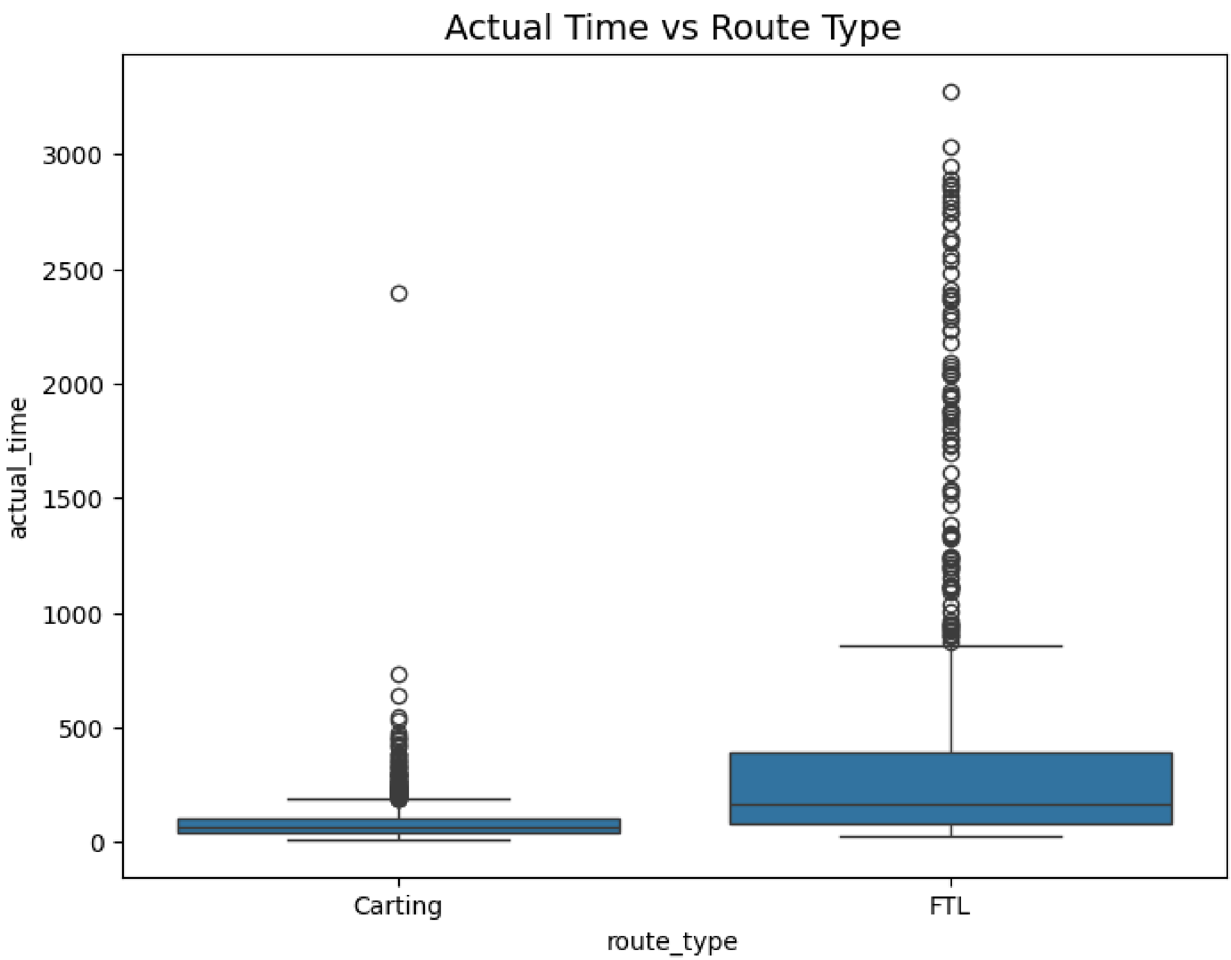
```
melted = final.melt(id_vars='route', value_vars=['avg_time', 'avg_distance'],
                            var_name='metric', value_name='value')

plt.figure(figsize=(15, 10))
sns.barplot(x='value', y='route', hue='metric', data=melted, palette='Set2')
plt.title('Busiest Corridors: Avg Time vs Distance',fontsize=18)
plt.xlabel('Value')
plt.ylabel('Corridor')
plt.legend(title='Metric')
plt.tight_layout()
plt.show()
```



Insights

- **Delhi - Gurgaon** performs well across metrics.
- Intra-city routes such as Delhi - Delhi, Mumbai - Mumbai, Jaipur- Iaipur show surprisingly high delivery times.
- **Pune - Bhiwandi** and **Gurgaon - Sonipat** despite being moderate average distances, take longer hours of delivery time, possibly due to traffic or poor road infrastructure.
- **Ludhiana - Chandigarh** has the longest average distance (~225), but its average time is not the highest.

```
plt.figure(figsize=(8,6))
sns.boxplot(x='route_type',y='actual_time',data=trip_level_df)
plt.title("Actual Time vs Route Type", fontsize=14)
plt.show()
```



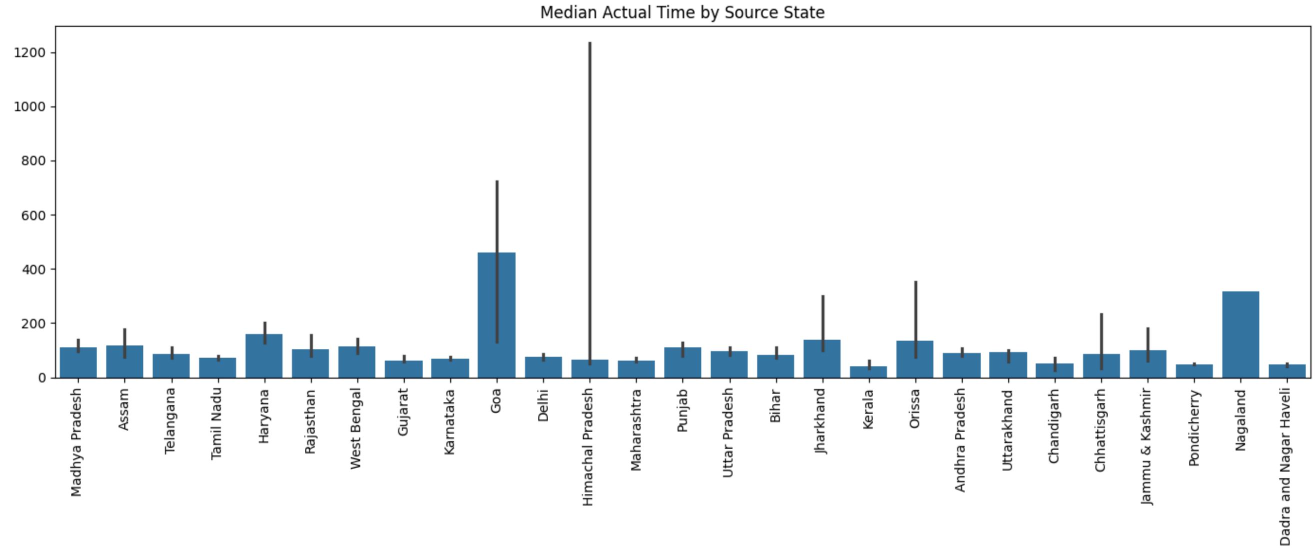
Insights

- From the box plot, we can infer that FTL (Full Truck Load) deliveries generally take more actual time compared to Carting deliveries with higher number of outliers.
- This may be due to longer distances, multiple checkpoints, or heavier load management involved in FTL shipments.
- Carting may be more local or regional, hence faster.

Delivery Time Trends Across Source States

```
plt.figure(figsize=(14,6))
sns.barplot(x='source_state', y='actual_time', data=trip_level_df, estimator='median')
plt.xticks(rotation=90)
plt.title("Median Actual Time by Source State")
plt.ylabel("Median Actual Time")
plt.xlabel("Source State")
plt.xlabel(""),plt.ylabel("")
```

```
plt.tight_layout()
plt.show()
```



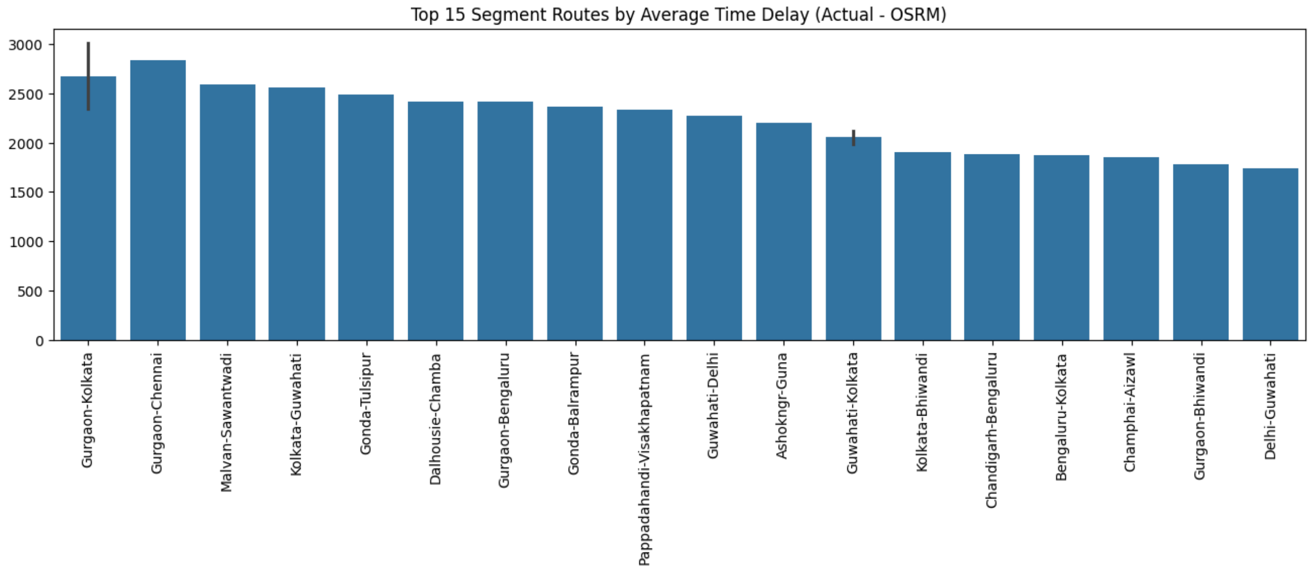
Insights

- Deliveries from Goa and Nagaland take much longer time than from other states.
- Although Himachal Pradesh has one of the lowest delivery volumes a few deliveriesfrom that state take significantly longer. This could be due to its hilly terrain, which might slow down transportation in certain areas.
- Kerala shows the shortest delivery time compared to other, indicating a highly efficient delivery process — possibly due to well-connected cities and optimized routes within the state.

Segment Time Deviation (Actual - OSRM)

```
segment_level_df['segment_route'] = segment_level_df['source_city'] + '-' + segment_level_df['destin

segment_level_df['segment_time_delay'] = (segment_level_df['segment_actual_time_sum'] - segment_leve
)

top_delay_routes = segment_level_df.sort_values('segment_time_delay',ascending=False).head(20)
plt.figure(figsize=(16,4))
sns.barplot(data=top_delay_routes, y='segment_time_delay', x='segment_route')
```

```
plt.xticks(rotation = 90)
plt.xlabel("")
plt.ylabel("")
plt.title('Top Segment Routes by Average Time Delay (Actual - OSRM)')
plt.show()
```



Insights

- The segment route from **Gonda to Balrampur** is the top delaying route at the segment level, showing the highest difference between actual and predicted (OSRM) delivery time. The system might be giving shorter distance estimates than what actually happens, which could lead to planning challenges or delays in deliveries.
- A pattern was observed where deliveries that either start from or end at Gurgaon usually take longer. This may be due to traffic congestion, urban delivery challenges, or route complexity in that area.

Challenging Segments

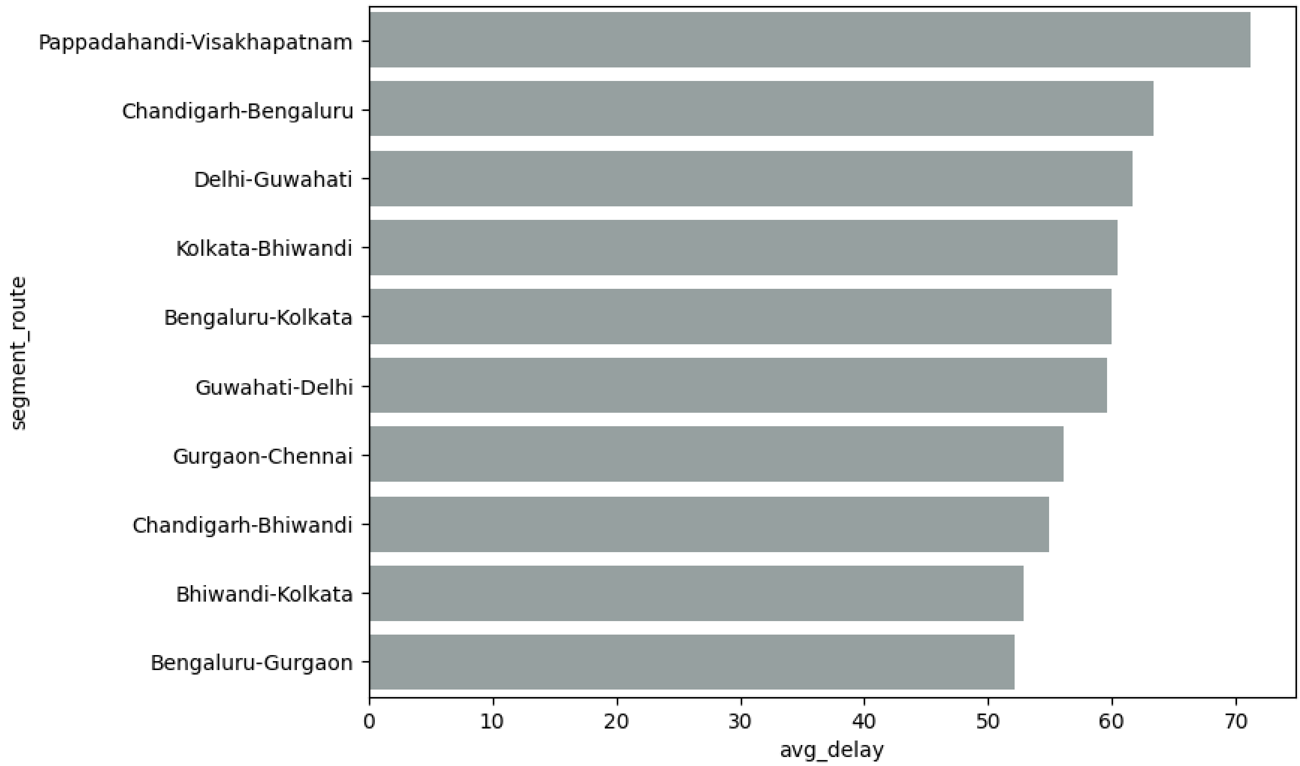
```
delay = (
    segment_level_df.groupby('segment_route')['od_time_diff_hour']
    .mean().reset_index()
    .rename(columns={'od_time_diff_hour':'avg_delay'})
    .sort_values('avg_delay',ascending=False)
)
delay
```

	segment_route	avg_delay
1656	Pappadahandi-Visakhapatnam	71.215047
425	Chandigarh-Bengaluru	63.368436
554	Delhi-Guwahati	61.716212
1222	Kolkata-Bhiwandi	60.507972
251	Bengaluru-Kolkata	59.925225
...	...	...
1252	Koraput-Jeypore	0.580511
1928	Sathyamangalam-Gobicheti	0.577847
117	Arsikere-Tiptur	0.525666
1483	Mundakayam-Parakkdavu	0.507069
1191	Khurdha-Khurdha	0.445126

2286 rows × 2 columns

Next steps: Generate code with delay | View recommended plots | New interactive sheet

```
plt.figure(figsize=(8,6))
top = delay.head(10)
sns.barplot(y='segment_route',x='avg_delay',data=top,color = '#95a5a6')
plt.show()
```



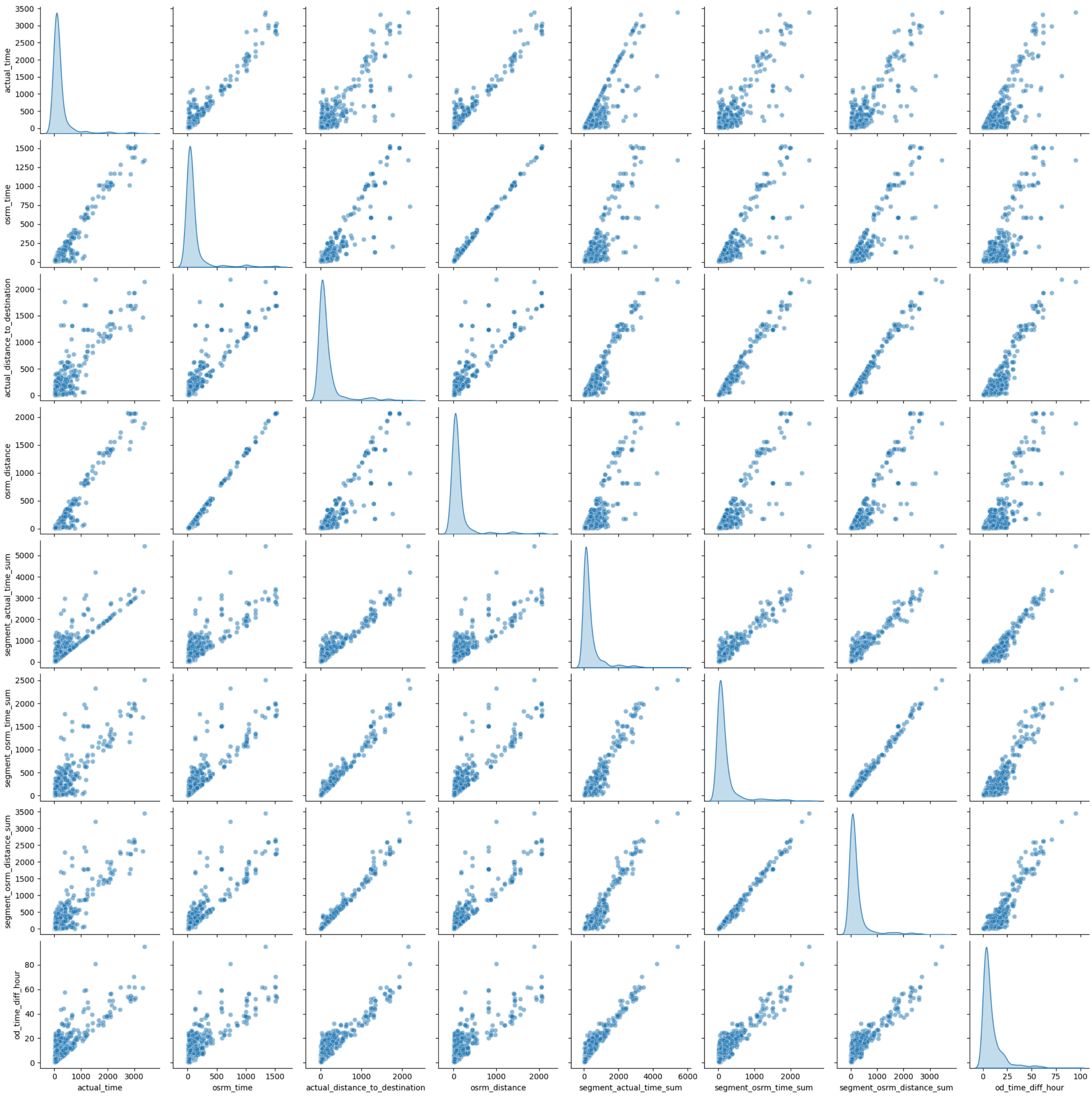
Insights

- The **Pappadahandi - Visakhapatnam** route has the highest average delay of ~71 hours, indicating significant operational inefficiencies or possible bottlenecks.
- Routes like **Chandigarh - Bengaluru**, **Delhi - Guwahati**, and** Kolkata - Bhiwandi** also show delays exceeding 59 hours, suggesting that long inter-state routes face more disruptions.
- Routes like Guwahati - Delhi and Delhi - Guwahati show delays in both directions, possibly due to challenging roads or route-specific issues.
- Bhiwandi, Gurgaon, and Bengaluru appear frequently in the delayed routes, either as starting or ending points. This may be due to congestion at hubs, traffic bottlenecks, or driver-related challenges like wait times or manpower issues.

Relationship between features

```
cols_for_pairplot = [
    'actual_time',
    'osrm_time',
    'actual_distance_to_destination',
    'osrm_distance',
    'segment_actual_time_sum',
    'segment_osrm_time_sum',
    'segment_osrm_distance_sum',
    'od_time_diff_hour'
]

sample_df = trip_level_df[cols_for_pairplot].sample(1000, random_state=42)
sns.pairplot(sample_df, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle("Pairplot of Key Numerical Variables", y=1.02, fontsize=16)
plt.show()
```

Pairplot of Key Numerical Variables



Insights

- There is a strong positive linear relationship between** osrm_time** and **osrm_distance**, forming a near-perfect straight line. This indicates that as the system-predicted distance increases, the predicted time also increases proportionally — as expected in route planning.
- The relationship between **actual_time and osrm_time** is mostly positive, but with noticeable deviations from the line. This suggests that while most deliveries align with predicted durations, there are some deliveries that took significantly longer than estimated — possibly due to traffic, weather, or operational delays.
- The correlation between **actual_time and actual_distance_to_destination** is positive, but more scattered and noisy. This indicates that delivery duration generally increases with distance, but some deliveries took unusually more or less time for the same distance — highlighting inconsistencies or inefficiencies.
- The relationship between **actual_time and osrm_distance** is also positive, though with some outliers falling below the trend line. This suggests that certain deliveries took more time than expected for the predicted distance, hinting at operational challenges or route inefficiencies.

one-hot encoding of categorical variables

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

route_type	count
Carting	1388
FTL	897

dtype: int64

```
encoded_df = pd.get_dummies(trip_level_df[['data', 'route_type']]).astype(int)
mod_df = trip_level_df.copy()
mod_df.drop(['data', 'route_type'], axis=1, inplace=True)
mod_df = pd.concat([encoded_df, mod_df], axis=1)
mod_df.head()
```

trip_uuid	data_test	data_training	route_type_Carting	route_type_FTL	
trip-153671079956500691	0	1	1	0	1536710799565
trip-153671110078355292	0	1	1	0	1536711100783
trip-153671191949943656	0	1	0	1	1536711919499
trip-153671237597058150	0	1	1	0	1536712375970
trip-153671262893947351	0	1	1	0	1536712628939

5 rows × 23 columns

Normalize/ Standardize the numerical features

```
hist_col=['start_scan_to_end_scan','actual_distance_to_destination','actual_time',
         'osrm_time','osrm_distance','segment_actual_time_sum','segment_osrm_distance_sum',
         'segment_osrm_time_sum','od_time_diff_hour']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(trip_level_df[hist_col])
scaled_df = pd.DataFrame(scaled_data, columns=hist_col)
```

```
scaled_df.head()
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance
0	0.003495	0.000398	0.003675	0.000623	0.000376
1	0.000932	0.000179	0.001838	0.001247	0.000770
2	0.059646	0.041824	0.020521	0.021820	0.018000
3	0.050792	0.014018	0.070444	0.016209	0.017311
4	0.054753	0.007059	0.005819	0.003117	0.003148

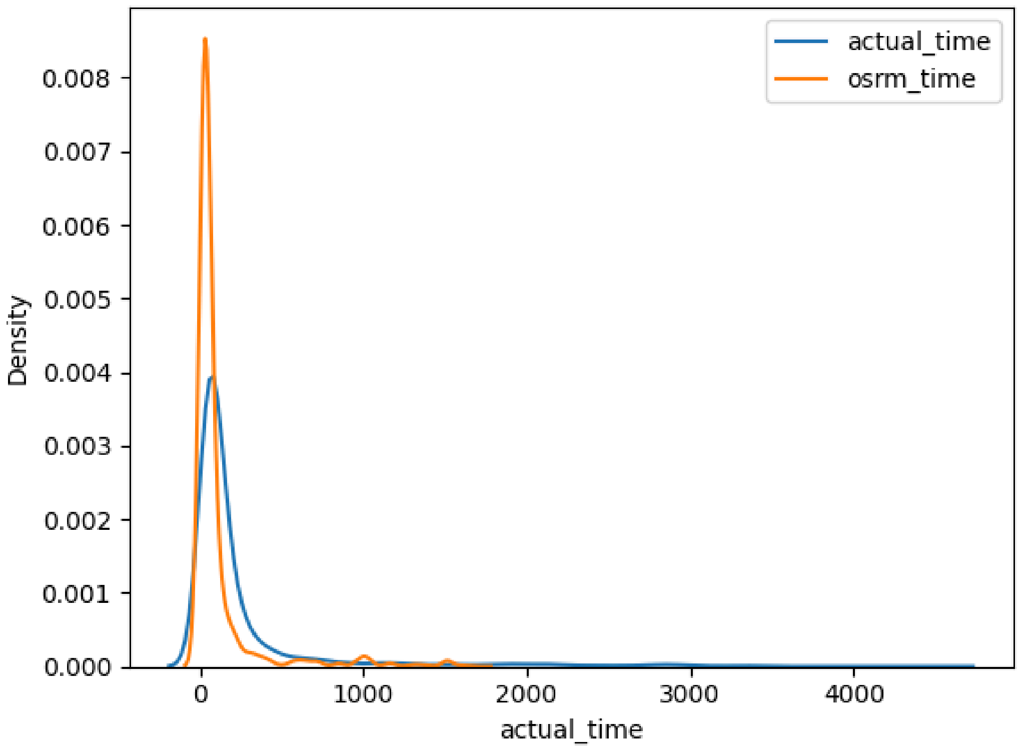
```
scaled_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
start_scan_to_end_scan	14787.0	0.064308	0.083588	0.0	0.016000	0.032508	0.077333	1.0
actual_distance_to_destination	14787.0	0.071222	0.140298	0.0	0.006326	0.018041	0.070993	1.0
actual_time	14787.0	0.048424	0.099426	0.0	0.009286	0.017024	0.037143	1.0
osrm_time	14787.0	0.058686	0.130888	0.0	0.010119	0.018452	0.039881	1.0
osrm_distance	14787.0	0.053753	0.130389	0.0	0.007249	0.013747	0.033195	1.0
segment_actual_time_sum	14787.0	0.055306	0.089434	0.0	0.009163	0.022183	0.057065	1.0
segment_osrm_distance_sum	14787.0	0.060785	0.118606	0.0	0.006688	0.017274	0.059037	1.0
segment_osrm_time_sum	14787.0	0.068222	0.123018	0.0	0.009382	0.023065	0.069586	1.0
od_time_diff_hour	14787.0	0.064361	0.083607	0.0	0.016030	0.032539	0.077469	1.0

Hypothesis Testing

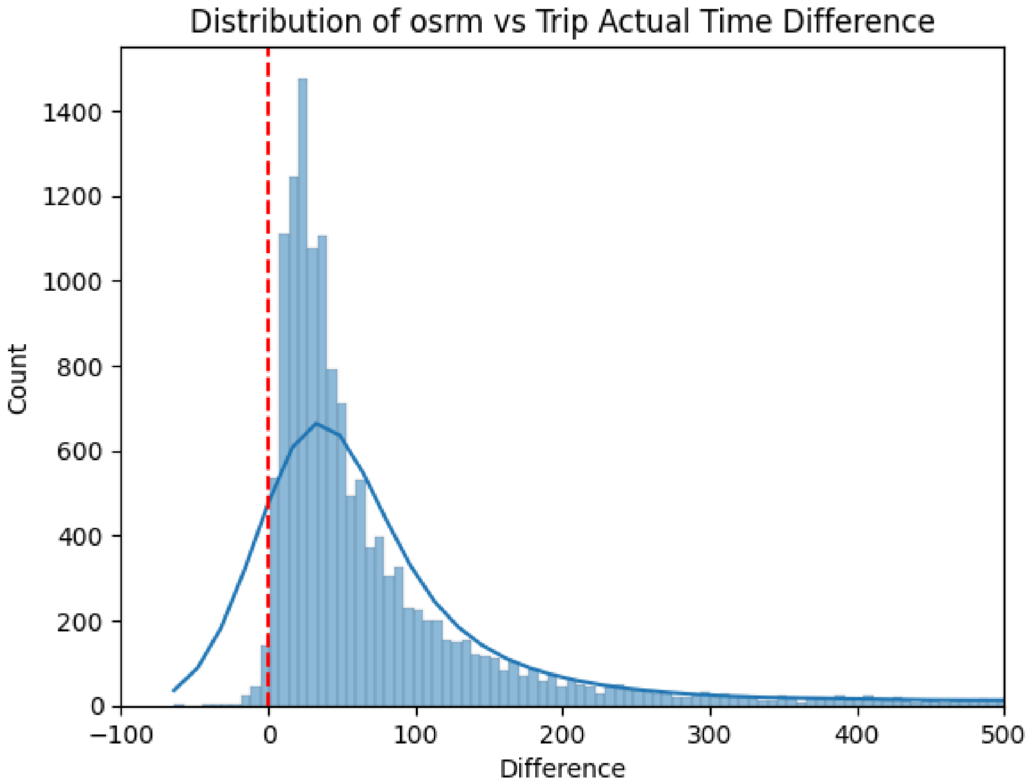
a. actual_time aggregated value and OSRM time aggregated value.

```
sns.kdeplot(trip_level_df['actual_time'],label='actual_time')
sns.kdeplot(trip_level_df['osrm_time'],label='osrm_time')
plt.legend()
plt.show()
```



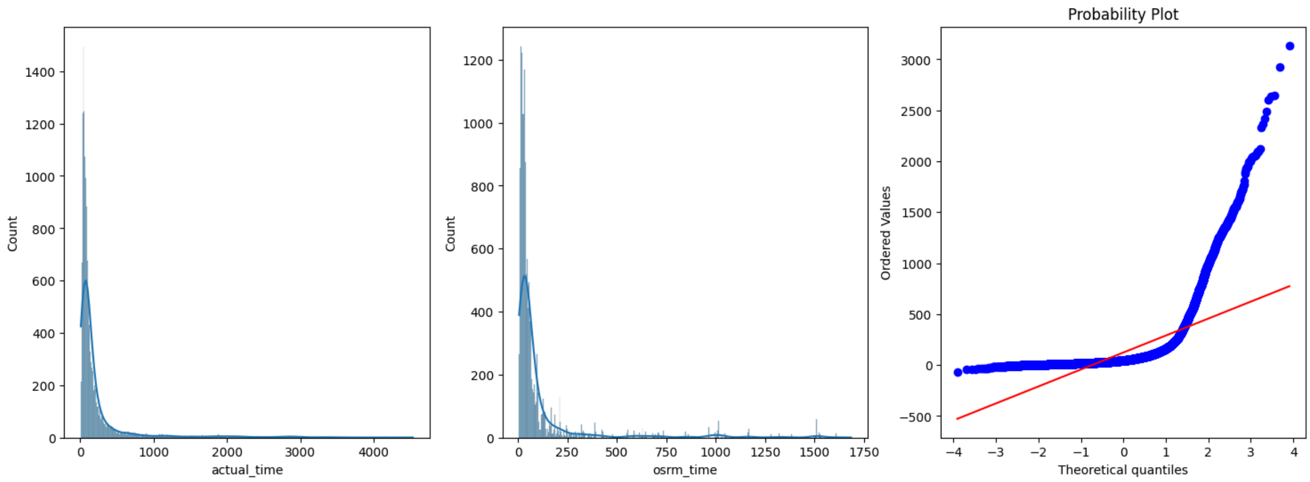
From the KDE plot, we observe that the distribution of actual delivery time is more spread out than the OSRM-predicted time, indicating higher variability and inconsistencies in deliveries

```
diff=trip_level_df['actual_time']-trip_level_df['osrm_time']
sns.histplot(x=diff,kde=True)
plt.axvline(x=0, color='red', linestyle='--')
plt.xlim(-100, 500)
plt.title("Distribution of osrm vs Trip Actual Time Difference")
plt.xlabel("Difference")
plt.show()
```



Most of the values in the distribution are greater than zero, meaning the actual delivery time is higher than the predicted OSRM time.

```
from scipy import stats
plt.figure(figsize=(18, 6))
plt.subplot(1,3,1)
sns.histplot(x='actual_time',data=trip_level_df,kde=True)
plt.subplot(1,3,2)
sns.histplot(x='osrm_time',data=trip_level_df,kde=True)
plt.subplot(1,3,3)
stats.probplot(x=diff,dist='norm',plot=plt)
plt.show()
```



Obeservation

- Since both actual_time and osrm_time are not normally distributed, as observed from their right-skewed histograms.
- It's confirmed using QQ plot.
- Since the two columns are paired samples and the distributions are not normal, we apply the non-parametric Wilcoxon signed-rank test to compare the paired observations.

hypothesis testing

Null hypothesis (H_0):

There is no significant difference between actual time and osrm_time. (mean of differences = 0)

Alternative hypothesis (Ha): **bold text** There is a significant difference between actual time and osrm_time. (mean of differences ≠ 0)

```
from scipy.stats import wilcoxon
actual = trip_level_df['actual_time']
osrm = trip_level_df['osrm_time']

stat, p_value = wilcoxon(actual, osrm)

print(f"Wilcoxon test statistic: {stat}")
print(f"P-value: {p_value}")
```

```
Wilcoxon test statistic: 179161.0
P-value: 0.0
```

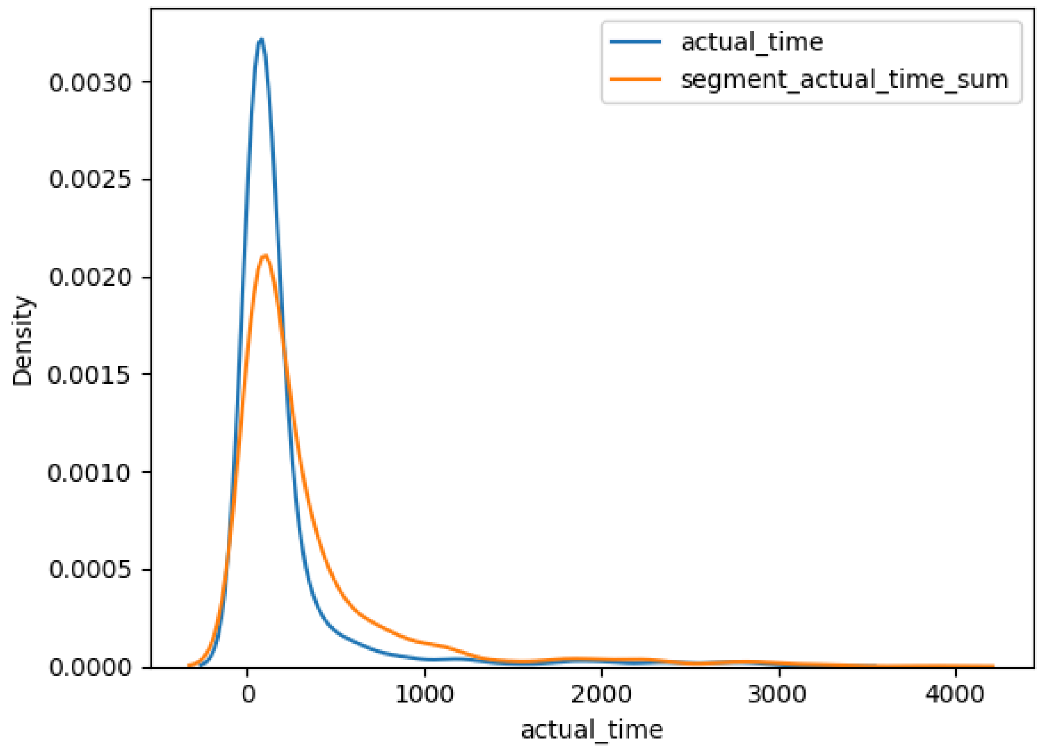
Interpretation: Since the p-value is 0.0, which is much less than 0.05, we reject the null hypothesis.

Conclusion

The test gave us a result that is almost zero. we can say that there is a significant difference between the actual delivery time and the estimated OSRM time.

b. actual_time aggregated value and segment actual time aggregated value

```
sns.kdeplot(trip_level_df['actual_time'],label='actual_time')
sns.kdeplot(trip_level_df['segment_actual_time_sum'],label='segment_actual_time_sum')
plt.legend()
plt.show()
```

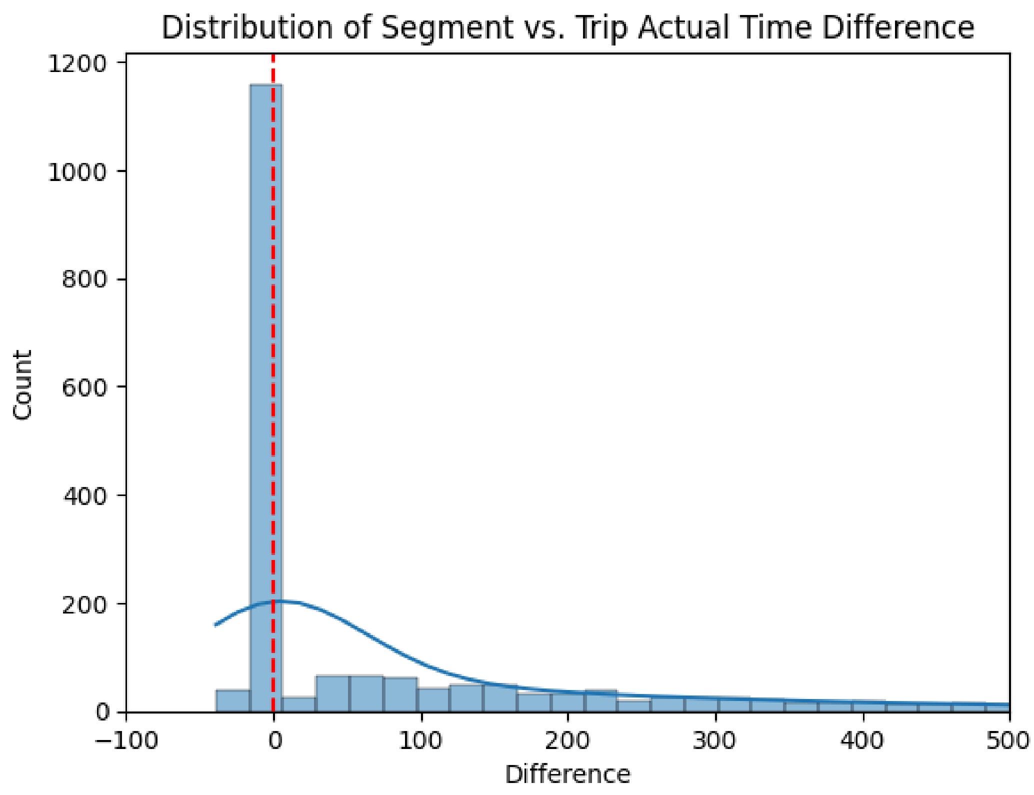


Observation

The KDE plot shows that the segment actual time is more spread out than the overall actual time, indicating higher variability in segment-level deliveries. This suggests that individual segments may experience more delays or inconsistencies compared to the overall trip duration.

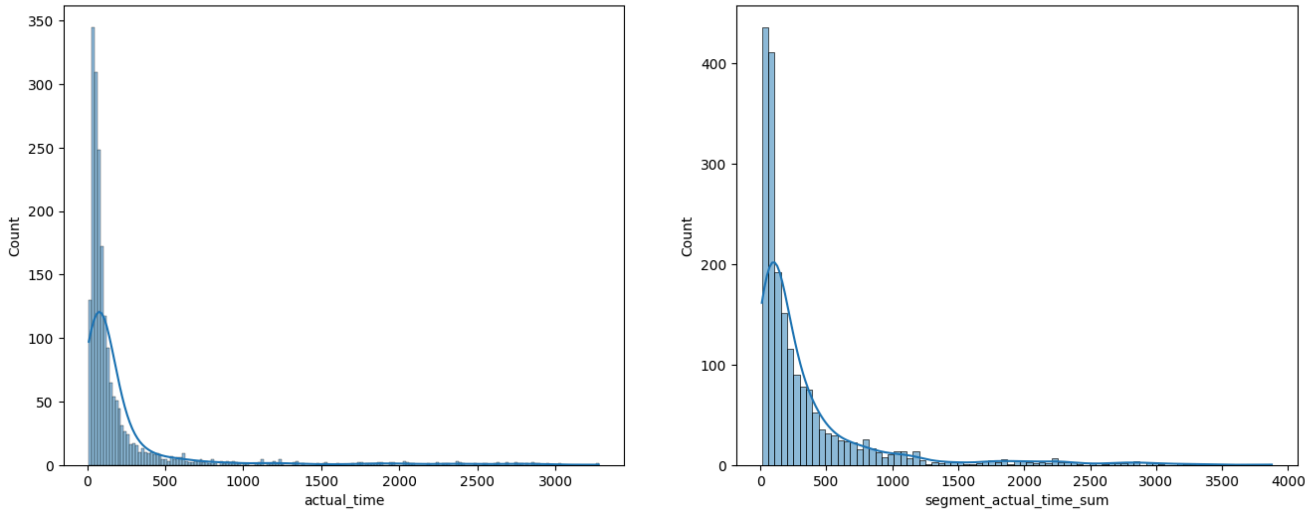
```
diff = trip_level_df['segment_actual_time_sum']-trip_level_df['actual_time']
sns.histplot(x=diff, kde=True)
plt.axvline(x=0, color='red', linestyle='--')
plt.xlim(-100, 500)
plt.title("Distribution of Segment vs. Trip Actual Time Difference")
```

```
plt.xlabel("Difference")
plt.show()
```



- Most of the bars lie to the right of the red line, meaning that in many cases, the sum of segment-level delivery times exceeds the total trip-level delivery time.
- This suggests that segment-level deliveries are taking longer than the full trip time, which is logically inconsistent.

```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.histplot(x='actual_time',data=trip_level_df,kde=True)
plt.subplot(1,2,2)
sns.histplot(x='segment_actual_time_sum',data=trip_level_df,kde=True)
plt.show()
```



Observation

Since both actual time and segment actual time are right-skewed and relatively dependent, we will use the Wilcoxon signed-rank test.

Hypothesis testing

Null hypothesis (H_0):

There is no significant difference between actual time and segment_actual_time. (mean of differences = 0)

Alternative hypothesis (Ha):

There is a significant difference between actual time and segment_actual_time. (mean of differences ≠ 0)

```
#Wilcoxon signed-rank test
stat, p = wilcoxon(trip_level_df['segment_actual_time_sum'], trip_level_df['actual_time'])

print(f"Wilcoxon test statistic: {stat}")
print(f"P-value: {p}")
```

```
Wilcoxon test statistic: 18807926.0
P-value: 0.0
```

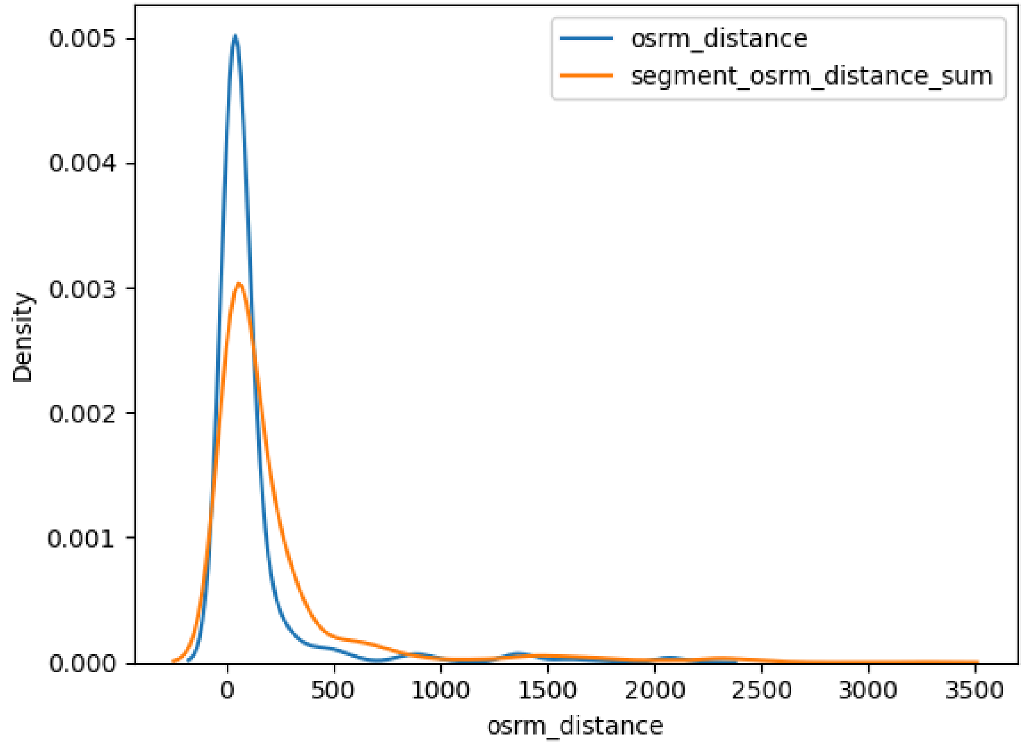
Interpretation: Since the p-value is 0.0, which is much less than 0.05, we reject the null hypothesis.

Conclusion

The test gave us a result that is almost zero. we can say that there is a significant difference between the overall actual delivery time and the segment level delivery time.

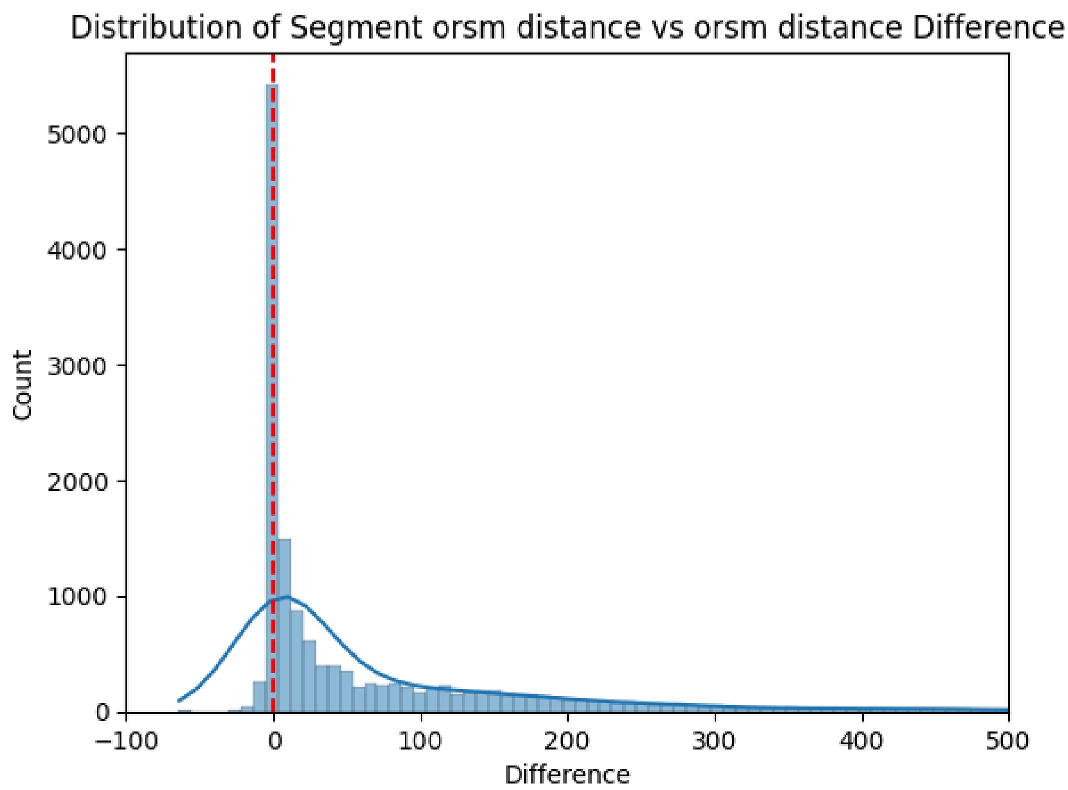
c. OSRM distance aggregated value and segment OSRM distance aggregated value.

```
sns.kdeplot(trip_level_df['osrm_distance'],label='osrm_distance')
sns.kdeplot(trip_level_df['segment_osrm_distance_sum'],label='segment_osrm_distance_sum')
plt.legend()
plt.show()
```



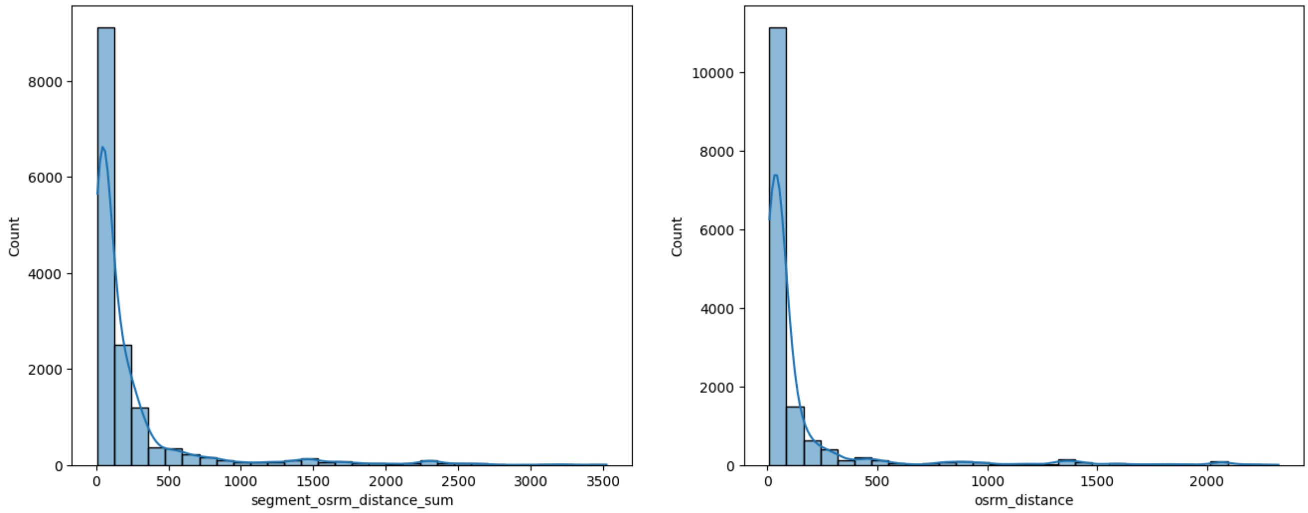
It is clearly visible that the segment-level OSRM distance is more spread out than the trip-level OSRM distance, indicating greater variability in the predicted distances across individual segments.

```
diff = trip_level_df['segment_osrm_distance_sum'] - trip_level_df['osrm_distance']
sns.histplot(x=diff,kde=True)
plt.axvline(x=0, color='red', linestyle='--')
plt.xlim(-100, 500)
plt.title("Distribution of Segment orsm distance vs orsm distance Difference")
plt.xlabel("Difference")
plt.show()
```



- Most of the bars (data) lie to the right of the red line, meaning in most cases, the sum of segment-level OSRM distances is greater than the trip-level OSRM distance. we confirn this using hypothesis test.
- To statistically validate this observation, we can perform a hypothesis test to test whether the mean of the difference is significantly greater than zero.

```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.histplot(x='segment_osrm_distance_sum',data=trip_level_df,kde=True,bins=30)
plt.subplot(1,2,2)
sns.histplot(x='osrm_distance',data=trip_level_df,kde=True,bins=30)
plt.show()
```



Observation

Since both segment_osrm_distance time and osrm distance are right-skewed and relatively dependent, we will use the Wilcoxon signed-rank test.

Hypothesis testing

Null hypothesis (H_0):

There is no significant difference between osrm_distance and segment_osrm_distance. (mean of differences = 0)

Alternative hypothesis (Ha):

There is a significant difference between osrm_distance and ssegment_osrm_distance. (mean of differences ≠ 0)

Wilcoxon signed_rank test

```
stat, p = wilcoxon(trip_level_df['segment_actual_time_sum'], trip_level_df['actual_time'])

print(f"Wilcoxon test statistic: {stat}")
print(f"P-value: {p}")
```

```
Wilcoxon test statistic: 18807926.0
P-value: 0.0
```

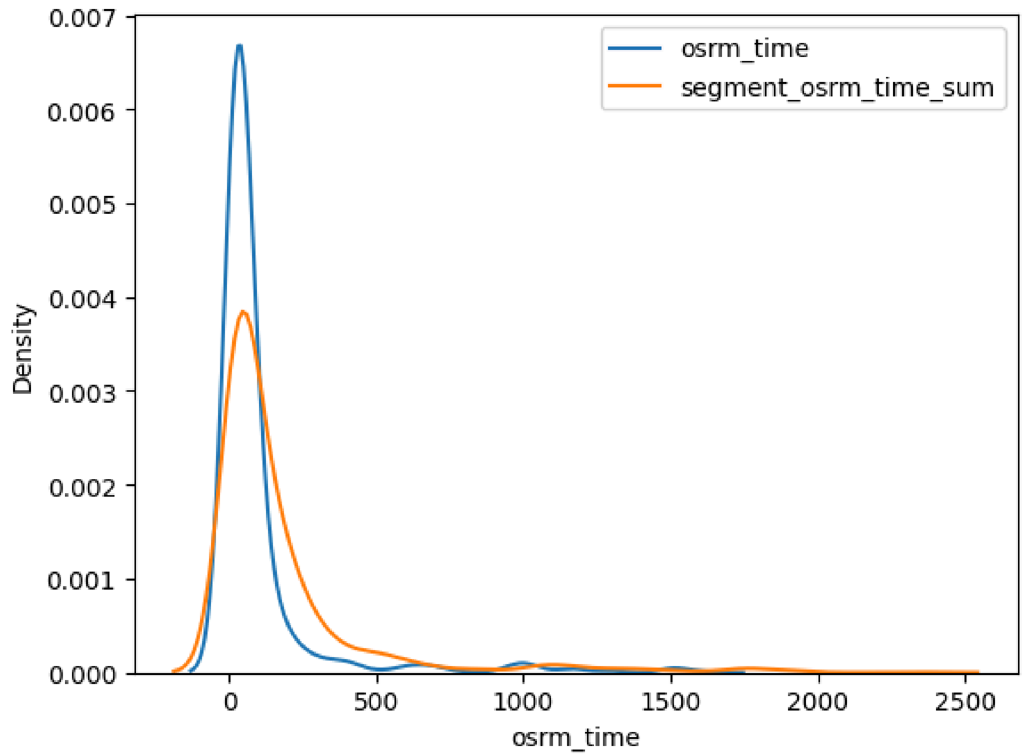
Interpretation: Since the p-value is 0.0, which is much less than 0.05, we reject the null hypothesis.

Conclusion

The test gave us a result that is almost zero. we can say that there is a significant difference between the system predicted distance at segment level is higher than OSRM distance predicted for the entire trip.

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

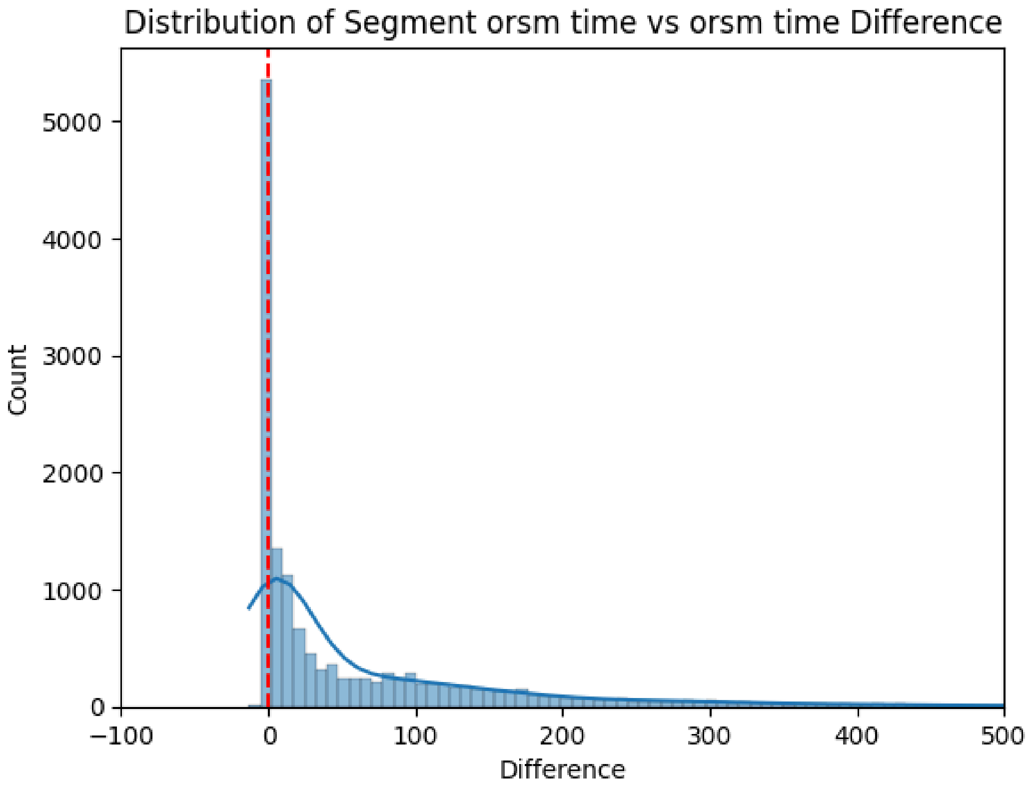
```
sns.kdeplot(trip_level_df['osrm_time'],label ='osrm_time')
sns.kdeplot(trip_level_df['segment_osrm_time_sum'],label ='segment_osrm_time_sum')
plt.legend()
plt.show()
```



Interpretation

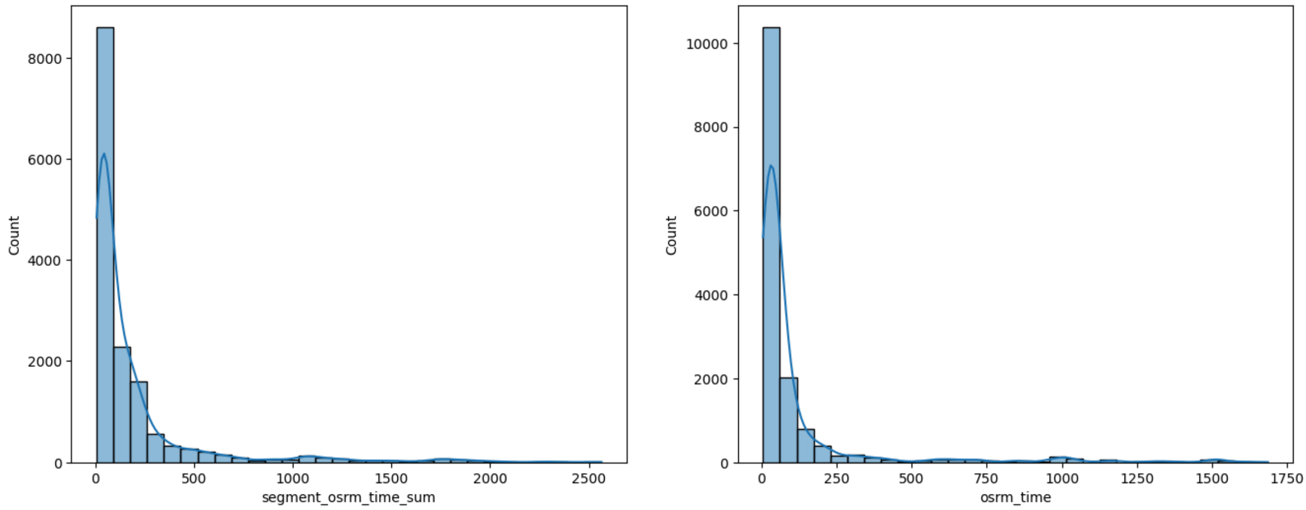
The KDE plot shows that the segment OSRM time is more spread out than the OSRM time, indicating that the predicted time has higher variability at the segment level.

```
diff = trip_level_df['segment_osrm_time_sum']-trip_level_df['osrm_time']
sns.histplot(x=diff,kde=True)
plt.axvline(x=0, color='red', linestyle='--')
plt.xlim(-100, 500)
plt.title("Distribution of Segment orsm time vs orsm time Difference")
plt.xlabel("Difference")
plt.show()
```



since most bars are to the right of the regresion line, it suggests segment OSRM times are often overestimated compared to the overall trip prediction.

```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.histplot(x='segment_osrm_time_sum',data=trip_level_df,kde=True,bins=30)
plt.subplot(1,2,2)
sns.histplot(x='osrm_time',data=trip_level_df,kde=True,bins=30)
plt.show()
```



Since both segment_osrm_distance time and osrm distance are right-skewed and relatively dependent, we will use the Wilcoxon signed-rank test.

Hypothesis testing

Null hypothesis (H_0):

There is no significant difference between osrm_time and segment_osrm_time. (mean of differences = 0)

Alternative hypothesis (Ha):

There is a significant difference between osrm_time and segment_osrm_time. (mean of differences ≠ 0)

```
#Wilcoxon signed-rank test
stat, p = wilcoxon(trip_level_df['segment_actual_time_sum'], trip_level_df['actual_time'])

print(f"Wilcoxon test statistic: {stat}")
print(f"P-value: {p}")
```

```
Wilcoxon test statistic: 18807926.0
P-value: 0.0
```

Interpretation: Since the p-value is 0.0, which is much less than 0.05, we reject the null hypothesis.

Conclusion

The test gave us a result that is almost zero. we can say that there is a significant difference between the time predicted at segment level is higher than the predicted time for the entire trip.

Recommendations

- A significant volume of orders originates from top-performing states like** Maharashtra**. Additionally, most of the busiest delivery corridors connect major metro cities, suggesting that focusing operational efforts on these urban hubs can lead to higher efficiency and better resource allocation.
- For low-volume states like **Nagaland (0.05%)**, consider minimal or on-demand support, given the low order volume and possible geographic or logistical challenges.
- Despite being short to moderate in distance, **Bhiwandi - pune** and **Gurgaon - Sonipat** corridors show unusually high delivery times. It's recommended to conduct a route audit, optimize hub operations, and adopt traffic-aware dynamic routing to improve efficiency.
- Most of the trips are created on Wednesdays (approximately 17%), it is recommended to ensure adequate staffing and resource allocation on this day to handle peak operational load efficiently.
- Since **FTL routes** are more challenging than Carting routes, it is recommended to analyze the FTL network for possible delays due to routing, loading/unloading, or long-distance planning. Route optimization and better scheduling can help reduce delivery time for FTL shipments.
- Since most delays are observed on long-distance routes (like Chandigarh - Bengaluru, Guwahati - Delhi, Kolkata - Bhiwandi), which likely pass through multiple hubs. it is recommended to reduce wait times and handover delays at high-traffic hubs like Bhiwandi, Bengaluru, and Gurgaon. Also Explore alternative routes or optimize dispatch based on real-time traffic conditions.
- The test results show a clear gap between predicted and actual delivery times, indicating that the current system does not fully capture real delivery conditions. It is recommended to enhance the time and distance predictions by incorporating real-world delivery data — such as traffic delays, wait times at hubs, and en-route stops. Also, improve how individual segment times are combined into a full trip. Avoid double-counting delays or overestimating time at each step, as this can lead to inflated total trip times.

These improvements will support better route planning, more accurate ETAs, and a smoother delivery experience overall.

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