DELHIVERY - Business Case Study

DELHIVERY

Author - Shishir Bhat

Business Problem:

Introduction

About Delhivery

- Delhivery is one of India's fastest-growing and leading integrated logistics providers.
- The company is on a mission to build the commerce operating system for India.
- It combines high-quality logistics operations, robust infrastructure, and advanced technology to achieve this vision.

Why This Case Study Matters

From Delhivery’s Perspective:

- Aligns with Delhivery’s strategic objective of becoming the top player in the logistics space.
- Offers a real-world framework to understand and process logistics-related data.
- Supports building and refining data engineering pipelines for scalable data handling.
- Helps ensure **data integrity** by addressing missing values and normalizing the dataset.
- Enables the extraction of critical features necessary for **forecasting and predictive models**.
- Aids in uncovering **patterns and trends** that can drive operational improvements.
- Supports the generation of **actionable business insights** through detailed analysis.
- Facilitates **hypothesis testing** and **outlier detection** to improve process accuracy.
- Ultimately contributes to **enhanced decision-making** and **operational efficiency** in Delhivery’s logistics ecosystem.

In [ ]:
```
# Importing  necessary Libraries

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import re
from sklearn.preprocessing import StandardScaler , MinMaxScaler , OneHotEncoder
```

In [ ]:
```
# Importing the Dataset

!gdown 1qsbUABF_eQdOlzVkiR6cnkLvDP9l4yq2
```

```
Downloading...
From: https://drive.google.com/uc?id=1qsbUABF_eQdOlzVkiR6cnkLvDP9l4yq2
To: /content/delhivery_data.csv
100% 55.6M/55.6M [00:01<00:00, 52.1MB/s]
```

In [ ]:
```
df = pd.read_csv('delhivery_data.csv')
```

Analysing basic metrics :

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

Out[ ]:
```
(144316, 28)
```

The dataset has 144,867 rows and 24 columns.

In [ ]:
```
df.head(15)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	de
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	
10	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	
11	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	
12	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	
13	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	
14	training	2018-09-23 06:42:06.021680	thanos::sroute:ff52ef7a-4d0d-4063-9bfe-cc21172...	FTL	trip-153768492602129387	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	

15 rows × 24 columns

In [ ]:

```
df.columns
```

Out[ ]:

```
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')
```

Column Profiling:

1. data - tells whether the data is testing or training data
2. trip_creation_time – Timestamp of trip creation
3. route_schedule_uuid – Unique ID for a particular route schedule
4. route_type – Transportation type a. 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 b. Carting: Handling system consisting of small vehicles (carts)
5. trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
6. source_center - Source ID of trip origin
7. source_name - Source Name of trip origin
8. destination_cente – Destination ID
9. destination_name – Destination Name

10. od_start_time – Trip start time
11. od_end_time – Trip end time
12. start_scan_to_end_scan – Time taken to deliver from source to destination
13. is_cutoff – Unknown field
14. cutoff_factor – Unknown field
15. cutoff_timestamp – Unknown field
16. actual_distance_to_destination – Distance in kms between source and destination warehouse
17. actual_time – Actual time taken to complete the delivery (Cumulative)
18. 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)
19. 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)
20. factor – Unknown field
21. segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
22. segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
23. segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
24. segment_factor – Unknown field

Dataset Structure & Key Insights

- **Trip Segmentation:**
  - Each `trip_uuid` represents a complete delivery from **source center** to **destination center**.
  - However, each trip is composed of **multiple rows**, each representing a **segment** or leg of the journey.
  - These segments can include transfers across city hubs, regional centers, and national warehouses.
- **Hierarchical Routing Structure:**
  - Real-world logistics involves **multi-hop paths** — not just point-to-point.
  - This dataset mimics that, with trips broken down into smaller legs between intermediate nodes.
- **Segment Aggregation:**
  - For each trip, segment-level metrics like:
    - `segment_actual_time_sum`
    - `segment_osrm_time_sum`
    - `segment_osrm_distance_sum`
  - ...represent the total for all legs combined under the same `trip_uuid`.
- **Why This Matters:**
  - Enables **micro vs macro** analysis:
    - Compare **total segment time** vs **direct route estimates**.
    - Identify where OSRM (predicted times/distances) differs from actuals.
  - Useful for detecting:
    - Delays at specific segments
    - Routing inefficiencies
    - Over- or under-estimations in delivery time
- **Business Value:**
  - Allows **fine-grained control and visibility** of operations.
  - Supports **SLA improvements**, **cost reduction**, and **routing optimization**.
  - Segment-wise tracking can help build better **ETA prediction models**.

In [ ]:
```
# Checking basic information and Null values

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 [ ]:
```
#Checking Missing value percentage

missing_percentage = (df.isnull().sum() / len(df)) * 100
print("Percentage of Missing Values:")
print(missing_percentage.sort_values(ascending=False))
```

```
Percentage of Missing Values:
source_name                       0.202254
destination_name                  0.180165
route_schedule_uuid               0.000000
data                              0.000000
route_type                        0.000000
trip_uuid                         0.000000
source_center                     0.000000
trip_creation_time                0.000000
destination_center                0.000000
od_start_time                     0.000000
od_end_time                       0.000000
start_scan_to_end_scan            0.000000
is_cutoff                         0.000000
cutoff_factor                     0.000000
cutoff_timestamp                  0.000000
actual_distance_to_destination    0.000000
actual_time                       0.000000
osrm_time                         0.000000
osrm_distance                     0.000000
factor                            0.000000
segment_actual_time               0.000000
segment_osrm_time                 0.000000
segment_osrm_distance             0.000000
segment_factor                    0.000000
dtype: float64
```

Very few missing values and makes sense to drop them

In [ ]:
```
df = df.dropna()
df = df.reset_index(drop=True)
```

In [ ]:
```
#Checking Duplicates

df.duplicated().sum()
```

Out[ ]:
```
np.int64(0)
```

No Duplicates

In [ ]:
```
# Converting time columns to pandas datetime format

time_cols = ['trip_creation_time', 'od_start_time', 'od_end_time','cutoff_timestamp']
for col in time_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144316 entries, 0 to 144315
Data columns (total 24 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  is_cutoff                       144316 non-null  bool
 13  cutoff_factor                   144316 non-null  int64
 14  cutoff_timestamp                140909 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), datetime64[ns](4), float64(10), int64(1), object(8)
memory usage: 25.5+ MB
```

All the formats are proper

In [ ]:
```
# Getting a statistical summary

df.describe()
```

Out[ ]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	cutoff_factor	cutoff_timestamp	actu
count	144316	144316	144316	144316.000000	144316.000000	140909	
mean	2018-09-22 13:05:09.454117120	2018-09-22 17:32:42.435769344	2018-09-23 09:36:54.057172224	963.697698	233.561345	2018-09-23 03:15:10.623693568	
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.000000	2018-09-12 00:10:27	
25%	2018-09-17 02:46:11.004421120	2018-09-17 07:37:35.014584832	2018-09-18 01:29:56.978912	161.000000	22.000000	2018-09-17 19:18:34	
50%	2018-09-22 03:36:19.186585088	2018-09-22 07:35:23.038482944	2018-09-23 02:49:00.936600064	451.000000	66.000000	2018-09-22 21:15:24	
75%	2018-09-27 17:53:19.027942912	2018-09-27 22:01:30.861209088	2018-09-28 12:13:41.675546112	1645.000000	286.000000	2018-09-28 06:12:35	
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.000000	2018-10-06 23:44:12	
std	NaN	NaN	NaN	1038.082976	345.245823	NaN	

In [ ]:
```
df.sample()
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	de
1365	training	2018-09-23 15:30:38.146740	thanos::sroute:2c33e360-7e52-4d2c-a9db-fe24996...	FTL	trip-153771663814650935	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	

1 rows × 24 columns

In [ ]:
```
# Finding the length of the dataset

print(f'Latest Start Date in the DataFrame is - {df["od_start_time"].max()}\n')
print(f'Oldest Start Date in the DataFrame is - {df["od_start_time"].min()}\n')
print(f'Time Delta - {df["od_start_time"].max() - df["od_start_time"].min()}')
```

```
Latest Start Date in the DataFrame is - 2018-10-06 04:27:23.392375

Oldest Start Date in the DataFrame is - 2018-09-12 00:00:16.535741

Time Delta - 24 days 04:27:06.856634
```

In [ ]:
```python
print(f"Latest Trip Creation Time - {df['trip_creation_time'].max()}\n")
print(f"Oldest Trip Creation Time - {df['trip_creation_time'].min()}\n")
print(f"Time Delta - {df['trip_creation_time'].max()-df['trip_creation_time'].min()}\n")
```

```
Latest Trip Creation Time - 2018-10-03 23:59:42.701692

Oldest Trip Creation Time - 2018-09-12 00:00:16.535741

Time Delta - 21 days 23:59:26.165951
```

In [ ]:
```python
print(f'Latest End Date in the DataFrame is - {df["od_end_time"].max()}\n')
print(f'Oldest End Date in the DataFrame is - {df["od_end_time"].min()}\n')
print(f'Time Delta - {df["od_end_time"].max() - df["od_end_time"].min()}')
```

```
Latest End Date in the DataFrame is - 2018-10-08 03:00:24.353479

Oldest End Date in the DataFrame is - 2018-09-12 00:50:10.814399

Time Delta - 26 days 02:10:13.539080
```

The dataset spans over 26 days in Sep - Oct month

Exploratory Data Analysis

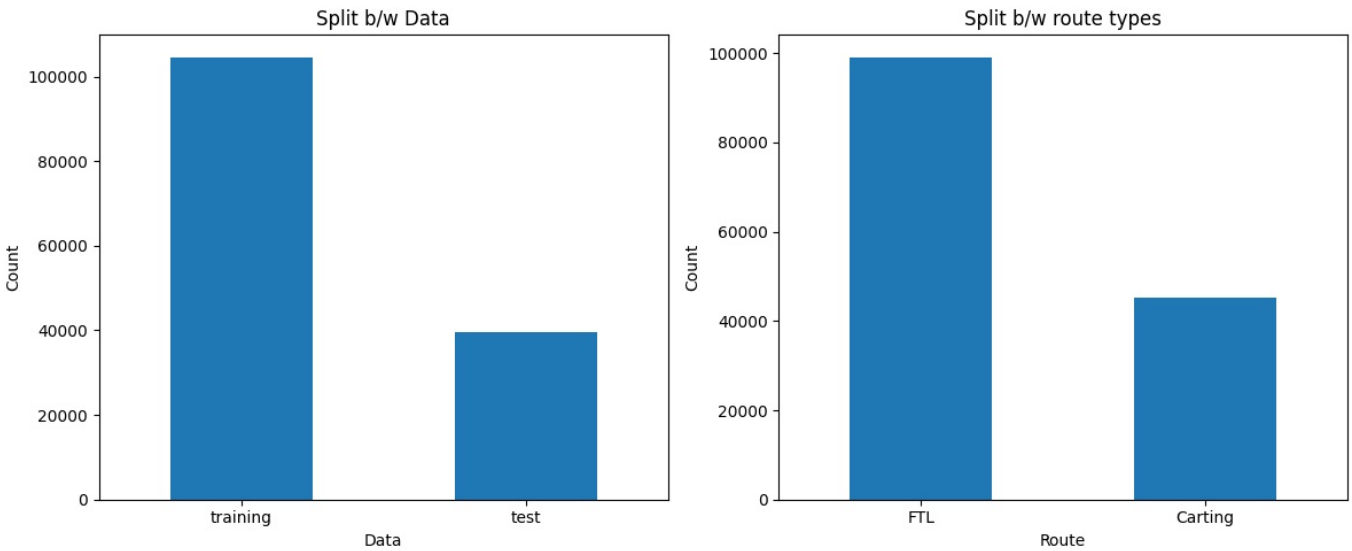
In [ ]:
```python
# checking how many unique values we have in each col

df.nunique()
```

Out[ ]:

	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
is_cutoff	2
cutoff_factor	501
cutoff_timestamp	89862
actual_distance_to_destination	143965
actual_time	3182
osrm_time	1531
osrm_distance	137544
factor	45588
segment_actual_time	746
segment_osrm_time	214
segment_osrm_distance	113497
segment_factor	5663

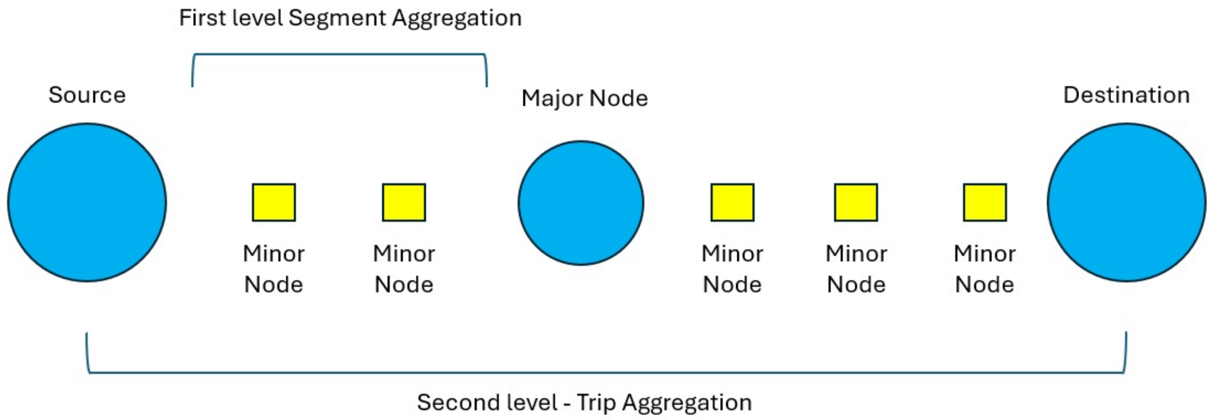
dtype: int64

In [ ]:
```python
# Plotting a graph of Data and route_type cols as they have only 2 unique values

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Plot for 'data' column
df['data'].value_counts().plot(kind='bar', ax=axes[0])
axes[0].set_title('Split b/w Data')
axes[0].set_xlabel('Data')
axes[0].set_ylabel('Count')
axes[0].tick_params(axis='x', rotation=0)

# Plot for 'route' column
df['route_type'].value_counts().plot(kind='bar', ax=axes[1])
axes[1].set_title('Split b/w route types')
axes[1].set_xlabel('Route')
axes[1].set_ylabel('Count')
axes[1].tick_params(axis='x', rotation=0)

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



- The Training and Test data seem to be split in 3:1
- FTL accounts for 2/3 of the Delivery type

Merging of rows and Aggregation



In [ ]:
```python
# Creating a unique segment key based on ['trip_uuid'],['source_center'] and ['destination_center']

segment_cols = ['segment_actual_time', 'segment_osrm_distance', 'segment_osrm_time']
df['segment_key'] = df['trip_uuid'] + '_' + df['source_center'] + '_' + df['destination_center']

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

df[['segment_key', 'segment_actual_time', 'segment_actual_time_sum','segment_osrm_distance',
    'segment_osrm_distance_sum','segment_osrm_time', 'segment_osrm_time_sum']]
```

Out[ ]:

	segment_key	segment_actual_time	segment_actual_time_sum	segment_osrm_distanc
0	trip-153741093647649320_IND388121AAA_IND388620AAB	14.0	14.0	11.965
1	trip-153741093647649320_IND388121AAA_IND388620AAB	10.0	24.0	9.759
2	trip-153741093647649320_IND388121AAA_IND388620AAB	16.0	40.0	10.815
3	trip-153741093647649320_IND388121AAA_IND388620AAB	21.0	61.0	13.022
4	trip-153741093647649320_IND388121AAA_IND388620AAB	6.0	67.0	3.915
...	...	...	...	..
144311	trip-153746066843555182_IND131028AAB_IND000000ACB	12.0	92.0	8.185
144312	trip-153746066843555182_IND131028AAB_IND000000ACB	26.0	118.0	17.372
144313	trip-153746066843555182_IND131028AAB_IND000000ACB	20.0	138.0	20.705
144314	trip-153746066843555182_IND131028AAB_IND000000ACB	17.0	155.0	18.888
144315	trip-153746066843555182_IND131028AAB_IND000000ACB	268.0	423.0	8.808

144316 rows × 7 columns

In [ ]:
```python
df.head()
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destina
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND

5 rows × 28 columns

In [ ]:
```python
# Aggregating at segment level & Creating a dictionary for aggregation at segment level

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

In [ ]:
```python
# Grouping by segment_key and aggregating
segment_agg_df = df.groupby('segment_key').agg(segment_dictionary).reset_index()
segment_agg_df = segment_agg_df.sort_values(by=['segment_key','od_end_time'])
segment_agg_df
```

Out[ ]:

	segment_key	data	trip_creation_time	route_schedule_uuid	route_type	
0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	15367104
1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	15367104
2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	15367104
3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	15367104
4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	15367104
...	...	...	...	...	...	
26217	trip-153861115439069069_IND628204AAA_IND627657AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26218	trip-153861115439069069_IND628613AAA_IND627005AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26219	trip-153861115439069069_IND628801AAA_IND628204AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26220	trip-153861118270144424_IND583119AAA_IND583101AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	15386111
26221	trip-153861118270144424_IND583201AAA_IND583119AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	15386111

26222 rows × 20 columns

In [ ]:
```python
segment_agg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26222 entries, 0 to 26221
Data columns (total 20 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   segment_key                      26222 non-null  object
 1   data                             26222 non-null  object
 2   trip_creation_time               26222 non-null  datetime64[ns]
 3   route_schedule_uuid              26222 non-null  object
 4   route_type                       26222 non-null  object
 5   trip_uuid                        26222 non-null  object
 6   source_center                    26222 non-null  object
 7   source_name                      26222 non-null  object
 8   destination_center               26222 non-null  object
 9   destination_name                 26222 non-null  object
 10  od_start_time                    26222 non-null  datetime64[ns]
 11  od_end_time                      26222 non-null  datetime64[ns]
 12  start_scan_to_end_scan           26222 non-null  float64
 13  actual_distance_to_destination   26222 non-null  float64
 14  actual_time                      26222 non-null  float64
 15  osrm_time                        26222 non-null  float64
 16  osrm_distance                    26222 non-null  float64
 17  segment_actual_time_sum          26222 non-null  float64
 18  segment_osrm_distance_sum        26222 non-null  float64
 19  segment_osrm_time_sum            26222 non-null  float64
dtypes: datetime64[ns](3), float64(8), object(9)
memory usage: 4.0+ MB
```

The number of rows are reduced to 26222. A significant decrease from the 144K rows before

In [ ]:
```python
segment_agg_df.nunique()
```

Out[ ]:

	0
segment_key	26222
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	26222
od_end_time	26222
start_scan_to_end_scan	1914
actual_distance_to_destination	26193
actual_time	1657
osrm_time	560
osrm_distance	25871
segment_actual_time_sum	1676
segment_osrm_distance_sum	25948
segment_osrm_time_sum	1102

dtype: int64

Feature Engineering

In [ ]:

```
# 1. Calculate time taken between od_start_time and od_end_time and keeping it as a feature named od_time_diff_

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

Out[ ]:

	segment_key	data	trip_creation_time	route_schedule_uuid	route_type	
0	trip-153671041653548748_IND209304AAA_IND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	15367104
1	trip-153671041653548748_IND462022AAA_IND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	15367104
2	trip-153671042288605164_IND561203AAB_IND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	15367104
3	trip-153671042288605164_IND572101AAA_IND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	15367104
4	trip-153671043369099517_IND000000ACB_IND160002AAC	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	15367104
...	...	...	...	...	...	
26217	trip-153861115439069069_IND628204AAA_IND627657AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26218	trip-153861115439069069_IND628613AAA_IND627005AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26219	trip-153861115439069069_IND628801AAA_IND628204AAA	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	15386111
26220	trip-153861118270144424_IND583119AAA_IND583101AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	15386111
26221	trip-153861118270144424_IND583201AAA_IND583119AAA	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	15386111

26222 rows × 22 columns

In [ ]:
```
trip_dictionary = {

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

In [ ]:
```
trip_df = segment_agg_df.groupby('trip_uuid').agg(trip_dictionary).reset_index(drop = True)
```

In [ ]:
```
trip_df.shape
```

Out[ ]: (14787, 18)

In [ ]:
```
trip_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14787 non-null  object
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  object
 4   trip_uuid                       14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  od_time_diff_hour               14787 non-null  float64
 11  actual_distance_to_destination  14787 non-null  float64
 12  actual_time                     14787 non-null  float64
 13  osrm_time                       14787 non-null  float64
 14  osrm_distance                   14787 non-null  float64
 15  segment_actual_time_sum         14787 non-null  float64
 16  segment_osrm_distance_sum       14787 non-null  float64
 17  segment_osrm_time_sum           14787 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(8)
memory usage: 2.0+ MB
```

In [ ]:

```
# Creating a Copy for further processing

data_1 = trip_df.copy()
```

In [ ]:

```
# using regex pattern to seperate the city,place,state

def extract_info(name):
 pattern = r'^(?P<city>[^\s_]+)_?(?P<place>[^\(\)]*)\s?\((?P<state>[A-Za-z\s&]+)\)$'
 match = re.match(pattern, name)
 if match:
  city = match.group('city').strip()
  place = match.group('place').strip() if match.group('place') else city
  state = match.group('state').strip()
  return city, place, state
 else:
  return None, None, None
```

In [ ]:

```
# Creating new cols for city and states of source and destination

data_1[['source_city', 'source_place', 'source_state']] = data_1['source_name'].apply(lambda x: pd.Series(extra
```

In [ ]:

```
data_1[['destination_city', 'destination_place', 'destination_state']] = data_1['destination_name'].apply(lambda
```

In [ ]:

```
data_1
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072AAB	Mumbai Hub (Maharashtra
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583101AAA	Bellary_Dc (Karnataka
...	...	...	...	...	...	...	..
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND121004AAB	FBD_Balabhgarh_DPC (Haryana
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND208006AAA	Kanpur_GovndNgr_DC (Uttar Pradesh
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND627005AAA	Tirunelveli_VdkkuSrt_ (Tamil Nadu
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka

14787 rows × 24 columns

In [ ]:

```
print(f'Delhivery has - {data_1["source_state"].nunique()} different source states')
print(f'Delhivery has - {data_1["source_city"].nunique()} different source cities\n')
print(f'Delhivery has - {data_1["destination_state"].nunique()} different destination states')
print(f'Delhivery has - {data_1["destination_city"].nunique()} different destination cities\n')
```

```
Delhivery has - 29 different source states
Delhivery has - 714 different source cities

Delhivery has - 31 different destination states
Delhivery has - 840 different destination cities
```

In [ ]:

```
data_1['source_state'].value_counts()
```

Out[ ]:

source_state	count
Maharashtra	2714
Karnataka	2143
Haryana	1823
Tamil Nadu	1039
Telangana	784
Uttar Pradesh	760
Gujarat	750
Delhi	725
West Bengal	665
Punjab	536
Rajasthan	514
Andhra Pradesh	435
Bihar	351
Madhya Pradesh	318
Kerala	289
Assam	268
Jharkhand	160
Uttarakhand	114
Orissa	107
Chandigarh	93
Goa	65
Chhattisgarh	43
Himachal Pradesh	34
Jammu & Kashmir	17
Dadra and Nagar Haveli	15
Pondicherry	12
Nagaland	5
Arunachal Pradesh	4
Mizoram	4

dtype: int64

In [ ]:

```
data_1['source_city'].value_counts()
```

Out[ ]:

	count
source_city	
Gurgaon	1128
Bengaluru	1052
Mumbai	968
Bhiwandi	697
Bangalore	648
...	...
Mahasamund	1
Mandla	1
Janakpuri	1
Phulera	1
Sandur	1

714 rows × 1 columns

dtype: int64

In [ ]:

```
data_1['destination_state'].value_counts()
```

Out[ ]:

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

In [ ]:

```
data_1['destination_city'].value_counts()
```

Out[ ]:

	count
destination_city	
Mumbai	1202
Bengaluru	1088
Gurgaon	877
Delhi	554
Bangalore	551
...	...
Daman	1
Chincholi	1
Malout	1
Thakurdwara	1
Manthani	1

840 rows × 1 columns

dtype: int64

Bengaluru appears twice - Once as Bangalore and another time as Bengaluru. Therefore we have to merge them

In [ ]:

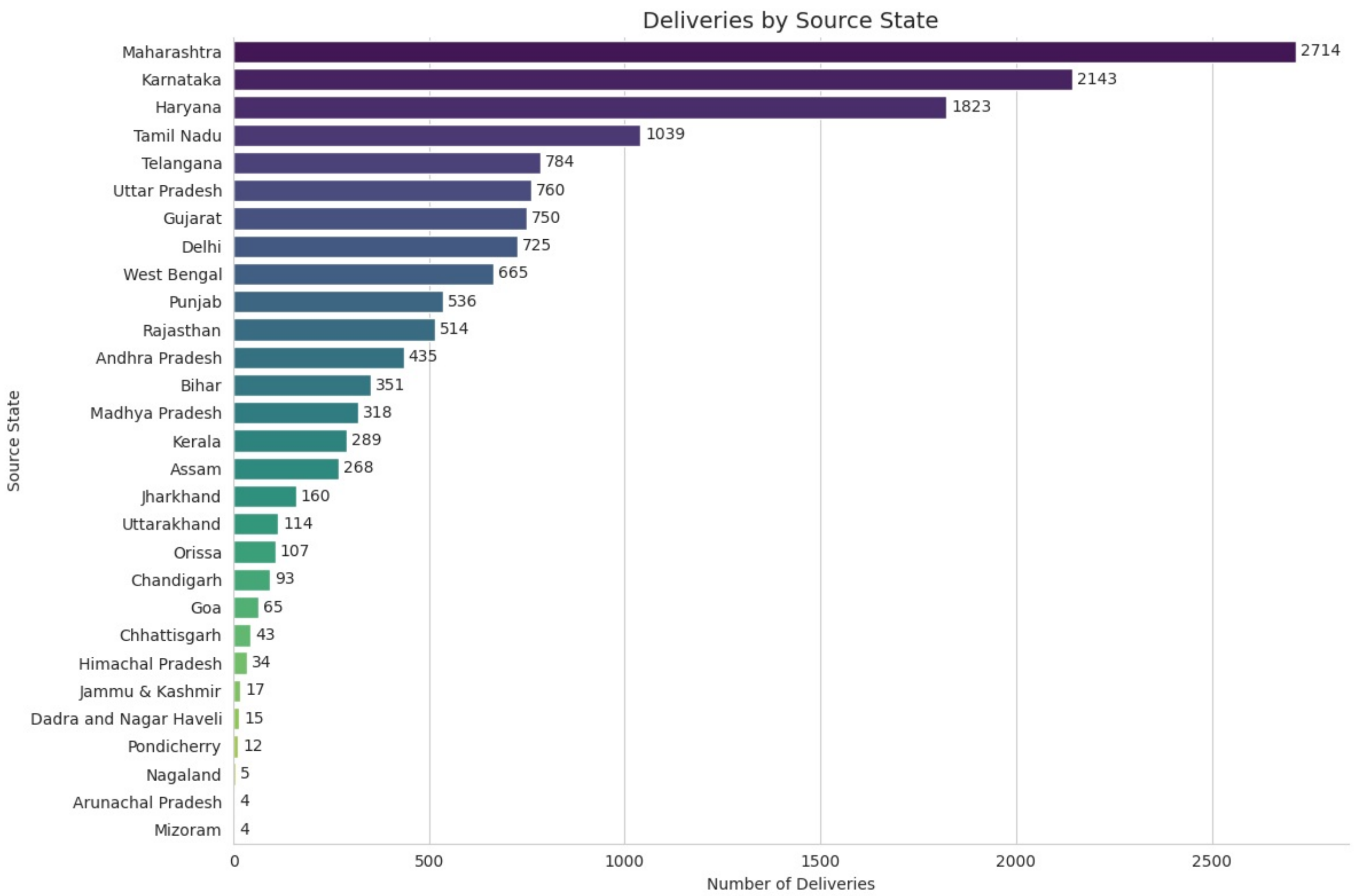
```
# Changing Bangalore to Bengaluru

data_1.loc[data_1.source_city=='Bangalore','source_city']='Bengaluru'
data_1.loc[data_1.destination_city=='Bangalore','destination_city']='Bengaluru'
```

Plotting Different State and city counts for source and destination

In [ ]:

```
# Calculate counts
state_counts = data_1['source_state'].value_counts().to_frame().reset_index()
state_counts.columns = ['State', 'Count']

# Plot
plt.figure(figsize=(12, 8))
sns.set_style("whitegrid")
colors = sns.color_palette("viridis", len(state_counts))

# Plot horizontal bar chart
barplot = sns.barplot(
    x='Count', y='State', data=state_counts,
    palette=colors
)

# Add value labels to the bars
for container in barplot.containers:
    barplot.bar_label(container, label_type='edge', padding=3)

# Customize axes and title
plt.xlabel('Number of Deliveries')
plt.ylabel('Source State')
plt.title('Deliveries by Source State', fontsize=14)
plt.tight_layout()
sns.despine()

# Show plot
plt.show()
```

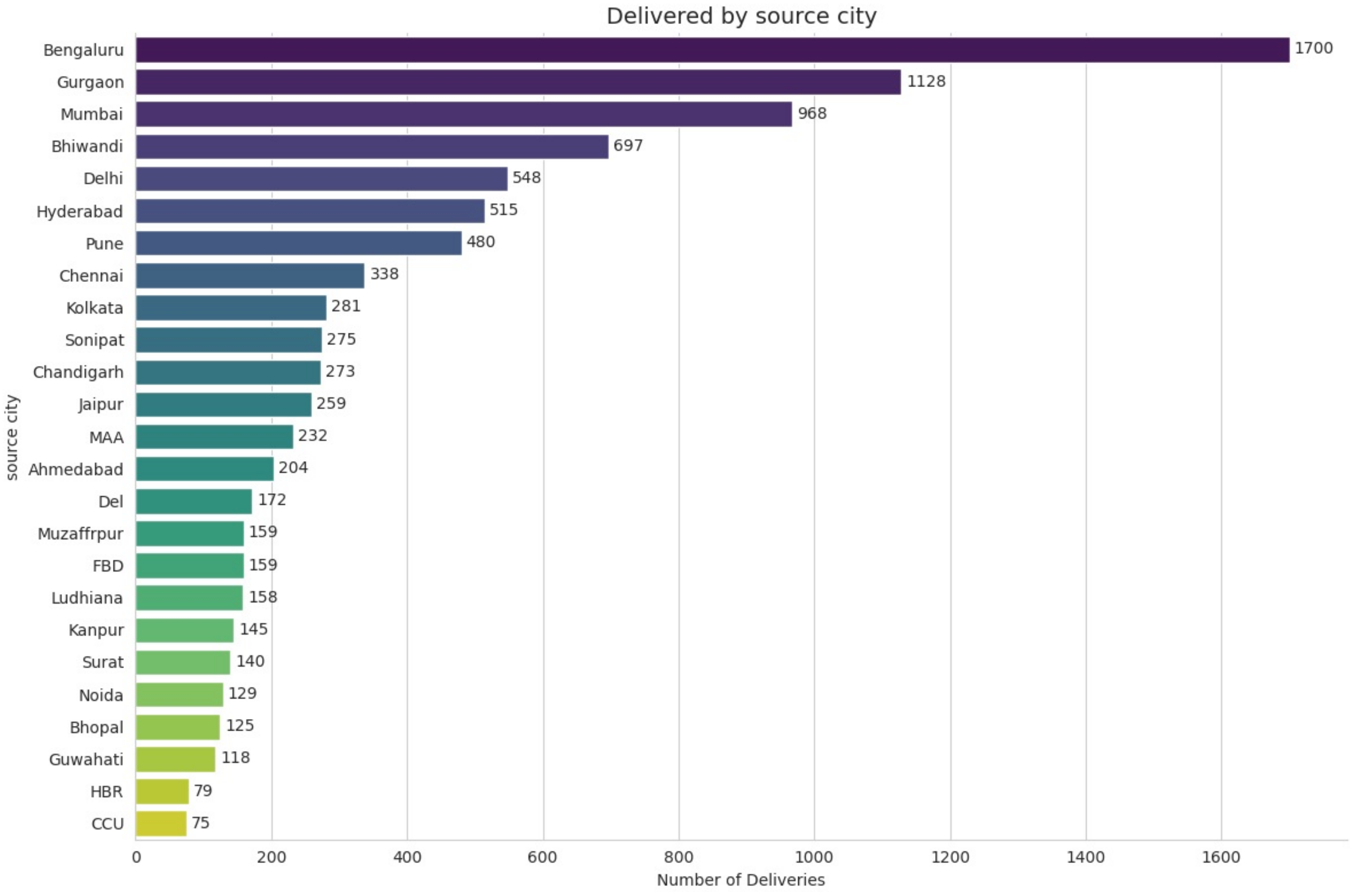
```
<ipython-input-123-26b8b879d80b>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable
to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(
```



In [ ]:

```
# Calculate counts
source_city = data_1['source_city'].value_counts().to_frame().reset_index()[:25]
source_city.columns = ['city', 'Count']

# Plot
plt.figure(figsize=(12, 8))
sns.set_style("whitegrid")
colors = sns.color_palette("viridis", len(source_city))

# Plot horizontal bar chart
barplot = sns.barplot(
    x='Count', y='city', data=source_city,
    palette=colors
)

# Add value labels to the bars
for container in barplot.containers:
    barplot.bar_label(container, label_type='edge', padding=3)

# Customize axes and title
plt.xlabel('Number of Deliveries')
plt.ylabel('source city')
plt.title('Delivered by source city', fontsize=14)
plt.tight_layout()
sns.despine()

# Show plot
plt.show()
```

```
<ipython-input-124-0146a1864fdd>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable
to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(
```



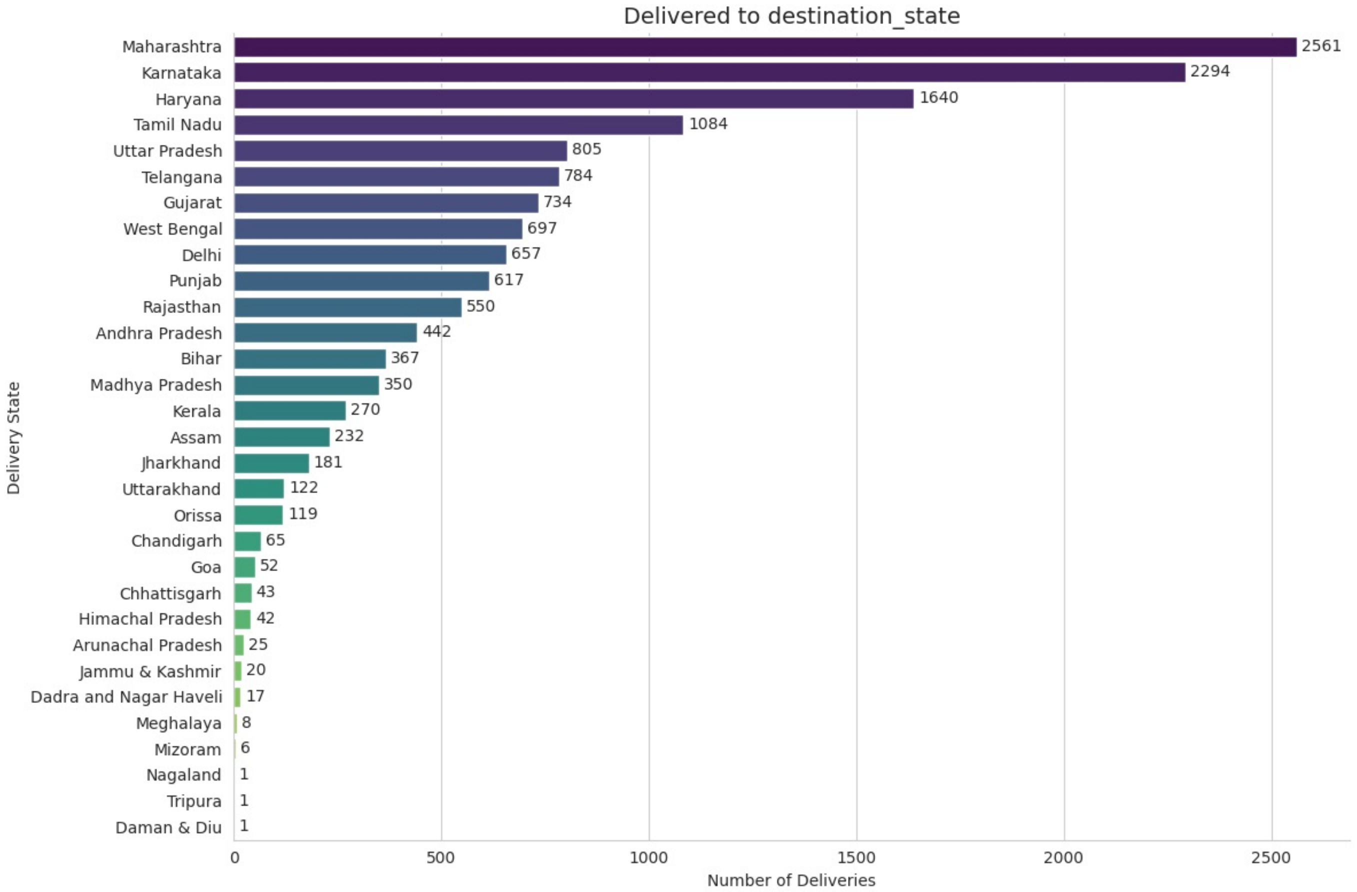
Insights: Source Trends

- **High-Engagement States:**
  - The majority of deliveries are sourced from states such as **Maharastra, karnataka, Tamil Nadu, Haryana, and telangana**, indicating strong hub of storage and manufacturing.
- **Top Destination Cities:**
  - Cities like **Bengaluru,Gurgaon,Mumbai,Bindiwadi** emerged as the most frequent source cities where the packages are shipped from

In [ ]:

```
# Calculate counts
destination_states = data_1['destination_state'].value_counts().to_frame().reset_index()
destination_states.columns = ['State', 'Count']

# Plot
plt.figure(figsize=(12, 8))
sns.set_style("whitegrid")
colors = sns.color_palette("viridis", len(destination_states))

# Plot horizontal bar chart
barplot = sns.barplot(
    x='Count', y='State', data=destination_states,
    palette=colors
)

# Add value labels to the bars
for container in barplot.containers:
    barplot.bar_label(container, label_type='edge', padding=3)

# Customize axes and title
plt.xlabel('Number of Deliveries')
plt.ylabel('Delivery State')
plt.title('Delivered to destination state', fontsize=14)
plt.tight_layout()
sns.despine()

# Show plot
plt.show()
```

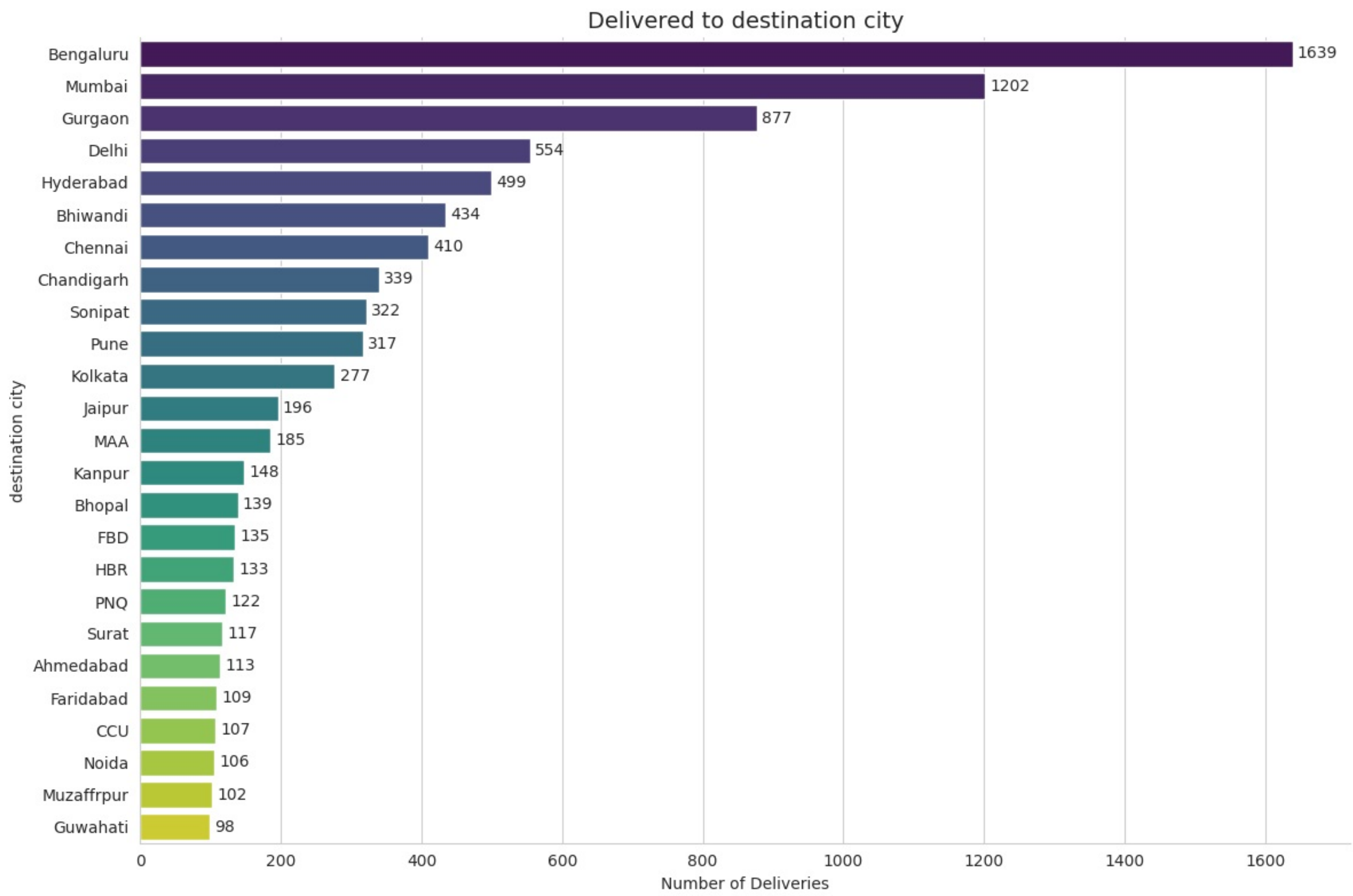
```
<ipython-input-125-8b3671bbc40a>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable
to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(
```



In [ ]:

```python
# Calculate counts
destination_city = data_1['destination_city'].value_counts().to_frame().reset_index()[:25]
destination_city.columns = ['city', 'Count']

# Plot
plt.figure(figsize=(12, 8))
sns.set_style("whitegrid")
colors = sns.color_palette("viridis", len(destination_city))

# Plot horizontal bar chart
barplot = sns.barplot(
    x='Count', y='city', data=destination_city,
    palette=colors
)

# Add value labels to the bars
for container in barplot.containers:
    barplot.bar_label(container, label_type='edge', padding=3)

# Customize axes and title
plt.xlabel('Number of Deliveries')
plt.ylabel('destination city')
plt.title('Delivered to destination city', fontsize=14)
plt.tight_layout()
sns.despine()

# Show plot
plt.show()
```

```
<ipython-input-126-d7fa5d45516e>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable
to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(
```



Insights: Destination Trends

- **High-Engagement States:**
  - The majority of deliveries are directed towards states such as **Maharastra, karnataka, Tamil Nadu, Haryana, and Uttar Pradesh**, indicating strong customer demand and business activity in these regions during the given period.
- **Top Destination Cities:**
  - Cities like **Bengaluru, Mumbai, Gurgaon,Delhi, Hyderabad** emerged as the most frequently targeted destinations, highlighting their importance as key urban hubs in the delivery network.

In [ ]:
```
data_1['route'] = data_1['source_name'] +' to '+ data_1['destination_name']
data_1['route'].value_counts()
```

Out[ ]:

route	count
Bangalore_Nelmngla_H (Karnataka) to Bengaluru_KGAirprt_HB (Karnataka)	151
Gurgaon_Bilaspur_HB (Haryana) to Gurgaon_Bilaspur_HB (Haryana)	123
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
...	...
Kuthuparamba_IdstrlAr_D (Kerala) to Kuthuparamba_IdstrlAr_D (Kerala)	1
Kozhikode_Central_H_4 (Kerala) to Thachnttukra_Nattukal_D (Kerala)	1
Mahasamund_RajpurRD_D (Chhattisgarh) to Durg_Bhilai_DC (Chhattisgarh)	1
Karad_Mundhe_D (Maharashtra) to Kolhapur_Central_H_2 (Maharashtra)	1
Bhiwani_DC (Haryana) to Loharu_BstndDPP_D (Haryana)	1

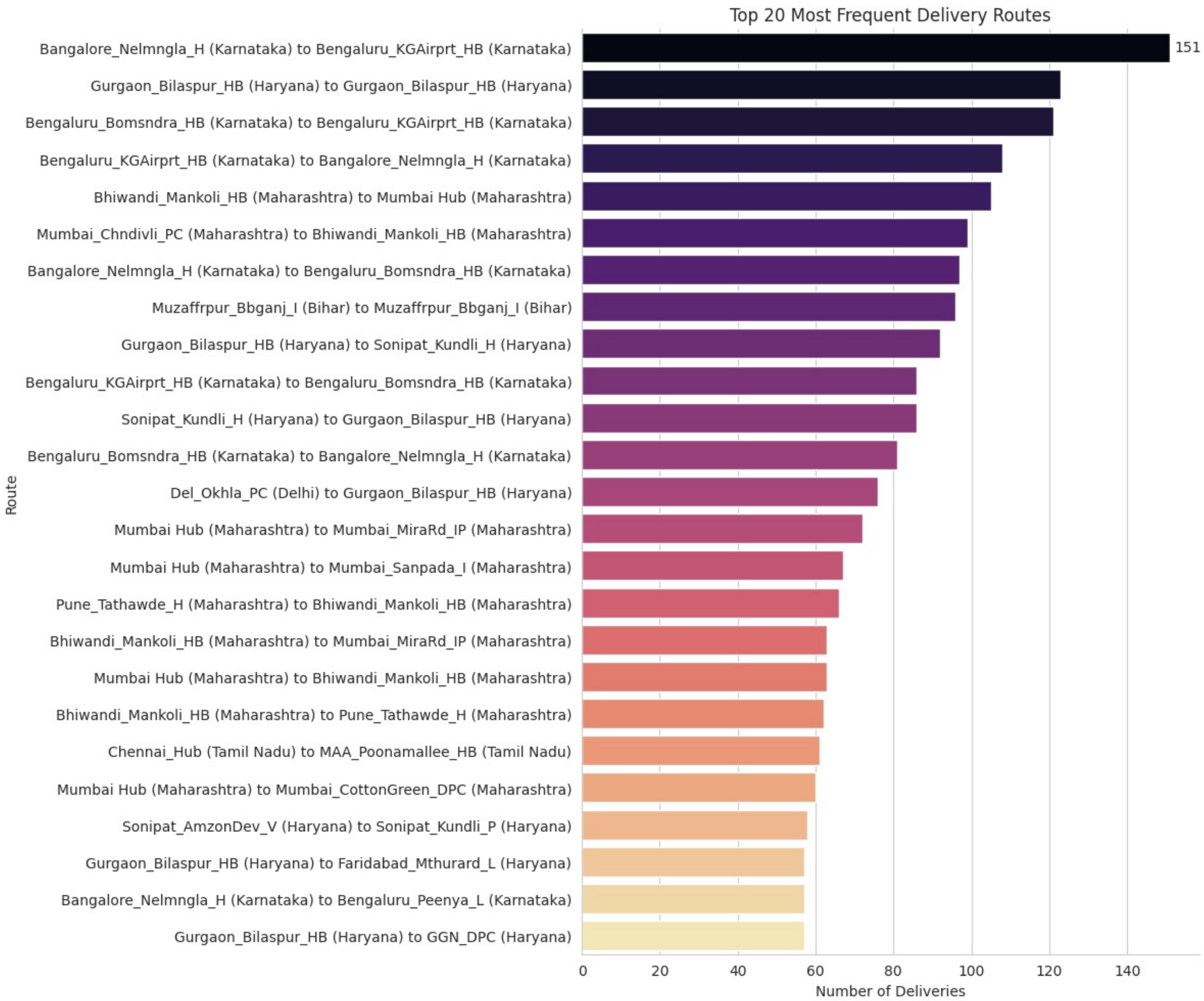
2165 rows × 1 columns

dtype: int64

In [ ]:

```
# Get route counts
route_counts = data_1['route'].value_counts().head(25).reset_index()
route_counts.columns = ['Route', 'Count']
route_counts = route_counts.sort_values('Count', ascending=False)

# Plot
plt.figure(figsize=(12, 10))
sns.set_style("whitegrid")
barplot = sns.barplot(x='Count', y='Route', data=route_counts, palette='magma')

# Label bars
barplot.bar_label(barplot.containers[0], label_type='edge', padding=3)

# Titles and labels
plt.xlabel('Number of Deliveries')
plt.ylabel('Route')
plt.title('Top 20 Most Frequent Delivery Routes')
plt.tight_layout()
sns.despine()

# Show plot
plt.show()
```

```
<ipython-input-128-18251887d91a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable
to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(x='Count', y='Route', data=route_counts, palette='magma')
```



Insights: Busiest Intra-Karnataka Routes

- The route from **Bangalore_Nelamangala_H** to **Bengaluru_KGAirport_HB** and **Bengaluru_Bommasandra_HB** records the highest volume of packages, with **151** and **127** packages sent respectively.
- The corridor between **Bengaluru_Bommasandra_HB** and **Bengaluru_KGAirport_HB** is also notably active, with **121** packages dispatched.
- The reverse direction, from **Bengaluru_KGAirport_HB** to **Bangalore_Nelamangala_H**, shows moderate flow, with **108** packages delivered.
- These figures underscore **Bengaluru's critical role as a central logistics hub within Karnataka**, efficiently managing high inter-

node package traffic.

In - Depth Analysis

In [ ]:
```
# Creating a copy for further analysis

analysis_df = data_1.copy()
```

In [ ]:
```
#  Extracting the important time metrics for in depth analysis

analysis_df['trip_creation_month'] = analysis_df['trip_creation_time'].dt.month
analysis_df['trip_creation_day'] = analysis_df['trip_creation_time'].dt.day
analysis_df['trip_creation_hour'] = analysis_df['trip_creation_time'].dt.hour
analysis_df['trip_creation_weekday'] = analysis_df['trip_creation_time'].dt.weekday
analysis_df['trip_creation_week'] = analysis_df['trip_creation_time'].dt.isocalendar().week
analysis_df
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name
0	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh
1	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka
2	training	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	trip-153671043369099517	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana
3	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Carting	trip-153671046011330457	IND400072AAB	Mumbai Hub (Maharashtra
4	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	trip-153671052974046625	IND583101AAA	Bellary_Dc (Karnataka
...	...	...	...	...	...	...	..
14782	test	2018-10-03 23:55:56.258533	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153861095625827784	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab
14783	test	2018-10-03 23:57:23.863155	thanos::sroute:b30e1ec3-3bfa-4bd2-a7fb-3b75769...	Carting	trip-153861104386292051	IND121004AAB	FBD_Balabhgarh_DPC (Haryana
14784	test	2018-10-03 23:57:44.429324	thanos::sroute:5609c268-e436-4e0a-8180-3db4a74...	Carting	trip-153861106442901555	IND208006AAA	Kanpur_GovndNgr_DC (Uttar Pradesh
14785	test	2018-10-03 23:59:14.390954	thanos::sroute:c5f2ba2c-8486-4940-8af6-d1d2a6a...	Carting	trip-153861115439069069	IND627005AAA	Tirunelveli_VdkkuSrt_ (Tamil Nadu
14786	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424	IND583119AAA	Sandur_WrdN1DPP_D (Karnataka

14787 rows × 30 columns

In [ ]:
```
analysis_df.describe()
```

Out[ ]:

	trip_creation_time	start_scan_to_end_scan	od_time_diff_hour	actual_distance_to_destination	actual_time	osrm_time
count	14787	14787.000000	14787.000000	14787.000000	14787.000000	14787.000000
mean	2018-09-22 12:26:28.269885696	529.429025	8.838559	164.090196	356.306012	160.990938
min	2018-09-12 00:00:16.535741	23.000000	0.391024	9.002461	9.000000	6.000000
25%	2018-09-17 02:38:18.128431872	149.000000	2.494975	22.777099	67.000000	29.000000
50%	2018-09-22 03:39:19.609193984	279.000000	4.661846	48.287894	148.000000	60.000000
75%	2018-09-27 19:23:14.074359552	632.000000	10.558962	163.591258	367.000000	168.000000
max	2018-10-03 23:59:42.701692	7898.000000	131.642533	2186.531787	6265.000000	2032.000000
std	NaN	658.254936	10.973591	305.502982	561.517936	271.459495

In [ ]:

```
analysis_df.sample(5)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name
5200	training	2018-09-19 02:10:24.254107	thanos::sroute:b680eb59-87ae-4c5c-8a7f-73f2cac...	FTL	trip-153732302425384294	IND312605AAB	Pratapgarh_Nimachrd_D (Rajasthan
12339	test	2018-09-29 19:40:58.970199	thanos::sroute:86ccbef6-71d7-4cf8-a2c5-aea371a...	FTL	trip-153825005896983644	IND495677AAA	Korba_Tilknagr_DC (Chhattisgarh
1425	training	2018-09-13 22:56:30.390163	thanos::sroute:bf427bc4-6fe3-4868-bc71-5c55d4c...	Carting	trip-153687939038990799	IND209206AAB	Ghatampur_StatinRD_D (Uttar Pradesh
11826	test	2018-09-28 22:25:15.504856	thanos::sroute:caf62782-95cc-4d47-a071-d1c7038...	FTL	trip-153817351550461336	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh
2220	training	2018-09-15 00:13:21.601914	thanos::sroute:8a120994-f577-4491-9e4b-b7e4a14...	Carting	trip-153697040160165046	IND160002AAC	Chandigarh_Mehmdpur_H (Punjab

5 rows × 30 columns

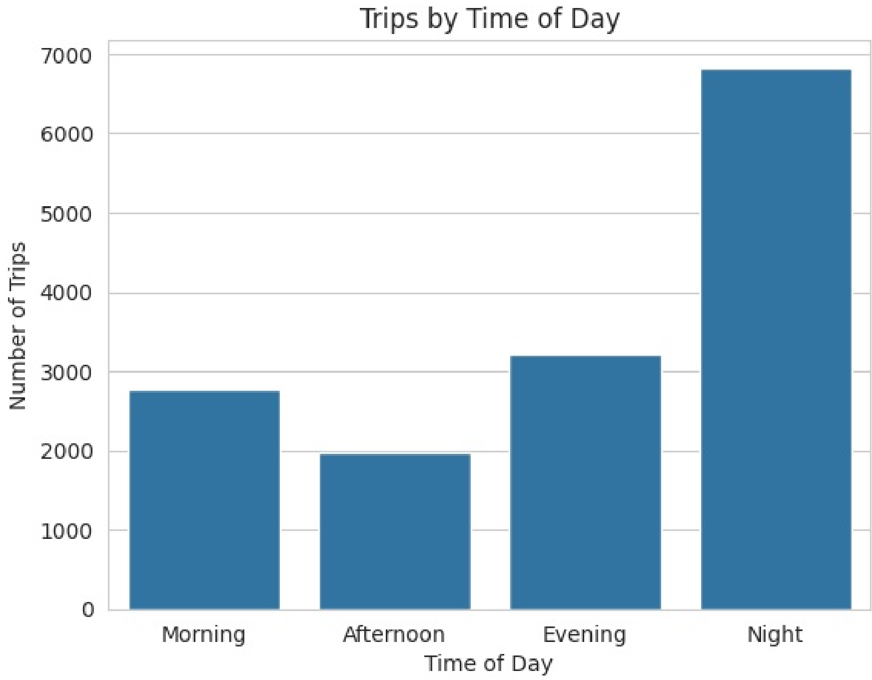
In [ ]:

```
# Bucketing trip_creation_hour into parts of the day
def hour_bucket(hour):
    if 5 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
        return 'Night'

analysis_df['trip_creation_hour_bucket'] = analysis_df['trip_creation_hour'].apply(hour_bucket)

# Bucketing trip_creation_day into start/mid/end of month
analysis_df['trip_creation_day_bucket'] = pd.cut(
    analysis_df['trip_creation_day'],
    bins=[0, 10, 20, 31],
    labels=['Start of Month', 'Mid Month', 'End of Month']
)

# Bucketing trip_creation_week into quarters of the year
analysis_df['trip_creation_week_bucket'] = pd.cut(
    analysis_df['trip_creation_week'],
    bins=[0, 13, 26, 39, 53],
    labels=['Q1 Weeks', 'Q2 Weeks', 'Q3 Weeks', 'Q4 Weeks']
)

# Creating readable versions for weekday and month
analysis_df['trip_creation_weekday_name'] = analysis_df['trip_creation_time'].dt.day_name()
analysis_df['trip_creation_month_name'] = analysis_df['trip_creation_time'].dt.month_name()
```

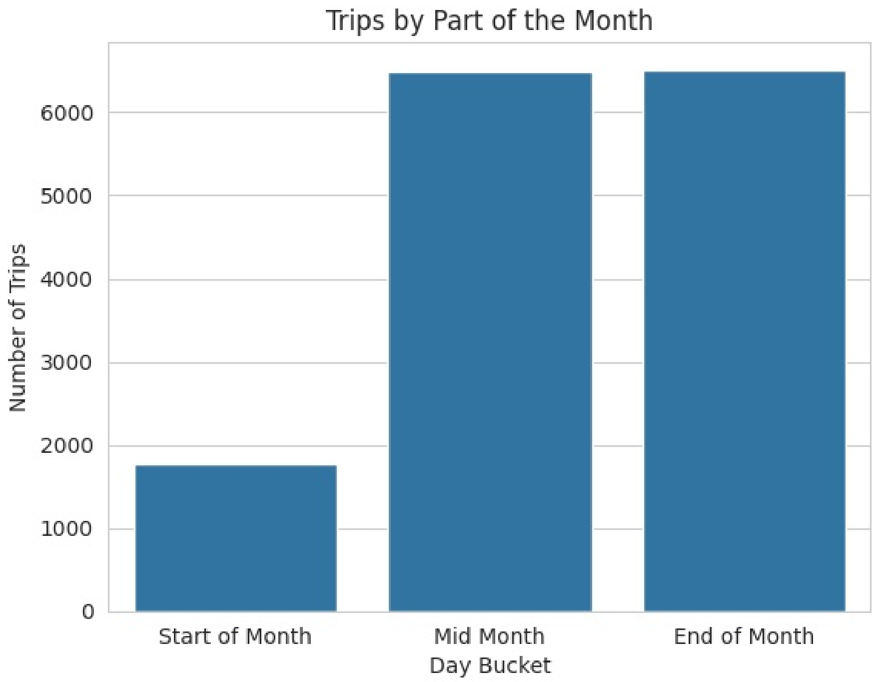
In [ ]:

```
sns.countplot(data=analysis_df, x='trip_creation_hour_bucket', order=['Morning', 'Afternoon', 'Evening', 'Night
plt.title('Trips by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Number of Trips')
```

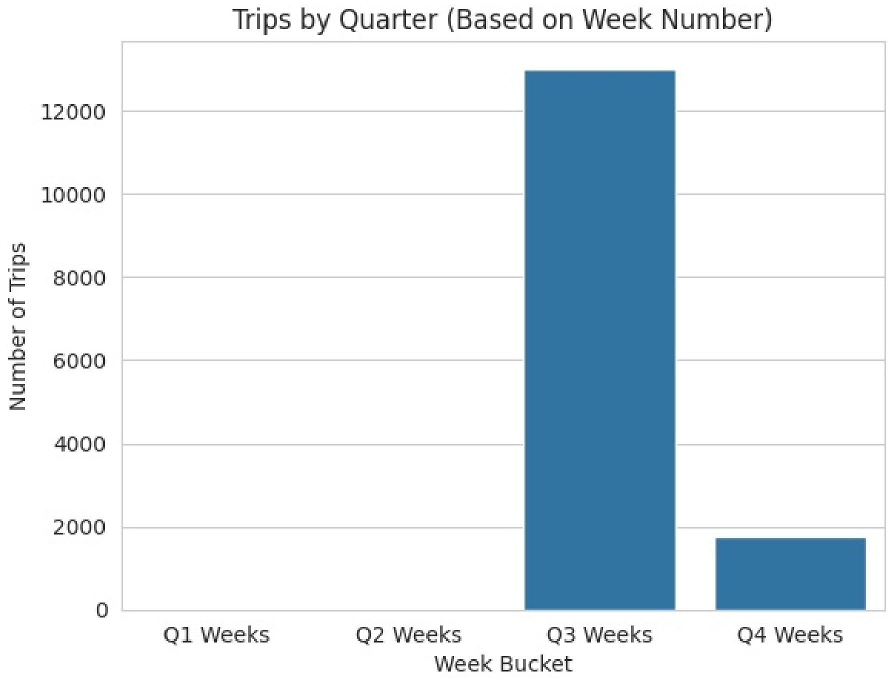
```
plt.show()
```



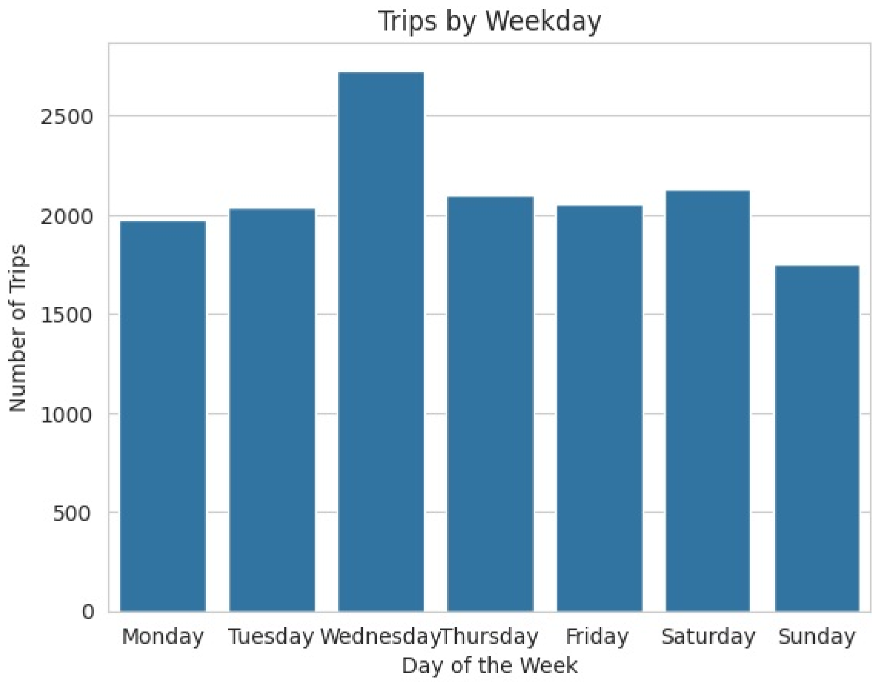
In [ ]:
```
sns.countplot(data=analysis_df, x='trip_creation_day_bucket')
plt.title('Trips by Part of the Month')
plt.xlabel('Day Bucket')
plt.ylabel('Number of Trips')
plt.show()
```



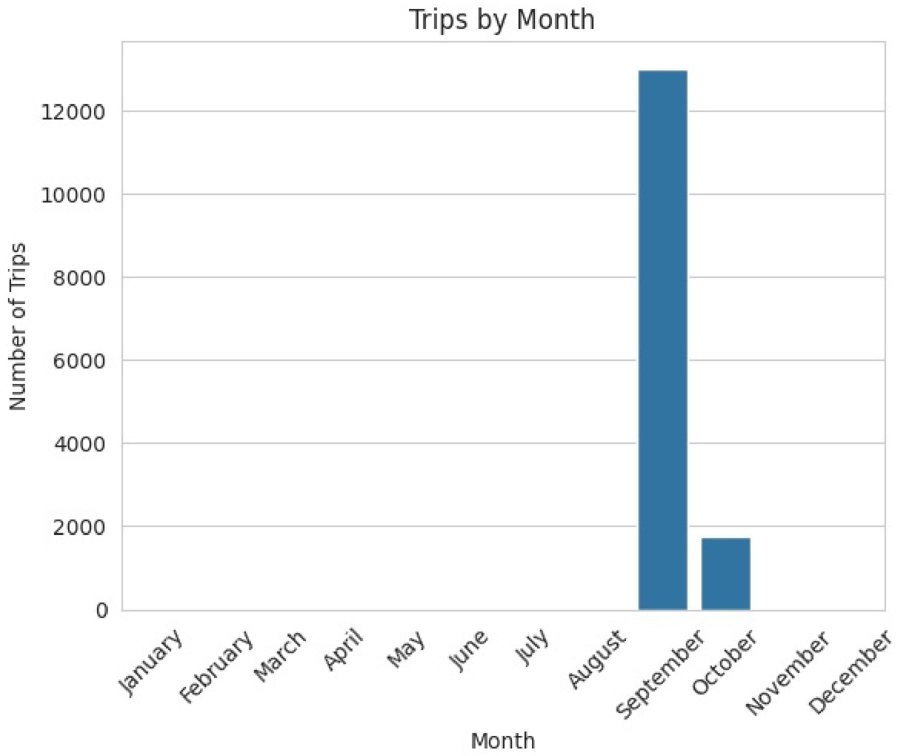
In [ ]:
```
sns.countplot(data=analysis_df, x='trip_creation_week_bucket')
plt.title('Trips by Quarter (Based on Week Number)')
plt.xlabel('Week Bucket')
plt.ylabel('Number of Trips')
plt.show()
```



In [ ]:
```
sns.countplot(data=analysis_df, x='trip_creation_weekday_name',
              order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Trips by Weekday')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Trips')
plt.show()
```



In [ ]:
```
sns.countplot(data=analysis_df, x='trip_creation_month_name',
              order=['January','February','March','April','May','June','July','August','September','October','N
plt.title('Trips by Month')
plt.xlabel('Month')
plt.ylabel('Number of Trips')
plt.xticks(rotation=45)
plt.show()
```



- Most trips are made during night time
- Most trips are made on Wednesday
- The others do not give much insight as the data is a small sample between sep and oct

In [ ]:
```
analysis_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 35 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            14787 non-null  object
 1   trip_creation_time              14787 non-null  datetime64[ns]
 2   route_schedule_uuid             14787 non-null  object
 3   route_type                      14787 non-null  object
 4   trip_uuid                       14787 non-null  object
 5   source_center                   14787 non-null  object
 6   source_name                     14787 non-null  object
 7   destination_center              14787 non-null  object
 8   destination_name                14787 non-null  object
 9   start_scan_to_end_scan          14787 non-null  float64
 10  od_time_diff_hour               14787 non-null  float64
 11  actual_distance_to_destination  14787 non-null  float64
 12  actual_time                     14787 non-null  float64
 13  osrm_time                       14787 non-null  float64
 14  osrm_distance                   14787 non-null  float64
 15  segment_actual_time_sum         14787 non-null  float64
 16  segment_osrm_distance_sum       14787 non-null  float64
 17  segment_osrm_time_sum           14787 non-null  float64
 18  source_city                     14787 non-null  object
 19  source_place                    14787 non-null  object
 20  source_state                    14787 non-null  object
 21  destination_city                14787 non-null  object
 22  destination_place               14787 non-null  object
 23  destination_state               14787 non-null  object
 24  route                           14787 non-null  object
 25  trip_creation_month             14787 non-null  int32
 26  trip_creation_day               14787 non-null  int32
 27  trip_creation_hour              14787 non-null  int32
 28  trip_creation_weekday           14787 non-null  int32
 29  trip_creation_week              14787 non-null  UInt32
 30  trip_creation_hour_bucket       14787 non-null  object
 31  trip_creation_day_bucket        14787 non-null  category
 32  trip_creation_week_bucket       14787 non-null  category
 33  trip_creation_weekday_name      14787 non-null  object
 34  trip_creation_month_name        14787 non-null  object
dtypes: UInt32(1), category(2), datetime64[ns](1), float64(9), int32(4), object(18)
memory usage: 3.5+ MB
```

Outlier Detection and treatment

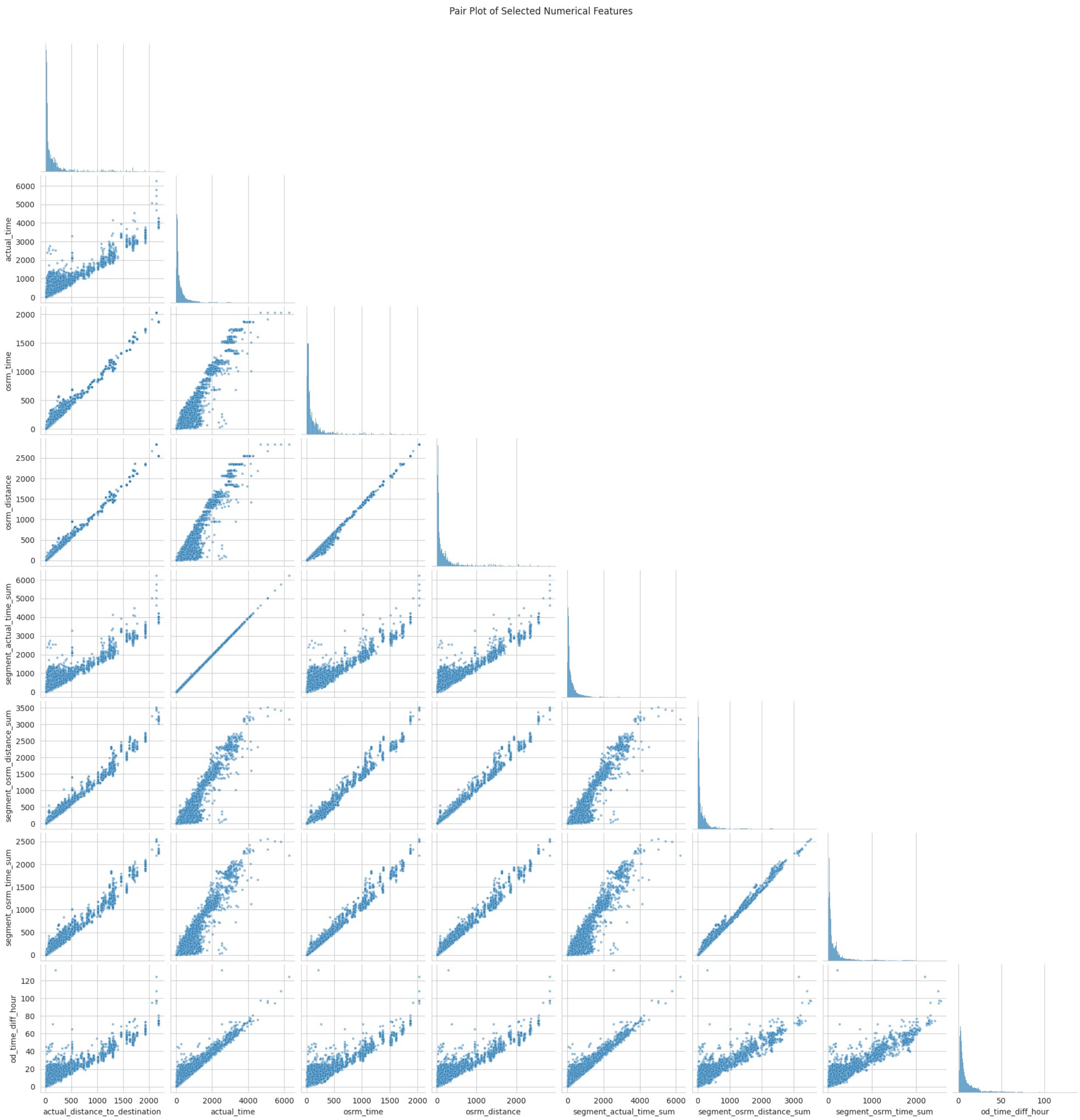
In [ ]:
```
analysis_df.skew(numeric_only = True)
```

Out[ ]:

	0
start_scan_to_end_scan	2.895337
od_time_diff_hour	2.893306
actual_distance_to_destination	3.562931
actual_time	3.375178
osrm_time	3.455256
osrm_distance	3.553619
segment_actual_time_sum	3.372042
segment_osrm_distance_sum	3.714017
segment_osrm_time_sum	3.602915
trip_creation_month	2.337439
trip_creation_day	-0.695241
trip_creation_hour	-0.206092
trip_creation_weekday	0.065904
trip_creation_week	0.181308

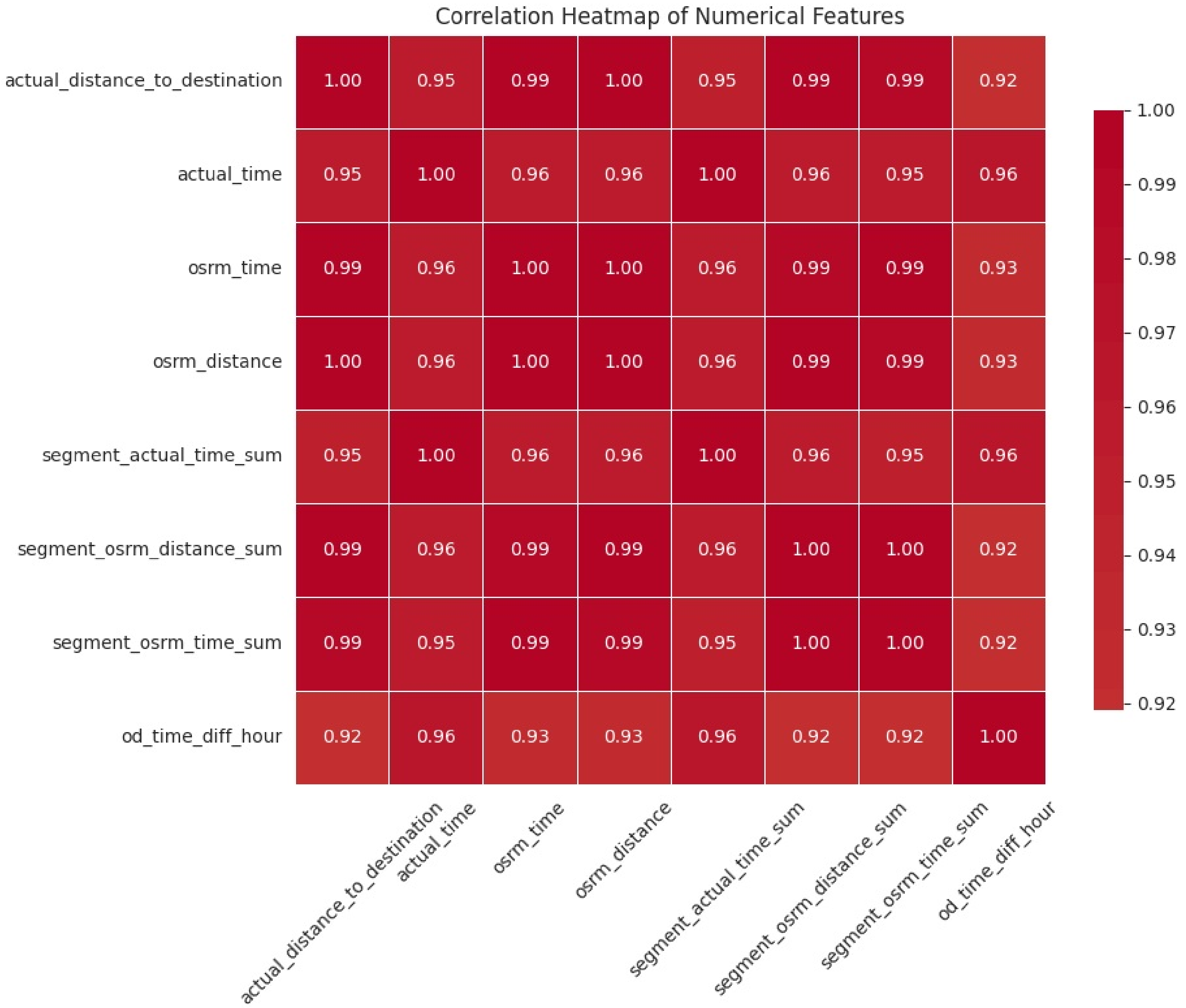
dtype: Float64

In [ ]:
```
# Selecting a subset of numeric columns for readability in pair plot
pairplot_cols = [
    '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'
]

# Pair Plot
sns.pairplot(analysis_df[pairplot_cols], corner=True, plot_kws={'alpha': 0.5, 's': 10})
plt.suptitle('Pair Plot of Selected Numerical Features', y=1.02)
plt.show()
```



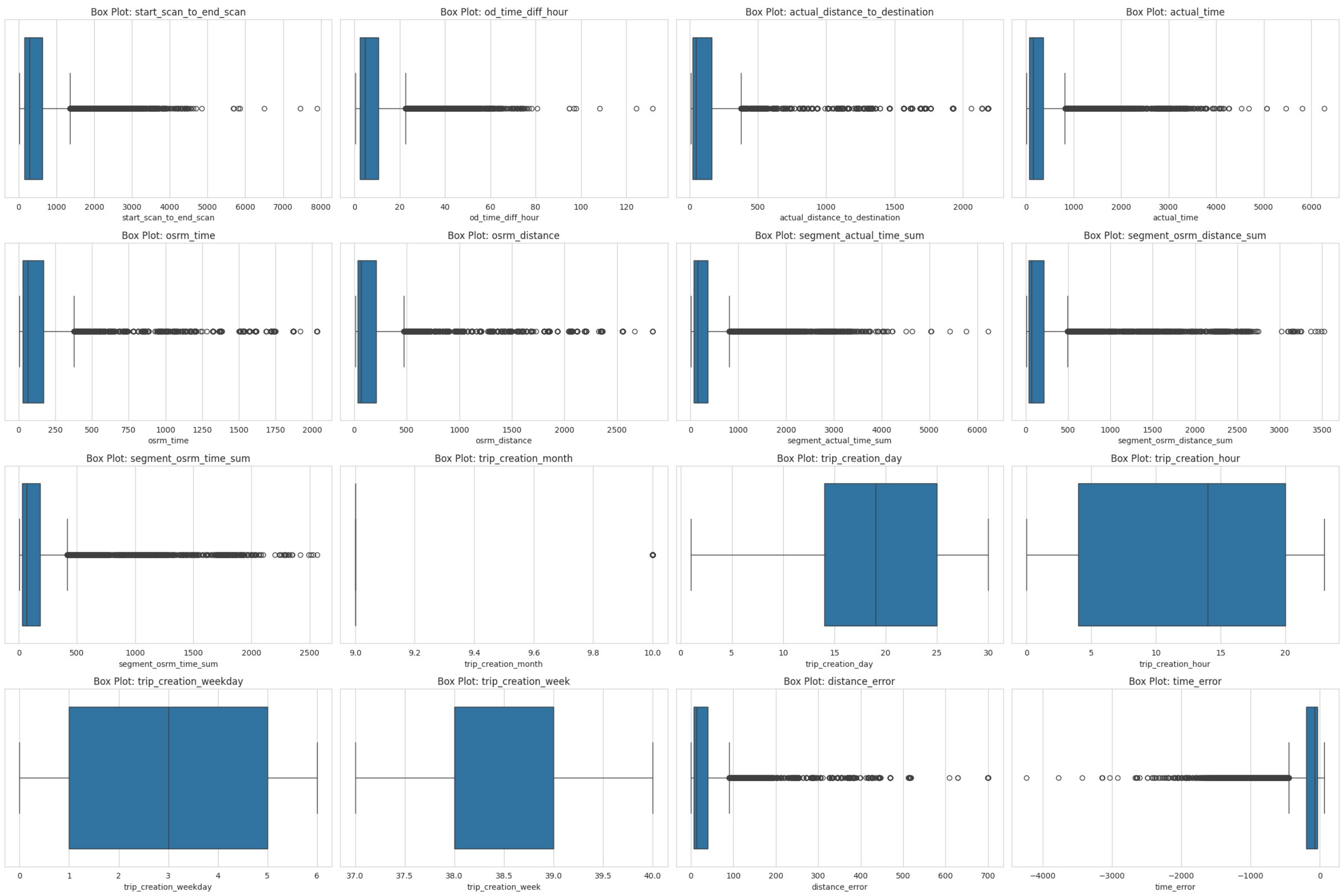
In [ ]:

```python
# Compute correlation matrix
corr = analysis_df[pairplot_cols].corr()

# Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm', center=0, square=True,
            linewidths=0.5, cbar_kws={'shrink': 0.8})
plt.title('Correlation Heatmap of Numerical Features')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [ ]:

```
# Select only numeric columns
numeric_cols = analysis_df.select_dtypes(include=['int', 'float', 'int32', 'float64', 'UInt32']).columns

# Set up subplots
n_cols = 4
n_rows = -(-len(numeric_cols) // n_cols)  # Ceiling division

plt.figure(figsize=(n_cols * 6, n_rows * 4))

for idx, col in enumerate(numeric_cols, 1):
    plt.subplot(n_rows, n_cols, idx)
    sns.boxplot(x=analysis_df[col])
    plt.title(f'Box Plot: {col}')
    plt.tight_layout()

plt.show()
```



Treating outliers using IQR method

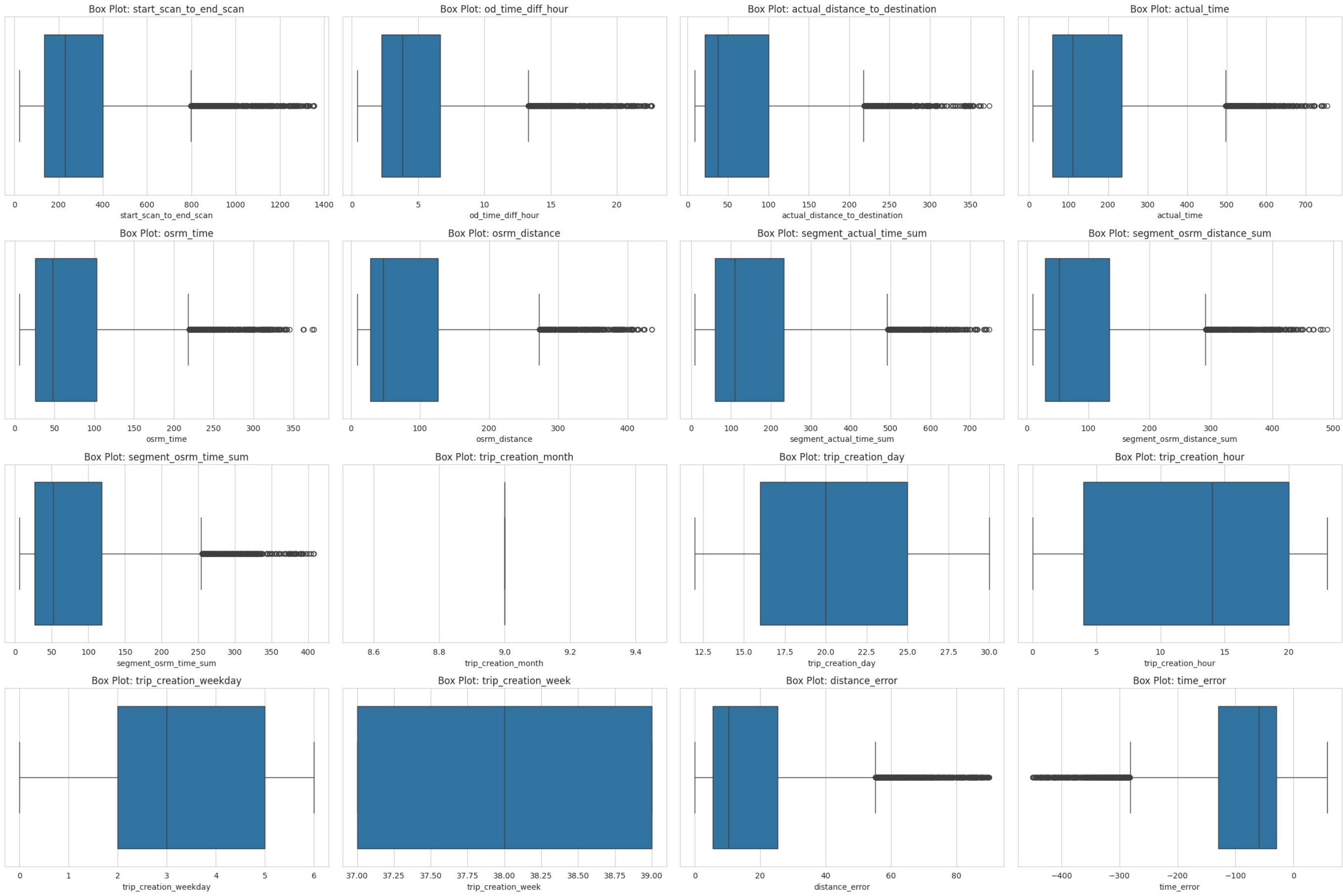
In [ ]:

```python
# Select numerical columns
num_cols = analysis_df.select_dtypes(include=['int64', 'float64', 'int32', 'float32', 'UInt32']).columns.tolist

# Calculate Q1 and Q3
Q1 = analysis_df[num_cols].quantile(0.25)
Q3 = analysis_df[num_cols].quantile(0.75)

# Calculate IQR
IQR = Q3 - Q1

# Filter rows where ALL numerical columns are within the IQR bounds
condition = ~((analysis_df[num_cols] < (Q1 - 1.5 * IQR)) | (analysis_df[num_cols] > (Q3 + 1.5 * IQR))).any(axis=

# Apply filter
analysis_df_clean = analysis_df[condition].reset_index(drop=True)

print(f"Original rows: {len(analysis_df)}")
print(f"Rows after removing outliers: {len(analysis_df_clean)}")
```

```
Original rows: 14787
Rows after removing outliers: 10835
```

In [ ]:

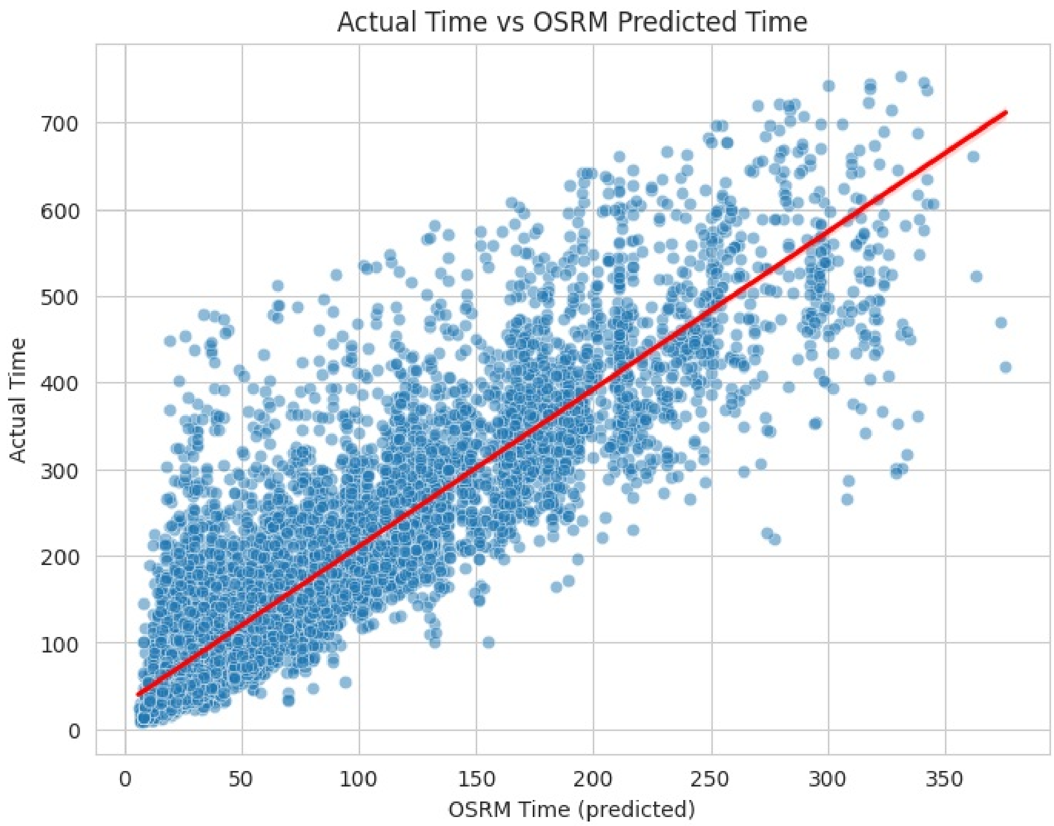
```python
# Select only numeric columns
numeric_cols = analysis_df_clean.select_dtypes(include=['int', 'float', 'int32', 'float64', 'UInt32']).columns

# Set up subplots
n_cols = 4
n_rows = -(-len(numeric_cols) // n_cols)  # Ceiling division

plt.figure(figsize=(n_cols * 6, n_rows * 4))

for idx, col in enumerate(numeric_cols, 1):
    plt.subplot(n_rows, n_cols, idx)
    sns.boxplot(x=analysis_df_clean[col])
    plt.title(f'Box Plot: {col}')
    plt.tight_layout()

plt.show()
```



Performing Hypothesis testing

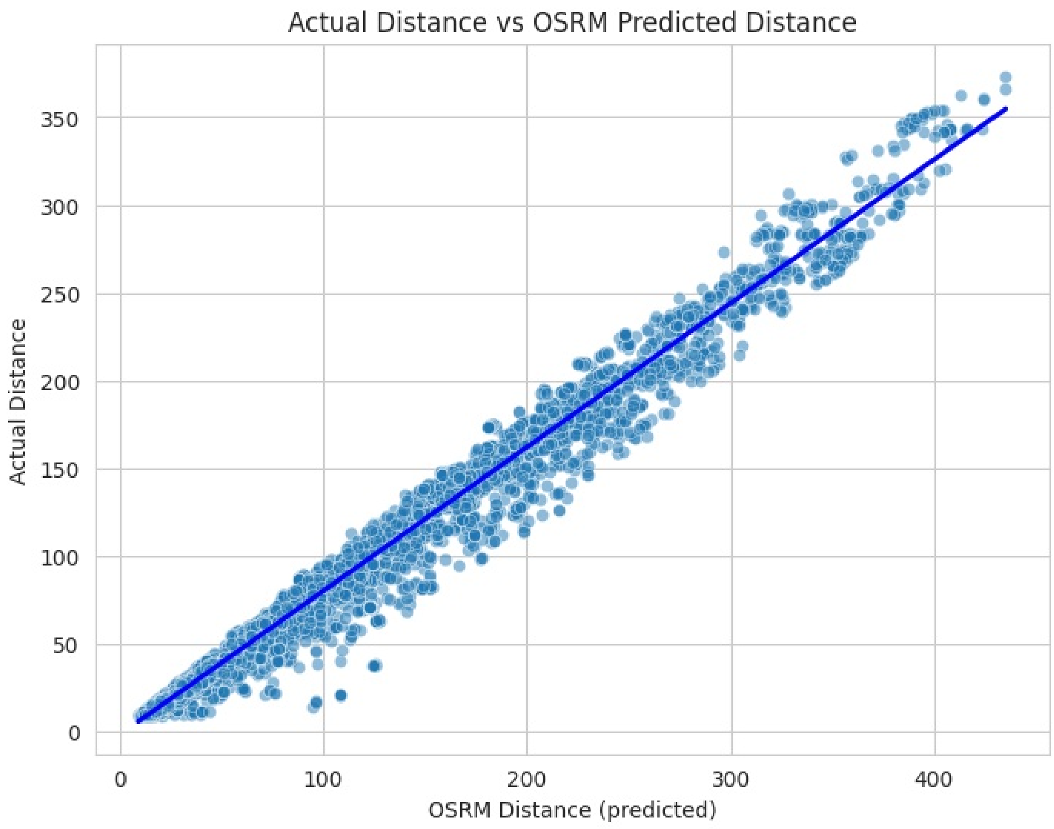
In [ ]:
```python
# Correlation matrix for times
corr_matrix = analysis_df_clean[['actual_time', 'osrm_time']].corr()
print("Correlation matrix (time variables):\n", corr_matrix)

# Paired t-test between actual_time and osrm_time
t_stat, p_value = stats.ttest_rel(analysis_df_clean['actual_time'], analysis_df_clean['osrm_time'])
print(f"Paired t-test result: t-statistic = {t_stat:.3f}, p-value = {p_value:.3e}")

if p_value < 0.05:
    print("Reject null hypothesis: Significant difference between actual and OSRM time.")
else:
    print("Fail to reject null hypothesis: No significant difference between actual and OSRM time.")

# Scatter plot with regression line: Actual vs OSRM time
plt.figure(figsize=(8,6))
sns.scatterplot(x='osrm_time', y='actual_time', data=analysis_df_clean, alpha=0.5)
sns.regplot(x='osrm_time', y='actual_time', data=analysis_df_clean, scatter=False, color='red')
plt.title('Actual Time vs OSRM Predicted Time')
plt.xlabel('OSRM Time (predicted)')
plt.ylabel('Actual Time')
plt.show()
```

```
Correlation matrix (time variables):
              actual_time  osrm_time
actual_time     1.000000   0.892343
osrm_time       0.892343   1.000000
Paired t-test result: t-statistic = 111.262, p-value = 0.000e+00
Reject null hypothesis: Significant difference between actual and OSRM time.
```



In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Correlation between actual_distance_to_destination and osrm_distance
corr = analysis_df_clean[['actual_distance_to_destination', 'osrm_distance']].corr()
print("Correlation matrix (actual distance vs OSRM distance):\n", corr)

# Paired t-test between actual_distance_to_destination and osrm_distance
t_stat, p_value = stats.ttest_rel(analysis_df_clean['actual_distance_to_destination'], analysis_df_clean['osrm_

print(f"Paired t-test result: t-statistic = {t_stat:.3f}, p-value = {p_value:.3e}")

if p_value < 0.05:
    print("Reject null hypothesis: Significant difference between actual and OSRM distance.")
else:
    print("Fail to reject null hypothesis: No significant difference between actual and OSRM distance.")

# Scatter plot with regression line: Actual distance vs OSRM distance
plt.figure(figsize=(8,6))
sns.scatterplot(x='osrm_distance', y='actual_distance_to_destination', data=analysis_df_clean, alpha=0.5)
sns.regplot(x='osrm_distance', y='actual_distance_to_destination', data=analysis_df_clean, scatter=False, color=
plt.title('Actual Distance vs OSRM Predicted Distance')
plt.xlabel('OSRM Distance (predicted)')
plt.ylabel('Actual Distance')
plt.show()
```

```
Correlation matrix (actual distance vs OSRM distance):
                                 actual_distance_to_destination  osrm_distance
actual_distance_to_destination                        1.000000       0.991735
osrm_distance                                         0.991735       1.000000
Paired t-test result: t-statistic = -103.422, p-value = 0.000e+00
Reject null hypothesis: Significant difference between actual and OSRM distance.
```



Calculating if the OSRM time Overshoots or undershoots

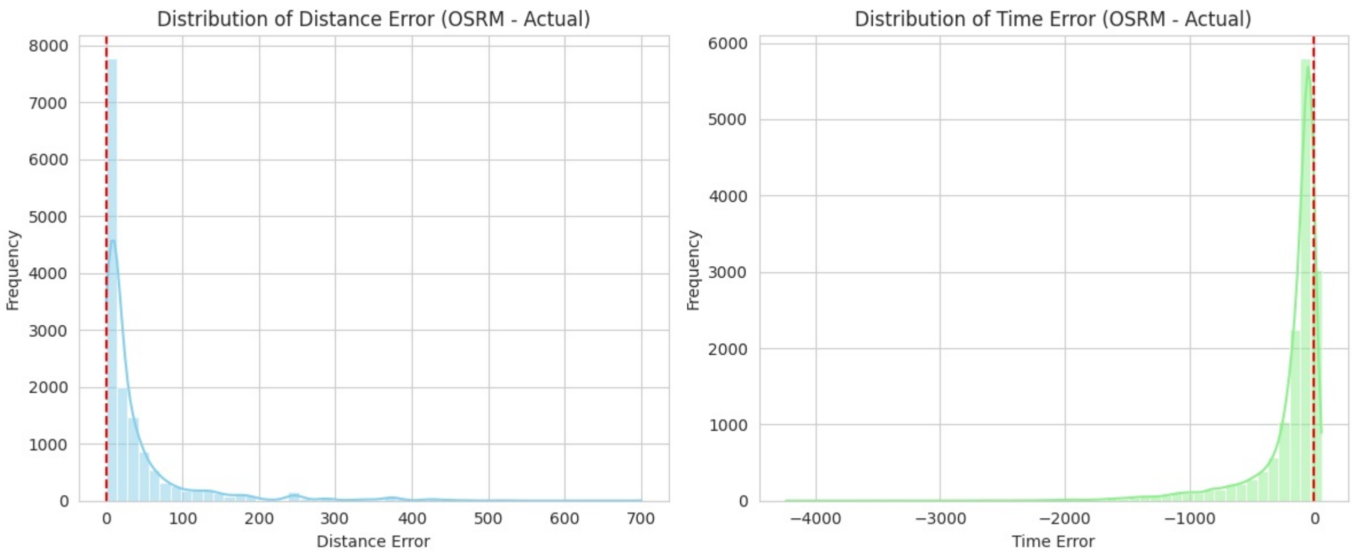
In [ ]:
```
analysis_df_clean['distance_error'] = analysis_df_clean['osrm_distance'] - analysis_df_clean['actual_distance_t
analysis_df_clean['time_error'] = analysis_df_clean['osrm_time'] - analysis_df_clean['actual_time']
```

In [ ]:
```
print("Distance error summary:")
print(analysis_df_clean['distance_error'].describe())

print("\nTime error summary:")
print(analysis_df_clean['time_error'].describe())
```

```
Distance error summary:
count    10835.000000
mean        17.661221
std         17.775556
min          0.014102
25%          5.467025
50%         10.378981
75%         25.333563
max         90.059368
Name: distance_error, dtype: float64

Time error summary:
count    10835.000000
mean       -90.313982
std         84.493398
min       -450.000000
25%       -130.000000
50%        -60.000000
75%        -29.000000
max         58.000000
Name: time_error, dtype: float64
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
sns.histplot(analysis_df_clean['distance_error'], bins=50, kde=True, color='skyblue')
plt.axvline(0, color='red', linestyle='--')
plt.title('Distance Error (OSRM - Actual)')
plt.xlabel('Distance Error')
```

```
plt.ylabel('Frequency')

plt.subplot(1,2,2)
sns.histplot(analysis_df_clean['time_error'], bins=50, kde=True, color='lightgreen')
plt.axvline(0, color='red', linestyle='--')
plt.title('Distribution of Time Error (OSRM - Actual)')
plt.xlabel('Time Error')
plt.ylabel('Frequency')

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



In [ ]:
```
dist_over = (analysis_df_clean['distance_error'] > 0).mean() * 100
dist_under = (analysis_df_clean['distance_error'] < 0).mean() * 100

time_over = (analysis_df_clean['time_error'] > 0).mean() * 100
time_under = (analysis_df_clean['time_error'] < 0).mean() * 100

print(f"Distance overestimated in {dist_over:.2f}% of trips")
print(f"Distance underestimated in {dist_under:.2f}% of trips\n")
print(f"Time overestimated in {time_over:.2f}% of trips")
print(f"Time underestimated in {time_under:.2f}% of trips")
```

```
Distance overestimated in 100.00% of trips
Distance underestimated in 0.00% of trips

Time overestimated in 0.60% of trips
Time underestimated in 99.25% of trips
```

Encoding the data for Feeding into a model

In [ ]:
```
encoded_df = analysis_df_clean.copy()
```

One hot encoding

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

In [ ]:
```
encoded_df
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name
0	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	1	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)
1	training	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	1	trip-153671046011330457	IND400072AAB	Mumbai Hub (Maharashtra)
2	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	0	trip-153671052974046625	IND583101AAA	Bellary_Dc (Karnataka)
3	training	2018-09-12 00:02:34.161600	thanos::sroute:9bf03170-d0a2-4a3f-aa4d-9aaab3d...	1	trip-153671055416136166	IND600056AAA	Chennai_Poonamallee (Tamil Nadu)
4	training	2018-09-12 00:04:22.011653	thanos::sroute:a97698cc-846e-41a7-916b-88b1741...	1	trip-153671066201138152	IND600044AAD	Chennai_Chrompet_DPC (Tamil Nadu)
...	...	...	...	...	...	...	...
10830	test	2018-09-30 23:57:07.576194	thanos::sroute:9bf03170-d0a2-4a3f-aa4d-9aaab3d...	1	trip-153835182757593609	IND600056AAA	Chennai_Poonamallee (Tamil Nadu)
10831	test	2018-09-30 23:57:09.911681	thanos::sroute:e7281daf-3cdf-4dc6-be95-95266c9...	0	trip-153835182991141457	IND384170AAA	Unjha_DC (Gujarat)
10832	test	2018-09-30 23:57:50.622170	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	1	trip-153835187062195567	IND400072AAB	Mumbai Hub (Maharashtra)
10833	test	2018-09-30 23:58:20.971972	thanos::sroute:e6763bf8-f3bc-4029-a67b-6794265...	1	trip-153835190097172173	IND732103AAB	Malda_Central_I_3 (West Bengal)
10834	test	2018-09-30 23:59:45.155123	thanos::sroute:27463ea7-5903-4530-92e7-6a4feca...	1	trip-153835198515486693	IND600116AAB	Chennai_Porur_DPC (Tamil Nadu)

10835 rows × 37 columns

Standardisation and Normalisation

In [ ]:

```
# Define numerical columns to scale (example list — replace with your actual numerical cols)
num_cols = [
    'start_scan_to_end_scan',
    'od_time_diff_hour',
    'actual_distance_to_destination',
    'actual_time',
    'osrm_time',
    'osrm_distance',
    'segment_actual_time_sum',
    'segment_osrm_distance_sum',
    'segment_osrm_time_sum'
]

scaler = StandardScaler()
scaler.fit(encoded_df[num_cols])

encoded_df[num_cols] = scaler.transform(encoded_df[num_cols])

# View the scaled numerical columns
print(encoded_df[num_cols].head())
```

```
   start_scan_to_end_scan  od_time_diff_hour  actual_distance_to_destination  \
0               -0.522707          -0.518764                        0.047610   
1               -0.854876          -0.855131                       -0.751943   
2                1.706972           1.706918                        0.822181   
3               -0.485338          -0.481958                       -0.645999   
4               -0.863180          -0.865453                       -0.867208   

   actual_time  osrm_time  osrm_distance  segment_actual_time_sum  \
0    -0.161202  -0.105370      -0.028321                -0.165076   
1    -0.761308  -0.875619      -0.799827                -0.755574   
2     1.253332   0.606747       0.698966                 1.267961   
3    -0.747020  -0.759355      -0.700962                -0.748373   
4    -1.011352  -0.904685      -0.890166                -1.007616   

   segment_osrm_distance_sum  segment_osrm_time_sum  
0                  -0.104496              -0.231235  
1                  -0.820380              -0.878067  
2                   0.592349               0.428798  
3                  -0.729236              -0.785663  
4                  -0.907852              -0.917669  
```

In [ ]:
```
encoded_df.describe()
```

Out[ ]:

	trip_creation_time	route_type	start_scan_to_end_scan	od_time_diff_hour	actual_distance_to_destination	actual_time
count	10835	10835.000000	1.083500e+04	1.083500e+04	1.083500e+04	1.083500e+04
mean	2018-09-21 03:30:13.800050688	0.700231	-1.206644e-16	5.770905e-17	8.000573e-17	-6.557847e-17
min	2018-09-12 00:00:22.886430	0.000000	-1.174588e+00	-1.174564e+00	-8.686079e-01	-1.118513e+00
25%	2018-09-16 08:54:47.897944064	0.000000	-7.095520e-01	-7.109503e-01	-6.934978e-01	-7.541638e-01
50%	2018-09-20 23:51:58.017022976	1.000000	-3.192544e-01	-3.206270e-01	-4.644675e-01	-3.898140e-01
75%	2018-09-25 20:40:12.389384448	1.000000	3.907552e-01	3.896195e-01	4.319087e-01	4.960559e-01
max	2018-09-30 23:59:45.155123	1.000000	4.356014e+00	4.356851e+00	4.333669e+00	4.203850e+00
std	NaN	0.458178	1.000046e+00	1.000046e+00	1.000046e+00	1.000046e+00

Business Insights

Timeframe & Order Patterns

- Data spans 26 days: September 12 to October 8, 2018.
- Around 88% of the trips occurred in October.

Shipment Mode Preference

- Full Truck Load (FTL) is the dominant mode of transportation.
- Suggests operational efficiency through consolidated shipments.
- Carting is less used – potential area to explore for flexible routing.

Geographic & Route Trends

- Busiest source states: Maharashtra, Karnataka.
- Key source cities: Gurgaon, Bangalore, Bhiwandi.
- Common destination cities: Gurgaon, Bangalore, Hyderabad.
- Most frequent corridor: Mumbai (Maharashtra) to Bangalore (Karnataka).

Delivery Performance Insights

- Actual delivery times tend to be longer than OSRM-estimated times.
- Indicates OSRM underestimates delivery durations.
- OSRM-predicted distances are higher than actual distances, which may lead to inflated planning assumptions.
- Segment-wise time aligns with overall actual time, but OSRM distances at segment level are more conservative.

Business Recommendations

Route Planning and Forecasting

- Calibrate OSRM models with historical data for more realistic ETA predictions.
- Prepare operations for increased mid-month demand.
- Investigate reasons for missing trip data between the 4th and 11th.

Operational Optimization

- Use actual vs. estimated data to fine-tune logistics operations.
- Promote FTL usage where applicable for time and cost efficiency.
- Improve Carting strategies for better load balancing and last-mile flexibility.

Corridor and City Focus

- Focus improvement efforts on high-traffic corridors such as Mumbai to Bangalore.
- Enhance infrastructure and delivery logistics in high-volume cities like Gurgaon and Bangalore.

Customer-Centric Actions

- Align delivery time promises with actual performance to build trust.
- Analyze customers in high-order states (e.g., Maharashtra, Karnataka) to enhance service offerings.

Cost and Resource Efficiency

- Reduce cost overruns by addressing prediction inaccuracies.
- Use segment-level analysis for targeted route optimizations.

Stakeholder Collaboration

- Work with transportation authorities and logistics partners to streamline busy corridors.
- Leverage traffic and demand insights for smarter real-time routing.

CASE STUDY COMPLETE

Author - Shishir Bhat