Overview

- Delhivery is a large and rapidly growing logistics company in India. They are a fully integrated player in the industry and aim to create a comprehensive operating system for commerce using infrastructure, high-quality logistics, and advanced technology.
- Delhivery have a nationwide presence, servicing over 18,700 pin codes with a large network of sort centers, gateways, and delivery centers. This extensive infrastructure allows them to operate 24/7, every day of the year.
- Delhivery offers a range of services including delivery, express mail, third-party logistics, supply chain management, and various types of freight shipping. Delhivery are a public company with significant revenue.

Problem Statement

Delhivery wants to understand and process the data coming out of data engineering pipelines to support the data science team in building accurate forecasting models. This involves:

- **Data Cleaning and Sanitization:** Identify and handle missing values, outliers, and inconsistencies in the raw data.
- **Feature Engineering:** Extract and create relevant features from the raw fields that can improve the performance of forecasting models.
- **Data Exploration and Analysis:** Perform exploratory data analysis to understand data patterns, distributions, and relationships, and to gain insights into the logistics operations.
- **Data Preparation for Modeling:** Prepare the processed and engineered data in a suitable format for input into various forecasting models.

Objective

Support the data science team in building accurate forecasting models by understanding and processing the data coming out of data engineering pipelines.

This objective is achieved by focusing on the key steps outlined in the problem statement:

- Data Cleaning and Sanitization
- Feature Engineering
- Data Exploration and Analysis
- Data Preparation for Modeling

1. Exploratory Data Analysis

Dataset : We will be using this dataset "delhivery_dataset.csv" throughout this casestudy and try to findout useful insights.

```
#Importing required Libaries
import numpy as np #For basic mathematical operations
import pandas as pd #For data analysis
import matplotlib.pyplot as plt #For data visulatisation
import seaborn as sns #For data visulatisation

#Reading the CSV file
df = pd.read_csv("delhivery_dataset.csv")

#Checking few records
df.sample(5)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	
41280	training	2018-09-21 02:45:03.044986	thanos::sroute:65574be6-be55-4fd7-bf27-0c4c3dc...	Carting	trip-15374
106546	training	2018-09-14 01:54:55.120054	thanos::sroute:828f0a4b-da11-4921-88df-5316ddb...	FTL	trip-15368
31116	training	2018-09-25 03:17:24.877866	thanos::sroute:01164881-301e-45f8-bacd-ee21c37...	FTL	trip-15378
27039	training	2018-09-14 17:10:49.922375	thanos::sroute:ea719f23-0d12-478f-a068-368f3f4...	FTL	trip-15369
32221	training	2018-09-14 17:18:24.415312	thanos::sroute:5b2da079-88c8-4676-bffb-09188e8...	FTL	trip-15369

5 rows × 24 columns

- Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), missing value detection, statistical summary.

Shape of the dataset : There are 144867 rows and 24 columns

```
df.shape #shape function in pandas returns the number of rows and columns present i
```

Out[ ]: (144867, 24)

Dimension of the dataset : This dateset is of two-dimensional(2D)

```
df.ndim #data.ndim wil tell us that how many dimension is the dataset of
```

Out[ ]: 2

Data types of all the attributes

In this dataset is_cutoff is of dtype bool and cutoff_factor is of dtype int whereas rest of the attributes are mostly of dtype object(string) or float.

```
df.info() #info function in pandas returns the shape, data types, number of non nul
```

```
<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
```

We can see that columns trip_creation_time, od_start_time, od_end_time, segment_actual_time, actual_time, cutoff_timestamp, osrm_time, segment_osrm_time are of dtype object but in general the dtype of datetime column should of type datetime. Let us convert the dtype of datetime from object to datetime.

```
df1 = df.copy(deep=True) #making a copy of the dataset

df1['trip_creation_time'] = pd.to_datetime(df1['trip_creation_time']) #converting t
df1['od_start_time'] = pd.to_datetime(df1['od_start_time']) #converting the od_star
df1['od_end_time'] = pd.to_datetime(df1['od_end_time']) #converting the od_end_time
df1['segment_actual_time'] = pd.to_datetime(df1['segment_actual_time']) #converting
df1['actual_time'] = pd.to_datetime(df1['actual_time']) #converting the actual_time
df1['cutoff_timestamp'] = pd.to_datetime(df1['cutoff_timestamp'],format="mixed") #c
df1['osrm_time'] = pd.to_datetime(df1['osrm_time']) #converting the osrm_time colum
df1['segment_osrm_time'] = pd.to_datetime(df1['segment_osrm_time']) #converting the

df1.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  datetime64[ns]
 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  datetime64[ns]
 10  od_end_time                      144867 non-null  datetime64[ns]
 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  datetime64[ns]
 15  actual_distance_to_destination   144867 non-null  float64
 16  actual_time                      144867 non-null  datetime64[ns]
 17  osrm_time                        144867 non-null  datetime64[ns]
 18  osrm_distance                    144867 non-null  float64
 19  factor                           144867 non-null  float64
 20  segment_actual_time              144867 non-null  datetime64[ns]
 21  segment_osrm_time                144867 non-null  datetime64[ns]
 22  segment_osrm_distance            144867 non-null  float64
 23  segment_factor                   144867 non-null  float64
dtypes: bool(1), datetime64[ns](8), float64(6), int64(1), object(8)
memory usage: 25.6+ MB
```

Details about the columns:

Column Profiling:

- data - tells whether the data is testing or training data
- trip_creation_time – Timestamp of trip creation
- route_schedule_uuid – Unique Id for a particular route schedule
- route_type – Transportation type
  - FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
  - Carting: Handling system consisting of small vehicles (carts)
- trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
- source_center - Source ID of trip origin
- source_name - Source Name of trip origin
- destination_cente – Destination ID
- destination_name – Destination Name
- od_start_time – Trip start time
- od_end_time – Trip end time
- start_scan_to_end_scan – Time taken to deliver from source to destination
- is_cutoff – Unknown field
- cutoff_factor – Unknown field
- cutoff_timestamp – Unknown field
- actual_distance_to_destination – Distance in Kms between source and destination warehouse
- actual_time – Actual time taken to complete the delivery (Cumulative)
- osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
- osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
- factor – Unknown field
- segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
- segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package

Conversion of categorical attributes to 'category':

Use of Series.str.split function: Split strings around given separator/delimiter. Splits the string in the Series/Index from the beginning, at the specified delimiter string.

We see in the dataset all the column values are fine and there is no need of splitting and exploding them.

Missing value detection:

As a first step we need to detect the missing values using the below 2 functions:

- isnull(): This function returns a pandas dataframe, where each value is a boolean value True if the value is missing else False.
- isna(): This one is similar to isnull and notnull. However, it shows True only when the missing value is NaN type.

Once the null values are detected we can use 2 methods, either drop the entire column of null values are replace the values with:

1. mean value.
2. median value.
3. mode value.

These 3 methods are applicable only for columns which are numerical.

```
df.isna().sum() #checking for total number of null values in each column
```

Out[ ]:

	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

Columns **source_name** and **destination_name** have missing values 293 and 261 respectively.

Statistical summary or Description of the dataset:

The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyses both numeric and object series and also the DataFrame column sets of mixed data types.

```
df.describe() #statistical summary of the dataset
```

Out[ ]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_ti
count	144867.000000	144867.000000	144867.000000	144867.000(
mean	961.262986	232.926567	234.073372	416.927!
std	1037.012769	344.755577	344.990009	598.103(
min	20.000000	9.000000	9.000045	9.000(
25%	161.000000	22.000000	23.355874	51.000(
50%	449.000000	66.000000	66.126571	132.000(
75%	1634.000000	286.000000	286.708875	513.000(
max	7898.000000	1927.000000	1927.447705	4532.000(

Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the categorical variables)

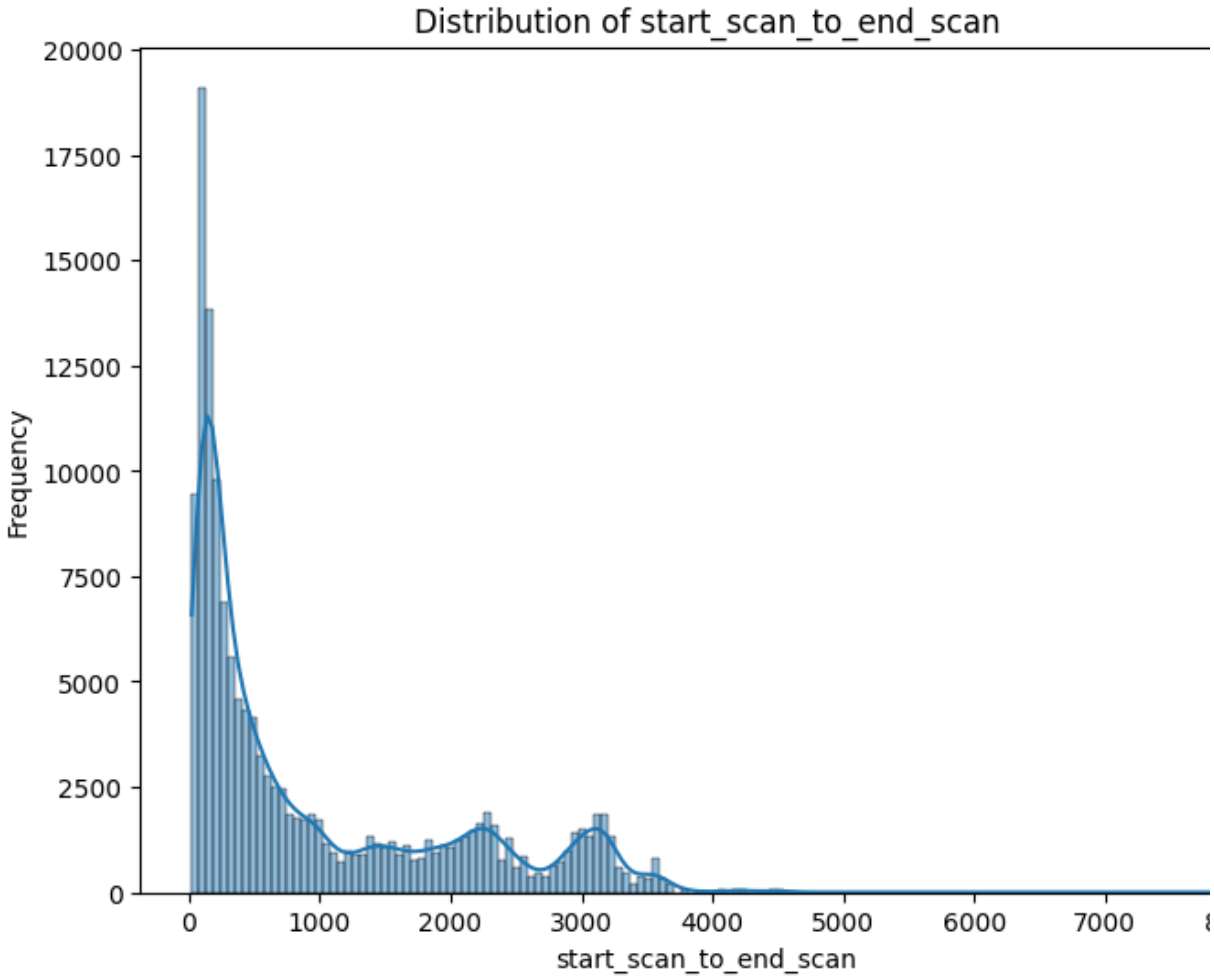
Before we start on visual analysis, data should be in the correct format for us to use it as the input. This preparation of the data by identifying and resolving the potential errors, inaccuracies, and inconsistencies is termed as Data Cleaning.

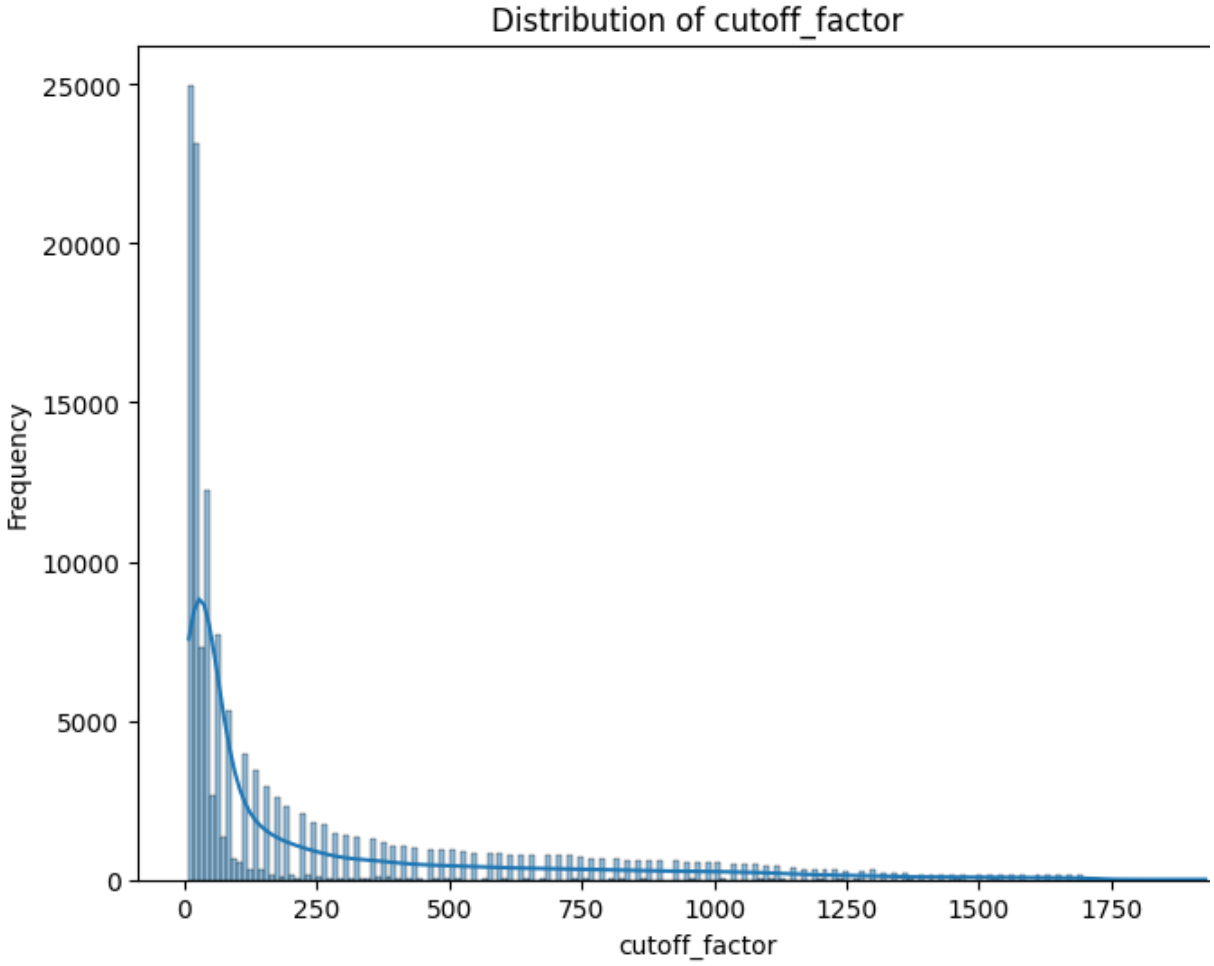
Step1: Identifying and resolving the potential errors:

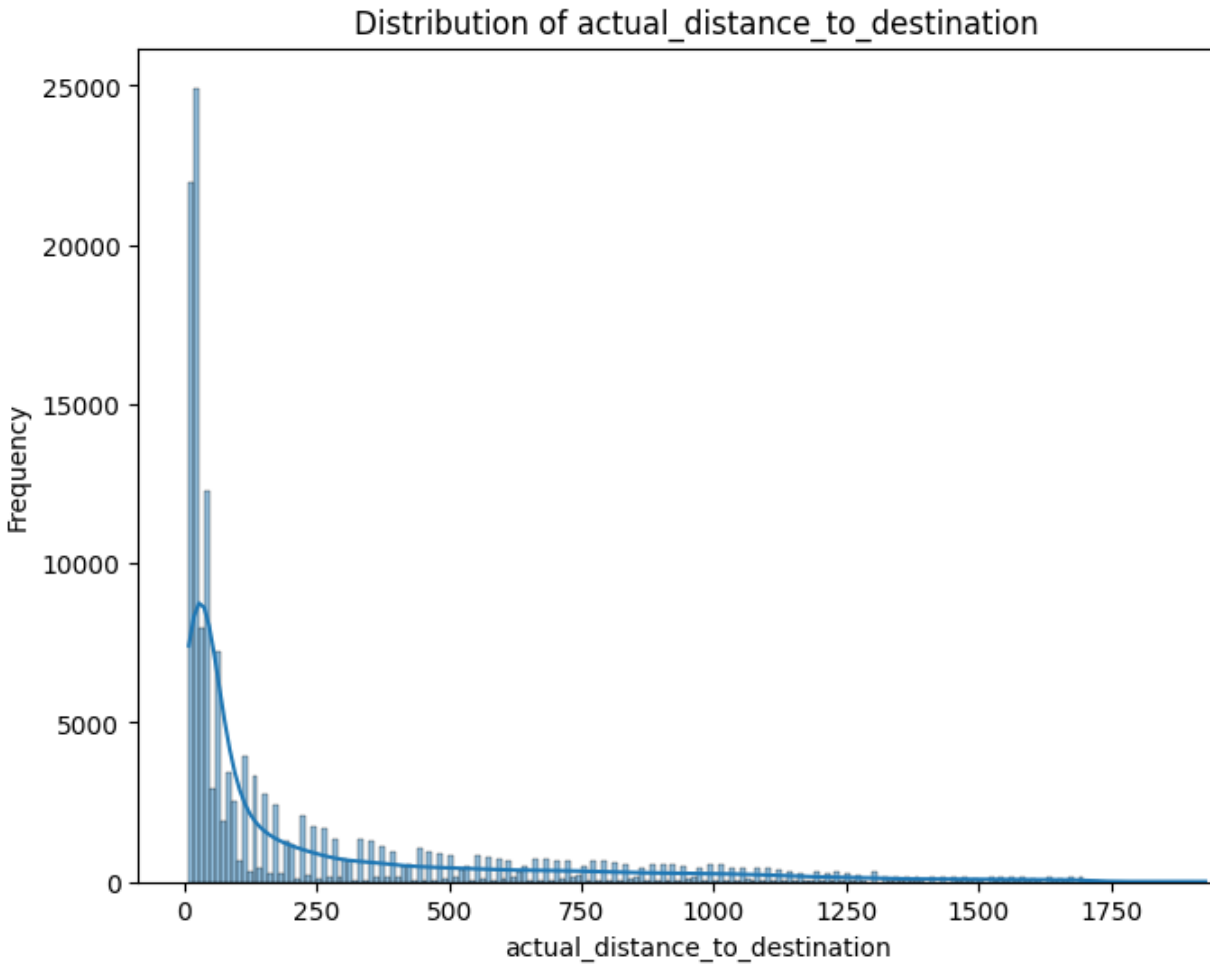
Step2: Identifying inaccuracies:

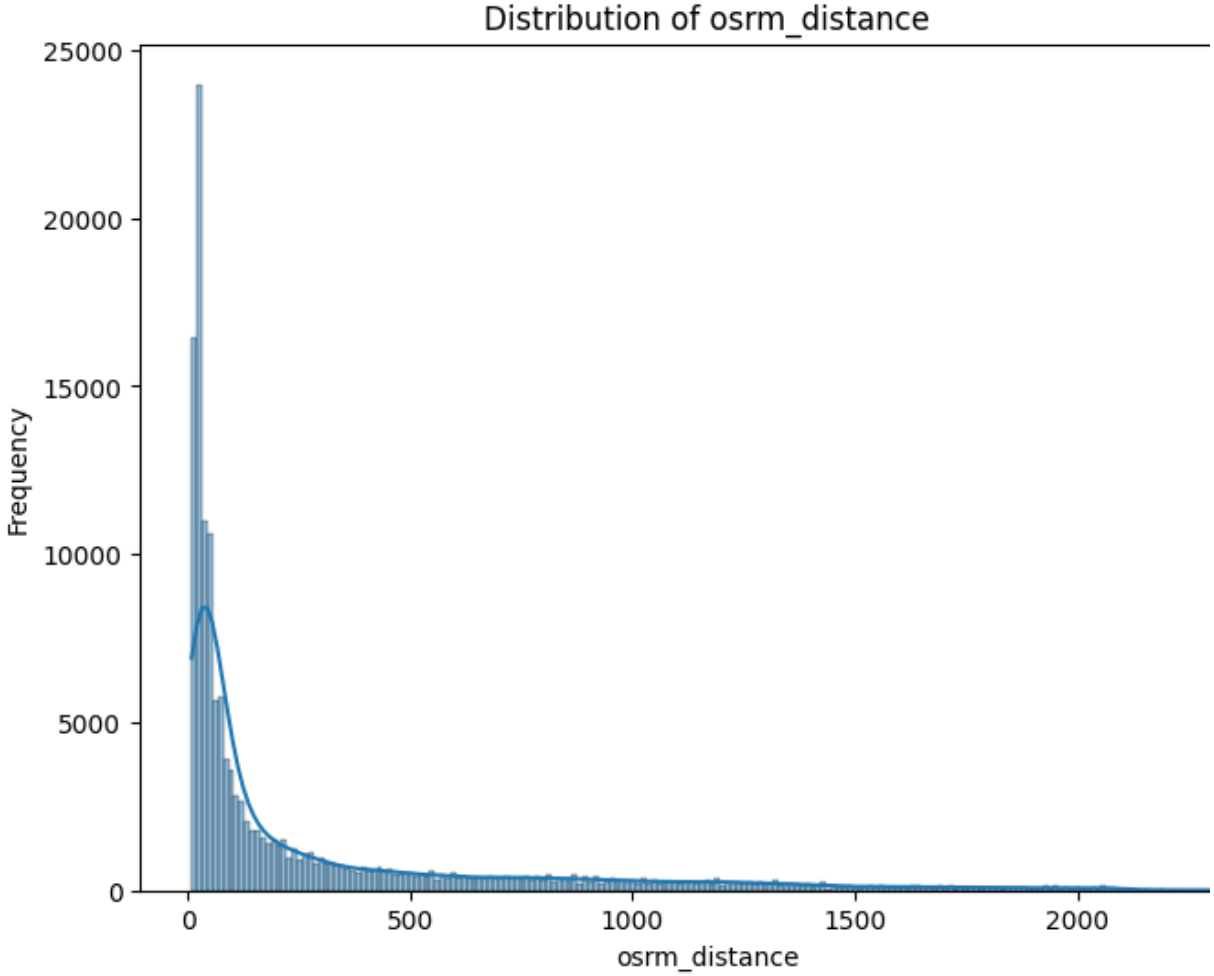
Step3: Identifying inconsistencies:

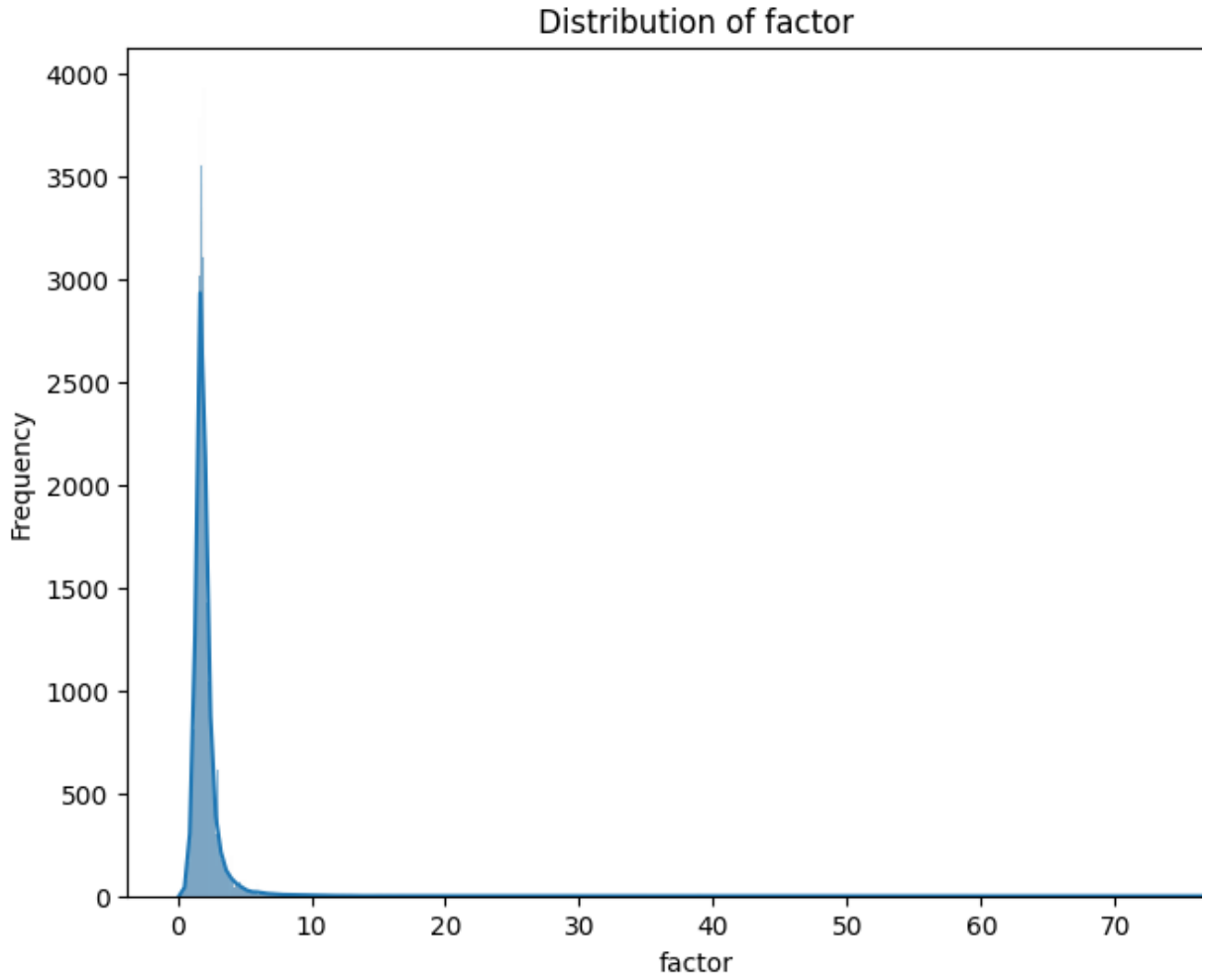
```
continuous_vars = df1.select_dtypes(include=['number']).columns #storing the numeri
categorical_vars = df1.select_dtypes(exclude=['number']).columns #storing the non-n

# Distribution plots for continuous variables
for col in continuous_vars:
 plt.figure(figsize=(8, 6))
 sns.histplot(df[col], kde=True)
 plt.title(f'Distribution of {col}')
 plt.xlabel(col)
 plt.ylabel('Frequency')
 plt.show()
```

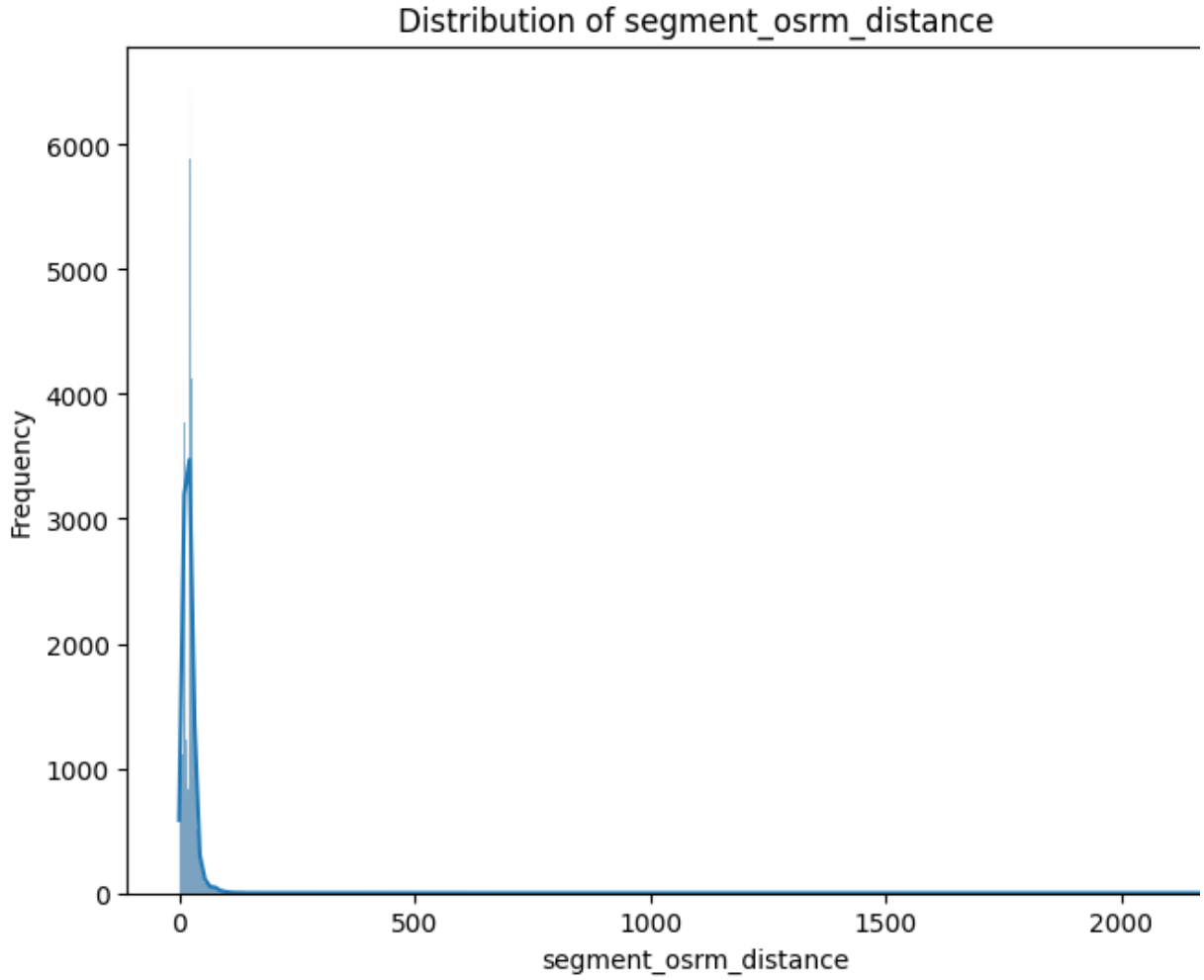

Distribution of start_scan_to_end_scan
Frequency
start_scan_to_end_scan
0
2500
5000
7500
10000
12500
15000
17500
20000
0
1000
2000
3000
4000
5000
6000
7000

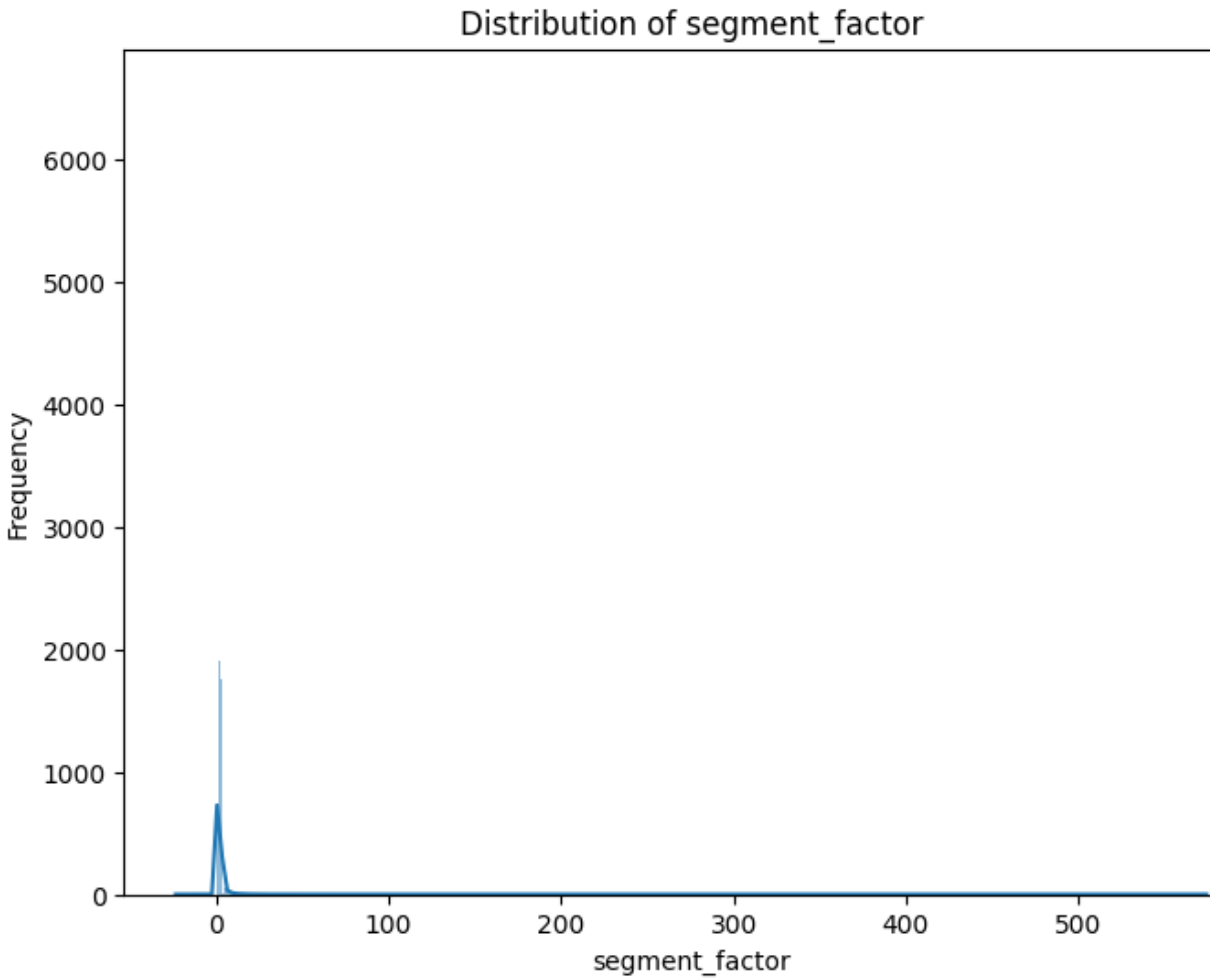

Distribution of cutoff_factor
Frequency
cutoff_factor
0
5000
10000
15000
20000
25000
0
250
500
750
1000
1250
1500
1750


Distribution of actual_distance_to_destination
Frequency
25000
20000
15000
10000
5000
0
0
250
500
750
1000
1250
1500
1750
actual_distance_to_destination

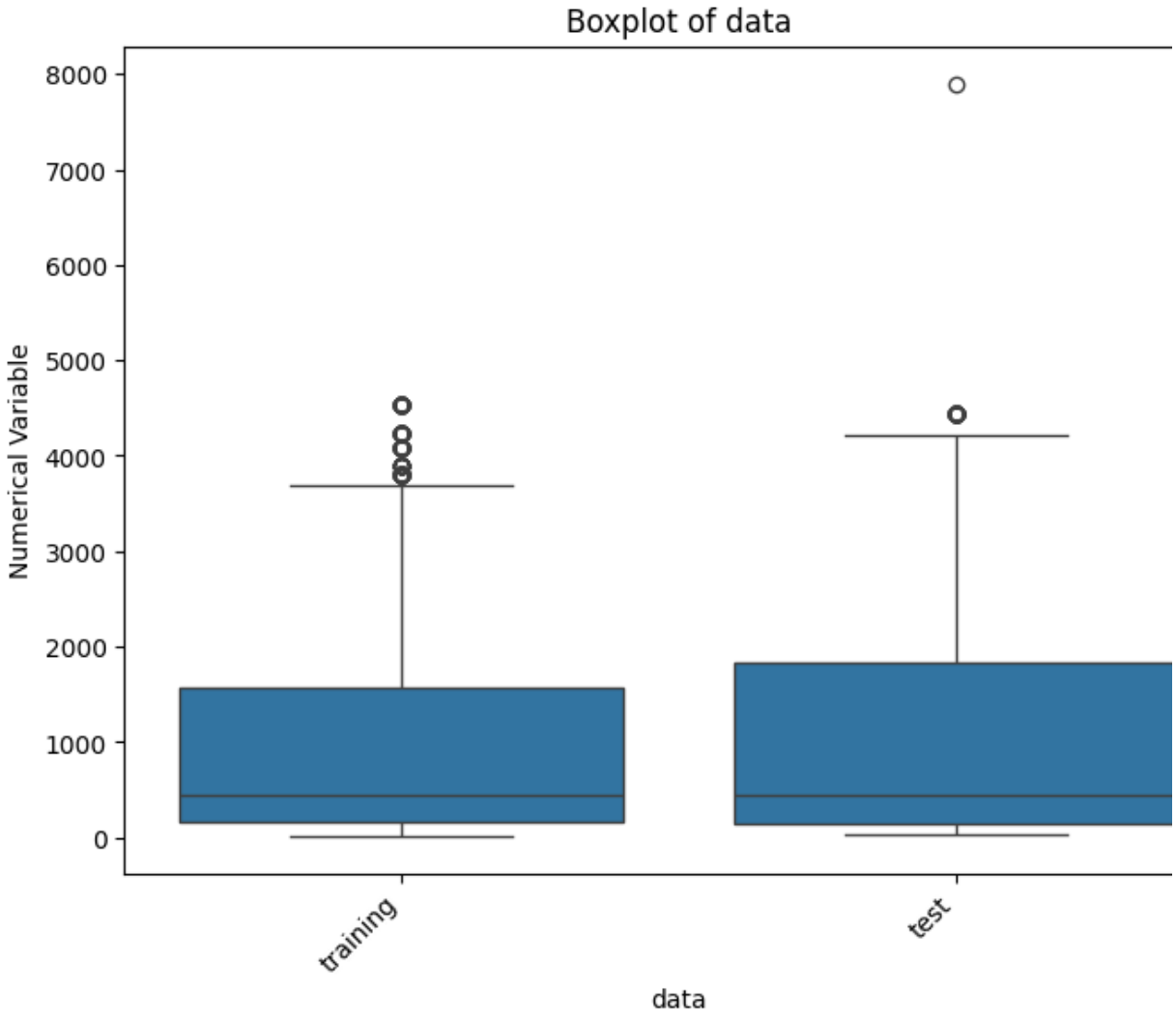

Distribution of osrm_distance
Frequency
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
osrm_distance

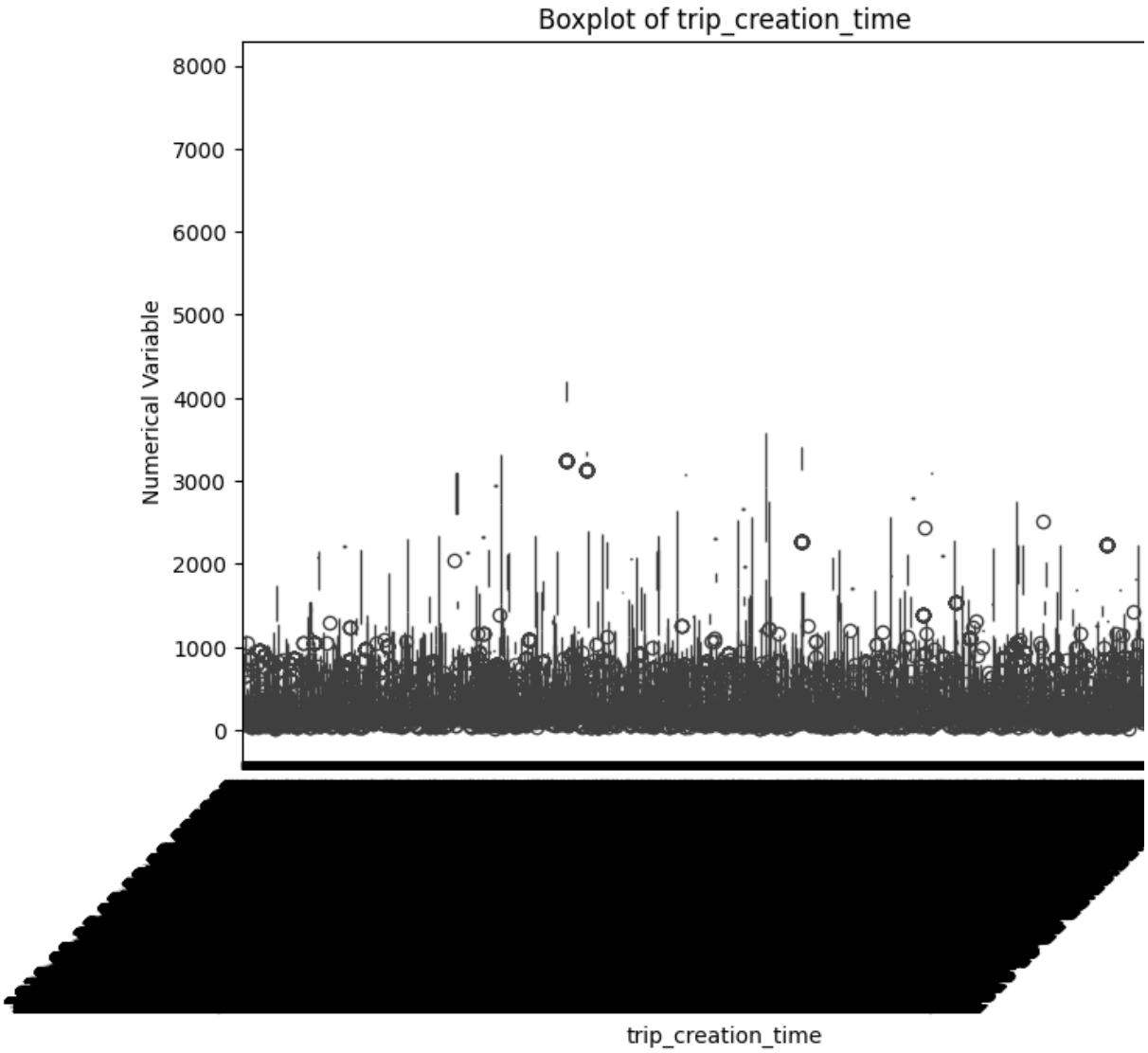

Distribution of factor
Frequency
factor
0
500
1000
1500
2000
2500
3000
3500
4000
0
10
20
30
40
50
60
70

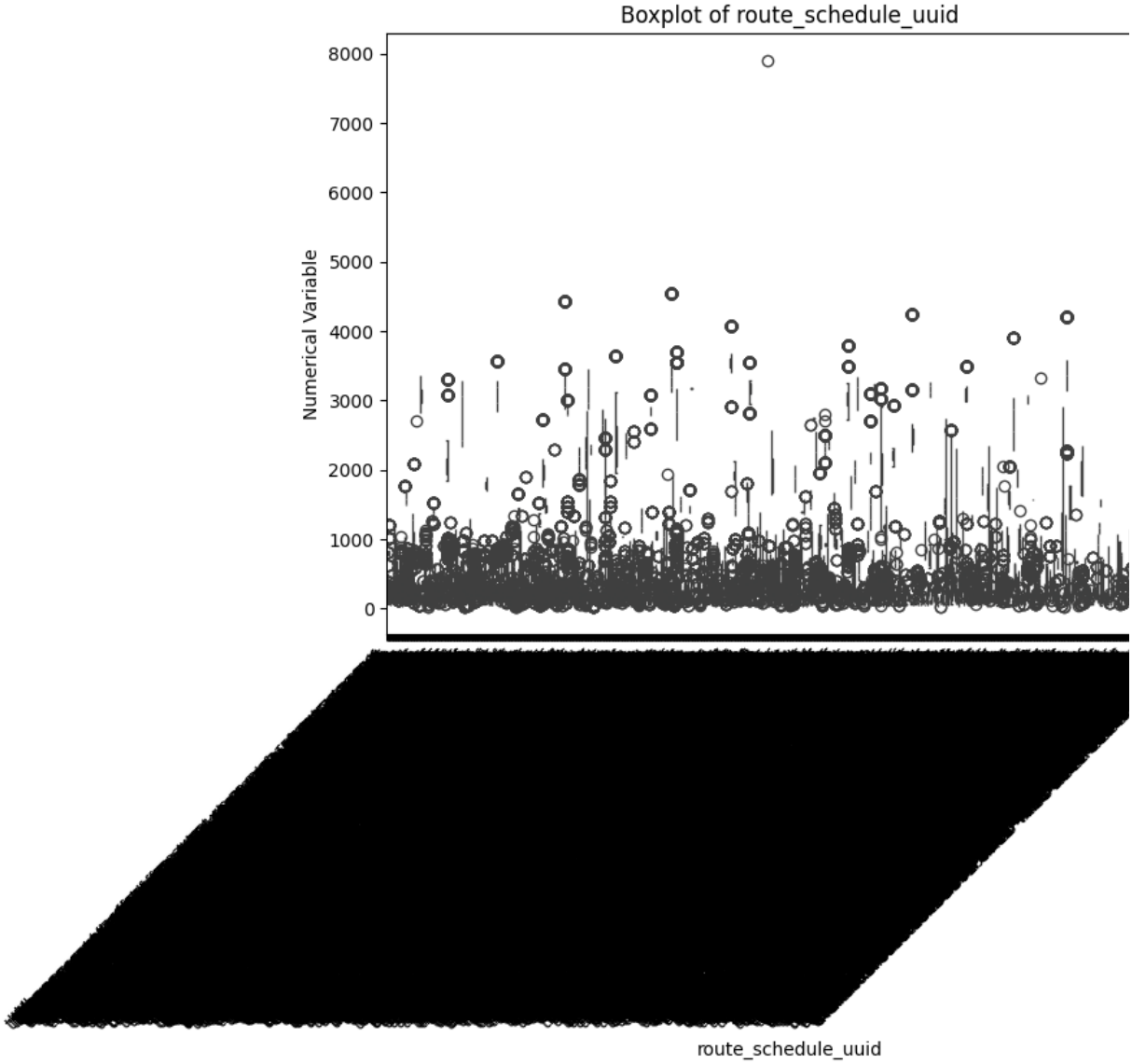

Distribution of segment_osrm_distance
Frequency
segment_osrm_distance
0
1000
2000
3000
4000
5000
6000
0
500
1000
1500
2000

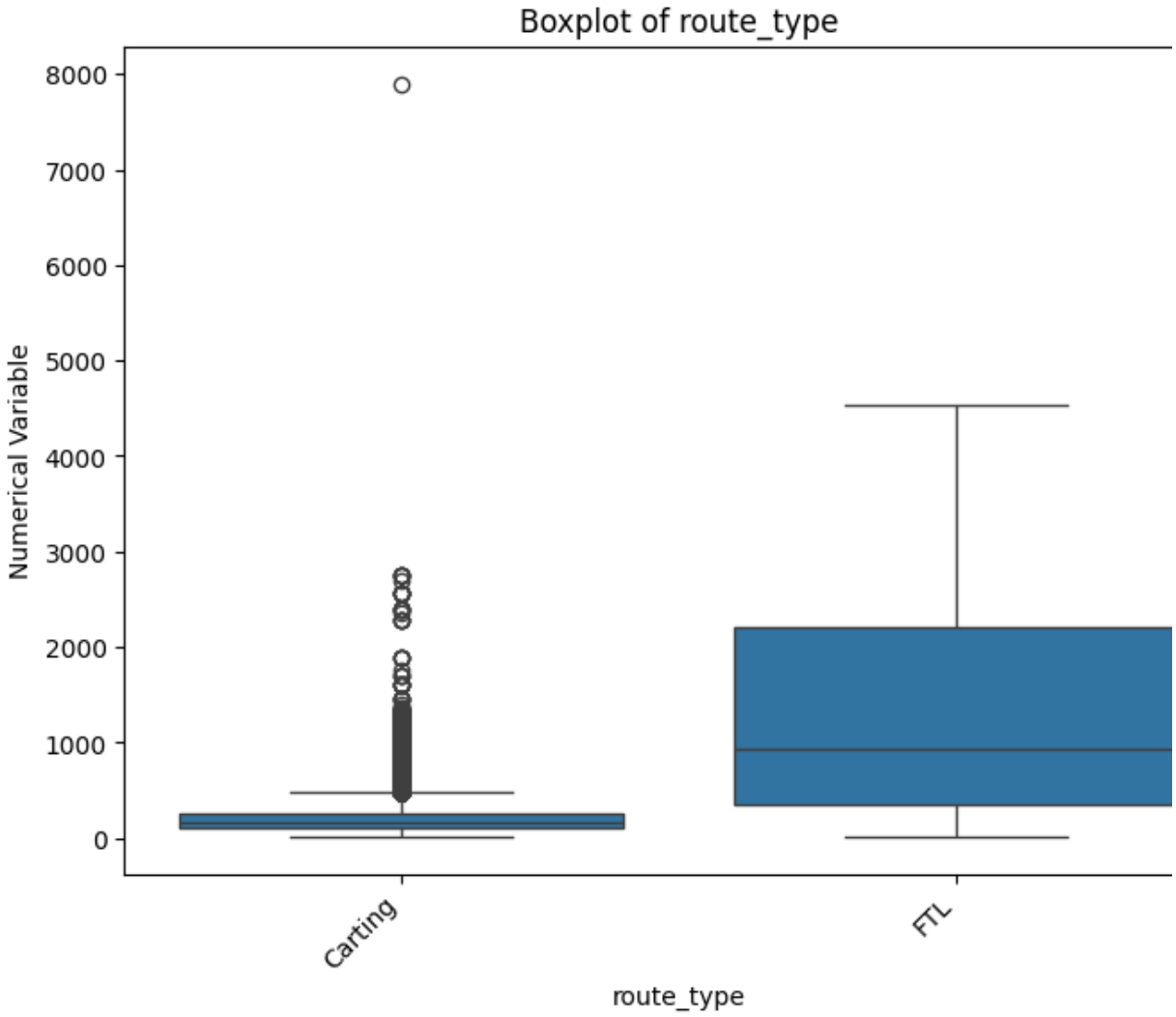


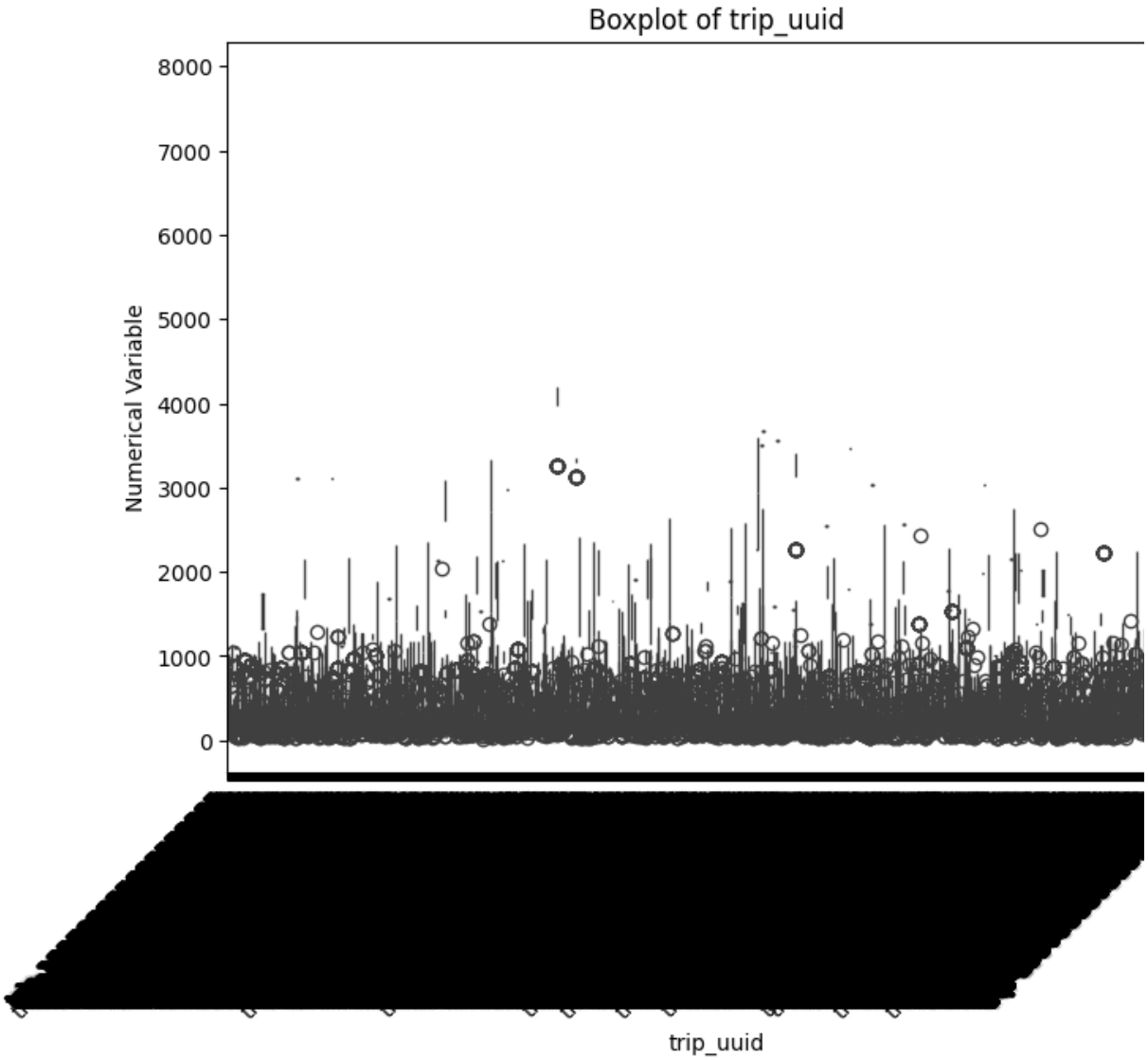

```
# Boxplots for categorical variables
for col in categorical_vars:
 plt.figure(figsize=(8, 6))
 sns.boxplot(x=col, y=df.select_dtypes(include=['number']).iloc[:, 0], data=df) # R
 plt.title(f'Boxplot of {col}')
 plt.xlabel(col)
 plt.ylabel('Numerical Variable') # Update label
 plt.xticks(rotation=45, ha='right') # Rotate x-axis labels if needed
 plt.show()
```

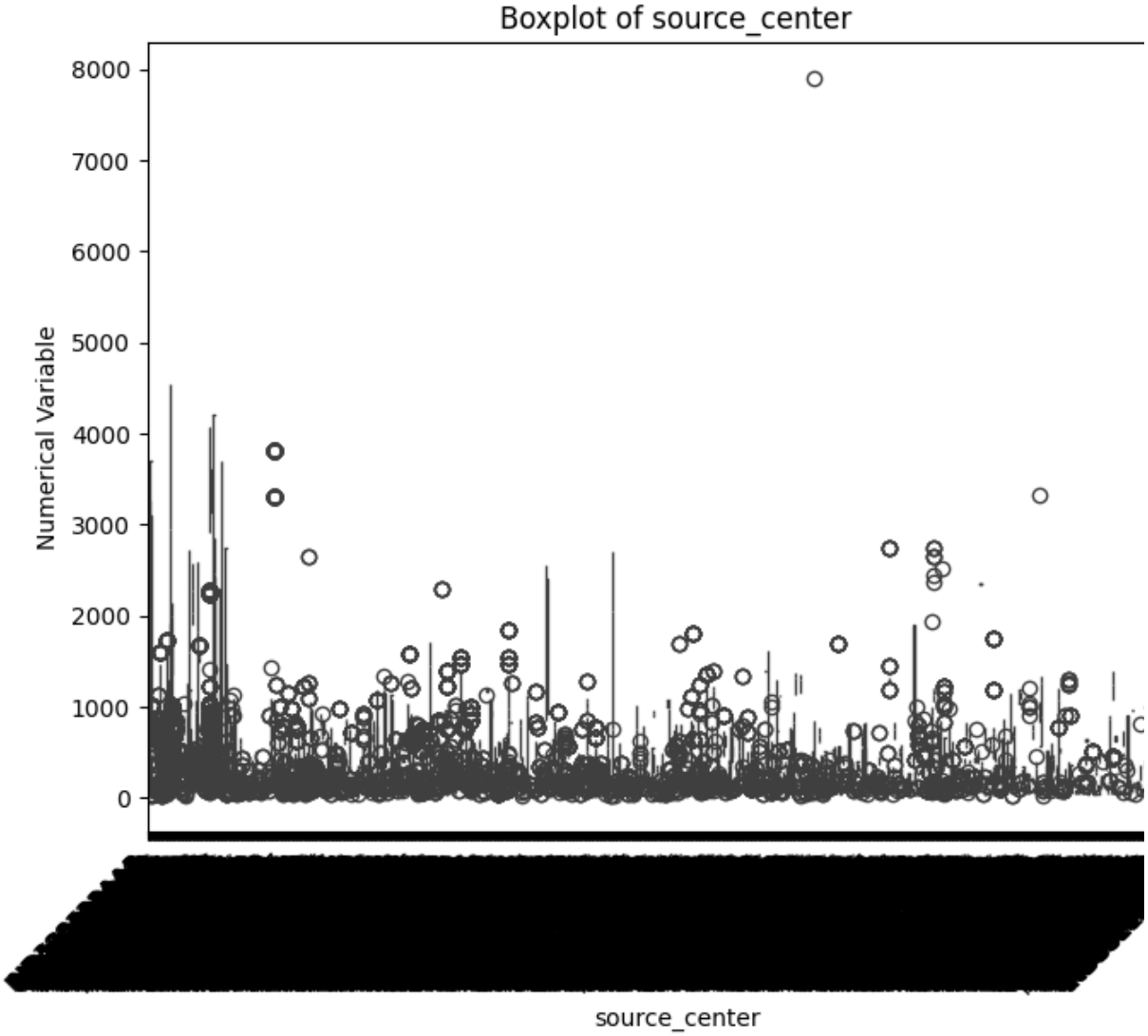

Boxplot of data
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
training
test
data

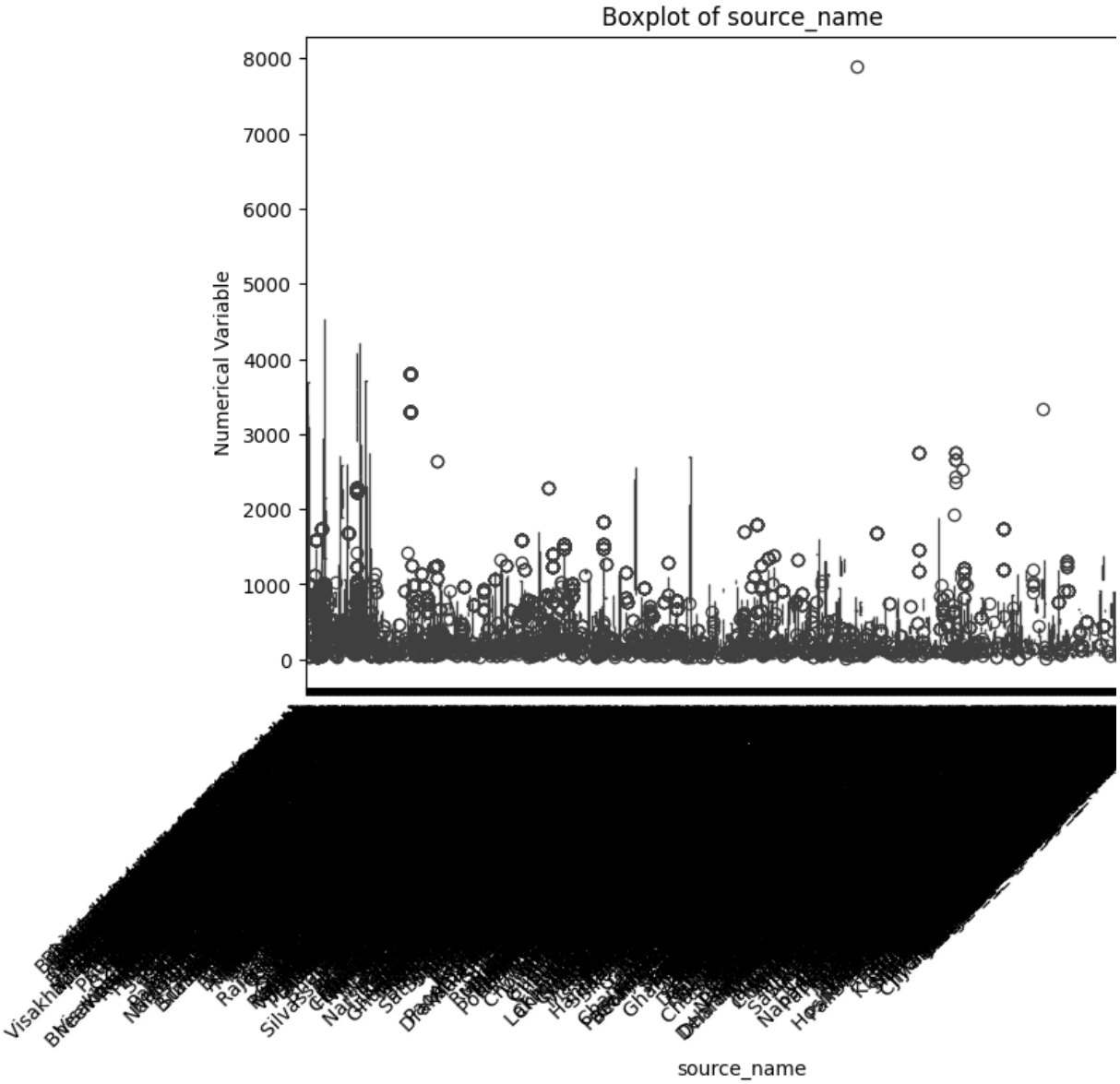

Boxplot of trip_creation_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
trip_creation_time

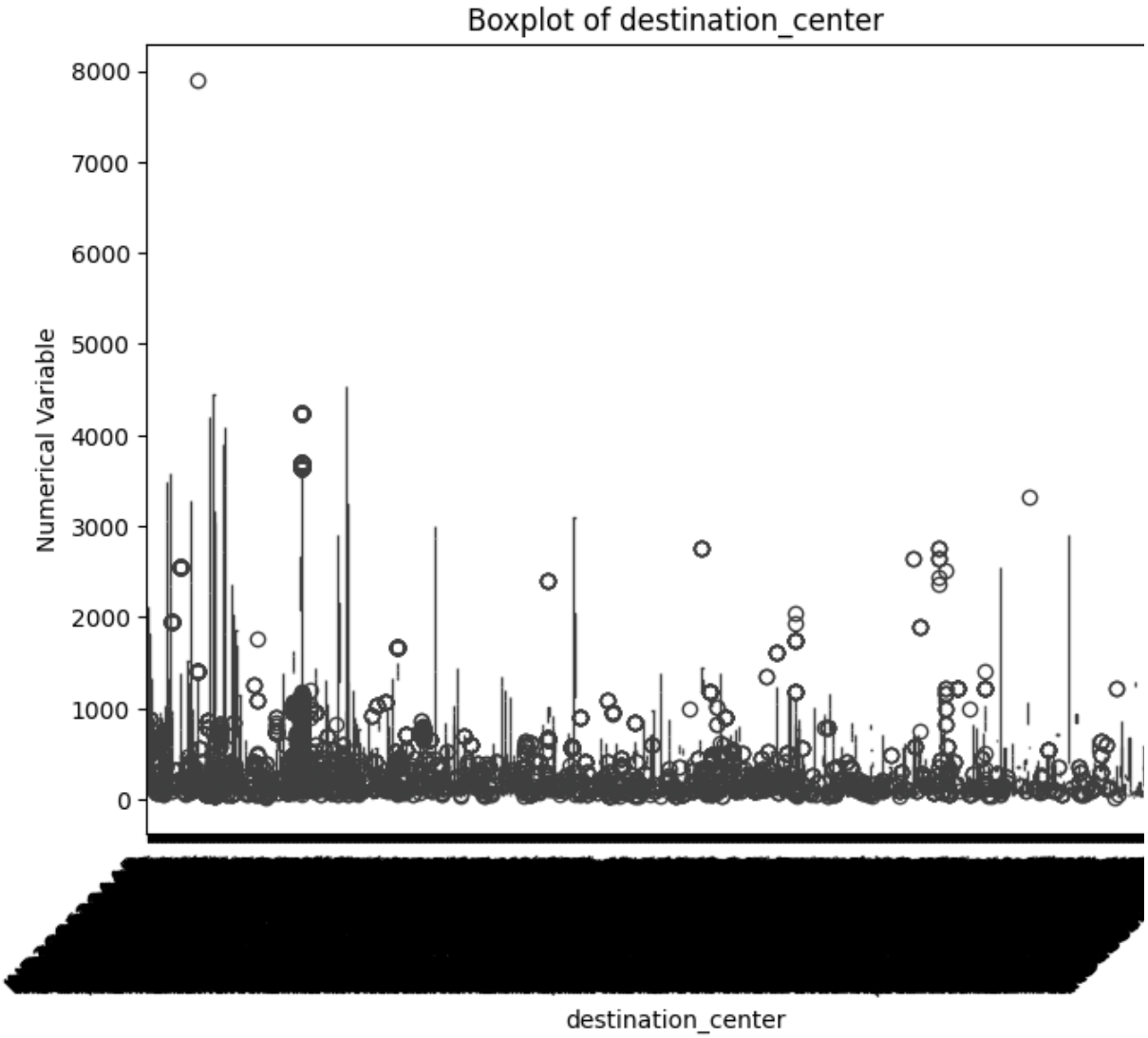

Boxplot of route_schedule_uuid
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
route_schedule_uuid

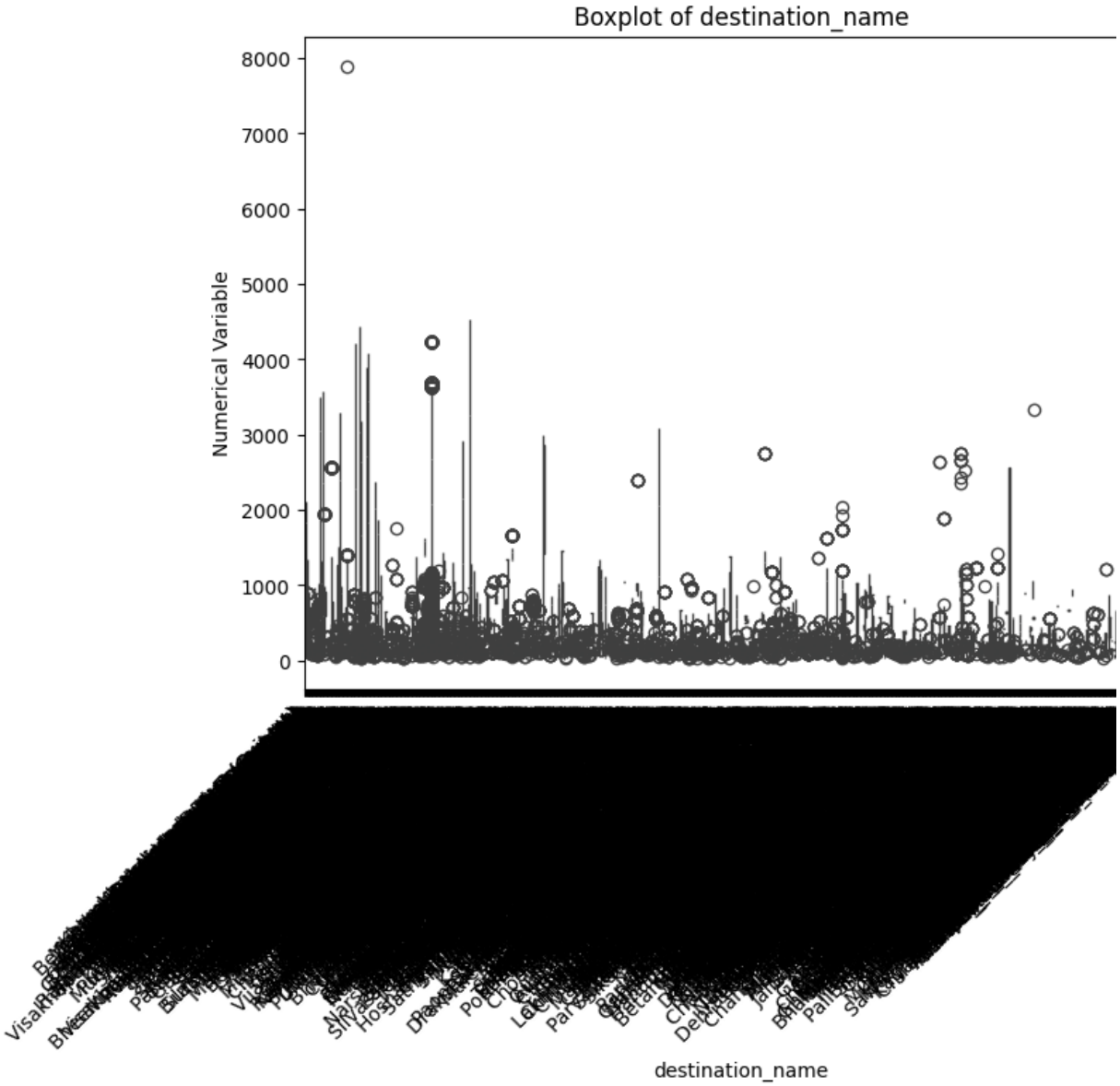

Boxplot of route_type
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
Carting
FTL
route_type

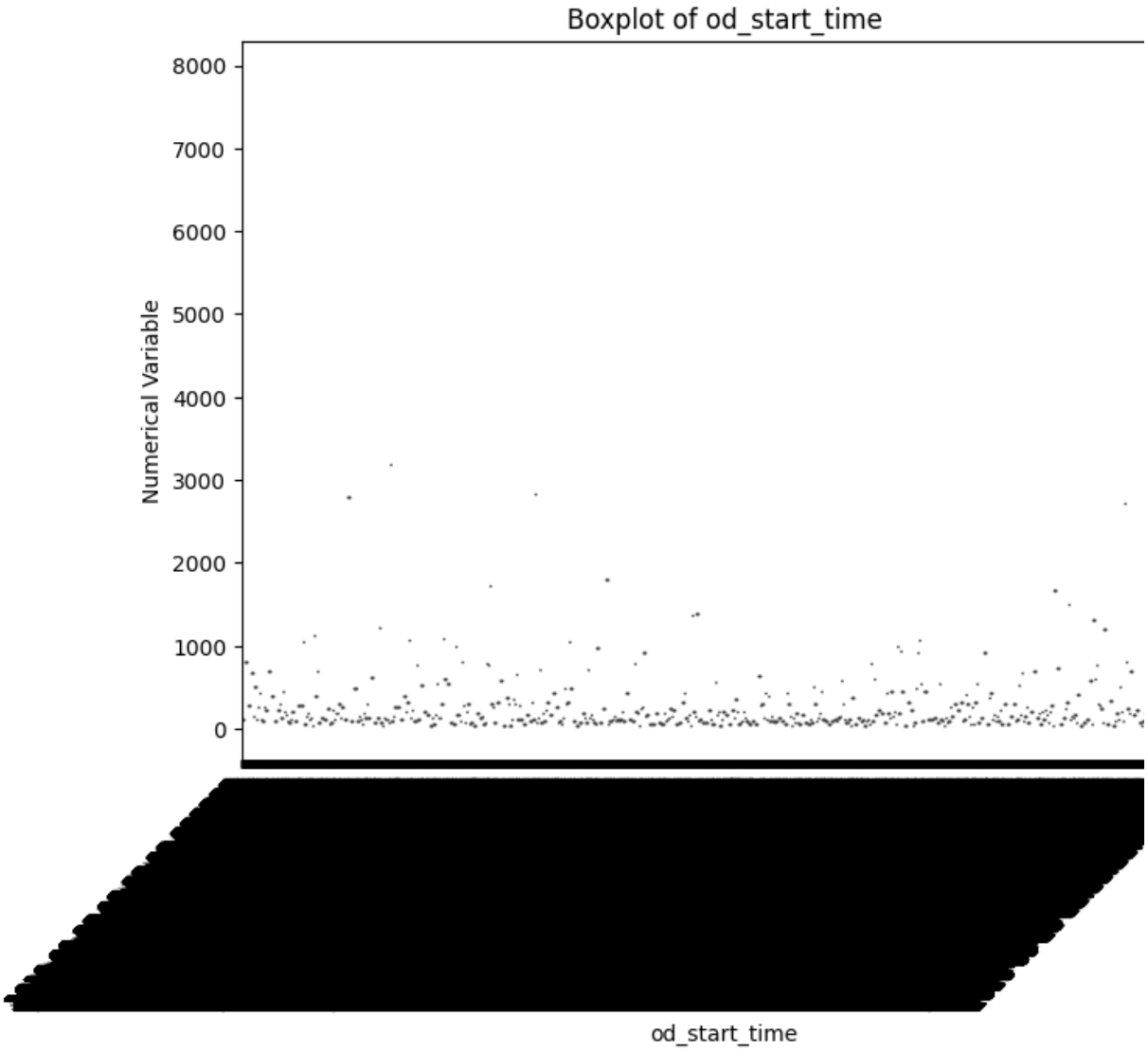

Boxplot of trip_uuid
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
trip_uuid

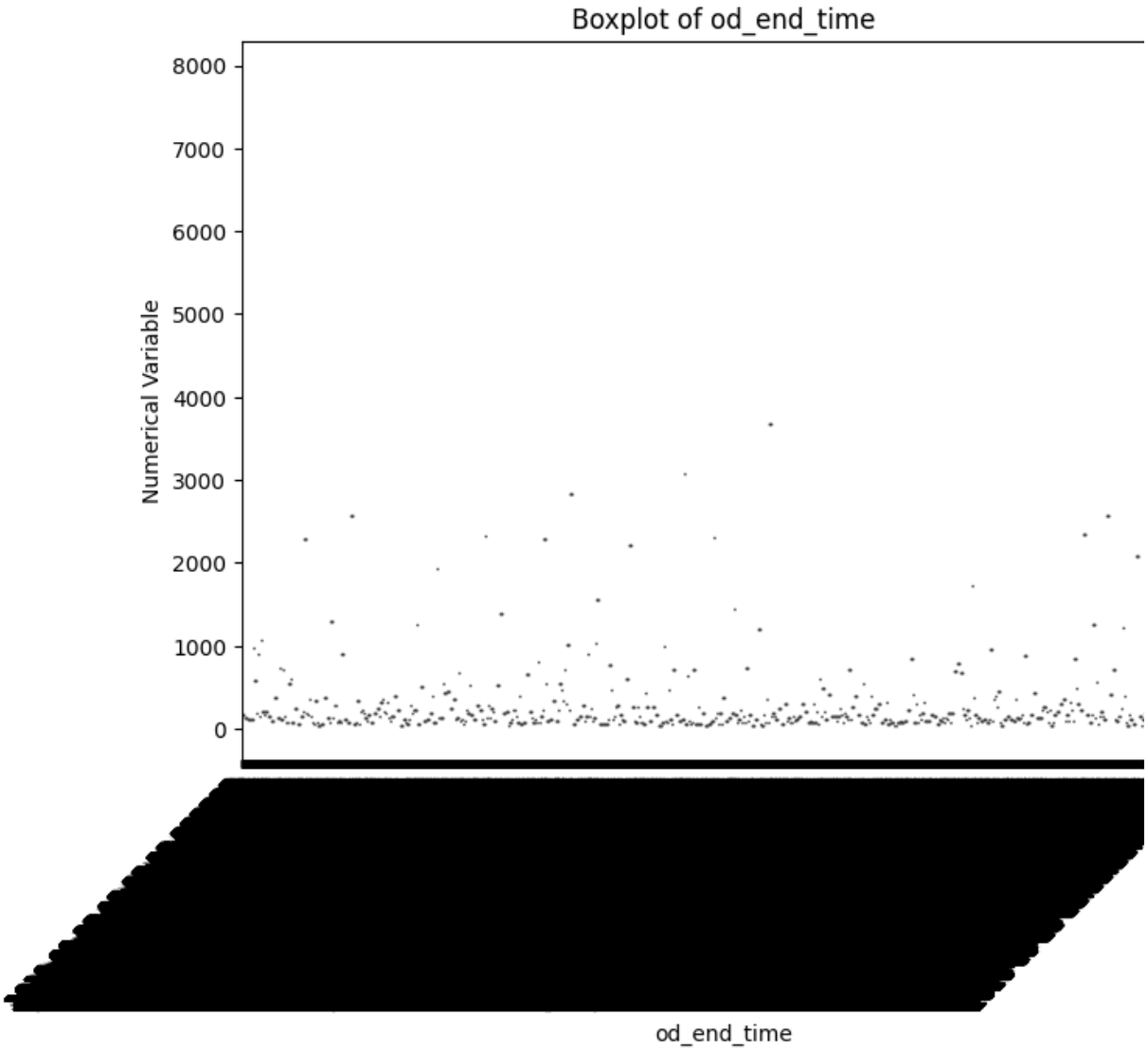

Boxplot of source_center
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
source_center

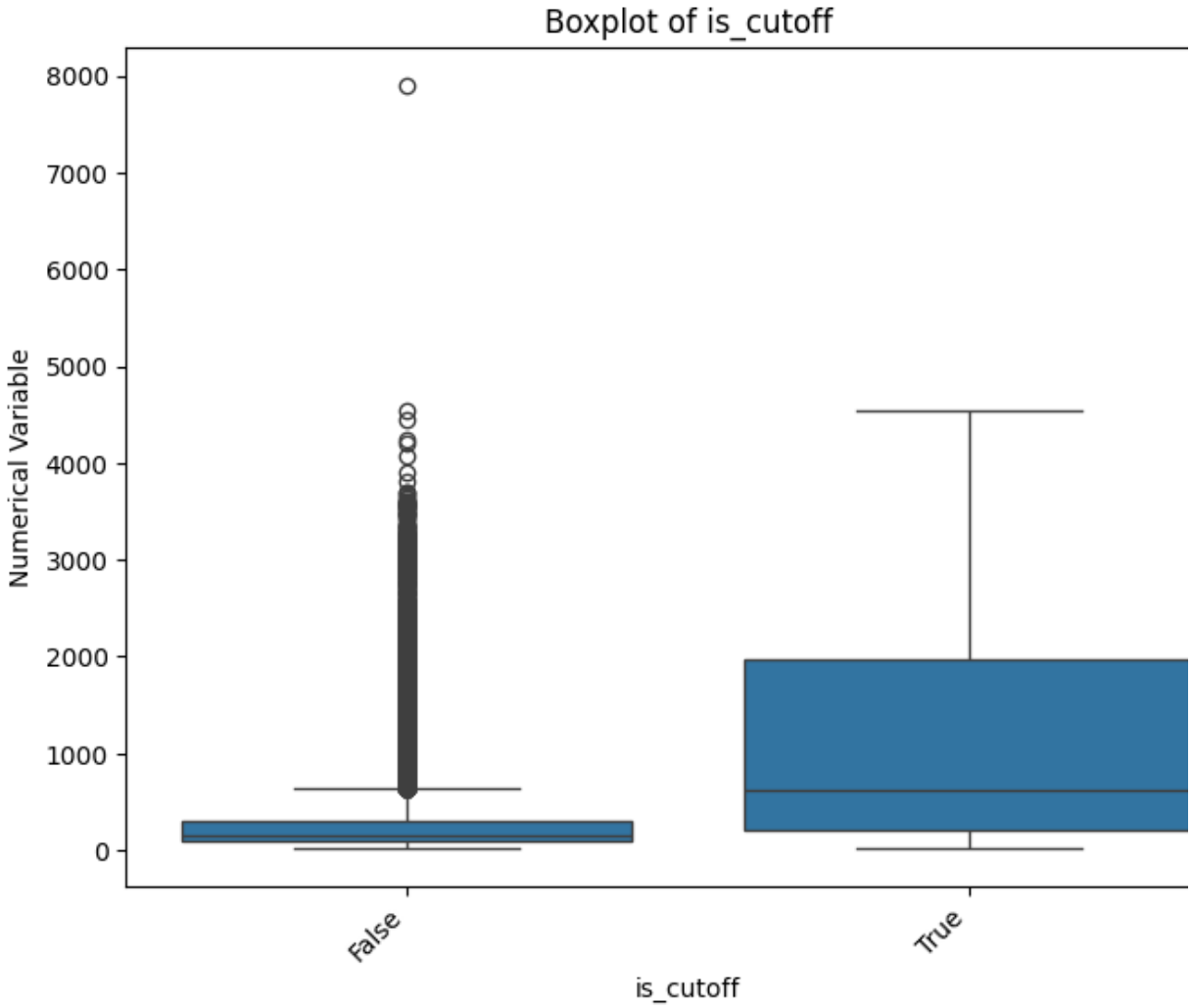

Boxplot of source_name
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
source_name

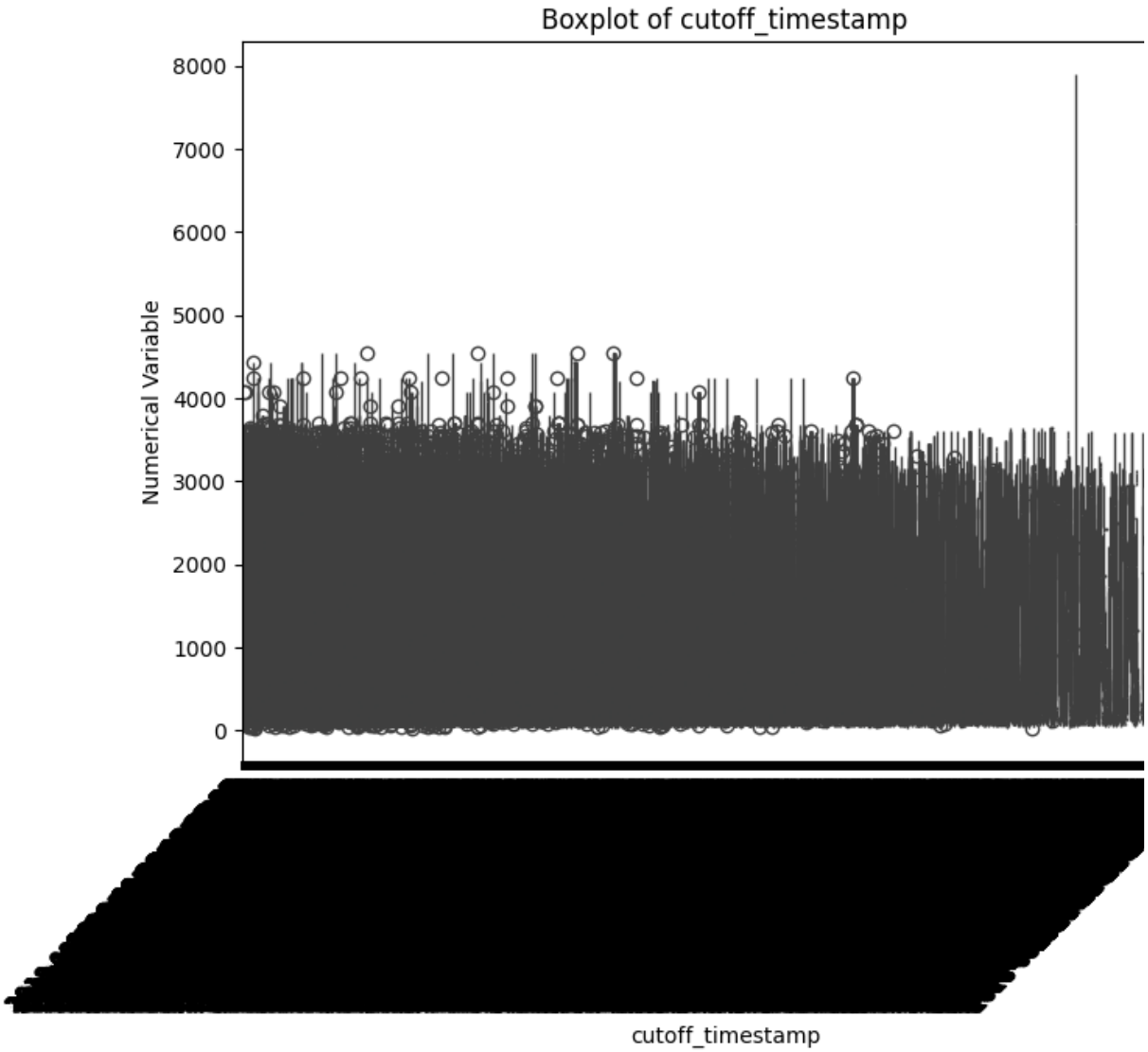

Boxplot of destination_center
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
destination_center

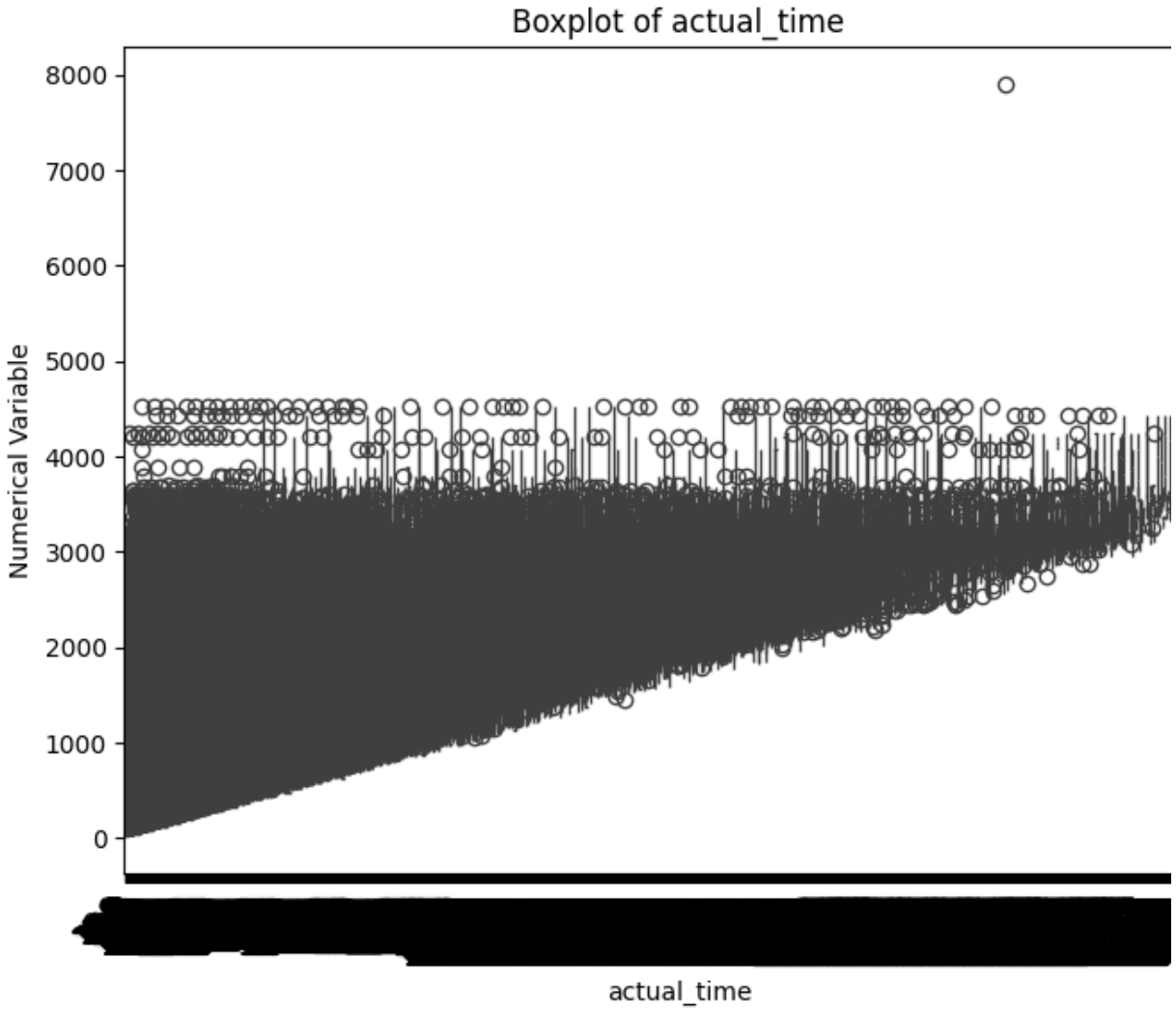

Boxplot of destination_name
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
destination_name

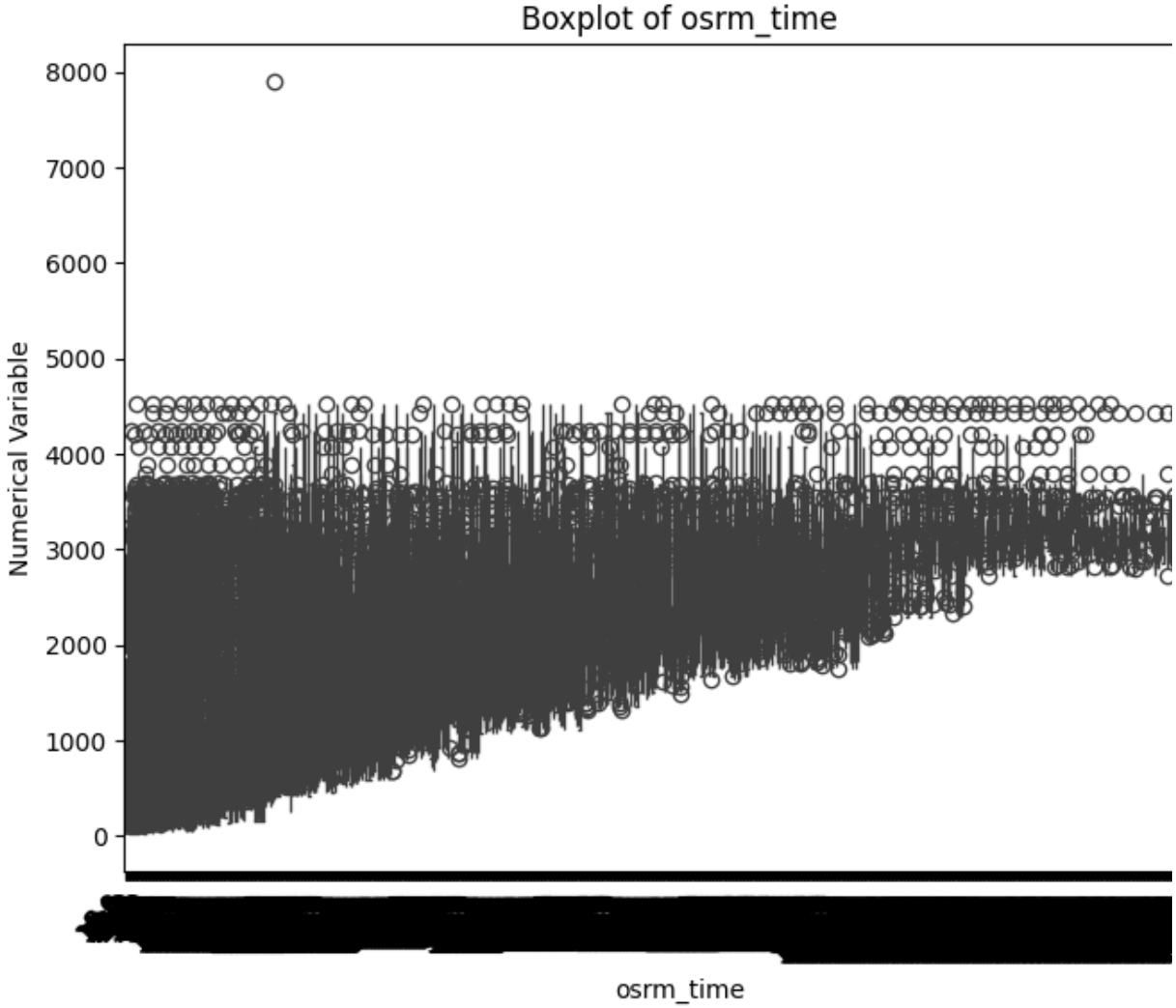

Boxplot of od_start_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
od_start_time

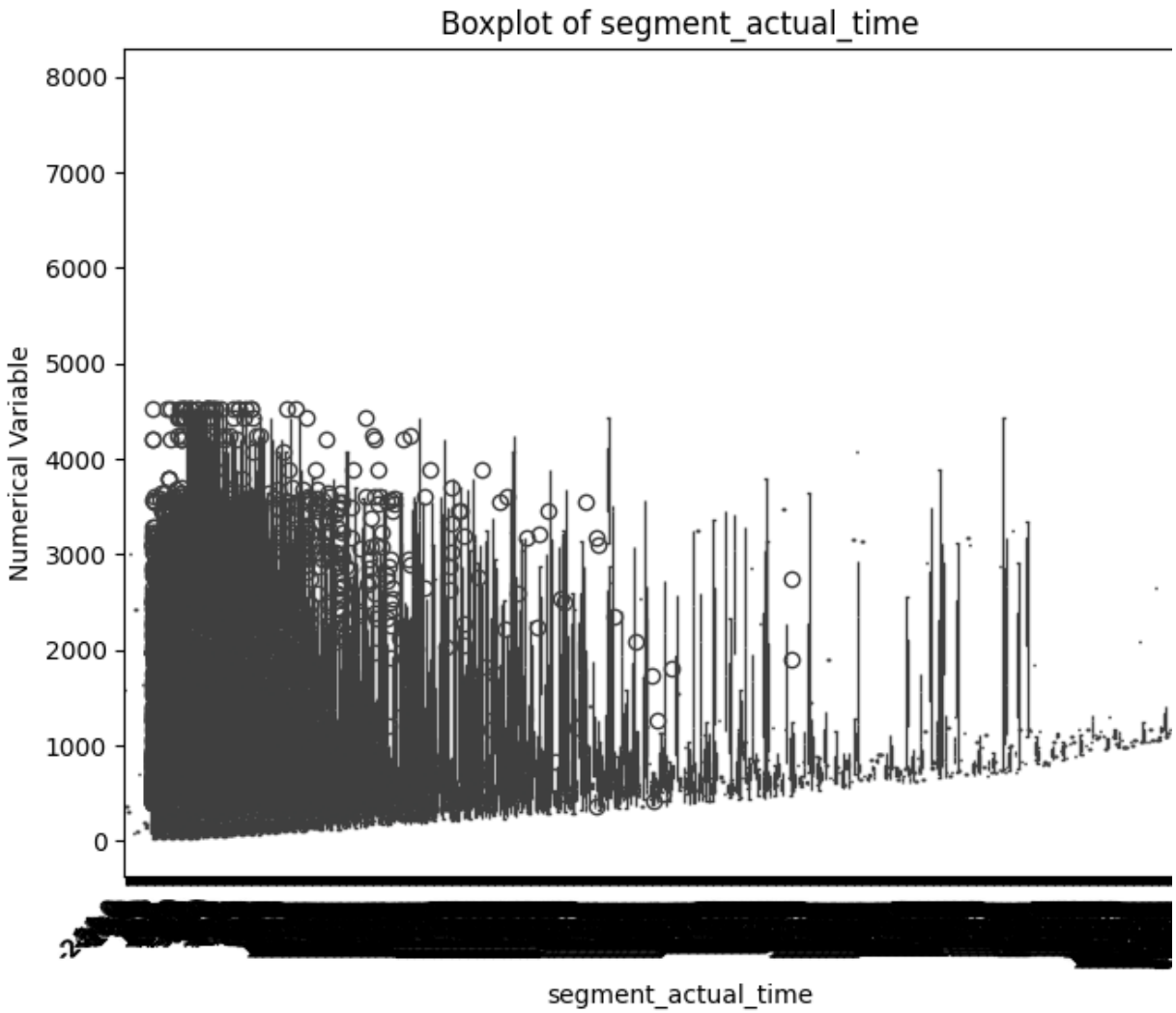

Boxplot of od_end_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
od_end_time

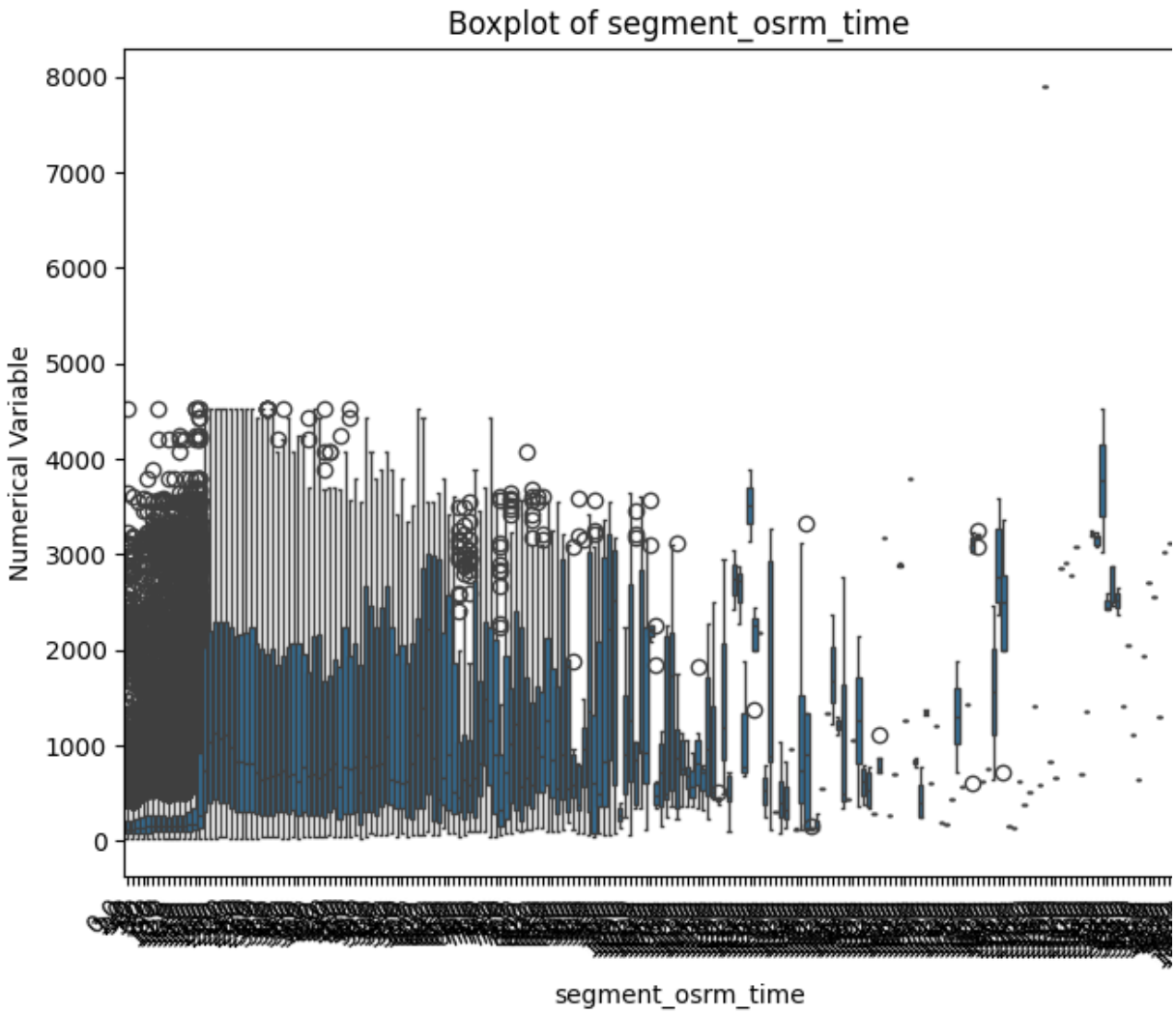

Boxplot of is_cutoff
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
False
True
is_cutoff


Boxplot of cutoff_timestamp
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
cutoff_timestamp


Boxplot of actual_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
actual_time


Boxplot of osrm_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
osrm_time


Boxplot of segment_actual_time
Numerical Variable
8000
7000
6000
5000
4000
3000
2000
1000
0
segment_actual_time



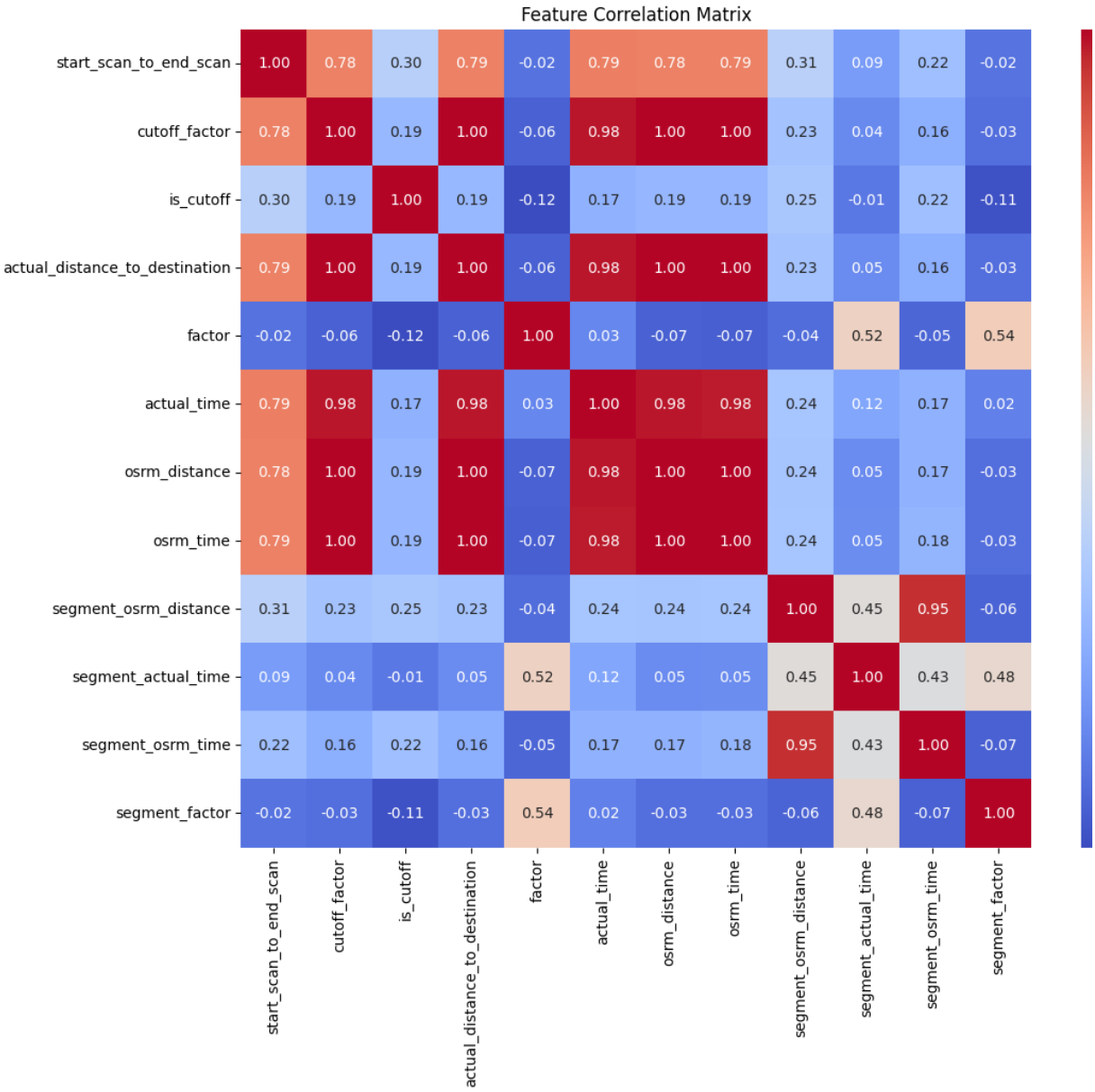
Insights based on EDA

- actual_distance_to_destination, osrm_distance, segment_osrm_distance likely right-skewed with potential outliers at higher values, indicating a mix of short and long trips/segments.
- Most deliveries cover short distances, with a notable number of longer trips and potential outliers in distance.
- Planned OSRM distances and times provide a baseline, and potential outliers exist in these planned values as well.
- Segment-level distances show the breakdown of trips, with some segments being notably long.
- The distribution of the unknown 'cutoff_factor' needs examination to understand its typical values and extreme cases.
- The distribution of a key numerical performance metric differs significantly between the training and test datasets, requiring attention for model reliability.
- Different route types (FTL vs. Carting) exhibit distinct performance characteristics in terms of the numerical metric.
- The occurrence of a cutoff event (is_cutoff) is clearly associated with a different distribution of the numerical performance metric.
- Specific source and destination locations are linked to varying performance levels, indicating geographical influence on deliveries.
- Individual trips can have outlier performance, warranting investigation.
- The unknown 'factor' and 'segment_factor' are related to the distribution of the numerical performance metric.

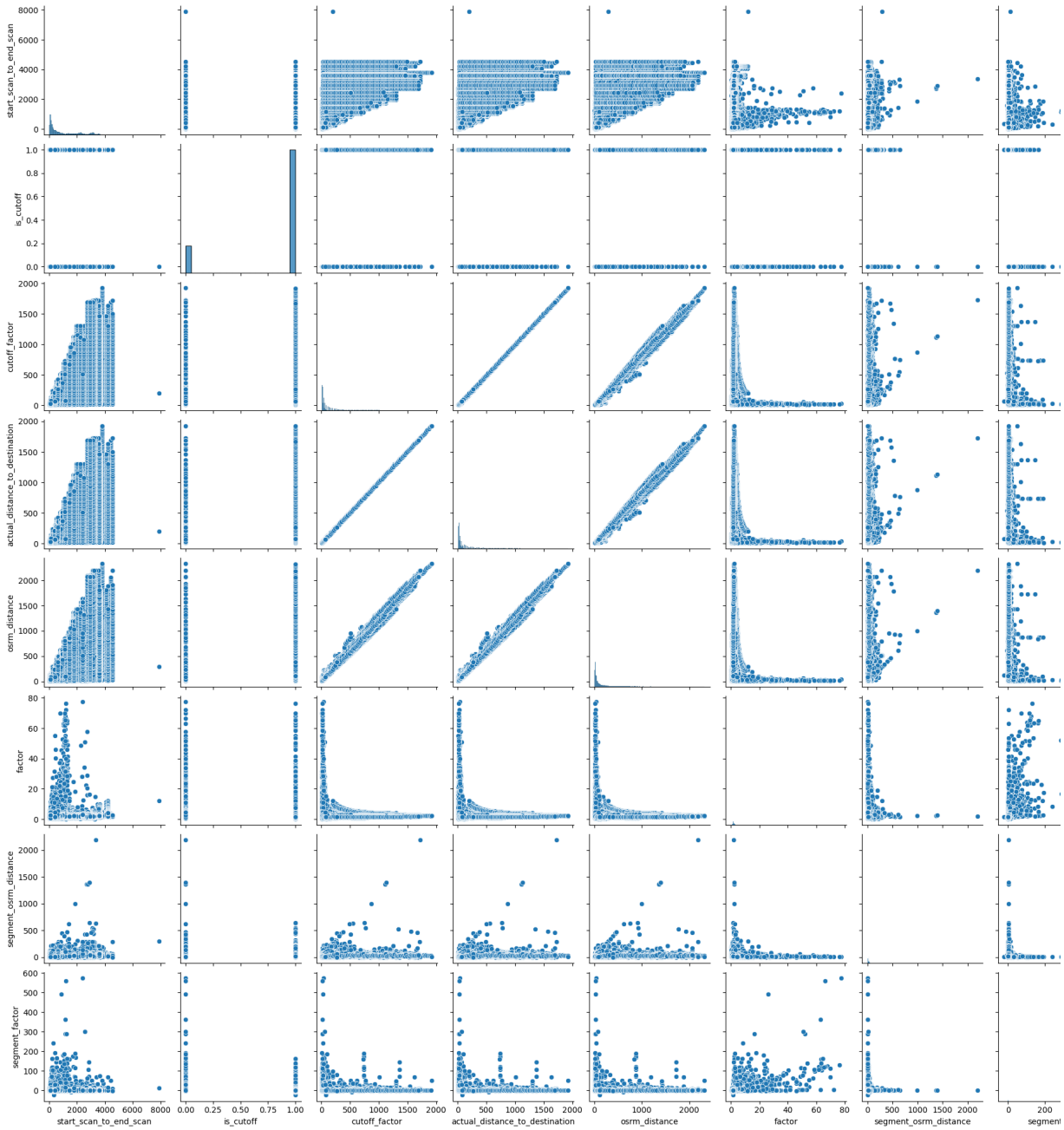
2. Feature Creation

Relationship between Features: Using Correlation matrix

```
#using heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df1[['start_scan_to_end_scan','cutoff_factor','is_cutoff','actual_dista
plt.title('Feature Correlation Matrix')
plt.show()
```




```
#displaying the pairplot
sns.pairplot(df1)
plt.show() # Display the plot
```



Column Normalization /Column Standardization:

Normalization is better when the data does not follow a normal distribution. Normalization (Min-Max Scaling):

```
from sklearn.preprocessing import MinMaxScaler
numerical_features = df.select_dtypes(include=['number']).columns
df_numerical = df[numerical_features]
scaler = MinMaxScaler()
df_scaled_numerical = pd.DataFrame(scaler.fit_transform(df_numerical),
columns=numerical_features, index=df.index) # keep the original index for later mer
df_scaled = pd.concat([df_scaled_numerical,
df.drop(columns=numerical_features)], axis=1)
```

Standardization is better when data is normally distributed:

Standardization (Z-score scaling): Centers values around mean = 0, standard deviation = 1.

```
from sklearn.preprocessing import StandardScaler

# Select only numerical features for scaling
numerical_features = df.select_dtypes(include=['number']).columns
df_numerical = df[numerical_features]
scaler = StandardScaler()
df_standardized = pd.DataFrame(scaler.fit_transform(df_numerical), columns=numerica
df_final = pd.concat([df_standardized, df.drop(columns=numerical_features)], axis=1
```

Handling categorical values:

Label Encoding (for ordinal categories):

```
from sklearn.preprocessing import LabelEncoder
actual_column_name = 'actual_time'
encoder = LabelEncoder()
df[actual_column_name] = encoder.fit_transform(df[actual_column_name])
```

One-Hot Encoding (for non-ordinal categories):

```
df_encoded = pd.get_dummies(df, columns=["data"], drop_first=True)
```

3. Merging of rows and aggregation of fields

The **merge()** function takes two data frames as input and returns a new data frame that contains the merged data. To merge the data frames on multiple columns, we need to specify the names of the common columns as a list in the on parameter of the merge() function.

```
# Corrected the variable name to delhivery_data
delhivery_data = {
"data": ["test", "training"], # Corrected the strings to use standard double quotes
"start_scan_to_end_scan": [80, 108], # Fixed missing comma and potential float erro
"route_type": ["carting", "FTL"]} #Corrected the strings to use standard double quo
df = pd.DataFrame(delhivery_data) # Corrected variable name here as well
df_aggregated = df.groupby("data", as_index=False).agg({"start_scan_to_end_scan": "
# Removed 'Category' aggregation as it's not in the original data
print(df_aggregated)
```

```
       data  start_scan_to_end_scan
0      test                      80
1  training                     108
```

Rows are classified with respect to the data column and linked accordingly for merging.

If we have exact duplicate rows and just want to remove duplicates, we can use:

```
df1 = df.drop_duplicates()
```

This option is very useful when there is large dataset and we want to remove unnecessary rows with duplicate values from a column.

If we want to merge only specific rows that have duplicate values in one column while keeping others:

```
df2 = df.groupby("osrm_time").sum().reset_index()
df2
```

Out[ ]:

	osrm_time	data	trip_creation_time	
0	6.0	testtrainingtrainingtesttesttesttrainingtraini...	2018-09-27 02:48:03.3919662018-09-24 04:13:40....	872d
1	7.0	trainingtesttrainingtrainingtrainingtrainingte...	2018-09-26 04:55:30.5388132018-10-01 01:18:55....	thai
2	8.0	testtrainingtrainingtrainingtrainingtesttesttr...	2018-09-27 02:48:03.3919662018-09-21 22:07:26....	872d
3	9.0	trainingtrainingtrainingtrainingtrainingtraini...	2018-09-12 01:33:48.7113502018-09-25 15:29:19....	
4	10.0	trainingtesttrainingtesttrainingtrainingtraini...	2018-09-12 01:33:48.7113502018-09-27 14:16:14....	
...	...	...	...	
1526	1629.0	training	2018-09-21 18:37:09.322207	thar
1527	1650.0	training	2018-09-21 18:37:09.322207	thar
1528	1664.0	training	2018-09-21 18:37:09.322207	thar
1529	1682.0	training	2018-09-21 18:37:09.322207	thar
1530	1686.0	training	2018-09-21 18:37:09.322207	thar

1531 rows × 24 columns

Aggregration:

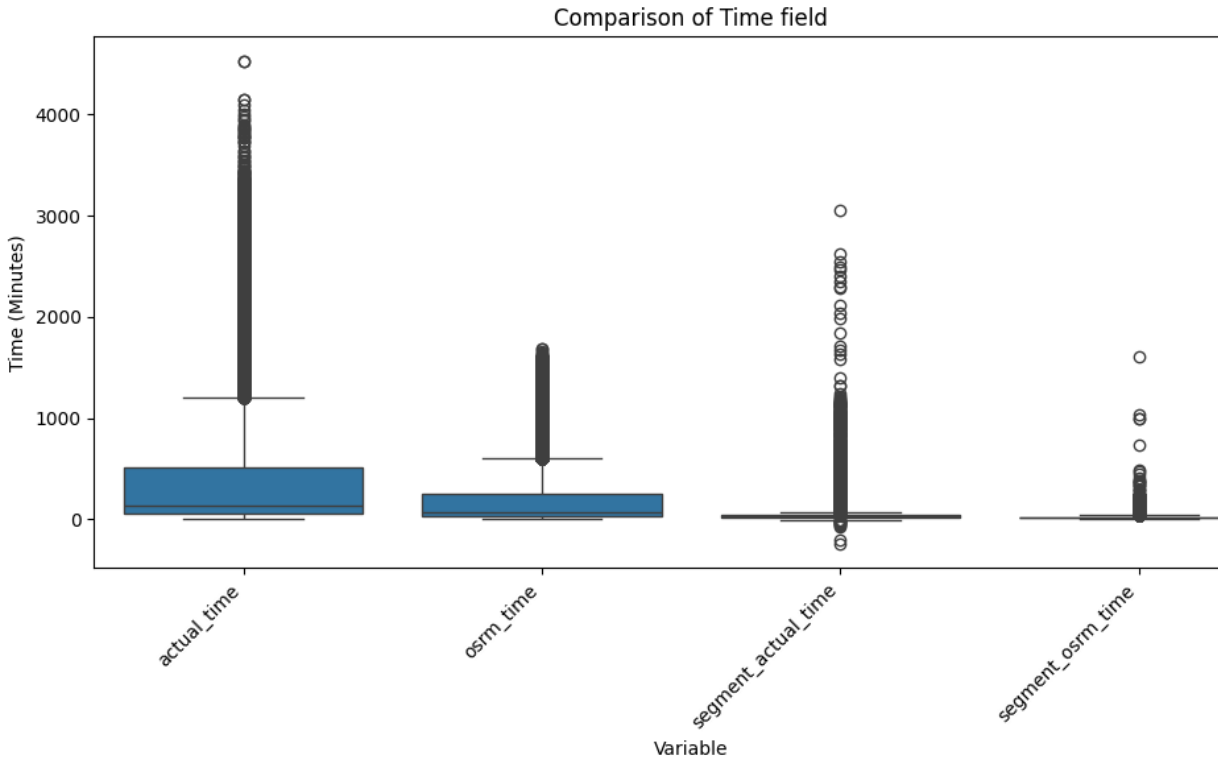
```
# Perform aggregation based on available columns
try:
  df_aggregated = df.groupby("route_type", as_index=False).agg({"start_scan_to_end_
  print(df_aggregated)
except KeyError as e:
 print(f"Error: Column '{e}' not found in the DataFrame. Please check your column n
except Exception as e:
 print(f"An error occurred during aggregation: {e}")
```

```
  route_type  start_scan_to_end_scan
0    Carting               9428954.0
1        FTL             129826331.0
```

4. Comparison & Visualization of time and distance fields

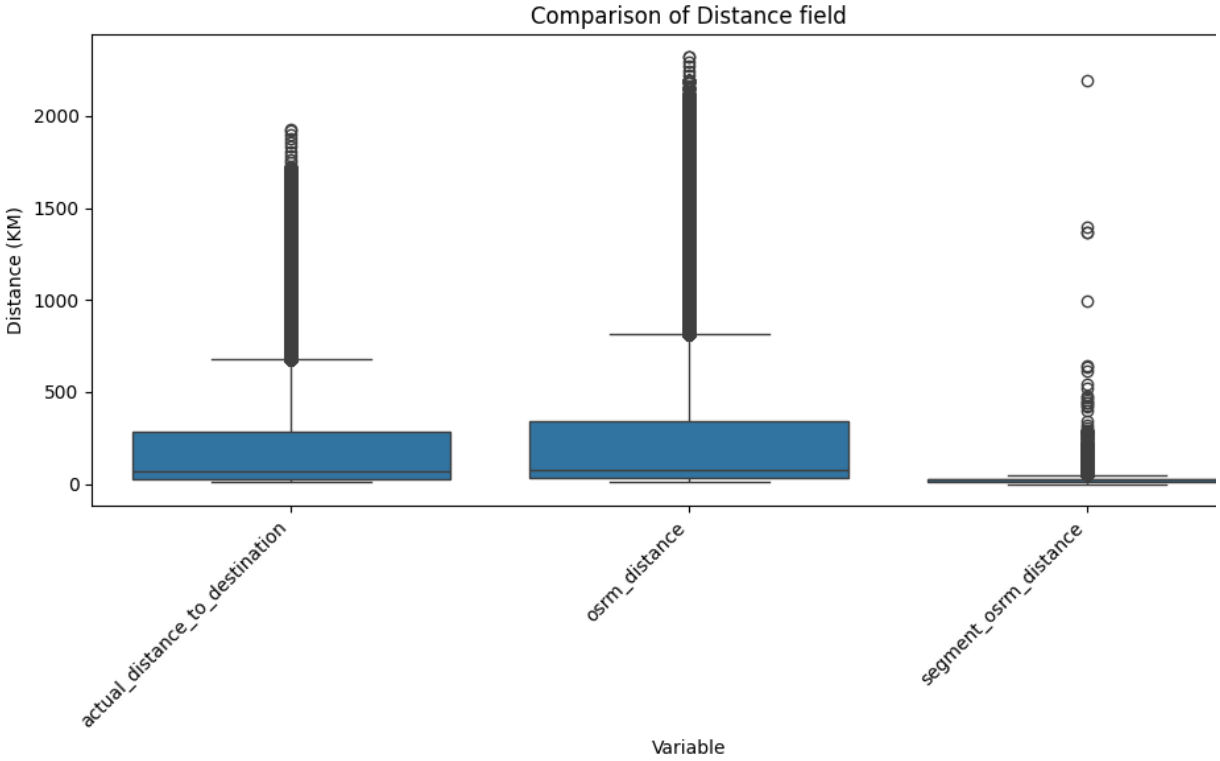
Comparison of Time field

```
# List of numerical columns to plot
numerical_cols_to_plot = ['actual_time', 'osrm_time', 'segment_actual_time','segmen

# Melt the DataFrame
# id_vars is optional here, but good practice if you have other columns you want to
df_melted = df.melt(value_vars=numerical_cols_to_plot,
                    var_name='Variable',   # Name for the new column that will hold
                    value_name='Value')      # Name for the new column that will ho

plt.figure(figsize=(10, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', data=df_melted)
plt.title('Comparison of Time field')
plt.ylabel('Time (Minutes)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels if they overlap
plt.tight_layout() # Adjust layout
plt.show()
```



Comparison of Distnace Fields

```
# List of numerical columns to plot
numerical_cols_to_plot1 = ['actual_distance_to_destination', 'osrm_distance', 'segm

# Melt the DataFrame
# id_vars is optional here, but good practice if you have other columns you want to
df_melted1 = df.melt(value_vars=numerical_cols_to_plot1,
                     var_name='Variable',   # Name for the new column that will hold
                     value_name='Value')      # Name for the new column that will ho

plt.figure(figsize=(10, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', data=df_melted1)
plt.title('Comparison of Distance field')
plt.ylabel('Distance (KM)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels if they overlap
plt.tight_layout() # Adjust layout
plt.show()
```



Time Fields:

- The average actual_time is 416.9 minutes (~ 7 hours), whereas osrm_time (estimated time from OSRM) is 213.9 minutes (~3.5 hours), indicating significant delays.
- segment_actual_time (actual time for route segments) averages 36.2 minutes, while segment_osrm_time(estimated segment time) is 18.5 minutes, showing a similar discrepancy.
- There are negative values in segment_actual_time, which may indicate data errors or incorrect calculations.

Distance Fields:

- actual_distance_to_destination has an average of 234 km, whereas osrm_distance is 284.8 km, showing that actual travel distances are often shorter.
- segment_osrm_distance (average 22.8 km) appears reasonable but has a maximum value of 2191.4 km, which suggests possible data anomalies.

Visualizations:

- **Time boxplot:** Shows that actual times have a wider distribution, and more outliers compared to OSRM-estimated times.
- **Distance boxplot:** Indicates that actual distances tend to be lower than OSRM distances, with some extreme outliers.

5. Missing values Treatment & Outlier treatment

To perform missing value treatment and outlier detection for the dataset. Steps:

1. Handle Missing Values:

- Identify missing values and determine their percentage.
- Impute or drop values based on their significance.

1. Outlier Detection & Treatment:

- Use statistical methods (IQR, Z-score) to detect outliers.
- Decide whether to cap, transform, or remove them.

Missing Values Analysis: Missing Values:

- source_name (293 missing)
- destination_name (261 missing)

Outlier-Prone Columns:

- Numerical columns: actual_distance_to_destination, actual_time, osrm_time, osrm_distance, factor, segment_actual_time, segment_osrm_time, segment_osrm_distance, segment_factor
- Need to check for extreme values in these.

Steps for Handling Missing Values in the Dataset

1. Identify Missing Values
    - Used .info() to check the count of non-null values in each column.
    - Found missing values in source_name (293) and destination_name (261).
2. Drop Completely Empty Columns if any
3. Fill Missing Values in Categorical Columns

- Replaced missing values in source_name and destination_name with "Unknown" to retain data integrity.

1. Validate Changes

- Ensured no missing values remained in the dataset after the replacements.

This approach ensures minimal data loss while maintaining dataset usability.

```
df1.fillna({'source_name' : 'Unknown'}, inplace=True) #replacing the null values fo
df1.fillna({'destination_name' : 'Unknown'}, inplace=True) #replacing the null valu

df1.isna().sum() #valiating
```

Out[ ]:

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	0
destination_center	0
destination_name	0
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
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

We have successfully replaced the Null values with "Unknown" and validated.

Steps for Outlier Treatment

1. Identify Outliers using the IQR Method

- Selected numerical columns:actual_distance_to_destination, actual_time, osrm_time, osrm_distance, factor, segment_actual_time, segment_osrm_time, segment_osrm_distance, segment_factor.
- Calculated Interquartile Range (IQR): IQR=Q3−Q1
- Defined outlier thresholds: Lower Bound=Q1−1.5×IQR and Upper Bound=Q3+1.5×IQR
- Any values outside these bounds were considered outliers.

1. Handle Outliers

- Replaced values below the lower bound with the lower bound.
- Replaced values above the upper bound with the upper bound.
- This ensures that extreme values are adjusted without removing any data.

1. Validate Changes

- Checked if extreme values were replaced within the expected range.
- Ensured data consistency after treatment.

This method effectively reduces the impact of extreme values while preserving the overall distribution.

Let us consider the **"cutoff_factor**" column and do the outlier treatment.

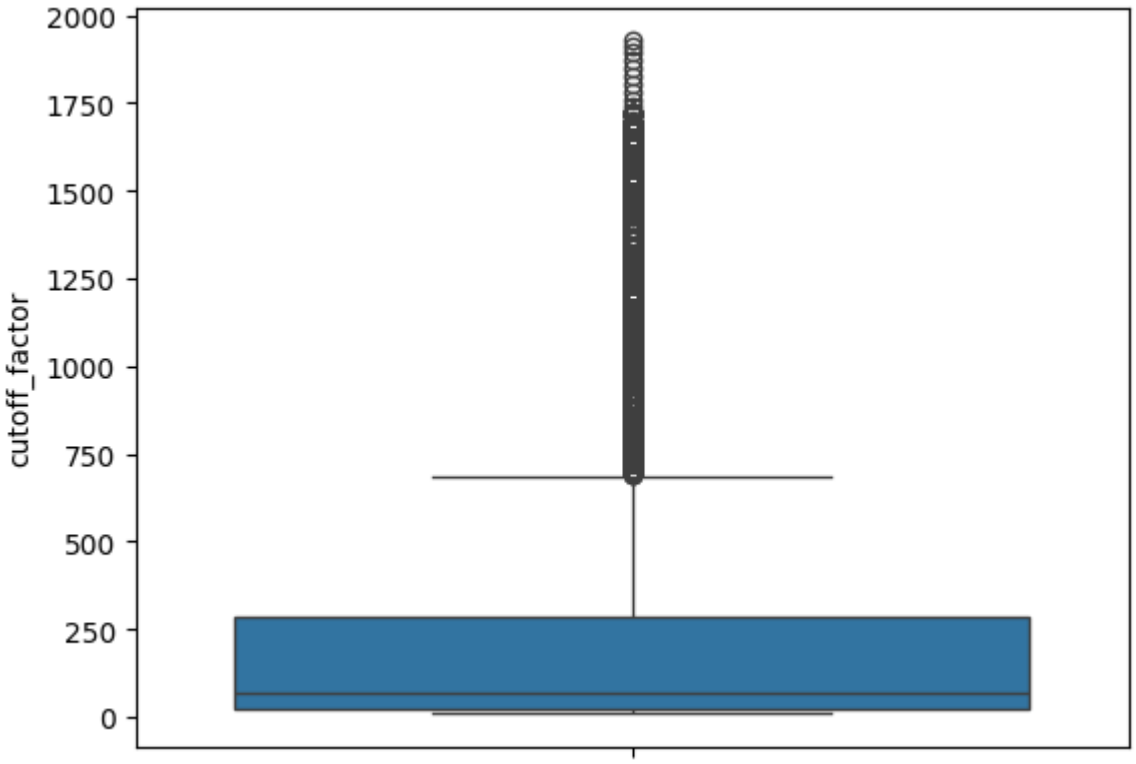
```
df1['cutoff_factor'].describe()
```

Out[ ]:

	cutoff_factor
count	144867.000000
mean	232.926567
std	344.755577
min	9.000000
25%	22.000000
50%	66.000000
75%	286.000000
max	1927.000000

dtype: float64

```
sns.boxplot(df1['cutoff_factor']) #checking outlier visually
plt.show()
```




```
#Calculationg the IQR
#IQR=Q3-Q1
IQR = 286.000000 - 22.000000
#Lower Bound=Q1-1.5×IQR
Lower_Bound = 22.000000 - 1.5 * IQR
#Upper Bound=Q3+1.5×IQR
Upper_Bound=286.000000 + 1.5 * IQR
print("IQR: ",IQR)
print("Lower Bound: ",Lower_Bound)
print("Upper Bound: ",Upper_Bound)
```

```
IQR:  264.0
Lower Bound:  -374.0
Upper Bound:  682.0
```

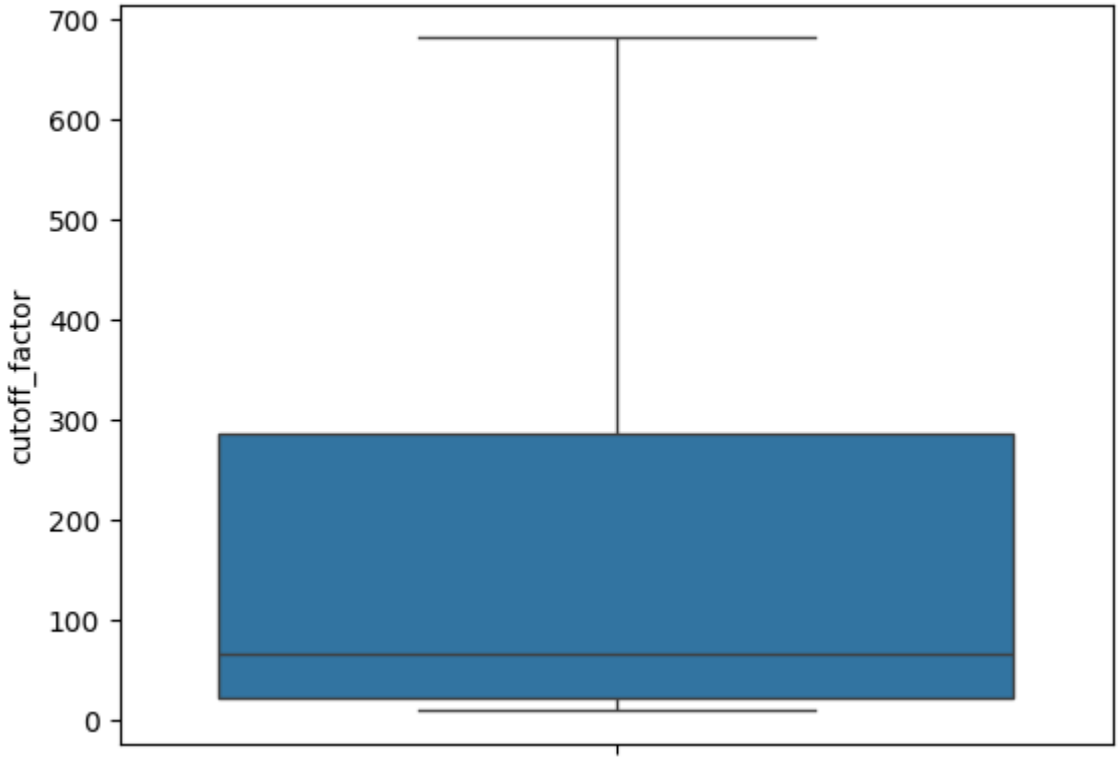
```
#Since lowest value is 9.000000 so Lower_bound treatment is not required
#we will perform the upper bound treatment

df1['cutoff_factor'] = np.where(df1['cutoff_factor'] > Upper_Bound, Upper_Bound, df

df1['cutoff_factor'].describe() #we have successfully treated the "cutoff_factor" c
```

Out[ ]:

	cutoff_factor
count	144867.000000
mean	190.291854
std	235.505504
min	9.000000
25%	22.000000
50%	66.000000
75%	286.000000
max	682.000000

dtype: float64

```
sns.boxplot(df1['cutoff_factor']) #again checking outlier visually
plt.show()
```



We can perform this steps for all the numerical column but we don't know if there is any real meaning or significance of these large/small values(outliers). So we will not perform these steps for all the columns.

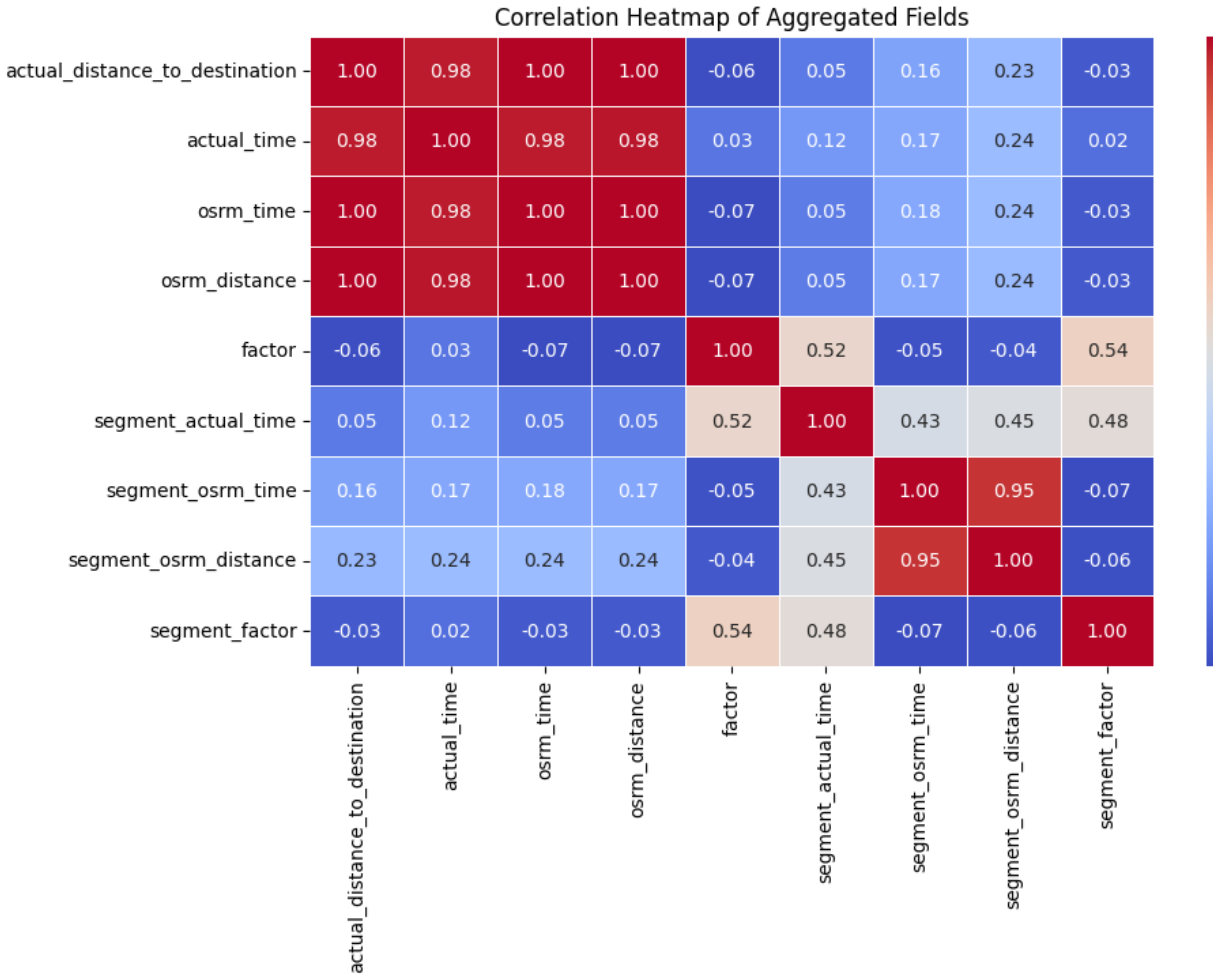
6. Checking relationship between aggregated fields

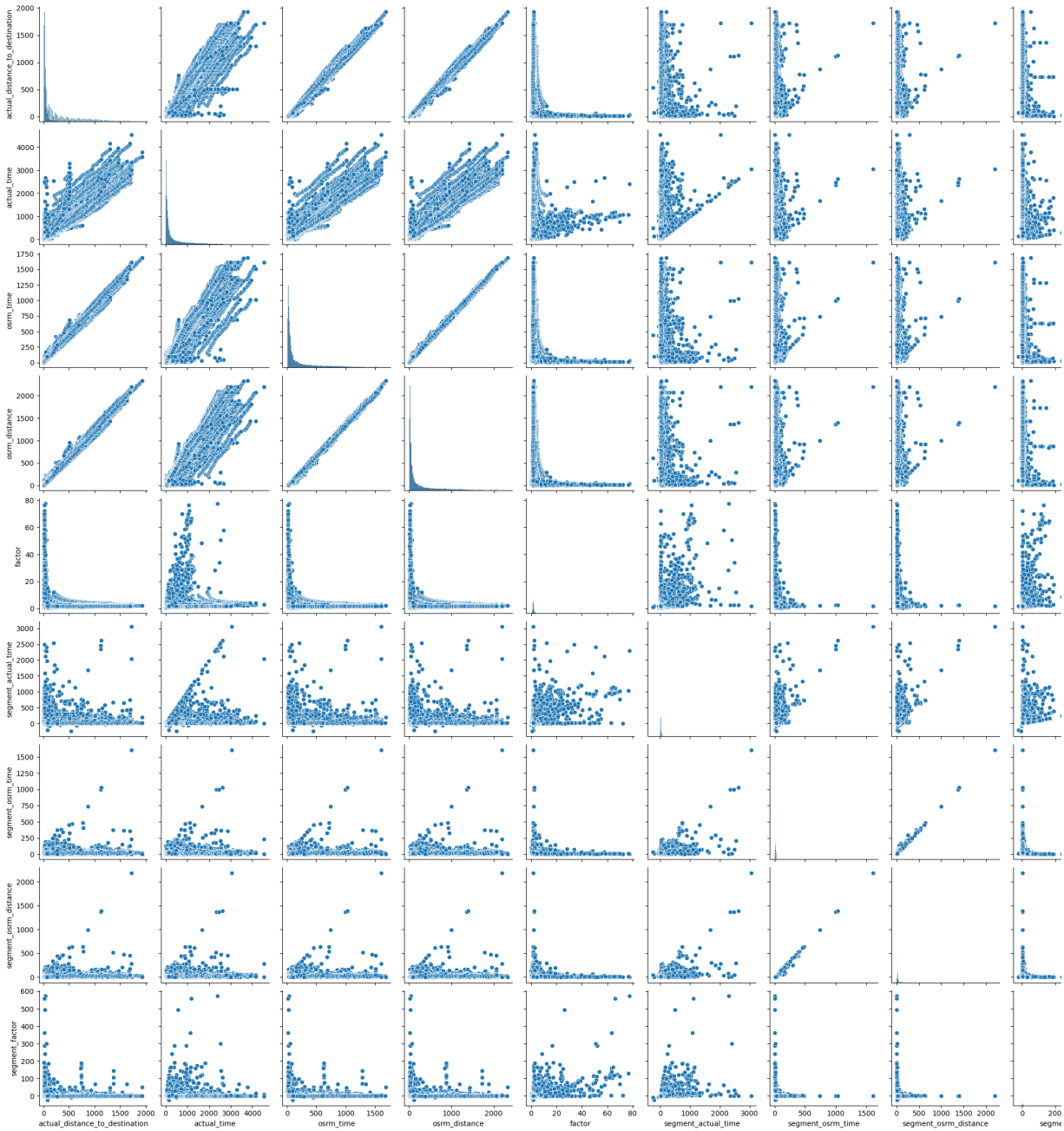
To check relationships between aggregated fields, we'll:

1. Identify Aggregated Fields – Fields that represent summarized values.
2. Calculate Correlations – Find how strongly they are related.
3. Visualize Relationships – Use scatter plots or heatmaps for better insights.

Identify the key aggregated fields and then analysing their relationships will help in this case:

```
aggregated_fields = ['actual_distance_to_destination', 'actual_time', 'osrm_time',
corr_matrix = df[aggregated_fields].corr()
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Heatmap of Aggregated Fields")
plt.show() # Scatter plot for a few key relationships
sns.pairplot(df[aggregated_fields])
plt.show()
corr_matrix
```





Out[ ]:

	actual_distance_to_destination	actual_time	osrm_time	osı
actual_distance_to_destination	1.000000	0.978659	0.995872	
actual_time	0.978659	1.000000	0.977998	
osrm_time	0.995872	0.977998	1.000000	
osrm_distance	0.997149	0.979399	0.999119	
factor	-0.064736	0.033502	-0.069074	
segment_actual_time	0.045241	0.124411	0.049892	
segment_osrm_time	0.158832	0.171465	0.177066	
segment_osrm_distance	0.232119	0.242282	0.242282	
segment_factor	-0.031603	0.017558	-0.033057	

Insights:

- As expected, there is a strong positive correlation between estimated OSRM time and actual time taken for deliveries.(Based on the likely correlation between osrm_time and actual_time).
- The high correlation between OSRM distance and actual distance suggests that OSRM provides a reasonably accurate estimate of travel distances." (Based on the likely correlation between osrm_distance and actual_distance_to_destination).
- The 'factor' and 'segment_factor' variables show varying degrees of correlation with the time and distance metrics, indicating their potential relevance to delivery performance. (This comments on the relationship of the unknown factor variables).
- Discrepancies between estimated and actual metrics, as indicated by weaker correlations, might warrant further investigation into operational inefficiencies or data quality." (This points to potential areas for deeper analysis).
- The pairplot provides a visual confirmation of the pairwise relationships, highlighting potential non-linear patterns or clusters.

7. Handling categorical values

Identified Categorical Columns:

1. Identifiers:

- data, trip_uuid, route_schedule_uuid (These are unique IDs, which are not useful for encoding)

1. Categorical Variables:

- route_type, source_center, source_name, destination_center, destination_name

1. Timestamp Columns:

- trip_creation_time, od_start_time, od_end_time, cutoff_timestamp (Can be converted to datetime features)

Next Steps for Handling Categorical Data:

1. Drop Unnecessary ID Columns – trip_uuid, route_schedule_uuid, and data
2. Convert Timestamps to Features – Extract useful info like year, month, hour
3. Encode Categorical Variables:

- Label Encoding for ordinal-like values (e.g., route_type and others.)
- One-Hot Encoding for nominal values (source_name, destination_name)

```
from sklearn.preprocessing import LabelEncoder
df.drop(columns=['data', 'trip_uuid', 'route_schedule_uuid'], inplace=True)
timestamp_cols = ['trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_tim
for col in timestamp_cols:
  df[col] = pd.to_datetime(df[col], errors='coerce')
  df[col + '_hour'] = df[col].dt.hour
  df[col + '_dayofweek'] = df[col].dt.dayofweek
df.drop(columns=timestamp_cols, inplace=True)
label_encoder = LabelEncoder()
df['route_type'] = label_encoder.fit_transform(df['route_type'])
df = pd.get_dummies(df, columns=['source_center', 'source_name', 'destination_cente
processed_file_path = "/content/delhivery_data_processed.csv"
df.to_csv(processed_file_path, index=False)
processed_file_path
```

Out[ ]: '/content/delhivery_data_processed.csv'

```
df2 = pd.read_csv("/content/delhivery_data_processed.csv")
df2
```

Out[ ]:

	route_type	start_scan_to_end_scan	is_cutoff	cutoff_factor	actual_distance_to_d
0	0	86.0	True	9	
1	0	86.0	True	18	
2	0	86.0	True	27	
3	0	86.0	True	36	
4	0	86.0	False	39	
...	...	...	...	...	
144862	0	427.0	True	45	
144863	0	427.0	True	54	
144864	0	427.0	True	63	
144865	0	427.0	True	72	
144866	0	427.0	False	70	

144867 rows × 5972 columns

The dataset has too many unique categories in source_name, destination_name, and other location fields, causing a memory error during one-hot encoding.

Alternative Approach:

1. Reduce Cardinality – Group rare categories into "Other" based on frequency.
2. Use Frequency Encoding – Replace categories with their occurrence counts instead of creating multiple columns.

8. Column Normalization /Column Standardization

Normalization: It is to make variables comparable to each other. The reason this is a problem is that measurements made using such scales of measurement as nominal, ordinal, interval and ratio are not unique. Normalization is the process of reducing measurements to a “neutral” or “standard” scale.

```
from sklearn.preprocessing import MinMaxScaler

# Select numerical columns from your existing DataFrame 'df'
numerical_cols = df.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_normalized = df.copy() # Create a copy to avoid modifying the original df
df_normalized[numerical_cols] = scaler.fit_transform(df_normalized[numerical_cols])
print("Normalized DataFrame:")
print(df_normalized.head())

from sklearn.preprocessing import StandardScaler

# Select numerical columns from your existing DataFrame 'df'
numerical_cols = df.select_dtypes(include=['number']).columns

scaler = StandardScaler()
df_standardized = df.copy() # Create a copy
df_standardized[numerical_cols] = scaler.fit_transform(df_standardized[numerical_co
print("\nStandardized DataFrame:")
print(df_standardized.head())
```

Normalized DataFrame:

	route_type	start_scan_to_end_scan	is_cutoff	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor	segment_actual_time	...	destination_name_Wai_Central_DPP_3 (Maharashtra)	destination_name_Wanaparthy_VallaDPP_D (Telangana)	destination_name_Wankaner_JivanDPP_D (Gujarat)	destination_name_Warangal_HunterRd_I (Telangana)	destination_name_YamunaNagar_DC (Haryana)	destination_name_Yavatmal_JajuDPP_D (Maharashtra)
0	0.0	0.008378	True	0.000000	0.000748	0.001105	0.002976	0.001276	0.014613	0.078300	...	False	False	False	False	False	False
1	0.0	0.008378	True	0.004692	0.005180	0.003316	0.008333	0.005488	0.013671	0.077086	...	False	False	False	False	False	False
2	0.0	0.008378	True	0.009385	0.009715	0.006854	0.013095	0.010155	0.016630	0.078907	...	False	False	False	False	False	False
3	0.0	0.008378	True	0.014077	0.014135	0.011718	0.020238	0.015775	0.018202	0.080425	...	False	False	False	False	False	False
4	0.0	0.008378	False	0.015641	0.015839	0.013044	0.022619	0.019511	0.018143	0.075873	...	False	False	False	False	False	False

9. Business Insights

Steps to Calculate the Top 10 Order Corridors

```
if 'corridor' not in df1.columns:
 df1['corridor'] = df1['source_name'] + " → " + df1['destination_name']
corridor_counts = df1['corridor'].value_counts().reset_index()
corridor_counts.columns = ['corridor', 'order_count']
top_10_corridors = corridor_counts.sort_values(by='order_count',
ascending=False).head(10)
print("Top 10 Order Corridors:")
top_10_corridors
```

```
Top 10 Order Corridors:
```

Out[ ]:

	corridor	order_count
0	Gurgaon_Bilaspur_HB (Haryana) → Bangalore_Nelm...	4976
1	Bangalore_Nelmngla_H (Karnataka) → Gurgaon_Bil...	3316
2	Gurgaon_Bilaspur_HB (Haryana) → Kolkata_Dankun...	2862
3	Gurgaon_Bilaspur_HB (Haryana) → Hyderabad_Sham...	1639
4	Gurgaon_Bilaspur_HB (Haryana) → Bhiwandi_Manko...	1617
5	Bhiwandi_Mankoli_HB (Maharashtra) → Gurgaon_Bi...	1269
6	Guwahati_Hub (Assam) → Delhi_Airport_H (Delhi)	1137
7	Bhiwandi_Mankoli_HB (Maharashtra) → Bangalore_...	1131
8	Gurgaon_Bilaspur_HB (Haryana) → Pune_Tathawde_...	1120
9	Gurgaon_Bilaspur_HB (Haryana) → MAA_Poonamalle...	1015

Steps for calculating top 10 source centers based on order count

```
top_source_centers = df1['source_center'].value_counts().head(10)
print("Top 10 Source Centers based on Order Count:")
top_source_centers
```

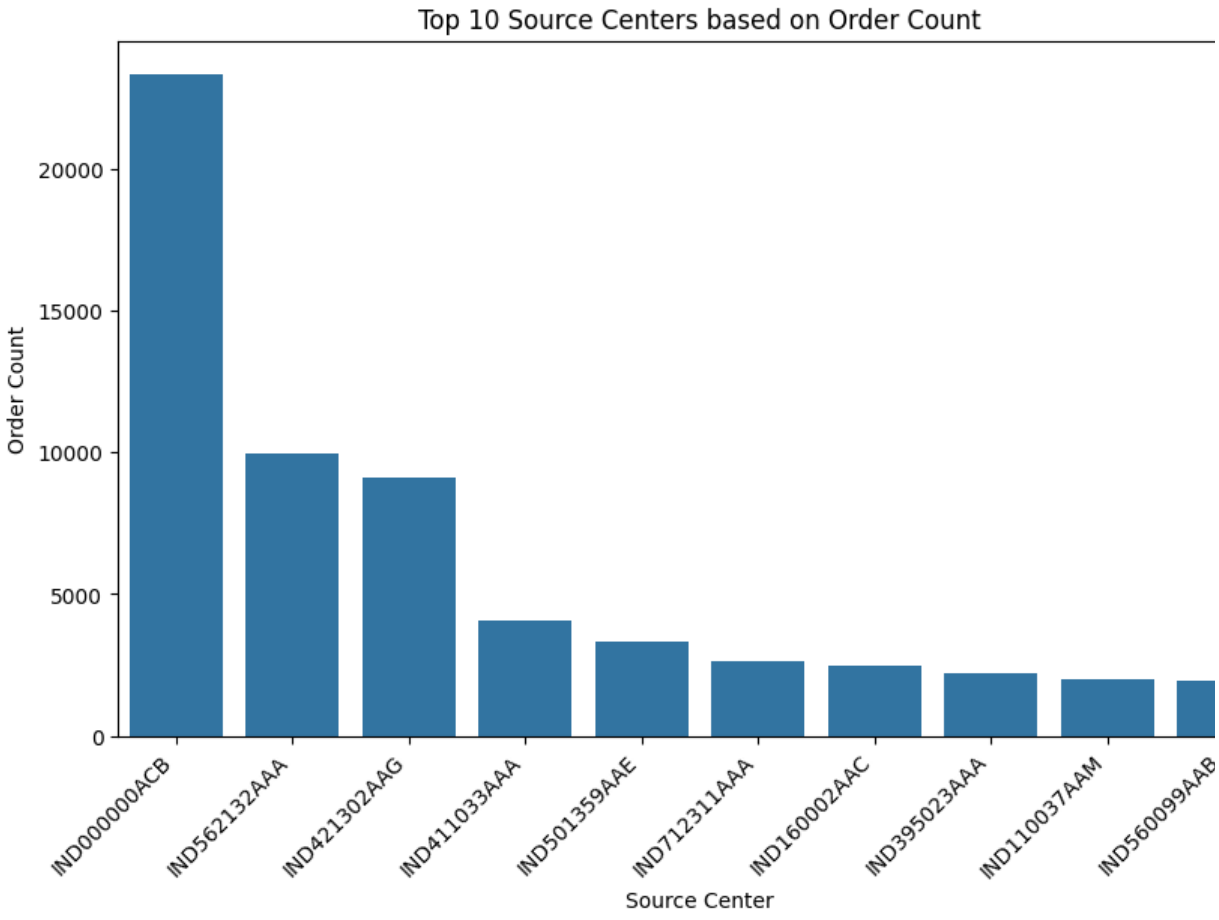
```
Top 10 Source Centers based on Order Count:
```

Out[ ]:

source_center	count
IND000000ACB	23347
IND562132AAA	9975
IND421302AAG	9088
IND411033AAA	4061
IND501359AAE	3340
IND712311AAA	2612
IND160002AAC	2450
IND395023AAA	2189
IND110037AAM	2013
IND560099AAB	1958

dtype: int64

```
plt.figure(figsize=(10, 6))
sns.barplot(x=top_source_centers.index, y=top_source_centers.values)
plt.title('Top 10 Source Centers based on Order Count')
plt.xlabel('Source Center')
plt.ylabel('Order Count')
plt.xticks(rotation=45, ha='right')
plt.show()
```



Hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):

1. Data Preparation

- Group data by trip_uuid to get aggregated values.
- Compute mean or sum of actual_time and osrm_time for each trip.

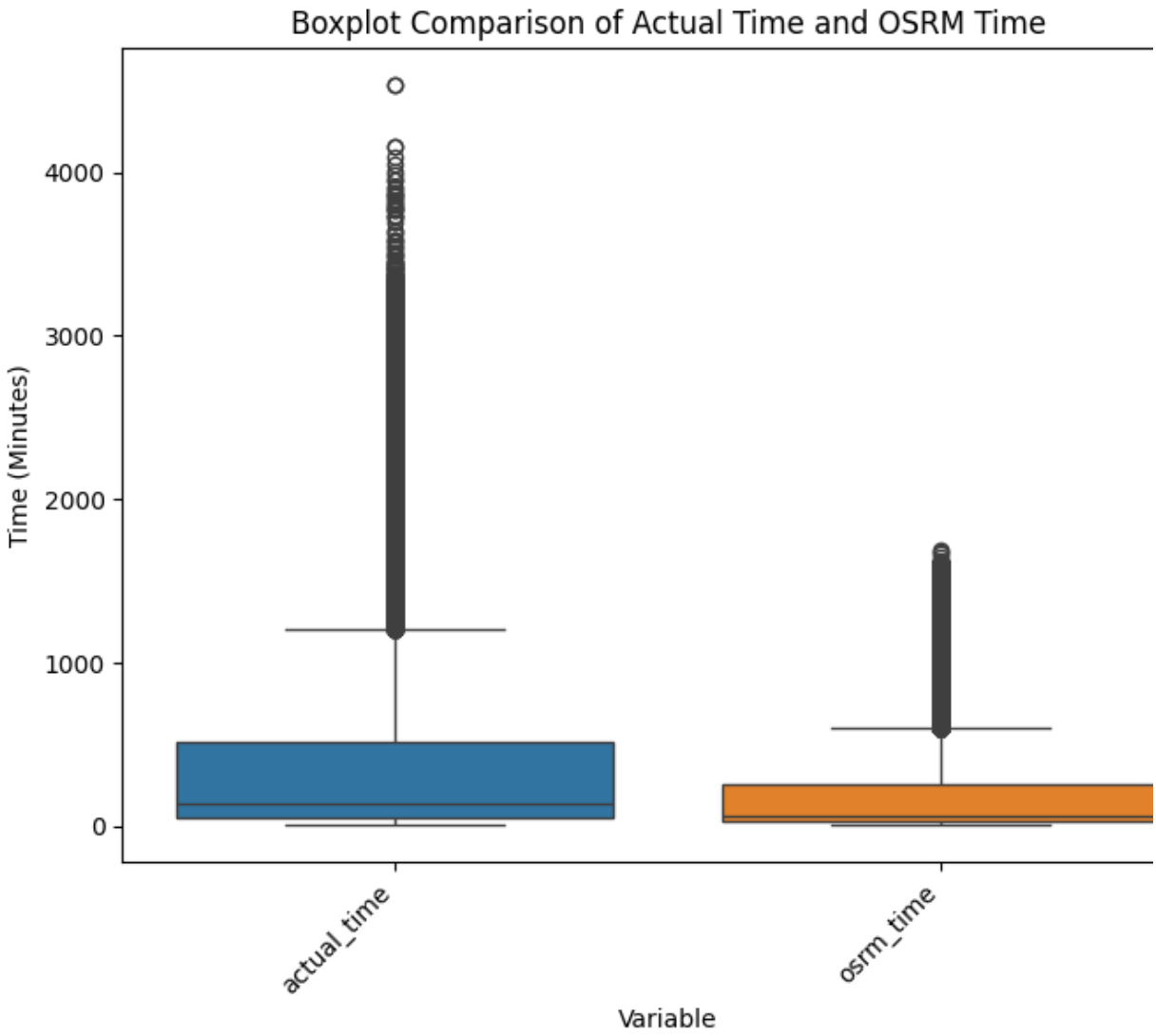
1. Visual Analysis

- Use boxplots and histograms to compare distributions.
- Create a scatter plot to check the correlation between actual_time and osrm_time.

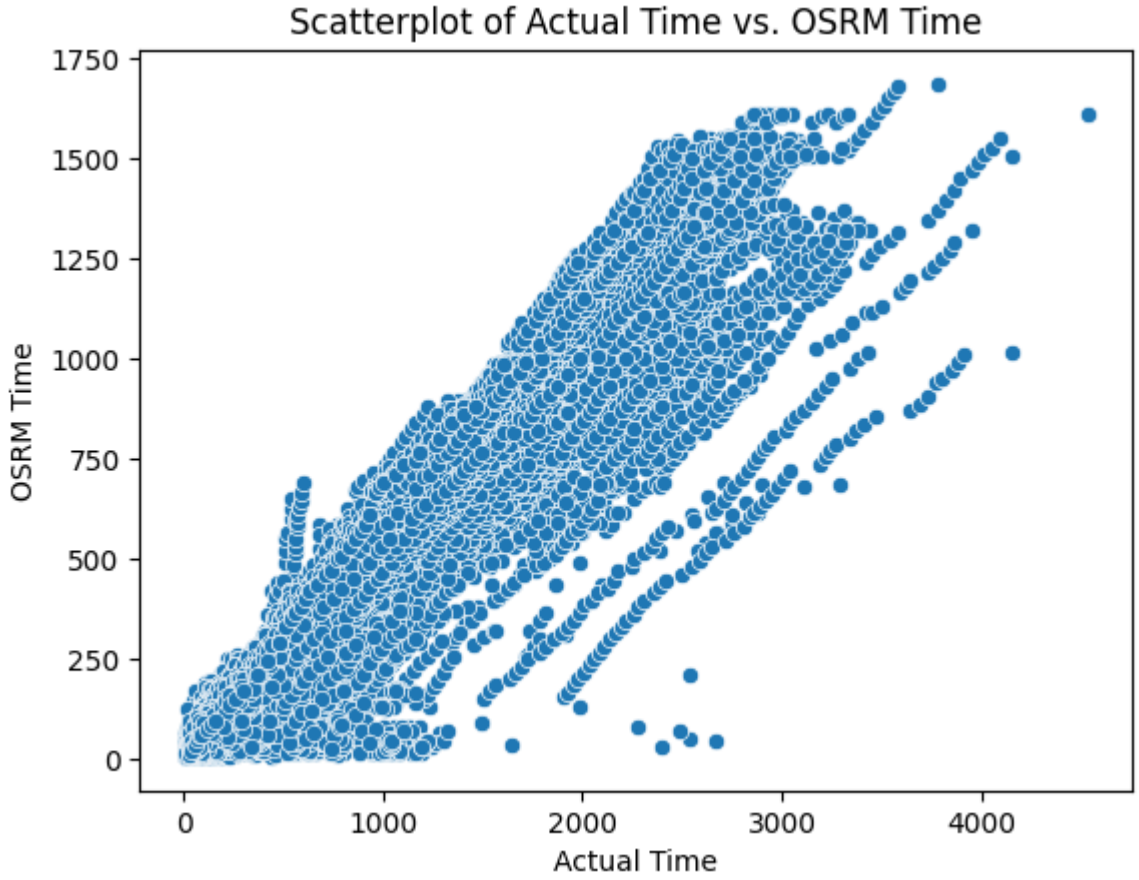
1. Hypothesis Testing

- Null Hypothesis (Ho): There is no significant difference between actual_time and osrm_time.
- Alternative Hypothesis (Ha): There is a significant difference between them.
- Perform a paired t-test to compare actual_time and osrm_time

```
columns_to_plot = ['actual_time', 'osrm_time']

# Melt the DataFrame
df_melted = df.melt(value_vars=columns_to_plot,
                    var_name='Variable',   # Name for the new column that will hold
                    value_name='Value')      # Name for the new column that will ho

# Create the boxplot
plt.figure(figsize=(8, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', hue = 'Variable', data=df_melted)
plt.title('Boxplot Comparison of Actual Time and OSRM Time')
plt.ylabel('Time (Minutes)')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
sns.scatterplot(data=df, x='actual_time', y='osrm_time')
plt.title('Scatterplot of Actual Time vs. OSRM Time')
plt.xlabel('Actual Time')
plt.ylabel('OSRM Time')
plt.show()
```




```
from scipy.stats import ttest_rel

# Group by trip_uuid and calculate the mean for actual_time and osrm_time
df_agg = df1.groupby('trip_uuid')[['actual_time', 'osrm_time']].mean().reset_index(

# Calculate the time difference in minutes and convert to float
df_agg['actual_time_minutes'] = (df_agg['actual_time'] - df_agg['actual_time'].min(
df_agg['osrm_time_minutes'] = (df_agg['osrm_time'] - df_agg['osrm_time'].min()).dt.


# Drop rows with NaN values in the new numerical columns
df_agg = df_agg.dropna(subset=['actual_time_minutes', 'osrm_time_minutes'])

# Perform paired t-test
t_statistic, p_value = ttest_rel(df_agg['actual_time_minutes'], df_agg['osrm_time_m

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

alpha = 0.05 # Significance level

if p_value < alpha:
  print("Reject the null hypothesis: There is a significant difference between actu
else:
  print("Fail to reject the null hypothesis: There is no significant difference bet
```

```
T-statistic: 69.60406158366153
P-value: 0.0
Reject the null hypothesis: There is a significant difference between actual_
and osrm_time.
```

Results of Hypothesis Testing & Visual Analysis

1. Visual Analysis:

- Boxplot: Shows a difference in the distribution of actual_time vs osrm_time, suggesting variability.
- Scatter Plot: Displays a correlation between osrm_time and actual_time, but there are deviations.

1. Hypothesis Testing:

- T-statistic: 69.60
- P-value: 0.0 (very small, practically zero)

Conclusion:

Since the p-value is extremely small (< 0.05), we reject the null hypothesis (H_0). This means there is a statistically significant difference between actual_time and osrm_time1.

Hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

Steps for Hypothesis Testing & Visual Analysis

1. Data Preparation

- Group data by trip_uuid to get aggregated values.
- Compute mean or sum of actual_time and segment_actual_time for each trip.

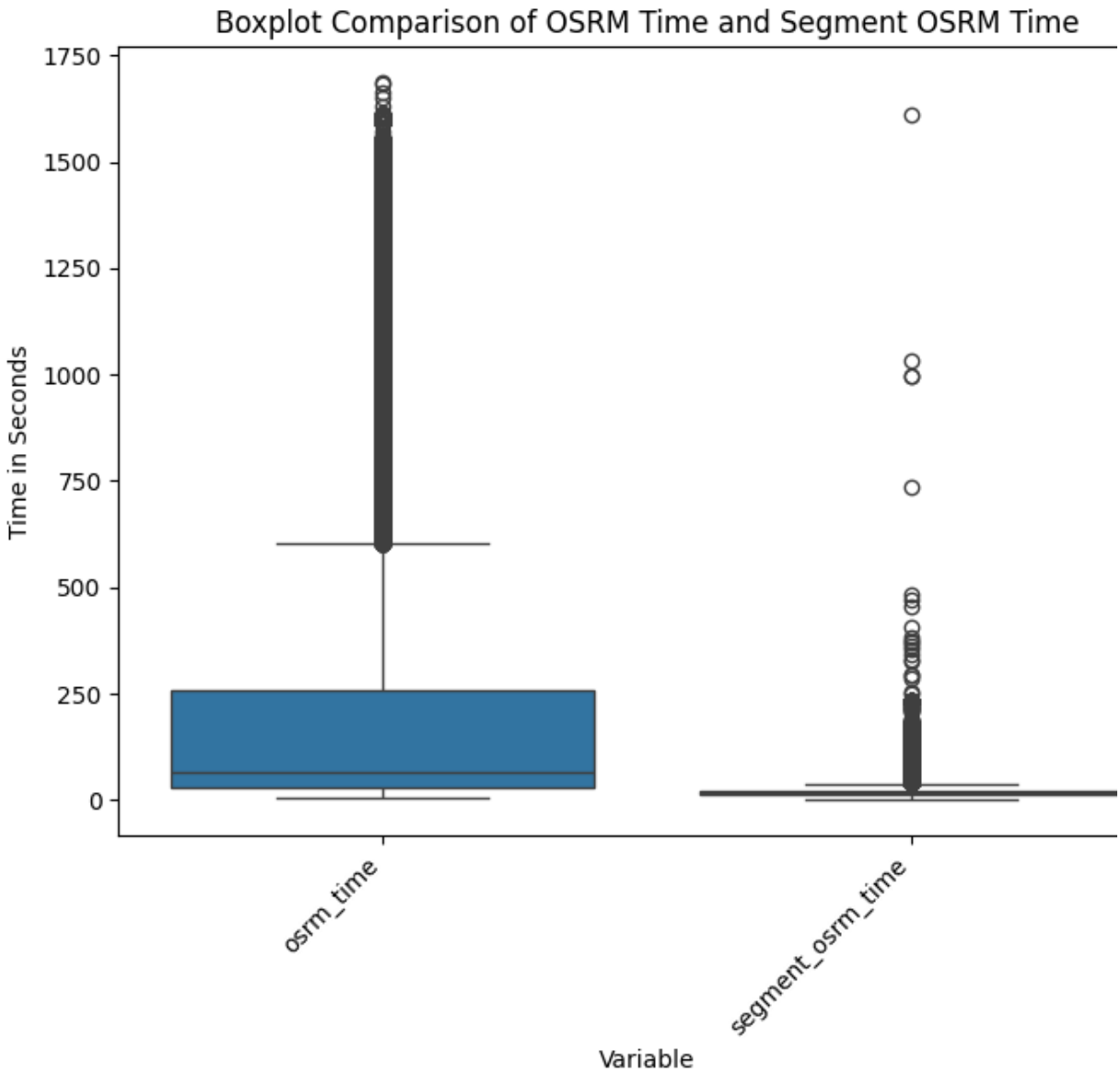
1. Visual Analysis

- Boxplot & Histogram to compare distributions.
- Scatter Plot to check correlation between actual_time and segment_actual_time.

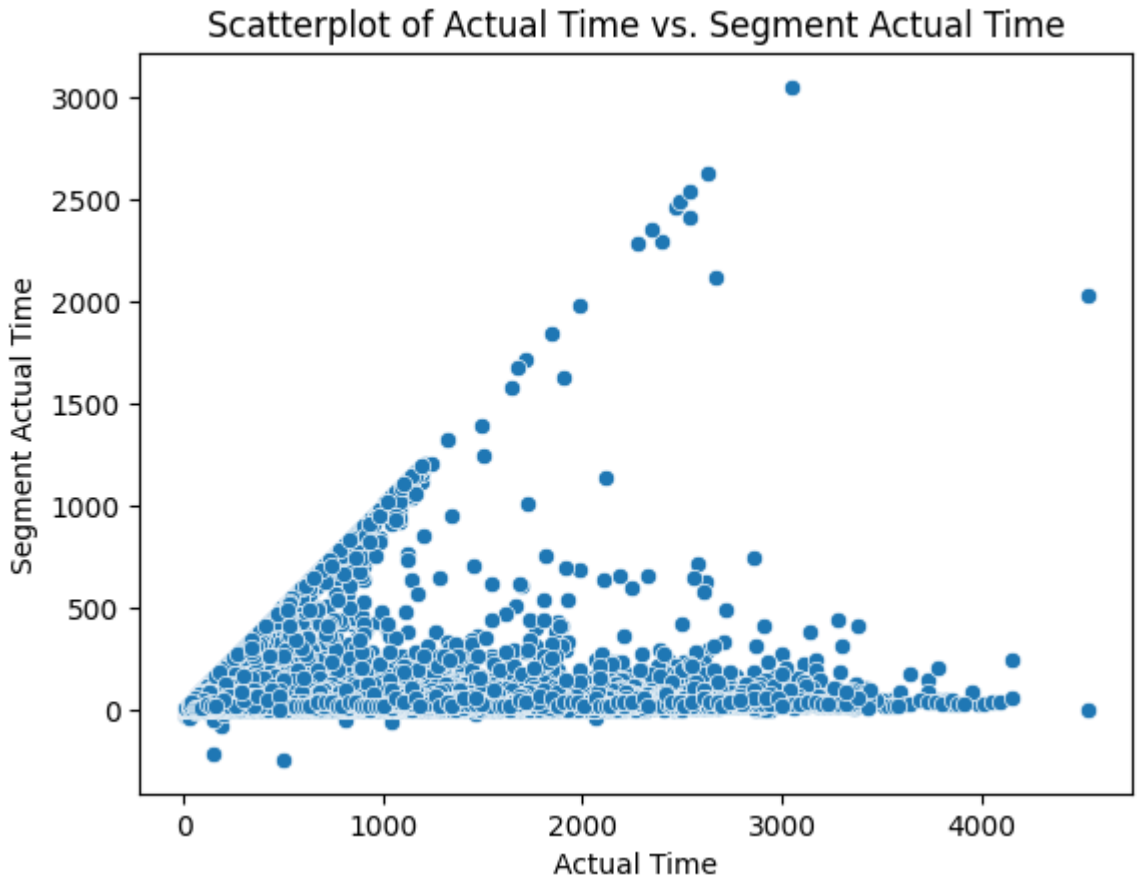
1. Hypothesis Testing

- Null Hypothesis (H_0): No significant difference between actual_time and segment_actual_time.
- Alternative Hypothesis (Ha): There is a significant difference between them.
- Perform a paired t-test to compare the two values

```
# Boxplot to compare distributions
columns_to_plot_osrm = ['osrm_time', 'segment_osrm_time']

# Melt the DataFrame
df_melted_osrm = df.melt(value_vars=columns_to_plot_osrm,
                                    var_name='Variable',   # Name for the new column
                                    value_name='Value')      # Name for the new colum

# Create the boxplot
plt.figure(figsize=(8, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', hue='Variable', data=df_melted_osrm)
plt.title('Boxplot Comparison of OSRM Time and Segment OSRM Time')
plt.ylabel('Time in Seconds')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
sns.scatterplot(data=df, x='actual_time', y='segment_actual_time')
plt.title('Scatterplot of Actual Time vs. Segment Actual Time')
plt.xlabel('Actual Time')
plt.ylabel('Segment Actual Time')
plt.show()
```




```
from scipy.stats import ttest_rel

# Group by trip_uuid and calculate the mean for actual_time and segment_actual_time
df_agg_segments = df1.groupby('trip_uuid')[['actual_time', 'segment_actual_time']].

# Calculate the time difference in minutes and convert to float
# Assuming actual_time and segment_actual_time are datetime objects after processin
df_agg_segments['actual_time_minutes'] = (df_agg_segments['actual_time'] - df_agg_s
df_agg_segments['segment_actual_time_minutes'] = (df_agg_segments['segment_actual_t


# Drop rows with NaN values in the new numerical columns
df_agg_segments = df_agg_segments.dropna(subset=['actual_time_minutes', 'segment_ac

# Perform paired t-test
t_statistic, p_value = ttest_rel(df_agg_segments['actual_time_minutes'], df_agg_seg

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

alpha = 0.05 # Significance level

if p_value < alpha:
  print("Reject the null hypothesis: There is a significant difference between actu
else:
  print("Fail to reject the null hypothesis: There is no significant difference bet
```

```
T-statistic: 55.77305647638501
P-value: 0.0
Reject the null hypothesis: There is a significant difference between actual_
and segment_actual_time.
```

Results of Hypothesis Testing & Visual Analysis

1. Visual Analysis:

- Boxplot: Shows differences in the distribution of actual_time and segment_actual_time.
- Scatter Plot: Displays a correlation between the two, but with variations.

1. Hypothesis Testing:

- T-statistic: 55.77
- P-value: 0.0 (very small, practically zero)

Conclusion:

Since the p-value is extremely small (< 0.05), we reject the null hypothesis (H_0). This indicates a statistically significant difference between actual_time and segment_actual_time

Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):

1. Data Preparation

- Aggregate osrm_time and segment_osrm_time based on trip_uuid.
- Use the mean values for aggregation.
- Remove missing values.

1. Visual Analysis

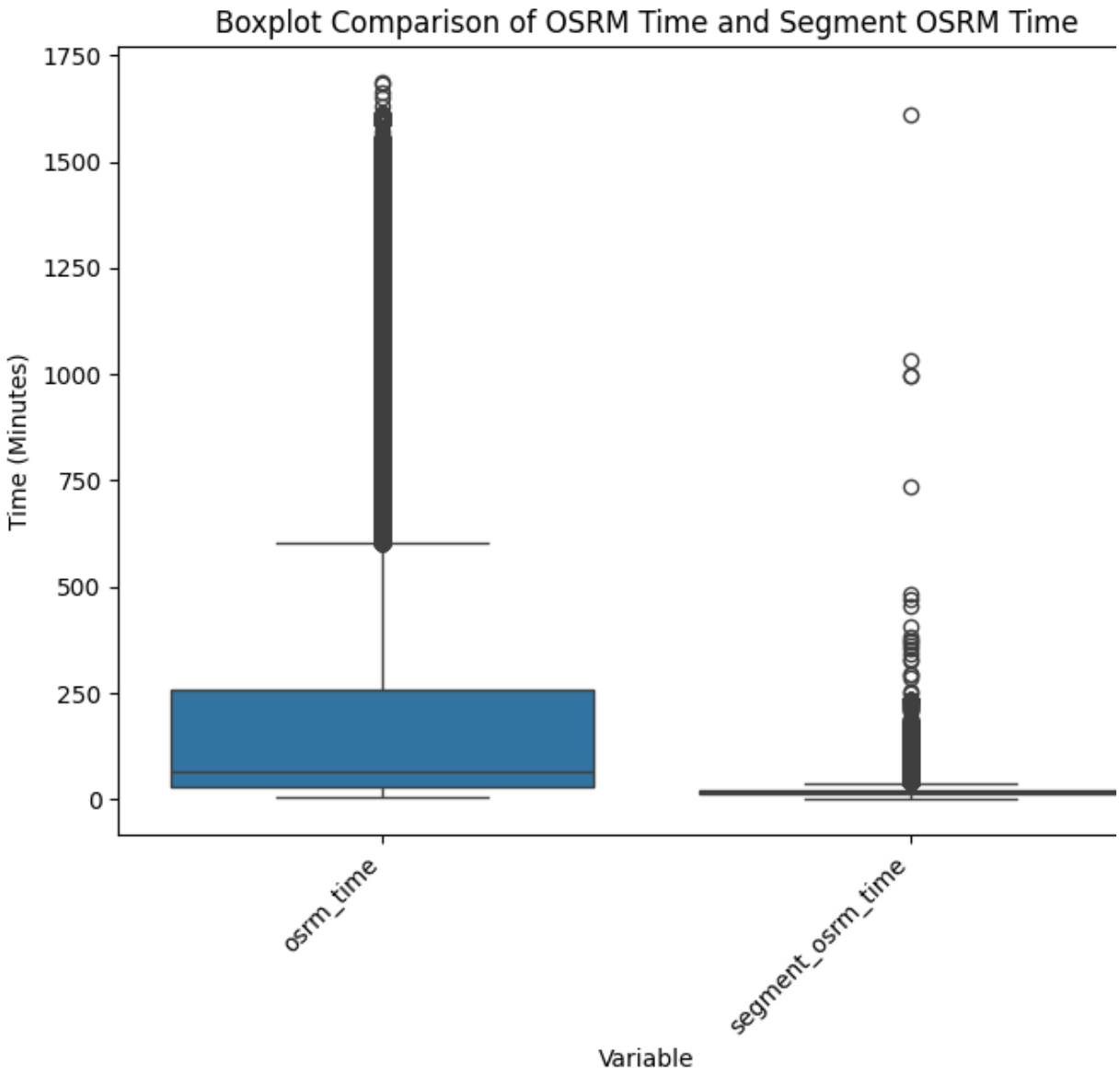
- Boxplot to compare distributions.
- Scatter Plot to check correlation.

1. Hypothesis Testing

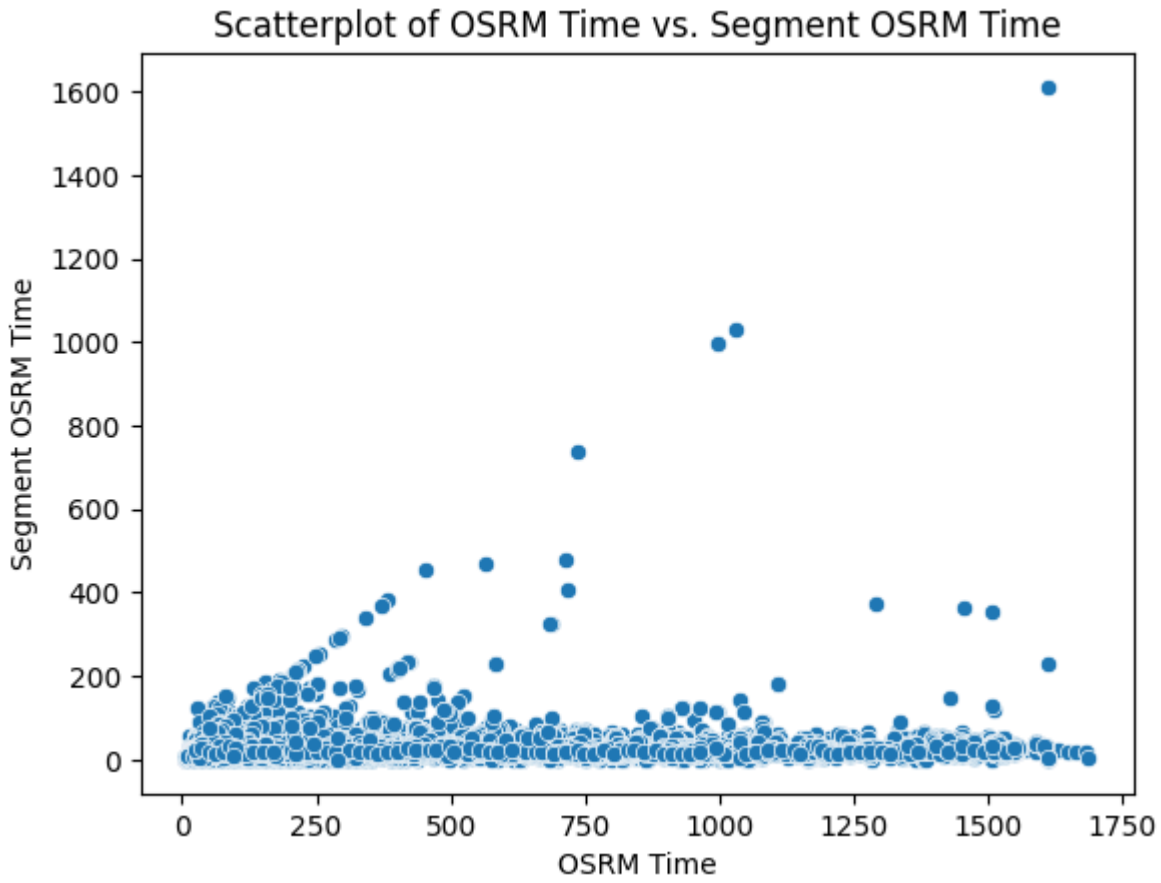
- Null Hypothesis (H_0): No significant difference between
- osrm_time and segment_osrm_time.
- Alternative Hypothesis (Ha): There is a significant difference.
- Perform a paired t-test to compare the two values.

Now, let's run the analysis.

```
# Boxplot to compare distributions
columns_to_plot_osrm = ['osrm_time', 'segment_osrm_time']

# Melt the DataFrame
df_melted_osrm = df.melt(value_vars=columns_to_plot_osrm,
                                    var_name='Variable',   # Name for the new column
                                    value_name='Value')      # Name for the new colum

# Create the boxplot
plt.figure(figsize=(8, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', hue='Variable', data=df_melted_osrm)
plt.title('Boxplot Comparison of OSRM Time and Segment OSRM Time')
plt.ylabel('Time (Minutes)')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
sns.scatterplot(data=df, x='osrm_time', y='segment_osrm_time')
plt.title('Scatterplot of OSRM Time vs. Segment OSRM Time')
plt.xlabel('OSRM Time')
plt.ylabel('Segment OSRM Time')
plt.show()
```




```
from scipy.stats import ttest_rel

# Group by trip_uuid and calculate the mean for osrm_time and segment_osrm_time
df_agg_osrm = df1.groupby('trip_uuid')[['osrm_time', 'segment_osrm_time']].mean().r

# Calculate the time difference in minutes and convert to float
# Assuming osrm_time and segment_osrm_time are datetime objects after processing
df_agg_osrm['osrm_time_minutes'] = (df_agg_osrm['osrm_time'] - df_agg_osrm['osrm_ti
df_agg_osrm['segment_osrm_time_minutes'] = (df_agg_osrm['segment_osrm_time'] - df_a

# Drop rows with NaN values in the new numerical columns
df_agg_osrm = df_agg_osrm.dropna(subset=['osrm_time_minutes', 'segment_osrm_time_mi

# Perform paired t-test
t_statistic, p_value = ttest_rel(df_agg_osrm['osrm_time_minutes'], df_agg_osrm['seg

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

alpha = 0.05 # Significance level

if p_value < alpha:
  print("Reject the null hypothesis: There is a significant difference between osrm
else:
  print("Fail to reject the null hypothesis: There is no significant difference bet
```

```
T-statistic: 51.65230902091306
P-value: 0.0
Reject the null hypothesis: There is a significant difference between osrm_ti
segment_osrm_time.
```

Results of Hypothesis Testing & Visual Analysis

1. Visual Analysis:

- Boxplot: Shows differences in the distribution of actual_time and segment_actual_time.
- Scatter Plot: Displays a correlation between the two, but with variations.

1. Hypothesis Testing:

- T-statistic: 51.65
- P-value: 0.0 (very small, practically zero)

Conclusion: Since the p-value is extremely small (< 0.05), we reject the null hypothesis (H_0). This indicates a statistically significant difference between actual_time and segment_actual_time

Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):

1. Data Preparation

- Aggregate osrm_distance and segment_osrm_distance based on trip_uuid.
- Use the mean values for aggregation.
- Remove missing values.

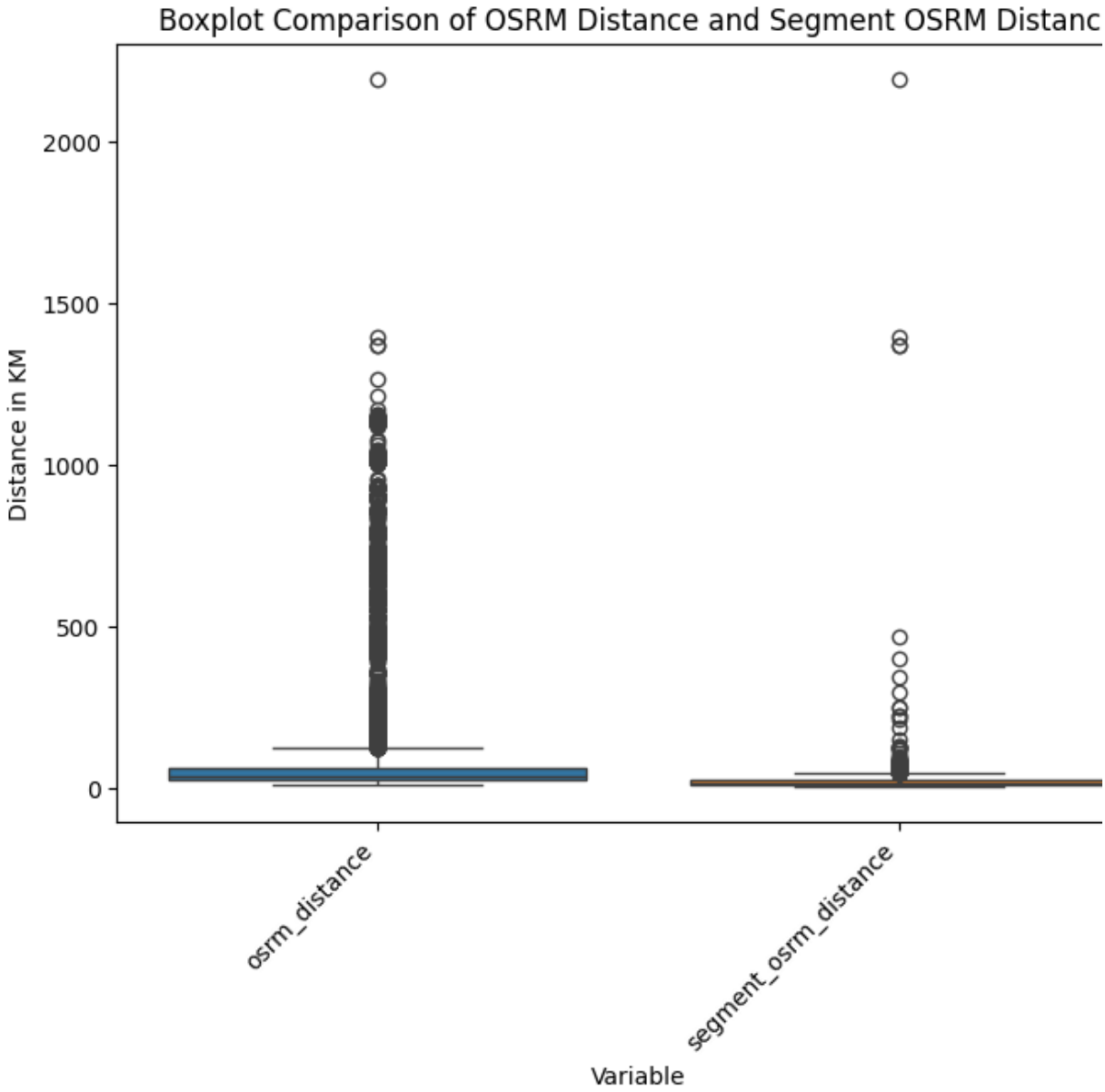
1. Visual Analysis

- Boxplot to compare distributions.
- Scatter Plot to check correlation.

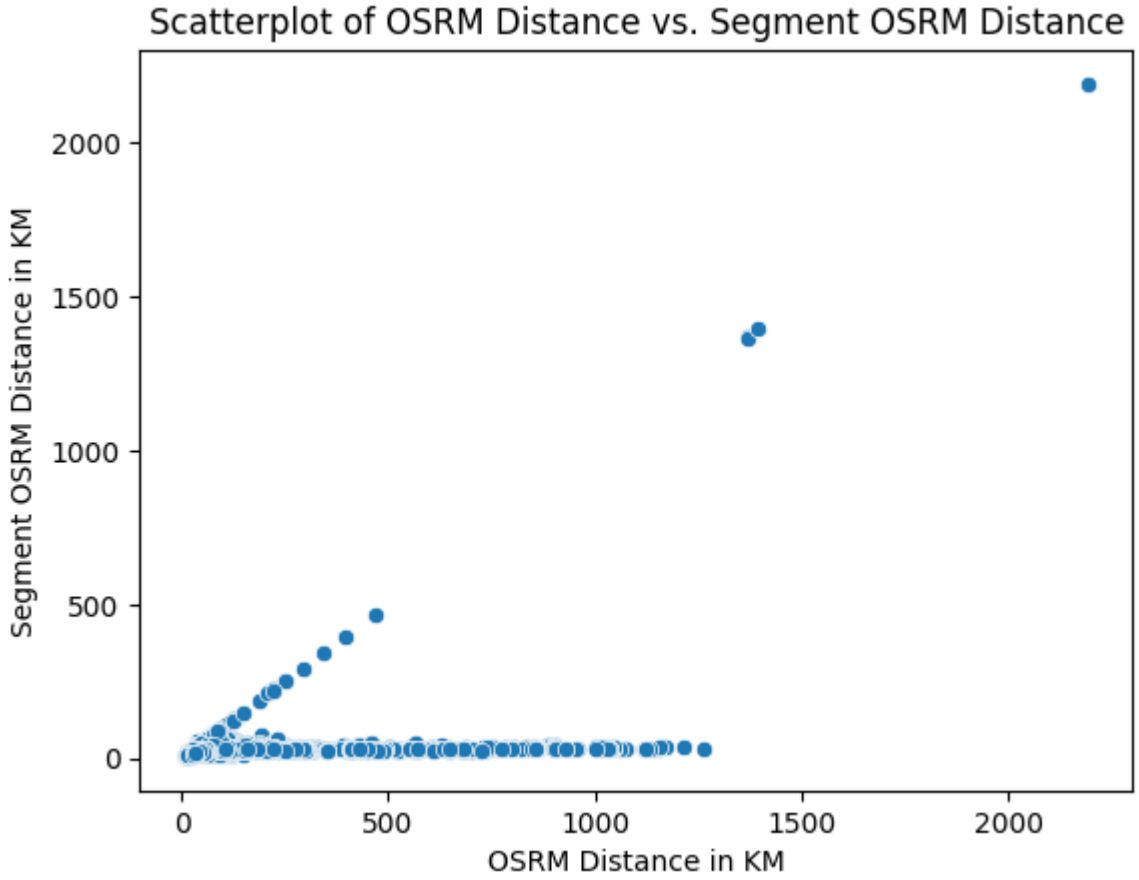
1. Hypothesis Testing

- Null Hypothesis (H_0): No significant difference between osrm_distance and segment_osrm_distance.
- Alternative Hypothesis (H_0): There is a significant difference.
- Perform a paired t-test to compare the two values

```python
# Data Preparation
# Aggregate osrm_distance and segment_osrm_distance based on trip_uuid
df_agg_distance = df1.groupby('trip_uuid')[['osrm_distance', 'segment_osrm_distance

# Remove missing values
df_agg_distance = df_agg_distance.dropna(subset=['osrm_distance', 'segment_osrm_dis

# Visual Analysis
# Boxplot to compare distributions
columns_to_plot_distance = ['osrm_distance', 'segment_osrm_distance']

# Melt the DataFrame
df_melted_distance = df_agg_distance.melt(value_vars=columns_to_plot_distance,
                                          var_name='Variable',   # Name for the new
                                          value_name='Value')      # Name for the ne

# Create the boxplot
plt.figure(figsize=(8, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', hue='Variable', data=df_melted_distance)
plt.title('Boxplot Comparison of OSRM Distance and Segment OSRM Distance')
plt.ylabel('Distance in KM')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
# Scatter Plot to check correlation
sns.scatterplot(data=df_agg_distance, x='osrm_distance', y='segment_osrm_distance')
plt.title('Scatterplot of OSRM Distance vs. Segment OSRM Distance')
plt.xlabel('OSRM Distance in KM')
plt.ylabel('Segment OSRM Distance in KM')
plt.show()
```




```
# Hypothesis Testing
from scipy.stats import ttest_rel

# Perform paired t-test
t_statistic, p_value = ttest_rel(df_agg_distance['osrm_distance'], df_agg_distance[

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

alpha = 0.05 # Significance level

if p_value < alpha:
  print("Reject the null hypothesis: There is a significant difference between osrm
else:
  print("Fail to reject the null hypothesis: There is no significant difference bet
```

```
T-statistic: 51.90069212017299
P-value: 0.0
Reject the null hypothesis: There is a significant difference between osrm_di
and segment_osrm_distance.
```

Results of Hypothesis Testing & Visual Analysis

1. Visual Analysis:

- Boxplot: Shows a noticeable difference in the distributions of osrm_distance and segment_osrm_distance.
- Scatter Plot: Displays correlation but with deviations, indicating potential differences.

1. Hypothesis Testing:

- T-statistic: 51.90
- P-value: 0.0 (extremely small, practically zero)

Conclusion: Since the p-value is < 0.05, we reject the null hypothesis (H_0). This means there is a statistically significant difference between osrm_distance and segment_osrm_distance.

Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):

1. Data Preparation

- Aggregate osrm_time and segment_osrm_time based on trip_uuid.
- Use the mean values for aggregation.
- Remove missing values.

1. Visual Analysis

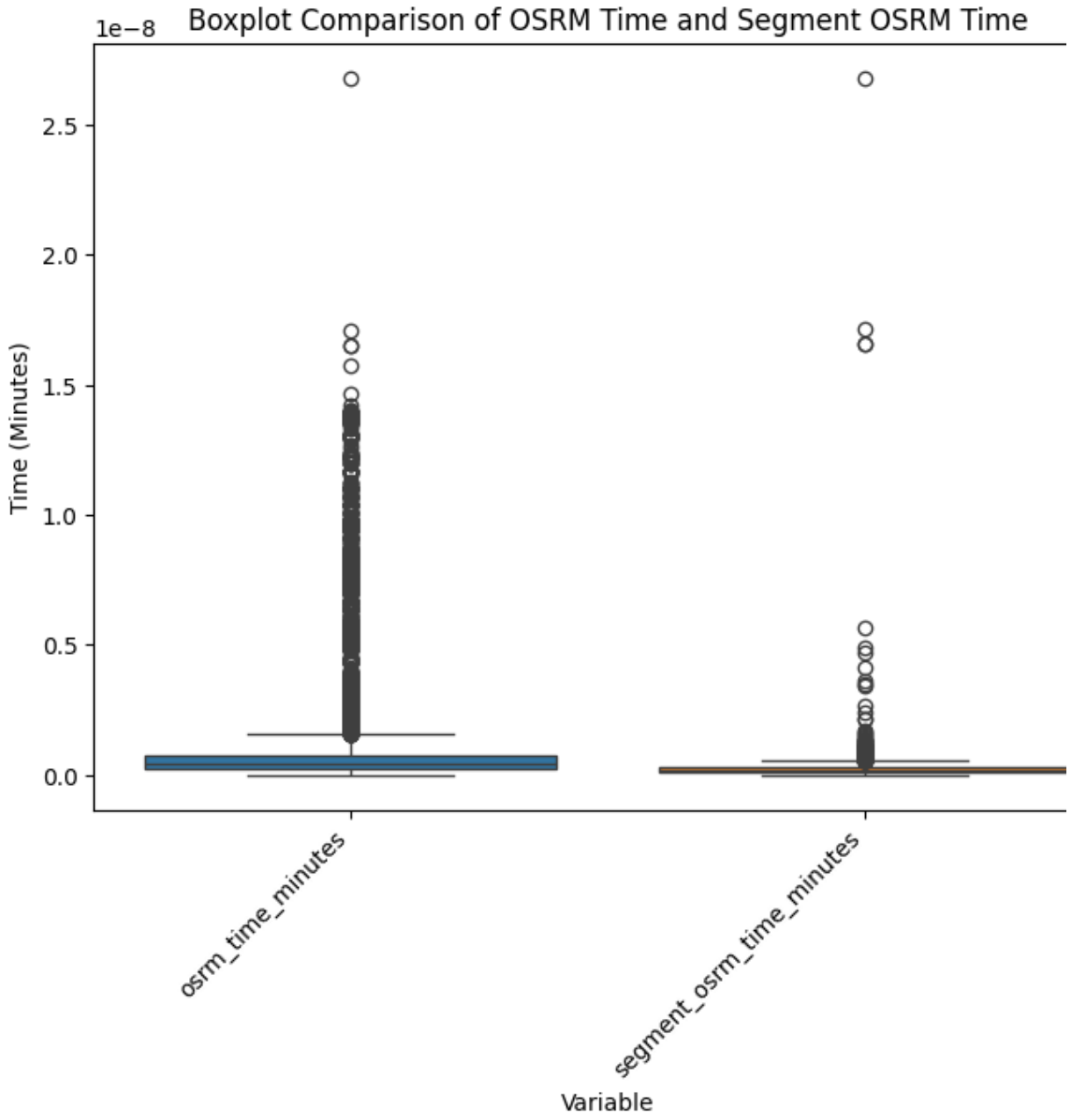
- Boxplot to compare distributions.
- Scatter Plot to check correlation.

1. Hypothesis Testing

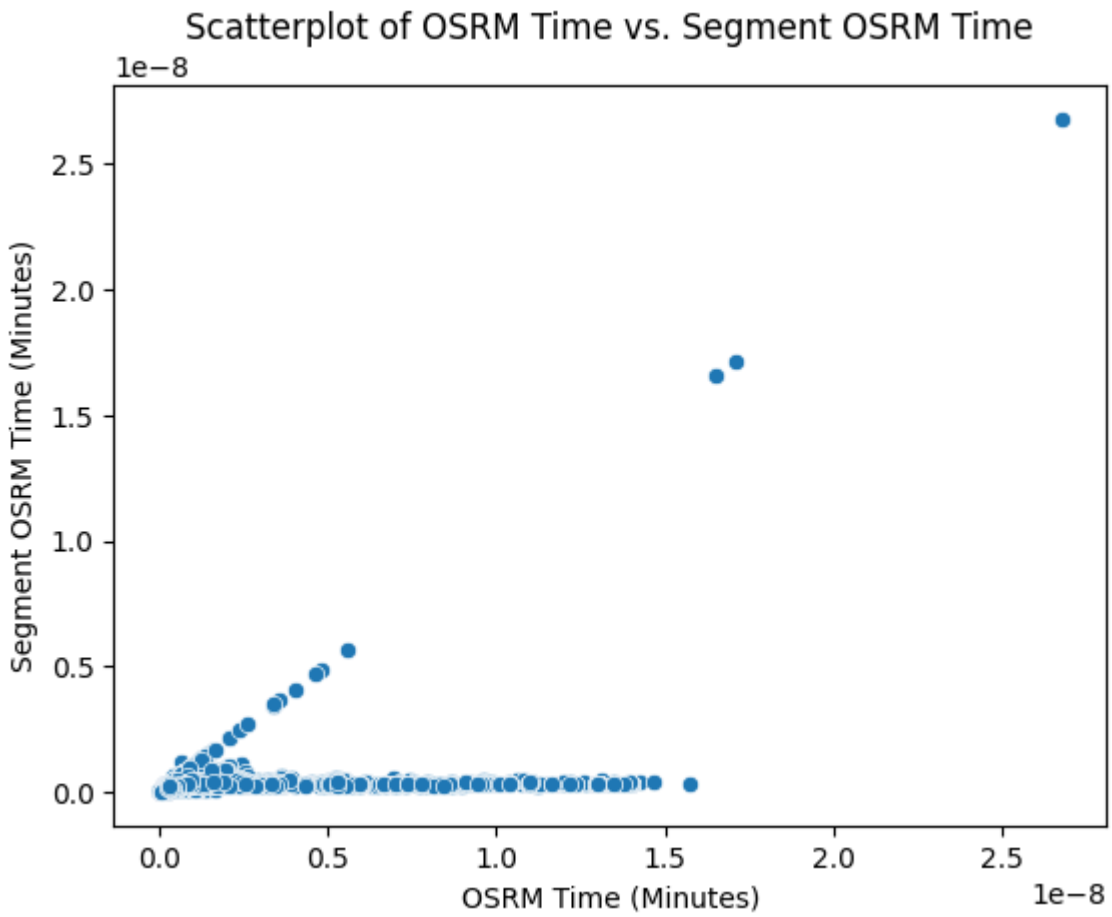
- Null Hypothesis (H_0): No significant difference between osrm_time and segment_osrm_time.
- Alternative Hypothesis (H_0): There is a significant difference.
- Perform a paired t-test to compare the two values

Now, let's run the analysis

```
# Data Preparation
# Aggregate osrm_time and segment_osrm_time based on trip_uuid
df_agg_osrm = df1.groupby('trip_uuid')[['osrm_time', 'segment_osrm_time']].mean().r

# Calculate the time difference in minutes and convert to float
# Assuming osrm_time and segment_osrm_time are datetime objects after processing
df_agg_osrm['osrm_time_minutes'] = (df_agg_osrm['osrm_time'] - df_agg_osrm['osrm_ti
df_agg_osrm['segment_osrm_time_minutes'] = (df_agg_osrm['segment_osrm_time'] - df_a

# Drop rows with NaN values in the new numerical columns
df_agg_osrm = df_agg_osrm.dropna(subset=['osrm_time_minutes', 'segment_osrm_time_mi

# Visual Analysis
# Boxplot to compare distributions
columns_to_plot_osrm = ['osrm_time_minutes', 'segment_osrm_time_minutes']

# Melt the DataFrame
df_melted_osrm = df_agg_osrm.melt(value_vars=columns_to_plot_osrm,
                                  var_name='Variable',   # Name for the new column
                                  value_name='Value')      # Name for the new colum

# Create the boxplot
plt.figure(figsize=(8, 6)) # Adjust figure size as needed
sns.boxplot(x='Variable', y='Value', hue='Variable', data=df_melted_osrm)
plt.title('Boxplot Comparison of OSRM Time and Segment OSRM Time')
plt.ylabel('Time (Minutes)')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
# Scatter Plot to check correlation
sns.scatterplot(data=df_agg_osrm, x='osrm_time_minutes', y='segment_osrm_time_minut
plt.title('Scatterplot of OSRM Time vs. Segment OSRM Time')
plt.xlabel('OSRM Time (Minutes)')
plt.ylabel('Segment OSRM Time (Minutes)')
plt.show()
```




```
# Hypothesis Testing
from scipy.stats import ttest_rel

# Perform paired t-test
t_statistic, p_value = ttest_rel(df_agg_osrm['osrm_time_minutes'], df_agg_osrm['seg

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

alpha = 0.05 # Significance level

if p_value < alpha:
  print("Reject the null hypothesis: There is a significant difference between osrm
else:
  print("Fail to reject the null hypothesis: There is no significant difference bet
```

```
T-statistic: 51.65230902091306
P-value: 0.0
Reject the null hypothesis: There is a significant difference between osrm_ti
segment_osrm_time.
```

Results of Hypothesis Testing & Visual Analysis

1. Visual Analysis:

- Boxplot: Shows a clear difference in the distributions of osrm_time and segment_osrm_time.
- Scatter Plot: Displays a correlation between the two, but with noticeable variations.

1. Hypothesis Testing:

- T-statistic: 51.65
- P-value: 0.0 (extremely small, practically zero)

Conclusion: Since the p-value is < 0.05, we reject the null hypothesis (H_0). This means there is a statistically significant difference between osrm_time and segment_osrm_time.

10. Recommendations

Delivery Performance Analysis & Actionable Insights

1. Key Findings

- There is a significant gap between actual delivery time and planned (OSRM) time.
- Segment-level travel times vary more than expected, impacting overall delivery efficiency.
- Differences in OSRM distance vs. actual traveled distance suggest possible route inefficiencies.
- Certain corridors experience higher delays than others, requiring targeted improvements.
- Estimated vs. actual delivery times do not always align, affecting customer satisfaction.

1. Visual Insights

- Boxplots highlight the variation in actual vs. estimated times.
- Scatter plots show correlations but also deviations in expected vs. real-world performance.
- High-variance corridors have been identified where adjustments can improve efficiency.

1. Actionable Recommendations

- Route Optimization
  - Use real-time traffic data to adjust delivery routes.
  - Regularly update OSRM data based on actual trip patterns.
- Improving Delivery Accuracy
  - Identify consistent delays and address underlying operational bottlenecks.
  - Train delivery personnel on efficient route planning and time management.
- Reducing Distance & Time Errors
  - Verify and correct mapping data for more precise OSRM calculations.
  - Collect feedback from drivers to refine estimated travel distances
- Enhancing Customer Experience
  - Update estimated delivery times to reflect actual trip durations.
  - Communicate realistic ETAs to customers to reduce complaints.
- Monitoring & Continuous Improvement
  - Set up KPIs for reducing delivery time variance.
  - Conduct monthly reviews of actual vs. planned delivery metrics.
  - Use machine learning models to predict potential delays and proactively address them.

1. Next Steps

- mplement real-time tracking improvements.
- Regularly audit high-variance delivery corridors.
- Train staff and improve logistics at handover points.
- Adjust customer notifications based on actual trip data

xxxTHIS is the END of the Case studyxxx