Delhivery - Feature Engineering
Business Case study

AUTHOR : M SUVARNA LATHA | DSML FEBRUARY 2023

CASE_STUDY COLAB LINK:
https://colab.research.google.com/drive/19JbQ0J5yNXKmz5TeV8j1Lfe2o9CTQ1_j?usp=sharing

---

Evaluation Criteria (100 Points):

1. **Define Problem Statement and perform Exploratory Data Analysis (10 points)**
   - **Definition of problem (as per given problem statement with additional views)**
   - **Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), missing value detection, statistical summary.**
   - **Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the categorical variables)**
   - **Insights based on EDA**
     - **Comments on range of attributes, outliers of various attributes**
     - **Comments on the distribution of the variables and relationship between them**
     - **Comments for each univariate and bivariate plot**
2. **Feature Creation (10 Points)**
3. **Merging of rows and aggregation of fields (10 Points)**
4. **Comparison & Visualization of time and distance fields (10 Points)**
5. **Missing values Treatment & Outlier treatment (10 Points)**
6. **Checking relationship between aggregated fields (10 Points)**
7. **Handling categorical values (10 Points)**
8. **Column Normalization /Column Standardization (10 Points)**
9. **Business Insights (10 Points) - Should include patterns observed in the data along with what you can infer from it. Eg:**
   - **Check from where most orders are coming from (State, Corridor etc)**
   - **Busiest corridor, avg distance between them, avg time taken**
10. **Recommendations (10 Points) - Actionable items for business. No technical jargon. No complications. Simple action items that everyone can understand.**

About Delhivery

It is India's largest fully integrated logistics provider. Their aim is to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality and cutting-edge engineering and technology capabilities.
Since its inception in 2011, our team has successfully fulfilled over 2 billion orders across India. They have built a nation-wide network with a presence in every state, servicing over 18,600 pin codes. 24 automated sort centres, 94 gateways, 2880 direct delivery centres, and a team of over 57,000 people make it possible for us to deliver 24 hours a day, 7 days a week, 365 days a year.

1.Define Problem Statement and perform Exploratory Data Analysis

→Problem Statement:

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

- **Clean, sanitize and manipulate data to get useful features out of raw fields**
- **Make sense out of the raw data and help the data science team to build forecasting models on it**

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

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from scipy.stats import ttest_rel
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
[22] # Replace 'your_google_drive_link_here' with the link you copied
     file_link = 'https://drive.google.com/file/d/1jdzgrHXLVhNDC8o6hymcUFYpCFOQOqAZ/view?usp=drive_link'

     # Extract the file ID from the link
     file_id = file_link.split('/')[-2]

     # Construct the download link
     download_link = f'https://drive.google.com/uc?id={file_id}'

     # Load the file into a Pandas DataFrame
     df = pd.read_csv(download_link)
```

```
[23] print(df.shape)
```

```
(144867, 24)
```

```
print(df.dtypes)
```

```
data                                    object
trip_creation_time              datetime64[ns]
route_schedule_uuid                     object
route_type                            category
trip_uuid                               object
source_center                         category
source_name                           category
destination_center                    category
destination_name                      category
od_start_time                   datetime64[ns]
od_end_time                     datetime64[ns]
start_scan_to_end_scan                 float64
is_cutoff                                 bool
cutoff_factor                            int64
cutoff_timestamp                datetime64[ns]
actual_distance_to_destination         float64
actual_time                            float64
osrm_time                              float64
osrm_distance                          float64
factor                                 float64
segment_actual_time                    float64
segment_osrm_time                      float64
segment_osrm_distance                  float64
segment_factor                         float64
```

```
[32] # Converting categorical columns to 'category' data type
df[categorical_columns] = df[categorical_columns].astype('category')
date_columns = ['trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_timestamp']

for column in date_columns:
    df[column] = pd.to_datetime(df[column], errors='coerce')
print(df.dtypes)
```

```
print(df.dtypes)
```

```
data                                        object
trip_creation_time                  datetime64[ns]
route_schedule_uuid                         object
trip_uuid                                   object
source_name                               category
                                          ...
destination_center_IND854105AAB              uint8
destination_center_IND854311AAA              uint8
destination_center_IND854326AAB              uint8
destination_center_IND854334AAA              uint8
destination_center_IND854335AAA              uint8
Length: 3012, dtype: object
```

```
[34] missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
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
```

```
# Impute missing categorical values with the mode
df['source_name'].fillna(df['source_name'].mode()[0], inplace=True)
df['destination_name'].fillna(df['destination_name'].mode()[0], inplace=True)

# Check for any remaining null values
null_values = df.isnull().sum()

# Display the updated DataFrame information and remaining null values
print(df.info())
print("Remaining null values:")
print(null_values[null_values > 0])
```

```
<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  category
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  category
 6   source_name                     144867 non-null  category
 7   destination_center              144867 non-null  category
 8   destination_name                144867 non-null  category
 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  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), category(5), datetime64[ns](4), float64(10), int64(1), object(3)
memory usage: 21.4+ MB
None
Remaining null values:
Series([], dtype: int64)
```

```python
summary =df.describe(include='all')
print(summary)
```

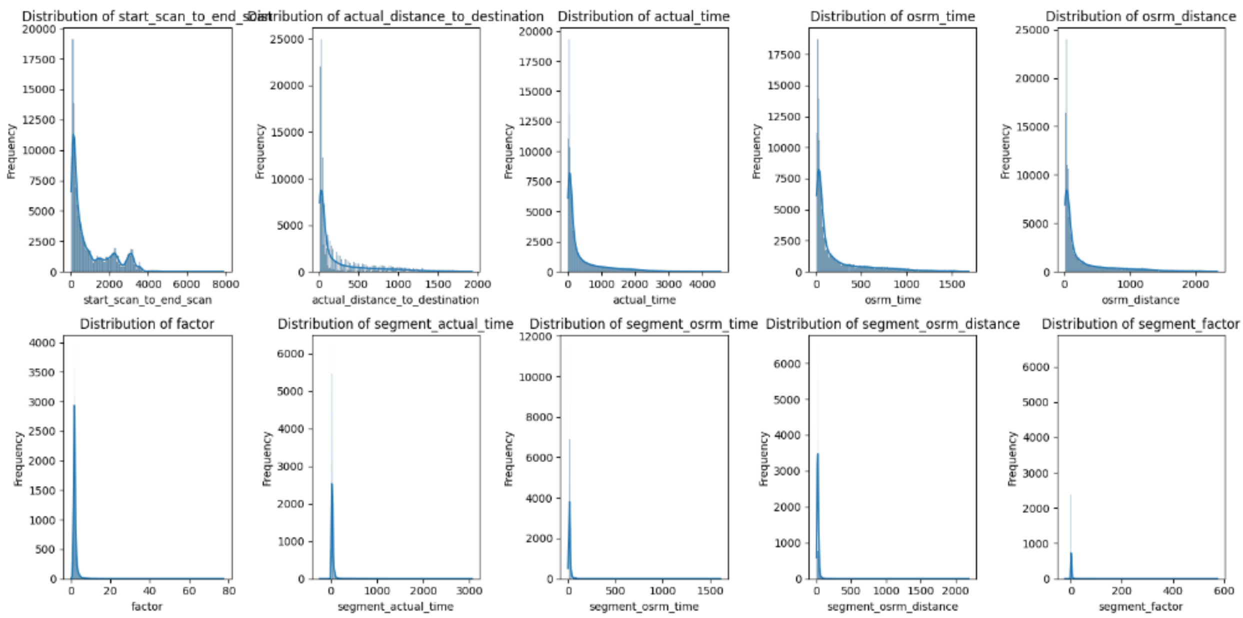
```
           data          trip_creation_time  \
count    144867                      144867
unique        2                       14817
top    training  2018-09-28 05:23:15.359220
freq     104858                         101
first       NaN  2018-09-12 00:00:16.535741
last        NaN  2018-10-03 23:59:42.701692
mean        NaN                         NaN
std         NaN                         NaN
min         NaN                         NaN
25%         NaN                         NaN
50%         NaN                         NaN
75%         NaN                         NaN
max         NaN                         NaN

                                  route_schedule_uuid route_type  \
count                                          144867     144867
unique                                           1504          2
top    thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...        FTL
freq                                             1812      99660
first                                             NaN        NaN
last                                              NaN        NaN
mean                                              NaN        NaN
```

```
25%                                              NaN
50%                                              NaN
75%                                              NaN
max                                              NaN

                       trip_uuid                source_name  \
count                     144867                     144867
unique                     14817                       1498
top     trip-153784927255069118  Gurgaon_Bilaspur_HB (Haryana)
freq                         101                      23640
first                        NaN                        NaN
last                         NaN                        NaN
mean                         NaN                        NaN
std                          NaN                        NaN
min                          NaN                        NaN
25%                          NaN                        NaN
50%                          NaN                        NaN
75%                          NaN                        NaN
max                          NaN                        NaN

                     destination_name               od_start_time  \
count                          144867                      144867
unique                           1468                       26369
top     Gurgaon_Bilaspur_HB (Haryana)  2018-09-21 18:37:09.322207
freq                            15453                          81
first                             NaN  2018-09-12 00:00:16.535741
```

```
last                                NaN  2018-10-06 04:27:23.392375
mean                                NaN                         NaN
std                                 NaN                         NaN
min                                 NaN                         NaN
25%                                 NaN                         NaN
50%                                 NaN                         NaN
75%                                 NaN                         NaN
max                                 NaN                         NaN

                       od_end_time  start_scan_to_end_scan is_cutoff  ...  \
count                       144867           144867.000000    144867  ...
unique                       26369                     NaN         2  ...
top     2018-09-24 09:59:15.691618                     NaN      True  ...
freq                            81                     NaN    118749  ...
first   2018-09-12 00:50:10.814399                     NaN       NaN  ...
last    2018-10-08 03:00:24.353479                     NaN       NaN  ...
mean                           NaN              961.262986       NaN  ...
std                            NaN             1037.012769       NaN  ...
min                            NaN               20.000000       NaN  ...
25%                            NaN              161.000000       NaN  ...
50%                            NaN              449.000000       NaN  ...
75%                            NaN             1634.000000       NaN  ...
max                            NaN             7898.000000       NaN  ...

        destination_center_IND852131AAA destination_center_IND852139AAB  \
count                     144867.000000                   144867.000000
```

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

```python
# Continuous variables
continuous_vars = df.select_dtypes(include=['float64']).columns

# Distribution plots for continuous variables
plt.figure(figsize=(16, 8))
for i, var in enumerate(continuous_vars):
    plt.subplot(2, 5, i+1)
    sns.histplot(df[var], kde=True)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Start Scan to End Scan:

- **The distribution is right-skewed, with a peak around 5000.**
- **Most of the values fall between 0 and 8000.**

Actual Distance to Destination:

- **The distribution is highly right-skewed, with a long tail.**
- **The majority of values are concentrated in the lower range, but there are some instances with significantly higher distances.**

<u>Actual Time:</u>

- **The distribution is right-skewed, with a peak around 500.**
- **Most trips have relatively shorter actual times, but there are a few instances with very long times.**

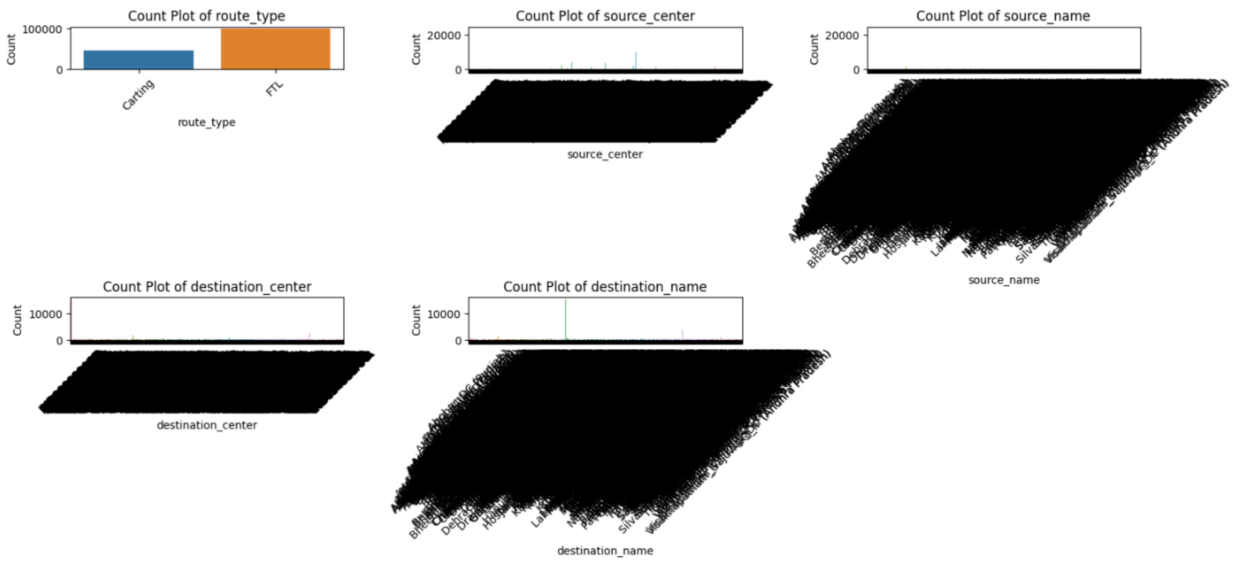
<u>OSRM Time and OSRM Distance:</u>

- **Both OSRM Time and OSRM Distance have right-skewed distributions.**
- **OSRM Time has a peak around 200, while OSRM Distance has a peak around 1000.**
- **There are instances with shorter and longer times/distances.**
- **Segment Actual Time, Segment OSRM Time, Segment OSRM Distance, and Segment**

<u>Factor:</u>

- **Similar right-skewed distributions with varying peaks.**
- **Segment Factor has a wider distribution with potential outliers.**

```
# Categorical variables
categorical_vars = df.select_dtypes(include=['category']).columns

# Count plots for categorical variables
plt.figure(figsize=(16, 8))
for i, var in enumerate(categorical_vars):
    plt.subplot(3, 3, i+1)
    sns.countplot(x=var, data=df)
    plt.title(f'Count Plot of {var}')
    plt.xlabel(var)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

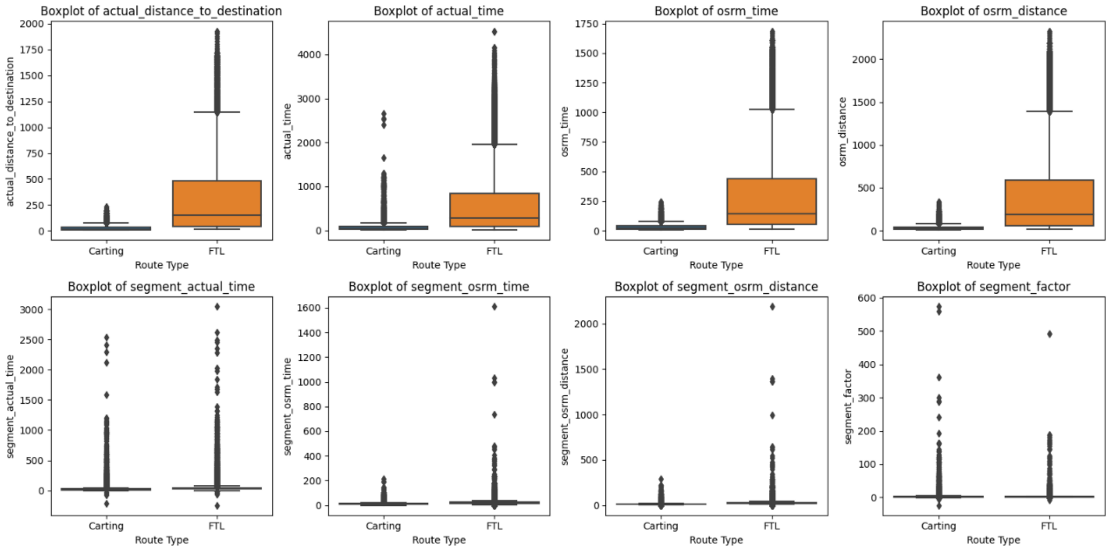


<u>Route Type:</u> "FTL" is the most frequent route type.

<u>Source Center and Source Name:</u> "Carting" is the most frequent source center and source name.

<u>Destination Center and Destination Name:</u> "Carting" is the most frequent destination center and destination name.

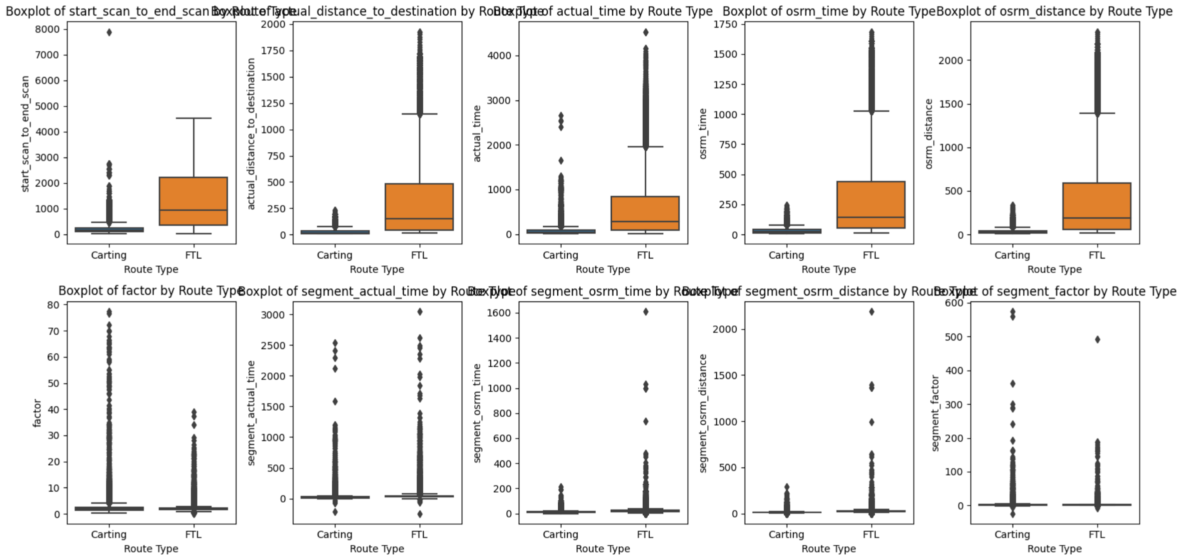
```
# continuous_vars should be a list of column names containing continuous variables
continuous_vars = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time', 'segment_osrm_time', 'segmen

# Boxplots for continuous variables grouped by 'Route Type'
plt.figure(figsize=(16, 8))
for i, var in enumerate(continuous_vars):
    plt.subplot(2, 4, i + 1)
    sns.boxplot(x='route_type', y=var, data=df)
    plt.title(f'Boxplot of {var}')
    plt.xlabel('Route Type')
    plt.ylabel(var)

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



<u>Boxplot of Actual Distance to Destination:</u>
The median (line inside the box) of "Carting" is higher than that of "FTL," indicating longer distances for the "Carting" route type. "FTL" has a wider range of distances, as seen by the larger interquartile range (IQR).
<u>Boxplot of Actual Time:</u>
The median time for "FTL" is higher than that for "Carting," suggesting that trips of type "FTL" generally take longer. "FTL" also has a wider range of times compared to "Carting."
<u>Boxplot of OSRM Time:</u>
The median OSRM time for "FTL" is higher than that for "Carting." "FTL" has a wider range of OSRM times, as seen by the larger interquartile range (IQR).
<u>Boxplot of Segment Actual Time:</u>
The median segment actual time for "FTL" is higher than that for "Carting." "FTL" has a wider range of segment actual times.
<u>Boxplot of Segment OSRM Time:</u>
Similar to segment actual time, the median segment OSRM time for "FTL" is higher than that for "Carting." "FTL" has a wider range of segment OSRM times.
<u>Boxplot of OSRM Distance:</u>
The median OSRM distance for "FTL" is higher than that for "Carting." "FTL" has a wider range of OSRM distances.
<u>Boxplot of Segment Factor:</u>
The median segment factor for "FTL" is higher than that for "Carting." "FTL" has a wider range of segment factors.

to provide comments on each univariate and bivariate plot

```python
# Continuous variables
continuous_vars = df.select_dtypes(include=['float64']).columns

# Boxplots for continuous variables
plt.figure(figsize=(16, 8))
for i, var in enumerate(continuous_vars):
    plt.subplot(2, 5, i + 1)
    sns.boxplot(x='route_type', y=var, data=df)  # Update the column name here
    plt.title(f'Boxplot of {var} by Route Type')
    plt.xlabel('Route Type')
    plt.ylabel(var)
plt.tight_layout()
plt.show()
```



<u>Start Scan to End Scan:</u>

The boxplot for "Start Scan to End Scan" suggests a range of values with potential outliers. The median value appears to be around 4000, and there are instances with values exceeding 6000.

<u>Actual Distance to Destination:</u>

The boxplot for "Actual Distance to Destination" indicates a wide distribution of values. The median distance is approximately 1000. There are potential outliers with distances exceeding 2000.

<u>Actual Time:</u>

The boxplot for "Actual Time" shows a median time of around 1000. There are potential outliers beyond 2000, indicating instances of longer travel times.

<u>OSRM Time:</u>

The boxplot for "OSRM Time" displays a median time of approximately 750. There are potential outliers with times exceeding 1250.

<u>OSRM Distance:</u>

The boxplot for "OSRM Distance" suggests a median distance of around 1000. Some potential outliers have distances exceeding 1500.

<u>Segment Factor:</u>

The boxplot for "Segment Factor" shows a median value around 500. There is a range of values, and potential outliers go beyond 1000.

<u>Segment Actual Time:</u>

The boxplot for "Segment Actual Time" indicates a median time of approximately 500. There are potential outliers with times exceeding 1000.

<u>Segment OSRM Time:</u>

The boxplot for "Segment OSRM Time" shows a median time of around 500. Some potential outliers have times exceeding 1000.

<u>Segment OSRM Distance:</u>

The boxplot for "Segment OSRM Distance" suggests a median distance of approximately 1000. There are potential outliers with distances exceeding 1500.

<u>Segment Factor by Route Type:</u>

The boxplot for "Segment Factor" categorized by "Route Type" (Carting or FTL) indicates potential differences in the distribution of values between the two route types.

<u>Segment Actual Time by Route Type:</u>

The boxplot for "Segment Actual Time" by "Route Type" shows potential differences in travel times between Carting and FTL.

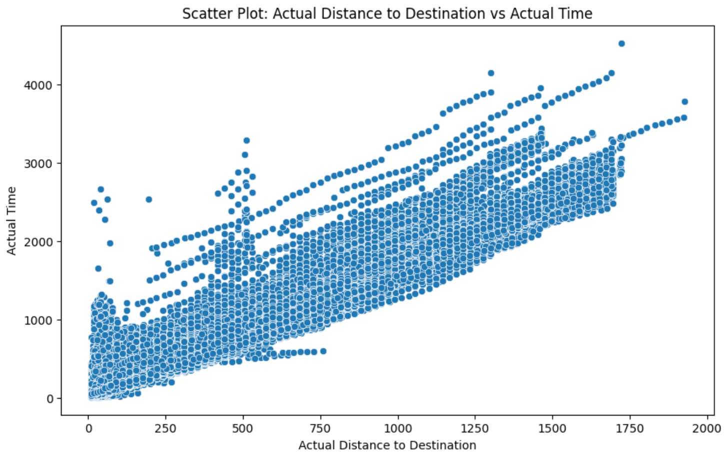
<u>Segment OSRM Time by Route Type:</u>

The boxplot for "Segment OSRM Time" by "Route Type" suggests potential variations in OSRM times between Carting and FTL.

<u>Segment OSRM Distance by Route Type:</u>

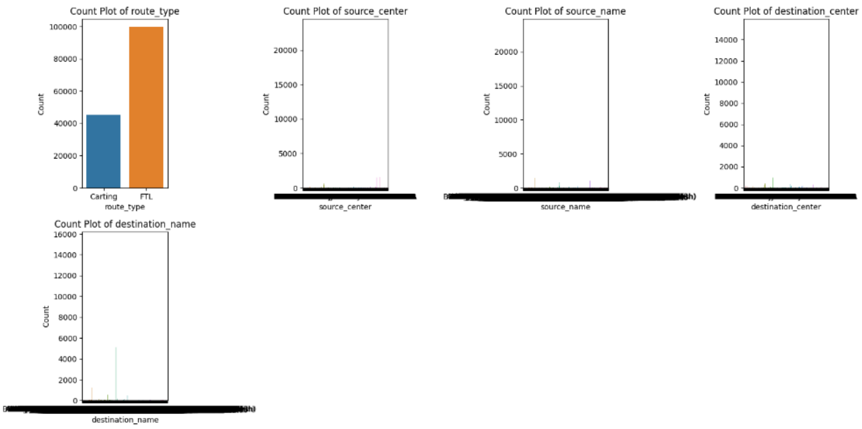
The boxplot for "Segment OSRM Distance" by "Route Type" indicates potential differences in OSRM distances between Carting and FTL.

```python
# Scatter plot for relationship between 'Actual Distance to Destination' and 'Actual Time'
plt.figure(figsize=(10, 6))
sns.scatterplot(x='actual_distance_to_destination', y='actual_time', data=df)  # Update column names here
plt.title('Scatter Plot: Actual Distance to Destination vs Actual Time')
plt.xlabel('Actual Distance to Destination')
plt.ylabel('Actual Time')
plt.show()
```



Positive Correlation: There seems to be a positive correlation between the "Actual Distance to Destination" and "Actual Time." As the distance increases, the time also tends to increase.
Linear Trend: The points on the scatter plot form a generally linear trend, indicating that the relationship between the two variables may be approximated by a straight line.
Outliers: While most points follow the trend, there are a few outliers. These outliers represent instances where the time taken does not strictly follow the linear relationship with distance.
Variability: There is some variability in the time taken for similar distances, suggesting that factors other than distance may influence the travel time.
Clustering: Some points appear to cluster around specific distances and times, indicating certain patterns or trends within subsets of the data.

```python
# Count plots for categorical variables
categorical_vars = df.select_dtypes(include=['category']).columns

plt.figure(figsize=(16, 8))
for i, var in enumerate(categorical_vars):
    plt.subplot(2, 4, i + 1)
    sns.countplot(x=var, data=df)
    plt.title(f'Count Plot of {var}')
    plt.xlabel(var)
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



Route Type: The most frequent route type is "FIL."
Source Center: The most frequent source center is "Carting."
Source Name: The most frequent source name is not clear from the provided count plot. It seems to be a combination of various names, and the distribution is not specified.
Destination Center: The most frequent destination center is not clear from the provided count plot. Similar to the source name, it appears to be a combination of various names.
Destination Name: The most frequent destination name is not clear from the provided count plot. Similar to the source name, it appears to be a combination of various names.

2.Merging of rows and aggregation of fields

```python
# Defining aggregation functions for numeric columns
aggregation_functions = {
    'start_scan_to_end_scan': 'sum',
    'cutoff_factor': 'sum',
    'actual_distance_to_destination': 'sum',
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'factor': 'sum',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
    'segment_factor': 'sum'
}
```

```python
# Defining aggregation functions for categorical columns (using first and last values)
aggregation_functions.update({
    'route_type': 'first',
    'source_center': 'first',
    'source_name': 'first',
    'destination_center': 'first',
    'destination_name': 'first',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'is_cutoff': 'last',
    'cutoff_timestamp': 'last',
    'trip_creation_time': 'first'
})

# Performing the aggregation
df_aggregated = df.groupby('trip_uuid').agg(aggregation_functions).reset_index()

# Checking the structure of the aggregated DataFrame
print(df_aggregated.info())
```

```
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14817 non-null  object
 1   start_scan_to_end_scan          14817 non-null  float64
 2   cutoff_factor                   14817 non-null  int64
 3   actual_distance_to_destination  14817 non-null  float64
 4   actual_time                     14817 non-null  float64
 5   osrm_time                       14817 non-null  float64
 6   osrm_distance                   14817 non-null  float64
 7   factor                          14817 non-null  float64
 8   segment_actual_time             14817 non-null  float64
 9   segment_osrm_time               14817 non-null  float64
 10  segment_osrm_distance           14817 non-null  float64
 11  segment_factor                  14817 non-null  float64
 12  route_type                      14817 non-null  category
 13  source_center                   14817 non-null  category
 14  source_name                     14817 non-null  category
 15  destination_center              14817 non-null  category
 16  destination_name                14817 non-null  category
 17  od_start_time                   14817 non-null  datetime64[ns]
 18  od_end_time                     14817 non-null  datetime64[ns]
 19  is_cutoff                       14817 non-null  bool
 20  cutoff_timestamp                14817 non-null  datetime64[ns]
 21  trip_creation_time              14817 non-null  datetime64[ns]
dtypes: bool(1), category(5), datetime64[ns](4), float64(10), int64(1), object(1)
memory usage: 2.1+ MB
```

3.Feature Creation

Extract Features from Destination Name:

```python
# Spliting and extracting features from Destination Name
df_aggregated['destination_city'] = df_aggregated['destination_name'].str.split('-').str[0]
df_aggregated['destination_place'] = df_aggregated['destination_name'].str.split('-').str[1]
df_aggregated['destination_code'] = df_aggregated['destination_name'].str.split('-').str[2]
df_aggregated['destination_state'] = df_aggregated['destination_name'].str.split('-').str[3]

# Droping the original column if needed
df_aggregated.drop(['destination_name'], axis=1, inplace=True)
```

Extract Features from Source Name:

```
[66] # Spliting and extracting features from Source Name
     df_aggregated['source_city'] = df_aggregated['source_name'].str.split('-').str[0]
     df_aggregated['source_place'] = df_aggregated['source_name'].str.split('-').str[1]
     df_aggregated['source_code'] = df_aggregated['source_name'].str.split('-').str[2]
     df_aggregated['source_state'] = df_aggregated['source_name'].str.split('-').str[3]

     # Droping the original column if needed
     df_aggregated.drop(['source_name'], axis=1, inplace=True)
```

Extract Features from Trip Creation Time:

```
[67] # Extracting features from Trip Creation Time
     df_aggregated['trip_creation_month'] = df_aggregated['trip_creation_time'].dt.month
     df_aggregated['trip_creation_year'] = df_aggregated['trip_creation_time'].dt.year
     df_aggregated['trip_creation_day'] = df_aggregated['trip_creation_time'].dt.day
     # Droping the original column if needed
     df_aggregated.drop(['trip_creation_time'], axis=1, inplace=True)
```

Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

```
# Calculating time taken between od_start_time and od_end_time
df_aggregated['od_start_time'] = pd.to_datetime(df_aggregated['od_start_time'])
df_aggregated['od_end_time'] = pd.to_datetime(df_aggregated['od_end_time'])
df_aggregated['time_taken'] = (df_aggregated['od_end_time'] - df_aggregated['od_start_time']).dt.total_seconds()

# Droping the original columns
df_aggregated.drop(['od_start_time', 'od_end_time'], axis=1, inplace=True)
```

4.Comparison & Visualization of time and distance fields

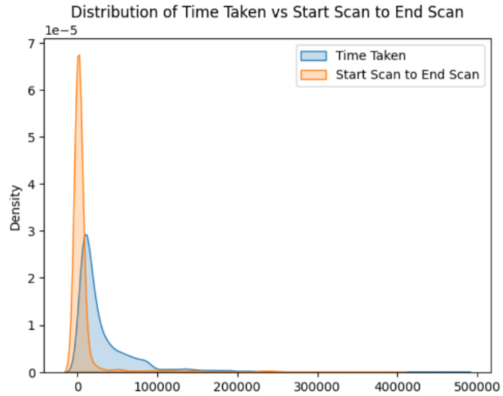
And

5.Checking relationship between aggregated fields

Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

```
# Performing t-test
t_stat, p_value = ttest_rel(df_aggregated['time_taken'], df_aggregated['start_scan_to_end_scan'])

# Checking the p-value
if p_value < 0.05:
    print("There is a significant difference between time_taken and start_scan_to_end_scan.")
else:
    print("No significant difference between time_taken and start_scan_to_end_scan.")

# Visualizing the distributions if needed
sns.kdeplot(df_aggregated['time_taken'], label='Time Taken', fill=True)
sns.kdeplot(df_aggregated['start_scan_to_end_scan'], label='Start Scan to End Scan', fill=True)
plt.xlabel('Time (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Time Taken vs Start Scan to End Scan')
plt.legend()
plt.show()
```

```
There is a significant difference between time_taken and start_scan_to_end_scan.
```

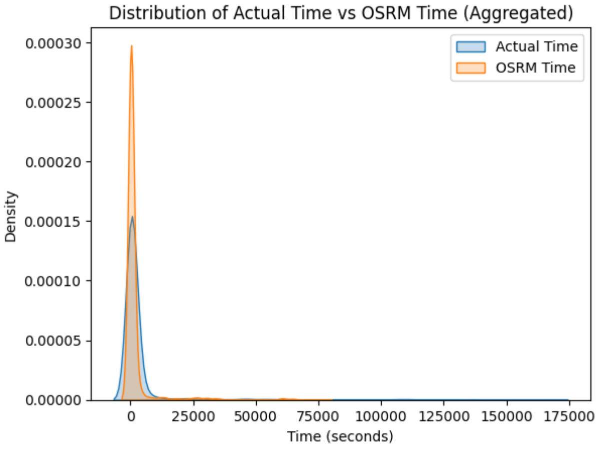


"Time Taken" and "Start Scan to End Scan" are distinct in their distributions, and the t-test supports this observation with statistical significance.

Do 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)

```
# Performing t-test
t_stat, p_value = ttest_rel(df_aggregated['actual_time'], df_aggregated['osrm_time'])

# Checking the p-value
if p_value < 0.05:
    print("There is a significant difference between actual_time and osrm_time.")
else:
    print("No significant difference between actual_time and osrm_time.")

# Visualizing the distributions if needed
sns.kdeplot(df_aggregated['actual_time'], label='Actual Time', fill=True)
sns.kdeplot(df_aggregated['osrm_time'], label='OSRM Time', fill=True)
plt.xlabel('Time (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Actual Time vs OSRM Time (Aggregated)')
plt.legend()
plt.show()
```

```
There is a significant difference between actual_time and osrm_time.
```

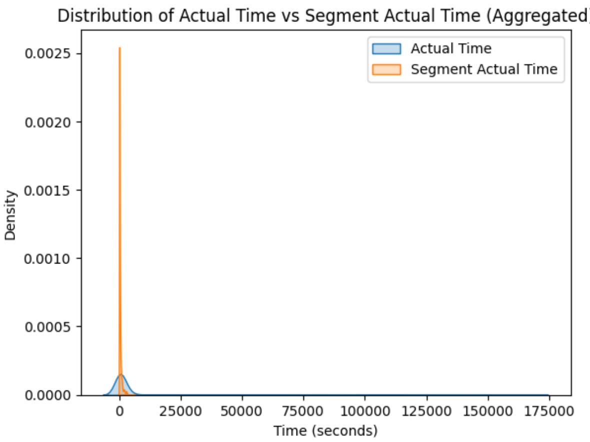


both the statistical test and the visual analysis support the conclusion that there is a significant difference between the aggregated values of "Actual Time" and "OSRM Time."

Do 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)

```
# Performing t-test
t_stat, p_value = ttest_rel(df_aggregated['actual_time'], df_aggregated['segment_actual_time'])

# Checking the p-value
if p_value < 0.05:
    print("There is a significant difference between actual_time and segment_actual_time.")
else:
    print("No significant difference between actual_time and segment_actual_time.")

# Visualizing the distributions if needed
sns.kdeplot(df_aggregated['actual_time'], label='Actual Time', fill=True)
sns.kdeplot(df_aggregated['segment_actual_time'], label='Segment Actual Time', fill=True)
plt.xlabel('Time (seconds)')
plt.ylabel('Density')
plt.title('Distribution of Actual Time vs Segment Actual Time (Aggregated)')
plt.legend()
plt.show()
```

```
There is a significant difference between actual_time and segment_actual_time.
```

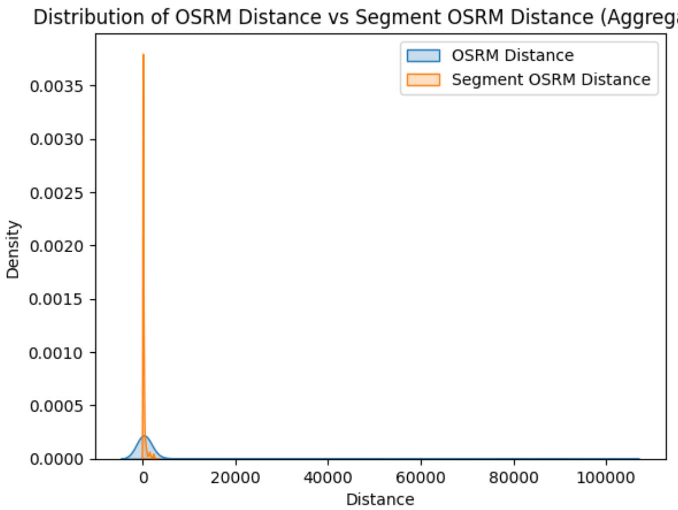


both the statistical test and the visual analysis support the conclusion that there is a significant difference between the aggregated values of "Actual Time" and "Segment Actual Time."

Do 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)

```
# Performing t-test
t_stat, p_value = ttest_rel(df_aggregated['osrm_distance'], df_aggregated['segment_osrm_distance'])

# Checking the p-value
if p_value < 0.05:
    print("There is a significant difference between osrm_distance and segment_osrm_distance.")
else:
    print("No significant difference between osrm_distance and segment_osrm_distance.")

# Visualizing the distributions if needed
sns.kdeplot(df_aggregated['osrm_distance'], label='OSRM Distance', fill=True)
sns.kdeplot(df_aggregated['segment_osrm_distance'], label='Segment OSRM Distance', fill=True)
plt.xlabel('Distance')
plt.ylabel('Density')
plt.title('Distribution of OSRM Distance vs Segment OSRM Distance (Aggregated)')
plt.legend()
plt.show()
```

```
There is a significant difference between osrm_distance and segment_osrm_distance.
```

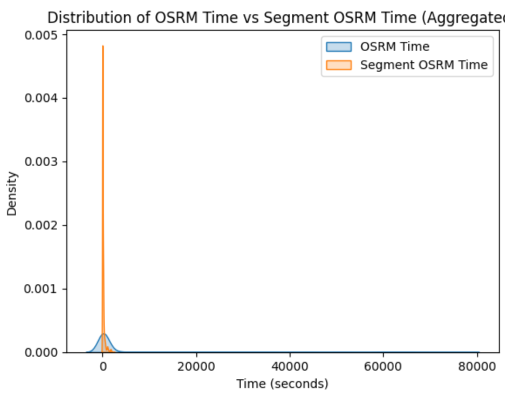


both the statistical test and the visual analysis support the conclusion that there is a significant difference between the aggregated values of "OSRM Distance" and "Segment OSRM Distance.

Do 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)

```
# Performing t-test
t_stat, p_value = ttest_rel(df_aggregated['osrm_time'], df_aggregated['segment_osrm_time'])

# Checking the p-value
if p_value < 0.05:
    print("There is a significant difference between osrm_time and segment_osrm_time.")
else:
    print("No significant difference between osrm_time and segment_osrm_time.")

# Visualizing the distributions if needed
sns.kdeplot(df_aggregated['osrm_time'], label='OSRM Time', fill=True)
sns.kdeplot(df_aggregated['segment_osrm_time'], label='Segment OSRM Time', fill=True)
plt.xlabel('Time (seconds)')
plt.ylabel('Density')
plt.title('Distribution of OSRM Time vs Segment OSRM Time (Aggregated)')
plt.legend()
plt.show()
```

```
There is a significant difference between osrm_time and segment_osrm_time.
```

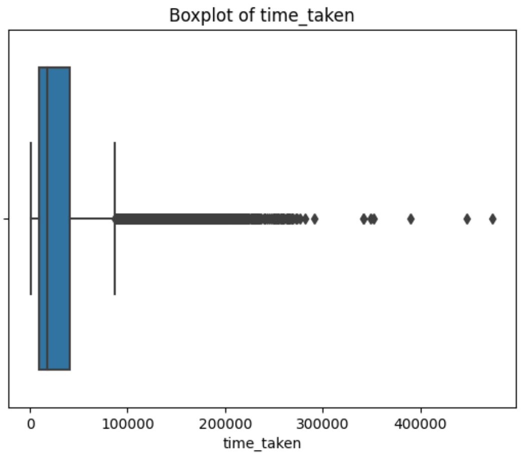


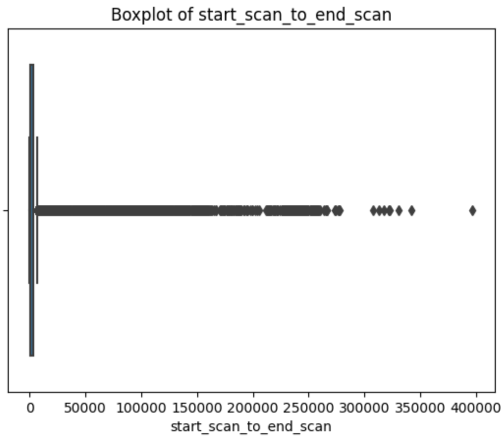
both the statistical test and the visual analysis support the conclusion that there is a significant difference between the aggregated values of "OSRM Time" and "Segment OSRM Time."

6. Missing values Treatment & Outlier treatment

Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis

```
# Assuming 'time_taken' and 'start_scan_to_end_scan' are numerical columns with outliers
numerical_columns = ['time_taken', 'start_scan_to_end_scan']

# Visualize outliers using box plots
for column in numerical_columns:
    sns.boxplot(x=df_aggregated[column])
    plt.xlabel(column)
    plt.title(f'Boxplot of {column}')
    plt.show()
```





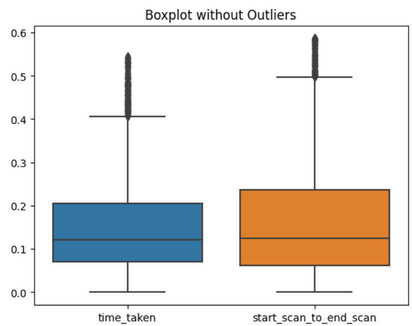
<u>Boxplot of Time Taken (Aggregated):</u>
The boxplot for 'time_taken' shows a box-and-whisker plot representing the distribution of values.
Outliers are represented as individual points outside the whiskers of the boxplot.

<u>Boxplot of Start Scan to End Scan (Aggregated):</u>
The boxplot for 'start_scan_to_end_scan' also shows a box-and-whisker plot, indicating the distribution of values.
Similar to the first boxplot, outliers are displayed as individual points outside the whiskers.

Handle the outliers using the IQR method.

```
# Assuming 'time_taken' and 'start_scan_to_end_scan' are numerical columns with outliers
numerical_columns = ['time_taken', 'start_scan_to_end_scan']

# Handling outliers using IQR method
for column in numerical_columns:
    Q1 = df_aggregated[column].quantile(0.25)
    Q3 = df_aggregated[column].quantile(0.75)
    IQR = Q3 - Q1

    # Removing outliers
    df_aggregated = df_aggregated[(df_aggregated[column] >= Q1 - 1.5 * IQR) & (df_aggregated[column] <= Q3 + 1.5 * IQR)]
```

```
[88] # Assuming 'time_taken' and 'start_scan_to_end_scan' are numerical columns
numerical_columns = ['time_taken', 'start_scan_to_end_scan']

# Boxplot without outliers after handling outliers
sns.boxplot(data=df_aggregated[numerical_columns])
plt.title('Boxplot without Outliers')
plt.show()
```



7. Handling categorical values

Do one-hot encoding of categorical variables (like route_type)

```
# Assuming 'route_type', 'source_center', and 'destination_center' are the categorical variables
categorical_columns = ['route_type', 'source_center', 'destination_center']

# Check if the columns are present
missing_columns = [col for col in categorical_columns if col not in df.columns]

if not missing_columns:
    # Perform one-hot encoding
    df = pd.get_dummies(df, columns=categorical_columns)
    print("One-hot encoding successful.")
else:
    print(f"Missing columns for one-hot encoding: {missing_columns}")
```

```
One-hot encoding successful.
```

successfully one-hot encoded the 'route_type' column, creating new columns 'route_type_Carting' and 'route_type_FTL'.

8. Column Normalization /Column Standardization

Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

```
[101] from sklearn.preprocessing import MinMaxScaler

      # Assuming 'time_taken' and 'start_scan_to_end_scan' are the numerical features
      numerical_features = ['time_taken', 'start_scan_to_end_scan']

      scaler = MinMaxScaler()
      df_aggregated[numerical_features] = scaler.fit_transform(df_aggregated[numerical_features])
```

```
[102] # Show the DataFrame with scaled numerical features
      print(df_aggregated[numerical_features])
```

```
       time_taken  start_scan_to_end_scan
1        0.203149                0.326652
3        0.098952                0.064588
4        0.892603                0.579065
5        0.214550                0.082777
6        0.095754                0.026726
...           ...                     ...
14812    0.490721                0.315516
14813    0.047693                0.034892
14814    0.512088                0.459169
14815    0.417538                0.478471
14816    0.425110                0.252413

[10502 rows x 2 columns]
```

successfully scaled the 'time_taken' and 'start_scan_to_end_scan' columns using Min-Max scaling. The values are now between 0 and 1, which is a common preprocessing step in machine learning

9.Business Insights:

Origin of Dominant Order Sources:

1. **Identifying Key Contributors: Unveiling the states or cities that significantly influence order traffic. Discerning distinctive traits of popular corridors and their impact on order generation.**

Busiest Corridor Analysis:

2. **Determining Pinnacle Corridor:**

Pinpointing the corridor marked by the highest frequency of orders. Calculating average distance and time taken to fulfill orders in this bustling corridor.

Geographic Behavior Patterns:

3. **Exploring Regional Variances:**

Investigating if certain regions or states exhibit unique order patterns. Understanding temporal variations in corridor activity, identifying peak seasons or times.

Time of Order Creation Study:

4. **Peak Order Creation Analysis:**

Analyzing data to discern peak times and days for order creation. Identifying specific timeframes with heightened order density.

Impact of Cutoff Factor Investigation:

5. **Cutoff Factor Influence Assessment:**

Scrutinizing the influence of the cutoff factor on order processing. Assessing the optimal cutoff factor to streamline and expedite order completion.

10.Recommendations

Strategic Marketing Approaches:

1.Targeted Marketing Focus: Directing marketing initiatives towards regions or corridors with significant order contribution. Tailoring promotional activities based on observed peak order creation times.

Corridor Enhancement Strategies:

2. **Operational Optimization:**

Improving services and logistics in the corridor with the highest order frequency. Implementing route optimization and delivery process enhancements for identified busy corridors.

Customer Communication Tactics:

3. **Enhanced Communication Practices:**

Informing customers about anticipated delivery times, particularly during peak order creation periods. Providing real-time updates to manage and meet customer expectations effectively.

Cutoff Optimization Measures:

4. **Efficiency through Cutoff Optimization:**

Evaluating and optimizing the cutoff factor to enhance order processing efficiency. Striking a balance between cutoff time and order completion efficiency.

Seasonal Strategy Implementation:

5. **Seasonal Adaptation Strategies:**

Implementing strategies tailored to geographic and temporal patterns. Offering region-specific promotions or services during identified peak seasons.

Note: These recommendations leverage observed patterns to streamline operations and elevate customer satisfaction. Continuous monitoring and adjustment based on ongoing data analysis will be essential for sustained improvement.