```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
```

Problem Statement:

Delhivery, India's leading logistics player, seeks to optimize its operations by leveraging data-driven insights. The company aims to clean, manipulate, and analyze its data to extract valuable features, detect outliers, and perform hypothesis testing. The primary objective is to improve operational efficiency and enhance customer satisfaction through informed decision-making.

Exploratory Data Analysis (EDA):

The EDA process involves handling missing values, profiling the dataset, creating new features, grouping and aggregating data, detecting outliers, and performing hypothesis testing. Visualization and interpretation of the data will provide actionable insights for optimizing logistics operations and driving business growth.

Conclusion:

Delhivery's data analysis aims to extract actionable insights to optimize logistics operations and improve customer satisfaction. Through effective data cleaning, manipulation, and analysis, the company can make informed decisions and drive business growth.

```python
df = pd.read_csv("delhivery_data.csv")
df.head(1)
```

```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840   

                                 route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting   

                trip_uuid source_center                 
source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)   


  destination_center               destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)   

                od_start_time  ...     cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  ...  2018-09-20 04:27:55   

   actual_distance_to_destination  actual_time  osrm_time 
osrm_distance  \
0                        10.43566         14.0       11.0 
11.9653  
```

```
     factor  segment_actual_time  segment_osrm_time
segment_osrm_distance  \
0  1.272727                 14.0               11.0
11.9653

   segment_factor
0        1.272727

[1 rows x 24 columns]
```

```
df.rename(columns={'factor': 'factor_ratio', 'segment_factor':
'segment_factor_ratio'}, inplace=True)
```

Converting time columns into pandas datetime.

```
df['od_start_time'] = pd.to_datetime(df['od_start_time'])
df['od_end_time'] = pd.to_datetime(df['od_end_time'])
df['cutoff_timestamp'] = pd.to_datetime(df['cutoff_timestamp'])
df['trip_creation_time'] = pd.to_datetime(df['cutoff_timestamp'])
```

Source_name and destination_name have an unequal number of data counts, let do more analysis below.

Filling Missing Values using Mode

Given the discrepancy in the number of non-null values between the source_name and destination_name columns, we have 144574 non-null values for source_name and 144606 non-null values for destination_name out of 144867 total entries. With such a small number of missing values compared to the total dataset size, imputation may be a reasonable approach to handle the missing values in this case. Let's proceed with imputing the missing values using the mode (most frequent value) of each respective colum-

```
df.shape
```

```
(144867, 24)
```

```
print(len(df['destination_name'].unique()))
print(len(df['source_name'].unique()))
```

```
1469
1499
```

```
source_name_mode = df['source_name'].mode()[0]
df['source_name'].fillna(source_name_mode, inplace = True)

destination_name_mode = df['destination_name'].mode()[0]
df['destination_name'].fillna(destination_name_mode, inplace = True)
```

```
cat_col = [col for col in df.columns if df[col].dtype == 'object']
print("CATEGORICAL COLUMNS= ", cat_col)
print("TOTAL = ",len(cat_col), end="\n\n")
num_col = [col for col in df.columns if df[col].dtype != 'object']
print("NUMERICAl COLUMNS= ", num_col)
print("TOTAL = ",len(num_col))
```

```
CATEGORICAL COLUMNS=  ['data', 'route_schedule_uuid', 'route_type',
'trip_uuid', 'source_center', 'source_name', 'destination_center',
'destination_name']
TOTAL =  8

NUMERICAl COLUMNS=  ['trip_creation_time', 'od_start_time',
'od_end_time', 'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
'osrm_time', 'osrm_distance', 'factor_ratio', 'segment_actual_time',
'segment_osrm_time', 'segment_osrm_distance', 'segment_factor_ratio']
TOTAL =  16
```

After filling the null values with the mode, our dataset now contains no null values, allowing us to proceed with further analysis.

Grouping by segment

Segmenting the data set based on trip_uuid, source_center & destination_center

```
df['trip_segment'] = df['trip_uuid'].astype(str) + '_' +
df['source_center'] + '_' + df['destination_center']

# Calculate cumulative sums for the segments
df['segment_actual_time_sum'] = df.groupby('trip_segment')
['segment_actual_time'].cumsum()
df['segment_osrm_distance_sum'] = df.groupby('trip_segment')
['segment_osrm_distance'].cumsum()
df['segment_osrm_time_sum'] = df.groupby('trip_segment')
['segment_osrm_time'].cumsum()
```

Aggregating at segment level

Aggregating the dataset based on the segment created above "trip_segment". Then we will create a dictionary based on the essential columns in the dataset.

```
# Create the aggregation dictionary
create_segment_dict = {
    'trip_uuid' : 'first',
    'od_start_time': 'first',  # Keep the first od_start_time
    'od_end_time': 'last',  # Keep the last od_end_time
    'osrm_time': 'sum',  # Sum of osrm_time
```

```
    'osrm_distance': 'sum',  # Sum of osrm_distance
    'segment_actual_time_sum': 'last',  # Sum of segment_actual_time
    'segment_osrm_time_sum': 'last',  # Sum of segment_osrm_time
    'segment_osrm_distance_sum':'last',
    'destination_name' : 'first',
    'source_name' : 'first',
    'trip_creation_time' : 'first',
    "route_type" : 'first',
    'actual_time' : sum,
      'segment_actual_time' :sum,
    'segment_osrm_distance' : sum,
    'segment_osrm_time' : sum,
}

# Group by the segment key and aggregate using the dictionary
aggregated_df =
df.groupby('trip_segment').agg(create_segment_dict).reset_index()

# Sort the resulting DataFrame
aggregated_df = aggregated_df.sort_values(by=['trip_segment',
'od_end_time'], ascending=[False, True]).reset_index(drop=True)

# Display the resulting DataFrameaggregated_df
aggregated_df.head(2)

                                        trip_segment
trip_uuid  \
0  trip-153861118270144424_IND583201AAA_IND583119AAA  trip-
153861118270144424
1  trip-153861118270144424_IND583119AAA_IND583101AAA  trip-
153861118270144424

               od_start_time                od_end_time  osrm_time  \
0 2018-10-04 02:51:44.712656 2018-10-04 03:58:40.726547       47.0
1 2018-10-04 03:58:40.726547 2018-10-04 08:46:09.166940       59.0

   osrm_distance  segment_actual_time_sum  segment_osrm_time_sum  \
0        51.2851                     41.0                   25.0
1        76.5169                    233.0                   42.0

   segment_osrm_distance_sum               destination_name  \
0                    28.0484  Sandur_WrdN1DPP_D (Karnataka)
1                    52.5303         Bellary_Dc (Karnataka)

                     source_name  trip_creation_time route_type
actual_time  \
0             Hospet (Karnataka) 2018-10-04 03:20:29        FTL
72.0
1  Sandur_WrdN1DPP_D (Karnataka) 2018-10-04 07:29:32        FTL
278.0
```

```
   segment_actual_time  segment_osrm_distance  segment_osrm_time
0                 41.0                28.0484               25.0
1                233.0                52.5303               42.0
```

Extracting features

```
#Calculating the od start & end time difference and rounding it off
for user readability
aggregated_df['od_time_diff_hour'] =
np.round((aggregated_df['od_end_time'] -
aggregated_df['od_start_time']).dt.total_seconds()/3600,3)
aggregated_df['od_time_diff_hour']

#Dropping the od time columns
aggregated_df.drop(['od_end_time', 'od_start_time'], axis=1)
aggregated_df.head(2)
```

```
                                        trip_segment
trip_uuid  \
0  trip-153861118270144424_IND583201AAA_IND583119AAA  trip-
153861118270144424
1  trip-153861118270144424_IND583119AAA_IND583101AAA  trip-
153861118270144424

               od_start_time                od_end_time  osrm_time  \
0 2018-10-04 02:51:44.712656 2018-10-04 03:58:40.726547       47.0
1 2018-10-04 03:58:40.726547 2018-10-04 08:46:09.166940       59.0

   osrm_distance  segment_actual_time_sum  segment_osrm_time_sum  \
0        51.2851                     41.0                   25.0
1        76.5169                    233.0                   42.0

   segment_osrm_distance_sum               destination_name  \
0                    28.0484  Sandur_WrdN1DPP_D (Karnataka)
1                    52.5303         Bellary_Dc (Karnataka)

                     source_name  trip_creation_time route_type
actual_time  \
0            Hospet (Karnataka) 2018-10-04 03:20:29        FTL
72.0
1  Sandur_WrdN1DPP_D (Karnataka) 2018-10-04 07:29:32        FTL
278.0

   segment_actual_time  segment_osrm_distance  segment_osrm_time  \
0                 41.0                28.0484               25.0
1                233.0                52.5303               42.0

   od_time_diff_hour
```

```
0              1.116
1              4.791

#Splitting the destination name and extracting the place_code and city
aggregated_df['destination_place_code'] =
aggregated_df['destination_name'].str.split(' ', expand=True)[0]
aggregated_df['destination_city'] =
aggregated_df['destination_name'].str.split(' ', expand=True)[1]

#Splitting the source name and extracting the place_code and city
aggregated_df['source_place_code'] =
aggregated_df['source_name'].str.split(' ', expand=True)[0]
aggregated_df['source_city'] =
aggregated_df['source_name'].str.split(' ', expand=True)[1]

#Removing the () from the city name
aggregated_df['destination_city'] =
aggregated_df['destination_city'].replace(r'[^a-zA-Z0-9\s]', '',
regex=True)
aggregated_df['source_city'] =
aggregated_df['source_city'].replace(r'[^a-zA-Z0-9\s]', '',
regex=True)
aggregated_df.head()

#Dropping the destination & source name
aggregated_df.drop(['destination_city', 'source_city'], axis=1)

aggregated_df.head(2)

                                        trip_segment
trip_uuid  \
0  trip-153861118270144424_IND583201AAA_IND583119AAA  trip-
153861118270144424
1  trip-153861118270144424_IND583119AAA_IND583101AAA  trip-
153861118270144424

               od_start_time               od_end_time  osrm_time  \
0 2018-10-04 02:51:44.712656 2018-10-04 03:58:40.726547       47.0
1 2018-10-04 03:58:40.726547 2018-10-04 08:46:09.166940       59.0

   osrm_distance  segment_actual_time_sum  segment_osrm_time_sum  \
0        51.2851                     41.0                   25.0
1        76.5169                    233.0                   42.0

   segment_osrm_distance_sum              destination_name  ...
route_type  \
0                    28.0484  Sandur_WrdN1DPP_D (Karnataka)  ...
FTL
1                    52.5303         Bellary_Dc (Karnataka)  ...
FTL
```

```
  actual_time segment_actual_time  segment_osrm_distance 
segment_osrm_time  \
0        72.0                 41.0                28.0484 
25.0
1       278.0                233.0                52.5303 
42.0

   od_time_diff_hour  destination_place_code  destination_city  \
0              1.116       Sandur_WrdN1DPP_D         Karnataka   
1              4.791              Bellary_Dc         Karnataka   

   source_place_code source_city  
0             Hospet   Karnataka  
1  Sandur_WrdN1DPP_D   Karnataka  

[2 rows x 22 columns]
```

```
aggregated_df['trip_creation_date'] = 
aggregated_df['trip_creation_time'].dt.date
aggregated_df['trip_creation_time'] = 
aggregated_df['trip_creation_time'].dt.time
aggregated_df.head()
print("Total columns:", len(aggregated_df.index))
```

```
Total columns: 26368
```

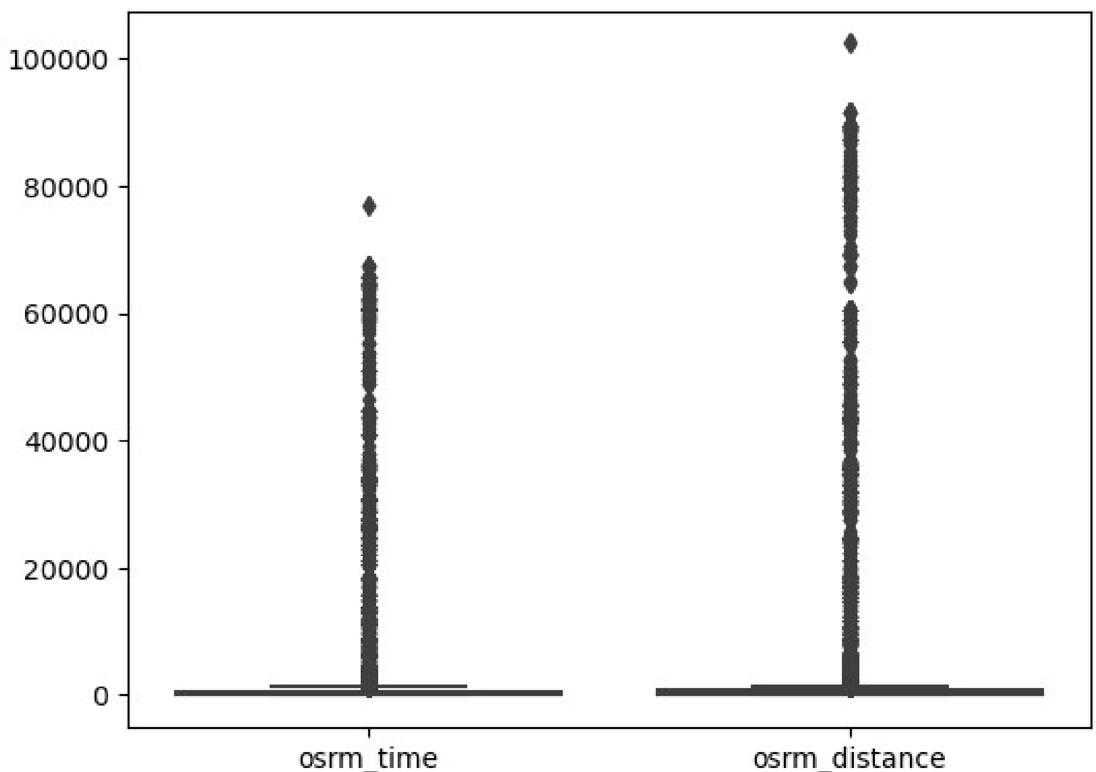
```
create_trip_dict = {
    'osrm_time' : sum,
    'osrm_distance' : sum,
    'segment_actual_time_sum' : sum,
    'od_time_diff_hour' : sum,
    'od_start_time' : 'first',
    'od_end_time' : 'last',
    'route_type' : 'first',
    'actual_time' : sum,
    'segment_actual_time' :sum,
    'segment_osrm_distance' : sum,
    'segment_osrm_time' : sum,
}
trip_summary_df = 
aggregated_df.groupby('trip_uuid').agg(create_trip_dict).reset_index()
trip_summary_df.rename(columns={'od_start_time': 'trip_start_time', 
'od_end_time': 'trip_end_time'}, inplace=True)
trip_summary_df.head(2)
```

```
                 trip_uuid  osrm_time  osrm_distance  
segment_actual_time_sum  \
0  trip-153671041653548748     7787.0     10577.7647   
1548.0
1  trip-153671042288605164      210.0       269.4308   
```

```
141.0

   od_time_diff_hour            trip_start_time
trip_end_time  \
0             37.668 2018-09-12 00:00:16.535741 2018-09-13
13:40:23.123744
1              3.027 2018-09-12 00:00:22.886430 2018-09-12
03:01:59.598855

  route_type  actual_time  segment_actual_time  segment_osrm_distance
\
0        FTL      15682.0               1548.0              1320.4733

1    Carting        399.0                141.0                84.1894


   segment_osrm_time
0             1008.0
1               65.0
```

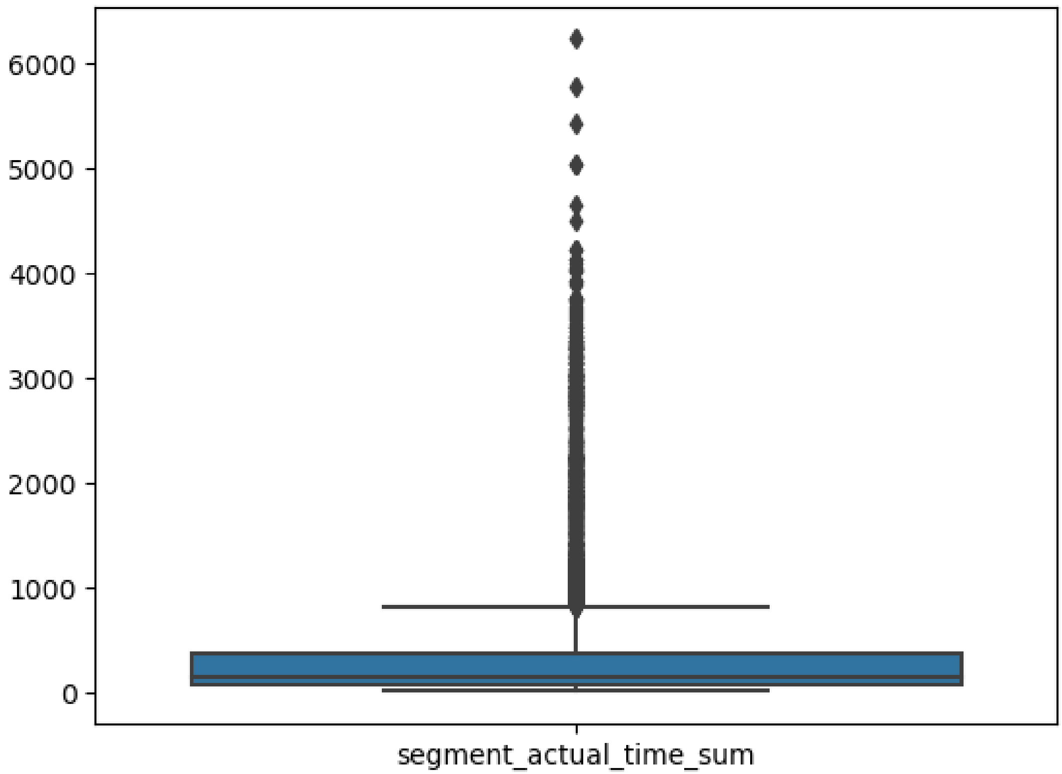
```python
data = trip_summary_df[['osrm_time', 'osrm_distance']]
sns.boxplot(data=data)
```

```
<Axes: >
```



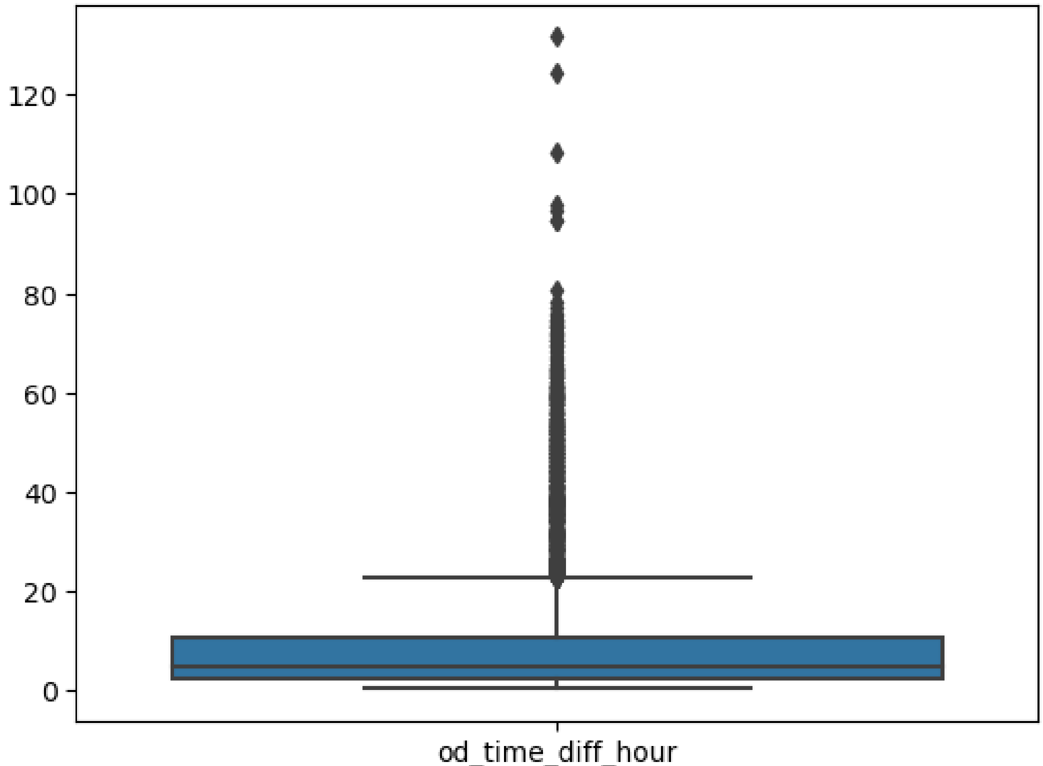
```python
data = trip_summary_df[['segment_actual_time_sum']]
sns.boxplot(data=data)
```

```
<Axes: >
```




```python
data = trip_summary_df[['od_time_diff_hour']]
sns.boxplot(data=data)
```

```
<Axes: >
```



Identifying Outliers using IQR - Before Log Transformation

```
#Identifying Outliers using IQR - Before Log Transformation
for i in ['osrm_time', 'osrm_distance', 'segment_actual_time_sum',
'od_time_diff_hour']:
    Q1 = np.percentile(trip_summary_df[i], 25)
    Q3 = np.percentile(trip_summary_df[i], 75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = [x for x in trip_summary_df[i] if x < lower_bound or x
> upper_bound]

    print("Lower bound for outliers(", i, ") :", lower_bound)
    print("Upper bound for outliers(", i, ") :", upper_bound)
    print("Outliers Length(", i, ") :", len(outliers))
```

```
Lower bound for outliers( osrm_time ) : -619.0
Upper bound for outliers( osrm_time ) : 1197.0
Outliers Length( osrm_time ) : 1948
Lower bound for outliers( osrm_distance ) : -747.1696000000001
Upper bound for outliers( osrm_distance ) : 1420.5856
Outliers Length( osrm_distance ) : 2069
```

```
Lower bound for outliers( segment_actual_time_sum ) : -385.5
Upper bound for outliers( segment_actual_time_sum ) : 818.5
Outliers Length( segment_actual_time_sum ) : 1643
Lower bound for outliers( od_time_diff_hour ) : -9.708
Upper bound for outliers( od_time_diff_hour ) : 22.844
Outliers Length( od_time_diff_hour ) : 1266
```

Handling outliers using log transformation

```
# Define the columns for which you want to apply log transformation
numeric_columns = ['osrm_time', 'osrm_distance',
'segment_actual_time_sum', 'od_time_diff_hour']

# Function to apply log transformation
def log_transform(column):
    # Shift the data if there are non-positive values
    shift = 0
    if (column <= 0).any():
        shift = np.abs(np.min(column)) + 1
        column = column + shift
    # Apply log transformation
    column = np.log(column)
    return column, shift

# Apply log transformation for each numeric column
shift_values = {}
for column in numeric_columns:
    trip_summary_df[column], shift =
log_transform(trip_summary_df[column])
    shift_values[column] = shift

# Verify the results
for column in numeric_columns:
    print(f"Shift value for {column}: {shift_values[column]}")
    print(f"Transformed {column} statistics:\
n{trip_summary_df[column].describe()}")

# Display the resulting DataFrame
print(trip_summary_df.head())
```

```
Shift value for osrm_time: 0
Transformed osrm_time statistics:
count    14817.000000
mean         5.375450
std          1.755661
min          1.791759
25%          4.127134
50%          5.117994
75%          6.246107
max         11.250950
```

```
Name: osrm_time, dtype: float64
Shift value for osrm_distance: 0
Transformed osrm_distance statistics:
count    14817.000000
mean         5.505835
std          1.822360
min          2.205292
25%          4.185686
50%          5.156717
75%          6.409644
max         11.536797
Name: osrm_distance, dtype: float64
Shift value for segment_actual_time_sum: 0
Transformed segment_actual_time_sum statistics:
count    14817.000000
mean         5.117768
std          1.179047
min          2.197225
25%          4.189655
50%          4.990433
75%          5.905362
max          8.737132
Name: segment_actual_time_sum, dtype: float64
Shift value for od_time_diff_hour: 0
Transformed od_time_diff_hour statistics:
count    14817.000000
mean         1.655298
std          0.999449
min         -0.939048
25%          0.915891
50%          1.543298
75%          2.364338
max          4.880094
Name: od_time_diff_hour, dtype: float64
                 trip_uuid  osrm_time  osrm_distance
segment_actual_time_sum  \
0  trip-153671041653548748   8.960211       9.266509
7.344719
1  trip-153671042288605164   5.347108       5.596312
4.948760
2  trip-153671043369099517  11.093889      11.401404
8.104099
3  trip-153671046011330457   3.178054       3.454659
4.077537
4  trip-153671052974046625   5.332719       5.584591
5.828946

   od_time_diff_hour            trip_start_time
trip_end_time  \
```

```
0            3.628811 2018-09-12 00:00:16.535741 2018-09-13
13:40:23.123744
1            1.107572 2018-09-12 00:00:22.886430 2018-09-12
03:01:59.598855
2            4.183164 2018-09-12 00:00:33.691250 2018-09-14
17:34:55.442454
3            0.515813 2018-09-12 00:01:00.113710 2018-09-12
01:41:29.809822
4            2.482654 2018-09-12 02:34:10.515593 2018-09-12
02:34:10.515593

  route_type  actual_time  segment_actual_time  segment_osrm_distance
\
0        FTL      15682.0               1548.0              1320.4733

1    Carting        399.0                141.0                84.1894

2        FTL     112225.0               3308.0              2545.2678

3    Carting         82.0                 59.0                19.8766

4        FTL        556.0                340.0               146.7919


   segment_osrm_time
0             1008.0
1               65.0
2             1941.0
3               16.0
4              115.0
```

Identifying Outliers using IQR - After Log Transformation

```
#Identifying Outliers using IQR - After Log Transformation
for i in ['osrm_time', 'osrm_distance', 'segment_actual_time_sum',
'od_time_diff_hour']:
    Q1 = np.percentile(trip_summary_df[i], 25)
    Q3 = np.percentile(trip_summary_df[i], 75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = [x for x in trip_summary_df[i] if x < lower_bound or x
> upper_bound]

    print("Lower bound for outliers(", i, ") :", lower_bound)
    print("Upper bound for outliers(", i, ") :", upper_bound)
    print("Outliers Length(", i, ") :", len(outliers))
```

```
Lower bound for outliers( osrm_time ) : 0.9486758143903851
Upper bound for outliers( osrm_time ) : 9.424565336136268
Outliers Length( osrm_time ) : 700
Lower bound for outliers( osrm_distance ) : 0.8497494564161165
Upper bound for outliers( osrm_distance ) : 9.745580964625692
Outliers Length( osrm_distance ) : 700
Lower bound for outliers( segment_actual_time_sum ) :
1.616094082984207
Upper bound for outliers( segment_actual_time_sum ) :
8.478922507096788
Outliers Length( segment_actual_time_sum ) : 5
Lower bound for outliers( od_time_diff_hour ) : -1.2567811042722536
Upper bound for outliers( od_time_diff_hour ) : 4.5370102453945975
Outliers Length( od_time_diff_hour ) : 7
```

Apply one-hot encoding on route_type

```
import pandas as pd

# Display the unique values in the 'route_type' column
print("Unique values in 'route_type':",
trip_summary_df['route_type'].unique())

# Apply one-hot encoding
trip_summary_df = pd.get_dummies(trip_summary_df,
columns=['route_type'], prefix='route')
trip_summary_df.head(3)
```

```
Unique values in 'route_type': ['FTL' 'Carting']

                 trip_uuid  osrm_time  osrm_distance
segment_actual_time_sum  \
0  trip-153671041653548748   8.960211       9.266509
7.344719
1  trip-153671042288605164   5.347108       5.596312
4.948760
2  trip-153671043369099517  11.093889      11.401404
8.104099

   od_time_diff_hour            trip_start_time
trip_end_time  \
0           3.628811 2018-09-12 00:00:16.535741 2018-09-13
13:40:23.123744
1           1.107572 2018-09-12 00:00:22.886430 2018-09-12
03:01:59.598855
2           4.183164 2018-09-12 00:00:33.691250 2018-09-14
17:34:55.442454

   actual_time  segment_actual_time  segment_osrm_distance
segment_osrm_time  \
```

```
0      15682.0               1548.0               1320.4733
1008.0
1        399.0                141.0                 84.1894
65.0
2     112225.0               3308.0               2545.2678
1941.0

   route_Carting  route_FTL
0              0          1
1              1          0
2              0          1
```

Column Normalization

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Select the columns to be scaled
numerical_features = ['osrm_time', 'osrm_distance',
'segment_actual_time_sum', 'od_time_diff_hour']

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the numerical features
trip_summary_df[numerical_features] =
scaler.fit_transform(trip_summary_df[numerical_features])

trip_summary_df.head(5)

                 trip_uuid  osrm_time  osrm_distance
segment_actual_time_sum  \
0  trip-153671041653548748   0.757829       0.756707
0.787090
1  trip-153671042288605164   0.375862       0.363395
0.420730
2  trip-153671043369099517   0.983396       0.985491
0.903205
3  trip-153671046011330457   0.146555       0.133887
0.287514
4  trip-153671052974046625   0.374341       0.362139
0.555317

   od_time_diff_hour            trip_start_time
trip_end_time  \
0           0.784971 2018-09-12 00:00:16.535741 2018-09-13
13:40:23.123744
1           0.351705 2018-09-12 00:00:22.886430 2018-09-12
03:01:59.598855
2           0.880235 2018-09-12 00:00:33.691250 2018-09-14
```

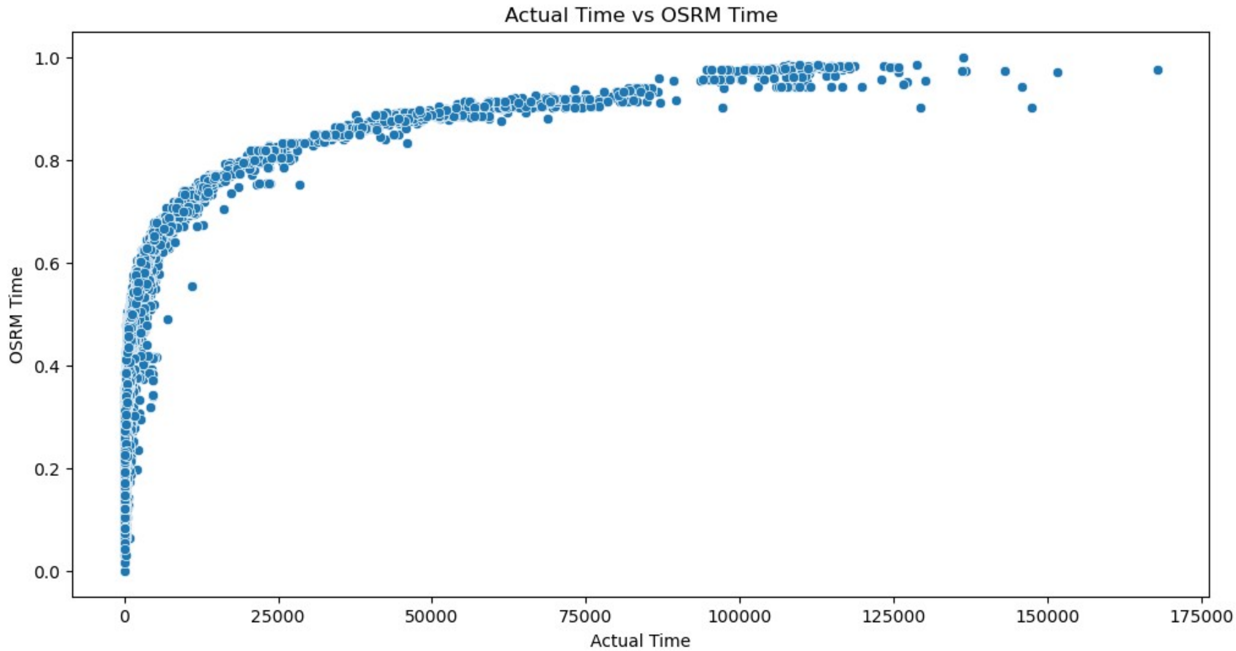
```
17:34:55.442454
3           0.250013 2018-09-12 00:01:00.113710 2018-09-12
01:41:29.809822
4           0.588008 2018-09-12 02:34:10.515593 2018-09-12
02:34:10.515593

   actual_time  segment_actual_time  segment_osrm_distance
segment_osrm_time  \
0      15682.0               1548.0              1320.4733
1008.0
1        399.0                141.0                84.1894
65.0
2     112225.0               3308.0              2545.2678
1941.0
3         82.0                 59.0                19.8766
16.0
4        556.0                340.0               146.7919
115.0

   route_Carting  route_FTL
0              0          1
1              1          0
2              0          1
3              1          0
4              0          1
```

Perform Hypothesis Testing and Visual Analysis

Actual Time vs OSRM Time

```
# Hypothesis Testing
t_stat, p_value = stats.ttest_rel(trip_summary_df['actual_time'],
trip_summary_df['osrm_time'])
print(f'T-test statistics: {t_stat}, p-value: {p_value}')

# Visual Analysis
plt.figure(figsize=(12, 6))
sns.scatterplot(x='actual_time', y='osrm_time', data=trip_summary_df)
plt.xlabel('Actual Time')
plt.ylabel('OSRM Time')
plt.title('Actual Time vs OSRM Time')
plt.show()
```

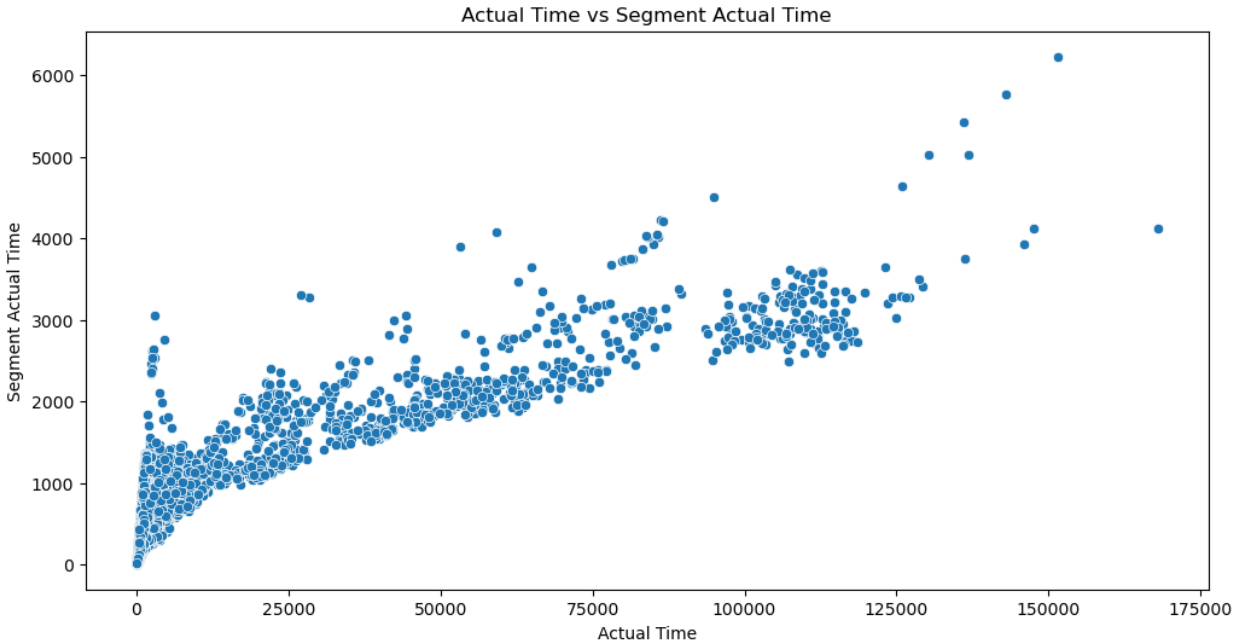
```
T-test statistics: 32.60526278723588, p-value: 2.880837087056349e-225
```



Perform Hypothesis Testing and Visual Analysis

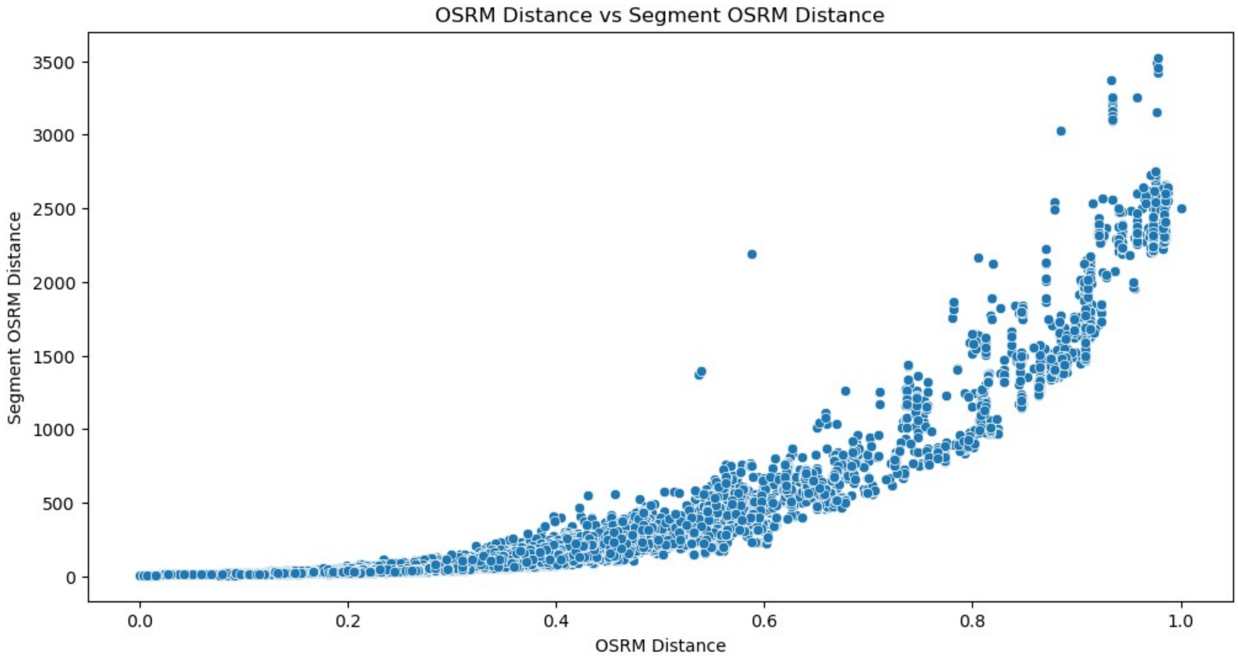
Actual Time vs Segment Actual Time

```
# Hypothesis Testing
t_stat, p_value = stats.ttest_rel(trip_summary_df['actual_time'],
trip_summary_df['segment_actual_time'])
print(f'T-test statistics: {t_stat}, p-value: {p_value}')

# Visual Analysis
plt.figure(figsize=(12, 6))
sns.scatterplot(x='actual_time', y='segment_actual_time',
data=trip_summary_df)
plt.xlabel('Actual Time')
plt.ylabel('Segment Actual Time')
plt.title('Actual Time vs Segment Actual Time')
plt.show()

T-test statistics: 30.75550616001704, p-value: 2.0773254218008745e-201
```



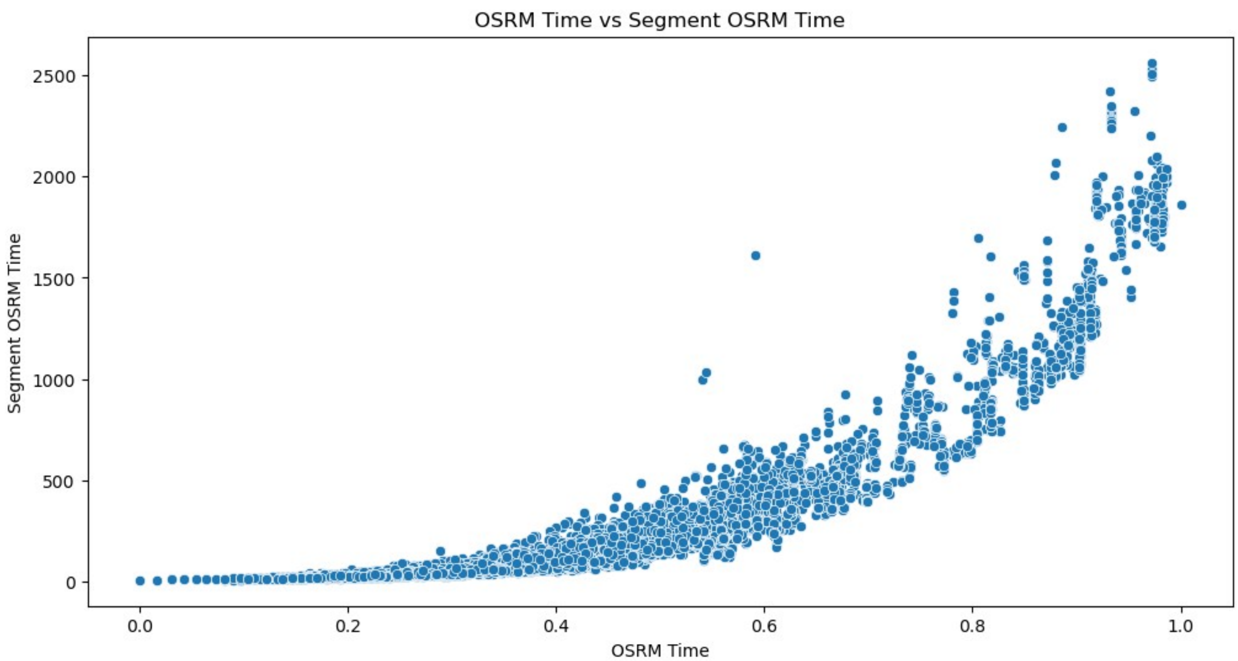
Osrm Distance vs Segment OSRM Distance

```
# Hypothesis Testing
t_stat, p_value = stats.ttest_rel(trip_summary_df['osrm_distance'],
trip_summary_df['segment_osrm_distance'])
print(f'T-test statistics: {t_stat}, p-value: {p_value}')

# Visual Analysis
plt.figure(figsize=(12, 6))
sns.scatterplot(x='osrm_distance', y='segment_osrm_distance',
data=trip_summary_df)
plt.xlabel('OSRM Distance')
plt.ylabel('Segment OSRM Distance')
plt.title('OSRM Distance vs Segment OSRM Distance')
plt.show()

T-test statistics: -65.13429922166652, p-value: 0.0
```



OSRM Time vs Segment OSRM Time

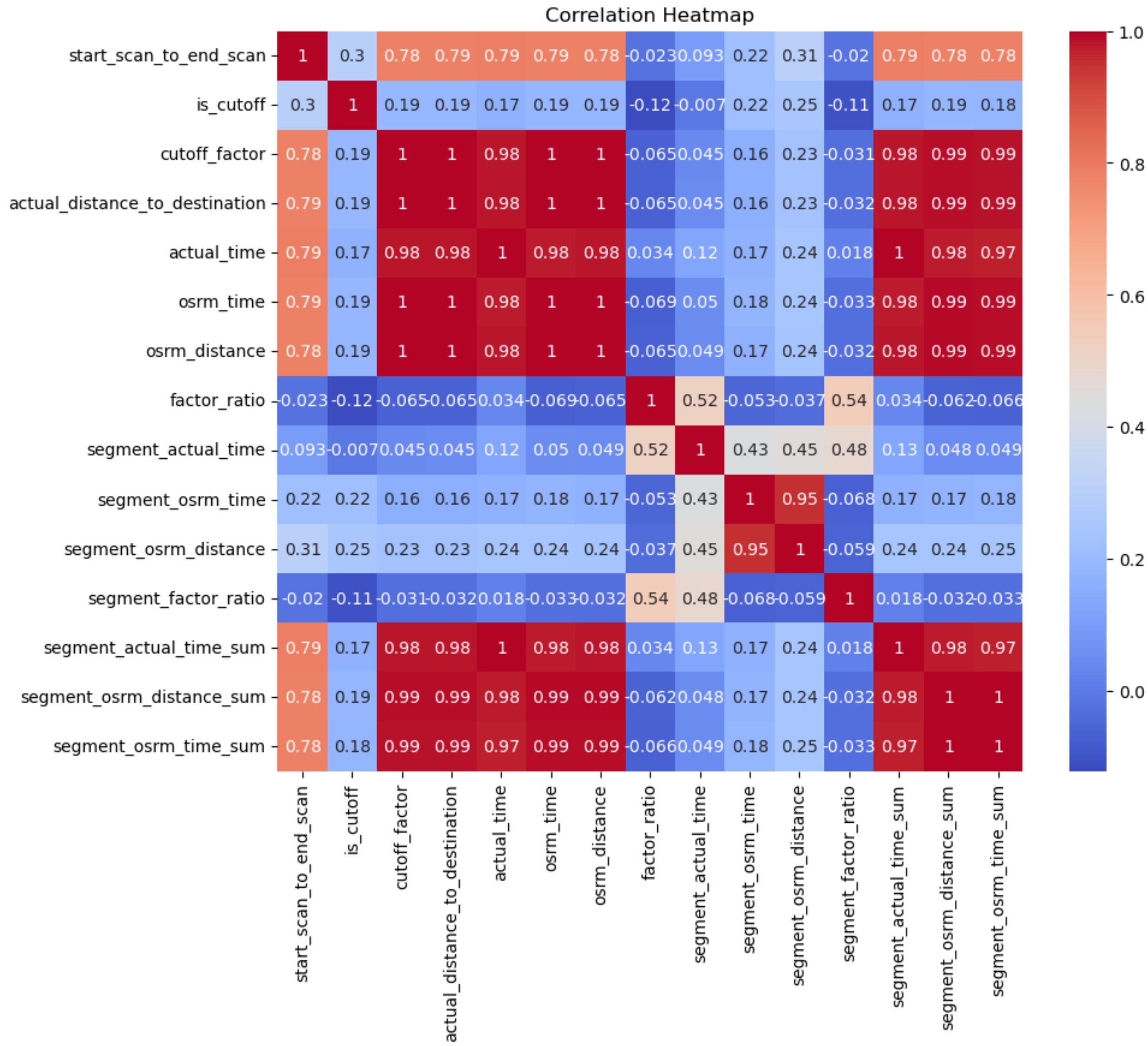
```
# Hypothesis Testing
t_stat, p_value = stats.ttest_rel(trip_summary_df['osrm_time'],
trip_summary_df['segment_osrm_time'])
print(f'T-test statistics: {t_stat}, p-value: {p_value}')

# Visual Analysis
plt.figure(figsize=(12, 6))
sns.scatterplot(x='osrm_time', y='segment_osrm_time',
data=trip_summary_df)
plt.xlabel('OSRM Time')
plt.ylabel('Segment OSRM Time')
plt.title('OSRM Time vs Segment OSRM Time')
plt.show()

T-test statistics: -69.91440627201277, p-value: 0.0
```



Correlation Heatmap

```
plt.figure(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

```
/var/folders/bc/byp79cv56b53hq4lpz78hyvh0000gn/T/
ipykernel_62169/1170417230.py:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corr = df.corr()
```



Strong Positive Correlations:

The variables start_scan_to_end_scan, actual_distance_to_destination, actual_time, osrm_time, osrm_distance, segment_actual_time_sum, segment_osrm_distance_sum, and segment_osrm_time_sum all have strong positive correlations with each other. This suggests that as one of these variables increases, the others tend to increase as well.

Weak Positive Correlations:

The variables is_cutoff and cutoff_factor have weak positive correlations with most other variables in the matrix.

Recommendations

1. **Focus on Busiest Corridor**: Allocate more resources and marketing efforts to the busiest corridor identified in the data to capitalize on the high demand and increase profitability.
2. **Optimize Routes**: Analyze the average distance and time taken between corridors to identify opportunities for route optimization. This can help reduce transportation costs and improve efficiency.
3. **Improve Delivery Time**: Identify factors contributing to longer delivery times, such as traffic congestion or inefficient routes, and implement strategies to reduce them. This could include using real-time traffic data or adjusting delivery schedules.
4. **Expand Service Coverage**: Explore opportunities to expand service coverage to areas with high demand or underserved regions identified in the data. This can help capture additional market share and increase revenue.
5. **Streamline Operations**: Streamline internal processes and workflows to improve overall efficiency and reduce operating costs. This could involve investing in technology solutions or implementing training programs for staff.This will also lead to more customer satisfaction

By focusing on these actionable items, the business can enhance its operations, improve customer satisfaction, and drive growth in the identified corridors.