Delhivery Case Study

Business problem

To extract, clean, and engineer meaningful features from raw data generated by Delhivery's data pipelines, enabling the team to build accurate forecasting models that support data-driven decisions aimed at improving operational efficiency, profitability, and customer service quality

```
import pandas as pd

df= pd.read_csv("delhivery_data csv.txt")
df.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886

5 rows × 24 columns

Exploratory Data Analysis (EDA)

```
print("Shape of the data (rows, columns):", df.shape)
```

```
Shape of the data (rows, columns): (144867, 24)
```

```
print("Data types of attributes:")
print(df.dtypes)
```

```
Data types of attributes:
data                              object
trip_creation_time                object
route_schedule_uuid               object
route_type                        object
trip_uuid                         object
source_center                     object
source_name                       object
destination_center                object
destination_name                  object
od_start_time                     object
od_end_time                       object
start_scan_to_end_scan           float64
is_cutoff                           bool
cutoff_factor                      int64
cutoff_timestamp                  object
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
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool   
 13  cutoff_factor                   144867 non-null  int64  
 14  cutoff_timestamp                144867 non-null  object 
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

```
print("Missing values in each column:")
print(df.isna().sum())
```

```
Missing values in each column:
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
```

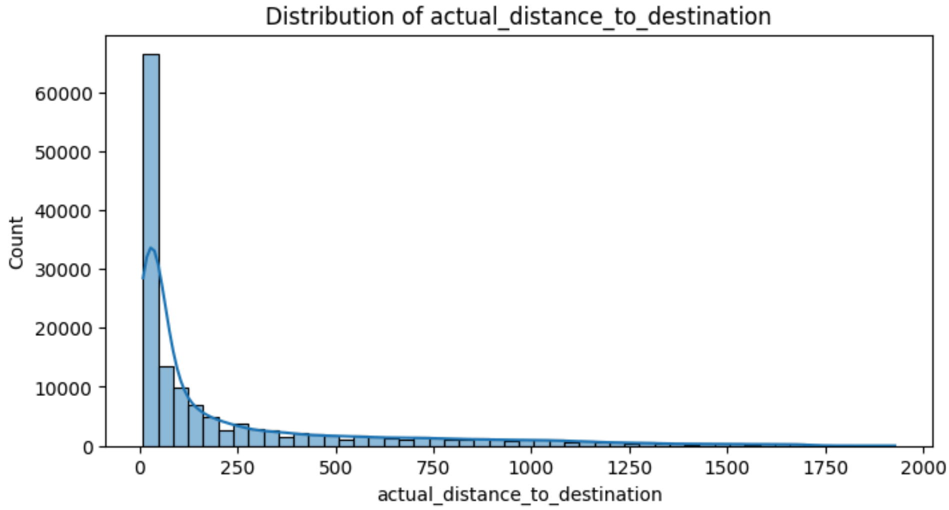
This mean there are two columns in their data that has source_name and destination_name has null values

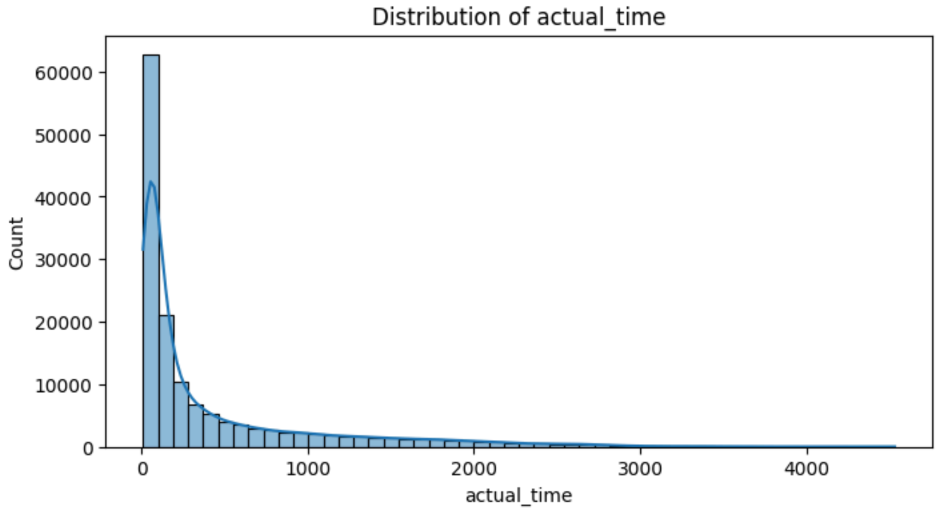
```
df.head(5)
```

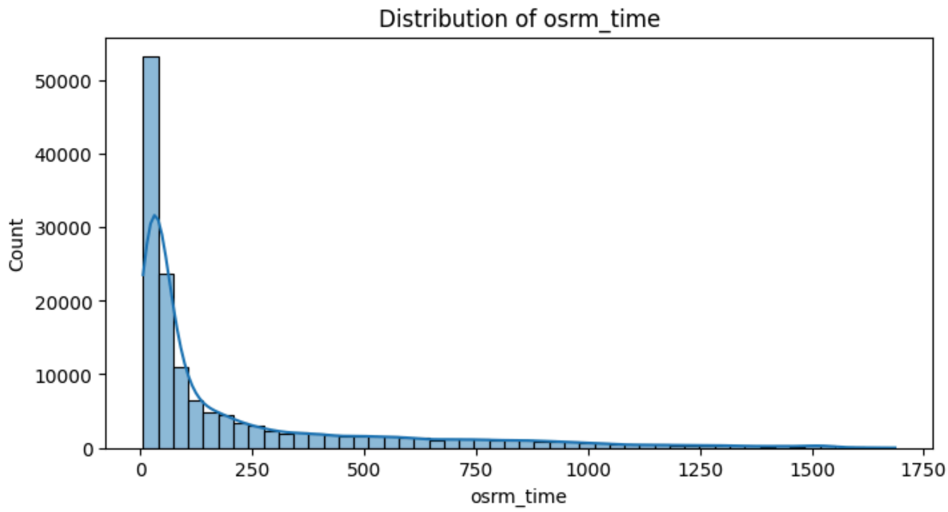
	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886

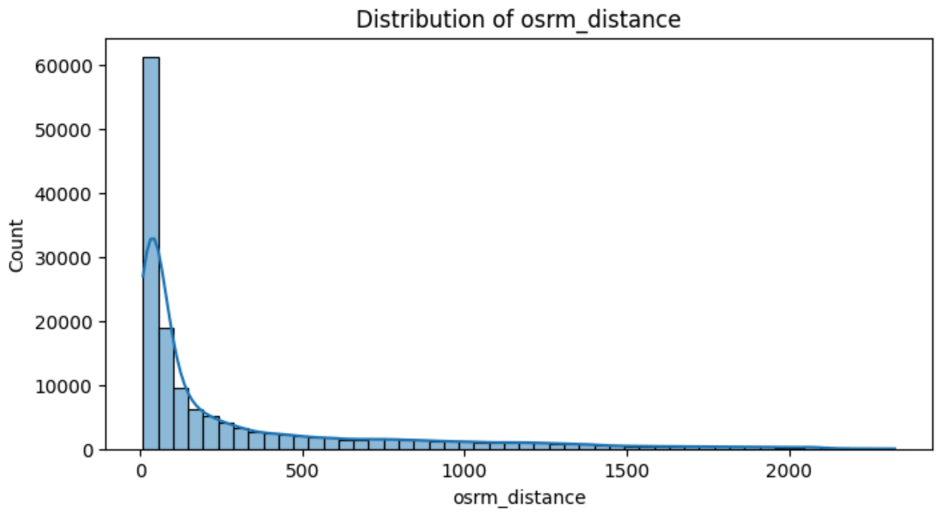
5 rows × 24 columns

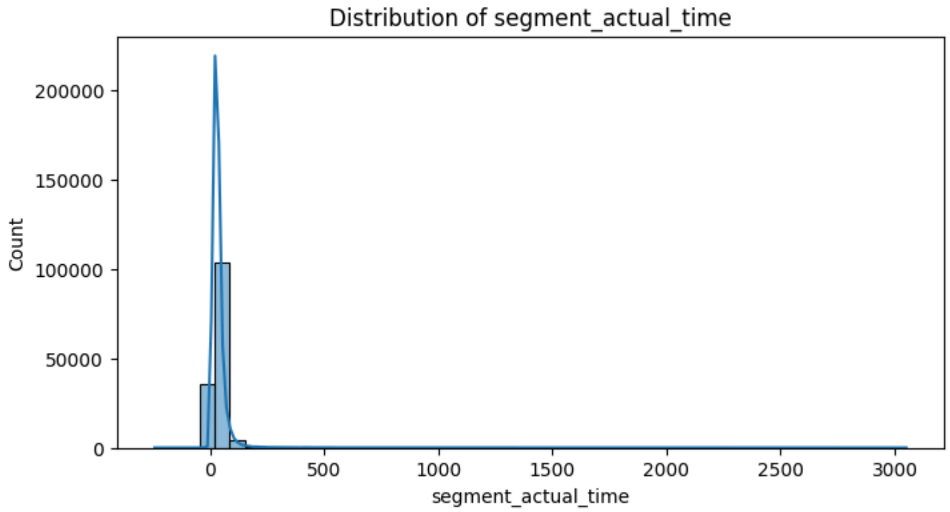
```
import matplotlib.pyplot as plt
import seaborn as sns

# Corrected continuous variable list (no trailing spaces)
cont_vars = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance', 'segment_actual_time', 'segment_osrm_distanc

for var in cont_vars:
    plt.figure(figsize=(8,4))
    sns.histplot(df[var], bins=50, kde=True)
    plt.title(f'Distribution of {var}')
    plt.show()
```

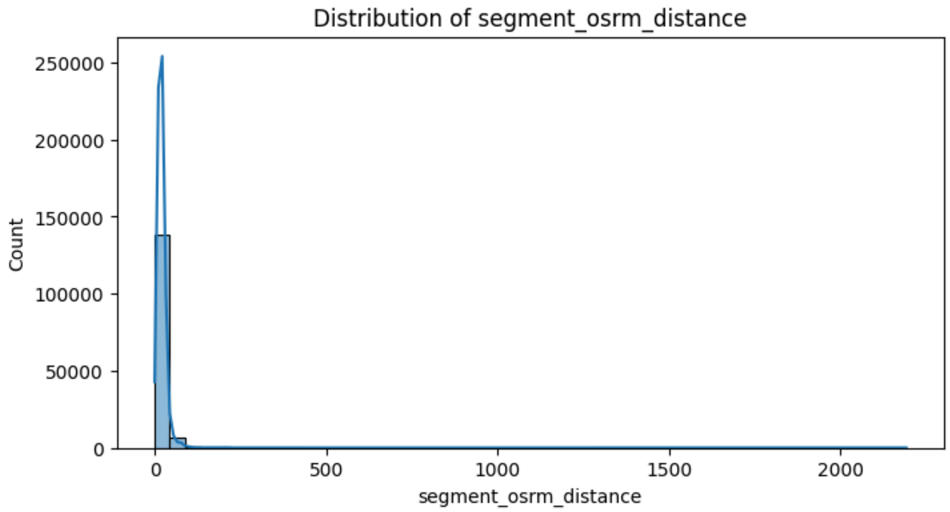

Distribution of actual_distance_to_destination
Count
actual_distance_to_destination

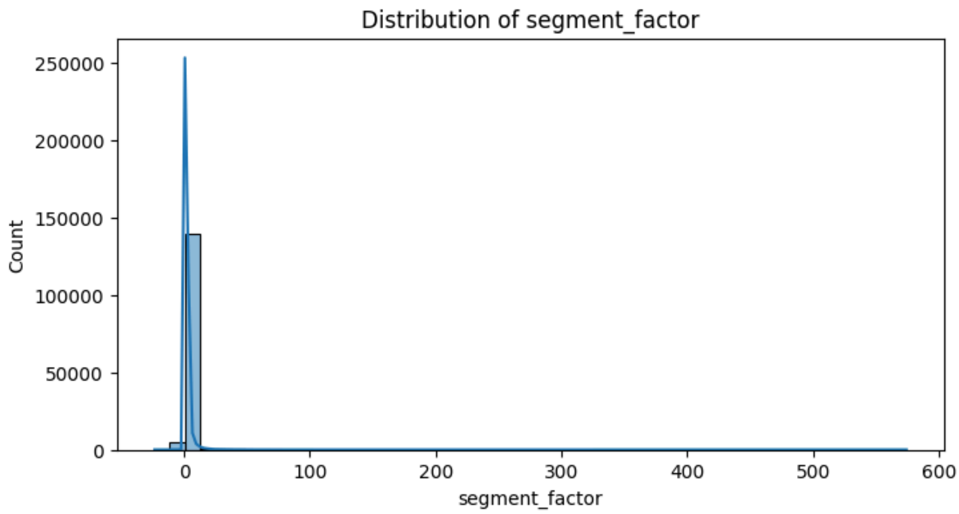

Distribution of actual_time
Count
actual_time


Distribution of osrm_time
Count
osrm_time


Distribution of osrm_distance
Count
osrm_distance


Distribution of segment_actual_time
Count
200000
150000
100000
50000
0
0
500
1000
1500
2000
2500
3000
segment_actual_time


Distribution of segment_osrm_distance
Count
250000
200000
150000
100000
50000
0
0
500
1000
1500
2000
segment_osrm_distance


Distribution of segment_factor
Count
250000
200000
150000
100000
50000
0
0
100
200
300
400
500
600
segment_factor

Feature Creations

```
df['speed'] = df['actual_distance_to_destination'] / df['actual_time']
```

```
df.head(5)
```

Time Difference Between OSRM and Actual

```
df['time_diff'] = df['actual_time'] - df['osrm_time']
df.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND3886

5 rows × 27 columns

Distance Difference Between OSRM and Actual

```
df['distance_diff'] = df['actual_distance_to_destination'] - df['osrm_distance']
df.head(5)
```

Segment-level Efficiency

```
df['segment_speed'] = df['segment_osrm_distance'] / df['segment_actual_time']
df.head(5)
```

*Merging of rows and aggregation of fields *

Aggregate by route_schedule_uuid

```
agg_route = df.groupby('route_schedule_uuid').agg(
    trip_count=('trip_uuid', 'nunique'),
    avg_actual_time=('actual_time', 'mean'),
    total_actual_time=('actual_time', 'sum'),
    avg_actual_distance=('actual_distance_to_destination', 'mean'),
    total_actual_distance=('actual_distance_to_destination', 'sum'),
    avg_speed=('speed', 'mean'),
    cutoff_trip_count=('is_cutoff', 'sum'),
    avg_cutoff_factor=('cutoff_factor', 'mean')
).reset_index()

print(agg_route.head())
```

```
                              route_schedule_uuid  trip_count  \
0  thanos::sroute:0007affd-fd01-4cf0-8a4f-90419df...          24
1  thanos::sroute:00435307-de7f-4439-bd6a-5a2a9a3...           9
2  thanos::sroute:00a74fab-a3ac-44df-b83a-cbf181b...           4
3  thanos::sroute:00b294b8-d2c3-4bca-a3be-684f462...           8
4  thanos::sroute:01164881-301e-45f8-bacd-ee21c37...          15

   avg_actual_time  total_actual_time  avg_actual_distance  \
0        61.430769             3993.0            13.409445
1        87.916667             3165.0            21.485936
2        71.500000              858.0            43.013852
```

```
3      73.589928        10229.0        26.988311
4      93.264368         8114.0        38.310154

   total_actual_distance  avg_speed  cutoff_trip_count  avg_cutoff_factor
0             871.613939   0.276111                 32          12.600000
1             773.493695   0.274662                 27          20.861111
2             516.166219   0.655998                  8          41.750000
3            3751.375293   0.423206                107          26.330935
4            3332.983421   0.444064                 54          37.436782
```

2. Aggregate by Corridor (source_center → destination_center)

```
agg_corridor = df.groupby(['source_center', 'destination_center']).agg(
    trip_count=('trip_uuid', 'nunique'),
    avg_actual_time=('actual_time', 'mean'),
    avg_actual_distance=('actual_distance_to_destination', 'mean'),
    avg_speed=('speed', 'mean'),
    cutoff_trip_count=('is_cutoff', 'sum')
).reset_index()

print(agg_corridor.head())
```

```
  source_center destination_center  trip_count  avg_actual_time  \
0  IND000000AAL       IND411033AAA          18        63.702703
1  IND000000AAQ       IND700028AAB           2        78.250000
2  IND000000AAS       IND783370AAC           9        50.555556
3  IND000000AAZ       IND444203AAA           1       181.666667
4  IND000000AAZ       IND444303AAA           1       122.333333

   avg_actual_distance  avg_speed  cutoff_trip_count
0            13.890556   0.282074                 19
1            10.295014   0.141828                  2
2            26.000435   0.550613                  9
3            42.244027   0.248681                  2
4            38.177141   0.361501                  2
```

3. Aggregate by Date or Time (optional if timestamp available)

```
df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'])
df['trip_date'] = df['trip_creation_time'].dt.date

agg_daily = df.groupby('trip_date').agg(
    trip_count=('trip_uuid', 'nunique'),
    avg_actual_time=('actual_time', 'mean'),
    avg_actual_distance=('actual_distance_to_destination', 'mean'),
    avg_speed=('speed', 'mean'),
    cutoff_trip_count=('is_cutoff', 'sum')
).reset_index()

print(agg_daily.head())
```

```
    trip_date  trip_count  avg_actual_time  avg_actual_distance  avg_speed  \
0  2018-09-12         747       376.343960           213.922594   0.533108
1  2018-09-13         750       385.313924           218.478347   0.545978
2  2018-09-14         712       378.134528           211.213672   0.545474
3  2018-09-15         783       422.588515           235.789133   0.542685
4  2018-09-16         616       414.699462           244.491189   0.572593

   cutoff_trip_count
0               5713
1               5800
2               5521
3               6081
4               5595
```

Comparison & Visualization of time and distance fields

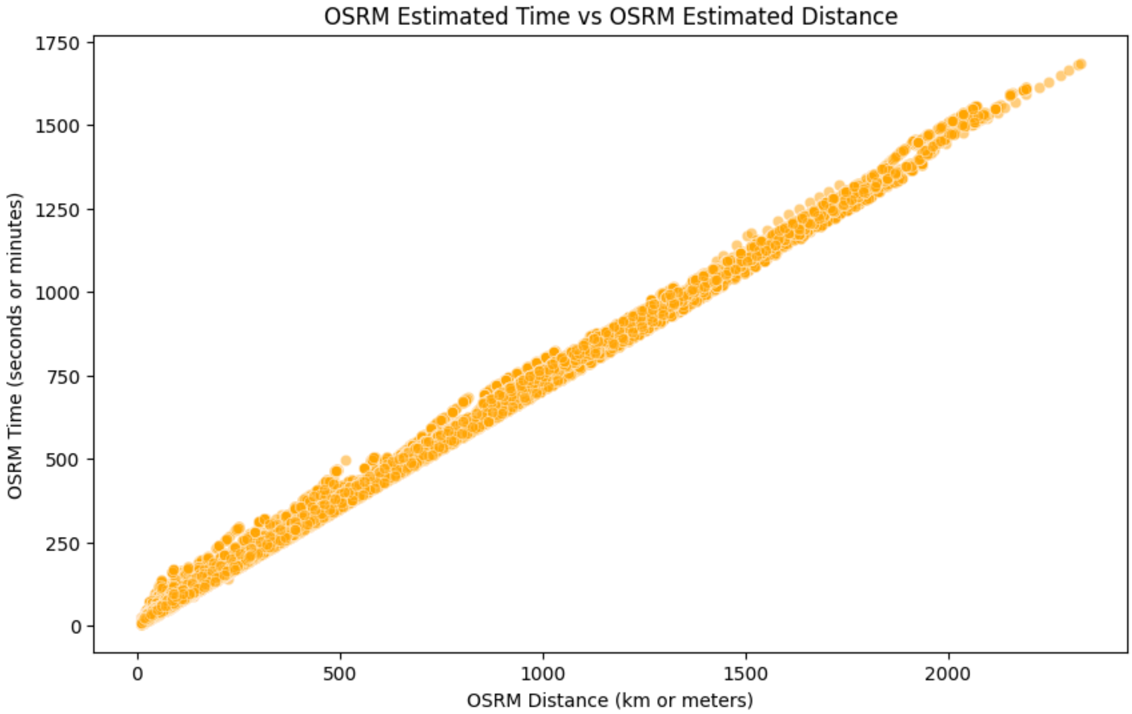
1. Scatter plot: Actual time vs Actual distance

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the size of the figure to 10 inches wide and 6 inches tall
plt.figure(figsize=(10,6))



sns.scatterplot(x='actual_distance_to_destination', y='actual_time', data=df, alpha=0.5)

# Add a title to the plot
plt.title('Actual Delivery Time vs Actual Distance')
```

```
# Label the x-axis
plt.xlabel('Actual Distance to Destination (km or meters)')

# Label the y-axis
plt.ylabel('Actual Time (seconds or minutes)')

# Display the plot
plt.show()
```

2. Scatter plot: OSRM time vs OSRM distance

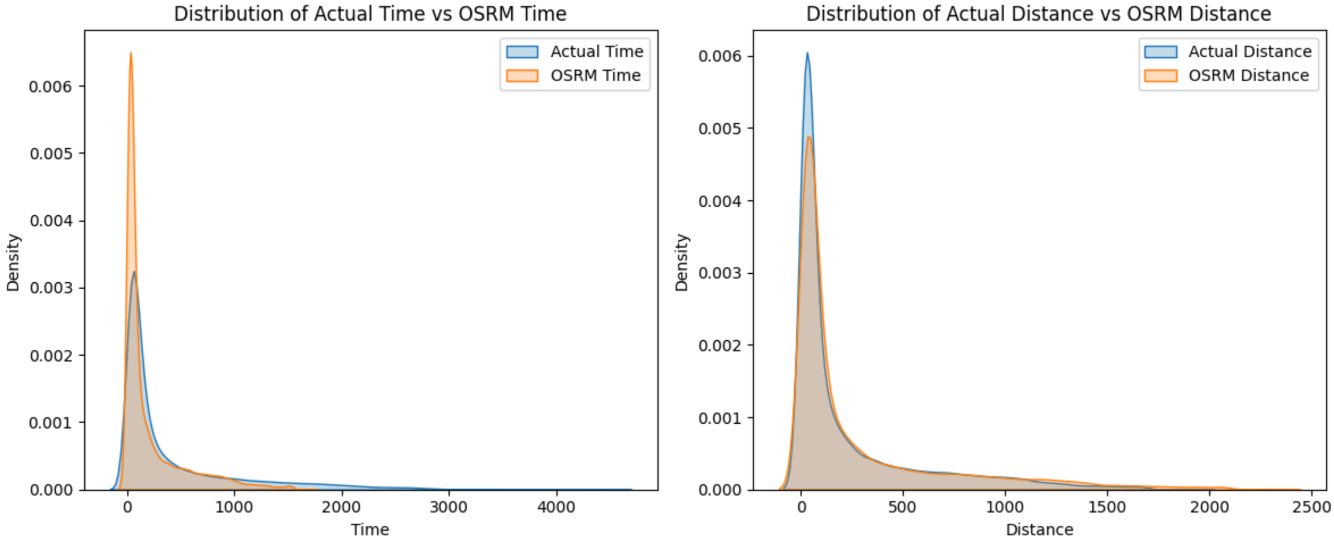
```
# Set up the figure size to 10 inches wide and 6 inches tall
plt.figure(figsize=(10,6))

# Create a scatter plot with:

sns.scatterplot(x='osrm_distance', y='osrm_time', data=df, alpha=0.5, color='orange')

# Set the plot title
plt.title('OSRM Estimated Time vs OSRM Estimated Distance')

# Label the x-axis
plt.xlabel('OSRM Distance (km or meters)')

# Label the y-axis
plt.ylabel('OSRM Time (seconds or minutes)')

# Show the plot
plt.show()
```



3. **Comparing Actual vs OSRM time and distance side by side**

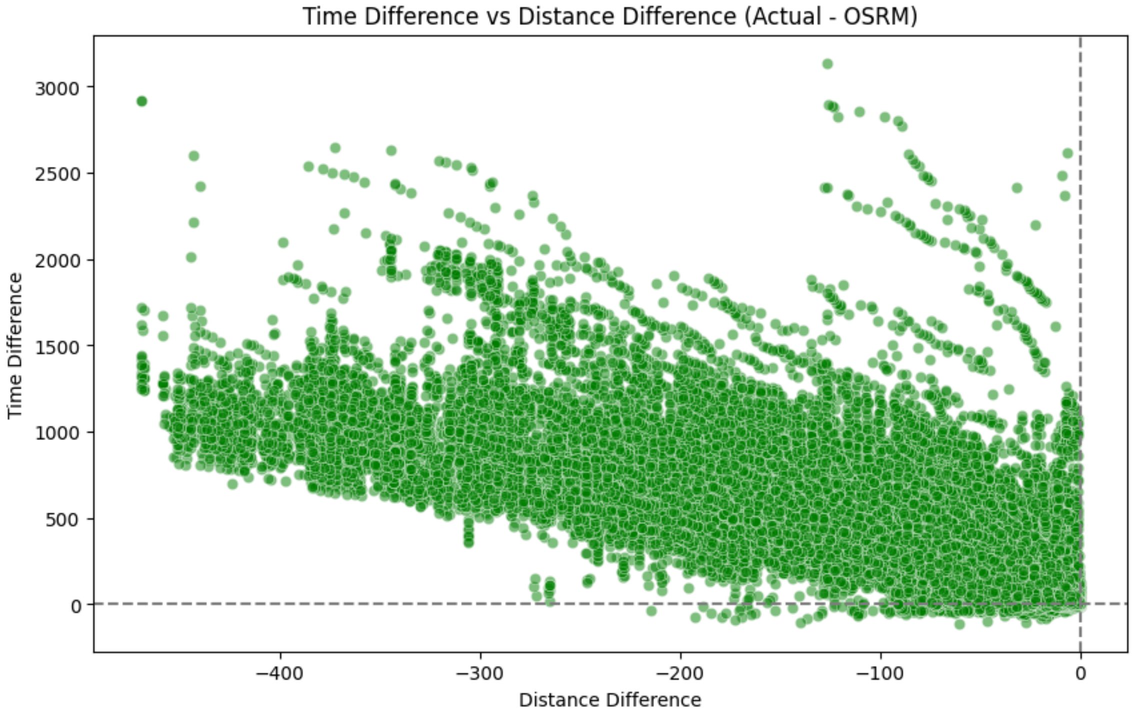
```
# Set up a figure with size 12 inches wide and 5 inches tall
plt.figure(figsize=(12,5))

# Create the first subplot in a 1 row x 2 columns grid (first plot)
plt.subplot(1,2,1)

# Plot kernel density estimate (KDE) for actual delivery times with shading
sns.kdeplot(df['actual_time'], label='Actual Time', shade=True)

# Plot KDE for OSRM estimated times on the same subplot with shading
sns.kdeplot(df['osrm_time'], label='OSRM Time', shade=True)

# Set title for the first subplot
plt.title('Distribution of Actual Time vs OSRM Time')

# Label the x-axis
plt.xlabel('Time')

# Show legend to differentiate actual and OSRM time distributions
plt.legend()

# Create the second subplot in the 1x2 grid (second plot)
plt.subplot(1,2,2)

# Plot KDE for actual delivery distances with shading
sns.kdeplot(df['actual_distance_to_destination'], label='Actual Distance', shade=True)

# Plot KDE for OSRM estimated distances with shading on the same subplot
sns.kdeplot(df['osrm_distance'], label='OSRM Distance', shade=True)

# Set title for the second subplot
plt.title('Distribution of Actual Distance vs OSRM Distance')

# Label the x-axis
plt.xlabel('Distance')

# Show legend to differentiate actual and OSRM distance distributions
plt.legend()

# Adjust subplot spacing to prevent overlap
plt.tight_layout()

# Display the plot
plt.show()
```

```
<ipython-input-57-26e20af51bca>:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['actual_time'], label='Actual Time', shade=True)
<ipython-input-57-26e20af51bca>:11: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['osrm_time'], label='OSRM Time', shade=True)
<ipython-input-57-26e20af51bca>:26: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['actual_distance_to_destination'], label='Actual Distance', shade=True)
<ipython-input-57-26e20af51bca>:29: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['osrm_distance'], label='OSRM Distance', shade=True)
```



Time difference vs distance difference scatter plot

```
# Calculate the difference between actual and estimated time (OSRM)
df['time_diff'] = df['actual_time'] - df['osrm_time']

# Calculate the difference between actual and estimated distance (OSRM)
df['distance_diff'] = df['actual_distance_to_destination'] - df['osrm_distance']

# Set up the plot with a figure size of 10x6 inches
plt.figure(figsize=(10,6))

sns.scatterplot(x='distance_diff', y='time_diff', data=df, alpha=0.5, color='green')

# Add a title to the plot
plt.title('Time Difference vs Distance Difference (Actual - OSRM)')

# Label x-axis
plt.xlabel('Distance Difference')

# Label y-axis
plt.ylabel('Time Difference')

# Add a horizontal dashed line at y=0 to show where time difference is zero
plt.axhline(0, linestyle='--', color='gray')

# Add a vertical dashed line at x=0 to show where distance difference is zero
plt.axvline(0, linestyle='--', color='gray')

# Display the plot
plt.show()
```



. Missing Values Treatment

```
# Fill missing source_name and destination_name with 'Unknown'
df['source_name'] = df['source_name'].fillna('Unknown')
df['destination_name'] = df['destination_name'].fillna('Unknown')
df.sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destir
19146	test	2018-10-01 23:43:30.389791	thanos::sroute:67c77992-49e3-4594-9a75-9861ef0...	FTL	trip-153843741038955003	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	I
43154	training	2018-09-18 04:59:16.125309	thanos::sroute:6be6529b-f2ad-4714-b7ab-ac58f24...	FTL	trip-153724675612503042	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	I
120571	training	2018-09-25 04:21:12.551117	thanos::sroute:96a80600-40e1-436b-9161-fa68f9e...	FTL	trip-153784927255069118	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	I
127654	training	2018-09-19 13:56:50.375577	thanos::sroute:38e6d56f-8d2e-44c8-9366-85501ed...	FTL	trip-153736541037531806	IND757037AAA	Karanjia_Sarubali_D (Orissa)	I
119608	test	2018-10-02 06:46:03.336668	thanos::sroute:41c12a39-f117-4645-8e2a-e4901e0...	FTL	trip-153846276333642499	IND421302AAG	Bhiwandi_Mankoli_HB (Maharashtra)	I

5 rows × 29 columns

Outlier Treatment

```
# List of continuous variables to check for outliers
cont_vars = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance',
             'segment_actual_time', 'segment_osrm_distance', 'segment_factor']

# Loop through each continuous variable
for var in cont_vars:
    # Calculate the 25th percentile (Q1) of the variable
    Q1 = df[var].quantile(0.25)

    # Calculate the 75th percentile (Q3) of the variable
    Q3 = df[var].quantile(0.75)

    # Calculate the Interquartile Range (IQR)
    IQR = Q3 - Q1
```

```
    # Define lower bound for outliers (1.5 times IQR below Q1)
    lower_bound = Q1 - 1.5 * IQR

    # Define upper bound for outliers (1.5 times IQR above Q3)
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers that lie outside the lower and upper bounds
    outliers = df[(df[var] < lower_bound) | (df[var] > upper_bound)]

    # Print the variable name and number of outliers found
    print(f"{var}: {len(outliers)} outliers")
```

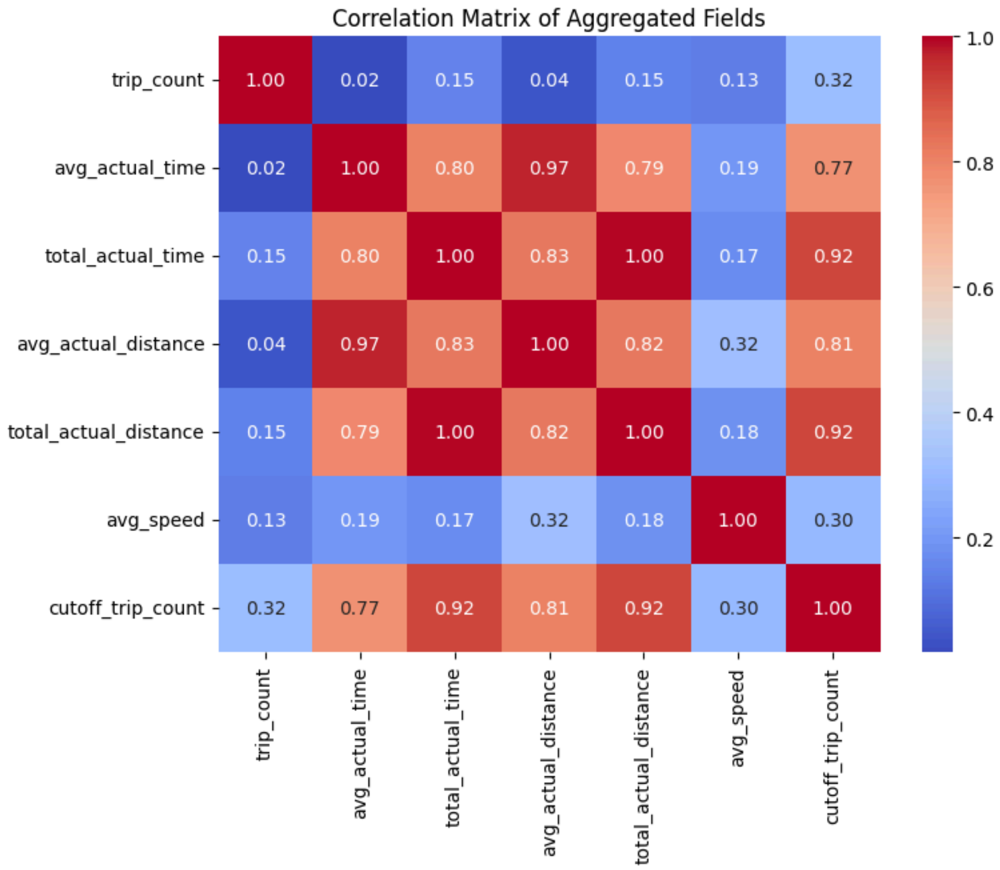
```
actual_distance_to_destination: 17992 outliers
actual_time: 16633 outliers
osrm_time: 17603 outliers
osrm_distance: 17816 outliers
segment_actual_time: 9298 outliers
segment_osrm_distance: 4315 outliers
segment_factor: 13976 outliers
```

**Checking relationship between aggregated fields **

Correlation Analysis

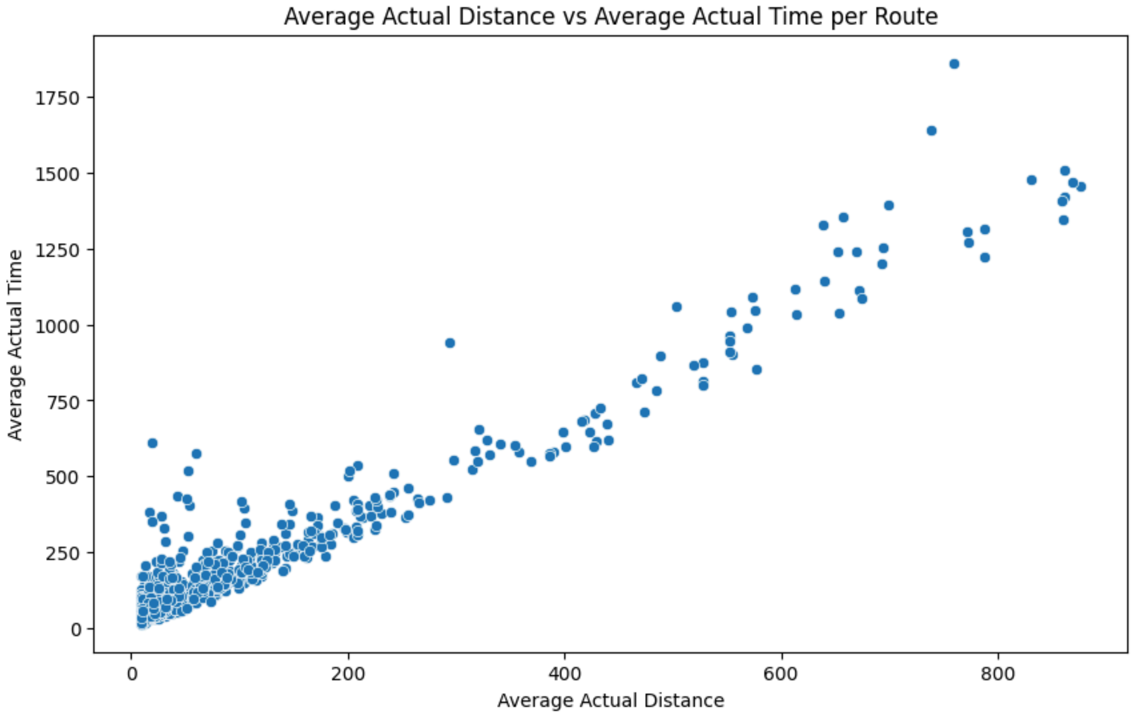
```
# Assuming agg_route is your aggregated dataframe at route_schedule_uuid level
corr_matrix = agg_route[['trip_count', 'avg_actual_time', 'total_actual_time',
                         'avg_actual_distance', 'total_actual_distance', 'avg_speed', 'cutoff_trip_count']].corr()

print(corr_matrix)

# Heatmap visualization of correlations
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Aggregated Fields')
plt.show()
```

```
                      trip_count  avg_actual_time  total_actual_time  \
trip_count              1.000000         0.016759           0.150566
avg_actual_time         0.016759         1.000000           0.802769
total_actual_time       0.150566         0.802769           1.000000
avg_actual_distance     0.036972         0.969634           0.825647
total_actual_distance   0.146592         0.787246           0.995095
avg_speed               0.133761         0.189815           0.171857
cutoff_trip_count       0.315825         0.766460           0.916447

                      avg_actual_distance  total_actual_distance  avg_speed  \
trip_count                       0.036972               0.146592   0.133761
avg_actual_time                  0.969634               0.787246   0.189815
total_actual_time                0.825647               0.995095   0.171857
avg_actual_distance              1.000000               0.821473   0.318457
total_actual_distance            0.821473               1.000000   0.181913
avg_speed                        0.318457               0.181913   1.000000
cutoff_trip_count                0.807325               0.916173   0.300866

                      cutoff_trip_count
trip_count                     0.315825
avg_actual_time                0.766460
total_actual_time              0.916447
avg_actual_distance            0.807325
total_actual_distance          0.916173
avg_speed                      0.300866
cutoff_trip_count              1.000000
```



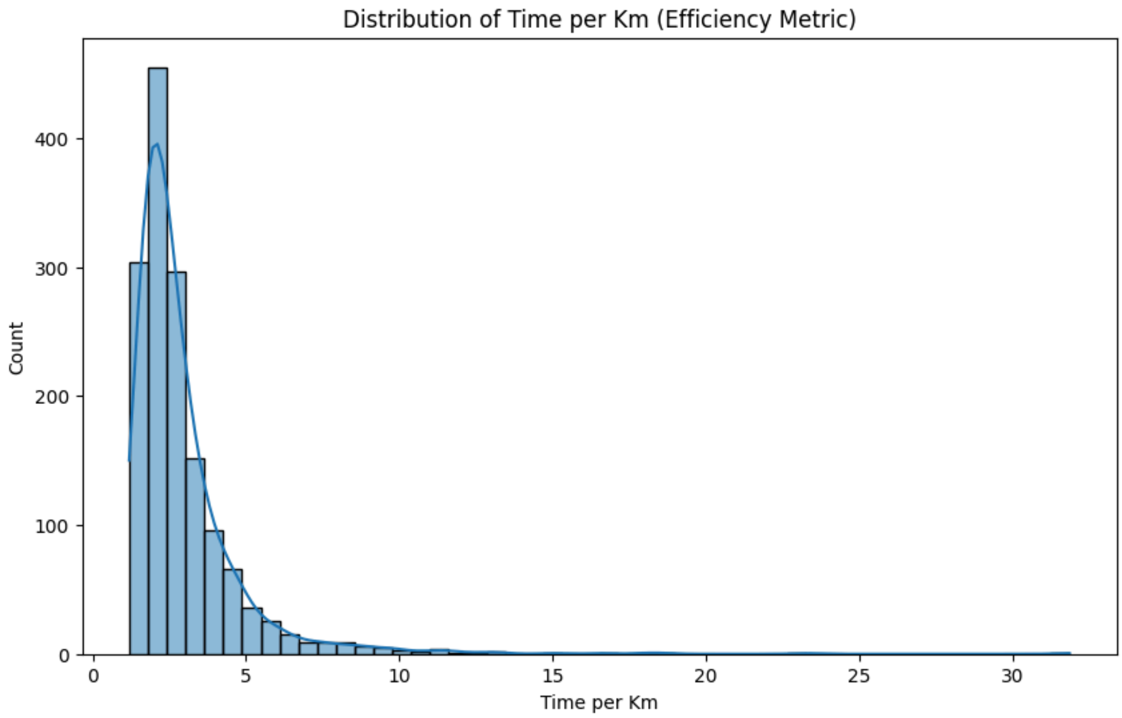
Scatterplots for Key Relationships

```
plt.figure(figsize=(10,6))
sns.scatterplot(x='avg_actual_distance', y='avg_actual_time', data=agg_route)
plt.title('Average Actual Distance vs Average Actual Time per Route')
plt.xlabel('Average Actual Distance')
plt.ylabel('Average Actual Time')
plt.show()
```



3. Time vs Distance Ratio

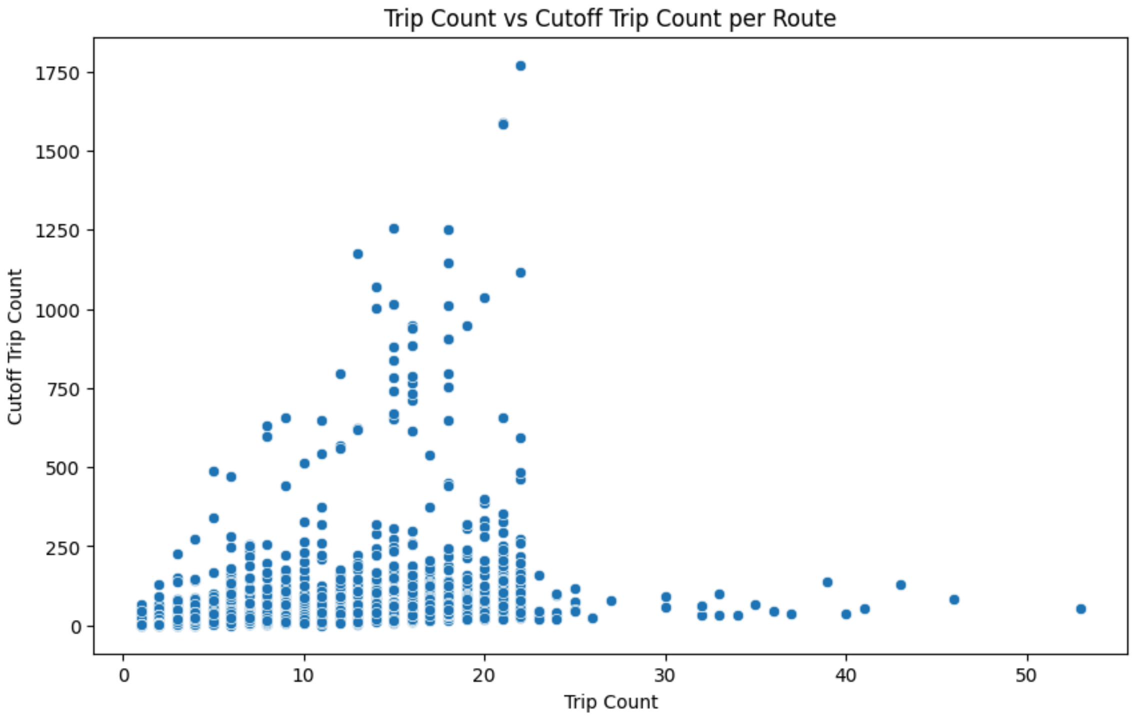
```
agg_route['time_per_km'] = agg_route['avg_actual_time'] / agg_route['avg_actual_distance']

plt.figure(figsize=(10,6))
sns.histplot(agg_route['time_per_km'], bins=50, kde=True)
plt.title('Distribution of Time per Km (Efficiency Metric)')
plt.xlabel('Time per Km')
plt.show()
```



4. Relationship between trip_count and cutoff_trip_count

```
plt.figure(figsize=(10,6))
sns.scatterplot(x='trip_count', y='cutoff_trip_count', data=agg_route)
plt.title('Trip Count vs Cutoff Trip Count per Route')
plt.xlabel('Trip Count')
```

```
plt.ylabel('Cutoff Trip Count')
plt.show()
```



*Handling categorical values *

```
# Convert Columns to 'category' dtype (memory optimization + clearer semantics)

cat_cols = ['route_schedule_uuid', 'route_type', 'trip_uuid',
            'source_center', 'source_name', 'destination_center', 'destination_name']

for col in cat_cols:
    df[col] = df[col].astype('category')
df.sample(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	c
5470	training	2018-09-12 20:24:29.188095	thanos::sroute:162f9a67-5ebe-4338-8450-1c6d57d...	FTL	trip-153678386918787011	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	
38432	training	2018-09-13 19:04:27.170139	thanos::sroute:48117ed8-ce82-41f4-9cd0-f065c10...	FTL	trip-153686546716987129	IND382430AAB	Ahmedabad_East_H_1 (Gujarat)	
105498	training	2018-09-18 02:28:11.866136	thanos::sroute:5bd33197-898a-47eb-bcde-7193b9f...	FTL	trip-153723769186587220	IND363310AAB	Dhrangadhra_NvygRDPP_D (Gujarat)	
112467	training	2018-09-20 07:00:39.868630	thanos::sroute:14c55592-ba2e-4f72-820c-3a22334...	FTL	trip-153742683986834433	IND562132AAA	Bangalore_Nelmngla_H (Karnataka)	
120588	training	2018-09-25 04:21:12.551117	thanos::sroute:96a80600-40e1-436b-9161-fa68f9e...	FTL	trip-153784927255069118	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	

5 rows × 29 columns

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

# Convert categorical columns to 'category' dtype
cat_cols = ['route_schedule_uuid', 'route_type', 'trip_uuid',
            'source_center', 'source_name', 'destination_center', 'destination_name']

for col in cat_cols:
    df[col] = df[col].astype('category')

# Convert datetime columns and extract categorical time features
df['od_start_time'] = pd.to_datetime(df['od_start_time'], errors='coerce')
df['cutoff_timestamp'] = pd.to_datetime(df['cutoff_timestamp'], errors='coerce')
```

```
df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'], errors='coerce')

df['od_start_hour'] = df['od_start_time'].dt.hour.astype('category')
df['od_start_dayofweek'] = df['od_start_time'].dt.dayofweek.astype('category')

df['cutoff_hour'] = df['cutoff_timestamp'].dt.hour.astype('category')
df['cutoff_dayofweek'] = df['cutoff_timestamp'].dt.dayofweek.astype('category')

# Label Encoding example — create a new LabelEncoder for each column inside the loop
for col in ['route_type', 'source_center', 'destination_center']:
    le = LabelEncoder()
    # Some categories might have missing values, so fillna before encoding
    df[col] = df[col].astype(str)  # Convert to string to avoid issues with categories
    df[col+'_encoded'] = le.fit_transform(df[col])

# Frequency Encoding example for high cardinality
freq = df['source_name'].value_counts(normalize=True)
df['source_name_freq_enc'] = df['source_name'].map(freq)

# One-hot encode some categorical features with low cardinality
df = pd.get_dummies(df, columns=['route_type', 'od_start_hour', 'od_start_dayofweek'], drop_first=True)
```

Column Normalization /Column Standardization

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

cont_vars = ['actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance',
             'segment_actual_time', 'segment_osrm_distance', 'segment_factor']

# Initialize scaler
scaler = StandardScaler()

# Create a copy to preserve original data
df_scaled = df.copy()

# Fit scaler on continuous variables and transform
df_scaled[cont_vars] = scaler.fit_transform(df[cont_vars])

# Check result
print(df_scaled[cont_vars].describe())
```

```
       actual_distance_to_destination   actual_time     osrm_time  \
count                    1.448670e+05  1.448670e+05  1.448670e+05
mean                    -1.035892e-16  3.060591e-17 -1.903060e-17
std                      1.000003e+00  1.000003e+00  1.000003e+00
min                     -6.524076e-01 -6.820372e-01 -6.748750e-01
25%                     -6.107952e-01 -6.118150e-01 -6.066954e-01
50%                     -4.868181e-01 -4.763865e-01 -4.865695e-01
75%                      1.525716e-01  1.606290e-01  1.400335e-01
max                      4.908490e+00  6.880224e+00  4.779493e+00

       osrm_distance  segment_actual_time  segment_osrm_distance  \
count   1.448670e+05         1.448670e+05           1.448670e+05
mean   -5.414892e-17         2.722160e-18          -9.750730e-17
std     1.000003e+00         1.000003e+00           1.000003e+00
min    -6.548359e-01        -5.230372e+00          -1.278178e+00
25%    -6.051907e-01        -3.023300e-01          -6.023829e-01
50%    -4.897572e-01        -1.343285e-01           3.829548e-02
75%     1.387307e-01         7.100653e-02           2.790629e-01
max     4.847640e+00         5.627682e+01           1.214167e+02

       segment_factor
count    1.448670e+05
mean    -2.285634e-17
std      1.000003e+00
min     -5.294016e+00
25%     -1.795853e-01
50%     -1.101921e-01
75%      6.525331e-03
max      1.180052e+02
```

Business Insights

1. **Top Sources and Destinations of Orders**

Most orders originate from certain source_center or source_name. Identify these top centers contributing the bulk of shipments.

Similarly, check destination_center or destination_name for key delivery hotspots.

This helps the business focus resources like staffing and vehicles on high-traffic centers.

2. **Busiest Corridors*

A corridor can be defined as a combination of (source_center, destination_center).

Identify which corridors have the highest volume of trips.

Knowing the busiest corridors helps optimize route planning and fleet allocation.

3. **Average Distance and Time per Corridor**

Calculate the average actual distance and average actual delivery time per corridor.

Compare actual time with OSRM estimated time to identify efficiency or delays.

Long distances with short delivery times may indicate express services, while long times suggest bottlenecks.

4. **Order Volume by Time**

Analyze the od_start_time and trip_creation_time to find peak hours and days for orders.

This helps in scheduling staff and vehicles to meet demand spikes.

5. **Cutoff Impact**

Use is_cutoff and cutoff_factor fields to assess how cutoff rules affect delivery schedules.

Identify how often cutoff policies cause delays or cancellations.

6. **Outliers and Exceptions**

Outliers in actual_time or actual_distance_to_destination could indicate exceptional cases like traffic jams, vehicle breakdowns, or incorrect data.

Investigate these for operational improvements.

7. **Accuracy of Estimated vs Actual Metrics**

Compare actual_time and osrm_time, actual_distance_to_destination and osrm_distance.

Identify if the route planning estimates are reliable or if adjustments are needed.

8. **Segment-Level Insights**

Analyze segment times and distances to understand which route segments consistently cause delays.

Optimize specific segments with infrastructure improvements or alternate routes.

9. **State-Level or Regional Trends**

If source_center and destination_center map to states or regions, identify states with highest order volumes or longest delivery times.

Tailor marketing or operational efforts regionally.

10. **Recommendations for Capacity Planning**

From volume, time, and distance insights, recommend fleet size, warehouse locations, and driver shifts to optimize delivery performance.

Recommendations

1-Focus Resources on Busy Locations

Increase staffing and vehicle availability at the top source and destination centers where most orders are coming from to ensure smooth and faster deliveries.

2-Optimize Busiest Routes

Prioritize route planning and vehicle assignment for the busiest corridors to reduce delays and improve delivery efficiency.

3-Adjust Schedules to Peak Times

Schedule more drivers and support staff during peak order times (like mid-morning to early afternoon) to handle increased demand without slowdowns.

4-Review and Improve Delivery Estimates

Since actual delivery times are often longer than the estimated times, update the planning system to better reflect real-world conditions and avoid customer disappointment.

5-Address Delays in Problem Segments

Identify and focus on route segments that consistently cause delays. Consider alternative routes or additional support to minimize hold-ups.

6-Manage Cutoff Times Smartly

Reevaluate cutoff policies to reduce their impact on delivery delays, making sure orders are processed timely without unnecessary hold-ups.

7-Plan for Outliers Have contingency plans for exceptional cases like traffic jams or vehicle issues to quickly respond and minimize delivery disruptions.

8-Expand Fleet or Shift Coverage

Based on delivery volume patterns, increase the number of delivery vehicles or extend driver shifts to meet demand without overtime delays.

9-Regional Focus for Growth

Target marketing and operational improvements in regions or states with the highest order volumes to grow business effectively.

10-Continuous Monitoring and Feedback

Regularly track delivery times and distances to spot new issues early and continuously improve operations.

Recommendations

1-Focus Resources on Busy Locations

Increase staffing and vehicle availability at the top source and destination centers where most orders are coming from to ensure smooth and faster deliveries.

2-Optimize Busiest Routes

Prioritize route planning and vehicle assignment for the busiest corridors to reduce delays and improve delivery efficiency.

3-Adjust Schedules to Peak Times

Schedule more drivers and support staff during peak order times (like mid-morning to early afternoon) to handle increased demand without slowdowns.

4-Review and Improve Delivery Estimates