Business Case: Delhivery - Feature Engineering

D A Santhosh

Loading the Dataset

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

```
# Load the dataset
file_path = '/content/delhivery_data.csv'
# Change this to your actual file path
data = pd.read_csv(file_path)
```

```
# Display the first few rows of the dataset
data.head()
data.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
```

Handling Missing Values and Convert Timestamps

```
# Handle missing values by filling them with a placeholder
data['source_name'].fillna('Unknown', inplace=True)
data['destination_name'].fillna('Unknown', inplace=True)
```

```
# Convert timestamp columns to datetime, handling errors
data['trip_creation_time'] = pd.to_datetime(data['trip_creation_time'])
data['od_start_time'] = pd.to_datetime(data['od_start_time'])
data['od_end_time'] = pd.to_datetime(data['od_end_time'])
data['cutoff_timestamp'] = pd.to_datetime(data['cutoff_timestamp'], errors='coerce')
```

```
# Extract features from timestamps
data['trip_creation_year'] = data['trip_creation_time'].dt.year
data['trip_creation_month'] = data['trip_creation_time'].dt.month
data['trip_creation_day'] = data['trip_creation_time'].dt.day
data['trip_creation_hour'] = data['trip_creation_time'].dt.hour
data['trip_creation_minute'] = data['trip_creation_time'].dt.minute
data['trip_creation_second'] = data['trip_creation_time'].dt.second
```

```
data['od_start_year'] = data['od_start_time'].dt.year
data['od_start_month'] = data['od_start_time'].dt.month
data['od_start_day'] = data['od_start_time'].dt.day
data['od_start_hour'] = data['od_start_time'].dt.hour
data['od_start_minute'] = data['od_start_time'].dt.minute
data['od_start_second'] = data['od_start_time'].dt.second
```

```
data['od_end_year'] = data['od_end_time'].dt.year
data['od_end_month'] = data['od_end_time'].dt.month
data['od_end_day'] = data['od_end_time'].dt.day
data['od_end_hour'] = data['od_end_time'].dt.hour
data['od_end_minute'] = data['od_end_time'].dt.minute
data['od_end_second'] = data['od_end_time'].dt.second
```

```
# Extract location information from source and destination names
data[['source_city', 'source_state']] = data['source_name'].str.extract(r'([^_]+)_.+\(([^)]+)\)')
data[['destination_city', 'destination_state']] = data['destination_name'].str.extract(r'([^_]+)_.+\(([^)]+)\)')
```

```
# Display the first few rows to verify the changes
data.head()
```

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

5 rows × 46 columns

Aggregation and Merging

```
# Group by trip_uuid, source_center, and destination_center
aggregated_data = data.groupby(['trip_uuid', 'source_center', 'destination_center']).agg({
    'trip_creation_time': 'first',
    'route_schedule_uuid': 'first',
    'route_type': 'first',
    'source_name': 'first',
    'destination_name': 'first',
    'od_start_time': 'first',
    'od_end_time': 'last',
    'start_scan_to_end_scan': 'sum',
    'actual_distance_to_destination': 'sum',
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'factor': 'mean',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
    'segment_factor': 'mean',
    'source_state': 'first',  # Include source_state in the aggregation
    'destination_state': 'first'  # Include destination_state in the aggregation
}).reset_index()

# Display the first few rows of the aggregated data
aggregated_data.head()
```

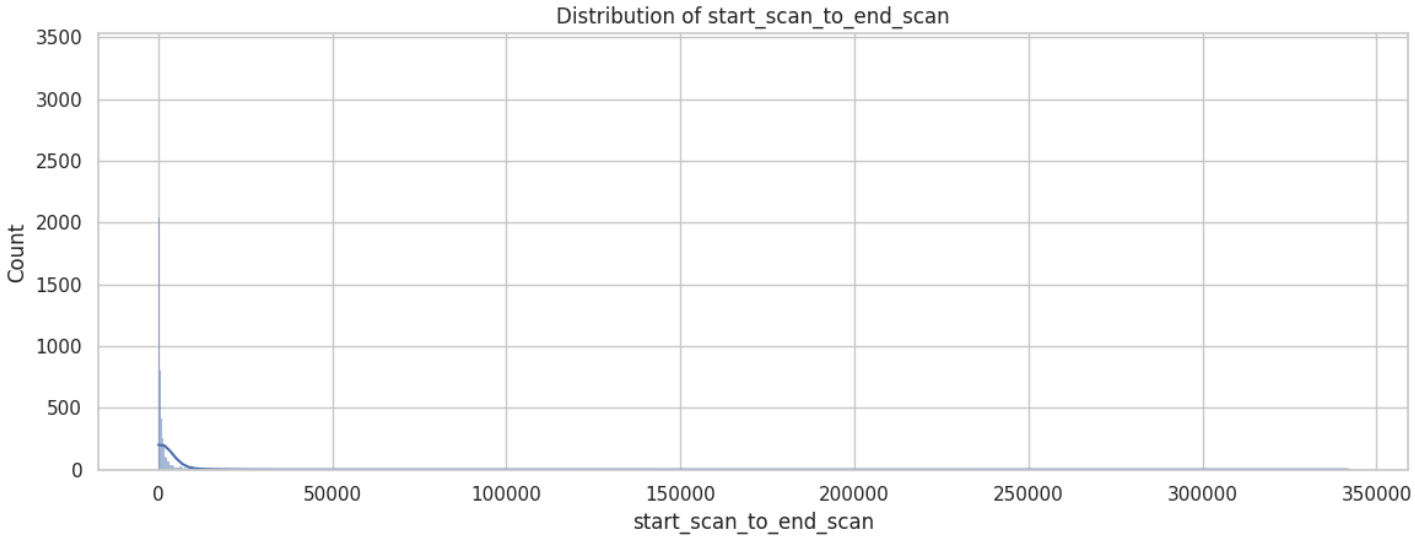
	trip_uuid	source_center	destination_center	trip_creation_time	route_schedule_uuid	route_type	sourc
0	trip-153671041653548748	IND209304AAA	IND000000ACB	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	Kanpur_Cent (Uttar P
1	trip-153671041653548748	IND462022AAA	IND209304AAA	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	Bhopal_Trn (Madhya P
2	trip-153671042288605164	IND561203AAB	IND562101AAA	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	Doddablpur_Chika (Ka
3	trip-153671042288605164	IND572101AAA	IND561203AAB	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Carting	Tumkur_Ve (Ka
4	trip-153671043369099517	IND000000ACB	IND160002AAC	2018-09-12 00:00:33.691250	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	Gurgaon_Bilas (H

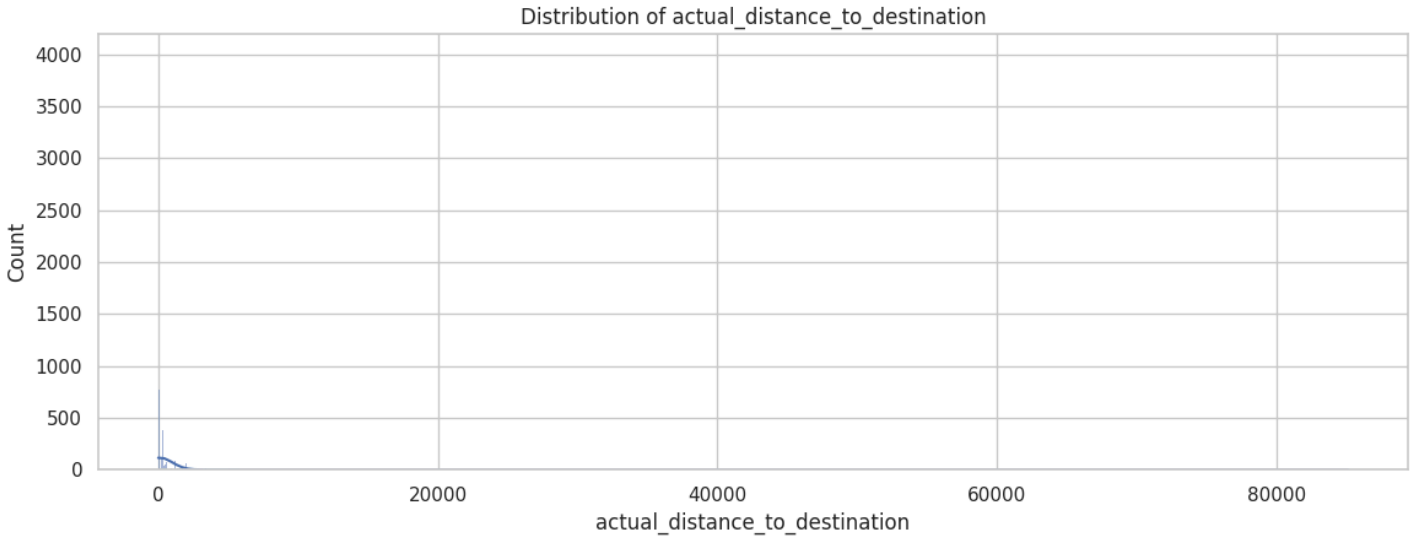
5 rows × 22 columns

Exploratory Data Analysis (EDA)

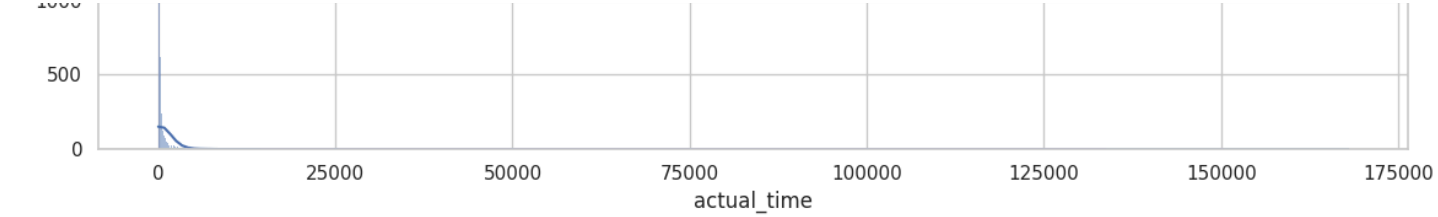
```
# Set up the matplotlib figure aesthetics
sns.set(style="whitegrid")

# Univariate analysis for numerical variables
numerical_columns = [
    'start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time',
    'osrm_time', 'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
    'segment_osrm_distance'
]
```

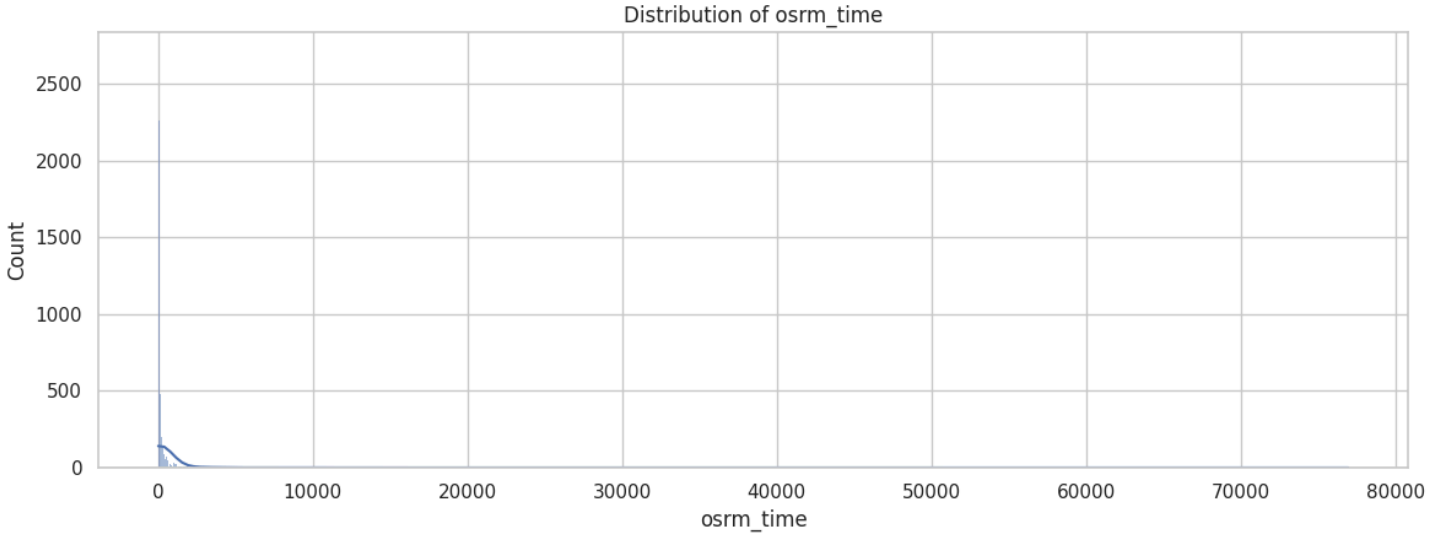
```
# Plot histograms for numerical columns
fig, axes = plt.subplots(len(numerical_columns), 1, figsize=(12, 36))
for ax, col in zip(axes, numerical_columns):
    sns.histplot(aggregated_data[col].dropna(), kde=True, ax=ax)
    ax.set_title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

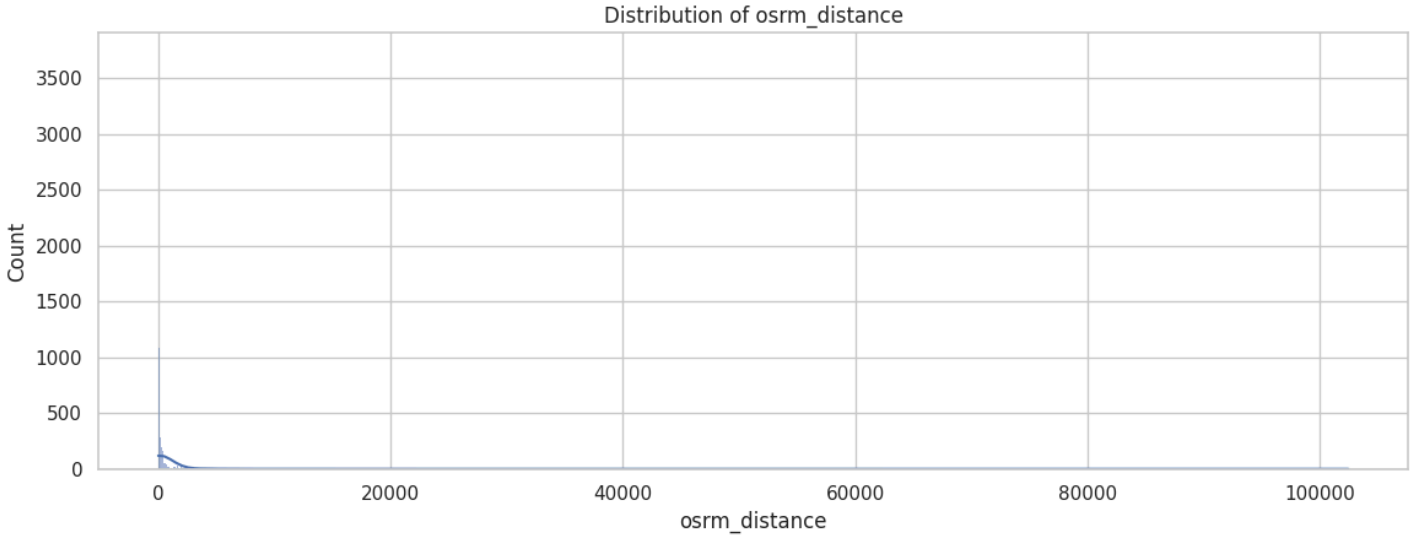

Distribution of start_scan_to_end_scan
Count
3500
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
200000
250000
300000
350000
start_scan_to_end_scan

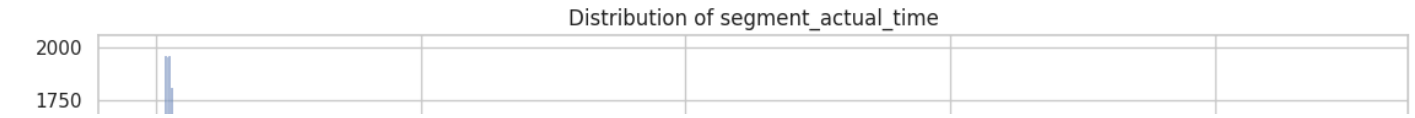

Distribution of actual_distance_to_destination
Count
4000
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
actual_distance_to_destination

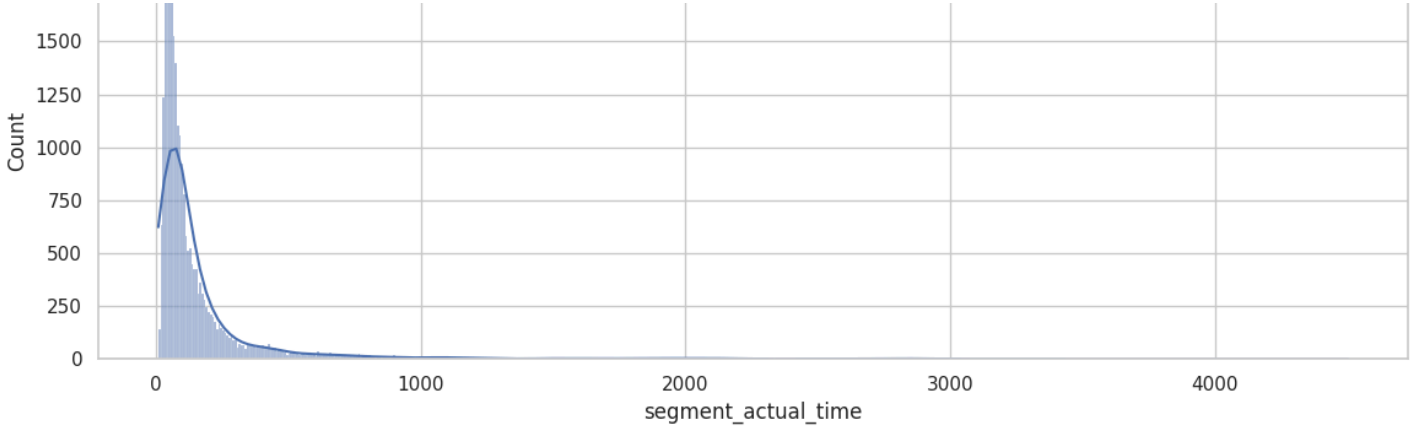

Distribution of actual_time
Count
2500
2000
1500
1000

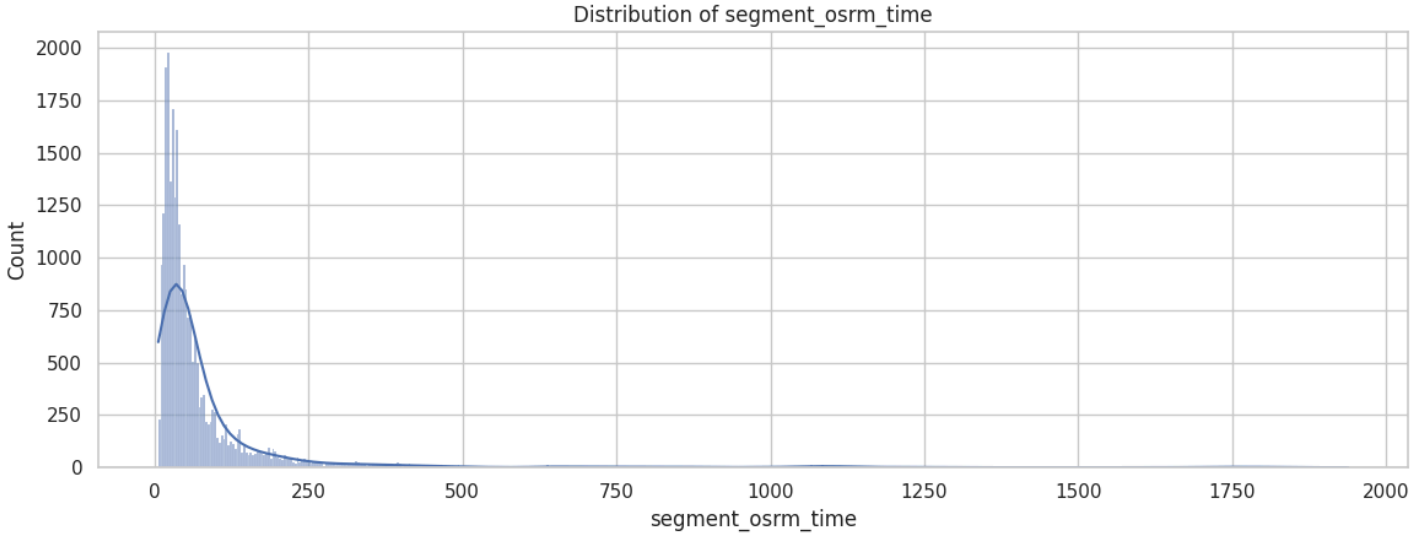

500
0
0
25000
50000
75000
100000
125000
150000
175000
actual_time

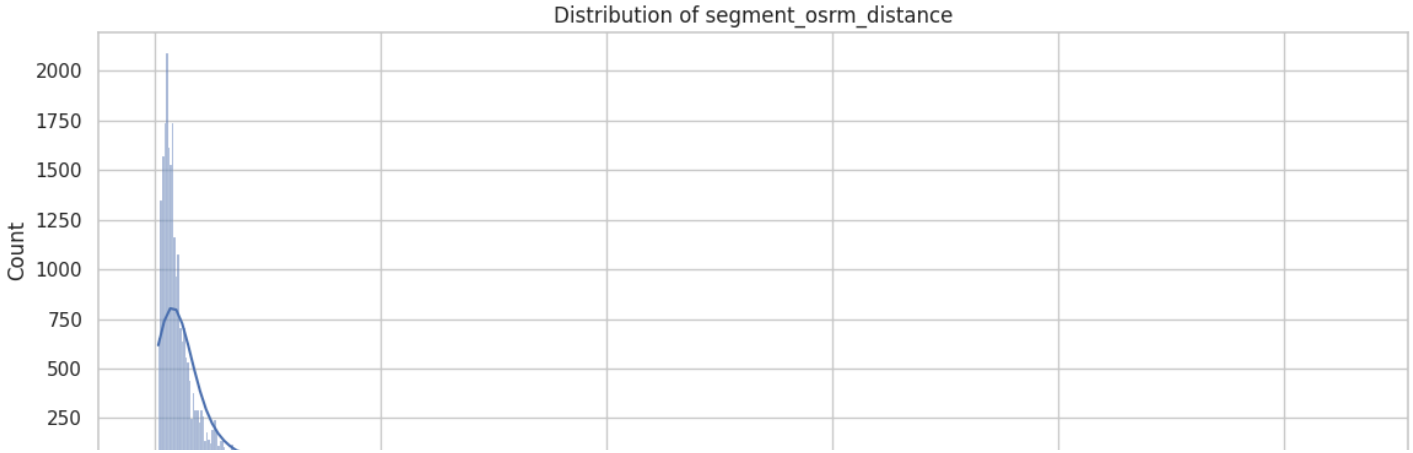

Distribution of osrm_time
Count
2500
2000
1500
1000
500
0
0
10000
20000
30000
40000
50000
60000
70000
80000
osrm_time

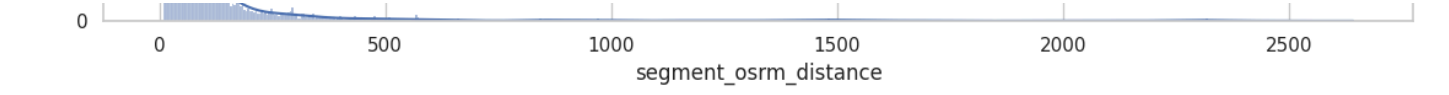

Distribution of osrm_distance
Count
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
100000
osrm_distance


Distribution of segment_actual_time
2000
1750

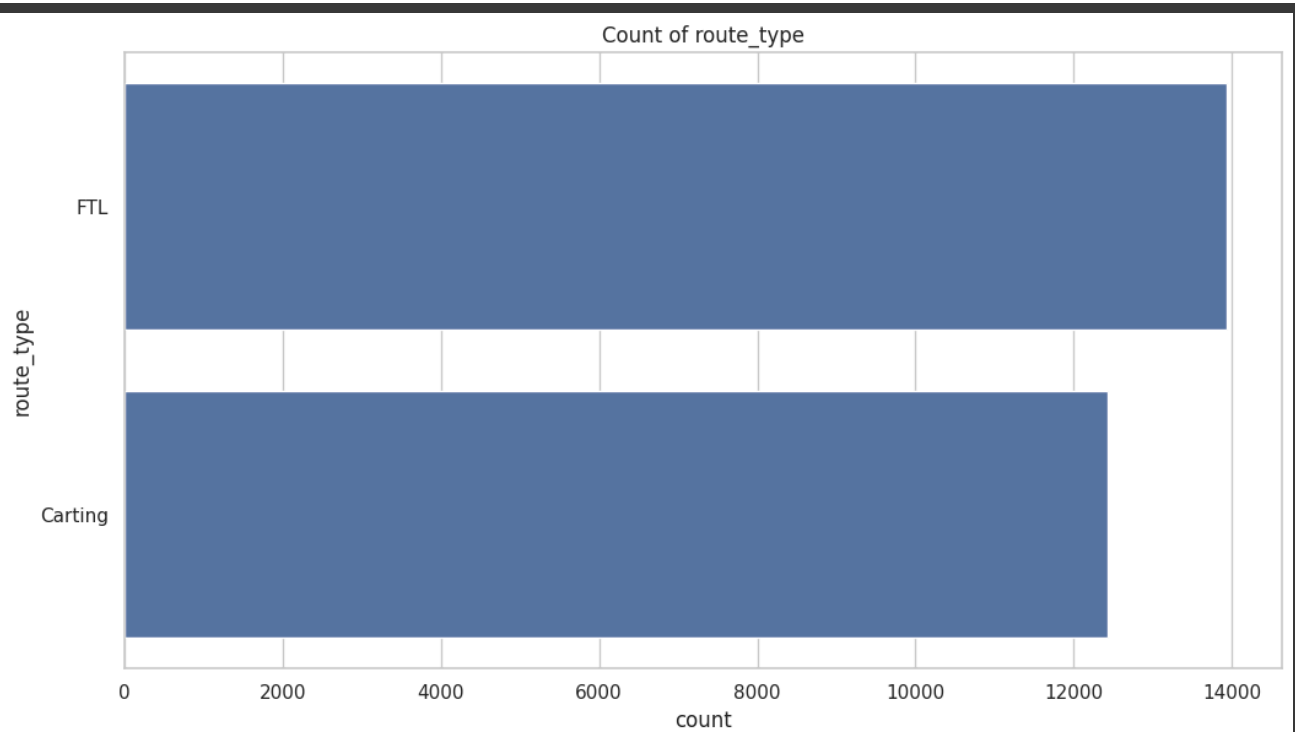

Count
1500
1250
1000
750
500
250
0
0
1000
2000
3000
4000
segment_actual_time

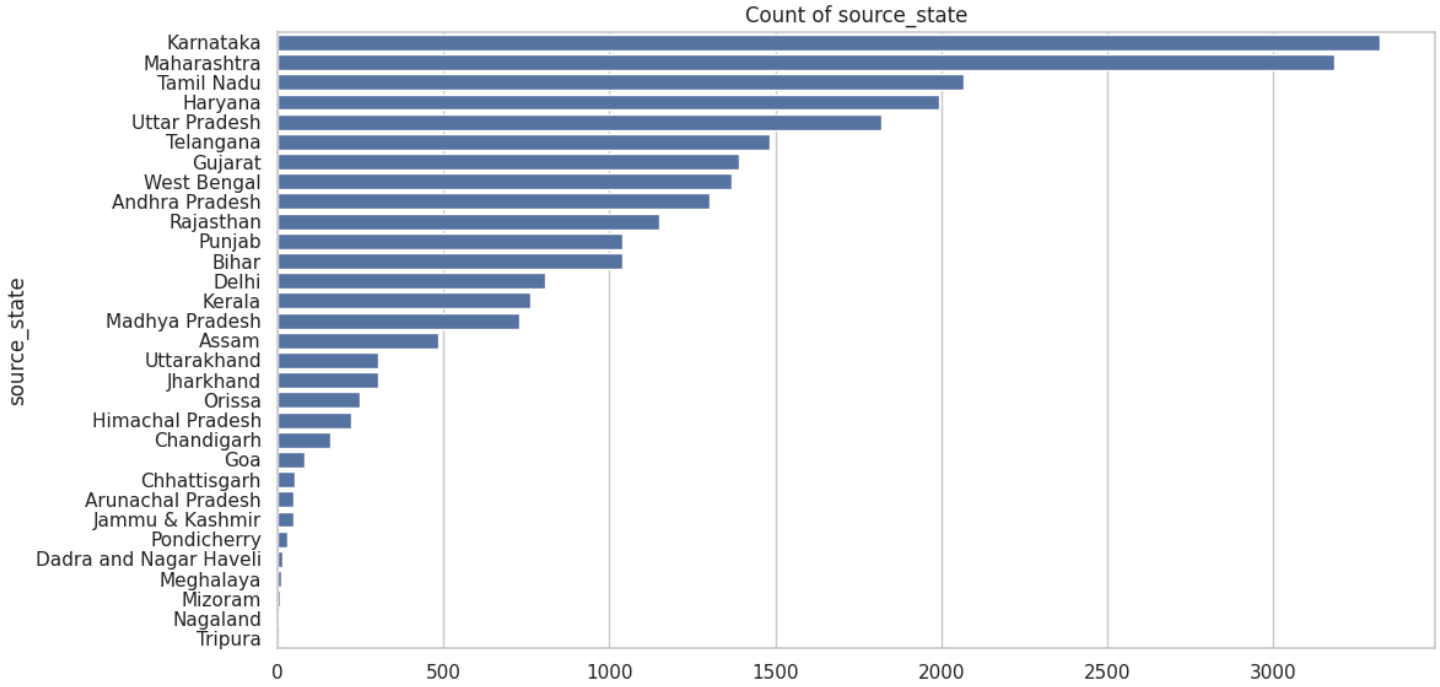

Distribution of segment_osrm_time
Count
2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000
segment_osrm_time

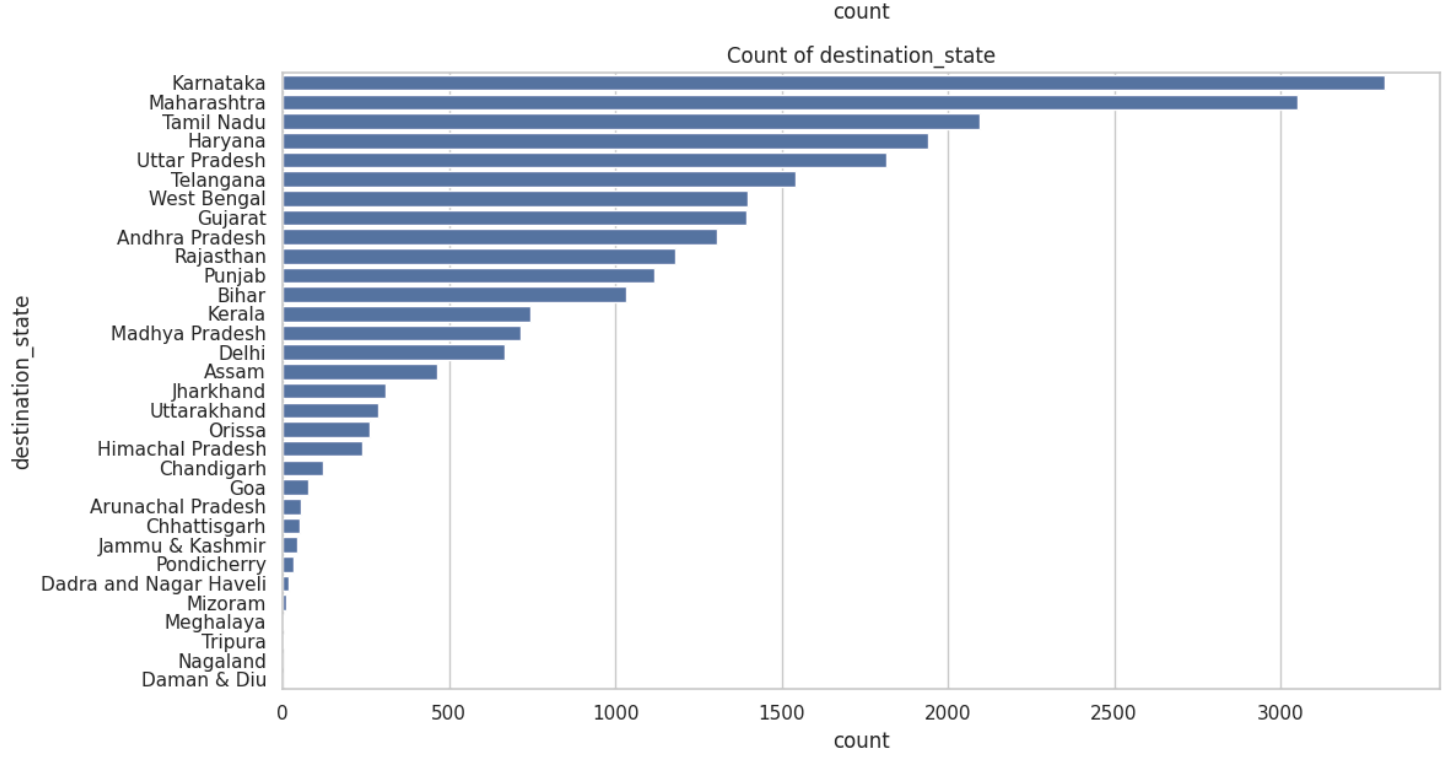

Distribution of segment_osrm_distance
Count
2000
1750
1500
1250
1000
750
500
250




```
# Univariate analysis for categorical variables
categorical_columns = ['route_type', 'source_state', 'destination_state']

# Plot countplots for categorical columns
fig, axes = plt.subplots(len(categorical_columns), 1, figsize=(12, 18))
for ax, col in zip(axes, categorical_columns):
    sns.countplot(data=aggregated_data, y=col, order=aggregated_data[col].value_counts().index, ax=ax)
    ax.set_title(f'Count of {col}')
plt.tight_layout()
plt.show()
```

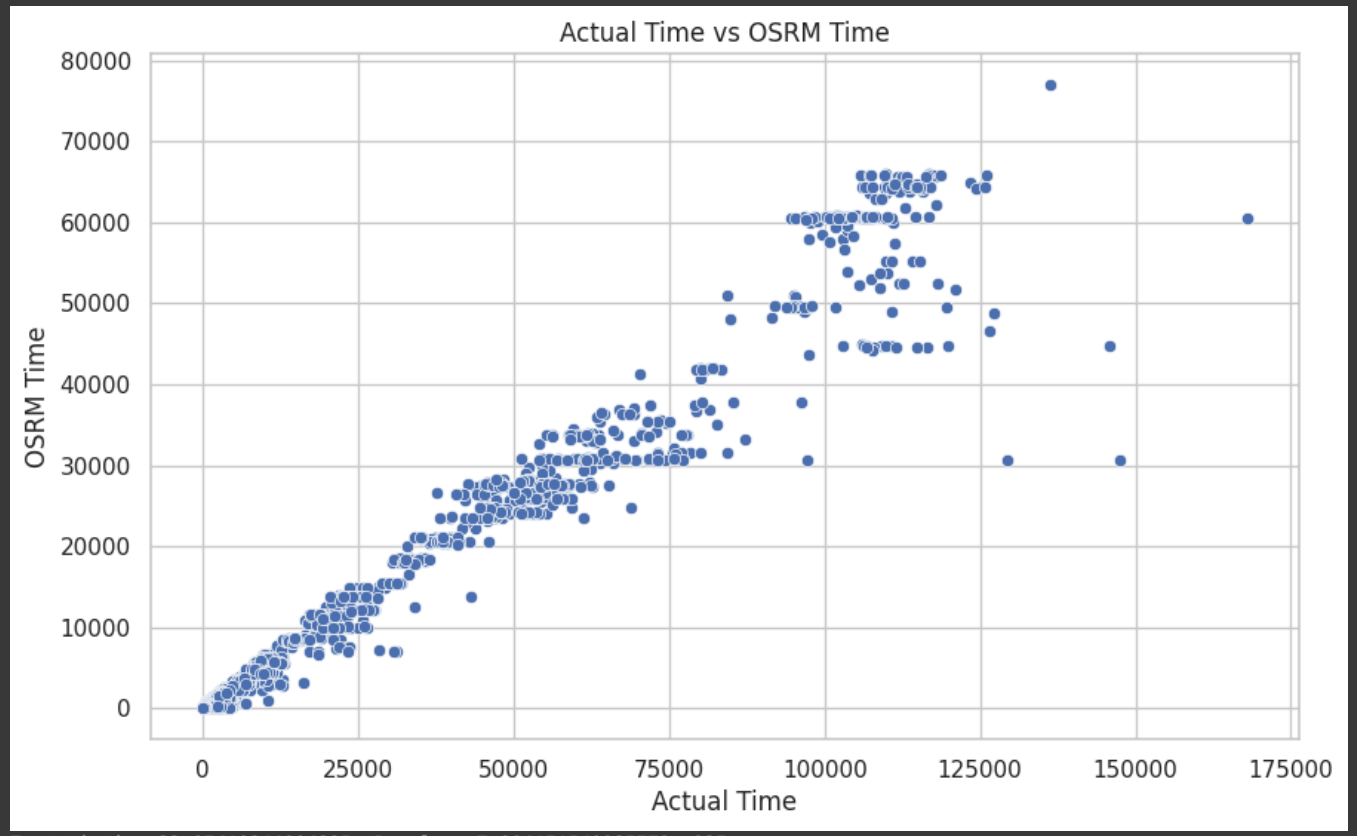

Count of route_type
route_type
FTL
Carting
0
2000
4000
6000
8000
10000
12000
14000
count


Count of source_state
source_state
Karnataka
Maharashtra
Tamil Nadu
Haryana
Uttar Pradesh
Telangana
Gujarat
West Bengal
Andhra Pradesh
Rajasthan
Punjab
Bihar
Delhi
Kerala
Madhya Pradesh
Assam
Uttarakhand
Jharkhand
Orissa
Himachal Pradesh
Chandigarh
Goa
Chhattisgarh
Arunachal Pradesh
Jammu & Kashmir
Pondicherry
Dadra and Nagar Haveli
Meghalaya
Mizoram
Nagaland
Tripura
0
500
1000
1500
2000
2500
3000



Hypothesis Testing and Visual Analysis

```
# Hypothesis testing and visual analysis
# Example: Compare actual_time and osrm_time


# Scatter plot to compare actual_time and osrm_time
plt.figure(figsize=(10, 6))
sns.scatterplot(x=aggregated_data['actual_time'], y=aggregated_data['osrm_time'])
plt.title('Actual Time vs OSRM Time')
plt.xlabel('Actual Time')
plt.ylabel('OSRM Time')
plt.show()

# Perform hypothesis testing (paired t-test)
t_stat, p_value = stats.ttest_rel(aggregated_data['actual_time'], aggregated_data['osrm_time'])
print(f'T-statistic: {t_stat}, P-value: {p_value}')
```




```
T-statistic: 33.05610864906825, P-value: 7.904151560985772e-235
```

Outlier Detection and Handling

```
# Identify and handle outliers using the IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Apply the function to numerical columns
for col in numerical_columns:
    aggregated_data = remove_outliers(aggregated_data, col)

# Display the dataset after outlier removal
aggregated_data.describe()
```

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_
count	17103	17103	17103	17103.000000	17103.000000	17103.00
mean	2018-09-22 19:01:17.842078464	2018-09-22 22:05:27.023269632	2018-09-23 00:21:43.344737792	376.892943	61.976152	134.14
min	2018-09-12 00:00:22.886430	2018-09-12 00:00:22.886430	2018-09-12 00:50:10.814399	22.000000	9.001351	9.00
25%	2018-09-17 13:12:52.009181440	2018-09-17 16:03:30.969369600	2018-09-17 19:11:17.876992512	152.000000	25.727122	72.00
50%	2018-09-22 15:16:26.623631104	2018-09-22 17:09:26.963151872	2018-09-22 20:19:18.454433024	267.000000	51.727305	112.00
75%	2018-09-27 23:28:49.024899072	2018-09-28 03:19:34.660721920	2018-09-28 04:51:38.700667904	480.000000	86.554789	179.00
max	2018-10-03 23:59:42.701692	2018-10-05 01:01:19.051758	2018-10-05 02:44:50.858859	2855.000000	217.777567	480.00
std	NaN	NaN	NaN	343.274771	40.424156	83.47

Categorical Encoding

```
# Perform one-hot encoding for categorical variables
encoded_data = pd.get_dummies(aggregated_data, columns=['route_type', 'source_state', 'destination_state'])

# Display the first few rows of the encoded data
encoded_data.head()
```

	trip_uuid	source_center	destination_center	trip_creation_time	route_schedule_uuid	source_name	
2	trip-153671042288605164	IND561203AAB	IND562101AAA	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Doddablpur_ChikaDPP_D (Karnataka)	C
3	trip-153671042288605164	IND572101AAA	IND561203AAB	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Tumkur_Veersagr_I (Karnataka)	Dod
6	trip-153671046011330457	IND400072AAB	IND401104AAA	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Mumbai Hub (Maharashtra)	
7	trip-153671052974046625	IND583101AAA	IND583201AAA	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	Bellary_Dc (Karnataka)	
8	trip-153671052974046625	IND583119AAA	IND583101AAA	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	Sandur_WrdN1DPP_D (Karnataka)	B

5 rows × 78 columns

Normalization/Standardization

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Normalize/Standardize numerical features
scaler = MinMaxScaler()  # or StandardScaler()
scaled_columns = ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time',
                  'osrm_time', 'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
                  'segment_osrm_distance']

encoded_data[scaled_columns] = scaler.fit_transform(encoded_data[scaled_columns])

# Display the dataset after scaling
encoded_data.head()
```

	trip_uuid	source_center	destination_center	trip_creation_time	route_schedule_uuid	source_name	
2	trip-153671042288605164	IND561203AAB	IND562101AAA	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Doddablpur_ChikaDPP_D (Karnataka)	C
3	trip-153671042288605164	IND572101AAA	IND561203AAB	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...	Tumkur_Veersagr_I (Karnataka)	Dod
6	trip-153671046011330457	IND400072AAB	IND401104AAA	2018-09-12 00:01:00.113710	thanos::sroute:f0176492-a679-4597-8332-bbd1c7f...	Mumbai Hub (Maharashtra)	

Business Insights and Recommendations

```
# Extract business insights and make recommendations
# Example: Check from where most orders are coming from (State)
```