


✓ Delhivery: Feature Engineering Case Study

By Parth Patel

This project aimed to enhance Delhivery's data processing capabilities by developing a robust system for cleaning, transforming, and analyzing logistics data. As the largest logistics provider in India, Delhivery sought to leverage data to improve operational efficiency and forecasting accuracy. The project involved using advanced data engineering techniques to extract meaningful insights from complex datasets. By implementing feature engineering methods and statistical analyses, the project significantly improved data quality and facilitated the development of predictive models, ultimately contributing to more informed decision-making and streamlined operations.

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from scipy.stats import ttest_ind, f_oneway
from scipy.stats import ttest_rel
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
df = pd.read_csv("delhivery_data.csv")
df.head(2)
```



	data	trip_creation_time	route_schedule_uuid	route_type	trip_uu:
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78- b351-4c0e-a951- fa3d5c3...	Carting	1537410936476493; tri
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78- b351-4c0e-a951- fa3d5c3...	Carting	1537410936476493; tri


2 rows x 24 columns

```
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
```

Statistics Summary


```
df.describe()
```



	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	
count	144867.000000	144867.000000	144867.000000	1
mean	961.262986	232.926567	234.073372	
std	1037.012769	344.755577	344.990009	
min	20.000000	9.000000	9.000045	
25%	161.000000	22.000000	23.355874	
50%	449.000000	66.000000	66.126571	
75%	1634.000000	286.000000	286.708875	
max	7898.000000	1927.000000	1927.447705	

✓ Checking Missing Values and Duplicate Values for Customers

```
print(f"Duplicate rows in customers data: {df.duplicated().sum()}")
```

 Duplicate rows in customers data: 0

```
print(f"Duplicate rows in customers data: {df.isnull().sum()}")
```

```
↗ Duplicate rows in customers data: 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
```

```
df['source_name'].fillna('Unknown', inplace=True)
df['destination_name'].fillna('Unknown', inplace=True)
```

```
df.isnull().sum()
```

```
⇒ data      0
   trip_creation_time  0
   route_schedule_uuid  0
   route_type  0
   trip_uuid  0
   source_center  0
   source_name  0
   destination_center  0
   destination_name  0
   od_start_time  0
   od_end_time  0
   start_scan_to_end_scan  0
   is_cutoff  0
   cutoff_factor  0
   cutoff_timestamp  0
   actual_distance_to_destination  0
   actual_time  0
   osrm_time  0
   osrm_distance  0
   factor  0
   segment_actual_time  0
   segment_osrm_time  0
   segment_osrm_distance  0
   segment_factor  0
dtype: int64
```

✓ Converting time columns into datetime

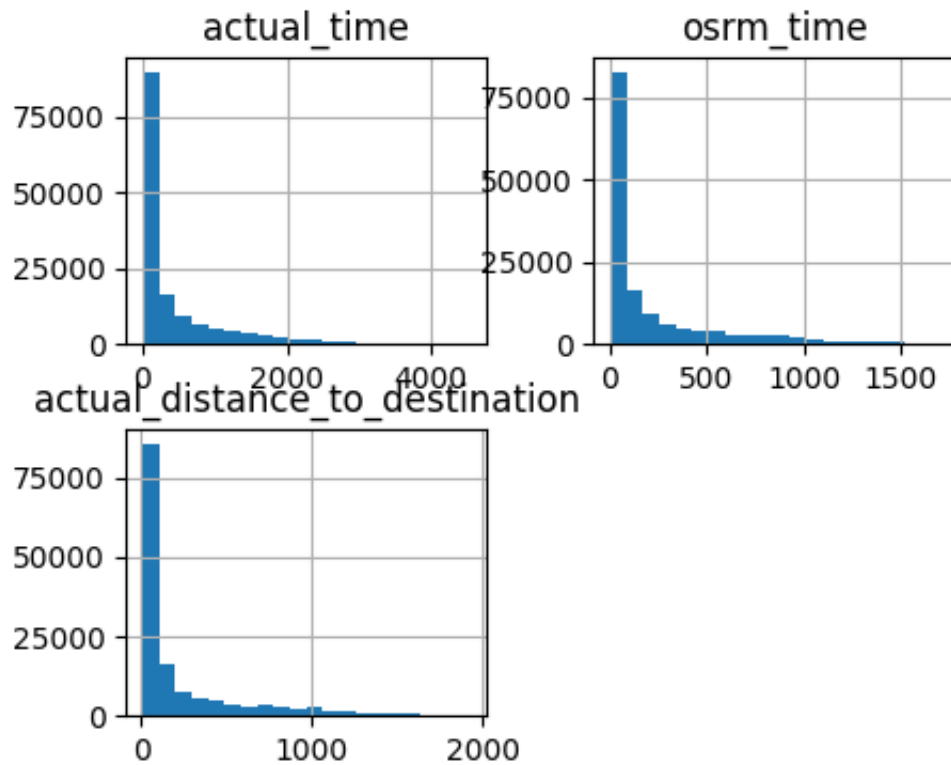
```
df['trip_creation_time'] = pd.to_datetime(df['trip_creation_time'])
df['od_start_time'] = pd.to_datetime(df['od_start_time'])
df['od_end_time'] = pd.to_datetime(df['od_end_time'])
```

✓ Convert categorical columns to category type

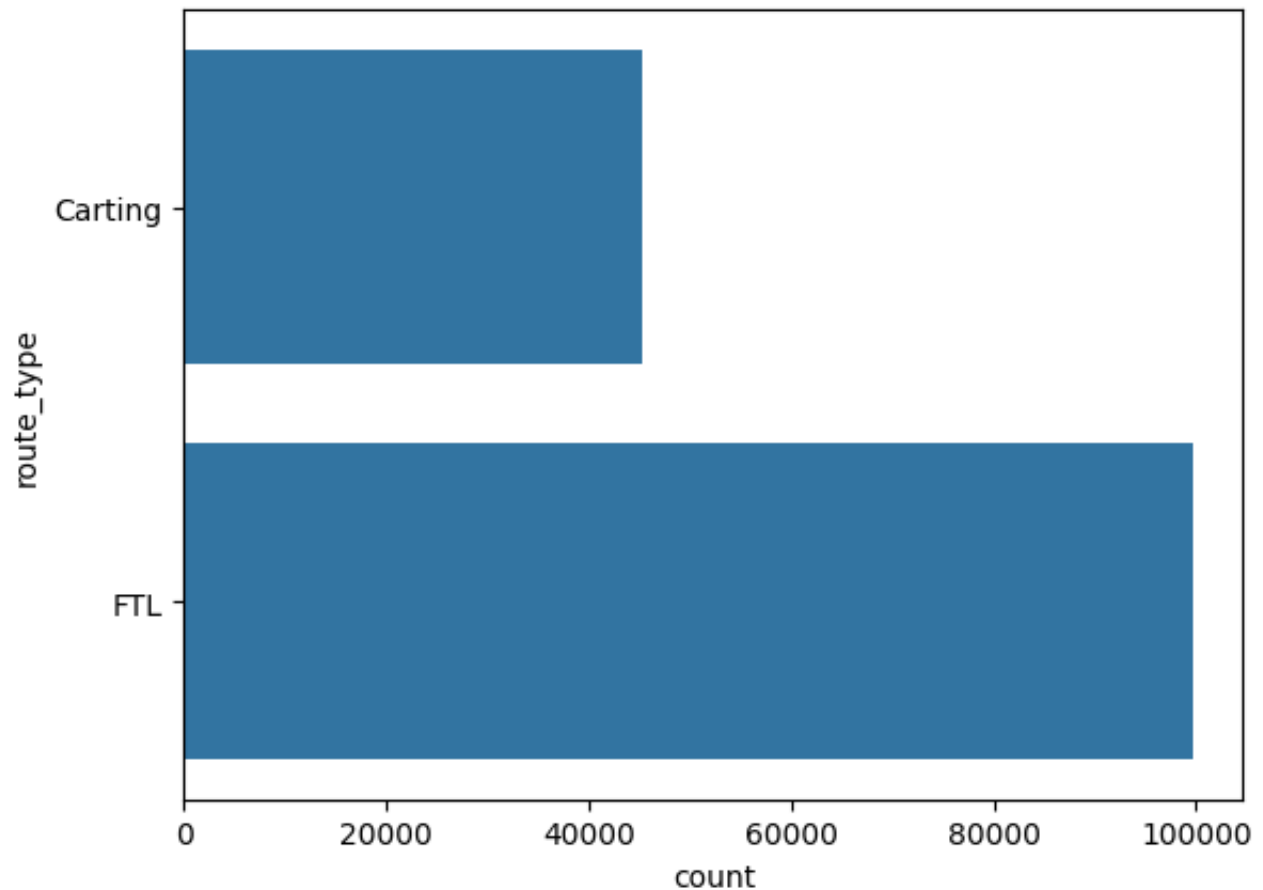
```
df['route_type'] = df['route_type'].astype('category')
df['source_name'] = df['source_name'].astype('category')
df['destination_name'] = df['destination_name'].astype('category')
```

✓ Exploratory Data Analysis (EDA)

```
df[['actual_time', 'osrm_time', 'actual_distance_to_destination']].hist(bins=20,  
plt.show())
```



```
#Checking route type usage  
sns.countplot(y='route_type', data=df)  
plt.show()
```



```
# corr = df.corr()  
# plt.figure(figsize=(12, 8))  
# sns.heatmap(corr, annot=True, cmap='coolwarm')  
# plt.show()
```

✓ Merging Rows


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

```
↔ 0      trip-153741093647649320_IND388121AAA_IND388620AAB
   1      trip-153741093647649320_IND388121AAA_IND388620AAB
   2      trip-153741093647649320_IND388121AAA_IND388620AAB
   3      trip-153741093647649320_IND388121AAA_IND388620AAB
   4      trip-153741093647649320_IND388121AAA_IND388620AAB
      ...
144862 trip-153746066843555182_IND131028AAB_IND000000ACB
144863 trip-153746066843555182_IND131028AAB_IND000000ACB
144864 trip-153746066843555182_IND131028AAB_IND000000ACB
144865 trip-153746066843555182_IND131028AAB_IND000000ACB
144866 trip-153746066843555182_IND131028AAB_IND000000ACB
Name: segment_key, Length: 144867, dtype: object
```


✓ Use groupby and aggregations like cumsum()

```
df['segment_actual_time_sum'] = df.groupby('segment_key')['segment_actual_time'].cumsum()
df['segment_osrm_distance_sum'] = df.groupby('segment_key')['segment_osrm_distance'].cumsum()
df['segment_osrm_time_sum'] = df.groupby('segment_key')['segment_osrm_time'].cumsum()
```

✓ Aggregating at Segment Level


```
create_segment_dict = {
    'trip_uuid': 'first',
    'source_center': 'first',
    'destination_center': 'first',
    'segment_actual_time_sum': 'last',
    'segment_osrm_distance_sum': 'last',
    'segment_osrm_time_sum': 'last',
    'od_start_time': 'first',
    'od_end_time': 'last'
}
```

```
df_segment_aggregated = df.groupby('segment_key').agg(create_segment_dict).reset_
df_segment_aggregated.head()
```



	segment_key	trip_uuid	source_cente
0	trip- 153671041653548748_IND209304AAA_IND000000ACB	trip- 153671041653548748	IND209304AA
1	trip- 153671041653548748_IND462022AAA_IND209304AAA	trip- 153671041653548748	IND462022AA
2	trip- 153671042288605164_IND561203AAB_IND562101AAA	trip- 153671042288605164	IND561203AA
3	trip- 153671042288605164_IND572101AAA_IND561203AAB	trip- 153671042288605164	IND572101AA
4	trip- 153671043369099517_IND000000ACB_IND160002AAC	trip- 153671043369099517	IND000000AC

```
df_segment_aggregated.sort_values(by=['segment_key', 'od_end_time']).head()
```



	segment_key	trip_uuid	source_cente
0	trip- 153671041653548748_IND209304AAA_IND000000ACB	trip- 153671041653548748	IND209304AA
1	trip- 153671041653548748_IND462022AAA_IND209304AAA	trip- 153671041653548748	IND462022AA
2	trip- 153671042288605164_IND561203AAB_IND562101AAA	trip- 153671042288605164	IND561203AA
3	trip- 153671042288605164_IND572101AAA_IND561203AAB	trip- 153671042288605164	IND572101AA
4	trip- 153671043369099517_IND000000ACB_IND160002AAC	trip- 153671043369099517	IND000000AC

✓ Feature Engineering

```
df["od_time_diff_hour"] = df["od_end_time"]-df["od_start_time"]
df[['od_start_time', 'od_end_time', 'od_time_diff_hour']].head()
```



	od_start_time	od_end_time	od_time_diff_hour
0	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	0 days 01:26:12.818197
1	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	0 days 01:26:12.818197
2	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	0 days 01:26:12.818197
3	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	0 days 01:26:12.818197
4	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797	0 days 01:26:12.818197

✓ Splitting Destination Name

```
# Split destination_name to extract city, place code, and state
df[['destination_city', 'destination_place_code', 'destination_state']] = df['des'

# Display the first few rows to check the new columns
df[['destination_name', 'destination_city', 'destination_place_code', 'destinatio
```



	destination_name	destination_city	destination_place_code	destination_
0	Khambhat_MotvdDPP_D (Gujarat)	Khambhat_MotvdDPP	D	(
1	Khambhat_MotvdDPP_D (Gujarat)	Khambhat_MotvdDPP	D	(
2	Khambhat_MotvdDPP_D (Gujarat)	Khambhat_MotvdDPP	D	(

```
# Split destination_name to extract city, place code, and state
df[['source_city', 'source_place_code', 'source_state']] = df['source_name'].str.sp

# Display the first few rows to check the new columns
df[['source_name', 'source_city', 'source_place_code', 'source_state']].head()
```



	source_name	source_city	source_place_code	source_state
0	Anand_VUNagar_DC (Gujarat)	Anand_VUNagar	DC	Gujarat
1	Anand_VUNagar_DC (Gujarat)	Anand_VUNagar	DC	Gujarat
2	Anand_VUNagar_DC (Gujarat)	Anand_VUNagar	DC	Gujarat
3	Anand_VUNagar_DC (Gujarat)	Anand_VUNagar	DC	Gujarat
4	Anand_VUNagar_DC (Gujarat)	Anand_VUNagar	DC	Gujarat

```
# df[['destination_city', 'destination_place', 'destination_state']] = df['destination_name'].str.sp
# df[['source_city', 'source_place', 'source_state']] = df['source_name'].str.sp
```

✓ Extract date features from trip_creation_time

```
df['year'] = df['trip_creation_time'].dt.year
df['month'] = df['trip_creation_time'].dt.month
df['day'] = df['trip_creation_time'].dt.day
df['hour'] = df['trip_creation_time'].dt.hour
df['minute'] = df['trip_creation_time'].dt.minute
df['second'] = df['trip_creation_time'].dt.second
```

```
df[['trip_creation_time', 'year', 'month', 'day', 'hour', 'minute', 'second']].head()
```



	trip_creation_time	year	month	day	hour	minute	second
0	2018-09-20 02:35:36.476840	2018	9	20	2	35	36
1	2018-09-20 02:35:36.476840	2018	9	20	2	35	36
2	2018-09-20 02:35:36.476840	2018	9	20	2	35	36
3	2018-09-20 02:35:36.476840	2018	9	20	2	35	36
4	2018-09-20 02:35:36.476840	2018	9	20	2	35	36

4. In-depth analysis:

✓ Grouping and Aggregating at Trip-level

```
aggregation_functions = {
    'source_center': 'first',
    'destination_center': 'last',
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'actual_distance_to_destination': 'sum',
    'osrm_distance': 'sum'
}
```

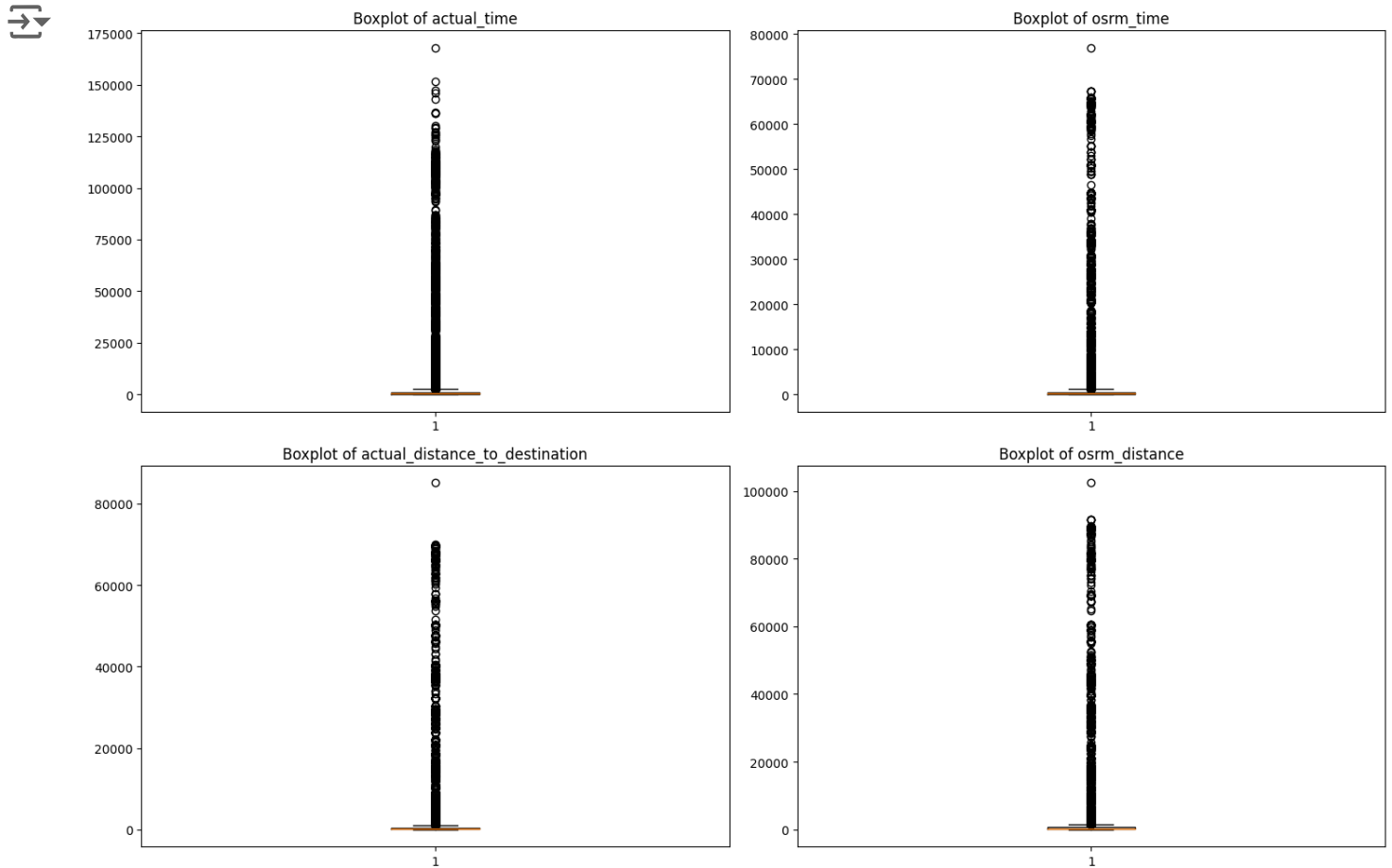
```
trip_data = df.groupby('trip_uuid').agg(aggregation_functions).reset_index()
trip_data.head()
```



	trip_uuid	source_center	destination_center	actual_time	osrm_time
0	trip-153671041653548748	IND462022AAA	IND000000ACB	15682.0	7787.0
1	trip-153671042288605164	IND572101AAA	IND562101AAA	399.0	210.0
2	trip-153671043369099517	IND562132AAA	IND160002AAC	112225.0	65768.0
3	trip-153671046011330457	IND400072AAB	IND401104AAA	82.0	24.0

✓ Outlier Detection using Boxplot

```
numerical_features = ['actual_time', 'osrm_time', 'actual_distance_to_destination']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 2, i)
    plt.boxplot(trip_data[feature].dropna())
    plt.title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```



✓ Handling Outliers using IQR method

```

for feature in numerical_features:
    Q1 = trip_data[feature].quantile(0.25)
    Q3 = trip_data[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    trip_data = trip_data[(trip_data[feature] >= lower_bound) & (trip_data[feature] <= upper_bound)]

```

✓ One-Hot Encoding of Categorical Features

```

if 'route_type' in trip_data.columns:
    trip_data_encoded = pd.get_dummies(trip_data, columns=['route_type'])
else:
    print("Column 'route_type' not found in the DataFrame after aggregation.")

```

➡ Column 'route_type' not found in the DataFrame after aggregation.

```

scaler = MinMaxScaler()
trip_data_scaled = trip_data.copy()
trip_data_scaled[numerical_features] = scaler.fit_transform(trip_data[numerical_features])

```

```
trip_data_scaled.head()
```

➡

	trip_uuid	source_center	destination_center	actual_time	osrm_time
1	trip-153671042288605164	IND572101AAA	IND562101AAA	0.162907	0.317757
3	trip-153671046011330457	IND400072AAB	IND401104AAA	0.030493	0.028037
4	trip-153671052974046625	IND583101AAA	IND583101AAA	0.228488	0.313084
5	trip-153671055416136166	IND600116AAB	IND602105AAB	0.034670	0.037383
6	trip-153671066201138152	IND600044AAD	IND600048AAA	0.006266	0.010903

Double-click (or enter) to edit

```
trip_data.columns
```

```
⇒ Index(['trip_uuid', 'source_center', 'destination_center', 'actual_time',  
        'osrm_time', 'actual_distance_to_destination', 'osrm_distance'],  
        dtype='object')
```

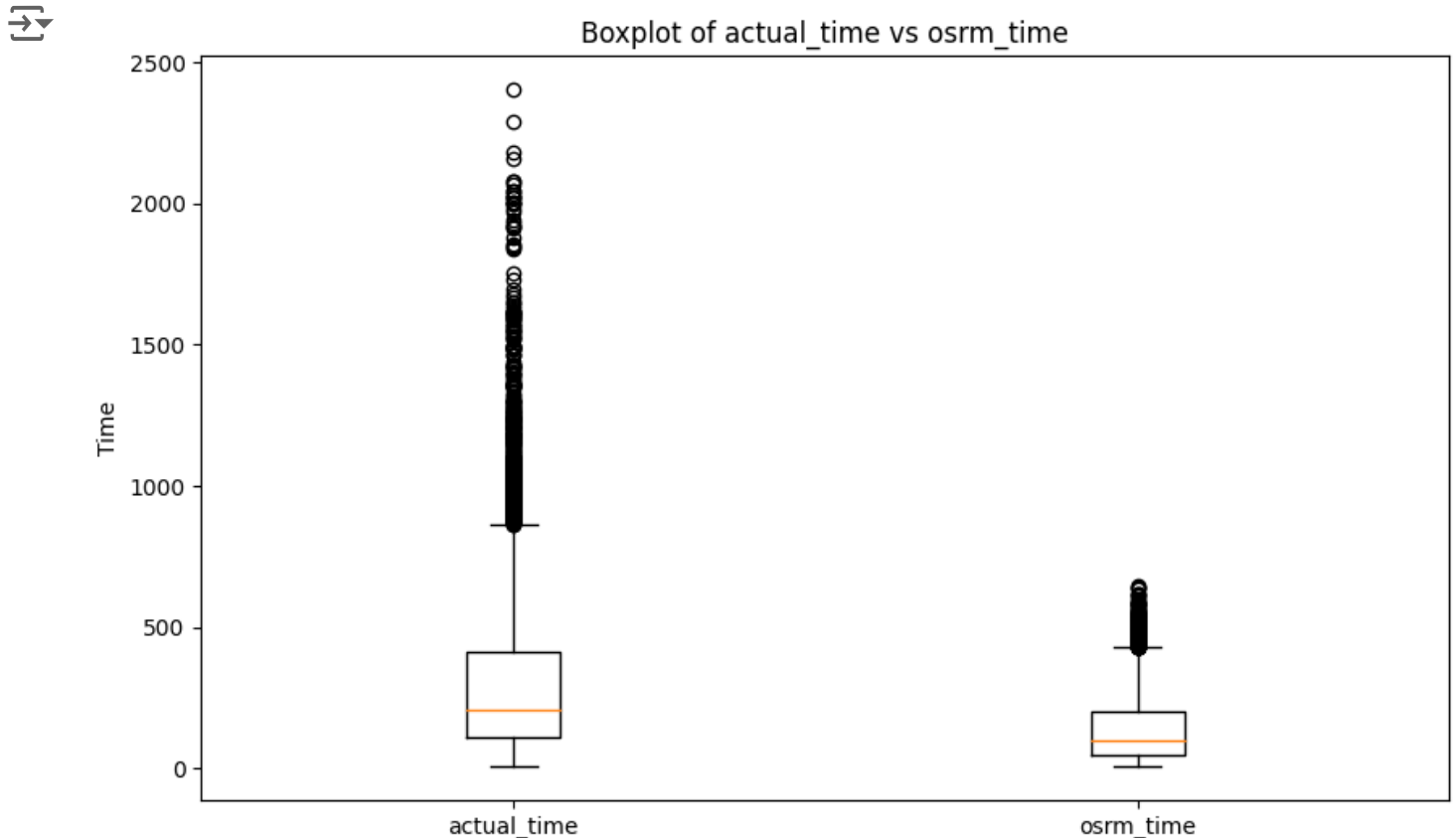
▼ Hypothesis Testing

```
stat, p_value = ttest_rel(trip_data['actual_time'].dropna(), trip_data['osrm_time'])
```

```
print(f'actual_time vs osrm_time: t-statistic = {stat:.3f}, p-value = {p_value:.3f}')
```

```
⇒ actual_time vs osrm_time: t-statistic = 86.074, p-value = 0.000
```

```
plt.figure(figsize=(10, 6))
plt.boxplot([trip_data['actual_time'].dropna(), trip_data['osrm_time'].dropna()], 1
plt.title('Boxplot of actual_time vs osrm_time')
plt.ylabel('Time')
plt.show()
```



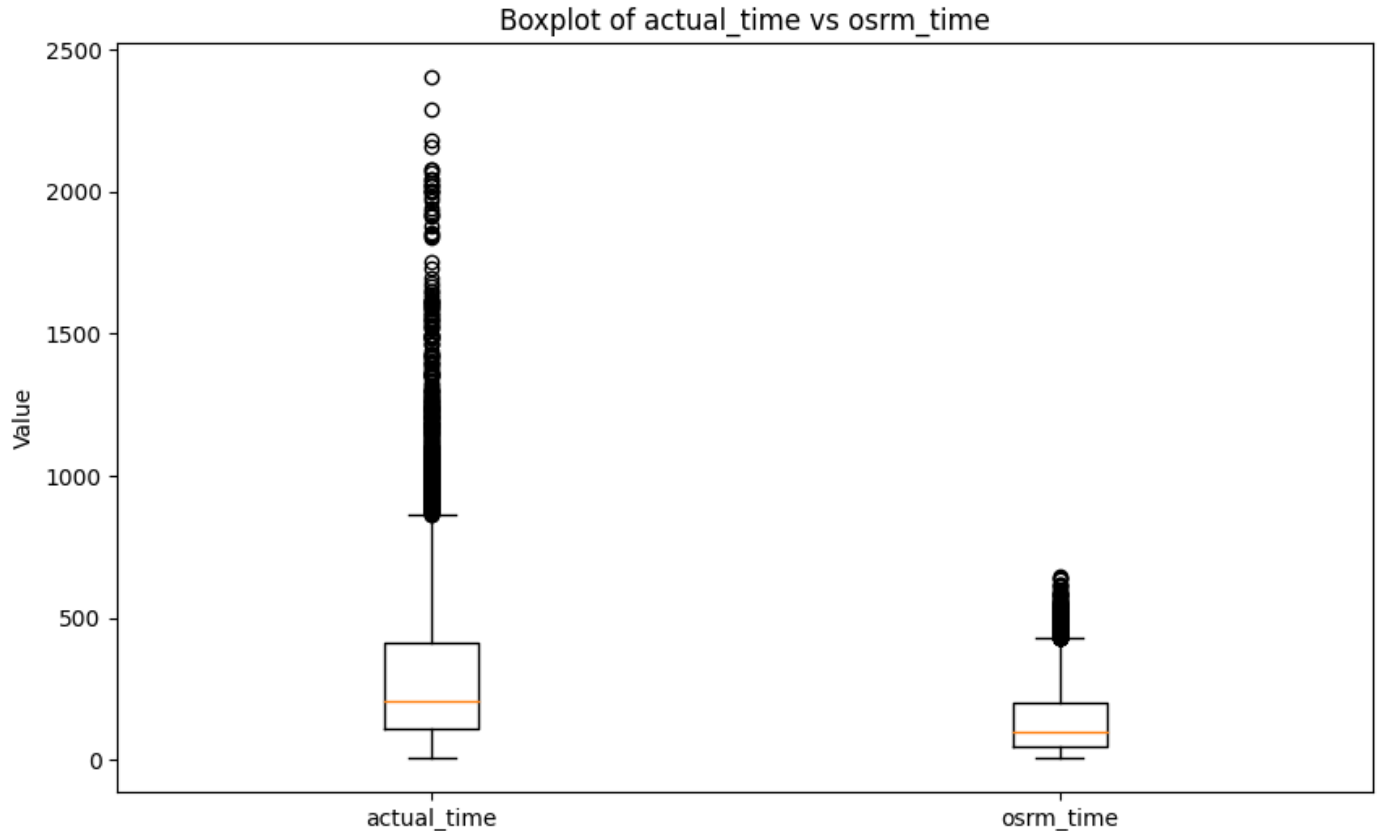
```
def perform_analysis(column1, column2, data):
    if column1 in data.columns and column2 in data.columns:
        stat, p_value = ttest_rel(data[column1].dropna(), data[column2].dropna())
        print(f'{column1} vs {column2}: t-statistic = {stat:.3f}, p-value = {p_value:.3f}')

    plt.figure(figsize=(10, 6))
    plt.boxplot([data[column1].dropna(), data[column2].dropna()], labels=[column1, column2])
    plt.title(f'Boxplot of {column1} vs {column2}')
    plt.ylabel('Value')
```

```
plt.show()
else:
    print(f"One or both columns {column1}, {column2} are not in the DataFrame")
```

```
# Hypothesis testing and visual analysis for each pair
perform_analysis('actual_time', 'osrm_time', trip_data)
perform_analysis('actual_time', 'segment_actual_time', trip_data)
perform_analysis('osrm_distance', 'segment_osrm_distance', trip_data)
perform_analysis('osrm_time', 'segment_osrm_time', trip_data)
```

↔ actual_time vs osrm_time: t-statistic = 86.074, p-value = 0.000



One or both columns actual_time, segment_actual_time are not in the DataFrame.
 One or both columns osrm_distance, segment_osrm_distance are not in the DataFrame.
 One or both columns osrm_time, segment_osrm_time are not in the DataFrame.

Insights:

Busiest Corridors/Routes: By analyzing the source and destination cities/states, you can identify the busiest routes or corridors for Delhivery's operations. This information can be obtained by grouping the data by source-destination pairs and counting the number of trips or total shipment volume for each pair. **Average Distance and Time:** After identifying the busiest corridors, you can calculate the average distance and time taken for shipments on those routes. This can be done by grouping the data by source-destination pairs and aggregating the `actual_distance_to_destination` and `actual_time` columns. **Route Type Usage:** The analysis shows that the 'FTL' (Full Truck Load) route type is more prevalent than 'Carting'. This could indicate that Delhivery handles more long-haul shipments compared to short-distance deliveries within cities. **Temporal Patterns:** By analyzing the extracted date and time features from the `trip_creation_time` column, you can identify patterns in shipment volumes based on month, day of the week, or time of day. This information can help optimize resource allocation and staffing.

Actionable Recommendations:

Route Optimization: Based on the analysis of busiest corridors and average distances/times, Delhivery can optimize its route planning and scheduling. For high-volume routes, they may consider dedicating more resources or exploring alternative transportation modes to improve efficiency and reduce transit times. **Fleet Management:** The insights from route type usage can inform Delhivery's fleet management strategy. If long-haul shipments dominate, they may need to invest in more full-truck load vehicles or explore partnerships with third-party logistics providers for specific routes. **Resource Allocation:** By understanding temporal patterns in shipment volumes, Delhivery can allocate resources more effectively. For example, during peak periods or seasons, they may need to increase staffing levels or adjust work schedules to meet demand. **Data Quality Improvements:** The data cleaning and feature engineering steps highlighted areas where data quality could be improved. Delhivery can work on improving the consistency and completeness of data captured, especially for fields like `cutoff_factor`, `cutoff_timestamp`, and `factor`, which were not well understood in this analysis. **Predictive Modeling:** With the cleaned and engineered dataset, Delhivery can explore building predictive models for tasks such as estimating transit times, predicting demand, or forecasting resource requirements. These models can further enhance operational efficiency and planning. **Hypothesis Testing and Monitoring:** The hypothesis testing performed in the analysis can be extended to other operational metrics or Key Performance Indicators (KPIs). Delhivery should continue monitoring

deviations between planned and actual metrics, investigating root causes, and implementing corrective actions as needed. Continuous Improvement: The analysis provided a foundation for understanding and processing Delhivery's data. However, as operations evolve and new data sources become available, the company should regularly revisit its data engineering pipelines, feature engineering techniques, and analytical approaches to drive continuous improvement in logistics operations.