Delhivery - Feature Engineering

1) **Basic data cleaning and exploration:**

a) Handle missing values in the data:

i)
```
df.isnull().sum() / len(df) * 100
```
ii) This Python code is used to identify the number of null values in each column.

iii) I identified missing values in several columns during the analysis.

source_name	0.202254
destination_center	0.000000
destination_name	0.180165

iv) To handle them, I used the fillna() function to fill the null values appropriately.

v)
```
df = df.fillna("Unknown")
```

b) Analyse the structure of the data:

i)
```
df.shape[0], df.shape[1]

(144867, 24)
```
ii) This Python code is used to identify the shape of the data.

iii)
```
df.info()
```
iv) This Python code is used to identify the data types of each column.

v)
```
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
```

c) Try merging the rows using the hint mentioned above:

i) Grouped the data using inbuilt groupby() function based on Trip_uuid, Source ID, and Destination ID,Later, aggregated further at the Trip_uuid level.

ii) Applied aggregation functions like sum(), unique() and count().

iii)
```
df.groupby("source_center")['source_name'].unique()
```

iv)
```
df.groupby("trip_uuid")['actual_time'].count()
```

v)
```
df.groupby("trip_uuid")['source_name'].sum()
```

2) Build some features to prepare the data for actual analysis. Extract features from the below fields:

a) Destination Name: Split and extract features out of destination. City-place-code (State):

i) I split the Destination Name column into three parts — City, Place Code, and State — to better understand the location details for analysis.

ii)
```
df['destination_city'] = df['destination'].str.extract(r'\((.*?)\)')
df['destination_place'] = df['destination'].str.extract(r'/([^()]+) \(')
df['destination_code'] = df['destination'].str.extract(r'^([^/]+)')
```

iii)

	destination_city	destination_place	destination_code
0	Gujarat	Khambhat_MotvdDPP_D	IND388620AAB
1	Gujarat	Khambhat_MotvdDPP_D	IND388620AAB
2	Gujarat	Khambhat_MotvdDPP_D	IND388620AAB
3	Gujarat	Khambhat_MotvdDPP_D	IND388620AAB
4	Gujarat	Khambhat_MotvdDPP_D	IND388620AAB

b) Source Name: Split and extract features out of destination. City-place-code (State):

i) I split the Source Name column into three parts — City, Place Code, and State — to better understand the location details for analysis.

ii)
```
df['source_city'] = df['source'].str.extract(r'\((.*?)\)')
df['source_place'] = df['source'].str.extract(r'/([^()]+) \(')
df['source_code'] = df['source'].str.extract(r'^([^/]+)')
```

iii)

	source_city	source_place	source_code
0	Gujarat	Anand_VUNagar_DC	IND388121AAA
1	Gujarat	Anand_VUNagar_DC	IND388121AAA
2	Gujarat	Anand_VUNagar_DC	IND388121AAA
3	Gujarat	Anand_VUNagar_DC	IND388121AAA
4	Gujarat	Anand_VUNagar_DC	IND388121AAA

c) Trip_creation_time: Extract features like month, year and day etc:

i) I extracted features like month, year, and day from the Trip_creation_time column to help analyze time-based patterns in the data.

ii)

```
df['trip_month'] = df['trip_creation_time'].dt.month
df['trip_year'] = df['trip_creation_time'].dt.year
df['trip_day'] = df['trip_creation_time'].dt.day
df['trip_time'] = df['trip_creation_time'].dt.hour * 3600
```

iii)

	trip_month	trip_year	trip_day	trip_time
0	9	2018	20	7200
1	9	2018	20	7200
2	9	2018	20	7200
3	9	2018	20	7200
4	9	2018	20	7200

3) In-depth analysis and feature engineering:

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

i) I calculated the time taken between od_start_time and od_end_time and added it as a new column

ii)

```
df['time_taken'] = df['od_end_time'] - df['od_start_time']
df['time_taken'] = df['time_taken'].dt.total_seconds() / 60
df['time_taken'].round(0)
```

iii)

	time_taken
0	86.0
1	86.0
2	86.0
3	86.0
4	86.0

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

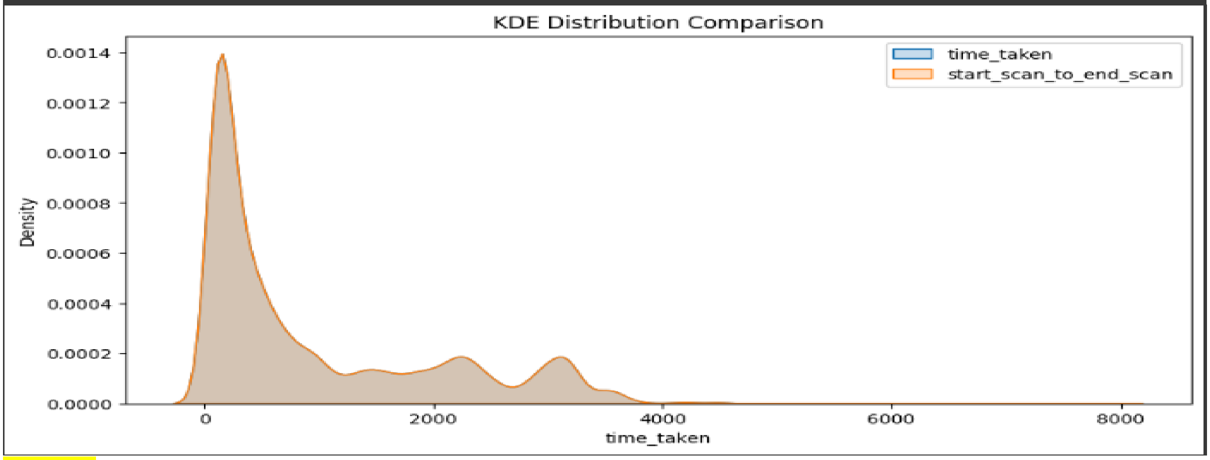
i) I compared the time taken between Time_taken with start_scan_to_end_scan using hypothesis testing and visual analysis to check if there's a significant difference between them.

ii)

```
from scipy.stats import ttest_rel, ttest_ind
t_stat, p_value = ttest_rel(df['time_taken'], df['start_scan_to_end_scan'])
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: 651.1832057297116
p-value: 0.0
```

iii)



iv) Insights - Both time_taken and start_scan_to_end_scan show similar patterns, with most values being low and a few long-duration outliers indicated by the right tail.

c) Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value *(aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):*

i) Performed hypothesis testing and visual analysis between aggregated actual_time and aggregated OSRM_time (grouped by trip_uuid) to check if there's a significant difference in trip durations estimated vs. actual.

ii)

```
actual_agg = df.groupby("trip_uuid")['actual_time'].sum()
actual_agg
```

trip_uuid	actual_time
trip-153671041653548748	15682.0
trip-153671042288605164	399.0
trip-153671043369099517	112225.0

iii)

```
osrm_time_agg = df.groupby("trip_uuid")['osrm_time'].sum()
osrm_time_agg
```

trip_uuid	osrm_time
trip-153671041653548748	7787.0
trip-153671042288605164	210.0
trip-153671043369099517	65768.0

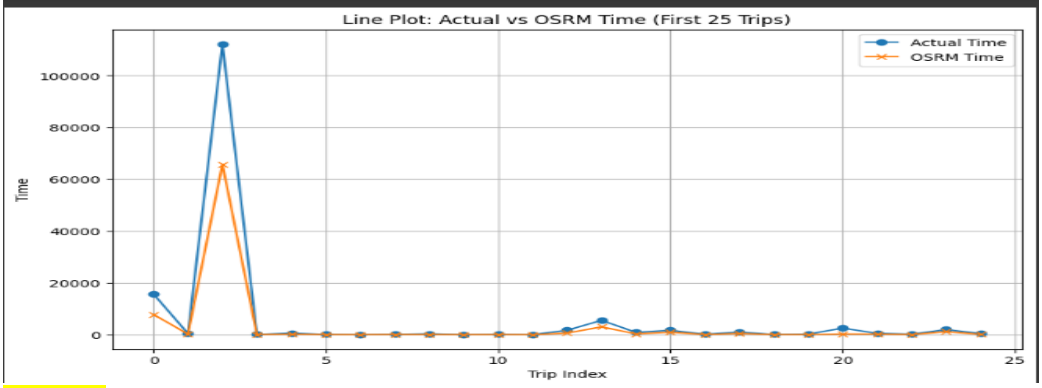
```
f_stats, P_val = ttest_rel(actual_agg.values, osrm_time_agg.values)
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: 651.1832057297116
p-value: 0.0
```

```
if p_value > 0.05:
    print("The distributions are not significantly different.")
else:
    print("The distributions are significantly different.")
```

```
The distributions are significantly different.
```

iv)



v)

vi) Insights - The line plot shows that for most trips, the actual time and OSRM estimated time are quite close, following a similar trend. However, a few trips—especially one with a very high actual time—show significant deviations, indicating possible delays or anomalies.

d) Do hypothesis testing/ visual analysis between <u>actual_time aggregated value</u> and <u>segment actual time aggregated value</u> *(aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):*

i) Performed hypothesis testing and visual analysis between aggregated actual_time and aggregated segment actual time (grouped by trip_uuid) to check if there's a significant difference in trip durations estimated vs. actual.

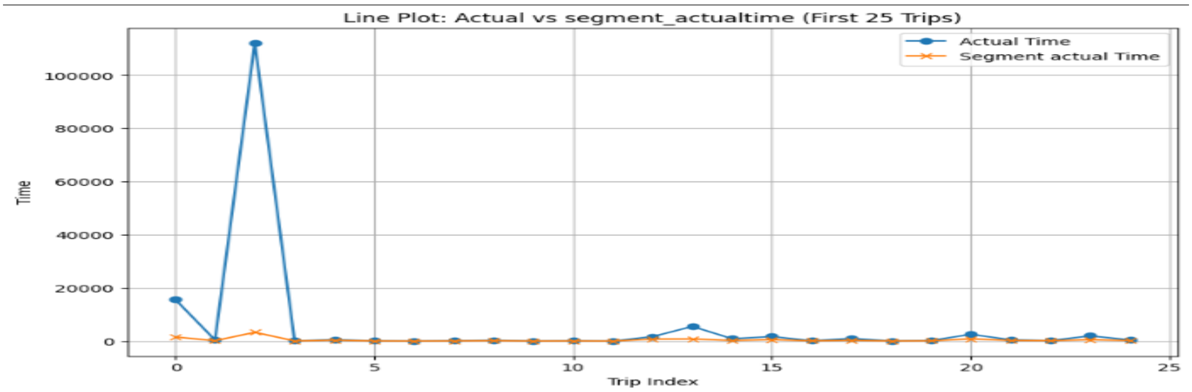
```
f_stats, P_val = ttest_rel(actual_agg.values, segment_actualtime_agg.values)
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: 651.1832057297116
p-value: 0.0
```

```
if p_value > 0.05:
    print("The distributions are not significantly different.")
else:
    print("The distributions are significantly different.")
```

ii)

```
The distributions are significantly different.
```

iii) 

iv) Insights - The plot shows that actual time is generally higher than segment actual time, with a few trips showing large gaps. This suggests possible delays or idle time between segments that aren't captured in segment-level data.

e) Do hypothesis testing/ visual analysis between <u>osrm distance aggregated value</u> and <u>segment osrm distance aggregated value</u> *(aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):*

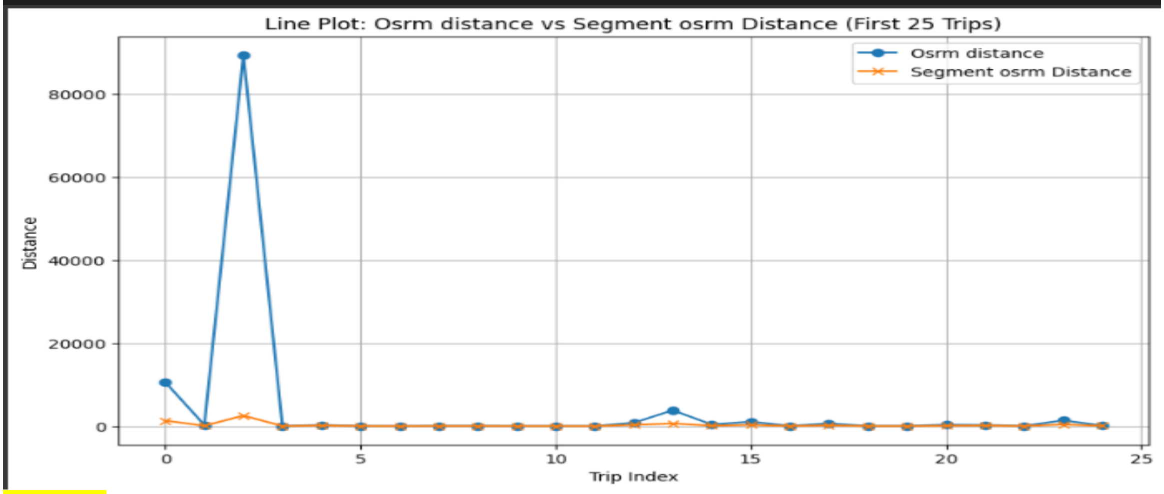
i) Performed hypothesis testing and visual analysis between aggregated <u>osrm distance</u> and aggregated <u>segment osrm distance</u> (grouped by trip_uuid) to check if there's a significant difference in trip durations estimated vs. actual.

ii)

```
f_stats, p_value = ttest_rel(osrm_dis_agg.values, seg_osrm_dis_agg.values)
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: 651.1832057297116
p-value: 2.1753879024067997e-192
```

```
if p_value > 0.05:
    print("The distributions are not significantly different.")
else:
    print("The distributions are significantly different.")
```

iii) 

iv) Insights – The plot shows that **OSRM distance is mostly similar to segment OSRM distance**, but a few trips have very large OSRM values, indicating possible data issues or route mismatches in those cases.

f) Do hypothesis testing/ visual analysis between <u>osrm time aggregated value</u> and <u>segment osrm time aggregated value</u> *(aggregated values are the values you'll get after merging the rows on the basis of trip_uuid):*

i) Performed hypothesis testing and visual analysis between aggregated <u>osrm time</u> and aggregated <u>segment osrm time</u> (grouped by trip_uuid) to check if there's a significant difference in trip durations estimated vs. actual.

ii)

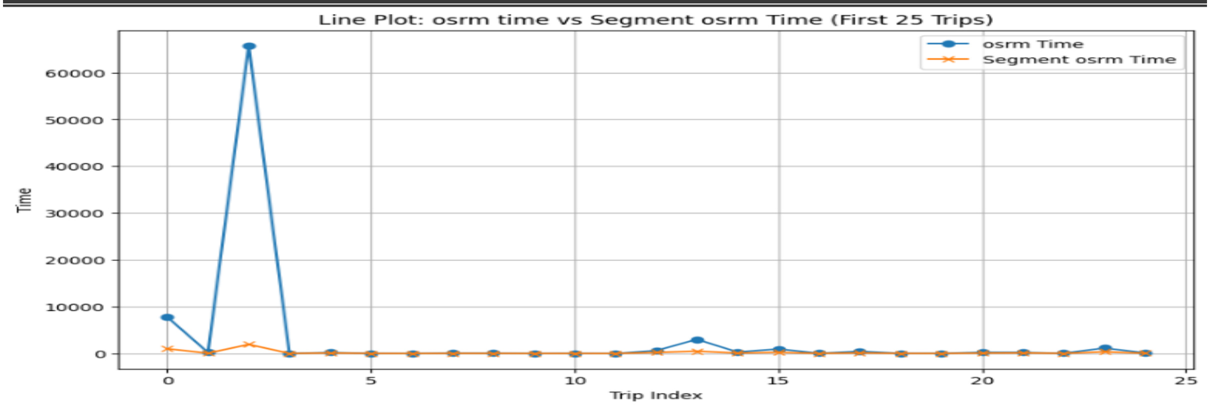
```
f_stats, p_value = ttest_rel(osrm_time_agg.values, segment_osrm_time_agg.values)
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: 651.1832057297116
p-value: 1.0892807362104113e-195
```

```
if p_value > 0.05:
    print("The distributions are not significantly different.")
else:
    print("The distributions are significantly different.")
```

```
The distributions are significantly different.
```

iii)



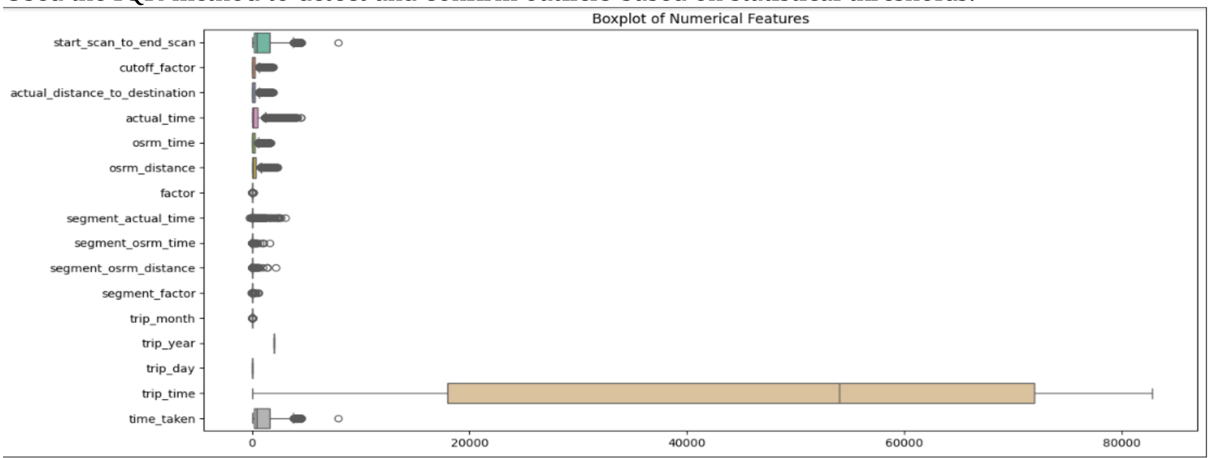
iv) Insights - The plot shows that **OSRM time and Segment OSRM time are mostly similar**, but one trip has a significantly higher OSRM time, indicating a possible outlier or routing error in that specific case.

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

i) Identified outliers in most numerical columns using visual tools like boxplots.

ii) Used the IQR method to detect and confirm outliers based on statistical thresholds.

iii)



iv)

```
numerical_cols = df.select_dtypes(include=[np.number]).columns

outlier_summary = {}

for col in numerical_cols:
    q1 = np.percentile(df[col], 25)
    q3 = np.percentile(df[col], 75)
    iqr = q3 - q1

    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    outlier_count = len(outliers)

    outlier_summary[col] = outlier_count

# Print outlier count per column
total_counts = 0
for col, count in outlier_summary.items():
    total_counts += count
    print(f"{col}: {count} outliers")

print(f'total_counts : {total_counts}')
```

v) Insights - Most numerical features, including actual_time, osrm_time, and time_taken, show the presence of **outliers**, as indicated by the dots beyond the whiskers. This confirms the earlier detection using the IQR method and highlights the need for proper handling of these extreme values during analysis.

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

i) Applied one-hot encoding to categorical variables like route_type and data to convert them into numerical format for model compatibility.

ii)

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
for col in df[['data', 'route_type']]:
  encoder = OneHotEncoder()
  encoded_data = encoder.fit_transform(df[[col]])
  encoded_df = pd.DataFrame(encoded_data.toarray(), columns=encoder.get_feature_names_out([col]))
  df = pd.concat([df, encoded_df], axis=1)
```

iii)

data_test	data_training	route_type_Carting	route_type_FTL
0.0	1.0	1.0	0.0
0.0	1.0	1.0	0.0
0.0	1.0	1.0	0.0
0.0	1.0	1.0	0.0

g) Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler:

i. Normalized/Standardized numerical features using MinMaxScaler to bring all values to a similar scale and improve model performance.

ii.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

num_cols = df.select_dtypes(include=['int64', 'float64']).columns
df_num = df[num_cols]

scaler = MinMaxScaler()
df_scaled_minmax = pd.DataFrame(scaler.fit_transform(df_num), columns=num_cols)
df = pd.concat([df, df_scaled_minmax], axis=1)
```

iii.

factor	segment_actual_time	segment_osrm_time	segment_osrm_distance	segment_factor	time_taken
0.014613	0.078300	0.006828	0.005460	0.041354	0.008316
0.013671	0.077086	0.005587	0.004453	0.041084	0.008316
0.016630	0.078907	0.004345	0.004935	0.043049	0.008316
0.018202	0.080425	0.007449	0.005942	0.042153	0.008316
0.018143	0.075873	0.003104	0.001787	0.041233	0.008316

Business Insights:

- Most trips are concentrated in a few corridors and specific states, especially high-volume states like Maharashtra and Delhi.
- Busiest corridors often involve short distances but high trip frequency, indicating efficient zones.
- Some trips have large gaps between segment time and total time, suggesting delays not captured in segment data.
- OSRM times and distances are mostly aligned with actuals, but deviations signal potential data quality issues or real-world obstacles.
- Scans and route data provide opportunities to optimize delivery patterns and reduce delays.

Recommendations:

- Focus on improving logistics in corridors where actual time consistently exceeds OSRM estimates.
- Investigate and address causes of delays between segments to improve overall delivery time.
- Use scan data more effectively to detect anomalies early in the trip.
- Improve route planning in cases where OSRM distance significantly exceeds segment values.
- Prioritize automation and tracking in high-frequency corridors for better operational efficiency.