delhivery

October 28, 2024

[7]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[3]:

```
import pandas as pd

# Load your dataset if it's not already loaded
data = pd.read_csv("C:/Users/shrad/Downloads/delhivery.csv")
```

[5]:

```
print(data.shape)
print(data.info())
print(data.describe())
print(data.isnull().sum())
```

```
(144867, 24)
<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  int64
 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  int64
 17  osrm_time                       144867 non-null  int64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  int64
 21  segment_osrm_time               144867 non-null  int64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(5), int64(6), object(12)
memory usage: 25.6+ MB
None
       start_scan_to_end_scan  cutoff_factor  actual_distance_to_destination  \
count           144867.000000  144867.000000                   144867.000000
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

         actual_time      osrm_time  osrm_distance         factor  \
count  144867.000000  144867.000000  144867.000000  144867.000000
mean      416.927527     213.868272     284.771297       2.120107
std       598.103621     308.011085     421.119294       1.715421
min         9.000000       6.000000       9.008200       0.144000
25%        51.000000      27.000000      29.914700       1.604264
50%       132.000000      64.000000      78.525800       1.857143
75%       513.000000     257.000000     343.193250       2.213483
max      4532.000000    1686.000000    2326.199100      77.387097

       segment_actual_time  segment_osrm_time  segment_osrm_distance  \
count        144867.000000      144867.000000           144867.00000
mean             36.196111          18.507548               22.82902
std              53.571158          14.775960               17.86066
min            -244.000000           0.000000                0.00000
25%              20.000000          11.000000               12.07010
50%              29.000000          17.000000               23.51300
75%              40.000000          22.000000               27.81325
max            3051.000000        1611.000000             2191.40370

       segment_factor
count   144867.000000
mean         2.218368
std          4.847530
min        -23.444444
25%          1.347826
50%          1.684211
```

```
75%          2.250000
max        574.250000
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
```

[11]:

```
# Handle missing values without using inplace
for col in data.select_dtypes(include=['float', 'int']).columns:
    data[col] = data[col].fillna(data[col].median())

for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].fillna(data[col].mode()[0])
```

[21]:

```
# Split 'destination_name' with a limit of 2 splits
destination_split = data['destination_name'].str.split('-', expand=True, n=2)

# Rename columns if there are fewer than 3 parts
destination_split.columns = [f'destination_part_{i+1}' for i in␣
 ↪range(destination_split.shape[1])]
# Ensure exactly 3 columns by reindexing with NaN for missing columns
destination_split = destination_split.reindex(columns=['destination_part_1',␣
 ↪'destination_part_2', 'destination_part_3'])

# Rename to meaningful column names
```

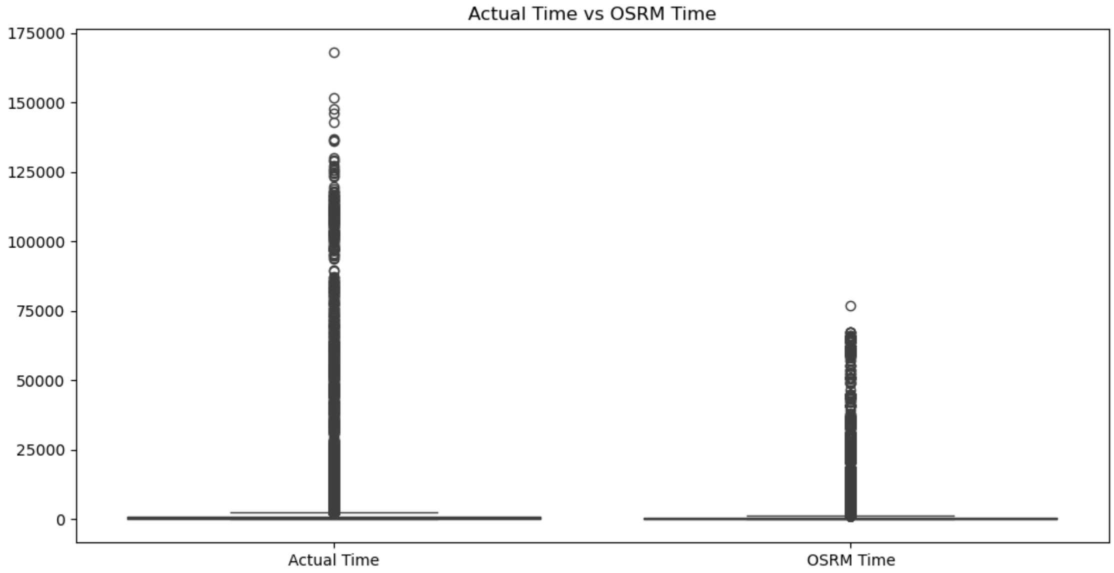
```
destination_split.columns = ['destination_city', 'destination_place',␣
  ↪'destination_code']

# Add the split columns back to the main DataFrame
data = pd.concat([data, destination_split], axis=1)

# Repeat for 'source_name'
source_split = data['source_name'].str.split('-', expand=True, n=2)
source_split.columns = [f'source_part_{i+1}' for i in range(source_split.
  ↪shape[1])]
source_split = source_split.reindex(columns=['source_part_1', 'source_part_2',␣
  ↪'source_part_3'])
source_split.columns = ['source_city', 'source_place', 'source_code']

data = pd.concat([data, source_split], axis=1)
```

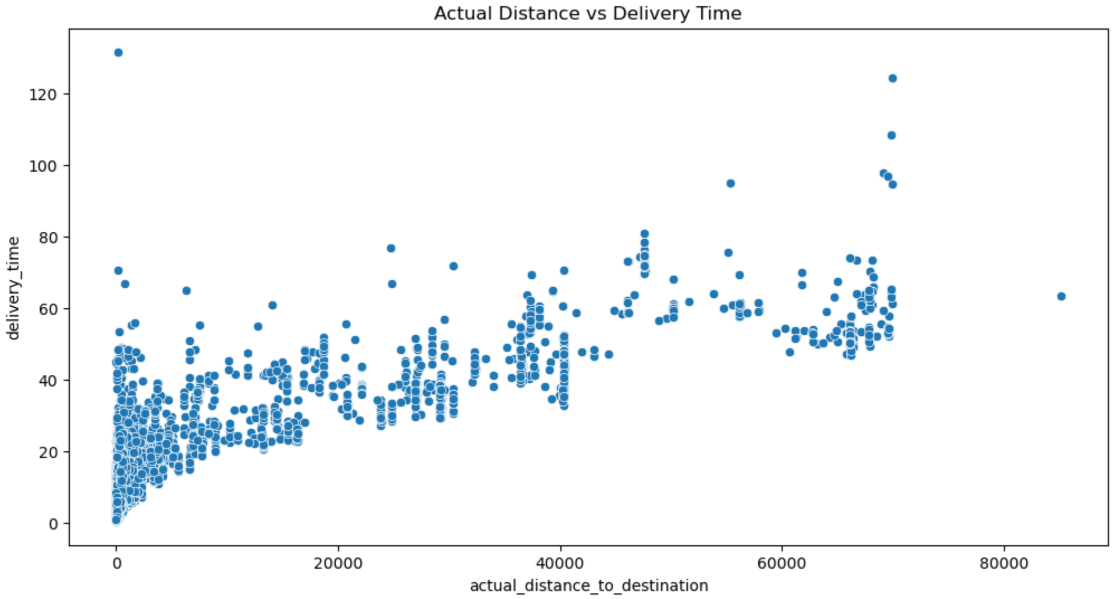
[23]:

```
# Aggregating based on 'trip_uuid'
agg_data = data.groupby('trip_uuid').agg({
    'actual_time': 'sum',  # cumulative
    'osrm_time': 'sum',  # cumulative
    'actual_distance_to_destination': 'sum',
    'start_scan_to_end_scan': 'first',  # or 'last', if appropriate
    'od_start_time': 'first',
    'od_end_time': 'last'
}).reset_index()

# Calculate delivery time and drop original columns if required
agg_data['delivery_time'] = (pd.to_datetime(agg_data['od_end_time']) - pd.
  ↪to_datetime(agg_data['od_start_time'])).dt.total_seconds() / 3600
agg_data.drop(['od_start_time', 'od_end_time'], axis=1, inplace=True)
```

[25]:

```
# Compare aggregated times
plt.figure(figsize=(12, 6))
sns.boxplot(data=[agg_data['actual_time'], agg_data['osrm_time']])
plt.xticks([0, 1], ['Actual Time', 'OSRM Time'])
plt.title("Actual Time vs OSRM Time")
plt.show()

# Compare distances
plt.figure(figsize=(12, 6))
sns.scatterplot(x='actual_distance_to_destination', y='delivery_time',␣
  ↪data=agg_data)
plt.title("Actual Distance vs Delivery Time")
plt.show()
```





[27]:

```
# Detecting outliers using the IQR method
Q1 = agg_data['actual_time'].quantile(0.25)
Q3 = agg_data['actual_time'].quantile(0.75)
IQR = Q3 - Q1

# Filtering out outliers
agg_data = agg_data[(agg_data['actual_time'] >= (Q1 - 1.5 * IQR)) &␣
 ↪(agg_data['actual_time'] <= (Q3 + 1.5 * IQR))]
```

[31]:

```
# Check if 'route_type' column exists in the DataFrame
if 'route_type' in agg_data.columns:
    # Perform one-hot encoding
    agg_data = pd.get_dummies(agg_data, columns=['route_type'], drop_first=True)
else:
    print("Column 'route_type' not found in agg_data.")
```

```
Column 'route_type' not found in agg_data.
```

[35]:

```
# Checking if 'route_type' is in the original data
if 'route_type' in data.columns:
    # Re-create agg_data to include 'route_type' if it was excluded
    agg_data = data.groupby(['trip_uuid', 'source_center',␣
 ↪'destination_center', 'route_type']).agg({
        'actual_time': 'sum',
        'osrm_time': 'sum',
        # add other columns and aggregation functions as required
    }).reset_index()

    # One-hot encoding for route_type if it exists now
    agg_data = pd.get_dummies(agg_data, columns=['route_type'], drop_first=True)
else:
    print("Column 'route_type' not found in original data either.")
```

[41]:

```
# One-hot encoding for categorical variables like route_type
agg_data = pd.get_dummies(agg_data, columns=['route_type_FTL'], drop_first=True)
```

[39]:

```
print(agg_data)
```

```
                     trip_uuid source_center destination_center  actual_time  \
0      trip-153671041653548748  IND209304AAA       IND000000ACB         6484
1      trip-153671041653548748  IND462022AAA       IND209304AAA         9198
2      trip-153671042288605164  IND561203AAB       IND562101AAA           96
3      trip-153671042288605164  IND572101AAA       IND561203AAB          303
4      trip-153671043369099517  IND000000ACB       IND160002AAC         2601
…                          …           …                  …          …
26363  trip-153861115439069069  IND628204AAA       IND627657AAA          119
26364  trip-153861115439069069  IND628613AAA       IND627005AAA          173
26365  trip-153861115439069069  IND628801AAA       IND628204AAA           51
26366  trip-153861118270144424  IND583119AAA       IND583101AAA          278
26367  trip-153861118270144424  IND583201AAA       IND583119AAA           72

       osrm_time  route_type_FTL
0           3464            True
1           4323            True
2             55           False
3            155           False
```

```
4           1427            True
…           …            …
26363        106           False
26364        108           False
26365         22           False
26366         59            True
26367         47            True

[26368 rows x 6 columns]
```

[44]:

```
# Check the columns in agg_data
print(agg_data.columns)
```

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

[52]:

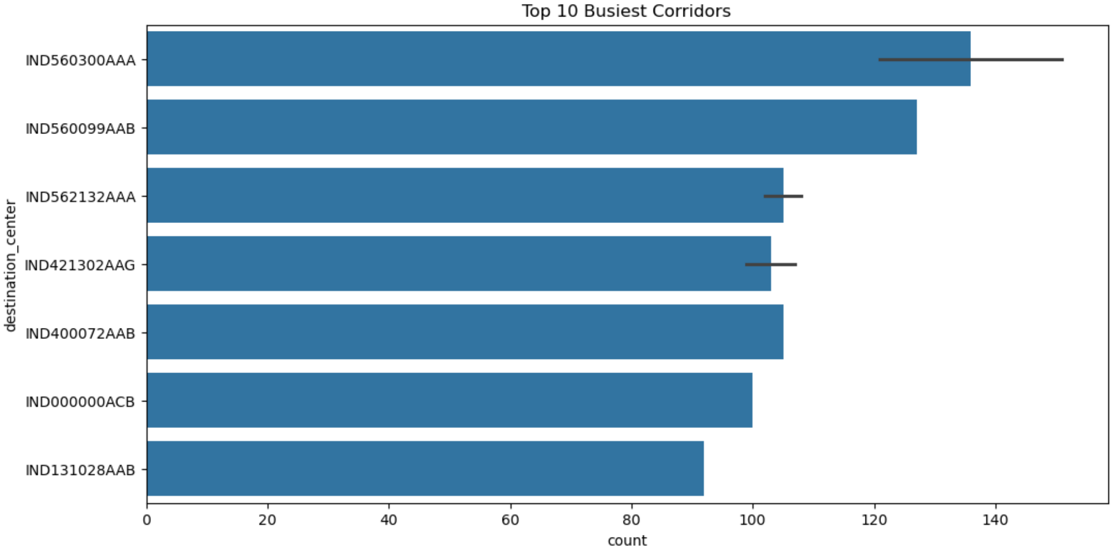
```
# Example of including delivery_time and actual_distance_to_destination in the␣
 ↪aggregation
agg_data = data.groupby(['trip_uuid', 'source_center', 'destination_center',␣
 ↪'route_type']).agg({
    'actual_time': 'sum',
    'osrm_time': 'sum',  # Ensure this column is being calculated
    'actual_distance_to_destination': 'sum',  # Ensure this column is being␣
 ↪calculated
    # Add other aggregations as needed
}).reset_index()
```

[54]:

```
scaler = StandardScaler()
agg_data[['actual_time', 'osrm_time', 'actual_distance_to_destination']] =␣
 ↪scaler.fit_transform(agg_data[['actual_time', 'osrm_time',␣
 ↪'actual_distance_to_destination']])
```

[60]:

```
#Business Insights and Recommendation
# High traffic corridors and average distance
top_corridors = agg_data.groupby(['source_center', 'destination_center']).
 ↪size().reset_index(name='count').sort_values(by='count', ascending=False)
avg_distance = agg_data['actual_distance_to_destination'].mean()

# Plot busiest corridors
plt.figure(figsize=(12, 6))
sns.barplot(x='count', y='destination_center', data=top_corridors.head(10))
plt.title("Top 10 Busiest Corridors")
plt.show()

print("Average Distance:", avg_distance)
```



```
Average Distance: 1.0778864316749092e-18
```

```
[ ]:
```