Data Analytics and Visualization by Piyush Joshi

COLAB LINK:https://colab.research.google.com/drive1djaiZer5uvb_RsZ-d19a9m-GQNWZ8Ek9?usp=sharing

Delhivery, India's leading and rapidly growing integrated player, has set its sights on creating the commerce operating system.The company wants to understand and process the data coming out of data engineering pipelines is clean, sanitize and manipulate data to get useful features out of raw fields to make sense out of the raw data and help the data science team to build forecasting models on it.

Column Profiling:

1. data - tells whether the data is testing or training data
2. trip_creation_time - Timestamp of trip creation
3. route_schedule_uuid - Unique ID for a particular route schedule
4. route_type- Transportation type a. FTL - Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way b. Carting: Handling system consisting of small vehicles (carts)
5. trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
6. source_center - Source ID of trip origin
7. source_name - Source Name of trip origin
8. destination_cente - Destination ID
9. destination_name - Destination Name
10. od_start_time- Trip start time
11. od_end_time - Trip end time
12. start_scan_to_end_scan- Time taken to deliver from source to destination
13. is_cutoff - Unknown field
14. cutoff_factor- Unknown field
15. cutoff_timestamp - Unknown field
16. actual_distance_to_destination- Distance in kms between source and destination warehouse
17. actual_time- Actual time taken to complete the delivery (Cumulative)
18. osrm_time- An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
19. osrm_distance- An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
20. factor- Unknown field
21. segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
22. segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
23. segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
24. segment_factor – Unknown field

1. Basic data cleaning and exploration:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
```

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181
```

```
--2024-10-13 06:34:18--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 3.163.19.158, 3.163.19.165, 3.163.19.93, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|3.163.19.158|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 55617130 (53M) [text/plain]
Saving to: ‘delhivery_data.csv?1642751181’

delhivery_data.csv? 100%[===================>]  53.04M  88.7MB/s    in 0.6s
```

```
2024-10-13 06:34:19 (88.7 MB/s) - ‘delhivery_data.csv?1642751181’ saved [55617130/55617130]
```

```
df=pd.read_csv('delhivery_data.CSV',dayfirst=True)
df.head(2)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_c
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 rows × 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
```

```
#Analyzing Null Values in Percentage
df.isnull().sum()/len(df)*100
```

	0
data	0.000000
trip_creation_time	0.000000
route_schedule_uuid	0.000000
route_type	0.000000
trip_uuid	0.000000
source_center	0.000000
source_name	0.202254
destination_center	0.000000
destination_name	0.180165
od_start_time	0.000000
od_end_time	0.000000
start_scan_to_end_scan	0.000000
is_cutoff	0.000000
cutoff_factor	0.000000
cutoff_timestamp	0.000000
actual_distance_to_destination	0.000000
actual_time	0.000000
osrm_time	0.000000
osrm_distance	0.000000
factor	0.000000
segment_actual_time	0.000000
segment_osrm_time	0.000000
segment_osrm_distance	0.000000
segment_factor	0.000000

dtype: float64

```
df.dropna(inplace=True)
df.isnull().sum()
```

	0
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

```
# Selecting the columns that we want to convert to datetime
columns_to_convert = ['trip_creation_time', 'od_start_time', 'od_end_time', 'cutoff_timestamp']

# Using apply `pd.to_datetime`to columns_to_convert
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_datetime, dayfirst=True, format='mixed', errors='coerce')
```

```
df[columns_to_convert].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
Data columns (total 4 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   trip_creation_time  144316 non-null  datetime64[ns]
 1   od_start_time       144316 non-null  datetime64[ns]
 2   od_end_time         144316 non-null  datetime64[ns]
 3   cutoff_timestamp    144316 non-null  datetime64[ns]
dtypes: datetime64[ns](4)
memory usage: 5.5 MB
```

```
#Checking to see duplicates
df.duplicated().any()
```

```
False
```

```
#3. Analyze structure & characteristics of the dataset.
print("No. of rows:{} ".format(df.shape[0]))
print("No. of columns:{} ".format(df.shape[1]))
```

```
No. of rows:144316
No. of columns:24
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 144316 entries, 0 to 144866
```

```
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144316 non-null  object
 1   trip_creation_time              144316 non-null  datetime64[ns]
 2   route_schedule_uuid             144316 non-null  object
 3   route_type                      144316 non-null  object
 4   trip_uuid                       144316 non-null  object
 5   source_center                   144316 non-null  object
 6   source_name                     144316 non-null  object
 7   destination_center              144316 non-null  object
 8   destination_name                144316 non-null  object
 9   od_start_time                   144316 non-null  datetime64[ns]
 10  od_end_time                     144316 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144316 non-null  float64
 12  is_cutoff                       144316 non-null  bool
 13  cutoff_factor                   144316 non-null  int64
 14  cutoff_timestamp                144316 non-null  datetime64[ns]
 15  actual_distance_to_destination  144316 non-null  float64
 16  actual_time                     144316 non-null  float64
 17  osrm_time                       144316 non-null  float64
 18  osrm_distance                   144316 non-null  float64
 19  factor                          144316 non-null  float64
 20  segment_actual_time             144316 non-null  float64
 21  segment_osrm_time               144316 non-null  float64
 22  segment_osrm_distance           144316 non-null  float64
 23  segment_factor                  144316 non-null  float64
dtypes: bool(1), datetime64[ns](4), float64(10), int64(1), object(8)
memory usage: 26.6+ MB
```

```
for col in df.columns:
    unique_values=df[col].unique()
    print(f"Unique values in Column-> {col}:")
    print(unique_values)
    print("-"*70)
```

Insights: After removal of null values the No. of rows are144316 and No. of columns are 24.

Recommendation: Go ahead with confidence for exploratory data analysis techniques.

› 2. Merging the rows

[ ] ↳ *10 cells hidden*

› 3. Feature Engineering:

[ ] ↳ *4 cells hidden*

› 4. In-depth analysis:

[ ] ↳ *14 cells hidden*

⌄ 5. Hypothesis Testing:

1. Perform hypothesis testing / visual analysis between :

a. Actual_time aggregated value and OSRM time aggregated value for Normalized data

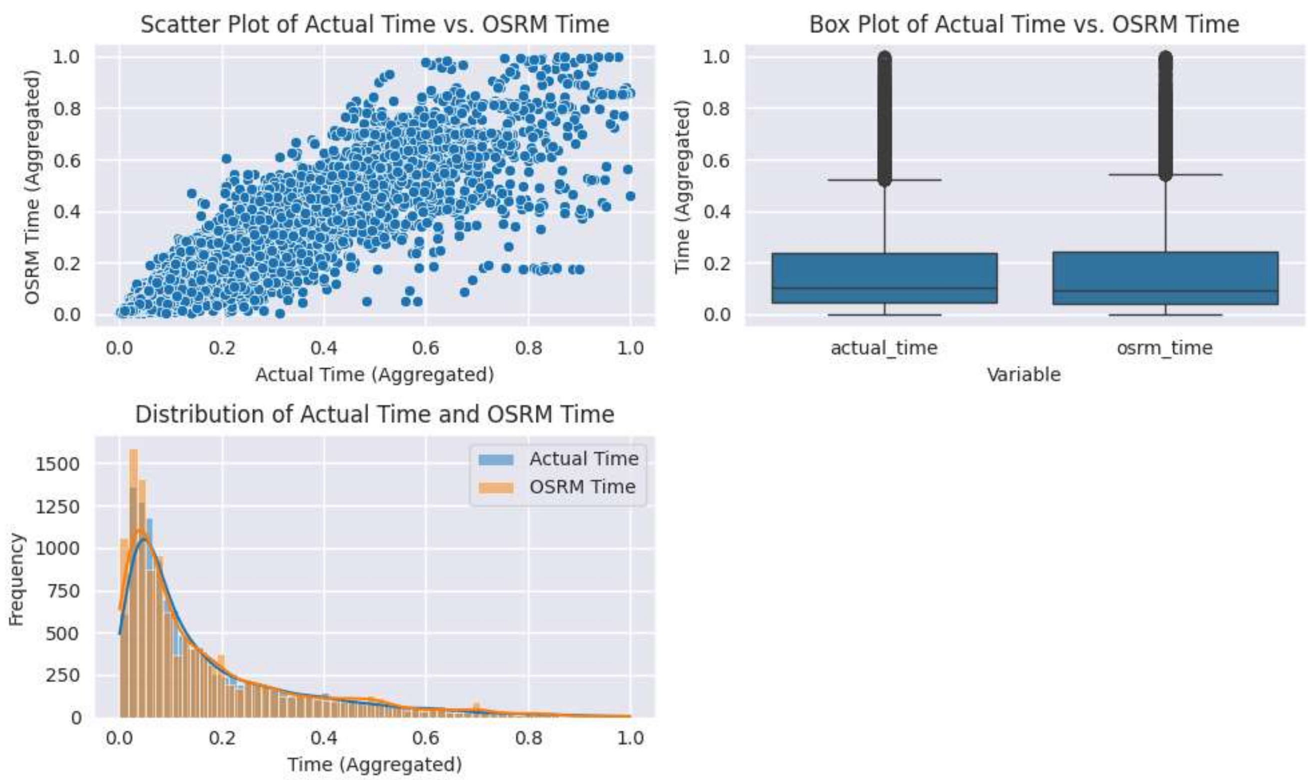
Insights:

1. Huge no. of outliers in the continuous variables despite removal of outliers outside the 5 to 95% IQR.
2. Boxplot comparision of actual time, OSRM time and segment OSRM time reveals the distribution of segment OSRM time on lower ranges followed by OSRM time.
3. Boxplot comparision of OSRM distance and segment OSRM distance reveals the distribution of OSRM distance calculations on lower ranges.

Recommendation: Leverage OSRM for shortest time and distance analysis but ensure its time and distance estimates remain realistic when compared with actual times. While lower OSRM times/distances are expected, if the difference is too large, further refinement or validation might be necessary to reflect real-world driving conditions more accurately.

```
# Create a 2x2 subplot for the visual analysis of NORMALIZED feat
```

```
# Creatng a 2x2 subplot for the visual analysis of NORMALIZED features
fig, axs = plt.subplots(2, 2, figsize=(10, 6))  # Adjusted figure size

# Scatter plot
sns.scatterplot(x='actual_time', y='osrm_time', data=data, ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of Actual Time vs. OSRM Time')
axs[0, 0].set_xlabel('Actual Time (Aggregated)')
axs[0, 0].set_ylabel('OSRM Time (Aggregated)')

# Box plot
melted_data = pd.melt(data, id_vars=[], value_vars=['actual_time', 'osrm_time'], var_name='Variable', value_name='Time')
sns.boxplot(x='Variable', y='Time', data=melted_data, ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of Actual Time vs. OSRM Time')
axs[0, 1].set_xlabel('Variable')
axs[0, 1].set_ylabel('Time (Aggregated)')

# Histogram
sns.histplot(data['actual_time'], label='Actual Time', kde=True, ax=axs[1, 0])
sns.histplot(data['osrm_time'], label='OSRM Time', kde=True, ax=axs[1, 0])
axs[1, 0].set_title('Distribution of Actual Time and OSRM Time')
axs[1, 0].set_xlabel('Time (Aggregated)')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].legend()

# Remove the empty fourth subplot
fig.delaxes(axs[1, 1])

# Adjust layout
plt.tight_layout()
```




```
# Shapiro-Wilk test
#H0: The data is normally distributed.
#HA: The data is not normally distributed.
import scipy.stats as stats
statistic, p_value = stats.shapiro(data['actual_time'])
print('Shapiro-Wilk Test for ACTUAL_TIME:')
print('Statistic:', statistic)
print('P-value:', p_value)
if p_value > 0.05:
  print('The data is normally distributed (fail to reject H0)')
else:
  print('The data is not normally distributed (reject H0)')

#H0: The data is normally distributed.
#HA: The data is not normally distributed.
statistic, p_value = stats.shapiro(data['osrm_time'])
print('Shapiro-Wilk Test for OSRM_TIME:')
print('Statistic:', statistic)
print('P-value:', p_value)
if p_value > 0.05:
```

```
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Shapiro-Wilk Test for ACTUAL_TIME:
Statistic: 0.7922480272371552
P-value: 4.142162717305357e-82
The data is not normally distributed (reject H0)
Shapiro-Wilk Test for OSRM_TIME:
Statistic: 0.790642236176533
P-value: 2.781243186209966e-82
The data is not normally distributed (reject H0)
```

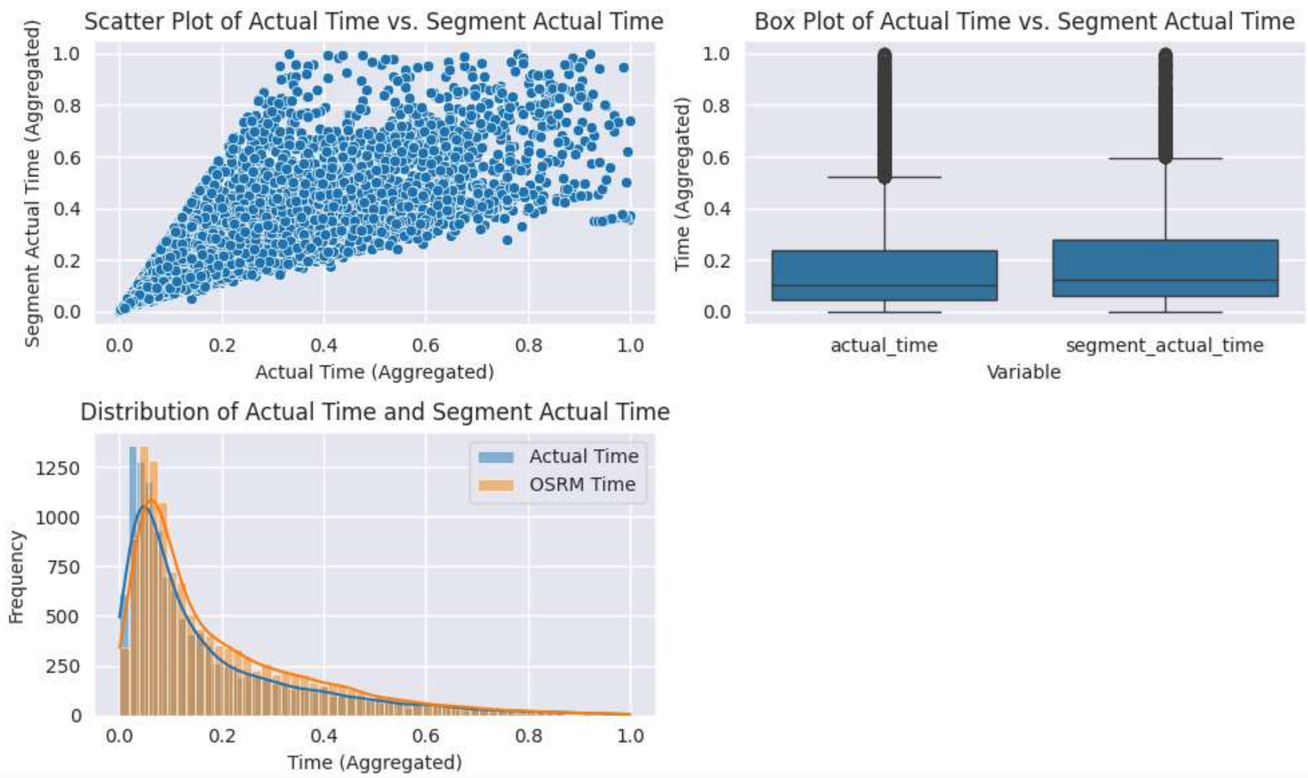
Since the t-test assumes normality, you should consider using a non-parametric alternative, which doesn't rely on this assumption. In this case, the Wilcoxon signed-rank test would be appropriate, as we're comparing two paired samples.

```
from scipy import stats
#H0:There is no significant difference between actual time and OSRM time
#HA:There is a significant difference between actual time and OSRM time
# Perform a paired t-test
statistic, p_value = stats.wilcoxon(data['actual_time'], data['osrm_time'])

# Print the results
print(f"statistic: {statistic}")
print(f"P-value: {p_value}")

# Interpret the results
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference between actual time and OSRM time.")
else:
    print("Fail to reject the null hypothesis. There is no significant difference between actual time and OSRM time.")
```

```
statistic: 36840328.5
P-value: 0.027076194525105318
Reject the null hypothesis. There is a significant difference between actual time and OSRM time.
```

b. actual_time aggregated value and segment actual time aggregated value.

```
#5b. actual_time aggregated value and segment actual time aggregated value.

# Creatng a 2x2 subplot for the visual analysis of NORMALIZED features
fig, axs = plt.subplots(2, 2, figsize=(10, 6))  # Adjusted figure size

# Scatter plot
sns.scatterplot(x='actual_time', y='segment_actual_time', data=data, ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of Actual Time vs. Segment Actual Time')
axs[0, 0].set_xlabel('Actual Time (Aggregated)')
axs[0, 0].set_ylabel('Segment Actual Time (Aggregated)')

# Box plot
melted_data = pd.melt(data, id_vars=[], value_vars=['actual_time', 'segment_actual_time'], var_name='Variable', value_name='Time')
sns.boxplot(x='Variable', y='Time', data=melted_data, ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of Actual Time vs. Segment Actual Time')
axs[0, 1].set_xlabel('Variable')
axs[0, 1].set_ylabel('Time (Aggregated)')

# Histogram
sns.histplot(data['actual_time'], label='Actual Time', kde=True, ax=axs[1, 0])
sns.histplot(data['segment_actual_time'], label='OSRM Time', kde=True, ax=axs[1, 0])
axs[1, 0].set_title('Distribution of Actual Time and Segment Actual Time')
axs[1, 0].set_xlabel('Time (Aggregated)')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].legend()

# Remove the empty fourth subplot
fig.delaxes(axs[1, 1])

# Adjust layout
plt.tight_layout()
```



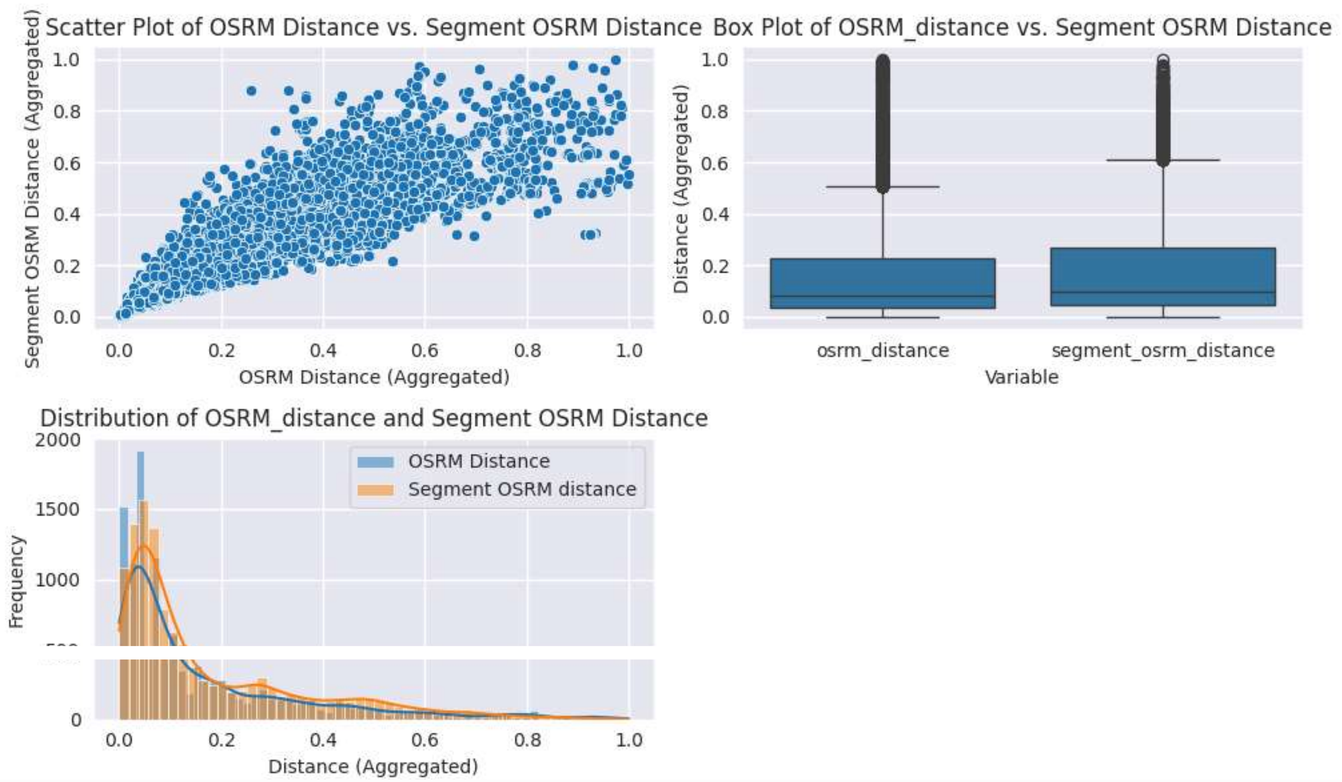

```
from scipy import stats
#H0:There is no significant difference between actual time and Segmented actual time
#HA:There is a significant difference between actual time and Segmented actual time
# Perform a paired t-test
statistic, p_value = stats.wilcoxon(data['actual_time'], data['segment_actual_time'])
# Print the results
print(f"statistic: {statistic}")
print(f"P-value: {p_value}")

# Interpret the results
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference between actual time and Segmented actual time.")
else:
    print("Fail to reject the null hypothesis. There is no significant difference between actual time and Segmented actual time.")
```

```
statistic: 21742758.0
P-value: 0.0
Reject the null hypothesis. There is a significant difference between actual time and Segmented actual time.
```

c. OSRM distance aggregated value and segment OSRM distance aggregated value.

```
#5c. OSRM distance aggregated value and segment OSRM distance aggregated value.


# Creatng a 2x2 subplot for the visual analysis of NORMALIZED features
fig, axs = plt.subplots(2, 2, figsize=(10, 6))  # Adjusted figure size

# Scatter plot
sns.scatterplot(x='osrm_distance', y='segment_osrm_distance', data=data, ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of OSRM Distance vs. Segment OSRM Distance')
axs[0, 0].set_xlabel('OSRM Distance (Aggregated)')
axs[0, 0].set_ylabel('Segment OSRM Distance (Aggregated)')

# Box plot
melted_data = pd.melt(data, id_vars=[], value_vars=['osrm_distance', 'segment_osrm_distance'], var_name='Variable', value_name='Distance
sns.boxplot(x='Variable', y='Distance', data=melted_data, ax=axs[0, 1]) # Now using 'Distance' for y-axis
axs[0, 1].set_title('Box Plot of OSRM_distance vs. Segment OSRM Distance')
axs[0, 1].set_xlabel('Variable')
axs[0, 1].set_ylabel('Distance (Aggregated)')

# Histogram
sns.histplot(data['osrm_distance'], label='OSRM Distance', kde=True, ax=axs[1, 0])
sns.histplot(data['segment_osrm_distance'], label='Segment OSRM distance', kde=True, ax=axs[1, 0])
axs[1, 0].set_title('Distribution of OSRM_distance and Segment OSRM Distance')
axs[1, 0].set_xlabel('Distance (Aggregated)')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].legend()
```

```
# Removing the empty fourth subplot
fig.delaxes(axs[1, 1])

# Adjust layout
plt.tight_layout()
```



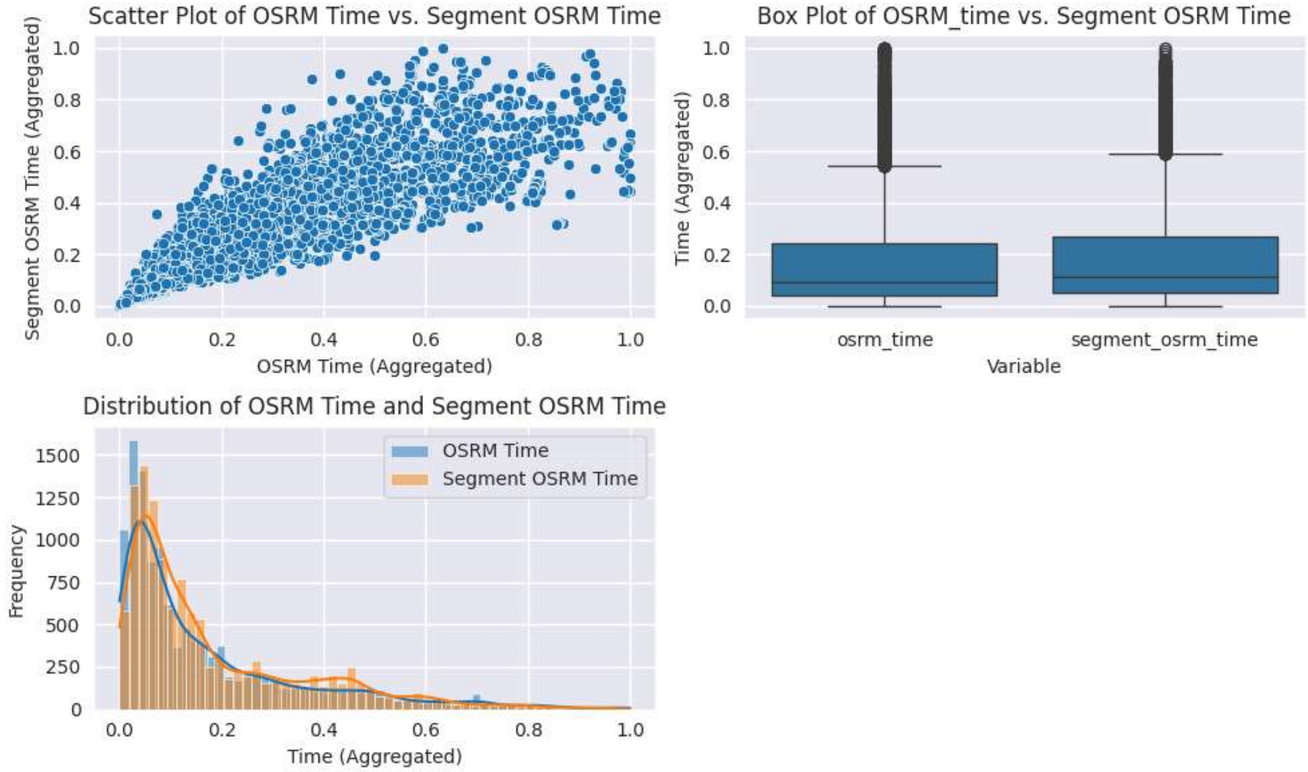

```
#Null Hypothesis (H0): There is no significant difference between the OSRM distance aggregated value and the segment OSRM distance aggre
#Alternative Hypothesis (HA): There is a significant difference between the OSRM distance aggregated value and the segment OSRM distance

statistic, p_value = stats.wilcoxon(data['osrm_distance'], data['segment_osrm_distance'])

# Print the results
print(f"statistic: {statistic}")
print(f"P-value: {p_value}")

alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference between OSRM distance and segment OSRM distance.")
else:
    print("Fail to reject the null hypothesis. There is no significant difference between OSRM distance and segment OSRM distance.")
```

```
statistic: 24607831.0
P-value: 1.310209278070086e-243
Reject the null hypothesis. There is a significant difference between OSRM distance and segment OSRM distance.
```

d. OSRM time aggregated value and segment OSRM time aggregated value.

```
#5d. OSRM time aggregated value and segment OSRM time aggregated value.

# Creatng a 2x2 subplot for the visual analysis of NORMALIZED features
fig, axs = plt.subplots(2, 2, figsize=(10, 6))


# Scatter plot
sns.scatterplot(x='osrm_time', y='segment_osrm_time', data=data, ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of OSRM Time vs. Segment OSRM Time')
axs[0, 0].set_xlabel('OSRM Time (Aggregated)')
axs[0, 0].set_ylabel('Segment OSRM Time (Aggregated)')

# Box plot
melted_data = pd.melt(data, id_vars=[], value_vars=['osrm_time', 'segment_osrm_time'], var_name='Variable', value_name='Time')
sns.boxplot(x='Variable', y='Time', data=melted_data, ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of OSRM_time vs. Segment OSRM Time')
axs[0, 1].set_xlabel('Variable')
axs[0, 1].set_ylabel('Time (Aggregated)')
```

```
# Histogram
sns.histplot(data['osrm_time'], label='OSRM Time', kde=True, ax=axs[1, 0])
sns.histplot(data['segment_osrm_time'], label='Segment OSRM Time', kde=True, ax=axs[1, 0])
axs[1, 0].set_title('Distribution of OSRM Time and Segment OSRM Time')
axs[1, 0].set_xlabel('Time (Aggregated)')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].legend()

# Removing the empty fourth subplot
fig.delaxes(axs[1, 1])

# Adjust layout
plt.tight_layout()
```




```
#Null Hypothesis (H0): There is no significant difference between the OSRM time aggregated value and the segment OSRM time aggregated val
#Alternative Hypothesis (HA): There is a significant difference between the OSRM time aggregated value and the segment OSRM time aggregat

statistic, p_value = stats.wilcoxon(data['osrm_time'], data['segment_osrm_time'])

# Print the results
print(f"statistic: {statistic}")
print(f"P-value: {p_value}")

alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference between OSRM time and segment OSRM time.")
else:
    print("Fail to reject the null hypothesis. There is no significant difference between OSRM time and segment OSRM time.")
```

```
statistic: 25297855.0
P-value: 7.890175526879054e-219
Reject the null hypothesis. There is a significant difference between OSRM time and segment OSRM time.
```

Insights:

1. OSRM consistently underestimates both time and distance compared to actual values, even after normalization.
2. All comparisons revealed statistically significant differences between the compared variables.

Recommendations:

1. Leverage segment-level data for analysis and route optimization.
2. Weigh statistical significance against practical relevance in decision-making.

6. Business Insights & Recommendations

Top 5 Origin and Desitnation Cities and States

```
# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(11, 6))

# Top 5 Source States
state_counts = df_trip_level['Source_State'].value_counts().reset_index().head(5)
state_counts.columns = ['Source_State', 'Order_Count']
sns.barplot(x='Source_State', y='Order_Count', data=state_counts, palette='viridis', ax=axs[0, 0])
axs[0, 0].set_title('Order Origins by State(Top 5)')
axs[0, 0].set_xlabel('State')
axs[0, 0].set_ylabel('Order Count')

# Top 5 Source Cities
city_counts = df_trip_level['Source_City'].value_counts().reset_index().head(5)
city_counts.columns = ['Source_City', 'Order_Count']
sns.barplot(x='Source_City', y='Order_Count', data=city_counts, palette='viridis', ax=axs[0, 1])
axs[0, 1].set_title('Order Origins by Source City (Top 5)')
axs[0, 1].set_xlabel('Source City')
axs[0, 1].set_ylabel('Order Count')

# Top 5 Destination States
state_counts = df_trip_level['Destination_State'].value_counts().reset_index().head(5)
state_counts.columns = ['Destination_State', 'Order_Count']
sns.barplot(x='Destination_State', y='Order_Count', data=state_counts, palette='viridis', ax=axs[1, 0])
axs[1, 0].set_title('Order by Destination State(Top 5)')
axs[1, 0].set_xlabel('State')
axs[1, 0].set_ylabel('Order Count')

# Top 5 Destination Cities
city_counts = df_trip_level['Destination_City'].value_counts().reset_index().head(5)
city_counts.columns = ['Destination_City', 'Order_Count']
sns.barplot(x='Destination_City', y='Order_Count', data=city_counts, palette='viridis', ax=axs[1, 1])
axs[1, 1].set_title('Order by Destination City (Top 5)')
axs[1, 1].set_xlabel('Destination City')
axs[1, 1].set_ylabel('Order Count')

# Adjust layout to add space between rows
plt.tight_layout(pad=4.0)
plt.show()
```

Busiest corridor

```
# Group by trip and sort by od_end_time
trip_segments = data.groupby('trip_uuid').apply(lambda x: x.sort_values(by='od_end_time')[['Source_City', 'Source_State', 'Destination_(

# Function to extract intermediate points
def get_intermediate_points(segments):
```

```
    source_points = segments[['Source_City', 'Source_State']].apply(lambda row: f"{row['Source_City']} ({row['Source_State']})", axis=1)
    # Get the last row using .iloc[-1] and access values directly
    dest_point = f"{segments['Destination_City'].iloc[-1]} ({segments['Destination_State'].iloc[-1]})"
    intermediate_points = source_points[1:] + [dest_point]
    return intermediate_points

# Apply the function to each group of the DataFrame
trip_segments['intermediate_points'] = trip_segments.groupby('trip_uuid').apply(lambda group: get_intermediate_points(group.reset_index(

# Create routes using city/state names
trip_segments['Most_frequent_route'] = trip_segments.apply(lambda x: f"{x['Source_City']} ({x['Source_State']}) -> {x['Destination_City

# Find the most frequent corridors
most_frequent_corridors = trip_segments['Most_frequent_route'].value_counts().head(5)
most_frequent_corridors
```

Most_frequent_route	count
Bengaluru (Karnataka) -> Bengaluru (Karnataka)	528
Bangalore (Karnataka) -> Bengaluru (Karnataka)	458
Bengaluru (Karnataka) -> Bangalore (Karnataka)	335
Bhiwandi (Maharashtra) -> Mumbai (Maharashtra)	331
Hyderabad (Telangana) -> Hyderabad (Telangana)	306

dtype: int64

Avg distance between the Busiest corridor

```
print(np.mean(data[(data["Source_City"]=="Bengaluru") & (data["Destination_City"]=="Bengaluru")]["actual_distance_to_destination"]),"kms
```

```
30.234301108402963 kms
```

Avg time taken

```
print(np.mean(data[(data["Source_City"]=="Bengaluru") & (data["Destination_City"]=="Bengaluru")]["actual_time"]), "hrs")
```

```
0.0908354942563293 hrs
```

Business Insights:

1. **OSRM Underestimation:** All comparisons between actual and OSRM values, as well as between aggregated and segment-level values, revealed statistically significant differences. This emphasizes the need to carefully consider these discrepancies in operational planning.
2. **Bengaluru Dominance**: Bengaluru is the most frequent source and destination for deliveries, indicating a high concentration of operations and customer base in this city.
3. **Maharashtra and Karnataka Significance**: Maharashtra and Karnataka are the top source and destination states, respectively, suggesting significant delivery activity in these regions.
4. **Route Optimization Potential**: The analysis of actual versus estimated times and distances suggests opportunities for route optimization and improved delivery efficiency.
5. **Salient Features**The data is given from the period '2018-09-12 00:00:16' to '2018-10-08 03:00:24'. There are about 14817 unique trip IDs, 1508 unique source centers, 1481 unique destination_centers, 690 unique source cities, 806 unique destination cities. Most of the data is for testing than for training. Most common route type is Carting.
6. Maximum number of trips originated from Mumbai city followed by Gurgaon Delhi, Bengaluru and Bhiwandi. That means that the seller base is strong in these cities.
7. Maximum number of trips ended in Maharashtra state followed by Karnataka, Haryana, Tamil Nadu and Uttar Pradesh. That means that the number of orders placed in these states is significantly high.
8. Maximum number of trips ended in Mumbai city followed by Bengaluru, Gurgaon, Delhi and Chennai. That means that the number of orders placed in these cities is significantly high.

Recommendations:

1. **Route Optimization:** Utilize segment-level data for fine-grained route optimization. Consider alternative routes for corridors where OSRM underestimates time and distance.
2. **Resource Allocation:** Allocate resources strategically based on order origins and destinations. Focus on optimizing logistics in high-volume areas like Bengaluru.

3. **Delivery Time Expectations:** Set realistic delivery time expectations with customers by incorporating actual time data rather than solely relying on OSRM estimates.
4. **External Data Integration:** Explore integrating weather, traffic, and road closure data to enhance route optimization and delivery time predictions.
5. **Customer Segmentation:** Analyze customer behavior and preferences to tailor delivery services. Offer premium options for time-sensitive deliveries and flexible windows for others.
6. **Data-Driven Decisions:** Leverage data analysis and insights to inform operational planning and decision-making. Continuously monitor data to identify trends, bottlenecks, and opportunities for improvement.

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```