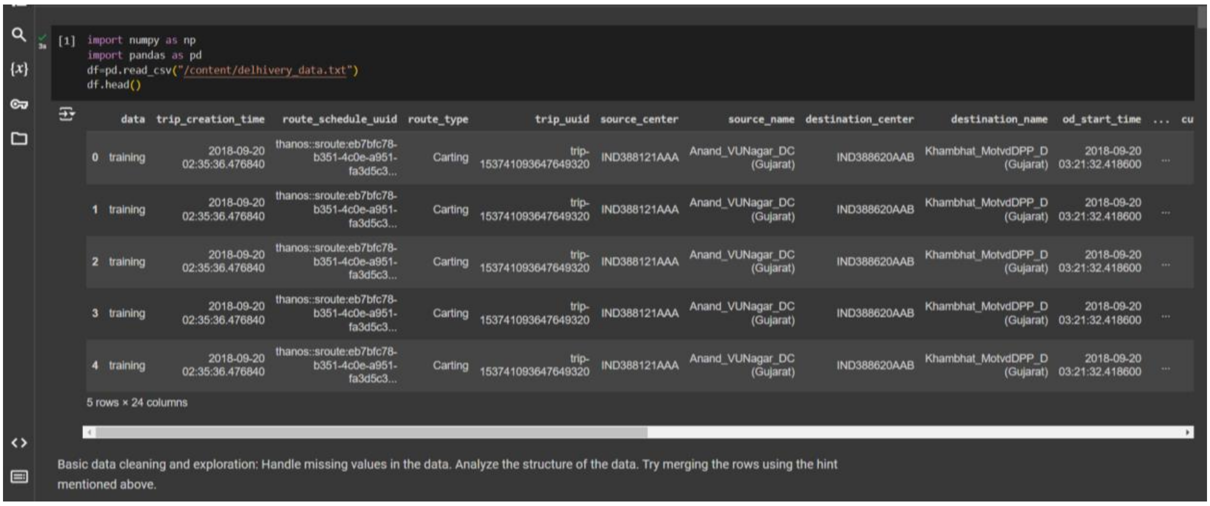
1. Define the Problem Statement

- **Objective**: The goal is to analyze trip data to understand patterns in delivery times, distances, and other route-related attributes. The insights will be used to improve route efficiency, identify key factors impacting delivery performance, and provide actionable recommendations for business improvements.
- **Key Aspects**:
    - Identify the main sources of delays or extended delivery times.
    - Explore the relationship between distance, time, and route type.
    - Detect patterns in trip creation times to optimize resource allocation.

---

2. Perform Exploratory Data Analysis (EDA)

- **Data Overview**:
    - **Shape of Data**: Look at the number of rows and columns.
    -



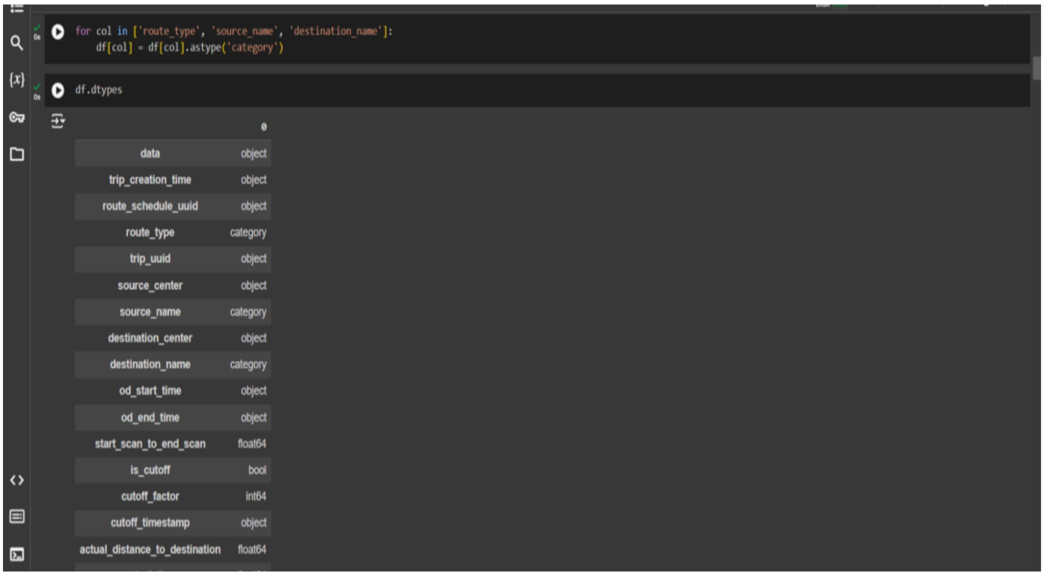
```
[1] import numpy as np
    import pandas as pd
    df=pd.read_csv("/content/delhivery_data.txt")
    df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time	...	cu
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_MotvdDPP_D (Gujarat)	2018-09-20 03:21:32.418600	...	

5 rows × 24 columns

Basic data cleaning and exploration: Handle missing values in the data. Analyze the structure of the data. Try merging the rows using the hint mentioned above.

- ○ **Convert Categorical Attributes to 'Category'**:
- ○ **Data Types**: List data types of all columns using df.dtypes.

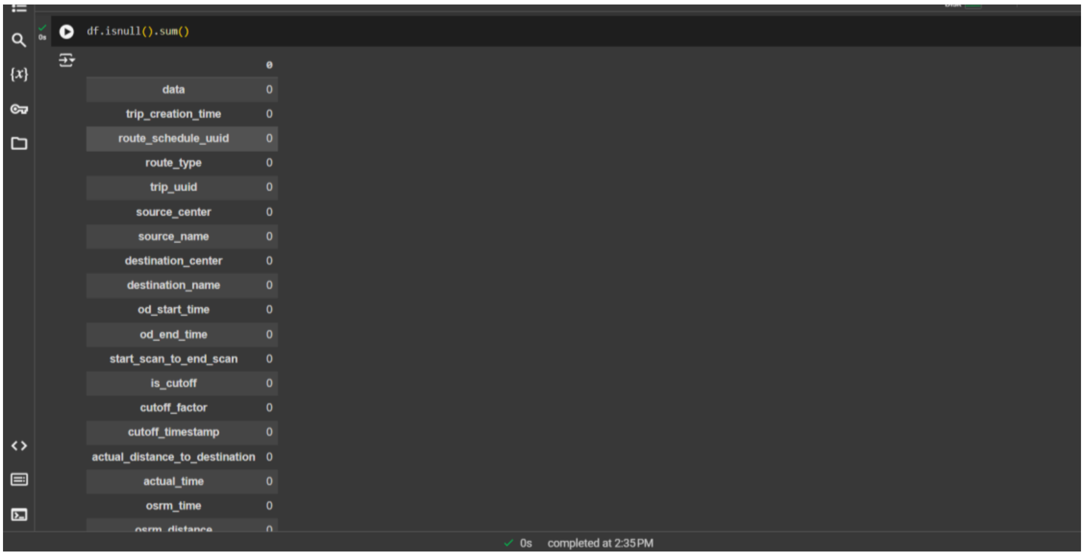


```
for col in ['route_type', 'source_name', 'destination_name']:
    df[col] = df[col].astype('category')
```

```
df.dtypes
```

	0
data	object
trip_creation_time	object
route_schedule_uuid	object
route_type	category
trip_uuid	object
source_center	object
source_name	category
destination_center	object
destination_name	category
od_start_time	object
od_end_time	object
start_scan_to_end_scan	float64
is_cutoff	bool
cutoff_factor	int64
cutoff_timestamp	object
actual_distance_to_destination	float64

- **Missing Values**:
  - ○ Detect missing values with final_df.isnull().sum().



```
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

- **Statistical Summary**:
  - View the summary statistics for numerical columns using final_df.describe().

```
df.describe()
```

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	factor	segment_actual_time	segment_osrm_time	segment_
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	
mean	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	2.120107	36.196111	18.507548	
std	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	1.715421	53.571158	14.775960	
min	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	0.144000	-244.000000	0.000000	
25%	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	1.604264	20.000000	11.000000	
50%	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	1.857143	29.000000	17.000000	
75%	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	2.213483	40.000000	22.000000	
max	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	77.387097	3051.000000	1611.000000	

  - For categorical columns, use final_df.describe(include='category').

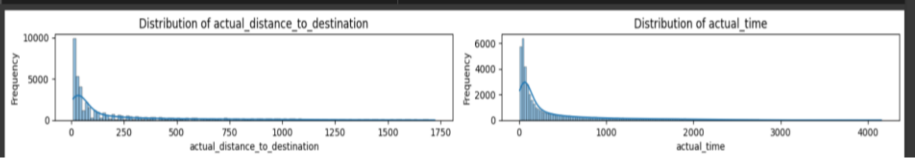
```
df.describe(include='category')
```

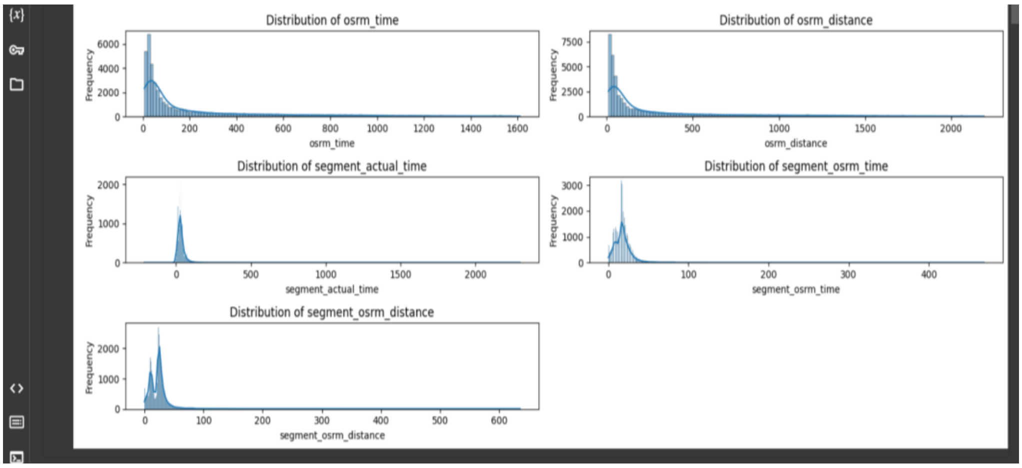
	route_type	source_name	destination_name
count	144316	144316	144316
unique	2	1496	1466
top	FTL	Gurgaon_Bilaspur_HB (Haryana)	Gurgaon_Bilaspur_HB (Haryana)
freq	99132	23267	15192

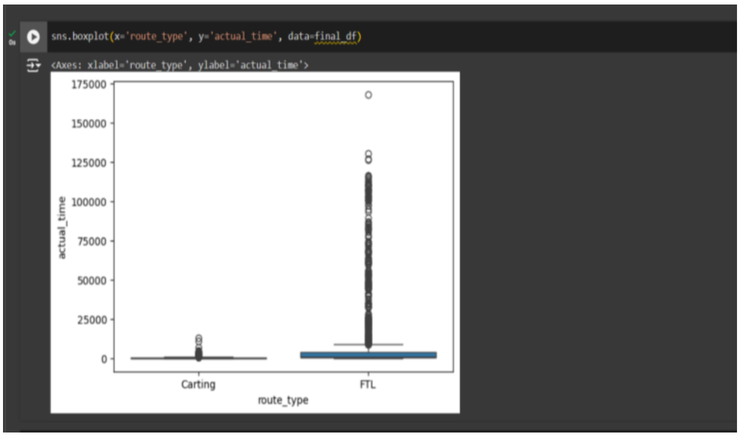
0s completed at 2:46 PM

- **Visual Analysis**:
  - **Continuous Variables**:
    - Plot distributions of each continuous variable to observe spread, skewness, and potential outliers.

```
# List of continuous variables you want to plot
continuous_cols = ['actual_distance_to_destination', 'actual_time',
                   'osrm_time', 'osrm_distance', 'segment_actual_time',
                   'segment_osrm_time','segment_osrm_distance'
                  ]

# Calculate the number of rows and columns for subplots based on the number of variables
num_cols = len(continuous_cols)
num_rows = (num_cols + 1) // 2  # Calculate rows, ensuring enough space

# Set up the plotting area for multiple subplots with adjusted layout
plt.figure(figsize=(15, 8))

# Loop through each continuous variable and create a histogram plot with KDE
for i, col in enumerate(continuous_cols, 1):
    plt.subplot(num_rows, 2, i)  # Dynamically adjust layout
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjusts spacing between plots
plt.show()
```





- **Categorical Variables**:
  - Use box plots to understand the distribution and spread of categorical 

variables.
- **Insights from EDA**:
  - Identify outliers in attributes like actual_time and actual_distance_to_destination.
  - Comment on the range of each attribute, noting variables with high variance.
  - Observe relationships between variables using correlation heatmaps and pairplots.
  - Insights into potential relationships, e.g., actual_time vs. osrm_time or actual_distance_to_destination vs. osrm_distance.

---

3. Feature Creation

- **Extract features from columns**:
  - Split destination_name and source_name into city, place code, and state.
  - Extract year, month, and day from trip_creation_time.
  - Calculate additional metrics, e.g., time_taken_to_reach_destination.

```python
[19] # 1. Split and extract features from 'destination_name'
     final_df['destination_city_place_code'] = final_df['destination_name'].str.extract(r'([A-Za-z_]+)')
     final_df['destination_state'] = final_df['destination_name'].str.extract(r'\(([^)]+)\)')
```

```python
[20] # 2. Split and extract features from 'source_name'
     final_df['source_city_place_code'] = final_df['source_name'].str.extract(r'([A-Za-z_]+)')
     final_df['source_state'] = final_df['source_name'].str.extract(r'\(([^)]+)\)')
```

```python
[21] # Convert 'trip_creation_time' to datetime format
     final_df['trip_creation_time'] = pd.to_datetime(final_df['trip_creation_time'])
```

```python
# 3. Extract features from 'trip_creation_time'
final_df['trip_year'] = final_df['trip_creation_time'].dt.year
final_df['trip_month'] = final_df['trip_creation_time'].dt.month
final_df['trip_day_of_week'] = final_df['trip_creation_time'].dt.day_name()
final_df['trip_day'] = final_df['trip_creation_time'].dt.day
final_df['trip_hour'] = final_df['trip_creation_time'].dt.hour
```

In-depth analysis and feature engineering: *Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the*

```python
[20] final_df['source_city_place_code'] = final_df['source_name'].str.extract(r'([A-Za-z_]+)')
     final_df['source_state'] = final_df['source_name'].str.extract(r'\(([^)]+)\)')
```

```python
[21] # Convert 'trip_creation_time' to datetime format
     final_df['trip_creation_time'] = pd.to_datetime(final_df['trip_creation_time'])
```

```python
[22] # 3. Extract features from 'trip_creation_time'
     final_df['trip_year'] = final_df['trip_creation_time'].dt.year
     final_df['trip_month'] = final_df['trip_creation_time'].dt.month
     final_df['trip_day_of_week'] = final_df['trip_creation_time'].dt.day_name()
     final_df['trip_day'] = final_df['trip_creation_time'].dt.day
     final_df['trip_hour'] = final_df['trip_creation_time'].dt.hour
```

```python
final_df.head()
```

ata	...	start_scan_to_end_scan	destination_city_place_code	destination_state	source_city_place_code	source_state	trip_year	trip_month	trip_day_of_week	trip_day	trip_hour
ing	...	98.0	Delhi_Bhogal	Delhi	Delhi_Lajpat_IP	Delhi	2018	9	Wednesday	12	0
ing	...	38.0	Faridabad	Haryana	FBD_Balabhgarh_DPC	Haryana	2018	9	Wednesday	12	0
ing	...	780.0	MAA_Poonamallee_HB	Tamil Nadu	Chennai_Chrompet_L	Tamil Nadu	2018	9	Wednesday	12	1
ing	...	150.0	Silvassa_Samrvmi_D	Dadra and Nagar Haveli	Vapi_IndEstat_I	Gujarat	2018	9	Wednesday	12	1
ing	...	228.0	Dhoraji_JmnvadRd_DC	Gujarat	Jetpur_DC	Gujarat	2018	9	Wednesday	12	1

---

4. Merging Rows and Aggregation

- **Grouping by Trip_uuid**:
    - Aggregate based on Trip_uuid, preserving the first and last values for selected fields.

+ Code + Text All changes saved

```python
# Step 1: Aggregation based on Trip_uuid, Source ID, and Destination ID
grouped_df = df_new.groupby(['trip_uuid', 'source_center', 'destination_center']).agg({
    'actual_distance_to_destination': 'sum',  # Sum the distances
    'actual_time': 'sum',                     # Sum the times
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
    'segment_factor': 'sum',                  # Sum for segment_factor or other numeric fields
    # For categorical or non-summable fields, you can keep first or last values
    'data': 'first',                          # Keeping the first value
    'trip_creation_time': 'first',            # Keeping the first value
    'route_schedule_uuid': 'first',           # Keeping the first value
    'route_type': 'first',                    # Keeping the first value
    'source_name': 'first',                   # Keeping the first value
    'destination_name': 'first',              # Keeping the first value
    'od_start_time': 'first',                 # Keeping the first value
    'od_end_time': 'last',                    # Keeping the last value
    'start_scan_to_end_scan': 'sum',          # Sum this numeric field
}).reset_index()
```

```python
[16] # Step 2: Further aggregation based on Trip_uuid only
     final_df = grouped_df.groupby('trip_uuid').agg({
         'actual_distance_to_destination': 'sum',  # Further sum for the entire trip
         'actual_time': 'sum',
         'osrm_time': 'sum',
         'osrm_distance': 'sum',
         'segment_actual_time': 'sum',
         'segment_osrm_time': 'sum',
         'segment_osrm_distance': 'sum',
         'segment_factor': 'sum',
         # Keep the first or last values for non-numeric fields as appropriate
```

```
    'segment_factor': 'sum',                  # Sum for segment_factor or other numeric fields
    # For categorical or non-summable fields, you can keep first or last values
    'data': 'first',                          # Keeping the first value
    'trip_creation_time': 'first',            # Keeping the first value
    'route_schedule_uuid': 'first',           # Keeping the first value
    'route_type': 'first',                    # Keeping the first value
    'source_name': 'first',                   # Keeping the first value
    'destination_name': 'first',              # Keeping the first value
    'od_start_time': 'first',                 # Keeping the first value
    'od_end_time': 'last',                    # Keeping the last value
    'start_scan_to_end_scan': 'sum',          # Sum this numeric field
}).reset_index()
```

```
# Step 2: Further aggregation based on Trip_uuid only
final_df = grouped_df.groupby('trip_uuid').agg({
    'actual_distance_to_destination': 'sum',  # Further sum for the entire trip
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
    'segment_factor': 'sum',
    # Keep the first or last values for non-numeric fields as appropriate
    'data': 'first',
    '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'
}).reset_index()
```

---

5. Comparison and Visualization of Time and Distance Fields

- **Time Fields**:
  - Compare actual_time and osrm_time to check for discrepancies.

```
from scipy.stats import ttest_ind
from scipy.stats import f_oneway
from scipy.stats import ttest_rel
import matplotlib.pyplot as plt
import seaborn as sns
# Paired t-test: Comparing actual_time with start_scan_to_end_scan
# Directly call ttest_rel instead of stats.ttest_rel
t_stat_actual, p_value_actual = ttest_rel(final_df['actual_time'], final_df['start_scan_to_end_scan'])

# Paired t-test: Comparing segment_actual_time with start_scan_to_end_scan
# Directly call ttest_rel instead of stats.ttest_rel
t_stat_segment, p_value_segment = ttest_rel(final_df['segment_actual_time'], final_df['start_scan_to_end_scan'])

# Results of hypothesis testing
print(f"Paired T-test (actual_time vs start_scan_to_end_scan): t-statistic = {t_stat_actual}, p-value = {p_value_actual}")
print(f"Paired T-test (segment_actual_time vs start_scan_to_end_scan): t-statistic = {t_stat_segment}, p-value = {p_value_segment}")

# Visual Analysis: Boxplot for the three variables
plt.figure(figsize=(10, 6))
sns.boxplot(data=[final_df['actual_time'], final_df['segment_actual_time'], final_df['start_scan_to_end_scan']], palette='Set3')
plt.xticks([0, 1, 2], ['Actual Time', 'Segment Actual Time', 'Start to End Scan'])
plt.title('Boxplot of Actual Time, Segment Actual Time, and Start to End Scan')
plt.show()

# ANOVA test if comparing all three together
# Directly call f_oneway instead of stats.f_oneway
anova_stat, anova_p_value = f_oneway(final_df['actual_time'], final_df['segment_actual_time'], final_df['start_scan_to_end_scan'])
print(f"ANOVA Test: statistic = {anova_stat}, p-value = {anova_p_value}")
```

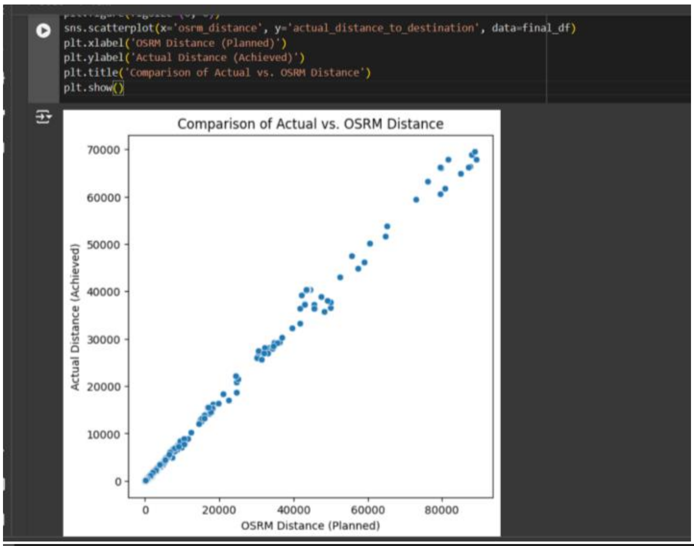
```
Paired T-test (actual_time vs start_scan_to_end_scan): t-statistic = -34.777431075476514, p-value = 8.585866880917482e-255
Paired T-test (segment_actual_time vs start_scan_to_end_scan): t-statistic = -33.10951648563558, p-value = 5.790776738896952e-232
```

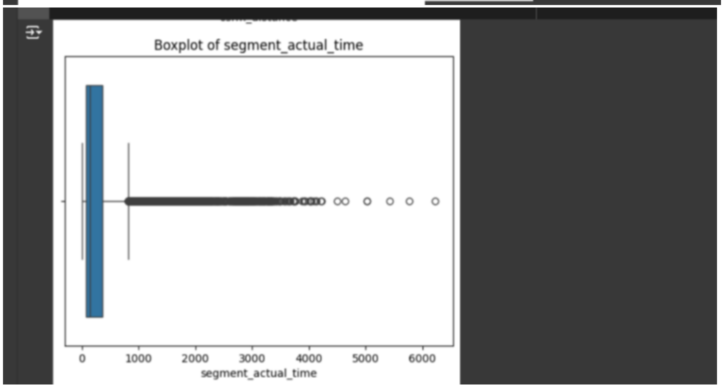
```
ttest_rel(final_df['osrm_time'], final_df['actual_time'])
```

```
TtestResult(statistic=-32.42514605229827, pvalue=7.093080524136582e-223, df=14786)
```

- **Distance Fields**:

- Plot the comparison between actual_distance_to_destination and osrm_distance.





---

6. Missing Values Treatment & Outlier Treatment

- **Missing Value Treatment**:
  - Use imputation methods like mean, median, or forward fill depending on the nature of the missing data.
- **Outlier Treatment**:

- Calculate IQR for each column, filtering out values beyond the upper and lower bounds.

```
[28] numerical_cols=final_df.select_dtypes(include=['float64','number']).columns
numerical_cols
```

```
Index(['actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'segment_factor', 'start_scan_to_end_scan',
       'trip_year', 'trip_month', 'trip_day', 'trip_hour'],
      dtype='object')
```

```
for col in numerical_cols:
    q1 = final_df[col].quantile(0.25)
    q3 = final_df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    final_df = final_df[(final_df[col] >= lower_bound) & (final_df[col] <= upper_bound)]
final_df.head()
```

	trip_uuid	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance	segment_factor	data	...
1	trip-153671079956500691	19.282605	35.0	16.0	19.9606	23.0	14.0	16.0860	3.333333	training	...
2	trip-153671110078355292	9.396526	17.0	9.0	10.8159	17.0	9.0	10.8159	1.888889	training	...
11	trip-153671464491487828	54.100365	119.0	77.0	87.0497	71.0	50.0	49.2386	4.772727	training	...
13	trip-153671506098357680	24.099697	58.0	21.0	26.5578	36.0	17.0	23.5739	4.305556	training	...
15	trip-153671531316758155	84.596916	147.0	65.0	89.3950	119.0	48.0	66.1415	10.559295	training	...

5 rows × 27 columns

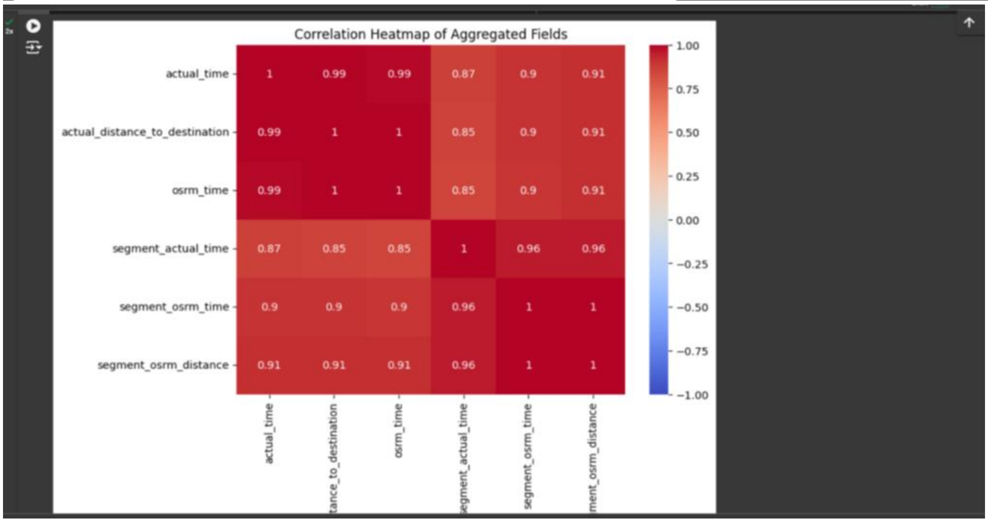
0s completed at 9:05PM

---

7. Checking Relationship Between Aggregated Fields

- **Correlations**:
    - Use a heatmap to explore correlations between aggregated fields like actual_time, actual_distance_to_destination, and other fields.

```
# Select aggregated numerical fields for correlation
aggregated_fields = ['actual_time', 'actual_distance_to_destination', 'osrm_time',
                     'segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance']

# Calculate correlation matrix
correlation_matrix = final_df[aggregated_fields].corr()

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Aggregated Fields')
plt.show()
```



- **Visual Analysis**:
    - Plot scatter plots to see how aggregated fields interact.

- Example: Relationship between cumulative distance and time can reveal insights about delivery efficiency.

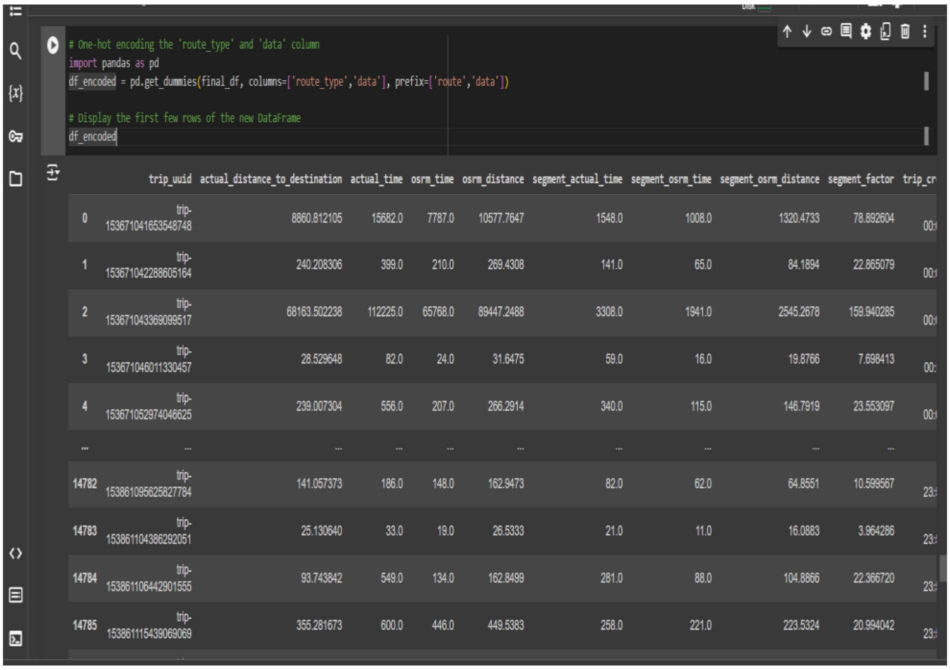
---

8. Handling Categorical Values

- **One-Hot Encoding**:
  - Convert categorical columns like route_type to one-hot encoding to prepare them for machine learning models.

    for col in ['route_type', 'source_name', 'destination_name']:

    final_df[col] = final_df[col].astype('category')

  - 

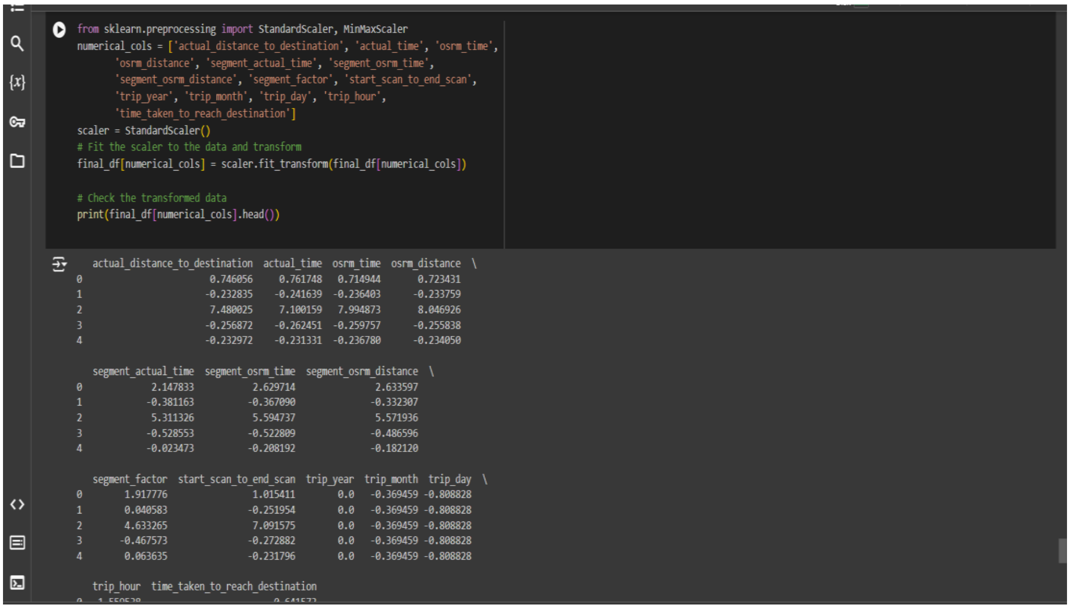
```
# One-hot encoding the 'route_type' and 'data' column
import pandas as pd
df_encoded = pd.get_dummies(final_df, columns=['route_type','data'], prefix=['route','data'])

# Display the first few rows of the new DataFrame
df_encoded
```

	trip_uuid	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance	segment_factor	trip_cr
0	trip-153671041653548748	8860.812105	15682.0	7787.0	10577.7647	1548.0	1008.0	1320.4733	78.892604	00:
1	trip-153671042288605164	240.208306	399.0	210.0	269.4308	141.0	65.0	84.1894	22.865079	00:
2	trip-153671043369099517	68163.502238	112225.0	65768.0	89447.2488	3308.0	1941.0	2545.2678	159.940285	00:
3	trip-153671046011330457	28.529648	82.0	24.0	31.6475	59.0	16.0	19.8766	7.698413	00:
4	trip-153671052974046625	239.007304	556.0	207.0	266.2914	340.0	115.0	146.7919	23.553097	00:
...	...	...	...	...	...	...	...	...	...	
14782	trip-153861095625827784	141.057373	186.0	148.0	162.9473	82.0	62.0	64.8551	10.599567	23:
14783	trip-153861104386292051	25.130640	33.0	19.0	26.5333	21.0	11.0	16.0883	3.964286	23:
14784	trip-153861106442901555	93.743842	549.0	134.0	162.8499	281.0	88.0	104.8866	22.366720	23:
14785	trip-153861115439069069	355.281673	600.0	446.0	449.5383	258.0	221.0	223.5324	20.994042	23:

---

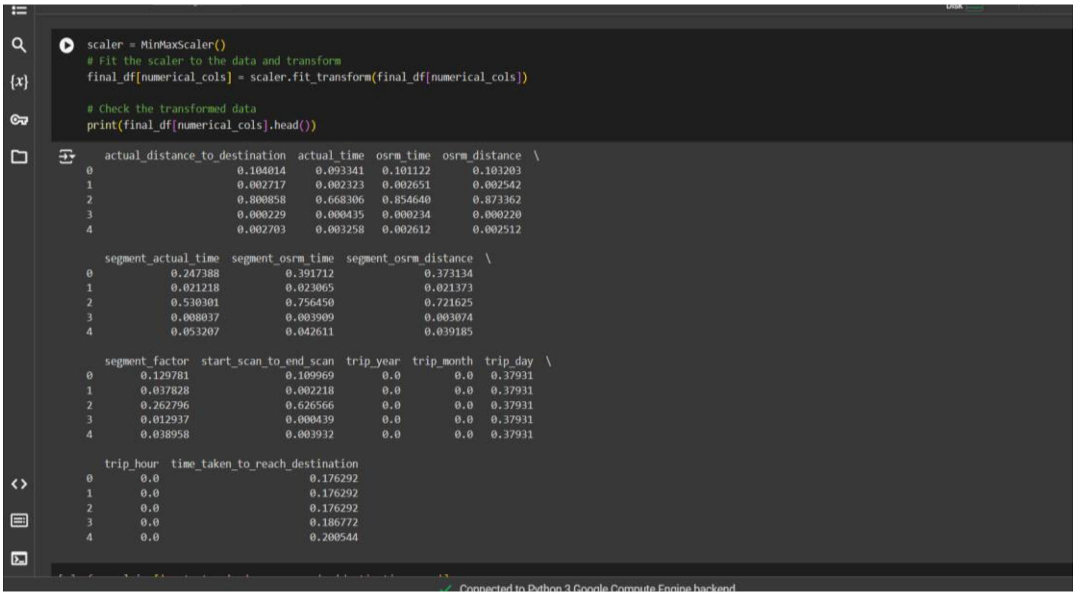
9. Column Normalization / Column Standardization

- Use **MinMaxScaler** for normalization or **StandardScaler** for standardization based on your analysis needs.



```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
numerical_cols = ['actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'segment_factor', 'start_scan_to_end_scan',
       'trip_year', 'trip_month', 'trip_day', 'trip_hour',
       'time_taken_to_reach_destination']
scaler = StandardScaler()
# Fit the scaler to the data and transform
final_df[numerical_cols] = scaler.fit_transform(final_df[numerical_cols])

# Check the transformed data
print(final_df[numerical_cols].head())
```

```
   actual_distance_to_destination  actual_time  osrm_time  osrm_distance  \
0                        0.746056     0.761748   0.714944       0.723431
1                       -0.232835    -0.241639  -0.236403      -0.233759
2                        7.480025     7.100159   7.994873       8.046926
3                       -0.256872    -0.262451  -0.259757      -0.255838
4                       -0.232972    -0.231331  -0.236780      -0.234050

   segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0             2.147833           2.629714               2.633597
1            -0.381163          -0.367090              -0.332307
2             5.311326           5.594737               5.571936
3            -0.528553          -0.522809              -0.486596
4            -0.023473          -0.208192              -0.182120

   segment_factor  start_scan_to_end_scan  trip_year  trip_month  trip_day  \
0        1.917776                1.015411        0.0   -0.369459 -0.808828
1        0.040583               -0.251954        0.0   -0.369459 -0.808828
2        4.633265                7.091575        0.0   -0.369459 -0.808828
3       -0.467573               -0.272882        0.0   -0.369459 -0.808828
4        0.063635               -0.231796        0.0   -0.369459 -0.808828

   trip_hour  time_taken_to_reach_destination
```



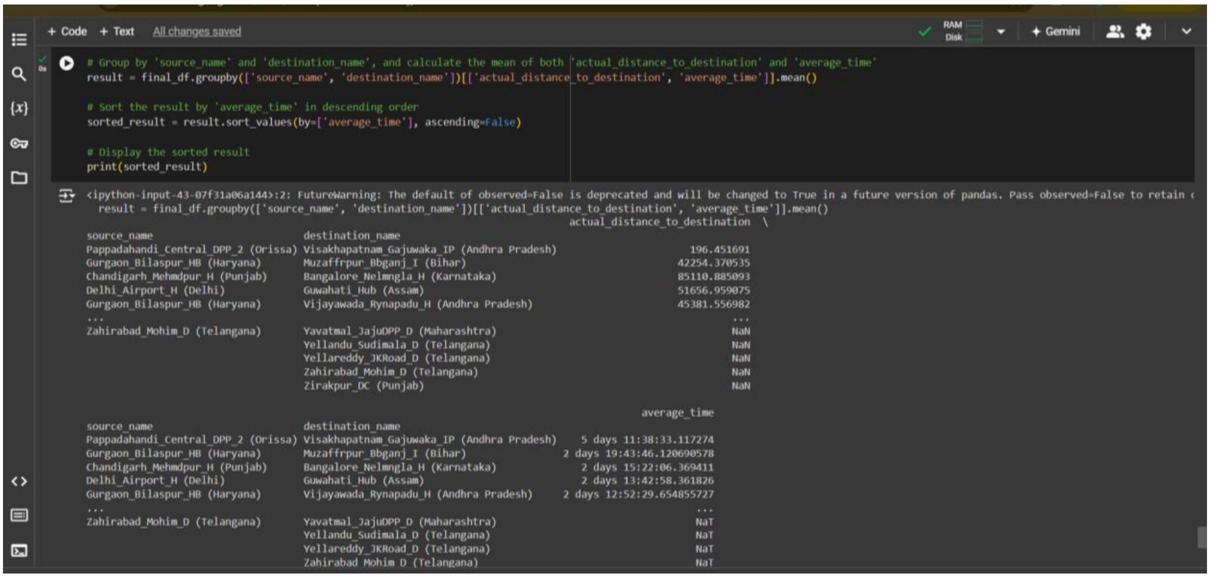
```
scaler = MinMaxScaler()
# Fit the scaler to the data and transform
final_df[numerical_cols] = scaler.fit_transform(final_df[numerical_cols])

# Check the transformed data
print(final_df[numerical_cols].head())
```

```
   actual_distance_to_destination  actual_time  osrm_time  osrm_distance  \
0                        0.104014     0.093341   0.101122       0.103203
1                        0.002717     0.002323   0.002651       0.002542
2                        0.800858     0.668306   0.854640       0.873362
3                        0.000229     0.000435   0.000234       0.000220
4                        0.002703     0.003258   0.002612       0.002512

   segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0             0.247388           0.391712               0.373134
1             0.021218           0.023065               0.021373
2             0.530301           0.756450               0.721625
3             0.008037           0.003909               0.003074
4             0.053207           0.042611               0.039185

   segment_factor  start_scan_to_end_scan  trip_year  trip_month  trip_day  \
0        0.129781                0.109969        0.0         0.0   0.37931
1        0.037828                0.002218        0.0         0.0   0.37931
2        0.262796                0.626566        0.0         0.0   0.37931
3        0.012937                0.000439        0.0         0.0   0.37931
4        0.038958                0.003932        0.0         0.0   0.37931

   trip_hour  time_taken_to_reach_destination
0        0.0                         0.176292
1        0.0                         0.176292
2        0.0                         0.176292
3        0.0                         0.186772
4        0.0                         0.200544
```


-

10. Business Insights

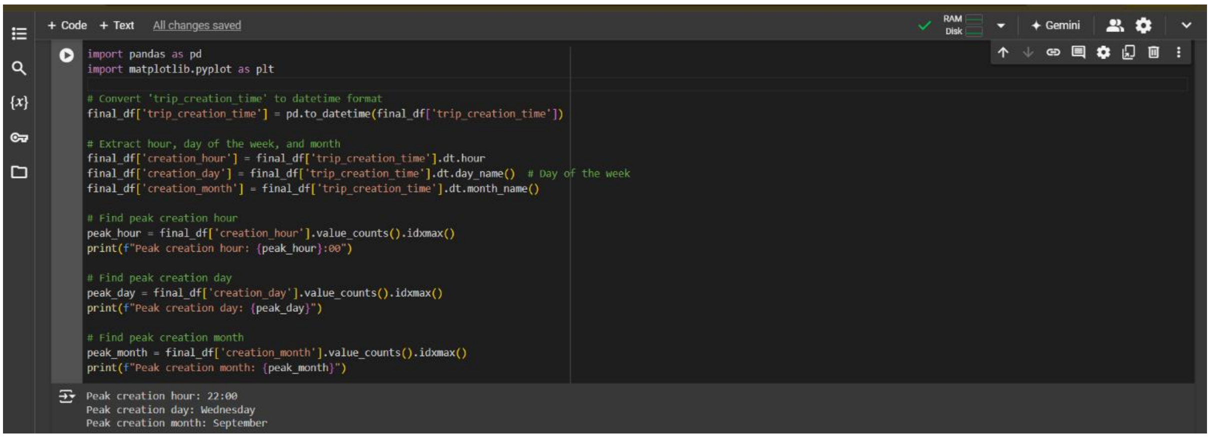
- **Pattern Analysis**:
  - Identify the busiest corridors, average time, and average distance between destinations.



```
# Group by 'source_name' and 'destination_name', and calculate the mean of both 'actual_distance_to_destination' and 'average_time'
result = final_df.groupby(['source_name', 'destination_name'])[['actual_distance_to_destination', 'average_time']].mean()

# Sort the result by 'average_time' in descending order
sorted_result = result.sort_values(by=['average_time'], ascending=False)

# Display the sorted result
print(sorted_result)
```

```
<ipython-input-43-07f31a06a144>:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain c
  result = final_df.groupby(['source_name', 'destination_name'])[['actual_distance_to_destination', 'average_time']].mean()
                                                                                          actual_distance_to_destination  \
source_name                        destination_name
Pappadahandi_Central_DPP_2 (Orissa) Visakhapatnam_Gajuwaka_IP (Andhra Pradesh)                        196.451691
Gurgaon_Bilaspur_HB (Haryana)       Muzaffrpur_Bbganj_I (Bihar)                                      42254.370535
Chandigarh_Mehmdpur_H (Punjab)      Bangalore_Nelmngla_H (Karnataka)                                 85110.885093
Delhi_Airport_H (Delhi)             Guwahati_Hub (Assam)                                             51656.959075
Gurgaon_Bilaspur_HB (Haryana)       Vijayawada_Rynapadu_H (Andhra Pradesh)                           45381.556982
...                                                                                                          ...
Zahirabad_Mohim_D (Telangana)       Yavatmal_JajuDPP_D (Maharashtra)                                           NaN
                                    Yellandu_Sudimala_D (Telangana)                                            NaN
                                    Yellareddy_JKRoad_D (Telangana)                                            NaN
                                    Zahirabad_Mohim_D (Telangana)                                              NaN
                                    Zirakpur_DC (Punjab)                                                       NaN

                                                                                                  average_time
source_name                        destination_name
Pappadahandi_Central_DPP_2 (Orissa) Visakhapatnam_Gajuwaka_IP (Andhra Pradesh)     5 days 11:38:33.117274
Gurgaon_Bilaspur_HB (Haryana)       Muzaffrpur_Bbganj_I (Bihar)                 2 days 19:43:46.120690578
Chandigarh_Mehmdpur_H (Punjab)      Bangalore_Nelmngla_H (Karnataka)               2 days 15:22:06.369411
Delhi_Airport_H (Delhi)             Guwahati_Hub (Assam)                           2 days 13:42:58.361826
Gurgaon_Bilaspur_HB (Haryana)       Vijayawada_Rynapadu_H (Andhra Pradesh)      2 days 12:52:29.654855727
...                                                                                                   ...
Zahirabad_Mohim_D (Telangana)       Yavatmal_JajuDPP_D (Maharashtra)                                  NaT
                                    Yellandu_Sudimala_D (Telangana)                                   NaT
                                    Yellareddy_JKRoad_D (Telangana)                                   NaT
                                    Zahirabad Mohim D (Telangana)                                     NaT
```

  - Observe which states or regions are generating the highest volume of deliveries.
  - Insights into peak trip creation times and delays.



```
import pandas as pd
import matplotlib.pyplot as plt

# Convert 'trip_creation_time' to datetime format
final_df['trip_creation_time'] = pd.to_datetime(final_df['trip_creation_time'])

# Extract hour, day of the week, and month
final_df['creation_hour'] = final_df['trip_creation_time'].dt.hour
final_df['creation_day'] = final_df['trip_creation_time'].dt.day_name()  # Day of the week
final_df['creation_month'] = final_df['trip_creation_time'].dt.month_name()

# Find peak creation hour
peak_hour = final_df['creation_hour'].value_counts().idxmax()
print(f"Peak creation hour: {peak_hour}:00")

# Find peak creation day
peak_day = final_df['creation_day'].value_counts().idxmax()
print(f"Peak creation day: {peak_day}")

# Find peak creation month
peak_month = final_df['creation_month'].value_counts().idxmax()
print(f"Peak creation month: {peak_month}")
```

```
Peak creation hour: 22:00
Peak creation day: Wednesday
Peak creation month: September
```

---

11. Recommendations

- **Route Optimization**:
  - Focus on optimizing the busiest routes by redistributing resources during peak times.
- **State-Specific Focus**:
  - Increase delivery resources in states with the highest order volume.
- **Improving Time Estimates**:
  - Update osrm_time estimates where there is a significant discrepancy between planned and actual times.