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

```
# Load the dataset
try:
    df = pd.read_csv('delhivery_data.csv')
    print("Dataset loaded successfully.")
    print("Shape of the dataset:", df.shape)
    print("\nFirst 5 rows:")
    print(df.head())
    print("\nData Info:")
    df.info()
    print("\nMissing values before handling:")
    print(df.isnull().sum())
    print("\nStatistical Summary:")
    print(df.describe(include='all'))

except FileNotFoundError:
    print("Error: 'delhivery_data.csv' not found in the current directory.")
    exit()
```

```
Dataset loaded successfully.
Shape of the dataset: (144867, 24)

First 5 rows:
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

                              route_schedule_uuid route_type  \
0  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
1  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
2  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
3  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting
4  thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...    Carting

                 trip_uuid source_center                source_name  \
0  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
1  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
2  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
3  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)
4  trip-153741093647649320  IND388121AAA  Anand_VUNagar_DC (Gujarat)

  destination_center              destination_name  \
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)

                od_start_time  ...            cutoff_timestamp  \
0  2018-09-20 03:21:32.418600  ...         2018-09-20 04:27:55
1  2018-09-20 03:21:32.418600  ...         2018-09-20 04:17:55
2  2018-09-20 03:21:32.418600  ...  2018-09-20 04:01:19.505586
3  2018-09-20 03:21:32.418600  ...         2018-09-20 03:39:57
4  2018-09-20 03:21:32.418600  ...         2018-09-20 03:33:55

   actual_distance_to_destination  actual_time  osrm_time osrm_distance  \
0                       10.435660         14.0       11.0       11.9653
1                       18.936842         24.0       20.0       21.7243
2                       27.637279         40.0       28.0       32.5395
3                       36.118028         62.0       40.0       45.5620
4                       39.386040         68.0       44.0       54.2181

     factor  segment_actual_time  segment_osrm_time  segment_osrm_distance  \
0  1.272727                 14.0               11.0                11.9653
1  1.200000                 10.0                9.0                 9.7590
2  1.428571                 16.0                7.0                10.8152
3  1.550000                 21.0               12.0                13.0224
4  1.545455                  6.0                5.0                 3.9153

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
```

```
# --- Basic Data Cleaning and Exploration ---

# 1. Handle Missing Values
# Strategy: Fill numerical NaNs with median/mean, categorical with mode or 'Unknown'.
# Let's examine the columns with missing values again.
# source_name, destination_name, od_start_time, od_end_time have few missing values.
# segment_actual_time, segment_osrm_time, segment_osrm_distance, segment_factor have many.
# is_cutoff, cutoff_factor, cutoff_timestamp have almost all missing - likely drop these.
# factor also has many missing values.

print("\nHandling Missing Values...")

# Drop columns with too many missing values
cols_to_drop = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']
df.drop(columns=cols_to_drop, inplace=True)
print(f"Dropped columns: {cols_to_drop}")

# Fill missing categorical names with 'Unknown'
df['source_name'].fillna('Unknown', inplace=True)
df['destination_name'].fillna('Unknown', inplace=True)

# Convert timestamp columns to datetime objects, coercing errors
time_cols = ['trip_creation_time', 'od_start_time', 'od_end_time']
for col in time_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')

# Check for NaT values introduced by coercion
print("\nNaT values after datetime conversion:")
print(df[time_cols].isnull().sum())

# Fill missing timestamps - this is tricky. Filling with mean/median doesn't make sense.
# Let's drop rows where od_start_time or od_end_time is missing, as they are crucial for time calculations.
df.dropna(subset=['od_start_time', 'od_end_time'], inplace=True)
print(f"Dropped rows with missing od_start_time or od_end_time. New shape: {df.shape}")

# Fill missing numerical segment times/distances. Median might be better due to potential outliers.
num_segment_cols = ['segment_actual_time', 'segment_osrm_time', 'segment_osrm_distance']
for col in num_segment_cols:
    median_val = df[col].median()
    df[col].fillna(median_val, inplace=True)
    print(f"Filled missing values in '{col}' with median: {median_val}")

print("\nMissing values after handling:")
print(df.isnull().sum())
```

```
Handling Missing Values...
Dropped columns: ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor', 'segment_factor']

NaT values after datetime conversion:
trip_creation_time    0
od_start_time         0
od_end_time           0
dtype: int64
Dropped rows with missing od_start_time or od_end_time. New shape: (144867, 19)
Filled missing values in 'segment_actual_time' with median: 29.0
Filled missing values in 'segment_osrm_time' with median: 17.0
Filled missing values in 'segment_osrm_distance' with median: 23.513

Missing values after handling:
<ipython-input-51-a12fb893a8e2>:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['source_name'].fillna('Unknown', inplace=True)
<ipython-input-51-a12fb893a8e2>:20: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['destination_name'].fillna('Unknown', inplace=True)
<ipython-input-51-a12fb893a8e2>:40: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[col].fillna(median_val, inplace=True)
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
actual_distance_to_destination  0
actual_time                     0
osrm_time                       0
osrm_distance                   0
segment_actual_time             0
segment_osrm_time               0
segment_osrm_distance           0
dtype: int64
```

```
# 2. Feature Extraction

print("\nExtracting Features...")

# Destination Name: City-Place-Code (State)
# Assuming format 'City_Suffix (State)' e.g., 'Bengaluru_DC (KA)'
df[['destination_city', 'destination_state_code']] = df['destination_name'].str.extract(r'^([^_]+)_[^_]+\s+\(([^)]+)\)') # Corrected reg
df['destination_state_code'].fillna('Unknown', inplace=True) # Handle cases that didn't match
df['destination_city'].fillna(df['destination_name'], inplace=True) # Use full name if pattern fails

# Source Name: City-Place-Code (State)
df[['source_city', 'source_state_code']] = df['source_name'].str.extract(r'^([^_]+)_[^_]+\s+\(([^)]+)\)') # Corrected regex
df['source_state_code'].fillna('Unknown', inplace=True) # Handle cases that didn't match
df['source_city'].fillna(df['source_name'], inplace=True) # Use full name if pattern fails

# Trip Creation Time
df['trip_creation_year'] = df['trip_creation_time'].dt.year
df['trip_creation_month'] = df['trip_creation_time'].dt.month
df['trip_creation_day'] = df['trip_creation_time'].dt.day
df['trip_creation_hour'] = df['trip_creation_time'].dt.hour
df['trip_creation_weekday'] = df['trip_creation_time'].dt.weekday # Monday=0, Sunday=6

print("Features extracted from names and timestamps.")
print(df[['source_city', 'source_state_code', 'destination_city', 'destination_state_code', 'trip_creation_year', 'trip_creation_month',
```

```
Extracting Features...
Features extracted from names and timestamps.
                source_city source_state_code  \
0  Anand_VUNagar_DC (Gujarat)           Unknown
1  Anand_VUNagar_DC (Gujarat)           Unknown
2  Anand_VUNagar_DC (Gujarat)           Unknown
3  Anand_VUNagar_DC (Gujarat)           Unknown
4  Anand_VUNagar_DC (Gujarat)           Unknown

             destination_city destination_state_code  trip_creation_year  \
0  Khambhat_MotvdDPP_D (Gujarat)               Unknown                2018
1  Khambhat_MotvdDPP_D (Gujarat)               Unknown                2018
2  Khambhat_MotvdDPP_D (Gujarat)               Unknown                2018
3  Khambhat_MotvdDPP_D (Gujarat)               Unknown                2018
4  Khambhat_MotvdDPP_D (Gujarat)               Unknown                2018

   trip_creation_month  trip_creation_day
0                    9                 20
1                    9                 20
2                    9                 20
3                    9                 20
4                    9                 20
<ipython-input-52-ac1a5b199970>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col

  df['destination_state_code'].fillna('Unknown', inplace=True) # Handle cases that didn't match
<ipython-input-52-ac1a5b199970>:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col

  df['destination_city'].fillna(df['destination_name'], inplace=True) # Use full name if pattern fails
<ipython-input-52-ac1a5b199970>:13: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col

  df['source_state_code'].fillna('Unknown', inplace=True) # Handle cases that didn't match
<ipython-input-52-ac1a5b199970>:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
```

```
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col

  df['source_city'].fillna(df['source_name'], inplace=True) # Use full name if pattern fails
```

Row Merging/Aggregation

```
# --- Row Merging/Aggregation ---

print("\nMerging Rows...")

# Define aggregations
# Numeric: sum for times/distances that accumulate, maybe mean for others?
# Categorical: first/last
# Timestamps: first for start, last for end

# Aggregation Level 1: trip_uuid, source_center, destination_cente
agg_funcs_level1 = {
    'data': 'first',
    'route_schedule_uuid': 'first', # Assuming constant within this group
    'route_type': 'first',
    'trip_creation_time': 'first', # Keep first creation time for the segment
    'source_name': 'first',
    'destination_name': 'first',
    'od_start_time': 'first', # Start of the first segment
    'od_end_time': 'last',   # End of the last segment within this group
    'start_scan_to_end_scan': 'sum', # Summing time for segments
    'actual_distance_to_destination': 'first', # Assuming this is overall distance, keep first
    'actual_time': 'sum',
    'osrm_time': 'sum',
    'osrm_distance': 'sum',
    'segment_actual_time': 'sum',
    'segment_osrm_time': 'sum',
    'segment_osrm_distance': 'sum',
    # Keep extracted features
    'destination_city': 'first',
    'destination_state_code': 'first',
    'source_city': 'first',
    'source_state_code': 'first',
    'trip_creation_year': 'first',
    'trip_creation_month': 'first',
    'trip_creation_day': 'first',
    'trip_creation_hour': 'first',
    'trip_creation_weekday': 'first'
}

# Grouping requires handling potential non-unique indices if any were created
df_grouped_level1 = df.groupby(['trip_uuid', 'source_center', 'destination_center'], as_index=False).agg(agg_funcs_level1) # Corrected (

print(f"Shape after grouping by trip_uuid, source, destination: {df_grouped_level1.shape}")
print(df_grouped_level1.head())
```

```
Merging Rows...
Shape after grouping by trip_uuid, source, destination: (26368, 28)
                trip_uuid source_center destination_center      data  \
0  trip-153671041653548748  IND209304AAA       IND000000ACB  training
1  trip-153671041653548748  IND462022AAA       IND209304AAA  training
2  trip-153671042288605164  IND561203AAB       IND562101AAA  training
3  trip-153671042288605164  IND572101AAA       IND561203AAB  training
4  trip-153671043369099517  IND000000ACB       IND160002AAC  training

                               route_schedule_uuid route_type  \
0  thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...        FTL
1  thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...        FTL
2  thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...    Carting
3  thanos::sroute:3a1b0ab2-bb0b-4c53-8c59-eb2a2c0...    Carting
4  thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...        FTL

          trip_creation_time                       source_name  \
0 2018-09-12 00:00:16.535741  Kanpur_Central_H_6 (Uttar Pradesh)
1 2018-09-12 00:00:16.535741  Bhopal_Trnsport_H (Madhya Pradesh)
2 2018-09-12 00:00:22.886430   Doddablpur_ChikaDPP_D (Karnataka)
3 2018-09-12 00:00:22.886430       Tumkur_Veersagr_I (Karnataka)
4 2018-09-12 00:00:33.691250       Gurgaon_Bilaspur_HB (Haryana)

                     destination_name             od_start_time  ...  \
0       Gurgaon_Bilaspur_HB (Haryana) 2018-09-12 16:39:46.858469  ...
```

```
1  Kanpur_Central_H_6 (Uttar Pradesh) 2018-09-12 00:00:16.535741  ...
2   Chikblapur_ShntiSgr_D (Karnataka) 2018-09-12 02:03:09.655591  ...
3   Doddablpur_ChikaDPP_D (Karnataka) 2018-09-12 00:00:22.886430  ...
4      Chandigarh_Mehmdpur_H (Punjab) 2018-09-14 03:40:17.106733  ...

  segment_osrm_distance                   destination_city  \
0             670.6205       Gurgaon_Bilaspur_HB (Haryana)
1             649.8528  Kanpur_Central_H_6 (Uttar Pradesh)
2              28.1995   Chikblapur_ShntiSgr_D (Karnataka)
3              55.9899   Doddablpur_ChikaDPP_D (Karnataka)
4             317.7408      Chandigarh_Mehmdpur_H (Punjab)

   destination_state_code                        source_city  \
0                 Unknown  Kanpur_Central_H_6 (Uttar Pradesh)
1                 Unknown  Bhopal_Trnsport_H (Madhya Pradesh)
2                 Unknown   Doddablpur_ChikaDPP_D (Karnataka)
3                 Unknown       Tumkur_Veersagr_I (Karnataka)
4                 Unknown       Gurgaon_Bilaspur_HB (Haryana)

   source_state_code  trip_creation_year  trip_creation_month  \
0            Unknown                2018                    9
1            Unknown                2018                    9
2            Unknown                2018                    9
3            Unknown                2018                    9
4            Unknown                2018                    9

   trip_creation_day  trip_creation_hour trip_creation_weekday
0                 12                   0                     2
1                 12                   0                     2
2                 12                   0                     2
3                 12                   0                     2
4                 12                   0                     2
```

```
# Aggregation Level 2: trip_uuid only
# We need to decide how to aggregate the segments for a whole trip.
# Summing times/distances makes sense. For start/end times, take the overall min/max.
# For source/destination, take the first source and last destination?

# To get the overall start/end times and first/last locations correctly, sort by time first
df_sorted = df.sort_values(by=['trip_uuid', 'od_start_time'])

agg_funcs_level2 = {
    'data': 'first',
    'route_schedule_uuid': 'first', # Assuming constant for the trip
    'route_type': 'first', # Assuming constant for the trip
    'trip_creation_time': 'first', # First creation time for the trip
    'source_center': 'first', # First source center of the trip
    'source_name': 'first', # First source name
    'destination_center': 'last', # Last destination center - Corrected column name
    'destination_name': 'last', # Last destination name
    'od_start_time': 'min', # Earliest start time
    'od_end_time': 'max',   # Latest end time
    'start_scan_to_end_scan': 'sum', # Total scan-to-scan time
    'actual_distance_to_destination': 'first', # Keep the first recorded distance? Or last? Or mean? Let's take first for now.
    'actual_time': 'sum', # Total actual time
    'osrm_time': 'sum', # Total OSRM time
    'osrm_distance': 'sum', # Total OSRM distance
    'segment_actual_time': 'sum', # Sum of segment actual times
    'segment_osrm_time': 'sum', # Sum of segment OSRM times
    'segment_osrm_distance': 'sum', # Sum of segment OSRM distances
   # Keep extracted features (first/last as appropriate)
    'destination_city': 'last',
    'destination_state_code': 'last',
    'source_city': 'first',
    'source_state_code': 'first',
    'trip_creation_year': 'first',
    'trip_creation_month': 'first',
    'trip_creation_day': 'first',
    'trip_creation_hour': 'first',
    'trip_creation_weekday': 'first'
}

df_agg_trip = df_sorted.groupby('trip_uuid', as_index=False).agg(agg_funcs_level2)

print(f"\nShape after aggregating by trip_uuid: {df_agg_trip.shape}")
print(df_agg_trip.head())
print("\nAggregated Data Info:")
df_agg_trip.info()
print("\nAggregated Data Missing Values:")
print(df_agg_trip.isnull().sum()) # Check if aggregation introduced NaNs
```

```
 8   destination_name                14817 non-null  object
 9   od_start_time                   14817 non-null  datetime64[ns]
 10  od_end_time                     14817 non-null  datetime64[ns]
 11  start_scan_to_end_scan          14817 non-null  float64
 12  actual_distance_to_destination  14817 non-null  float64
 13  actual_time                     14817 non-null  float64
 14  osrm_time                       14817 non-null  float64
 15  osrm_distance                   14817 non-null  float64
 16  segment_actual_time             14817 non-null  float64
 17  segment_osrm_time               14817 non-null  float64
 18  segment_osrm_distance           14817 non-null  float64
 19  destination_city                14817 non-null  object
 20  destination_state_code          14817 non-null  object
 21  source_city                     14817 non-null  object
 22  source_state_code               14817 non-null  object
 23  trip_creation_year              14817 non-null  int32
 24  trip_creation_month             14817 non-null  int32
 25  trip_creation_day               14817 non-null  int32
 26  trip_creation_hour              14817 non-null  int32
 27  trip_creation_weekday           14817 non-null  int32
dtypes: datetime64[ns](3), float64(8), int32(5), object(12)
memory usage: 2.9+ MB

Aggregated Data Missing Values:
trip_uuid                         0
data                              0
route_schedule_uuid               0
route_type                        0
trip_creation_time                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
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
destination_city                  0
destination_state_code            0
source_city                       0
source_state_code                 0
trip_creation_year                0
trip_creation_month               0
trip_creation_day                 0
trip_creation_hour                0
trip_creation_weekday             0
dtype: int64
```

```
# --- In-depth Analysis and Feature Engineering (on df_agg_trip) ---

print("\nPerforming In-depth Analysis...")

# a. Calculate time taken between od_start_time and od_end_time
# Ensure columns are datetime
df_agg_trip['od_start_time'] = pd.to_datetime(df_agg_trip['od_start_time'])
df_agg_trip['od_end_time'] = pd.to_datetime(df_agg_trip['od_end_time'])

# Calculate difference in hours
df_agg_trip['od_time_diff_hours'] = (df_agg_trip['od_end_time'] - df_agg_trip['od_start_time']).dt.total_seconds() / 3600.0

# Drop original columns if required (optional for now)
# df_agg_trip.drop(columns=['od_start_time', 'od_end_time'], inplace=True)

print("Calculated 'od_time_diff_hours'.")
print(df_agg_trip[['od_start_time', 'od_end_time', 'od_time_diff_hours']].head())
```

```
Performing In-depth Analysis...
Calculated 'od_time_diff_hours'.
                od_start_time               od_end_time  od_time_diff_hours
0 2018-09-12 00:00:16.535741 2018-09-13 13:40:23.123744          37.668497
1 2018-09-12 00:00:22.886430 2018-09-12 03:01:59.598855           3.026865
2 2018-09-12 00:00:33.691250 2018-09-14 17:34:55.442454          65.572709
3 2018-09-12 00:01:00.113710 2018-09-12 01:41:29.809822           1.674916
4 2018-09-12 00:02:09.740725 2018-09-12 12:00:30.683231          11.972484
```

```
# b. Compare od_time_diff_hours and start_scan_to_end_scan
print("\nComparing 'od_time_diff_hours' and 'start_scan_to_end_scan'...")
# Visual Analysis
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_agg_trip, x='od_time_diff_hours', y='start_scan_to_end_scan', alpha=0.5)
```

```
plt.title('od_time_diff_hours vs. start_scan_to_end_scan')
plt.xlabel('OD Start to End Time Difference (hours)')
plt.ylabel('Sum of Scan-to-Scan Time (assumed hours)') # Clarified assumed units
plt.grid(True)
plt.savefig('od_diff_vs_scan_time.png')
plt.close()
print("Saved scatter plot: od_diff_vs_scan_time.png")

# Basic statistics of the difference
df_agg_trip['scan_od_diff'] = df_agg_trip['od_time_diff_hours'] - df_agg_trip['start_scan_to_end_scan']
print("Statistics for (od_time_diff_hours - start_scan_to_end_scan):")
print(df_agg_trip['scan_od_diff'].describe())
# Hypothesis Testing (e.g., Paired t-test if we assume they measure the same underlying duration)
# Note: Requires assumptions (normality of differences). Visual inspection suggests non-normality.
# A non-parametric test like Wilcoxon signed-rank test might be more appropriate.
try:
    stat, p_value = stats.wilcoxon(df_agg_trip['od_time_diff_hours'], df_agg_trip['start_scan_to_end_scan'])
    print(f"\nWilcoxon test between od_time_diff_hours and start_scan_to_end_scan:")
    print(f"Statistic: {stat}, p-value: {p_value}")
    if p_value < 0.05:
        print("Significant difference detected (p < 0.05).")
    else:
        print("No significant difference detected (p >= 0.05).")
except ValueError as e:
    print(f"\nCould not perform Wilcoxon test: {e}") # Might happen if identical values exist
```

```
Comparing 'od_time_diff_hours' and 'start_scan_to_end_scan'...
Saved scatter plot: od_diff_vs_scan_time.png
Statistics for (od_time_diff_hours - start_scan_to_end_scan):
count     14817.000000
mean      -9389.220868
std       33692.707172
min     -396675.684341
25%       -2815.586670
50%        -979.891122
75%        -405.110514
max         -25.558347
Name: scan_od_diff, dtype: float64

Wilcoxon test between od_time_diff_hours and start_scan_to_end_scan:
Statistic: 0.0, p-value: 0.0
Significant difference detected (p < 0.05).
```

```
# c. Compare actual_time vs OSRM time (aggregated)
print("\nComparing aggregated 'actual_time' vs 'osrm_time'...")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_agg_trip, x='osrm_time', y='actual_time', alpha=0.5)
plt.title('Aggregated Actual Time vs. OSRM Time')
plt.xlabel('Aggregated OSRM Time')
plt.ylabel('Aggregated Actual Time')
plt.plot([0, df_agg_trip[['osrm_time', 'actual_time']].max().max()], [0, df_agg_trip[['osrm_time', 'actual_time']].max().max()], ls="--'
plt.grid(True)
plt.savefig('actual_vs_osrm_time.png')
plt.close()
print("Saved scatter plot: actual_vs_osrm_time.png")

try:
    stat, p_value = stats.wilcoxon(df_agg_trip['actual_time'], df_agg_trip['osrm_time'])
    print(f"\nWilcoxon test between aggregated actual_time and osrm_time:")
    print(f"Statistic: {stat}, p-value: {p_value}")
    if p_value < 0.05:
        print("Significant difference detected (p < 0.05).")
    else:
        print("No significant difference detected (p >= 0.05).")
except ValueError as e:
    print(f"\nCould not perform Wilcoxon test: {e}")
```

```
Comparing aggregated 'actual_time' vs 'osrm_time'...
Saved scatter plot: actual_vs_osrm_time.png

Wilcoxon test between aggregated actual_time and osrm_time:
Statistic: 134624.0, p-value: 0.0
Significant difference detected (p < 0.05).
```

```
# d. Compare actual_time vs segment_actual_time (aggregated)
print("\nComparing aggregated 'actual_time' vs 'segment_actual_time'...")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_agg_trip, x='segment_actual_time', y='actual_time', alpha=0.5)
plt.title('Aggregated Actual Time vs. Aggregated Segment Actual Time')
plt.xlabel('Aggregated Segment Actual Time')
```

```
plt.ylabel('Aggregated Actual Time')
plt.plot([0, df_agg_trip[['segment_actual_time', 'actual_time']].max().max()], [0, df_agg_trip[['segment_actual_time', 'actual_time']].m
plt.grid(True)
plt.savefig('actual_vs_segment_actual_time.png')
plt.close()
print("Saved scatter plot: actual_vs_segment_actual_time.png")

try:
    stat, p_value = stats.wilcoxon(df_agg_trip['actual_time'], df_agg_trip['segment_actual_time'])
    print(f"\nWilcoxon test between aggregated actual_time and segment_actual_time:")
    print(f"Statistic: {stat}, p-value: {p_value}")
    if p_value < 0.05:
        print("Significant difference detected (p < 0.05).")
    else:
        print("No significant difference detected (p >= 0.05).")
except ValueError as e:
    print(f"\nCould not perform Wilcoxon test: {e}")
```

```
Comparing aggregated 'actual_time' vs 'segment_actual_time'...
Saved scatter plot: actual_vs_segment_actual_time.png

Wilcoxon test between aggregated actual_time and segment_actual_time:
Statistic: 0.0, p-value: 0.0
Significant difference detected (p < 0.05).
```

```
# e. Compare osrm_distance vs segment_osrm_distance (aggregated)
print("\nComparing aggregated 'osrm_distance' vs 'segment_osrm_distance'...")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_agg_trip, x='segment_osrm_distance', y='osrm_distance', alpha=0.5)
plt.title('Aggregated OSRM Distance vs. Aggregated Segment OSRM Distance')
plt.xlabel('Aggregated Segment OSRM Distance')
plt.ylabel('Aggregated OSRM Distance')
plt.plot([0, df_agg_trip[['segment_osrm_distance', 'osrm_distance']].max().max()], [0, df_agg_trip[['segment_osrm_distance', 'osrm_dista
plt.grid(True)
plt.savefig('osrm_dist_vs_segment_osrm_dist.png')
plt.close()
print("Saved scatter plot: osrm_dist_vs_segment_osrm_dist.png")

try:
    stat, p_value = stats.wilcoxon(df_agg_trip['osrm_distance'], df_agg_trip['segment_osrm_distance'])
    print(f"\nWilcoxon test between aggregated osrm_distance and segment_osrm_distance:")
    print(f"Statistic: {stat}, p-value: {p_value}")
    if p_value < 0.05:
        print("Significant difference detected (p < 0.05).")
    else:
        print("No significant difference detected (p >= 0.05).")
except ValueError as e:
    print(f"\nCould not perform Wilcoxon test: {e}")
```

```
Comparing aggregated 'osrm_distance' vs 'segment_osrm_distance'...
Saved scatter plot: osrm_dist_vs_segment_osrm_dist.png

Wilcoxon test between aggregated osrm_distance and segment_osrm_distance:
Statistic: 26304.0, p-value: 0.0
Significant difference detected (p < 0.05).
```

```
# f. Compare osrm_time vs segment_osrm_time (aggregated)
print("\nComparing aggregated 'osrm_time' vs 'segment_osrm_time'...")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_agg_trip, x='segment_osrm_time', y='osrm_time', alpha=0.5)
plt.title('Aggregated OSRM Time vs. Aggregated Segment OSRM Time')
plt.xlabel('Aggregated Segment OSRM Time')
plt.ylabel('Aggregated OSRM Time')
plt.plot([0, df_agg_trip[['segment_osrm_time', 'osrm_time']].max().max()], [0, df_agg_trip[['segment_osrm_time', 'osrm_time']].max().max(
plt.grid(True)
plt.savefig('osrm_time_vs_segment_osrm_time.png')
plt.close()
print("Saved scatter plot: osrm_time_vs_segment_osrm_time.png")

try:
    stat, p_value = stats.wilcoxon(df_agg_trip['osrm_time'], df_agg_trip['segment_osrm_time'])
    print(f"\nWilcoxon test between aggregated osrm_time and segment_osrm_time:")
    print(f"Statistic: {stat}, p-value: {p_value}")
    if p_value < 0.05:
        print("Significant difference detected (p < 0.05).")
    else:
        print("No significant difference detected (p >= 0.05).")
except ValueError as e:
    print(f"\nCould not perform Wilcoxon test: {e}")
```

```
Comparing aggregated 'osrm_time' vs 'segment_osrm_time'...
Saved scatter plot: osrm_time_vs_segment_osrm_time.png

Wilcoxon test between aggregated osrm_time and segment_osrm_time:
Statistic: 21161.5, p-value: 0.0
Significant difference detected (p < 0.05).
```

```
# --- Outlier Detection and Treatment ---

print("\nHandling Outliers...")

numerical_cols = df_agg_trip.select_dtypes(include=np.number).columns.tolist()
# Remove identifier/categorical-like numeric columns if any (e.g., year, month, day)
cols_to_exclude_outliers = ['trip_creation_year', 'trip_creation_month', 'trip_creation_day', 'trip_creation_hour', 'trip_creation_week(
numerical_cols = [col for col in numerical_cols if col not in cols_to_exclude_outliers]

print(f"Numerical columns for outlier check: {numerical_cols}")

# Visual Analysis (Boxplots)
plt.figure(figsize=(15, len(numerical_cols) * 2))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols), 1, i + 1)
    sns.boxplot(x=df_agg_trip[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.savefig('boxplots_before_outlier_treatment.png')
plt.close()
print("Saved boxplots: boxplots_before_outlier_treatment.png")

# Handle outliers using IQR method
df_cleaned = df_agg_trip.copy()
outliers_info = {}

for col in numerical_cols:
    Q1 = df_cleaned[col].quantile(0.25)
    Q3 = df_cleaned[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    original_count = df_cleaned.shape[0]
    # Cap the outliers instead of removing
    df_cleaned[col] = np.where(df_cleaned[col] < lower_bound, lower_bound, df_cleaned[col])
    df_cleaned[col] = np.where(df_cleaned[col] > upper_bound, upper_bound, df_cleaned[col])

    outliers_count = original_count - df_cleaned[(df_cleaned[col] >= lower_bound) & (df_cleaned[col] <= upper_bound)].shape[0] # Re-che(
    outliers_info[col] = {'lower': lower_bound, 'upper': upper_bound, 'count_capped': outliers_count}
    print(f"Capped outliers in '{col}' using IQR bounds [{lower_bound:.2f}, {upper_bound:.2f}]. Approx Capped: {outliers_count}")


print("\nOutlier capping summary:")
# print(outliers_info) # Can be verbose

# Visual Analysis After Capping
plt.figure(figsize=(15, len(numerical_cols) * 2))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols), 1, i + 1)
    sns.boxplot(x=df_cleaned[col])
    plt.title(f'Boxplot of {col} (After Capping)')
plt.tight_layout()
plt.savefig('boxplots_after_outlier_treatment.png')
plt.close()
print("Saved boxplots after capping: boxplots_after_outlier_treatment.png")
```

```
Handling Outliers...
Numerical columns for outlier check: ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_
Saved boxplots: boxplots_before_outlier_treatment.png
Capped outliers in 'start_scan_to_end_scan' using IQR bounds [-3219.00, 6453.00]. Approx Capped: 0
Capped outliers in 'actual_distance_to_destination' using IQR bounds [-10.14, 42.12]. Approx Capped: 0
Capped outliers in 'actual_time' using IQR bounds [-1239.50, 2444.50]. Approx Capped: 0
Capped outliers in 'osrm_time' using IQR bounds [-619.00, 1197.00]. Approx Capped: 0
Capped outliers in 'osrm_distance' using IQR bounds [-747.17, 1420.59]. Approx Capped: 0
Capped outliers in 'segment_actual_time' using IQR bounds [-385.50, 818.50]. Approx Capped: 0
Capped outliers in 'segment_osrm_time' using IQR bounds [-200.00, 416.00]. Approx Capped: 0
Capped outliers in 'segment_osrm_distance' using IQR bounds [-246.57, 498.02]. Approx Capped: 0
Capped outliers in 'od_time_diff_hours' using IQR bounds [-10.53, 24.28]. Approx Capped: 0
Capped outliers in 'scan_od_diff' using IQR bounds [-6431.30, 3210.60]. Approx Capped: 0

Outlier capping summary:
Saved boxplots after capping: boxplots_after_outlier_treatment.png
```

```
# --- Categorical Variable Handling ---

print("\nHandling Categorical Variables...")

categorical_cols = df_cleaned.select_dtypes(include=['object', 'category']).columns.tolist()
# Also include state codes which were extracted
categorical_cols.extend(['source_state_code', 'destination_state_code'])
# Remove high cardinality columns like names/IDs if they are still object type
cols_to_exclude_encoding = ['trip_uuid', 'route_schedule_uuid', 'source_name', 'destination_name', 'source_city', 'destination_city']
categorical_cols = [col for col in categorical_cols if col not in cols_to_exclude_encoding and col in df_cleaned.columns]

print(f"Categorical columns for encoding: {categorical_cols}")

# One-Hot Encoding
df_encoded = pd.get_dummies(df_cleaned, columns=categorical_cols, drop_first=True, dummy_na=False) # drop_first to avoid multicollinear

print(f"Shape after One-Hot Encoding: {df_encoded.shape}")
print("Columns after encoding (sample):", df_encoded.columns[:20].tolist(), "...") # Show some new columns
```

```
Handling Categorical Variables...
Categorical columns for encoding: ['data', 'route_type', 'source_center', 'destination_center', 'destination_state_code', 'source_st
Shape after One-Hot Encoding: (14817, 1924)
Columns after encoding (sample): ['trip_uuid', 'route_schedule_uuid', 'trip_creation_time', 'source_name', 'destination_name', 'od_s
```

```
# --- Normalization / Standardization ---

print("\nNormalizing/Standardizing Numerical Features...")

# Select numerical columns again from the encoded dataframe
numerical_cols_encoded = df_encoded.select_dtypes(include=np.number).columns.tolist()
# Exclude identifiers and previously excluded cols
ids_and_time_parts = ['trip_creation_year', 'trip_creation_month', 'trip_creation_day', 'trip_creation_hour', 'trip_creation_weekday'] #
numerical_cols_to_scale = [col for col in numerical_cols_encoded if col not in ids_and_time_parts and col in numerical_cols] # Use origi

print(f"Numerical columns to scale: {numerical_cols_to_scale}")

# Using StandardScaler
scaler = StandardScaler()
df_scaled = df_encoded.copy()
df_scaled[numerical_cols_to_scale] = scaler.fit_transform(df_scaled[numerical_cols_to_scale])

print("Applied StandardScaler to numerical features.")
print("\nScaled Data Sample (first 5 rows, selected columns):")
print(df_scaled[numerical_cols_to_scale].head())
```

```
Normalizing/Standardizing Numerical Features...
Numerical columns to scale: ['start_scan_to_end_scan', 'actual_distance_to_destination', 'actual_time', 'osrm_time', 'osrm_distance
Applied StandardScaler to numerical features.

Scaled Data Sample (first 5 rows, selected columns):
   start_scan_to_end_scan  actual_distance_to_destination  actual_time  \
0                2.054511                        1.031426     2.048605
1               -0.501487                       -0.802293    -0.422573
2                2.054511                        1.610061     2.048605
3               -0.826804                       -0.587269    -0.805542
4               -0.188150                        0.965086    -0.232900

   osrm_time  osrm_distance  segment_actual_time  segment_osrm_time  \
0   2.100171       2.063424             2.149256           2.256743
1  -0.363660      -0.309542            -0.465311          -0.475861
2   2.100171       2.063424             2.149256           2.256743
3  -0.827968      -0.799703            -0.781760          -0.857336
4  -0.371149      -0.316013             0.302658          -0.086601

   segment_osrm_distance  od_time_diff_hours  scan_od_diff
0               2.262408            2.258645     -2.054434
1              -0.404369           -0.673529      0.500689
2               2.262408            2.258645     -2.054434
3              -0.818804           -0.860091      0.826368
4              -0.000954            0.560922      0.190536
```

```
# --- Final Data ---
print("\nFinal processed data shape:", df_scaled.shape)
# Save the processed data
df_scaled.to_csv('delhivery_data_processed.csv', index=False)
print("Saved processed data to 'delhivery_data_processed.csv'")
```

```
Final processed data shape: (14817, 1924)
```

```
Saved processed data to 'delhivery_data_processed.csv'
```

```
# --- Basic Business Insights  ---
print("\n--- Basic Business Insights ---")

# 1. Most frequent routes (Source State -> Destination State)
df_agg_trip['route'] = df_agg_trip['source_state_code'] + ' -> ' + df_agg_trip['destination_state_code']
top_routes = df_agg_trip['route'].value_counts().head(10)
print("\nTop 10 Routes (Source State -> Destination State):")
print(top_routes)

# 2. Busiest Corridors (Source City -> Destination City) - High Cardinality Warning
df_agg_trip['corridor'] = df_agg_trip['source_city'] + ' -> ' + df_agg_trip['destination_city']
top_corridors = df_agg_trip['corridor'].value_counts().head(10)
print("\nTop 10 Corridors (Source City -> Destination City):")
print(top_corridors)

# 3. Average time/distance for top routes
print("\nAverage Metrics for Top 5 Routes:")
for route in top_routes.head(5).index:
    route_data = df_agg_trip[df_agg_trip['route'] == route]
    avg_actual_time = route_data['actual_time'].mean()
    avg_osrm_time = route_data['osrm_time'].mean()
    avg_osrm_dist = route_data['osrm_distance'].mean()
    avg_od_diff = route_data['od_time_diff_hours'].mean()
    print(f"\nRoute: {route}")
    print(f"  Avg Actual Time: {avg_actual_time:.2f}")
    print(f"  Avg OSRM Time: {avg_osrm_time:.2f}")
    print(f"  Avg OSRM Distance: {avg_osrm_dist:.2f}")
    print(f"  Avg Trip Duration (OD): {avg_od_diff:.2f} hours")

# 4. Distribution of Route Types
print("\nDistribution of Route Types:")
print(df_agg_trip['route_type'].value_counts(normalize=True) * 100)


print("\n--- End of Analysis ---")
```

```
Gujarat -> Unknown          120
Maharashtra -> Unknown      119
Unknown -> Gujarat           99
Unknown -> Rajasthan         86
Jharkhand -> Unknown         71
Rajasthan -> Unknown         68
Punjab -> Punjab             67
Unknown -> Haryana           67
Name: count, dtype: int64

Top 10 Corridors (Source City -> Destination City):
corridor
Chandigarh_Mehmdpur_H (Punjab) -> Chandigarh_Mehmdpur_H (Punjab)          175
Bangalore_Nelmngla_H (Karnataka) -> Bengaluru_KGAirprt_HB (Karnataka)     151
Muzaffrpur_Bbganj_I (Bihar) -> Muzaffrpur_Bbganj_I (Bihar)                130
Bengaluru_Bomsndra_HB (Karnataka) -> Bengaluru_KGAirprt_HB (Karnataka)    121
Bhiwandi_Mankoli_HB (Maharashtra) -> Bhiwandi_Mankoli_HB (Maharashtra)    113
Bengaluru_KGAirprt_HB (Karnataka) -> Bangalore_Nelmngla_H (Karnataka)     108
Ahmedabad_East_H_1 (Gujarat) -> Ahmedabad_East_H_1 (Gujarat)              107
Bhiwandi_Mankoli_HB (Maharashtra) -> Mumbai Hub (Maharashtra)             105
Mumbai_Chndivli_PC (Maharashtra) -> Bhiwandi_Mankoli_HB (Maharashtra)      99
Bangalore_Nelmngla_H (Karnataka) -> Bengaluru_Bomsndra_HB (Karnataka)      97
Name: count, dtype: int64

Average Metrics for Top 5 Routes:

Route: Unknown -> Unknown
  Avg Actual Time: 3990.09
  Avg OSRM Time: 2075.11
  Avg OSRM Distance: 2757.19
  Avg Trip Duration (OD): 9.08 hours

Route: Tamil Nadu -> Unknown
  Avg Actual Time: 124.24
  Avg OSRM Time: 57.11
  Avg OSRM Distance: 65.09
  Avg Trip Duration (OD): 1.68 hours

Route: Gujarat -> Unknown
  Avg Actual Time: 8323.50
```

```
        Avg OSRM Distance: 1033.66
        Avg Trip Duration (OD): 5.01 hours

    Route: Unknown -> Gujarat
        Avg Actual Time: 3358.87
        Avg OSRM Time: 1830.51
        Avg OSRM Distance: 2377.93
        Avg Trip Duration (OD): 8.20 hours

    Distribution of Route Types:
    route_type
    Carting     60.120122
```

Start coding or generate with AI.