About Delhivery

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

In [1]:
```
# importing dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
# load data set
df = pd.read_csv('/content/drive/MyDrive/[01]DataScience/Data sets/Python files/CSV
df.head()
```

Out[3]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121

5 rows × 24 columns

1. Basic data cleaning and Exploration

In [4]:
```
# shape of data
rows, cols = df.shape
print('Total count of rows :', rows)
print('Total count of columns :', cols)
```

```
Total count of rows : 144867
Total count of columns : 24
```

In [5]:
```
# checking columns
df.columns
```

Out[5]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

In [6]:
```
# checking entire data set with info() function
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
```

In [7]:
```
# checking number of unique values in each columns
df.nunique()
```

Out[7]:

	0
data	2
trip_creation_time	14817
route_schedule_uuid	1504
route_type	2
trip_uuid	14817
source_center	1508
source_name	1498
destination_center	1481
destination_name	1468
od_start_time	26369
od_end_time	26369
start_scan_to_end_scan	1915
is_cutoff	2
cutoff_factor	501
cutoff_timestamp	93180
actual_distance_to_destination	144515
actual_time	3182
osrm_time	1531
osrm_distance	138046
factor	45641
segment_actual_time	747
segment_osrm_time	214
segment_osrm_distance	113799
segment_factor	5675

dtype: int64

In [8]:
```
# let's drop some unknown columns
df.drop(columns=['is_cutoff','route_schedule_uuid', 'cutoff_factor','cutoff_timesta
df.head(3)
```

Out[8]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_ce
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121

3 rows × 24 columns

In [9]:
```
#converting to catogorical value for data and route type
df[["data","route_type"]] = df[["data","route_type"]].astype("category")

#converting into datetime
df["trip_creation_time"] = pd.to_datetime(df["trip_creation_time"])
df["od_start_time"] = pd.to_datetime(df["od_start_time"])
df["od_end_time"] = pd.to_datetime(df["od_end_time"])
```

In [10]:
```
# statistical summary
df.describe()
```

Out[10]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	cutoff_fa
count	144867	144867	144867	144867.000000	144867.00
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	961.262986	232.92
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.00
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	161.000000	22.00
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	449.000000	66.00
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	1634.000000	286.00
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.00
std	NaN	NaN	NaN	1037.012769	344.75

In [11]:
```
df['data'].unique() # two different sets of data present
```

Out[11]:
```
['training', 'test']
Categories (2, object): ['test', 'training']
```

In [12]:
```
# let's find out percentage of training/test data in df
df['data'].value_counts(normalize=True)
```

Out[12]:

data	proportion
training	0.723823
test	0.276177

dtype: float64

since data team wants to build a forecasting model on the data, data has been divided into 70 % -> training and 30 % -> testing data

In [13]:
```
duration_data = df['od_end_time'].max() - df['trip_creation_time'].min()
print('Total duration of data is :', duration_data)
```

```
Total duration of data is : 26 days 03:00:07.817738
```

In [14]:
```
# lets handle null values
df.isna().sum()
```

Out[14]:

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
is_cutoff	0
cutoff_factor	0
cutoff_timestamp	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
factor	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0
segment_factor	0

dtype: int64

In [15]:
```
# percentage of null values
null_percentage = df.isna().sum()/df.shape[0]*100
null_percentage
```

Out[15]:

	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

In [16]:
```
# two columns have null values -> sourcename, destinationname
# before dropping null values, let us find out missing values
source_centers = df.loc[df['source_name'].isnull(), 'source_center'].unique()
destination_centers = df.loc[df['destination_name'].isnull(), 'destination_center']
```

In [17]:
```
print('source centers missing values')
print(source_centers)
print('destination centers missing values')
print(destination_centers)
```

```
source centers missing values
['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND509103AAC' 'IND126116AAA' 'IND331022A1B'
 'IND505326AAB' 'IND852118A1B']
destination centers missing values
['IND342902A1B' 'IND577116AAA' 'IND282002AAD' 'IND465333A1B'
 'IND841301AAC' 'IND505326AAB' 'IND852118A1B' 'IND126116AAA'
 'IND509103AAC' 'IND221005A1A' 'IND250002AAC' 'IND331001A1C'
 'IND122015AAC']
```

In [18]:
```
# drop missing data
drop_missing = df.dropna(axis=0, how='any', inplace=True)
```

In [19]:
```
# after dropped missing, checking null count
df.isna().sum()
```

Out[19]:

	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

In [20]:
```
# Summary statistics
df.describe()
```

Out[20]:

	trip_creation_time	od_start_time	od_end_time	start_scan_to_end_scan	cutoff_fa
count	144316	144316	144316	144316.000000	144316.00
mean	2018-09-22 13:05:09.454117120	2018-09-22 17:32:42.435769344	2018-09-23 09:36:54.057172224	963.697698	233.56
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	20.000000	9.00
25%	2018-09-17 02:46:11.004421120	2018-09-17 07:37:35.014584832	2018-09-18 01:29:56.978912	161.000000	22.00
50%	2018-09-22 03:36:19.186585088	2018-09-22 07:35:23.038482944	2018-09-23 02:49:00.936600064	451.000000	66.00
75%	2018-09-27 17:53:19.027942912	2018-09-27 22:01:30.861209088	2018-09-28 12:13:41.675546112	1645.000000	286.00
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	1927.00
std	NaN	NaN	NaN	1038.082976	345.24

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

In [21]:
```
# grouping data on trip_uuid, socurce center and destination center
merged_df = df.groupby(by=['trip_uuid', 'source_center', 'destination_center'],
                       axis=0, as_index=False).agg({'data':'first',
                                                    'route_type' : 'first',
                                                    'trip_creation_time' : 'first',
                                                    'source_name' : 'first',
                                                    'destination_name' : 'last',
                                                    'od_start_time' : 'first',
                                                    'od_end_time' : 'last',
                                                    'start_scan_to_end_scan' : 'las
                                                    'actual_distance_to_destinatior
                                                    'actual_time' : 'last',
                                                    'osrm_time' : 'last',
                                                    'osrm_distance' : 'last',
                                                    'segment_actual_time' : 'sum',
                                                    'segment_osrm_time' : 'sum',
                                                    'segment_osrm_distance' : 'sum'
```

```
<ipython-input-21-979ef486d6d8>:2: FutureWarning: The 'axis' keyword in DataFrame.
groupby is deprecated and will be removed in a future version.
  merged_df = df.groupby(by=['trip_uuid', 'source_center', 'destination_center'],
```

In [22]:
```
merged_df.head()
```

Out[22]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	FTL	2018-09-12 00:00:16.535741
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	Carting	2018-09-12 00:00:22.886430
3	trip-153671042288605164	IND572101AAA	IND561203AAB	training	Carting	2018-09-12 00:00:22.886430
4	trip-153671043369099517	IND000000ACB	IND160002AAC	training	FTL	2018-09-12 00:00:33.691250

In [23]:
```
merged_df['od_total_time(mins)'] = (merged_df['od_end_time'] - merged_df['od_start_
merged_df.head(3)
```

Out[23]:

	trip_uuid	source_center	destination_center	data	route_type	trip_creation_time
0	trip-153671041653548748	IND209304AAA	IND000000ACB	training	FTL	2018-09-12 00:00:16.535741
1	trip-153671041653548748	IND462022AAA	IND209304AAA	training	FTL	2018-09-12 00:00:16.535741
2	trip-153671042288605164	IND561203AAB	IND562101AAA	training	Carting	2018-09-12 00:00:22.886430

In [24]:
```
# grouping the data on trip unique ID
new_df = merged_df.groupby(by = ["trip_uuid"],
                           axis = 0,
                           as_index = False).agg({'data' : 'first',
                                                  'route_type' : 'first',
                                                  'trip_creation_time' : 'first',
                                                  'source_name' : 'first',
                                                  'destination_name' : 'last',
                                                  'od_total_time(mins)' : 'last',
                                                  'start_scan_to_end_scan' : 'last',
                                                  'actual_distance_to_destination' : 'l
                                                  'actual_time' : 'last',
                                                  'osrm_time' : 'last',
                                                  'osrm_distance' : 'last',
                                                  'segment_actual_time' : 'sum',
                                                  'segment_osrm_time' : 'sum',
                                                  'segment_osrm_distance' : 'sum'})
```

```
<ipython-input-24-873bf041ab84>:2: FutureWarning: The 'axis' keyword in DataFrame.
groupby is deprecated and will be removed in a future version.
  new_df = merged_df.groupby(by = ["trip_uuid"],
```

In [25]:
```
new_df.head()
```

Out[25]:

	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (I
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgao
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mum
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_

In [26]:
```
# New feature extraction from source
# source state # destination state # source city # destination city
```

In [27]:
```
import re
```

In [28]:
```
# Define the function to extract state
def extract_state(location):
    # Regex pattern to extract text inside parentheses
    match = re.search(r"\((.*?)\)", location)
    if match:
        return match.group(1)
    return None
```

In [29]:
```
new_df['source_state'] = new_df['source_name'].apply(extract_state)
new_df['source_state'].unique()
```

Out[29]:
```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
       'Assam', 'Madhya Pradesh', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Goa', 'Jharkhand', 'Pondicherry',
       'Orissa', 'Uttarakhand', 'Himachal Pradesh', 'Kerala',
       'Arunachal Pradesh', 'Bihar', 'Chhattisgarh',
       'Dadra and Nagar Haveli', 'Jammu & Kashmir', 'Mizoram', 'Nagaland'],
      dtype=object)
```

In [30]:
```
new_df["destination_state"] = new_df["destination_name"].apply(extract_state)
new_df["destination_state"].unique()
```

Out[30]:
```
array(['Uttar Pradesh', 'Karnataka', 'Haryana', 'Maharashtra',
       'Tamil Nadu', 'Gujarat', 'Delhi', 'Telangana', 'Rajasthan',
       'Madhya Pradesh', 'Assam', 'West Bengal', 'Andhra Pradesh',
       'Punjab', 'Chandigarh', 'Dadra and Nagar Haveli', 'Orissa',
       'Bihar', 'Jharkhand', 'Goa', 'Uttarakhand', 'Himachal Pradesh',
       'Kerala', 'Arunachal Pradesh', 'Mizoram', 'Chhattisgarh',
       'Jammu & Kashmir', 'Nagaland', 'Meghalaya', 'Tripura',
       'Daman & Diu'], dtype=object)
```

In [31]:
```
new_df.head(2)
```

Out[31]:

	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (I
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu

In [32]:
```
cities = new_df['source_name'].str.split().str[0].str.split('_').str[0].unique()
```

In [33]:
```
cities[:5]
```

Out[33]:
```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Bellary'],
      dtype=object)
```

In [34]:
```
city = []
for i in cities:
  if len(i) > 3:
    pass
  else:
    city.append(i)
```

In [35]:
```
city
```

Out[35]:
```
['HBR',
 'FBD',
 'CCU',
 'Goa',
 'MAA',
 'BOM',
 'Pen',
 'AMD',
 'BLR',
 'PNQ',
 'OK',
 'Del',
 'GZB',
 'Hyd',
 'Wai',
 'GGN',
 'Cjb',
 'Amd',
 'Blr',
 'Mau']
```

In [36]:
```
#function to convert cities with 3 letter short code name
def city(city):
  c = city.split()[0].split('_')[0]
  if 'CCU' in city:
    return 'Kolkata'
  elif 'MAA' in city.upper():
    return 'Chennai'
  elif ('HBR' in city.upper()) or ('BLR' in city.upper()):
    return 'Bengaluru'
  elif 'FBD' in city.upper():
    return 'Faridabad'
  elif 'BOM' in city.upper():
    return 'Mumbai'
  elif 'DEL' in city.upper():
    return 'Delhi'
  elif 'OK' in city.upper():
```

```
    return 'Delhi'
  elif 'GZB' in city.upper():
    return 'Ghaziabad'
  elif 'GGN' in city.upper():
    return 'Gurgaon'
  elif 'AMD' in city.upper():
    return 'Ahmedabad'
  elif 'CJB' in city.upper():
    return 'Coimbatore'
  elif 'HYD' in city.upper():
    return 'Hyderabad'
  elif "GOA" in city.upper():
    return "Goa"
  elif "PNQ" in city.upper():
    return "Pune"
  else:
    return c
```

In [37]:

```
#extraction of city from source and destination name
new_df["source_city"] = new_df["source_name"].apply(city)
new_df["destination_city"] = new_df["destination_name"].apply(city)
new_df["source_city"].unique()
```

Out[37]:
```
array(['Kanpur', 'Doddablpur', 'Gurgaon', 'Mumbai', 'Bellary', 'Chennai',
       'Bengaluru', 'Surat', 'Delhi', 'Pune', 'Faridabad', 'Shirala',
       'Hyderabad', 'Thirumalagiri', 'Gulbarga', 'Jaipur', 'Allahabad',
       'Guwahati', 'Narsinghpur', 'Shrirampur', 'Madakasira', 'Sonari',
       'Dindigul', 'Jalandhar', 'Chandigarh', 'Deoli', 'Pandharpur',
       'Kolkata', 'Bhandara', 'Kurnool', 'Bhiwandi', 'Bhatinda',
       'RoopNagar', 'Bantwal', 'Lalru', 'Kadi', 'Shahdol', 'Gangakher',
       'Durgapur', 'Vapi', 'Jamjodhpur', 'Jetpur', 'Mehsana', 'Jabalpur',
       'Junagadh', 'Gundlupet', 'Mysore', 'Goa', 'Bhopal', 'Sonipat',
       'Himmatnagar', 'Jamshedpur', 'Pondicherry', 'Anand', 'Udgir',
       'Nadiad', 'Villupuram', 'Purulia', 'Bhubaneshwar', 'Bamangola',
       'Tiruppattur', 'Kotdwara', 'Medak', 'Bangalore', 'Dhrangadhra',
       'Hospet', 'Ghumarwin', 'Agra', 'Sitapur', 'Bilimora', 'SultnBthry',
       'Lucknow', 'Vellore', 'Bhuj', 'Dinhata', 'Margherita', 'Boisar',
       'Vizag', 'Tezpur', 'Koduru', 'Tirupati', 'Pen', 'Ahmedabad',
       'Faizabad', 'Gandhinagar', 'Anantapur', 'Betul', 'Panskura',
       'Rasipurm', 'Sankari', 'Jorhat', 'Srikakulam', 'Dehradun',
       'Jassur', 'Sawantwadi', 'Shajapur', 'Ludhiana', 'GreaterThane',
       'Tirupur', 'Salem', 'Darjeeling', 'Tiruchi', 'Noida', 'Thiruvarur',
       'Ranchi', 'Guna', 'Raver', 'Jairampur', 'Chamoli', 'Pali',
       'Kamareddy', 'Gopiganj', 'Varanasi', 'Dharmapuri', 'Hubli',
       'Duddhi', 'Sasaram', 'Davangere', 'Panipat', 'Chittaurgarh',
       'Solapur', 'Pratapgarh', 'Vinukonda', 'Ongole', 'LowerParel',
       'Sagara', 'Tikamgarh', 'Ghaziabad', 'Chhapra', 'BiharSarif',
       'Pallakad', 'Kanakapura', 'Mangalore', 'Aurangabad', 'Barh',
       'Coimbatore', 'Bhadrak', 'Narnaul', 'Hisar', 'Bihta', 'Silchar',
       'Sillod', 'Nellore', 'Katwa', 'Thamarassery', 'Safidon',
       'Vijayawada', 'Machilipatnam', 'Nazirpur', 'Vikarabad',
       'Rampurhat', 'Visakhapatnam', 'Lalgola', 'Rampur', 'Kakinada',
       'Amalapuram', 'Muzaffrpur', 'Kalka', 'Buldhana', 'Karad',
       'JoguGadwal', 'Madhepura', 'Simrahi', 'Atmakur', 'Hassan',
       'Chikodi', 'Rohtak', 'Patiala', 'Ajmer', 'Channaraya', 'Naugchia',
       'Ambala', 'Korba', 'Pithorgarh', 'Deoghar', 'Alwar', 'Gorakhpur',
       'Bhatpara', 'Dumka', 'Bahadurgarh', 'Kanth', 'Nichlaul',
       'Warangal', 'Aonla', 'Dhar', 'Bagnan', 'Naraingarh', 'Kashipur',
       'Ratanpura', 'Gondia', 'Zahirabad', 'Samana', 'Bhadrachalam',
       'Baraut', 'Sikar', 'Jamnagar', 'Kakdwip', 'Gadarwara', 'Gwalior',
       'Akola', 'Kalluvathukal', 'Surendranagar', 'Buxar', 'Trivandrum',
       'Etawah', 'Bhagalpur', 'Vadodara', 'Chhata', 'Luxettipet',
       'Mancherial', 'Kottayam', 'Parakkdavu', 'Pthnmthitt', 'Dhule',
       'DehriSone', 'Brahmapuri', 'Ramagundam', 'Gomoh', 'Kollam',
       'Wardha', 'Barnala', 'Latur', 'Ghatampur', 'Upleta', 'Khammam',
       'Akbarpur', 'Bhanvad', 'Basti', 'Mussoorie', 'Kalpetta', 'Phalodi',
       'Guskhara', 'Mainaguri', 'Gosainganj', 'Bhusawal', 'Nalbari',
       'Talegaon', 'SrinagarUK', 'Shimoga', 'Bailhongal', 'Gonda',
       'Manapparai', 'Udaipur', 'Ghazipur', 'Guruvayoor', 'Chetpet',
       'Wai', 'Karkala', 'Patancheru', 'Kumbakonam', 'Rameswram',
       'Shirur', 'Degana', 'Pattukotai', 'Srisailam', 'Lalpet', 'Madurai',
       'Sathyamangalam', 'Usilampatti', 'Khurai', 'Nuzvid', 'Koppa',
       'Tiptur', 'Rajamundry', 'Haveri', 'Dumraon', 'Machhiwara',
       'NeemKaThana', 'Baheri', 'Dharapuram', 'Mohania', 'Bilaspur',
       'Naugarh', 'Patran', 'Mahbubabad', 'Tirunelveli', 'Bhavnagar',
       'Dhanbad', 'Mahadevpur', 'Puttur', 'Jammikunta', 'Narsingpur',
       'Jagtial', 'Manthani', 'Aligarh', 'Rudrapur', 'Manamelkudi',
       'Malegaon', 'Sindhanur', 'Murbad', 'Medchal', 'Kanker', 'Unjha',
       'Khambhat', 'Malda', 'Haridwar', 'Anjar', 'Pathankot',
       'Bhubaneswar', 'Sholinghur', 'Jhansi', 'Silvassa', 'Balasore',
       'Nagaur', 'Bhilwara', 'Ghanpur', 'Achrol', 'Hazaribag', 'Dharwad',
       'Chhatarpur', 'Arrah', 'Udupi', 'Gooty', 'Bareilly', 'Kallachi',
       'Devarakonda', 'Mahabubnagar', 'Hailakandi', 'Jeypore',
       'Wanaparthy', 'Ramnthpurm', 'Sitamari', 'Makrana',
       'Sankaramangalam', 'Ratnagiri', 'Meerut', 'Chikhli', 'Cumbum',
       'Sakleshpur', 'Anthiyour', 'Khanna', 'Bharatpur', 'Bina',
       'Lonavala', 'AurngbadBR', 'Ambah', 'Amreli', 'Dadri',
```

```
'SikandraRao', 'Kaman', 'Pukhrayan', 'Raichur', 'Raipur',
'Bellmpalli', 'Chinnur', 'Bankura', 'Bareli', 'Panagarh',
'Chhindwara', 'Mananthavady', 'Kharagpur', 'JognderNgr',
'Phagwara', 'Srivijaynagar', 'Thoppur', 'Bongaigaon',
'Rajgurunagar', 'Deoband', 'Chopan', 'Chomu', 'Satara', 'Rewari',
'Mainpuri', 'Nandigama', 'Kolhapur', 'Tirurangadi', 'Vadakara',
'Mariani', 'Baharampur', 'Almora', 'Sonepur', 'Karnal', 'Bettiah',
'YamunaNagar', 'Godda', 'Ratlam', 'Sagar', 'Kaptanganj', 'Katni',
'Umaria', 'Sambhal', 'Sitarganj', 'Vaijiapur', 'Akhnoor', 'Ashta',
'Aluva', 'ChrkhiDdri', 'Kattappana', 'Dharamshala', 'Dausa',
'Katihar', 'Shirpur', 'Bangarapet', 'Dwarka', 'Bagepalli',
'Khurja', 'Haldwani', 'Asangaon', 'Moodbidri', 'Deesa',
'Kodaikanal', 'Bhabhar', 'Khedbrahma', 'Kodinar', 'RaisingNgr',
'Mejia', 'Vidisha', 'Jammu', 'Malvan', 'Roha', 'Hoskote', 'Tezu',
'Hooghly', 'Mau', 'Sujangarh', 'Gohpur', 'Peterbar', 'Thrissur',
'Rajgir', 'Polur', 'Ankola', 'Kanhangad', 'Chalakudy', 'Midnapore',
'Mungeli', 'Palampur', 'Mungaoli', 'SirhindFatehgarh', 'Jangipur',
'DalsinghSarai', 'Bewar', 'Pakur', 'Jasai', 'Kankavali', 'Hapur',
'Nanded', 'Palani', 'Palanpur', 'Narsapur', 'Dalkhola', 'Purnia',
'Airport', 'Kalpakkam', 'MughalSarai', 'Dohrighat', 'Manthuka',
'Bishwanath', 'Tulsipur', 'Aizawl', 'Tirur', 'Cochin', 'Uchila',
'Shevgaon', 'Athani', 'Amravati', 'Nilambur', 'Karimganj',
'Shamli', 'HanumanJNC', 'Bikramgang', 'Fatepur', 'Gangarampr',
'Itahar', 'Lakhnadon', 'Manikchak', 'Sihora', 'Jamtara', 'Giridih',
'Alappuzha', 'Bethamangala', 'Rajkot', 'Gola', 'Majalgaon',
'Hanumangarh', 'Kapurthala', 'Barmer', 'Tamluk', 'Palakonda',
'Mahad', 'Chamba', 'Krishnagiri', 'Tirchngode', 'Dholpur',
'Kabuganj', 'Bhadohi', 'Madnapalle', 'Kundapura', 'Irinjlkuda',
'Chapra', 'Lalitpur', 'Murshidabad', 'Bijapur', 'Beed', 'Madhupur',
'Hajipur', 'Khurdha', 'Wankaner', 'Hindupur', 'Bulndshahr',
'Aland', 'BariSadri', 'Husnabad', 'Bhuvanagiri', 'Islampure',
'Manjhaul', 'Bikaner', 'Siwan', 'Rupnarayanpur', 'Plassey',
'Mylduthuri', 'Modinagar', 'Nowda', 'Theni', 'Sagardighi',
'PaontSahib', 'Kaliyaganj', 'Taranagar', 'Jath', 'Chiplun',
'Suratgarh', 'Khonsa', 'Talala', 'Vadnagar', 'Arambag', 'Haldia',
'Sehore', 'Hura', 'Erode', 'Gadag', 'Shahganj', 'Balrampur',
'Mehkar', 'Kalyandurg', 'Berhampore', 'Dhaka', 'Bassi',
'Ukkadagatri', 'Sultana', 'Banka', 'Asifabad', 'Sivasagar',
'Jodhpur', 'Khatra', 'LakhimpurN', 'Kishangarh', 'Narktiganj',
'Aliganj', 'Bongaon', 'Nedumangad', 'Chandausi', 'Sujanpur',
'Karukachal', 'Kamarpukur', 'Keshiary', 'Firozabad', 'Melur',
'Thuraiyur', 'Nakashipara', 'Nasirabad', 'Nagamangala', 'Morgram',
'Triveninganj', 'Barhi', 'Bhatiya', 'Chotila', 'Falna',
'Kopargaon', 'Karimnagar', 'AnandprShb', 'Tinusukia', 'Modasa',
'Palasa', 'Dahanu', 'Gudur', 'Khanapur', 'Udala', 'Kathua', 'Moga',
'Ganga', 'Khed', 'Brajrajnagar', 'Sambalpur', 'Ghanashyampur',
'Seoni', 'Rajpura', 'Kadaba', 'Sangola', 'Jaleswar', 'Bhilad',
'Umreth', 'Pachore', 'Shegaon', 'Sundargarh', 'Sunam', 'Morbi',
'Fatehabad', 'Mundakayam', 'Vrindavan', 'Jalalabad', 'Angamaly',
'Asansol', 'Kadiri', 'Vadakkencherry', 'Balangir', 'Raxaul',
'Sirohi', 'Manmad', 'Halvad', 'Nagpur', 'Shoranur', 'Kaithal',
'Ranaghat', 'Sakri', 'Bangana', 'Kangayam', 'Palitana', 'Valsad',
'Dabhoi', 'Muktsar', 'Jhunir', 'Bheemunipatnam', 'Sedam',
'Virudhchlm', 'Gangavathi', 'Moradabad', 'Karanjia', 'Chimkurthy',
'Phusro', 'Jhajjar', 'Kozhikode', 'Kottarakkara', 'Shikohabad',
'Munger', 'Chhaygaon', 'Hathras', 'Kusumnchi', 'Pauri',
'Rishikesh', 'Khatauli', 'Baddi', 'Mandi', 'Merta', 'Kuthuparamba',
'Kaghaznagar', 'Auraiya', 'Giddarbaha', 'Paradip', 'Jharsuguda',
'Gobicheti', 'Arakkonam', 'Pilani', 'Simlapal', 'Baripada',
'Cuttack', 'Saharsa', 'Rajgarh', 'Durg', 'Balurghat', 'Dola',
'Pappadahandi', 'Sinnar', 'Barasat', 'Khanakul', 'Sendhwa',
'Ramgarh', 'BilaspurHP', 'Sidhmukh', 'Angul', 'SawaiMadhopur',
'Ambegaon', 'Thakurdwara', 'Malemruvathur', 'Bishnupur', 'Dhoraji',
```

```
'Meham', 'Uthangarai', 'Shadnagar', 'Bhiwani', 'Mahasamund',
'Mandla', 'Phulera', 'Sandur'], dtype=object)
```

In [38]:
```
#extraction of year,month, day, weekday
new_df['trip_creation_date'] = pd.to_datetime(new_df['trip_creation_time'].dt.date)
new_df['trip_creation_year'] = new_df['trip_creation_time'].dt.year
new_df['trip_creation_month'] = new_df['trip_creation_time'].dt.month
new_df['trip_completion_month'] = df['od_end_time'].dt.month
new_df['trip_creation_day'] = new_df['trip_creation_time'].dt.day
new_df['trip_creation_weekday'] = new_df['trip_creation_time'].dt.day_name()
new_df['trip_end_weekday'] = df['od_end_time'].dt.day_name()
new_df['trip_end_day'] = df['od_end_time'].dt.day
```

In [39]:
```
new_df.head(2)
```

Out[39]:

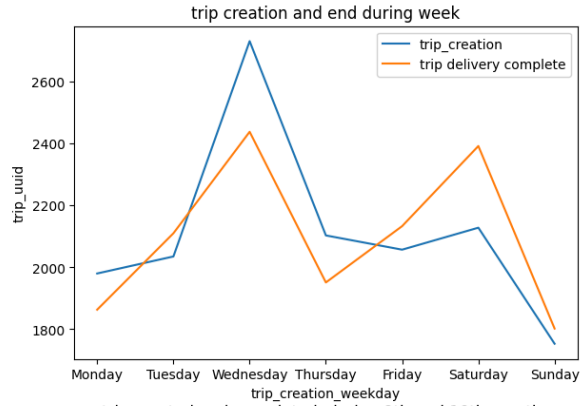
	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (U
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu

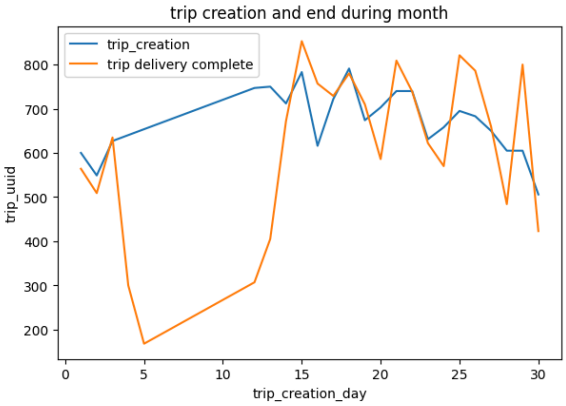
2 rows × 27 columns

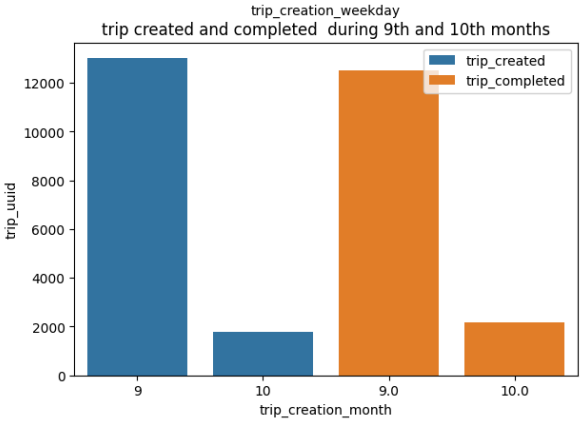
In [40]:
```
#Setting Up Weekday Categories
cats = [ 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda

# to check trip creation distribution through the weekdays
daily_trip = new_df.groupby(by = 'trip_creation_weekday')['trip_uuid'].count().reir

# to check trip end distribution through the weekdays
daily_trip_end = new_df.groupby(by = 'trip_end_weekday')['trip_uuid'].count().reinc

# to check trip creation distribution through the month time
monthly_trip = new_df.groupby(by = 'trip_creation_day')['trip_uuid'].count().reset_

# to check trip end distribution through the month time
monthly_trip_end = new_df.groupby(by = 'trip_end_day')['trip_uuid'].count().reset_i

# trips created during month
monthly_created_trip = new_df.groupby(by = 'trip_creation_month')['trip_uuid'].cour
monthly_completed_trip =  new_df.groupby(by = 'trip_completion_month')['trip_uuid']
```

In [41]:
```
# visualizations for trips
plt.figure(figsize = (15, 10))

plt.subplot(2,2,1)
sns.lineplot(data = daily_trip,x = daily_trip['trip_creation_weekday'],y = daily_tr
sns.lineplot(data = daily_trip_end,x = daily_trip_end['trip_end_weekday'],y = daily
plt.title("trip creation and end during week")

plt.subplot(2,2,2)
sns.lineplot(data = monthly_trip,x = monthly_trip['trip_creation_day'],y = monthly_
sns.lineplot(data = monthly_trip_end,x = monthly_trip_end['trip_end_day'],y = month
plt.title("trip creation and end during month")

plt.subplot(2,2,3)
sns.barplot(data = monthly_created_trip,x = monthly_created_trip["trip_creation_mor
sns.barplot(data = monthly_completed_trip, x =monthly_completed_trip["trip_completi
```

```
plt.title("trip created and completed  during 9th and 10th months")
plt.show()
```





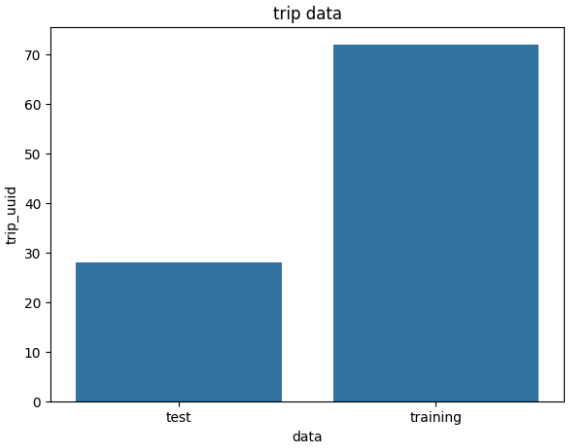


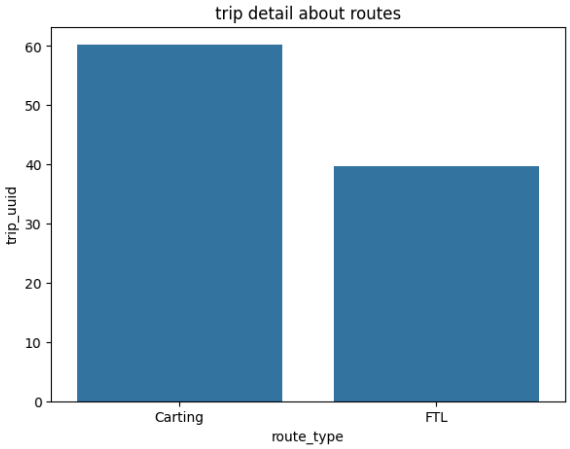
1. Highest number of trip cretion is being done on wednesday and highest number of trip completion is being done on wednesday and saturday
2. Highest number of trip cretion and trip complete is being done during mid month and start decresing after that. Very low trip completion during 2nd week of month
3. Highest number of trips are created and completed in 9th month

In [42]:
```
trip_route = new_df.groupby(by = 'route_type')['trip_uuid'].count().reset_index()
trip_data = new_df.groupby(by = "data")['trip_uuid'].count().reset_index()
```

```
<ipython-input-42-ccd221277853>:1: FutureWarning: The default of observed=False is
deprecated and will be changed to True in a future version of pandas. Pass observe
d=False to retain current behavior or observed=True to adopt the future default an
d silence this warning.
  trip_route = new_df.groupby(by = 'route_type')['trip_uuid'].count().reset_index
()
<ipython-input-42-ccd221277853>:2: FutureWarning: The default of observed=False is
deprecated and will be changed to True in a future version of pandas. Pass observe
d=False to retain current behavior or observed=True to adopt the future default an
d silence this warning.
  trip_data = new_df.groupby(by = "data")['trip_uuid'].count().reset_index()
```

In [43]:
```
plt.figure(figsize = (15, 5))
plt.subplot(1,2,1)
sns.barplot(data = trip_data,x = trip_data["data"],y = (trip_data['trip_uuid'] * 10
plt.title("trip data")

plt.subplot(1,2,2)
sns.barplot(data = trip_route,x = trip_route["route_type"],y = (trip_route['trip_uu
plt.title("trip detail about routes")
plt.show()
```





1. 70 % data is training data set

2. 60 % trips are made through carting route

In [44]:
```
source_state_trip = new_df.groupby(by = 'source_state')['trip_uuid'].count().reset_
destination_state_trip = new_df.groupby(by = 'destination_state')['trip_uuid'].cour
source_city_trip = new_df.groupby(by = 'source_city')['trip_uuid'].count().reset_ir
destination_city_trip = new_df.groupby(by = 'destination_city')['trip_uuid'].count(
```

In [45]:
```
#data extraction for city wise trip creation devilvery
source_destination_city_trip = pd.merge(source_city_trip, destination_city_trip, ho
source_destination_city_trip = source_destination_city_trip.rename({"source_city":
source_destination_city_trip = source_destination_city_trip[["city","source_trip",'

# Melt the dataframe to reshape it for Seaborn's barplot
city_source_destination_trip = source_destination_city_trip.melt(id_vars="city", va
                                                 var_name="trip_type", value_name="total_c
```

In [46]:
```
total_city_trip = city_source_destination_trip.groupby(by="city")["total_city_trips
total_city_trip = total_city_trip.sort_values(ascending = False)

total_city_trip["Bengaluru"] = total_city_trip["Bengaluru"] + total_city_trip["Bang
total_city_trip = total_city_trip.head(10).drop("Bangalore")
total_city_trip
```

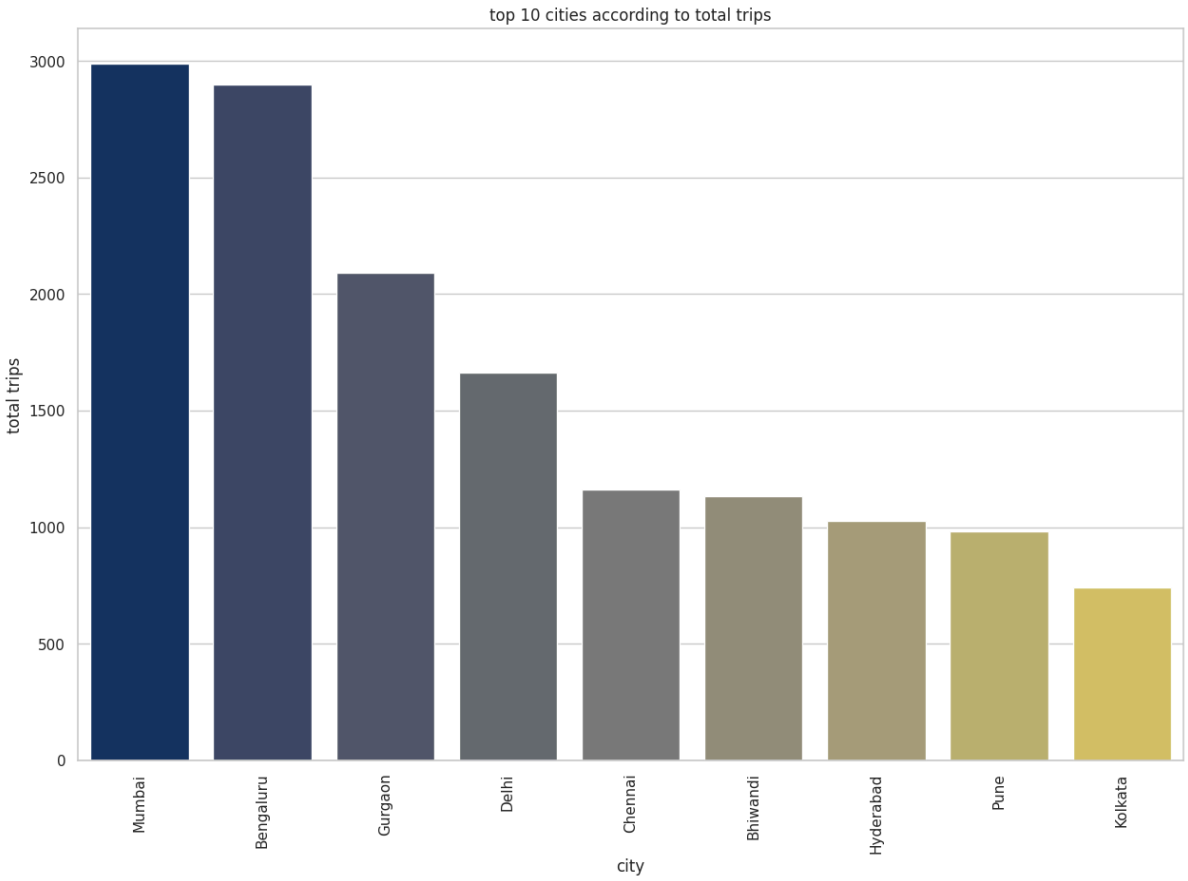
Out[46]:

city	total_city_trips
Mumbai	2990.0
Gurgaon	2090.0
Bengaluru	2899.0
Delhi	1663.0
Chennai	1163.0
Bhiwandi	1131.0
Hyderabad	1027.0
Pune	981.0
Kolkata	740.0

dtype: float64

In [47]:
```
#barplot for top 10 cities according to total trips
plt.figure(figsize = (15, 10))
sns.set(style="whitegrid")

sorted_values = total_city_trip.sort_values(ascending = False).index

# Create a bar chart with state on the x-axis and the count on the y-axis
barplot = sns.barplot(x="city", y="total_city_trips", data=total_city_trip.reset_ir

# Rotate the x-axis labels for better readability
plt.xticks(rotation=90)

# Add a title and labels for the axes
plt.title("top 10 cities according to total trips")
plt.xlabel("city")
plt.ylabel("total trips")


plt.show()
```

```
<ipython-input-47-5a293d5ef9cc>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(x="city", y="total_city_trips", data=total_city_trip.reset
_index(), order=sorted_values, palette="cividis")
```



Based on the above graph, Mumbai, Bengaluru and Gurgaon do the heighest number of trip creation and trip delivery

In [48]:
```
#data extraction for state wise trip creation devilvery
source_destination_state_trip = pd.merge(source_state_trip, destination_state_trip,
source_destination_state_trip = source_destination_state_trip.rename({"source_state
source_destination_state_trip = source_destination_state_trip[["state","source_trip

# Melt the dataframe to reshape it for Seaborn's barplot
state_source_destination_trip = source_destination_state_trip.melt(id_vars="state",
                                        var_name="trip_type", value_name="total_s
```

In [49]:
```
total_state_trip = state_source_destination_trip.groupby(by="state")["total_state_t
total_state_trip.reset_index()
```

Out[49]:

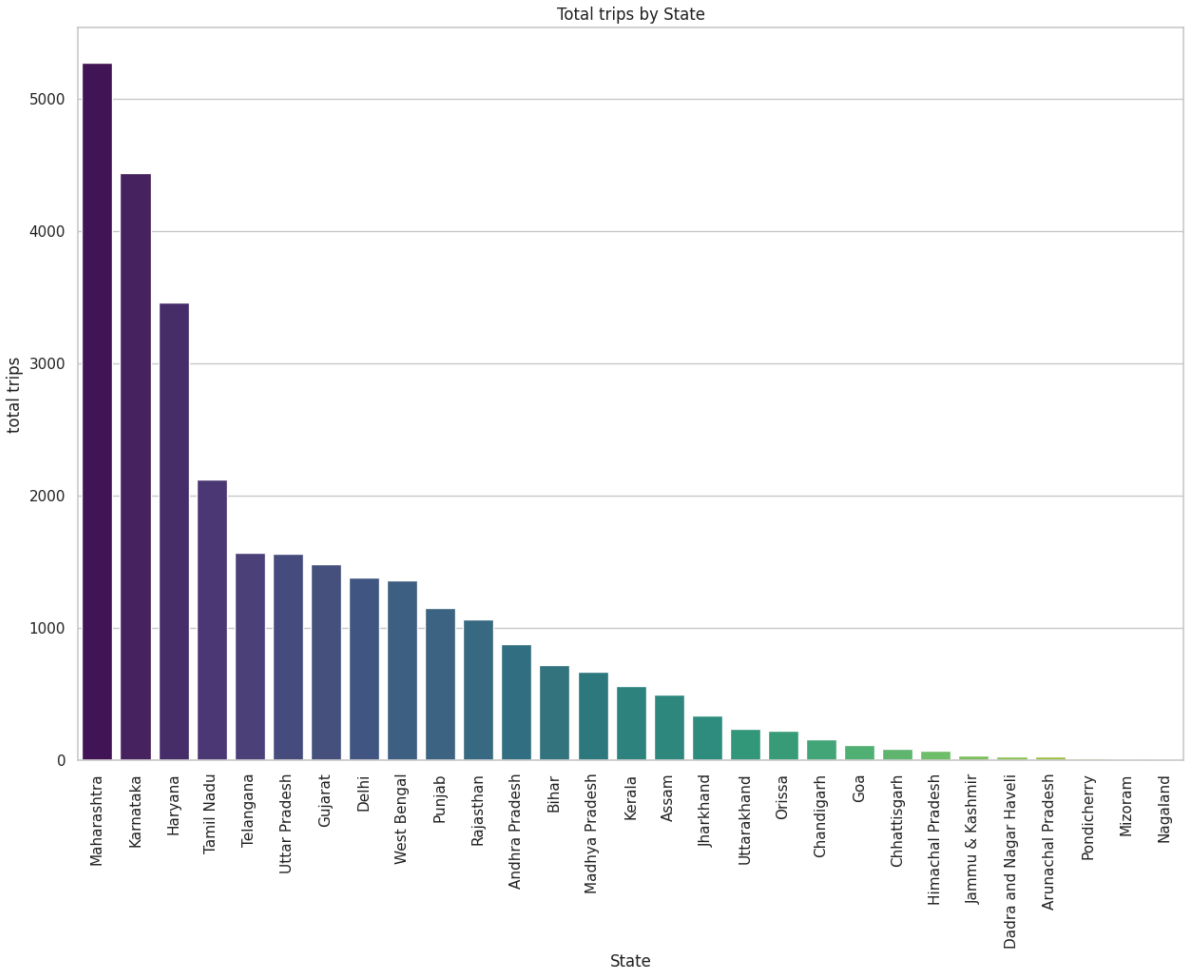
	state	total_state_trips
0	Andhra Pradesh	877.0
1	Arunachal Pradesh	29.0
2	Assam	500.0
3	Bihar	718.0
4	Chandigarh	158.0
5	Chhattisgarh	86.0
6	Dadra and Nagar Haveli	32.0
7	Delhi	1382.0
8	Goa	117.0
9	Gujarat	1484.0
10	Haryana	3463.0
11	Himachal Pradesh	76.0
12	Jammu & Kashmir	37.0
13	Jharkhand	341.0
14	Karnataka	4437.0
15	Kerala	559.0
16	Madhya Pradesh	668.0
17	Maharashtra	5275.0
18	Mizoram	10.0
19	Nagaland	6.0
20	Orissa	226.0
21	Pondicherry	12.0
22	Punjab	1153.0
23	Rajasthan	1064.0
24	Tamil Nadu	2123.0
25	Telangana	1568.0
26	Uttar Pradesh	1565.0
27	Uttarakhand	236.0
28	West Bengal	1362.0

In [50]:

```
#barplot for state wise total trips
plt.figure(figsize = (15, 10))
sns.set(style="whitegrid")

sorted_values = total_state_trip.sort_values(ascending = False).index

# Create a bar chart with state on the x-axis and the count on the y-axis
barplot = sns.barplot(x="state", y="total_state_trips", data=total_state_trip.reset

# Rotate the x-axis labels for better readability
```

```
plt.xticks(rotation=90)

# Add a title and labels for the axes
plt.title("Total trips by State")
plt.xlabel("State")
plt.ylabel("total trips")


plt.show()
```

```
<ipython-input-50-fac1c81c2838>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  barplot = sns.barplot(x="state", y="total_state_trips", data=total_state_trip.re
set_index(), order=sorted_values, palette="viridis")
```



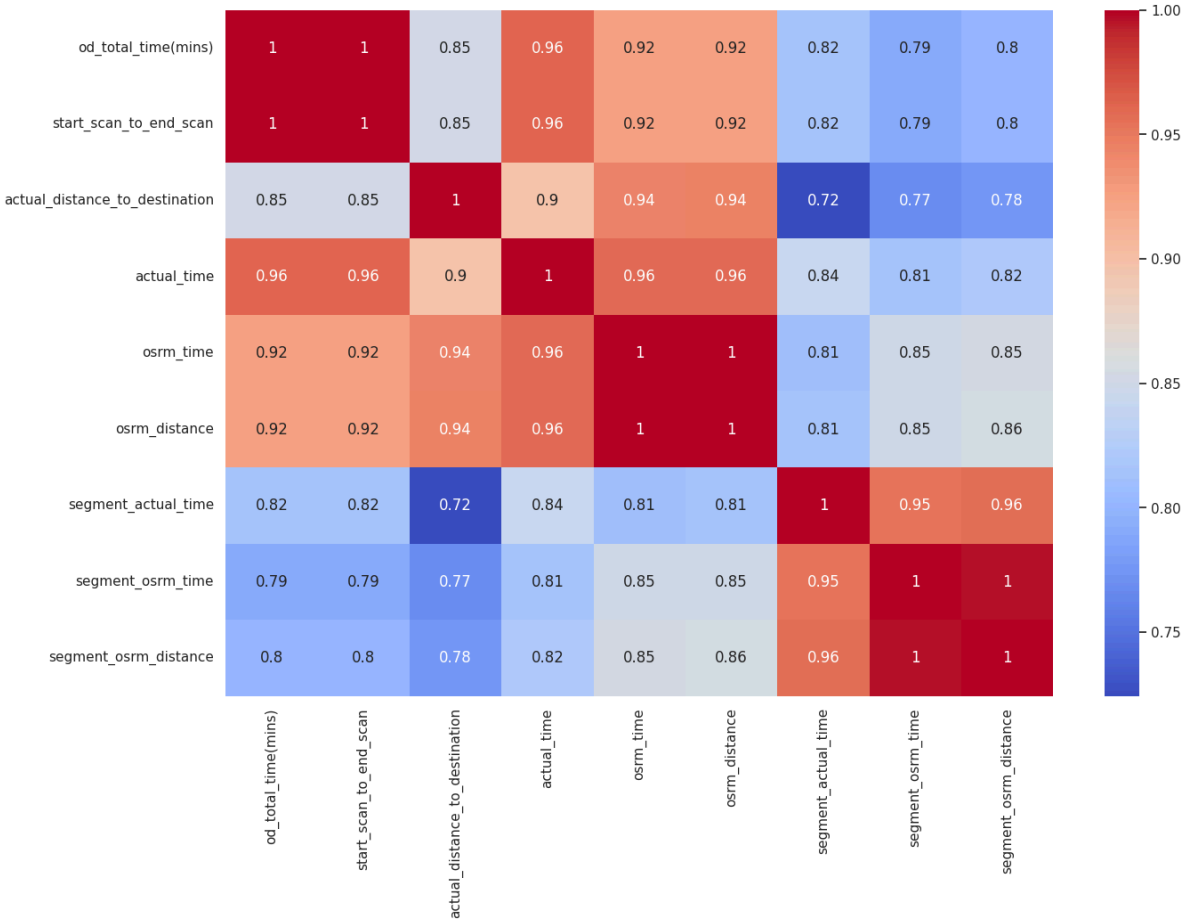
In [51]:
```
pairplot_columns = ['od_total_time(mins)', 'start_scan_to_end_scan','actual_distanc
                    'osrm_distance', 'segment_actual_time', 'segment_osrm_time','se
#sns.pairplot(data = new_df, vars = pairplot_columns, kind = 'reg',hue = 'route_typ
#plt.show()
```

In [52]:
```
df_corr = new_df[pairplot_columns].corr()
df_corr
```

Out[52]:

	od_total_time(mins)	start_scan_to_end_scan	actual_distance_to_destin
od_total_time(mins)	1.000000	1.000000	0.85
start_scan_to_end_scan	1.000000	1.000000	0.85
actual_distance_to_destination	0.852371	0.852376	1.00
actual_time	0.962198	0.962201	0.89
osrm_time	0.924495	0.924501	0.94
osrm_distance	0.924276	0.924282	0.94
segment_actual_time	0.815792	0.815794	0.72
segment_osrm_time	0.791690	0.791695	0.76
segment_osrm_distance	0.799721	0.799726	0.77

In [53]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, annot = True, cmap="coolwarm")
plt.show()
```



- All correlation values are between 0.7 to 1 hence all columns shows a correlation but columns with correlation value 1 shows a strong correlation

3. 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 original columns, if required

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

In [54]:
```
time_taken = new_df[["trip_uuid", "od_total_time(mins)","start_scan_to_end_scan"]]
time_taken
```

Out[54]:

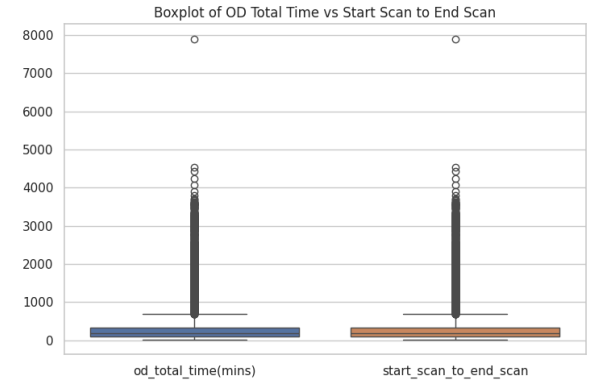
	trip_uuid	od_total_time(mins)	start_scan_to_end_scan
0	trip-153671041653548748	1000	999.0
1	trip-153671042288605164	123	122.0
2	trip-153671043369099517	3100	3099.0
3	trip-153671046011330457	100	100.0
4	trip-153671052974046625	81	80.0
...	...	...	...
14782	trip-153861095625827784	105	105.0
14783	trip-153861104386292051	61	60.0
14784	trip-153861106442901555	174	173.0
14785	trip-153861115439069069	44	44.0
14786	trip-153861118270144424	67	66.0

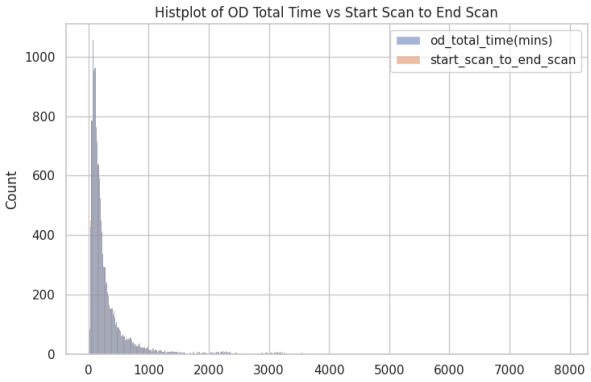
14787 rows × 3 columns

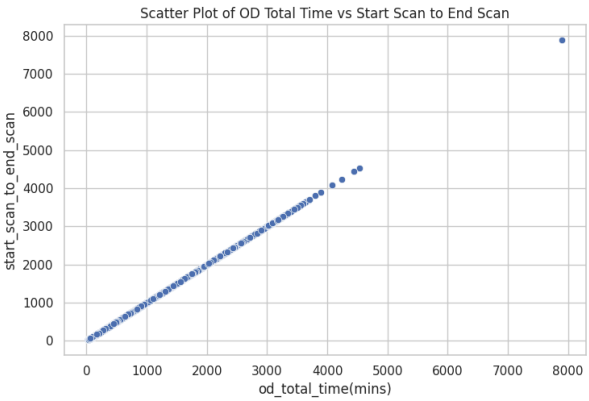
visual analysis

- Box plot: to visualize the distribution of the two variables and to detect any significant differences or outliers.
- Histogram: to show the distribution shapes and to identify whether the data is normally distributed or not
- Scatter plot: to see the relationship between two variables

In [55]:
```
fig = plt.figure(figsize=(15, 10))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=time_taken) #x="od_total_time(mins)", y="start_scan_to_end_scan")
plt.title('Boxplot of OD Total Time vs Start Scan to End Scan')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=time_taken)#, x="od_total_time(mins)", y="start_scan_to_end_scan'
plt.title('Histplot of OD Total Time vs Start Scan to End Scan')

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=time_taken, x="od_total_time(mins)", y="start_scan_to_end_scar
plt.title('Scatter Plot of OD Total Time vs Start Scan to End Scan')

# Display the plots
plt.tight_layout()
plt.show()
```







- based on histogram both variable does not follow normal distribution
- outliers are present in both variables

- Levene's Test: Statistic = 2.9409497054352514e-07,
- p-value = 0.9995673067810471
- The variances across the groups are equal.

- Variances across the variables are equal but it is not normally distributed hence we will perform non-paramatric Mann-Whitney U Test

- t_statistic : 109589340.0
- p_value : 0.7215156248063554
- Thesre is no difference between total time taken and expected start_scan_to_end_scan

- Statistically od_total_time and start_scan_to_end_scan are similar.

Hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value

In [56]:
```
actual_osrm = new_df[["trip_uuid","actual_time","osrm_time"]]
actual_osrm
```

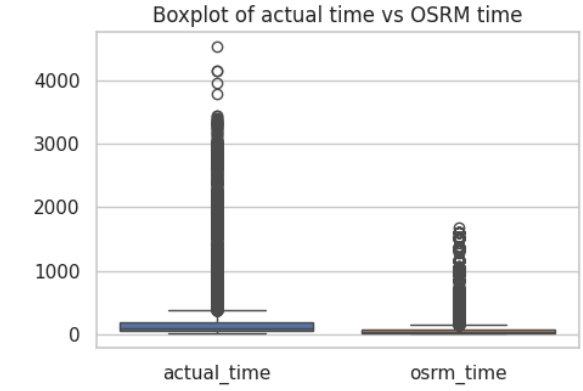
Out[56]:

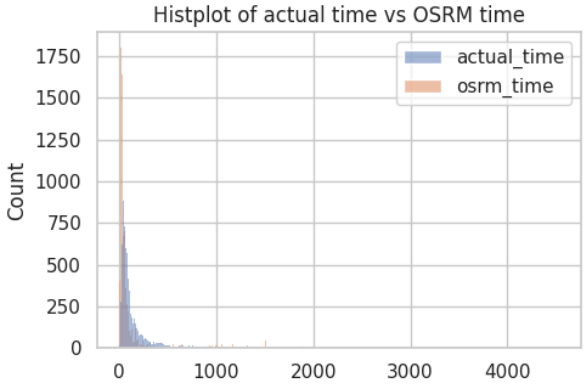
	trip_uuid	actual_time	osrm_time
0	trip-153671041653548748	830.0	388.0
1	trip-153671042288605164	96.0	42.0
2	trip-153671043369099517	2736.0	1528.0
3	trip-153671046011330457	59.0	15.0
4	trip-153671052974046625	63.0	27.0
...	...	...	...
14782	trip-153861095625827784	49.0	34.0
14783	trip-153861104386292051	21.0	12.0
14784	trip-153861106442901555	92.0	24.0
14785	trip-153861115439069069	30.0	14.0
14786	trip-153861118270144424	42.0	26.0

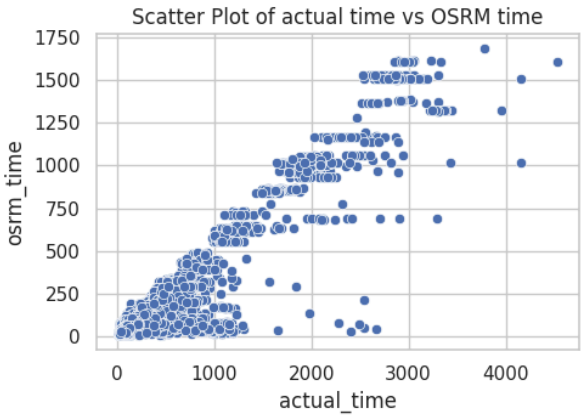
14787 rows × 3 columns

In [ ]:
```
# Visual analysis
```

In [58]:
```
fig = plt.figure(figsize=(10, 7))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=actual_osrm)
plt.title('Boxplot of actual time vs OSRM time')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=actual_osrm)
plt.title('Histplot of actual time vs OSRM time')

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=actual_osrm, x="actual_time", y="osrm_time")
plt.title('Scatter Plot of actual time vs OSRM time')

# Display the plots
plt.tight_layout()
plt.show()
```







- based on histogram both variable does not follow normal distribution
- outliers are present in both variables

- Levene's Test: Statistic = 615.4847491495041,
- p-value = 1.7065702203571972e-134
- The variances across the actual time and osrm time are not equal.

Hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value

In [59]:
```
actual_segment = new_df[["trip_uuid","actual_time","segment_actual_time"]]
actual_segment
```

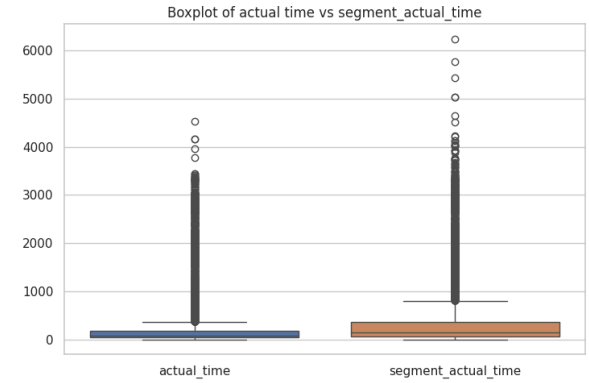
Out[59]:

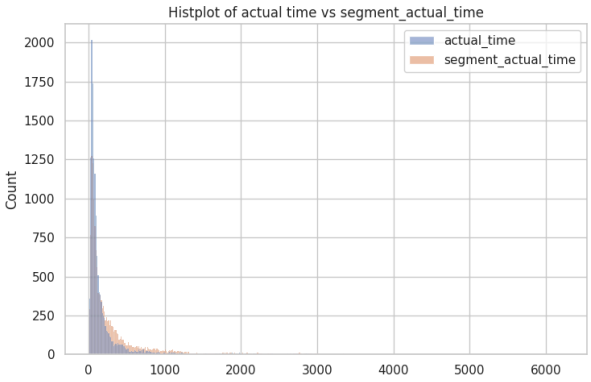
	trip_uuid	actual_time	segment_actual_time
0	trip-153671041653548748	830.0	1548.0
1	trip-153671042288605164	96.0	141.0
2	trip-153671043369099517	2736.0	3308.0
3	trip-153671046011330457	59.0	59.0
4	trip-153671052974046625	63.0	340.0
...	...	...	...
14782	trip-153861095625827784	49.0	82.0
14783	trip-153861104386292051	21.0	21.0
14784	trip-153861106442901555	92.0	281.0
14785	trip-153861115439069069	30.0	258.0
14786	trip-153861118270144424	42.0	274.0

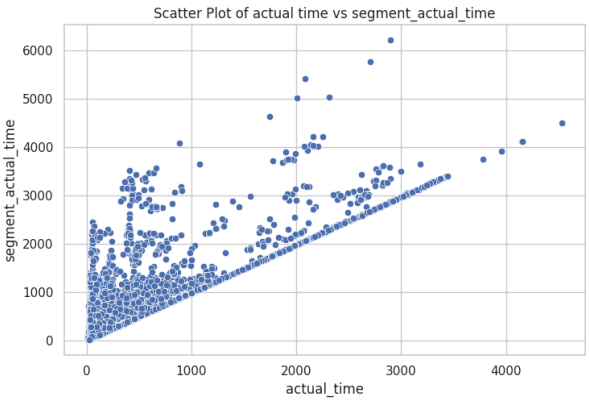
14787 rows × 3 columns

In [60]:

```
fig = plt.figure(figsize=(15, 10))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=actual_segment)
plt.title('Boxplot of actual time vs segment_actual_time')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=actual_segment)
plt.title('Histplot of actual time vs segment_actual_time')

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=actual_segment, x="actual_time", y="segment_actual_time")
plt.title('Scatter Plot of actual time vs segment_actual_time')

# Display the plots
plt.tight_layout()
plt.show()
```







- These two variables does not follow the normal distribution
- outliers are present in both variables

- Levene's Test: Statistic = 363.2235493377645,
- p-value = 1.7012850612953372e-80
- The variances across the actual time and segment actual time are not equal.

- homogenity of Variances and normality of data assumptions are not satisfied hence we will perform non-paramatric Mann-Whitney U Test
- t_statistic : 162076565.0
- p_value : 0.0
- There is a difference between actual time and OSRM time

Based on statistical analysis actual time and OSRM time are not same.

Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

In [61]:
```
actual_segment = new_df[["trip_uuid","actual_time","segment_actual_time"]]
actual_segment
```

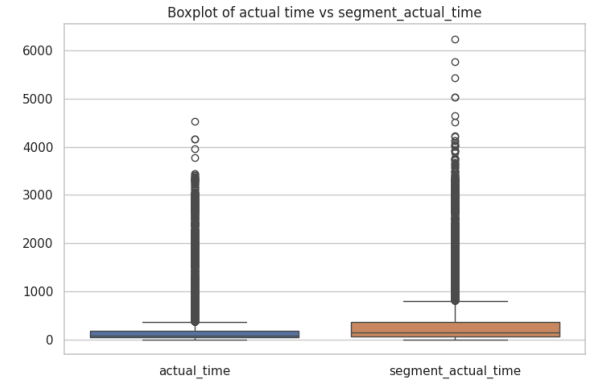
Out[61]:

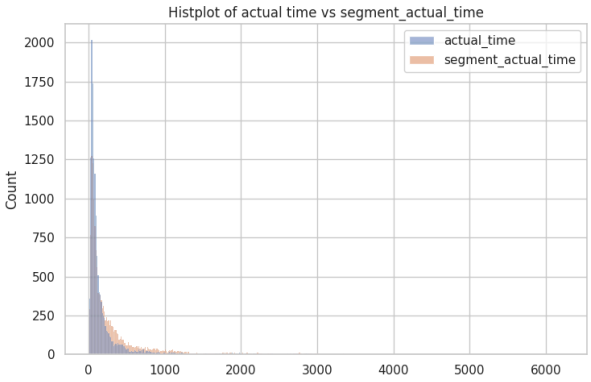
	trip_uuid	actual_time	segment_actual_time
0	trip-153671041653548748	830.0	1548.0
1	trip-153671042288605164	96.0	141.0
2	trip-153671043369099517	2736.0	3308.0
3	trip-153671046011330457	59.0	59.0
4	trip-153671052974046625	63.0	340.0
...	...	...	...
14782	trip-153861095625827784	49.0	82.0
14783	trip-153861104386292051	21.0	21.0
14784	trip-153861106442901555	92.0	281.0
14785	trip-153861115439069069	30.0	258.0
14786	trip-153861118270144424	42.0	274.0

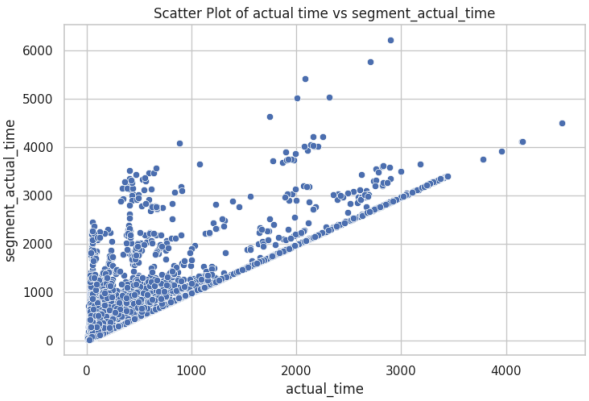
14787 rows × 3 columns

In [62]:

```
fig = plt.figure(figsize=(15, 10))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=actual_segment)
plt.title('Boxplot of actual time vs segment_actual_time')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=actual_segment)
plt.title('Histplot of actual time vs segment_actual_time')

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=actual_segment, x="actual_time", y="segment_actual_time")
plt.title('Scatter Plot of actual time vs segment_actual_time')

# Display the plots
plt.tight_layout()
plt.show()
```







- These two variables does not follow the normal distribution
- outliers are present in both variables

- Levene's Test: Statistic = 363.2235493377645,

- p-value = 1.7012850612953372e-80
- The variances across the actual time and segment actual time are not equal.

Both variables does not follow assumptions of normality and varience of homogeneity for t-test, hence we will perform perform non-paramatric Mann-Whitney U Test

- statistic : 84217886.5
- p_value : 2.0025997464382218e-256
- There is a difference between actual time and segment actual time

Based on non paramatric Mann-Whitney U Test, actual time and segment actual time are not same.

Hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

In [64]:
```
osrm_segment = new_df[["trip_uuid","osrm_distance", "segment_osrm_distance"]]
osrm_segment
```

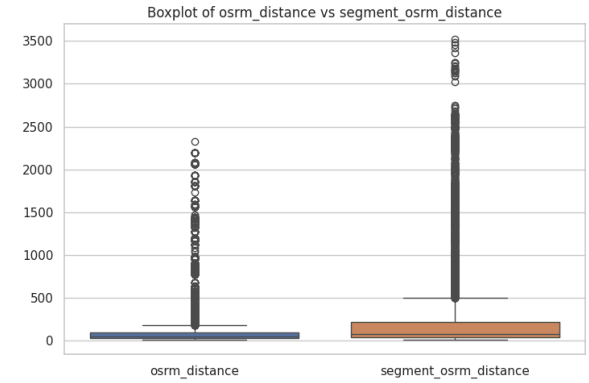
Out[64]:

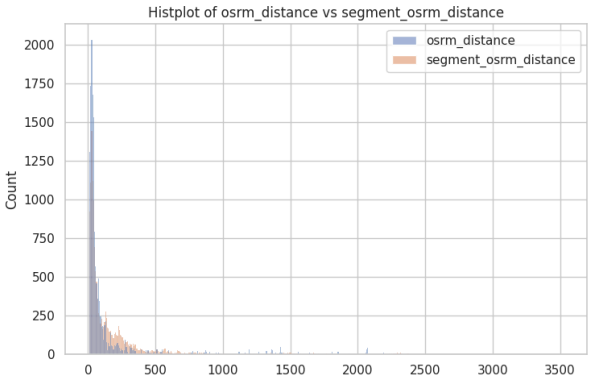
	trip_uuid	osrm_distance	segment_osrm_distance
0	trip-153671041653548748	544.8027	1320.4733
1	trip-153671042288605164	56.9116	84.1894
2	trip-153671043369099517	2072.8556	2545.2678
3	trip-153671046011330457	19.6800	19.8766
4	trip-153671052974046625	29.5696	146.7919
...	...	...	...
14782	trip-153861095625827784	44.5639	64.8551
14783	trip-153861104386292051	16.0882	16.0883
14784	trip-153861106442901555	28.8492	104.8866
14785	trip-153861115439069069	16.0185	223.5324
14786	trip-153861118270144424	28.0484	80.5787

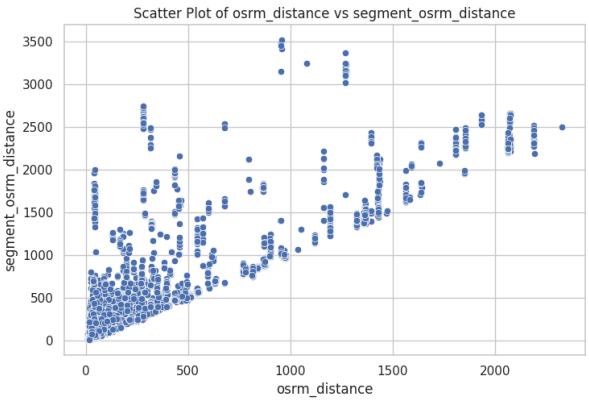
14787 rows × 3 columns

In [65]:

```
fig = plt.figure(figsize=(15, 10))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=osrm_segment)
plt.title('Boxplot of osrm_distance vs segment_osrm_distance')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=osrm_segment)
plt.title('Histplot of osrm_distance vs segment_osrm_distance')

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=osrm_segment, x="osrm_distance", y="segment_osrm_distance")
plt.title('Scatter Plot of osrm_distance vs segment_osrm_distance')

# Display the plots
plt.tight_layout()
plt.show()
```







- These two variables does not follow the normal distribution
- outliers are present in both variables

- statistic : 82950404.0
- p_value : 9.763204777245944e-283
- There is a difference between osrm distance and segment osrm distance

- Levene's Test: Statistic = 432.1415884126009,
- p-value = 2.6727169494145576e-95
- The variances across the osrm distance and segment osrm distance are not equal.
- Both variables does not follow assumptions of normality and varience of homogeneity for t-test, hence we will perform perform non-paramatric Mann-Whitney U Test

Based on non paramatric Mann-Whitney U Test, osrm distance and segment osrm distance are not same.

Hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value

In [66]:
```
osrm_segment_osrm_time = new_df[["trip_uuid","osrm_time", "segment_osrm_time"]]
osrm_segment_osrm_time
```

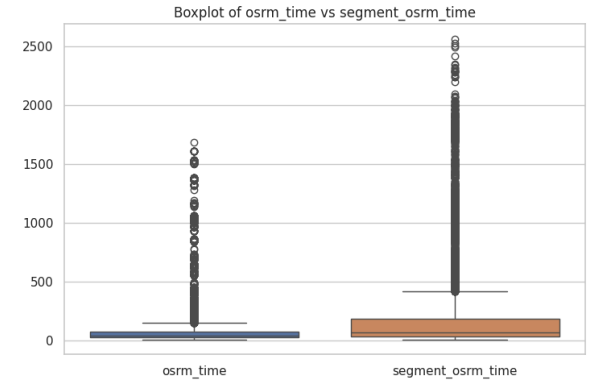
Out[66]:

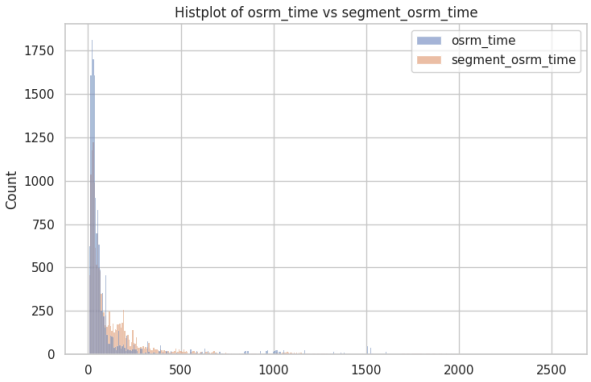
	trip_uuid	osrm_time	segment_osrm_time
0	trip-153671041653548748	388.0	1008.0
1	trip-153671042288605164	42.0	65.0
2	trip-153671043369099517	1528.0	1941.0
3	trip-153671046011330457	15.0	16.0
4	trip-153671052974046625	27.0	115.0
...	...	...	...
14782	trip-153861095625827784	34.0	62.0
14783	trip-153861104386292051	12.0	11.0
14784	trip-153861106442901555	24.0	88.0
14785	trip-153861115439069069	14.0	221.0
14786	trip-153861118270144424	26.0	67.0

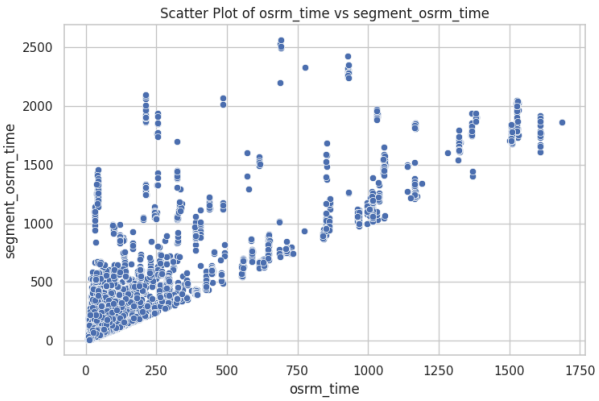
14787 rows × 3 columns

In [67]:

```
fig = plt.figure(figsize=(15, 10))

# Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(data=osrm_segment_osrm_time)
plt.title('Boxplot of osrm_time vs segment_osrm_time')

# Histogram
plt.subplot(2, 2, 2)
sns.histplot(data=osrm_segment_osrm_time)
plt.title("Histplot of osrm_time vs segment_osrm_time")

# Scatter Plot
plt.subplot(2, 2, 3)
sns.scatterplot(data=osrm_segment_osrm_time, x="osrm_time", y="segment_osrm_time")
plt.title("Scatter Plot of osrm_time vs segment_osrm_time")

# Display the plots
plt.tight_layout()
plt.show()
```







- Both variables, osrm_time, segment_osrm_time does not follow the assumptions of normality.
- outliers are present in both variables

- Levene's Test: Statistic = 540.1857224759467,
- p-value = 1.983498152793614e-118
- The variances across the osrm_time and segment_osrm_time are not equal.

- Both variables, osrm_time and segment_osrm_time does not follow assumptions of normality and varience of homogeneity for t-test, hence we will perform perform non-paramatric Mann-Whitney U Test

- statistic : 81360794.5

- p_value : 0.0
- There is a difference between osrm_time and segment_osrm_time

Based on non paramatric Mann-Whitney U Test, osrm time and segment osrm time are not same.

one-hot encoding of categorical variables

In [69]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
new_df["route_type_encoded"] = label_encoder.fit_transform(new_df["route_type"])
new_df["data_type_encoded"] = label_encoder.fit_transform(new_df["data"])
new_df.head(5)
```

Out[69]:

	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (I
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgao
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mum
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_

5 rows × 29 columns

Normalization Standardization of the numerical features using MinMaxScaler or StandardScaler.

In [70]:
```
new_df.columns
```

Out[70]:
```
Index(['trip_uuid', 'data', 'route_type', 'trip_creation_time', 'source_name',
       'destination_name', 'od_total_time(mins)', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'source_state', 'destination_state',
       'source_city', 'destination_city', 'trip_creation_date',
       'trip_creation_year', 'trip_creation_month', 'trip_completion_month',
       'trip_creation_day', 'trip_creation_weekday', 'trip_end_weekday',
       'trip_end_day', 'route_type_encoded', 'data_type_encoded'],
      dtype='object')
```

In [71]:
```
#MinMaxScaler
num_column_scale = ['od_total_time(mins)', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
new_df[num_column_scale] = scaler.fit_transform(new_df[num_column_scale])
new_df.head(5)
```

Out[71]:

	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (I
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgao
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mum
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_

5 rows × 29 columns

In [72]:
```
#StandardScaler
num_column_scale = ['od_total_time(mins)', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance']
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
new_df[num_column_scale] = scaler.fit_transform(new_df[num_column_scale])
new_df.head(5)
```

Out[72]:

	trip_uuid	data	route_type	trip_creation_time	source_name	desti
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Kanpu (I
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	Doddablpur_ChikaDPP_D (Karnataka)	Doddablpu
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	Gurgaon_Bilaspur_HB (Haryana)	Gurgao
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	Mumbai Hub (Maharashtra)	Mum
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	Bellary_Dc (Karnataka)	Sandur_

5 rows × 29 columns

In [73]:
```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14787 entries, 0 to 14786
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   trip_uuid                       14787 non-null  object
 1   data                            14787 non-null  category
 2   route_type                      14787 non-null  category
 3   trip_creation_time              14787 non-null  datetime64[ns]
 4   source_name                     14787 non-null  object
 5   destination_name                14787 non-null  object
 6   od_total_time(mins)             14787 non-null  float64
 7   start_scan_to_end_scan          14787 non-null  float64
 8   actual_distance_to_destination  14787 non-null  float64
 9   actual_time                     14787 non-null  float64
 10  osrm_time                       14787 non-null  float64
 11  osrm_distance                   14787 non-null  float64
 12  segment_actual_time             14787 non-null  float64
 13  segment_osrm_time               14787 non-null  float64
 14  segment_osrm_distance           14787 non-null  float64
 15  source_state                    14787 non-null  object
 16  destination_state               14787 non-null  object
 17  source_city                     14787 non-null  object
 18  destination_city                14787 non-null  object
 19  trip_creation_date              14787 non-null  datetime64[ns]
 20  trip_creation_year              14787 non-null  int32
 21  trip_creation_month             14787 non-null  int32
 22  trip_completion_month           14689 non-null  float64
 23  trip_creation_day               14787 non-null  int32
 24  trip_creation_weekday           14787 non-null  object
 25  trip_end_weekday                14689 non-null  object
 26  trip_end_day                    14689 non-null  float64
 27  route_type_encoded              14787 non-null  int64
 28  data_type_encoded               14787 non-null  int64
dtypes: category(2), datetime64[ns](2), float64(11), int32(3), int64(2), object(9)
memory usage: 2.9+ MB
```

Insights:

1. Since data scientist team wants to create a forecastingmodel, data set is already been devided into training(72%) and test data(28%)
2. Entire data is of 26 days
3. 20 source name and 18 destination name is missing in data set. If drop those null values , it will lead to miscalculation for further analysis
4. Highest number of trip cretion is being done on wednesday and highest number of trip completion is being done on wednesday and saturday
5. Highest number of trip cretion and trip completion is being done during mid month and start decresing after that. Very low trip completion during 2nd week of month
6. Highest number of trips are created and completed in 9th month
7. 60 % trips are made through carting route and 40% trips are made through FTL
8. Mumbai, Gurgaon and Bengaluru has done the more number of total trips(created and completed). Hence Maharashtra, Karnataka and Haryana has also done the more number of total trips.

9. od_total_time and start_scan_to_end_scan are statistically similar with 95% confidence.
10. With 95% confidence actual_time and OSRM_time are not similar. This will give a wrong information to customore about estimate delivery date.
11. With 95 % confidence actual_time and segment_actual_time are statistically different.
12. osrm_distance and segment_osrm_distance are statistically different with 95% confidence.
13. osrm_time and segment_osrm_time are also statistically different with 95% confidence.

Recommandations:

1. An open-source routing engine(OSRM) for time, distance, segment time and distance calculator needs to be optimized to reduced the error between computed data and actual data to give a nearly right estimated information about delevery package.
2. The reason for difference between actual_time and segment_actual_time can be due to the delivery person is not taking the predefine route or not starting the trip on time after scanning. Teams need to look into it.
3. Team should increase the number of delivery partenrs in mumbai, bengaluru and gurgaon during fetival seasons.