Open in Colab

DELHIVERY- case study

> *analyzed by-Suchi Sharma*

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.

Problem Statement

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.

It wants to understand and process the data coming out of data engineering pipelines. So we need to

• Clean, sanitize and manipulate data to get useful features out of raw fields

• Make sense out of the raw data and help the data science team to build forecasting models on it

• Detectiong outliers to refine the process of delivry and enhance the quality of services.

Benefits of case study

From Delhivery’s Perspective:

● It provides a practical framework for understanding and processing data, which is integral to their operations. By leveraging data engineering pipelines and data analysis techniques, Delhivery can achieve several critical goals:

● First, it allows them to ensure data integrity and quality by addressing missing values and structuring the dataset appropriately.

● Second, it enables the extraction of valuable features from raw data, which can be utilized for building accurate forecasting models.

● Moreover, it facilitates the identification of patterns, insights, and actionable recommendations crucial for optimizing their logistics operations.

● By conducting hypothesis testing and outlier detection, Delhivery can refine their processes and further enhance the quality of service they provide.

Importing necessary Libraries

In [ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import OneHotEncoder,StandardScaler
from scipy.stats import ttest_ind

import warnings
warnings.filterwarnings('ignore')
```

Downloading Dataset

In [ ]:
```
!gdown 1Rr38ickcD_qawNf2WrFVUVEsFo2oFEnb
```

```
Downloading...
From: https://drive.google.com/uc?id=1Rr38ickcD_qawNf2WrFVUVEsFo2oFEnb
To: /content/delhivery_dat.csv
100% 55.6M/55.6M [00:00<00:00, 67.6MB/s]
```

Reading Dataset

In [ ]:
```
dlvry_df=pd.read_csv('delhivery_dat.csv')
dlvry_df.head()
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

5 rows × 24 columns

Checking Data Structure and its attributes

In [ ]:

```
print(f'Rows of data: {dlvry_df.shape[0]}\nColumns of data: {dlvry_df.shape[
```

```
Rows of data: 144867
Columns of data: 24
```

In [ ]:

```
dlvry_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
```

There are 10 float columns, 1 bool , 1 int and 12 object datatype columns which needs to be checked.

Checking Duplicate Values

In [ ]:
```
dlvry_df.duplicated().value_counts()
```

Out[ ]:
```
False    144867
Name: count, dtype: int64
```

No duplicate values are given in the dataset.

Changing Datatypes

As we can see the columns like **trip_creation_time** , **od_start_time** , **od_end_time** are datetime columns but given as object dtype, so changing them to datetime datatype.

In [ ]:
```
dlvry_df['trip_creation_time'] = pd.to_datetime(dlvry_df['trip_creation_time
dlvry_df['od_start_time']=pd.to_datetime(dlvry_df['od_start_time'])
```

```
dlvry_df['od_end_time']=pd.to_datetime(dlvry_df['od_end_time'])
```

To check if various columns which are given as float datatype have int data or decimal/float data.

```
In [ ]: (dlvry_df['start_scan_to_end_scan'].astype('int')==  dlvry_df['start_scan_to
```

```
Out[ ]: start_scan_to_end_scan
        True     144867
        Name: count, dtype: int64
```

```
In [ ]: (dlvry_df['actual_time'].astype('int')==  dlvry_df['actual_time']).value_cou
```

```
Out[ ]: actual_time
        True     144867
        Name: count, dtype: int64
```

```
In [ ]: (dlvry_df['osrm_time'].astype('int')==  dlvry_df['osrm_time']).value_counts(
```

```
Out[ ]: osrm_time
        True     144867
        Name: count, dtype: int64
```

```
In [ ]: (dlvry_df['segment_actual_time'].astype('int')==  dlvry_df['segment_actual_t
```

```
Out[ ]: segment_actual_time
        True     144867
        Name: count, dtype: int64
```

```
In [ ]: (dlvry_df['segment_osrm_time'].astype('int')==  dlvry_df['segment_osrm_time'
```

```
Out[ ]: segment_osrm_time
        True     144867
        Name: count, dtype: int64
```

As we have seen that all these 5 columns are having int data only so can converting them to **int** datatype.

```
In [ ]: col_int=['start_scan_to_end_scan','actual_time','osrm_time','segment_actual_
        for col in col_int:
          dlvry_df[col]=dlvry_df[col].astype('int')
```

Coverting column **data** and **route_type** to Category datatype.

```
In [ ]: dlvry_df['data']=dlvry_df['data'].astype('category')
        dlvry_df['route_type']=dlvry_df['route_type'].astype('category')
```

```
In [ ]: # @title Lets check the datatypes of columns again
        dlvry_df.dtypes
```

Out[ ]:

```
data                                       category
trip_creation_time                   datetime64[ns]
route_schedule_uuid                          object
route_type                                 category
trip_uuid                                    object
source_center                                object
source_name                                  object
destination_center                           object
destination_name                             object
od_start_time                        datetime64[ns]
od_end_time                          datetime64[ns]
start_scan_to_end_scan                        int64
is_cutoff                                      bool
cutoff_factor                                 int64
cutoff_timestamp                             object
actual_distance_to_destination              float64
actual_time                                   int64
osrm_time                                     int64
osrm_distance                               float64
factor                                      float64
segment_actual_time                           int64
segment_osrm_time                             int64
segment_osrm_distance                       float64
segment_factor                              float64
dtype: object
```

Checking Missing Values

In [ ]:

```
round(dlvry_df.isnull().sum()/dlvry_df.shape[0]*100,2)
```

Out[ ]:
```
data                            0.00
trip_creation_time              0.00
route_schedule_uuid             0.00
route_type                      0.00
trip_uuid                       0.00
source_center                   0.00
source_name                     0.20
destination_center              0.00
destination_name                0.18
od_start_time                   0.00
od_end_time                     0.00
start_scan_to_end_scan          0.00
is_cutoff                       0.00
cutoff_factor                   0.00
cutoff_timestamp                0.00
actual_distance_to_destination  0.00
actual_time                     0.00
osrm_time                       0.00
osrm_distance                   0.00
factor                          0.00
segment_actual_time             0.00
segment_osrm_time               0.00
segment_osrm_distance           0.00
segment_factor                  0.00
dtype: float64
```

Treating Missing Values

In [ ]:
```
# Checking  if the missing name are there in other rows corresponding to sou
unknwn_source_code=dlvry_df[dlvry_df['source_name'].isnull()]['source_center
unknwn_source_code
```

Out[ ]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [ ]:
```
for code in unknwn_source_code:
  if code in dlvry_df[~dlvry_df['source_name'].isnull()]['source_center'].ur
    print(code,': Found')
  elif code in dlvry_df[~dlvry_df['destination_name'].isnull()]['destinatior
    print(code,':  found')
  else:
    print(code,': Not Found')
```

```
IND342902A1B : Not Found
IND577116AAA : Not Found
IND282002AAD : Not Found
IND465333A1B : Not Found
IND841301AAC : Not Found
IND509103AAC : Not Found
IND126116AAA : Not Found
IND331022A1B : Not Found
IND505326AAB : Not Found
IND852118A1B : Not Found
```

In [ ]:
```
unknwn_dest_code=dlvry_df[dlvry_df['destination_name'].isnull()]['destinatic
unknwn_dest_code
```

Out[ ]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

In [ ]:
```
for code in unknwn_dest_code:
  if code in dlvry_df[~dlvry_df['source_name'].isnull()]['source_center'].ur
    print(code,': Found')
  elif code in dlvry_df[~dlvry_df['destination_name'].isnull()]['destinatior
    print(code,':  found')
  else:
    print(code,': Not Found')
```

```
IND342902A1B : Not Found
IND577116AAA : Not Found
IND282002AAD : Not Found
IND465333A1B : Not Found
IND841301AAC : Not Found
IND505326AAB : Not Found
IND852118A1B : Not Found
IND126116AAA : Not Found
IND509103AAC : Not Found
IND221005A1A : Not Found
IND250002AAC : Not Found
IND331001A1C : Not Found
IND122015AAC : Not Found
```

Filling Missing Values

In [ ]:
```
# Filling missing values with corresponding center code and unknown to not t
for code in unknwn_source_code:
  dlvry_df.loc[dlvry_df['source_center']==code,'source_name']= dlvry_df.loc[
```

In [ ]:
```
for code in unknwn_dest_code:
  dlvry_df.loc[dlvry_df['destination_center']==code,'destination_name']= dlv
```

In [ ]:
```
dlvry_df.isnull().sum()
```

Out[ ]:
```
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
```

Woah! No more missing values!

In [ ]:
```
# @title Finding the time period for which data is available
print(f"Data is given from [{dlvry_df['trip_creation_time'].min()}] to [{dlv
```

```
Data is given from [2018-09-12 00:00:16.535741] to [2018-10-08 03:00:24.3534
79] time period.
```

Checking Unique Values

In [ ]:
```
for col in dlvry_df:
  print(f"Unique values for {col:<40}:{dlvry_df[col].nunique()}")
```

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

Dropping irrelavant columns

In [ ]:

```
dlvry_df.drop(columns=['is_cutoff','cutoff_factor','cutoff_timestamp','facto

# I am not able to find any usage of these column so dropping them
```

Statistical Summary

In [ ]:

```
dlvry_df.describe()
```

Out[ ]:

	trip_creation_time	od_start_time	od_end_time	start_scan_t
count	144867	144867	144867	14
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	
std	NaN	NaN	NaN	

As I came to know there is negative values in **segment_actual_time** column,which is not possible to have negative time for deliveries so we need to clean that too. So, I am that the negative values are putted by mistake.

In [ ]:
```
dlvry_df['segment_actual_time']=dlvry_df['segment_actual_time'].abs()
```

In [ ]:
```
dlvry_df.describe()
```

Out[ ]:

	trip_creation_time	od_start_time	od_end_time	start_scan_t
count	144867	144867	144867	14
mean	2018-09-22 13:34:23.659819264	2018-09-22 18:02:45.855230720	2018-09-23 10:04:31.395393024	
min	2018-09-12 00:00:16.535741	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	
25%	2018-09-17 03:20:51.775845888	2018-09-17 08:05:40.886155008	2018-09-18 01:48:06.410121984	
50%	2018-09-22 04:24:27.932764928	2018-09-22 08:53:00.116656128	2018-09-23 03:13:03.520212992	
75%	2018-09-27 17:57:56.350054912	2018-09-27 22:41:50.285857024	2018-09-28 12:49:06.054018048	
max	2018-10-03 23:59:42.701692	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	
std	NaN	NaN	NaN	

In [ ]:
```
dlvry_df.describe(include=object)
```

Out[ ]:

	route_schedule_uuid	trip_uuid	source_center	sou
count	144867	144867	144867	
unique	1504	14817	1508	
top	thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069f...	trip-153811219535896559	IND000000ACB	Gurgaon_B
freq	1812	101	23347	

- The trip creation time is from 12 september, 2018 to 03 October, 2018, while the trip start time is given till 06-October, 2018 and end time is till 08 Oct,2018.
- The average time taken for deliveries is 961 minutes that is near to 16 hours hile the maximum is 7898 hours which is near to 5-6 days and min is 20 minutes only. The median is quite low than mean showing presence of many outliers.
- Outliers detection in actual_distance_to_destination,actual_time,osrm_time,osrm_distance too.
- The min distance is 9 km while the max is 1927 km.
- The actual time taken for a delivery is as low as 9 min and as high as 4532 mins which is very high as orsm maximum time is only 1686 min means taken 2 days extra for a delivery which is not a good sign.
- The osrm max distance is 2326 while in actually we took short distance as max actual distance is only 1927 ,which can be a good sign if time taken is also less.
- Only 1504 unique route ids shows deliveries are repeated at same routes as 1812 times to a single route in a period of less than 30 days is surely a good thing.
- same trip_uuid is repeated 101 times shows its not a good sign to send so many deliveries in between as will lead to late deliveries.
- Most deliveries are from Gurgaon and to Gurgaon, haryana too. though for source it is more in number.

In [ ]:

```
# didnot understood the use of cumsum here as ultimately we have to do sum c
# we are not dealing with intermediate deliveries in our study so not making
dlvry_df['segment']=dlvry_df['trip_uuid']+'_'+dlvry_df['source_center']+'_'+
```

In [ ]:

```
segment_col=['segment_actual_time','segment_osrm_time','segment_osrm_distanc
for col in segment_col:
  dlvry_df[col+'_sum']=dlvry_df.groupby('segment')[col].cumsum()
dlvry_df.head()
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

5 rows × 23 columns

Aggregating Data

In [ ]:

```
dlvry_df[dlvry_df['trip_uuid']=='trip-153784572117438961'].iloc[:,4:15]
```

Out[ ]:

	trip_uuid	source_center	source_name	destinatio
38244	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38245	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38246	trip-153784572117438961	IND370001AAA	Bhuj_DC (Gujarat)	IND37
38247	trip-153784572117438961	IND370001AAA	Bhuj_DC (Gujarat)	IND37
38248	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38249	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38250	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38251	trip-153784572117438961	IND370110AAA	Anjar_DC (Gujarat)	IND37
38252	trip-153784572117438961	IND370615AAB	Nakhatrana_ClgRDDPP_D (Gujarat)	IND37
38253	trip-153784572117438961	IND370001AAA	Bhuj_DC (Gujarat)	IND37
38254	trip-153784572117438961	IND370001AAA	Bhuj_DC (Gujarat)	IND37

As we can see there are trip ids where same source is repeated twice and destination too at different time so it gives wrong analysis if we take first for time for samesource and last time for destiantion so using group by for start time too to aggregate data.

In [ ]:

```
# checking if group by working fine on a single trip id
df1=dlvry_df[dlvry_df['trip_uuid']=='trip-153741093647649320']
df1
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
5	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
6	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
7	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
8	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364
9	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364

10 rows × 23 columns

In [ ]:

```
agg_col={'data':'first',
         'trip_creation_time':'first',
        'route_schedule_uuid':'unique',
         'route_type':'first',
         'source_name':'first',
         'destination_name':'last',
         'od_end_time':'max',
       'start_scan_to_end_scan':'last',
       'actual_distance_to_destination':'last',
         'actual_time':'last',
         'osrm_time':'last',
         'osrm_distance':'max',
       'segment_actual_time':'sum',
       'segment_osrm_time':'sum',
        'segment_osrm_distance':'sum'}
```

In [ ]:
```
df2=df1.groupby(['trip_uuid','source_center','destination_center','od_start_
df2
```

Out[ ]:

	trip_uuid	source_center	destination_center	od_start_time	
0	trip-153741093647649320	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	tra
1	trip-153741093647649320	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	tra

In [ ]:
```
# Lets find out the differnece between start and end time before aggregating
# accurate results after aggregationg as there are some instances where end
df2['od_time_diff_min']=(df2['od_end_time']-df2['od_start_time']).dt.total_s
```

In [ ]:
```
agg_col_combine={'data':'first',
        'trip_creation_time':'first',
        'route_schedule_uuid':'first',
         'route_type':'first',
         'source_center':'first',
          'source_name':'first',
         'destination_center':'last',
          'destination_name':'last',
          'od_start_time':'min',
         'od_end_time':'max',
        'od_time_diff_min':'sum',
       'start_scan_to_end_scan':'sum',
       'actual_distance_to_destination':'sum',
         'actual_time':'sum',
         'osrm_time':'sum',
         'osrm_distance':'sum',
       'segment_actual_time':'sum',
       'segment_osrm_time':'sum',
        'segment_osrm_distance':'sum'}
```

In [ ]:
```
df2.groupby('trip_uuid').agg(agg_col_combine)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	rou
trip_uuid				
trip-153741093647649320	training	2018-09-20 02:35:36.476840	[thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c...	

In [ ]:
```
# performing aggregation on whole data
df_intermediate=dlvry_df.groupby(['trip_uuid','source_center','destination_c
df_intermediate=df_intermediate.sort_values(by=['trip_uuid','od_end_time'],a
df_intermediate
```

Out[ ]:

	trip_uuid	source_center	destination_center	od_start_tim
1	trip-153671041653548748	IND462022AAA	IND209304AAA	2018-09-1 00:00:16.53574
0	trip-153671041653548748	IND209304AAA	IND000000ACB	2018-09-1 16:39:46.85846
3	trip-153671042288605164	IND572101AAA	IND561203AAB	2018-09-1 00:00:22.88643
2	trip-153671042288605164	IND561203AAB	IND562101AAA	2018-09-1 02:03:09.65559
5	trip-153671043369099517	IND562132AAA	IND000000ACB	2018-09-1 00:00:33.69125
...	...	...	...	..
26364	trip-153861115439069069	IND628204AAA	IND627657AAA	2018-10-0 02:29:04.27219
26363	trip-153861115439069069	IND627657AAA	IND628613AAA	2018-10-0 03:31:11.18379
26365	trip-153861115439069069	IND628613AAA	IND627005AAA	2018-10-0 04:16:39.89487
26368	trip-153861118270144424	IND583201AAA	IND583119AAA	2018-10-0 02:51:44.71265
26367	trip-153861118270144424	IND583119AAA	IND583101AAA	2018-10-0 03:58:40.72654

26369 rows × 19 columns

In [ ]:

```
df_intermediate['od_time_diff_min']=round((df_intermediate['od_end_time']-df
```

In [ ]:

```
df=df_intermediate.groupby('trip_uuid').agg(agg_col_combine).reset_index()
df
```

Out[ ]:

	trip_uuid	data	trip_creation_time	route_schedule_uui
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	[thanos::sroute:d7c989b a29b-4a0b-b2f4-288cdc
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	[thanos::sroute:3a1b0ab bb0b-4c53-8c59-eb2a2c
2	trip-153671043369099517	training	2018-09-12 00:00:33.691250	[thanos::sroute:de5e208 7641-45e6-8100-4d9fb1
3	trip-153671046011330457	training	2018-09-12 00:01:00.113710	[thanos::sroute:f017649 a679-4597-8332-bbd1c7
4	trip-153671052974046625	training	2018-09-12 00:02:09.740725	[thanos::sroute:d9f07b1 65e0-4f3b-bec8-df0613
...	...	...	...	
14812	trip-153861095625827784	test	2018-10-03 23:55:56.258533	[thanos::sroute:8a12099 f577-4491-9e4b-b7e4a1
14813	trip-153861104386292051	test	2018-10-03 23:57:23.863155	[thanos::sroute:b30e1ec 3bfa-4bd2-a7fb-3b7576
14814	trip-153861106442901555	test	2018-10-03 23:57:44.429324	[thanos::sroute:5609c26 e436-4e0a-8180-3db4a7
14815	trip-153861115439069069	test	2018-10-03 23:59:14.390954	[thanos::sroute:c5f2ba2 8486-4940-8af6-d1d2a6
14816	trip-153861118270144424	test	2018-10-03 23:59:42.701692	[thanos::sroute:412fea1 6d1f-4222-8a5f-a51704

14817 rows × 20 columns

In [ ]:

```
dlvry_df['trip_uuid'].nunique()
```

Out[ ]: 14817

We got the 14817 rows after aggregation based on trip uuid which is same as trip unique ids, which shows our aggregation is perfect.

In [ ]:

```
df[~(df['od_time_diff_min']-df['start_scan_to_end_scan']<=6)]
# tried various values here to get maximum difference
```

Out[ ]:

trip_uuid	data	trip_creation_time	route_schedule_uuid	route_type	source_

From the above code we can see that the given start to end scan column have similar data to the column we featured by subtracting start time from end time. so we can drop any of them.
As we have source name and destination name, their codes name are not needed for analysis.

In [ ]:
```
df.drop(columns=['od_time_diff_min','source_center','destination_center'],ir
```

Splitting Columns to get features

In [ ]:
```
# Functions to split data
def ext_state(col):
  state=col.split(' (')[-1]
  if len(state)>1:
    return state[:-1]
  else:
    return col

def ext_city(col):
  city=col.split("_")
  if len(city)>1:      # handling exception cases
    return city[0]
  else:
    city=col.split()
    if len(city)>1:
      return city[0]
    else:
      return col            # handling missing data values

def ext_place(col):
  place=col.split("_")
  if len(place)>2:
    return place[1]
  elif len(place)>1:
    return place[0]
  else:
    place=col.split()
    if len(place)>2:
      return place[1]
    else:
      return place[0]
```

In [ ]:
```
df['source_state']=df['source_name'].apply(lambda x:ext_state(x))
df['source_city']=df['source_name'].apply(lambda x:ext_city(x))
df['source_place']=df['source_name'].apply(lambda x:ext_place(x))
df['destination_state']=df['destination_name'].apply(lambda x:ext_state(x))
df['destination_city']=df['destination_name'].apply(lambda x:ext_city(x))
df['destination_place']=df['destination_name'].apply(lambda x:ext_place(x))
df.iloc[:,10:]
```

Out[ ]:

	actual_distance_to_destination	actual_time	osrm_time	osrm_distance
0	824.732854	1562	717	991.3523
1	73.186911	143	68	85.1110
2	1927.404273	3347	1740	2372.0852
3	17.175274	59	15	19.6800
4	127.448500	341	117	146.7918
...	...	...	...	...
14812	57.762332	83	62	73.4630
14813	15.513784	21	12	16.0882
14814	38.684839	282	48	63.2841
14815	134.723836	264	179	177.6635
14816	66.081533	275	68	80.5787

14817 rows × 13 columns

In [ ]:

```
df['trip_hour']=df['trip_creation_time'].dt.hour
df['trip_day']=df['trip_creation_time'].dt.day
df['trip_month']=df['trip_creation_time'].dt.month
df['trip_week']=df['trip_creation_time'].dt.isocalendar().week
df['trip_weekday']=df['trip_creation_time'].dt.dayofweek
df.iloc[:50,15:]
```

Out[ ]:

	segment_osrm_time	segment_osrm_distance	source_state	source_city	s
0	1008	1320.4733	Madhya Pradesh	Bhopal	
1	65	84.1894	Karnataka	Tumkur	
2	1941	2545.2678	Karnataka	Bangalore	
3	16	19.8766	Maharashtra	Mumbai	
4	115	146.7919	Karnataka	Bellary	
5	23	28.0647	Tamil Nadu	Chennai	
6	13	12.0184	Tamil Nadu	Chennai	
7	34	28.9203	Karnataka	HBR	
8	29	30.9358	Gujarat	Surat	
9	14	16.0860	Delhi	Delhi	
10	17	18.5887	Maharashtra	Pune	
11	9	10.8159	Haryana	FBD	
12	224	297.1037	Maharashtra	Kolhapur	
13	492	623.3792	Telangana	Hyderabad	
14	98	109.5132	Telangana	Thirumalagiri	
15	258	293.8447	Karnataka	Gulbarga	
16	27	31.1996	Rajasthan	Jaipur	
17	130	184.8169	Uttar Pradesh	Allahabad	
18	25	22.6548	Delhi	Delhi	
19	19	21.4180	Assam	Guwahati	
20	91	97.0273	Uttar Pradesh	Kanpur	
21	132	140.5623	Madhya Pradesh	Narsinghpur	
22	29	30.5457	Gujarat	Surat	
23	357	399.7294	Maharashtra	Nashik	
24	66	71.3328	West Bengal	Kolkata	
25	78	86.9866	Andhra Pradesh	Madakasira	
26	49	56.7577	Assam	Sonari	
27	83	76.1272	Karnataka	Bengaluru	
28	69	59.1472	Karnataka	Bengaluru	
29	26	30.4646	Telangana	Hyderabad	
30	329	198.9714	Tamil Nadu	Dindigul	

	segment_osrm_time	segment_osrm_distance	source_state	source_city	s
31	72	93.6079	Punjab	Jalandhar	
32	81	87.1703	Haryana	Faridabad	
33	109	98.7879	Punjab	Chandigarh	
34	17	21.2879	Maharashtra	Mumbai	
35	37	52.0204	Maharashtra	Deoli	
36	83	92.4425	Maharashtra	Pandharpur	
37	20	20.8831	West Bengal	CCU	
38	104	135.0386	Maharashtra	Bhandara	
39	471	596.8154	Karnataka	Bangalore	
40	55	80.1495	Haryana	FBD	
41	1003	1360.3053	Maharashtra	Bhiwandi	
42	185	204.5152	Punjab	Bhatinda	
43	1131	1472.7442	Delhi	Delhi	
44	180	235.7202	Rajasthan	Jaipur	
45	199	240.9679	Delhi	Delhi	
46	562	700.0514	Maharashtra	Pune	
47	65	69.0651	Punjab	Bhatinda	
48	26	40.1680	Maharashtra	Bhiwandi	
49	203	202.9714	Punjab	Chandigarh	

In [ ]:

```
# dropping columns which are no more useful as extracted data from them into
df.drop(columns=['trip_creation_time','route_schedule_uuid','source_name','d
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype   
---  ------                          --------------  -----   
 0   data                            14817 non-null  category
 1   route_type                      14817 non-null  category
 2   start_scan_to_end_scan          14817 non-null  int64   
 3   actual_distance_to_destination  14817 non-null  float64 
 4   actual_time                     14817 non-null  int64   
 5   osrm_time                       14817 non-null  int64   
 6   osrm_distance                   14817 non-null  float64 
 7   segment_actual_time             14817 non-null  int64   
 8   segment_osrm_time               14817 non-null  int64   
 9   segment_osrm_distance           14817 non-null  float64 
 10  source_state                    14817 non-null  object  
 11  source_city                     14817 non-null  object  
 12  source_place                    14817 non-null  object  
 13  destination_state               14817 non-null  object  
 14  destination_city                14817 non-null  object  
 15  destination_place               14817 non-null  object  
 16  trip_hour                       14817 non-null  int32   
 17  trip_day                        14817 non-null  int32   
 18  trip_month                      14817 non-null  int32   
 19  trip_week                       14817 non-null  UInt32  
 20  trip_weekday                    14817 non-null  int32   
dtypes: UInt32(1), category(2), float64(3), int32(4), int64(5), object(6)
memory usage: 1.9+ MB
```

We can see after cleaning, aggregating and handling data, data space is optimized from 25+ MB to approx. 2 MB only.

EXPLORATORY DATA ANALYSIS

In [ ]:
```
df['data'].value_counts()/len(df)*100
```

Out[ ]:
```
data
training    71.903894
test        28.096106
Name: count, dtype: float64
```

In [ ]:
```
df['route_type'].value_counts()/len(df)*100
```

Out[ ]:
```
route_type
Carting    60.120132
FTL        39.879868
Name: count, dtype: float64
```

In [ ]:
```
temp=df['start_scan_to_end_scan'].value_counts()
temp.sort_index(ascending=False).head(20)
```

Out[ ]:

```
start_scan_to_end_scan
7898    1
7458    1
6495    1
5864    1
5807    1
5688    1
5686    1
4846    1
4699    1
4616    1
4562    1
4535    1
4488    1
4475    1
4467    1
4461    1
4440    1
4410    2
4395    1
4384    1
Name: count, dtype: int64
```

In [ ]:

```
df['source_state'].value_counts()
```

Out[ ]:
```
source_state
Maharashtra                     2682
Karnataka                       2229
Haryana                         1684
Tamil Nadu                      1085
Delhi                            793
Telangana                        780
Gujarat                          746
Uttar Pradesh                    713
West Bengal                      677
Punjab                           630
Rajasthan                        493
Andhra Pradesh                   407
Bihar                            357
Madhya Pradesh                   332
Kerala                           289
Assam                            273
Jharkhand                        160
Uttarakhand                      114
Orissa                           107
Goa                               65
Chandigarh                        48
Chhattisgarh                      43
Himachal Pradesh                  34
Jammu & Kashmir                   17
IND282002AAD_unknownsourc         16
Dadra and Nagar Haveli            15
Pondicherry                       12
Nagaland                           5
Mizoram                            4
Arunachal Pradesh                  4
IND841301AAC_unknownsourc          1
IND577116AAA_unknownsourc          1
IND331022A1B_unknownsourc          1
Name: count, dtype: int64
```

In [ ]:
```
df['source_city'].value_counts()
```

Out[ ]:
```
source_city
Gurgaon          1024
Bengaluru        1015
Mumbai            893
Bhiwandi          811
Bangalore         755
                 ...
Thiruvadanai        1
Bulndshahr          1
Sindagi             1
Rupnarayanpur       1
Phulera             1
Name: count, Length: 668, dtype: int64
```

In [ ]:
```
# I analysed that bangalore and bengaluru is given as 2 different names in c
df.replace('Bangalore','Bengaluru',inplace=True)
```

In [ ]:
```
df['source_city'].value_counts()
```

Out[ ]:
```
source_city
Bengaluru        1770
Gurgaon          1024
Mumbai            893
Bhiwandi          811
Delhi             620
                 ...
Thiruvadanai        1
Bulndshahr          1
Sindagi             1
Rupnarayanpur       1
Phulera             1
Name: count, Length: 667, dtype: int64
```

In [ ]:
```
df[['source_city','source_place']].value_counts()
```

Out[ ]:
```
source_city  source_place
Gurgaon      Bilaspur        970
Bhiwandi     Mankoli         811
Bengaluru    Nelmngla        732
             Bomsndra        428
Chandigarh   Mehmdpur        370
                            ...
Dhaka        PchpkrRD          1
Dhampur      NaginaRD          1
Dharmavram   SaiNgr            1
Mudigere     HesglDPP          1
Kalpakkam    Sadras            1
Name: count, Length: 824, dtype: int64
```

In [ ]:
```
df['destination_state'].value_counts()
```

Out[ ]:
```
destination_state
Maharashtra                  2591
Karnataka                    2275
Haryana                      1667
Tamil Nadu                   1072
Telangana                     838
Gujarat                       746
Uttar Pradesh                 728
West Bengal                   708
Punjab                        693
Delhi                         675
Rajasthan                     516
Andhra Pradesh                414
Bihar                         361
Madhya Pradesh                337
Kerala                        273
Assam                         234
Jharkhand                     168
Orissa                        119
Uttarakhand                   113
Goa                            65
Chhattisgarh                   43
Himachal Pradesh               40
Chandigarh                     29
Arunachal Pradesh              23
IND282002AAD_unknownsdes       19
Dadra and Nagar Haveli         17
Jammu & Kashmir                15
Pondicherry                    10
Meghalaya                       8
Mizoram                         6
IND250002AAC_unknownsdes        3
IND122015AAC_unknownsdes        2
IND221005A1A_unknownsdes        1
IND331001A1C_unknownsdes        1
IND841301AAC_unknownsdes        1
IND505326AAB_unknownsdes        1
IND852118A1B_unknownsdes        1
IND577116AAA_unknownsdes        1
Tripura                         1
Nagaland                        1
Daman & Diu                     1
Name: count, dtype: int64
```

In [ ]:
```
df['destination_city'].value_counts()
```

Out[ ]:
```
destination_city
Bengaluru     1702
Mumbai        1127
Gurgaon        869
Hyderabad      630
Bhiwandi       604
              ... 
Shindkheda       1
Aliganj          1
Shevgaon         1
Sillod           1
Lunawada         1
Name: count, Length: 766, dtype: int64
```

In [ ]:
```
df[['destination_city','destination_place']].value_counts()
```

Out[ ]:
```
destination_city  destination_place
Gurgaon           Bilaspur             856
Bengaluru         Nelmngla             628
Bhiwandi          Mankoli              604
Hyderabad         Shamshbd             459
Chandigarh        Mehmdpur             434
                                      ... 
Baraut            SrnprHwy               1
Nalgonda          HydRoad                1
Champhai          AwmpiVng               1
Champa            Brplicwk               1
Chennai           Poonamallee            1
Name: count, Length: 914, dtype: int64
```

In [ ]:
```
df['trip_day'].value_counts().reset_index().sort_values(by='trip_day')
```

Out[ ]:

	trip_day	count
19	1	605
20	2	552
15	3	631
3	12	747
2	13	750
7	14	712
1	15	783
16	16	616
6	17	722
0	18	791
11	19	676
8	20	704
4	21	740
5	22	740
14	23	631
12	24	660
9	25	697
10	26	685
13	27	652
17	28	608
18	29	607
21	30	508

In [ ]:

```
df['trip_hour'].value_counts().reset_index()
```

Out[ ]:

	trip_hour	count
0	22	1125
1	23	1107
2	20	1082
3	0	994
4	21	873
5	19	837
6	1	750
7	2	702
8	18	698
9	3	652
10	4	636
11	6	611
12	17	595
13	16	526
14	5	509
15	7	473
16	15	469
17	14	379
18	8	346
19	13	329
20	9	324
21	12	271
22	11	267
23	10	262

In [ ]:

```
df['trip_week'].value_counts().reset_index().sort_values(by='trip_week')
```

Out[ ]:

	trip_week	count
2	37	3608
0	38	5004
1	39	4417
3	40	1788

In [ ]:
```python
df['trip_weekday'].value_counts().reset_index()
```

Out[ ]:

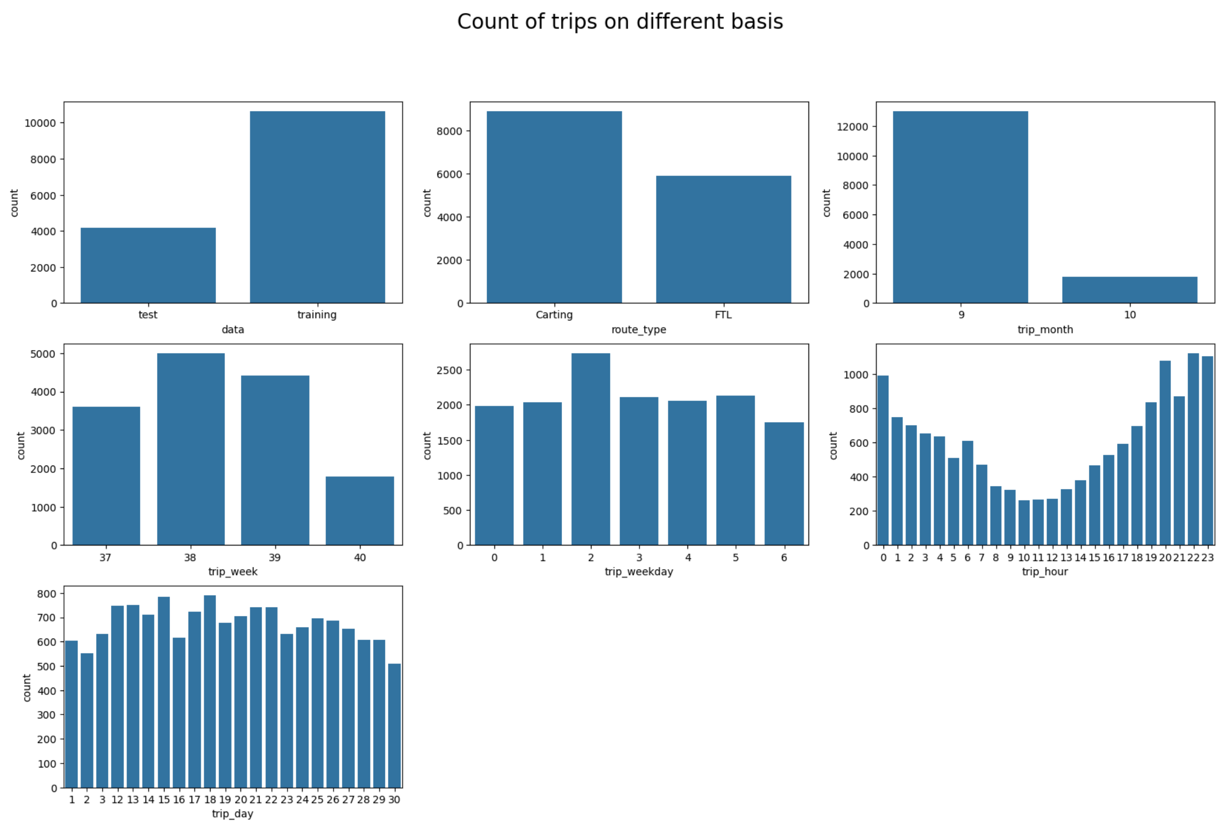
	trip_weekday	count
0	2	2739
1	5	2130
2	3	2106
3	4	2060
4	1	2040
5	0	1987
6	6	1755

In [ ]:
```python
df['trip_month'].value_counts()
```

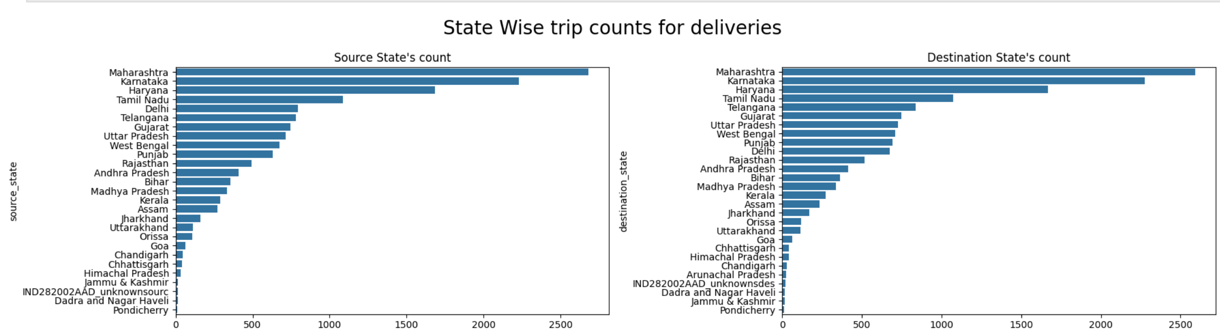
Out[ ]:
```
trip_month
9     13029
10     1788
Name: count, dtype: int64
```

Visual Analysis

In [ ]:
```python
plot_col=['data','route_type' , 'trip_month', 'trip_week',
       'trip_weekday' ,'trip_hour', 'trip_day']
i=1
plt.figure(figsize=(20,12))
plt.suptitle("Count of trips on different basis", fontsize=20)
for col in plot_col:
  plt.subplot(3,3,i)
  sns.countplot(data=df,x=col)
  i+=1
plt.show()
```

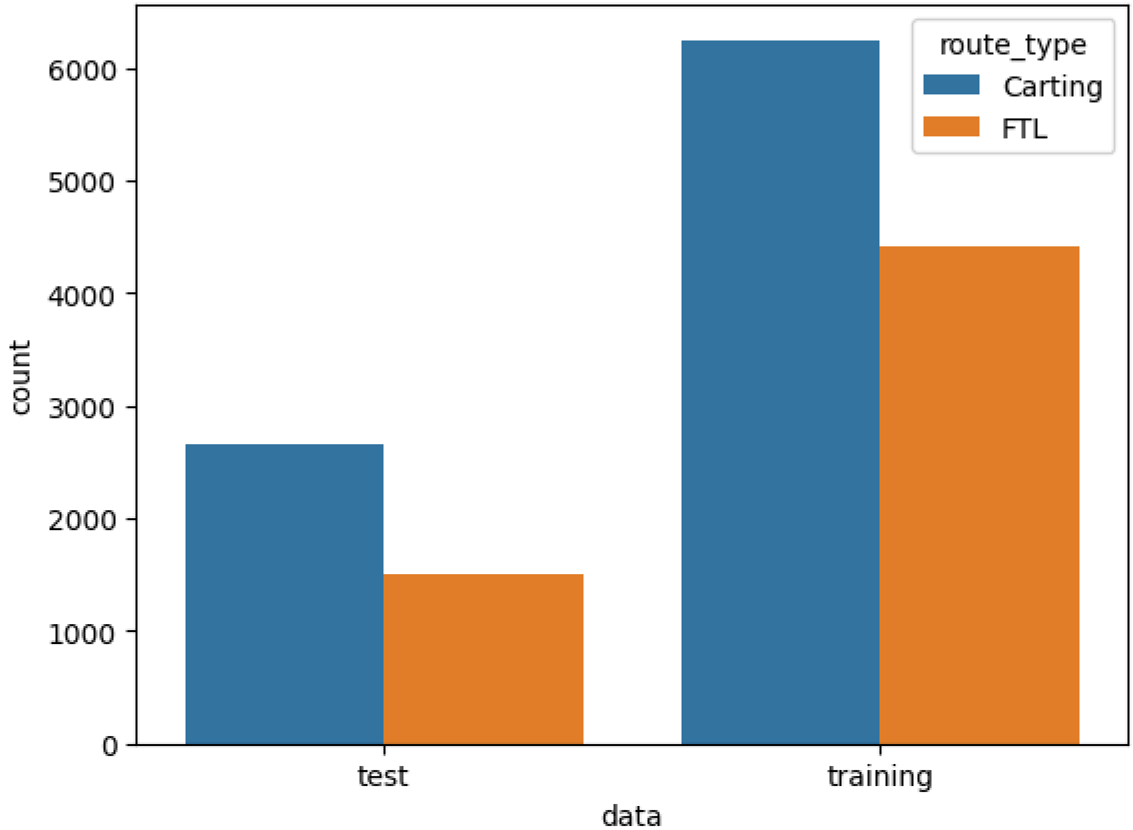



```
In [ ]: plt.figure(figsize=(18,5))
        plt.suptitle('State Wise trip counts for deliveries',fontsize=20)

        plt.subplot(1,2,1)
        plt.title("Source State's count")
        source_counts=df['source_state'].value_counts()
        source_counts=source_counts[source_counts>=10]
        sns.barplot(y=source_counts.index,x=source_counts,orient='h')
        plt.tight_layout()

        plt.subplot(1,2,2)
        plt.title("Destination State's count")
        dest_counts=df['destination_state'].value_counts()
        dest_counts=dest_counts[dest_counts>=10]
        sns.barplot(y=dest_counts.index,x=dest_counts,orient='h')
        plt.tight_layout()

        plt.show()
```




```
In [ ]: sns.countplot(data=df,x='data',hue='route_type')
```

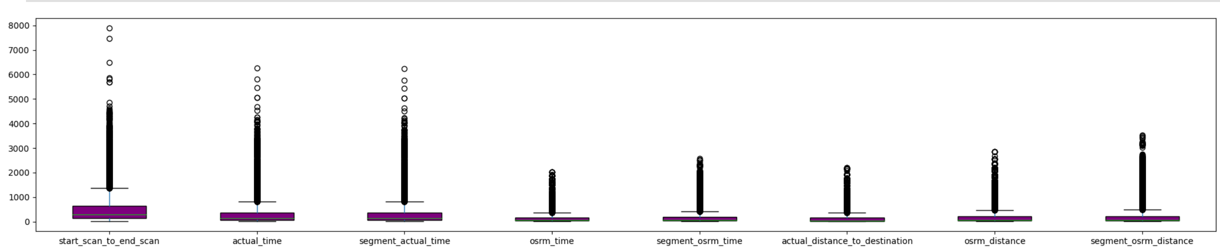
Out[ ]: <Axes: xlabel='data', ylabel='count'>



In [ ]:
```
num_cols=['start_scan_to_end_scan',
 'actual_time',
  'segment_actual_time',
 'osrm_time',
 'segment_osrm_time',
 'actual_distance_to_destination',
 'osrm_distance',
 'segment_osrm_distance']
```

Detecting Outliers

In [ ]:
```
plt.figure(figsize=(20,4))
df[num_cols].boxplot(grid=False,patch_artist=True, boxprops=dict(facecolor='
plt.tight_layout()
plt.show()
```



As we can see there are various outliers in all columns on upper side of data which means many a times the delivery time is too high but as they are same with osrm too so it is not a high concern because it may be a reason for more distance area too as indicated by distance outliers.

In [ ]:
```
q1=df[num_cols].quantile(0.25)
q3=df[num_cols].quantile(0.75)
upper_whisker=(q3+(q3-q1)*1.5)
upper_whisker
for i in range(len(num_cols)):

  outlier=df[df[num_cols[i]]>upper_whisker[i]]
# q1,q3,upper_whisker
  print(f'Outliers for column {num_cols[i]:<35} = {len(outlier)}-------> Thi
```

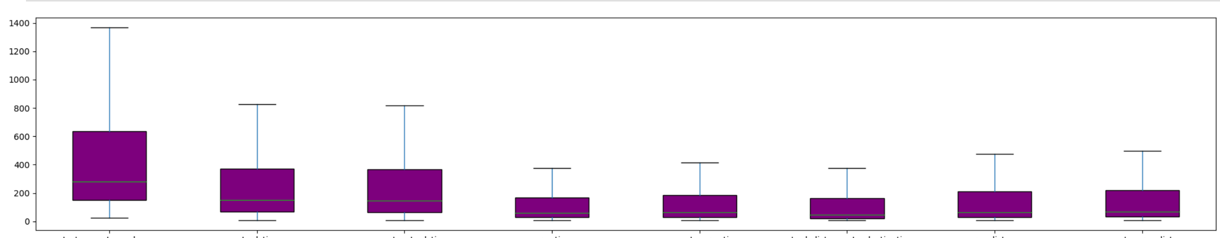
```
Outliers for column start_scan_to_end_scan              = 1267-------> This
is 8.55 % of dataset.
Outliers for column actual_time                         = 1643-------> This
is 11.09 % of dataset.
Outliers for column segment_actual_time                 = 1643-------> This
is 11.09 % of dataset.
Outliers for column osrm_time                           = 1517-------> This
is 10.24 % of dataset.
Outliers for column segment_osrm_time                   = 1492-------> This
is 10.07 % of dataset.
Outliers for column actual_distance_to_destination      = 1449-------> This
is 9.78 % of dataset.
Outliers for column osrm_distance                       = 1526-------> This
is 10.3 % of dataset.
Outliers for column segment_osrm_distance               = 1548-------> This
is 10.45 % of dataset.
```

We can see that outliers are too much approx 10 % of data so they are not outliers basically but the data itself as 10 % of data is not outliers but business only. It may be required to have long route deliveries. So I am not removing them in original data just showing it as different dataframe

Treating Outliers

In [ ]:
```
df_outliers=df[num_cols]
df_outliers[num_cols]=np.clip(df[num_cols],0,upper_whisker,axis=1)
```

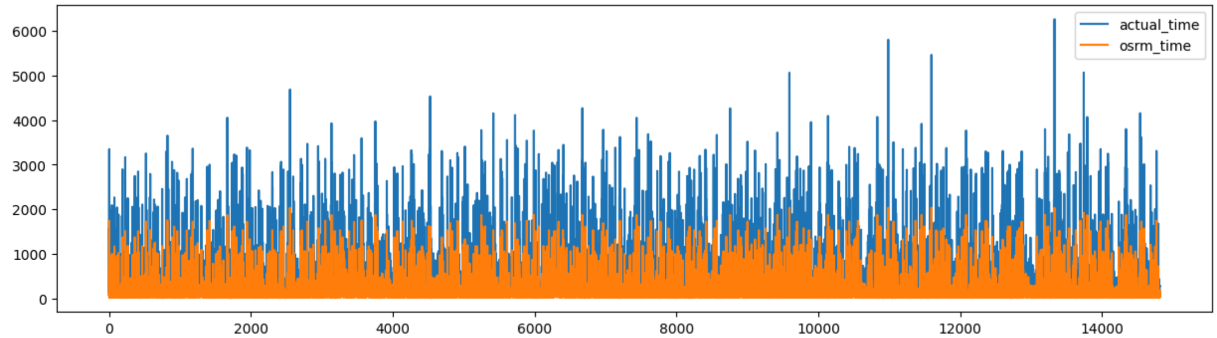
In [ ]:
```
plt.figure(figsize=(20,4))
df_outliers[num_cols].boxplot(grid=False,patch_artist=True, boxprops=dict(fa
plt.tight_layout()
plt.show()
```



Finding difference between actual and osrm time

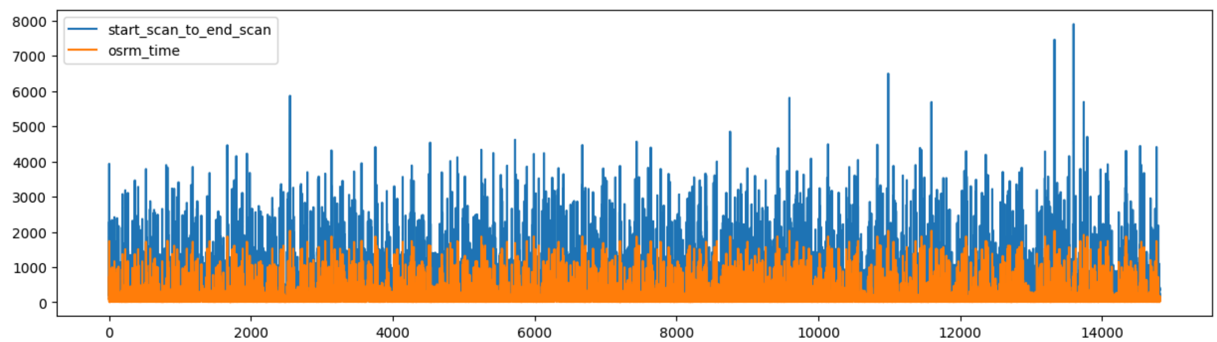
In [ ]:
```
plt.figure(figsize=(20,4))
ax=df[['actual_time','osrm_time']].plot(kind='line')
ax.figure.set_size_inches(15, 4)
plt.show()
```

<Figure size 2000x400 with 0 Axes>



In [ ]:
```
plt.figure(figsize=(20,4))
ax=df[['start_scan_to_end_scan','osrm_time']].plot(kind='line')
ax.figure.set_size_inches(15, 4)
plt.show()
```

<Figure size 2000x400 with 0 Axes>



We can see the actual time is much more than the osrm time which should not be the case for best logistics.
If we compare the start to scan time, the difference is more wider.

Finding routes where difference between actual and osrm time is more than 1 day

In [ ]:
```
# lets check  out the routes where such difference is high
df_lag=df[df['actual_time']>df['osrm_time']+1440][['source_state','source_ci
df_lag
# checking for more than 1 day difference only
```

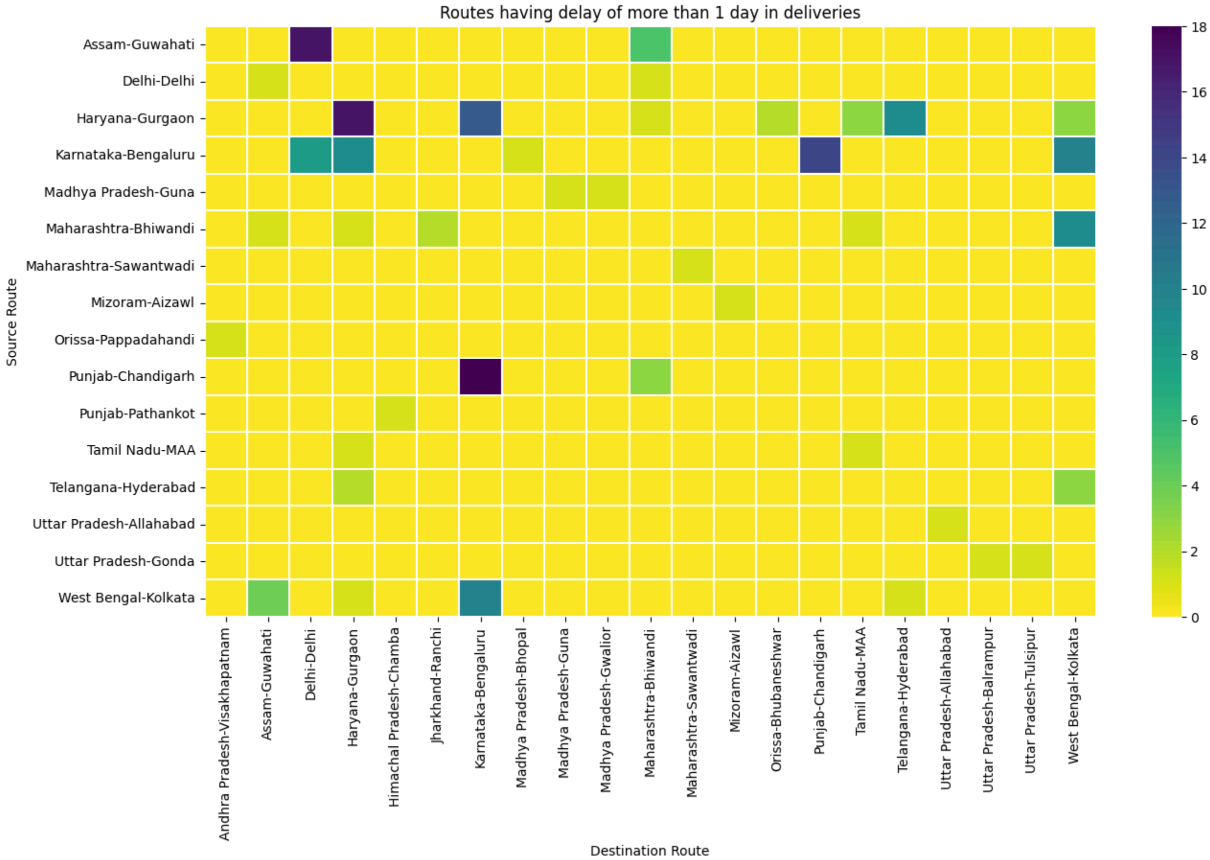
Out[ ]:

	index	source_state	source_city	destination_state	destination_city
0	2	Karnataka	Bengaluru	Punjab	Chandigarh
1	190	West Bengal	Kolkata	Karnataka	Bengaluru
2	228	Haryana	Gurgaon	Karnataka	Bengaluru
3	520	Punjab	Chandigarh	Karnataka	Bengaluru
4	805	West Bengal	Kolkata	Assam	Guwahati
...	...	...	...	...	...
176	14349	Maharashtra	Bhiwandi	West Bengal	Kolkata
177	14538	Haryana	Gurgaon	Karnataka	Bengaluru
178	14555	Punjab	Chandigarh	Karnataka	Bengaluru
179	14592	Karnataka	Bengaluru	Delhi	Delhi
180	14769	Karnataka	Bengaluru	Punjab	Chandigarh

181 rows × 5 columns

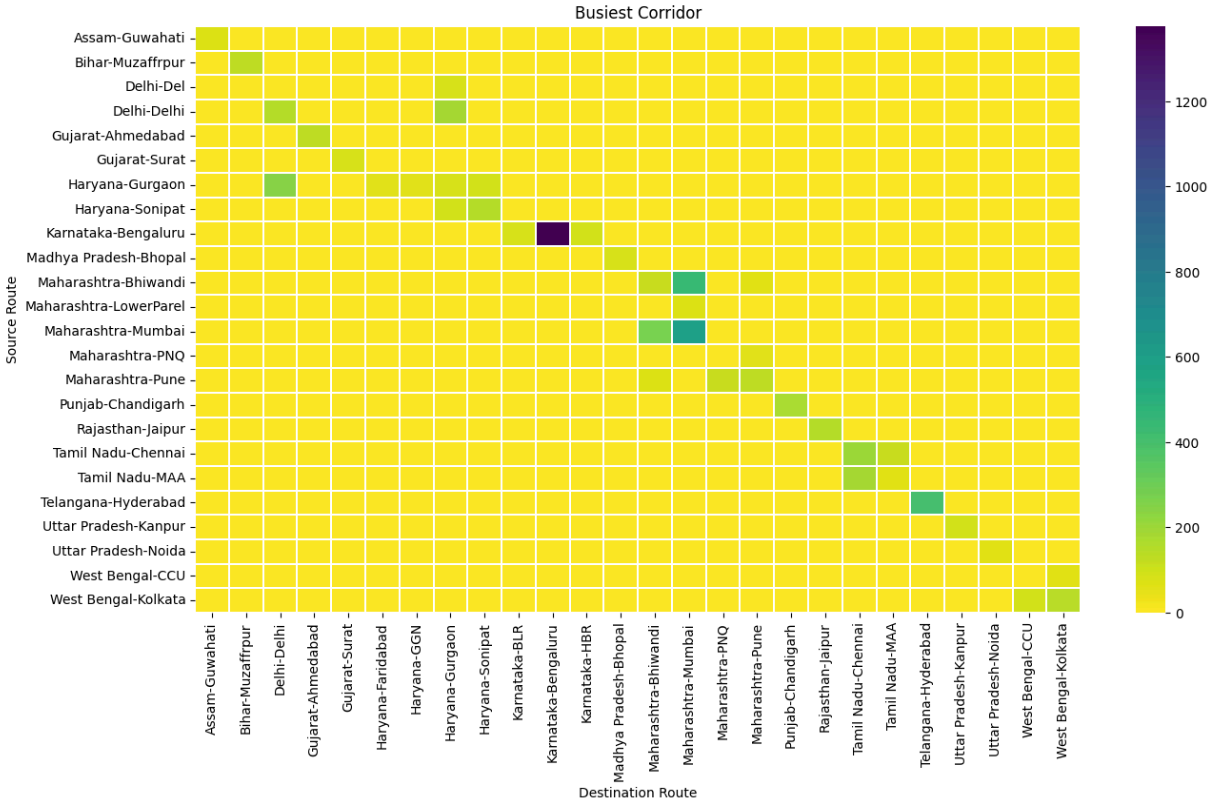
In [ ]:

```
df_cross=pd.crosstab(index=[df_lag['source_state'],df_lag['source_city']],cc
plt.figure(figsize=(15,8))
plt.title('Routes having delay of more than 1 day in deliveries')
sns.heatmap(df_cross,cmap='viridis_r',linewidths=0.01)
plt.ylabel("Source Route")
plt.xlabel("Destination Route")
plt.show()
```



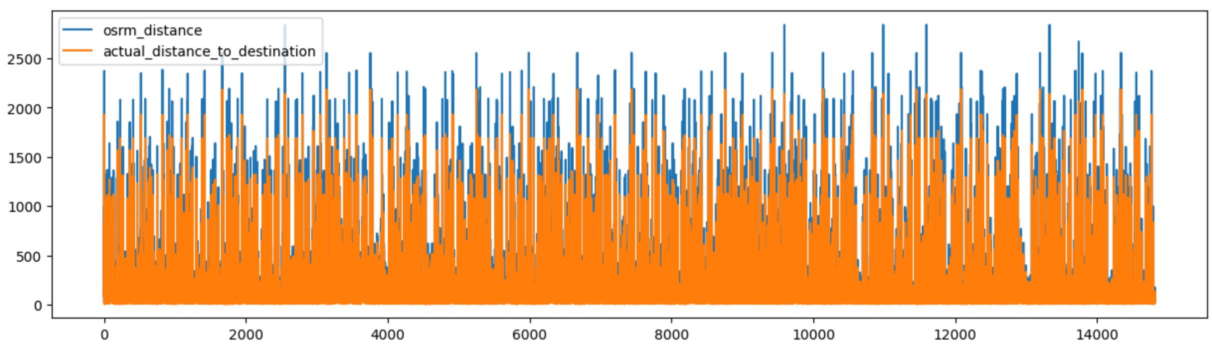
Finding busiest corridor

```
In [ ]: df_cross=df[['source_state','source_city','destination_state','destination_c
        df_cross1=df_cross[df_cross.values>50].reset_index()
        df_cross1=df_cross1.pivot_table(index=['source_state','source_city'],columns
        plt.figure(figsize=(15,8))
        plt.title('Busiest Corridor')
        sns.heatmap(df_cross1,cmap='viridis_r',linewidths=0.01)
        plt.ylabel("Source Route")
        plt.xlabel("Destination Route")
        plt.show()
```



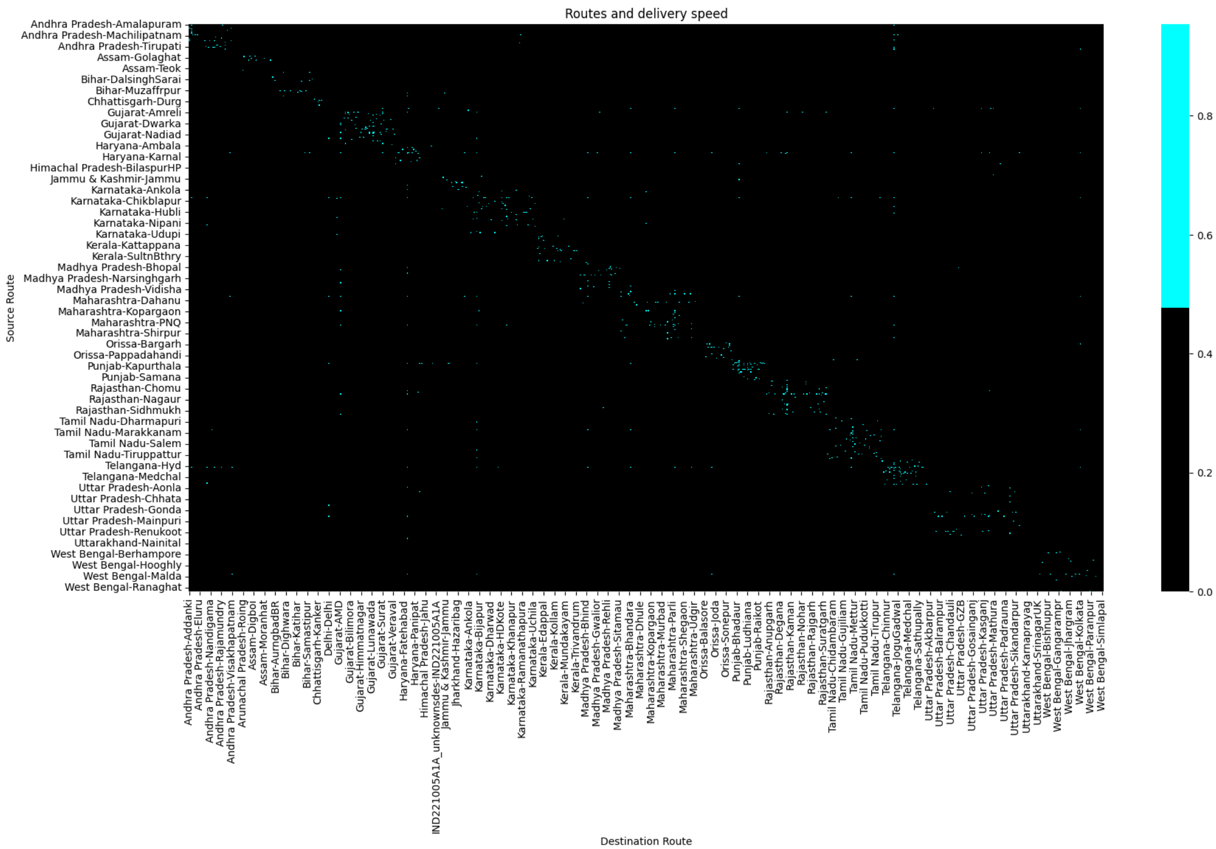

```
In [ ]: plt.figure(figsize=(20,4))
ax=df[['osrm_distance','actual_distance_to_destination']].plot(kind='line')
ax.figure.set_size_inches(15, 4)
plt.show()
```

<Figure size 2000x400 with 0 Axes>



This is good that actual distance the deliveries are taking are less than osrm distance as it may reduce cost but the time taken is still more is not right.

```
In [ ]: df1=df.groupby(['source_state','source_city','destination_state','destinatic
df1['speed']=df1['actual_distance_to_destination']/df1['actual_time']
df_cross1=df1.pivot_table(index=['source_state','source_city'],columns=['des
plt.figure(figsize=(20,10))
plt.title('Routes and delivery speed')
sns.heatmap(df_cross1,cmap=['black','cyan'])
plt.ylabel("Source Route")
plt.xlabel("Destination Route")
plt.show()
```



Hypothesis Testing

Computing the significant difference between actual_time -- osrm_time.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - actual_time is not greater than osrm_time (Expected total trip time).

Alternate Hypothesis (HA) - actual_time is greater than osrm_time (Expected total trip time).

STEP-2 : Setting up Confidence Level Lets assume the confidence level to be 95 % so our alpha will be **0.05**

STEP-3 : Choosing the distribution and test Statistics As we have only near to 20 days sample data, it doesnt makes sense to check normality of sample data but as we know almost every population distribution in world follows the normal distribution we are assuming it.
AS we dont know the population parameters doing testing with **T Statistics**

STEP-4 : Which Tail

We are trying to find the difference between 2 columns so assuming it to be right tail.

STEP-5 : Computing p value

In [ ]:
```
ttest,pval=ttest_ind(df['actual_time'],df['osrm_time'],alternative='greater'
print(f"P Value is {pval}\nAlpha is 0.05")
alpha=0.05
if pval<=alpha:
  print("We reject the null hypothesis and concludes that actual_time is sig
else:
  print("Fail to reject the null Hypothesis. actual_time is not statisticall
```

```
P Value is 0.0
Alpha is 0.05
We reject the null hypothesis and concludes that actual_time is significantl
y greater than osrm_time.
```

Computing the significant difference between actual_time -- segment_actual_time.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - actual_time is not statistically different than segment_actual_time.

Alternate Hypothesis (HA) - actual_time is statistically different than segment_actual_time..

STEP-2 : Setting up Confidence Level Lets assume the confidence level to be 95 % so our alpha will be **0.05**

STEP-3 : Choosing the distribution and test Statistics As we have only near to 20 days sample data, it doesnt makes sense to check normality of sample data but as we know almost every population distribution in world follows the normal distribution we are assuming it.
AS we dont know the population parameters doing testing with **T Statistics**

STEP-4 : Which Tail We are trying to find the difference between 2 columns so assuming it to be two tail test.

STEP-5 : Computing p value

In [ ]:
```
ttest,pval=ttest_ind(df['actual_time'],df['segment_actual_time'])
print(f"P Value is {'%1.4f'%pval}\nAlpha is 0.05")
```

```
alpha=0.05
if pval<=alpha:
  print("We reject the null hypothesis and concludes that actual_time is sig
else:
  print("Fail to reject the null Hypothesis. actual_time is not statisticall
```

```
P Value is 0.6284
Alpha is 0.05
Fail to reject the null Hypothesis. actual_time is not statistically differe
nt than segment_actual_time.
```

So, we can drop one of these two column to feed data into ML model.

Computing the significant difference between osrm_distance -- segment_osrm_distance.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - osrm_distance is not statistically different than segment_osrm_distance.

Alternate Hypothesis (HA) - osrm_distance is statistically different than segment_osrm_distance.

STEP-2 : Setting up Confidence Level Lets assume the confidence level to be 95 % so our alpha will be **0.05**

STEP-3 : Choosing the distribution and test Statistics As we have only near to 20 days sample data, it doesnt makes sense to check normality of sample data but as we know almost every population distribution in world follows the normal distribution we are assuming it.
AS we dont know the population parameters doing testing with **T Statistics**

STEP-4 : Which Tail We are trying to find the difference between 2 columns so assuming it to be two tail test.

STEP-5 : Computing p value

In [ ]:
```
tstat,pval=ttest_ind(df['osrm_distance'],df['segment_osrm_distance'])
print(f"P Value is {pval}\nAlpha is 0.05")
alpha=0.05
if pval<=alpha:
  print("We reject the null hypothesis and concludes that osrm_distance is s
else:
  print("Fail to reject the null Hypothesis. osrm_distance is not statistica
```

```
P Value is 7.840520928201551e-05
Alpha is 0.05
We reject the null hypothesis and concludes that osrm_distance is significan
tly different than segment_osrm_distance.
```

Computing the significant difference between osrm_time -- segment_osrm_time.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - osrm_time is not statistically different than segment_osrm_time.

Alternate Hypothesis (HA) - osrm_time is statistically different than segment_osrm_time..

STEP-2 : Setting up Confidence Level Lets assume the confidence level to be 95 % so our alpha will be **0.05**

STEP-3 : Choosing the distribution and test Statistics As we have only near to 20 days sample data, it doesnt makes sense to check normality of sample data but as we know almost every population distribution in world follows the normal distribution we are assuming it.
AS we dont know the population parameters doing testing with **T Statistics**

STEP-4 : Which Tail We are trying to find the difference between 2 columns so assuming it to be two tail test.

STEP-5 : Computing p value

In [ ]:
```
ttest_ind(df['osrm_time'],df['segment_osrm_time'])
print(f"P Value is {pval}\nAlpha is 0.05")
alpha=0.05
if pval<=alpha:
  print("We reject the null hypothesis and concludes that osrm_time is signi
else:
  print("Fail to reject the null Hypothesis. osrm_time is not statistically
```

```
P Value is 7.840520928201551e-05
Alpha is 0.05
We reject the null hypothesis and concludes that osrm_time is significantly
different than segment_osrm_time.
```

We saw that except segment_actual_time and actual_time, all other columns are statistically different. so we can drop any of these two for machine learning.

Feature engineering

In [ ]:
```
num_cols=(df.dtypes=='float')|(df.dtypes=='int')
num_cols=list(num_cols[num_cols].index)
num_cols
```

Out[ ]:
```
['start_scan_to_end_scan',
 'actual_distance_to_destination',
 'actual_time',
 'osrm_time',
 'osrm_distance',
 'segment_actual_time',
 'segment_osrm_time',
 'segment_osrm_distance']
```

In [ ]:
```
# converting numreic columns to standard scaling
df[num_cols]=StandardScaler().fit_transform(df[num_cols])
df.head()
```

Out[ ]:

	trip_uuid	data	route_type	od_start_time	od_end_time
0	trip-153671041653548748	training	FTL	2018-09-12 00:00:16.535741	2018-09-13 13:40:23.123744
1	trip-153671042288605164	training	Carting	2018-09-12 00:00:22.886430	2018-09-12 03:01:59.598855
2	trip-153671043369099517	training	FTL	2018-09-12 00:00:33.691250	2018-09-14 17:34:55.442454
3	trip-153671046011330457	training	Carting	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822
4	trip-153671052974046625	training	FTL	2018-09-12 00:02:09.740725	2018-09-12 12:00:30.683231

5 rows × 24 columns

In [ ]:
```
# performing one hot encoding on data and route type columns as they have 2
cat_cols=['data','route_type']
encoder = OneHotEncoder(sparse=False)

encoded_data=encoder.fit_transform(df[cat_cols])
# created one hot encoder data

encoded_df = pd.DataFrame(
    encoded_data,
    columns=encoder.get_feature_names_out(cat_cols))
# converted one hot encoded data with categories name as column name
df = pd.concat([df, encoded_df], axis=1)
# concating the original df with this encoded columns
df.drop(columns=['data','route_type'],inplace=True)
df.head()
```

Out[ ]:

	trip_uuid	od_start_time	od_end_time	start_scan_to_end_sca
0	trip-153671041653548748	2018-09-12 00:00:16.535741	2018-09-13 13:40:23.123744	2.62355
1	trip-153671042288605164	2018-09-12 00:00:22.886430	2018-09-12 03:01:59.598855	-0.53258
2	trip-153671043369099517	2018-09-12 00:00:33.691250	2018-09-14 17:34:55.442454	5.16486
3	trip-153671046011330457	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	-0.65403
4	trip-153671052974046625	2018-09-12 00:02:09.740725	2018-09-12 12:00:30.683231	0.28263

5 rows × 26 columns

*Business Insights and Analysis *

- 72% of data is Training data and 28% testing.
- 60% deliveries are carting based while 40 % is Full truck loading deliveries.
- Most orders went from and to Maharashtra followed by Karnataka and Haryana while the least from Nagaland, Mizoram and Arunachal Pradesh.
- The least deliveries were made to Tripura, Nagaland and Daman & Diu.
- If we see from city perspective, the Bengalauru followed by Gurgaon and Mumbai are cities from where maximum deliveries were sent.
- And they were sent most to Bengaluru followed by Mumbai and Gurgaon.
- if we talk about specific places most deliveries were out from Gurgaon-Bilaspur, Bengaluru- Nelmngla and Bhiwandi-Mankoli area and to same area too in terms of receipts of deliveries with little up and down.
- As we have data only from 12 sept, 2018 to 3 October, 2018 which is less than a month, we can't decide about some days but among the given data the most orders were made in September and comparatively less in starting days of October and ending days of September.
- Most trips were created during night hours between 8:00 PM to 1:00 AM as compared to day time.
- Most orders were made on Mondays while least on Sundays.
- Cant comment on month wise data but it seems a high possibility that there are less orders in month start (need more data to validate).
- The delay in deliveries are mainly at routes:

1. Chandigarh -- Bengaluru
2. Guwahati -- Delhi
3. Gurgaon -- Gurgaon
4. Gurgaon -- Bengaluru
5. Kolkata -- Bengaluru

- We can see the busiest corridors are within a state and city itself which means that we are lacking with intra city and intra state deliveries as they are taking more times than osme too.
- The highest speed of delivery is for destination Delhi, Hyderabad, Telangana and Ahmedabad

Recommendations

- Revisit information fed to routing engine for trip planning. Check for discrepancies with transporters, if the routing engine is configured for optimum results. If it is working fine, there is a possibility that drivers taking shorter distance to reduce cost but it is resulting in more time due to inferiore roads etc. So, need to find and work on that.
- Actual time taking in deliveries are very high. The reason can be the more resting period and no enough transports available. We must ensure proper transpor tfacility with 2 drivers at lon routes so that the deliveries can be done without delay at high speed.
- osrm_time and actual_time are different. Team needs to make sure this difference is reduced, so that better delivery time prediction can be made and it becomes convenient for the customer to expect an accurate delivery time. to increase customer satisfaction.
- Only 1504 unique route ids shows deliveries are repeated at same routes as high as 1812 times to a single route in a period of less than 22 days is surely a positive sign.
- Need to ensure more speedy deivery in those routes like Bengaluru to Chandigarh, Kolkata, Gurgaon etc by ensuring free loaded trucks to be sent directly rather than stopping in between.
- Same trip_uuid is repeated 101 times shows its not a good sign to send so many deliveries in between as will lead to late deliveries, try to avoid these kind of carting trucks as can lead to lose of customers.
- We have seen that most of the deliveries are within same state and cities too where speed of delivery is high but less than expected time. so need to ensure local area deliveries via two wheeleres riders etc.
- We should attract more customers in routes where we have not much high deliveries like from Orrisa, Uttarakhand, Jharkhand etc to ensure better coverage of area and as these are mid routes having more deliveries from here can also lead to more flt to nearby routes and more speedy deliveries.