Delhivery_analysis

December 5, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from datetime import datetime
import scipy.stats as stats

%matplotlib inline
```

1. Why 2 rows with almost same actual data have different estimated values?
2. What we will do with null values (Added expected values)
3. Should we create some other features from current data?(Added some - More can be done)

[2]:
```
data = pd.read_csv('./delhivery_data.csv', encoding = 'utf8')
data.head()
```

[2]:
```
       data          trip_creation_time  \
0  training  2018-09-20 02:35:36.476840
1  training  2018-09-20 02:35:36.476840
2  training  2018-09-20 02:35:36.476840
3  training  2018-09-20 02:35:36.476840
4  training  2018-09-20 02:35:36.476840

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

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

  destination_center              destination_name  \
```

```
0       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
1       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
2       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
3       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)
4       IND388620AAB  Khambhat_MotvdDPP_D (Gujarat)

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

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

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

   segment_factor
0        1.272727
1        1.111111
2        2.285714
3        1.750000
4        1.200000

[5 rows x 24 columns]
```

[3]:
```
data.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
```

[4]:
```
data.describe()
```

[4]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination
count	144867.000000	144867.000000	144867.000000
mean	961.262986	232.926567	234.073372
std	1037.012769	344.755577	344.990009
min	20.000000	9.000000	9.000045
25%	161.000000	22.000000	23.355874
50%	449.000000	66.000000	66.126571
75%	1634.000000	286.000000	286.708875
max	7898.000000	1927.000000	1927.447705

	actual_time	osrm_time	osrm_distance	factor
count	144867.000000	144867.000000	144867.000000	144867.000000
mean	416.927527	213.868272	284.771297	2.120107
std	598.103621	308.011085	421.119294	1.715421
min	9.000000	6.000000	9.008200	0.144000
25%	51.000000	27.000000	29.914700	1.604264
50%	132.000000	64.000000	78.525800	1.857143
75%	513.000000	257.000000	343.193250	2.213483
max	4532.000000	1686.000000	2326.199100	77.387097

	segment_actual_time	segment_osrm_time	segment_osrm_distance
count	144867.000000	144867.000000	144867.00000
mean	36.196111	18.507548	22.82902

```
std              53.571158          14.775960              17.86066
min            -244.000000           0.000000               0.00000
25%              20.000000          11.000000              12.07010
50%              29.000000          17.000000              23.51300
75%              40.000000          22.000000              27.81325
max            3051.000000        1611.000000            2191.40370

       segment_factor
count   144867.000000
mean         2.218368
std          4.847530
min        -23.444444
25%          1.347826
50%          1.684211
75%          2.250000
max        574.250000
```

[5]:
```
data.nunique()
```

[5]:
```
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
```

[6]:
```
data_for_single_trip = data[data['trip_uuid'] == 'trip-153741093647649320']
```

```
data_for_single_trip[['source_center', 'destination_center', 'od_start_time',␣
 ↪'od_end_time']]
```

[6]:

	source_center	destination_center	od_start_time	od_end_time
0	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797
1	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797
2	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797
3	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797
4	IND388121AAA	IND388620AAB	2018-09-20 03:21:32.418600	2018-09-20 04:47:45.236797
5	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	2018-09-20 06:36:55.627764
6	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	2018-09-20 06:36:55.627764
7	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	2018-09-20 06:36:55.627764
8	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	2018-09-20 06:36:55.627764
9	IND388620AAB	IND388320AAA	2018-09-20 04:47:45.236797	2018-09-20 06:36:55.627764

So for a single trip we are observing that all values are same for rows 0-4 and also same for rows 5-9. So we will check for each suspected rows if all values in the row are same or not

[7]:
```
row0 = data_for_single_trip.loc[0]
row1 = data_for_single_trip.loc[1]
row2 = data_for_single_trip.loc[2]
row3 = data_for_single_trip.loc[3]
row4 = data_for_single_trip.loc[4]

are_rows_equal = (row1 == row2).all()
print(are_rows_equal)
```

```
False
```

Sow row1 and row2 does not have all values equal. Lets check where the difference lies.

[8]:
```
print(row0.compare(row1))
print(row2.is_cutoff)
print(row1.is_cutoff)
```

```
                                              self                other
cutoff_factor                                    9                   18
```

cutoff_timestamp	2018-09-20 04:27:55	2018-09-20 04:17:55
actual_distance_to_destination	10.43566	18.936842
actual_time	14.0	24.0
osrm_time	11.0	20.0
osrm_distance	11.9653	21.7243
factor	1.272727	1.2
segment_actual_time	14.0	10.0
segment_osrm_time	11.0	9.0
segment_osrm_distance	11.9653	9.759
segment_factor	1.272727	1.111111

True

True

So we are obeserving that 5 rows have almost same data but have difference in estimated info. Ex OSRM distance, osrm time etc etc. We need to investigate why this is coming as different for 2 entries which look exactly about same. And why I am saying that 2 entries look almost exactly same is because they have exactly same od_start_time and od_end_time which suggest they should be a single instance

One observation is that is_cutoff is always true for the last row for a single trip_uuid.

[9]:
```
data.isnull().sum()
```

[9]:
```
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
```

[10]:
```
train_df = data[data['data'] == 'training']
test_df = data[data['data'] == 'test']
```

[11]:
```
print(train_df.info())
print(train_df.describe())
print(train_df.nunique())
print(train_df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 104858 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            104858 non-null  object
 1   trip_creation_time              104858 non-null  object
 2   route_schedule_uuid             104858 non-null  object
 3   route_type                      104858 non-null  object
 4   trip_uuid                       104858 non-null  object
 5   source_center                   104858 non-null  object
 6   source_name                     104729 non-null  object
 7   destination_center              104858 non-null  object
 8   destination_name                104758 non-null  object
 9   od_start_time                   104858 non-null  object
 10  od_end_time                     104858 non-null  object
 11  start_scan_to_end_scan          104858 non-null  float64
 12  is_cutoff                       104858 non-null  bool
 13  cutoff_factor                   104858 non-null  int64
 14  cutoff_timestamp                104858 non-null  object
 15  actual_distance_to_destination  104858 non-null  float64
 16  actual_time                     104858 non-null  float64
 17  osrm_time                       104858 non-null  float64
 18  osrm_distance                   104858 non-null  float64
 19  factor                          104858 non-null  float64
 20  segment_actual_time             104858 non-null  float64
 21  segment_osrm_time               104858 non-null  float64
 22  segment_osrm_distance           104858 non-null  float64
 23  segment_factor                  104858 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 19.3+ MB
None
       start_scan_to_end_scan  cutoff_factor  actual_distance_to_destination  \
count           104858.000000  104858.000000                   104858.000000
mean               944.954119     229.039167                      230.286725
std               1017.591095     339.250432                      339.500527
min                 20.000000       9.000000                        9.000267
25%                165.000000      22.000000                       23.702352
50%                453.000000      66.000000                       66.383466
75%               1579.000000     266.000000                      270.843536
```

```
max               4535.000000    1927.000000                    1927.447705

         actual_time      osrm_time  osrm_distance         factor  \
count  104858.000000  104858.000000  104858.000000  104858.000000
mean      410.314263     210.743444     280.215620       2.112316
std       587.562589     303.417187     414.804815       1.643065
min         9.000000       6.000000       9.008200       0.144000
25%        53.000000      28.000000      30.558325       1.604167
50%       134.000000      66.000000      79.847700       1.857143
75%       498.000000     250.000000     330.617850       2.214286
max      4532.000000    1686.000000    2326.199100      77.387097

       segment_actual_time  segment_osrm_time  segment_osrm_distance  \
count        104858.000000      104858.000000          104858.000000
mean             36.393952          18.631511              22.955750
std              52.651730          15.039765              18.299019
min            -244.000000           0.000000               0.000000
25%              20.000000          11.000000              12.269725
50%              29.000000          17.000000              23.576950
75%              41.000000          23.000000              27.939650
max            3051.000000        1611.000000            2191.403700

       segment_factor
count   104858.000000
mean         2.211386
std          4.974192
min        -23.444444
25%          1.347826
50%          1.681818
75%          2.250000
max        574.250000
data                                1
trip_creation_time              10654
route_schedule_uuid              1385
route_type                          2
trip_uuid                       10654
source_center                    1425
source_name                      1417
destination_center               1409
destination_name                 1399
od_start_time                   18948
od_end_time                     18948
start_scan_to_end_scan           1681
is_cutoff                           2
cutoff_factor                     480
cutoff_timestamp                66370
actual_distance_to_destination  104555
actual_time                      3080
```

```
osrm_time                          1520
osrm_distance                    100770
factor                            36029
segment_actual_time                 648
segment_osrm_time                   192
segment_osrm_distance             87268
segment_factor                     4940
dtype: int64
data                                  0
trip_creation_time                    0
route_schedule_uuid                   0
route_type                            0
trip_uuid                             0
source_center                         0
source_name                         129
destination_center                    0
destination_name                    100
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
```

First lets deal with Null values in training data (train_df). I am creating a mapping of source_center vs source_name and destination_center vs destination_name. This mapping will help me estimate values of source_name and destination_name where these values are null.

[12]:
```
#  can use the statement below to replace NA values
# dbc_train.fillna('', inplace = True)

source_codes = train_df['source_center']
source_names = train_df['source_name']

destination_codes = train_df['destination_center']
destination_names = train_df['destination_name']

codeToNameMapping_Sources = dict(zip(source_codes, source_names))
```

```
codeToNameMapping_Destinations = dict(zip(destination_codes, destination_names))

code_to_name_mapping = codeToNameMapping_Sources.copy()
code_to_name_mapping.update(codeToNameMapping_Destinations)
```

[13]:
```
list_null_names = []
for key in code_to_name_mapping:
    value = code_to_name_mapping[key]
    if(isinstance(value, float) and np.isnan(value)):
        list_null_names.append(key)

# list_null_names contains all the source_centres/destination_centres which␣
 ↪have null source_names/destination_centres
```

Analyzing over the mapping of centre-to-name suggest that for every first 6 alphabets of centre code, the state is unique. For example, every entry in the mapping which starts with 'IND342' will be having a single state in each value of mapping

Example, if key starts with 'IND342', all these values are present and state is common for all : IND342005AAD Jodhpur_Basni_I (Rajasthan) IND342601AAA Piparcity_BsstdDPP_D (Rajasthan) IND342902A1B nan IND342602AAC Bilara_Central_DPP_2 (Rajasthan) IND342301AAA Phalodi_PalikDPP_D (Rajasthan) IND342902AAA Gotan_DKLogDPP_D (Rajasthan)

So this observation can be used to infer state for the null values. Lets replace the null values with their respective states:

[14]:
```
def findStateName(x):
    x = str(x)
    startingIdx = 0
    endingIdx = 0
    for i in range(len(x)):
        if(x[i] == '('):
            startingIdx = i
        if(x[i] == ')'):
            endingIdx = i
    return x[startingIdx+1:endingIdx]


def findCityName(x):
    x = str(x)
    endingIdx = 0
    for i in range(len(x)):
        if(x[i] == ' '):
            endingIdx = i
            break
        if(x[i] == '_'):
            endingIdx = i
            break
```

```
    return x[0:endingIdx]

# dbc_train['source_state'] = list(map(findStateName, dbc_train['source_name']))
# dbc_train['destination_state'] = list(map(findStateName,␣
 ↪dbc_train['destination_name']))
```

[15]:
```
temp_data = train_df[train_df['source_name'].isnull()]
for index, row in temp_data.iterrows():
    source_code = row['source_center']
    for key in code_to_name_mapping:
        if(key.startswith(source_code[0:6])):
            stateName = findStateName(code_to_name_mapping[key])
            train_df.loc[index, 'source_name'] = '(' + stateName + ')'

temp_data = train_df[train_df['destination_name'].isnull()]
for index, row in temp_data.iterrows():
    destination_code = row['destination_center']
    for key in code_to_name_mapping:
        if(key.startswith(destination_code[0:6])):
            stateName = findStateName(code_to_name_mapping[key])
            train_df.loc[index, 'destination_name'] = '(' + stateName + ')'


print(train_df.isnull().sum())
```

```
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
```

We can see there are no null values present now for source_name and destination_name and all are having atleast state name.

Lets create 2 new features from the older ones (lets add 4 new columns source_state, source_city, destination_state and destination_city)

[16]:
```
train_df['source_state'] = train_df['source_name'].apply(lambda x:␣
 ↪findStateName(x))
train_df['destination_state'] = train_df['destination_name'].apply(lambda x:␣
 ↪findStateName(x))
train_df['source_city'] = train_df['source_name'].apply(lambda x:␣
 ↪findCityName(x))
train_df['destination_city'] = train_df['destination_name'].apply(lambda x:␣
 ↪findCityName(x))

train_df.info()
```

```
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1157035015.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['source_state'] = train_df['source_name'].apply(lambda x:
findStateName(x))
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1157035015.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['destination_state'] = train_df['destination_name'].apply(lambda x:
findStateName(x))
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1157035015.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['source_city'] = train_df['source_name'].apply(lambda x:
findCityName(x))
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 104858 entries, 0 to 144866
Data columns (total 28 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            104858 non-null  object
 1   trip_creation_time              104858 non-null  object
 2   route_schedule_uuid             104858 non-null  object
 3   route_type                      104858 non-null  object
 4   trip_uuid                       104858 non-null  object
 5   source_center                   104858 non-null  object
 6   source_name                     104858 non-null  object
 7   destination_center              104858 non-null  object
 8   destination_name                104858 non-null  object
 9   od_start_time                   104858 non-null  object
 10  od_end_time                     104858 non-null  object
 11  start_scan_to_end_scan          104858 non-null  float64
 12  is_cutoff                       104858 non-null  bool
 13  cutoff_factor                   104858 non-null  int64
 14  cutoff_timestamp                104858 non-null  object
 15  actual_distance_to_destination  104858 non-null  float64
 16  actual_time                     104858 non-null  float64
 17  osrm_time                       104858 non-null  float64
 18  osrm_distance                   104858 non-null  float64
 19  factor                          104858 non-null  float64
 20  segment_actual_time             104858 non-null  float64
 21  segment_osrm_time               104858 non-null  float64
 22  segment_osrm_distance           104858 non-null  float64
 23  segment_factor                  104858 non-null  float64
 24  source_state                    104858 non-null  object
 25  destination_state               104858 non-null  object
 26  source_city                     104858 non-null  object
 27  destination_city                104858 non-null  object
dtypes: bool(1), float64(10), int64(1), object(16)
memory usage: 26.5+ MB
```

```
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1157035015.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['destination_city'] = train_df['destination_name'].apply(lambda x:
findCityName(x))
```

Now we have added 4 new features.

Lets check that if each unique row (uniqe with respect to ['trip_uuid','od_start_time',

'od_end_time']) contains a row with is_cutoff as False.

[17]:
```
df_train_unique = train_df.groupby(['trip_uuid','od_start_time', 'od_end_time'])
print(len(df_train_unique))
print(len(train_df[train_df.is_cutoff == False]))

# So we can see that for every unique value of trip_uuid it is not the case␣
 ↪that it have atleast one row with is_cutoff as False
```

```
18948
18808
```

So there are some entries which does not have any 'is_cutoff' that is False. But one observation is seen that every segment has maximum only 1 value of 'is_cutoff' as False and all others are always true

since at this point of time is_cutoff value is not making any point, I will be dropping cutoff fields is_cutoff, cutoff_factor, cutoff_timestamp and also dropping off factor and segment_factor

[18]:
```
columns_to_drop = ['is_cutoff', 'cutoff_factor', 'cutoff_timestamp', 'factor',␣
 ↪'segment_factor']
for col in columns_to_drop:
    del train_df[col]
print(train_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 104858 entries, 0 to 144866
Data columns (total 23 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            104858 non-null  object
 1   trip_creation_time              104858 non-null  object
 2   route_schedule_uuid             104858 non-null  object
 3   route_type                      104858 non-null  object
 4   trip_uuid                       104858 non-null  object
 5   source_center                   104858 non-null  object
 6   source_name                     104858 non-null  object
 7   destination_center              104858 non-null  object
 8   destination_name                104858 non-null  object
 9   od_start_time                   104858 non-null  object
 10  od_end_time                     104858 non-null  object
 11  start_scan_to_end_scan          104858 non-null  float64
 12  actual_distance_to_destination  104858 non-null  float64
 13  actual_time                     104858 non-null  float64
 14  osrm_time                       104858 non-null  float64
 15  osrm_distance                   104858 non-null  float64
 16  segment_actual_time             104858 non-null  float64
 17  segment_osrm_time               104858 non-null  float64
 18  segment_osrm_distance           104858 non-null  float64
 19  source_state                    104858 non-null  object
```

```
 20  destination_state               104858 non-null  object
 21  source_city                     104858 non-null  object
 22  destination_city                104858 non-null  object
dtypes: float64(8), object(15)
memory usage: 23.2+ MB
None
```

Lets create columns for trip creation year, month, date, time

[19]:
```
datetime_object = "%Y-%m-%d %H:%M:%S.%f"
conversion_array = list(map(lambda x: datetime.strptime(x, datetime_object) ,␣
 ↪train_df['trip_creation_time']))
train_df['trip_creation_year'] = list(map(lambda x: x.year, conversion_array))
train_df['trip_creation_month'] = list(map(lambda x: x.month, conversion_array))
train_df['trip_creation_day'] = list(map(lambda x: x.day, conversion_array))
```

```
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1917433496.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['trip_creation_year'] = list(map(lambda x: x.year, conversion_array))
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1917433496.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['trip_creation_month'] = list(map(lambda x: x.month,
conversion_array))
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/1917433496.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['trip_creation_day'] = list(map(lambda x: x.day, conversion_array))
```

For further analysis, we would be requiring an extra feature, that is segment_actual_distance which will be needed to extracted from actual_distance_to_destinations:

[20]:
```
distance_array = []
for index, row in train_df.iterrows():
    segment_actual_distance = 0
    is_same_segment = index-1 in train_df.index and (
```

```
            row['trip_uuid'] == train_df.loc[index-1,'trip_uuid']
            and row['od_start_time'] == train_df.loc[index-1,'od_start_time']
            and row['od_end_time'] == train_df.loc[index-1,'od_end_time']
        )
    if(index == 0 or not(is_same_segment)):
        segment_actual_distance = row['actual_distance_to_destination']
    else:
        segment_actual_distance = row['actual_distance_to_destination'] -␣
 ↪train_df.loc[index-1,'actual_distance_to_destination']
    distance_array.append(segment_actual_distance)
train_df['segment_actual_distance'] = distance_array
```

```
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/3073222032.py:14
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['segment_actual_distance'] = distance_array
```

Lets now treat the categorical column route_type using a label encoder from scikit learn

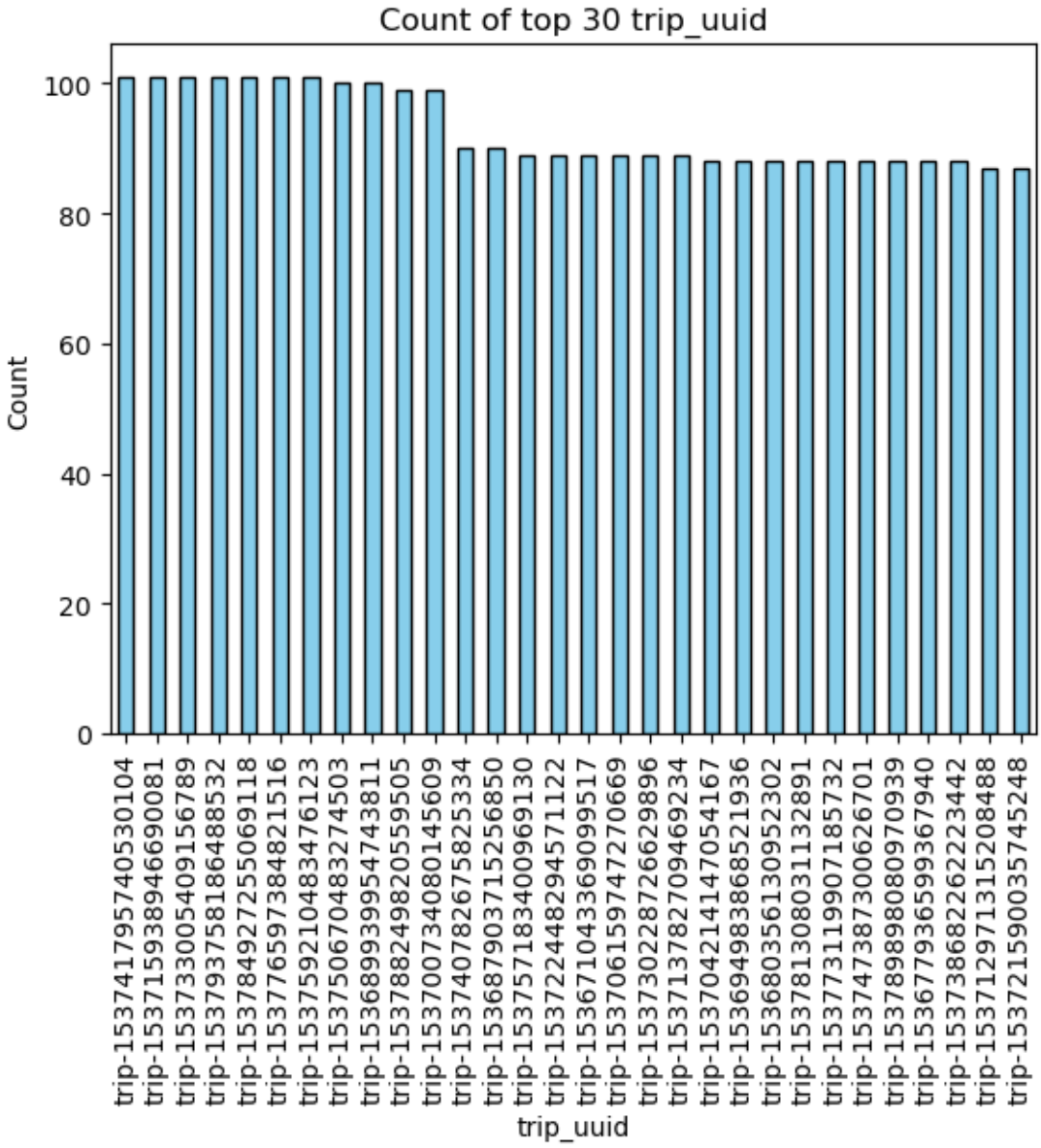
[21]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
train_df['route_type_encoded'] = le.fit_transform(train_df['route_type'])
```

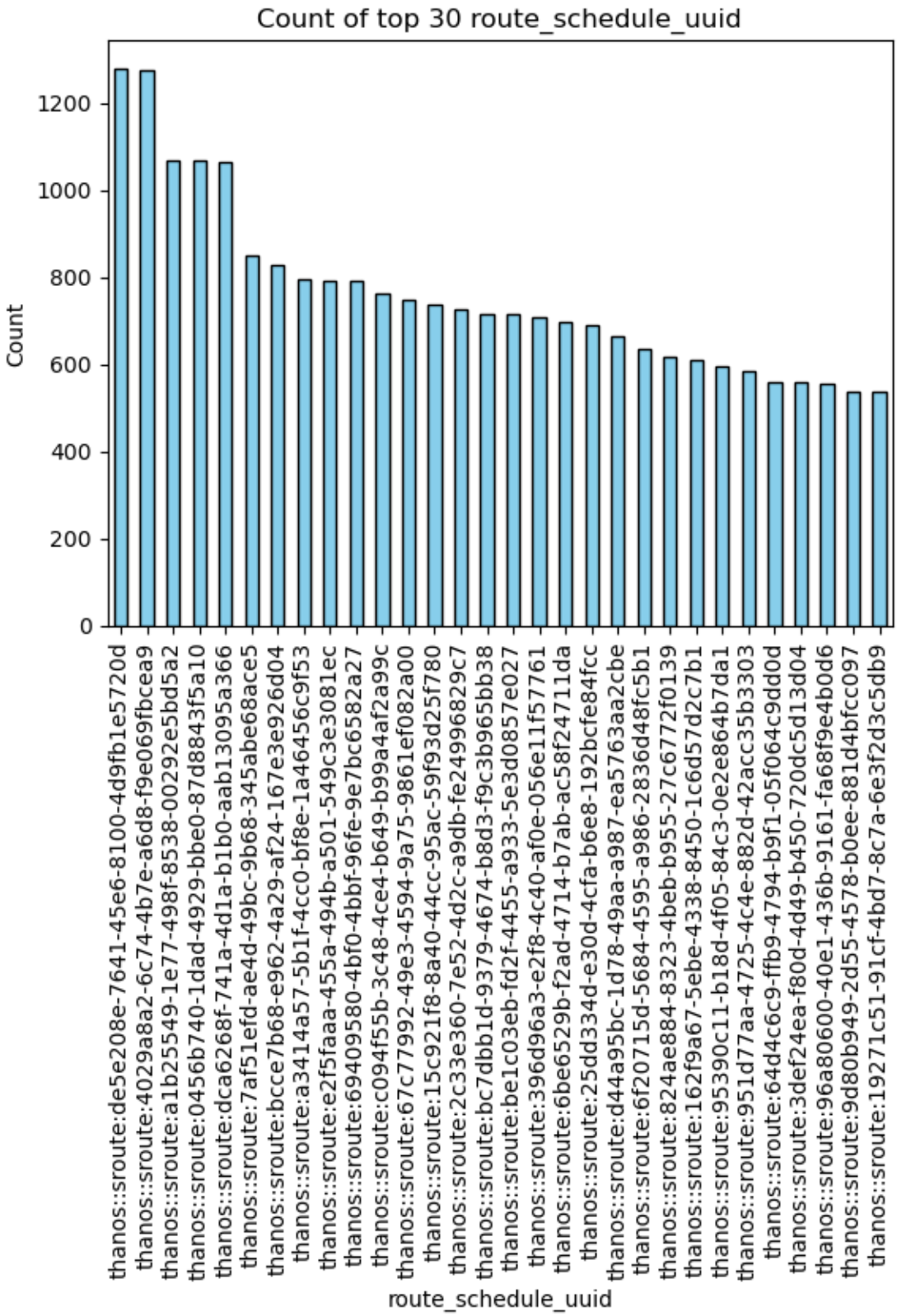
```
/var/folders/0y/1vh1g37j61j0kv4hcjwfgm2h0000gn/T/ipykernel_1874/2730868925.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['route_type_encoded'] = le.fit_transform(train_df['route_type'])
```

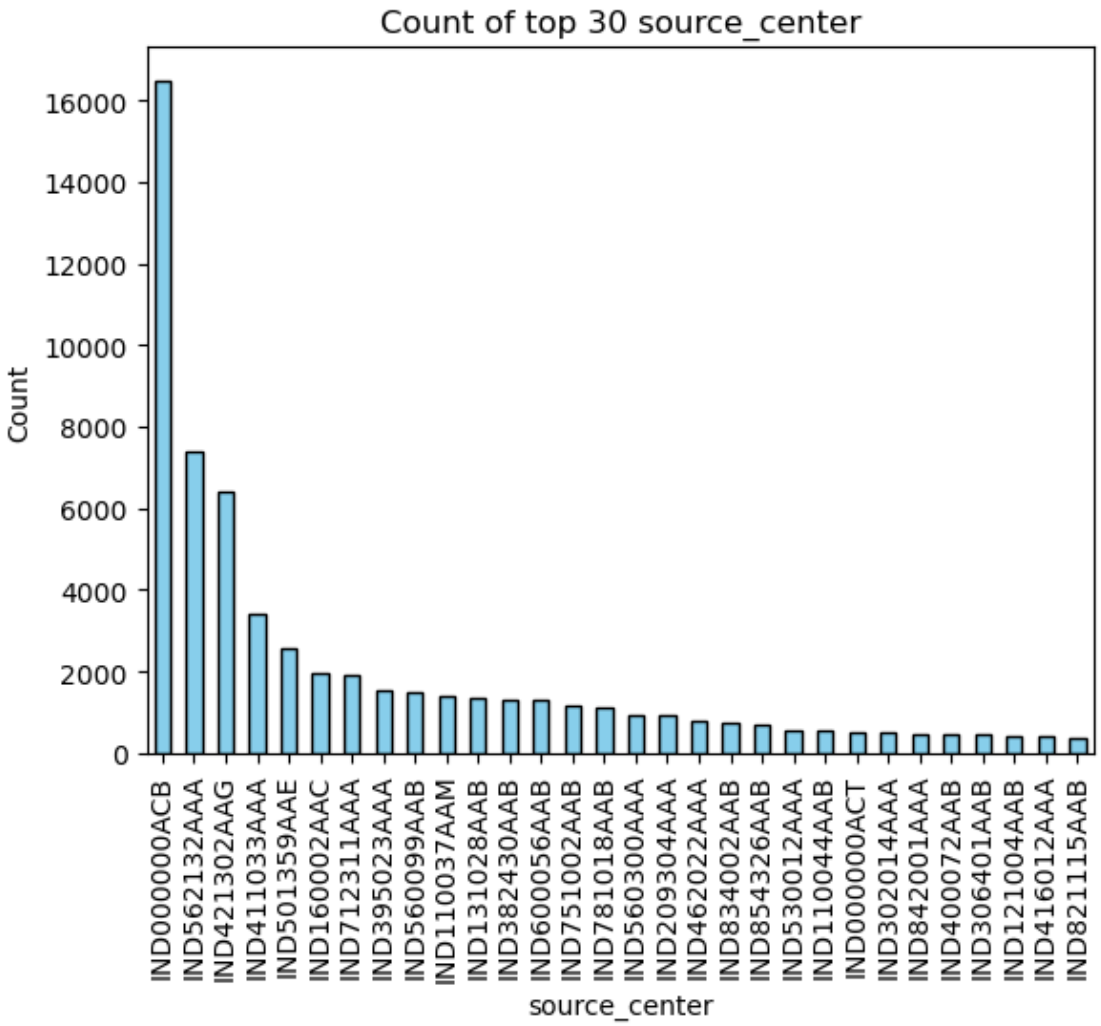
Lets build the distribution plots:

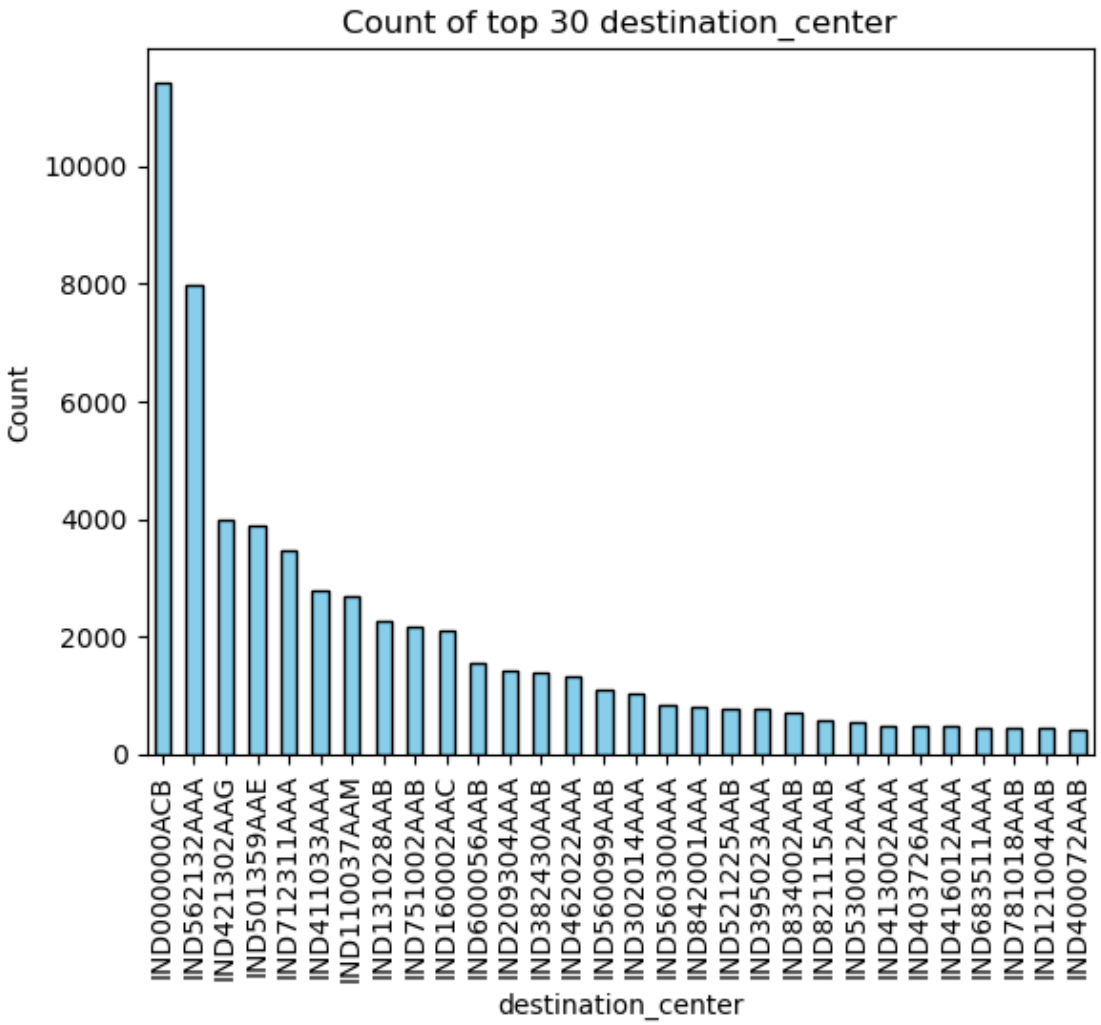
[22]:
```
count_elements = ['trip_uuid','route_schedule_uuid', 'source_center',␣
 ↪'destination_center', 'source_city', 'destination_city',
                  'source_state', 'destination_state', 'trip_creation_day' ,␣
 ↪'trip_creation_month', 'trip_creation_year', 'route_type']
for element in count_elements:
    element_counts = train_df[element].value_counts().head(30)
    element_counts.plot(kind='bar', color='skyblue', edgecolor='black')
    # Add labels and title
    plt.xlabel(element)
    plt.ylabel('Count')
    plt.title('Count of top 30 ' + element)
```

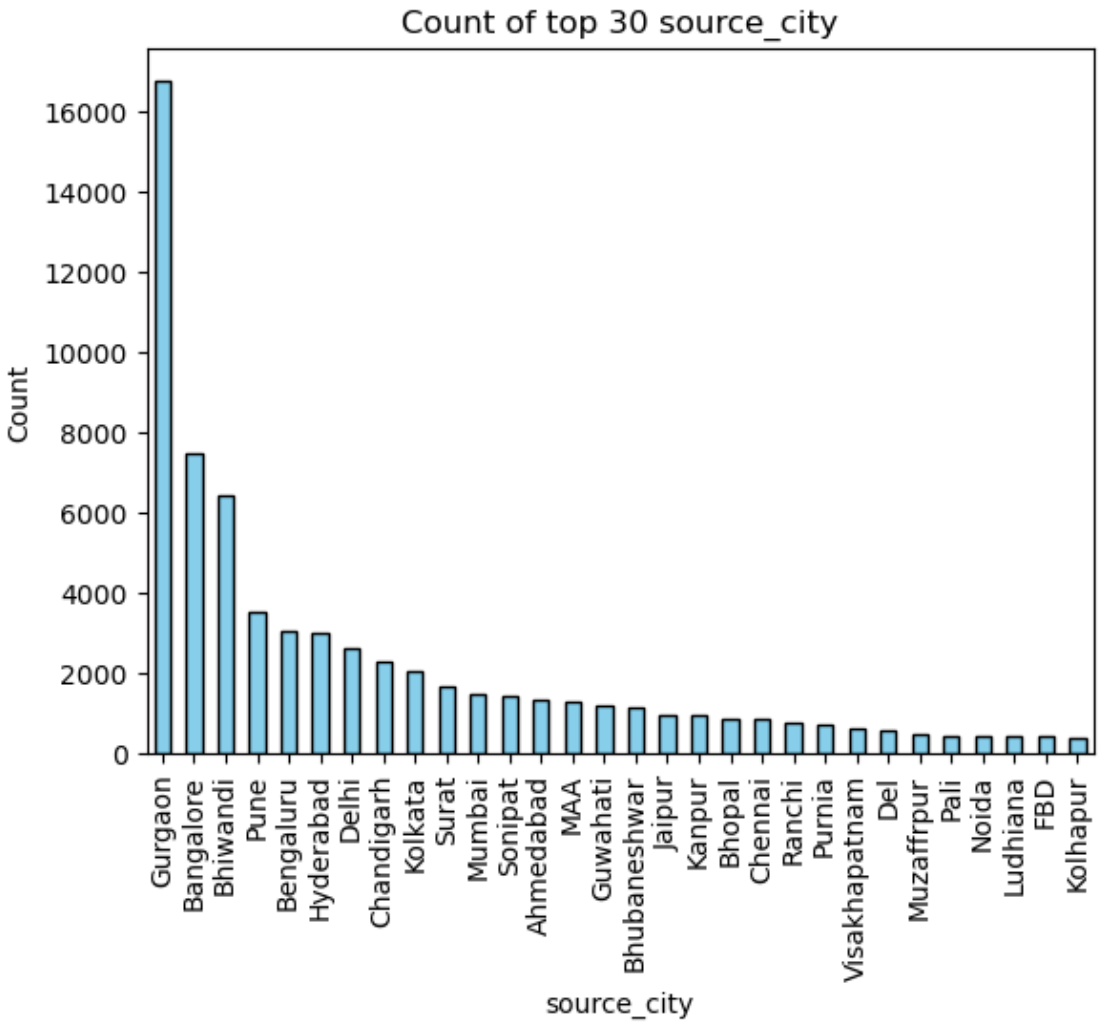
```
    # Display the plot
    plt.show()
```

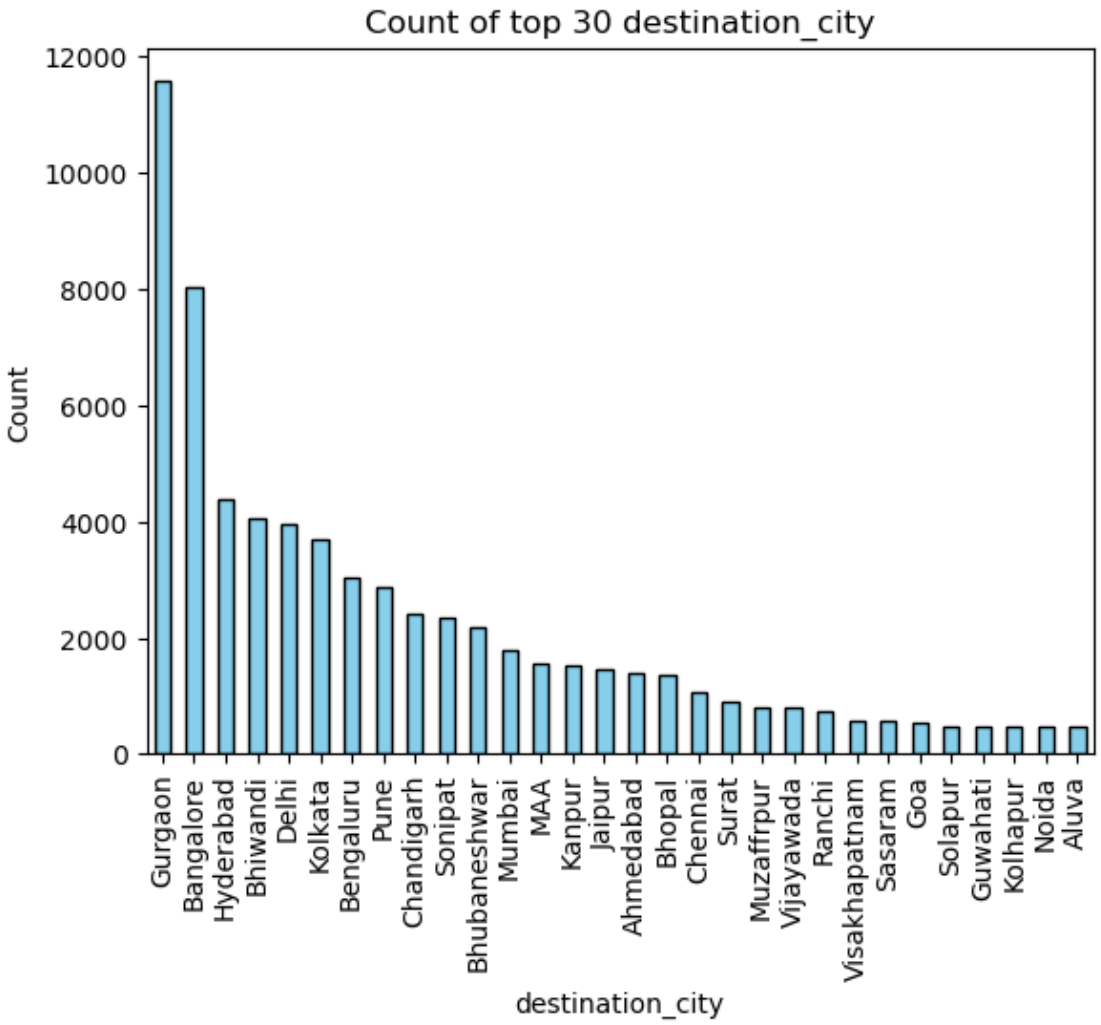


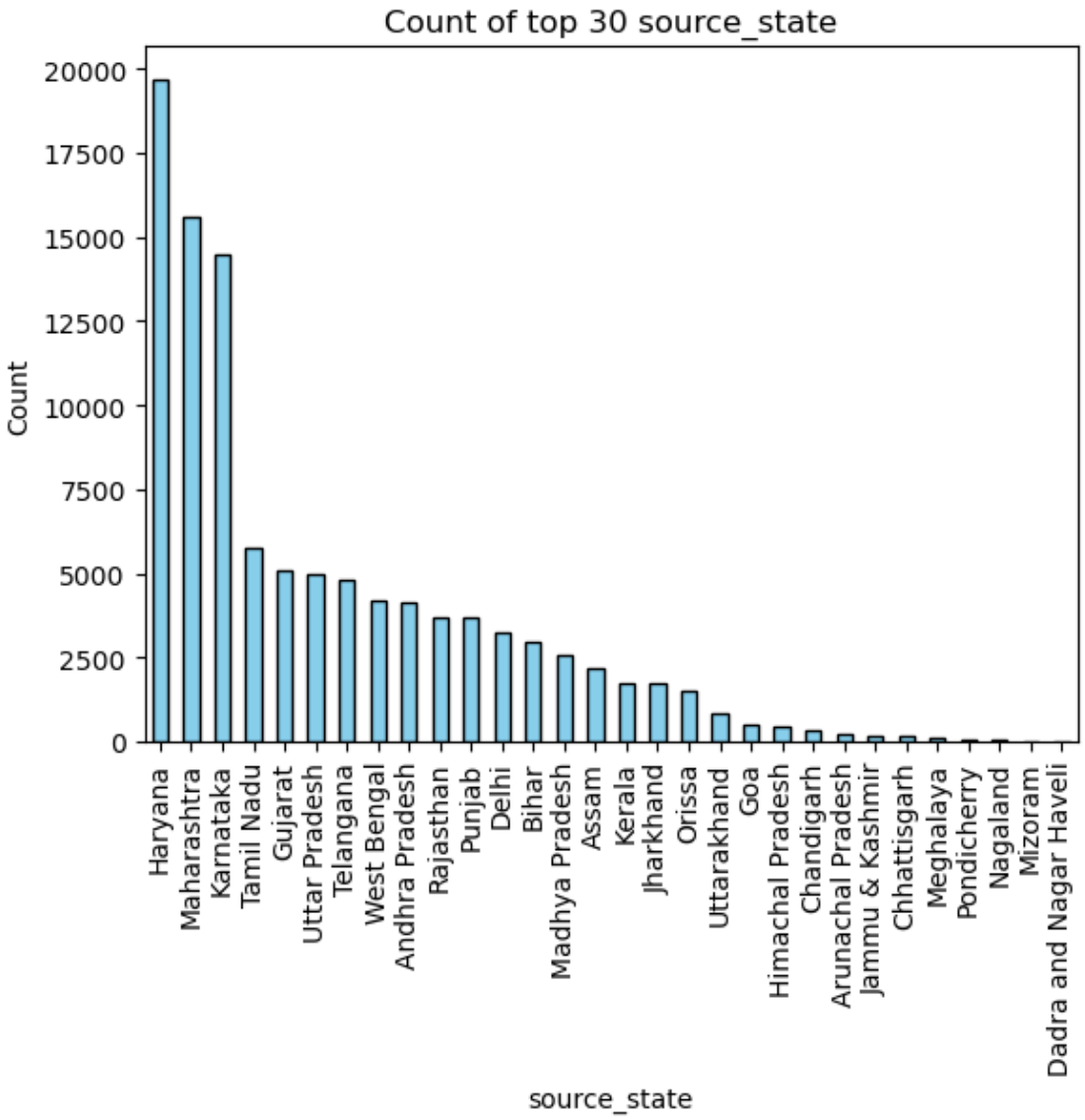

Count of top 30 route_schedule_uuid
Count
0
200
400
600
800
1000
1200
thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e5720d
thanos::sroute:4029a8a2-6c74-4b7e-a6d8-f9e069fbcea9
thanos::sroute:a1b25549-1e77-498f-8538-00292e5bd5a2
thanos::sroute:0456b740-1dad-4929-bbe0-87d8843f5a10
thanos::sroute:dca6268f-741a-4d1a-b1b0-aab13095a366
thanos::sroute:7af51efd-ae4d-49bc-9b68-345abe68ace5
thanos::sroute:bcce7b68-e962-4a29-af24-167e3e926d04
thanos::sroute:a3414a57-5b1f-4cc0-bf8e-1a46456c9f53
thanos::sroute:e2f5faaa-455a-494b-a501-549c3e3081ec
thanos::sroute:69409580-4bf0-4bbf-96fe-9e7bc6582a27
thanos::sroute:c094f55b-3c48-4ce4-b649-b99a4af2a99c
thanos::sroute:67c77992-49e3-4594-9a75-9861ef082a00
thanos::sroute:15c921f8-8a40-44cc-95ac-59f93d25f780
thanos::sroute:2c33e360-7e52-4d2c-a9db-fe24996829c7
thanos::sroute:bc7dbb1d-9379-4674-b8d3-f9c3b965bb38
thanos::sroute:be1c03eb-fd2f-4455-a933-5e3d0857e027
thanos::sroute:396d96a3-e2f8-4c40-af0e-056e11f57761
thanos::sroute:6be6529b-f2ad-4714-b7ab-ac58f24711da
thanos::sroute:25dd334d-e30d-4cfa-b6e8-192bcfe84fcc
thanos::sroute:d44a95bc-1d78-49aa-a987-ea5763aa2cbe
thanos::sroute:6f20715d-5684-4595-a986-2836d48fc5b1
thanos::sroute:824ae884-8323-4beb-b955-27c6772f0139
thanos::sroute:162f9a67-5ebe-4338-8450-1c6d57d2c7b1
thanos::sroute:95390c11-b18d-4f05-84c3-0e2e864b7da1
thanos::sroute:951d77aa-4725-4c4e-882d-42acc35b3303
thanos::sroute:64d4c6c9-ffb9-4794-b9f1-05f064c9dd0d
thanos::sroute:3def24ea-f80d-4d49-b450-720dc5d13d04
thanos::sroute:96a80600-40e1-436b-9161-fa68f9e4b0d6
thanos::sroute:9d80b949-2d55-4578-b0ee-881d4bfcc097
thanos::sroute:19271c51-91cf-4bd7-8c7a-6e3f2d3c5db9
route_schedule_uuid

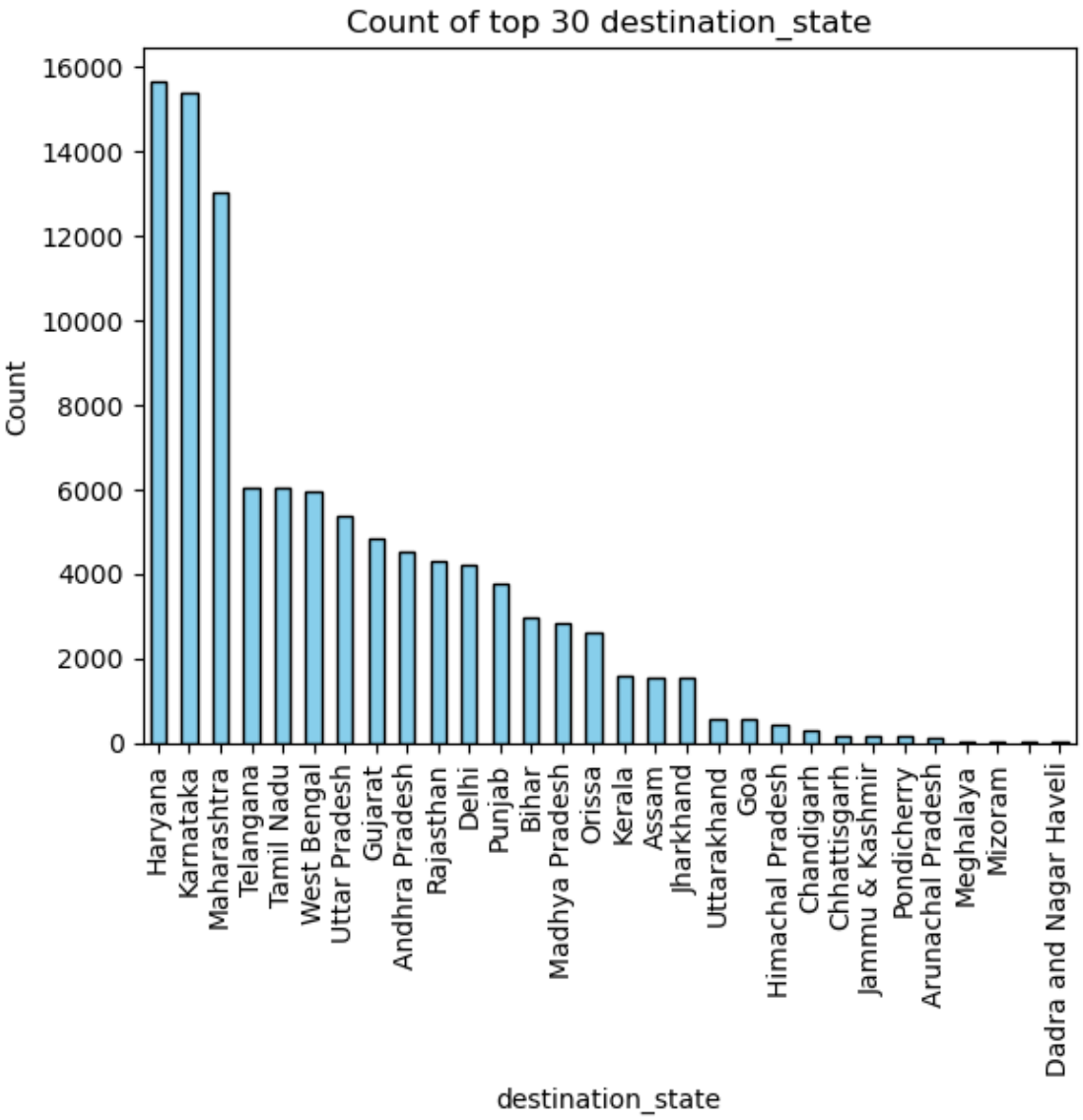

Count of top 30 source_center
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
IND000000ACB
IND562132AAA
IND421302AAG
IND411033AAA
IND501359AAE
IND160002AAC
IND712311AAA
IND395023AAA
IND560099AAB
IND110037AAM
IND131028AAB
IND382430AAB
IND600056AAB
IND751002AAB
IND781018AAB
IND560300AAA
IND209304AAA
IND462022AAA
IND834002AAB
IND854326AAB
IND530012AAA
IND110044AAB
IND000000ACT
IND302014AAA
IND842001AAA
IND400072AAB
IND306401AAB
IND121004AAB
IND416012AAA
IND821115AAB
source_center

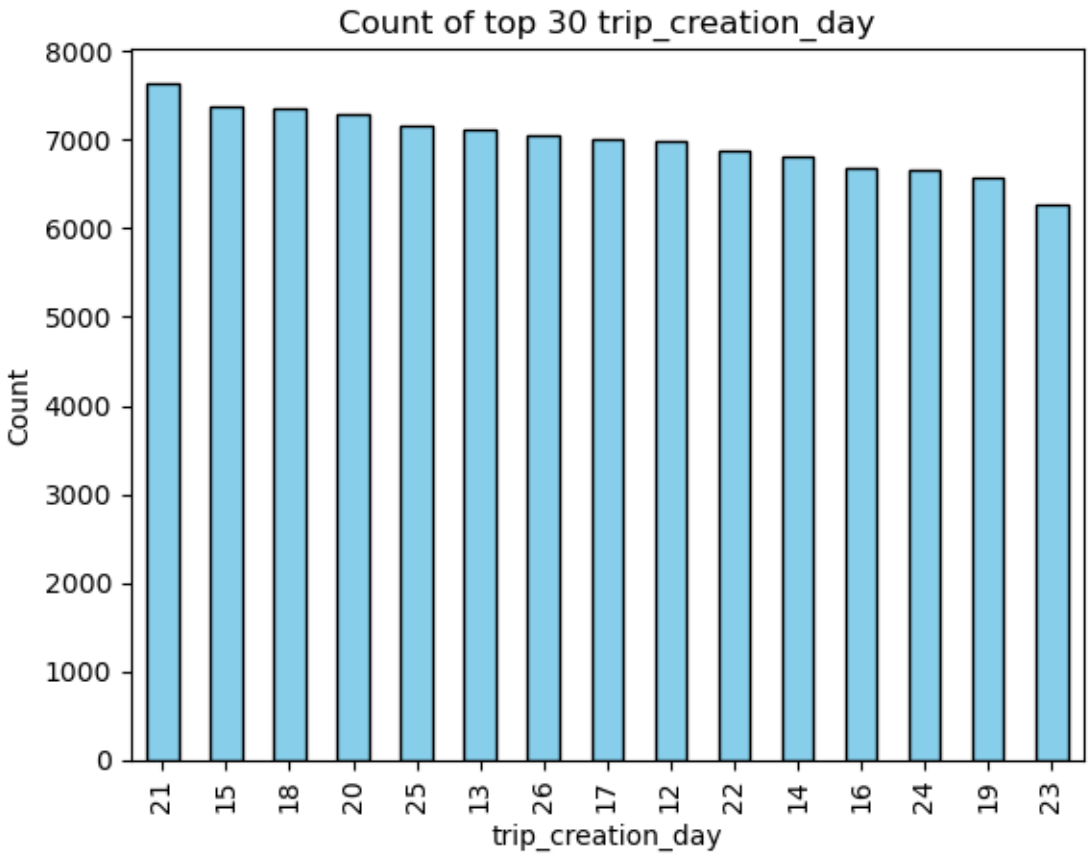

Count of top 30 destination_center
Count
0
2000
4000
6000
8000
10000
IND000000ACB
IND562132AAA
IND421302AAG
IND501359AAE
IND712311AAA
IND411033AAA
IND110037AAM
IND131028AAB
IND751002AAB
IND160002AAC
IND600056AAB
IND209304AAA
IND382430AAB
IND462022AAA
IND560099AAB
IND302014AAA
IND560300AAA
IND842001AAA
IND521225AAB
IND395023AAA
IND834002AAB
IND821115AAB
IND530012AAA
IND413002AAA
IND403726AAA
IND416012AAA
IND683511AAA
IND781018AAB
IND121004AAB
IND400072AAB
destination_center

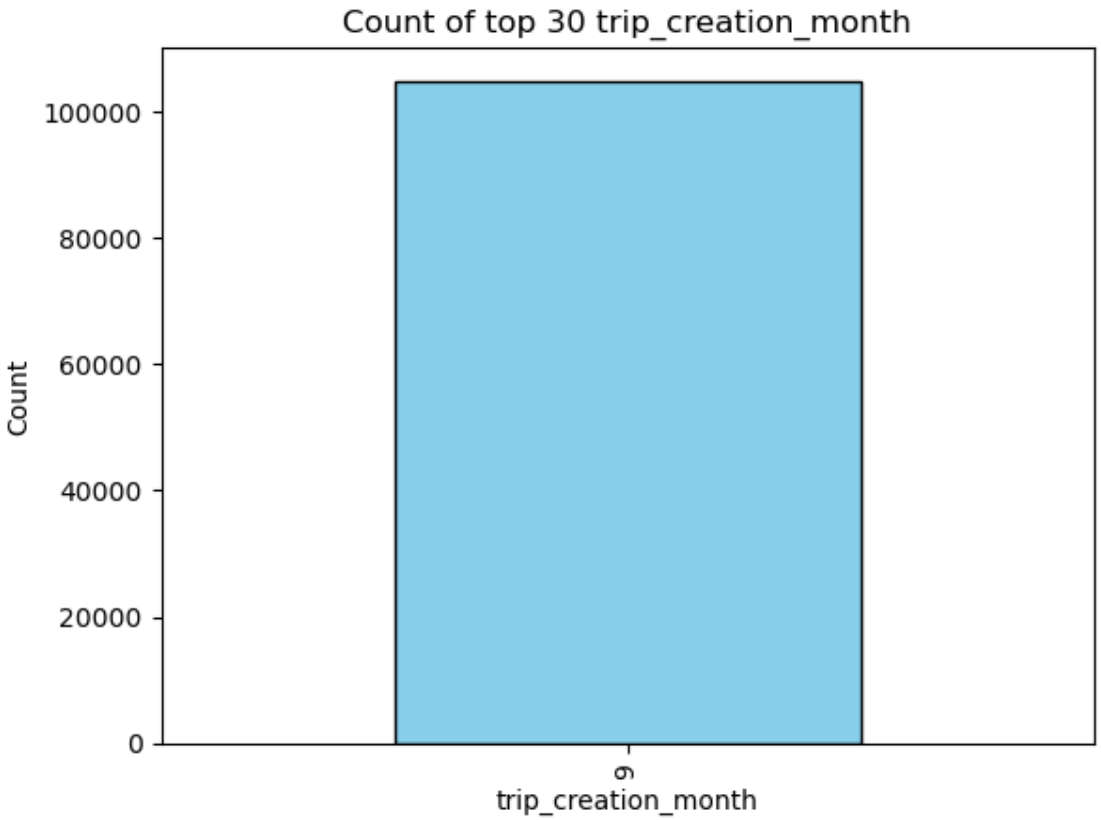

Count of top 30 source_city
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
Gurgaon
Bangalore
Bhiwandi
Pune
Bengaluru
Hyderabad
Delhi
Chandigarh
Kolkata
Surat
Mumbai
Sonipat
Ahmedabad
MAA
Guwahati
Bhubaneshwar
Jaipur
Kanpur
Bhopal
Chennai
Ranchi
Purnia
Visakhapatnam
Del
Muzaffrpur
Pali
Noida
Ludhiana
FBD
Kolhapur
source_city

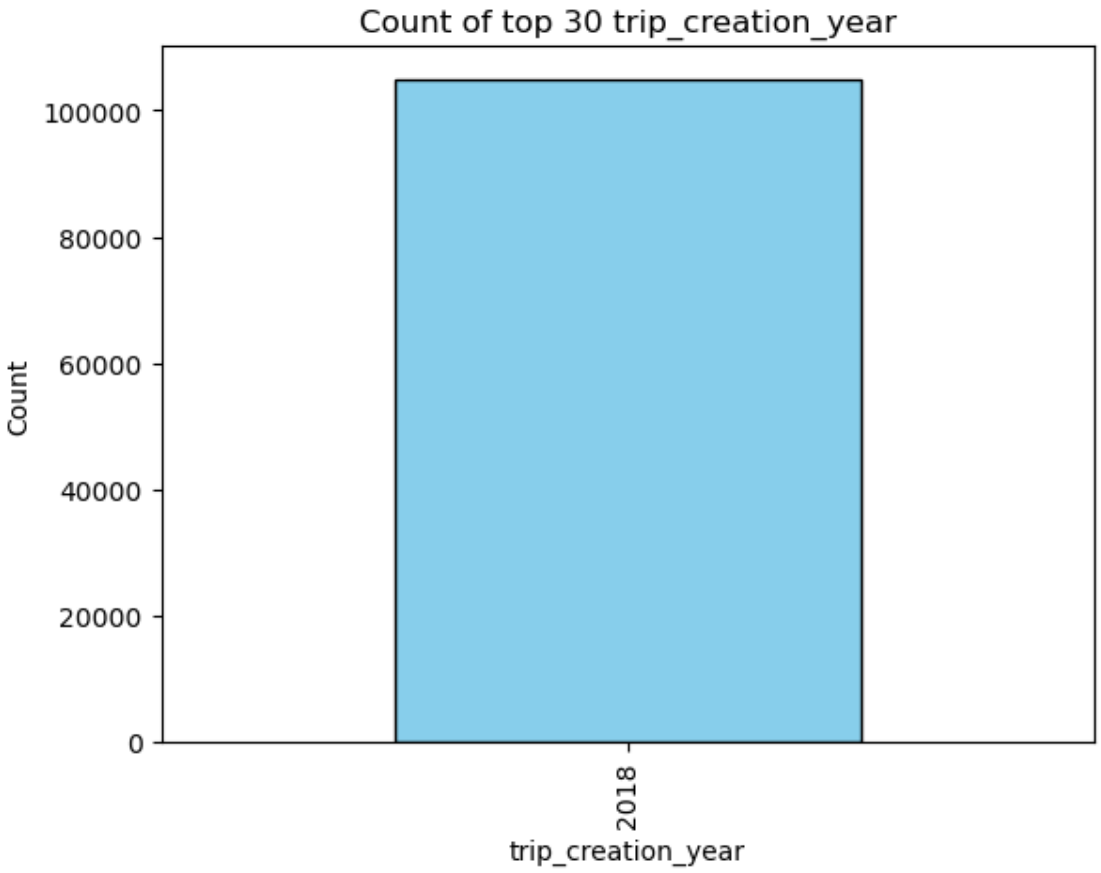

Count of top 30 destination_city
Count
12000
10000
8000
6000
4000
2000
0
Gurgaon
Bangalore
Hyderabad
Bhiwandi
Delhi
Kolkata
Bengaluru
Pune
Chandigarh
Sonipat
Bhubaneshwar
Mumbai
MAA
Kanpur
Jaipur
Ahmedabad
Bhopal
Chennai
Surat
Muzaffrpur
Vijayawada
Ranchi
Visakhapatnam
Sasaram
Goa
Solapur
Guwahati
Kolhapur
Noida
Aluva
destination_city

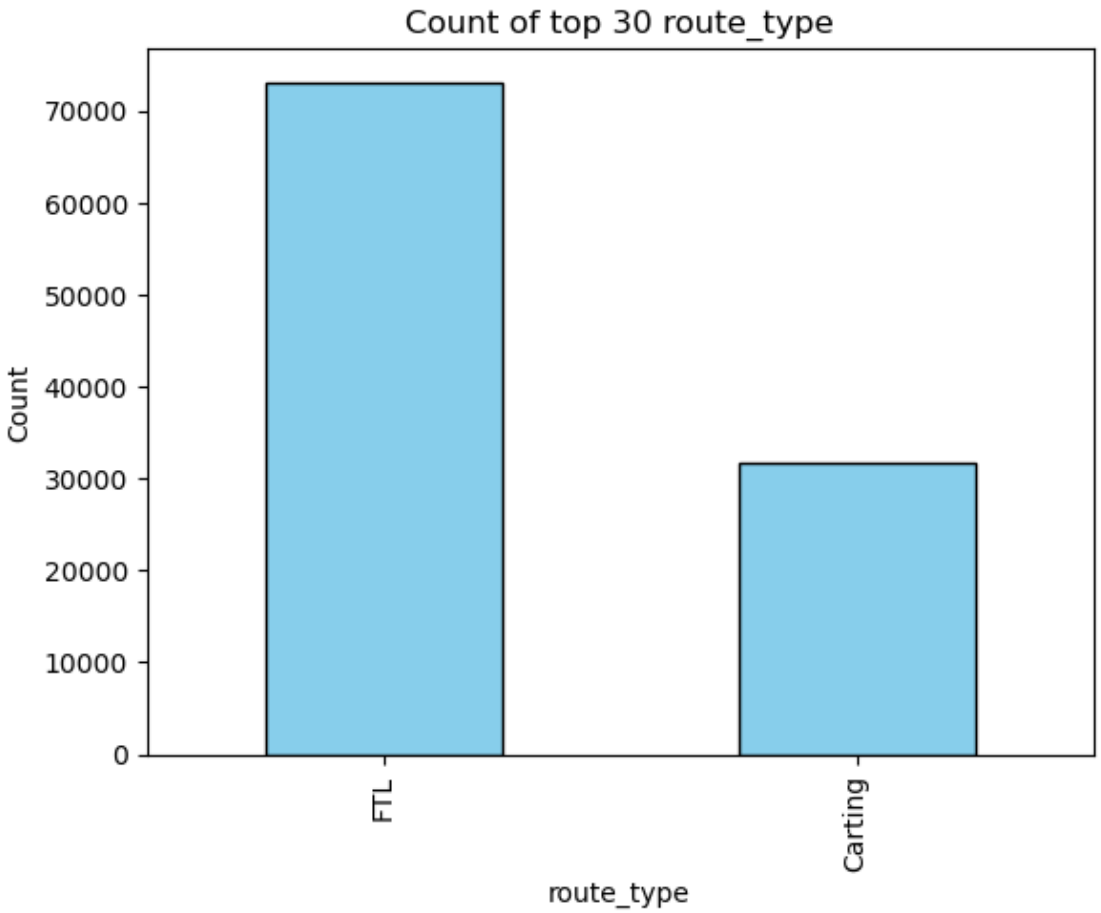

Count of top 30 source_state
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
Haryana
Maharashtra
Karnataka
Tamil Nadu
Gujarat
Uttar Pradesh
Telangana
West Bengal
Andhra Pradesh
Rajasthan
Punjab
Delhi
Bihar
Madhya Pradesh
Assam
Kerala
Jharkhand
Orissa
Uttarakhand
Goa
Himachal Pradesh
Chandigarh
Arunachal Pradesh
Jammu & Kashmir
Chhattisgarh
Meghalaya
Pondicherry
Nagaland
Mizoram
Dadra and Nagar Haveli
source_state


Count of top 30 destination_state
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
Haryana
Karnataka
Maharashtra
Telangana
Tamil Nadu
West Bengal
Uttar Pradesh
Gujarat
Andhra Pradesh
Rajasthan
Delhi
Punjab
Bihar
Madhya Pradesh
Orissa
Kerala
Assam
Jharkhand
Uttarakhand
Goa
Himachal Pradesh
Chandigarh
Chhattisgarh
Jammu & Kashmir
Pondicherry
Arunachal Pradesh
Meghalaya
Mizoram
Dadra and Nagar Haveli
destination_state


Count of top 30 trip_creation_day
Count
8000
7000
6000
5000
4000
3000
2000
1000
0
21
15
18
20
25
13
26
17
12
22
14
16
24
19
23
trip_creation_day

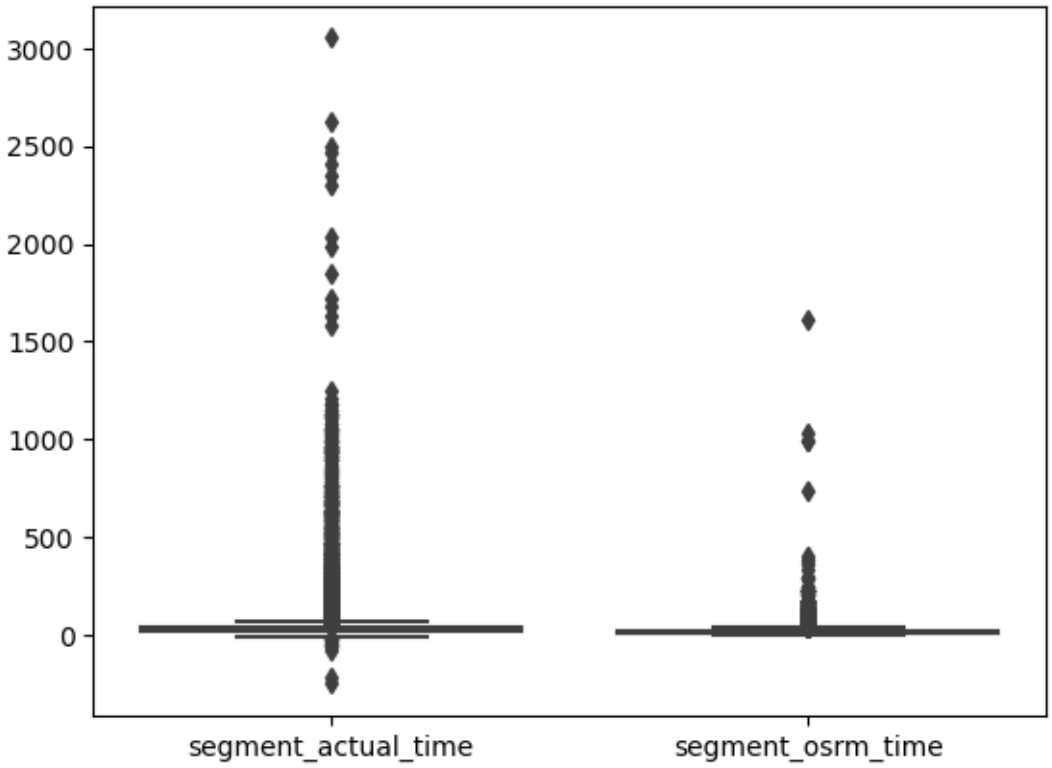

Count of top 30 trip_creation_month
100000
80000
60000
40000
20000
0
Count
9
trip_creation_month

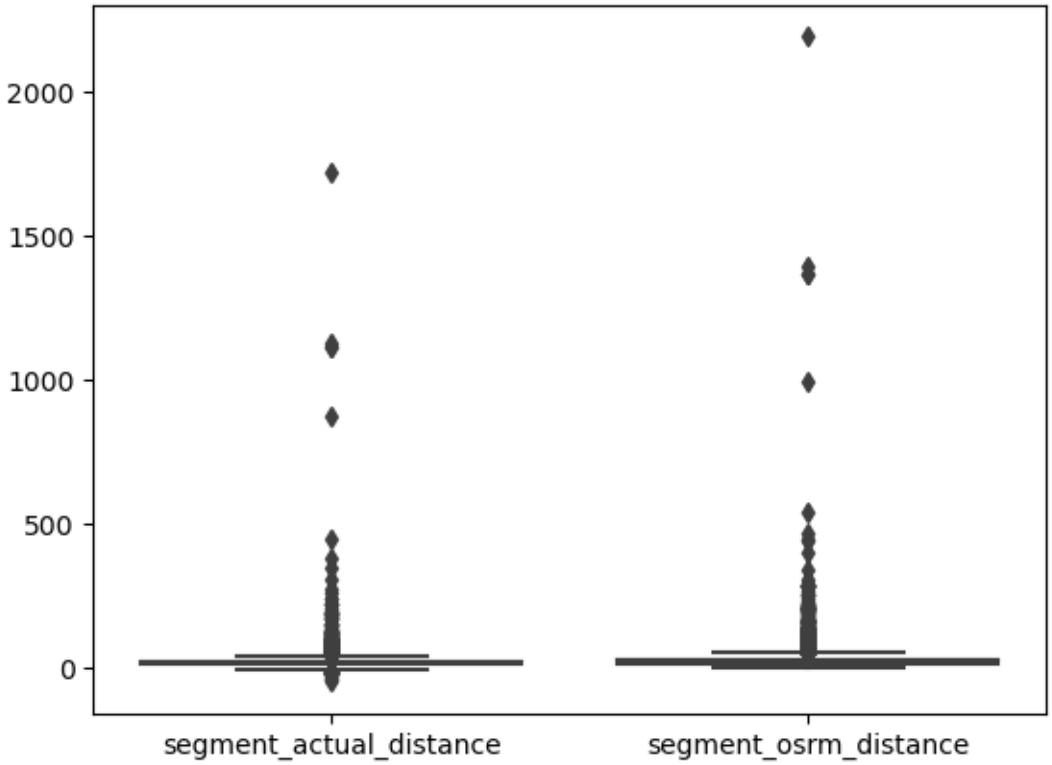

Count of top 30 trip_creation_year
Count
100000
80000
60000
40000
20000
0
2018
trip_creation_year



I will be adding plots for: 1. Distribution for Actual Time taken per segment vs segment osrm time 2. Distribution for Actual Distance per segment vs segment osrm distance

[23]:
```
sb.boxplot(data=[train_df['segment_actual_time'],␣
 ↪train_df['segment_osrm_time']])
plt.xticks([0,1], ['segment_actual_time', 'segment_osrm_time'])
plt.show()

sb.boxplot(data=[train_df['segment_actual_distance'],␣
 ↪train_df['segment_osrm_distance']])
plt.xticks([0,1], ['segment_actual_distance', 'segment_osrm_distance'])
plt.show()
```


3000
2500
2000
1500
1000
500
0
segment_actual_time
segment_osrm_time


2000
1500
1000
500
0
segment_actual_distance
segment_osrm_distance

Graph1 –> We can see a lot of variance b/w segment_actual_time and segment_osrm_time and they have a lot of skewed values for them. There seems to be much higher values of actual time when compared to the osrm time. At the same time the interquartile range and the median seems to be similar. So question arises that can some improvements can be made to:

Either 1. Take a personal look at such skewed values in actual time Or 2. The osrm algorithm is producing wrong results

Also lets look at some analysis over the outliers.

Graph2 –> The distribution around actual distance and osrm distance seems to be matching quite much and outliers are also looking good. Lets check about these outliers personally that they align with each other or no.

[24]:
```
list_zero_osrm_time = train_df[train_df['segment_actual_time']/
 ↪train_df['segment_osrm_time'] > 1000000].get(["segment_actual_time",␣
 ↪"segment_osrm_time"])
print("Number of rows with zero segment osrm time: ", len(list_zero_osrm_time))
list_very_low_osrm_time = train_df[train_df['segment_actual_time']/
 ↪train_df['segment_osrm_time'] > 100].get(["segment_actual_time",␣
 ↪"segment_osrm_time"])
print("Number of rows with very high actual to osrm time ratio: ",␣
 ↪len(list_very_low_osrm_time))
list_low_osrm_time = train_df[train_df['segment_actual_time']/
 ↪train_df['segment_osrm_time'] > 10].get(["segment_actual_time",␣
 ↪"segment_osrm_time"])
print("Number of rows with high actual to osrm time ratio: ",␣
 ↪len(list_low_osrm_time))
```

```
Number of rows with zero segment osrm time:  391
Number of rows with very high actual to osrm time ratio:  428
Number of rows with high actual to osrm time ratio:  1692
```

With so many osrm time values as 0 it is highly anticipated that there is something wrong with the algorithm computing the value. Lets look at skewed values for distances:

[25]:
```
list_zero_osrm_distance = train_df[train_df['segment_actual_distance']/
 ↪train_df['segment_osrm_distance'] > 1000000].get(["segment_actual_distance",␣
 ↪"segment_osrm_distance"])
print("Number of rows with zero segment osrm distance: ",␣
 ↪len(list_zero_osrm_distance))
list_very_low_osrm_distance = train_df[train_df['segment_actual_distance']/
 ↪train_df['segment_osrm_distance'] > 100].get(["segment_actual_distance",␣
 ↪"segment_osrm_distance"])
print("Number of rows with very high actual to osrm distance ratio: ",␣
 ↪len(list_very_low_osrm_distance))
```

```
list_low_osrm_distance = train_df[train_df['segment_actual_distance']/
 ↪train_df['segment_osrm_distance'] > 10].get(["segment_actual_distance",␣
 ↪"segment_osrm_distance"])
print("Number of rows with high actual to osrm distance ratio: ",␣
 ↪len(list_low_osrm_distance))
```

```
Number of rows with zero segment osrm distance:  1089
Number of rows with very high actual to osrm distance ratio:  1095
Number of rows with high actual to osrm distance ratio:  1111
```

There is a very high number of rows with value of segment_osrm_distance and this needs to be checked why the computing algorithm is producing so many 0 values.

Lets look at information present for a single trip so that we can further proceed with merging rows with common trip_id values:

[26]:
```
data_for_single_trip = train_df[train_df['trip_uuid'] ==␣
 ↪'trip-153741093647649320']
row0 = data_for_single_trip.loc[0]
row1 = data_for_single_trip.loc[1]
row2 = data_for_single_trip.loc[2]
row3 = data_for_single_trip.loc[3]
row4 = data_for_single_trip.loc[4]
row5 = data_for_single_trip.loc[5]

print(row0.compare(row1))
```

```
                                  self      other
actual_distance_to_destination  10.43566  18.936842
actual_time                        14.0       24.0
osrm_time                          11.0       20.0
osrm_distance                   11.9653    21.7243
segment_actual_time                14.0       10.0
segment_osrm_time                  11.0        9.0
segment_osrm_distance           11.9653      9.759
segment_actual_distance         10.43566   8.501182
```

Lets create a new dataframe by merging information for unique trip_id

for that, first we will have to merge information for unique set of trip_id, od_start_time, od_end_time

Next we will create a new dataframe by merging the rows with info about same trip id, lets begin. We will group all rows using columns [trip_uuid, od_start_time, od_end_time]

The following values of merged rows will be evaluated by:

1. actual_distance_to_destination - max of actual_distance_to_destination values in group by
2. actual_time - max of actual_time values in group by
3. osrm_total_time - sum of all values of osrm_time
4. osrm_distance - max of osrm_distance values in group by

5. total_actual_time_by_segment - sum of all segment_actual_time
6. total_osrm_time_by_segment - sum of all segment_osrm_time
7. total_osrm_distance_by_segment - sum of all segment_osrm_distance
8. total_actual_distance_by_segment = sum of all segment_actual_distance

[27]:
```
train_merged_df_object = train_df.groupby(['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',␣
 ↪'source_state', 'destination_state',
                                            'source_city', 'destination_city',␣
 ↪'trip_creation_year',
                                            'trip_creation_month',␣
 ↪'trip_creation_day'])

train_merged_df = train_merged_df_object.agg(
    {
        'actual_distance_to_destination': 'max',
        'actual_time': 'max',
        'osrm_time': 'sum',
        'osrm_distance': 'max',
        'segment_actual_time': 'sum',
        'segment_osrm_time': 'sum',
        'segment_osrm_distance': 'sum',
        'segment_actual_distance': 'sum',
    }
).reset_index()

# renaming column names

train_merged_df.rename(columns = {
    'osrm_time': 'osrm_total_time',
    'osrm_distance': 'osrm_total_distance',
    'segment_actual_time': 'total_actual_time_by_segment',
    'segment_osrm_time': 'total_osrm_time_by_segment',
    'segment_osrm_distance': 'total_osrm_distance_by_segment',
    'segment_actual_distance': 'total_actual_distance_by_segment'
}, inplace = True)

print(train_merged_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18948 entries, 0 to 18947
Data columns (total 27 columns):
 #   Column                            Non-Null Count  Dtype
```

```
---  ------                          --------------  -----
 0   data                            18948 non-null  object
 1   trip_creation_time              18948 non-null  object
 2   route_schedule_uuid             18948 non-null  object
 3   route_type                      18948 non-null  object
 4   trip_uuid                       18948 non-null  object
 5   source_center                   18948 non-null  object
 6   source_name                     18948 non-null  object
 7   destination_center              18948 non-null  object
 8   destination_name                18948 non-null  object
 9   od_start_time                   18948 non-null  object
 10  od_end_time                     18948 non-null  object
 11  start_scan_to_end_scan          18948 non-null  float64
 12  source_state                    18948 non-null  object
 13  destination_state               18948 non-null  object
 14  source_city                     18948 non-null  object
 15  destination_city                18948 non-null  object
 16  trip_creation_year              18948 non-null  int64
 17  trip_creation_month             18948 non-null  int64
 18  trip_creation_day               18948 non-null  int64
 19  actual_distance_to_destination  18948 non-null  float64
 20  actual_time                     18948 non-null  float64
 21  osrm_total_time                 18948 non-null  float64
 22  osrm_total_distance             18948 non-null  float64
 23  total_actual_time_by_segment    18948 non-null  float64
 24  total_osrm_time_by_segment      18948 non-null  float64
 25  total_osrm_distance_by_segment  18948 non-null  float64
 26  total_actual_distance_by_segment  18948 non-null  float64
dtypes: float64(9), int64(3), object(15)
memory usage: 3.9+ MB
None
```

[28]:
```
train_merged_df.nunique()
```

[28]:
```
data                                1
trip_creation_time              10654
route_schedule_uuid              1385
route_type                          2
trip_uuid                       10654
source_center                    1425
source_name                      1423
destination_center               1409
destination_name                 1405
od_start_time                   18948
od_end_time                     18948
start_scan_to_end_scan           1681
source_state                       32
```

```
destination_state                      32
source_city                          1189
destination_city                     1193
trip_creation_year                      1
trip_creation_month                     1
trip_creation_day                      15
actual_distance_to_destination      18920
actual_time                          1434
osrm_total_time                      2303
osrm_total_distance                 18762
total_actual_time_by_segment         1451
total_osrm_time_by_segment            951
total_osrm_distance_by_segment      18811
total_actual_distance_by_segment    18920
dtype: int64
```

[29]:
```
train_merged_df.nunique()
train_merged_df.head().
  ↪get(['trip_uuid','od_start_time','od_end_time','total_actual_time_by_segment',␣
  ↪'total_osrm_time_by_segment','start_scan_to_end_scan', 'actual_time'])
# train_merged_df.info()
```

[29]:

	trip_uuid	od_start_time	od_end_time	total_actual_time_by_segment	total_osrm_time_by_segment	start_scan_to_end_scan	actual_time
0	trip-153671041653548748	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	728.0	534.0	1260.0	732.0
1	trip-153671041653548748	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	820.0	474.0	999.0	830.0
2	trip-153671042288605164	2018-09-12 02:03:09.655591	2018-09-12 03:01:59.598855	46.0	26.0	58.0	47.0
3	trip-153671042288605164	2018-09-12 00:00:22.886430	2018-09-12 02:03:09.655591	95.0	39.0	122.0	96.0
4	trip-153671043369099517	2018-09-14 03:40:17.106733	2018-09-14 17:34:55.442454	608.0	231.0	834.0	611.0

So the first merge was based on three unique values them being trip_UUID, source_centre, destination_centre and we can see that there are total 18948 rows but only about 10654 rows with unique trip ID. We can further merge these rows to get information of a single tripid. Currently we would like to have some analysis on the current data in which we can infer some trends about trips which have different source and destination.

Lets create a new column 'od_journey_time' whose value is equal to od_end_time - od_start_time

[30]:
```
conversion_array = list(map(lambda x: datetime.strptime(x, datetime_object) ,␣
↪train_df['trip_creation_time']))

def computeTimeDifference(od_start_time, od_end_time):
        datetime_object = "%Y-%m-%d %H:%M:%S.%f"
        end_datetime = datetime.strptime(od_end_time, datetime_object)
        start_datetime = datetime.strptime(od_start_time, datetime_object)
        diff = end_datetime-start_datetime
        return diff.seconds/60


computeTimeDifference('2018-09-12 16:39:46.858469' ,'2018-09-13 13:40:23.
↪123744')

for index, row in train_merged_df.iterrows():
    diff_in_minutes = computeTimeDifference(row['od_start_time'],␣
↪row['od_end_time'])
    train_merged_df.loc[index, 'od_journey_time'] = diff_in_minutes

train_merged_df.head().get(['trip_uuid',␣
↪'total_osrm_time_by_segment','total_actual_time_by_segment','actual_time',␣
↪'od_journey_time','start_scan_to_end_scan'])
```

[30]:

	trip_uuid	total_osrm_time_by_segment	total_actual_time_by_segment	actual_time	od_journey_time	start_scan_to_end_scan
0	trip-153671041653548748	534.0	728.0	732.0	1260.600000	1260.0
1	trip-153671041653548748	474.0	820.0	830.0	999.500000	999.0
2	trip-153671042288605164	26.0	46.0	47.0	58.816667	58.0
3	trip-153671042288605164	39.0	95.0	96.0	122.766667	122.0
4	trip-153671043369099517	231.0	608.0	611.0	834.633333	834.0

[31]:
```
train_merged_df.head().get(['trip_uuid','osrm_total_distance',␣
↪'total_osrm_distance_by_segment','total_actual_distance_by_segment',␣
↪'actual_distance_to_destination'])
```

[31]:

	trip_uuid	osrm_total_distance	total_osrm_distance_by_segment	total_actual_distance_by_segment	actual_distance_to_destination
0	trip-153671041653548748	446.5496	670.6205	383.759164	383.759164
1	trip-153671041653548748	544.8027	649.8528	440.973689	440.973689
2	trip-153671042288605164	28.1994	28.1995	24.644021	24.644021
3	trip-153671042288605164	56.9116	55.9899	48.542890	48.542890
4	trip-153671043369099517	281.2109	317.7408	237.439610	242.309306

Lets analyse od_journey_time and start_scan_to_end_scan:

[32]:
```
cases_of_variance = []
query = abs(train_merged_df['od_journey_time'] -␣
↪train_merged_df['start_scan_to_end_scan'])/
↪train_merged_df['start_scan_to_end_scan'] > 0.5
cases_of_variance = train_merged_df[query]
cases_of_variance.get(['trip_uuid','od_journey_time','start_scan_to_end_scan'])
```

[32]:

	trip_uuid	od_journey_time	start_scan_to_end_scan
5	trip-153671043369099517	219.716667	3099.0
82	trip-153671321710455800	898.366667	2338.0
87	trip-153671328307356992	862.983333	2302.0
166	trip-153671715851493285	978.016667	2418.0
240	trip-153672000309775410	309.950000	1749.0
…	…	…	…
18715	trip-153799938034632401	104.266667	2984.0
18732	trip-153800006302758138	298.550000	3178.0
18739	trip-153800038563813178	29.783333	1469.0
18743	trip-153800060736604010	388.250000	1828.0
18828	trip-153800280431751148	499.566667	1939.0

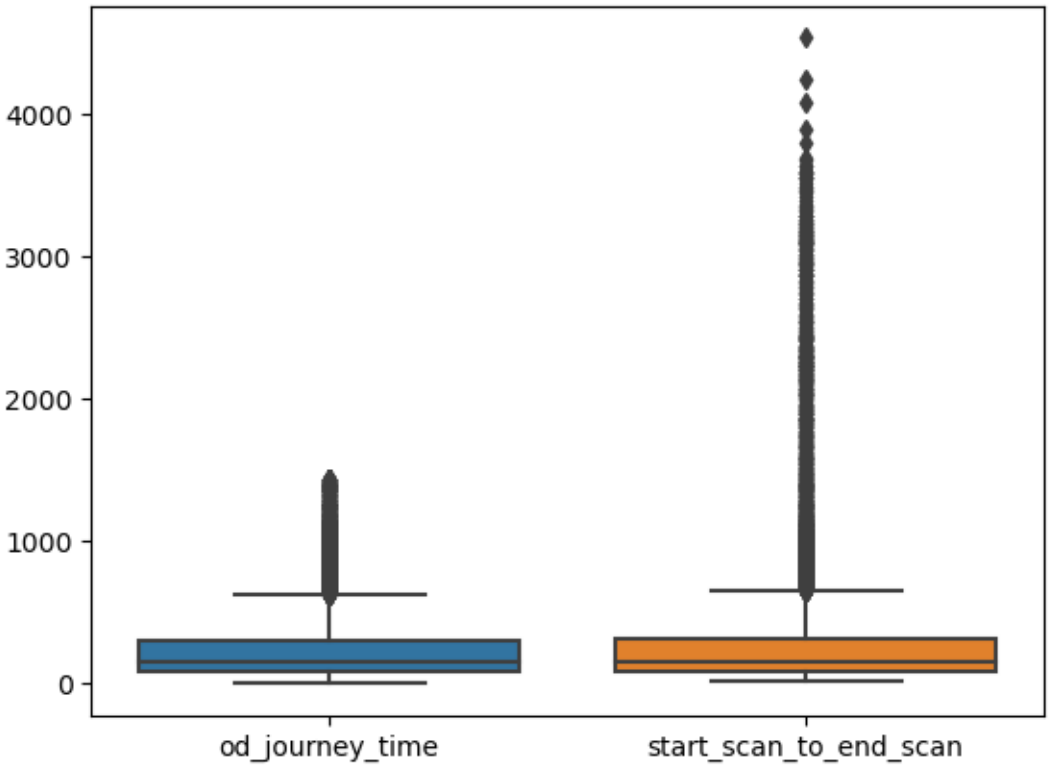
[555 rows x 3 columns]

Here we can see that there are total 555 entries which have exceptionally high start_scan_to_end_scan values when compared to actual journey time which is od_journey_time. These outliers can be found in the table above. It needs to be checked why we have such a high value of start_scan_to_end_scan value for 555/18948 i.e. about 3% of values.

Following are the different hypothesis testing and visual analysis of different columns:

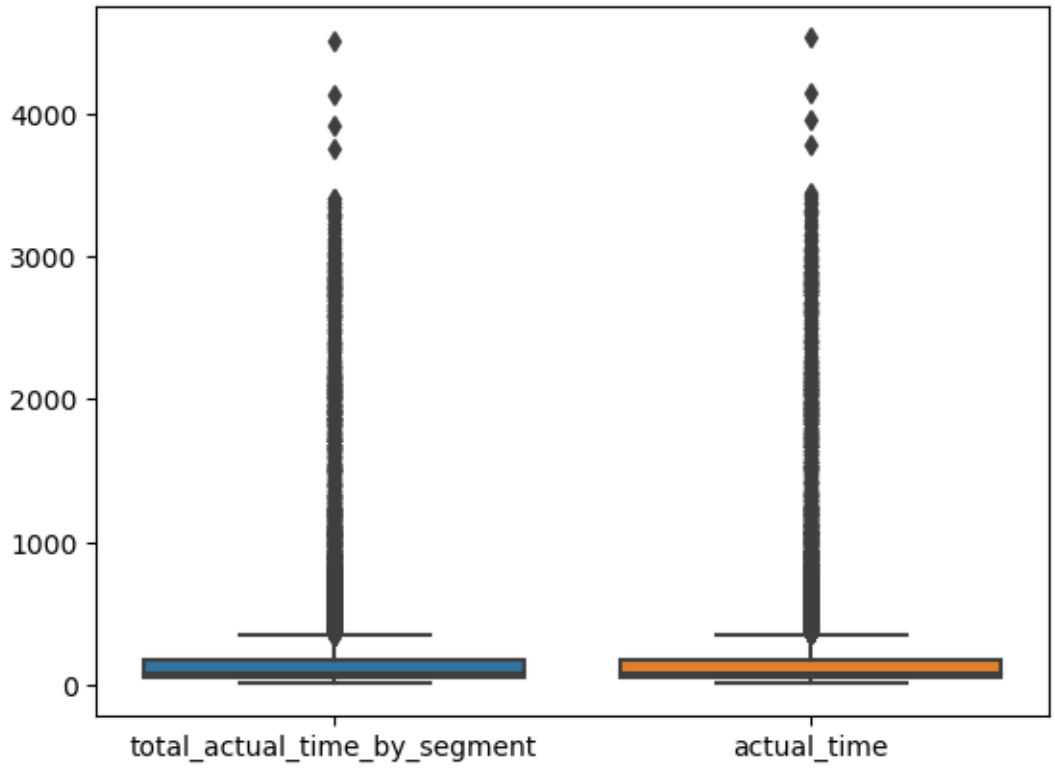
[55]:
```
# Hypothesis testing and visual analysis for od_journey_time and␣
 ↪start_scan_to_end_scan
print("Hypothesis testing and visual analysis for od_journey_time and␣
 ↪start_scan_to_end_scan")
_, p_value = stats.ttest_rel(a = train_merged_df['od_journey_time'], b =␣
 ↪train_merged_df['start_scan_to_end_scan'])
print(p_value)
if(p_value<0.05):
    print("We are rejecting the null hypothesis")
else:
    print("We are accepting the null hypothesis")

sb.boxplot(data=[train_merged_df['od_journey_time'],␣
 ↪train_merged_df['start_scan_to_end_scan']])
plt.xticks([0,1], ['od_journey_time', 'start_scan_to_end_scan'])
plt.show()
```

```
Hypothesis testing and visual analysis for od_journey_time and
start_scan_to_end_scan
3.2190875827907e-109
We are rejecting the null hypothesis
```



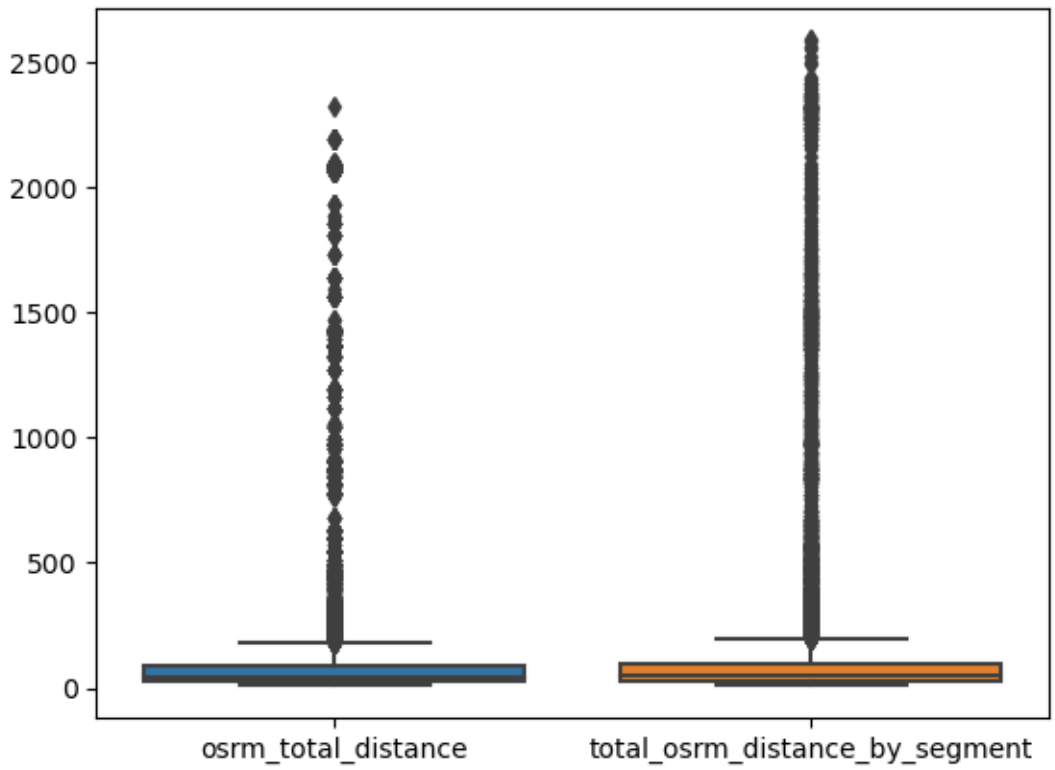
[56]:
```
# Hypothesis testing and visual analysis for total_actual_time_by_segment and␣
 ↪actual_time
print("Hypothesis testing and visual analysis for total_actual_time_by_segment␣
 ↪and actual_time")
_, p_value = stats.ttest_rel(a =␣
 ↪train_merged_df['total_actual_time_by_segment'], b =␣
 ↪train_merged_df['actual_time'])
print(p_value)
if(p_value<0.05):
    print("We are rejecting the null hypothesis")
else:
    print("We are accepting the null hypothesis")

sb.boxplot(data=[train_merged_df['total_actual_time_by_segment'],␣
 ↪train_merged_df['actual_time']])
plt.xticks([0,1], ['total_actual_time_by_segment', 'actual_time'])
plt.show()
```

```
Hypothesis testing and visual analysis for total_actual_time_by_segment and
actual_time
0.0
We are rejecting the null hypothesis
```



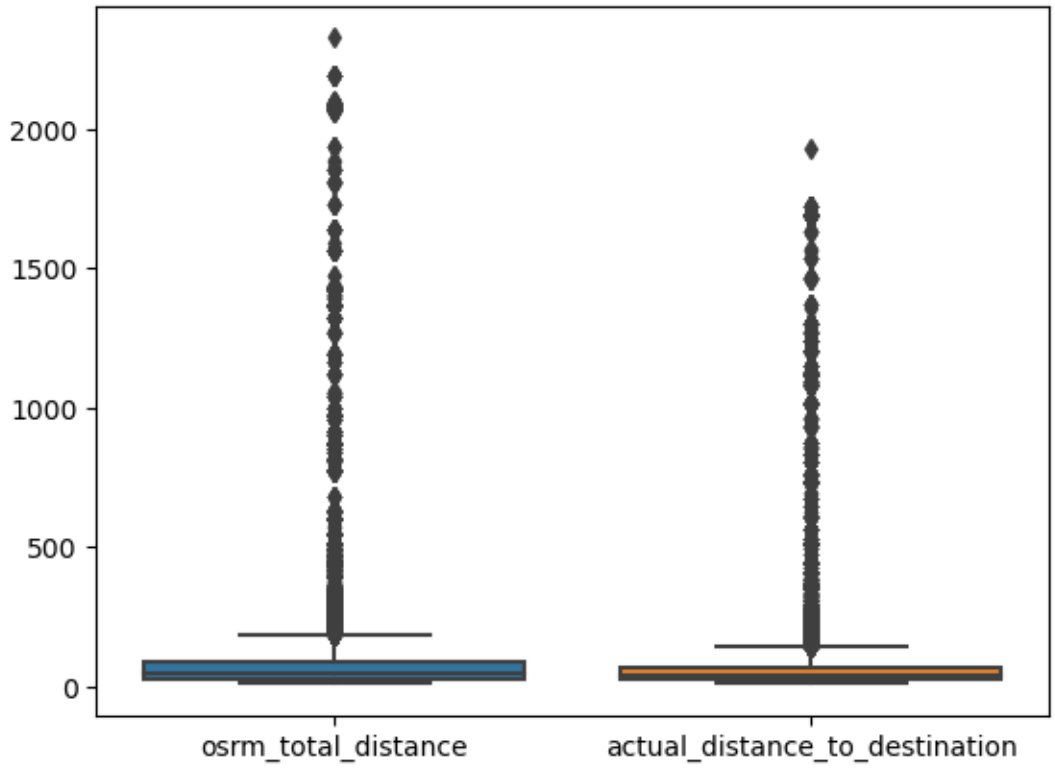
[57]:
```
# Hypothesis testing and visual analysis for osrm_total_distance and␣
 ↪total_osrm_distance_by_segment
print("Hypothesis testing and visual analysis for osrm_total_distance and␣
 ↪total_osrm_distance_by_segment")
_, p_value = stats.ttest_rel(a = train_merged_df['osrm_total_distance'], b =␣
 ↪train_merged_df['total_osrm_distance_by_segment'])
print(p_value)
if(p_value<0.05):
    print("We are rejecting the null hypothesis")
else:
    print("We are accepting the null hypothesis")

sb.boxplot(data=[train_merged_df['osrm_total_distance'],␣
 ↪train_merged_df['total_osrm_distance_by_segment']])
plt.xticks([0,1], ['osrm_total_distance', 'total_osrm_distance_by_segment'])
plt.show()
```

```
Hypothesis testing and visual analysis for osrm_total_distance and
total_osrm_distance_by_segment
1.5498527220740392e-230
We are rejecting the null hypothesis
```



[58]:
```
# Hypothesis testing and visual analysis for osrm_total_distance and␣
 ↪actual_distance_to_destination
print("Hypothesis testing and visual analysis for osrm_total_distance and␣
 ↪actual_distance_to_destination")
_, p_value = stats.ttest_rel(a = train_merged_df['osrm_total_distance'], b =␣
 ↪train_merged_df['actual_distance_to_destination'])
print(p_value)
if(p_value<0.05):
    print("We are rejecting the null hypothesis")
else:
    print("We are accepting the null hypothesis")

sb.boxplot(data=[train_merged_df['osrm_total_distance'],␣
 ↪train_merged_df['actual_distance_to_destination']])
plt.xticks([0,1], ['osrm_total_distance', 'actual_distance_to_destination'])
plt.show()
```

```
Hypothesis testing and visual analysis for osrm_total_distance and
actual_distance_to_destination
0.0
We are rejecting the null hypothesis
```


2000
1500
1000
500
0
osrm_total_distance
actual_distance_to_destination

[ ]: