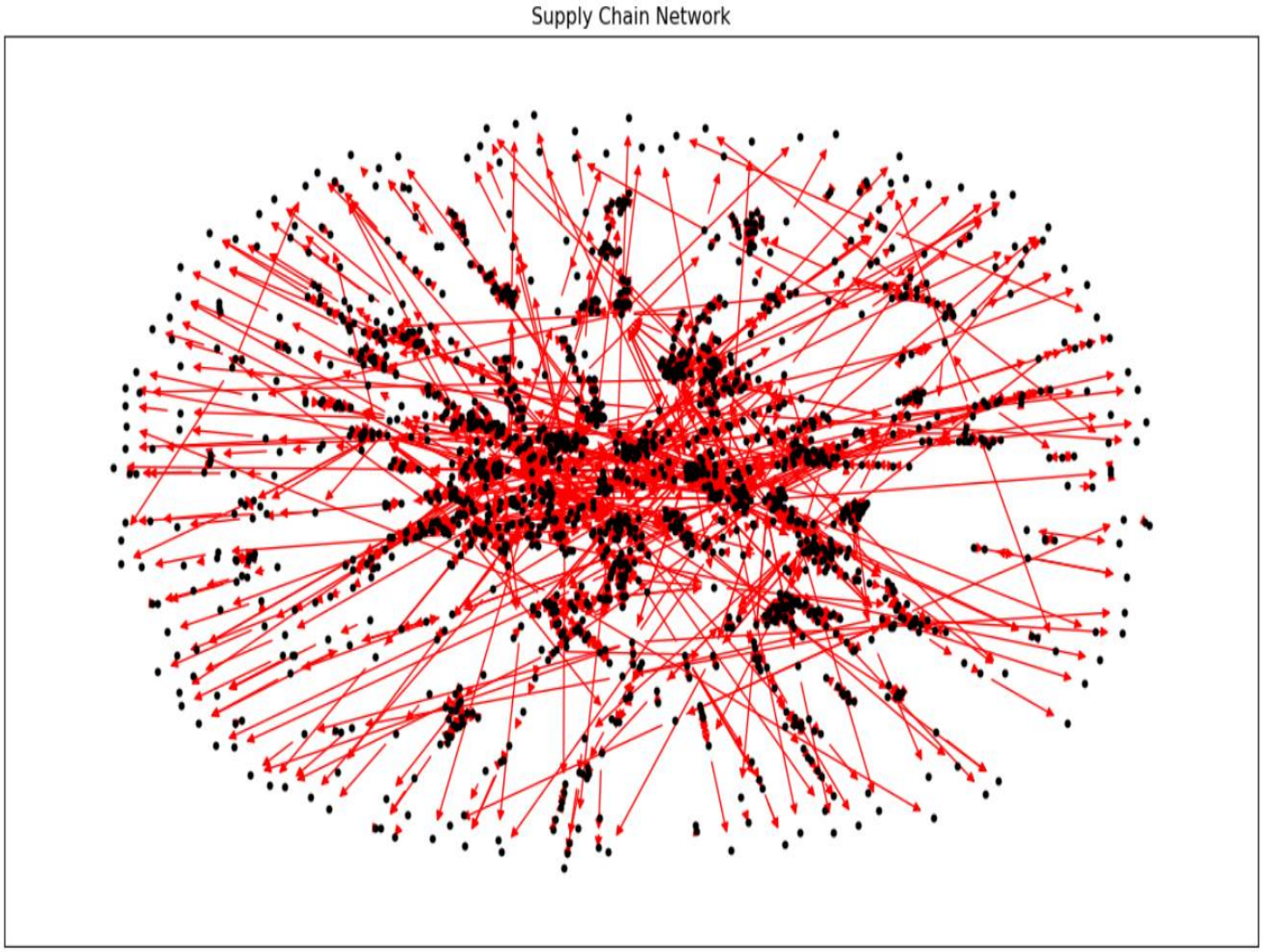
Exploratory Data Analysis of a Supply Chain Dataset



Abhijit Naik

Table of Contents

Exploratory Data Analysis of a Supply Chain Dataset:

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import *
from statsmodels import robust
import networkx as nx
import numpy as np
```

```
table = pd.read_csv("delhivery.csv")
```

```
table.columns
```

[3]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

```
table.head(5)
```

[4]:

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

5 rows × 24 columns

```
table.shape
```

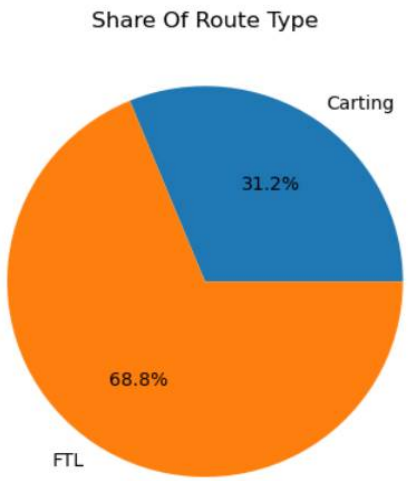
[5]:
```
(144867, 24)
```

[6]:
```
if True in table.isna():
    print("Null Present")
else:
    print("No Null Values Present.")
```

```
No Null Values Present.
```

1. Share among **Route Type:**

```
route = pd.DataFrame({'route_type':table["route_type"]})
```



2. Analysis of a Centers as a Source / Number of Orders Sent from a source:

(Each `Count` represents Number of Orders sent by the Source)

•••

[8]:

	count
count	1508.000000
mean	96.065650
std	728.748699
min	1.000000
25%	11.750000
50%	27.000000
75%	50.250000
max	23347.000000

2.1 From the above table we can see that there are:

> 1. Total Number of Source Center is `1508`.
> 2. On an Average a Source Center dispatch `96` Orders.
> 3. Point to be noted that : 25th, 50th and 75th Percentile is `11` , `27` , `50`
> 4. The Maximum and Minimum is `23347` and `1`.

> So it is Clear that there are presence of outliers in the Data.

2.2 Exploring the Data Distribution of Source :

Breaking down the Data into range of percentile we get:

```
percentage = [0.05,0.25,0.50,0.75,0.95,.98,.99] •••
```

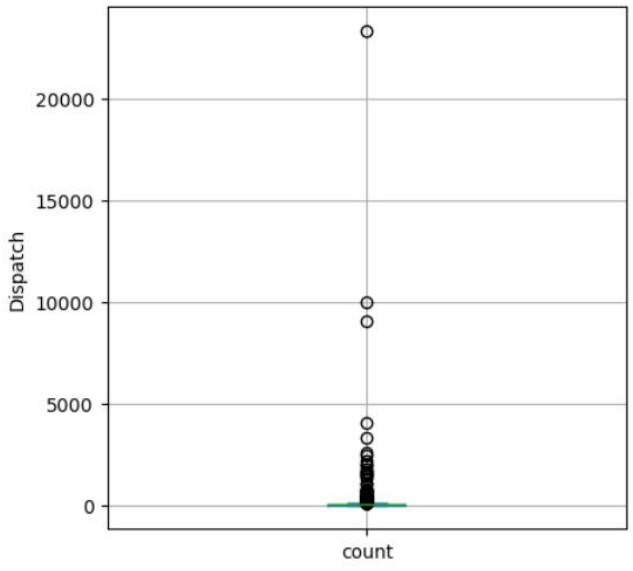
```
        5%    25%   50%    75%    95%    98%      99%
count  2.0  11.75  27.0  50.25  181.3  503.5  1467.67
```

Speard of the Source.

•••

[10]:

```
Text(0, 0.5, 'Dispatch')
```



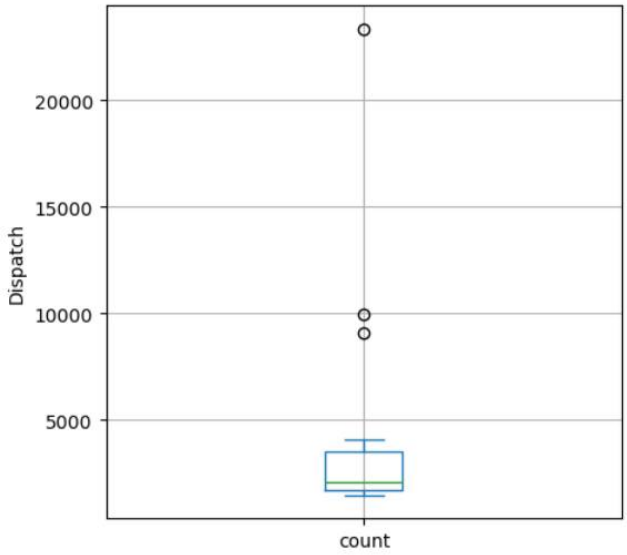
2.21 Analysing the top 99% of Source Center:

```
print(source[source["count"]>1467].sum()) •••
```

```
count    70703
dtype: int64
```

```
•••
```

[12]: Text(0, 0.5, 'Dispatch')



2.3 Mean of the Source:

```
source.mean() •••
```

[13]:
```
count    96.06565
dtype: float64
```

2.4 Trim Mean of the Source with P = 5% ie 0.05:

```
•••
```

[14]: 36.23416789396171

2.5 Median of the source:

```
source["count"].median() •••
```

[15]: 27.0

2.6 Variablity Estimates of the Source Center:

2.61 Standard Deviation:

```
source["count"].std() •••
```

[16]: 728.7486991799797

2.62 Variance:

```
source["count"].var() •••
```

[17]: 531074.6665565125

2.63 IQR :

```
•••
```

[18]: 38.5

2.64 Median Absolute Deviation:

```
robust.scale.mad(source["count"]) •••
```

[19]: 26.686839933100835

2.7 Frequency Tables and Histogram of source Center:

2.71 Frequency Table:

```
•••
```

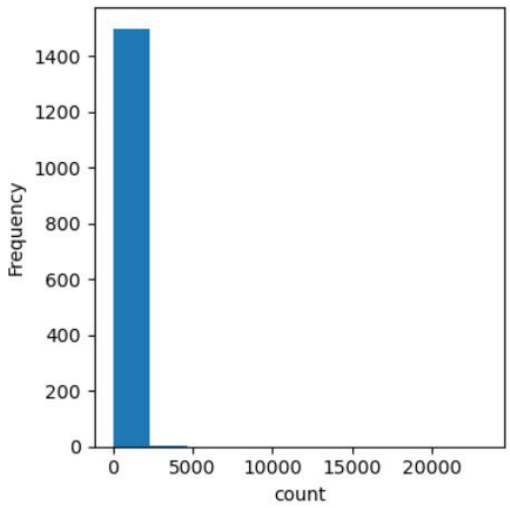
[20]:
```
count
(-22.346, 1168.3]     1490
(1168.3, 2335.6]        11
(2335.6, 3502.9]         3
(8172.1, 9339.4]         1
(9339.4, 10506.7]        1
(22179.7, 23347.0]       1
(3502.9, 4670.2]         1
(7004.8, 8172.1]         0
(5837.5, 7004.8]         0
(4670.2, 5837.5]         0
(10506.7, 11674.0]       0
(12841.3, 14008.6]       0
(14008.6, 15175.9]       0
(15175.9, 16343.2]       0
(16343.2, 17510.5]       0
(17510.5, 18677.8]       0
(18677.8, 19845.1]       0
(19845.1, 21012.4]       0
(21012.4, 22179.7]       0
(11674.0, 12841.3]       0
Name: count, dtype: int64
```

```
print(source[source["count"] < 0 ]) •••
```

```
Empty DataFrame
Columns: [count]
Index: []
```

2.71 Histogram:

```
ax = (source["count"]).plot.hist(figsize=(4, 4)) •••
```



3. Analysising Destination Centers:

Each `count` represents number or orders recieved by a center.

[23]:
```
a = pd.DataFrame({"destination":table['destination_center'],"count":1})
destination = a.groupby('destination').count()
destination.head(10)
```

[23]:

destination	count
IND000000AAL	37
IND000000AAS	24
IND000000AAZ	6
IND000000ABA	47
IND000000ABD	26
IND000000ACA	474
IND000000ACB	15192
IND000000ACN	22

3.1 Mean of Destination:

```
destination.mean()
```

[24]:
```
count    97.817016
dtype: float64
```

3.2 Trim Mean of Destination with P = 10% ie 0.1:

```

```

[25]:
```
32.151898734177216
```

The Difference in Mean and Trim Mean Suggests Presence of Outliers.

3.3 Median of the Destination:

```
destination["count"].median()
```

[26]:
```
26.0
```

3.4 Variability Estimates Of Destination:

3.41 Standard Deviation:

```
destination["count"].std()
```

[27]:
```
587.9041113963759
```

3.42 Variance:

```
destination["count"].var()
```

[28]:
```
345631.24419676245
```

3.45 IQR:

```

```

[29]:
```
40.0
```

3.46 Median Absolute Deviation:

```
robust.scale.mad(destination['count'])
```

[30]:
```
26.686839933100835
```

3.5 Exploring the Data Distribution: ¶

3.51 Percentiles :

```
percentage = [0.05,0.25,0.50,0.75,0.95]
```

```
        5%   25%   50%   75%    95%
count  2.0  10.0  26.0  50.0  210.0
```

3.5 Exploring the Data Distribution: ¶

3.51 Percentiles :

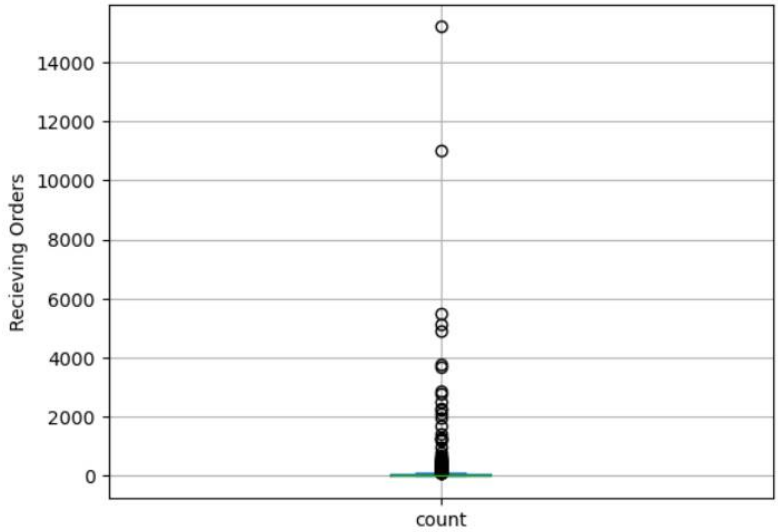
```
percentage = [0.05,0.25,0.50,0.75,0.95] •••
```

```
        5%   25%   50%   75%    95%
count  2.0  10.0  26.0  50.0  210.0
```

3.52 Spread of the Destination:

```
•••
```

[32]:
```
Text(0, 0.5, 'Recieving Orders')
```

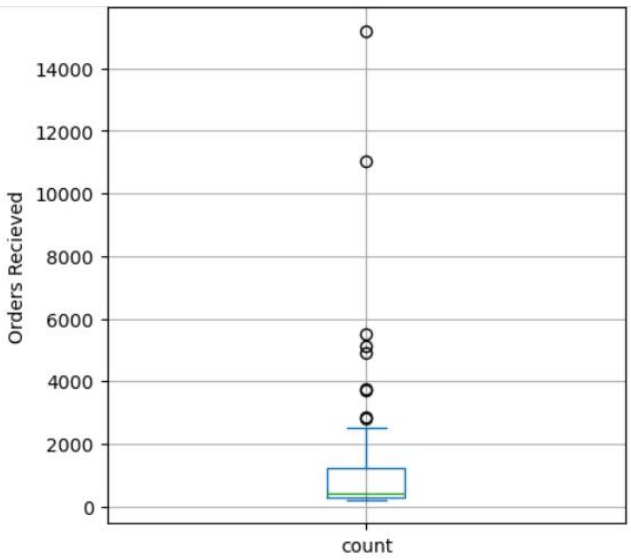


Similar to Source, there are Groups of Destination Center which recieves significant amount of goods.

3.6 Analysing the top 95% of Destinations:

```
•••
```

[33]:
```
Text(0, 0.5, 'Orders Recieved')
```



3.7 Frequency Tables and Histograms of Destination:

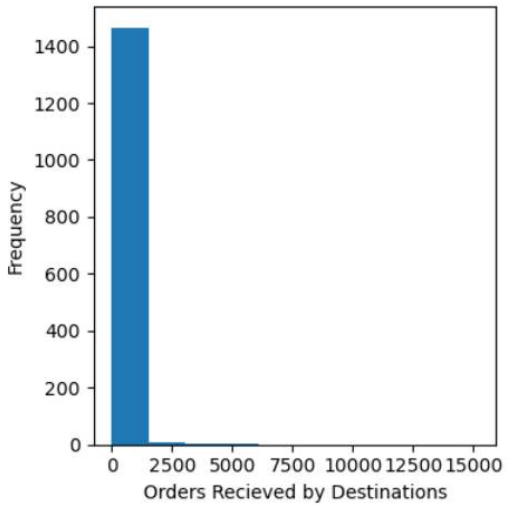
```
binneddestination = pd.cut(destination['count'],10)
```

[34]:
```
count
(-14.191, 1520.1]     1466
(1520.1, 3039.2]         8
(4558.3, 6077.4]         3
(3039.2, 4558.3]         2
(10634.7, 12153.8]       1
(13672.9, 15192.0]       1
(6077.4, 7596.5]         0
(7596.5, 9115.6]         0
(9115.6, 10634.7]        0
(12153.8, 13672.9]       0
Name: count, dtype: int64
```

```
destination.shape
```

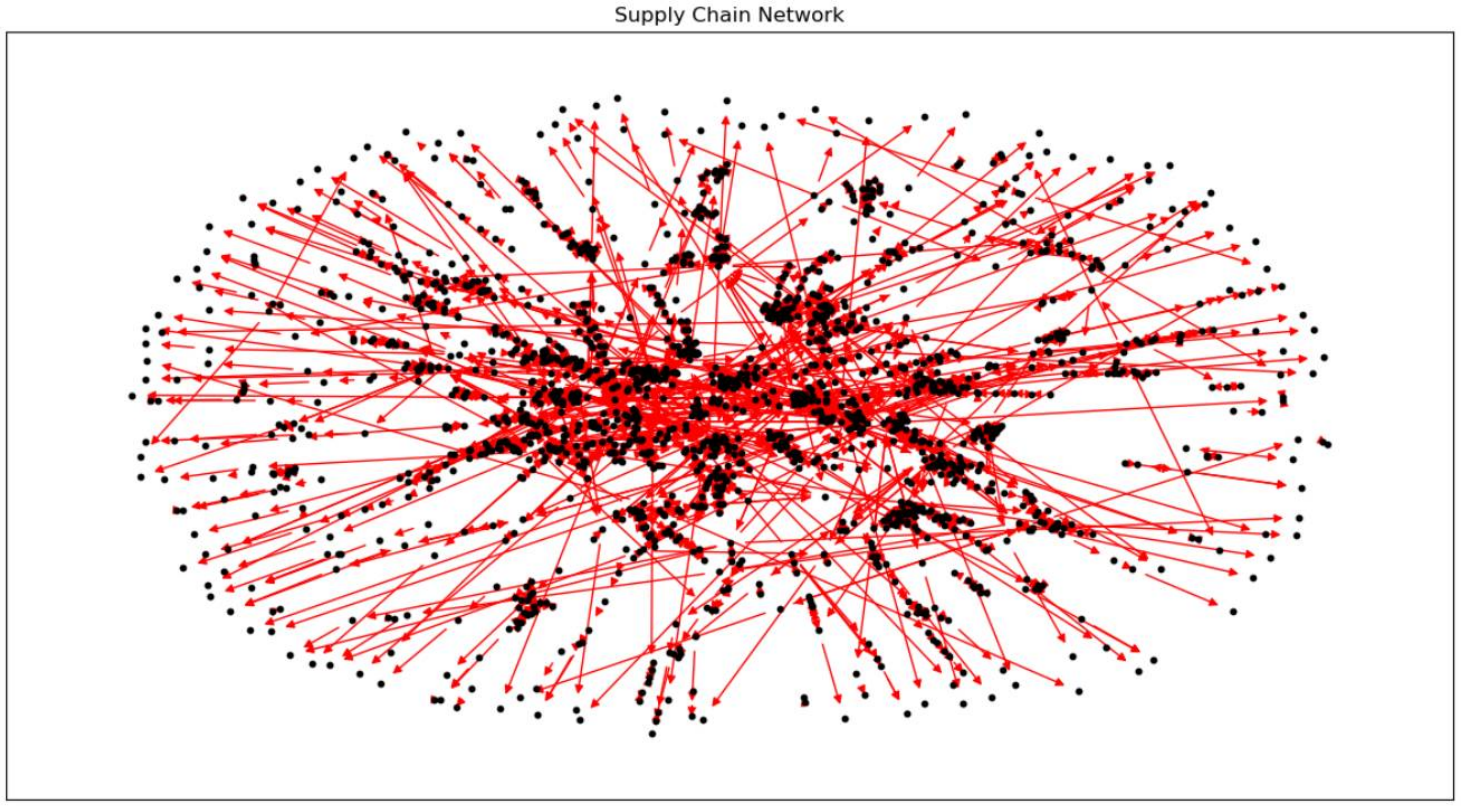
[35]:
```
(1481, 1)
```

```
ax = (destination["count"]).plot.hist(figsize=(4, 4))
```



4. Supply Chain Network from Source to Destination:

```
source = list(table["source_center"])
```



5. Analysis of start Scan to End Scan:

```
time = table['start_scan_to_end_scan']
```

[38]:
```
0          86.0
1          86.0
2          86.0
3          86.0
4          86.0
           ...
144862    427.0
144863    427.0
144864    427.0
144865    427.0
144866    427.0
Name: start_scan_to_end_scan, Length: 144867, dtype: float64
```

5.1 Variability Estimates of Time from start to End.

```
time.describe()
```

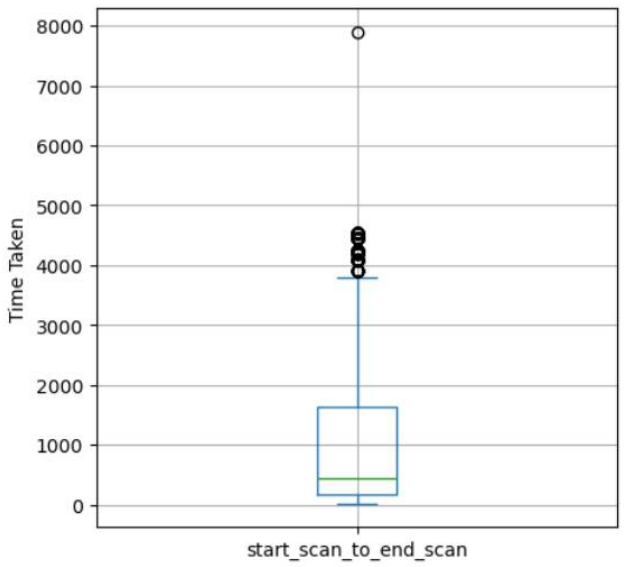
[39]:
```
count    144867.000000
mean        961.262986
std        1037.012769
min          20.000000
25%         161.000000
50%         449.000000
75%        1634.000000
max        7898.000000
Name: start_scan_to_end_scan, dtype: float64
```

```
```

5.2 Data Distribution of Start to End Scan time:

```
```

[40]:
```
Text(0, 0.5, 'Time Taken')
```



5.3 From the above plot we can see :

> 1. The mean time taken is `961 min`.
> 2. The Standard Deviation is `1037min`.
> 3. The 25th, Median, 75th Percentile are `161`, `449`, `1634` mins.
> 4. The Maximum time taken is `7898` min or `31` hrs approx and min time taken is `20` min

> The Visualizations shows there are few outliers suggesting certain products are not available in the nearest centers.

5.4 Frequency Tables and Histograms:

Frequency Table of start to End Scan time:

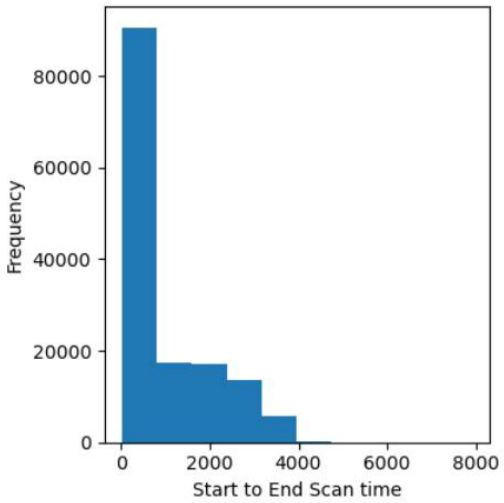
5.41 Frequency Table:

```
•••
```

[41]:
```
start_scan_to_end_scan
(12.122, 807.8]     90702
(807.8, 1595.6]     17403
(1595.6, 2383.4]    17001
(2383.4, 3171.2]    13657
(3171.2, 3959.0]     5754
(3959.0, 4746.8]      349
(7110.2, 7898.0]        1
(4746.8, 5534.6]        0
(5534.6, 6322.4]        0
(6322.4, 7110.2]        0
Name: count, dtype: int64
```

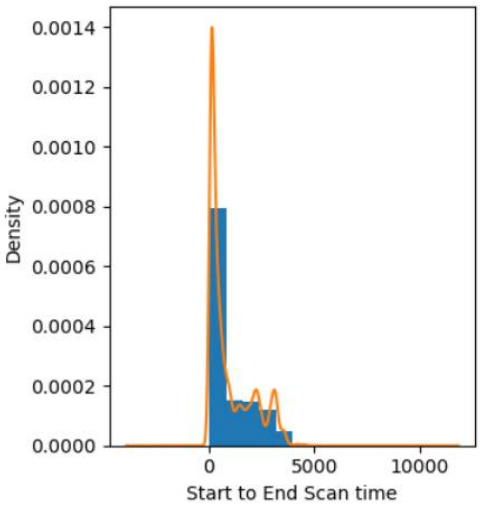
5.42 Histogram :

```
ax = (time).plot.hist(figsize=(4, 4)) •••
```



5.5 Density Estimates:

```
•••
```



6. Difference between actual time and estimated time(OSRM time):

```
table.columns •••
```

[44]:
```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

6.1 Speard of Difference(Table):

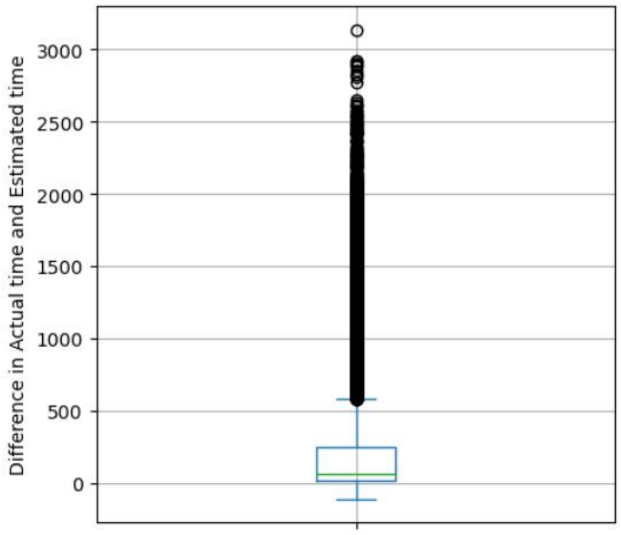
```
diff = pd.Series(table["actual_time"] - table["osrm_time"]) •••
```

[45]:
```
count    144867.000000
mean        203.059254
std         303.743664
min        -110.000000
25%          21.000000
50%          65.000000
75%         247.000000
max        3137.000000
dtype: float64
```

6.2 Vizualization of Spread:

```
•••
```

[46]:
```
Text(0, 0.5, 'Difference in Actual time and Estimated time')
```




```
•••
```

From the above plot it can be seen that :

1. Below 25th Percentile the journey ended before estimated time.
2. Certain routes have shown significant amount of difference of time than the estimated.

```
from datetime import datetime •••
```

[47]:
```
144867
```

```
timediff = pd.DataFrame({"od_start_hr":start_hour,"time_difference":table["actual_time"] - table["osrm_time"]})
```

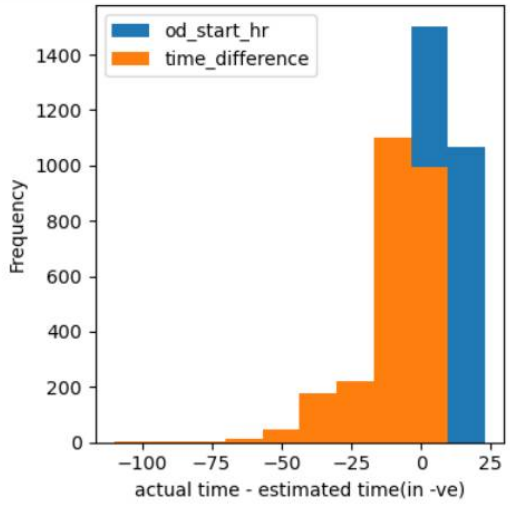
[48]:

	od_start_hr	time_difference
0	3	3.0
1	3	4.0
2	3	12.0
3	3	22.0
4	3	24.0
...	...	...
144862	16	34.0
144863	16	44.0
144864	16	52.0
144865	16	60.0
144866	16	331.0

144867 rows × 2 columns

6.3 Posibility of Error in OSRM Devices:

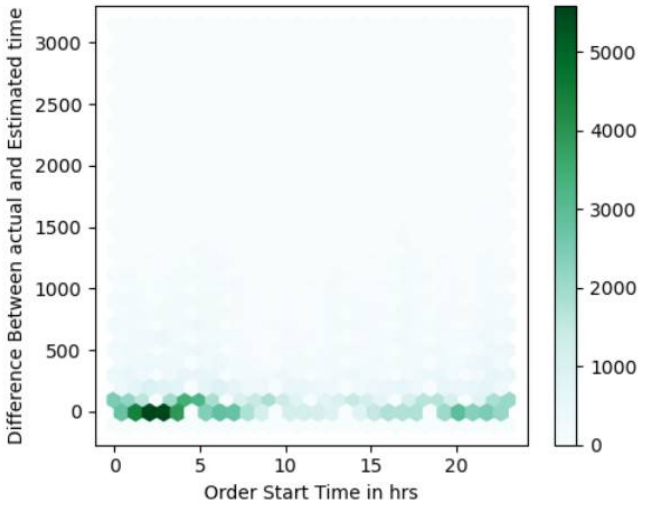
```
ax = timediff[timediff["time_difference"] < 0].plot.hist(figsize = (4,4))
```

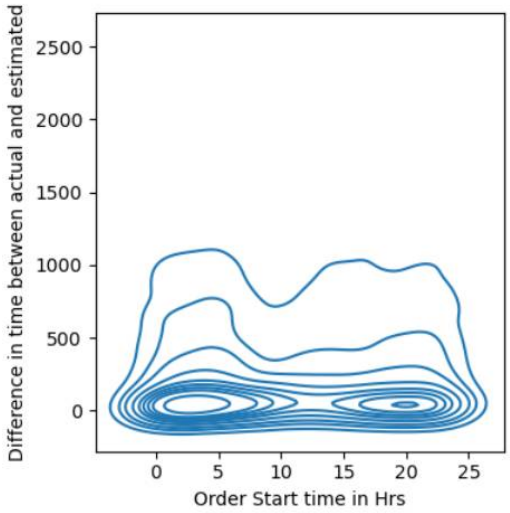


From the histogram above it is evident:

> 1. In certain case the estimated time is greater than actual time.
> 2. The difference went more than `-100` mins. There is a possibility of Error by the system in calculating the estimated time.

7. Relation between Delay and order start hour.





1. Between time `**0 Hrs to 5 Hrs**` and after `20 Hrs` , There is Huge Possibility of Delay.

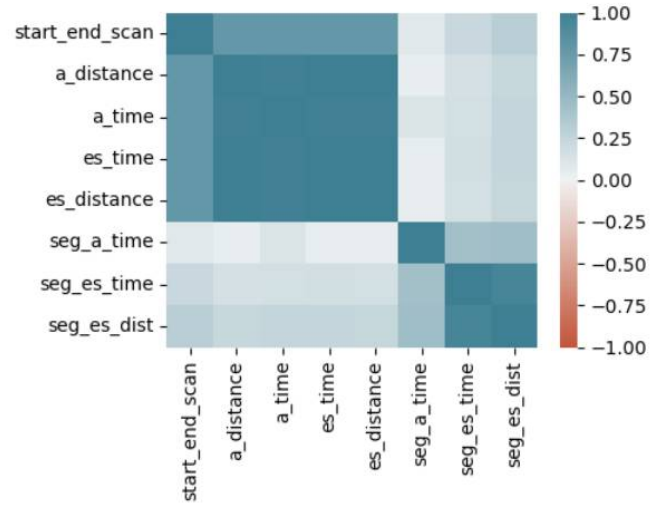
8. Correlation :

```
subset = pd.DataFrame({"start_end_scan":table['start_scan_to_end_scan'],
```

```
subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   start_end_scan  144867 non-null  float64
 1   a_distance      144867 non-null  float64
 2   a_time          144867 non-null  float64
 3   es_time         144867 non-null  float64
 4   es_distance     144867 non-null  float64
 5   seg_a_time      144867 non-null  float64
 6   seg_es_time     144867 non-null  float64
 7   seg_es_dist     144867 non-null  float64
dtypes: float64(8)
memory usage: 8.8 MB
```

```
...
```




```
from matplotlib.collections import EllipseCollection ...
```