main ML / Transportation_Mode_Detection.ipynb

Go to file ···

rajkaran08 Created using Colaboratory Latest commit eed9124 1 minute ago History

1 contributor

7754 lines (7754 sloc) | 501 KB

Raw Blame

Open in Colab

In [80]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [266…

```
import matplotlib.pyplot as plt
import os
import glob
from skimage import io, color
from skimage.feature.texture import greycomatrix, greycoprops
import numpy as np
import pandas as pd
from scipy.stats import kurtosis
from scipy.stats import skew
from scipy.stats import entropy
import cv2
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, callbacks
import PIL
import skimage
% matplotlib inline
```

In [ ]:

```

```

In [267…

```
#Read the CSV file and store in a dataframe called df
df=pd.read_csv("/content/drive/MyDrive/cleaned.csv")
```

In [268…

```
#Print first five row of the dataframe
df.head()
```

Out[268…

	user	timestamp	x	y	z	class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus

In [269…

```
#program to output the number of rows,columns and columns name of input data
rows=len(df.axes[0])
columns=len(df.axes[1])


print("number of rows :",rows)
print("number of columns :",columns)
print("columns_name:")
for col_name in df.columns:
    print(col_name)
```

```
number of rows : 5653053
number of columns : 6
columns_name:
user
timestamp
x
y
z
class
```

In [382…

```
df.shape
```

Out[382…

```
(5653053, 17)
```

In [270…

```
#Program to output the number of unique users present in the dataset
n = len(pd.unique(df['user']))

print("No.of.unique values of user :",
      n)
```

```
No.of.unique values of user : 32
```

In [271…

```
df.describe()
```

Out[271…

	x	y	z
count	5.653053e+06	5.653053e+06	5.653053e+06
mean	1.499442e+00	1.483885e+00	2.484874e+00
std	4.657316e+00	6.262899e+00	5.800348e+00
min	-7.321000e+01	-7.840000e+01	-7.844000e+01
25%	-1.300000e+00	-1.790000e+00	-9.600000e-01
50%	7.100000e-01	2.130000e+00	3.500000e+00
75%	4.650000e+00	6.260000e+00	7.320000e+00
max	7.840000e+01	7.834000e+01	7.840000e+01

Q2 Determine the number of unique sequences

In [272…

```
# Approached this problem by shifting the user column by 1 row and stored it in a different column name
#so that when we will encounter a different user it is a possible case of unique sequence
import time
begin = time.time()
df['next_user']=df.user.shift(1)
df.head()
```

Out[272…

	user	timestamp	x	y	z	class	next_user
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73

In [273…

```
# Similarly shifted the timestamp column by 1 row and stored it in a different column name
df['next_time']=df.timestamp.shift(1)
df.head()
```

Out[273…

	user	timestamp	x	y	z	class	next_user	next_time
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaN
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228

In [274…

```
#Similarly shifted the class column to obtain a possible sequence
df['next_class']=df['class'].shift(1)
df.head()
```

Out[274…

	user	timestamp	x	y	z	class	next_user	next_time	next_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaN	NaN
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus

3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus

In [275…

```
# To check where a particular user change in our given dataframe did the following giving us a true or false value
df['different_user']=df['next_user']!=df['user']
df.head()
```

Out[275…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	differen
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaN	NaN	
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	

In [276…

```
#To check where a particular class change in our given dataframe did the following giving us a true or false value
df['different_class']=df['next_class']!=df['class']
df.head()
```

Out[276…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	differen
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaN	NaN	
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	

In [277…

```
# since  timestamp was originally of object datatype and here we changed it into datetime datatype and took difference of timestamps and changed it into se
from datetime import datetime
import time
from dateutil.relativedelta import relativedelta
#df['times']= datetime.strptime(df['timestamp'], "%d-%m-%Y %H:%M:%S")
df.dtypes
df['timestamp']=pd.to_datetime(df['timestamp'])
df['next_time']=pd.to_datetime(df['next_time'])
df.dtypes
df['diff']=(df['timestamp']-df['next_time']).dt.total_seconds()
df.head()


#df['secs'] = df['next_time'].dt.total_seconds()
#df['diff_sec']=df.next_time-df.timestamp
#df['diff_sec']=df.diff_sec/np.timedelta64(1,'S')
```

Out[277…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	

In [278…

```
# INTUITION: Since we can get a unique sequence if we have either a different user or different class or time difference greater than 10 seconds
df['res']=((df['different_user']) | (df['different_class']) | (df['diff']>10))
df.head()
```

Out[278…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False

In [279…

```
# Count of all true value in the res column give us the number of unique sequences present for each transportation mode for each user
df['res'].value_counts()
```

Out[279…

```
False    5649562
True        3491
Name: res, dtype: int64
```

In [455…

```
end = time.time()
print(f"Total runtime of the program is {end - begin}")
```

```
Total runtime of the program is 1415.4215002059937
```

In [280…

```
df['class'].unique()
```

Out[280…

```
array(['bus', 'walk', 'car', 'bike', 'train', 'e-bike'], dtype=object)
```

In [281…

```
# Used groupby function in two columns so that we can combine the data and this will combine unique sequence for each user for each transporation mode
# stored it in df2 dataframe
df2= df.groupby(['user','class']).res.sum()
```

In [282…

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
```

In [283…

```
df2
```

Out[283…

```
user                                      class
a2d80ed662f34d32951eb1c6ed076c313e358b73  bus        13
a526f3566e9c9024dfa7378eb4291d787a09fd37  car        14
                                          walk       19
a59868c6eb3645eedbb343ce8c336ec6f2ef2324  bike       36
                                          bus        18
                                          car        67
                                          walk       10
a92dee88f61123f923dccec01eeecf1a81953b36  bus         3
ac4c17afeb69b39169eb301ab592696a8f353976  car        20
                                          walk       58
adaaae1a67ea9e43abd60ba945eccda0cb8821e0  bus        20
                                          car        17
                                          walk       10
b138d165100ef60bc793cac143742eb5aea4d6ba  car         9
b45157069942d01310c3e7b74034166717bb25f9  car         3
                                          walk        2
b7b165e5637b5a0226068d907748f4bbfc61a320  car       194
                                          walk       10
c453226e3616ae821cdcb38f38481c2a20f2482f  bike      169
                                          bus       289
c5702d34b238fe68683f818e82cd3a3cd8a16366  bike       15
                                          bus        13
                                          walk      129
ca7950f223a8037b897d0547075dc138f9e43b20  walk        3
cace4ec0999436917986b4fa6e9317262c897bc2  car        72
cbde60baea002b694ecf2a3ff2d95be16b00efe1  bus         4
ce39f5d0705695fcd70a04ba6d84ac6beecd6f9c  bus        22
                                          car        80
d429974540bfd38c3367fe9f0c8682775ff4fa18  bus         6
                                          car       131
                                          walk        2
d7a1230d94f91a32cc079809748e52e8a4a6a22f  bike      151
                                          train      12
                                          walk        6
d7dd12d83c81574137f858034b99f4cc83ab0718  car       147
d8c047eaaee204b7b5cd71e2d67308b87b038ed3  car        12
```

```
                                          walk        24
dc0bdce306ec3b624fe0e6ecd1ffbd82cb970120  bike       113
                                          car         32
                                          walk         5
dd82e3df4bebc74ed6b67877be79e29f401c16a3  car         68
                                          walk         2
dde95e125d89843f7032baa734ee4d34ec775aaf  bus          5
                                          car          2
                                          walk         8
de9892b879c83ea3d24fb4560873107cc4e86d48  car        114
                                          walk        36
dfcfc0404691b73b69884073159f90843f2ac35b  bus         50
                                          car        108
                                          walk       154
e429a95c532f1117130c11e4a18379d84fa4ffa9  bus         40
                                          car         38
eb9e7854290fd6ea9ebaf448b640fc1f1dbeb076  bus          2
                                          train        1
                                          walk         2
ecfb0929250fb6dda66a4065441230ab27f094e5  car        157
                                          e-bike      16
                                          train        1
ed623d28c1e0071632a6110b8f8ed93f8af78b99  bus         10
                                          car        117
                                          walk         4
f1b7331b66e404c11eebb22933e733117bbb12c9  bike       172
                                          car        139
                                          walk        73
f5edd999397145a2ec1b244226fc83f99631760c  bus         16
                                          walk        13
f7ae1ce141c26db40ea8b090fb568a0c965310aa  car          2
faae5be800be2dfa897eea0bd2e5988cd53c4ec0  bike       136
                                          car         10
                                          walk        35
Name: res, dtype: int64
```

In [384…

```python
#the time taken for this code to run this particular section of code

import time
begin = time.time()
df['next_user']=df.user.shift(1)

df['next_time']=df.timestamp.shift(1)

df['next_class']=df['class'].shift(1)

df['different_user']=df['next_user']!=df['user']

df['different_class']=df['next_class']!=df['class']
df.head()
from datetime import datetime
import time
from dateutil.relativedelta import relativedelta
#df['times']= datetime.strptime(df['timestamp'], "%d-%m-%Y %H:%M:%S")
df.dtypes
df['timestamp']=pd.to_datetime(df['timestamp'])
df['next_time']=pd.to_datetime(df['next_time'])
df.dtypes
df['diff']=(df['timestamp']-df['next_time']).dt.total_seconds()
df.head()
df['res']=((df['different_user']) | (df['different_class']) | (df['diff']>10))
df.head()
df['res'].value_counts()
end = time.time()
print(f"Total runtime of the program is {end - begin}")
```

```
Total runtime of the program is 2.167815923690796
```

Time Window Partition

In [386…

```python
import time
begin = time.time()

df['newres']=df['res'].cumsum()
df.head()
```

Out[386…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False	False
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False	False

In [398…

```python
#Created a dataframe that will store the minimum timestamp value for each different sequence
df3= df.groupby(['newres'])['timestamp'].min()
df3.head(20)
```

Out[398…

```
newres
1    2018-06-04 16:26:55.053
2    2018-06-04 16:28:14.647
3    2018-06-04 16:28:33.051
4    2018-06-04 16:31:41.981
5    2018-06-04 16:31:58.785
6    2018-06-04 16:32:43.420
7    2018-06-04 16:33:44.741
8    2018-06-04 16:35:20.106
9    2018-06-04 16:36:40.674
10   2018-06-04 16:38:14.326
11   2018-06-04 16:38:47.604
12   2018-06-04 16:39:49.194
13   2018-06-04 16:41:34.783
14   2018-04-10 12:37:51.251
15   2018-04-10 12:39:51.572
16   2018-04-10 12:41:48.454
17   2018-04-10 12:44:01.581
18   2018-04-10 13:26:28.061
19   2018-04-10 13:29:17.771
20   2018-04-10 13:29:54.214
Name: timestamp, dtype: datetime64[ns]
```

In [388…

```python
df3.size
```

Out[388…

```
3491
```

In [402…

```python
df.head()
```

Out[402…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False	False
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False	False

In [404…

```python
# created a new column which will store the minimum timestamp value for each sequence in every row
df['min_time'] = df.groupby('newres').timestamp.transform('min')
df.head()
```

Out[404…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False	False

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False	False

In [405…

```
# A column that contain time diffrence of timestamp and minimum time stamp for each sequence
df['diff_t']=(df['timestamp']-df['min_time']).dt.total_seconds()
df.head()
```

Out[405…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False	False
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False	False

In [406…

```
# Since we have a huge dataset so instead of using a for loop used a function name 'divide' which is used to compute the value in the form of x@y
# as a form of string which seems feasible  because we don't have to change the value of new sequence if our user changes
def divide(n,t):
  return str(n)+'@'+str(int(t/5))
```

In [407…

```
# Created a column name ans which store our sequence using the divide function in each two values in the same row and for this we used np.vectorize functic
df['ans']=np.vectorize(divide)(df['newres'],df['diff_t'])
df.head()
```

Out[407…

	user	timestamp	x	y	z	class	next_user	next_time	next_class	different_user	different_class
0	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	0.78	-9.13	-3.74	bus	NaN	NaT	NaN	True	True
1	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	0.79	-9.11	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.053	bus	False	False
2	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	0.80	-9.12	-3.75	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.111	bus	False	False
3	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	0.78	-9.14	-3.76	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.169	bus	False	False
4	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.286	0.83	-9.12	-3.80	bus	a2d80ed662f34d32951eb1c6ed076c313e358b73	2018-06-04 16:26:55.228	bus	False	False

In [408…

```
# All the unique values in ans column give us the  number of sequences for each transportation mode keeping the time window length as 5 seconds
df['ans'].unique()
df['ans'].nunique()
```

Out[408…

```
40258
```

In [409…

```
end = time.time()
print(f"Total runtime of the program is {end - begin}")
```

```
Total runtime of the program is 984.6934185028076
```

Feature Extraction

In [410…

```
# Created a dataframe having column name as all features and Used groupby function for two rows in the original dataframe
df4 = pd.DataFrame(columns = ['x_min', 'y_min','z_min','x_max','y_max','z_max','x_mean','y_mean','z_mean','x_std','y_std','z_std'])
df4['x_min']= df.groupby(['ans'])['x'].min()
df4['y_min']= df.groupby(['ans'])['y'].min()
df4['z_min']= df.groupby(['ans'])['z'].min()
df4['x_max']= df.groupby(['ans'])['x'].max()
df4['y_max']= df.groupby(['ans'])['y'].max()
df4['z_max']= df.groupby(['ans'])['z'].max()
df4['x_mean']= df.groupby(['ans'])['x'].mean()
df4['y_mean']= df.groupby(['ans'])['y'].mean()
df4['z_mean']= df.groupby(['ans'])['z'].mean()
df4['x_std']= df.groupby(['ans'])['x'].std()
df4['y_std']= df.groupby(['ans'])['y'].std()
df4['z_std']= df.groupby(['ans'])['z'].std()

df4['class'] = df.groupby(['ans'])['class'].unique()
df4.head(100)
```

Out[410…

	x_min	y_min	z_min	x_max	y_max	z_max	x_mean	y_mean	z_mean	x_std	y_std	z_std	class
ans													
1000@0	-8.72	-31.190001	-19.719999	7.910000	4.28	11.560000	1.351463	-9.829594	0.169878	2.837040	6.414270	4.783370	[walk]
1000@1	-6.66	-32.990002	-16.870001	7.980000	6.43	13.310000	1.319355	-9.822137	-0.065282	2.913423	5.937518	4.439168	[walk]
1000@10	-6.12	-25.910000	-14.420000	9.670000	2.35	12.740000	1.523306	-9.898000	-0.099388	2.806146	4.795777	4.374881	[walk]
1000@11	-7.81	-18.620001	-12.650000	8.330000	-2.02	10.160000	1.257056	-9.735202	0.219153	2.660034	3.916937	3.897041	[walk]
1000@12	-6.55	-22.010000	-14.580000	12.900000	-1.34	11.090000	1.585823	-9.894980	-0.005181	2.982561	3.993989	4.243417	[walk]
1000@13	-7.55	-25.620001	-13.480000	8.270000	0.26	14.420000	1.449597	-9.845524	-0.031734	2.636352	4.384765	4.340549	[walk]
1000@14	-10.05	-20.620001	-13.710000	9.800000	-1.84	9.340000	1.587258	-9.673064	-0.092581	2.707403	3.918021	4.031544	[walk]
1000@15	-9.22	-19.549999	-15.250000	10.270000	-0.69	8.640000	1.552661	-9.739113	0.088508	2.935777	4.043636	4.079651	[walk]
1000@16	-8.97	-21.040001	-17.629999	10.990000	1.24	12.720000	1.363077	-9.702846	0.004923	3.163096	5.097840	4.680512	[walk]
1000@17	-9.01	-21.650000	-14.530000	11.140000	-1.06	9.860000	1.234124	-9.542165	-0.037577	2.737257	4.307967	4.288275	[walk]
1000@18	-7.58	-22.459999	-12.180000	8.330000	-0.82	11.700000	1.511538	-10.085897	0.344231	2.937179	4.396953	4.000998	[walk]
1000@19	-3.85	-19.350000	-11.900000	7.540000	0.57	14.960000	1.618280	-10.205591	0.754301	2.453276	4.099339	3.604290	[walk]
1000@2	-7.23	-34.680000	-17.980000	8.470000	6.18	13.160000	1.267742	-9.860444	0.104476	2.798923	6.852062	5.096520	[walk]
1000@20	-8.30	-24.370001	-17.120001	10.700000	1.09	14.650000	1.415000	-9.407051	0.597692	3.440858	5.306618	4.770315	[walk]
1000@3	-12.39	-34.139999	-16.299999	8.250000	3.95	14.330000	1.258629	-9.859597	-0.080645	2.725205	6.523774	5.180010	[walk]
1000@4	-9.00	-29.370001	-14.180000	9.260000	7.39	14.770000	1.439798	-9.846613	0.100242	3.252334	6.418673	4.599581	[walk]
1000@5	-8.78	-29.700001	-15.630000	7.910000	4.94	13.920000	1.200403	-9.923589	-0.102258	2.831261	6.463504	4.866365	[walk]
1000@6	-9.57	-33.599998	-17.770000	7.750000	5.20	12.610000	1.165202	-10.083387	0.032056	2.987304	7.079593	5.216695	[walk]
1000@7	-7.44	-28.350000	-17.360001	7.790000	7.20	11.620000	1.272500	-9.800484	0.098548	2.731310	6.526001	4.857404	[walk]
1000@8	-10.79	-30.969999	-17.770000	7.740000	4.54	11.050000	1.190607	-9.739028	-0.018219	2.785000	6.505979	4.984922	[walk]
1000@9	-6.83	-31.170000	-17.520000	9.290000	6.92	10.150000	1.224032	-9.894153	-0.154153	2.879435	6.640505	4.734818	[walk]
1001@0	-9.22	-22.670000	-17.620001	10.280000	-0.15	12.930000	1.545401	-9.722166	0.271684	2.918830	4.746429	4.418703	[walk]
1001@1	-6.31	-23.469999	-20.660000	8.940000	-0.20	10.690000	1.325200	-9.890900	-0.172800	2.662631	5.152335	4.472909	[walk]
1001@2	-6.69	-27.030001	-15.560000	9.100000	4.64	12.360000	0.625556	-9.934222	-0.269111	3.460391	7.433849	5.758431	[walk]
1001@4	-8.56	-30.469999	-17.260000	9.610000	4.05	10.720000	1.324626	-10.074907	-0.056028	3.278706	6.718057	5.067422	[walk]
1001@5	-8.67	-36.549999	-15.520000	8.120000	2.82	11.390000	1.482857	-9.607429	0.255714	2.814466	6.737617	4.752500	[walk]
1001@6	-6.76	-26.930000	-16.700001	7.150000	6.04	11.770000	1.089875	-9.752875	0.622750	2.826452	6.669008	4.333534	[walk]
1001@7	-5.83	-23.389999	-17.799999	12.000000	-0.39	10.960000	1.670685	-10.501781	-0.672877	3.229523	5.785751	5.025383	[walk]
1001@8	-3.26	-22.030001	-5.170000	6.630000	2.32	10.150000	1.852353	-8.872941	1.531176	3.050706	5.722982	3.996237	[walk]
1002@0	-4.97	-31.230000	-12.960000	8.520000	3.63	7.440000	1.400000	-9.995806	-0.166452	2.867604	7.110198	4.359786	[walk]
1003@0	-7.92	-30.840000	-16.370001	21.510000	2.63	10.810000	5.254851	-8.446700	-0.153399	4.639914	5.176996	4.519487	[walk]
1003@1	-6.96	-29.570000	-17.840000	19.980000	1.42	9.840000	4.816653	-8.891229	0.087542	4.547480	5.449697	4.412200	[walk]
1003@10	-12.45	-34.180000	-27.639999	12.250000	5.11	14.030000	1.522500	-9.933508	0.083427	2.925182	7.054544	5.394783	[walk]
1003@11	-7.48	-29.340000	-21.020000	10.840000	7.48	12.110000	1.414234	-9.920887	-0.065726	2.928952	6.558600	5.060152	[walk]
1003@12	-5.91	-30.900000	-28.990000	6.610000	4.34	9.400000	1.253505	-9.962062	0.047835	2.410469	6.512385	4.867195	[walk]

1003@12	-5.91	-30.900000	-28.990000	6.610000	4.34	9.400000	1.253503	-9.962062	-0.047835	2.410469	6.312385	4.867195	[walk]
1003@13	-3.11	-14.970000	-23.780001	33.290001	6.44	9.470000	6.512136	-2.895534	1.806019	5.495314	5.613156	4.592010	[walk]
1003@14	-7.60	-35.860001	-17.830000	17.660000	6.84	14.040000	3.234940	-9.096185	-0.056265	3.856609	6.381855	4.604284	[walk]
1003@15	-6.93	-29.510000	-20.480000	9.000000	2.21	11.560000	1.713508	-9.849798	-0.044637	2.751527	6.329608	4.848556	[walk]
1003@16	-6.37	-31.969999	-20.040001	8.540000	3.81	12.100000	1.440202	-9.858468	0.012298	2.702928	6.662632	4.906110	[walk]
1003@17	-11.50	-31.830000	-18.660000	9.800000	10.54	11.830000	1.381371	-9.856492	0.046169	3.110281	6.923996	4.927380	[walk]
1003@18	-9.43	-31.080000	-19.980000	8.900000	8.85	10.940000	1.296976	-10.001169	0.249758	2.875444	6.882999	4.927502	[walk]
1003@19	-6.99	-30.790001	-22.500000	9.310000	7.30	13.480000	1.286327	-9.902408	0.222980	3.017875	6.690399	4.906197	[walk]
1003@2	-8.87	-35.310001	-18.139999	17.320000	5.33	13.800000	4.534000	-9.320125	0.145375	4.557064	6.078447	5.028529	[walk]
1003@20	-9.90	-33.320000	-10.300000	9.620000	6.23	10.960000	0.995455	-9.650950	-0.140041	3.106601	4.735822	3.211218	[walk]
1003@21	-11.69	-36.240002	-21.860001	10.530000	10.60	16.340000	0.862576	-10.100742	0.529258	3.600295	8.842710	6.174799	[walk]
1003@22	-9.40	-36.610001	-22.250000	10.560000	7.80	16.100000	1.014244	-9.812143	0.682773	3.586584	8.233498	6.391271	[walk]
1003@23	-8.52	-38.900002	-20.500000	11.520000	8.32	17.080000	0.973891	-10.281046	0.388284	3.527664	8.351120	6.203314	[walk]
1003@24	-6.57	-36.279999	-26.660000	10.180000	7.90	12.410000	1.169583	-10.067500	0.522375	3.365069	8.310656	5.954876	[walk]
1003@25	-7.08	-34.849998	-26.290001	9.800000	5.99	12.680000	1.295353	-9.955270	0.328257	3.020221	7.915143	5.799307	[walk]
1003@26	-10.73	-34.650002	-21.059999	9.340000	4.45	11.930000	1.133083	-10.127000	0.254292	3.119538	7.348313	5.275707	[walk]
1003@27	-7.68	-33.959999	-22.760000	10.850000	5.21	12.650000	1.080610	-10.222642	0.555610	3.323252	7.644687	5.499181	[walk]
1003@28	-11.45	-29.959999	-21.850000	11.430000	6.27	13.600000	1.069958	-9.833445	0.479832	3.224029	7.167602	5.576016	[walk]
1003@29	-6.58	-35.189999	-24.049999	9.680000	7.42	12.780000	1.011905	-9.976349	0.394167	3.008342	8.005429	5.898026	[walk]
1003@3	-7.75	-31.270000	-19.790001	19.750000	3.77	10.720000	4.129150	-9.176194	0.170081	4.142596	6.077257	4.565079	[walk]
1003@30	-9.17	-36.709999	-26.450001	9.890000	5.36	13.850000	1.076638	-9.833702	0.429787	3.253279	7.856714	5.873663	[walk]
1003@31	-9.04	-37.880001	-29.770000	11.190000	10.26	14.090000	0.956891	-10.191975	0.379496	3.282263	8.220851	5.785993	[walk]
1003@32	-9.07	-36.860001	-21.770000	10.090000	9.24	12.210000	0.970723	-9.875149	0.564213	3.256790	8.162234	5.647164	[walk]
1003@33	-10.33	-35.400002	-25.480000	12.100000	9.76	18.780001	1.200548	-9.862055	0.697717	3.764429	8.272984	5.969679	[walk]
1003@34	-7.81	-37.209999	-27.090000	10.220000	9.74	12.940000	1.094872	-10.173291	0.436026	3.294101	8.784093	6.163497	[walk]
1003@35	-9.09	-38.400002	-26.020000	9.050000	11.37	19.350000	1.212092	-10.124352	0.550460	3.310869	8.454007	5.819076	[walk]
1003@36	-8.39	-36.910000	-21.520000	10.690000	11.73	13.550000	1.048809	-10.321404	0.708553	3.309730	8.191274	5.330385	[walk]
1003@37	-10.78	-34.639999	-23.680000	11.370000	9.79	13.080000	1.029831	-10.179536	0.143671	3.403468	7.897519	5.706511	[walk]
1003@38	-7.57	-35.259998	-28.059999	10.260000	8.55	15.490000	1.028697	-9.964076	0.363782	3.153941	7.825857	5.551633	[walk]
1003@39	-7.23	-33.660000	-26.170000	10.430000	8.86	13.220000	1.077906	-10.085085	0.194402	3.048158	8.161063	6.001492	[walk]
1003@4	-8.09	-34.970001	-18.930000	13.770000	4.45	10.400000	3.957085	-9.476883	-0.265101	3.603889	5.999715	4.810912	[walk]
1003@40	-5.94	-30.360001	-25.920000	21.030001	7.44	11.800000	5.835720	-4.844691	1.063951	4.662760	6.968093	5.205246	[walk]
1003@41	-6.36	-29.770000	-12.430000	17.250000	2.15	10.250000	5.506113	-7.735870	-0.590202	3.961388	4.760114	3.883715	[walk]
1003@42	-10.07	-32.430000	-16.590000	9.920000	5.85	12.230000	1.736290	-9.845363	-0.195766	3.070319	6.585942	5.145164	[walk]
1003@43	-7.17	-29.719999	-19.139999	8.230000	8.14	11.960000	1.502097	-9.949758	0.065323	3.010763	6.472896	5.033936	[walk]
1003@44	-11.15	-38.400002	-16.540001	8.700000	6.00	12.130000	1.635444	-9.843992	0.116532	3.047945	6.337273	5.038037	[walk]
1003@45	-16.15	-32.500000	-19.200001	8.690000	7.07	16.809999	1.631365	-9.790321	0.138675	3.432583	6.331595	5.260748	[walk]
1003@46	-9.04	-25.309999	-16.950001	8.810000	5.73	13.110000	1.539837	-9.842439	-0.059024	2.863563	6.113113	4.970523	[walk]
1003@47	-8.42	-29.389999	-17.340000	10.220000	5.82	13.040000	1.383640	-9.735063	0.010377	3.082437	6.232977	4.952950	[walk]
1003@48	-7.17	-31.160000	-13.490000	9.560000	5.96	11.350000	1.615825	-10.221845	-0.107476	2.938548	6.601224	4.482131	[walk]
1003@49	-6.72	-24.900000	-13.380000	9.960000	7.82	8.280000	1.529754	-9.366066	-0.234344	3.021496	5.482102	4.018227	[walk]
1003@5	-10.08	-33.450001	-18.410000	21.090000	8.89	11.540000	3.393024	-9.557137	0.027540	4.084231	6.533701	4.975209	[walk]
1003@50	-7.09	-33.740002	-14.650000	10.820000	9.89	11.160000	1.790426	-9.995489	-0.450298	3.110617	5.943837	4.787533	[walk]
1003@51	-9.00	-31.290001	-13.750000	10.770000	8.62	12.310000	1.518678	-10.013760	-0.436033	3.282385	6.240728	4.594989	[walk]
1003@52	-12.82	-31.889999	-12.600000	10.490000	7.90	16.250000	1.293381	-9.424810	-0.439048	3.327873	5.901852	4.357428	[walk]
1003@53	-4.04	-21.540001	-14.400000	7.460000	10.44	9.830000	1.389500	-9.705000	-0.428500	3.284085	6.383304	4.737353	[walk]
1003@6	-6.62	-34.240002	-19.410000	12.590000	6.42	12.580000	2.874170	-9.664696	-0.076154	3.293203	6.106935	4.940512	[walk]
1003@7	-7.07	-36.790001	-19.580000	9.350000	6.15	12.710000	2.151393	-9.719467	0.128033	2.939624	6.918558	5.248515	[walk]
1003@8	-7.03	-35.400002	-18.969999	9.740000	3.37	10.500000	1.745579	-9.679545	0.001570	2.644976	6.324190	4.696757	[walk]
1003@9	-6.33	-27.760000	-16.040001	10.320000	5.10	11.910000	1.624699	-9.698474	-0.048153	2.829313	5.716571	4.707166	[walk]
1004@0	-13.77	-30.250000	-17.040001	9.150000	7.07	19.350000	1.056890	-9.540669	-0.017756	3.120469	6.587647	5.108968	[walk]
1005@0	-6.92	-34.520000	-18.709999	8.280000	13.79	15.330000	1.047290	-9.914466	-0.057443	2.551610	7.731742	5.309274	[walk]
1005@1	-7.57	-29.610001	-18.270000	9.160000	4.98	11.410000	1.250755	-10.132547	0.010377	2.687888	6.531593	4.843951	[walk]
1005@2	-1.70	-22.230000	-16.740000	8.910000	-0.53	6.200000	1.730571	-10.190857	-0.747429	2.462238	5.648424	4.821575	[walk]
1005@3	-6.87	-31.840000	-21.480000	6.930000	9.79	12.280000	0.737683	-9.300854	-0.301463	2.914951	8.139942	5.836409	[walk]
1005@4	-8.03	-31.280001	-20.850000	7.350000	6.43	12.760000	1.186118	-10.073882	-0.296447	2.621802	7.545656	5.755690	[walk]
1005@5	-6.44	-26.280001	-15.740000	8.760000	3.95	11.450000	1.082000	-10.603556	-0.057778	3.099995	6.106422	4.969593	[walk]
1006@0	-10.34	-33.529999	-14.760000	9.450000	8.69	12.810000	1.032092	-9.418418	0.264592	2.927369	6.387570	4.550067	[walk]
1007@0	-9.59	-36.880001	-24.910000	9.270000	6.23	16.120001	1.065537	-10.192397	0.210689	2.976694	7.399382	5.489272	[walk]
1007@1	-10.49	-38.770000	-23.040001	7.980000	6.53	12.790000	0.968219	-10.134696	0.206761	2.852490	7.785552	5.494403	[walk]
1007@10	-10.24	-33.119999	-18.150000	7.820000	4.87	13.210000	0.991296	-10.070324	0.221336	2.779587	6.831902	5.208843	[walk]
1007@11	-6.07	-36.060001	-20.799999	7.570000	4.34	11.010000	1.021429	-9.978277	0.131050	2.472115	6.940367	5.189739	[walk]
1007@12	-10.29	-30.799999	-15.950000	8.050000	4.24	11.850000	1.000980	-9.876367	0.271878	2.850533	6.699176	5.100455	[walk]
1007@13	-8.29	-34.270000	-17.250000	14.510000	11.19	12.330000	1.114730	-9.916100	-0.008963	3.149350	7.554902	5.563938	[walk]
1007@14	-10.94	-31.510000	-19.520000	8.130000	6.80	13.360000	0.814128	-10.083021	0.189277	3.002847	6.625419	5.063528	[walk]
1007@15	-6.70	-30.520000	-18.750000	7.660000	3.50	11.000000	0.909095	-10.094397	0.159569	2.540303	6.256181	4.755093	[walk]

In [411…

```
def removeBox(x):
  return x[0]
```

In [412…

```
df4['class'] = np.vectorize(removeBox)(df4['class'])
```

In [413…

```
df4.head()
```

Out[413…

	x_min	y_min	z_min	x_max	y_max	z_max	x_mean	y_mean	z_mean	x_std	y_std	z_std	class
ans													
1000@0	-8.72	-31.190001	-19.719999	7.91	4.28	11.56	1.351463	-9.829594	0.169878	2.837040	6.414270	4.783370	walk
1000@1	-6.66	-32.990002	-16.870001	7.98	6.43	13.31	1.319355	-9.822137	-0.065282	2.913423	5.937518	4.439168	walk
1000@10	-6.12	-25.910000	-14.420000	9.67	2.35	12.74	1.523306	-9.898000	-0.099388	2.806146	4.795777	4.374881	walk
1000@11	-7.81	-18.620001	-12.650000	8.33	-2.02	10.16	1.257056	-9.735202	0.219153	2.660034	3.916937	3.897041	walk
1000@12	-6.55	-22.010000	-14.580000	12.90	-1.34	11.09	1.585823	-9.894980	-0.005181	2.982561	3.993989	4.243417	walk

In [414…

```
df4['x_min'].size
```

Out[414…

```
40258
```

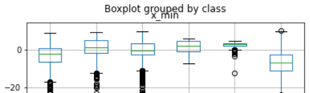
Boxplot and Whisker Plot

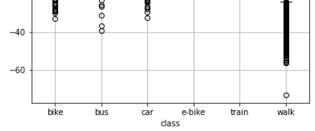
In [415…

```
df4.boxplot(by='class',column=['x_min'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
<matplotlib.axes._subplots.AxesSubplot at 0x7f46577b2390>
```

Out[415…





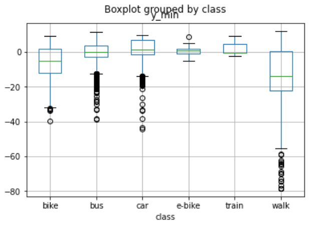
In [416…

```
df4.boxplot(by='class',column=['y_min'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[416…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46c469f1d0>
```



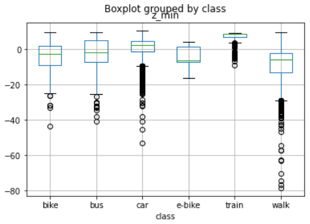
In [417…

```
df4.boxplot(by='class',column=['z_min'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[417…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46577af6d0>
```



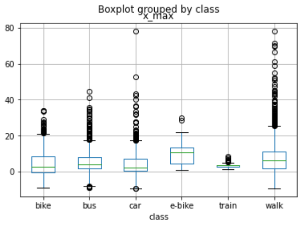
In [418…

```
df4.boxplot(by='class',column=['x_max'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[418…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46577fa450>
```



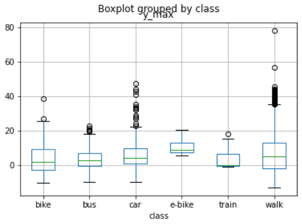
In [419…

```
df4.boxplot(by='class',column=['y_max'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[419…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657899d50>
```



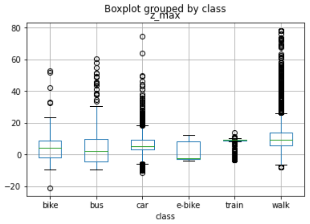
In [420…

```
df4.boxplot(by='class',column=['z_max'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[420…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46578b9250>
```



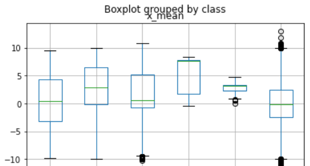
In [421…

```
df4.boxplot(by='class',column=['x_mean'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[421…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657929c50>
```



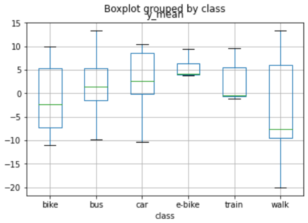
In [422…

```
df4.boxplot(by='class',column=['y_mean'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[422…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f465799f350>
```



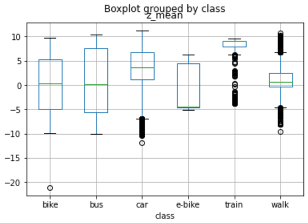
In [423…

```
df4.boxplot(by='class',column=['z_mean'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[423…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657a309d0>
```



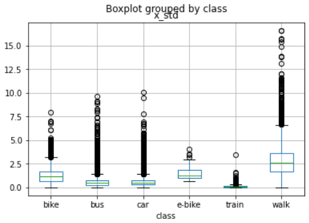
In [424…

```
df4.boxplot(by='class',column=['x_std'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[424…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657a30950>
```



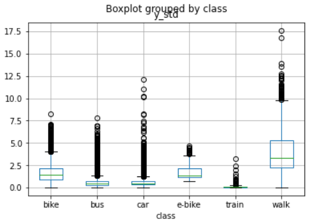
In [425…

```
df4.boxplot(by='class',column=['y_std'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[425…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657ad63d0>
```



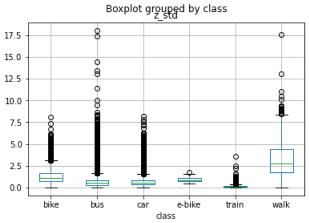
In [426…

```
df4.boxplot(by='class',column=['z_std'])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
ts-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

Out[426…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4657ae6050>
```



Balancing Dataset

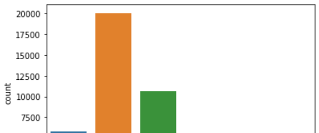
In [427…

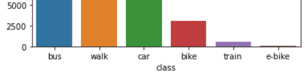
```
import seaborn as sns
```

In [428…

```
# A plot to show the number of different mode of trnasportation
g=sns.countplot(df4['class'])
g.set_xticklabels(['bus', 'walk', 'car', 'bike', 'train', 'e-bike'])
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the onl
t will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```





In [429…

```
df4.head()
```

Out[429…

	x_min	y_min	z_min	x_max	y_max	z_max	x_mean	y_mean	z_mean	x_std	y_std	z_std	class
ans													
1000@0	-8.72	-31.190001	-19.719999	7.91	4.28	11.56	1.351463	-9.829594	0.169878	2.837040	6.414270	4.783370	walk
1000@1	-6.66	-32.990002	-16.870001	7.98	6.43	13.31	1.319355	-9.822137	-0.065282	2.913423	5.937518	4.439168	walk
1000@10	-6.12	-25.910000	-14.420000	9.67	2.35	12.74	1.523306	-9.898000	-0.099388	2.806146	4.795777	4.374881	walk
1000@11	-7.81	-18.620001	-12.650000	8.33	-2.02	10.16	1.257056	-9.735202	0.219153	2.660034	3.916937	3.897041	walk
1000@12	-6.55	-22.010000	-14.580000	12.90	-1.34	11.09	1.585823	-9.894980	-0.005181	2.982561	3.993989	4.243417	walk

In [438…

```
df4=df4.dropna()
```

In [439…

```
X = df4[['x_min',       'y_min',       'z_min',       'x_max',       'y_max',       'z_max',       'x_mean',       'y_mean',       'z_mean',       'x_
Y=df4['class']
```

In [455…

In [441…

```
# Used Random under sampler to balance the number of different mode of transportation
# Program to output number of data points obtained for each transportation mode in the balanced dataset

import imblearn
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

rus = RandomUnderSampler(random_state=42, replacement=True)
x_new, y_new = rus.fit_resample(X, Y)
print('original dataset shape:', Counter(Y))
print('Resample dataset shape', Counter(y_new))
```

```
original dataset shape: Counter({2: 20033, 1: 10597, 5: 5772, 0: 3123, 4: 597, 3: 77})
Resample dataset shape Counter({0: 77, 1: 77, 2: 77, 3: 77, 4: 77, 5: 77})
```

Training and Split Dataset

In [442…

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df4['class']= label_encoder.fit_transform(df4['class'])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_new, y_new, test_size=0.2,random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25,random_state=1)

print(len(X_train))
print(len(X_val))
print(len(X_test))
```

```
276
93
93
```

SVM

In [443…

```
from sklearn.svm import SVC

z = SVC (kernel='linear')
z.fit(X_train, y_train)
```

Out[443…

```
SVC(kernel='linear')
```

In [444…

```
pre=z.predict(X_test)
pre
```

Out[444…

```
array([4, 1, 1, 0, 4, 0, 1, 3, 0, 5, 0, 0, 4, 2, 4, 0, 1, 4, 0, 3, 4, 1,
       0, 3, 5, 2, 3, 0, 4, 2, 3, 3, 4, 2, 0, 3, 5, 1, 4, 0, 1, 5, 3, 1,
       1, 0, 3, 4, 5, 5, 2, 4, 1, 2, 5, 5, 4, 5, 4, 0, 2, 5, 4, 1, 1, 0,
       1, 3, 4, 4, 3, 3, 5, 2, 1, 3, 0, 5, 3, 3, 1, 2, 5, 5, 3, 5, 4, 1,
       1, 2, 3, 3, 0])
```

In [445…

```
print(y_test)
```

```
330    4
101    1
191    2
66     0
326    4
450    5
225    2
246    3
67     0
407    5
457    5
168    2
366    4
186    2
185    2
4      0
192    2
329    4
47     0
273    3
201    2
146    1
172    2
304    3
447    5
189    2
283    3
31     0
379    4
161    2
245    3
300    3
338    4
309    4
5      0
299    3
431    5
307    3
359    4
17     0
164    2
425    5
292    3
65     0
90     1
132    1
232    3
162    2
386    5
296    3
311    4
323    4
224    2
29     0
453    5
439    5
187    2
427    5
208    2
62     0
117    1
415    5
341    4
218    2
139    1
128    1
102    1
298    3
```

```
340    4
375    4
290    3
409    5
452    5
92     1
214    2
286    3
6      0
411    5
272    3
233    3
180    2
107    1
440    5
403    5
270    3
58     0
347    4
78     1
399    5
267    3
294    3
388    5
392    5
Name: class, dtype: int64
```

In [446…

```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pre)
```

Out[446…

```
0.6344086021505376
```

Logistic Regression

In [447…

```python
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[447…

```
LogisticRegression()
```

In [448…

```python
predictions = logisticRegr.predict(X_test)
```

In [449…

```python
accuracy_score(y_test,predictions)
```

Out[449…

```
0.6666666666666666
```

ANN

In [450…

```python
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(128,), random_state=1)
clf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:549: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

Out[450…

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(128,), random_state=1,
              solver='lbfgs')
```

In [451…

```python
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15,), random_state=1,
              solver='lbfgs')
```

Out[451…

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15,), random_state=1,
              solver='lbfgs')
```

In [452…

```python
PreAnn=clf.predict(X_test)
```

In [453…

```python
PreAnn
```

Out[453…

```
array([4, 1, 2, 0, 4, 5, 5, 3, 5, 5, 5, 5, 4, 3, 4, 0, 1, 4, 0, 3, 2, 1,
       2, 3, 5, 3, 3, 5, 4, 2, 3, 3, 4, 4, 5, 3, 5, 2, 4, 0, 1, 5, 3, 0,
       1, 3, 3, 2, 5, 3, 4, 4, 1, 3, 5, 5, 3, 5, 4, 0, 0, 5, 4, 2, 1, 0,
       1, 3, 2, 4, 3, 0, 5, 1, 1, 3, 5, 1, 3, 3, 2, 1, 5, 5, 3, 1, 4, 1,
       5, 3, 3, 3, 5])
```

In [454…

```python
accuracy_score(y_test,PreAnn)
```

Out[454…

```
0.7311827956989247
```

In [194…

```python

```