codelab_1_BJeanson

September 13, 2023

1 CodeLab 1 - Basics of Python (Numpy and Pandas) & Data Processing

Radar sensors have previously been successfully used to classify actions such as walking, carrying an item, and discriminating between people and animals' gaits, drones, and bird targets. All of this analysis used the phenomenon called Micro-Doppler, which is the additional movements a target has on top of its bulk velocity. For example, a person may walk forward at 3 m/s, but their arms and legs oscillate back and forth as they move at this speed. This movement created a signature that was coined as Micro-Doppler [1(references in the pdf document)].

DopNet is a large radar database that contains Frequency Modulated Continuous Wave (FMCW) and Continuous Wave (CW) radar measurements of different gestures.

To evaluate how effectively a radar recognizes gestures, this challenge provides data that can be used to apply classification methods. A database of gestures has been created and uploaded here using the Ancortek Radar system. This database includes signals from 4 distinct types (Wave / Pinch / Click / Swipe). The data has been pre-processed so that the signatures have been cut into individual actions from a long data stream, filtered to enhance the desired components, and processed to produce the Doppler vs. time-domain data. The data is then stored in this format for it to be read in, features to be extracted and the classification process to be performed.

In this CodeLab, you will use basic Python libraries Numpy, Matplotlib, and Pandas to get familiar with Python basics and data processing. Provide your answers in the Jupyter Notebook either as a comment or insert a markdown cell. Please print your Jupyter Notebook in PDF (or HTML) format and upload it to Brightspace. You will get points for each part separately, and you will get the grades based on your overall performance. Table 1 in the pdf document illustrates the grading scheme of CodeLab 1.

[ ]:
```python
## Following libararies are used in this codelab
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2 Part A : Numpy Basics

Basics (lists, arrays,dictionary) **A1)** Processed Dopnet data of 4 people is given as: - Person A: 37.2210,-36.3014,-35.4000,-40.9004 - Person B: -34.8698, -34.3005, -33.7987,-39.0423 - Person C: -40.8702, -42.6151, -35.8469, -35.2198 - Person D: -34.5080, -40.1056 ,-42.3942 ,-45.5277

Use the data provided above for all of Part A, unless mentioned otherwise

Create a list named "P_A" which contains the data of Person A. Print the created list.

[ ]:
```
#A1
P_A = 37.2210,-36.3014,-35.4000,-40.9004
print("Person A: " + str(P_A))
```

```
Person A: (37.221, -36.3014, -35.4, -40.9004)
```

A2) Using Person B's data create a list named "P_B_abs" that contains absolute values of the "Person B" data and print the list. (Hint1: Use the abs() function from the Numpy library to return the magnitude of the entries in the list.)

[ ]:
```
#A2
P_B = -34.8698, -34.3005, -33.7987,-39.0423
P_B_abs = np.abs(P_B)
print("Absolute person B: " + str(P_B_abs))
```

```
Absolute person B: [34.8698 34.3005 33.7987 39.0423]
```

A3) Make 2 separate lists "P_C_up" and "P_D_down" where lists contain rounded up "Person C" and rounded down "Person D" data. Print both lists. (Hint: You can use ceil() and floor() functions from Numpy)

[ ]:
```
#A3
P_C = -40.8702, -42.6151, -35.8469, -35.2198
P_C_up = np.ceil(P_C)
print("Round up C:" + str(P_C_up))

P_D = -34.5080, -40.1056 ,-42.3942 ,-45.5277
P_D_down = np.floor(P_D)
print("Round down D:" + str(P_D_down))
```

```
Round up C:[-40. -42. -35. -35.]
Round down D:[-35. -41. -43. -46.]
```

A4) For this part, create an array called "CD" that contains the newly created lists of "Person C" and "Person D" in a way that the first entry represents the list of Person C and the second entry Person D. To do so, you need to use the array() function from numpy. Print the new array, its shape and summation of the two elements (as a list) in array "CD". (CD[0]+CD[1]).

[ ]:
```
#A4
CD = np.array([P_C, P_D])
print("Array CD:" + str(CD))
print("Shape of CD:" + str(np.shape(CD)))
print("Sum of CD:" + str(np.sum(CD)))
```

```
Array CD:[[-40.8702 -42.6151 -35.8469 -35.2198]
 [-34.508  -40.1056 -42.3942 -45.5277]]
Shape of CD:(2, 4)
Sum of CD:-317.0875
```

A5) Define a dictionary "DopNet" with people as **keys** and their corresponding data as **values** (enter the values in the form of an array). Print the data of Person D from the "DopNet" dictionary.

[ ]:
```
#A5
a = np.array(P_A)
b = np.array(P_B)
c = np.array(P_C)
d = np.array(P_D)

H1_Dict = {'P_A':a, 'P_B':b, 'P_C':c, 'P_D':d}

print('Person D: ', H1_Dict['P_D'])
```

```
Person D:  [-34.508  -40.1056 -42.3942 -45.5277]
```

A6) Create a vector (1-d array) of values starting from 1 to 24 (including 24) with steps of 0.5. Use arange() function from Numpy. Print out the multiplication of all elements in the vector with prod() function.

[ ]:
```
#A6
V = np.arange(1, 24.1, 0.5)
print(np.prod(V))
```

```
8.820617546615546e+46
```

A7) Find and print the dot(.dot()) and cross(.cross()) products of given arrays "a" and "b". What is the difference between dot and cross product?

a=[3,5,6] , b=[1,2,8]

[ ]:
```
#A7
a=[3,5,6]
b=[1,2,8]
print(np.dot(a,b))
print(np.cross(a,b))
```

```
61
[ 28 -18   1]
```

The **dot** product result is a scalar which is the $\sum(a_i, b_i)$ of 2 vectors of the same dimension. The **cross** product is only defined for 2 3-dimension vectors and result in a 3d vector. _____

A8) W1 list is given as "W1=[1,2,3]", assign a new list "W2" which is defined as "W2=W1". Replace the first element of W2 with 0. Print both W1 and W2. Write your observations.

Create "W3" with copy() function from "W1" and change its last element with 0. Print W1, W2 and W3 together.

[ ]:
```
#A8
W1 = [1, 2, 3]
W2 = W1
W2[0] = 0
```

```
print("W1", W1, "W2", W2)
W3 = W1.copy()
W3[2] = 0
print("W1", W1, "W2", W2, "W3", W3)
```

```
W1 [0, 2, 3] W2 [0, 2, 3]
W1 [0, 2, 3] W2 [0, 2, 3] W3 [0, 2, 0]
```

3 Part B- Statistics

B1) Import the dataset "PersonAclick.npy" into array "A" using load() function of numpy. Using the dataset's 6th (5th index) row, create an array called "Click".

This array will be used to perform basic statistical operations.

- Find and print the largest element in the "Click"
- Find and print the smallest element in the "Click"
- Find and print the mean of the "Click"
- Find and print the median of the "Click"
- Find and print the standard deviation of the "Click"

Use only numpy functions to perform the tasks above.

[ ]:
```
#B1
A = np.load("PersonAclick.npy")
Click = A[5]
print("Max: ", np.max(Click))
print("Min: ", np.min(Click))
print("Mean: ", np.mean(Click))
print("Median: ", np.median(Click))
print("Std: ", np.std(Click))
```

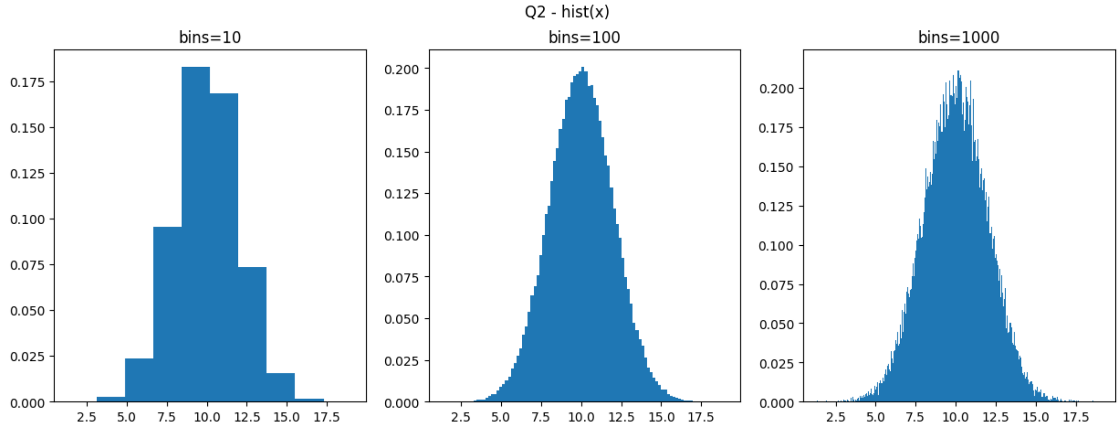
```
Max:  -26.377522754047824
Min:  -46.355691671939425
Mean:  -34.06158363575621
Median:  -33.65093668159305
Std:  4.322823033393506
```

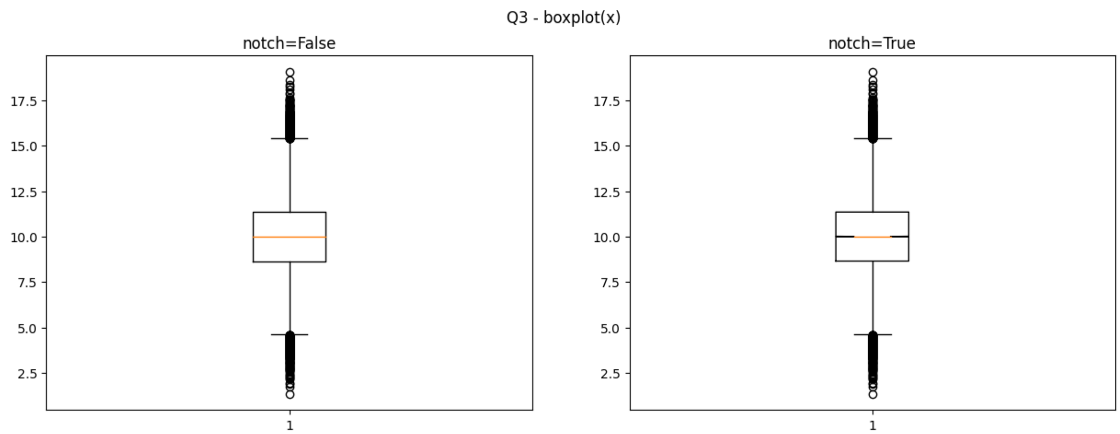
B2) Histogram and Boxplot

- 2) First, generate a vector X containing 100,000 random variables (random.normal() function from Numpy) with a normal distribution of mean=10, and std=2. Then, plot the histogram of the vector X by using the **hist()** function. Set the setting to "bins=100 , density=True" to get the distribution of the data.
- 3) Plot the **boxplot** of X to show the quartiles of the data.
- 4) Plot the histogram of "Click" in part B1. Set density=True to get a density function. Try different **bins**. Define a title and labels for each axis (leave the axes labels as density (y) and units (x)). Write your observations. How does number of bins effect the distribution?

- 5) Plot the boxplot of "Click".Define a title and labels for each axis (leave the axes labels as units (y) and null (x)). Compare histograms and boxplots of X and "Click".

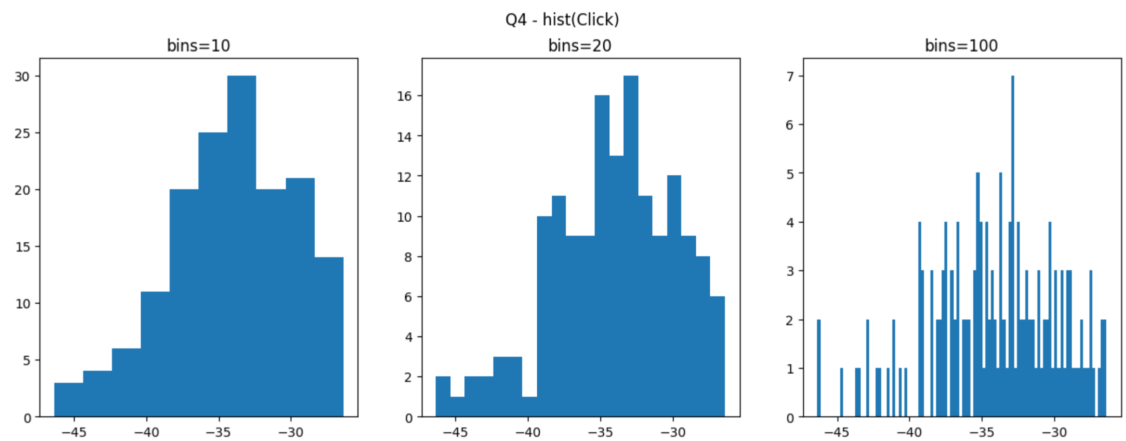
[ ]:
```
#B2

# #Q2

x = np.random.normal(10, 2, 100000)

plt.figure(figsize=(15,5))
plt.suptitle("Q2 - hist(x)")
plt.subplot(1,3,1)
plt.title("bins=10")
plt.hist(x, bins=10, density=True)                       #Try different bins for␣
↪the hist() function.
plt.subplot(1,3,2)
plt.title("bins=100")
plt.hist(x, bins=100, density=True)                       #Try different bins␣
↪for the hist() function.
plt.subplot(1,3,3)
plt.title("bins=1000")
plt.hist(x, bins=1000, density=True)                       #Try different bins␣
↪for the hist() function.
plt.show()

# #Q3

plt.figure(figsize=(15,5))
plt.suptitle("Q3 - boxplot(x)")
plt.subplot(1,2,1)
plt.boxplot(x)
plt.title("notch=False")
plt.subplot(1,2,2)
plt.title("notch=True")
plt.boxplot(x, notch=True)
plt.show()
# plt....
# plt...
# plt...
# plt...
```

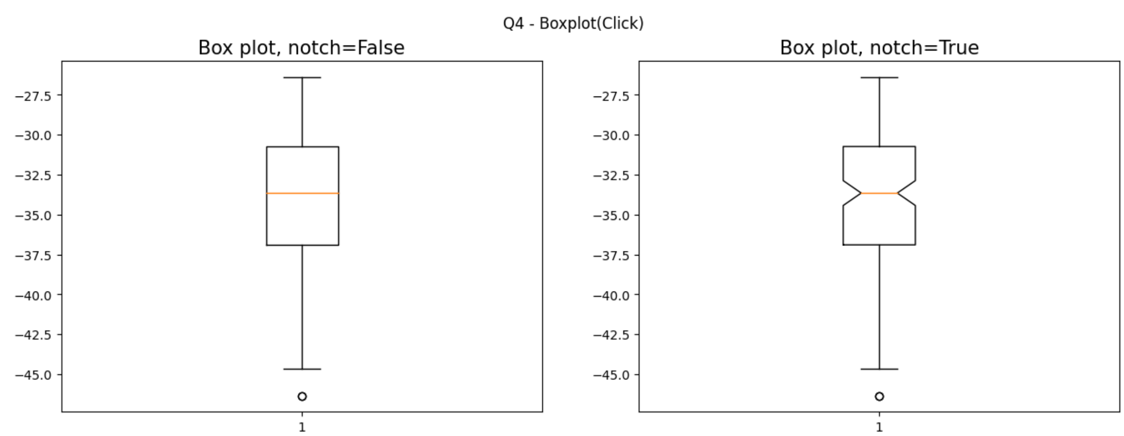




[ ]:
```
#B2
#Q 4
plt.figure(figsize=(15,5))
plt.suptitle("Q4 - hist(Click)")
plt.subplot(1,3,1)
plt.title("bins=10")
plt.hist(Click, 10)                    #Try different bins for the hist()␣
 ↪function.
plt.subplot(1,3,2)
plt.title("bins=20")
plt.hist(Click, 20)                    #Try different bins for the hist()␣
 ↪function.
plt.subplot(1,3,3)
plt.title("bins=100")
plt.hist(Click, 100)                    #Try different bins for the hist()␣
 ↪function.
```

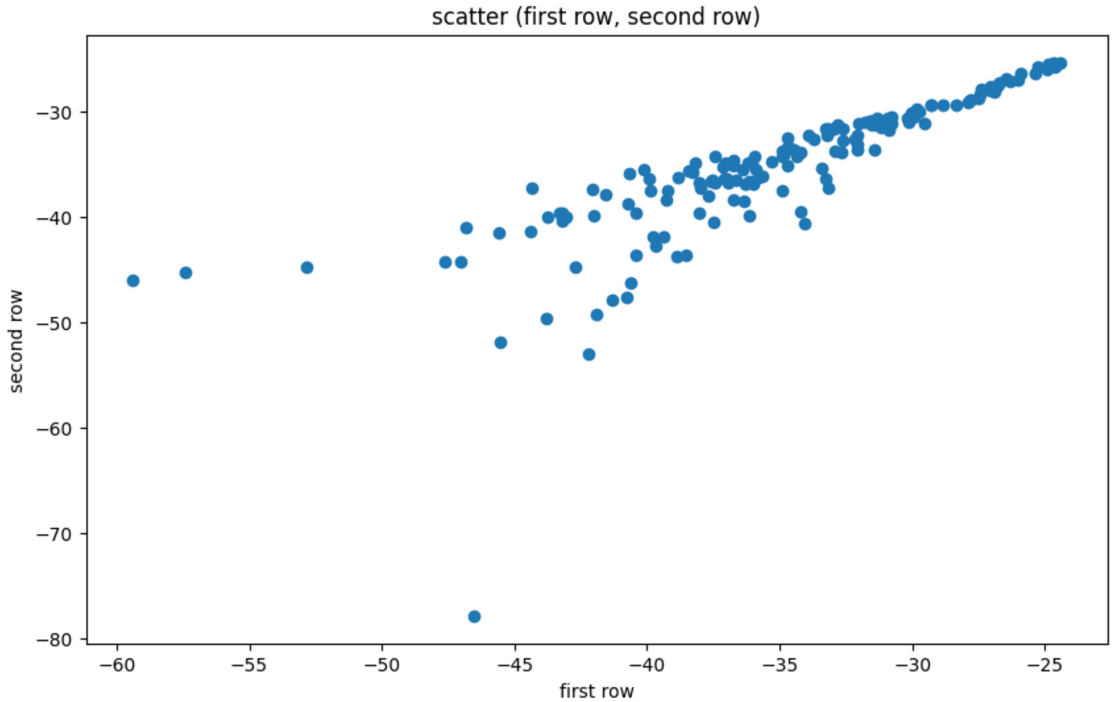
```python
plt.show()

# #Q4
plt.figure(figsize=(15,5))
plt.suptitle("Q4 - Boxplot(Click)")
plt.subplot(1,2,1)
plt.title("Box plot, notch=False", fontsize=15)
plt.boxplot(Click)
plt.subplot(1,2,2)
plt.title("Box plot, notch=True", fontsize=15)
plt.boxplot(Click, notch=True)
plt.show()
```





B3) Scatter Plot - 6) Using the "PersonAClick.npy" (the one that was loaded), present a scatter plot where the x-axis is the first row and y-axis is the second row . Define proper labels and title.

[ ]:
```
#scatterplot
plt.figure(figsize=(10,6))
plt.title("scatter (first row, second row)")
plt.scatter(A[0],A[1])
plt.xlabel("first row")
plt.ylabel("second row")
plt.show()
```



4 Part C : For, While, If/Else

C1) Using the unscaled version of "personAclick.npy", ('.npy file'), assign the first 10 values from the file to an array according to the following pattern in the figure.

[ ]:
```
# from IPython.display import Image
# Image("pattern.png")
```

Explanation: apart from the first element, every first row(r+1) and second column (c+2).

[ ]:
```
A = np.load("PersonAclick.npy")
Reduction_A = [A[i, 2 * i] for i in range(10)]
print("Reduction_A:", Reduction_A)
```

```
Reduction_A: [-30.02243593345215, -29.346431886462106, -30.33548646493486,
-32.62667369655402, -34.12127671198736, -33.15698735808775, -35.43463365723958,
-48.733732762956706, -40.99852625211449, -36.389457522475645]
```

C2) Use If/Else:

- Using the array obtained by using the first row of the personAclick.npy file, print the number of positive,negative and 0 elements.

[ ]:
```
pctr = 0
nctr = 0
zctr = 0

for a in A[0]:
    if a > 0:
        pctr += 1
    elif a == 0:
        zctr += 1
    else:
        nctr += 1

print("positive:",pctr)
print("negative:",nctr)
print("zero:",zctr)
```

```
positive: 0
negative: 154
zero: 0
```

5 Part D: Data manipulation (Introduction to Pandas)

D1) {you could also use numpy(arrays) for this task}

- Import the given four '.npy' files. Create four separate data frames corresponding to the imported files. Check the dimensions of these data frames, using the 'dataframe.size()' function. Print out dimensions.

- Reduce data frame size by creating new data frames by only taking every 8th column and then by taking every 12th row of the data frames created above. Hint: You can do this process in 2 separate steps, first select every 8th column from the original set. Then select every 12th row from your reduced version.

- Plot a spectrogram of these scaled-down data frames (4 separate plots), then compare it with the spectrogram created using the original data frame.(Use the given spectrogram function "spec-plot"). Are the spectrograms, produced by the scaled-down data frame, able to produce the same observations as the original spectogram? Why do we need this kind of scale-down?

[ ]:
```
#spectogram function
def specplot(df_reduce, df, title):
  plt.figure(figsize=(15,5))
  plt.suptitle(title)
  plt.subplot(1,2,1)
  plt.title('reduced')
```

```python
    plt.imshow(df_reduce,vmin=-50, vmax=0,cmap='jet', aspect='auto',extent=[0,df.
 ↪shape[1],-501,500]) # let's give this line and tell the students to complete␣
 ↪the function
    plt.ylabel("Doppler",fontsize=17)
    plt.xlabel("Time",fontsize=17)
    plt.subplot(1,2,2)
    plt.title('original')
    plt.imshow(df,vmin=-50, vmax=0,cmap='jet', aspect='auto',extent=[0,df.
 ↪shape[1],-501,500]) # let's give this line and tell the students to complete␣
 ↪the function
    plt.ylabel("Doppler",fontsize=17)
    plt.xlabel("Time",fontsize=17)
```

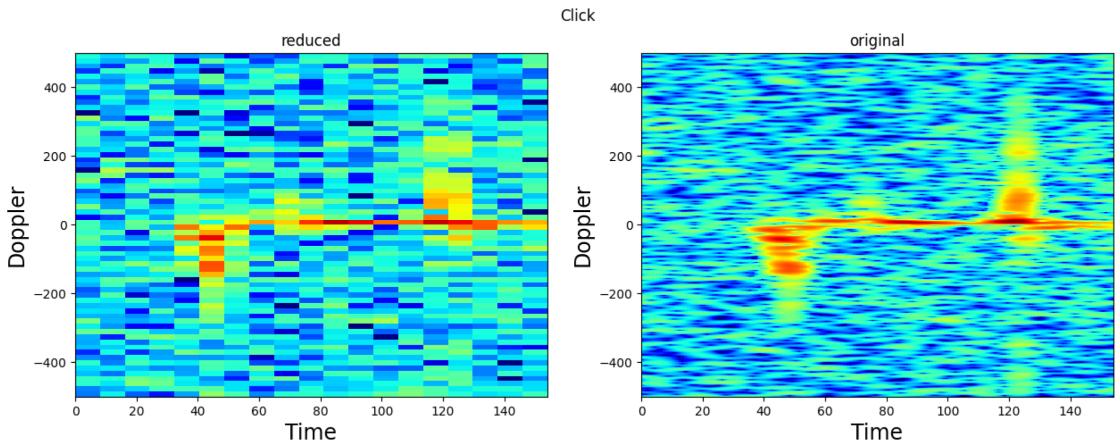
[ ]:
```python
a_click = pd.DataFrame(np.load('personAclick.npy'))
a_pinch = pd.DataFrame(np.load('personApinch.npy'))
a_wave = pd.DataFrame(np.load('personAwave.npy'))
a_swipe = pd.DataFrame(np.load('personAswipe.npy'))
print(f"size of \n\
    a_click: {a_click.size}\n\
    a_pinch: {a_pinch.size}\n\
    a_wave: {a_wave.size}\n\
    a_swipe: {a_swipe.size}")
```

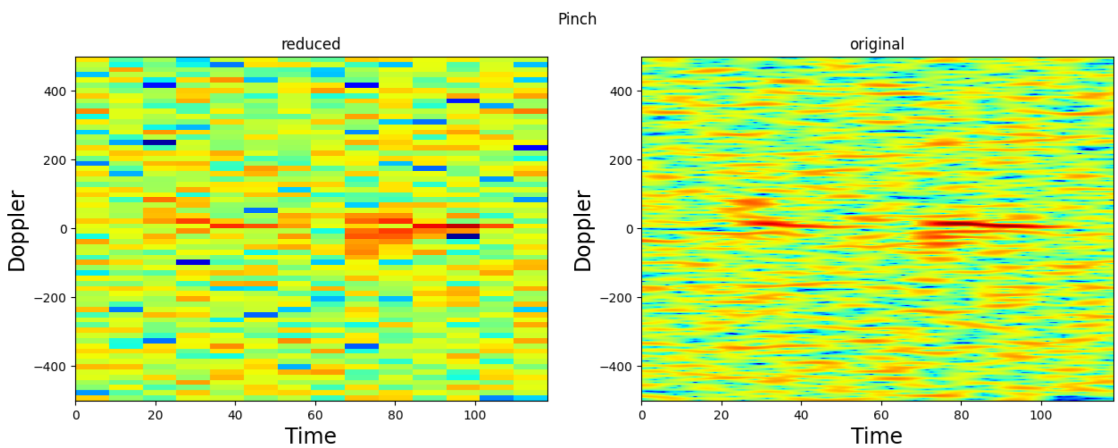
```
size of
    a_click: 123200
    a_pinch: 94400
    a_wave: 108800
    a_swipe: 209600
```

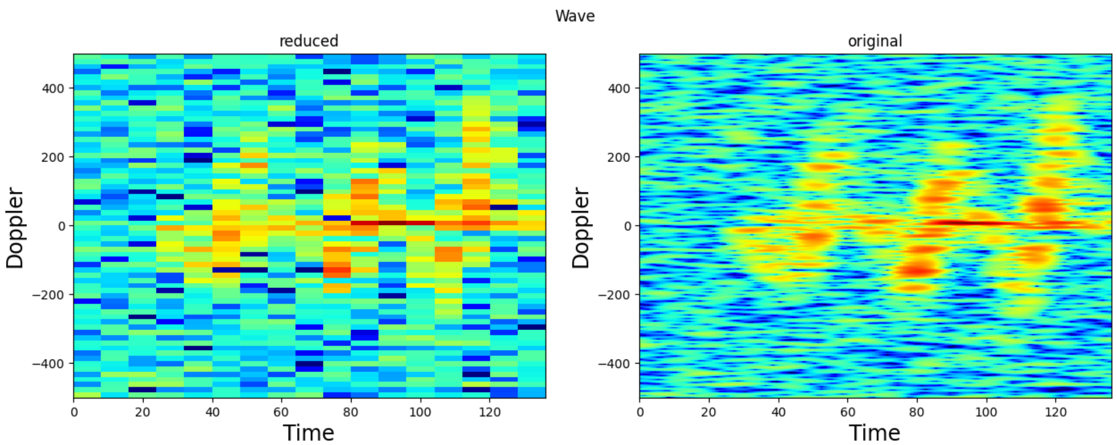
[ ]:
```python
def reduce(df):
    nb_row, nb_col = df.shape
    nb_col / 8
    return df\
        .iloc[:, [(i+1)*8 - 1 for i in range(nb_col//8)]]\
        .iloc[[(i+1)*12 - 1 for i in range(nb_row//12)], :]

reduce_a_click = reduce(a_click)
reduce_a_pinch = reduce(a_pinch)
reduce_a_wave = reduce(a_wave)
reduce_a_swipe = reduce(a_swipe)
print(a_click.shape, a_pinch.shape, a_wave.shape, a_swipe.shape)
print(reduce_a_click.shape, reduce_a_pinch.shape, reduce_a_wave.shape,␣
 ↪reduce_a_swipe.shape)
```

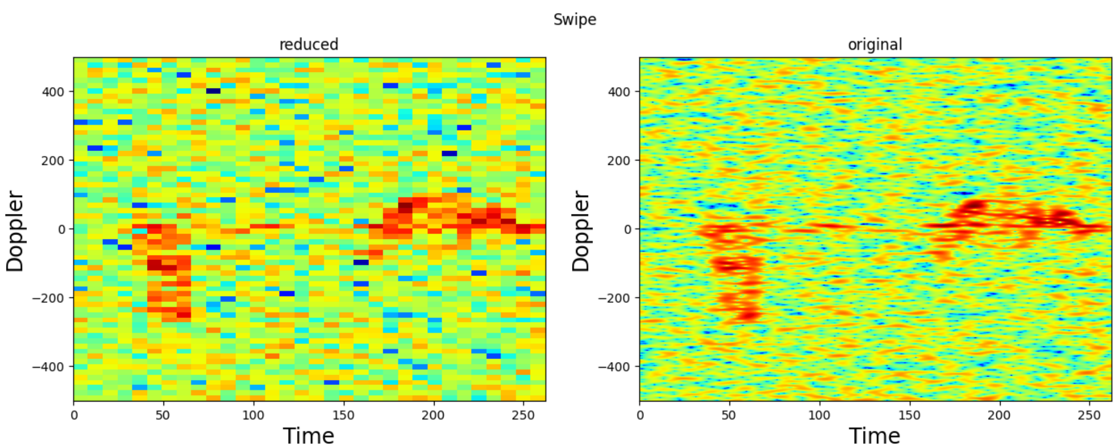
```
(800, 154) (800, 118) (800, 136) (800, 262)
(66, 19) (66, 14) (66, 17) (66, 32)
```

```
[ ]: specplot(reduce_a_click, a_click, 'Click')
     specplot(reduce_a_pinch, a_pinch, 'Pinch')
     specplot(reduce_a_wave, a_wave, 'Wave')
     specplot(reduce_a_swipe, a_swipe, 'Swipe')
```









Are the spectrograms, produced by the scaled-down data frame, able to produce the same observations as the original spectogram?

The scaled-down data frames differ from one another significantly. Based on that example, the remaining information seems to be sufficient to classify a gesture into one of the 4 categories.

Why do we need this kind of scale-down?

Reducing the size of the observations enables to lighten the learning and inference processes.

6 Part E: Train-Test Split

E1) Create a training-test split function called "My_split". The function shuffles the input data and selects random samples to generate training and test data sets which is determined by another user input "ratio" between [0.0-1.0].

Divide the data frame of "personAclick" into training and test sets with a ratio of 0.8 (80% Training, 20% Test) where each row represents a single data entry. What are the dimensions of the train and test data frames? Why do we need this process for machine learning applications?

[ ]:
```
import random
def My_split(df, ratio):
    nb_row = df.shape[0]
    indexes = [i for i in range(nb_row)]
    random.shuffle(indexes)
    split_index = int(nb_row * ratio)
    return df.iloc[indexes[:split_index]], df.iloc[indexes[split_index:]]

train,test = My_split(a_click, 0.8)
print(f"nb of rows in the:\n\
    data frame:{a_click.shape[0]},\n\
    training set:{train.shape[0]}\n\
    test set:{test.shape[0]}\n")
```

```
nb of rows in the:
    data frame:800,
    training set:640
    test set:160
```

[ ]:
```
train
```

[ ]:
```
             0          1          2          3          4          5    \
752 -27.393211 -27.938250 -28.497299 -29.359881 -30.848302 -32.879433
545 -31.848555 -32.523930 -33.944357 -35.026143 -36.221523 -36.171784
125 -31.043092 -29.395861 -28.082364 -27.244654 -26.701760 -26.387718
43  -29.937862 -29.176509 -28.663837 -28.209392 -27.984362 -27.913316
149 -39.204160 -42.345498 -38.252605 -34.586521 -32.139781 -30.425668
..         …          …          …          …          …          …
657 -33.578478 -34.723272 -35.909162 -36.232245 -35.664464 -35.351058
790 -35.142166 -35.356957 -35.812841 -36.234739 -36.768249 -37.587706
573 -28.946116 -30.518577 -31.695739 -32.614279 -32.770689 -32.247868
595 -36.615982 -35.879527 -35.943653 -35.953860 -35.862942 -36.169548
712 -38.187247 -35.659061 -33.998118 -32.772468 -32.367864 -32.389424

             6          7          8          9    …        144        145  \
752 -35.402157 -40.040508 -45.706772 -45.025502  … -42.916049 -43.750508
545 -34.837988 -32.651332 -31.133804 -29.598261  … -28.533855 -28.908692
125 -26.442429 -26.342965 -26.407507 -26.641302  … -26.827092 -27.170569
43  -27.791256 -27.853823 -28.090131 -28.088248  … -44.333101 -39.304204
149 -29.402009 -29.058579 -28.993582 -29.347182  … -49.639448 -46.678984
..         …          …          …          …  …         …          …
657 -34.980427 -35.521656 -35.705971 -36.619256  … -32.980667 -31.480827
790 -38.911453 -39.584292 -39.463701 -38.933969  … -30.101188 -28.663242
573 -31.735709 -32.000687 -32.241433 -33.067081  … -29.028303 -28.779381
595 -35.326626 -34.552940 -33.603323 -32.187141  … -36.912983 -41.039240
```

```
712 -32.841249 -33.756728 -35.539366 -37.777287  … -31.672599 -32.204571

           146        147        148        149        150        151  \
752 -42.994262 -45.130239 -47.422448 -49.064936 -48.007185 -44.903750
545 -29.857044 -31.096746 -32.667585 -35.257255 -39.876840 -49.923356
125 -27.908489 -28.844419 -29.893609 -30.888756 -31.841609 -32.349629
43  -36.400671 -34.986603 -34.472015 -34.436717 -34.310522 -34.660278
149 -40.689362 -37.137538 -35.233652 -34.682096 -34.252266 -34.925017
..         …          …          …          …          …          …
657 -30.686589 -31.036588 -32.273698 -35.015975 -39.321801 -50.043827
790 -27.607411 -26.867966 -26.672102 -26.929007 -27.060528 -27.895930
573 -28.779772 -29.510252 -30.804014 -32.314357 -35.104126 -38.967926
595 -53.460991 -51.806866 -42.289352 -39.332979 -38.510110 -39.174579
712 -32.841628 -34.433333 -36.533521 -40.326635 -46.875076 -64.043468

           152        153
752 -40.042869 -38.286865
545 -46.025207 -38.329100
125 -32.298282 -32.082583
43  -35.135139 -35.336896
149 -35.349834 -35.898767
..         …          …
657 -41.643190 -36.213741
790 -29.222190 -31.276315
573 -45.503694 -63.045561
595 -41.193177 -46.807234
712 -48.179440 -45.609851

[640 rows x 154 columns]
```

[ ]:
```
test
```

[ ]:
```
           0          1          2          3          4          5  \
87  -29.845616 -29.243597 -28.932760 -29.303434 -30.072016 -31.067956
795 -42.032878 -40.517971 -38.846271 -36.305948 -34.585444 -33.679724
341 -37.403115 -39.058335 -40.969765 -40.689014 -38.540432 -35.472444
396 -34.156562 -33.594094 -33.502769 -34.068895 -33.346647 -32.551215
578 -25.603693 -26.246011 -26.961248 -27.573585 -28.464785 -28.964322
..         …          …          …          …          …          …
697 -37.704786 -38.072547 -38.802757 -39.560015 -39.996796 -41.425307
399 -37.154111 -36.951750 -37.856443 -38.484333 -39.319417 -37.525072
143 -29.249091 -30.060996 -31.207230 -32.856595 -35.659655 -39.511533
745 -50.912899 -48.800093 -49.168965 -48.264978 -44.166126 -41.024518
733 -31.121866 -33.177573 -36.588998 -42.275909 -48.912716 -39.943498

           6          7          8          9  …        144        145  \
87  -31.942759 -32.842988 -33.357587 -33.557302  … -37.389361 -36.556083
```

```
795 -32.732342 -32.688214 -33.455049 -34.657646  … -37.201434 -40.635040
341 -33.486891 -31.911338 -31.373057 -30.908984  … -24.744919 -25.356003
396 -31.212171 -29.940276 -29.045154 -28.571675  … -13.104271 -13.355606
578 -29.591108 -30.166565 -31.134574 -32.140356  … -34.018635 -38.346642
..         …          …          …          …  …          …          …
697 -41.164968 -42.867824 -44.210669 -48.893693  … -32.857663 -32.765503
399 -34.629755 -31.854641 -29.850709 -28.601113  … -16.119267 -17.077614
143 -49.372334 -53.468500 -42.181016 -38.304196  … -39.529809 -40.633956
745 -38.260097 -36.275860 -34.664804 -34.231479  … -31.151211 -31.442227
733 -35.920351 -33.244360 -31.672321 -30.173320  … -38.590914 -38.784674

           146        147        148        149        150        151  \
87  -35.594059 -34.424342 -33.227307 -31.926399 -31.207757 -30.879671
795 -49.161695 -53.552630 -43.617474 -38.653434 -36.922688 -35.168783
341 -25.365336 -25.857225 -26.117928 -25.983078 -25.994464 -25.579738
396 -13.655117 -13.995127 -14.328767 -14.581803 -14.740573 -14.746797
578 -40.025497 -35.232218 -31.887123 -29.878983 -28.617832 -28.049661
..         …          …          …          …          …          …
697 -33.168939 -33.564814 -33.682893 -33.173875 -32.804397 -31.735652
399 -19.409874 -24.026764 -36.505601 -29.339942 -22.537589 -19.636716
143 -40.161042 -38.326757 -38.582019 -37.667482 -38.254647 -38.243614
745 -31.706246 -32.685530 -33.716626 -35.286515 -37.049350 -38.778726
733 -37.394510 -35.741591 -34.300035 -33.105600 -32.114544 -31.249522

           152        153
87  -31.178963 -31.518067
795 -33.934560 -33.406833
341 -25.402025 -25.562321
396 -14.795678 -14.862880
578 -27.642892 -27.931875
..         …          …
697 -30.569062 -29.229666
399 -18.356278 -18.188624
143 -37.173144 -35.163113
745 -40.904079 -40.787129
733 -30.890553 -30.135237

[160 rows x 154 columns]
```

What are the dimensions of the train and test data frames?

The number of rows of the train set is 640 (=800*0.8) and of the test set is 160 (=800*0.8) (the number of column is the one of the original set = 154)

Why do we need this process for machine learning applications?

It is important to have a test set to identify if the trained model is under or over fitted.

7 Bonus: Train-Test split of labels

Add a column (label vector) containing 0s and 1s to data frame of "personAclick" of 800 elements with a random 40-60 split of 0s and 1s. Split the data frame into training and test sets with a ratio of 0.8, and the ratio of 0s to 1s remains the same. This split process is called a stratified train-test split. Show the ratio of 0s to 1s for both the training and test set. You can even explain your solution in words without implementing the code.

[ ]:
```python
nb_zeros = int(.4 * 800)
zeros_ones = [0] * nb_zeros
zeros_ones += ([1] * (800 - nb_zeros))
random.shuffle(zeros_ones)
a_click2 = a_click.copy()
a_click2.insert(0, "zeros_ones", zeros_ones)

train2, test2 = My_split(a_click2, .8)

def ratio_zeros_ones(df):
    nb_zeros = 0
    for a in df.loc[:, "zeros_ones"]:
        if a == 0:
            nb_zeros+=1
    ratio_zeros = nb_zeros / df.shape[0]
    return ratio_zeros, 1 - ratio_zeros

print(f"the ratio of (0s, 1s) are:\n\
    original data frame: {ratio_zeros_ones(a_click2)}\n\
    train2 data frame: {ratio_zeros_ones(train2)}\n\
    test2 data frame: {ratio_zeros_ones(test2)}")
```

```
the ratio of (0s, 1s) are:
    original data frame: (0.4, 0.6)
    train2 data frame: (0.39375, 0.60625)
    test2 data frame: (0.425, 0.575)
```