Final-Project-DGMDS-14

August 5, 2020

1 DGMD S-14 WEARABLE DEVICES AND COMPUTER VISION

1.0.1 Evaluating the use of Micro-Electro-Mechanical Systems sensor in detecting and monitoring motor developmental delays in children with Autism Spectrum Disorder - An Exploratory Study

1.1 Harvard University, Extension

Summer 2020
Name: Mildred Fakoya

1.2 Introduction

The measurement and analysis of motor skills involves measurement of a set of movement profiles. In this project, 2 activities categorized as Gross-motor movements and 2 activities as Fine-motor movement were conducted. I used hand movement to measure fine motor.
A sensortile was placed on the wrist and foot of the subject for measurements of Fine and Gross motor movements respectively. The sensortile used are lightweight and does not affect the natural movement of the hands and legs.

Objective: Motor disorders are known in autism spectrum disorder (ASD). I collected motion profile of an autism subject and a control subject to evaluate the use of motion sensor for early detection of motor disorder/delays and monitoring of progress during therapy.

Methods: A sensortile was placed on the hand and wrist of the subjects. The sensortile holds an accelerometer that was used to collect the linear acceleration and a gyroscope used to collect the angular rate of the subject over a period of 60 seconds at a time.

1.2.1 Background

The frequent association of ASD with other neurological and extra-neurological signs suggests that autism could be considered as a multiorgan systemic disorder with a primary central nervous system involvement.

Evidence of neuromuscular disorder (dystrophinopathies and congenital muscular dystrophy due to mutations of POMGnT1 gene) has been documented in a subset of patients affected by syndromic ASD, however, only few case reports and small samples studies with specific features have been reported in literature. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3235842/

1.3 Tools Used / Definitions

1.3.1 Acceleration:

It is the rate of change of the velocity of an object

1.3.2 Angular rate:

It is the change in angular velocity about an axis. It is used to check the stability of the subject The unit of measurement is deg/sec

1.3.3 Accelerometer:

Measures the linear acceleration along X, Y, Z axes. It measures the acceleration of a body in its instantenous rest frame. The unit of measurement is m/s^2 or g (G-forces). **The STEVAL-STLKT01V1 development kit was used for prototyping and evaluation. The sensortile holds a MEMS accelerometer that was used to collect the linear acceleration of the subject during an activity.**

1.3.4 Gyroscope:

MEMS gyroscope was used to measure angular velocity. The units of angular velocity are measured in degrees per second (°/s) or revolutions per second (RPS). The angular rate is used to evaluate the balance of the subject since it senses rotation.

Fig.1 - STEVAL-STLKT01V1 Development Kit

To upload firmware to the sensortile an external SWD debugger - ST-LINK/V2-1(STM32 Nucleo-64) development board was used.

Fig.2 - ST-LINK/V2-1 Development Board

1.4 Softwares Used

1.4.1 Algobuilder:

Used to design prototypes/ firmware for the project

1.4.2 Unicleo GUI

User interface for data collection and export

1.5 Data Collection and Exploratory Analysis

1.5.1 Data Collection process

Four sets of activities were used to collect data. The activities are:
1. Activity 1 - Hop on one leg 2. Activity 2 - Hop on both legs 3. Activity 3 - Color tiny dots 4. Activity 4 - Stack tiny blocks

Two types of measurements were taken. The measurements are:
1. Linear Acceleration in 3D space : Used to determine the magnitude and direction of the acceleration of the subject during the activity and to sense the orientation or change in orientation.
2. Angular Rate in 3D space: Used to determine the stability of the subject.

Two type of subjects were used in this analysis:
1. A control subject without developmental delays 2. A subject with Autism Spectrum Disorder and with motor difficulties.

Fig.3 - SensorTile on the Toe of a subject

Fig.4- SensorTile on the wrist of the subject

Further project description - https://youtu.be/O–TZ43lbMY

1.6 Let's Take a look at collected data

1.6.1 first, let's read in the data files

[1]:
```
import glob
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#For Activity 1
path_1_autism =  "Data logs/Activity_1_autism/*.csv"
path_1_control =  "Data logs/Activity_1_control/*.csv"

#For Activity 2
path_2_autism =  "Data logs/Activity_2_autism/*.csv"
path_2_control =  "Data logs/Activity_2_control/*.csv"


#For Activity 3
path_3_autism =  "Data logs/Activity_3_autism/*.csv"
path_3_control =  "Data logs/Activity_3_control/*.csv"

#For Activity 4
path_4_autism =  "Data logs/Activity_4_autism/*.csv"
path_4_control =  "Data logs/Activity_4_control/*.csv"
```

1.6.2 next create numpy arrays to store each of the data files for easier reference

[2]:
```
#For Activity 1 - Hopping on one leg
path_activity_1_files_autism = np.array([])
path_activity_1_files_control = np.array([])

#For Activity 2 - Hopping on both legs
path_activity_2_files_autism = np.array([])
path_activity_2_files_control = np.array([])

#For Activity 3 - Color tiny dots
path_activity_3_files_autism = np.array([])
path_activity_3_files_control = np.array([])
```

```
#For Activity 4 - Stack small blocks
path_activity_4_files_autism = np.array([])
path_activity_4_files_control = np.array([])
```

1.6.3 Next, let's read in the files into the created arrays

[3]:
```
# For activity 1 - Hopping on one leg
for fname_1_autism in glob.glob(path_1_autism):
    path_activity_1_files_autism = np.append(path_activity_1_files_autism ,␣
 ↪fname_1_autism)
for fname_1_control in glob.glob(path_1_control):
    path_activity_1_files_control= np.append(path_activity_1_files_control ,␣
 ↪fname_1_control)
```

[4]:
```
# For activity 2 - Hopping on both legs
for fname_2_autism in glob.glob(path_2_autism):
    path_activity_2_files_autism = np.append(path_activity_2_files_autism ,␣
 ↪fname_2_autism)
for fname_2_control in glob.glob(path_2_control):
    path_activity_2_files_control= np.append(path_activity_2_files_control ,␣
 ↪fname_2_control)
```

[5]:
```
# For activity 3 - Color tiny dots
for fname_3_autism in glob.glob(path_3_autism):
    path_activity_3_files_autism = np.append(path_activity_3_files_autism ,␣
 ↪fname_3_autism)
for fname_3_control in glob.glob(path_3_control):
    path_activity_3_files_control= np.append(path_activity_3_files_control ,␣
 ↪fname_3_control)
```

[6]:
```
# For activity 4 - Stack small blocks
for fname_4_autism in glob.glob(path_4_autism):
    path_activity_4_files_autism = np.append(path_activity_4_files_autism ,␣
 ↪fname_4_autism)
for fname_4_control in glob.glob(path_4_control):
    path_activity_4_files_control= np.append(path_activity_4_files_control ,␣
 ↪fname_4_control)
```

[7]:
```
#for j in path_activity_1_files_autism:
    #data = pd.read_csv(j)
    #fig1,ax1 = plt.subplots()
    #ax1.set_title("Outlier detection")
    #ax1.boxplot(data)
```

1.6.4 Next let's read in a file of each type of activity for visualization to have a better look at the data

[8]:
```
# For Activity 1
df_activity_1_autism = pd.read_csv(path_activity_1_files_autism[1])
df_activity_1_control = pd.read_csv(path_activity_1_files_control[1])
# For Activity 2
df_activity_2_autism = pd.read_csv(path_activity_2_files_autism[3])
df_activity_2_control = pd.read_csv(path_activity_2_files_control[3])
# For Activity 3
df_activity_3_autism = pd.read_csv(path_activity_3_files_autism[2])
df_activity_3_control = pd.read_csv(path_activity_3_files_control[2])
#For Activity 4
df_activity_4_autism = pd.read_csv(path_activity_4_files_autism[14])
df_activity_4_control = pd.read_csv(path_activity_4_files_control[14])
```

[9]:
```
df_activity_3_control.head()
```

[9]:

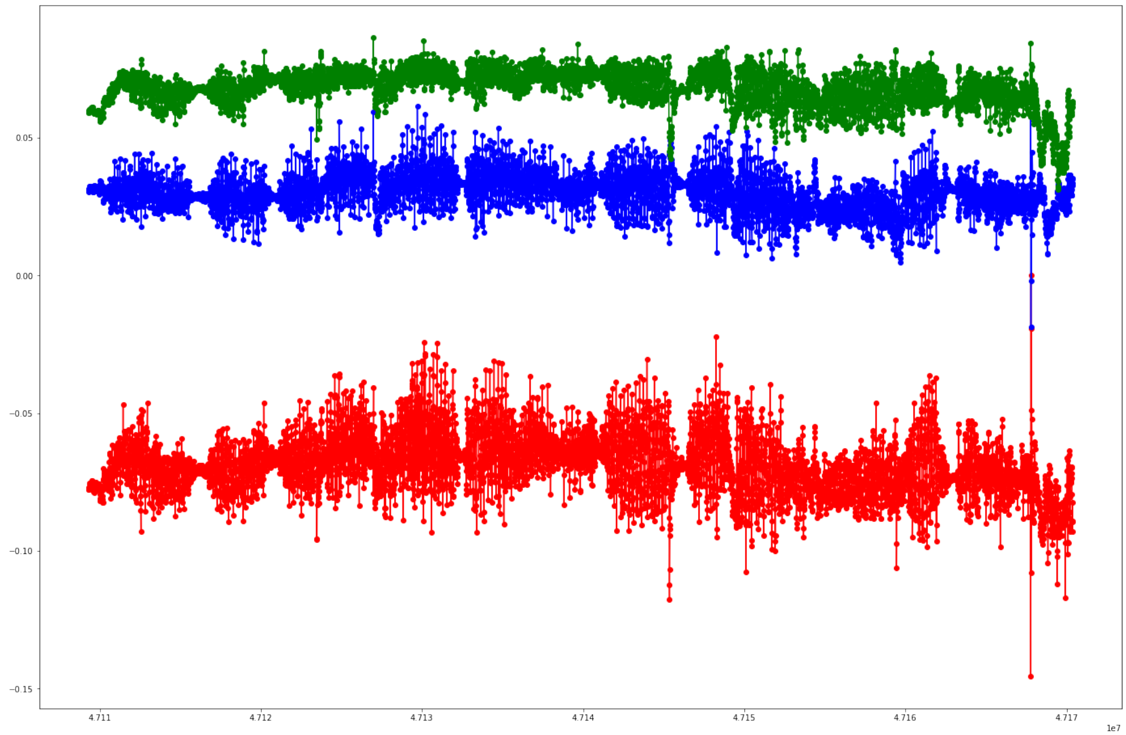
	time[ms]	accX[g]	accY[g]	accZ[g]	Value 1	Value 2	Value 3
0	47109300	-0.761	0.299	0.580	0.787	-4.077	1.662
1	47109300	-0.762	0.303	0.583	0.175	-4.340	2.327
2	47109320	-0.755	0.302	0.581	-0.350	-5.162	2.712
3	47109320	-0.757	0.306	0.585	-0.962	-5.232	3.307
4	47109330	-0.762	0.306	0.587	-1.330	-6.002	3.640

The data holds 7 columns. The time in milliseconds, the acceleration in 3 dimensional space (accX, accY, and accZ) and the angular rate in 3 dimensional space (Value 1, Value 2 and Value 3).

First of all, I want to convert irregular time series to a regular frequency. I will use the resample functionality from pandas Let's take a look at an axis with unevenly spaced datapoints.

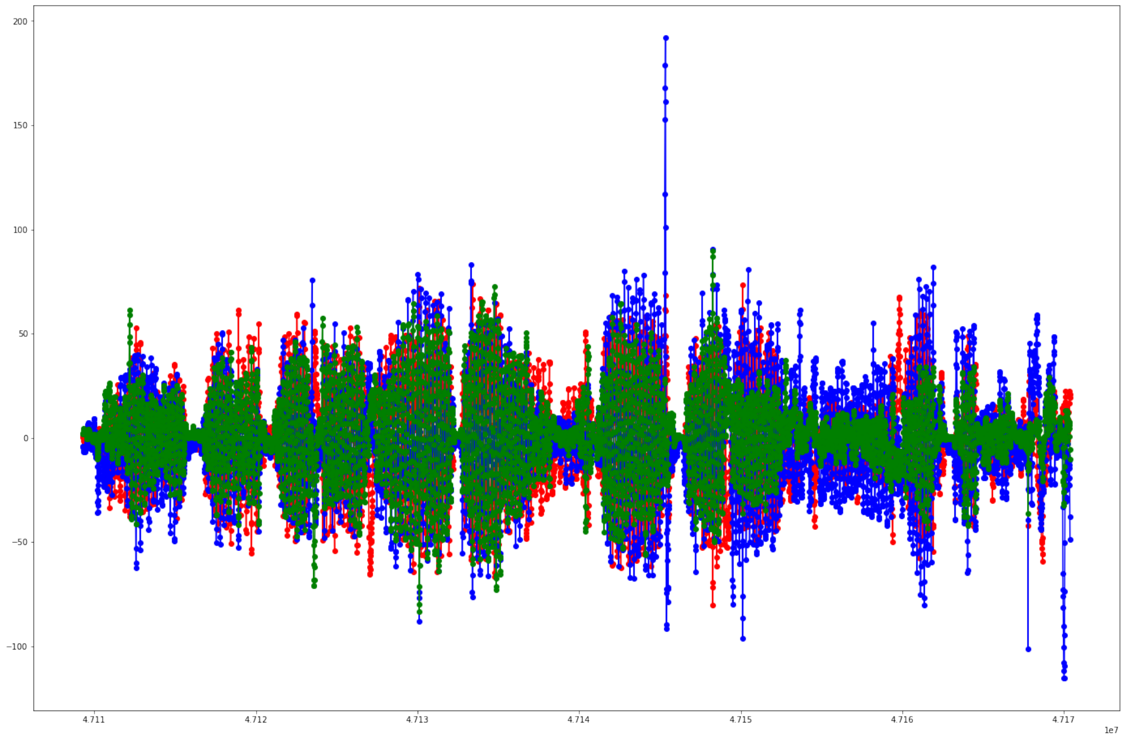
[10]:
```
accX = df_activity_3_control['accX[g]']/9.81
accY = df_activity_3_control['accY[g]']/9.81
accZ = df_activity_3_control['accZ[g]']/9.81
time = df_activity_3_control['time[ms]']
plt.figure(figsize=(24,16))
plt.plot(time, accX, "ro-")
plt.plot(time, accY, "bo-")
plt.plot(time, accZ, "go-")
```

[10]:
```
[<matplotlib.lines.Line2D at 0x2a476dbe5c8>]
```




```
[11]: #For the Angular rate
      Value1= df_activity_3_control['Value 1']
      Value2 = df_activity_3_control['Value 2']
      Value3 = df_activity_3_control['Value 3']
      time = df_activity_3_control['time[ms]']
      plt.figure(figsize=(24,16))
      plt.plot(time, Value1, "ro-")
      plt.plot(time, Value2, "bo-")
      plt.plot(time, Value3, "go-")
```

```
[11]: [<matplotlib.lines.Line2D at 0x2a47756e0c8>]
```



This is a high sample rate data. As we can see 1 second of data contains several datapoints. It might be beneficial to resample and select peaks from a Fast Fourier Transform. I will resample by downsampling

[12]:
```
df_activity_3_control['accX[g]'] = df_activity_3_control['accX[g]']/9.81
df_activity_3_control['accY[g]']= df_activity_3_control['accY[g]']/9.81
df_activity_3_control['accZ[g]'] = df_activity_3_control['accZ[g]']/9.81
df_activity_3_control['Timedelta'] = pd.
 ↪to_timedelta(df_activity_3_control['time[ms]'],'ms')
df_activity_3_control.head()
```

[12]:
```
   time[ms]   accX[g]   accY[g]   accZ[g]  Value 1  Value 2  Value 3  \
0  47109300 -0.077574  0.030479  0.059123    0.787   -4.077    1.662
1  47109300 -0.077676  0.030887  0.059429    0.175   -4.340    2.327
2  47109320 -0.076962  0.030785  0.059225   -0.350   -5.162    2.712
3  47109320 -0.077166  0.031193  0.059633   -0.962   -5.232    3.307
4  47109330 -0.077676  0.031193  0.059837   -1.330   -6.002    3.640

        Timedelta
0 13:05:09.300000
1 13:05:09.300000
2 13:05:09.320000
3 13:05:09.320000
4 13:05:09.330000
```

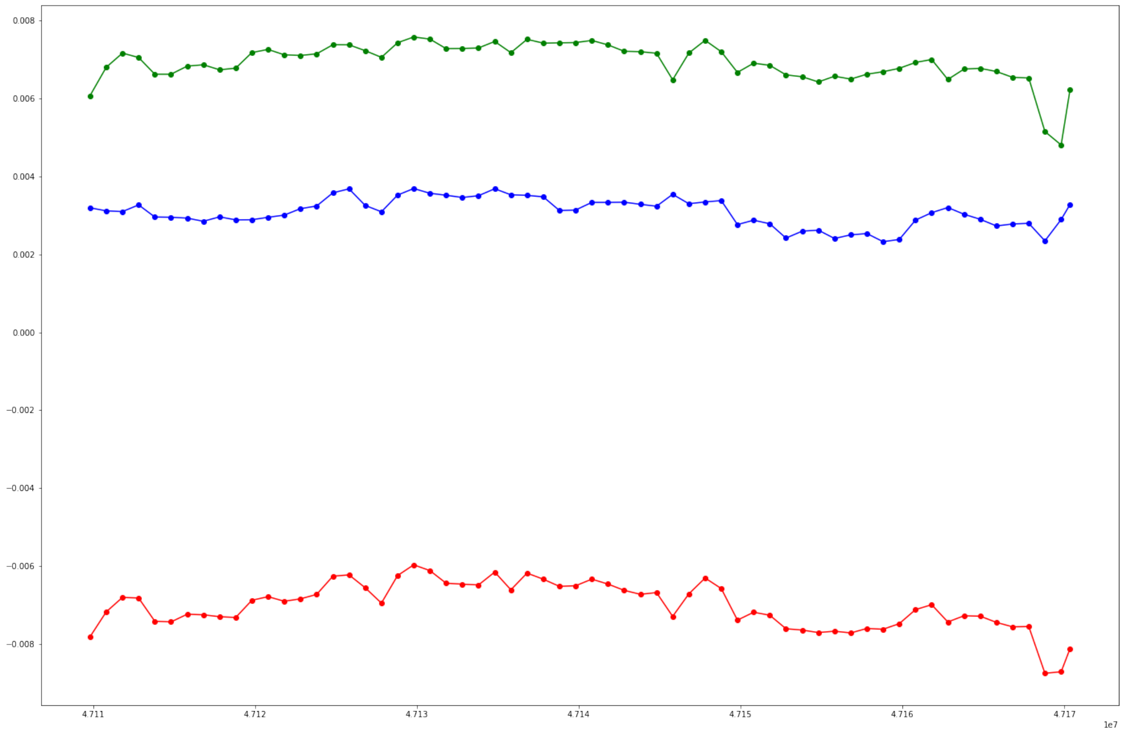
[13]:
```
df_activity_3_control = df_activity_3_control.
 ↪set_index(df_activity_3_control['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
 ↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
df_activity_3_control
```

[13]:
```
                      time[ms]   accX[g]   accY[g]   accZ[g]    Value 1  \
Timedelta
13:05:09.300000  4.710979e+07 -0.076713  0.031355  0.059594  -1.277864
13:05:10.300000  4.711079e+07 -0.070431  0.030573  0.066745  -5.093602
13:05:11.300000  4.711180e+07 -0.066791  0.030397  0.070316  -3.409398
13:05:12.300000  4.711279e+07 -0.066951  0.032077  0.069207  -1.129039
13:05:13.300000  4.711379e+07 -0.072789  0.029034  0.065007  -1.773748
…                          …       …       …       …        …
13:06:06.300000  4.716680e+07 -0.074208  0.027273  0.064207  -1.794670
13:06:07.300000  4.716779e+07 -0.074127  0.027443  0.064043   3.171029
13:06:08.300000  4.716879e+07 -0.085899  0.023018  0.050564 -15.561243
13:06:09.300000  4.716979e+07 -0.085533  0.028395  0.047174   5.881078
13:06:10.300000  4.717034e+07 -0.079737  0.032167  0.061026  18.018889

                   Value 2   Value 3
Timedelta
13:05:09.300000  -4.853864  0.366078
13:05:10.300000  -7.353573  5.897388
13:05:11.300000   0.220097  4.677748
13:05:12.300000   0.509392 -0.060441
13:05:13.300000  -1.469660 -0.161097
…                      …       …
13:06:06.300000  -2.562068  2.480447
13:06:07.300000   5.955147  0.181961
13:06:08.300000   4.639485  5.181583
13:06:09.300000 -13.843583  1.308660
13:06:10.300000 -23.498444  2.099778

[62 rows x 7 columns]
```

[14]:
```
X = df_activity_3_control['accX[g]']/9.81
Y = df_activity_3_control['accY[g]']/9.81
Z = df_activity_3_control['accZ[g]']/9.81
time = df_activity_3_control['time[ms]']
plt.figure(figsize=(24,16))
plt.figure(figsize=(24,16))
plt.plot(time, X, "ro-")
plt.plot(time, Y, "bo-")
plt.plot(time, Z, "go-")
```

[14]:
```
[<matplotlib.lines.Line2D at 0x2a477592e08>]
```

<Figure size 1728x1152 with 0 Axes>



[15]:
```
X = df_activity_3_control['Value 1']
Y = df_activity_3_control['Value 2']
Z = df_activity_3_control['Value 3']
time = df_activity_3_control['time[ms]']
plt.figure(figsize=(24,16))
plt.figure(figsize=(24,16))
plt.plot(time, X, "ro-")
plt.plot(time, Y, "bo-")
plt.plot(time, Z, "go-")
```

[15]: [<matplotlib.lines.Line2D at 0x2a477655b88>]

<Figure size 1728x1152 with 0 Axes>

1.6.5 Let's try applying filter to the angular rate and visualize to see what it produces.

[16]:
```
# Filter requirements.
from scipy.signal import butter,filtfilt

# Sample period = 60secs
T = 60
# sample frequency 1 sample per sec
fs = 1
# desired cutoff frequency of the filter, Hz , slightly higher than actual 0.
↪017 Hz,
#Signal Freq = 1 signal / 60 sec = 0.017 Hz
cutoff = 0.20
nyq = 0.5 * fs
#polynomial order of the frequency
order = 2
#Total number of samples
n = int(T * fs)

def butter_lowpass_filter(data, cutoff, fs, order):
    normal_cutoff = cutoff / nyq
    # Get the filter coefficients
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
```

```
    y = filtfilt(b, a, data)
    return y
```

[17]:

```
import plotly.graph_objects as go
filteredx = butter_lowpass_filter(df_activity_3_control['Value 1'], cutoff, fs,␣
 ↪order)
filteredy = butter_lowpass_filter(df_activity_3_control['Value 2'], cutoff, fs,␣
 ↪order)
filteredz = butter_lowpass_filter(df_activity_3_control['Value 3'], cutoff, fs,␣
 ↪order)
fig = go.Figure()
fig.add_trace(go.Scatter(
            y =  df_activity_3_control['Value 1'],
            line =  dict(shape =  'spline' ),
            name = 'Angular Rate X with noise'
            ))

fig.add_trace(go.Scatter(
            y = filteredx,
            line =  dict(shape =  'spline' ),
            name = 'filtered signal'
            ))
fig.show()

fig = go.Figure()
fig.add_trace(go.Scatter(
            y =  df_activity_3_control['Value 2'],
            line =  dict(shape =  'spline' ),
            name = 'Angular Rate Y with noise'
            ))

fig.add_trace(go.Scatter(
            y = filteredy,
            line =  dict(shape =  'spline' ),
            name = 'filtered signal'
            ))
fig.show()

fig = go.Figure()
fig.add_trace(go.Scatter(
            y =  df_activity_3_control['Value 3'],
            line =  dict(shape =  'spline' ),
            name = 'Angular Rate Z with noise'
            ))

fig.add_trace(go.Scatter(
            y = filteredz,
```

```
            line =  dict(shape =  'spline' ),
            name = 'filtered signal'
            ))
fig.show()
```

Looking at the plot for resampling one data point, low pass filter, average all data and store it in a dataframe will now apply resampling to all datapoints It will be more convienent to use a loop for this

[18]:
```
df_average = pd.DataFrame()
for i in path_activity_1_files_autism:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Value 1'] = df['Value 1']/9.81
    df['Value 2']= df['Value 2']/9.81
    df['Value 3'] = df['Value 3']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d = {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average= pd.concat([df_average,e], axis=1)
df_activity_1_autism=df_average.dropna(axis=1, how="any")
# I will now plot the Activity 1
df_activity_1_autism_transpose = df_activity_1_autism.T
df_activity_1_autism_transpose.reset_index(drop=True, inplace=True)
#df_activity_1_autism_transpose.drop(['measurements'], axis=1)
df_activity_1_autism_transpose['activity'] = "Activity 1"
df_activity_1_autism_transpose['Category'] = "Autism"
df_activity_1_autism_transpose
```

[18]:
```
measurements           time      accx      accy      accz   value 1   value 2  \
0              4.062770e+07 -0.010769 -0.020634  0.084067 -0.031882 -0.321877
1              4.299278e+07  0.001887 -0.026029  0.084207 -0.044473 -0.226855
2              3.708076e+07 -0.007972 -0.011524  0.089105  0.002636 -0.564236
3              3.879517e+07 -0.006569  0.004802  0.090554 -0.405788 -0.292554
```

```
4             3.916542e+07 -0.001322 -0.002085  0.092543 -0.179700 -0.231876
5             6.490116e+07  0.001502 -0.036549  0.081872 -0.386098  0.604522
6             6.030650e+07  0.028276 -0.022667  0.085413 -0.071182 -0.402601
7             6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731

measurements   value 3    activity Category
0             0.187230  Activity 1   Autism
1            -0.065279  Activity 1   Autism
2            -0.104948  Activity 1   Autism
3            -0.012199  Activity 1   Autism
4             0.094250  Activity 1   Autism
5             0.222695  Activity 1   Autism
6             0.213504  Activity 1   Autism
7             0.276178  Activity 1   Autism
```

[19]:

```
df_average_2_autism = pd.DataFrame()
for i in path_activity_2_files_autism:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d_control = {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1',␣
↪'value 2', 'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].
↪mean(), df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(),␣
↪df['Value 2'].mean(), df['Value 3'].mean()]}
    e_control = pd.DataFrame(d).set_index('measurements').
↪rename_axis('columns', axis=1)
    df_average_2_autism = pd.concat([df_average_2_autism ,e], axis=1)
df_activity_2_autism=df_average_2_autism.dropna(axis=1, how="any")
df_activity_2_autism_transpose =df_activity_2_autism.T
df_activity_2_autism_transpose.reset_index(drop=True, inplace=True)
df_activity_2_autism_transpose['activity'] = "Activity 2"
df_activity_2_autism_transpose['Category'] = "Autism"
df_activity_2_autism_transpose
```

[19]:

```
measurements          time      accx      accy      accz   value 1   value 2  \
0             6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
1             6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
```

```
2              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
3              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
4              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
5              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
6              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
7              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
8              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731
9              6.042138e+07  0.026345 -0.023243  0.085093 -0.067943 -0.365731

measurements   value 3    activity Category
0              0.276178  Activity 2   Autism
1              0.276178  Activity 2   Autism
2              0.276178  Activity 2   Autism
3              0.276178  Activity 2   Autism
4              0.276178  Activity 2   Autism
5              0.276178  Activity 2   Autism
6              0.276178  Activity 2   Autism
7              0.276178  Activity 2   Autism
8              0.276178  Activity 2   Autism
9              0.276178  Activity 2   Autism
```

[20]:

```
df_average_3_autism = pd.DataFrame()
for i in path_activity_3_files_autism:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d = {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_3_autism = pd.concat([df_average_3_autism ,e], axis=1)
df_activity_3_autism=df_average_3_autism.dropna(axis=1, how="any")
df_activity_3_autism_transpose =df_activity_3_autism.T
df_activity_3_autism_transpose.reset_index(drop=True, inplace=True)
df_activity_3_autism_transpose['activity'] = "Activity 3"
df_activity_3_autism_transpose['Category'] = "Autism"
```

```
df_activity_3_autism_transpose
```

```
[20]: measurements          time      accx      accy      accz   value 1   value 2  \
0             3.749441e+07 -0.064680  0.008266  0.074322 -0.514734 -2.481144
1             5.227850e+07 -0.098334  0.020179  0.023384 -0.693515 -2.481461
2             5.434130e+07 -0.089278  0.034757  0.032880 -1.097528 -2.549963
3             5.443443e+07 -0.087207  0.039475  0.033553 -1.342991 -1.719701
4             5.463719e+07 -0.088813  0.039425  0.032043 -0.928936 -2.469702
5             5.474373e+07 -0.093695  0.030325  0.026423 -1.267768 -2.495648
6             5.482829e+07 -0.087407  0.039820  0.030216 -0.756676 -1.940886
7             5.491915e+07 -0.091664  0.034122  0.029191 -1.401264 -2.207781

measurements   value 3    activity Category
0             0.282931  Activity 3   Autism
1             0.824650  Activity 3   Autism
2             0.189184  Activity 3   Autism
3             0.953654  Activity 3   Autism
4             0.601889  Activity 3   Autism
5             0.969343  Activity 3   Autism
6             0.519664  Activity 3   Autism
7             0.591613  Activity 3   Autism
```

```
[21]: df_average_4_autism = pd.DataFrame()
for i in path_activity_4_files_autism:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d = {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_4_autism = pd.concat([df_average_4_autism ,e], axis=1)
df_activity_4_autism=df_average_4_autism.dropna(axis=1, how="any")
df_activity_4_autism_transpose =df_activity_4_autism.T
df_activity_4_autism_transpose.reset_index(drop=True, inplace=True)
df_activity_4_autism_transpose['activity'] = "Activity 4"
```

```
df_activity_4_autism_transpose['Category'] = "Autism"
df_activity_4_autism_transpose
```

```
[21]: measurements          time      accx      accy      accz   value 1   value 2  \
      0             4.242729e+07 -0.027478 -0.010368  0.087677 -0.233807 -3.324453
      1             5.057041e+07 -0.069042  0.012629  0.066505 -0.715228 -2.449000
      2             5.068875e+07 -0.072470  0.042816  0.035506 -1.105661 -2.409499
      3             5.080718e+07 -0.067720  0.015585  0.065191 -0.446576 -3.389572
      4             5.110959e+07 -0.070735  0.011466  0.065945 -0.367087 -2.983890
      5             5.120279e+07 -0.057154  0.036216  0.063378 -1.530486 -2.960564
      6             5.140863e+07 -0.061707  0.018761  0.060257 -2.275717 -4.104935
      7             5.152367e+07 -0.072038  0.009820  0.060065  0.065699 -3.637341

      measurements   value 3    activity Category
      0             1.056386  Activity 4   Autism
      1             1.485308  Activity 4   Autism
      2             1.206666  Activity 4   Autism
      3             2.075268  Activity 4   Autism
      4             0.864342  Activity 4   Autism
      5             1.945478  Activity 4   Autism
      6             1.093555  Activity 4   Autism
      7             0.628181  Activity 4   Autism
```

```
[22]: df_average_1_control = pd.DataFrame()
for i in path_activity_1_files_control:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d= {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_1_control = pd.concat([df_average_1_control ,e], axis=1)
df_activity_1_control=df_average_1_control.dropna(axis=1, how="any")
df_activity_1_control_transpose = df_activity_1_control.T
df_activity_1_control_transpose.reset_index(drop=True, inplace=True)
```

```
df_activity_1_control_transpose['activity'] = "Activity 1"
df_activity_1_control_transpose['Category'] = "Control"
df_activity_1_control_transpose
```

[22]:
```
measurements          time      accx      accy      accz   value 1   value 2  \
0             4.331349e+07  0.013761 -0.036507  0.077227 -0.831764 -1.899300
1             6.516636e+07  0.013932 -0.034768  0.080039 -0.995646 -0.773063
2             3.971244e+07 -0.008387 -0.044174  0.076686 -2.336139 -0.903891
3             3.980852e+07 -0.008788 -0.047016  0.075121 -3.075735 -0.747311
4             4.021818e+07 -0.004378 -0.048051  0.074864 -2.852900 -3.356807
5             5.984827e+07  0.033373 -0.035705  0.074678 -1.814165 -2.388022
6             6.144462e+07  0.017282 -0.035049  0.079043 -1.977758 -3.071024

measurements   value 3    activity Category
0            -1.613869  Activity 1  Control
1             0.098677  Activity 1  Control
2             3.918810  Activity 1  Control
3             4.147824  Activity 1  Control
4             2.193568  Activity 1  Control
5            -0.303408  Activity 1  Control
6             1.735413  Activity 1  Control
```

[23]:
```
df_average_2_control = pd.DataFrame()
for i in path_activity_2_files_control:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
 ↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d= {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
 ↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
 ↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
 ↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_2_control = pd.concat([df_average_2_control ,e], axis=1)
df_activity_2_control=df_average_2_control.dropna(axis=1, how="any")
df_activity_2_control_transpose = df_activity_2_control.T
df_activity_2_control_transpose.reset_index(drop=True, inplace=True)
df_activity_2_control_transpose['activity'] = "Activity 2"
```

```
df_activity_2_control_transpose['Category'] = "Control"
df_activity_2_control_transpose
```

[23]:
```
measurements          time      accx      accy      accz   value 1   value 2  \
0             4.113005e+07  0.005294 -0.033949  0.059543 -1.449774 -3.703889
1             4.342686e+07  0.011204 -0.022986  0.062121 -6.595416 -1.069205
2             6.525925e+07  0.011016 -0.034819  0.061407 -1.952180 -1.155815
3             6.105447e+07  0.023131 -0.036939  0.066959  0.670589 -0.149835

measurements   value 3    activity Category
0             0.473900  Activity 2  Control
1             3.727392  Activity 2  Control
2            -0.494543  Activity 2  Control
3             1.569838  Activity 2  Control
```

[24]:
```
df_average_3_control = pd.DataFrame()
for i in path_activity_3_files_control:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d= {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_3_control = pd.concat([df_average_3_control ,e], axis=1)
df_activity_3_control=df_average_3_control.dropna(axis=1, how="any")
df_activity_3_control_transpose = df_activity_3_control.T
df_activity_3_control_transpose.reset_index(drop=True, inplace=True)
df_activity_3_control_transpose['activity'] = "Activity 3"
df_activity_3_control_transpose['Category'] = "Control"
df_activity_3_control_transpose
```

[24]:
```
measurements          time      accx      accy      accz   value 1   value 2  \
0             3.804327e+07 -0.055859  0.002811  0.083171 -0.528052 -2.981148
1             3.935075e+07 -0.063125  0.018824  0.076031 -0.916560 -2.829821
2             4.714029e+07 -0.068943  0.030241  0.067967 -0.780010 -2.599359
```

```
3              4.724336e+07 -0.072841  0.030521  0.063182 -0.817679 -2.380241
4              4.747006e+07 -0.074406  0.031325  0.061504 -1.421655 -2.147237
5              4.757370e+07 -0.074901  0.031966  0.059451 -0.656969 -2.438388
6              4.777272e+07 -0.074521  0.030696  0.058717 -1.649565 -1.887135
7              4.786611e+07 -0.077665  0.030095  0.057661 -1.338912 -2.244394
8              4.796363e+07 -0.076950  0.035151  0.052385 -0.672078 -2.755308
9              4.804827e+07 -0.079935  0.030028  0.053596 -0.994333 -2.345595
10             4.815376e+07 -0.077318  0.029672  0.057909 -0.794311 -2.411319
11             4.827358e+07 -0.070523  0.012429  0.071335 -0.722332 -3.239399

measurements   value 3    activity Category
0              0.137463  Activity 3  Control
1              0.511944  Activity 3  Control
2              1.141107  Activity 3  Control
3              0.993927  Activity 3  Control
4              0.992178  Activity 3  Control
5              0.710262  Activity 3  Control
6              1.153094  Activity 3  Control
7              1.292613  Activity 3  Control
8              0.792830  Activity 3  Control
9              1.184219  Activity 3  Control
10             1.050141  Activity 3  Control
11             0.472131  Activity 3  Control
```

[25]:

```
df_average_4_control = pd.DataFrame()
for i in path_activity_1_files_control:
    df = pd.read_csv(i)
    df['accX[g]'] = df['accX[g]']/9.81
    df['accY[g]']= df['accY[g]']/9.81
    df['accZ[g]'] = df['accZ[g]']/9.81
    df['Timedelta'] = pd.to_timedelta(df['time[ms]'],'ms')
    df = df.set_index(df['Timedelta'])[['time[ms]','accX[g]','accY[g]',␣
↪'accZ[g]', 'Value 1', 'Value 2', 'Value 3']].resample('1s').mean()
    df['accX[g]'] = butter_lowpass_filter(df['accX[g]'], cutoff, fs, order)
    df['accY[g]'] = butter_lowpass_filter(df['accY[g]'], cutoff, fs, order)
    df['accZ[g]'] = butter_lowpass_filter(df['accZ[g]'], cutoff, fs, order)
    df['Value 1'] = butter_lowpass_filter(df['Value 1'], cutoff, fs, order)
    df['Value 2'] = butter_lowpass_filter(df['Value 2'], cutoff, fs, order)
    df['Value 3'] = butter_lowpass_filter(df['Value 3'], cutoff, fs, order)
    d= {'measurements': ['time', 'accx', 'accy', 'accz', 'value 1', 'value 2',␣
↪'value 3'],'Column 1': [df['time[ms]'].mean(), df['accX[g]'].mean(),␣
↪df['accY[g]'].mean(), df['accZ[g]'].mean(), df['Value 1'].mean(), df['Value␣
↪2'].mean(), df['Value 3'].mean()]}
    e = pd.DataFrame(d).set_index('measurements').rename_axis('columns', axis=1)
    df_average_4_control = pd.concat([df_average_4_control ,e], axis=1)
df_activity_4_control=df_average_4_control.dropna(axis=1, how="any")
df_activity_4_control_transpose = df_activity_4_control.T
```

```
df_activity_4_control_transpose.reset_index(drop=True, inplace=True)
df_activity_4_control_transpose['activity'] = "Activity 4"
df_activity_4_control_transpose['Category'] = "Control"
df_activity_4_control_transpose
```

[25]:
```
measurements            time      accx      accy      accz   value 1   value 2  \
0             4.331349e+07  0.013761 -0.036507  0.077227 -0.831764 -1.899300
1             6.516636e+07  0.013932 -0.034768  0.080039 -0.995646 -0.773063
2             3.971244e+07 -0.008387 -0.044174  0.076686 -2.336139 -0.903891
3             3.980852e+07 -0.008788 -0.047016  0.075121 -3.075735 -0.747311
4             4.021818e+07 -0.004378 -0.048051  0.074864 -2.852900 -3.356807
5             5.984827e+07  0.033373 -0.035705  0.074678 -1.814165 -2.388022
6             6.144462e+07  0.017282 -0.035049  0.079043 -1.977758 -3.071024

measurements   value 3    activity Category
0            -1.613869  Activity 4  Control
1             0.098677  Activity 4  Control
2             3.918810  Activity 4  Control
3             4.147824  Activity 4  Control
4             2.193568  Activity 4  Control
5            -0.303408  Activity 4  Control
6             1.735413  Activity 4  Control
```

[26]:
```
frames = [df_activity_1_control_transpose, df_activity_1_autism_transpose,␣
 ↪df_activity_2_control_transpose, df_activity_2_autism_transpose,␣
 ↪df_activity_3_control_transpose,␣
 ↪df_activity_3_autism_transpose,df_activity_4_control_transpose,␣
 ↪df_activity_4_autism_transpose]
result = pd.concat(frames)
result
```

[26]:
```
measurements            time      accx      accy      accz   value 1   value 2  \
0             4.331349e+07  0.013761 -0.036507  0.077227 -0.831764 -1.899300
1             6.516636e+07  0.013932 -0.034768  0.080039 -0.995646 -0.773063
2             3.971244e+07 -0.008387 -0.044174  0.076686 -2.336139 -0.903891
3             3.980852e+07 -0.008788 -0.047016  0.075121 -3.075735 -0.747311
4             4.021818e+07 -0.004378 -0.048051  0.074864 -2.852900 -3.356807
..                     …         …         …         …         …         …
3             5.080718e+07 -0.067720  0.015585  0.065191 -0.446576 -3.389572
4             5.110959e+07 -0.070735  0.011466  0.065945 -0.367087 -2.983890
5             5.120279e+07 -0.057154  0.036216  0.063378 -1.530486 -2.960564
6             5.140863e+07 -0.061707  0.018761  0.060257 -2.275717 -4.104935
7             5.152367e+07 -0.072038  0.009820  0.060065  0.065699 -3.637341

measurements   value 3    activity Category
0            -1.613869  Activity 1  Control
1             0.098677  Activity 1  Control
```

```
2             3.918810  Activity 1  Control
3             4.147824  Activity 1  Control
4             2.193568  Activity 1  Control
..                 …           …        …
3             2.075268  Activity 4   Autism
4             0.864342  Activity 4   Autism
5             1.945478  Activity 4   Autism
6             1.093555  Activity 4   Autism
7             0.628181  Activity 4   Autism

[64 rows x 9 columns]
```

Now All preprocessed data have been placed in a dataframe called result. Let's see if we can make predictions on the data. Prediction is to be done on the ativity and control column.

[27]:
```python
import seaborn as sns
%matplotlib inline

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

import statsmodels.api as sm #Linear Regression
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import log_loss, accuracy_score

from keras.callbacks import EarlyStopping, ModelCheckpoint


import keras
from keras.models import Sequential # intitialize the ANN
from keras.layers import Dense, Dropout      # create layers

from sklearn.ensemble import BaggingClassifier

categorical = ['activity', 'Category']

lbl = preprocessing.LabelEncoder()
for col in categorical:
    result[col] = lbl.fit_transform(result[col].astype(str))
print(result[['activity','Category']])
```

```
Using TensorFlow backend.

measurements  activity  Category
```

```
0                    0         1
1                    0         1
2                    0         1
3                    0         1
4                    0         1
..                 …        …
3                    3         0
4                    3         0
5                    3         0
6                    3         0
7                    3         0

[64 rows x 2 columns]
```

Divide the dataset into X_train, y_train, X_test, y_test, ratio 80:20, random_state = 109, stratify on `Category`.

Check the ratio for y_train and y_test

[28]:
```
X_train, X_test, y_train, y_test = train_test_split(result.loc[:, result.
↪columns != 'Category'], result.Category, test_size=0.2, random_state=109,␣
↪stratify=result.Category)
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

[28]:
```
(0    0.529412
 1    0.470588
 Name: Category, dtype: float64, 0    0.538462
 1    0.461538
 Name: Category, dtype: float64)
```

[29]:
```
X_train.shape
```

[29]:
```
(51, 8)
```

[30]:
```
X_test.shape
```

[30]:
```
(13, 8)
```

1.6.6 Evaluate different models to see the optimal

Linear Regression - Ordinary Least Squares

[31]:
```
model = sm.OLS(y_train, X_train).fit()
print_model = model.summary()
linear_reg_log_loss = log_loss(y_test, model.predict(X_test))
print(linear_reg_log_loss)
print(accuracy_score(y_test, model.predict(X_test)>0.5),␣
↪accuracy_score(y_test,np.zeros(y_test.shape)))
```

```
0.4508026704427284
0.8461538461538461 0.5384615384615384
```

Logistic Regression

[32]:
```
model = LogisticRegression(C=10000, random_state=109, max_iter=1000,␣
 ↪solver='liblinear')
model.fit(X_train,y_train)
preds = model.predict_proba(X_test)
logistic_reg_log_loss = log_loss(y_test, preds[:,1])
print(logistic_reg_log_loss)
print(accuracy_score(y_test, preds[:,1]>0.5), accuracy_score(y_test,np.
 ↪zeros(y_test.shape)))
```

```
0.6832364425488933
0.5384615384615384 0.5384615384615384
```

Logistic Regression —Scaled

[33]:
```
scaler = StandardScaler().fit(X_train.astype(np.float64))
train_scaled = scaler.transform(X_train.astype(np.float64))
test_scaled = scaler.transform(X_test.astype(np.float64))
model = LogisticRegression(C=0.11, random_state=109,penalty='l2'␣
 ↪,max_iter=1000, solver='liblinear')
model.fit(train_scaled,y_train)
preds = model.predict_proba(test_scaled)
reg_log_loss = log_loss(y_test, preds[:,1])
print(reg_log_loss)
```

```
0.5313932236466319
```

Bagging Classifier

[34]:
```
model = BaggingClassifier(random_state=109, n_estimators=100)
model.fit(X_train, y_train)
preds = model.predict_proba(X_test)
bagging_log_loss = log_loss(y_test, preds[:,1])
print(bagging_log_loss)
```

```
0.029101113621990724
```

Random Forest

[35]:
```
model = RandomForestClassifier(n_estimators = 500, random_state = 109)
model.fit(X_train,y_train)
preds = model.predict_proba(X_test)
rf_log_loss = log_loss(y_test, preds[:,1])
print(rf_log_loss)
```

```
0.0661655489905261
```

Neural Network

[36]:
```
# Initializing the NN
model = Sequential()
# layers
model.add(Dense(units = 40, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu', input_dim = 8))
model.add(Dense(units = 30, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dense(units = 25, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dense(units = 20, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dense(units = 15, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dense(units = 10, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dense(units = 5, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1, kernel_initializer = 'glorot_uniform', activation =␣
 ↪'sigmoid'))
print(model.summary())
# Compiling the ANN
model.compile(optimizer = 'Nadam',
              loss = 'binary_crossentropy' ,
              metrics = ['accuracy'])

# Set callback functions to early stop training and save the best model so far
callbacks = [EarlyStopping(monitor='val_loss', patience=4),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',␣
 ↪save_best_only=True)]

# Train the ANN
history = model.fit(train_scaled, y_train, batch_size = 256, epochs = 50,
                    callbacks=callbacks, # Early stopping
                    validation_data = (test_scaled, y_test))

nn_log_loss = model.evaluate(test_scaled, y_test)[0]
print(nn_log_loss)
```

```
Model: "sequential_1"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 40)                360

_________________________________________________________________
```

```
dense_2 (Dense)              (None, 30)                1230
_________________________________________________________________
dense_3 (Dense)              (None, 25)                775
_________________________________________________________________
dense_4 (Dense)              (None, 20)                520
_________________________________________________________________
dense_5 (Dense)              (None, 15)                315
_________________________________________________________________
dense_6 (Dense)              (None, 10)                160
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 55
_________________________________________________________________
dropout_1 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 6
=================================================================
Total params: 3,421
Trainable params: 3,421
Non-trainable params: 0
_________________________________________________________________
None
Train on 51 samples, validate on 13 samples
Epoch 1/50
51/51 [==============================] - 2s 29ms/step - loss: 0.7008 - accuracy:
0.4510 - val_loss: 0.6950 - val_accuracy: 0.5385
Epoch 2/50
51/51 [==============================] - 0s 346us/step - loss: 0.6918 -
accuracy: 0.4706 - val_loss: 0.6943 - val_accuracy: 0.5385
Epoch 3/50
51/51 [==============================] - 0s 315us/step - loss: 0.6901 -
accuracy: 0.5098 - val_loss: 0.6937 - val_accuracy: 0.5385
Epoch 4/50
51/51 [==============================] - 0s 384us/step - loss: 0.6841 -
accuracy: 0.5686 - val_loss: 0.6920 - val_accuracy: 0.4615
Epoch 5/50
51/51 [==============================] - 0s 475us/step - loss: 0.6799 -
accuracy: 0.5882 - val_loss: 0.6917 - val_accuracy: 0.6154
Epoch 6/50
51/51 [==============================] - 0s 196us/step - loss: 0.6718 -
accuracy: 0.6667 - val_loss: 0.6883 - val_accuracy: 0.5385
Epoch 7/50
51/51 [==============================] - 0s 397us/step - loss: 0.6746 -
accuracy: 0.6078 - val_loss: 0.6872 - val_accuracy: 0.6923
Epoch 8/50
51/51 [==============================] - 0s 370us/step - loss: 0.6667 -
accuracy: 0.6275 - val_loss: 0.6849 - val_accuracy: 0.6923
Epoch 9/50
51/51 [==============================] - 0s 426us/step - loss: 0.6607 -
```

```
accuracy: 0.7255 - val_loss: 0.6839 - val_accuracy: 0.6923
Epoch 10/50
51/51 [==============================] - 0s 235us/step - loss: 0.6541 -
accuracy: 0.7255 - val_loss: 0.6801 - val_accuracy: 0.5385
Epoch 11/50
51/51 [==============================] - 0s 321us/step - loss: 0.6530 -
accuracy: 0.6471 - val_loss: 0.6810 - val_accuracy: 0.7692
Epoch 12/50
51/51 [==============================] - 0s 223us/step - loss: 0.6529 -
accuracy: 0.6863 - val_loss: 0.6744 - val_accuracy: 0.7692
Epoch 13/50
51/51 [==============================] - 0s 281us/step - loss: 0.6353 -
accuracy: 0.6667 - val_loss: 0.6716 - val_accuracy: 0.6923
Epoch 14/50
51/51 [==============================] - 0s 307us/step - loss: 0.6424 -
accuracy: 0.7059 - val_loss: 0.6691 - val_accuracy: 0.6923
Epoch 15/50
51/51 [==============================] - 0s 315us/step - loss: 0.6176 -
accuracy: 0.8431 - val_loss: 0.6647 - val_accuracy: 0.7692
Epoch 16/50
51/51 [==============================] - 0s 295us/step - loss: 0.6335 -
accuracy: 0.6863 - val_loss: 0.6582 - val_accuracy: 0.8462
Epoch 17/50
51/51 [==============================] - 0s 297us/step - loss: 0.6176 -
accuracy: 0.6471 - val_loss: 0.6687 - val_accuracy: 0.7692
Epoch 18/50
51/51 [==============================] - 0s 304us/step - loss: 0.6040 -
accuracy: 0.7255 - val_loss: 0.6533 - val_accuracy: 0.7692
Epoch 19/50
51/51 [==============================] - 0s 258us/step - loss: 0.5869 -
accuracy: 0.7059 - val_loss: 0.6568 - val_accuracy: 0.7692
Epoch 20/50
51/51 [==============================] - 0s 204us/step - loss: 0.5836 -
accuracy: 0.8431 - val_loss: 0.6441 - val_accuracy: 0.7692
Epoch 21/50
51/51 [==============================] - 0s 240us/step - loss: 0.5617 -
accuracy: 0.8431 - val_loss: 0.6488 - val_accuracy: 0.6923
Epoch 22/50
51/51 [==============================] - 0s 238us/step - loss: 0.5465 -
accuracy: 0.8627 - val_loss: 0.6351 - val_accuracy: 0.7692
Epoch 23/50
51/51 [==============================] - 0s 266us/step - loss: 0.5328 -
accuracy: 0.8824 - val_loss: 0.6171 - val_accuracy: 0.7692
Epoch 24/50
51/51 [==============================] - 0s 516us/step - loss: 0.5127 -
accuracy: 0.8431 - val_loss: 0.6000 - val_accuracy: 0.8462
Epoch 25/50
51/51 [==============================] - 0s 351us/step - loss: 0.4847 -
```

```
accuracy: 0.8824 - val_loss: 0.5951 - val_accuracy: 1.0000
Epoch 26/50
51/51 [==============================] - 0s 525us/step - loss: 0.5032 -
accuracy: 0.7843 - val_loss: 0.5673 - val_accuracy: 1.0000
Epoch 27/50
51/51 [==============================] - 0s 433us/step - loss: 0.4915 -
accuracy: 0.8235 - val_loss: 0.5626 - val_accuracy: 1.0000
Epoch 28/50
51/51 [==============================] - 0s 407us/step - loss: 0.4512 -
accuracy: 0.9020 - val_loss: 0.5166 - val_accuracy: 1.0000
Epoch 29/50
51/51 [==============================] - 0s 420us/step - loss: 0.4212 -
accuracy: 0.9020 - val_loss: 0.4854 - val_accuracy: 1.0000
Epoch 30/50
51/51 [==============================] - 0s 370us/step - loss: 0.4048 -
accuracy: 0.9216 - val_loss: 0.4405 - val_accuracy: 1.0000
Epoch 31/50
51/51 [==============================] - 0s 319us/step - loss: 0.3957 -
accuracy: 0.8431 - val_loss: 0.4077 - val_accuracy: 1.0000
Epoch 32/50
51/51 [==============================] - 0s 339us/step - loss: 0.3785 -
accuracy: 0.9216 - val_loss: 0.3946 - val_accuracy: 1.0000
Epoch 33/50
51/51 [==============================] - 0s 338us/step - loss: 0.3286 -
accuracy: 0.9216 - val_loss: 0.3551 - val_accuracy: 0.9231
Epoch 34/50
51/51 [==============================] - 0s 302us/step - loss: 0.3173 -
accuracy: 0.9412 - val_loss: 0.2956 - val_accuracy: 1.0000
Epoch 35/50
51/51 [==============================] - 0s 273us/step - loss: 0.2684 -
accuracy: 0.9412 - val_loss: 0.2133 - val_accuracy: 1.0000
Epoch 36/50
51/51 [==============================] - 0s 286us/step - loss: 0.2609 -
accuracy: 0.9412 - val_loss: 0.2274 - val_accuracy: 1.0000
Epoch 37/50
51/51 [==============================] - 0s 254us/step - loss: 0.2788 -
accuracy: 0.9608 - val_loss: 0.1867 - val_accuracy: 1.0000
Epoch 38/50
51/51 [==============================] - 0s 229us/step - loss: 0.2623 -
accuracy: 0.8627 - val_loss: 0.1533 - val_accuracy: 1.0000
Epoch 39/50
51/51 [==============================] - 0s 462us/step - loss: 0.2241 -
accuracy: 0.9608 - val_loss: 0.1320 - val_accuracy: 1.0000
Epoch 40/50
51/51 [==============================] - 0s 316us/step - loss: 0.1726 -
accuracy: 0.9412 - val_loss: 0.1101 - val_accuracy: 1.0000
Epoch 41/50
51/51 [==============================] - 0s 413us/step - loss: 0.1458 -
```

```
accuracy: 0.9608 - val_loss: 0.1026 - val_accuracy: 1.0000
Epoch 42/50
51/51 [==============================] - 0s 467us/step - loss: 0.1508 -
accuracy: 0.9804 - val_loss: 0.0877 - val_accuracy: 1.0000
Epoch 43/50
51/51 [==============================] - 0s 401us/step - loss: 0.0813 -
accuracy: 0.9804 - val_loss: 0.0825 - val_accuracy: 1.0000
Epoch 44/50
51/51 [==============================] - 0s 411us/step - loss: 0.0993 -
accuracy: 0.9804 - val_loss: 0.0655 - val_accuracy: 1.0000
Epoch 45/50
51/51 [==============================] - 0s 273us/step - loss: 0.1366 -
accuracy: 0.9608 - val_loss: 0.0587 - val_accuracy: 1.0000
Epoch 46/50
51/51 [==============================] - 0s 483us/step - loss: 0.0967 -
accuracy: 0.9804 - val_loss: 0.0508 - val_accuracy: 1.0000
Epoch 47/50
51/51 [==============================] - 0s 335us/step - loss: 0.0812 -
accuracy: 0.9804 - val_loss: 0.0425 - val_accuracy: 1.0000
Epoch 48/50
51/51 [==============================] - 0s 406us/step - loss: 0.1283 -
accuracy: 0.9608 - val_loss: 0.0559 - val_accuracy: 1.0000
Epoch 49/50
51/51 [==============================] - 0s 365us/step - loss: 0.1251 -
accuracy: 0.9412 - val_loss: 0.0342 - val_accuracy: 1.0000
Epoch 50/50
51/51 [==============================] - 0s 373us/step - loss: 0.0831 -
accuracy: 0.9608 - val_loss: 0.0575 - val_accuracy: 1.0000
13/13 [==============================] - 0s 223us/step
0.05746658891439438
```

Create a table to compare all the validation log losses.

[42]:
```python
data = {'Model':['Linear Regression', 'Logistic Regression', 'Regularized␣
↪Logistic regression', 'Bagged decision tree', 'Random Forest', 'NN Model'],
        'Logloss':[linear_reg_log_loss , logistic_reg_log_loss, reg_log_loss,␣
↪bagging_log_loss, rf_log_loss, nn_log_loss]}
pd.DataFrame(data).sort_values('Logloss')
```

[42]:

	Model	Logloss
3	Bagged decision tree	0.029101
5	NN Model	0.057467
4	Random Forest	0.066166
0	Linear Regression	0.450803
2	Regularized Logistic regression	0.531393
1	Logistic Regression	0.683236

Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0. The Bagging classifier performed best in this model.

[41]:
```
from sklearn import model_selection
model = BaggingClassifier(random_state=109, n_estimators=100)
model.fit(X_train, y_train)
preds = model.predict_proba(X_test)
bagging_log_loss = log_loss(y_test, preds[:,1])
print(bagging_log_loss)
seed = 7
kfold = model_selection.KFold(n_splits=5, random_state=seed)
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean())
```

```
0.029101113621990724
0.9018181818181817
```

To improve the accuracy of the bagging classifier, I applied a 5 fold CV. Which resulted in a robust estimate of model accuracy.

1.6.7 Conclusion

Given the successful performance of the Bagging classifier and the neural network in this experiment more sample size and subjects will be needed to come to a definite conclusion on the use of motion sensors in convienent ways that will lead to discoveries that are difficult to evaluate by humans. Other classifiers as it relates to the types of activities and the category of motor with delays can also be build using the data collected.