Data Analysis and Recommendation for Colibri Wireless unit

In Trivisio, we have developed the new Colibri Wireless, which is an inertial measurement unit (IMU). The Colibri Wireless consists of 3-axis state-of-art sensors to measure acceleration, angular rate, magnetic field and a built-in temperature sensor. The primary objective of this unit is to capture user activities, recording timestamps for calculating the total duration of each activity.

This report aims to enhance the competitiveness of our unit in the market. To achieve this goal, we are going to present recommendations based on the results of data analysis and a mathematical model devised by the Trivisio data team.

In [1]:
```
"""
Importing libraries for data analysis and creating a mathematical model
"""

import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

How We Processed the Data

First of all, data was gathered from nine individuals, identified as subjects 101 to 109. According to the data collection protocol, each subject participated in twelve different activities, including lying, sitting, standing, ironing, vacuuming, ascending stairs, descending stairs, normal walking, Nordic walking, cycling, running, and rope jumping. Moreover, five subjects (101, 105, 106, 108, 109) conducted six optional activities: watching TV, computer work, car driving, folding laundry, house cleaning, and playing soccer.

Each data was saved in different files thus the data team merged all individual data files into a single dataset called Physical Activity Monitoring Data to facilitate analysis.

In [2]:
```
"""
Defining a function that checks if a file exists in the directory
"""

def check_file_exist(path, file):
    file_path = os.path.join(path, file)
    return os.path.exists(file_path)
```

In [3]:
```
"""
Merging all datasets into one dataframe
"""

# Data file name, unit parts, features list
data_from = ['subject101.dat','subject102.dat','subject103.dat','subject104.dat','subject105.dat','subject106.d
             'subject107.dat','subject108.dat','subject109.dat']

class_ = ['arm_', 'chest_','ankle_']

subset = ['temperature', '3D_acc_16g_x', '3D_acc_16g_y', '3D_acc_16g_z'
    , '3D_acc_6g_x', '3D_acc_6g_y', '3D_acc_6g_z',
    '3D_gyro_x','3D_gyro_y','3D_gyro_z','3D_magnet_x','3D_magnet_y','3D_magnet_z',
    'orientation_1','orientation_2','orientation_3','orientation_4']

list_columns = ['timestamp', 'activityID', 'bpm']

for i in class_:
    for i2 in subset:
        together = i + i2
        list_columns.append(together)

path1 = "Protocol/"
path2 = "Optional/"
df_final = pd.DataFrame(columns=list_columns)
for file_name in data_from:
    if check_file_exist(path1, file_name) and check_file_exist(path2, file_name):
        address = path1 + file_name
        df = pd.read_csv(address, delimiter = ' ')
        first_value = df.columns
        dic = {}
        for i, i2 in zip(list_columns,first_value):
            dic[i] = i2
        first_value = pd.DataFrame(data = dic, index = [0])
        df.columns = list_columns
        df = pd.concat([first_value,df], axis = 0,ignore_index = True)
        address2 = path2 + file_name
        df2 = pd.read_csv(address2, delimiter = ' ')
        first_value = df2.columns
        dic = {}
        for i, i2 in zip(list_columns,first_value):
            dic[i] = i2
```

```
        first_value = pd.DataFrame(data = dic, index = [0])
        df2.columns = list_columns
        df2 = pd.concat([first_value,df2], axis = 0,ignore_index = True)
        df = pd.concat([df,df2], axis = 0,ignore_index = True)
        df['ID'] = int(file_name.split('subject')[1].split('.')[0])
        df_final = pd.concat([df_final, df], axis = 0, ignore_index = True)
    else:
        address = path1 + file_name
        df = pd.read_csv(address, delimiter = ' ')
        first_value = df.columns
        dic = {}
        for i, i2 in zip(list_columns,first_value):
            dic[i] = i2
        first_value = pd.DataFrame(data = dic, index = [0])
        df.columns = list_columns
        df = pd.concat([first_value,df], axis = 0,ignore_index = True)
        df['ID'] = int(file_name.split('subject')[1].split('.')[0])
        df_final = pd.concat([df_final,df], axis = 0, ignore_index = True)
```

Following the consolidation of all datasets into a single dataset, certain variables were excluded from further analysis.

The 3D acceleration data obtained from chest, arm, and ankle units, with a scale of ±6g, was removed. This decision was made by instances where high-impact movements led to saturation of acceleration beyond 6g, rendering the data imprecise and not accurately calibrated with the 3D accelerometer having a ±16g scale.

In addition, all orientation data from each sensor component was deemed invalid during the data collection process and, as a result, was excluded from further data analysis.

In [4]:
```
"""
Eliminating some columns that will not be utilised in data analysis
"""

df_final.drop(columns=['arm_3D_acc_6g_x',
       'arm_3D_acc_6g_y', 'arm_3D_acc_6g_z','chest_3D_acc_6g_x', 'chest_3D_acc_6g_y',
       'chest_3D_acc_6g_z',  'ankle_3D_acc_6g_x', 'ankle_3D_acc_6g_y', 'ankle_3D_acc_6g_z','ankle_orientation_1
       'ankle_orientation_4', 'chest_orientation_1', 'chest_orientation_2',
       'chest_orientation_3', 'chest_orientation_4', 'arm_orientation_1',
       'arm_orientation_2', 'arm_orientation_3', 'arm_orientation_4'], inplace = True)
```

In [5]:
```
"""
To construct a mathematical model and perform data analysis, all variables excluding SubjectID were converted t
"""

cols = ['timestamp', 'activityID', 'bpm', 'arm_temperature', 'arm_3D_acc_16g_x',
       'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z', 'arm_3D_gyro_x',
       'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x', 'arm_3D_magnet_y',
       'arm_3D_magnet_z', 'chest_temperature', 'chest_3D_acc_16g_x',
       'chest_3D_acc_16g_y', 'chest_3D_acc_16g_z', 'chest_3D_gyro_x',
       'chest_3D_gyro_y', 'chest_3D_gyro_z', 'chest_3D_magnet_x',
       'chest_3D_magnet_y', 'chest_3D_magnet_z', 'ankle_temperature',
       'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y', 'ankle_3D_acc_16g_z',
       'ankle_3D_gyro_x', 'ankle_3D_gyro_y', 'ankle_3D_gyro_z',
       'ankle_3D_magnet_x', 'ankle_3D_magnet_y', 'ankle_3D_magnet_z']

for column in cols:
    try:
        df_final[column] = df_final[column].astype(float)
    # Error detection
    except ValueError as e:
        print(f"Error in column {column} to float: {e}")
```

```
Error in column ankle_temperature to float: could not convert string to float: '29.5.1'
```

In [6]:
```
"""
Addressing the error detected in the previous stage
"""

df_final[df_final['ankle_temperature']== '29.5.1']
error_index = df_final[df_final['ankle_temperature']== '29.5.1'].index
df_final.drop(index=error_index, inplace = True)
```

In [7]:
```
"""
Changing the data type to float for 'ankle_temperature' variable
"""

df_final['ankle_temperature'] = df_final['ankle_temperature'].astype(float)
```

The data team detected a total of 3,501,717 missing values. This is due to the data collected from the new Colibri Wireless unit were recorded at a frequency of 100Hz, with measurements taken every 0.01 seconds. Conversely, the heart rate monitor had a lower sampling frequency of approximately 9Hz, equivalent to 0.1-second intervals. Thus, numerous missing values were detected in the dataset. As a result of a discussion with the data collection officer, we determined to drop all missing values.

In [8]:
```
"""
To identify missing values
```

```
"""

df_final[df_final.isna().any(axis=1)]
```

Out[8]:

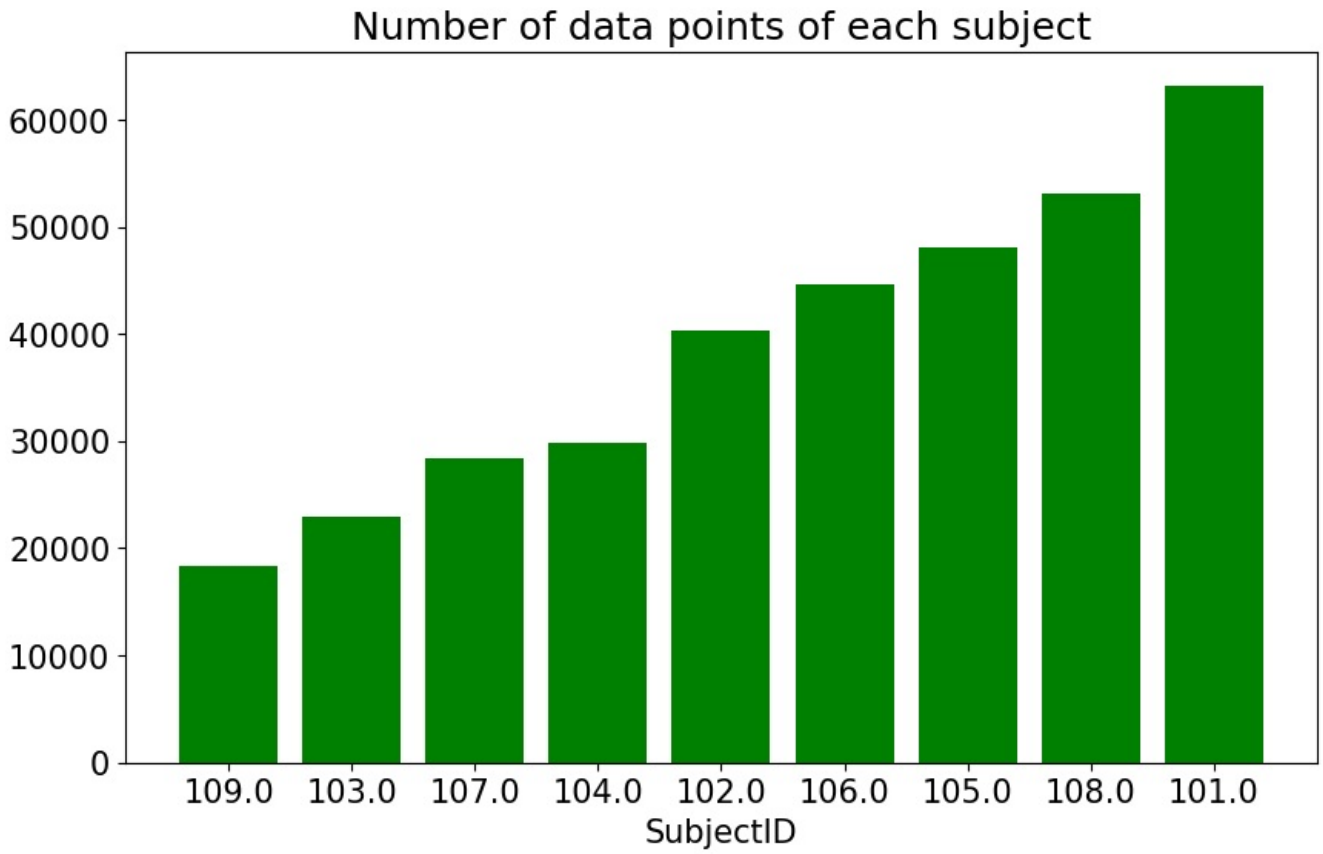
	timestamp	activityID	bpm	arm_temperature	arm_3D_acc_16g_x	arm_3D_acc_16g_y	arm_3D_acc_16g_z	arm_3D_gyro_x	arm_3D_
1	8.39	0.0	NaN	30.000	2.18837	8.56560	3.66179	-0.024413	0.
2	8.40	0.0	NaN	30.000	2.37357	8.60107	3.54898	-0.057976	0.
3	8.41	0.0	NaN	30.000	2.07473	8.52853	3.66021	-0.002352	0.
4	8.42	0.0	NaN	30.000	2.22936	8.83122	3.70000	0.012269	0.
5	8.43	0.0	NaN	30.000	2.29959	8.82929	3.54710	0.003238	0.
...	...	...	...	...	...	...	...	...	
3850500	1940.78	0.0	NaN	24.875	-6.51915	6.87682	2.34395	0.007735	0.
3850501	1940.79	0.0	NaN	24.875	-6.47548	6.95375	2.49829	0.011587	0.
3850502	1940.80	0.0	NaN	24.875	-6.58698	6.84148	2.57398	0.020748	0.
3850503	1940.81	0.0	NaN	NaN	NaN	NaN	NaN	NaN	
3850504	1940.82	0.0	176.0	NaN	NaN	NaN	NaN	NaN	

3501717 rows × 34 columns

In [9]:
```
"""
Dropping all the missing values
"""

df_final.dropna(inplace = True)
```

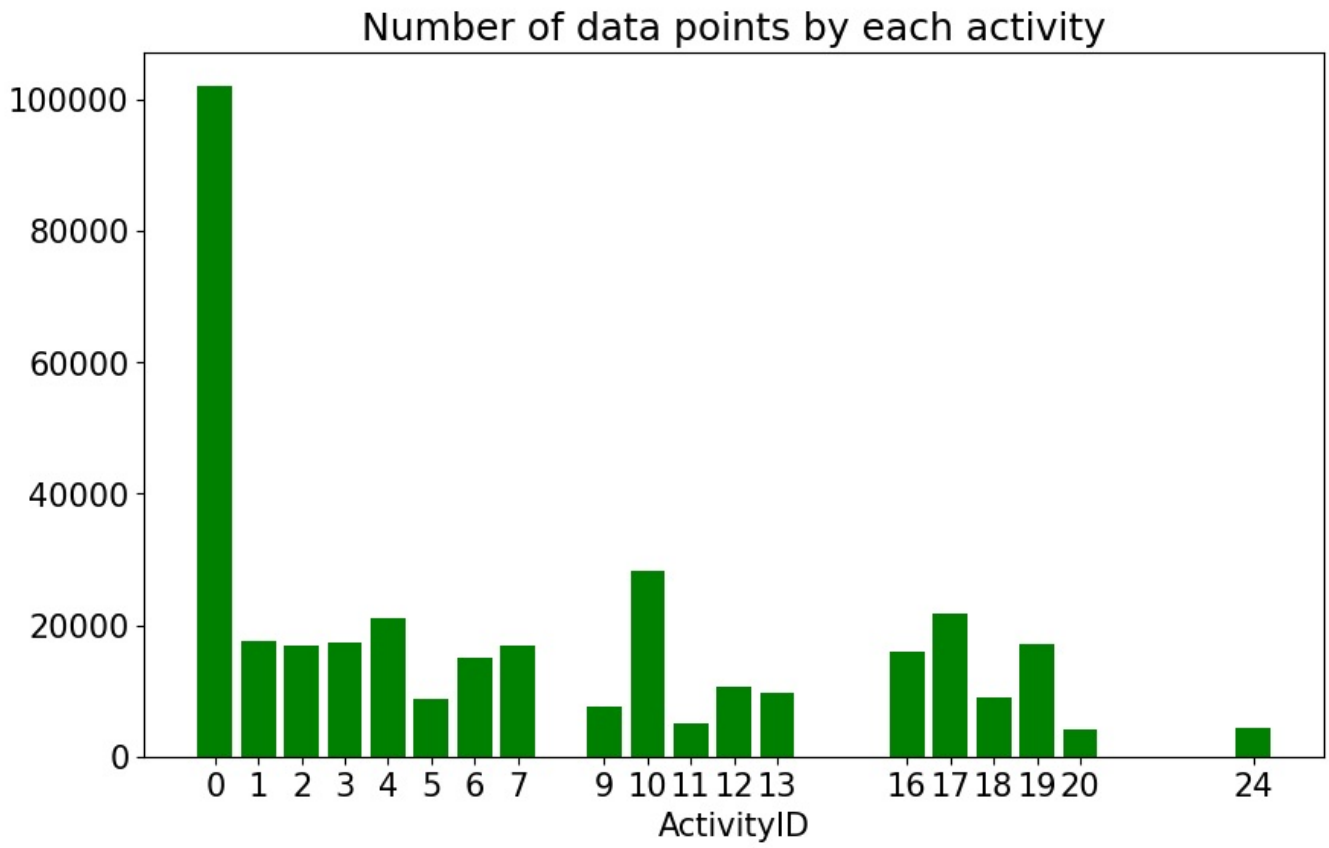
In sequence, we examined the quantity of data points for each subject and activity. Accordingly, we observed that subject 101 engaged in activities for the longest duration, while subject 109 exhibited the opposite trend. Additionally, the data team identified an imbalance in the distribution of data points across various activities. When we conducted this analysis, we did not take into account the activity code '0' because it contains all transient activities and we have decided to exclude this code from the dataset in subsequent steps.

In [10]:
```
"""
Drawing a bar chart to check the number of data points of each subject
"""

samples = df_final.groupby(['ID']).count().reset_index()

plt.rcParams.update({'font.size': 15})

samples_df = pd.DataFrame({'ID': samples['ID'], 'samples': samples['timestamp']})
samples_df = samples_df.sort_values(by=['samples'])

fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(range(len(samples_df)), samples_df['samples'], color='green')
ax.set_xticks(np.arange(len(samples_df)))
ax.set_title("Number of data points of each subject")
ax.set_xticklabels(samples_df['ID'])
ax.set_xlabel('SubjectID')

plt.show()
```



In [11]:

```
"""
Drawing a bar chart to check the number of data points by each activity
"""

samples = df_final.groupby(['activityID']).count().reset_index()

plt.rcParams.update({'font.size': 15})

plt.subplots(figsize=(10, 6))
plt.bar(samples['activityID'], samples['timestamp'], color='green')

plt.xticks(samples['activityID'])
plt.title('Number of data points by each activity')
plt.xlabel('ActivityID')
plt.show()
```



Furthermore, to handle the extensive volume of data, we determined to narrow down the data into 1-second intervals, based on

timestamp values. This involved calculating the average of all values within each 1-second interval from both the sensors and the heart rate monitor. Subsequently, the original values were replaced with their corresponding average values.

Besides, transient activities labelled as "other" were excluded from the dataset because this data only covers occasional activities occurring between different activities such as waiting to acquire equipment for the next activity.

In [12]:
```
"""
Data compression with a 1-second interval (from a 0.01-second interval)
-> Using the mean value of all numerical data within a 1-second interval
"""

data_columns = df_final.columns.difference(['timestamp', 'ID','activityID'])
df_final['timestamp'] = df_final['timestamp'].astype(float)
df_final['timestamp'] = np.floor(df_final['timestamp'])
df_final = df_final.groupby(['timestamp', 'ID','activityID'])[data_columns].mean(numeric_only=True).reset_index
```

In [13]:
```
"""
Excluding the rows with activityID '0' from the dataset.
"""

delete = df_final[df_final['activityID']==0].index
df_final.drop(index=delete, inplace = True)
df_final = df_final.reset_index().drop(columns='index')
```

In [14]:
```
"""
Sorting the data by 'ID' and 'timestamp'
"""

df_final.sort_values(by= ['ID', 'timestamp'], inplace = True)
df_final
```

Out[14]:

	timestamp	ID	activityID	ankle_3D_acc_16g_x	ankle_3D_acc_16g_y	ankle_3D_acc_16g_z	ankle_3D_gyro_x	ankle_3D_gyro_y	ankle
6	37.0	101.0	1.0	9.749530	-1.923307	0.057149	0.007516	-0.026196	
8	38.0	101.0	1.0	9.666413	-1.887806	-0.093994	0.003784	-0.022030	
10	39.0	101.0	1.0	9.727602	-1.741162	-0.017098	0.021087	0.016340	
12	40.0	101.0	1.0	9.746160	-1.126264	0.094616	-0.005636	-0.016094	
14	41.0	101.0	1.0	9.805887	-1.012875	0.112105	-0.000508	-0.008577	
...	...	...	...	...	...	...	...	...	
18402	1932.0	109.0	20.0	9.469699	-0.558191	-3.012414	-0.000410	-0.025951	
18410	1933.0	109.0	20.0	9.422736	-0.726419	-3.169429	-0.024273	-0.000921	
18418	1934.0	109.0	20.0	9.415436	-0.412814	-3.338671	0.000428	-0.001009	
18426	1935.0	109.0	20.0	9.381788	-0.180110	-3.396201	0.009619	-0.006896	
18434	1936.0	109.0	20.0	9.419690	-0.487734	-3.466790	-0.026614	-0.022865	

27360 rows × 34 columns

Subsequently, we removed the total of 13 duplicated entries identified by identical values in the 'timestamp,' 'arm_temperature,' and 'ID' variables.

Following that, each resultant feature was computed by the data team. The resultant values are important for several reasons. They offer comprehensive information captured by the sensors, enabling us to conduct a three-dimensional analysis based on these derived features. The data team employed the following formula in this process.

Acceleration : $A = \sqrt{a_x^2 + a_y^2 + a_z^2}$

Angular rate : $G = \sqrt{g_x^2 + g_y^2 + g_z^2}$

Magnetic Field: Same with the acceleration and angular rate formula

In consequence, the resultants of acceleration, angular rate, and magnetic field on each part of the unit have been added to the dataset. Additionally, the data team has retained the original 3-axis data from each sensor, as it still carries the potential for meaningful information.

In [15]:
```
"""
Checking for any duplicated values in the dataset
"""

df_final[df_final.duplicated(keep = 'first', subset = ['timestamp','arm_temperature','ID'])]
```

Out[15]:

	timestamp	ID	activityID	ankle_3D_acc_16g_x	ankle_3D_acc_16g_y	ankle_3D_acc_16g_z	ankle_3D_gyro_x	ankle_3D_gyro_y	ankle
2806	309.0	101.0	2.0	9.804673	-0.344985	-1.476367	0.002426	-0.016852	
5439	544.0	101.0	3.0	9.839798	-0.298698	-0.582304	0.017784	-0.024801	
8277	769.0	103.0	3.0	9.848396	0.612559	0.296014	-0.106739	-0.008985	
10526	974.0	103.0	17.0	9.800833	0.183160	-1.055199	0.019436	-0.017311	
17209	1763.0	103.0	13.0	9.765130	0.512951	-0.729602	0.005153	0.016749	
18199	1903.0	103.0	13.0	9.762646	0.666634	-0.284044	-0.009567	0.007423	
6323	613.0	104.0	3.0	9.652095	1.639225	-1.406637	0.017028	0.004738	
9204	860.0	104.0	17.0	9.513097	1.131608	-2.684800	0.005386	-0.022627	
7171	679.0	105.0	3.0	9.344913	-0.614036	-3.441050	0.011126	0.014426	
9230	862.0	106.0	17.0	9.864020	-1.423190	-0.832341	0.045003	-0.013400	
8350	775.0	107.0	17.0	9.193659	0.336175	-3.967313	0.013850	0.011451	
6172	601.0	108.0	3.0	9.605916	-0.614169	-2.653719	0.002558	-0.003288	
18675	1978.0	108.0	13.0	9.311808	-0.751818	-3.591680	0.003520	0.009991	

13 rows × 34 columns

In [16]:
```
"""
Dropping the duplicated entries identified by identical values in the 'timestamp,' 'arm_temperature,' and 'ID'
"""

df_final.drop_duplicates(keep = 'first', inplace = True, subset = ['timestamp','arm_temperature','ID'])
df_final.reset_index(drop= True, inplace=True)
```

In [17]:
```
"""
Defining a function that creates a resultant value (based on x-y-z axis)
"""

def resultant(x , y, z):
    r = (x**2 + y**2 + z**2)**0.5
    return r
```

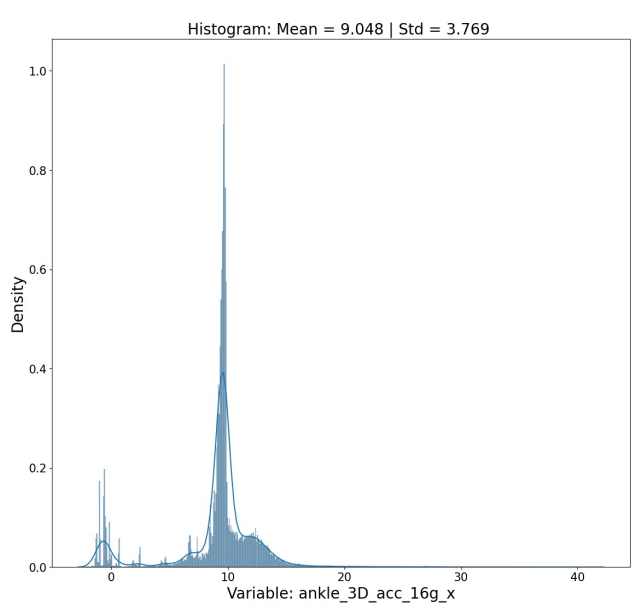
In [18]:
```
"""
Creating resultants values in the dataset (acc, gyro, magneto resultants)
"""

df_final['ankle_acc_res'] = resultant(df_final['ankle_3D_acc_16g_x'],df_final['ankle_3D_acc_16g_y'],df_final['a
df_final['ankle_gyro_res'] = resultant(df_final['ankle_3D_gyro_x'],df_final['ankle_3D_gyro_y'],df_final['ankle_
df_final['ankle_magneto_res'] = resultant(df_final['ankle_3D_magnet_x'],df_final['ankle_3D_magnet_y'],df_final[

df_final['chest_acc_res'] = resultant(df_final['chest_3D_acc_16g_x'],df_final['chest_3D_acc_16g_y'],df_final['c
df_final['chest_gyro_res'] = resultant(df_final['chest_3D_gyro_x'],df_final['chest_3D_gyro_y'],df_final['chest_
df_final['chest_magneto_res'] = resultant(df_final['chest_3D_magnet_x'],df_final['chest_3D_magnet_y'],df_final[

df_final['arm_acc_res'] = resultant(df_final['arm_3D_acc_16g_x'],df_final['arm_3D_acc_16g_y'],df_final['arm_3D_
df_final['arm_gyro_res'] = resultant(df_final['arm_3D_gyro_x'],df_final['arm_3D_gyro_y'],df_final['arm_3D_gyro_
df_final['arm_magneto_res'] = resultant(df_final['arm_3D_magnet_x'],df_final['arm_3D_magnet_y'],df_final['arm_3
```

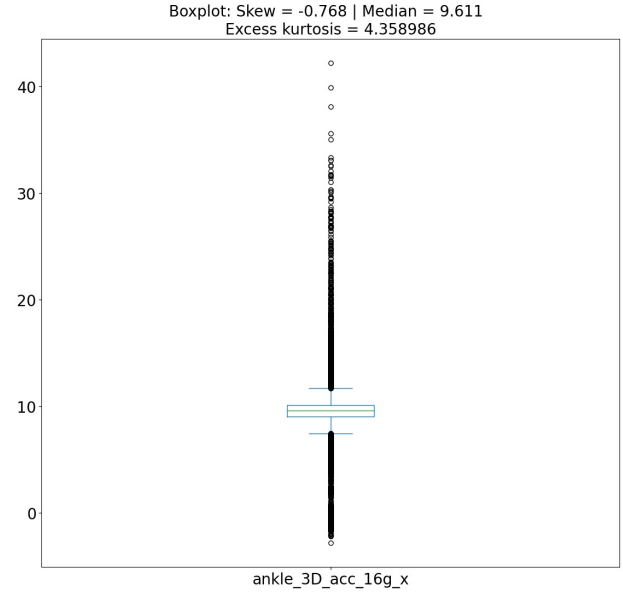
Afterwards, we visualised the features using histograms and boxplots to examine the data distribution shape and identify outliers.

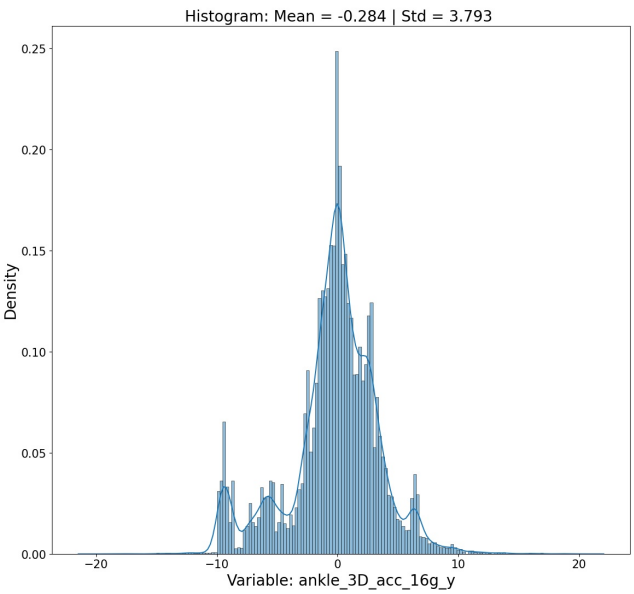
The data team has concluded that there are 18 different activities included in the dataset, and each activity has a different range of values. Consequently, although there can be some outliers in most of the features in the dataset, the team has decided not to perform outlier removal to mitigate the risk.

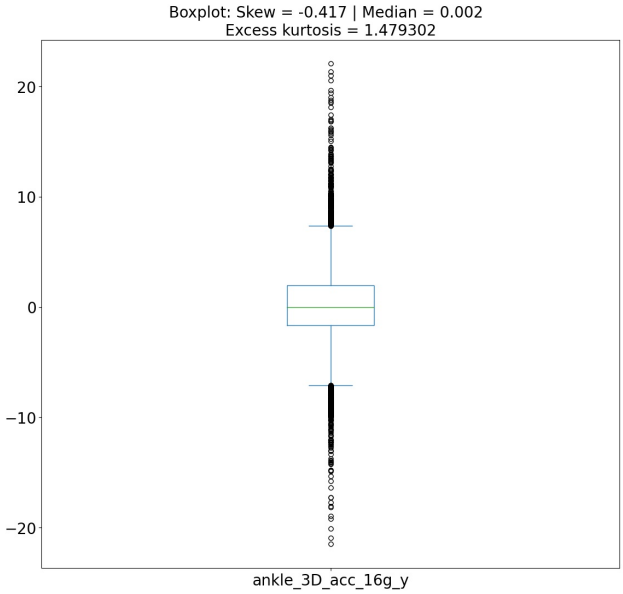
In [19]:
```
"""
Drawing histograms and boxplots (Original codes from James, revised by Jungyeon Lee)
"""

eda_df = df_final.loc[:,['ankle_3D_acc_16g_x',
       'ankle_3D_acc_16g_y', 'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x',
       'ankle_3D_gyro_y', 'ankle_3D_gyro_z', 'ankle_3D_magnet_x',
       'ankle_3D_magnet_y', 'ankle_3D_magnet_z', 'ankle_temperature',
       'arm_3D_acc_16g_x', 'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z',
       'arm_3D_gyro_x', 'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x',
       'arm_3D_magnet_y', 'arm_3D_magnet_z', 'arm_temperature', 'bpm',
       'chest_3D_acc_16g_x', 'chest_3D_acc_16g_y', 'chest_3D_acc_16g_z',
       'chest_3D_gyro_x', 'chest_3D_gyro_y', 'chest_3D_gyro_z',
       'chest_3D_magnet_x', 'chest_3D_magnet_y', 'chest_3D_magnet_z',
       'chest_temperature', 'ankle_acc_res', 'ankle_gyro_res',
       'ankle_magneto_res', 'chest_acc_res', 'chest_gyro_res',
       'chest_magneto_res', 'arm_acc_res', 'arm_gyro_res', 'arm_magneto_res']]

columns_for_graph = eda_df.columns

fig, axes = plt.subplots(len(columns_for_graph), 2, figsize = (30, 10))
```

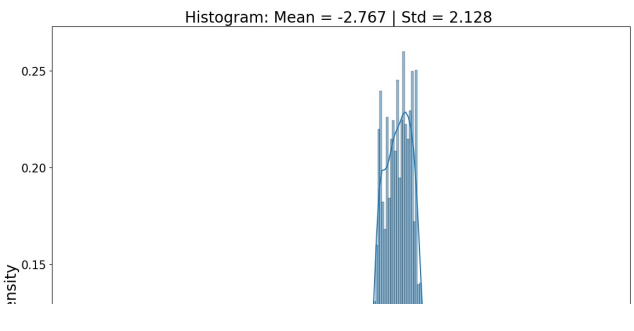
```
plt.rcParams.update({'font.size': 20})

for idx, col in enumerate(columns_for_graph):
    for idx2 in range(2):
        # Histograms
        if idx2 == 0:
            sns.histplot(data=df_final,ax=axes[idx,idx2],kde=True,x=col,stat="density")
            mean = df_final[col].mean()
            std = df_final[col].std(ddof=1)
            axes[idx,idx2].set_xlabel(f"Variable: {col}", fontsize = 20)
            axes[idx,idx2].set_ylabel("Density", fontsize = 20)
            axes[idx,idx2].set_title(f"Histogram: Mean = {mean:.3f} | Std = {std:.3f} ", fontsize = 20)
        # boxplots
        elif idx2 == 1:
            df_final[col].plot(kind="box",ax=axes[idx,idx2], fontsize = 20)
            median = df_final[col].median()
            skew = df_final[col].skew()
            kurt = df_final[col].kurtosis()
            axes[idx,idx2].set_ylabel("", fontsize = 20)
            axes[idx,idx2].set_title(f"Boxplot: Skew = {skew:.3f} | Median = {median:.3f}  \nExcess kurtosis =

fig.set_figheight(fig.get_figheight() * len(columns_for_graph)*1.5)
plt.show()
```

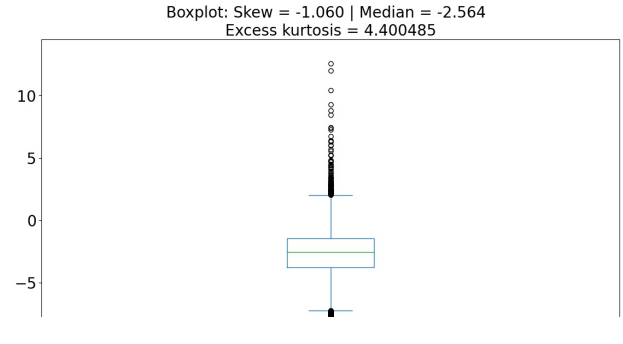


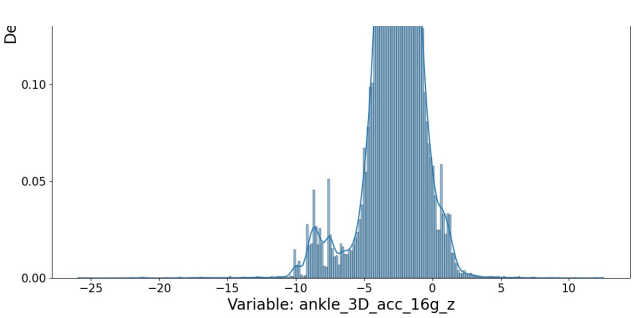


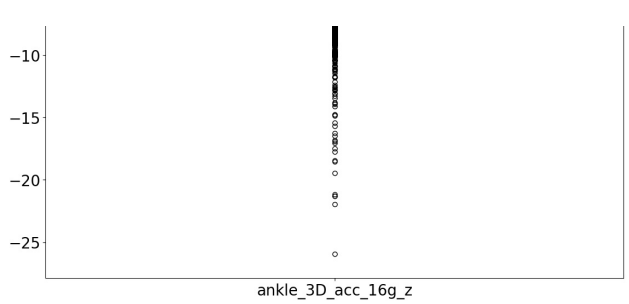


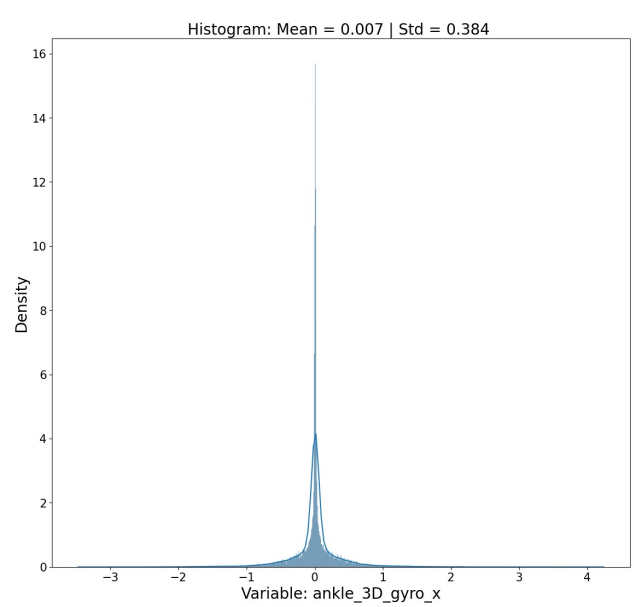


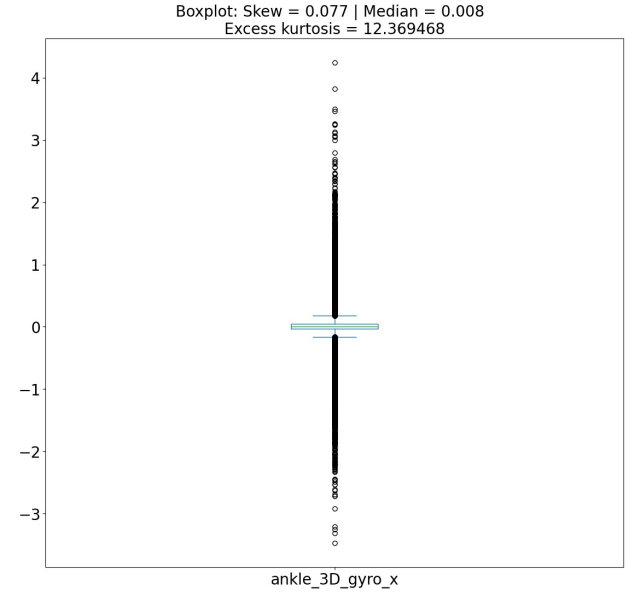


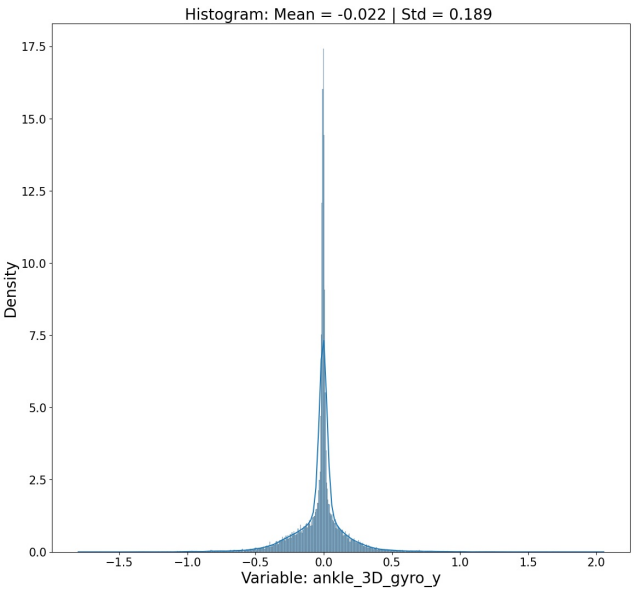


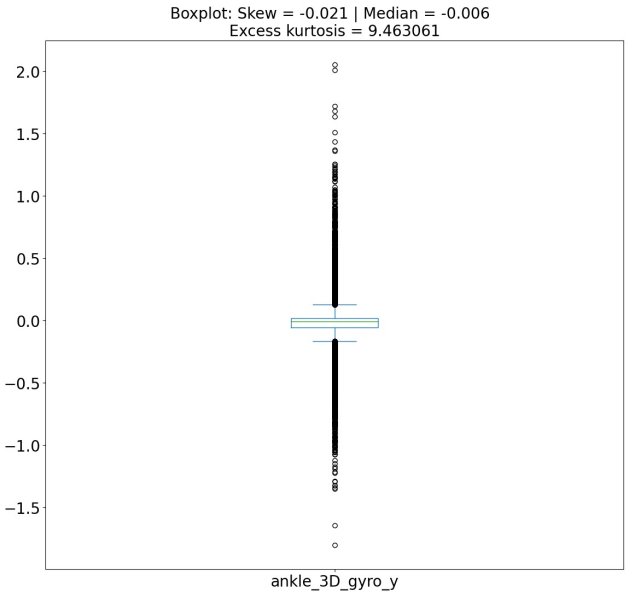

De
0.10
0.05
0.00
−25
−20
−15
−10
−5
0
5
10
Variable: ankle_3D_acc_16g_z

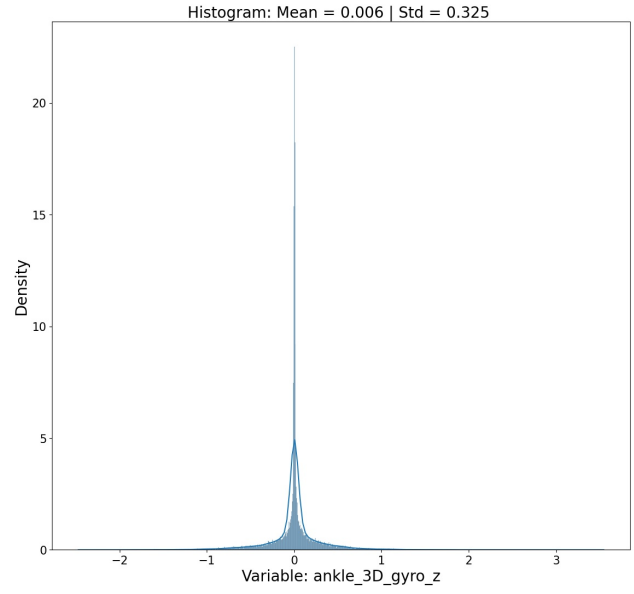

−10
−15
−20
−25
ankle_3D_acc_16g_z

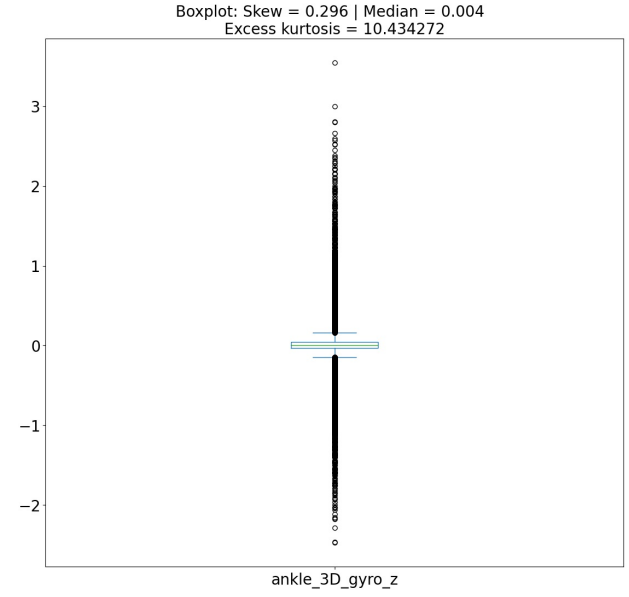

Histogram: Mean = 0.007 | Std = 0.384
Density
16
14
12
10
8
6
4
2
0
−3
−2
−1
0
1
2
3
4
Variable: ankle_3D_gyro_x

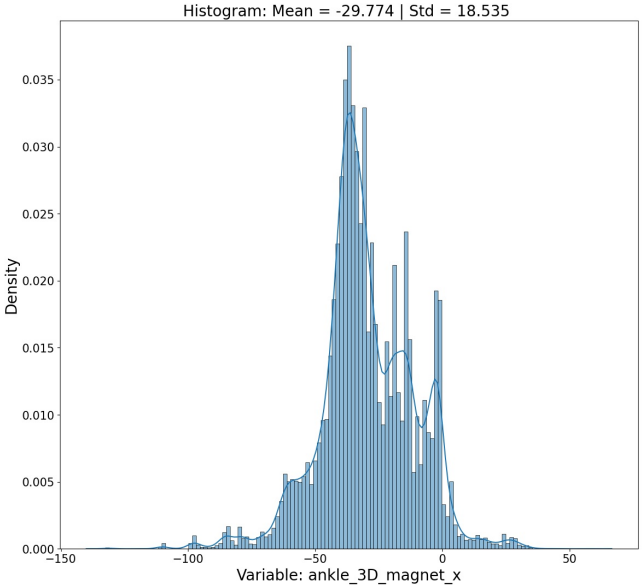

Boxplot: Skew = 0.077 | Median = 0.008
Excess kurtosis = 12.369468
4
3
2
1
0
−1
−2
−3
ankle_3D_gyro_x

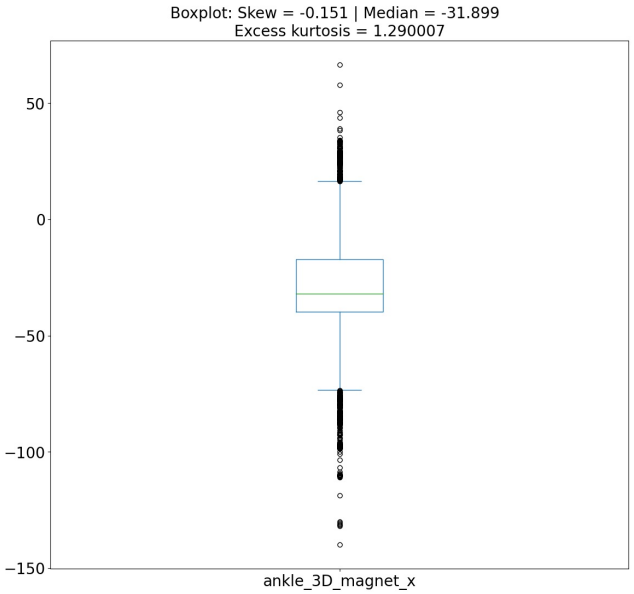

Histogram: Mean = -0.022 | Std = 0.189
Density
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Variable: ankle_3D_gyro_y

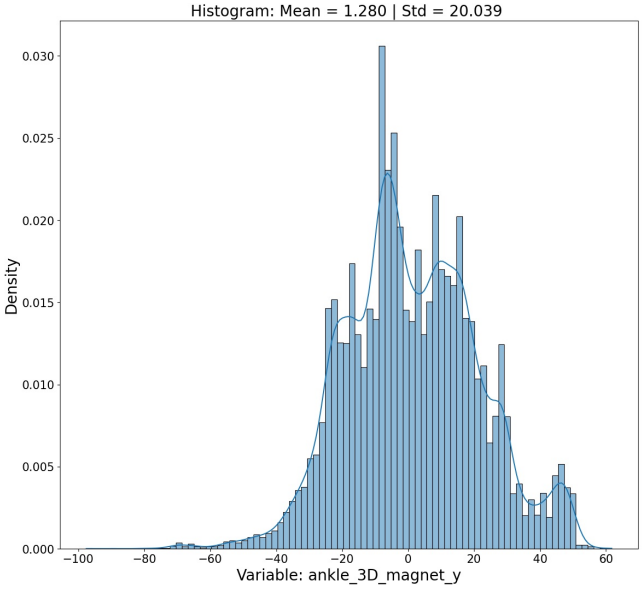

Boxplot: Skew = -0.021 | Median = -0.006
Excess kurtosis = 9.463061
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
ankle_3D_gyro_y

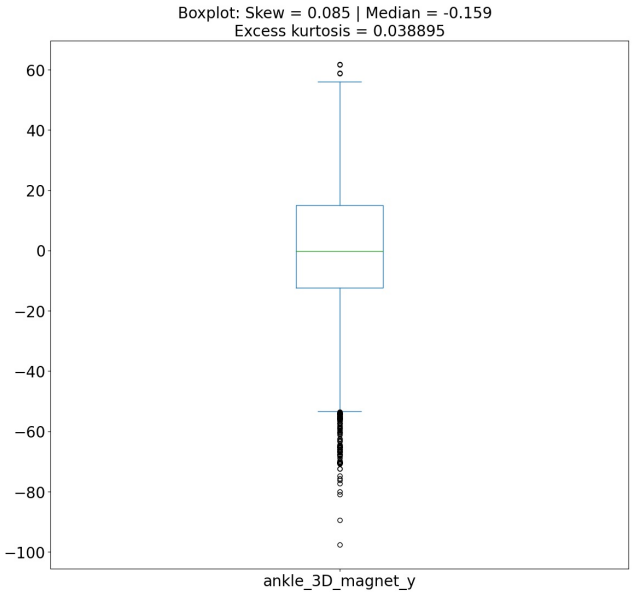

Histogram: Mean = 0.006 | Std = 0.325
Density
20
15
10
5
0
−2
−1
0
1
2
3
Variable: ankle_3D_gyro_z

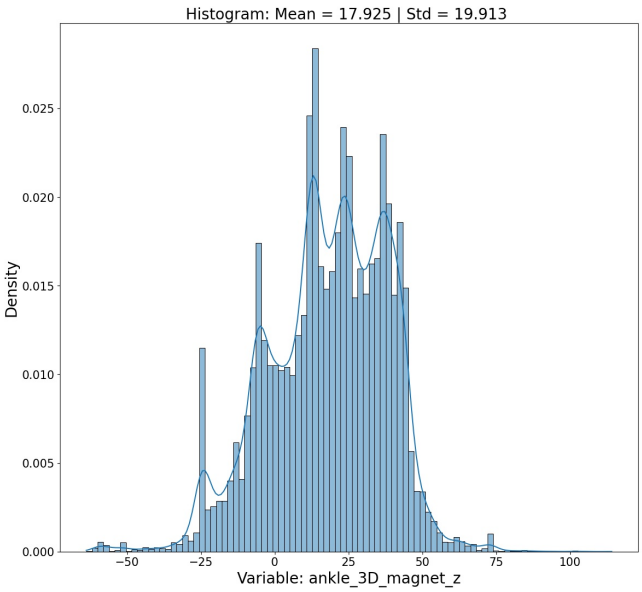

Boxplot: Skew = 0.296 | Median = 0.004
Excess kurtosis = 10.434272
3
2
1
0
−1
−2
ankle_3D_gyro_z

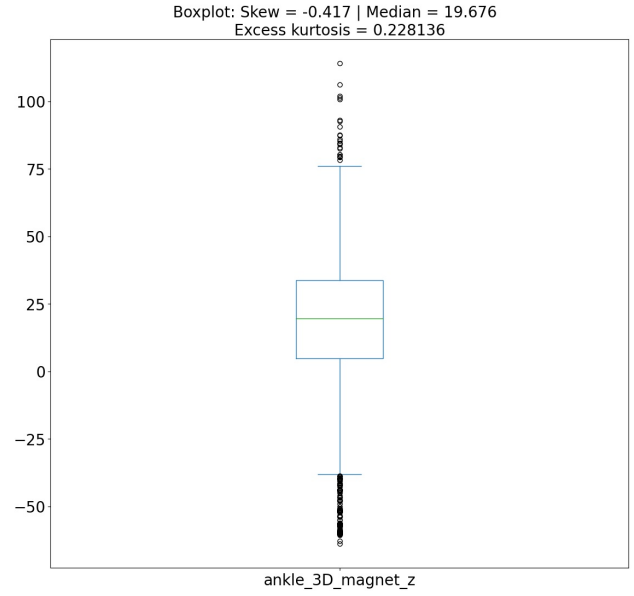

Histogram: Mean = -29.774 | Std = 18.535
Density
Variable: ankle_3D_magnet_x

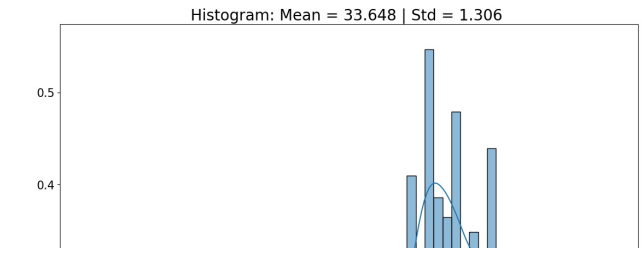

Boxplot: Skew = -0.151 | Median = -31.899
Excess kurtosis = 1.290007
ankle_3D_magnet_x

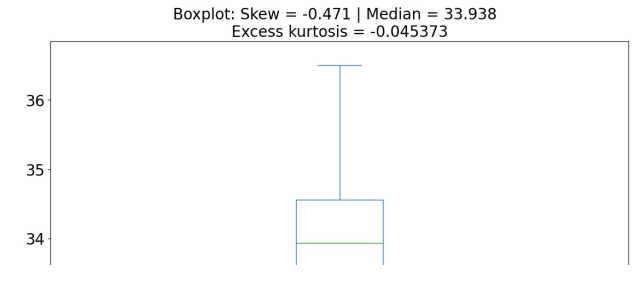

Histogram: Mean = 1.280 | Std = 20.039
Density
Variable: ankle_3D_magnet_y

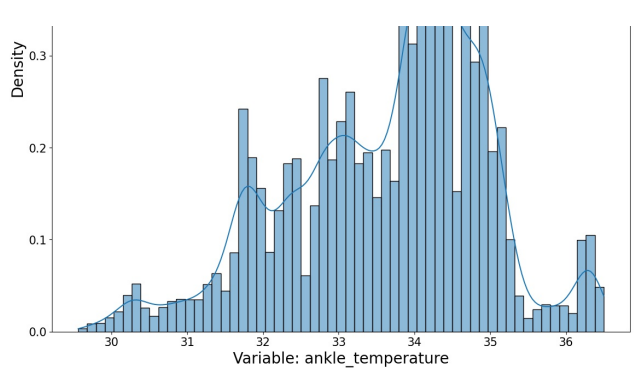

Boxplot: Skew = 0.085 | Median = -0.159
Excess kurtosis = 0.038895
ankle_3D_magnet_y

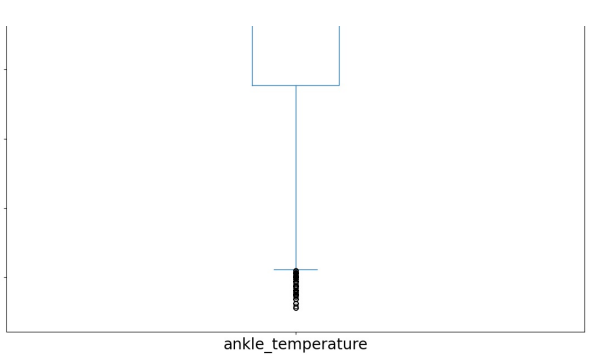

Histogram: Mean = 17.925 | Std = 19.913
Density
Variable: ankle_3D_magnet_z

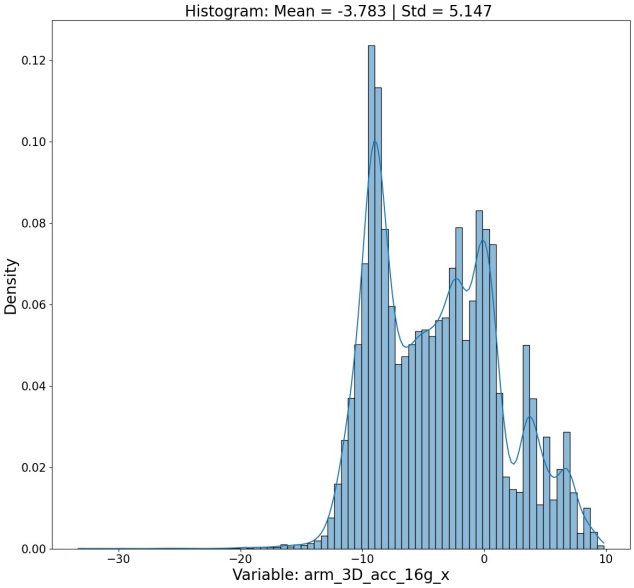

Boxplot: Skew = -0.417 | Median = 19.676
Excess kurtosis = 0.228136
ankle_3D_magnet_z

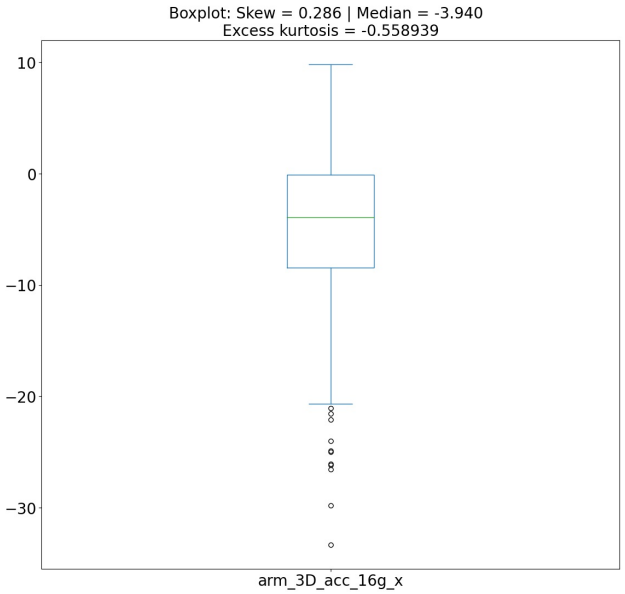

Histogram: Mean = 33.648 | Std = 1.306

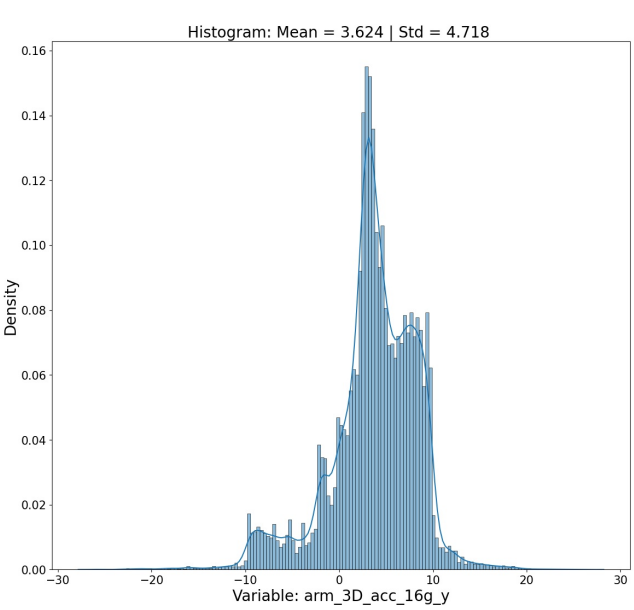

Boxplot: Skew = -0.471 | Median = 33.938
Excess kurtosis = -0.045373

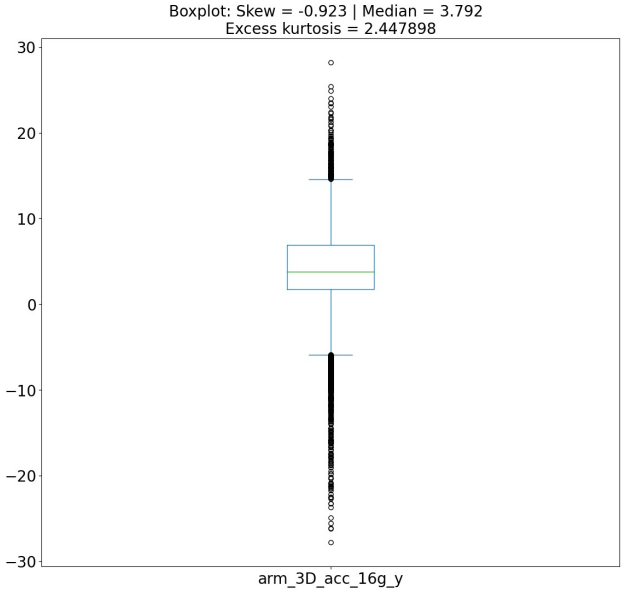

Density
Variable: ankle_temperature

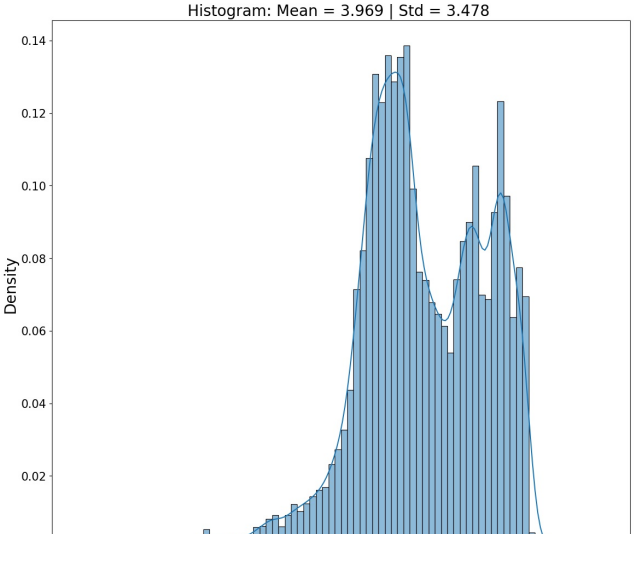

ankle_temperature

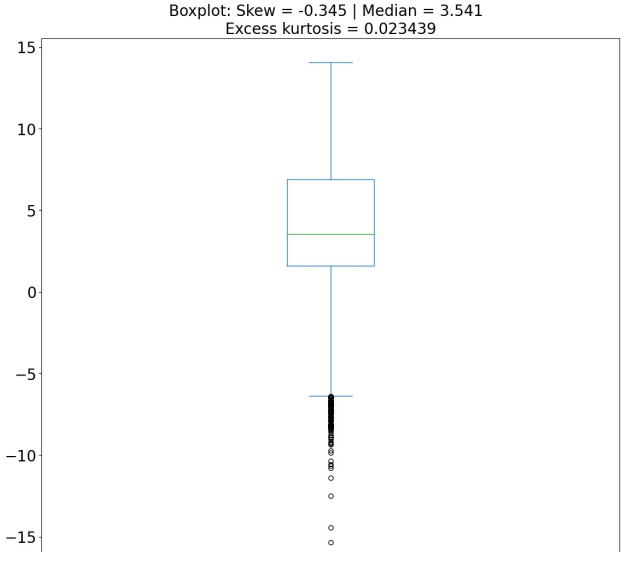

Histogram: Mean = -3.783 | Std = 5.147
Density
Variable: arm_3D_acc_16g_x

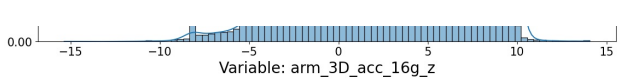

Boxplot: Skew = 0.286 | Median = -3.940
Excess kurtosis = -0.558939
arm_3D_acc_16g_x

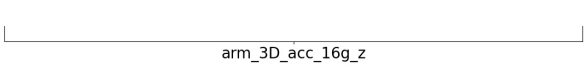

Histogram: Mean = 3.624 | Std = 4.718
Density
Variable: arm_3D_acc_16g_y

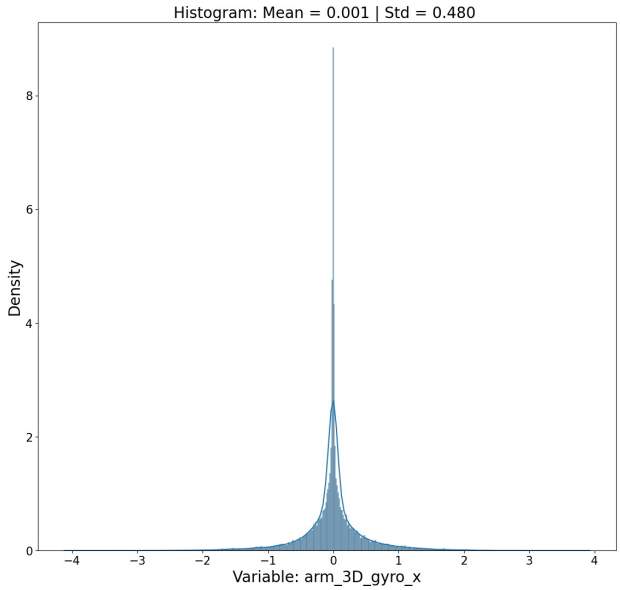

Boxplot: Skew = -0.923 | Median = 3.792
Excess kurtosis = 2.447898
arm_3D_acc_16g_y

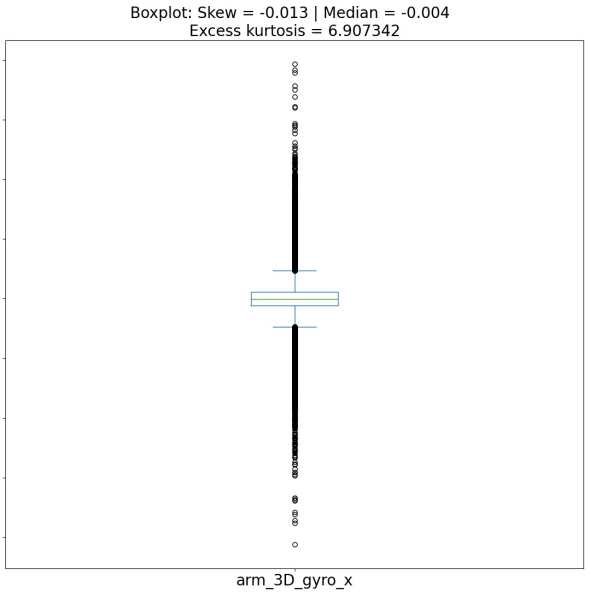

Histogram: Mean = 3.969 | Std = 3.478
Density

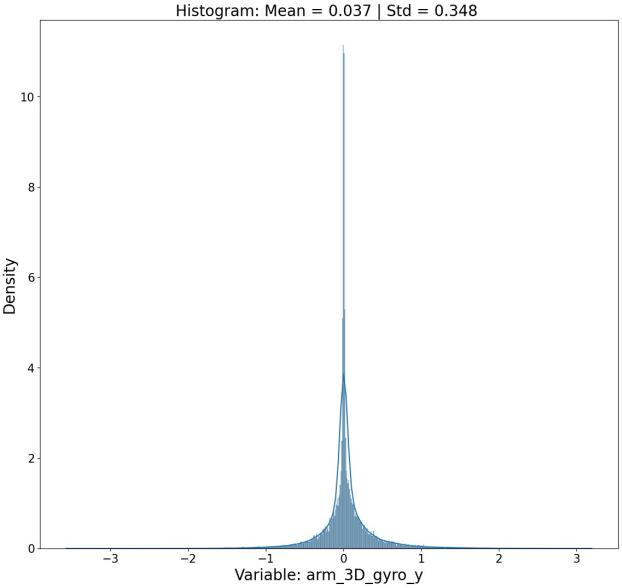

Boxplot: Skew = -0.345 | Median = 3.541
Excess kurtosis = 0.023439

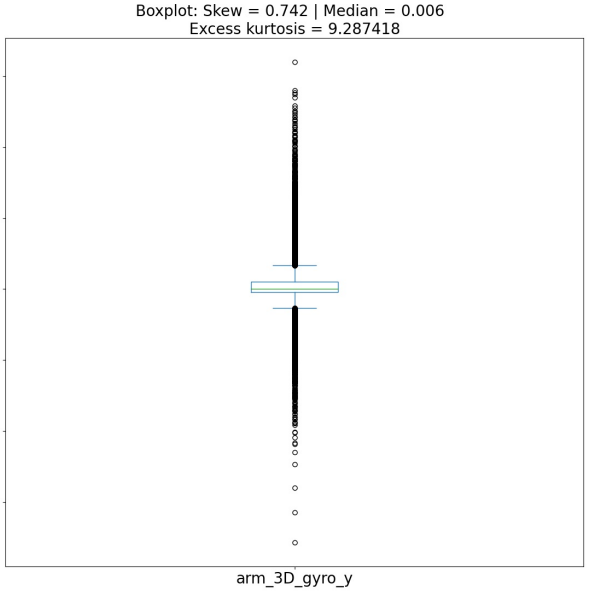

0.00
−15
−10
−5
0
5
10
15
Variable: arm_3D_acc_16g_z

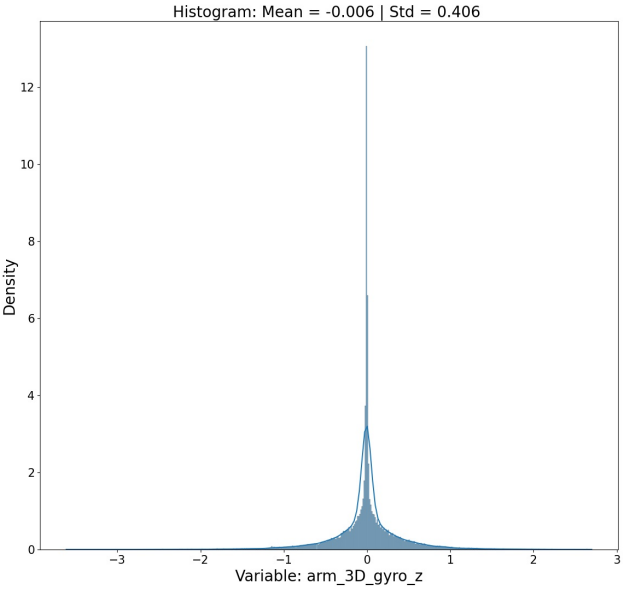

arm_3D_acc_16g_z

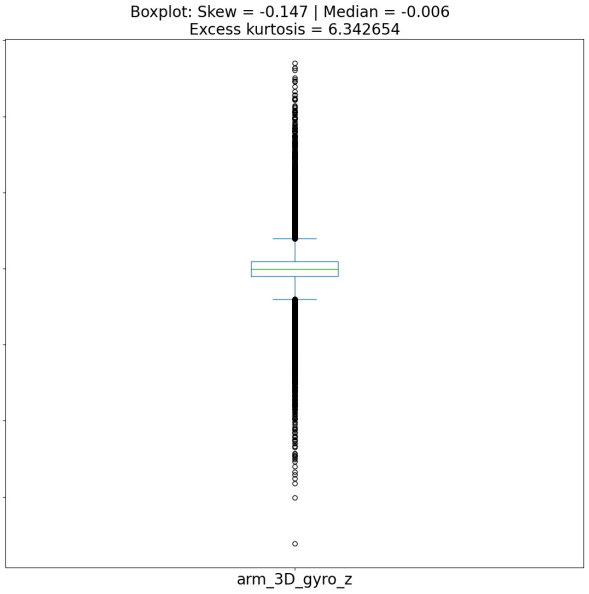

Histogram: Mean = 0.001 | Std = 0.480
Density
Variable: arm_3D_gyro_x

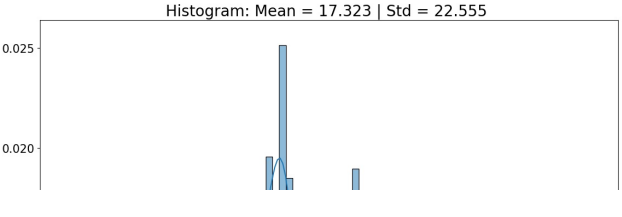

Boxplot: Skew = -0.013 | Median = -0.004
Excess kurtosis = 6.907342
arm_3D_gyro_x

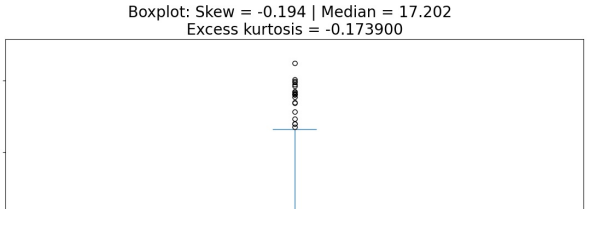

Histogram: Mean = 0.037 | Std = 0.348
Density
Variable: arm_3D_gyro_y

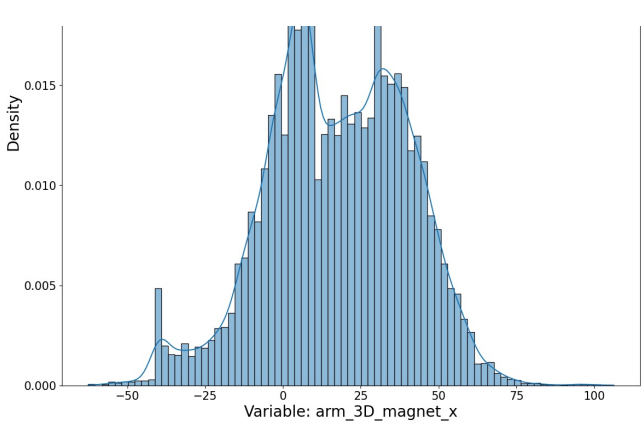

Boxplot: Skew = 0.742 | Median = 0.006
Excess kurtosis = 9.287418
arm_3D_gyro_y

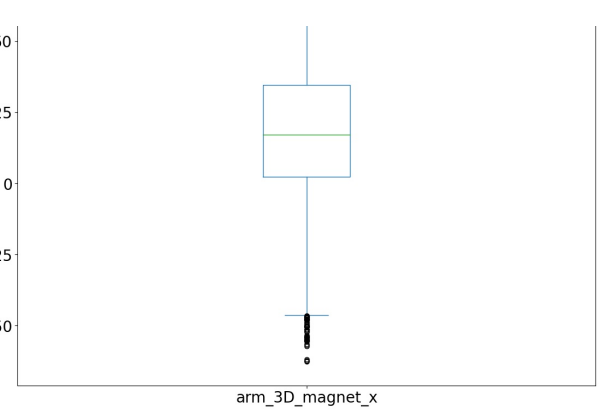

Histogram: Mean = -0.006 | Std = 0.406
Density
Variable: arm_3D_gyro_z


Boxplot: Skew = -0.147 | Median = -0.006
Excess kurtosis = 6.342654
arm_3D_gyro_z

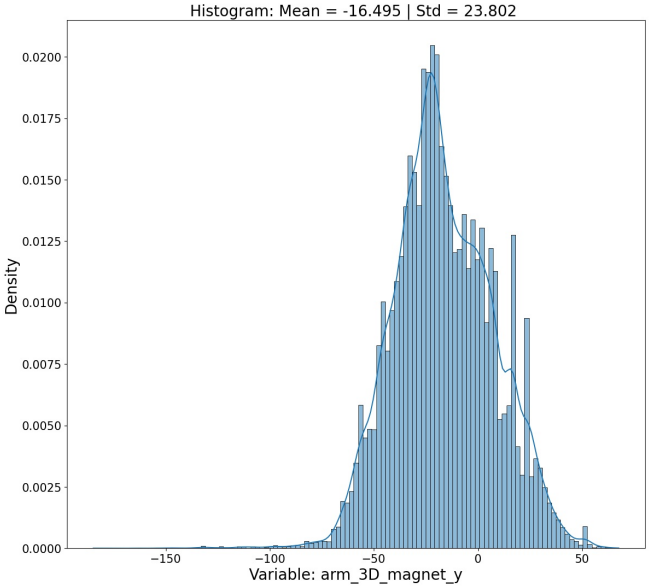

Histogram: Mean = 17.323 | Std = 22.555


Boxplot: Skew = -0.194 | Median = 17.202
Excess kurtosis = -0.173900

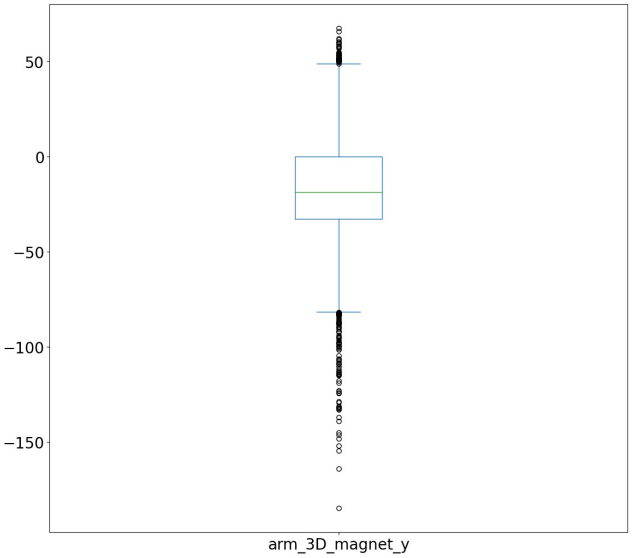




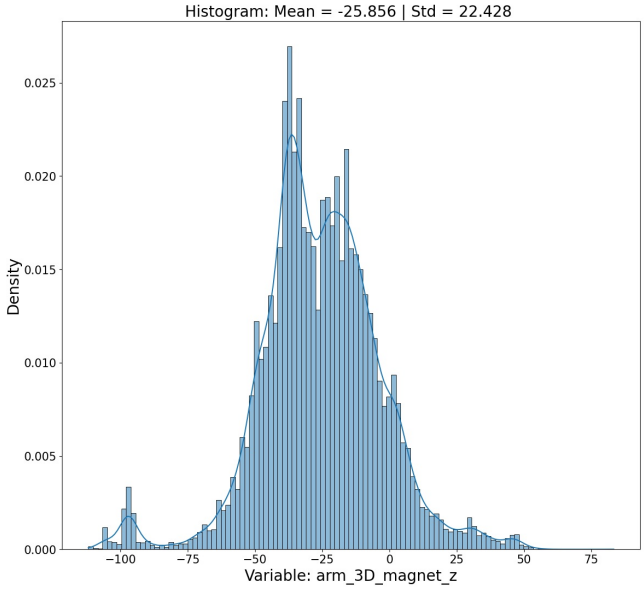
Histogram: Mean = -16.495 | Std = 23.802



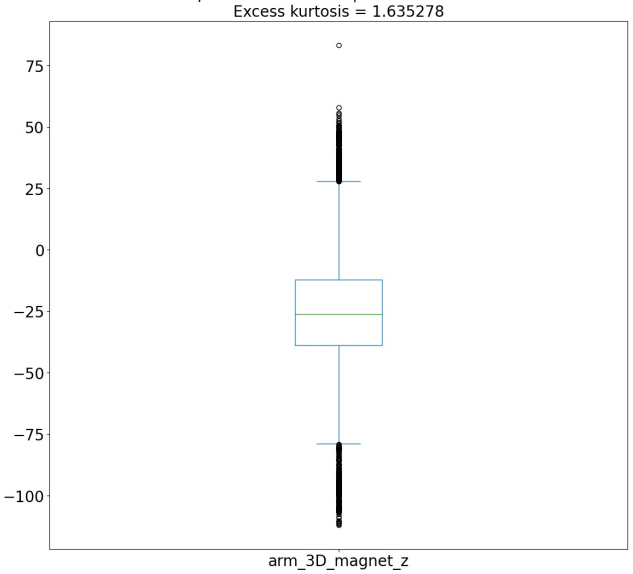
Boxplot: Skew = 0.014 | Median = -18.676
Excess kurtosis = 0.439202



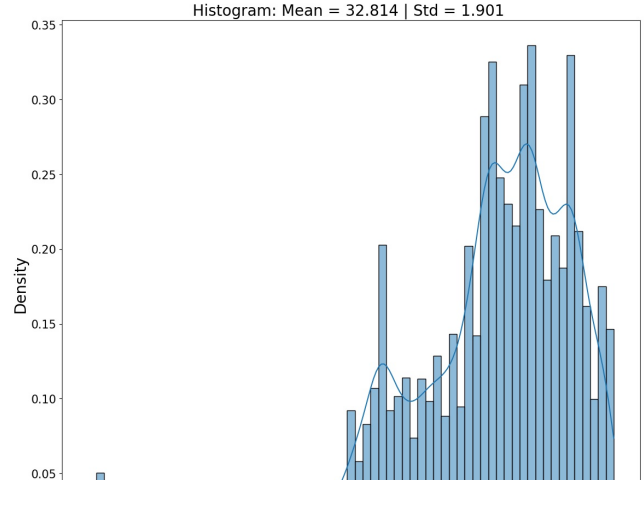
Histogram: Mean = -25.856 | Std = 22.428



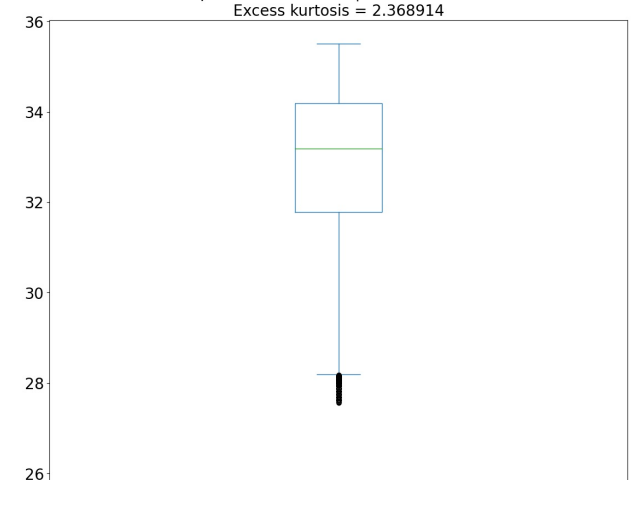
Boxplot: Skew = -0.175 | Median = -26.209
Excess kurtosis = 1.635278

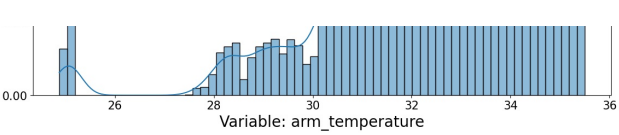


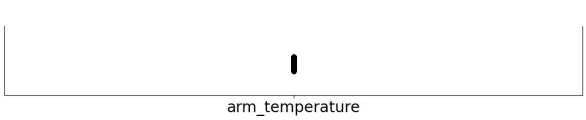
Histogram: Mean = 32.814 | Std = 1.901

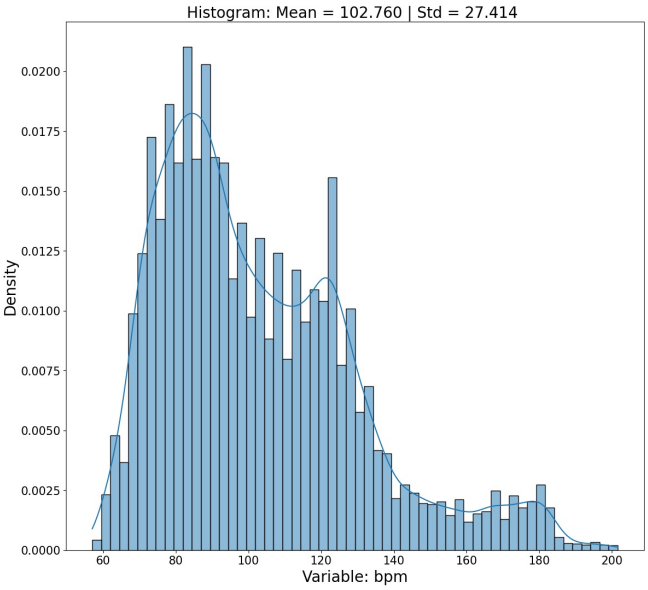


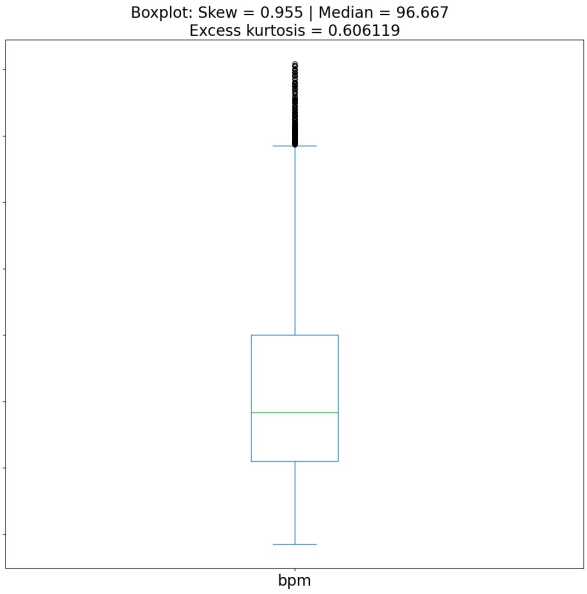
Boxplot: Skew = -1.297 | Median = 33.188
Excess kurtosis = 2.368914

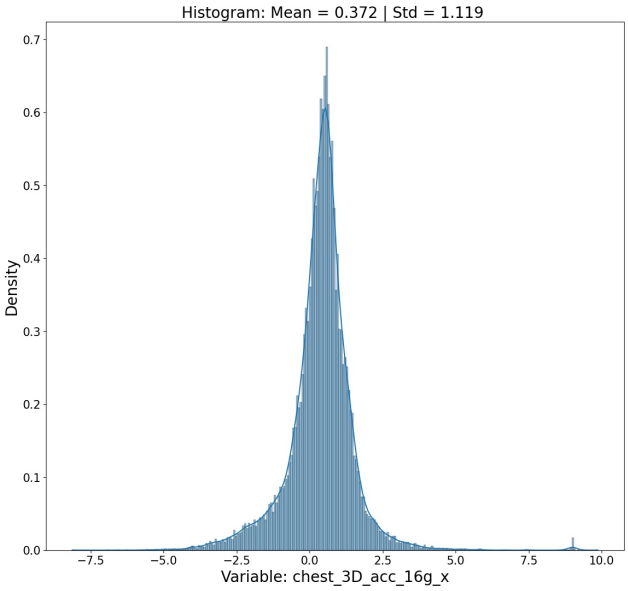


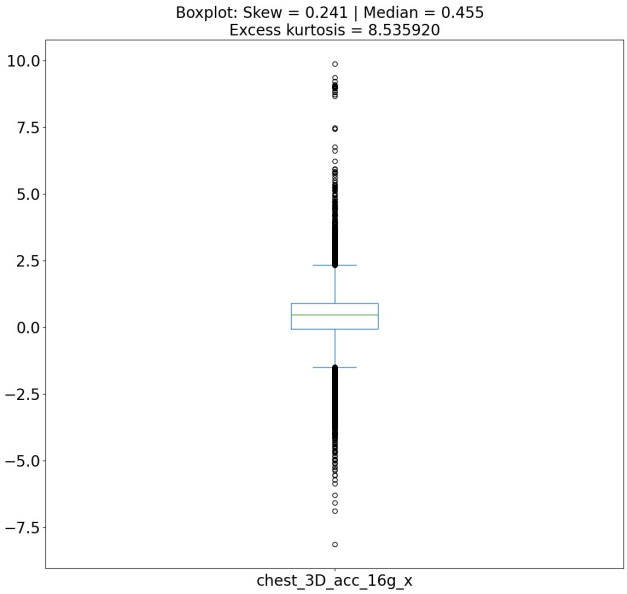

0.00
26
28
30
32
34
36
Variable: arm_temperature

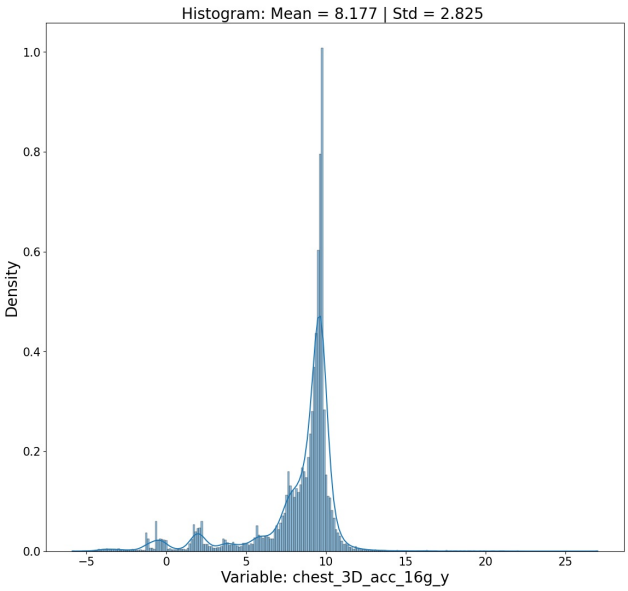

arm_temperature

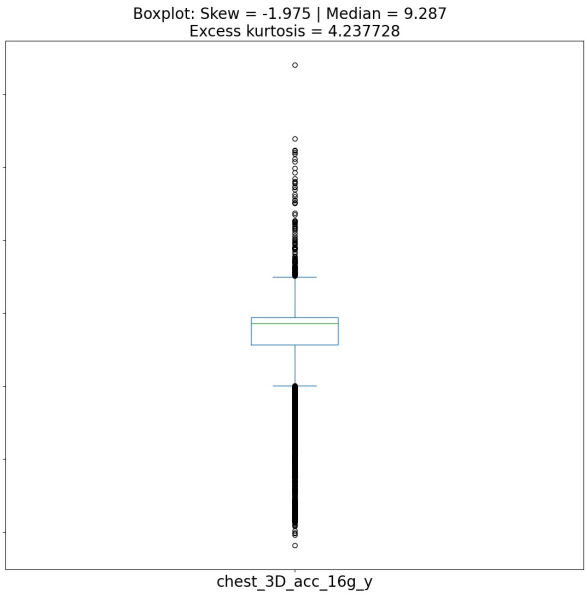

Histogram: Mean = 102.760 | Std = 27.414
Density
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
60
80
100
120
140
160
180
200
Variable: bpm

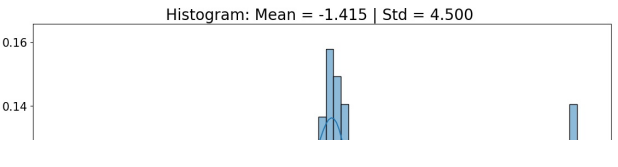

Boxplot: Skew = 0.955 | Median = 96.667
Excess kurtosis = 0.606119
200
180
160
140
120
100
80
60
bpm

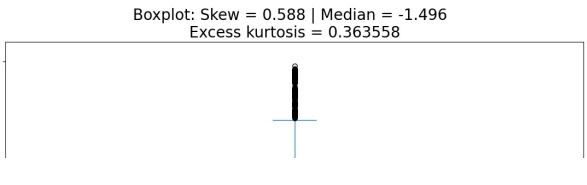

Histogram: Mean = 0.372 | Std = 1.119
Density
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Variable: chest_3D_acc_16g_x

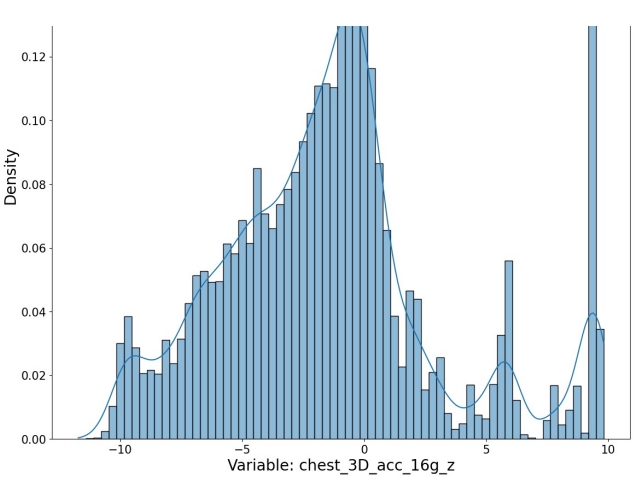

Boxplot: Skew = 0.241 | Median = 0.455
Excess kurtosis = 8.535920
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
chest_3D_acc_16g_x

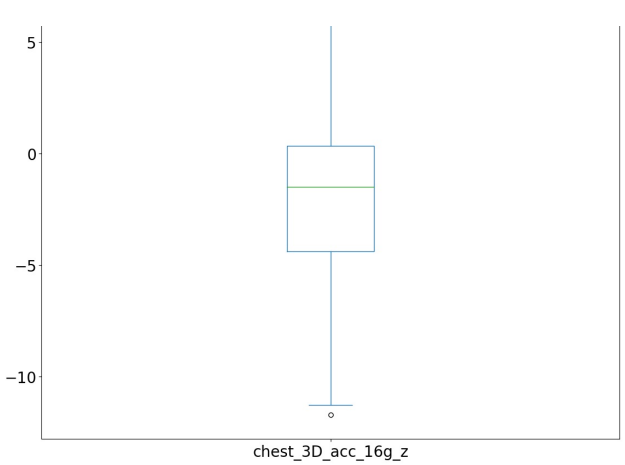

Histogram: Mean = 8.177 | Std = 2.825
Density
1.0
0.8
0.6
0.4
0.2
0.0
−5
0
5
10
15
20
25
Variable: chest_3D_acc_16g_y

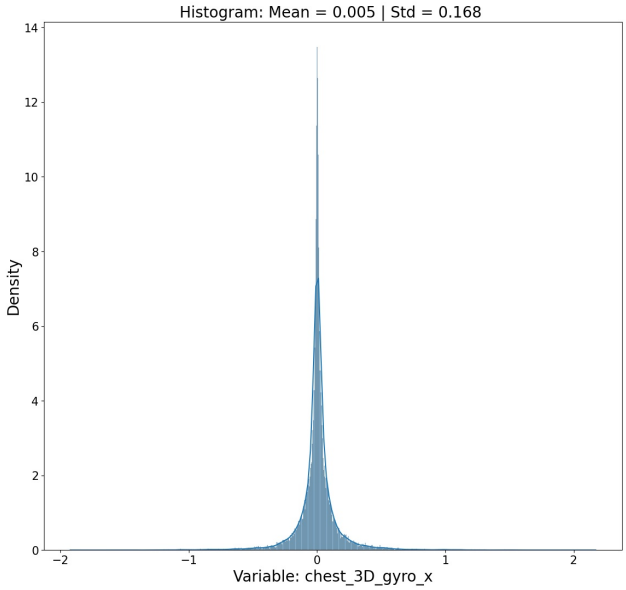

Boxplot: Skew = -1.975 | Median = 9.287
Excess kurtosis = 4.237728
25
20
15
10
5
0
−5
chest_3D_acc_16g_y

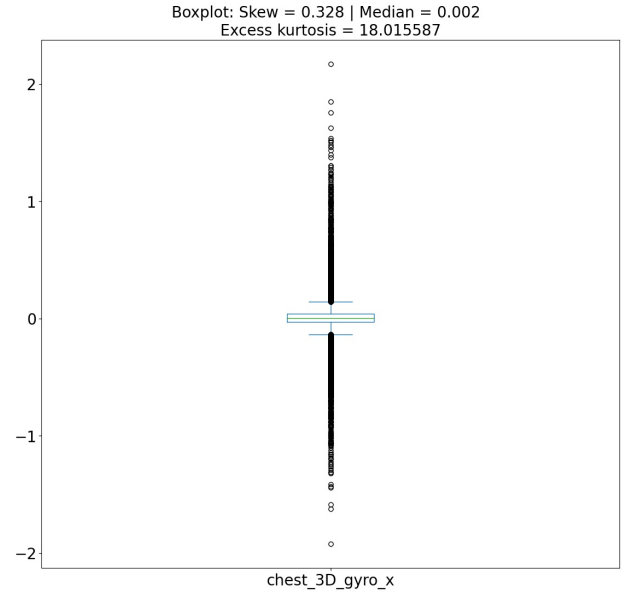

Histogram: Mean = -1.415 | Std = 4.500
0.16
0.14

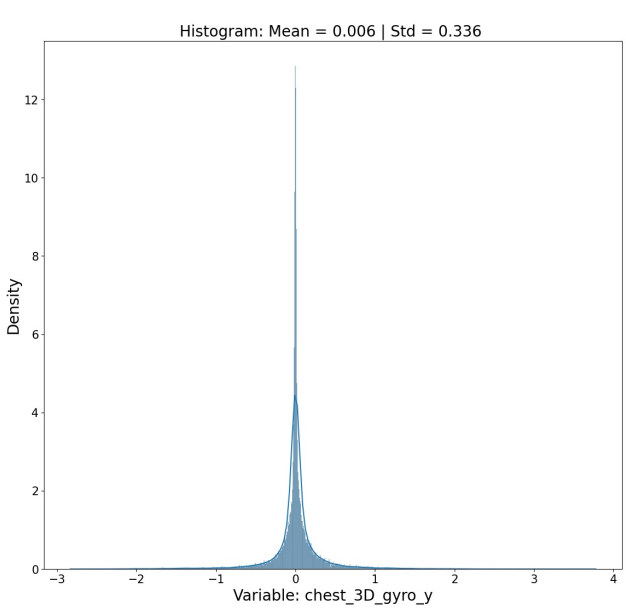

Boxplot: Skew = 0.588 | Median = -1.496
Excess kurtosis = 0.363558
10

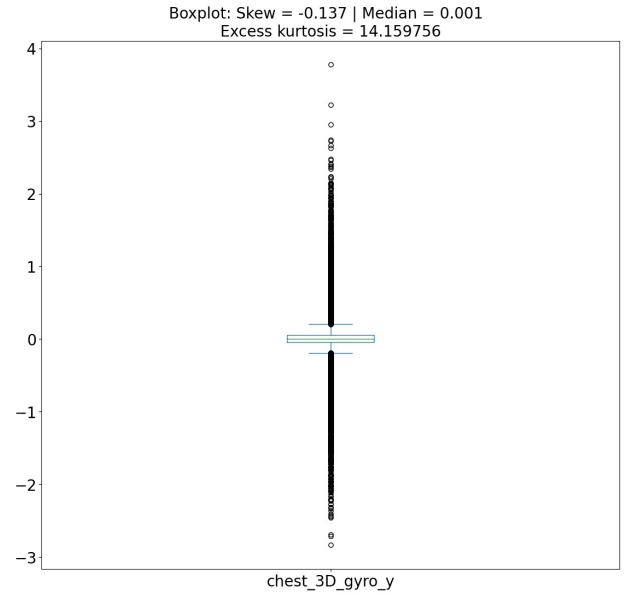

Density
Variable: chest_3D_acc_16g_z

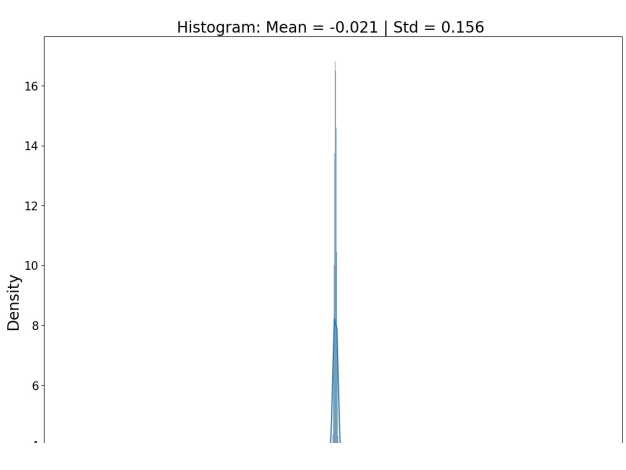

chest_3D_acc_16g_z

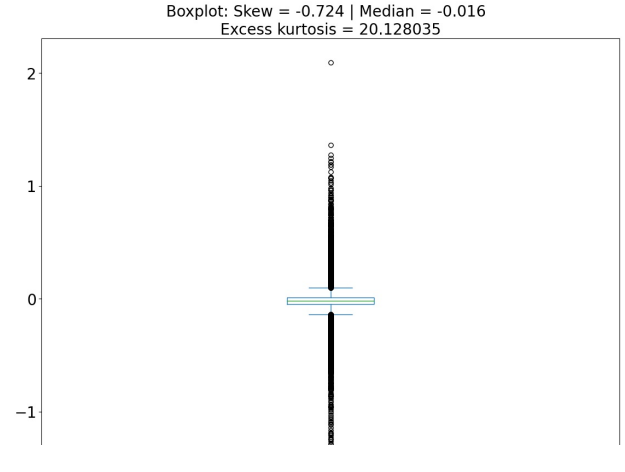

Histogram: Mean = 0.005 | Std = 0.168
Density
Variable: chest_3D_gyro_x

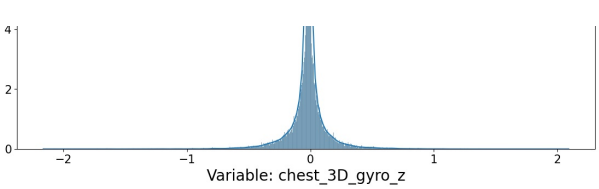

Boxplot: Skew = 0.328 | Median = 0.002
Excess kurtosis = 18.015587
chest_3D_gyro_x


Histogram: Mean = 0.006 | Std = 0.336
Density
Variable: chest_3D_gyro_y

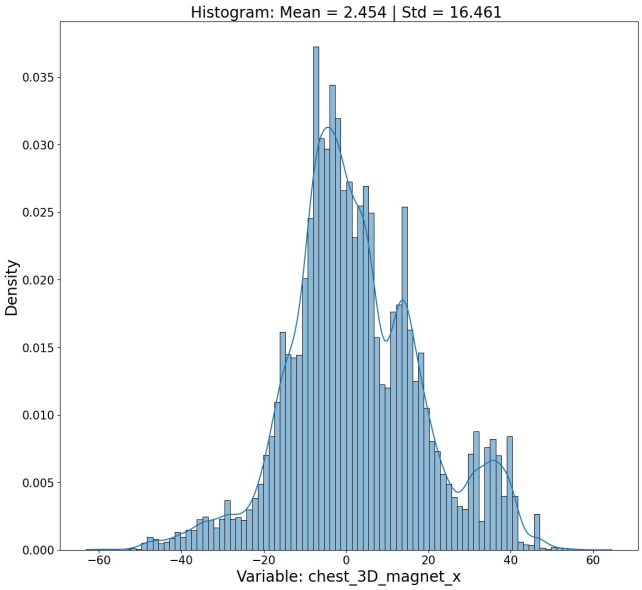

Boxplot: Skew = -0.137 | Median = 0.001
Excess kurtosis = 14.159756
chest_3D_gyro_y

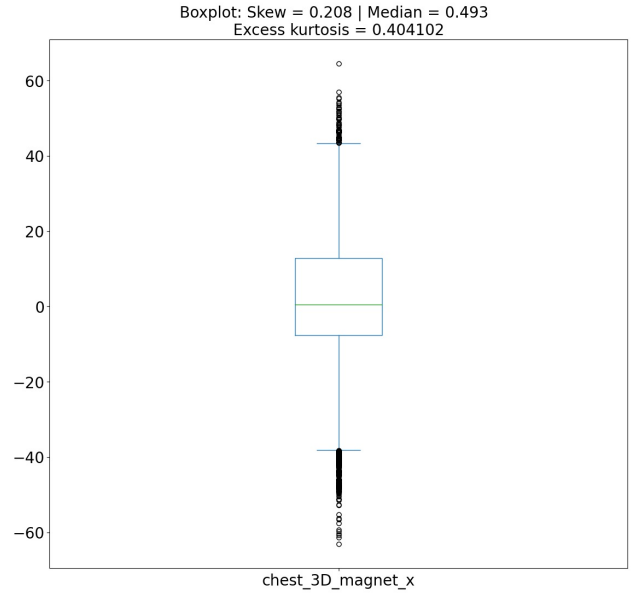

Histogram: Mean = -0.021 | Std = 0.156
Density

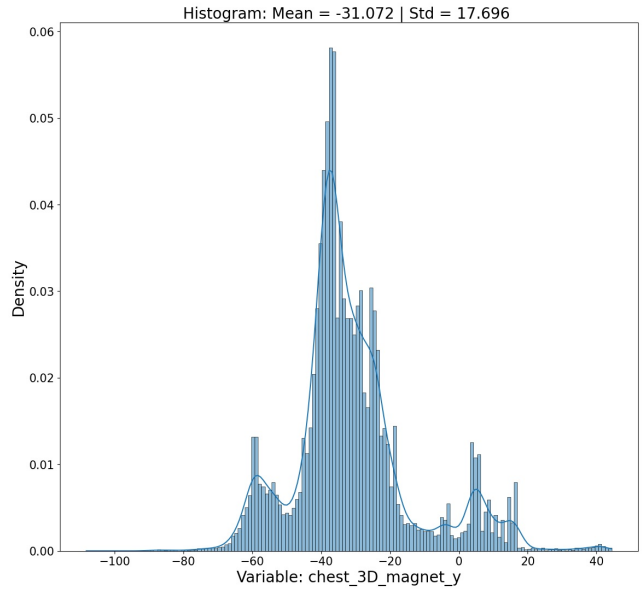

Boxplot: Skew = -0.724 | Median = -0.016
Excess kurtosis = 20.128035

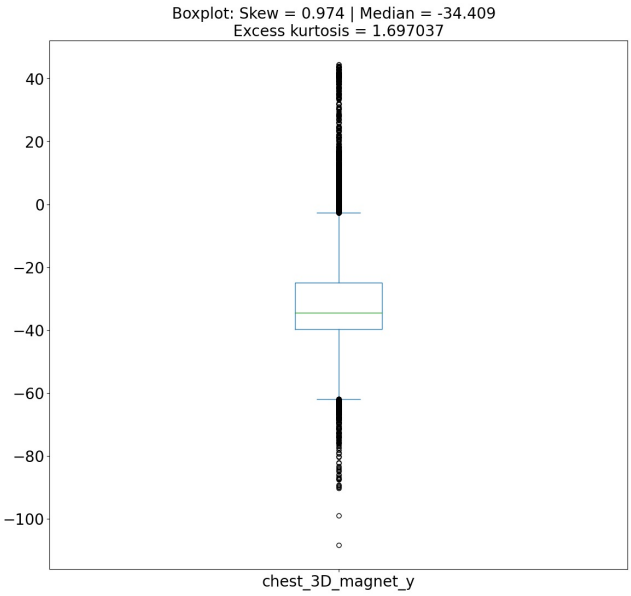

Variable: chest_3D_gyro_z

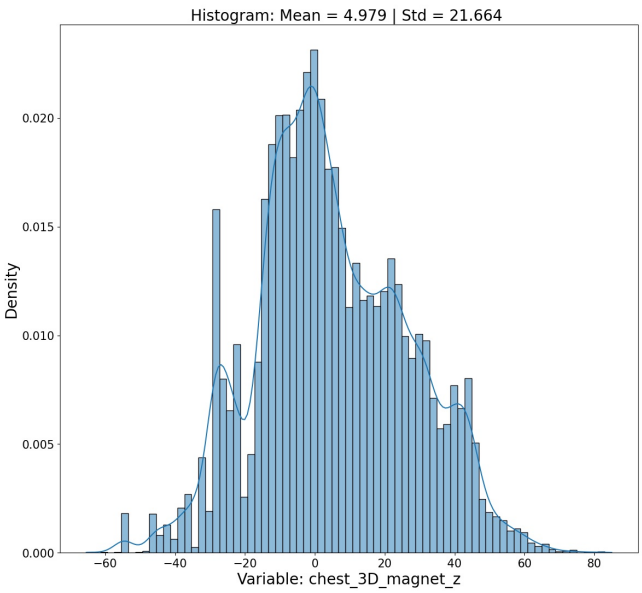

chest_3D_gyro_z

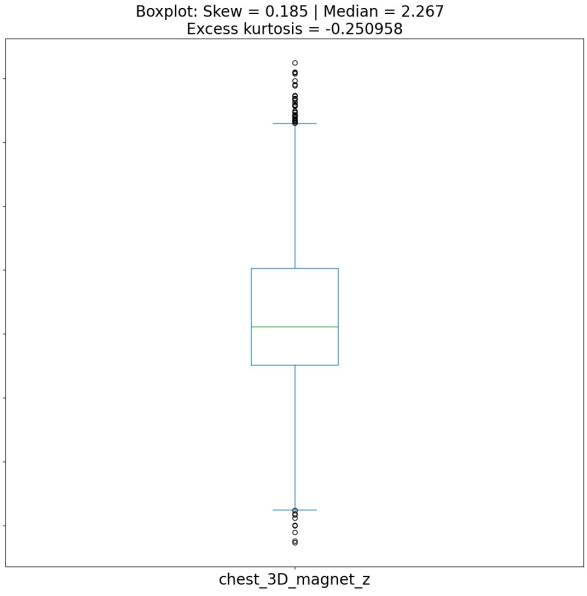

Histogram: Mean = 2.454 | Std = 16.461
Density
Variable: chest_3D_magnet_x

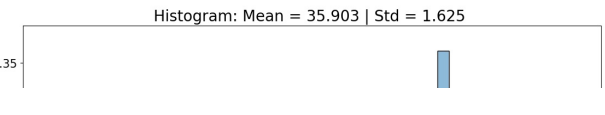

Boxplot: Skew = 0.208 | Median = 0.493
Excess kurtosis = 0.404102
chest_3D_magnet_x

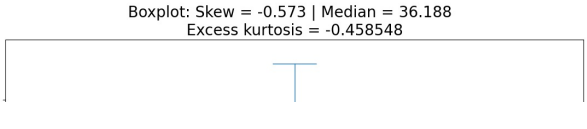

Histogram: Mean = -31.072 | Std = 17.696
Density
Variable: chest_3D_magnet_y

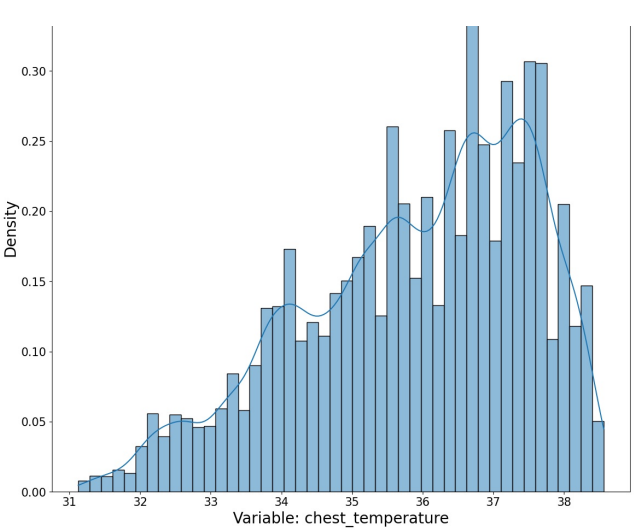

Boxplot: Skew = 0.974 | Median = -34.409
Excess kurtosis = 1.697037
chest_3D_magnet_y

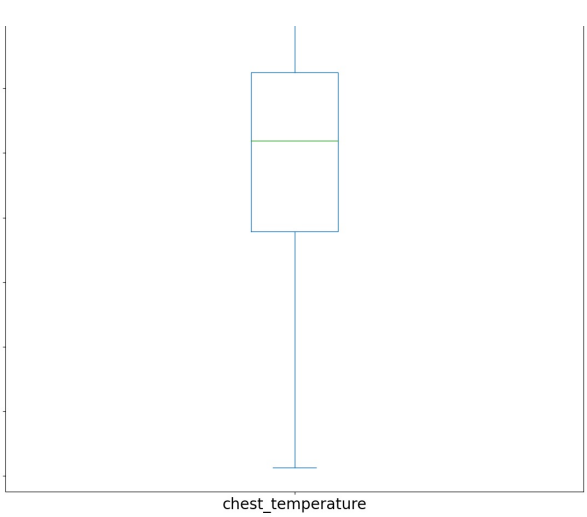

Histogram: Mean = 4.979 | Std = 21.664
Density
Variable: chest_3D_magnet_z

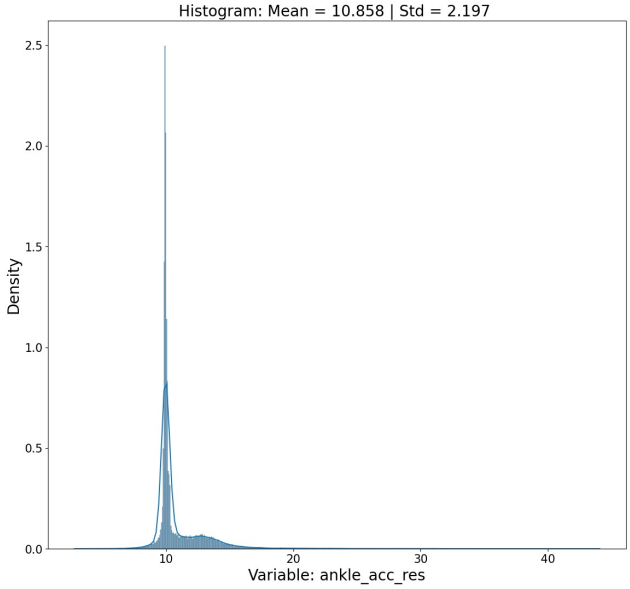

Boxplot: Skew = 0.185 | Median = 2.267
Excess kurtosis = -0.250958
chest_3D_magnet_z

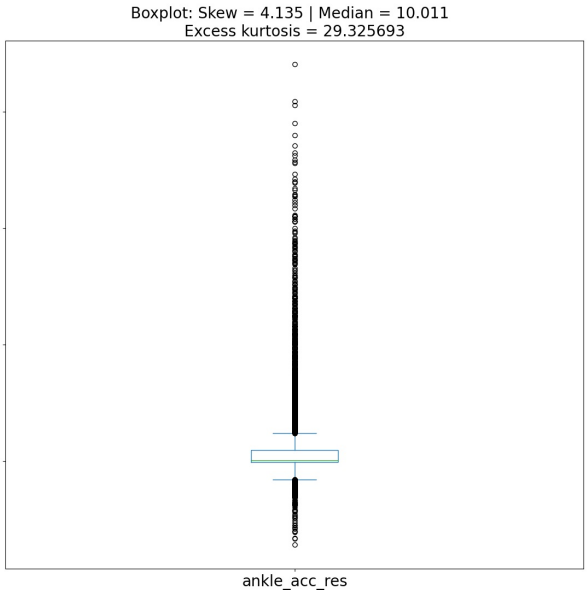

Histogram: Mean = 35.903 | Std = 1.625

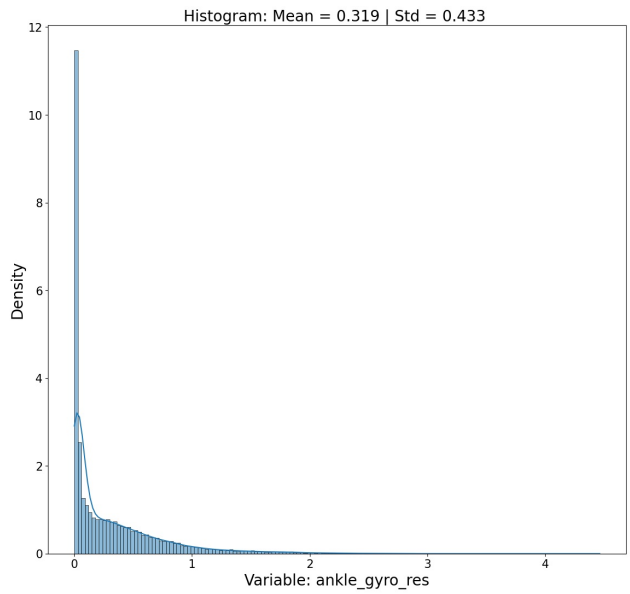

Boxplot: Skew = -0.573 | Median = 36.188
Excess kurtosis = -0.458548

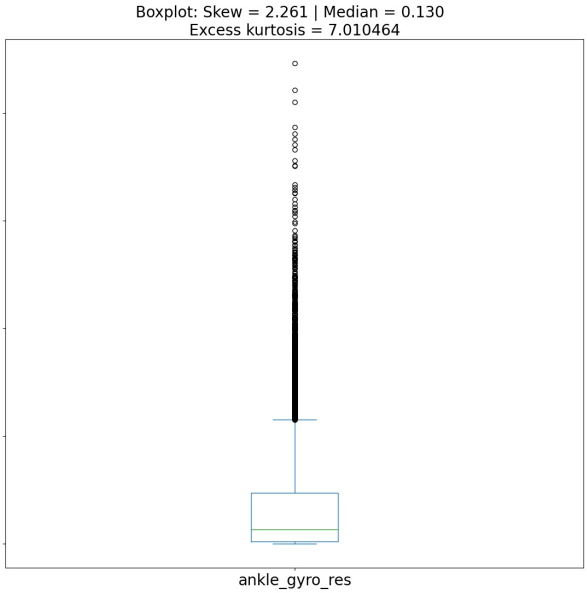

Density
Variable: chest_temperature

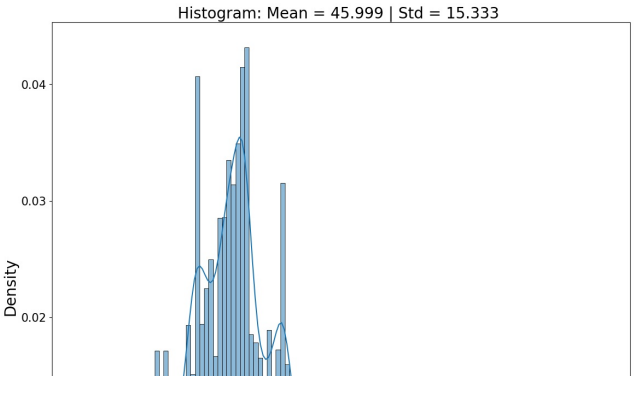

chest_temperature

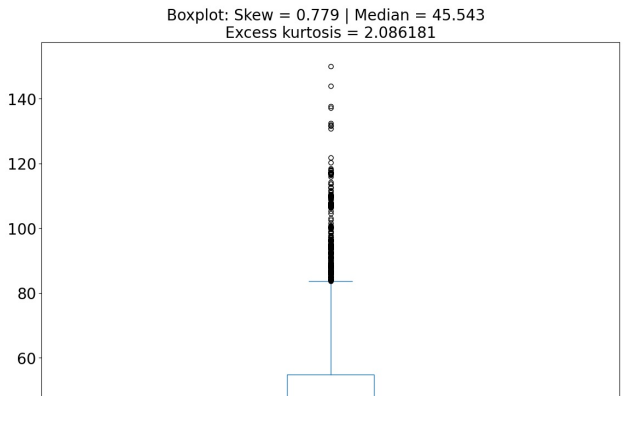

Histogram: Mean = 10.858 | Std = 2.197
Density
Variable: ankle_acc_res

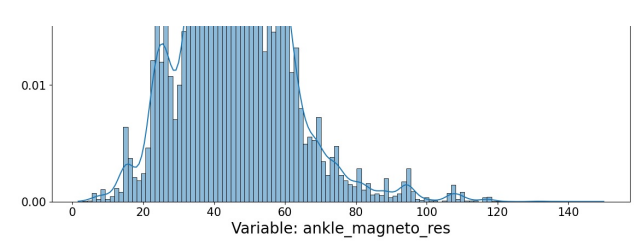

Boxplot: Skew = 4.135 | Median = 10.011
Excess kurtosis = 29.325693
ankle_acc_res

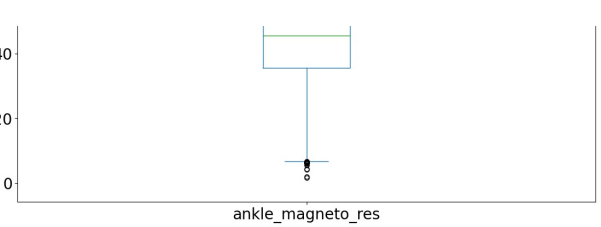

Histogram: Mean = 0.319 | Std = 0.433
Density
Variable: ankle_gyro_res

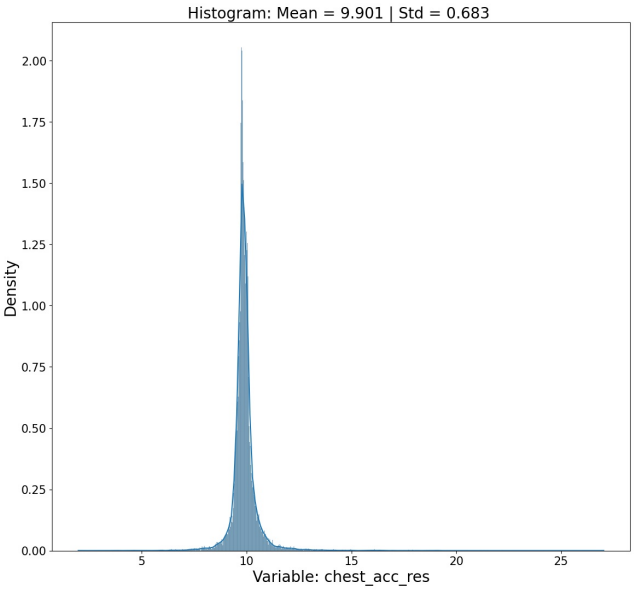

Boxplot: Skew = 2.261 | Median = 0.130
Excess kurtosis = 7.010464
ankle_gyro_res

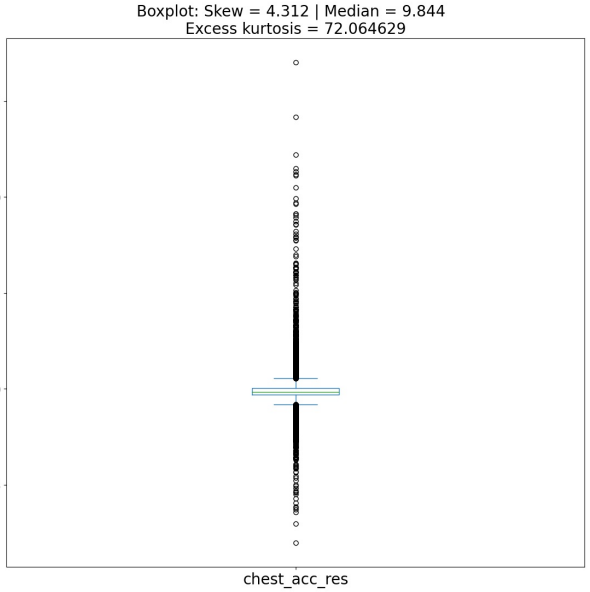

Histogram: Mean = 45.999 | Std = 15.333
Density

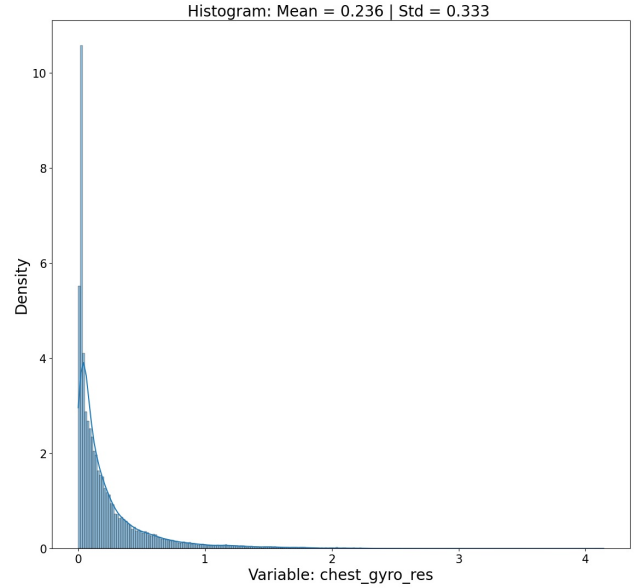

Boxplot: Skew = 0.779 | Median = 45.543
Excess kurtosis = 2.086181

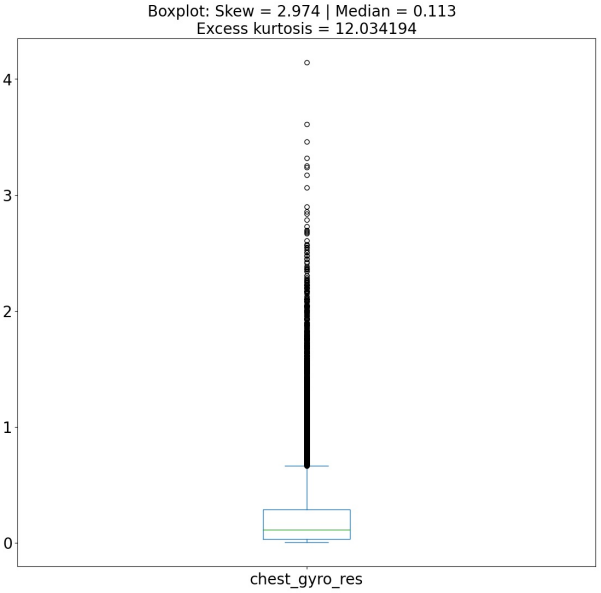

0.01
0.00
0
20
40
60
80
100
120
140
Variable: ankle_magneto_res

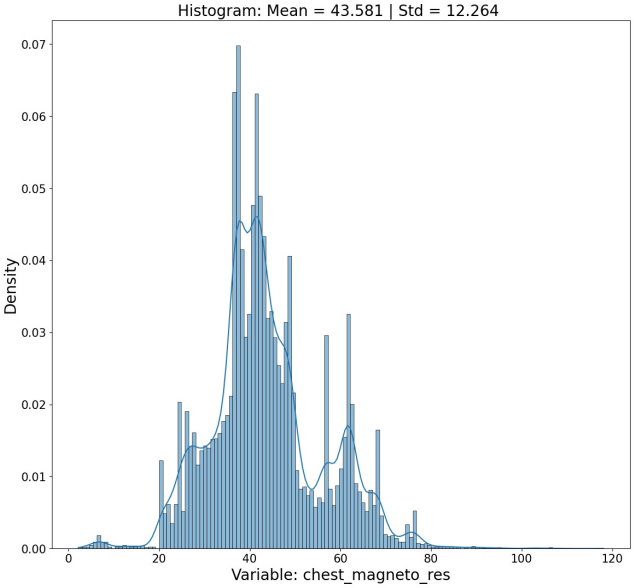

40
20
0
ankle_magneto_res

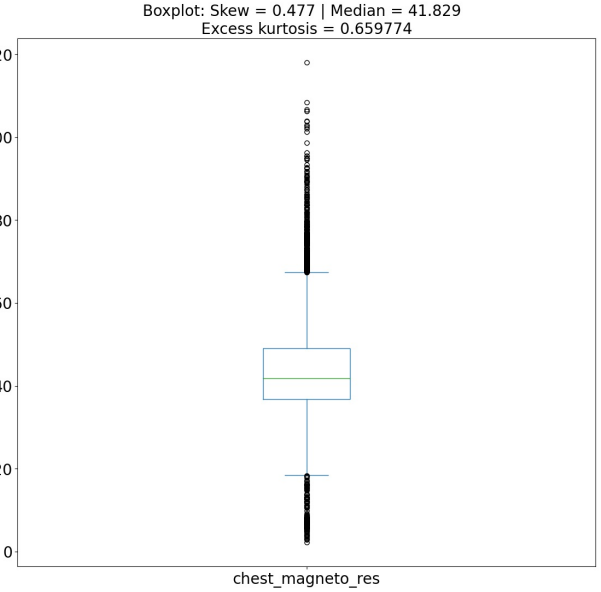

Histogram: Mean = 9.901 | Std = 0.683
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
5
10
15
20
25
Variable: chest_acc_res

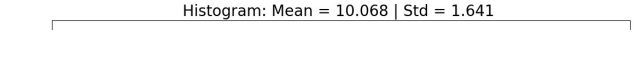

Boxplot: Skew = 4.312 | Median = 9.844
Excess kurtosis = 72.064629
25
20
15
10
5
chest_acc_res

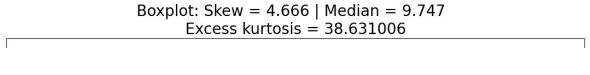

Histogram: Mean = 0.236 | Std = 0.333
10
8
6
4
2
0
Density
0
1
2
3
4
Variable: chest_gyro_res

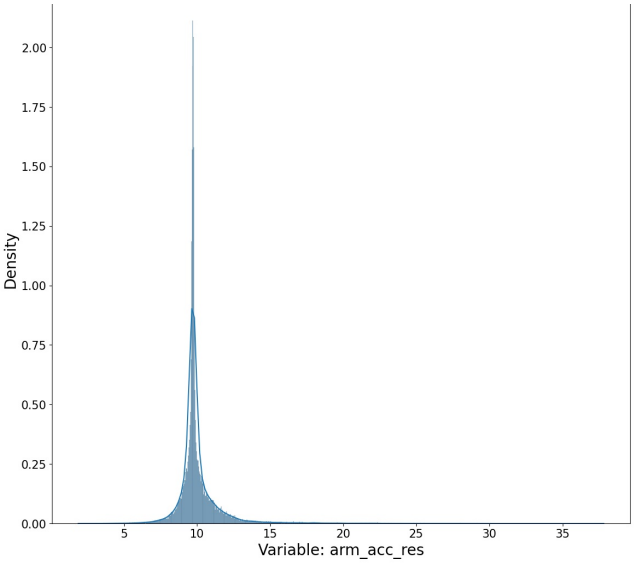

Boxplot: Skew = 2.974 | Median = 0.113
Excess kurtosis = 12.034194
4
3
2
1
0
chest_gyro_res

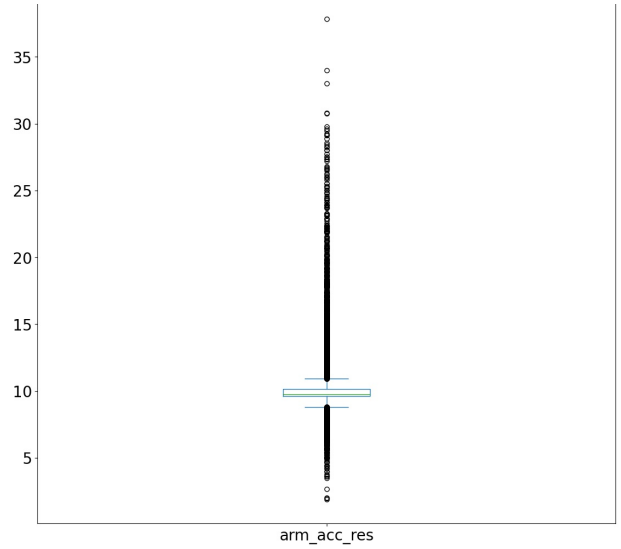

Histogram: Mean = 43.581 | Std = 12.264
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
0
20
40
60
80
100
120
Variable: chest_magneto_res

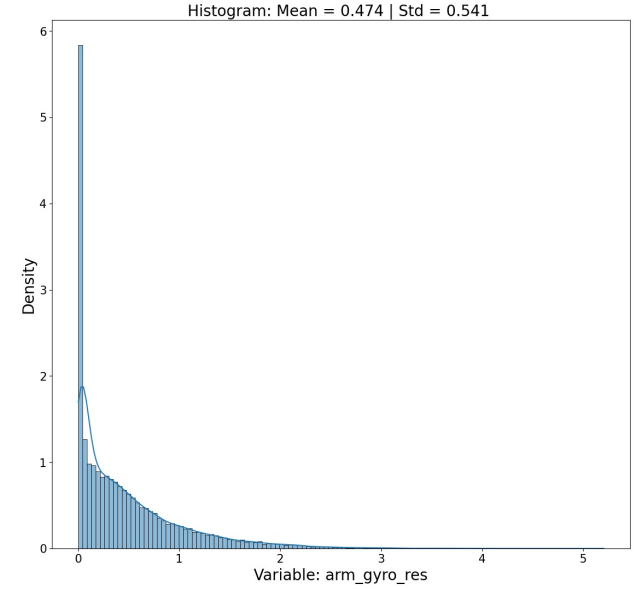

Boxplot: Skew = 0.477 | Median = 41.829
Excess kurtosis = 0.659774
120
100
80
60
40
20
0
chest_magneto_res

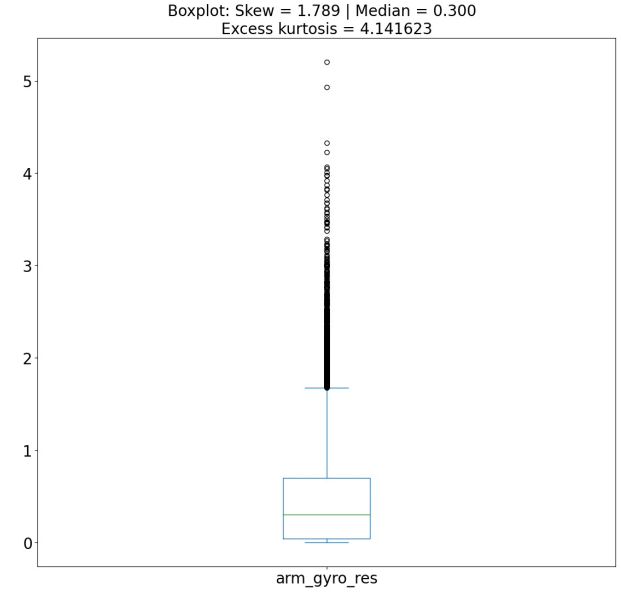

Histogram: Mean = 10.068 | Std = 1.641

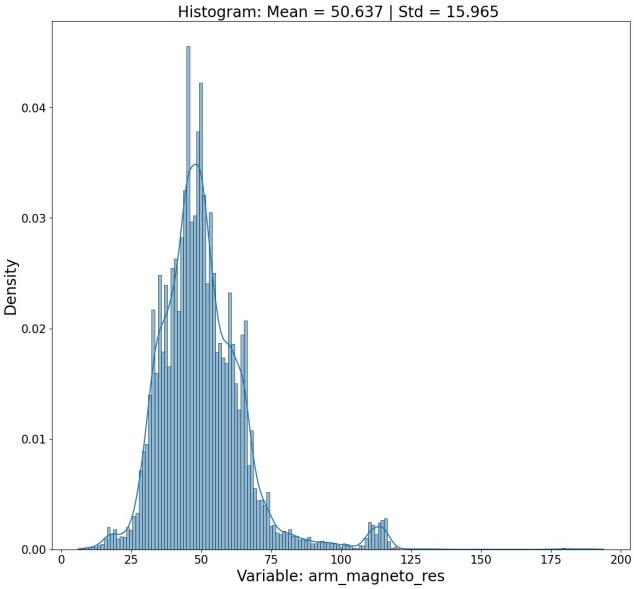

Boxplot: Skew = 4.666 | Median = 9.747
Excess kurtosis = 38.631006

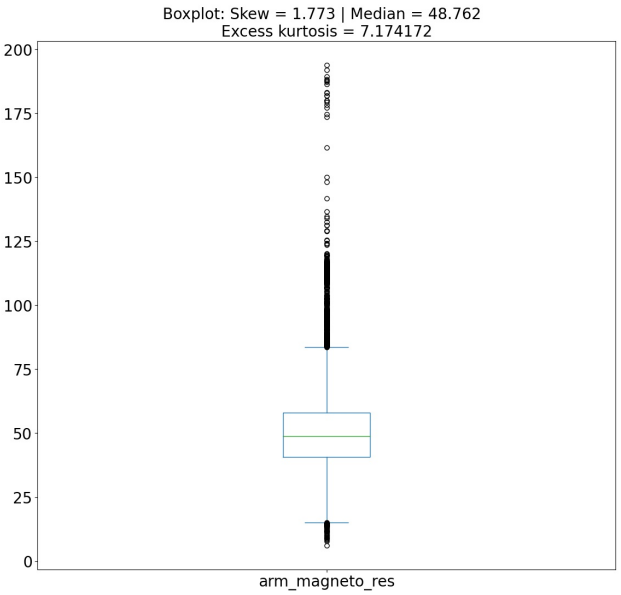










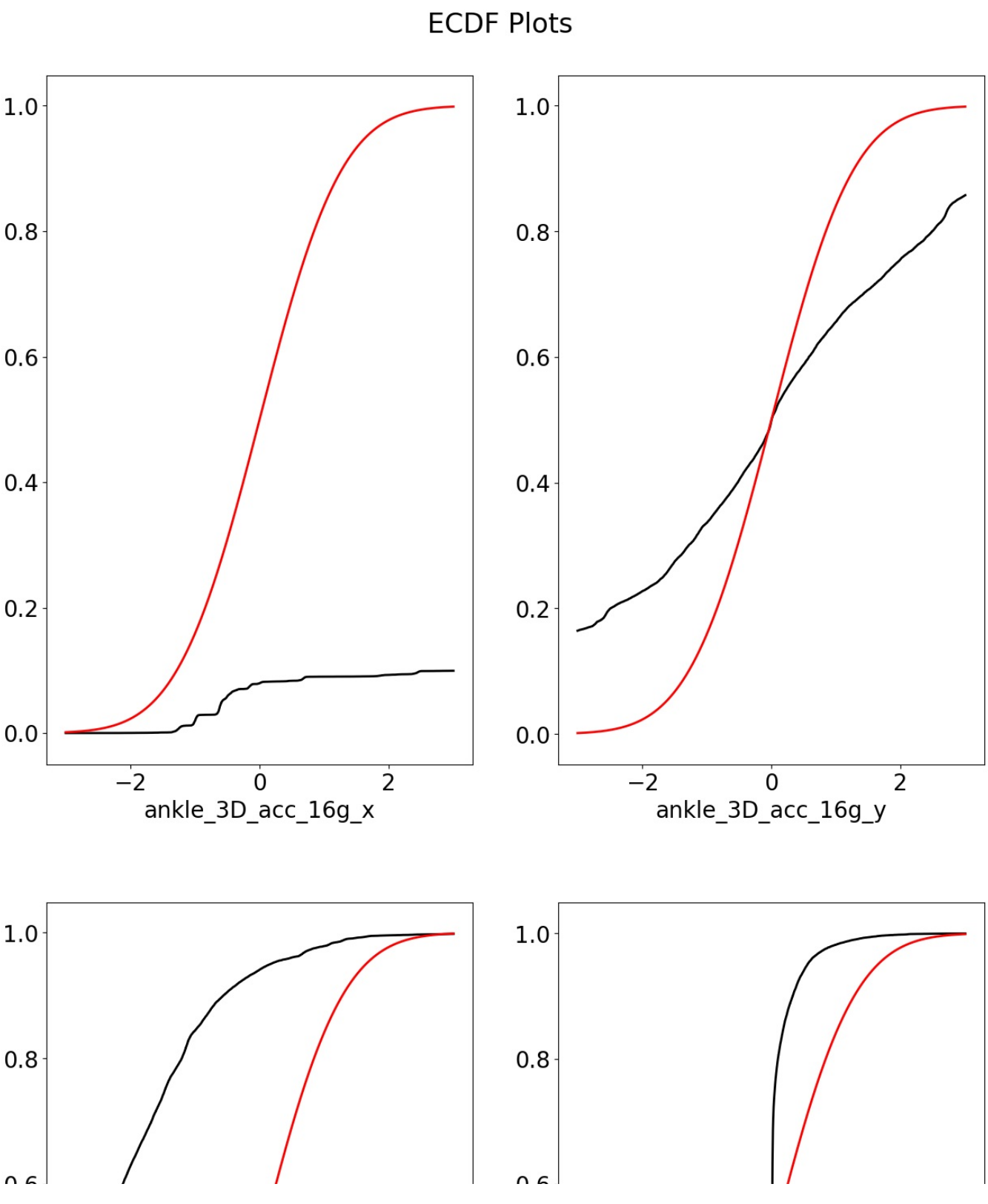


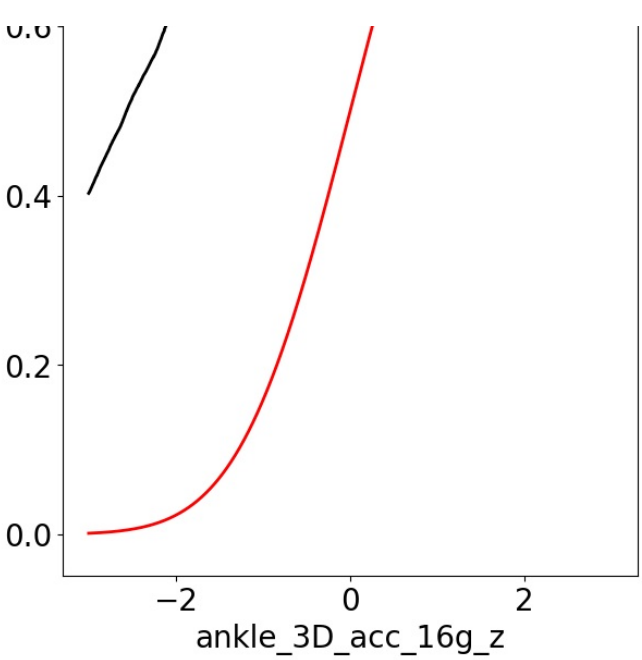
Additionally, we conducted normality tests to assess whether the features in the dataset follow a statistically Gaussian distribution. This step is crucial for deciding on the appropriate methodology between normalisation and standardisation for the subsequent mathematical model.

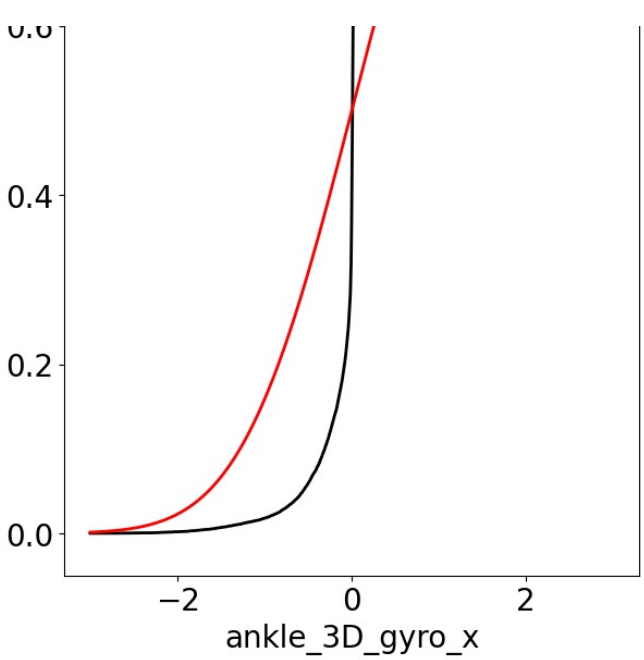
To confirm normality, we visualised the features in the dataset using an Empirical Cumulative Distribution Function (ECDF) plot, if the shape of the ECDF curve closely resembles that of the data distribution, there is a high likelihood that the data follows a Gaussian distribution. At the same time, we formulated a null hypothesis and an alternative hypothesis. The null hypothesis postulates that the feature in the dataset follows a normal distribution. Conversely, the alternative hypothesis suggests that the feature in the dataset does not follow a normal distribution. Subsequently, D'Agostino's K-squared and Lilliefors tests were conducted at the 1% significance level to determine whether the null hypothesis could be rejected.

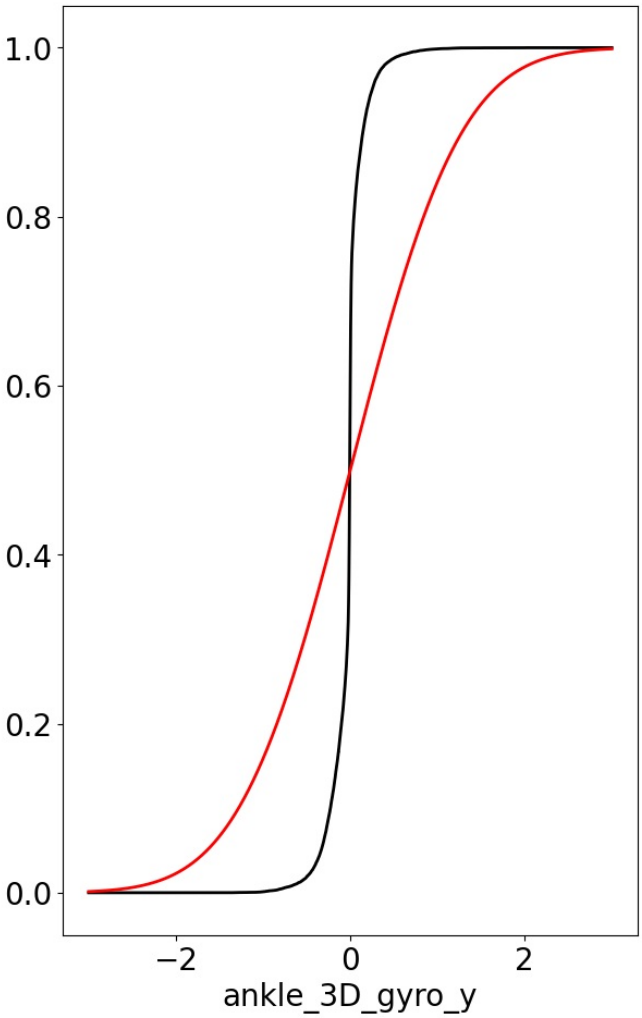
In [20]:

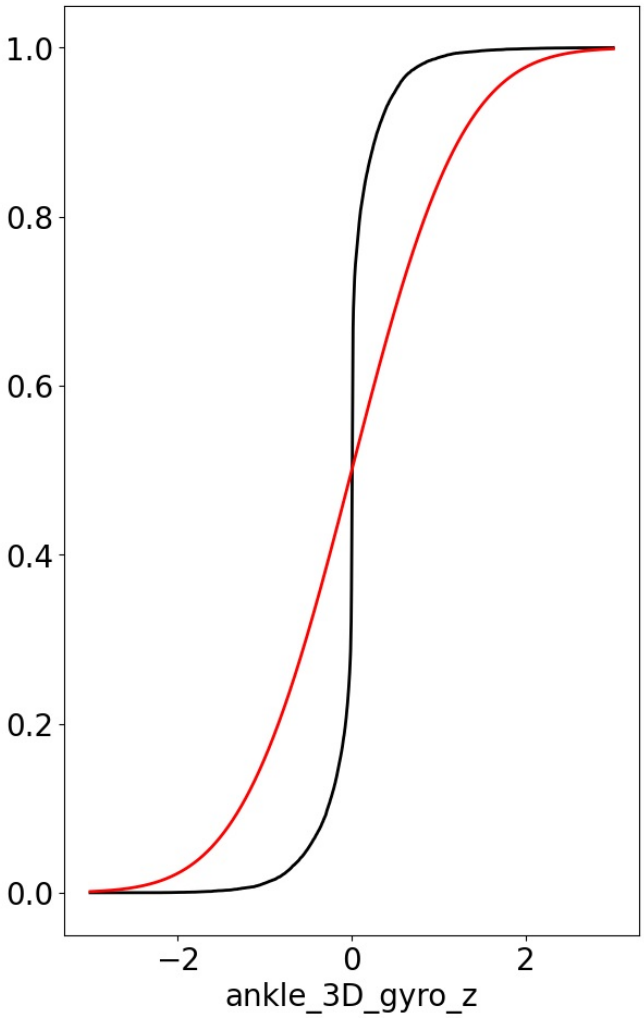
```
"""
Creating an ECDF plot to compare the distribution of each column with a normal distribution (Original codes fro
"""

from statsmodels.distributions.empirical_distribution import ECDF
from scipy.stats import norm

fig, axes = plt.subplots(20, 2, figsize=(15,6))

plt.rcParams.update({'font.size': 20})

# Drawing ECDF plots using statsmodels
for ax, col in zip(axes.flat, eda_df.columns):
    samples = eda_df[col].dropna()
    xax = np.linspace(-3, 3, len(eda_df[col]))
    ecdf = ECDF(samples)
    ax.plot(xax, ecdf(xax), c="k", lw=2)
    ax.plot(xax, norm.cdf(xax), c="r", lw=2)
    ax.set_xlabel(col)

fig.set_figheight(fig.get_figheight() * len(eda_df.columns)*1.1)
plt.suptitle('ECDF Plots', y=0.883)
plt.show()
```

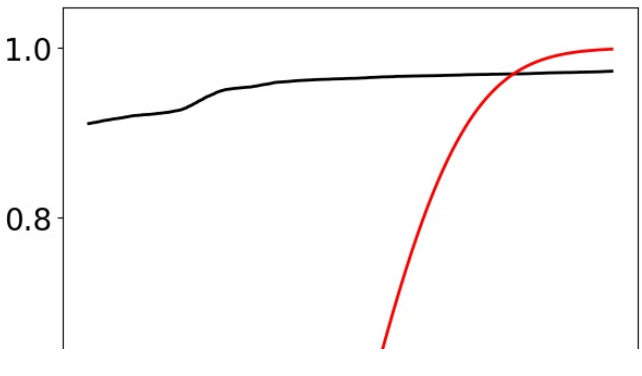


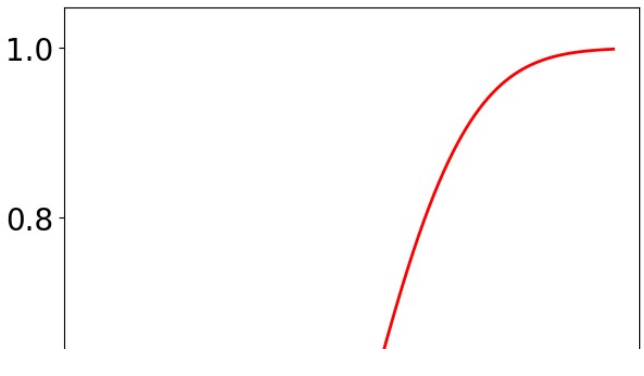

0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_acc_16g_z

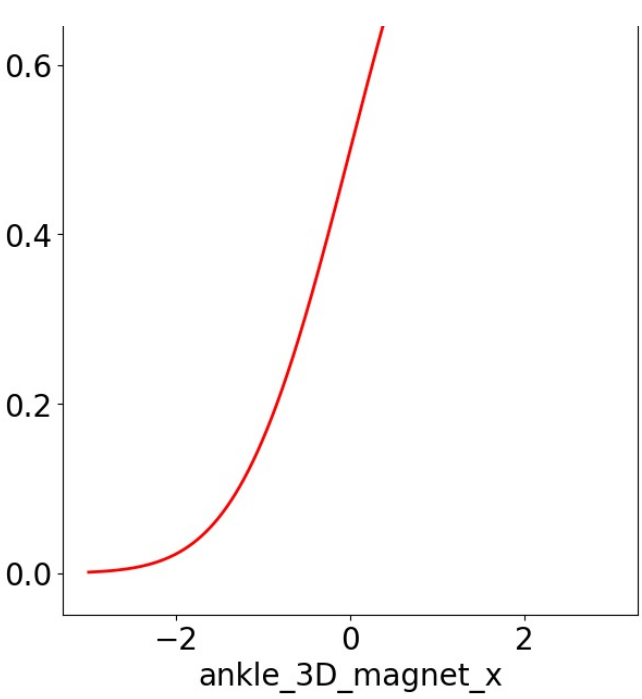

0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_gyro_x

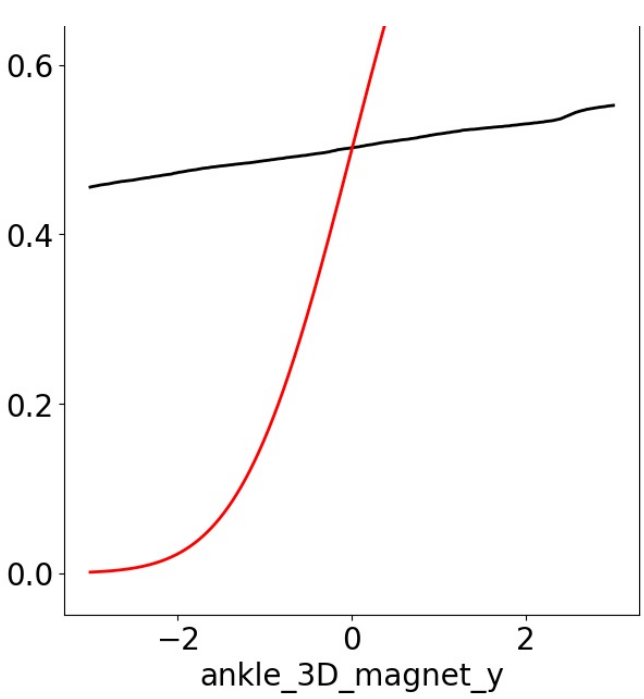

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_gyro_y

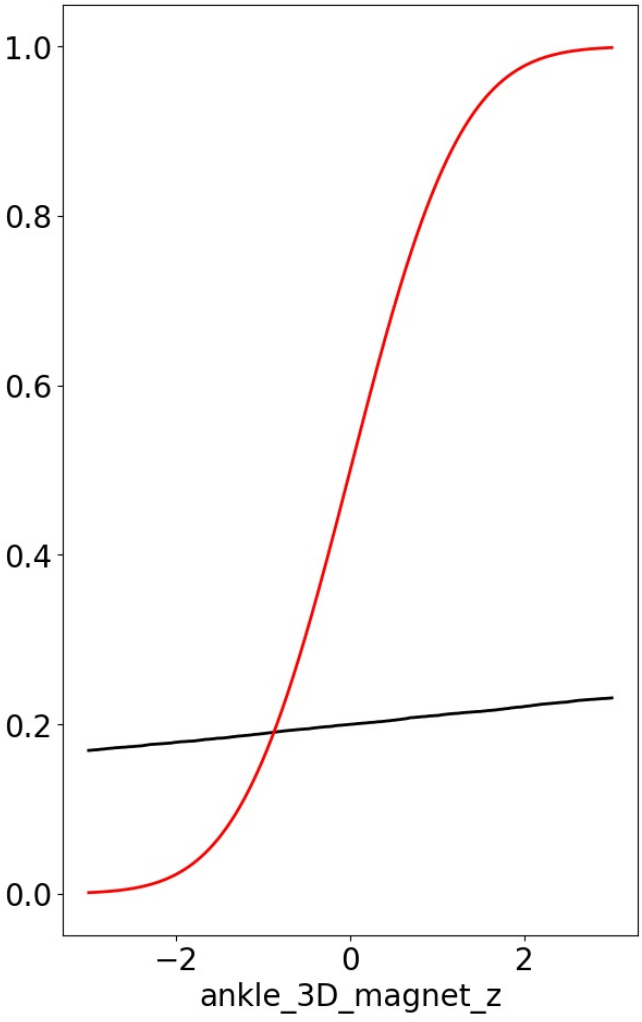

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_gyro_z

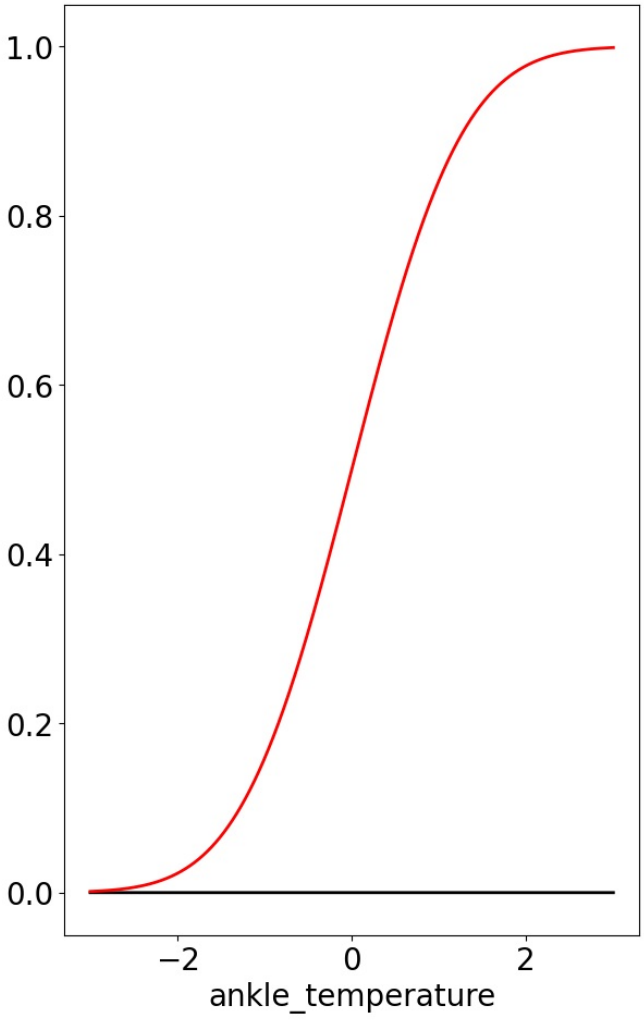

1.0
0.8

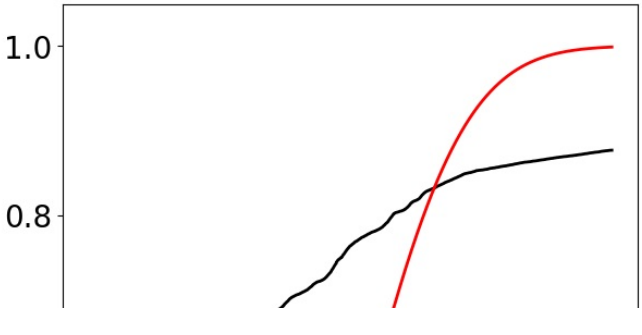

1.0
0.8

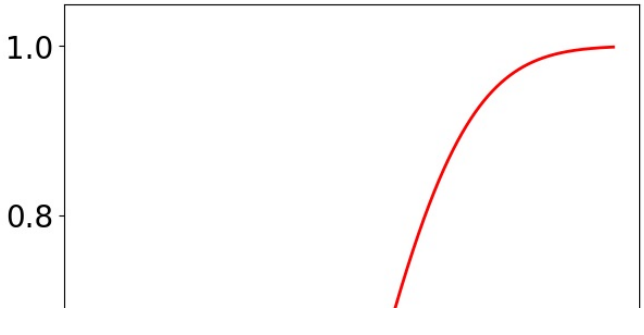

0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_magnet_x

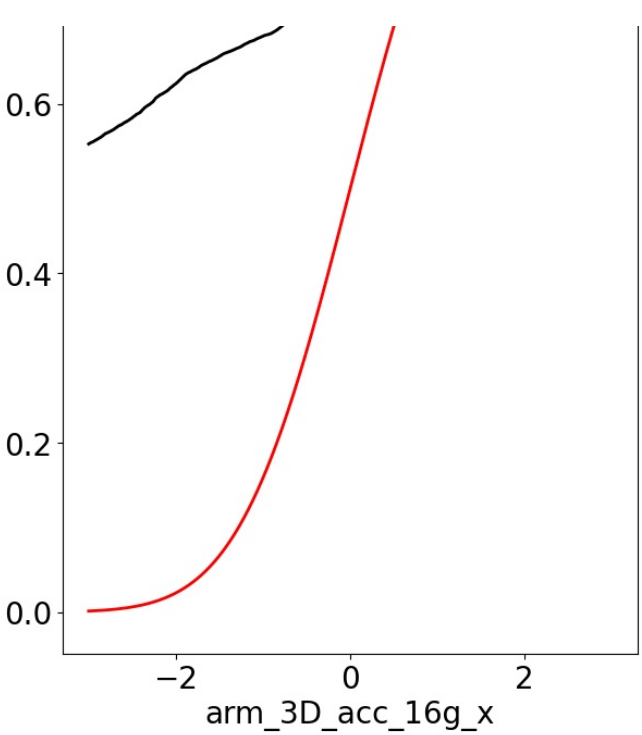

0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_magnet_y

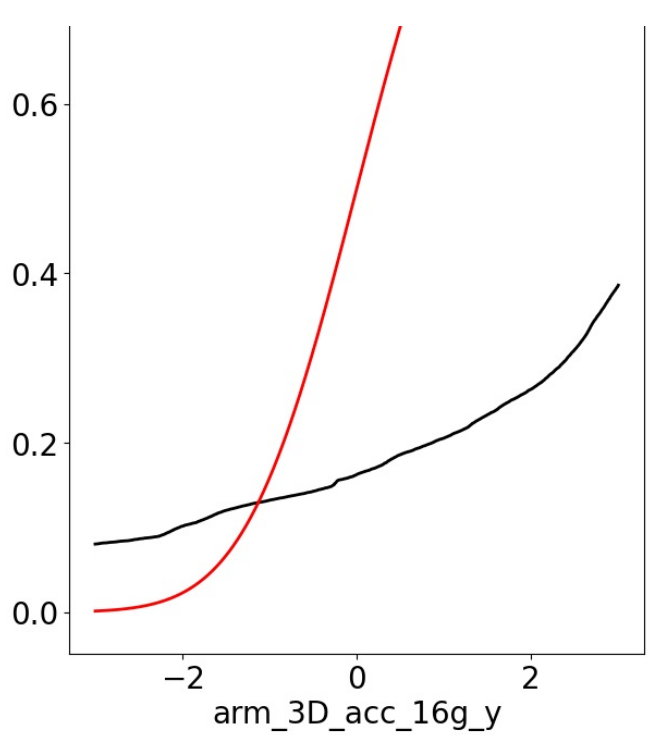

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_3D_magnet_z

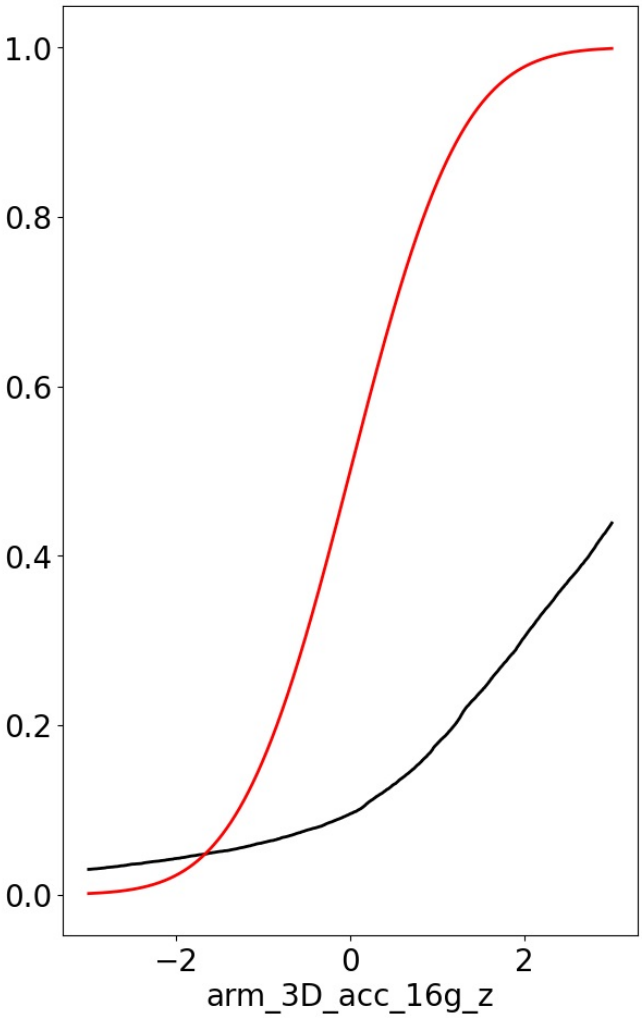

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_temperature

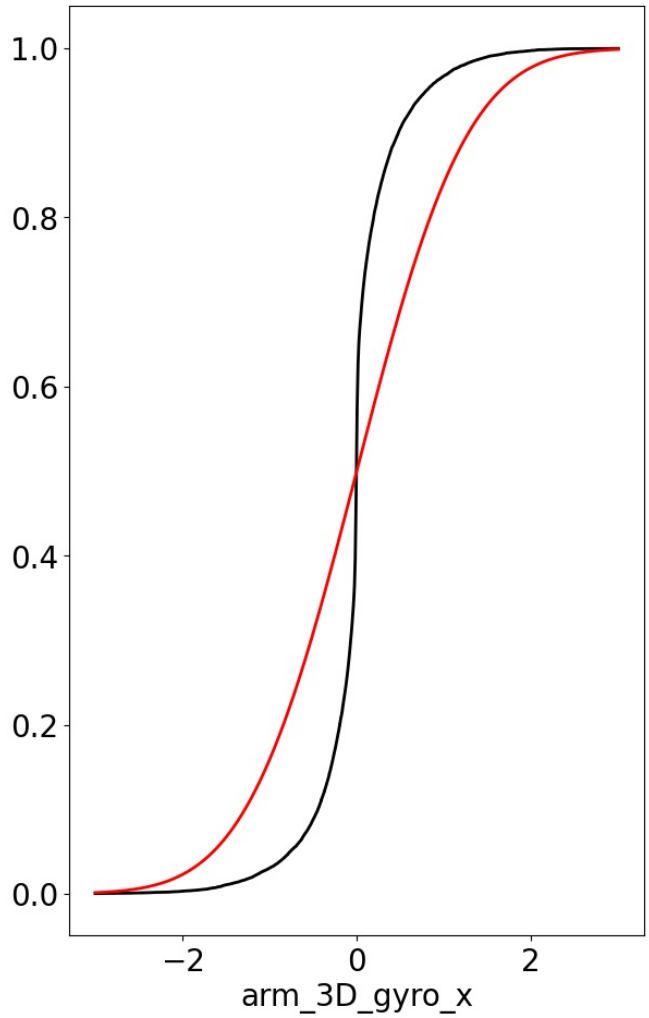

1.0
0.8

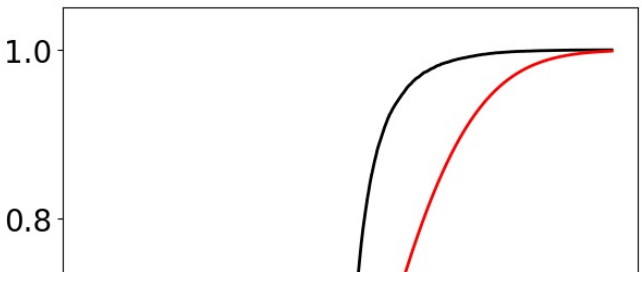

1.0
0.8

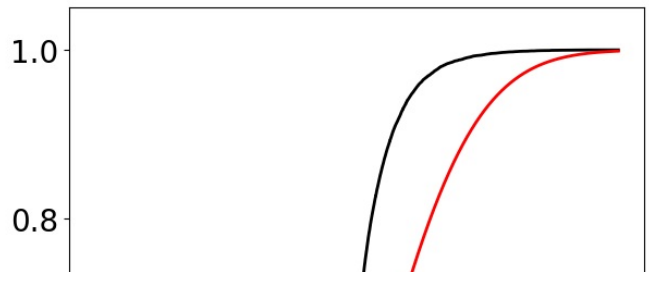

0.6
0.4
0.2
0.0
−2
0
2
arm_3D_acc_16g_x

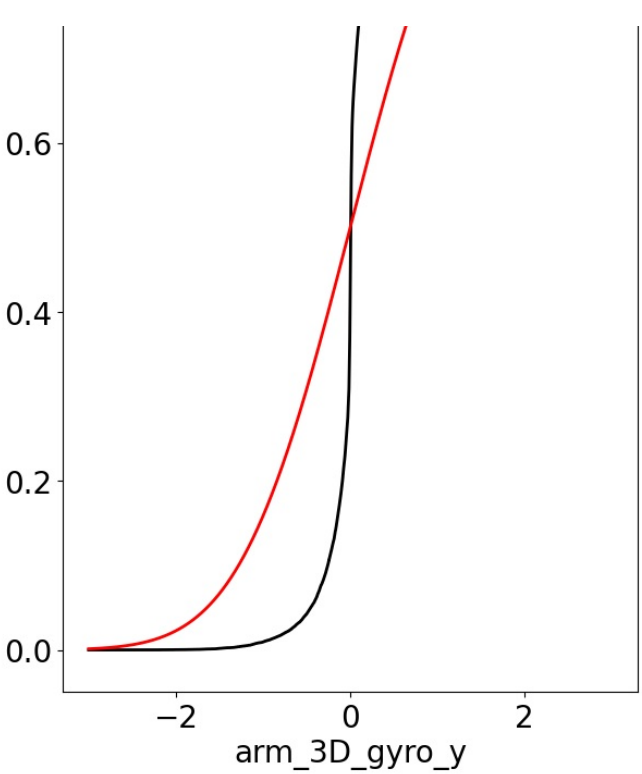

0.6
0.4
0.2
0.0
−2
0
2
arm_3D_acc_16g_y

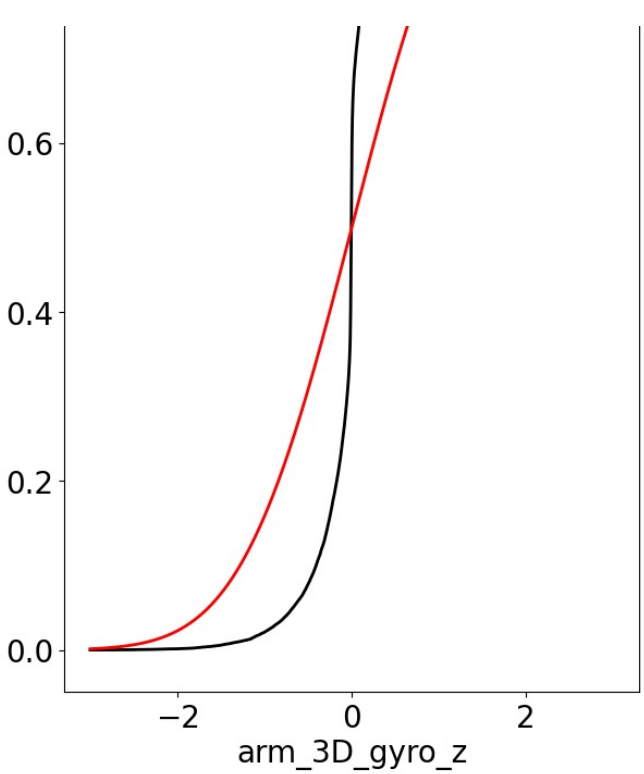

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
arm_3D_acc_16g_z

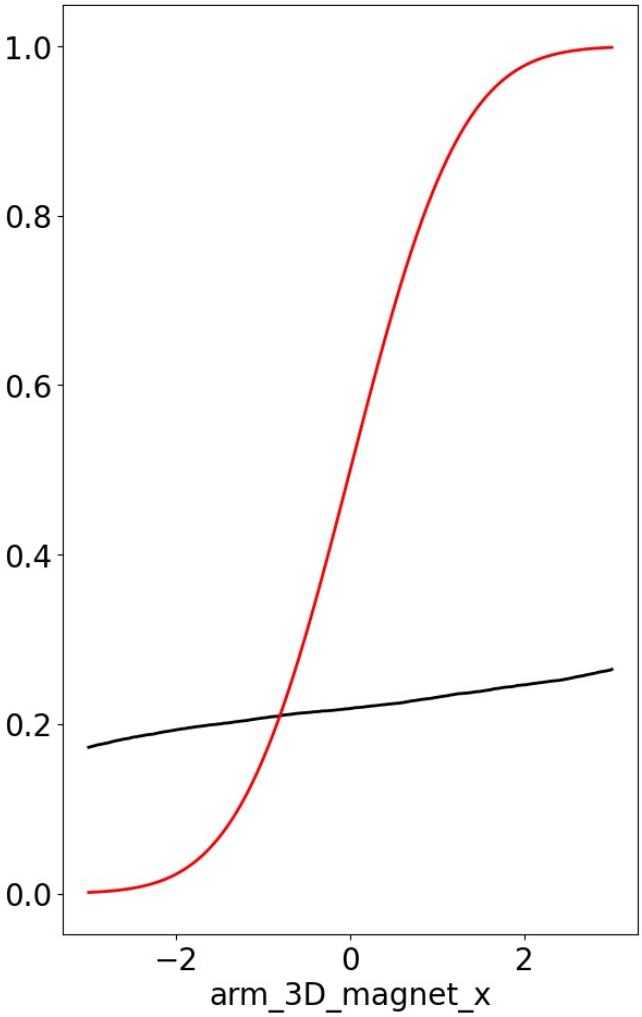

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
arm_3D_gyro_x

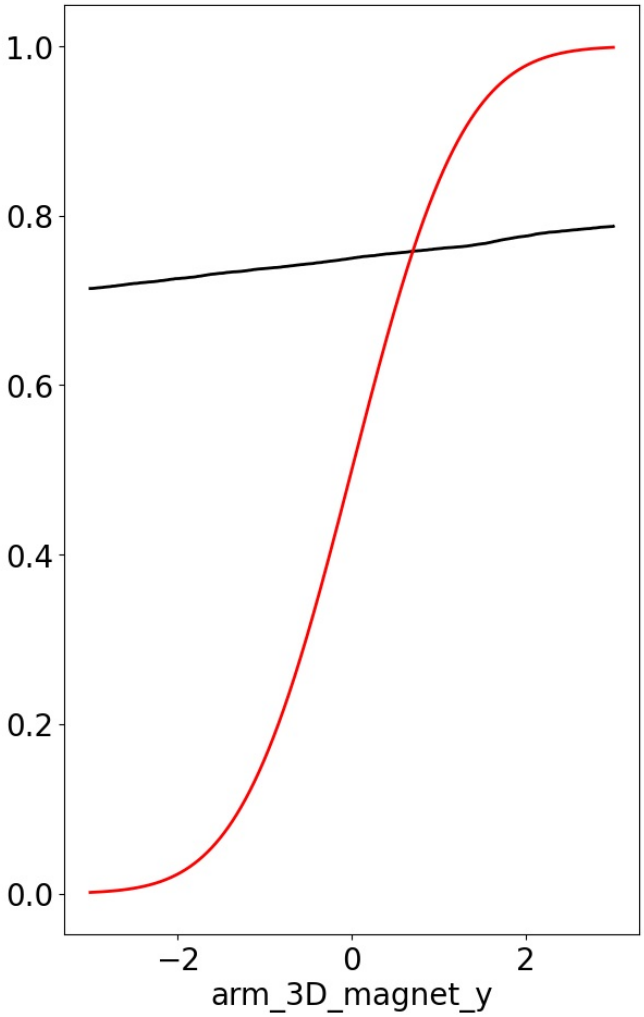

1.0
0.8

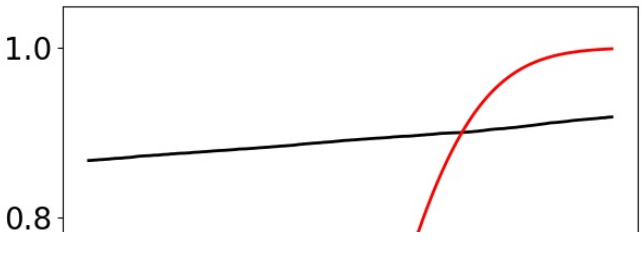

1.0
0.8


0.6
0.4
0.2
0.0
−2
0
2
arm_3D_gyro_y

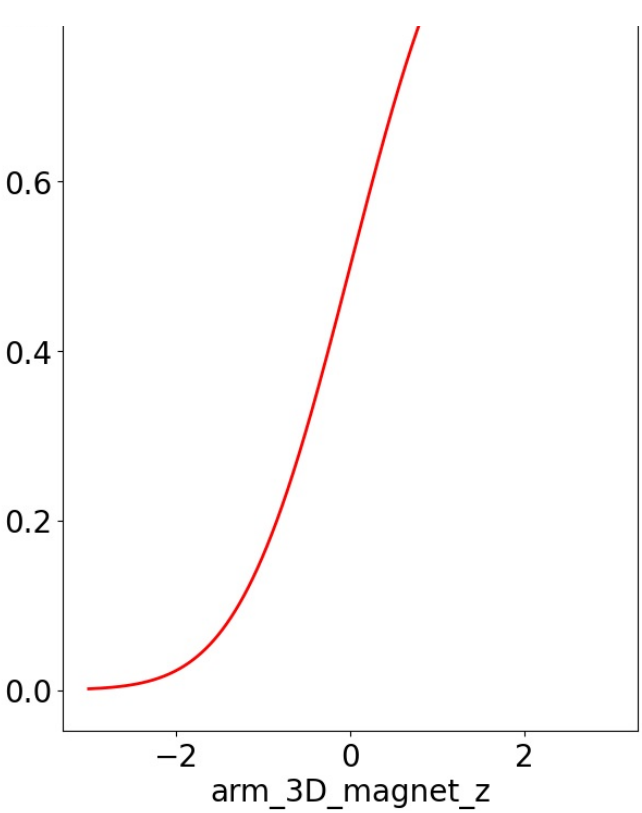

0.6
0.4
0.2
0.0
−2
0
2
arm_3D_gyro_z

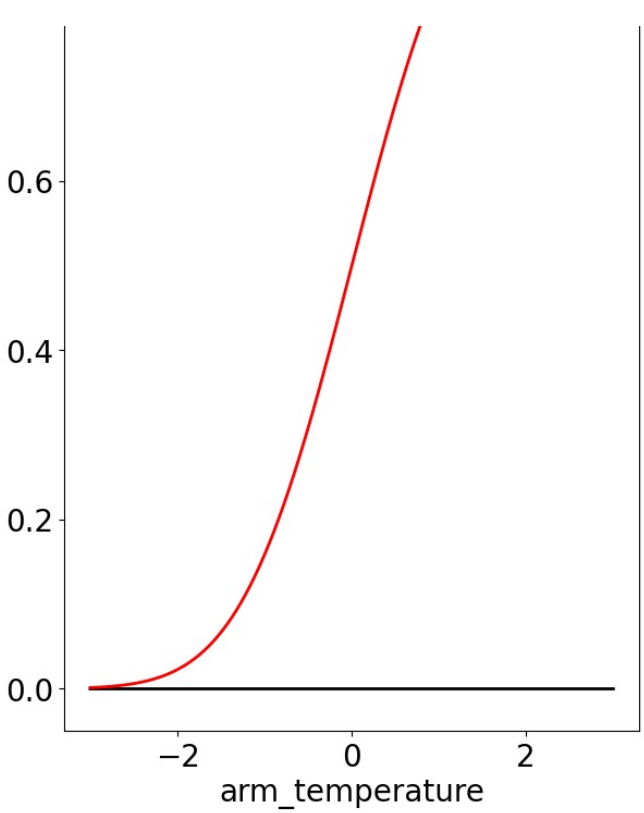

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
arm_3D_magnet_x

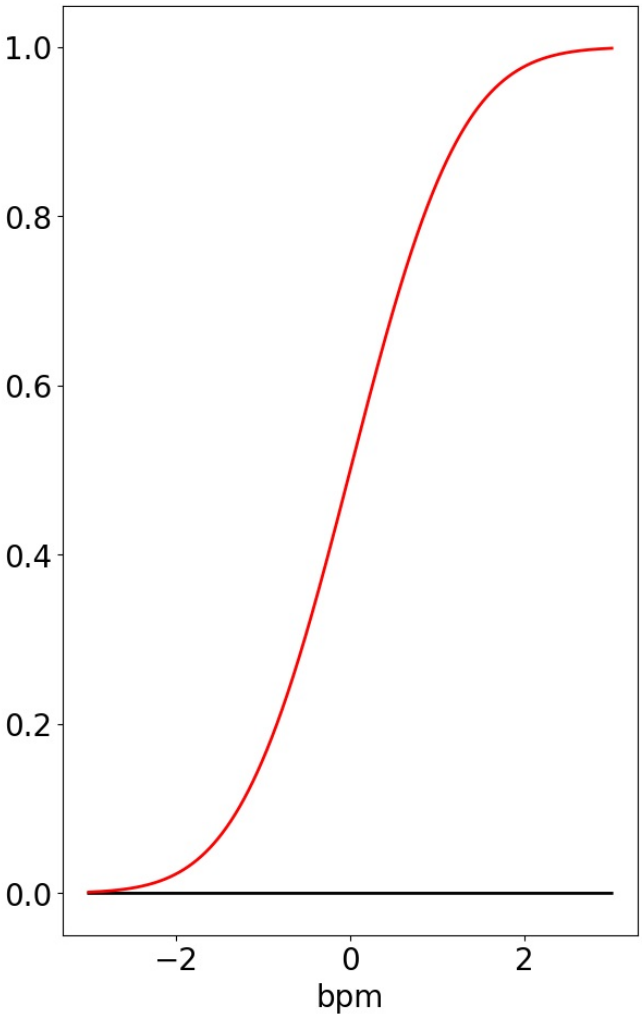

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
arm_3D_magnet_y

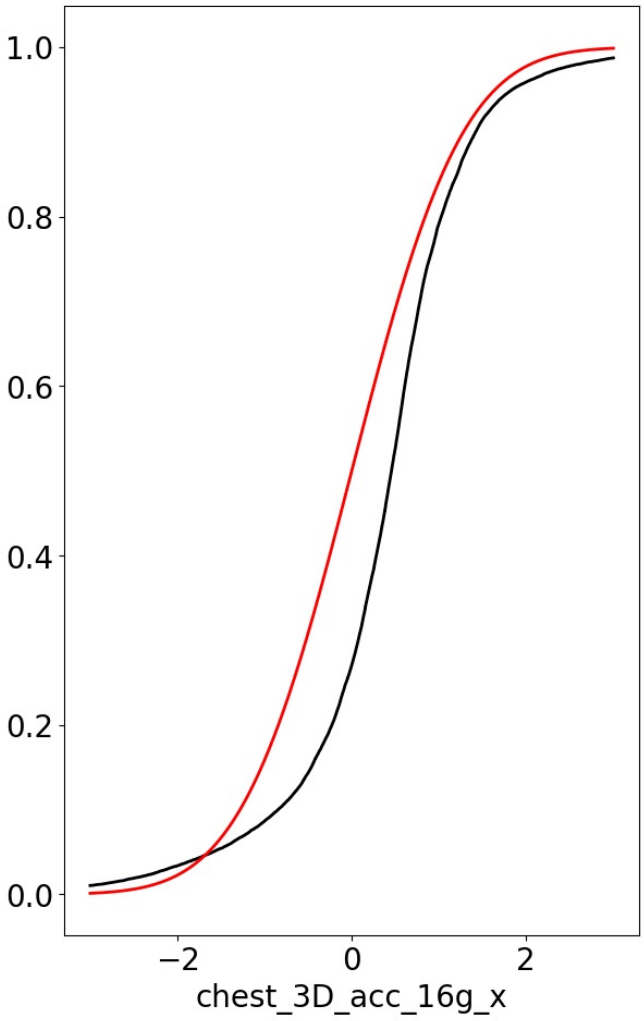

1.0
0.8


1.0
0.8


0.6
0.4
0.2
0.0
−2
0
2
arm_3D_magnet_z

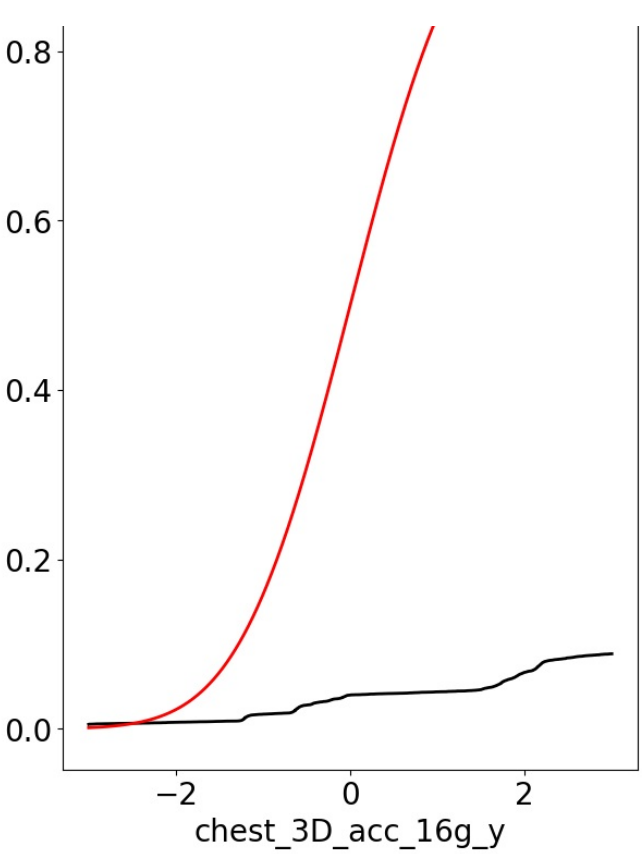

0.6
0.4
0.2
0.0
−2
0
2
arm_temperature

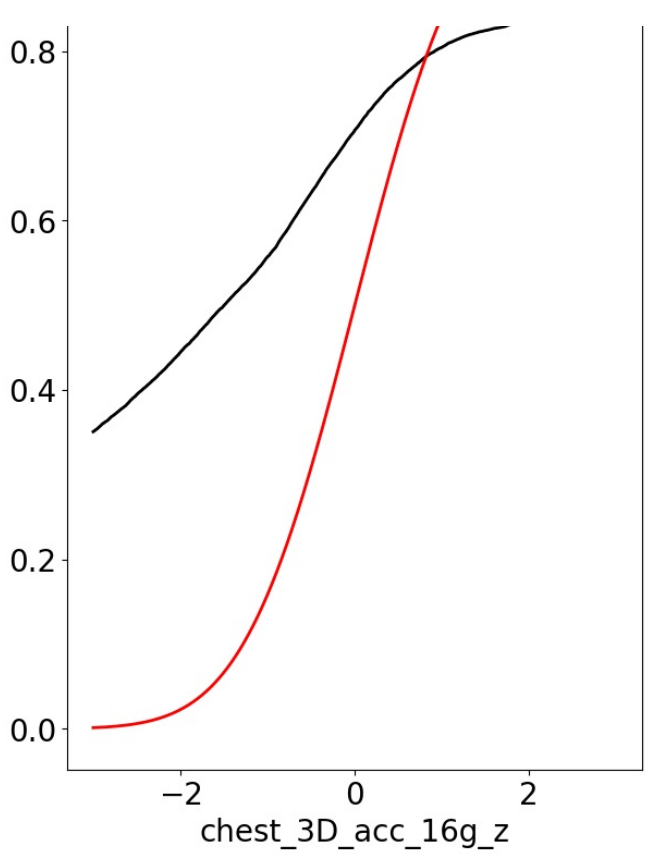

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
bpm

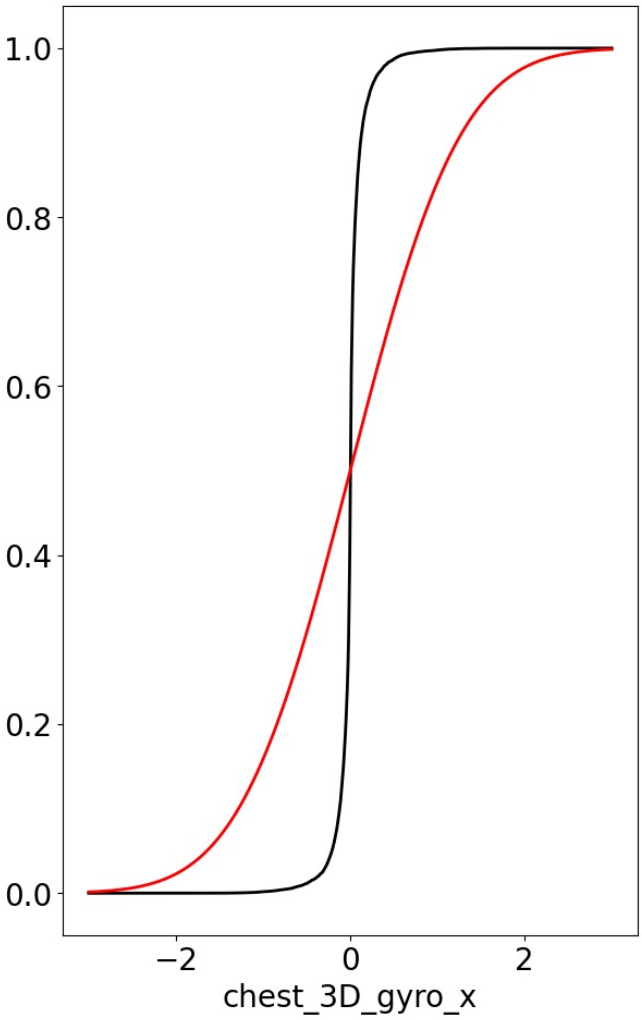

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_acc_16g_x

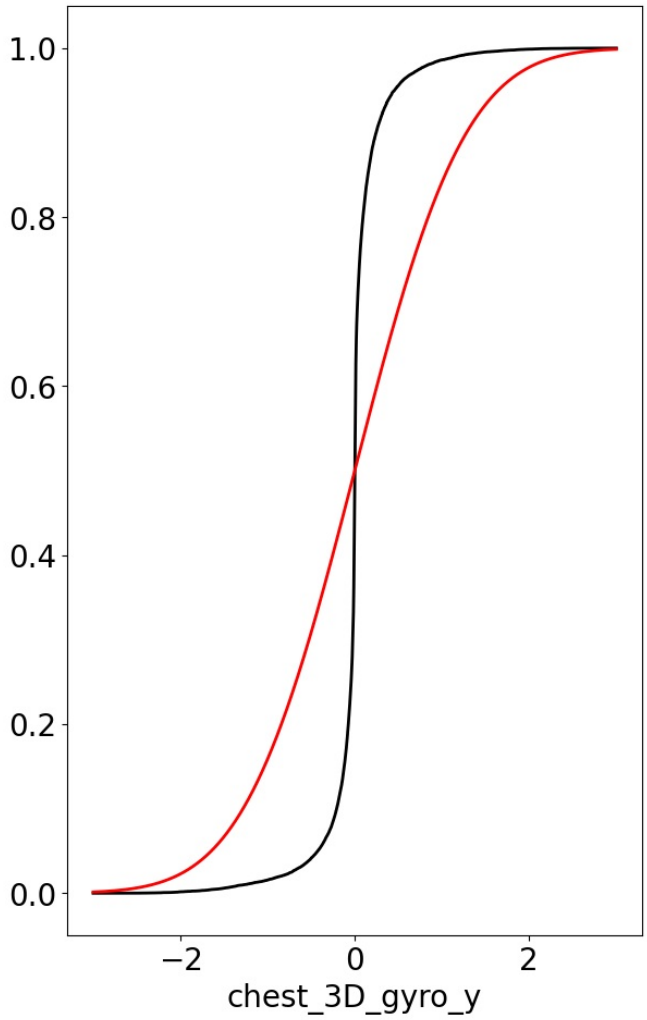

1.0

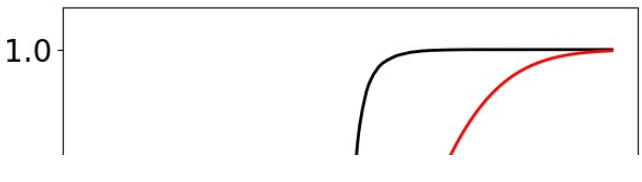

1.0


0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_acc_16g_y

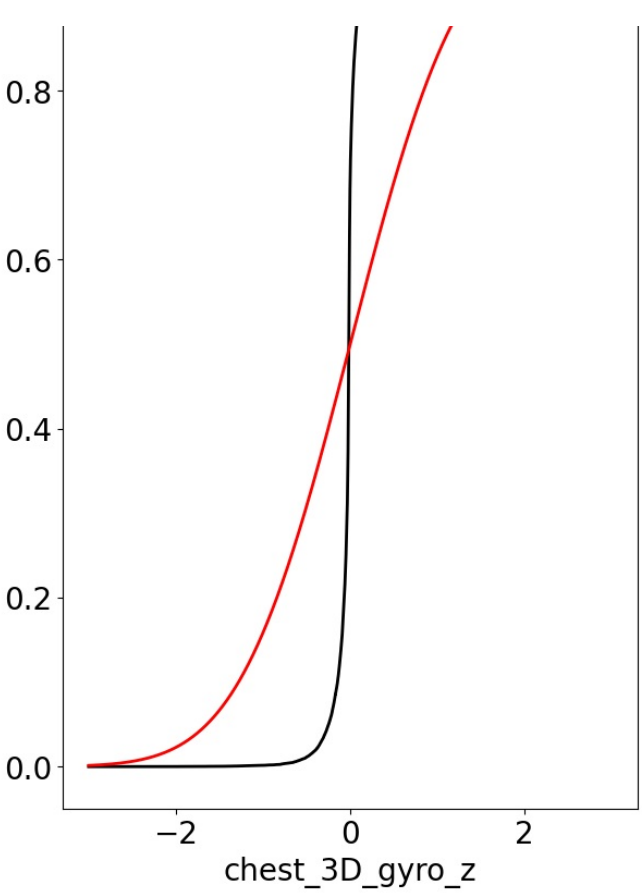

0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_acc_16g_z

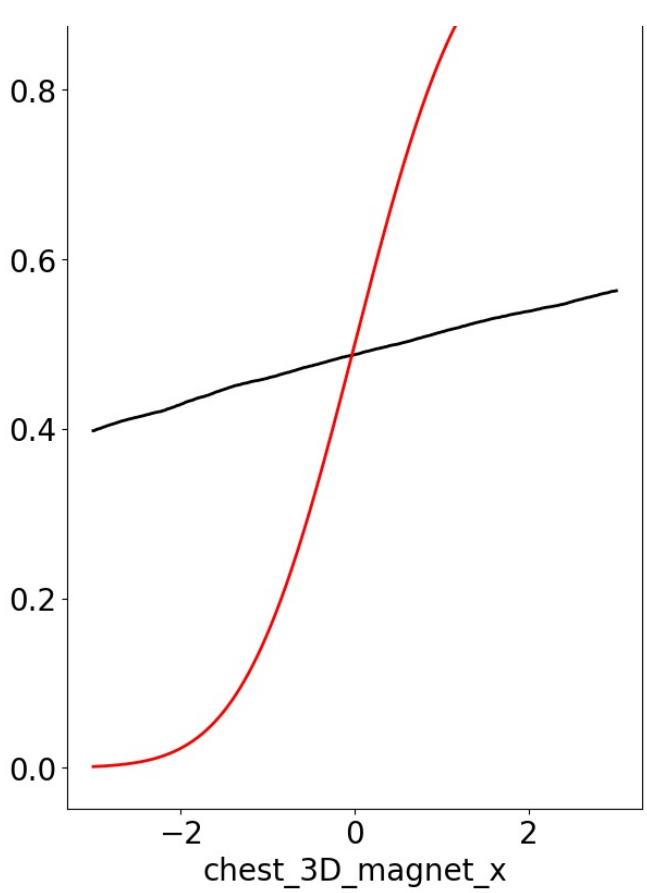

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_gyro_x

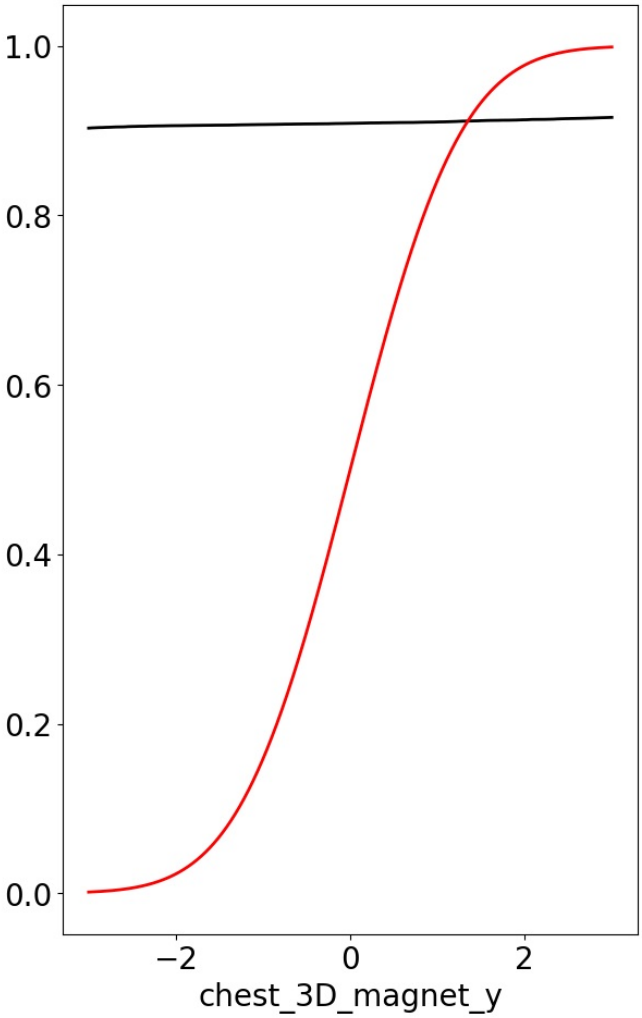

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_gyro_y

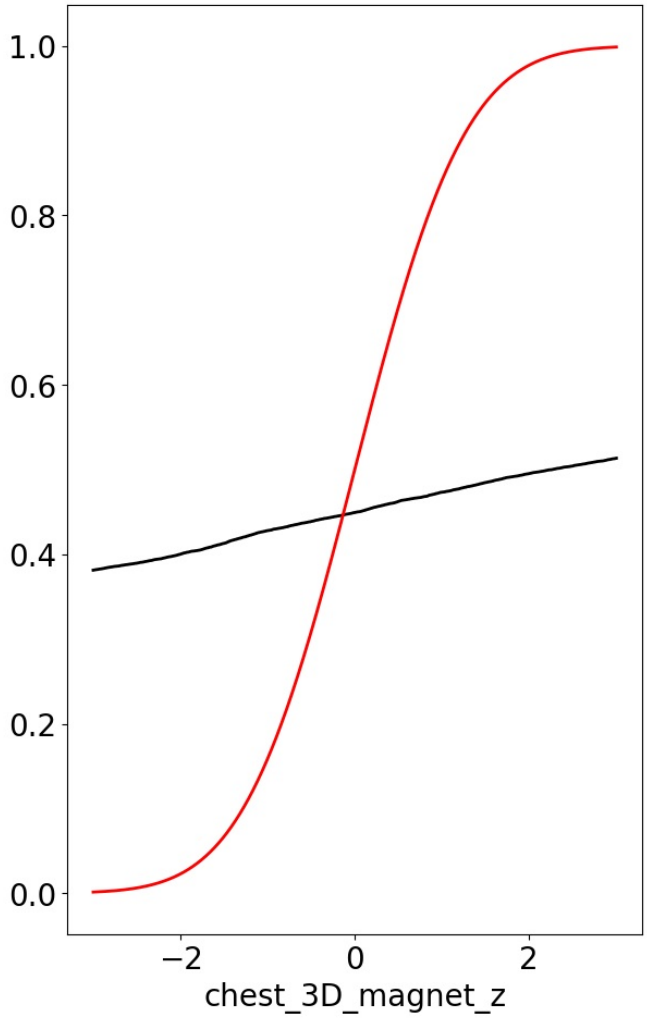

1.0


1.0


0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_gyro_z

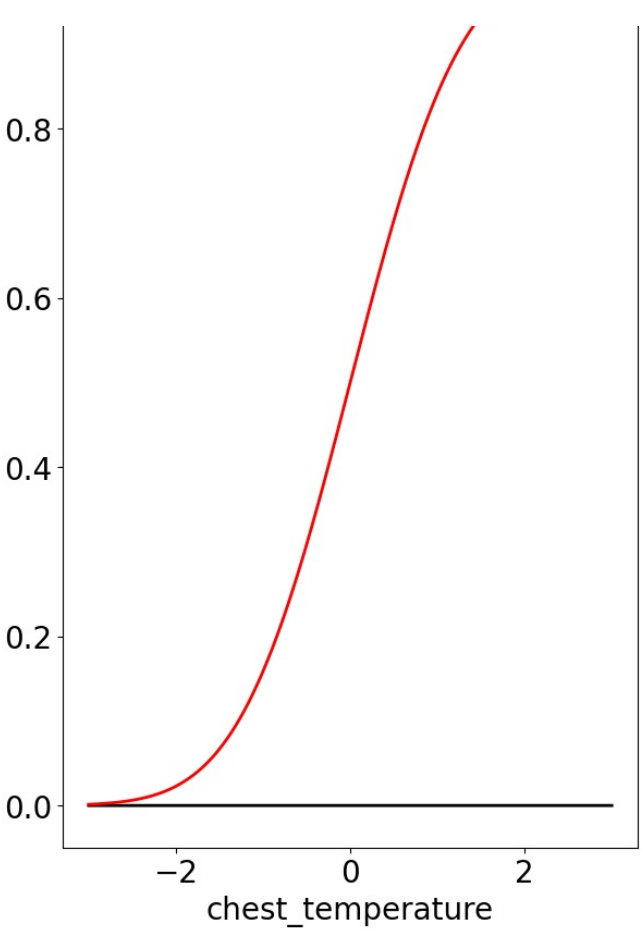

0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_magnet_x

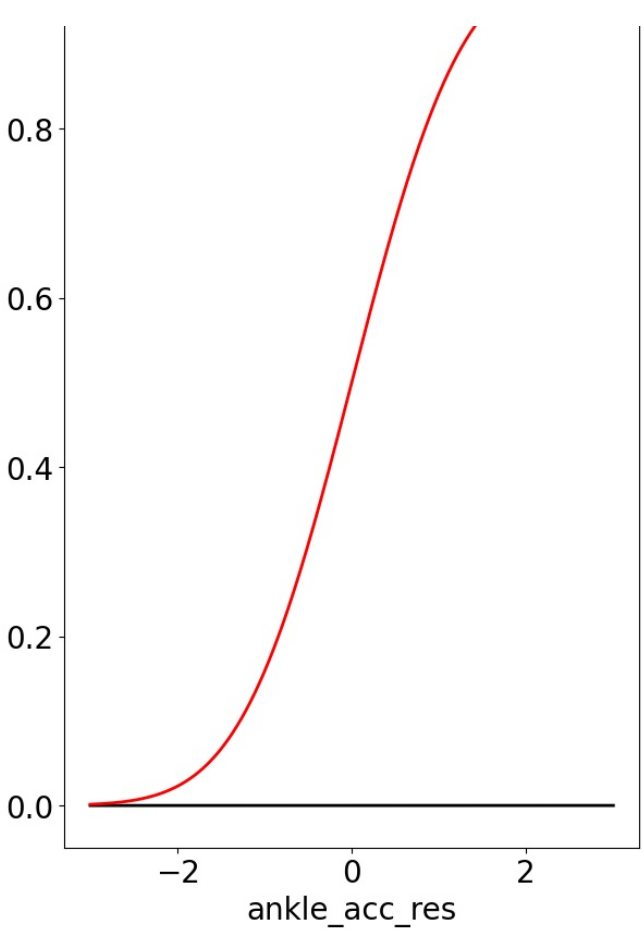

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_magnet_y

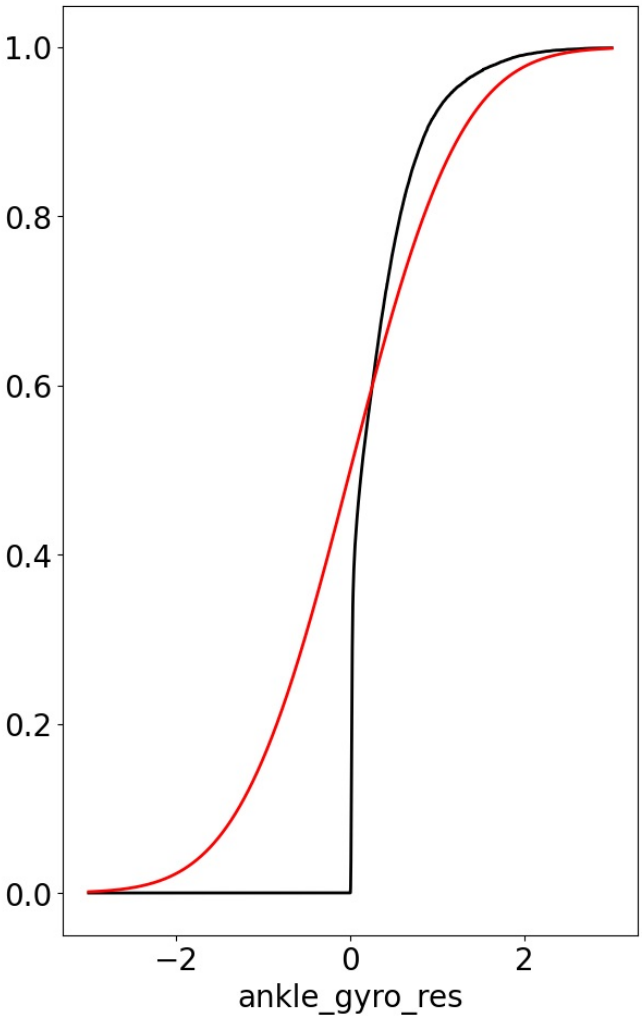

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_3D_magnet_z

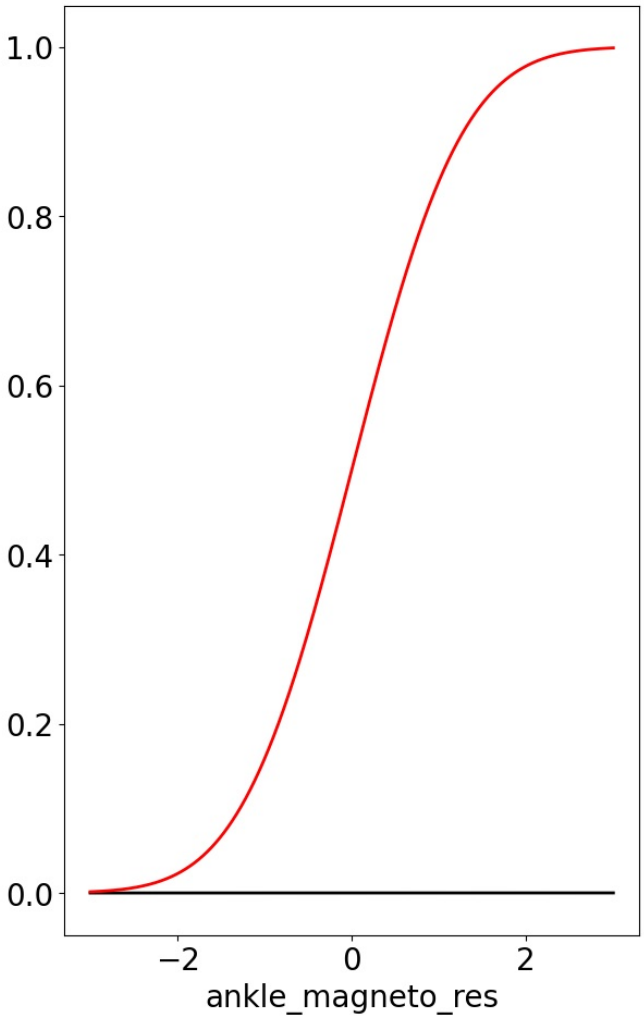

1.0


1.0


0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_temperature

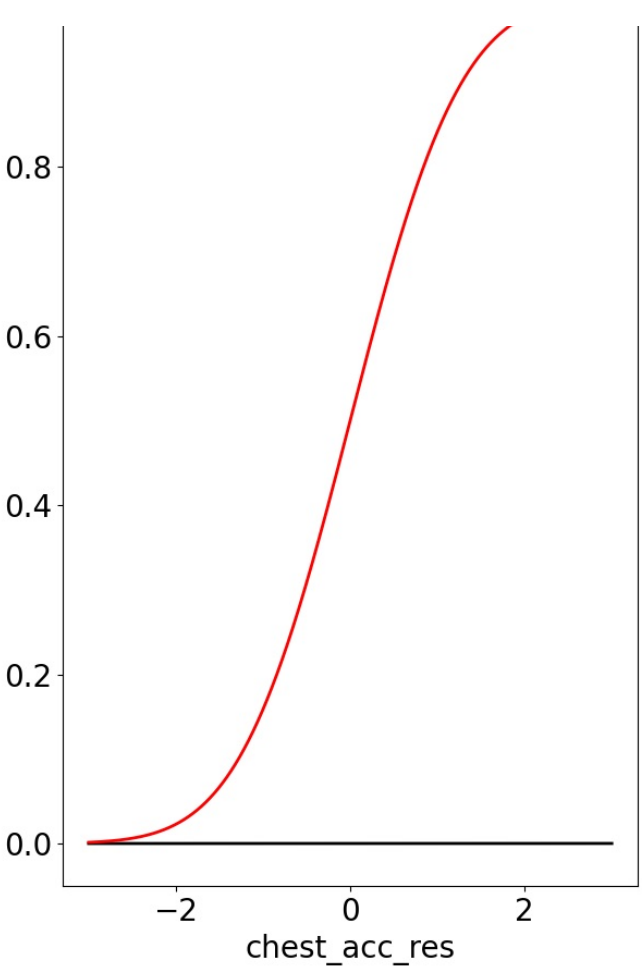

0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_acc_res

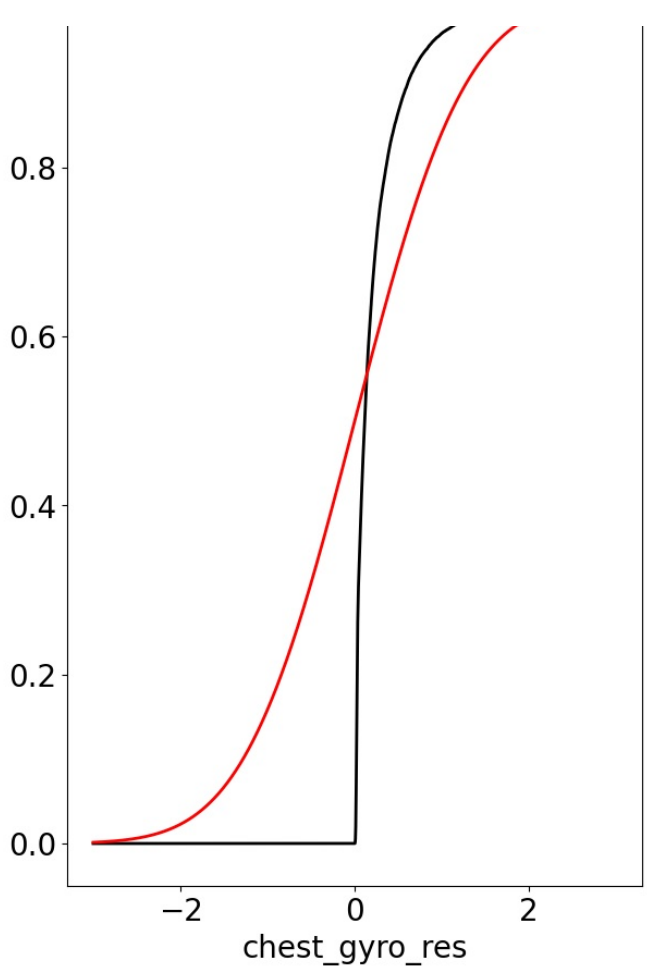

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_gyro_res

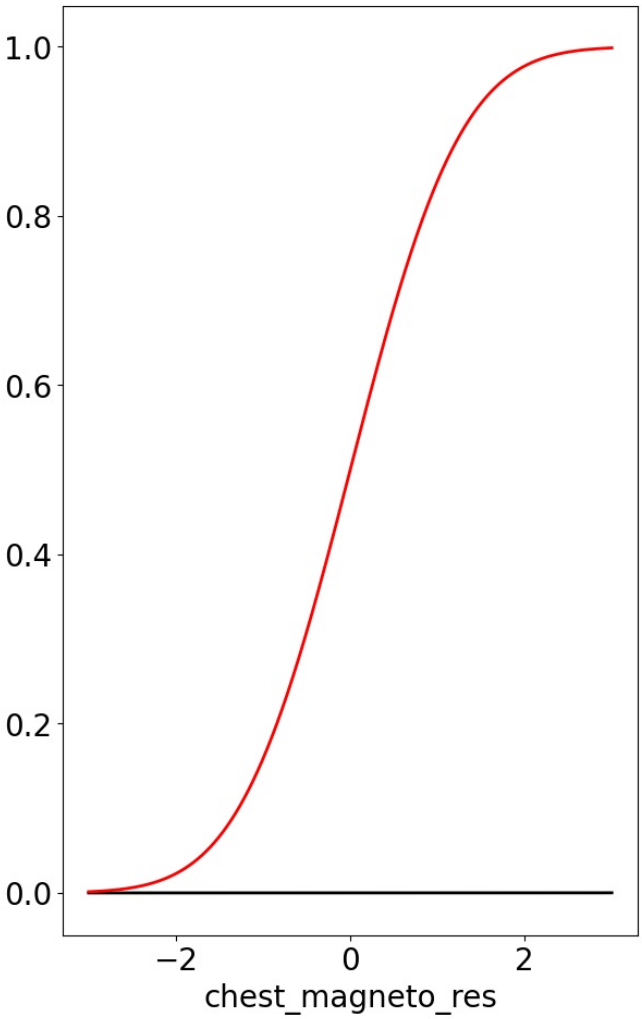

1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
ankle_magneto_res

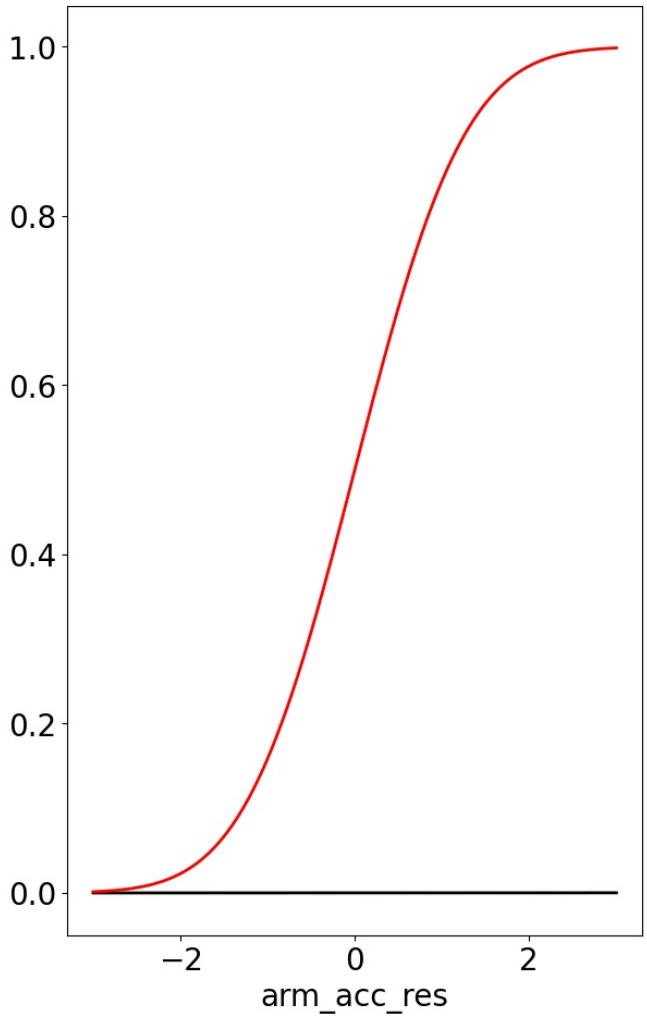

1.0

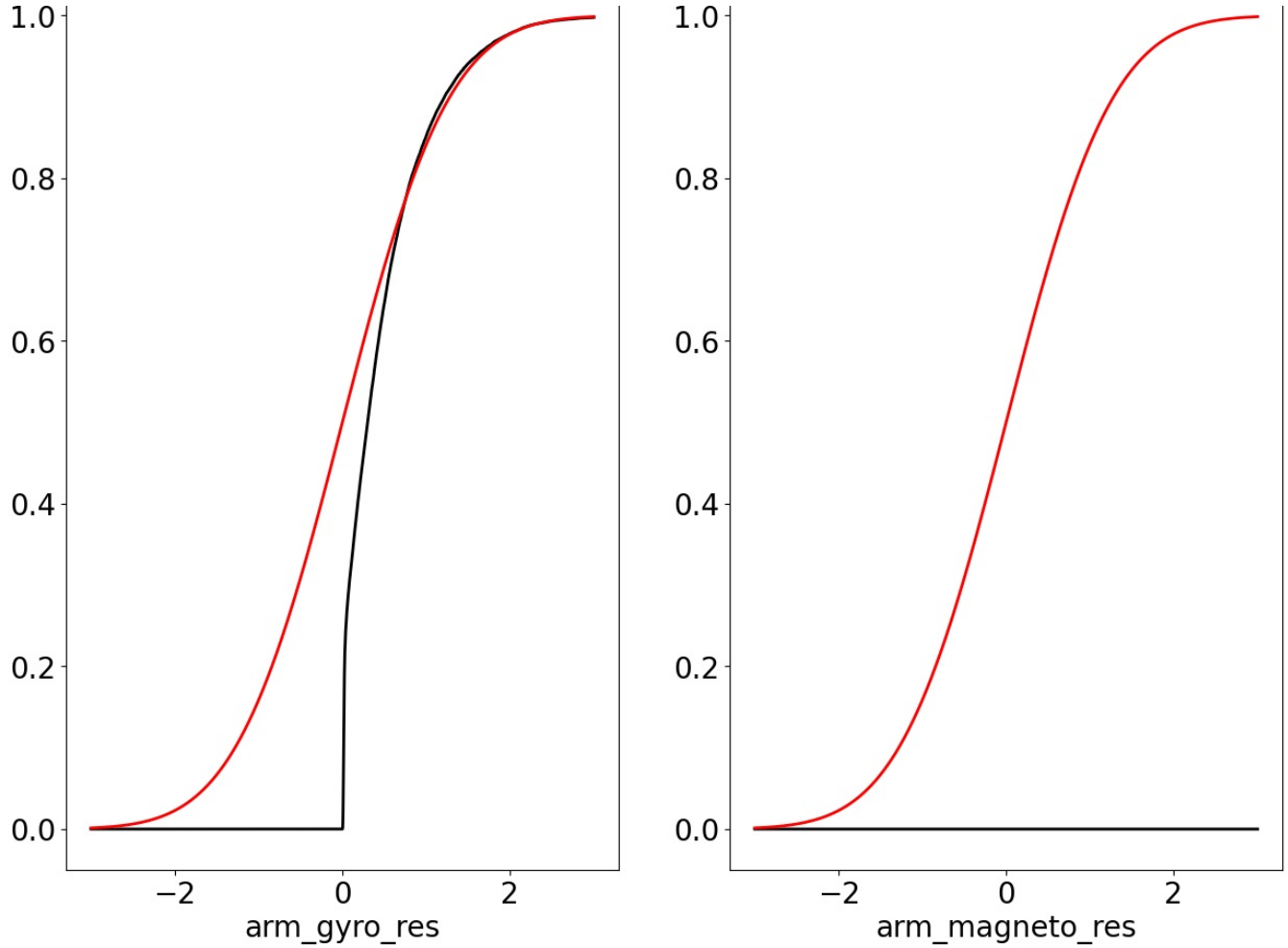

1.0


0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_acc_res


0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_gyro_res


1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
chest_magneto_res


1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
arm_acc_res



In [21]:

```
"""
Defining functions for normality hypothesis tests ((Original codes from James, revised by Jungyeon Lee))
"""

from scipy.stats import normaltest
from statsmodels.stats.diagnostic import lilliefors

# Statement function
def hypothesis_statement(p_value, alpha):
    c = 100.0*alpha
    if p_value < alpha:
        return f"We reject the Null Hypothesis in favour of the Alternative Hypothesis at the {c}% level, p-val
    else:
        return f"We accept the Null Hypothesis at the {c}% level, p-value = {p_value}", True

# Normality test function (D'Agostino's Ksquared and Lilleifors test)
def normality_tests(df, alpha=0.01):
    norm_list = []
    not_norm_list = []

    for col in df.columns:
        _, p_value1 = normaltest(df[col])
        print(f"Null hypothesis: The distribution of {col} feature is a Gaussian")
        txt, res = hypothesis_statement(p_value1,alpha)
        print(f"D'Agostino's Ksquared test. {txt}")
        _, p_value1 = lilliefors(df[col],dist="norm")
        txt, res1 = hypothesis_statement(p_value1,alpha)
        print(f"Lilliefors test. {txt}")
        print("")
        if res1 == True or res == True:
            norm_list.append(col)
        else: not_norm_list.append(col)
    return norm_list, not_norm_list

# Determinant function
def determinant(norm_list, not_norm_list):
    if len(norm_list) == 0:
        print("Every columns does not follow a Gaussian distribution")
    else:
        print(f'Gaussian Distribution: {norm_list}')
        print(f'Non-Gaussian Distribution: {not_norm_list}')
```

In [22]:

```
"""
Conducting normality tests on the dataset
"""

norm_list, not_norm_list = normality_tests(eda_df , alpha=0.01)
```

```
Null hypothesis: The distribution of ankle_3D_acc_16g_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_acc_16g_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_acc_16g_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_gyro_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_gyro_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_gyro_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_magnet_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 6.884901809461993e-185
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_magnet_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 3.715077590119312e-08
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_3D_magnet_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 4.6305544226590125e-170
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_temperature feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 8.071398760697929e-200
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_acc_16g_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 9.849945573955792e-235
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_acc_16g_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_acc_16g_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 2.4444906378900333e-112
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_gyro_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_gyro_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
```

```
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_gyro_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_magnet_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 1.8705846597664474e-46
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_magnet_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 1.723803501355605e-33
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_3D_magnet_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 2.6753088237876936e-248
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_temperature feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of bpm feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_acc_16g_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_acc_16g_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_acc_16g_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_gyro_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_gyro_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_gyro_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_magnet_x feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 4.391355410955139e-71
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_magnet_y feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_3D_magnet_z feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 1.1823604314150791e-54
```

```
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_temperature feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_acc_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_gyro_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of ankle_magneto_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_acc_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_gyro_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of chest_magneto_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 2.5091984580223124e-266
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_acc_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_gyro_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899

Null hypothesis: The distribution of arm_magneto_res feature is a Gaussian
D'Agostino's Ksquared test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% l
evel, p-value = 0.0
Lilliefors test. We reject the Null Hypothesis in favour of the Alternative Hypothesis at the 1.0% level, p-val
ue = 0.0009999999999998899
```

The result of the normality tests indicated that none of the features follow a Gaussian distribution, as the null hypothesis was rejected for each feature. Furthermore, the ECDF curve did not closely resemble that of any specific data distribution.

In summary, most of the variables in the dataset were measured across various activities, resulting in a broad range of values, and do not follow a normal distribution. Thus, it is advisable to opt for normalisation over standardisation. Standardisation is less suitable when dealing with non-Gaussian distributions and datasets that contain a diverse range of values.

In [23]:
```
"""
Check every columns using determinant function
"""

determinant(norm_list, not_norm_list)
```

```
Every columns does not follow a Gaussian distribution
```

The data team considered incorporating subject information into the Physical Activity Monitoring Data, including 'Subject ID', 'Sex', 'Age (years)', 'Height (cm)', 'Weight (kg)', 'Resting HR (bpm)', 'Max HR (bpm)', and 'Dominant hand'. We decided to integrate certain subject features such as 'Age,' 'Height,' and 'Weight,' as these could prove meaningful for further data analysis.

However, we also determined not to include some features in the subject information. Given that subjects had sufficient rest during the data collection stage, 'Resting HR' was excluded. In addition, since the data team can easily identify the maximum heart rate value within

the dataset, 'Max HR' was exempted as well. Considering the imbalance in gender distribution (eight males and one female) and the predominant right-handedness among subjects (eight right-handed and one left-handed), including 'Gender' and 'Dominant hand' features would pose challenges in maintaining dataset balance. As a result, we decided to not use the 'Gender' and 'Dominant Hand' features in the subject information, as their inclusion might hinder the deriving of appropriate insights.

Moreover, we associated the 'activityID' feature with its corresponding real activity name, creating the 'activity' feature in the dataset for further analysis.

In [24]:
```
"""
Integrating subjects' data (ID, Height, Weight, and Age) into the original dataset
"""

subject_data = [
    {'ID': 101, 'height': 182, 'weight': 83, 'age': 27},
    {'ID': 102, 'height': 169, 'weight': 78, 'age': 25},
    {'ID': 103, 'height': 187, 'weight': 92, 'age': 31},
    {'ID': 104, 'height': 194, 'weight': 95, 'age': 24},
    {'ID': 105, 'height': 180, 'weight': 73, 'age': 26},
    {'ID': 106, 'height': 183, 'weight': 69, 'age': 26},
    {'ID': 107, 'height': 173, 'weight': 86, 'age': 23},
    {'ID': 108, 'height': 179, 'weight': 87, 'age': 32},
    {'ID': 109, 'height': 168, 'weight': 65, 'age': 31}]

subject_df = pd.DataFrame(subject_data)

subject_df

df_final = pd.merge(df_final, subject_df, on='ID')
```

In [25]:
```
df_final
```

Out[25]:

	timestamp	ID	activityID	ankle_3D_acc_16g_x	ankle_3D_acc_16g_y	ankle_3D_acc_16g_z	ankle_3D_gyro_x	ankle_3D_gyro_y	ankle
0	37.0	101.0	1.0	9.749530	-1.923307	0.057149	0.007516	-0.026196	
1	38.0	101.0	1.0	9.666413	-1.887806	-0.093994	0.003784	-0.022030	
2	39.0	101.0	1.0	9.727602	-1.741162	-0.017098	0.021087	0.016340	
3	40.0	101.0	1.0	9.746160	-1.126264	0.094616	-0.005636	-0.016094	
4	41.0	101.0	1.0	9.805887	-1.012875	0.112105	-0.000508	-0.008577	
...	...	...	...	...	...	...	...	...	
27342	1932.0	109.0	20.0	9.469699	-0.558191	-3.012414	-0.000410	-0.025951	
27343	1933.0	109.0	20.0	9.422736	-0.726419	-3.169429	-0.024273	-0.000921	
27344	1934.0	109.0	20.0	9.415436	-0.412814	-3.338671	0.000428	-0.001009	
27345	1935.0	109.0	20.0	9.381788	-0.180110	-3.396201	0.009619	-0.006896	
27346	1936.0	109.0	20.0	9.419690	-0.487734	-3.466790	-0.026614	-0.022865	

27347 rows × 46 columns

In [26]:
```
"""
Mapping the activity names based on activityID and creating a new column
"""

activities = {0:'transient', 1:'lying', 2:'sitting', 3:'standing',
              4:'walking', 5:'running', 6:'cycling', 7:'nordic walking',
              9:'watching TV', 10:'computer work', 11:'car driving',
              12:'ascending stairs', 13:'descending stairs', 16:'vacuum cleaning',
              17:'ironing', 18:'folding laundry', 19:'house cleaning',
              20:'playing soccer', 24:'rope jumping'}

df_final['activity'] = df_final['activityID'].map(activities)
```

In [27]:
```
"""
Saving the dataframe for further data analysis
"""

df_final.to_csv('total_data.csv', index=False)
```

Bootstrapping for Data Balancing

The data team identified the imbalance in the distribution of data points among activities beforehand. Hence, we determined to bootstrap certain combinations of activities and subjects with fewer than 500 data points. Even if the sampled data is created arbitrarily, the application of the bootstrap method helps ensure a more representative and reliable dataset for our analysis.

Bootstrapping is often utilised when a dataset does not follow a normal distribution and has a limited number of data points.

Furthermore, combinations with fewer than 50 data points were excluded, as deriving meaningful insights from such a small number of data points is challenging. Therefore, the combination of 'activityID 24 and subject 106' and 'activityID 5 and subject 107' have been excluded.

In [28]:
```
df_final2 = pd.read_csv('total_data.csv')
```

In [29]:
```
"""
Detecting combinations with extremely small number of data points (fewer than 50 data points)
"""

timestamp_df = df_final2.copy(deep=True)

timestamp_df = timestamp_df.groupby(['activityID', 'ID'])['timestamp'].apply(list).reset_index()

activities = timestamp_df.activityID.unique()
subjects = timestamp_df.ID.unique()

print("Activitiy and Subject combinations (for bootstrap)")
print("")
for activity in activities:
    ts_1 = timestamp_df[timestamp_df['activityID'] == activity]
    for subject in subjects:
        ts_2 = ts_1[ts_1['ID'] == subject]
        if not ts_2.empty:
            num_samples = len(list(ts_2['timestamp'])[0])
            if num_samples < 50:
                print(f"Activity ID : {activity} | Subject : {subject} has to be deleted ({num_samples} data po
```

```
Activitiy and Subject combinations (for bootstrap)

Activity ID : 5.0 | Subject : 107.0 has to be deleted (37 data points)
Activity ID : 24.0 | Subject : 106.0 has to be deleted (4 data points)
```

In [30]:
```
"""
Removing the combinations with extremely small number of data points (fewer than 50 data points)
"""

delete_df = df_final2[(df_final2['activityID']==24) & (df_final2['ID']==106)]
delete_index = delete_df.index
df_final2.drop(index=delete_index, inplace = True)

delete_df2 = df_final2[(df_final2['activityID']==5) & (df_final2['ID']==107)]
delete_index2 = delete_df2.index
df_final2.drop(index=delete_index2, inplace = True)

df_final2.reset_index(drop= True, inplace=True)
```

The data team identified the imbalance in the distribution of data points among activities beforehand. Hence, we determined to bootstrap certain combinations of activities and subjects with fewer than 500 data points. Even if the sampled data is created arbitrarily, the application of the bootstrap method helps ensure a more representative and reliable dataset for our analysis.

Bootstrapping is often utilised when a dataset does not follow a normal distribution and has a limited number of data points.

Furthermore, combinations with fewer than 50 data points were excluded. Even if we utilise bootstrapping, it is hard to derive meaningful insights from such a small number of data points is challenging. Therefore, the combination of 'activityID 24 and subject 106' and 'activityID 5 and subject 107' have been excluded.

In [31]:
```
"""
Defining a function called 'split_data' to split the dataset into two sets (for training and testing)
-> Using train_test_split library from sklearn
"""

from sklearn.model_selection import train_test_split

def split_data(df_origin):
    df = df_origin.copy()
    # Test size : 30%, Training size : 70%
    df_train, df_test = train_test_split(df, test_size=0.3, random_state=42)
    df_train.sort_values(by=['ID','timestamp'], ascending=True, inplace = True)
    df_train = df_train.reset_index().drop(columns='index')
    df_test.sort_values(by=['ID','timestamp'], ascending=True, inplace = True)
    df_test = df_test.reset_index().drop(columns='index')
    return df_train, df_test
```

In [32]:
```
"""
Defining a function called 'bootstrap' to resample the dataset (Original codes from James, revised by Jungyeon
"""

def bootstrap(df, bootstrap_size=500):
    np.random.seed(41)
    groups_to_bootstrap = df.groupby(['activityID', 'ID']).filter(lambda x: len(x) < bootstrap_size)


    bootstrap_sample = groups_to_bootstrap.groupby(['activityID', 'ID']).apply(lambda x: x.sample(bootstrap_siz
```

```
    bootstrap_sample.reset_index(drop= True, inplace=True)


    non_bootstrap_sample = df.groupby(['activityID', 'ID']).filter(lambda x: len(x) >= bootstrap_size)
    non_bootstrap_sample.reset_index(drop= True, inplace=True)

    bootstrap_df = pd.concat([non_bootstrap_sample, bootstrap_sample])
    bootstrap_df.reset_index(drop= True, inplace=True)
    return bootstrap_df
```

In [33]:
```
"""
Splitting the dataset into two sets
"""

df_train, df_test = split_data(df_final2)
```

In [34]:
```
"""
Checking which combinations need to be bootstrapped
"""

activities = df_train.activityID.unique()
subjects = df_train.ID.unique()

for activity in activities:
    ts_1 = df_train[df_train['activityID'] == activity]
    for subject in subjects:
        ts_2 = ts_1[ts_1['ID'] == subject]
        if not ts_2.empty:
            num_samples = ts_2.shape[0]
            # if a combination has less than 500 data points, it needs to be bootstrapped
            if num_samples < 500:
                print(f"Activity ID : {activity} | Subject : {subject} has to be bootstrapped ({num_samples})")
```

```
Activity ID : 1.0 | Subject : 101.0 has to be bootstrapped (203)
Activity ID : 1.0 | Subject : 102.0 has to be bootstrapped (176)
Activity ID : 1.0 | Subject : 103.0 has to be bootstrapped (156)
Activity ID : 1.0 | Subject : 104.0 has to be bootstrapped (159)
Activity ID : 1.0 | Subject : 105.0 has to be bootstrapped (168)
Activity ID : 1.0 | Subject : 106.0 has to be bootstrapped (167)
Activity ID : 1.0 | Subject : 107.0 has to be bootstrapped (176)
Activity ID : 1.0 | Subject : 108.0 has to be bootstrapped (163)
Activity ID : 11.0 | Subject : 101.0 has to be bootstrapped (386)
Activity ID : 2.0 | Subject : 101.0 has to be bootstrapped (165)
Activity ID : 2.0 | Subject : 102.0 has to be bootstrapped (154)
Activity ID : 2.0 | Subject : 103.0 has to be bootstrapped (192)
Activity ID : 2.0 | Subject : 104.0 has to be bootstrapped (176)
Activity ID : 2.0 | Subject : 105.0 has to be bootstrapped (183)
Activity ID : 2.0 | Subject : 106.0 has to be bootstrapped (152)
Activity ID : 2.0 | Subject : 107.0 has to be bootstrapped (84)
Activity ID : 2.0 | Subject : 108.0 has to be bootstrapped (157)
Activity ID : 3.0 | Subject : 101.0 has to be bootstrapped (159)
Activity ID : 3.0 | Subject : 102.0 has to be bootstrapped (185)
Activity ID : 3.0 | Subject : 103.0 has to be bootstrapped (140)
Activity ID : 3.0 | Subject : 104.0 has to be bootstrapped (174)
Activity ID : 3.0 | Subject : 105.0 has to be bootstrapped (158)
Activity ID : 3.0 | Subject : 106.0 has to be bootstrapped (169)
Activity ID : 3.0 | Subject : 107.0 has to be bootstrapped (197)
Activity ID : 3.0 | Subject : 108.0 has to be bootstrapped (171)
Activity ID : 17.0 | Subject : 101.0 has to be bootstrapped (157)
Activity ID : 17.0 | Subject : 102.0 has to be bootstrapped (203)
Activity ID : 17.0 | Subject : 103.0 has to be bootstrapped (185)
Activity ID : 17.0 | Subject : 104.0 has to be bootstrapped (176)
Activity ID : 17.0 | Subject : 105.0 has to be bootstrapped (240)
Activity ID : 17.0 | Subject : 106.0 has to be bootstrapped (289)
Activity ID : 17.0 | Subject : 107.0 has to be bootstrapped (214)
Activity ID : 17.0 | Subject : 108.0 has to be bootstrapped (226)
Activity ID : 16.0 | Subject : 101.0 has to be bootstrapped (169)
Activity ID : 16.0 | Subject : 102.0 has to be bootstrapped (157)
Activity ID : 16.0 | Subject : 103.0 has to be bootstrapped (137)
Activity ID : 16.0 | Subject : 104.0 has to be bootstrapped (138)
Activity ID : 16.0 | Subject : 105.0 has to be bootstrapped (168)
Activity ID : 16.0 | Subject : 106.0 has to be bootstrapped (142)
Activity ID : 16.0 | Subject : 107.0 has to be bootstrapped (162)
Activity ID : 16.0 | Subject : 108.0 has to be bootstrapped (178)
Activity ID : 12.0 | Subject : 101.0 has to be bootstrapped (118)
Activity ID : 12.0 | Subject : 102.0 has to be bootstrapped (131)
Activity ID : 12.0 | Subject : 103.0 has to be bootstrapped (74)
Activity ID : 12.0 | Subject : 104.0 has to be bootstrapped (112)
Activity ID : 12.0 | Subject : 105.0 has to be bootstrapped (100)
Activity ID : 12.0 | Subject : 106.0 has to be bootstrapped (103)
Activity ID : 12.0 | Subject : 107.0 has to be bootstrapped (113)
Activity ID : 12.0 | Subject : 108.0 has to be bootstrapped (74)
Activity ID : 13.0 | Subject : 101.0 has to be bootstrapped (97)
Activity ID : 13.0 | Subject : 102.0 has to be bootstrapped (103)
Activity ID : 13.0 | Subject : 103.0 has to be bootstrapped (116)
Activity ID : 13.0 | Subject : 104.0 has to be bootstrapped (104)
Activity ID : 13.0 | Subject : 105.0 has to be bootstrapped (86)
Activity ID : 13.0 | Subject : 106.0 has to be bootstrapped (83)
```

```
Activity ID : 13.0 | Subject : 107.0 has to be bootstrapped (87)
Activity ID : 13.0 | Subject : 108.0 has to be bootstrapped (71)
Activity ID : 19.0 | Subject : 101.0 has to be bootstrapped (372)
Activity ID : 19.0 | Subject : 105.0 has to be bootstrapped (206)
Activity ID : 19.0 | Subject : 106.0 has to be bootstrapped (212)
Activity ID : 19.0 | Subject : 108.0 has to be bootstrapped (288)
Activity ID : 19.0 | Subject : 109.0 has to be bootstrapped (231)
Activity ID : 4.0 | Subject : 101.0 has to be bootstrapped (140)
Activity ID : 4.0 | Subject : 102.0 has to be bootstrapped (218)
Activity ID : 4.0 | Subject : 103.0 has to be bootstrapped (201)
Activity ID : 4.0 | Subject : 104.0 has to be bootstrapped (231)
Activity ID : 4.0 | Subject : 105.0 has to be bootstrapped (218)
Activity ID : 4.0 | Subject : 106.0 has to be bootstrapped (175)
Activity ID : 4.0 | Subject : 107.0 has to be bootstrapped (232)
Activity ID : 4.0 | Subject : 108.0 has to be bootstrapped (233)
Activity ID : 7.0 | Subject : 101.0 has to be bootstrapped (140)
Activity ID : 7.0 | Subject : 102.0 has to be bootstrapped (206)
Activity ID : 7.0 | Subject : 104.0 has to be bootstrapped (188)
Activity ID : 7.0 | Subject : 105.0 has to be bootstrapped (186)
Activity ID : 7.0 | Subject : 106.0 has to be bootstrapped (185)
Activity ID : 7.0 | Subject : 107.0 has to be bootstrapped (205)
Activity ID : 7.0 | Subject : 108.0 has to be bootstrapped (203)
Activity ID : 6.0 | Subject : 101.0 has to be bootstrapped (164)
Activity ID : 6.0 | Subject : 102.0 has to be bootstrapped (180)
Activity ID : 6.0 | Subject : 104.0 has to be bootstrapped (151)
Activity ID : 6.0 | Subject : 105.0 has to be bootstrapped (176)
Activity ID : 6.0 | Subject : 106.0 has to be bootstrapped (140)
Activity ID : 6.0 | Subject : 107.0 has to be bootstrapped (145)
Activity ID : 6.0 | Subject : 108.0 has to be bootstrapped (187)
Activity ID : 18.0 | Subject : 101.0 has to be bootstrapped (193)
Activity ID : 18.0 | Subject : 106.0 has to be bootstrapped (162)
Activity ID : 18.0 | Subject : 108.0 has to be bootstrapped (167)
Activity ID : 18.0 | Subject : 109.0 has to be bootstrapped (188)
Activity ID : 5.0 | Subject : 101.0 has to be bootstrapped (147)
Activity ID : 5.0 | Subject : 102.0 has to be bootstrapped (76)
Activity ID : 5.0 | Subject : 105.0 has to be bootstrapped (173)
Activity ID : 5.0 | Subject : 106.0 has to be bootstrapped (170)
Activity ID : 5.0 | Subject : 108.0 has to be bootstrapped (106)
Activity ID : 24.0 | Subject : 101.0 has to be bootstrapped (92)
Activity ID : 24.0 | Subject : 102.0 has to be bootstrapped (93)
Activity ID : 24.0 | Subject : 105.0 has to be bootstrapped (53)
Activity ID : 24.0 | Subject : 108.0 has to be bootstrapped (57)
Activity ID : 24.0 | Subject : 109.0 has to be bootstrapped (46)
Activity ID : 10.0 | Subject : 106.0 has to be bootstrapped (433)
Activity ID : 10.0 | Subject : 108.0 has to be bootstrapped (475)
Activity ID : 10.0 | Subject : 109.0 has to be bootstrapped (498)
Activity ID : 20.0 | Subject : 108.0 has to be bootstrapped (129)
Activity ID : 20.0 | Subject : 109.0 has to be bootstrapped (209)
```

In [35]:
```
"""
Conducting the bootstrapping
"""

df_boot = bootstrap(df_train)
df_boot.drop(columns='timestamp', inplace = True)
df_boot.sort_values(by= ['ID', 'activityID'], inplace = True)
df_boot.reset_index(drop= True, inplace=True)
```

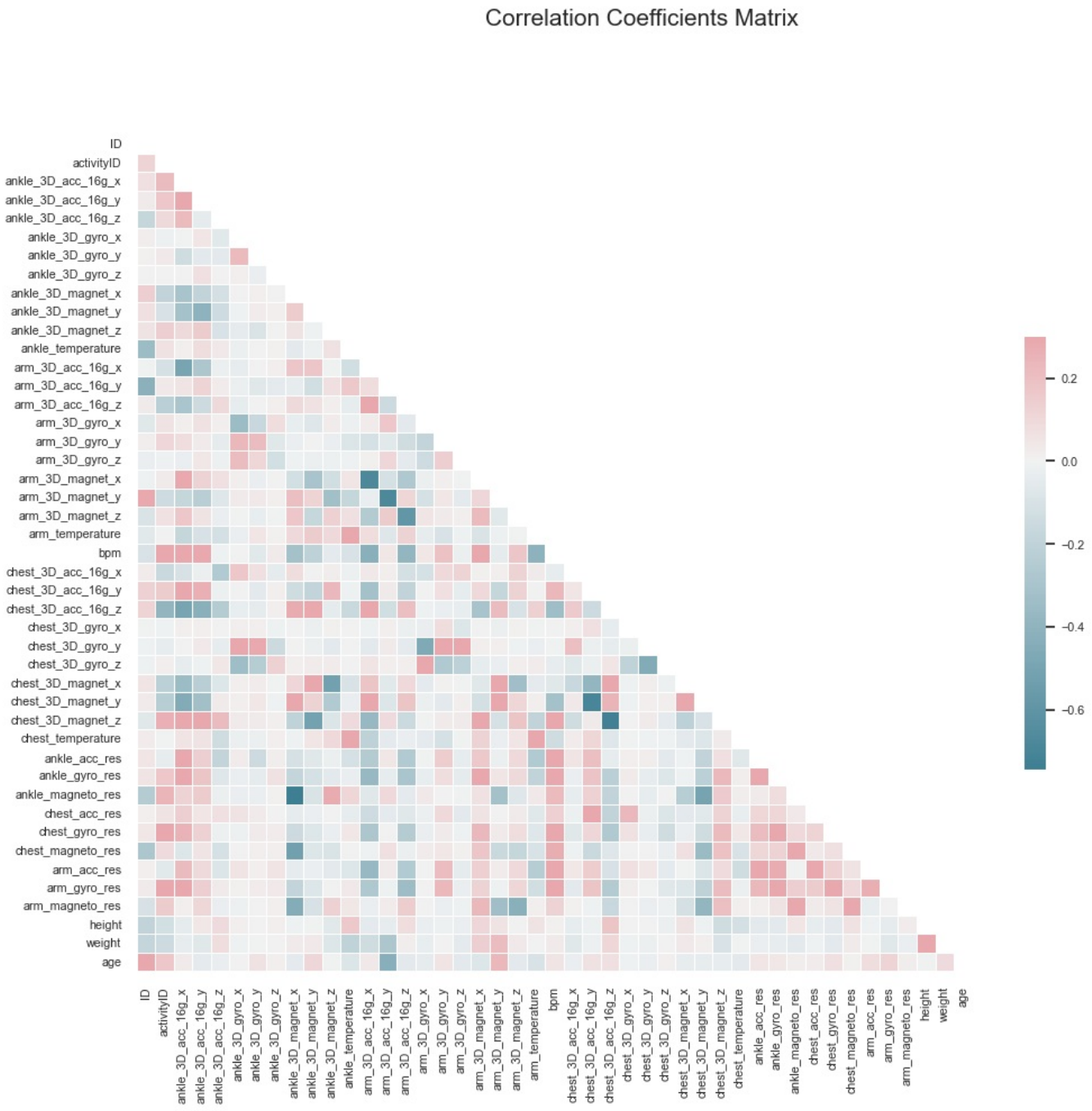
In [36]:
```
"""
Checking one more time if there is any combination with fewer than 500 data points
"""

activities = df_boot.activityID.unique()
subjects = df_boot.ID.unique()
b_count = 0

for activity in activities:
    ts_1 = df_boot[df_boot['activityID'] == activity]
    for subject in subjects:
        ts_2 = ts_1[ts_1['ID'] == subject]
        if not ts_2.empty:
            num_samples = ts_2.shape[0]
            if num_samples < 500:
                b_count += 1
                print(f"Activity ID : {activity} | Subject : {subject} = Number of data points : {num_samples}"

if b_count == 0:
    print('Every combination has sufficient number of data')
```

```
Every combination has sufficient number of data
```

Afterwards, we created a correlation matrix to identify relationships among the features. High positive or negative correlations between two features can adversely affect the results and outputs of data analysis and mathematical models due to multicollinearity. Therefore, we attempted to assess these correlations through the correlation matrix in order to diminish the multicollinearity in the dataset.

In [37]:
```
"""
Creating Correlation Coefficients Matrix (Source from Udemy course : Pandas Bootcamp)
```

```
"""

sns.set(style='white',font_scale = 0.7)
corr = df_boot.corr(numeric_only=True)
mask = np.zeros_like(corr, dtype = np.bool_)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize = (12,10))
f.suptitle('Correlation Coefficients Matrix', fontsize = 15)
cmap = sns.diverging_palette(220,10, as_cmap = True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, square=True,linewidths=.5,cbar_kws={'shrink':.5},yti
plt.show()
```



To assess multicollinearity, we examined correlations exceeding a coefficient of 0.7 or -0.7. In cases where two features exhibited high correlation, a decision was made to exclude one feature from the dataset.

As a consequence, we observed a strong correlation between certain features, and in order to address multicollinearity issues, we have decided to arbitrarily eliminate one feature from each correlated pair when constructing a mathematical model. The correlated pairs with a correlation coefficient of 0.7 or higher are as follows:

1. ankle_3D_gyro_x and chest_3D_gyro_y
2. ankle_3D_magnet_x and ankle_magneto_res
3. arm_temperature and chest_temperature
4. chest_3D_acc_16g_z and chest_3D_magnet_z
5. ankle_gyro_res and chest_gyro_res
6. chest_magneto_res and arm_magneto_res

To enhance the overall predictive capability of the model and address potential collinearity issues, we have opted to remove specific features from correlated pairs within these sets. Specifically, for the model based on arm and chest units, we will exclude

'chest_temperature' and 'chest_magneto_res'. Additionally, 'chest_3D_magnet_z' will be deleted from the model associated with the chest unit. Furthermore, features 'ankle_3D_gyro_x' and 'ankle_gyro_res' will be eliminated from the model utilising data from the ankle and chest units. Lastly, the feature 'ankle_3D_magnet_x' will be removed from the model incorporating data from the ankle unit.

In [38]:
```
"""
Identifying correlated pairs with a correlation coefficient of 0.7 or higher (Original Source from Udemy course
"""

high_corr = corr[abs(corr) > 0.7]

# Setting the diagonal and lower triangular part to NaN to prevent duplicate output
high_corr = high_corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))

high_corr_pairs = high_corr.stack().reset_index()
high_corr_pairs.columns = ['Variable1', 'Variable2', 'Correlation']

# Selecting only the non-NaN portions of the correlation
high_corr_pairs = high_corr_pairs.dropna(subset=['Correlation'])

print(high_corr_pairs)
```

```
            Variable1          Variable2  Correlation
0     ankle_3D_gyro_x    chest_3D_gyro_y     0.726641
1   ankle_3D_magnet_x  ankle_magneto_res    -0.744996
2     arm_temperature  chest_temperature     0.772046
3  chest_3D_acc_16g_z  chest_3D_magnet_z    -0.727175
4      ankle_gyro_res     chest_gyro_res     0.712262
5   chest_magneto_res    arm_magneto_res     0.709899
```

Exploratory Data Analysis

Before constructing a mathematical model, we conducted Exploratory Data Analysis (EDA) in order to assess the importance of certain features.

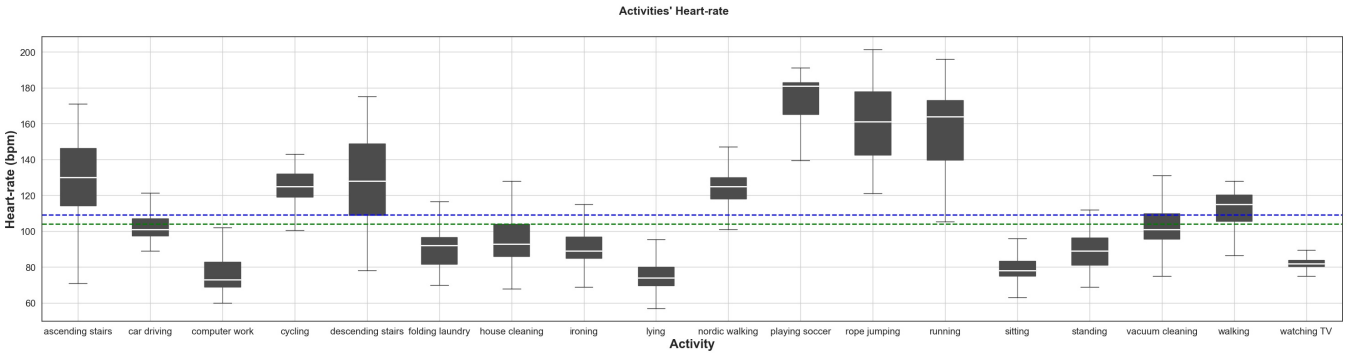
First of all, the data team identified the Heart rate (BPM) variable as crucial for the mathematical model. In static activities like car driving, computer work, folding laundry, house cleaning, ironing, lying, etc., the median* BPM is below the average**. However, in activities that lie between static and active, such as vacuum cleaning and walking, the median BPM slightly overlaps with the average. Moreover, for active activities such as ascending stairs, cycling, nordic walking, playing soccer, rope jumping and so on, the median BPM is greater than the average.

* Median is the green line.

** Average is the blue line.

Thus, it is imperative to include the 'Heart rate (BPM)' variable when building the mathematical model.

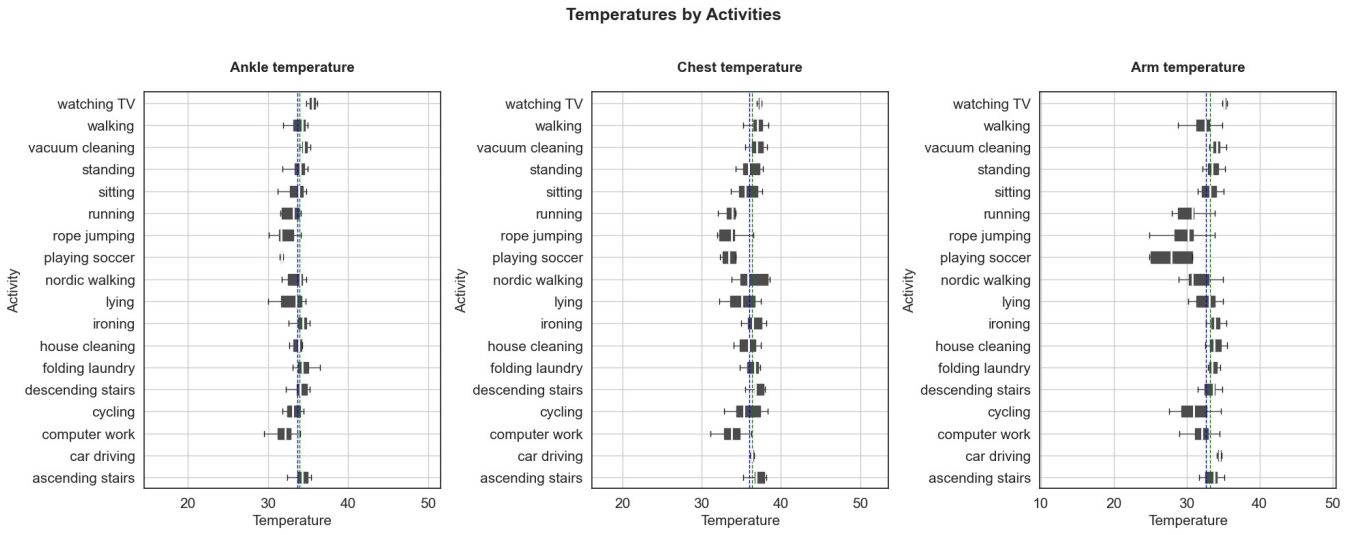
In [39]:
```
"""
Creating a boxplot to assess the statistical significance of BPM data
"""

fig, ax = plt.subplots(figsize = (30,8))

plt.rcParams.update({'font.size': 20})

middle = df_boot['bpm'].median()
average = df_boot['bpm'].mean()

plot = df_boot.boxplot(column = 'bpm', ax=ax, by='activity', patch_artist = True, fontsize = 15, showfliers = F
                       medianprops = {'linestyle':'-', 'linewidth':2, 'color':'white'})
plot.axhline(y=middle, color= 'green', linewidth=2, linestyle = '--')
plot.axhline(y=average, color= 'blue', linewidth=2, linestyle = '--')
ax.xaxis.set_tick_params(labelsize=15)
ax.yaxis.set_tick_params(labelsize=15)

ax.set_xlabel('Activity',fontsize = 20, fontweight = 'bold')
ax.set_ylabel('Heart-rate (bpm)',fontsize = 20, fontweight = 'bold')

ax.set_title('')
fig.suptitle("Activities' Heart-rate", size = 18, fontweight = 'bold')
fig.tight_layout()
```



Secondly, we concluded that the Temperature features from each part do not provide meaningful information. These variables are significantly influenced by factors such as the clothing worn by subjects and environmental conditions, including the weather. Hence, it is better to not take into account these features for our further analysis and modelling.

In [40]:
```
"""
Defining a function called 'plot_activity' in order to draw a boxplot
-> Original codes from https://www.kaggle.com/code/gm472sussex/physical-activity-prediction (5. EDA), revised b
"""

def plot_activity(df_boot, columns, x_label):
    for index, col in enumerate(columns):
        df_boot.boxplot(column = col, ax=ax[index], by='activity', vert = False,patch_artist = True, fontsize =
                        medianprops = {'linestyle':'-', 'linewidth':2, 'color':'white'})
        min_value = df_boot[col].min() - 15
        max_value = df_boot[col].max() + 15
        x_lim = (min_value, max_value)
        middle = df_boot[col].median()
        average = df_boot[col].mean()
        ax[index].axvline(x=middle, color= 'green', linewidth=1,linestyle = '--')
        ax[index].axvline(x=average, color= 'blue', linewidth=1,linestyle = '--')
        ax[index].set_xlabel(x_label,fontsize = 15)
        ax[index].set_ylabel('Activity',fontsize = 15)
        ax[index].set_xlim(x_lim)
```

In [50]:
```
"""
Creating a boxplot to evaluate the statistical significance of the temperature features
"""

fig, ax = plt.subplots(1,3, figsize = (20,8))

plt.rcParams.update({'font.size': 15})

columns = ['ankle_temperature','chest_temperature','arm_temperature']
plot_activity(df_boot, columns, x_label='Temperature')

ax[0].set_title('Ankle temperature\n', size = 15, fontweight = 'bold')
ax[1].set_title('Chest temperature\n', size = 15, fontweight = 'bold')
ax[2].set_title('Arm temperature\n', size = 15, fontweight = 'bold')
fig.suptitle("Temperatures by Activities\n", size = 18, fontweight = 'bold')
fig.tight_layout()
```



On the other hand, a resultant contains information from the x-y-z axes, so we determined to examine whether meaningful statistical features are exhibited in each resultant.

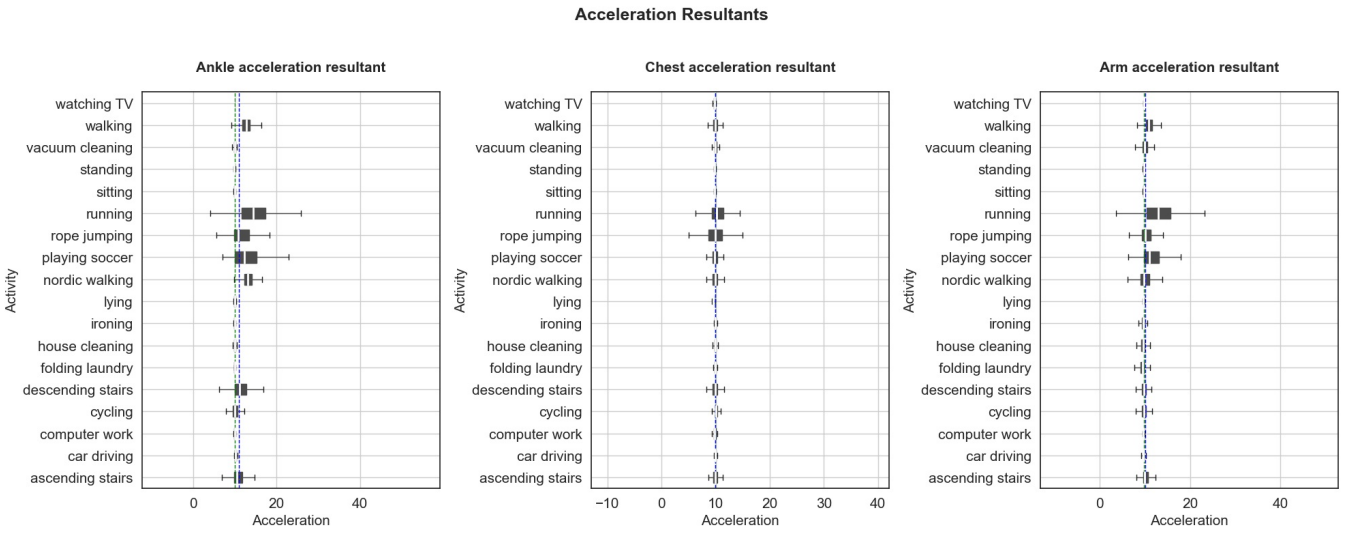
In the acceleration resultant features from each part, we have observed that the data barely provides crucial insights. This is due to the median values for each activity being very closely aligned, a consistent trend observed across all units.

Likewise, the resultant data from the 3-axis gyroscope sensors lacks significant insights. The similarity in median values for each activity

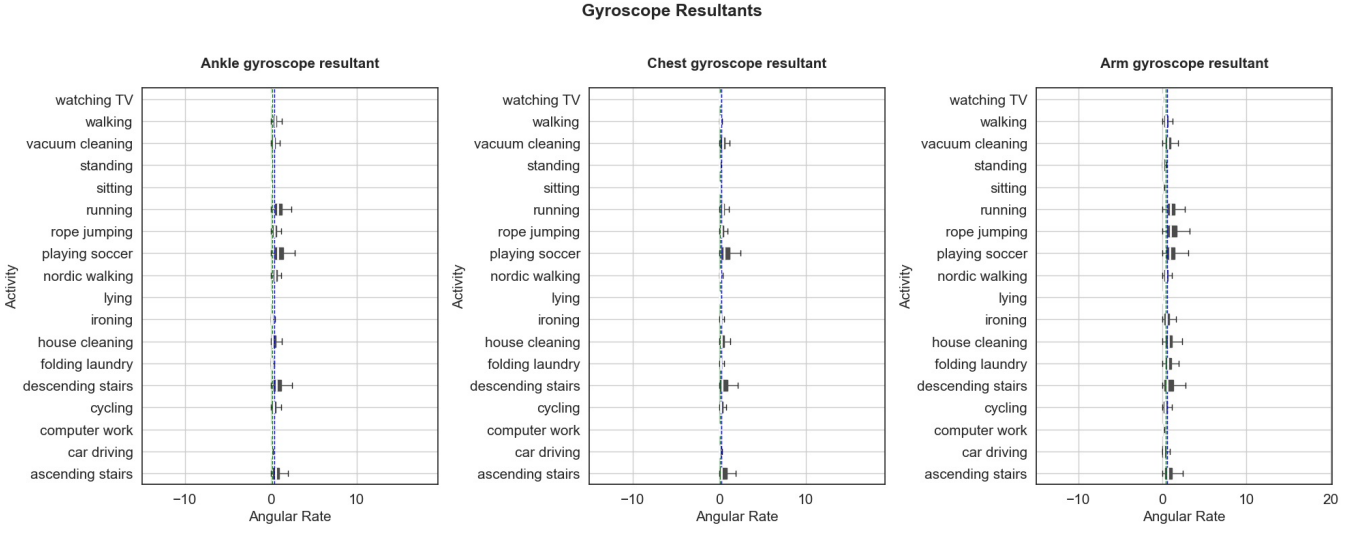
is a consistent pattern observed across all units.

Therefore, the acceleration and gyroscope resultant data will be excluded when creating the model.

In [48]:
```
"""
Creating a boxplot to assess the statistical significance of the acceleration resultant features
"""

fig, ax = plt.subplots(1,3, figsize = (20,8))

plt.rcParams.update({'font.size': 15})

columns = ['ankle_acc_res','chest_acc_res','arm_acc_res']
plot_activity(df_boot, columns, x_label='Acceleration')

ax[0].set_title('Ankle acceleration resultant\n', size = 15, fontweight = 'bold')
ax[1].set_title('Chest acceleration resultant\n', size = 15, fontweight = 'bold')
ax[2].set_title('Arm acceleration resultant\n', size = 15, fontweight = 'bold')
fig.suptitle('Acceleration Resultants\n', size = 18, fontweight = 'bold')
fig.tight_layout()
```



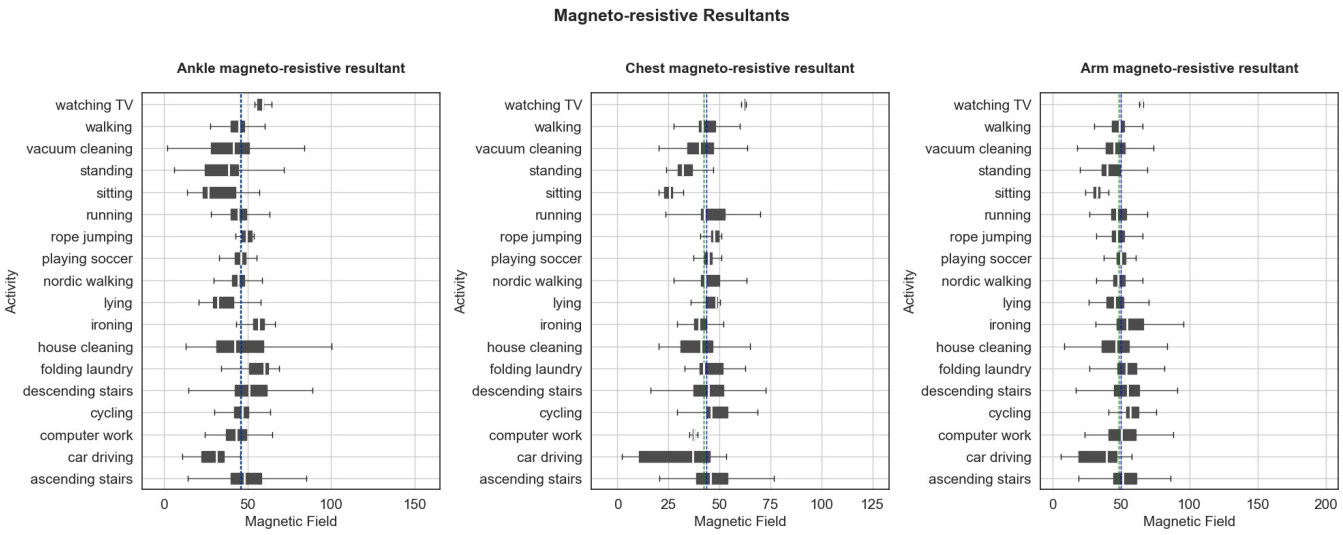
In [47]:
```
"""
Creating a boxplot to assess the statistical significance of the gyroscope resultant features
"""

fig, ax = plt.subplots(1,3, figsize = (20,8))

plt.rcParams.update({'font.size': 15})

columns = ['ankle_gyro_res','chest_gyro_res','arm_gyro_res']
plot_activity(df_boot, columns, x_label='Angular Rate')

ax[0].set_title('Ankle gyroscope resultant\n', size = 15, fontweight = 'bold')
ax[1].set_title('Chest gyroscope resultant\n', size = 15, fontweight = 'bold')
ax[2].set_title('Arm gyroscope resultant\n', size = 15, fontweight = 'bold')
fig.suptitle('Gyroscope Resultants\n', size = 18, fontweight = 'bold')
fig.tight_layout()
```



Meanwhile, the magneto-resistive resultant exhibits more distinct variations between each activity compared to other resultant data. This is because the median values exhibit a wider distribution within each unit.

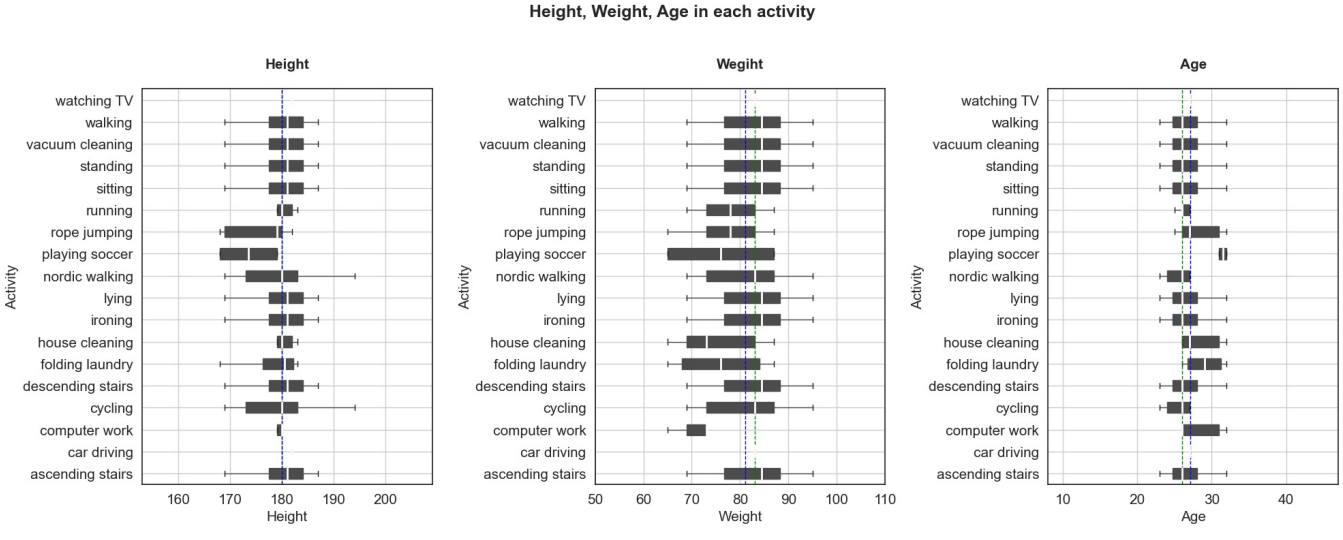
In [46]:
```
"""
```

```
Creating a boxplot to assess the statistical significance of the magneto-resistive resultant features
"""

fig, ax = plt.subplots(1,3, figsize = (20,8))

plt.rcParams.update({'font.size': 15})

columns = ['ankle_magneto_res','chest_magneto_res','arm_magneto_res']
plot_activity(df_boot, columns, x_label='Magnetic Field')

ax[0].set_title('Ankle magneto-resistive resultant\n', size = 15, fontweight = 'bold')
ax[1].set_title('Chest magneto-resistive resultant\n', size = 15, fontweight = 'bold')
ax[2].set_title('Arm magneto-resistive resultant\n', size = 15, fontweight = 'bold')
fig.suptitle('Magneto-resistive Resultants\n', size = 18, fontweight = 'bold')
fig.tight_layout()
```



Last but not least, the data team also examined the three features extracted from the subject information. Due to the limited number of initial samples (only 9), we were unable to obtain diverse values for Height, Weight, and Age variables. Regarding height and age, excluding 1 or 2 activities, most activities' median values align closely with the overall median and mean values. Thus, we have decided not to consider these two features due to the insufficient possibility to derive meaningful insights. However, the median values of the weight feature are Moderately distinguishable in each activity, so we retained this feature in the dataset.

In [51]:
```
"""
Creating a boxplot to evaluate the statistical significance of the height, weight, and age features
"""

fig, ax = plt.subplots(1,3, figsize = (20,8))

plt.rcParams.update({'font.size': 15})

columns = ['height','weight','age']
plot_activity(df_boot, columns, x_label='x')

ax[0].set_xlabel('Height',fontsize = 15)
ax[1].set_xlabel('Weight',fontsize = 15)
ax[2].set_xlabel('Age',fontsize = 15)
ax[0].set_title('Height\n', size = 15, fontweight = 'bold')
ax[1].set_title('Wegiht\n', size = 15, fontweight = 'bold')
ax[2].set_title('Age\n', size = 15, fontweight = 'bold')
fig.suptitle('Height, Weight, Age in each activity\n', size = 18, fontweight = 'bold')
fig.tight_layout()
```



Methodology for Mathematical Model

In developing a mathematical model, we considered two techniques called linear regression and clustering. Linear regression, a statistical method, is employed to predict a continuous dependent variable based on its linear relationship with one or more independent variables. In contrast, clustering is an unsupervised learning method that involves grouping similar data points. The aim is to identify patterns within the dataset by forming clusters where data points within the same cluster share similarities.

Given the primary objective of classifying the activity a user has engaged in, so we evaluated that linear regression may not be suitable for this specific task. Consequently, we decided to choose the clustering method as the foundation for our mathematical model.

In [52]:
```
"""
Creating a DataFrame for all subsets of the arm, ankle, and chest unit combinations
-> only using the features that are not correlated to each other and not excluded by exploratory data analysis
"""

y = df_boot['activityID']

subject = df_boot['ID']

arm = df_boot[['bpm', 'arm_3D_acc_16g_x',
       'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z', 'arm_3D_gyro_x',
       'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x', 'arm_3D_magnet_y',
       'arm_3D_magnet_z','arm_magneto_res','weight']].copy(deep=True)

chest = df_boot[['bpm', 'chest_3D_acc_16g_x',
       'chest_3D_acc_16g_y', 'chest_3D_acc_16g_z', 'chest_3D_gyro_x',
       'chest_3D_gyro_y', 'chest_3D_gyro_z', 'chest_3D_magnet_x',
       'chest_3D_magnet_y', 'chest_magneto_res','weight']].copy(deep=True)

ankle = df_boot[['bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','weight']].copy(deep=Tru

arm_chest = df_boot[['bpm', 'chest_3D_acc_16g_x',
       'chest_3D_acc_16g_y', 'chest_3D_acc_16g_z', 'chest_3D_gyro_x',
       'chest_3D_gyro_y', 'chest_3D_gyro_z', 'chest_3D_magnet_x',
       'chest_3D_magnet_y', 'arm_3D_acc_16g_x',
       'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z', 'arm_3D_gyro_x',
       'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x', 'arm_3D_magnet_y',
       'arm_3D_magnet_z','arm_magneto_res', 'weight']].copy(deep=True)

arm_ankle = df_boot[['bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','arm_3D_acc_16g_x',
       'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z', 'arm_3D_gyro_x',
       'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x', 'arm_3D_magnet_y',
       'arm_3D_magnet_z','arm_magneto_res','weight']].copy(deep=True)

chest_ankle = df_boot[['bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res',
       'chest_3D_acc_16g_x','chest_3D_acc_16g_y', 'chest_3D_acc_16g_z', 'chest_3D_gyro_x',
       'chest_3D_gyro_y', 'chest_3D_gyro_z', 'chest_3D_magnet_x',
       'chest_3D_magnet_y', 'chest_magneto_res','weight']].copy(deep=True)

every_part = df_boot[['bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','arm_3D_acc_16g_x',
       'arm_3D_acc_16g_y', 'arm_3D_acc_16g_z', 'arm_3D_gyro_x',
       'arm_3D_gyro_y', 'arm_3D_gyro_z', 'arm_3D_magnet_x', 'arm_3D_magnet_y',
       'arm_3D_magnet_z','arm_magneto_res','chest_3D_acc_16g_x',
       'chest_3D_acc_16g_y', 'chest_3D_acc_16g_z', 'chest_3D_gyro_x',
       'chest_3D_gyro_y', 'chest_3D_gyro_z', 'chest_3D_magnet_x',
       'chest_3D_magnet_y', 'weight']].copy(deep=True)
```

Initially, it is essential to find the best combination of sensors. During data collection, sensors were strategically placed at the ankle, chest, and arm. Subsequently, we assessed the entire subset of the combination of these sensors using two clustering methods: K-Means and Gaussian Mixture. In the subset datasets, we have selected only features that are not correlated to each other and not excluded by the exploratory data analysis process. In addition, we chose to use 'normalisation' as a result of the previous normality test.

K-Means clustering is an unsupervised machine learning algorithm, that segments data into K clusters. The algorithm calculates the cluster centroids, assigns each data point to the nearest centroid, and iteratively updates the centroids, optimising the clusters. On the other hand, the Gaussian Mixture Model (GMM) is a probabilistic generative model that represents data as a mixture of multiple Gaussian distributions. It assumes that the data is generated by a combination of these Gaussian clusters, each characterised by its mean and covariance.

In [53]:
```
"""
1. Importing libraries (Kmeans, Normalize, GaussianMixture)
2. Normalising the dataset
3. Creating KMeans clustering models
4. Creating Gaussian Mixture clustering models
"""
```

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import normalize
from sklearn.mixture import GaussianMixture

clusters = df_boot['activityID'].nunique()

# KMeans clustering
kmeans = KMeans(n_clusters=clusters, random_state=42)

arm_data = normalize(arm.values, norm='l2')
arm['cluster'] = kmeans.fit_predict(arm_data)

chest_data = normalize(chest.values, norm='l2')
chest['cluster'] = kmeans.fit_predict(chest_data)

ankle_data = normalize(ankle.values, norm='l2')
ankle['cluster'] = kmeans.fit_predict(ankle_data)

arm_chest_data = normalize(arm_chest.values, norm='l2')
arm_chest['cluster'] = kmeans.fit_predict(arm_chest_data)

arm_ankle_data = normalize(arm_ankle.values, norm='l2')
arm_ankle['cluster'] = kmeans.fit_predict(arm_ankle_data)

chest_ankle_data = normalize(chest_ankle.values, norm='l2')
chest_ankle['cluster'] = kmeans.fit_predict(chest_ankle_data)

every_part_data = normalize(every_part.values, norm='l2')
every_part['cluster'] = kmeans.fit_predict(every_part_data)

# GaussianMixture clustering
mixture = GaussianMixture(n_components=clusters,covariance_type="diag").fit(arm_data)
arm["cluster2"] = mixture.predict(arm_data)

mixture1 = GaussianMixture(n_components=clusters,covariance_type="diag").fit(chest_data)
chest["cluster2"] = mixture1.predict(chest_data)

mixture2 = GaussianMixture(n_components=clusters,covariance_type="diag").fit(ankle_data)
ankle["cluster2"] = mixture2.predict(ankle_data)

mixture = GaussianMixture(n_components=clusters,covariance_type="diag").fit(arm_chest_data)
arm_chest["cluster2"] = mixture.predict(arm_chest_data)

mixture = GaussianMixture(n_components=clusters,covariance_type="diag").fit(arm_ankle_data)
arm_ankle["cluster2"] = mixture.predict(arm_ankle_data)

mixture = GaussianMixture(n_components=clusters,covariance_type="diag").fit(chest_ankle_data)
chest_ankle["cluster2"] = mixture.predict(chest_ankle_data)

mixture = GaussianMixture(n_components=clusters,covariance_type="diag").fit(every_part_data)
every_part["cluster2"] = mixture.predict(every_part_data)
```

Following these analyses, the data team computed the silhouette score in order to evaluate each model. The silhouette score is an indicator to assess the quality of clustering. It takes into account both the similarity of data points within clusters and the dissimilarity between clusters. A high silhouette score indicates well-defined clusters, with values ranging from -1 to 1.

In [54]:
```
"""
Calculating the silhouette score to derive the best combination of units
"""

from sklearn.metrics import silhouette_score

silhouette_kmeans_arm = silhouette_score(arm_data, arm['cluster'])
silhouette_kmeans_chest = silhouette_score(chest_data, chest['cluster'])
silhouette_kmeans_ankle = silhouette_score(ankle_data, ankle['cluster'])
silhouette_kmeans_arm_chest = silhouette_score(arm_chest_data, arm_chest['cluster'])
silhouette_kmeans_arm_ankle = silhouette_score(arm_ankle_data, arm_ankle['cluster'])
silhouette_kmeans_chest_ankle = silhouette_score(chest_ankle_data, chest_ankle['cluster'])
silhouette_kmeans_every_part = silhouette_score(every_part_data, every_part['cluster'])

silhouette_GM_arm = silhouette_score(arm_data, arm['cluster2'])
silhouette_GM_chest = silhouette_score(chest_data, chest['cluster2'])
silhouette_GM_ankle = silhouette_score(ankle_data, ankle['cluster2'])
silhouette_GM_arm_chest = silhouette_score(arm_chest_data, arm_chest['cluster2'])
silhouette_GM_arm_ankle = silhouette_score(arm_ankle_data, arm_ankle['cluster2'])
silhouette_GM_chest_ankle = silhouette_score(chest_ankle_data, chest_ankle['cluster2'])
silhouette_GM_every_part = silhouette_score(every_part_data, every_part['cluster2'])
```

To choose the best combination, the highest silhouette score from each K-Means and Gaussian Mixture were selected. Then, the data team compared the two best scores to determine the optimal combination of sensors and clustering methodology.

The K-Means silhouette score reached its peak at 0.2593, derived from the ankle sensor data. In contrast, the Gaussian Mixture achieved its best score of 0.1101, from the data from the chest sensor. Based on this evaluation, the data team decided to solely use the data from the ankle sensor and K-Means clustering as the core of our mathematical model.

In [55]:
```
"""
1. Printing out the best score in KMeans Clustering
2. Printing out the best score in Gaussian Mixture Clustering
3. Comparing the two best scores and determining which model is the best
"""

for_select = {'Arm' : silhouette_kmeans_arm,
              'Chest' : silhouette_kmeans_chest,
              'Ankle' : silhouette_kmeans_ankle,
             'Arm_chest' : silhouette_kmeans_arm_chest,
              'Arm_ankle' : silhouette_kmeans_arm_ankle,
              'Chest_ankle' : silhouette_kmeans_chest_ankle,
             'Every_part' : silhouette_kmeans_every_part}

best_key = max(for_select, key=for_select.get)
best_value = for_select[best_key]
print(f"The best part for this item: {best_key} in K-means | Silhouette Score: {best_value:.4f}")

for_select2 = {'Arm' : silhouette_GM_arm,
              'Chest' : silhouette_GM_chest,
              'Ankle' : silhouette_GM_ankle,
             'Arm_chest' : silhouette_GM_arm_chest,
              'Arm_ankle' : silhouette_GM_arm_ankle,
              'Chest_ankle' : silhouette_GM_chest_ankle,
             'Every_part' : silhouette_GM_every_part}

best_key2 = max(for_select2, key=for_select2.get)
best_value2 = for_select2[best_key2]
print(f"The best part for this item: {best_key2} in Gaussian Mixture | Silhouette Score: {best_value2:.4f}")

if best_key == best_key2:
    print(f'Therefore, {best_key} is the best part for this product')
elif best_value > best_value2:
    print(f'{best_key} is the best part for this product becuase the Silhouette Score is higher!')
elif best_value < best_value2:
    print(f'{best_key2} is the best part for this product becuase the Silhouette Score is higher!')
```

```
The best part for this item: Ankle in K-means | Silhouette Score: 0.2593
The best part for this item: Chest in Gaussian Mixture | Silhouette Score: 0.1101
Ankle is the best part for this product becuase the Silhouette Score is higher!
```

Developing a Mathematical Model

In the initial phase of model development, we focused on feature selection to find the variables for the K-means clustering method. The chosen features were determined based on the insights gained from exploratory data analysis. Please refer to the selected features below.

- Selected features: BPM, Accelerometer_x-axis, Accelerometer_y-axis, Accelerometer_z-axis, Gyroscope_x-axis, Gyroscope_y-axis, Gyroscope_z-axis, Magneto-resistive_y-axis, Magneto-resistive_z-axis, Magnetic Field_Resultant, Weight (Data from the ankle unit's sensors)

To construct the mathematical model, we adopted the K-means clustering algorithm to build our mathematical model. This model was developed using the selected features and trained on the bootstrapped training dataset.

In [56]:
```
"""
Defining multiple functions to create a mathematical model
"""

from sklearn.metrics import adjusted_rand_score
from scipy import stats
import warnings

# Creating new clustering model using the df_origin (KMeans Clustering Model)
def creating_clustering(df_origin):
    df = df_origin.copy()
    subject = df['ID']
    y = df['activityID']
    df.drop(columns= ['ID','activityID'], inplace = True)

    df_scaled = normalize(df.values, norm='l2')

    kmeans = KMeans(n_clusters=y.nunique(), random_state=42).fit(df_scaled)
    cluster_pred = kmeans.predict(df_scaled)

    df['activityID'] = y
    df['cluster'] = cluster_pred
    df['ID'] = subject

    match = {}

    for i in df['activityID'].unique():
        list_index = []
        for i2 in range(len(df)):
            if df['activityID'][i2] == i:
```

```
                list_index.append(i2)
        cluster = df.loc[list_index, 'cluster'].mode().iloc[0]
        match[cluster] = i
        print(f"activityID == {i} : cluster {cluster}")

    return df, kmeans, match

# Making a new cluster based on df_origin and a KMeans clustering model
def new_prediction(df_origin, kmeans):
    df = df_origin.copy()
    subject = df['ID']
    y = df['activityID']
    timestamp = df['timestamp']
    df.drop(columns= ['ID','timestamp','activityID'], inplace = True)

    df_scaled = normalize(df.values, norm='l2')

    cluster_pred = kmeans.predict(df_scaled)

    df['activityID'] = y
    df['cluster'] = cluster_pred

    match = {}

    for i in df['activityID'].unique():
        list_index = []
        for i2 in range(len(df)):
            if df['activityID'][i2] == i:
                list_index.append(i2)
        cluster = df.loc[list_index, 'cluster'].mode().iloc[0]
        match[cluster] = i

    df['ID'] = subject
    df['timestamp'] = timestamp

    return df

# Assessing the performance of the model using adjusted rand score
def assess_model(df):

    true_labels = df['activityID']
    cluster_labels = df['cluster']

    score = adjusted_rand_score(true_labels, cluster_labels)

    print(f"Adjusted Rand Index (ARI) Score: {score:.4f}")

# Generating the prediction model using the KMeans Clustering model and a clustered dataframe
def prediction(df, kmeans, match, target_id, start, end):

    activities = {0:'transient', 1:'lying', 2:'sitting', 3:'standing',
              4:'walking', 5:'running', 6:'cycling', 7:'nordic walking',
              9:'watching TV', 10:'computer work', 11:'car driving',
              12:'ascending stairs', 13:'descending stairs', 16:'vacuum cleaning',
              17:'ironing', 18:'folding laundry', 19:'house cleaning',
              20:'playing soccer', 24:'rope jumping'}

    model_df = df[(df['ID'] == target_id) & (df['timestamp'] >= start) & (df['timestamp'] <= end)]

    start_point = model_df['timestamp'].min()
    end_point = model_df['timestamp'].max()

    most_frequent_activity_id = stats.mode(model_df['activityID']).mode[0]
    most_frequent_activity = activities[most_frequent_activity_id]
    print(f"Most frequent activityID: {most_frequent_activity} | {most_frequent_activity_id}")

    most_frequent_cluster = stats.mode(model_df['cluster']).mode[0]
    print(f"Most frequent cluster: {most_frequent_cluster}")

    matched_activity_id = match.get(most_frequent_cluster, None)
    if matched_activity_id is not None:
        matched_activity = activities.get(matched_activity_id, "Not found")
        print(f"Matched activity for cluster {most_frequent_cluster}: {matched_activity} | {matched_activity_id
    else:
        print(f"No matched activity found for cluster {most_frequent_cluster}")
        matched_activity = 'Not found'

    duration = end_point - start_point
    print(f"Activity {matched_activity} : {duration} seconds")

# Basic model setting (Training)
def model_setting(df):
    df, kmeans, match = creating_clustering(df)
    return df, kmeans, match

# Setting up the test environment (Testing)
def model(df_test, kmeans, match, target_id, start, end):

    activities = {0:'transient', 1:'lying', 2:'sitting', 3:'standing',
```

```
                4:'walking', 5:'running', 6:'cycling', 7:'nordic walking',
                9:'watching TV', 10:'computer work', 11:'car driving',
                12:'ascending stairs', 13:'descending stairs', 16:'vacuum cleaning',
                17:'ironing', 18:'folding laundry', 19:'house cleaning',
                20:'playing soccer', 24:'rope jumping'}

    df = new_prediction(df_test, kmeans)
    warnings.filterwarnings("ignore", category=FutureWarning)

    prediction(df, kmeans, match, target_id, start, end)

    return df
```

In [57]:
```
"""
Duplicating the df_boot and df_test dataframes for running the mathematical model.
"""

df_clustering = df_boot[['ID','activityID','bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','weight']].copy(deep=Tru

df_test2 = df_test.loc[:,['timestamp','ID','activityID','bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','weight']].copy(deep=Tru
```

In [58]:
```
"""
Creating and training the mathematical model
"""

_, kmeans, match = model_setting(df_clustering)
```

```
activityID == 1.0 : cluster 4
activityID == 2.0 : cluster 9
activityID == 3.0 : cluster 9
activityID == 4.0 : cluster 7
activityID == 5.0 : cluster 2
activityID == 6.0 : cluster 13
activityID == 7.0 : cluster 13
activityID == 9.0 : cluster 1
activityID == 11.0 : cluster 0
activityID == 12.0 : cluster 16
activityID == 13.0 : cluster 3
activityID == 16.0 : cluster 7
activityID == 17.0 : cluster 11
activityID == 18.0 : cluster 6
activityID == 19.0 : cluster 7
activityID == 24.0 : cluster 2
activityID == 10.0 : cluster 10
activityID == 20.0 : cluster 8
```

For practical application, we tested the developed model on the test dataset. The model takes the outputs from the model setting phase, including the trained K-means model and the activity-cluster matching information referred to as 'match', a target ID and a specified time range as input. Along with a target ID and a specified time range, the model predicts the most likely activity during that period and calculates the total duration for the given timeframe. When the model selects the dominant activity and cluster, it calculates the most frequent value in the data points. This information holds significance in gaining insights into user behaviour or facilitating real-time predictions across various circumstances.

However, due to the limitations of K-means clustering, each cluster may contain more than one activity, leading to a potential decrease in matching accuracy.

In [59]:
```
"""
Testing the mathematical model
"""

df_test2 = model(df_test2, kmeans, match, 101, 40, 309)
```

```
Most frequent activityID: lying | 1.0
Most frequent cluster: 4
Matched activity for cluster 4: lying | 1.0
Activity lying : 269.0 seconds
```

In [60]:
```
"""
Testing the mathematical model 2 (New prediction)
"""

prediction(df_test2, kmeans, match, 105, 3679, 3751)
```

```
Most frequent activityID: rope jumping | 24.0
Most frequent cluster: 2
Matched activity for cluster 2: rope jumping | 24.0
Activity rope jumping : 70.0 seconds
```

In [61]:
```
"""
Assessing the mathematical model using the Adjusted Rand Index
"""
```

```
assess_model(df_test2)
```

```
Adjusted Rand Index (ARI) Score: 0.1927
```

Predictive Hypothesis and testing

To assess and make a blueprint for improving the Colibri Wireless unit and model, the data team postulated and tested two predictive hypotheses. This process is necessary due to the low score of the silhouette Score, 0.2593, and the *adjusted Rand Index score, 0.1927.

* The Adjusted Rand Index (ARI) score represents how accurately a clustering algorithm has grouped data. A score closer to 1 means more accurate clustering and negative values up to -1 imply clustering less accurately than random grouping.

(1) The mathematical model can distinguish between sitting and standing. This hypothesis is deemed important because if this model cannot classify two activities, it is necessary to integrate some activities into one category.

(2) The mathematical model can differentiate between static activities and dynamic activities. The classification between static and dynamic activities is essential for understanding the model's performance in capturing the distinction between activities with varying levels of physical movement. For this hypothesis, we defined static activities and dynamic activities based on the mean value of BPM feature.

Please refer to the list of activities below.

- Static activities: car driving, computer work, folding laundry, house cleaning, ironing, lying, sitting, standing, watching TV, walking, and vacuum cleaning
- Dynamic activities: ascending stairs, cycling, descending stairs, nordic walking, playing soccer, rope jumping, and running

In [63]:
```
"""
Duplicating the df_boot and df_test dataframes to test the predictive hypotheses
"""


df_train_hyp = df_boot[['ID','activityID','bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','weight']].copy(deep=Tru
df_test_hyp = df_test[['timestamp','ID','activityID','bpm', 'ankle_3D_acc_16g_x', 'ankle_3D_acc_16g_y',
       'ankle_3D_acc_16g_z', 'ankle_3D_gyro_x', 'ankle_3D_gyro_y',
       'ankle_3D_gyro_z', 'ankle_3D_magnet_y','ankle_3D_magnet_z', 'ankle_magneto_res','weight']].copy(deep=Tru

# Defining a function that removes all activities except sitting and standing
def hypothesis_1(df):
    indexes = []
    for index in range(len(df)):
        if df['activityID'][index] == 2:
            indexes.append(index)
        elif df['activityID'][index] == 3:
            indexes.append(index)

    df_hyp1 = df.loc[indexes, :].copy(deep = True)
    df_hyp1.reset_index(drop= True, inplace=True)
    return df_hyp1
```

In [66]:
```
"""
Removing all activities except sitting and standing
"""

df_train_hyp1 = hypothesis_1(df_train_hyp)
df_test_hyp1 = hypothesis_1(df_test_hyp)
```

In [67]:
```
"""
Creating and training the mathematical model for the predictive hypothesis 1
"""

_, kmeans, match = model_setting(df_train_hyp1)
```

```
activityID == 2.0 : cluster 0
activityID == 3.0 : cluster 0
```

Predictive Hypothesis 1. The mathematical model can distinguish between sitting and standing

Following the testing, the predictive hypothesis has been rejected. We created a *scatterplot to visualise the data points for the two activities. Simultaneously, we examined the model's result.

As a result, the model struggles to distinguish between two distinct activities, namely 'sitting' and 'standing.' In particular, the model tends to misclassify the 'sitting' activity as 'standing.'

To enhance the prediction accuracy of the model, it is essential to combine similar activities into a single category. Hence, the data team conducted a test on the predictive hypothesis 2 to validate the necessity of this recommendation.

*Even if the model uses more than two features for K-means clustering, we chose to represent it using only two arbitrary features in a

scatterplot, aiming to enhance the ease of result comprehension through data visualisation. Consequently, the actual distribution of data points and the scatterplot may have some discordances.

In [68]:
```
"""
Testing the mathematical model for the predictive hypothesis 1
-> Using the sitting activity timestamps
"""

df_test_hyp1 = model(df_test_hyp1, kmeans, match, 102, 359, 575)
```

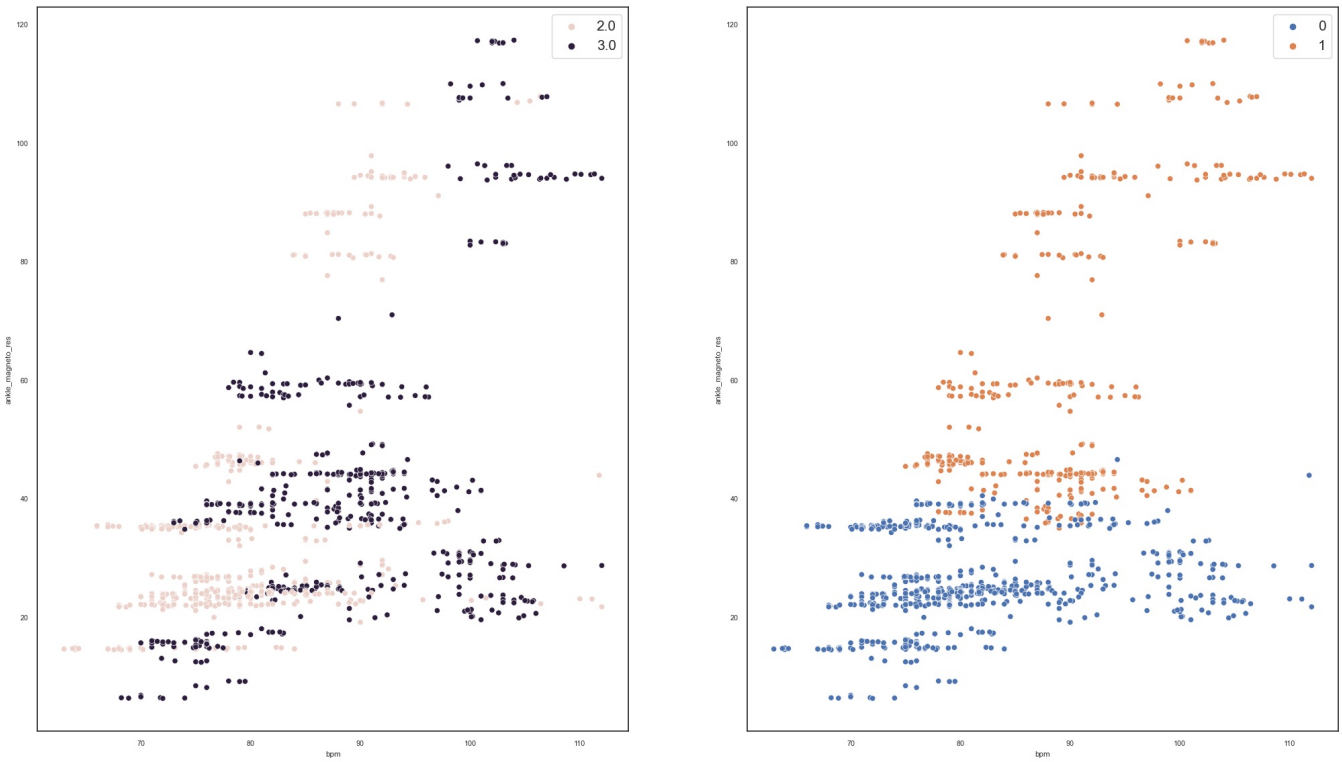
```
Most frequent activityID: sitting | 2.0
Most frequent cluster: 0
Matched activity for cluster 0: standing | 3.0
Activity standing : 216.0 seconds
```

In [69]:
```
"""
Testing the mathematical model for the predictive hypothesis 1
-> Using the standing activity timestamps
"""

prediction(df_test_hyp1, kmeans, match, 104, 623, 856)
```

```
Most frequent activityID: standing | 3.0
Most frequent cluster: 0
Matched activity for cluster 0: standing | 3.0
Activity standing : 233.0 seconds
```

In [77]:
```
"""
Creating a scatter plot for easy observation of data distribution (Original codes from James, revised by Jungye
< Predictive Hypothesis 1 >
"""

fig, axes = plt.subplots(nrows=1,ncols=2,figsize=(25, 14))
plt.rcParams.update({'font.size': 20})

sns.scatterplot(df_test_hyp1 ,x='bpm',y='ankle_magneto_res',hue=df_test_hyp1['activityID'],ax=axes[0])
sns.scatterplot(df_test_hyp1 ,x='bpm',y='ankle_magneto_res',hue=df_test_hyp1['cluster'],ax=axes[1])

axes[0].legend(prop={'size': 15})
axes[1].legend(prop={'size': 15})

fig.suptitle("K-means Cluster Scatter plot for Predictive Hypothesis 1", fontsize=20)
plt.show()
```

K-means Cluster Scatter plot for Predictive Hypothesis 1



Predictive Hypothesis 2. The mathematical model can differentiate between static activities and dynamic activities

To assess the second hypothesis, the data team established two activity categories: static activities and dynamic activities. Activities were grouped based on their BPM values. If an average value of BPM was greater than 109, the activity was defined as dynamic activities. In contrast, the activity with an average BPM value lower than 109 was classified as static activities.

In [71]:
```
"""
1. Defining a function that transforms each activity into static activities and dynamic activities
2. Re-defining the 'prediction' function to identify activities into only two categories
"""

def hypothesis_2(df):

    feature_merging = {1:1, 2:1, 3:1, 4:1, 5:2, 6:2, 7:2, 9:1, 10:1, 11:1,
                  12:2, 13:2, 16:1, 17:1, 18:1, 19:1, 20:2, 24:2}

    origin_id = []
    new_id = []

    for i in range(len(list(feature_merging.items()))):
        origin, new = list(feature_merging.items())[i]
        origin_id.append(origin)
        new_id.append(new)

    df['activityID'].replace(to_replace=origin_id, value=new_id, inplace=True)

    df_hyp2 = df

    return df_hyp2

def prediction(df, kmeans, match, target_id, start, end):

    activities = {1:'Static Activities', 2:'Dynamic Activities'}

    model_df = df[(df['ID'] == target_id) & (df['timestamp'] >= start) & (df['timestamp'] <= end)]

    start_point = model_df['timestamp'].min()
    end_point = model_df['timestamp'].max()

    most_frequent_activity_id = stats.mode(model_df['activityID']).mode[0]
    most_frequent_activity = activities[most_frequent_activity_id]
    print(f"Most frequent activityID: {most_frequent_activity} | {most_frequent_activity_id}")

    most_frequent_cluster = stats.mode(model_df['cluster']).mode[0]
    print(f"Most frequent cluster: {most_frequent_cluster}")

    matched_activity_id = match.get(most_frequent_cluster, None)
    if matched_activity_id is not None:
        matched_activity = activities.get(matched_activity_id, "Not found")
        print(f"Matched activity for cluster {most_frequent_cluster}: {matched_activity} | {matched_activity_id
    else:
        print(f"No matched activity found for cluster {most_frequent_cluster}")
        matched_activity = 'Not found'

    duration = end_point - start_point
    print(f"Activity {matched_activity} : {duration} seconds")
```

In [72]:
```
"""
Transforming each activity into static activities and dynamic activities
"""

df_train_hyp2 = hypothesis_2(df_train_hyp)
df_test_hyp2 = hypothesis_2(df_test_hyp)
```

As a result, it appears that this model can distinguish between static activities and dynamic activities. The model's output indicated that both activities belong to distinct dominant clusters. In addition, the predictive functionality within the model demonstrated reasonably effective matching between activities and clusters. Nevertheless, the data points in real activity and cluster features still show different shapes. Therefore, the data team cannot affirm that this model perfectly distinguishes between the two activity categories.

In [73]:
```
"""
Creating and training the mathematical model for the predictive hypothesis 2
"""

_, kmeans, match = model_setting(df_train_hyp2)
```

```
activityID == 1.0 : cluster 1
activityID == 2.0 : cluster 0
```

In [74]:
```
"""
Testing the mathematical model for the predictive hypothesis 2
-> Using the static activity timestamps
"""

df_test_hyp2 = model(df_test_hyp2, kmeans, match, 101, 40, 150)
```

```
Most frequent activityID: Static Activities | 1.0
Most frequent cluster: 1
Matched activity for cluster 1: Static Activities | 1.0
Activity Static Activities : 100.0 seconds
```

In [75]:
```
"""
Testing the mathematical model for the predictive hypothesis 2
```

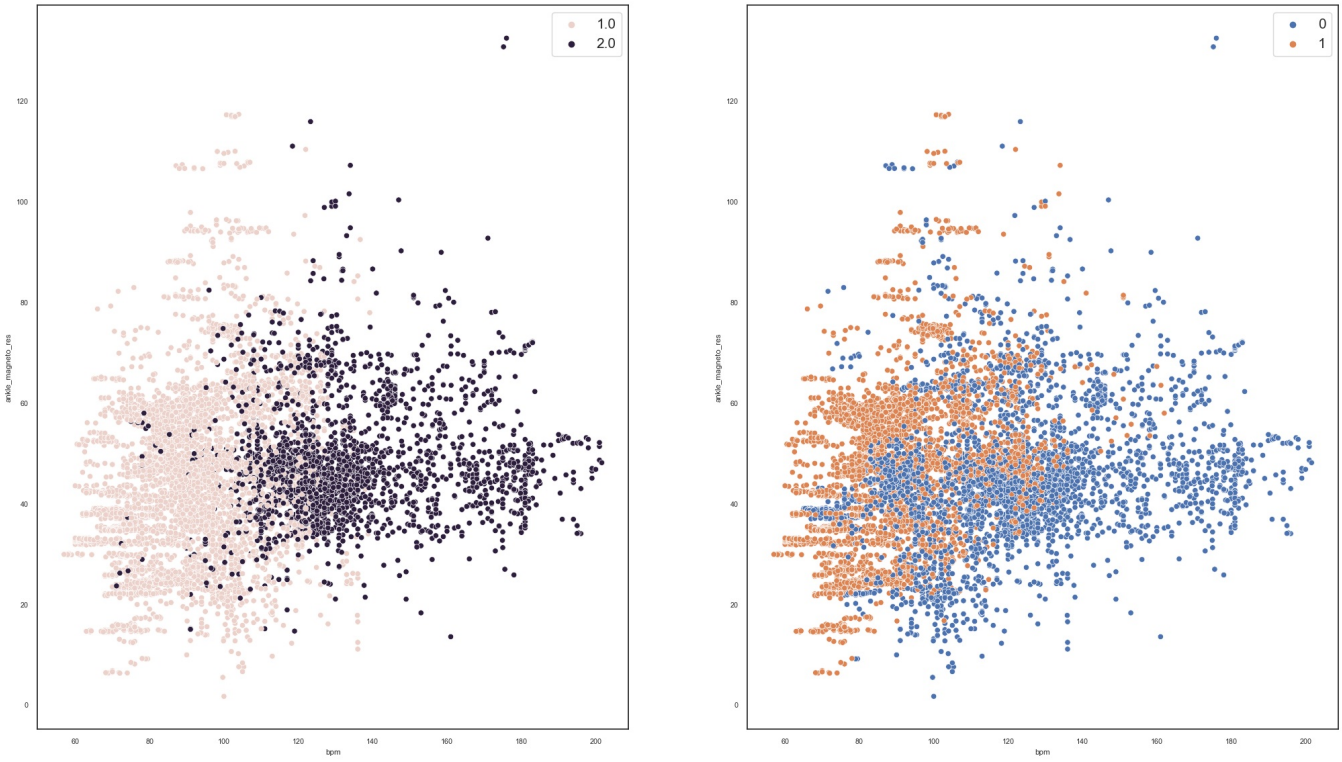
```
-> Using the dynamic activity timestamps
"""

prediction(df_test_hyp2, kmeans, match, 104, 1497, 1560)
```

```
Most frequent activityID: Dynamic Activities | 2.0
Most frequent cluster: 0
Matched activity for cluster 0: Dynamic Activities | 2.0
Activity Dynamic Activities : 63.0 seconds
```

In [78]:

```
"""
Creating a scatter plot for easy observation of data distribution (Original codes from James, revised by Jungye
< Predictive Hypothesis 2 >
"""

fig, axes = plt.subplots(nrows=1,ncols=2,figsize=(25, 14))

plt.rcParams.update({'font.size': 20})

sns.scatterplot(df_test_hyp2 ,x='bpm',y='ankle_magneto_res',hue=df_test_hyp2['activityID'],ax=axes[0])
sns.scatterplot(df_test_hyp2 ,x='bpm',y='ankle_magneto_res',hue=df_test_hyp2['cluster'],ax=axes[1])

axes[0].legend(prop={'size': 15})
axes[1].legend(prop={'size': 15})

fig.suptitle("K-means Cluster Scatter plot for Predictive Hypothesis 2")
plt.show()
```

K-means Cluster Scatter plot for Predictive Hypothesis 2



Conclusion and Actionable Insights

Conclusion

Overall, following refining the data, we handled the imbalance in data point distribution across activities by applying bootstrap resampling to activity-subject combinations with less than 500 data points. Hence, each combination includes a minimum of 500 data points. In addition, the data team chose eleven features from the original dataset through the correlation coefficient matrix and exploratory data analysis. Subsequently, we conducted the Silhouette test, comparing scores between combinations of all subsets of unit parts and two model methods: K-Means clustering and the Gaussian Mixture Model.

The Silhouette test results, coupled with feature selection, guided us to create a mathematical model using K-means clustering with the eleven selected features from the ankle unit. These features include BPM, Weight, Three-axis acceleration, Three-axis angular rate, y-axis and z-axis magnetic field, and the resultant of the magnetic field.

The model was trained using *the training dataset, and we evaluated its performance on the test dataset. Following the test, the model predicted the most probable activity during that period and computed the total duration for the specified timeframe. However, owing to the constraints of K-means clustering, it's important to note that each cluster may contain more than one activity, as it is hard to classify certain activities that share similar features and movements. This factor could potentially lead to a reduction in matching accuracy.

*Prior to initiating the bootstrap process, the data team partitioned the original dataset into two datasets, namely train and test datasets.

The training dataset encompasses 70% of the data from the original dataset, while the test dataset comprises the remaining 30%.

Therefore, we specified the two predictive hypotheses. Based on the result of the predictive hypothesis testing and insights from the data analysis, we propose some actionable recommendations.

Actionable Recommendations

The first hypothesis, the mathematical model can distinguish between sitting and standing, was rejected and the second hypothesis, the mathematical model can differentiate between static activities and dynamic activities, was confirmed. This implies that the model is unable to recognise activities that share similarities. Thus, we recommend merging similar activities into a unified category. For instance, combining Nordic walking and walking into a 'walking' category. If approved by the board, the data team will proceed to integrate these activities based on the clustering output of the model and the activities' unique characteristics to not only enhance prediction accuracy but also maintain as many activities as possible in our model.

Alternatively, additional sensors may be incorporated into the unit to help the model in predicting user activities. For example, in the MTw Awinda by Xsens, a barometer is a sensing element to measure atmospheric pressure. In motion sensing applications, it aids the sensor in acquiring height information. Considering that the barometer is used as a main sensor by MTw, incorporating it into our unit holds the potential to enhance its capabilities. Moreover, the InertiaCube3 by InterSense utilises compass measurements to prevent the accumulation of gyroscopic drift. If the company decides to add a compass to our unit, it would significantly enhance the meaningfulness of gyroscope data, providing an opportunity for improvements to our mathematical model through the utilisation of more relevant features.

On the other hand, in the subject information, there is an uneven collection of data for gender and the dominant hand. It is advised to gather the almost same amount of data in both male and female gender and collect several combinations of weight, height, and age. It could elevate the data diversity and allow us to analyse if it has a statistically significant difference between these features. Additionally, there exists an imbalance in the distribution of data points among activities, posing challenges for the data team during the analysis. Despite implementing bootstrapping, the sampled data still has limitations due to its random creation. Therefore, it is imperative for the company to ensure the acquisition and collection of sufficient data, especially across all activities to enhance the precision of the data analysis results.

The data team is confident that implementing these recommendations can enhance the performance of the Colibri Wireless unit and its associated mathematical model. Such improvements in our product and model are anticipated to guide us as a leading company among competitors. Based on the future decision that the boarder will make, we will work on further information collection, data analysis, and model improvement.