DSRM_FINAL_251046

January 6, 2022

Candidate Number : 251046

0.1 PAMAP2_Dataset: Physical Activity Monitoring

0.1.1 Introduction

Modern world is a hub of technology and innovation.In the past decade or so, number of individuals spending time for physical activities has increased drastically. It is common to see individuals wearing smartwatch to track their day to day fitness goals. Focusing on our topic, which is physical activity monitoring: the main challenge would be measuring the activity with high accuracy in free-living conditions. The major factors influencing our physical activity are discussed in following sections. We try to find a pattern to predict any activity based on the provided dataset. Our goal is to derive actionable insights that will allow the development of software and/or hardware to determine the amount and type of physical activity carried out by an individual.

Given dataset includes various physical activities performed by 9 subjects (8 men & 1 woman) and data is collected from 3 inertial measurement units (IMU) and a heart rate monitor. Following tasks will be performed in the upcoming sections :

1. Perform exploratory data analysis to handle missing or dirty data;

2. Develop and test hypothesis for a relationship between a single pair of attributes;
3. Develop and test a model which uses multiple attributes to make predictions.

- In **Part 1**, EDA is carried out on the provided dataset as per various activities to gain insights on how one parameter affects the rest. Here, data is loaded from the files and then cleaning methods are performed. Handling of missing values, data normalization and cleaning are key activities performed here.
- In **Part 2**, various hypothesis are tested to prove relations between pair of attributes by statistical tests.
- Finally, In **Part 3**, a model is created with relevant insights from the exploratory datas performed.

[4]:
```
## importing all the libraries
## importing all the libraries
import pandas as pd
import numpy as np
import math
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from scipy import stats
%matplotlib inline
import matplotlib.pyplot as plt
from itertools import cycle, islice

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
from sklearn import cluster
from collections import defaultdict
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics.cluster import silhouette_score
from sklearn.metrics import precision_score,recall_score, f1_score,␣
↪confusion_matrix,roc_auc_score,roc_curve, accuracy_score
from sklearn import svm
import csv
```

Data Overview

[12]:
```
# opening the subject information CSV file
df = pd.read_csv(r'C:\Users\jovyj\OneDrive\Desktop\subjectInformation.csv')
df
```

[12]:

	Subject ID	Sex	Age(years)	Height(cm)	Weight(kg)	Resting HR(bpm)	Max HR (bpm)	Dominant hand
0	101	Male	27	182	83	75	193	right
1	102	Female	25	169	78	74	195	right
2	103	Male	31	187	92	68	189	right
3	104	Male	24	194	95	58	196	right
4	105	Male	26	180	73	70	194	right
5	106	Male	26	183	69	60	194	right
6	107	Male	23	173	86	60	197	right
7	108	Male	32	179	87	66		
8	109	Male	31	168	65	54		

```
7           188          left
8           189         right
```

Above table is obtained from '***subjectInformation.pdf***' file which comes along the dataset.It shows the 9 subject's physical parameters and the main observations are :

- subject 8 is left handed
- subject 2 is the only female

[13]:
```
# opening the PerformedActivities Summary CSV file
df = pd.read_csv(r'C:\Users\jovyj\OneDrive\Desktop\PerformedActivitiesSummary.
↪csv')
df
```

[13]:

	Activity	subject101	subject102	subject103	subject104	\
0	1 - lying	271.86	234.29	220.43	230.46	
1	2 - sitting	234.79	223.44	287.60	254.91	
2	3 - standing	217.16	255.75	205.32	247.05	
3	4 - walking	222.52	325.32	290.35	319.31	
4	5 - running	212.64	92.37	0.00	0.00	
5	6 - cycling	235.74	251.07	0.00	226.98	
6	7 - Nordic walking	202.64	297.38	0.00	275.32	
7	9 - watching TV	836.45	0.00	0.00	0.00	
8	10 - computer work	0.00	0.00	0.00	0.00	
9	11 - car driving	545.18	0.00	0.00	0.00	
10	12 - ascending stairs	158.88	173.40	103.87	166.92	
11	13 - descending stairs	148.97	152.11	152.72	142.83	
12	16 - vacuum cleaning	229.40	206.82	203.24	200.36	
13	17 - ironing	235.72	288.79	279.74	249.94	
14	18 - folding laundry	271.13	0.00	0.00	0.00	
15	19 - house cleaning	540.88	0.00	0.00	0.00	
16	20 - playing soccer	0.00	0.00	0.00	0.00	
17	24 - rope jumping	129.11	132.61	0.00	0.00	
18	Labeled total	4693.07	2633.35	1743.27	2314.08	
19	Total	6957.67	4469.99	2528.32	3295.75	

	subject105	subject106	subject107	subject108	subject109	Sum	\
0	236.98	233.39	256.10	241.64	0.00	1925.15	
1	268.63	230.40	122.81	229.22	0.00	1851.80	
2	221.31	243.55	257.50	251.59	0.00	1899.23	
3	320.32	257.20	337.19	315.32	0.00	2387.53	
4	246.45	228.24	36.91	165.31	0.00	981.92	
5	245.76	204.85	226.79	254.74	0.00	1645.93	
6	262.70	266.85	287.24	288.87	0.00	1881.00	
7	0.00	0.00	0.00	0.00	0.00	836.45	
8	1108.82	617.76	0.00	687.24	685.49	3099.31	
9	0.00	0.00	0.00	0.00	0.00	545.18	
10	142.79	132.89	176.44	116.81	0.00	1172.00	

11	127.25	112.70	116.16	96.53	0.00	1049.27
12	244.44	210.77	215.51	242.91	0.00	1753.45
13	330.33	377.43	294.98	329.89	0.00	2386.82
14	0.00	217.85	0.00	236.49	273.27	998.74
15	284.87	287.13	0.00	416.90	342.05	1871.83
16	0.00	0.00	0.00	181.24	287.88	469.12
17	77.32	2.55	0.00	88.05	63.90	493.54
18	4117.97	3623.56	2327.63	4142.75	1652.59	27248.27
19	5295.54	4917.78	3135.98	5884.41	2019.47	38504.91

	Nr. of subjects
0	8.0
1	8.0
2	8.0
3	8.0
4	6.0
5	7.0
6	7.0
7	1.0
8	4.0
9	1.0
10	8.0
11	8.0
12	8.0
13	8.0
14	4.0
15	5.0
16	2.0
17	6.0
18	NaN
19	NaN

Above table is obtained from 'PerformedActivitiesSummary.pdf' file and clearly '**subject 1**' , '**subject 8**' and '**subject 5**' has performed maximum activities.

0.2 Part I : Exploratory Data Analysis

Data Munging

[72]:
```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

[74]:
```
## List for filepath of each individuals
```

```
subject_files=['/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject101.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject102.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject103.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject104.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject105.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject106.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject107.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject108.dat','/content/drive/My Drive/Colab Notebooks/DSRM/Protocol/
 ↪subject109.dat']

## List for main parameters
columns_main= ['timestamp','activityID','heart_rate']

## List for Hand IMU parameters
columns_IMU_hand=['hand_temperature','hand_acceleration_16g␣
 ↪(x)','hand_acceleration_16g (y)','hand_acceleration_16g␣
 ↪(z)','hand_acceleration_6g (x)','hand_acceleration_6g␣
 ↪(y)','hand_acceleration_6g (z)','hand_gyroscope (x)','hand_gyroscope␣
 ↪(y)','hand_gyroscope (z)','hand_magnetometer (x)','hand_magnetometer␣
 ↪(y)','hand_magnetometer␣
 ↪(z)','hand_orientation(a)','hand_orientation(b)','hand_orientation(c)','hand_orientation(d)

## List for Chest IMU parameters
columns_IMU_chest=['chest_temperature','chest_acceleration_16g␣
 ↪(x)','chest_acceleration_16g (y)','chest_acceleration_16g␣
 ↪(z)','chest_acceleration_6g (x)','chest_acceleration_6g␣
 ↪(y)','chest_acceleration_6g (z)','chest_gyroscope (x)','chest_gyroscope␣
 ↪(y)','chest_gyroscope (z)','chest_magnetometer (x)','chest_magnetometer␣
 ↪(y)','chest_magnetometer␣
 ↪(z)','chest_orientation(a)','chest_orientation(b)','chest_orientation(c)','chest_orientatio

## List for Ankle IMU parameters
columns_IMU_ankle=['ankle_temperature','ankle_acceleration_16g␣
 ↪(x)','ankle_acceleration_16g (y)','ankle_acceleration_16g␣
 ↪(z)','ankle_acceleration_6g (x)','ankle_acceleration_6g␣
 ↪(y)','ankle_acceleration_6g (z)','ankle_gyroscope (x)','ankle_gyroscope␣
 ↪(y)','ankle_gyroscope (z)','ankle_magnetometer (x)','ankle_magnetometer␣
 ↪(y)','ankle_magnetometer␣
 ↪(z)','ankle_orientation(a)','ankle_orientation(b)','ankle_orientation(c)','ankle_orientatio

## Final list of column headings
columns = columns_main+columns_IMU_hand+columns_IMU_chest+columns_IMU_ankle

## Dictionary representing the activityID and corresponding tags
```

```
activity_ID = {0: 'transient activity', 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'}
```

First, a list is created with location of filenames for various subject's data which is used to create the dataframe.Then, further lists are created to store column headings for various parameters like IMU data (hand,chest and ankle) and heart rate,timestamp, activityID. Also, a dictionary is created to store activities name as well as corresponding number to comprehend which activity is being performed at the instant.

[75]:
```
# Classification of activity level as per Metabolic equivalent of task (MET)

#lying, sitting, standing and ironing
light_activity = [1,2,3,17]
#vacuum cleaning,descending stairs, walking, Nordic walking and cycling
medium_activity = [16,13,4,7,6]
# ascending stairs,running and rope jumping
heavy_activity = [12,5,24]

relevant_activities=light_activity+medium_activity+heavy_activity

#Function used to classify activities
def activity_class(activity):
    if activity in light_activity:
        return 'light'
    if activity in medium_activity:
        return 'moderate'
    if activity in heavy_activity:
        return 'vigorous'
```

The activities in the dataset are classified as per Metabolic equivalent of task (MET):

Light Effort : lying, sitting, standing and ironing.

Moderate Effort : vacuum cleaning, descending stairs, walking, Nordic walking and cycling

Vigorous Effort : ascending stairs, running and rope jumping

Data Collection

[76]:
```
## creating a dataframe from subjects list
all_data=pd.DataFrame()
for subject in subject_files:
    df = pd.read_table(subject, sep='\s+')
    df.columns = columns
    df['subject_id'] = subject[-7:-4]
```

```
    df['activity_level'] = df['activityID'].apply(activity_class)
    all_data = all_data.append(df, ignore_index=True)
```

Here,we create our initial dataframe by loading all subject's data by iterating over subjects list, which contains the file-location. Then, column names are assigned and a new column is created for **subject_id** as well as activity level.

[77]:
```
print("Number of rows and columns = {}".format(all_data.shape))
all_data
```

```
Number of rows and columns = (2872524, 56)
```

[77]:

	timestamp	activityID	…	subject_id	activity_level
0	8.39	0	…	101	None
1	8.40	0	…	101	None
2	8.41	0	…	101	None
3	8.42	0	…	101	None
4	8.43	0	…	101	None
…	…	… …	…	…	
2872519	100.19	0	…	109	None
2872520	100.20	0	…	109	None
2872521	100.21	0	…	109	None
2872522	100.22	0	…	109	None
2872523	100.23	0	…	109	None

[2872524 rows x 56 columns]

Taking a glance at the above table, the underlying requisite of data clensing is evident. The readme file along with the dataset explains about various missing data, for instance the '**activityID 0**', which is *transient period* and must be removed from the analysis. Data cleaning will be done thoroughly in the following section.

[78]:
```
### backup of datset
bkp_data =all_data
```

Data Cleansing Due to wireless data dropping, few sensory datas are missing and corresponding values are indicated with NaN. Since data is given every 0.01s meaning the sampling frequency of the HR-monitor is less, the missing HR-values are also indicated with NaN in the data-files. Also, each activity has NaN values for various subjects as most subjects did not perform all activities. To begin with, all Nan values of ***transient activity*** is dropped and also, the ***orientation*** colums are dropped as the data is irrelevant. The ***6g accelerometer*** is not precisely calibrated and due to high impacts caused by certain movements (e.g.running) with acceleration over 6g gets saturated. Hence, we refer only the data from the first accelerometer (with the scale of ±16g).

[79]:
```
## removal of transient activity
all_data = all_data.drop(all_data[all_data['activityID']==0].index)

## removal of orientation columns as they are irrelevant
```

```
all_data = all_data.drop␣
↪(['hand_orientation(a)','hand_orientation(b)','hand_orientation(c)','hand_orientation(d)','

## removal of 6g accelerometer
all_data = all_data.drop (['hand_acceleration_6g (x)','hand_acceleration_6g␣
↪(y)','hand_acceleration_6g (z)','ankle_acceleration_6g␣
↪(x)','ankle_acceleration_6g (y)','ankle_acceleration_6g (z)'],axis=1)
```

[80]:
```
print("Number of rows and columns = {}".format(all_data.shape))
```

```
Number of rows and columns = (1942872, 38)
```

[81]:
```
## finding missing heartrates
total_cells = np.product(all_data['heart_rate'].shape)
total_missing = all_data['heart_rate'].isnull().sum()
print('Total missing heart rate vales = {}'.format(total_cells-total_missing))
print('Percentage of missing Heart Rate Values␣
↪=',((1-(total_cells-total_missing)/total_cells)*100),'%')
```

```
Total missing heart rate vales = 177408
Percentage of missing Heart Rate Values = 90.86877570936223 %
```

Heart rate data is very crucial for the analysis and a whopping **90%** of the heart rate values are missing in the given dataset. This shows the irregularities in data acquisition. For instance, when the real-time requirement of the system is high, the machine has no time to judge and make decisions. This is a major challenge in the provided dataset and a judgement has to be taken whether to drop NaN or interpolate. I decided to drop all NaN, since the sample is obtained from data collected at 0.01s interval and data removed for certain interval of time does not affect our final goal, which is to co-relate each features of dataset.

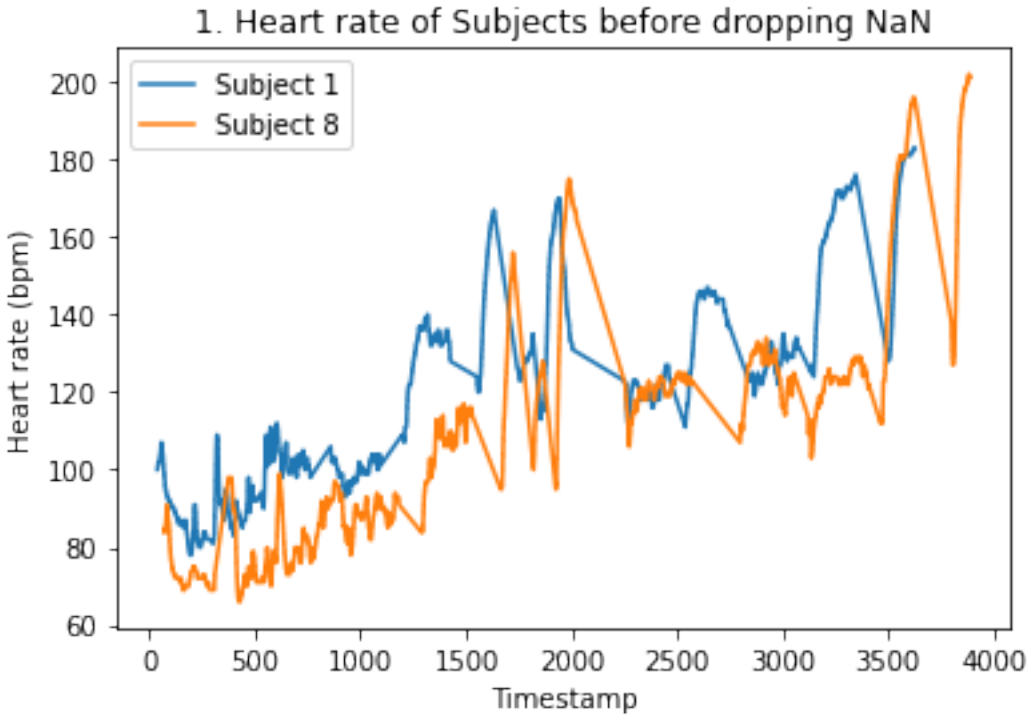
[82]:
```
## seperating subjects who performed maximum activities
filtered_subject1_data = all_data[all_data["subject_id"]=="101"]
filtered_subject8_data = all_data[all_data["subject_id"]=="108"]
```

From the activities performed chart showed initially, **Subject1** and **Subject2** needs to be considered for analyzing any effect of dropping heart rate NaN values as they have the maximum activities. This can be visualized by plotting 2 graphs :

- a. Actual Dataset before dropping NaN's
- b. After transformation

[83]:
```
## heart rate graph before dropping NaN values
plt.title('1. Heart rate of Subjects before dropping NaN')
plt.xlabel('Timestamp')
plt.ylabel('Heart rate (bpm)')
read_data=filtered_subject1_data.iloc[:,:-1].dropna(axis=0,how='any')
read_data=read_data.drop(['activityID'],axis=1).astype(float).
↪dropna(axis=0,how='any')
```

```
plt.plot(read_data['timestamp'].values.astype(float),read_data['heart_rate'].
 ↪values.astype(float),label='Subject 1')

read_data=filtered_subject8_data.iloc[:,:-1].dropna(axis=0,how='any')
read_data=read_data.drop(['activityID'],axis=1).astype(float).
 ↪dropna(axis=0,how='any')
plt.plot(read_data['timestamp'].values.astype(float),read_data['heart_rate'].
 ↪values.astype(float),label='Subject 8')
plt.legend()
plt.show()
```



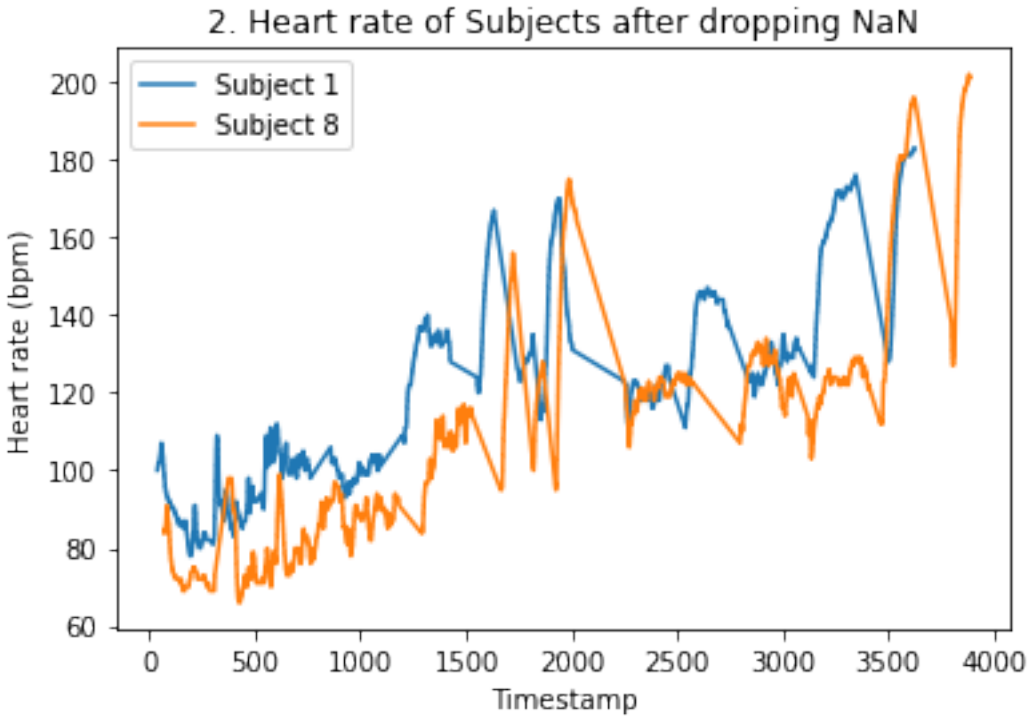
As per above chart of heart rate data, values of heart rate varyation with timeperiod for 'subject 1' and 'subject 8' is captured.

[84]:
```
## removal of any NaN values across the dataset
all_data = all_data.dropna()
all_data.reset_index(drop = True, inplace = True)
```

Here, dropping of all NaN values are completed.

[85]:
```
## seperating subjects who performed maximum activities
filtered_subject1_data = all_data[all_data["subject_id"]=="101"]
filtered_subject8_data = all_data[all_data["subject_id"]=="108"]
```

[86]:
```
## heart rate graph after dropping NaN values
plt.title('2. Heart rate of Subjects after dropping NaN')
plt.xlabel('Timestamp')
plt.ylabel('Heart rate (bpm)')
read_data=filtered_subject1_data.iloc[:,:-1].dropna(axis=0,how='any')
read_data=read_data.drop(['activityID'],axis=1).astype(float).
 ↪dropna(axis=0,how='any')
plt.plot(read_data['timestamp'].values.astype(float),read_data['heart_rate'].
 ↪values.astype(float),label='Subject 1')

read_data=filtered_subject8_data.iloc[:,:-1].dropna(axis=0,how='any')
read_data=read_data.drop(['activityID'],axis=1).astype(float).
 ↪dropna(axis=0,how='any')
plt.plot(read_data['timestamp'].values.astype(float),read_data['heart_rate'].
 ↪values.astype(float),label='Subject 8')
plt.legend()
plt.show()
```

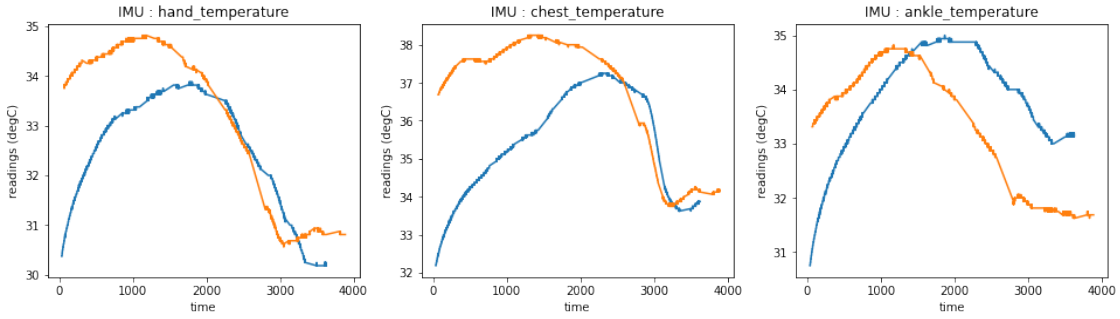


Comparing graph 1 & 2, it is clear that the effect of dropping NaN values are not at all significant. If we had interpolated the NaN, variance of data would be traded for future analysis.

[87]:
```
##plotting 'IMU : hand_temperature' readings based on timestamp
plt.figure(figsize=(3,3))
```

```
plt.subplots_adjust(2,2,6.2,3)
plt.subplot(131)
plt.title('IMU : hand_temperature')
plt.xlabel('time')
plt.ylabel('readings (degC)')
plt.plot(filtered_subject1_data['timestamp'].
 ↪values,filtered_subject1_data['hand_temperature'].values)
plt.plot(filtered_subject8_data['timestamp'].
 ↪values,filtered_subject8_data['hand_temperature'].values)

##plotting 'IMU : chest_temperature' readings based on timestamp
plt.subplot(132)
plt.title('IMU : chest_temperature')
plt.xlabel('time')
plt.ylabel('readings (degC)')
plt.plot(filtered_subject1_data['timestamp'].
 ↪values,filtered_subject1_data['chest_temperature'].values)
plt.plot(filtered_subject8_data['timestamp'].
 ↪values,filtered_subject8_data['chest_temperature'].values)

##plotting 'IMU : ankle_temperature' readings based in timestamp
plt.subplot(133)
plt.xlabel('time')
plt.ylabel('readings (degC)')
plt.title('IMU : ankle_temperature')
plt.plot(filtered_subject1_data['timestamp'].
 ↪values,filtered_subject1_data['ankle_temperature'].values)
plt.plot(filtered_subject8_data['timestamp'].
 ↪values,filtered_subject8_data['ankle_temperature'].values)
```

[87]: [<matplotlib.lines.Line2D at 0x7fc8a35988d0>]



Here, the chest temperature is peak at around 36 DegC, which is in normal human body temperature range.

[88]:
```
## storing the data into a new variable
df_new = all_data
df_new
```

[88]:

	timestamp	activityID	…	subject_id	activity_level
0	37.70	1	…	101	light
1	37.81	1	…	101	light
2	37.92	1	…	101	light
3	38.03	1	…	101	light
4	38.14	1	…	101	light
…	…	…	…	…	…
175493	94.66	24	…	109	vigorous
175494	94.77	24	…	109	vigorous
175495	94.88	24	…	109	vigorous
175496	94.98	24	…	109	vigorous
175497	95.09	24	…	109	vigorous

[175498 rows x 38 columns]

[89]:
```
print("Final Number of rows and columns in Dataset = {}".format(all_data.shape))
```

```
Final Number of rows and columns in Dataset = (175498, 38)
```

[90]:
```
## checking if any null values exist in dataframe
df_new.isnull().sum()
```

[90]:
```
timestamp                     0
activityID                    0
heart_rate                    0
hand_temperature              0
hand_acceleration_16g (x)     0
hand_acceleration_16g (y)     0
hand_acceleration_16g (z)     0
hand_gyroscope (x)            0
hand_gyroscope (y)            0
hand_gyroscope (z)            0
hand_magnetometer (x)         0
hand_magnetometer (y)         0
hand_magnetometer (z)         0
chest_temperature             0
chest_acceleration_16g (x)    0
chest_acceleration_16g (y)    0
chest_acceleration_16g (z)    0
chest_acceleration_6g (x)     0
chest_acceleration_6g (y)     0
chest_acceleration_6g (z)     0
chest_gyroscope (x)           0
chest_gyroscope (y)           0
```

```
chest_gyroscope (z)           0
chest_magnetometer (x)        0
chest_magnetometer (y)        0
chest_magnetometer (z)        0
ankle_temperature             0
ankle_acceleration_16g (x)    0
ankle_acceleration_16g (y)    0
ankle_acceleration_16g (z)    0
ankle_gyroscope (x)           0
ankle_gyroscope (y)           0
ankle_gyroscope (z)           0
ankle_magnetometer (x)        0
ankle_magnetometer (y)        0
ankle_magnetometer (z)        0
subject_id                    0
activity_level                0
dtype: int64
```

From above, It is clear that there are no missing values and thus we proceed with further analysis.

[91]:
```
## giving names to activityID
df_new.activityID=df_new.activityID.apply(lambda x:activity_ID[x])
```

[92]:
```
df_new.describe()
```

[92]:
```
           timestamp  …  ankle_magnetometer (z)
count  175498.000000  …           175498.000000
mean     1694.983481  …               17.242890
std      1091.288920  …               19.753212
min        31.220000  …             -100.864000
25%       739.225000  …                3.705748
50%      1467.170000  …               18.771050
75%      2654.185000  …               31.244975
max      4245.650000  …              138.163000

[8 rows x 35 columns]
```

Above table provides a summary of various statistics, which can be quite useful for gaining useful insights.

Splitting Dataset into Train & Test

[93]:
```
## Dataset information prior to Split
print('Dataset information prior to Split\n')
print('Size of the data: ', df_new.size)
print('Shape of the data: ', df_new.shape)
print('Number of columns in the data: ', len(df_new.columns))
result_id = df_new.groupby(['subject_id']).mean().reset_index()
print('Number of "subjects" in the data: ', len(result_id))
```

```
result_act = df_new.groupby(['activityID']).mean().reset_index()
print('Number of "activities" in the data: ',len(result_act))
```

```
Dataset information prior to Split

Size of the data:  6668924
Shape of the data:  (175498, 38)
Number of columns in the data:  38
Number of "subjects" in the data:  9
Number of "activities" in the data:  12
```

[94]:
```
## Splitting Dataset into Training & Testing
my_training_data = df_new.sample(frac=0.8, random_state=1)
my_testing_data = df_new.drop(my_training_data.index)
```

Perfore performing any EDA, first the dataset is split into "Testing" and "Training" datasets. Here, **80%** of the dataset is kept as training dataset and the rest is considered for testing. This is considerd as an industry standard for most Machine learining models. In any scenario, the model has to be predicting the inputs accurately. So, the model should not be having all the dataset while training as the performance of the same can be evaluated only by unseen datas.

[95]:
```
## Dataset information after Split
print('Dataset information after split :\n')
print('Size of the data: ', my_training_data.size)
print('Shape of the data: ', my_training_data.shape)
print('Number of columns in the data: ', len(my_training_data.columns))
result_id = my_training_data.groupby(['subject_id']).mean().reset_index()
print('Number of "subjects" in the data: ', len(result_id))
result_act = my_training_data.groupby(['activityID']).mean().reset_index()
print('Number of "activities" in the data: ',len(result_act))
```

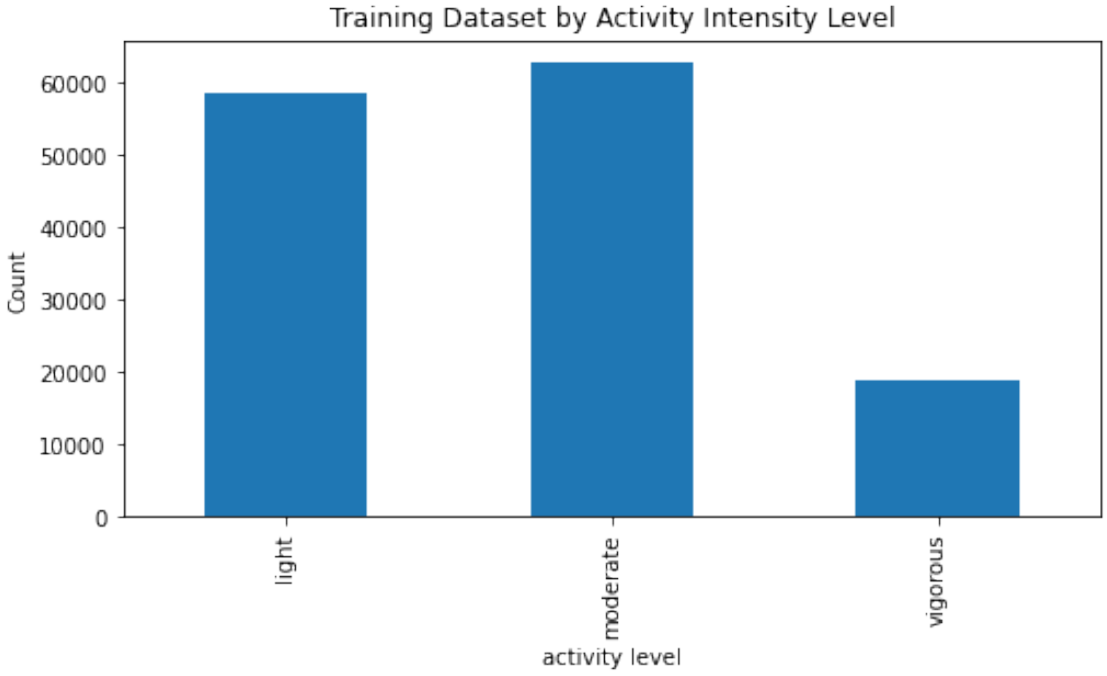
```
Dataset information after split :

Size of the data:  5335124
Shape of the data:  (140398, 38)
Number of columns in the data:  38
Number of "subjects" in the data:  9
Number of "activities" in the data:  12
```

Taking a look at the above list, the information to create our model is understood.

[96]:
```
## plotting Training Dataset by Activity Intensity
activity_class = my_training_data.groupby('activity_level').
 ↪count()['activityID']
print(('Training Dataset by Activity Intensity'))
display(activity_class)

ax = activity_class.plot(kind='bar', figsize=(8,4))
_ = ax.set_ylabel('Count')
```

```
_ = ax.set_xlabel('activity level')
_ = ax.set_title('Training Dataset by Activity Intensity Level')
```

```
Training Dataset by Activity Intensity

activity_level
light       58686
moderate    62731
vigorous    18981
Name: activityID, dtype: int64
```



Here, Individuals performing various activities based on the MET tags is displayed.

Data Inbalance Analysis

[97]:
```
print("Training data is displayed below {}".format(my_training_data.head()))
```

```
Training data is displayed below         timestamp       activityID  …
subject_id  activity_level
119017     1491.67  vacuum cleaning  …         106        moderate
20674      3258.58          running  …         101        vigorous
1052        152.88            lying  …         101           light
111406      576.84          sitting  …         106           light
98999      2429.16          walking  …         105        moderate

[5 rows x 38 columns]
```

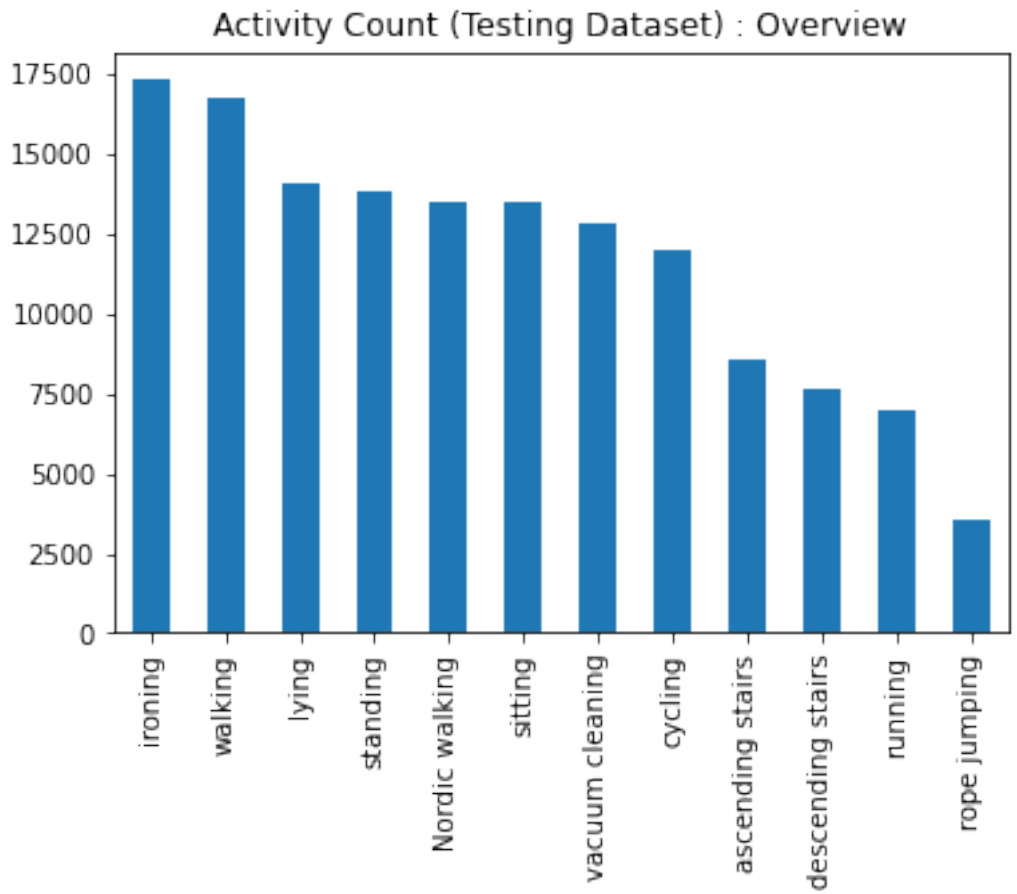
[98]:
```
print("Various statistics of Training data is displayed below {}".
 ↪format(my_training_data.head()))
```

```
Various statistics of Training data is displayed below         timestamp
activityID  …  subject_id  activity_level
119017    1491.67  vacuum cleaning  …          106        moderate
20674     3258.58           running  …          101        vigorous
1052       152.88             lying  …          101           light
111406     576.84           sitting  …          106           light
98999     2429.16           walking  …          105        moderate

[5 rows x 38 columns]
```

Correlation of every columns are easily observed by using the Pandas correlation method.By default, Pearsons correlation method is considered for calculating correlation.

[99]:
```
## Training Dataset Information by activities
print(('Training Dataset Information'))
display(my_training_data.groupby(['activityID'])['activityID'].count())
## Plotting Activity counts
plt.title('Activity Count (Testing Dataset) : Overview')
my_training_data.activityID.value_counts().plot(kind='bar')
plt.show()
```

```
Training Dataset Information

activityID
Nordic walking       13505
ascending stairs      8533
cycling              11980
descending stairs     7681
ironing              17305
lying                14061
rope jumping          3516
running               6932
sitting              13494
standing             13826
vacuum cleaning      12857
walking              16708
Name: activityID, dtype: int64
```



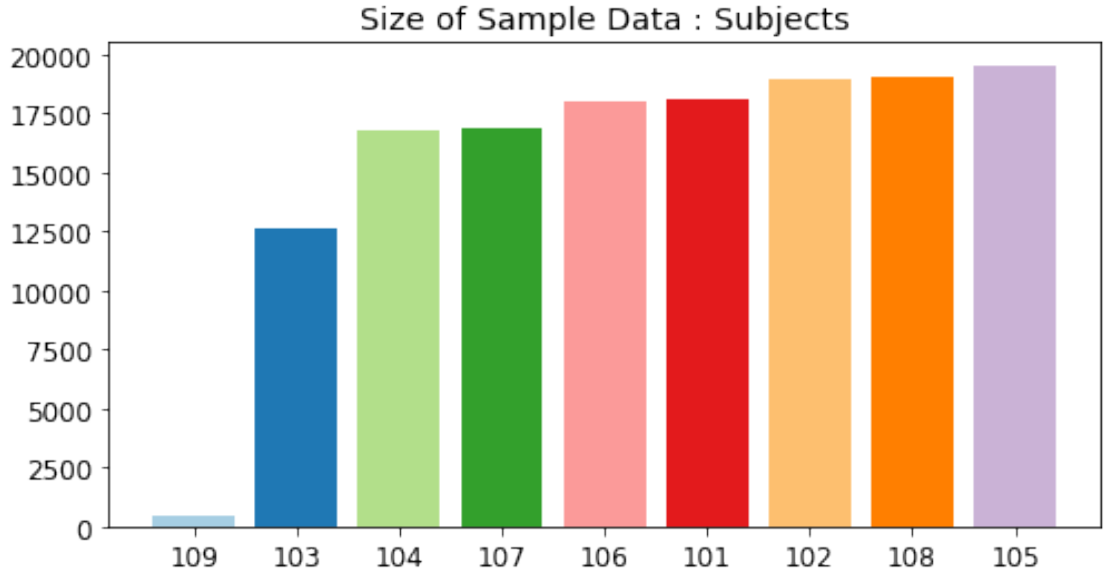
From above table, it is evident that **ironing** and **walking** has most number of data points followed by *lying* and *standing*.

0.2.1 EDA

The correlation between various featues are identified by plotting various graphs/ charts. These visualizations would help us in gaining valuable insights.

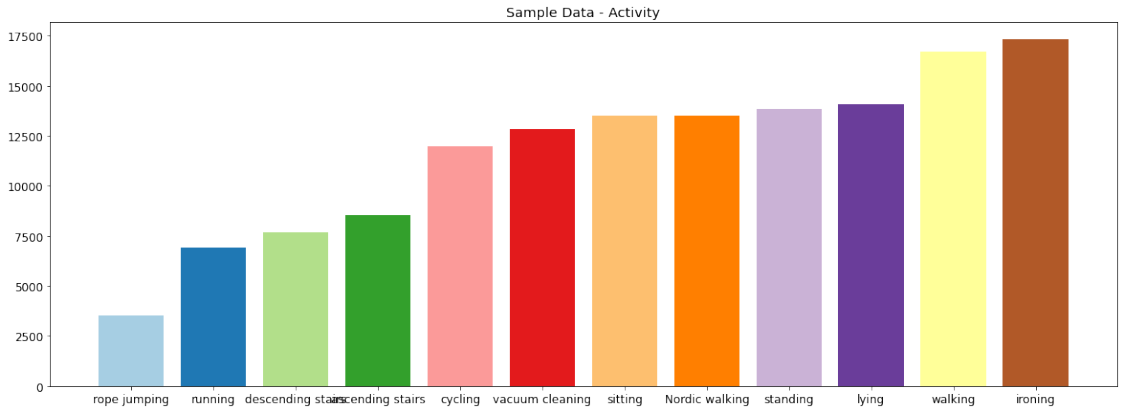
[100]:
```
## function to plot various graphs
def my_pandas_plot(param,column_a,column_b,title, figsize=(8,4)):
    plt.rcParams.update({'font.size': 12})
    size = range(len(param))
    f, ax = plt.subplots(figsize=figsize)
    plt.bar(size, param[column_a], color=plt.cm.Paired(size))
    a = ax.set_xticklabels(param[column_b])
    #b = ax.legend(fontsize = 15)
    c = ax.set_xticks(np.arange(len(param)))
    d = ax.set_title(title)
    plt.show()
```

[101]:
```
## plotting samples_to_subject
samples = my_training_data.groupby(['subject_id']).count().reset_index()
samples_to_subject = pd.DataFrame()
samples_to_subject['subject_id'] = samples['subject_id']
samples_to_subject['samples'] = samples['timestamp']
samples_to_subject = samples_to_subject.sort_values(by=['samples'])
my_pandas_plot(samples_to_subject,'samples','subject_id','Size of Sample Data :␣
↪Subjects')
plt.show()
```



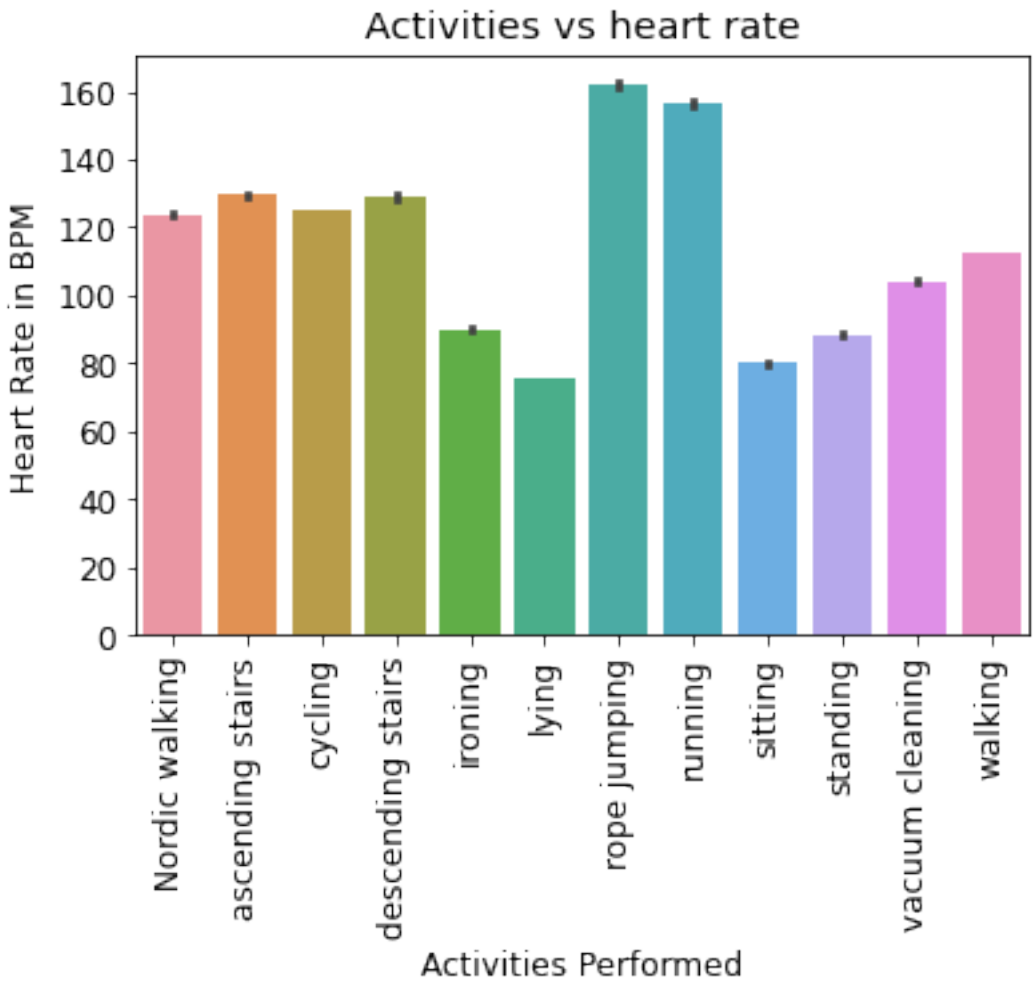
'**Subject 9'** has very few data whereas'**Subject 5**' has got the maximum datapoints.

[102]:
```
# plotting samples_to_activities
samples = my_training_data.groupby(['activityID']).count().reset_index()
samples_to_activities = pd.DataFrame()
samples_to_activities['activity'] = samples['activityID']
samples_to_activities['samples'] = samples['timestamp']
samples_to_activities = samples_to_activities.sort_values(by=['samples'])
my_pandas_plot(samples_to_activities,'samples','activity','Sample Data -␣
↪Activity',figsize=(20,7))
plt.show()
```



'**rope jumping**' activity got least samples than other activities and '**ironing**' and '**walking**' has highest datapoints.
From the above graphs, It is clear that provided dataset has got class inbalance. Now, heart rate details are compared.

[103]:
```
## plotting heart rate against each activities
heart_rate_data=my_training_data[['activityID','heart_rate']]
heart_rate_data.activityID=my_training_data.activityID.astype("category")
plt.xticks(rotation=90)
sns.barplot(x='activityID',y='heart_rate',data=heart_rate_data)
plt.ylabel('Heart Rate in BPM')
plt.xlabel('Activities Performed')
plt.title('Activities vs heart rate')
plt.show()
```

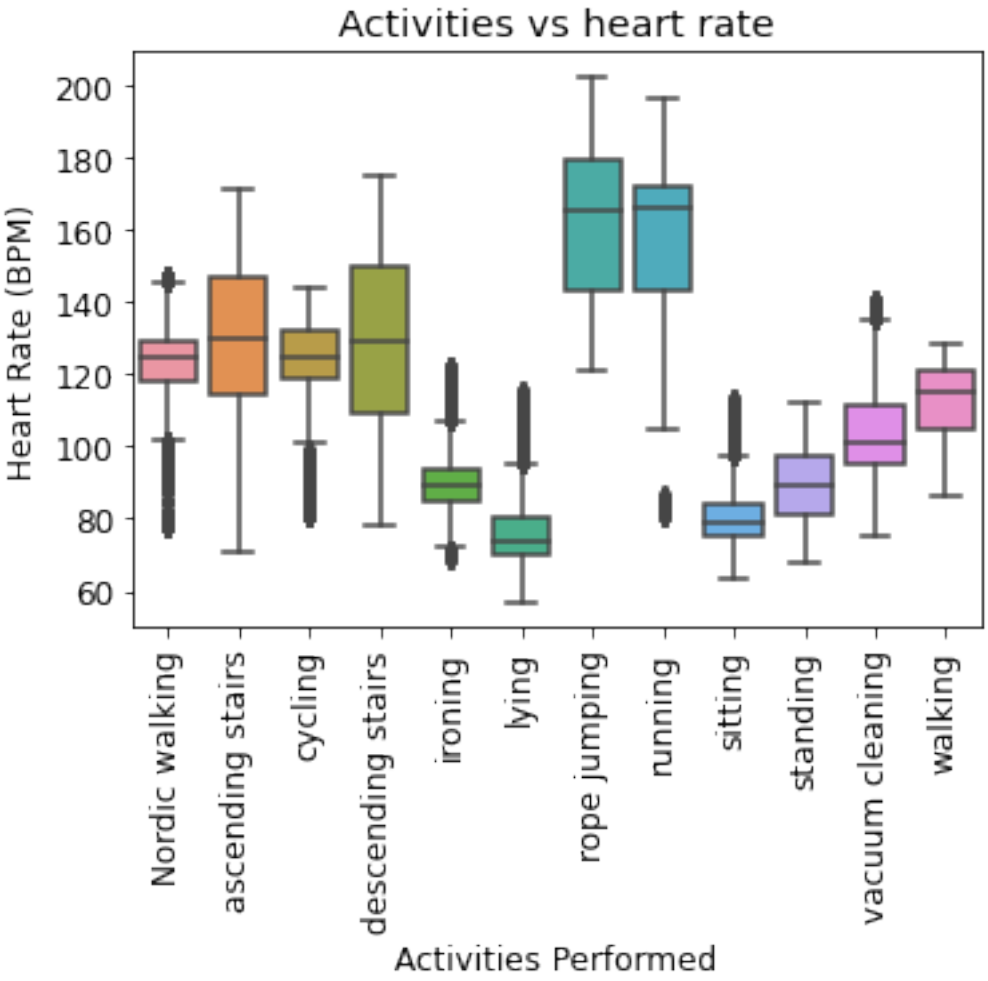


Above graph illustrates **performed activity** and its corresponding **heart rates**.It is clear that '**rope jumping**' is the most severe activity followed by '**running**'. Also, '**lying**' is the least severe among the lot.

EDA based on Heart Rate More detailed analysis is performed below, giving focus on Heart rate.

[104]:
```
## plotting Activities vs heart rate
heart_rate_data.activityID=heart_rate_data.activityID.astype("category")
plt.figure(figsize=(5,3))
plt.subplots_adjust(2,1,5,2)
plt.subplot(131)

plt.xticks(rotation=90)
sns.boxplot(x='activityID',y='heart_rate',data=heart_rate_data)
plt.ylabel('Heart Rate (BPM)')
```

```
plt.xlabel('Activities Performed')
plt.title('Activities vs heart rate')
plt.show()
```

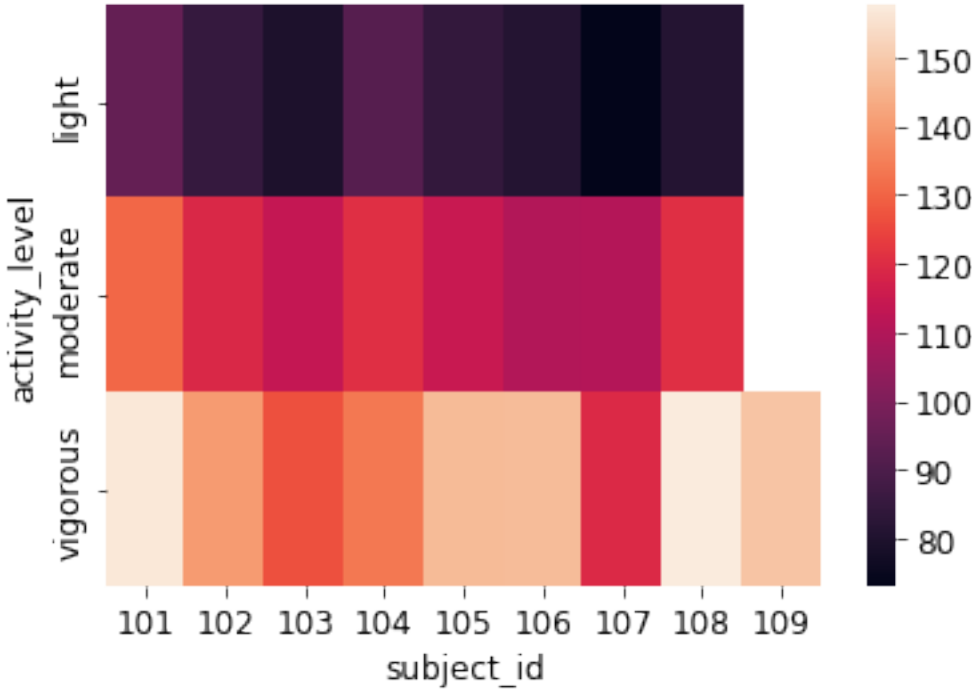


From above box plot, the relation between various activities and its corresponding heart rate is understood. Also,the outliers for each corresponding activities are visible.The need for classifying the activities according to their severity is well understood here. Further visualization is possible with heat maps for better understanding of relation between heart rates and activities, based on MET tags. On a promising note, the outliers are present only for certain activities.

[105]:
```
## plotting heatmap for Activity intensity of various subjects based on heart␣
↪rates
act_heart_df = my_training_data.groupby(['activity_level','subject_id'],␣
↪as_index=False).mean() \
            [['activity_level','subject_id','heart_rate']].
↪pivot(index='activity_level', columns='subject_id', values='heart_rate')
```

```
sns.heatmap(act_heart_df)
```

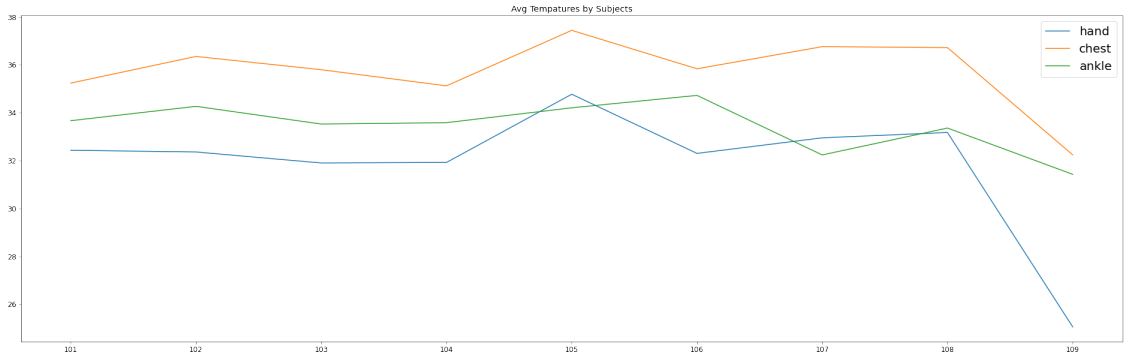
[105]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc8a30c5b50>



From the above heat map, the relationship between activity intensity of each subjects based on their heart rates is understood. There is a significant relation between heart rates and activity level for vigorous and moderate tags.

Explorations based on Temperature Now, further explorations are carried out based on Temperature and various features.

[106]:
```
## plotting hand,chest and ankle temperature of various subjects
samples_temp = pd.DataFrame()
samples_temp['hand'] = result_id['hand_temperature']
samples_temp['chest'] = result_id['chest_temperature']
samples_temp['ankle'] = result_id['ankle_temperature']

ax = samples_temp.plot(kind='line', figsize=(33,10), title='Avg Tempatures by␣
↪Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
b = ax.legend(fontsize = 20)
c = ax.set_xticks(np.arange(len(samples_temp)))
```

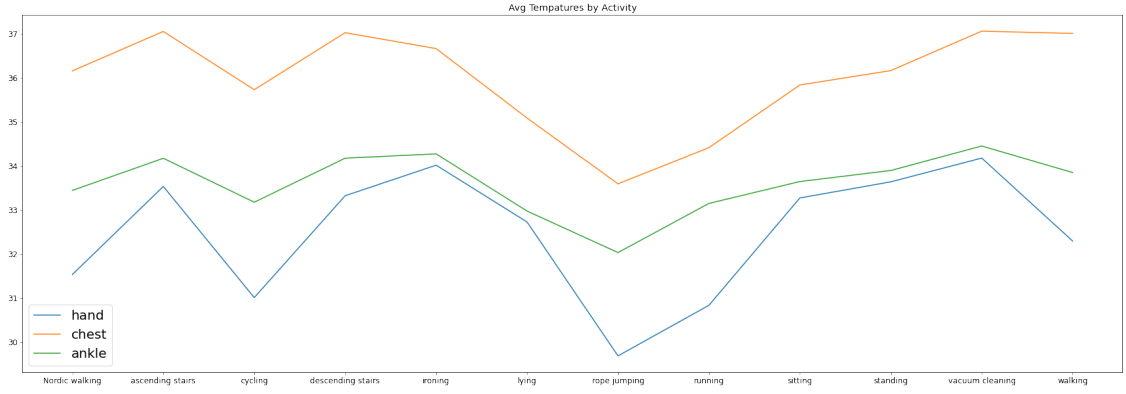


Above line graph shows three data :

- Hand Temperature
- Chest Temperature
- Ankle Temperature

'**subject5**' has the observed the highest temperature whereas '**subject9**' has very low temperature for some reason.

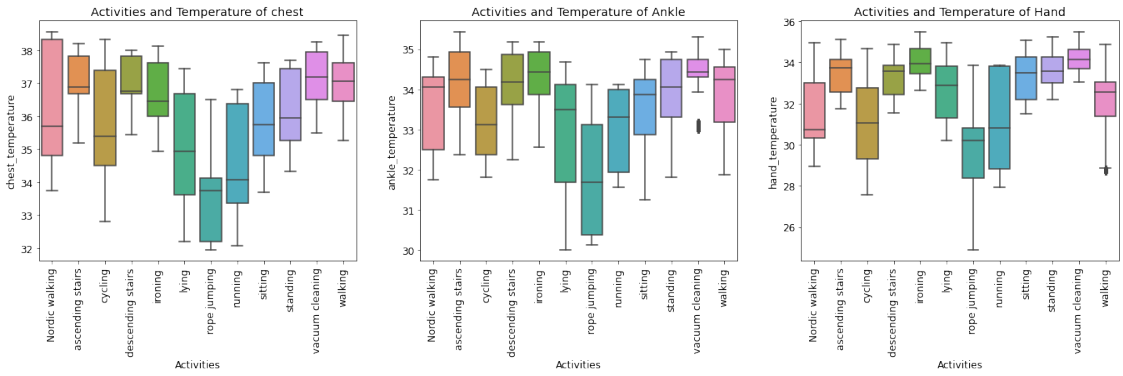
[107]:
```
activity_temp = pd.DataFrame()
activity_temp['activity'] = samples['activityID']
activity_temp['hand'] = result_act['hand_temperature']
activity_temp['chest'] = result_act['chest_temperature']
activity_temp['ankle'] = result_act['ankle_temperature']

ax = activity_temp.plot(kind='line', figsize=(30,10), title='Avg Tempatures by␣
↪Activity')
a = ax.set_xticklabels(activity_temp['activity'])
b = ax.legend(fontsize = 20)
c = ax.set_xticks(np.arange(len(activity_temp)))
```



From above graph, it is clear that '**Ironing**' and '**Vaccum cleaning**' will be higher temperature

and '**rope jumping**' will have the least.A better understanding of similarity can be obtained from the below box plots.

[108]:
```
#ploting activityID' and 'chest_temperature'

plt.subplots_adjust(2,1,5,2)
plt.subplot(131)
temp_data=my_training_data[['activityID','chest_temperature']]
temp_data.activityID=temp_data.activityID.astype("category")
plt.xticks(rotation=90)
sns.boxplot(x='activityID',y='chest_temperature',data=temp_data)
plt.ylabel('chest_temperature')
plt.xlabel('Activities')
plt.title('Activities and Temperature of chest')

#ploting activityID' and 'ankle_temperature'
plt.subplot(132)
temp_data=my_training_data[['activityID','ankle_temperature']]
temp_data.activityID=temp_data.activityID.astype("category")
plt.xticks(rotation=90)
sns.boxplot(x='activityID',y='ankle_temperature',data=temp_data)
plt.ylabel('ankle_temperature')
plt.xlabel('Activities')
plt.title('Activities and Temperature of Ankle')

#ploting activityID' and 'hand_temperature'
plt.subplot(133)
temp_data=my_training_data[['activityID','hand_temperature']]
temp_data.activityID=temp_data.activityID.astype("category")
plt.xticks(rotation=90)
sns.boxplot(x='activityID',y='hand_temperature',data=temp_data)
plt.ylabel('hand_temperature')
plt.xlabel('Activities')
plt.title('Activities and Temperature of Hand')

plt.show()
```



From the above box plots, it is clear that various activities influence **hand, chest and ankle temperatures**.
Hence, prediction of activity can be based on temperatures and this will be proved in the upcoming Hypothesis Test.

[109]:
```
## statistics for various temperature IMU
print('Stats for Chest Temperature IMU\n',my_training_data.chest_temperature.
 ↪agg(['mean','median','std']))
print('\nStats for Ankle Temperature IMU\n',my_training_data.ankle_temperature.
 ↪agg(['mean','median','std']))
print('\nStats for Hand Temperature IMU\n',my_training_data.hand_temperature.
 ↪agg(['mean','median','std']))
```

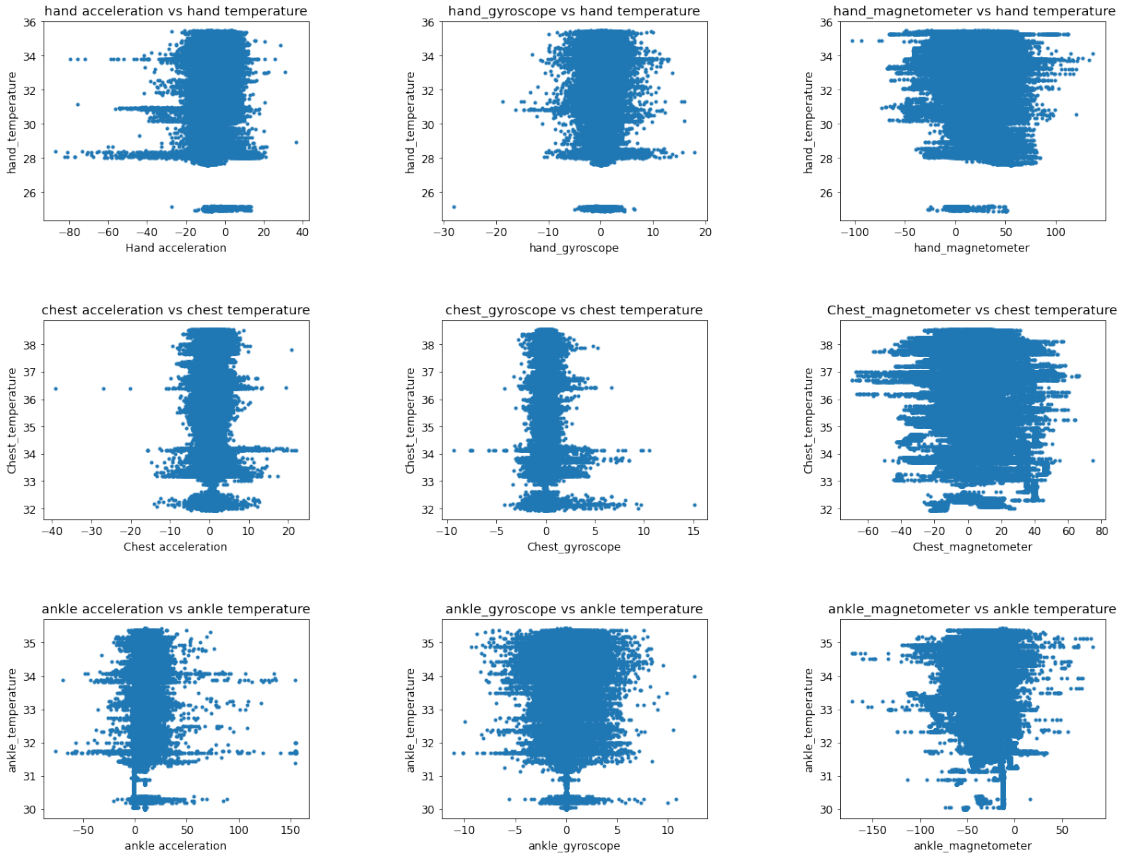
```
Stats for Chest Temperature IMU
 mean      36.179718
median    36.437500
std        1.492742
Name: chest_temperature, dtype: float64

Stats for Ankle Temperature IMU
 mean      33.713788
median    34.062500
std        1.092038
Name: ankle_temperature, dtype: float64

Stats for Hand Temperature IMU
 mean      32.759226
median    33.187500
std        1.791442
Name: hand_temperature, dtype: float64
```

From the above statistics, it is clear that the measurements with '**chest temperature IMU**' is accurate as the mean value of **36.2°C** is very close to typical human body temperature of **36.5–37 °C**. Before performing the hypothesis testing, possible relations between IMU's like *magnetometer*, *gyroscope* and *accelerometer* is verified by scatter plots.

[110]:
```
## plotting hand acceleration vs hand temperature
plt.figure(figsize=(8,6))
plt.subplots_adjust(2,3,4,5,.5,.5)
plt.subplot(331)
plt.scatter(my_training_data['hand_acceleration_16g␣
 ↪(x)'],my_training_data['hand_temperature'],marker='.')
plt.ylabel('hand_temperature')
plt.xlabel('Hand acceleration')
plt.title('hand acceleration vs hand temperature')
```

```
## plotting hand gyro vs hand temperature
plt.subplot(332)
plt.scatter(my_training_data['hand_gyroscope␣
 ↪(x)'],my_training_data['hand_temperature'],marker='.')
plt.ylabel('hand_temperature')
plt.xlabel('hand_gyroscope')
plt.title('hand_gyroscope vs hand temperature')

## plotting hand magnetometer vs hand temperature
plt.subplot(333)
plt.scatter(my_training_data['hand_magnetometer␣
 ↪(x)'],my_training_data['hand_temperature'],marker='.')
plt.ylabel('hand_temperature')
plt.xlabel('hand_magnetometer')
plt.title(' hand_magnetometer vs hand temperature')

## plotting chest acceleration vs chest temperature
plt.subplot(334)
plt.scatter(my_training_data['chest_acceleration_16g␣
 ↪(x)'],my_training_data['chest_temperature'],marker='.')
plt.ylabel('Chest_temperature')
plt.xlabel('Chest acceleration')
plt.title('chest acceleration vs chest temperature')

## plotting chest gyro vs chest temperature
plt.subplot(335)
plt.scatter(my_training_data['chest_gyroscope␣
 ↪(x)'],my_training_data['chest_temperature'],marker='.')
plt.ylabel('Chest_temperature')
plt.xlabel('Chest_gyroscope')
plt.title('chest_gyroscope vs chest temperature')

## plotting chest magnetometer vs chest temperature
plt.subplot(336)
plt.scatter(my_training_data['chest_magnetometer␣
 ↪(x)'],my_training_data['chest_temperature'],marker='.')
plt.ylabel('Chest_temperature')
plt.xlabel('Chest_magnetometer')
plt.title('Chest_magnetometer vs chest temperature')

## plotting ankle accelerometer vs ankle temperature
plt.subplot(337)
plt.scatter(my_training_data['ankle_acceleration_16g␣
 ↪(x)'],my_training_data['ankle_temperature'],marker='.')
plt.ylabel('ankle_temperature')
plt.xlabel('ankle acceleration')
```

```
plt.title('ankle acceleration vs ankle temperature')

## plotting ankle gyro vs ankle temperature
plt.subplot(338)
plt.scatter(my_training_data['ankle_gyroscope␣
 ↪(x)'],my_training_data['ankle_temperature'],marker='.')
plt.ylabel('ankle_temperature')
plt.xlabel('ankle_gyroscope')
plt.title('ankle_gyroscope vs ankle temperature')

## plotting ankle magnetometer vs ankle temperature
plt.subplot(339)
plt.scatter(my_training_data['ankle_magnetometer␣
 ↪(x)'],my_training_data['ankle_temperature'],marker='.')
plt.ylabel('ankle_temperature')
plt.xlabel('ankle_magnetometer')
plt.title('ankle_magnetometer vs ankle temperature')
plt.show()
```



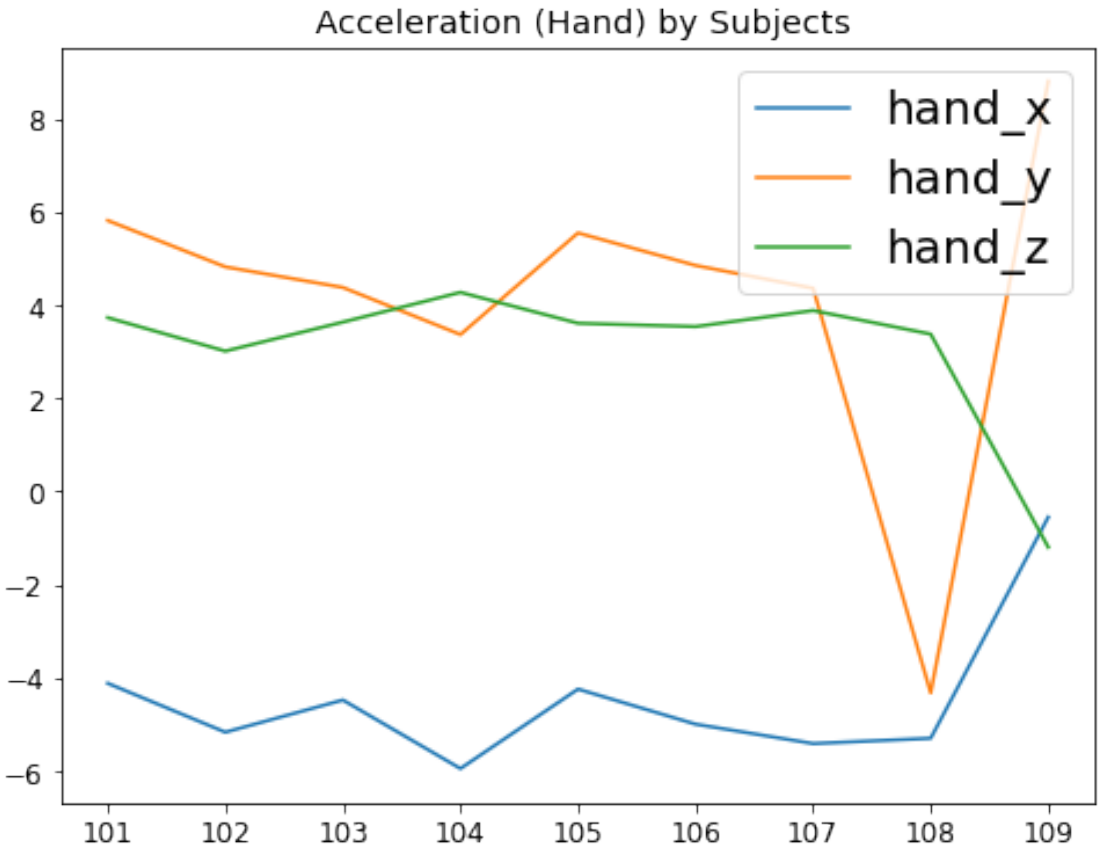
From above plots, the central tendency is clear, i.e. outliers are minimum for chest IMU. This

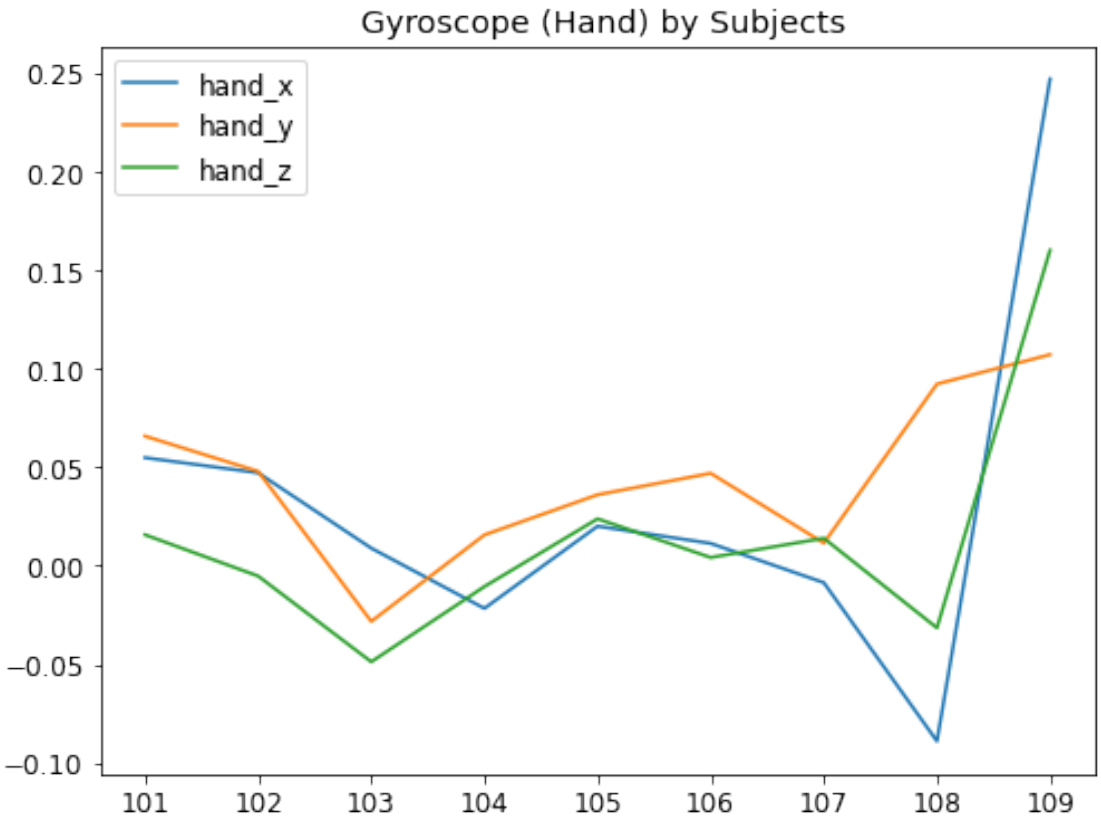
statement is also supported by the earlier standard deviation calculation.

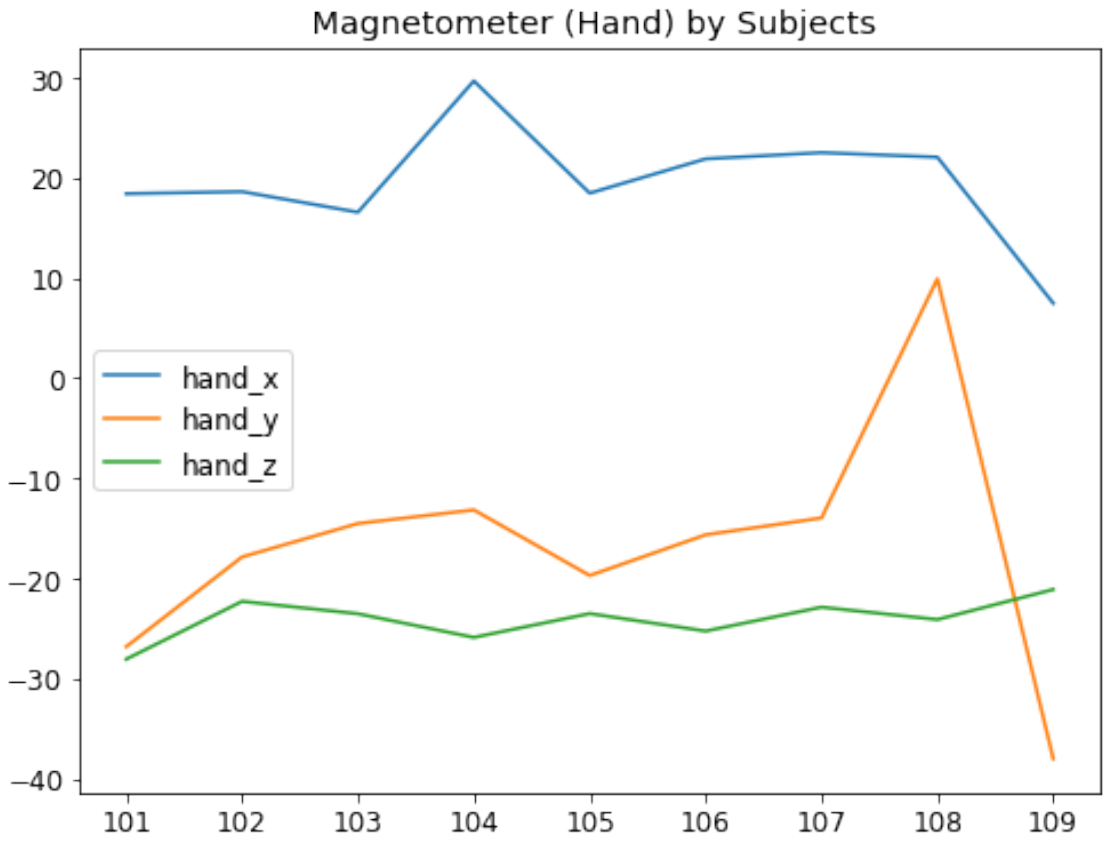
EDA on IMU's

- **Hand IMU sensor analysis by subjects**

[111]:
```
## plotting hand acceleration sensor analysis based on subjects
hand_sensor_data = pd.DataFrame()
hand_sensor_data['hand_x'] = result_id['hand_acceleration_16g (x)']
hand_sensor_data['hand_y'] = result_id['hand_acceleration_16g (y)']
hand_sensor_data['hand_z'] = result_id['hand_acceleration_16g (z)']
ax = hand_sensor_data.plot(kind='line', figsize=(8,6), title=' Acceleration␣
 ↪(Hand) by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
b = ax.legend(fontsize = 20)
c = ax.set_xticks(np.arange(len(hand_sensor_data)))

## plotting hand gyroscope sensor analysis based on subjects
hand_gyro_data = pd.DataFrame()
hand_gyro_data['hand_x'] = result_id['hand_gyroscope (x)']
hand_gyro_data['hand_y'] = result_id['hand_gyroscope (y)']
hand_gyro_data['hand_z'] = result_id['hand_gyroscope (z)']
ax = hand_gyro_data.plot(kind='line', figsize=(8,6), title=' Gyroscope (Hand)␣
 ↪by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
c = ax.set_xticks(np.arange(len(hand_sensor_data)))

## plotting hand magnetometer sensor analysis based on subjects
hand_mag_data = pd.DataFrame()
hand_mag_data['hand_x'] = result_id['hand_magnetometer (x)']
hand_mag_data['hand_y'] = result_id['hand_magnetometer (y)']
hand_mag_data['hand_z'] = result_id['hand_magnetometer (z)']
ax = hand_mag_data.plot(kind='line', figsize=(8,6), title=' Magnetometer (Hand)␣
 ↪by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
c = ax.set_xticks(np.arange(len(hand_sensor_data)))
```


Acceleration (Hand) by Subjects
hand_x
hand_y
hand_z
8
6
4
2
0
−2
−4
−6
101
102
103
104
105
106
107
108
109


Gyroscope (Hand) by Subjects
hand_x
hand_y
hand_z
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
101
102
103
104
105
106
107
108
109



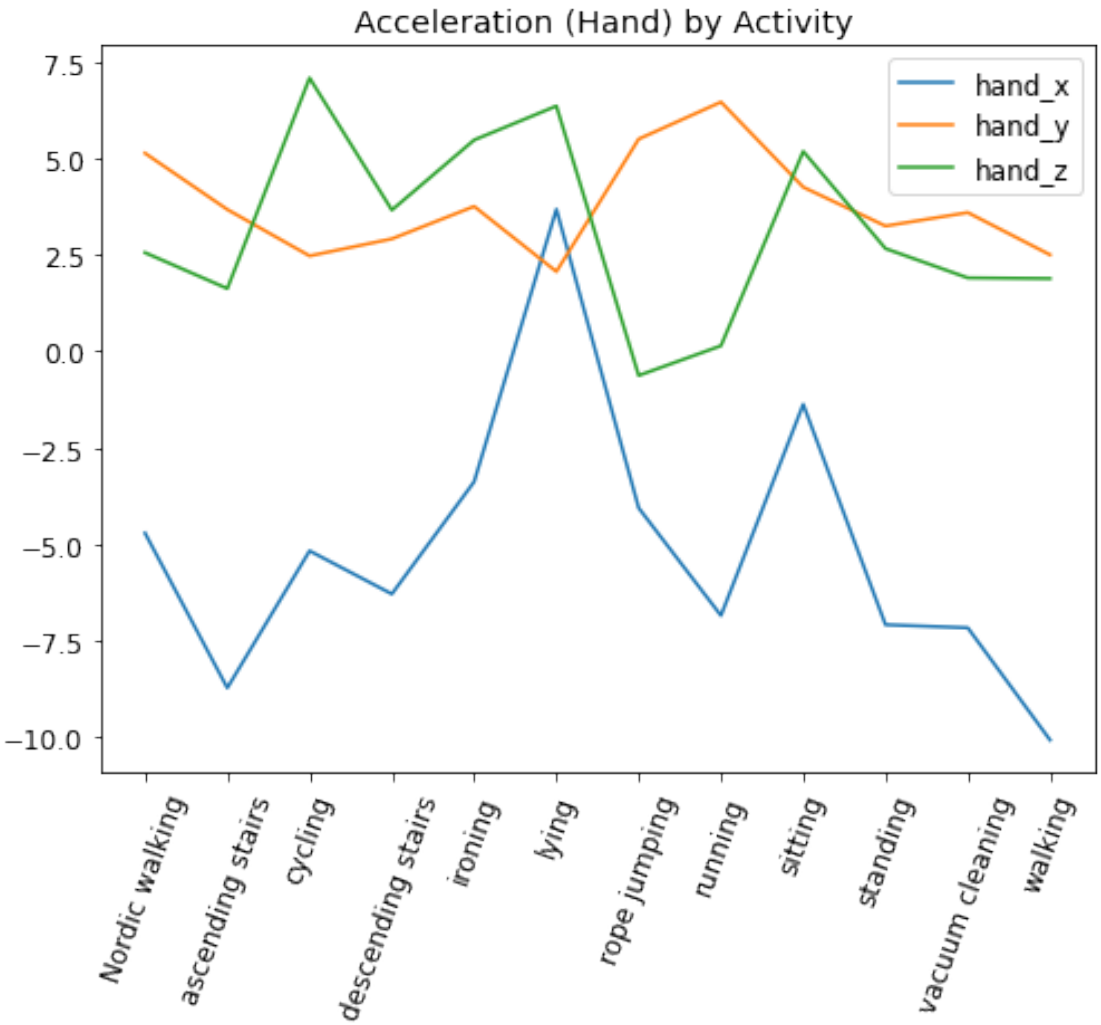
From above, all the subjects hand accelerometer sensor are showing similar readings however there is something strange in **subject 8** readings, especially **'y'** reading is off the chart for some reason. For Gyroscope, subject3 and subject 8 is showing variations in readings from x and y axis.

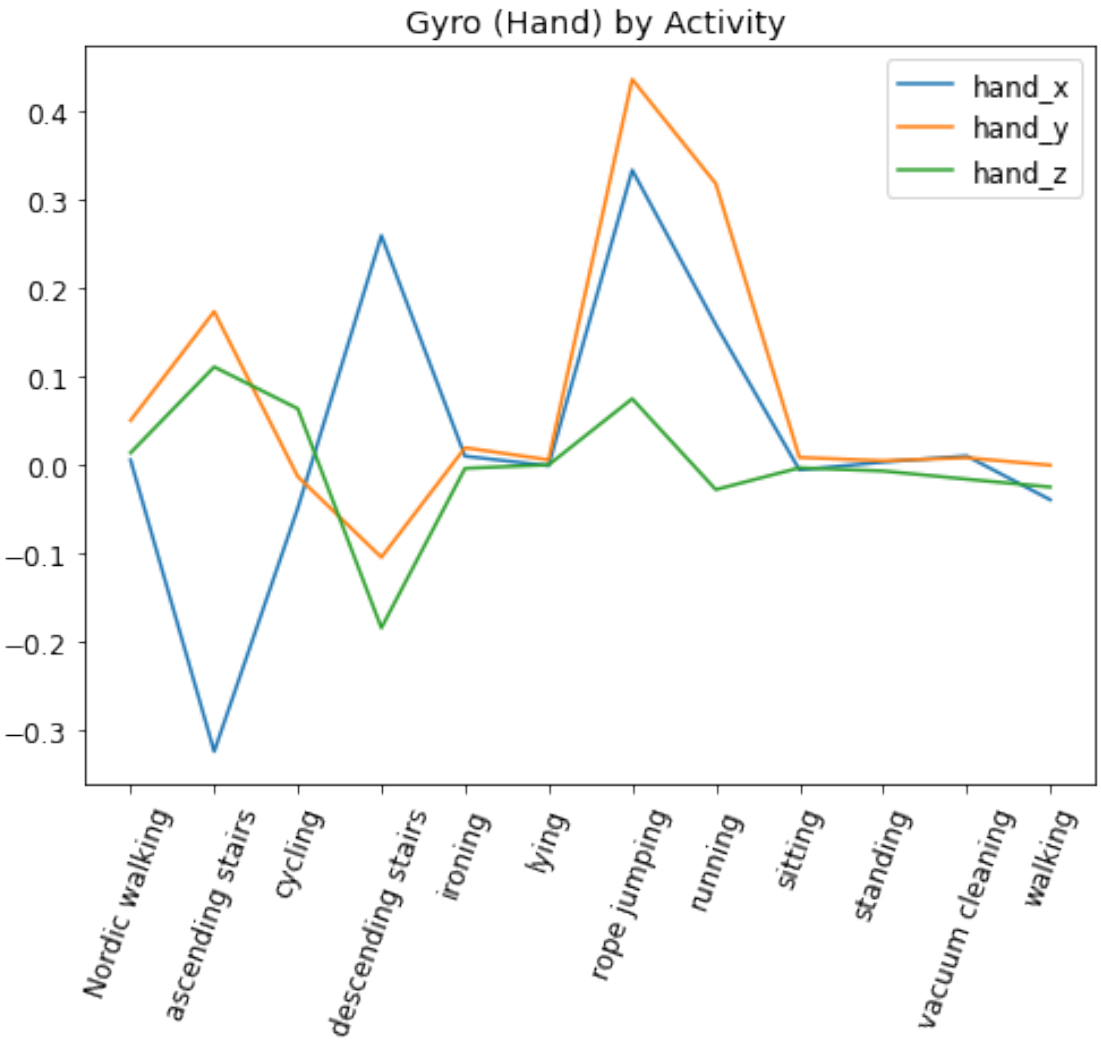
This could be due to sensor calibration issue.The challenges in integrity of datacollection is proved here, as many sensors are delivering incorrect readings at particular axis. Hence, there should be a real time sanity check of outputs while recording the data.

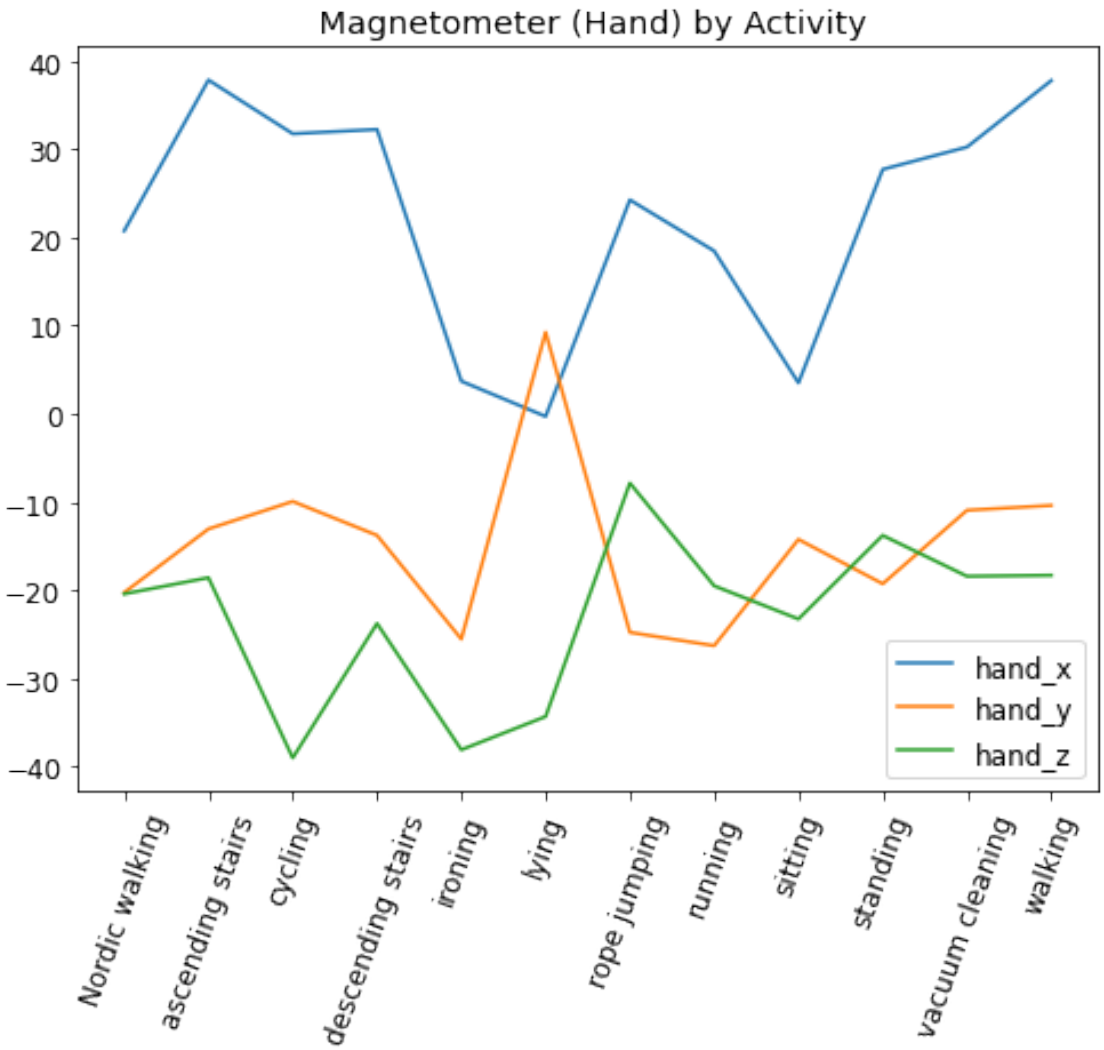
Hand IMU sensor analysis by activity

[112]:
```
## plotting hand acceleration sensor analysis based on activity

hand_sensor_activity = pd.DataFrame()
hand_sensor_activity['activity'] = samples['activityID']
hand_sensor_activity['hand_x'] = result_act['hand_acceleration_16g (x)']
hand_sensor_activity['hand_y'] = result_act['hand_acceleration_16g (y)']
hand_sensor_activity['hand_z'] = result_act['hand_acceleration_16g (z)']
ax = hand_sensor_activity.plot(kind='line', figsize=(8,6), title=' Acceleration␣
↪(Hand) by Activity')
a = ax.set_xticklabels(hand_sensor_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(hand_sensor_activity)))
```

```
## plotting hand gyroscope sensor analysis based on activity
hand_gyro_activity = pd.DataFrame()
hand_gyro_activity['activity'] = samples['activityID']
hand_gyro_activity['hand_x'] = result_act['hand_gyroscope (x)']
hand_gyro_activity['hand_y'] = result_act['hand_gyroscope (y)']
hand_gyro_activity['hand_z'] = result_act['hand_gyroscope (z)']
ax = hand_gyro_activity.plot(kind='line', figsize=(8,6), title=' Gyro (Hand) by␣
↪Activity')
a = ax.set_xticklabels(hand_gyro_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(hand_gyro_activity)))

## plotting hand magnetometer sensor analysis based on activity
hand_magn_activity = pd.DataFrame()
hand_magn_activity['activity'] = samples['activityID']
hand_magn_activity['hand_x'] = result_act['hand_magnetometer (x)']
hand_magn_activity['hand_y'] = result_act['hand_magnetometer (y)']
hand_magn_activity['hand_z'] = result_act['hand_magnetometer (z)']
ax = hand_magn_activity.plot(kind='line', figsize=(8,6), title=' Magnetometer␣
↪(Hand) by Activity')
a = ax.set_xticklabels(hand_magn_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(hand_magn_activity)))
plt.show()
```


Acceleration (Hand) by Activity
hand_x
hand_y
hand_z
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking


Gyro (Hand) by Activity
hand_x
hand_y
hand_z
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking

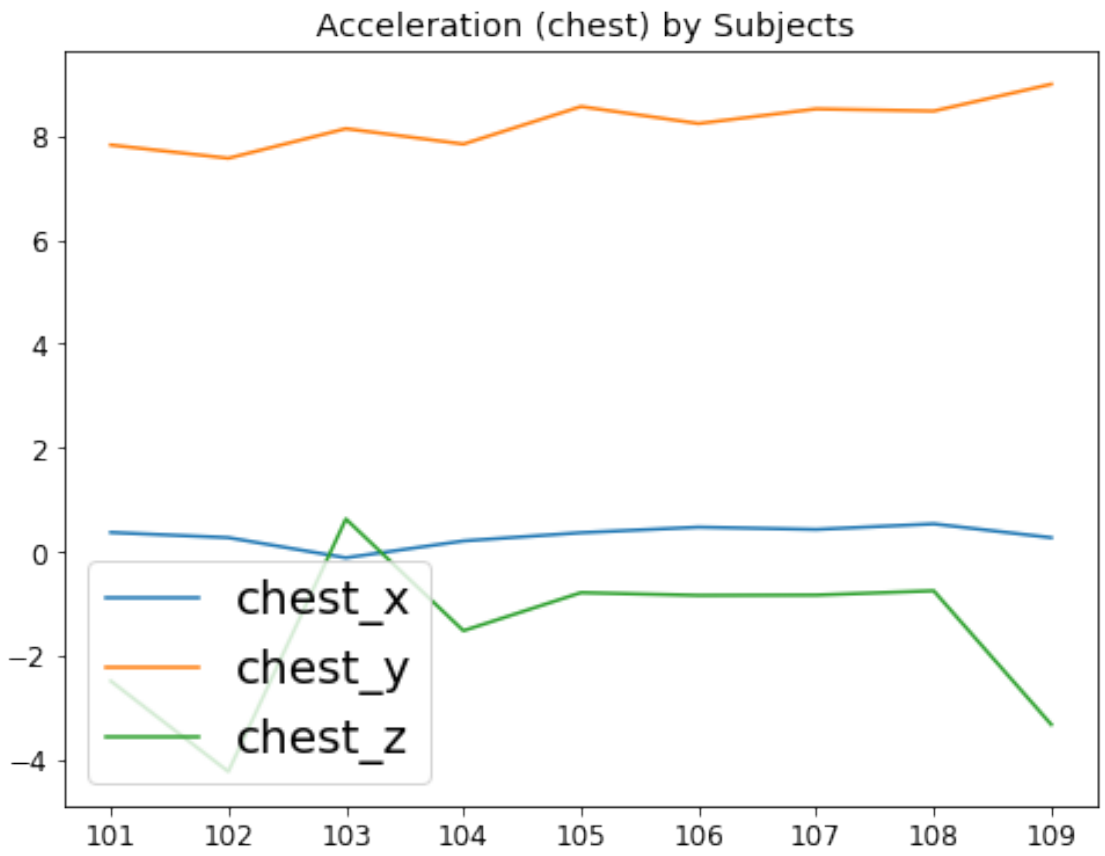


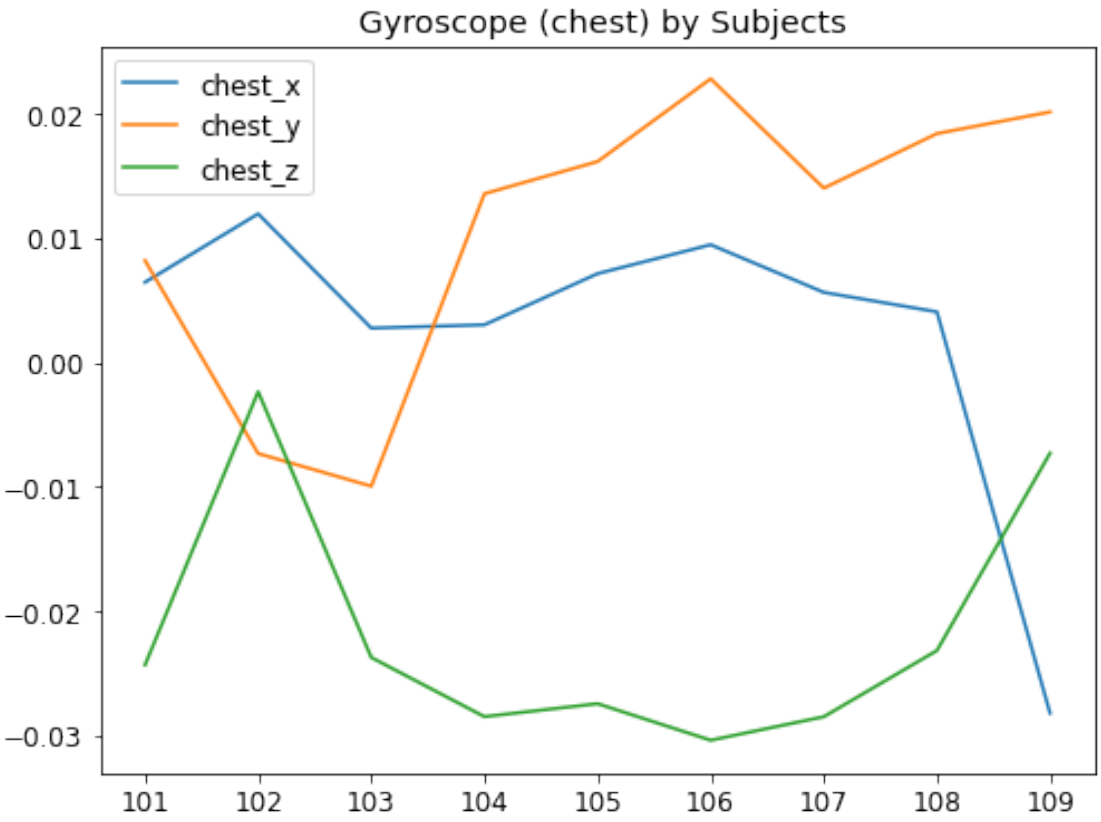
From above IMU data, it is not surprising to see that each activity has got a disctinct reading. From these, IMU's role in predicting an activity is evident. Infact, it can be considered as the most significant feature for activity recognition only after heart rate. The trend will remain same for the chest as well as ankle, which is seen in later stage.

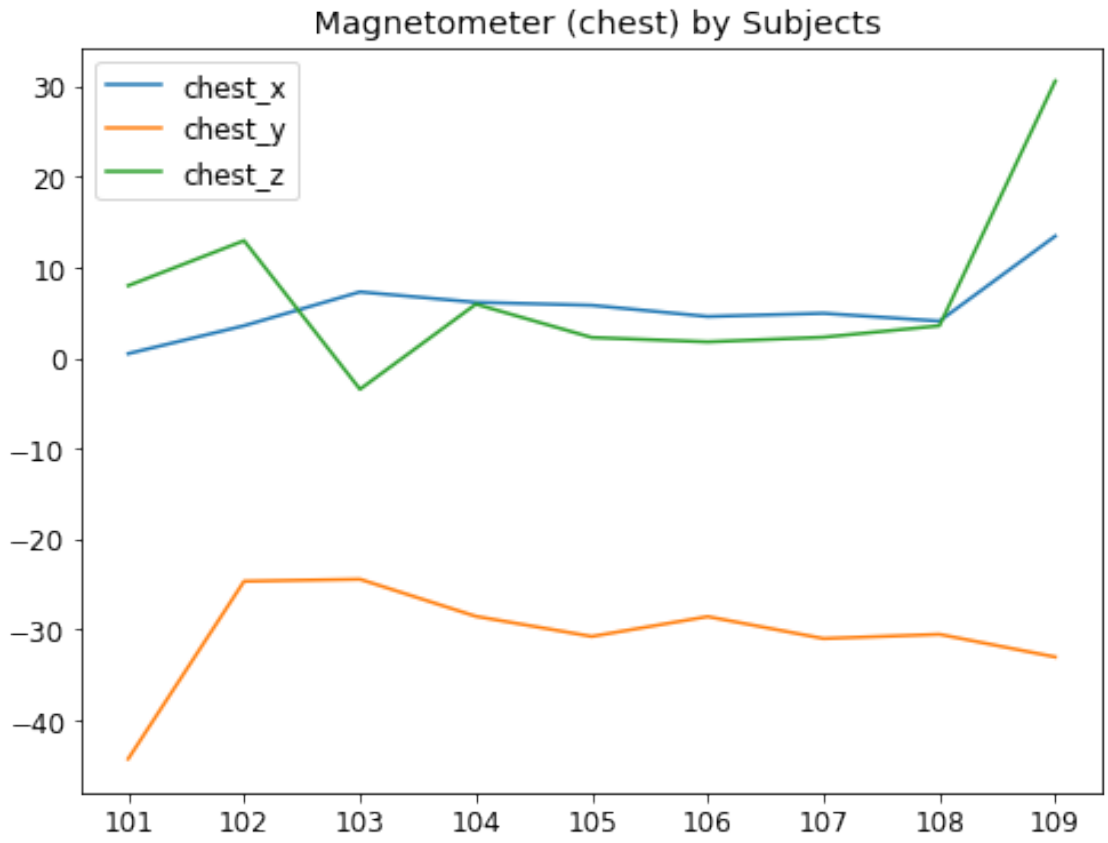
- **chest IMU sensor analysis by subjects**

[113]:
```
## plotting chest acceleration sensor analysis based on subjects
chest_sensor_data = pd.DataFrame()
chest_sensor_data['chest_x'] = result_id['chest_acceleration_16g (x)']
chest_sensor_data['chest_y'] = result_id['chest_acceleration_16g (y)']
chest_sensor_data['chest_z'] = result_id['chest_acceleration_16g (z)']
ax = chest_sensor_data.plot(kind='line', figsize=(8,6), title=' Acceleration␣
↪(chest) by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
```

```
b = ax.legend(fontsize = 20)
c = ax.set_xticks(np.arange(len(chest_sensor_data)))

## plotting chest gyroscope sensor analysis based on subjects
chest_gyro_data = pd.DataFrame()
chest_gyro_data['chest_x'] = result_id['chest_gyroscope (x)']
chest_gyro_data['chest_y'] = result_id['chest_gyroscope (y)']
chest_gyro_data['chest_z'] = result_id['chest_gyroscope (z)']
ax = chest_gyro_data.plot(kind='line', figsize=(8,6), title=' Gyroscope (chest)␣
 ↪by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
c = ax.set_xticks(np.arange(len(chest_sensor_data)))

## plotting chest magnetometer sensor analysis based on subjects
chest_mag_data = pd.DataFrame()
chest_mag_data['chest_x'] = result_id['chest_magnetometer (x)']
chest_mag_data['chest_y'] = result_id['chest_magnetometer (y)']
chest_mag_data['chest_z'] = result_id['chest_magnetometer (z)']
ax = chest_mag_data.plot(kind='line', figsize=(8,6), title=' Magnetometer␣
 ↪(chest) by Subjects')
a = ax.set_xticklabels(result_id['subject_id'])
c = ax.set_xticks(np.arange(len(chest_sensor_data)))
```




Gyroscope (chest) by Subjects
chest_x
chest_y
chest_z
0.02
0.01
0.00
−0.01
−0.02
−0.03
101
102
103
104
105
106
107
108
109

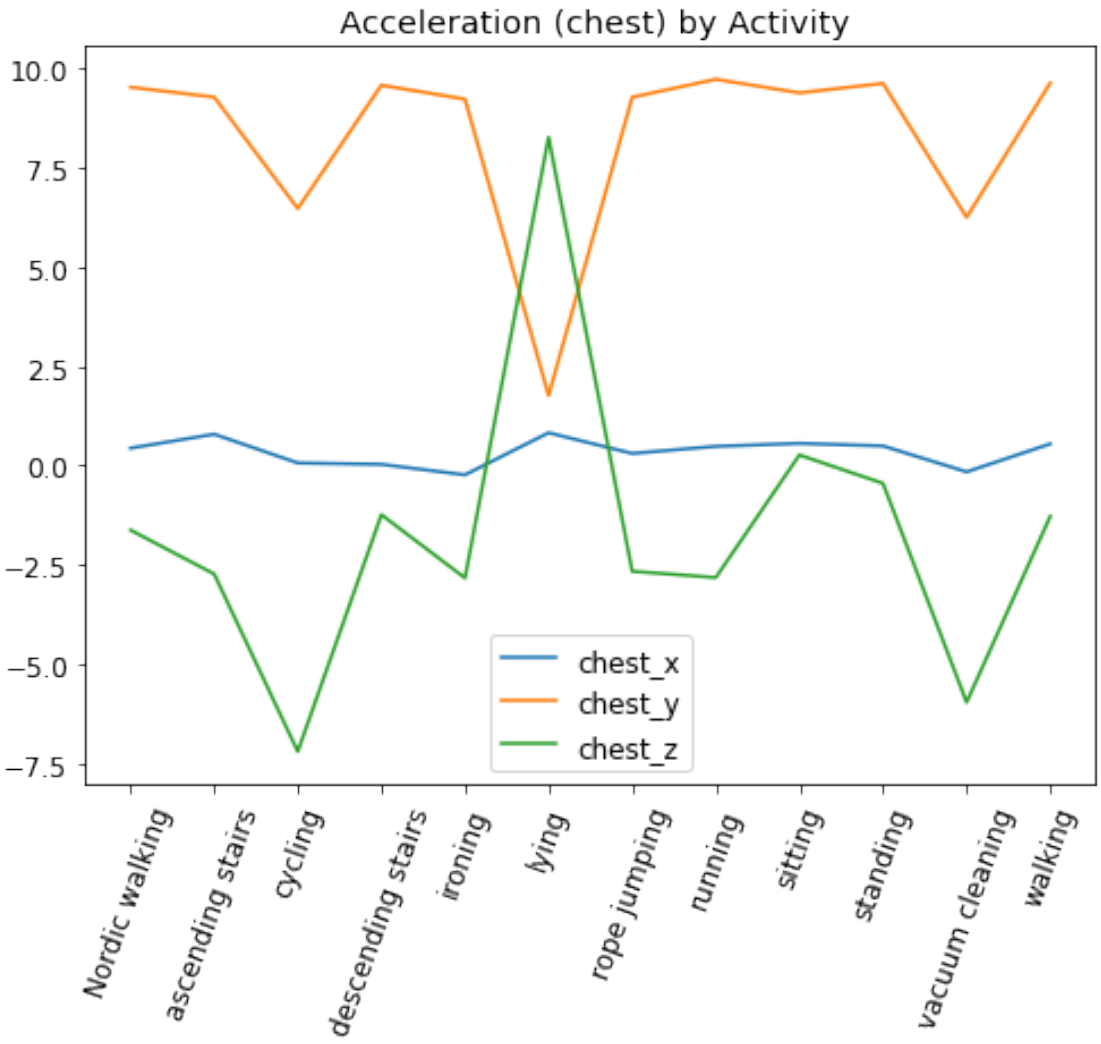


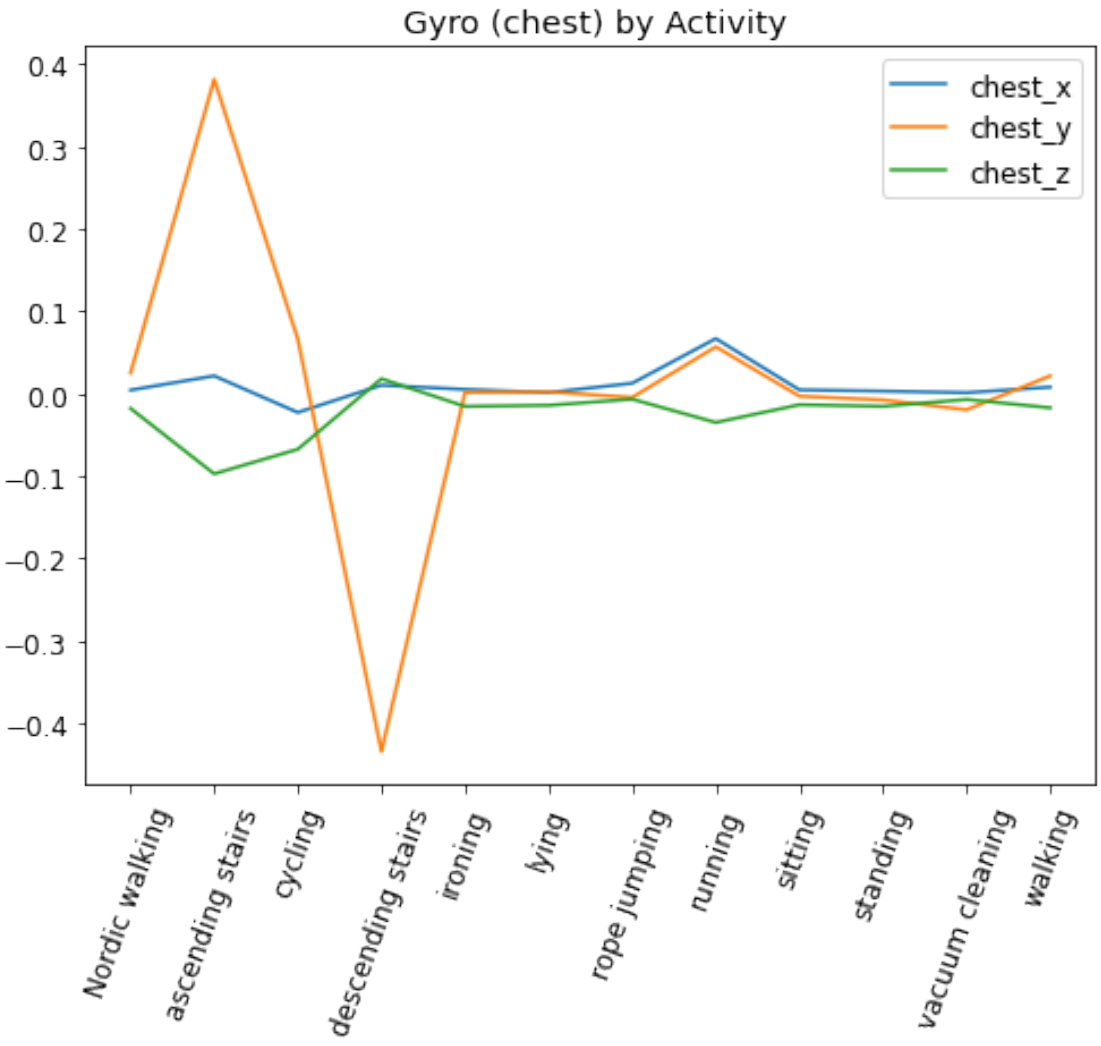
There is a slight irregularity in subject3's accelerometer z axis data. In gyroscope, we can see that every individual has got distinct patterns. Magnetometer readings seems to be similar except for subject 3 and 8 z axis reading.

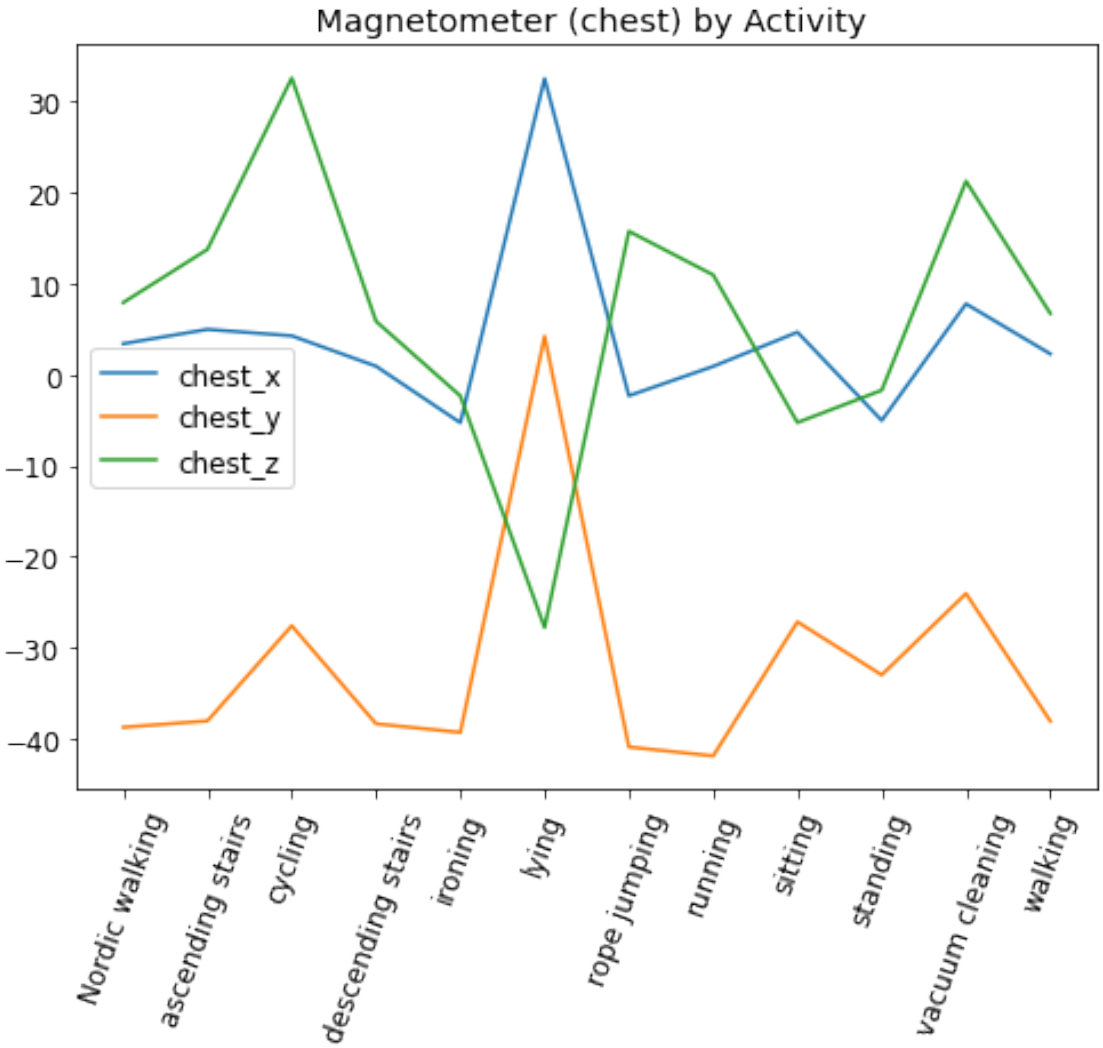
- **chest IMU sensor analysis by activities**

[114]:
```
## plotting chest acceleration sensor analysis based on activity

chest_sensor_activity = pd.DataFrame()
chest_sensor_activity['activity'] = samples['activityID']
chest_sensor_activity['chest_x'] = result_act['chest_acceleration_16g (x)']
chest_sensor_activity['chest_y'] = result_act['chest_acceleration_16g (y)']
chest_sensor_activity['chest_z'] = result_act['chest_acceleration_16g (z)']
ax = chest_sensor_activity.plot(kind='line', figsize=(8,6), title='␣
 ↪Acceleration (chest) by Activity')
a = ax.set_xticklabels(chest_sensor_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(chest_sensor_activity)))

## plotting chest gyroscope sensor analysis based on activity
chest_gyro_activity = pd.DataFrame()
chest_gyro_activity['activity'] = samples['activityID']
```

```
chest_gyro_activity['chest_x'] = result_act['chest_gyroscope (x)']
chest_gyro_activity['chest_y'] = result_act['chest_gyroscope (y)']
chest_gyro_activity['chest_z'] = result_act['chest_gyroscope (z)']
ax = chest_gyro_activity.plot(kind='line', figsize=(8,6), title=' Gyro (chest)␣
↪by Activity')
a = ax.set_xticklabels(chest_gyro_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(chest_gyro_activity)))

## plotting chest magnetometer sensor analysis based on activity
chest_magn_activity = pd.DataFrame()
chest_magn_activity['activity'] = samples['activityID']
chest_magn_activity['chest_x'] = result_act['chest_magnetometer (x)']
chest_magn_activity['chest_y'] = result_act['chest_magnetometer (y)']
chest_magn_activity['chest_z'] = result_act['chest_magnetometer (z)']
ax = chest_magn_activity.plot(kind='line', figsize=(8,6), title=' Magnetometer␣
↪(chest) by Activity')
a = ax.set_xticklabels(chest_magn_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(chest_magn_activity)))
plt.show()
```


Acceleration (chest) by Activity
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
chest_x
chest_y
chest_z
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking

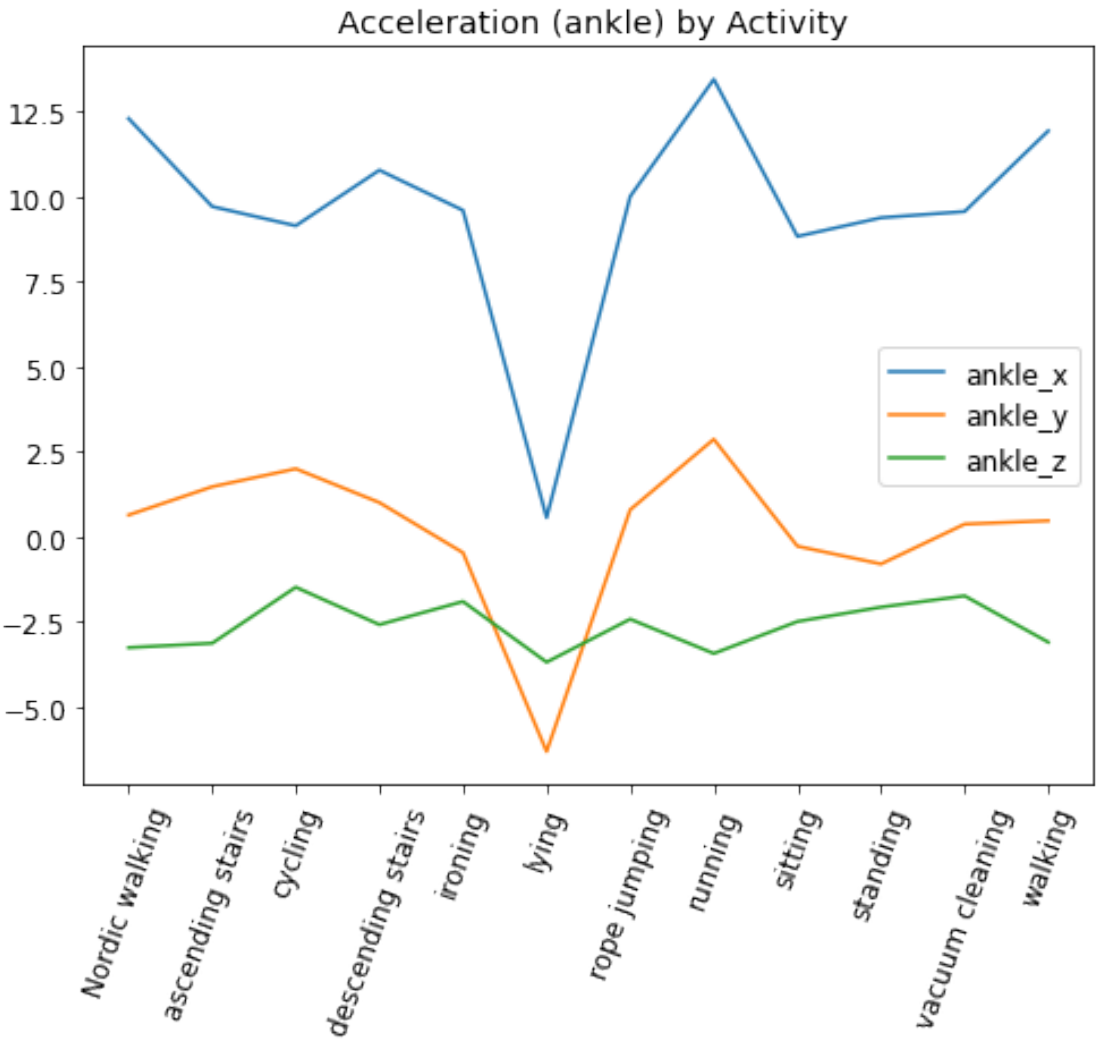

Gyro (chest) by Activity
chest_x
chest_y
chest_z
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking

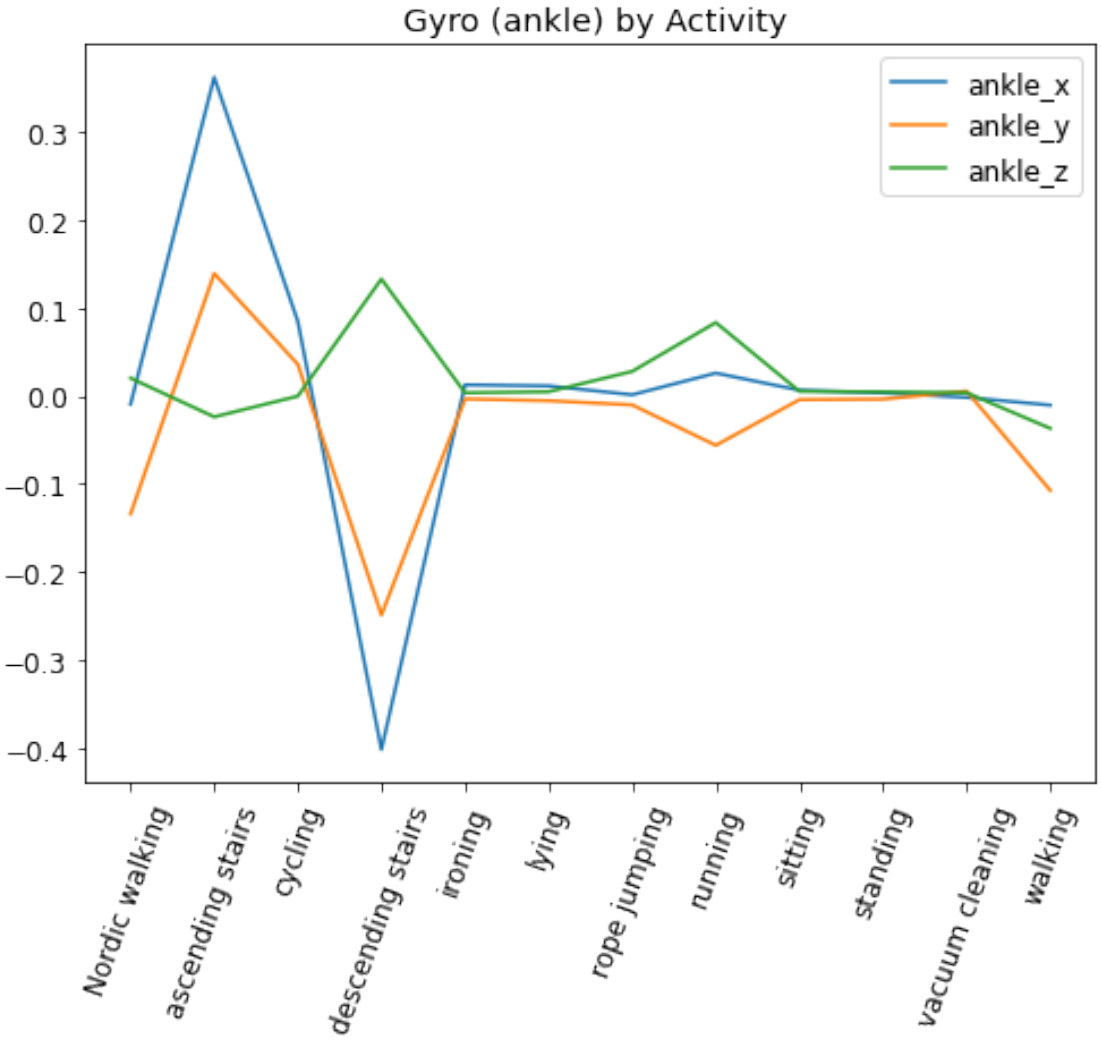


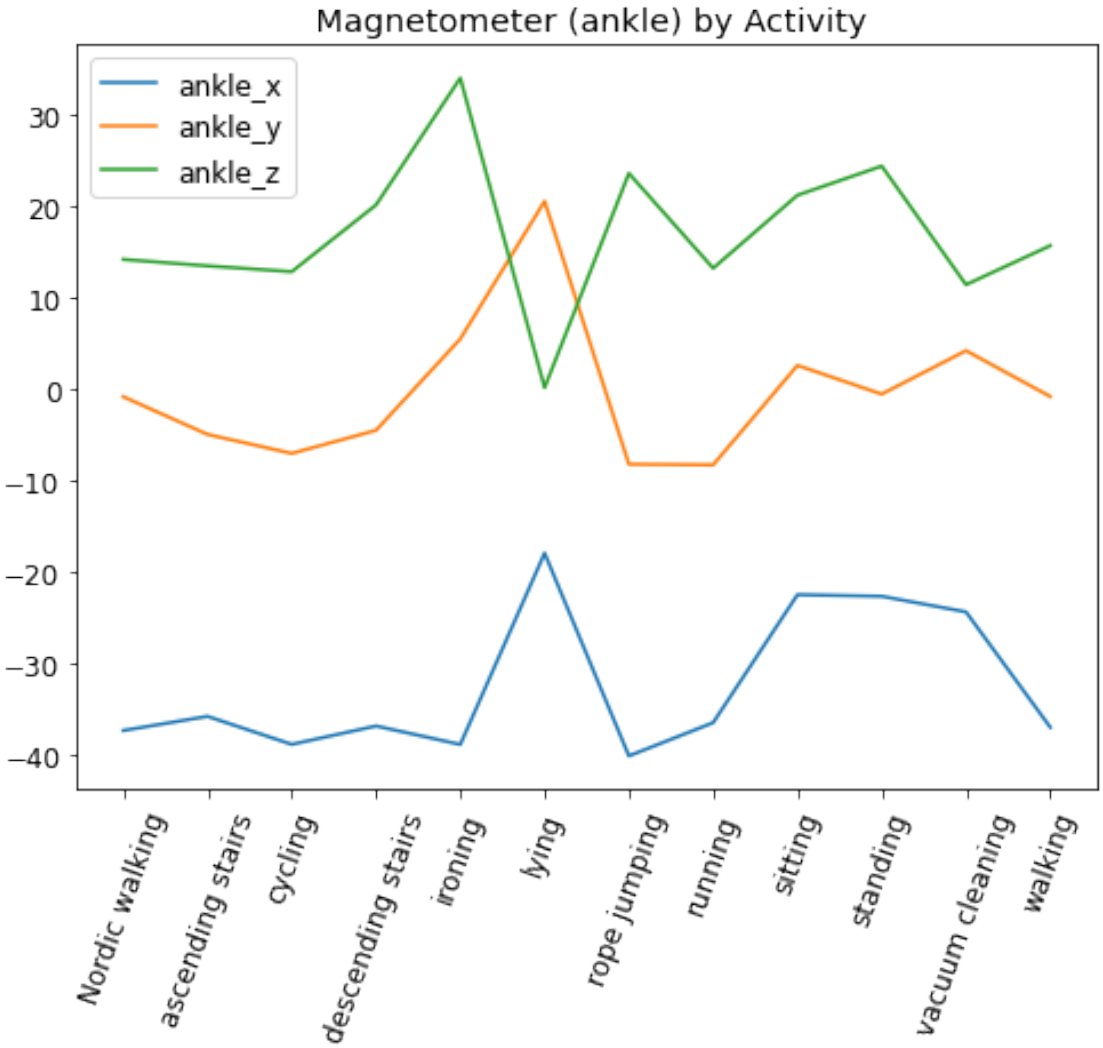
- **Ankle IMU sensor by activity**

[115]:
```
## plotting ankle acceleration sensor analysis based on activity

ankle_sensor_activity = pd.DataFrame()
ankle_sensor_activity['activity'] = samples['activityID']
ankle_sensor_activity['ankle_x'] = result_act['ankle_acceleration_16g (x)']
ankle_sensor_activity['ankle_y'] = result_act['ankle_acceleration_16g (y)']
ankle_sensor_activity['ankle_z'] = result_act['ankle_acceleration_16g (z)']
ax = ankle_sensor_activity.plot(kind='line', figsize=(8,6), title='␣
↪Acceleration (ankle) by Activity')
a = ax.set_xticklabels(ankle_sensor_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(ankle_sensor_activity)))

## plotting ankle gyroscope sensor analysis based on activity
```

```
ankle_gyro_activity = pd.DataFrame()
ankle_gyro_activity['activity'] = samples['activityID']
ankle_gyro_activity['ankle_x'] = result_act['ankle_gyroscope (x)']
ankle_gyro_activity['ankle_y'] = result_act['ankle_gyroscope (y)']
ankle_gyro_activity['ankle_z'] = result_act['ankle_gyroscope (z)']
ax = ankle_gyro_activity.plot(kind='line', figsize=(8,6), title=' Gyro (ankle)␣
 ↪by Activity')
a = ax.set_xticklabels(ankle_gyro_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(ankle_gyro_activity)))

## plotting ankle magnetometer sensor analysis based on activity
ankle_magn_activity = pd.DataFrame()
ankle_magn_activity['activity'] = samples['activityID']
ankle_magn_activity['ankle_x'] = result_act['ankle_magnetometer (x)']
ankle_magn_activity['ankle_y'] = result_act['ankle_magnetometer (y)']
ankle_magn_activity['ankle_z'] = result_act['ankle_magnetometer (z)']
ax = ankle_magn_activity.plot(kind='line', figsize=(8,6), title=' Magnetometer␣
 ↪(ankle) by Activity')
a = ax.set_xticklabels(ankle_magn_activity['activity'],rotation=70)
c = ax.set_xticks(np.arange(len(ankle_magn_activity)))
plt.show()
```


Acceleration (ankle) by Activity
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
ankle_x
ankle_y
ankle_z
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking


Gyro (ankle) by Activity
ankle_x
ankle_y
ankle_z
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
Nordic walking
ascending stairs
cycling
descending stairs
ironing
lying
rope jumping
running
sitting
standing
vacuum cleaning
walking



0.2.2 Outcomes of EDA:

From the various observations in this section, we arrive at a conclusion that the dataset is not balanced due to some individuals not performing specific activity. Also, the following insights can be derived from the provide dataset:

- Heart Rate values were missing in multiple rows, which required dropping to interpret various methods performed during EDA.
- Heart rate is the best feature which could be used to predict the activity being performed by the individual.
- Chest temperature IMU readings seems to be more accurate among the various tempertature IMU's being used.
- Accelerometer is another feature which can be used to predict the activity after hear rate. All the IMU's readings from accelerometer,gyroscope and magnetomer are proving to be significant for activity recognition.

- Model which consider all the above features would be most accurate in predicting physical activity being performed.

0.3 Part II : Hypothesis Testing

The mean calculated from a sample, $\bar{x}$ is an estimate of a population mean, . For the hypothesis test, samples of size n from a population X is taken. Since the value of n is large enough (n>30), the central limit theorem holds good and thus the sampling distribution is approximated to be normal. The p-value is a number describing how likely the data would have occurred by random chance. A p-value 0.05 is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than a 5% probability the null is correct. So we reject the null hypothesis, and choose the alternative hypothesis. In case the p-value > 0.05, it is considered to be statistically insignificant and indicates strong evidence for the null hypothesis. This means we retain the null hypothesis and reject the alternative hypothesis.

Multiple hypothesis is performed on the training data to prove various hypothesis. I have shortlisted the below parameters for the hypothesis testing :

- Hypothesis Testing based on Heart Rate
- Hypothesis Testing based on Temperaure

Hypothesis Testing based on Heart Rate

From above experiments, it is understood that the heart rates of various activities will be a deciding factor in activity recognition and we try to prove the same by performing two Hypothesis. For this, training dataset is categorized as **vigorous** , **light** and **moderate**.

" If heart rate is related to intensity of physical activity, then subjects performing moderate activities would have higher heart rates than light activities"

Tests will be conducted on mean heart rate of subjects performing moderate activities and the mean heart rate of subjects performing light activities.

Null Hypothesis: h0 : Heart rate of subjects performing moderate activities will be equal to the Heart rate of subjects performing light activities. (1 - 2 = 0)

Alternate Hypothesis: h1 : Heart rate of subjects performing moderate activities will higher than the heart rate of subjects performing light activities.(1 - 2 > 0)

[116]:
```
## Light activities statistics
light_act = my_testing_data.loc[(my_testing_data["activity_level"] == 'light')]
light_act_mean = light_act['heart_rate'].mean()
light_act_stdev = light_act['heart_rate'].std()
light_act_count = light_act['heart_rate'].count()

print("Mean : Heart rate of individuals performing light activities =  {} ".
 ↪format(light_act_mean))
print("Standard deviaion : Heart rate of individuals performing light␣
 ↪activities =  {} ".format(light_act_stdev))
```

```
print("Count : Heart rate of individuals performing light activities =  {} ".
 ↪format(light_act_count))
```

```
Mean : Heart rate of individuals performing light activities =
83.90076076620025
Standard deviaion : Heart rate of individuals performing light activities =
11.060756344862344
Count : Heart rate of individuals performing light activities =  14722
```

[117]:
```
## Moderate activities statistics
moderate_act=my_testing_data.loc[(my_testing_data["activity_level"] ==␣
 ↪'moderate')]
moderate_act_mean = moderate_act['heart_rate'].mean()
moderate_act_stdev = moderate_act['heart_rate'].std()
moderate_act_count = moderate_act['heart_rate'].count()

print("Mean : Heart rate of individuals performing moderate activities =  {} ".
 ↪format(moderate_act_mean))
print("Standard deviaion : Heart rate of individuals performing moderate␣
 ↪activities =  {} ".format(moderate_act_stdev))
print("Count : Heart rate of individuals performing moderate activities =  {} ".
 ↪format(moderate_act_count))
```

```
Mean : Heart rate of individuals performing moderate activities =
117.60879851652919
Standard deviaion : Heart rate of individuals performing moderate activities =
15.713016146226286
Count : Heart rate of individuals performing moderate activities =  15639
```

Difference between means : D ~ N (1 - 2 , 1^2 / n1 + 2^2 / n2)

If two (or more) random variables are normally distributed, then their sum and difference of those variables will also be normally distributed.

[118]:
```
## calculationg z-score for 1st hypothesis
mean_diff=moderate_act_mean-light_act_mean
sig_moderate = (moderate_act_stdev**2)/moderate_act_count
sig_light = (light_act_stdev**2)/light_act_count

print ("Mean difference = {}".format(mean_diff))
## (  12  / n1 +  22  / n2)


den=np.sqrt(sig_moderate+sig_light)

Z1=mean_diff/den
print("Value of Z = {}".format(Z1))
```

```
Mean difference = 33.70803775032894
Value of Z = 217.1441943444517
```

Both samples are of a sufficient size to assume a normal distribution, hence we use it to calculate the p-value.

P(D 33.70803775032894) = P(Z 33.70803775032894 / √(1^2 / n1 + 2^2 / n2))

P(D 33.70803775032894) = P(Z 217.1441943444517)

[119]:
```
## conductiong z-test to p-value
pValue = stats.norm.sf(Z1)

if pValue > 0.05:
    print("The p-value is ", pValue, " and We fail to reject h0")
else:
    print("The p-value is ", pValue, " and We are rejecting h0.")
```

```
The p-value is  0.0  and We are rejecting h0.
```

Z-Test is conducted above to obtain the probability value (p-value) with a condifence interval of 95%.

The p-value remains '0' from the above z-test. So, there is a strong evidence to reject the null hypothesis. Thus,we reject the null hypothesis statement and conclude with good confidence that "the heart rate of moderate activities will be higher than light activities".

Now, the second hypothesis is conducted to prove that heart rate of subjects performing vigorous activity is higher than subjects performing moderate activities.

" If heart rate is related to intensity of physical activity, then subjects performing vigorous activities would have higher heart rates than moderate activities"

Tests will be conducted on mean heart rate of subjects performing moderate activities and the mean heart rate of subjects performing vigorous activities.

Null Hypothesis: h0 : Heart rate of subjects performing vigorous activities will be equal to the Heart rate of subjects performing moderate activities. (1 - 2 = 0)

Alternate Hypothesis: h1 : Heart rate of subjects performing vigorous activities will higher than the heart rate of subjects performing moderate activities.(1 - 2 > 0)

[120]:
```
## Vigorous activities statistics
vigorous_act= my_testing_data.loc[(my_testing_data["activity_level"] ==␣
 ↪'vigorous')]
vigorous_act_mean = vigorous_act['heart_rate'].mean()
vigorous_act_stdev = vigorous_act['heart_rate'].std()
vigorous_act_count = vigorous_act['heart_rate'].count()

print("Mean : Heart rate of individuals performing vigorous activities =  {} ".
 ↪format(vigorous_act_mean))
print("Standard deviaion : Heart rate of individuals performing vigorous␣
 ↪activities =  {} ".format(vigorous_act_stdev))
```

```
print("Count : Heart rate of individuals performing vigorous activities =  {} ".
 ↪format(vigorous_act_count))
```

```
Mean : Heart rate of individuals performing vigorous activities =
145.08461700780757
Standard deviaion : Heart rate of individuals performing vigorous activities =
26.03455454804163
Count : Heart rate of individuals performing vigorous activities =  4739
```

Difference between means : D ~ N (1 - 2 , 1^2 / n1 + 2^2 / n2)

[121]:
```
## calculating z-score for 2nd hypothesis
mean_diff= vigorous_act_mean-moderate_act_mean
sig_vigorous = (vigorous_act_stdev**2)/vigorous_act_count

print ("Mean difference = {}".format(mean_diff))
## (  12  / n1 +  22  / n2)

den=np.sqrt(sig_moderate+sig_vigorous)
Z2=mean_diff/den
print("Value of Z = {}".format(Z2))
```

```
Mean difference = 27.475818491278375
Value of Z = 68.94578499677506
```

[122]:
```
## conductiong z-test to p-value
pValue = stats.norm.sf(Z2)

if pValue > 0.05:
    print("The p-value is ", pValue, " and We fail to reject h0")
else:
    print("The p-value is ", pValue, " and We are rejecting h0.")
```

```
The p-value is  0.0  and We are rejecting h0.
```

The p-value remains '0' from the above z-test and there is a strong evidence to reject the null hypothesis. Thus,we go with alternate hypothesis statement as the p-value is less than significance value and can be concluded with good confidence that "the heart rate of vigorous activities will be higher than moderate activities".

From above 2 hypothesis, we conclude that individuals performing vigorous activity will have the highest heart rate.

Hypothesis Testing based on Heart rate & Temperaure

"" If heart rate is related to Chest temperature, then heart rate could increase with chest temperature".

Here, tests are conducted on heart rates and chest temperature for all the subjects.

Null Hypothesis: h0 : there is no relation between heart rate of subjects and their chest temperature.

Alternate Hypothesis: h1 : There is relation between heart rate of subjects and their chest temperature

We assume the null hypothesis is true, and to compute the probability value by using Pearson's correlation.

[123]:
```
#Calculating correlation using pearsons
df_chesttemp=my_testing_data['chest_temperature']
df_heart=my_testing_data['heart_rate']

pearsoncc=stats.pearsonr(np.array(df_chesttemp),np.array(df_heart))
print("Pearson's correlation coefficient is {}".format(pearsoncc))
```

```
Pearson's correlation coefficient is (-0.11655850190117642,
2.0329881307110941e-106)
```

Obtained Pearson's correlation coefficient **(r) = -0.116** and p-value(probablility value) = **2.0329881307110941e-106**. Calculated p-value is very small and the correlation coefficienct is negative.Hence, this test can be considered significant and we reject the null hypothesis H0, meaning that there is a negative correlation between Chest temperature and Heart rate of subjects. So,as the heart rate increase, the chest temperature will be decreasing. Various factors like sweating during the excessive workout could be lowering the body temperature.

0.4 Part III : Modelling

Before modelling, PCA and clustering is performed to provide a better understanding of any underlying relations between given feature set.

Algorithms which use set of unlabelled dataset comes under unsupervised learing and, the underlying relationships are unknown since the data is unlabelled. Whereas in supervised learning, training data uses the known labels to produce an inferred function, which can be used for predicting new labes of data. Selection of best feature (or features) is important for optimum performance of the model being developed. Pertaining to provided dataset, all the IMU parameters should be considered for the models performance. The same can be seen in various EDA performed in section 1. After splitting the dataset, first set is used for model training and performance of developed model is tested by using the second set (by observing how well it predicts testing set parameters).Various parameters like Precision, Accuracy, Recall and F1 Score is used to compare performance of developed model. An ideal model should have high Accuracy, meaning that the model is predicting inputs correctly (i.e. the number of false positives and negatives should be lowest).

0.4.1 Principal Component Analysis

For PCA, inputs from the training dataset, which is 80% of the total sample data is used. The major challenge in the provided dataset is the number of features. It is challenging to interpret how each feature is contributing to the model as each feature can be represented as a dimension. It is very difficult to interpret hugher dimensions, hence we need to reduce the dimensionality. This

is the reason to perform the PCA i.e. for dimensionality reduction. PCA will extract the principal components as per the variance level, so that it becomes easy for the clustering algoritm in later stage.

[ ]:
```
train_model_df=my_training_data
```

[ ]:
```
test_model_df=my_testing_data
```

Some irrelevant columns needs to be dropped before doing the PCA. Here, "timestamp" and "subject_id" is not relevant and is dropped for the model creation.

[ ]:
```
train_model_df = train_model_df.drop(["timestamp", "subject_id"],1)
train_model_df
```

[ ]:

	activityID	heart_rate	…	ankle_magnetometer (z)	activity_level
119017	vacuum cleaning	116.0	…	-1.48390	moderate
20674	running	171.0	…	36.29810	vigorous
1052	lying	86.0	…	25.27570	light
111406	sitting	80.0	…	10.84890	light
98999	walking	114.0	…	21.95510	moderate
…	…	…	…	…	…
46641	lying	76.0	…	-8.88257	light
7423	ironing	94.0	…	58.49410	light
38570	walking	123.0	…	12.73780	moderate
48932	sitting	77.0	…	13.86440	light
5491	standing	105.0	…	54.91700	light

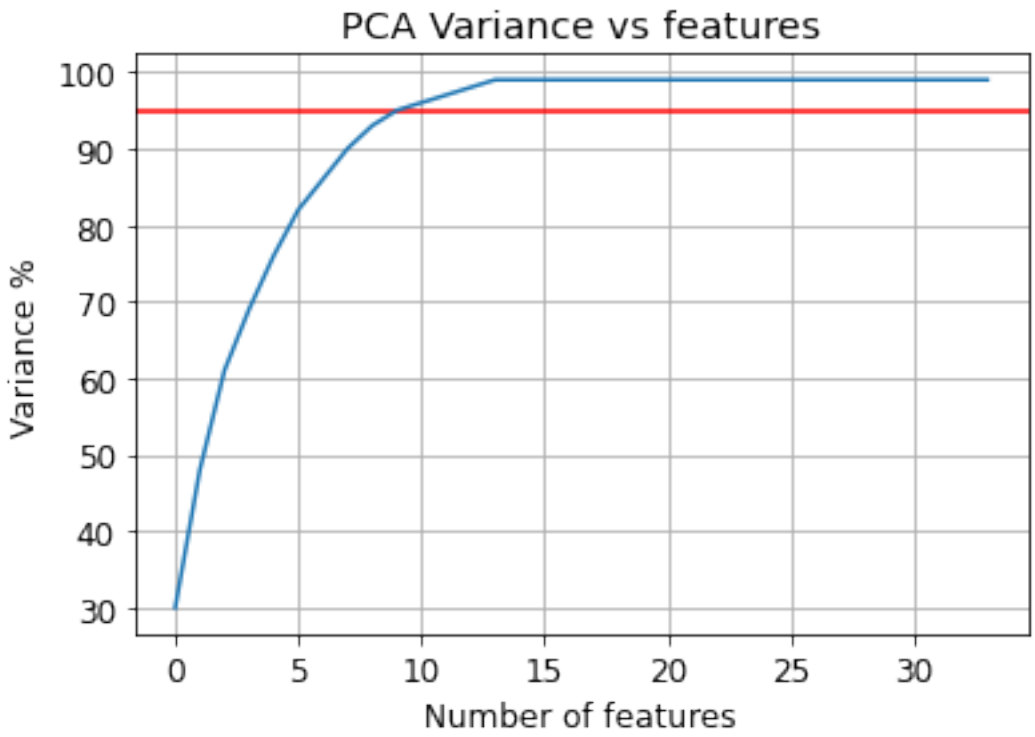
[140398 rows x 36 columns]

[ ]:
```
## Train dataset
X_train = train_model_df.drop(['activityID','activity_level'], axis=1)
y_train = train_model_df['activityID'].values

# Test Dataset
X_test = test_model_df.drop(['activityID','activity_level'], axis=1)
y_test = test_model_df['activityID'].values
```

Our goal is to predict the activityID and only the features which are to be considered to predict the activityID must be present in the training dataset (X_train). This is achieved by dropping the activityID data before passing it over to the PCA algorithm. If the model already knows what the activities are, then it would be resulting in class inbalance or bias.The tags to be predicted are stored and passed to a variable(y_train).

[ ]:
```
## PCA on provided dataset
pca_pamap2 = PCA()
pca_pamap2.fit(X_train)

feature_var=np.cumsum(np.round(pca_pamap2.explained_variance_ratio_,2)*100)
```

```
plt.title("PCA Variance vs features")
plt.ylabel("Variance %")
plt.xlabel("Number of features")
l = plt.axhline(95, color="red")

plt.plot(feature_var)
plt.grid()
```



Usually 90% of variance explains most relevant features out of the dataset and by plotting variance ratio aginst the number of features,we can find the important components. From the graph above, around 8 components fall around to required level of the variance for analysis.

[ ]:
```
## variance expalied for each principal component
My_pca = PCA(n_components=8)
X_train=My_pca.fit_transform(X_train)
print('Explained variation per principal component: {}'.format(My_pca.
 ↪explained_variance_ratio_))
```

```
Explained variation per principal component: [0.30016784 0.18108406 0.12505156
0.07997004 0.07190191 0.06311139
 0.04441721 0.03922649]
```

Looking at the above readings, the significance of each principal componenets are understood. It is evident that most of the features are captured in the 1st principal component and the significance

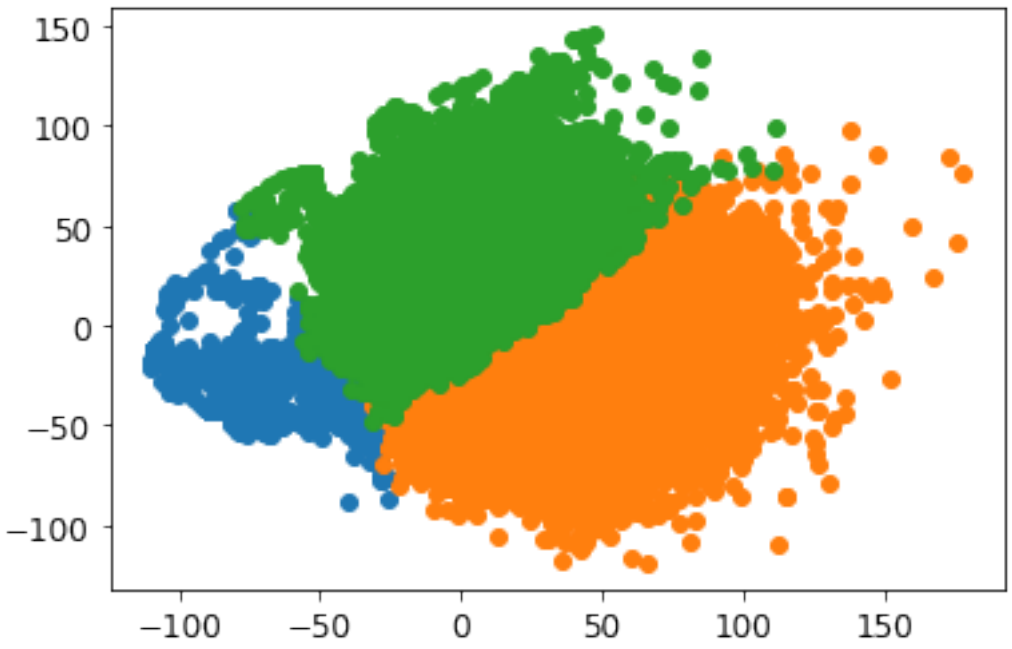
of rest components are following.

Clustering

Cluster analysis is an unsupervised machine learning task. It automatically discovers natural grouping in data and clusters them based on correlation. Unlike supervised learning , clustering algorithms only interpret the input data and find natural groups or clusters in feature space. Inorder to test this algorithm, labelled data in our dataset (activityID) is dropped.

[ ]:
```
from sklearn import cluster
from collections import defaultdict

K=3 # specify number of clusters

cluster_K3=cluster.KMeans(init='random',n_clusters=K)
cluster_K3.fit(X_train)

cluster_K3.predict(X_train)

#In order to plot the clusters we need the co-ordinates of the points to be put␣
 ↪in appropriate lists. One way is:

def divide(data,labels):
    xclusters=defaultdict(list)
    yclusters=defaultdict(list)
    for datapoint,label in zip(data,labels):
        x=datapoint[0]
        y=datapoint[1]
        xclusters[label].append(x)
        yclusters[label].append(y)
    return xclusters,yclusters

clusters=divide(X_train,cluster_K3.predict(X_train)) # apply the function that␣
 ↪makes the data point lists to the predicted clusters

#plot a 2d scatter plots of the generated clusters
plt.scatter(clusters[0][0],clusters[1][0])
plt.scatter(clusters[0][1],clusters[1][1])
plt.scatter(clusters[0][2],clusters[1][2])
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fc8c2864a90>
```



Dataset transformed after the PCA is used in the above clustering. K-clustering algorithm is used above with number of clusters as 3. Above cluster seems to hold good for the model creation. There appears to be overlapping of the various clusters, which may hinder the performace of the model.

[ ]:
```
score= silhouette_score(X_train, cluster_K3.labels_)

print('Silhouette Score = ', score)
```

```
Silhouette Score =  0.2098942583912616
```

Silhouette Score for the model is about 0.21, which is fair. To improve the score, the number of features being passed to the model needs to be revisited. Also, the training dataset can be further cleansed, so that any outliers would be minimal. Further, models are created by using various library functions.

Clustering with limited features

[136]:
```
## relevant features are extracted
cluster_df=my_training_data[["chest_temperature","heart_rate","hand_acceleration_16g␣
↪(x)","hand_acceleration_16g (y)","hand_acceleration_16g (z)","hand_gyroscope␣
↪(x)","hand_gyroscope (y)","hand_gyroscope (z)"]]
cluster_df
```

[136]:

	chest_temperature	heart_rate	…	hand_gyroscope (y)	hand_gyroscope (z)
119017	36.7500	116.0	…	0.636643	0.785932

```
20674              33.8125       171.0  …            2.901680
4.234370
1052               33.0625        86.0  …            0.125737
0.182229
111406             35.8750        80.0  …            0.000435
-0.086043
98999              38.2500       114.0  …            0.453831
0.885213
…                    …          …  …                 …
…
46641              33.3125        76.0  …            0.018704
0.005499
7423               35.1250        94.0  …           -1.010030
2.254600
38570              36.1875       123.0  …            0.767846
-4.391340
48932              34.6250        77.0  …           -0.007783
-0.038713
5491               34.5000       105.0  …           -0.067666
-0.059835

[140398 rows x 8 columns]
```

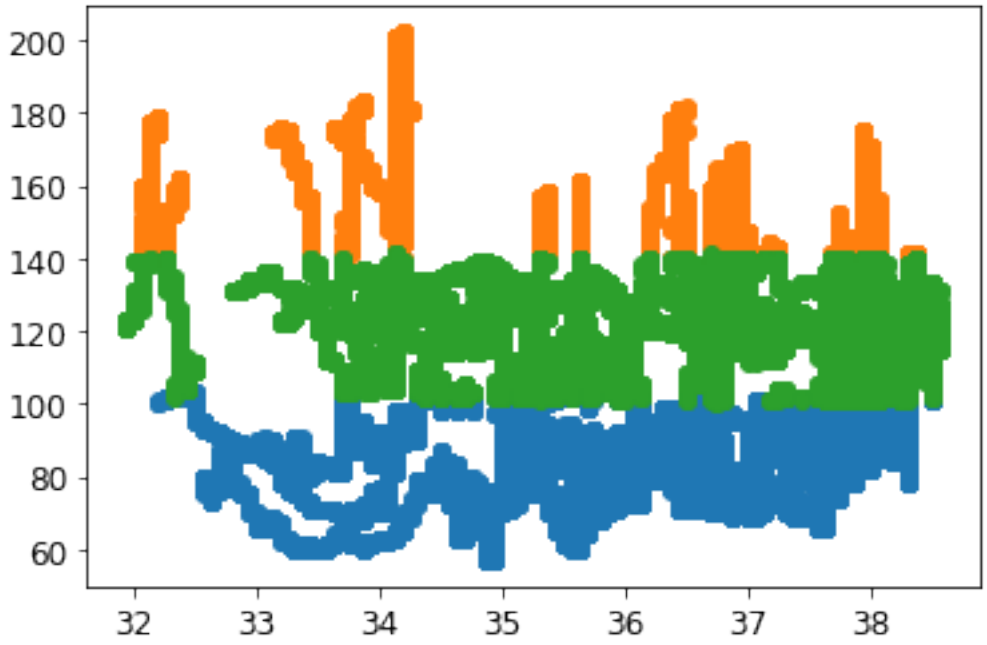
Based on the above EDA, we have already proven the relationship between the **chest temperature,heart rate, gyroscope and accelerometer**. Hence, we are doing another cluster considering the above features alone to find the Silhouette score.

[137]:
```
## converting the dataset to numpy array
short_df = cluster_df.astype(np.float32)
X=short_df.to_numpy()
```

[135]:
```
from sklearn import cluster
from collections import defaultdict

K=3 # specify number of clusters

cl_K3=cluster.KMeans(init='random',n_clusters=K)
cl_K3.fit(X)

cl_K3.predict(X)

#In order to plot the clusters we need the co-ordinates of the points to be put␣
 ↪in appropriate lists. One way is:

def divide(data,labels):
    xclusters=defaultdict(list)
    yclusters=defaultdict(list)
```

```
    for datapoint,label in zip(data,labels):
        x=datapoint[0]
        y=datapoint[1]
        xclusters[label].append(x)
        yclusters[label].append(y)
    return xclusters,yclusters


clusters=divide(X,cl_K3.predict(X)) # apply the function that makes the data␣
↪point lists to the predicted clusters

#plot a 2d scatter plots of the generated clusters - remember the data is 3d!␣
↪(plot 3d if you want...)
plt.scatter(clusters[0][0],clusters[1][0])
plt.scatter(clusters[0][1],clusters[1][1])
plt.scatter(clusters[0][2],clusters[1][2])
```

[135]: <matplotlib.collections.PathCollection at 0x7fc8a04376d0>



[138]:
```
## Calculating Silhouette score
from sklearn.metrics.cluster import silhouette_score
SC_3=silhouette_score(X,cl_K3.predict(X))
print('Silhouette Score = ', SC_3)
```

```
Silhouette Score =  0.480303
```

New Silhouette score is 0.48, which is much better than the earlier score of 0.2 (considering all the features). Hence, the earlier hypothesis and the above clustering paves way for the best model. This cluster can be used with the labelled data for predicting the activity performed by the subject.

Modelling

In this section,We are going to focus on three modelling algorithms,Logistic-regression and RandomForest. These are best suited for most supervised learning modelling.

[ ]:
```
## confusion matrix to evaluate the performance of model
def my_confusion_matrix (predict_activity, real_activity, title):

    #Accuracy, precision, recall and f-score are calculated using sklearn␣
↪library
    precision = precision_score(real_activity, predict_activity,␣
↪average='macro')
    recall = recall_score(real_activity, predict_activity, average='macro')
    accuracy = accuracy_score(real_activity, predict_activity)
    fscore = f1_score(real_activity, predict_activity, average='macro')

    #Display results
    print((title))
    print(('Accuracy: ' + str(accuracy)))
    print(('Precision: ' + str(precision)))
    print(('Recall: ' + str(recall)))
    print(('F-score: ' + str(fscore)))
```

Above Function is used to display confusion matrix for predicted and real activity. sklearn library is used to calculate accuracy, precision, recall and F-scores. Macro type calculates values separated by class and not using weights for the aggregation,which leads to bigger penalisation as model does not perform well with the minority classes.This is exactly what we want as there is imbalance in the dataset.

[ ]:
```
features_df=my_training_data
features_df=features_df.drop(['activityID','activity_level'],axis=1)
```

[ ]:
```
## extracting the features list from dataframe
features=np.array(features_df.columns)
features
```

[ ]:
```
array(['timestamp', 'heart_rate', 'hand_temperature',
       'hand_acceleration_16g (x)', 'hand_acceleration_16g (y)',
       'hand_acceleration_16g (z)', 'hand_gyroscope (x)',
       'hand_gyroscope (y)', 'hand_gyroscope (z)',
       'hand_magnetometer (x)', 'hand_magnetometer (y)',
       'hand_magnetometer (z)', 'chest_temperature',
       'chest_acceleration_16g (x)', 'chest_acceleration_16g (y)',
       'chest_acceleration_16g (z)', 'chest_acceleration_6g (x)',
       'chest_acceleration_6g (y)', 'chest_acceleration_6g (z)',
```

```
       'chest_gyroscope (x)', 'chest_gyroscope (y)',
       'chest_gyroscope (z)', 'chest_magnetometer (x)',
       'chest_magnetometer (y)', 'chest_magnetometer (z)',
       'ankle_temperature', 'ankle_acceleration_16g (x)',
       'ankle_acceleration_16g (y)', 'ankle_acceleration_16g (z)',
       'ankle_gyroscope (x)', 'ankle_gyroscope (y)',
       'ankle_gyroscope (z)', 'ankle_magnetometer (x)',
       'ankle_magnetometer (y)', 'ankle_magnetometer (z)', 'subject_id'],
      dtype=object)
```

[ ]:
```
#features_used for training the model
train_data = np.array(my_training_data.loc[:, features])
train_activity = np.array(my_training_data.loc[:, 'activityID'])

#features used while testing the model
test_data = np.array(my_testing_data.loc[:, features])
real_activity = np.array(my_testing_data.loc[:, 'activityID'])
```

[ ]:
```
## fitting the model using Logistic Regression algorithm
log_reg = LogisticRegression(max_iter=300)
log_reg.fit(X=train_data, y=train_activity )

## predicting the model using Logistic Regression algorithm
y_pred_lr = log_reg.predict(test_data)
my_confusion_matrix(y_pred_lr, real_activity,"Confusion Matrix using all␣
 ↪features")
```

```
Confusion Matrix using all features
Accuracy: 0.8451566951566951
Precision: 0.8091938732754898
Recall: 0.7895192669526202
F-score: 0.7964216700031211
```

Interpreting the above results, the performance of our model using LogisticRegression is around 85%. During EDA, we have seen the undelying outliers in the dataset. These outliers acts like nosie, hindering the accuracy of the model. Also, during the PCA, the number of principal components was around 8 and if we were to only give these components to our model, it would have the best accuracy (more than 95%). Nonetheless, the model looks to be okay as we are able to get around 85% accuracy.(The goldstandard will be 95% accuracy).

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
## fitting the model using RadomForestClassifier
rf_clf = RandomForestClassifier(criterion='entropy')
rf_clf.fit(train_data, train_activity)

## predicting the model using RFC
y_predict = rf_clf.predict(test_data)
```

```
my_confusion_matrix(y_predict, real_activity, "Confusion Matrix using all␣
↪features ")
```

```
Confusion Matrix using all features
Accuracy: 0.9993447293447294
Precision: 0.9992295842125437
Recall: 0.9992365142207612
F-score: 0.9992329507609575
```

Random Forest algorithm is widely used for classification as well as regression, which makes it versatile. Random Forest is a forest of trees, specifically decision trees which are randomly populating the forest. This algorithm combines various decision trees together (more trees in the forest, the better the accuracy of its predictions).Random Forest gives good performance especially for high dimensionality datasets, as is the case for the provided dataset.

0.4.2 Conclusion

Earlier in the introduction, the need for activity monitoring devices was discussed and in this report, we have analysed physical activity and their corelation with various IMU's. During EDA, dataframe is loaded with subject's data and new columns are created for **subject_id** and activity level (MET).Then, 'activityID 0' is removed and missing sensory datas were dropped. Finally,The ***6g accelerometer*** data were removed as it was not precisely calibrated.

Further, we gained insights on how one parameter affects the rest, various hypothesis was tested and proven to study the underlying relations between pair of attributes and finally, a model was created with best accuracy, which correctly predicts the physical activity being performed.

Differenct features from IMU's was used to create 2 unsupervised and supervised models , which predicts the activity being performed. Performance of various modelling algorithms were compared and the accuracy of prediction was almost 99% for random forest model(supervised model) and Silhouette score is 0.48 for the unsupervised model. These models are the core of any hardware/software which could be developed to predict the physical activity being performed by an individual. We also discussed about few challenges occured while doing various analysis to arrive at the final models.

[ ]: