Table of Contents

Introduction

Goal of the Report

The goal of this report is to develop hardware and/or software which can determine the amount (using start/end times and heart rates) and type of physical activity carried out by an individual, and to also determine the actionable insights that are derivable from the dataset.

The dataset used for this assignment is Physical Activity Monitoring dataset "PAMAP2", an activity monitoring dataset that contains 18 different physical activities that were performed by 9 different subjects — eight men and one woman wearing 3 inertial measurement units and a heart rate monitor. The data is stored in individual text files per subject. Each row in each file represents one reading and contains 54 attributes (including timestamp, activity ID, heart rate and IMU sensory data).

Specific Requirements for Report

1. Carry out thorough exploratory data analysis and appropriately handle missing or dirty data;
2. Develop and test at least one hypothesis for a relationship between a single pair of attributes;
3. Develop and test at least one model which uses multiple attributes to make predictions.

The first step i will be taking is to import the neccessary libraries and functions needed for all of my code to run properly.

Preliminary Tasks

In [1]:
```
#importing necessary Libraries
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.decomposition import PCA
import numpy as np
from sklearn.model_selection import StratifiedKFold
import seaborn as sn
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler, RobustScaler
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from sklearn.model_selection import train_test_split
from IPython.display import display
sn.set()
```

In the cell below, i will be loading the dataset.

In [2]:
```
# Loading the dataset
list_of_files = ['PAMAP2_Dataset/Protocol/subject101.dat',
                 'PAMAP2_Dataset/Protocol/subject102.dat',
                 'PAMAP2_Dataset/Protocol/subject103.dat',
                 'PAMAP2_Dataset/Protocol/subject104.dat',
                 'PAMAP2_Dataset/Protocol/subject105.dat',
                 'PAMAP2_Dataset/Protocol/subject106.dat',
                 'PAMAP2_Dataset/Protocol/subject107.dat',
                 'PAMAP2_Dataset/Protocol/subject108.dat',
                 'PAMAP2_Dataset/Protocol/subject109.dat' ]

subjectID = [1,2,3,4,5,6,7,8,9]

activity_id = {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' }

colNames = ["timestamp", "activity_id","heartrate"]

IMUhand = ['handTemperature',
           'handAcc16_1', 'handAcc16_2', 'handAcc16_3',
           'handAcc6_1', 'handAcc6_2', 'handAcc6_3',
           'handGyro1', 'handGyro2', 'handGyro3',
           'handMagne1', 'handMagne2', 'handMagne3',
           'handOrientation1', 'handOrientation2', 'handOrientation

IMUchest = ['chestTemperature',
           'chestAcc16_1', 'chestAcc16_2', 'chestAcc16_3',
           'chestAcc6_1', 'chestAcc6_2', 'chestAcc6_3',
           'chestGyro1', 'chestGyro2', 'chestGyro3',
           'chestMagne1', 'chestMagne2', 'chestMagne3',
           'chestOrientation1', 'chestOrientation2', 'chestOrientat

IMUankle = ['ankleTemperature',
           'ankleAcc16_1', 'ankleAcc16_2', 'ankleAcc16_3',
           'ankleAcc6_1', 'ankleAcc6_2', 'ankleAcc6_3',
           'ankleGyro1', 'ankleGyro2', 'ankleGyro3',
           'ankleMagne1', 'ankleMagne2', 'ankleMagne3',
           'ankleOrientation1', 'ankleOrientation2', 'ankleOrientat

columns = colNames + IMUhand + IMUchest + IMUankle  #all columns in

len(columns)
```

Out[2]: 54

In [3]:
```
columns
```

Out[3]:
```
['timestamp',
 'activity_id',
 'heartrate',
 'handTemperature',
 'handAcc16_1',
 'handAcc16_2',
 'handAcc16_3',
 'handAcc6_1',
 'handAcc6_2',
 'handAcc6_3',
 'handGyro1',
 'handGyro2',
 'handGyro3',
 'handMagne1',
 'handMagne2',
 'handMagne3',
 'handOrientation1',
 'handOrientation2',
 'handOrientation3',
 'handOrientation4',
```

```
'chestTemperature',
'chestAcc16_1',
'chestAcc16_2',
'chestAcc16_3',
'chestAcc6_1',
'chestAcc6_2',
'chestAcc6_3',
'chestGyro1',
'chestGyro2',
'chestGyro3',
'chestMagne1',
'chestMagne2',
'chestMagne3',
'chestOrientation1',
'chestOrientation2',
'chestOrientation3',
'chestOrientation4',
'ankleTemperature',
'ankleAcc16_1',
'ankleAcc16_2',
'ankleAcc16_3',
'ankleAcc6_1',
'ankleAcc6_2',
'ankleAcc6_3',
'ankleGyro1',
'ankleGyro2',
'ankleGyro3',
'ankleMagne1',
'ankleMagne2',
'ankleMagne3',
'ankleOrientation1',
'ankleOrientation2',
'ankleOrientation3',
'ankleOrientation4']
```

In [4]:
```
#Loading all the seperate files into a single dataframe
df_raw = pd.DataFrame()
for i in list_of_files :
  int_data = pd.read_table(i , names = columns, sep = '\s+')
  int_data['SubjectID'] = int(i[-5])
  df_raw = df_raw.append(int_data, ignore_index = True)
```

```
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
/var/folders/43/4rz2qdq10nn_s7lrzvhpc8gc0000gp/T/ipykernel_29280/2
837029192.py:6: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas
.concat instead.
  df_raw = df_raw.append(int_data, ignore_index = True)
```

In [5]:
```
pd.set_option('display.max_columns',None)
df_raw
```

Out[5]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	h
0	8.38	0	104.0	30.0000	2.37223	8.60074	
1	8.39	0	NaN	30.0000	2.18837	8.56560	
2	8.40	0	NaN	30.0000	2.37357	8.60107	
3	8.41	0	NaN	30.0000	2.07473	8.52853	
4	8.42	0	NaN	30.0000	2.22936	8.83122	
...	...	...	...	...	...	...	
2872528	100.19	0	NaN	25.1875	-4.71493	10.22250	
2872529	100.20	0	NaN	25.1875	-4.95932	10.37130	
2872530	100.21	0	NaN	25.1875	-4.93997	9.83615	
2872531	100.22	0	NaN	25.1875	-4.64941	9.11129	
2872532	100.23	0	161.0	25.1875	-4.09726	8.15642	

2872533 rows × 55 columns

As observed from the table above, the dataframe requires cleaning. We move on to cleaning the data in next section.

Data Cleaning

For the data cleaning, I will be dropping data from the accelerometer under the scale of 6, because due to high impacts caused by certain movements saturation could occur with a scale 6 acceleration. Also I am dropping the orientation data as it is not useful to predict the activities. Also, from observing the "PerformedActivitiesSummary" file it is noticed that some data is missing which has to be accounted for and there are also some activities with NaN values for various subjects and i will be using "interpolation" to fill the data.

In [6]:
```
drop_columns = ['handAcc6_1', 'handAcc6_2', 'handAcc6_3', 'handOrie
```

In [7]:
```
df_raw1 = df_raw.drop(drop_columns, axis = 1)
```

In [8]:
```
#check for null values
df_raw1.isnull().sum()
```

Out[8]:
```
timestamp                0
activity_id              0
heartrate          2610265
handTemperature      13141
handAcc16_1          13141
handAcc16_2          13141
handAcc16_3          13141
handGyro1            13141
handGyro2            13141
handGyro3            13141
handMagne1           13141
handMagne2           13141
handMagne3           13141
chestTemperature      3563
chestAcc16_1          3563
chestAcc16_2          3563
chestAcc16_3          3563
chestGyro1            3563
chestGyro2            3563
chestGyro3            3563
chestMagne1           3563
chestMagne2           3563
chestMagne3           3563
ankleTemperature     11749
ankleAcc16_1         11749
ankleAcc16_2         11749
ankleAcc16_3         11749
ankleGyro1           11749
ankleGyro2           11749
ankleGyro3           11749
ankleMagne1          11749
ankleMagne2          11749
ankleMagne3          11749
SubjectID                0
dtype: int64
```

In [9]:
```
df_raw.head()
```

Out[9]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	handAc
0	8.38	0	104.0	30.0	2.37223	8.60074	3.5
1	8.39	0	NaN	30.0	2.18837	8.56560	3.6
2	8.40	0	NaN	30.0	2.37357	8.60107	3.5
3	8.41	0	NaN	30.0	2.07473	8.52853	3.6
4	8.42	0	NaN	30.0	2.22936	8.83122	3.7

In [10]:
```
df_raw1 = df_raw1.dropna(subset = ['heartrate'])
df_raw1
```

Out[10]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	h
0	8.38	0	104.0	30.0000	2.37223	8.60074	
10	8.48	0	104.0	30.0000	2.29745	8.90450	
21	8.59	0	104.0	30.0000	2.40867	9.16819	
32	8.70	0	104.0	30.0000	2.18114	8.86676	
43	8.81	0	104.0	30.0000	2.40681	8.71326	
...	...	...	...	...	...	...	
2872489	99.80	0	161.0	25.1875	-2.36226	10.75660	
2872500	99.91	0	161.0	25.1875	-5.04845	11.03180	
2872511	100.02	0	161.0	25.1875	-6.37083	11.64810	
2872522	100.13	0	161.0	25.1875	-4.98611	8.70795	
2872532	100.23	0	161.0	25.1875	-4.09726	8.15642	

262268 rows × 34 columns

In [11]:
```
df_raw1.isnull().sum()
```

Out[11]:
```
timestamp              0
activity_id            0
heartrate              0
handTemperature     1195
handAcc16_1         1195
handAcc16_2         1195
handAcc16_3         1195
handGyro1           1195
handGyro2           1195
handGyro3           1195
handMagne1          1195
handMagne2          1195
handMagne3          1195
chestTemperature     295
chestAcc16_1         295
chestAcc16_2         295
chestAcc16_3         295
chestGyro1           295
chestGyro2           295
chestGyro3           295
chestMagne1          295
chestMagne2          295
chestMagne3          295
ankleTemperature    1016
ankleAcc16_1        1016
ankleAcc16_2        1016
ankleAcc16_3        1016
ankleGyro1          1016
ankleGyro2          1016
ankleGyro3          1016
ankleMagne1         1016
ankleMagne2         1016
ankleMagne3         1016
SubjectID              0
dtype: int64
```

it is observed that there are still some missing values present in other columns after removing the rows with missing values. To deal with this missing values i will be introducing "data interpolation", which is simply constructing a new data point out of a set of known data points.

In [12]:
```
drop_index = []

#Getting indexes of activity 0
drop_index += list(df_raw1.index[df_raw1['activity_id']==0])

#Keep only activities as documented on file "PerformedActivitiesSum
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==1) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==2) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==3) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==4) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==5) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==6) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==7) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==8) & (df_ra
drop_index += list(df_raw1.index[(df_raw1['SubjectID']==9) & (df_ra

df_raw1 = df_raw1.drop(drop_index)
```

In [13]:
```
#interpolating the data
df_raw1 =df_raw1.interpolate(limit_direction = 'both')
```

In [14]:
```
df_raw1.head()
```

Out[14]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	hand
2932	37.70	1	100.0	30.375	2.30106	7.25857	
2943	37.81	1	100.0	30.375	2.24615	7.48180	
2954	37.92	1	100.0	30.375	2.30000	7.10681	
2965	38.03	1	100.0	30.375	2.49455	7.52335	
2976	38.14	1	101.0	30.375	2.71654	8.30596	

Also, i will be dropping the "magnetic" measurements as they are not so useful in the detection of activities performed by an individual which is the aim of this report.

In [16]:
```
drop_columns = ['handMagne1', 'handMagne2', 'handMagne3', 'chestMag
```

In [17]:
```
df_rawN = df_raw1.drop(drop_columns, axis = 1)
```

In [18]:
```
df_rawN
```

Out[18]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	h
2932	37.70	1	100.0	30.375	2.30106	7.25857	
2943	37.81	1	100.0	30.375	2.24615	7.48180	
2954	37.92	1	100.0	30.375	2.30000	7.10681	
2965	38.03	1	100.0	30.375	2.49455	7.52335	
2976	38.14	1	101.0	30.375	2.71654	8.30596	
...	...	...	...	...	...	...	
2871975	94.66	24	162.0	25.125	4.78601	6.75444	
2871986	94.77	24	162.0	25.125	4.34732	6.90337	
2871997	94.88	24	162.0	25.125	4.70704	6.59291	
2872007	94.98	24	162.0	25.125	4.81452	6.51482	
2872018	95.09	24	162.0	25.125	5.13914	5.63724	

177408 rows × 25 columns

In [19]:
```
df_rawN.isnull().sum()
```

Out[19]:
```
timestamp           0
activity_id         0
heartrate           0
handTemperature     0
handAcc16_1         0
handAcc16_2         0
handAcc16_3         0
handGyro1           0
handGyro2           0
handGyro3           0
chestTemperature    0
chestAcc16_1        0
chestAcc16_2        0
chestAcc16_3        0
chestGyro1          0
chestGyro2          0
chestGyro3          0
ankleTemperature    0
ankleAcc16_1        0
ankleAcc16_2        0
ankleAcc16_3        0
ankleGyro1          0
ankleGyro2          0
ankleGyro3          0
SubjectID           0
dtype: int64
```

It is observed from the cell above that there is no missing/null value still present in the data frame and now we move on to Exploratory Data Analysis.

Exploratory Data Analysis

To gain more insight on the data, it is imperative to perform exploratory data analysis and for that i will be splitting the data into "train" and "test" sets. The train set will contain 80% of the data while the test set will contain 20% of the data and the EDA will be performed on the train set.

In [20]:
```
#split data into train and test set
df_train = df_rawN.sample(frac=0.8, random_state=1)
df_test = df_rawN.drop(df_train.index)
```

In [21]:
```
df_train
```

Out[21]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	h
68817	696.55	3	98.0	33.0000	3.220410	9.288790	
2773370	3179.34	7	117.0	30.6875	0.664431	-2.854710	
2622702	1672.66	12	98.0	34.3750	-15.613000	-0.937918	
331735	3325.73	5	173.0	30.3750	-10.538700	3.428580	
1747730	3424.74	5	161.0	33.8125	10.868100	-0.198022	
...	...	...	...	...	...	...	
1880404	1003.84	17	94.0	33.9375	-2.965030	5.032000	
1756819	3515.63	5	168.0	33.8125	-2.457860	15.232000	
91282	921.20	17	95.0	33.2500	5.235950	7.221640	
2547441	920.05	17	86.0	34.6875	0.738579	-6.168880	
2793266	3378.30	7	125.0	30.8125	0.627402	-0.846997	

141926 rows × 25 columns

In [22]:
```
df_test
```

Out[22]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	h
2932	37.70	1	100.0	30.3750	2.301060	7.25857	
2987	38.25	1	101.0	30.3750	2.549540	7.63122	
3009	38.47	1	101.0	30.3750	2.736260	7.94195	
3031	38.69	1	101.0	30.3750	1.813200	6.85639	
3107	39.45	1	101.0	30.4375	-2.157460	10.30340	
...	...	...	...	...	...	...	
2871876	93.67	24	161.0	25.1250	-4.193620	5.94664	
2871898	93.89	24	161.0	25.1250	-0.073787	6.97545	
2871931	94.22	24	162.0	25.1250	3.854480	5.54336	
2871942	94.33	24	162.0	25.1250	3.767120	6.33631	
2871953	94.44	24	162.0	25.1250	4.098400	6.21785	

35482 rows × 25 columns

it is imperative to check some statistics with the "describe" method from pandas as it provides further results needed to know as to how the data is being processed for analysis.

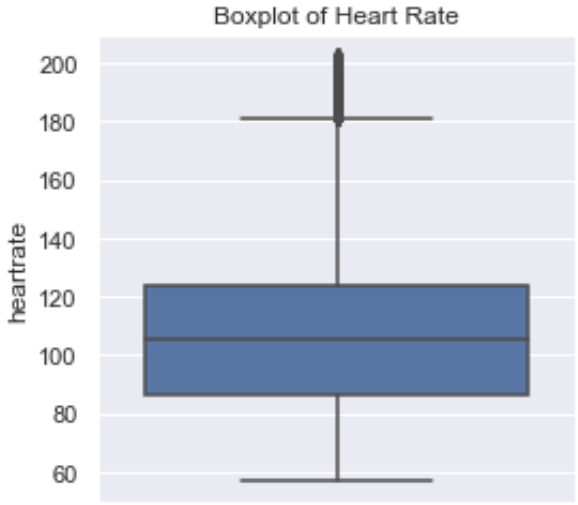
In [23]:
```
df_train.describe()
```

Out[23]:

	timestamp	activity_id	heartrate	handTemperature	handAcc16_1	ha
count	141926.000000	141926.000000	141926.000000	141926.000000	141926.000000	141
mean	1705.094311	8.078034	107.490044	32.749716	-4.977467	
std	1093.879660	6.175274	26.990819	1.794006	6.253213	
min	31.220000	1.000000	57.000000	24.875000	-94.135900	
25%	742.772500	3.000000	86.000000	31.687500	-8.971937	
50%	1479.530000	6.000000	105.000000	33.125000	-5.460830	
75%	2665.130000	13.000000	124.000000	34.062500	-0.972719	
max	4245.650000	24.000000	202.000000	35.500000	60.912600	

It is observed from the description of the train data above that the mean heartrate is 107.5 with the minimum heart rate being 57 and the maximum heart being 202. The quartiles shown in the description above can be analysed by plotting a boxplot that will aid in understanding the outliers.

In [24]:
```
import seaborn as sns

fig, ax = plt.subplots(figsize=(4,4))
plt.title("Boxplot of Heart Rate")
ax = sns.boxplot(y=df_train["heartrate"])
```

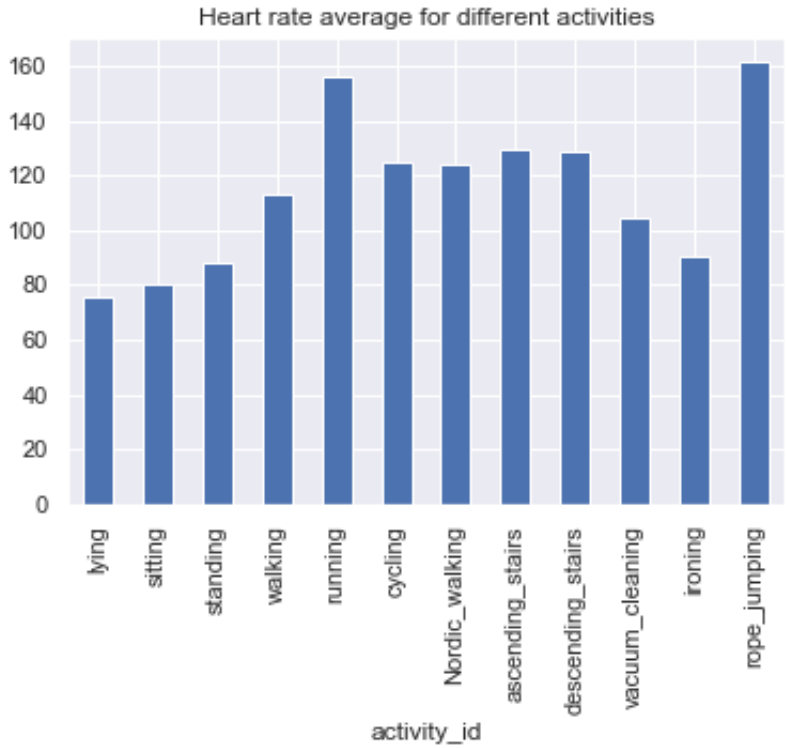


From looking at the boxplot, it is observed that the outliers have a heart rate range within 180 and 202. The highest quartile group (75%) starts from 124 and finishes at 180. The third quartile group starts from 105 and finishes at 124. The second quartile group starts at 86 and finishe at 124. While the first quartile group starts at 57 which is the lowest data point and finishes at 86-the inter quartile range.

In the following cells, i will be analyzing different activities/attributes in the dataframe

In [25]:
```
df_hr_act = df_train['heartrate'].groupby(df_train['activity_id']).
df_hr_act.index = df_hr_act.index.map(activity_id)
plt.title('Heart rate average for different activities')
df_hr_act.plot(kind='bar')
```

Out[25]:
```
<AxesSubplot:title={'center':'Heart rate average for different act
ivities'}, xlabel='activity_id'>
```



From the above bar chart it is evident that rope jumping and running are the most tedious activities as they have a heart rate higher than every other activity. However the slightly physically challenging activities like cycling, Nordic Walking, ascending and descending stairs have an average heart rate higher than the less tedious activities. While the physically easy activities like sitting, lying, standing and ironing have lower heart rates. Among all the activities, rope jumping has the highest heart rate of about 160 while lying has the least heart rate of about 75.

I move on to see the "hand temperature" data description for different activities

In [26]:
```
#To see how the temperature data description of the hand for differ
df_tmp_hand = df_train[['handTemperature']].groupby(df_train['activ
k = df_tmp_hand.describe()
k.index = df_tmp_hand.describe().index.map(activity_id)# To change
k
```

Out[26]:

	handTemperatur							
	count	mean	std	min	25%	50%	75%	ma
activity_id								
lying	14101.0	32.732067	1.404955	30.1875	31.3125	32.8750	33.8125	34.937
sitting	13503.0	33.266162	1.103005	31.5000	32.1875	33.5000	34.2500	35.062
standing	13927.0	33.638916	0.855892	32.1875	33.0000	33.5625	34.2500	35.250
walking	17403.0	32.294685	1.378107	28.6875	31.3125	32.5625	33.0625	34.875
running	7131.0	30.815160	2.095377	27.9375	28.8125	30.8125	33.8125	33.875
cycling	12027.0	31.007691	1.994362	27.5625	29.3125	31.0625	32.7500	34.687
Nordic_walking	13818.0	31.527517	1.752760	28.9375	30.3125	30.7500	33.0000	34.937
ascending_stairs	8601.0	33.527744	0.873469	31.7500	32.5625	33.7500	34.1250	35.125
descending_stairs	7585.0	33.322322	0.882146	31.5625	32.4375	33.5625	33.8750	34.875
vacuum_cleaning	12869.0	34.177418	0.651571	33.0625	33.6875	34.1250	34.6250	35.500
ironing	17335.0	34.017351	0.772009	32.6250	33.4375	33.9375	34.6875	35.500
rope_jumping	3626.0	29.733384	2.497353	24.8750	28.4375	30.1875	30.8125	33.875

From the above cell, it is observed that "vacuum_cleaning" has the highest temperature mean followed by "ironing" which could be as a result of the emission of hot air from the machines. Activities like "rope jumping" and "running" have the lowest temperature mean.

I move on to further plot boxplots for temperature variations for the different activities

In [27]:
```
#Box plot for temperature variation for different activities
df_tmp_hand.boxplot(figsize = (25,25))
plt.show()
```

From the boxplot above, it is observed that "rope jumping"(24) has the maximum temperature variance while "vacuum cleaning"(16) has the least temperature variance.

For further exploratory data analysis, i move on to finding the correlation between variables in the dataframe using the "spearman correlation method".

In [28]:
```
#removing unsuitable columns for correlation
df_train1 = df_train.drop(['SubjectID','activity_id','timestamp'],a
```

In [29]:
```
df_train1.corr(method = 'spearman').style.background_gradient()
```

Out[29]:

	heartrate	handTemperature	handAcc16_1	handAcc16_2	handAcc16_3
heartrate	1.000000	-0.346925	-0.332908	0.017736	-0.283034
handTemperature	-0.346925	1.000000	0.056358	-0.004137	0.009933
handAcc16_1	-0.332908	0.056358	1.000000	0.080315	0.335664
handAcc16_2	0.017736	-0.004137	0.080315	1.000000	-0.044612
handAcc16_3	-0.283034	0.009933	0.335664	-0.044612	1.000000
handGyro1	-0.006982	-0.016461	0.047953	0.092580	0.001794
handGyro2	0.029049	-0.008307	-0.056976	0.001304	0.024598
handGyro3	0.003364	-0.002010	0.005026	-0.019967	-0.062115
chestTemperature	-0.040121	0.743749	-0.162817	-0.064645	-0.131251
chestAcc16_1	-0.043398	0.015175	0.014801	0.016522	-0.105142
chestAcc16_2	0.052684	0.065422	-0.331452	0.180100	-0.100394
chestAcc16_3	-0.410743	0.023660	0.333718	-0.129315	0.075126
chestGyro1	-0.019035	0.006505	-0.021655	0.003081	-0.011729
chestGyro2	0.032508	-0.009635	-0.038410	0.005062	-0.021102
chestGyro3	-0.032227	0.017979	0.047271	0.056927	0.043441
ankleTemperature	-0.056197	0.503030	-0.121514	0.089441	-0.064480
ankleAcc16_1	0.320740	-0.069841	-0.323069	0.095335	-0.204193
ankleAcc16_2	0.266147	-0.038192	-0.154564	0.052232	-0.074964
ankleAcc16_3	0.042111	-0.092984	-0.011891	0.045581	0.045915
ankleGyro1	-0.035970	0.018988	0.059893	0.012596	0.067054
ankleGyro2	0.033539	-0.006912	-0.045442	-0.053136	-0.045747
ankleGyro3	-0.090423	0.049524	0.142910	-0.011981	0.086750

From the above cell, it is observed that the gyroscopes do not correlate with any other data and seems not useful.

Observed also is the negative correlation between the temperatures and heartrate while all the temperature measurements are positively correlated with each other.

Also, ankle accelerometer for all 3 instances seem to have the most positive correlation with heartrate compared to others.

Now we move further to hypothesis testing

Hypothesis Testing

From the Exploratory data analysis done, it is observed that the ankle accelerometer for all 3 instances seem to have the most positive correlation with heartrate compared to others. Hence i will be testing the hypothesis that "An increase in the acceleration of the ankle is directly proportional to an increase in heartrate".

Null Hypothesis: An increase in the acceleration of the ankle is not directly proportional to an increase in heartrate".

Alternate Hypothesis: An increase in the acceleration of the ankle is directly proportional to an increase in heartrate".

In [30]:
```
#Hypothesis test function
def hypothesis_test(data1,data2):
  correlation,p_value = spearmanr(data1,data2)
  print('correlation between the parameters = {}'.format(correlatio
  print('p_value = {}'.format(p_value))
  for i in [0.01 , 0.05, 0.1]:
      if p_value < i:
        print('The null Hypothesis is rejected')
        break

      else:
        print('Failed to reject the null Hypothesis')
```

For the sake of the hypothesis test, i will be using just one instance of the ankle acceleration "ankleAcc16_1" which is more positively correlated to the heartrate than the other instances

In [31]:
```
hypothesis_test(df_test['ankleAcc16_1'],df_test['heartrate'])
```

```
correlation between the parameters = 0.320075504219535
p_value = 0.0
The null Hypothesis is rejected
```

Obtaining a p-value less than 0.05 indicates strong evidence against the null hypothesis, as there is a less than 5% chance that it is true. So we reject the null hypothesis that states that "An increase in the acceleration of the ankle is not directly proportional to an increase in heartrate" and retain the alternate hypothesis that states that "An increase in the acceleration of the ankle is directly proportional to an increase in heartrate."

Modelling

After testing my hypothesis, i move on to modelling and if we recall, the goal of this report is to develop a hardware/software that can determine the amount/type of physical activity performed by an individual.

To be able to predict the activity performed by an individual, developing and testing some models to carry out the operation is imperative.

In [32]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,cross_val_scor
from sklearn.metrics import precision_score,recall_score, f1_score,
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [33]:
```
df_train = df_train.drop(['SubjectID','timestamp'],axis =1)
```

In [34]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler,RobustScaler

#apply scaling to all columns except subject and activity
scaler = RobustScaler()
df_scaled = df_train.copy()
df_scaled_test = df_test.copy()

df_scaled.iloc[:,1:41] = scaler.fit_transform(df_scaled.iloc[:,1:41
df_scaled_test.iloc[:,1:41] = scaler.fit_transform(df_scaled_test.i

df_scaled.head()
```

Out[34]:

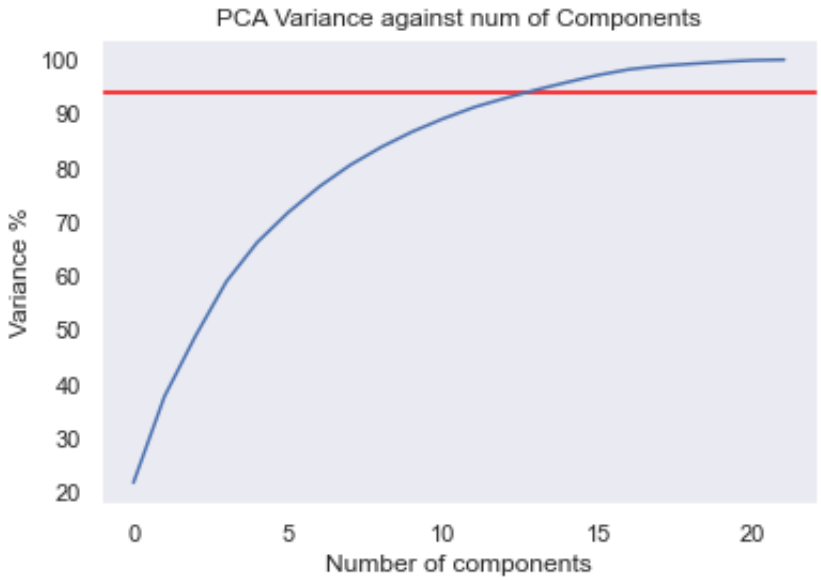
	activity_id	heartrate	handTemperature	handAcc16_1	handAcc16_2	handAcc16_3
68817	3	-0.184211	-0.052632	1.085261	1.066579	-0.523858
2773370	7	0.315789	-1.026316	0.765732	-1.179732	-0.193581
2622702	12	-0.184211	0.526316	-1.269145	-0.825163	-0.863913
331735	5	1.789474	-1.157895	-0.634796	-0.017446	-0.759301
1747730	5	1.473684	0.289474	2.041316	-0.688296	-0.876520

In [35]:
```
X_train = df_scaled.drop('activity_id', axis=1).values
y_train = df_scaled['activity_id'].values

# Test Dataset
X_test = df_scaled.drop('activity_id', axis=1).values
y_test = df_scaled['activity_id'].values
```

In [36]:
```
def get_metrics (y_true,y_pred):# function to get accuracy,precisio
    acc = accuracy_score(y_true, y_pred)
    p = precision_score(y_true, y_pred,average=None).mean()#average
    r = recall_score(y_true, y_pred, average=None).mean()#average r
    f1 = f1_score(y_true, y_pred, average=None).mean()#average f1 s

    print("Accuracy:",acc)
    print("Precision:", p)
    print("Recall:", r)
    print("F1:", f1)
```

Dimensionality reduction using Principal Component Analysis(PCA)

In [37]:
```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_train)
var= pca.explained_variance_ratio_
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*

plt.title("PCA Variance against num of Components")
plt.ylabel("Variance %")
plt.xlabel("Number of components")
l = plt.axhline(94, color="red")

plt.plot(var1)
plt.grid()
```



We can observe from the graph above plotting the variance ratio against the number of components that both lines intersect at x = 13, which indicates that for our data, 13 features are required to capture 94% of the variation.

In [38]:
```
pca = PCA(n_components=13)
X_train=pca.fit_transform(X_train)
X_test=pca.fit_transform(X_test)
```

In [39]:
```
X_train_df = pd.DataFrame(X_train,columns=['PCA_1','PCA_2','PCA_3',
X_train_df
```

Out[39]:

	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5	PCA_6	PCA_7	PC
0	0.048179	-0.013182	-0.075063	-0.029107	0.206312	-0.236477	0.319640	-0.46
1	-2.361642	-0.423118	0.720398	1.658753	0.667414	-0.344554	0.436879	-1.10
2	-8.191242	1.058814	13.204969	0.377203	-2.930123	0.530401	1.109078	-4.59
3	5.395772	6.698376	-2.895275	10.272973	-8.422572	2.004811	1.879913	0.05
4	2.853552	-1.216991	5.276446	-3.803120	0.320415	1.457067	-2.566704	1.77
...	...	...	...	...	...	...	...	
141921	-0.005890	0.247404	1.152142	1.275659	-2.044003	1.887403	-0.201962	-1.29
141922	-15.563172	-1.048566	-6.054545	2.199785	4.431146	-1.967364	-1.143897	0.09
141923	0.374691	-0.065378	-0.038743	0.467259	-1.323707	-0.415102	-0.159716	-0.77
141924	-0.249302	0.044441	0.810664	2.901721	-0.006263	-0.670164	-0.418428	-1.13
141925	-2.478990	-0.296094	-0.320361	1.938155	0.489766	-0.146568	0.318072	-1.24

141926 rows × 13 columns

Support Vector Classifier (SVC) Model with PCA

In [40]:
```
from sklearn.svm import SVC
```

In [41]:
```
%%time
SVCmodel = SVC(kernel = 'rbf')
SVCmodel.fit(X_train, y_train)
```

```
CPU times: user 24min 8s, sys: 18.7 s, total: 24min 26s
Wall time: 25min 1s
```

Out[41]:
```
SVC()
```

It can be seen that the SVC model takes 25 minutes to train the data using the PCA method.

In [42]:
```
%%time
SVCmodel_y_pred = SVCmodel.predict(X_test)
print(len(SVCmodel_y_pred))
print(len(y_test))
#compare the first 5 predictions with original classification
print(SVCmodel_y_pred[0:5])
print(y_test[0:5])
```

```
141926
141926
[ 2  7 12  5  5]
[ 3  7 12  5  5]
CPU times: user 36min 33s, sys: 8.16 s, total: 36min 42s
Wall time: 37min 16s
```

The SVC model takes 37 minutes to test the data using PCA method.

In [43]:
```
get_metrics(y_test,SVCmodel_y_pred)# to get the score
```

```
Accuracy: 0.7602905739610784
Precision: 0.8068220881457773
Recall: 0.7585967379527294
F1: 0.776839100584863
```

With an accuracy of 76, the SVC model using PCA gives good results but takes a long time to train/test the data.

I will move on to training and testing the data with Random Forest Classifier model.

Random Forest Classifier

In [77]:
```
#Using RandomForest model for classification
from sklearn.ensemble import RandomForestClassifier
```

In [78]:
```
%%time
RFmodel = RandomForestClassifier()#creating the model object
RFmodel.fit(X_train,y_train)
```

```
CPU times: user 30 s, sys: 219 ms, total: 30.2 s
Wall time: 30.8 s
```

Out[78]:
```
RandomForestClassifier()
```

The Random Forest Classifier only takes 30 seconds to train the model whereas the SVC model took 25 minutes.

In [89]:
```
%%time
RFmodel_y_pred = RFmodel.predict(X_test)
print(len(RFmodel_y_pred))
print(len(y_test))
print(RFmodel_y_pred[0:5])
print(y_test[0:5])
```

```
35482
35482
[ 5  1 16 17  6]
2114271     5
2166203     1
973135     16
2227352    17
1716930     6
Name: activity_id, dtype: int64
CPU times: user 879 ms, sys: 99.1 ms, total: 979 ms
Wall time: 1.06 s
```

The Random Forest Classifier only takes approximately 1 minute 6 seconds to test the model

In [82]:
```
get_metrics(y_test,RFmodel_y_pred)
```

```
Accuracy: 0.999323600698946
Precision: 0.9991527383552973
Recall: 0.9991858705889106
F1: 0.9991690375234925
```

With an accuracy of 99.93, the Random Forest Model from observation of the cell above gives nearly perfect scores on all counts!

Summary & Conclusion

I have analyzed the Physical Activity Monitoring (PAMAP2) dataset and gained various insights from it.

The first thing i did was to lay out the goal and requirements of the report afterwhich i imported necessary libraries and loaded the dataset.

After loading the dataset, i proceeded to cleaning the data(dropping irrelevant columns,null values and interpolation to cater for missing values). After cleaning, i moved on to perform some exploratory data analysis but first i randomly split the data into "train" and "test" sets to avoid overfitting before i commenced my analysis on the "train" data. From my EDA, the following observations were made:

1) the gyroscopes do not correlate with any other data and seems not useful.

2) there is a negative correlation between the temperatures and heartrate while all the temperature measurements are positively correlated with each other.

3) ankle accelerometer for all 3 instances seem to have the most positive correlation with heartrate compared to others.

And the third observation led to me testing the hypothesis that "An increase in the acceleration of the ankle is directly proportional to an increase in heartrate". After testing my hypothesis, i moved on to modeling and from training and testing the data with 2 models: Support Vector Machine Model(with Principal Component Analysis) and the Random Forest mclassifier, the RFC has a very high accuracy and it also takes low time to calculate accuracy compared to the SVM.

In conclusion, it is my suggestion that the Random Forest Classifier should be used in developing the hardware and/or software for determining the type of physical activity carried out by an individual.

References

Donges, N. (2018). The Random Forest Algorithm – Towards Data Science. [online] Towards Data Science. Available at: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd (https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd)

Kyratzis A. (2019). PAMAP2-Physical-Activity-Monitoring-Data-Analysis-and-ML. Available at: https://github.com/andreasKyratzis/PAMAP2-Physical-Activity-Monitoring-Data-Analysis-and-ML (https://github.com/andreasKyratzis/PAMAP2-Physical-Activity-Monitoring-Data-Analysis-and-ML)