Data Science Research Methods

PAMAP2 Dataset Report

1. Introduction

PAMAP2 data set is a collection of 18 different physical activities; like walking, cycling, playing soccer among others performed by 9 subjects. Each of these subjects were wearing 3 inertial measurement units (IMUs) along with a heart rate monitor. The data for each subject is stored separately in text files. Each row represents an enrty and in total there are 54 columns. However, orientation columns are not valid in this collection so we will drop it in the data cleaning step. The goal of the analysis is to extract some actionable insights from the data which can help in further development of some software/harware to measure the amount and type of physical activity performed by an indivisual. The rest of the report is structured in the following fashion:

- Data Preparation: reading and collating data from all text files
- Data Cleaning: performing sanity checks and treating erroneous data
- Exploratory Data Analysis: analysing data for any underlying trens and relationships between variables
- Hypothesis Testing: conducting statistical tests to confirm the observed relationships
- Model Development: developing a suitable model which builds upon the previous analyses

2. Data Preparation

In [193]:
```
# Importing required libraries
from matplotlib import pyplot as plt
import numpy as np
import math
from sklearn.model_selection import train_test_split
from scipy import stats
import pandas as pd
import os
# may nee to install PyPDF2 before using
# pip install PyPDF2
import PyPDF2
import re
import seaborn as sns
from statsmodels.stats.weightstats import ztest as ztest
import warnings
warnings.filterwarnings('ignore') # setting ignore as a parameter
```

To read data from 9 different text files firstly a list of all the file names are required. Below code is reading the text file names located in the "Protocol" folder.

In [2]:
```
# Reading data from directory
files = os.listdir(".\Protocol")
print(files)
```

```
['subject101.dat', 'subject102.dat', 'subject103.dat', 'subject104.dat', 'subject105.dat', 'subject106.dat', 'subject107.dat', 'subject1
08.dat', 'subject109.dat']
```

In [3]:
```
# creating columns names
columnNames = ["timeStamp", "activityID", "heartRate"]

for imu in ["hand", "chest", "ankle"]:
    for col in ["Temp", "Acc16_1", "Acc16_2", "Acc16_3", "Acc6_1", "Acc6_2", "Acc6_3", "Gyr_1", "Gyr_2", "Gyr_3",
                "Magn_1", "Magn_2", "Magn_3", "Orientation_1", "Orientation_2", "Orientation_3", "Orientation_4"]:
        columnNames.append(imu + col)

print(columnNames)
```

```
['timeStamp', 'activityID', 'heartRate', 'handTemp', 'handAcc16_1', 'handAcc16_2', 'handAcc16_3', 'handAcc6_1', 'handAcc6_2', 'handAcc6_
3', 'handGyr_1', 'handGyr_2', 'handGyr_3', 'handMagn_1', 'handMagn_2', 'handMagn_3', 'handOrientation_1', 'handOrientation_2', 'handOrie
ntation_3', 'handOrientation_4', 'chestTemp', 'chestAcc16_1', 'chestAcc16_2', 'chestAcc16_3', 'chestAcc6_1', 'chestAcc6_2', 'chestAcc6_
3', 'chestGyr_1', 'chestGyr_2', 'chestGyr_3', 'chestMagn_1', 'chestMagn_2', 'chestMagn_3', 'chestOrientation_1', 'chestOrientation_2',
'chestOrientation_3', 'chestOrientation_4', 'ankleTemp', 'ankleAcc16_1', 'ankleAcc16_2', 'ankleAcc16_3', 'ankleAcc6_1', 'ankleAcc6_2',
'ankleAcc6_3', 'ankleGyr_1', 'ankleGyr_2', 'ankleGyr_3', 'ankleMagn_1', 'ankleMagn_2', 'ankleMagn_3', 'ankleOrientation_1', 'ankleOrient
ation_2', 'ankleOrientation_3', 'ankleOrientation_4']
```

The activity names will also be required to map to the activityIDs for analysis. Below code chunk creates a dictionary of activities with IDs as keys and activity names as the values.

In [4]:
```
# creating a pdf file object
pdfFileObj = open('PerformedActivitiesSummary.pdf', 'rb')

# creating a pdf reader object
PdfReader = PyPDF2.PdfReader(pdfFileObj)

# creating a page object
pageObj = PdfReader.pages[0]

# extracting text from page
activityList1 = re.findall(r'[0-9]+ - [aA-z]+ ?[a-z]+', pageObj.extract_text())

# creating a dictionary of
activityIDList = {re.findall(r'[0-9]+', activity)[0]: re.findall(r'[aA-z]+ ?[a-z]+', activity)[0].replace(" ", "_") for
                  activity in activityList1}
activityIDList["0"] = "transient"
print(activityIDList)
# closing the pdf file object
pdfFileObj.close()
```

```
{'1': 'lying', '2': 'sitting', '3': 'standing', '4': 'walking', '5': 'running', '6': 'cycling', '7': 'Nordic_walking', '9': 'watching',
'10': 'computer_work', '11': 'car_driving', '12': 'ascending_stairs', '13': 'descending_stairs', '16': 'vacuum_cleaning', '17': 'ironin
g', '18': 'folding_laundry', '19': 'house_cleaning', '20': 'playing_soccer', '24': 'rope_jumping', '0': 'transient'}
```

To read and collate all the text files into one pandas data frame a function is created which takes the directory path of the text files and returns the collated data frame with all the columns from the text files.

In [5]:
```
# Function to read data from the text files

def read_files(file_path):
    collatedData = pd.DataFrame()
    fileList = os.listdir(file_path)
    for txtFile in fileList:
        protocolDf = pd.read_csv(file_path + "/" + txtFile, sep="\s+", header=None, names=columnNames)
        protocolDf["subjectID"] = int(re.findall(r'[0-9]+', txtFile)[0])
        collatedData = pd.concat([collatedData, protocolDf], ignore_index=True)
    collatedData.reset_index(inplace=True, drop=True)
    return collatedData
```

In [6]:
```
# Reading and collating data into one pandas dataframe

file_path = "./Protocol"
collatedData = read_files(file_path)
```

In [7]:
```
collatedData.head()
```

Out[7]:

timeStamp	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handAcc6_1	handAcc6_2	handAcc6_3	...	ankleGyr_2	ankleGyr_3	ankleMagn_1
8.38	0	104.0	30.0	2.37223	8.60074	3.51048	2.43954	8.76165	3.35465	...	0.009250	-0.017580	-61.1888
8.39	0	NaN	30.0	2.18837	8.56560	3.66179	2.39494	8.55081	3.64207	...	-0.004638	0.000368	-59.8479
8.40	0	NaN	30.0	2.37357	8.60107	3.54898	2.30514	8.53644	3.73280	...	0.000148	0.022495	-60.7361
8.41	0	NaN	30.0	2.07473	8.52853	3.66021	2.33528	8.53622	3.73277	...	-0.020301	0.011275	-60.4091
8.42	0	NaN	30.0	2.22936	8.83122	3.70000	2.23055	8.59741	3.76295	...	-0.014303	-0.002823	-61.5199

ows × 55 columns

From the first look at the data it can be observed that there are some NaN and other undesired data which can issues further down the analysis. The following section focuses on cleaning data to make it suitable for further analysis.

3. Data Cleaning

This step will ensure that the data is ready for further analysis. As mentioned in the "readme.pdf" file, which describes the dataset in detail, that orientation data is invalid in this data collection, therefore orientation columns can be deleted from the dataset altogether. Furthermore, as we observed earlier that there are some rows which contain transient movement information and are labeled as 0. These rows can also be dropped as we only need the actual activity data. Moreover, as mentioned in the "PerformedActivitiesSummary.pdf" document there is some missing data due to two main reasons:

1. Due to the use of wireless sensors.
2. Problems with hardware setup

As this data is a timeseries we can impute the missing values using linear interpolation. But we should only consider data from one subject while interpolating. Also we sould first interpolate the data and then remove any transient state information for better results.

In [11]:
```
# Data cleaning function
def dataCleaner(dirtyData):
    remCols = columnNames[16:20]+columnNames[33:37]+columnNames[50:54]

    # remove orientation columns
    dirtyData = dirtyData.drop(remCols, axis = 1)

    # Converting data to numeric
    dirtyData = dirtyData.apply(pd.to_numeric, errors = 'coerce')

    # Interpolating data
    for ID in collatedData['subjectID'].unique():
        dirtyData[dirtyData['subjectID']==ID] = dirtyData[dirtyData['subjectID']==ID].interpolate()

    # Removing Transient state data
    dirtyData = dirtyData[dirtyData['activityID'] != 0]
    dirtyData.reset_index(inplace=True, drop=True)
    return dirtyData
```

3.1 Changes made to dataset

Following changes were made, in the described order, to the data to render it useful for further steps:

1. Removed orientation columns as they are invalid for these measurements.
2. Converted all the columns to numeric data in case any data column was not numeric.
3. Interpolated the timeseries data using in-built pandas interpolate() function. Please note that the interpolation has been done separately for each subject ID so as to avoid any faulty interpolation.

In [12]:
```
cleanData = dataCleaner(collatedData)
```

In [13]:
```
cleanData.head()
```

Out[13]:

	timeStamp	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handAcc6_1	handAcc6_2	handAcc6_3	...	ankleAcc6_1	ankleAcc6_2	ankleAc
0	37.66	1	100.0	30.375	2.21530	8.27915	5.58753	2.24689	8.55387	5.77143	...	9.63162	-1.76757	0.26
1	37.67	1	100.0	30.375	2.29196	7.67288	5.74467	2.27373	8.14592	5.78739	...	9.58649	-1.75247	0.25
2	37.68	1	100.0	30.375	2.29090	7.14240	5.82342	2.26966	7.66268	5.78846	...	9.60196	-1.73721	0.35
3	37.69	1	100.0	30.375	2.21800	7.14365	5.89930	2.22177	7.25535	5.88000	...	9.58674	-1.78264	0.31
4	37.70	1	100.0	30.375	2.30106	7.25857	6.09259	2.20720	7.24042	5.95555	...	9.64677	-1.75240	0.29

5 rows × 43 columns

In [19]:
```
# Checking if any Nan values remaining in the dataset
cleanData.isna().sum()
```

Out[19]:
```
timeStamp       0
activityID      0
heartRate       0
handTemp        0
handAcc16_1     0
handAcc16_2     0
handAcc16_3     0
handAcc6_1      0
handAcc6_2      0
handAcc6_3      0
handGyr_1       0
handGyr_2       0
handGyr_3       0
handMagn_1      0
handMagn_2      0
handMagn_3      0
chestTemp       0
chestAcc16_1    0
chestAcc16_2    0
chestAcc16_3    0
chestAcc6_1     0
chestAcc6_2     0
chestAcc6_3     0
chestGyr_1      0
chestGyr_2      0
chestGyr_3      0
chestMagn_1     0
chestMagn_2     0
chestMagn_3     0
ankleTemp       0
ankleAcc16_1    0
ankleAcc16_2    0
ankleAcc16_3    0
ankleAcc6_1     0
ankleAcc6_2     0
ankleAcc6_3     0
ankleGyr_1      0
ankleGyr_2      0
ankleGyr_3      0
ankleMagn_1     0
ankleMagn_2     0
ankleMagn_3     0
subjectID       0
dtype: int64
```

```
<p>It can be seen that all the NaN values are now treated and the data is ready for further analysis.</p>
```

4. Exploratory Data Analysis

Before starting any analysis the data should be split into training and test datasets so as to prevent the model from overfitting and provide a test sample for analysing the predicting power of the models.

4.1 Splitting Data

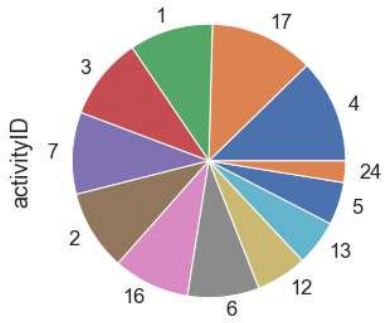
Before splitting the data into training and test samples it is important to know which sampling methodology is suitable for the given dataset. It can be observed from the below piechart that the data is not proportionately divided among the different activity groups hence random sampling can create even more skewed distributions in the training/testing samples. To avoid this from happening we can use stratified sampling approach which ensures the proportionate distribution.

In [158]:
```
# Defining plotting function
def nicePlots(plottype,data,xaxis,yaxis=None,xlabel = None,ylabel = None,title = None):
    if xlabel == None:
        xlabel = xaxis
    if ylabel == None:
        ylabel = yaxis
    plt.figure(figsize=(20,5))
    sns.set_style("whitegrid")
    sns.set(font_scale = 1.5)
    if plottype == "scatter":
        sns.scatterplot(data = data, x = xaxis, y = yaxis)
    elif plottype == "boxplot":
        plt.figure(figsize=(15,8))
        sns.boxplot(data = data, x=xaxis, y=yaxis,orient = "h")
    elif plottype == "barplot":
        sns.barplot(x=xaxis, y=yaxis, data=data)
    elif plottype == "piechart":
        data[xaxis].value_counts().plot(kind="pie", label=xaxis, title=title)

    if plottype != "piechart":
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title)
    return plt
```

In [112]:
```
nicePlots("piechart",cleanData, 'activityID')
```

Out[112]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kshit\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In [113]:
```
# Function to create stratified samples
def stratSample(df_input, stratify_colname, frac_train, random_state):
    X = df_input # Contains all columns.
    y = df_input[[stratify_colname]] # Dataframe of just the column on which to stratify.

    # Split original dataframe into train and temp dataframes.
    df_train, df_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                          test_size=(1.0 - frac_train),
                                                          random_state=random_state)
    df_train.reset_index(inplace=True, drop=True)
    df_test.reset_index(inplace=True, drop=True)
    return df_train, df_test
```

In [114]:
```
dataTrain, dataTest = stratSample(cleanData, "activityID", 0.67, 1)
dataTrain.head()
```

Out[114]:

	timeStamp	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handAcc6_1	handAcc6_2	handAcc6_3	...	ankleAcc6_1	ankleAcc6_2	ankleAc
0	2038.56	13	158.0	33.2500	-7.35968	2.28006	5.194930	-7.16176	2.71678	5.778740	...	8.72863	-2.997710	-3.6
1	384.34	2	85.0	32.1875	1.88464	8.23650	4.774480	1.92184	8.31384	4.987040	...	-1.22488	0.955483	-9.6
2	726.66	3	71.0	33.2500	-9.31532	2.44787	1.046920	-9.23365	2.50123	1.522910	...	9.08857	0.344542	-4.0
3	2549.15	4	95.0	31.5625	-13.43790	4.44816	0.478085	-13.19810	4.91535	0.811943	...	9.18422	-3.359120	-2.4
4	3016.27	6	116.0	30.6875	-4.69561	3.34432	7.732920	-5.09554	3.98959	10.092700	...	16.81000	-3.298580	-3.8

5 rows × 43 columns

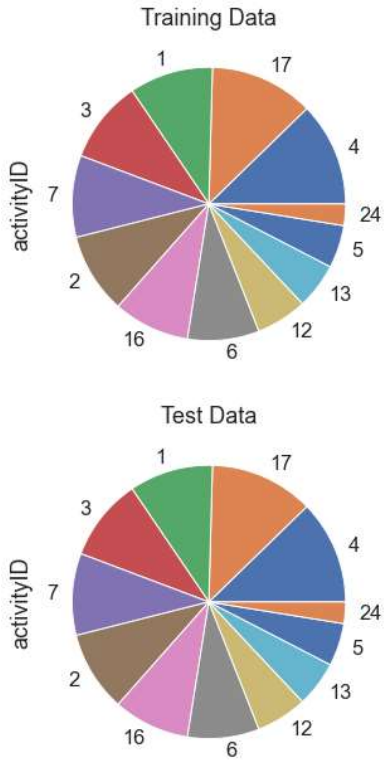
In [115]:
```
print("Taining sample size:",dataTrain.shape)
print("Testing sample size:",dataTest.shape)
```

```
Taining sample size: (1301724, 43)
Testing sample size: (641148, 43)
```

The data has been split into two parts namely, training and test datasets. As there is no dearth of data the it was decided to split the data in 2:1 proportion respectively. In other words 67% data was allocated to training data and rest for testing sample. This is a commonly used proportion used by data scientists.

In [116]:
```
# Checking the samples for proportions of activity IDs
nicePlots("piechart",dataTrain, 'activityID',title='Training Data')
nicePlots("piechart",dataTest, 'activityID',title='Test Data')
```

Out[116]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kshit\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



4.2 Data Visualisation

Now that we have our training and testing samples separated we can proceed with visualising the data to find underlying patterns and subsequently suggest a hypothesis to test based on the observed patterns. Firstly, let us summarize the data:

In [117]:
```
dataTrain.describe()
```

Out[117]:

	timeStamp	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handAcc6_1	handAcc6_2	handAcc6_3	...	ankleAcc6_
count	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	1.301724e+06	...	1.301724e+0
mean	1.705175e+03	8.081829e+00	1.074843e+02	3.275187e+01	-4.951930e+00	3.589473e+00	3.604053e+00	-4.883827e+00	3.578747e+00	3.788453e+00	...	9.376002e+0
std	1.093369e+03	6.174787e+00	2.699058e+01	1.793795e+00	6.244245e+00	6.880990e+00	3.952872e+00	6.248383e+00	6.587162e+00	3.942381e+00	...	6.065998e+0
min	3.120000e+01	1.000000e+00	5.700000e+01	2.487500e+01	-1.453670e+02	-1.043010e+02	-7.394970e+01	-6.112920e+01	-6.184170e+01	-6.157710e+01	...	-5.888080e+0
25%	7.444700e+02	3.000000e+00	8.600000e+01	3.168750e+01	-8.970333e+00	1.061095e+00	1.162575e+00	-8.867080e+00	1.059318e+00	1.363850e+00	...	8.396860e+0
50%	1.480295e+03	6.000000e+00	1.040000e+02	3.312500e+01	-5.449660e+00	3.528340e+00	3.433780e+00	-5.376205e+00	3.570500e+00	3.663430e+00	...	9.551000e+0
75%	2.664082e+03	1.300000e+01	1.240000e+02	3.406250e+01	-9.514508e-01	6.457210e+00	6.532330e+00	-8.991005e-01	6.460233e+00	6.776822e+00	...	1.028590e+0
max	4.245670e+03	2.400000e+01	2.020000e+02	3.550000e+01	6.285960e+01	1.556990e+02	1.577600e+02	4.621100e+01	6.225210e+01	6.192340e+01	...	6.196930e+0

8 rows × 43 columns

From common logic it is known that hear-rate is a function of the type and intensity of the activity performed by the subject. Therefore, hear-rate is going to be our variable of concern. We should be focusing on the variables which show higher level of association with hear-rate variable in order to capture the underlying patterns. Let us try to find the correlations among the variables so that we can shortlist the variables and avoid any information ovelap. Using two variables having a high correlation do not provide significantly more information than just using one of those, but increases computation requirements and chances of overfitting. Therefore, some variables should be dropped.

In [118]:
```
# Creating a Correlation matrix
formatCols = {i: '{:.2f}%' for i in dataTrain}
corrMatrix = (dataTrain.corr()*100).sort_index().sort_index(axis =1).style.background_gradient(cmap='Greens').format(formatCols)
corrMatrix
```

Out[118]:

	activityID	ankleAcc16_1	ankleAcc16_2	ankleAcc16_3	ankleAcc6_1	ankleAcc6_2	ankleAcc6_3	ankleGyr_1	ankleGyr_2	ankleGyr_3	ankleMagn_1	ankleMag
activityID	100.00%	13.35%	9.66%	8.44%	14.44%	10.38%	9.93%	-0.54%	1.29%	0.22%	-19.99%	-6
ankleAcc16_1	13.35%	100.00%	13.05%	7.25%	86.59%	13.51%	-12.66%	-0.59%	-5.04%	-3.66%	-20.68%	-11
ankleAcc16_2	9.66%	13.05%	100.00%	-13.07%	17.36%	83.04%	-10.03%	15.00%	-5.16%	14.11%	-11.05%	-5
ankleAcc16_3	8.44%	7.25%	-13.07%	100.00%	2.64%	-13.10%	68.33%	-9.07%	-7.04%	-2.04%	-2.76%	-5
ankleAcc6_1	14.44%	86.59%	17.36%	2.64%	100.00%	15.99%	-1.16%	-2.68%	2.01%	-6.39%	-21.45%	-12
ankleAcc6_2	10.38%	13.51%	83.04%	-13.10%	15.99%	100.00%	-14.79%	11.99%	-6.69%	8.04%	-11.89%	-5
ankleAcc6_3	9.93%	-12.66%	-10.03%	68.33%	-1.16%	-14.79%	100.00%	-8.24%	1.17%	-1.68%	-3.44%	-6
ankleGyr_1	-0.54%	-0.59%	15.00%	-9.07%	-2.68%	11.99%	-8.24%	100.00%	-6.64%	32.43%	-2.13%	5
ankleGyr_2	1.29%	-5.04%	-5.16%	-7.04%	2.01%	-6.69%	1.17%	-6.64%	100.00%	2.22%	2.14%	-2
ankleGyr_3	0.22%	-3.66%	14.11%	-2.04%	-6.39%	8.04%	-1.68%	32.43%	2.22%	100.00%	-0.55%	0
ankleMagn_1	-19.99%	-20.68%	-11.05%	-2.76%	-21.45%	-11.89%	-3.44%	-2.13%	2.14%	-0.55%	100.00%	6
ankleMagn_2	-6.93%	-11.45%	-5.17%	-5.48%	-12.28%	-5.54%	-6.49%	5.59%	-2.86%	0.90%	6.24%	100
ankleMagn_3	20.99%	11.80%	9.73%	-2.04%	12.43%	10.57%	-2.49%	-1.74%	-2.55%	-1.75%	-3.14%	2
ankleTemp	19.50%	9.22%	7.52%	8.30%	10.44%	7.96%	11.01%	-1.03%	0.85%	0.59%	-4.69%	-7
chestAcc16_1	-15.00%	-5.23%	4.52%	-6.77%	-5.81%	3.60%	-7.85%	6.95%	-4.26%	-4.25%	3.24%	3
chestAcc16_2	10.43%	31.83%	11.50%	-0.62%	35.02%	12.41%	-1.32%	-3.86%	2.97%	-10.97%	-9.55%	-15
chestAcc16_3	-42.87%	-31.28%	-23.52%	-12.65%	-35.08%	-24.57%	-15.03%	1.21%	-2.25%	6.96%	24.15%	29
chestAcc6_1	-14.76%	-5.30%	5.90%	-7.78%	-5.71%	5.16%	-8.58%	5.86%	-3.16%	-4.29%	3.08%	3
chestAcc6_2	10.26%	31.16%	11.60%	-0.70%	34.50%	12.83%	-0.92%	-4.38%	3.32%	-9.97%	-9.61%	-16
chestAcc6_3	-42.98%	-29.57%	-23.64%	-12.30%	-33.22%	-25.22%	-14.86%	1.17%	-2.05%	6.34%	24.11%	29
chestGyr_1	-0.06%	-2.06%	3.32%	-1.45%	-1.58%	3.31%	-1.82%	1.74%	-1.71%	-4.60%	1.16%	-0
chestGyr_2	-2.06%	5.75%	6.54%	-1.17%	5.00%	8.03%	-1.45%	12.46%	6.53%	-10.05%	-4.63%	7
chestGyr_3	0.42%	-7.20%	6.75%	0.87%	-8.00%	6.76%	0.85%	14.92%	-12.61%	27.43%	3.63%	-1
chestMagn_1	-24.18%	-25.89%	-16.05%	-3.96%	-27.67%	-17.34%	-4.67%	0.54%	1.42%	-0.01%	17.37%	29
chestMagn_2	-29.79%	-35.24%	-19.04%	-1.03%	-37.66%	-20.56%	-1.15%	-0.10%	1.99%	-0.31%	54.40%	18
chestMagn_3	26.54%	25.68%	19.23%	13.62%	27.42%	20.79%	15.81%	0.78%	0.48%	0.30%	-16.25%	-50
chestTemp	15.96%	10.59%	6.00%	-5.57%	11.83%	6.38%	-5.89%	-0.17%	-0.36%	0.25%	0.32%	-2
handAcc16_1	-12.99%	-28.48%	-9.11%	-0.94%	-31.96%	-10.04%	-1.40%	3.82%	-7.25%	9.34%	8.44%	20
handAcc16_2	3.92%	4.95%	11.39%	0.75%	6.11%	12.85%	0.74%	-0.43%	-3.04%	-3.73%	-4.95%	-9
handAcc16_3	-11.68%	-15.81%	-7.01%	1.88%	-16.37%	-7.58%	2.37%	1.23%	-1.11%	2.38%	6.14%	10
handAcc6_1	-12.93%	-27.95%	-9.03%	-0.98%	-31.40%	-10.03%	-1.41%	3.89%	-7.09%	8.21%	8.53%	21
handAcc6_2	4.26%	4.73%	11.48%	0.92%	5.70%	13.20%	0.93%	-0.23%	-3.17%	-1.40%	-5.31%	-10
handAcc6_3	-11.85%	-16.23%	-7.15%	1.75%	-16.92%	-7.60%	2.18%	1.19%	-1.10%	2.56%	6.26%	11
handGyr_1	1.98%	1.98%	1.31%	-1.12%	2.56%	0.84%	-1.55%	2.92%	-3.41%	13.92%	1.22%	-6
handGyr_2	2.78%	5.14%	0.41%	-3.09%	5.53%	1.09%	-3.58%	4.98%	9.83%	-3.42%	-0.77%	-0
handGyr_3	0.13%	-2.29%	5.16%	3.58%	-2.95%	6.78%	4.04%	-4.93%	-1.70%	-16.96%	-6.26%	12
handMagn_1	5.32%	17.16%	12.28%	6.69%	18.36%	13.26%	7.76%	4.86%	-8.42%	5.51%	-6.76%	-29
handMagn_2	-18.96%	-16.48%	-9.42%	-6.12%	-17.69%	-10.13%	-7.19%	7.34%	-1.74%	5.05%	23.80%	15
handMagn_3	-2.28%	11.32%	9.51%	-2.04%	12.03%	10.24%	-2.53%	0.79%	-4.76%	1.21%	16.42%	-20
handTemp	15.94%	-6.09%	-4.78%	-4.22%	-6.06%	-5.29%	-4.26%	-0.12%	3.31%	0.21%	9.28%	11
heartRate	26.52%	28.32%	19.32%	-0.33%	29.75%	21.05%	-0.80%	0.07%	-2.89%	0.12%	-32.66%	-24
subjectID	-0.23%	0.24%	-1.43%	-14.25%	0.05%	-1.61%	-16.56%	1.61%	-0.75%	-0.49%	19.36%	10
timeStamp	14.38%	32.46%	20.12%	0.66%	34.27%	21.85%	0.42%	0.09%	-4.54%	-0.14%	-30.93%	-23

High correlation can be observed between the two categories of acceleration variables with different resolutions, **6g and 16g**. We can drop the lower resolution variables because it is recommended to use the higher resolution accelerometer data is the "readme.pdf" document. Furthermore, gyroscope data is not correlated with heartrate or any other data therefore we can also drop gyroscope data from further analysis. On the other hand, there is significant correlations between hand accelerometers and hand temperature which is logical. Additionally, chest magnetometers seem to have a noticeable correlation with heartrate which is also logical as chest magnetometers are closest to the body core.

In [119]:
```
# Dropping low resolution accelerometer data
remCols = [col for col in columnNames if "Acc6" in col]
print(remCols)
dataTrain = dataTrain.drop(remCols, axis = 1)
print(dataTrain.shape)
```

```
['handAcc6_1', 'handAcc6_2', 'handAcc6_3', 'chestAcc6_1', 'chestAcc6_2', 'chestAcc6_3', 'ankleAcc6_1', 'ankleAcc6_2', 'ankleAcc6_3']
(1301724, 34)
```

In [120]:
```
# Dropping gyroscope data
remCols = [col for col in columnNames if "Gyr" in col]
print(remCols)
dataTrain = dataTrain.drop(remCols, axis = 1)
print(dataTrain.shape)
```

```
['handGyr_1', 'handGyr_2', 'handGyr_3', 'chestGyr_1', 'chestGyr_2', 'chestGyr_3', 'ankleGyr_1', 'ankleGyr_2', 'ankleGyr_3']
(1301724, 25)
```

Now that we have the required data we can go ahead and visualise the data to discover the underlying patterns. As mentioned above that heartrate is a function of the intensity of the activity, let us try and find if the heartrate actually varies significantly across different activities.

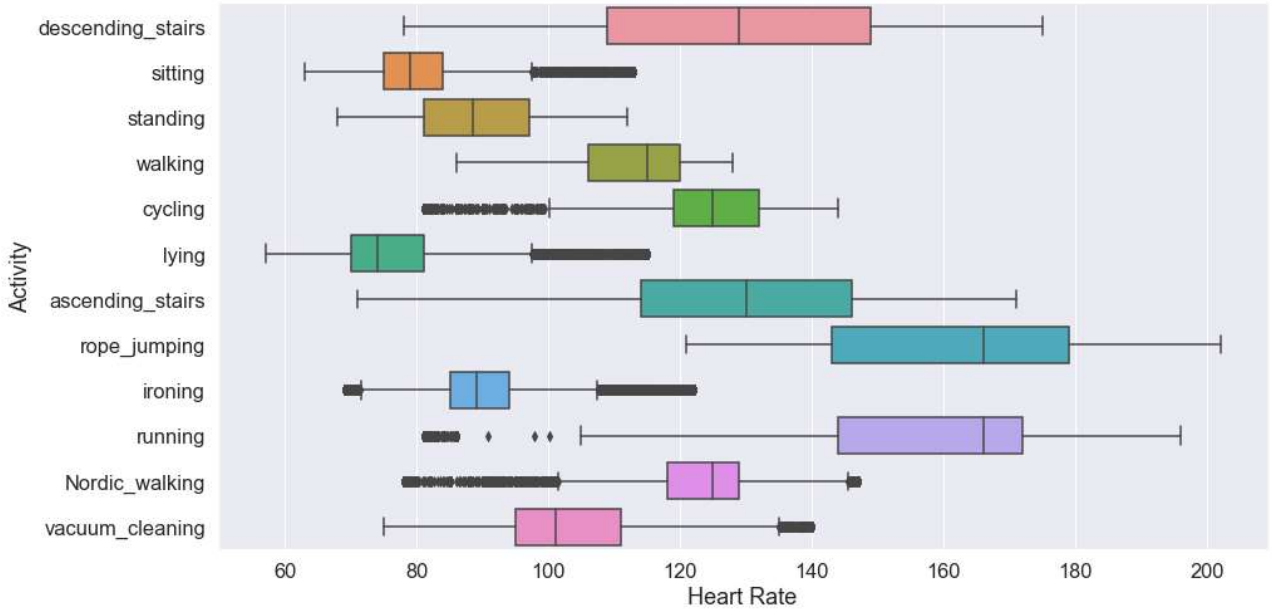
In [161]:
```
subData = dataTrain[['heartRate',"activityID"]]
subData["activityID"] = subData["activityID"].apply(str).map(activityIDList)
nicePlots('boxplot',subData,xaxis='heartRate',yaxis="activityID",xlabel = "Heart Rate",ylabel = "Activity",title = None)
```

```
C:\Users\kshit\AppData\Local\Temp\ipykernel_10788\1215648486.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  subData["activityID"] = subData["activityID"].apply(str).map(activityIDList)
```

Out[161]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kshit\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```

```
<Figure size 1440x360 with 0 Axes>
```

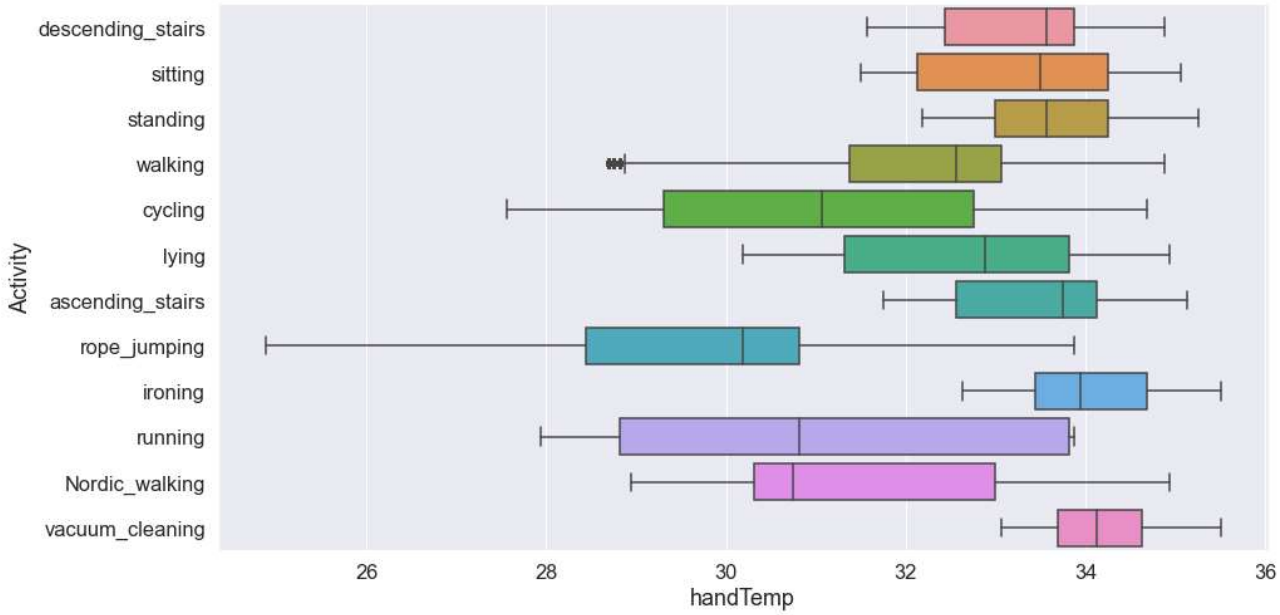


From the above boxplots it can be clearly seen that some activities are evidently more intense than other. For example, hear rate distributions for intense activities such as rope jumping, running, descending and ascending stairs are towards the higher end as compared with the other activities which can be labeled as less intense activities. Moreover, these intense activities have a wider distributions which also suggests that different subjects may have performed these activities in a unique manner but on an average the intensity remains towards the higher end of the spectrum. On the other hand the activities which require less effort have relatively narrower distributions as effort required is not much so the subject's bodies behave more similarly when at rest while when doing some activity.

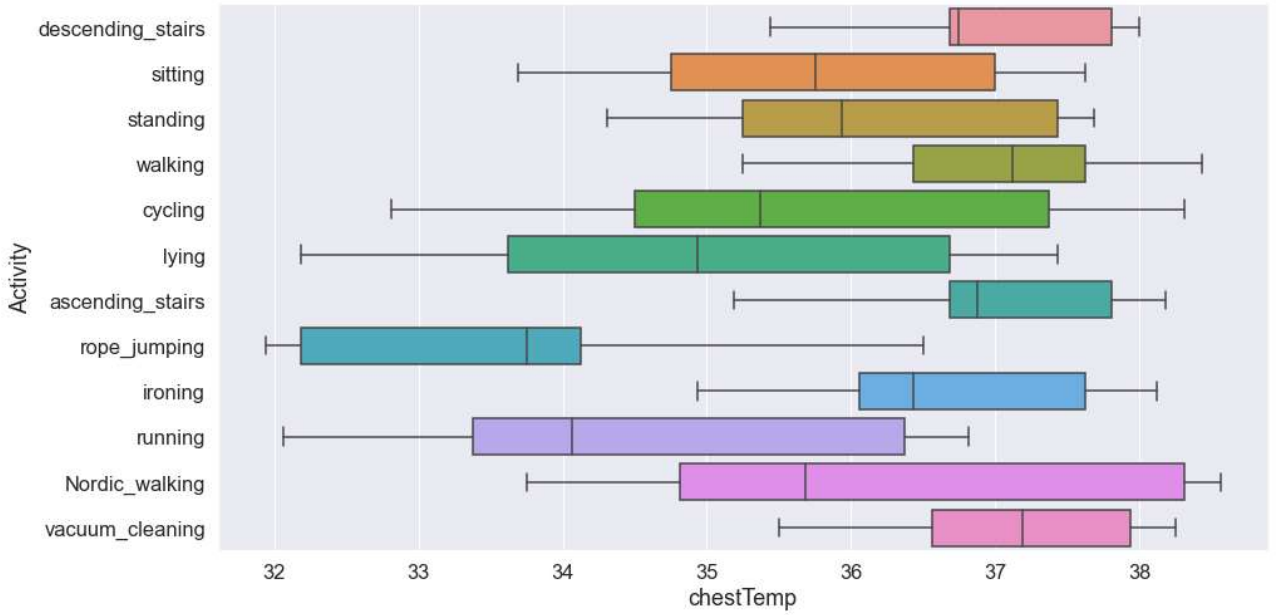
Let us try to create a similar plot for body temperatures to identify any trend in temperatures of body parts and activities being performed.

In [163]:
```
for plotme in ['handTemp','chestTemp','ankleTemp']:
    subData = dataTrain[[plotme,"activityID"]]
    subData["activityID"] = subData["activityID"].apply(str).map(activityIDList)
    nicePlots('boxplot',subData,xaxis=plotme,yaxis="activityID",xlabel = plotme,ylabel = "Activity",title = None)
```

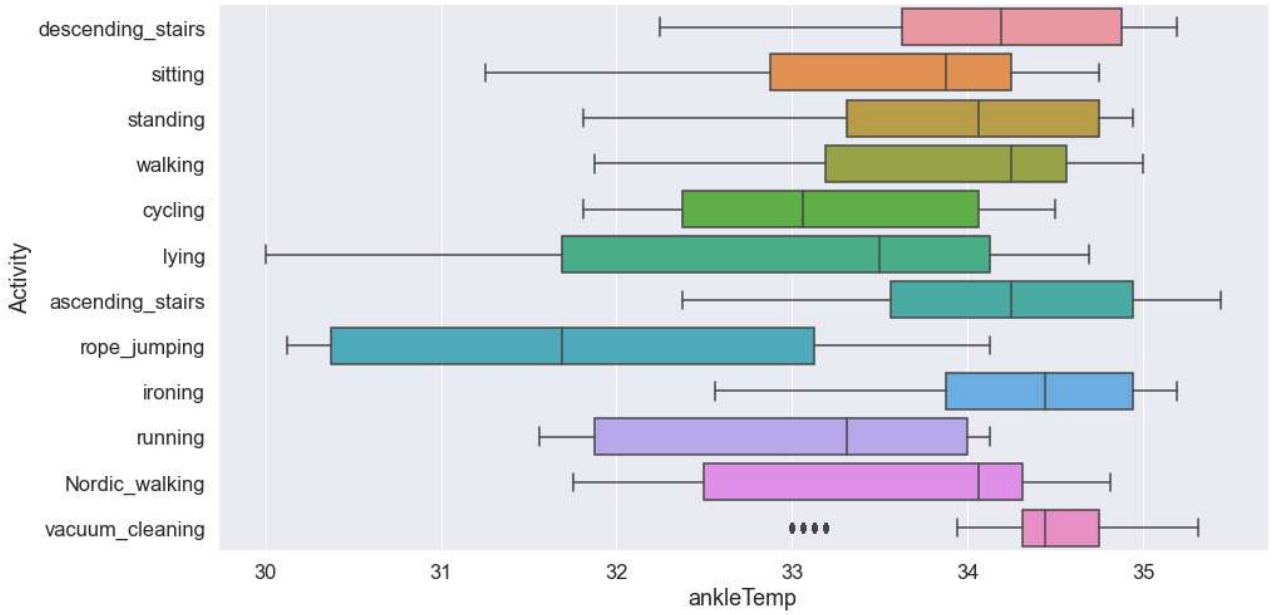
```
<Figure size 1440x360 with 0 Axes>
```




```
<Figure size 1440x360 with 0 Axes>
```



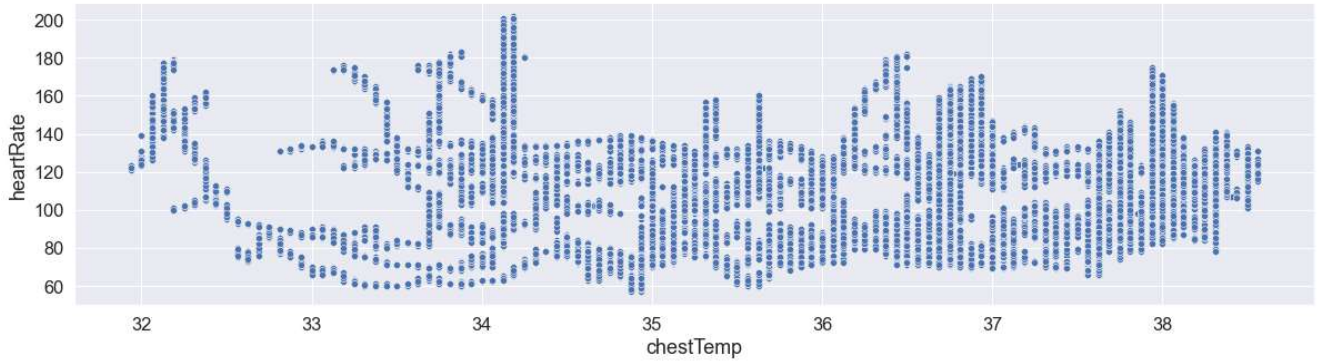

```
<Figure size 1440x360 with 0 Axes>
```

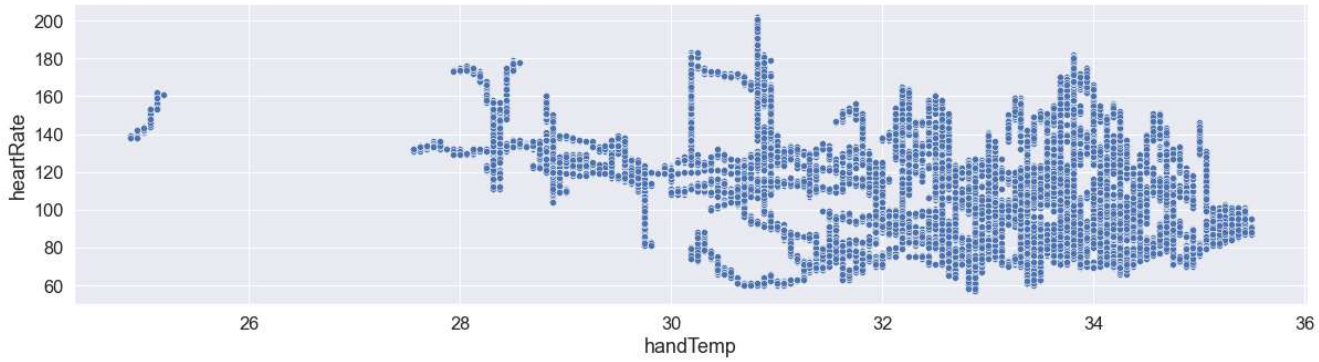


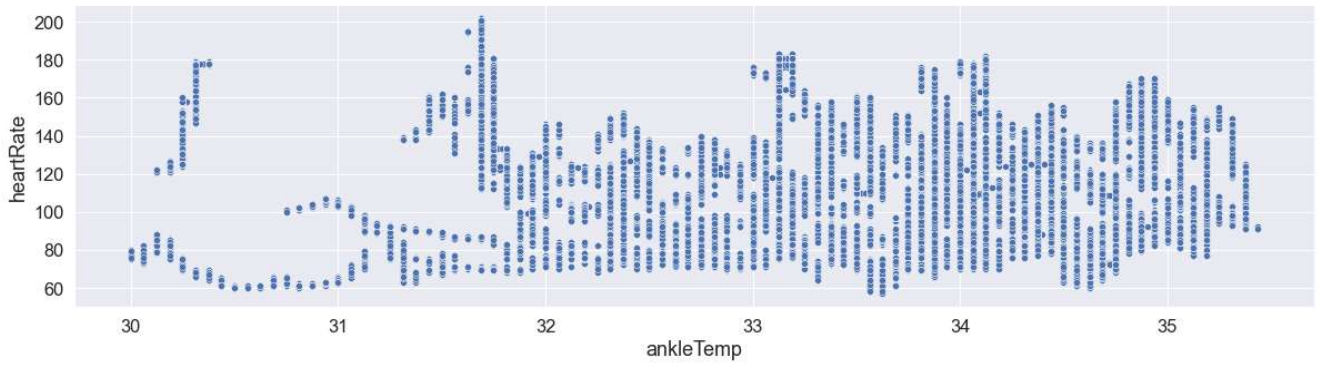
It seems, from the plots, that the median temperature while rope skipping is lower than other activities. There is no logical resoning for the same however we conduct a hypothesis test to statistically verify the observation.

In [148]:
```
nicePlots('scatter', dataTrain, 'chestTemp','heartRate')
nicePlots('scatter', dataTrain, 'handTemp','heartRate')
nicePlots('scatter', dataTrain, 'ankleTemp','heartRate')
```

Out[148]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kshit\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```







No clear pattern observed in the scatterplot between heartrate and body temperatures.

5. Hypothesis Testing

Hypothesis testing is conducted to find statistical evidence for the observations. We will conduct two hypothesis tests to check whether the observations made in previous sections are statistically significant or just random coincidence. Two observations for which hypothesis test will be conducted are mentioned below:

1. Mean heart rate of more intense activities like rope jumping, running, descending and ascending stairs is higher as compared to that other less intense activities.
2. Mean temperature while skipping ropes is lower as compared to other activies

5.1 Testing Hypothesis 1

We would like to know if the difference between the mean heart rate of subjects while performing intense activities like rope jumping, running, descending and ascending stairs is significantly different as compared to that other less intense activities.

- Null Hypothesis: H0 - There is no difference in the mean heart rates of subjects while performing more intense and less intense activities.
- Alternative Hypothesis: H1 - There is a significant difference between the mean heart rates of subjects while performing more intense and less intense activities.

The favourable outcome would be if we can reject the null hypothesis with a 95% confidence interval. In other words, we would like to confirm that 95% of the times the means are statistically different.

In [208]:
```
dataTrain['activityIntense'] = ['intense' if i in [5,12,13,24] else 'not_intense' for i in  dataTrain['activityID']]
```

In [223]:
```
# Function to Conduct Hypothesis testing
def hypTest(data, bifurcateBy, testCol, label1, label2):
    hr_means=data.groupby(bifurcateBy)[testCol].mean()
    hr_std=data.groupby(bifurcateBy)[testCol].std()
    hr_count=data.groupby(bifurcateBy)[testCol].count()
    print(hr_means, hr_std, hr_count)
    combinederror=(((hr_std[label1]**2)/hr_count[label1])+((hr_std[label2]**2)/hr_count[label2]))**0.5
    z=(hr_means[label1]-hr_means[label2])/combinederror
    print(z)
    p_value_from_normal_for_diff = (1-stats.norm.cdf(z))
    print ('\n\nOne Tail intense vs non intense activity difference: ', p_value_from_normal_for_diff)
    return p_value_from_normal_for_diff
```

In [224]:
```
hypTest(dataTrain, "activityIntense", "heartRate", "intense", "not_intense")
```

```
activityIntense
intense        140.928407
not_intense     99.624325
Name: heartRate, dtype: float64 activityIntense
intense        26.329880
not_intense    20.300822
Name: heartRate, dtype: float64 activityIntense
intense         247711
not_intense    1054013
Name: heartRate, dtype: int64
731.340265658893


One Tail intense vs non intense activity difference:  0.0
```

Out[224]:
```
0.0
```

As the p-value is just 0% which below 5% significance level, we can neglect the null hypothesis and accept the alternate hypothesis which states that there is a significant difference between the mean heart rates of subjects while performing more intense and less intense activities.

In [232]:
```
dataTrain = dataTrain.drop('activityIntense', axis=1)
```

5.2 Testing Hypothesis 2

We would like to know if the difference between the mean chest temperature of subjects while performing rope jumping is significantly different as compared to that other activities.

- Null Hypothesis: H0 - There is no significant difference in the mean chest temperature of subjects while performing rope jumping as compared to that of other activities.
- Alternative Hypothesis: H1 - There is a significant difference in the mean chest temperature of subjects while performing rope jumping as compared to that of other activities.

The favourable outcome would be if we can reject the null hypothesis with a 95% confidence interval. In other words, we would like to confirm that 95% of the times the means are statistically different.

In [226]:
```
dataTrain['isropejump'] = ['yes' if i == 24 else 'no' for i in  dataTrain['activityID']]
```

In [228]:
```
hypTest(dataTrain, "isropejump", "chestTemp", "yes", "no")
```

```
isropejump
no     36.246640
yes    33.602023
Name: chestTemp, dtype: float64 isropejump
no     1.435306
yes    1.474523
Name: chestTemp, dtype: float64 isropejump
no     1268653
yes      33071
Name: chestTemp, dtype: int64
-322.20823437357456


One Tail intense vs non intense activity difference:  1.0
```

Out[228]:
```
1.0
```

As the p-value is just 100% which higher than 5% significance level, we can not neglect the null hypothesis and therefore there is no statistical evidence that mean temperatures are different while performing rope skipping as compared to other activities.

In [234]:
```
dataTrain = dataTrain.drop('isropejump', axis=1)
```

6. Model Development

In this section we will try to fit a couple of model to find the bet model for the given data. We will be using the training data to fit the model and subsequently perform some tests to select the best model which shall produce desired results. In order to start the modeling process we need to remove a couple columns which can create issues during the process. These columns are timeStamp and subjectID.

In [230]:
```
# Importing modeing packages
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,cross_val_score,StratifiedShuffleSplit
from sklearn.metrics import precision_score,recall_score, f1_score, confusion_matrix,roc_auc_score,roc_curve, accuracy_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 [ ]:
```
train_df = train_df.drop(["timestamp", "subject_id"],1)
```

In [233]:
```
dataTrain
```

Out[233]:

gn_1	handMagn_2	handMagn_3	...	chestMagn_3	ankleTemp	ankleAcc16_1	ankleAcc16_2	ankleAcc16_3	ankleMagn_1	ankleMagn_2	ankleMagn_3	subjectID	isropejump
5710	-13.23800	-3.18283	...	-10.08810	35.0000	8.67393	-2.906790	-4.231220	-36.96500	19.15340	28.06230	106	no
1400	-26.03470	-69.41560	...	-35.34630	32.6250	-1.12358	0.846263	-9.989060	-11.53370	33.69820	72.68790	101	no
7400	-16.89900	-5.17380	...	-3.38165	32.5000	9.17009	0.205475	-4.476980	4.24416	-6.37674	13.48630	107	no
0500	-13.48340	-36.86990	...	-17.50200	34.8750	9.86913	-1.061070	-3.342860	-55.83180	32.10370	-3.36399	106	no
3900	-27.42880	-44.46840	...	10.31630	31.8750	19.64920	-3.690840	-4.962770	-38.39830	-18.15970	11.64800	108	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9889	-63.81860	-7.55063	...	26.88770	34.1250	13.96340	0.277704	-3.039050	-47.41050	-41.48580	21.67770	101	no
8200	-114.65900	22.94990	...	-10.75220	34.5000	9.65815	-0.569270	-2.962180	-38.38520	15.79290	43.23720	105	no
8700	-7.80815	-42.28130	...	46.71520	32.3750	2.98744	2.610350	-1.313790	-33.99440	-32.79580	12.67330	107	no
2000	-62.31370	-9.81195	...	39.23550	32.3125	9.78438	-0.014349	-1.266390	-83.14140	-39.19380	-56.97650	101	no
9000	6.48574	-11.12530	...	31.87010	32.7500	10.89940	0.388399	-0.292711	-43.66890	-23.86180	-2.83079	104	no

In [ ]:
```
dataTrain = dataTrain.drop(['timeStamp','subjectID'], axis=1)
```

In [249]:
```
dataTest = dataTest[dataTrain.columns]
dataTest.head()
```

Out[249]:

	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handMagn_1	handMagn_2	handMagn_3	chestTemp	...	chestMagn_1	chestMagn_2	chest
0	7	129.0	34.7500	-9.52337	12.01650	-2.56780	10.8357	-37.42810	-13.40760	38.5000	...	7.34547	-33.6202	-
1	16	110.0	34.6875	-10.15230	-9.22155	2.18989	53.5705	9.29874	-28.47550	38.2500	...	-30.27050	-11.6429	3
2	2	98.0	34.2500	-1.97168	-8.70099	8.16075	11.7041	18.85380	-30.54840	37.5625	...	4.86426	-25.9225	-
3	4	93.0	31.7500	-8.21659	3.66181	1.98298	34.4071	24.73230	-5.82714	36.5000	...	9.01484	-22.1131	2
4	4	123.0	31.6875	-11.16700	2.46217	1.01914	34.4899	-19.18560	-24.95060	36.4375	...	12.88610	-21.5281	3

5 rows × 23 columns

In [250]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler,RobustScaler

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

df_scaled.iloc[:,1:23] = scaler.fit_transform(df_scaled.iloc[:,1:23])
df_scaled_test.iloc[:,1:23] = scaler.fit_transform(df_scaled_test.iloc[:,1:23])

df_scaled.head()
```

Out[250]:

	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handMagn_1	handMagn_2	handMagn_3	chestTemp	...	chestMagn_1	chestMagn_2	chest
0	13	1.421053	0.052632	-0.238190	-0.231329	0.327976	-0.409592	0.093330	0.751544	0.111111	...	-0.349002	0.964657	-0
1	2	-0.500000	-0.394737	0.914629	0.872509	0.249676	-0.958908	-0.316160	-1.654626	-1.111111	...	0.555032	-2.087486	-1
2	3	-0.868421	0.052632	-0.482070	-0.200231	-0.444501	0.229011	-0.023821	0.679214	-0.222222	...	-0.287104	0.182913	-0
3	4	-0.236842	-0.657895	-0.996179	0.170460	-0.550434	0.922364	0.085477	-0.472273	-0.027778	...	0.826636	-1.003054	-0
4	6	0.315789	-1.026316	0.094034	-0.034102	0.800621	0.069995	-0.360770	-0.748319	-0.750000	...	-0.598716	-0.525199	0

5 rows × 23 columns

In [251]:
```
df_scaled_test.head()
```

Out[251]:

	activityID	heartRate	handTemp	handAcc16_1	handAcc16_2	handAcc16_3	handMagn_1	handMagn_2	handMagn_3	chestTemp	...	chestMagn_1	chestMagn_2	chest
0	7	0.642178	0.684211	-0.508943	1.575137	-1.116214	-0.350471	-0.685094	0.380955	0.916667	...	0.229381	-0.010580	-0
1	16	0.142178	0.657895	-0.587554	-2.363753	-0.230256	0.886195	0.813675	-0.166635	0.805556	...	-1.651197	1.373936	1
2	2	-0.173611	0.473684	0.434963	-2.267208	0.881614	-0.325341	1.120155	-0.241967	0.500000	...	0.105335	0.474356	-0
3	4	-0.305190	-0.578947	-0.345605	0.025644	-0.268786	0.331641	1.308709	0.656439	0.027778	...	0.312840	0.714339	0
4	4	0.484284	-0.605263	-0.714384	-0.196846	-0.448268	0.334037	-0.099964	-0.038535	0.000000	...	0.506380	0.751193	0

5 rows × 23 columns

In [252]:
```
X_train = df_scaled.drop('activityID', axis=1).values
y_train = df_scaled['activityID'].values

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

6.1 Pricipal component Analysis

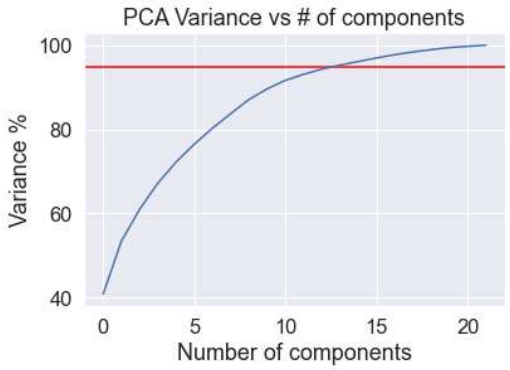
```
Principla component analysis is a dimentionality reduction tool which is used when there are high number of dimensions/features per
observation. It help in increasing the interpretability of data while preserving the maximum amount of information, and enabling the
visualization of multidimensional data.
Usually 90-98% of the variance will explain our data really well. So by plotting the variance ratio aginst the number of components we
can find how many of those we could use. As we see from the graph below 13 components fall around to 94% of the variance.
```

In [259]:
```
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)*100)

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

plt.plot(var1)
```

Out[259]:
```
[<matplotlib.lines.Line2D at 0x200be1a4d30>]
```



In [ ]:
```
# Taking 15 components
pca = PCA(n_components=15)
X_train=pca.fit_transform(X_train)
X_test=pca.fit_transform(X_test)
```

6.2 Model Selection

In the following section we will fit two model in our data and subsequently test their performance against the test dataset to finally recommend a suitable model for our purpose.

The two models which we are going to use are **Logistic Regression and Random forest**:

1. Logistic Regression algorithm is a simple algorithm that can be used for binary/multivariate classification tasks. The result of it is a probability that a data point is a part of a particular class.
2. Random Forest algorithm can be used for both classification and regression which makes it a very versatile modeling algorithm. As the name suggests, Random Forest is a forest of decision trees, which are randomly populating the forest. The algorithms creates and combines decision trees together, the more trees in the forest, the better the accuracy of its predictions. Random Forest algorithms are good because of the high accuracy they provide as well as beacuse of their flexibility

i.e. it can work for both classification and regression modelling. In addition, Random Forest performs well with high dimensionality datasets which is the case with our dataset.

In [264]:
```
def get_metrics (y_true,y_pred):
    acc = accuracy_score(y_true, y_pred)
    err = 1-acc
    p = precision_score(y_true, y_pred,average=None).mean()
    r = recall_score(y_true, y_pred, average=None).mean()
    f1 = f1_score(y_true, y_pred, average=None).mean()

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

In [265]:
```
# Running Logistic regression model and printing out test results
log_reg = LogisticRegression()
log_reg.fit(X=X_train, y=y_train )
y_pred_lr = log_reg.predict(X_test)
get_metrics(y_test, y_pred_lr)
```

```
Accuracy:  0.8114485098223586
Error:  0.1885514901776414
Precision 0.7840655661660696
Recall 0.7713822228520518
F1 0.7759365358747266
```

It can be seen that the accuracy and precision of the logistic model is quite high therefore it reduces both type 1 and type 2 errors, which is a good sign for a model.

In [266]:
```
# Running Random forest model and printing out test results
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_jobs =4)
rfc.fit(X_train,y_train)
y_pred_rf = rfc.predict(X_test)
get_metrics(y_test,y_pred_rf)
```

```
Accuracy:  1.0
Error:  0.0
Precision 1.0
Recall 1.0
F1 1.0
```

6.3 Cross validation

Even though the above models seem to perform good, the metrics used for analysis does not provide a full picture as the model were built on a specific part of the parent dataset. By using cross validation, we can create k number of different samples from the dataset and subsequently testing the model on each of the sample. By performing this exercise we will get k different sets of metrics. The mean value of these metrics can show a better picture of the model's performance.

In [*]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict

classifiers = [LogisticRegression(),
               RandomForestClassifier(n_jobs = 4)]

score_lst = []
for cls in classifiers:
    # Cross validating the accurancy
    accs  = accuracy_score(y_train, cross_val_predict(cls,X_train,y_train,cv = 10))
    # calcuating the error
    scores = cross_val_score(cls,X_train,y_train,scoring = "neg_mean_squared_error",cv= 10)
    score = np.sqrt(-scores)
    f1 = cross_val_score(cls,X_test, y_test,scoring = "f1_macro", cv =10)
    score_lst.append([cls.__class__.__name__,accs,score.mean(), f1.mean()])

df_scores=pd.DataFrame(columns = ["Classifier","Accuracy","MSE","F1"],data =  score_lst)
display(df_scores)
```

7. Conclusion

We performed the necessary steps for model development beginning from data preparation to data cleaning and subsequently exploring hidden trends within the data. Hypothesis tests conducted gave more clarity about the data and helped us to form a platform for final model development.

It can be concluded from the above results that randomforest classifier is an appropriate model for predicting the activity type given the required sensory data. High accuracy and precision ensure that the model minimises the type 1 and type 2 errors. Therefore, it is recommended to use the model developed by using random forest modeling algorithm.

8. References

Archive.ics.uci.edu. (2012) PAMAP2 Physical Activity Monitoring Data Set http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring (http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring) [Accessed 15 Dec. 2018].

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) [Accessed 3 Jan. 2019].

https://en.wikipedia.org/wiki/Principal_component_analysis (https://en.wikipedia.org/wiki/Principal_component_analysis)

https://www.statology.org/z-test-python/ (https://www.statology.org/z-test-python/)

PAMAP2_Dataset: Physical Activity Monitoring. (n.d.). [ebook] Available at: http://archive.ics.uci.edu/ml/machine-learning-databases/00231/readme.pdf (http://archive.ics.uci.edu/ml/machine-learning-databases/00231/readme.pdf) [Accessed 15 Dec. 2018].

Scikit-learn.org. (n.d.). Robust Scaling on Toy Data — scikit-learn 0.18.2 documentation. https://scikit-learn.org/0.18/auto_examples/preprocessing/plot_robust_scaling.html (https://scikit-learn.org/0.18/auto_examples/preprocessing/plot_robust_scaling.html)

In [ ]: